RuStore

RuStore开发者技术文档



目录

面向开发者 3

SDK 187

RuStore API 606

面向开发者

为了管理应用程序，提供了一种内容管理系统 - RuStore 控制台。

通过统一的 VK ID 账户进入系统。

开发者访问 RuStore 控制台系统是通过 web-接口进行的。

开发者账户

开发者注册

您可以在 RuStore 控制台系统中创建法人或自然人账户。

注册法人/个体工商户适用于：

- 商业和工业企业；
- 国有公司；
- 专业开发者。

选择注册法人/个体工商户时，需要用合格的电子签名签署注册表格。

自然人注册旨在用于个人目的。账户适用于：

- 学生；
- 业余开发者；
- 准专业开发者。

选择注册自然人时，需要进行远程身份验证。

如果您的账户通过 Google 授权停止工作，请尝试其他方式重新登录：通过电话号码、VK、亚德克斯（Yandex）或 国家服务（Goluslugi）。请使用之前授权时的同一电话号码。

如何了解您的 VK ID

VK ID 是 VK 服务的统一账户。如果您有 VKontakte 社交网络的用户档案，它将自动成为您的 VK ID。

如果您不希望将 RuStore 控制台系统中的账户与 VKontakte 用户档案关联，或者您没有 VKontakte 账户，您可以创建一个新的用户档案并将其用作企业账户。

如何找到 VK ID:

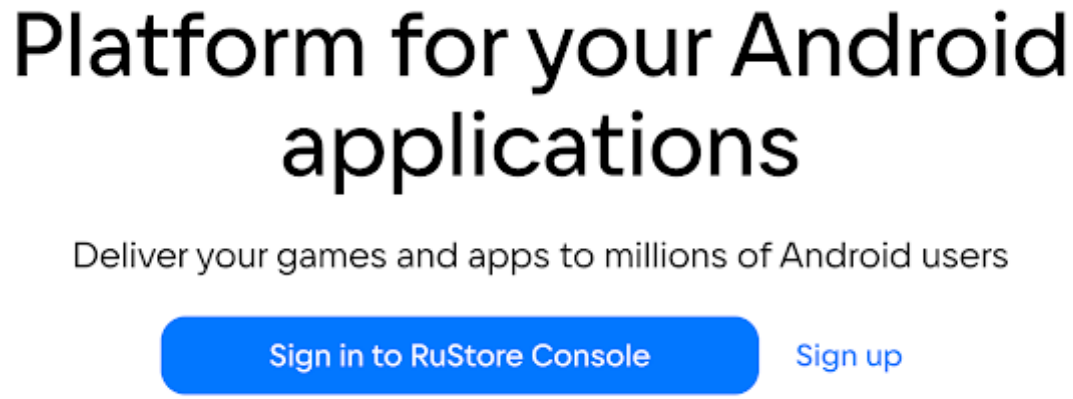
1. 登录您的 VKontakte 用户档案。
2. 转到“设置”部分。
3. 找到“页面地址”区块。
4. 如果您之前分配了字母数字地址，请点击“修改”查看页面号码。

这就是您的 VK ID。

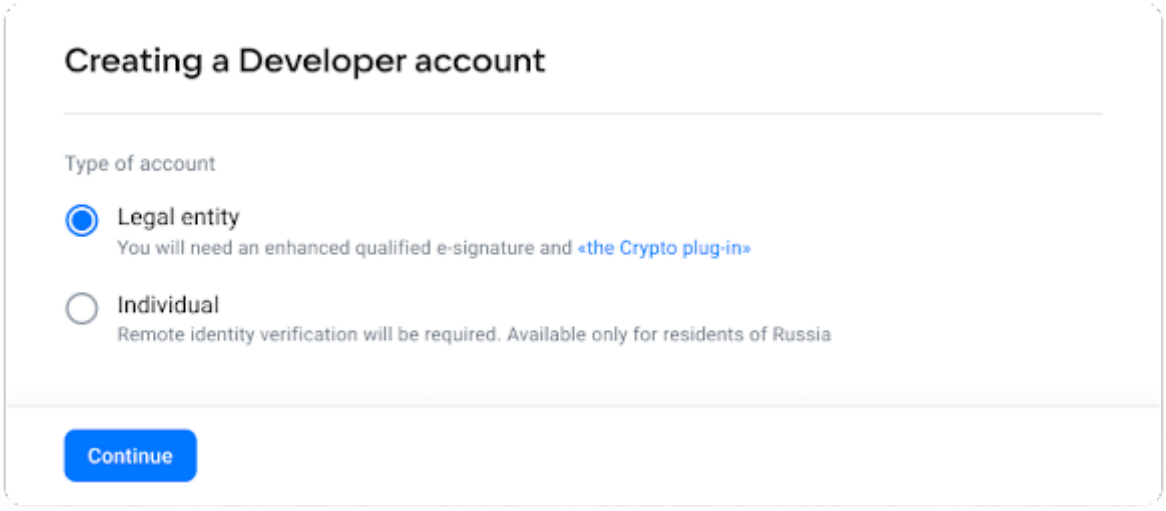
如果您的账户通过 Google 授权停止工作，请尝试其他方式重新登录：通过电话号码、VK、亚德克斯（Yandex）或 国家服务(Gosuslugi)。请使用之前授权时的同一电话号码。

注册外国公司

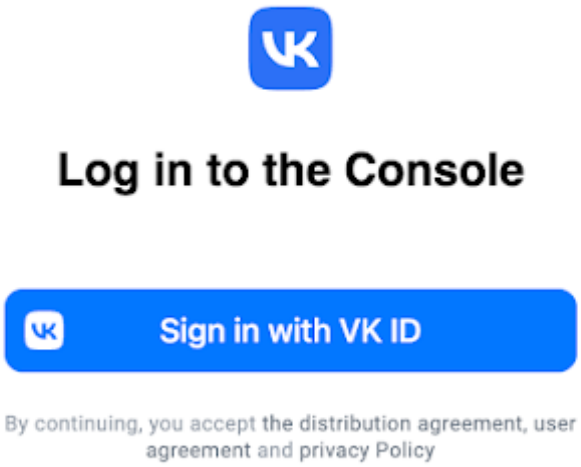
被视为非居民的公司是所有在俄罗斯联邦以外注册的公司。在浏览器中打开 RuStore 控制台系统。点击“创建账户”。



1. 选择“法人或个体工商户”。

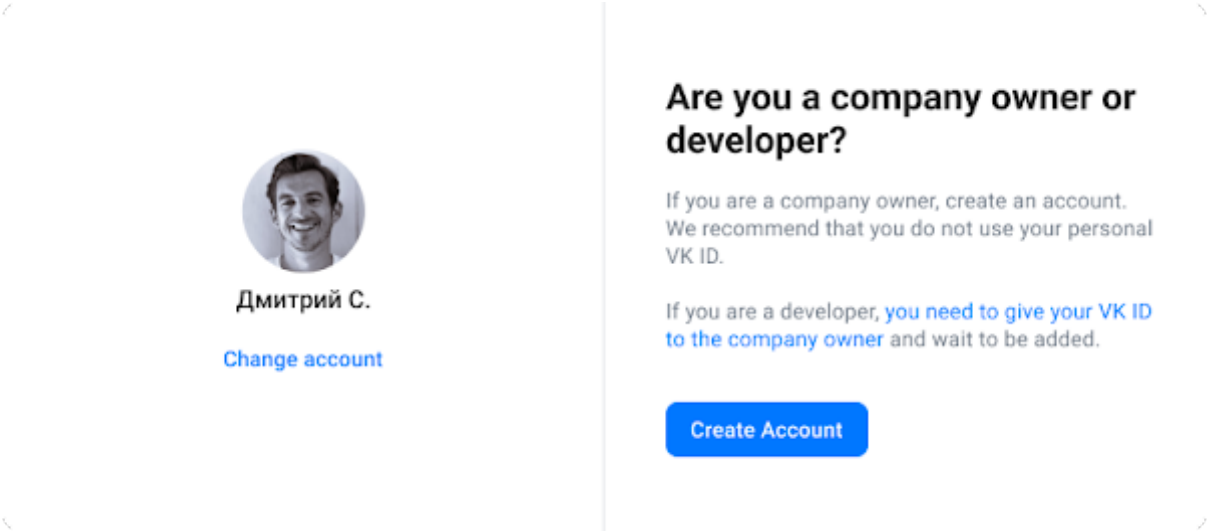


1. 点击“通过 VK ID 登录”。
2. 号码，或创建新账户。可以使用国际电话号码创建账户。
3. 点击“继续”。



1. 输入绑定到您 VK ID 账户的电话
1. 点击“创建账户”。

一个账户最多只能绑定一家公司。如果公司账户已由其他员工创建，请联系他将您的账户添加到公司账户。



1. 点击“继续”。

Legal entity registration

Company information

Country of residence
China

Trade name (visible to users)
Tencent

Legal name
Tencent LLC

Tax or registration number
657800008021001 23A

Company phone number
+1 40 0061 7810

Company website
www.tencent.com

Contact Person

Name
Guangming

Last name
Taibai

Middle name (if included)

The representative's VK ID
785222546

The representative's phone number
+1 40 0061 7810

The representative's e-mail
taibai.corp@gmail.com

Documents

Please attach documents in English or Russian

Link to your app in other stores (Google Play, GetApps or AppGallery)
https://play.google.com/store/apps/details?id=com.plarium.mechlegion&hl=ru&gl=US

Documents of the company incorporation

.doc Establishment 8 MB ✓ Uploaded

.pdf Document 8 MB ✓ Uploaded

Upload 1-3 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG
Choose file

Partner representative's authority documents for RuStore registration

.docx Registration 8 MB ✓ Uploaded

Authority 8 MB Uploaded

Upload 1-3 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG

Partner's domain name/trademark/logo registration certificate

.jpg Certificate 8 MB ✓ Uploaded

Upload 1-5 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG
Choose file

Submit a request for moderation

1. 填写所有字段。

- 商业名称 – 这是 RuStore 应用和 www.rustore.ru 网站的用户会看到的名称。请指定用户熟悉的名称，或您用于推广的名称。这将帮助用户在 RuStore 目录中找到您。

我们可能会要求您确认对所指定商业名称的权利。不要求注册权利，但在这种情况下，我们会要求您保证使用该商业名称不侵犯第三方的权利。

- 法律名称 – 这是您公司在其创立文件中指定的名称。请在填写此字段时保持注意，因为我们将会根据您提供的文件检查此处指定的名称是否相符。
- 在“联系人”区块中，请提供我们在需要时可以联系到的用户信息，以便我们可以就您的账户相关事宜进行询问。
- 在“电话号码”和 Email 的字段中填写的信息仅用于与 RuStore 内部沟通，不会对用户可见。在“代表号码”字段中，应填写国际电话号码。
- 确保“税号或注册号”字段中的信息与您在创立文件中的数据相符。

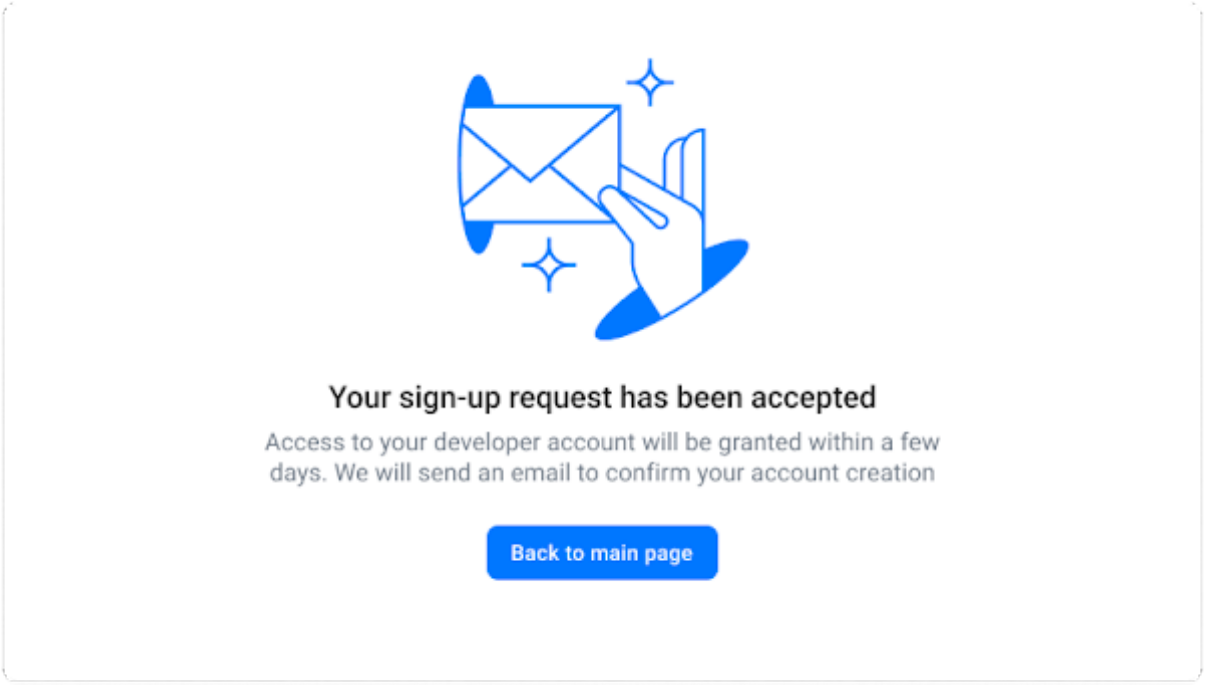
10. 在“文件”区块中附上所有要求的文件，文件需为英文或俄文：

“您的应用在备选商店的链接”字段是必填的。如果您的应用未在备选商店上架，您可以附上应用发布的官方网站链接。

- 提交用以证明公司存在且在其司法管辖区内注册的创立文件，例如商业执照、由政府机关认证的公司章程、注册证书等。
- 提交证明代表签名权的文件，例如：授权书、任命命令、公司章程等。这份文件中的代表信息应与您在“联系人”区块中填写的信息相符。
- 提交证明对域名、标志或商标使用权的文件，例如：注册证书、许可协议、权利所有者的同意书等。若无注册权利，则需提供一封确认没有注册标志、域名、商标权利的信件，以及一封保证拥有未注册权利的担保信。

请附上经过公证的英文或俄文翻译件。

11. 点击“提交审核”。注册申请将被发送进行审核。



如果不符合 RuStore 的规定，注册申请可能会被拒绝。在这种情况下，我们将向您在注册时提供的电子邮件地址发送一封邮件。

添加额外的开发者

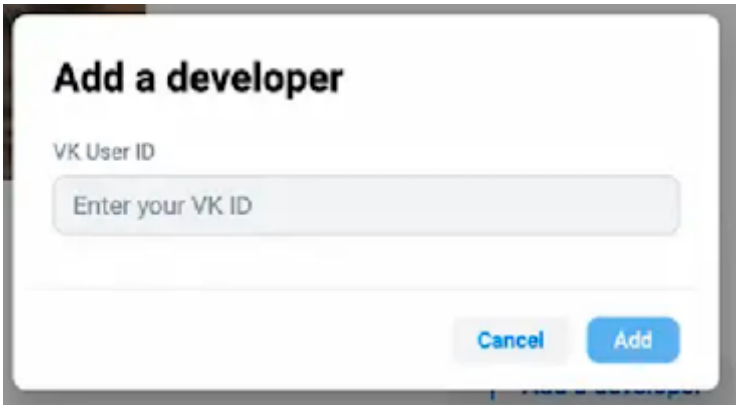
在注册时指定 VK ID 的公司员工是公司的所有者。每家公司只能有一个所有者。

公司的所有者只能注册一个公司，因此建议创建一个统一的企业 VK ID（个人）账户来管理应用程序。

公司的所有者可以创建应用程序，并根据他们的 VK ID 为每个应用程序指定/添加开发者。

如何添加额外的开发者

1. 需要添加至应用程序管理的开发者，应在 RuStore 控制台系统中使用其个人 VK ID 进行登录（在 RuStore 控制台系统登录后此阶段无需进行任何操作）。
2. 开发者将其 VK ID 以“12345678”格式传递给公司所有者。如何了解 VK ID?
3. 公司所有者可以将此开发者的 VK ID 添加到相应的应用程序中。



在为应用程序添加开发者后，该开发者将获得对相应应用程序的访问权限。他们可以更改自己应用程序的描述和数据，并且能够创建、准备和发布应用程序的新版本。

开发者无权注册公司、创建新的应用程序或向应用程序中添加其他开发者。

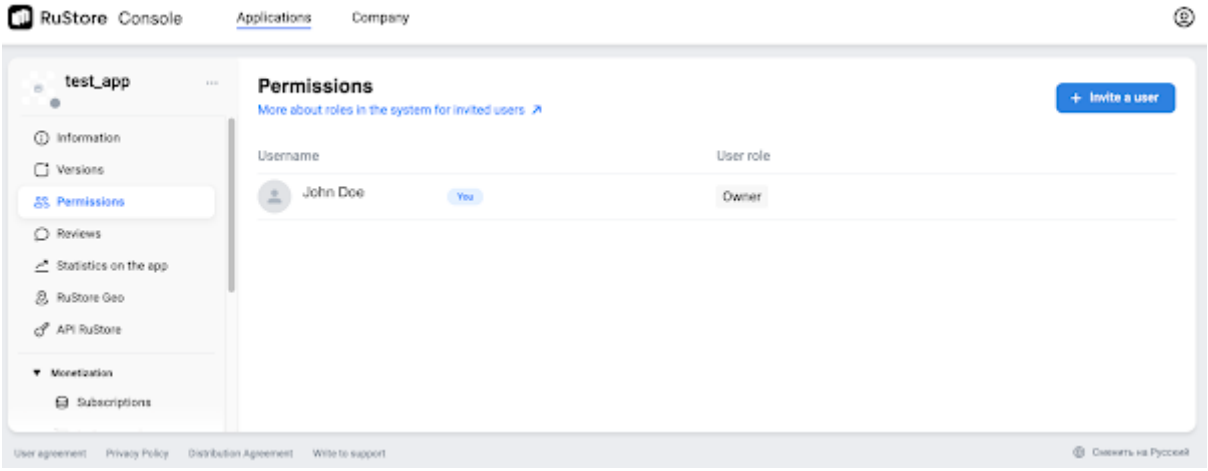
添加用户和选择角色

您可以为员工设置操作应用程序的可能限制，并为他们指定特定角色。这将允许您明确设置与应用程序的交互选项。例如，允许员工仅发布应用程序或处理退款。

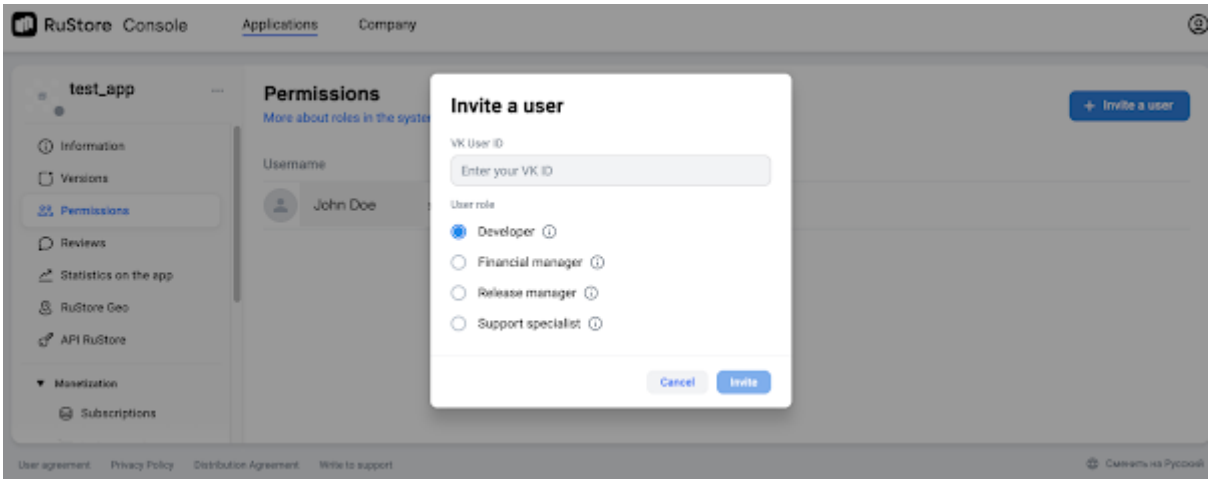
只有至少登录过一次 RuStore 控制台的用户才能添加和指派角色。

如何添加用户

1. 选择应用程序。
2. 转到左侧菜单的“访问权限”选项卡。
3. 点击屏幕顶部的“邀请用户”按钮。



4. 输入用户的 VK ID。如何了解 VK ID?
5. 选择用户角色。列表中只能选择一个选项。



6. 点击“邀请”按钮。

如何更改应用程序的所有者

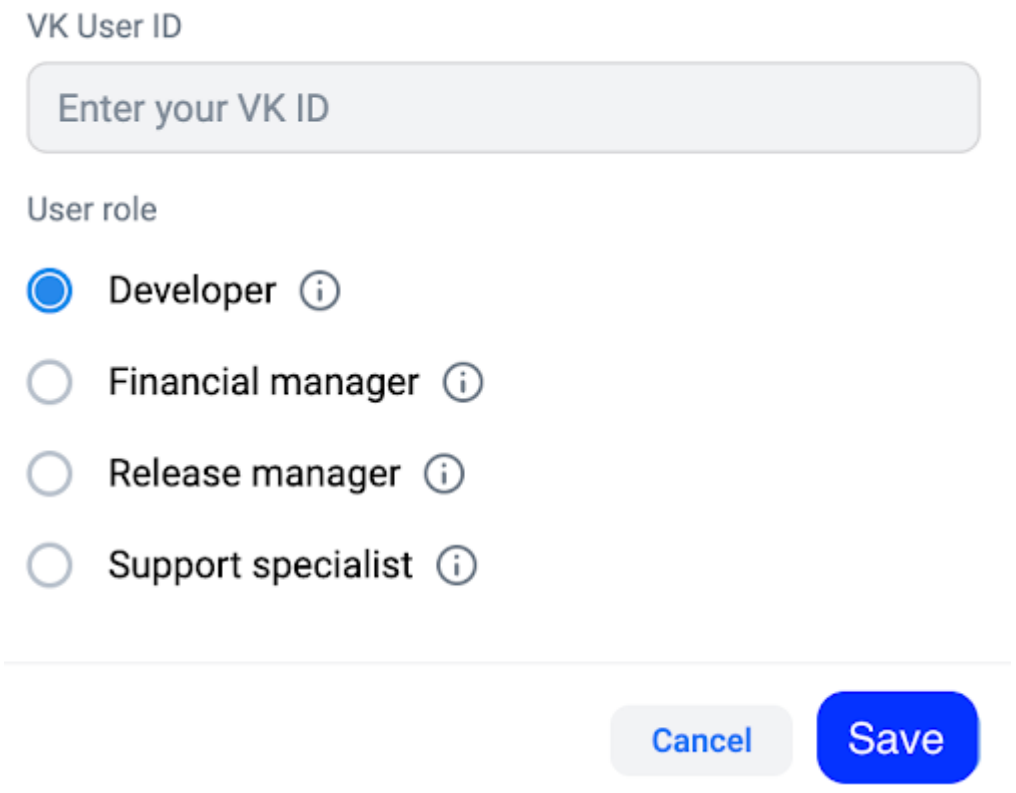
为了将应用程序转移给另一个发布者，应用程序的所有者需要向 support@rustore.ru 发送支持请求。在邮件中，需要指明当前和新应用程序所有者的 VK ID，并附上从 RuStore 控制台的截图以证明对该账户的所有权。

我们建议至少在月底前3个工作日向支持团队发送请求。这样，应用程序就可以在下个月的第一天转移到新所有者那里。

带有货币化功能的应用程序的转移以及向新所有者提供报告，将从提出支持请求的下一个月的第一天开始进行。

如何编辑权限

1. 选择用户。
2. 点击  他旁边的按钮。
3. 选择新角色。



4. 点击“保存”按钮。

如何删除用户

1. 选择用户。
2. 点击  他旁边的按钮。
3. 点击“删除”按钮。

请注意，删除操作无法撤销，如果错误删除了用户，则需要重新添加。

如何选择用户角色

在 RuStore 中，用户可以担任五种角色：

- 公司所有者；
- 开发者；
- 财务经理；
- 版本发布经理；
- 支持专员。

	所有者	管理员	版本发布经理	开发者	财务经理	支持专员
应用程序分发						
查看应用程序下载统计数据	✔		✔	✔	✔	✔
上传、编辑应用程序版本并提交审核	✔		✔	✔		
邀请用户并管理权限	✔		✔			
创建和删除应用程序	✔					
服务						
集成 push-通知，查看统计数据	✔		✔	✔		
集成 RuStore Geo	✔		✔	✔		
集成 RuStore API	✔					
货币化						
创建订阅，inn-app 产品	✔		✔		✔	
处理支付，进行退款	✔				✔	
查看财务统计	✔				✔	
填写并编辑公司银行详细信息	✔					
集成货币化	✔					
支持						
查看并回复评论	✔		✔			✔

发布和审核应用程序

在RuStore控制台系统注册后，您可以发布您的应用程序。

在RuStore发布应用程序的费用

在RuStore，没有应用程序发布费用。您可以免费发布您的应用程序，无论是免费的还是付费的。

权限声明

对于RuStore来说，用户安全是关键标准。为了确保用户数据安全，RuStore对每个应用程序所请求的所有权限（permissions）进行审核。此外，系统中实现了一个功能，用于获取有关应用程序收集和传输的用户数据类型的信息。

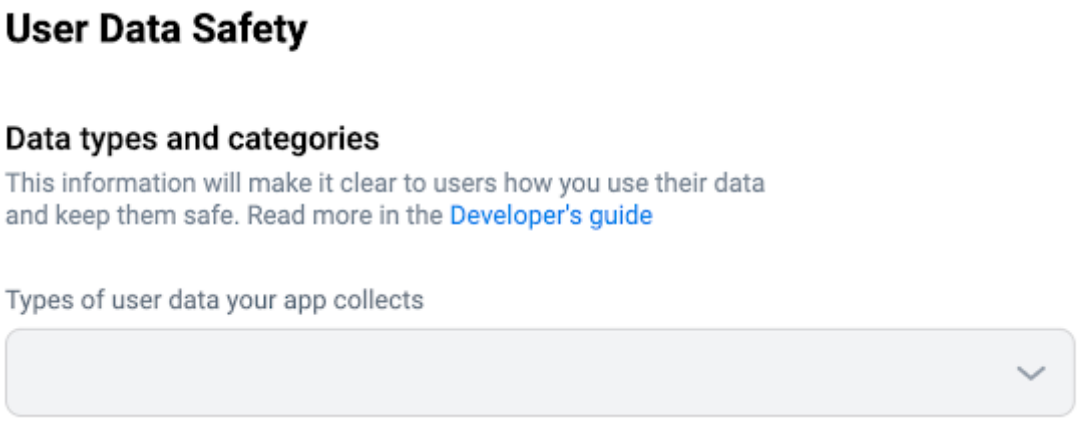
权限声明以及提供所请求数据的信息增加了透明度，并帮助商店和用户了解用户数据是如何处理的以及使用目的。

如何声明权限

为了确保用户安全，在RuStore控制台中实现了一个新的“用户数据安全”模块，在这个模块中，开发者需要填写关于数据类别和类型的信息，以及提供关于敏感权限的信息。该模块由两部分组成：“敏感权限”和“数据类别和类型”。

只有在应用中发现敏感权限时，“敏感权限”部分才会显示，并包含这些权限的列表。

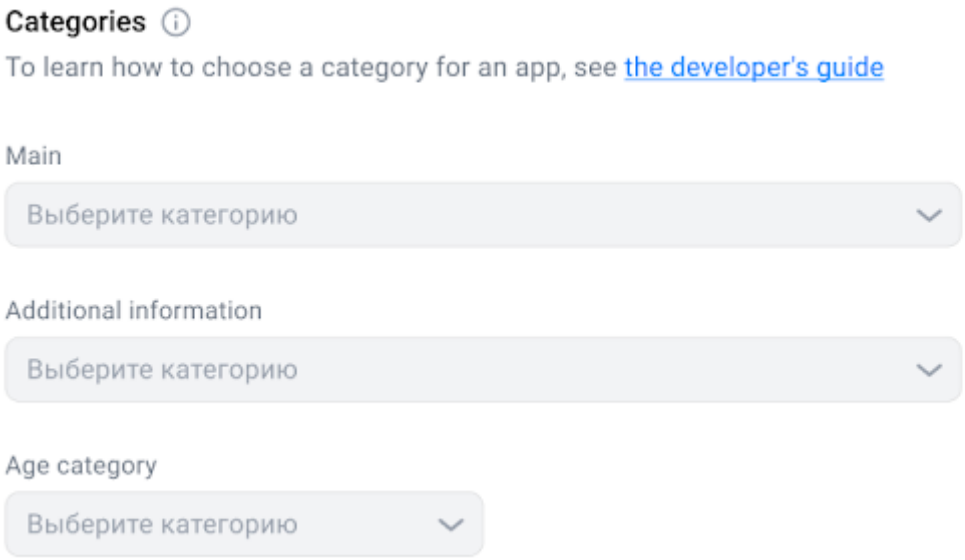
如果应用程序中发现敏感权限，在“理由”字段中需要说明每个权限的用途。



在“数据类别和类型”部分，开发者需要标记应用程序请求访问的所有数据类型。

有关可用数据类型的详细信息，请参见数据类别和类型部分。

“数据类别和类型”部分对所有应用程序版本均显示，无论使用哪种类型的权限。



在填写完应用程序版本卡中的所有信息并提交审核后，RuStore团队将对应用程序进行审核。

在将应用程序提交审核前，请务必检查应用程序是否符合应用程序要求。

如果应用程序使用敏感权限，将对应用程序进行扩展审核。

如果应用程序不符合权限使用规则，RuStore团队有权向开发者要求解释，并向账户所有者和开发者账户中的电子邮箱地址发送审核结果的信函。

如果应用程序符合开发者程序规则，申请将被批准，应用程序将可在RuStore上发布。

数据类别和类型

数据类别和类型是帮助用户理解应用程序如何使用他们的数据以及如何确保安全的信息。

关于应用程序可以收集和传递给第三方的数据类型的信息，将在RuStore移动应用程序的页面上提供给用户。

在上传或更新应用程序时，需要指定应用程序收集和传输的所有用户数据类型，具体包括：

- 应用程序收集和/或传输的所有用户数据类型；
- 应用程序中有的库或SDK通过设备发送的所有用户数据，无论信息是传给开发者还是第三方；
- 所有从开发者服务器传输给第三方或同一设备上的第三方应用程序的用户数据。

数据类别和类型

类别	类型	描述

位置	大致位置	用户或设备的位置在至少三平方公里的区域内，例如用户所在的城市，或通过Android ACCESS_COARSE_LOCATION权限获得的位置
精确位置	用户或设备的位置在不超过三平方公里的区域内，例如通过Android ACCESS_FINE_LOCATION权限获得的位置	
个人信息	姓名	用户首选的称呼：名字、姓或昵称
电子邮件地址	用户的电子邮件地址	
用户ID	用户标识符。例如，编号、标识符或账户名称	
地址	用户地址。例如，邮寄或家庭	
电话号码	用户电话号码	
种族和民族	用户的种族或民族信息	
政治或宗教信仰	用户的政治或宗教信仰信息	
性取向	用户的性取向信息	
其他数据	任何其他个人信息。例如，出生日期、性别认同、参战状态等。	
财务数据	支付数据	用户的金融账户信息。例如，信用卡号
购买历史	用户的购买和交易信息	
信用评级	用户的信用能力信息	
其他财务数据	其他财务信息。例如，用户的工资或债务	
健康和健身	医疗数据	用户健康信息。例如，医疗卡数据或症状描述
关于身体活动的数据	用户的身体活动信息。例如，关于训练	
消息	电子邮件	用户电子邮件的内容、主题、发件人和收件人信息
SMS或MMS	用户消息的内容以及发件人和收件人信息	

应用程序中的其他消息	其他类型的消息。例如，即时消息或聊天内容	
照片和视频	照片	用户的照片
视频	用户的视频	
音频文件	音频录音和语音消息	用户的语音记录。例如，语音消息或声音录制
音乐文件	用户的音乐文件	
其他音频文件	用户创建或提供的其他音频文件	
文件和文档	文件和文档	用户的文件和其他文件或关于它们的任何信息。例如，文件名
日历	日历事件	用户日历中的信息。例如，关于活动、笔记及参与者
联系人	联系人	用户联系人的信息。例如，他们的名字、通信记录，以及用户姓名、通话记录和通信的时长、频率和历史
应用程序中的行为	与应用程序的互动	关于用户如何与应用程序互动的信息。例如，他浏览了页面多少次或访问了哪些部分
应用内搜索历史	用户在应用程序中搜索的信息	
已安装的应用程序	用户设备上已安装应用程序的信息	
其他用户内容	用户创建的未在任何部分提及的其他内容。例如，传记、笔记或详细回答	
其他活动	应用程序内未在任何部分提及的其他用户活动。例如，游戏信息、点赞或对话框中选择的选项。	
浏览网页	网页浏览历史	用户访问过的网站信息
应用程序运行数据	错误报告	应用程序崩溃日志中的数据。例如，关于崩溃次数和堆栈跟踪，或与错误直接相关的其他信息。

诊断	关于应用程序性能的信息。例如，电池运行时间、加载和响应时间、帧率或技术诊断数据。
应用程序运行的其他数据	未在此处提及的应用程序性能的其他信息。
标识符	设备ID或其他标识符

权限类型

应用程序添加到RuStore控制台系统后，使用的权限会进行初步审核。

所有非**Normal**（“安全”）类别的权限都需声明，即Dangerous（“危险”）、Special（“特殊”）、Signature（“需要特殊签名”）。

使用**Not for use by third-party applications**（“禁止”）类别权限的应用程序在上传时将自动被拒绝。

Dangerous、Special和Signature类型的权限属于敏感权限类别。

下表列出了禁止和敏感权限。

权限级别

权限级别	权限名称	权限描述
禁止	ACCESS_CHECKIN_PROPERTIES	允许对checkin数据库中的“属性”表进行读/写访问，以更改上传的值。
ACCOUNT_MANAGER	允许应用程序调用AccountAuthenticators。	
BIND_APPWIDGET	允许应用程序通知AppWidget服务，哪个应用程序可以访问AppWidget的数据。	
BLUETOOTH_PRIVILEGED	允许应用程序创建蓝牙配对设备，无需用户互动，以及授权或禁止访问电话簿或消息。	
BROADCAST_PACKAGE_REMOVED	允许应用程序发送有关应用程序包删除的通知。	
BROADCAST_SMS	允许应用程序广播接收SMS的通知。	
CALL_PRIVILEGED	允许应用程序拨打任何电话号码，包括紧急服务号码，而无需经过Dialer用户界面	

	，以便用户可以确认正在进行的呼叫。
CAPTURE_AUDIO_OUTPUT	允许应用程序捕获音频输出。
CHANGE_COMPONENT_ENABLED_STATE	允许应用程序更改应用程序组件（除了它自己的）是否启用的状态。
CONTROL_LOCATION_UPDATES	允许开启/关闭来自无线电站的位置更新通知。
DELETE_PACKAGES	允许应用程序删除包。
DIAGNOSTIC	允许应用程序重复使用诊断资源。
FACTORY_TEST	允许实现以root用户身份运行的测试应用程序的启动。
INSTALL_LOCATION_PROVIDER	允许应用程序在位置服务管理器中设置位置服务提供者。
INSTALL_PACKAGES	允许应用程序安装包。
LOCATION_HARDWARE	允许应用程序使用地理位置硬件功能，如地理围栏API。
MANAGE_WIFI_INTERFACES	允许应用程序在无法满足Wi-Fi界面请求而不禁用一个或多个其他界面的情况下接收通知，并决定是批准请求还是拒绝它。
MANAGE_WIFI_NETWORK_SELECTION	此权限用于OEM制造商可以为他们信任的应用程序提供Wi-Fi API高级接口的子集以提高Wi-Fi性能。
MASTER_CLEAR	不适用于第三方应用程序。
MEDIA_CONTENT_CONTROL	允许应用程序了解正在播放的内容并控制其播放。
MODIFY_PHONE_STATE	允许更改电话状态 - 电源、mmi等。
MOUNT_FORMAT_FILESYSTEMS	允许格式化可移动存储的文件系统。
MOUNT_UNMOUNT_FILESYSTEMS	允许挂载和卸载可移动存储的文件系统。

OVERRIDE_WIFI_CONFIG	允许应用程序更改任何Wi-Fi配置，即使是由其他应用程序创建的。
READ_INPUT_STATE	这个权限在API 16级别已经过时。
READ_LOGS	允许应用程序读取低级系统日志文件。
REBOOT	用于重启设备。
REQUEST_COMPANION_PROFILE_APP_STREAMING	允许应用程序请求与支持Android应用程序流式传输（AssociationRequest.DEVICE_PROFILE_APP_STREAMING）的虚拟显示器连接，通过CompanionDeviceManager实现。
REQUEST_COMPANION_PROFILE_AUTOMOTIVE_PROJECTION	允许应用程序请求与支持汽车投影的车辆主设备的连接（AssociationRequest.DEVICE_PROFILE_AUTOMOTIVE_PROJECTION），通过CompanionDeviceManager实现。
REQUEST_COMPANION_PROFILE_COMPUTER	允许应用程序请求与计算机建立关联，以实现与其他设备如通知、照片和媒体文件的功能和/或数据交换（AssociationRequest.DEVICE_PROFILE_COMPUTER），通过CompanionDeviceManager实现。
SEND_RESPOND_VIA_MESSAGE	允许应用程序（电话）向其他应用发送请求，以处理来电期间的“通过消息回复”操作。
SET_ALWAYS_FINISH	允许应用程序控制是否在转入后台模式时立即结束活动。
SET_ANIMATION_SCALE	更改动画缩放的全局系数。

	SET_DEBUG_APP	允许设置应用程序进行调试。
	SET_PROCESS_LIMIT	允许应用程序设置可以运行的应用进程的最大数量（非强制性）。
	SET_TIME	允许应用程序直接设置系统时间。
	SET_TIME_ZONE	允许应用程序直接设置系统时区。
	SIGNAL_PERSISTENT_PROCESSES	允许应用程序请求向所有持久进程发送信号。
	START_FOREGROUND_SERVICES_FROM_BACKGROUND	允许应用程序随时从后台启动前台服务。
	STATUS_BAR	允许应用程序打开、关闭或禁用状态栏及其图标。
	UNINSTALL_SHORTCUT	请不要在您的应用程序中使用此权限。
	UPDATE_DEVICE_STATS	允许应用程序更新设备统计信息。
	WRITE_APN_SETTINGS	允许应用程序编写apn设置并阅读现有apn设置的敏感字段，如用户名和密码。
	WRITE_GSERVICES	允许应用程序更改Google服务地图。
	WRITE_SECURE_SETTINGS	允许应用程序读取或写入安全系统设置。
敏感的	ACCEPT_HANDOVER	允许呼叫应用程序继续在其他应用程序中启动的呼叫。
	ACCESS_BACKGROUND_LOCATION	允许应用程序在后台模式下访问位置。
	ACCESS_COARSE_LOCATION	允许应用程序访问大致位置。
	ACCESS_FINE_LOCATION	允许应用程序获取精确位置。
	ACCESS_MEDIA_LOCATION	允许应用程序访问用户共享集合中保存的任何地理位置。
	ACTIVITY_RECOGNITION	允许应用程序识别身体活动。

ADD_VOICEMAIL	允许应用程序向系统添加语音消息。
ANSWER_PHONE_CALLS	允许应用程序回答来电。
BLUETOOTH_ADVERTISE	用于向附近的蓝牙设备广告。
BLUETOOTH_CONNECT	用于连接到配对的蓝牙设备。
BLUETOOTH_SCAN	用于发现和配对附近的蓝牙设备。
BODY_SENSORS	允许应用程序获取对用于测量体内发生情况的传感器数据的访问，例如心跳频率。
BODY_SENSORS_BACKGROUND	允许应用程序访问用于测量体内发生情况的传感器数据，例如心跳频率。
CALL_PHONE	允许应用程序发起电话呼叫，不需要通过Dialer用户界面，以便用户可以确认呼叫。
CAMERA	需要用于访问相机设备。
DUMP	允许应用程序从系统服务中获取状态转储信息。
GET_ACCOUNTS	允许访问账户服务中的账户列表。
NEARBY_WIFI_DEVICES	用于通过Wi-Fi广告和连接到附近的设备。
POST_NOTIFICATIONS	允许应用程序发布通知。
PROCESS_OUTGOING_CALLS	这个权限在API 29级别已经过时。
READ_CALENDAR	允许应用程序读取用户的日历数据。
READ_CALL_LOG	允许应用程序读取用户的通话记录。
READ_CONTACTS	允许应用程序读取用户的联系人数据。
READ_EXTERNAL_STORAGE	允许应用程序读取外部存储。
READ_MEDIA_AUDIO	允许应用程序读取外部存储中的音频文件。

READ_MEDIA_IMAGES	允许应用程序读取外部存储中的图像文件。
READ_MEDIA_VIDEO	允许应用程序读取外部存储中的视频文件。
READ_PHONE_NUMBERS	允许读取设备的电话号码。
READ_PHONE_STATE	提供只读访问电话状态，包括当前蜂窝网络信息、任何正在进行的呼叫状态以及在设备上注册的任何电话账户列表。
READ_SMS	允许应用程序读取SMS消息。
RECEIVE_MMS	允许应用程序追踪进来的MMS消息。
RECEIVE_SMS	允许应用程序接收SMS消息。
RECEIVE_WAP_PUSH	允许应用程序接收WAP push-消息。
RECORD_AUDIO	允许应用程序录制音频。
SEND_SMS	允许应用程序发送SMS消息。
USE_SIP	允许应用程序使用SIP服务。
UWB_RANGING	需要连接到使用超宽带接入的设备。
WRITE_CALENDAR	允许应用程序写入用户的日历数据。
WRITE_CALL_LOG	允许应用程序写入（但不读取）用户的通话记录数据。
WRITE_CONTACTS	允许应用程序写入用户的联系人数据。
WRITE_EXTERNAL_STORAGE	允许应用程序写入外部存储。
ACCESS_BLOBS_ACROSS_USERS	允许应用程序访问不同用户的大型二进制数据对象。
BATTERY_STATS	允许应用程序收集电池充电统计信息。
BIND_ACCESSIBILITY_SERVICE	对于无障碍服务，应作为必需，以确保只有系统可以绑定到它。

BIND_AUTOFILL_SERVICE	对于自动填充服务，应作为必需，以确保只有系统可以绑定到它。
BIND_CALL_REDIRECTION_SERVICE	对于CallRedirectionService，应作为必需，以确保只有系统可以绑定到它。
BIND_CARRIER_MESSAGING_CLIENT_SERVICE	CarrierMessagingClientService的子类必须受此权限保护。
BIND_CARRIER_SERVICES	允许系统进程绑定到具有指定权限的应用程序中的服务
BIND_CHOOSER_TARGET_SERVICE	此常量在API级别30上已过时。此常量在API级别30上已过时。要发布直接分享目标指标，请遵循 https://developer.android.com/training/sharing/receive.html#providing-direct-share-targets 中的说明
BIND_COMPANION_DEVICE_SERVICE	对于CompanionDeviceServices，必须是必需的，以确保只有系统可以绑定到它。
BIND_CONDITION_PROVIDER_SERVICE	对于ConditionProviderService，必须是必需的，以确保只有系统可以绑定到它。
BIND_CONTROLS	允许SystemUI请求第三方控件。
BIND_DEVICE_ADMIN	设备管理接收器必须要求，以确保只有系统可以与之交互。
BIND_DREAM_SERVICE	对于DreamService，必须是必需的，以确保只有系统可以绑定到它。
BIND_INCALL_SERVICE	对于InCallService，必须是必需的，以确保只有系统可以绑定到它。

BIND_INPUT_METHOD	对于InputMethodService，必须是必需的，以确保只有系统可以绑定到它。
BIND_MIDI_DEVICE_SERVICE	对于MidiDeviceService，必须是必需的，以确保只有系统可以绑定到它。
BIND_NFC_SERVICE	对于HostApduService或OffHostApduService，必须是必需的，以确保只有系统可以绑定到它。
BIND_NOTIFICATION_LISTENER_SERVICE	对于NotificationListenerService，必须是必需的，以确保只有系统可以绑定到它。
BIND_PRINT_SERVICE	对于PrintService，必须是必需的，以确保只有系统可以绑定到它。
BIND_QUICK_ACCESS_WALLET_SERVICE	对于QuickAccessWalletService，必须是必需的，以确保只有系统可以绑定到它。
BIND_QUICK_SETTINGS_TILE	允许应用程序绑定到第三方快速设置瓦片。
BIND_REMOTEVIEWS	对于RemoteViewsService，必须是必需的，以确保只有系统可以绑定到它。
BIND_SCREENING_SERVICE	对于CallScreeningService，必须是必需的，以确保只有系统可以绑定到它。
BIND_TELECOM_CONNECTION_SERVICE	对于ConnectionService，必须是必需的，以确保只有系统可以绑定到它。
BIND_TEXT_SERVICE	对于TextService（例如，SpellCheckerService），必须是必需的，以确保只有系统可以绑定到它。
BIND_TV_INPUT	对于TvInputService，必须是必需的，以确保只有系统可以绑定到它。

BIND_TV_INTERACTIVE_APP	对于TvInteractiveAppService，必须是必需的，以确保只有系统可以绑定到它。
BIND_VISUAL_VOICEMAIL_SERVICE	对于VisualVoicemailService，必须是必需的，以确保只有系统可以绑定到它。
BIND_VOICE_INTERACTION	对于VoiceInteractionService，必须是必需的，以确保只有系统可以绑定到它。
BIND_VPN_SERVICE	对于VpnService，必须是必需的，以确保只有系统可以绑定到它。
BIND_VR_LISTENER_SERVICE	对于VrListenerService，必须是必需的，以确保只有系统可以绑定到它。
BIND_WALLPAPER	对于WallpaperService，必须是必需的，以确保只有系统可以绑定到它。
CHANGE_CONFIGURATION	允许应用程序更改当前配置，例如语言标准。
CLEAR_APP_CACHE	允许应用程序清除设备上所有已安装应用程序的缓存。
DELETE_CACHE_FILES	旧的应用程序缓存文件删除权限不再使用，但它提示我们应该安静地忽略调用，而不是引发异常。
GET_ACCOUNTS_PRIVILEGED	允许访问账户服务中的账户列表。
GLOBAL_SEARCH	这个权限可以供内容提供商使用，以允许全球搜索系统访问他们的数据。
INSTANT_APP_FOREGROUND_SERVICE	允许即时应用创建前台服务。
INTERACT_ACROSS_PROFILES	允许在同一用户组内的不同用户之间进行互动。
LAUNCH_MULTI_PANE_SETTINGS_DEEP_LINK	应用程序需要forSettings.ACTION_SETTIN

	GS_EMBED_DEEP_LINK_ACTIVITY权限才能显示嵌入在应用设置中的Activityembedded。
LOADER_USAGE_STATS	允许数据加载器读取包访问日志。
MANAGE_DOCUMENTS	允许应用程序管理对文档的访问。
MANAGE_EXTERNAL_STORAGE	为应用程序提供对外部存储的广泛访问，包括指定的区域。
MANAGE_MEDIA	允许应用程序更改和删除这台设备或任何连接的存储设备上的媒体文件，无需用户确认。
MANAGE_ONGOING_CALLS	允许请求有关当前通话的信息并管理当前通话
PACKAGE_USAGE_STATS	对于InCallService，必须是必需的，以确保只有系统可以绑定到它
READ_ASSISTANT_APP_SEARCH_DATA	允许应用程序在AppSearch中请求对ASSISTANT角色可见的全球数据。
READ_HOME_APP_SEARCH_DATA	允许应用程序在AppSearch中请求对HOME角色可见的全球数据。
READ_PRECISE_PHONE_STATE	提供只读访问手机精确状态。
READ_VOICEMAIL	允许应用程序读取系统中的语音消息。
REQUEST_COMPANION_SELF_MANAGED	允许应用程序创建“自管理”的关联。
REQUEST_INSTALL_PACKAGES	允许应用程序请求安装包。
SCHEDULE_EXACT_ALARM	允许应用程序使用精确的警报API。
START_VIEW_APP_FEATURES	允许所有者启动应用功能列表屏幕。
START_VIEW_PERMISSION_USAGE	允许所有者启动应用程序权限使用屏幕。

SUBSCRIBE_TO_KEYGUARD_LOCKED_STATE	允许应用程序订阅keyguard locked（即showing）状态。	
SYSTEM_ALERT_WINDOW	允许应用程序使用typeWindowManagerWindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY创建窗口，显示在所有其他应用程序之上。	
USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER	允许读取设备标识符并使用基于ICC的认证，例如EAP-AKA。	
WRITE_SETTINGS	允许应用程序读取或写入系统设置。	
WRITE_VOICEMAIL	允许应用程序更改和删除系统中现有的语音消息。	
安全的	ACCESS_LOCATION_EXTRA_COMMANDS	允许应用程序访问位置提供商的附加命令。
BLUETOOTH	允许应用程序连接到配对的蓝牙设备。	
BLUETOOTH_ADMIN	允许应用程序发现并与蓝牙设备配对。	
BROADCAST_STICKY	允许应用程序广播最后的意图对象。	
CALL_COMPANION_APP	允许实现InCallService API的应用程序有权作为通话的配套应用程序被启用。	
CHANGE_NETWORK_STATE	允许应用程序更改网络连接状态。	
CHANGE_WIFI_MULTICAST_STATE	允许应用程序进入Wi-Fi多播模式。	
CHANGE_WIFI_STATE	允许应用程序更改Wi-Fi连接状态。	
DELIVER_COMPANION_MESSAGES	允许应用程序在系统中传递附加消息。	
DISABLE_KEYGUARD	允许应用程序在不安全时禁用keyguard。	
EXPAND_STATUS_BAR	允许应用程序展开或收起状态栏。	

FOREGROUND_SERVICE	允许应用程序使用Service.startForeground。
GET_PACKAGE_SIZE	允许应用程序获取包大小。
HIDE_OVERLAY_WINDOWS	允许应用程序阻止非系统窗口在其上层绘制。
HIGH_SAMPLING_RATE_SENSORS	允许应用程序访问采样率超过200Hz的传感器数据。
INSTALL_SHORTCUT	允许应用程序在Launcher中创建快捷方式。
INTERNET	允许应用程序打开网络套接字。
KILL_BACKGROUND_PROCESSES	允许应用程序调用ActivityManager.killBackgroundProcesses(String)。
MANAGE_OWN_CALLS	允许调用通过ConnectionService自管理API管理其自己呼叫的应用程序。
MODIFY_AUDIO_SETTINGS	允许应用程序修改全局音频设置。
NFC	允许应用程序通过NFC执行输入输出操作。
NFC_PREFERRED_PAYMENT_INFO	允许应用程序获取NFC首选支付服务信息。
NFC_TRANSACTION_EVENT	允许应用程序接收NFC交易事件。
QUERY_ALL_PACKAGES	允许查询设备上的任何常规应用程序，不考虑清单声明。
READ_BASIC_PHONE_STATE	提供只读访问手机状态的非危险权限，包括蜂窝网络类型、软件版本等信息。
READ_NEARBY_STREAMING_POLICY	允许应用程序读取附近的流媒体政策。
READ_SYNC_SETTINGS	允许应用程序读取同步设置。
READ_SYNC_STATS	允许应用程序读取同步统计信息。

RECEIVE_BOOT_COMPLETED	允许应用程序接收 Intent.ACTION_BOOT_COMPLETED，系统启动完成后发送此广播。
REORDER_TASKS	允许应用程序更改任务顺序。
REQUEST_COMPANION_PROFILE_WATCH	允许配套应用程序在后台运行。
REQUEST_COMPANION_RUN_IN_BACKGROUND	允许配套应用程序在后台运行。
REQUEST_COMPANION_START_FOREGROUND_SERVICES_FROM_BACKGROUND	允许配套应用程序在后台启动前台服务。
REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	允许配套应用程序在后台使用数据。
REQUEST_DELETE_PACKAGES	允许应用程序请求删除包。
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	应用程序必须拥有的权限，才能使用 Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS。
REQUEST_OBSERVE_COMPANION_DEVICE_PRESENCE	允许应用程序订阅关联设备状态更改的通知。
REQUEST_PASSWORD_COMPLEXITY	允许应用程序请求屏幕锁定的复杂性，并建议用户将屏幕锁定更新到特定的复杂性级别。
SET_ALARM	允许应用程序广播设置用户警报信号的意图。
SET_WALLPAPER	允许应用程序设置壁纸。
SET_WALLPAPER_HINTS	允许应用程序设置壁纸提示。
TRANSMIT_IR	允许使用设备的红外发射器（如果可用）。
UPDATE_PACKAGES_WITHOUT_USER_ACTION	允许应用程序通过 PackageInstaller.SessionParams.setRequireUserAction(int)指定用户无需采取行动即可更新应用程序。

USE_BIOMETRIC	允许应用程序使用设备支持的生物识别方法。	
USE_EXACT_ALARM	允许应用程序使用精确警报信号，如SCHEDULE_EXACT_ALARM中所述，无需向用户请求此权限。	
USE_FINGERPRINT	这个权限在API 28级别已经过时。应用程序应改为请求USE_BIOMETRIC。	
USE_FULL_SCREEN_INTENT	对于面向Build.VERSION_CODES.Q 的应用程序，如果想要使用全屏通知的意图，此为必需。	
VIBRATE	提供访问振动功能的权限。	
WAKE_LOCK	允许使用PowerManager WakeLocks防止处理器进入睡眠模式或屏幕变暗。	
WRITE_SYNC_SETTINGS	允许应用程序记录同步设置。	
已过时	BIND_CARRIER_MESSAGING_SERVICE	这个权限在API 23级别已经过时。请改用BIND_CARRIER_SERVICES
GET_TASKS	这个权限在API 21级别已经过时。不再使用。	
PERSISTENT_ACTIVITY	这个权限在API 15级别已经过时。此功能将在未来被移除；请勿使用。	
RESTART_PACKAGES	这个权限在API 15级别已经过时。	
SET_PREFERRED_APPLICATIONS	这个权限在API 15级别已经过时。	
SMS_FINANCIAL_TRANSACTIONS	这个权限在API 31级别已经过时。	

RuStore对应用程序的权限要求不比Google Play的规则更严格，并保留它们的术语以方便开发人员。

关于权限类别的最新信息可通过以下链接获得：

https://developer.android.com/reference/android/Manifest.permission

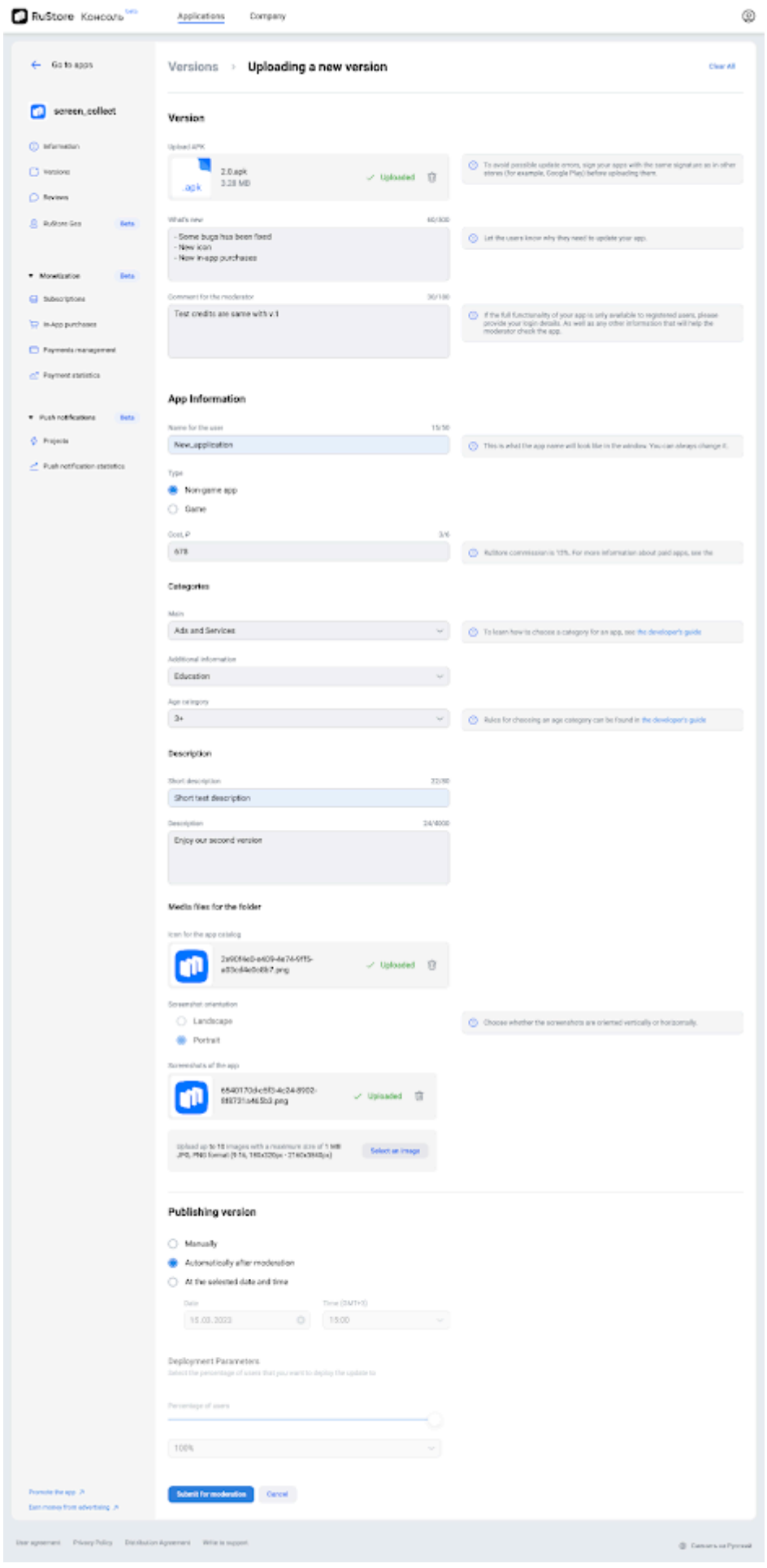
发布应用程序

发布您的应用程序，以便用户可以从 RuStore 下载它。

1. 打开 RuStore 控制台。
2. 转到“应用程序”选项卡。
3. 点击“添加应用程序”。
4. 在弹出的窗口中输入应用程序的名称。
5. 点击“添加”。
6. 选择您添加的应用程序。
7. 点击“上传版本”。

上传版本的步骤将根据应用程序的类型而有所不同：“通用”（适用于手机、平板电脑和电视）或“仅限电视”。如果您选择了“通用”应用程序类型，还将额外提供填写电视版本的信息。如果您的应用程序仅适用于电视，请按照页面上的指示进行操作，为电视设备添加应用程序。

如果创建了一个通用应用程序并上传了适用于移动设备和电视的版本，则后续版本只能是通用的。如果需要添加仅适用于电视的新版本，请创建应用程序的不同版本：通用 + 电视。


RuStore Консоль
Applications
Company
Go to apps
screen_collect
Information
Versions
Reviews
RuStore Gas
Beta
Monetization
Beta
Subscriptions
In-App purchases
Payments management
Payment statistics
Push notifications
Beta
Projects
Push notification statistics
Versions
Uploading a new version
Clear All
Version
Upload APK
2.0.apk
3.28 MB
Uploaded
To avoid possible update errors, sign your apps with the same signature as in other stores (for example, Google Play) before uploading them.
What's new
- Some bugs has been fixed
- New icon
- New in-app purchases
Let the users know why they need to update your app.
Comment for the moderator
Test credits are same with v.1
If the full functionality of your app is only available to registered users, please provide your login details. As well as any other information that will help the moderator check the app.
App Information
Name for the user
New_application
This is what the app name will look like in the window. You can always change it.
Type
Non-game app
Game
Cost, P
678
Categories
Main
Ads and Services
To learn how to choose a category for an app, see the developer's guide
Additional information
Education
Age category
3+
Rules for choosing an age category can be found in the developer's guide
Description
Short description
Short test description
Description
Enjoy our second version
Media files for the folder
Icon for the app catalog
Uploaded
Screenshot orientation
Landscape
Portrait
Choose whether the screenshots are oriented vertically or horizontally.
Screenshots of the app
Uploaded
Select an image
Publishing version
Manually
Automatically after moderation
At the selected date and time
Date
15.03.2023
Time (GMT+3)
15:00
Deployment Parameters
Percentage of users
100%
Submit for moderation
Cancel
Promote the app
Earn money from advertising
User agreement
Privacy Policy
Distribution Agreement
Write to support
Contact us on Pyrexat

应用程序版本上传

RuStore 支持以 APK 和 AAB 格式上传应用程序。

APK 文件上传要求

上传第一个版本时，请确保：

- 文件大小不超过 2.5 GB；
- 文件已用数字证书签名；
- 包名是唯一的；
- 已验证并配置好构建。

上传第二个和后续版本时，请检查：

- 文件大小不超过 2.5 GB；
- 文件已用数字证书签名；
- 签名与前一个版本相符；
- 包名与前一个版本相符；
- 已验证并配置好构建；
- 应用程序的新版本号大于前一个版本号。

上传 AAB 文件的要求

- 应用程序的签名应在上传 .aab 文件之前单独添加。
- 应用程序文件大小不应超过 500 MB；
- 包名必须唯一；
- 构建应经过验证和配置。

重要提示

如何处理 APK 文件的签名密钥？

上传 AAB 格式的应用程序

“更新内容”部分

在上传应用程序的新版本时填写“更新内容”部分。在这里描述主要更新内容，比如新增了哪些功能或修复了哪些错误。在上传通用版本的应用程序时，您也可以添加电视版本的变更历史。

用户数据安全

填写应用程序请求访问的数据类别和类型的信息，并提供有关敏感权限的信息。有关详细信息，请参阅如何声明权限部分。

应用程序信息

1. 指定应用程序名称：
    - 不超过 30 个字符；
    - 应用程序名称必须唯一。
2. 从提供的列表中选择一个类别。怎么做？
3. 从提供的列表中选择一个年龄限制。如何正确选择年龄限制？
4. 为移动设备和电视版本输入应用程序的简短描述（最多 80 个字符）。
5. 输入应用程序的详细描述（最多 4000 个字符，在编写时需要考虑用户可能将描述缩减至 2000 个字符的可能性）。

目录的媒体文件

图标

将应用程序图标从桌面或电脑上的文件夹拖放到“移动设备图标”区域。您还可以通过点击“添加电视版图标”来单独添加电视版本的图标。

重要提示

如果留空此字段，则移动设备的图标也将显示于电视版本。

图标要求

- 图像分辨率和文件大小：512x512像素，最大3MB；
- 允许的格式：.png或.jpg。

截屏

您可以仅为手机添加截屏。在这种情况下，相同的截屏将显示在所有类型的设备上：手机和平板电脑。

您还可以为平板电脑单独上传截屏。在这种情况下，手机的截屏将显示在屏幕小于7英寸的设备上，而平板电脑的截屏将出现在屏幕尺寸为7英寸及以上设备的应用程序卡片上。

重要提示

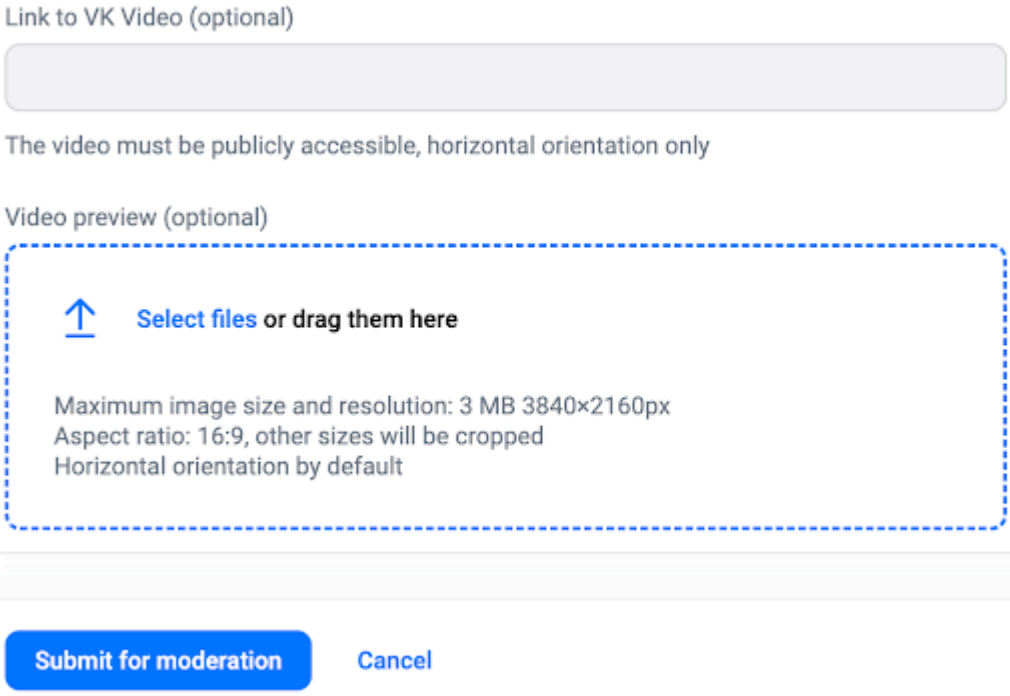
务必上传应用程序的移动版本截屏。

如果您的应用程序文件是APK或AAB格式且适用于电视，请同样上传电视的截屏和横幅。

如何添加截屏？

图像要求

- 文件大小：手机不超过**3MB**，平板电脑不超过**5MB**；
- 允许的格式：**.png**或**.jpg**；
- 最大分辨率：移动版本为**2160x3840**像素；电视版本为**1920x1080**像素；
- 推荐的纵横比 - **16:9**；
- 如果宽度或高度小于推荐值 - 截屏将被拉伸至**16:9**比例；
- 如果宽度或高度大于推荐值 - 截屏将被裁剪至**16:9**比例。



添加视频链接

您也可以添加一个公开视频链接。链接中的视频将在手机和平板电脑的应用卡片中可见。

视频要求

- 视频应直接与应用程序相关（例如，您可以上传预告片、游戏玩法录像等）；
- 视频不应包含不允许的内容（参见应用程序要求）；
- 视频必须是公开的；
- 视频的最大长度：1分钟；
- 只能添加一个视频。

创建版本草稿并提交审核

如果您已经完整填写了应用版本上传表单，请点击“提交审核”。应用程序将被提交审核。

重要提示

在提交应用程序审核前，请务必检查应用程序是否符合应用程序要求。

选择“保存为草稿”以创建草稿，并稍后返回继续编辑应用程序版本。

要使用草稿，必须遵守以下条件：

只有账户所有者、发布管理者和开发者角色可以创建、编辑和删除草稿；

您只能创建一个应用程序版本的草稿；

在RuStore控制台中，草稿只能在没有APK或AAB文件的情况下保存。要将版本保存为草稿，请从表单中删除文件，或者直接提交应用程序版本进行审核。

编辑草稿

要继续填写应用程序上传表单，请执行以下步骤：

1. 转到“应用程序”选项卡并选择“版本”部分。
2. 点击草稿旁的“编辑”以进入应用程序上传表单。

如果需要删除草稿，请进入操作菜单并选择“删除”。

添加开发者的条件

要添加将负责特定应用程序运营活动的开发者，需要满足以下条件：

- 开发者已在RuStore控制台进行授权；
- 开发者已向上传应用程序的员工传递VK ID（数字格式）。

如何上传应用程序的新版本

在发布后，您可以上传应用程序的新版本。

1。打开RuStore控制台系统。

2。转到“应用程序”选项卡。

3。选择应用程序。

4。点击“上传版本”。

5。填写应用程序参数。

6。将截屏从桌面或电脑文件夹拖到“手机应用程序截屏”或“平板电脑截屏”区域。

7。对于付费应用程序，请输入价格。

8。选择应用程序发布参数。如何设置发布？

9. 点击“提交审核”。

上传APK文件

当您在RuStore控制台上传APK文件时，指定与其他应用程序商店中相同的签名。这样，用户便可更新他们设备上已安装的应用程序到RuStore上的最新版本。

您还可以上传多个APK文件。例如，如果您的应用程序在Google Play有签名，但您为RuStore更换了签名。请上传具有不同签名的文件，以避免用户更新时出现错误。如何处理APK 文件的签名密钥？

如果您想指定一个新的签名密钥，该密钥之前未在其他商店中使用，您需要在所有商店进行更换。如何在Google Play更新签名？

要在RuStore更换签名密钥，请联系support@rustore.ru的支持服务。

如果签名密钥不匹配，用户将无法更新通过其他商店安装的应用程序。

除了APK文件，RuStore控制台现在还支持上传AAB格式的文件。详见。

提交审核

在提交应用程序审核前，请务必检查应用程序是否符合应用程序要求。

如果在发布应用程序的新版本时更改了价格，请在“描述”部分写明此版本的更改。

应用程序将被提交审核。

付费应用程序中的佣金
付费应用程序中的佣金

RuStore的佣金为开发者收入的15%。

关于佣金大小的当前条款可以在“RuStore软件产品分发协议”部分找到。

为电视设备添加应用程序

在发布应用程序时，需要指定将使用该应用程序的设备类型。

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡并点击“添加应用程序”。

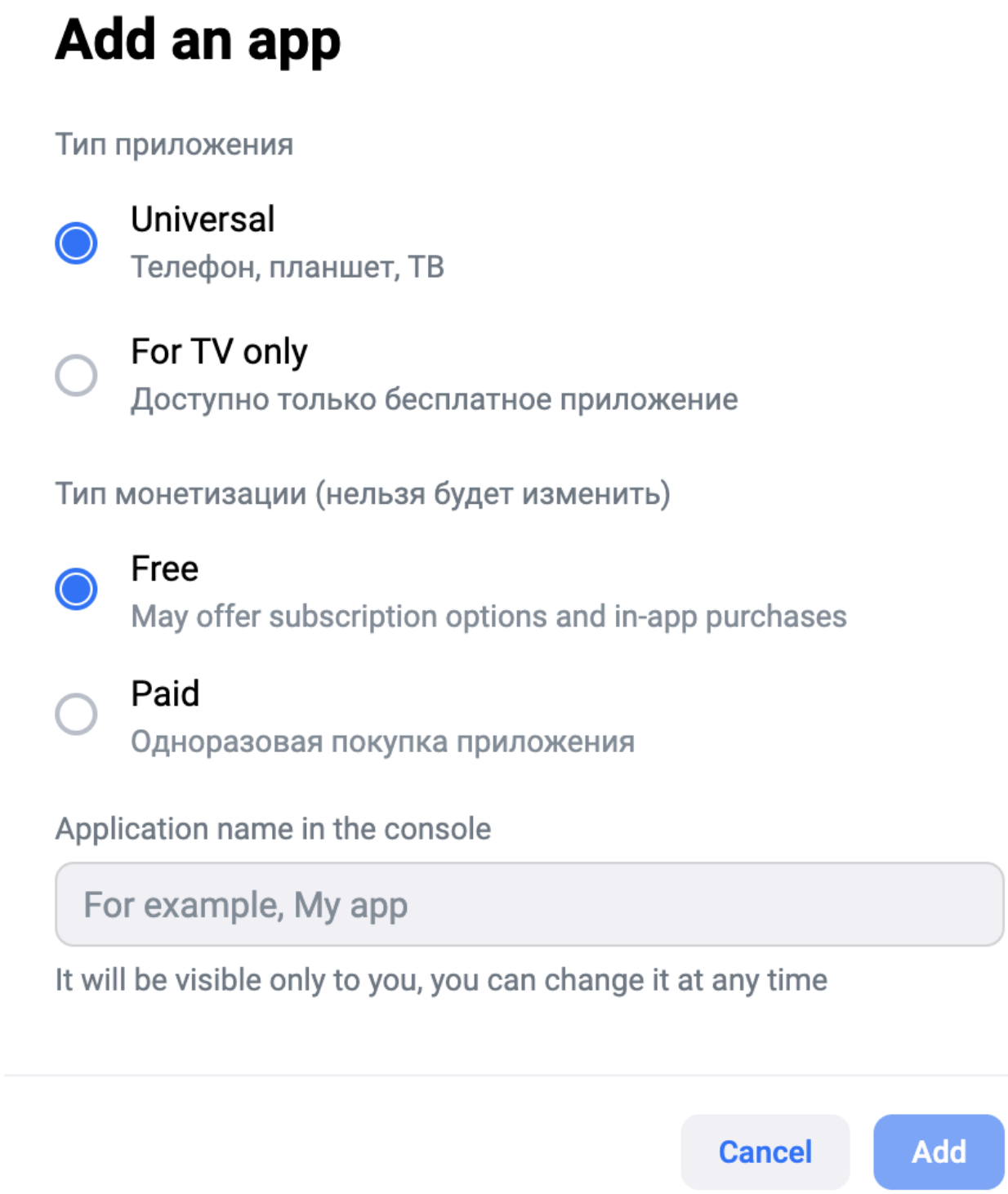
1. 在新窗口中选择设备类型“电视(Beta)”，输入应用程序名称并点击“添加”。

选择设备类型：“手机”用于在手机和平板电脑上发布，或者“电视”仅用于在电视设备上发布。

重要提示！

1。发布到电视时，package_name必须与移动设备应用程序的package_name不同。

2。电视目录独立于移动设备目录。将来，您将能够在一个目录中上传电视和移动设备的版本，请关注更新。



1. 在“版本”标签中上传新版本应用程序，请点击“上传版本”并填写以下信息：

上传应用程序的APK文件，并在必要时为审核员留下评论：

- APK文件大小：不超过2.5GB；
- 允许的格式：.apk；

- APK的签名应与其他应用程序商店使用的签名相匹配（例如，在Google Play中）。这将允许用户更新他们设备上已安装的，RuStore上有更新版本的应用程序。您可以指定一个新的签名密钥，但那么需要在所有商店中更换。如何在Google Play更新签名？
- 包名必须唯一。

填写应用程序信息：

1. 指定应用程序名称：
- 不超过 50 个字符；
- 应用程序名称必须唯一。
1. 从提供的列表中选择一个类别。怎么做？
2. 从提供的列表中选择一个年龄限制。如何正确选择年龄限制？
3. 输入应用程序的简短描述（最多80个字符）。
4. 输入应用程序的详细描述（最多4000个字符）。

将应用程序图标从桌面或电脑上的文件夹拖放到“应用程序目录图标”区域。图标要求：

- 图像大小：不超过3MB；
- 允许的格式：.png或.jpg；
- 图像分辨率：512x512。

将截屏从桌面或电脑文件夹拖到“手机应用程序截屏”窗口中。截屏要求：

- 必须选择两个或更多截屏。
- 格式：JPEG 或 PNG（24 位，无阿尔法通道）；
- 推荐尺寸：1920 x 1080像素（纵横比16:9）；
- 推荐大小：不超过3MB；
- 分辨率：从320p到4k

电视的横幅将是发布电视应用程序时的**第一个**截屏。

配置应用程序发布：“审核后自动发布”或“手动发布”。

有关配置应用程序发布的详细信息，请参见自动和手动应用程序发布部分。

1. 点击“提交审核”。

在将应用程序提交审核前，请务必检查应用程序是否符合应用程序要求。

应用程序将被提交审核。

审核通过后，应用程序的状态将改变，并出现“发布”按钮。

点击“发布”，您的应用程序将在电视上可用。

如果有任何疑问，请联系support@rustore.ru客户支持。

应用程序发布设置

自动和手动应用程序发布

您可以配置应用程序版本的自动/即时发布，而该版本仍在审核中。

即时发布设置在首次发布版本和更新先前上传的版本时可用。

自动/即时发布

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择处于“等待审核”状态的应用程序版本。
4. 在菜单中选择“自动”并点击“应用”。

审核通过后，应用程序版本将在一个小时内在RuStore发布。您的邮箱将收到关于应用程序版本状态的通知。

您可以在审核过程中将发布设置从自动更改为手动。

手动发布

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择处于“等待审核”状态的应用程序版本。
4. 在菜单中选择“手动”。

审核通过后，应用程序版本的状态将变为“准备发布”。

5。点击“发布”。

您可以取消当前版本的发布。点击菜单按钮并选择“取消发布”。已发布的版本将被停用，将启用最新的活跃版本。

如果用户在应用程序版本被停用前已经安装了新版本，他们将继续使用该版本。

应用程序的延迟发布

您可以将应用程序版本的发布延迟到指定的日期和时间。

只有在更新先前上传的版本时，才能设置延迟发布。

管理应用程序新版本的发布

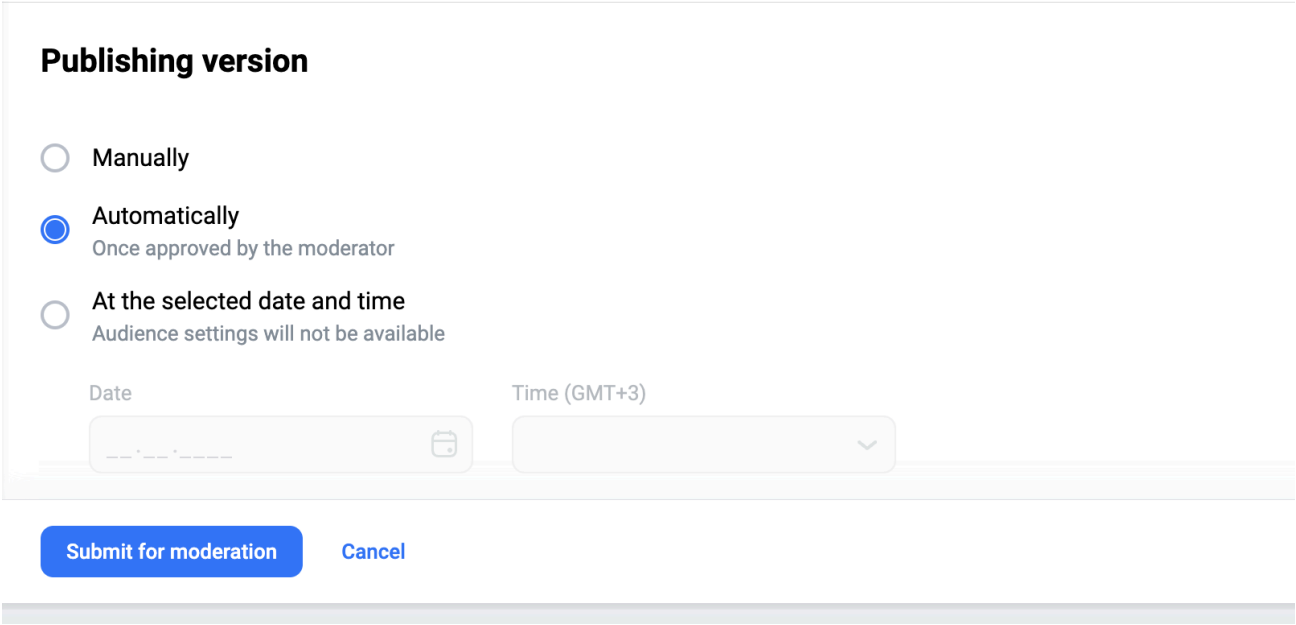
1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择应用程序。
4. 上传应用程序的新版本。怎么做？
5. 转到“版本发布”区块。
6. 选择“设定的日期和时间”。
7. 设置发布日期和时间。
8. 点击“提交审核”。

应用程序将被提交审核。

在审核过程中管理发布

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择处于“等待审核”状态的应用程序版本。
4. 点击先前选择的发布类型字段。
5. 选择“设定的日期和时间”。
6. 设置发布日期和时间。

审核通过后，应用程序版本将在指定的日期和时间自动在RuStore发布。您的邮箱将收到关于应用程序版本状态的通知。



审核后管理发布

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择处于“准备发布”状态的应用程序版本。
4. 在菜单中点击“编辑描述”。将打开应用程序版本发布页面。
5. 转到“版本发布”区块。
6. 选择“设定的日期和时间”。
7. 点击“提交审核”。

应用程序版本将在指定的日期和时间自动在RuStore发布。您的邮箱将收到关于应用程序版本状态的通知。

如果以下情况发生，您可以更改延迟发布的设置：

- 应用程序正在审核中；
- 应用程序已通过审核，但尚未到达发布日期。

分阶段应用程序发布

您可以分阶段地向一定百分比的用户发布应用程序。这有助于及时跟踪特定版本应用程序的错误、问题或负面评论。

分阶段发布仅适用于更新先前上传的版本。

可用于选择用户百分比的数值包括：

- 5%.

- 10%.
- 25%.
- 50%.
- 75%.
- 100%.

可以增加百分比值。完成分阶段发布 — 100%的用户。

管理上传新版本的发布

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择应用程序。
4. 上传应用程序的新版本。怎么做？
5. 转到“版本发布”区块。
6. 指定用户百分比。

如果选择了自动发布或手动发布，您可以选择用户百分比。

1. 点击“提交审核”。

应用程序将被提交审核。

在审核过程中管理发布

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择处于“等待审核”状态的应用程序版本。
4. 在菜单中点击“编辑描述”。将打开应用程序版本发布页面。
5. 转到“部署参数”区块并指定用户百分比。
6. 点击“提交审核”。

如果您将自动或手动发布更改为延迟发布，用户百分比将自动变为100%。

审核后管理发布

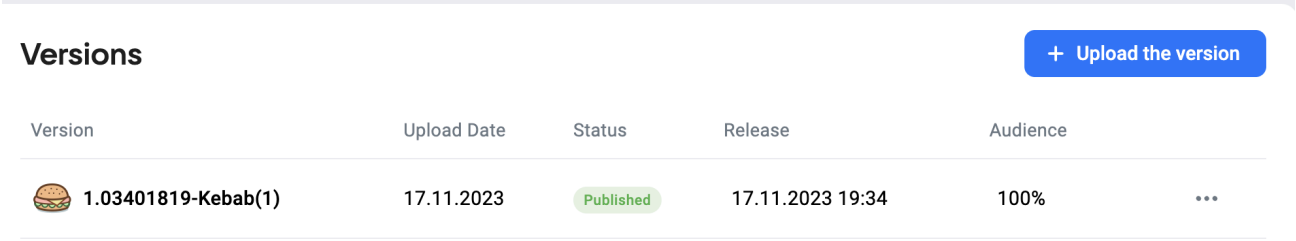
1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择处于“准备发布”状态的应用程序版本。

4. 点击之前设置的受众百分比按钮。
5. 指定用户百分比。

只能增加用户百分比。

1. 点击“应用”。

应用程序版本发布后，随机选定的用户将收到版本更新。



您可以停止分阶段发布。点击菜单按钮并选择“取消发布”。更新的版本将被停用，最后一个有效版本将被激活。

如果用户在应用程序版本被停用前已经安装了新版本，他们将继续使用该版本。

如何处理APK文件的签名密钥

什么是Android应用程序的签名？

RuStore支持一种应用程序格式 — APK文件。

每个APK文件必须用数字证书签名，Android使用它来识别应用程序的作者。签名需要像个人密码一样安全地保存。

Android如何验证版本？

Android会比较每个已签名APK文件的证书数字指纹。

数字指纹是一串字节，通过对公钥应用加密哈希函数生成。

数字指纹大致如下所示：

43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:73:a8

为什么需要进行签名验证？

如果应用程序已在Google Play上发布，这一过程就是必需的。在RuStore上传此类应用程序时，需要使用与Google Play相同的签名。这样，用户就能无误地更新他们设备上已安装的应用程序，RuStore上提供了更新版本。

是否必须使用Google Play的签名，如果有自己的签名？

不必。如果您的应用程序在Google Play上，但您有自己的签名，您不需要进行签名验证。您可以直接将带有自己签名的文件上传到RuStore。这个过程只在您使用Google Play的签名时才必要。您还可以上传带有两种签名的APK包。

为什么用户无法更新应用程序？

出版商常见的一个错误是在RuStore和其他商店发布应用程序时使用不同的签名。

例如，最初上传的应用程序版本用一个证书签名。而下一个版本则用另一个证书签名。由于这些差异，Android不允许安装此应用程序的更新。

为什么会这样发生：

1. 开发者可能丢失了证书，并为在商店发布应用程序生成了新证书。
2. 开发者可以在不同的商店发布相同的应用程序。例如，最初开发者在Google Play发布应用程序时使用了一个证书。但转到RuStore后开始使用另一个证书。

因此，用户分为两类：

- 从Google Play安装应用程序的用户；
- 从RuStore下载的用户。

如果用户安装了RuStore并想更新之前从Google Play下载的应用程序，他将无法这样做，因为证书不同。

如何解决这个问题？

1. 推荐的方式是对所有版本的应用程序使用同一个证书，以避免来自不同来源的版本更新问题。
2. 激进的方式是要求用户删除无法更新的“旧”版本应用程序，并请求下载新版本。但这种方法可能会导致部分用户流失。
3. 还可以通过RuStore的技术支持来更新应用程序的签名。详细信息在下面的部分中有描述。

RuStore建议使用本地保存的证书，以更好地控制应用程序的发布。

如果您使用Google Play App Signing，它允许Google Play独立生成和存储签名，您可能会面临无法在Google Play外使用证书的问题。

如何通过RuStore支持解决问题？

如果需要更新签名，请向support@rustore.ru的技术支持发送相应的邮件。

专家会创建一个请求并检查应用程序是否属于请求者开发者。

如果身份验证成功，专家将启动停用旧证书的过程。之后，开发者只需上传一个新证书签名的APK文件版本，该证书适用于RuStore和其他商店。

如果需要使用不同的签名怎么办？

如果您为RuStore发布了单独的签名，但应用程序已在Google Play上发布 — 上传带有两种签名的APK文件，并默认选择在Google服务之外发布的签名。这将减少用户更新时的错误数量。

如何删除应用程序卡片

您可以从RuStore控制台系统中删除应用程序卡片。只有那些没有任何版本被提交审核的应用程序才能被删除。

只有账户所有者才能删除应用程序卡片。RuStore中的用户角色

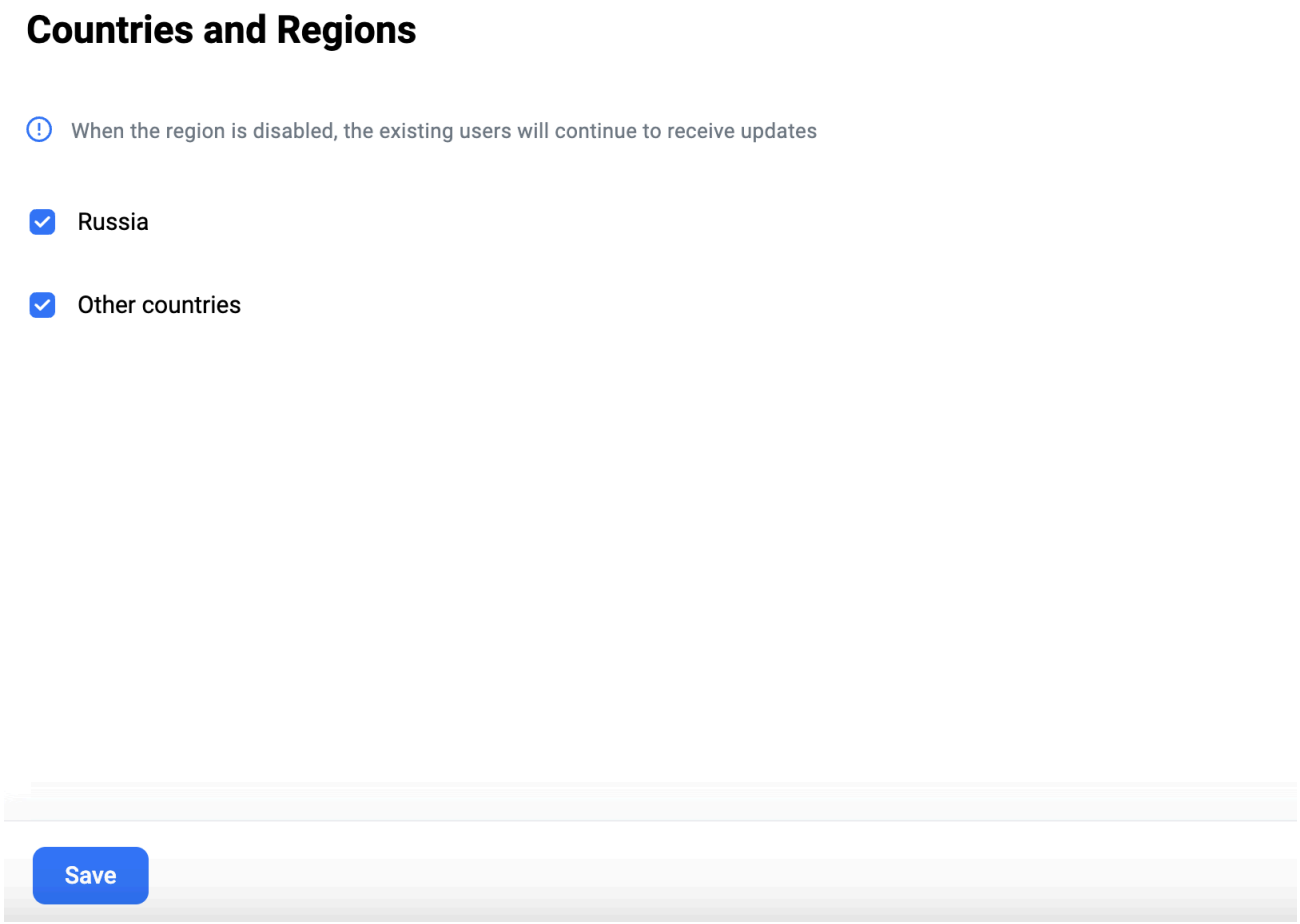
1. 转到“应用程序”部分。
2. 选择所需应用程序并点击 ，以进入上下文菜单。
3. 选择“删除应用程序”选项。
4. 点击“删除”按钮。

如何选择应用程序发布的地区

选择用户可以在其设备上安装您的应用程序的地区。目前在RuStore，应用程序可以在俄罗斯和其他国家显示。

用户可以独立于可用性设置更新已下载的应用程序。

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择所需应用程序。
4. 在左侧菜单中选择“国家和地区”。



5。检查您的应用程序将可用的地区名称旁边是否有勾选。

取消选中地区名称旁的勾选框，以防止应用程序在所选地区显示。

付费应用程序

选择应用程序类型以添加到RuStore控制台：免费或付费。

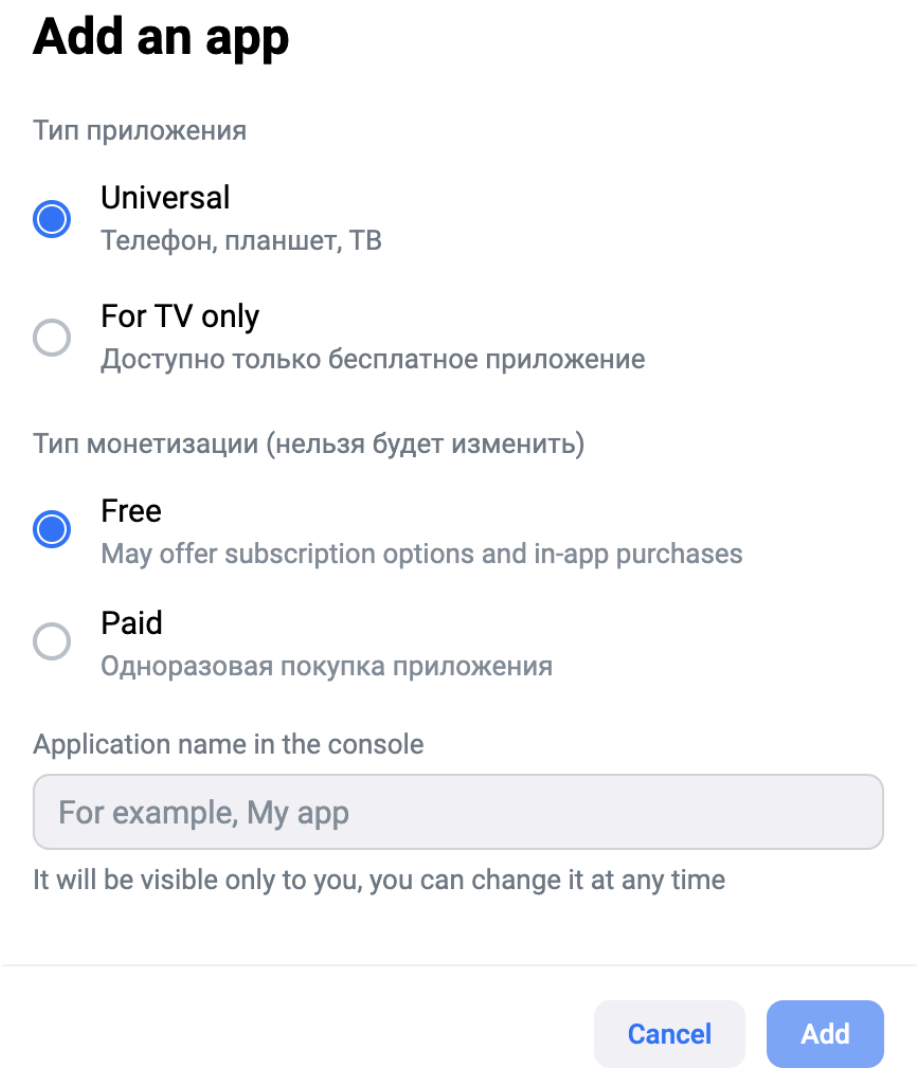
启用货币化以上传付费应用程序。怎么做？

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 点击“添加应用程序”。
4. 在弹出的窗口中输入应用程序的名称。

5. 选择“免费”或“付费”选项。

这个版本的应用程序将无法更改该参数。

6。点击“添加”。



在编辑或上传应用程序的新版本时，您可以更新应用程序的价格。

如何更改应用程序价格

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择应用程序。

4. 点击“编辑”。
5. 输入应用程序的新价格。
6. 点击“提交审核”。

在将应用程序提交审核前，请务必检查应用程序是否符合应用程序要求。

将创建应用程序的新版本，并提交审核。审核通过后，应用程序的状态将改变，并出现“发布”按钮。

要发布应用程序，请点击“发布”。应用程序的新价格将在RuStore显示。

要设置应用程序内商品的价格，请了解如何在应用程序中创建付费商品。

要设置订阅价格，请了解如何为应用程序创建订阅。

付费应用程序中的佣金

RuStore的佣金为开发者收入的15%。

关于佣金大小的当前条款可以在“RuStore软件产品分发协议”部分找到。

应用程序要求

每个应用程序在RuStore发布前必须经过强制性审核。文中列出的所有要求对开发者来说都是必须的。不符合这些要求可能会导致应用程序发布被拒绝，或者如果违规在审核后被发现，可能会导致应用程序从商店中移除。

带有此图标的区块包含推荐执行的建议。

带有此图标的区块包含对要求的重要说明。

在发布前，开发者需要重新检查：

1. **应用程序的运行能力：**

- 应用程序必须能够启动并稳定运行，无故障和错误；
- 应用程序的所有声明功能都必须正确运行；
- 禁止发布引起故障、强制关闭、卡顿或错误运行的应用程序；
- 应用程序必须是自成一体的并具有原创内容。我们不欢迎唯一目的是重定向到网站（WebView）的应用程序。应用程序可以复制网站功能或使用WebView执行某些任务，但

它应被用户感知为一个完整的、独立的产品。如果我们怀疑应用程序发布的目的是吸引网站流量，我们可能会阻止访问它；

- 应用程序不应包含大量广告，这些广告会干扰应用程序的充分使用（例如，在用户每次操作后显示广告），或者不提供从屏幕上移除它们的方法；
- 如果应用程序中有授权功能，用户应能够成功完成授权，而不需依赖外部资源；
- 如果您的应用程序面向特定用户群体，您需要在向审核员的评论中提供测试账户的信息；

请不要忘记在“审核员评论”栏中提供授权的测试数据，供RuStore审核员检查功能。确保在演示模式中展示了应用程序的所有功能。

- 确保您的应用程序适配大多数现代移动设备 — 不允许有错误和应用程序布局不当；
- 确保您上传的是正确的应用程序版本 — 针对移动设备。

为了确保用户内容的及时性，RuStore保留阻止和（或）删除在其他商店中有更高版本的应用程序的权利。

- 如果应用程序是从RuStore下载的，当有新版本可用时，用户应收到通知，建议通过RuStore进行更新。

2. **应用程序内容：**

应用程序、其描述和功能不得违反俄罗斯联邦法律！

RuStore规则禁止发布违反俄罗斯联邦法律或用于规避法律规定的应用程序。RuStore保留拒绝发布违反俄罗斯联邦法律的应用程序的权利，并可能隐藏之前已发布的此类应用程序。

应用程序、其卡片以及其中的用户内容不得包含：

- 不道德的言论或观点：如明显基于宗教、种族、性取向、性别、民族和种族起源的歧视，以及暴力呼吁、非幽默背景下的侮辱等；
- 提议销售或分发非法商品（处方药品、毒品、武器、烟草和尼古丁制品及其交付等）和服务（卖淫、伪造文件、黑客服务、刷量等）；
- 禁止的色情内容，包括绘制的。此限制同样适用于交友应用和可能包含色情内容的其他应用；

- 真实描绘被杀害、伤害、折磨或虐待的人或动物的图像，以及其他令人震惊的内容，除非它们用于相应的主题和年龄类别（例如医疗内容）；
- 以任何形式和任何背景下的自杀呼吁，包括图像或美化自我伤害；
- 使用他人的商标和知识产权，包括可能误导用户的标志和名称；

如果您的应用程序是其他服务的官方合作伙伴，请提供此身份的文件证明。RuStore保留对证明知识产权的文件进行抽查的权利。

- 链接到其他应用商店或展示窗口，钓鱼和不相关的链接。

RuStore禁止发布与组织和进行赌博、在赌场接受赌注以及用真钱支付赢利有关的应用程序。

开发者发布具有发布用户内容的能力的应用程序必须确保对该内容进行及时和适当的预审核或根据用户投诉的追后审核。

具有用户内容或社交网络的应用程序必须包括：

- 不可接受内容的定义和标准；
- 有关冒犯性内容的报告机制；
- 对滥用服务的用户进行封锁；
- 与技术支持或审查团队的联系信息。

应用程序的版主应及时响应用户的问题和RuStore技术支持的查询。

如果应用程序包含来自可能显示成人内容的网站的用户内容，则该内容应默认隐藏，只对在网站上启用此内容的用户显示。

3. **应用程序安全性和用户权利保护：**

- 应用程序中不得包含恶意和网络钓鱼链接；
- 如果应用程序涉及个人数据处理，请确保您拥有个人数据操作者的官方地位。您还必须通知用户其数据正在被收集，并获得其同意进行数据处理；
- 如果您的应用程序中有购买功能，请向用户提供关于商品和服务支付、交付及退货的详细信息。遵守“消费者权益保护法”。

4. **隐私与设备数据：**

在应用程序运行过程中，您的应用程序可能会访问设备上用户的个人和敏感数据。

建议您查阅权限声明文档。在RuStore中，以下权限级别被接受：

- 禁止；
- 敏感；
- 安全；
- 过时。

如果在上传到“RuStore 控制台”时发现应用程序含有被禁止的权限（protection level — «not for use by third party applications»），该版本将被拒绝。APK文件需要进行修改，删除禁止的权限后重新上传。

如果在上传应用程序时发现敏感权限，开发者应在“敏感权限”模块中为每项权限的使用提供合理解释。

在处理权限时，请确保：

- 用户授权的数据不用于出售，也不传播或转让给第三方；
- 请求权限不是为了绕过设备隐私设置；
- 应用程序仅请求为提供服务所需的权限；
- 如果用户拒绝授权，您不应努力说服他。您可以提供不需要权限的替代方案。例如，建议用户自行在地图上查找最近的办公室，而不是基于设备位置数据；
- 用户对授权的同意应以主动行为的形式表达。例如，自动关闭的弹窗、故意或偶然退出对话框不视为同意；
- 请求的权限信息对用户可见。您可以在应用程序内提供此信息，并在RuStore商店的应用程序描述中发布链接或信息块；
- 您已向用户明确了所有请求的权限及其使用目的。如果发现不符合情况，我们可能会要求您提供并/或发布额外的解释。

我们要求您采取合理和有根据的措施，以防止您提供的服务被用于非法目的。如果不遵守这些要求，应用程序及其发布者可能会在RuStore中被封禁。

5. **应用程序类型：**

在发布应用程序时，请指明如果您的应用程序没有免费功能，则它是“付费的”。

免费应用程序可以提供付费订阅和应用程序内购买产品的功能。

在订阅的情况下，用户应该能够在订阅前使用应用程序和它的部分功能。同时，必须明确告知用户订阅的存在及其费用。

请参考“如何在应用程序中创建付费产品”和“如何为应用程序创建订阅”部分，以设置您的应用程序的付费内容。

请注意，故意误导用户可能会导致您的应用程序从RuStore中被删除。

6. 应用程序的名称、描述、图标和截图

6.1 在应用程序的名称和描述中，包括在版本变更历史中，不得包含：

- 诱人标题：避免使用“最好的”、“唯一的”、“禁止的”、“秘密的”、“最”的等类似词语。如果您在应用程序的描述中使用“官方”一词，我们将要求您提供该状态的文件证明；

在RuStore发布应用程序时，开发者必须确认应用程序及其内部的任何内容的使用不侵犯第三方权利。我们可以在检查的任何阶段要求确认这种权利。

避免使用其他文字和图形方式误导用户。

- 隐晦或直接的侮辱、脏话和其他粗俗语言；
- 其他人的商标和知识产权，包括标志和名称；
- 链接到其他应用商店或展示窗口，钓鱼和不相关的链接。

在将应用程序发送审核之前，请使用拼写检查服务来校正描述。

6.2 在应用程序的名称和简短描述中禁止使用表情符号和特殊字符，除非它们是品牌的一部分。

如果应用程序名称包含缩写或专业术语和缩写词，建议在简短描述中说明应用程序的用途。

应用程序在展示窗口和用户设备上安装时应该有相同的名称。

6.3 应用程序的详细描述应该清晰、连贯、易懂，并且用俄语撰写。术语和名称可能是个例外。描述中只应包含应用程序发布时可用的功能和内容。

建议在详细描述中不要使用表情符号、笑脸符号或重复的特殊字符。不应滥用关键词、大写字母或其他方式过度吸引用户注意。

6.4 图标不得：

- 模仿政府应用程序的图标；
- 使用他人的商标和知识产权，包括可能误导用户的标志和名称；
- 显示不存在于应用程序中的未读消息或通知的图标。

应用程序在展示窗口和用户设备上安装时应该有相同的图标。

6.5 截图应展示应用程序或游戏的实际机制和功能，而不仅仅是标题、登录页面、开场画面或艺术作品。不允许形成仅由艺术作品或与应用程序界面无关的元素组成的截图序列。

对于游戏应用程序，截图应展示当前的图形水平。

截图不得包含：

- Android或其他应用程序的界面元素；
- 应用程序中还不存在或已不存在的功能。

6.6 视频要求：

- 视频应直接与应用程序相关（例如，您可以上传预告片、游戏玩法录像等）；
- 视频不得包含不允许的内容（参见第2条。应用程序内容）；
- 视频必须是公开的；
- 放置视频的页面必须对所有用户开放。如果您添加的视频来自隐私设置受限的页面，则必须将视频移到一个开放访问的群组或公共页面。
- 只能添加来自VK Video服务的视频链接；
- 视频时长不得超过1分钟；
- 只能添加一个视频。

6.7 应用程序的分类：

- 应用程序应具有正确的类别。请查阅可能的类别列表。如果类别指定不正确，管理员可以更改应用程序的类别；
- 应用程序应具有正确的年龄类别。如果您不确定应该设置哪种年龄限制，请查阅年龄限制信息。

ASO建议

影响RuStore发展的一个主要因素是应用程序的可用性。为了让用户能够找到您的应用程序，详细描述和有趣的设计非常重要。这将有助于增加安装数量并改善应用程序列表和搜索中的位置。

RuStore使用多个参数来形成应用程序搜索结果。以下是我们想要特别强调的一些参数：

- 评分。评分越高，应用程序在列表中的位置就越高。对于开发者来说，积极回应用户的评论非常重要。RuStore进行自动和手动评分审核，以确保没有操纵评分的行为。您可以通过使用RuStore SDK来连接评论和评分，请求用户增加评价数量。
- 安装数量。数量越多，应用程序进入顶部的可能性就越高。但是，不应该使用刷量手段，只有真实的有机流量才重要。为了推广应用程序，开发者可以在自己的资源上放置下载按钮或使用广告。这将有助于吸引观众并提高位置。
- **转化为安装的数量。**RuStore建议立即设计您的应用程序，以便获得高转化率。重要的是正确设计截图，在描述中添加足够的数量，并维护“新功能”部分。这将有助于提高应用程序的下载量。
- **评分数量及其分布。**正面评分越多越好！RuStore建议及时回复用户，改进应用程序以增强其功能，鼓励用户进行评分和撰写评论。这也将帮助应用程序在搜索结果中排名更高。必须记住，RuStore会追踪刷分行为并制止其使用，这只会造成伤害。RuStore SDK用于连接评论和评分，可以简化处理评论的工作，自然地吸引用户直接在应用程序中留下它们。

年龄限制

在RuStore发布的应用程序有一个年龄限制系统。这些限制帮助用户理解应用程序是否包含某个特定年龄段的人可能不适宜的内容。

年龄限制并不提供有关应用程序目标群体的信息。

如果更新带来了特定年龄群体可能不适宜的内容，请在发布应用程序或其新版本时指定年龄限制。这一信息将帮助RuStore的用户决定是否安装应用程序。

如何指定年龄限制

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择应用程序。
4. 选择应用程序的版本。

5. 点击  。
6. 找到“年龄类别”子部分。
7. 在下拉菜单中选择一个限制选项。
8. 点击“保存”。

如何选择限制

RuStore使用以下年龄限制：

限制	特征
0+	无暴力，不雅内容（侮辱、粗话）。请注意，并非所有带有此年龄限制的应用程序都是专门为儿童开发的。
6+	某些暴力或恐怖场面。不建议6岁以下儿童使用。请注意，并非所有带有此年龄限制的应用程序都是专门为儿童开发的。
12+	适度的暴力，恐怖场景，不雅内容。不建议12岁以下儿童使用。
16+	暴力或性内容。不建议16岁以下儿童使用。
18+	高水平的暴力，性内容，提及毒品或酒精。只推荐给成人。

年龄限制并不代表游戏的难度。建议父母或监护人参考年龄限制，以确保游戏内容适合孩子。

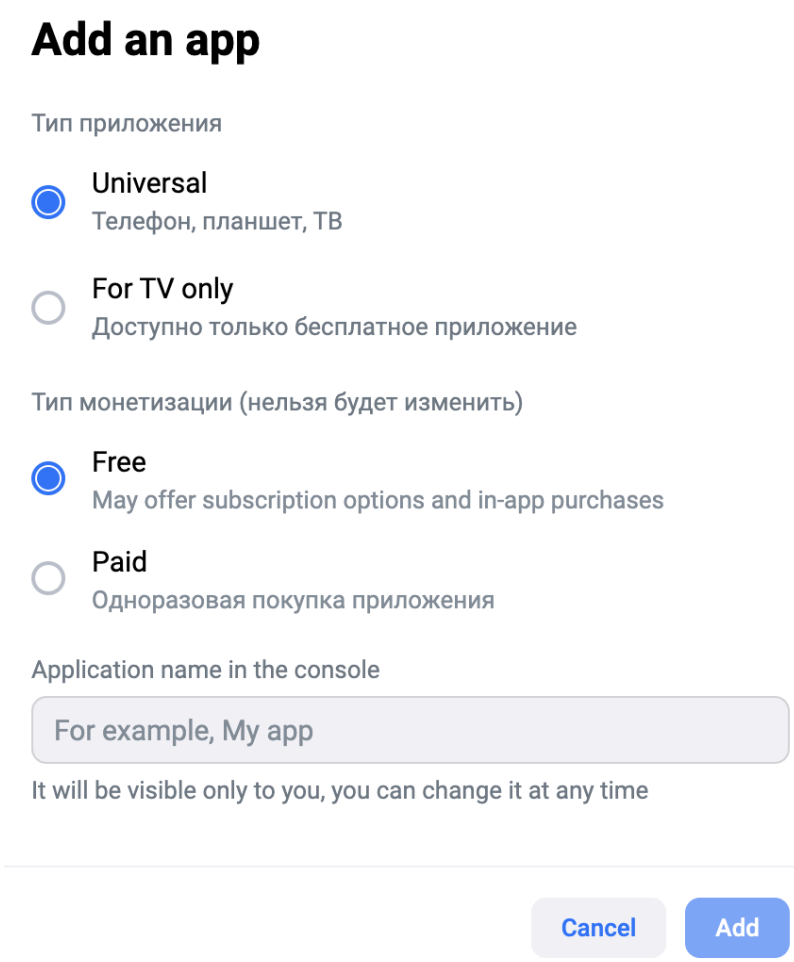
RuStore应用程序类别指南

这里收集了关于哪些应用程序属于每个类别的总体信息，以及一些示例以便直观了解。在选择要上传的应用程序的类别并提交审核之前，请仔细研究这个表格。

如何选择类别

1. 打开RuStore 控制台系统。

2. 转到“应用程序”选项卡。
3. 点击“添加应用程序”。
4. 在弹出的窗口中输入应用程序的名称。
5. 在“类型”区块中选择您的应用程序类型：非游戏应用程序和游戏。第一种类型的应用程序位于RuStore主页的“应用程序”标签下，第二种位于“游戏”标签下。
6. 根据您的应用程序特点在下拉菜单中选择主要类别。应用程序将被放置在商店中选择的类别中。
7. 选择一个额外的类别。应用程序将同时出现在商店的两个类别中。



对于应用程序，您可以选择两个类别 — 主要和附加：主要类别指示最相关的，而附加类别可能不存在。如果您对选择有疑问，请查看下面的表格。

在审核阶段，支持专家将根据需要更正应用程序的类别。

请注意“要求”列：其中列出了在指定类别时必须考虑的要求和说明。

这些要求也是审核人员遵循的指导方针。这些规则帮助我们在类别中正确设置搜索结果，并确保所有应用程序在RuStore中得到公正的布置。

按照指南，只为应用程序指定相关的类别，以使发布过程尽可能快速高效，并使用户更容易找到应用程序。

应用程序类别

1。商务服务

描述：公司员工的内部服务，包括培训和商业管理工具 - 会计、物流、人员/采购管理、CRM。特殊工具 - 门测量者、出租车和货车司机应用程序、快递员等。带有广告观看货币化或其他赚钱方式的应用程序。

要求：不要将这一类别的应用程序划归为“有用工具”的附加类别。不需要将快递员、出租车司机和其他交通工具的应用程序额外归入“交通”类别。为商务用途的银行应用程序指定“金融”作为主要类别，可以额外指定“商务服务”作为附加类别。

示例：

- Yandex Pro：司机和快递员；
- Delivery Club：快递员工作；
- 俄农业银行员工办公室；
- 1C：文件流转；
- 门测量工；
- 小型仓库。

2。政府服务

描述：政府机构的应用程序，以及处理文件、记账和纳税、政府机构和多功能服务中心的在线预约、社会国家项目的服务。

要求：此类别只分配给政府开发者的应用程序。

示例：

- Gosuslugi（国家服务）；
- 国家服务汽车；
- 中央银行在线；
- 统一自动采购管理系统门户；
- 个人税务；
- 俄罗斯银行钞票。

3。食品和饮料

描述：咖啡馆和餐厅的应用程序，食品和产品配送服务，餐桌预订，食谱集。

要求：不要指定“购物”为附加类别。

示例：

- Samokat；
- VkusVill；
- Yandex Lavka；
- Pyaterochka；
- 汉堡王；
- Domino’s Pizza。

4。健康

描述：药房应用程序，寻找并预约医生或心理医生的服务，寻找和订购药物，休闲和放松应用程序，饮食和营养控制，健康、睡眠和习惯追踪器。

要求：健身和家庭训练应用程序属于“体育”类别 — 不要将“健康”作为附加类别。不要为药房指定“购物”作为附加类别，对于健康追踪器不需要指定“生活方式”类别。

示例：

- 看医生；
- 看医生 — 在线预约；
- 奥泽尔基药房；
- 急救；
- Pro U — 营养构建器；
- 过敏日记；
- 经期日历；
- 减压方形呼吸法。

5。书籍

描述：包括宗教书籍、漫画和有声读物的书籍。

要求：儿童童话应用程序归属于“家长”类别。可以额外为它们指定“书籍”类别。

示例：

- LitRes；
- 有声读物；
- beeline书籍和有声读物；
- Zen；
- PowerAudioBook；
- Inspirebook。

6。新闻和事件

描述：新闻门户和城市新闻服务，各城市或地区的应用程序带有新闻部分，事件海报以及购买活动门票。

示例：

- RIA新闻；
- Mail.ru新闻；
- Lenta.ru；
- 俄罗斯报纸；
- 我的城市伏尔加格勒；
- Afisha（海报）。

7。生活方式

描述：爱好和兴趣、宗教和神秘学、星座、心理学和其他非专业测试、手工艺、贺卡和祝贺、讽刺激励图片和图片。

要求：将宗教书籍和有声读物的主要类别指定为“书籍”。不要为健康追踪器、女性日历和“健康”类别中的其他应用程序指定额外的“生活方式”类别。可以为“交通”类别中的自行车和电动滑板车租赁服务额外指定“生活方式”类别。

示例：

- 园艺指南；
- 钓鱼手册
- 根据天气情况钓鱼；
- 数字命理学和塔罗牌；
- 生日祝福；
- 音乐与脉轮；
- 心理测试；
- 趣味笑话、恶搞、图片。

8。教育

描述：适用于专业教育的应用程序、针对学生和大学生的应用程序、外语学习服务、培训课程、专业资格认证和考试。

要求：学习交通规则和驾驶培训的应用程序应仅归类于“交通”类别。对于学生的电子学习日记，您可以额外选择“家长”类别。

示例：

- SkyEng；
- 教育课程 — Beeline；
- 2023年高考俄语文学；
- 投资培训；
- Dnevach — 课程时间表；
- Uchi.ru。

9。交流

描述：社交网络和博客，即时通讯软件，约会服务。

要求：不要为约会应用程序指定额外的“娱乐”类别。不要为即时通讯软件指定额外的“实用工具”类别。对于具有搜索/播放/创建视觉或音频内容功能的社交网络，可以指定额外的“娱乐”类别。为视频通话和会议指定“实用工具”作为主要类别，并可以额外添加“交流”类别。

示例：

- VK；
- Odnoklassniki (同学)；
- ЯRus；
- Yappy；
- Galaxy约会聊天；
- Pikabu。

10。广告和服务

描述：带有广告和专家搜索的服务、邮件和货物配送、快递服务、美容院应用程序、干洗、清洁服务、修理和其他服务。

要求：不要为应用程序指定额外的“购物”类别。对于快递服务，可以额外选择“交通”类别。

示例：

- DomKlik（房屋点击）；
- Sutochno.ru (按夜租赁)；
- HH.ru (猎头)；
- Auto.ru (汽车)；
- Yandex 不动产；
- CDEK；
- MoiProfi (我的专家)。

11。宠物

描述：兽医预约服务和在线兽医咨询，寻找动物和支持动物收容所的宠物广告，学习动物品种的应用程序。

示例：

- Vetexpert — 24/7兽医；
- 狂犬病 — 疫苗接种时间表；
- Gooddog — 来自收容所的宠物；
- 犬种；
- 寻找动物；

- 找个爱心家庭。

12。购物

描述：商店和市场应用程序，购物清单服务，折扣和返现服务。

要求：对于药店应用程序，仅选择“健康”类别。对于食品和产品配送，餐厅和咖啡馆的应用程序，只选择“食品和饮料”类别。对于销售广告的应用程序，选择“广告和服务”类别。

示例：

- Yandex.Market (Yandex市场)；
- SberMegaMarket；
- Ozon；
- Chitai-gorod (阅读城市)；
- Eldorado；
- Flor2U；
- Rive Gauche；
- 你好，世界！— 促销和返现；
- 价格比较器：价格，折扣，促销。

13。实用工具

描述：用于处理文档的应用程序，任务计划器，通讯和互联网服务，视频通话和视频会议。智能家居和物业管理公司的公共事业支付应用程序。照片和视频编辑器，手机壁纸和小工具，天气，手电筒，键盘，实用程序等。

要求：对于即时通讯软件，只选择“交流”类别。用于商业管理的工具（CRM，会计等）属于“商务服务”类别。不要指定额外的“实用工具”类别。每个应用程序都是一个实用工具，不建议过度填充这一类别。

示例：

- 天气；
- 键盘；
- 语音助手；
- 日历；
- 密码生成器和管理器；
- 文本编辑器；
- 浏览器；
- 通信运营商。

14。旅行

描述：为游客设计的应用程序，购买机票和旅游团的服务，酒店预订，住宿租赁，以及旅游导航器，旅行团。

要求：不要为购票和旅游团服务指定额外的“交通”类别。对于带有地图的旅游导航器，您可以额外选择“交通”类别。

示例：

- Aeroflot；
- S7；
- 铁路票务；
- 向俄罗斯铁路乘客；
- 折扣旅游；
- Карта.ру (地图.ру)；
- 俄罗斯名胜。

15。娱乐

描述：用于收听音乐和广播的应用程序、电视频道和视频服务、在线电影院、卡拉OK、用于流媒体和游戏的应用程序、在电影院和剧院购票、寻找活动和其他娱乐活动的应用程序，以及其他不属于其他类别的娱乐服务。

不要为游戏选择“娱乐”类别。在RuStore中，它们有单独的类别。

要求：对于有声读物，请选择“书籍”作为主要类别。对于带有体育赛事直播的应用程序，除了主要类别“体育”外，还可以指定附加类别“娱乐”。对于具有搜索/播放/创建视觉或音频内容功能的社交网络，除了主要类别“交流”外，还可以指定附加类别“娱乐”。不要为约会应用程序指定额外的“娱乐”类别。

示例：

- VK音乐；
- Yandex.音乐；
- Zaycev.net；
- Okko；
- Kinopoisk (影视搜索)；
- Rutube；
- VK Play；
- Afisha (海报)。

16。家长

描述：适用于儿童和父母的应用程序、儿童位置追踪器和定位器，以及孕期管理和儿童发展服务、母乳喂养日记、摇篮曲和助眠白噪声。

要求：对于儿童童话应用程序，请选择“家长”作为主要类别，并可额外选择“书籍”类别。对于学生的电子学习日记，请选择“教育”作为主要类别。您还可以额外选择“家长”类别。

示例：

- 婴儿日记 – 儿童模式；
- 我的孩子在哪里：GPS-跟踪器 0+；
- 摇篮诗；
- 数到1000；
- 学习阅读；
- 我怀孕了；
- 言语治疗师 – 发音启动。

17。体育

描述：体育俱乐部和健身中心的应用程序、体育新闻和直播服务、健身训练。

要求：对于博彩应用程序，只选择‘投注和彩票’类别。不要为健身训练应用程序指定额外的‘健康’类别。可以为体育赛事直播应用程序额外指定‘娱乐’类别。

示例：

- Match TV （比赛电视）；
- Sports.ru：体育新闻；
- FC Spartak （斯巴达克足球俱乐部）；
- Sport24；
- 足球 — 新闻和结果；
- Gym up — 训练日记；
- 俯卧撑计数器。

18。投注和彩票

描述：含有彩票和体育投注的应用程序。

要求：不要为博彩应用程序和彩票指定任何额外的类别。

示例：

- Fonbet — 体育投注；
- 投注联盟；
- 1 x 投注 — 体育投注；
- Stoloto（一百种彩票）。

19。交通和导航

描述：出租车服务、共享汽车服务和拼车服务、为驾驶员和乘客设计的应用程序、地图和导航器、公共交通跟踪服务和充值交通票务、租赁电动滑板车和自行车服务，以及学习交通规则和驾驶培训的应用程序，OSAGO（强制汽车民事责任保险）和CASCO（全面车辆保险）保险。

要求：对于学习交通规则和驾驶培训的应用程序，不需要指定“教育”类别。对于邮件和货物递送、快递服务的应用程序，请选择主要类别“广告和服务”。您还可以为它们额外指定“交通”类别。

您还可以为带地图的旅游导航器额外指定“交通”类别。不要为购买旅行票务和旅行服务的服务指定额外的“交通”类别，只归类于“旅行”。不要为出租车驾驶员、交通公司员工和快递员应用程序指定额外的“交通”类别。只归类于“商务服务”。可以为自行车和电动滑板车租赁服务额外指定“生活方式”类别。

示例：

- 2GIS；
- Yandex 地图和导航器；
- Yandex Go；
- Citymobil；
- 交通警察罚款；
- Delimobil — 你的共享汽车；
- YouRent （你租）。

20。金融

描述：银行和金融机构的应用程序、用于账目核算和预算管理、资金转账、贷款和信用卡的服务。

要求：

不要为通过广告观看和其他方式实现货币化的应用程序指定“金融”类别 — 这些应用程序属于“商务服务”类别。

对于商业银行应用程序，可以额外指定“商务服务”作为附加类别。

示例：

- Sberbank with Salut；
- Tinkoff Bank；
- VTB；
- 个人财务；
- 贷款计算器；
- 财务 — 预算、开支；
- Alfa-Business。

游戏类别

对于所有游戏，以下内容是禁止的：现实的暴力场景、大量的不规范词汇、震惊和冒犯性内容、宣扬对人群的仇恨、暴力或歧视，基于族群、宗教、性别、年龄和性取向的歧视，诽谤性的言论或侮辱性的声明，以及其他违反俄罗斯联邦法律的内容。

1。街机游戏

描述：快速解决虚拟任务的游戏。玩家需要迅速准确地行动，成功解决任务后可获得奖励：积分、角色新技能或游戏世界的元素等。

示例：

- 跑动的小球；
- 红线；
- 猫医生；
- 僵尸驾驶；
- Simple Snake；
- Dino Trex。

2。知识竞赛

描述：在有限的时间内从不同知识领域寻找问题的答案：包括带有文字的谜题和测验。任务以文本、图形或音频格式呈现给玩家。

示例：

- 猜单词；
- 谜语；
- 与Dino一起学数学；
- 知识竞赛。知识竞赛-对战；
- Quiz Kids；
- TriviaChallenge。

3。益智游戏

描述：游戏玩法包括向玩家提供不同难度的谜题解决方案：选择合适的元素和寻找逻辑顺序。

示例：

- 数独；
- 纸牌；
- 寻找单词：填字游戏；

- 球体排序；
- 开锁游戏；
- Wobbler。

4。赛车

描述：玩家需要控制各种类型的交通工具，并与机器人或其他玩家在各种赛道上进行速度竞赛。

示例：

- 3D巴士模拟器；
- Huggy Wuggy 摩托越野；
- Shift Race：变形赛车；
- Kart Race；
- Gravity Moto Race；
- Traffic Racer。

5。儿童

描述：旨在促进儿童发展和丰富他们休闲时间的游戏。特点是简单的难度，大而清晰的图形元素，以及直观的操作。

示例：

- 儿童游戏。学习Luntik；
- 儿童数学。Fixiki；
- 儿童英语。Fixiki；
- 儿童单词搜索；
- 儿童绘画；
- Quiz Kids。

6。带AR的游戏

描述：利用增强现实技术将部分游戏过程从虚拟世界转移到现实世界中。利用周围的元素帮助玩家完成各种任务。

示例：

- AR Travel；
- AR Gotek New Year；
- GimaQuest；
- AR Tourism；
- Amusement Park VR Game；
- KidZLab。

7。独立游戏

描述：由独立作者和小团队创建的游戏项目，无出版商帮助。

示例：

- 大脑训练器；
- 书架；
- 神秘之地；
- 吉萨之梦；
- Escape Sideways；
- Running Fish。

8。赌场

描述：模拟赌场中流行的游戏：扑克、老虎机、轮盘等。玩家的目标是获得最多的分数或虚拟货币。

示例：

- Poker Games: World Poker Club；
- Fruit paradise老虎机游戏；
- Livegames Club；
- Fred’s Slots；
- Pocket Chips：扑克筹码。

9。休闲游戏

描述：具有简单游戏玩法、直观控制和清晰胜利条件的游戏。玩家不需要做出太多操作即可获胜。适合短暂的游戏体验。

示例：

- 花园主：农场与冒险；
- 遗忘的宝藏2：三消游戏；
- 弹球：Bubble Breaker；
- 彩色线条；
- 寻找差异；
- Bridge Stick。

10。纸牌游戏

描述：使用基本的一套卡牌，玩家需要使用这些卡牌来赢得游戏。游戏既可以与机器人进行，也可以与真实对手进行。

示例：

- 纸牌单人游戏合集；
- 扑克。在线对决；

- 千（1000）；
- 笨蛋在线3D；
- 经典纸牌-7；
- Livegames Club。

11。音乐

描述：带有各种音效的应用程序，以及用于学习演奏乐器的应用。

示例：

- GLGuitar：吉他模拟器；
- 猫咪的声音，成猫和小猫：喵叫声，呼噜声；
- 给幼儿的动物声音；
- 汽车声音：喇叭，警报器，钥匙链，刹车声；
- Breakout 3D；
- Ice Samurai。

12。桌游

描述：流行桌面游戏的虚拟对应版本，其中玩家轮流在有限的游戏场地上行动，遵循预先设定的规则。

示例：

- 国际象棋；
- 18世纪海战游戏；
- 迷你台球；
- 扑克；
- 麻将；
- 15块滑动拼图游戏。

13。冒险游戏

描述：一种基于与周围虚拟世界或角色互动来解谜的游戏类型。为了帮助玩家，游戏提供了提示和建议。

示例：

- 花园主：农场与冒险；
- Megacraft：我的世界创造模式；
- Slaughter；
- Raid & Rush；
- 镜中世界；
- 奇迹世界：合成与魔法。

14。角色扮演游戏

描述：玩家需要通过完成各种任务和与游戏世界中的元素互动来发展虚拟角色。接下来要寻找各种物品，与敌人战斗，创建自己的世界元素。

示例：

- 野蛮人：血与荣耀；
- 英雄公会：动作 MMO RPG；
- 我的农场；
- 坦克：钢铁军团；
- 科技魔法；
- 法师之战：新纪元。

15。家庭游戏

描述：适合孩子和父母一起玩的游戏。它们以简单的剧情和任务以及直观的控制元素为特点。

示例：

- 学习阅读 Azbuka Kikoriki；
- 儿童英语。Fixiki；
- 我们的电影 — 照片智力竞赛；
- 海底世界的宝藏；
- 组合和匹配；
- Tasty World: 女孩的厨房。

16。模拟器

描述：在虚拟世界中最准确地呈现现实的游戏。操作技术和设备，参加比赛和完成不同职业的任务。

示例：

- World of Fishers钓鱼；
- 3D巴士模拟器；
- Tank Force；
- 欢乐农场；
- 梦想摩天大楼：新高度；
- 鲍勃的梦想咖啡厅：Bikini Bottom。

17。词语游戏

描述：提供各种与单词相关的任务的游戏：寻找序列，猜单词等。

示例：

- Krossy — 俄语填字游戏；
- 填充单词 — 寻找单词的游戏；
- 猜单词；
- 单词迷宫：寻找单词；
- 每日填字游戏；
- 字母。

18。体育游戏

描述：体育游戏和运动项目的模拟。玩家将成为团队的一员、教练或经理。

示例：

- 足球经理 2023；
- 极限篮球。布娃娃物理扣篮；
- 迷你台球；
- 3D Gorodki；
- 多米诺骨牌；
- 俄罗斯足球智力竞赛。

19。策略游戏

描述：需要选择最佳策略来实现设定目标的游戏，无论是实时还是回合制。通常，玩家将不仅控制一个角色，而是一个团队 — 一个军事单位、专家团队、企业、宇宙等等。

示例：

- Last Day on Earth: Survival；
- Armor Age：第二次世界大战坦克；
- 战争与金钱英雄：策略；
- Space Arena: 组装飞船；
- 法师之战：新纪元；
- Caribbean Pirates。

20。动作游戏

描述：玩家在游戏中与敌人或危险物品互动的游戏。他将处于故事的中心，并与敌人战斗。

示例：

- Warface: GO；
- Tank Force；
- 英雄公会：动作 MMO RPG；
- Art of War 3: RTS策略；
- Inspire Helicopter；
- Gun War Z2。

用户评分和评论的管理

用户从RuStore下载应用程序后，可以对其进行评分和留下评论。一个用户只能添加一次评分和评论，但随时可以更改它们。

现在在RuStore 控制台系统中，您可以跟踪评分和评论。未来，我们将添加分析这些数据的工具。

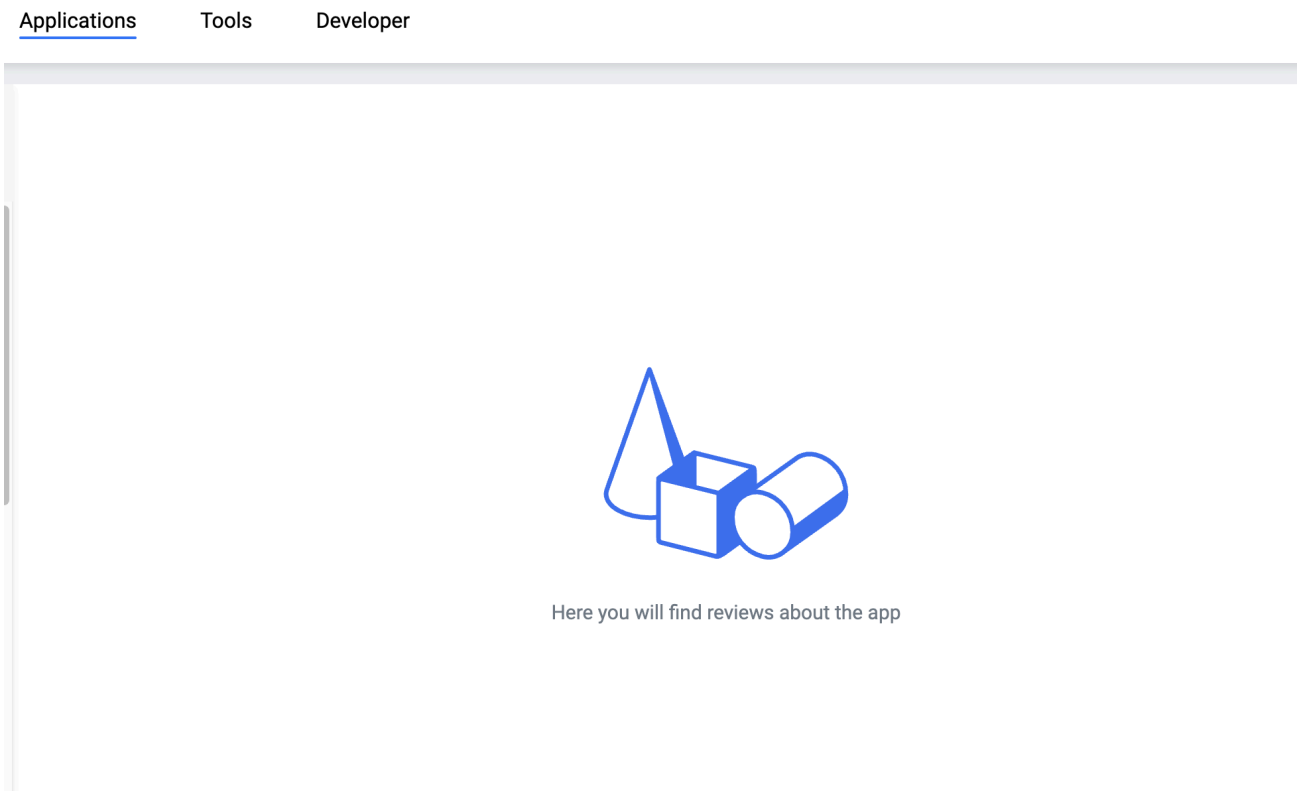
您可以对评论撰写公开回复。这个功能是为了帮助您与用户建立关系。您可以：

- 感谢用户的好评；
- 回答问题；
- 了解用户希望在您的应用程序中使用哪些功能；
- 了解应用程序中的错误。

如何回复评论

要回复评论，请：

1. 打开RuStore 控制台。
2. 在左侧面板上点击“评论”选项卡。
3. 点击您想回复的评论下方的“回复”。
4. 在“开发者回复”框中输入回复内容，然后点击“发布”。



您的回复将被发送进行审核，并将被标记为“审核中”。审核通过后，状态将变更为“已发布”。

编辑和删除对用户评论的回复

如需，您可以随时编辑或删除回复。

如何编辑您的评论回复

1. 打开RuStore 控制台。
2. 在左侧面板上点击“评论”选项卡。
3. 点击您的评论回复下的“编辑”。
4. 在“开发者回复”框中进行必要的更改，然后点击“发布”。

修改后，您的回复将重新提交审核，并在审核通过后发布。

如何删除您的评论回复

1. 打开RuStore 控制台。
2. 在左侧面板上点击“评论”选项卡。
3. 点击评论回复下的“删除”，它将被删除。

回复评论的要求

1. 不要偏离主题。回复应清晰、真实、有用并符合评论内容。
2. 保持礼貌。您的目标是帮助用户找到问题的解决方案，而不是将他们赶走。不要发表带有侮辱、威胁、轻视或歧视性言论的评论回复。不要激怒用户留下类似评论。
3. 回复所有评论。这样可以向用户展示您重视他们的意见。通常，那些回复评论的开发者的应用程序会获得更高的评分。
4. 如果用户表现出极度消极态度，不要以消极回应。如何回复负面评论？
5. 努力加快处理正面评论，这样您将有更多时间解决用户的复杂问题。例如，可以在本地电脑上创建快速回复的模板并插入现成的回复。目前RuStore尚无内置模板，但我们将确保添加此功能。
6. 如果您使用快速回复模板，请努力改变从一条评论到另一条评论的措辞。用户不应感觉到是机器人在回复他。
7. 不要在回复中发布链接到第三方应用商店或展示窗口、网络钓鱼和不相关的链接。
8. 不要在回复中发布其他服务或应用程序内购买的广告和促销代码，也不要提供从其他商店或展示窗口下载应用程序的建议。用户通常不认为此类信息有用。
9. 不要发布性的内容，也不要使用不礼貌的表达。

违反这些要求可能导致应用程序或整个开发者账户被封禁。

处理负面评论

如果正确回应评论，可以将用户对应用程序的看法改变为更好的方向。

如何回应负面评论

1. 避免简单的、没有道歉的回答。通常，一行回复对愤怒的用户来说似乎并不友好，这会显得粗鲁并可能引发更多负面反应。
2. 不要用官方口吻回答。正式的表达可能会让用户觉得回复者不是人，而是机器人。
3. 重要的是要表现出您理解并共情用户的感受。如果问题出在您的应用程序上，应该承认错误。
4. 努力在回答中帮助用户。尽量不要在回应负面评论时使用快速回复模板。总是需要考虑上下文，看看模板回复是否适合该情况。
5. 感谢用户报告应用程序中的错误。这样的评论可以帮助您使应用程序更加方便，从而让人们更想使用它。
6. 向用户解释为什么在应用程序中采取了这样的问题解决方案。有时用户不明白为什么您移除了他们喜欢的功能。如果您能够合理解释，可能会减少用户的负面情绪。
7. 快速回应评论。您越早帮助用户解决问题，他们继续使用应用程序并提高评分的可能性就越大。

如果您发现包含不客观信息、脏话或侮辱的评论，请告知我们。我们将尽力解决问题。

如何收集HAR日志

HAR日志（HTTP-归档）包含浏览器加载特定网页时所做的所有网络请求和响应的记录。其中包括请求资源的URL地址、用于请求的HTTP方法、响应代码等数据。

HAR日志用于诊断网页加载问题、发现加载速度慢的资源和网络错误、监控性能和优化网站、比较不同页面的性能。

大多数浏览器允许将HAR收集到单独的文件中，然后可以将其传给技术支持，以了解错误发生的原因。

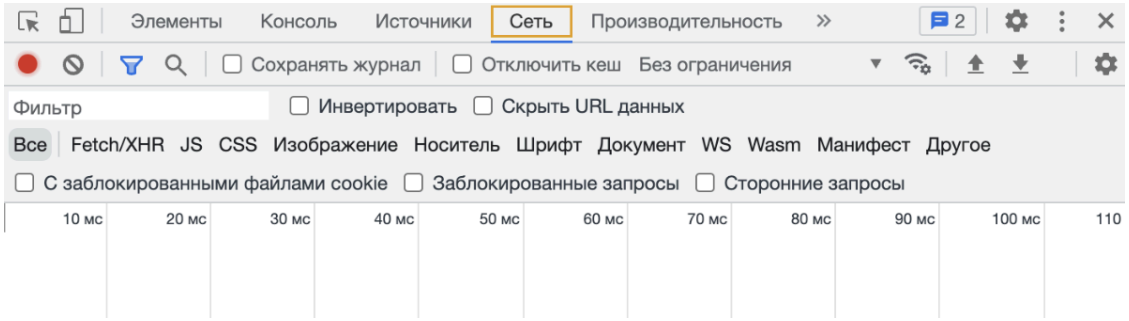
开始之前，请在控制台开启日志记录，以准确记录错误。

Google Chrome

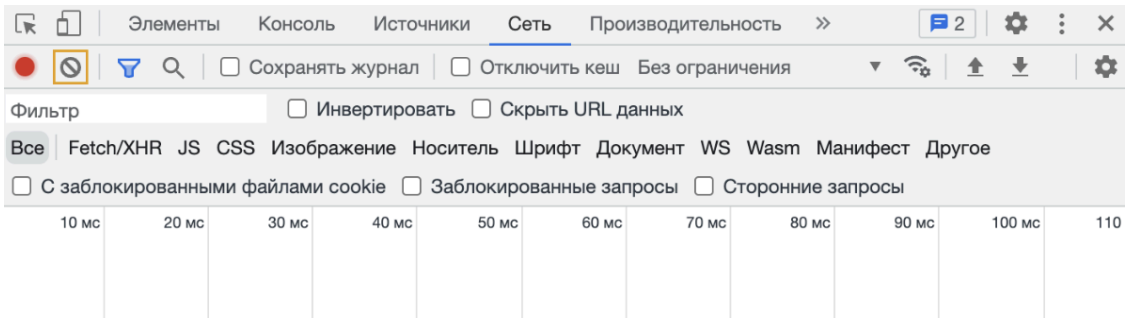
为了收集HAR日志：

1。按键盘上的F12。将打开开发者工具窗口。

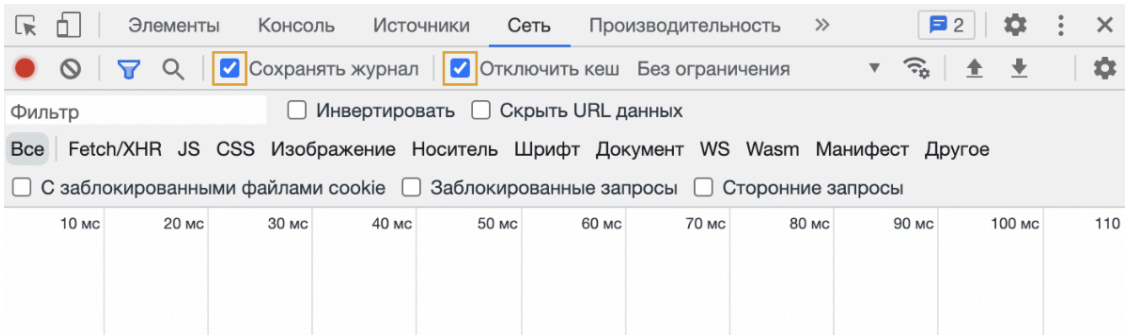
2。选择“网络”/“Network”选项卡。



3。点击  ，以清除网络日志。

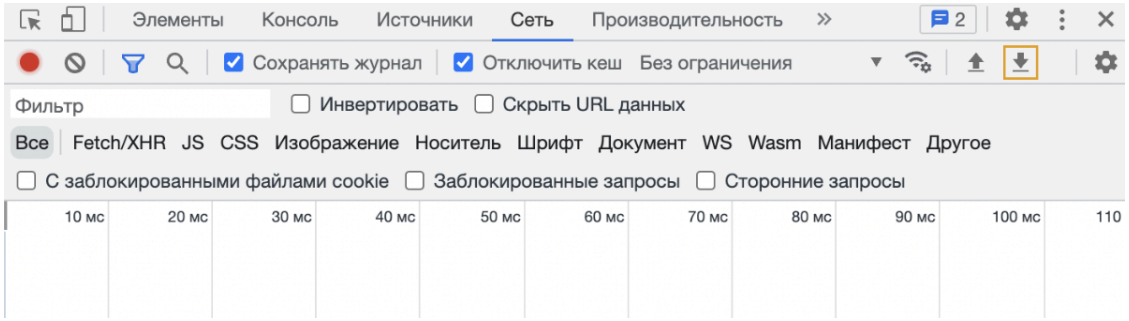


4。在“保持日志”/“Preserve log”和“禁用缓存”/“Disable cache”的字段中设置勾选。



5。重复导致问题的操作。

6。点击  ，以保存HAR日志。



7。在弹出的窗口中选择保存文件夹。

8。点击“保存”。

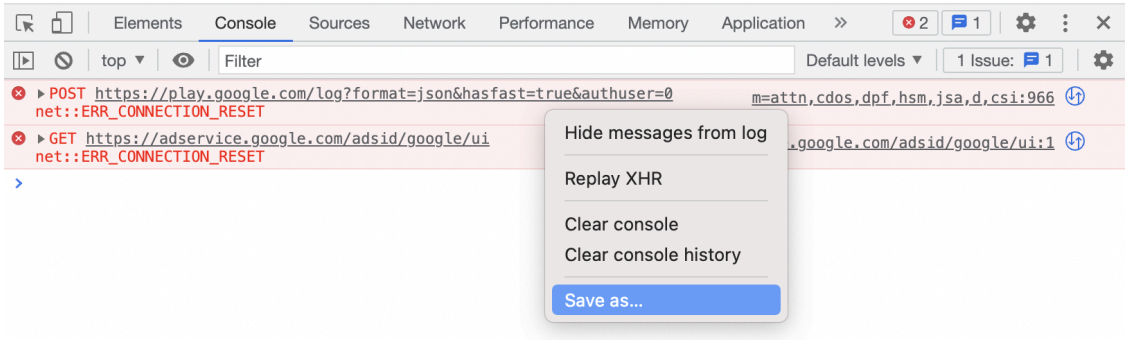
HAR日志将被收集并保存在指定文件夹中。

为了收集控制台日志：

1。按键盘上的F12，以打开开发者工具窗口。

2。转到“Console”选项卡。

3。在窗口的任意区域右键点击，然后在上下文菜单中选择“Save as”。



4。选择保存日志的文件夹，然后点击“保存”。

控制台日志将被收集并保存在指定文件夹中。

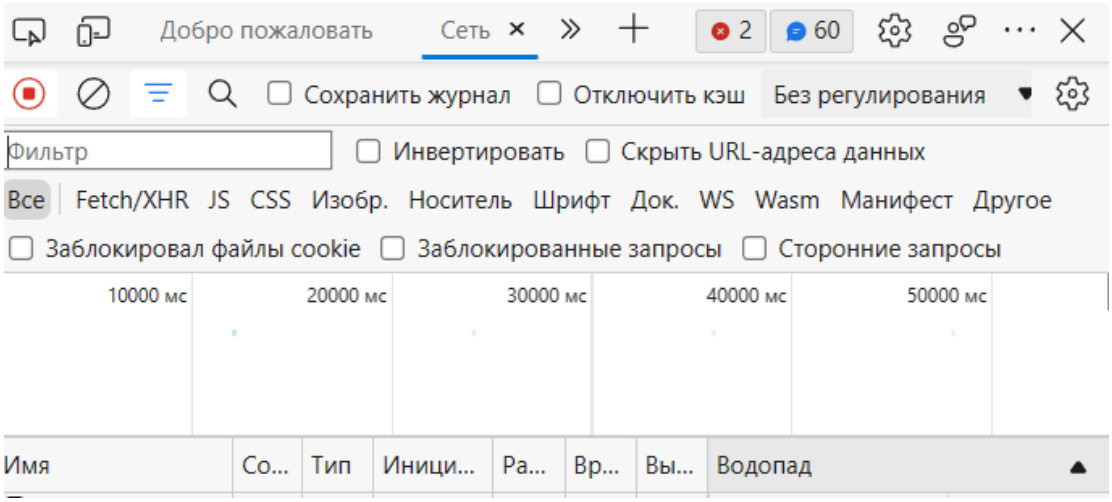
Microsoft Edge

为了收集HAR日志：

1。按键盘上的F12，以打开开发者工具窗口。

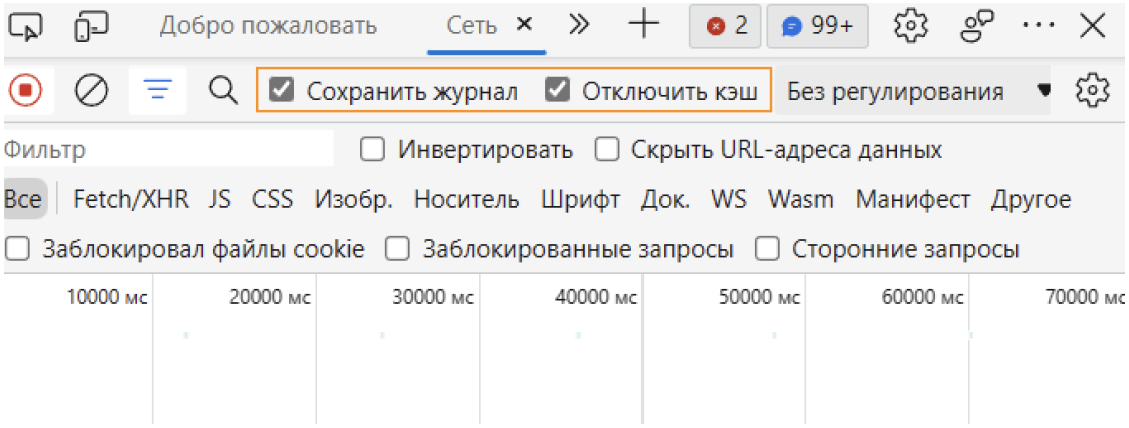
2。点击“开启开发者工具”。

3。打开下拉菜单并选择“网络”。



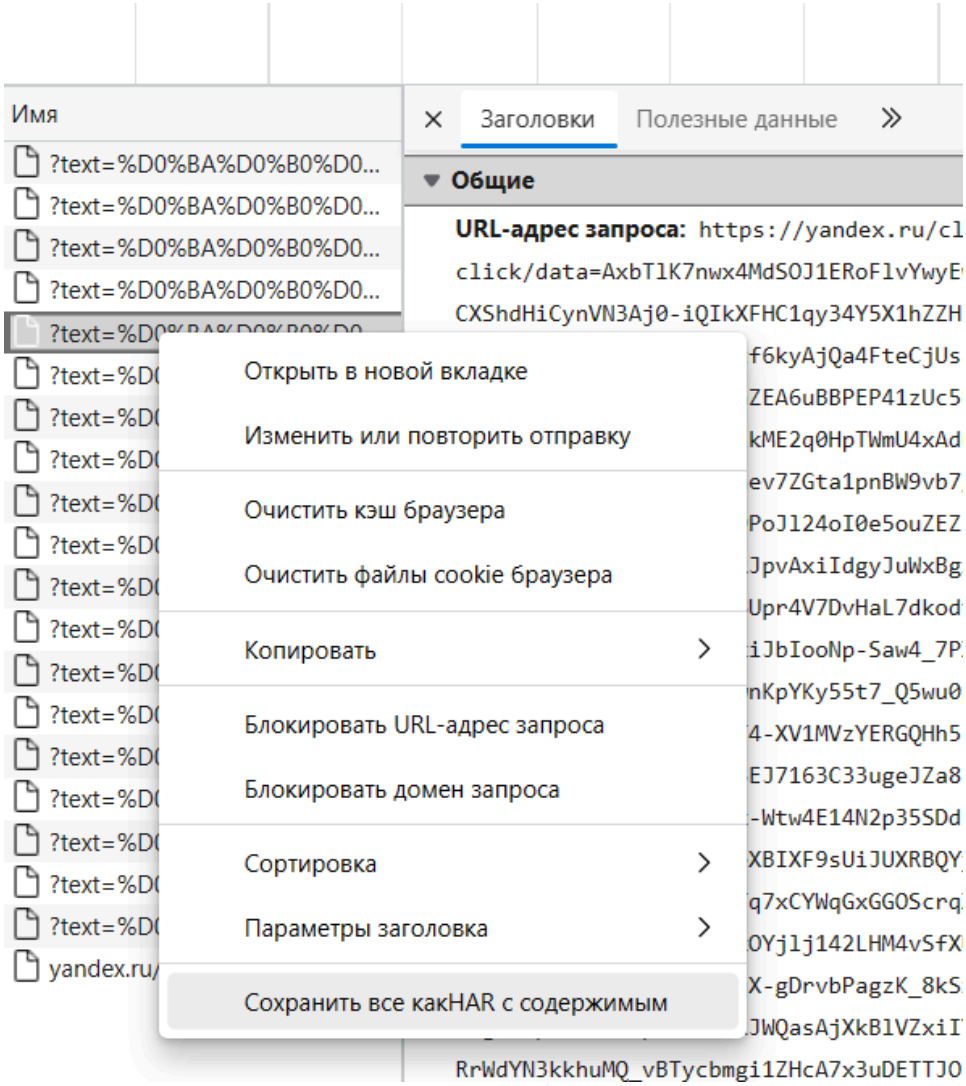
4。点击  ，以清除网络日志。

5。设置勾选“保持日志”和“禁用缓存”。



6。重现问题。

7。在任意日志上右键点击，选择“保存所有内容为HAR”，以保存HAR日志。



8。选择保存日志的文件夹，然后点击“保存”。

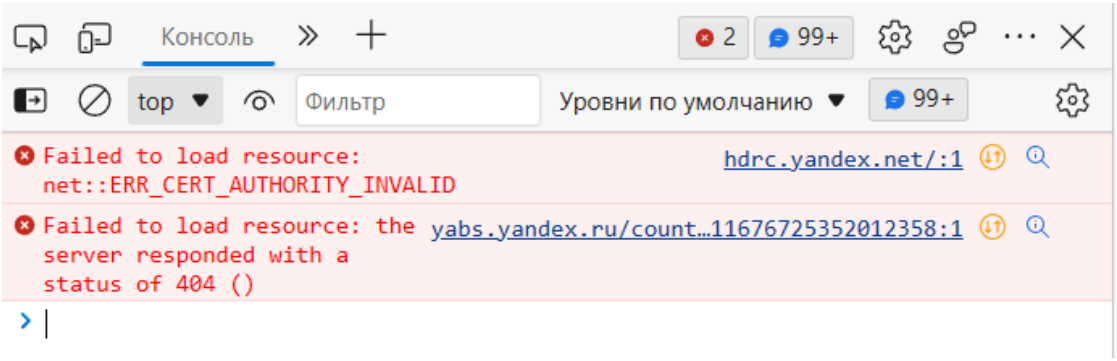
HAR日志将被收集并保存在指定文件夹中。

为了收集控制台日志：

1。按键盘上的F12，以打开开发者工具窗口。

2。点击“开启开发者工具”。

3。打开下拉菜单并选择“控制台”。



4。在窗口的任意区域右键点击，然后在上下文菜单中选择“另存为”。

5。选择保存日志的文件夹，然后点击“保存”。

控制台日志将被收集并保存在指定文件夹中。

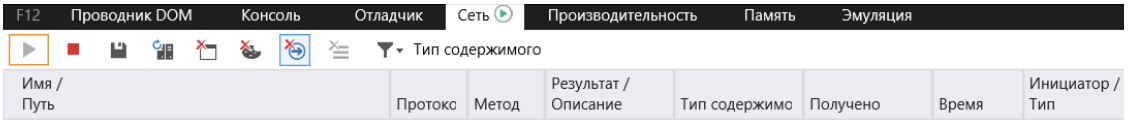
Internet Explorer

为了收集HAR日志：

1。按键盘上的F12，以打开开发者工具窗口。

2。转到“网络”选项卡。

3。点击 ，开始性能分析会话。



4。重现问题。

5。问题重现后点击 停止性能分析会话。

6。点击 ，导出HAR文件。

7。选择保存日志的文件夹，然后点击“保存”。

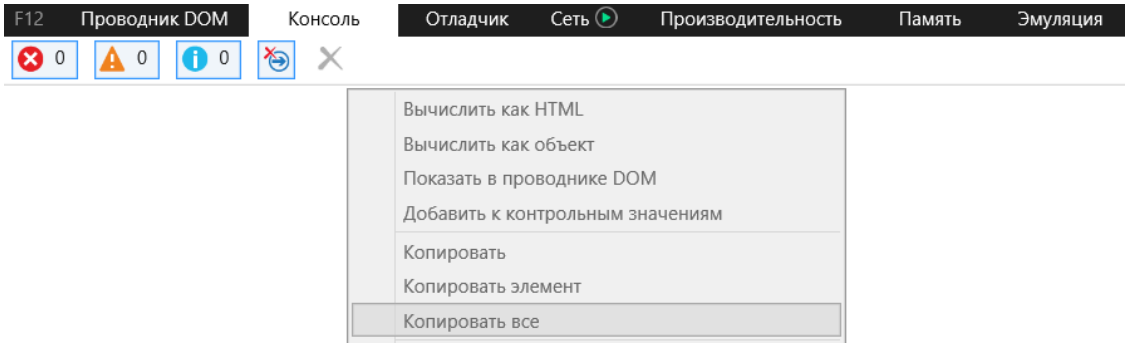
HAR日志将被收集并保存在指定文件夹中。

为了收集控制台日志：

1。按键盘上的F12，以打开开发者工具窗口。

2。转到“控制台”选项卡。

3。在窗口的任意区域右键点击，然后在上下文菜单中选择“全部复制”。



4。创建一个新的TXT格式的文本文件并打开它。

5。粘贴复制的数据并保存文件。

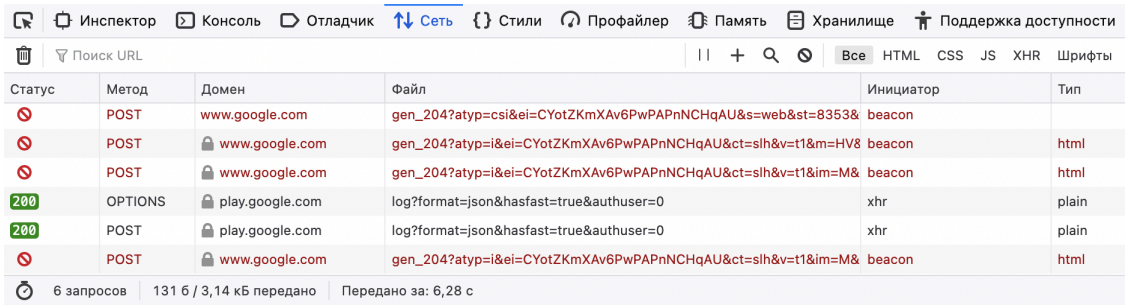
控制台日志将被收集并保存在创建的文件中。

Firefox

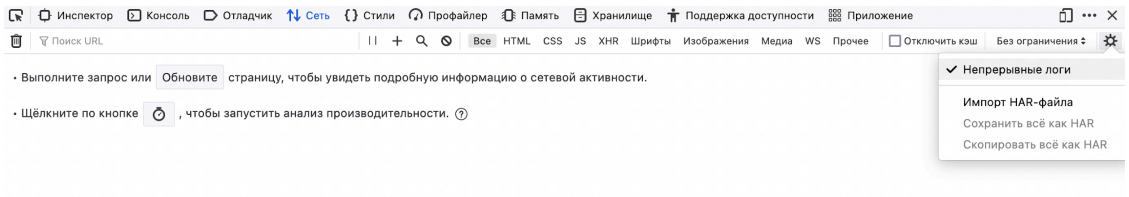
为了收集HAR日志：

1。按键盘上的F12，以打开开发者工具窗口。

2。转到“网络”选项卡。

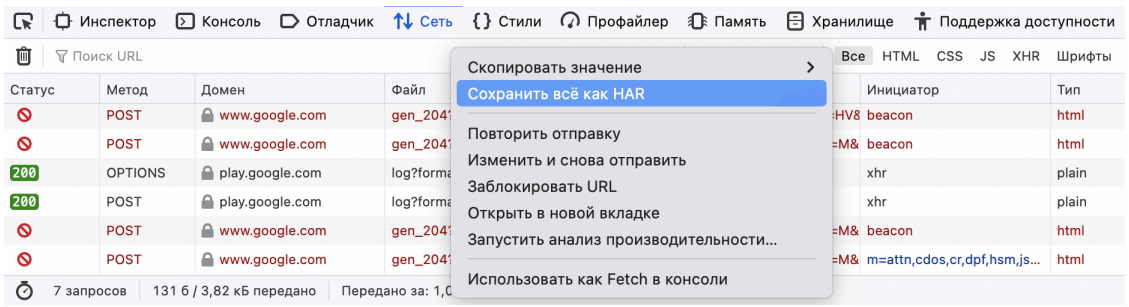


3。勾选“持续日志”。



4。重现问题。

5。在问题重现后，在网络表中右键点击并选择保存所有内容为HAR。



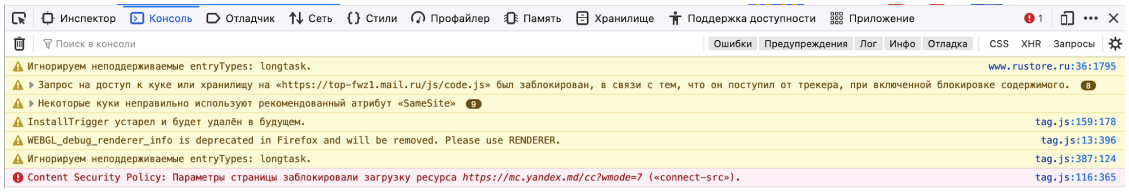
6。选择保存日志的文件夹，然后点击“保存”。

HAR日志将被收集并保存在指定文件夹中。

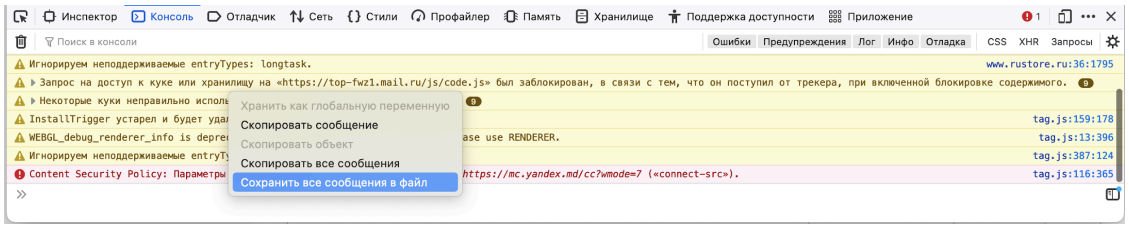
为了收集控制台日志：

1。按键盘上的F12，以打开开发者工具窗口。

2。转到“控制台”选项卡。



3。选择保存日志的文件夹，然后点击“保存”。



控制台日志将被收集并保存在指定文件夹中。

Safari

为了收集HAR日志：

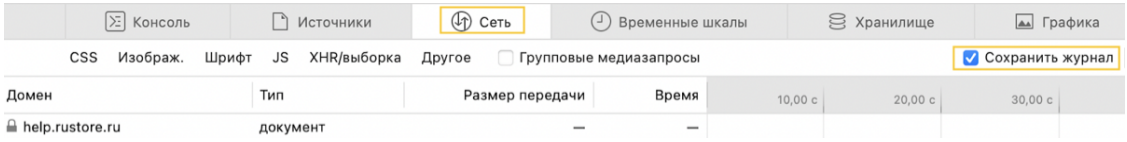
1。在菜单栏中选择“开发”。

如果菜单栏中没有“开发”，点击“Safari”，进入“设置”，点击“高级”，设置勾选框“在菜单栏显示‘开发’菜单”。

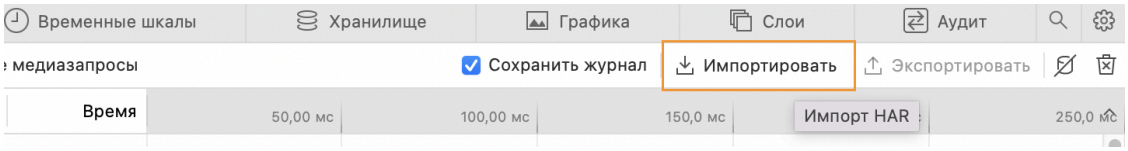
2。点击“连接网络检查器”。

3。打开“网络”选项卡。

4。在“保存日志”复选框中打勾。



5。点击“导入”。



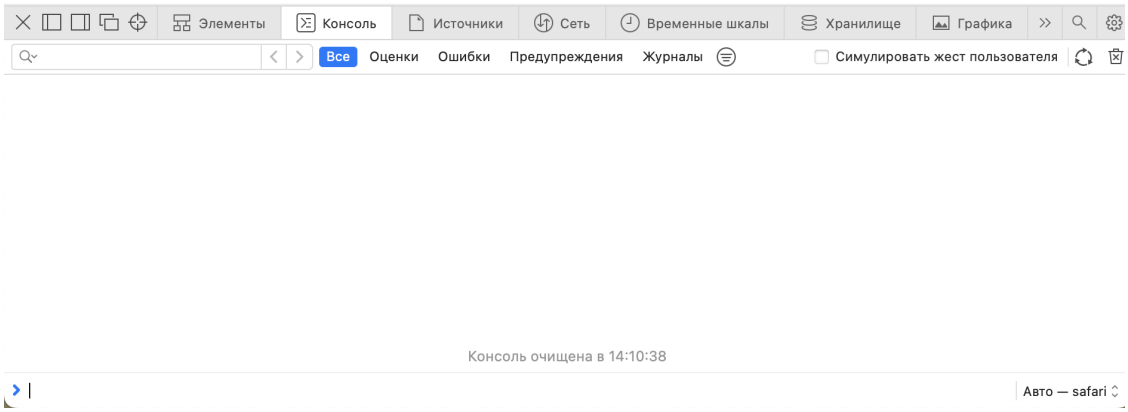
6。在弹出的窗口中选择保存日志的文件夹。

7。点击“保存”。

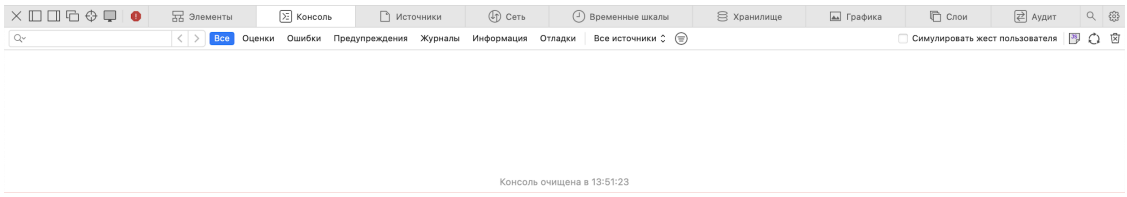
HAR日志将被收集并保存在指定文件夹中。

为了收集控制台日志：

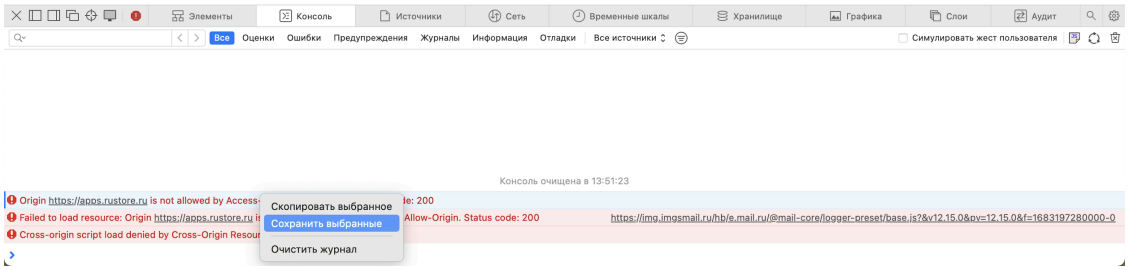
1。在菜单栏中选择“开发”。



2。点击“连接网络检查器”。



3。打开“控制台”选项卡。



4。在窗口的任意区域右键点击，并在上下文菜单中选择“保存所选”。

5。选择保存日志的文件夹，然后点击“保存”。

控制台日志将被收集并保存在指定文件夹中。

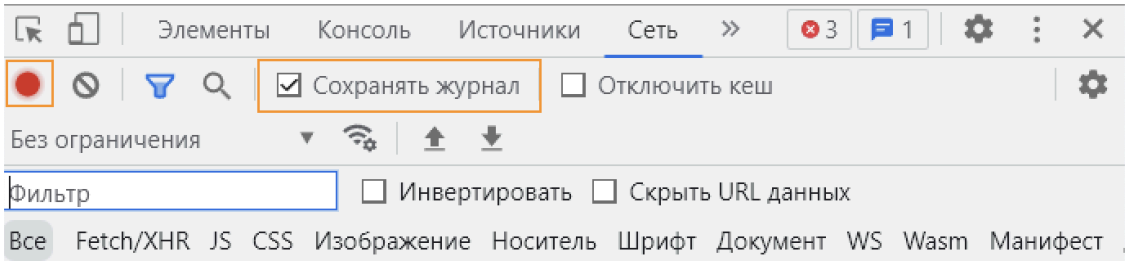
Opera

为了收集HAR日志：

1。按“Ctrl+Shift+I”或“Ctrl+Shift+J”打开开发者工具窗口。

2。在“网络”选项卡上，检查录制指示器。它应显示为红色。如果没有，请点击它开始录制。

3。勾选“保持日志”，以在切换到另一个页面时保存日志。



4。使用F5刷新页面并重现问题。

5。在日志中的任意记录上右键点击，选择“将所有内容保存为HAR”。

6。在弹出的窗口中选择保存文件夹。

7。点击“保存”。

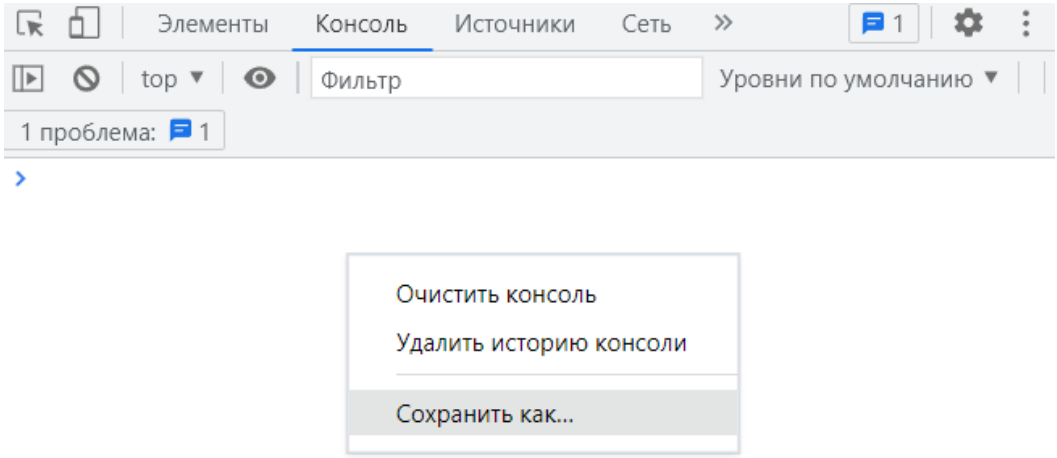
HAR日志将被收集并保存在指定文件夹中。

为了收集控制台日志：

1。按“Ctrl+Shift+I”或“Ctrl+Shift+J”，以打开开发者工具窗口。

2。转到“控制台”选项卡。

3。在窗口的任意区域右键点击，并在上下文菜单中选择“另存为”。



选择保存日志的文件夹并点击“保存”。

控制台日志将被收集并保存在指定文件夹中。

货币化

要集成货币化功能：

1. 填写货币化申请。
2. 在应用程序中配置RuStoreSDK以集成支付，或通过RuStore控制台添加付费应用程序。
3. 创建订阅或一次性购买。

非居民货币化

在RuStore注册的外国公司现在可以集成货币化功能，并从部署付费应用程序或应用内购买（包括订阅）中获得收益。

集成货币化功能可在标准附加协议的条件下获得。申请提交时的实际货币将显示在表格中。

如何创建和签署申请：

1. 登录RuStore控制台。
2. 在屏幕顶部选择“公司”选项卡。
3. 在左侧菜单中选择“货币化”。
4. 点击“填写申请”。

1. 填写弹出窗口中的信息。

主要信息

- “商业名称和法定名称”字段的信息根据公司基础文件自动加载。如果信息填写不正确，请联系支持并在更正数据后提交申请。

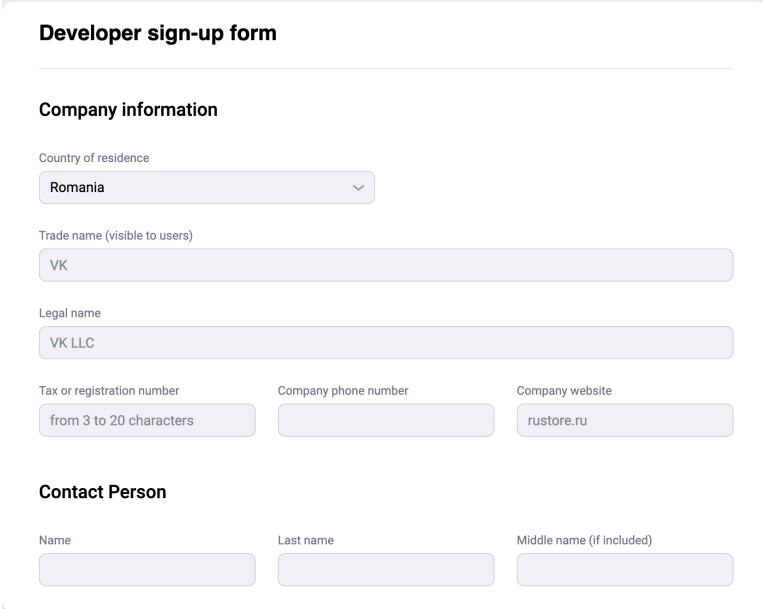
我们可能会要求您确认对所指定商业名称的权利。不要求注册权利，但在这种情况下，我们会要求您保证使用该商业名称不侵犯第三方的权利。

代表与公司所有者不同

如果RuStore中的代表与公司所有者不同，请根据代表的基础文件填写代表信息。作为此类文件，请附上任命决议、授权书等。确保此处指明的是将代表贵公司签署附加协议的人。

公司信息

- 确保“税号”和“法定地址”字段中的信息与基础文件中的数据相符。
- 如果法定地址与实际地址不一致，还需指明公司实际运营的实际地址。

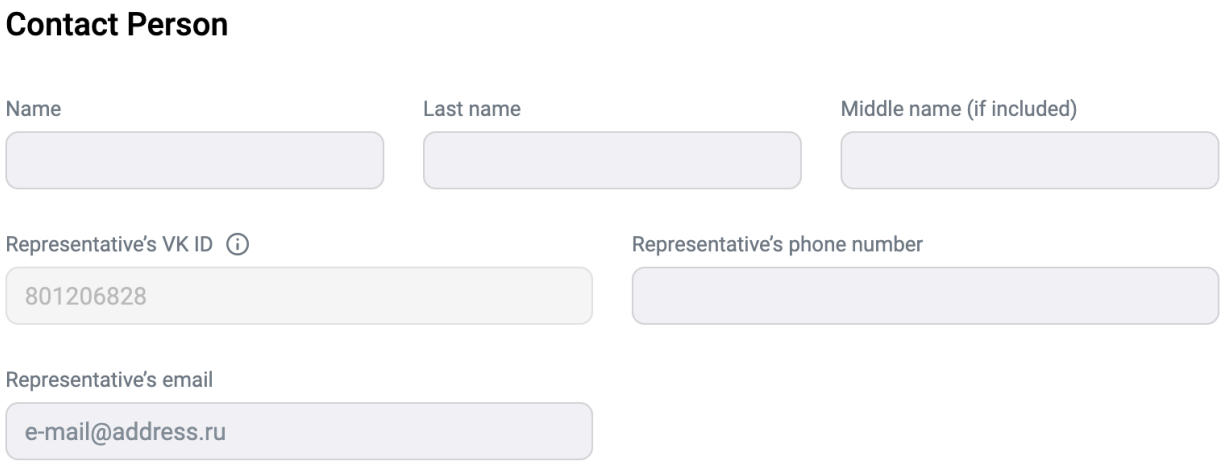


用户反馈联系方式

RuStore客户服务可以将这些联系方式传递给用户或在应用程序展示中发布。

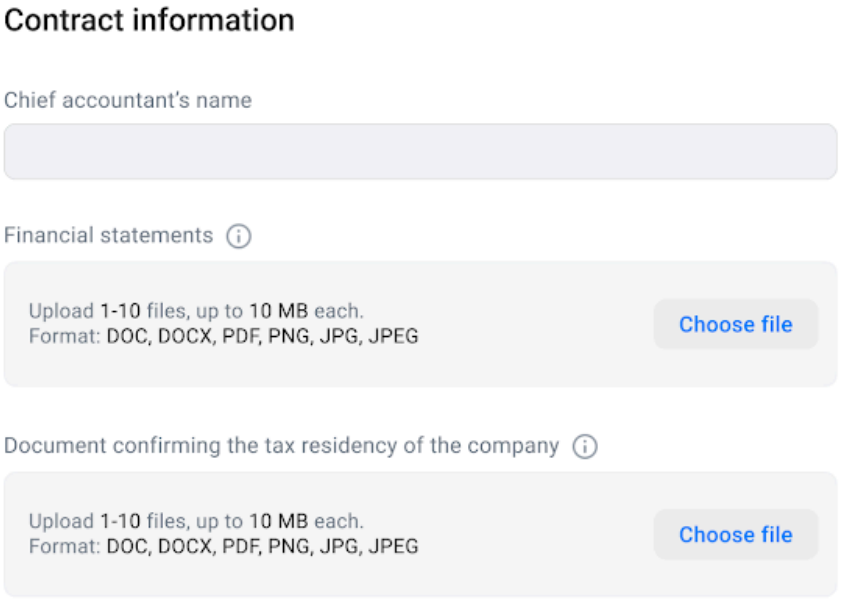
- 在“联系人”区块中，请提供RuStore用户可通过其联系您的联系方式，涉及应用程序支持问题，以及提交财务索赔（退款/取消支付请求，欺诈支付通知）。

- “电话号码”和Email信息区块用于与RuStore内部沟通，并对用户可见。在“代表号码”字段中，应填写国际电话号码。
- 您还可以指明首选的通讯方式，通过添加即时通讯软件链接、反馈表单或其他用于接收用户反馈的工具（可选）。建议您检查这些工具对俄罗斯联邦用户的可用性。



合同信息

- 公司主会计师的姓名。
- 财务报告：利润和损失报告，资产负债表。
- 税务居民证明文件：提供税务居民证明以避免双重征税。未来我们会要求您每年更新此文件。



银行详细信息

- 请提供公司在俄罗斯或国外银行的账户信息，用于进行结算。

- 如果您希望通过特定的代理银行进行结算，请在此区块中提供其详细信息。我们将向该代理银行转账您的收入，以便进一步汇给您。否则，我们将通过我们的标准代理银行，按照您上述提供的详细信息转账您的收入。

在支付时的货币兑换程序反映在附加协议中。

警告

RuStore对于付费应用程序和应用程序内购买（订阅、in-app购买）收取手续费。非居民的手续费为15%。在RuStore必须作为非居民的税务代理时，除了手续费外，还将扣留根据俄罗斯联邦法律要求支付的税额。

1. 点击“转到提交申请”。
2. 检查申请文本并点击“提交审核”。申请将提交给审核员。

重要提示

- 如有需要，您可以在任何阶段撤回申请，并进行必要的更改，然后再次提交审核。
- 如果存在不符情况，我们将通过电子邮件通知您，并要求您提交正确的文件，同时，我们保留拒绝集成和/或禁止集成货币化功能的权利。

提交货币化申请后，我们将向您的电子邮件发送附加协议。请仔细阅读协议并签署，以便集成货币化功能。

警告

请注意，附加协议必须由您在“公司代表”部分所提供信息的人签署。

集成货币化后，您可以配置付费产品并发布付费应用程序。您将每月收到关于接受的支付的报告。收入的支付将根据附加协议执行。

重要提示

支付不会比每月一次更频繁，前提是自上次支付以来已达到最低金额。

详细信息载于附加协议中。

货币化报告

关于按照RuStore关于软件产品分发协议执行任务的报告示例

每月开发者的邮箱会收到一份货币化报告。以下是带有字段描述的报告示例。

Отчет/Report

по Соглашению RuStore о распространении программных продуктов/ under the RuStore software distribution Agreement

№ ЗР/CRZ-123.45/18_07_2023 от/from 18.07.2023 г.

г. Москва/Moscow 31.01.2023

ООО "Компания", именуемое в дальнейшем **"Компания"/ LLC "Company"**, hereinafter referred to as the "**Company**",

в лице Генерального директора Иванова Ивана Ивановича, действующего на основании Устава/ represented by the General director of the managing organization LLC "Company" Ivan Ivanovich Ivanov, acting on the basis of the Charter,

предоставляет/provides

Best Games LLC

, именуемому в дальнейшем "Разработчик"/ hereinafter referred to as the "Developer",

настоящий **Отчет** о нижеследующем/this Report as follows:

1	Дата, время начала отчетного периода / Date, time of the beginning of the reporting period	01.01.2023 0:00:00
2	Дата, время конца отчетного периода/ Date, time of the end of the reporting period	31.01.2023 23:59:59
3	Нераспределенные платежи на начало отчетного периода, руб./ Unallocated payments at the beginning of the reporting period, rub.	0,00
4	Общая сумма принятых платежей от Пользователей за отчетный период (включая возвраты) **в том числе НДС**, руб./ The total amount of payments received from Users for the reporting period (including refunds and unallocated payments at the end of the reporting period), **including VAT,** rub.	0,00
5	Общая сумма принятых платежей от Пользователей за отчетный период (включая возвраты) **за вычетом НДС,** руб./ The total amount of payments received from Users for the reporting period (including refunds and unallocated payments at the end of the reporting period), net of VAT,rub.	0,00
6	Сумма вознаграждения Компании за отчетный период, руб./ The amount of the Company's remuneration for the reporting period, rub.	0,00
7	Общая сумма возвратов Пользователям за отчетный период, руб./ Total amount of refunds to Users for the reporting period, rub.	0,00
8	Нераспределенные платежи на конец отчетного периода, руб./ Unallocated payments at the end of the reporting period, rub.	0,00
9	Задолженность перед Компанией на конец отчетного периода, руб./ Debt to the Company at the end of the reporting period, rub.	0,00
10	Сумма причитающаяся Разработчику за отчетный период, рубли РФ (Сумма к перечислению)/ Amount due to the Developer for the reporting period, Russian rubles (Amount to transfer)	0,00
11	Сумма причитающаяся Разработчику за отчетный период, валюта (Сумма к перечислению) / Amount due to the Developer for the reporting period, currency (Amount to transfer)	0,00
12	Курс валюты на конец отчетного периода (в соответствии с Дополнительным соглашением)/ Exchange rate at the end of the reporting period (in accordance with Additional Agreement)	0,00

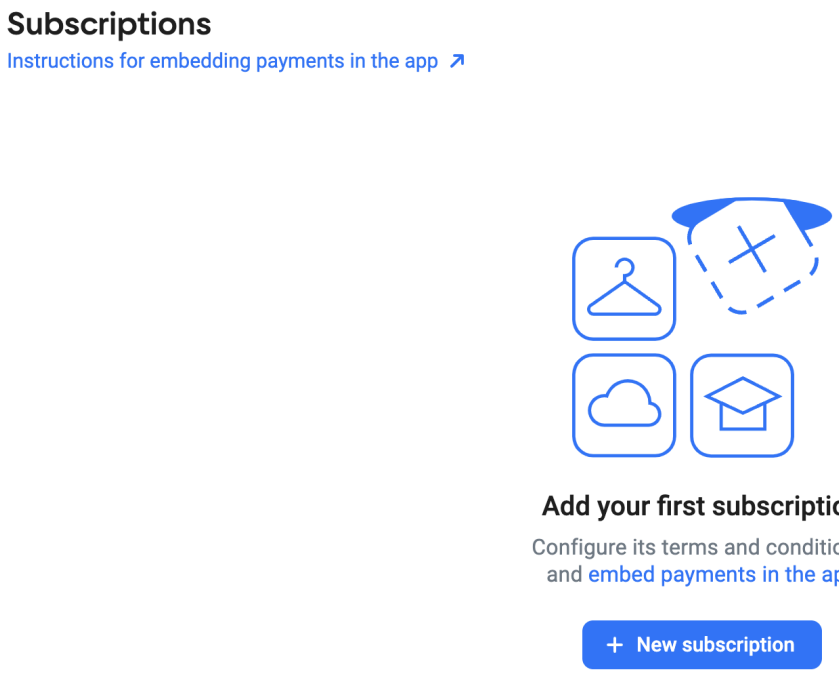
Разработчик должен ознакомиться с Отчетом в течение 3 (трех) рабочих дней с момента получения Отчета. В случае наличия каких-либо возражений по данным Отчёта, Разработчик должен в тот же срок направить Компании подписанный мотивированный отказ принимать Отчет. / The developer must familiarize himself with the Report within 3 (three) working days from the date of receipt of the Report. If there are any objections to the Report data, the Developer must send the Company a signed motivated refusal to accept the Report at the same time.

字段	描述
报告期开始的日期和时间。	报告期开始的日期和时间。
报告期结束的日期和时间。	报告期结束的日期和时间。
报告期开始时未分配的支付，卢布。	您应用的用户支付的金额，但在上一个报告期结束时未纳入结算。金额以卢布表示。
总接受用户在报告期内支付的金额（包括退款），**包括增值税**，卢布。	这是您的应用用户在报告期内支付的金额（税前）。金额以卢布表示。
总接受用户在报告期内支付的金额（包括退款），**扣除增值税后**，卢布。	这是您的应用用户在报告期内支付的金额（扣除我们作为税务代理按照俄罗斯联邦法律规定应扣留的增值税后的金额）。金额以卢布表示。
公司在报告期内的报酬总额，卢布。	公司扣留的佣金总额。金额以卢布表示。
用户在报告期内的总退款金额，卢布。	用户接收的退款总额。包括技术性退款（在支付程序开始后取消购买）。金额以卢布表示。
报告期末未分配的支付，卢布。	您的应用程序用户支付的，但在报告期末未计入结算的总金额。下个月，这笔金额将反映在“报告期初未分配支付，卢布”一栏。金额以卢布表示。
当前报告期内对公司的债务，卢布。	如果在任何报告期的一天内，退款金额超过了您的应用程序中的购买金额，就可能产生债务。在这种情况下，我们等待来自用户新购买的资金到达，这些资金将自动抵消产生的债务。如果在报告期结束时，债务未被后续购买完全清偿，我们将在报告栏目中反映这一信息。金额以卢布表示。
报告期内应付给开发者的金额，俄罗斯卢布（应转账金额）	您的收入总额，计算支付给您的金额（用户支付金额减去退款和佣金）。支付可能会根据开发者与RuStore之间协议中规定的限制进行。金额以卢布表示。
报告期内应支付给开发者的金额，货币（应转账金额）。	您的收入总额，用于计算报告期内支付的金额（用户支付金额减去退款和佣金）。支付可能会考虑开发者与RuStore之间协议中规定的限制。金额以协议货币表示。
报告期末的汇率（根据附加协议）。	用于当前报告结算的货币汇率的参考数据。汇率的应用方式在开发者与RuStore之间的协议中有所体现。

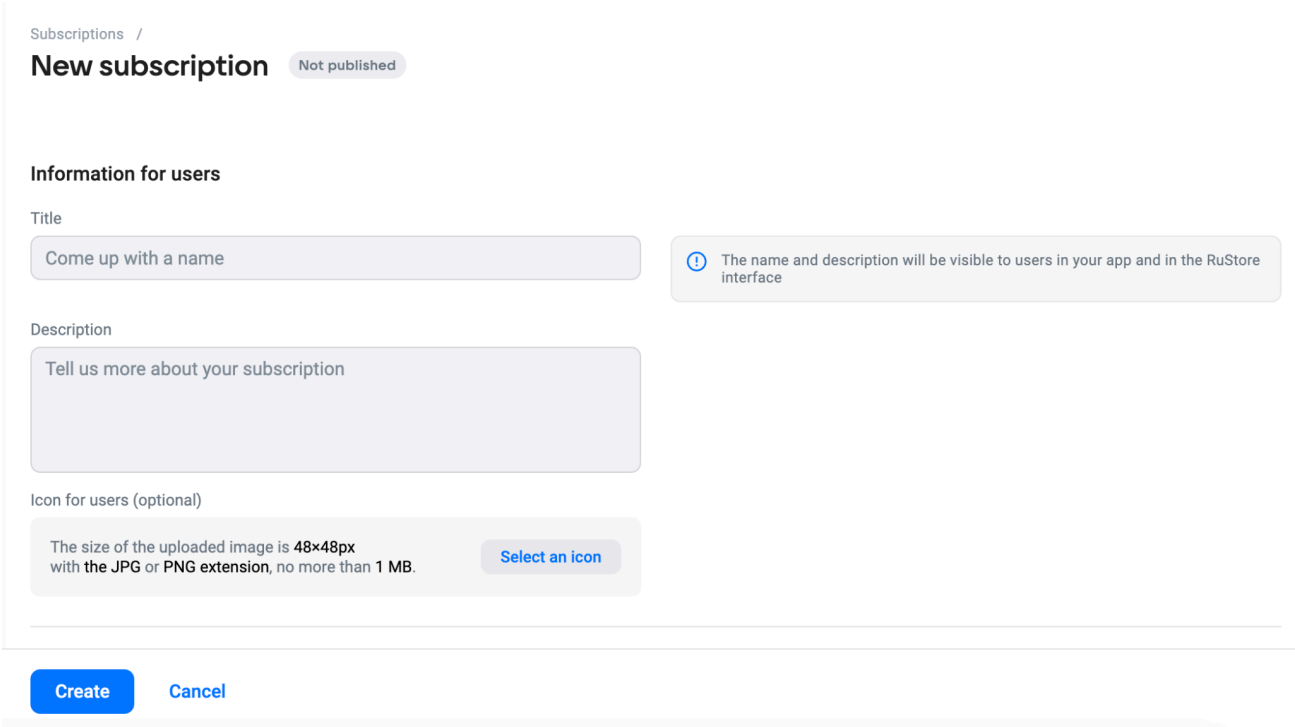
如何为应用程序创建订阅

在RuStore控制台系统中创建订阅，让用户能够在RuStore商店购买应用程序的订阅：

1. 打开RuStore 控制台系统。
2. 在屏幕顶部转到“应用程序”选项卡。
3. 在左侧菜单的“货币化”部分，选择“订阅”。
4. 要创建订阅，请点击“新建订阅”。如果之前未创建订阅，按钮将出现在屏幕中央。如果已有订阅，按钮将出现在右上角。



5. 提供订阅的用户和技术信息，以及额外设置。



6. 点击屏幕左下角的“创建”。
7. 点击屏幕左下角的“发布”。如果需要在此步骤编辑订阅信息，请点击“编辑”。
8. 在弹出窗口中点击“发布”。

定期支付将在订阅结束前的8小时内扣除。例如，如果订阅在16:00开始，那么后续的定期支付将在08:00至16:00之间扣除。扣费条件应由开发者的用户文档规定。

附加设置

您可以为订阅设置一个免费期和一个降价的起始期。例如，第一个月免费，第二个月收费50卢布，之后进入标准期，每个月收费100卢布。

用户只能使用一次免费期和起始期。如果他取消订阅然后再次购买，订阅将直接从标准期开始。

免费期

如果需要在一定期限内提供用户免费访问应用程序，请开启此功能。例如，用于市场推广。

当免费期结束时，如果设置了起始期，订阅将转入起始期，否则直接进入标准期并开始扣费。

警告

在免费期间，用户仅可使用新卡、之前保存的卡、SberID或SberPay钱包中的卡进行支付。不可用的支付方式（通过电话号码和VK ID钱包中的卡）将不会向用户提供。我们建议在计划包含免费期的订阅的市场营销活动时考虑这一点。

起始期

如需提供特别优惠条件下的应用程序访问权限，请启用此功能。

警告

指定金额在整个起始期内只扣除一次。例如，如果设定期限为“三个月”和费用为“一卢布”，则在三个月内只会扣除一次一卢布的费用。

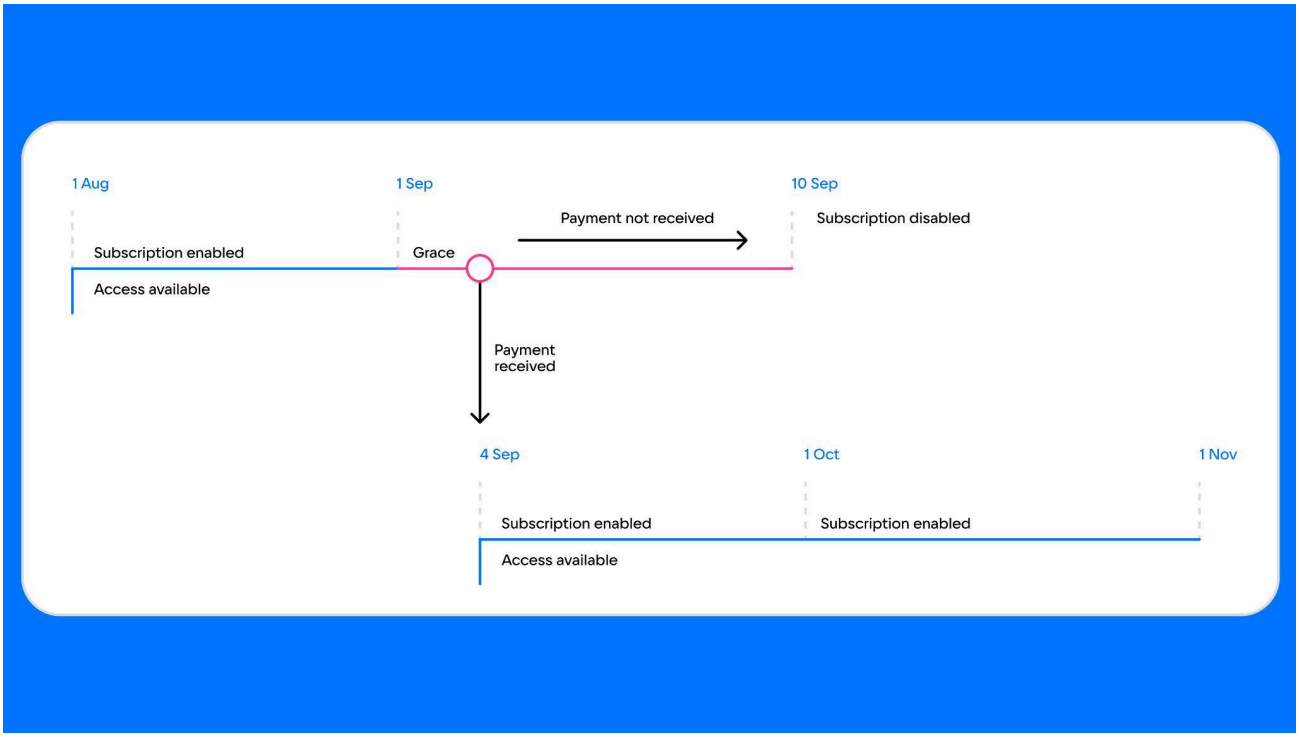
当起始期结束后，订阅将进入标准期，并开始扣费。

宽限期

如果需要在订阅支付失败时保持用户对内容的访问，请设定宽限期。

在宽限期内，每天都会尝试重复进行支付。如果到宽限期结束时仍未收到支付，对内容的访问将被取消。如果在宽限期内收到支付，扣费尝试将停止，用户继续保有对内容的访问。此时，下一次订阅的预定支付日期保持不变。

宽限期的持续时间可以等于或小于订阅的标准期。宽限期的工作原理如下图所示。



图中的虚线表示预定支付日期。所示数值仅为示例，实际日期在每个具体案例中可能有所不同。

暂停期

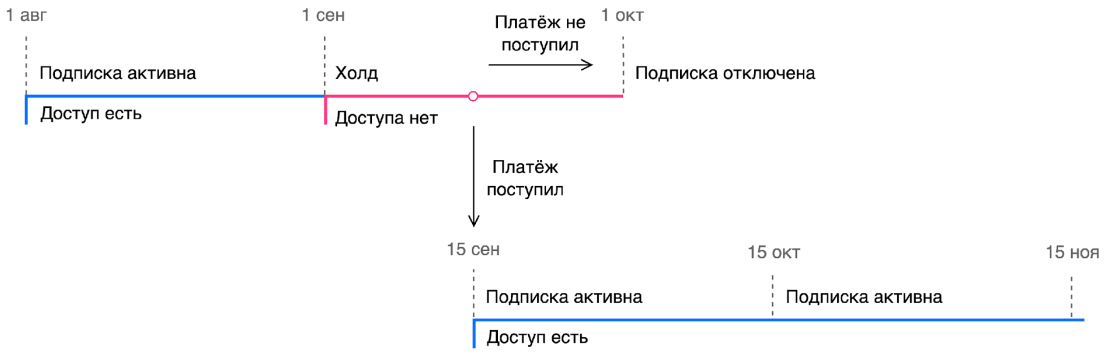
如果需要在订阅支付尝试失败时暂停用户对内容的访问，请设定暂停期。

在暂停期内，每天都会尝试重复进行支付。如果在暂停期内收到支付，扣费尝试将停止。扣款日期成为标准期的新开始日期。订阅的下一个支付日期将根据暂停期内支付日期的变动而调整。

如果在暂停期结束前未收到支付，订阅将被取消。暂停期的工作原理如下图所示。

如果您同时设定了宽限期，那么首先应用宽限期，然后是暂停期，如下图所示。

图中的虚线表示预定支付日期。所示数值仅为示例，实际日期在每个具体案例中可能有所不同。



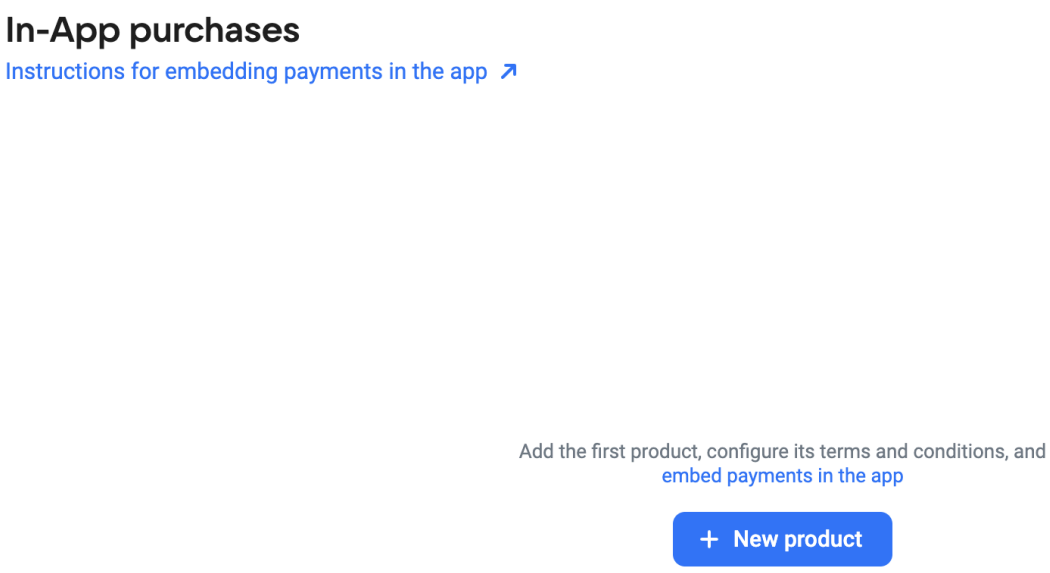
订阅价格变更

您可以提高或降低订阅的价格：

- 在降低价格时，新价格将适用于所有订阅。这意味着已购买订阅的所有用户的价格都会更新。
- 提高价格只会影响新订阅的购买。对于之前已购买订阅的用户，价格保持不变

如何在应用程序中创建付费商品

1. 打开RuStore 控制台系统。
2. 在屏幕顶部转到“应用程序”选项卡。
3. 在左侧菜单的“货币化”部分，选择“一次性购买”。
4. 点击屏幕中央的“新建商品”。
5. 提供商品的用户和技术信息。
6. 点击屏幕左下角的“创建”。
7. 点击屏幕左下角的“发布”。如果需要在此步骤编辑商品信息，请点击“编辑”。
8. 在弹出窗口中点击“发布”。



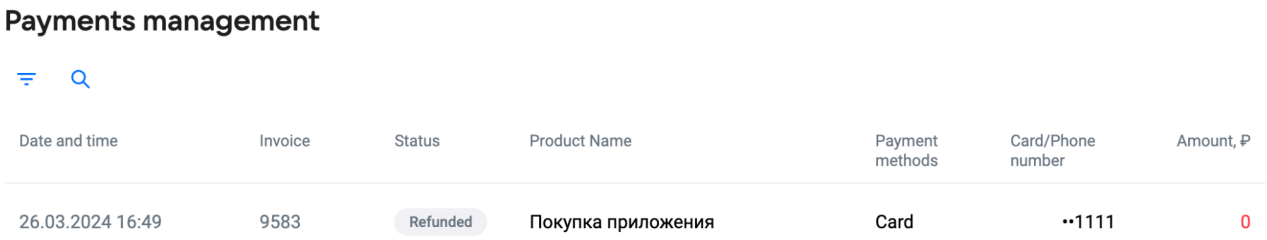
支付管理

在RuStore控制台中，您可以查看已启用货币化功能的每个应用程序的支付情况。

如何查看支付信息

1. 打开RuStore控制台。

2. 在顶部菜单中选择“应用程序”。
3. **转到“货币化 → 支付管理”选项卡。**



如果应用程序有相关购买活动，将显示一个包含购买数据的表格。在表格中，您可以查看：

- 购买的日期和时间；
- 账单号和订单号；
- 进行支付的卡号或电话号码；
- 购买金额；
- 以及账单状态。

订单号仅适用于订阅支付。这个号码在同一订阅的所有支付中都是相同的。

您也可以点击您感兴趣的支付，查看有关它的详细信息。

搜索支付

要搜索您感兴趣的支付并查看其信息，请执行以下操作：

1. 打开RuStore控制台。
2. 在顶部菜单中选择“应用程序”。
3. **转到“货币化 → 支付管理”选项卡。**
4. 在搜索框中输入账单号。将显示有关支付金额和状态的信息。选择您想要进行搜索的标识符 — 账单号或订单号。然后相应地在搜索框中输入账单号或订单号。在搜索结果中，您将看到您应用程序中符合搜索条件的所有支付。

支付可能处于以下几种状态之一：

处理中：购买者卡上的资金被暂时保留。购买正在等待消耗。

已取消：购买者卡上的资金被解锁。购买已被取消。

已支付：交易已确认，资金从购买者账户中扣除。

已退款：资金已退还至购买者账户。退款到账可能需要长达5个工作日。

退款

要向购买者退还支付，请执行以下操作：

1. 打开RuStore控制台。
2. 在顶部菜单中选择“应用程序”。
3. **转到“货币化 → 支付管理”选项卡。**
4. 选择需要进行退款的账单。
5. 在弹出窗口中点击“退还支付”，进入退款处理页面。

6。阅读新窗口中的信息，然后再次选择“退还支付”。

退款操作不会取消用户的商品或订阅。

完成退款操作后，购买的状态将变为“已退款”。之前支付的资金将被退还至购买者的账户。

测试支付

在移动应用程序中测试订阅和付费商品的支付。测试模式下的购买过程模拟了常规支付，但使用测试银行卡进行。

要测试支付，应用程序需要通过审核。至少是其第一个版本。

功能

“测试支付”功能提供以下功能：

- 通过RuStore控制台管理测试模式。
- 在您的应用程序中测试购买支付场景，这在使用特殊测试卡的隔离测试环境中进行，不影响应用程序的工作版本。
- 在RuStore控制台查看测试支付的历史记录。

技术特性

- 可以在应用程序的版本向所有用户发布之前进行测试。
- 在测试模式中，可以购买那些已添加到应用程序工作版本中的同样商品和订阅。不必发布这些商品和订阅。
- 可以多次测试购买非消耗性商品。例如，访问应用程序的完整版本或游戏中的皮肤。如果通过RuStore控制台退款，之后可以在应用程序中再次进行购买。
- 在测试模式下，订阅使用特殊参数。这些参数不依赖于您为应用程序工作版本选择的参数，并且不对它们产生影响。这些技术特性旨在简化和加速测试。测试模式中所有订阅的参数在下表中是相同的。

参数	含义
订阅的最大扣款次数。达到指定的扣款次数后，订阅将自动关闭	12
订阅扣款之间的时间	10分钟
免费和起始期间重置的频率（购买带有免费和起始期的订阅条件）。重置后，您将能够再次免费或以起始价格购买订阅。这样，您可以多次测试按特殊条件订阅的支付	每3小时一次

限制

- 应用程序的所有者是唯一可以测试支付的用户。只有他才能访问“测试支付”部分。我们正在努力添加其他测试人员的可能性。
- 在RuStore商店中测试购买付费应用程序的功能不可用。
- 无法通过public API获取有关测试支付和订阅的信息。
- RuStore移动应用程序中不显示测试购买和测试订阅的历史记录。这意味着无法取消和恢复测试订阅。
- 测试购买的支付只能通过测试银行卡进行。

启用测试模式

在开始测试支付之前

- 在被测试的应用程序中集成支持支付测试功能的SDK版本 — RuStore SDK billingclient 3.1.0或更高版本。
- 确保计划在其上测试您的应用程序的Android设备安装了1.29版本或更高版本的RuStore。
- 确保您的公司未被封锁，已集成货币化，并且应用程序内能够进行购买。您可以在RuStore控制台设置订阅和付费商品，但不需要向用户发布它们。

- 如何为应用程序创建订阅

启用测试模式

在审核完成并且新版本应用程序发布之前，您可以进行支付测试。您也可以对已发布的应用程序进行购买测试。

测试模式仅为测试者和所选应用程序启用。在其他应用程序中，同一用户将进行实际购买。此外，该应用程序中的所有其他用户也将进行实际购买。

默认情况下，测试模式是关闭的。

要启用测试模式：

1. 打开RuStore 控制台。
2. 在顶部菜单中选择“应用程序”。
3. 转到“货币化 — 测试支付”选项卡。
4. 切换到“测试者”选项卡。如果选项卡上显示“添加自己到测试者列表”，点击“成为测试者”。
5. 激活“测试模式”列中的开关。

启用测试模式后，测试者将在您的应用程序中进行测试购买，并且他将能够购买尚未发布的商品。

测试支付

测试支付

在切换到测试模式后要应用更改，您需要等待几分钟，或者如果您的应用程序已经打开了，那么需要重新启动应用程序。

为了进行支付测试：

1. 启用测试模式。
2. 如果您还未在进行测试的设备上登录RuStore，请使用您在RuStore 控制台注册为应用程序所有者的VK ID进行登录。

- 如何在RuStore 控制台找到VK ID? - 如何用VK ID账户登录RuStore移动应用程序?

3. 在您的设备上打开您的应用程序，或者如果还未安装，请先安装。

4. 在应用程序内进行购买。例如，订阅一个服务。在支付时会出现一个标有“测试购买”的支付窗口。

如果没有出现“测试购买”的标记，那么您不是在测试环境中购买，而是在应用程序的工作版本中购买。确保测试模式已启用，并且您已在RuStore移动应用程序中使用正确的VK ID登录。

1. 添加一张测试银行卡并选择它进行支付。添加后，此卡将保存在测试配置文件中，以后使用这张卡的购买将可以一键完成。当您的账户处于测试模式时，只使用测试卡。如果尝试使用真实银行卡进行支付，将会出现错误。

- 使用不同的测试卡来检验成功和失败的支付场景。例如，模拟支付金额超过银行卡限额时的错误。
- 如果需要测试宽限期和暂停期，请使用余额为1戈比的测试卡。使用这张卡可以购买带有免费期的订阅。免费期结束后，将有一次失败的扣款尝试，然后订阅会立即切换到宽限期或暂停期，如果这些期间为该订阅设置的话。关于这些期间的详细信息，请参见文章《如何为应用程序创建订阅》。

1. 点击“支付”。
2. 检查支付是否成功，以及您的应用程序是否处理了购买。

如果支付成功但无法访问购买内容，请检查您应用程序中的实现。如果在测试场景中遇到其他问题，请联系技术支持 support@rustore.ru。

在测试模式开启时，获取购买信息的方法和获取购买列表的方法将只返回该用户的测试购买。

查看测试支付

要查看测试支付的历史记录：

1. 打开RuStore 控制台。
2. 在顶部菜单中选择“应用程序”，并打开进行了测试购买的应用程序。
3. 转到“货币化 — 测试支付”选项卡。

在“支付历史”选项卡上，显示了在测试模式下完成的所有支付。功能上，这一部分与显示应用程序工作版本中用户完成的购买的“支付管理”相似。

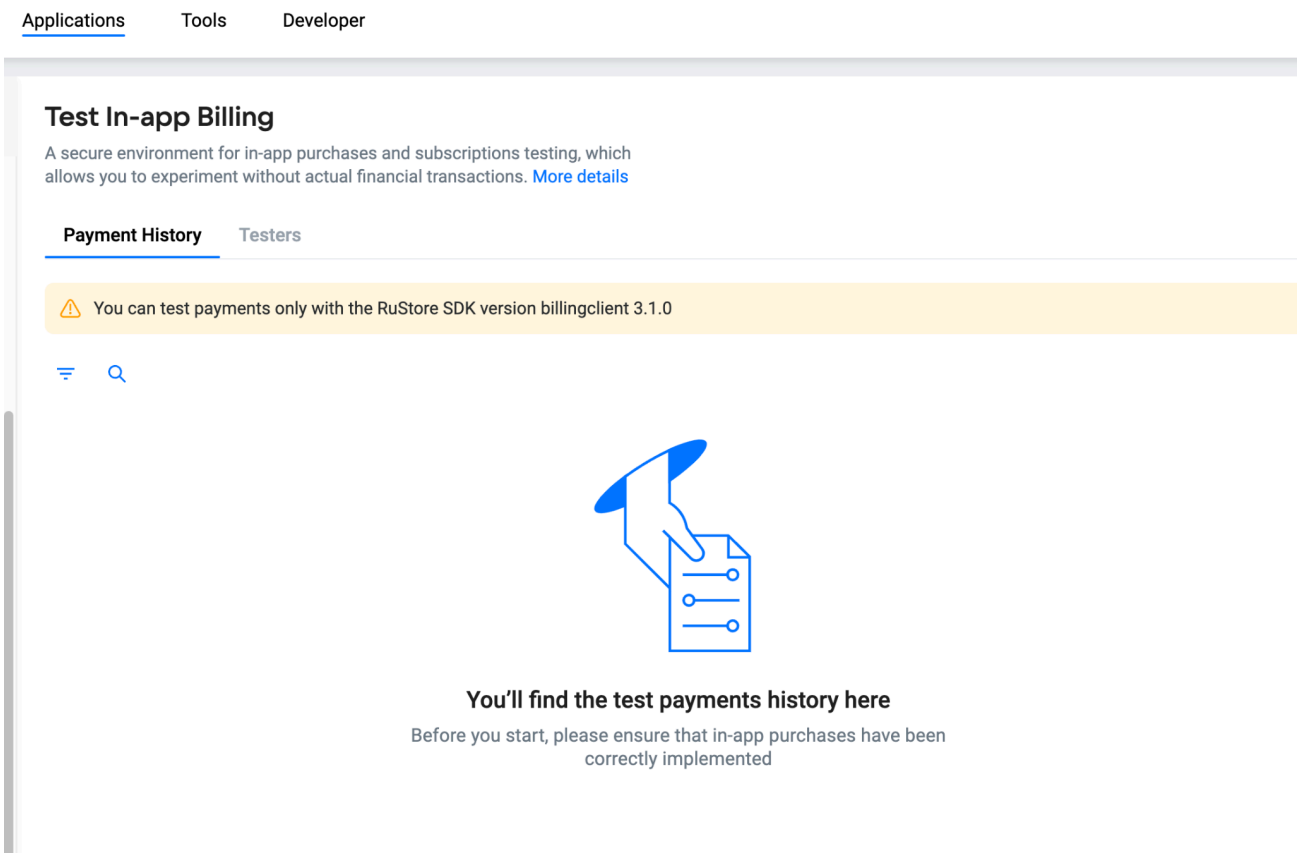
在“支付历史”选项卡上，您可以：

- 根据账单号和订单号搜索测试支付。
- 点击某个支付，查看其详细信息。
- 执行退款操作，以重新测试非消耗性商品的购买。为此，请点击支付，然后点击“退还支付”按钮。

关闭测试模式

在完成支付测试后，您可以关闭测试模式。这不是必须的，您可以根据需要保持在测试模式下。

1. 打开RuStore 控制台。
2. 在顶部菜单中选择“应用程序”。
3. 转到“货币化 — 测试支付”选项卡。
4. 切换到“测试者”选项卡。
5. 在“测试模式”列中关闭开关。



关闭测试模式时：

- 测试者将在应用程序中进行真实的购买。

- 在测试模式下完成的所有支付都会保存在RuStore 控制台的“支付历史”选项卡中。
- 购买的付费商品将在下次启用测试模式时对账户仍然可用。
- 测试订阅将在12次扣款后自动取消。

下一步

在您测试了支付并确保它们的功能性后，您可以在应用程序中发布订阅和付费商品，然后发布应用程序的新版本。

测试银行卡

您可以使用以下测试卡进行购买测试。

总是成功的支付

卡号	4111 1111 1111 1111
ECI	05
CVC\CVV	123
有效期	2024/12
3-D Secure确认代码	12345678

卡号	5100 0000 0000 0008
ECI	05
CVC\CVV	123
有效期	2024/12
3-D Secure确认代码	12345678

向购买者提供选择3-D Secure验证结果的选项。

卡号	2201 3820 0000 0013
ECI	02
CVC\CVV	123
有效期	2024/12

返回错误的卡

在三次支付尝试失败后，将对支付进行退款，并在RuStore的支付界面上显示支付错误。

卡号	5100 0000 0000 0180
ECI	00
CVC\CVV	123
有效期	2024/12

卡号	4444 4444 4444 4422
CVC\CVV	123
有效期	2024/12

卡号	4444 4444 1111 1111
CVC\CVV	123
有效期	2024/12

余额为1戈比的卡

卡号	4954 8493 9714 9582
CVC\CVV	123
有效期	2024/12
3-D Secure确认代码	12345678

广告和推广

添加“从RuStore下载”按钮

为了让用户了解您的应用程序，请使用“从RuStore下载”按钮。

双色版本的标志





下载：SVG，PNG

下载： SVG，PNG

单色版本的标志





下载： SVG，PNG

下载： SVG，PNG

使用示例：

Минимальный размер

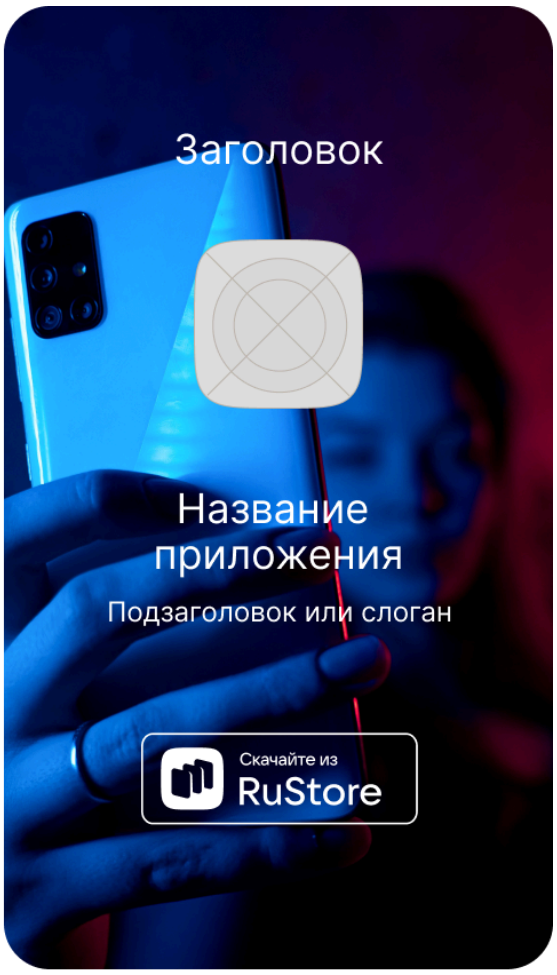
Скачайте из RuStore

10 mm 52 px

Скачайте из RuStore

Название приложения

Скачайте из RuStore



Скачайте из RuStore

Минимальный размер квадратного логотипа 60x60 px

RuStore RuStore RuStore RuStore

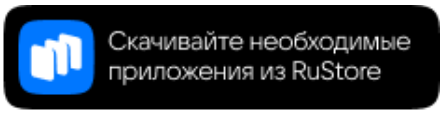
Логотипы с местом для системного шрифта

下载 SVG, PNG 下载 SVG, PNG

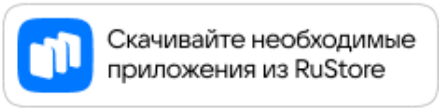
喜爱/必需的应用程序

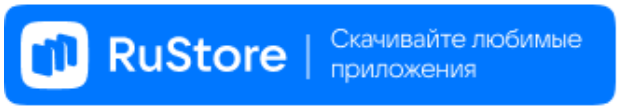
下载：SVG, PNG

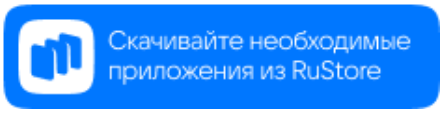








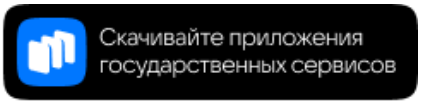




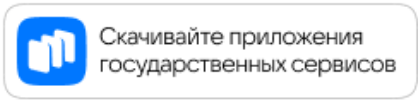
政府应用程序

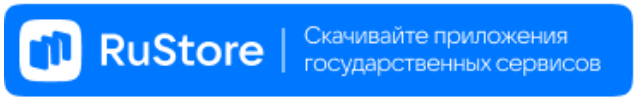
下载：SVG, PNG

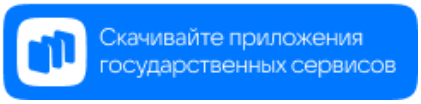












为了在网站上安装标志：

1. 下载“从RuStore下载”的标志。
2. 将其上传到您的服务器。
3. 在RuStore网站上找到您的应用程序并复制链接中的bundle_id。

链接示例：https://trk.mail.ru/c/me10h4?bundle_id=com.vkontakte.android

1. 插入到网站的代码：

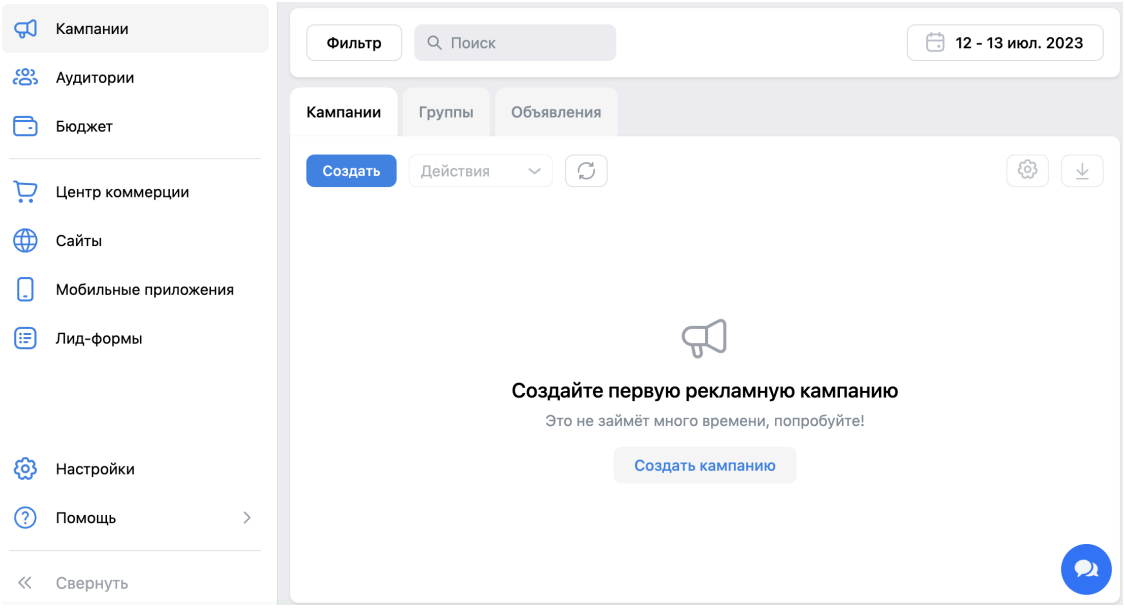
推荐的图片大小为188px x 63px。

VK广告在RuStore

现在开发者们有机会通过广告在RuStore推广应用程序。

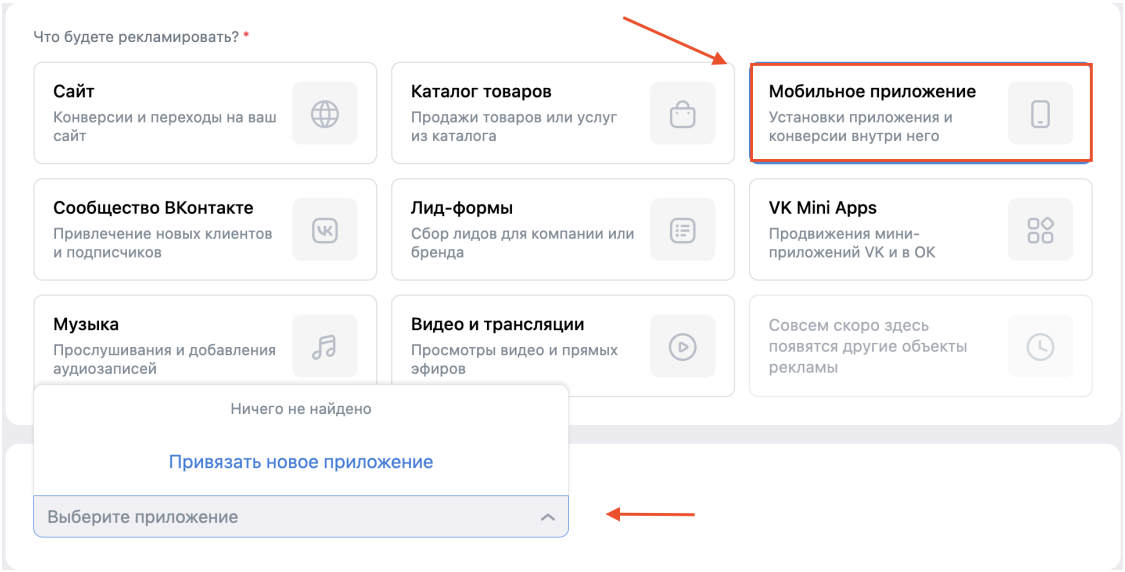
如何通过 VK 广告设置广告活动

1. 在浏览器中打开 VK 广告平台并登录。
2. 转到“活动”部分。
3. 点击“创建活动”。



1. 点击“移动应用程序”图标。从列表中选择要推广的应用程序或通过 VK 广告绑定新应用。

在首次连接时，需要添加应用程序并设置与跟踪器的集成。有关详细信息，请参阅下面的“通过 VK 广告添加应用程序”部分。

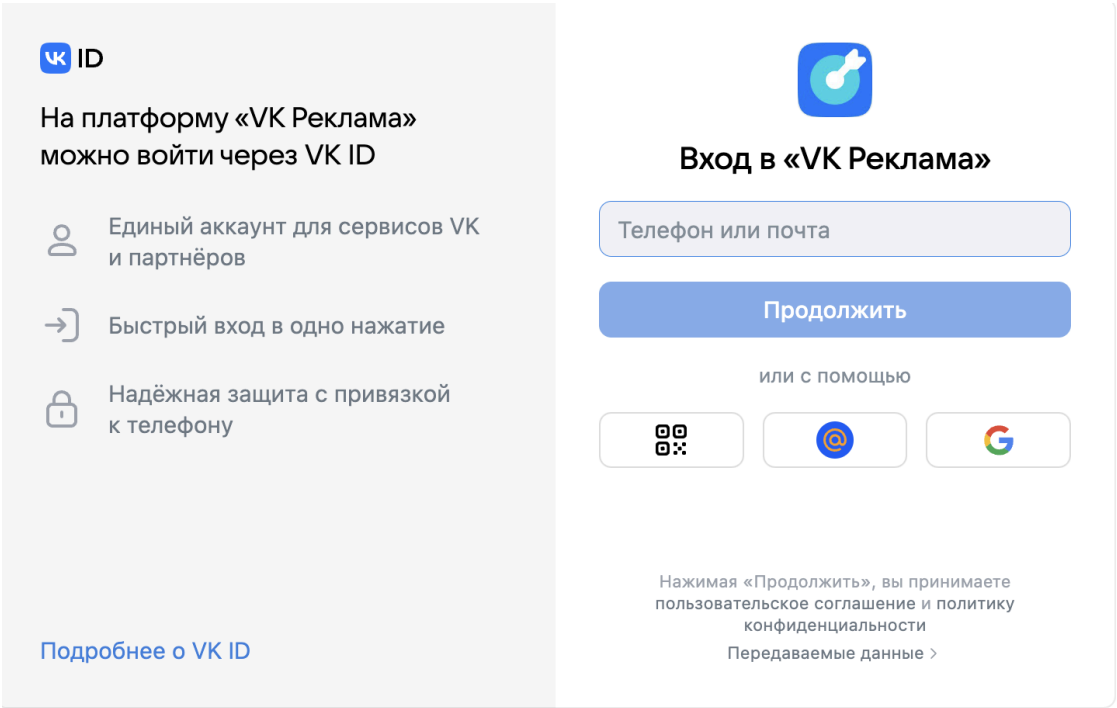


1. 设置广告活动。

VK 广告提供了许多工具和技术，您可以使用它们来精确地设置广告。有关设置广告活动的更详细指南以及许多其他问题的答案可以在这里找到。

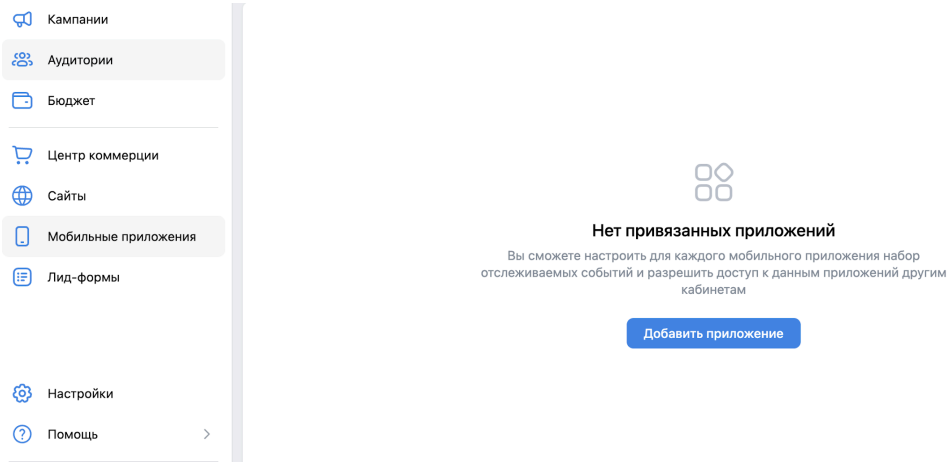
如何通过 VK 广告添加应用程序

1. 在浏览器中打开 VK 广告系统。



1. 选择一个方便的授权方法或创建一个新帐户。

2. 转到“移动应用程序”部分并点击“添加应用程序”。



1. 将打开一个添加应用程序的表单。输入 RuStore 中的应用程序链接并点击“添加”。

您可以通过访问 RuStore 的网页版本来查看您的应用程序的链接。

应用程序的状态将更改为“检查访问”。

1. 开始设置与追踪器的集成。

RuStore 还允许开发者通过 VK 广告在平台上直接链接而无需使用第三方跟踪系统来启动广告活动。有关从 RuStore 启动活动的说明，请参阅指南。

如何为用户显示广告

广告应用程序在“精选”部分可用。

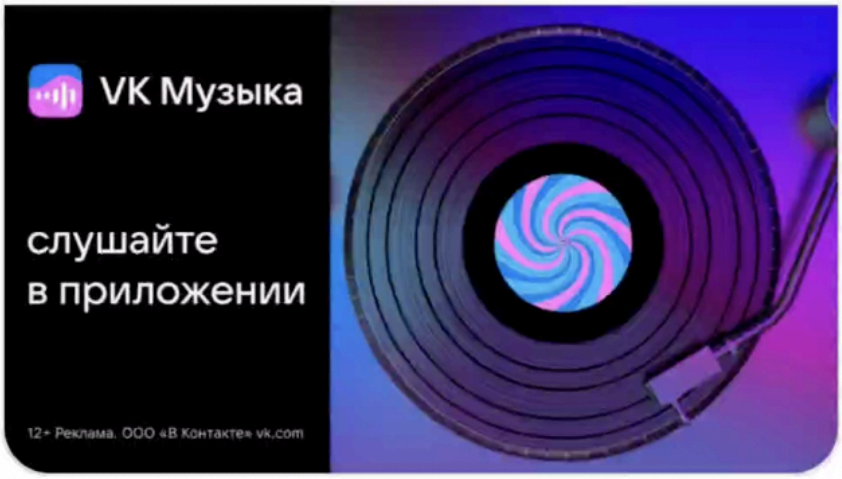
广告显示给授权和未授权的用户。

在RuStore的“精选”部分显示广告：

当进入“精选”部分时，会显示广告横幅。用户可以点击横幅以转到应用程序页面。

Интересное

VK Музыка
Реклама • 12+



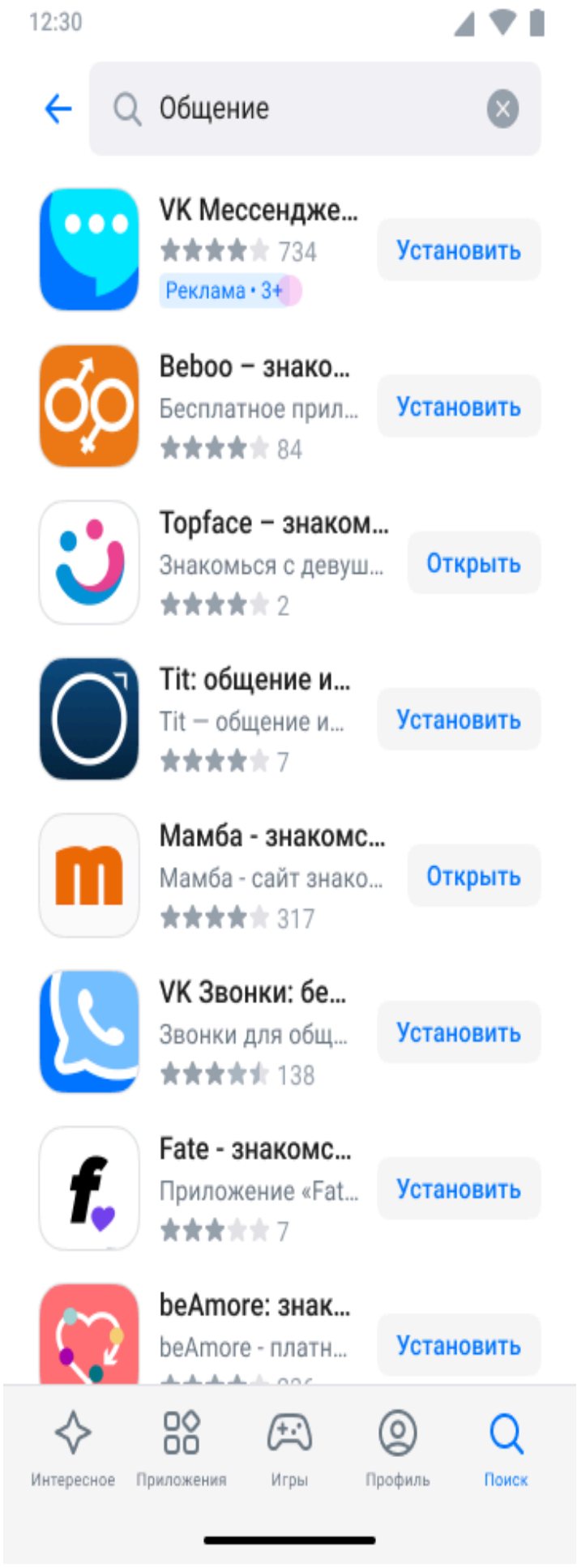


Интересное | Приложения | Игры | Профиль | Поиск

在搜索结果中显示广告：

广告区块也显示在搜索结果的列表中的第一个位置。

点击“安装”后，广告应用程序的下载将开始，与搜索结果中的其他应用程序类似。


12:30
Общение
VK Мессендже...
734
Реклама • 3+
Установить
Beboo – знако...
Бесплатное прил...
84
Установить
Topface – знаком...
Знакомься с девуш...
2
Открыть
Tit: общение и...
Tit — общение и...
7
Установить
Мамба - знакомс...
Мамба - сайт знако...
317
Открыть
VK Звонки: бе...
Звонки для общ...
138
Установить
Fate - знакомс...
Приложение «Fat...
7
Установить
beAmore: знак...
beAmore - платн...
Установить
Интересное
Приложения
Игры
Профиль
Поиск

申请活动

活动（in-app events）是提高应用程序安装动机的一种方式，例如：对于网上商店的销售、游戏中的特别优惠和活动，以及其他有限时限或使用条件的促销活动。

RuStore 控制台允许您提交活动的申请，其活动的信息将在您的应用程序中显示。您可以为包含已发布版本的应用程序创建活动申请。

要管理活动申请，请登录RuStore 控制台。

活动申请的审查期限最长为14天。如果14天内没有联系您，这意味着您的申请未获RuStore编辑部的批准。

一个应用程序内最多可以同时提交5个即将发生活动的申请。

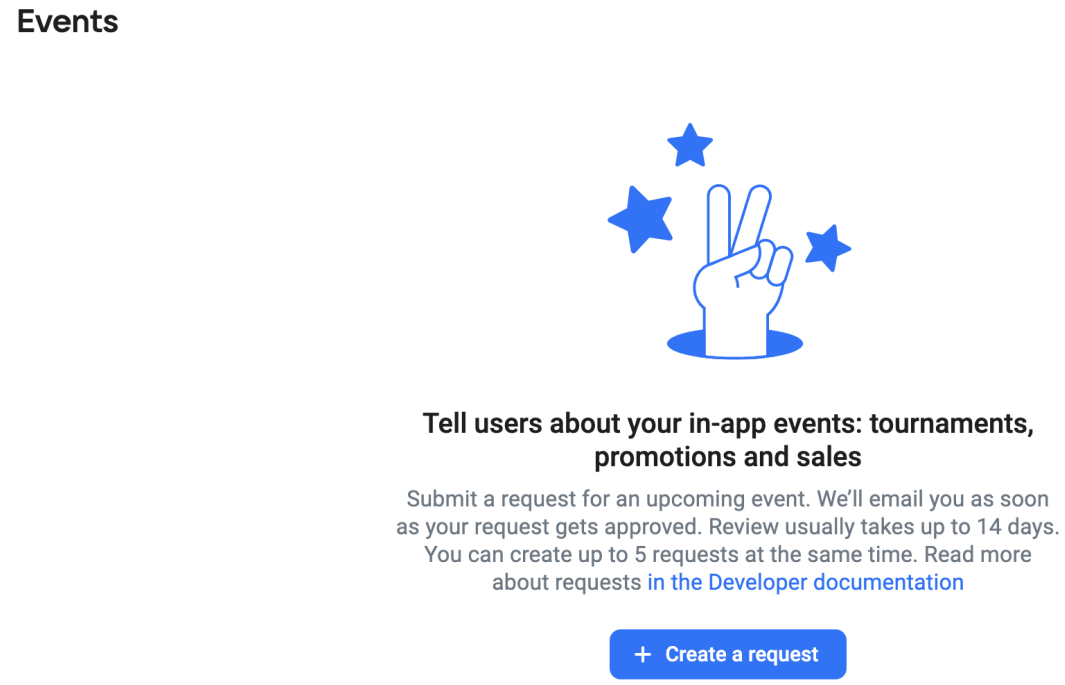
创建申请

要创建一个活动申请，请执行以下步骤。

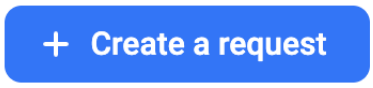
1。在左侧导航菜单中选择**推广** > **活动申请**。

根据您是否已经创建了申请，将显示以下页面。

没有申请

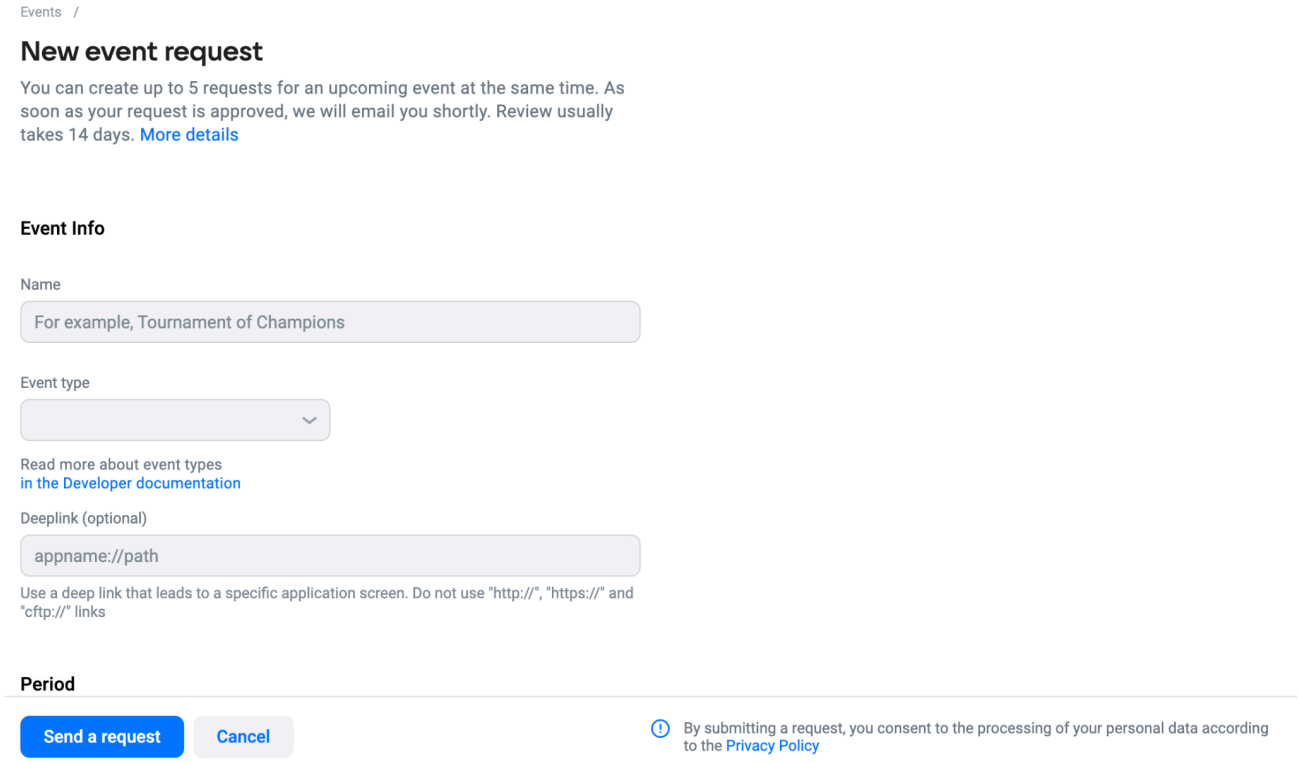


有已创建的申请



2。点击 。

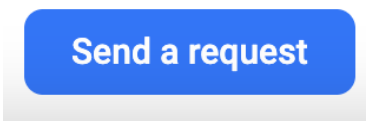
将显示创建活动申请的表单（见下图）。



3。根据下方提供的表格输入必要信息。

设置	描述
活动名称	输入活动名称（最多34个字符）。名称应简洁明了，尽可能简短地传达活动的本质。
活动申请类型	从列表中选择活动类型：- **新季节**；- **促销活动**；- **比赛**；- **销售**；- **大更新**；- **特别活动**；- **首映**；- **直播**。
开始日期/时间	活动开始的日期和时间。确保用户能立即利用特别条件。
结束日期/时间	活动结束的日期和时间。应该大于当前日期和时间。- 活动的最长持续时间为90天。- 活动的最短持续时间为24小时。确保活动的结束不会延迟过久，以免用户将这一机会推迟到以后。
简短描述	输入活动的简短描述 — 最多32个字符。- 简短描述中建议包含行动号召或有关活动的其他有用信息（例如，“享受高达90%的折扣”/“仅在RuStore独家”等）。- 以礼貌的“您”形式与用户交流，以保持RuStore的总体语调一致。- 如果活动中用户获得的使

设置	描述
	用条件与常规条件不同，请创造性地展示这一点 — 这应成为行动号召。
描述	输入活动的详细描述 — 最多448个字符。- 描述不仅应包含有关您应用的详细信息，还应包括相关活动的上下文。- 描述应简洁并包含活动的主要优势或参与条件（如激活促销码/在哪个部分寻找礼品盒/在注册时领取礼品等）。
封面	使用链接指定封面文件的路径，或将其拖放到指定区域： 文件格式：JPG； 文件大小：不超过3MB； 宽高比：4:3； 分辨率：2880x2160 px。 图像质量应该很好。封面的详细要求：见智能手机上的活动；见平板电脑上的活动。设计师制作活动的指南
底色	请输入HEX颜色代码以在封面上自动生成渐变色。输入的代码对应的颜色将显示在下方的 方框中。最好选择与您的图像颜色相符的代码。
文本的颜色	选择显示文本的颜色——白色或黑色。请注意，文本在底色上应清晰可见：如果底色是暗色，选择白色文本；如果底色是亮色，文本应为黑色。
申请考虑的优先级	选择您的申请考虑优先级：- **标准** — 默认设置，申请按一般顺序考虑；- **高** — 申请将被优先考虑。高优先级的考虑只能设置在即将发生的活动的1个申请上。
电话	请使用**+79876543210**格式输入联系方式电话号码。
电子邮件	输入电子邮件地址，以便在申请获得批准或需要澄清问题时与您联系。
给审核员的评论	如果有需要在活动框架内考虑的额外信息，请在此字段输入。

4。点击 

。

提交申请后，您在申请卡中指定的邮箱将收到一封邮件（关于RuStore编辑团队已收到申请的通知）。如果没有收到邮件，请检查垃圾邮件文件夹。

获批准的活动将在应用程序卡的活动日期中显示。此外，如果您的活动被RuStore编辑团队选中，它还可能出现在“精选”，“游戏”，“应用程序”页面的精选列表中。

查看申请

要查看已创建的活动申请，请执行以下操作：

1. 在左侧导航菜单中选择**推广** > **活动申请**。

   将显示已创建申请的列表页面。

1. 从列表中选择所需的申请。

   将显示该申请的详细信息页面。

智能手机上的活动

参见：设计师活动设计指南。

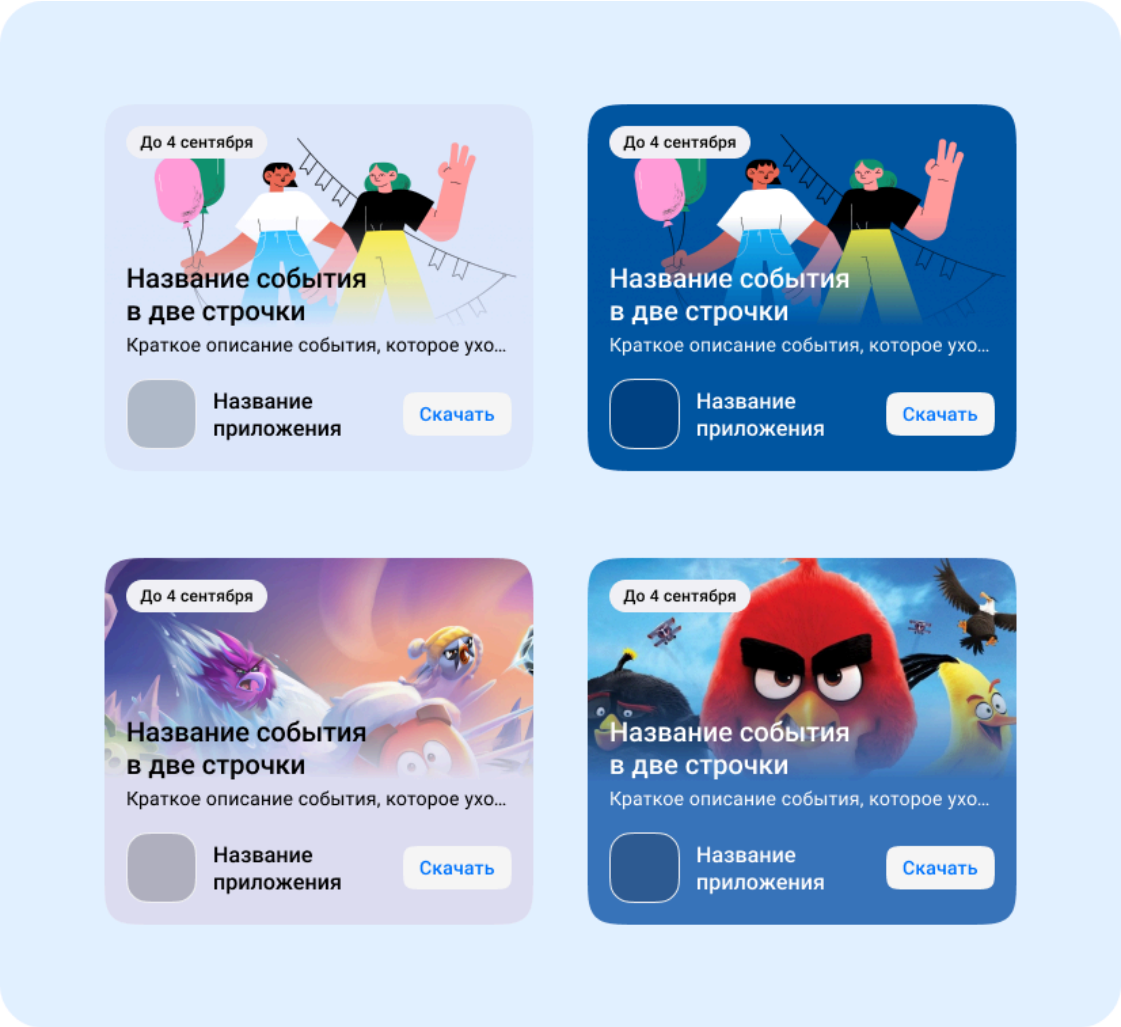
封面可以是：

- 在亮色背景上：文本为黑色。
- 在暗色背景上：文本为白色。

移动应用中最终卡片的尺寸为**316x270**。

卡片包括：

- 标题：最大字符数 — 34（两行）。
- 简要描述：最大字符数 — 32（一行）。
- 应用程序的图标和名称，以及“下载”按钮。



在开放状态下的活动

在开放状态下的活动是一个具有固定高度和宽度的弹出式活动窗口。

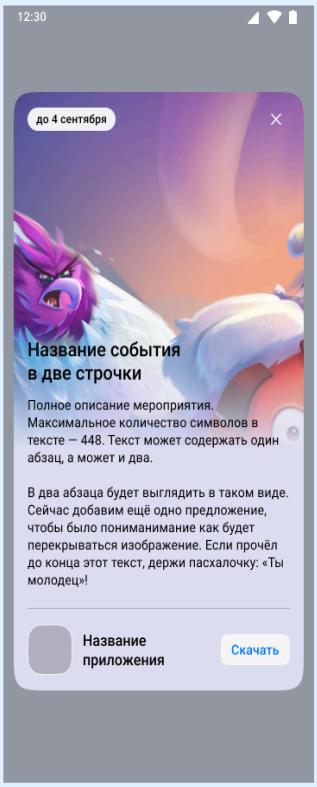
- 边缘留有XS（12px）的间隙。
- 上方 — XL17（88 px）。
- 下方 — XL19（96 px）。

卡片包括：

- “关闭”按钮
- 标题：最大字符数 — 34（两行）。
- 描述：最大字符数 — 448。


12:30
до 4 сентября
Название события
в две строчки
Полное описание мероприятия. Максимальное количество символов в тексте — 448. Текст может содержать один абзац, а может и два.
В два абзаца будет выглядить в таком виде. Сейчас добавим ещё одно предложение, чтобы было пониманимание как будет перекрываться изображение. Если прочёл до конца этот текст, держи пасхалочку: «Ты молодец»!
Название приложения
Скачать


12:30
до 4 сентября
Название события
в две строчки
Полное описание мероприятия. Максимальное количество символов в тексте — 448. Текст может содержать один абзац, а может и два.
В два абзаца будет выглядить в таком виде. Сейчас добавим ещё одно предложение, чтобы было пониманимание как будет перекрываться изображение. Если прочёл до конца этот текст, держи пасхалочку: «Ты молодец»!
Название приложения
Скачать

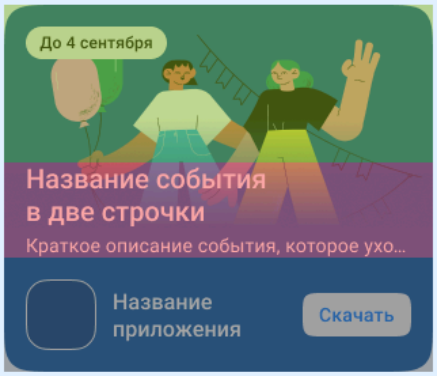

12:30
до 4 сентября
Название события
в две строчки
Полное описание мероприятия. Максимальное количество символов в тексте — 448. Текст может содержать один абзац, а может и два.
В два абзаца будет выглядить в таком виде. Сейчас добавим ещё одно предложение, чтобы было пониманимание как будет перекрываться изображение. Если прочёл до конца этот текст, держи пасхалочку: «Ты молодец»!
Название приложения
Скачать


12:30
до 4 сентября
Название события
в две строчки
Полное описание мероприятия. Максимальное количество символов в тексте — 448. Текст может содержать один абзац, а может и два.
В два абзаца будет выглядить в таком виде. Сейчас добавим ещё одно предложение, чтобы было пониманимание как будет перекрываться изображение. Если прочёл до конца этот текст, держи пасхалочку: «Ты молодец»!
Название приложения
Скачать

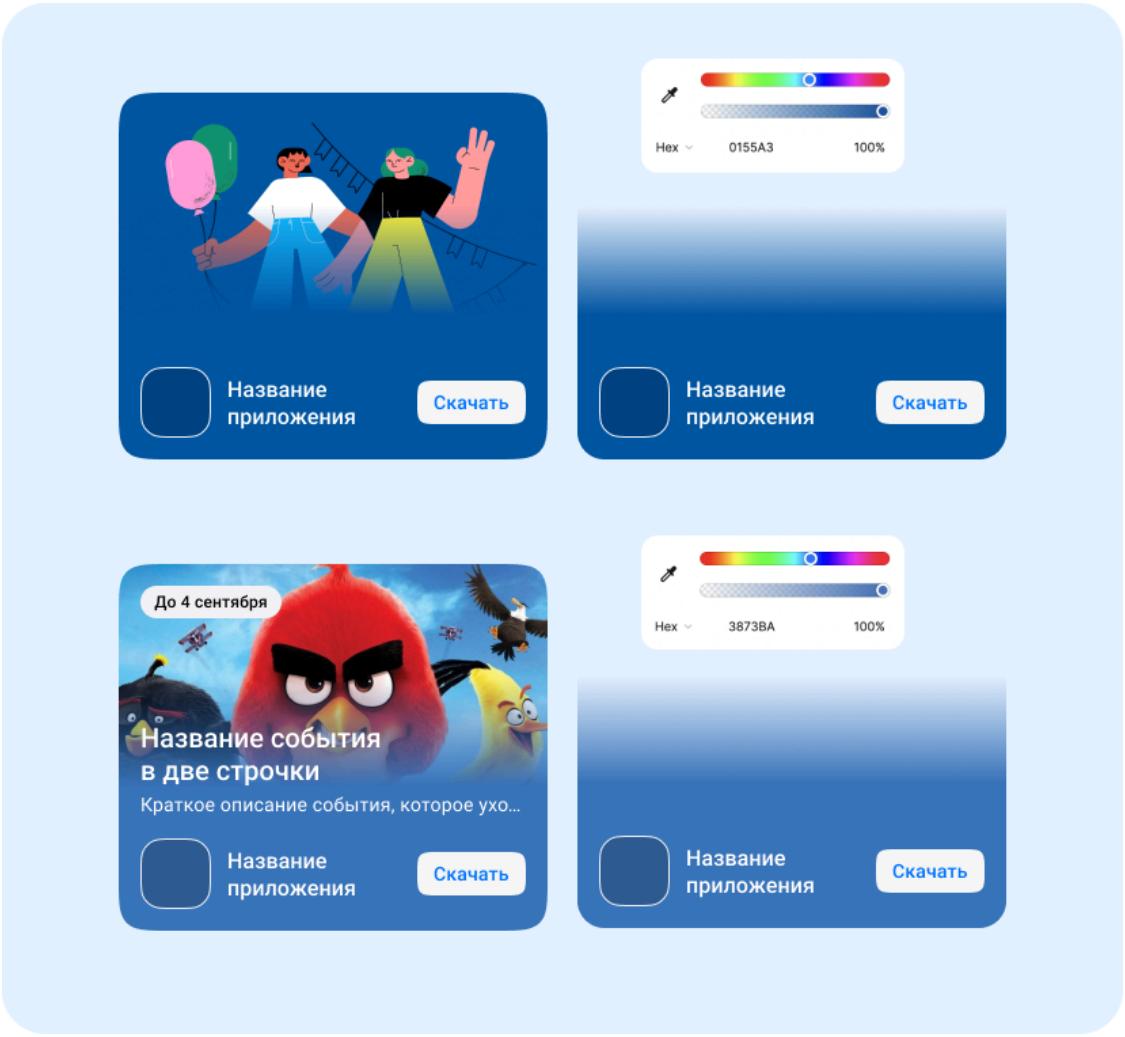
图片

上传的图片格式为JPG，大小3MB，比例4:3，分辨率2880x2160 px（2x）。

图片中的重要元素应位于顶部中央。



图片底部，包括应用图标、名称以及“下载”按钮处，将添加填充和渐变效果。填充颜色最好与图片颜色相匹配。颜色以HEX格式指定。


Название приложения
Скачать
Hex
0155A3
100%
Название приложения
Скачать
До 4 сентября
Название события в две строчки
Краткое описание события, которое ухо...
Название приложения
Скачать
Hex
3873BA
100%
Название приложения
Скачать

图片将在容器中心拉伸。



ИСХОДНОЕ ИЗОБРАЖЕНИЕ



ИЗОБРАЖЕНИЕ В КАРТОЧКЕ СОБЫТИЯ



ИСХОДНОЕ ИЗОБРАЖЕНИЕ



ИЗОБРАЖЕНИЕ В КАРТОЧКЕ СОБЫТИЯ

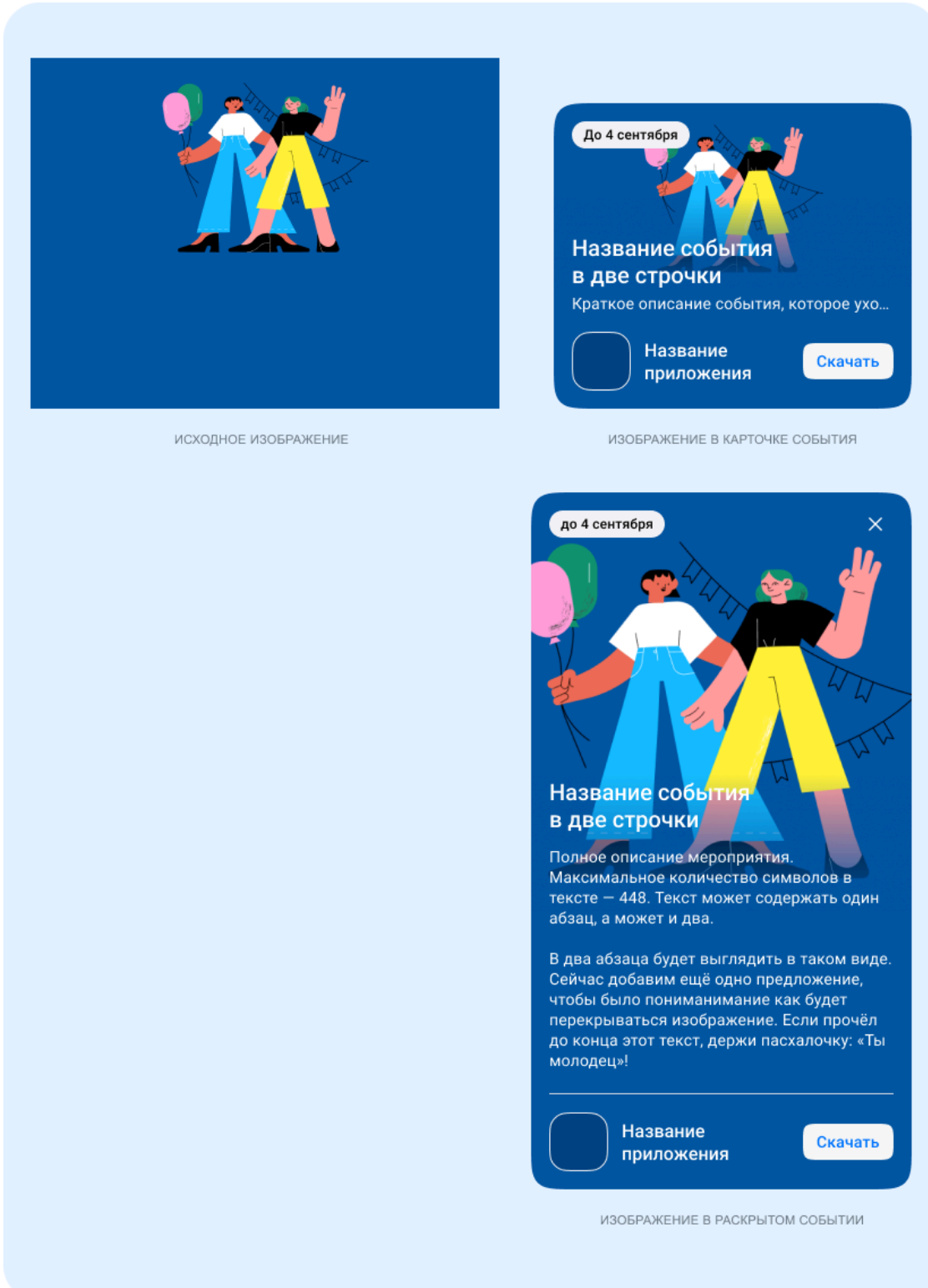


ИЗОБРАЖЕНИЕ В РАСКРЫТОМ СОБЫТИИ



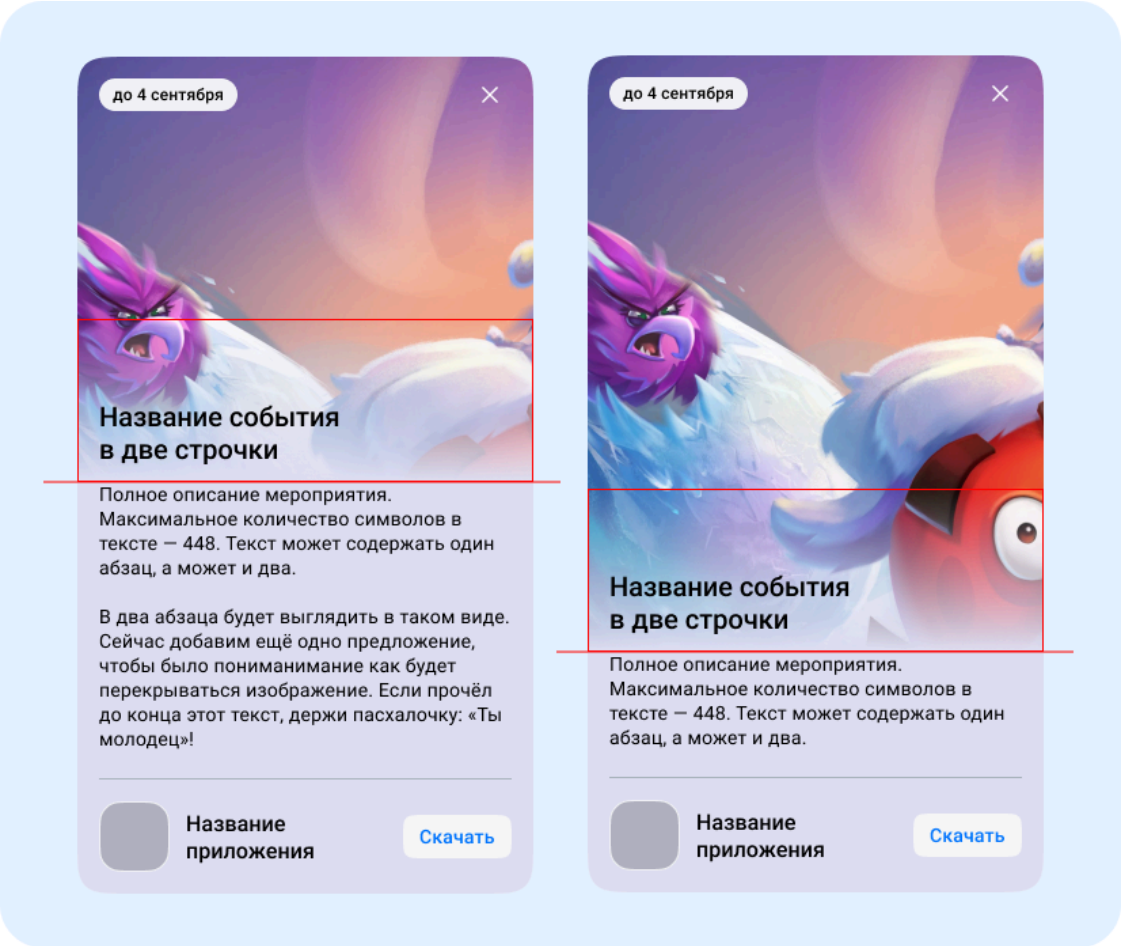
ИЗОБРАЖЕНИЕ В РАСКРЫТОМ СОБЫТИИ

如果图片上有重要细节，且希望图片不在整个卡片上拉伸，可以将其放置在图片顶部中心。



渐变和底层

底层覆盖整个活动描述区域和带有按钮的应用程序区域，渐变从描述的开始向上延伸120px。



平板上的活动

参见：设计师活动设计指南。

封面可以是：

- 在亮色背景上：文本为黑色。
- 在暗色背景上：文本为白色。

在平板电脑上的最终卡片尺寸为**357x404**。

卡片包括：

- 标题：最大字符数 — 34（两行）。
- 简要描述：最大字符数 — 32（一行）。
- 应用程序的图标和名称，以及“下载”按钮。



在开放状态下的活动

在开放状态下的活动是一个具有固定高度和宽度的弹出式活动窗口。

- 边缘留有S（16px）的间隙。
- 卡片固定高度为560px，居中放置于平板电脑上。

卡片包括：

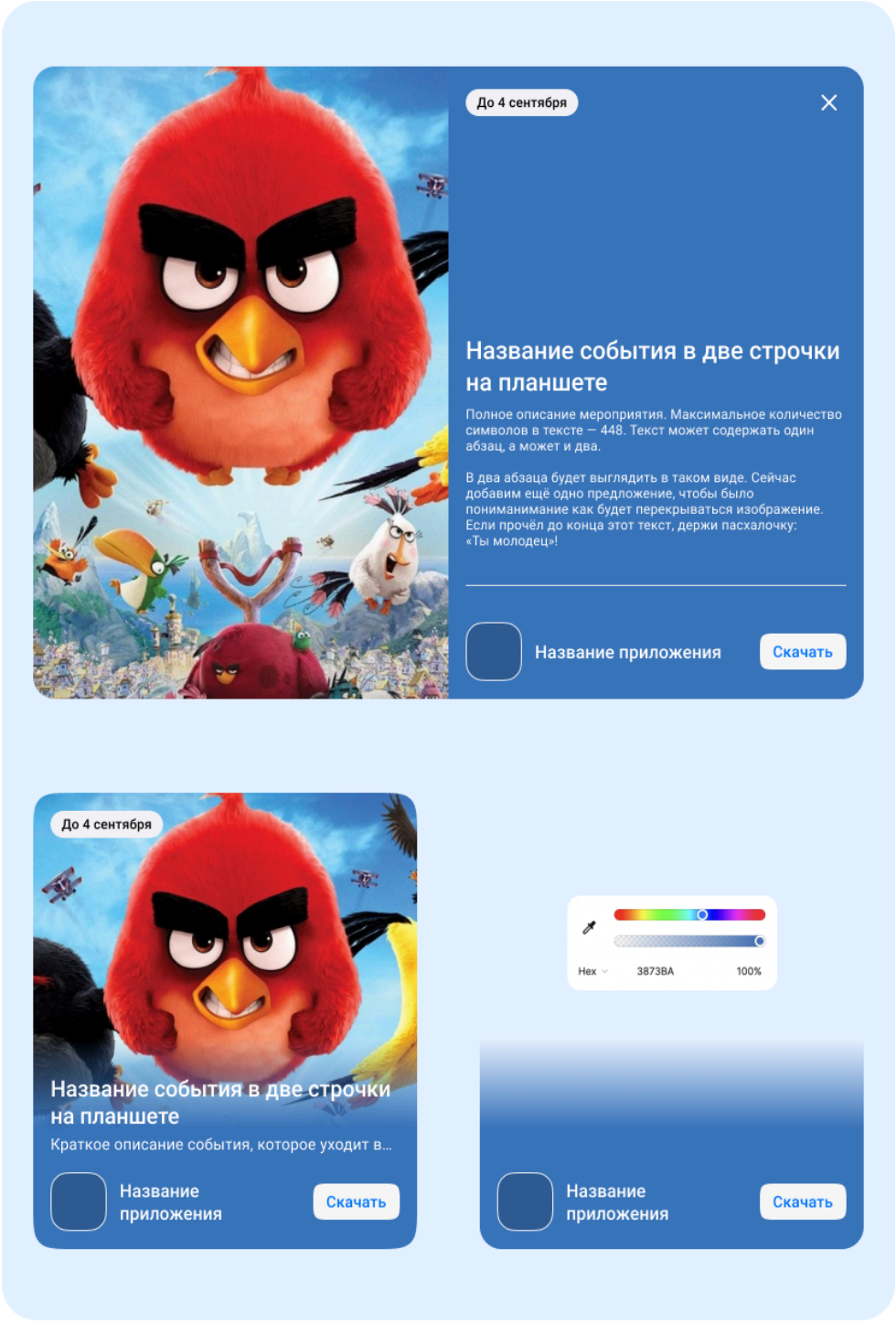
- “关闭”按钮。
- 标题：最大字符数 — 34（两行）。
- 描述：最大字符数 — 448。

内容布局：

内容分为两部分，各占50%，左边是图片，右边是信息：

- 活动名称；
- 完整描述；
- 应用程序的图标和名称，以及“下载”按钮。

右侧信息区域将填充在添加活动时指定的颜色，颜色以HEX格式给出。这种颜色也会出现在摆放在精选列表中的缩略图卡片上。



图片

图片将在容器中心拉伸。





布局

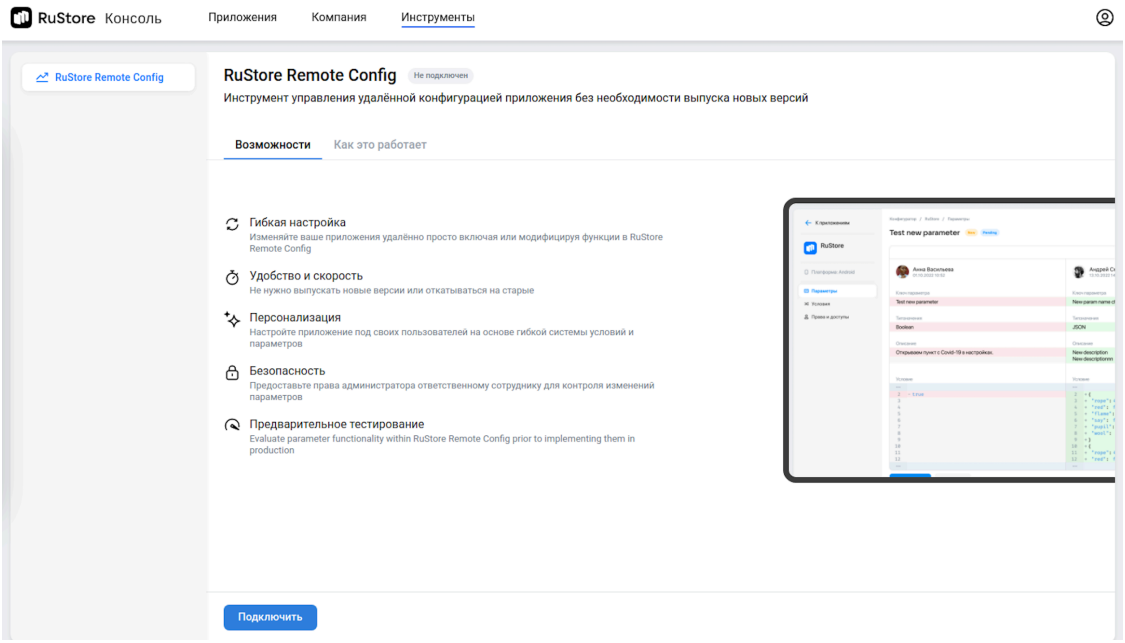
- 如果是单一活动：与封面的精选集成对。
- 如果有两个活动 — 它们并排显示。
- 如果有三个或更多活动，会出现水平滚动。

工具

使用RuStore控制台中的“工具”部分将开发者工具集成到您的应用程序中。

这些工具为开发者提供了额外的应用程序管理、错误查找和与各种服务集成的功能。

只有拥有组织所有者权限的用户才能集成工具。



工具集成

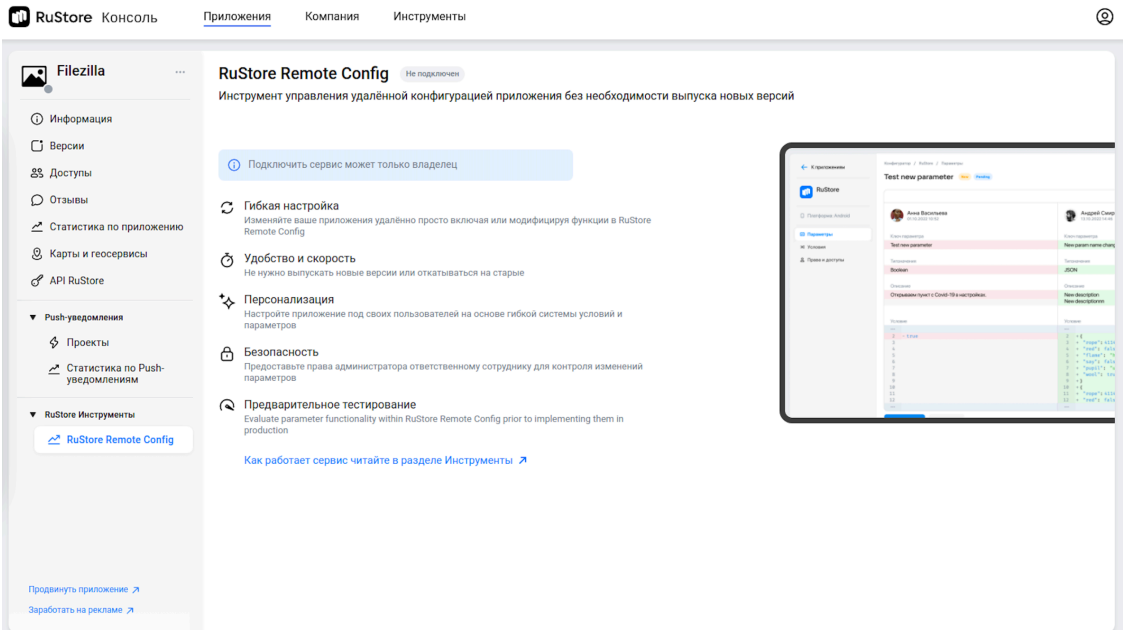
1。转到“工具”选项卡。

2。从侧边菜单中选择您感兴趣的工具并点击“集成”。系统将显示集成结果的消息。

如果出现错误，请写信给支持团队。

如果从共用页面进行集成，该工具将集成到您在RuStore 控制台中至少创建了一个版本的所有应用程序。

工具也可以通过应用卡片的侧边菜单进行集成。在这种情况下，工具只会集成到选定的应用程序。



集成后，您将能够在工具部分查看所有集成的应用程序，并直接进入工具界面。

在特定服务中进行集成时，将为RuStore中组织的所有者创建一个账户。为了设置您组织中其他员工的访问权限，集成后您需要进入服务并在工具界面设置访问权限。

RuStore Remote Config

概述

RuStore Remote Config 是一款用于管理移动应用程序配置的工具。它允许实现 feature toggle（或 feature switcher）功能，并通过方便的图形界面进行管理。

快速启动

- 安装 SDK 并配置应用程序；
- 创建将启用参数的条件；
- 在 Remote Config 中描述参数。

附加信息

- 用户权限。
- 服务工作原理。

工作原理

在创建新的应用程序时，其代码必须配置为包含可以通过配置更改的参数。因此，Remote Config 不能与任何应用程序一起工作：首先需要将服务代码适配到 RuStore Remote Config 的工作中。

应用程序启动时，SDK 向 RuStore Remote Config 发送请求，并作为响应获得配置 - 一个包含一组特定参数的 JSON。与配置一起（更改应用程序行为的参数），响应中还会收到配

置的哈希。应用程序记住这个哈希，并在后续对 Remote Config 的请求中将其作为一个参数发送。

Remote Config 将收到的哈希与此请求的配置哈希进行比较。如果它们匹配，服务将返回“未修改”的响应，而不是返回未更改的配置。这是为了避免向客户端返回未更改的配置。

获取配置后，设备上的界面将被绘制，或者根据 RuStore Remote Config 中的配置参数更改功能。

应用程序仅在重启后应用收到的配置。用户不会实时看到界面的变化。

SDK 和应用程序配置

在项目中集成

连接存储库：

build.gradle

```
repositories {    ``maven {        ``url = uri( "https://artifactory-external.vkpartner.ru/artifactory/maven" )    ``}}
```

集成依赖项

在您的配置文件中添加以下代码以集成依赖项：

build.gradle

```
dependencies {    ``implementation( "ru.rustore.sdk:remoteconfig:0.0.2" )}
```

集成 SDK 的应用程序示例。

概述

SDK Remote Config 是一种云服务，允许更改您的应用程序的行为和外观，而无需用户下载应用程序更新。SDK 封装了从服务器请求配置、缓存和后台更新。它具有方便的 API 用于获取数据。

示例应用程序可通过链接访问。

主要特点

- SDK 允许选择最方便的配置更新机制。
- 能够指定配置在受众中的分布百分比。
- 为构建特定配置的漏斗，可以传递附加信息。甚至可以为特定用户定制配置。
- 具有一组关于 SDK 工作的回调，可用于分析。
- 外部依赖项的数量最少。

在项目中集成

连接存储库：

build.gradle

```
repositories {    ``maven {        ``url = uri(
"https://artifactory-external.vkpartner.ru/artifactory/maven" )    ``}}
```

集成依赖项

在您的配置文件中添加以下代码以集成依赖项：

build.gradle

```
dependencies {    ``implementation( "ru.rustore.sdk:remoteconfig:0.0.2" )}
```

创建和初始化 RemoteConfigClient

RemoteConfigClient 的初始化应该在 Application.onCreate() 时刻进行，因为在启动 SDK 的后台同步时，它需要被初始化。

创建 RemoteConfigClient

```
val remoteConfigClient = RemoteConfigClientBuilder(appId = AppId( "your_app_id"
), context = applicationContext).build()
```

通过 RemoteConfigClientBuilder，您可以设置额外的参数，这些参数可用于获取特定配置。

重复调用 RemoteConfigClientBuilder.build() 将导致 RemoteConfigClientAlreadyExist 错误。

调用 RemoteConfigClientBuilder.build() 方法后，将创建 RemoteConfigClient 的实例并记录为单例。可以通过以下方式获取创建的实例：

```
RemoteConfigClient.instance
```

在通过 RemoteConfigClientBuilder.build() 创建之前通过静态变量访问将导致 RemoteConfigClientNotCreated 错误。

RemoteConfigClientBuilder 的可选参数

参数	描述
OsVersion	**配置器中的条件：Os Version**。允许将 OsVersion 与界面中设置的值进行比较。默认情况下，OsVersion 不传递，在这种情况下，将返回默认配置。

参数	描述
DeviceModel	**配置器中的条件：Device Model**。允许将 DeviceModel 与界面中设置的值进行比较。默认情况下，DeviceModel 不传递，在这种情况下，将返回默认配置。
Language	**配置器中的条件：Language**。允许将 Language 与界面中设置的值进行比较。默认情况下，Language 不传递，在这种情况下，将返回默认配置。要传递 Language，需要实现 ConfigRequestParameterProvider。
Account	**配置器中的条件：Account**。允许将 account 与界面中设置的值进行比较。**配置器中的条件：Account Percentile**。允许根据 account 值在特定百分比下分发配置。**配置器中的条件：Interval Account Percentile**。允许在特定的一天和特定百分比下根据 account 值分发配置。要传递 Account，需要实现 ConfigRequestParameterProvider。
DeviceId	**配置器中的条件：DeviceID**。允许将 DeviceId 与界面中设置的值进行比较。**配置器中的条件：DeviceID Percentile**。允许根据 DeviceId 值在特定百分比下分发配置。**配置器中的条件：Interval DeviceID Percentile**。允许在特定的一天和特定百分比下根据 DeviceId 值分发配置
AppVersion	**配置器中的条件：App Version**。允许将 AppVersion 与界面中设置的值进行比较
Environment	**配置器中的条件：App Environment**。允许将 Environment 与界面中设置的值进行比较。Environment 可以采用以下值：Alpha、Beta、Release。此参数适用于在应用程序的不同构建版本上测试配置。
AppBuild	**配置器中的条件：App Build**。允许将 AppBuild 与界面中设置的值进行比较

UpdateBehaviour

UpdateBehaviour 是一个参数，用于确定 SDK 的行为。创建 RemoteConfigClientBuilder 实例时，默认使用 UpdateBehaviour.Default，同步间隔设为 15 分钟。

可以通过以下方式设置 SDK 的其他行为：

```
remoteConfigClientBuilder.setUpdateBehaviour(UpdateBehaviour.Actual)
```

UpdateBehaviour 的差异

UpdateBehaviour	描述
UpdateBehaviour.Actual	使用这种更新类型进行初始化时，**每个**配置请求都将通过服务器请求进行。**这种更新类型保证了获取最新的配置，但配置请求的速度将取决于网络速度。**这种初始化类型取消了后台更新过程。
UpdateBehaviour.Default(minSyncInterval)	使用这种更新类型进行初始化时，配置请求将来自本地持久存储，该存储将在指定的时间间隔内更新。如果是第一次初始化（本地持久存储为空），则会向服务器发出请求以填充它，时间取决于数据量和网络速度。配置访问将等待初始化完成。**这种初始化类型不保证在进程的生命周期内配置获取将给出相同的结果，因为后台可能会发生数据同步。**这种初始化类型启动了后台更新过程。
UpdateBehaviour.Snapshot(minSyncInterval)	使用此更新类型进行初始化时，配置请求将从本地 inMemory 存储中获取。inMemory 存储将使用来自持久存储的首次请求的结果填充，并将其保存到进程生命周期结束。如果是首次初始化（本地持久存储为空），则会向服务器发出请求以填充它，时间取决于数据量和网络速度。配置访问将等待初始化完成。**这种初始化类型保证在进程的生命周期内配置请求将返回相同的结果。**这种初始化类型启动了后台更新过程。

ConfigRequestParameterProvider

实现 ConfigRequestParameterProvider 可以动态地向服务器传递参数以同步配置。

支持的参数列表包括：Language 和 Account

```
class ConfigRequestParameterProviderImpl : ConfigRequestParameterProvider
{    ``override fun getConfigRequestParameter(): ConfigRequestParameter
=        ``ConfigRequestParameter(            ``language =
Language(Locale.getDefault().language),            ``account =
Session.id,        ``)}
```

方法的实现应快速完成，不包括长时间的操作。实现耗时较长的 getConfigRequestParameter 方法可能会导致获取配置时出现较大延迟。

该方法的调用将在 RemoteConfig SDK 的多个线程上进行。

可以按以下方式添加 ConfigRequestParameterProvider 的实现：

```
val provider =
ConfigRequestParameterProviderImpl()remoteConfigClientBuilder.setConfigRequestPa
rameterProvider(provider)
```

RemoteConfigClientEventListener

实现 RemoteConfigClientEventListener 可以获得关于 SDK 工作的回调，如初始化完成、持久存储的更新等。

监听器的回调在 MainThread 上调用。

可以按以下方式添加 RemoteConfigClientEventListener 的实现：

```
val listener =
RemoteConfigClientEventListenerImpl()remoteConfigClientBuilder.setRemoteConfigCl
ientEventListener(listener)
```

RemoteConfigEventListener 的实现示例：

```
class RemoteConfigListenerImpl: RemoteConfigClientEventListener {        ``override
fun backgroundJobErrors(exception: RemoteConfigException.BackgroundConfigUpdateError)
{                ``//返回后台操作错误        ``}        ``override fun
firstLoadComplete() {                ``//在首次加载结束时调用
        ``}        ``override fun initComplete() {                ``//在初始化结束时调
用        ``}        ``override fun memoryCacheUpdated() {                ``//当内存中
的缓存发生变化时调用        ``}        ``override fun persistentStorageUpdated()
{                ``//当持久存储发生变化时调用        ``}        ``override fun
remoteConfigNetworkRequestFailure(throwable: Throwable) {                ``//当 Remote
Config 的网络请求错误时调用``}}
```

RemoteConfigClient 初始化

RemoteConfigClient 的初始化是异步的，可以通过返回 init() 方法的 Task 类或通过监听器来跟踪初始化结果。

```
remoteConfigClient.init()
```

从 RemoteConfigClient 获取配置

从 RemoteConfigClient 获取配置是通过调用 getRemoteConfig() 方法进行的。

```
remoteConfigClient.getRemoteConfig()
```

如果之前没有初始化 RemoteConfigClient，则在首次调用 getRemoteConfig() 方法时将进行初始化，这可能需要一些时间。为了更快地获取配置，应提前进行 RemoteConfigClient 的初始化。

GetRemoteConfig() 方法返回 Task<RemoteConfig> 并且是异步执行的。Task 可能会失败。SDK RemoteConfig 错误

成功完成 Task 后，将可以访问 RemoteConfig，这是根据选择的更新策略和初始化参数获得的所有数据的当前集合。在某些更新策略下，获取配置将不依赖于当前的初始化参数，而是依赖于在后台同步期间使用的参数。

RemoteConfig 类

RemoteConfig 的实例是根据初始化时选择的更新策略获得的所有数据的当前集合，实例拥有从服务器传递的所有键，这些键取决于初始化时指定的参数。

检查键的可用性

```
remoteConfig.containsKey( "my_key" )
```

此方法检查当前 RemoteConfig 实例中是否存在键。

获取类型化数据

RemoteConfig 类具有获取数据和将其转换为特定类型的方法。

- getBoolean(key: String): Boolean
- getInt(key: String): Int
- getLong(key: String): Long
- getString(key: String): String
- getDouble(key: String): Double
- getFloat(key: String): Float

这些方法可能会返回 RemoteConfigCastException 错误，如果数据类型不符合所选方法或传递的键没有值。

配置的后台同步

在某些更新策略类型下，会使用配置的后台同步。

配置的后台同步结果是更新配置的持久存储，使用最新数据，以便根据所选更新策略进行后续使用。

后台同步之间的最小允许间隔为 15 分钟。

为了后台同步的正确运行，SDK RemoteConfig 的初始化应在 Application.onCreate() 方法中进行。

SDK RemoteConfig 错误

所有 SDK RemoteConfig 错误都有 RemoteConfigException 超类。

RemoteConfigException 的派生类：

错误	描述
BackgroundConfigUpdateError	如果后台同步操作出错，将抛出此异常。

错误	描述
FailedToReceiveRemoteConfig	如果调用获取配置的方法时出错，将抛出此异常。
RemoteConfigCastException	如果从 RemoteConfig 类通过键获取数据不正确，将抛出此异常。错误可能与无法转换类型有关，或者传递的键没有值。
RemoteConfigClientAlreadyExist	如果在进程生命周期内重复创建 RemoteConfigClient，将抛出此异常。
RemoteConfigClientNotCreated	如果在创建 RemoteConfigClient 之前通过静态字段 instance 访问 RemoteConfigClient，将抛出此异常。
RemoteConfigCommonException	常规意外错误。
RemoteConfigNetworkException	如果在同步操作中出现网络错误，将抛出此异常。

SDK Remote Config 更新历史

SDK版本 0.0.2

- SDK的内部更新。

条件

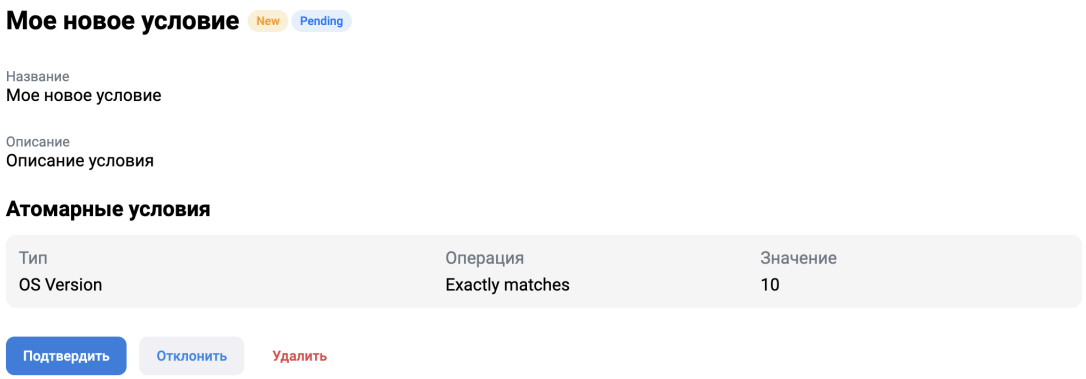
条件定义了应用程序将如何更新。例如，“Android版本高于11版本且设备的语言设置为俄语”。可以将条件称为“目标定位”或“用户细分”。条件本身的创建不会产生任何影响，需要将其应用于要推出的参数。Remote Config 参数

条件的目的是将参数应用于特定的用户细分。所有列表都位于 Conditions 部分。在配置参数时，条件也显示在下拉列表中。

不要在没有必要和没有把握的情况下编辑条件。特别是如果参数是基于条件分发的。这可能会影响应用程序的稳定运作。

创建新条件

要创建新条件，请点击“Add Condition”按钮。



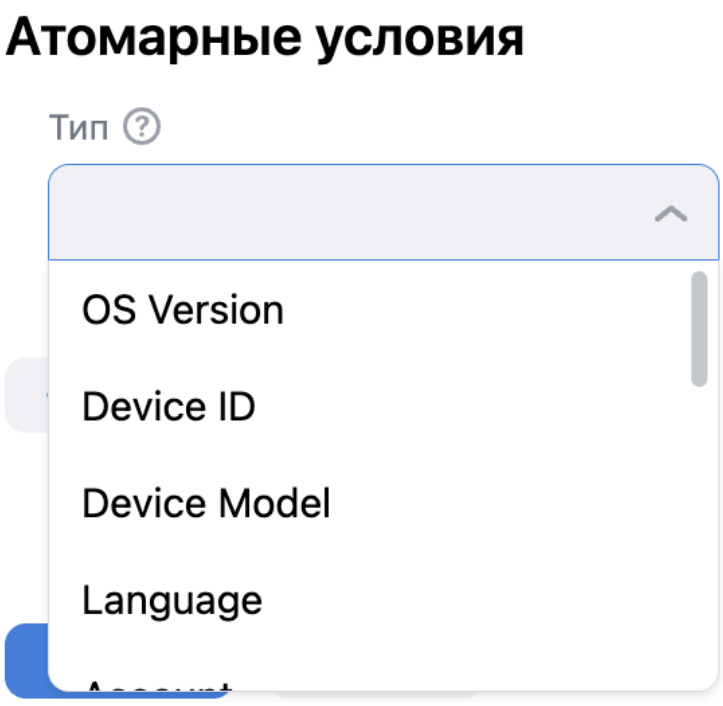
在“名称”字段中指定事件的名称。在“描述”中添加解释性注释。

选择构成条件的“原子条件”。

所有原子条件通过逻辑“与”组合。用户要满足创建的条件，必须满足所有原子条件。

原子条件

原子条件可以是文本或数字。文本字段填写时不使用引号。



文本条件支持的运算符包括：

- Exactly matches.要求完全匹配。

实例。*条件“OS Version exactly matches 9.0.0”只适用于os_version=9.0.0的应用请求，而不适用于 os_version=9.0。*

- Contains.包含子字符串。
- Does not contain.不包含子字符串。
- Regexp.符合正则表达式。

当使用 Golang 的 Regexp 时，需要通过链接检查条件的工作情况，并在测试时进行。

- Presented in file.符合文件中列表的元素。文件的最大大小 — 5 MB。
- Not presented in file.条件包括所有未在上传文件中指定的用户。

对于数字，可用的运算符有：

- =
- != (不等于)
- =
- <
- <=

原子条件列表

Atomic Condition	**描述**	**类型**	使用示例及评论
OS Version	操作系统版本	文字 + 数字	操作系统版本<= 9.90S Version matches regex “^[1-8](\.\d+)*$\|^9(\\.\\d(\\.\\d+)*)*$”
Device Model	设备型号	文字	Samsung 设备：Device Model Contains samsung
Language	设备上的语言	文字	设备上的俄语 Language matches regex ^ru
Account	账户	文字	Account exactly matches mrg.test@mail.ru在这种情况下，如果至少有一个连接的账户是

Atomic Condition	描述	类型	使用示例及评论
			mrg.test@mail.ru，则条件将满足。
App Version	应用程序版本	文字	App Version exactly matches 1.5.3
App Environment	Alpha、Beta 或 Store 版本	可能的值：Alpha，Beta，Release。	该字段的值在初始化 SDK RuStore Remote Config 时设置。
App Build Id	应用程序构建编号	数字	用户在不低于 22563 的构建上 App Build Id >= 22563
DeviceID Percentile	DeviceId 的百分位数	数字	用于对 % 用户发布参数。百分位数 — 一个从 0.00 到 99.99 的数字。该数字由 device_id 的值的函数确定。RuStore Remote Config 将为每个 device_id 返回某个数字。还有一个附加字段：Salt — 当需要对不同的非交叉用户组应用参数时，需要使用它。
Account Percentile	相同，但基于账户	数字	
DeviceID Interval Percentile	由 DeviceID 计算	数字	允许按时间间隔发布功能。**实例**。*你选择了 10% 的用户，指定了Salt和 7 天的间隔。将会看到更新的用户将分布在整个间隔期内：从选定的 10% 中有 1/7 的用户每 7 天收到配置一次。*
Account Interval Percentile	由 Account 计算	数字	类似于 DeviceID Interval Percentile
Random Percent	每次新的百分比		

Atomic Condition	描述	类型	使用示例及评论
Show Time	配置提供时间		指定开始和/或结束提供的时间以及时区。例如，如果需要在周五设置周一启动的事情，或在特定日子关闭某个功能。

Salt 参数

为了使用户进入不同的百分位数，Salt 参数的值必须不同。

例如，如果您希望在对 % 用户发布时始终启用新功能的相同用户，请使用原子条件 **deviceId percentile**。如果您希望每次更新都被不同的用户看到，更改 Salt 参数的值。

条件创建确认

如果条件不是由应用程序所有者或有确认权的用户创建，则需要有此权限的用户或应用程序所有者的确认。用户角色

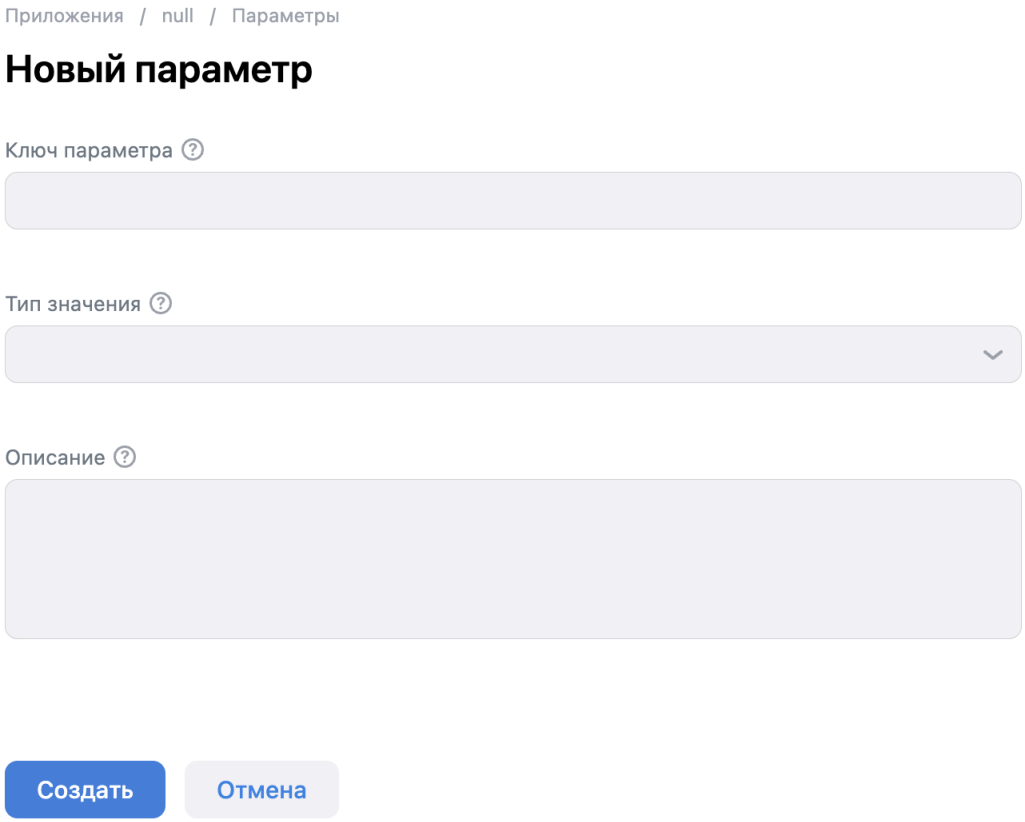
参数

在 RuStore Remote Config 中设置的参数将用于调整应用程序的行为。这些值可以根据满足的条件而有所不同。在应用程序集成时，其代码应该通过 SDK 进行配置，以便能够通过配置更改其中的参数。

小心编辑参数，以免更改影响到用户和应用程序的稳定运行。

创建新参数

点击“创建参数”按钮。您将看到创建参数的界面：



在“参数键”字段中指定其名称。添加评论或描述。

参数类型：

- Boolean — True/False.
- Numeric — 数字。
- String — 文本（单行）。
- Text — 文本（多行）。
- JSON — 传递配置。

参数值的类型只能有一种。

选择参数类型后，添加预先创建的条件。

每个参数可以有多个不同条件下的设定值。条件的计算是按照它们在 RuStore Remote Config 内参数中的描述顺序进行的。例如，如果用户符合条件“A”，则不会检查后续条件。通过这种方式，可以创建任何复杂度的条件组合。

为了重新排列条件，请拖动图标 

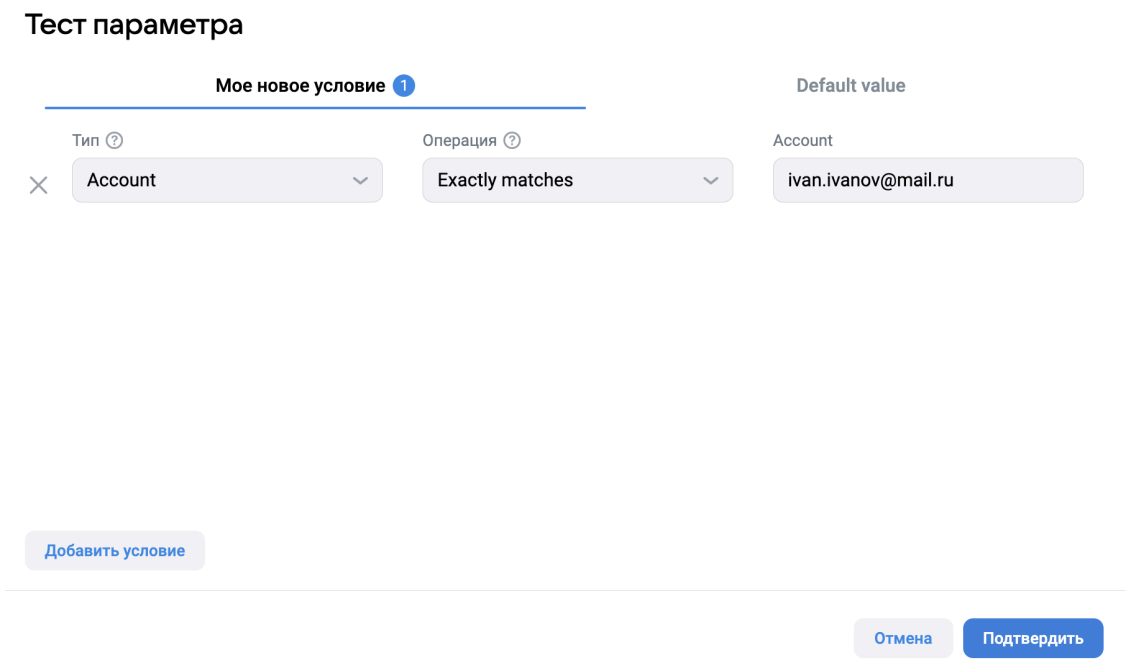
创建参数时，名称旁边会出现一个 New 指示器，表明这是一个新参数。这样的参数可以在不确认的情况下删除，因为它是一个草稿。

测试新参数

在创建、修改或删除参数时，它会进入 **Pending** 状态，即等待确认的模式。但与条件不同，可以在确认更改之前对参数进行测试。测试机制允许将参数应用于特定设备或账户，并确保应用程序的行为正确。

测试是系统的一个重要功能，可以防止应用配置的不正确值。测试的基础是对每个可能的参数版本应用自定义原子条件。

点击“配置测试”按钮后，会出现如下窗口：



顶部的标签页对应于修改参数中添加的所有条件和默认值。在上面的截图中，测试的是根据“我的新条件”获得的值。

点击“启动测试”按钮后，参数进入 **Testing** 状态，在这个状态下可以确认或拒绝更改。

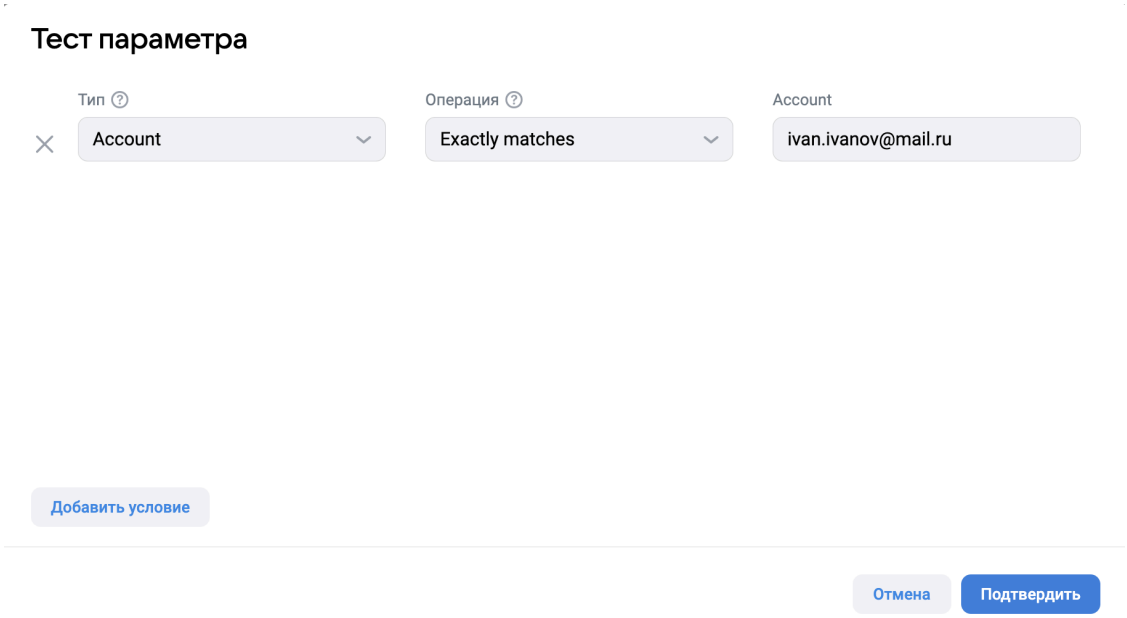
添加的条件完全替代了参数“我的新条件”中给出的原子条件。这是为了测试那些难以在用户设备上重现的复杂情况而设计的。例如，这可能是与应用程序版本相关的条件。您可以将

原始条件更改为自定义条件，并查看在满足参数“我的新条件”时界面的样子。换句话说，服务允许通过自定义条件在设备上获得任何参数值。

第一个条件在选择允许的类型和操作上受限。只有 DeviceID 和 Account 类型以及 Exactly matches 操作是可用的。这样的保护特意设置，以避免在测试中选择过于广泛的用户范围。从第二个条件开始就没有这样的限制。

请注意，同一个 Account 或 DeviceID 只适用于一个条件或默认值。否则，系统将无法确切知道该提供哪个具体值，因此该值可能是任意的并且会不断变化。

删除参数时，标签页不存在，选定的原子条件将全局应用。因为参数正在被删除，我们需要测试其删除，并且在这种情况下，参数内单独条件的具体测试并不重要，因为它们也将与参数一起被删除。实例：



应用新参数

在具有相应访问权限的用户确认参数之后，参数开始生效。用户角色

用户角色和权限

Remote config 为用户提供了使用权限发放机制来灵活管理访问权限的能力。权限针对每个应用程序**单独**授予。

权限列表及其描述

公司所有者拥有公司所有应用程序中的所有权限。

权利	可用操作
查看	查看参数、条件和变更历史。
写入	创建、修改、删除和拒绝参数和条件，以及测试参数。此权限**不允许**确认更改。
确认	确认或拒绝系统中的任何更改。
授予权限	在当前应用程序中向系统中的任何员工授予权限。

处理参数和条件

Remote Config 保护用户免受参数和条件的非故意操作，因此任何可能影响应用程序行为的更改都将被发送以确认。并非所有更改都需要确认。例如，更改参数或条件的描述不会影响系统的行为，因此更改将立即生效。

权利和访问

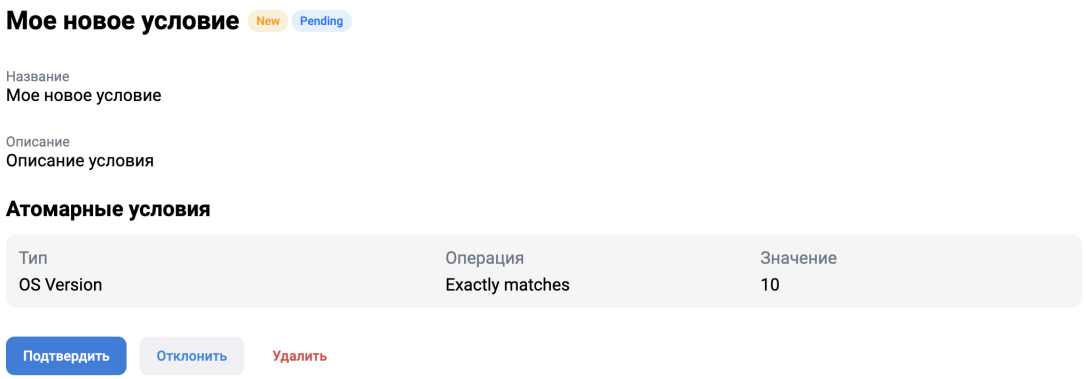
重要的是要说，执行更改确认操作的权利是分开的。用户角色

处理条件

在创建、修改或删除条件时，它会进入 **Pending** 状态，即进入等待确认的模式。

根据操作类型，界面将呈现不同的样式。

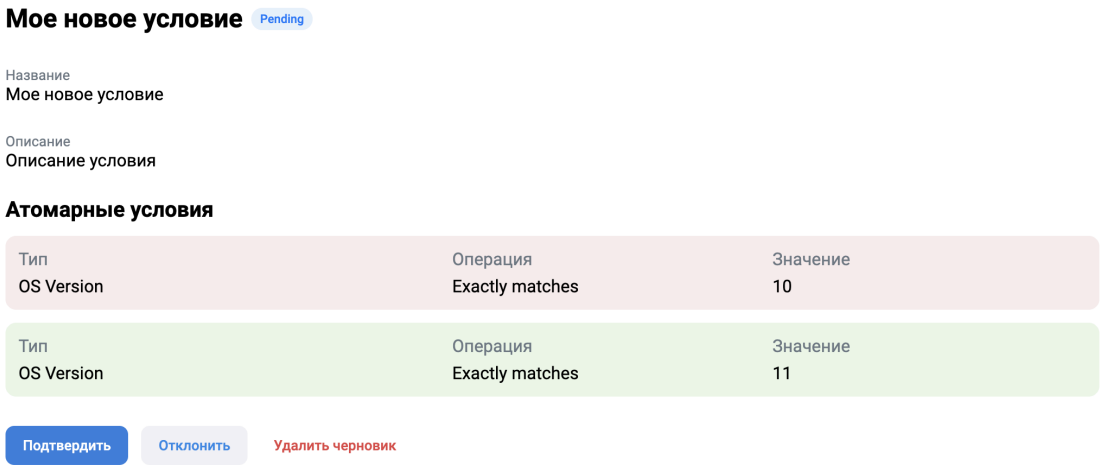
创建条件



当创建条件时，名称旁会出现一个标示为 New 的指示器，表明这是一个新条件。这种条件可以在不经确认的情况下删除，因为它仅是一个草稿。

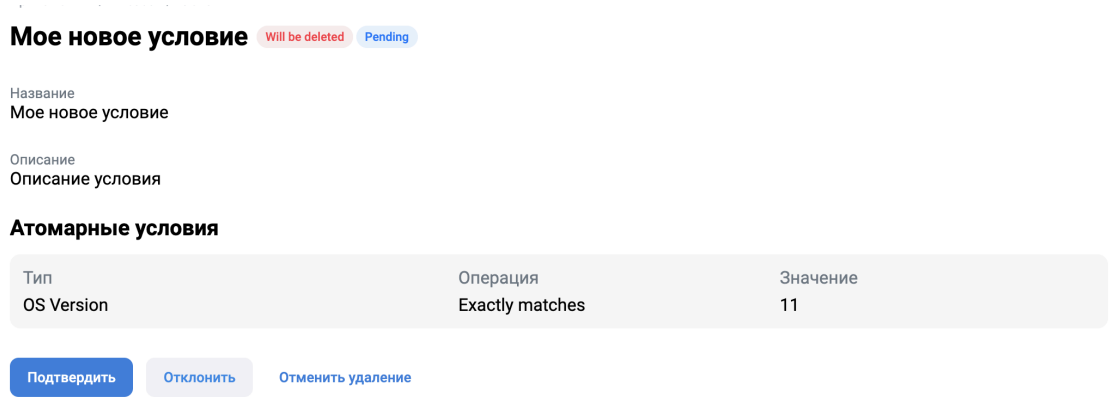
一旦确认，就会创建一个完整的条件。在参数中使用它。

条件的更改



您可以比较旧条件与新条件，并通过删除草稿来撤销更改。任何参数都将与原始版本而非草稿一起工作。一旦确认，所有更改将生效，用户将根据新条件接收所有相关参数。

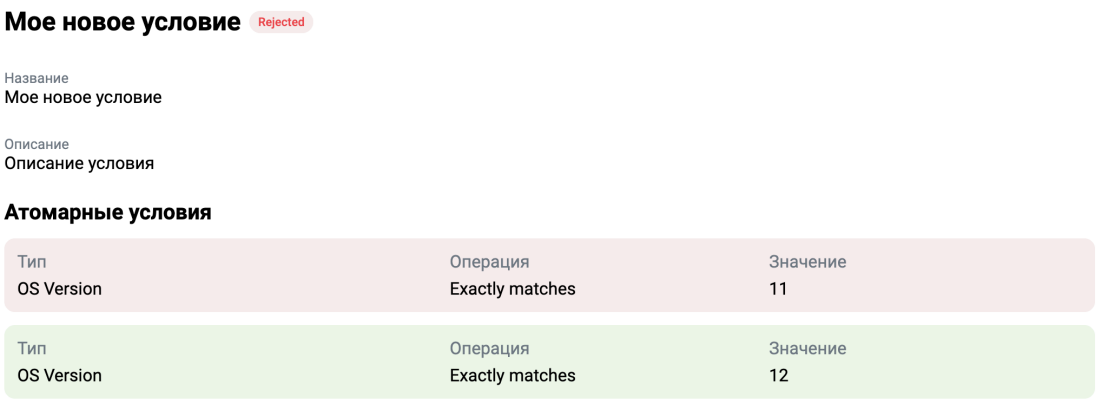
条件的删除



删除条件时，名称旁会出现一个标示为 `Will be deleted` 的指示器。如果您想要从系统中完全删除该条件，请确认此操作。

更改的拒绝

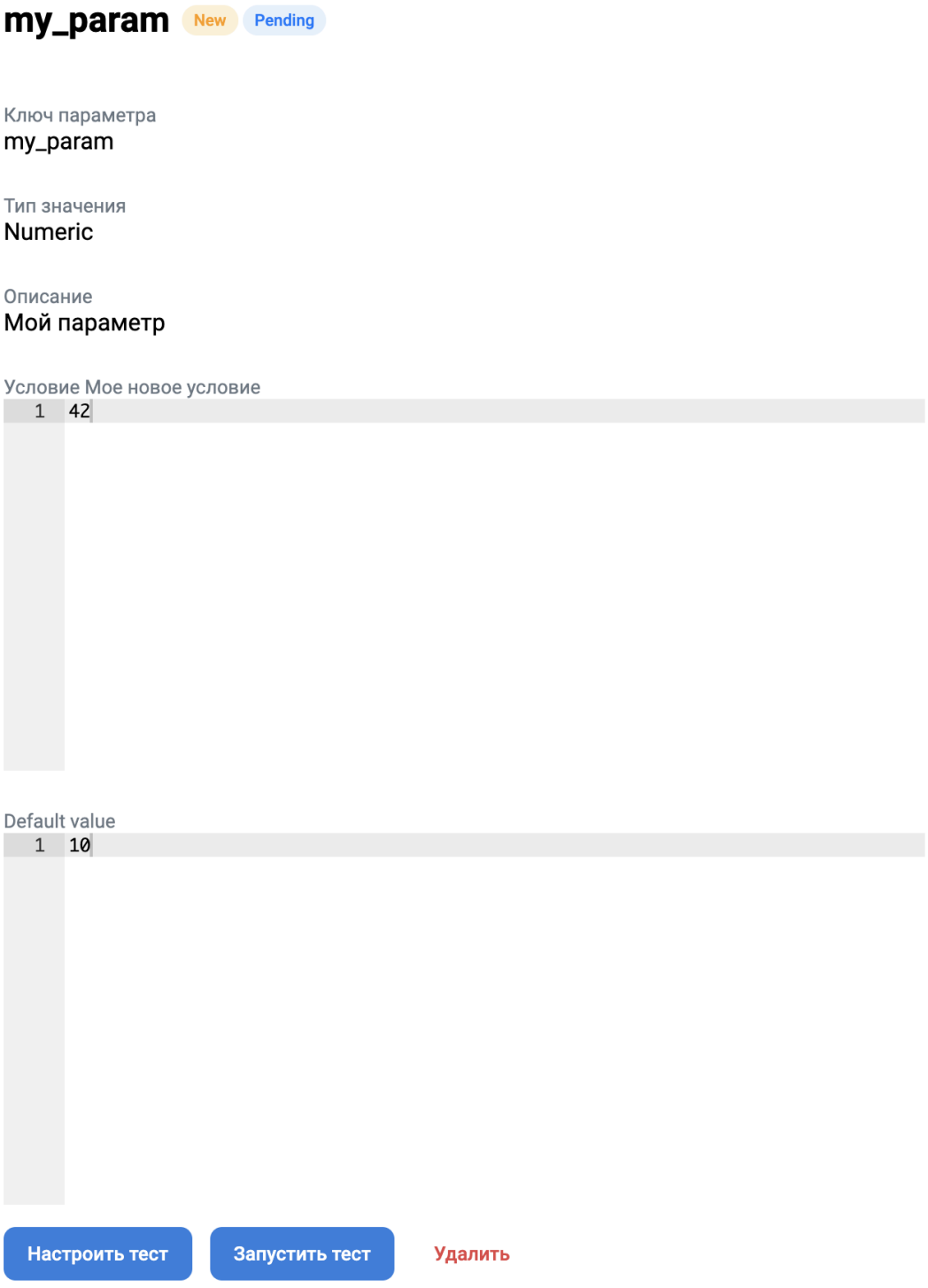
通过拒绝更改，您可以在犯错时调整条件。



处理参数

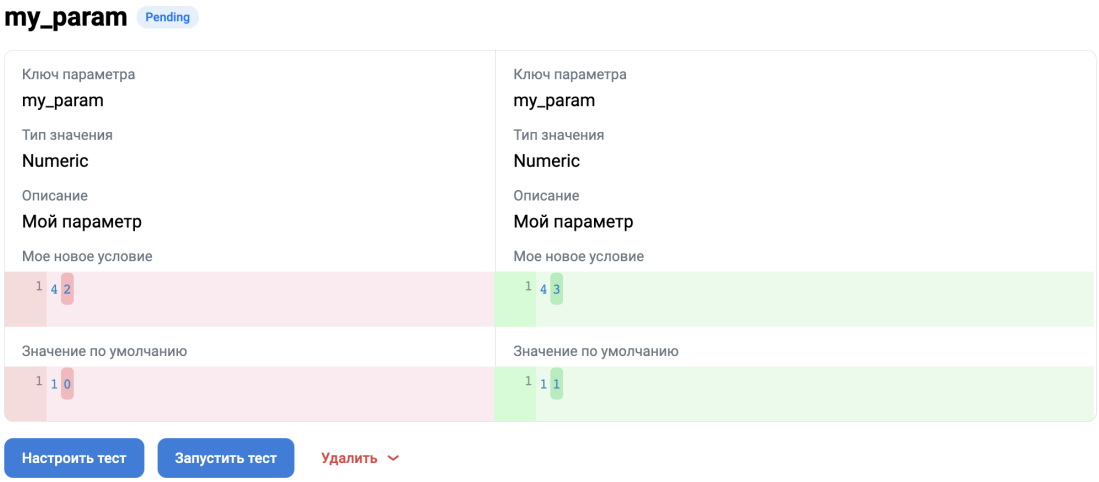
在创建、修改或删除参数时，参数进入待定**Pending**状态，即进入等待确认的模式。但与条件不同，可以在确认更改之前对参数进行测试。测试机制允许将参数应用于特定设备或账户，并确保应用程序的行为正确。

创建参数



在创建参数时，名称旁会出现一个标示为 New 的指示器。这样的参数可以在不确认的情况下删除，因为它是一个草稿。

参数的更改



在更改参数时，可以比较旧版本与新版本，并通过删除草稿来撤销更改。所有用户将接收到原始参数。

参数的删除

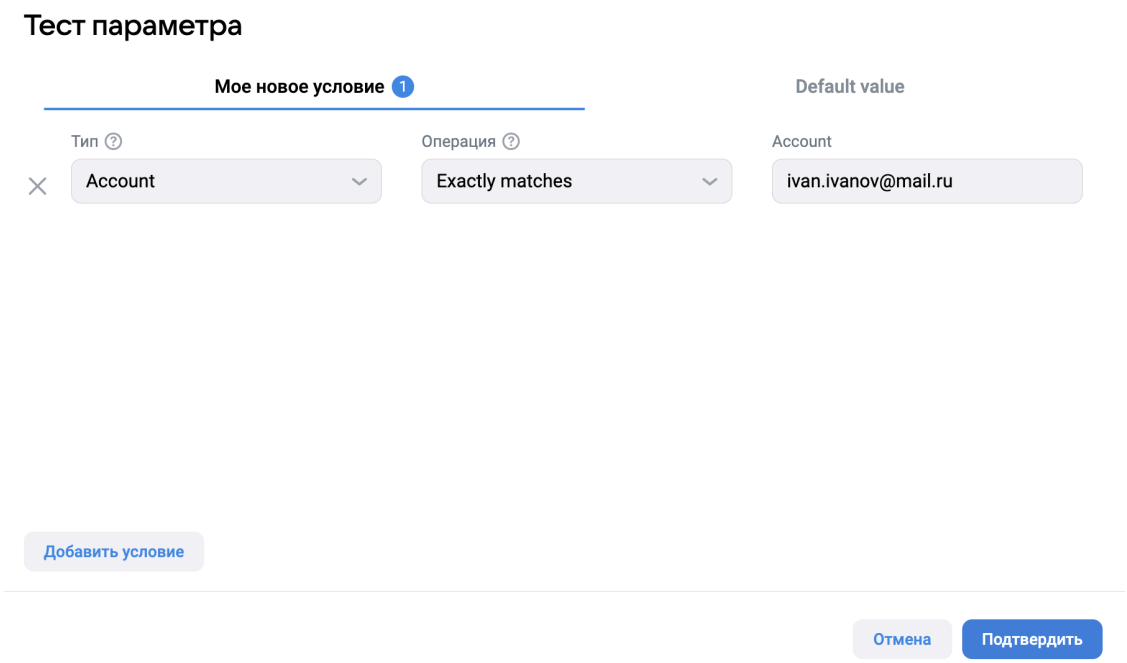


在删除参数时，名称旁会出现一个标示为 `Will be deleted` 的指示器。

测试参数

测试有助于避免应用错误的配置值。测试的基础是将自定义的原子条件应用于参数的每个可能版本。

点击“配置测试”按钮后，会出现如下窗口：



顶部的标签页对应于修改参数中添加的所有条件和默认值。在上面的截图中，测试的是根据“我的新条件”获得的值。

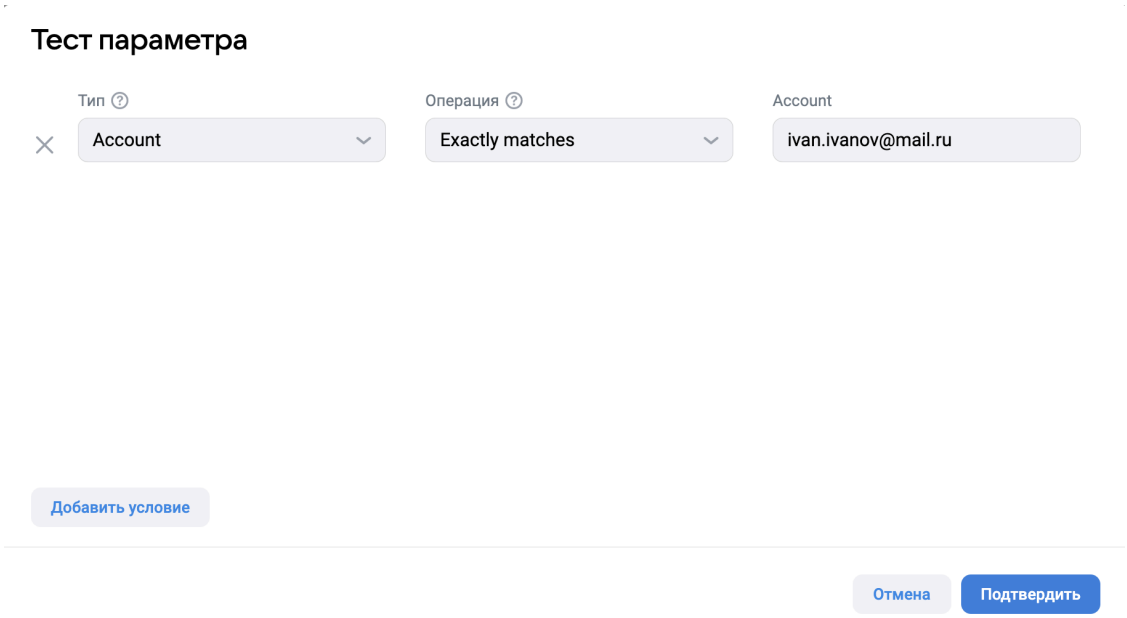
点击“启动测试”按钮后，参数进入 **Testing** 状态，在这个状态下可以确认或拒绝更改。

添加的条件完全替代了参数“我的新条件”中给出的原子条件。这是为了测试那些难以在用户设备上重现的复杂情况而设计的。例如，这可能是与应用程序版本相关的条件。您可以将原始条件更改为自定义条件，并查看在满足参数“我的新条件”时界面的样子。换句话说，服务允许通过自定义条件在设备上获得任何参数值。

第一个条件在选择允许的类型和操作上受限。只有 DeviceID 和 Account 类型以及 Exactly matches 操作是可用的。这样的保护特意设置，以避免在测试中选择过于广泛的用户范围。从第二个条件开始就没有这样的限制。

请注意，同一个 Account 或 DeviceID 只适用于一个条件或默认值。否则，系统将无法确切知道该提供哪个具体值，因此该值可能是任意的并且会不断变化。

删除参数时，标签页不存在，选定的原子条件将全局应用。因为参数正在被删除，我们需要测试其删除，并且在这种情况下，参数内单独条件的具体测试并不重要，因为它们也将与参数一起被删除。实例：



Remote Config服务的许可协议

本Remote Config服务的许可协议（以下简称“协议”）规定了有**限责任公司“VK”**（以下称为许可方）与您（以下称为“许可接受方”）及/或您的代表、员工（以下称为“用户”）使用许可方的RuStore Remote Config服务的关系。

“RuStore Remote Config”服务（以下简称“服务”）旨在远程设置许可接受方应用程序的配置（例如，为许可接受方的应用程序不同用户群体启用和禁用有用的功能，这些功能可以通过使用许可接受方的应用程序和用户设备的参数灵活设置）。

接受本协议，许可接受方表示完全且无条件地同意其所有条款。如果许可接受方不完全接受协议条款，他将无权使用服务并必须立即停止使用。

本协议的现行版本可在互联网上的以下地址获得：https://help.rustore.ru/rustore/for_developers/tools/remote_config/license_agreement。协议的版本自其在上述地址发布之时起生效。

接受规则并/或使用服务的个体，声明并保证其代表许可接受方拥有执行这些行为所需的所有必要权限和授权。

1。本协议使用的术语

1.1 **服务**：为“RuStore Remote Config”软件，它是针对移动应用程序的配置器，旨在实现移动应用程序内不同有用功能、附加组件、特定特性和程序特点的设置。服务托管于许可方的服务器上，许可接受方可通过网站上的Web界面来使用此服务。

1.2 **网站**：位于http://remote-config.rustore.ru/的互联网站点，向许可接受方提供访问服务并使用其功能的权限。

1.3 **许可方**：为有限责任公司“VK”，按照本协议确定的非独占性许可条件向许可接受方提供使用服务的权利。

1.4 **许可接受方**：具有签订本协议所需的行为能力和法律资格的个体，并被授予在本协议规定范围内使用服务的权利。许可接受方是本协议的一方。许可方有权设定个体必须满足的标准，以获得许可接受方的地位。

1.5 **许可协议**：本协议文本、网站上提供的文档以及本协议中指明的其他规则，包含了授予使用服务和网站权利的许可合同的所有必要和重要条款。

2. 加入本协议和服务注册的条件

2.1 在开始使用服务之前，许可接受方必须熟悉本协议。

2.2 许可接受方加入本协议是通过其主动行为（如在“我接受协议条款”的框中勾选或类似内容），根据俄罗斯联邦民法典第435条和第438条的意义，这构成对许可方提议的接受，并且相当于签订许可合同，从而使许可接受方有义务遵守本协议的条款。

2.3 服务中的注册和/或授权是通过使用VK ID工具进行的，该工具是VK生态系统的一部分。VK生态系统是用户、服务和专业工具互动的共同空间，旨在提高用户熟悉服务的使用便利性。

由“VKontakte”有限责任公司提供的VK ID工具（基本国家注册号1079847035179，地址：191024，圣彼得堡，赫尔松街12-14号，楼A，1-N室），使用户能够在服务中进行注册和/或授权，执行创建和维护VK生态系统中用户账户的功能，为用户提供VK生态系统中统一安全的账户数据管理功能，以及在VK生态系统用户账户中连接或断开用户所需服务，以履行与VK生态系统用户的协议。

通过VK ID工具在服务中进行注册和/或授权，意味着许可接受方同意VK生态系统的用户协议，该协议可在互联网上公开获取，网址为：https://id.vk.com/terms，以及VK生态系统的隐私政策，该政策也可在互联网上公开获取，网址为：https://id.vk.com/privacy，本协议和隐私政策。

2.4 许可接受方每次后续使用服务均表示其同意在服务实际使用时生效的本协议条件。

3。协议的主题

3.1 根据本协议的条件，许可方向许可接受方和/或用户（以下统称为“许可接受方”）提供了使用服务的简易（非排他性）许可权，该权利的范围由本协议确定。

3.2 服务的使用权**在非有偿基础上**授予许可接受方，除非许可方另有规定。

3.3 服务和网站的运营和维护，以及向许可接受方提供对服务和网站的访问，均由许可方独家执行。

3.4 许可方是服务及网站上可用文档的唯一权利所有者。

4。使用服务的方法

4.1 许可接受方有权以以下方式使用服务：

4.1.1 使用服务的主要功能在网站上进行许可接受方移动应用程序的远程配置设置。

为了交换移动应用程序中包含的数据，这些数据对于配置设置是必需的，许可接受方将根据位于 https://help.rustore.ru/rustore/for_developers/developer-documentation/remote_config/地址的指南，将许可方的SDK集成到许可接受方的移动应用程序中。

4.2 许可接受方不得：

4.2.1 在没有相应版权所有者许可的情况下，复制、分发、在商业或非商业目的下改编服务中属于许可方或第三方的著作权和相关权利的元素；

4.2.2 除非许可方直接许可，否则不得复制服务和/或网站的设计元素或用户界面；

4.2.3 通过签订次级许可协议或其他方式，将授予许可接受方的服务使用权转让给其他许可接受方或第三方；

4.2.4 使用服务作为产品的基础，该产品包含与服务相同的功能；

4.2.5 违反本协议中规定的服务使用规则；

4.2.6 以本协议未预见的其他方式使用服务。

5。服务使用规则

5.1 要使用服务，许可接受方需要在网站上注册并接受本协议的条款。

5.2 在网站上授权后，用户有权将许可方的SDK集成到许可接受方的移动应用程序中，并在本协议确定的范围内使用服务。

5.3 使用服务的访问权是通过在网站上注册时许可接受方指定的登录名和密码实现的。

5.4 在网站上进行授权的个体被视为本协议的许可接受方。

5.5 许可接受方使用网站时，可能会应用cookies技术以实现许可接受方在网站上的自动授权，以及收集统计数据。

5.6 许可接受方有权通过应用浏览器的相应设置来限制或禁止使用cookies技术。

5.7 如果由于密码丢失或其他原因无法进行授权，许可接受方在履行条件之外，有权通过电子邮件地址support@rustore.ru联系许可方的客户支持服务。访问账户的恢复方法由许可方的支持服务规定。

6。许可方的权利和义务

6.1 许可方负责服务和网站的当前管理，独立决定结构、外观，违反本协议条款时允许或限制许可接受方访问服务和/或网站，并执行其所拥有的其他权利。

6.2 许可方有权：

6.2.1 随时更改服务和/或网站的设计和用户界面、内容、提供的功能内容，更改或补充用于服务和/或网站内的脚本、软件和其他对象，无论是否通知许可接受方；

6.2.2 随时暂停、限制或停止许可接受方访问服务、网站和/或SDK，无需说明理由，无论是否事先通知。

6.2.3 向许可接受方发送信息性消息（包括电子邮件消息）；

6.2.4 为了收集统计数据和识别许可接受方，设置并保存有关许可接受方访问服务和/或网站的IP地址信息，使用存储在许可接受方个人计算机上的技术信息文件（cookies）。

6.3 许可方承诺：

6.3.1 根据本协议所述条件，提供使用服务的权利；

6.3.2 通过在网站上发布信息或向许可接受方电子邮箱发送消息，通知许可接受方本协议条款的变更。

7。许可接受方的权利和义务

7.1 许可接受方有权：

7.1.1 在服务的功能范围内，通过网站进行许可接受方移动应用程序的设置；

7.1.2 进行本协议和适用法律不禁止的其他与使用服务和网站相关的行为。

7.2 许可接受方应：

7.2.1 无任何限制地遵守本协议的条款；

7.2.2 在使用服务时不超过本协议设定的权利限制；

7.2.3 不以其他方式侵犯许可方对服务和网站的知识产权。

7.2.4 遵守许可方的所有指示和规定。如许可接受方未遵守这些指示，许可方有权暂停、限制或终止向许可接受方提供服务的使用权或其部分功能，以及访问网站的权限；

7.2.5 定期查阅互联网上的本协议内容，地址为：https://help.rustore.ru/rustore/for_developers/tools/remote_config/license_agreement，并关注其中所做的任何更改；

7.2.6 遵守并履行本协议规定的所有其他要求和义务。

7.2.7 通知终端用户，许可接受方的移动应用程序是使用服务进行设置的，该服务可能会收集关于他们的信息（包括用于访问应用程序的设备的IP地址、设备标识符和其他信息），这些信息对许可方可用。许可接受方应告知终端用户有关拒绝收集其信息的可能性，并通过在移动应用程序中选择相应的设置来确保此类拒绝。

8。保证和责任限制

8.1 服务及其所有元素按“原样”（as is）提供。这意味着许可方不对使用服务过程中出现的问题承担责任，也不提供服务及其元素适用于特定使用目的的任何保证。许可方不能保证也不承诺使用服务或其元素会带来任何特定结果。

8.2 许可方不对可能的服务和/或网站运行故障和中断以及由此导致的信息丢失承担责任。许可方不对任何因使用服务和/或网站而对许可接受方的计算机、移动设备、任何其他设备或软件造成的损害承担责任。

8.3 许可方不对任何由于使用服务和/或网站而导致的包括损失的利润在内的任何损害或伤害承担责任。

8.4.许可接受方保证，将采取适当措施确保其用于网站授权的账户信息（登录名和密码）的保密性，并防止其他人使用这些信息进行授权。

8.5 许可接受方保证，拥有签订本协议所需的所有必要权限。

8.6 许可方无义务向许可接受方提供任何证据、文件等，证明许可接受方违反了协议条款，导致其被拒绝提供服务和/或网站的访问权，或者此类访问权被终止和/或限制。

9。协议的适用领域和有效期

9.1 许可接受方有权在本协议规定的范围内，不受地域限制地使用服务和网站。

9.2 使用服务和网站的权利被授予许可接受方，为无限期，直至协议被许可方或许可接受方终止。

9.3 本协议在以下情况下终止其效力：

9.3.1 许可方决定更改本协议的条款，需要与许可接受方签订新协议，终止对许可接受方的本协议，停止管理和维护服务，停止访问服务或停止许可接受方使用服务和网站。

9.3.2 许可接受方决定停止使用服务和网站，通过在网站上通过support@rustore.ru的技术支持服务删除账户来实现。

9.4 许可方有权在任何时候无需通知许可接受方且无需说明理由，暂停许可接受方访问和使用服务的权限，不予以任何成本、损失或根据协议得到的退款补偿，包括在许可接受方任何，包括单次，违反本协议条件的情况下；自此时起，许可接受方使用服务的权利被视为终止。

9.5 许可方有权随时终止本协议，无论是否通知许可接受方，此后，许可接受方必须在接收通知之日起不超过5天内完全停止使用服务。

9.6 本协议不包括许可方向许可接受方转让或出让服务和网站任何部分的任何排他权利或授予任何排他性许可。

10。结束条款

10.1 本协议可由许可方单方面更改，无需任何预先通知。许可方单方面所做的协议更改自其在网站上发布的次日生效。许可接受方应自行检查协议的变更。许可接受方未审阅协议和/或更改版协议的行为不能成为其未履行自身义务和未遵守协议设定限制的理由。

10.2 协议的一个或几个条款被法院判定无效，并不意味着整个协议无效。若协议的一个或几个条款被法定程序认定为无效，各方应尽可能按照双方在签署和/或共同更改协议时所预期的方式履行协议义务。

10.3 本协议以及各方因本协议和使用服务而产生的关系，受到俄罗斯联邦法律的管辖。

10.4 关于本协议的形式和签订方式，适用俄罗斯联邦民法典的规范。

10.5 各方关于本协议的所有争议应通过信函和谈判方式解决，并使用强制性的诉前（索赔）程序。如果双方在收到书面索赔后60（六十）个日历日内无法通过谈判达成一致，则任何感兴趣方可将争议提交到许可方所在地的法院审理（排除任何其他法院的管辖）。

10.6 有关履行协议的问题，请联系许可方的注册地址：俄罗斯联邦，125167，莫斯科，列宁格勒大道39号， 79号楼。

版本日期为2023年9月12日。

Remote Config服务的隐私政策

1. 常规规定

1.1。1.1. 本隐私政策（以下简称“政策”），可在互联网上通过以下链接获得：https://help.rustore.ru/rustore/for_developers/tools/remote_config/privacy_policy 是有限责任公司“VK”（基本国家注册号 1027739850962, 俄罗斯，125167，莫斯科，列宁格勒大道39号，79号楼）（以下简称“公司”）的文件，并规定了公司处理和保护信息（以下简称“信息”），包括个人数据，这些信息是公司在您（以下简称“许可接受方”）或您的代表、员工（以下简称“用户”）使用**RuStore Remote Config**服务（以下简称“服务”）时获得的，该服务在互联网上的地址为 http://remote-config.rustore.ru/（以下简称“网站”）。

1.2。1.2. 本政策是许可协议（以下简称“许可协议”）不可分割的一部分，该协议可在互联网上通过以下地址获得：https://help.rustore.ru/rustore/for_developers/tools/remote_config/license_agreement（以下政策和许可协议统称为“规则”）。

接受规则并/或使用服务的个体，声明并保证其代表许可接受方拥有执行这些行为所需的所有必要权限和授权。

1.3。使用服务的条件是许可接受方无条件接受规则，这些个体使用服务和网站。每次访问和/或实际使用服务时，许可接受方均表示同意在服务实际使用时生效的规则版本。政策中使用的一些术语在许可协议中有确定的含义。

1.4.本政策可由公司更改，条件是通过在互联网上发布新版本的文件通知许可接受方，网址为：https://help.rustore.ru/rustore/for_developers/tools/remote_config/privacy_policy。许可接受方应定期自行检查本政策以寻找任何更改。公司所做的政策更改自其在网上发布次日起生效。

1.5。如果许可接受方不同意本政策的条款（包括对政策的更改和补充），则必须立即停止使用服务（包括用户）。

1.6。使用服务时的数据处理是基于俄罗斯联邦的法律。

2。本政策的适用范围

2.1 本政策定义了公司如何收集和使用通过网站和服务获得的信息，该服务被提供给许可接受方使用，以远程配置许可接受方移动应用程序的配置。

2.2。在提供服务的过程中，公司可能会获取以下类别人员的信息：

2.2.1。许可接受方，即接受许可协议条款并获得使用服务权利的个体或法人；

2.2.2。用户，即代表许可接受方或作为许可接受方员工使用服务的个体；

2.2.3 许可接受方移动应用程序的用户（以下简称“终端用户”）；

2.3。使用服务时，许可接受方应合理且诚实地保证：

2.3.1 拥有使用服务所需的所有必要权利；

2.3.2 提供在使用服务所需范围内的真实信息；

2.3.3 已完全熟悉并接受本政策。

3。信息处理目的

3.1。公司处理第2.2点所述个人的信息，以实现许可协议的目的，包括与许可接受方和/或用户进行沟通。

4。被处理信息的构成

4.1。被处理的信息包括账户数据和其他数据（在表1中有定义），公司处理这些信息是为了实现许可协议的目的，具体为：

表1

收集的数据	处理目的
关于许可接受方	
账户数据（个人数据），具体包括：1）许可接受方和/或用户提供的数据（包括使用VK ID进行注册/授权以在服务中创建账户），即：VK ID号码、姓氏、名字、电子邮件地址、许可接受方和/或用户的手机号码。在注册过程中，用户会被分配一个内部ID号，用于在服务中识别用户。2）许可接受方和/或用户在服务中编辑其个人资料页面时提供的额外数据。3）许可接受方和/或用户应公司要求提供的额外数据（例如，许可接受方和/或用	为了实现许可协议，特别是为了提供访问服务的权限，管理和运营服务，确认向服务中的账户申请恢复访问权的申请人的账户所有权，以及在许可协议范围内与许可接受方和/或用户进行沟通。

户在账户中指明的信息，如电子邮件地址、电话号码等）。	
其他数据（非个人数据），具体包括：4）访问服务时获取的技术数据，包括关于技术设备（设备）、与服务的技术互动（如主机的IP地址、许可接受方和/或用户的操作系统类型、浏览器类型、地理位置、互联网服务提供商、访问服务的日期和时间、事件信息（例如系统故障）以及其他类似的网络日志数据。5）使用cookies访问服务时自动获取的信息。6）许可接受方和/或用户在使用服务期间的活动信息，包括服务界面内的点击和跳转。	为了签署和执行许可协议，特别是为了管理和运营服务，以及为了改善服务运行和个性化服务使用。
关于终端用户	
其他数据（非个人数据），具体包括：7）在访问使用服务配置的许可接受方应用程序时获取的技术信息，包括：a）终端用户的设备信息，例如：浏览器类型、设备类型和型号、处理器、系统语言、内存、操作系统版本、屏幕分辨率、时间戳和时区、存储介质、位置、应用程序版本数据、已安装的应用程序、互联网提供商、终端用户访问许可接受方应用程序的日期和时间以及其他类似信息。b）技术标识符信息，用于仅识别计算机、设备、浏览器或应用程序，例如IP地址（也可提供大致位置信息）、User Agent、网络数据（包括设备连接的MAC地址、Wi-Fi和其他网络）、IDFA（广告商标识符）、Android ID（在Android设备中）、Google Advertiser ID、用户自发行的ID。c）与终端用户在应用中的行为相关的参与信息。	为了与许可接受方执行协议，特别是为了提供服务的功能。

4.2。4.2. 通过使用VK ID工具在服务中进行注册和/或授权的许可接受方和/或用户，同意VK生态系统的隐私政策，该政策可通过链接 https://id.vk.com/privacy 获取。

在服务中进行注册和/或授权时，会收集个人数据以及其他非个人数据，这包括由许可接受方和/或用户填写注册表或使用其他可用方法完成，其中包括使用VK ID工具或其他第三方服务，以及随后由许可接受方和/或用户编辑之前提供的信息或自行添加个人数据（如适用）使用服务工具。

在许可接受方和/或用户使用VK ID工具时，其个人数据以及其他非个人数据可能被传输给第三方（VK生态系统的服务和使用VK ID工具的其他服务，在许可接受方和/或用户使用这些服

务和/或工具的过程中），同时公司也可能从这些第三方获得个人数据以及其他非个人数据，其范围在VK生态系统用户的个人账户中有所指定，以实现与VK生态系统用户的协议。

5。处理账户及其他数据

5.1. 处理账户及其他数据时遵循以下原则：

a) 处理个人数据的目的和方法的合法性；

b) 诚信；

c) 数据处理目的与数据收集时预先定义和声明的目的的一致性；

d) 处理的账户和其他数据的范围和性质，其处理方法与账户和其他数据处理目的的一致性；

e) 禁止合并为彼此不兼容目的创建的数据库，这些数据库包含账户和其他数据。

5.2. 收集账户和其他数据

在服务中注册许可接受方和/或用户时，以及许可接受方随后使用服务工具自行提供额外信息时，会进行账户数据的收集。其他数据的收集发生在许可接受方使用服务以及终端用户使用许可接受方应用程序的过程中。

公司负责收集，并进一步记录、系统化、积累、存储、精确化（更新、修改）、提取作为俄罗斯联邦公民个人数据的账户数据，这些数据使用位于俄罗斯联邦境内的数据库。

5.3. 存储和使用账户及其他数据

账户和其他数据在俄罗斯联邦境内处理和存储，存储仅在电子媒介上进行，处理则采用自动化手段，除非法律要求对账户和其他数据进行非自动化处理。

5.4. 转移账户及其他数据

除非适用规则直接规定，否则账户和其他数据不向任何第三方转移。根据政策规定的情况，向公司的第三方合同方转移账户和其他数据时，这些合同方须承诺保证所获信息的保密性。

根据政府机构（地方自治机构）的要求提供个人数据，须在俄罗斯联邦法律规定的情况和程序下进行。

5.5. 终止处理账户及其他数据

当达到处理账户及其他数据的目的时，公司将按照《关于个人数据的联邦法律》规定的方法停止处理账户及其他数据。

5.6. 处理账户及其他数据的方法

5.6.1. 为了履行许可协议（包括提供和改进服务），许可接受方同意并委托公司在遵守适用法律的前提下处理（包括收集、记录、系统化、积累、存储、精确化（更新、修改）、匹配、提取、使用、匿名化、封锁、删除和销毁）许可接受方和/或用户的账户及其他数据，包括他们自动化处理的结果，例如以整数和/或文本值及标识符的形式。

5.6.2。为了使用服务（执行许可协议），许可接受方委托公司处理（包括收集、从许可接受方和公司合作伙伴处获取、记录、系统化、积累、存储、精确化（更新、修改）、匹配、提取、使用、匿名化、封锁、删除和销毁）终端用户的其他数据，这些用户使用由服务配置的许可接受方应用程序，包括他们自动化处理的结果，例如以整数和/或文本值及标识符的形式，以及将终端用户的其他数据及/或其处理结果传输给许可接受方。

公司在处理终端用户的其他数据时应按照许可接受方的指示遵守俄罗斯联邦的法律要求。在此类处理过程中，公司必须保持终端用户其他数据的保密性并确保在处理过程中的安全性。

许可接受方向公司保证和担保，根据适用法律，有合法基础来处理公司上述的终端用户和用户（许可接受方的代表、联系人）的数据，并符合政策中指定的条件。

公司要求时，许可接受方应在收到要求之日起5（五）个工作日内向公司发送执行此保证的确认。

6。许可接受方的权利和义务

6.1. 许可接受方有权：

6.1.1. 通过使用登录名和密码访问服务中的账户，自由免费地访问有关自己的信息；

6.1.2. 使用服务工具为自己的信息设定所需的隐私级别（访问信息的条件），考虑到服务的功能（如适用）；

6.1.3. 自行在服务中的账户里对有关自己的信息进行更改和修正，条件是这些更改和修正包含最新的真实信息；

6.1.4. 删除账户中指定的信息；

6.1.5. 如果账户数据不完整、过时、不真实、非法获取或不必要用于声明的处理目的，要求公司对其账户数据进行精确化、封锁或销毁，如果不能自行执行精确化或删除操作；

6.1.6. 根据请求，按照俄罗斯联邦关于个人数据的法律规定，从公司获得与处理其账户数据有关的信息。

7。信息保护措施

7.1. 公司采取技术和组织法律措施，以确保保护许可接受方和/或用户的账户及其他数据，以及终端用户免受非法或偶然的访问、销毁、修改、封锁、复制、传播，以及其他非法行为。

7.2. 为了授权访问服务，使用许可接受方和/或用户的登录名（电子邮件地址或手机号码）和密码。保管此信息的责任由许可接受方和/或用户承担。许可接受方和/或用户不得将自己的登录名和密码转让给第三方，并且必须采取措施确保其保密性。

7.3。如果许可接受方和/或用户忘记了其登录数据，可以联系VK ID工具的支持服务以恢复其账户和访问服务。

8。许可接受方的申请

8.1。在许可接受方和/或用户使用服务的情况下，关于公司处理的许可接受方和/或用户的账户和其他数据，包括个人数据的信息，将根据俄罗斯联邦法律规定的情况在许可接受方和/或用户的申请（请求）时提供。

8.2. 请求应以书面形式发送至公司的注册地址。

8.3. 在许可协议终止的情况下，公司应停止信息的处理，并销毁相应的个人数据和其他信息。许可接受方有权单方面终止许可协议，并须以书面形式通知公司。

尽管有上述规定，如果根据俄罗斯联邦法律有合法基础，公司有权继续处理信息。

版本日期为2023年9月12日。

Tracer

服务 Tracer 收集并分析错误，然后向开发者自动发送关于它们的报告。这有助于更快更有效地修复错误。该服务在 iOS 和 Android 的移动应用程序中运行。

使用 Tracer，您可以：

- 在生产环境中进行性能分析，以确定卡顿和冻结的原因；
- 在用户的设备上自动或半自动地查找内存泄漏的原因；
- 发现磁盘上被遗忘和遗漏的应用程序文件 – 这样可以减少应用数据清理数据后应用程序被意外删除或用户被登出的风险。

Android

工作开始

快速启动

注册和设置

首先你需要：

1. 登录 Tracer 账户。
2. 创建或加入组织。
3. 添加 Android 项目（需要是组织的管理员或所有者）。

将追踪器的依赖项添加到项目中

在您的 <project>/settings.gradle

```
pluginManagement {     repositories {         maven { url
'https://artifactory-external.vkpartner.ru/artifactory/maven/' }     }}
dependencyResolutionManagement {      repositories {          maven { url
'https://artifactory-external.vkpartner.ru/artifactory/maven/' }      }}<br>
```

在您的 <project>/<app-module>/build.gradle

```
plugins {        ``id   'ru.ok.tracer' version   '0.2.7'}  tracer
{        ``defaultConfig {                  ``// 参见“设置”部分
                ``pluginToken =   "PLUGIN_TOKEN"                ``appToken =
"APP_TOKEN"                ``// 包括为构建加载映射。默认启用
                ``uploadMapping =   true        ``}        ``// 也可以为每个 flavor,
buildType, buildVariant设置配置.        ``// 配置继承自 defaultConfig.        ``debug
{                ``// 参数...        ``}        ``demoDebug {                ``// 参
数...        ``}}  dependencies {        ``// 插件相互独立。可以只连接,        ``// 当
前所需的.        ``// 收集和分析崩溃及 ANR        ``implementation
"ru.ok.tracer:tracer-crash-report:0.2.7"        ``// 收集和分析原生崩溃
        ``implementation   "ru.ok.tracer:tracer-crash-report-native:0.2.7"        ``//
收集和分析OOM时的堆转储        ``implementation
"ru.ok.tracer:tracer-heap-dumps:0.2.7"        ``// 分析设备的磁盘空间使用
        ``implementation   "ru.ok.tracer:tracer-disk-usage:0.2.7"        ``// 采样分析
器        ``implementation
"ru.ok.tracer:tracer-profiler-sampling:0.2.7"          ``// 系统追踪``implementation
"ru.ok.tracer:tracer-profiler-systrace:0.2.7"}<br>
```

启用追踪器插件及其在项目中的配置

在您的 Application.kt 中，实现 HasTracerConfiguration 接口。

```
class MyApplication : Application(), HasTracerConfiguration {        ``override val
tracerConfiguration: List<TracerConfiguration>                ``get() =
listOf(                        ``CoreTracerConfiguration.build
{                                ``// 追踪器核心选项
                        ``},                                ``CrashReportConfiguration.build
{                                ``// 崩溃报告选项
                        ``},                                ``CrashFreeConfiguration.build
{                                ``// crash free统计选项
                        ``},                                ``HeapDumpConfiguration.build
{                                ``// ＯＯＭ时堆转储选项
                        ``},                                ``DiskUsageConfiguration.build
{                                ``// 磁盘使用分析选项
                        ``},                                ``SystraceProfilerConfiguration.bu
ild {                                ``// 生产环境中的systrace分析器选项
                        ``},                                ``SamplingProfilerConfiguration.bu
ild {                                ``// 采样分析器选项``},                ``)}
```

HasTracerConfiguration.tracerConfiguration属性将在进程启动时仅被请求一次，这发生在Application.onCreate被调用之前但在Application.attachBaseContext之后。在getter中，您已经可以访问应用程序的上下文，但还为时过早访问在onCreate中初始化的内容。

各选项的详细描述

- CoreTracerConfiguration — 请参阅本页下方；

- CrashReportConfiguration 和 CrashFreeConfiguration — 请参阅“Crash 和 ANR”页面；
- HeapDumpConfiguration — 请参阅“Heap Dumps”页面；
- DiskUsageConfiguration — 请参阅“Disk Usage”页面；
- SystraceProfilerConfiguration — 请参阅“Systrace Profiler”页面；
- SamplingProfilerConfiguration — 请参阅“Sampling Profiler”页面。

CoreTracerConfiguration 描述

在罕见的情况下，当您想要在项目中配置追踪器核心时，需要从 HasTracerConfiguration 返回 CoreTracerConfiguration

```
class MyApplication : Application(), HasTracerConfiguration {        ``override val
tracerConfiguration: List<TracerConfiguration>                ``get() =
listOf(                          ``CoreTracerConfiguration.build
{                                ``// 您的选项``},                ``)}
```

CoreTracerConfiguration.Builder选项：

- setEnabled — 该选项不再使用，将在0.3.x版本中移除。追踪器核心始终启用，但在没有启用的插件时不处于活动状态；
- setHost, provideHost — 更改追踪器地址；
- setStatHost, provideStatHost — 更改crash free功能的追踪器地址；
- setCustomAppKey, provideCustomAppKey — 替换gradle插件配置中的sampleUploadToken。

迁移到新版本

从0.2.3迁移到0.2.7

Condition类和SystraceProfiler.start(context, profileDuration, activeCondition)以及SamplingProfiler.run(context, duration, condition)方法已被标记为过时，并将很快被移除。请使用SystraceProfiler和SamplingProfiler API进行手动分析。

在使用TracerEvents.addEvent()标记interestingEvents以进行分析的地方，现在应该根据您使用的分析类型，调用SystraceProfiler.commit()或SamplingProfiler.commit()。而在可以到达这些interestingEvents的地方，则应该分别调用SystraceProfiler.start()或SamplingProfiler.start()。

从0.2.2迁移到0.2.3

TracerCrashReport.log(Throwable) 方法用于发送 non-fatal，已被标记为过时，并将很快被移除。请使用 TracerCrashReport.report(Throwable) 方法代替。这不适用于 TracerCrashReport.log(String) 方法，该方法用于向 crash 和 non-fatal 事件添加日志，它将继续像以前一样工作。具体使用示例请参见“Crash и ANR”页面。

从0.2.3版本开始，tracer-plugin的连接方式发生变化。依赖项从 ru.ok.tracer:plugin:0.1.1 更改为 ru.ok.tracer:tracer-plugin:0.2.3（注意，不仅版本发生了变化）。Plugin id 从 ru.ok.tracer.mapping_plugin 更改为 ru.ok.tracer。未来插件版本和 runtime 依赖将保持一致，并将一起发布。最新信息请见“快速启动”页面。

从0.1.15迁移到0.2.2

Tracer.configure和Tracer.configureAsync方法。你应该在你的Application类中实现HasTracerConfiguration接口，并返回之前传递给Tracer.configure的内容。详细信息请参见“快速启动”页面。

现在，追踪器的配置将在启动时而不是下一次启动时进行。

Tracer模块

Crash 和 ANR

集成依赖项到项目中

在您的 <project>/<app-module>/build.gradle

```
dependencies {    implementation "ru.ok.tracer:tracer-crash-report:0.2.7"}
```

Tracer 还支持收集和分析原生崩溃。如果您想要收集发生在原生代码中的崩溃，需要连接相应的依赖项：

```
dependencies {    ``implementation
"ru.ok.tracer:tracer-crash-report-native:0.2.7"}
```

目前，这个功能正在积极开发中。崩溃的收集已经可以工作，但目前显示的是所有原生崩溃，暂时没有堆栈跟踪和其他详细信息。您现在就可以收集和计算它们，而显示和分类正在完善中。

更多关于依赖项的详细说明，请参见“快速启动”页面。

CrashReportConfiguration 和 CrashFreeConfiguration 描述

在您的 Application.kt

```
class MyApplication : Application(), HasTracerConfiguration {        ``override val
tracerConfiguration: List<TracerConfiguration>                ``get() =
listOf(                        ``CrashReportConfiguration.build
{                              ``// 您的选项
                       ``},                           ``CrashFreeConfiguration.build
{                              ``// 您的选项``},                ``)}
```

CrashReportConfiguration.Builder 选项：

- setEnabled — 启用/禁用崩溃报告。默认情况下启用，可以保持不变））
- setSendAnr — 禁用发送应用程序无响应 (ANR)。默认情况下启用
- setNativeEnabled — 启用/禁用原生崩溃报告。默认值取决于是否连接了 tracer-crash-report-native **依赖项。如果连接了，那么默认情况下崩溃报告也是启用的。如果未连接，则禁用。⚠ 未连接相应依赖项时无法启用原生崩溃报告，但如果存在依赖项，可以禁用崩溃报告。**

CrashFreeConfiguration.Builder 选项：

- setEnabled — 启用/禁用 crash free 用户的计算。默认情况下禁用！

CrashReportConfiguration.Builder 的过时或危险选项：

- setHost — 已过时，将在0.3.x版本中移除。如果您确实需要更改主机，请使用 CoreTracerConfiguration.setHost
- setSendLogs — 已过时且无效，将在0.3.x版本中移除
- setCountCrashFreeUsers — 已过时且无效，将在0.3.x版本中移除。替代此选项的是 CrashFreeConfiguration.Builder.setEnabled
- setSendThreadsDump、setSendAsap、setMaxNonFatalExceptions — 已过时且无效，将在0.3.x版本中移除

CrashFreeConfiguration.Builder 的过时或危险选项：

- setExperimentalMaxSessionsToUpload — 需要累积多少会话以发送批量数据。默认为10。仅用于测试！
- setExperimentalMaxSessionTimeSpanToUpload — 累积会话的时长以发送批量数据。默认为4小时（以毫秒计）。仅用于测试！
- setExperimentalUploadSessionsFromYesterday — 如果发现昨天（或更早）的会话，发送批量数据。默认情况下启用

TracerCrashReport 描述

使用 TracerCrashReport.report(throwable) 方法发送 non-fatals。

```
// 记录非致命错误.  TracerCrashReport.report(NonFatalException("I'll be ok soon"))
```

崩溃总是按堆栈跟踪的共同部分分组。默认情况下，non-fatals 也是如此，但有方法可以影响这种分组。

为了让所有 non-fatals 分到一个组，不论堆栈跟踪如何，可以使用带有 issueKey 参数的 report 方法。

```
// 记录带有 ISSUE- 键的非致命错误001  TracerCrashReport.report(NonFatalException("What a terrible failure"),  issueKey  =  "ISSUE-001")<br>
```

目前 Tracer 的限制是每天 100 万个事件。因此建议不要滥用此方法。

事件也可以添加额外信息。

Heap Dumps

集成依赖项到项目中

在您的 <project>/<app-module>/build.gradle

```
dependencies {    ``implementation "ru.ok.tracer:tracer-heap-dumps:0.2.7"}
```

更多关于依赖项的详细说明，请参见“快速启动”页面。

HeapDumpConfiguration 描述

在您的 Application.kt

```
class MyApplication : Application(), HasTracerConfiguration {        ``override val
tracerConfiguration: List<TracerConfiguration>                ``get() =
listOf(                          ``HeapDumpConfiguration.build
{                                ``// 您的选项``},                ``)}
```

HeapDumpConfiguration.Builder 选项：

- setEnabled — 启用/禁用在 OOM 时收集堆转储。默认为启用状态。

HeapDumpConfiguration.Builder 的过时或危险选项：

- setProbability — 已过时且无效，将在0.3.x版本中移除。
- setInterestingSize — 已过时且无效，将在0.3.x版本中移除

堆转储会在夜间发送，当用户不使用设备时。

Disk Usage

集成依赖项到项目中

在您的 <project>/<app-module>/build.gradle

```
dependencies {    ``implementation "ru.ok.tracer:tracer-disk-usage:0.2.7"}
```

更多关于依赖项的详细说明，请参见“快速启动”页面。

DiskUsageConfiguration 描述

在您的 Application.kt

```
class MyApplication : Application(), HasTracerConfiguration {        ``override val
tracerConfiguration: List<TracerConfiguration>                ``get() =
listOf(                          ``DiskUsageConfiguration.build
{                                ``// 您的选项``},                ``)}
```

DiskUsageConfiguration.Builder 选项：

- setEnabled — 启用/禁用插件。默认情况下启用；
- setProbability — 每天该用户的设备在后台检查磁盘空间使用的概率（1/n）。默认为0，等同于启用插件；
- setInterestingSize — 占用空间的限制，超过此限制，SDK将发出警告并向 Tracer 发送报告。以字节为单位。默认为10Gb；
- setExcludePath — 应该从检查中排除的已知大文件的路径。只接受通过 GlobalDirs 形成的路径。

GlobalDirs 描述：

- GlobalDirs.INTERNAL_DATA.excludePath('foo/bar') — 应用的内部存储；
- GlobalDirs.EXTERNAL_DATA.excludePath('foo/bar') — SD卡上的应用文件；
- GlobalDirs.SRC.excludePath('foo/bar') — 应用程序本身的文件。

Systrace Profiler

集成依赖项到项目中

在您的 <project>/<app-module>/build.gradle

```
dependencies {    ``implementation
"ru.ok.tracer:tracer-profiler-systrace:0.2.7"}
```

更多关于依赖项的详细说明，请参见“快速启动”页面。

SystraceProfilerConfiguration 描述

在您的 Application.kt

```
class MyApplication : Application(), HasTracerConfiguration {        ``override val
tracerConfiguration: List<TracerConfiguration>                ``get() =
listOf(                          ``SystraceProfilerConfiguration.build
{                                ``// 您的选项``},                ``)}
```

SystraceProfilerConfiguration.Builder 选项：

- setEnabled — 启用/禁用性能分析。默认为启用状态。

SystraceProfilerConfiguration.Builder 的过时或危险选项：

- setDurationMs — 分析器的运行时间，以毫秒为单位；
- addCondition — 添加 Condition 以开始性能分析。

Condition, TracerEvents, SystraceProfiler API 等的描述。

具体描述请见“采样分析器”部分。

Sampling Profiler

集成依赖项到项目中

在您的 <project>/<app-module>/build.gradle

```
dependencies {    ``implementation "ru.ok.tracer:tracer-profiler-sampling:0.2.7"}
```

更多关于依赖项的详细说明，请参见“快速启动”页面。

SamplingProfilerConfiguration 描述

在您的 Application.kt

```
class MyApplication : Application(), HasTracerConfiguration {        ``override val tracerConfiguration: List<TracerConfiguration>                ``get() = listOf(                        ``SamplingProfilerConfiguration.build {                            ``// 您的选项``},                ``)}
```

SamplingProfilerConfiguration.Builder 选项：

- setEnabled — 启用/禁用性能分析。默认为启用状态。

SamplingProfilerConfiguration.Builder 的过时或危险选项：

- setBufferSizeMb — 请参阅 android.os.Debug.startMethodTracingSampling 的描述；
- setSamplingIntervalUs — 请参阅 android.os.Debug.startMethodTracingSampling 的描述。默认值为5000；
- setDurationMs — 分析器的运行时间，以毫秒为单位；
- addCondition — 添加 Condition 以开始性能分析。

Condition 描述。Deprecated。

Condition 结构用于控制性能分析的启动和结果发送。使用示例：

```
Condition.appStart(10_000, 7_000)<br>
```

以100%的概率在应用启动时启动分析器，并且如果应用启动时间超过7000毫秒，将其工作结果发送到服务器。

创建自定义事件：

```
val condition = Condition.build { // ваши
опции }
```

Condition.Builder 选项

- setTag('my_tag') — 将结果上传到 Tracer 的标签。
- setTagLimit(n) — 服务器每天接受该标签的最大报告数量。
- setProbability(n) — 发生 startEvent 事件时，启动分析器的概率（1/n）。
- setStartEvent('my_event') — 发生上述事件时，将启动分析器。
- setInterestingEvent('my_other_event') — 可选。如果存在，性能分析结果将仅在遇到此事件时发送到后端。
- setInterestingDuration(n) — 如果开始事件和感兴趣事件之间的时间超过此数值，性能分析结果将被发送到后端。

如果感兴趣的事件有自己的计数器，比较将与它进行。例如，app_freeze 有一个计数器 — UI 线程挂起的时间。因此，如果 interestingEvent == 'app_freeze' 并且 interestingDuration == 700，那么如果在分析器运行期间发生了超过 700 毫秒的冻结，则会发生发送。

手动性能分析

还可以在代码中标记性能分析的开始和结束。

SamplingProfiler.start() — 启动分析器。接受参数：

- context: Context — 应用程序的 App context。
- tag: String — 结果将以此标签上传到追踪器。
- duration: Long — 分析器运行时间，以毫秒为单位。

SamplingProfiler.abort() — 停止分析器并清除结果。

SamplingProfiler.commit() — 停止分析器并将结果发送到后端。如果在调用时分析器尚未完成，那么最终标签将等于 <tag><tagSuffix>

- tagSuffix: String — 可选。如果分析器提前停止，将添加到标签的后缀。

实例：

"以1/100000的概率，分析器将开始其工作

```
if (Random.nextInt( 100 _ 0 0 0 ) ==   0 )
{                ``SamplingProfiler.start(                                ``context =
appContext,                                ``tag =   "stream_request"
,                                ``duration =   10 _ 0 0 0 ,                ``)}//
... 代码。例如，加载信息流SamplingProfiler.commit( "loaded" )<br>
```

“系统”事件描述

“系统”事件在 Condition 类和手动性能分析中使用：

- TracerEvents.EVENT_APP_START_BEGIN — 'app_start_begin' — 应用程序启动开始。
- TracerEvents.EVENT_APP_START_END — 'app_start_end' — Application.onCreate() 方法结束。
- TracerEvents.EVENT_FIRST_ACTIVITY_CREATED — 'app_first_activity_created' — 创建了第一个活动。
- TracerEvents.EVENT_ACTIVITY_CREATED — “activity_created” — 任何活动进入 created 状态。
- TracerEvents.EVENT_FREEZE — “app_freeze” — UI 线程挂起了 N 毫秒。如果该事件被用作 interesting event，那么 interesting duration 将与 N 进行比较。例如，如果 N == 500，那么如果在分析器运行期间 UI 线程挂起超过 500 毫秒，分析结果将被发送。
- EVENT_ANR — "app_anr" — UI 线程挂起并未在分析器结束时恢复。从发现挂起到分析器结束，过去了 N 毫秒。如果该事件被用作 interesting event，那么 interesting duration 将与 N 进行比较。例如，如果 N == 5000，那么如果在分析器运行期间 UI 线程挂起超过 5000 毫秒，分析结果将被发送。

添加自定义事件：

```
TracerEvents.addEvent(eventName, duration)
```

- eventName — 事件名称。在 Condition 类的 startEvent 和 interestingEvent 方法中使用。
- duration — 与此事件相关联的时间。如果设置，interestingDuration 不是基于 startEvent 和 interestingEvent 之间的差异计算，而是与此事件的 duration 比较。

附加功能

向事件添加信息

HasTracerSystemInfo 描述

Tracer SDK 提供了关于崩溃的最重要信息，您无需为此担心：

- date — 事件发生的日期和时间；
- board — Build.BOARD;
- brand — Build.BRAND;
- cpuABI — Build.SUPPORTED_ABIS;
- device — Build.DEVICE;
- display — Build.DISPLAY;
- manufacturer — Build.MANUFACTURER;

- model — Build.MODEL;
- osVersionSdkInt — Build.VERSION.SDK_INT;
- osVersionRelease — Build.VERSION.RELEASE;
- cpuCount — Runtime.getRuntime().availableProcessors();
- operatorName — 运营商名称（如果可用）。

您可以在应用中实现 HasTracerSystemInfo 来添加额外的崩溃信息：

```
class MyApplication : Application(), HasTracerSystemInfo {    ``override val tracerSystemInfo: Map<String, String>        ``get() = mapOf{            ``"foo" to "bar",        ``}}
```

例如，userId、build flavor 或 vendor 可以是有用的信息，特别是如果构建是为了预安装。

每次发生 CRASH 和 NON-FATAL 事件时都会请求 HasTracerSystemInfo.tracerSystemInfo。它也会在应用启动时被调用一次，以防发生 ANR。

TracerCrashReport.log 描述

事件发生时，也会附带之前的日志。您可以使用 TracerCrashReport.log 方法添加日志：

```
// 当某个事件发生时TracerCrashReport.log( "button 100500 clicked" )// 经过  100500 行之后TracerCrashReport.report(NoSuchElementException( "Nothing here, see logs" ))// 在这个 non-fatal 的 logs 标签页中，您会看到一个行 event button  100500 clicked
```

日志不会无限积累。目前，它们被收集在一个大小为 64KB 的循环缓冲区中。

Tracer 的自定义日志记录

这些信息涉及 Tracer SDK 的自定义日志记录。它与 SDK 发送到服务器的内容无关。自定义日志记录仅用于 Tracer SDK 本身及其在您项目中的集成。

HasTracerLogger 和 TracerLoggerDelegate 描述

追踪器通过 TracerLoggerDelegate 接口进行所有内部日志记录。要更换日志记录机制，请在您的应用中实现 HasTracerLogger。

如果您只想更改日志记录级别，可以创建 DefaultTracerLoggerDelegate 实例：

```
class MyApplication : Application, HasTracerLogger {    ``override val tracerLoggerDelegate: TracerLoggerDelegate        ``get() =
```

```
DefaultTracerLoggerDelegate(            ``minLogPriority =
Log.VERBOSE,        ``)}
```

您可以在自己的 TracerLoggerDelegate 实现中完全重写日志记录机制：

```
class MyApplication : Application, HasTracerLogger {        ``override val
tracerLoggerDelegate: TracerLoggerDelegate                ``get() =
MyLoggerDelegate()}class MyLoggerDelegate : TracerLoggerDelegate {        ``// 从哪个
级别开始记录事件。使用 android.util.Log 的常量        ``// 默认为
Log.DEBUG        ``override val minLogPriority: Int                ``get() =
Log.VERBOSE                ``// 主要的日志记录方法        ``//        ``// priority -
日志级别。使用 android.util.Log 的常量        ``// msg - 可选的信息.        ``// tr -
可选的异常.        ``override fun println(priority: Int, msg: String?, tr: Throwable?)
{                ``// 这里是您的日志记录代码
                ``TODO()                ``// 可选择转发到内置实现                ``//
将以 ru.ok.tracer 标签发送到 android
logcat``DefaultTracerLoggerDelegate.println(priority, msg, tr)        ``}}
```

开发者统计数据

在RuStore 控制台中有几个统计部分，您可以在其中查看以下数据：

- 应用程序的浏览量和下载量；
- 获得的收入总额和退款；
- 发送给用户的推送通知数量等。

这些统计数据将帮助您收集必要的数据，以提高应用程序的功能性，使其对用户更加友好，评估销售动态，并在必要时更改您的应用程序的货币化策略。

版本下载统计

在RuStore 控制台中，您可以查看应用程序的各种统计信息。

只有经过授权的用户才能查看统计数据。

如何查看版本下载统计？

1. 打开RuStore 控制台。
2. 转到“应用程序”选项卡。
3. 在侧边菜单中点击“应用程序统计”。

在统计选项卡中显示以下元素：

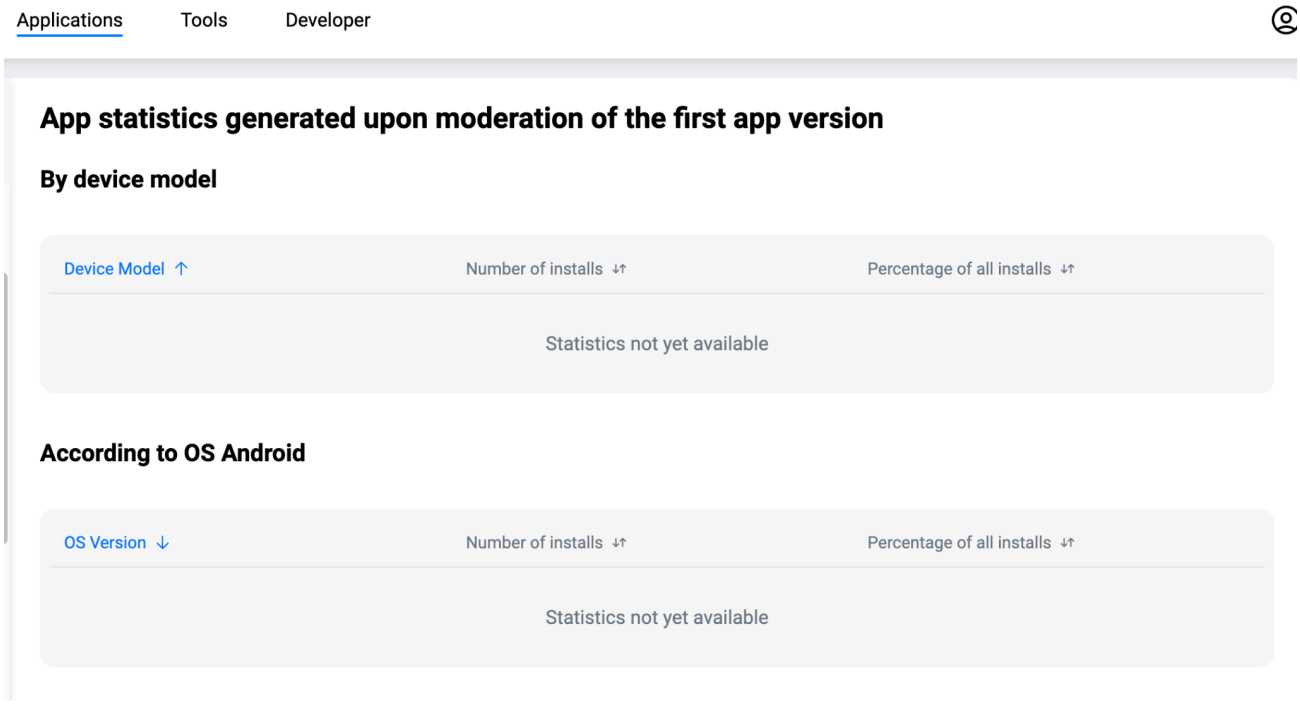
- 包含以下内容的应用程序总体统计表：

  - 在RuStore界面（Web和移动端）上应用程序的总浏览量；
  - 在RuStore界面上应用程序的总下载量；
- 包含以下内容的用户操作系统（以下简称OS）版本统计表：
  - RuStore用户OS版本名称；
  - 在特定OS版本上应用程序的下载量；
  - 在特定OS版本上应用程序的下载百分比；
- 包含以下内容的RuStore用户设备模型统计表：
  - RuStore用户设备模型名称；
  - 应用程序在各种设备模型上的下载量；
  - 应用程序在各种设备模型上的下载百分比；

如有必要，您可以对操作系统版本和设备型号统计表的元素进行排序，具体为：

- 点击一次排序图标 — 表中的所有元素将按该参数的从大到小的顺序排序；
- 点击排序图标两次 — 表中的所有元素将按该参数的从小到大的顺序排序；
- 点击排序图标三次 — 元素列表将恢复到默认状态。

为了方便排序，可以同时使用多个过滤器。



货币化的统计数据

在 RuStore 控制台中可以查看财务数据，这使您可以评估销售动态，包括特定内容或订阅的销售。收入数据基于交易量的信息。

通用要求

要查看货币化统计数据，需要满足以下条件：

- 应用程序已集成一种货币化方式，并且之前已进行交易。更多详情见货币化部分。
- 用户具有查看统计数据的权限；
- 用户必须在 RuStore 控制台中进行授权。

如何查看支付统计数据？

1. 打开RuStore 控制台。
2. 在顶部菜单中选择“应用程序”。
3. **转到“货币化 → 支付统计数据”选项卡。**

在“支付统计数据”页面上，数据按照所有来源的总收入和所有来源的退款显示。

所谓的收入来源指集成到应用程序的货币化产品类型：

1. 收费应用程序：从 RuStore 展示平台下载应用程序的收入；
2. in-app 产品：应用程序内 in-app 产品交易的收入；
3. 订阅：定期支付使用订阅的收入。

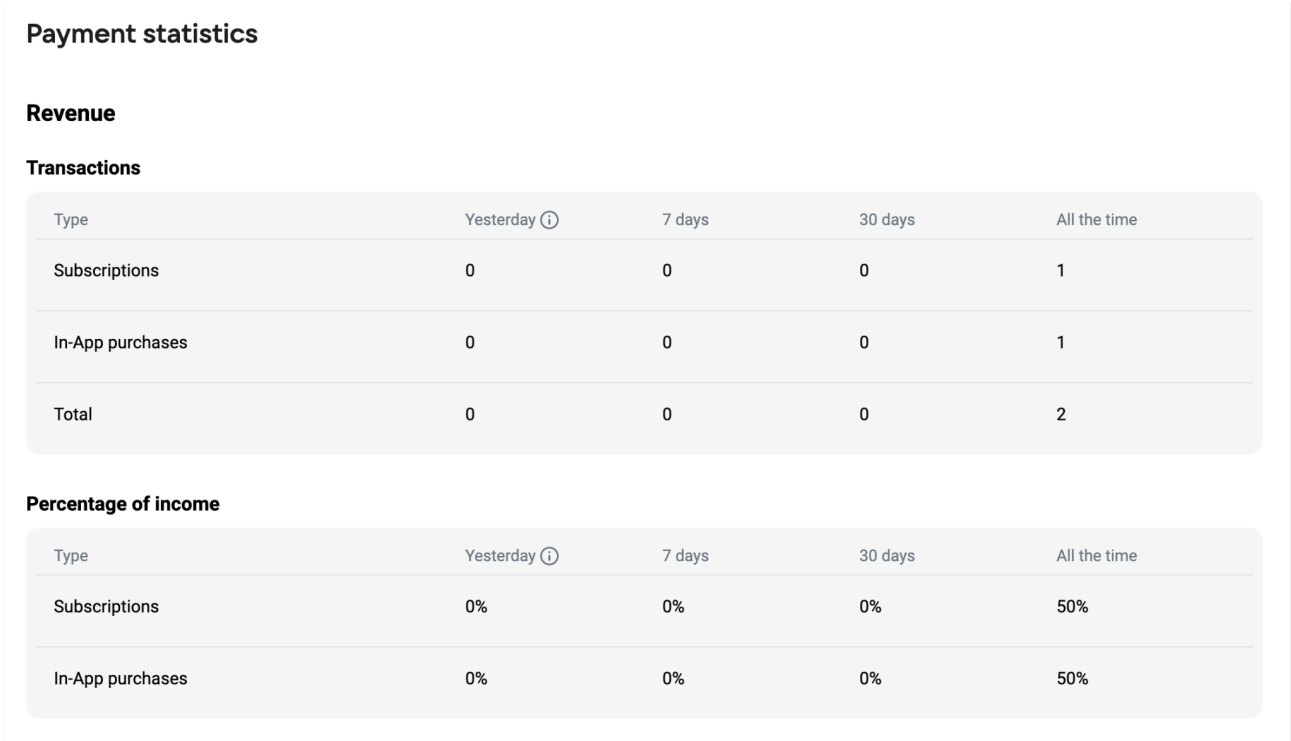
对于退款和收入统计，显示以下类型的数据：

- 金额，包括货币单位；
- 交易数量；
- 收入百分比。

如果公司集成了多于一种的产品类型（例如，订阅 + in-app 支付），则会显示以下信息：

- 退款百分比
- 期间 — 可以查看统计数据的时间段。可用的值包括：
- 过去24小时从00:00到00:00；
- 包括当前日在内的最近7天；
- 包括当前日在内的最近30天；
- 全部时间 = 从获得第一笔交易到包括当前日在内。

1. 报告是在所选时间段结束时生成的。



Payment statistics

Revenue

Transactions

Type	Yesterday	7 days	30 days	All the time
Subscriptions	0	0	0	1
In-App purchases	0	0	0	1
Total	0	0	0	2

Percentage of income

Type	Yesterday	7 days	30 days	All the time
Subscriptions	0%	0%	0%	50%
In-App purchases	0%	0%	0%	50%

Push通知统计

在 RuStore 控制台中，可以查看已发布应用程序的 Push 通知统计数据。

统计是根据设置了 Push 通知的每个应用程序的数据计算的。

如何查看 Push 通知的统计数据？

1. 打开RuStore 控制台。
2. 在顶部菜单中选择“应用程序”。
3. **转到“Push 通知 → Push 通知统计”选项卡。**

Push 通知的统计数据在发布应用程序新版本的第二天显示。

在 Push 通知统计页面上，显示以下数量：

- 发送的所有 Push 通知；
- 送达的 Push 通知；
- 点击 Push 通知进入应用程序的用户。

收集这些数据是为了按时间段查看：每天、每周、每月或所有时间。

APK 签名验证

大多数 SDK 要求应用程序使用正确的签名才能运行。要了解 APK 文件是用哪个签名签署的，需要使用额外的工具来显示签名指纹。

签名信息在 build.gradle 文件中说明。如果没有明确指定签名，将使用默认签名 ~/.android/debug.keystore。

下面介绍了如何知道签名的方法。

使用 Android Studio 获取签名

gradle signingreport

执行命令的结果：

```
Variant: debugAndroidTest
Config: debug
Store: /Users/igor.bobrov/StudioProjects/rustore-push-sdk/cert/sdk-debug.keystore
Alias: androiddebugkey
MD5: A0:C3:A8:EB:38:38:76:57:93:F5:36:7F:B0:92:57:05
SHA1: 76:2E:37:F7:45:06:8F:96:12:8C:D4:21:8A:FA:A6:32:D8:BC:FA:8B
SHA-256: ED:E1:1C:C1:95:0C:69:8F:92:32:5F:FB:A8:A7:C1:15:29:43:D6:F1:9E:A7:14:E2:49:67:03:05:03:95:ED:B8
Valid until: понедельник, 2 июля 2046 г.
```

签名指纹在 SHA-256 字符串中指定。

使用 apksigner 获取签名

apksigner verify –print-certs -v example.apk

example.apk — 您的应用。

执行 Mac OS 命令的结果：

```
artem@MacBook-Pro-7 ~/w/apks> apksigner verify --print-certs ./example.apk
Signer #1 certificate DN: CN=Debug Android, O=Android, C=US
Signer #1 certificate SHA-256 digest: ede11cc1950c698f92325ffba8a7c1152943d6f19ea714e2496703050395edb8
Signer #1 certificate SHA-1 digest: 762e37f745068f96128cd4218afaa632d8bcfa8b
Signer #1 certificate MD5 digest: a0c3a8eb3838765793f5367fb0925705
```

Signer #1 certificate SHA-256 digest 的值就是所需的签名指纹。

使用 keytool 获取签名

keytool -printcert -jarfile example.apk

example.apk — 您的应用。

执行 Mac OS 命令的结果：

```
artem@MacBook-Pro-7 ~/w/apks> keytool -printcert -jarfile ./example.apk
Signer #1:

Certificate #1:
Owner: CN=Debug Android, O=Android, C=US
Issuer: CN=Debug Android, O=Android, C=US
Serial number: 67317089
Valid from: Thu Jul 08 14:01:37 MSK 2021 until: Mon Jul 02 14:01:37 MSK 2046
Certificate fingerprints:
         SHA1: 76:2E:37:F7:45:06:8F:96:12:8C:D4:21:8A:FA:A6:32:D8:BC:FA:8B
         SHA256: ED:E1:1C:C1:95:0C:69:8F:92:32:5F:FB:A8:A7:C1:15:29:43:D6:F1:9E:A7:14:E2:49:67:03:05:03:95:ED:B8
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

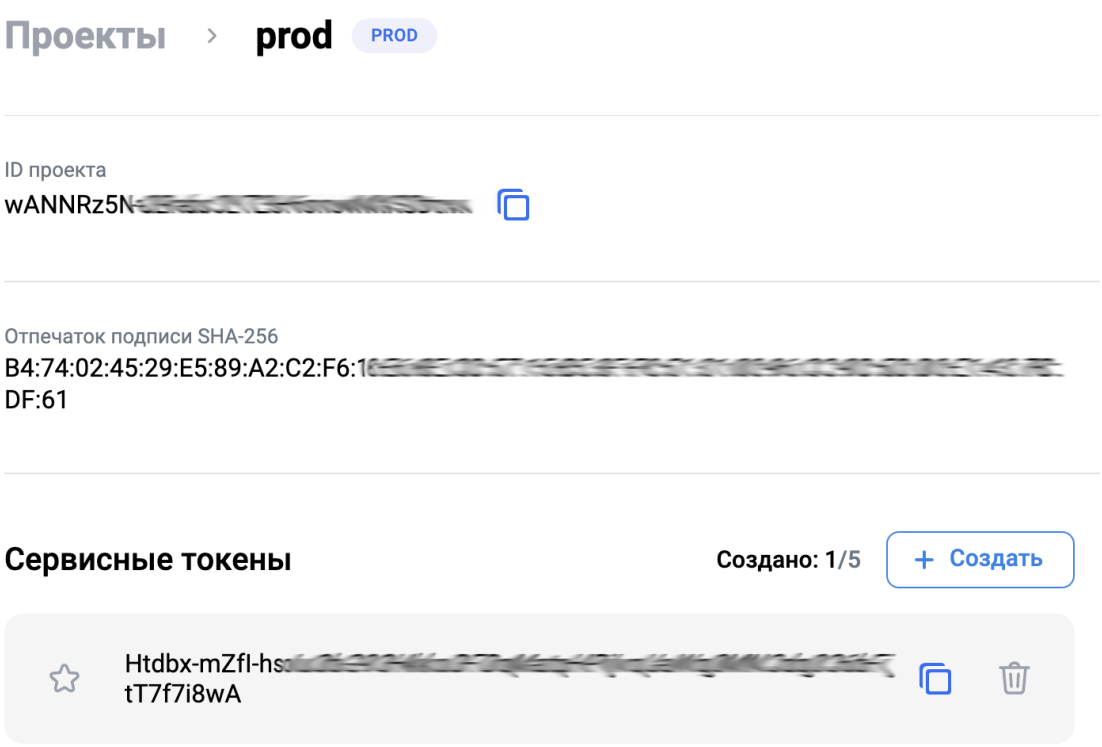
Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 17 FF FA BA 23 8F 2A BB   92 9A 37 07 27 CA 9A 4F  ....#.*...7.'..O
0010: D3 42 EA CC                                        .B..
]
]
```

签名指纹在 SHA256 字符串中指定。

推送通知验证

使用上述工具获取的签名指纹必须与推送通知项目中指定的指纹匹配 — “SHA-256 签名指纹”字段。



关于签名的详细信息，请参见 Google 的官方文档。

关于与潜在合作伙伴的信息交互协议

本协议（以下简称“**协议**”），由“VK”LLC（有限责任公司）（OGRN（基本国家注册号）1027739850962，俄罗斯，125167，莫斯科，列宁格勒大道39号，建筑79）（以下简称“**公司**”）与表现出对潜在与公司签订RuStore软件产品分发协议的兴趣的申请人（以下简称“**申请人**”）订立(https://help.rustore.ru/rustore/legal_info/developers_agreement).

本协议确定了公司与申请人之间信息交换的条件。

申请人通过提交填写完整的公司潜在合作伙伴问卷，该问卷包含对本协议的链接（以下简称“**问卷**”），通过Telegram上的聊天机器人的网页表单无条件且完全接受协议条款（根据俄罗斯联邦民法典第438条第3款）。

1. **协议主题**

1.1.公司无偿为申请人提供以下列出的服务（以下简称“服务”）：

I. 审查申请人是否符合公司合作伙伴的要求，以决定是否由公司签订、维护、修改、终止合同。

II. 将申请人记为公司的潜在合作伙伴，包括简化对申请人是否符合公司合作伙伴要求的重复审查。

III. 向申请人通报公司产品信息，包括邀请参加公司举办的活动。

1.2.为了提供服务，公司可能会从申请人处获取并处理关于申请人及其代表的信息，包括：

- 申请人身为自然人/个体经营者的全名；
- 申请人代表的全名；
- 申请人代表的职务；
- 申请人的电子邮件地址；
- 申请人的电话号码；
- 申请人在Telegram的用户名；
- 申请人代表的电子邮件地址；
- 申请人代表的电话号码；
- 申请人代表在Telegram的用户名；
- 作为法人的申请人的名称；
- 申请人的网站。

在上述清单中列出了通过Telegram聊天机器人中的问卷可能被请求的数据的最大范围。

1.3.申请人保证，其在问卷中提供的数据是由他个人提供的，且是真实、准确和有效的，并同意允许公司对提供的数据进行核查。

1.4.申请人特此授权公司在协议有效期内处理（收集、接收、记录、系统化、累积、存储、澄清（更新、更改）、提取、使用、封锁、删除、销毁）在填写问卷过程中提供的个人数据，无论是使用自动化工具还是非自动化方式，以实现本协议目的。

公司在填写问卷过程中收集提供的个人数据，此过程涉及第三方，即 “许多应用程序” 有限责任公司，地址：119071，俄罗斯，莫斯科，城市内部地区市政区域Donskoy（东斯科伊），列宁斯基大道，15A号，16楼，6室，OGRN（基本国家注册号）1237700094752，INN（纳税人识别号）9725113777。

1.5.为了履行协议（获得服务），申请人授权公司在协议有效期内，使用自动化工具或非自动化方式，按照本协议预定的方式处理有关申请人代表的信息，包括接收、记录、系统化、累积、存储、澄清（更新、更改）、使用、封锁、删除、销毁等手段。

申请人保证其有合法依据委托公司处理在问卷中指明的有关申请人代表和联系人的信息。如果这些合法依据对相应的数据主体失效，申请人承诺立即以书面形式通知公司，地址为公司所在地。

公司承诺在本协议框架内，按照申请人的委托保证所处理信息的保密性。公司承诺遵守适用法律对申请人委托处理的信息的保护要求。

申请人保证已通知其代表和联系人，公司将按照本协议的条款，应申请人的委托处理其数据。

1.6.公司承诺按照本协议和俄罗斯联邦法律的规定处理根据本协议获得的个人数据。

1.7.如果处理申请人代表信息的法律依据终止，申请人必须以书面形式通知公司，并在通知中注明相应代表的数据，地址为公司所在地。

1.8.公司有权在其拥有的关于与对手方关系管理的信息系统中存储由申请人提供的个人数据。

2. 协议的有效期、修改和解除

2.1.本协议自申请人接受之时起对申请人生效，并持续有效直至双方协商解除或根据本协议及俄罗斯联邦现行法律规定的理由和程序由任一方要求解除。

2.2.申请人有权通过向公司所在地地址发送通知来解除本协议。自公司接收到申请人通知之日起30个日历日后，本协议视为解除。

2.3.即使在申请人发送解除协议的通知后，如果存在其他合法的数据处理理由，公司仍有权继续处理申请人提供的个人数据。这些理由可能包括履行申请人作为一方的其他协议的需要，以及履行俄罗斯联邦法律赋予公司的功能、权力和义务。

2.4.本协议可由公司在通知申请人的条件下进行修改，通过在以下网址发布新版本的文件：https://help.rustore.ru/rustore/for_developers/potentialpartners。申请人有责任定期检查协议是否有更改。公司所做的协议更改自发布次日起生效。

2.5.在不履行和/或不适当履行协议义务的情况下，双方将根据俄罗斯联邦现行法律承担相应的责任。

3. 争议解决

3.1.与协议的签订、解释、执行和终止有关的所有争议，应通过双方协商解决。必须遵守解决争议的预审索赔程序。收到索赔的一方必须在收到索赔之日起 30（三十）个日历日内考虑索赔。

3.2.如果无法通过谈判解决分歧，双方有权向法院求助解决争端。在这种情况下，如果俄罗斯联邦法律没有其他规定，争议应提交公司所在地的相应法院处理。

修订日期：2023年11月9日

“RuStore奖品”活动规则

（以下简称“**活动规则**”、“**规则**”）

*本活动规则构成您（以下简称“**参与者**”）与活动组织者之间的协议，旨在规定参与活动的条款、参与者的权利与义务，以及活动的组织和进行方式。*

参与本次活动即表示您无条件完全接受这些活动规则。

1. 术语与定义

1.1. **"RuStore奖品"活动**（以下简称“**活动**”）是一个活动或一系列活动，其目的是吸引消费者关注归活动组织者所有的RuStore平台（https://www.rustore.ru/），提高该产品的知名度，并在商品、劳务和服务市场上推广它。

1.2. **活动组织者**为“VK”LLC（有限责任公司），注册地址：125167，俄罗斯，莫斯科，列宁格勒大道39号，建筑79，OGRN（基本国家注册号）1027739850962，INN（纳税人识别号）7743001840；

1.3. **活动的技术合作伙伴**为 “许多应用程序” 有限责任公司，地址：119071，俄罗斯，莫斯科，城市内部地区市政区域Donskoy（东斯科伊），列宁斯基大道，15A号，16楼，6室，OGRN（基本国家注册号）1237700094752，INN（纳税人识别号）9725113777。

1.4. **获奖者**是指根据活动规则条件有权获得奖品的参与者。

1.5. **活动地点：**

2023年11月9日至10日，活动将在“Mobius”会议期间的RuStore展位举行，地址为圣彼得堡，胜利广场1号，普尔科夫斯卡娅酒店；

2023年11月18日，将举行“RuStore Mobile Conf”会议，地址为莫斯科，列宁格勒大道39号，建筑79，SkyLight商务中心；

2023年11月27日至28日，“Highload++”会议，地点：斯科尔科沃校区，莫斯科州，奥金佐夫区，斯科尔科沃，诺娃娅街100号；

2023年11月30日至12月1日，“Team Lead Conf”会议，地点：斯科尔科沃校区，莫斯科州，奥金佐夫区，斯科尔科沃，诺娃娅街100号。

1.6. **活动参与者**是指已满18岁的自然人，具备必要的行为能力，并已执行参与活动所需的动作，如活动规则第3.1条所述。

1.7. **奖品**是指活动规则第6节中指定的奖品之一，将作为奖励传递给获奖者。

1.8. **不当优势**是指通过欺骗、伪造、勒索、胁迫、滥用、非法获取信息、数据访问或绕过活动条件，包括使用技术/软件工具，以及利用技术/软件等漏洞来获得奖品或活动中的其他

优势的结果。这些行为特别包括非法访问（黑客攻击）活动参与者的数据以及其他获得不当优势的方式。

2. 常规规定

2.1. 通过在活动地点发布信息和其他不受适用法律禁止的方式，向参与者和潜在参与者通知活动参与条件。

2.2. 如果活动期限、奖金基金或其他活动条件发生变化，组织者将在以下链接发布这些变化：https://help.rustore.ru/rustore/for_developers/rules_promotion。组织者有权单方面更改活动规则的条款。

2.3. 与活动的进行相关的所有问题，参与者可以通过发送电子邮件到 support@rustore.ru 向组织者提出。组织者将在工作日（星期一至星期五，莫斯科时间上午10:00至下午18:00）内自收到请求之日起7（七）个工作日内回复参与者的询问。

2.4. 此活动不是彩票、激励彩票或任何基于风险的赌博游戏。

2.5. 活动的官方语言为俄语。

2.6. 活动的参与者不能是活动组织者的员工、与组织者有关联的法人实体的员工、参与活动组织的人员、活动的技术合作伙伴的员工，以及上述所有人员的家庭成员。

2.7. 活动结果是最终的，不可复审，除非活动组织者在公布这些结果后发现了由获胜者/参与者违反活动规则的情况。

3. 参加活动的条件

3.1. 要参加活动，参与者必须执行以下操作：

3.1.1. 成为活动地点举办的会议的注册参与者；

3.1.2. 扫描活动地点放置的QR码，通过提供的链接进入Telegram的聊天机器人，确认已阅读本规则，并点击“Start”按钮。

3.1.3. 完成聊天机器人以游戏形式提出的任务，并因此获得一定数量的虚拟游戏货币。

3.2. 只有执行了第3.1款所述行为的参与者才有资格参加奖品抽奖。活动规则如果发现参与者违反活动规则的条件，组织者可以拒绝其参加活动。

3.3. 活动通过Telegram的聊天机器人进行，其功能由活动的技术合作伙伴提供。

4. 活动期限

4.1. 活动将在2023年11月9日莫斯科时间上午9:00至2023年12月1日莫斯科时间下午18:30期间举行（以下简称“**活动期限**”）。

4.2. 执行活动规则第3.1款所指行为的期限：在活动期间内。

4.3. 总结结果、确定/宣布获胜者和颁发奖品的期限：在活动期间内。

5. 确定获胜者的程序

5.1. 获胜者的数量限于奖品的数量，根据规则第6.1条。

5.2. 根据规则第3.1.2.-3.1.3款款完成聊天机器人任务的结果，每位活动参与者将获得一定数量的虚拟游戏货币，可在活动地点兑换可用的奖品。

6. 奖品基金

6.1. 奖品基金由活动组织者资助，包括：250个热水瓶，3700个装在小盒子里的 “扎布科因” 硬币，2400双袜子，3800个手机挡光帘，以及970套小型卡牌游戏（尺寸90x60毫米）。

6.1.1. 领取和使用奖品的条件：

a) 活动范围内的所有奖品均由活动组织者提供。

b) 参与者同意并承认，奖品的数量和奖金基金的构成可以根据活动组织者的判断进行更改。奖品的质量责任由其制造商承担。

c) 获奖者无权将奖品转让给他人，奖品也不得用作抵押或以其他方式负担。

d) 不提供奖品的现金等价物。

e) 活动期间，每位参与者只能获得一个奖品。

f) 奖品的领取和使用的附加条件，以及使用限制，可能在规则页面由活动组织者指定：https://help.rustore.ru/rustore/for_developers/rules_promotion.

6.2. 奖品的价值不超过4000.00卢布，因此根据俄罗斯联邦税法典第226条，活动组织者不作为被宣布为活动获胜者的人的税务代理，不负责支付超过4000.00卢布的奖品价值的个人所得税（根据税法典第224条，税率为35%，适用于税务居民），也无需计算、扣留并汇缴扣除的税款至国家预算，并且不必向税务机关提交获胜者以奖品形式收到的收入信息。

7. 颁发奖品的条件

7.1. 完成任务后（根据规则的3.1.2-3.1.3条），活动参与者有权在聊天机器人中从可用奖品列表中选择奖品，并联系活动组织者为在活动地点领取。为了颁发奖品，获奖者必须根据活动组织者的要求提供规则第8节中指定的信息。

7.2. 如获奖者未在活动结束前提供规则第8节中指定的信息，视为获奖者放弃领取奖品。

7.3. 在活动期间，只要获奖者提供了规则第8节中所需的信息，活动组织者将在活动地点颁发奖品。

7.4. 如果由于获奖者的行为导致奖品未能颁发，例如，获奖者错过领奖时间或在领奖期限内未能索取奖品、提供了颁发奖品所需的错误信息，组织者将不会再次颁发奖品。在这种情况下，奖品视为未被索取，活动组织者有权自行决定如何处理这些奖品。

7.5. 奖品不可退还、不可交换，也不能替换为其他奖品。不提供等同于奖品价值的现金支付。

8. 处理参与者信息的程序

8.1. 与处理参与者数据信息相关的关系受本活动规则以及俄罗斯联邦现行法律的调节。

8.2. 活动组织者根据以下条件处理参与者的信息：

收集的信息	处理目的
关于参与者和获奖者：a）参与者的姓名；b）电子邮件地址；c）公司名称； d）Telegram昵称。	为了签订、执行、修改、维护、解除与参与者和获奖者的活动规则，同时符合现行法律的要求。

8.3. 组织者通过收集、接收、记录、系统化、累积、存储、修订（更新、更改）、匹配、使用、阻止、删除和销毁参与者数据来处理参与者的数据。组织者有权委托第三方处理数据，这些第三方在活动规则的1.3条中有说明。活动规则

8.4. 组织者在活动期间处理参与者的数据，活动结束后（如果适用），只在现行法律允许的情况和期限内处理。

8.5. 如果活动规则终止，组织者将停止处理参与者的数据，并必须由组织者销毁相应数据。

8.6. 组织者采取技术和组织法律措施，以确保保护参与者数据信息不受非法或偶然的访问、破坏、更改、阻止、复制、传播，以及其他非法行为的影响。

8.7. 组织者实施的技术安全措施考虑到法律要求、现代技术水平、处理数据的性质以及与其处理相关的风险。

9. 其他条件

9.1. 与活动相关的所有争议问题均受俄罗斯联邦现行法律的调节。若出现可能导致活动规则条件有歧义的情况，以及（或）未在活动规则中明确的问题，最终解释和（或）澄清的决定由组织者独家作出。

9.2. 组织者有权单方面且无需解释原因，在任何时刻将以下人员从参与者中排除：

9.2.1. 被组织者怀疑试图获得或企图获得不当优势，以及（或）实施其他不诚实行为，其目的是不合理地获取奖品的个人；

9.2.2. 不符合活动规则对参与者提出的要求；

9.2.3. 侵犯第三方权利（例如，著作权、专有权和其他知识产权）；

9.2.4. 使用破坏聊天机器人运行和/或可能改变活动结果的软件；

9.2.5. 使用他人的数据或包含不完整和/或不真实信息的数据参与活动；

9.2.6. 违反活动规则的其他条件和（或）俄罗斯联邦法律的规定。

9.3. 如果参与者违反了活动规则第9.2条的义务，该参与者将按照俄罗斯联邦现行法律承担责任。在这种情况下，参与者将被视为已被取消资格，活动结果将被宣布无效。

如果由于参与者违反活动规则的条款或侵犯第三方权利而导致组织者遭受损失，参与者应自行承担与侵犯任何第三方权利相关的损失，并且如因此类违反行为组织者发生损失，参与者应赔偿组织者的损失。

9.4. 参与者有权在活动进行期间的任何时刻通过向组织者发送电子邮件通知（消息）至 support@rustore.ru来放弃参与活动。

9.5. 如果获奖者书面拒绝接受奖品，或进行活动规则第7.4条中提到的行为，他将失去从活动组织者那里获得奖品的权利。

9.6. 组织者不对参与者因参与活动而产生的费用（包括与互联网接入、通信服务费用以及可能出现的其他费用）给予补偿。

9.7. 如果出现组织者无法控制的情况，如非计划事件或特殊情况（火灾、洪水、自然灾害、互联网服务器遭受攻击、罢工），这些情况威胁到活动的财务或技术可行性，即使这些情况的责任在组织者（在组织者诚信履行其职责的条件下），组织者保留取消整个活动或其部分的权利。在活动被取消的情况下，参与者无权以任何形式要求补偿（包括赔偿损失）。

9.8. 如果由于任何原因，活动的任何方面不能按计划进行，包括由于计算机病毒感染、互联网问题、缺陷、操纵、未经授权的干预、伪造、技术故障或任何组织者无法控制的原因，这些原因扭曲或影响活动的执行、安全、公正、完整性或适当进行，组织者可以自行决定取消、终止、修改或暂时停止活动。

9.9. 活动组织者不对参与者由于参与活动而产生的任何直接或间接损失承担责任，包括由于电信和能源网络故障、恶意软件行为、第三方不正当行为导致未经授权的访问和/或破坏活动组织者的软件和/或硬件设施，以及不可抗力的不可预见情况造成的损失。在这些情况下，活动组织者无需向参与者赔偿损失。

2023年11月17日版本

SDK

目录

开发者文档 189
- RuStore SDK 用于 in-app 支付 189
  - Kotlin 189
  - Java 217
  - Unity 242
  - React Native 259
  - Flutter 265
  - Unreal Engine 270
- RuStore SDK 连接推送通知 285
  - Kotlin 285
  - Java 294
  - Unreal 300
  - Unity 310
  - Defold 314
  - Flutter 316
  - 发送推送通知 320
  - 通过主题发送推送通知 323
  - 向用户群组发送推送通知 326
- RuStore SDK用于通用推送通知 333
  - Kotlin 333
  - 发送通用推送通知 340
  - 通过主题发送通用推送通知 344
- RuStore SDK 用于集成评论和评分 348
  - Kotlin 348
  - Java 351
  - Unity 355
  - Unreal 358
  - Flutter 363

React Native 364
RuStore 应用程序更新SDK 365
Kotlin 365
Java 372
Unity 380
Flutter 385
Unreal 389
RuStore Geo 398
RuStore Geo 功能在 RuStore 的使用条件 398
概述 401
搜索和地理编码服务 403
地图显示服务 411
路线规划服务 431
MapsMobileSDK 590
附加服务 602
RuStore Deeplinks 602
TaskAPI 603
与其他SDK的兼容性 605
可用SDK列表 605

开发者文档

RuStore SDK 用于 in-app 支付

了解如何根据您的开发环境在您的应用程序中连接支付：

Kotlin

关键词: RuStore，开发者，文档，RuStoreSDK，连接支付

快速启动

实现示例

查看示例应用程序，了解如何正确集成支付。

支付条件

为了进行支付，必须遵守所有条件：

1. 用户的设备上安装了RuStore应用程序。
2. 用户在RuStore应用程序中已授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 在RuStore 控制台系统中为应用程序启用了购买功能。

该服务在俄罗斯联邦以外有一些运作限制。

添加存储库

连接存储库：

repositories { ``maven { ``url = uri("https://artifactory-external.vkpartner.ru/artifactory/maven") ``}}

集成依赖项

在您的配置文件中添加以下代码以连接依赖项：

—

dependencies { ``implementation("ru.rustore.sdk:billingclient:3.1.0")}

要了解连接依赖项的详细信息，请参阅相关信息。

项目中的库初始化

在调用库的方法之前，需要先进行其初始化。

使用 RuStoreBillingClientFactory.create() 创建 RuStoreBillingClient：

valbillingClient: RuStoreBillingClient = RuStoreBillingClientFactory.create(``context = app, ``consoleApplicationId = "111111", ``deeplinkScheme = "yourappscheme", ``// 可选参数 ``themeProvider = null, ``debugLogs = false, ``externalPaymentLoggerFactory = null,``)

- context—Android 的上下文。可以是任何上下文，在实现中使用 applicationContext。
- consoleApplicationId—来自RuStore开发者控制台的应用程序代码(例如：https://console.rustore.ru/apps/111111)。
- deeplinkScheme—用于通过第三方应用程序支付后返回您的应用程序的deeplink方案(例如通过 SberPay 或快速支付系统(SBP))。SDK 会为这个方案生成它自己的宿主。
- themeProvider—提供 BillingClientTheme 主题的接口。有2种 BillingClientTheme 主题实现：亮色(Light)和暗色(Dark)。这个接口是可选的，默认情况下使用亮色主题。
- externalPaymentLoggerFactory—提供对外部日志器访问的接口。事件日志记录
- debugLogs—控制日志记录的标志(Release 构建将自动关闭日志)。事件日志记录

重要提示。在 build.gradle 中指定的 ApplicationId 必须与您在 RuStore 控制台系统中发布的 apk 文件的 applicationId 匹配。

传递给 deeplinkScheme 的 deeplink 方案必须与 AndroidManifest.xml 中处理 deeplink部分指定的方案匹配。

keystore 的签名必须与发布在 RuStore 控制台系统中的应用程序签名匹配。确保使用的 buildType 使用与发布的应用程序相同的签名。

该库支持事件日志记录，可以在库初始化时单独连接。

要了解库初始化的详细信息，请查阅相关信息。

处理deeplink

为了在通过第三方应用程序(如快速支付系统(SBP)、SberPay 等)支付后返回您的应用程序，您需要正确实现 deeplink 的处理。在 AndroidManifest.xml 中指定带有您项目 scheme 的 intent-filter：

< activity ``android:name = ".YourBillingActivity" > ``< intent-filter > ``< action android:name = "android.intent.action.MAIN" /> ``< category android:name = "android.intent.category.LAUNCHER" /> ``</ intent-filter > ``< intent-filter > ``< action android:name = "android.intent.action.VIEW" /> ``< category android:name = "android.intent.category.DEFAULT" /> ``< category android:name = "android.intent.category.BROWSABLE" /> ``< data android:scheme = "yourappscheme" /> ``</ intent-filter ></ activity >

其中“yourappscheme”—是您的 deeplink 方案，可以更改为其他方案。这个方案必须与传递给 init() 的 deeplinkScheme 参数匹配。

为了成功返回应用程序，您还需要添加以下代码：

class YourBillingActivity: AppCompatActivity() { ``// Previously created with RuStoreBillingClientFactory.create() ``private val billingClient: RuStoreBillingClient =

YourDependencyInjection.getBillingClient() ``override fun onCreate(savedInstanceState: Bundle?)
{ ``super .onCreate(savedInstanceState) ``if (savedInstanceState == null)
{ ``billingClient.onNewIntent(intent) ``} ``} ``override fun onNewIntent(intent: Intent?)
{ ``super .onNewIntent(intent) ``billingClient.onNewIntent(intent) ``}}

要了解有关处理 deeplink 的详细信息，请查阅相关信息。

为了在通过 deeplink 返回时恢复您的应用程序状态，您需要在 AndroidManifest.xml 中添加：

android:launchMode="singleTask"

AndroidManifest.xml

<activity ``android:name= ".YourBillingActivity" ``android:launchMode=
"singleTask" ``android:exported= "true" ``android:screenOrientation=
"portrait" ``android:windowSoftInputMode= "adjustResize" >

概述

您可以查看快速集成应用程序支付的相关部分。

实现示例

查看示例应用程序，了解如何正确集成支付。

支付条件

为了进行支付，必须遵守所有条件：

1. 用户的设备上安装了RuStore应用程序。
2. 用户在RuStore应用程序中已授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 在RuStore 控制台系统中为应用程序启用了购买功能。

该服务在俄罗斯联邦以外有一些运作限制。

添加存储库

连接存储库：

repositories { ``maven { ``url = uri("https://artifactory-external.vkpartner.ru/artifactory/maven"
) ``}}

集成依赖项

在您的配置文件中添加以下代码以连接依赖项：

dependencies { ``implementation("ru.rustore.sdk:billingclient:3.1.0")}

初始化

在调用库的方法之前，需要先进行其初始化。

使用 RuStoreBillingClientFactory.create() 创建 RuStoreBillingClient：

创建 **RuStoreBillingClient**

valbillingClient: RuStoreBillingClient = RuStoreBillingClientFactory.create(``context = app, ``consoleApplicationId = "111111", ``deeplinkScheme = "yourappscheme", ``// 可选参数 ``themeProvider = null, ``debugLogs = false, ``externalPaymentLoggerFactory = null,``)

- context—Android 的上下文。可以是任何上下文，在实现中使用 applicationContext。
- consoleApplicationId—来自RuStore开发者控制台的应用程序代码(例如：https://console.rustore.ru/apps/111111)。
- deeplinkScheme—用于通过第三方应用程序支付后返回您的应用程序的deeplink方案(例如通过 SberPay 或快速支付系统(SBP))。SDK 会为这个方案生成它自己的宿主。
- themeProvider—提供 BillingClientTheme 主题的接口。有2种 BillingClientTheme 主题实现：亮色(Light)和暗色(Dark)。这个接口是可选的，默认情况下使用亮色主题。
- externalPaymentLoggerFactory—提供对外部日志器访问的接口。事件日志记录
- debugLogs—控制日志记录的标志(Release 构建将自动关闭日志)。事件日志记录

重要提示。在 build.gradle 中指定的 ApplicationId 必须与您在 applicationIdRuStore 控制台系统中发布的 apk 文件的 匹配。

传递给 deeplinkScheme 的 deeplink 方案必须与 AndroidManifest.xml 中处理 deeplink部分指定的方案匹配。

keystore 的签名必须与发布在 RuStore 控制台系统中的应用程序签名匹配。确保使用的 buildType(debug)使用与发布的应用程序相同的签名。

处理deeplink

为了通过第三方应用程序(如快速支付系统(SBP)或 SberPay)正确进行支付，您需要正确实现 deeplink 的处理。为此，您需要在 AndroidManifest.xml 中指定带有您项目 scheme 的 intent-filter：

< activity ``android:name = ".YourBillingActivity" > ``< intent-filter > ``< action android:name = "android.intent.action.MAIN" /> ``< category android:name = "android.intent.category.LAUNCHER" /> ``</ intent-filter > ``< intent-filter > ``< action android:name = "android.intent.action.VIEW" /> ``< category android:name = "android.intent.category.DEFAULT" /> ``< category android:name = "android.intent.category.BROWSABLE" /> ``< data android:scheme = "yourappscheme" /> ``</ intent-filter ></ activity >

其中“yourappscheme”—是您的 deeplink 方案，可以更改为其他方案。

这个方案必须与传递给 init() 的 deeplinkScheme 参数匹配。

接下来，在需要在支付后返回的 Activity（您的商店页面）中，您需要添加：

```
class YourBillingActivity: AppCompatActivity() {  ``// Previously created with RuStoreBillingClientFactory.create()  ``private val billingClient: RuStoreBillingClient = YourDependencyInjection.getBillingClient()  ``override fun onCreate(savedInstanceState: Bundle?) {  ``super .onCreate(savedInstanceState)  ``if (savedInstanceState ==  null ) {  ``billingClient.onNewIntent(intent)  ``}  ``}  ``override fun onNewIntent(intent: Intent?) {  ``super .onNewIntent(intent)  ``billingClient.onNewIntent(intent)  ``}}
```

为了在通过 deeplink 返回时恢复您的应用程序状态，您需要在 AndroidManifest.xml 中添加：

android:launchMode="singleTask"

```
<activity  ``android:name= ".YourBillingActivity"  ``android:launchMode= "singleTask"  ``android:exported= "true"  ``android:screenOrientation= "portrait"  ``android:windowSoftInputMode= "adjustResize" >
```

检查支付功能的可用性

要检查支付功能的可用性，请调用 checkPurchasesAvailability 方法。在调用时，将检查以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持支付功能。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序必须在 RuStore 控制台系统中启用购买功能。

如果所有条件都满足，则返回 FeatureAvailabilityResult.Available。否则，返回 FeatureAvailabilityResult.Unavailable(val cause: RuStoreException)，其中 cause — 是未满足条件的错误。所有可能的 RuStoreException 错误都被描述在错误处理部分。其他错误（例如，“没有互联网连接”）将在 onFailure 中返回。

```
RuStoreBillingClient.checkPurchasesAvailability(context)  ``.addOnSuccessListener { result ->  ``when (result) {  ``FeatureAvailabilityResult.Available -> {  ``// Process purchases available  ``}  ``is FeatureAvailabilityResult.Unavailable -> {  ``// Process purchases unavailable  ``}  ``}  ``}.addOnFailureListener { throwable ->  ``// Process unknown error  ``}
```

其中 context — 是 Android 的上下文。

获取产品列表的最新信息

关键词：RuStore 开发者，文档RuStoreSDK，连接支付，产品列表，产品。要获取产品列表，请使用 getProducts 方法：

—

```
val productsUseCase: ProductsUseCase =
billingClient.productsproductsUseCase.getProducts(productIds = listOf( "id1" , "id2"
))  ``.addOnSuccessListener { products: List<Product> ->      ``// Process
success  ``}  ``.addOnFailureListener { throwable: Throwable ->      ``// Process error  ``}
```

- productIds: list — 产品标识符列表。列表中的最大长度为 — 2083个字符。

此方法返回：

```
data  class Product(  ``val productId: String,  ``val productType: ProductType?,  ``val productStatus:
ProductStatus,  ``val priceLabel: String?,  ``val price: Int?,  ``val currency: String?,  ``val language:
String?,  ``val #### String?,`  ``val
```

- productId—产品标识符；
- productType—产品类型；
- productStatus—产品状态；
- priceLable—格式化的产品价格，包括[language]语言的货币符号；
- price—以最小单位（戈比）表示的价格；
- currency—ISO 4217货币代码；
- language—使用 BCP 47 编码指定的语言；
- title—产品名称，以[language]语言表示；
- description—产品描述，以[language]语言表示；
- imageUrl—图片链接；
- promoImageUrl—促销图片链接；
- subscription—订阅描述，仅对于类型为 subscription 的产品返回。

服务器对产品请求的响应结构：

产品结构

```
data  class ProductsResponse(  ``override val meta: RequestMeta?,  ``override val code:
Int,  ``override val errorMessage: String?,`  ``override val error
```

- meta—请求的附加信息；
- code—响应代码；
- errorMessage—用户的错误消息；
- errorDescription—错误消息的详细解释；
- errors—请求产品的错误列表；
- products—请求的产品列表。

订阅结构：

```
data  class ProductSubscription(  ``val subscriptionPeriod: SubscriptionPeriod?,  ``val freeTrialPeriod:
SubscriptionPeriod?,  ``val gracePeriod: SubscriptionPeriod?,  ``val introductoryPrice: String?,  ``val
introductoryPriceAmount: String?,  ``val introductoryPricePeriod: SubscriptionPeriod?)
```

- subscriptionPeriod—订阅周期；
- freeTrialPeriod—订阅试用期；

- gracePeriod—订阅宽限期；
- introductoryPrice—格式化的介绍性订阅价格，包括product:language语言的货币符号；
- introductoryPriceAmount—以货币最小单位（戈比）表示的介绍性价格；
- introductoryPricePeriod—介绍性价格计算周期。

订阅周期结构：

data class SubscriptionPeriod(``val years: Int,  ``val months: Int, ``val days: Int,)

- years—年数；
- months—月数；
- days—天数。

获取用户购买列表

关键词：RuStore 开发者，文档，RuStoreSDK，连接支付，购买，购买列表。要获取用户的购买列表，请使用 getPurchases 方法。getPurchases

```
val purchasesUseCase: PurchasesUseCase =
billingClient.purchasespurchasesUseCase.getPurchases()  ``.addOnSuccessListener { purchases:
List<Purchase> ->  ``// Process success  ``}  ``.addOnFailureListener { throwable: Throwable
->  ``// Process error  ``}
```

购买结构：

data class Purchase(``val purchaseId: String?,  ``val productId: String, ``val productType:
ProductType?, ``val invoiceId: String?,`  ``val

- purchaseId—购买标识符；
- productId—产品标识符；
- productType—产品类型；
- invoiceId—账单标识符；
- description—购买描述；
- language—使用 BCP 47 编码指定的语言；
- purchaseTime—购买时间（RFC 3339 格式）；
- orderId—应用程序生成的唯一支付标识符（uuid）；
- amountLable—格式化的购买价格，包括[language]语言的货币符号；
- amount—以最小货币单位表示的价格；
- currency—ISO 4217货币代码；
- quantity—产品数量；
- purchaseState—购买状态：
  - 购买状态的可能值：
    - CREATED—已创建；
    - INVOICE_CREATED—已创建，等待支付；
    - CONFIRMED—已确认；
    - PAID—已支付；
    - CANCELLED—购买已取消；
    - CONSUMED—已确认购买消耗；

- CLOSED—订阅已取消。
- developerPayload—开发者指定的字符串，包含关于订单的附加信息；
- subscriptionToken—用于在服务器端验证购买的令牌。了解有关在服务器上验证购买的更多信息。

服务器对购买请求的响应结构：

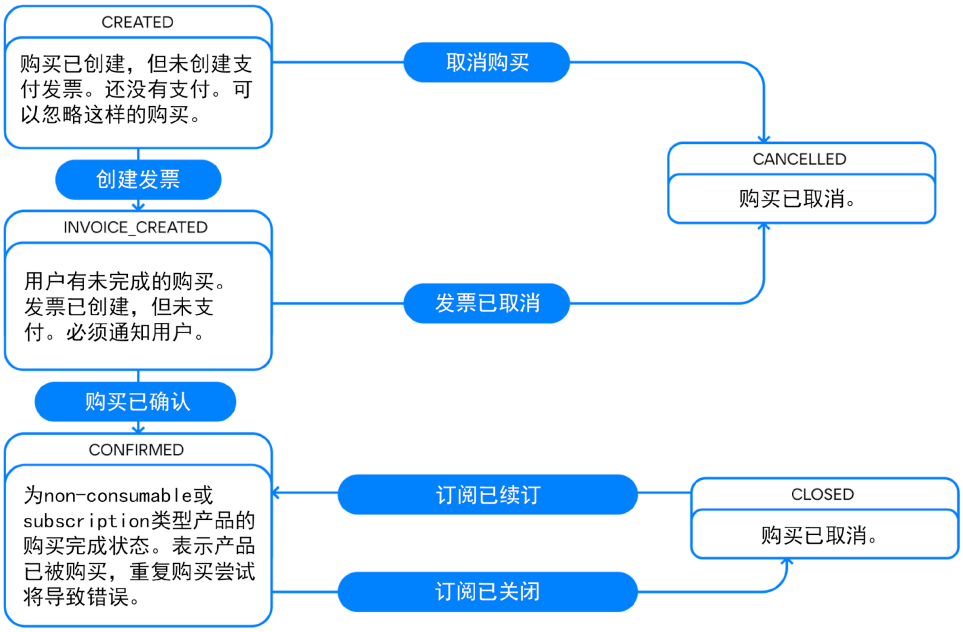
产品结构

data class PurchasesResponse(``override val meta: RequestMeta?, ``override val code: Int, ``override val errorMessage: String?,` ``override val error

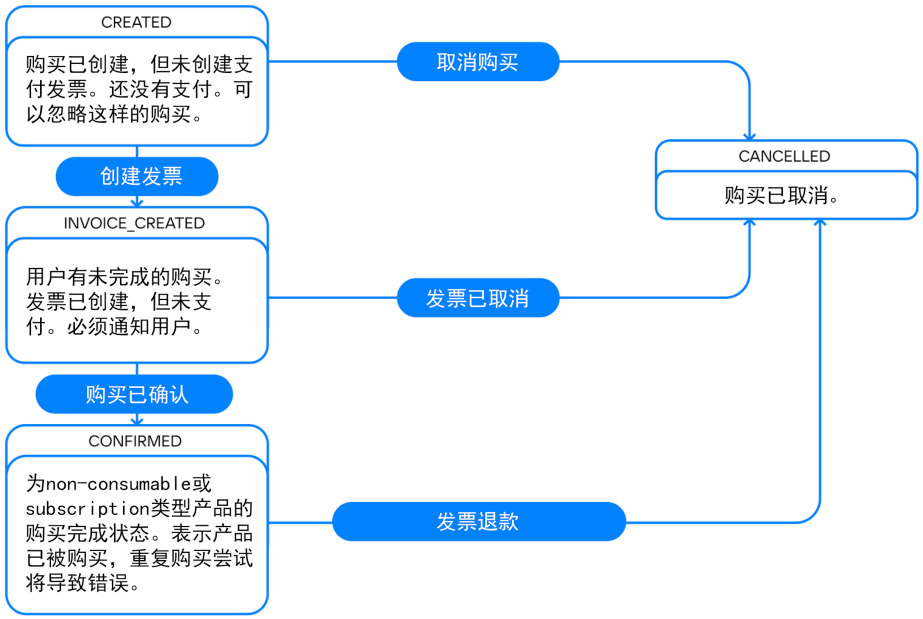
- meta—请求的附加信息；
- code—响应代码；
- errorMessage—用户的错误消息；
- errorDescription—错误消息的详细解释；
- errors—请求产品的错误列表；
- purchases—请求的购买列表。

状态模型 **(purchaseState)**：

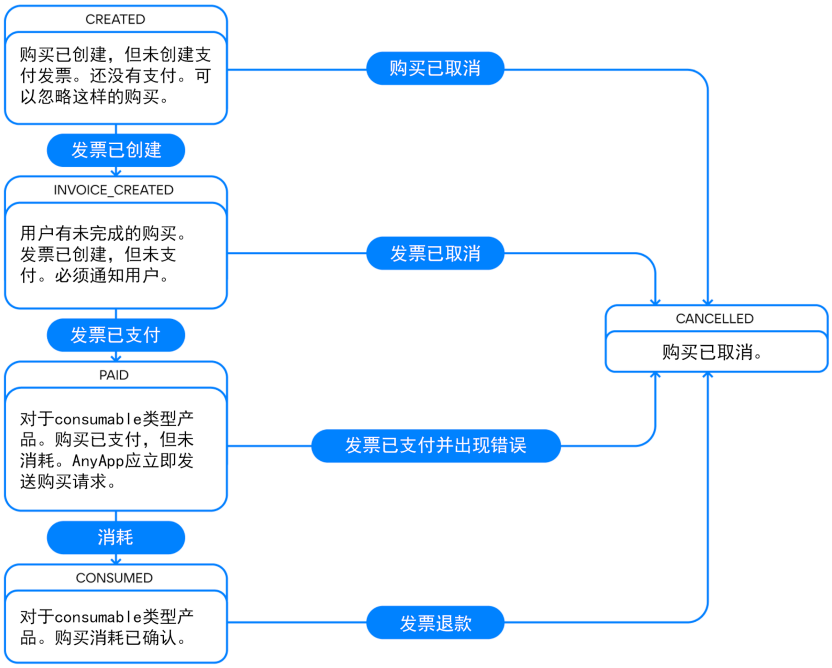
订阅购买状态模型 (SUBSCRIPTIONS)：



非消耗性产品购买状态模型 (NON-CONSUMABLES)：


CREATED
购买已创建，但未创建支付发票。还没有支付。可以忽略这样的购买。
取消购买
CANCELLED
购买已取消。
创建发票
INVOICE_CREATED
用户有未完成的购买。发票已创建，但未支付。必须通知用户。
发票已取消
购买已确认
CONFIRMED
为non-consumable或subscription类型产品的购买完成状态。表示产品已被购买，重复购买尝试将导致错误。
发票退款

消耗性产品购买状态模型 (CONSUMABLES)：



获取购买信息

关键词：RuStore，开发者，文档，RuStoreSDK，连接支付，购买，购买列表。要获取关于购买的信息，请使用 getPurchaseInfo 方法。

val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.getPurchaseInfo("purchaseId") ``.addOnSuccessListener { purchase: Purchase -> ``// Process success ``} ``.addOnFailureListener { throwable: Throwable -> ``// Process error ``}

购买结构：

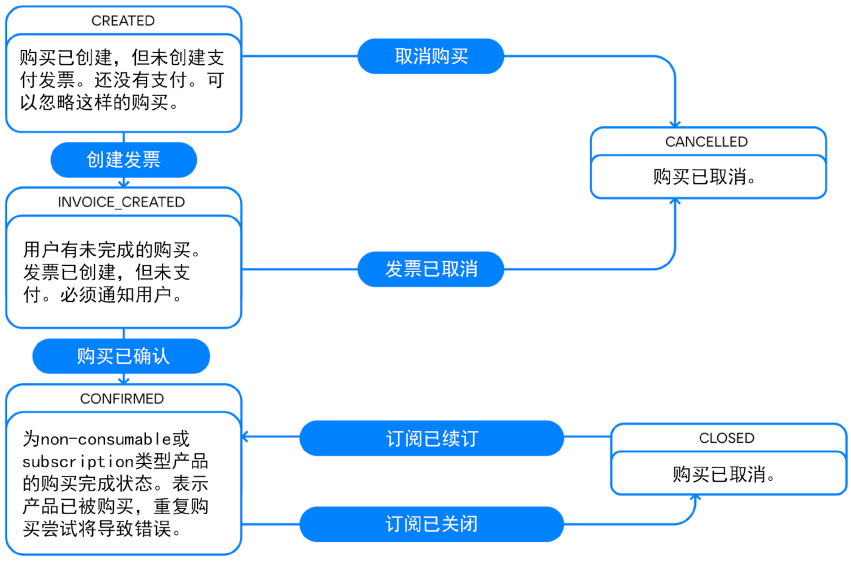
data class Purchase(``val purchaseId: String?, ``val productId: String, ``val productType: ProductType?, ``val invoiceId: String?,` ``val

- purchaseId—购买标识符；
- productId—产品标识符；
- productType—产品类型；
- invoiceId—账单标识符；
- description—购买描述；
- language—使用 BCP 47 编码指定的语言；
- purchaseTime—购买时间（RFC 3339 格式）；
- orderId—应用程序生成的唯一支付标识符（uuid）；
- amountLable—格式化的购买价格，包括[language]语言的货币符号；
- amount—以最小货币单位表示的价格；
- currency—ISO 4217货币代码；
- quantity—产品数量；

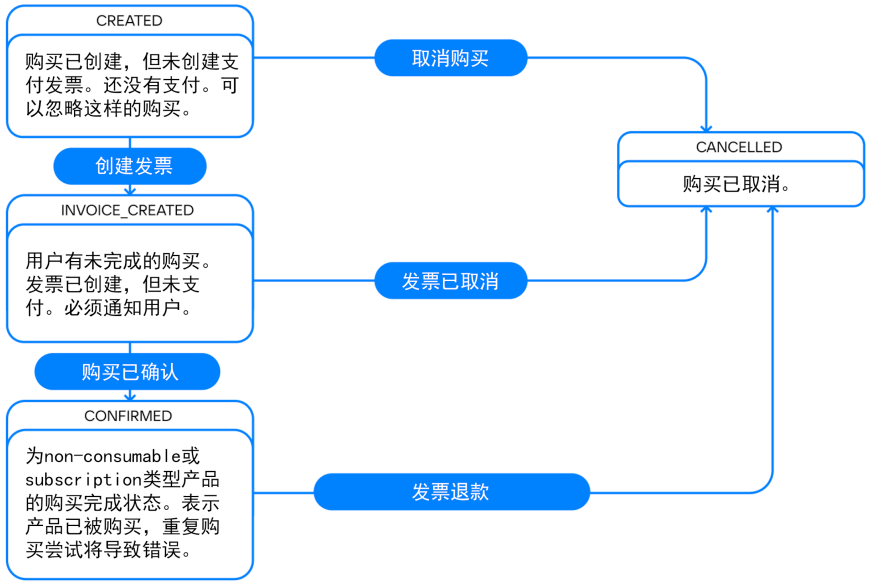
- purchaseState—购买状态：
  - 购买状态的可能值：
    - CREATED—已创建；
    - INVOICE_CREATED—已创建，等待支付；
    - CONFIRMED—已确认；
    - PAID—已支付；
    - CANCELLED—购买已取消；
    - CONSUMED—已确认购买消耗；
    - CLOSED—订阅已取消。
- developerPayload—开发者指定的字符串，包含关于订单的附加信息；
- subscriptionToken—用于在服务器端验证购买的令牌。了解有关在服务器上验证购买的更多信息。

状态模型 **(purchaseState)**：

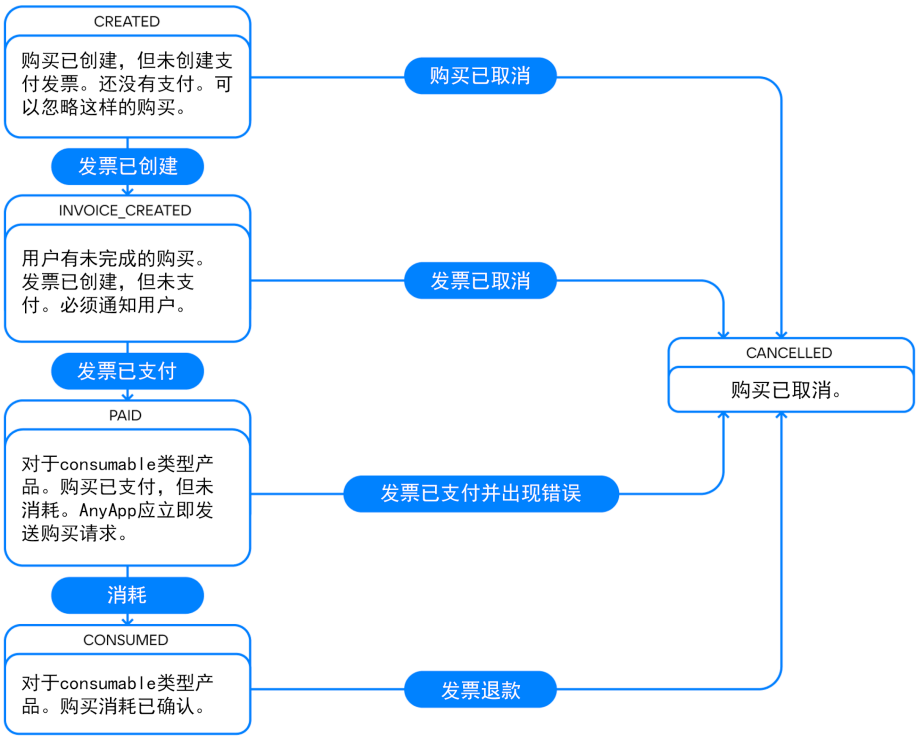
订阅购买状态模型 (SUBSCRIPTIONS)：



非消耗性产品购买状态模型 (NON-CONSUMABLES)：



消耗性产品购买状态模型 (CONSUMABLES)：



购买产品

关键词：RuStore，开发者，文档，RuStoreSDK，连接支付，购买产品。要调用购买产品，请使用 purchaseProduct 方法：

```
val purchasesUseCase: PurchasesUseCase =
billingClient.purchasespurchasesUseCase.purchaseProduct( ``productId = productId, ``orderId =
UUID.randomUUID().toString(), ``quantity = 1 , ``developerPayload = null ,).addOnSuccessListener {
paymentResult: PaymentResult -> ``when (paymentResult) { ``// Process
PaymentResult ``}}.addOnFailureListener { throwable: Throwable -> ``// Process error}
```

- productId: String — 产品标识符；
- orderId: String — 订单标识符，在AnyApp端创建（可选。如果未指定，则会自动生成）；
- quantity: Int — 产品数量（可选）；
- developerPayload—AnyApp 开发者的附加信息（可选）。

购买结果的结构：

—

```
public sealed interface PaymentResult { ``public data class Success( ``val orderId:
String?, ``val purchaseId: String, ``val productId: String, ``val invoiceId: String, ``val
subscriptionToken: String? = null , ``) : PaymentResult ``public data class Cancelled( ``val
purchaseId: String, ``) : PaymentResult ``public data class Failure( ``val purchaseId:
String?, ``val invoiceId: String?, ``val orderId: String?, ``val quantity: Int?, ``val productId:
```

String?, ``val errorCode: Int?, ``) : PaymentResult ``public object InvalidPaymentState : PaymentResult()}

- Success成功完成数字产品购买的结果。
- Failure购买数字产品时错误的结果。
- Cancelled数字产品购买被取消的结果。
- InvalidPaymentStateSDK支付错误。可能因deeplink回调不正确而发生。|

购买的服务器端验证

如果您需要在服务器端验证成功的购买，您可以使用 purchaseProduct 成功购买产品时返回的 PurchaseResult 中的 subscriptionToken。

SubscriptionToken由购买的invoiceId和RuStore的userId组成，用点书写：“*invoiceId*.userId”。

从购买结果中获取**subscriptionToken**

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.purchaseProduct(productId).addOnSuccessListener { paymentResult -> ``if (paymentResult is PaymentResult.Success) { ``val subscriptionToken = paymentResult.subscriptionToken ``yourApi.validate(subscriptionToken) ``}}
```

您还可以从 Purchase 实体中获取 subscriptionToken。可以通过使用 getPurchases() 方法来获取 Purchase 实体。

从购买结果中获取**subscriptionToken**

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.getPurchases().addOnSuccessListener { purchases -> ``purchases.forEach { purchase -> ``yourApi.validate(purchase.subscriptionToken) ``}}
```

购买消耗（确认）

关键词：RuStore，开发者，文档，RuStoreSDK，购买确认，消耗。RuStore 包含以下类型的产品：

- CONSUMABLE—消耗的（可以多次购买，例如应用中的水晶）；
- NON_CONSUMABLE—非消耗的（只能购买一次，例如禁用应用程序中的广告）；
- SUBSCRIPTION—订阅（可以购买一段时间，例如流媒体服务的订阅）。

只有处于 PurchaseState.PAID 状态的 CONSUMABLE 类型产品需要消耗。

您可以使用 confirmPurchase 方法来消耗购买：

调用确认方法

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.confirmPurchase(purchaseId = "purchaseId", developerPayload = null) ``.addOnSuccessListener { ``// Process success ``}.addOnFailureListener { throwable: Throwable -> ``// Process error ``}
```

- purchaseId—购买标识符；
- developerPayload—开发者指定的字符串，包含订单的附加信息（可选）。

取消购买

关键词：RuStore，开发者，文档，RuStoreSDK，购买取消。要取消购买，您可以使用 deletePurchase 方法：

```
val purchasesUseCase: PurchasesUseCase =
billingClient.purchasespurchasesUseCase.deletePurchase(purchaseId =  "purchaseId"
)  ``.addOnSuccessListener {      ``// Process success    ``}.addOnFailureListener { throwable:
Throwable ->      ``// Process error    ``}
```

- purchaseId—购买标识符。

如果您有与删除购买相关的逻辑，请使用此方法。购买将在20分钟后自动取消，或者在同一客户端进行重复购买时取消。

错误处理

可能的错误：

- RuStoreNotInstalledException() — 用户设备上未安装 RuStore。
- RuStoreOutdatedException() — 用户设备上安装的 RuStore 不支持支付。
- RuStoreUserUnauthorizedException() — 用户未在 RuStore 中授权。
- RuStoreApplicationBannedException() — 应用程序在 RuStore 中被禁止。
- RuStoreUserBannedException() — 用户在 RuStore 中被禁止。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

调用 RuStoreBillingClient.purchases.purchaseProduct() 方法时，错误会自动处理。

要向用户显示错误对话框，您可以使用 resolveForBilling 方法：

错误处理

```
public fun RuStoreException.resolveForBilling(context: Context)
```

购买消耗和取消的*流程*

关键词：RuStore，开发者，文档，RuStoreSDK，消耗，取消，购买确认。未完成的支付处理应由 AnyApp 的开发者执行。

如果出现以下情况，应使用取消购买方法（deletePurchase）：

1。获取购买列表方法（getPurchases）返回了具有以下状态的购买：

1. PurchaseState.CREATED.
2. PurchaseState.INVOICE_CREATED.

在某些情况下，通过银行应用程序（如快速支付系统(SBP)、SberPay、TinkoffPay等）支付后，当用户随后返回到 AnyApp 应用程序时，购买的状态可能仍然是

PurchaseState.INVOICE_CREATED。这与银行端处理购买的时间有关。因此，开发者需要正确地将获取购买列表的逻辑与屏幕生命周期相结合。

解决这个问题的另一种方法是，只有在用户与应用程序交互时，才取消处于 PurchaseState.INVOICE_CREATED 状态的购买。例如，可以将这个逻辑放到一个单独的按钮中。

购买方法 (purchaseProduct) 返回 PaymentResult.Cancelled。

购买方法 (purchaseProduct) 返回 PaymentResult.Failure。

如果获取购买列表的方法 (getPurchases) 返回了类型为 CONSUMABLE 且状态为 PurchaseState.PAID 的购买，则需要使用产品消耗方法 (confirmPurchase)。

事件日志记录

关键词：RuStore，开发者，文档，RuStoreSDK，连接支付，事件日志记录。如果您想在支付库中日志记录事件，可以在调用 RuStoreBillingClientFactory.create() 时添加 externalPaymentLoggerFactory 和 debugLogs 参数（这些参数对初始化是可选的）：

```
val billingClient: RuStoreBillingClient = RuStoreBillingClientFactory.create(    ``context =
app,    ``consoleApplicationId =  "111111" ,    ``deeplinkScheme =  "yourappscheme"
,    ``externalPaymentLoggerFactory = { tag -> PaymentLogger(tag) },    ``debugLogs =  true)class
PaymentLogger( private val tag: String) : ExternalPaymentLogger {    ``override fun d(e: Throwable?,
message: () -> String) {        ``Log.d(tag, message.invoke(), e)    ``}    ``override fun e(e: Throwable?,
message: () -> String) {        ``Log.e(tag, message.invoke(), e)    ``}    ``override fun i(e: Throwable?,
message: () -> String) {        ``Log.i(tag, message.invoke(), e)    ``}    ``override fun v(e: Throwable?,
message: () -> String) {        ``Log.v(tag, message.invoke(), e)    ``}    ``override fun w(e: Throwable?,
message: () -> String) {        ``Log.w(tag, message.invoke(), e)    ``}}
```

启用日志记录的参数：

- externalPaymentLoggerFactory—允许创建将库日志传递到宿主应用的记录器的接口。
- debugLogs—启用日志记录（Release 构建会自动关闭日志）

其中，PaymentLogger —是支付事件日志记录实现的一个示例。

更换主题

SDK 支持通过 BillingClientThemeProvider 提供者接口的动态主题更换：

```
valbillingClient: RuStoreBillingClient = RuStoreBillingClientFactory.create(    ``context =
app,    ``consoleApplicationId =  "111111" ,    ``deeplinkScheme =  "yourappscheme"
,    ``themeProvider? = BillingClientThemeProviderImpl(),)class BillingClientThemeProviderImpl:
BillingClientThemeProvider {        ``override fun provide(): BillingClientTheme {        ``// 检查已设置主
题的逻辑应该放置于此处``val darkTheme = ....        ``if
(darkTheme){        ``BillingClientTheme.Dark        ``}  else {        ``BillingClientTheme.Light        ``}    ``}}
```

错误处理

关键词：RuStore，开发者，文档，RuStoreSDK，连接支付，错误，处理错误。可能的错误：

- RuStoreNotInstalledException() — 用户设备上未安装 RuStore；
- RuStoreOutdatedException() — 用户设备上安装的 RuStore 不支持支付；
- RuStoreUserUnauthorizedException() — 用户未在 RuStore 中授权；
- RuStoreApplicationBannedException() — 应用程序在 RuStore 中被禁止；
- RuStoreUserBannedException() — 用户在 RuStore 中被禁止；
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

调用 RuStoreBillingClient.purchases.purchaseProduct() 方法时，错误会自动处理。

要向用户显示错误对话框，您可以使用 resolveForBilling 方法：

public fun RuStoreException.resolveForBilling(context: Context)

支付*SDK*错误代码

在“ResponseWithCode”接口的“code”字段中，可能出现的错误描述：

http code	code	描述
200	0	请求成功
400	40001	请求参数错误 — 缺少必填参数/参数格式错误
400	40003	应用未找到
400	40004	应用状态为“inactive”
400	40005	产品未找到
400	40006	产品状态为“inactive”
400	40007	产品类型无效。支持的类型包括：“consumable”，“non-consumable”，“subscription”
400	40008	已存在具有该“order_id”的购买
400	40009	当前客户已有此产品的购买记录，状态为“invoice_created”。需要提示客户支付/取消购买
400	40010	对于“consumable”类型产品。当前客户已有此产品的购买记录，状态为“paid”。首先需要在设备上确认购买消耗，然后才能发送购买此产品的下一个请求。
400	40011	对于“non-consumable”类型产品。当前客户已有此产品的购买记录，状态为“pre_confirmed”/“confirmed”。该产品已被购买。产品不能多次销售
400	40012	对于“subscription”类型产品。当前客户已有此产品的购买记录，状态为“pre_confirmed”/“confirmed”。该产品已被购买。产品不能多次销售
400	40013	对于“subscription”类型产品。在请求订阅服务的产品列表“GET/products”（“serviceId”，“user_id”）时，数据未被接收。
400	40014	请求中缺少必要属性
400	40015	在更新购买时无法更改状态（禁止转换）
400	40016	购买非消耗产品的订阅时，指定的 quantity > 1
400	40017	产品已删除，新的购买不可用
400	40018	无法消耗类型为“产品类型”的产品

http code	code	描述
401	40101	无效的令牌
401	40102	令牌的有效期已过
403	40301	请求的资源访问被拒绝（未授权）
403	40302	当前令牌不授权当前调用（方法被禁止）
403	40303	请求中的应用程序标识符与令牌不匹配
403	40305	令牌类型错误
404	40401	未找到
408	40801	请求中指定的通知等待时间已过
500	50***	支付服务的内部错误

迁移到支付版本*2.2.0*

概述

在支付版本 2.2.0 中，PaymentResult 中的产品购买结果模型发生了显著变化。

为了快速且无大问题地迁移到 SDK 的新版本，请遵循此迁移指南。

更新依赖项

为了更新依赖项，请在您的 build.gradle 文件的 dependencies 区域中升级 billingclient 的版本：

build.gradle

```
dependencies {    ``implementation( "ru.rustore.sdk:billingclient:2.2.0" )}
```

更改模型

获取产品列表

获取产品列表的结果模型已发生了变化。现在，**getProducts()** 方法直接返回产品列表：

```
val productsUseCase: ProductsUseCase =
billingClient.productsproductsUseCase.getProducts(productIds = listOf( "id1" ,  "id2"
))    ``.addOnSuccessListener { products: List<Product> ->       ``// Process
success    ``}    ``.addOnFailureListener { throwable: Throwable ->       ``// Process error    ``}
```

产品模型和错误模型保持不变。

获取购买列表

获取购买列表的结果模型发生了变化。现在，**getPurchases()** 方法会直接返回购买列表：

```
val purchasesUseCase: PurchasesUseCase =
billingClient.purchasespurchasesUseCase.getPurchases()    ``.addOnSuccessListener { purchases:
List<Purchase> ->       ``// Process success    ``}    ``.addOnFailureListener { throwable: Throwable
->       ``// Process error    ``}
```

购买模型和错误模型保持不变。

获取购买信息

获取购买信息的结果模型发生了变化。现在，**getPurchaseInfo()** 方法会直接返回购买模型：

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.getPurchaseInfo( "purchaseId" )  ``.addOnSuccessListener { purchase: Purchase ->  ``// Process success  ``}  ``.addOnFailureListener { throwable: Throwable ->  ``// Process error  ``}
```

错误模型保持不变。

购买产品

购买产品的结果模型发生了变化。更新后的模型如下所示：

```
public sealed  interface PaymentResult {  ``public data  class Success(  ``val orderId: String?,  ``val purchaseId: String,  ``val productId: String,  ``val invoiceId: String,  ``val subscriptionToken: String? =  null ,  ``) : PaymentResult  ``public data  class Cancelled(  ``val purchaseId: String,  ``) : PaymentResult  ``public data  class Failure(  ``val purchaseId: String?,  ``val invoiceId: String?,  ``val orderId: String?,  ``val quantity: Int?,  ``val productId: String?,  ``val errorCode: Int?,  ``) : PaymentResult  ``public object InvalidPaymentState : PaymentResult()}
```

其中：

- Success—成功完成数字产品购买的结果。
- Failure—购买数字产品时错误的结果。
- Cancelled—数字产品购买被取消的结果。
- InvalidPaymentState—支付SDK操作错误。可能因deeplink回调不正确而发生。

特别注意购买消耗和取消的流程。

购买消耗（确认）

购买消耗的结果模型发生了变化。现在，消耗可能以成功或错误结束：

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.confirmPurchase(purchaseId =  "purchaseId" , developerPayload =  null )  ``.addOnSuccessListener {  ``// Process success  ``}.addOnFailureListener { throwable: Throwable ->  ``// Process error  ``}
```

取消购买

购买取消的结果模型发生了变化。现在，取消购买可能以成功或错误结束：

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.deletePurchase(purchaseId =  "purchaseId" )  ``.addOnSuccessListener {  ``// Process success  ``}.addOnFailureListener { throwable: Throwable ->  ``// Process error  ``}
```

购买消耗和取消的流程

由于产品购买结果模型发生了变化，因此购买的消耗和取消的逻辑也随之发生了变化。

如果出现以下情况，应使用取消购买方法（**deletePurchase**）：

1. 获取购买列表的方法（getPurchases）返回的购买状态为PurchaseState.CREATED或PurchaseState.INVOICE_CREATED。

在某些情况下，通过银行应用程序（如快速支付系统(SBP)、SberPay、TinkoffPay等）支付后，当用户随后返回到AnyApp应用程序时，购买的状态可能仍然是PurchaseState.INVOICE_CREATED。这与银行端处理购买的时间有关。因此，开发者需要正确地将获取购买列表的逻辑与屏幕生命周期相结合。解决这个问题的另一种方法是，只有在用户与应用程序交互时，才取消处于 PurchaseState.INVOICE_CREATED 状态的购买。例如，可以将这个逻辑放到一个单独的按钮中。

2. 购买方法 (purchaseProduct) 返回 PaymentResult.Cancelled。
3. 购买方法 (purchaseProduct) 返回 PaymentResult.Failure。

如果出现以下情况，应使用产品消耗方法（**confirmPurchase**）：

1. 获取购买列表的方法 (getPurchases) 返回了类型为 CONSUMABLE 且状态为 PurchaseState.PAID 的购买。

支付问题的*FAQ*

Q: 如何修复**“Application is not verified yet”**错误？

A: 出现此错误的情况包括：

- 应用程序未通过RuStore控制台的审核；
- 开发者测试的apk与上传到RuStore控制台的apk不匹配。

第二点需要按以下方式重新检查：

- 在 build.gradle 中指定的 applicationId 必须与您在 RuStore 控制台发布的 apk 文件的 applicationId 匹配。
- keystore 的签名必须与发布在 RuStore 控制台的应用程序签名匹配。确保使用的 buildType（例如，debug）使用与发布的应用程序（例如，release）相同的签名。

为了支付功能的运作，需要完全发布应用程序，仅通过审核是不够的。不久的将来，逻辑将被重新设计，以便测试支付功能仅需通过审核即可。

Q: 如何修复**“**之前创建的产品**'...'**的购买，数量为**...**，金额为**...**卢布，在另一个会话中支付**”**的错误？

A: 当尝试购买之前中断且未通过deletePurchase和confirmPurchase方法转入最终状态的产品时，会出现此错误。

这通常发生在流程被中断，且由于流程未正确结束，未在purchaseProduct中调用删除或消耗操作时。

对于这种情况，在应用启动或打开商店时，需要对“悬而未决”的购买进行取消或消耗处理。

以下是实现购买列表处理的示例。启动应用程序或打开商店屏幕时运行此代码：

```
val purchasesUseCase = billingClient.purchasesval purchases =
purchasesUseCase.getPurchases().await().purchases.orEmpty()purchases.forEach { purchase
->    val purchaseId = purchase.purchaseIdif(purchaseId != null) {       when
(purchase.purchaseState) {             PurchaseState.CREATED, PurchaseState.INVOICE_CREATED
->
{                          purchasesUseCase.deletePurchase(purchaseId).await()          }
PurchaseState.PAID ->
{              purchasesUseCase.confirmPurchase(purchaseId).await()          }else->
Unit      }   }}
```

在purchaseProduct()中，也需要执行购买的取消或消耗操作。

在purchaseProduct()中处理购买，将购买转换为最终状态，确认或取消它，可以这样实现：

```
privatefun purchaseProduct(product: Product) {    val purchasesUseCase =
billingClient.purchases    purchasesUseCase.purchaseProduct(product.productId)      .addOnS
uccessListener { paymentResult ->          handlePaymentResult(paymentResult,
product)      }      .addOnFailureListener {// Handle error      }}privatefun
handlePaymentResult(paymentResult: PaymentResult, product: Product) {    when
(paymentResult) {      is PaymentResult.InvalidPurchase ->
{         paymentResult.purchaseId?.let { deletePurchase(it) }      }      is
PaymentResult.PurchaseResult -> {          when (paymentResult.finishCode)
{            PaymentFinishCode.SUCCESSFUL_PAYMENT -> {if(product.productType ==
ProductType.CONSUMABLE)
{                confirmPurchase(paymentResult.purchaseId)          }          }          Payme
ntFinishCode.CLOSED_BY_USER,          PaymentFinishCode.UNHANDLED_FORM_ERROR,
      PaymentFinishCode.PAYMENT_TIMEOUT,          PaymentFinishCode.DECLINED_BY_SER
VER,          PaymentFinishCode.RESULT_UNKNOWN,          ->
{              deletePurchase(paymentResult.purchaseId)          }        }      }else-> Unit    }}
```

如何消耗和取消购买在“购买消耗和取消的流程”部分有描述。

可以在“获取购买列表”部分了解购买的状态模型。

Q: 如何进行购买的服务器端验证？

A: 首先，需要获取subscriptionToken，它是用户购买的唯一标识符，此过程在“购买的服务器端验证”文章中有描述。

然后，需要将subscriptionToken发送到您的backend，在那里可以使用“根据其subscription token获取支付信息的方法”来请求购买信息。

Q: 如何在调用**confirmPurchase**或**deletePurchase**时修复**404**错误？

A: 确保您在confirmPurchase和deletePurchase方法的参数中传递purchaseId：

val purchasesUseCase = billingClient.purchasesval purchases = purchasesUseCase.getPurchases().await().purchases.orEmpty()purchases.forEach { purchase -> val purchaseId = purchase.purchaseIdif(purchaseId != null) { when (purchase.purchaseState) { PurchaseState.CREATED, PurchaseState.INVOICE_CREATED -> {// purchasesUseCase.deletePurchase(purchaseId = purchase.productId).await() WRONG purchasesUseCase.deletePurchase(purchaseId = purchaseId).await() // CORRECT } PurchaseState.PAID -> {// purchasesUseCase.confirmPurchase(purchaseId = purchase.productId).await() WRONG purchasesUseCase.confirmPurchase(purchaseId = purchaseId).await() // CORRECT }else-> Unit } }}

Q: 如何修复“方法不可用”的错误？

A: consoleApplicationId 必须与 RuStore 控制台中的应用程序代码匹配(例如：https://console.rustore.ru/apps/111111)。

Q: 如何取消订阅？

A: 没有用于取消订阅的方法，您只能在RuStore应用中取消自动续订。

可以通过deeplink打开订阅页面：

startActivity(Intent(Intent.ACTION_VIEW, Uri.parse(“rustore://profile/subscriptions”)))

以下是带有deeplinks列表的页面：
https://help.rustore.ru/rustore/for_developers/developer-documentation/RuStore_deeplinks

Q: 可以在**Google Play**、**Huawei Store**上发布带有**RuStore SDK**的应用吗？

A: 由于开发者在Google Play上用Google的签名对应用进行签名，而在RuStore上使用他们自己的签名，因此签名始终不会匹配，RuStore SDK在Google Play(或其他应用商店)内将无法工作。这意味着，为了RuStore SDK的正确运行，用户需要从RuStore下载应用并在其中支付订阅费用。

Q: RuStore的**packageName**是什么？

A: ru.vk.store.

Q: 如何确定应用是从哪个商店安装的？

A: 可以通过以下方式进行判断：

val installerPackage = packageManager.getInstallerPackageName(applicationInfo.packageName)

将返回ru.vk.store，但此功能不稳定：

- 这种方法只适用于最初从RuStore安装的应用。如果最初是从Google Play或其他方式安装的，那么安装来源将是标准的包安装程序。
- 如果使用兼容模式安装（如部分Xiaomi机型），则安装源将为小米系统安装程序。
- 如果删除了RuStore，则安装来源将被完全删除。重新安装不会恢复安装来源。

我们建议为RuStore制作单独的buildFlavor。

Q: 为什么在支付方法中出现**timeout**？ **(PayLibBackendFailure$TimeoutError)**

A: RuStore SDK的支付功能在俄罗斯境外不可用。开启VPN也可能会导致问题。

Q: 如何测试支付？使用真实的银行卡吗？

A: 只能使用真实的银行卡进行测试。测试沙箱目前正在开发中。

Q: 是否支持**Java**？

A: 是的，支持。Kotlin与Java有向后兼容性，但有一些特点。

例如，我们来看看Kotlin中的object实体，它相当于Java中的static class：

RuStoreReviewManagerFactory.create(context)

在Java中访问object RuStoreReviewManagerFactory的方式如下：

RuStoreReviewManagerFactory.INSTANCE.create(getContext());

有关在Java类中使用Kotlin代码的更多信息，请参阅文章《Calling Kotlin from Java》。

*RuStore SDK*支付的更新历史

SDK版本 3.1.0 日期01.12.2023
- 添加了支付沙箱（sandbox）功能
- SDK内部更新

SDK版本 3.0.0 日期26.10.2023
- 更新了暗色主题的新色彩范围功能；
- 错误修复。

SDK版本 2.2.0 日期24.08.2023
- 添加了动态主题切换功能（亮色和暗色）；
- 对库的稳定性进行了优化；
- 修复了通过deeplink支付后恢复状态的问题。

SDK版本 2.1.2 日期02.08.2023
- 修复了在设备上没有RuStore应用程序时的错误

SDK版本 2.1.1 日期25.07.2023
- 安全更新

SDK版本 2.1.0 日期09.06.2023
- 更改了响应模型：

  - 获取产品列表
  - 获取购买列表
  - 购买产品
  - 消耗购买
  - 取消购买
- 支付对话框的外观进行了优化。

SDK版本 1.1.1 日期03.05.2023

- 修复了Task API的*await()*方法的操作

SDK版本 1.1.0 日期19.04.2023

- 使用TinkoffPay进行支付。
- 支付时保存银行卡的功能。
- 支付对话框的外观和行为进行了优化。
- 移除了多余的依赖项和uses permissions。
- PurchaseResult模型增加了新的账单标识符字段“invoiceId”。

SDK版本 1.0.0 日期11.04.2023

- 从单例模式转换到创建实例：RuStoreBillingClient.init() 更改为 RuStoreBillingClientFactory.create()。
- 与单例模式相关的方法（init, products, purchases, getSingleton）标记为 deprecated，并将在后续版本中移除。
- 关于迁移的更多详细信息，请参阅迁移指南：[Kotlin] 到1.0.0版本支付的**迁移** / [Java] 到 1.0.0版本支付的**迁**移
- checkPurchasesAvailability() 方法已变为静态 - 您无需创建 RuStoreBillingClient 实例即可检查付款的可用性。

SDK版本 0.1.8 日期13.03.2023

- 添加了方法RuStoreBillingClient.purchases.getPurchaseInfo(“purchaseId”)，用以返回购买信息。
- 修复了“Intent Redirection vulnerability”漏洞，该漏洞阻止应用程序在Google Play上发布。

SDK版本 0.1.7 日期10.02.2023

- SDK的内部更新。

SDK版本 0.1.6 日期03.02.2023

- 在Purchase实体中添加了subscriptionToken字段，用于服务器端购买验证。

SDK版本 0.1.5 日期30.01.2023

- 转换到俄数字部的证书。
- 修复了在快速切换到服务条款并返回时通过 快速支付系统（SBP）支付按钮的显示问题。
- 在ResponseWithCode接口中，将traceId: String字段替换为包含traceId的meta: RequestMeta字段。

SDK版本 0.1.4 日期20.01.2023

- 为**快速支付系统**(SBP)和通过手机账户支付添加了服务条款链接。
- 如果支付确认超过15秒，为用户添加了额外的免责声明。
- 修复了与OTP代码操作和发送短信时的错误。
- 修复了在RuStore端为发票生成orderId字段的问题，如果在purchaseProduct()中将orderId传递为null。
- 如果银行应用不支持快速支付系统(SBP)的deeplink，添加了显示阻塞错误的功能。
- 添加了PurchaseResult.subscriptionToken，允许在服务器端进行购买验证。

SDK版本 0.1.3 日期28.12.2022

- 在purchaseProduct()中添加了通过电话号码进行支付的功能。
- 在初始化过程中，将参数externalPaymentLogger更换为externalPayemtnLoggerFactory — 这是一个工厂，在其中需要创建ExternalPaymentLogger的实现(参见日志记录部分)。

SDK版本 0.1.2 日期20.12.2022

- 添加了快速支付系统(SBP)(快速支付系统(SBP))。
- 在init方法中添加了deeplinkScheme参数。
- 在init方法中删除了deeplinkPrefix参数。
- 修复了应用程序最小化后，在输入信用卡数据的表单中数据丢失的错误。

SDK版本 0.1.1 日期25.11.2022

- 在以下方法中删除了“language”参数：
  - 获取产品列表 — “getProducts”。
  - 获取购买列表 — “getPurchases”。
  - 产品消耗 — “confirmPurchase”。
  - 取消购买 — “deletePurchase”。
- 添加了RuStoreBillingClient.isInitialized字段，用于返回库初始化的状态。

SDK版本 0.1.0 日期11.11.2022

- Init中的参数发生了变化。
- 添加了检查支付功能可用性的方法 — “checkPurchasesAvailability”。
- 所有方法中的“language”参数变为可选。
- SuspendSuspend方法在以下方法中被task API替换：
  - 获取产品列表 — “getProducts”。
  - 获取购买列表 — “getPurchases”。
  - 购买产品 — “purchaseProduct”。
  - 产品消耗 — “confirmPurchase”。
  - 取消购买 — “deletePurchase”。
- 在“purchaseProduct”购买方法中删除了“context”参数。
- 删除了“resultObserver”方法，现在购买结果通过“purchaseProduct”返回。

- 在“resolveForBilling”方法中删除了“onFail”参数。

SDK版本 0.0.9 日期27.10.2022

- 在init函数中添加了“ExternalPaymentLogger”和“debugLogs”可选参数。

*SDK*支付从版本 *1.x.x* 迁移到版本 *3.x.x*

在支付版本 3.0.0 中，PaymentResult 中的产品购买结果模型发生了显著变化。

为了快速且无大问题地迁移到 SDK 的新版本，请遵循此迁移指南。

更新依赖项

为了更新依赖项，请在您的 build.gradle 文件的 billingclient 区域中升级 dependencies 的版本：

build.gradle

```
dependencies {    ``implementation( "ru.rustore.sdk:billingclient:3.1.0" )}
```

更改模型

获取产品列表

获取产品列表的结果模型已发生了变化。现在，**getProducts()** 方法直接返回产品列表：

调用方法**getProducts**

```
val productsUseCase: ProductsUseCase =
billingClient.productsproductsUseCase.getProducts(productIds = listOf( "id1" ,  "id2"
))    ``.addOnSuccessListener { products: List<Product> ->       ``// Process
success    ``}    ``.addOnFailureListener { throwable: Throwable ->       ``// Process error    ``}
```

产品模型和错误模型保持不变。

获取购买列表

获取购买列表的结果模型发生了变化。现在，**getPurchases()** 方法会直接返回购买列表：

调用获取用户购买列表的方法

```
val purchasesUseCase: PurchasesUseCase =
billingClient.purchasespurchasesUseCase.getPurchases()    ``.addOnSuccessListener { purchases:
List<Purchase> ->       ``// Process success    ``}    ``.addOnFailureListener { throwable: Throwable
->       ``// Process error    ``}
```

购买模型和错误模型保持不变。

获取购买信息

获取购买信息的结果模型发生了变化。现在，**getPurchaseInfo()** 方法会直接返回购买模型：

调用获取用户购买列表的方法

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.getPurchaseInfo( "purchaseId" )  ``.addOnSuccessListener { purchase: Purchase ->  ``// Process success  ``}  ``.addOnFailureListener { throwable: Throwable ->  ``// Process error  ``}
```

错误模型保持不变。

购买产品

购买产品的结果模型发生了变化。更新后的模型如下所示：

购买结果的结构体

```
public sealed  interface PaymentResult {  ``public data  class Success(  ``val orderId: String?,  ``val purchaseId: String,  ``val productId: String,  ``val invoiceId: String,  ``val subscriptionToken: String? =  null ,  ``) : PaymentResult  ``public data  class Cancelled(  ``val purchaseId: String,  ``) : PaymentResult  ``public data  class Failure(  ``val purchaseId: String?,  ``val invoiceId: String?,  ``val orderId: String?,  ``val quantity: Int?,  ``val productId: String?,  ``val errorCode: Int?,  ``) : PaymentResult  ``public object InvalidPaymentState : PaymentResult()}
```

其中：

- Success- 成功完成数字产品购买的结果。
- Failure- 购买数字产品时错误的结果。
- Cancelled- 数字产品购买被取消的结果。
- InvalidPaymentState- SDK支付错误。可能因deeplink回调不正确而发生。

需要特别注意消耗和取消购买的流程！

购买消耗（确认）

购买消耗的结果模型发生了变化。现在，消耗可能以成功或错误结束：

调用确认方法

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.confirmPurchase(purchaseId =  "purchaseId" , developerPayload =  null )  ``.addOnSuccessListener {  ``// Process success  ``}.addOnFailureListener { throwable: Throwable ->  ``// Process error  ``}
```

取消购买

购买取消的结果模型发生了变化。现在，取消购买可能以成功或错误结束：

调用取消购买方法。

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchasespurchasesUseCase.deletePurchase(purchaseId =  "purchaseId"
```

) ``.addOnSuccessListener { ``// Process success ``}.addOnFailureListener { throwable: Throwable -> ``// Process error ``}

购买消耗和取消的流程

由于产品购买结果模型发生了变化，因此购买的消耗和取消的逻辑也随之发生了变化。

如果出现以下情况，应使用取消购买方法（deletePurchase）：

1. 获取购买列表的方法（getPurchases）返回的购买状态为PurchaseState.CREATED或PurchaseState.INVOICE_CREATED。

   在某些情况下，通过银行应用程序（如快速支付系统(SBP)、SberPay、TinkoffPay等）支付后，当用户随后返回到 AnyApp 应用程序时，购买的状态可能仍然是 PurchaseState.INVOICE_CREATED。这与银行端处理购买的时间有关。因此，开发者需要正确地将获取购买列表的逻辑与屏幕生命周期相结合。解决这个问题的另一种方法是，只有在用户与应用程序交互时，才取消处于 PurchaseState.INVOICE_CREATED 状态的购买。例如，可以将这个逻辑放到一个单独的按钮中。

2. 购买方法 (purchaseProduct) 返回 PaymentResult.Cancelled。

3. 购买方法 (purchaseProduct) 返回 PaymentResult.Failure。

如果出现以下情况，应使用产品消耗方法（confirmPurchase）：

1. 获取购买列表的方法 (getPurchases) 返回了类型为 CONSUMABLE 且状态为 PurchaseState.PAID 的购买。

Java

快速启动

实现示例

查看示例应用程序（在Kotlin上），了解如何正确集成支付。

支付条件

为了进行支付，必须遵守所有条件：

1. 用户的设备上安装了RuStore应用程序。
2. 用户在RuStore应用程序中已授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 在RuStore 控制台系统中为应用程序启用了购买功能。

该服务在俄罗斯联邦以外有一些运作限制。

添加存储库

连接存储库：

repositories { ``maven { ``url "https://artifactory-external.vkpartner.ru/artifactory/maven" ``}}

集成依赖项

在您的配置文件中添加以下代码以连接依赖项：

dependencies { ``implementation 'ru.rustore.sdk:billingclient:3.1.0'}

要了解连接依赖项的详细信息，请参阅相关信息。

项目中的库初始化

在调用库的方法之前，需要先进行其初始化。

使用 RuStoreBillingClientFactory.create() 创建 RuStoreBillingClient：

创建 **RuStoreBillingClient**

finalContextcontext= getContext();final String consoleApplicationId = "111111" ;final String deeplinkScheme = "yourappscheme" ; // 可选参数final BillingClientThemeProvider themeProvider = null ;final boolean debugLogs = false ;final ExternalPaymentLoggerFactory externalPaymentLoggerFactory = null ; ``RuStoreBillingClient billingClient = RuStoreBillingClientFactory.INSTANCE.create( ``context, ``consoleApplicationId, ``deeplinkScheme, ``// 可选参数``themeProvider, ``debugLogs, ``externalPaymentLoggerFactory ``);

- context—Android 的上下文。可以是任何上下文，在实现中使用 applicationContext。
- consoleApplicationId—来自RuStore开发者控制台的应用程序代码(例如：https://console.rustore.ru/apps/111111)。
- deeplinkScheme—用于通过第三方应用程序支付后返回您的应用程序的deeplink方案(例如通过 SberPay 或快速支付系统(SBP))。SDK 会为这个方案生成它自己的宿主。
- themeProvider—提供 BillingClientTheme 主题的接口。有2种 BillingClientTheme 主题实现：亮色(Light)和暗色(Dark)。这个接口是可选的，默认情况下使用亮色主题。
- externalPaymentLoggerFactory—提供对外部日志器访问的接口。事件日志记录
- debugLogs—控制日志记录的标志(Release 构建将自动关闭日志)。事件日志记录

重要提示。在 build.gradle 中指定的 ApplicationId 必须与您在 RuStore 控制台系统中发布的 apk 文件的 applicationId 匹配。

传递给 deeplinkScheme 的 deeplink 方案必须与 AndroidManifest.xml 中处理 deeplink部分指定的方案匹配。

keystore 的签名必须与发布在 RuStore 控制台系统中的应用程序签名匹配。确保使用的 buildType(debug)使用与发布的应用程序相同的签名。

该库支持事件日志记录，可以在库初始化时单独连接。

要了解库初始化的详细信息，请查阅相关信息。

处理deeplink

为了在通过第三方应用程序（如快速支付系统(SBP)、SberPay 等）支付后返回您的应用程序，您需要正确实现 deeplink 的处理。在 AndroidManifest.xml 中指定带有您项目 intent-filter 的 scheme：

<activity ``android:name= ".YourBillingActivity" > ``<intent-filter> ``<action android:name= "android.intent.action.MAIN" /> ``<category android:name= "android.intent.category.LAUNCHER" /> ``</intent-filter> ``<intent-filter> ``<action android:name= "android.intent.action.VIEW" /> ``<category android:name= "android.intent.category.DEFAULT" /> ``<category android:name= "android.intent.category.BROWSABLE" /> ``<data android:scheme= "yourappscheme" /> ``</intent-filter></activity>

其中“yourappscheme”—是您的 deeplink 方案，可以更改为其他方案。这个方案必须与传递给 init() 的 deeplinkScheme 参数匹配。

为了成功返回应用程序，您还需要添加以下代码：

public class YourBillingActivityextends AppCompatActivity { ``// Previously created with RuStoreBillingClientFactory.create(); ``RuStoreBillingClient billingClient = YourDependencyInjection.getBillingClient(); ``@Override ``public void onCreate(@Nullable Bundle savedInstanceState) { ``super .onCreate(savedInstanceState); ``if (savedInstanceState == null) { ``billingClient.onNewIntent(getIntent()); ``} ``} ``@Override ``protected void onNewIntent(Intent intent) { ``super .onNewIntent(intent); ``billingClient.onNewIntent(intent); ``} }

要了解有关处理 deeplink 的详细信息，请查阅相关信息。

获取购买列表

RuStore Billing SDK要求正确处理购买状态，以提供最佳使用流程。因此，已购买的消耗性产品需要消耗掉，未完成的购买应 — 取消，以便能重新开始新的购买。

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchases().addOnSuccessListener(purchases -> { ``for (Purchase purchase: purchases) { ``if (purchase.getPurchaseId() != null) { ``if (purchase.getPurchaseState() == PurchaseState.CREATED || purchase.getPurchaseState() == PurchaseState.INVOICE_CREATED) { ``purchasesUseCase.deletePurchase(purchase.getPurchaseId()); ``} else if (purchase.getPurchaseState() == PurchaseState.PAID) { ``purchasesUseCase.confirmPurchase(purchase.getPurchaseId()); ``} ``} ``}});

在某些情况下，通过银行应用程序（如快速支付系统(SBP)、SberPay、TinkoffPay等）支付后，当用户随后返回到 AnyApp 应用程序时，购买的状态可能仍然是 PurchaseState.INVOICE_CREATED。这与银行端处理购买的时间有关。因此，开发者需要正确地将获取购买列表的逻辑与屏幕生命周期相结合。

解决这个问题的另一种方法是，只有在用户与应用程序交互时，才取消处于PurchaseState.INVOICE_CREATED 状态的购买。例如，可以将这个逻辑放到一个单独的按钮中。

要了解关于处理购买状态的详细信息，请参阅相关信息。

要了解获取购买列表的详细信息，请参阅相关信息。

购买产品

处理购买结果的步骤如下：

```
private void purchaseProduct(Product product) {  ``PurchasesUseCase purchasesUseCase = billingClient.getPurchases();  ``purchasesUseCase.purchaseProduct(product.getProductId())  ``.addOnSuccessListener(paymentResult -> handlePaymentResult(paymentResult, product))  ``.addOnFailureListener(throwable -> {  /* Handle error */ });  ``}private void handlePaymentResult(PaymentResult paymentResult, Product product) {  ``PurchasesUseCase purchasesUseCase = billingClient.getPurchases();  ``if (paymentResult  instanceof PaymentResult.Cancelled) {  ``String purchaseId = ((PaymentResult.Cancelled) paymentResult).getPurchaseId();  ``purchasesUseCase.deletePurchase(purchaseId);  ``}  else if (paymentResult  instanceof PaymentResult.Success) {  ``PaymentResult.Success purchaseResult = ((PaymentResult.Success) paymentResult);  ``purchasesUseCase.confirmPurchase(purchaseResult.getPurchaseId());  ``}  else if (paymentResult  instanceof PaymentResult.Failure) {  ``String purchaseId = ((PaymentResult.Failure) paymentResult).getPurchaseId();  ``if (purchaseId !=  null ) {  ``purchasesUseCase.deletePurchase(purchaseId);  ``}  ``}}
```

要了解关于产品购买的详细信息，请参阅相关信息。

要了解关于处理购买状态的详细信息，请参阅相关信息。

概述

您可以查阅快速集成应用程序内支付的相关部分。

实现示例

查看示例应用程序，了解如何正确集成支付。

SDK方法返回Task，这是一个异步任务，在相应的callback中返回错误或值。请记住这一点，在调用callback以同步获取结果时，应该从后台线程中使用它们。

支付条件

为了进行支付，必须遵守所有条件：

1. 用户的设备上安装了RuStore应用程序。

2. 用户在RuStore应用程序中已授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 在RuStore 控制台系统中为应用程序启用了购买功能。

该服务在俄罗斯联邦以外有一些运作限制。

添加存储库

连接存储库：

repositories { ``maven { ``url "https://artifactory-external.vkpartner.ru/artifactory/maven" ``}}

集成依赖项

在您的配置文件中添加以下代码以连接依赖项：

—

dependencies { ``implementation 'ru.rustore.sdk:billingclient:3.1.0'}

初始化

在调用库的方法之前，需要先进行其初始化。

使用 RuStoreBillingClientFactory.create() 创建 RuStoreBillingClient：

创建 **RuStoreBillingClient**

final Context context = getContext();final String consoleApplicationId = "111111" ;final String deeplinkScheme = "yourappscheme" ;// 可选参数final BillingClientThemeProvider themeProvider = null ;final boolean debugLogs = false ;final ExternalPaymentLoggerFactory externalPaymentLoggerFactory = null ; ``RuStoreBillingClient billingClient = RuStoreBillingClientFactory.INSTANCE.create( ``context, ``consoleApplicationId, ``deeplinkScheme, ``// 可选参数``themeProvider, ``debugLogs, ``externalPaymentLoggerFactory ``);

- context—Android 的上下文。可以是任何上下文，在实现中使用 applicationContext。
- consoleApplicationId—来自RuStore开发者控制台的应用程序代码(例如：https://console.rustore.ru/apps/111111)。
- deeplinkScheme—用于通过第三方应用程序支付后返回您的应用程序的deeplink方案(例如通过 SberPay 或快速支付系统(SBP))。SDK 会为这个方案生成它自己的宿主。
- themeProvider—提供 BillingClientTheme 主题的接口。有2种 BillingClientTheme 主题实现：亮色(Light)和暗色(Dark)。这个接口是可选的，默认情况下使用亮色主题。
- externalPaymentLoggerFactory—提供对外部日志器访问的接口。事件日志记录
- debugLogs—控制日志记录的标志(Release 构建将自动关闭日志)。事件日志记录

重要提示。在 build.gradle 中指定的 ApplicationId 必须与您在 applicationIdRuStore 控制台系统中发布的 apk 文件的 匹配。

传递给 deeplinkScheme 的 deeplink 方案必须与 AndroidManifest.xml 中处理 deeplink部分指定的方案匹配。

keystore 的签名必须与发布在 RuStore 控制台系统中的应用程序签名匹配。确保使用的 buildType（例如，debug）使用与发布的应用程序（例如，release）相同的签名。

处理deeplink

为了通过第三方应用程序（如快速支付系统（SBP），SberPay等）正确进行支付，您需要正确实现 deeplink 的处理。在 AndroidManifest.xml 中指定带有您项目 scheme 的 intent-filter：

< activity ``android:name = ".YourBillingActivity" > ``< intent-filter > ``< action android:name = "android.intent.action.MAIN" /> ``< category android:name = "android.intent.category.LAUNCHER" /> ``</ intent-filter > ``< intent-filter > ``< action android:name = "android.intent.action.VIEW" /> ``< category android:name = "android.intent.category.DEFAULT" /> ``< category android:name = "android.intent.category.BROWSABLE" /> ``< data android:scheme = "yourappscheme" /> ``</ intent-filter ></ activity >

其中“yourappscheme”—是您的 deeplink 方案，可以更改为其他方案。

这个方案必须与传递给 init() 的 deeplinkScheme 参数匹配。

接下来，在需要在支付后返回的 Activity（您的商店页面）中，您需要添加：

`public` `class` `YourBillingActivityextends AppCompatActivity {`
` `
` // Previously created with RuStoreBillingClientFactory.create();<br> RuStoreBillingClient billingClient = YourDependencyInjection.getBillingClient();`
` `
` @Override<br> public` `void` `onCreate(` `@Nullable` `Bundle savedInstanceState) {`
` super.onCreate(savedInstanceState);<br> if `(savedInstanceState == ``null` `) {`<br>` billingClient.onNewIntent(getIntent());
 }`<br>` }
 @Override`<br>` protectedvoidonNewIntent(Intent intent) {
 super` `.onNewIntent(intent);`<br>` billingClient.onNewIntent(intent);
 `} }

为了在通过 deeplink 返回时恢复您的应用程序状态，您需要在 AndroidManifest.xml 中添加：

android:launchMode=“singleTask”

<activity ``android:name= ".YourBillingActivity" ``android:launchMode= "singleTask" ``android:exported= "true" ``android:screenOrientation= "portrait" ``android:windowSoftInputMode= "adjustResize" >

检查支付功能的可用性

要检查支付功能的可用性，请调用 checkPurchasesAvailability 方法。在调用时，将检查以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持支付功能。
3. 用户必须在 RuStore 中获得授权。

4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序必须在 RuStore 控制台系统中启用购买功能。

如果所有条件都满足，则返回 FeatureAvailabilityResult.Available。否则，返回 FeatureAvailabilityResult.Unavailable(val cause: RuStoreException)，其中 cause — 是未满足条件的错误。所有可能的 RuStoreException 错误都被描述在错误处理部分。其他错误（例如，“没有互联网连接”）将在 onFailure 中返回。

```
RuStoreBillingClientExtKt.checkPurchasesAvailability(RuStoreBillingClient.Companion, getContext())  ``.addOnSuccessListener(result -> {  ``if (result  instanceof FeatureAvailabilityResult.Available) {  ``// Hanlde purchases available  ``} else {  ``RuStoreException exception = ((FeatureAvailabilityResult.Unavailable) result).getCause();  ``// Hanlde purchases unavailable  ``}  ``})  ``.addOnFailureListener(error -> {  ``// Handle error  ``});
```

获取产品列表

要获取产品，使用getProducts方法：

```
ProductsUseCase productsUseCase = billingClient.getProducts();productsUseCase.getProducts(Arrays.asList( "id1" , "id2" )).addOnCompleteListener( new OnCompleteListener<List<Product>>() {  ``@Override  ``public void onFailure( @NonNull Throwable throwable) {  ``// Process error  ``}  ``@Override  ``public void onSuccess(List<Product> products) {  ``// Process success  ``}});
```

- productIds: List — 产品标识符列表。

产品结构：

```
interface Product {  ``public String getProductId();  ``@Nullable  ``public ProductType getProductType();  ``public ProductStatus getProductStatus();  ``@Nullable  ``public String getPriceLabel();  ``@Nullable  ``public Integer getPrice();  ``@Nullable  ``public String getCurrency();  ``@Nullable  ``public String getLanguage();  ``@Nullable  ``public String getTitle();  ``@Nullable  ``public String getDescription();  ``@Nullable  ``public Uri getImageUrl();  ``@Nullable  ``public Uri getPromoImageUrl();  ``@Nullable  ``public ProductSubscription getProductSubscription();}
```

- getProductId()—产品标识符；
- getProductType()—产品类型；
- getProductStatus()—产品状态；
- getPriceLabel()—格式化的产品价格，包括[language]语言的货币符号；
- getPrice()—以最小单位（戈比）表示的价格；
- getCurrency()—ISO 4217货币代码；
- getLanguage()—使用 BCP 47 编码指定的语言；
- getTitle()—产品名称，以[language]语言表示；

- getDescription()—产品描述，以[language]语言表示；
- getImageUrl()—图片链接；
- getPromoImageUrl()—促销图片链接；
- getSubscription()—订阅描述，仅对于类型为 subscription 的产品返回。

订阅结构：

订阅结构

interface ProductSubscription { ``@Nullable ``public SubscriptionPeriod getSubscriptionPeriod(); ``@Nullable ``public SubscriptionPeriod getFreeTrialPeriod(); ``@Nullable ``public SubscriptionPeriod getGracePeriod(); ``@Nullable ``public String getIntroductoryPrice(); ``@Nullable ``public String getIntroductoryPriceAmount(); ``@Nullable ``public SubscriptionPeriod getIntroductoryPricePeriod();}

- getSubscriptionPeriod()—订阅周期；
- getFreeTrialPeriod()—订阅试用期；
- getGracePeriod()—订阅宽限期；
- getIntroductoryPrice()—格式化的介绍性订阅价格，包括product:language语言的货币符号；
- getIntroductoryPriceAmount()—以货币最小单位（戈比）表示的介绍性价格；
- getIntroductoryPricePeriod()—介绍性价格计算周期。

订阅周期结构：

interface SubscriptionPeriod { ``int getYears(); ``int getMonths(); ``int getDays();}

- getYears()—年数；
- getMonths()—月数；
- getDays()—天数。

获取购买列表

要获取购买列表，请使用getPurchases方法：

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchases().addOnCompleteListener(new OnCompleteListener<List<Purchase>>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(List<Purchase> purchases) { ``// Process success ``}});

购买结构：

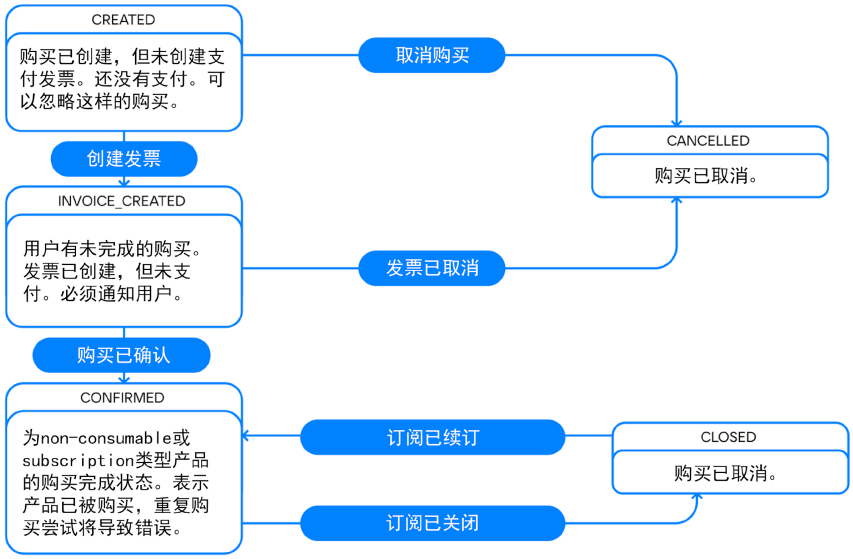
interface Purchase { ``@Nullable ``public String getPurchaseId(); ``public String getProductId(); ``@Nullable ``public ProductType getProductType(); ``@Nullable ``public String getInvoiceId(); ``@Nullable ``public String getDescription(); ``@Nullable ``public String getLanguage(); ``@Nullable ``public Date getPurchaseTime(); ``@Nullable ``public String getOrderId(); ``@Nullable ``public String getAmountLabel(); ``@Nullable ``public Integer getAmount(); ``@Nullable ``public String getCurrency(); ``@Nullable ``public Integer

getQuantity(); ``@Nullable ``public PurchaseState getPurchaseState(); ``@Nullable ``public String getDeveloperPayload(); ``@Nullable ``public String getSubscriptionToken();}

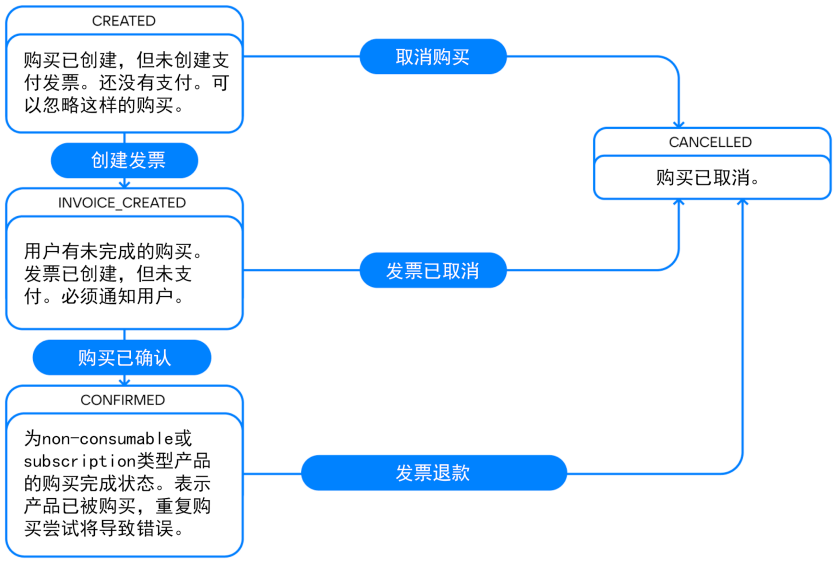
- getPurchaseId()—购买标识符。
- getProductId()—产品标识符。
- getProductType()—产品类型。
- getInvoiceId()—账单标识符。
- getDescription()—购买描述。
- getLanguage()—使用 BCP 47 编码指定的语言。
- getPurchaseTime()—购买时间（RFC 3339 格式）。
- getOrderId()—应用程序生成的唯一支付标识符（uuid）。
- getAmountLabel()—格式化的购买价格，包括[language]语言的货币符号。
- getAmount()—以最小货币单位表示的价格。
- getCurrency()—ISO 4217 货币代码。
- getQuantity()—产品数量。
- getPurchaseState()—购买状态：
- 购买状态的可能值：
  - CREATED—已创建；
  - INVOICE_CREATED—已创建，等待支付；
  - CONFIRMED—已确认；
  - PAID—已支付；
  - CANCELLED—购买已取消；
  - CONSUMED—已确认购买消耗；
  - CLOSED—订阅已取消。
- getDeveloperPayload()—开发者指定的字符串，包含关于订单的附加信息；
- getSubscriptionToken()—用于在服务器端验证购买的令牌。

状态模型 **(purchaseState)**：

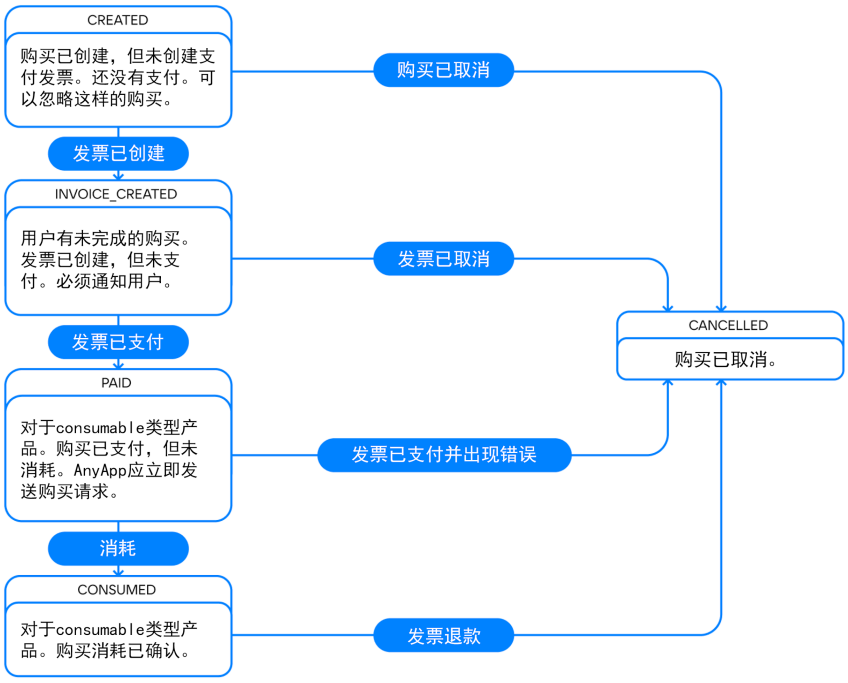
订阅购买状态模型 (SUBSCRIPTIONS)：



非消耗性产品购买状态模型 (NON-CONSUMABLES)：



消耗性产品购买状态模型 (CONSUMABLES)：



获取购买信息

要获取关于购买的信息，请使用 getPurchaseInfo 方法：

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchaseInfo("purchaseId").addOnCompleteListener(new OnCompleteListener<Purchase>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(Purchase result) { ``// Process success ``}});

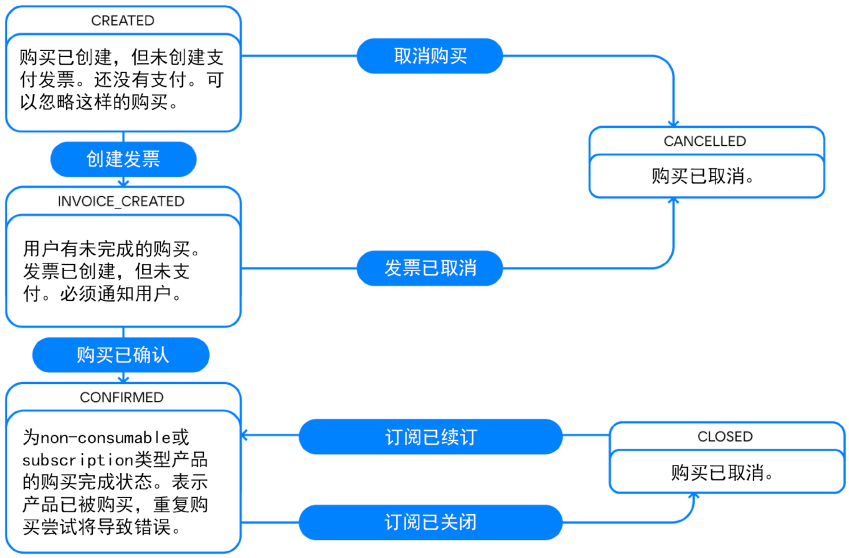
购买结构：

```
interface Purchase {    ``@Nullable    ``public String getPurchaseId();    ``public String getProductId();    ``@Nullable    ``public ProductType getProductType();    ``@Nullable    ``public String getInvoiceId();    ``@Nullable    ``public String getDescription();    ``@Nullable    ``public String getLanguage();    ``@Nullable    ``public Date getPurchaseTime();    ``@Nullable    ``public String getOrderId();    ``@Nullable    ``public String getAmountLabel();    ``@Nullable    ``public Integer getAmount();    ``@Nullable    ``public String getCurrency();    ``@Nullable    ``public Integer getQuantity();    ``@Nullable    ``public PurchaseState getPurchaseState();    ``@Nullable    ``public String getDeveloperPayload();    ``@Nullable    ``public String getSubscriptionToken();}
```

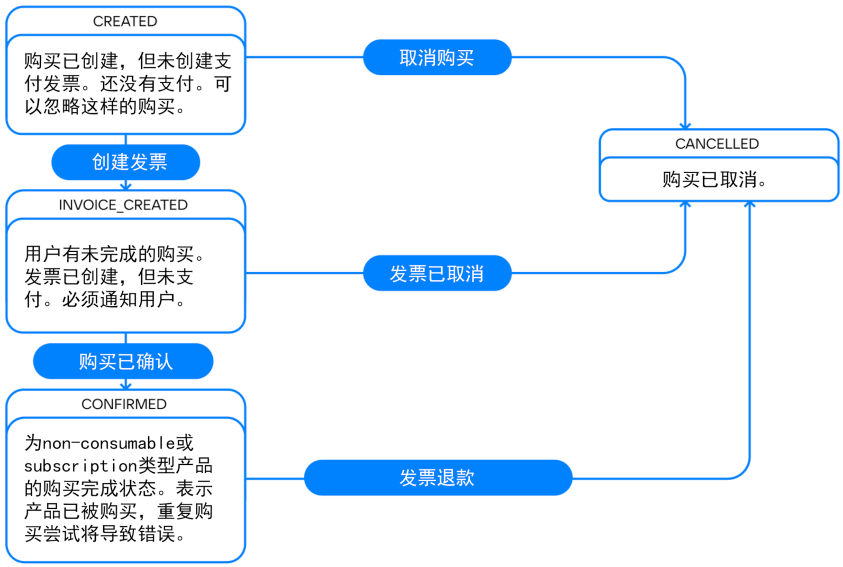
- getPurchaseId()—购买标识符；
- getProductId()—产品标识符；
- getProductType()—产品类型；
- getInvoiceId()—账单标识符；
- getDescription()—购买描述；
- getLanguage()—使用 BCP 47 编码指定的语言；
- getPurchaseTime()—购买时间（RFC 3339 格式）；
- getOrderId()—应用程序生成的唯一支付标识符（uuid）；
- getAmountLabel()—格式化的购买价格，包括[language]语言的货币符号；
- getAmount()—以最小货币单位表示的价格；
- getCurrency()—ISO 4217货币代码；
- getQuantity()—产品数量；
- getPurchaseState()—购买状态：
  - 购买状态的可能值：
    - CREATED—已创建；
    - INVOICE_CREATED—已创建，等待支付；
    - CONFIRMED—已确认；
    - PAID—已支付；
    - CANCELLED—购买已取消；
    - CONSUMED—已确认购买消耗；
    - CLOSED—订阅已取消。
- getDeveloperPayload()—开发者指定的字符串，包含关于订单的附加信息；
- getDeveloperPayload()—用于在服务器端验证购买的令牌。

状态模型 **(purchaseState)**：

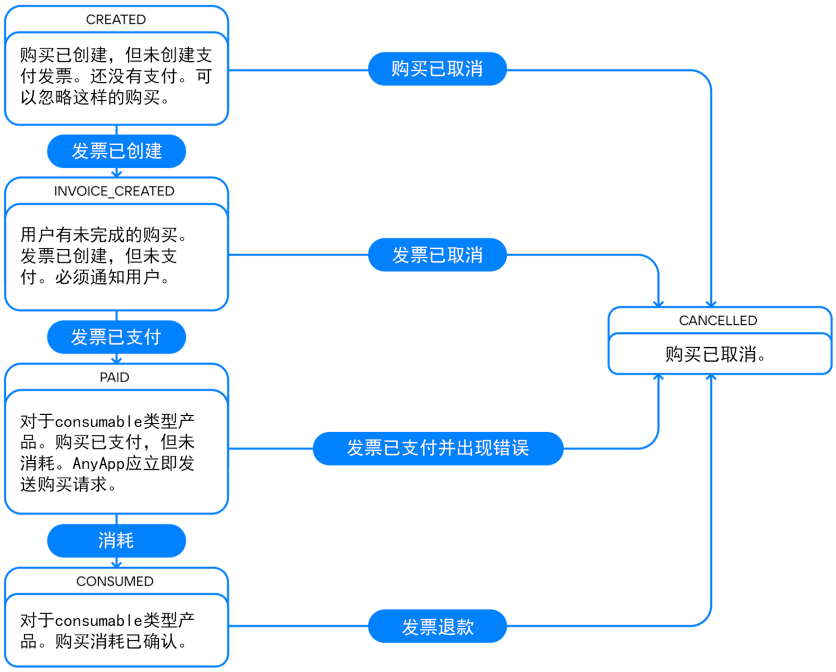
订阅购买状态模型 (SUBSCRIPTIONS)：



非消耗性产品购买状态模型 (NON-CONSUMABLES)：



消耗性产品购买状态模型 (CONSUMABLES)：



购买产品

要调用购买产品，请使用 purchaseProduct 方法：

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.purchaseProduct("productId").addOnCompleteListener(new OnCompleteListener<PaymentResult>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(PaymentResult result) { ``// Process PaymentResult ``}});

- productId：String — 产品标识符；
- orderId: String — 订单标识符，在AnyApp端创建（可选。如果未指定，则会自动生成）；
- quantity: Int — 产品数量（可选）；
- developerPayload—AnyApp 开发者的附加信息（可选）。

购买结果的结构：

interface PaymentResult { ``interface Success extends PaymentResult { ``@Nullable ``public String getOrderId(); ``public String getPurchaseId(); ``public String getProductId(); ``public String getInvoiceId(); ``@Nullable ``public String getSubscriptionToken(); ``} ``interface Failure extends PaymentResult { ``@Nullable ``public String getPurchaseId(); ``@Nullable ``public String getInvoiceId(); ``@Nullable ``public String getOrderId(); ``@Nullable ``public Integer getQuantity(); ``@Nullable ``public String getProductId(); ``@Nullable ``public Integer getErrorCode(); ``} ``interface Cancelled extends PaymentResult { ``public String getPurchaseId(); ``} ``interface InvalidPaymentState extends PaymentResult {}}

- Success—成功完成数字产品购买的结果；

- Failure—购买数字产品时错误的结果；
- Cancelled—数字产品购买被取消的结果；
- InvalidPaymentState—支付SDK操作错误。可能因deeplink回调不正确而发生。

购买的服务器端验证

如果您需要在服务器端验证成功的购买，您可以使用 subscriptionToken 成功购买产品时返回的 PurchaseResult 中的 purchaseProduct。

SubscriptionToken由购买的invoiceId和RuStore的userId组成，用点书写：“*invoiceId*.userId”。

—

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.purchaseProduct(productId).addOnSuccessListener(paymentResult -> {  ``if (paymentResult  instanceof PaymentResult.Success) {  ``String subscriptionToken = ((PaymentResult.Success) paymentResult).getSubscriptionToken();  ``yourApi.validate(subscriptionToken);  ``}});
```

您还可以从 Purchase 实体中获取 subscriptionToken。可以通过使用 Purchase 方法来获取 getPurchases() 实体。

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchases().addOnSuccessListener(purchases -> {  ``for (Purchase purchase : purchases) {  ``yourApi.validate(purchase.getSubscriptionToken());  ``}});
```

购买消耗（确认）

RuStore 包含以下类型的产品：

- CONSUMABLE—消耗的（可以多次购买，例如应用中的水晶）；
- NON_CONSUMABLE—非消耗的（只能购买一次，例如禁用应用程序中的广告）；
- SUBSCRIPTION—订阅（可以购买一段时间，例如流媒体服务的订阅）。

只有处于 PurchaseState.PAID 状态的 CONSUMABLE 类型产品需要消耗。

您可以使用 confirmPurchase 方法来消耗购买：

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.confirmPurchase( "purchaseId" , "developerPayload" ).addOnCompleteListener( new OnCompleteListener<Unit>() {  ``@Override  ``public void onFailure( @NonNull Throwable throwable) {  ``// Process error  ``}  ``@Override  ``public void onSuccess(Unit result) {  ``// Process success  ``}});
```

- purchaseId—购买标识符。
- developerPayload—AnyApp 开发者的附加信息（可选）。

取消购买

要取消购买，您可以使用 deletePurchase 方法：

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.deletePurchase("purchaseId").addOnCompleteListener(new OnCompleteListener<Unit>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(Unit result) { ``// Process success ``}});

- purchaseId—购买标识符。

如果您有与删除购买相关的逻辑，请使用此方法。购买将在20分钟后自动取消，或者在同一客户端进行重复购买时取消。

购买消耗和取消的*流程*

未完成的支付处理应由 AnyApp 的开发者执行。

如果出现以下情况，应使用取消购买方法（deletePurchase）：

1. 获取购买列表的方法（getPurchases）返回的购买状态为PurchaseState.CREATED或PurchaseState.INVOICE_CREATED。

   在某些情况下，通过银行应用程序（如快速支付系统(SBP)、SberPay、TinkoffPay等）支付后，当用户随后返回到 AnyApp 应用程序时，购买的状态可能仍然是PurchaseState.INVOICE_CREATED。这与银行端处理购买的时间有关。因此，开发者需要正确地将获取购买列表的逻辑与屏幕生命周期相结合。解决这个问题的另一种方法是，只有在用户与应用程序交互时，才取消处于 PurchaseState.INVOICE_CREATED 状态的购买。例如，可以将这个逻辑放到一个单独的按钮中。

2. 购买方法 (purchaseProduct) 返回 PaymentResult.Cancelled。
3. 购买方法 (purchaseProduct) 返回 PaymentResult.Failure。

如果获取购买列表的方法 (getPurchases) 返回了类型为 CONSUMABLE 且状态为 PurchaseState.PAID 的购买，则需要使用产品消耗方法 (confirmPurchase)。

事件日志记录

如果您想在支付库中日志记录事件，可以在调用 RuStoreBillingClient.init 时添加 externalPaymentLoggerFactory 和 debugLogs 参数（这些参数对初始化是可选的）：

final Context context = getContext();final String consoleApplicationId = "111111" ;final String deeplinkScheme = "yourappscheme" ;final ExternalPaymentLoggerFactory externalPaymentLoggerFactory = (tag) -> new PaymentLogger(tag);final boolean debugLogs = true ;RuStoreBillingClient billingClient = RuStoreBillingClientFactory.INSTANCE.create(``context, ``consoleApplicationId, ``deeplinkScheme, ``externalPaymentLoggerFactory, ``debugLogs);public class PaymentLogger implements ExternalPaymentLogger { ``private final String tag; ``public PaymentLogger(String tag) { ``this .tag = tag; ``} ``@Override ``public void d( @Nullable Throwable throwable, @NonNull Function0<String> function0) { ``Log.d(tag, function0.invoke()); ``} ``@Override ``public void e(

@Nullable Throwable throwable, @NonNull Function0<String> function0) { ``Log.e(tag, function0.invoke()); ``} ``@Override ``public void i(@Nullable Throwable throwable, @NonNull Function0<String> function0) { ``Log.i(tag, function0.invoke()); ``} ``@Override ``public void v(@Nullable Throwable throwable, @NonNull Function0<String> function0) { ``Log.v(tag, function0.invoke()); ``} ``@Override ``public void w(@Nullable Throwable throwable, @NonNull Function0<String> function0) { ``Log.w(tag, function0.invoke()); ``}}

启用日志记录的参数：

- externalPaymentLoggerFactory—允许创建将库日志传递到宿主应用的记录器的接口；
- debugLogs—启用日志记录（Release 构建会自动关闭日志）。

其中，PaymentLogger —是支付事件日志记录实现的一个示例。

更换主题

SDK 支持通过 BillingClientThemeProvider 提供者接口的动态主题更换：

final Context context = getContext();final String consoleApplicationId = "111111" ;final String deeplinkScheme = "yourappscheme" ;final BillingClientThemeProvider themeProvider = BillingClientThemeProviderImpl();RuStoreBillingClient billingClient = RuStoreBillingClientFactory.INSTANCE.create(``context, ``consoleApplicationId, ``deeplinkScheme, ``themeProvider); public class BillingClientThemeProviderImpl implements BillingClientThemeProvider { ``@NonNull ``@Override ``public BillingClientTheme provide() { ``// 检查已设置主题的逻辑应该放置于此处``boolean darkTheme = ...; ``if (darkTheme) { ``return BillingClientTheme.Dark; ``} else { ``return BillingClientTheme.Light; ``} ``}}

错误处理

可能的错误：

- RuStoreNotInstalledException() — 用户设备上未安装 RuStore；
- RuStoreOutdatedException() — 用户设备上安装的 RuStore 不支持支付；
- RuStoreUserUnauthorizedException() — 用户未在 RuStore 中授权；
- RuStoreApplicationBannedException() — 应用程序在 RuStore 中被禁止；
- RuStoreUserBannedException() — 用户在 RuStore 中被禁止；
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

调用 RuStoreBillingClient.purchases.purchaseProduct() 方法时，错误会自动处理。

要向用户显示错误对话框，您可以使用 resolveForBilling 方法：

BillingRuStoreExceptionExtKt.resolveForBilling(exception, getContext());

迁移到支付版本*2.2.0*

概述

在支付版本 2.2.0 中，PaymentResult 中的产品购买结果模型发生了显著变化。

为了快速且无大问题地迁移到 SDK 的新版本，请遵循此迁移指南。

更新依赖项

为了更新依赖项，请在您的 build.gradle 文件的 billingclient 区域中升级 dependencies 的版本：

build.gradle

dependencies { ``implementation("ru.rustore.sdk:billingclient:2.2.0")}

更改模型

获取产品列表

获取产品列表的结果模型已发生了变化。现在，**getProducts()** 方法直接返回产品列表：

ProductsUseCase productsUseCase = billingClient.getProducts();productsUseCase.getProducts(Arrays.asList("id1" , "id2")).addOnCompleteListener(new OnCompleteListener<List<Product>>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(List<Product> products) { ``// Process success ``}});

产品模型和错误模型保持不变。

获取购买列表

获取购买列表的结果模型发生了变化。现在，**getPurchases()** 方法会直接返回购买列表：

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchases().addOnCompleteListener(new OnCompleteListener<List<Purchase>>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(List<Purchase> purchases) { ``// Process success ``}});

购买模型和错误模型保持不变。

获取购买信息

获取购买信息的结果模型发生了变化。现在，**getPurchaseInfo()** 方法会直接返回购买模型：

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchaseInfo("purchaseId").addOnCompleteListener(new OnCompleteListener<Purchase>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(Purchase purchase) { ``// Process success ``}});

错误模型保持不变。

购买产品

购买产品的结果模型发生了变化。更新后的模型如下所示：

```
interface PaymentResult {        ``interface Success  extends PaymentResult
{        ``@Nullable        ``public String
getOrderId();                ``public String
```

```
getPurchaseId();                ``public String getProductId();        ``public
String getInvoiceId();                ``@Nullable        ``public String
getSubscriptionToken();     ``}        ``interface Failure  extends PaymentResult
{        ``@Nullable        ``public String
getPurchaseId();                ``@Nullable        ``public String
getInvoiceId();                ``@Nullable        ``public String
getOrderId();                ``@Nullable        ``public Integer
getQuantity();                ``@Nullable        ``public String
getProductId();        ``@Nullable        ``public Integer
getErrorCode();    ``}    ``interface Cancelled  extends PaymentResult
{        ``public String getPurchaseId();    ``}              ``interface
InvalidPaymentState  extends PaymentResult {}}
```

其中：

- Success—成功完成数字产品购买的结果。
- Failure—购买数字产品时错误的结果。
- Cancelled—数字产品购买被取消的结果。
- InvalidPaymentState—支付SDK操作错误。可能因deeplink回调不正确而发生。

特别注意购买消耗和取消的流程。

购买消耗（确认）

购买消耗的结果模型发生了变化。现在，消耗可能以成功或错误结束：

```
PurchasesUseCase purchasesUseCase =
billingClient.getPurchases();purchasesUseCase.confirmPurchase( "purchaseId" , "developerPayload"
).addOnCompleteListener( new OnCompleteListener<Unit>() {  ``@Override  ``public void onFailure(
@NonNull Throwable throwable) {    ``// Process error  ``}  ``@Override  ``public void
onSuccess(Unit result) {    ``// Process success  ``}});
```

取消购买

购买取消的结果模型发生了变化。现在，取消购买可能以成功或错误结束：

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.deletePurchase(
"purchaseId" ).addOnCompleteListener( new OnCompleteListener<Unit>() {  ``@Override  ``public
void onFailure( @NonNull Throwable throwable) {    ``// Process error  ``}  ``@Override  ``public
void onSuccess(Unit result) {    ``// Process success  ``}});
```

购买消耗和取消的流程

由于产品购买结果模型发生了变化，因此购买的消耗和取消的逻辑也随之发生了变化。

如果出现以下情况，应使用取消购买方法（deletePurchase）：

1. 获取购买列表方法（getPurchases）返回了具有以下状态的购买：

2. PurchaseState.CREATED.
3. PurchaseState.INVOICE_CREATED.
4. 购买方法 (purchaseProduct) 返回 PaymentResult.Cancelled。
5. 购买方法 (purchaseProduct) 返回 PaymentResult.Failure。

如果获取购买列表的方法 (confirmPurchase) 返回了类型为 getPurchases 且状态为 PurchaseState.PAID 的购买，则需要使用产品消耗方法 (CONSUMABLE)。

*SDK*支付从版本 *1.x.x* 迁移到版本 *3.x.x*

在支付版本 3.0.0 中，PaymentResult 中的产品购买结果模型发生了显著变化。

为了快速且无大问题地迁移到 SDK 的新版本，请遵循此迁移指南。

更新依赖项

为了更新依赖项，请在您的 build.gradle 文件的 billingclient 区域中升级 dependencies 的版本：

build.gradle

dependencies { ``implementation("ru.rustore.sdk:billingclient:3.1.0")}

更改模型

获取产品列表

获取产品列表的结果模型已发生了变化。现在，**getProducts()** 方法直接返回产品列表：

调用方法**getProducts**

ProductsUseCase productsUseCase = billingClient.getProducts();productsUseCase.getProducts(Arrays.asList("id1" , "id2")).addOnCompleteListener(new OnCompleteListener<List<Product>>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(List<Product> products) { ``// Process success ``}});

产品模型和错误模型保持不变。

获取购买列表

获取购买列表的结果模型发生了变化。现在，**getPurchases()** 方法会直接返回购买列表：

调用获取用户购买列表的方法

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchases().addOnCompleteListener(new OnCompleteListener<List<Purchase>>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(List<Purchase> purchases) { ``// Process success ``}});

购买模型和错误模型保持不变。

获取购买信息

获取购买信息的结果模型发生了变化。现在，**getPurchaseInfo()** 方法会直接返回购买模型：

调用获取用户购买列表的方法

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.getPurchaseInfo( "purchaseId" ).addOnCompleteListener( new OnCompleteListener<Purchase>() {  ``@Override  ``public void onFailure( @NonNull Throwable throwable) {  ``// Process error  ``}  ``@Override  ``public void onSuccess(Purchase purchase) {  ``// Process success  ``}});
```

错误模型保持不变。

购买产品

购买产品的结果模型发生了变化。更新后的模型如下所示：

购买结果的结构体

```
interface PaymentResult {  ``interface Success  extends PaymentResult {  ``@Nullable  ``public String getOrderId();  ``public String getPurchaseId();  ``public String getProductId();  ``public String getInvoiceId();  ``@Nullable  ``public String getSubscriptionToken();  ``}  ``interface Failure  extends PaymentResult {  ``@Nullable  ``public String getPurchaseId();  ``@Nullable  ``public String getInvoiceId();  ``@Nullable  ``public String getOrderId();  ``@Nullable  ``public Integer getQuantity();  ``@Nullable  ``public String getProductId();  ``@Nullable  ``public Integer getErrorCode();  ``}  ``interface Cancelled extends PaymentResult {  ``public String getPurchaseId();  ``}  ``interface InvalidPaymentState  extends PaymentResult {}}
```

其中：

- Success成功完成数字产品购买的结果。
- Failure购买数字产品时错误的结果。
- Cancelled数字产品购买被取消的结果。
- InvalidPaymentStateSDK支付错误。可能因deeplink回调不正确而发生。

需要特别注意消耗和取消购买的流程！

购买消耗（确认）

购买消耗的结果模型发生了变化。现在，消耗可能以成功或错误结束：

调用确认方法

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.confirmPurchase( "purchaseId" , "developerPayload" ).addOnCompleteListener( new OnCompleteListener<Unit>() {  ``@Override  ``public void onFailure(
```

@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(Unit result) { ``// Process success ``}});

取消购买

购买取消的结果模型发生了变化。现在，取消购买可能以成功或错误结束：

调用取消购买方法。

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();purchasesUseCase.deletePurchase("purchaseId").addOnCompleteListener(new OnCompleteListener<Unit>() { ``@Override ``public void onFailure(@NonNull Throwable throwable) { ``// Process error ``} ``@Override ``public void onSuccess(Unit result) { ``// Process success ``}});

购买消耗和取消的流程

由于产品购买结果模型发生了变化，因此购买的消耗和取消的逻辑也随之发生了变化。

如果出现以下情况，应使用取消购买方法（deletePurchase）：

1. 获取购买列表的方法（getPurchases）返回的购买状态为PurchaseState.CREATED或PurchaseState.INVOICE_CREATED。

在某些情况下，通过银行应用程序（如快速支付系统(SBP)、SberPay、TinkoffPay等）支付后，当用户随后返回到 AnyApp 应用程序时，购买的状态可能仍然是 PurchaseState.INVOICE_CREATED。这与银行端处理购买的时间有关。因此，开发者需要正确地将获取购买列表的逻辑与屏幕生命周期相结合。解决这个问题的另一种方法是，只有在用户与应用程序交互时，才取消处于 PurchaseState.INVOICE_CREATED 状态的购买。例如，可以将这个逻辑放到一个单独的按钮中。

2. 购买方法 (purchaseProduct) 返回 PaymentResult.Cancelled。
3. 购买方法 (purchaseProduct) 返回 PaymentResult.Failure。

如果出现以下情况，应使用产品消耗方法（confirmPurchase）：

1. 获取购买列表的方法 (getPurchases) 返回了类型为 CONSUMABLE 且状态为 PurchaseState.PAID 的购买。

*RuStore SDK*支付的更新历史

SDK版本2.2.0

- 添加了动态主题切换功能（亮色和暗色）；
- 对库的稳定性进行了优化；
- 修复了通过deeplink支付后恢复状态的问题。

SDK版本2.1.1

- 安全更新。

SDK版本2.1.0

- 更改了响应模型：
  - 获取产品列表

  - 获取购买列表
  - 购买产品
  - 消耗购买
  - 取消购买
- 支付对话框的外观进行了优化。

SDK版本1.1.1

- 修复了Task API的await()方法的操作。

SDK版本1.1.0

- 使用TinkoffPay进行支付。
- 支付时保存银行卡的功能。
- 支付对话框的外观和行为进行了优化。
- 移除了多余的依赖项和 uses-permissions
- PurchaseResult模型增加了新的账单标识符字段“invoiceId”。

SDK版本1.0.0

- 从单例模式转换到创建实例：RuStoreBillingClient.init() 更改为 RuStoreBillingClientFactory.create()。
- 与单例模式相关的方法（init, products, purchases, getSingleton）标记为 deprecated，并将在后续版本中移除。
- 关于迁移的更多详细信息，请参阅迁移指南：“到1.0.0版本支付的迁移”
- checkPurchasesAvailability()方法已变为静态 — 您无需创建 RuStoreBillingClient 实例即可检查付款的可用性。

SDK版本0.1.8

- 添加了方法RuStoreBillingClient.purchases.getPurchaseInfo(“purchaseId”)，用以返回购买信息。
- 修复了“Intent Redirection vulnerability”漏洞，该漏洞阻止应用程序在Google Play上发布。

SDK版本0.1.7

- SDK的内部更新。

SDK版本0.1.6

- 在Purchase实体中添加了subscriptionToken字段，用于服务器端购买验证。

SDK版本0.1.5

- 转换到俄数字部的证书。
- 修复了在快速切换到服务条款并返回时通过快速支付系统（SBP）支付按钮的显示问题。
- 在String接口中，将traceId: ResponseWithCode字段替换为包含traceId的meta: RequestMeta字段。

SDK版本0.1.4

- 为快速支付系统（SBP）和通过手机账户支付添加了服务条款链接。
- 如果支付确认超过15秒，为用户添加了额外的免责声明。

- 修复了与OTP代码操作和发送短信时的错误。
- 修复了在RuStore端为发票生成orderId字段的问题，如果在orderId中将purchaseProduct()传递为null。
- 如果银行应用不支持快速支付系统(SBP)的deeplink，添加了显示阻塞错误的功能。
- 添加了PurchaseResult.subscriptionToken，允许在服务器端进行购买验证。

SDK版本0.1.3

- 在purchaseProduct()中添加了通过电话号码进行支付的功能。
- 在初始化过程中，将参数externalPaymentLogger更换为externalPayemtnLoggerFactory— 这是一个工厂，在其中需要创建ExternalPaymentLogger的实现(参见日志记录部分)。

SDK版本0.1.2

- 添加了快速支付系统(SBP)(快速支付系统(SBP))。
- 在init方法中添加了deeplinkScheme参数。
- 在init方法中删除了deeplinkPrefix参数。
- 修复了应用程序最小化后，在输入信用卡数据的表单中数据丢失的错误。

SDK版本0.1.1

- 在以下方法中删除了“language”参数：
  - 获取产品列表 — «getProducts»。
  - 获取购买列表 — «getPurchases»。
  - 产品消耗 — «confirmPurchase»。
  - 取消购买 — «deletePurchase»。
- 添加了RuStoreBillingClient.isInitialized字段，用于返回库初始化的状态。

SDK版本0.1.0

- Init中的参数发生了变化。
- 添加了检查支付功能可用性的方法 — “checkPurchasesAvailability”。
- 所有方法中的“language”参数变为可选。
- SuspendSuspend方法在以下方法中被task API替换：
  - 获取产品列表 — «getProducts»。
  - 获取购买列表 — «getPurchases»。
  - 购买产品 — «purchaseProduct»。
  - 产品消耗 — «confirmPurchase»。
  - 取消购买 — «deletePurchase»。
- 在“purchaseProduct”购买方法中删除了“context”参数。
- 删除了“resultObserver”方法，现在购买结果通过“purchaseProduct”返回。
- 在“resolveForBilling”方法中删除了“onFail”参数。

SDK版本0.0.9

- 在init函数中添加了“ExternalPaymentLogger”和“debugLogs”可选参数。

Unity

概述

通过以下链接下载Unity插件以集成支付功能。

为了进行支付，必须遵守所有条件：

1. 用户的设备上安装了RuStore应用程序。
2. 用户在RuStore中已授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 应用程序必须在 RuStore 控制台系统中启用购买功能。

该服务在俄罗斯联邦以外有一些运作限制。

在项目中集成

要集成，请下载RuStoreBilling SDK并将其导入项目（Assets → Import Package → Custom Package）。依赖项将通过SDK中包含的External Dependency Manager自动连接。

为了正确处理SDK的依赖项，需要设置以下参数：

- Edit-> Project Settings -> Player Settings -> Publishing Settings，启用Custom Main Gradle Template和CustomGradlePropertiesTemplate
- Assets-> External Dependencies Manager -> Android Resolver -> Settings，启用Use Jetifier, Patch mainTemplate.gradle，PatchgradleTemplate.properties

完成设置后，请确保执行Assets -> External Dependencies Manager -> Android Resolver -> Force Resolve。

MinimumAPIlevel应设置为不低于24。应用程序的缩减（ProGuard/R8）目前不受支持，因此需要在项目设置中将其关闭（File → Build Settings → Player Settings → Publishing Settings → Minify）。

处理deeplink

当用户通过SberPay支付时，会跳转到俄罗斯储蓄银行应用程序。支付完成后，应使用deeplink返回原应用程序。要处理deeplink，需在AndroidManifest.xml中设置带有您项目scheme的intent-filter：

< activity ``android:name = "ru.rustore.unitysdk.RuStoreUnityActivity" android:theme = "@style/UnityThemeSelector" android:exported = "true" > ``< intent-filter > ``< action android:name = "android.intent.action.MAIN" /> ``< category android:name = "android.intent.category.LAUNCHER" /> ``</ intent-filter > ``< intent-filter > ``< action android:name = "android.intent.action.VIEW" /> ``< category android:name = "android.intent.category.DEFAULT" /> ``< category android:name =

"android.intent.category.BROWSABLE" /> ``< data android:scheme = "yourappscheme" /> ``</ intent-filter ></ activity >

其中“yourappscheme”—您的deeplink方案，可以更改为其他方案。

接下来，扩展UnityPlayerActivity类，并在onNewIntent中添加对传入intent的处理。

package ru.rustore.unitysdk;import android.os.Bundle;import android.content.Intent;import ru.rustore.unitysdk.billingclient.RuStoreUnityBillingClient;import com.unity3d.player.UnityPlayerActivity;public class RuStoreUnityActivity extends UnityPlayerActivity { ``@Override protected void onCreate(Bundle savedInstanceState) { ``super .onCreate(savedInstanceState); ``if (savedInstanceState == null ) { ``RuStoreUnityBillingClient.onNewIntent(getIntent()); ``} ``} ``@Override protected void onNewIntent(Intent intent) { ``super .onNewIntent(intent); ``RuStoreUnityBillingClient.onNewIntent(intent); ``}}

将包含UnityPlayerActivity扩展代码的Java文件放置在项目的Assets文件夹中。如果已有UnityPlayerActivity扩展，需合并onCreate和onNewIntent功能代码至该扩展。

初始化

在调用库的方法之前，需要先进行其初始化。库将使用在Unity编辑器中设置的参数进行初始化。在编辑器菜单中选择 Window → RuStoreSDK → Settings → Billing Client。

RuStoreBillingClient.Instance.Init();

如果需要使用其他设置进行初始化，可以直接从代码中传递这些设置：

var config = new RuStoreBillingClientConfig() { ``consoleApplicationId = "11111", ``deeplinkPrefix = "yourappscheme", ``allowNativeErrorHandling = true, enableLogs = true};RuStoreBillingClient.Instance.Init(config);

- consoleApplicationId—来自RuStore 控制台的应用程序代码。（例如：https://console.rustore.ru/apps/111111）
- deeplinkPrefix—url用于实现deeplink的url。可以使用任何唯一名称作为名称（例如：yourappscheme）
- allowAllowNativeErrorHandling — 允许原生SDK处理错误（参见错误处理部分）；
- enableLogs—启用日志。

重要的是，传递给deeplinkPrefix的deeplink方案必须与AndroidManifest.xml中指定的方案相匹配。

如果需要检查库是否已初始化，请使用RuStoreBillingClient.Instance.isInitialized属性。如果库已初始化，其值为true；如果Init还未被调用，则为false。

var isInitialized = RuStoreBillingClient.Instance.IsIninialized;

检查支付功能的可用性

要检查支付功能的可用性，请调用 checkPurchasesAvailability 方法。在调用时，将检查以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持支付功能。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序必须在 RuStore 控制台系统中启用购买功能。

如果所有条件都满足，则在onSuccess返回 FeatureAvailabilityResult.isAvailable == true。否则，返回 FeatureAvailabilityResult.isAvailable == false，其中 FeatureAvailabilityResult.cause—是未满足条件的错误。 所有可能的 RuStoreException 错误在 «错误处理»部分有描述。其他错误（例如，“没有互联网连接”）将在 onFailure 中返回。

```
RuStoreBillingClient.Instance.CheckPurchasesAvailability( ``onFailure: (error) => { ``// Process error ``}, ``onSuccess: (response) => { ``if (response.isAvailable) { ``// Process purchases available ``} else { ``// Process purchases unavailable ``} ``});
```

获取产品列表

要获取产品，使用GetProducts方法：

```
RuStoreBillingClient.Instance.GetProducts(productIds, ``onFailure: (error) => { ``// Process error ``}, ``onSuccess: (response) => { ``// Process response ``});
```

- string[] productIds — 产品标识符列表。

此方法返回：

- Listresponse—产品列表。

产品结构：

```
public class Product { ``public enum ProductStatus { ``ACTIVE, ``INACTIVE ``} ``public enum ProductType { ``NON_CONSUMABLE, ``CONSUMABLE, ``SUBSCRIPTION ``} ``public string productId; ``public ProductType productType; ``public ProductStatus productStatus; ``public string priceLabel; ``public int price; ``public string currency; ``public string language; ``public string title; ``public string description; ``public string imageUrl; ``public string promoImageUrl; ``public ProductSubscription subscription;}
```

- productId—产品标识符；
- productType—产品类型；
- productStatus—产品状态；
- priceLable—格式化的产品价格，包括[language]语言的货币符号；
- price—以货币最小单位（戈比）表示的价格；
- currency—ISO 4217货币代码；
- language—使用 BCP 47 编码指定的语言；
- title—产品名称，以[language]语言表示；
- description—产品描述，以[language]语言表示；
- imageUrl—图片链接；
- promoImageUrl—促销图片链接；
- subscription—订阅描述，仅对于类型为 subscription 的产品返回。

订阅结构：

public class ProductSubscription { ``public SubscriptionPeriod subscriptionPeriod; ``public SubscriptionPeriod freeTrialPeriod; ``public SubscriptionPeriod gracePeriod; ``public string introductoryPrice; ``public string introductoryPriceAmount; ``public SubscriptionPeriod introductoryPricePeriod;}

- subscriptionPeriod—订阅周期；
- freeTrialPeriod—订阅试用期；
- gracePeriod—订阅宽限期；
- introductoryPrice—格式化的介绍性订阅价格，包括product:language语言的货币符号；
- introductoryPriceAmount—以货币最小单位（戈比）表示的介绍性价格；
- introductoryPricePeriod—介绍性价格计算周期。

订阅周期结构：

public class SubscriptionPeriod { ``public int years; ``public int months; ``public int days;}

- years—年数；
- months—月数；
- days—天数。

获取购买列表

要获取购买列表，请使用getPurchases方法：

RuStoreBillingClient.Instance.GetPurchases(``onFailure: (error) => { ``// Process error ``}, ``onSuccess: (response) => { ``// Process response ``});

此方法返回：

- Listresponse—购买列表。

购买结构：

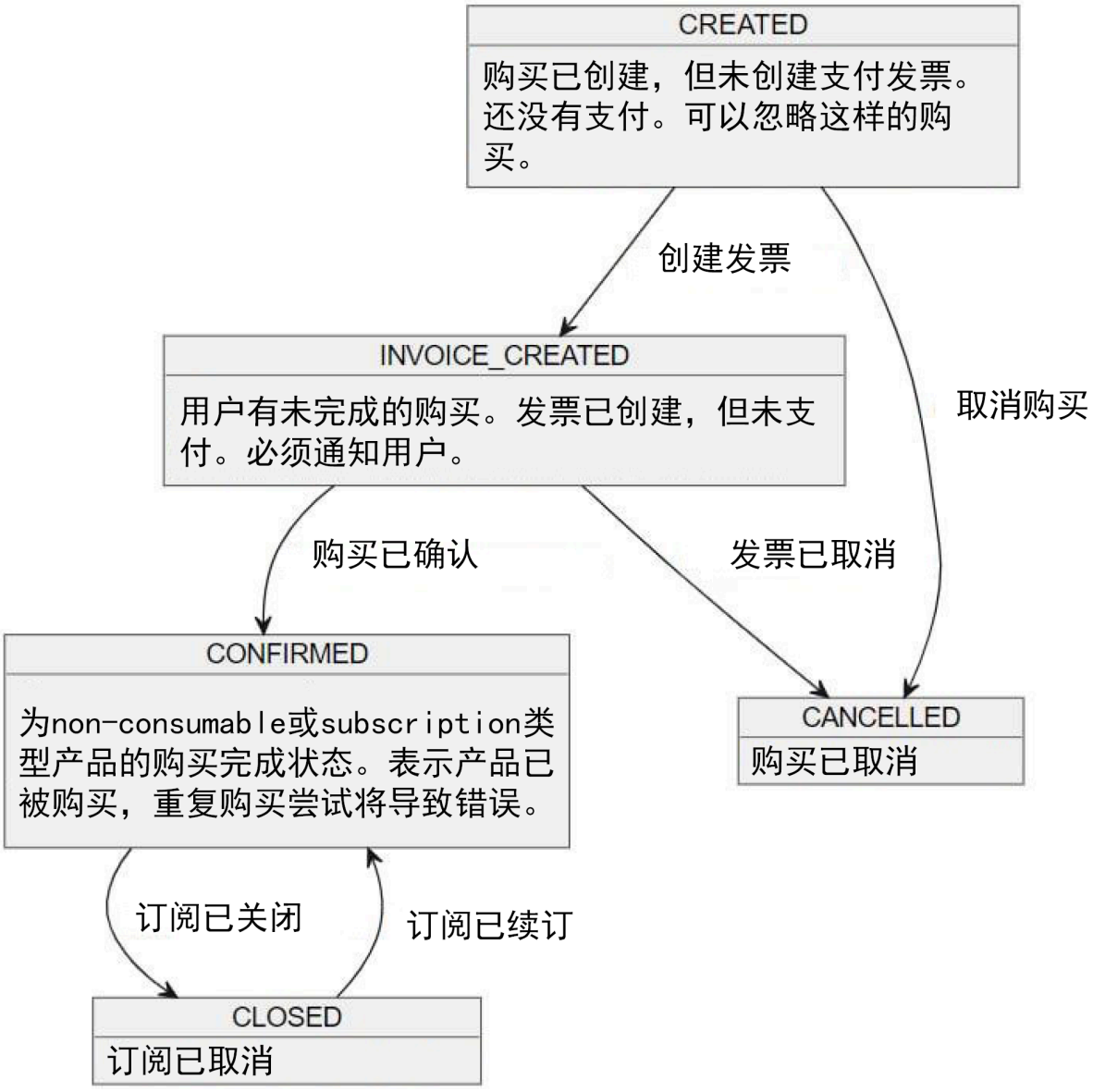
public class Purchase { ``public enum PurchaseState ``{ ``CREATED, ``INVOICE_CREATED, ``CONFIRMED, ``PAID, ``CANCELLED, ``CONSUMED, ``CLOSED ``} ``public string purchaseId; ``public string productId; ``public string description; ``public string invoiceId; ``public string language; ``public DateTime purchaseTime; ``public string orderId; ``public string amountLabel; ``public int amount; ``public string currency; ``public int quantity; ``public PurchaseState purchaseState; ``public string developerPayload; ``public string subscriptionToken;}

- purchaseId—购买标识符；
- productId—产品标识符；
- description—购买描述；
- invoiceId—账单标识符；
- language—使用 BCP 47 编码指定的语言；
- purchaseTime—购买时间；
- orderId—应用程序生成的唯一支付标识符（uuid）；
- amountLable—格式化的购买价格，包括[language]语言的货币符号；

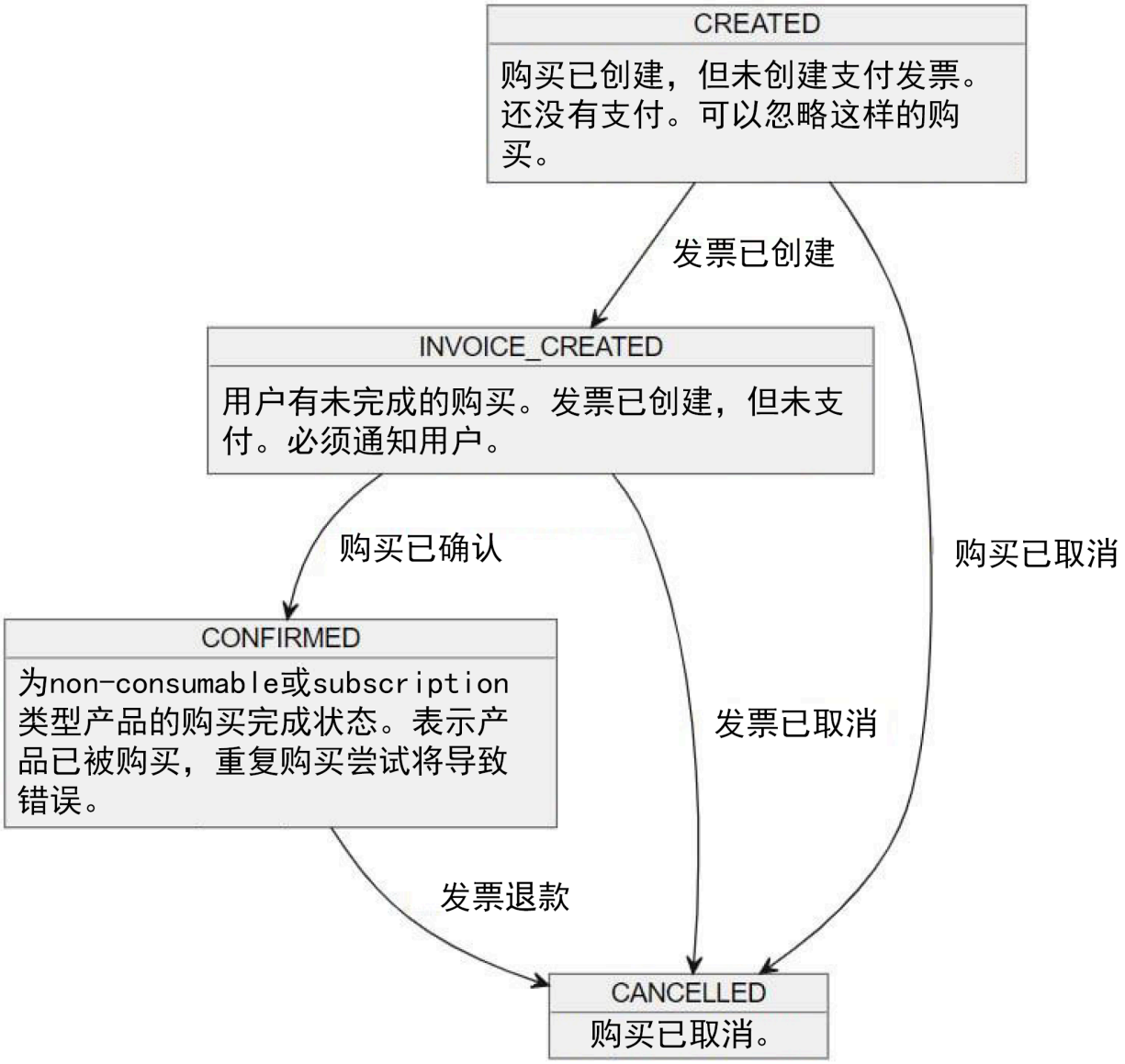
- amount—以最小货币单位表示的价格；
- currency—ISO 4217货币代码；
- quantity—产品数量；
- purchaseState—购买状态：
  - 购买状态的可能值：
    - CREATED—已创建；
    - INVOICE_CREATED—已创建，等待支付；
    - CONFIRMED—已确认；
    - PAID—已支付；
    - CANCELLED—购买已取消；
    - CONSUMED—已确认购买消耗；
    - CLOSED—订阅已取消；
    - TERMINATED—订阅已完成。
- developerPayload—开发者指定的字符串，包含关于订单的附加信息；
- subscriptionToken—用于在服务器端验证购买的令牌。

状态模型 **(purchaseState)**：

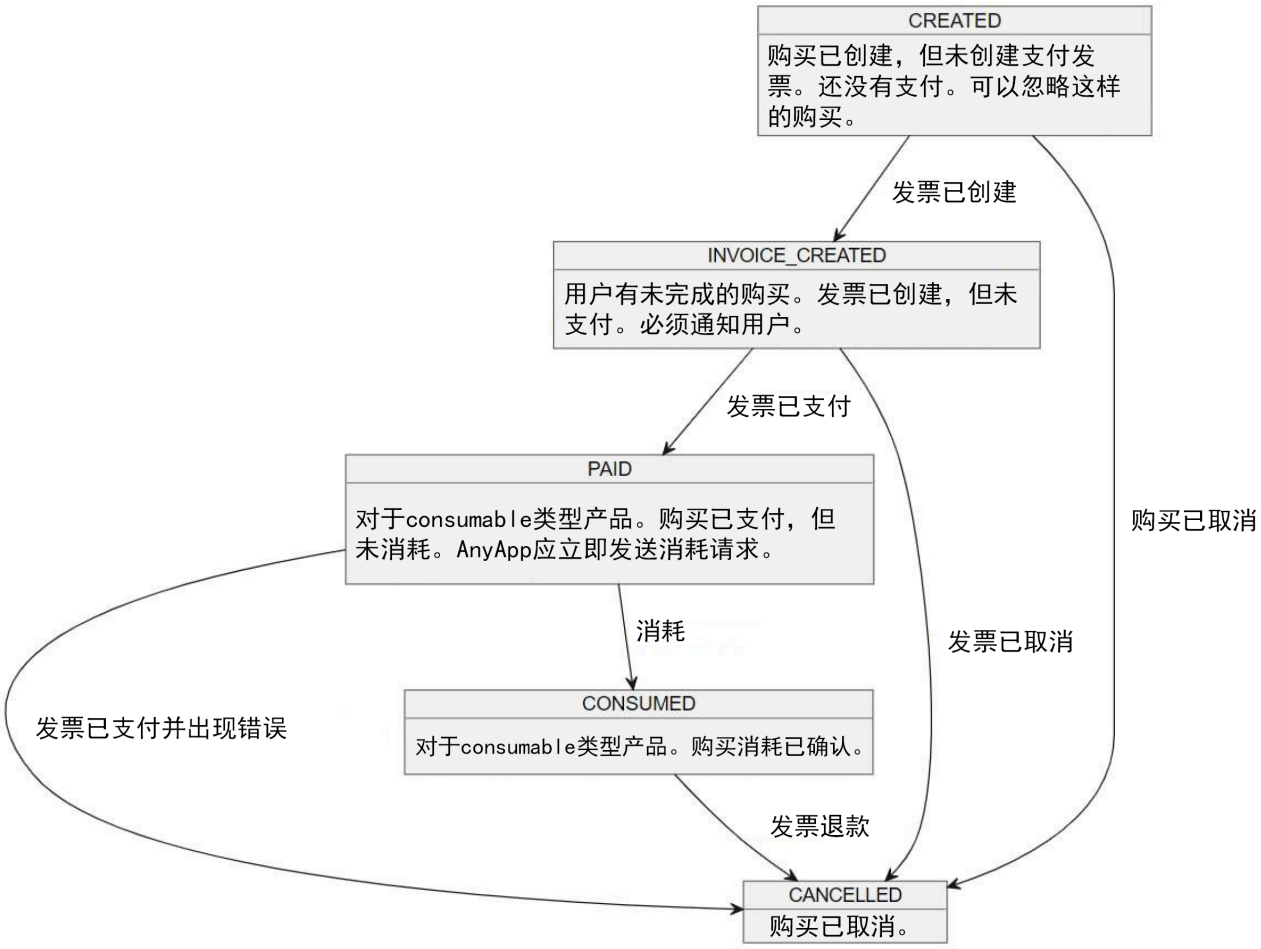
订阅购买状态模型 (SUBSCRIPTIONS)：



非消耗性产品购买状态模型 (NON-CONSUMABLES)：



消耗性产品购买状态模型 (CONSUMABLES)：



购买产品

要调用购买产品，请使用 purchaseProduct 方法：

RuStoreBillingClient.Instance.PurchaseProduct(``productId: "productId", ``quantity: 1, ``developerPayload: "", ``onFailure: (error) => { ``// Process error ``}, ``onSuccess: (response) => { ``switch (response) { ``case PaymentSuccess paymentSuccess: ``// Process PaymentSuccess ``break; ``case PaymentCancelled paymentCancelled: ``// Process PaymentCancelled ``break; ``case PaymentFailure paymentFailure: ``// Process PaymentFailure ``break; ``case InvalidPaymentState invalidPaymentState: ``// Process InvalidPaymentState ``break; ``} ``});

- stringproductId—产品标识符；
- intquantity—产品数量；
- stringdeveloperPayload—AnyApp开发者的附加信息。

购买结果的结构：

public class PaymentResult {}public class PaymentSuccess : PaymentResult { ``public string orderId; ``public string purchaseId; ``public string productId; ``public string

invoiceId; ``public string subscriptionToken;}public class PaymentCancelled : PaymentResult { ``public string purchaseId;}public class PaymentFailure : PaymentResult { ``public string purchaseId; ``public string invoiceId; ``public string orderId; ``public int quantity; ``public string productId; ``public int errorCode;}public class InvalidPaymentState : PaymentResult {}

- PaymentSuccess—成功完成数字产品购买的结果；
- PaymentCancelled—数字产品购买被取消的结果；
- PaymentFailure—购买数字产品时错误的结果；
- InvalidPaymentState—支付SDK操作错误。可能因deeplink回调不正确而发生。

购买消耗（确认）

RuStore 包含以下类型的产品：

- CONSUMABLE—消耗的（可以多次购买，例如应用中的水晶）；
- NON_CONSUMABLE—非消耗的（只能购买一次，例如禁用应用程序中的广告）；
- SUBSCRIPTION—订阅（可以购买一段时间，例如流媒体服务的订阅）。

只有处于 PurchaseState.PAID 状态的 CONSUMABLE 类型产品需要消耗。

您可以使用 confirmPurchase 方法来消耗购买：

RuStoreBillingClient.Instance.ConfirmPurchase(``purchaseId: "purchaseId", ``onFailure: (error) => { ``// Process error ``}, ``onSuccess: () => { ``// Process success ``});

- purchaseId—购买标识符。

取消购买

要取消购买，需要使用 deletePurchase 方法：

RuStoreBillingClient.Instance.DeletePurchase(``purchaseId: "purchaseId", ``onFailure: (error) => { ``// Process error ``}, ``onSuccess: () => { ``// Process success ``});

- purchaseId—购买标识符。

如果您有与删除购买相关的逻辑，请使用此方法。购买将在20分钟后自动取消，或者在同一客户端进行重复购买时取消。

购买消耗和取消的*流程*

未完成的支付处理应由 AnyApp 的开发者执行。

使用取消购买方法的条件：

1. 获取购买列表方法返回的购买状态为：
   - PurchaseState.CREATED;
   - PurchaseState.INVOICE_CREATED.
2. 如果PurchaseProduct返回InvalidPurchase。
3. 如果PurchaseProduct返回包含以下PaymentFinishCode的PurchaseResult：

- CLOSED_BY_USER—由用户取消；
- UNHANDLED_FORM_ERROR—未知错误；
- PAYMENT_TIMEOUT—支付超时错误；
- DECLINED_BY_SERVER—由服务器拒绝；
- RESULT_UNKNOWN—未知支付状态。

使用产品消耗方法的条件：

1. 获取购买列表方法返回的购买状态为：
   - PurchaseState.PAID.
2. 如果 PurchaseProduct 返回的 PaymentResult.PurchaseResult 包含以下 PaymentFinishCode：
   - SUCCESSFUL_PAYMENT—成功支付。

错误处理

在 SDK 方法的 onFailure 处理程序中传递发生的错误。

错误结构：

```
public class RuStoreError {    ``public string name;    ``public string description;}
```

- name—错误名称；
- description—错误描述。

可能的错误：

- RuStoreNotInstalledException — 用户设备上未安装 RuStore；
- RuStoreOutdatedException — 用户设备上安装的 RuStore 不支持支付；
- RuStoreUserUnauthorizedException — 用户未在 RuStore 中授权；
- RuStoreApplicationBannedException — 应用程序在 RuStore 中被禁止；
- RuStoreUserBannedException — 用户在 RuStore 中被禁止；
- RuStoreException — RuStore的基本错误，其他错误均继承自此。

调用 PurchaseProduct 方法时，错误会自动处理。

如果在 SDK 初始化时传递了参数 allowNativeErrorHandling == true，在出现错误时，除了调用相应的 onFailure 处理程序外，该错误还会传递到原生 SDK 的 resolveForBilling 方法中，以便向用户显示错误对话框：

```
public fun RuStoreException.resolveForBilling(context: Context)
```

在初始化后，可以通过设置 AllowNativeErrorHandling 属性来更改此行为：

```
RuStoreBillingClient.Instance.AllowNativeErrorHandling =  false ;
```

Unity 插件用于支付连接的更新历史：

插件版本2.2.0

- 插件内部更新。

插件版本2.1.1

- 更改了获取产品列表 GetProducts() 方法返回值的格式。
- 更改了获取购买列表 GetPurchases() 方法返回值的格式。
- 更改了获取购买信息 GetPurchaseInfo() 方法返回值的格式。
- 更改了确认购买 ConfirmPurchase() 方法返回值的格式。
- 更改了删除购买 DeletePurchase() 方法返回值的格式。
- 更改了产品购买 PurchaseProduct() 方法返回值的格式。

插件版本1.1.1

- 在 Unity 编辑器中添加了 SDK 参数配置的功能。
- 在 PurchaseProduct() 函数中，orderId 参数改为非必需。
- 在 Purchase 和 PurchaseResult 类中添加了 invoiceId 字段，作为账单标识符。
- 修正了从 Activity 中的 SDK 初始化。

插件版本0.1.9

- 添加了 RuStoreBillingClientConfig.enableLogs 标志，允许启用日志记录。
- 修复了 CallbackHandler 初始化错误。

插件版本0.1.8

- 在Purchase实体中添加了subscriptionToken字段，用于服务器端购买验证。

插件版本0.1.7

- 插件内部更新。

概述

实现示例

为了正确集成支付，建议您参阅示例应用程序：

https://gitflic.ru/project/rustore/godot-rustore-billing/file?file=example&branch=master

支付条件

为了进行支付，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. 用户必须在 RuStore 中获得授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 应用程序必须在 RuStore 控制台中启用购买功能。

在项目中集成

为了在您的项目中进行集成，您需要从 https://gitflic.ru/project/rustore/godot-rustore-billing/release页面下载归档文件。

解压缩 zip 文件，并将 rustore-billing.aar 和 RustoreBilling.gdap 文件添加到您项目的 /android/plugins 文件夹中。android/plugins

example/ ``android/ ``plugins/ ``rustore-billing-release.aar ``RustoreBilling.gdap

处理deeplink

为了通过第三方应用程序（如快速支付系统（SBP）或SberPay）正确进行支付，您需要正确实现 deeplink 的处理。为此，您需要在 android/build/AndroidManifest.xml 中指定带有您项目方案的 intent-filter：

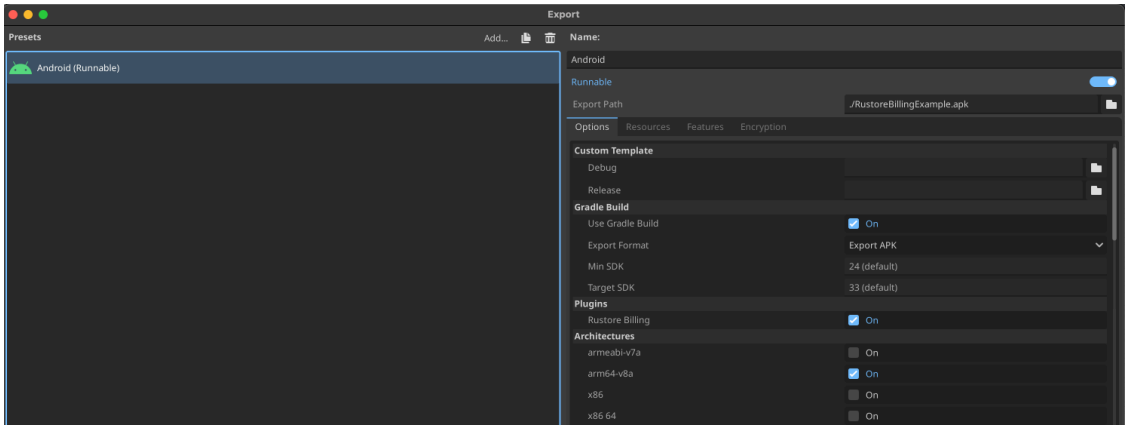
<activity ``android:name= ".sample.MainActivity" > ``<intent-filter> ``<action android:name= "android.intent.action.MAIN" /> ``<category android:name= "android.intent.category.LAUNCHER" /> ``</intent-filter> ``<intent-filter> ``<action android:name= "android.intent.action.VIEW" /> ``<category android:name= "android.intent.category.DEFAULT" /> ``<category android:name= "android.intent.category.BROWSABLE" /> ``<data android:scheme= "yourappscheme" /> ``</intent-filter> </activity>

其中“yourappscheme”—是您的 deeplink 方案，可以更改为其他方案。

这个方案应与 init() 方法中传递的方案匹配。

项目导出与连接插件

为了在您的Godot项目中使用插件，您需要在导出设置中启用它。



完成所有设置后，您可以在项目中使用插件。

初始化

在调用库的方法之前，需要先进行其初始化。要进行初始化，请调用 init() 方法：

if Engine.has_singleton("RustoreBilling"): ``billing = Engine.get_singleton( "RustoreBilling" ) ``billing.init("123456" , "yourappscheme://iamback") ``billing.rustore_is_available.connect(_on_availability) ``billing.rustore_purchase_product.connect(_on_purchase) ``billing.rustore_delete_purchase.connect(_on_delete) ``billing.rustore_confirm_purchase.connect(_on_confirm) ``billing.rustore_get_purchases.connect(_on_get_purchases) ``billing.rustore_get_purchase.connect(_on_get_purchase) ``billing.rustore_get_products.connect(_on_get_products)

- 123456—来自 RuStore 控制台的应用程序代码(例如:https://console.rustore.ru/apps/123456)。
- yourappscheme://iamback — 用于通过第三方应用程序支付后返回您的应用程序的deeplink方案 (例如SberPay 或 快速支付系统(SBP))。SDK 会为这个方案生成它自己的宿主。

重要的是,传递给 deeplinkScheme 的 deeplink 方案必须与 AndroidManifest.xml 中"处理 deeplink"部分指定的方案匹配。

插件初始化后,将连接到所有可用的信号。

检查支付功能的可用性

为了检查支付的可用性,需要满足以下条件:

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持支付功能。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序必须在 RuStore 控制台中启用购买功能。

如果所有条件都满足,isAvailable() 方法将返回 true 值。

```
func _availability():  ``if billing ! = null:  ``billing.isAvailable()func _on_availability(data: Dictionary):  ``if data[ 'status' ]  = = 'success' :  ``print ( 'success' )  ``print (data[ 'result' ])  ``elif data[ 'status' ]  = = 'failure' :  ``print ( 'failure' )  ``print (data[ 'message' ])
```

_on_availability(data: Dictionary) 方法—处理 rustore_is_available 信号的处理器,它接收关于 RuStore 可用性的消息。

通过 data['status'] 键存储的请求状态。可能的值包括:

- success—请求成功完成。在这种情况下,data['result'] 键将存储 true(如果 RuStore 可用)或 false(如果不可用)。
- failure—请求出错。错误信息存储在 data['message'] 键中。

获取产品列表

为了获取产品,您需要使用 getProducts(ids) 方法。

```
func _get_products():  ``if billing ! = null:  ``billing.getProducts([  ``"example1" ,  ``"example2"  ``])func _on_get_products(data: Dictionary):  ``if data[ 'status' ]  = = 'success' and data.has( 'items' ):  ``var items  = data[ 'items' ]  ``for key  in items:  ``print (items[key])  ``elif data[ 'status' ]  = = 'failure' :  ``print ( 'failure' )  ``print (data[ 'message' ])
```

_on_get_products(data: Dictionary) 方法—处理 rustore_get_products 信号的处理器,它接收包含可用产品列表的消息。

- ids—产品标识符列表。

通过 data['status'] 键存储请求的执行状态。可能的值包括:

- success—请求成功完成。在这种情况下，data['items'] 键将存储可用产品的列表。
- failure—请求出错。错误信息存储在 data['message'] 键中。

产品的可用字段:

- product_id—产品标识符；
- product_type—产品类型；
- product_status—产品状态；
- price_lable—格式化的产品价格，包括[language]语言的货币符号；
- price—以最小单位（戈比）表示的价格；
- currency—ISO 4217货币代码；
- language—使用 BCP 47 编码指定的语言；
- title—产品名称，以[language]语言表示；
- description—产品描述，以[language]语言表示；
- image_url—图片链接；
- promo_image_url—促销图片链接；
- subscription—订阅描述，仅对于类型为 subscription 的产品返回。

可用字段:subscription

- subscription_period—订阅周期；
- free_trial_period—订阅试用期；
- grace_period—订阅宽限期；
- introductory_price—格式化的介绍性订阅价格，包括product:language语言的货币符号；
- introductory_price_amount—以货币最小单位（戈比）表示的介绍性价格；
- introductory_price_period—介绍性价格计算周期。

可用于 subscription_period, free_trial_period, grace_period 和 introductory_price_period 键的字段：introductory_price_period

- years—年数；
- months—月数；
- days—天数。

获取购买列表

要获取购买列表，请使用getPurchases()方法：

```
func _get_purchases():  ``if billing ! = null:  ``billing.getPurchases()func _on_get_purchases(data: Dictionary):  ``if data[ 'status' ]  = = 'success' and data.has( 'items' ):  ``var items  = data[ 'items' ]  ``for key  in items:  ``print (items[key])  ``elif data[ 'status' ]  = = 'failure' :  ``print ( 'failure' )  ``print (data[ 'message' ])
```

_on_get_purchases(data: Dictionary) 方法—处理 rustore_get_purchases 信号的处理器，它接收包含可用购买列表的消息。

通过 data['status'] 键存储请求的执行状态。可能的值包括：

- success—请求成功完成。在这种情况下，data['items'] 键将存储可用购买的列表。
- failure—请求出错。错误信息存储在 data['message'] 键中。

购买的可用字段:

- purchase_id—购买标识符；
- product_id—产品标识符；
- product_type—产品类型；
- invoice_id—账单标识符；
- description—购买描述；
- language—使用 BCP 47 编码指定的语言；
- purchase_time—购买时间（RFC 3339 格式）；
- order_id—应用程序生成的唯一支付标识符（uuid）；
- amount_lable—格式化的购买价格，包括[language]语言的货币符号；
- amount—以最小货币单位表示的价格；
- currency—ISO 4217货币代码；
- quantity—产品数量；
- purchase_state—购买状态；
- developer_payload—开发者指定的字符串，包含关于订单的附加信息；
- subscription_token—用于在服务器端验证购买的令牌。有关服务器端购买验证的更多信息，请参阅购买的服务器端验证部分。

购买状态的可能值：

- CREATED—已创建；
- INVOICE_CREATED—已创建，等待支付；
- CONFIRMED—已确认；
- PAID—已支付；
- CANCELLED—购买已取消；
- CONSUMED—已确认购买消耗；
- CLOSED—订阅已取消。

有关购买状态模型的更多信息，请参阅“获取用户购买列表“部分

获取购买信息

要获取购买列表，请使用purchaseInfo()方法：

```
func _purchase_info( id : String):    ``if billing ! = null:       ``billing.purchaseInfo( id )func _on_get_purchase(data: Dictionary):    ``if data[ 'status' ]  = = 'success' :       ``print ( 'success' )       ``print (data[ 'purchase' ])    ``elif data[ 'status' ]  = = 'failure' :       ``print ( 'failure' )       ``print (data[ 'message' ])
```

_on_get_purchase (data: Dictionary) 方法—处理 rustore_get_purchase 信号的处理器，它接收包含购买的消息。

- id—购买标识符。

通过 data['status'] 键存储的请求状态。可能的值包括：

- success—请求成功完成。在这种情况下，data['items'] 键将存储可用购买的列表。
- failure—请求出错。错误信息存储在 data['message'] 键中。

购买的可用字段data[‘purchase’]

- purchase_id—购买标识符；
- product_id—产品标识符；
- product_type—产品类型；
- invoice_id—账单标识符；
- description—购买描述；
- language—使用 BCP 47 编码指定的语言；
- purchase_time—购买时间（RFC 3339 格式）；
- order_id—应用程序生成的唯一支付标识符（uuid）；
- amount_lable—格式化的购买价格，包括[language]语言的货币符号；
- amount—以最小货币单位表示的价格；
- currency—ISO 4217货币代码；
- quantity—产品数量；
- purchase_state—购买状态；
- developer_payload—开发者指定的字符串，包含关于订单的附加信息；
- subscription_token—用于在服务器端验证购买的令牌。有关服务器端购买验证的更多信息，请参阅购买的服务器端验证部分。

  购买状态的可能值：

- CREATED—已创建；
- INVOICE_CREATED—已创建，等待支付；
- CONFIRMED—已确认；
- PAID—已支付；
- CANCELLED—购买已取消；
- CONSUMED—已确认购买消耗；
- CLOSED—订阅已取消。

有关购买状态模型的更多信息，请参阅“获取用户购买列表“部分

购买产品

您可以使用 purchaseProduct(id) 方法来进行购买：

func _purchase(id : String): ``var params  = {  ``"order_id" : "1234" , ``"quantity" :  1 ,  ``"payload" : "example" ``}  ``if billing ! = null: ``billing.purchaseProduct( id , params)func _on_purchase(data: Dictionary):  ``if data['status'] = = 'cancelled' : ``print ( 'cancelled' )  ``if data['purchase'] ! = '': ``_delete(data[ 'purchase' ])  ``elif data['status'] = = 'success' : ``print ( 'success' )  ``print (data) ``elif data[ 'status' ]  = = 'failure' :  ``print ('failure') ``print (data['message'])

_on_purchase(data: Dictionary) 方法—处理 rustore_purchase_product 信号的处理器，它接收包含购买的消息。

- id—产品标识符；
- order_id—订单标识符，在AnyApp端创建（可选。如果未指定，则会自动生成）；
- quantity—产品数量（可选）；
- payload—AnyApp 开发者的附加信息（可选）。

通过 data['status'] 键存储的请求状态。可能的值包括：

- success—请求成功完成。
- failure—请求出错。错误信息存储在 data['message'] 键中。
- cancelled—用户取消了购买。

购买的可用字段:

- purchase_id—购买标识符；
- order_id—订单标识符，在AnyApp端创建（可选。如果未指定，则会自动生成）；
- invoice_id—账单标识符；
- product_id—产品标识符；
- quantity—产品数量（可选）；
- payload—AnyApp 开发者的附加信息（可选）。
- error_code—错误代码，如果请求状态为 failure。

购买确认

RuStore 包含以下类型的产品：

- CONSUMABLE—消耗的（可以多次购买，例如应用中的水晶）。
- NON_CONSUMABLE非消耗的（只能购买一次，例如禁用应用程序中的广告）。
- SUBSCRIPTION—订阅（可以购买一段时间，例如流媒体服务的订阅）。

只有处于 purchase_state == “PAID” 状态的 CONSUMABLE 类型产品需要消耗。

您可以使用 confirmPurchase(id) 方法来消耗购买：

func _confirm(id : String): ``var params  = {  ``"payload" : "123" ``}  ``if billing ! = null: ``billing.confirmPurchase( id , params)func _on_confirm(data: Dictionary):  ``print (data) ``if data[ 'status' ]  = = 'success' :  ``print ('success') ``elif data[ 'status' ]  = = 'failure' :  ``print ('failure') ``print (data['message'])

_on_confirm(data: Dictionary)方法处理 rustore_get_purchase 信号的处理器，它接收包含购买的消息。

- id—购买标识符。
- payload—开发者指定的字符串，包含订单的附加信息（可选）。

通过 data['status'] 键存储的请求状态。可能的值包括：

- success—请求成功完成。
- failure—请求出错。错误信息存储在 data['message'] 键中

取消购买

要取消购买，您可以使用 deletePurchase(id) 方法：

func _delete(id : String): ``if billing ! = null:  ``billing.deletePurchase(id)func _on_delete(data: Dictionary): ``print (data)  ``if data['status'] = = 'success' : ``print ( 'success' )  ``elif data['status'] = = 'failure' : ``print ( 'failure' )  ``print (data['message'])

_on_delete(data: Dictionary) 方法—处理 rustore_delete_purchase 信号的处理器，它接收包含删除购买的消息。

- id—购买标识符。

通过 data['status'] 键存储的请求状态。可能的值包括：

- success—请求成功完成。
- failure—请求出错。错误信息存储在 data['message'] 键中。

如果您有与删除购买相关的逻辑，请使用此方法。购买将在20分钟后自动取消，或者在同一客户端进行重复购买时取消。

React Native

概述

实现示例

为了正确集成支付，建议您参阅位于 example 文件夹中的示例应用程序：

https://gitflic.ru/project/rustore/react-native-rustore-billing-sdk/file?file=example

支付条件

为了进行支付，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. 用户必须在 RuStore 中获得授权。
3. 用户和应用程序不应在RuStore中被封锁。

4. 应用程序必须在 RuStore 控制台中启用购买功能。

在项目中集成

要将包连接到项目，您需要执行以下命令：

// HTTPSnpm install git+https: //git @gitflic.ru:rustore /react-native-rustore-billing-sdk .git// SSHnpm install git+ ssh : //git @gitflic.ru:rustore /react-native-rustore-billing-sdk .git

处理deeplink

为了通过第三方应用程序（如快速支付系统（SBP）或SberPay）正确进行支付，您需要正确实现 deeplink 的处理。为此，您需要在 AndroidManifest.xml 中指定带有您项目方案的 intent-filter ：

< activity ``android:name = ".sample.MainActivity" > ``< intent-filter > ``< action android:name = "android.intent.action.MAIN" /> ``< category android:name = "android.intent.category.LAUNCHER" /> ``</ intent-filter > ``< intent-filter > ``< action android:name = "android.intent.action.VIEW" /> ``< category android:name = "android.intent.category.DEFAULT" /> ``< category android:name = "android.intent.category.BROWSABLE" /> ``< data android:scheme = "yourappscheme" /> ``</ intent-filter > </ activity >

其中“yourappscheme”—是您的 deeplink 方案，可以更改为其他方案。

这个方案应与 RustoreBillingClient. initialize() 方法中传递的方案匹配。

初始化

在调用库的方法之前，需要先进行其初始化。为了初始化，请调用 RustoreBillingClient.initialize() 方法：

try { ``RustoreBillingClient.initialize({ ``consoleApplicationId: 'appId' , ``deeplinkScheme: 'scheme' , ``}); ```console.log(initialize success: ${result});``
} catch(err) {
 `console.log(`initialize err: ${err}`);}

- consoleApplicationId—来自RuStore开发者控制台的应用程序代码（例如：https://console.rustore.ru/apps/123456）。
- deeplinkScheme—用于通过第三方应用程序支付后返回您的应用程序的deeplink方案（例如通过 SberPay 或快速支付系统(SBP)）。SDK 会为这个方案生成它自己的宿主。

重要的是，传递给 deeplinkScheme 的 deeplink 方案必须与 AndroidManifest.xml 中上面的“处理 deeplink”部分指定的方案匹配。

检查支付功能的可用性

为了检查支付的可用性，需要满足以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持支付功能。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序必须在 RuStore 控制台中启用购买功能。

如果所有条件都满足，RustoreBillingClient.checkPurchasesAvailability() 方法将返回 true 值。

try { ``const isAvailable = await RustoreBillingClient.checkPurchasesAvailability(); ```console.log(available success ${isAvailable});``
} catch(err) {
 `console.log(`available error ${err}`);}

获取产品列表

为了获取产品，您需要使用 RustoreBillingClient.getProducts(productIds) 方法。

try { ``const products = await RustoreBillingClient.getProducts(productIds); ``for (const product of products) { ``console.log(product?.productId); ``}} catch (err) { ```console.log(products err: ${err});``
}`

- productIds—产品标识符列表。

该方法返回一个产品列表 Product[]。以下是产品模型的描述：

interface Product { ``productId: string; ``productType?: ProductType; ``productStatus: ProductStatus; ``priceLabel?: string; ``price?: number; ``currency?: string; ``language?: string; ``title?: string; ``description?: string; ``imageUrl?: string; ``promoImageUrl?: string; ``subscription?: ProductSubscription;}

- productId—产品标识符。
- productType—产品类型。
- productStatus—产品状态。
- priceLabel—格式化的产品价格，包括[language]语言的货币符号。
- price—以最小货币单位表示的价格。
- currency—ISO 4217 货币代码。
- language—使用 BCP 47 编码指定的语言。
- title—产品名称，以[language]语言表示。
- description—产品描述，以[language]语言表示。
- imageUrl—图片链接。
- promoImageUrl—促销图片链接。
- subscription—订阅描述，仅对于类型为 subscription 的产品返回。

订阅结构Subscription：

interface ProductSubscription { ``subscriptionPeriod?: SubscriptionPeriod; ``freeTrialPeriod?: SubscriptionPeriod; ``gracePeriod?: SubscriptionPeriod; ``introductoryPrice?: string; ``introductoryPriceAmount?: string; ``introductoryPricePeriod?: SubscriptionPeriod;}

- subscriptionPeriod—订阅周期。
- freeTrialPeriod—订阅试用期。
- gracePeriod—订阅宽限期。
- introductoryPrice—格式化的介绍性订阅价格，包括product:language语言的货币符号。
- introductoryPriceAmount—以货币最小单位（戈比）表示的介绍性价格。
- introductoryPricePeriod—介绍性价格计算周期。

订阅周期结构SubscriptionPeriod：

interface SubscriptionPeriod { ``years: number; ``months: number; ``days: number;}

- years—年数。
- months—月数。
- days—天数。

获取购买列表

要获取购买列表，请使用RustoreBillingClient.getPurchases()方法：

try { ``const purchases = await RustoreBillingClient.getPurchases(); ``for (const purchase of purchases) { ``console.log(purchase?.purchaseId); ``}} catch (err) { ```console.log(purchase err: ${err});``
}`

该方法返回一个购买列表 `Purchase[]`。以下是购买模型的描述：

interface Purchase { ``purchaseId?: string; ``productId: string; ``productType?: ProductType; ``invoiceId?: string; ``description?: string; ``language?: string; ``purchaseTime?: string; ``orderId?: string; ``amountLabel?: string; ``amount?: number; ``currency?: string; ``quantity?: number; ``purchaseState?: PurchaseState; ``developerPayload?: string; ``subscriptionToken?: string;}

- purchaseId—购买标识符。
- productId—产品标识符。
- productType—产品类型。
- invoiceId—账单标识符。
- description—购买描述。
- language—使用 BCP 47 编码指定的语言。
- purchaseTime—购买时间（RFC 3339 格式）。
- orderId—应用程序生成的唯一支付标识符（uuid）。
- amountLable—格式化的购买价格，包括[language]语言的货币符号。
- amount—以最小货币单位表示的价格。
- currency—ISO 4217 货币代码。
- quantity—产品数量。
- purchaseState购买状态。
  - 购买状态的可能值：
    - CREATED—已创建。
    - INVOICE_CREATED—已创建，等待支付。
    - CONFIRMED—已确认。
    - PAID—已支付。
    - CANCELLED—购买已取消。
    - CONSUMED—已确认购买消耗。
    - CLOSED—订阅已取消。
    - TERMINATED—订阅已完成。
- developerPayload—开发者指定的字符串，包含关于订单的附加信息。
- subscriptionToken—用于在服务器端验证购买的令牌。

获取特定购买

要获取特定购买，需使用 RustoreBillingClient.getPurchaseInfo(purchaseId) 方法：

try { ``const purchase = await RustoreBillingClient.getPurchaseInfo( 'purchaseId' ); ``console.log(purchase?.purchaseId);} catch (err) { ```console.log(purchase err: ${err});``
}`

- purchaseId—购买标识符。

该方法返回 Purchase，如上所描述。

购买产品

要调用购买产品，请使用 `RustoreBillingClient.purchaseProduct({…})` 方法：

try { ``const response = await RustoreBillingClient.purchaseProduct({ ``productId: 'productId' , ``orderId: 'orderId' , ``quantity: 0, ``developerPayload: 'developerPayload' ``}); ```console.log(purchase success: ${response});``
} catch(err) {
 `console.log(`purchase err: ${err}`);}

- productId—产品标识符。
- orderId—订单标识符，在AnyApp端创建（可选。如果未指定，则会自动生成）。
- quantity—产品数量（可选）。
- developerPayload—AnyApp 开发者的附加信息（可选）。

购买的结果可能是以下几种接口之一：SuccessPayment、CancelledPayment 或 FailurePayment：

enum PaymentResult { ``SUCCESS = 'SUCCESS' , ``CANCELLED = 'CANCELLED' , ``FAILURE = 'FAILURE' ,}interface SuccessPaymentResult { ``orderId?: string; ``purchaseId: string; ``productId: string; ``invoiceId: string; ``subscriptionToken?: string;}interface SuccessPayment { ``type: PaymentResult.SUCCESS; ``result: SuccessPaymentResult;}interface CancelledPaymentResult { ``purchaseId: string;}interface CancelledPayment { ``type: PaymentResult.CANCELLED; ``result: CancelledPaymentResult;}interface FailurePaymentResult { ``purchaseId?: string; ``invoiceId?: string; ``orderId?: string; ``quantity?: number; ``productId?: string; ``errorCode?: number;}interface FailurePayment { ``type: PaymentResult.FAILURE; ``result: FailurePaymentResult;}

- SuccessPayment—成功完成数字产品购买的结果。
- FailurePayment—购买数字产品时错误的结果。
- CancelledPayment—数字产品购买被取消的结果。

购买确认

RuStore 包含以下类型的产品：

- CONSUMABLE—消耗的（可以多次购买，例如应用中的水晶）。
- NON_CONSUMABLE—非消耗的（只能购买一次，例如禁用应用程序中的广告）。
- SUBSCRIPTION—订阅（可以购买一段时间，例如流媒体服务的订阅）。

只有处于 PurchaseState.PAID 状态的 CONSUMABLE 类型产品需要消耗。

您可以使用 RustoreBillingClient.confirmPurchase({…}) 方法来消耗购买：

try { ``const isConfirmed = await RustoreBillingClient.confirmPurchase({ ``purchaseId: 'purchaseId' , ``developerPayload: 'developerPayload' ``}) ```console.log(confirm success: ${isConfirmed});``
} catch(err) {
 `console.log(`confirm err: ${err}`);}

- purchaseId—购买标识符。
- developerPayload—开发者指定的字符串，包含附加信息。

如果所有条件都满足，RustoreBillingClient.confirmPurchase() 方法将返回 true 值。

取消购买

要取消购买，您可以使用 RustoreBillingClient.deletePurchase(purchaseId) 方法：

try { ``const isDeleted = await RustoreBillingClient.deletePurchase(purchaseId) ```console.log(`delete`success: ${isDeleted});``
} catch(err) {
 `console.log(` delete err: ${err}`);}

- purchaseId—购买标识符。

如果所有条件都满足，RustoreBillingClient.deletePurchase() 方法将返回 true 值。

如果您有与删除购买相关的逻辑，请使用此方法。购买将在20分钟后自动取消，或者在同一客户端进行重复购买时取消。

购买消耗和取消的*流程*

未完成的支付处理应由 AnyApp 的开发者执行。

如果出现以下情况，应使用取消购买方法（deletePurchase）：

1. 获取购买列表方法（getPurchases）返回了具有以下状态的购买：
    1. PurchaseState.CREATED.
    2. PurchaseState.INVOICE_CREATED.
2. 购买方法 (purchaseProduct) 返回 PaymentResult.Cancelled。
3. 购买方法 (purchaseProduct) 返回 PaymentResult.Failure。

如果获取购买列表的方法 (getPurchases) 返回了类型为 CONSUMABLE 且状态为 PurchaseState.PAID 的购买，则需要使用产品消耗方法 (confirmPurchase)。

Flutter

概述

实现示例

为了正确集成支付，建议您参阅示例应用程序：
https://gitflic.ru/project/rustore/flutter-rustore-billing

支付条件

为了进行支付，必须遵守所有条件：

1. 用户的设备上安装了RuStore应用程序。
2. 用户在RuStore应用程序中已授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 在RuStore 控制台系统中为应用程序启用了购买功能。

该服务在俄罗斯联邦以外有一些运作限制。

在项目中集成

要将支付包连接到项目，您需要执行以下命令：

flutter pub add flutter_rustore_billing

这个命令将在 pubspec.yaml 文件中添加一个字符串。

dependencies: ``flutter_rustore_billing: ^3 .0 .0

处理deeplink

为了通过第三方应用程序（如快速支付系统（SBP）或SberPay）正确进行支付，您需要正确实现 deeplink 的处理。为此，您需要在 AndroidManifest.xml 中指定带有您项目scheme的 intent-filter：

<activity ``android:name= ".sample.MainActivity" > ``<intent-filter> ``<action android:name= "android.intent.action.MAIN" /> ``<category android:name= "android.intent.category.LAUNCHER" /> ``</intent-filter> ``<intent-filter> ``<action android:name= "android.intent.action.VIEW" /> ``<category android:name= "android.intent.category.DEFAULT" /> ``<category android:name= "android.intent.category.BROWSABLE" /> ``<data android:scheme= "yourappscheme" /> ``</intent-filter> </activity>

其中“yourappscheme”—您的deeplink方案，可以更改为其他方案。

这个方案应与 initialize() 方法中传递的方案匹配。

初始化

在调用库的方法之前，需要先进行其初始化。为了初始化，请调用 RustoreBillingClient.initialize() 方法：

RustoreBillingClient.initialize(``"123456" , ``"yourappscheme://iamback" , ``).then((value) { ``print("initialize success: $value"); ``}, onError: (err) { ``print("initialize err: $err");});

- 123456—来自RuStore开发者控制台的应用程序代码（例如：https://console.rustore.ru/apps/123456）。
- yourappscheme://iamback—用于通过第三方应用程序支付后返回您的应用程序的 deeplink方案 (例如通过 SberPay 或快速支付系统(SBP))。SDK 会为这个方案生成它自己的宿主。

重要的是，传递给 deeplinkScheme 的 deeplink 方案必须与 AndroidManifest.xml 中“处理 deeplink”部分指定的方案匹配。

检查支付功能的可用性

为了检查支付的可用性，需要满足以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持支付功能。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序必须在 RuStore 控制台系统中启用购买功能。

如果所有条件都满足，RustoreBillingClient.available() 方法将返回 true 值。

```
RustoreBillingClient.available().then((value) {      ``print( "available success $value" );}, onError: (err) {      ``print( "available err: $err" );});
```

获取产品列表

为了获取产品，您需要使用 RustoreBillingClient.products(ids) 方法。

```
RustoreBillingClient.products(ids).then((response) {     ``for ( final product in response.products) {      ``print(product?.productId);     ``}}, onError: (err) {     ``print( "products err: $err" );});
```

- ids: List<String?> —产品标识符列表。

此方法返回ProductsResponse：

```
class ProductsResponse { ``int code; ``String? errorMessage; ``String? errorDescription; ``String? traceId; ``List<Product?> products; ``List<DigitalShopGeneralError?> errors;}
```

- code—响应代码；
- errorMessage—错误消息；
- errorDescription—错误描述；
- traceId—错误标识符；
- errors—错误列表；
- products—产品列表。

DigitalShopGeneralError 错误的结构：

```
class DigitalShopGeneralError { ``String? name; ``int ? code; ``String? description}
```

- name—错误名称；
- code—错误代码；
- description—错误描述。

产品结构Product：

```
class Product { ``String productId; ``String? productType; ``String productStatus; ``String? priceLabel; ``int ? price; ``String? currency; ``String? language; ``String? title; ``String? description; ``String? imageUrl; ``String? promoImageUrl; ``Subscription? subscription;}
```

- productId—产品标识符；
- productType—产品类型；
- productStatus—产品状态；
- priceLable—格式化的产品价格，包括[language]语言的货币符号；
- price—以最小单位（戈比）表示的价格；
- currency—ISO 4217货币代码；
- language—使用 BCP 47 编码指定的语言；
- title—产品名称，以[language]语言表示；
- description—产品描述，以[language]语言表示；
- imageUrl—图片链接；
- promoImageUrl—促销图片链接；
- subscription—订阅描述，仅对于类型为 subscription 的产品返回。

订阅结构Subscription：

```
class Subscription { ``SubscriptionPeriod? subscriptionPeriod; ``SubscriptionPeriod? freeTrialPeriod; ``SubscriptionPeriod? gracePeriod; ``String? introductoryPrice; ``String? introductoryPriceAmount; ``SubscriptionPeriod? introductoryPricePeriod;}
```

- subscriptionPeriod—订阅周期；
- freeTrialPeriod—订阅试用期；
- gracePeriod—订阅宽限期；
- introductoryPrice—格式化的介绍性订阅价格，包括product:language语言的货币符号；
- introductoryPriceAmount—以货币最小单位（戈比）表示的介绍性价格；
- introductoryPricePeriod—介绍性价格计算周期。

订阅周期结构SubscriptionPeriod：

```
class SubscriptionPeriod { ``int years; ``int months; ``int days;}
```

- years—年数；
- months—月数；
- days—天数。

获取购买列表

要获取购买列表，请使用RustoreBillingClient.purchases()方法：

```
RustoreBillingClient.purchases().then((response) {     ``for ( final product in response.purchases) {     ``print(product?.purchaseId);     ``}}, onError: (err) {     ``print( "purchases err: $err" );});
```

此方法返回PurchasesResponse：

```
class PurchasesResponse { ``int code; ``String? errorMessage; ``String? errorDescription; ``String? traceId; ``List<Purchase?> purchases; ``List<DigitalShopGeneralError?> errors;}
```

- code—响应代码；
- errorMessage—错误消息；
- errorDescription—错误描述；
- traceId—错误标识符；

- errors—错误列表；
- purchases—购买列表。

DigitalShopGeneralError 错误的结构：

```
class DigitalShopGeneralError { ``String? name; ``int ? code; ``String? description;}
```

- name—错误名称；
- code—错误代码；
- description—错误描述。

购买结构Purchase：

```
class Purchase { ``String? purchaseId; ``String? productId; ``String? description; ``String? language; ``String? purchaseTime; ``String? orderId; ``String? amountLabel; ``int ? amount; ``String? currency; ``int ? quantity; ``String? purchaseState; ``String? developerPayload; ``String? invoiceId; ``String? subscriptionToken;}
```

- purchaseId—购买标识符；
- productId—产品标识符；
- description—购买描述；
- language—使用 BCP 47 编码指定的语言；
- purchaseTime—购买时间；
- orderId—应用程序生成的唯一支付标识符（uuid）；
- amountLable—格式化的购买价格，包括[language]语言的货币符号；
- amount—以货币最小单位（戈比）表示的价格；
- currency—ISO 4217货币代码；
- quantity—产品数量；
- purchaseState—购买状态：
  - 购买状态的可能值：
    - CREATED—已创建；
    - INVOICE_CREATED—已创建，等待支付；
    - CONFIRMED—已确认；
    - PAID—已支付；
    - CANCELLED—购买已取消；
    - CONSUMED—已确认购买消耗；
    - CLOSED—订阅已取消。
- developerPayload—开发者指定的字符串，包含关于订单的附加信息；
- invoiceId — 账单标识符;
- subscriptionToken—用于在服务器端验证购买的令牌。

购买产品

要调用购买产品，请使用 RustoreBillingClient.purchase(id) 方法：

```
RustoreBillingClient.purchase(id).then((response) {    ``print( "purchase success: $response" );   ``}, onError: (err) {    ``print( "purchase err: $err" );});
```

- id—产品标识符。

购买结果的结构PaymentResult：

```
class PaymentResult { ``SuccessInvoice? successInvoice; ``InvalidInvoice?
invalidInvoice; ``SuccessPurchase? successPurchase; ``InvalidPurchase? invalidPurchase;}
```

SuccessInvoice结构：

```
class SuccessInvoice { ``String invoiceId; ``String finishCode;}
```

InvalidInvoice 结构：

```
class InvalidInvoice { ``String? invoiceId;}
```

SuccessPurchase 结构：

```
class SuccessPurchase { ``String finishCode; ``String? orderId; ``String purchaseId; ``String productId;}
```

InvalidPurchase 结构：

```
class InvalidPurchase { ``String? purchaseId; ``String? invoiceId; ``String? orderId; ``int?
quantity; ``String? productId; ``int? errorCode;}
```

- SuccessInvoice—支付以结果结束；
- InvalidInvoice—支付结束时没有指定发票。可能是因为启动了带有错误发票的支付（例如，空字符串）；
- SuccessPurchase—成功完成数字产品购买的结果；
- InvalidPurchase—在支付数字产品时支付结束后出现错误。

可能的状态，finishCode可能包括：

- SUCCESSFUL_PAYMENT—成功支付；
- CLOSED_BY_USER—由用户取消；
- UNHANDLED_FORM_ERROR—未知错误；
- PAYMENT_TIMEOUT—支付超时错误；
- DECLINED_BY_SERVER—由服务器拒绝；
- RESULT_UNKNOWN—未知支付状态。

购买消耗（确认）

RuStore 应用程序包含以下类型的产品：

- CONSUMABLE—消耗的（可以多次购买，例如应用中的水晶）；
- NON_CONSUMABLE—非消耗的（只能购买一次，例如禁用应用程序中的广告）；
- SUBSCRIPTION—订阅（可以购买一段时间，例如流媒体服务的订阅）。

只有处于 PurchaseState.PAID 状态的 CONSUMABLE 类型产品需要消耗。

您可以使用 RustoreBillingClient.confirm(id) 方法来消耗购买：

```
RustoreBillingClient.confirm(id).then((response) {    ``print( "confirm success: $response" );   ``},
onError: (err) {    ``print( "confirm err: $err" );});
```

- id—购买标识符。

此方法返回ConfirmPurchaseResponse：

class ConfirmPurchaseResponse { ``int code; ``String? errorMessage; ``String? errorDescription; ``String? traceId; ``List<DigitalShopGeneralError?> errors;}

- code—响应代码；
- errorMessage—用户的错误消息；
- errorDescription—错误消息的详细解释；
- traceId—错误消息标识符；
- errors—错误列表。

DigitalShopGeneralError 错误的结构：

class DigitalShopGeneralError { ``String? name; ``int ? code; ``String? description;}

- name—错误属性名称；
- code—错误代码；
- description—错误描述。

Unreal Engine

概述

为了进行支付，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. 用户必须在 RuStore 中获得授权。
3. 用户和应用程序不应在RuStore中被封锁。
4. 应用程序必须在 RuStore 控制台系统中启用购买功能。
5. Unreal Engine 4.26或更高版本。

在项目中集成

1. 从RuStore的官方gitflic仓库复制*'Plugins'*文件夹的内容到您项目中的*'Plugins'*文件夹。重启Unreal Engine，在插件列表中（Edit → Plugins → Project → Mobile），选择'RuStoreBilling'和'RuStoreCore'插件。
2. 在*'YourProject.Build.cs'*文件中，将'RuStoreCore'和'RuStoreBilling'模块添加到PublicDependencyModuleNames列表中。
3. 在项目设置中（Edit → Project Settings → Android），设置Minimum SDK Version为不低于24，Target SDK Version为不低于31。

处理deeplink

当用户通过SberPay支付时，会跳转到俄罗斯储蓄银行应用程序。支付完成后，应使用deeplink返回原应用程序。为了处理deeplink，RuStore Billing插件会自动在AndroidManifest.xml中为主要activity添加android:exported='true'属性和以下intent-filter：

AndroidManifest.xml

< intent-filter > ``< action android:name = "android.intent.action.VIEW" /> ``< category android:name = "android.intent.category.DEFAULT" /> ``< category android:name =

"android.intent.category.BROWSABLE" /> ``< data android:scheme = "yourscheme" /></ intent-filter >

其中yourscheme—你的deeplink方案，可以更改为其他方案，并且应该与在初始化billing客户端库时指定的deeplinkSheme值相匹配。

您可以在'RuStoreCore_UPL_Android.xml'文件中更改'yourscheme'到您的方案。

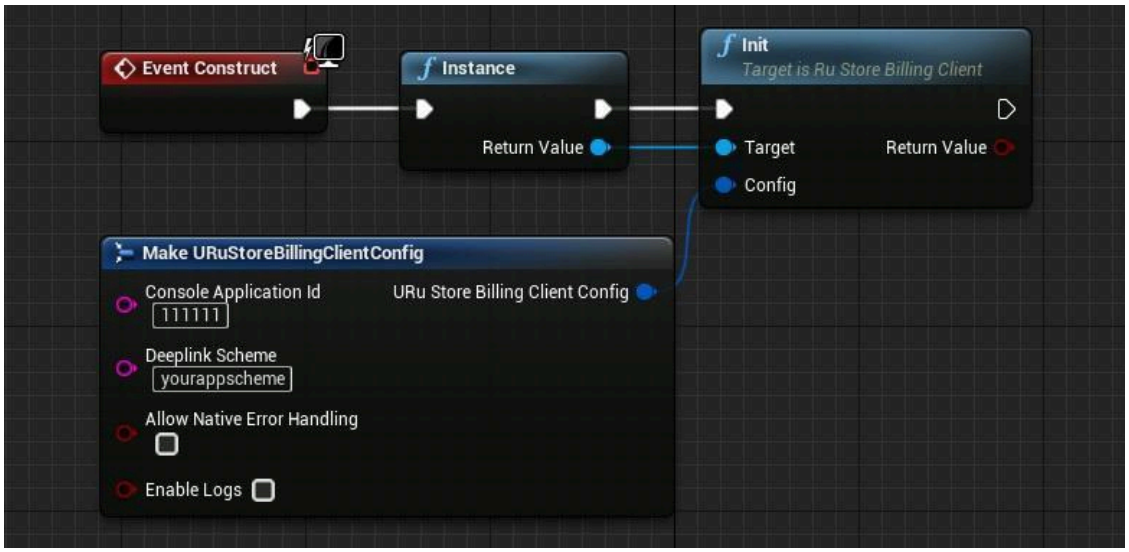
初始化

在调用库的方法之前，需要先进行其初始化。初始化将使用的参数由FURuStoreBillingClientConfig结构进行设置。

初始化

```
FURuStoreBillingClientConfig config;config.consoleApplicationId = "111111" ;config.deeplinkScheme = "yourscheme" ;URuStoreBillingClient::Instance()->Init(config);
```

所有与客户端的操作也可以通过Blueprints进行。初始化的例子：



- consoleApplicationId—来自RuStore开发者控制台的应用程序代码（例如：https://console.rustore.ru/apps/111111）"
- deeplinkSheme—url用于实现deeplink的url。可以使用任何唯一的名称作为名称（例如：yourappscheme）。
- allowAllowNativeErrorHandling — 允许在原生SDK中处理错误（详见错误处理部分）
- enableLogs—启用事件日志记录

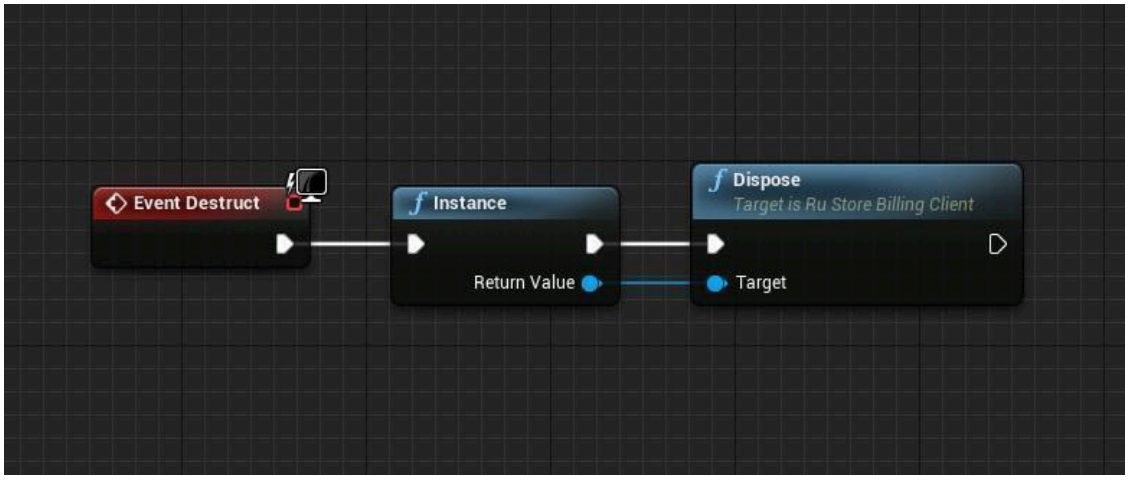
重要提示：

1. 传递给deeplinkPrefix的deeplink方案，需要与AndroidManifest.xml中指定的方案相匹配（详见“处理deeplink”部分）。
2. 调用Init()将对象绑定到场景的根部，如果之后不再计划使用该对象，为了释放内存，需要执行Dispose()方法。

调用Dispose()方法会将对象从根部解绑，并安全地完成所有已发送的请求。

反初始化

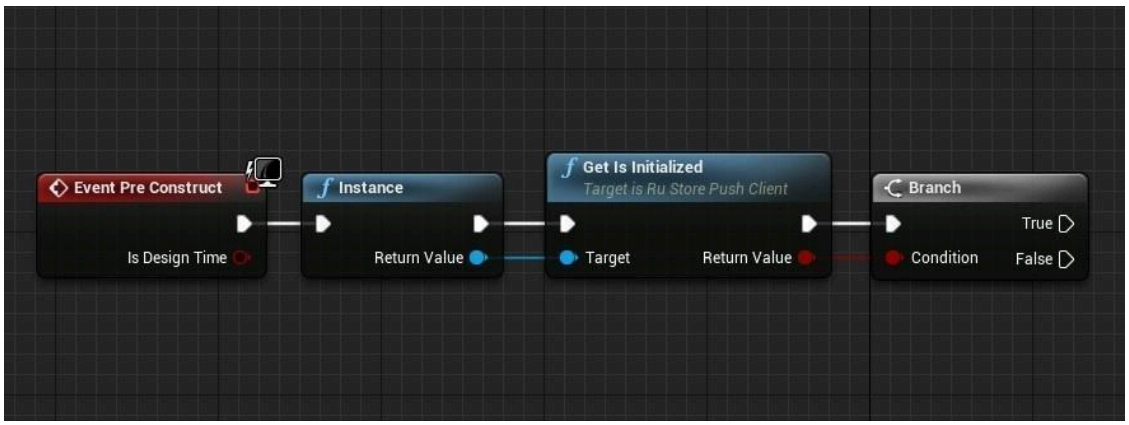
URuStoreBillingClient::Instance()->Dispose();



如果您需要检查库的初始化情况，请使用 URuStoreBillingClient::Instance()->getIsInitialized() 属性。如果库已初始化，其值为 true；如果尚未调用 Init，则为 false。

检查初始化情况

bool isInitialized = URuStoreBillingClient::Instance()->IsIninialized();



检查支付功能的可用性

要检查支付功能的可用性，请调用 checkPurchasesAvailability 方法。在调用时，将检查以下条件：

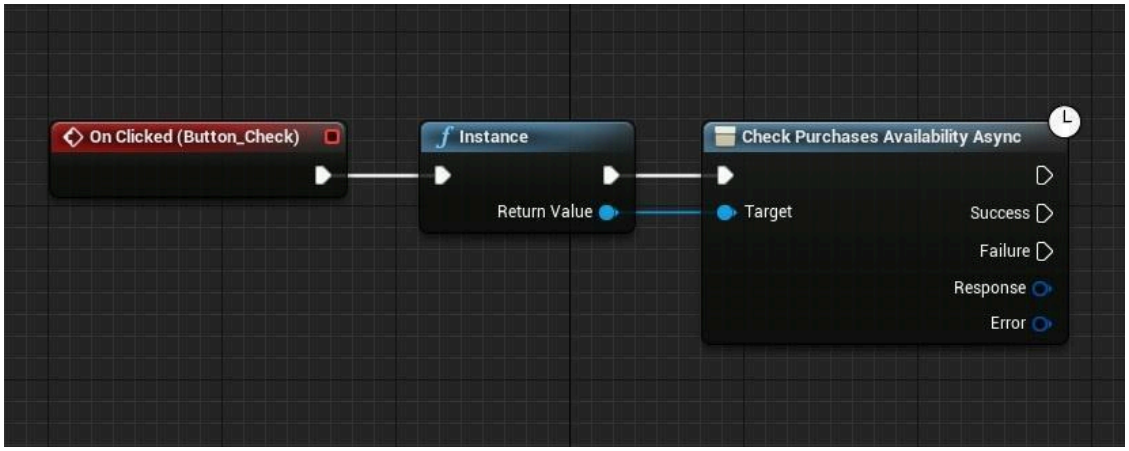
1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持支付功能。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序必须在 RuStore 控制台系统中启用购买功能。

每次调用 CheckPurchasesAvailability() 都会返回一个在应用程序单次启动期间唯一的 requestId。每个事件都会返回触发该事件的请求的 requestId。

调用方法**CheckPurchasesAvailability**

long requestId = URuStoreBillingClient::Instance()->CheckPurchasesAvailability(``[]( long requestId, TSharedPtr<FURuStoreFeatureAvailabilityResult, ESPMode::ThreadSafe> response) { ``// Process response ``}, ``[](long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

Blueprint实现:



Success回调返回FURuStoreFeatureAvailabilityResult结构体, 在Response参数中:

响应**CheckPurchasesAvailability**

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreFeatureAvailabilityResult{ ``GENERATED_USTRUCT_BODY() ``FURuStoreFeatureAvailabilityResult() ``{ ``isAvailable = false ; ``} ``UPROPERTY(BlueprintReadWrite) ``bool isAvailable; ``UPROPERTY(BlueprintReadWrite) ``FURuStoreError cause;};

isAvailable表示是否满足执行支付的条件。

cause错误信息。

Failure回调返回FURuStoreError结构体, 包含错误信息, 在Error参数中。所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name– 错误名称。
- description– 错误描述。

获取产品列表

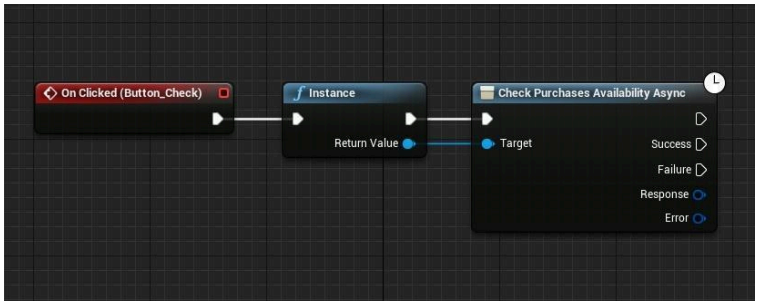
要获取产品, 使用GetProducts()方法:

调用方法**GetProducts**

long requestId = URuStoreBillingClient::Instance()->GetProducts(``productIds, ``[](long requestId, TSharedPtr<FURuStoreProductsResponse, ESPMode::ThreadSafe> response) { ``// Process response ``}, ``[]( long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

- TArrayproductIds– 产品标识符列表。

Blueprint实现:



Success回调返回FURuStoreProductsResponse结构体, 在Response参数中:

响应**GetProducts**

USTRUCT(BlueprintType)struct FURuStoreProductsResponse{ ``GENERATED_USTRUCT_BODY() ``UPROPERTY(BlueprintReadOnly) ``TArray<FURuStoreProduct> products;};

- products– 产品列表。

产品结构

USTRUCT(BlueprintType)struct FURuStoreProduct{ ``GENERATED_USTRUCT_BODY() ``FURuStoreProduct() ``{ ``productId = "" ; ``productType = EURuStoreProductType::NON_CONSUMABLE; ``productStatus = EURuStoreProductStatus::INACTIVE; ``priceLabel = "" ; ``price = 0; ``currency = "" ; ``language = "" ; ``title = "" ; ``description = "" ; ``imageUrl = "" ; ``promoImageUrl = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString productId; ``UPROPERTY(BlueprintReadOnly) ``EURuStoreProductType productType; ``UPROPERTY(BlueprintReadOnly) ``EURuStoreProductStatus productStatus; ``UPROPERTY(BlueprintReadOnly) ``FString priceLabel; ``UPROPERTY(BlueprintReadOnly) ``int price; ``UPROPERTY(BlueprintReadOnly) ``FString currency; ``UPROPERTY(BlueprintReadOnly) ``FString language; ``UPROPERTY(BlueprintReadOnly) ``FString title; ``UPROPERTY(BlueprintReadOnly) ``FString description; ``UPROPERTY(BlueprintReadOnly) ``FString imageUrl; ``UPROPERTY(BlueprintReadOnly) ``FString promoImageUrl; ``UPROPERTY(BlueprintReadOnly) ``FURuStoreProductSubscription subscription;};

- productId- 产品标识符。
- productType- 产品类型。
- productStatus- 产品状态。
- priceLable- 格式化的产品价格, 包括[language]语言的货币符号。

- price- 以货币最小单位（戈比）表示的价格。
- currency- 货币代码ISO 4217。
- language使用 BCP 47编码指定的语言。
- title产品名称，以FURuStoreProduct::language语言表示。
- description产品描述，以FURuStoreProduct::language语言表示。
- imageUrl- 图片链接。
- promoImageUrl- 促销图片链接。
- subscription- 订阅描述，仅对于类型为 subscription 的产品返回。

产品类型

```
UENUM(BlueprintType)enum class EURuStoreProductType : uint8{ ``NON_CONSUMABLE UMETA(DisplayName = "NON_CONSUMABLE" ), ``CONSUMABLE UMETA(DisplayName = "CONSUMABLE" ), ``SUBSCRIPTION UMETA(DisplayName = "SUBSCRIPTION" )};
```

产品状态

```
UENUM(BlueprintType)enum class EURuStoreProductStatus : uint8{ ``ACTIVE UMETA(DisplayName = "ACTIVE" ), ``INACTIVE UMETA(DisplayName = "INACTIVE" )};
```

订阅结构

```
USTRUCT(BlueprintType)struct FURuStoreProductSubscription{ ``GENERATED_USTRUCT_BODY() ``FURuStoreProductSubscription( ) ``{ ``introductoryPrice = "" ; ``introductoryPriceAmount = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FURuStoreSubscriptionPeriod subscriptionPeriod; ``UPROPERTY(BlueprintReadOnly) ``FURuStoreSubscriptionPeriod freeTrialPeriod; ``UPROPERTY(BlueprintReadOnly) ``FURuStoreSubscriptionPeriod gracePeriod; ``UPROPERTY(BlueprintReadOnly) ``FString introductoryPrice; ``UPROPERTY(BlueprintReadOnly) ``FString introductoryPriceAmount; ``UPROPERTY(BlueprintReadOnly) ``FURuStoreSubscriptionPeriod introductoryPricePeriod;};
```

- subscriptionPeriod订阅宽限期。
- freeTrialPeriod– 订阅试用期。
- gracePeriod– 订阅宽限期。
- introductoryPrice– 格式化的介绍性订阅价格，包括FURuStoreProduct::language语言的货币符号。
- introductoryPriceAmount– 以货币最小单位（戈比）表示的介绍性价格。
- introductoryPricePeriod– 介绍性价格计算周期。

订阅周期的期限结构体

```
USTRUCT(BlueprintType)struct FURuStoreSubscriptionPeriod{ ``GENERATED_USTRUCT_BODY() ``FURuStoreSubscriptionPeriod() ``{ ``years = 1970; ``months = 1; ``days = 1; ``} ``UPROPERTY(BlueprintReadOnly) ``int years; ``UPROPERTY(BlueprintReadOnly) ``int months; ``UPROPERTY(BlueprintReadOnly) ``int days;};
```

- years– 年数。
- months– 月数。

- days– 天数。

Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。

所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

```
USTRUCT(BlueprintType)struct RUSTORECORE_API
FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = ""
; ``description = ""; ``} ``UPROPERTY(BlueprintReadOnly) ``FString
name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};
```

- name– 错误名称。
- description– 错误描述。

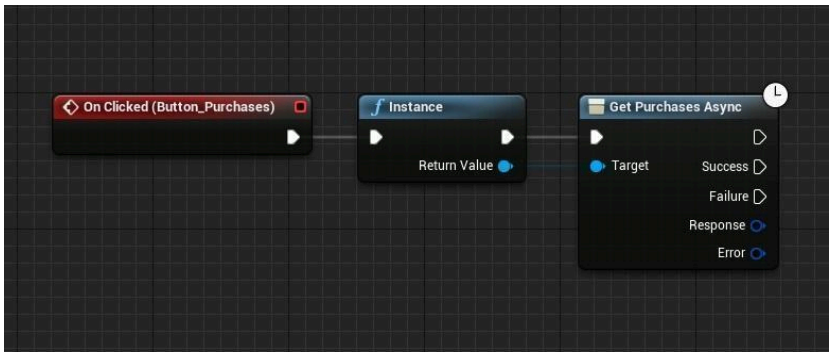
获取购买列表

要获取购买列表，请使用GetPurchases()方法：

调用获取用户购买列表的方法

```
long requestId = URuStoreBillingClient::Instance()->GetPurchases( ``[]( long requestId,
TSharedPtr<FURuStorePurchasesResponse, ESPMode::ThreadSafe> response) { ``// Process
response ``}, ``[]( long requestId, TSharedPtr<FURuStoreRuStoreError, ESPMode::ThreadSafe>
error) { ``// Process error ``});
```

Blueprint实现：



Success回调返回FURuStorePurchasesResponse结构体，在Response参数中：

响应**GetPurchases**

```
USTRUCT(BlueprintType)struct
FURuStorePurchasesResponse{ ``GENERATED_USTRUCT_BODY() ``UPROPERTY(BlueprintReadOnly
) ``TArray<FURuStorePurchase> purchases;};
```

- purchases– 购买列表。

结构体包含关于购买的信息

```
USTRUCT(BlueprintType)struct
FURuStorePurchase{ ``GENERATED_USTRUCT_BODY() ``FURuStorePurchase() ``{ ``purchaseId
= ""; ``productId = ""; ``invoiceId = ""; ``description = ""; ``language = ""
```

; ``purchaseTime = FDateTime(0); ``orderId = "" ; ``amountLabel = "" ; ``amount = 0; ``currency = "" ; ``quantity = 0; ``purchaseState = EURuStorePurchaseState::CANCELLED; ``developerPayload = "" ; ``subscriptionToken = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString purchaseId; ``UPROPERTY(BlueprintReadOnly) ``FString productId; ``UPROPERTY(BlueprintReadOnly) ``FString invoiceId; ``UPROPERTY(BlueprintReadOnly) ``FString description; ``UPROPERTY(BlueprintReadOnly) ``FString language; ``UPROPERTY(BlueprintReadOnly) ``FDateTime purchaseTime; ``UPROPERTY(BlueprintReadOnly) ``FString purchaseTimeLabel; ``UPROPERTY(BlueprintReadOnly) ``FString orderId; ``UPROPERTY(BlueprintReadOnly) ``FString amountLabel; ``UPROPERTY(BlueprintReadOnly) ``int amount; ``UPROPERTY(BlueprintReadOnly) ``FString currency; ``UPROPERTY(BlueprintReadOnly) ``int quantity; ``UPROPERTY(BlueprintReadOnly) ``EURuStorePurchaseState purchaseState; ``UPROPERTY(BlueprintReadOnly) ``FString developerPayload; ``UPROPERTY(BlueprintReadOnly) ``FString subscriptionToken;};

- purchaseId– 购买标识符。
- productId– 产品标识符。
- description– 购买描述。
- invoiceId– 发票标识符。
- language使用 BCP 47 编码指定的语言。
- purchaseTime– 购买时间
- purchaseTimeLabel– 购买时间（格式）DD.MM.YYYYHH:MM:SS
- orderId– 应用程序生成的唯一支付标识符（uuid）。
- amountLable– 格式化的购买价格，包括FURuStorePurchase::language语言的货币符号。
- amount– 以货币最小单位表示的介绍性价格。
- currency– 货币代码ISO 4217。
- quantity– 产品数量。
- purchaseState– 购买状态。
- developerPayload– 开发者指定的字符串，包含关于订单的附加信息。
- subscriptionToken– 用于在服务器端验证购买的令牌。

购买状态

UENUM(BlueprintType)enum class EURuStorePurchaseState : uint8{ ``CREATED UMETA(DisplayName = "CREATED" ), ``INVOICE_CREATED UMETA(DisplayName = "INVOICE_CREATED"), ``CONFIRMED UMETA(DisplayName = "CONFIRMED" ), ``PAID UMETA(DisplayName = "PAID UMETA"), ``CANCELLED UMETA(DisplayName = "CANCELLED" ), ``CONSUMED UMETA(DisplayName = "CONSUMED"), ``CLOSED UMETA(DisplayName = "CLOSED")};

- CREATED- 已创建。
- INVOICE_CREATED- 已创建，等待支付。
- CONFIRMED- 已确认。
- PAID- 已支付。

- CANCELLED- 购买已取消。
- CONSUMED- 已确认购买消耗。
- CLOSED- 订阅已取消。

Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。

所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name– 错误名称。
- description– 错误描述。

获取购买信息

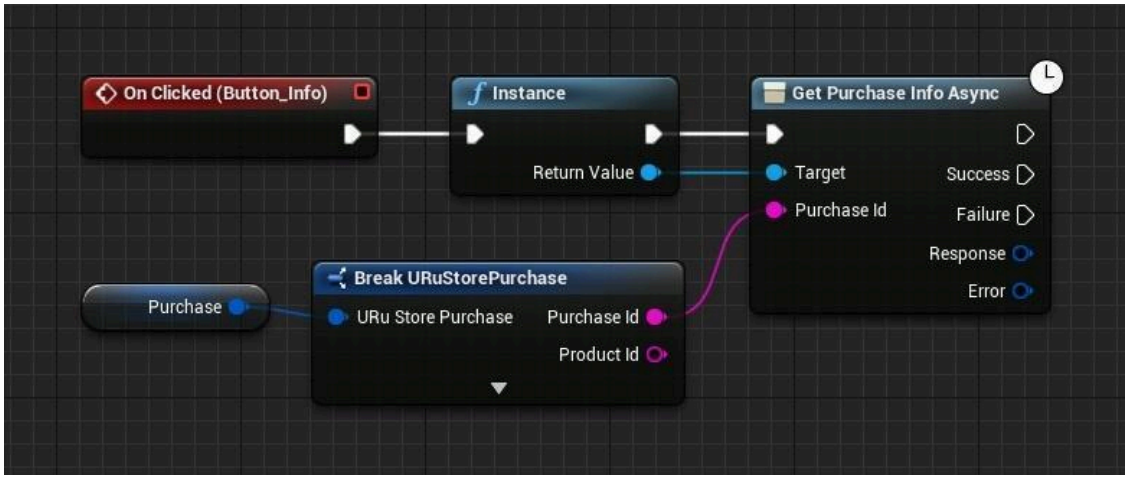
要获取关于购买的信息，请使用 GetPurchaseInfo 方法：

调用取消购买方法。

long requestId = URuStoreBillingClient::Instance()->GetPurchaseInfo(``purchaseId, ``[](long requestId, TSharedPtr<FURuStorePurchase, ESPMode::ThreadSafe> response) { ``// Process response ``}, ``[]( long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

- purchaseId– 购买标识符。

Blueprint实现：



Success回调返回FURuStorePurchase结构体，在Response参数中：

结构体包含关于购买的信息

USTRUCT(BlueprintType)struct FURuStorePurchase{ ``GENERATED_USTRUCT_BODY() ``FURuStorePurchase() ``{ ``purchaseId

= "" ; ``productId = "" ; ``invoiceId = "" ; ``description = "" ; ``language = "" ; ``purchaseTime = FDateTime(0); ``orderId = "" ; ``amountLabel = "" ; ``amount = 0; ``currency = "" ; ``quantity = 0; ``purchaseState = EURuStorePurchaseState::CANCELLED; ``developerPayload = "" ; ``subscriptionToken = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString purchaseId; ``UPROPERTY(BlueprintReadOnly) ``FString productId; ``UPROPERTY(BlueprintReadOnly) ``FString invoiceId; ``UPROPERTY(BlueprintReadOnly) ``FString description; ``UPROPERTY(BlueprintReadOnly) ``FString language; ``UPROPERTY(BlueprintReadOnly) ``FDateTime purchaseTime; ``UPROPERTY(BlueprintReadOnly) ``FString purchaseTimeLabel; ``UPROPERTY(BlueprintReadOnly) ``FString orderId; ``UPROPERTY(BlueprintReadOnly) ``FString amountLabel; ``UPROPERTY(BlueprintReadOnly) ``int amount; ``UPROPERTY(BlueprintReadOnly) ``FString currency; ``UPROPERTY(BlueprintReadOnly) ``int quantity; ``UPROPERTY(BlueprintReadOnly) ``EURuStorePurchaseState purchaseState; ``UPROPERTY(BlueprintReadOnly) ``FString developerPayload; ``UPROPERTY(BlueprintReadOnly) ``FString subscriptionToken;};

Failure回调返回FURuStoreError结构体，包含错误信息。

所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name– 错误名称。
- description– 错误描述。

购买产品

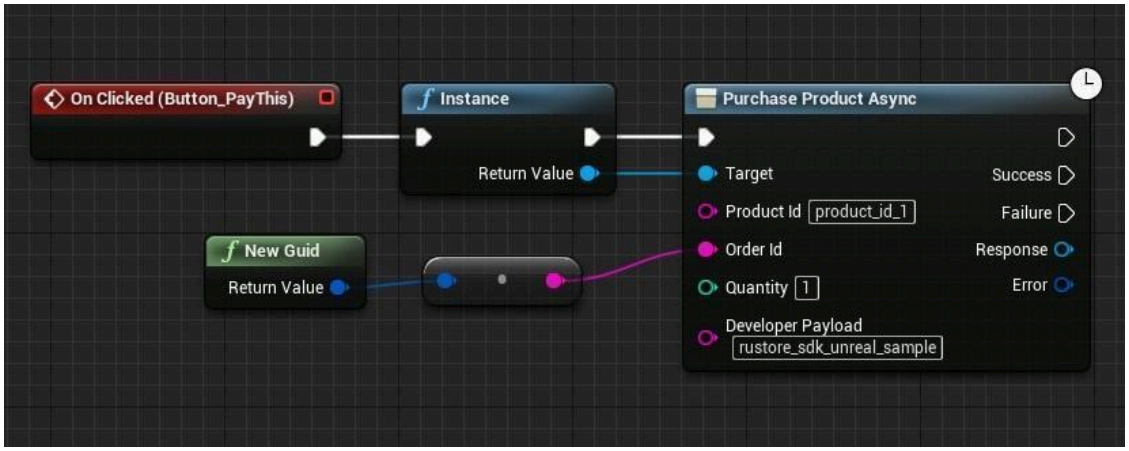
要调用购买产品，请使用 PurchaseProduct() 方法。

调用购买产品的方法

long requestId = URuStoreBillingClient::Instance()->PurchaseProduct(``productId, ``orderId, ``quantity, ``developerPayload, ``[]( long requestId, TShardPtr<FURuStorePaymentResult, ESPMode::ThreadSafe> response) { ``// Process response ``}, ``[](long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

- stringproductId– 产品标识符。
- intquantity– 产品数量。
- stringdeveloperPayload– AnyApp开发者的附加信息。

Blueprint实现：



Success 回调返回由 UE 管理的指针（无需手动删除）到 URuStorePaymentResultClass 类在 Response 参数中：

购买结果的结构体

UCLASS(BlueprintType)class RUSTOREBILLING_API URuStorePaymentResultClass : public UObject{ ``GENERATED_BODY()};USTRUCT(BlueprintType)struct RUSTOREBILLING_API FURuStorePaymentResult{ ``GENERATED_USTRUCT_BODY() ``virtual ~FURuStorePaymentResult() {} ``virtual FString GetTypeName() { return "FURuStorePaymentResult" ; }};

购买结果的结构体

UCLASS(BlueprintType)class RUSTOREBILLING_API URuStoreSuccess : public URuStorePaymentResultClass{ ``GENERATED_BODY()public : ``UPROPERTY(BlueprintReadOnly) ``FURuStoreSuccess value;};USTRUCT(BlueprintType)struct RUSTOREBILLING_API FURuStoreSuccess : public FURuStorePaymentResult{ ``GENERATED_USTRUCT_BODY() ``FURuStoreSuccess() ``{ ``orderId = "" ; ``purchaseId = "" ; ``productId = "" ; ``invoiceId = "" ; ``subscriptionToken = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString orderId; ``UPROPERTY(BlueprintReadOnly) ``FString purchaseId; ``UPROPERTY(BlueprintReadOnly) ``FString productId; ``UPROPERTY(BlueprintReadOnly) ``FString invoiceId; ``UPROPERTY(BlueprintReadOnly) ``FString subscriptionToken; ``virtual FString GetTypeName() override { return "FURuStoreSuccess" ; }};

购买结果的结构体

UCLASS(BlueprintType)class RUSTOREBILLING_API URuStoreCancelled : public URuStorePaymentResultClass{ ``GENERATED_BODY()public : ``UPROPERTY(BlueprintReadOnly) ``FURuStoreCancelled value;};USTRUCT(BlueprintType)struct RUSTOREBILLING_API FURuStoreCancelled : public FURuStorePaymentResult{ ``GENERATED_USTRUCT_BODY() ``FURuStoreCancelled() ``{ ``purchaseId = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString purchaseId; ``virtual FString GetTypeName() override { return "FURuStoreCancelled" ; }};

购买结果的结构体

```
UCLASS(BlueprintType)class RUSTOREBILLING_API URuStoreFailure : public
URuStorePaymentResultClass{ ``GENERATED_BODY()public
: ``UPROPERTY(BlueprintReadOnly) ``FURuStoreFailure value;};USTRUCT(BlueprintType)struct
RUSTOREBILLING_API FURuStoreFailure : public
FURuStorePaymentResult{ ``GENERATED_USTRUCT_BODY()public
: ``FURuStoreFailure() ``{ ``purchaseId = "" ; ``invoiceId = "" ; ``orderId = ""
; ``quantity = 0; ``productId = "" ; ``errorCode =
0; ``} ``UPROPERTY(BlueprintReadOnly) ``FString
purchaseId; ``UPROPERTY(BlueprintReadOnly) ``FString
invoiceId; ``UPROPERTY(BlueprintReadOnly) ``FString
orderId; ``UPROPERTY(BlueprintReadOnly) ``int
quantity; ``UPROPERTY(BlueprintReadOnly) ``FString
productId; ``UPROPERTY(BlueprintReadOnly) ``int errorCode; ``virtual FString GetTypeName()
override { return "FURuStoreFailure" ; }};
```

购买结果的结构体

```
UCLASS(BlueprintType)class RUSTOREBILLING_API URuStoreInvalidPaymentState : public
URuStorePaymentResultBase{ ``GENERATED_BODY()public
: ``UPROPERTY(BlueprintReadOnly) ``FURuStoreInvalidPaymentState
value;};USTRUCT(BlueprintType)struct RUSTOREBILLING_API FURuStoreInvalidPaymentState : public
FURuStorePaymentResult{ ``GENERATED_USTRUCT_BODY() ``virtual FString GetTypeName()
override { return "FURuStoreInvalidPaymentState" ; }};
```

- Success成功完成数字产品购买的结果。
- Failure购买数字产品时错误的结果。
- Cancelled数字产品购买被取消的结果。
- InvalidPaymentStateSDK支付错误。可能因deeplink回调不正确而发生。

Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。

所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

```
USTRUCT(BlueprintType)struct RUSTORECORE_API
FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = ""
; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString
name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};
```

- name– 错误名称。
- description– 错误描述。

购买消耗（确认）

RuStore 包含以下类型的产品：

- CONSUMABLE– 消耗的（可以多次购买，例如应用中的水晶）。
- NON_CONSUMABLE– 非消耗的（只能购买一次，例如禁用应用程序中的广告）。
- SUBSCRIPTION– 订阅（可以购买一段时间，例如流媒体服务的订阅）。

只有处于 PurchaseState.PAID 状态的 CONSUMABLE 类型产品需要消耗。

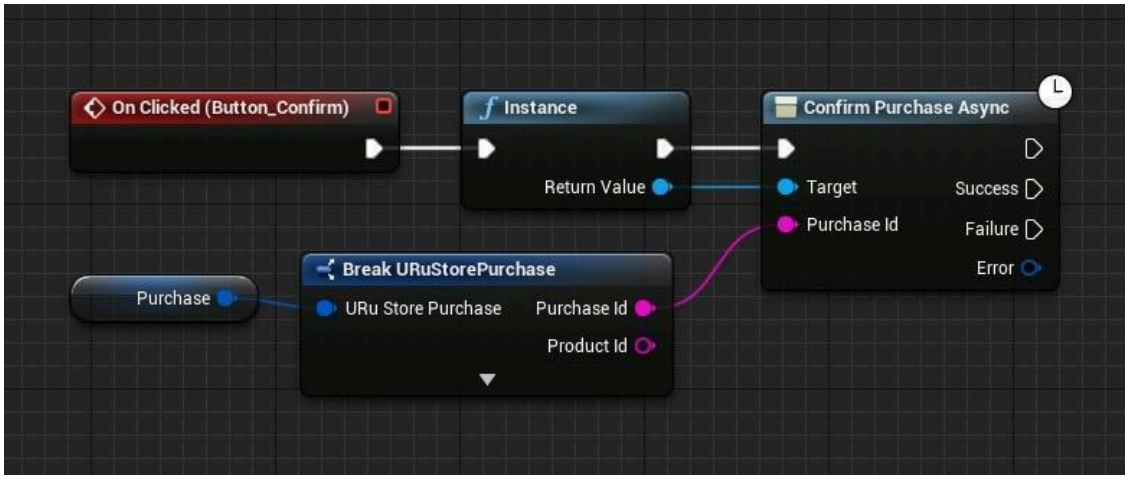
您可以使用 confirmPurchase 方法来消耗购买：

调用确认方法

long requestId = RuStoreBillingClient::Instance()->ConfirmPurchase(``purchaseId, ``[](long requestId) { ``// Process error ``}, ``[]( long requestId, TSharedPtr<FURuStoreConfirmPurchaseResponse, ESPMode::ThreadSafe> response) { ``// Process response ``});

- purchaseId– 购买标识符。

Blueprint实现：



Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。

所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name– 错误名称。
- description– 错误描述。

取消购买

要取消购买，您可以使用 DeletePurchase 方法：

调用取消购买方法。

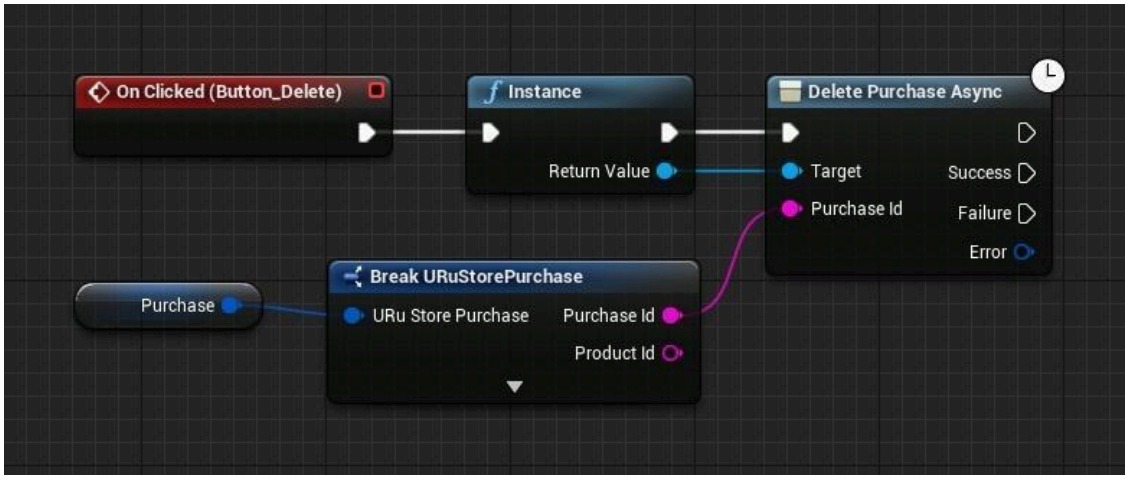
long requestId = URuStoreBillingClient::Instance()->DeletePurchase(``purchaseId, ``[](long requestId) { ``// Process response ``}, ``[]( long requestId, TSharedPtr<FURuStoreRuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

- purchaseId– 购买标识符。

如果您有与删除购买相关的逻辑，请使用此方法。

购买将在20分钟后自动取消，或者在同一客户端进行重复购买时取消。

Blueprint实现：



Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。

所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

```
USTRUCT(BlueprintType)struct RUSTORECORE_API
FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = ""
; ``description = ""; ``} ``UPROPERTY(BlueprintReadOnly) ``FString
name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};
```

- name– 错误名称。
- description– 错误描述。

购买消耗和取消的*流程*

由于产品购买结果模型发生了变化，因此购买的消耗和取消的逻辑也随之发生了变化。

如果出现以下情况，应使用取消购买方法DeletePurchase()：

1. 获取购买列表方法GetPurchases()返回了具有以下状态的购买：
    1. FURuStorePurchaseState::CREATED.
    2. FURuStorePurchaseState::INVOICE_CREATED.
2. 购买方法 PurchaseProduct() 返回 FURuStorePaymentResult::CANCELLED。
3. 购买方法 PurchaseProduct() 返回 FURuStorePaymentResult::FAILURE。

如果获取购买列表的方法 GetPurchases() 返回了类型为 CONSUMABLE 且状态为 FURuStorePurchaseState::PAID 的购买，则需要使用产品消耗方法 ConfirmPurchase()。

错误处理

在 SDK 方法的 onFailure 处理程序中传递发生的错误。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreRuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreRuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name– 错误名称。
- description– 错误描述。

可能的错误：

- RuStoreNotInstalledException – 用户设备上未安装 RuStore。
- RuStoreOutdatedException – 用户设备上安装的 RuStore应用程序不支持支付。
- RuStoreUserUnauthorizedException –用户未在 RuStore 中授权。
- RuStoreApplicationBannedException – 应用程序在 RuStore 中被禁止。
- RuStoreUserBannedException – 用户在 RuStore 中被禁止。
- RuStoreException – RuStore的基本错误，其他错误均继承自此。

调用 PurchaseProduct 方法时，错误会自动处理。

如果在 SDK 初始化时传递了参数 allowNativeErrorHandling == true，在出现错误时，除了调用相应的 onFailure 处理程序外，该错误还会传递到原生 SDK 的 resolveForBilling 方法中，以便向用户显示错误对话框：

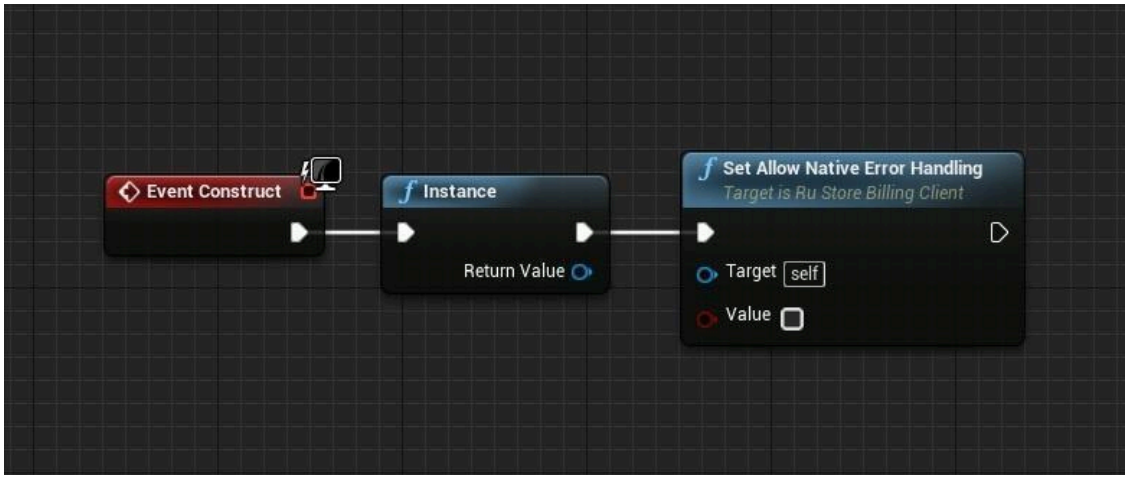
错误处理

public fun RuStoreException.resolveForBilling(context: Context)

在初始化后，可以通过设置 AllowNativeErrorHandling 属性来更改此行为：

禁止原生的错误处理

RuStoreBillingClient::Instance()->SetAllowNativeErrorHandling(false);



更换主题

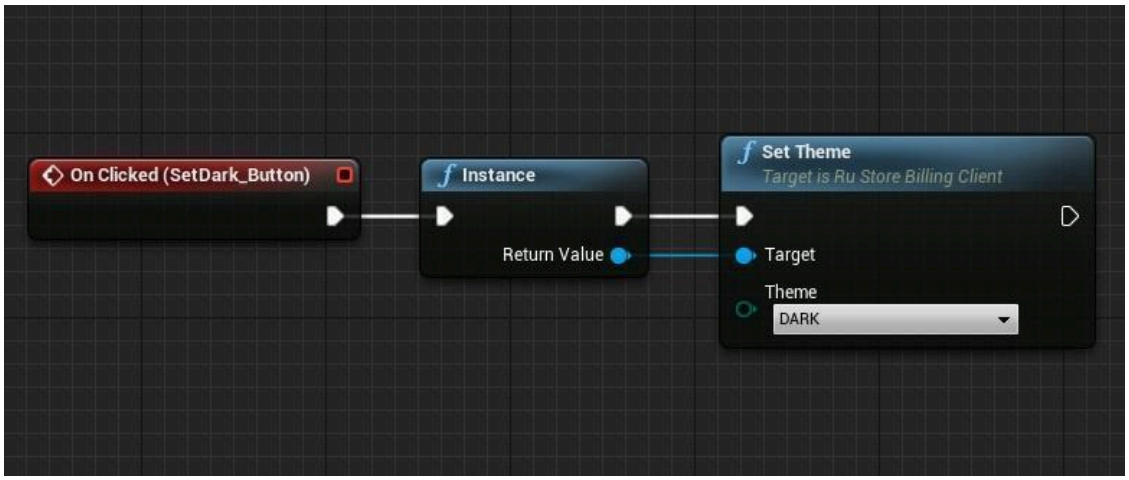
为了动态更换主题，必须使用 SetTheme 方法:

调用主题安装方法

```
EURuStoreTheme theme =
EURuStoreTheme::DARK;URuStoreBillingClient::Instance()->SetTheme(theme);
```

- theme– 来自 EURuStoreTheme 枚举的主题类型。

Blueprint实现：

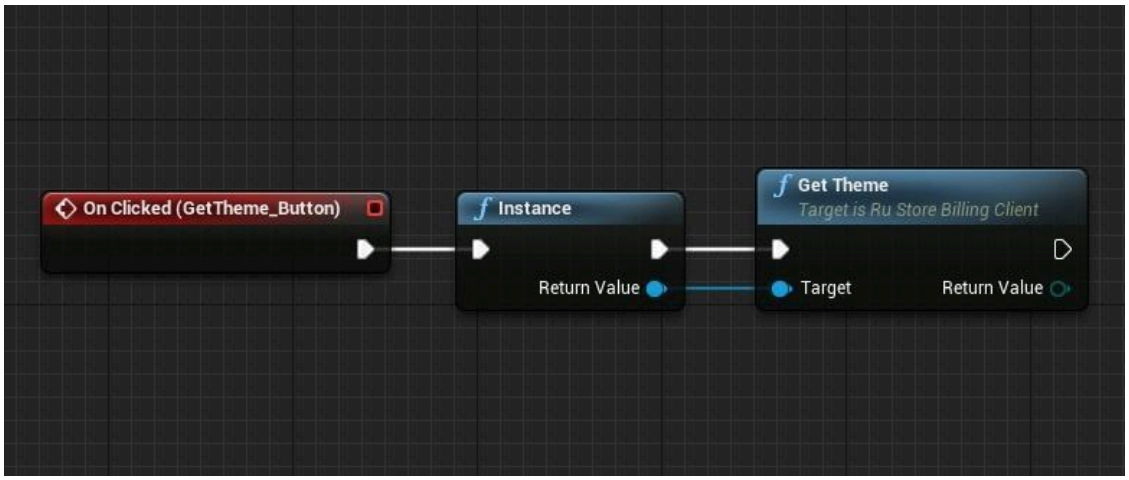


可以使用 GetTheme 方法获取关于已设置的主题的信息:

调用主题安装方法

```
EURuStoreTheme theme = URuStoreBillingClient::Instance()->GetTheme();
```

Blueprint实现：



主题类型

```
UENUM(BlueprintType)enum class EURuStoreTheme : uint8{    ``DARK UMETA(DisplayName =
"DARK" ),    ``LIGHT UMETA(DisplayName =  "LIGHT" )};
```

变更记录

插件版本3.0

- 添加了 blueprint 方法的异步选项，取代对事件的订阅。
- 添加了更改主题（暗色/亮色）的功能。
- 在 FURuStoreProduct 结构体中，productType 字段的默认值从 CONSUMABLE 更改为 NON_CONSUMABLE。
- RuStoreBilling_UPL_Android.xml 不再向 GameActivity 和 SplashActivity 添加 android:exported 属性。
- 错误修复。

插件版本0.3

- 插件内部更新。

依赖列表 **for in-app** 支付 **toc**:**true**

RuStore SDK 连接推送通知

了解如何根据您的开发环境在您的应用程序中集成推送通知：

Kotlin

关键词：RuStore，开发者，文档，RuStoreSDK，集成推送通知

概述

为了推送通知的运行，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore 应用程序。
2. RuStore 应用程序必须支持推送通知的功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 应用程序中获得授权。
5. 应用程序的签名指纹必须与开发者控制台中添加的指纹匹配。

添加存储库

连接本地存储库：

repositories { ``maven { ``url = uri("https://artifactory-external.vkpartner.ru/artifactory/maven") ``}}

集成依赖项

在您的配置文件中添加以下代码以连接依赖项：

dependencies { ``implementation("ru.rustore.sdk:pushclient:1.2.0")}

编辑您的应用程序的清单

声明扩展 RuStoreMessagingService 的服务：

< service ``android:name = ".MyRuStoreMessagingService" ``android:exported = "true" ``tools:ignore = "ExportedService" > ``< intent-filter > ``< action android:name = "ru.rustore.sdk.pushclient.MESSAGING_EVENT" /> ``</ intent-filter ></ service >

如果您想更改标准通知的图标或颜色，可以添加以下元数据：

`<`
`meta-data`
` android:name="ru.rustore.sdk.pushclient.default_notification_icon"<br> android:resource` `=` `"@drawable/ic_baseline_android_24"` `/>`
`<`
`meta-data`
` android:name="ru.rustore.sdk.pushclient.default_notification_color"<br> `android:resource = "@color/your_favorite_color" />

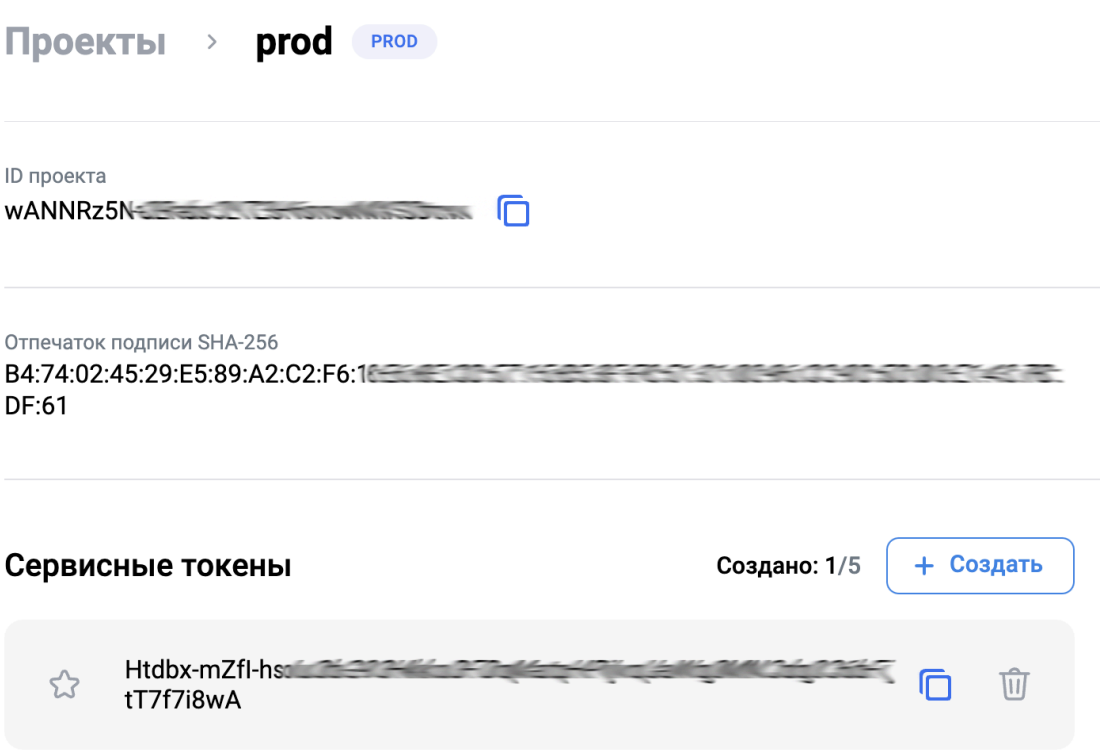
您可以添加以下元数据以重写通知渠道：

< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_channel_id" ``android:value = "@string/pushes_notification_channel_id" />

添加推送通知渠道时，您必须自行创建该渠道。

初始化

要进行初始化，需要在RuStore 控制台系统中获得项目 ID。为此，请转到应用程序页面的“推送通知”部分，然后选择“项目”。



要进行初始化，请在项目的 Application 中添加以下代码：

classApp: Application() { ``override fun onCreate() { ``super .onCreate() ``RuStorePushClient.init( ``application = this , ``projectId = "i5UTx96jw6c1C9LvdlE4cdNrWHMNyRBt" , ``logger = DefaultLogger() ``) ``}}

- application—Application 类的实例;
- projectId—您在 VKPNS 系统中项目的标识符;
- logger—日志器, 默认使用 logcat 输出。

事件日志记录

如果要记录推送通知库事件，请在 RuStorePushClient.init 调用中添加 logger 参数（初始化时该参数为可选参数）。

为此, 您需要实现 Logger 接口:

接口**Logger**

interface Logger { ``fun verbose(message: String, throwable: Throwable? = null ) ``fun debug(message: String, throwable: Throwable? = null) ``fun info(message: String, throwable: Throwable? = null ) ``fun warn(message: String, throwable: Throwable? = null) ``fun error(message: String, throwable: Throwable? = null ) ``fun createLogger(tag: String): Logger}

如果 Logger 没有被传递, 将使用默认的实现, 它使用 AndroidLog。

日志器实现示例

public class DefaultLogger(``private val tag: String? = null ,) : Logger { ``override fun verbose(message: String, throwable: Throwable?) { ``Log.v(tag, message, throwable) ``} ``override fun debug(message: String, throwable: Throwable?) { ``Log.d(tag, message, throwable) ``} ``override fun info(message: String, throwable: Throwable?) { ``Log.i(tag, message, throwable) ``} ``override fun warn(message: String, throwable: Throwable?) { ``Log.w(tag, message, throwable) ``} ``override fun error(message: String, throwable: Throwable?) { ``Log.e(tag, message, throwable) ``} ``override fun createLogger(tag: String): Logger { ``val newTag = if (this .tag != null) { ``"${this.tag}:$tag" ``} else { ``tag ``} ``return DefaultLogger(newTag) ``}}

检查获得推送通知的可能性

关键词:RuStore，开发者，文档，RuStoreSDK，推送通知，检查获得推送通知。为了推送通知的运行, 必须遵守几个条件:

- 用户的设备上必须安装 RuStore。
- RuStore 必须支持推送通知功能。
- RuStore 应用程序应允许在后台模式下运行。
- 用户必须在 RuStore 中获得授权。

为了检查设备上是否安装了 RuStore, 可以使用 RuStorePushClient.checkPushAvailability 方法:

RuStorePushClient.checkPushAvailability() ``.addOnCompleteListener(object : OnCompleteListener<FeatureAvailabilityResult> { ``override fun onSuccess(result: FeatureAvailabilityResult) { ``when (result) { ``FeatureAvailabilityResult.Available -> { ``// Process push available ``} ``is FeatureAvailabilityResult.Unavailable ->

{ ``result.cause.resolveForPush(requireContext()) ``} ``} ``} ``override fun onFailure(throwable: Throwable) { ``// Process error ``} ``})

使用推送令牌的方法

关键词:RuStore，开发者，文档，RuStoreSDK，推送通知，使用推送令牌，推送令牌#####。用户获取推送令牌

在库初始化之后，您可以使用 RuStorePushClient.getToken() 方法来获取当前用户的push令牌。

如果用户没有推送令牌，则该方法将创建并返回新的推送令牌。

RuStorePushClient.getToken().addOnCompleteListener(object : OnCompleteListener<String?> { ``override fun onFailure(throwable: Throwable) { ``// Process error ``} ``override fun onSuccess(result: String?) { ``// Process success ``}})

删除用户的推送令牌

在初始化库之后，您可以使用 RuStorePushClient.deleteToken() 方法来删除用户当前的推送令牌。

RuStorePushClient.deleteToken().addOnCompleteListener(object : OnCompleteListener<Unit> { ``override fun onFailure(throwable: Throwable) { ``// Process error ``} ``override fun onSuccess(result: Unit) { ``// Process success ``}})

用于处理推送主题的方法

订阅主题的推送通知

在初始化库之后，您可以使用 RuStorePushClient.subscribeToTopic(“your_topic_name”) 方法来订阅主题。

RuStorePushClient.subscribeToTopic("your_topic_name").addOnCompleteListener(object : OnCompleteListener<Unit> { ``override fun onFailure(throwable: Throwable) { ``// Process subscribe error ``} ``override fun onSuccess(result: Unit) { ``// Process subscribe success ``}})

取消订阅主题的推送通知

在初始化库之后，您可以使用 RuStorePushClient.unsubscribeFromTopic(“your_topic_name”) 方法来取消订阅主题。

RuStorePushClient.unsubscribeFromTopic("your_topic_name").addOnCompleteListener(object : OnCompleteListener<Unit> { ``override fun onFailure(throwable: Throwable) { ``// Process unsubscribe error ``} ``override fun onSuccess(result: Unit) { ``// Process unsubscribe success ``}})

从*RuStore SDK* 获取数据

关键词: RuStore，开发者，文档，RuStoreSDK，推送通知 数据 数据获取。为了从RuStoreSDK获取数据，创建您自己的服务，该服务继承自 RuStoreMessagingService。

```
class MessagingService: RuStoreMessagingService() {  ``override fun onNewToken(token: String) {  ``}  ``override fun onMessageReceived(message: RemoteMessage) {  ``}  ``override fun onDeletedMessages() {  ``}  ``override fun onError(errors: List<RuStorePushClientException>) {  ``}  ``}
```

方法

1. **onNewToken**—在接收到新的推送令牌时将被调用。调用此方法后，您的应用程序负责将新的推送令牌传递到自己的服务器。
2. **onMessageReceived**—在接收到新的推送通知时将被调用。如果notification对象中有数据，RuStoreSDK将自动显示通知。如果您不希望RuStoreSDK自动显示通知，应使用data对象，并保留notification对象为空。但无论如何都会调用onMessageReceived方法。可以从message.data字段获取推送通知的payload（Map<String, String>）。
3. **onDeletedMessages**—如果一个或多个推送通知没有被送达到设备，将会调用此方法。这可能是因为通知的生命周期在送达设备前已经结束。在调用此方法时，建议与您的服务器同步，以免错过数据。
4. **onError**—如果在初始化过程中发生错误，将会调用此方法。

可能的错误

- UnauthorizedException—用户在RuStore应用程序中未授权。
- HostAppNotInstalledException—用户设备上未安装RuStore应用程序。
- HostAppBackgroundWorkPermissionNotGranted—RuStore应用程序没有后台运行权限。

所有方法将在后台模式下被调用。

通知结构体

关键词: RuStore，开发者，文档，RuStoreSDK，推送通知，结构体#####，完整通知的结构体

```
public data  class RemoteMessage(  ``val messageId: String?,  ``val priority: Int,  ``val ttl: Int,  ``val collapseKey: String?,  ``val data: Map<String, String>,  ``val rawData: ByteArray?,  ``val notification: Notification?)
```

- messageId—消息的唯一ID。它是每个消息的标识符；
- priority—返回优先级值（当前不考虑）。可能的选项包括：
  - 0 — UNKNOWN:
  - 1 - HIGH;
  - 2 - NORMAL.
- ttl—返回Int类型的推送通知生命周期（以秒为单位）；
- collapseKey—通知组的标识符（当前不考虑）；
- data—一个字典，在其中可以传递额外的通知数据；

- rawData—以二进制数组形式的data字典；
- notification—通知对象。

对象的结构体Notification

public data class Notification(``val #### String?,  ``val body: String?, ``val channelId: String?,  ``val imageUrl: Uri?, ``val color: String?,  ``val icon: String?, ``val clickAction: String?  ``)

- title—通知的标题；
- body—通知的正文；
- channelId—可以指定的渠道，用于发送通知（适用于Android 8.0及更高版本）；
- imageUrl—插入通知中的图片的直接链接（图片大小不超过1兆字节）；
- color—通知的颜色（Notification.color）。颜色需要以十六进制格式传递，为字符串格式（例如："#A52A2A"）；
- icon—通知的图标。图标应存放在应用程序的资源中（res/drawable）。参数的值是一个字符串，与资源的名称匹配：
  - 在res/drawable中有一个名为small_icon.xml的图标，在代码中通过R.drawable.small_icon访问。为了在通知中显示此图标，服务器需要在"icon"参数中设置值"small_icon"。
- clickAction—intent点击通知时，用于打开活动的intent action。

创建发送通知的渠道

对于将要发送通知的渠道，适用以下优先级顺序：

1. 如果推送通知中包含channelId字段，那么RuStoreSDK将把通知发送到这个指定的渠道。同时，您的应用程序负责提前创建这个渠道。
2. 如果推送通知中没有channelId字段，但您的应用程序在AndroidManifest.xml中指定了一个渠道参数，则将使用AndroidManifest.xml中的渠道。您的应用程序负责创建这个渠道。
3. 如果推送通知中没有channelId字段，并且您的应用程序的AndroidManifest.xml中没有设置默认渠道，则RuStoreSDK将自行创建一个渠道并向其发送通知。以后，所有未指定渠道的通知都将发送到这个渠道。

当点击通知时打开Activity

默认情况下，点击通知时，RuStoreSDK会打开具有"android.intent.action.MAIN"action的活动。如果存在clickAction字段，RuStoreSDK将打开符合指定action的Intent filter的活动。

为了在点击通知时（也适用于默认活动）RuStoreSDK能够打开活动，您需要在应用程序的清单中对应的活动元素添加一个字符串。没有这字符串，点击通知时RuStoreSDK将无法打开活动。

事件日志记录

关键词：RuStore，开发者，文档，RuStoreSDK，推送通知，连接，事件日志记录。如果您想要日志记录推送通知库的事件，请在调用 RuStorePushClient.init 时添加 logger 参数—这个参数对于初始化是可选的。

为此，您需要实现 Logger 接口：

```
interface Logger {  ``fun verbose(message: String, throwable: Throwable? =  null )  ``fun debug(message: String, throwable: Throwable? =  null )  ``fun info(message: String, throwable: Throwable? =  null )  ``fun warn(message: String, throwable: Throwable? =  null )  ``fun error(message: String, throwable: Throwable? =  null )  ``fun createLogger(tag: String): Logger}
```

如果 Logger 没有被传递，将使用默认的实现，它使用 AndroidLog。

```
public class DefaultLogger(  ``private val tag: String? =  null ,) : Logger {  ``override fun verbose(message: String, throwable: Throwable?) {  ``Log.v(tag, message, throwable)  ``}  ``override fun debug(message: String, throwable: Throwable?) {  ``Log.d(tag, message, throwable)  ``}  ``override fun info(message: String, throwable: Throwable?) {  ``Log.i(tag, message, throwable)  ``}  ``override fun warn(message: String, throwable: Throwable?) {  ``Log.w(tag, message, throwable)  ``}  ``override fun error(message: String, throwable: Throwable?) {  ``Log.e(tag, message, throwable)  ``}  ``override fun createLogger(tag: String): Logger {  ``val newTag =  if ( this .tag !=  null ) {  ``"${this.tag}:$tag"  ``}  else {  ``tag  ``}  ``return DefaultLogger(newTag)  ``}}
```

错误处理

可能的错误：

- RuStoreNotInstalledException() — 用户设备上未安装 RuStore。
- RuStoreOutdatedException() — 用户设备上安装的 RuStore 不支持推送通知功能。
- RuStoreUserUnauthorizedException() — 用户未在 RuStore 中授权。
- RuStoreFeatureUnavailableException() — RuStore 没有在后台工作的权限。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

如果您想使用 UI 接口来处理错误，请使用 resolveForPush 方法。

```
fun RuStoreException.resolveForPush(context: Context)
```

E2ESDK push-通知的集成测试

进行测试需要遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持推送通知功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 中获得授权。

为了开始测试 SDK，您需要启用测试模式：

```
RuStorePushClient.init(  ``application =  this ,  ``projectId =  "some_project_id" ,  ``testModeEnabled =  true)
```

在测试模式下，将生成测试 push-令牌，并且只有使用以下方法发送的测试 push-通知才会被送达：

val testNotificationPayload = TestNotificationPayload(``title = "Test notification title", ``body = "Test notification message", ``imgUrl = "some_image_http_url", ``data = mapOf("some_key" to "some_value"))RuStorePushClient.sendTestNotification(testNotificationPayload).addOnCompleteListener(object : OnCompleteListener<Unit> { ``override fun onFailure(throwable: Throwable) { ``// Process send test push error ``} ``override fun onSuccess(result: Unit) { ``// Process send test push success ``}})

RuStore SDK 推送通知的变更历史

SDK版本1.1.0

- 错误修复。

SDK版本1.0.0

- 更新了 sdk 的 core 和 analytics 库至 1.0.0 版本1.0.0
- pushclient被拆分成几个模块，这些模块是以传递性地连接的：
  - ru.rustore.sdk:push-common
  - ru.rustore.sdk:push-core
  - ru.rustore.sdk:push-core-network

SDK版本0.7.0

- 修正了 “checkPushAvailability” 方法的不正确工作。
- 方法 “checkPushAvailability(context: Context)” 已被标记为 deprecated，并将在将来被删除。请使用无参数的版本 “checkPushAvailability()”。
- 在方法 “RuStorePushClient.init” 中添加了参数 analyticsCallback: AnalyticsCallback?。该参数用于支持未来的功能 “分发针对目标群体”。
- 其他错误修复。

SDK版本0.6.0

- 错误修复。

SDK版本0.5.0

- 错误修复。

SDK版本0.4.0

- 更新了处理推送通知的流程。不再需要添加 ProGuard 规则 -keep public class com.vk.push.** extends android.os.Parcelable，可以将其删除。SDK 会提供所有必要的规则；
- 混淆结构已更改。现在，混淆的类不会超出 Push SDK 根包的范围。

SDK版本0.3.0

- 错误修复；
- 添加了 SDK 推送通知的 E2E 集成测试。

SDK版本0.2.0

- 错误修复；
- 添加了用于处理推送主题的方法。

SDK版本0.1.9

- SDK的内部更新。

SDK版本0.1.8

- 修复了()Task API的await方法的操作。

SDK版本0.1.7

- SDK的内部更新。

SDK版本0.1.6

- SDK的内部更新。

SDK版本0.1.5

- SDK的内部更新。

SDK版本0.1.4

- 修复了 “Intent Redirection vulnerability” 安全漏洞，这个漏洞之前阻止了应用在 Google Play 上的发布。

SDK版本0.1.3

- SDK的内部更新。

SDK版本0.1.2

- SDK的内部更新。

SDK版本0.1.1

- SDK的内部更新。
- 当没有推送令牌时，“getToken” 方法将创建并返回一个新的推送令牌。
- “deleteToken” 方法现在在删除当前推送令牌后不再创建新的推送令牌。

SDK版本0.1.0

- SuspendSuspend方法在以下方法中被task API替换：
  - 检查获取推送通知的可能性 — “checkPushAvailability”。
  - 获取用户的推送令牌 — “getToken”。
  - 删除用户的推送令牌 — “deleteToken”。
- 现在，“checkPushAvailability” 方法返回 “FeatureAvailabilityResult” 对象。

SDK版本0.0.9

- SDK的内部更新。

Java

概述

实现示例

查看示例应用程序，了解如何正确集成推送通知。

推送通知的运行条件

为了推送通知的运行，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore 应用程序。
2. RuStore 应用程序必须支持推送通知的功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 应用程序中获得授权。
5. 应用程序的签名指纹必须与添加到 RuStore 控制台系统中的指纹匹配。

添加存储库

连接本地存储库：

`repositories { `
` maven {<br> url = uri(` `"https://artifactory-external.vkpartner.ru/artifactory/maven"` `)`
` }<br>}`

集成依赖项

在您的配置文件中添加以下代码以连接依赖项：

dependencies { ``implementation("ru.rustore.sdk:pushclient:1.2.0")}

编辑您的应用程序的清单

声明扩展 RuStoreMessagingService 的服务：

< service ``android:name = ".MyRuStoreMessagingService" ``android:exported = "true" ``tools:ignore = "ExportedService" > ``< intent-filter > ``< action android:name = "ru.rustore.sdk.pushclient.MESSAGING_EVENT" /> ``</ intent-filter ></ service >

如果您想更改标准通知的图标或颜色，您可以添加以下元数据：

`<` `meta-data`
` android:name="ru.rustore.sdk.pushclient.default_notification_icon"<br> android:resource` `=` `"@drawable/ic_baseline_android_24"` `/>`
`<` `meta-data`
` android:name="ru.rustore.sdk.pushclient.default_notification_color"<br> `android:resource = "@color/your_favorite_color" />

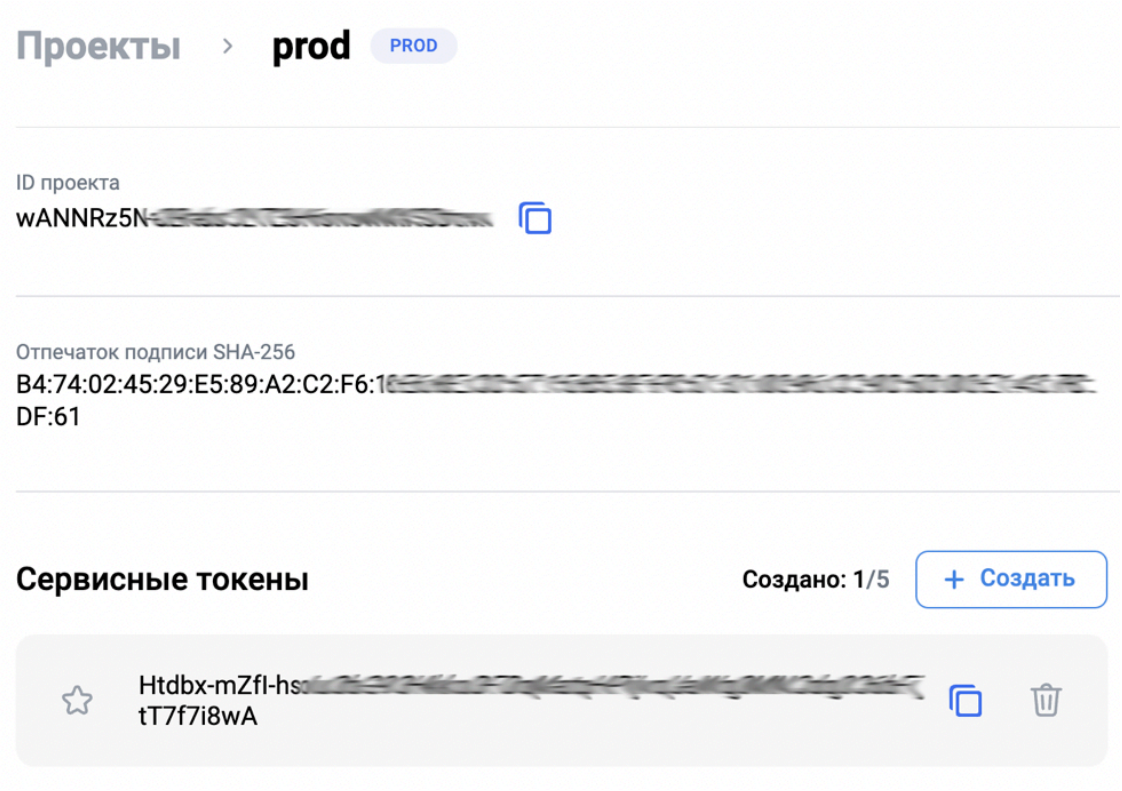
您可以添加以下元数据以重写通知渠道：

< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_channel_id" ``android:value = "@string/pushes_notification_channel_id" />

添加推送通知渠道时，您必须自行创建该渠道。

初始化

要进行初始化，需要在 RuStore 控制台系统中获取项目 ID。为此，在应用程序页面转到 “Push-通知” 部分并选择 “项目”。



要进行初始化，请在项目的 Application 中添加以下代码：

class App : Application() { ``@Override ``public void onCreate() { ``super .onCreate() ``RuStorePushClient.INSTANCE.init(``this , ``"i5UTx96jw6c1C9LvdlE4cdNrWHMNyRBt" , ``DefaultLogger() ``); ``}}

- application—Application 类的实例；
- projectId—您在 RuStore 开发者控制台中项目的标识符；
- logger—日志器，默认使用 logcat 输出（可选）。

事件日志记录

如果您想要日志记录推送通知库的事件，请在调用 RuStorePushClient.init 时添加 logger 参数 — 这个参数对于初始化是可选的。

为此，您需要实现 Logger 接口：

public interface Logger { ``void verbose(String message, Throwable throwable); ``void debug(String message, Throwable throwable); ``void info(String message, Throwable throwable); ``void warn(String message, Throwable throwable); ``void error(String message, Throwable throwable); ``Logger createLogger(String tag);}

如果 Logger 没有被传递，将使用默认的实现，它使用 AndroidLog。

`public` `class` `PushLogger ` `implements` `Logger {`
` privatefinalString tag;<br> public` `PushLogger(String tag) {`
` this.tag = tag;<br> }`
` @Override<br> public` `void` `debug(` `@NonNull` `String message, Throwable throwable) {`
` Log.d(tag, message, throwable);<br> }`
` @Override<br> public` `void` `error(` `@NonNull` `String message, Throwable throwable) {`
` Log.e(tag, message, throwable);<br> }`
` @Override<br> public` `void` `info(` `@NonNull` `String message, ` `@Nullable` `Throwable throwable) {`
` Log.i(tag, message, throwable);<br> }`
` @Override<br> public` `void` `verbose(` `@NonNull` `String message, ` `@Nullable` `Throwable throwable) {`
` Log.v(tag, message, throwable);<br> }`
` @Override<br> public` `void` `warn(` `@NonNull` `String message, ` `@Nullable` `Throwable throwable) {`
` Log.w(tag, message, throwable);<br> }`
` @NonNull<br> @Override`
` publicLogger createLogger(@NonNullString newTag) {<br> String combinedTag = (tag != ` `null` `) ? tag + ` `":"` `+ newTag : newTag;`
` returnnewPushLogger(combinedTag);<br> `}}

用于处理推送主题的方法

订阅主题的推送通知

在初始化库之后，您可以使用 RuStorePushClient.subscribeToTopic("your_topic_name") 方法来订阅主题。

RuStorePushClient.INSTANCE.subscribeToTopic("your_topic_name").addOnCompleteListener(new OnCompleteListener<Void>() { ``@Override ``public void onComplete(Task<Void> task) { ``if (task.isSuccessful()) { ``// Process subscribe success ``} else { ``Exception exception = task.getException(); ``// Process subscribe error ``} ``}});

取消订阅主题的推送通知

在初始化库之后，您可以使用 RuStorePushClient.unsubscribeFromTopic("your_topic_name") 方法来取消订阅主题。

RuStorePushClient.INSTANCE.unsubscribeFromTopic("your_topic_name").addOnCompleteListener(new OnCompleteListener<Void>() { ``@Override ``public void onComplete(Task<Void> task) { ``if (task.isSuccessful()) { ``// Process unsubscribe success ``} else { ``Exception exception = task.getException(); ``// Process unsubscribe error ``} ``}});

使用 *RuStore* 的操作*MessagingService*

RuStoreMessagingService

public class MessagingService extends RuStoreMessagingService { ``@Override ``public void onNewToken(String token) { ``} ``@Override ``public void onMessageReceived(RemoteMessage message) { ``} ``@Override ``public void onDeletedMessages() { ``} ``@Override ``public void onError(List<RuStorePushClientException> errors) { ``}}

onNewToken—在接收到新的推送令牌时将被调用。

调用此方法后，您的应用程序负责将新的推送令牌传递到自己的服务器。

onMessageReceived—在接收到新的推送通知时将被调用。

如果 notification 对象中有数据，SDK 将尝试自动显示通知。

如果您不希望 SDK 自动显示通知，应使用 data 对象，并保持 notification 对象为空。但无论如何都会调用onMessageReceived方法。

可以从message.data字段获取推送通知的payload（Map<String, String>）。

onDeletedMessages—如果一个或多个推送通知没有被送达到设备，将会调用此方法。

这可能是因为通知的生命周期在送达设备前已经结束。

在调用此方法时，建议与您的服务器同步，以免错过数据。

onError—如果在初始化过程中发生错误，将会调用此方法。

可能的错误：

- UnauthorizedException—用户未在 RuStore 中授权。
- HostAppNotInstalledException—用户设备上未安装RuStore应用程序。
- HostAppBackgroundWorkPermissionNotGranted—RuStore 没有在后台工作的权限。

所有上述方法都将在后台线程上被调用。

通知结构体

完整通知的结构体

```
public RemoteMessage(  ``String messageId,  ``int priority,  ``int ttl,  ``String collapseKey,  ``Map<String, String> data,  ``byte [] rawData,  ``Notification notification) {  ``this .messageId = messageId;  ``this .priority = priority;  ``this .ttl = ttl;  ``this .collapseKey = collapseKey;  ``this .data = data;  ``this .rawData = rawData;  ``this .notification = notification;}
```

- messageId—消息的唯一ID。它是每个消息的标识符；
- priority—返回优先级值（当前不考虑）。可能的选项包括：
  - 0 — UNKNOWN:
  - 1 - HIGH;
  - 2 - NORMAL.
- ttl—返回Int类型的推送通知生命周期（以秒为单位）；
- collapseKey—通知组的标识符（当前不考虑）；
- data—一个字典，在其中可以传递额外的通知数据；
- rawData—以二进制数组形式的data字典；
- notification—通知对象。

对象的结构体Notification

```
public Notification(  ``String title,  ``String body,  ``String channelId,  ``Uri imageUrl,  ``String color,  ``String icon,  ``String clickAction) {  ``this .title = title;  ``this .body = body;  ``this .channelId =
```

channelId; ``this .imageUrl = imageUrl; ``this .color = color; ``this .icon = icon; ``this .clickAction = clickAction;}

- title—通知的标题;
- body—通知的正文;
- channelId—可以指定的渠道，用于发送通知（适用于Android 8.0及更高版本）;
- imageUrl—插入通知中的图片的直接链接（图片大小不超过1兆字节）;
- color—通知的颜色（Notification.color）。颜色需要以十六进制格式传递，为字符串格式（例如:“#A52A2A”）;
- icon—通知的图标。图标应存放在应用程序的资源中（res/drawable）。参数的值—一个字符串，与资源的名称匹配:
  - 在res/drawable中有一个名为small_icon.xml的图标，在代码中通过R.drawable.small_icon访问。为了在通知中显示此图标，服务器需要在“icon”参数中设置值“small_icon”。
- clickAction—intent点击通知时，用于打开活动的intent action。

创建发送通知的渠道

对于将要发送通知的渠道，适用以下优先级顺序:

1. 如果推送通知中包含channelId字段，那么RuStoreSDK将把通知发送到这个指定的渠道。同时，您的应用程序负责提前创建这个渠道。
2. 如果推送通知中没有channelId字段，但您的应用程序在AndroidManifest.xml中指定了一个渠道参数，则将使用AndroidManifest.xml中的渠道。您的应用程序负责创建这个渠道。
3. 如果推送通知中没有channelId字段，并且您的应用程序的AndroidManifest.xml中没有设置默认渠道，则RuStoreSDK将自行创建一个渠道并向其发送通知。以后，所有未指定渠道的通知都将发送到这个渠道。

当点击通知时打开Activity

默认情况下，点击通知时，RuStoreSDK会打开具有action “android.intent.action.MAIN”的活动。如果存在clickAction字段，RuStoreSDK将打开符合指定action的Intent filter的活动。

为了在点击通知时（也适用于默认活动）RuStoreSDK能够打开活动，您需要在应用程序的清单中对应的活动元素添加一个字符串。没有这字符串，点击通知时RuStoreSDK将无法打开活动。

错误处理

可能的错误:

- RuStoreNotInstalledException() — 用户设备上未安装 RuStore。
- RuStoreOutdatedException() — 用户设备上安装的 RuStore 不支持推送通知功能。
- RuStoreUserUnauthorizedException() — 用户未在 RuStore 中授权。
- RuStoreFeatureUnavailableException() — RuStore 没有在后台工作的权限。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

如果您想使用 UI 接口来处理错误，请使用 resolveForPush() 方法：

`public` `class` `RuStoreExceptionExtension
{`<br>` publicstaticvoidresolveForPush(RuStoreException exception, Context context)
{
 // Your implementation here to resolve the exception for push`<br>` }
}`

E2ESDK 推送通知的集成测试

进行测试需要遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持推送通知功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 中获得授权。

为了开始测试 SDK，您需要启用测试模式：

RuStorePushClient.INSTANCE.init(``this , ``"some_project_id" , ``true)

在测试模式下，将生成测试 push-令牌，并且只有使用以下方法发送的测试 push-通知才会被送达：

TestNotificationPayload testNotificationPayload = new TestNotificationPayload(``"Test notification title" , ``"Test notification message" , ``"some_image_http_url" , ``data);RuStorePushClient.INSTANCE.sendTestNotification(testNotificationPayload) ``.addOnCompleteListener( new OnCompleteListener<Void>() { ``@Override ``public void onComplete(Task<Void> task) { ``if (task.isSuccessful()) { ``// Process send test push success ``} else { ``Exception exception = task.getException(); ``// Process send test push error ``} ``} ``});

Unreal

概述

为了推送通知的运行，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持推送通知功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 中获得授权。
5. 应用程序的签名指纹必须与 RuStore 控制台中指定的指纹匹配。
6. Unreal Engine 4.26或更高版本。

在项目中集成

1. 从RuStore的官方*gitflic*仓库复制'Plugins'文件夹的内容到您项目中的'*Plugins*'文件夹。重启Unreal Engine，在插件列表中(Edit → Plugins → Project → Mobile)，选择'RuStoreBilling'和'RuStoreCore'插件。
2. 在文件 *"YourProject.Build.cs"* 中，在 PublicDependencyModuleNames 列表中连接模块 "RuStoreCore" 和 "RuStorePush"。

3. 在项目设置中（Edit → Project Settings → Android），设置 Minimum SDK Version 为不低于24，Target SDK Version 为不低于31。

编辑您的应用程序的清单

“RuStorePush” 插件将声明 RuStoreUnityMessagingService 服务。

AndroidManifest.xml

< service ``android:name = "ru.rustore.unitysdk.pushclient.RuStoreUnityMessagingService" ``android:exported = "true" ``tools:ignore = "ExportedService" > ``< intent-filter > ``< action android:name = "ru.rustore.sdk.pushclient.MESSAGING_EVENT" /> ``</ intent-filter ></ service >

如果您想更改标准通知的图标或颜色，可以添加以下元数据：

AndroidManifest.xml

< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_icon" ``android:resource = "@drawable/ic_baseline_android_24" />< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_color" ``android:resource = "@color/your_favorite_color" />

您可以添加以下元数据以重写通知渠道：

AndroidManifest.xml

< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_channel_id" ``android:value = "@string/pushes_notification_channel_id" />

当您添加自己的推送通知渠道时，您需要自行创建该渠道。

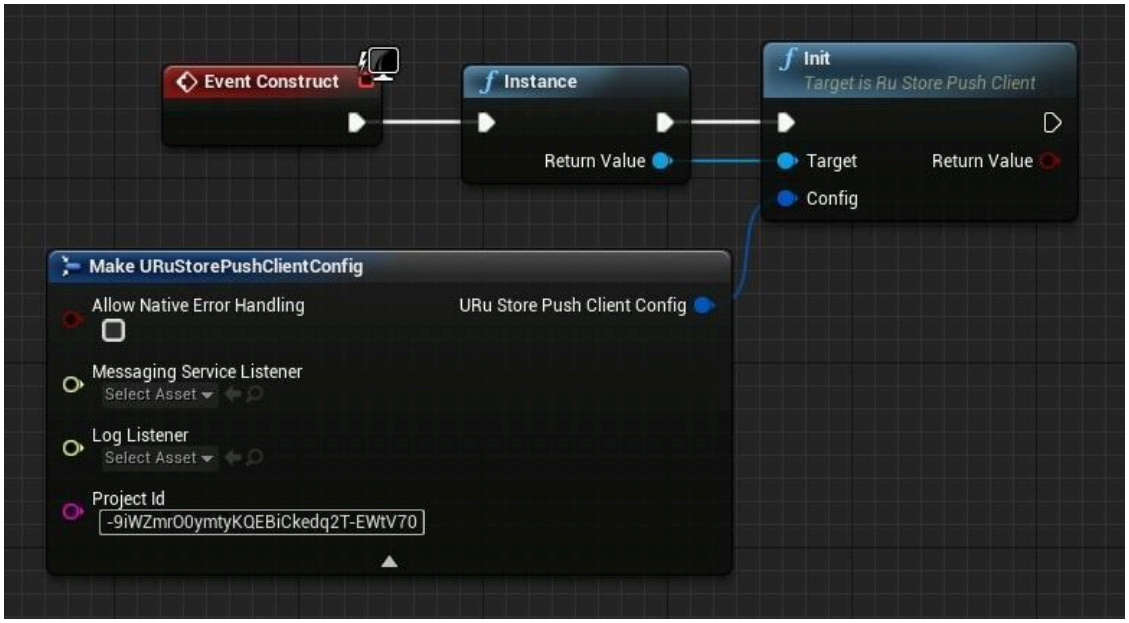
初始化

在调用库的方法之前，必须先调用其初始化：

初始化

FURuStorePushClientConfig config;config.allowNativeErrorHandling = true ;config.messagingServiceListener = pushMessagingServiceListener;config.logListener = pushLogListener;FURuStorePushClient::Instance()->Init(config);

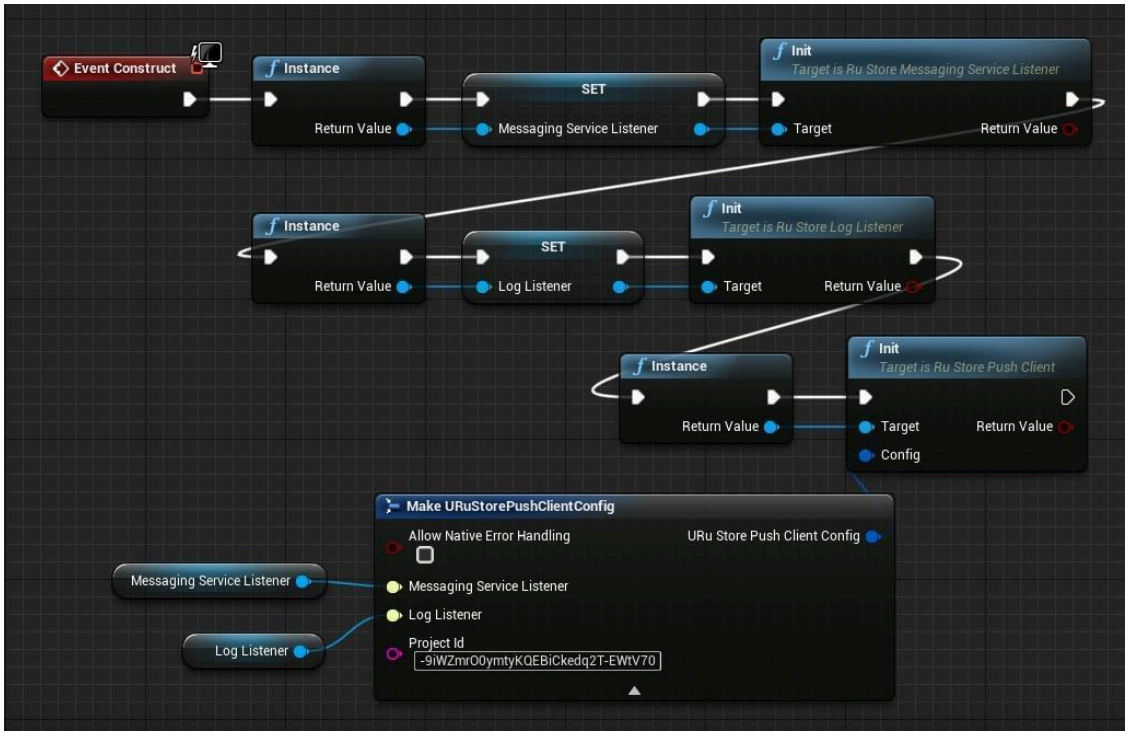
所有与客户端的操作也可以通过Blueprints进行。初始化的例子：



- allowallowNativeErrorHandling – 允许原生 SDK 中的错误处理。
- messagingServiceListener– 实现 IRuStoreMessagingServiceListenerInterface 接口的类对象。
- logListener– 实现 IRuStoreLogListenerInterface 接口的类对象LogListenerInterface
- projectId– 您在 RuStore 控制台的项目标识符

URuStoreMessagingServiceListener 和 URuStoreLogListener 类分别实现了 IRuStoreMessagingServiceListenerInterface 和 IRuStoreLogListenerInterface 接口。它们的应用允许直接从 blueprint 中处理接口事件。

URuStoreMessagingServiceListener 和 URuStoreLogListener 的初始化与 URuStorePushClient 一起进行：



IRuStore接口MessagingServiceListenerInterface

class RUSTOREPUSH_API IRuStoreMessagingServiceListenerInterface{ ``GENERATED_BODY() public: ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Messaging Service Listener Interface") ``void NewTokenResponse(int64 requestId, FString& token); ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Messaging Service Listener Interface") ``void MessageReceivedResponse(int64 requestId, FURuStoreRemoteMessage& message); ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Messaging Service Listener Interface") ``void DeletedMessagesResponse(int64 requestId); ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Messaging Service Listener Interface") ``void ErrorResponse(int64 requestId, TArray<FURuStoreError>& errors);};

- **NewTokenResponseResponse**– 在接收到新的推送令牌时将被调用。调用此方法后，您的应用程序负责将新的推送令牌传递到自己的服务器。
- **MessageReceivedResponse**– 在接收到新的推送通知时将被调用。
- **DeletedMessagesResponse**– 如果一个或多个推送通知没有被送达到设备，将会调用此方法。这可能是因为通知的生命周期在送达设备前已经结束。在调用此方法时，建议与您的服务器同步，以免错过数据。
- **ErrorResponse**– 如果在初始化过程中发生错误，将会调用此方法。

可能的错误：

- UnauthorizedException– 用户未在 RuStore 中授权。
- HostAppNotInstalledException– 用户设备上未安装RuStore应用程序。
- HostAppBackgroundWorkPermissionNotGranted– RuStore 没有在后台工作的权限。

通知结构体。

完整通知的结构体

USTRUCT(BlueprintType)struct RUSTOREPUSH_API FURuStoreRemoteMessage{ ``GENERATED_USTRUCT_BODY()public : ``UPROPERTY() ``FString collapseKey; ``UPROPERTY() ``TMap<FString, FString> data; ``UPROPERTY() ``FString messageId; ``UPROPERTY() ``FURuStoreNotification notification; ``UPROPERTY() ``int priority; ``char * rawData; ``UPROPERTY() ``int ttl;};

- messageId消息的唯一ID。它是每个消息的标识符。
- priority-（当前不考虑）返回优先级值。目前有以下选项：
  - 0 - UNKNOWN.
  - 1 - HIGH.
  - 2 - NORMAL.
- ttlInt类型的推送通知生命周期（以秒为单位）；
- collapseKey-（当前不考虑）通知组的标识符。
- data- 一个字典，在其中可以传递额外的通知数据。
- rawData- data 字典以字节数组形式。
- notification- 通知对象。

对象的结构体**Notification**

USTRUCT(Blueprintable)struct RUSTOREPUSH_API FURuStoreNotification{ ``GENERATED_USTRUCT_BODY()public : ``FURuStoreNotification() ``{ ``title = "0" ; ``body = "0" ; ``channelId = "0" ; ``imageUrl = "0" ; ``color = "0" ; ``icon = "0" ; ``clickAction = "0" ; ``} ``UPROPERTY() ``FString title; ``UPROPERTY() ``FString body; ``UPROPERTY() ``FString channelId; ``UPROPERTY() ``FString imageUrl; ``UPROPERTY() ``FString color; ``UPROPERTY() ``FString icon; ``UPROPERTY() ``FString clickAction;};

- title- 通知的标题。
- body- 通知的正文。
- channelId- 可以指定的渠道，用于发送通知（适用于 Android 8.0 及更高版本）。
- imageUrl- 插入通知中的图片的直接链接（图片大小不超过1兆字节）。
- color- 通知的颜色（Notification.color）。颜色需要以十六进制格式传递，为字符串格式（例如："#A52A2A"）。
- icon- 通知的图标。图标应位于应用程序的资源中（res/drawable）。参数的值是一个字符串，与资源名称相匹配。例如：在 res/drawable 中有一个名为 small_icon.xml 的图标，它在代码中通过 R.drawable.small_icon 访问。为了在通知中显示这个图标，服务器需要在 "icon" 参数中放入 "small_icon" 的值。
- clickAction- 点击通知时，用于打开activity的intent action。

IRuStore接口**LogListenerInterface**

class RUSTOREPUSH_API IRuStoreLogListenerInterface{ ``GENERATED_BODY()public : ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Log Listener Interface") ``void LogResponse(int64 requestId, FString& logString); ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Log Listener Interface") ``void LogWarningResponse(int64 requestId, FString& logString); ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Log Listener Interface") ``void LogErrorResponse(int64

requestId, FString& logString); ``UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "RuStore Log Listener Interface" )  ``void LogExceptionResponse(int64 requestId, FURuStoreError& error);};

- LogResponse- 当普通日志记录出现在事件日志中时将被调用。
- LogWarningResponse- 当警告记录出现在事件日志中时将被调用。
- LogErrorResponse- 当错误日志记录出现在事件日志中时将被调用。
- LogExceptionResponse- 当异常日志记录出现在事件日志中时将被调用。

错误结构体

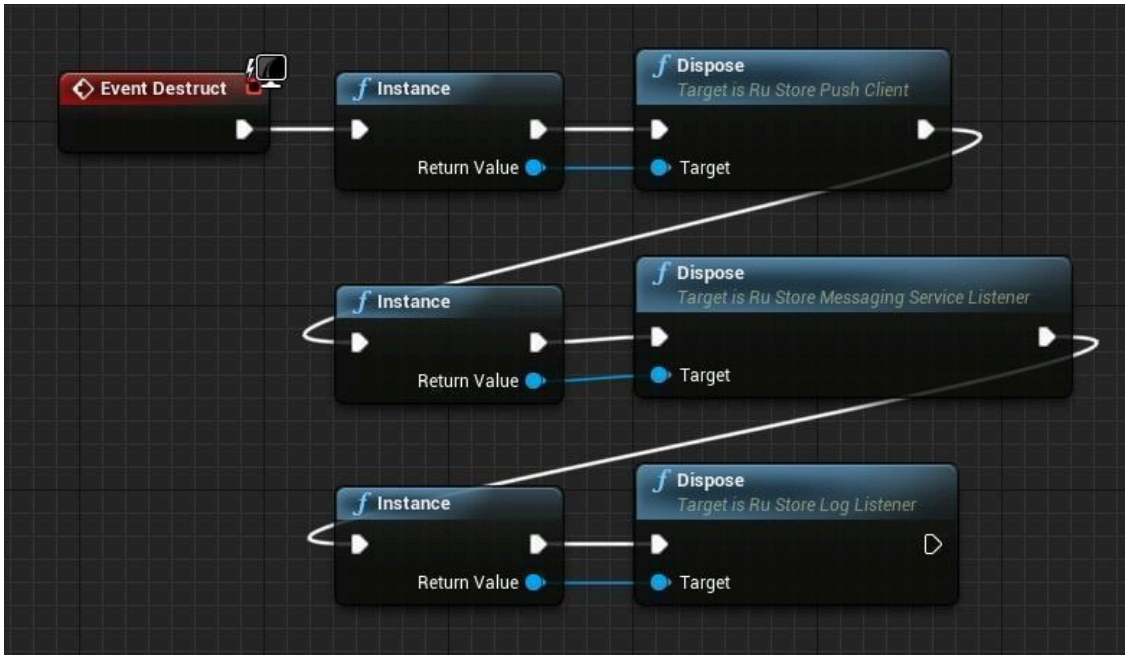
USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY()  ``FURuStoreError() ``{  ``name = "" ; ``description = "" ;  ``} ``UPROPERTY(BlueprintReadOnly)  ``FString name; ``UPROPERTY(BlueprintReadOnly)  ``FString description;};

- name- 错误名称。
- description- 错误描述。

对 URuStorePushClient、URuStoreMessagingServiceListener、URuStoreLogListener 调用 Init() 将对象绑定到场景根部。如果不打算进一步使用这些对象，为了释放内存，需要执行 Dispose() 方法。调用 Dispose() 方法将从根部解绑对象，并安全地完成所有已发送的请求。

反初始化

URuStorePushClient::Instance()->Dispose();URuStoreMessagingServiceListener::Instance()->Dispose();URuStoreLogListener::Instance()->Dispose();



检查获得推送通知的可能性

要检查支付的可用性，需要使用 CheckPushAvailability() 方法。在调用时，将检查以下条件：

1. 用户的设备上必须安装 RuStore。

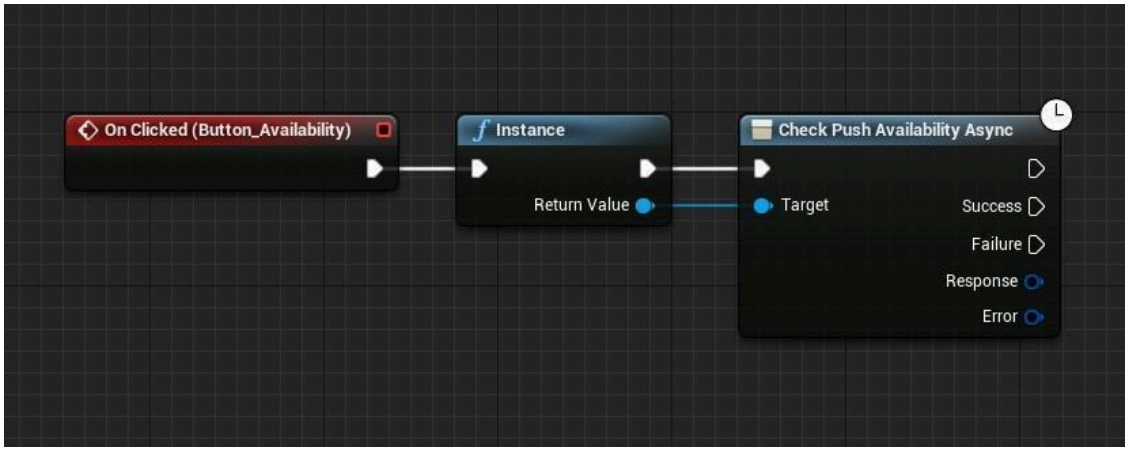
2. RuStore 必须支持推送通知功能。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. 应用程序的签名指纹必须与 RuStore 控制台中指定的指纹匹配。

每次调用 CheckPushAvailability() 都会返回一个在应用程序单次启动期间唯一的 requestId。每个事件都会返回触发该事件的请求的 requestId。

RuStore实现示例**PushClient.CheckPushAvailability**

long responseId = URuStorePushClient::Instance()->CheckPushAvailability(``[]( long responseId, TSharedPtr<FUFeatureAvailabilityResult, ESPMode::ThreadSafe> response) { ``// Process response ``}, ``[](long responseId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

Blueprint实现：



Success回调返回FURuStoreFeatureAvailabilityResult结构体，在Response参数中：

响应**CheckPushAvailability**

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreFeatureAvailabilityResult{ ``GENERATED_USTRUCT_BODY() ``FURuStoreFeatureAvailabilityResult() ``{ ``isAvailable = false ; ``} ``UPROPERTY(BlueprintReadWrite) ``bool isAvailable; ``UPROPERTY(BlueprintReadWrite) ``FURuStoreError cause;};

isAvailable- 表示是否满足执行支付的条件。

cause- 错误信息。

Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。所有可能的FURuStoreException错误在"错误处理"部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = ""

; ``description = ""; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name- 错误名称。
- description- 错误描述。

使用推送令牌的方法

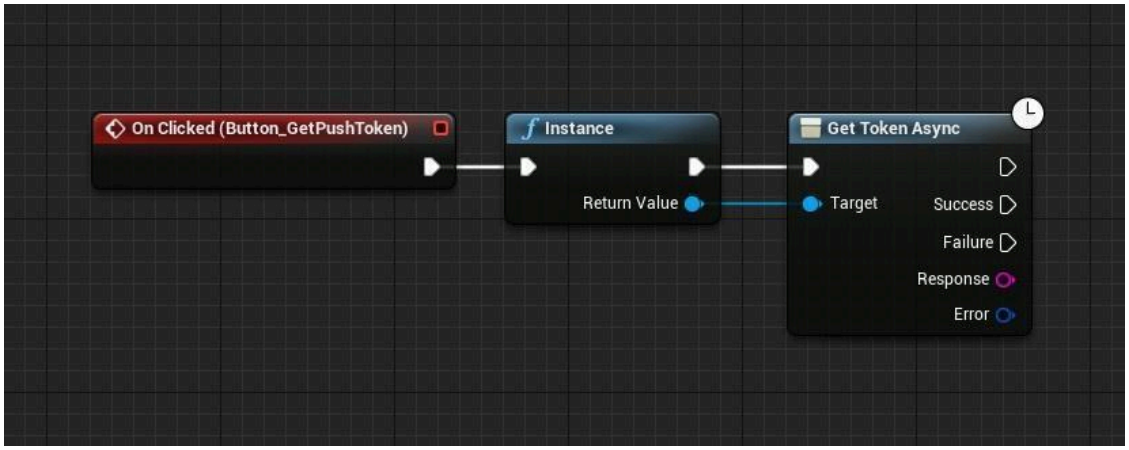
获取用户的推送令牌

在库初始化之后，您可以使用 URuStorePushClient::Instance()->GetToken() 方法来获取当前用户的push令牌。如果用户没有推送令牌，则该方法将创建并返回新的推送令牌。

RuStore实现示例**PushClient.GetToken**

long responseId = URuStorePushClient::Instance()->GetToken(``[]( long responseId, FString response) { ``// Process response ``}, ``[](long responseId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

Blueprint实现：



Success 回调以字符串形式在 Response 参数中返回令牌：

响应**GetProducts**

FString response

- response- 当前的 push-令牌。

Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = ""; ``description = ""; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name- 错误名称。
- description- 错误描述。

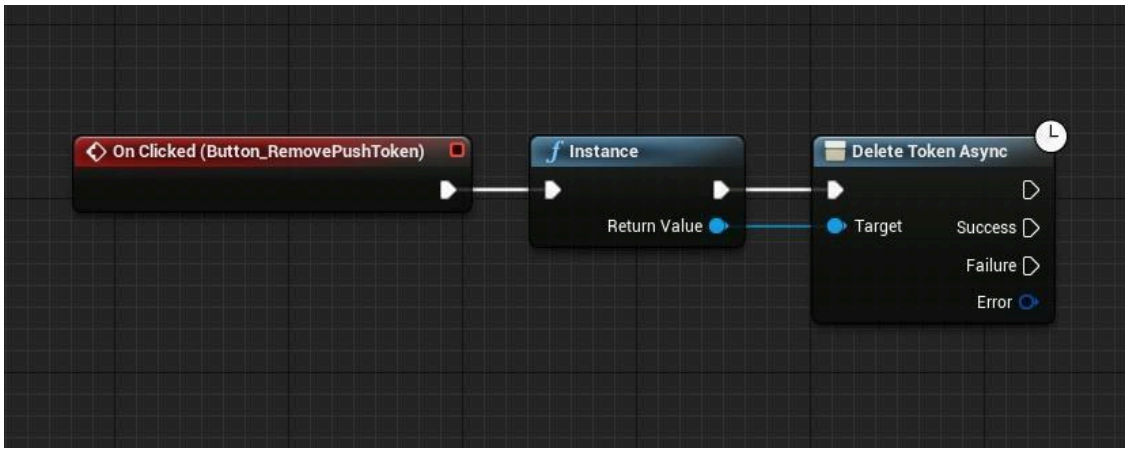
删除用户的推送令牌

在库初始化之后，您可以使用 RuStorePushClientt::Instance()->DeleteToken() 方法来删除当前用户的push令牌。

RuStore实现示例**PushClient.DeleteToken**

long responseId = URuStorePushClient::Instance()->DeleteToken(``[]( long responseId) { ``// Process success ``}, ``[](long responseId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

Blueprint实现：



Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name- 错误名称。
- description- 错误描述。

用于处理推送主题的方法

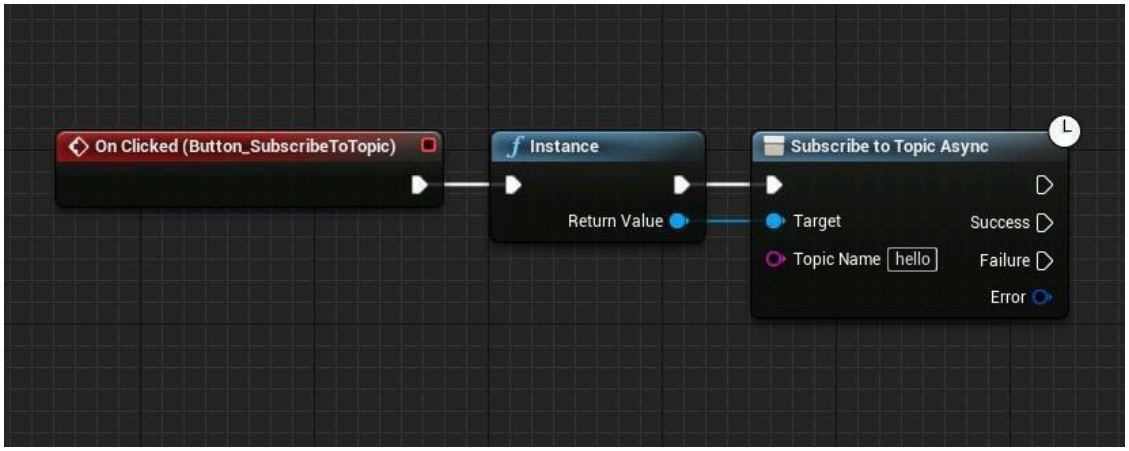
订阅主题的推送通知

在初始化库之后，您可以使用 URuStorePushClientt::Instance()->SubscribeToTopic() 方法来订阅主题。

RuStore实现示例**PushClient.GetToken**

long requestId = URuStorePushClient::Instance()->SubscribeToTopic(``topicName, ``[](long requestId) { ``// Process error ``}, ``[]( long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

Blueprint实现：



Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name- 错误名称。
- description- 错误描述。

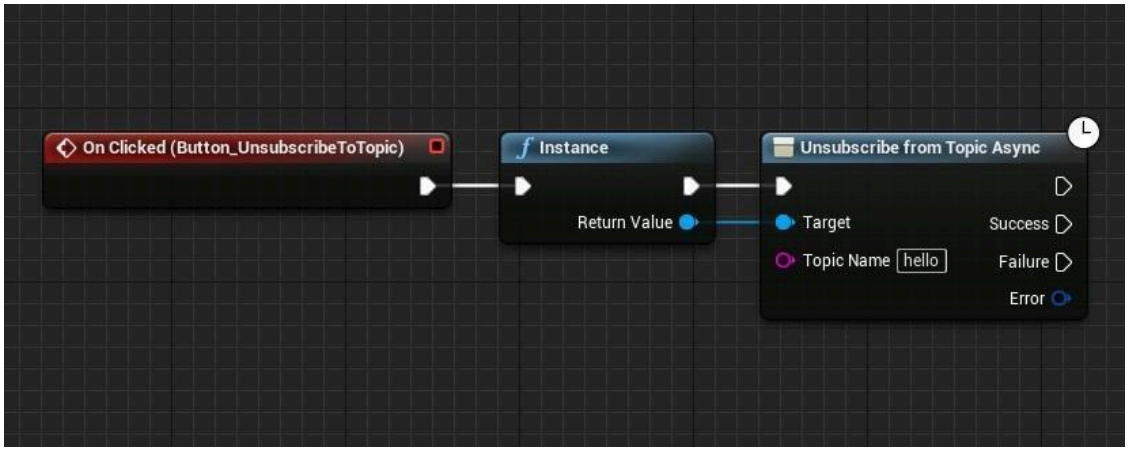
取消订阅主题的推送通知

在初始化库之后，您可以使用 RuStorePushClientt::Instance()->UnsubscribeToTopic() 方法来取消订阅主题。

RuStore实现示例**PushClient.DeleteToken**

long responseId = URuStorePushClient::Instance()->UnsubscribeFromTopic(``[]( long responseId) { ``// Process success ``}, ``[](long responseId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) { ``// Process error ``});

Blueprint实现：



Failure回调返回FURuStoreError结构体，包含错误信息，在Error参数中。所有可能的FURuStoreException错误在错误处理部分有描述。

错误结构体

USTRUCT(BlueprintType)struct RUSTORECORE_API FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{ ``name = "" ; ``description = "" ; ``} ``UPROPERTY(BlueprintReadOnly) ``FString name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};

- name- 错误名称。
- description- 错误描述。

创建发送通知的渠道

对于将要发送通知的渠道，适用以下优先级顺序：

1. 如果 push 通知中包含 channelId 字段，那么 SDK 将会把通知发送到这个渠道。同时，您的应用程序负责提前创建这个渠道。

2. 如果 push 通知中没有 channelId 字段，但您的应用程序在 AndroidManifest.xml 中指定了带有渠道的参数，则将使用 AndroidManifest.xml 中的渠道。您的应用程序负责创建这个渠道。

3. 如果 push 通知中没有 channelId 字段，并且在您的应用程序的 AndroidManifest.xml 中没有设置默认渠道，则 SDK 将自行创建一个渠道并发送通知到该渠道。以后，所有未指定渠道的通知都将发送到这个渠道。

点击通知时打开 Activity

默认情况下，当点击通知时，SDK 会打开带有 action “android.intent.action.MAIN”的 activity。如果 clickAction 字段存在，则 SDK 会打开符合带有指定 action 的 Intent filter 的 activity。

为了使 SDK 能够在点击通知时打开 activity（这也适用于默认的 activity），在应用程序的清单中相应的 activity 元素下需要添加字符串。如果没有这个字符串，SDK 将无法打开 activity。

错误处理

可能的错误：

1. RuStoreNotInstalledException – 用户设备上未安装 RuStore。
2. RuStoreOutdatedException – 用户设备上安装的 RuStore 不支持推送通知功能。
3. RuStoreUserUnauthorizedException – 用户未在 RuStore 中授权。
4. RuStoreFeatureUnavailableException – RuStore 没有在后台工作的权限。
5. RuStoreException – RuStore的基本错误，其他错误均继承自此。

如果在 SDK 初始化时传递了参数 allowNativeErrorHandling == true，在出现错误时，除了调用相应的 onFailure 处理程序外，该错误还会传递到原生 SDK 的 resolveForPush 方法中，以便向用户显示错误对话框：

resolveForPush

fun RuStoreException.resolveForPush(context: Context)

在初始化后，可以通过设置 AllowNativeErrorHandling 属性来更改此行为：

禁止原生的错误处理

URuStorePushClient::Instance()->SetAllowNativeErrorHandling(false);

Unity

概述

为了推送通知的运行，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持推送通知功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 中获得授权。
5. 应用程序的签名指纹必须与 RuStore 控制台中添加的指纹匹配。

在项目中集成

要集成，请下载RuStorePush SDK并将其导入项目（Assets → Import Package → Custom Package）。依赖项将通过SDK中包含的External Dependency Manager自动连接。

MinimumAPIlevel应设置为不低于24。应用程序的缩减（ProGuard/R8）目前不受支持，因此需要在项目设置中将其关闭（File → Build Settings → Player Settings → Publishing Settings → Minify）。

编辑应用程序清单

您必须声明 RuStoreUnityMessagingService 服务：

< service ``android:name = "ru.rustore.unitysdk.pushclient.RuStoreUnityMessagingService" ``android:exported = "true" ``tools:ignore = "ExportedService" > ``< intent-filter > ``< action android:name = "ru.rustore.sdk.pushclient.MESSAGING_EVENT" /> ``</ intent-filter ></ service >

如果您想更改标准通知的图标或颜色，可以添加以下元数据：

< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_icon" ``android:resource = "@drawable/ic_baseline_android_24" />< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_color" ``android:resource = "@color/your_favorite_color" />

您可以添加以下元数据以重写通知渠道：

< meta-data ``android:name = "ru.rustore.sdk.pushclient.default_notification_channel_id" ``android:value = "@string/pushes_notification_channel_id" />

当您添加自己的推送通知渠道时，您需要自行创建该渠道。

初始化

为了初始化，需要在项目中重写 Application 类。所需的源代码已包含在文件 Assets/RuStoreSDK/PushClient/Android/RuStoreUnityApplication.java 中：

```
package ru.rustore.unitysdk;import android.app.Application;import ru.rustore.unitysdk.pushclient.RuStoreUnityPushClient;public class RuStoreUnityApplication  extends Application {    ``@Override public void onCreate() {        ``super .onCreate();        ``RuStoreUnityPushClient.init(            ``application =  this        ``);    ``}}
```

- application—Application类的实例。

需要在项目的 AndroidManifest.xml 中指定这个类：

```
< application android:name = "ru.rustore.unitysdk.RuStoreUnityApplication" >
```

库将使用在Unity编辑器中设置的参数进行初始化。在编辑器菜单中选择Window → RuStoreSDK → Settings → Push Client:

- VKPNSProjectId—您在 RuStore 开发者控制台中项目的标识符；
- AllowAllow Native Error Handling — 允许原生 SDK 中的错误处理。错误处理

在从 C#### 代码中调用库方法之前，必须先调用它的初始化：

```
var config =  new RuStorePushClientConfig() {    ``allowNativeErrorHandling =  true ,    ``messagingServiceListener = pushServiceListener,    ``logListener = pushLogListener};RuStorePushClient.Instance.Init(config);
```

- allowallowNativeErrorHandling — 允许原生 SDK 中的错误处理；错误处理
- messagingServiceListener—实现 IMessagingServiceListener 接口的类对象；
- logListener—实现 ILogListener 接口的类对象。

检查获得推送通知的可能性

为了推送通知的运行，必须遵守几个条件：

- 用户的设备上必须安装 RuStore。
- RuStore 必须支持推送通知功能。
- RuStore 应用程序应允许在后台模式下运行。
- 用户必须在 RuStore 中获得授权。

为了检查上述条件，可以使用 RuStorePushClient.checkPushAvailability 方法。

```
RuStorePushClient.Instance.CheckPushAvailability(    ``onFailure: (error) => {        ``// Process error    ``},    ``onSuccess: (response) => {        ``if (!response.isAvailable) {            ``// Process push unavailable        ``}    ``});
```

使用推送令牌的方法

获取用户的推送令牌

在库初始化之后，您可以使用 RuStorePushClient.GetToken() 方法来获取当前用户的推送令牌。如果用户没有推送令牌，则该方法将创建并返回新的推送令牌。

```
RuStorePushClient.Instance.GetToken(  ``onFailure: (error) => {  ``// Process error  ``}, ``onSuccess: (token) => {  ``// Process success  ``});
```

删除用户的推送令牌

在库初始化之后，您可以使用 RuStorePushClient.DeleteToken() 方法来删除当前用户的推送令牌。

```
RuStorePushClient.Instance.DeleteToken(  ``onFailure: (error) => {  ``// Process error  ``}, ``onSuccess: () => {  ``// Process success  ``});
```

用于处理推送主题的方法

订阅主题的推送通知

在初始化库之后，您可以使用 SubscribeToTopic("your_topic_name") 方法来订阅主题。

```
RuStorePushClient.Instance.SubscribeToTopic(  ``topicName: "your_topic_name", ``onFailure: (error) => {  ``// Process error  ``}, ``onSuccess: () => {  ``// Process success  ``});
```

取消订阅主题的推送通知

在初始化库之后，您可以使用 UnsubscribeFromTopic("your_topic_name") 方法来取消订阅主题。

```
RuStorePushClient.Instance.UnsubscribeFromTopic(  ``topicName: "your_topic_name", ``onFailure: (error) => {  ``// Process error  ``}, ``onSuccess: () => {  ``// Process success  ``});
```

使用 *RuStore* 的操作*MessagingService*

```
public interface IMessagingServiceListener {  ``public void OnNewToken( string token);  ``public void OnMessageReceived(RemoteMessage message);  ``public void OnDeletedMessages();  ``public void OnError(List<RuStoreError> errors);}
```

OnNewToken—在接收到新的推送令牌时将被调用。

调用此方法后，您的应用程序负责将新的推送令牌传递到自己的服务器。

OnMessageReceived—在接收到新的推送通知时将被调用。

可以从message.data字段获取推送通知的payload(Dictionary<string, string>)。

OnDeletedMessages—如果一个或多个推送通知没有被送达到设备，将会调用此方法。

这可能是因为通知的生命周期在送达设备前已经结束。

在调用此方法时，建议与您的服务器同步，以免错过数据。

OnError—如果在初始化过程中发生错误，将会调用此方法。

可能的错误：

- UnauthorizedException—用户未在 RuStore 中授权。
- HostAppNotInstalledException—用户设备上未安装RuStore应用程序。
- HostAppBackgroundWorkPermissionNotGranted—RuStore 没有在后台工作的权限。

所有上述方法都将在后台线程上被调用。

通知结构体

完整通知的结构体

public class RemoteMessage { ``public string collapseKey; ``public Dictionary< string , string > data; ``public string messageId; ``public Notification notification; ``public int priority; ``public sbyte [] rawData; ``public int ttl;}

- messageId—消息的唯一ID。它是每个消息的标识符。
- priority—(当前不考虑)返回优先级值。目前有以下选项:
  - 0 — UNKNOWN.
  - 1 — HIGH.
  - 2 — NORMAL.
- ttl—Int类型的推送通知生命周期(以秒为单位)。
- collapseKey—(当前不考虑)通知组的标识符。
- data—一个字典, 在其中可以传递额外的通知数据。
- rawData—data 字典以字节数组形式。
- notification—通知对象。

public class Notification { ``public string title; ``public string body; ``public string channelId; ``public string imageUrl; ``public string color; ``public string icon; ``public string clickAction;}

- title—通知的标题;
- body—通知的正文;
- channelId—可以指定的渠道, 用于发送通知(适用于Android 8.0及更高版本);
- imageUrl—插入通知中的图片的直接链接(图片大小不超过1兆字节);
- color—通知的颜色(Notification.color)。颜色需要以十六进制格式传递, 为字符串格式(例如:"#A52A2A");
- icon—通知的图标。图标应存放在应用程序的资源中(res/drawable)。参数的值—一个字符串, 与资源名称相匹配;
  - 例如:在 res/drawable 中有一个名为small_icon.xml的图标, 在代码中通过 R.drawable.small_icon访问。为了在通知中显示这个图标, 服务器需要在 "icon" 参数中放入 "small_icon" 的值。
- clickAction—intent点击通知时, 用于打开活动的intent action。

创建发送通知的渠道

对于将要发送通知的渠道, 适用以下优先级顺序:

1. 如果 push 通知中包含 channelId 字段, 那么 SDK 将会把通知发送到这个渠道。同时, 您的应用程序负责提前创建这个渠道。

2. 如果 push 通知中没有 channelId 字段, 但您的应用程序在 AndroidManifest.xml 中指定了带有渠道的参数, 则将使用 AndroidManifest.xml 中的渠道。您的应用程序负责创建这个渠道。

3. 如果推送通知中没有 channelId 字段, 并且您的应用程序的 AndroidManifest.xml 中没有设置默认渠道, 则RuStoreSDK将自行创建一个渠道并向其发送通知。以后, 所有未指定渠道的通知都将发送到这个渠道。

点击通知时打开 Activity

默认情况下，当点击通知时，SDK 会打开带有 “android.intent.action.MAIN”的 action。如果存在 clickAction 字段，RuStoreSDK将打开符合指定action的 Intent filter 的活动。

为了使 SDK 能够在点击通知时打开活动（这也适用于默认的活动），在应用程序的清单中相应的活动元素下需要添加字符串。如果没有这个字符串，SDK 将无法打开活动。

错误处理

可能的错误：

1. RuStoreNotInstalledException — 用户设备上未安装 RuStore。
2. RuStoreOutdatedException — 用户设备上安装的 RuStore 不支持推送通知功能。
3. RuStoreUserUnauthorizedException — 用户未在 RuStore 中授权。
4. RuStoreFeatureUnavailableException — RuStore 没有在后台工作的权限。
5. RuStoreException — RuStore的基本错误，其他错误均继承自此。

如果在 SDK 初始化时传递了参数 allowNativeErrorHandling == true，在出现错误时，除了调用相应的 onFailure 处理程序外，该错误还会传递到原生 SDK 的 resolveForPush 方法中，以便向用户显示错误对话框：

```
fun RuStoreException.resolveForPush(context: Context)
```

在初始化后，可以通过设置 AllowNativeErrorHandling 属性来更改此行为：

```
RuStorePushClient.Instance.AllowNativeErrorHandling =  false ;
```

Defold

概述

实现示例

为了了解如何正确集成用于处理 push-通知的包，建议您参阅示例应用程序。

必要条件

为了推送通知的运行，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持推送通知功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 中获得授权。
5. 应用程序的签名指纹必须与 RuStore 控制台系统中添加的指纹匹配。

在项目中集成

要将包连接到项目中，需要将其添加到依赖项中：

https://gitflic.ru/project/rustore/defold-extension-rustore-push/file/downloadAll?branch=master

更好的做法是指定特定的发布版本：
https://gitflic.ru/project/rustore/defold-extension-rustore-push/release/

初始化

为了初始化 push-通知服务，您需要在项目的 game_project 中添加一个值。使用任何方便的文本编辑器打开它，在 [android] 部分添加

[android]rustore_project_id = %your project id%package = %your package%

在这里，**%your project id%** —项目的标识符，**%your package%** — Android 包。在开发者控制台，该字段被称为 “项目 ID” 和 “Android Package Name”，位于 “Push-通知” -> “项目” 页面。

默认设置的推送消息标题和正文

如果 data 消息中没有，可以为推送通知的标题和正文添加默认值。

[android]push_field_title = default push titlepush_field_text = default push body

Android 设备上的推送通知服务会自动启动，当调用ruStorePush.set_on_token()

处理 *push-token* 和消息的方法

获取用户的推送令牌

为获取推送令牌，需要在 ruStorePush.set_on_token() 中添加 callback 到 listener。
listenerruStorePush.set_on_token()

如果用户没有推送令牌，该方法将返回一个新的推送令牌。

local function new_token(self, token, error) ``if token
then ``print(token) ``else ``print(error.error) ``endendlocal function
push_android() ``ruStorePush.set_on_token(new_token)end

令牌更改事件

旧令牌可能会定期失效。可以重新发放令牌。这些事件在 _ 中返回
callbackruStorePush.set_on_token()

处理 push 消息

要从 push 通知中获取信息，需要添加回调函数
ruStorePush.set_on_message()ruStorePush.set_on_message()

local function listener(self, payload, activated) ``-- The payload arrives
here. ``pprint(payload)endlocal function push_android() ``ruStorePush.set_on_message(listener)end

监听器结构体

- payload推送通知的 data 字段 (lua table)
- activated用户是否通过点击推送进入应用程序 (bool)

完整示例

local function listener(self, payload, activated) ``-- The payload arrives here. ``pprint(payload)endlocal function new_token(self, token, error) ``if token then ``print(token) ``else ``print(error.error) ``endendlocal function push_android() ``ruStorePush.set_on_token(new_token) ``ruStorePush.set_on_message(listener) ``print("Rustore pushes registered")endfunction init(self) ``local sysinfo = sys.get_sys_info() ``if sysinfo.system_name == "Android" then ``push_android() ``else ``print( "Notifications work only Android" ) ``end ``msg.post("." , "acquire_input_focus")end

重要提示

不要发送包含 notification 和 android.notification.title 的 push 消息，这些将由 RuStore 处理。并且它们不会正确工作，不能通过这些消息进入应用程序。

发送 data push 时包括以下字段

- **title**—用于 push 消息的标题；
- **message**—用于 push 消息的正文。

Flutter

概述

实现示例

请参阅示例应用程序，了解如何正确集成用于处理 push 通知的包：
https://gitflic.ru/project/rustore/flutter-rustore-push

推送通知的运行条件

为了推送通知的运行，必须遵守以下条件：

1. 用户的设备上必须安装 RuStore 应用程序。
2. RuStore 应用程序必须支持推送通知的功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 应用程序中获得授权。
5. 应用程序的签名指纹必须与 RuStore 控制台中添加的指纹匹配。

在项目中集成

要将包集成到项目，请执行以下命令：

flutter pub add flutter_rustore_push

这个命令将在 pubspec.yaml 文件中添加一个字符串。

dependencies: ``flutter_rustore_push: ^1.0.0

初始化

为了初始化推送通知服务，请在您的 android 项目的 values 中添加值：

<resources> ``<string name= "flutter_rustore_push_project" translatable= "false" >xxx</string></resources>

xxx—这是项目标识符。在 RuStore 控制台系统中，该字段被称为“项目 ID”，位于“推送通知 -> 项目”部分。

要启动推送通知服务，需要添加一个从 FlutterRustoreApplication 继承的 Application 类。

在 Kotlin 中这样做的例子：

package ru.rustore.flutter_rustore_push_exampleimport ru.rustore.flutter_rustore_push.FlutterRustoreApplicationopen class Application: FlutterRustoreApplication() {}

在 AndroidManifest.xml 中需要指定这个类：

<application ``android:label= "flutter_rustore_push_example"  ``android:name= ".Application" ``android:icon= "@mipmap/ic_launcher" >  ``// ... ``</application>

设置ProGuard

要配置 ProGuard，请添加以下规则：

-keep public class com.vk.push.** extends android.os.Parcelable

在 android/app/build.gradle 文件中添加：

buildTypes { ``release {  ``// ... ``proguardFiles getDefaultProguardFile( 'proguard-android.txt' ), 'proguard-rules.pro'  ``} ``// ...}

检查获得推送通知的可能性

为了推送通知的运行，必须遵守几个条件：

1. 用户的设备上必须安装 RuStore。
2. RuStore 必须支持推送通知功能。
3. RuStore 应用程序应允许在后台模式下运行。
4. 用户必须在 RuStore 中获得授权。

要检查上述条件，可以使用 RustorePushClient.available() 方法：

RustorePushClient.available().then((value) { ``print( "available success: ${value}" );}, onError: (err) {  ``print("available error: ${err}");});

用于处理推送令牌和推送通知的方法

获取用户的推送令牌

在初始化库之后，您可以使用 RustorePushClient.getToken() 方法来获取用户当前的 push 令牌。

如果用户没有推送令牌，则该方法将创建并返回新的推送令牌。

```
RustorePushClient.getToken().then((value) {    ``print( "get token success: ${value}" );}, onError: (err) {    ``print( "get token error: ${err}" );})
```

删除用户的推送令牌

您可以使用 RustorePushClient.deleteToken() 方法来删除用户当前的 push 令牌。

```
RustorePushClient.deleteToken().then(() {    ``print( "delete success:" );}, onError: (err) {    ``print( "delete error: ${err}" );})
```

令牌更改事件

旧令牌可能会定期失效。可以重新发放令牌。为了了解是否生成了新的令牌，需要使用回调函数 RustorePushClient.onNewToken()。

```
RustorePushClient.onNewToken((value) {    ``print( "on new token success: ${value}" );}, error: (err) {    ``print( "on new token err: ${err}" );});
```

处理推送通知

要从推送通知中获取信息，需要添加回调函数 RustorePushClient.onMessageReceived()。

```
RustorePushClient.onMessageReceived((value) {    ``print( "on message received success: id=${value.messageId}, data=${value.data}, notification.body: ${value.notification?.body}" );}, error: (err) {    ``print( "on message received error: ${err}" );});
```

删除推送通知

要删除推送通知，需要添加回调函数 RustorePushClient.onDeletedMessages()。

```
RustorePushClient.onDeletedMessages(() {    ``print( "deleted messages" );}, error: (err) {    ``print( "on message received error: ${err}" );});
```

错误处理

要处理错误，需要使用回调函数 RustorePushClient.onError()。

```
RustorePushClient.onError((err) {    ``print( "on error: ${err}" );});
```

通知结构体

通知结构体

```
class Message { ``String? messageId; ``int priority; ``int ttl; ``String? collapseKey; ``Map<String?, String?> data; ``Notification? notification;}
```

- messageId—消息的唯一ID。它是每个消息的标识符；
- priority—返回优先级值。目前有以下选项：
  - 0 — UNKNOWN.

    - 1 — HIGH.
    - 2 — NORMAL.
- ttl—Int类型的推送通知生命周期（以秒为单位）；
- collapseKey—通知组的标识符；
- data—一个字典，在其中可以传递额外的通知数据；
- notification—通知对象。

结构体Notification

class Notification { ``String? title; ``String? body; ``String? channelId; ``String? imageUrl; ``String? color; ``String? icon; ``String? clickAction;}

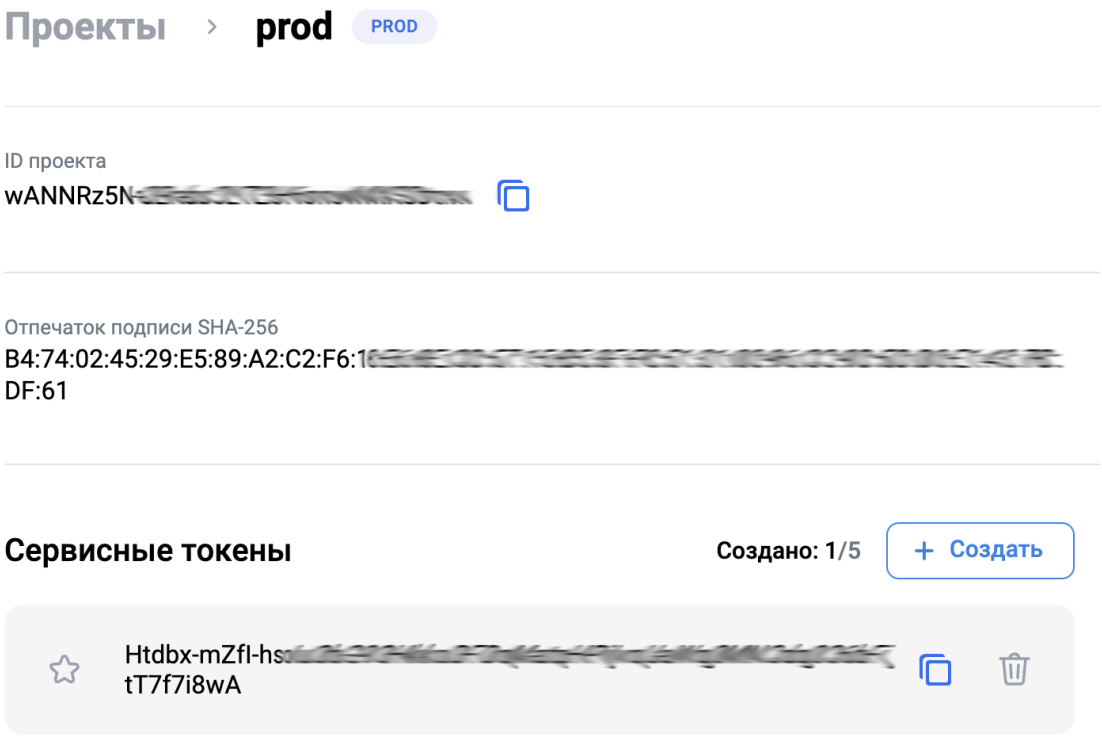
- title—通知的标题；
- body—通知的正文；
- channelId—可以指定的渠道，用于发送通知（适用于Android 8.0及更高版本）；
- imageUrl—插入通知中的图片的直接链接（图片大小不超过1兆字节）；
- color—通知的颜色（Notification.color）。颜色需要以十六进制格式传递，为字符串格式（例如：“#A52A2A”）；
- icon—通知的图标。图标应存放在应用程序的资源中（res/drawable）。参数的值是一个字符串，与资源的名称匹配：
    - 在res/drawable中有一个名为small_icon.xml的图标，在代码中通过R.drawable.small_icon访问。为了在通知中显示此图标，服务器需要在“icon”参数中设置值“small_icon”。
- clickAction—intent点击通知时，用于打开活动的intent action。

发送推送通知

关键词：RuStore，开发者，文档，RuStoreSDK，推送通知，推送通知发送。API 旨在提供 Firebase 的 drop-in replacement 功能。

要发送推送通知，请使用 POST 方法 https://vkpns.rustore.ru/v1/projects/$project_id/messages:send。

指定“项目 ID”和“服务令牌”来发送推送通知。您可以在 RuStore 控制台系统中获取这些值。为此，请在应用程序页面上转到“推送通知”部分并选择“项目”。



服务令牌需要在“Authorization: Bearer {service-token}”标题中指定。

请求正文

参数	类型	描述
validate_only	bool	在不发送推送通知的情况下进行请求验证
message	object (message)	推送通知的结构体

message

参数	类型	描述
token	string	在应用程序中获得的用户推送令牌
data	map	包含“key”: value 对的对象value
notification	object (message.notification)	适用于所有平台的基本通知模板
android	object (message.android)	Android 消息的特殊参数

message.notification

参数	类型	描述
title	string	通知标题
body	string	通知的主要文本
image	string	包含将在通知中显示的图片的 URL 地址

message.android

参数	类型	描述
ttl	string (duration format)	消息在存储中保留的时间（以秒为单位）。例如：“3.5s”
notification	object (message.android.notification)	用于发送到 Android 设备的通知Android

message.android.notification

参数	类型	描述
title	string	通知标题
body	string	通知的主要文本
icon	string	通知图标
color	string	通知图标的颜色，格式为 #rrggbb
image	string	包含将在通知中显示的图片的 URL 地址
channel_id	string	通知渠道的标识符
click_action	string	用户点击通知时关联的动作

在 message 结构体中，目前只支持上述字段。

成功响应的正文

参数	类型	描述
—	—	在成功响应的情况下，将返回一个带有空 payload 的消息payload

错误响应的正文

参数	类型	描述
error	object (error)	错误

error

参数	类型	描述
code	int	数字错误代码 (404, 400, 403, 401, …)
message	string	错误的详细描述
status	string	文本格式的错误代码(INVALID_ARGUMENT, UNREGISTERED, …)

HTTPstatus对应于字段 code。

发送消息时可能的错误：

- INVALID_ARGUMENT—当发送消息时，请求参数指定不正确。
- INTERNAL—服务的内部错误。
- TOO_MANY_REQUESTS—发送消息的尝试次数超过限制。
- PERMISSION_DENIED—服务密钥指定不正确。

- NOT_FOUND—用户的推送令牌指定不正确。

验证算法**Message**

1. 如果message.data的payload不为空(内部至少有一个键值对），那么该消息是有效的。message.notification和message.android部分可以不包含。
2. 如果message.data字段不存在，则必须有notification。在这种情况下，需要检查message.notification或message.android.notification字段的存在。这些字段中至少有一个必须存在，但两个都可以存在(如果两者都存在，某些字段将被重写)。

限制

1. 如果推送中没有ttl字段或其值为0，则考虑默认值为4周。如果推送中缺少message.android部分，则会添加该部分并包含ttl字段。
2. 消息的最大容量为4096字节。

推送通知发送示例

关键词:RuStore，开发者，文档，RuStoreSDK，推送通知，发送推送通知，示例，无效的S2S-令牌，过期的推送令牌。成功请求的示例

POST https: //vkpns.rustore.ru/v1/projects/myproject-b5ae1/messages:send HTTP/2Content-Type: application/jsonAuthorization: Bearer $ss_token{ ``"message" :{ ``"token" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1..." , ``"notification" :{ ``"body" : "This is a notification message!" , ``"title" : "Message" , ``"image" : "https://image-hosting.org/284239234.jpeg" ``} ``}}

响应

HTTP/ 2 200{}

请求示例，包括无效的推送令牌

POST https: //vkpns.rustore.ru/v1/projects/U95076bdd5KDJ3LjYkNp91o05Y6LkfQk/messages:send HTTP/2Content-Type: application/jsonAuthorization: Bearer Fw9FgDx9FQtya6k-7UkSOnzpHYhDq0SQY4-8QKJ6wKZI9OUPiCCYyNmS-CV2-ZQ5{ ``"message" :{ ``"token" : "bad-push-token" , ``"notification" : { ``"body" : "This is a notification message!" , ``"title" : "Message" , ``"image" : "https://image-hosting.org/284239234.jpeg" ``} ``}}

响应

HTTP/ 2 400{ ``"error" : { ``"code" : 400 , ``"message" : "The registration token is not a valid FCM registration token" , ``"status" : "INVALID_ARGUMENT" ``}}

带有已过期有效推送令牌的请求示例

POST https: //vkpns.rustore.ru/v1/projects/U95076bdd5KDJ3LjYkNp91o05Y6LkfQk/messages:send HTTP/2Content-Type: application/jsonAuthorization: Bearer Fw9FgDx9FQtya6k-7UkSOnzpHYhDq0SQY4-8QKJ6wKZI9OUPiCCYyNmS-CV2-ZQ5{ ``"message" :{ ``"token" : "eH4tgqKEfFKqH6cMJ2WLttVibgQO9hfn" , ``"notification" : { ``"body" : "This is a notification message!" , ``"title" : "Message" , ``"image" : "https://image-hosting.org/284239234.jpeg" ``} ``}}

响应

HTTP/ 2 404{ ``"error" : { ``"code" : 404 , ``"message" : "Requested entity was not found." , ``"status" : "NOT_FOUND" ``}}

通过主题发送推送通知

要处理主题，需要以下工件：

- **project_id**project_id（ID 项目），**ss_token**（服务令牌）— 这些值可以在开发者控制台中获取。为此，请转到“Push-通知”部分，在应用程序页面上打开“项目”选项卡。详见推送通知发送部分。
- **push_token—**用于订阅主题和取消订阅的设备推送令牌。
- **topic—**项目中的主题。

向主题发送推送通知

需要包含Authorization标题：**Authorization: Bearer $ss_token**。

请求正文

参数	类型	描述
data	map	包含“ key”: value 对的对象value
notification	object (message.notification)	适用于所有平台的基本通知模板
android	object (message.android)	Android 消息的特殊参数

成功响应的正文

参数	类型	描述
message	text	推送成功发送的消息

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

订阅主题的推送通知

需要包含Authorization标题：**Authorization: Bearer $ss_token**。

请求正文

参数	类型	描述
push_tokens	array (string)	需要订阅主题的推送令牌

成功响应的正文

参数	类型	描述
message	text	订阅成功的消息或存在错误
errors	objectarray(令牌的错误体)	令牌的错误

令牌的错误体

参数	类型	描述
push_token	text	推送令牌
error	object（错误响应的正文）	错误

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

取消订阅主题的推送通知

需要包含Authorization标题：**Authorization: Bearer $ss_token**。

请求正文

参数	类型	描述
push_tokens	string array	需要从主题中取消订阅的推送令牌

成功响应的正文

参数	类型	描述
message	text	订阅成功的消息或存在错误
errors	objectarray(令牌的错误体)	令牌的错误

令牌的错误体

参数	类型	描述
push_token	text	推送令牌
error	object（错误响应的正文）	错误

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

通过主题发送推送通知的示例

订阅主题**hello**

POST 'https://vkpns-topics.rustore.ru/v1/projects/{project_id}/topics/{topic}/batch_subscribe' HTTP /2Content-Type: application /jsonAuthorization: Bearer $ss_token{ ``"push_tokens" : [ ``"kHrfTow3AjqdgUeCJLwbIHd7c0....." , ``"9mbJVS0-2e0cCcIx7X5nxMIgp8....." ``]}

成功的响应

HTTP/ 2 200{ ``"message" : "success"}

部分成功的响

HTTP/ 2 200{ ``"message" : "1 error(s)" , ``"errors" : [``{ ``"push_token" : "9mbJVS0-2e0cCcIx7X5nxMIgp8....." , ``"error" : { ``"code" : 1001 , ``"status" : "FORBIDDEN" , ``"message" : "s2s token invalid" ``} ``} ``]}

取消订阅主题**hello**

POST 'https://vkpns-topics.rustore.ru/v1/projects/{project_id}/topics/{topic}/batch_unsubscribe' HTTP /2Content-Type: application /jsonAuthorization: Bearer $ss_token{ ``"push_tokens" : [ ``"kHrfTow3AjqdgUeCJLwbIHd7c0....." , ``"9mbJVS0-2e0cCcIx7X5nxMIgp8....." ``]}

成功的响应

HTTP/ 2 200{ ``"message" : "success"}

部分成功的响

HTTP/ 2 200{ ``"message" : "1 error(s)" , ``"errors" : [``{ ``"push_token" : "9mbJVS0-2e0cCcIx7X5nxMIgp8....." , ``"error" : { ``"code" : 1001 , ``"status" : "FORBIDDEN" , ``"message" : "s2s token invalid" ``} ``} ``]}

向主题**hello**发送消息**hello**

POST 'https://vkpns-topics.rustore.ru/v1/projects/Lpvk29a2Al2P6tWu-a-aN3PKwO6-qvXZ/topics/hello/publish' HTTP /2Content-Type: application /jsonAuthorization: Bearer $ss_token{ ``"message" : { ``"notification" :{ ``"body" : "This is a notification message!" , ``"title" : "Message" , ``"image" : "https://image-hosting.org/284239234.jpeg" ``} ``}}

成功的响应

HTTP/ 2 200{ ``"message" : "payload has been successfully published to topic hello"}

错误示例**BAD_REQUEST**

HTTP/ 2 400{ ``"code" : 2000 , ``"status" : "BAD_REQUEST" , ``"message" : "failed to parse request: invalid character 's' after object key:value pair"}

错误示例**FORBIDDEN**

HTTP/ 2 403{ ``"code" : 1001 , ``"status" : "FORBIDDEN" , ``"message" : "s2s token invalid"}

向用户群组发送推送通知

常常需要向特定用户群组发送推送。这可以通过RuStore和MyTracker的推送来实现。

需要在应用程序中集成Push和MyTracker的SDK。文档中详细描述了如何做到这一点。

使用这些SDK，您可以向用户细分群体进行分发。

细分群体—根据您设定的参数选择的用户群。例如，带来最大收入的用户，或注册了旧版本Android的用户。

作为MyTracker分析工具的"细分群体"，提供了与产品受众工作的所有可能性：

- 允许评估单独用户组的大小和动态；
- 为定向广告选择目标受众；
- 将属于细分群体的设备和用户标识符传递给第三方分析系统或合作伙伴；
- 构建任何复杂程度的报告，针对一个或多个细分群体，而不会用不必要的数据过载统计信息。

*API*推送通知用于细分群体

概述

要处理细分群体，需要以下工件：

- **project_id**project_id（项目ID），**ss_token**（服务令牌）。这些值可以在开发者控制台获得。为此，需要进入应用页面的「推送通知」>「项目」部分。发送推送通知
- **push_token**.用于订阅和取消订阅主题分发的设备的推送令牌。
- **mt_api_user_id**。来自myTracker的***API User ID***。
- **mt_secret_key**。来自myTracker的***API***的秘密密钥。
- **mt_segment_id**.来自myTracker的细分群体标识符。
- **mt_app_id**.应用程序标识符。
- **export_project_id**export_project_id **(uuid)**。带有导出设置的项目标识符。
- **export_segment_id**export_segment_id **(uuid)**。用于导出的细分群体标识符。

创建带有导出设置的项目

POSThttps://vkpns-segments.rustore.ru/v1/export_settings/project/create

需要标题Authorization：**Authorization**：**Bearer $ss_tokenss_token**

请求正文

参数	类型	描述
参数	类型	描述
—	—	—
project	text	**project_id**
api_user_id	text	**mt_api_user_id**
secret_key	text	**mt_secret_key**
app_id	int	**mt_app_id**

成功响应的正文

参数	类型	描述
export_project_id	text	**export_project_id**

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

获取项目的导出设置

POST https://vkpns-segments.rustore.ru/v1/export_settings/project/get

需要标题Authorization：**Authorization**：**Bearer $ss_tokenss_token**

请求正文

参数	类型	描述
id	text	**export_project_id**

成功响应的正文

参数	类型	描述
id	text	**export_project_id**
project	text	**project_id**
app_id	int	**mt_app_id**

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

为导出创建细分群体

POSThttps://segments-vkpns.rustore.ru/v1/export_settings/project//segments/create

需要标题Authorization：**Authorization**：**Bearer $ss_tokenss_token**

请求正文

参数	类型	描述
period	int	按小时的数据导出频率
segment	text	**mt_segment_id**

成功响应的正文

参数	类型	描述
export_segment_id	text	**export_segment_id**

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

获取用于导出的细分群体数据

POSThttps://segments-vkpns.rustore.ru/v1/export_settings/project//segments/get

需要标题Authorization：**Authorization**：**Bearer $ss_tokenss_token**

成功响应的正文

参数	类型	描述
segments	array	带有细分群体的数组

Segment

参数	类型	描述
id	text	**export_segment_id**
segment	text	**mt_segment_id**
period	int	按小时的数据导出频率
is_enabled	boolean	标志，表示是否启用了这个细分群体

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

将推送通知发送到细分群体

POST https://segments-vkpns.rustore.ru/v1/projects//segments//publish

需要标题Authorization：**Authorization**：**Bearer $ss_tokenss_token**

请求正文

参数	类型	描述
message	object(message)	消息

成功响应的正文

参数	类型	描述
message	text	推送成功发送的消息

错误响应的正文

参数	类型	描述
code	int	错误代码
status	text	状态
message	text	详细错误

示例

创建带有导出设置的项目

```
curl --location  'https://vkpns-segments.rustore.ru/v1/export_settings/project/create' \--header 'Content-Type: application/json' \--header  'Accept: application/json' \--header  'Authorization: Bearer <ss_token>' \--data '{ ``"project" :  "<project_id>" , ``"api_user_id" :  "<mt_api_user_id>" , ``"secret_key" :  "<mt_secret_key>" , ``"app_id" : <mt_app_id>}'
```

成功的响应

```
HTTP/ 2 200{   ``"project_id" :  "b04b48ab-3125-444f-94eb-aad511c074e7"}
```

无效的**s2s-**令牌

```
HTTP/ 2 400{   ``"code" :  2000 ,   ``"status" :  "BAD_REQUEST" ,   ``"message" :  "Invalid S2S token"}
```

获取项目的导出设置

```
curl --location  'https://segments-vkpns.rustore.ru/v1/export_settings/project/get' \--header 'Content-Type: application/json' \--header  'Accept: application/json' \--header  'Authorization: Bearer <ss_token>' \--data '{ ``"id" :  "<export_project_id>"}'
```

成功的响应

```
HTTP/ 2 200{   ``"id" :  "<export_project_id>" ,   ``"app_id" : <mt_app_id>,   ``"project" :  "<project_id>"}
```

为导出创建细分群体

```
curl --location
'https://segments-vkpns.rustore.ru/v1/export_settings/project/<export_project_id>/segments/create
```

' \--header 'Content-Type: application/json' \--header 'Accept: application/json' \--header 'Authorization: Bearer <ss_token>' \--data '{ ``"period" : 24,  ``"segment" : "<mt_segment_id>"}'

成功的响应

HTTP/ 2 200{ ``"segment_id" : "<export_segment_id>"}

获取用于导出的细分群体数据

curl --location --request POST 'https://segments-vkpns.rustore.ru/v1/export_settings/project/<export_project_id>/segments/get' \--header 'Accept: application/json' \--header 'Authorization: Bearer <ss_token>'

成功的响应

HTTP/ 2 200{ ``"segments" : [  ``{ ``"id" :  "<export_segment_id>" ,  ``"segment" : "<mt_segment_id>" , ``"period" :  24 ,  ``"is_enabled" : true ``}  ``]}

向细分群体发送消息

curl --location 'https://segments-vkpns.rustore.ru/v1/projects/<project_id>/segments/<mt_segment_id>/publish' \--header 'Content-Type: application/json' \--header 'Accept: application/json' \ ``--header 'Authorization: Bearer <ss_token>' \--data '{  ``"message" : { ``"notification" :{  ``"body" : "This is a notification message!" , ``"title" : "Message" ,  ``"image" : "https://image-hosting.org/284239234.jpeg" ``}  ``}}'

成功的响应

HTTP/ 2 200{ ``"message" : "payload has been successfully published to segment <mt_segment_id>"}

在*MyTracker*中使用细分群体*MyTracker*

集成MyTracker到应用程序

为了按细分群体发送推送通知，您需要将MyTracker的SDK和推送通知的SDK集成到您的应用程序中

创建细分群体

RuStore的推送通知提供了向不同用户细分群体发送推送的可能性。为此，需要与MyTracker集成MyTracker

- 在应用程序中配置MyTracker的SDK
- 在MyTracker接口中配置细分群体的收集MyTracker

细分群体—根据您设定的参数选择的用户群。例如，带来最大收入的用户，或注册了旧版本Android的用户。

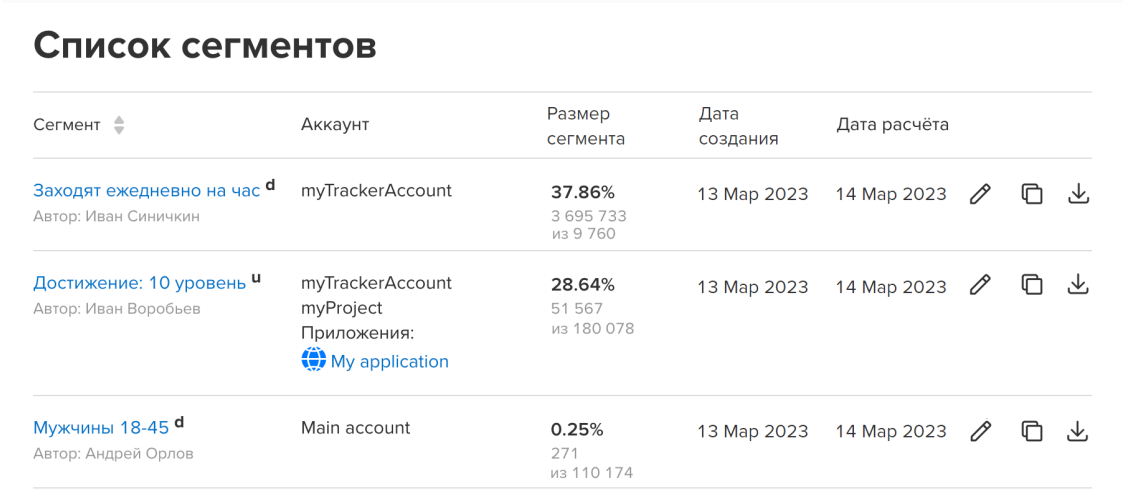
作为MyTracker分析工具的"细分群体"，提供了与产品受众工作的所有可能性：

- 允许评估单独用户组的大小和动态；
- 为定向广告选择目标受众；

- 将属于细分群体的设备和用户标识符传递给第三方分析系统或合作伙伴；
- 构建任何复杂程度的报告，针对一个或多个细分群体，而不会用不必要的数据过载统计信息。

有关分段的详细信息，请参阅 MyTracker 文档
https://docs.tracker.my.com/reports/segments

MyTracker根据您设置的参数计算细分群体的大小。计算每日进行，因此在细分群体页面上可以轻松追踪变化动态。关键数据如细分群体大小、最后计算日期和相关应用程序显示在“细分群体列表”页面上。



在“细分群体列表”页面点击“添加”。

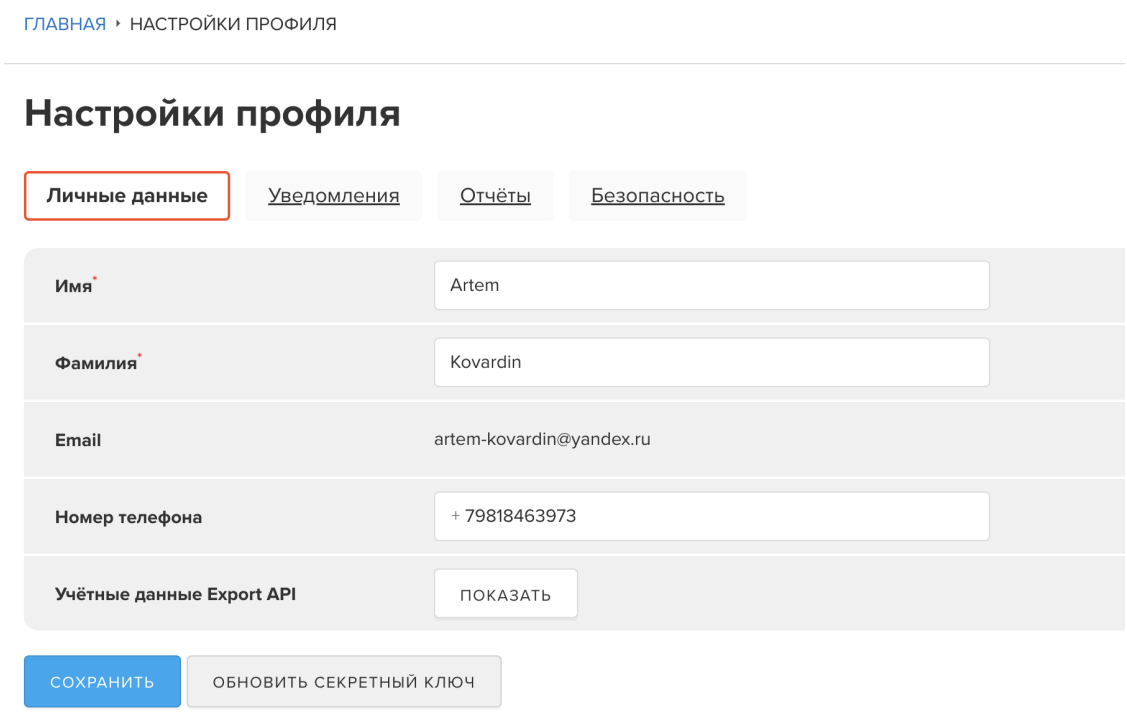
您可以以已经存在的细分群体为基础。为此，请打开细分群体页面并点击“复制”。

填写表单：

- 名称* — MyTracker的列表和报告中显示的细分群体名称。
- 受众类型* — 构建细分群体的受众类型：按实际设备或按用户账户。
- 账户* — 将要添加细分群体的账户。如果您只有一个账户，它将自动被选中。更多信息请参阅账户部分。
- 项目 — 一个或多个项目，其应用将程序用于形成细分群体。更多信息请参阅项目部分。
- 应用程序 — 一个或多个应用程序，其受众将用于形成细分群体。

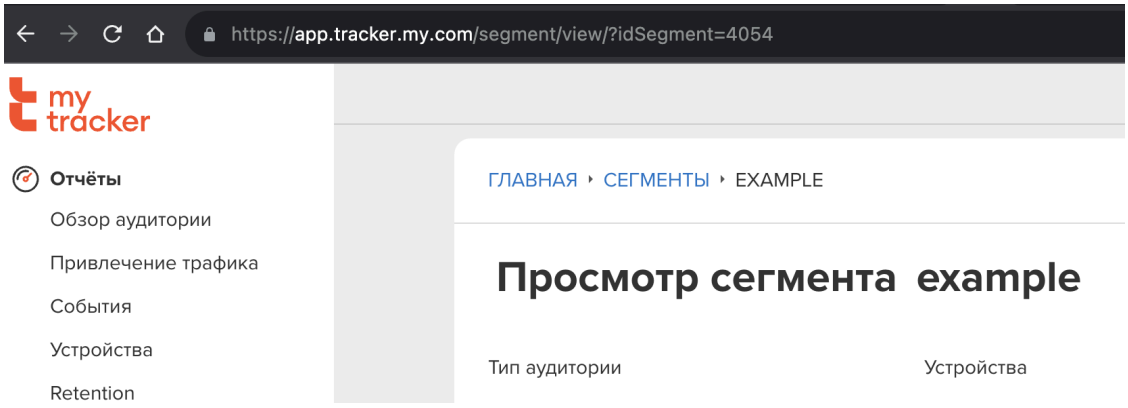
集成MyTracker APIAPIMyTracker

要处理细分群体，必须集成MyTracker API。需要在用户个人资料页面获取令牌。



点击“显示”按钮。复制令牌，并在为处理细分群体的推送通知API创建设置时，在secret_key字段中指定它。

在创建项目之后，您可以将新的细分群体添加到导出中。为此，您需要细分群体的ID，您将在请求中指定它。这个ID可以在跳转到细分群体页面后从url中获取。



从url中复制idSegment并在请求中使用它。

curl --location 'https://segments-vkpns.rustore.ru/v1/export_settings/project/<export_project_id>/segments/create' \--header 'Content-Type: application/json' \--header 'Accept: application/json' \--header 'Authorization: Bearer <ss_token>' \--data '{ ``"period" : 24 , ``"segment" : "<mt_segment_id>"}'

export_project_id—ID在第一步中获取的项目ID。

mt_segment_id—ID从MyTracker获取的细分群体ID。

ss_token—用于授权的服务令牌，从推送项目页面的开发者控制台获取。

RuStore SDK用于通用推送通知

Kotlin

概述

RuStore SDK用于通用推送通知—一套用于处理推送通知的包。通过多个渠道发送和接收消息：

- FCM—FirebaseCloudMessaging;
- HMS—HuaweiMobileServices;
- RuStore.

将SDK与已配置的HMS和FCM服务一起使用。

实现示例

查看示例应用程序，了解如何正确集成通用推送通知。

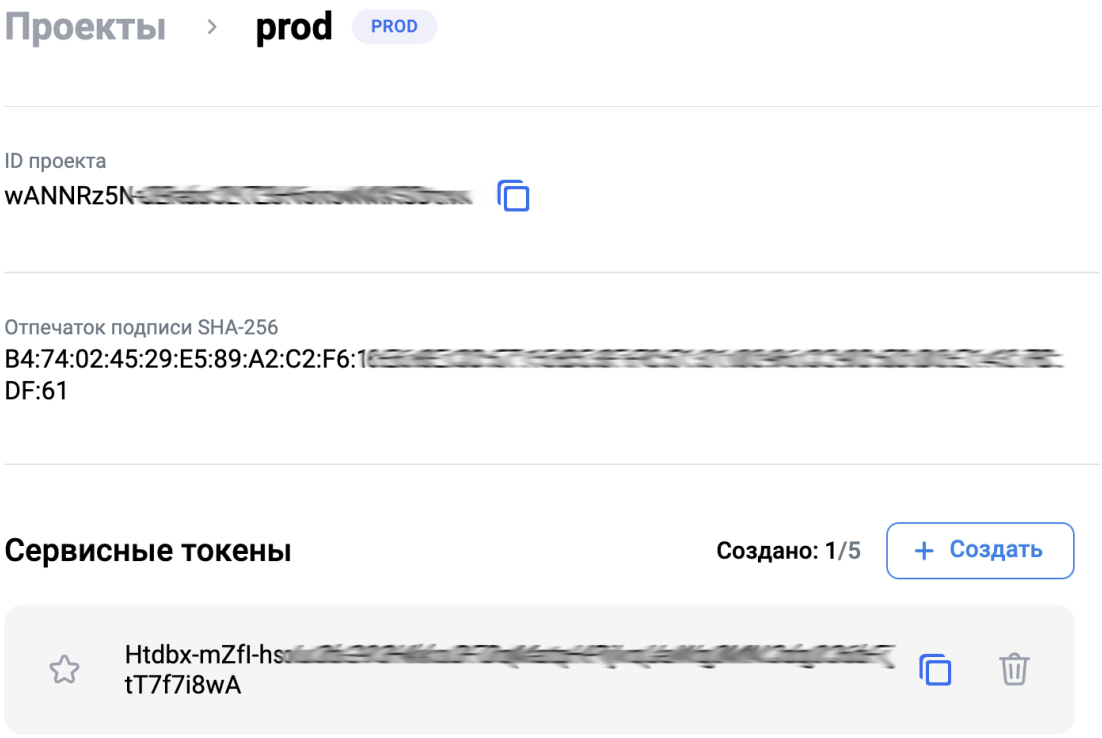
推送通知的运行条件

为了通用推送通知运作，需要遵守以下条件：

- 用户的设备上必须安装 RuStore 应用程序。
- RuStore 应用程序必须支持推送通知的功能。
- RuStore 应用程序应允许在后台模式下运行。
- 用户必须在 RuStore 应用程序中获得授权。
- 应用程序的签名指纹必须与开发者控制台中添加的指纹匹配。

应用程序设置

要进行初始化，需要在“RuStore 控制台”获取项目ID。在应用程序页面，进入“推送通知”部分并选择“项目”。



在项目中集成

在settings.gradle文件中集成存储库：

```
dependencyResolutionManagement
{    ``repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)    ``repositories
{      ``google()      ``mavenCentral()      ``maven {url = uri( "https://developer.huawei.com/repo/"
)}      ``maven {          ``url = uri( "https://artifactory-external.vkpartner.ru/artifactory/maven"
)      ``}    ``}}
```

集成依赖项

要集成依赖项，请在您的app级别的build.gradle文件中添加以下代码：

```
dependencies {    ``implementation( 'ru.rustore.sdk:universalpush:0.1.2' )    ``implementation(
'ru.rustore.sdk:universalrustore:0.1.2' )    ``implementation( 'ru.rustore.sdk:universalhms:0.1.2'
)    ``implementation( 'ru.rustore.sdk:universalfcm:0.1.2' )}
```

当您在应用程序中一起使用RuStore Push SDK和ProGuard时，请添加以下规则：

```
-keep public class com.vk.push.** extends android.os.Parcelable
```

这仅对ru.rustore.sdk:universalrustore包的运作是必需的。

使用FCM和HMS需：

- 在app级别的 build.gradle 中添加app

plugins{ ``// ... ``// 用于FCM所需 ``id 'com.google.gms.google-services' ``//, 用于HMS所需 ``id 'com.huawei.agconnect'}

- 在根级别的build.gradle中添加

dependencies { ``// 用于FCM所需``classpath 'com.google.gms:google-services:4.3.15' ``// , 用于HMS所需``classpath 'com.huawei.agconnect:agcp:1.6.0.300' ``classpath 'com.android.tools.build:gradle:7.4.0'}

- 在根级别的settings.gradle中添加

pluginManagement { ``repositories { ``google() ``mavenCentral() ``gradlePluginPortal() ``// 用于HMS运作所需``maven {url = uri("https://developer.huawei.com/repo/")} ``}}

Google Play

应用程序设置

要在Google Play上发布应用程序，您只需添加通过FCM和RuStore进行推送的依赖项。要开始使用FCM，请在Firebase服务中配置您的项目。

1. 在Firebase Console创建一个新项目。
2. 在Firebase Console选择您计划集成推送通知的项目。
3. 在左侧菜单，项目名称旁边，点击齿轮图标并进入Project Settings部分。
4. 转到Your apps部分并下载google-services.json文件。
5. 将google-services.json文件放置在app/google-services.json。

在项目中集成

要在Goole Play上发布应用程序，您只需使用universalfcm、universalrustore和universalpush包。

在根级别的settings.gradle中集成存储库：

dependencyResolutionManagement{ ``repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS) ``repositories { ``google() ``mavenCentral() ``// 用于rustore运作所需``maven { ``url = uri( "https://artifactory-external.vkpartner.ru/artifactory/maven" ) ``} ``}}

集成依赖项

要集成依赖项，请在您的build.gradle级别的app文件中添加以下代码：

dependencies { ``implementation( 'ru.rustore.sdk:universalpush:0.1.1' ) ``implementation('ru.rustore.sdk:universalrustore:0.1.0') ``implementation('ru.rustore.sdk:universalfcm:0.1.0')}

在您的应用程序中使用SDK时，如果一起使用ProGuard，请添加以下规则：

-keep public class com.vk.push.** extends android.os.Parcelable

要使用FCM，需要：

- 在app级别的 build.gradle 中添加app

plugins { ``// ... ``// 用于fcm所需 ``id 'com.google.gms.google-services'}

- 在根级别的build.gradle中添加

buildscript { ``dependencies { ``// 用于fcm所需 ``classpath 'com.google.gms:google-services:4.3.15' ``}}

Huawei Mobile Services

应用程序设置

要在AppGallery中发布应用程序，只需添加依赖项以通过HMS和RuStore进行推送运作。为了开始与HMS运作，您需要在developer.huawei.com服务中配置您的项目。

1. 在developer.huawei.com创建一个新项目。
2. 在AppGallery Connect中选择您计划集成推送通知的项目。
3. 转到项目设置页面，然后转到基本信息选项卡。
4. 转到应用程序数据部分并下载agconnect-services.json文件。
5. 将agconnect-services.json文件放在app/agconnect-services.json中。

在项目中集成

要在AppGallery中发布应用程序，您只需使用universalhms、universalrustore和universalpush包。

在settings.gradle文件中集成仓库：

```
dependencyResolutionManagement
{ ``repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS) ``repositories
{ ``google() ``mavenCentral() ``maven {url = uri( "https://developer.huawei.com/repo/"
)} ``maven { ``url = uri( "https://artifactory-external.vkpartner.ru/artifactory/maven"
) ``} ``}}
```

集成依赖项

要集成依赖项，请在您的build.gradle级别的app文件中添加以下代码：

```
dependencies { ``implementation( 'ru.rustore.sdk:universalpush:0.1.1' ) ``implementation(
'ru.rustore.sdk:universalrustore:0.1.0' ) ``implementation( 'ru.rustore.sdk:universalhms:0.1.0' )}
```

在您的应用程序中使用SDK时，如果一起使用ProGuard，请添加以下规则：

```
-keep public class com.vk.push.** extends android.os.Parcelable
```

要使用HMS，需要：

- 在app级别的 build.gradle 中添加app

```
plugins { ``// ... ``// 用于hms所需 ``id 'com.huawei.agconnect'}
```

- 在根级别的build.gradle中添加

```
dependencies { ``// 用于hms所需``classpath 'com.huawei.agconnect:agcp:1.6.0.300' ``classpath
'com.android.tools.build:gradle:7.4.0'}
```

- 在根级别的settings.gradle中添加

```
pluginManagement { ``repositories { ``google() ``mavenCentral() ``gradlePluginPortal()
``// 用于HMS运作所需``maven {url = uri( "https://developer.huawei.com/repo/" )} ``}}
```

初始化

要进行初始化，需要在App.kt文件中写入以下代码：

```
import android.app.Applicationimport
ru.rustore.sdk.universalpush.RuStoreUniversalPushClientimport
ru.rustore.sdk.universalpush.firebase.provides.FirebasePushProviderimport
ru.rustore.sdk.universalpush.hms.providers.HmsPushProviderimport
ru.rustore.sdk.universalpush.rustore.logger.DefaultLoggerimport
ru.rustore.sdk.universalpush.rustore.providers.RuStorePushProviderclass App: Application()
{  ``private val tag =  "UniversalPushExampleApp"  ``override fun onCreate()
{  ``super.onCreate()  ``RuStoreUniversalPushClient.init(  ``context = this,  ``rustore =
RuStorePushProvider(  ``application = this,  ``projectId =  "m3Id6aPeXq36mp..."
,  ``logger = DefaultLogger(tag = tag),  ``),  ``firebase =
FirebasePushProvider(  ``context = this,  ``),  ``hms =
HmsPushProvider(  ``context = this,  ``appid =  "108003365" ,  ``),  ``)  ``}}
```

如果您只使用几个推送通知提供商，那么您只需初始化所使用的提供商。

```
RuStoreUniversalPushClient.init(  ``context = this,  ``rustore = RuStorePushProvider(  ``application
= this,  ``projectId =  "m3Id6aPeXq36mp..." ,  ``logger = DefaultLogger(tag = tag),  ``),  ``hms =
HmsPushProvider(  ``context = this,  ``appid =  "108003365" ,  ``),)
```

检查推送通知的工作能力

要检查推送通知提供商的可用性，需要调用方法checkAvailability (context)。

```
RuStoreUniversalPushClient.checkAvailability(this)  ``.addOnCompleteListener(object :
OnCompleteListener<Map<String, Boolean>> {  ``override fun onSuccess(result: Map<String,
Boolean>) {  ``Log.w(tag,  "get availability success ${result}" )  ``}  ``override fun
onFailure(throwable: Throwable) {  ``Log.e(tag,  "get tokens err: ${throwable}"
)  ``}  ``})
```

result—这是一个带有键的字典：

```
public const val UNIVERSAL_FCM_PROVIDER: String =  "firebase"public const val
UNIVERSAL_HMS_PROVIDER: String =  "hms"public const val UNIVERSAL_RUSTORE_PROVIDER: String
=  "rustore"
```

您可以这样检查特定推送通知提供商的可用性：

```
if (result[UNIVERSAL_HMS_PROVIDER] ?:  false ) {  ``// hms提供商可用
```

处理令牌

获取令牌

为了获取所有提供商的令牌列表，需要调用getTokens()方法。

```
RuStoreUniversalPushClient.getTokens()  ``.addOnCompleteListener(object :
OnCompleteListener<Map<String, Boolean>> {  ``override fun onSuccess(result: Map<String,
Boolean>) {  ``Log.w(tag,  "get availability success ${result}" )  ``}  ``override fun
```

onFailure(throwable: Throwable) { ``Log.e(tag, "get tokens err: ${throwable}" ) ``} ``})

result—这是一个带有键的字典：

public const val UNIVERSAL_FCM_PROVIDER: String = "firebase"public const val UNIVERSAL_HMS_PROVIDER: String = "hms"public const val UNIVERSAL_RUSTORE_PROVIDER: String = "rustore"

要获取特定令牌，请使用以下代码：

result[UNIVERSAL_FCM_PROVIDER].orEmpty()

删除令牌

要删除令牌，需要调用deleteTokens(token)方法并传递含有令牌列表的字典。

RuStoreUniversalPushClient.deleteTokens(``mapOf( ``UNIVERSAL_RUSTORE_PROVIDER to "xxx" , ``UNIVERSAL_FCM_PROVIDER to "yyy" , ``UNIVERSAL_HMS_PROVIDER to "zzz" ``))

其中**xxx**, **yyy**, **zzz** — 是来自不同推送通知提供商的令牌。

处理主题

订阅主题

要订阅主题，需要调用subscribeToTopic（“topic”）方法。

RuStoreUniversalPushClient.subscribeToTopic("some_topic")

取消订阅主题

要取消订阅主题，需要调用unsubscribeToTopic（“topic”）方法。

RuStoreUniversalPushClient.unsubscribeFromTopic("some_topic")

事件处理

事件处理onDeletedMessages

为了处理onDeletedMessages事件，您需要在初始化RuStoreUniversalPushClient后在App类中添加OnDeletedMessagesListener回调。在onDeletedMessages事件发生时，将调用带有providerType参数的回调，该参数可帮助您了解触发事件的推送提供商。

RuStoreUniversalPushClient.setOnDeletedMessagesListener { providerType -> ``// process event}

事件处理onNewToken

为了处理onNewToken事件，您需要在初始化RuStoreUniversalPushClient后在App类中添加OnNewTokenListener回调。在onNewToken事件发生时，将调用具有以下参数的回调：

- providerType—用于识别触发事件的推送提供商。
- token—新的推送令牌。

RuStoreUniversalPushClient.setOnNewTokenListener { providerType, token -> ``// process event}

处理通知

为了接收通知，您需要在初始化RuStoreUniversalPushClient之后在App类中添加OnMessageReceiveListener回调。如果通知是通过universal api发送的，则它们将在客户端进行去重，通知接收回调将被调用1次。

如果notification对象中有数据，RuStoreSDK将自动显示通知。如果您不希望RuStoreSDK自动显示通知，应使用data对象，并保留notification对象为空。但是，无论如何都会调用OnMessageReceiveListener回调。可以从remoteMessage.data字段获取推送通知的payload（Map<String, String>）。

RuStoreUniversalPushClient.setOnMessageReceiveListener { remoteMessage -> ``// process message}

处理推送提供商的错误

为了处理错误，您需要在初始化RuStoreUniversalPushClient后在App类中添加OnPushClientErrorListener回调。出现错误时，将调用带有参数的回调：

- providerType—用于识别触发事件的推送提供商。
- errors—错误列表。

RuStoreUniversalPushClient.setOnPushClientErrorListener { providerType, error -> ``// process error}

如果已经在使用*HMS/FCMHMS/FCM*

如果您在应用程序中使用FCM/HMS服务，请在这些服务中添加额外代码。

FCM

在FCM服务中添加以下代码：

import ru.rustore.sdk.universalpush.firebase.messaging.toNotificationPayloadclass MyFirebaseMessagingService: FirebaseMessagingService() { ``override fun onMessageReceived(message: RemoteMessage) { ``super .onMessageReceived(message) ``RuStoreUniversalPushManager.processMessage(message.toNotificationPayload()) ``} ``override fun onNewToken(token: String) { ``super .onNewToken(token) ``RuStoreUniversalPushManager.processToken(token) ``}}

HMS

在HMS服务中添加以下代码：

import ru.rustore.sdk.universalpush.hms.messaging.toNotificationPayloadclass MyMessagePushService: HmsMessageService() { ``override fun onMessageReceived(msg: RemoteMessage?) { ``super .onMessageReceived(msg) ``RuStoreUniversalPushManager.processMessage(msg.toNotificationPayload()) ``} ``override fun onNewToken(token: String?) { ``super .onNewToken(token) ``RuStoreUniversalPushManager.processToken(token) ``}}

变更记录

SDK版本1.0.0

- 添加了推送通知的去重
- 添加了以下回调：
  - 处理通知处理通知
  - 事件处理onDeletedMessages
  - 事件处理onNewToken
  - 处理推送提供商的错误

发送通用推送通知

此API旨在提供通过多个推送提供商同时发送push通知的能力。

要发送push通知，需要使用POST方法 https://vkpns-universal.rustore.ru/v1/send。

要发送push通知，请为计划发送push的每个提供商指定“项目ID”和“授权令牌”。

RuStore

使用**project_id**（项目ID）和**ss_token**（服务令牌）。您可以在RuStore控制台系统中获取这些值。为此，请在应用程序页面转到“Push通知”部分并选择“项目”。

作为“授权令牌”，请使用“服务令牌”。

FCM

您可以在RuStore控制台系统中获取这些值。获取令牌的示例在FCM文档中有描述。

- https://firebase.google.com/docs/cloud-messaging/migrate-v1#use-credentials-to-mint-access-tokens
- https://firebase.google.com/docs/cloud-messaging/auth-server

RuStore不存储“项目ID”和“授权令牌”。请自行验证令牌。

HMS

您可以在RuStore控制台系统中获取这些值。获取“授权令牌”的示例在文档中有说明。“项目ID”可以在开发者控制台中获取，不要与“应用程序ID”混淆，文档中有相关描述。

RuStore不存储“项目ID”和“授权令牌”。请自行验证令牌。

请求正文

参数	类型	必须的	描述
providers	object (providers)	是	用于发送push通知的提供商
tokens	object (tokens)	是	按提供商分类的Push令牌
message	object (message)	是	Push通知的结构

providers

参数	类型	必须的	描述

rustore	object (provider)	否	RuStore 提供商
fcm	object (provider)	否	提供商Firebase
hms	object (provider)	否	提供商Huawei

providers.provider

参数	类型	必须的	描述
project_id	string	是	ID项目
auth_token	string	是	授权令牌

tokens

参数	类型	必须的	描述
rustore	array (string)	否	RuStore Push-令牌
fcm	array (string)	否	Push-令牌Firebase
hms	array (string)	否	Push-令牌Huawei

message

参数	类型	必须的	描述
data	map	否	包含“ key”: value 对的对象value
notification	object (message.notification)	是	适用于所有平台的基本通知模板
android	object (message.android)	*否	Android 消息的特殊参数

*在 hms 中是必须的

message.notification

参数	类型	必须的	描述
title	string	是	通知标题
body	string	是	通知的主要文本
image	string	否	包含将在通知中显示的图片的 URL 地址

message.android

参数	类型	必须的	描述
ttl	string (duration format)	否	消息在存储中保留的时间（以秒为单位）。例如：“3.5s”
notification	object (message.android.notification)	*否	用于发送到 Android 设备的通知 Android

*在 hms 中是必须的

message.android.notification

参数	类型	必须的	描述
title	string	是	通知标题
body	string	是	通知的主要文本

icon	string	否	通知图标
color	string	否	通知图标的颜色，格式为 #rrggbb
image	string	否	包含将在通知中显示的图片的 URL 地址
channel_id	string	否	通知渠道的标识符
*click_action	string	否	用户点击通知时关联的动作

*在 hms 中默认为类型 1 (intent)。

在 message 结构体中，仅支持上述字段。

成功响应的正文

参数	类型	描述
status	string	在成功响应的情况下返回带有“OK”状态的消息

错误正文

参数	类型	描述
code	int	数字错误代码 (404, 400, 403, 401, …)
status	string	错误的一般描述
errors	array (string)	按提供商分类的错误或验证错误

HTTPstatus对应于字段 code。

错误分为三种类型

- 消息验证错误。
- 按提供商发送的错误。
- 服务错误。

发送消息时可能出现的错误

- VALIDATION_ERROR—当发送消息时，请求参数指定不正确。
- PROVIDER_ERROR—向推送提供商发送时出错。
- INTERNAL_ERROR—服务的内部错误。

发送消息时可能的验证错误

- providers.%provider_name% —提供商中的错误。
- tokens.%provider_name%—提供商令牌中的错误。
- message.%field% —消息中的错误。

发送消息时可能的提供商错误，格式为 **%provider_name%**:**%error%**

- internal—提供商的内部错误。
- validationerror—提供商端的验证错误。

- invalidauthtoken—auth 令牌错误。
- toomanyrequests—请求过多。
- invalidtokens—push 令牌错误，将通过逗号显示切割的令牌列表（前六个字符）。

验证算法

提供商必须有令牌。

Providers

1. 必须至少有一个提供商，没有提供商请求是无效的。

Tokens

1. 每个提供商至少必须有一个push-令牌。

Message

1. 如果payload的**message.data**不为空（内部至少有一个键值对），那么该消息是有效的。**message.notification**和**message.android**部分可以不包含。
2. 如果**message**. **data**字段不存在，则必须有**notification**。在这种情况下，将检查**message.notification** 或 **message.android.notification** 字段的存在。这些字段中至少有一个必须存在，但两个都可以存在（如果两者都存在，某些字段将被覆盖）。

限制

- 如果推送中没有**ttl**字段或其值为0，则考虑默认值为4周。如果推送中缺少**message.android**部分，则会添加该部分并包含**ttl**字段。
- 消息的最大容量为**4096**字节。

通用推送通知发送示例

成功请求的示例

POST https: //vkpns-universal.rustore.ru/v1/send HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``"auth_token" : "321CCbbAAA" , ``"project_id" : "ccbbaa" ``} ``}, ``"tokens" : { ``"rustore" :["bk3RNwTe3H"], ``"fcm" :[ "CI2k_HHwgIpoDKC" ] ``}, ``"message" :{ ``"notification" :{ ``"body" : "This is a notification message!" , ``"title" : "Message" ``} ``}}HTTP/2 200Content-Type: application/json{ ``"Status" : "OK"}

无效提供商的示例

POST https: //vkpns-universal.rustore.ru/v1/send HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``} ``}, ``"tokens" : { ``"rustore" :["bk3RNwTe3H"], ``"fcm" :[ "CI2k_HHwgIpoDKC" ] ``}, ``"message" :{ ``"notification" :{ ``"body" : "This is a notification message!" , ``"title" : "Message" ``} ``}}HTTP/2 400Content-Type: application/json { ``"status" : "VALIDATION_ERROR" , ``"code" : 400, ``"errors" : [ ``"providers.fcm: project_id is a required field" , ``"providers.fcm: auth_token is a required field" ``]}

无效消息的示例

POST https: //vkpns-universal.rustore.ru/v1/send HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``"auth_token" : "321CCbbAAA" , ``"project_id" : "ccbbaa" ``} ``}, ``"tokens" : { ``"rustore" :["bk3RNwTe3H"], ``"fcm" :[ "CI2k_HHwgIpoDKC" ] ``}, ``"message" :{ ``"notification" :{ ``} ``}}HTTP/2 400Content-Type: application/json { ``"status" : "VALIDATION_ERROR" , ``"code" : 400, ``"errors" : [ ``"message.notification: title is a required field" , ``"message.notification: body is a required field" ``]}

提供商错误的示例

POST https: //vkpns-universal.rustore.ru/v1/send HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``"auth_token" : "321CCbbAAA" , ``"project_id" : "ccbbaa" ``} ``}, ``"tokens" : { ``"rustore" :["bk3RNwTe3H"], ``"fcm" :[ "CI2k_HHwgIpoDKC" ] ``}, ``"message" :{ ``"data" :{ "test" : "test" } ``} ``}}HTTP/2 400Content-Type: application/json { ``"status" : "PROVIDER_ERROR" , ``"code" : 400, ``"errors" : [ ``"fcm: invalid auth token" ``]}

通过主题发送通用推送通知

API此API旨在提供通过多个推送提供商同时通过主题发送通知的能力。

要发送推送通知，需要使用POST方法 https://vkpns-universal.rustore.ru/v1/send/topic 。

要向主题发送推送通知，请为计划发送推送的每个提供商指定“项目ID”和“授权令牌”。

RuStore

使用**project_id**(项目ID)和**ss_token**(服务令牌)。您可以在RuStore控制台系统中获取这些值。为此，请在应用程序页面转到“Push通知”部分并选择“项目”。发送推送通知

FCM

您可以在RuStore控制台系统中获取这些值。获取令牌的示例在FCM文档中有描述。

- https://firebase.google.com/docs/cloud-messaging/migrate-v1#use-credentials-to-mint-access-tokens
- https://firebase.google.com/docs/cloud-messaging/auth-server

RuStore不存储“项目ID”和“授权令牌”。请自行验证令牌。

HMS

您可以在RuStore控制台系统中获取这些值。获取“授权令牌”的示例在文档中有说明。“项目ID”可以在开发者控制台中获取，不要与“应用程序ID”混淆，文档中有相关描述。

RuStore不存储“项目ID”和“授权令牌”。请自行验证令牌。

请求正文

参数	类型	必须的	描述
providers	object (providers)	是	用于发送push通知的提供商
topic	string	是	主题名称
message	object (message)	是	Push通知的结构

providers

参数	类型	必须的	描述
rustore	object (provider)	否	RuStore 提供商
fcm	object (provider)	否	提供商Firebase
hms	object (provider)	否	提供商Huawei

providers.provider

参数	类型	必须的	描述
project_id	string	是	ID项目
auth_token	string	是	授权令牌

message

参数	类型	必须的	描述
data	map	否	包含“ key”: value 对的对象value
notification	object (message.notification)	是	适用于所有平台的基本通知模板
android	object (message.android)	*否	Android 消息的特殊参数

*在 hms 中是必须的

message.notification

参数	类型	必须的	描述
title	string	是	通知标题
body	string	是	通知的主要文本
image	string	否	包含将在通知中显示的图片的 URL 地址

message.android

参数	类型	必须的	描述
ttl	string (duration format)	否	消息在存储中保留的时间（以秒为单位）。例如：“3.5s”
notification	object (message.android.notification)	*否	用于发送到 Android 设备的通知 Android

*在 hms 中是必须的

message.android.notification

参数	类型	必须的	描述
title	string	是	通知标题
body	string	是	通知的主要文本

icon	string	否	通知图标
color	string	否	通知图标的颜色，格式为 #rrggbb
image	string	否	包含将在通知中显示的图片的 URL 地址
channel_id	string	否	通知渠道的标识符
*click_action	string	否	用户点击通知时关联的动作

*在 hms 中默认为类型 1 (intent)。

在 message 结构体中，仅支持上述字段。

成功响应的正文

参数	类型	描述
status	string	在成功响应的情况下返回带有“OK”状态的消息

错误正文

参数	类型	描述
code	int	数字错误代码 (404, 400, 403, 401, …)
status	string	错误的一般描述
errors	array (string)	按提供商分类的错误或验证错误

HTTPstatus对应于字段 code。

错误分为三种类型

- 消息验证错误。
- 按提供商发送的错误。
- 服务错误。

发送消息时可能出现的错误

- VALIDATION_ERROR—当发送消息时，请求参数指定不正确。
- PROVIDER_ERROR—向推送提供商发送时出错。
- INTERNAL_ERROR—服务的内部错误。

发送消息时可能的验证错误

- providers.%provider_name% —提供商中的错误。
- message.%field% —消息中的错误。

发送消息时可能的提供商错误，格式为 **%provider_name%**:**%error%**

- internal—提供商的内部错误。
- validationerror—提供商端的验证错误。
- invalidauthtoken—auth 令牌错误。
- toomanyrequests—请求过多。

验证算法

提供商必须有令牌。

Providers

1. 必须至少有一个提供商，没有提供商请求是无效的。

Message

1. 如果payload的**message.data**不为空（内部至少有一个键值对），那么该消息是有效的。**message.notification**和**message.android**部分可以不包含。
2. 如果**message. data**字段不存在，则必须有**notification**。在这种情况下，将检查**message.notification** 或 **message.android.notification** 字段的存在。这些字段中至少有一个必须存在，但两个都可以存在（如果两者都存在，某些字段将被覆盖）。

限制

- 如果推送中没有**ttl**字段或其值为0，则考虑默认值为4周。如果推送中缺少**message.android**部分，则会添加该部分并包含**ttl**字段。
- 消息的最大容量为**4096**字节。

通用推送通知通过主题发送的示例

成功请求的示例

POST https: //vkpns-universal.rustore.ru/v1/send/topic HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``"auth_token" : "321CCbbAAA" , ``"project_id" : "ccbbaa" ``} ``}, ``"topic" : "test" , ``"message" :{ ``"notification" :{ ``"body" : "This is a notification message!" , ``"title" : "Message" ``} ``}}HTTP/2 200Content-Type: application/json{ ``"Status" : "OK"}

无效提供商的示例

POST https: //vkpns-universal.rustore.ru/v1/send/topic HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``} ``}, ``"topic" : "test" , ``"message" :{ ``"notification" :{ ``"body" : "This is a notification message!" , ``"title" : "Message" ``} ``}}HTTP/2 400Content-Type: application/json { ``"status" : "VALIDATION_ERROR" , ``"code" : 400, ``"errors" : [``"providers.fcm: project_id is a required field" , ``"providers.fcm: auth_token is a required field" ``]}

无效消息的示例

POST https: //vkpns-universal.rustore.ru/v1/send/topic HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``"auth_token" : "321CCbbAAA" , ``"project_id" : "ccbbaa" ``} ``}, ``"topic" : "test" , ``"message" :{ ``"notification" :{ ``} ``}}HTTP/2 400Content-Type: application/json { ``"status" : "VALIDATION_ERROR" , ``"code" : 400, ``"errors" : [``"message.notification: title is a required field" , ``"message.notification: body is a required field" ``]}

提供商错误的示例

POST https: //vkpns-universal.rustore.ru/v1/send/topic HTTP/2Content-Type: application/json{ ``"providers" :{ ``"rustore" : { ``"auth_token" : "AAAbbbCC123" , ``"project_id" : "aabbcc" ``}, ``"fcm" : { ``"auth_token" : "321CCbbAAA" , ``"project_id" : "ccbbaa" ``} ``}, ``"topic" : "test" , ``"message" :{ ``"data" :{ "test" : "test" } ``} ``}}HTTP/2 400Content-Type: application/json { ``"status" : "PROVIDER_ERROR" , ``"code" : 400, ``"errors" : [``"fcm: invalid auth token" ``]}

RuStore SDK 用于集成评论和评分

Kotlin

概述

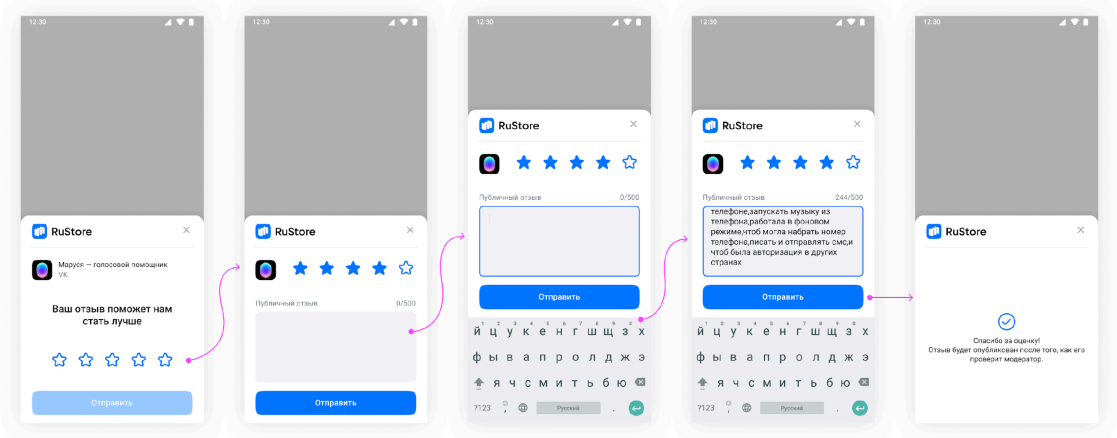
RuStore In-app Review SDK 允许用户在不离开应用的情况下在 RuStore 中对您的应用进行评分和评论。

用户可以在应用的整个使用过程中的任何时候启动评分和评论的用户场景。用户可以按照从 1 到 5 的等级对您的应用进行评分，并留下评论。评论是可选的。

实现示例

了解示例应用程序，以了解如何正确集成评分和评论 SDK。

用户场景示例



SDK 正确运作的条件

为了评分和评论应用程序 SDK 的正常运作，需要满足以下条件：

1. Android操作系统版本需为6.0或更高。
2. 用户的设备上安装了RuStore应用程序。
3. 用户的设备上的 RuStoreApp 版本是最新的。
4. 用户在RuStore应用程序中已授权。
5. 应用程序必须在 RuStore 上发布。

何时请求评分和评论

遵循以下建议来决定何时向用户请求应用程序的评分和评论：

- 在用户充分使用您的应用程序之后启动该过程。
- 不要过于频繁地启动该过程 — 这会降低用户对您应用程序的体验并限制评分 SDK 的使用。
- 不要使用行动号召，例如“评价应用程序”的按钮 — 用户可能已经达到了启动流程的限制。
- 您的应用程序在启动或进行流程期间不应询问用户任何问题，包括关于他们的意见（例如，“您喜欢这个应用程序吗？”）或预测性问题（例如，“您会给这个应用程序评5星吗？”）。

设计建议

遵循以下建议来决定如何集成流程：

- 保持流程原样显示，不对当前设计进行任何干预或更改，包括大小、不透明度、形状和其他属性。
- 不要在流程的顶部或边缘添加任何内容。
- 流程应在所有层上方打开。启动流程后不要关闭它。流程将在用户明确操作后自行结束。

在项目中集成

连接存储库：

```
repositories {  ``maven {  ``url = uri( "https://artifactory-external.vkpartner.ru/artifactory/maven" )  ``}}
```

集成依赖项

在您的配置文件中添加以下代码以连接依赖项：

```
dependencies {  ``implementation( "ru.rustore.sdk:review:1.0.1" )}
```

创建 *RuStoreReviewManager*

要处理评分，需要使用 RuStoreReviewManagerFactory 创建 RuStoreReviewManager：

```
val manager = RuStoreReviewManagerFactory.create(context)
```

获取对象*ReviewInfo*

提前调用 requestReviewFlow()，在调用 launchReviewFlow(reviewInfo) 之前准备显示屏幕所需的信息。

ReviewInfo有其生命周期 — 大约五分钟。

```
manager.requestReviewFlow().addOnCompleteListener(object : OnCompleteListener<ReviewInfo> {  ``override fun onFailure(throwable: Throwable) {  ``// Handle error  ``}  ``override fun onSuccess(result: ReviewInfo) {  ``// Save reviewInfo  ``}})
```

如果收到 onSuccess 响应，则需要在本地保存 ReviewInfo，以便后续调用 launchReviewFlow(reviewInfo)。

如果收到 onFailure 响应，建议不要自行向用户显示错误，因为用户没有启动这个过程。

可能的错误

启动应用程序评价

要请求用户对应用程序进行评价和评论，请调用 `launchReviewFlow(reviewInfo)` 方法，使用之前获得的 `ReviewInfo`。

```
manager.launchReviewFlow(reviewInfo).addOnCompleteListener(object: OnCompleteListener<Unit> { ``override fun onFailure(throwable: Throwable) { ``// Review flow has finished, continue your app flow. ``} ``override fun onSuccess(result: Unit) { ``// Review flow has finished, continue your app flow. ``}})
```

等待用户完成表单的通知，在 onSuccess 或 onFailure 中，以继续应用程序的操作。

完成评分表单后，无论结果如何（onSuccess 或 onFailure），我们不建议显示与评分和评论相关的任何额外表单。

频繁调用 launchReviewFlow 不会导致用户界面显示评分表单，RuStore 会对允许显示的频率进行调控。

可能的错误

在 onFailure 中您可能接收的可能错误：

- RuStoreNotInstalledException() — 用户设备上未安装 RuStore。
- RuStoreOutdatedException() — 用户设备上安装的 RuStore 不支持启动评分和评论过程。
- UserUnauthorizedException() — 用户未在 RuStore 中授权。
- RuStoreRequestLimitReached() — 自上次显示过程以来时间太短。
- RuStoreReviewExists() — 该用户已对您的应用程序进行了评价。
- RuStoreInvalidReviewInfo() — ReviewInfo 出现问题。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

RuStore SDK 评论和评分的更新历史

SDK版本1.0.1

- SDK的内部更新。

SDK版本1.0.0

- SDK的内部更新。

SDK版本0.2.0

- 类包统一更改为 ru.rustore.**

SDK版本0.1.6

- 修复了()Task API的await方法的操作。

SDK版本0.1.5

- 修复了 “Intent Redirection vulnerability” 安全漏洞，这个漏洞之前阻止了应用在 Google Play 上的发布。

SDK版本0.1.4

- SDK的内部更新。

SDK版本0.1.3

- launchReviewFlow(activity, reviewInfo) 方法已被标记为 deprecated。请使用 launchReviewFlow(reviewInfo)。

Java

概述

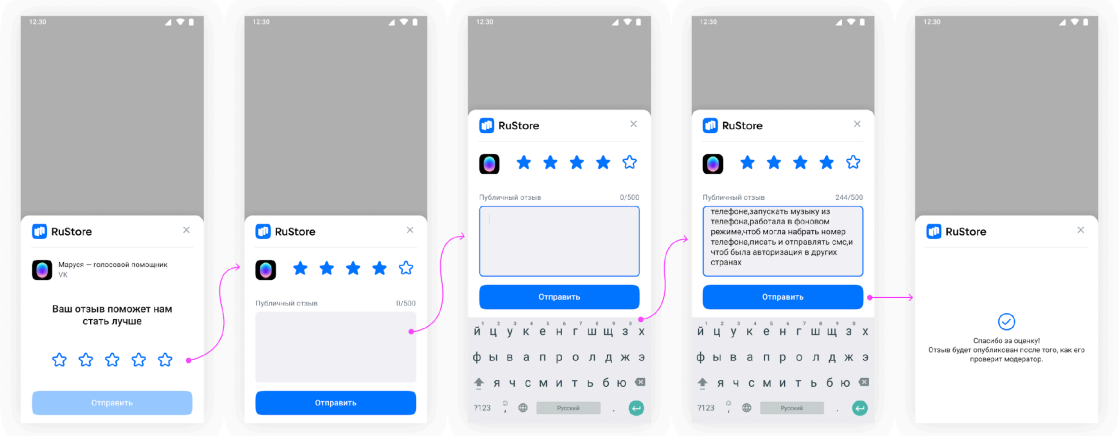
RuStore In-app Review SDK 允许用户在不离开应用的情况下在 RuStore 中对您的应用进行评分和评论。

用户可以在任何时间启动评分和评论的用户流程。用户可以根据1到5的评分等级来评价您的应用程序。评论是可选的。

实现示例

查看示例应用程序以了解如何正确集成评分和评论SDK。

用户流程示例



SDK 正确运作的条件

为了评分和评论应用程序 SDK 的正常运作，需要满足以下条件：

1. Android操作系统版本6.0或更高。

2. 用户设备上必须安装了RuStore。
3. 用户设备上的RuStore版本必须是最新的。
4. 用户必须在 RuStore 中获得授权。
5. 应用程序必须在 RuStore 上发布。

何时请求评分和评论

遵循以下建议来决定何时向用户请求应用程序评分和评论：

- 用户充分使用您的应用程序后启动流程。
- 不要过于频繁启动流程，否则会降低用户体验并限制评分SDK的使用。
- 不要使用诸如“评价应用程序”之类的行动号召按钮，因为用户可能已达到启动流程的次数上限。
- 您的应用程序在启动或进行流程时不应向用户提出任何问题，包括他们的意见（例如：“您喜欢这个应用程序吗？”）或预测性问题（例如：“您会给这个应用程序评5星吗？”）。

设计建议

遵循以下建议以决定如何集成流程：

- 如实显示流程，不进行任何干预或更改当前设计，包括大小、不透明度、形状和其他属性。
- 不要在流程的顶部或边缘添加任何内容。
- 流程应在所有层的上方打开。启动流程后不要关闭它。流程将在用户明确操作后自行结束。

在项目中集成

集成存储库

build.gradle

```
repositories {    ``maven {        ``url = uri(
"https://artifactory-external.vkpartner.ru/artifactory/maven")    ``}}
```

集成依赖项

在您的配置文件中添加以下代码以集成依赖项：

build.gradle

```
dependencies {        ``implementation( "ru.rustore.sdk:review:1.0.1" )}
```

创建 RuStoreReviewManager

为了处理评分，需要使用RuStoreReviewManagerFactory创建RuStoreReviewManager：

初始化

```
RuStoreReviewManager manager = RuStoreReviewManagerFactory.INSTANCE.create(context);
```

获取ReviewInfo对象

提前调用requestReviewFlow()，在调用launchReviewFlow(reviewInfo)之前准备显示屏幕所需的信息。

ReviewInfo的生命周期大约为五分钟。

requestReviewFlow

```
manager.requestReviewFlow().addOnCompleteListener( new
OnCompleteListener<ReviewInfo>() {        ``@Override    ``public void
onFailure(throwable: Throwable) {        ``// Handle
error    ``}    ``@Override    ``public void onSuccess(result: ReviewInfo)
{        ``// Save reviewInfo    ``}})
```

如果收到onSuccess响应，则需要在本地保存ReviewInfo，以便后续调用launchReviewFlow(reviewInfo)。

如果收到 onFailure 响应，建议不要自行向用户显示错误，因为用户没有启动这个流程。

可能的错误

启动应用程序评价

使用之前获得的ReviewInfo，调用launchReviewFlow(reviewInfo)方法启动向用户请求评分和评论的流程。

launchReviewFlow(reviewInfo)

```
manager.launchReviewFlow(reviewInfo).addOnCompleteListener( new
OnCompleteListener<Unit>() {            ``@Override    ``public void
onFailure(throwable: Throwable) {        ``// Review flow has finished, continue
your app flow.    ``}    ``@Override    ``public void onSuccess(result: Unit)
{        ``// Review flow has finished, continue your app flow.    ``}})
```

要继续应用程序的流程，请等待用户在onSuccess或onFailure中完成流程的通知。

无论评分流程的结果(是onSuccess还是onFailure)，我们都不建议显示与评分和评论相关的任何其他流程。

频繁调用launchReviewFlow不会导致用户显示评分流程，因为显示配额由RuStore调控。

可能的错误

在 onFailure 中您可能接收的可能错误：

- RuStoreNotInstalledException() — 用户设备上未安装RuStore。
- RuStoreOutdatedException() — 用户设备上的RuStore不支持启动评分和评论的流程。
- RuStoreUserUnauthorizedException() — 用户未在RuStore中授权。
- RuStoreRequestLimitReached() — 自上次显示流程以来时间太短。
- RuStoreReviewExists() — 该用户已对您的应用程序进行了评价。
- RuStoreInvalidReviewInfo() — ReviewInfo出现问题。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

RuStore SDK 评论和评分的更新历史

SDK版本 1.0.1

- SDK的内部更新。

SDK版本 1.0.0

- SDK的内部更新。

SDK版本 0.2.0

- 类包统一更改为 ru.rustore.**

SDK版本 0.1.6

- 修复了Task API中的*await()*方法的操作。

SDK版本 0.1.5

- 修复了Intent Redirection vulnerability安全漏洞，之前阻止了在Google Play发布应用程序。

SDK版本 0.1.4

- SDK的内部更新。

SDK版本 0.1.3

- 方法launchReviewFlow(activity, reviewInfo)标记为deprecated。请使用launchReviewFlow(reviewInfo)。

SDK版本 0.1.2

- SDK的内部更新。

SDK版本 0.1.1

- 发布版本

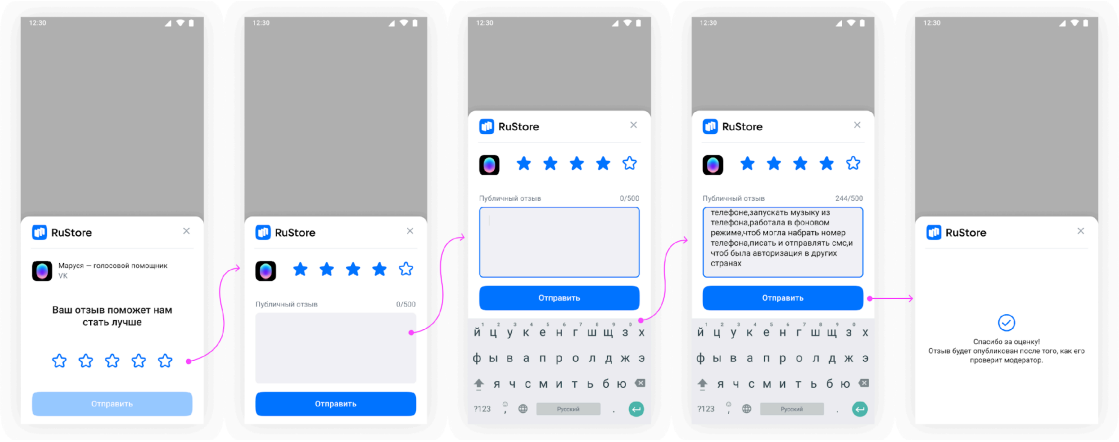
Unity

概述

RuStore In-app Review SDK推出用户在不离开应用程序的情况下，在RuStore中给您的应用程序评分和评论。

用户可以在应用程序的整个使用流程中的任何时候启动评分和评论的用户流程。用户可以按照从 1 到 5 的等级对您的应用进行评分，并留下评论。评论是可选的。

用户流程示例



SDK 正确运作的条件

为了评分和评论应用程序 SDK 的正常运作，需要满足以下条件：

1. Android操作系统版本7.0或更高。
2. 用户的设备上安装了RuStore应用程序。
3. 用户的设备上的 RuStoreApp 版本是最新的。
4. 用户在RuStore应用程序中已授权。
5. 应用程序必须在 RuStore 上发布。

何时请求评分和评论

遵循以下建议来决定何时向用户请求应用程序的评分和评论：

- 在用户充分使用您的应用程序之后启动该流程。
- 不要过于频繁地启动该流程 — 这会降低用户对您应用程序的体验并限制评分 SDK 的使用。
- 不要使用行动号召，例如“评价应用程序”的按钮 — 用户可能已经达到了启动流程的限制。
- 您的应用程序在启动或进行流程期间不应询问用户任何问题，包括关于他们的意见（例如，“您喜欢这个应用程序吗？”）或预测性问题（例如，“您会给这个应用程序评5星吗？”）。

设计建议

遵循以下建议来决定如何集成流程：

- 保持流程原样显示，不对当前设计进行任何干预或更改，包括大小、不透明度、形状和其他属性。
- 不要在流程的顶部或边缘添加任何内容。
- 流程应在所有层上方打开。启动流程后不要关闭它。流程将在用户明确操作后自行结束。

在项目中集成

为了集成，您需要下载RuStore Review SDK**并将其导入到项目中**(Assets → Import Package → Custom Package)。**依赖项将通过**External Dependency Manager(**包含在SDK中**)**自动集成。**

Minimum API level**应设置不低于**24。**目前不支持应用程序的缩减**(ProGuard/R8)，**需要在项目设置中关闭它**(File → Build Settings → Player Settings → Publishing Settings → Minify)。

创建RuStoreReviewManager

为了处理评分，需要使用RuStoreReviewManagerFactory创建RuStoreReviewManager：

```
RuStoreReviewManager.Instance.Init();
```

准备启动应用程序评价

提前调用RequestReviewFlow()，在调用LaunchReviewFlow()之前，以便准备显示屏幕所需的信息。

```
RuStoreReviewManager.Instance.RequestReviewFlow(        ``onFailure: (error) =>
{                   ``// Handle error          ``},        ``onSuccess: () =>
{                   ``// Handle success          ``});
```

如果收到onSuccess响应，则可以在大约五分钟内启动请求评价和对应用程序的评论的流程LaunchReviewFlow()。

如果收到 onFailure 响应，建议不要自行向用户显示错误，因为用户没有启动这个流程。

可能的错误

启动应用程序评价

为了启动请求用户评价和评论的流程，请调用LaunchReviewFlow()方法。

```
RuStoreReviewManager.Instance.LaunchReviewFlow(        ``onFailure: (error) =>
{                   ``// Handle error         ``},        ``onSuccess: () =>
{                   ``// Handle success         ``});
```

等待用户完成表单的通知，在 onSuccess 或 onFailure 中，以继续应用程序的操作。

评价表单完成后，无论结果是onSuccess还是onFailure，我们都不建议显示任何与评价和评论相关的额外表单。

频繁调用launchReviewFlow不会导致用户看到评价表单，允许的显示由RuStore端调控。

错误处理

出现的错误会传递给SDK方法的onFailure处理程序。

错误结构：

```
public class RuStoreError {        ``public string name;        ``public string description;}
```

- name — 错误名称；
- description — 错误描述。

在 onFailure 中您可能接收的可能错误：

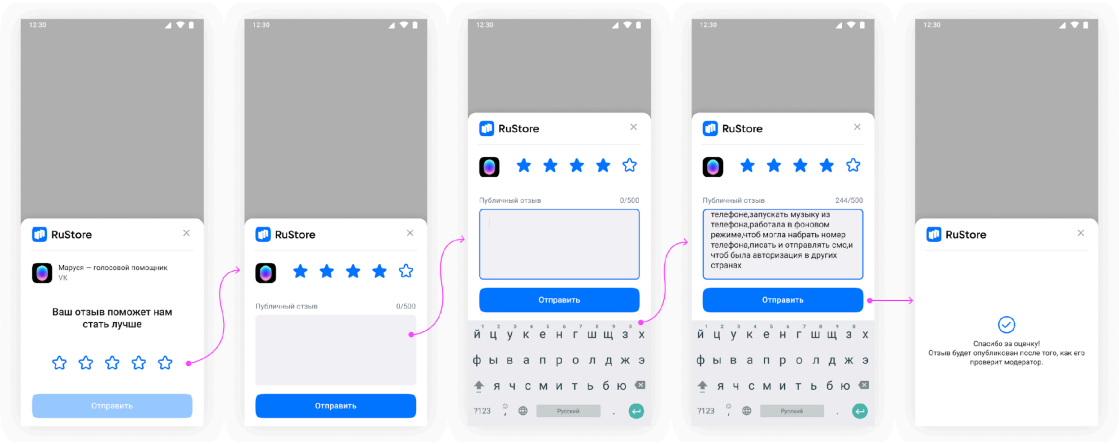
- RuStoreNotInstalledException — 用户设备上未安装RuStore。
- RuStoreOutdatedException — 用户设备上的RuStore不支持启动评分和评论的流程。
- RuStoreUserUnauthorizedException — 用户未在RuStore中授权。
- RuStoreRequestLimitReached — 自上次显示流程以来时间太短。
- RuStoreReviewExists — 该用户已对您的应用程序进行了评价。
- RuStoreInvalidReviewInfo — ReviewInfo出现问题。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

Unreal

概述

RuStore In-app Review SDK推出用户在不离开应用程序的情况下，在RuStore中给您的应用程序评分和评论。

用户可以在应用程序的整个使用流程中的任何时候启动评分和评论的用户流程。用户可以按照从 1 到 5 的等级对您的应用进行评分，并留下评论（评论是可选的）。



SDK 正确运作的条件

为了评分和评论应用程序 SDK 的正常运作，需要满足以下条件：

1. Android操作系统版本7.0或更高。
2. 用户设备上必须安装了RuStore。
3. 用户设备上的RuStoreApp版本必须是最新的。
4. 用户必须在 RuStore 中获得授权。
5. 应用程序必须在 RuStore 上发布。

何时请求评分和评论

遵循以下建议来决定何时向用户请求应用程序评分和评论：

- 在用户充分使用您的应用程序之后启动该流程。
- 不要过于频繁地启动该流程，否则这会降低用户对您应用程序的体验并限制评分 SDK 的使用。
- 不要使用诸如“评价应用程序”之类的行动号召按钮，因为用户可能已达到启动流程的次数上限。
- 您的应用程序在启动或进行流程时不应向用户提出任何问题，包括他们的意见（例如：“您喜欢这个应用程序吗？”）或预测性问题（例如：“您会给这个应用程序评5星吗？”）。

设计建议

遵循以下建议以决定如何集成流程：

- 保持流程原样显示，不对当前设计进行任何干预或更改，包括大小、不透明度、形状和其他属性。
- 不要在流程的顶部或边缘添加任何内容。
- 流程应在所有层上方打开。启动流程后不要关闭它。流程将在用户明确操作后自行结束。

在项目中集成

1. 从RuStore的官方GitFlic仓库复制 Plugins 文件夹的内容到您项目中的'Plugins'文件夹。
2. 重启Unreal Engine。
3. **在插件列表中(Edit → Plugins → Project → Mobile)，选择RuStoreReview和RuStoreCore插件。**
4. 在YourProject.Build.cs文件的PublicDependencyModuleNames列表中，集成RuStoreCore和RuStoreReview模块。
5. **在项目设置中(Edit → Project Settings → Android)，将Minimum SDK Version参数设置为不低于24，Target SDK Version参数设置为不低于31。**

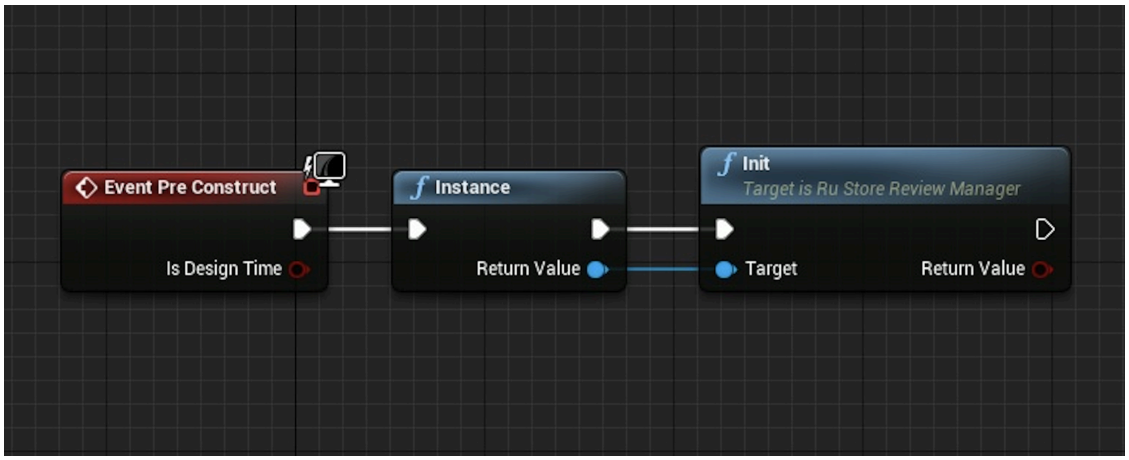
RuStoreReviewManager初始化

为了使用评分功能，需要初始化RuStoreReviewManager：

初始化

```
URuStoreReviewManager::Instance()->Init();
```

Blueprints中的使用示例：



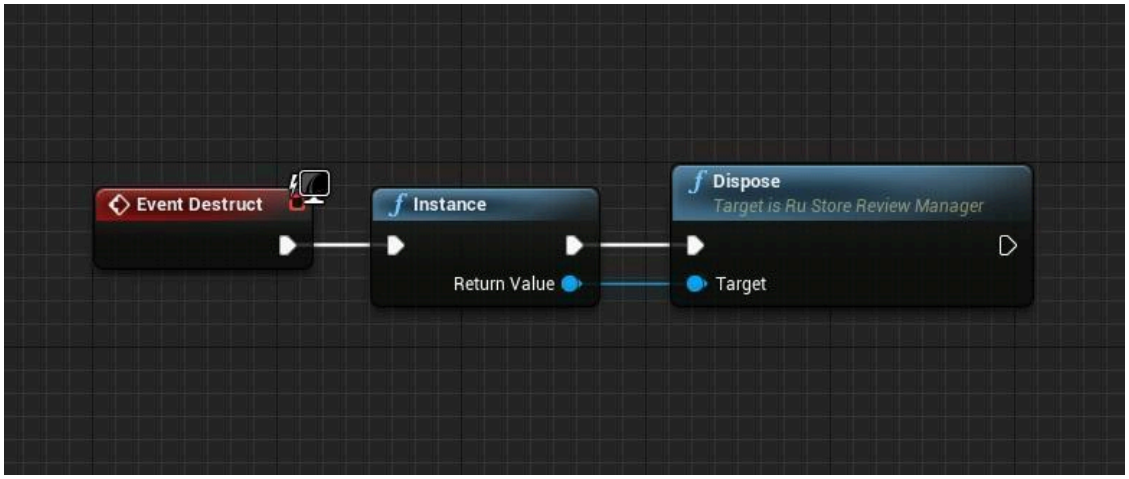
重要提示：

1. 调用Init()将对象绑定到场景的根部，如果之后不再计划使用该对象，为了释放内存，需要执行Dispose()方法。

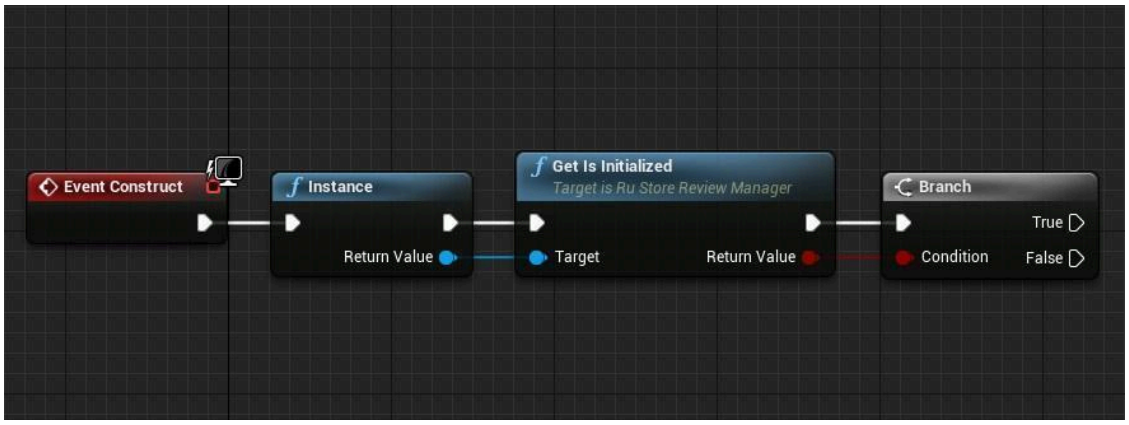
调用 Dispose 方法会将对象从根部解绑，并安全地完成所有已发送的请求。

初始化

```
URuStoreReviewManager::Instance()->Dispose();
```



如果您需要检查库的初始化情况，请使用 getIsInitialized()方法。如果库已初始化，其值为 true；如果尚未调用 Init，则为 false。



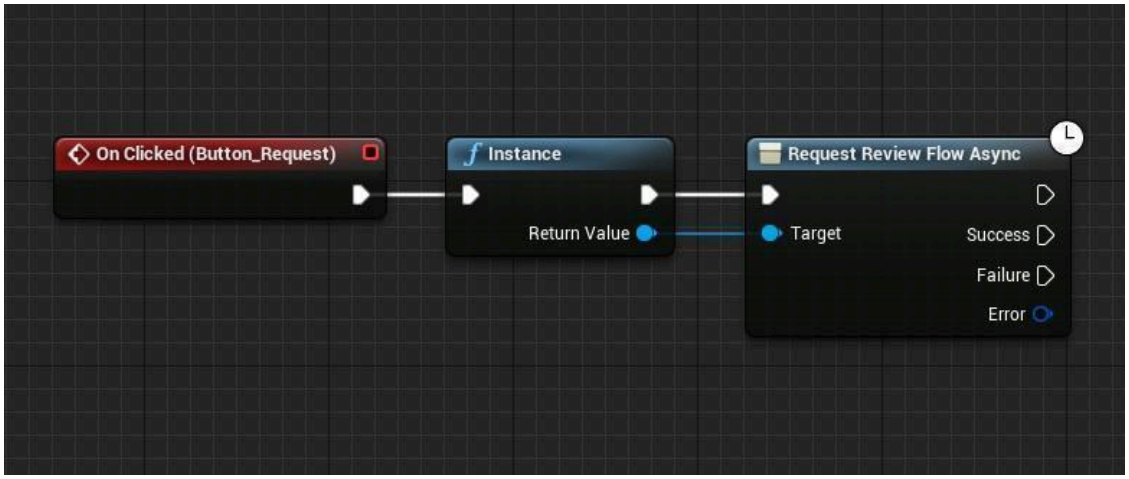
准备启动应用程序评价

提前调用RequestReviewFlow()，在调用LaunchReviewFlow()之前，以便准备显示屏幕所需的信息。

RequestReviewFlow

```
long requestId = URuStoreReviewManager::Instance()->RequestReviewFlow(        ``[](
long requestId) {                ``// Process response        ``},        ``[]( long
requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error)
{                ``// Process error        ``});
```

Blueprints中的使用示例：



如果收到回调Success，则可以在大约五分钟后启动应用程序评价和评论的流程LaunchReviewFlow()。

Failure 回调返回 FURuStoreError 结构体，包含错误信息，在Error参数中。所有可能的FURuStoreException错误都在“可能的错误”部分有所描述。

启动应用程序评价

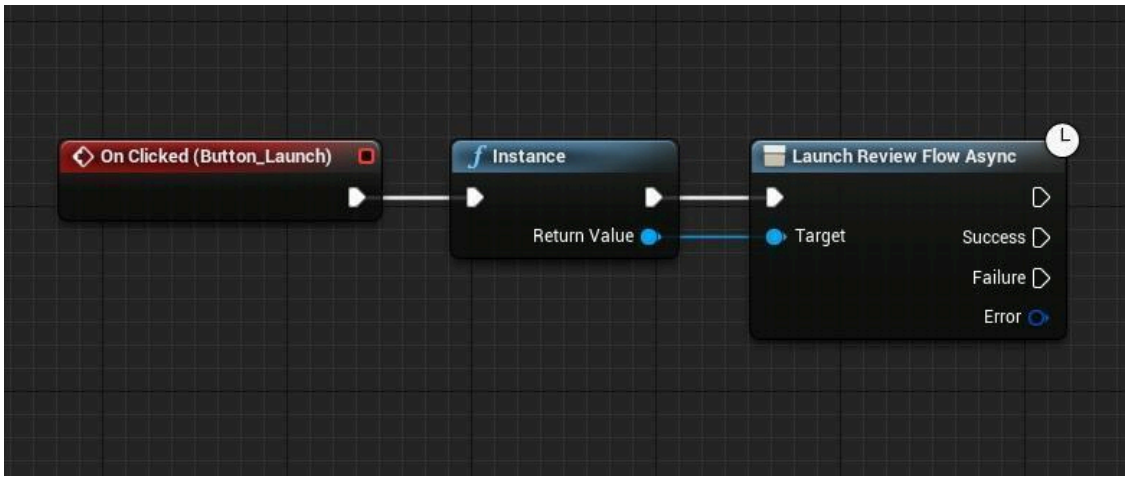
调用LaunchReviewFlow()方法，启动向用户请求评价和评论的流程（图1）。

每个请求在应用程序单次运行期间获得一个独特的requestId。每个事件都会返回触发该事件的请求的 requestId。

LaunchReviewFlow

```
long requestId = URuStoreReviewManager::Instance()->LaunchReviewFlow(        ``[](
long requestId) {                ``// Process response        ``},          ``[](
long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error)
{                ``// Process error        ``});
```

Blueprints中的使用示例：



要继续应用程序的流程，请等待用户在onSuccess或onFailure中完成流程的通知。

评价流程完成后，无论结果是onSuccess还是onFailure，我们都不建议显示任何与评价和评论相关的额外流程。

频繁调用launchReviewFlow不会导致用户显示评分流程，因为显示配额由RuStore调控。

可能的错误

出现的错误会传递给SDK方法的onFailure处理程序。

错误结构：

错误结构体

```
USTRUCT(BlueprintType)struct RUSTORECORE_API
FURuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreError() ``{
``name = ”” ; ``description = ””
; ``} ``UPROPERTY(BlueprintReadOnly) ``FString
name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};
```

- name — 错误名称。
- description — 错误描述。

在 onFailure 中您可能接收的可能错误：

- RuStoreNotInstalledException — 用户设备上未安装RuStore。
- RuStoreOutdatedException — 用户设备上的RuStore不支持启动评分和评论的流程。
- RuStoreUserUnauthorizedException — 用户未在RuStore中授权。
- RuStoreRequestLimitReached — 自上次显示流程以来时间太短。
- RuStoreReviewExists — 该用户已对您的应用程序进行了评价。
- RuStoreInvalidReviewInfo — ReviewInfo出现问题。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。

RuStore SDK 评论和评分的更新历史

SDK版本 1.0

- 添加了 blueprint 方法的异步选项，取代对事件的订阅。
- 错误修复。

Flutter

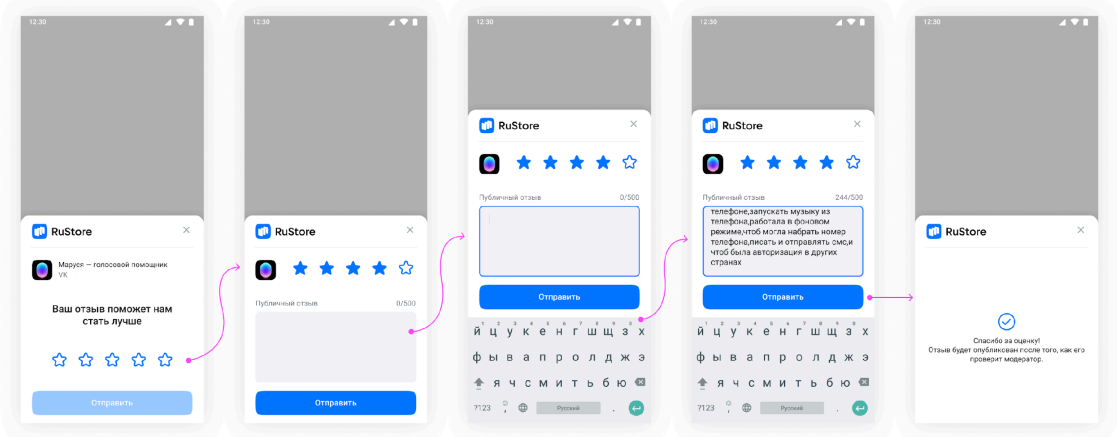
概述

RuStore In-app Review SDK推出用户在不离开应用程序的情况下，在RuStore中给您的应用程序评分和评论。

用户可以在应用程序的整个使用流程中的任何时候启动评分和评论的用户流程。用户可以按照从 1 到 5 的等级对您的应用进行评分，并留下评论。评论是可选的。

参阅示例应用程序，以了解如何正确集成评分和评论SDK：
https://gitflic.ru/project/rustore/flutter-rustore-review

用户流程示例



SDK 正确运作的条件

为了评分和评论应用程序 SDK 的正常运作，需要满足以下条件：

1. Android操作系统版本7.0或更高。
2. 用户的设备上安装了RuStore应用程序。
3. 用户的设备上的 RuStoreApp 版本是最新的。
4. 用户在RuStore应用程序中已授权。
5. 应用程序必须在 RuStore 上发布。

何时请求评分和评论

遵循以下建议来决定何时向用户请求应用程序的评分和评论：

- 在用户充分使用您的应用程序之后启动该流程。
- 不要过于频繁地启动该流程 — 这会降低用户对您应用程序的体验并限制评分 SDK 的使用。
- 不要使用行动号召，例如“评价应用程序”的按钮 — 用户可能已经达到了启动流程的限制。
- 您的应用程序在启动或进行流程期间不应询问用户任何问题，包括关于他们的意见（例如，“您喜欢这个应用程序吗？”）或预测性问题（例如，“您会给这个应用程序评5星吗？”）。

设计建议

遵循以下建议来决定如何集成流程：

- 保持流程原样显示，不对当前设计进行任何干预或更改，包括大小、不透明度、形状和其他属性。
- 不要在流程的顶部或边缘添加任何内容。
- 流程应在所有层上方打开。启动流程后不要关闭它。流程将在用户明确操作后自行结束。

在项目中集成

要将包集成到项目，您需要执行以下命令：

```
flutter pub add flutter_rustore_review
```

集成依赖项

这个命令将在 pubspec.yaml 文件中添加一个字符串。

```
dependencies:    ``flutter_rustore_review: ^0.0.3 <br>
```

请求评分

为了显示带有评分和评论表单的窗口，需要执行插件的初始化：

```
RustoreReviewClient.initialize();
```

初始化后可以进行请求并显示表单。

```
RustoreReviewClient.request().then((value)
{  ``RustoreReviewClient.review().then((value) {    ``print( "success review"
);  ``}, onError: (err) {    ``print( "on err ${err}" );  ``});});
```

React Native

概述

实现示例

查看示例应用程序，了解如何正确集成评论。

SDK 正确运作的条件

为了评分和评论应用程序 SDK 的正常运作，需要满足以下条件：

1. Android操作系统版本7.0或更高。
2. 用户的设备上安装了RuStore应用程序。
3. 用户的设备上的 RuStoreApp 版本是最新的。
4. 用户在RuStore应用程序中已授权。
5. 应用程序必须在 RuStore 上发布。

何时请求评分和评论

遵循以下建议来决定何时向用户请求应用程序的评分和评论：

- 在用户充分使用您的应用程序之后启动该流程。
- 不要过于频繁地启动该流程 — 这会降低用户对您应用程序的体验并限制评分 SDK 的使用。
- 不要使用行动号召，例如“评价应用程序”的按钮 — 用户可能已经达到了启动流程的限制。
- 您的应用程序在启动或进行流程期间不应询问用户任何问题，包括关于他们的意见（例如，“您喜欢这个应用程序吗？”）或预测性问题（例如，“您会给这个应用程序评5星吗？”）。

设计建议

遵循以下建议以决定如何集成流程：

- 保持流程原样显示，不对当前设计进行任何干预或更改，包括大小、不透明度、形状和其他属性。
- 不要在流程的顶部或边缘添加任何内容。
- 流程应在所有层上方打开。启动流程后不要关闭它。流程将在用户明确操作后自行结束。

在项目中集成

要将包集成到项目，您需要执行以下命令：

```
// HTTPSnpm   install git+https: //git @gitflic.ru:rustore
/react-native-rustore-review-sdk .git// SSHnpm   install git+ ssh : //git
@gitflic.ru:rustore /react-native-rustore-review-sdk .git
```

请求评分

为了显示带有评分和评论表单的窗口，需要执行插件的初始化：

```
RustoreReviewClient.init();
```

初始化后可以进行请求并显示表单。

```
try{  ``const isRequested = await
RustoreReviewClient.requestReviewFlow();  ``if(isRequested) {    ``await
RustoreReviewClient.launchReviewFlow();  ``}} catch(err)
{  ``console.log(err);}
```

RuStore 应用程序更新SDK

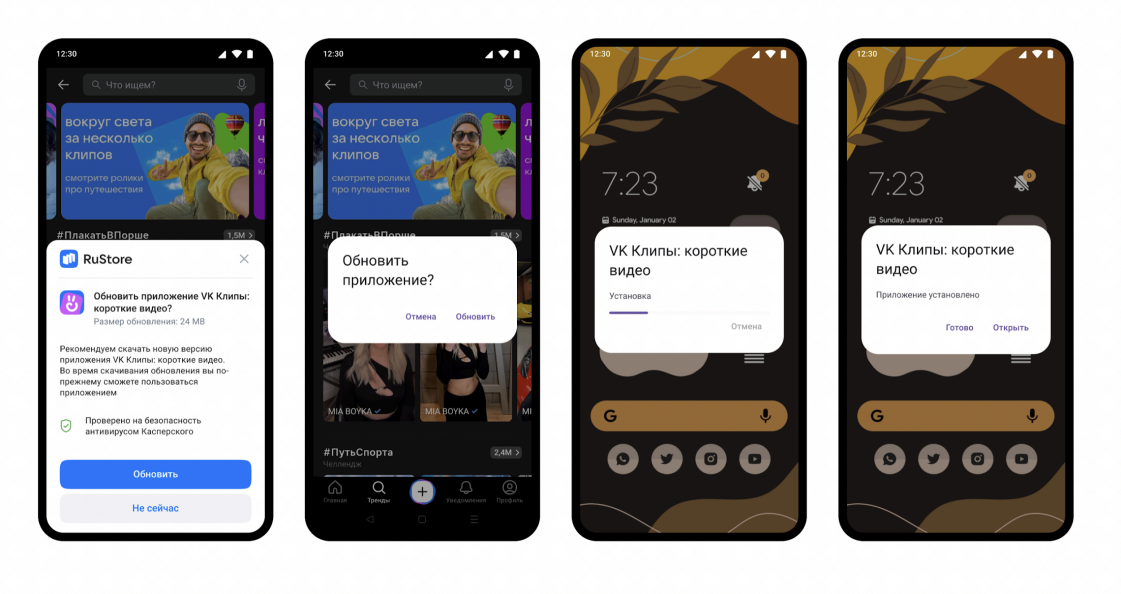
Kotlin

概述

RuStore In-app updates SDK 支持用户设备上应用程序的最新版本。这有助于用户看到更新，评估性能改进和错误修复的结果。

使用 RuStore In-app updates SDK 实现各种更新方式。目前支持的更新方式包括：延迟更新、无界面（无 RuStore UI）和强制更新。

用户流程示例



SDK 正确运作的条件

RuStore In-app updates SDK 运行需要满足以下条件：

1. Android操作系统版本6.0或更高。
2. 用户的设备上安装了RuStore。
3. 用户的设备上的 RuStore 版本是最新的。
4. RuStore应用程序被授权安装应用程序。

在项目中集成

集成存储库：

```
repositories {    ``maven {        ``url = uri( "https://artifactory-external.vkpartner.ru/artifactory/maven" )    ``}}
```

集成依赖项

在您的配置文件中添加以下代码以集成依赖项：

build.gradle

dependencies { ``implementation(` “ru.rustore.sdk:appup

创建更新管理器

在调用库方法之前，需要使用工厂创建一个更新管理器。

```
val updateManager = RuStoreAppUpdateManagerFactory.create(context)
```

检查更新的可用性

在请求更新之前，请检查您的应用是否有更新可用。要检查更新的可用性，请调用 getAppUpdateInfo() 方法。调用此方法时，将检查以下条件：

1. 用户设备上必须安装了RuStore。
2. 用户设备上的RuStoreApp版本必须是最新的。
3. 用户和应用程序不应在RuStore中被封锁。

在响应此方法时，您将获得一个 AppUpdateInfo 对象，其中包含有关更新需求的信息。请提前请求此对象并对其进行缓存，以便在用户方便的时间无延迟地请求用户启动更新下载。

```
var appUpdateInfo: AppUpdateInfo? =
nullupdateManager    ``.getAppUpdateInfo()    ``.addOnSuccessListener { info
->        ``appUpdateInfo =
info                    ``}    ``.addOnFailureListener { throwable
->                    ``}
```

AppUpdateInfo 对象包含一组参数，这些参数是确定更新可用性所必需的：

- updateAvailability — 更新的可用性：
  - UPDATE_NOT_AVAILABLE (Int == 1) — 不需要更新。
  - UPDATE_AVAILABLE (Int == 2) — 需要下载更新或用户设备上已下载更新。
  - DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS (Int == 3) — 更新已在下载或安装已启动。
  - UNKNOWN (Int == 0) — 默认状态。
- installStatus — 如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED (Int == 1) — 已下载。
  - DOWNLOADING (Int == 2) — 正在下载。
  - FAILED (Int == 3) — 错误。
  - INSTALLING (Int == 4) — 正在安装。
  - PENDING (Int == 5) — 等待中。

- UNKNOWN (Int == 0) — 默认状态。

只有当 updateAvailability 字段包含 UPDATE_AVAILABLE 值时，才能启动更新下载。

方法可能返回错误。可能的错误

更新流程

强制更新

检查可用性

在获取 AppUpdateInfo 后，可以检查强制更新的可用性。

调用方法 registerListener() 示例

```
if (appUpdateInfo.isUpdateTypeAllowed(IMMEDIATE)) {    ``TODO()}
```

建议使用 isUpdateTypeAllowed 函数的结果来决定是否启动强制更新，但此结果不影响启动流程的可能性。更新流程的启动需求可以根据您的内部逻辑进行。

启动流程

调用方法 startUpdateFlow() 示例

```
updateManager    ``.startUpdateFlow(appUpdateInfo,
AppUpdateOptions.Builder().appUpdateType(IMMEDIATE).build())    ``.addOnSuccessL
istener { resultCode ->    ``}    ``.addOnFailureListener { throwable ->    ``}
```

resultCode (Int) :

- Activity.RESULT_OK (-1) — 更新完成，代码可能未被接收，因为应用程序在更新时会结束。
- Activity.RESULT_CANCELED (0) — 流程被用户中断或发生错误。当接收到此代码时，通常应终止应用程序。
- ActivityResult.ACTIVITY_NOT_FOUND (2) — RuStore 未安装，或安装的版本不支持强制更新（RuStore versionCode < 191）

throwable — 启动更新流程的错误。更多错误列表可以在“可能的错误”部分找到。

延迟更新

下载更新

在确认更新可用性后，您可以启动延迟更新流程，其第一步是进行背景下载更新。

为了跟踪下载过程，需要使用registerListener()方法添加监听器

调用方法 registerListener() 示例

```
updateManager.registerListener { state ->    ``if (state.installStatus ==
InstallStatus.DOWNLOADED) {        ``// Update is ready to
install    ``}          }
```

State对象描述了更新下载的当前状态。对象包含：

- installStatus — 如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED (1) — 已下载。
  - DOWNLOADING (2) — 正在下载。
  - FAILED (3) — 错误。
  - INSTALLING (4) — 正在安装。
  - PENDING (5) — 等待中。
  - UNKNOWN (0) — 默认状态。
- bytesDownloaded — 已下载的字节数。
- totalBytesToDownload — 需要下载的总字节数。
- installErrorCode — 下载过程中的错误代码。更多可能的错误可以在“可能的错误”部分找到。

如果不再需要监听器，则可以使用unregisterListener()方法删除监听器，传递先前注册的监听器。

方法调用示例unregisterListener()

```
updateManager.unregisterListener(listener)
```

要启动应用程序更新的下载，请调用startUpdateFlow()方法，并在其中包含从getAppUpdateInfo()方法获得的AppUpdateInfo对象。

AppUpdateInfo对象在一次使用后将失效。要重新调用startUpdateFlow()方法，请再次使用getAppUpdateInfo()方法请求AppUpdateInfo。

调用方法 startUpdateFlow() 示例

```
updateManager    ``.startUpdateFlow(appUpdateInfo,
AppUpdateOptions.Builder().build())    ``.addOnSuccessListener { resultCode
->    ``}    ``.addOnFailureListener { throwable ->    ``}
```

如果用户确认了下载更新，则resultCode = Activity.RESULT_OK；如果拒绝，则resultCode = Activity.RESULT_CANCEL。

方法可能返回错误。可能的错误

在监听器中获得“DOWNLOADED”状态后，可以调用更新安装方法。建议通知用户更新已准备好安装。

更新安装

要启动更新的安装，请调用CompleteUpdate()方法。

调用方法 CompleteUpdate() 示例

```
updateManager    ``.completeUpdate()    ``.addOnFailureListener { throwable
->                    ``}
```

更新通过android原生工具进行。如果更新成功，应用程序将关闭。

在更新阶段可能会出现错误。可能的错误

无界面更新

下载更新

建议实现自己的接口来处理更新。

在确认更新可用性后，您可以启动无界面更新流程，其第一步是进行背景下载更新。

为了跟踪下载过程，需要使用registerListener()方法添加监听器

调用方法 registerListener() 示例

```
updateManager.registerListener { state ->    ``if(state.installStatus ==
InstallStatus.DOWNLOADED) {        ``// Update is ready to
install    ``}          }
```

State对象描述了更新下载的当前状态。对象包含：

- installStatus — 如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED (1) — 已下载。
  - DOWNLOADING (2) — 正在下载。
  - FAILED (3) — 错误。
  - INSTALLING (4) — 正在安装。
  - PENDING (5) — 等待中。
  - UNKNOWN (0) — 默认状态。
- bytesDownloaded — 已下载的字节数。
- totalBytesToDownload — 需要下载的总字节数。
- installErrorCode — 下载过程中的错误代码。更多可能的错误可以在“可能的错误”部分找到。

如果不再需要监听器，则可以使用unregisterListener()方法删除监听器，传递先前注册的监听器。

调用方法 unregisterListener() 示例

```
updateManager.unregisterListener(listener)
```

要启动应用程序更新的下载，需要调用startUpdateFlow()方法，带有从getAppUpdateInfo()方法获取的AppUpdateInfo参数，并将更新类型设置在AppUpdateOptions为SILENT。

AppUpdateInfo对象在一次使用后将失效。要重新调用startUpdateFlow()方法，请再次使用getAppUpdateInfo()方法请求AppUpdateInfo。

调用方法 startUpdateFlow() 示例

```
val appUpdateOptions =
AppUpdateOptions.Builder().appUpdateType(SILENT).build()updateManager    ``.star
tUpdateFlow(appUpdateInfo, appUpdateOptions)    ``.addOnSuccessListener {
resultCode ->    ``}    ``.addOnFailureListener { throwable ->    ``}
```

在调用onSuccessListener时，如果resultCode = Activity.RESULT_OK，将注册一个更新下载任务。

在此流程中，只能调用onSuccessListener，其resultCode = Activity.RESULT_OK，或onFailureListener，详细的错误列表可在“可能的错误”部分找到。

在监听器中获得“DOWNLOADED”状态后，可以调用更新安装方法。建议通知用户更新已准备好安装。

更新安装

要启动更新的安装，请调用CompleteUpdate()方法。

调用方法 CompleteUpdate() 示例

```
updateManager    ``.completeUpdate()    ``.addOnFailureListener { throwable
->                    ``}
```

更新通过android原生工具进行。如果更新成功，应用程序将关闭。

在更新阶段可能会出现错误。可能的错误

可能的错误

如果您收到onFailure响应，则不建议自行向用户显示错误。显示错误可能会对用户体验产生负面影响。

可能的错误列表：

- RuStoreNotInstalledException() — 用户设备上未安装RuStore。
- RuStoreOutdatedException() — 用户设备上的RuStore不支持应用程序的更新；
- RuStoreUserUnauthorizedException() — 用户未在RuStore中授权。更新付费应用程序需要授权；
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。
- RuStoreInstallException(code: Int) — 下载和安装错误：

- 错误代码：- ERROR_UNKNOWN(Int = 4001) — 未知错误；
  - ERROR_DOWNLOAD(Int = 4002) — 下载错误；
  - ERROR_BLOCKED(Int = 4003) — 系统阻止安装；
  - ERROR_INVALID_APK(Int = 4004) — 更新APK文件无效；
  - ERROR_CONFLICT(Int = 4005) — 与当前应用程序版本冲突；
  - ERROR_STORAGE(Int = 4006) — 设备存储空间不足；
  - ERROR_INCOMPATIBLE(Int = 4007) — 与设备不兼容；
  - ERROR_APP_NOT_OWNED(Int = 4008) — 应用程序未购买；
  - ERROR_INTERNAL_ERROR(Int = 4009) — 内部错误；
  - ERROR_ABORTED(Int = 4010) — 用户拒绝安装更新；
  - ERROR_APK_NOT_FOUND(Int = 4011) — 未找到用于启动安装的APK文件；
  - ERROR_EXTERNAL_SOURCE_DENIED(Int = 4012) — 禁止启动更新。例如，在第一个方法中返回更新不可用的响应，但用户调用第二个方法。
  - ERROR_ACTIVITY_SEND_INTENT(Int = 9901) — 发送intent以开启活动的错误
  - ERROR_ACTIVITY_UNKNOWN(Int = 9902) — 开启活动的未知错误

RuStore SDK应用程序更新历史

SDK版本 1.0.1

- SDK的内部更新。

SDK版本 1.0.0

- SDK的内部更新。

SDK版本 0.2.0

- 添加了强制更新流程。
- 支持传递VersionCode类型为Long。

SDK版本 0.1.2

- 添加了Silent更新功能。

SDK版本 0.1.1

- 修复了*Task API*中的await()方法的操作。

Java

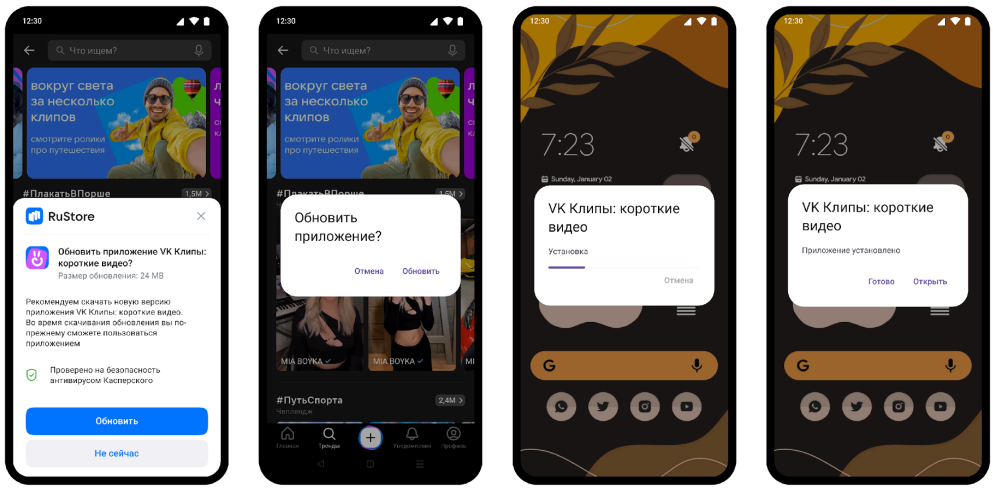
概述

RuStore In-app updates SDK帮助维持用户设备上应用程序的最新版本。

当用户维持应用程序的最新状态时，他们可以体验新功能并利用性能改进和错误修复。

您可以使用RuStore In-app updates SDK来实现各种更新方式。目前支持的更新方式包括：延迟更新、无界面（无 RuStore UI）和强制更新。

用户流程示例



SDK 正确运作的条件

RuStore In-app updates SDK 运行需要满足以下条件：

1. Android操作系统版本6.0或更高。
2. 用户设备上必须安装了RuStore。
3. 用户设备上的RuStoreApp版本必须是最新的。
4. RuStore应用程序必须被授权安装应用程序。

在项目中集成

集成存储库

build.gradle

```
repositories {    ``maven {        ``url = uri( "https://artifactory-external.vkpartner.ru/artifactory/maven" )    ``}}
```

集成依赖项

在您的配置文件中添加以下代码以集成依赖项：

build.gradle

```
dependencies {        ``implementation( "ru.rustore.sdk:review:1.0.1" )}
```

创建更新管理器

在调用库方法之前，需要使用工厂创建一个更新管理器。

```
RuStoreAppUpdateManager ruStoreAppUpdateManager =
RuStoreAppUpdateManagerFactory.INSTANCE.create(context);
```

检查更新的可用性

在启动更新流程之前，请检查您的应用程序是否有可用的更新。要检查更新的可用性，请调用getAppUpdateInfo()方法。调用此方法时，将检查以下条件：

1. 用户设备上必须安装了RuStore。
2. 用户设备上的RuStoreApp版本必须是最新的。
3. 用户和应用程序不应在RuStore中被封锁。

在响应此方法时，您将获得一个 AppUpdateInfo 对象，其中包含有关更新需求的信息。请提前请求此对象并对其进行缓存，以便在用户方便的时间无延迟地请求用户启动更新下载。

调用方法 getAppUpdateInfo() 示例

```
AppUpdateInfo appUpdateInfo =
nullruStoreAppUpdateManager      ``.getAppUpdateInfo()      ``.addOnSuccessListe
ner(info -> {              ``appupdateInfo =
info           ``.addOnFailureListener(throwable
->                                ``);
```

AppUpdateInfo 对象包含一组参数，这些参数是确定更新可用性所必需的：

- updateAvailability—更新的可用性：
  - UPDATE_NOT_AVAILABLE(Int == 1)—不需要更新。
  - UPDATE_AVAILABLE(Int == 2)—需要下载更新或用户设备上已下载更新。
  - DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS(Int == 3)—更新已在下载或安装已启动。
  - UNKNOWN(Int == 0)—默认状态。
- installStatus—如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED(Int == 1)—已下载。
  - DOWNLOADING(Int == 2) — 正在下载。
  - FAILED(Int == 3) — 错误。
  - INSTALLING(Int == 4) — 正在安装。
  - PENDING(Int == 5) — 等待中。
  - UNKNOWN(Int == 0) — 默认状态。

启动更新流程只有在updateAvailability字段包含UPDATE_AVAILABLE值时才可能。

可能的错误

更新流程

强制更新

检查可用性

在获取 AppUpdateInfo 后，可以检查强制更新的可用性。

方法调用示例registerListener()

```
if (appUpdateInfo.isUpdateTypeAllowed(IMMEDIATE)) {    ``TODO()}
```

建议使用 isUpdateTypeAllowed 函数的结果来决定是否启动强制更新，但此结果不影响启动流程的可能性。更新流程的启动需求可以根据您的内部逻辑进行。

启动流程

方法调用示例startUpdateFlow()

```
IruStoreAppUpdateManager       ``.startUpdateFlow(appUpdateInfo, new
AppUpdateOptions.Builder().build())       ``.addOnSuccessListener(resultCode ->
{                    ``})       ``.addOnFailureListener(throwable
->                    ``);
```

resultCode (Int) :

- Activity.RESULT_OK(-1) — 更新完成，代码可能未被接收，因为应用程序在更新时会结束。
- Activity.RESULT_CANCELED(0) — 流程被用户中断或发生错误。当接收到此代码时，通常应终止应用程序。
- ActivityResult.ACTIVITY_NOT_FOUND(2) — RuStore 未安装，或安装的版本不支持强制更新（RuStore versionCode (RuStore versionCode < 191)

throwable—启动更新流程的错误。

可能的错误

延迟更新与RuStore的 UI

下载更新

确认更新可用后，您可以启动延迟更新流程，其第一步是后台下载更新。

要跟踪下载进程，需要使用registerListener()方法添加监听器。

方法调用示例registerListener()

```
ruStoreAppUpdateManager.registerListener(installState -> {                ``if
(installState.getInstallStatus() == InstallStatus.DOWNLOADED)
{                          ``// 更新已准备好安装              ``}});
```

state对象描述了更新下载的当前状态。对象包含：

- installStatus—如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED(1) — 已下载。
  - DOWNLOADING(2) — 正在下载。
  - FAILED(3) — 错误。
  - INSTALLING(4) — 正在安装。
  - PENDING(5) — 等待中。
  - UNKNOWN(0) — 默认状态。
- bytesDownloaded—已下载的字节数。
- totalBytesToDownload—需要下载的总字节数。
- installErrorCode—下载过程中的错误代码。

可能的错误

如果不再需要监听器，则可以使用unregisterListener()方法删除监听器，传递先前注册的监听器。

方法调用示例unregisterListener()

```
ruStoreAppUpdateManager.unregisterListener(installStateUpdateListener);
```

要启动应用程序更新的下载，请调用startUpdateFlow()方法，并在其中包含从getAppUpdateInfo()方法获得的AppUpdateInfo对象。

AppUpdateInfo对象在一次使用后将失效。要重新调用startUpdateFlow()方法，请再次使用getAppUpdateInfo()方法请求AppUpdateInfo。

方法调用示例startUpdateFlow()

```
updateManager    ``.startUpdateFlow(appUpdateInfo,
AppUpdateOptions.Builder().build())    ``.addOnSuccessListener { resultCode
->    ``}    ``.addOnFailureListener { throwable
->    ``}ruStoreAppUpdateManager     ``.startUpdateFlow(appUpdateInfo, new
AppUpdateOptions.Builder().build())     ``.addOnSuccessListener(resultCode ->
```

```
{                    ``})      ``.addOnFailureListener(throwable
->               ``);
```

如果用户确认了下载更新，则resultCode = Activity.RESULT_OK；如果拒绝，则resultCode = Activity.RESULT_CANCEL。

可能的错误

在监听器中获得DOWNLOADED状态后，可以调用更新安装方法。建议通知用户更新已准备好安装。

更新安装

要启动更新的安装，请调用completeUpdate()方法。

方法调用示例completeUpdate()

```
ruStoreAppUpdateManager    ``.completeUpdate()    ``.addOnFailureListener(throwa
ble -> {                        ``});
```

更新通过android原生工具进行。如果更新成功，应用程序将关闭。

在更新阶段可能会出现错误。

可能的错误

无界面更新(没有UI来自RuStore)

下载更新

我们建议实现自己的接口来处理更新，以避免破坏您应用程序中的用户体验。

在确认更新可用性后，您可以启动无界面更新流程，其第一步是后台下载更新。为了跟踪下载过程，需要使用registerListener()方法添加监听器。

方法调用示例registerListener()

```
ruStoreAppUpdateManager.registerListener { state ->    ``if(state.installStatus
== InstallStatus.DOWNLOADED) {        ``// Update is ready to
install    ``}          }
```

state对象描述了更新下载的当前状态。对象包含：

- installStatus—如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED(1) — 已下载。

  - DOWNLOADING(2) — 正在下载。
  - FAILED(3) — 错误。
  - INSTALLING(4) — 正在安装。
  - PENDING(5) — 等待中。
  - UNKNOWN(0) — 默认状态。
- bytesDownloaded—已下载的字节数。
- totalBytesToDownload—需要下载的总字节数。
- installErrorCode—下载过程中的错误代码。

可能的错误

如果不再需要监听器，则可以使用unregisterListener()方法删除监听器，传递先前注册的监听器。

方法调用示例unregisterListener()

```
ruStoreAppUpdateManager.unregisterListener(listener)
```

要启动应用程序更新的下载，需要调用startUpdateFlow()方法，带有从getAppUpdateInfo()方法获取的AppUpdateInfo参数，并将更新类型设置在AppUpdateOptions为SILENT。

AppUpdateInfo对象在一次使用后将失效。要重新调用startUpdateFlow()方法，请再次使用AppUpdateInfo方法请求getAppUpdateInfo()。

方法调用示例startUpdateFlow()

```
val appUpdateOptions = AppUpdateOptions.Builder().appUpdateType(SILENT).build()
 ruStoreAppUpdateManager      ``.startUpdateFlow(appUpdateInfo,
appUpdateOptions)    ``.addOnSuccessListener { resultCode
->    ``}    ``.addOnFailureListener { throwable ->    ``}
```

在调用onSuccessListener时，如果resultCode = Activity.RESULT_OK，将注册一个更新下载任务。

在此流程中，只能调用onSuccessListener，其resultCode = Activity.RESULT_OK，或onFailureListener。

可能的错误

在监听器中获得DOWNLOADED状态后，可以调用更新安装方法。建议通知用户更新已准备好安装。

更新安装

要启动更新的安装，请调用completeUpdate()方法。

方法调用示例completeUpdate()

```
ruStoreAppUpdateManager  ``.completeUpdate()  ``.addOnFailureListener {
throwable ->                    ``}
```

更新通过android原生工具进行。如果更新成功，应用程序将关闭。

在更新阶段可能会出现错误。

可能的错误

可能的错误

如果您收到onFailure响应，则不建议自行向用户显示错误。显示错误可能会对用户体验产生负面影响。

可能的错误列表：

- RuStoreNotInstalledException() — 用户设备上未安装RuStore。
- RuStoreOutdatedException() — 用户设备上的RuStore不支持应用程序的更新。
- RuStoreUserUnauthorizedException() — 用户未在RuStore中授权。更新付费应用程序需要授权。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。在其他所有情况下出现错误，例如，当与服务器连接出现问题时。
- RuStoreInstallException(code: Int) — 下载和安装错误：
  - 错误代码: - ERROR_UNKNOWN(Int = 4001) — 未知错误。
    - ERROR_DOWNLOAD(Int = 4002) — 下载错误。
    - ERROR_BLOCKED(Int = 4003) — 系统阻止安装。
    - ERROR_INVALID_APK(Int = 4004) — 更新APK无效。
    - ERROR_CONFLICT(Int = 4005) — 与当前应用程序版本冲突。
    - ERROR_STORAGE(Int = 4006) — 设备存储空间不足。
    - ERROR_INCOMPATIBLE(Int = 4007) — 与设备不兼容。
    - ERROR_APP_NOT_OWNED(Int = 4008) — 应用程序未购买。
    - ERROR_INTERNAL_ERROR(Int = 4009) — 内部错误。
    - ERROR_ABORTED(Int = 4010) — 用户拒绝安装更新。

- ERROR_APK_NOT_FOUND(Int = 4011) — 未找到用于启动安装的APK。
- ERROR_EXTERNAL_SOURCE_DENIED(Int = 4012) — 禁止启动更新。例如，在第一个方法中返回更新不可用的响应，但用户调用第二个方法
- ERROR_ACTIVITY_SEND_INTENT(Int = 9901) — 发送intent以开启活动的错误
- ERROR_ACTIVITY_UNKNOWN(Int = 9902) — 开启活动的未知错误

RuStore SDK应用程序更新历史

SDK版本 1.0.1

- SDK的内部更新。

SDK版本 1.0.0

- SDK的内部更新。

SDK版本 0.2.0

- 添加了强制更新流程。
- 添加了支持传递类型为Long的VersionCode。

SDK版本 0.1.2

- 添加了Silent更新功能。

SDK版本 0.1.1

- 修复了*Task API*中的await()方法的操作。

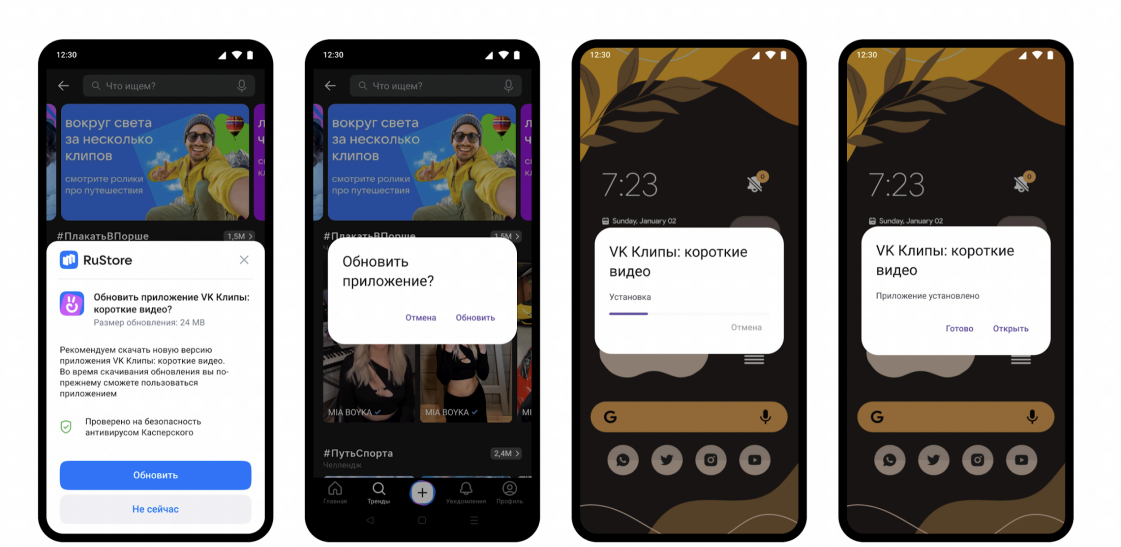
Unity

概述

RuStore In-app updates SDK 支持用户设备上应用程序的最新版本。这有助于用户看到更新，评估性能改进和错误修复的结果。

使用RuStore In-app updates SDK，以通知用户应用程序的新版本并提议安装。用户将能够在后台下载版本并查看安装进度。

用户流程示例



SDK 正确运作的条件

RuStore In-app updates SDK 运行需要满足以下条件：

1. Android操作系统版本7.0或更高。
2. 用户的设备上安装了RuStore。
3. 用户的设备上的 RuStore 版本是最新的。
4. 用户在RuStore中已授权。
5. RuStore应用程序被授权安装应用程序。

在项目中集成

要集成，请下载RuStore AppUpdate SDK**并将其导入项目(Assets → Import Package → Custom Package)。依赖项将通过SDK中包含的External Dependency Manager自动连接。**

MinimumAPIlevel应设置为不低于24。应用程序的缩减(ProGuard/R8)目前不受支持，因此需要在项目设置中将其关闭(File → Build Settings → Player Settings → Publishing Settings → Minify)。

创建更新管理器

在调用库方法之前，需要使用工厂创建一个更新管理器。

```
RuStoreAppUpdateManager.Instance.Init();
```

检查更新的可用性

在请求更新之前，请检查您的应用是否有更新可用。要检查更新的可用性，请调用 getAppUpdateInfo() 方法。调用此方法时，将检查以下条件：

1. 用户设备上必须安装了RuStore。
2. 用户设备上的RuStoreApp版本必须是最新的。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. RuStore应用程序必须被授权安装应用程序。

在响应此方法时，您将获得一个 AppUpdateInfo 对象，其中包含有关更新需求的信息。请提前请求此对象并对其进行缓存，以便在用户方便的时间无延迟地请求用户启动更新下载。

调用方法 getAppUpdateInfo() 示例

```
RuStoreAppUpdateManager.Instance.GetAppUpdateInfo(onFailure: (error) => {        ``// Handle error},onSuccess: (info) => {        ``// Process update info});
```

AppUpdateInfo 对象包含一组参数，这些参数是确定更新可用性所必需的：

- updateAvailability — 更新的可用性：
  - UPDATE_NOT_AVAILABLE — 不需要更新。
  - UPDATE_AVAILABLE — 需要下载更新或用户设备上已下载更新。
  - DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS — 更新已在下载或安装已启动。
  - UNKNOWN — 默认状态。
- installStatus — 如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED — 已下载。
  - DOWNLOADING — 正在下载。
  - FAILED — 错误。
  - INSTALLING — 正在安装。
  - PENDING — 等待中。
  - UNKNOWN — 默认状态。

只有当 updateAvailability 字段包含 UPDATE_AVAILABLE 值时，才能启动更新下载。

方法可能返回错误。详细信息关于可能的错误在“**可能的错误**”部分有说明。

下载更新

确认更新可用后，您可以请求用户下载更新，但在此之前，需要使用registerListener()方法启动更新下载状态监听器。

调用方法 registerListener() 示例

```
RuStoreAppUpdateManager.Instance.RegisterListener(listener);
```

listener — 实现 IInstallStateUpdateListener 接口的类对象。

IInstallStateUpdateListener

```
public interface IInstallStateUpdateListener {       ``public void OnStateUpdated(InstallState state);}
```

state对象描述了更新下载的当前状态，并包含：

- installStatus — 如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED — 已下载。
  - DOWNLOADING — 正在下载。
  - FAILED — 错误。
  - INSTALLING — 正在安装。
  - PENDING — 等待中。
  - UNKNOWN — 默认状态。
- bytesDownloaded — 已下载的字节数。
- totalBytesToDownload — 需要下载的总字节数。
- installErrorCode — 下载过程中的错误代码：
  - ERROR_UNKNOWN — 未知错误。
  - ERROR_DOWNLOAD — 下载错误。
  - ERROR_BLOCKED — 系统阻止安装。
  - ERROR_INVALID_APK — 更新APK无效。
  - ERROR_CONFLICT — 与当前应用程序版本冲突。
  - ERROR_STORAGE — 设备存储空间不足。
  - ERROR_INCOMPATIBLE — 与设备不兼容。
  - ERROR_APP_NOT_OWNED — 应用程序未购买。
  - ERROR_INTERNAL_ERROR — 内部错误。
  - ERROR_ABORTED — 用户拒绝安装更新。
  - ERROR_APK_NOT_FOUND — 未找到用于启动安装的apk文件。
  - ERROR_EXTERNAL_SOURCE_DENIED — 禁止启动更新。例如，在第一个方法中返回更新不可用的响应，但用户调用第二个方法。

如果不再需要监听器，则可以使用unregisterListener()方法删除监听器，传递先前注册的监听器。

调用方法 unregisterListener() 示例

```
RuStoreAppUpdateManager.Instance.UnregisterListener(listener);
```

要启动应用程序更新的下载，请调用StartUpdateFlow()方法。

AppUpdateInfo对象在一次使用后将失效。要重新调用startUpdateFlow()方法，请再次使用getAppUpdateInfo()方法请求AppUpdateInfo。

调用方法 startUpdateFlow() 示例

```
RuStoreAppUpdateManager.Instance.StartUpdateFlow(        ``onFailure: (error) =>
{                ``// Handle error        ``},        ``onSuccess: (resultCode) =>
{                ``// Handle flow result        ``});
```

如果用户确认了下载更新，则 resultCode = UpdateFlowResult.RESULT_OK, 如果拒绝，则 resultCode = UpdateFlowResult.RESULT_CANCELED。

调用方法后，您可以在监听器中跟踪更新下载的状态。如果在监听器中接收到DOWNLOADED状态，您可以调用更新安装方法。建议通知用户更新安装已准备就绪。

方法可能返回错误。详细信息在 **可能的错误**部分有说明。

更新安装

下载APK更新文件完成后，您可以启动更新的安装。要启动更新的安装，请调用CompleteUpdate()方法。

调用方法 CompleteUpdate() 示例

```
RuStoreAppUpdateManager.Instance.CompleteUpdate(        ``onFailure: (error) =>
{                ``// Handle error        ``});
```

更新通过android原生工具进行。如果更新成功，应用程序将关闭。

在更新阶段可能会出现错误。可能的错误

可能的错误

如果您收到onFailure响应，则不建议自行向用户显示错误。显示错误可能会对用户体验产生负面影响。

错误结构：

```
public class RuStoreError {        ``public string name;        ``public string
description;}
```

可能的错误列表：

- RuStoreNotInstalledException — 用户设备上未安装RuStore。
- RuStoreOutdatedException — 用户设备上的RuStore应用程序不支持应用程序的更新。
- RuStoreUserUnauthorizedException — 用户未在RuStore中授权。

- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。
- RuStoreInstallException — 下载和安装错误。

Flutter

概述

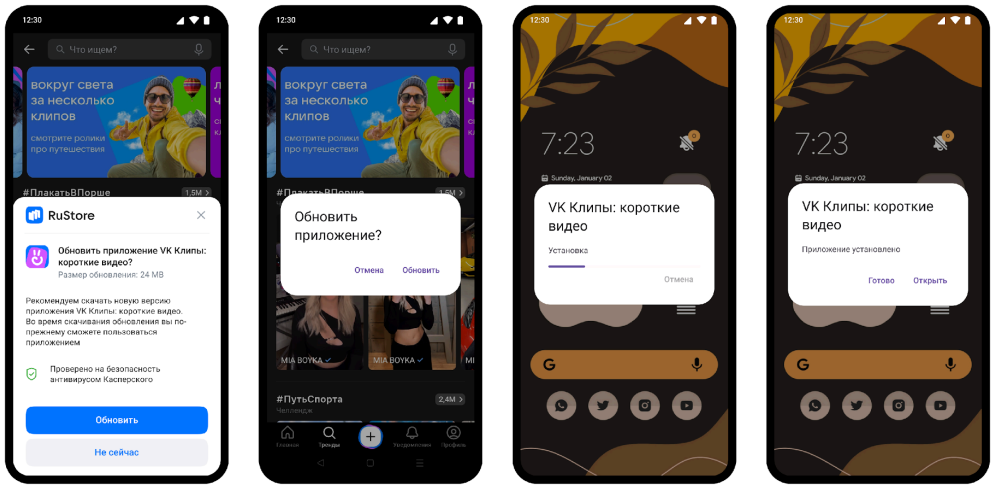
RuStore In-app updates SDK帮助维持用户设备上应用程序的最新版本。

当用户维持应用程序的最新状态时，他们可以体验新功能并利用性能改进和错误修复。

您可以使用RuStore In-app updates SDK来显示应用程序更新过程，该过程支持后台下载和安装更新以及状态控制。用户在更新下载期间可以继续使用您的应用程序。

参阅示例应用程序，以了解如何正确集成更新
SDK: https://gitflic.ru/project/rustore/flutter-rustore-update

用户流程示例



SDK 正确运作的条件

RuStore In-app updates SDK 运行需要满足以下条件：

1. Android操作系统版本6.0或更高。
2. 用户设备上必须安装了RuStore。
3. 用户设备上的RuStoreApp版本必须是最新的。
4. RuStore应用程序必须被授权安装应用程序。

实现示例

为了了解如何正确集成用于处理 push-通知的包，请参阅示例应用程序：

https://gitflic.ru/project/rustore/flutter-rustore-update/file?file=example

在项目中集成

要将包集成到项目，您需要执行以下命令：

```
flutter pub add flutter_rustore_update
```

这个命令将在 pubspec.yaml 文件中添加一个字符串：

```
dependencies:` ``flutter_rustore_up
```

检查更新的可用性

在请求更新之前，请检查您的应用是否有更新可用。要检查更新的可用性，请调用info()方法。调用此方法时，将检查以下条件：

1. 用户设备上必须安装了RuStore。
2. 用户设备上的RuStoreApp版本必须是最新的。
3. 用户和应用程序不应在RuStore中被封锁。

在响应此方法时，您将获得一个 info 对象，其中包含有关更新需求的信息。

```
RustoreUpdateClient.info().then((info) {    ``print(info);}).catchError((err)
{    ``print(err);});
```

info 对象包含一组参数，这些参数是确定更新可用性所必需的：

- updateAvailability — 更新的可用性：
  - UPDATE_AILABILITY_NOT_AVAILABLE — 不需要更新。
  - UPDATE_AILABILITY_AVAILABLE — 需要下载更新或用户设备上已下载更新。
  - UPDATE_AILABILITY_IN_PROGRESS — 更新已在下载或安装已启动。
  - UPDATE_AILABILITY_UNKNOWN — 默认状态。
- installStatus — 如果用户当前正在安装更新，则为更新安装状态：
  - INSTALL_STATUS_DOWNLOADED — 已下载。
  - INSTALL_STATUS_DOWNLOADING — 正在下载。
  - INSTALL_STATUS_FAILED — 错误。
  - INSTALL_STATUS_INSTALLING — 正在安装。
  - INSTALL_STATUS_PENDING — 等待中。
  - INSTALL_STATUS_UNKNOWN — 默认状态。

只有当 updateAvailability 字段包含 UPDATE_AILABILITY_AVAILABLE值时，才能启动更新下载。

方法可能返回错误。可能的错误

下载更新

确认更新可用后，您可以请求用户下载更新，但在此之前，需要使用listener()方法启动更新下载状态监听器。

```
RustoreUpdateClient.listener((value) {  ``print( "listener installStatus ${value.installStatus}" );  ``print( "listener bytesDownloaded ${value.bytesDownloaded}" );  ``print( "listener totalBytesToDownload ${value.totalBytesToDownload}" );  ``print( "listener installErrorCode ${value.installErrorCode}" );  ``if (value.installStatus == INSTALL_STATUS_DOWNLOADED) {    ``//你可以在这里调用 complete()方法  ``}});
```

State对象描述了更新下载的当前状态。对象包含：

- installStatus — 如果用户当前正在安装更新，则为更新安装状态：
  - INSTALL_STATUS_DOWNLOADED — 已下载。
  - INSTALL_STATUS_DOWNLOADING — 正在下载。
  - INSTALL_STATUS_FAILED — 错误。
  - INSTALL_STATUS_INSTALLING — 正在安装。
  - INSTALL_STATUS_PENDING — 等待中。
  - INSTALL_STATUS_UNKNOWN — 默认状态。
- bytesDownloaded — 已下载的字节数。
- totalBytesToDownload — 需要下载的总字节数。
- installErrorCode — 下载过程中的错误代码。

更多可能的错误可以在“可能的错误”部分找到。

下载与RuStore UI

要启动应用程序更新的下载，请调用download()方法。

```
RustoreUpdateClient.download().then((value) {  ``print( "download code ${value.code}" );}).catchError((err) {  ``print( "download err ${err}" );});
```

如果用户确认了下载更新，则 value.code = ACTIVITY_RESULT_OK, 如果拒绝，则 value.code = ACTIVITY_RESULT_CANCELED。

调用方法后，您可以在监听器中跟踪更新下载的状态。如果在监听器中接收到INSTALL_STATUS_DOWNLOADED状态，您可以调用更新安装方法。建议通知用户更新安装已准备就绪。

方法可能返回错误。可能的错误

强制更新

要启动应用程序强制更新的下载，请调用immediate()方法。

```
<br>RustoreUpdateClient.immediate().then((value) {<br>  print("immediate code ${value.code}");<br>}).catchError((err) {<br>  print("immediate err ${err}");<br>});<br>
```

如果用户确认了下载更新，则 value.code = ACTIVITY_RESULT_OK, 如果拒绝，则 value.code = ACTIVITY_RESULT_CANCELED。

调用方法后，您可以在监听器中跟踪更新下载的状态。如果在监听器中接收到INSTALL_STATUS_DOWNLOADED状态，您可以调用更新安装方法。建议通知用户更新安装已准备就绪。

方法可能返回错误。更多错误列表可以在“可能的错误”部分找到。

无界面更新

要启动应用程序更新的无界面RuStore下载，请调用silent()方法。

```
<br>RustoreUpdateClient.silent().then((value) {<br>  print("silent code ${value.code}");<br>}).catchError((err) {<br>  print("silent err ${err}");<br>});<br>
```

调用方法后，您可以在监听器中跟踪更新下载的状态。如果在监听器中接收到INSTALL_STATUS_DOWNLOADED状态，您可以调用更新安装方法。建议通知用户更新安装已准备就绪。

方法可能返回错误。更多错误列表可以在“可能的错误”部分找到。

更新安装

下载apk更新文件完成后，您可以启动更新的安装。要启动更新的安装，请调用complete()方法。

```
RustoreUpdateClient.complete().catchError((err) {  ``print( "complete err ${err}" );});
```

更新通过android原生工具进行。如果更新成功，应用程序将关闭。

在更新阶段可能会出现错误。可能的错误

可能的错误

所有插件中的错误都以常量形式实现。常量的描述在const.dart文件中。

如果您收到onFailure响应，则不建议自行向用户显示错误。显示错误可能会对用户体验产生负面影响。

可能的错误列表：

- UPDATE_ERROR_DOWNLOAD — 下载错误。
- UPDATE_ERROR_BLOCKED — 系统阻止安装。
- UPDATE_ERROR_INVALID_APK — 更新APK无效。

- UPDATE_ERROR_CONFLICT — 与当前应用程序版本冲突。
- UPDATE_ERROR_STORAGE — 设备存储空间不足。
- UPDATE_ERROR_INCOMPATIBLE — 与设备不兼容。
- UPDATE_ERROR_APP_NOT_OWNED — 应用程序未购买。
- UPDATE_ERROR_INTERNAL_ERROR — 内部错误。
- UPDATE_ERROR_ABORTED — 用户拒绝安装更新。
- UPDATE_ERROR_APK_NOT_FOUND — 未找到用于启动安装的APK。
- UPDATE_ERROR_EXTERNAL_SOURCE_DENIED — 禁止启动更新。例如，在第一个方法中返回更新不可用的响应，但用户调用第二个方法。

SDK更新的变更历史

SDK版本 0.0.3

- 内部更新

SDK版本 0.0.2

- 内部更新

SDK版本 0.0.1

- 实现了更新功能

Unreal

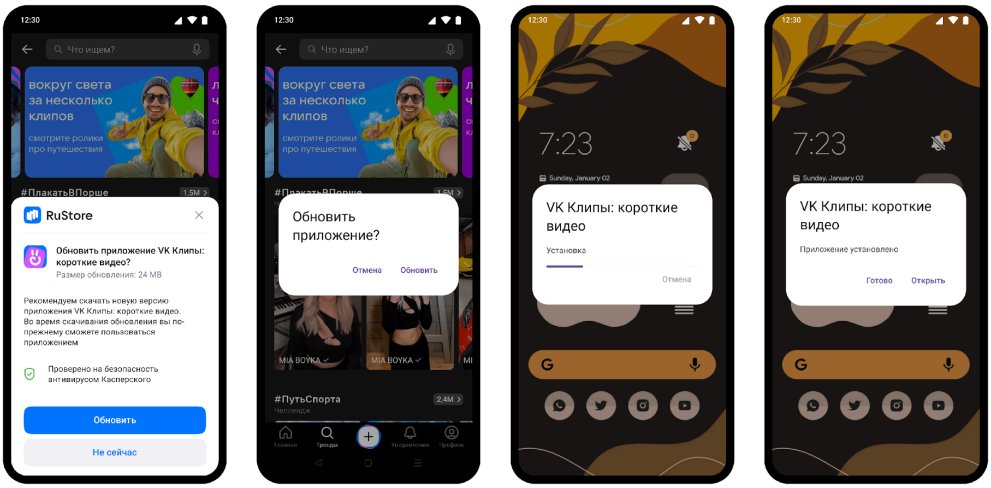
概述

插件帮助保持用户设备上您的应用程序的最新版本。

当用户维持应用程序的最新状态时，他们可以体验新功能并利用性能改进和错误修复。

您可以使用RuStore In-app updates 来显示应用程序更新过程，该过程支持后台下载和安装更新以及状态控制。例如，您可以提醒用户您的应用程序有新版本并建议进行更新。用户在更新下载期间可以继续使用您的应用程序。

用户流程示例



插件正确运作的条件

RuStore In-app updates 运行需要满足以下条件：

1. Android操作系统版本7.0或更高。
2. 用户设备上必须安装了RuStore。
3. 用户设备上的RuStoreApp版本必须是最新的。
4. 用户必须在 RuStore 中获得授权。
5. RuStore应用程序必须被授权安装应用程序。

在项目中集成

要进行集成，请执行以下步骤：

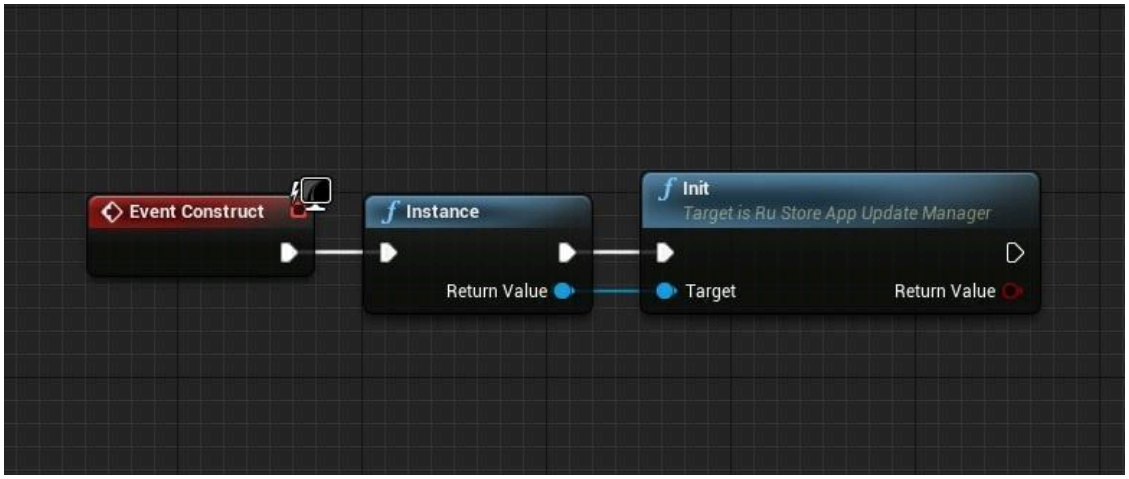
1. 从RuStore的官方gitflic仓库复制*'Plugins'*文件夹的内容到您项目中的*'Plugins'*文件夹。重启Unreal Engine，在插件列表中 (Edit → Plugins → Project → Mobile)，选择“RuStoreAppUpdate”和“RuStoreCore”插件。
2. 在 *“YourProject.Build.cs”* 文件中，将“RuStoreCore”和“RuStoreAppUpdate”模块添加到PublicDependencyModuleNames列表中。
3. 在项目设置中 (Edit → Project Settings → Android) 设置 Minimum SDK Version 为不低于 24, Target SDK Version 为不低于31。

创建更新管理器

在调用库方法之前，需要创建一个更新管理器。

```
URuStoreAppUpdateManager::Instance()->Init();
```

所有与管理器端的操作也可以通过Blueprints进行。初始化的示例：



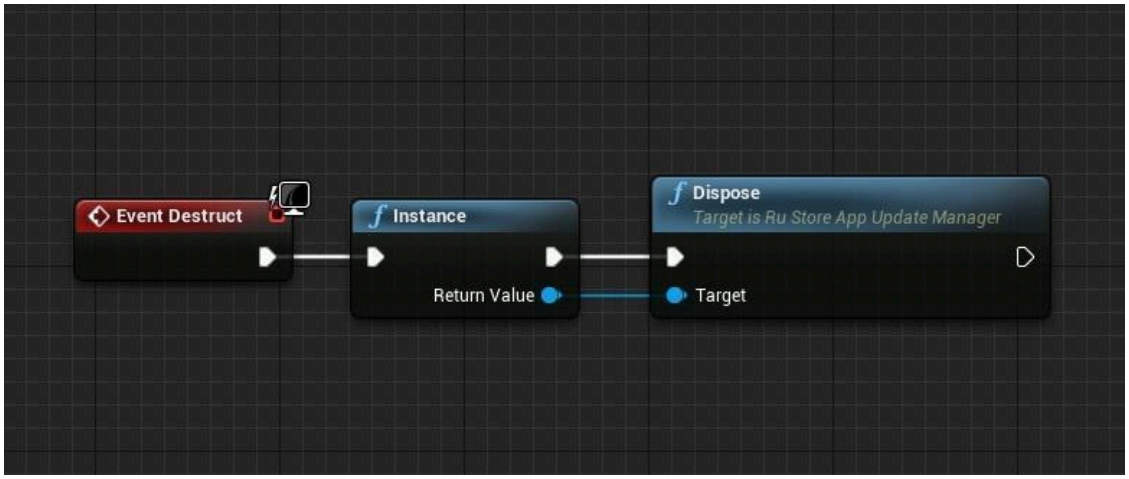
调用Init()将对象绑定到场景的根部，如果之后不再计划使用该对象，为了释放内存，需要执行Dispose()方法。

调用 Dispose() 方法会将对象从根部解绑，并安全地完成所有已发送的请求。

反初始化

```
URuStoreAppUpdateManager::Instance()->Dispose();
```

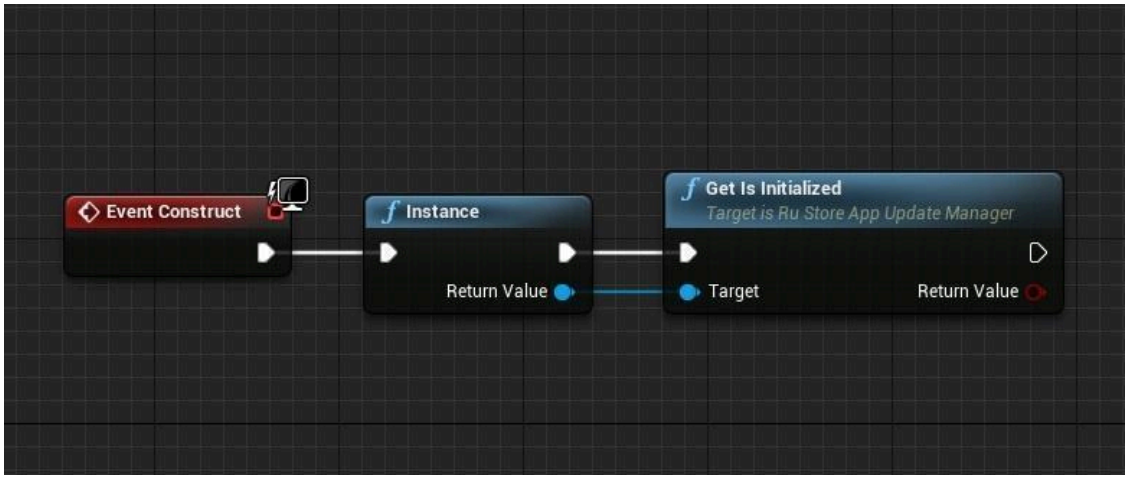
Blueprint实现：



如果您需要检查库的初始化情况，请使用URuStoreBillingClient::Instance()->getIsInitialized()属性。如果库已初始化，其值为 true；如果尚未调用 Init，则为 false。

检查初始化情况

```
URuStoreAppUpdateManager::Instance()->getIsInitialized();
```



检查更新的可用性

在请求更新之前，请检查您的应用是否有更新可用。要检查更新的可用性，请调用 GetAppUpdateInfo() 方法。调用此方法时，将检查以下条件：

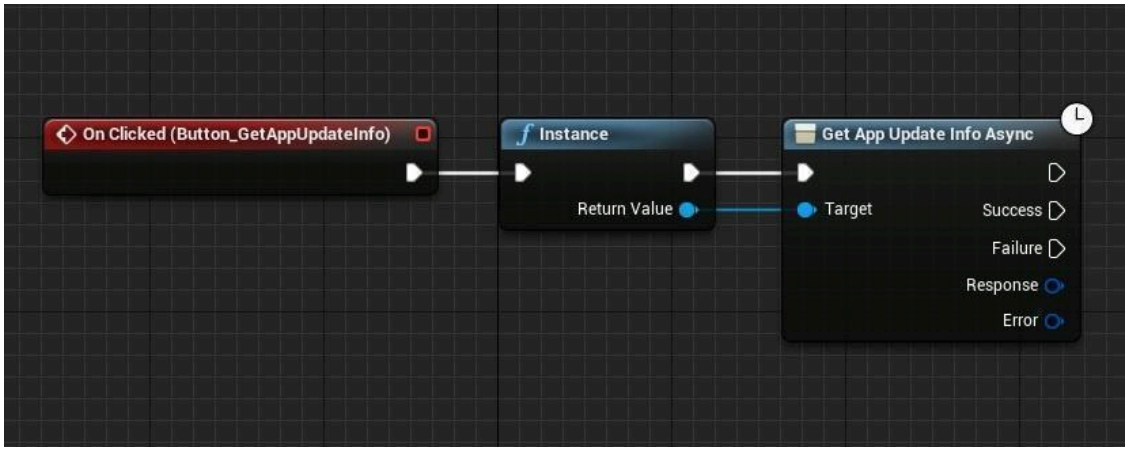
1. 用户设备上必须安装了RuStore。
2. 用户设备上的RuStoreApp版本必须是最新的。
3. 用户必须在 RuStore 中获得授权。
4. 用户和应用程序不应在RuStore中被封锁。
5. RuStore应用程序必须被授权安装应用程序。

在响应此方法时，您将获得一个 AppUpdateInfo 对象，其中包含有关更新需求的信息。请提前请求此信息，以便在用户方便的时间无延迟地请求用户启动更新下载。

调用方法 getAppUpdateInfo() 示例

```
long requestId = GetAppUpdateInfo(        ``[]( long requestId,
TSharedPtr<FURuStoreAppUpdateInfo, ESPMode::ThreadSafe> response)
{                ``// Process response        ``},        ``[]( long requestId,
TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) {                ``// Process
error        ``});
```

Blueprint实现：



Success回调返回AppUpdateInfo结构体，在Response参数中：

AppUpdateInfo 对象包含一组参数，这些参数是确定更新可用性所必需的：

- updateAvailability — 更新的可用性：
  - UPDATE_NOT_AVAILABLE - 不需要更新。
  - UPDATE_AVAILABLE - 需要下载更新或用户设备上已下载更新。
  - DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS - 更新已在下载或安装已启动。
  - UNKNOWN - 默认状态。
- installStatus - 如果用户当前正在安装更新，则为更新安装状态：
  - DOWNLOADED - 已下载。
  - DOWNLOADING - 正在下载。
  - FAILED - 错误。
  - INSTALLING - 正在安装。
  - PENDING - 等待中。
  - UNKNOWN - 默认状态。

只有当 updateAvailability 字段包含 UPDATE_AVAILABLE值时，才能启动更新下载。

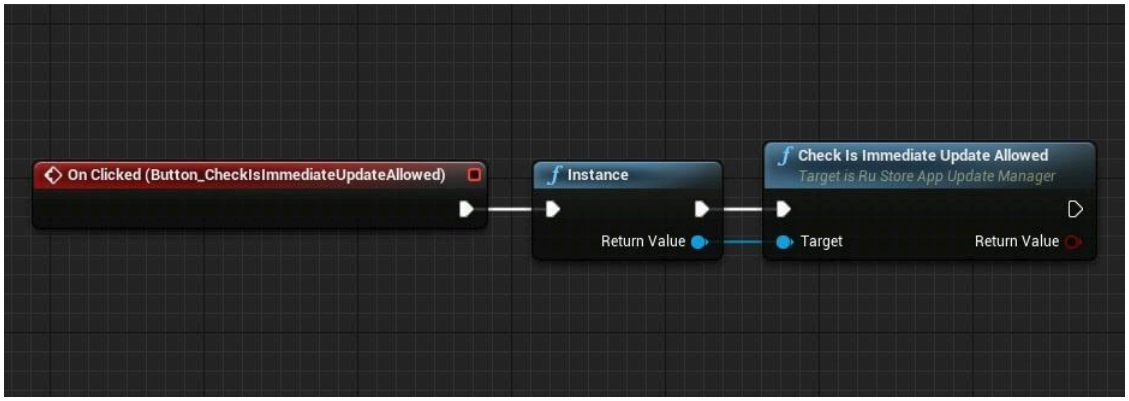
Failure 回调返回 FURuStoreError 结构体，包含错误信息，在Error参数中。更多信息可以在“可能的错误”部分找到。

获取AppUpdateInfo后，可以使用CheckIsImmediateUpdateAllowed()方法检查强制更新的可用性。

调用方法 CheckIsImmediateUpdateAllowed() 示例

```
bool available =
URuStoreAppUpdateManager::Instance()->CheckIsImmediateUpdateAllowed();
```

Blueprint实现：



下载更新

要启动应用程序更新的下载，请调用StartUpdateFlow()方法。

可用的三种更新流程：

EURuStoreAppUpdateOptions::DELAYED - 延迟更新。

EURuStoreAppUpdateOptions::SILENT - 无界面更新。

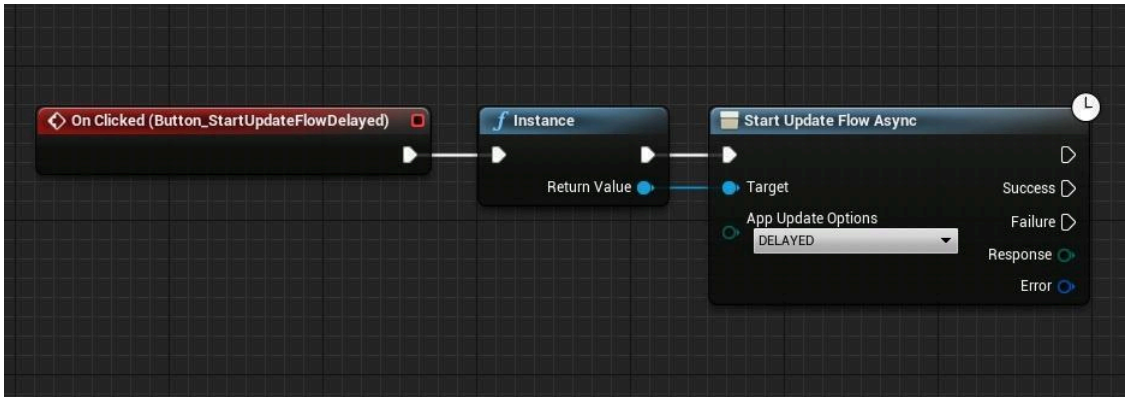
EURuStoreAppUpdateOptions::IMMEDIATE - 强制更新。

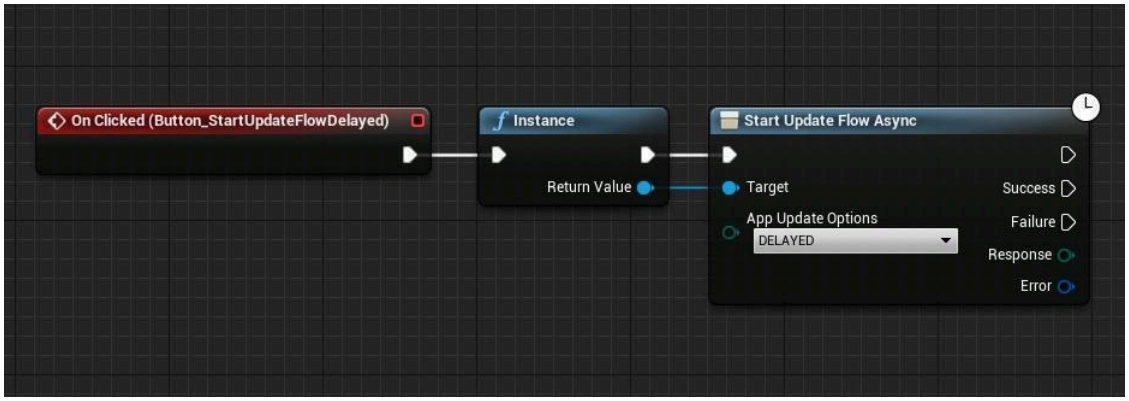
要重新调用StartUpdateFlow()方法，请再次使用GetAppUpdateInfo()方法请求AppUpdateInfo。

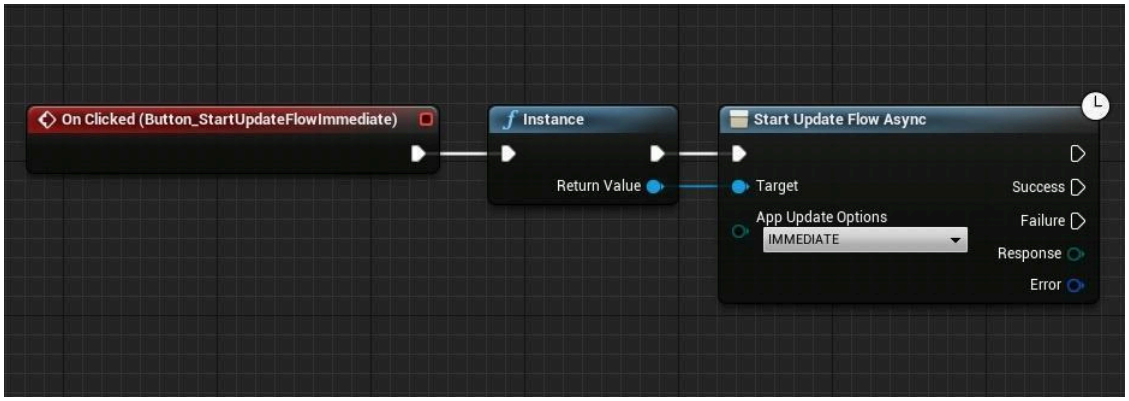
调用方法 startUpdateFlow() 示例

```
EURuStoreAppUpdateOptions appUpdateOptions = EURuStoreAppUpdateOptions::DELAYED;long requestId = StartUpdateFlow(        ``appUpdateOptions,        ``[]( long requestId, EURuStoreUpdateFlowResult response) {                ``// Process response        ``},        ``[]( long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) {                ``// Process error        ``});
```

Blueprint实现：







Failure 回调返回 FURuStoreError 结构体，包含错误信息，在Error参数中。更多信息可以在“可能的错误”部分找到。

Success回调返回EURuStoreUpdateFlowResult值，在Response参数中：

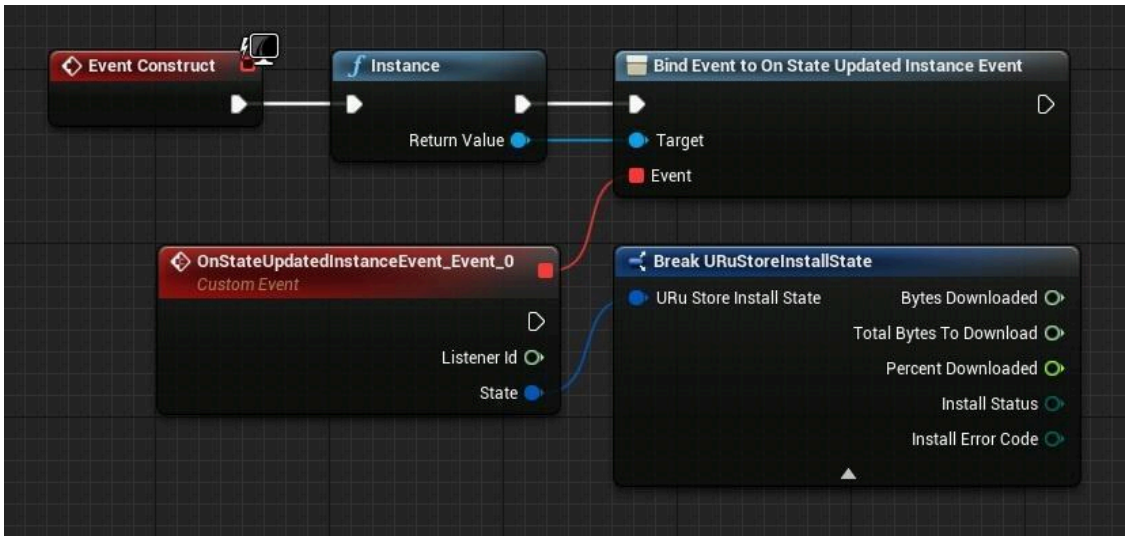
- EURuStoreUpdateFlowResult::RESULT_OK (-1) - 更新完成，代码可能未被接收，因为应用程序在更新时会结束。
- EURuStoreUpdateFlowResult::RESULT_CANCELED (0) - 流程被用户中断或发生错误。当接收到此代码时，通常应终止应用程序。
- EURuStoreUpdateFlowResult::ACTIVITY_NOT_FOUND (2) — RuStore 未安装，或安装的版本不支持强制更新（RuStore versionCode < 191）

调用 StartUpdateFlow() 方法后，您可以在 OnStateUpdatedInstanceEvent事件中跟踪更新下载的状态。

OnStateUpdatedInstanceEvent事件

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FRuStoreOnStateUpdatedInstanceDelegate, int64, listenerId, FURuStoreInstallState, state);UPROPERTY(BlueprintAssignable, Category = "RuStore AppUpdate Manager")FRuStoreOnStateUpdatedInstanceDelegate OnStateUpdatedInstanceEvent;
```

Blueprint实现：



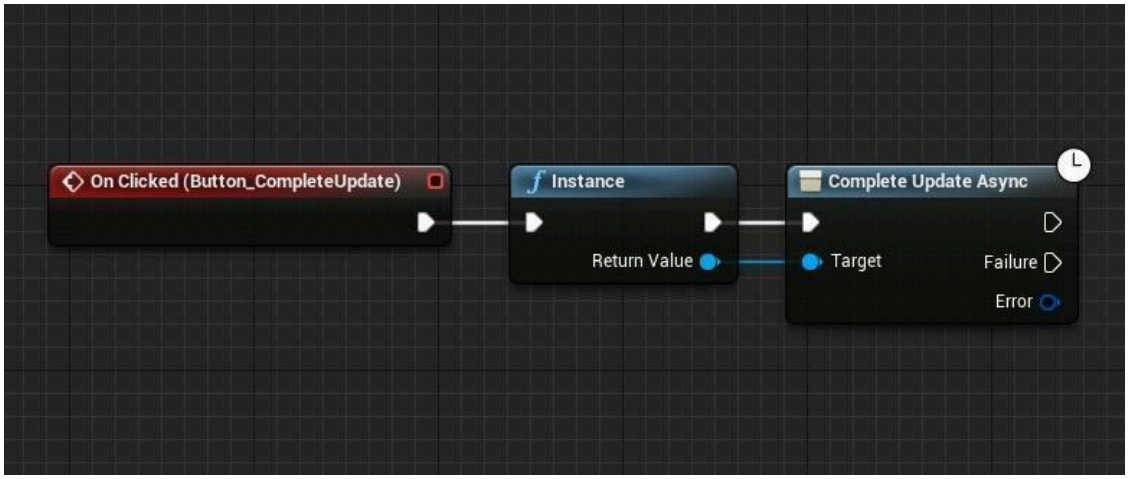
在installStatus字段收到EURuStoreInstallStatus::DOWNLOADED状态后，可以调用更新安装方法。

更新安装

下载apk更新文件完成后，您可以启动更新的安装。要启动更新的安装，请调用CompleteUpdate()方法。

调用方法 CompleteUpdate() 示例

```
requestId = CompleteUpdate(        ``[]( long requestId, TSharedPtr<FURuStoreError, ESPMode::ThreadSafe> error) {                ``// Process error        ``});
```



更新通过android原生工具进行。如果更新成功，应用程序将关闭。

在更新阶段可能会出现错误。您可以在“可能的错误”部分详细了解它们。

可能的错误

如果您收到Failure响应，则不建议自行向用户显示错误。显示错误可能会对用户体验产生负面影响。

错误结构体

```
USTRUCT(BlueprintType)struct RUSTORECORE_API
FURuStoreRuStoreError{ ``GENERATED_USTRUCT_BODY() ``FURuStoreRuStoreEr
ror() ``{ ``name = ”” ; ``description = ””
; ``} ``UPROPERTY(BlueprintReadOnly) ``FString
name; ``UPROPERTY(BlueprintReadOnly) ``FString description;};
```

可能的错误列表：

- RuStoreNotInstalledException — 用户设备上未安装RuStore。
- RuStoreOutdatedException — 用户设备上的RuStore不支持应用程序的更新。
- RuStoreUserUnauthorizedException — 用户未在RuStore中授权。
- RuStoreException(message: String) — RuStore的基本错误，其他错误均继承自此。
- RuStoreInstallException - 下载和安装错误。RuStoreInstallException错误代码：
  - ERROR_UNKNOWN - 未知错误。
  - ERROR_DOWNLOAD - 下载错误。
  - ERROR_BLOCKED - 系统阻止安装。
  - ERROR_INVALID_APK - 更新APK无效。
  - ERROR_CONFLICT - 与当前应用程序版本冲突。

- ERROR_STORAGE - 设备存储空间不足。
- ERROR_INCOMPATIBLE - 与设备不兼容。
- ERROR_APP_NOT_OWNED - 应用程序未购买。
- ERROR_INTERNAL_ERROR - 内部错误。
- ERROR_ABORTED - 用户拒绝安装更新。
- ERROR_APK_NOT_FOUND - 未找到用于启动安装的apk。
- ERROR_EXTERNAL_SOURCE_DENIED - 禁止启动更新。例如，在第一个方法中返回更新不可用的响应，但用户调用第二个方法。
- ERROR_ACTIVITY_SEND_INTENT - 发送intent以开启活动的错误。
- ERROR_ACTIVITY_UNKNOWN - 开启活动的未知错误。

变更记录

插件版本1.0

- 添加了 blueprint 方法的异步选项，取代对事件的订阅。
- 在枚举EURuStoreInstallErrorCode中添加了错误代码。
- 在枚举EURuStoreUpdateFlowResult中添加了状态代码。
- RuStoreAppUpdate_UPL_Android.xml不再向GameActivity和SplashActivity添加android:exported属性。
- 错误修复。

插件版本0.2

- 插件内部更新。

应用程序更新依赖项列表toc：true

RuStore Geo

RuStore Geo —基于 OpenStreetMap 开放数据的地图产品和高负载服务。

RuStore Geo 功能在 RuStore 的使用条件

下列使用 RuStore Geo 功能在 RuStore 的条件（以下简称“**条件**”）规定了您（以下简称“**您**”，“**开发者**”）使用 RuStore Geo 功能的服务和工具的一般条件，包括可能由 VK 有限责任公司（以下简称"**公司**"）提供的用于移动或其他平台的软件，工具以及公司提供的任何其他应用程序、服务、软件、组件、RuStore Geo 功能（统称为"**RuStore Geo**"）。

连接并使用 RuStore Geo 功能时，您明确同意以下条件。

1。公司根据 “OpenStreetMap” 数据库的开放许可证运营 RuStore Geo。OpenStreetMap 许可证的使用条件可在此地址获得http://www.openstreetmap.org/copyright/en。

2。公司不对OpenStreetMap数据库中地图上的领土归属（对象名称、边界位置、其他描述）做出决定。如果用户对任何领土的归属有疑问，相关信息可以在以下链接找到：https://wiki.osmfoundation.org/w/images/d/d8/DisputedTerritoriesInformation.pdf
。

3。本公司现在授予开发者使用由公司提供的软件（包括任何移动应用程序）的访问权限（以下简称“**软件**”），根据本条款的规定。

4。公司保证提供访问 RuStore Geo 的服务每天24小时（24/7）每周，并且可用性水平不低于99,9%。

5。公司保留权利在包括工作日在内的任何时间中断RuStore Geo的运作，以进行必要的预防性维护工作。这种情况将不被视为服务提供或RuStore Geo可用性的中断。

6。如RuStore Geo不可用，公司不承担任何责任，如果这是由于：1）开发者违反了这些条款；2）开发者的其他行为；3）不可抗力（不可抗力情况）；4）根据俄罗斯联邦现行法律，根据国家或市政机构的要求暂停RuStore Geo的运作。

7。RuStore Geo的技术支持是通过开发者向RuStore支持服务发送消息/请求来实现的。

8。该公司及其许可方拥有关于RuStore Geo的所有知识产权以及公司的专有数据库，包括但不限于RuStore Geo的组件和算法，以及对RuStore Geo服务器复杂体的访问，除了根据OpenStreetMap许可证授予公司的知识产权外。

9. 开发者不得更改、发布、传递、展示、参与传播或销售、创造衍生作品或以任何方式使用RuStore Geo的内容或其任何部分。除非版权法明确允许，否则开发者不得复制、分发、发布、展示或将公司拥有的任何材料用于商业目的，未经公司明确许可。除非版权法明确允许，否则开发者不得复制、分发、发布、展示或将公司拥有的任何材料用于商业目的，未经公司明确许可。

10. 这些条件不授予开发者任何直接或间接的权利，使用任何公司的商标、服务标志、名称或徽标，除非开发者在其产品内部放置并使用RuStore Geo或其部分功能，在这种情况下，必须指明公司/RuStore Geo的徽标、RuStore Geo版权信息以及链接 https://www.rustore.ru。

11. 为了使公司能够向开发者提供使用RuStore Geo的机会，仅限于放置开发者内容的目的，开发者特此授予公司非独占的、全球性的、免费的、完全支付的、可转让和可再授权的权利和许可，以使用、复制、缓存、发布、展示、分发、修改、创造衍生作品和存储开发者内容在RuStore Geo的范围内。这一权利和许可允许公司仅在RuStore Geo功能内放置和复制开发者的内容。用户特此保证、确认并同意，他有权向公司提供这些权利。

12。除了开发者自己的内容以外，展示在RuStore Geo上或通过RuStore Geo可获取的所有内容，包括文本、图片、地图、软件或源代码，均为公司和/或第三方所有，并受到俄罗斯联邦现行法律和国际知识产权法的保护。出现在RuStore Geo上或与之相关的产品徽标和名称是公司或我们的许可方的财产。开发者不得删除RuStore Geo中的任何产权通知或产品标识标签。

13。除了公司提供并且开发者通过使用RuStore Geo功能获得访问权限的信息外，该信息由其他开发者提供、输入或发布，并且在任何情况下都不受公司的控制、审核或确认。公司有权但无义务检查和监控通过RuStore Geo功能发布的内容，确定其是否符合本条款和RuStore服务的任何其他规则，这些规则可能会不时由公司设立。关于用户内容/信息，公司不控制此类内容/信息，但公司可能使用某些自动计算机系统和过滤器扫描此类内容/消息，以限制通过RuStore Geo功能发送垃圾邮件或其他不希望的商业内容。对于此类内容/消息的内容，发布或发送它的人将承担全部责任。

14。RuStore Geo的功能由公司提供，并以“现状”和没有任何明示或暗示的保证运作，包括但不限于：商业价值和适用于特定目的的隐含保证。

15。公司不保证RuStore Geo及其功能性能将不间断或无误，或者RuStore Geo的运行缺陷将被纠正。

16。公司不保证并且不作任何声明关于使用服务或任何在RuStore Geo上提到的第三方网站的使用结果，就其正确性、准确性、可靠性或任何其他特性而言。

17。如果在开发者的产品中使用RuStore Geo进行驾驶或操作任何交通工具，该等使用RuStore Geo完全由用户自行承担风险。RuStore Geo提供的信息不旨在替代道路上的信息（例如，交通信号灯、道路标志或交警/警察），如果道路上的信息与RuStore Geo提供的信息不一致，用户不应依据RuStore Geo的数据。使用RuStore Geo时，用户必须遵守所有交通法规和规则。

18。开发者应自行对其使用RuStore Geo功能所涉及的行为对第三方负责，包括如果这些行为导致侵犯第三方的权利和合法利益，以及在使用RuStore Geo时遵守法律法规。

19。在使用RuStore Geo功能时，开发者不得：

a）上传、发送、传输或以其他方式发布和/或传播非法、有害、诽谤性、侵犯道德、展示（或宣传）暴力和残酷、侵犯知识产权、宣扬仇恨和/或基于种族、民族、性别、宗教、社会特征的歧视、包含对任何个人或组织的侮辱、包含（或宣传）色情、儿童色情、广告（或宣传）性服务（包括以其他服务的形式）、解释制造、使用或其他使用毒品或其类似物、炸药或其他武器的方法的内容；

b）侵犯第三方权利，包括未成年人，并以任何形式对他们造成伤害；

c）冒充他人或某个组织/社区的代表，除非拥有足够的权利，包括冒充公司员工、论坛版主、网站所有者等，并且不得以任何其他方式非法代表他人，也不得误导用户或公司关于任何主体或对象的属性和特性；

d）上传、发送、传递或以任何其他方式发布和/或分发无权进行此类操作的内容，根据法律或任何合同关系；

e）上传、发送、传递或以任何其他方式发布和/或分发未经特别授权的广告信息、垃圾邮件（包括搜索引擎优化），电子邮件地址列表、"金字塔"方案、多层次（网络）营销（MLM），互联网赚钱系统和电子邮件业务、"幸运信"，以及使用公司服务参与这些活动，或仅使用公司服务将用户重定向到其他域的页面；

f）上传、发送、传递或以任何其他方式发布和/或分发含有病毒或其他计算机代码、文件或旨在破坏、摧毁或限制任何计算机或电信设备或程序功能的程序的任何材料，用于未经授权的访问，以及商业软件产品的序列号和它们的生成程序、登录名、密码及其他获取未经授权的访问付费互联网资源的手段，以及发布上述信息的链接；

g）未经授权收集和存储他人的个人数据；

h）破坏网站、RuStore服务和RuStore Geo功能的正常运作；

i）协助违反本条款所施加的限制和禁令的行为；

j）以其他方式违反俄罗斯联邦法律，包括国际法的规定。

20。开发者无权暂时存储（缓存）RuStore Geo的任何工作成果及/或从中获得的信息，也不得更改对RuStore Geo的查询结果的内容，该查询结果是对收到的查询请求的响应。

21。开发者必须显示公司为使用RuStore Geo功能提供给开发者的所有属性信息（包括品牌、徽标以及版权和商标通知）。

22。公司可以不时更改这些条件，并通知开发者，包括通过在以下地址发布修改后条件的新版本：
https://help.rustore.ru/rustore/for_developers/developer-documentation/RuStore_Geo。开发者负有自行检查条件以了解任何更改的责任。公司所作的条件更改自发布新版本条件的次日起生效。继续使用RuStore Geo的功能，在条件变更后意味着开发者接受并同意这些变更。如果开发者不同意条件的变更，他应该停止使用（关闭）RuStore Geo的功能。

23。关于本条件的所有问题，开发者可以使用电子邮件地址 support@rustore.ru。

版本为2022年12月20日。

概述

要使用RuStore Geo产品，您需要使用访问权限密钥。

支持的语言

我们致力于创建全世界的全功能地图，并尽可能多地支持我们客户感兴趣的语言。目前，不同语言的可用性由地理区域决定。主要支持的语言是俄语、英语和该区域的本地语言。

请求速度限制

默认情况下，所有访问权限密钥都有相同的每秒请求数量限制。如果需要，您可以减少或增加单个密钥的请求速度。要增加请求速度，您需要联系您的RuStore Geo经理。

服务访问的标准速度限制：10 RPS（用于地图显示的10 SPS，用于计算距离矩阵和解决旅行推销员问题的服务 - 10 EPS）。

访问服务

要使用RuStore Geo的产品，您必须使用有效的访问密钥，该密钥能够唯一地识别客户。访问密钥用于在访问地理服务时在您的网络和移动产品中使用，也用于追踪与您项目相关的RuStore Geo产品的请求。

RuStore Geo的所有产品均可通过以下地址访问：https://geo.rustore.ru/ — 要访问所需的服务，您需要在地址中指定其版本、名称、您的访问密钥和请求正文。

向服务发出的请求格式如下：

```
https://geo.rustore.ru /api/[endpoint]?[api_key]&[query]
```

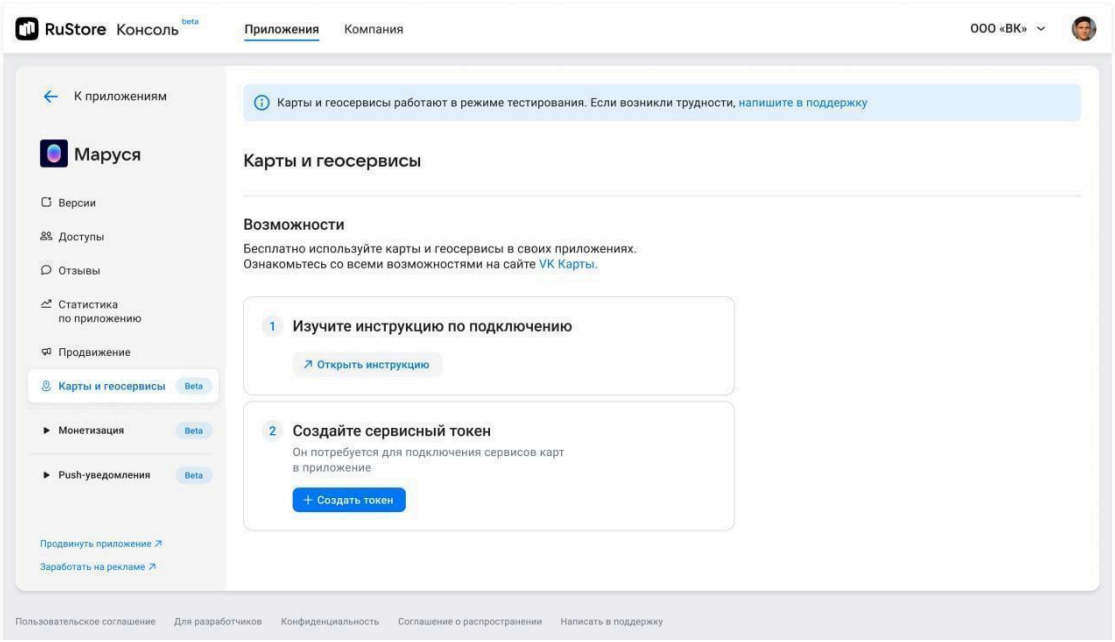
其中：

- endpoint—服务调用点；
- query—带有请求参数的请求正文；
- api_key—您的访问权限密钥。

要获取访问密钥：

1. 打开RuStore 控制台系统。
2. 转到“应用程序”选项卡。
3. 选择应用程序。
4. 找到“地图和地理服务”部分。
5. 点击“创建令牌”。

对于一个应用程序，您只能请求一个用于地图工作的密钥。



将显示创建的密钥。您可以复制并粘贴到您的应用程序中。点击  。

密钥的续订会在应用程序更新发布时自动发生。密钥的有效期为—90个日历日。

密钥的有效期和适用于密钥的限制由软件产品分发协议规定。

如果您在获取密钥时遇到问题，请联系我们。我们会尽力解决问题。

搜索和地理编码服务

兴趣点搜索

兴趣点搜索—一项服务，允许用户搜索兴趣点及其附加信息。

`/places` — 兴趣点搜索服务的调用点以及获取关于它们的附加信息。

请求

请求的必要参数

字段名	格式	描述	示例
` api_key `	hex-string	访问服务	—

字段名	格式	描述	示例
api_key=fa749bace6d8a3b1....			
q	string	搜索请求的正文。搜索请求的正文。在请求中可以使用以下其中一个选项进行搜索：**坐标**格式为lat,lon，其中： - **lat** — 为所寻找点的纬度（度数格式，小数点后使用6位数字）；- **lon** — 为所寻找点的经度（度数格式，使用逗号后6位数字）；**文本字符串**（可能包含所寻找的兴趣点名称或地址）。	
—			
q=rustore.ruq=地铁 q=55.797041,37.537830			

请求的附加参数

字段名	格式	描述	示例
fields	fieldname1,fieldname2,....fieldnameN	选择在响应中显示的字段。可能的值包括：- **name**（默认）— 兴趣点的名称；- **place_details** — 兴趣点的详细信息（目前不支持）；- **address_details**（默认）— 找到的地址的详细信息，包括邮政编码但不包括坐标；- **address** — 找到的兴趣点的地址，以一行文本形式展示；- **pin**（默认）- 找到的对象的坐标（经度和纬度）；- **bbox** — 对象所在区域；- **geometry** — 对象的边界；- **type** — 对象的类型。	
—			
fields=pin,bbox,name			
types	string	目前不支持。搜索的对象类型。默认情况下，搜索涵盖所有兴趣对象类型。目录不断更新。要获取完整且最新的类型	

字段名	格式	描述	示例
		目录，可以使用服务 /v3/places/types	
types			`types=hotel,metro`
lang	2位字符语言代码	答案的语言是可用语言之一。默认为找到对象的区域的语言。	
lang			`lang=en`
location	string	将根据指定坐标进行搜索。坐标格式为 lat, lon，其中：- **lat** — 目标点的纬度,以度为单位(小数点后使用6位数字)；- **lon** — 目标点的经度,以度为单位(小数点后使用6位数字)。搜索结果将按照与指定坐标的距离进行排序。	
location			`location=55.796743,37.537354`
radius	integer	搜索半径（以米为单位），相对于location参数。使用location参数时，radius参数是必需的。如果使用较小的搜索半径，可能找不到对象。	
radius			`radius=500`
limit	integer	响应中对象的数量限制。支持从1到100的值范围。默认值：5。使用较大的值时，获取响应的时间可能会增加	
limit			`limit=10`
isocode	2char	根据ISO 3166-1 alpha-2的两个字母国家代码	
isocode			`isocode=RU`

响应

响应仅以 JSON格式提供，并包含根据请求找到的信息。

JSON格式

字段名	格式	描述	示例
request	string	请求	—

```
"request":
"/v3/places?locati
on=55.7967432,37.5
373542&api_key=int
ernal_&q=prime&fie
lds=name,place_det
ails,address,addre
ss_details,pin,bbo
x,geometry,type"
```

results	list	找到的结果	—

```
″results″:
[        ``{
    ``″address″: ″俄
罗斯，莫斯科，北部行
政区，莫斯科，机场
区，列宁格勒大道，72
4栋
″,            ``″add
ress_details″:
{                ``″
building″: ″72 4栋
″,                ``
″country″: ″俄罗斯
″,                ``
″isocode″:
″RU″,
  ``″locality″: ″莫
斯科
″,                ``
″region″: ″莫斯科
″,                ``
″street″: ″列宁格勒
大道
″,                ``
″subregion″: ″北部行
政区
″,                ``
″suburb″: ″机场区
″            ``},
          ``″bbox″:
```

字段名	格式	描述	示例
null, ``"geometry": { ``"coordinates": [``37.519963, ``55.805396 ``], ``"type": "Point" `}, ``"name": "普赖姆"。 ``"pin": [``37.519963, ``55.805396 ``], ``"type": "food/cafe" `} ``]}			
address	string	完整的找到的对象地址	—
"address":"俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 39 14 号楼"			
address_details	list	关于找到的对象地址的详细信息	—
"address_details": { ``"building": "39 80号楼", ``"country": "俄罗斯", ``"isocode": "RU", ``"locality": "莫斯科", ``			

字段名	格式	描述	示例
″postal_code″: ″125167″, ``″region″: ″莫斯科″, ``″street″: ″列宁格勒大街″, ``″subregion″: ″北行政区″, ``″suburb″: ″霍罗绍夫斯基″}			
pin	list	找到的对象的坐标（经度和纬度）	
—			
"pin": [``37.538851, ``55.796731]			
bbox	list	找到的对象的位置边界，以便在地图上定位	
—			
"bbox": [``37.538253, ``55.796405, ``37.539368, ``55.79694]			
geometry	list	找到的对象的几何形状	
—			
"geometry": { ``"coordinates": [ ``44.795525, ``41.775552 ``], ``"type": "Point"}			
name	string	找到的对象的名称	
—			

字段名	格式	描述	示例
″name″:地铁“机场”站			
type	string	找到的对象的类型。	
— "type": "public_transport/station_train"			

字段描述address_details

字段名	格式	描述	示例
country	string	国家	
— ″country″:“俄罗斯”			
isocode	2char	根据ISO 3166-1 alpha-2的两个字母国家代码	
— "isocode": "RU"			
region	string	州	
— ″region″:“莫斯科”			
subregion	string	区	
— ″subregion″: “北行政区”			
locality	string	居民点	
— ″locality″: “莫斯科”			
sublocality	string	小区或住宅综合体名称	
— ″sublocality″:“机场”			
street	string	街道	
— ″street″:列宁格勒大街			

字段名	格式	描述	示例
building	string	房屋号码、房屋部分或建筑物	—
″building″:″39 80号楼″			
suburb	string	居民点的区域/小区	—
″suburb″:“霍罗绍夫斯基”			
postal_code	string	邮政编码	—
"postal_code": "125167"			

字段描述geometry

字段名	格式	描述	示例
type	string	几何类型：**Point**— 点；**MultiPoint** — 多个点；**Linestring** — 线；**MultiLineString** — 多条线；**Polygon** — 多边形；**MultiPoligon** — 多个多边形。	—
"type": "Polygon"			
coordinates	list	点数组（经度和纬度），描述对象的几何形状	—
"coordinates": [``[100.0, 0.0], [101.0, 0.0], ``[101.0, 1.0], [100.0, 1.0], ``[100.0, 0.0]]			

示例

请求

```
https://geo.rustore.ru/api/places?api_key=<YOUR_API_KEY>&q=prime&fields=name,place_details,address,address_details,pin,bbox,geometry,type&location=55.7967432,37.5373542&limit=1
```

地理编码服务

地理编码器—一种提供直接和反向地理编码的服务。

直接地理编码—将地址转换为地理坐标（纬度和经度）的过程，这些坐标可以用于在地图上放置标记或在屏幕上放置地图。

反向地理编码—将地理坐标（纬度和经度）转换为地址或地址部分（国家、城市、地区等）的过程。

所有地理编码服务都在全球范围内的所有可用语言中运行。

/search — 地址对象直接和反向地理编码服务的统一调用点。

请求

请求的必要参数

字段名	格式	描述	示例
api_key	hex-string	访问服务	— `api_key=fa749bace6d8a3b1....`
q	string	搜索请求的正文。对于反向地理编码 — 坐标格式为 lat,lon，其中:- **lat** — 目标点的纬度（以度为单位，小数点后使用6位数字）;- **lon** — 目标点的经度（以度为单位，小数点后使用6位数字）。对于直接地理编码 — 文本字符串。	— `q=莫斯科 列宁格勒大街 39号80号楼`

q=慕尼黑

```
q=Austria
```

```
q=55.7967432,37.5373542
```

搜索请求的通用附加参数

字段名	格式	描述	示例
lang	2位字符语言代码	响应的语言是可用语言之一。默认为找到对象的区域语言。	—
			`lang=en`
limit	unsigned integer	响应中对象数量的限制。支持从1到100的数值范围。默认值：5。	—
			`limit=10`
fields	fieldname1,fieldname2,….fieldnameN	选择在响应中显示的字段。可能的值包括：address_details（默认） — 关于找到的地址的详细信息，包括邮政编码但不包括坐标；address — 完整的找到的地址，以单行文本形式；pin（默认） — 找到的对象的坐标（经度和纬度）；bbox — 对象的位置区域；geometry — 对象的边界；type — 对象的类型。	—
			`fields=geometry,address_details`
admin_level	admin level	按照行政级别限制响应。可能的值包括：	

字段名	格式	描述	示例
		1 — 国家；2 — 地区；3 — 居民点；4 — 街道；5 — 房屋。默认情况下没有设置限制 — 搜索是针对所有行政级别进行的。	
—			
admim_level=2			

直接地理编码的附加请求参数

字段名	格式	描述	示例
location	string	用于搜索和响应结果排名的参考坐标。坐标格式为 lat, lon，其中：lat — 目标点的纬度，以度为单位。小数点后使用6位数字；lon — 目标点的经度，以度为单位。小数点后使用6位数字。搜索结果将根据距离指定坐标的远近进行排序。	
—			
location=55.7967432,37.5373542			
isocode	2char	根据ISO 3166-1 alpha-2的两个字母国家代码	
—			
isocode=RU			

反向地理编码的附加请求参数

字段名	格式	描述	示例
radius	integer	搜索半径（以米为单位）相对于请求中q参数指定的坐标。使用较小的搜索半径可能导致找不到对象。	

字段名	格式	描述	示例

—

```
radius=300
```

响应

响应仅以 JSON 格式提供。使用为地理编码器 v.3 服务特别开发的 JSON 格式，其中包含了查询到的信息。

JSON格式

字段名	格式	描述	示例
request	string	请求	

—

```
"request":
"/v3/search?api_ke
y=demo_demo_main&l
imit=1&q=55.796668
,37.538871&fields=
address_details,ad
dress,pin,bbox,geo
metry,type",
```

字段名	格式	描述	示例
results	list	找到的结果	

—

```
"results":
[       ``{
    ``"address":"俄
罗斯, 莫斯科, 北行政
区, 俄罗斯, 霍罗绍夫
斯基, 列宁格勒大街,
39 14号楼
",            ``"add
ress_details":
{               ``"
building": "39 14号
楼
",                ``
"country": "俄罗斯
",                ``
"postal_code":
"125167",
      ``"isocode":
"RU",
  ``"locality": "莫
斯科
",                ``
```

字段名 格式 描述 示例

```
"region": "莫斯科
",                  ``
"street": "列宁格勒
大街
",                  ``
"subregion": "北行政
区
",                  ``
"suburb": "霍罗绍夫
斯基
"             ``},
          ``"bbox":
[                 ``3
7.538253,
      ``55.796405,
             ``37.5
39368,
  ``55.79694
    ``],
``"geometry":
{                 ``"
coordinates":
[
  ``[
          ``[

  ``37.538253,

  ``55.796822
                  ``
],
      ``[
                  ``
37.539204,
                  ``
55.796405
              ``],

  ``[
              ``37.5
39368,
              ``55.7
96523
          ``],      ``
[
```

字段名 格式 描述 示例

``37.539105,
``55.796639
``],
``[
``37.539082,
``55.796674
``],
``[
``37.539052,
``55.796705
``],
``[
``37.53901,
``55.796737
``],
``[
``37.538958,
``55.796763
``],
``[
``37.538899,
``55.796784
``],
``[
``37.538835,
``55.796801
``],

字段名	格式	描述	示例
address	string	完整找到的地址	—

```
          ``[

  ``37.538772,

  ``55.796808
                    ``
],
      ``[
                    ``
37.538714,
                    ``
55.79681
             ``],

``[
             ``37.538
416,
             ``55.796
94
      ``],
               ``[

      ``37.538253,

      ``55.796822

  ``]
      ``]
      ``],
      ``"type":
"Polygon"
  ``},              ``
"pin":
[                 ``3
7.538851,
      ``55.796731
         ``],
      ``"type":
"building"          ``
}     ``]
```

```
"address":"俄罗斯,
莫斯科, 北行政区, 莫
斯科, 霍罗绍夫斯基,
```

字段名	格式	描述	示例
列宁格勒大街, 39 14号楼",			
address_details	list	关于找到的地址的详细信息	
—			
"address_details": { ``"building": "39 80号楼", ``"country": "俄罗斯", ``"isocode": "RU", ``"locality": "莫斯科", ``"postal_code": "125167", ``"region": "莫斯科", ``"street": "列宁格勒大街", ``"subregion": "北行政区", ``"suburb": "霍罗绍夫斯基"}			
pin	list	找到的对象的坐标（经度和纬度）	
—			
"pin": [``37.538851, ``55.796731]			
bbox	list	找到的对象的位置边界，以便在地图上定位	
—			
"bbox": [``37.5382			

字段名	格式	描述	示例
53, ``55.79 6405, ``37. 539368, ``5 5.79694]			
geometry	list	找到的对象的几何图形	

—

```
"geometry":
{       ``"coordi
nates":
[         ``[
        ``[
       ``37.538253
,              ``5
5.796822
  ``],
``[             `
`37.539204,
       ``55.796405
          ``],
         ``[
         ``37.5393
68,              `
`55.796523
    ``],
  ``[
 ``37.539105,
        ``55.7966
39           ``],
           ``[
           ``37.53
9082,
 ``55.796674
      ``],
    ``[
   ``37.539052,
          ``55.79
6705            ``
],           ``[
            ``37.
53901,
  ``55.796737
      ``],
     ``[
    ``37.538958,
           ``55.7
96763            `
`],           ``[
             ``37
```

字段名	格式	描述	示例
.538899, ``55.796784 ``], ``[ ``37.538835, ``55 .796801 ``], ` `[`` 37.538772, ``55.796808 ``], ``[``37.53871 4, `` 55.79681 ``], ``[` `37.538416, ``55.79694 ``], ``[ ``37.53825 3, `` 55.796822 ``] `` ] ``], ``"type": "Polygon" ``}			
type	string	对象的类型	—
"type": "building"			

字段描述address_details

字段名	格式	描述	示例
country	string	国家	—
″country″:“俄罗斯”			
isocode	2char	根据ISO 3166-1 alpha-2的两个字母国家代码	—
"isocode": "RU"			
region	string	州	

字段名	格式	描述	示例
—			
″region″: “莫斯科”			
subregion	string	区	
—			
″subregion″: “北行政区”			
locality	string	居民点	
—			
″locality″: “莫斯科”			
sublocality	string	小区或住宅综合体名称	
—			
″sublocality″:“机场”			
street	string	街道	
—			
″street″: 列宁格勒大街			
building	string	房屋号码、房屋部分或建筑物	
—			
″building″:″39 80号楼″			
suburb	string	居民点的区域/小区	
—			
″suburb″:“霍罗绍夫斯基”			
postal_code	string	邮政编码	
—			
"postal_code": "125167"			

字段描述geometry

字段名	格式	描述	示例
type	string	几何类型：- **Point** — 点;- **MultiPoint** — 多个点;-	

字段名	格式	描述	示例
		LineString — 线;- **MultiLineString** — 多条线;- **Polygon** — 多边形;- **MultiPolygon** — 多个多边形。	
—			
"type": "Polygon"			
coordinates	list	点数组（经度和纬度），描述对象的几何形状	
—			

```
"coordinates":
[    ``[100.0,
0.0], [101.0,
0.0],    ``[101.0,
1.0], [100.0,
1.0],    ``[100.0,
0.0]]
```

如果查询没有找到任何内容，响应将如下所示：

```
{        ``"results": [],        ``"request": "/v3/search?limit=1&q=列宁格勒大街
```

示例

直接地理编码

请求

```
https://geo.rustore.ru/api/search?api_key=<YOUR_API_KEY>q=莫斯科%20号列宁格勒%2039%20号14号楼&fields=address_details,address,pin,bbox,geometry,type&limit=1
```

响应

```
{    ``"request": "/search?api_key=demo_demo_main&limit=1&q=莫斯科 列宁格勒 39 14号楼&fields=address_details,address,pin,bbox,geometry,type",    ``"results":
[        ``{            ``"address": "俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 39 14号楼",            ``"address_details":
{                ``"building": "39 14号楼",                ``"country": "俄罗斯
",                ``"postal_code": "125167",                ``"isocode":
"RU",                ``"locality": "莫斯科",                ``"region": "莫斯科
",                ``"street": "列宁格勒大街",                ``"subregion": "北行政区
",                ``"suburb": "霍罗绍夫斯基"            ``},            ``"bbox":
[                ``37.538253,                ``55.796405,                ``37.539368,
                ``55.79694            ``],            ``"geometry":
```

```
{ ``"coordinates":[ ``[ ``[ ``37.538253, ``55.796822 ``], ``[ ``37.539204, ``55.796405 ``], ``[ ``37.539368, ``55.796523 ``], ``[ ``37.539105, ``55.796639 ``], ``[ ``37.539082, ``55.796674 ``], ``[ ``37.539052, ``55.796705 ``], ``[ ``37.53901, ``55.796737 ``], ``[ ``37.538958, ``55.796763 ``], ``[ ``37.538899, ``55.796784 ``], ``[ ``37.538835, ``55.796801 ``], ``[ ``37.538772, ``55.796808 ``], ``[ ``37.538714, ``55.79681 ``], ``[ ``37.538416, ``55.79694 ``], ``[ ``37.538253, ``55.796822 ``] ``] ``] ``], ``"type": "Polygon" ``}, ``"pin": [ ``37.538851, ``55.796731 ``], ``"type": "building" ``} ``]}
```


反向地理编码

请求

```
https://geo.rustore.ru/api/search?api_key=<YOUR_API_KEY>q=55.796668,37.538871&fields=address_details,address,pin,bbox,geometry,type&limit=1
```

响应


```
{ ``"request": "/v3/search?api_key=demo_demo_main&limit=1&q=55.796668,37.538871&fields=address_details,address,pin,bbox,geometry,type", ``"results": [ ``{ ``"address": "俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 39 14号楼", ``"address_details": { ``"building": "39 14号楼", ``"country": "俄罗斯", ``"isocode": "RU", ``"locality": "莫斯科", ``"region": "莫斯科", ``"street": "列宁格勒大街", ``"subregion": "北行政区", ``"suburb": "霍罗绍夫斯基"``}, ``"bbox": [ ``37.538253, ``55.796405, ``37.539368, ``55.79694 ``], ``"geometry":
```

{ ``"coordinates": [ ``[``[ ``37.538253, ``55.796822 ``], ``[ ``37.539204, ``55.796405 ``], ``[ ``37.539368, ``55.796523 ``], ``[ ``37.539105, ``55.796639 ``], ``[ ``37.539082, ``55.796674 ``], ``[ ``37.539052, ``55.796705 ``], ``[ ``37.53901, ``55.796737 ``], ``[ ``37.538958, ``55.796763 ``], ``[ ``37.538899, ``55.796784 ``], ``[ ``37.538835, ``55.796801 ``], ``[ ``37.538772, ``55.796808 ``], ``[ ``37.538714, ``55.79681 ``], ``[ ``37.538416, ``55.79694 ``], ``[ ``37.538253, ``55.796822 ``] ``] ``], ``"type": "Polygon" ``}, ``"pin": [ ``37.538851, ``55.796731 ``], ``"type": "building" ``} ``]}

地址和兴趣点提示器

提示器—一项服务，它在搜索框中逐字符输入时提供地址和/或兴趣点名称的自动完成提示。

/suggest — 在逐字符输入时，用于调用地址和兴趣点名称提示器的点。

请求

请求的必要参数

字段名	格式	描述	示例
 api_key 	hex-string	访问服务	—
api_key=fa749bace6d8a3b1....			
q	string	搜索请求的正文	—

字段名	格式	描述	示例
q=俄罗斯 莫斯科q=莫斯科 列宁格勒 c q=莫斯科 列宁格勒大街q=莫斯科 列宁大街 1号 q=地铁“机场”站			

请求的附加参数

字段名	格式	描述	示例
fields	fieldname1,fieldname2,…. fieldnameN	选择在响应中显示的字段。可能的值包括：**name**（默认）— 找到的对象；**address_details** — 关于找到的地址的详细信息，包括邮政编码但不包含坐标；**address**（默认）— 完整的找到的地址，以单行文本形式；**ref** — 找到的对象的 ID；**type**（默认）— 对象的类型。	
—			
fields=name			
types	string	执行搜索的对象类型：**address**（默认）— 地址；**place** — 兴趣点名称。	
—			
type=address,place			
lang	2位字符语言代码	响应的语言是可用语言之一。默认为找到对象的区域语言。	
—			
lang=en			
location	string	根据搜索的相关坐标进行搜索和响应结果的排名。坐标格式为 lat, lon，其中：	

字段名	格式	描述	示例
		lat — 搜索点的纬度，以度为单位。小数点后使用6位数字；lon — 目标点的经度，以度为单位。小数点后使用6位数字。搜索结果将根据距离指定坐标的远近进行排序。	—
location=55.796743,37.537354			
radius	integer	搜索半径（以米为单位），相对于location参数。使用location参数时，radius参数是必需的。如果使用较小的搜索半径，可能找不到对象。	—
radius=500			
admin_level	rank number	按照行政级别限制响应。与types=address 一起使用。可能的值包括：1 — 国家；2 — 地区；3 — 居民点；4 — 街道；5 — 房屋。默认情况下没有设置限制 — 搜索是针对所有行政级别进行的。	—
admin_level=3			
limit	integer	响应中对象的数量限制。支持从1到100的值范围。默认值：5。使用较大的值时，	

字段名	格式	描述	示例
		获取响应的时间可能会增加。	

—

limit=10

响应

仅以 JSON 格式提供响应。所使用的 JSON 格式是为提示器 v.3 服务专门开发的，并包含了查询所找到的信息。

JSON格式

字段名	格式	描述	示例
request	**string**	请求	

—


```
"request":
"/v3/suggest?limit=3
&q=莫斯科 列宁格勒
39&fields=name,addre
ss_details,address,r
ef,type"
```


results	**list**	找到的结果	

—


```
"results":
[       ``{
    ``"address": "俄
罗斯, 莫斯科, 北行政
区, 莫斯科, 霍罗绍夫
斯基, 列宁格勒大街,
39 79号楼
",          ``"add
ress_details":
{               ``"
building": "39
 c 79",
  ``"country": "俄罗
斯
",                ``
,
"isocode":
"RU",
  ``"locality": "莫
斯科
",                ``
,
"region": "莫斯科
```

字段名	格式	描述	示例
″, ″street″: ″列宁格勒大街 ″, ″subregion″: ″北行政区 ″, ″suburb″: ″霍罗绍夫斯基 ″ ``}, ``″ref″: ″1000000C4D63818″, ``″type″: ″building″ ``″name″: ″列宁格勒大街, 39 79号楼 ″, ``} ``]			
address — ″address″: ″俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 39 79号楼″,	string	完整找到的地址	
address_details — ″address_details″: { ``″building″: ″39 79号楼 ″, ``″country″: ″俄罗斯 ″, ``″isocode″: ″RU″, ``″locality″: ″莫斯科 ″, ``″region″: ″莫斯科 ″, ``″street″: ″列宁格勒大街 ″, ``	list	关于找到的地址的详细信息	

字段名	格式	描述	示例
″subregion″: ″北行政区 ″, ``″suburb″: ″霍罗绍夫斯基″ ``}			
ref	hex	ID找到的对象的 ID 。对象 ID 不是固定的，可能会发生变化。	
—			
"ref": "1000000C4D63818"			
type	string	找到的对象的类型。	
—			
"type": "building"			
name	string	找到的对象的名称	
—			
″name″:″列宁格勒大街, 39 79号楼″			

字段描述address_details

字段名	格式	描述	示例
country	string	国家	
—			
″country″: “俄罗斯”			
isocode	2char	根据ISO 3166-1 alpha-2的两个字母国家代码	
—			
"isocode": "RU"			
region	string	州	
—			
″region″:“莫斯科”			
subregion	string	区	
—			
″subregion″:“北行政区”			

字段名	格式	描述	示例
locality	string	居民点	— "locality": “莫斯科”
sublocality	string	小区或住宅综合体名称	— "sublocality": “机场”
street	string	街道	— "street":列宁格勒大街
building	string	房屋号码、房屋部分或建筑物	— "building": "39 80号楼"
suburb	string	居民点的区域/小区	— "suburb": “霍罗绍夫斯基”
postal_code	string	邮政编码	— "postal_code": "125167"

示例

请求

```
https://geo.rustore.ru/api/suggest?api_key=<YOUR_API_KEY>&limit=3&q=莫斯科%20列宁格勒%2039
```

响应

```
{    ``"request": "/suggest?limit=2&q=莫斯科 列宁格勒 39&api_key=demo_demo_main",    ``"results": [        ``{            ``"address": "俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 39 79号楼",            ``"name": "列宁格勒大街, 39 79号楼",            ``"type": "building"        ``},        ``{            ``"address": "俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 39 80号楼",            ``"name": "列宁格勒大街, 39
```

80号楼", ``"type": "building" ``}, ``{ ``"address": "俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 39 3号楼", ``"name": "CSKA通用体育综合体。戈梅利斯基 “, ``"type": "building" ``} ``]}

地图显示服务

地图显示服务允许产品中嵌入交互式和静态地图。

交互式地图—这是一个服务，它允许将地图嵌入到网站或应用程序中，用户可以改变地图的缩放级别、视角和显示位置，并且可以使用JS在地图上添加各种元素：图钉、线条、高亮必要区域等。

静态地图—这是一个服务，它允许获取带有图钉的地图图像，格式为PNG或JPG。

服务

调用点	描述
	交互式地图
/staticmap	静态地图

交互式地图

要显示交互式地图，需要以下组件：

- 显示数据（瓦片）；
- 显示样式；
- 在网页上显示地图的程序代码。

显示数据由方形马赛克片 — 瓦片组成。整个世界的瓦片集合形成一个金字塔结构。在金字塔的零级（缩放级别）上有一个瓦片。在每个后续级别（缩放级别）上，瓦片的数量是4n，其中n—缩放级别的编号。每个瓦片包含几何和属性信息。

地图显示样式是一个包含数据来源和如何绘制各种元素描述的json对象。

要在您的项目中嵌入地图，需要在页面上添加一个布局元素来显示地图。

创建交互式地图

为了简单地显示地图，需要在页面上添加来自 SDK 的 JS & CSS 文件，并添加地图初始化代码，同时不要忘记指定有效的 API 访问密钥。

初始化时需要指定所需的样式：

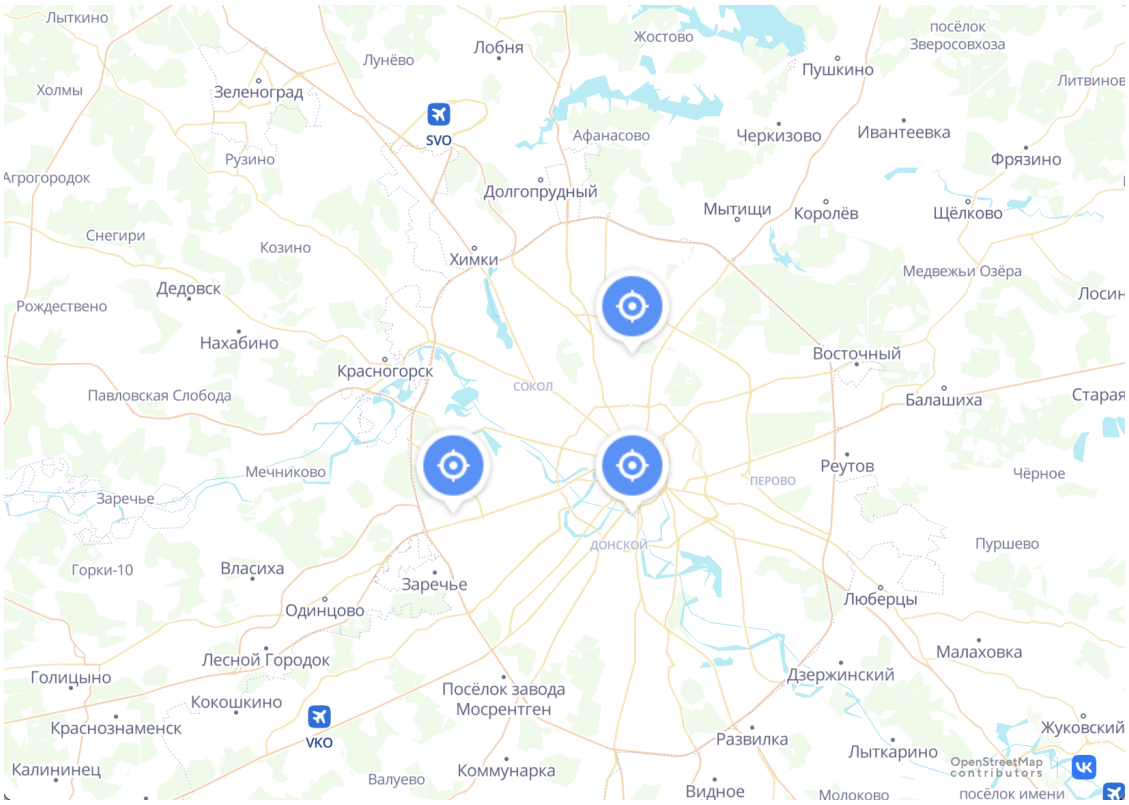
- light_style.json — mmr://api/styles/light_style.json
- mapsme_style.json — mmr://api/styles/mapsme_style.json
- main_style.json — mmr://api/styles/main_style.json

在地图上添加对象

要在地图上添加自己的对象，首先需要描述它们的坐标，通常使用 GeoJson 格式进行描述。

添加带图钉的点

为了简单地添加带有图标的单独图层，只需执行以下描述的操作。



创建一个点列表以添加到地图上：

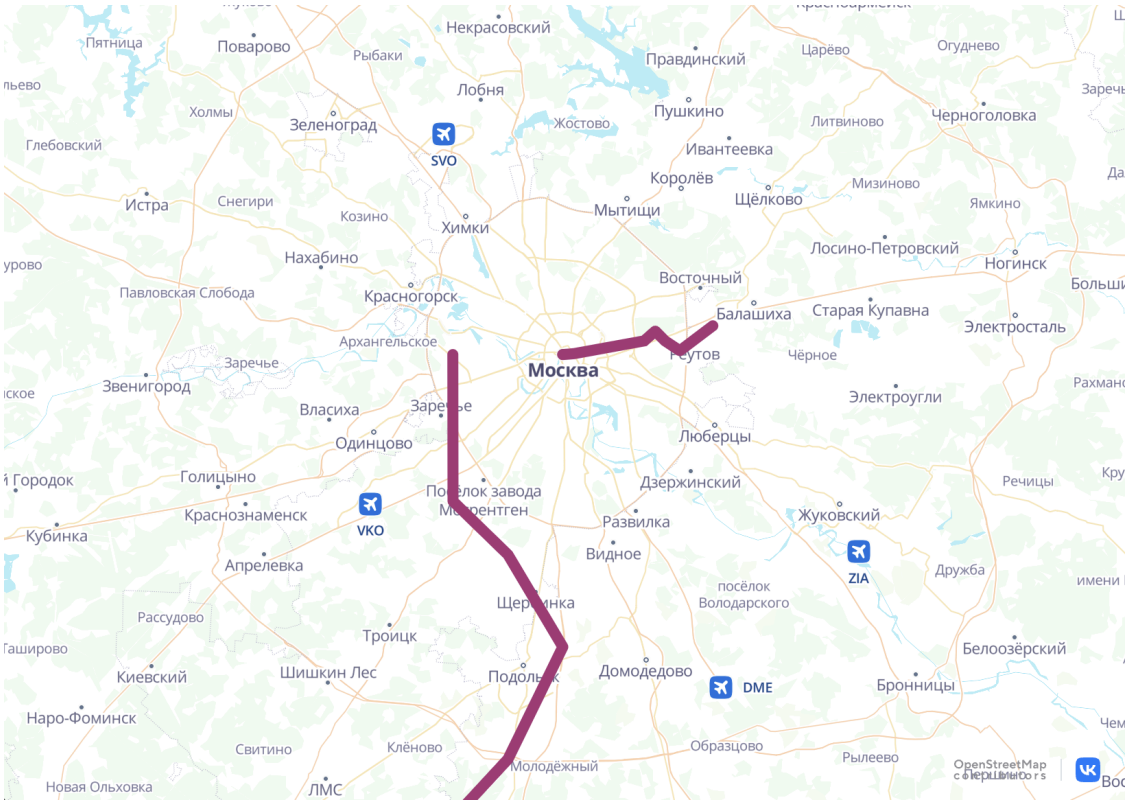
```
let userPointData = {    ``"type": "FeatureCollection",    ``"features":
[{       ``"type": "Feature",       ``"geometry": {           ``"type":
"Point",           ``"coordinates": [37.6165, 55.7505]       ``}    ``},
{       ``"type": "Feature",       ``"geometry": {           ``"type":
"Point",           ``"coordinates": [37.4165, 55.7505]       ``}    ``},
{       ``"type": "Feature",       ``"geometry": {           ``"type":
"Point",           ``"coordinates": [37.6165, 55.8505]       ``}    ``}]};
```

下一步，描述点的图标并添加一个图层，该图层描述了需要用这个图标来绘制点。这应该在地图加载事件发生时进行。

```
map.on('load', function()
{    ``map.loadImage(       ``'https://geo.rustore.ru/api/styles/pins/blue_targ
et.png',       ``function(error, image) {           ``if(error) throw
error;           ``map.addImage('custom_pin',
image);           ``map.addLayer({              ``"id": "points"
```

```
, ``"type": "symbol", ``"source":
{ ``"type": "geojson", ``"data":
userPointData ``}, ``"layout":
{ ``"icon-image": "custom_pin"
, ``"icon-size":
1 ``} ``}); ``} ``);});
```

添加线条



线性对象以与点对象相同的方式添加到地图上 — 需要准备数据并用样式描述如何绘制它们。

就像在前一个示例中一样，需要准备数据，比如两条线。

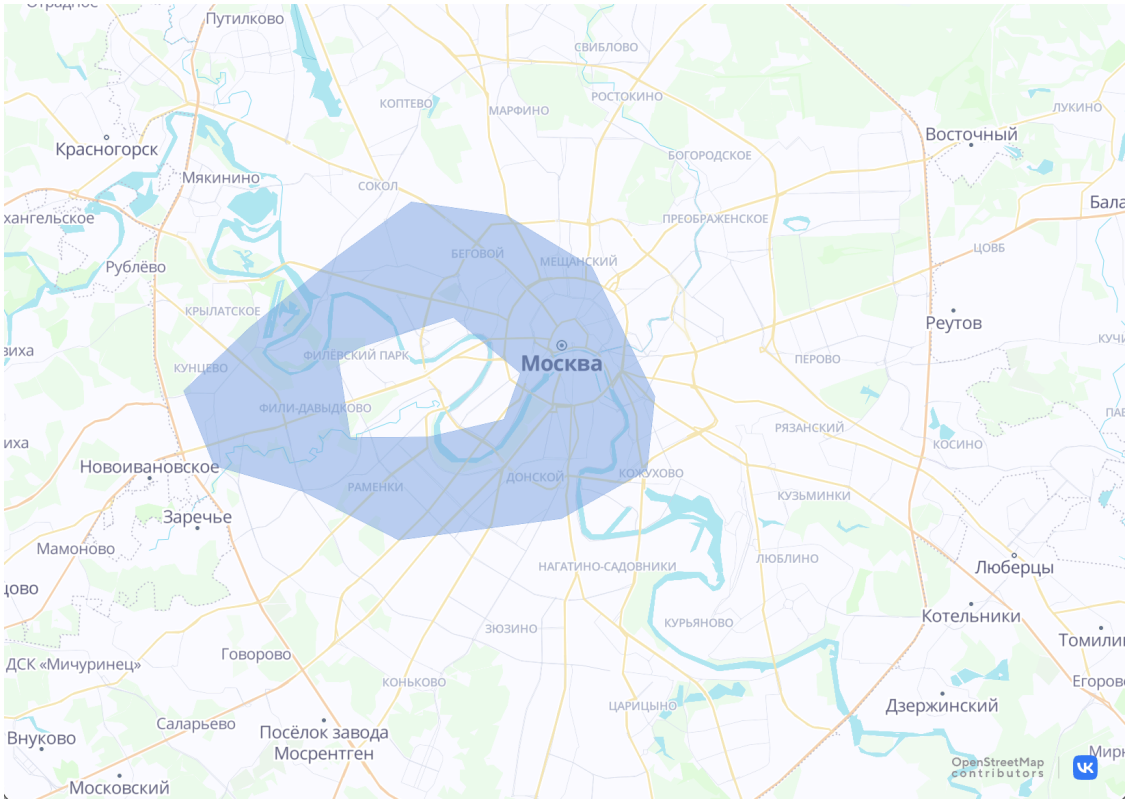
```
let userLineData = { ``"type": "FeatureCollection", ``"features":
[{ ``"type": "Feature", ``"geometry": { ``"type":
"LineString", ``"coordinates": [ ``[37.6165,
55.7505], ``[37.6375, 55.7515], ``[37.6375,
55.7515], ``[37.665, 55.7545], ``[37.765,
55.7645], ``[37.785, 55.7745], ``[37.803,
55.7645], ``[37.83, 55.7545], ``[37.89,
55.78] ``] ``} ``}, { ``"type": "Feature"
, ``"geometry": { ``"type": "LineString"
, ``"coordinates": [ ``[37.4165,
55.7505], ``[37.4166, 55.7505], ``[37.4169,
55.6], ``[37.5169, 55.545], ``[37.6169,
```

```
55.45],                ``[37.5169, 55.334],                ``[37.3169,
55.211]            ``]        ``}    ``}]};
```

接下来，添加一个图层的描述来进行绘制。在主地图的加载事件中也添加一个新图层。

```
map.on('load', function() {    ``map.addLayer({        ``'id': 'route'
,        ``'type': 'line',        ``'source': {            ``'type':
'geojson',            ``'data': userLineData        ``},        ``'layout':
{            ``'line-join': 'round',            ``'line-cap':
'round'        ``},        ``'paint': {            ``'line-color': '#AE3478'
,            ``'line-width': 8        ``}    ``});});
```

添加多边形



添加多边形图层的方式与创建线条图层类似。多边形可以是实心的，也可以有空洞。作为示例，请向地图添加一个复杂的多边形：

```
let userPolygonData = {    ``"type": "FeatureCollection",    ``"features":
[{        ``"type": "Feature",        ``"geometry": {            ``"type":
"Polygon",            ``"coordinates":
[                ``[                    ``[37.427278,
55.756486],                    ``[37.387117,
55.734843],                    ``[37.405653,
55.709126],                    ``[37.4592,
55.70042],                    ``[37.518239,
55.683585],                    ``[37.617439,
55.690939],                    ``[37.669614,
```

```
55.707578],                    ``[37.674419,
55.73291],                   ``[37.635975,
55.777345],                    ``[37.583457,
55.795105],                    ``[37.525791,
55.799544],                    ``[37.478765,
55.780049],                    ``[37.427278, 55.756486]                ``],
[                  ``[37.495585, 55.749917],                    ``[37.481511,
55.745666],                    ``[37.48872,
55.718796],                    ``[37.536432,
55.718989],                    ``[37.582428,
55.724983],                    ``[37.592382,
55.740835],                    ``[37.551535,
55.759964],                    ``[37.495585,
55.749917]               ``]           ``]       ``}    ``}]};
```

并描述如何在地图上绘制它：

```
map.on('load', function() {    ``map.addLayer({        ``'id': 'maine'
,        ``'type': 'fill',        ``'source': {            ``'type':
'geojson',            ``'data': userPolygonData        ``},        ``'layout':
{},        ``'paint': {            ``'fill-color': '#6ea5e8'
,            ``'fill-opacity': 0.5        ``}    ``});});
```

快速启动

MMRGLJS——一个JavaScript库，使用WebGL来渲染交互式地图。

快速启动

在html页面上的连接

需要添加

` |

创建令牌

为了安装 @geors/maps-sdk-js，在github上创建一个令牌：

1. 转到用户设置。
2. 在左下方找到 Developer settings 按钮。
3. 创建一个具有读取包权限 (read:packages) 的 Personal access token。
4. 在项目根目录中创建一个 .npmrc 文件。
5. 在创建的文件中添加两行。

@geors:registry=https://npm.pkg.github.com//npm.pkg.github.com/:_authToken=ACCESS_TOKEN

将 ACCESS_TOKEN 替换为您创建的令牌。

安装包

转到项目根目录下的 package.json，并使用 npm 或 yarn 安装包

```
yarn add
@geors/maps-sdk-js
```

或

```
npm install
@geors/maps-sdk-js
```

要初始化地图，请添加以下代码（以React为例）：

```
import mmrgl from  '@geors/maps-sdk-js';import { useEffect } from  'react'import
'@geors/maps-sdk-js/dist/mmr-gl.css';export  function Map() {  ``useEffect( () =>
{  ``mmrgl.accessToken =  'accessToken';  ``const map =  new
mmrgl.Map({  ``container:  'map',  ``zoom:
8,  ``center: [37.6165, 55.7505],  ``style:
'mmr://api/styles/main_style.json',  ``hash:  true
,  ``})  ``return () => {  ``if (map)
map.remove();  ``}  ``}  ``return <div id= "map" style={{ width:  '800px'
, height:  '600px' }} />}
```

详细信息

关于使用 MMR GL JS SDK 的更详细信息，请参阅以下部分:

1。地图

页面上的地图与Map对象相关联。它提供了从代码中修改地图的方法和属性。创建地图时，MMR GL JS会初始化它并返回Map对象。

参数

名称	类型	描述
container	HTMLElement | string	HTMLMMR GL JS将在其内部渲染地图的HTML元素或区块id。指定的元素不应包含子元素。
minZoom	numberdefault: 0	最小zoom级别（0-20）。
maxZoom	numberdefault: 20	最大zoom级别（0-20）。
minPitch	numberdefault: 0	最小地图倾斜级别（0-85）。
maxPitch	numberdefault: 85	最大地图倾斜级别（0-85）。

名称	类型	描述
style	object | string	地图样式的JSON格式或样式链接。这JSON一个根据样式设计规则编写的JSON对象。light_style.json— —mmr://api/styles/light_style.json-dark_style.json— —mmr://api/styles/dark_style.json-main_style.json—mmr://api/styles/main_style.json
hash	boolean | stringdefault: false	如果为true，页面的所有参数（zoom、latitude、longitude and pitch）将通过#参数与URL同步。
interactive	booleandefault: false	如果为false，则地图不会对任何控制元素（鼠标、屏幕、键盘）作出反应。
bearingSnap	numberdefault: 7	以度数测量的边界，用于确定地图的方位角何时锁定到北方。
pitchWithRotate	booleandefault: true	如果为false，使用“drag and rotate”控制地图倾斜将被禁用。
clickTolerance	numberdefault: 3	用户在点击时可以移动鼠标指针的最大像素数，以便被认为是有效点击（而不是拖动鼠标）。
attributionControl	booleandefault: true	如果为true，AttributionControl将被添加到地图上。
customAttribution	string | Array	在AttributionControl中显示的字符串或字符串行。只有当attributionControl = true时才可能。
failIfMajorPerfomanceCaveat	booleandefault: false	如果为true，如果地图性能不可接受，地图初始化将以错误结束。
preserveDrawingBuffer	booleandefault: false	如果为true，可以使用map.getCanvas().toDataUrl

名称	类型	描述
		() 将地图画布导出为PNG图像。默认设置为false，以提高性能。
antialias	booleandefault: false	如果为true，gl上下文将创建时应用MSAA平滑，这可能对平滑自定义层很有用。默认设置为false，以提高性能。
refreshExpiredTiles	booleandefault: true	如果为false，则地图不会根据cacheControl/expires头文件中的过期时间来请求瓦片。
maxBounds	LngLatBoundsLike	如果设置，地图将被限制在指定的边界内。
scrollZoom	boolean | objectdefault: true	如果为true，那么使用滚动放大的功能将会生效。对象的值将作为ScrollZoomHandler的参数传递。
boxZoom	booleandefault: true	如果为true，"box zoom"类型的交互将被启用（更多详情见BoxZoomHandler）
dragRotate	booleandefault: true	如果为true，"drag to rotate"类型的交互将被启用（更多详情见DragRotateHandler）
dragPan	boolean | objectdefault: true	如果为true，"drag to pan"类型的交互将被启用（更多详情见DragPanHandler）
keyboard	booleandefault: true	如果为true，可以使用键盘和快捷键与地图交互（详见KeyboardHandler）
doubleClickZoom	booleandefault: true	如果为true，"double click to zoom"交互将被启用（详见 DoubleClickZoomHandler ）
touchZoomRotate	boolean | objectdefault: true	如果为true，"pinch to rotate and zoom"交互将被

名称	类型	描述
		启用（详见 TouchZoomRotateHandler）
touchPitch	boolean | objectdefault: true	如果为true，"drag to pitch"交互将被启用（详见 TouchPitchHandler）
trackResize	booleandefault: true	如果为true，地图将自动调整大小以适应窗口变化。
center	LngLatLikedefault: [0,0]	地图的起始点。如果构造器参数中没有指定中心点，MMR GL JS 将在地图样式对象中查找。如果样式中也未指定，则默认为 [0, 0]。注意：使用经度、纬度的坐标顺序（与纬度、经度相反）以匹配 GeoJSON。
zoom	numberdefault: 0	地图的初始缩放级别。如果构造器参数中未指定缩放级别，MMR GL JS 将在地图样式对象中查找。如果样式中也未指定，则默认为 0。
bearing	numberdefault: 0	初始地图方位角（旋转），以北为基准逆时针度量。如果构造器参数中未指定方位角，MMR GL JS 将在地图样式对象中查找。如果样式中也未指定，则默认为 0。
pitch	numberdefault: 0	初始地图倾斜角度，从屏幕平面以度数计量（0-85）。如果构造器参数中未指定倾斜角度，MMR GL JS 将在地图样式对象中查找。如果样式中也未指定，则默认为 0。
bounds	LngLatBoundsLike	初始地图边界。如果设置了边界，它将覆盖 center 和 zoom 设置。
fitBoundsOptions	object	MapMap#fitBounds 对象参数应仅在调整上述初始边界 (bounds) 时使用。

名称	类型	描述
renderWorldCopies	booleandefault: true	如果为 true，则多个世界副本将并排显示在 -180 和 180 度经度之外。如果设置为 false：- 当地图放大到世界的任何一个图像无法填满地图容器的边界外的 180 和 -180 度经度时，将留下空白空间。- 穿越 180 和 -180 度经度的对象将在每个缩放级别上被切割成两部分（一部分位于地图的右边缘，另一部分位于左边缘）。
maxTileCache	numberdefault: null	对于特定来源（source）存储在缓存中的瓦片的最大数量。如果省略此参数，缓存大小将根据当前视图窗口动态确定。
localIdeographFontFamily	stringdefault: sans-serif	定义 CSS font-family 以在 *‘CJK Unified Ideographs’*、*‘Hiragana’*、*‘Katakana’* 和 *‘Hangul Syllables’* 中本地覆盖字形生成。在这些语言结构中，地图样式中的字体设置将被忽略，除了 font-weight（light/regular/medium/bold）。设置为 false 以启用这些字形的地图样式中的字体设置。
localFontFamily	stringdefault: false	定义 CSS font-family 以覆盖所有字形的生成。地图样式中的字体设置将被忽略，除了 font-weight（light/regular/medium/bold）。如果设置了此参数，则它将覆盖 localIdeographFontFamily 中的设置。

名称	类型	描述
transformRequest	RequestTransformFunctiondefault: null	该函数在执行 URL 请求之前执行。它可用于更改 URL、设置请求头或跨域请求的 credentials 属性。
collectResourceTiming	booleandefault: false	如果为 true，则 Resource Timing API 的信息将可用于执行 GeoJSON 和 Vector Tile 的请求（这些信息通常无法从 JavaScript 的主线程获取）。信息将在 ResourceTiming 属性中返回。
fadeDuration	numberdefault: 300	控制 fade-in/fade-out 动画的持续时间，用于标签冲突的毫秒数。此参数不影响执行期间样式转换的持续时间或栅格瓦片的交叉淡出。
crossSourceCollisions	booleandefault: true	如果为 true，则多个来源的符号在碰撞检测期间可以相互碰撞。如果为 false，则对每个来源的符号分别进行碰撞检测。
accessToken	stringdefault: null	如果指定，那么将使用此令牌代替 mmrgl.accessToken 中指定的令牌。mmrgl.accessToken
locale	objectdefault: null	用于应用于默认本地化表的补丁，例如用户界面的提示控制。locale 对象将用户界面字符串的标识符与目标语言中翻译的字符串相匹配；请参阅 src/ui/default_locale.js，例如，其中包含所有支持的字符串标识符。对象可以指定全部用户界面字符串（从而添加新翻译支持）或仅指定部分字符串（从而修正默认翻译表）。

示例

```
var map = new mmrgl.Map({    ``container: 'map',    ``center: [37.6165,
55.7505],    ``zoom: 8,    ``style: styleObject,    ``hash: true
,    ``transformRequest: (url, resourceType)=> {           ``if(resourceType
=== 'Source' && url.startsWith('http://myHost')) {               ``return
{                   ``url: url.replace('http', 'https'
),                   ``headers: { 'my-custom-header': true
},                   ``credentials: 'include' // Include cookies for
cross-origin requests               ``}       ``}    ``}});
```

方法

名称	描述
addControl(control, position?)	将 IControl 添加到地图，调用 control.onAdd(this)。参数： **control:iControl** 添加的 IControl **position:string** 地图上的位置（‘top-left’，‘top-right’，‘bottom-left’ and ‘bottom-right’）。默认为 ‘top-right’。
addImage(id, image, options)	将图像添加到样式中。这个图像可以像使用图像 ID 的任何其他图标一样在地图上显示，使用 icon-image、background-pattern、fill-pattern 或 line-pattern。如果精灵中没有足够的空间添加此图像，将触发 Map.event:error 事件。

参数：

id:stringID图像

image:HTMLImageElement| ImageBitmap | ImageData | {width: number, height: number, data: (Uint8Array | Uint8ClampedArray)} | StyleImageInterface 以 HTMLImageElement、ImageData、ImageBitmap 或具有 width、height 和 data 属性（与 ImageData 相同格式）的对象形式的图像ImageData

options{

pixelRatio 图像中像素与屏幕物理像素的比率。

sdf 是否将图像解释为 SDF 图像。

stretchX[[x1, x2], …] 如果在含有此图像的图层中使用 icon-text-fit，此选项定义图像的部分（几部分）可以水平拉伸。

stretchY[[y1, y2], …] 如果在含有此图像的图层中使用 icon-text-fit，此选项定义图像的部分（几部分）可以垂直拉伸。

content[x1, y1, x2, y2] 如果在含有此图像的图层中使用 icon-text-fit，此选项定义图像的部分可以被文本字段中的内容覆盖。

```
}// If the style's sprite does not already contain an image with ID 'cat',// add the image 'cat-icon.png' to the style's sprite with the ID 'cat'.map.loadImage( 'https://upload.wikimedia.org/wikipedia/commons/thumb/6/60/Cat_silhouette.svg/400px-Cat_silhouette.svg.png', function (error, image) {    ``if (error) throw error;    ``if (!map.hasImage('cat')) map.addImage('cat', image);});  // Add a stretchable image that can be used with `icon-text-fit`// In this example, the image is 600px wide by 400px high.map.loadImage( 'https://upload.wikimedia.org/wikipedia/commons/8/89/Black_and_White_Boxed_%28bordered%29.png', function (error, image) {    ``if (error) throw error;    ``if (!map.hasImage('border-image')) {        ``map.addImage('border-image', image, {            ``content: [16, 16, 300, 384], // place text over left half of image, avoiding the 16px border            ``stretchX: [[16, 584]], // stretch everything horizontally except the 16px border            ``stretchY: [[16, 384]], // stretch everything vertically except the 16px border        ``});    ``}});
```

addLayer(layer, beforedId?)

图层定义了来自指定来源的数据如何被样式化。

参数：

layer ((Object | CustomLayerInterface)) 添加的图层符合样式规范或CustomLayerInterface规范。

Layer对象的描述：

名称	描述
id:string	唯一图层标识符
type:string	图层类型（例如填充或符号）。图层类型列表可在样式规范中找到。（也可以是自定义的。更多信息请见CustomLayerInterface部分）
source:(string | object)	图层的数据源。参考已使用唯一标识符定义的来源。直接使用原始对象引用新来源（如样式规范中所定义）。这对所有layer.type参数都是必需的，除了custom。
sourceLayer	（可选）指定的layer.source中的图层名称，将用于样式层。这仅适用于矢量瓦片，且当layer.source为vector类型时需要。
filter:array	（可选）用于设定对象条件的表达式。只显示符合筛选器条件的对象。样式规范中包含了筛选器参数限制和可用表达式的完整列表。如果未设置筛选器，则会显示所有对象。

名称	描述
layer:object	（可选）图层属性。可用的属性取决于图层类型。每种类型图层的完整属性列表可在样式规范中找到。如果未设置布局属性，则将使用默认值。
maxzoom:number	（可选）图层的最大缩放级别。在等于或超过 maxzoom 的缩放级别下，图层将被隐藏。该值可以是从 0 到 24（包括 24）的任何数字。如果未指定 maxzoom，则在可用瓦片的所有缩放级别上都可以看到图层。
minzoom:number	（可选）图层的最小缩放级别。在小于 minzoom 的缩放级别下，图层将被隐藏。该值可以是从 0 到 24（包括 24）的任何数字。如果未指定 minzoom，则在可用瓦片的所有缩放级别上都可以看到图层。
metadata:object	（可选）可以与图层一起跟踪但不影响渲染的自定义属性。
renderingMode:string	这仅适用于类型为 custom 的图层。有关更多信息，请参阅 CustomLayerInterface 部分。

beforeId:stringbeforeId:string 在现有图层ID之前插入新图层，这样新图层在视觉上会出现在现有图层下方。如果未指定此参数，图层将被添加到图层数组的末尾，并在视觉上出现在所有其他图层之上。

```
// Add a circle layer with a vector sourcemap.addLayer({    ``id: 'points-of-interest',    ``source: {        ``type: 'vector',        ``url: 'path-to-source'    ``},    ```'source-layer': 'poi_label',    ``type: 'circle',    ``paint: {        ``// Style Specification paint properties    ``},    ``layout: {    ``// Style Specification layout properties    ``}});


// Define a source before using it to create a new layermap.addSource('state-data', {        ``type: 'geojson',        ``data: 'path/to/data.geojson'});  map.addLayer({        ``id: 'states',        ``// References the GeoJSON source defined above        ```// and does not require asource-layer``<br>        source: ``'state-data'`` `,`<br>`        type: 'symbol',<br>        layout: {`<br>`                // Set the label content to the<br>                `// feature's `name` property                ``text-field: ['get ', ' name']        ``}});
```

```
// Add a new symbol layer before an existing layermap.addLayer({    ``id: 'states',    ``// References a source that's already been defined    ``source: 'state-data',    ``type: 'symbol',    ``layout: {        ``// Set the label content to the        ``// feature's `name` property        ``text-field: ['get', 'name']    ``}    ```// Add the layer before the existingcitieslayer``<br>}, 'cities');`
```

addSource(id, source)

参数：

id:string添加源的标识符。不应与现有源冲突。

source:object源对象，符合源规范或 CanvasSourceOptions。

```
map.addSource('my-data', {    ``type: 'vector',    ``url: 'path-to-source'});
```

```
map.addSource('my-data', {    ``"type": "geojson",    ``"data": {        ``"type": "Feature",        ``"geometry": {            ``"type": "Point",            ``"coordinates": [-77.0323, 38.9131]        ``},        ``"properties": {            ``"title": "title",            ``"marker-symbol": "monument"        ``}    ``}});
```

areTilesLoaded()

```
var tilesLoaded = map.areTilesLoaded();
```

boxZoom

cameraForBounds(bounds, options?)

bounds:LngLatBoundsLike 根据最高缩放级别计算窗口中边界的中心，直到Map#getMaxZoom()，这个缩放级别适合于窗口。LngLatBounds 是一个始终沿着0方位角轴对齐的盒子。

名称	描述
padding:number|PaddingOptions	添加到指定边界的像素数量的填充量。
bearing:number	动画结束时地图的预期方位角，以度为单位。
offset:PointLike	相对于地图中心的设定边界中心，以像素为单位测量。
maxZoom:number	在摄像机转移到设定边界时允许的最大缩放级别。

```
var bbox = [[-79, 43], [-73, 45]];  ``var newCameraTransform = map.cameraForBounds(bbox, {  ``padding: {top: 10, bottom:25, left: 15, right: 5}});
```

doubleClickZoom
dragPan
dragRotate
easeTo(options, eventData?)
fitBounds(bounds, options?, eventData?)

参数：

bounds: LngLatBoundsLike 将这些边界在窗口中居中，并使用最高的缩放级别，直到Map#getMaxZoom()，它们在窗口中相对应。

options:Objectoptions:Object 选项支持来自 AnimationOptions 和 CameraOptions 的所有属性，以及以下额外字段。

名称	描述
padding:number|PaddingOptions	添加到指定边界的像素数量的填充量。
linear:boolean	如果为true，地图将使用 Map#easeTo 进行过渡。如果为false，地图将使用 Map#flyTo 进行过渡。这些功能和 AnimationOptions 用于获取关于可用选项的信息。
easing:function?	减弱动画过渡的函数。参见 AnimationOptions 部分。
offset:PointLike	相对于地图中心的设定边界中心，以像素为单位测量。
maxZoom:number	在摄像机转移到设定边界时允许的最大缩放级别。

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
var bbox = [[-79, 43], [-73, 45]];map.fitBounds(bbox, {  ``padding: {top: 10, bottom:25, left: 15, right: 5}});
```

fitScreenCoordinates(p0, p1, bearing, options?, eventData?)

参数：

p0: PointLike 屏幕上的第一个点，以像素坐标表示

p1: PointLike 屏幕上的第二个点，以像素坐标表示

bearing:number动画结束时地图的预期方位角，以度为单位。如果地图有非零倾斜，此值将被忽略。

options:

名称	描述
padding:number|PaddingOptions	添加到指定边界的像素数量的填充量。

名称	描述
linear:boolean	如果为true，地图将使用 Map#easeTo 进行过渡。如果为false，地图将使用 Map#flyTo 进行过渡。这些功能和 AnimationOptions 用于获取关于可用选项的信息。
easing:function?	减弱动画过渡的函数。参见 AnimationOptions 部分。
offset:PointLike	相对于地图中心的设定边界中心，以像素为单位测量。
maxZoom:number	在摄像机转移到设定边界时允许的最大缩放级别。

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
var p0 = [220, 400];var p1 = [500, 900];map.fitScreenCoordinates(p0, p1, map.getBearing(), {  ``padding: {top: 10, bottom:25, left: 15, right: 5}});
```

flyTo(options, eventData?)

如果用户在其操作系统中启用了减少运动的可用性功能（reduced motion），动画将被跳过，这将等同于 jumpTo，除非“参数”包含 essential: true。

名称	描述
curve:numberdefault: 1.42	“curve” 的缩放，将沿着移动轨迹进行。高值将最大化缩放以产生夸张的动画效果，而低值将最小化缩放，从而产生类似于 Map#easeTo 的效果。1.42— 是 van Wijk (2003) 讨论的用户研究参与者选择的平均值。Math.pow(6, 0.25) 的值将等同于均方根平均速度。值 1 将产生圆周运动。
minZoom:number	在运动轨迹的顶点上，缩放级别为零。如果指定了 options.curve 参数，则忽略此参数。
speed:numberdefault: 1.2	动画的平均速度是相对于 options.curve 来确定的。速度 1.2 表示地图似乎沿着飞行轨迹移动速度快 1.2 倍。Screenful —地图的可视区间。它不对应于固定的物理距离，而是随缩放级别的变化而变化。

名称	描述
screenSpeed:number?	动画的平均速度以屏幕秒为单位测量，假设是线性时间曲线。如果指定了 options.speed 参数，则此参数将被忽略。
maxDuration:number?	动画的最大持续时间以毫秒为单位测量。如果持续时间超过最大持续时间，它将被重置为 0。

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
// fly with default options to null islandmap.flyTo({center: [0, 0], zoom: 9});// using flyTo optionsmap.flyTo({  ``center: [0, 0],  ``zoom: 9,  ``speed: 0.2,  ``curve: 1,  ``easing(t) {    ``return t;  ``}});
```

getBearing()

getBounds()

```
var bounds = map.getBounds();
```

getCanvas()

getCanvasContainer()

getCenter()

```
// return a LngLat object such as {lng: 0, lat: 0}var center = map.getCenter();// access longitude and latitude values directlyvar {longitude, latitude} = map.getCenter();
```

getContainer()

getFeatureState(feature)

为了获取对象的值以用于对象的样式化，请使用特性表达式。

参数：

f **eature:Object** 对象的标识符。从 Map#queryRenderedFeatures 或事件处理器返回的特性对象，可以作为对象的标识符使用。

options:

名称	描述
id:number|string	功能的唯一标识符。可以是整数或字符串，但仅在应用了 promoteId 参数到源或字符串可以转换为整数时，支持字符串值。

名称	描述
source:string	矢量或 GeoJSON 源的对象标识符。
sourceLayer:string	（可选）对于矢量瓦片源，需要 sourceLayer。

```
// When the mouse moves over the `my-layer` layer, <br>`// get the feature state for the feature under the mouse`<br>`map.on(` `'mousemove'` `, ` `'my-layer'` `, ` `function` `(e) {`<br>` if(e.features.length > 0) {<br> map.getFeatureState({`<br>` source: ‘my-source’,<br> sourceLayer: ` `'my-source-layer'` `,`<br>` id: e.features[0].id<br> });`<br>` }<br>});`
```

getFilter(layerId)

getFreeCameraOptions()

getLayer(id)

参数：

id:stringid:string 是需要获取的图层的标识符。

```
var stateDataLayer = map.getLayer( 'state-data' );
```

getLayoutProperty(layerId, name)

getLight()

getMaxBounds()

```
var maxBounds = map.getMaxBounds();
```

getMaxPitch()

getMaxZoom()

```
var maxZoom = map.getMaxZoom();
```

getMinPitch()

getMinZoom()

```
var minZoom = map.getMinZoom();
```

getPadding()

getPaintProperty(layerId, name)

getPitch()

getRenderWorldCopies()

- 当地图放大到世界的任何图像都无法填满地图容器时，在180度和-180度经度之外将会留下空白空间。

- 横跨180度和-180度经度的对象将在每个缩放级别被切成两半（地图的右边缘和左边缘各有一部分）。

```
var worldCopiesRendered = map.getRenderWorldCopies();
```

getSource(id)

这种方法常用于使用实例元素更新源，适用于在源中定义的相应类型的源。例如，为 GeoJSON 源设置数据或更新图像源的 URL 地址和坐标。

参数：

id:string 需要获取的源的标识符。

```
var sourceObject = map.getSource( 'points' );
```

getStyle()

```
map.on( 'load' , function () {  ``var styleJson = map.getStyle();});
```

getTerrain()

getZoom()

```
map.getZoom();
```

hasControl(control)

参数：

control:IControlIControl用于检查的 IControl。

```
// Define a new navigation control.var navigation = new
mmrgl.NavigationControl();// Add zoom and rotation controls to the
map.map.addControl(navigation);// Check that the navigation control exists on
the map.const added = map.hasControl(navigation);// added === true
```

hasImage(id)

参数：

id:stringid:string 图像的标识符。

```
// Check if an image with the ID 'cat' exists in// the style's sprite.var
catIconExists = map.hasImage('cat');
```

isMoving()

```
var isMoving = map.isMoving();
```

isRotating()

```
map.isRotating();
```

isSourceLoaded(id)

参数：

id:string需要检查的源（source）的标识符。

```
var sourceLoaded = map.isSourceLoaded( 'bathymetry-data' );
```

isStyleLoaded()

```
var styleLoadStatus = map.isStyleLoaded();
```

isZooming()

```
var isZooming = map.isZooming();
```

jumpTo(options, eventData?)

参数：

options: CameraOptions 选项对象

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
// jump to coordinates at current zoommap.jumpTo({center: [0, 0]});// jump with zoom, pitch, and bearing optionsmap.jumpTo({  ``center: [0, 0],  ``zoom: 8,  ``pitch: 45,  ``bearing: 90});Copy
```

keyboard

listImages()

```
var allImages = map.listImages();
```

loaded()

loadImage(url, callback)

参数：

url:string图像文件的 URL 地址。图像文件应为 png、webp 或 jpg 格式。

callback:Function等待callback（error, data）。在图像加载时调用，或在有错误时带有 error 参数。

```
// Load an image from an external URL.map.loadImage( 'http://placekitten.com/50/50' , function (error, image) {if (error) throw error;  ``// Add the loaded image to the style's sprite with the ID 'kitten'.  ``map.addImage('kitten', image);});
```

moveLayer(id, beforeId?)

参数：

id:stringid:string 移动图层的标识符。

beforeId:string 新图层将被插入到此图层标识符之前。在查看地图时，id图层将出现在BeforeID图层下方。如果省略BeforeID参数，则该图层将被添加到图层数组的末尾，并且将出现在地图上所有其他图层之上。

```
// Move a layer with ID 'polygon' before the layer with ID 'country-label'. The `polygon` layer will appear beneath the `country-label` layer on the map.map.moveLayer( 'polygon' , 'country-label' );
```

off(type, listener)

off(type, layerId, listener)

on(type, layerId, listener)

参数：

type:stringtype:string 监听器（listener）的事件类型。与可选参数layerId兼容的事件会在光标进入指定图层的可见部分时从该图层外部或地图画布的边界外部启动。

事件	与 layerId 的兼容性 layerId
mousedown	yes
mouseup	yes
mouseover	yes
mouseout	yes
mousemove	yes
mouseenter	yes (required)
mouseleave	yes (required)
click	yes
dbclick	yes
contextmenu	yes
touchstart	yes
touchend	yes
touchcancel	yes
wheel	
resize	
remove	
touchmove	
movestart	
move	
moveend	
dragstart	
drag	

事件	与 layerId 的兼容性 layerId
dragend	
zoomstart	
zoom	
zoomend	
rotatestart	
rotate	
rotateend	
pitchstart	
pitch	
pitchend	
boxzoomstart	
boxzoomend	
boxzoomcancel	
webglcontextlost	
webglcontextrestored	
load	
render	
idle	
error	
data	
styledata	
sourcedata	
dataloading	
styledataloading	
sourcedataloading	
styleimagemissing	

layerId:string（可选）样式层的标识符。事件将仅在其位置位于该层中可见对象的范围内时被触发。事件将具有 features 属性，包含相应对象的数组。如果未指定 layerId，事件将不具有 features 属性。请注意，许多类型的事件与额外的 layerId 参数不兼容。

listener:Function触发事件时调用的函数。

```
// Set an event listener that will fire// when the map has finished
loadingmap.on( 'load' , function () {    ``// Once the map has finished
loading,    ``// add a new layer    ``map.addLayer({        ``id:
'points-of-interest' ,        ``source: {            ``type: 'vector'
,            ``url: 'path-to-source'        ``},        ``'source-layer' :
'poi_label' ,        ``type: 'circle' ,        ``paint: {            ``// Style
```

```
Specification paint properties        ``},        ``layout: {            ``// Style Specification layout properties        ``}    ``});});


// Set an event listener that will fire// when a feature on the countries layer of the map is clickedmap.on( 'click' , 'countries' , function (e) {    ``new mmrgl.Popup()        ``.setLngLat(e.lngLat)        ```.setHTML(Country name: ${e.features[0].properties.name})``<br>    `.addTo(map);});
```

once(type, listener)

once(type, layerId, listener)

panBy(offset, options?, eventData?)

panTo(lnglat, options?, eventData?)

参数：

lnglat: LngLatLike 用于地图平移的位置。

options: AnimationOptions 选项对象

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
map.panTo([-74, 38]);


// Specify that the panTo animation should last 5000 milliseconds.map.panTo([-74, 38], {duration: 5000});
```

project(lnglat)

当地图倾斜且 lnglat 完全位于摄像机后面时，没有任何像素坐标对应于此位置。在这种情况下，返回点的 x 和 y 组件值为 Number.MAX_VALUE。

参数：

lnglat: LngLatLike 项目的地理位置。

```
var coordinate = [-122.420679, 37.772537];var point = map.project(coordinate);
```

queryRenderedFeatures(geometry? options?)

参数：

geometry: PointLike | Array 请求区域的几何图形，以像素为单位：可以是一个点，或者是描述边界框的左下角和右上角的点，其中坐标原点位于左上角。如果省略此参数（即调用 Map#queryRenderedFeatures 时没有参数或仅带 options 参数），相当于传递一个覆盖整个地图视图屏幕的边界矩形。

options:

名称	描述
layers:Array	用于检查请求的样式层标识符数组。只会返回这些层内的对象。如果未定义此参数，则将检查所有层。
filter:Array	用于限制请求结果的过滤器。
vali	

返回：

Array: GeoJSON对象数组。

每个返回对象的属性值包含其原始对象的属性。对于GeoJSON源，仅支持属性的字符串和数字值（即不支持null、Array和Object值）。

每个对象都包括其顶层层级、源和原始层的属性。layer属性是一个表示该对象所属的样式层的对象。此对象中的layout和paint属性包含了针对当前缩放级别和对象完全计算的值。

只包含当前正在呈现的功能。某些功能将不被包括，例如：

- 层中的对象，其可见性属性等于“none”。
- 层中的对象，其缩放范围排除了当前缩放级别。
- 由于文本或图标冲突而被隐藏的符号功能。

包括来自所有其他层的对象，包括可能没有可视化结果的对象；

在返回的数组中，最顶层的可视化对象首先出现，其后的对象按照 z-顺序递减排序。多次可视化的对象（由于在低缩放级别下沿反子午线的包裹），只返回一次。

由于对象来自矢量瓦片或转换为瓦片的 GeoJSON 数据，对象的几何可能跨越瓦片边界被分割或重复，因此在查询结果中可能多次出现。例如，假设有一条高速公路穿过查询请求的限制矩形。查询结果将是那些位于覆盖限制矩形的地图瓦片内的高速公路部分，即使高速公路延伸到其他瓦片中，每个瓦片内的高速公路部分也将作为单独的对象返回。同样，靠近瓦片边界的点对象可能由于瓦片缓冲而在多个瓦片中出现。

```
// Find all features at a pointvar features = map.queryRenderedFeatures(  ``[20, 35],  ``{ layers: [ 'my-layer-name' ] });


// Find all features within a static bounding boxvar features = map.queryRenderedFeatures(  ``[[10, 20], [30, 50]],  ``{ layers: [ 'my-layer-name' ] });


// Find all features within a bounding box around a pointvar width = 10;var height = 20;var features = map.queryRenderedFeatures([  ``[point.x - width / 2, point.y - height / 2],  ``[point.x + width / 2, point.y + height / 2]], { layers: [ 'my-layer-name' ] });
```

```
// Query all rendered features from a single layervar features =
map.queryRenderedFeatures({ layers: [ 'my-layer-name' ] });
```

querySourceFeatures(sourceId, parameters?)

参数：

sourceId:string用于查询的矢量瓦片或GeoJSON源的标识符。

parameters:

名称	描述
sourceLayer:string	查询的原始图层名称。对于矢量瓦片源，此参数是必需的。对于GeoJSON源，此参数被忽略。
filter:Array	用于限制请求结果的过滤器。

vali

返回：

Array: GeoJSON对象数组。

与Map#queryRenderedFeatures不同，此函数返回所有符合查询参数的对象，无论它们是否以当前样式显示（即是否可见）。查询域包括当前加载的所有矢量瓦片和GeoJSON源瓦片：此函数不检查当前可视视窗之外的瓦片。

由于对象来自矢量瓦片或内部转换为瓦片的 GeoJSON 数据，对象的几何可能跨越瓦片边界被分割或重复，因此在查询结果中可能多次出现。例如，假设有一条高速公路穿过查询请求的限制矩形。查询结果将包括位于覆盖边界矩形的地图瓦片内的公路部分，即使公路延伸到其他瓦片。地图上每块瓦片内的公路部分都将作为单独的对象返回。同样，靠近瓦片边界的点对象可能因瓦片的缓冲而出现在多个瓦片中。

```
// Find all features in one source layer in a vector sourcevar features =
map.querySourceFeatures( 'your-source-id' , {  ``sourceLayer:
'your-source-layer'});
```

remove()

removeControl(control)

参数：

control:iControl可删除的 IControlIControl

```
// Define a new navigation control.var navigation =  new
mmrgl.NavigationControl();// Add zoom and rotation controls to the
map.map.addControl(navigation);// Remove zoom and rotation controls from the
map.map.removeControl(navigation);
```

removeFeatureState(feature, key)

参数：

feature:Object删除状态的对象标识符。这可能是源、对象或特定对象键。从 Map#queryRenderedFeatures或事件处理器返回的对象可用作对象标识符。

名称	描述
id:number|string	对象的唯一标识符。可以是整数或字符串，但仅在应用了 promoteId 参数到源或字符串可以转换为整数时，支持字符串值。
source:string	矢量或 GeoJSON 源的对象标识符。
sourceLayer:string	（可选）对于矢量瓦片源，需要 sourceLayer 。

key:string(可选) 对象状态中用于重置的键。

```
// Reset the entire state object for all features// in the `my-source`
sourcemap.removeFeatureState({  ``source:  'my-source'});


// When the mouse leaves the `my-layer` layer, <br>`// reset the entire state
object for the`<br>`// feature under the mouse`<br>`map.on(` `'mouseleave'` `, `
`'my-layer'` `, ` `function` `(e)
{`<br>`  map.removeFeatureState({<br>  source: ` `'my-source'`
`,`<br>`    sourceLayer: 'my-source-layer',<br>  id:
e.features[0].id`<br>`  });<br>});`


// When the mouse leaves the `my-layer` layer, <br>// reset only the hover key-value
pair in the<br>`// state for the feature under the mouse`<br>`map.on(`
`'mouseleave'` `, ` `'my-layer'` `, ` `function` `(e)
{`<br>`  map.removeFeatureState({<br>  source: ` `'my-source'`
`,`<br>`    sourceLayer: 'my-source-layer',<br>  id:
e.features[0].id`<br>`  }, 'hover');<br>});`
```

removeImage(id)

参数：

id:stringid:string 图像的标识符。

```
// If an image with the ID 'cat' exists in// the style's sprite, remove it.if
(map.hasImage('cat')) map.removeImage('cat');
```

removeLayer(id)

如果该图层不存在，将触发错误事件。

参数：

id:string要删除的图层的标识符。

```
// If a layer with ID 'state-data' exists, remove it.if (map.getLayer(
'state-data' )) map.removeLayer( 'state-data' );
```

removeSource(id)

参数：

id:string要删除的源的标识符。

```
map.removeSource( 'bathymetry-data' );
```

repaint

resetNorth(options?, eventData?)

resetNorthPitch(options?, eventData?)

resize(eventData?)

检查地图容器的大小是否已更改，并在更改后更新地图。在以编程方式更改地图容器或在地图最初被CSS隐藏后显示时，应调用此方法。

参数：

eventData:Object额外的属性，传递给由大小改变触发的movestart, move, resize 和 moveend事件。这对区分事件来源（例如，用户启动或程序启动的事件）可能很有用。

```
// Resize the map when the map container is shown// after being initially hidden
with CSS.var mapDiv = document.getElementById( 'map' );if (mapDiv.style.visibility
=== true ) map.resize();
```

rotateTo(bearing, options?, eventData?)

scrollZoom

setBearing(bearing, eventData?)

相当于 jumpTo({bearing: bearing})。

参数：

bearing:number期望的方位角

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
// rotate the map to 90 degreesmap.setBearing(90);
```

setCenter(center, eventData?)

参数：

center: LngLatLike 要设置的中心点。

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
map.setCenter([-74, 38]);
```

setFeatureState(feature, state)

这种方法只能与具有 feature.id 属性的来源一起使用。Feature.id 属性可以通过三种方式定义：

- 对于矢量源或GeoJSON源，在原始数据文件中包括 id 属性。
- 对于矢量源或GeoJSON源，在定义来源时使用 promoteId 选项。
- 对于GeoJSON源，使用 generateId 选项自动基于数据源中对象的索引分配标识符。如果您通过 map.getSource('someid').setData(..) 更改了对象数据，您可能需要考虑更新的标识符值来重新应用状态。

参数：

feature:Object对象的标识符。从Map#queryRenderedFeatures或事件处理器返回的对象可用作对象标识符。

名称	描述
id:number|string	对象的唯一标识符。可以是整数或字符串，但仅在应用了 promoteId 参数到源或字符串可以转换为整数时，支持字符串值。
source:string	矢量或 GeoJSON 源的对象标识符。
sourceLayer:string	（可选）对于矢量瓦片源，需要 sourceLayer。

state:Object键值对集合。值必须是有效的JSON类型。

```
// When the mouse moves over the `my-layer` layer, update// the feature state for the feature under the mousemap.on( 'mousemove' , 'my-layer' , function (e) {  ``if (e.features.length > 0) {    ``map.setFeatureState({      ``source: 'my-source' ,      ``sourceLayer: 'my-source-layer' ,      ``id: e.features[0].id,    ``}, {      ``hover: true    ``});  ``}});
```

setFilter(layerId, filter, options = {})

过滤器控制哪些样式层的对象从其来源被渲染。任何过滤器表达式为true的对象将被显示在地图上。为false的则会被隐藏。

使用setFilter来展示源数据的子集。

要清除过滤器，请传递null或undefined作为第二个参数。

参数：

layerId:string应用过滤器的图层的标识符。

filter: Array | null | undefined 过滤器，与样式规范的过滤器定义相对应。如果设置为null或undefined，则函数将删除该图层的任何现有过滤器。

options:object 默认为{} 选项对象。

名称　　描述

vali

```
// display only features with the 'name' property 'USA'map.setFilter( 'my-layer' , [ '==' , [ 'get' , 'name' ], 'USA' ]);
```

```
// display only features with five or more 'available-spots'map.setFilter( 'bike-docks' , [ '>=' , [ 'get' , 'available-spots' ], 5]);
```

```
// remove the filter for the 'bike-docks' style layermap.setFilter( 'bike-docks' , null );
```

setFreeCameraOptions(options, eventData?)

setLayerZoomRange(layerId, minzoom, maxzoom)

对于使用矢量源的样式层，如果样式层的缩放级别低于源层的最小缩放级别，样式层将无法渲染，因为这些缩放级别上不存在数据。如果源层的最小缩放级别高于样式层定义的最小缩放级别，则样式层将不会在缩放范围的所有级别上显示。

参数：

layerId:string应用缩放级别的图层的标识符。

minzoom:number设置的最小缩放级别（0-24）。

maxzoom:number设置的最大缩放级别（0-24）。

```
map.setLayerZoomRange( 'my-layer' , 2, 5);
```

setLayoutProperty(layerId, name, value, options = {})

参数：

layerId:string设置属性的图层的标识符。

name:string设置的属性的名称。

value:any属性的值。必须具有符合该属性的类型，如样式规范中定义。

options:object 默认为{} 选项对象。

名称　　描述

vali

```
map.setLayoutProperty( 'my-layer' , 'visibility' , 'none' );
```

setLight(light, options = {})

参数：

light:LightSpecification照明属性。必须符合样式规范。

options:object 默认为{} 选项对象。

名称 描述

vali

```
var layerVisibility = map.getLayoutProperty( 'my-layer' , 'visibility' );
```

setMaxBounds(bounds)

平移和缩放操作受到这些边界的限制。如果进行的平移或缩放显示了这些边界以外的区域，地图将显示尽可能接近操作请求的位置和缩放级别，同时仍然保持在这些边界内。

参数：

bounds: LngLatBoundsLike **| null | undefined** 设置的最大边界。如果指定为 null 或 null，则该函数将删除地图的最大边界。

```
// Define bounds that conform to the `LngLatBoundsLike` object.var bounds =
[  ``[-74.04728, 40.68392], // [west, south]  ``[-73.91058, 40.87764]  //
[east, north]];// Set the map's max bounds.map.setMaxBounds(bounds);
```

setMaxPitch(maxPitch)

setMaxZoom(maxZoom)

参数：

maxZoom:number | null | undefined 设置的最大缩放级别。如果指定为 null 或 undefined，则该函数将删除当前的最大缩放级别（将其设置为22）。

```
map.setMaxZoom(18.75);
```

setMinPitch(minPitch)

setMinZoom(minZoom)

并不总是能够缩小并达到设置的 minZoom。其他因素，如地图的高度，可能会限制缩放。例如，如果地图的高度为512像素，则无论 minZoom 设置为何值，缩放级别低于 zoom 0 都将是不可能的。

参数：

minZoom:number | null | undefined 设置的最小缩放级别（-2 - 24）。如果指定为 null 或 undefined，则该函数将删除当前的最小缩放级别（即将其设置为 -2）。

```
map.setMinZoom(12.25);
```

setPadding(padding, eventData?)

相当于 jumpTo({padding: padding})。

参数：

padding：PaddingOptions 所需的边距。格式：{ left: number, right: number, top: number, bottom: number }

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
// Sets a left padding of 300px, and a top padding of 50pxmap.setPadding({ left: 300, top: 50 });
```

setPaintProperty(layerId, name, value, options = {})

参数：

layerId:string 设置属性的图层的标识符。

name:string 要设置的属性的名称。

value:any属性的值。必须具有符合该属性的类型，如样式规范中定义。

options:object 默认为{} 选项对象。

名称　　描述

vali

```
map.setPaintProperty( 'my-layer' , 'fill-color' , '#faafee' );
```

setPitch(pitch, eventData?)

setRenderWorldCopies(renderWorldCopies)

参数：

renderWorldCopies：boolean如果为 true，则世界的多个副本将在-180度和180度经度之外并排显示。如果设置为 false：

- 当地图放大到世界的任何图像都无法填满地图容器时，在180度和-180度经度之外将会留下空白空间。
- 横跨180度和-180度经度的对象将在每个缩放级别被切成两半（地图的右边缘和左边缘各有一部分）。

undefinedundefined 被解释为 true，null—被解释为 false。

```
map.setRenderWorldCopies( true );
```

setStyle(style, options?)

如果在使用这个参数时样式已经设置，并且 options.diff 的值为 true，地图渲染工具将尝试比较给定的样式与地图的当前状态，并且只执行那些使地图样式符合期望状态所必需的更改。在精灵（用于图标和图案的图像）和字形（标签文字的字体）中的更改无法被区分。如果当前样式和给定样式中使用的精灵或字体有任何差异，地图渲染器将强制执行完全更新，删除当前样式并从头构建给定样式。

参数：

style:StyleSpecification | string | null 符合样式规范中描述的模式的 JSON 对象，或者是该 JSON 的 URL 地址。

options:Object对象选项

名称	描述
diff:booleandefault: true	如果为 false，强制执行完整更新，删除当前样式并创建新样式，而不是尝试基于 diff 来更新它。
localIdeographFontFamily:stringdefault: ‘sans-serif’	定义CSS字体族，用于在“CJK Unified Ideographs”、“Hiragana”、“Katakana”和“Hangul Syllables”范围内本地覆盖生成字形。在这些范围内，将忽略地图样式中的字体设置，除了字重关键词（light/regular/medium/bold）。设置为 false 以启用这些字形范围内的地图样式字体设置。强制执行完整更新。

```
map.setStyle( "path-to-style" );
```

setTerrain(terrain)

参数：

terrain:TerrainSpecification设置的属性。必须符合样式规范。如果指定了 null 或 undefined，函数将删除 terrain。

```
map.addSource( 'mmr-dem' , {  ``'type' :  'raster-dem' ,  ``'url' :  'path-to-source' ,  ``'tileSize' : 512,  ``'maxzoom' : 14});// add the DEM source as a terrain layer with exaggerated heightmap.setTerrain({  'source' :  'mmr-dem' ,  'exaggeration' : 1.5 });
```

setZoom(zoom, eventData?)

参数：

zoom:number设置的缩放级别（0-20）。

eventData:Object将被添加到由此方法触发的事件对象的附加属性。

```
// Zoom to the zoom level 5 without an animated transitionmap.setZoom(5);
```
showCollisionBoxes
showPadding
showTerrainWireframe

框架总是红色的，只在活动地形时绘制。

```
map.showTerrainWireframe = true;
```
showTileBoundaries

第一个矢量源的未压缩文件大小显示在每个瓦片的左上角，靠近瓦片的标识符。

```
map.showTileBoundaries = true;
```
snapToNorth(options?, eventData?)
stop()
touchPitch
touchZoomRotate
triggerRepaint()

```
map.triggerRepaint();
```
unproject(point)

参数：

point: PointLike 用于 unproject 的像素坐标unproject

```
map.on('click', function(e) {  ``// When the map is clicked, get the geographic coordinate.  ``var coordinate = map.unproject(e.point);});
```
updateImage(id, image)

参数：

id:stringid:string 图像的标识符。

image:(HTMLImageElement | ImageBitmap | ImageData | {width: number, height: number, data: (Uint8ClampedArray | Uint8ClampedArray)} | StyleImageInterface) 以 HTMLImageElement、ImageData、ImageBitmap 形式的图像，或具有 width、height 和 data 属性的对象，格式与 ImageData 相同。

```
// If an image with the ID 'cat' already exists in the style's sprite,// replace that image with a new image, 'other-cat-icon.png'.if(map.hasImage('cat'))
map.updateImage('cat', './other-cat-icon.png');
```
version
zoomIn(options?, eventData?)

参数：

options:AnimationOptions 选项对象

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
// zoom the map in one level with a custom animation durationmap.zoomIn({duration: 1000});
```

zoomOut(options?, eventData?)

参数：

options:AnimationOptions 选项对象

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
// zoom the map out one level with a custom animation offsetmap.zoomOut({offset: [80, 60]});
```

zoomTo(zoom, options, eventData?)

参数：

zoom:number用于转换的缩放级别。

options:AnimationOptions 选项对象

eventData 附加属性，添加到由此方法初始化的事件对象中。

```
// Zoom to the zoom level 5 without an animated transitionmap.zoomTo(5);// Zoom to the zoom level 8 with an animated transitionmap.zoomTo(8, {  ``duration: 2000,  ``offset: [100, 50]});
```

事件

名称	描述
boxzoomcancel	当用户取消“框缩放”交互或者当限制矩形不符合最小尺寸阈值时触发。参见 BoxZoomHandler。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// the user cancels a "box zoom" interaction.map.on( 'boxzoomcancel', function () {  ``console.log( 'A boxzoomcancel event occurred.' );});
```

boxzoomend

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just after a "box zoom" interaction ends.map.on( 'boxzoomend', function () {  ``console.log( 'A boxzoomend event occurred.' );});
```

boxzoomstart

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just before a "box zoom" interaction starts.map.on( 'boxzoomstart', function () { ``console.log( 'A boxzoomstart event occurred.' );});
```

click

此事件与可选参数 layerId 兼容。如果在 Map#on 中作为第二个参数包含了 layerId，事件监听器只有在被点击的点包含指定图层的可见部分时才会触发。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listenermap.on( 'click', function (e) { ``console.log( 'A click event has occurred at ' + e.lngLat);});
```

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener for a specific layermap.on( 'click', 'poi-label', function (e) { ``console.log( 'A click event has occurred on a visible portion of the poi-label layer at ' + e.lngLat);});
```

contextmenu

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when the right mouse button is// pressed within the map.map.on( 'contextmenu', function () { ``console.log( 'A contextmenu event occurred.' );});
```

data

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when map data loads or changes.map.on( 'data', function () { ``console.log( 'A data event occurred.' );});
```

dataloading

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when any map data begins loading// or changing asynchronously.map.on( 'dataloading', function () { ``console.log( 'A dataloading event occurred.' );});
```

dbclick

此事件与可选参数 layerId 兼容。如果在 Map#on 中作为第二个参数包含了 layerId，事件监听器只有在双击的点包含指定图层的可见部分时才会触发。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listenermap.on( 'dblclick' , function (e) { ``console.log( 'A dblclick event has occurred at ' + e.lngLat);});
```

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener for a specific layermap.on( 'dblclick' , 'poi-label' , function (e) { ``console.log( 'A dblclick event has occurred on a visible portion of the poi-label layer at ' + e.lngLat);});
```

drag

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// repeatedly during a "drag to pan" interaction.map.on( 'drag' , function () { ``console.log( 'A drag event occurred.' );});
```

dragend

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when a "drag to pan" interaction ends.map.on( 'dragend' , function () { ``console.log( 'A dragend event occurred.' );});
```

dragstart

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when a "drag to pan" interaction starts.map.on( 'dragstart' , function () { ``console.log( 'A dragstart event occurred.' );});
```

error

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when an error occurs.map.on( 'error' , function () { ``console.log( 'A error event occurred.' );});
```

idle

- 没有发生任何摄像机转换
- 当前所有请求的瓦片都已加载
- 所有的渐变/过渡动画都已完成

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just before the map enters an "idle" state.map.on( 'idle' , function () { ``console.log( 'A idle event occurred.' );});
```

load

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when the map has finished loading.map.on( 'load' , function () {  ``console.log( 'A load event occurred.' );});
```

mousedown

此事件与可选参数 layerId 兼容。如果在 Map#on 中作为第二个参数包含了 layerId，当点击发生在指定图层的可视部分内时，事件监听器将只会被触发。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listenermap.on( 'mousedown' , function () {  ``console.log( 'A mousedown event has occurred.' );});
```

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener for a specific layermap.on( 'mousedown' , 'poi-label' , function () {  ``console.log( 'A mousedown event has occurred on a visible portion of the poi-label layer.' );});
```

mouseenter

此事件只能在 Map#on 包含三个参数时监听，其中第二个参数指定所需图层。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listenermap.on( 'mouseenter' , 'water' , function () {  ``console.log( 'A mouseenter event occurred on a visible portion of the water layer.' );});
```

mouseleave

此事件只能在 Map#on 包含三个参数时监听，其中第二个参数指定所需图层。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when the pointing device leaves// a visible portion of the specified layer.map.on( 'mouseleave' , 'water' , function () {  ``console.log( 'A mouseleave event occurred.' );});
```

mousemove

此事件与可选参数 layerId 兼容。如果在 Map#on 中作为第二个参数包含了 layerId，事件监听器将只会在光标位于指定图层的可见部分时被触发。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listenermap.on( 'mousemove' , function () {  ``console.log( 'A mousemove event has occurred.' );});
```

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener for a specific layermap.on( 'mousemove' , 'poi-label' , function () {  ``console.log( 'A mousemove event has occurred on a visible portion of the poi-label layer.' );});
```

mouseout

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when the pointing device leave's// the map's canvas.map.on( 'mouseout' , function () {  ``console.log( 'A mouseout event occurred.' );});
```

mouseover

此事件与可选参数 layerId 兼容。如果在 Map#on 中作为第二个参数包含了 layerId，事件监听器将只会在光标移动到指定图层的可见部分时被触发。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listenermap.on( 'mouseover' , function () {  ``console.log( 'A mouseover event has occurred.' );});
```

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener for a specific layermap.on( 'mouseover' , 'poi-label' , function () {  ``console.log( 'A mouseover event has occurred on a visible portion of the poi-label layer.' );});
```

mouseup

此事件与可选参数 layerId 兼容。如果在 Map#on 中作为第二个参数包含了 layerId，当光标被释放时，如果位于指定图层的可见部分内，事件监听器才会被触发。

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listenermap.on( 'mouseup' , function () {  ``console.log( 'A mouseup event has occurred.' );});
```

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener for a specific layermap.on( 'mouseup' , 'poi-label' , function () {  ``console.log( 'A mouseup event has occurred on a visible portion of the poi-label layer.' );});
```

move

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// repeatedly during an animated transition.map.on( 'move' , function () {  ``console.log( 'A move event occurred.' );});
```

moveend

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just after the map completes a transition.map.on( 'moveend' , function () {  ``console.log( 'A moveend event occurred.' );});
```

movestart

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just before the map begins a transition// from one view to another.map.on( 'movestart' , function () {  ``console.log( 'A movestart` event occurred.' );});
```

pitch

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// repeatedly during a pitch (tilt) transition.map.on( 'pitch' , function () {  ``console.log( 'A pitch event occurred.' );});
```

pitchend

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just after a pitch (tilt) transition ends.map.on( 'pitchend' , function () {  ``console.log( 'A pitchend event occurred.' );});
```

pitchstart

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just before a pitch (tilt) transition starts.map.on( 'pitchstart' , function () {  ``console.log( 'A pitchstart event occurred.' );});
```

remove

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just after the map is removed.map.on( 'remove' , function () { ``console.log( 'A remove event occurred.' );});
```

render

- 地图位置、缩放、高度调整或方位角的改变
- 地图样式的更改
- GeoJSON 数据源的更改JSON
- 矢量瓦片、GeoJSON 文件、字形或精灵的加载

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// whenever the map is drawn to the screen.map.on( 'render' , function () { ``console.log( 'A render event occurred.' );});
```

resize

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// immediately after the map has been resized.map.on( 'resize' , function () { ``console.log( 'A resize event occurred.' );});
```

rotate

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// repeatedly during "drag to rotate" interaction.map.on( 'rotate' , function () { ``console.log( 'A rotate event occurred.' );});
```

rotateend

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just after a "drag to rotate" interaction ends.map.on( 'rotateend' , function () { ``console.log( 'A rotateend event occurred.' );});
```

rotatestart

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just before a "drag to rotate" interaction starts.map.on( 'rotatestart' , function () { ``console.log( 'A rotatestart event occurred.' );});
```

sourcedata

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when one of the map's sources loads or
```

```
changes.map.on('sourcedata', function() {  ``console.log('A sourcedata event occurred.');});
```

sourcedataloading

```
// Initialize the mapvar map = new mmrgl.Map({  // map options });// Set an event listener that fires// map's sources begin loading or// changing asyncronously.map.on('sourcedataloading', function() {  ``console.log('A sourcedataloading event occurred.');});
```

styledata

```
// Initialize the mapvar map = new mmrgl.Map({  // map options });// Set an event listener that fires// when the map's style loads or changes.map.on('styledata', function() {  ``console.log('A styledata event occurred.');});
```

styledataloading

```
// Initialize the mapvar map = new mmrgl.Map({  // map options });// Set an event listener that fires// map's style begins loading or// changing asyncronously.map.on('styledataloading', function() {  ``console.log('A styledataloading event occurred.');});
```

styleimagemissing

```
// Initialize the mapvar map = new mmrgl.Map({  // map options });// Set an event listener that fires// an icon or pattern is missing.map.on( 'styleimagemissing' , function () {  ``console.log( 'A styleimagemissing event occurred.' );});
```

touchcancel

```
// Initialize the mapvar map = new mmrgl.Map({  // map options });// Set an event listener that fires// when a touchcancel event occurs within the map.map.on( 'touchcancel' , function () {  ``console.log( 'A touchcancel event occurred.' );});
```

touchend

```
// Initialize the mapvar map = new mmrgl.Map({  // map options });// Set an event listener that fires// when a touchstart event occurs within the map.map.on( 'touchstart' , function () {  ``console.log( 'A touchstart event occurred.' );});
```

touchmove

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when a touchmove event occurs within the map.map.on( 'touchmove', function () { ``console.log( 'A touchmove event occurred.' );});
```

touchstart

参数:

data:MapMouseEvent

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when a touchstart event occurs within the map. map.on('touchstart', function() { ``console.log( 'A touchstart event occurred.' );});
```

webglcontextlost

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when the WebGL context is lost.map.on( 'webglcontextlost', function () { ``console.log( 'A webglcontextlost event occurred.' );});
```

webglcontextrestored

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when the WebGL context is restored.map.on( 'webglcontextrestored', function () { ``console.log( 'A webglcontextrestored event occurred.' );});
```

wheel

参数:

data:MapMouseEvent

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// when a wheel event occurs within the map.map.on( 'wheel', function () { ``console.log( 'A wheel event occurred.' );});
```

zoom

参数:

data:(MapMouseEvent | MapTouchEvent)

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// repeatedly during a zoom transition.map.on( 'zoom', function () { ``console.log( 'A zoom event occurred.' );});
```

zoomend

参数:

data:(MapMouseEvent | MapTouchEvent)

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just after a zoom transition finishes.map.on( 'zoomend' , function () { ``console.log( 'A zoomend event occurred.' );});
```

zoomstart

参数:

data:(MapMouseEvent | MapTouchEvent)

```
// Initialize the mapvar map = new mmrgl.Map({ // map options });// Set an event listener that fires// just before a zoom transition starts.map.on( 'zoomstart' , function () { ``console.log( 'A zoomstart event occurred.' );});
```

2. 属性和选项

accessToken:string

访问地图的令牌。The event type。

src/index.js

实例:

```
mmrgl.accessToken = myAccessToken
```

baseApiUrl:string

用于获取瓦片、样式、精灵和字形的初始 API URL

src/index.js

实例:

```
mmrgl.baseApiUrl = 'https://geo.rustore.ru/api' ;
```

workerCount:number

GL JS 地图页面上的网络工作者数量。默认是核心数的一半（限制为6）。在初始化地图之前必须设置此参数。

src/index.js

实例:

```
mmrgl.workerCount = 2;
```

maxParallelImageRequests:number

最大图像数量（栅格瓦片、精灵、图标）并行加载。影响栅格地图的性能。默认为16。

src/index.js

实例:

```
mmrgl.maxParallelImageRequests = 10;
```

supported:function

检查浏览器是否支持 GL JSGLJS

src/index.js

实例:

```
if (!mmrgl.supported()) {  ``alert( 'Your browser does not support MMR GL' );}
```

version:string

当前构建版本MMRGL

src/index.js

实例:

```
console.log(mmrgl.version);//> 1.2.3;
```

setRTLTextPlugin:function

安装支持 RTL 的插件。这对于支持阿拉伯语和希伯来语是必要的。

src/index.js

参数:

pluginURL:string—RTL 插件的路径。

callback:function(error:object) — 如果存在错误，将调用此函数。

lazy:boolean—延迟加载。

实例:

```
mmrgl.setRTLTextPlugin(pluginUrl, function (error) {    ``if (error)
{        ``console.log( 'something was wrong' , error);    ``} else
{        ``console.log( 'rtl-text-plugin loaded successfully' );    ``}}, true );
```

getRTLTextPluginStatus:function

获取 RTL 插件的状态。插件状态可以是：unavailable（即未请求或已删除）、loading、loaded 或 error。如果状态为 loaded 并且插件再次被请求，则会出现错误。

src/index.js

实例:

```
const pluginStatus = mmrgl.getRTLTextPluginStatus();
```

clearStorage:function

清空可能存储瓦片缓存的 cacheStorage

src/index.js

参数：

callback:function(error:object) — RTL 插件的路径。

实例:

```
mmrgl.clearStorage()
```

AnimationOptions(参数组)

影响动画的参数（用于 Map#panBy, Map#easeTo 方法），控制动画的持续时间和平滑函数（easing function）。这些参数是可选的。

src/ui/camera.js

参数：

duration:number—动画持续时间（以毫秒为单位）。

easing:function—一个接收时间范围在 0..1 的函数，并返回一个数字，其中 0 表示初始状态，1 表示最终状态。

offset:PointLike—动画结束时相对于地图容器实际中心的中心偏移。

animate:boolean—如果为 false，则动画将被禁用。

essential:boolean—如果为 true，则动画被认为是关键的，不会受到 prefers-reduced-motion（更喜欢减少动作）的影响。

CameraOptions(参数组)

影响摄像机的参数（用于 Map#jumpTo、Map#easeTo 和 Map#flyTo 方法），控制 location（起始位置）、zoom（缩放）、摄像机的bearing和pitch。所有属性都是可选的。

src/ui/camera.js

参数：

center:LngLatLike—地图中心。

zoom:number—地图的远近（缩放）。

bearing:number—所需的方位角（度）。方位角—指罗盘的“向上”方向。例如，方位角：90将使地图以东为上。

pitch:number—期望的地图倾斜度（以度为单位）。倾斜角 — 与地平线的角度，范围从0到60度。例如，pitch: 0 提供从上往下直视地图的视角，而 pitch: 60 则将用户的视角倾斜至地平线。增加倾斜值通常用于显示3D对象。

around:LngLatLike—如果设置了 zoom，around 定义了缩放中心点。

padding:PaddingOptions—应用于viewport每一侧，用于偏移消失点（当地图倾斜时适用）。

PaddingOptions(参数组)

边距参数（用于 Map#fitBounds、Map#fitScreenCoordinates 和 Map#setPadding 方法）。调整这些参数以设置地图边缘添加的像素数量的边距。这个对象的所有属性必须是非负整数。

src/ui/camera.js

参数：

top:number—地图顶部的像素边距。

bottom:number—地图底部的像素边距。

left:number—地图左侧的像素边距。

right:number—地图右侧的像素边距。

实例:

```
var bbox = [[-79, 43], [-73, 45]];map.fitBounds(bbox, {  ``padding: {top: 10, bottom:25, left: 15, right: 5}});
```

实例:

```
var bbox = [[-79, 43], [-73, 45]];map.fitBounds(bbox, {  ``padding: 20});
```

RequestParameters(参数组)

由 Map.options.transformRequest 的回调方法返回的对象Map.options.transformRequest

src/ui/ajax.js

参数：

url:string—URL请求的 URL。

headers:object—随请求发送的头部信息。

method:string—请求方法 ‘GET’ ’ | ‘POST’ ’ | ‘PUT’ 。

type:string—请求体响应（body response）的返回类型 ‘string’ ’ | ‘json’ ’ | ‘arrayBuffer’ 。

credentials:string—‘same-origin’ | ‘include’ 在跨域请求中使用 ‘include’ 发送库克（cookies）。

collectResourceTiming:boolean—如果为 true，则 Resource Timing API 的信息将可用于执行 GeoJSON 和 Vector Tile 的请求（这些信息通常无法从 JavaScript 的主线程获取）。信息将在 ResourceTiming 属性中返回。

实例:

```
transformRequest: function (url, resourceType) {    ``if (resourceType ===
'Source' && url.indexOf( 'http://myHost' ) > -1) {        ``return
{            ``url: url.replace( 'http' , 'https' ),            ``headers: {
'my-custom-header' : true },            ``credentials: 'include' // Include
cookies for cross-origin requests        ``}    ``}};
```

StyleImageInterface（开发者规范）

这不是一个方法或类。它是用于动态生成图像的接口。

实现这个接口的图像可以为每个帧重新绘制。它们可用于图标和图案的动画制作，或响应用户输入的数据。这些图像可以实现 StyleImageInterface#render 方法。该方法按帧调用。

src/style/style_image.js

参数:

width:number

height:number

data:(Uint8Array|Uint8ClampedArray)

实例:

```
var flashingSquare = {    ``width: 64,    ``height: 64,    ``data: new
Uint8Array(64 * 64 * 4),        ``onAdd: function (map) {        ``this .map =
map;    ``},        ``render: function () {        ``// keep repainting while
the icon is on the map        ``this .map.triggerRepaint();                ``//
alternate between black and white based on the time        ``var value =
Math.round(Date.now() / 1000) % 2 === 0  ? 255 : 0;                ``// check
if image needs to be changed        ``if (value !== this .previousValue)
{            ``this .previousValue = value;                        ``var
bytesPerPixel = 4;            ``for ( var x = 0; x < this .width; x++)
{                ``for ( var y = 0; y < this .height; y++)
{                    ``var offset = (y * this .width + x) *
bytesPerPixel;                    ``this .data[offset + 0] =
value;                    ``this .data[offset + 1] =
value;                    ``this .data[offset + 2] =
value;                    ``this .data[offset + 3] =
255;                ``}            ``}                        ``// return true
```

```
to indicate that the image changed            ``return true
;       ``}    ``}} map.addImage( 'flashing_square' , flashingSquare);
```

CustomLayerInterface（开发者规范）

这不是一个方法或类。它是用于自定义图层样式的接口。

自定义层允许用户直接在地图的 GL context 中使用摄像机进行可视化。这些层可以通过 Map#addLayer 添加到任何普通层之间。

自定义层必须具有唯一标识符和“custom”类型。它们必须实现 render 方法，并且可以实现 prerender、onAdd 和 onRemove。它们可以通过 Map#triggerRepaint 启动渲染，并且应适当地处理 Map.event:webglcontextlost 和 Map.event:webglcontextrestored 事件。

属性 RenderingMode 定义图层将被处理为“2d”或“3d”地图图层：

- “RenderingMode”：“3d” — 用于使用深度缓冲区并与其他图层共享。
- “RenderingMode”：“2d” — 用于添加无深度图层。如果您需要为“2d”图层使用深度缓冲区，您应使用帧后帧缓冲区和 CustomLayerInterface#prerenderprerender

src/style/style_layer/custom_style_layer.js

参数：

id:string—图层的唯一ID。

type:string—图层类型应为“custom”。

renderingMode:string—“3d”或“2d”，默认为“2d”。

实例：

```
// Custom layer implemented as ES6 classclass NullIslandLayer
{    ``constructor() {        ``this .id =  'null-island' ;        ``this .type =
'custom' ;        ``this .renderingMode =  '2d' ;    ``}        ``onAdd(map, gl)
{        ```const vertexSource = <br>`            uniform mat4
u_matrix;<br>     void main() {`<br>`               gl_Position = u_matrix *
vec4(0.5, 0.5, 0.0, 1.0);<br>       gl_PointSize =
20.0;`<br>`            ```}`;                ```const fragmentSource =
<br>`            void main() {<br>       gl_FragColor = vec4(1.0, 0.0, 0.0,
1.0);`<br>`            ```}`;                ``const vertexShader =
gl.createShader(gl.VERTEX_SHADER);        ``gl.shaderSource(vertexShader,
vertexSource);        ``gl.compileShader(vertexShader);        ``const
fragmentShader =
gl.createShader(gl.FRAGMENT_SHADER);        ``gl.shaderSource(fragmentShader,
fragmentSource);        ``gl.compileShader(fragmentShader);                ``th
is .program = gl.createProgram();        ``gl.attachShader( this .program,
vertexShader);        ``gl.attachShader( this .program,
fragmentShader);        ``gl.linkProgram( this
.program);    ``}        ``render(gl, matrix) {        ``gl.useProgram( this
```

```
.program);        ``gl.uniformMatrix4fv(gl.getUniformLocation( this .program,
"u_matrix" ), false , matrix);        ``gl.drawArrays(gl.POINTS, 0,
1);    ``}}     map.on( 'load' , function () {    ``map.addLayer( new
NullIslandLayer());});
```

prewarm:function

执行所有资源的“预热”（例如，初始化网络工作者），以减少地图加载时间。如果使用 mmrgl.workerUrl 和 mmrgl.workerCount，需要在调用 prewarm() 之前定义它们。

默认情况下，这些资源的管理是自动执行的，并且在首次创建地图时根据需要（懒加载）进行初始化。调用 prewarm() 后，资源将提前创建，并且在从页面上删除最后一张地图时不会被清除。这允许使用新的地图实例重新使用它们。可以通过调用 mmrgl.clearPrewarmedResources() 手动清除它们。只有当您的网页仍然活跃但完全停止使用地图时，才需要这样做。

这在单页面应用程序（SPA）中使用地图时非常有用，用户将在不同的页面/屏幕之间移动，这可能导致地图实例的持续创建和销毁。

src/index.js

实例:

```
mmrgl.prewarm()
```

clearPrewarmedResources:function

清除之前由 mmrgl.prewarm() 创建的资源。请注意，通常没有必要这样做。您只应该在预期您的应用程序的用户不会在任何时候返回查看地图的情况下调用此功能。

src/index.js

实例:

```
mmrgl.clearPrewarmedResources()
```

3. 标记和控件

Marker

创建标记

src/ui/marker.js

参数

名称	类型	描述
element	HTMLElement	使用 DOM 元素作为标记。默认情况下，使用浅蓝色，滴状 svg 标记 (light blue, droplet-shaped)。

名称	类型	描述
anchor	stringdefault: ‘center’	定义图标相对于其中心的位置。可用值包括：‘center’、‘top’、‘bottom’、‘left’、‘right’、‘top-left’、‘top-right’、‘bottom-left’ 和 ‘bottom-right’。
offset	PointLike	相对于元素中心应用的偏移。负值指向左和上。
color	stringdefault: ‘#3FB1CE’	如果未指定 options.element，则将使用此颜色。默认使用浅蓝色。
scale	numberdefault: 1	如果未指定 options.element，则将使用此比例。默认比例对应于高度 41px 和宽度 27px。
draggable	booleandefault: false	如果为 true，则可以拖放标记到新位置 (drag and drop)。
clickTolerance	numberdefault: 0	用户在点击标记时可以移动鼠标指针的最大像素数量，以便它被认为是有效点击（而不是拖动标记）。默认情况下，使用地图的 clickTolerance 继承。
rotation	numberdefault: 0	标记相对于其相应 rotationAlignment 的旋转角度。正值会顺时针旋转标记。
pitchAlignment	stringdefault: ‘auto’	- ‘map’ 将标记对齐到地图（map）的平面。- ‘viewport’ 将标记对齐到屏幕（viewport）的平面。- ‘auto’ 自动对应于 rotationAlignment 的值。
rotationAlignment	stringdefault: ‘auto’	rotationAlignment - ‘map’ 将标记的旋转对齐到地图（map），在地图旋转时保持方位角。-

名称	类型	描述
		‘viewport’ 将标记的旋转对齐到屏幕（viewport），不受地图旋转的影响。- ‘auto’ 等同于屏幕（viewport）。

实例:

```
var marker = new mmrgl.Marker() ``.setLngLat([30.5, 50.5]) ``.addTo(map);// Set optionsvar marker = new mmrgl.Marker({ ``color: "#FFFFFF", ``draggable: true}).setLngLat([30.5, 50.5]) ``.addTo(map);
```

方法

名称	描述
addTo(map)	将标记附加到地图对象上。

实例:

```
var marker = new mmrgl.Marker().setLngLat([30.5, 50.5]).addTo(map); // add the marker to the map
```

getElement()

getLngLat()

getOffset()

getPitchAlignment()

getPopup()

实例:

```
var marker = new mmrgl.Marker().setLngLat([0, 0]).setPopup( new mmrgl.Popup().setHTML( "<h1>Hello World!</h1>" )).addTo(map); console.log(marker.getPopup()); // return the popup instance
```

getRotation()

getRotationAlignment()

isDraggable()

remove()

实例:

```
var marker = new mmrgl.Marker().addTo(map);marker.remove();
```

setDraggable(shouldBeDraggable)

setLngLat(lnglat)

参数:

lnglat:LngLat，标记应放置的位置。

实例:

```
// Create a new marker, set the longitude and latitude, and add it to the mapnew
mmrgl.Marker().setLngLat([-65.017, -16.457]).addTo(map);
```

setOffset(offset)

setPitchAlignment(alignment?)

setPopup(popup)

参数:

popup:Popup弹出类的实例。如果值为 undefined 或 null，则设置在此标记实例上的任何弹出窗口都将被移除。

实例:

```
var marker = new mmrgl.Marker() ``.setLngLat([0, 0]) ``.setPopup( new
mmrgl.Popup().setHTML( "<h1>Hello World!</h1>" )) // add popup ``.addTo(map);
```

setRotation(rotation)

setRotationAlignment(alignment)

togglePopup()

实例:

```
var marker = new mmrgl.Marker() ``.setLngLat([0, 0]) ``.setPopup( new
mmrgl.Popup().setHTML( "<h1>Hello World!</h1>"
)) ``.addTo(map); marker.togglePopup(); // toggle popup open or closed
```

事件

名称	描述
drag	在拖动时触发
dragend	当拖动结束时触发
dragstart	在开始拖动时触发

Popup

创建弹出块。

src/ui/popup.js

参数

名称	类型	描述
maxWidth	stringdefault: ‘240px’	定义弹出窗口最大宽度的CSS属性的字符串，例如‘300px’。要确保弹出窗口

名称	类型	描述
		的大小与其内容相匹配，请将此属性设置为 “none”。可用值可以在这里找到：https://developer.mozilla.org/en-US/docs/Web/CSS/max-width
anchor	stringdefault: ‘bottom’	定义图标相对于其中心的位置。可用值包括：‘center’、‘top’、‘bottom’、‘left’、‘right’、‘top-left’、‘top-right’、‘bottom-left’ 和 ‘bottom-right’。
offset	number | PointLike | Object	应用于弹出窗口位置的像素偏移，指定为：- 数字（number），指示距离弹出窗口位置的距离- PointLike，指示固定偏移（constant offset）- 形式为 {‘center’: PointLike, ‘top’: PointLike…} 的点对象，为每个 anchor 位置指定偏移。负偏移表示向左和向上。
className	string	类名用于容器
focusAfterOpen	booleandefault: true	如果为 true，则在打开窗口后，将尝试将焦点（光标）设置在窗口内的第一个支持此功能的元素上，例如。
closeOnMove	booleandefault: false	如果为 true，则在移动地图时会关闭弹出窗口。
closeOnClick	booleandefault: true	如果为 true，则在点击地图时会关闭弹出窗口。
closeButton	booleandefault: true	如果为 true，则在弹出窗口的右上角将出现一个关闭按钮。

实例：

```
var markerHeight = 50, markerRadius = 10, linearOffset = 25;var popupOffsets = {  ``'top' : [0, 0],  ``'top-left' : [0,0],  ``'top-right' : [0,0],  ``'bottom' : [0, -markerHeight],  ``'bottom-left' : [linearOffset, (markerHeight - markerRadius + linearOffset) * -1],  ``'bottom-right' : [-linearOffset, (markerHeight - markerRadius + linearOffset) * -1],  ``'left' : [markerRadius, (markerHeight - markerRadius) * -1],  ``'right' : [-markerRadius, (markerHeight - markerRadius) * -1]};var popup =  new mmrgl.Popup({offset: popupOffsets, className:  'my-class' })  ``.setLngLat(e.lngLat)  ``.setHTML( "<h1>Hello World!</h1>" )  ``.setMaxWidth( "300px" )  ``.addTo(map);
```

方法

名称	描述
addClassName(className)	向弹出容器元素添加CSS类。

参数：

className:string非空的类名字符串，用于添加到弹出容器

实例：

```
let popup =  new mmrgl.Popup()popup.addClassName( 'some-class' )
```

addTo(map)

实例：

```
new mmrgl.Popup().setLngLat([0, 0]).setHTML( "<h1>Null Island</h1>" ).addTo(map);
```

getElement()

实例：

```
// Change the `Popup` element's font sizevar popup =  new mmrgl.Popup()  ``.setLngLat([-96, 37.8])  ``.setHTML( "<p>Hello World!</p>" )  ``.addTo(map);var popupElem = popup.getElement();popupElem.style.fontSize = "25px" ;
```

getLngLat()

getMaxWidth()

isOpen()

remove()

实例：

```
var popup =  new mmrgl.Popup().addTo(map);popup.remove();
```

removeClassName(className)

参数：

className:string非空的CSS类名字符串，用于从弹出容器中删除

实例:

```
let popup = new mmrgl.Popup()popup.removeClassName( 'some-class' )
```

setDOMContent(htmlNode)

参数:

htmlNode:Node将作为弹出窗口内容的DOM节点。

实例:

```
// create an element with the popup contentvar div =
window.document.createElement( 'div' );div.innerHTML = 'Hello, world!' ;var popup
= new
mmrgl.Popup() ``.setLngLat(e.lngLat) ``.setDOMContent(div) ``.addTo(map);
```

setHTML(html)

此方法不执行HTML过滤或消毒，应仅与可信内容一起使用。如果内容是不可信的文本字符串，请考虑使用Popup#setText。

参数:

html:string 代表弹出窗口的HTML内容的字符串。

实例:

```
var popup = new mmrgl.Popup() ``.setLngLat(e.lngLat) ``.setHTML( "<h1>Hello
World!</h1>" ) ``.addTo(map);
```

setLngLat(lnglat)

setMaxWidth(maxWidth)

setOffset(offset?)

setText(text)

这个函数在DOM中创建一个文本节点，因此它不能插入原始HTML。如果弹出内容是用户提供的，使用这个方法可以防止XSS攻击。

参数:

text:string弹出窗口的文本内容。

实例:

```
var popup = new mmrgl.Popup() ``.setLngLat(e.lngLat) ``.setText( 'Hello,
world!' ) ``.addTo(map);
```

toggleClassName(className)

参数:

className:string一个非空的CSS类名字符串，用于添加/删除。

实例:

```
let popup = new mmrgl.Popup()popup.toggleClassName( 'toggleClass' )
```

trackPointer()

实例:

```
var popup = new mmrgl.Popup({ closeOnClick: false , closeButton: false
}) ``.setHTML( "<h1>Hello World!</h1>" ) ``.trackPointer() ``.addTo(map);
```

事件

名称	描述
close	当弹出窗口手动或程序性地关闭时触发。

实例:

```
// Create a popupvar popup = new mmrgl.Popup();// Set an event listener that
will fire// any time the popup is closedpopup.on( 'close' , function
(){ ``console.log( 'popup was closed' );});
```

open

实例:

```
// Create a popupvar popup = new mmrgl.Popup();// Set an event listener that
will fire// any time the popup is openedpopup.on( 'open' , function
(){ ``console.log( 'popup was opened' );});
```

IControl（开发者规范）

用于添加到地图上的交互式控件的界面。这是给开发者的规范：它不是方法或类。

控件必须实现 onAdd 和 onRemove，也必须是 。

。要使用 MMR GL JS 控件的样式，请添加 mmrgl-ctrl 类。

src/ui/map.js

实例:

```
// Control implemented as ES6 classclass HelloWorldControl {    ``onAdd(map)
{       ``this ._map = map;       ``this ._container = document.createElement(
'div' );       ``this ._container.className = 'mmrgl-ctrl' ;       ``this
._container.textContent = 'Hello, world' ;       ``return this
._container;    ``}    ``onRemove() {       ``this
._container.parentNode.removeChild( this ._container);       ``this ._map =
undefined;    ``}} // Control implemented as ES5 prototypical classfunction
HelloWorldControl() { } HelloWorldControl.prototype.onAdd = function (map)
{    ``this ._map = map;    ``this ._container = document.createElement( 'div'
);    ``this ._container.className = 'mmrgl-ctrl' ;    ``this
```

```
._container.textContent = 'Hello, world';    ``return this
._container;}; HelloWorldControl.prototype.onRemove = function () {    ``this
._container.parentNode.removeChild( this ._container);    ``this ._map =
undefined;};
```

方法

名称	描述
getDefaultPosition()	如果需要，请指定此控件的默认位置。如果实现了这个方法并且在调用 Map#addControl 时没有指定 position 参数，那么将使用 getDefaultPosition 返回的值作为控件的位置。
onAdd(map)	在地图上注册控件并允许其注册事件监听器和资源。这个方法由 Map#addControl 内部调用。
onRemove(map)	在地图上注销控件并允许其解除事件监听器和资源的绑定。这个方法由 Map#removeControl 内部调用。

NavigationControl

NavigationControl控件包含缩放（zoom）按钮和指南针（compass）。

src/ui/control/navigation_control.js

参数

名称	类型	描述
showCompass	booleandefault: true	如果为true，则显示指南针按钮。
showZoom	booleandefault: true	如果为true，则显示放大和缩小按钮。
visualizePitch	booleandefault: false	如果为true，则通过旋转X轴来可视化步骤。

实例:

```
var nav = new mmrgl.NavigationControl();map.addControl(nav, 'top-left');
```

GeolocateControl

GeolocateControl控件提供了一个按钮，该按钮使用浏览器的地理定位API来定位用户在地图上的位置。

并非所有浏览器都支持地理定位，一些用户可能会禁用此功能。对于现代浏览器（包括Chrome）的地理定位支持，要求网站通过HTTPS协议提供服务。如果地理定位支持不可用，则GeolocateControl将显示为禁用状态。

缩放级别将取决于设备提供的地理定位精度。

GeolocateControl有两种模式。如果trackUserLocation为false（默认值），则该控件作为一个按钮，单击时会将地图相机设置到用户的位置。如果用户移动，地图不会更新。这更适合桌面使用。如果trackUserLocation为true，则控件作为一个切换按钮，跟踪用户的移动。在这种模式下，GeolocateControl有三种交互状态：

- ‘active’ — 当用户的位置发生变化时，地图相机会自动更新，保持定位点居中。
- ‘passive’ — 用户的定位点会自动更新，但地图相机不会。这种情况发生在用户手动移动地图时。
- ‘disabled’ — 如果地理定位不可用、被禁用或被拒绝，则会发生此状态。

这些交互状态不能通过程序控制，而是基于用户的交互来设置。

src/ui/control/geolocate_control.js

参数

名称	类型	描述
positionOptions	Objectdefault: { enableHighAccuracy:false, timeout:6000}	地理定位 API 的 PositionOptions 对象。
fitBoundsOptions	Objectdefault: { maxZoom:15}	MapMap#fitBounds Options 对象，当地图移动并缩放到用户的位置时使用。默认使用maxZoom 15，以限制地图的缩放级别，以适应非常精确的位置。
trackUserLocation	booleandefault: false	如果为true，则Geolocate控件变为开关，当它被激活时，地图将根据用户位置的变化获得更新。
showAccuracyCircle	booleandefault: true	默认情况下，如果 showUserLocation 的值为 true，在用户位置周围将绘制一个透明的圆圈，表示用户位置的准确性（准确性为 95%）。设置为 false 以禁用此功能。当 showUserLocation 的值为 false 时，此功能总是被禁用。

名称	类型	描述
showUserLocation	booleandefault: true	默认情况下，用户位置处将在地图上显示一个点。设置为 false 以禁用此功能。设置为 false 以禁用此功能。

实例:

```
map.addControl( new mmrgl.GeolocateControl({    ``positionOptions:
{        ``enableHighAccuracy:   true    ``},    ``trackUserLocation:   true}));
```

方法

名称	描述
trigger	编程请求并将地图移动到用户位置。

实例:

```
// Initialize the geolocate control.var geolocate =   new
mmrgl.GeolocateControl({    ``positionOptions: {        ``enableHighAccuracy:
true    ``},    ``trackUserLocation:   true});// Add the control to the
map.map.addControl(geolocate);map.on( 'load' ,   function ()
{    ``geolocate.trigger();});
```

事件

名称	描述
error	每次地理定位 API 更新位置时触发，并且这个更新返回错误。

参数:

data:PositionError从 Geolocation.getCurrentPosition() 或 Geolocation.watchPosition() 的回调中返回的 PositionError 对象。

实例:

```
// Initialize the geolocate control.var geolocate =   new
mmrgl.GeolocateControl({    ``positionOptions: {        ``enableHighAccuracy:
true    ``},    ``trackUserLocation:   true});// Add the control to the
map.map.addControl(geolocate);// Set an event listener that fires// when an error
event occurs.geolocate.on( 'error' ,   function () {    ``console.log( 'An error
event has occurred.' )});
```

geolocate

参数:

data:Position从 Geolocation.getCurrentPosition() 或 Geolocation.watchPosition() 调用中返回的 Position 对象。

实例:

```
// Initialize the geolocate control.var geolocate =  new mmrgl.GeolocateControl({   ``positionOptions: {       ``enableHighAccuracy: true   ``},   ``trackUserLocation:  true});// Add the control to the map.map.addControl(geolocate);// Set an event listener that fires// when a geolocate event occurs.geolocate.on( 'geolocate' ,  function () {   ``console.log( 'A geolocate event has occurred.' )});
```

outofmaxbounds

参数:

data:Position从 GeoPosition 或 Geolocation.getCurrentPosition() 调用中返回的 location.watchPosition() 对象。

实例:

```
// Initialize the geolocate control.var geolocate =  new mmrgl.GeolocateControl({   ``positionOptions: {       ``enableHighAccuracy: true   ``},   ``trackUserLocation:  true});// Add the control to the map.map.addControl(geolocate);// Set an event listener that fires// when an outofmaxbounds event occurs.geolocate.on( 'outofmaxbounds' ,  function () {   ``console.log( 'An outofmaxbounds event has occurred.' )});
```

trackuserlocationend

实例:

```
// Initialize the geolocate control.var geolocate =  new mmrgl.GeolocateControl({   ``positionOptions: {       ``enableHighAccuracy: true   ``},   ``trackUserLocation:  true});// Add the control to the map.map.addControl(geolocate);// Set an event listener that fires// when a trackuserlocationend event occurs.geolocate.on( 'trackuserlocationend' ,  function () {   ``console.log( 'A trackuserlocationend event has occurred.' )});
```

trackuserlocationstart

实例:

```
// Initialize the geolocate control.var geolocate =  new mmrgl.GeolocateControl({   ``positionOptions: {       ``enableHighAccuracy: true   ``},   ``trackUserLocation:  true});// Add the control to the map.map.addControl(geolocate);// Set an event listener that fires// when a trackuserlocationstart event occurs.geolocate.on( 'trackuserlocationstart' , function () {   ``console.log( 'A trackuserlocationstart event has occurred.' )});
```

ScaleControl

ScaleControl控件显示地图上的距离与实际地面距离的对应关系。

src/ui/control/scale_control.js

参数

名称	类型	描述
maxWidth	numberdefault: ‘100’	控件的最大长度，单位为像素。
unit	stringdefault: ‘metric’	距离的度量单位（‘imperial’、‘metric’或‘nautical’）。

实例:

```
var scale = new mmrgl.ScaleControl({ ``maxWidth: 80, ``unit: 'imperial'});map.addControl(scale);scale.setUnit( 'metric' );
```

方法

名称	描述
setUnit(unit)	设置刻度上的距离单位。参数：unit:Unit距离的度量单位（“imperial”、“metric”或“nautical”）。

FullscreenControl

FullscreenControl控件包含一个按钮，用于将地图切换到全屏模式和退出全屏模式。

src/ui/control/fullscreen_control.js

参数

名称	类型	描述
container	HTMLElement	container—container是一个DOM元素，在全屏模式下打开。默认情况下，将使用地图容器作为全屏模式。

实例:

```
map.addControl( new mmrgl.FullscreenControl({container: ``document.querySelector( 'body' )}));
```

4。地理和几何

LngLat

LngLat对象由经度和纬度（以度为单位）组成。这些坐标基于WGS84标准（EPSG:4326）。MMR GL使用经度和纬度的坐标顺序（与纬度和经度不同），符合GeoJSON规范。请注意，接受LngLat对象作为参数或选项的任何MMR GL方法也可以接受两个数字的数组，并进行隐式转换。这种灵活的类型被记录为LngLatLike。

src/geo/lng_lat.js

参数

lng:number—经度，以度为单位

lat:number—纬度，以度为单位

实例:

```
var ll =  new mmrgl.LngLat(-123.9749, 40.7736);ll.lng;  // = -123.9749
```

static方法

名称	描述
convert(input)	将包含lng和lat或lon和lat属性的两个数字的数组或对象转换为LngLat对象。

如果传递了LngLat对象，函数将返回它而不进行更改。

参数:

input:LngLatLike用于转换的两个数字数组或对象，或用于返回的LngLat对象。

实例:

```
var arr = [-73.9749, 40.7736];var ll = mmrgl.LngLat.convert(arr);ll;    // =
LngLat {lng: -73.9749, lat: 40.7736}
```

instance方法

名称	描述
distanceTo(lngLat)	返回两个坐标之间的大致距离（以米为单位），使用Haversine公式计算（根据R。W。Sinnott, «Haversine的优点», Sky and Telescope, vol. 68, no. 2, 1984, p. 159)

参数:

lngLat:LngLat 用于计算到目标的距离的坐标

实例:

```
varnewYork=  new mmrgl.LngLat(-74.0060, 40.7128);var losAngeles =  new mmrgl.LngLat(-118.2437, 34.0522);newYork.distanceTo(losAngeles);  // = 3935751.690893987, "真实距离" 使用非球面近似计算大约为 ~3966 公里
```

toArray()

实例:

```
var ll = new mmrgl.LngLat(-73.9749, 40.7736);ll.toArray(); // = [-73.9749, 40.7736]
```

toBounds(radius)

参数:

radius:number（默认值为0）从坐标扩展的距离，单位为米。

实例:

```
var ll = new mmrgl.LngLat(-73.9749, 40.7736);ll.toBounds(100).toArray(); // = [[-73.97501862141328, 40.77351016847229], [-73.97478137858673, 40.77368983152771]]
```

toString()

实例:

```
var ll = new mmrgl.LngLat(-73.9749, 40.7736);ll.toString(); // = "LngLat(-73.9749, 40.7736)"
```

wrap()

实例:

```
var ll = new mmrgl.LngLat(286.0251, 40.7736);var wrapped = ll.wrap();wrapped.lng; // = -73.9749
```

LngLatLike

LngLat可以是两个数字的数组（经度和纬度），或者是具有lng和lat或lon和lat属性的对象。

src/geo/lng_lat.js

实例:

```
var v1 = new mmrgl.LngLat(-122.420679, 37.772537);var v2 = [-122.420679, 37.772537];var v3 = {lon: -122.420679, lat: 37.772537};
```

LngLatBounds

LngLatBounds 对象表示由其西南和东北点的经度和纬度定义的边界。

如果没有提供参数，则创建一个值为null的边界。

任何接受LngLatBounds作为参数或选项的MMR方法也可以接受两个LngLatLike结构的数组，并进行隐式转换。这种灵活的类型被记录为LngLatBoundsLike。

src/geo/lng_lat_bounds.js

参数

sw:number—经度，以度为单位

ne:number—纬度，以度为单位

static方法

名称	描述
convert(input)	将数组转换为LngLatBounds对象。

如果传递了LngLatBounds对象，函数将返回它而不进行更改。

内部函数调用LngLat#convert来将数组转换为LngLat值。

参数：

input:LngLatBoundsLike 两个坐标的数组用于转换，或LngLatBounds对象用于返回。

实例：

```
var arr = [[-73.9876, 40.7661], [-73.9397, 40.8002]];var llb = mmrgl.LngLatBounds.convert(arr);llb;    // = LngLatBounds {_sw: LngLat {lng: -73.9876, lat: 40.7661}, _ne: LngLat {lng: -73.9397, lat: 40.8002}}
```

instance方法

名称	描述
contains(lnglat)	检查点是否在边界矩形内。

参数：

lnglat:LngLatLike

实例：

```
var llb =  new mmrgl.LngLatBounds(new mmrgl.LngLat(-73.9876, 40.7661),new mmrgl.LngLat(-73.9397, 40.8002)); var ll =  new mmrgl.LngLat(-73.9567, 40.7789); console.log(llb.contains(ll));  // = true
```

extend(obj)

getCenter()

实例:

```
var llb = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397, 40.8002]);llb.getCenter(); // = LngLat {lng: -73.96365, lat: 40.78315}
```

getEast()

getNorth()

getNorthEast()

getNorthWest()

getSouth()

getSouthEast()

getSouthWest()

getWest()

isEmpty()

setNorthEast(ne)

setSouthWest(sw)

toArray()

实例:

```
var llb = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397, 40.8002]);llb.toArray(); // = [[-73.9876, 40.7661], [-73.9397, 40.8002]]
```

toString()

实例:

```
var llb = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397, 40.8002]);llb.toString(); // = "LngLatBounds(LngLat(-73.9876, 40.7661), LngLat(-73.9397, 40.8002))"
```

实例:

```
var sw = new mmrgl.LngLat(-73.9876, 40.7661);var ne = new mmrgl.LngLat(-73.9397, 40.8002);var llb = new mmrgl.LngLatBounds(sw, ne);
```

LngLatBoundsLike

LngLatBounds对象是LngLatLike对象的数组，按[sw, ne]顺序排列，或者是按[west, south, east, north]顺序排列的数字数组。

src/geo/lng_lat_bounds.js

实例:

```
var v1 = new mmrgl.LngLatBounds(new mmrgl.LngLat(-73.9876, 40.7661),new mmrgl.LngLat(-73.9397, 40.8002));var v2 = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397, 40.8002])var v3 = [[-73.9876, 40.7661], [-73.9397, 40.8002]];
```

Point

Point对象由x和y组成，表示地图上的点位置。

src/ui/map.js

实例:

```
var point = new mmrgl.Point(-77, 38);
```

PointLike

Point对象或代表屏幕x和y坐标的两个数字数组。

src/ui/map.js

实例:

```
var p1 = new mmrgl.Point(-77, 38); // a PointLike which is a Pointvar p2 = [-77, 38]; // a PointLike which is an array of two numbers
```

MercatorCoordinate

MercatorCoordinate对象表示三维位置。

MercatorCoordinate使用web墨卡托投影（EPSG:3857），其测量单位有所不同：

- 1个单位的大小 — 是投影世界的宽度（而不是“墨卡托米”）。
- 坐标空间的原点位于西北角，而不是中间。

实例:

MercatorCoordinateMercatorCoordinate(0, 0, 0) — 是墨卡托世界的西北角，而MercatorCoordinate(1, 1, 0) 是东南角。如果您熟悉矢量瓦片，可以将坐标空间想象为0/0/0瓦片，大小/比例为1。

MercatorCoordinate的Z维度是保角的。在墨卡托坐标空间中的立方体将显示为立方体。

src/geo/mercator_coordinate.js

参数

x:number—X位置

y:number—Y位置

z:number—Z位置

static方法

名称	描述
fromLngLat(lngLatLike, altitude)	将LngLat投影到MercatorCoordinate。

参数:

lngLatLike:LngLatLike用于项目的位置。

altitude:number（默认为0）位置的高度，以米为单位。

实例:

```
var coord = mmrgl.MercatorCoordinate.fromLngLat({ lng: 0, lat: 0}, 0);coord; // MercatorCoordinate(0.5, 0.5, 0)
```

instance方法

名称	描述
meterInMercatorCoordinateUnits()	返回在给定纬度上1米的距离对应的MercatorCoordinate单位。对于使用米作为实际测量单位的坐标，这自然为转换为MercatorCoordinate提供了比例尺。MercatorCoordinate
toAltitude()	返回高度（以米为单位）从坐标。

实例:

```
var coord = new mmrgl.MercatorCoordinate(0, 0, 0.02);coord.toAltitude(); // 6914.281956295339
```

toLngLat()

实例:

```
var coord = new mmrgl.MercatorCoordinate(0.5, 0.5, 0);var lngLat = coord.toLngLat(); // LngLat(0, 0)
```

实例:

```
var nullIsland = new mmrgl.MercatorCoordinate(0.5, 0.5, 0);
```

EdgeInsets

EdgeInset对象代表应用于视口（viewport）边缘的屏幕填充。这会偏移地图的可见中心或消失点。这对于在地图上添加浮动的用户界面元素和在用户界面元素大小变化时偏移消失点非常有用。

src/geo/edge_insets.js

参数

top:number—默认为0

bottom:number—默认为0

left:number—默认为0

right:number—默认为0

static方法

名称	描述
getCenter(width, height)	一个服务方法，用于在应用填充后计算应用的新中心或消失点。以像素为单位，左上角值为 (0.0)，+y 向下。参数：**width:number** 地图宽度（以像素为单位）。**height: number** 地图高度（以像素为单位）。
interpolate(start, target, t)	在原地插入填充。同时保留 target 中不存在的任何填充的当前值。参数：**start: (PaddingOptions | EdgeInsets)** 填充参数。**target:PaddingOptions** target 的填充参数。**t:number** 插值变量。
toJSON()	返回当前状态的 json 表示，这在您只想要一个只读的填充表示时很有用。

5。Handlers（处理程序）

BoxZoomHandler

BoxZoomHandler允许用户缩放地图以适应边界框。要设置矩形，需要同时按住 Shift 键和鼠标左键，并拖动光标。

src/ui/handler/box_zoom.js

方法

名称	描述
disable()	禁用“box zoom”

实例:

```
map.boxZoom.disable();
```

enable()

实例:

```
map.boxZoom.enable()
```

isActive()

isEnabled()

ScrollZoomHandler

ScrollZoomHandler允许用户通过滚动来缩放地图。

src/ui/handler/scroll_zoom.js

方法

名称　　描述

disable()　禁用“scroll to zoom”

实例:

```
map.scrollZoom.disable()
```

enable(options?)

around—如果过了“center”，地图将围绕地图中心进行缩放

实例:

```
map.scrollZoom.enable();
```

实例:

```
map.scrollZoom.enable({ around: 'center' })
```

isEnabled()

setWheelZoomRate(wheelZoomRate)

参数

wheelZoomRate—(默认值为 1/450) 缩放速率。

实例:

```
// Slow down zoom of mouse wheelmap.scrollZoom.setWheelZoomRate(1/600);
```

setZoomRate(zoomRate)

参数

zoomRate—(默认值为 1/100) 缩放速率。

实例:

```
// Speed up trackpad zoommap.scrollZoom.setZoomRate(1/25);
```

DragPanHandler

DragPanHandler允许用户通过点击和拖动光标来移动地图。

src/ui/handler/shim/drag_pan.js

方法

名称　　描述

disable()　禁用“drag to pan”

实例:

```
map.dragPan.disable();
```

enable(options?)

参数

linearity—用于缩放阻力速度（drag velocity）的系数

easing—平滑函数（easing function），在拖动时应用于 map.panTo。

maxSpeed—额定前阻力的最大速度值。

deceleration—触摸后减速的速率。

实例:

```
map.dragPan.enable();
```

实例:

```
map.dragPan.enable({  ``linearity: 0.3,  ``easing: bezier(0, 0, 0.3, 1),  ``maxSpeed: 1400,  ``deceleration: 2500,});
```

isActive()

isEnabled()

DragRotateHandler

DragRotateHandler允许用户通过点击和拖动光标来旋转地图，同时按住鼠标右键或 ctrl 键。

src/ui/handler/shim/drag_rotate.js

方法

名称	描述
disable()	禁用“drag to rotate”

实例:

```
map.dragRotate.disable();
```

enable()

实例:

```
map.dragRotate.enable();
```

isActive()

isEnabled()

KeyboardHandler

KeyboardHandler允许用户使用以下键盘组合来缩放、旋转和平移地图：

- = / + ：放大到 1 的级别。
- Shift-= / Shift-+：放大缩放级别 2。
- -：缩小 1 的级别。
- ShiftShift-：缩小 2 的级别。
- 箭头键 (Arrow keys)：平移 100 像素。
- Shift+⇢:增加 15 度的旋转。
- Shift+⇠:减少 15 度的旋转。
- Shift+⇡:增加 10 度的倾斜。
- Shift+⇣:减少 10 度的倾斜。

src/ui/handler/keyboard.js

方法

名称　　描述

disable()　禁用“keyboard rotate and zoom”

实例:

```
map.keyboard.disable();
```

disableRotation()

实例:

```
map.keyboard.disableRotation();
```

enable()

实例:

```
map.keyboard.enable();
```

enableRotation()

实例:

```
map.keyboard.enableRotation();
```

isActive()

isEnabled()

DoubleClickZoomHandler

DoubleClickZoomHandler允许用户通过双击鼠标左键或双击来在特定点缩放地图。

src/ui/handler/shim/dblclick_zoom.js

方法

名称　　描述

disable()　禁用“double click to zoom”

实例:

```
map.doubleClickZoom.disable();
```

enable()

实例:

```
map.doubleClickZoom.enable();
```

isActive()

isEnabled()

TouchZoomRotateHandler

TouchZoomRotateHandler允许用户通过触摸屏幕来放大和旋转地图。

他们可以通过用一个手指双击并拖动来放大。在第二次触碰时，保持手指按住并向上或向下拖动以放大或缩小。

src/ui/handler/shim/touch_zoom_rotate.js

方法

名称	描述
disable()	禁用“pinch to rotate and zoom”。

实例:

```
map.touchZoomRotate.disable();
```

disableRotation()

实例:

```
map.touchZoomRotate.disableRotation();
```

enable(options?)

参数

around—如果过了“center”，地图将围绕地图中心进行缩放

实例:

```
map.touchZoomRotate.enable();
```

实例:

```
map.touchZoomRotate.enable({ around: 'center' });
```

enableRotation()

实例:

```
map.touchZoomRotate.enable();map.touchZoomRotate.enableRotation();
```

isActive()
isEnabled()

TouchPitchHandler

TouchPitchHandler允许用户通过用两个手指上下拖动来倾斜地图。

src/ui/handler/touch_zoom_rotate.js

方法

名称　　描述
disable()　禁用“drag to pitch”

实例:

```
map.touchPitch.disable();
```

enable()

实例:

```
map.touchPitch.enable();
```

isActive()
isEnabled()

6。Sources（数据源）

GeoJSONSource

包含 GeoJSON 的源（source）。

src/source/geojson_source.js

实例:

```
map.addSource('some id', {  ``type: 'geojson',  ``data:
'https://d2ad6b4ur7yvpq.cloudfront.net/naturalearth-3.3.0/ne_10m_ports.geojson'}
);
```

实例:

```
map.addSource('some id', {    ``type: 'geojson',    ``data: {        ``"type"
: "FeatureCollection",        ``"features": [{            ``"type": "Feature"
,            ``"properties": {},            ``"geometry":
{                ``"type": "Point",                ``"coordinates":
[                    ``-76.53063297271729,                    ``39.1817407799410
8                ``]            ``}        ``}]    ``}});
```

实例:

```
map.getSource( 'some id' ).setData({    ``"type" :  "FeatureCollection"
,    ``"features" : [{        ``"type" :  "Feature" ,        ``"properties" : {
"name" :  "Null Island" },        ``"geometry" : {            ``"type" :  "Point"
,            ``"coordinates" : [ 0, 0 ]        ``}    ``}]});
```

方法

名称	描述
getClusterChildren(clusterId, callback)	对于聚类源（sources），提取给定聚类在下一个缩放级别上的子元素（以 GeoJSON 对象数组的形式）。参数 **clusterId:number** — 聚类 ID**callback:function<Array>** — 当提取对象时将被调用的函数 (callback) ((error features) => { … }).
getClusterExpansionZoom(clusterId, callback)	对于聚类源（source），选择展开此聚类的缩放级别。参数 **clusterId:number** — 聚类 ID**callback:function**—当获取缩放级别 (callback) 时将被调用的函数 (zoom) ((error features) => { … })
getClusterLeaves(clusterId, limit, offset, callback)	对于聚类源（source），提取属于该聚类的原始点（以 GeoJSON 对象数组的形式）。

参数

clusterId:number—聚类ID

limit:number—最大特征数 (features)

offset:number—需要跳过的特征数 (features) (例如，用于分页 - pagination)。

callback:function<Array> — 提取对象时将调用的函数 (callback) ((error features) => { … })。

实例:

```
// Retrieve cluster leaves on clickmap.on( 'click' ,  'clusters' ,  function (e)
{  ``var features = map.queryRenderedFeatures(e.point, {      ``layers: [
'clusters' ]  ``});   ``var clusterId = features[0].properties.cluster_id;  ``var
pointCount = features[0].properties.point_count;  ``var clusterSource =
map.getSource( 'clusters' );   ``clusterSource.getClusterLeaves(clusterId,
pointCount, 0,  function (error, features) {    ``// Print cluster leaves in the
console    ``console.log( 'Cluster leaves:' , error, features);  ``})});
```

setData(data)

VideoSource

数据源，包含视频。

src/source/video_source.js

实例:

```
// add to mapmap.addSource( 'some id' , {    ``type: 'video' ,    ``url:
[        ``'path-to-video.mp4'
,        ``'path-to-video.webm'    ``],    ``coordinates: [        ``[-76.54,
39.18],        ``[-76.52, 39.18],        ``[-76.52, 39.17],        ``[-76.54,
39.17]    ``]}); // updatevar mySource = map.getSource( 'some id'
);mySource.setCoordinates([    ``[-76.54335737228394,
39.18579907229748],    ``[-76.52803659439087,
39.1838364847587],    ``[-76.5295386314392,
39.17683392507606],    ``[-76.54520273208618,
39.17876344106642]]); map.removeSource( 'some id' );  // remove
```

方法

名称	描述
getVideo()	返回 HTML 视频
pause()	在视频上设置暂停
play()	继续播放视频
setCoordinates()	设置视频坐标并重新渲染(re-render)地图

ImageSource

图片数据源。

src/source/image_source.js

实例:

```
// add to mapmap.addSource( 'some id' , {    ``type: 'image' ,    ``url:
'foo.png' ,    ``coordinates: [        ``[-76.54, 39.18],        ``[-76.52,
39.18],        ``[-76.52, 39.17],        ``[-76.54, 39.17]    ``]}); // update
coordinatesvar mySource = map.getSource( 'some id'
);mySource.setCoordinates([    ``[-76.54335737228394,
39.18579907229748],    ``[-76.52803659439087,
39.1838364847587],    ``[-76.5295386314392,
39.17683392507606],    ``[-76.54520273208618, 39.17876344106642]]); // update
url and coordinates simultaneouslymySource.updateImage({    ``url: 'bar.png'
,    ``coordinates: [        ``[-76.54335737228394,
39.18579907229748],        ``[-76.52803659439087,
39.1838364847587],        ``[-76.5295386314392,
39.17683392507606],        ``[-76.54520273208618,
39.17876344106642]    ``]}) map.removeSource( 'some id' );  // remove
```

方法

名称	描述
setCoordinates(coordinates)	设置图片坐标并重新渲染（re-render）地图。
updateImage(options)	更新 URL — 图片地址以及必要时的坐标。为避免图片闪烁，请将 raster-fade-duration 属性设置为 0。参数 **url:string** — 图片的 URL 地址 **coordinates:Array<Array>** — 四个坐标，以经度和纬度数字数组形式表示，定义图片的角落。坐标从图片的左上角开始，顺时针方向。

CanvasSource

数据源，包含 HTML 画布（HTML canvas）内容。

src/source/canvas_source.js

实例：

```
// add to mapmap.addSource( 'some id' , {    ``type: 'canvas' ,    ``canvas: 'idOfMyHTMLCanvas' ,    ``animate: true ,    ``coordinates: [       ``[-76.54, 39.18],       ``[-76.52, 39.18],       ``[-76.52, 39.17],       ``[-76.54, 39.17]    ``]}); // updatevar mySource = map.getSource( 'some id' );mySource.setCoordinates([    ``[-76.54335737228394, 39.18579907229748],    ``[-76.52803659439087, 39.1838364847587],    ``[-76.5295386314392, 39.17683392507606],    ``[-76.54520273208618, 39.17876344106642]]); map.removeSource( 'some id' );  // remove
```

方法

名称	描述
getCanvas()	返回 HTML 画布
pause()	停用动画。地图将显示画布图像的静态副本。
play()	启用动画。地图将会交互。
setCoordinates(coordinates)	设置画布的坐标并重新渲染（re-render）地图。参数：**coordinates:Array<Array>** — 四个坐标，以经度和纬度的数字数组形式表示，定义画布的角落。坐标从画布的左上角开始，顺时针方向。

CanvasSourceOptions

添加画布源类型到地图的参数。

src/source/canvas_source.js

参数

type:string—源类型必须等于 “canvas”。

canvas:(string | HTMLCanvasElement) — 画布源，可以从中读取像素。这可以是表示 canvas 元素标识符的字符串，或者是 HTMLCanvasElement 本身。

coordinates:Array<Array> — 四个坐标，表示放置画布角的位置，以［经度，纬度］对的形式设置。

animate:boolean—画布源是否为动画。如果画布是静态的（即不需要在每一帧重新读取像素），则应将 animate 设置为 false 以提高性能。

7。Events（事件）

Evented

添加和移除监听器（listeners）的方法

src/util/evented.js

方法

名称	描述
off(type, listener)	移除之前注册的监听器（listener）。参数 **type:string** — 事件类型 **listener:function** — 监听器（listener）函数
on(type, listener)	注册新的监听器（listener）。参数 **type:string** — 事件类型 **listener:function** — 监听器（listener）函数
once(type, listener)	添加将仅对指定事件类型调用一次的监听器（listener）。参数 **type:string** — 事件类型 **listener:function** — 监听器（listener）函数

MapMouseEvent

鼠标相关事件

src/ui/events.js

实例：

```
// The `click` event is an example of a `MapMouseEvent`.// Set up an event listener on the map.map.on( 'click' , function (e) {  ``// The event object (e)
```

```
contains information like the  ``// coordinates of the point on the map that was
clicked.  ``console.log( 'A click event has occurred at ' + e.lngLat);});
```

属性

名称	描述
lngLat	地图上光标的坐标
originalEvent	触发地图事件的 DOM 事件
point	鼠标光标相对于地图的像素坐标，从左上角开始测量
preventDefault()	阻止事件的后续处理。调用此方法将阻止以下事件：- mousedown，DragPanHandler 的行为mousedown mousedown，DragRotateHandler 的行为- mousedown，BoxZoomHandler 的行为- dblclick，DoubleClickZoomHandler 的行为 DoubleClickZoomHandler
target	触发事件的地图对象
type	事件类型（一个的以下 Map.event:mousedown, Map.event:mouseup, Map.event:click, Map.event:dblclick, Map.event:mousemove, Map.event:mouseover, Map.event:mouseenter, Map.event:mouseleave, Map.event:mouseout, Map.event:contextmenu）。

MapTouchEvent

触摸事件（touch）

src/ui/events.js

属性

名称	描述
lngLat	地图上触摸的坐标
lngLats	对应于地图上触摸的坐标数组
originalEvent	触发地图事件的 DOM 事件
point	鼠标光标相对于地图的像素坐标，从左上角开始测量
points	对应于触摸的像素坐标数组

名称	描述
preventDefault()	阻止地图后续处理事件。调用此方法将阻止以下事件：- touchstart，DragPanHandler 的行为 - touchstart，TouchZoomRotateHandler 的行为 TouchZoomRotateHandler
target	触发事件的地图对象
type	事件类型

MapBoxZoomEvent

与 BoxZoomHandler 使用的 ‘boxzoom’ 事件相关BoxZoomHandler

src/ui/events.js

参数

originalEvent:MouseEvent—DOM触发 ‘boxzoom’ 事件的 DOM 事件。可以是 MouseEvent 或 KeyboardEvent。

type:string—‘boxzoom’ 事件的类型。可以是：boxzoomstart boxzoomend 或 boxzoomcancel。

target:Map—触发事件的地图实例。

MapDataEvent

MapDataEvent 对象与 Map.event:data 和 Map.event:dataloading 事件一起创建。数据类型的可能值包括：

- “source”：与任何源关联但不与瓦片关联的数据。
- “style”：地图的样式。

参数

type:string—事件类型

dataType:string—改变的数据类型。可以是：“source”、“style”。

isSourceLoaded:boolean—如果事件的数据类型为 “source” 并且源（source）没有未解决的网络请求（outstanding network requests），则为 true。

source:Object—source 类型的对象。

sourceDataType:string—激活此字段如果事件的数据类型为 ‘source’ 并且事件报告内部数据已被获取或更改。可能的值包括 ‘metadata’、‘content’ 和 ‘visibility’。

tile:Object—如果事件的数据类型为 ‘source’’ 并与瓦片的加载有关，则为加载或更改的瓦片。

coord:Coordinate—如果事件的数据类型为 ‘source’ 并且事件与瓦片的加载有关，则为瓦片的坐标。

src/ui/events.js

实例:

```
// The sourcedata event is an example of MapDataEvent.// Set up an event
listener on the map.map.on( 'sourcedata' , function (e) {  ``if
(e.isSourceLoaded) {    ``// Do something when the source has finished
loading  ``}});
```

MapWheelEvent

与 ‘wheel’ 相关的事件

src/ui/events.js

属性

名称	描述
orignalEvent	触发 map 事件的 DOM 事件。
preventDefault()	阻止事件的后续处理。调用此方法将阻止 ScrollZoomHandler 的行为。
target	触发事件的地图对象。
type	事件类型

8。在 React 应用中的使用

React —用于创建用户界面的 JavaScript 库。由于 React 操作 DOM，将 React 与其他同样操作 DOM 并管理状态的库（如 MMR GL JS）结合使用可能会有些困难。在本节中，您将学习如何创建一个使用 MMR GL JS 来可视化地图、显示地图中心点的坐标和其缩放级别，并随着用户与地图的互动更新显示的 React 应用。您将能够使用本指南中描述的原理来创建更复杂的应用，这些应用同时使用 React 和 MMR GL JS。

配置 React 应用的结构

创建一个以项目名称命名的新文件夹。

在项目文件夹中创建一个新文件：

- **package.json**—此文件用于指定应用所需的所有包。

在项目文件夹中创建一个名为 public 的文件夹。在这个文件夹中创建两个新文件：

- **index.html**—此 HTML 文件将显示可视化地图，用户可以与之互动。
- **site.css**—此文件将包含 CSS，用于正确格式化地图和侧边栏。创建 HTML 框架

在项目文件夹中创建另一个名为 src 的文件夹。在 src 文件夹中创建一个新文件：在 src 文件夹中创建一个新文件：

- **index.js**—此 JavaScript 文件将包含所有 React 逻辑，用于设置和管理应用程序的状态以及显示地图。

创建这些文件夹和文件后，您将拥有以下的文件结构：

```
project└── package.json└── public|      └── index.html|      └── site.css└── src      ``└── index.js
```

接下来执行 npm install。

创建 React 应用

打开 public/index.html 文件并插入以下代码：

index.html

```
<!DOCTYPE html>< html lang = "en" >< head >      ``< title >MMR GL JS and React</ title >      ``< meta charset = "utf-8" />      ``< meta name = "viewport" content = "width=device-width, initial-scale=1" />      ``< link href="https://geo.rustore.ru/sdk/js/<version>/mmr-gl.css" rel="stylesheet" />      ``< link href = "%PUBLIC_URL%/site.css" rel = "stylesheet" />      ``< script src="https://geo.rustore.ru/sdk/js/<version>/mmr-gl.js"></ script ></ head >< body >      ``< div id = "app" ></ div ></ body ></ html >
```

这段代码创建了用户将看到的 HTML 页面结构。在

页面有一个标签

带有应用程序标识符这个

是 React 应用将在页面上显示的容器。

这段代码还引用了两个不同样式文件

。第一个样式— MMR GL JS 样式，它确保您的地图上的元素正确显示。第二个样式—之前创建的 site.css 文件，您将在其中添加特定应用的 CSS。

设置默认状态（default state）

打开 src/index.js 文件。在文件的顶部添加以下导入：

src/index.js

使用钩子（hooks）的版本

```
import React, { useRef, useEffect, useState } from 'react';import ReactDOM from 'react-dom';
```

使用类（classes）的版本

```
import React from 'react';import ReactDOM from 'react-dom';
```

导入 MMR GL 并添加您的访问令牌。设置访问令牌（accessToken）：

```
mmrgl.accessToken = 'token';
```

您可以配置 React 应用！为了创建一个您将在接下来的几步中添加代码的结构，请在 index.js 中添加以下内容：

使用钩子（hooks）的版本

```
const Map = () => {};  ReactDOM.render(<Map />, document.getElementById( 'app' ));
```

使用类（classes）的版本

```
class Map extends React.PureComponent {}  ReactDOM.render(<Map />,
document.getElementById( 'app' ));
```

这定义了地图，然后指明它应该显示在

带有应用程序标识符

绘制地图

绘制地图 您需要在应用中可视化地图。在 React 应用中初始化地图的入口点通过 return 中指定的一个元素。在您的应用中添加以下代码：

src/index.js

使用钩子（hooks）的版本

```
return (    ``<div>        ``<div className="map-container" ref={mapContainer}
/>    ``</div>);
```

使用类（classes）的版本

```
render() {    ``const { lng, lat, zoom } = this.state;    ``return
(        ``<div>            ``<div ref={this.mapContainer} className=
"map-container" />        ``</div>    ``);}
```

mapContainerref指出地图应该在 HTML 页面的新元素中绘制。

。

地图需要几个样式规则才能正确显示。在 site.css 文件中添加以下代码：

```
.map-container {    ``position: absolute;    ``top: 0;    ``right: 0
;    ``left: 0;    ``bottom: 0;}
```

保存更改。在命令行中执行 npm start 命令。这将启动本地服务器并在浏览器中打开一个新页面，显示您的应用。

如果您打开浏览器的控制台，您可能会看到错误：'map' is assigned a value but never used。不用担心 — 在下一步中您将使用 map 变量！

存储坐标

然后，您需要创建一个函数，当用户与地图交互时，它将保存新的纬度、经度和缩放级别。您将编写一个MMR GL JS map.on('move')函数，当用户移动地图时，它将设置状态值。在useEffect中，如果您使用钩子，或者在componentDidMount()中，如果您使用类，请添加以下代码：

使用钩子（hooks）的版本

```
map.on( 'move' , () =>
{    ``setLng(map.getCenter().lng.toFixed(4));    ``setLat(map.getCenter().lat.t
oFixed(4));    ``setZoom(map.getZoom().toFixed(2));});
```

使用类（classes）的版本

```
map.on( 'move' , () => {    ``this .setState({        ``lng:
map.getCenter().lng.toFixed(4),        ``lat:
map.getCenter().lat.toFixed(4),        ``zoom:
map.getZoom().toFixed(2)    ``});});
```

在这个函数中，使用useState()，如果您使用钩子（hooks），或者setState()，如果您使用类（classes），在地图移动时重置lng, lat和zoom值。该函数使用以下方法：

- getCenter()，一个MMR MMR GL方法，用来获取地图中心点的新经度和纬度。
- getZoom()，一个MMR GL JS方法，用来确定地图设置的缩放级别。
- toFixed()，一个JavaScript方法，允许将浮点数结果截断为指定位数的数字。

显示坐标

当您能够收集和存储这些信息时，您可以使用 return 来在地图上显示它。在开放标签内部

为存储地图创建的，添加一个新标签

用于显示地图的经度、纬度和缩放级别。现在 return 操作符看起来像这样：

使用钩子（hooks）的版本

```
return (    ``<div>        ``<div className= "sidebar" >            ``Longitude:
{lng} | Latitude: {lat} | Zoom: {zoom}        ``</div>        ``<div className=
"map-container" ref={mapContainer} />    ``</div>);
```

使用类（classes）的版本

```
render() {    ``const { lng, lat, zoom } = this .state;    ``return
(        ``<div>            ``<div className= "sidebar"
>                ``Longitude: {lng} | Latitude: {lat} | Zoom:
```

```
{zoom}            ``</div>            ``<div ref={this.mapContainer} className=
"map-container" />        ``</div>    ``);}
```

侧边栏需要一些样式才能在页面上正确显示。在 site.css 文件中添加以下 CSS：

site.css

```
.sidebar {    ``background-color: rgba(35, 55, 75, 0.9);    ``color:
#ffffff;    ``padding: 6px 12px;    ``font: 15px/24px monospace;    ``z-index
: 1;    ``position: absolute;    ``top: 0;    ``left: 0;    ``margin: 12px
;    ``border-radius: 4px;}
```

保存您的工作并返回到浏览器页面。地图的左上角有一个侧边栏，按照 site.css 中设置的 CSS 规则进行样式设计。侧边栏显示了地图中心当前的纬度和经度以及缩放级别。当您缩放和移动地图时，侧边栏的内容将更新。

描述地图上的额外对象

AttributionControl

AttributionControl 控件显示地图属性信息（版权等）。

src/ui/control/attribution_control.js

参数：

compact:boolean—如果为 true，则始终以紧凑模式显示；如果为 false，则始终以全尺寸显示；默认情况下，这取决于地图大小（viewport < 640? compact: full）

customAttribution:string|Array — 在地图属性中显示的字符串或字符串的数组

实例：

```
var map = new mmrgl.Map({attributionControl: false})  ``.addControl(new
mmrgl.AttributionControl({      ``compact: true    ``}));
```

LngLatBoundsLike

LngLatBounds 对象，LngLatLike 对象数组按 [sw ne] 顺序或按 [west south east north] 顺序的数字数组。

src/geo/lng_lat_bound.js

实例：

```
var v1 = new mmrgl.LngLatBounds(    ``new mmrgl.LngLat(-73.9876,
40.7661),    ``new mmrgl.LngLat(-73.9397, 40.8002));var v2 = new
mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397, 40.8002])var v3 = [[-73.9876,
40.7661], [-73.9397, 40.8002]];
```

静态地图

`/staticmap/png` — 这是一个允许获取地图图像的服务。

请求

请求的必要参数

字段名	格式	描述	示例
api_key	hex-string	访问服务	— `fa749bace6d8a3b1...`
` latlon `	float,float	地图所需图像的几何中心的纬度和经度（以度为单位）。纬度和经度通过逗号指定。如果请求中指定了 bbox，则忽略 latlon 中指定的值	— `latlon=55.727,37.59`
bbox	float,float,float,float	描述对象位置的区域，由一对坐标组成，顺序为：lat1,lon1,lat2,lon2。坐标的纬度和经度通过逗号指定。坐标之间用逗号分隔	— `bbox=55.7,37.65,55.8,37.66`

请求的附加参数

字段名	格式	描述	示例
zoom	integer	缩放级别范围从 0 到 17，其中：- **0**（默认）— 对应于世界概览级别；- **17** — 对应于建筑物概览级别。	—

字段名	格式	描述	示例
zoom=13			
width	integer	地图图像的宽度。值可以在 32 到 1024 像素之间（默认 512）	—
width=640			
height	integer	地图图像的高度。值可以在 32 到 1024 像素之间（默认 512）	—
height=480			
pins	string	定义地图上添加图钉的位置和类型的参数。参数应以以下格式传递：lat1,lon1,icon1|lat2,lon2,icon2|…，其中：**lat1…latN** —图钉的纬度，以度为单位；**lon1…lonN** —图钉的经度，以度为单位；**icon1…iconN** —图钉的格式。可用图钉的格式在图钉集合中给出	—
pins=55.7505,37.6165,blue_star|60.6543,38.1255,red_camera			
style	string	选择地图显示样式的参数。默认使用样式：main	—
style=light			
padding	integer	地图属性的偏移参数。偏移以像素为单	

字段名	格式	描述	示例
		位，等于标签右边缘到属性右边缘的距离。限制：值可在5到width/2像素的范围内（默认为5）	
— padding=40			
scale	integer	图像缩放参数。可能的值为1或2（默认为1）。如果scale=2，则最终图像大小将是由于更高的像素密度而增加两倍。例如，请求 width=200&height=200&scale=2时，结果图像将是400x400像素，但看起来像请求 width=200&height=200，而不是 width=400&height=400：  width=200&height=200	

字段名	格式	描述	示例
			 width=200&height=200&scale=2

字段名	格式	描述	示例
			 width=400&height=400
—			
scale= 2			

请求(POST)

请求的必要参数(GET)

字段名	格式	描述	示例
api_key	hex-string	访问服务	—
fa749bace6d8a3b1....			

请求的必要参数 (POST)

参数在请求体中以JSON对象的形式接收

字段名	格式	描述	示例
 coord 	**coord**—JSON包含float字段“lat”和“lon”的JSON对象	地图所需图像的几何中心的纬度和经度（以度为单位）。如果请求中指定了“bbox”，则忽略“coord”中的值	
—			
{ ``"lat": 55.727, ``"lon": 37.59}			
bbox	JSON两个元素的数组	由一对坐标描述的对象所在区域。	
—			
"bbox": [``{ ``"lat": 55.71, ``"lon": 37.65 ``}, ``{ ``"lat": 55.8, ``"lon": 37.6535 ``}]			

请求的附加参数

字段名	格式	描述	示例
字段名	格式	描述	示例
—	—	—	—
zoom	integer	缩放级别范围从0到17，其中- **0**（默认）— 对应于世界级别的视图；- **17** — 对应于建筑级别的视图	
—			
"zoom": 13			
width	integer	地图图像的宽度。值可以在 32 到 1024 像素之间（默认 512）	
—			

字段名	格式	描述	示例
"width": 640			
height	integer	地图图像的高度。值可以在 32 到 1024 像素之间（默认 512）	
—			
"height": 480			
pins	JSON对象数组	添加到地图上的图钉的描述	
—			
"pins": [{ ``"coord": { ``"lat": 55.73, ``"lon": 37.59 ``}, ``"icon": { ``"symbol": "rustore-corp_photo" ``}},{ ``"coord": { ``"lat": 55.76, ``"lon": 37.59 ``}, ``"icon": { ``"base64": "iVBORw0KGgoA"... ``}},{ ``"coord": { ``"lat": 55.745, ``"lon": 37.67 ``}, ``"icon": { ``"url": "https://geo.rustore.ru/welcome/static-map-logo-50x50.png" ``}}]			
coord	JSON数组	定义添加到地图上的图钉的位置	
icon	JSON数组	描述添加到地图上的图钉的视觉表现	

字段名	格式	描述	示例
symbol	string	“symbol”字段包含图钉的视觉表现格式。可用图钉的格式在图钉集合中给出	
base64	string	图钉的PNG格式图片，以base64编码形式（不超过250千字符）	
url	string	URLPNG图片。URL长度不得超过1000字符，图片大小不得超过512KB。仅支持HTTPS传输，无压缩和重定向。	
features	GeoJSON对象	描述在地图顶层显示的几何图形。properties属性是可选的。	

—

```
"features" :
{  ``"type" :
"FeatureCollection
" ,  ``"features" :
[  ``{    ``"type"
:  "Feature"
,    ``"geometry" :
{      ``"type" :
"Point"
,      ``"coordina
tes" : [ 37.6165 ,
55.7505
]    ``},    ``"pr
operties" :
{      ``"title" :
"points"    ``}  `
`},  ``{    ``"typ
e" :  "Feature"
,    ``"geometry" :
{      ``"type" :
"LineString"
,      ``"coordina
tes" : [        ``[
37.6163 ,  55.7503
```

字段名	格式	描述	示例
], [37.6164 , 55.7504], [37.6165 , 55.7505], [37.6166 , 55.7506] ``] ``}, ``"properties" : { ``"title" : "line" ``} ``}, ``{ ``"type" : "Feature" , ``"geometry" : { ``"type" : "Polygon" , ``"coordinates" : [ ``[[37.6395 , 55.73], [37.6066 , 55.73], [37.6066 , 55.751], [37.6395 , 55.751], [37.6395 , 55.73]], ``[[ 37.6345 , 55.735 ], [ 37.6345 , 55.746 ], [ 37.6116 , 55.746 ], [ 37.6116 , 55.735 ], [ 37.6345 , 55.735 ]] ``] ``}, ``"properties" : { ``"title" : "polygon" ``} ``}]}			
features-style	JSON对象	配置传递到features中的GeoJSON的视觉显示。分别配置点、线和多边形的显示：- point — 点的设置。点以圆形显示。- circle-color — 圆的颜色 - circle-opacity —	

字段名	格式	描述	示例
		圆的透明度 - circle-radius — 圆的半径（像素） - line — 线的设置 - line-width — 线宽（像素） - line-color — 线的颜色 - line-opacity — 线的透明度 - polygon — 多边形的设置 - fill-color — 填充颜色 - fill-opacity — 填充透明度	
—			"features-style": { ``"point": {  ``"circle-color": "#ffffff", ``"circle-opacity": 0.8, ``"circle-radius": 4.0 ``}, ``"line": { ``"line-color": "#2688eb", ``"line-width": 2.0 ``}, ``"polygon": { ``"fill-color": "#2688eb", ``"fill-opacity": 0.5 ``} ``}
style	string	选择地图显示样式的参数。默认使用“main”样式	
—			"style": "light"

字段名	格式	描述	示例
padding	integer	地图属性的偏移参数。偏移以像素为单位，等于标签右边缘到属性右边缘的距离。限制：值可在5到width/2像素的范围内（默认为5）	—

```
"padding": 40
```

scale	integer	图像缩放参数。可能的值为1或2（默认为1）。	—

```
"scale": 2
```

响应

作为响应，您将收到一张与请求中指定的参数相符的地图部分的图片。

示例

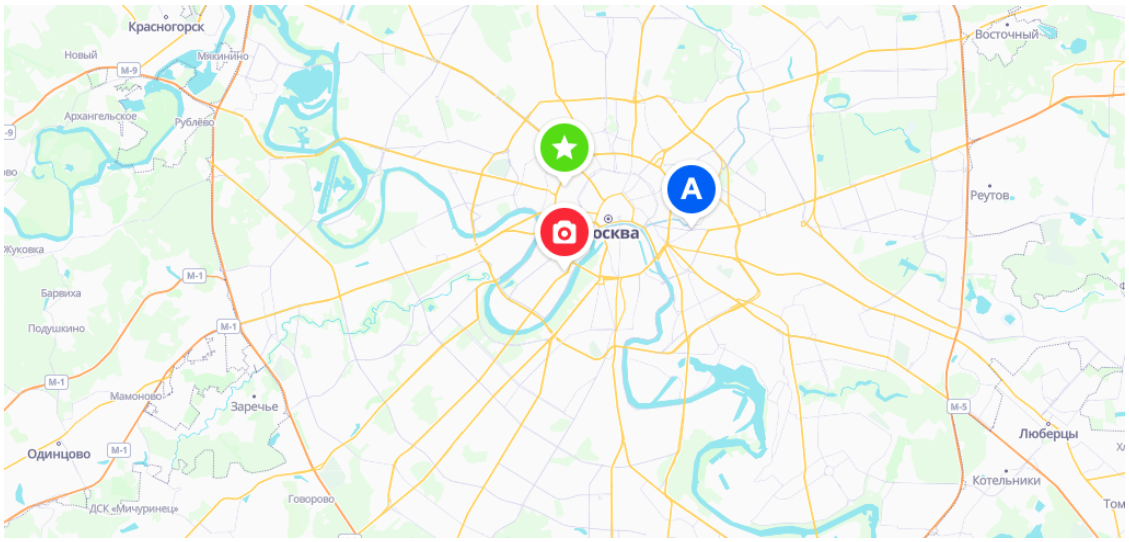
请求(GET)

```
https://geo.rustore.ru/api/staticmap/png?api_key=<YOUR_API_KEY>&latlon=55.727,37.59&style=main&zoom=10&width=1024&height=512&padding=5&pins=55.73,37.59,rustore-corp_photo|55.76,37.59,green_star|55.745,37.67,rustore-electric_a
```

请求(POST)

```
{ ``"width": 660, ``"height": 600, ``"bbox": [{ ``"lat": 55.71, ``"lon": 37.65 ``}, { ``"lat": 55.8, ``"lon": 37.6535 ``} ``], ``"padding": 200, ``"scale": 1, ``"features": {}, ``"coord": { ``"lat": 55.7505, ``"lon": 37.6165 ``}, ``"pins": [ ``{ ``"coord": { ``"lat": 55.73, ``"lon": 37.59 ``}, ``"icon": { ``"symbol": "rustore-corp_photo" ``} ``}, ``{ ``"coord": { ``"lat": 55.76, ``"lon": 37.59 ``}, ``"icon": { ``"symbol": "green_star" ``} ``}, ``{ ``"coord": { ``"lat": 55.745, ``"lon": 37.67 ``}, ``"icon": { ``"symbol": "rustore-electric_a" ``} ``} ``], ``"zoom": 11, ``"style": "main"}
```

响应



图钉集合

图标的名称（icon请求参数）由“颜色”+“_”+“类型”组成。icon=color_image

例如，如果您需要一个红色带相机图像的图钉，则icon=red_cameraicon=red_camera

在使用静态地图图像创建服务时，您只能使用预先定义的图钉。

如果您需要为您的项目添加额外的图钉，请联系RuStore支持

图钉类型

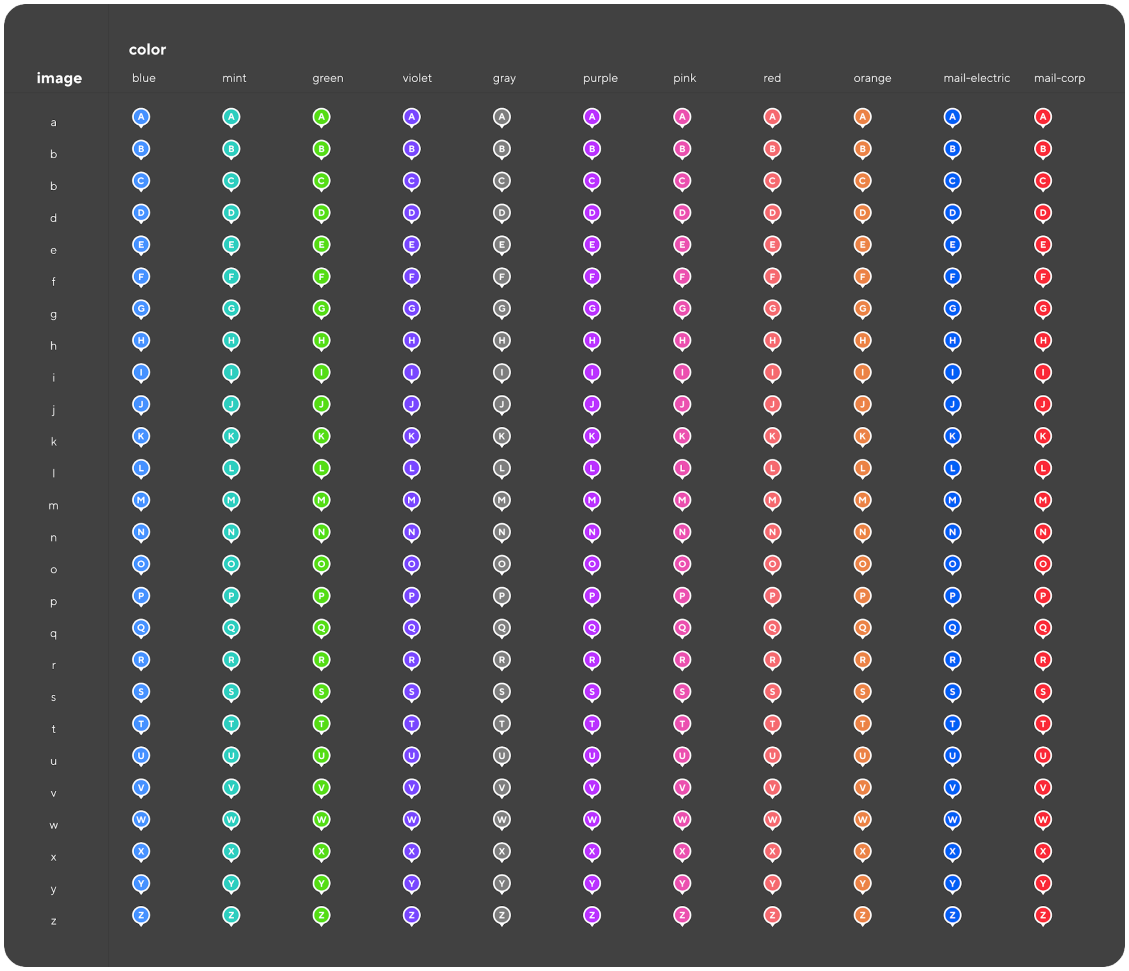
图标



数字

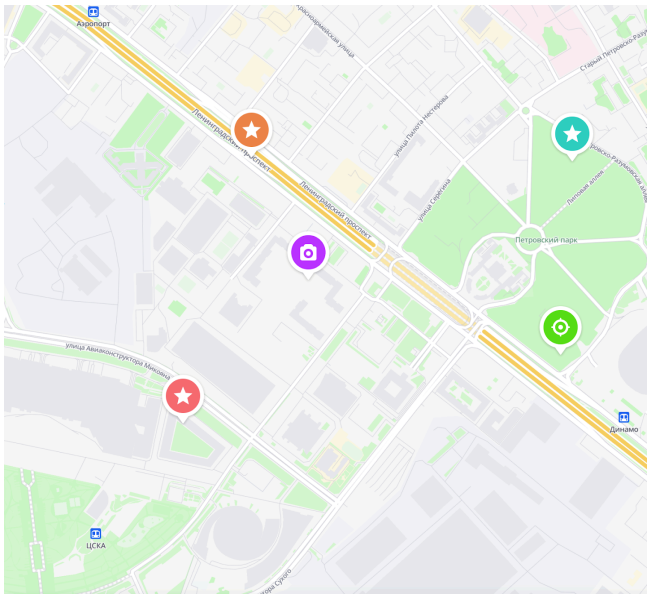

color
image
blue
mint
green
violet
gray
purple
pink
red
orange
mail-electric
mail-corp
1
2
3
4
5
6
7
8
9
0

字母

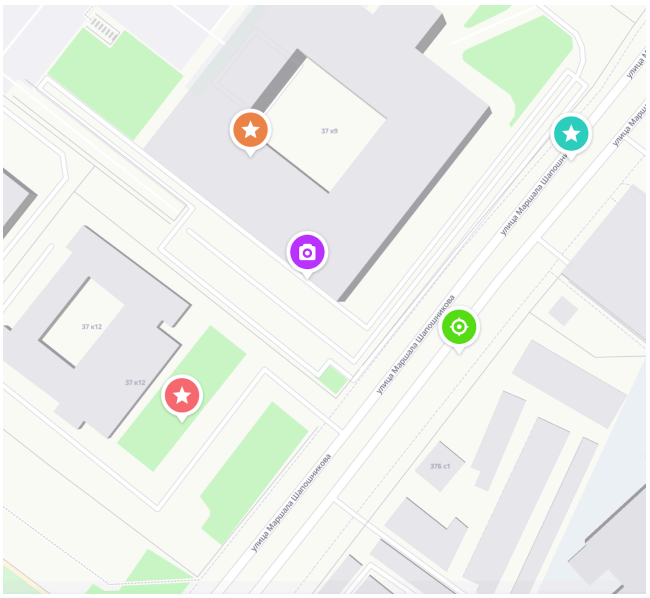

color
image
blue
mint
green
violet
gray
purple
pink
red
orange
mail-electric
mail-corp
a
b
b
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z

显示示例

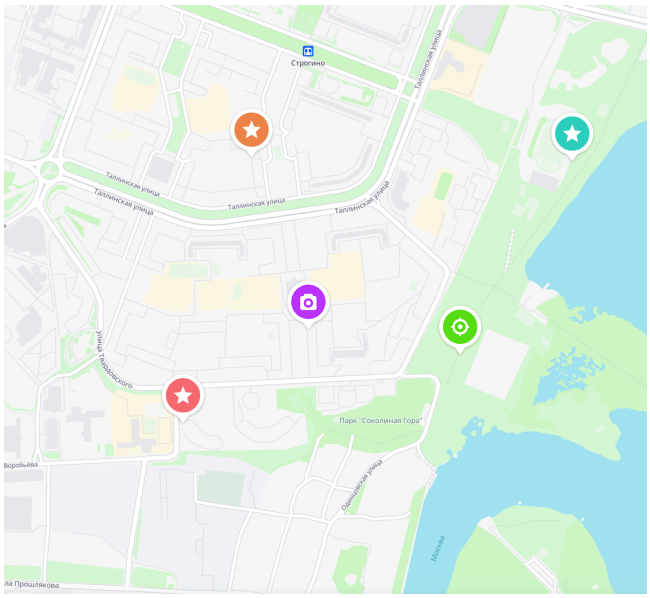
标准视图



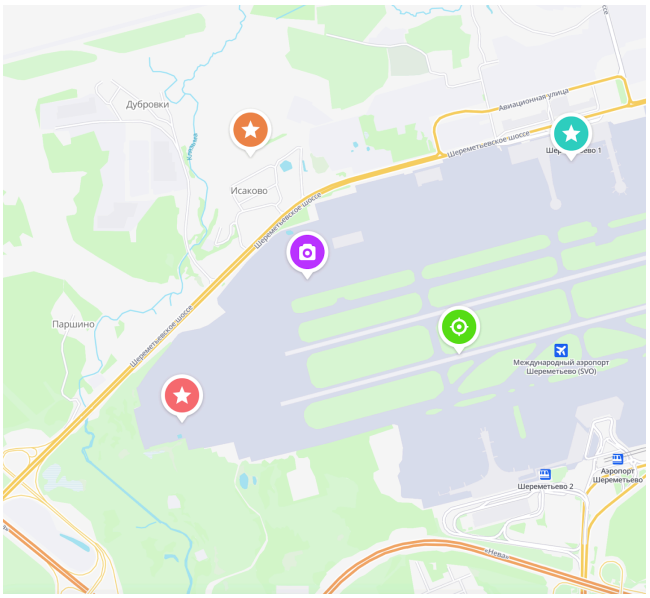
放大视图



水

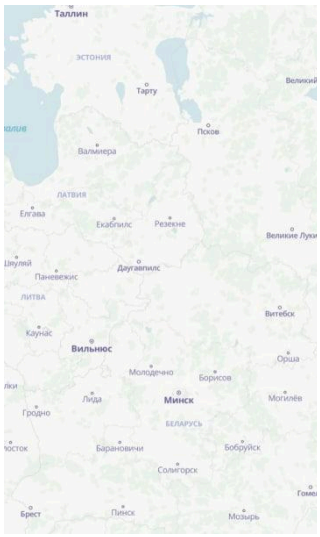
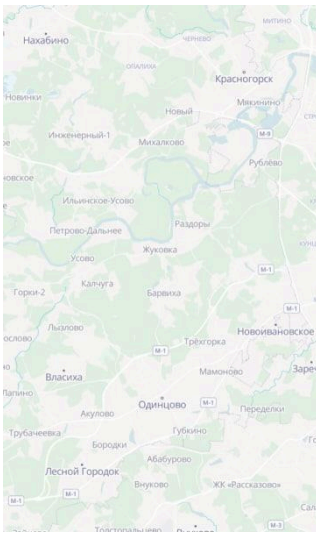
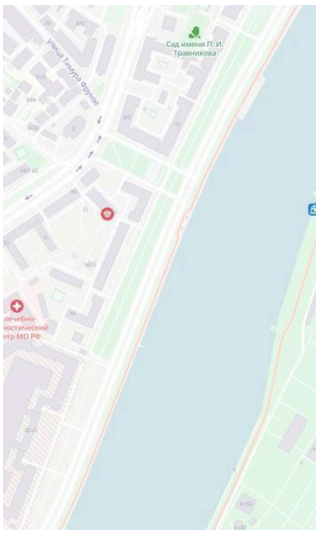
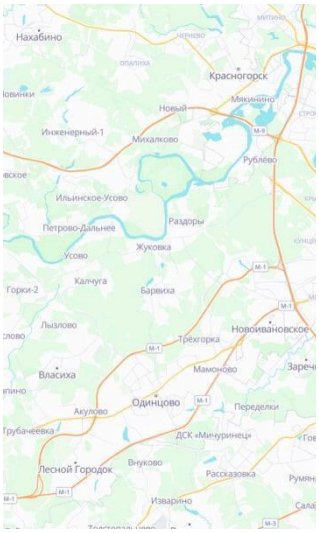
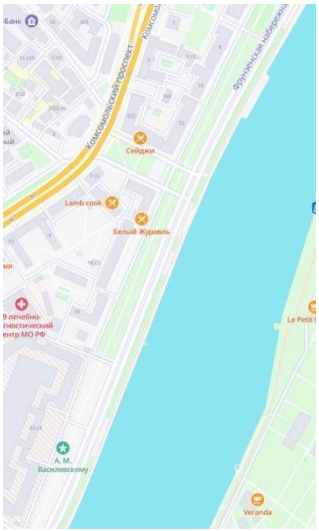
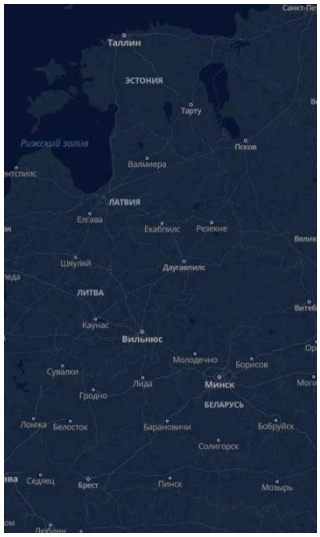
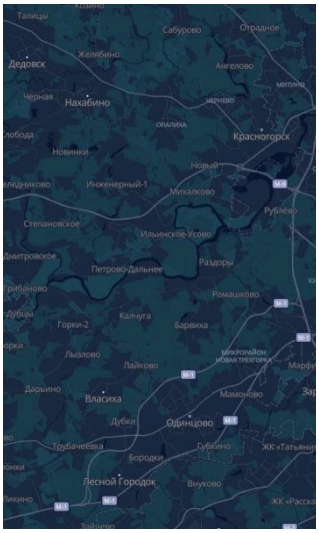
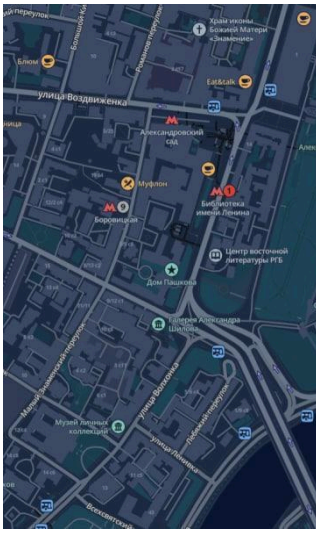


机场 + 叠加



地图样式

目前有几种地图样式可用于服务，这个列表在不断更新。如果没有一个样式满足您的需求，我们可以为您的项目单独开发一个样式。

样式	zoomlevel= 5	zoomlevel=10	zoomlevel=15
light			
main			
dark			

路线规划服务

目前，路线规划由以下服务提供：

- 路线构建；
- 距离矩阵计算；
- 可达区域计算；
- 最优路线构建。

路线构建

路线规划服务允许构建汽车、自行车和步行路线，包括两点之间的路线，并提供路线中途点的选项。

/directions — 调用路线规划服务的点，能够计算一个请求的多个替代路线并支持交通拥堵情况。

请求

请求以HTTP body中的json格式传递。Json请求包括必填字段和非必填字段。在请求的url中必须设置必填参数。

简单的json请求示例：

```
{   ``"locations":[      ``{         ``"lat":43.133200,         ``"lon":131.911300      ``},      ``{         ``"lat":50.266000,         ``"lon":127.535600   ``}   ``],   ``"costing":"auto",   ``"costing_options":{      ``"auto":{   ``"use_border_crossing":0      ``}   ``},   ``"directions_options":{      ``"units":"kilometers"   ``},   ``"id":"my_route"}
```

在这个请求中，构建了从符拉迪沃斯托克到布拉戈维申斯克的路线。它被构建为避开通过中国的路线，并添加了穿越国界的罚款。得到的路线以公里显示。

必需的url请求参数

参数名称	格式	描述	示例
api_key	hex-string	访问服务	— fa749bace6d8a3b1....

必需的HTTP body中的json字段

字段名	格式	描述	示例
locations	list	JSON数组中的路线构建点列表。点之间的路线按请求中指定的	

字段名	格式	描述	示例
		顺序构建。每个点由以下参数定义	
— "locations":[` `{ ``"lat":55.796932, ``"lon":37.537849, ` `"heading":150 ``}, ``{ `` "lat":55.865625, ``"lon":37.462290, ``"type":"via" ``}, `` { ``"lat":55.962139, ``"lon":37.406377 ``}]			
lat	float	路线点的纬度（度）。使用小数点后6位	
— "lat":55.796932			
lon	float	路线点的经度（度）。使用小数点后6位	
— "lon":37.537849			

HTTP body请求中的额外json字段

字段名	格式	描述	示例
type	string	路线点的类型。这是一个可选的请求字段，影响两个参数：在该点的掉头能力和在legs列表中创建单独的分支：- **break**（默认）— 允许在这个点掉头，将为这个点在legs列表中创建一个单独的分支- **through** — 禁止在这个点掉头，不会为这个点在legs列表中	

字段名	格式	描述	示例
		创建单独的分支- **via** — 允许在这个点掉头，不会为这个点在legs列表中创建单独的分支- **break_through** — 禁止在这个点掉头，将为这个点在legs列表中创建一个单独的分支路线的第一个和最后一个点的类型被忽略，总是被视为breakbreak	
—			`"type":"break""type":"via"`
heading	float	起始时首选的行驶方向（是请求的可选参数）。方向以北为0°，顺时针指示，北为 — 0°，东为 — 90°，南为 — 180°，西为 — 270°	
—			`"heading":150`
costing	string	路线构建的交通类型：- **auto**（默认）— 汽车；- **truck** — 卡车；- **pedestrian** — 行人；- **bicycle** — 自行车；- **taxi** — 出租车和其他允许使用公共交通车道的交通工具。	
—			`"costing":"pedestrian"`
costing_options	dict	路线计算参数列表。对于不同类型的交通工具，使用不同的选	

字段名	格式	描述	示例
		项和限制。使用路线计算参数时，必须指定将采用其参数的交通工具类型：- **auto**（默认）— 汽车；- **truck** — 卡车；- **pedestrian** — 行人；- **bicycle** — 自行车；- **taxi** — 出租车和其他允许使用公共交通车道的交通工具。	
—			`"costing_options":{ ``"auto":``{ ``"use_border_crossing":0, ``"use_tolls":0 ``}}`
units	string	响应中的距离单位：- **kilometers**（默认）— 公里；- **miles** — 英里。	
—			`"units":"miles"`
language	string	响应语言：- **ru-RU**（默认）— 俄语；- **en-US** — 英语。	
—			`"language":"en-US"`
id	string	请求的标识符，它随着响应一起返回，这使得可以准确地确定请求和响应之间的对应关系。	
—			`"id":"route_to_airport"`
directions_type	- none- maneuvers-instructions	启用驾驶操作描述。可能的值包括：- **none**（默认）— 从	

字段名	格式	描述	示例
		响应中排除驾驶操作描述；- **maneuvers** — 在响应中包含驾驶操作描述；- **instructions** — 在驾驶操作描述中添加相应语言的指令	
—			
`"directions_type": "instructions"`			
avoid_locations	list	一组需要在构建路线时避免的位置。以点数组的形式给出，指定纬度和经度。	
—			
`"avoid_locations": [``{ ``"lat":55.871899, ``"lon":37.457765 ``}, ``{ ``"lat":55.884556, ``"lon":37.441633 ``}, ``{ ``"lat":55.923995, ``"lon":37.395115 ``}]`			
date_time	list	出发点或目的地的日期和时间，以获得更精确的路线构建结果。由两个参数确定：**type** — 日期和时间的类型：**0**（默认）— 出发点的当前日期和时间，value值被忽略；**1** — 出发日期和时间；**2** — 到达日期和时间。**value** — 所需日期和时间的值，以ISO 8601格式（YYYY-MM-DDTHH:MM）表示，并根据type参	

字段名	格式	描述	示例
		数指定的出发或到达的本地时区。	
—			
`"date_time":{ ``"type":2, `` "value":"2020-12-33T21:00"}`			
alternates	integer	最大可选额外路线数量（若存在，则返回），除了主路线外。该值可在0到4之间选择，其中：**0**（默认）— 只构建一个路线，不计算其他选择；**4** — 将构建五条路线；—不允许使用小数值。	
—			
`"alternates":3`			
alternates_multi_points	boolean	用于在请求中包含途径点时可能包括构建额外路线的属性。默认值为—false。	
—			
`"alternates_multi_points":true`			
completeness	string	用于可能扩展响应的属性。可能的值包括：**minimal**（默认）— 只返回最少信息（仅转向指示，不包括边）；**enriched** — 返回所有路线边的信息。	
—			
`"completeness":"enriched"`			

Costing options

Auto

字段名	格式	描述	示例
use_unpaved	float	指示在构建路线时使用未铺设道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用土路；- **1** — 使用土路；— 允许使用小数值。默认值：0.5	— "use_unpaved":0.5
use_highways	float	指示在构建路线时使用高速公路的准备度的值。此值在0到1之间，其中：- **0** — 不使用高速公路；- **1**（默认） — 使用高速公路；— 允许使用小数值。	— "use_highways":0
use_tolls	float	指示在构建路线时使用收费道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用收费道路；- **1** — 使用收费道路；— 允许使用小数值。默认值：0.5	— "use_tolls":0
use_ferry	float	指示在构建路线时使用渡轮的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用渡轮；- **1** — 使用渡轮；— 可以使用小数值。默认值：0.5	— "use_ferry":0.1

字段名	格式	描述	示例
use_border_crossing	float	指示在构建路线时使用穿越其他国家边界的道路的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用穿越国界的道路；- **1**（默认情况下） — 使用穿越国界的道路；— 可以使用小数值。	—
`"use_border_crossing":0`			
traffic	boolean	在构建路线时考虑交通堵塞和道路事件。可能的值：- **false**（默认情况下）— 不考虑交通堵塞和道路事件；- **true** — 考虑交通堵塞和道路事件。	—
`"traffic":true`			

Truck

字段名	格式	描述	示例
weight	float	指定汽车重量（单位为吨），用于考虑道路网的许可重量限制来构建路线。默认值：21.77吨。根据全俄国家标准，道路标志编号：3.11。	—
`"weight":10`			
height	float	卡车的高度（单位为米）。默认值：4.11米。根据全俄国家标准，道路标志编号：3.13。	

字段名	格式	描述	示例
—			
"height":4.0			
width	float	卡车的宽度（单位为米）。默认值：9.07米。根据全俄国家标准，道路标志编号：3.14。	
—			
"width":11.1			
length	float	卡车的长度（单位为米）。默认值：21.64米。根据全俄国家标准，道路标志编号：3.15。	
—			
"length":19.4			
axle_load	float	轴重（单位为吨）。默认值：9.07吨。根据全俄国家标准，道路标志编号：3.12。	
—			
"axle_load":10.3			
hazmat	boolean	标志，用于确定卡车是否运输危险物质。默认值：false。根据全俄国家标准，道路标志编号：3.32。	
—			
"hazmat":true			
use_unpaved	float	指示在构建路线时使用未铺设道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用土路；- **1** — 使用土路；— 允许使用小数值。默认值：0.5	
—			
"use_unpaved":0.5			

字段名	格式	描述	示例
use_highways	float	指示在构建路线时使用高速公路的准备度的值。此值在0到1之间，其中：- **0** — 不使用高速公路；- **1**（默认） — 使用高速公路；— 允许使用小数值。	—
			`"use_highways":0`
use_tolls	float	指示在构建路线时使用收费道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用收费道路；- **1** — 使用收费道路；— 允许使用小数值。默认值：0.5	—
			`"use_tolls":0`
use_ferry	float	指示在构建路线时使用渡轮的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用渡轮；- **1** — 使用渡轮；— 可以使用小数值。默认值：0.5	—
			`"use_ferry":0.1`
use_border_crossing	float	指示在构建路线时使用穿越其他国家边界的道路的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用穿越国界的道路；- **1**（默认情况下） — 使用穿越国界的道路；— 可以使用小数值。	

字段名	格式	描述	示例
—			
`"use_border_crossing":0`			
traffic	boolean	在构建路线时考虑交通堵塞和道路事件。可能的值：- **false**（默认情况下）— 不考虑交通堵塞和道路事件；- **true** — 考虑交通堵塞和道路事件。	
—			
`"traffic":true`			

Pedestrian

字段名	格式	描述	示例
walking_speed	float	行走的平均速度。以公里/小时为单位。默认值：5.1公里/小时	
—			
`"walking_speed":3`			
use_ferry	float	指示在构建路线时使用渡轮的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用渡轮；- **1** — 使用渡轮；— 可以使用小数值。默认值：0.5	
—			
`"use_ferry":0.1`			
use_unpaved	float	指示在构建路线时使用未铺设道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用土路；- **1** — 使用土路；— 允许使用小数值。默认值：0.5	

字段名	格式	描述	示例
— "use_unpaved":0.5			
use_roads	float	指示在构建路线时使用汽车道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用高速公路；- **1**（默认） — 使用高速公路；— 允许使用小数值。默认值：0.5	
— "use_road":0			
use_border_crossing	float	指示在构建路线时使用穿越其他国家边界的道路的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用穿越国界的道路；- **1**（默认情况下） — 使用穿越国界的道路；— 可以使用小数值。	
— "use_border_crossing":0			
use_hills	float	指示遵循具有高度变化的路线的准备度的的值。这在选择合适的路线和计算ETA时起作用。此值在0到1之间，其中：- **0** — 尽可能避免地形的高低起伏，即使这会导致路线更长；- **1** — 不避免地形的高低起伏。允许使用小数值。默认值：0.5。重要的是要记住，在某些情况下，可能无法考虑use_hills来	

字段名	格式	描述	示例
		构建替代路线（例如，如果只有一个道路通往山顶）。	
—			
`"use_hills":0.25`			
step_penalty	float	指示每次上下楼梯/台阶时增加的秒数罚时的值。数值越高，对楼梯的避免程度越强。	
—			
`"step_penalty":120.0`			

Bicycle

字段名	格式	描述	示例
cycling_speed	float	自行车行驶的平均速度。以公里/小时为单位。默认值：20公里/小时	
—			
`"cycling_speed":30`			
use_ferry	float	指示在构建路线时使用渡轮的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用渡轮；- **1** — 使用渡轮；— 可以使用小数值。默认值：0.5	
—			
`"use_ferry":0.1`			
use_unpaved	float	指示在构建路线时使用未铺设道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用土路；- **1** — 使用土路；— 允许	

字段名	格式	描述	示例
		使用小数值。默认值：0.5	
—			
"use_unpaved":0.5			
use_roads	float	指示在构建路线时使用汽车道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用高速公路；- **1**（默认） — 使用高速公路；— 允许使用小数值。默认值：0.5	
—			
"use_road":0			
use_border_crossing	float	指示在构建路线时使用穿越其他国家边界的道路的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用穿越国界的道路；- **1**（默认情况下） — 使用穿越国界的道路；— 可以使用小数值。	
—			
"use_border_crossing":0			
use_hills	float	指示遵循具有高度变化的路线的准备度的的值。这在选择合适的路线和计算ETA时起作用。此值在0到1之间，其中：- **0** — 尽可能避免地形的高低起伏，即使这会导致路线更长；- **1** — 不避免地形的高低起伏。允许使用小数值。默认值：0.5。重要的是要记住，在	

字段名	格式	描述	示例
		某些情况下，可能无法考虑use_hills来构建替代路线（例如，如果只有一个道路通往山顶）。	

—

```
"use_hills":0.25
```

Taxi

字段名	格式	描述	示例
use_unpaved	float	指示在构建路线时使用未铺设道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用土路；- **1** — 使用土路；— 允许使用小数值。默认值：0.5	

—

```
"use_unpaved":0.5
```

use_highways	float	指示在构建路线时使用高速公路的准备度的值。此值在0到1之间，其中：- **0** — 不使用高速公路；- **1**（默认） — 使用高速公路；— 允许使用小数值。	

—

```
"use_highways":0
```

use_tolls	float	指示在构建路线时使用收费道路的准备度的值。此值在0到1之间，其中：- **0** — 不使用收费道路；- **1** — 使用收费道路；— 允许使用小数值。默认值：0.5	

—

```
"use_tolls":0
```

字段名	格式	描述	示例
use_ferry	float	指示在构建路线时使用渡轮的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用渡轮；- **1** — 使用渡轮；— 可以使用小数值。默认值：0.5	—
"use_ferry":0.1			
use_border_crossing	float	指示在构建路线时使用穿越其他国家边界的道路的准备度的值。这个值介于0到1之间，其中：- **0** — 不使用穿越国界的道路；- **1**（默认情况下） — 使用穿越国界的道路；— 可以使用小数值。	—
"use_border_crossing":0			

响应

基本信息

字段名	格式	描述	示例
trips	list	返回的根路线列表。在请求备选路线时包含多个元素。由下面描述的trip对象组成	
trip	dict	包含路线完整描述的字典	
status	number	错误代码	—
"status": 0			
status_message	string	错误描述	—

字段名	格式	描述	示例
"status_message": "Found 2 route(s) between points"			
units	string	响应中距离的度量单位：- **kilometers** — 公里;- **miles** — 英里。	—
"units": "kilometers"			
language	string	响应的语言：- **ru-RU** — 俄语;- **en-US** — 英语。	—
"language": "ru-RU"			
locations	list	根据请求带有额外信息的路线点列表。	—
"locations": [``{ ``"original_index": 0, ``"type": "break", ``"lat": 55.796932, ``"lon": 37.537849, ``"heading": 150 ``}, ``{ ``"original_index": 1, ``"lon": 37.462292, ``"lat": 55.865623, ``"type": "via" ` `}, ``{ `` "original_index":			

字段名	格式	描述	示例
2, ``"lon": 37.406376, ``"lat": 55.962139, ``"type": "break" ``}``]			
original_index	integer	路线点的顺序编号	— "original_index": 0
type	string	路线点的类型：- **break** — 允许掉头，这个点将在legs列表中创建一个单独的分支- **through** — 禁止掉头，这个点不会在legs列表中创建一个单独的分支- **via** — 允许掉头，这个点不会在legs列表中创建一个单独的分支 legs - **break_through** — 禁止掉头，这个点将在legs列表中创建一个单独的分支 | | --- | | “type”：“break” | | | lat | float | 路线点的纬度，以度表示。使用小数点后6位 | | | | --- | | “lat”：55.796932 | | | lon | float | 路线点的经度，以度表示。使用小数点后6位 | | | | --- | | “lon”：37.537849 | | | heading | float | 启动时的行进方向。 方向以北为0°，顺时	

字段名	格式	描述	示例
		针表示，北为—0°，东为 — 90°，南为 — 180°，西为 — 270°。				---		“heading”：150|		id	string	请求的标识符，它与响应一起返回，这允许准确匹配请求和响应。				---		“id”：“route_to_airport”	

概述

字段名	格式	描述	示例
summary	dict	关于路线的总体信息	

—

```
"summary":
{            ``"ll
_boxes":
[{
``"max_lon":
37.546925,
       ``"max_lat
":
55.962685,
       ``"min_lat
":
55.793781,
       ``"min_lon
":
37.406376
   ``}],
  ``"time":
2168.574,
   ``"length":
29.121,        ``}
```

ll_boxes	list	描述占据路线的区域的边界框集合。通常情况下这里有一个元素。当路线穿过反子午线时，返回两个框（在子午线的不同侧）。	

字段名	格式	描述	示例
—			
"ll_boxes": [{ ``"max_lon": 37.546925, ``"max_lat": 55.962685, ``"min_lat": 55.793781, ``"min_lon": 37.406376 ` `}]			
max_lon	float	包括路线的查看区域的最大经度边界。使用小数点后6位	
—			
"max_lon": 37.546925			
max_lat	float	包括路线的查看区域的最大纬度边界。使用小数点后6位	
—			
"max_lat": 55.962685			
time	float	预计所需时间，用于路线行驶。预计时间以秒为单位表示。	
—			
"time": 2168.574			
length	float	路线的总长度，以所选测量单位表示。	
—			
"length": 29.121			
min_lat	float	包括路线的查看区域的最小纬度边界。使用小数点后6位	
—			
"min_lat": 55.793781			

字段名	格式	描述	示例
min_lon	float	包括路线的查看区域的最小经度边界。使用小数点后6位	

—

```
"min_lon":
37.406376
```

路线和驾驶操作

字段名	格式	描述	示例
legs	dict	对象 legs 描述了在具有“type”:“break”属性的一对路线点之间包含的路线和驾驶操作。对于 n 个类型为 break 的路线点，响应包含 n-1 个路线描述元素（对于没有替代选项的请求）。对于带有替代选项的请求，每一对 break 点可能包含的 legs 数量少于或等于替代选项的数量。每个路线描述对象包含：字典 summary，类似于通用的，但仅涉及两个类型为 break 的路线点之间的部分；字符串 shape；maneuvers 列表。	

—

```
"legs":
[            ``{
              ``"s
hape":
"...",
    ``"summary":
{
           ``...
              ``},
```

字段名	格式	描述	示例
`` "maneuvers": [ ``{ ``... ``}, ``... ``{ ``... ``} ``] ``} ``]			
shape	string	编码的折线格式用于存储纬度和经度坐标序列的一行字符串（另见：折线路线的解码）	请参阅示例响应
edges	list	所有驾驶操作中包含的边列表（在输入请求中有“completeness”：“enriched”时返回）。	
— "edges": [``{``"id" : 12345,``}, ``...``{`` ...``}]			
id	int	边的唯一标识符（边的属性）	
— "id": 3459922			
length	float	边的长度（边的属性），默认单位为公里，单位在“units”字段中指定	
—			

字段名	格式	描述	示例
"length": 0.314			
use	string	边的类型（边的属性）：- road- ramp- turn_channel- track- driveway- alley- parking_aisle- emergency_access- drive_through- culdesac- living_street- service_road- cycleway（自行车道)- mountain_bike- sidewalk- footway- steps（楼梯，台阶）- path- pedestrian- bridleway- rest_area- service_area- other- rail—ferry- ferry- rail- bus- egress_connection- platform_connection-transit_connection	—
"use": "cycleway"			
road_class	string	道路等级（边的属性）：-motorway- trunk- primary- secondary- tertiary- unclassified- residential- service_other	—

字段名	格式	描述	示例
"road_class": "secondary"			
surface	string	根据粗糙程度递增的路面类型（边的属性）：-paved_smooth- paved- paved_rough- compacted- dirt- gravel- path- impassable	
—			
"surface": "gravel"			
toll	boolean	标记是否属于收费道路段的特征（边的属性）。仅适用于auto、motorcycle、truck、taxi等交通类型taxi	
—			
"toll": true			
speed_limit	int	最大可能速度（以km/h为单位）（边的属性）	
—			
"speed_limit": 60			
iso_code	string	国家ISO代码（边的属性）。仅当边为路线穿越边界的边界部分时指定。格式为ISO 3166-1 alpha-2（两个字母）	
—			
"iso_code": "RU"			
maneuvers	list	路线上的驾驶操作列表，包括在折线上的驾驶操作点、驾驶操作的长度、持续时间	

字段名	格式	描述	示例
		以及选定语言的驾驶操作提示	
—			
"maneuvers": [``{ ``"travel_type": "car", ``"travel_mode": "drive", ``"verbal_pre_transition_instruction": "沿M—11行驶9.5公里。", ``"verbal_transition_alert_instruction":"继续沿M—11行驶。", ``"length": 9.490, ``"toll": true, ``"instruction":"继续沿M—11行驶。", ``"end_shape_index": 459, ``"type": 8, ``"time": 421.573, ``"street_names": ["M—11"], ``"begin_shape_index": 344 ``}, ``... ``{ ``... ``}]			
travel_mode	string	路线规划模式：- **drive** — 用于交通类型 "costing"="auto" 和 "costing"="truck"	

字段名	格式	描述	示例
		-**pedestrian**- **bicycle**	
—			
`"travel_mode": "drive"`			
travel_type	string	路线规划类型。- **car** — 用于 ″travel_mode″=″drive″ 和 ″costing″=″auto″ - **tractor_trailer** — 用于 ″travel_mode″=″drive″ 和 ″costing″=″truck″ - **foot** — 用于 ″travel_mode″=″pedestrian″ 自行车导航可能的值：- hybrid, road, cross, mountain。	
—			
`"travel_type": "car"`			
verbal_pre_transition_instruction	string	可用作驾驶操作前准备的口头指令文本。	
—			
″verbal_pre_transition_instruction″: ″沿M—11行驶9,5公里。″			
verbal_transition_alert_instruction	string	可用作驾驶操作时的口头警示指令文本。	
—			
″verbal_transition_alert_instruction″: ″进入环岛，在第二个出口驶出。″			
verbal_multi_cue	boolean	当“verbal_pre_transition_instruction”中的口头指令文本	

字段名	格式	描述	示例
		包含多个连续、紧密相连的驾驶操作时，该变量值为“true”	
—			
"verbal_multi_cue": true			
verbal_post_transition_instruction	string	可用作驾驶操作完成后立即的口头指令文本	
—			
″verbal_post_transition_instruction″:″继续行驶100米。″			
verbal_succinct_transition_instruction	string	简要描述驾驶操作的文本	
instruction	string	建议用于显示驾驶操作的文本提示	
—			
″instruction″:″在列宁格勒大街左转。″			
length	float	驾驶操作在“begin_shape_index”和“end_shape_index”之间的总长度，以所选的测量单位表示	
—			
"length": 9.490			
begin_shape_index	integer	驾驶操作在折线上开始点的指示	
—			
"begin_shape_index": 344			
end_shape_index	integer	驾驶操作在折线上结束点的指示	
—			
"end_shape_index": 459			
begin_edge_index	integer	开边的索引从返回的边的数组为开始驾驶	

字段名	格式	描述	示例
		操作（在输入请求中包含“completeness”：“enriched”时返回）	
—			
"begin_edge_index": 5			
end_edge_index	integer	最后边的索引从返回的边的数组为结束驾驶操作（在输入请求中包含“completeness”：“enriched”时返回）	
—			
"end_edge_index": 9			
type	integer	驾驶操作的类型代码	
—			
"type": 8			
time	float	驾驶操作所需的估算时间，在“begin_shape_index”和“end_shape_index”之间。估算时间以秒为单位表示。	
—			
"time": 421.573			
cost	float	驾驶操作或路线的复杂度等级，以某些抽象单位表示。允许比较不同的路线和驾驶操作，并评估哪个在时间、距离等方面更“耗费”。	
street_names	list	驾驶操作的街道名称列表	
—			

字段名	格式	描述	示例
"street_names": ["M–11"]			
toll	boolean	当驾驶操作完全或部分需要通过收费道路时，变量具有“true”的值。例如：驾驶操作的一部分通过收费道路	—
"toll": true			
ferry	boolean	当驾驶操作不可能在没有使用渡轮的情况下进行时，变量具有“true”的值	—
"ferry": true			
rough	boolean	当驾驶操作完全或部分在未铺砌道路上进行时，变量具有“true”的值	—
"rough": true			
gate	boolean	当驾驶操作穿越门/路障时，变量具有'true'的值	—
"gate": true			
roundabout_exit_count	integer	环形交通出口的顺序编号	—
"roundabout_exit_count": 2			

驾驶操作的“type”值

“type”值	解码
0	kNone
1	kStart
2	kStartRight
3	kStartLeft

“type”值	解码
4	kDestination
5	kDestinationRight
6	kDestinationLeft
7	kBecomes
8	kContinue
9	kSlightRight
10	kRight
11	kSharpRight
12	kUturnRight
13	kUturnLeft
14	kSharpLeft
15	kLeft
16	kSlightLeft
17	kRampStraight
18	kRampRight
19	kRampLeft
20	kExitRight
21	kExitLeft
22	kStayStraight
23	kStayRight
24	kStayLeft
25	kMerge
26	kRoundaboutEnter
27	kRoundaboutExit
28	kFerryEnter
29	kFerryExit
30	kTransit
31	kTransitTransfer
32	kTransitRemainOn
33	kTransitConnectionStart
34	kTransitConnectionTransfer
35	kTransitConnectionDestination
36	kPostTransitConnectionDestination
37	kMergeRight
38	kMergeLeft

路线未找到

如果根据请求未找到任何内容，响应将如下所示：

```
{"error_code":171,"error":"No suitable edges near
location","status_code":400,"status":"Bad Request"}
```

示例

请求

```
curl -X POST \  ``-H "Content-type: application/json" \  ``-H "Accept:
application/json" \  ``-d
'{"locations":[{"lat":55.796932,"lon":37.537849,"heading":150},{"lat":55.865625,
"lon":37.462290,"type":"via"},{"lat":55.962139,"lon":37.406377}],"costing":"auto
","language":"ru-RU","directions_type":"instructions","id":"route_to_airport"}'
\  ``"https://geo.rustore.ru/api/directions?api_key=<YOUR_API_KEY>"
```

响应


```
{      ``"trips":[            ``{                  ``"trip":{                        ``
"locations":[                              ``{                                  ``"
type":"break",                                  ``"lat":55.796932,
                  ``"lon":37.537849,                                  ``"heading":15
0,                                  ``"city":"left",
    ``"original_index":0                              ``},
  ``{                                  ``"type":"via",
      ``"lat":55.865625,                                  ``"lon":37.46229,
                          ``"original_index":1                              ``},
                        ``{                                  ``"type":"break",
                              ``"lat":55.962139,
``"lon":37.406377,                                  ``"original_index":2
                  ``}                        ``],                        ``"legs":[
                          ``{                                  ``"maneuvers":[
                                  ``{
    ``"type":3,                                                  ``"instruction":"向东北
方向行驶。
—",                                                  ``"verbal_succinct_transition_inst
ruction":"向西北方向行驶，—然后向右转进列宁格勒大道。
",                                                  ``"verbal_pre_transition_instruction
":"向西北方向行驶，—然后向右转进列宁格勒大道。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 50 米。
",                                                  ``"time":8.72,
                          ``"length":0.048,
      ``"cost":364.824,                                                  ``"begin_shape_
index":0,                                                  ``"end_shape_index":3,
                                          ``"verbal_multi_cue":true,
                              ``"travel_mode":"drive",
                ``"travel_type":"car"                                          ``},
                                        ``{
```

```
      ``"type":10,                                                  ``"instruction":"右
转进列宁格勒大街。
",                                                  ``"verbal_transition_alert_instructi
on":"右转进列宁格勒大街。
",                                                  ``"verbal_succinct_transition_instru
ction":"右转。
",                                                  ``"verbal_pre_transition_instruction
":"右转进列宁格勒大街。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 600 米。
",                                                  ``"street_names":[
                                    ``"列宁格勒大街
"                                                  ``],
                ``"time":37.931,                                                  ``"len
gth":0.593,                                                  ``"cost":72.21,
                                    ``"begin_shape_index":3,
                      ``"end_shape_index":12,
        ``"travel_mode":"drive",                                                  ``"tra
vel_type":"car"                                          ``},
                  ``{                                                  ``"type":13,
                                          ``"instruction":"左掉头以留在列宁格勒大街
上。
",                                                  ``"verbal_transition_alert_instructi
on":"左掉头以留在列宁格勒大街上。
",                                                  ``"verbal_succinct_transition_instru
ction":"左掉头。
",                                                  ``"verbal_pre_transition_instruction
":"左掉头以留在列宁格勒大街上。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 10 公里。
",                                                  ``"street_names":[
                                    ``"列宁格勒大街
"                                                  ``],
                ``"time":535.509,                                                  ``"le
ngth":10.369,                                                  ``"cost":682.537,
                                        ``"begin_shape_index":12,
                            ``"end_shape_index":164,
              ``"travel_mode":"drive",
``"travel_type":"car"                                          ``},
                        ``{                                                  ``"type":23
,                                                  ``"instruction":"在分叉路口保持右侧车
道。.
",                                                  ``"verbal_transition_alert_instructi
on":"在分叉路口保持右侧车道。
",                                                  ``"verbal_pre_transition_instruction
":"在分叉路口保持右侧车道。
",                                                  ``"verbal_post_transition_instructio
```

n":"继续行驶 400 米
",``"time":34.122,
``"length":0.378,
``"cost":55.773,``"begin_shape_index":164,``"end_shape_index":174,
``"travel_mode":"drive",
``"travel_type":"car"
``},``{
``"type":15,``"instruction":"左转。
",``"verbal_transition_alert_instruction":"左转。
",``"verbal_succinct_transition_instruction":"左转。
",``"verbal_pre_transition_instruction":"左转。
",``"verbal_post_transition_instruction":"继续行驶 400 米。
",``"time":45.832,
``"length":0.435,
``"cost":75.448,``"begin_shape_index":174,``"end_shape_index":195,
``"travel_mode":"drive",
``"travel_type":"car"
``},``{
``"type":9,``"instruction":"右转进列宁格勒高速公路。
",``"verbal_transition_alert_instruction":"右转进列宁格勒高速公路。
",``"verbal_succinct_transition_instruction":"右转。, 然后行驶 300 米后右转。
",``"verbal_pre_transition_instruction":"右转进列宁格勒高速公路，然后行驶 300 米后右转。
",``"verbal_post_transition_instruction":"继续行驶 300 米。
",``"street_names":[
``""列宁格勒高速公路
"``],
``"time":12.615,``"length":0.289,``"cost":28.807,
``"begin_shape_index":195,
``"end_shape_index":199,
``"verbal_multi_cue":true,
``"travel_mode":"drive",``"travel_type":"car"``},
``{``"type":9,
``"instruction":"右转。

```
",                                                  ``"verbal_transition_alert_instructi
on":"右转。
",                                                  ``"verbal_succinct_transition_instru
ction":"右转，然后行驶 50 米后右转。
",                                                  ``"verbal_pre_transition_instruction
":"右转，然后行驶 50 米后右转。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 50 米。
",                                                  ``"time":9.245,
                            ``"length":0.05,
      ``"cost":91.808,                                                  ``"begin_shape_i
ndex":199,                                                  ``"end_shape_index":201,
                                        ``"verbal_multi_cue":true,
                              ``"travel_mode":"drive",
                  ``"travel_type":"car"                                                ``},
                                        ``{
        ``"type":9,                                                  ``"instruction":"右
转。
",                                                  ``"verbal_transition_alert_instructi
on":"右转。
",                                                  ``"verbal_succinct_transition_instru
ction":"右转。
",                                                  ``"verbal_pre_transition_instruction
":"右转。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 100 米。
",                                                  ``"time":40.258,
                            ``"length":0.123,
        ``"cost":759.908,                                                  ``"begin_shap
e_index":201,                                                  ``"end_shape_index":213,
                                                ``"travel_mode":"drive",
                                  ``"travel_type":"car"
            ``},                                                  ``{
                        ``"type":10,                                                  ``
"instruction":"右转。
",                                                  ``"verbal_transition_alert_instructi
on":"右转。
",                                                  ``"verbal_succinct_transition_instru
ction":"右转。
",                                                  ``"verbal_pre_transition_instruction
":"右转。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 100 米。
",                                                  ``"time":24.898,
                            ``"length":0.127,
        ``"cost":36.26,                                                  ``"begin_shape_
index":213,                                                  ``"end_shape_index":217,
```

``"travel_mode":"drive",
``"travel_type":"car"
``},
``{
``"type":15,
``"instruction":"左转。
",
``"verbal_transition_alert_instruction":"左转。
",
``"verbal_succinct_transition_instruction":"左转，然后右转进列宁格勒高速公路。
",
``"verbal_pre_transition_instruction":"左转，然后右转进列宁格勒高速公路。
",
``"verbal_post_transition_instruction":"继续行驶 20 米。
",
``"time":8.515,
``"length":0.018,
``"cost":22.563,
``"begin_shape_index":217,
``"end_shape_index":218,
``"verbal_multi_cue":true,
``"travel_mode":"drive",
``"travel_type":"car"
``},
``{
``"type":10,
``"instruction":"右转进列宁格勒高速公路/Leningrad Avenue.",
``"verbal_transition_alert_instruction":"右转进列宁格勒高速公路。
",
``"verbal_succinct_transition_instruction":"右转，然后驶入环形交叉路口，—在第三个出口转向白海街。
",
``"verbal_pre_transition_instruction":"右转进列宁格勒高速公路, Leningrad Avenue., 然后驶入环形交叉路口—在第三个出口转向白海街。
",
``"verbal_post_transition_instruction":"继续行驶 80 米。
",
``"street_names":[
``"列宁格勒高速公路
",
``"Leningrad Avenue"
``],
``"time":5.75,
``"length":0.081,
``"cost":20.979,
``"begin_shape_index":218,
``"end_shape_index":220,
``"verbal_multi_cue":true,
``"travel_mode":"drive",
``"travel_type":"car"
``},
``{
``"type":26,
``"instruction":"驶入环形交叉路口—在第三个出口转向白海街。
",
``"verbal_transition_alert_instructi

on":"驶入环形交叉路口—在第三个出口转向白海街。
",``"verbal_succinct_transition_instruction":"驶入环形交叉路口—在第三个出口转弯。
",``"verbal_pre_transition_instruction":"驶入环形交叉路口—在第三个出口转向白海街。
",``"time":20.267,
``"length":0.255,
``"cost":19.688,``"begin_shape_index":220,``"end_shape_index":239,
``"roundabout_exit_count":3,
``"travel_mode":"drive",
``"travel_type":"car"``},
``{
``"type":27,``"instruction":"从环形交叉路口转向白海街。
",``"verbal_succinct_transition_instruction":"从环形交叉路口转出.
",``"verbal_pre_transition_instruction":"从环形交叉路口转出。
",``"verbal_post_transition_instruction":"继续行驶 200 米。
",``"street_names":[
``"白海街
"``],
``"time":14.073,``"length":0.189,``"cost":24.333,
``"begin_shape_index":239,
``"end_shape_index":244,
``"travel_mode":"drive",``"travel_type":"car"``},
``{``"type":20,
``"instruction":"在出口处转向列宁格勒高速公路。
",``"verbal_transition_alert_instruction":"在出口处转向列宁格勒高速公路。
",``"verbal_pre_transition_instruction":"在出口处转向列宁格勒高速公路。
",``"verbal_post_transition_instruction":"继续行驶8公里。
",``"street_names":[
``"列宁格勒高速公路
"``],
``"time":394.793,``"length":8.069,``"cost":500.92,
``"begin_shape_index":244,
``"end_shape_index":355,
``"sign":{``"exit_br

```
anch_elements":[                                                    ``{
                                                        ``"text":"列宁格勒高速公路
"                                                       ``}
                                ``]                                                 ``}
,                                                   ``"travel_mode":"drive",
                                    ``"travel_type":"car"
            ``},                                            ``{
                            ``"type":23,
``"instruction":"保持右侧车道到谢列梅捷沃
—2.",                                                   ``"verbal_transition_alert_instr
uction":"保持右侧车道到谢列梅捷沃—2。
",                                                      ``"verbal_pre_transition_instruction
":"保持右侧车道到谢列梅捷沃—2。
",                                                      ``"verbal_post_transition_instructio
n":"继续行驶 600 米。
",                                                     ``"street_names":[
                                ``"46Н—13925"
            ``],                                                ``"time":34.818,
                                        ``"length":0.578,
                        ``"cost":38.3,
``"begin_shape_index":355,                                                  ``"end_shape
_index":372,                                            ``"sign":{

``},                                                ``"travel_mode":"drive",
                                ``"travel_type":"car"
              ``},                                          ``{
                            ``"type":24,
  ``"instruction":"直行以转向46Н—13925/国际公路。
",                                                      ``"verbal_transition_alert_instructi
on":"保持左侧车道以转向
—13925.",                                                   ``"verbal_pre_transition_ins
truction":"保持左侧车道以转向 46Н—13925, 国际公路。
",                                                      ``"verbal_post_transition_instructio
n":"继续行驶 2.5 公里。
",                                                     ``"street_names":[
                                ``"46Н—13925",
                  ``"国际公路
"                                                   ``],
              ``"time":89.576,                                                  ``"len
gth":2.654,                                             ``"cost":75.352,
                                ``"begin_shape_index":372,
                        ``"end_shape_index":384,
            ``"travel_mode":"drive",                                                ``
"travel_type":"car"                                             ``},
                    ``{                                                 ``"type":24,
                                            ``"instruction":"保持左侧车道以留在
46Н—13925/国际公路上。
```

```
",                                                  ``"verbal_transition_alert_instructi
on":"保持左侧车道以留在46Ｈ—13925。
",                                                  ``"verbal_pre_transition_instruction
":"保持左侧车道以留在46Ｈ—13925, 国际公路上。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 1.5 公里。
",                                                  ``"street_names":[
                                        ``"46Ｈ—13925",
                        ``"国际公路
"                                                  ``],
                  ``"time":50.683,                                                  ``"len
gth":1.254,                                                  ``"cost":46.611,
                                        ``"begin_shape_index":384,
                            ``"end_shape_index":403,
              ``"travel_mode":"drive",                                                  ``
"travel_type":"car"                                                  ``},
                        ``{                                                  ``"type":24,
                                                  ``"instruction":"直行到谢列梅捷沃机场
/D航站楼。
",                                                  ``"verbal_transition_alert_instructi
on":"保持左侧车道到谢列梅捷沃机场。
",                                                  ``"verbal_pre_transition_instruction
":"保持左侧车道到谢列梅捷沃机场，D航站楼。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 500 米。
",                                                  ``"time":33.139,
                              ``"length":0.46,
      ``"cost":38.839,                                                  ``"begin_shape_i
ndex":403,                                                  ``"end_shape_index":413,
                                                ``"sign":{
                                                                          ``},
                                        ``"travel_mode":"drive",
                        ``"travel_type":"car"                                                  `
`},                                                  ``{
          ``"type":23,                                                  ``"instruction":
"在分叉路口保持右侧车道。
",                                                  ``"verbal_transition_alert_instructi
on":"在分叉路口保持右侧车道。
",                                                  ``"verbal_pre_transition_instruction
":"在分叉路口保持右侧车道。
",                                                  ``"verbal_post_transition_instructio
n":"继续行驶 600 米。
",                                                  ``"time":43.991,
                              ``"length":0.61,
      ``"cost":63.974,                                                  ``"begin_shape_i
ndex":413,                                                  ``"end_shape_index":439,
                                                ``"travel_mode":"drive",
```

```
                    ``"travel_type":"car"
        ``},                                    ``{
                ``"type":4,                                    ``"inst
ruction":"您已到达目的地。
",                                          ``"verbal_transition_alert_instructi
on":"您将到达目的地。
",                                          ``"verbal_pre_transition_instruction
":"您已到达目的地。
",                                          ``"time":0.0,
                    ``"length":0.0,
    ``"cost":0.0,                                          ``"begin_shape_index"
:439,                                          ``"end_shape_index":439,
                                ``"travel_mode":"drive",
                    ``"travel_type":"car"
    ``}                                ``],                                    ``"
summary":{                                      ``"ll_boxes":[{
                    ``"min_lat":55.79378,
            ``"min_lon":37.39366,                                          ``"
max_lat":55.962686,                                          ``"max_lon":37.5469
25,                                        ``}],
      ``"has_time_restrictions":false,                                    ``"tim
e":1444.743,                                      ``"length":26.583,
                    ``"cost":3019.145                                    ``},
                                ```"shape":"ydqliBeqcrfA_KyO_AyA{EsHv^aeA~FkPl[g_A
d[q~@nO}d@bt@stBtM}_@CiHvUmr@zAsF\yEBcEKyDg@sDgAcDgKyPaBsA_Bc@iBEuBj@oBjBoDhFuOhf@
mWlu@kCzH}d@tA}f@fxAwTlo@mk@bBuU
```


### *距离矩阵*

距离矩阵构建服务允许计算ETA和出发点与目的对象对之间的距离，这些点对定义了矩阵。该服务支持不同维度的矩阵：

- 一对多；
- 多对多；
- 多对一。

**/dm** — 距离矩阵计算服务的调用点。

#### 请求

请求通过HTTP POST方法传递。POST请求的正文中应指定带有必填和选填字段的json。

简单的json示例：

```
{"sources":[{"lat":55.796932,"lon":37.537849},{"lat":55.801551,"lon":37.531575}]
,"targets":[{"lat":55.790412,"lon":37.534313},{"lat":55.788644,"lon":37.536507}]
,"costing":"pedestrian","id":"DM_Test"}
```

该请求为每对步行网络图中的起始点和终点计算时间和距离矩阵。
```

url请求的必需参数

字段名	格式	描述	示例
api_key	hex-string	访问服务	—
			api_key=fa749bace6d8a3b1....

json正文请求的必需参数

字段名	格式	描述	示例
sources	list	出发点列表，其中lat和lon—示点的纬度（使用小数点后6位）	—
			“sources”:[{“lat”: 55.796932,“lon”: 37.537849},{“lat”: 55.801551,“lon”: 37.531575}]
targets	list	目的对象列表，其中lat和lon—示点的纬度（使用小数点后6位）	—

			"targets":[{
			"lat": 55.790412,
			"lon": 37.534313
			}, {
			"lat": 55.788644,
			"lon": 37.536507
			}]

出发点（sources）和目的地（targets）的最大总数不得超过50个点。

json请求体的附加参数

字段名	格式	描述	示例
costing	string	路线构建的交通类型：- **auto**（默认）— 小型车；- **truck**	

字段名	格式	描述	示例
		（不支持）— 货车；- **pedestrian** — 步行；- **bicycle** — 骑自行车。	
—			
`"costing":"pedestrian"`			
id	string	请求的标识符，它将与响应一起返回，这允许准确匹配请求和响应	
—			
`"id":"DM_Test"`			
costing_options	dict	路线计算参数列表。对于不同类型的交通工具，使用不同的选项和限制，类似于路线构建服务中使用的选项	
—			
`"costing_options":{ ``"use_border_crossing":0, ``"use_tolls":0}`			
units	string	响应中距离的度量单位：- **kilometers**（默认）— 千米；- **miles** — 英里。	
—			
`"units":"miles"`			

响应

字段名	格式	描述	示例
id	string	请求的标识符，它将与响应一起返回，这允许准确匹配请求和响应	
—			
`"id": "DM_Test"`			

字段名	格式	描述	示例
targets	list	目的对象列表，其中lat和lon—示点的纬度（使用小数点后6位）	

—

```
"targets":
[        ``[
      ``{
       ``"lon":
37.534313,
        ``"lat":
55.790413
   ``},
 ``{
  ``"lon":
37.536507,
        ``"lat":
55.788643
   ``}        ``]
   ``]
```

sources	list	出发点列表，其中lat和lon—示点的纬度（使用小数点后6位）	

—

```
"sources":
[        ``[
      ``{
       ``"lon":
37.537849,
        ``"lat":
55.796932
   ``},
 ``{
  ``"lon":
37.531574,
        ``"lat":
55.801552
   ``}        ``]
   ``]
```

sources_to_targets	list	计算的矩阵，包含每个计算结果的出发和目的地矩阵元素编号以及ETA和EDA的计算值	

字段名	格式	描述	示例
—			
"sources_to_targets": [``[ ``{ ``"distance": 1.076, ``"time": 773, ``"to_index": 0, ``"from_index": 0 ``}, ``{ ``"distance": 1.459, ``"time": 1041, ``"to_index": 1, ``"from_index": 0 ``} ``], ``[ ``{ ``"distance": 1.828, ``"time": 1311, ``"to_index": 0, ``"from_index": 1 ``}, ``{ ``"distance": 2.211, ``"time": 1579, ``"to_index": 1, ``"from_index": 1 ``} ``] ``]			
distance	float	总路线长度，以所选的测量单位（EDA）表示	

字段名	格式	描述	示例
—			
″distance″:1.828			
time	float	路线的估计时间（ETA）。估计时间以秒为单位	
—			
"time": 1311			
to_index	integer	请求中目的地矩阵元素的序号	
—			
"to_index": 0			
from_index	integer	请求中出发点矩阵元素的序号	
—			
"from_index": 0			
units	string	响应中的距离单位	
—			
"units": "kilometers"			

如果根据请求未找到任何内容，响应将如下所示：

```
{"status_code":400,"status":"Bad Request"}
```

示例

使用curl工具的HTTP POST请求curl

```
curl -X POST \  ``-H "Content-type: application/json" \  ``-H "Accept:
application/json" \  ``-d
'{"sources":[{"lat":55.796932,"lon":37.537849},{"lat":55.801551,"lon":37.531575}
],"targets":[{"lat":55.790412,"lon":37.534313},{"lat":55.788644,"lon":37.536507}
],"costing":"pedestrian","id":"DM_Test"}'
\  ``"https://geo.rustore.ru/api/dm?api_key=<YOUR_API_KEY>"
```

可达性区域

可达性区域 — 一个服务，它可以计算从选定位置可以达到的区域（等值线）可达轮廓，这些区域是根据请求中的时间或距离限制确定的。计算的轮廓以封闭线或多边形的形式返回，可以在地图上显示。

/iso — 可达性区域计算服务的调用点。构建等值线的可能选项：

- 等时线（isochrone，路程时间等值线）；

- 等距线（isodistance，等距离线）。

请求

请求以json body中的HTTP格式传递。json请求包括必填字段和非必填字段。在请求的url中必须设置必填参数。

简单的json请求示例：

```
{   ``"locations":[      ``{         ``"lat":55.796932,         ``"lon":37.537849      ``}   ``],   ``"costing":"pedestrian",   ``"speed":5.6,   ``"contours":[     ``{         ``"time":15      ``},      ``{         ``"time":30      ``}   ``],   ``"id":"Iso_Test",   ``"generalize":5}
```

根据此请求，将在指定坐标周围生成两个等时线，限定行人从该坐标步行15分钟和30分钟可达的区域。行人的步行速度设定为5.6公里/小时。

必需的url请求参数

参数名称	格式	描述	示例
api_key	hex-string	访问服务	—

```
fa749bace6d8a3b1...
```

必需的HTTP body中的json字段

字段名	格式	描述	示例
locations	list	计算可达性矩阵的基准点，其中lat和lon为点的纬度和经度（使用小数点后6位）	—

```
"locations":[    `
`{        ``"lat":
55.796932,
``"lon":37.537849
   ``}]
```

contours	list	构建等值线的参数列表。每个参数定义一个度量（时间或距离）和可通过所选度量从locations参数定义的点达到的轮廓的颜色：- **time** — 达到轮廓边界所需的时	

字段名	格式	描述	示例
		间（从1到120分钟）。用于构建等时线；- **distance** — 公里距离（可以是小数）。一个请求中最多轮廓数：3（三）。每个轮廓只能定义一个度量：时间或距离。	

—

```
"contours":[    ``
{        ``"time":
15    ``},    ``{
       ``"time":30
    ``}]
```

HTTP body请求中的额外json字段

字段名	格式	描述	示例
speed	float	速度，公里/小时。速度只能为步行（costing=pedestrian）和自行车（costing=bicycle）等时线指定。如果未指定该字段，则使用默认速度：步行者5公里/小时，自行车20公里/小时。	

—

```
"speed":20.1
```

costing	string	路线构建的交通类型：- **auto**（默认）— 汽车；- **truck**（不支持） — 货车；- **pedestrian** — 步行；- **bicycle** — 自行车。	

—

```
"costing":"pedestr
ian"
```

字段名	格式	描述	示例
id	string	请求的标识符，它将与响应一起返回，这允许准确匹配请求和响应	—
`"id":"Iso_Test"`			
costing_options	dict	路线计算参数列表。对于不同类型的交通工具，使用不同的选项和限制，类似于路线构建服务中使用的选项	—
`"costing_options":{ ``"use_border_crossing":0, ``"use_tolls":0}`			
units	string	响应中的距离单位：- **kilometers**（默认） — 千米；- **miles** — 英里。	—
`"units":"miles"`			
generalize	float	用于Ramer-Douglas-Peucker算法中曲线轮廓简化的可接受偏差值（以米为单位）。曲线点数的减少可能导致相邻轮廓的交叉和自我交叉。	—
`"generalize":5`			
polygons	boolean	定义GeoJson结构：在响应中返回可达性轮廓的多边形还是线条：- **false**（默认）— 返回线条；- **true** — 返回多边形。	

字段名	格式	描述	示例
—			
"polygons":true			
show_locations	boolean	在返回等值线的同时，返回源点（在locations中指定）和映射到道路网络图的点（实际上等值线的起点）。可能的值：- **false**（默认）— 不返回源点和映射到道路网的点；- **true** — 返回这些点。	
—			
"show_locations":true			

响应

返回的响应符合GeoJson结构，并包含：

字段名	格式	描述	示例
id	string	请求的标识符，它将与响应一起返回，这允许准确匹配请求和响应	
—			
"id": "Iso_Test"			
properties	list	对于线条或多边形轮廓的时间和可达类型的描述，其中**contour**—请求中的时间值（以分钟为单位）	
—			
"properties": { ``"contour": 15, ``"metric": "time" ``}			
geometry	list	关于可达性轮廓及其结构的描述，包括：coordinates — 描述	

字段名	格式	描述	示例
		可达性轮廓的折线集，格式为：[经度, 纬度] - 点的经度和纬度（保留小数点后六位）；type — 结构的可能值：Polygon — 可达性多边形；LineString — 描述可达性轮廓的线条。	

—


```
"geometry":
{    ``"coordinate
s":
[        ``[
      ``37.537922,
            ``55.7
98004        ``],
       ``[
    ``37.536846,
          ``55.797
928        ``],
     ``[
  ``37.536957,
        ``55.79693
2        ``],
   ``[
``37.537849,
      ``55.796383
       ``],
 ``[             ``
37.539318,
    ``55.796932
     ``],        `
`[             ``37
.537922,
  ``55.798004
     ``]    ``],
  ``"type":
"LineString"}
```


如果根据请求未找到任何内容，响应将如下所示：


```
{"status_code":400,"status":"Bad Request"}
```


示例

请求

```
curl -X POST \  ``-H "Content-type: application/json" \  ``-H "Accept:
application/json" \  ``-d
'{"locations":[{"lat":55.796932,"lon":37.537849}],"costing":"pedestrian","contou
rs":[{"time":15,"color":"ff0000"},{"time":30,"color":"00ff00"}],"id":"Iso_Test",
"generalize":5}' \  ``"https://geo.rustore.ru/api/iso?api_key=<YOUR_API_KEY>"
```

响应


```
{    ``"id": "Iso_Test",    ``"type": "FeatureCollection",    ``"features":
[        ``{            ``"properties": {                ``"contour":
30            ``},            ``"geometry": {                "coordinates":
[[37.550148, 55.814228], [37.549480, 55.814301], [37.549141, 55.814220],
[37.548851, 55.814262], [37.546852, 55.813473], [37.545994, 55.814072],
[37.545681, 55.814106], [37.545200, 55.813931], [37.544643, 55.813137],
[37.544186, 55.812927], [37.544849, 55.812748], [37.545116, 55.812195],
[37.545120, 55.811932], [37.544849, 55.811672], [37.543846, 55.811695],
[37.543369, 55.811932], [37.543404, 55.812378], [37.543674, 55.812927],
[37.542850, 55.813446], [37.542519, 55.813602], [37.541729, 55.813808],
[37.540901, 55.813980], [37.539993, 55.814072], [37.539852, 55.814152],
[37.539547, 55.813931], [37.539261, 55.813522], [37.539066, 55.812931],
[37.538933, 55.812855], [37.537849, 55.812763], [37.532852, 55.812763],
[37.531853, 55.812664], [37.531693, 55.812771], [37.531349, 55.813435],
[37.530849, 55.813683], [37.527847, 55.813553], [37.527512, 55.813591],
[37.526852, 55.813824], [37.525101, 55.812931], [37.525043, 55.812744],
[37.525280, 55.812359], [37.525257, 55.811932], [37.525066, 55.811714],
[37.524853, 55.811634], [37.522850, 55.811668], [37.520851, 55.811581],
[37.520531, 55.811611], [37.519794, 55.811874], [37.518070, 55.811714],
[37.517254, 55.811531], [37.516560, 55.811222], [37.514851, 55.810818],
[37.514248, 55.810532], [37.512482, 55.810295], [37.511650, 55.809933],
[37.510849, 55.809696], [37.510563, 55.809216], [37.509514, 55.808933],
[37.508846, 55.808617], [37.508575, 55.808655], [37.507908, 55.808990],
[37.507851, 55.809124], [37.506790, 55.808994], [37.506432, 55.808353],
[37.506348, 55.807926], [37.506130, 55.807652], [37.505852, 55.807522],
[37.504852, 55.807327], [37.504086, 55.806927], [37.504608, 55.805935],
[37.504890, 55.804932], [37.505463, 55.804539], [37.506413, 55.803490],
[37.506596, 55.802933], [37.506027, 55.801758], [37.505489, 55.801296],
[37.505211, 55.800571], [37.504852, 55.800243], [37.503761, 55.800022],
[37.503632, 55.799149], [37.503334, 55.798931], [37.504494, 55.797573],
[37.505337, 55.796932], [37.506489, 55.795567], [37.506851, 55.795300],
[37.507401, 55.794479], [37.508106, 55.794186], [37.508385, 55.793926],
[37.507927, 55.791931], [37.508575, 55.791653], [37.509132, 55.791214],
[37.509377, 55.790932], [37.509697, 55.789928], [37.510567, 55.789654],
[37.510849, 55.789474], [37.511265, 55.789516], [37.511852, 55.789795],
[37.512287, 55.789368], [37.513012, 55.789093], [37.514851, 55.789059],
[37.516853, 55.789448], [37.517147, 55.788933], [37.517376, 55.787933],
[37.518219, 55.787567], [37.518467, 55.787548], [37.518852, 55.787048],
[37.520317, 55.787395], [37.520351, 55.786930], [37.519840, 55.785942],
[37.519733, 55.784813], [37.519814, 55.783936], [37.520851, 55.783707],
[37.522850, 55.783810], [37.523712, 55.783798], [37.524124, 55.783657],
[37.524662, 55.783737], [37.524937, 55.782932], [37.525475, 55.782558],
[37.526855, 55.782169], [37.527237, 55.781937], [37.527851, 55.781296],
[37.528114, 55.781197], [37.528477, 55.781303], [37.529850, 55.781086],
```

[37.530849, 55.781151], [37.531528, 55.780933], [37.531853, 55.780731],
[37.532845, 55.780922], [37.533852, 55.780361], [37.534851, 55.780132],
[37.535851, 55.780499], [37.536850, 55.780613], [37.537590, 55.780933],
[37.538418, 55.781933], [37.538094, 55.782177], [37.536850, 55.782494],
[37.536671, 55.782757], [37.536758, 55.783024], [37.537849, 55.783115],
[37.538853, 55.783100], [37.539192, 55.783272], [37.539516, 55.783272],
[37.539516, 55.783596], [37.539848, 55.784061], [37.542732, 55.784050],
[37.543095, 55.784172], [37.543610, 55.784172], [37.543610, 55.784691],
[37.543850, 55.785027], [37.543968, 55.784927], [37.543980, 55.784069],
[37.544151, 55.783630], [37.544151, 55.783234], [37.544548, 55.783234],
[37.544849, 55.783089], [37.545853, 55.783096], [37.546055, 55.782932],
[37.545853, 55.782459], [37.544849, 55.782249], [37.544636, 55.782143],
[37.544563, 55.781651], [37.544788, 55.780872], [37.544926, 55.780857],
[37.545509, 55.781269], [37.546185, 55.781269], [37.546432, 55.781349],
[37.547852, 55.781448], [37.548126, 55.781658], [37.548252, 55.781929],
[37.548279, 55.782932], [37.548252, 55.783337], [37.547714, 55.783791],
[37.547665, 55.783928], [37.547745, 55.785042], [37.547852, 55.785114],
[37.548851, 55.785156], [37.549404, 55.785378], [37.549690, 55.786087],
[37.549850, 55.786221], [37.551277, 55.786362], [37.551544, 55.786240],
[37.551849, 55.786335], [37.552105, 55.786674], [37.552048, 55.787128],
[37.551849, 55.787205], [37.551548, 55.787628], [37.551472, 55.787937],
[37.551559, 55.788227], [37.551849, 55.788536], [37.552174, 55.788609],
[37.552494, 55.788929], [37.552151, 55.789234], [37.550850, 55.789551],
[37.550755, 55.789837], [37.550903, 55.789989], [37.552853, 55.789989],
[37.552979, 55.789814], [37.552906, 55.788986], [37.553734, 55.789055],
[37.553852, 55.788998], [37.554855, 55.788998], [37.555008, 55.788773],
[37.555096, 55.788177], [37.555695, 55.788090], [37.555851, 55.788006],
[37.556850, 55.788013], [37.556938, 55.787930], [37.556973, 55.785805],
[37.556450, 55.785332], [37.556080, 55.783703], [37.555851, 55.783459],
[37.555420, 55.783360], [37.554760, 55.783024], [37.554733, 55.782936],
[37.555012, 55.782093], [37.555439, 55.781933], [37.555851, 55.781876],
[37.556301, 55.782482], [37.556850, 55.782600], [37.558849, 55.782555],
[37.559853, 55.782436], [37.560852, 55.782436], [37.561218, 55.782566],
[37.561405, 55.782490], [37.561852, 55.782536], [37.562847, 55.782425],
[37.563286, 55.782501], [37.563850, 55.782726], [37.564850, 55.782578],
[37.565845, 55.782696], [37.566853, 55.782688], [37.568306, 55.782932],
[37.568703, 55.783081], [37.569214, 55.783569], [37.569412, 55.783932],
[37.569366, 55.784931], [37.569077, 55.785160], [37.568851, 55.785187],
[37.568340, 55.785419], [37.567787, 55.785931], [37.568188, 55.786594],
[37.568398, 55.787384], [37.568851, 55.787849], [37.569027, 55.787930],
[37.568558, 55.788933], [37.568157, 55.789230], [37.567852, 55.789318],
[37.567600, 55.789680], [37.567585, 55.789932], [37.567852, 55.790161],
[37.568855, 55.790195], [37.569851, 55.790462], [37.570263, 55.790337],
[37.570557, 55.789932], [37.570576, 55.789658], [37.570850, 55.789375],
[37.571117, 55.789665], [37.571796, 55.789932], [37.572170, 55.790607],
[37.572571, 55.790936], [37.571640, 55.791931], [37.571121, 55.792206],
[37.570133, 55.792931], [37.569618, 55.793930], [37.568855, 55.794933],
[37.568947, 55.795933], [37.569851, 55.796734], [37.570312, 55.796932],
[37.570080, 55.797165], [37.569851, 55.797222], [37.569534, 55.797619],
[37.569134, 55.798931], [37.568851, 55.799126], [37.567608, 55.799686],
[37.567398, 55.799927], [37.567631, 55.800930], [37.568256, 55.801937],
[37.568562, 55.802933], [37.568115, 55.803204], [37.567337, 55.803417],

```
[37.566853, 55.803654], [37.566193, 55.804276], [37.565849, 55.804287],
[37.564850, 55.804539], [37.562851, 55.804642], [37.562603, 55.804688],
[37.562321, 55.804928], [37.562374, 55.805931], [37.562851, 55.806541],
[37.563114, 55.806667], [37.563370, 55.806927], [37.563446, 55.807926],
[37.563210, 55.808292], [37.562851, 55.808395], [37.561855, 55.808449],
[37.561455, 55.808323], [37.561081, 55.807697], [37.560852, 55.807652],
[37.560570, 55.807930], [37.560551, 55.808628], [37.560448, 55.808937],
[37.560158, 55.809238], [37.559292, 55.809376], [37.558571, 55.809658],
[37.557850, 55.809826], [37.557652, 55.809132], [37.557041, 55.808743],
[37.556850, 55.808704], [37.555855, 55.809750], [37.555717, 55.809799],
[37.555286, 55.810364], [37.554852, 55.811588], [37.554558, 55.811642],
[37.553555, 55.812637], [37.551849, 55.813408], [37.550850, 55.813587],
[37.550594, 55.813931], [37.550148, 55.814228]],                ``"type":
"LineString"            ``},            ``"type":
"Feature"        ``},        ``{            ``"properties":
{               ``"contour": 15            ``},            ``"geometry":
{               "coordinates": [[37.538876, 55.805958], [37.537895, 55.805969],
[37.536850, 55.806187], [37.534851, 55.806030], [37.533852, 55.806026],
[37.530849, 55.805218], [37.530231, 55.804928], [37.530651, 55.803932],
[37.530163, 55.803619], [37.530067, 55.802933], [37.529850, 55.802757],
[37.528851, 55.802734], [37.528469, 55.802933], [37.528511, 55.803272],
[37.529060, 55.803932], [37.528996, 55.804081], [37.527851, 55.804409],
[37.526024, 55.804108], [37.524857, 55.803989], [37.523849, 55.803200],
[37.522850, 55.803543], [37.521851, 55.803284], [37.521061, 55.802933],
[37.521778, 55.800926], [37.521568, 55.799931], [37.521996, 55.798927],
[37.522476, 55.798553], [37.522850, 55.798439], [37.523849, 55.798332],
[37.524094, 55.798180], [37.524254, 55.797932], [37.524155, 55.797630],
[37.523849, 55.797279], [37.523552, 55.797230], [37.522991, 55.796932],
[37.523434, 55.795933], [37.523846, 55.795574], [37.524853, 55.795555],
[37.525974, 55.795811], [37.526047, 55.795933], [37.527344, 55.796440],
[37.527584, 55.796932], [37.527706, 55.797081], [37.527851, 55.797115],
[37.528137, 55.796646], [37.528137, 55.796215], [37.528564, 55.796215],
[37.528851, 55.796078], [37.529942, 55.796028], [37.530178, 55.795601],
[37.530178, 55.795261], [37.530521, 55.795261], [37.530849, 55.795101],
[37.531178, 55.795261], [37.531521, 55.795261], [37.531521, 55.795601],
[37.531769, 55.796017], [37.533852, 55.796032], [37.534050, 55.796127],
[37.534657, 55.796127], [37.534657, 55.796738], [37.534790, 55.796989],
[37.535896, 55.796978], [37.535950, 55.794933], [37.535625, 55.794704],
[37.535076, 55.794704], [37.534946, 55.792835], [37.534645, 55.792721],
[37.534058, 55.792721], [37.533852, 55.792637], [37.531853, 55.792431],
[37.531441, 55.792336], [37.530727, 55.791931], [37.531166, 55.790932],
[37.531544, 55.790623], [37.532330, 55.790409], [37.533852, 55.790245],
[37.534039, 55.790123], [37.534431, 55.789516], [37.535152, 55.789234],
[37.535465, 55.788933], [37.535500, 55.788578], [37.535851, 55.788170],
[37.536205, 55.788574], [37.536320, 55.788933], [37.536610, 55.789173],
[37.536850, 55.789261], [37.538105, 55.789185], [37.538727, 55.788811],
[37.539024, 55.788757], [37.539921, 55.788868], [37.540951, 55.788834],
[37.541851, 55.789043], [37.542439, 55.789349], [37.542850, 55.789448],
[37.543850, 55.789272], [37.544254, 55.789337], [37.544502, 55.789280],
[37.544849, 55.789421], [37.546322, 55.789463], [37.546852, 55.789597],
[37.547276, 55.789356], [37.547852, 55.789242], [37.548000, 55.789787],
[37.548676, 55.789757], [37.549397, 55.789932], [37.549641, 55.790142],
```

```
[37.549850, 55.790207], [37.551849, 55.790260], [37.553345, 55.790436],
[37.553852, 55.790600], [37.554203, 55.791931], [37.554337, 55.792934],
[37.553852, 55.793671], [37.553501, 55.793930], [37.553276, 55.794353],
[37.553158, 55.794933], [37.552265, 55.795349], [37.551750, 55.795837],
[37.551640, 55.796932], [37.551228, 55.797928], [37.551395, 55.798931],
[37.551365, 55.799931], [37.552040, 55.800930], [37.552097, 55.801933],
[37.551849, 55.802193], [37.551334, 55.802418], [37.548805, 55.802883],
[37.548664, 55.803932], [37.547852, 55.804455], [37.547508, 55.804588],
[37.546852, 55.804703], [37.545853, 55.805096], [37.541851, 55.805519],
[37.540852, 55.805511], [37.539623, 55.805702], [37.538876,
55.805958]],                ``"type":
"LineString"            ``},            ``"type": "Feature"        ``}    ``]}
```

路线优化

路线优化 — 一个服务，允许解决旅行推销员问题并计算一组任意目标点的最优路线。计算最优路线可以使用汽车、步行和自行车网络图。

/optimal_route — 调用计算最优路线服务的点。对于输入的坐标集，构建基于时间或距离的优化路线，并按访问顺序形成坐标列表。

请求

请求以带有必需和可选参数的JSON形式传递。

简单的请求示例：

```
{"locations":[{"lon":49.22088,"lat":55.77055},{"lon":49.21999,"lat":55.77246},{"
lon":49.21933,"lat":55.77222},{"lon":49.22999,"lat":55.78246},{"lon":49.26842,"l
at":55.75043}],"costing":"pedestrian","directions_options":{"units":"miles"},"id
":"optimal_route_test"}
```

这个请求计算从“locations”列表中的第一个点开始，以最后一个点结束的最优步行路线。

请求的必要参数

字段名	格式	描述	示例
api_key	hex-string	访问服务	— `api_key=fa749bace6d8a3b1....`
locations	list	需要按照访问顺序排序的点列表。列表中的第一个和最后一个点保持不变，而其中间的点可以重新排列以优化时间路线。点的最小数量：2。	

字段名	格式	描述	示例
—			

```
"locations":[    `
`{        ``"lat":
55.77055,        `
`"lon":49.22088
 ``},    ``{
  ``"lat":55.79693
2,        ``"lon":
37.537849    ``}]
```

请求的附加参数

字段名	格式	描述	示例
costing	string	用于构建路线的交通类型。支持的值有：- **auto**（默认） — 汽车；- **truck**（不支持） — 货车；- **pedestrian** — 步行；- **bicycle** — 自行车。	
—			

```
"costing":"pedestr
ian"
```

id	string	请求的标识符，它将与响应一起返回，这允许准确匹配请求和响应	
—			

```
"id":"optimal_rout
e_test"
```

costing_options	dict	路线计算参数列表。对于不同类型的交通工具，使用不同的选项和限制，类似于路线规划服务中使用的。此外，还有一个“shortest”选项，默认设置为false。如果shortest=false，则路线按时间优化。如果	

字段名	格式	描述	示例
		shortest=true，则路线按距离优化。	
—			
"costing_options": { ``"auto":{ ``"shortest":true, ``"use_tolls":0 ``}}			
units	string	响应中距离的度量单位：- **kilometers**（默认） — 公里；- **miles** — 英里。	
—			
"units":"miles"			
fix_destination	bool	指示是否需要将locations列表中的最后一个坐标设定为终点的标志。如果fix_destination=true，则服务响应中指定的最后一个位置将确保是终点。如果fix_destination=false，则指定的最后一个位置可能是路线中的一个中间点。默认值：true。	
—			
"fix_destination": "false"			

响应

返回的响应符合GeoJson结构，并包含：

字段名	格式	描述	示例
id	string	请求的标识符，它将与响应一起返回，这允许准确匹配请求和响应	
—			

字段名	格式	描述	示例
"id": "optimal_route_test"			
trip	object map	关于路线的信息由以下参数定义：- **locations** — 路线点的列表，为了最优的时间效率进行排序；- **legs** — 关于路线的折线信息（另见：折线解码）；- **summary** — 关于路线的简要信息；- **status_message** — 执行请求状态的文字描述；- **status** — 请求执行状态；- **units** — 测量单位；- **language** — 提供驾驶操作信息的语言。	

—

```
{   ``"trip":{
  ``"locations":[
        ``{
     ``"type":"bre
ak",            ``
"lat":43.1332,
       ``"lon":13
1.9113,
 ``"original_index
":0         ``},
      ``{
    ``"type":"brea
k",             ``"
lat":50.266,
     ``"lon":127.
5356,           `
`"original_index":
1         ``}
 ``],      ``"legs
":[         ``{
        ``"summar
y":{
 ``"min_lat":43.13
1777,
```

字段名 格式 描述 示例

```
  ``"min_lon":127.
535648,
    ``"max_lat":50
.291427,
     ``"max_lon":1
32.040931,
       ``"time":58
965.898,
     ``"length":14
24.861,
    ``"cost":54399
.585            ``
},            ``"s
hape":"shape. In
this example we
skip
it."          ``}
    ``],       ``"s
ummary":{
``"min_lat":43.131
777,          ``"mi
n_lon":127.535648,
         ``"max_la
t":50.291427,
    ``"max_lon":13
2.040931,
``"time":58965.898
,          ``"lengt
h":1424.861,
   ``"cost":54399.
585      ``},
 ``"status_message
":"Found route
between
points",      ``"s
tatus":0,      ``"
units":"kilometers
",      ``"languag
e":"ru-RU"   ``},
  ``"id":"my_route
"}
```

如果根据请求未找到任何内容，响应将如下所示：

```
{"status_code":400,"status":"Bad Request"}
```

示例

请求

```
curl -X POST \   ``-H "Content-type: application/json" \   ``-H "Accept:
application/json" \   ``-d
'{"locations":[{"lon":49.22088,"lat":55.77055},{"lon":49.21999,"lat":55.77246},{
"lon":49.26842,"lat":55.75043}],"costing":"pedestrian","directions_options":{"un
its":"miles"},"id":"optimal_route_test"}'
\   ``"https://geo.rustore.ru/api/optimal_route?api_key=<YOUR_API_KEY>"
```

响应

```
{    ``"trip": {        ``"locations": [{           ``"type":
"break",           ``"lat": 55.77055,           ``"lon":
49.22088,           ``"original_index": 0        ``}, {           ``"type":
"break",           ``"lat": 55.77246,           ``"lon":
49.21999,           ``"original_index": 1        ``}, {           ``"type":
"break",           ``"lat": 55.75043,           ``"lon":
49.26842,           ``"original_index": 2        ``}],        ``"legs":
[{           ``"summary": {                ``"min_lat":
55.77065,               ``"min_lon": 49.21999,               ``"max_lat":
55.772514,               ``"max_lon": 49.220883,               ``"time":
188.317,               ``"length": 0.164,               ``"cost":
193.317           ``},           ```"shape": "u~jiBepe{
```

折线路线的解码

路线构建服务使用编码的折线格式来存储纬度和经度坐标序列作为一个字符串。折线编码显著减少了响应的大小。

要将折线解码为坐标序列，您可以使用下面提供的示例。

JavaScript

```
polyline.decode =  function (str, precision) {    ``var index = 0,        ``lat =
0,        ``lng = 0,        ``coordinates = [],        ``shift =
0,        ``result = 0,        ``byte =  null
,        ``latitude_change,        ``longitude_change,        ``factor =
Math.pow(10, precision || 6);    ``// Coordinates have variable length when
encoded, so just keep    ``// track of whether we've hit the end of the string.
In each    ``// loop iteration, a single coordinate is decoded.    ``while
(index < str.length) {        ``// Reset shift, result, and byte        ``byte
=  null ;        ``shift = 0;        ``result = 0;        ``do
{            ``byte = str.charCodeAt(index++) - 63;            ``result |= (byte
& 0x1f) << shift;            ``shift += 5;        ``} while (byte >=
0x20);        ``latitude_change = ((result & 1) ? ~(result >> 1) : (result >>
1));        ``shift = result = 0;        ``do {            ``byte =
str.charCodeAt(index++) - 63;            ``result |= (byte & 0x1f) <<
shift;            ``shift += 5;        ``} while (byte >=
0x20);        ``longitude_change = ((result & 1) ? ~(result >> 1) : (result >>
1));        ``lat += latitude_change;        ``lng +=
longitude_change;        ``coordinates.push([lat / factor, lng /
factor]);    ``}    ``return coordinates;};
```

C++ 11

```
#include <vector>constexpr double kPolylinePrecision = 1E6;constexpr double kInvPolylinePrecision = 1.0 / kPolylinePrecision;struct PointLL {  ``float lat;  ``float lon;};std::vector<PointLL> decode( const std::string& encoded) {  ``size_t i = 0;      // what byte are we looking at  ``// Handy lambda to turn a few bytes of an encoded string into an integer  ``auto deserialize = [&encoded, &i]( const int previous) {    ``// Grab each 5 bits and mask it in where it belongs using the shift    ``int byte, shift = 0, result = 0;    ``do {      ``byte =  static_cast < int >(encoded[i++]) - 63;      ``result |= (byte & 0x1f) << shift;      ``shift += 5;    ``} while (byte >= 0x20);    ``// Undo the left shift from above or the bit flipping and add to previous    ``// since its an offset    ``return previous + (result & 1 ? ~(result >> 1) : (result >> 1));  ``};  ``// Iterate over all characters in the encoded string  ``std::vector<PointLL> shape;  ``int last_lon = 0, last_lat = 0;  ``while (i < encoded.length()) {    ``// Decode the coordinates, lat first for some reason    ``int lat = deserialize(last_lat);    ``int lon = deserialize(last_lon);    ``// Shift the decimal point 5 places to the left    ``shape.emplace_back( static_cast < float >( static_cast < double >(lat) *                                        ``kInvPolylinePrecision),          ``static_cast < float >( static_cast < double >(lon) *                                        ``kInvPolylinePrecision));    ``// Remember the last one we encountered    ``last_lon = lon;    ``last_lat = lat;  ``}  ``return shape;}
```

Python

```
#!/usr/bin/env pythonimport sys#six degrees of precision in valhallainv  = 1.0 / 1e6 ;def decode(encoded):    ``"""Decodes route polyline which is returned by rose."""    ``decoded  = []    ``i  = 0    ``previous_coords  = { 'lat' :  0 ,  'lon' :  0 }    ``while i  <  len (encoded):        ``coords  = dict ()        ``for coord_name  in ( 'lat' ,  'lon' ):            ``coord  = 0            ``shift  = 0            ``byte  = 0x20            ``#### Keep decoding bytes until you have this coord.            ``while byte  > = 0x20 :                ``byte  = ord (encoded[i])  - 63                ``i  + = 1                ``coord  | = (byte & 0x1f ) << shift                ``shift  + = 5            ``#### Get the final value adding the previous offset and            ``#### remember it for the next.            ``coords[coord_name]  = previous_coords[coord_name]  + (                ``~(coord >>  1 )                ``if coord & 1                ``else (coord >>  1 )            ``)        ``#### Scale by the precision and chop off long coords.        ``#### Also flip the positions so its the far more standard        ``#### (lon, lat) instead of (lat, lon).        ``decoded.append([            ``float (f "{coords['lon'] * inv:.6f}" ),            ``float (f "{coords['lat'] * inv:.6f}" ),        ``])        ``previous_coords  = coords    ``return decodedprint decode(sys.argv[ 1 ])
```

MapsMobileSDK

MapsMaps SDK 允许您在 iOS 和 Android 应用程序中添加地图。

Android

集成

要集成 Maps SDK，您需要在项目的主 gradle- 文件中添加指向 maven 仓库的链接。

gradle 文件的 maven 仓库链接代码示例

```
allprojects {        ``repositories {                ``maven
{                      ``url
'https://maven.pkg.github.com/geors/maps-sdk-android'                        ``credentia
ls {                             ``username =   "your personal github
account"                           ``password =   "yor personal
token"                    ``}              ``}        ``}  }
```

在使用 SDK 的模块的 gradle- 文件中，您还需要按照示例添加依赖项。

添加依赖项的示例

```
implementation('ru.rustore.geo:mapkit:x.x.x')
```

其中 x.x.x — SDK 的版本。

建议使用 SDK 的最新版本。SDK 的发行版本号为 1.0.x。可以在以下地址找到 SDK 的最新版本：https://github.com/geors/maps-sdk-android/packages/1735721

使用

在使用 SDK 的其他组件之前，需要初始化地图的全局设置。最佳选择—在 Application 类的子类中进行。

全局设置对象示例

```
MapViewConfig(        ``val apiKey: String, // 用于提供对 SDK 工作的访问的唯一密钥)
```

按照下面的示例进行设置。

设置示例

```
MapGlobalConfig.setMapGlobalConfig(            ``MapViewConfig(                ``
`apiKey = apiKey        ``))
```

对象示例MapStartOptions

```
dataclassMapStartOptions(        ``val center: LatLon,    // 地图初始位置点（只考虑 lat
和 lon） ``val zoomLevel: Float,      // 缩放级别（zoomLevel）的初始值
(zoomLevel)        ``val style: MapStyle,      可以从相应的 enum 中选择或使用自己的地图
样式 ``val compassLocationMode: CompassLocationMode,     指南针设置可以从相应的 enum 中
选择 ``  ``val logoConfig: LogoConfig // 徽标配置）
```

如示例所示设置启动配置。

设置示例

```
MapGlobalConfig.setMapStartOptions(MapStartOptions(...))
```

对象示例LogoConfig

dataclassLogoConfig(``val logoAlignment: Alignment, // 徽标对齐，可从四个可能的 Alignment 选项中选择：BottomRight、BottomLeft、TopRight、TopLeft val logoAdditionalPaddings: AdditionalPaddings // 额外填充，可以用来设置水平和垂直间距)

为了使用 SDK，提供了一系列实现不同形式 View 的类：

- MapView—作为 SDK 中的主要显示界面，用于显示地图；
- ZoomView—用于显示 + 和 - 控件，可用于放大和缩小地图；
- CurrentLocationView—用于将地图聚焦于用户当前位置的按钮，并跟踪其位置；
- CompassView—显示物理设备相对于北方的方向的组件。

要显示这些控件，需要将它们放置在页面的 xml 布局文件中。

地图显示示例

```
< FrameLayout ``android:id = "@+id/mainLayout" ``android:layout_width = "match_parent" ``android:layout_height = "match_parent" ``...> ``< ru.mail.maps.sdk.views.MapView ``android:id = "@+id/mapView" ``android:layout_width = "match_parent" ``android:layout_height = "match_parent" /></ FrameLayout >
```

如果不需要其他控件，这可能就足以用于地图操作。否则，需要在包含地图的同一个layout中放置其他控件。在上面的示例中，这是mainLayout。还需要为每个控件编写地图的链接。

带控件的示例

```
< FrameLayout ``xmlns:custom = "http://schemas.android.com/apk/res-auto" ``android:id = "@+id/mainLayout" ``android:layout_width = "match_parent" ``android:layout_height = "match_parent" ``...> ``< ru.mail.maps.sdk.views.MapView ``android:id = "@+id/mapView" ``android:layout_width = "match_parent" ``android:layout_height = "match_parent" /> ``< ru.mail.maps.sdk.views.ZoomView ``android:id = "@+id/zoomView" ``android:layout_width = "wrap_content" ``android:layout_height = "wrap_content" ``... ``custom:mapView = "@+id/mapView" /> ``< ru.mail.maps.sdk.views.CompassView ``android:layout_width = "wrap_content" ``android:layout_height =
```

```
"wrap_content"                ``...                ``custom:mapView = "@+id/mapView"
/>        ``<
ru.mail.maps.sdk.views.CurrentLocationView                ``android:layout_width =
"wrap_content"                ``android:layout_height =
"wrap_content"                ``...                ``custom:mapView = "@+id/mapView"
/></ FrameLayout >
```

这样，所有控件都将显示在地图上方，并且与地图关联。要直接调用地图的方法，需要获取Map实体 — 在地图的视觉控件上调用getMapAsync方法。

地图方法直接调用示例

```
mapView= findViewById(R.id.mapView)mapView.getMapAsync { map ->    // 相同的 Map 对象
  ``...}
```

为了确保CurrentLocationView和CompassView控件的正确工作，需要向Map对象传递LocationSource接口的实现。

向Map对象传递LocationSource接口实现的示例LocationSource

```
map.setLocationSource(locationSource)
```

接口的方法LocationSource

```
funactivate(listener: (mapLocation: MapLocation) -> Unit)    // 用于启动系统获取gps定位的过程fun deactivate() // 在需要停止接收gps数据时调用
```

SDK会根据内部逻辑自动调用activate和deactivate方法。在调用activate时，会传递一个listener作为参数，每接收到一个新的gps定位就需要调用这个listener，参数类型为MapLocation。

示例

```
data    class MapLocation(        ``val latitude: Double? =    null ,    //纬度 [-90, 90]        ``val longitude: Double? =    null ,    // 经度 [-180, 180]        ``val speed: Float? =    null ,    // 速度        ``val bearing: Float? =    null ,    // 方向 [0, 360]        ``val accuracy: Float? =    null ,    // 精确度        ``val altitude: Double? =    null // 海拔高度（米））
```

该类的每个字段都可以为null，但为了确保正确的工作，需要插入系统提供的相应值。除了setLocationSource方法外，Map对象还有以下方法。

对象的方法示例Map

```
// 更改地图中心坐标// animated参数负责动画效果，duration参数负责持续时间，cameraCurve参数负责摄像机移动轨迹。// 设置动画持续时间和摄像机移动轨迹为可选项fun flyTo(mapLocation: MapLocation, animated: Boolean, durationMs: Int?, cameraCurve: CameraCurve = CameraCurve.Standard())// 放大地图, step — 为放大步长fun zoomIn(step: Float = .5f, animated: Boolean =    true )// 缩小地图, step — 为缩小步长fun zoomOut(step: Float = .5f, animated: Boolean =    true )// 设置地图方向, bearing 值在半
```

开区间 [0, 360)fun setBearing(bearing: Float, animated: Boolean = true)// 设置地图缩放级别, zoom 值在半开区间 (0, 20]fun setZoom(zoom: Float, animated: Boolean = true) // 在地图上添加标记 MarkerEntity — 为标记模型, id — 为每个标记的唯一标识符, coordinates — 为标记绑定的坐标点 (仅考虑 latitude 和 longitude, 其他为 null) 和 image — 中的值 enum MarkerImagefun addMarker(marker: MarkerEntity) // 向地图添加标记列表 addMarker(markers: List<MarkerEntity>)// 根据指定标识符删除标记fun removeMarker(id: String)// 删除地图上所有标记fun removeAllMarkers()// 显示 PopUp 窗口在标记上显示, markerId — 为窗口绑定的标记标识符; content — 为窗口中显示的HTML字符串 fun showPopUp(markerId: String, content: String) // 隐藏对应标记标识符的 PopUp 窗口 fun hidePopUp(markerId: String)// 添加点击标记时调用的callback方法 id — 为用户点击的标记标识符, location — 为对应标记的坐标 (仅考虑 latitude 和 longitude, 其他为 null)fun setOnMarkerClickListener(onCLickListener: (id: String, location: MapLocation) -> Unit)// 删除点击标记的callback方法fun removeMarkerClickListener()// 设置标记移动随相应的 id 到位置. animated — 标记移动是否需要动画 duration — 移动动画的持续时间fun moveMarker(id: String, location: MapLocation, animated: Boolean, duration: Double)// 添加在SDK出现各种错误时将被调用的回调。所有错误都是 MapErrorfun addOnErrorListener类的子类(onErrorListener: (error: MapError) -> Unit)// 将地图移动，使屏幕中心与 center 参数对齐(仅考虑 latitude 和 longitude, 其他为 null)fun setCenter(center: MapLocation, animated: Boolean)// 更改地图样式，无需重新初始化fun changeStyle(style: MapStyle)// 允许或禁止使用手势控制地图fun enableDragPan(enable: Boolean)// 允许或禁止用户更改地图缩放和旋转fun enableZoomRotate(enable: Boolean)// 设置缩放更改时调用的回调方法fun setOnZoomChangedListener(listener: (zoom: Double) -> Unit)// 删除缩放更改的回调方法fun removeZoomChangedListener()// 设置地图点击事件的回调方法fun setOnMapClickListener(onClickListener: (location: MapLocation, screenLocation: ScreenLocation) -> Unit) ``// 设置地图长按事件的回调方法fun setOnMapLongClickListener(onClickListener: (location: MapLocation, screenLocation: ScreenLocation) -> Unit)  ``// 在地图上添加新图层fun addLayer(layer: Layer)// 为地图添加新的数据源fun addMapDataSource(source: MapDataSource)// 删除数据源根据 idfun removeSource(sourceId: String)// 从地图上删除图层根据 idfun removeLayer(layerId: String)// 在地图上添加集群， Cluster 对象包含： id — 集群的标识符, markers — 标记对象列表, radius — 集群的半径（以米为单位）, textColor — 文本颜色 hex ("#ff0000"), backgroundColor — 背景颜色 hex ("#ffffff")fun addCluster(cluster: Cluster)// 根据标识符删除集群fun removeCluster(id: String)

通过这些方法，可以手动控制地图，而不依赖于SDK提供的控件。所有SDK方法都需要在main线程中调用。

回调也在main线程中返回。要使用addMapDataSource方法，需要创建一个抽象类型的MapDataSource对象，该对象可以是以下三种具体类型之一：CircleSource、GeojsonSource和PolylineSource。

```
classCircleSource(  ``val id: String,  // 数据源的唯一 id  ``val center: LatLon,  // 圆心  ``val radius: Double,  // 半径（米）  ``val steps: Int  // 圆的插值边数)class GeojsonSource(  ``val id: String,  // 数据源的唯一 id  ``val geojsonData: ByteArray  // GeoJson 字符串 ByteArray)class
```

```
PolylineSource(        ``val id: String,    // 数据源的唯一 id          ``val
polylineData: String   // 编码成字符串的折线信息)
```

限制

Android最低支持版本：: 7.0 (min （min sdk 24） 24)

iOS

集成

您可以使用Maps Swift Manager来集成PackageSDK。

通过Swift Package Manager安装SwiftPackageManager

MapsMapsSDK作为Swift Package Manager（SPM）的包可用。SPM—一个依赖项管理器，它集成在Swift构建系统中，自动化下载、编译和链接依赖项的过程。安装包请按照以下步骤操作：

1. 在Xcode中打开项目。
2. 点击File -> Add Packages。

您还可以在Project Navigator面板中选择项目文件，然后转到项目设置，选择Package Dependencies标签，并点击Add Package Dependency按钮。

在弹出的窗口中输入仓库地址：https://github.com/georsmaps-sdk-ios。

应用中地图的集成

在您将使用地图的文件中导入MapsSDK。

```
import  MapsSDK
```

开始使用地图需要：

1. 创建MapView类型的对象。
2. 配置地图。
3. 指定地图的代理。

MapView对象的创建MapView

```
let mapView = MapView()
```

MapView对象的配置MapView

要进行配置，需要创建一个MapViewConfig类型的对象，包含以下信息：

- 用于SDK的API密钥；

- 地图中心的坐标；
- 初始zoom级别；
- 瓦片样式。

设置和配置安装的示例

```
let mapConfig = MapViewConfig ( ``apiKey : "##API_KEY##", ``center : Coordinates ( lng : 33, lat : 55), ``zoomLevel : 11, ``style : . automatic)mapView . setup ( mapConfig )
```

缩放级别（zoomLevel）可以从0（最远）到22（最近）不等。瓦片样式（style）可以采用以下几种值之一：

- .automatic— 根据isDriveMode和设备当前的颜色方案自动选择样式；
- .main— 基础样式；
- .light— 亮色样式；
- .dark— 暗色样式；
- .navMain— 导航的基础样式，以道路为重点；
- .navDark— 导航的暗色样式，以道路为重点。

地图代理的指定

代理必须实现MapViewDelegate协议的要求。它将用于处理来自地图的各种事件的通知，例如：

- 地图已加载；
- 用户发起的事件已被接收（例如触摸）；
- 由地图控件发起的事件已被接收等。

地图集成到应用程序的示例

```
import MapsSDK...override func viewDidLoad () { ``let mapView = MapView () ``let mapConfig = MapViewConfig ( ``apiKey : ″##API_KEY##″ , ``center : Coordinates ( lng : 33, lat : 55), ``zoomLevel : 11, ``style : . automatic ``) ``mapView . setup ( mapConfig ) ``mapView . delegate = self ``view . addSubview ( mapView )}
```

使用和管理地图

设置用户当前参数

setCurrentLocation方法设置：

- 用户当前坐标；
- 用户观察或移动的方向（bearing）；
- 用户位置的精确度（单位为米）（accuracy）；

- 缩放级别（zoom）。此参数是可选的。

设置用户当前坐标和方向

// 不设置缩放级别mapView . setCurrentLocation (Coordinates (lng : 33, lat : 55), bearing : 0, accuracy : 0)// 设置缩放级别mapView。setCurrentLocation(Coordinates (lng : 33, lat : 55), bearing : 0, accuracy : 0, zoom : 10)

缩放级别 (zoom) 可以在 0（最远）到 22（最近）的范围内变化。

摄像机控制

要控制摄像机，使用 flyTo 方法，以及 setBearing 和 setZoom。

方法flyTo

允许更改地图中心的坐标。变化动画由 animated 参数控制，持续时间由—duration 控制。

// 不指定动画持续时间mapView . flyTo (Coordinates (lng : 33, lat : 55), animated : true)// 指定动画持续时间mapView . flyTo (Coordinates (lng : 33, lat : 55), animated : true , duration : 2)

另外，您可以传递 MapCameraOptions 类型的 options 参数，用以定义：

- 方位角 (bearing) — 地图旋转角度；
- 缩放 (zoom) — 地图的缩放级别；
- 边距 (padding) — 窗口每侧的边距，用于移动视口的中心点；
- 摄像机运动曲线 (curve) — 用于确定摄像机移动轨迹的系数；
- 重置跟随线行进模式标志 (resetFollowMode) - 参见“设置跟随线行进模式”部分。

以上每个参数都是可选的。

flyTo 方法的示例，附加 MapCameraOptions 选项MapCameraOptions

```
let options =  MapCameraOptions ( bearing :  180,  ``zoom :  12,  ``padding :  UIEdgeInsets . init ( top :  20,  left :  20,  bottom :  150,  right :  20),  ``curve :  1.42,  ``resetFollowMode :  false )mapView . flyTo ( Coordinates ( lng :  33,  lat :  55),  options :  options ,  animated :  true )
```

设置地图方向

setBearing 方法用于设置地图的方向，并且是 flyTo 方法的简化版本。

```
mapView.setBearing(90, animated: true)
```

地图上北方固定向上

如果 isNorthAlwaysUp 属性设置为 true，则禁止以任何方式旋转地图。北方将始终位于地图的上方。默认值为 false。

```
mapView.isNorthAlwaysUp = true
```

隐藏当前位置标记

isCurrentLocationHidden 属性允许在确定当前位置时隐藏位置标记。默认值为 false。

```
mapView . isCurrentLocationHidden =  true
```

设置缩放级别

setZoom 方法设置地图的缩放级别，是 flyTo 方法的简化版本。

```
mapView.setZoom(11, animated: true)
```

设置跟随模式 (followLocation, followBearingAndLocation, free)

```
mapView.mode = .followLocation
```

将区域适配到视口

```
mapView.fitBounds(northWest: coords1, southEast: coords2, animated: true)
```

地图的附加功能

设置最小和最大缩放级别

```
mapView.setMinZoom(10, maxZoom: 15)
```

启用和禁用控件

```
mapView . isZoomButtonsHidden =  truemapView . isCompassHidden =  truemapView .
isMyLocationButtonHidden =  true
```

启用和禁用手势

```
mapView . isDragPanEnabled =  truemapView . isZoomRotateEnabled =  true
```

对齐徽标

```
mapView . logoAlignment =  . bottomRightmapView . logoIgnoresSafeArea =  falsemapView
. logoInsets =  . zero
```

默认情况下，徽标显示在地图的右下角，并考虑到 safeArea。

标记

创建标记时使用 Marker 结构体。在初始化器中传递标记的唯一标识符、坐标和图片，以及（可选的）对齐方式。推荐使用尺寸为 48 x 48 像素的图片。可以使用提供的图片集，或使用自己的图片。

添加单独的标记

```
let marker1 = Marker(id: "marker_id_1", ``coords: Coordinates(lng: 33, lat: 55), ``pin: .electricPin)
let marker2 = Marker(id: "marker_id_2", ``coords: Coordinates(lng: 33, lat: 55), ``pin: .electricInfo, ``alignment: .bottomLeft)
```

在这种情况下，传递的是 SDK 中预设的图片集。如果需要，您也可以传递自己的图片。

```
let markerImage = UIImage(...)let marker3 = Marker(id: "marker_id_3", ``coords: Coordinates(lng: 33, lat: 55), ``pin: .custom(markerImage), ``alignment: .center)
```

默认情况下，标记的对齐方式设置为 .center，也就是说标记的中心与传递的坐标对齐。您可以灵活地控制这个参数，选择众多可用值中的一个，或者传递自定义的偏移量。

```
// 底部中心对齐let marker4 = Marker(id: "marker_id_4", ``coords: Coordinates(lng: 33, lat: 55), ``pin:.electricPhoto, ``alignment:.bottom)
// 带偏移的对齐// 偏移量相对于标记的中心计算let marker5 = Marker(id: "marker_id_5", ``coords: Coordinates(lng: 33, lat: 55), ``pin:.electricStar, ``alignment:.center(offsetByX: 10, byY: -10))
```

使用 addMarker(_: Marker) 方法将标记放置在地图上addMarker(_: Marker).

逐个添加标记

```
mapView.addMarker(marker1)mapView.addMarker(marker2)
```

以数组形式添加标记

```
mapView.addMarkers([marker1, marker2])
```

指定其标识符以删除单个标记

```
mapView.removeMarker(id: "marker_id_1")
```

删除所有标记

```
mapView.removeAllMarkers()
```

跟踪标记点击事件

```
funcMapDelegate:  MapViewDelegate {  ``func mapView ( _ map :  MapView , didSelectMarkerID markerID :  String ) {  ``// ...  ``// markerID - ID 选定的标记``// ...  ``}}
```

另外，点击任何标记时，会调用 MapViewDelegate.mapView(_: MapView, didReceiveEvent: MapEvent) 代理方法。在这种情况下，didReceiveEvent.type 属性的值为 .clickOnMarker。

```
funcMapDelegate:  MapViewDelegate {  ``func mapView ( _ map :  MapView , didReceiveEvent event :  String ) {  ``// ...  ``// 对于点击标记的事件``// event.type = .clickOnMarker  ``// ...  ``}}
```

聚类

可以将标记合并为聚类。聚类创建了一个新的数据源在地图上。您可以指定聚类的半径（以米为单位）、文字和背景颜色。

```
func mapViewDidLoad ( _ mapView :  MapView ) {  ``let markers : [ Marker ] = ...  ``mapView . addCluster ( markers ,  id :  "clusterId" ,  radius :  50,  textColor :  . white ,  backgroundColor :  . blue )}
```

删除聚类是通过指定聚类的标识符进行的。

```
mapView.removeCluster(id: "clusterId")
```

弹出窗口

要在地图上显示弹出窗口，需要指定将要显示的标记的标识符以及弹出窗口内的文本。

```
mapView.displayPopup(markerId: "marker_id_1", content: "Hello world")
```

要隐藏弹出窗口，需要指定标记的标识符。

```
mapView.hidePopup(markerId: "marker_id_1")
```

要在地图上选择标记后显示弹出窗口，需要实现 MapViewDelegate 的 mapView(_:, didSelectMarkerID:) 方法。

```
extension MyController :  MapViewDelegate {  ``func mapView ( _ mapView :  MapView ,  didSelectMarkerID id :  String ) {  ``mapView . displayPopup ( id :  id ,  content :  "Hello world" )  ``}}
```

样式

地图支持更换样式。.automatic 样式意味着地图将根据系统界面主题设置自动更改亮色和暗色样式。

mapView.changeStyle(.automatic)mapView.changeStyle(.dark)

GeoJSON

地图支持从 GeoJSON 数据源绘制多边形和线条。

添加数据源和绘制图层

```
let sourceData = Data(...) let source = MapDataSource(id: "sourceID", type: .geoJSON(sourceData))mapView.addSource(source)let fillLayer = MapLayer(    ``id: "fillLayer",    ``sourceID: "sourceID",    ``paint: FillPaintProperties(fillColor: .iuColor(.green), fillOpacity: .value(0.3)))let strokeLayer = MapLayer(    ``id: "strokeLayer",    ``sourceID: "sourceID",    ``paint: LinePaintProperties())mapView.addLayer(fillLayer)mapView.addLayer(strokeLayer)
```

可以在数据源中设置颜色和透明度的值。在这种情况下，使用 .source(“field”)，其中 field — 是 properties 源中的字段名称，用于获取值。可以通过指定其标识符来删除数据源和图层。

mapView.removeLayer(“fillLayer”)mapView.removeLayer(“strokeLayer”)mapView.removeSource(“sourceID”))

支持以编码字符串格式从 构建路线 服务绘制路线。

```
let routeSource = MapDataSource(id: "routeSourceID", type: .encodedString(encodedRoute))mapView.addSource(routeSource)let routeLayer = MapLayer(    ``id: "routeLineLayer",    ``sourceID: "routeSourceID",    ``paint: LinePaintProperties(lineColor: .iuColor(.red), lineWidth: 2))mapView.addLayer(routeLayer)
```

支持绘制具有渐变颜色过渡的线条。为此，需要在 MapDataSource 中启用 lineMetrics 选项，创建 LineGradient 对象，指定沿线条长度范围为 0.0 … 1.0 的 offset 列表和 color hex 字符串或 CSS 的标准颜色名称。

```
let encodedRoute: String = ...let routeSourceID = "route"let routeSource = MapDataSource(    ``id: routeSourceID,    ``type: .encodedString(encodedRoute),    ``lineMetrics: true)let gradient = LineGradient(stops: [    ``LineGradientStop(offset: 0, color: "#0000FF"),    ``LineGradientStop(offset: 0.1, color: "royalblue"),    ``LineGradientStop(offset: 0.3, color: "cyan"),    ``LineGradientStop(offset: 0.5, color: "lime"),    ``LineGradientStop(offset: 0.7, color: "yellow"),    ``LineGradientStop(offset: 1, color: "#FF0000")])let routeLayer = MapLayer(    ``id: "routeLineLayer",    ``sourceID: routeSourceID,    ``paint: LinePaintProperties(        ``lineColor: .iuColor(.green),        ``lineWidth
```

```
: 6,        ``lineGradient : gradient    ``))mapView . addSourcesAndLayers
(    ``sources : [ routeSource ],    ``layers : [ routeLayer ])
```

支持绘制圆形。由于 GeoJSON 不支持圆形，它们通过指定边数的多边形来模拟。

```
mapView.addCircleSource(center: coords, radius: 500, steps: 32, id: sourceID)
```

如果数据源(sources)和图层(layers)是预先知道的，那么您可以通过一个调用来添加它们。

```
mapView . addSourcesAndLayers (    ``sources : [        ``source
,        ``routeSource    ``],    ``layers : [        ``fillLayer
,        ``strokeLayer ,        ``routeLayer    ``])
```

交通拥堵和等值线

地图可以显示道路上的交通高堵、地铁线路和高度等级（等高线）。要启用此功能，请使用setLayoutVisible方法。

mapView.setLayoutVisible(true, layout:.traffic)mapView.setLayoutVisible(true, layout:.isolines)mapView.setLayoutVisible(true, layout:.isolinesLabel)mapView.setLayoutVisible(true, layout:.subway)

错误处理

为了处理错误，使用了mapView(_:, didFailWithError:)方法，它是MapViewDelegate的代理方法。

```
extension MyController : MapViewDelegate {    ``func mapView ( _ mapView : MapView
, didFailWithError error : Error ) {        ``print ( "Did fail with error:
\(error.localizedDescription)" )    ``}}
```

地理编码

coder — MapsSDK的组件，用于正向和反向地理编码。初始化需要API KEY。

直接地理编码示例

```
letgeocoder=  Geocoder ( apiKey :  "your apiKey" )geocoder . geocode
(        ``query :  "列宁格勒大街 39号79号楼",        ``lang :  "ru"
,        ``location :  Coordinates ( lng : 37.537892,  lat : 55.796926)) {  result
in        ``switch result {        ``case let . success ( response
):            ``print ( response )        ``case let . failure ( error
):            ``print ( error . localizedDescription )        ``}}
```

SwiftUI

在SwiftUI中使用地图，提供了Map组件。

SwiftUI中的使用示例SwiftUI

```
importSwiftUIimportMapsSDKclass StateObject :  ObservableObject {  ``let
mapConfig =  MapViewConfig (  ``apiKey :  apiKey
,  ``center :  Coordinates ( lng : 37.537892,  lat :
55.796926)  ``zoomLevel : 15,  ``style : .
automatic  ``)  ``let mapView =  MapView ()}extension StateObject :
MapViewDelegate {  ``// 代理方法}struct ContentView :  View {  ``@
StateObject private var state =  StateObject ()  ``var body :  some View
{  ``Map (  ``config :  state . mapConfig
,  ``view :  state . mapView
,  ``delegate :  state  ``)  ``}}
```

限制

MapsMapsSDK可以集成到支持iOS 12.4及更高版本的应用程序中。

从iPadOS 13开始，用户可以使用多窗口模式。MapsSDK尚未针对这种模式进行优化。

您可以轻松地将MapsSDK集成到基于UIKit或SwiftUI框架的应用程序中。

附加服务

在这个页面上收集了一些附加服务，这些服务是获取额外非标准信息所必需的。

服务

调用点	描述
/elevation	确定点的高度或沿路线的高程剖面
/ip2geo	根据IP地址确定位置
/timezone	确定地球上任意点的时区以及相对于UTC的时间偏移。
/postcode	根据地址确定邮政编码和使用该编码的地址。在俄罗斯，可以按编码搜索邮局

高度确定

/elevation — 调用服务的点，该服务提供获取海拔高度数据的能力。该服务允许请求以下高度：

- 地图上的单个点；
- 一组不相互连接的点；
- 路径 — 一系列相互连接的点（使用请求json中的resample_distance字段）。

请求

使用HTTP POST可以进行API请求。

POST请求的必要参数

字段名	格式	描述	示例
 api_key 	hex-string	访问服务	

—

```
api_key=fa749bace6
d8a3b1....
```

json	string	POST请求的正文，包含获取高度数据所需的参数。	

JSON请求

JSON在请求体中传输。

必要字段

字段	格式	描述	带有此字段的json示例	服务响应示例
locations	在此字段中，可以以三种允许的格式之一传输坐标：			

- 坐标列表：

```
[ {"lat": 56.1, "lon": 43.2}, {"lat": 56.2, "lon": 43.3}]
```

- 编码字符串：路线点以特殊方式编码成字符串（另见：折线路线的解码）。例如，路线构建服务以此格式返回折线（响应字段“shape”）。

```
"s{cplAfiz{pCa]xBxBx`AhC|gApBrz@{[h..."
```

- 字符串：一个或多个用 | 符号分隔的坐标。使用lat,lon格式的坐标，其中：
  - lat—纬度（小数点后最多6位）；
  - lon** —**经度（小数点后最多6位）。

```
152.2, 34.8|158.8, 23.0|120.0, 10.0
```

—

```
{   ``"locations":[      ``{         ``"lat":55.601897,         ``"lon":
37.581305      ``},      ``{         ``"lat":55.885040,         ``"lon":
37.680869      ``}   ``]}
```

—

```
{   ``"locations":[      ``{         ``"lat":55.601897,         ``"lon":
37.581305      ``},      ``{         ``"lat":55.885040,         ``"lon":
37.680869      ``}   ``],   ``"height":[      ``201,      ``139   ``]}
```

可选字段

字段	格式	描述	带有此字段的json示例	服务响应示例
range	Boolean	当设置标志为true时，服务不返回一维数组“height”，而返回包含值对的二维数组“range_height”。第一个值—是从前一个locations点开始的距离（以米为单位），第二个值—是高度。这个标志对于构建高度剖面以及计算上升和下降的坡度很有用。默认值：false。		

—

```
{   ``"range":
true
,   ``"locatio
ns"
:[      ``{
      ``"lat":
55.601897
,         ``"l
on":
37.581305
 ``},      ``{
        ``"la
t":55.885040
,         ``"l
on":
37.680869
 ``}   ``]}
```

—

```
{   ``"locatio
ns"
:[      ``{
      ``"lat":
55.601897
```

字段	格式	描述	带有此字段的json示例	服务响应示例
``"lon":37.581305``},``{``"lat":55.885040,``"lon":37.680869``}``],``"range_height":[``[``0,``201``],``[``32131,``139``]``]}				
resample_distance	Integer	用于指定需要获取高度的点之间距离的米数值。原始路线由连续点分割为resample_distance长度的段。服务返回这样形成的点和这些点的高度值。使用示例：有一个用户路线，需要为其建立一个每50米高度相隔的高度剖面。默认值：该字段不存在，服务返回每个指定坐标的一个高度值。		
—				
{``"resample_distance":10000				

字段	格式	描述	带有此字段的json示例	服务响应示例


```
,   ``"locatio
ns"
:[      ``{
      ``"lat" :
55.601897
,         ``"l
on" :
37.581305
 ``},      ``{
         ``"la
t" : 55.885040
,         ``"l
on" :
37.680869
 ``}   ``]}
—
{   ``"locatio
ns"
:[      ``{
      ``"lat" :
55.601897
,         ``"l
on" :
37.581305
 ``},      ``{
         ``"la
t" : 55.690027
,         ``"l
on" :
37.612137
 ``},      ``{
         ``"la
t" : 55.778150
,         ``"l
on" :
37.643109
 ``},      ``{
         ``"la
t" : 55.866264
,         ``"l
on" :
37.674221
 ``},      ``{
         ``"la
t" : 55.885040
,         ``"l
on" :
```

字段	格式	描述	带有此字段的json示例	服务响应示例
37.680869 ``} ``], ``"range_height" :[ ``[``0 , ``20 1 ``], ``[``10000 , ``14 3 ``], ``[``20000 , ``16 1 ``], ``[``30000 , ``15 1 ``], ``[``32131 , ``13 9 ``] ` `]}				
height_precision	Integer	小数点后高度值的位数。例如，为了提高高度剖面的准确性和平滑度，需要高度的小数值。可能的值：0-2。默认值：0（高度为整数）。		
— { ``"height_precision":2 , ``"locations" :[``{ ``"lat":55.601897 , ``"lon": 37.581305				

字段	格式	描述	带有此字段的json示例	服务响应示例

```
 ``},      ``{
         ``"la
t" : 55.885040
,         ``"l
on" :
37.680869
 ``}   ``]}
—
{   ``"locatio
ns"
:[      ``{
      ``"lat" :
55.601897
,         ``"l
on" :
37.581305
 ``},      ``{
         ``"la
t" : 55.885040
,         ``"l
on" :
37.680869
 ``}   ``],
``"height"
:[      ``200.
94
,      ``138.8
4   ``]}
```

响应

如果某个点无法确定高度，则会返回null，而不是以米为单位的值。

字段名	格式	描述	示例
locations	响应字段格式与请求选择的格式相同	此字段返回请求高度的坐标	“坐标列表”格式：

```
"locations" :[      ``{         ``"lat" : 55.601897 ,         ``"lon" :
37.581305      ``},      ``{         ``"lat" : 55.885040 ,         ``"lon" :
37.680869      ``}   ``]
```

“编码字符串”格式：

"s{cplAfiz{pCa]xBxBx`AhC|gApBrz@{[h..."

“字符串”格式：

152.2 , 34.8 | 158.8 , 23.0 | 120.0 , 10.0

height

—

"height" :[221 , 172 , 206 , 188 , 153]

range_height

—

"range_height" :[[0 , 221],[248 , 172],[902 , 206],[11308 , 188],[1601 , 153]]

示例

单个坐标的请求

```
curl -X POST \  ``-H "Content-type: application/json" \  ``-H "Accept:
application/json" \  ``-d '{"locations":"55.601897, 37.581305"}'
\  ``"https://geo.rustore.ru/api/elevation?api_key=<YOUR_API_KEY>"
```

响应

```
{"locations":"55.601897, 37.581305","height":[201]}
```

没有高度数据的坐标请求

```
curl -X POST \  ``-H "Content-type: application/json" \  ``-H "Accept:
application/json" \  ``-d '{"locations":[{"lat":0.0,"lon":0.0},
{"lat":0.1,"lon":0.1}]}'
\  ``"https://geo.rustore.ru/api/elevation?api_key=<YOUR_API_KEY>"
```

响应

```
{ "locations" :[{ "lat" : 0.000000 , "lon" : 0.000000 },{ "lat" : 0.100000 , "lon" :
0.100000 }], "height" :[ null , null ]}
```

通过IP确定位置IP

/ip2geo — 通过IP地址确定位置服务的调用点。

请求

请求的必要参数

字段名	格式	描述	示例
api_key	hex-string	访问服务	

—

字段名	格式	描述	示例
fa749bace6d8a3b1.. ..			

请求的附加参数

字段名	格式	描述	示例
q	string	搜索请求的正文包含IPv4协议的IP地址的十进制表示。默认情况下，如果请求中未指定所需的IP地址，则会自动确定并使用发出请求的客户端的IP地址。	—
192.168.1.1			
lang	2位字符语言代码	响应的语言是可用语言之一。默认为找到对象的区域语言。	—
lang=en			

响应

字段名	格式	描述	示例
geoid	object	内部地理 IDID	```

```
|      |
| --- |
| `"geo_id": "546"` | |
| address | string | 找到的对象地址 | |      |
| --- |
| `"address": "俄罗斯，莫斯科"` | |
| bbox | list | 找到的对象的位置边界，用于在地图上定位 | |      |
| --- |
| `"bbox":
[`<br>`    ``37.326228,`<br>`    ``55.491308,`<br>`    ``37.967428,`<br>`    ``55.9577
72`<br>`]` | |
| isocode | 2char | 根据ISO 3166-1 alpha-2的两字母国家代码 | |      |
| --- |
| `"isocode": "RU"` | |
| type | string | 对象类型 | ```

"type": "locality"
```

ref

```
|     |
| --- |
| `"ref": "1000000C4D63818"` | |
| pin | list |找到的对象的坐标（经度和纬度）| ```

"pin": [        ``37.538851,        ``55.796731]
```

示例

请求

```
https://geo.rustore.ru/api/ip2geo?api_key=<YOUR_API_KEY>&q=46.138.195.192
```

响应

```
{    ``"request": "/ip2geo?q=46.138.195.192&api_key",        ``"results":
[        ``{            ``"address": "俄罗斯，莫斯科"            ``"bbox":
[                ``37.326228,                ``55.491308,                ``37.967428,
                ``55.957772            ``],            ``"geo_id":
5506,            ``"isocode": "RU",            ``"pin":
[                ``37.617494,                ``55.750446            ``],            ``
"ref": "030000000026FCFD",            ``"type": "city"        ``}    ``]}
```

确定时区

/timezone — 用于确定地球任何点的时区以及相对于UTC的时间偏移的服务调用点。

请求

请求的必要参数

字段名	格式	描述	示例
 api_key 	hex-string	访问服务	— fa749bace6d8a3b1....
q	string	搜索请求正文。对于逆地理编码 — 坐标格式为lat，lon，其中：- **lat** — 所求点的纬度（以度为单位，使用小数点后6位）；- **lon** — 所求点的经度（以度为	

字段名	格式	描述	示例
		单位，使用小数点后6位）。	
—			
q=55.479205,37.32733			

请求的附加参数

字段名	格式	描述	示例
 timestamp 	unix timestamp	UTC时区的时间戳，用于计算时区参数。如果未指定，—则将使用当前时间。	
—			
timestamp=1264753640			

响应

字段名	格式	描述	示例
tzid	string	由CLDR项目定义的时区标识符	
—			
"tzid": "Europe/Moscow"			
tzname_short	string	时区的简称	
—			
"tzname_short": "MSK"			
utc_delta	int	UTC的请求位置的时区偏移（以秒为单位）	
—			
"utc_delta": 10800			

示例

请求

```
https://geo.rustore.ru/api/timezone?api_key=<YOUR_API_KEY>&q=55.479205,37.32733
```

响应

```
{    ``"request": "/v1/timezone?api_key&q=55.479205,37.32733",    ``"results":
[{        ``"tzid": "Europe/Moscow",        ``"utc_delta": 10800    ``}]}
```

邮政编码搜索

/postcode — 此服务允许查找：

- 使用所请求邮政编码的地址；
- 根据地址查找邮政编码；
- 对于俄罗斯，还可以额外请求有关服务邮政编码的俄罗斯邮局分部的信息。

请求

请求的必要参数

字段名	格式	描述	示例
api_key	hex-string	访问服务	—
			`api_key=fa749bace6d8a3b1....`
q	string	包含邮政编码、ref地址或需要查找邮政编码的地址的搜索请求正文。	—
			`q= 125167`
			`q=020000003792BE0E`
			q=莫斯科纳雷什金娜 5号 1号楼

请求的附加参数

字段名 | 格式 | 描述 | 示例 |
fields | fieldname1,fieldname2,….fieldnameN | 选择在响应中显示的字段。可能的值包括：- **postoffice** — 俄罗斯邮政局提供的邮政编码的详细信息;- **postcode** （默认） — 找到的/搜索的邮政编码；- **addresses** （默认） — 使用找到的邮政编码的地址。 | | |
— |

fields=postoffice,postcode,addresses
 | |
addresses | string | 对于字段addresses的响应格式：- **short** （默认） — 使用找到的邮政编码的地址的简洁层次表示。 | | |
— |

addresses=full
 | |
isocode | char2 | 根据 ISO 3166-1 alpha-2的两字母国家代码。目前仅支持 RU | | |
— |

isocode=ru
 | |

响应

字段名	格式	描述	示例
request	string	请求	—

```
"request" :
"125167"
```

results	list	找到的结果	—

```
″results″ :
[       ``{
       ``″addresses
″ :
{
     ``″country″ :″
俄罗斯″
,
     ``″localities″
:
[
            ``{
                 ``
″name″ :  ″莫斯科″
,
  ``″streets″ :
[
         ``{
         ``″buildin
gs″ :
[
     ``″22
c 1″
```

字段名	格式	描述	示例

``],

``"name" : "斯塔罗卡恰洛夫斯卡娅街"

``}

``]

``}

``

],

``"region" : "莫斯科",

``"subregion" : "南西行政区"

``}

``"postcode" : "117216"

``}]

postcode	string	找到的邮政编码	—
"postcode": 125167			
addresses	dict	地址，使用找到的邮政编码	addresses=short:

"addresses" : { ``"country" : "俄罗斯", ``"localities" : [``{ ``"name" : "莫斯科", ``"streets" : [ ``{ ``"buildings" : ["8号 1栋", "1D", "1B", "1A", "1号 1栋"], ``"name" : "斯塔罗卡恰洛夫斯卡娅街"

``}, ``{ ``"buildings" : [ "4G", "6", "6号 2号楼", "2号 1号楼", "2A" ], ``"name" : "科克特别尔斯卡娅街"

``}, ``{

```
``"buildings" : [ "16" ,  "18" ,  "20 1号楼" ,  "10 1号楼" ,  "1G" ,  "9号 2栋" ],  ``"name" :  "库利科夫斯卡娅街
"  ``}  ``]  ``}  ``],  ``"region" :  "莫斯科" ,  ``"subregion" :  "南西行政区"}
```


postoffice

—


```
"postoffice" : {  ``"address" :  "俄罗斯, 莫斯科, 南西行政区, 莫斯科, 北布托沃区, 3号小区, 格里纳街, 5Б" ,  ``"address_details" :
{  ``"building" :  "5Б" ,  ``"country" :  "俄罗斯"
,  ``"isocode" :  "RU" ,  ``"locality" :  "莫斯科"
,  ``"postal_code" :  "117216" ,  ``"region" :  "莫斯科
" ,  ``"street" :  "格里纳街" ,  ``"sublocality" :  "3号小区" ,  ``"subregion" :  "南西行政区" ,  ``"suburb" :
"北布托沃区"  ``},  ``"pin" : [  ``37.570932
,  ``55.566898  ``],  ``"ref" :  "22000000023FE098"}
```


字段描述postoffice

字段名	格式	描述	示例
address	string	完整找到的地址	

—


```
"address" :  "俄罗斯, 莫斯科, 北行政区, 莫斯科, 霍罗绍夫斯基, 列宁格勒大街, 56"
```


address_details	list	关于找到的地址的详细信息	

—


```
"address_details" :
{  ``"
building" :  "56"
,  ``"
country" :  "俄罗斯"
,  ``"
isocode" :  "RU"
,  ``"
locality" :  "莫斯科"
,  ``"
postal_code" :
"125167"
,  ``"
```

字段名	格式	描述	示例
region″ : ″莫斯科″ , street″ : ″列宁格勒大街″ , subregion″ : ″北行政区″ , suburb″ : ″霍罗绍夫斯基″}			```
opening_hours	list	邮局营业时间	—
"opening_hours" : { ``"Mo" : [ 09 : 00 - 13 : 00 , 14 : 00 - 21 : 00 ], ``"Tu" : [09 : 00 - 13 : 00 , 14 : 00 - 21 : 00], ``"We" : [ 09 : 00 - 13 : 00 , 14 : 00 - 21 : 00 ], ``"Th" : [09 : 00 - 13 : 00 , 14 : 00 - 21 : 00], ``"Fr" : [ 09 : 00 - 13 : 00 , 14 : 00 - 21 : 00 ], ``"Sa" : [10 : 00 - 18 : 00], ``"Su" : []}			
ref	hex	邮局ID。邮局ID不是固定的，可能会改变	```

```
|     |
| --- |
| `″ref″` `:  ` `″1000000C4D63818″` | |
| pin | list | 邮局坐标（经度和纬度） | ```

"pin" : [        ``37.538851 ,        ``55.796731]
```

示例

请求

https: //geo.rustore.ru/api/postcode?api_key=<YOUR_API_KEY>&q=125167&fields=*

响应


```
{    ``″request″ :   ″/postcode?q=125167&fields=*″ ,    ``″results″ :
[       ``{           ``″addresses″ : {                ``″country″ :   ″俄罗斯″
,                ``″localities″ :
[                   ``{                       ``″name″ :   ″莫斯科″
,                       ``″streets″ :
[                          ``{                                ``″buildings″ :
[                                 ``″56/2″
,                                 ``″37B″
,                                 ``″39 3号楼″
,                                 ``″39 9号楼″
,                                 ``″39″
,                                 ``″39″
,                                 ``″37″ ,                                          ``″37
8栋″ ,                                ``″39″
,                                 ``″37 12栋″
,                                 ``″39 17号楼″
,                                 ``″36 37号楼″
,                                 ``″36 30号楼″
,                                 ``″36 38号楼
″                              ``],                                ``″name″ :   ″列
宁格勒大街
″                           ``},                             ``{
         ``″buildings″ : [                                  ``″7A″
,                                 ``″5″ ,                                  ``″3″
,                                 ``″4号 4号楼″
,                                 ``″4A″ ,                                  ``″4″
,                                 ``″4″ ,                                  ``″4号
12号楼″ ,                                ``″4号 11号楼″
,                                 ``″4号 7号楼″
,                                 ``″2号 1号楼″
,                                 ``″4号 6号楼″
,                                 ``″4号 8号楼
″                              ``],                                ``″name″ :   ″新
兹科夫斯基巷
″                           ``},                             ``{
         ``″buildings″ : [                                  ``″1″
,                                 ``″5 D栋″
,                                 ``″5号 1栋″
,                                 ``″10″ ,                                  ``″3″
,                                 ``″5号 3栋
″                              ``],                                ``″name″ :   ″谢
尔约金街
```

```
"                        ``},                        ``{
        ``"buildings" : [                              ``"39 12号楼
"                          ``],                            ``"name" :   "霍
金斯基大道
"                        ``},                        ``{
        ``"buildings" : [                              ``"11"
,                                ``"11A"
,                                ``"11B"
,                                ``"3" ,                                ``"10"
,                                ``"11A号 1号楼"
,                                ``"6" ,                                ``"8号
3号楼" ,                                ``"8"
,                                ``"8号 14号楼"
,                                ``"8号 11号楼"
,                                ``"1A"                              ``],
                      ``"name" :   "阿艾罗波塔通道
"                        ``},                        ``{
        ``"buildings" : [                              ``"5号 2号楼"
,                                ``"5号 1号楼
"                          ``],                            ``"name" :   "纳
雷什金巷
"                        ``},                        ``{
        ``"buildings" : [                              ``"4"
,                                ``"5号 3栋"
,                                ``"5号 2栋"
,                                ``"5号 1栋"
,                                ``"8号 2栋"
,                                ``"8号 1栋"
,                                ``"7" ,                                ``"6"
,                                ``"3"
,                                ``"3B"                              ``],
                      ``"name" :   "康斯坦丁·西蒙诺夫街
"                        ``},                        ``{
        ``"buildings" : [                              ``"4"
,                                ``"6" ,                                ``"4A"
,                                ``"8" ,                                ``"5号
25号楼" ,                                ``"5号 22栋"
,                                ``"5号 8栋"
,                                ``"5号 к7栋"
,                                ``"5号 2栋"
,                                ``"5号 9号楼"
,                                ``"5号 11栋"
,                                ``"5号 25栋"
,                                ``"5号 26栋"
,                                ``"5号 35号楼"
,                                ``"5号 27A栋"
```

```
,                                   ``"5号 19号楼"
,                                   ``"5号 20号楼
"                                ``],                                ``"name" :   "阿
维亚齐奥尼小巷
"                             ``},                                ``{
          ``"buildings" : [                                   ``"4"
,                                   ``"5"
,                                   ``"3"                                ``],
                        ``"name" :   "埃尔多拉多巷
"                             ``},                                ``{
          ``"buildings" : [                                   ``"2A"
,                                   ``"2A 2号楼"
,                                   ``"3/5"
,                                   ``"1A" ,                                ``"2
号 1栋" ,                                ``"2号 2栋"
,                                   ``"2号 3栋"
,                                   ``"2号 5号楼
"                                ``],                                ``"name" :   "克
拉斯诺阿尔梅伊斯卡亚街
"                             ``},                                ``{
          ``"buildings" :
[                                   ``"3"                                ``],
                        ``"name" :   "斯捷潘·苏普伦街
"                             ``},                                ``{
          ``"buildings" : [                                   ``"3"
,                                   ``"4" ,                                ``"6号
2栋" ,                                ``"6号 1栋"
,                                   ``"5"                                ``],
                        ``"name" :   "史塔雷齐科夫斯基通道
"                             ``},                                ``{
          ``"buildings" : [                                   ``"3"
,                                   ``"3号 2号楼
"                                ``],                                ``"name" :   "3月
8日街1号
"                             ``},                                ``{
          ``"buildings" :
[                                   ``"3"                                ``],
                        ``"name" :   "特鲁多瓦娅大街
"                             ``},                                ``{
          ``"buildings" : [                                   ``"7"
,                                   ``"5" ,                                ``"9"
,                                   ``"9A"
,                                   ``"11"                                ``],
                          ``"name" :   "皮洛塔涅斯捷洛娃街
"                             ``}                                ``]                    ``}
```

```
            ``],                ``"region" :  "莫斯科",                ``"subregion"
:  "北行政区"            ``},            ``"postcode" :  "125167"
,            ``"postoffice" : {                ``"address" :  "俄罗斯, 莫斯科, 北行政
区, 莫斯科, 阿艾罗波塔区, 列宁格勒大街, 56",                ``"address_details" :
{                    ``"building" :  "56",                    ``"country" :  "俄罗
斯",                    ``"isocode" :  "RU",                    ``"locality" :  "
莫斯科",                    ``"region" :  "莫斯科",                    ``"street"
:  "列宁格勒大街",                    ``"subregion" :  "北行政区"
,                    ``"suburb" :  "阿艾罗波塔区
"                ``},                ``"pin" : [                    ``37.537903
,                    ``55.798518                ``],                ``"ref" :
"2100000045438C4B"            ``}        ``}    ``]}
```

RuStore Deeplinks

您可以通过deeplinks从您的应用程序打开RuStore的某些应用屏幕。

例如，您可以使用以下代码打开用户所有订阅的屏幕：

```
try {        ``startActivity(Intent(Intent.ACTION_VIEW, Uri.parse(
"rustore://profile/subscriptions" )))}   catch (ex: ActivityNotFoundException)
{        ``// Handle error when RuStore is not installed}
```

其中“rustore://profile/subscriptions”是RuStore deeplink。您可以将其替换为下面列出的任何deeplink，以打开所需的RuStore应用屏幕。

支持的浏览器

浏览器	RuStore Deeplinks支持Deeplinks
Google Chrome	是
Yandex浏览器	是
Mozilla Firefox	否
Opera	否
Mi Browser	否
Samsung Internet	否

Deeplinks列表

### 功能	**Deeplinks**
应用程序屏幕	- rustore://apps.rustore.ru/app/{packageName} — 将打开RuStore; - https://apps.rustore.ru/app/{packageName} — 如果用户设备上安装了RuStore应用程序，则打开RuStore，否则在展示台上打开应

用程序的web页面；- market://details?id={packageName} — 将提议打开RuStore或其他应用商店。其中{packageName}是应用程序的包名。

订阅屏幕 - rustore://profile/subscriptions

账户/更新屏幕 - rustore://apps.rustore.ru/updates

登录屏幕 - rustore://auth

TaskAPI

Task — 这是一个异步任务，在相应的回调（onFailure, onSuccess）中返回错误或值。

作为实现示例，使用了SDK支付方法getProducts()。

处理Task执行结果

异步执行的方法返回Task。例如，RuStoreBillingClient.getProducts()返回Task。这意味着如果方法执行成功，Task将返回ProductsResponse：

```
val task: Task<ProductsResponse> = RuStoreBillingClient.products.getProducts()
```

要获取方法执行成功的结果，请向Task添加OnSuccessListener回调：

```
val task: Task<ProductsResponse> = RuStoreBillingClient.products.getProducts()task.addOnSuccessListener {        ``// Process success}
```

要获取执行错误，请向Task添加OnFailureListener回调：

```
val task: Task<ProductsResponse> = RuStoreBillingClient.products.getProducts()task.addOnFailureListener {        ``// Process error}
```

如果需要同时获取成功结果和错误，向Task添加OnCompleteListener回调：

```
val task: Task<ProductsResponse> = RuStoreBillingClient.products.getProducts()task.addOnCompleteListener(object : OnCompleteListener<ProductsResponse> {        ``override fun onFailure(throwable: Throwable) {                ``// Process Error        ``}        ``override fun onSuccess(result: ProductsResponse) {                ``// Process success        ``}})
```

多线程

添加到Task的Callback将在应用程序的主线程中执行。如果需要在其他线程执行Callback，请在添加Callback的方法中传递您的Executor。

通过协程添加executor：

```
val task: Task<ProductsResponse> =
RuStoreBillingClient.products.getProducts()task.addOnCompleteListener(Dispatchers.IO.a
sExecutor(), object : OnCompleteListener<ProductsResponse> {        ``override fun
onFailure(throwable: Throwable) {                ``// Process
Error        ``}        ``override fun onSuccess(result: ProductsResponse)
{                ``// Process success        ``}})
```

同步执行

如果您的代码已在工作线程中运行，并且您需要同步获取结果，您可以使用task.await()：

```
try {        ``val task: Task<ProductsResponse> =
RuStoreBillingClient.products.getProducts()        ``task.await()} catch (e:
CancellationException) {        ``// Process error}
```

Await()方法会阻塞正在执行它的线程。如果您使用Kotlin，建议使用协程，或者如果您使用的语言不支持协程，则应在后台线程中调用此方法。

建议仅在以下SDK版本之后使用*await()*方法：

- In-app支付SDK - billingclient:1.1.1
- 推送通知集成SDK - pushclient:0.1.8
- 评论和评分集成SDK - review:0.1.6
- 应用程序更新SDK - appup

对于更早版本的SDK调用*await()*方法可能导致电池使用量增加！

通过协程处理Task API

可以使用以下代码在协程中处理Task：

```
suspend fun <T> Task<T>.wrapInCoroutine(): Result<T> {        ``return
suspendCancellableCoroutine { continuation
->                ``addOnCompleteListener(object : OnCompleteListener<T>
{                        ``override fun onSuccess(result: T)
{                                ``if (continuation.isActive)
{                                        ``continuation.resume(Result.success(result))
                                ``}                        ``}
``override fun onFailure(throwable: Throwable) {                                ``if
(continuation.isActive)
{                                        ``continuation.resume(Result.failure(throwabl
e))                                ``}                        ``}                ``})
```

```
        ``continuation.invokeOnCancellation
{               ``cancel()          ``}     ``}}
```

与其他SDK的兼容性

使用的SDK版本必须兼容。以下是版本兼容性表。

RuStore SDK 用于 in-app 支付	RuStore SDK 用于集成评论和评分	RuStore 应用程序更新SDK	RuStore SDK用于集成推送通知
3.0.0	1.0.0	1.0.0	1.0.0

使用上述任何一个版本的SDK时，必须使用表中指定的其他SDK版本（不低于指定版本）。

可用SDK列表

平台	SDK用于 in-app支付	SDK用于集成推送通知	SDK用于通用推送通知	SDK用于集成评论和评分	SDK用于应用程序更新
Kotlin	文档 示例	文档 示例	文档 示例	文档 示例	文档 示例
Java	文档 示例	文档 示例	文档 示例	文档 示例	文档 示例
Unity	文档 示例	文档		文档	文档
Flutter	文档 示例	文档 示例		文档 示例	文档 示例
Unreal Engine	文档 示例				
Godot	文档 示例				
React Native	文档 示例				
Defold		文档 示例			

RuStore API

目录

使用RuStore API的操作指南 608
RuStore API可用方法列表 608
在RuStore API中的授权和工作原理 610
生成密钥对 610
获取令牌 615
授权成功 615
授权令牌获取方法 616
交互参数 616
使用 API 获取支付和订阅数据 619
通过购买令牌获取支付数据 619
通过订阅令牌获取订阅数据 627
通过订阅令牌获取订阅数据（V2） 631
通过订阅令牌获取订阅状态 634
通过订阅令牌确认收到订阅 635
使用API RuStore上传和发布应用程序 636
创建应用程序版本草稿 637
上传APK文件 640
上传应用程序图标 643
上传应用程序截图 644
修改发布设置 645
获取应用程序版本状态 647
发送应用程序版本草稿进行审核 650
手动发布 651
删除草稿 652
应用类别列表 652
使用RuStore API处理评论 654
处理用户评论的一般流程 654
获取应用程序评论 654
获取.csv格式的评论。 658
对评论进行回复 660
获取对评论回复的状态 661
获取对多个评论回复的状态 662

更改对评论的回复..... 664
删除对评论的回复..... 665
获取应用程序评分..... 666

使用RuStore API的操作指南

API使用的宿主 -public-api.rustore.ru

RuStore API可用方法列表

名称	方法	描述
使用API获取支付和订阅数据（通用方法）		
通过购买令牌获取支付数据	GEThttps://public-api.rustore.ru/public/purchase/{purchaseToken}	此方法允许通过购买令牌获取支付信息。
通过订阅令牌获取订阅数据	GEThttps://public-api.rustore.ru/public/subscription/{subscriptionToken}	此方法允许通过订阅令牌获取订阅信息。
通过订阅令牌获取订阅状态	GET https://public-api.rustore.ru/public/subscription/{subscriptionToken}/state	此方法允许通过订阅令牌获取订阅状态信息。
使用API获取支付和订阅数据（应用程序方法）		
通过订阅令牌获取订阅数据（V2）	GEThttps://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{subscriptionToken}	此方法允许通过订阅令牌获取订阅信息。
通过订阅令牌确认收到订阅	POST https://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{purchaseToken}:acknowledge	此方法允许通过订阅令牌确认用户内容的发放。
使用API RuStore上传和发布应用程序		
创建应用程序版本草稿	POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version	此方法允许创建应用程序版本草稿并填写其主要信息。
上传APK文件	POSThttps://public-api.rustore.ru/public/v1/applic	此方法允许上传.apk扩展名的文件以便发布应用程序。

	ation/{packageName}/version/{versionId}/apk	
上传应用程序图标	POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/icon	此方法允许上传应用程序图标。
上传应用程序截图	POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/screenshot/{orientation}/{ordinal}	此方法允许上传应用程序截图。
修改发布设置	POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/publish-settings	此方法允许更改发布类型、延迟发布日期和版本部分发布的百分比。
获取应用程序版本状态	GEThttps://public-api.rustore.ru/public/v1/application/{packageName}/version?ids=704095&page=0&size=2	此方法允许获取版本的基本信息，并检查版本状态。
发送应用程序版本草稿进行审核	POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/commit?priorityUpdate={priorityUpdate}	用于提交应用程序版本草稿进行审核的方法。
手动发布	POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/publish	此方法允许手动发布已通过审核的版本。
删除草稿	DELETEhttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}	用于删除之前创建的草稿的方法。
使用RuStore API处理评论		
获取应用程序评论	GET https://public-api.rustore.ru/public/v1/application/{packa	此方法允许获取您的应用程序的所有最新评论列表或单独获取某个评论。

	geName}/comment?id={id}&page={number}&size={size}	
获取.csv格式的评论。csv	GET https://public-api.rustore.ru/public/v1/application/{packageName}/comment/export?from={date_from}&to={date_to}	此方法允许在特定时间内以.csv格式获取所有评论。
对评论进行回复	POST https://public-api.rustore.ru/public/v1/application/{packageName}/feedback?commentId={commentId}	此方法允许对评论进行回复。
获取对评论回复的状态	GET https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}	此方法允许获取对评论回复的审核状态。
获取对多个评论回复的状态	GET https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/?id={id}&page={number}&size={size}	此方法允许获取对单个评论回复的审核状态或获取特定回复的信息。
更改对评论的回复	POST https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}	此方法允许更改对评论的回复文本。
删除对评论的回复	DELETE https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}	此方法允许删除对评论的回复。
获取应用程序评分	POST https://public-api.rustore.ru/public/v1/application/{packageName}/comment/statistic	此方法允许获取应用程序的评分。

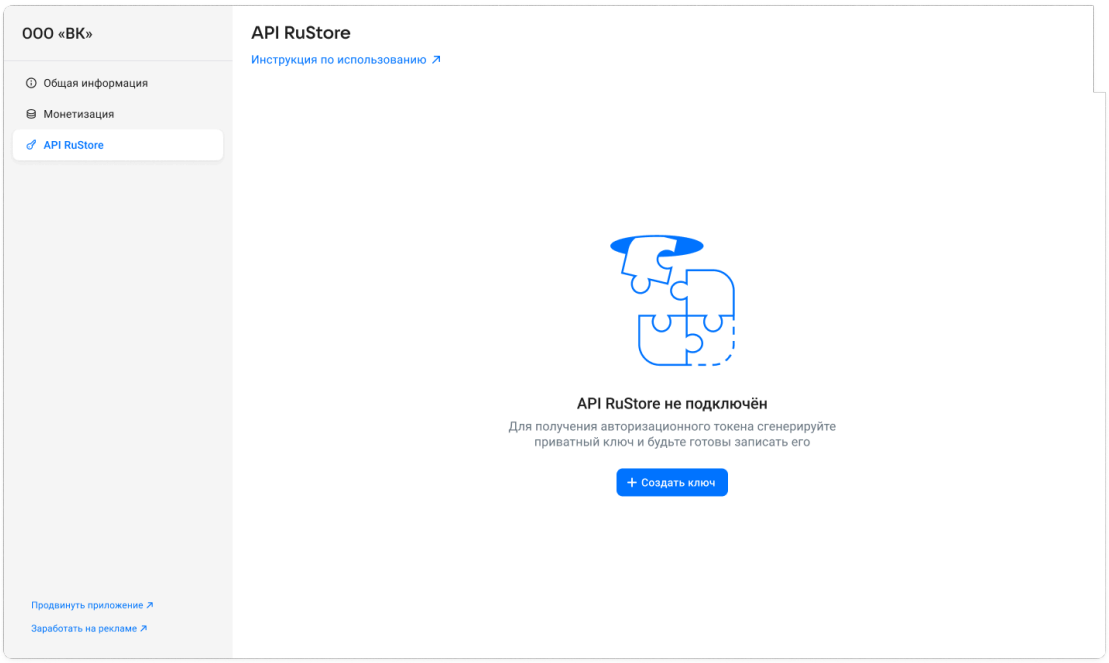
在RuStore API中的授权和工作原理

要获得使用RuStore API功能的访问权限，请进行授权。为此，在RuStore 控制台生成一对密钥 — 公钥和私钥 — 并获得JWE令牌。

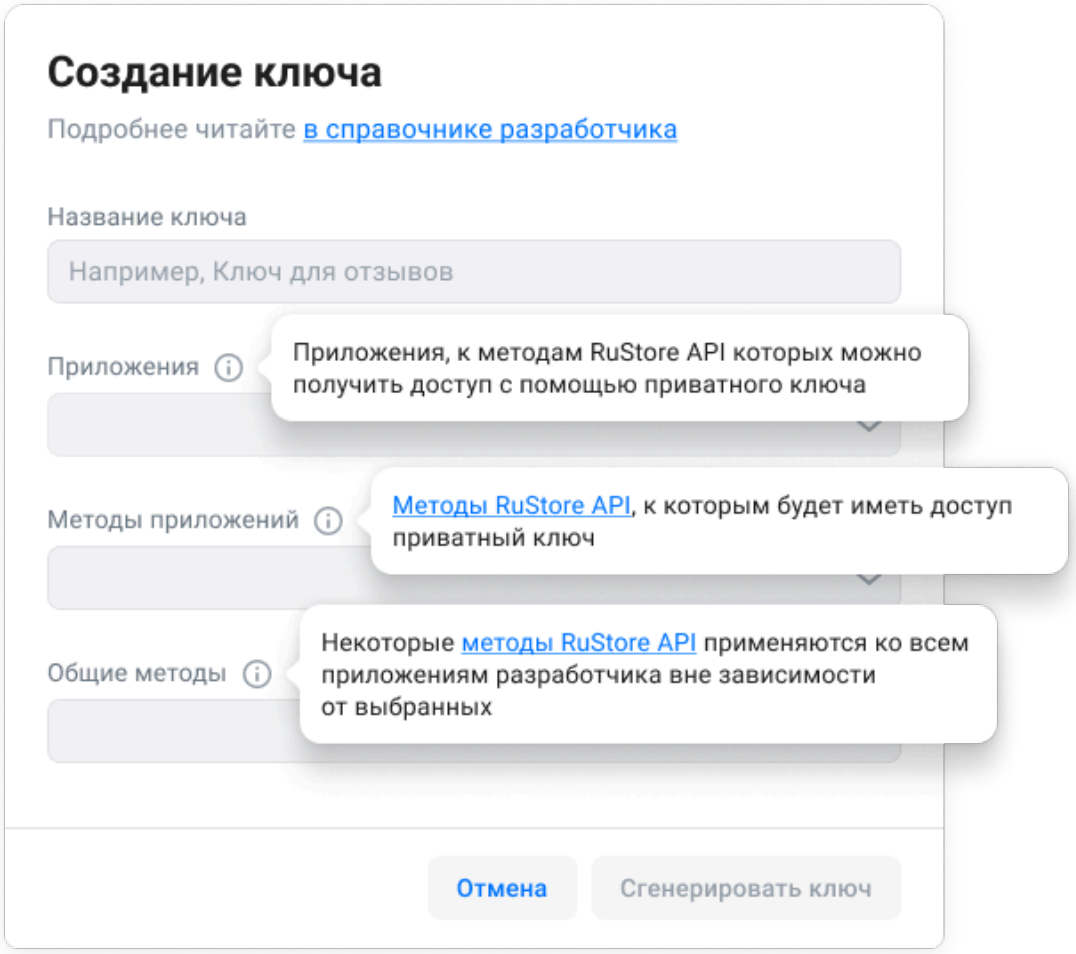
生成密钥对

用于生成密钥的是RSA加密算法。

1. 打开RuStore 控制台系统。
2. 在法人账户的屏幕顶部，点击“公司”选项卡。对于个人账户，这个选项卡被称为“开发者”。
3. 在左侧菜单中选择“API RuStore”。
4. 点击“创建密钥”。

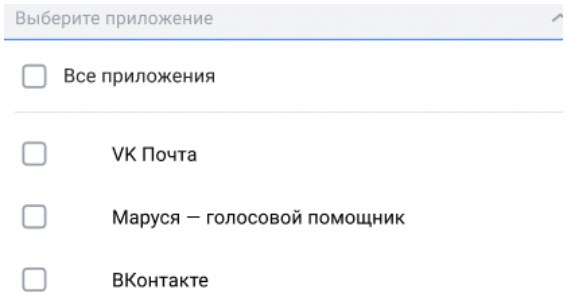


5. 输入密钥名称 — 不超过 255 个字符。

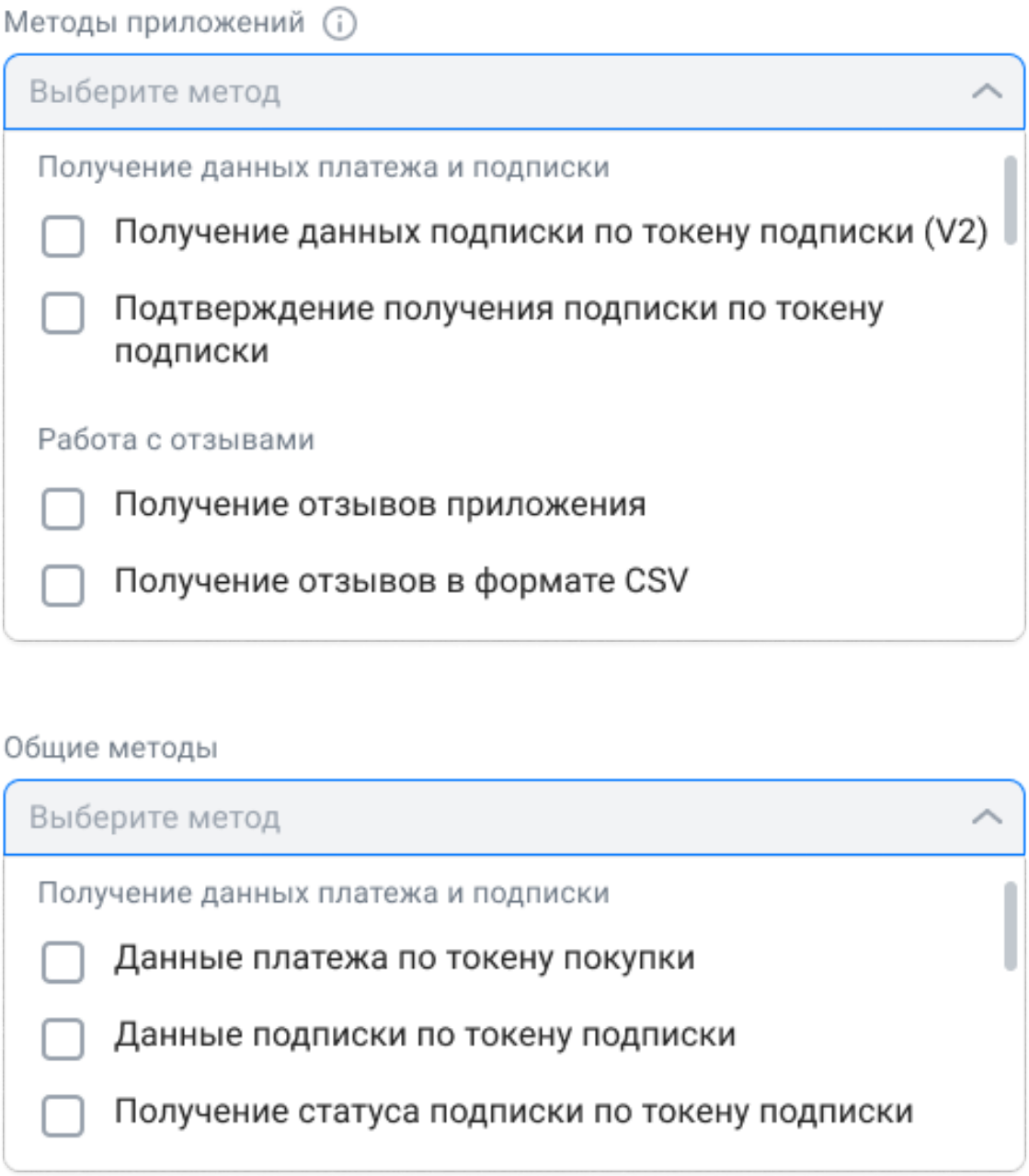


6. 选择要分发私钥的应用程序。

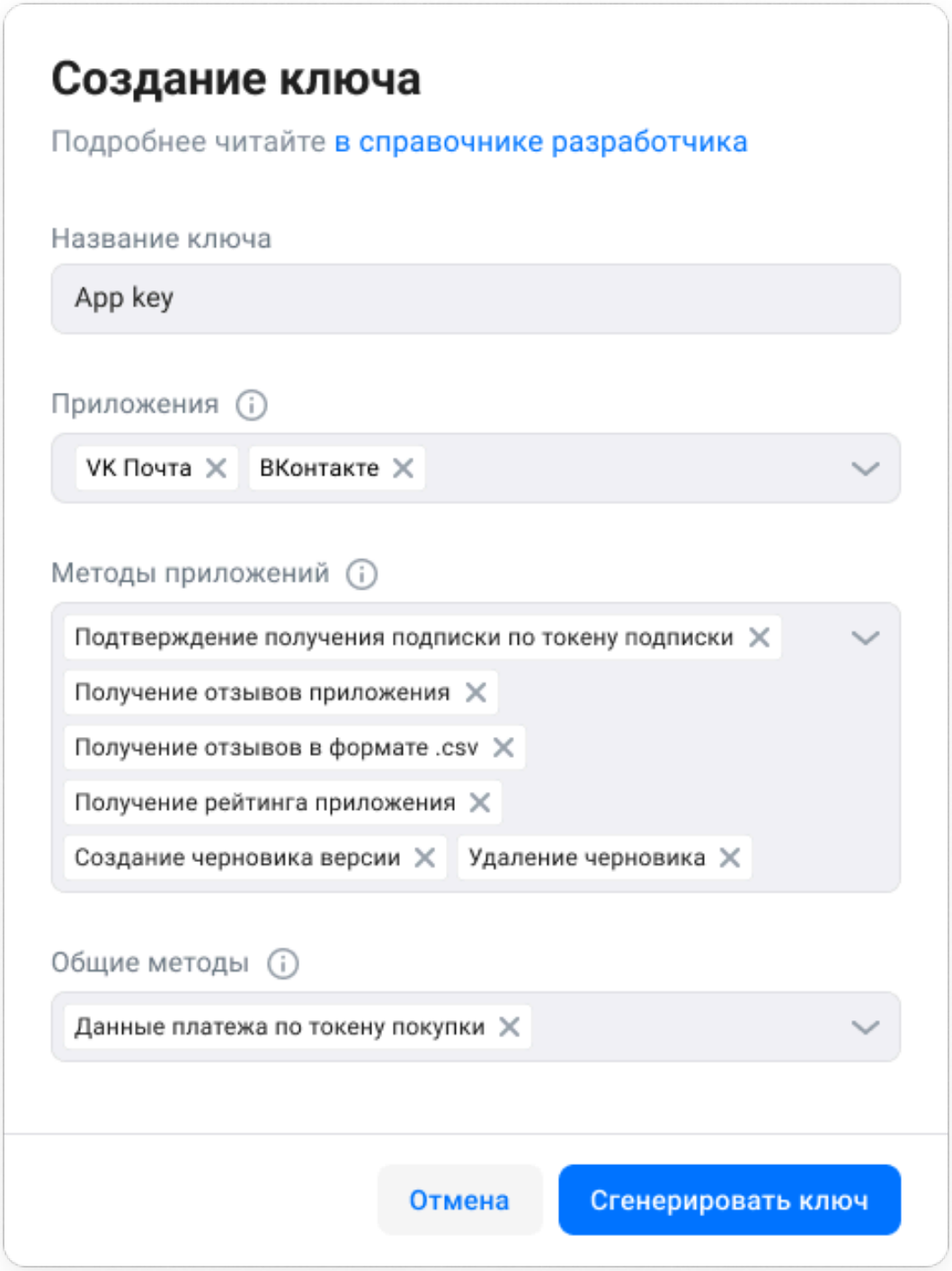
您可以选择一个或多个应用程序，也可以选择所有应用程序。如果在生成密钥时选中“所有应用程序”旁边的复选框，然后创建新应用程序，则该密钥将不会应用于该应用程序。



7. 选择一个或多个RuStore API方法。您可以选择私钥将访问的应用程序方法，也可以选择通用方法。通用方法将提供所有应用程序的数据，无论密钥设置适用于哪些应用程序。


Методы приложений
Выберите метод
Получение данных платежа и подписки
Получение данных подписки по токену подписки (V2)
Подтверждение получения подписки по токену подписки
Работа с отзывами
Получение отзывов приложения
Получение отзывов в формате CSV
Общие методы
Выберите метод
Получение данных платежа и подписки
Данные платежа по токену покупки
Данные подписки по токену подписки
Получение статуса подписки по токену подписки

8. 点击“生成密钥”。



9. 私钥将显示在“私钥”弹窗中，复制并保存它。

公钥在生成私钥时会自动形成，并独立于用户写入数据库。只需要使用生成的私钥值进行授权。

只有公司所有者可以生成密钥对。用户角色

当有密钥对时，创建的私钥将出现在RuStore 控制台系统的表格中。表格中显示了密钥的名称和ID、选择的方法和应用程序，以及密钥的修改时间（创建或更新日期）。

1. 打开RuStore 控制台系统。
2. 在屏幕顶部转到“公司”或“开发者”选项卡。
3. 在密钥名称区域找到所需的密钥。
4. 点击三个点。
5. 点击“更新”。
6. 公钥将自动更新。

您也可以在RuStore 控制台系统中删除密钥对。

1. 打开RuStore 控制台系统。
2. 在屏幕顶部转到“公司”或“开发者”选项卡。
3. 在密钥名称区域找到所需的密钥。
4. 点击三个点。
5. 点击“删除”。

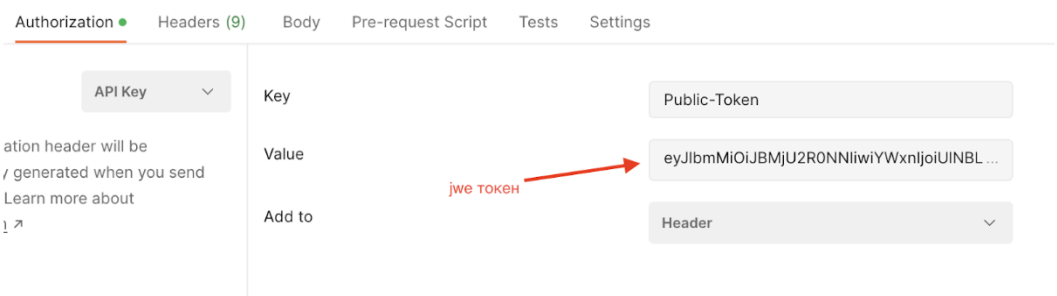
获取令牌

使用生成的私钥通过POST /public/auth/方法获取JWE令牌。

JWE令牌的有效期为900秒，之后必须通过重复POST /public/auth/方法再次获取它。

授权成功

获取的JWE令牌允许使用RuStore API的功能。为此，必须使用 API 密钥将其值传输到“授权”部分。



授权令牌获取方法

该方法允许使用在RuStore 控制台系统中获取的私钥生成JWE令牌。此外，该方法还对应用程序所有者的活动进行检查。

要成功执行请求，请生成私钥。

交互参数

POST /public/auth/

属性	类型	必填	布局	描述	填写示例
companyId	string	是，如果没有指定keyId	body	ID公司	123
keyId	string	是，如果没有指定companyId	body	ID钥	123
timestamp	string	是	body	发送请求的时间。与当前服务器时间的差异不应超过60秒。	2022-07-08T13:24:41.8328711+03:00
signature	string	是	body	RSA签名的SHA-512 hash，由包含companyId或keyId和timestamp的字符串生成。创建和检验签名的算法为SHA512withRSA。例如：companyId: 123, timestamp: 2022-07-08T13:24:41.8328711+03:00，用于计算hash的字符串	

属性	类型	必填	布局	描述	填写示例
				hash:1232022-07-08T13:24:41.8328711+03:00SHA-512hash:0976c61cce96fccd9daaae5f594db43dd287c0e266561669184276a2e86578c0e2a39cd0b183a458d0e47b17c68548daac83db97bc710dcd07d01bae40033235	

如果您创建了多于一个密钥，则需要使用keyId参数而不是companyId进行授权，并生成signature参数。

如果您在理解生成signature参数的算法上遇到问题，请使用以下.jar文件来生成该参数：

- 对于companyId的.jar文件；
- 对于keyId的.jar文件。

传递公司ID或密钥ID以及获得的私钥。作为响应，您将获得用于POST /public/auth/请求的请求体，其中包含companyId/keyId、timestamp和signature参数。

生成的signature参数的生成参数有效期为1分钟，因为它直接依赖于timestamp。

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	错误情况下的响应代码解释	Range timestamp not valid
body{}	object	是		

属性	类型	必须	描述	填写示例
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

body{}

属性	**类型**	**必须**	**描述**	**示例**
jwe	string	是	API RuStore 访问令牌（payload内容如下）。可多次使用。	eyJjdHkiOiJK…sv16aBl8tTg.VkWuTw
ttl	int	是	令牌的生命周期（以秒为单位）。生命周期为900秒。	900

可能的错误

code	**message**	**描述**	**可行操作**
400	Range timestamp not valid	timestamp参数相差超过60秒	更新timestamp和signature
404	Company key not found	按传递的companyId未找到私钥companyId	检查此companyId是否已创建私钥及其有效性
400	Company key disabled	传递的companyId的私钥已删除	检查此companyId是否已创建私钥及其有效性
400	Signature encode error	signature参数生成不正确	重新生成signature参数signature
404	You cannot use this action because the company is not found	传递的companyId对应的公司不存在或未激活	请检查companyId参数的有效性
400	You can’t use this action because the company is banned	传递的companyId对应的公司已被封锁	请检查companyId参数的有效性
400	KeyId or companyId must be not null	必须使用keyId或companyId中的一个参数	请确认已指定keyId或companyId参数
400	Incorrect usage of companyId. Please use keyId	您创建了多于一个私钥，需要使用keyId参数	请使用keyId参数

请求示例

curl –location ‘https://public-api.rustore.ru/public/auth’
–header ‘Content-Type: application/json’
–data ’{“companyId”:“1275328”,“timestamp”:“2023-08-11T13:31:17.580+03:00”,“signature”:“U4kh…….nFkbuw==”}

成功响应示例

{ “code”: “OK”, “message”: null, “body”: { “jwe”: “eyJlbmMiOiJBM……nuuM227D_O1A”, “ttl”: 900 }, “timestamp”: “2023-08-11T13:31:33.171847393+03:00”}

使用 API 获取支付和订阅数据

通过购买令牌获取支付数据

此方法允许通过购买令牌获取支付信息。

交互参数

GEThttps://public-api.rustore.ru/public/purchase/{purchaseToken}

属性	类型	描述	必须	布局	填写示例
Public-Token	string	Public API Rustore的访问令牌	是	header	
purchaseToken	string	如何获取购买令牌	是	path	111.123

响应参数

属性	类型	描述	必须	布局	填写示例
code	string	响应代码	是	body	OK,ERROR,BAD_REQUEST,NOT_FOUND
message	string	响应代码解释	否	body	
timestamp	string	响应时间	是	body	
body{}	object	响应正文	否	body	

body{}

属性	类型	描述	填写示例
error {}	object	错误解释	
invoice_id	string	发票号码	12345
invoice_date	string	发票创建日期	2020-04-29T08:18:03+03

invoice_status	string	发票状态	created — 已创建; executed — 用户选择了支付方式，支付正在执行; cancelled — 用户取消; paid — 资金已预留，发票等待确认（只适用于消耗性产品的购买）; confirmed — 支付成功; refunded — 资金已退还给购买者。
invoice {}	object	发票信息	
image	string	图片链接	https://i-love-png.com/images/grim-reaper-icon.png
application_code	string	应用程序代码	com.MashaAndTheBear.HairSalon
application_name	string	应用程序名称	《玛莎和熊》美容院
owner_code	string	应用程序所有者代码	com.MashaAndTheBear
owner_name	string	应用程序所有者名称	玛莎和熊
payment_info {}	object	支付信息	
payment_methods{}	object	支付工具	

body.error {}

属性	类型	描述	填写示例
user_message	string	错误文本	
error_description	object	错误描述	
error_code	number	错误的数字代码	0

body.invoice {}

属性	类型	描述	填写示例
purchaser{}	object	关于买方的信息	
delivery_info{}	object	配送信息	
invoice_params[]	array	订单的额外参数	
order{}	object	关于订单的信息	

body.invoice.purchaser {}

属性	类型	描述	填写示例
email	string	Email买方	qq@dd.eof
phone	string	电话号码	9123456789
contact	string	首选的通信方式	email

body.invoice.`delivery_info` {}

属性	类型	描述	填写示例
address{}	object	地址	
delivery_type	string	配送方式	courier
description	string	附加信息	在1.5小时之前回电话

body.invoice.`delivery_info.address` {}

属性	类型	描述	填写示例
country	string	国家标识符	RU
city	string	城市	莫斯科
address	string	地址	瓦维洛娃街19号，办公室1

body.invoice.invoice_params []

属性	类型	描述	填写示例
key	string	参数名称	packageName
value	string	参数值	com.MashaAndTheBear.HairSalon

body.invoice.order {}

属性	类型	描述	填写示例
order_id	string	订单的唯一标识符	d290f1ee-6c54-4b01-90e6-d701748f0851
order_number	string	订单号	145
order_date	string	订单日期	2020-04-29T08:17:03+03
service_id	string	服务标识符	223
amount	number	以货币最小单位（戈比）表示的订单金额	11836
currency	string	货币代码	RUB
purpose	string	付款的简要目的	在游戏“玛莎和熊，美容院魔女”中的购买

description	string	订单描述	在游戏“玛莎和熊，美容院魔女”中购买游戏内内容
language	string	语言	ru-RU
expiration_date	string	发票有效期的日期和时间	2022-10-11T14:05:44.741Z
tax_system	number	税收制度	0
trade_name	string	组织的商业名称	Romashka
visual_name	string	操作名称	购买/续订订阅
org_name	string	组织名称	有限责任公司″甘菊″
org_inn	string	组织的纳税人识别号	1234567890
visual_amount	string	金额	1 500,45 ₽
order_bundle []	array	订单组成	

body.invoice.order.order_bundle []

属性	类型	描述	填写示例
position_id	number	唯一产品标识符	1
name	string	产品名称或描述	水晶
item_params[]	array	订单的额外参数	
quantity{}	object	产品总数量的描述	
item_amount	number	所有产品项的价值总额，以货币的最小单位（即戈比）表示	11836
currency	string	货币代码	RUB
item_code	string	产品编号（标识符）	com.MashaAndTheBear.HairSalon.crystal100
item_price	number	单个产品的价格，以货币的最小单位表示（即戈比）	11836
discount_type	string	产品折扣类型	percent
discount_value	float	产品折扣值	5.25
interest_type	string	销售产品的代理佣金类型	agentPercent
interest_value	float	销售产品的代理佣金值	15.105
tax_type	number	增值税税率	6

tax_sum	number	税款总额，以货币的最小单位表示（即戈比）	2367
image	string	图片链接	https://i-love-png.com/images/grim-reaper-icon.png

body.invoice.order.order_bundle.item_params []

属性	类型	描述	填写示例
key	string	参数名称	packageName
value	string	参数值	com.MashaAndTheBear.HairSalon

body.invoice.order.order_bundle.quantity {}

属性	类型	描述	填写示例
value	float	含义	1.05
measure	string	计量单位	公斤

body.payment_info {}

属性	类型	描述	填写示例
payment_date	number	发票支付日期和时间	2022-10-11T14:05:44.741Z
payment_id	string	唯一支付标识符	d290f1ee-6c54-4b01-90e6-d701748f0851
payment_params{}	object	支付的附加参数	
device_info{}	object	设备信息	
loyalty_info{}	object	忠诚计划信息	
card_id	string	银行卡的唯一标识符	ad454ffg-6c54-4b01-90e6-d701748f0851
name	string	卡别名	重要
paysys_code	string		RBS-shortname
masked_pan	string	银行卡掩码号	**1111
expiry_date	string	卡到期日期	201912
cardholder	string	卡持有人姓名	Ivan Petrov
payment_system	string	支付系统	Visa
payment_system_image	string	支付系统标志链接	https://smartmarkettestift.online.sberbank.ru/icons/logo_visa.png

image	string	卡标志链接	https://smartmarkettestift.online.sberbank.ru/icons/sberbank_mastercard_league_legends.jpeg
paysys	string	支付运营商名称	RBS
paysys_image	string	支付运营商标志链接	https://www.sberbank.ru/common/img/uploaded/redirected/s_m_business/acquiring/assets/images/intro@2x.png
payment_way	string	支付方式	SberPay
payment_way_code	string	支付方式标识符	SberPay
payment_way_logo	string	支付方式标志链接	https://cdn1.telegram.one/i/f7640dada78306b1c993e04001b8738d/828b1eb30921659e22e53a9edc92c4c4/24e01830d213d75deb99c22b9cd91ddd
bank_info{}	object	银行信息	

body.payment_info.payment_params {}

属性	类型	描述	填写示例
key	string	参数名称	googlePurchaseToken
value	string	参数值	ameinkbophchljaejnocadib

body.payment_info. `device_info` {}

属性	类型	描述	填写示例
device_platform_type	string	设备平台	iOS
device_platform_version	string	平台操作系统版本	13.6.1
device_model	string	设备型号	iPhone7
device_manufacturer	string	制造商	Apple
device_id	string	设备序列号	83c3f257-46d8-41fe-951b-f79d04e288c2
surface	string		RuStore

surface_version	string	软件版本	11.5.0

body.payment_info.loyalty_info {}

属性	类型	描述	填写示例
service_code	string	奖励计划代码	sbrf_spasibo
service_name	string	奖励计划名称	Sberbank 谢谢
change_rate	number	积分兑换卢布的系数	1
payment_bonus	number	使用积分支付发票的金额（以戈比计算）	19800
award_bonus	number	用于积分计算的资金总额，在支付订单时指明以最小单位（即戈比）	21850
image	string	图片链接	https://i-love-png.com/images/grim-reaper-icon.png

body.payment_info.bank_info{}

属性	类型	描述	填写示例
bank_name	string	发行银行名称	俄罗斯联邦储蓄银行股份公司
bank_country_code	string	发行银行的国家代码	RU
bank_country_name	string	发行银行的国家名称	俄罗斯
bank_image	string	银行标志链接	https://emoji.slack-edge.com/TKK9DHNCV/sber/ad2df81a6cd9812d.png

body. payment_methods {}

属性	类型	描述	填写示例
user_message	string	用户的消息（非必需）	为了连接订阅，请在移动应用程序中保存您的银行卡信息
methods[]	array	支付选项	

body. payment_methods.methods []

属性	类型	描述	填写示例
method	string	代码	QR

action string 名称 通过QR码支付

示例回复：

{
 "code": "OK",
 "message": null,
 "body": {
"invoice_id": "2850",
 "invoice_date": "2023-07-18T14:31:33+03",
"invoice_status": "confirmed",
 "application_code":
"3399750",
 "application_name": "玛莎和熊",
 "owner_code":
"4384191",
 "owner_name": "有限责任公司 \"娜拉娜\"",
"payment_info": {
 "payment_date": "2023-07-18T14:31:42+03",
"payment_id": "1736",
 "payment_params": null,
"loyalty_info": null,
 "card_id": "193",
"paysys_code": "RBS-shortname",
 "masked_pan": "XX1111",
"expiry_date": "202412",
 "payment_system": "Visa",
"payment_system_image":
"https://smartmarket.online.sberbank.ru/image/visa.png",
"paysys_image": null,
 "payment_way": "使用已保存的银行卡付款
",
 "payment_way_code": "CARD_BINDING",
"payment_way_logo":
"https://static.tildacdn.com/tild6236-3530-4235-b966-326630656238/___14_-removeb
g-prev.png",
 "bank_info": {
 "bank_name":
"Sberbank",
 "bank_country_code": "SU",
"bank_country_name": null,
 "bank_image": null
},
 "device_info": null,
 "name": null,
"cardholder": "CARDHOLDER NAME",
 "image": null,
"paysys": "RBS"
 },
 "payment_methods": null,
"error": {
 "user_message": "发票已确认",
"error_description": "",
 "error_code": 0
 },
"invoice": {
 "delivery_info": {
"delivery_type": null,
 "address": {
"country": null,
 "city": null,
"address": null
 },
 "description":
null
 },
 "invoice_params": [
{
 "key": "inapp_serviceparam_message_about_loyalty",
"value": "现在可以使用奖励积分支付订阅，但续订订阅只能用卢布。
"
 },
 {
 "key":
"inapp_serviceparam_action_name",
 "value": "链接订阅
"
 },
 {
 "key":
"inapp_serviceparam_features",
 "value": "VERIFY"
},
 {
 "key":
"period_type",
 "value": "DAY"
 },
{
 "key": "period_duration",
"value": "1"
 },
 {
"key": "current_period",
 "value": "STANDARD"
},
 {
 "key":
"payment_type",
 "value": "INITIAL"
}
 ],
 "purchaser": {
 "email":
null,
 "phone": null,
 "contact": null
},
 "order": {
 "order_id":
"a090a93c-ca06-493d-a90a-ce2bac722358",
 "order_number":
"311",
 "order_date": "2023-07-18T14:31:33+03",
"service_id": "4720",
 "expiration_date":

```
"2023-07-18T14:51:33+03",<br>                "tax_system": null,<br>
    "trade_name": null,<br>                "visual_name": "购买/续订订阅",<br>
          "org_name": "有限责任公司 超级 有限责任公司",<br>
"org_inn": "4419198349",<br>                "visual_amount": "1 卢布",<br>
         "order_bundle": [<br>                      {<br>
"position_id": 1,<br>                        "item_params": [<br>
           {<br>                                "key":
"_auto_itemAttributes_agent_info.type",<br>
"value": "7"<br>                                },<br>
{<br>                              "key":
"_auto_itemAttributes_supplier_info.name",<br>
"value": "有限责任公司 \"娜拉娜\""<br>                                    },<br>
                  {<br>                              "key":
"_auto_itemAttributes_supplier_info.inn",<br>
"value": "4419198349"<br>                            }<br>
  ],<br>                        "item_amount": 100,<br>
"item_code": "1day",<br>                          "item_price": 100,<br>
             "discount_type": null,<br>
"discount_value": null,<br>                          "interest_type": null,<br>
                   "interest_value": null,<br>
"tax_type": 6,<br>                          "tax_sum": null,<br>
      "name": "订阅购买的支付 \"1day\".提供商: \"购买/续订订阅\"",<br>
           "quantity": {<br>                              "value": 1,<br>
                  "measure": "个"<br>                        },<br>
           "currency": "RUB",<br>                          "image": ""<br>
           }<br>                ],<br>                "amount": 100,<br>
        "currency": "RUB",<br>                "purpose": "1day",<br>
    "description": "1day",<br>                "language": "ru-RU"<br>
}<br>        },<br>        "image": ""<br>    },<br>    "timestamp":
"2023-08-02T10:11:04.655684723+03:00"<br>}
```


请求验证

在接收请求时，将检查通过获取授权令牌方法获得的授权令牌：

- 所有者和/或应用程序不应被封锁；
- 令牌应该是有效的；
- 应用程序代码 application_code 应该与获得授权令牌的所有者相对应。

通过订阅令牌获取订阅数据

此方法允许通过订阅令牌获取订阅信息。

交互参数

GEThttps://public-api.rustore.ru/public/subscription/{subscriptionToken}

属性	类型	描述	必须	布局	填写示例
Public-Token	string	Public API Rustore的访问令牌	是	header	

属性	类型	描述	必须	布局	填写示例
subscriptionToken	string	如何获取订阅令牌	是	path	111.123

响应参数

属性	类型	描述	必须	布局	填写示例
code	string	响应代码	是	body	OK, ERROR, BAD_REQUEST, NOT_FOUND
message	string	响应代码解释	否	body	
timestamp	timestamptz	响应时间	是	body	
body{}	object	响应正文	否	body	

body{}

属性	类型	描述	必须	布局	填写示例
code	number	响应代码	是	body	
success	boolean	响应成功标志	是	body	true
message	string	响应代码解释	否	body	未知错误
body{}	object	消息正文	否	body	

body.body{}

属性	类型	描述	填写示例
serviceName	string	服务名称	Okko
subscriptionId	integer	订阅的唯一标识符	12345
addParameters	string	订阅的附加参数	Something about subscription
productType	string	产品类型	SUBSCRIPTION
productName	string	产品名称	Okko优选
productCode	string	产品代码	monthly_sub
recurrent	boolean	自动续订标志	true
countOfDay	integer	天数	10
periodType	string	周期类型	可用值：DAY, MONTH, YEAR
periodDuration	integer	周期时长	10
nextPaymentDate	string	下次支付日期	2021-03-23
price	integer	以货币最小单位（即戈比）表示的价格	9999

属性	类型	描述	填写示例
currency	string	货币	RUB
imageUrl	string	产品图片链接	
state	string	订阅状态	**活跃状态**。ACTIVATED — 订阅已激活。**中间状态**。ACCEPTED — 等待支付。DEPOSITED — 支付已成功，订阅等待激活；CLOSE_PENDING — 订阅正在关闭中；REPEATING — 订阅处于续订阶段，正在尝试续订支付。**最终状态**。DECLINED — 订阅已关闭，所有尝试进行订阅支付均失败。GRACE 和 HOLD 期限以及续订尝试次数已结束。CANCELED — 用户未支付订阅费；CLOSED — 订阅已关闭；用户已关闭自动续订功能，已到期的订阅期满，且提供商已确认订阅关闭；REFUNDED — 启动向用户退款。
currentPeriod	string	当前期间的名称	**可用值**：PROMO — 免费期间；START — 初始期间，享有降价优惠；STANDARD — 标准期间；GRACE — 宽限期，用户未支付订阅费，扣费尝试仍在继续，内容访问权限保留；HOLD — 暂停期，用户未支付订阅费，扣费尝试仍在继续，但应停止访问内容。

属性	类型	描述	填写示例
debtPaymentPeriod	string	支付周期的名称	Null如果周期是标准的
description	string	订阅产品描述	描述
tariffId	integer	费率的唯一标识符	12345
periods []	array[object]	当前周期信息	费率周期

body.body.periods []

属性	类型	描述	填写示例
periodName	string	当前期间的名称	可用的值包括：PROMO, START, STANDARD, GRACE, HOLD
periodType	string	周期类型	可用的值包括：DAY, MONTH, YEAR
periodDuration	number	周期时长	10
periodPrice	number	以货币最小单位（即戈比）表示的价格	10000
nextPeriod	string	下一个周期的名称	可用的值包括：PROMO, START, STANDARD, GRACE, HOLD

示例回复：


```
"code":"200","message":"something","timestamp":"2023-08-02T10:11:04.655684723+03
:00","body": {    ``"code" :  40401 ,    ``"success" :  false ,    ``"message"
:  "未知错误" ,    ``"body" : {      ``"serviceName" :  "Okko" ,
 ``"subscriptionId" :  100500 ,      ``"addParameters" :  "something" ,
 ``"productType" :  "string" ,      ``"productName" :  "OKKO 优选" ,
 ``"productCode" :  "string" ,      ``"recurrent" :  true ,      ``"countOfDay"
:  100 ,      ``"periodType" :  "DAY" ,      ``"periodDuration" :  30 ,
 ``"nextPaymentDate" :  "2021-03-23" ,      ``"price" :  999 ,      ``"currency"
:  "RUB" ,      ``"imageUrl" :
"https://static-eu.insales.ru/images/products/1/7435/306650379/thumb_15865248178
49_158324636645650539901068 68.jpg" ,      ``"state" :  "ACTIVATED" ,
 ``"currentPeriod" :  "STANDARD" ,      ``"debtPaymentPeriod" :  "string" ,
 ``"description" :  "描述" ,      ``"tariffId" :  100500 ,      ``"periods" :
[        ``{          ``"periodName" :  "STANDARD" ,          ``"periodType" :
"DAY" ,          ``"periodDuration" : 10 ,          ``"periodPrice" : 10000
,          ``"nextPeriod" :  "STANDARD"``}``]}}
```


请求验证

在接收请求时，会检查通过授权令牌获取方法获得的授权令牌。

- 所有者和/或应用程序不应被封锁；

- 令牌应该是有效的；
- 应用程序代码 application_code 应该与获得授权令牌的所有者相对应。

通过订阅令牌获取订阅数据（V2）

此方法允许通过订阅令牌获取订阅信息。

交互参数

GEThttps://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{subscriptionToken}

属性	类型	描述	必须	布局	填写示例
Public-Token	string	Public API Rustore的访问令牌	是	header	
packageName	string	应用程序包的名称	是	path	
subscriptionId	string	产品订阅代码	是	path	
subscriptionToken	string	如何获取订阅令牌	是	path	111.123

成功响应的参数

属性	类型	描述	必须	填写示例
startTimeMillis	string	订阅开始时间，以自纪元开始的毫秒计。	是	1577826955637
expiryTimeMillis	string	订阅到期时间，以自纪元开始的毫秒计。	是	1609456386128
autoRenewing	boolean	订阅是否会在当前期限结束后自动续订。	是	false
priceCurrencyCode	string	订阅价格的ISO 4217货币代码。	是	RUB
priceAmountMicros	string	订阅价格以微单位表示，其中1,000,000微单位等于一单位货币。例如，如果订阅价格为1.99欧元，则	是	749000000

属性	类型	描述	必须	填写示例
		AmountMicros为1,990,000。		
countryCode	string	用户发票国家/地区代码，在订阅提供时刻。	是	RU
paymentState	int	订阅支付状态。可能的值包括：0.等待支付 1.支付接收 2.免费试用版 3.等待延迟的升级/降级。已取消且过期的订阅不适用。	否，只适用于活跃的订阅	1
cancelReason	int	订阅取消或不自动续订的原因。可能的值包括：0.用户取消订阅 1.系统取消订阅，如支付问题 2.订阅被新订阅替换 3.开发者取消订阅	否，仅适用于状态为CLOSED的已取消订阅	0
orderId	string	首次支付发票的标识符，与订阅购买相关。如果订阅有多个发票，标识符后会加上数量，以“..”分隔。	是	41456..3
acknowledgementState	int	订阅产品确认状态。可能的值包括：0.还未确认 1.已确认	是	1
introductoryPriceInfo{}	object	关于订阅的促销期信息。此字段并不表明订阅当前处于促销期。	否，只适用于PROMO期间的订阅。	

属性	类型	描述	必须	填写示例
kind	string	始终传递值“androidpublisher#subscriptionPurchase”	是	androidpublisher#subscriptionPurchase
purchaseType	int	始终传递值“0”	是	0

introductoryPriceInfo

属性	类型	描述	必须	填写示例
introductoryPriceCurrencyCode	string	ISO 4217货币代码用于订阅的初始价格	否，只适用于PROMO期间的订阅。	RUB
introductoryPriceAmountMicros	string	订阅的初始价格。货币与priceCurrencyCode匹配。以微单位表示，其中1,000,000微单位等于一单位货币。例如，如果订阅价格为1.99欧元，则AmountMicros为1,990,000。	否，只适用于PROMO期间的订阅。	599000000
introductoryPricePeriod	string	以ISO 8601格式指定的初始价格期限。例如，“P1W”（一周）、“P1M”（一个月）、“P3M”（三个月）、“P6M”（六个月）和“P1Y”（一年）。	否，只适用于PROMO期间的订阅。	P1Y
introductoryPriceCycles	string	计算初始价格提议的周期数。	否，只适用于PROMO期间的订阅。	1

JSON响应示例

{“startTimeMillis”: “1694431707000”,“expiryTimeMillis”: “1697034507000”,“autoRenewing”: true,“priceCurrencyCode”: “RUB”,“priceAmountMicros”: “749000000”,“countryCode”: “RU”,“paymentState”: 1,“orderId”: “33252..1”,“acknowledgementState”: 1,“kind”: “androidpublisher#subscriptionPurchase”,“purchaseType”: 0,“introductoryPriceInfo”: { “introductoryPriceCurrencyCode”: “RUB”, “introductoryPriceAmountMicros”: “599000000”, “introductoryPricePeriod”: “P1M”, “introductoryPriceCycles”: “1” }}

错误响应的参数

属性	类型	描述	必须	填写示例
error{}	object	错误数据对象	是，如果响应代码!=200	

error

属性	类型	描述	必须	填写示例
code	number	http响应代码	是	
message	string	响应代码解释	否	

message

code	message
400	The subscription purchase token does not match the subscription ID.
404	No subscription purchase matches the subscription ID.

通过订阅令牌获取订阅状态

此方法允许通过订阅令牌获取订阅状态信息。

交互参数

GET https://public-api.rustore.ru/public/subscription/{subscriptionToken}/state

属性	类型	描述	必须	布局	填写示例
Public-Token	string	Public API Rustore的访问令牌	是	header	
subscriptionToken	string	如何获取订阅令牌	是	path	111.123

响应参数

属性	类型	描述	必须	布局	填写示例
code	string	响应代码	是	body	ＯＫ,ERROR,BAD_REQUEST,NOT_FOUND
message	string	响应代码解释	否	body	
timestamp	timestamptz	请求时间	是	body	
body{}	object	响应正文	否	body	

body{}

属性	类型	描述	必须	布局	填写示例
is_active	boolean	订阅活动标志。- true — 订阅活跃，在以下周期之一：PROMO, START, STANDARD, GRACE- false — 订阅不活跃：处于HOLD周期或已结束	是	body.body{}	true

示例回复：

```
"code":"200","message":"something","timestamp":"2023-08-02T10:11:04.655684723+03:00","body":{"is_active": "true"}
```

请求验证

在接收请求时，会检查通过授权令牌获取方法获得的授权令牌。

- 所有者和/或应用程序不应被封锁；
- 令牌应该是有效的；
- 订阅是在其所有者为获得Public API访问令牌的公司的应用程序中购买的。

通过订阅令牌确认收到订阅

此方法允许通过订阅令牌确认用户内容的发放。

确认收到订阅对于订阅的工作不是必需的。

交互参数

POST
https://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{purchaseToken}:acknowledge

属性	类型	描述	必须	布局	填写示例
Public-Token	string	Public API Rustore的访问令牌	是	header	
packageName	string	应用程序包的名称	是	path	
subscriptionId	string	产品订阅代码	是	path	
purchaseToken	string	如何获取订阅令牌	是	path	

成功响应的参数

如果成功响应，response body为空。

请求验证

在接收请求时，会检查通过授权令牌获取方法获得的授权令牌。

- 所有者和/或应用程序不应被封锁；
- 令牌应该是有效的；
- 订阅是在其所有者为获得Public API访问令牌的公司的应用程序中购买的。

使用API RuStore上传和发布应用程序

1. 在发布应用程序版本之前，请执行以下操作：
    1. 使用“创建版本草稿”方法创建带有应用信息的版本草稿；
    2. 上传APK — 使用“上传APK文件”方法；
    3. 上传图片 — 使用“上传应用图标”和“上传截图”方法。
2. 草稿无法编辑，但可以使用“删除草稿”方法删除。
3. 在填写完所有信息后，可以使用“发送草稿版本至审核”方法将版本提交审核。
4. 使用“获取应用版本状态”方法，可以获取应用版本列表、其基本信息和状态。
5. 可以通过“更改发布设置”方法更改发布设置（发布类型、延迟发布日期以及部分发布的百分比）。
6. 审核通过后，如果选择的是手动发布，可以使用“手动发布”方法发布版本。

请注意，要使用RuStore API，必须在RuStore控制台至少有一个活跃的应用版本。

创建应用程序版本草稿

此方法允许创建应用程序版本草稿并填写其主要信息。

限制：

- 对于一个应用程序只能创建一个草稿。尝试创建多个草稿将导致错误；
- 通过API发送的草稿在开发者控制台的网页版本中不会显示。它们将在发送APK进行审核并且应用版本状态改变后变得可用；
- 只有在应用程序有一个活跃版本的情况下，才能创建草稿；
- 草稿中缺失的数据将从应用程序的活跃版本中获取；
- 草稿必须符合应用类型 — 免费或付费。如果为免费应用发送付费版本，草稿将被视为免费；
- 每种应用版本类型（appType）都有相应的版本类别。如果类别不匹配，在尝试上传应用时将显示错误；
- 应用的部分发布只能在自动（INSTANTLY）或手动（MANUAL）发布类型（参数 — publishType）下进行。

交互参数

POST https://public-api.rustore.ru/public/v1/application/{packageName}/version

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	Public API Rustore的访问令牌	
packageName	string	是	path	应用程序包的名称	com.myapp.example
appName	string	否	body	应用版本名称：最大长度为50个字符	我的应用程序
appType	string	否	body	应用版本类型：可能的值为：GAMES — 对于游戏；MAIN — 对于非游戏应用程序。提交的类别必须在同一请求中是唯一的	GAMES

属性	类型	必须	布局	描述	填写示例
categories	string	否	body	版本类别：最多两个类别	“health”, “news”
ageLegal	string	否	body	年龄分类：可能的选项包括：″18+″、″16+″、″12+″、″7+″、″3+″	7+
shortDescription	string	否	body	版本简述：最大长度为80个字符	
fullDescription	string	否	body	版本完整描述：最大长度为4000个字符	
whatsNew	string	否	body	描述“新功能”最大长度 - 500个字符	
moderInfo	string	否	body	开发者对审核员的评论。最大长度 - 180个字符	
priceValue	string	否	body	应用程序的价格，以货币的最小单位（即戈比）计算，例如，“87.99卢布” = 8799，数值必须>0	8799
publishType(NEW)	string	否	body	**发布类型**。可能的值包括：- MANUAL — 手动发布；- INSTANTLY — 自动发布，审核通过后立即发布；- DELAYED — 延迟发布。备注：如果此参数未指定，默	MANUAL

属性	类型	必须	布局	描述	填写示例
				认为 INSTANTLY。	
publishDateTime(NEW)	timestamptz	否 是，如果 publishType = DELAYED.	body	延迟发布的日期和时间：格式：yyyy-MM-dd’T’HH:mm:ssXXX。指定的日期不得早于提交审核的计划日期24小时且不晚于60天。之后可以更改延迟发布的日期。备注：如果 publishType 指定为 publishType 或 MANUAL，该参数可以为任意值且不会被考虑。	2022-07-08T13:24:41.8328711+03:00
partialValue(NEW)	number	否	body	应用程序部分发布的百分比。可能的值包括：- 5% - 10% - 25% - 50% - 75% - 100%	5

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
content{}	object	是		

content

属性	类型	必须	描述	示例
versionId	number	是	应用程序版本	243242

示例

```
curl-location-requestPOST’https://public-api.rustore.ru/public/v1/application/com.package.com/version’<br>-header’Content-Type:application/json’<br>-header’Public-Token:{YOURtoken}‘<br>-data-raw’{“appName”:“适用于RuStore的应用程序”,
“appType”:“MAIN”,“categories”:[“news”,“education”],“ageLegal”:“7+”,“shortDescription”:“适用于RuStore的应用程序”,“fullDescription”:“fullDescription-适用于RuStore的应用程序”,“whatsNew”:“whatsNew-适用于RuStore的应用程序”,
“moderInfo”:“moderInfo-适用于RuStore的应用程序”,“priceValue”:1100}’
```

示例回复

```
{    “code”: “OK”,    “message”: null,    “body”: 243242,    “timestamp”: “2023-07-27T10:28:59.039649+03:00”}
```

上传APK文件

此方法允许上传.apk扩展名的文件以便发布应用程序。

APK文件的版本可以更新。

限制：

- 仅限扩展.apk的文件；
- 文件大小不超过2.5 GB；
- 上传的APK版本必须高于当前活跃版本。

可以同时上传两个APK文件：

- 具有不同开发者签名 - 以排除用户更新错误。在RuStore的展示页面下载时，将为每个用户单独提供具有相同签名的文件。
- 支持不同服务（Huawei Mobile Services 和 Google Mobile Services）- 以便RuStore能够根据用户设备提供下载含有适配服务的APK文件。
- 在上传多个文件时，必须指明哪个文件包含Huawei Mobile Services，以及哪个文件将作为默认对所有用户可用。最多可上传10个文件。

我们提醒您注意，在7月20日的更新中，一个名为IsMainApk的必填参数已被添加到下面的方法中。同时，我们请求您更新您的pipelines。

交互参数

POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/apk

属性	类型	必须	布局	描述
Public-Token	string	是	header	Public API Rustore的访问令牌
packageName	string	是	path	应用程序的包名
versionId	number	是	path	应用程序版本
servicesType(NEW)	string	否	query	应用中使用的服务类型。可能的选项包括：- HMS — 用于含有Huawei Mobile Services的APK文件，- Unknown — 如果字段未填写，则默认设置
isMainApk (NEW)	boolean	是	query	指定给主apk文件的标志。值为：- true — 主要的APK文件- false — 默认值

form data

属性	类型	必须	布局	描述
file	multipart	是		以二进制代码形式的文件

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	错误详细信息。可能的情况包括： - 已上传最大数量的apk文件 / Maximum number of apk

files is already uploaded - 已上传主apk文件 / Main apk file is already uploaded - 已上传含有HMS的apk文件 / Apk file with the Huawei Mobile Services is already uploaded - 含Huawei Mobile Services的文件不能作为主文件 / Apk file with the Huawei Mobile Services can not be main file - 您上传的apk文件与其他版本代码不同 / Apk file has different version code than the one previously uploaded - 版本代码必须高于之前上传的 / The code of this version must be larger than that of the previous one - 版本与前一个不匹配 / The package does not match

				the previous version - 已存在同名版本 / A package with this name already exists. Rename the package.
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

示例

```
curl -location -request POST 'https://public-api.rustore.ru/public/v1/application/com.package.example/version/123/apk?servicesType=Unknown&isMainApk=true' <br>-header 'Public-Token: {YOURtoken}' <br>-form 'file=@"/Users/User/Downloads/package.apk"'
```

上传应用程序图标

此方法允许上传应用程序图标。

限制：

- 文件仅限于.jpeg和 .png扩展；
- 文件大小不超过3MB；
- 上传图片的尺寸为512×512 px。

交互参数

POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/icon

属性	类型	必须	布局	描述
Public-Token	string	是	header	Public API Rustore的访问令牌
packageName	number	是	path	应用程序标识符
versionId	number	是	path	应用程序版本

Form data

属性	类型	必须	描述
file	multipart/form-data	是	文件

响应参数

属性	类型	必填	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

示例

```
curl-location-requestPOST’https://public-api.rustore.ru/public/v1/application/com.package.example/version/123/image/icon’<br>-header’Content-Type:application/json’<br>-header’Public-Token:{YOURtoken}‘<br>-form’file=@“/Users/User/Downloads/img.jpg”’
```

上传应用程序截图

此方法允许上传应用程序截图。

限制：

- 截图可以是垂直或水平的；
- 截图的纵横比为 16:9（垂直）和 9:16（水平）；
- 上传文件的尺寸不得小于 320px 且不得超过 3840px；
- 上传文件的格式为 .jpg或 .png；
- 文件大小不超过3MB；
- 禁止上传不同方向的截图。例如，用户上传了竖直方向（PORTRAIT）的截图，然后想上传横向（LANDSCAPE）的截图，请求将返回错误；
- 上传超过 10 张截图时，已存在的截图将变为非活动状态。例如，如果上传了两张带有“ordinal”参数 = 7 的截图，最后上传的截图将成为活动截图。

交互参数

POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/screenshot/{orientation}/{ordinal}

属性	类型	必须	布局	描述
Public-Token	string	是	header	Public API Rustore的访问令牌
packageName	number	是	path	应用程序标识符
versionId	number	是	path	应用程序版本
orientation	string	是	path	图片方向。可能的选项包括：● LANDSCAPE —

属性	类型	必须	布局	描述
				横向；● PORTRAIT — 纵向。
ordinal	number	是	path	截图的序号。可能的值为：从 0 到 9。

Form data

属性	类型	必须	描述
file	multipart/form-data	是	文件

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

示例

```
curl -location -request POST 'https://public-api.rustore.ru/public/v1/application/com.package.example/version/123/image/screenshot/landscape/1' <br>-header 'Content-Type: application/json' <br>-header 'Public-Token: {YOURtoken}' <br>-form 'file=@"/Users/User/Downloads/img.jpg"'
```

修改发布设置

此方法允许更改发布类型、延迟发布日期和版本部分发布的百分比。

限制：

- 部分发布的百分比值只能向更大的方向编辑；
- 如果 partialValue 是 100，则应用被视为完全部署；
- 可以更改 partialValue 或 publishType 和 publishDateTime（用于延迟发布）。

交互参数

POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/publish-settings

属性	**类型**	**必填**	**布局**	**描述**	**填写示例**
Public-Token	string	是	header	Public API Rustore的访问令牌	

packageName	string	是	path	应用程序的包名	
versionId	number	是	path	应用程序版本	
publishType	string	否	body	发布类型。可能的值有：- MANUAL— 手动发布。- INSTANTLY— 自动发布，审核通过后立即发布。- DELAYED— 延迟发布。	MANUAL
publishDateTime	timestamptz	否 是，如果 publishType =DELAYED	body	延迟发布的日期和时间：格式：yyyy-MM-dd’T’HH:mm:ssXXX。指定的日期不得早于提交审核的计划日期24小时且不晚于60天。之后可以更改延迟发布的日期。备注：如果 publishType 指定为 publishType 或 MANUAL，该参数可以为任意值且不会被考虑。	2022-07-08T13:24:41.8328711+03:00
partialValue	number	否	body	应用程序部分发布的百分比。可能的值包括：- 5% - 10% - 25% - 50% - 75% - 100%	5

响应参数

属性	类型	必填	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

更改部分部署的百分比的示例

```
curl -location 'https://public-api.rustore.ru/public/v1/application/com.example.pblsh_v2/version/704372/publish-settings' <br>-header 'accept: application/json' <br>-header 'Content-Type: application/json' <br>-header 'Public-Token: {YOURtoken}<br>-data '{"partialValue": 100}'
```

示例回复

```
{    "code": "OK",    "message": null,    "body": null,    "timestamp": "2023-08-14T15:35:12.701709488+03:00"}
```

获取应用程序版本状态

此方法允许获取版本的基本信息，并检查版本的状态

限制：

- 默认情况下，每个页面显示20个版本，您可以在size参数中指定，最多显示100个版本；
- 不能同时使用分页参数和按1个版本过滤的参数（要么使用分页，指定page和size参数，要么使用ids）。

交互参数

GEThttps://public-api.rustore.ru/public/v1/application/{packageName}/version?ids=704095&page=0&size=2

属性	类型	必填	布局	描述	填写示例
Public-Token	string	是	header	Public API Rustore的访问令牌	
packageName	string	是	path	应用程序的包名	com.myapp.example
ids	number	否	query	特定版本的id。在需要获取特定版本时设置。	743103

page	number	否	query	页面编号。从0开始	0
size	number	否	query	页面上的评论数量。默认为20，最多为100	100

响应参数

属性	类型	必填	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
body{}	object	是		

body{}

属性	类型	必填	描述	示例
content[]	massive	是	包含版本列表的数组	
pageNumber	number	是	当前页面编号	0
pageSize	number	是	页面大小	2
totalElements	number	是	总元素数	5
totalPages	number	是	总页数	3

content[]

属性	类型	必填	描述	示例
versionId	number	是	id版本	704372
appName	string	是	应用程序名称	测试 API
appType	string	是	应用程序类型	MAIN或GAME
versionName	string	是	版本名称	1.0
versionCode	number	是	版本代码	6
versionStatus	string	是	版本状态	可能的值 ACTIVE,PARTIAL_ACTIVE,READY_FOR_PUBLICATION,PREVIOUS_ACTIVE,ARCHIVED,REJECTED_BY_MODERATOR,TAKEN_F

				OR_MODERATION, MODERATION, AUTO_CHECK, AUTO_CHECK_FAILED, DRAFT, DELETED_DRAFT, REJECTED_BY_SECURITY;
publishType	string	是	发布类型	可能的值- MANUAL- INSTANTLY-DELAYED
publishDateTime	timestamptz	是	延迟发布时间	2023-08-04T09:36:06.431+00:00
sendDateForModer	timestamptz	是	提交审核时间	2023-08-11T12:03:06.303+00:00
partialValue	number	是	部分发布的百分比	-1 = 100%其他值对应
whatsNew	string	是	新增内容是什么？	修复了错误
priceValue	number	是	价格，如果应用程序是付费的	0
paid	boolean	是	应用程序是否付费	true/false

请求示例

```
curl -location ‘https://public-api.rustore.ru/public/v1/application/com.example.pblsh_v2/version?page=0&size=2’ <br>-header ‘accept: application/json’ <br>-header ‘Public-Token: {YOURtoken}’
```

示例回复

```
{    “code”: “OK”,    “message”: null,    “body”: {        “content”: [
          {                “versionId”: 704372,                “appName”: “测试
API”,                “appType”: “MAIN”,                “versionName”: “1.0”,                “versionCode”: 6,                “versionStatus”: “ACTIVE”,                “publishType”: “MANUAL”,                “publishDateTime”: “2023-08-14T12:34:43.925+00:00”,                “sendDateForModer”: “2023-08-11T12:03:06.303+00:00”,                “partialValue”: -1,
            “whatsNew”: “修复了错误”,
                “priceValue”: 0,                “paid”: false
```

```
},            {                “versionId”: 704197,                “appNa
me”: “PO test API”,                “appType”: “MAIN”,                “ver
sionName”: “1.0”,                “versionCode”: 1,                “versio
nStatus”: “PREVIOUS_ACTIVE”,                “publishType”: “INSTANTLY”,
            “publishDateTime”: “2023-08-04T09:36:06.431+00:00”,
       “sendDateForModer”: “2023-08-04T09:20:23.551+00:00”,
   “partialValue”: -1,                “whatsNew”: “第一个版本”,
               “priceValue”: 0,                “paid”: false
}        ],        “pageNumber”: 0,        “pageSize”: 2,        “totalEl
ements”: 2,        “totalPages”: 1},“timestamp”:“2023-08-14T15:38:50.4131
86769+03:00”}
```


发送应用程序版本草稿进行审核

用于提交应用程序版本草稿进行审核的方法。

交互参数

POSThttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/commit?priorityUpdate={priorityUpdate}

属性	类型	必须	布局	描述
Public-Token	string	是	header	Public API Rustore的访问令牌
packageName	string	是	path	应用程序的包名
versionId	number	是	path	应用程序版本
priorityUpdate	number	否	query	更新优先级。可能的值：从 0 到 5，其中 0 是最小的，5 是最大的。默认等于 0。

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	错误详细信息。可能的值包括：请至少上传一个主要的非HMS APK文件。/ Version must have at least one main

				non-HMS apk-file 您不能只上Huawei Mobile Services的文件。/ Version must have not only HMS apk-file 在创建草稿时您没有上传APK。 / Packages for version with id= Is not found.
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

手动发布

此方法允许手动发布已通过审核的版本。

限制：

- 只有通过审核后，才能发布版本；
- 如果发布类型是手动指定的，可以发布版本。

交互参数

POST https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/publish

属性	类型	必填	布局	描述
Public-Token	string	是	header	Public API Rustore的访问令牌
packageName	string	是	path	应用程序的包名
versionId	number	是	path	应用程序版本

响应参数

属性	类型	必填	描述	填写示例
code	string	是	响应代码	error/OK

message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

请求示例

curl –location –request POST 'https://public-api.rustore.ru/public/v1/application/com.example.pblsh_v2/version/704372/publish'
–header 'accept: application/json'
–header 'Public-Token: {YOURtoken}'

示例回复

{ "code": "OK", "message": null, "body": null, "timestamp": "2023-08-14T15:34:44.016339151+03:00"}

删除草稿

用于删除之前创建的草稿的方法。

限制：

- 只有那些尚未发布的版本才能被删除。

交互参数

DELETEhttps://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}

属性	类型	必须	布局	描述
Public-Token	string	是	header	Public API Rustore的访问令牌
packageName	string	是	path	应用程序的包名
versionId	number	是	path	应用程序版本

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

示例

```
curl –location –request DELETE ‘https://public-api.rustore.ru/public/v1/application/com.package.example/version/123’ <br>–header ‘Content-Type: application/json’ <br>–header ‘Public-Token: {YOURtoken}’
```

应用类别列表

类别名称	翻译	应用版本类型
business	商务服务	MAIN
state	政府	MAIN
foodAndDrink	食品和饮料	MAIN
health	健康	MAIN
books	书籍	MAIN
news	新闻和事件	MAIN
lifestyle	生活方式	MAIN
education	教育	MAIN
social	社交	MAIN
adsAndServices	广告和服务	MAIN
pets	宠物	MAIN
purchases	购物	MAIN
tools	实用工具	MAIN
travelling	旅行	MAIN
entertainment	娱乐	MAIN
parenting	家长	MAIN
sport	运动	MAIN
gambling	投注和彩票	MAIN
transport	交通和导航	MAIN
finance	金融	MAIN
arcade	街机游戏	GAMES
quiz	竞猜游戏	GAMES
puzzle	益智游戏	GAMES
race	赛车	GAMES
children	儿童	GAMES
ar	AR游戏	GAMES
indie	独立游戏	GAMES
casino	赌场	GAMES
casual	休闲	GAMES

类别名称	翻译	应用版本类型
card	卡牌	GAMES
music	音乐	GAMES
board	桌面游戏	GAMES
adventure	冒险	GAMES
rolePlaying	角色扮演	GAMES
family	家庭	GAMES
simulator	模拟	GAMES
word	文字游戏	GAMES
sports	体育游戏	GAMES
strategy	策略	GAMES
action	动作	GAMES

使用RuStore API处理评论

处理用户评论的一般流程

1. 开发者可以获取应用评论：
   - 使用“获取应用评论”的方法；
   - 以CSV格式 — “获取.csv格式评论”的方法；
2. 开发者可以回复用户留下的评论 — 使用“对评论回复”的方法；
3. 查询评论回复审核状态 — 使用“获取评论回复状态”的方法；
4. 可以修改已发布的评论回复 — 使用“修改评论回复”的方法；
5. 可以删除评论回复 — 使用“删除评论回复”的方法；
6. 获取应用评分 — 使用“获取应用评分”的方法。

为使用所有方法，需要获取RuStore API的访问令牌。令牌有效期为15分钟。

可能的错误代码选项

代码	含义
200	OK
400	错误请求
401	未授权

404　未找到

500　服务器内部错误

在错误情况下的响应示例

{“code”:“ERROR”,“message”:“404”,“body”:null,“timestamp”:“2023-05-30T20:08:14.120231216+03:00”}

获取应用程序评论

此方法允许获取您的应用程序的所有最新评论列表或单独获取某个评论。

限制：

- 默认情况下，每个页面显示20条评论，您可以在size参数中设置，最多显示100条评论；
- 不能同时使用分页和过滤参数来仅显示1条评论（要么用page和size参数进行分页，要么使用id）。
- 如果用户编辑了评论，则该评论将被分配一个更新的评论id，并且评论的编辑标志（edited）将设置为true；
- 默认情况下，评论按发布日期（updatedAt参数）排序，从最新的评论开始排序。

交互参数

GEThttps://public-api.rustore.ru/public/v1/application/{packageName}/comment?id={id}&page={number}&size={size}

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	RuStore API访问令牌	
packageName	string	是	path	应用程序包名称。最大长度为50个字符	com.myapp.example
id	number	否	query	特定评论的Id。在需要获取特定评论时设置	743103
page	number	否	query	页面编号。从0开始	0
size	number	否	query	每页的评论数量。默认为20，最大为100	100

响应参数

属性	类型	必须	描述	填写示例

code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
body{}	object	是		

body

属性	类型	必须	描述	示例
packageName	string	是	应用程序包的名称	com.myapp.example
appId	number	是	应用程序ID	385727
commentId	number	是	评论ID	697535
userName	string	是	评论作者名称	Irina
appRating	number	是	评论评分	4
commentStatus	string	是	评论状态	PUBLISHED
commentDate	timestamptz	是	评论留下的日期	2023-05-22 16:32:08.008
commentText	string	是	评论文本	一切都很棒！
likeCounter	number	是	RuStore展示平台上其他用户对该评论的点赞数	5
dislikeCounter	number	是	RuStore展示平台上其他用户对该评论的点踩数	0
updatedAt	timestamptz	是	评论通过审核并在系统中发布的日期	2022-10-14 15:14:33.033
appVersionName	string	是	应用程序版本	1.4
edited	boolean	是	评论编辑标志	True - 已编辑（评论已编辑并已重新写入）。 False - 否

多条评论的输出示例

```
curl-location’https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comment’<br>-header’Content-Type:application/json’<br>-header’Public-Token:eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBLU9BRVAtMjU2In0.jrVl3YT99saGcata9fzN6_QpoQhDsv8oBUAj9p7UyR4Ga5PM8TTyNbpTKIeaIjGoEfsBMJx0aw0b7fhD04bhSp7SWEBKGBTzFCjwOZ5_Fcezq0-NOMSayzoPttYa7oMRDiqNS4rqaUOdCUrf9qlDyEq3BGoUYCaUD7L5399l51NhSLrWpoPx1l4ZLVJ5bjlhiCoPAtLArnulq5LBoDk3naoGaRHabkkffcucEj
```

A45uNpCsq0fx77Lk4YhN30LOccyImE-O8fUq8YryMWv4w-ZmWjax1oT9nRgO95r9EYG7Gwdek
g2ILuWnsofiMXME8t1EiEuUmDpNTEyS9SiUMRwQ.lSJBV1mSmHehuqVs.8pXL_GRLwEJgIWZO
zkklKgrsGKKJrG9kv1AIdD0PPU8KtsY8GVAc5xaaQgeyjSsJiSUvma_IohAaIBwP-tjTRxrnz
VinMhKAJMCbiMIVqsQSRDB5j_mf91nTNewQkWJwB33Rvxd9F4Tt-Tk-1QKALU8tAT_HXAI.v9
WBgx8T6yFDpeMrjw-ECQ’

示例回复

{“code”:“OK”,“message”:null,“body”:[{“packageName”:“ru.voonsh.push”,“appI
d”:227169215,“commentId”:2142370751,“userName”:“Saber”,“appRating”:5,“com
mentStatus”:“PUBLISHED”,“commentDate”:“2023-06-1311:58:06.006”,“commentTe
xt”:“GoodApp,nicedone!”,“likeCounter”:0,“dislikeCounter”:0,“updatedAt”:“2
023-06-1311:59:50.050”,“appVersionName”:“1.4”,“edited”:false},{“packageNa
me”:“ru.voonsh.push”,“appId”:227169215,“commentId”:1981700287,“userName”:
“维克托”,
“appRating”:5,“commentStatus”:“PUBLISHED”,“commentDate”:“2023-04-2718:54:
27.027”,“commentText”:“优秀的应用程序！;
#”,“likeCounter”:0,“dislikeCounter”:0,“updatedAt”:“2023-04-2718:55:30.030
”,“appVersionName”:“1.3”,“edited”:false},{“packageName”:“ru.voonsh.push”,
“appId”:227169215,“commentId”:1981699775,“userName”:“格里高里”,
“appRating”:5,“commentStatus”:“PUBLISHED”,“commentDate”:“2023-04-2718:54:
18.018”,“commentText”:“优秀的应用程序！”,
“likeCounter”:1,“dislikeCounter”:0,“updatedAt”:“2023-04-2718:55:30.030”,“
appVersionName”:“1.3”,“edited”:false},{“packageName”:“ru.voonsh.push”,“ap
pId”:227169215,“commentId”:237681343,“userName”:“伊布拉欣”,
“appRating”:5,“commentStatus”:“PUBLISHED”,“commentDate”:“2022-09-1517:26:
46.046”,“commentText”:“提高DAU(日活跃用户数)”,
“likeCounter”:1,“dislikeCounter”:1,“updatedAt”:“2022-09-1618:06:39.039”,“
appVersionName”:null,“edited”:false}],“timestamp”:“2023-06-15T07:32:55.50
5979576+03:00”}

用于显示特定评论的示例

{ “code”: “OK”, “message”: null, “body”: [{ “
packageName”: “ru.voonsh.push”, “appId”: 227169215,
“commentId”: 2142370751, “userName”: “Saber”, “app
Rating”: 5, “commentStatus”: “PUBLISHED”, “commentD
ate”: “2023-06-13 11:58:06.006”, “commentText”: “Good App, nic
e done!”, “likeCounter”: 0, “dislikeCounter”: 0,
“updatedAt”: “2023-06-13 11:59:50.050”, “appVersionNa
me”: “1.4”, “edited”: false }, { “pac
kageName”: “ru.voonsh.push”, “appId”: 227169215, “c
ommentId”: 1981700287, “userName”: “维克托”,
“appRating”: 5, “commentStatus”: “PUBLISHED”,
“commentDate”: “2023-04-27 18:54:27.027”, “commentText”
: “优秀的应用程序！
;#”, “likeCounter”: 0, “dislikeCounter”: 0,
“updatedAt”: “2023-04-27 18:55:30.030”, “appVersionName”:
“1.3”, “edited”: false }, { “package
Name”: “ru.voonsh.push”, “appId”: 227169215, “comme

ntId": 1981699775, "userName": "格里高里",
"appRating": 5, "commentStatus": "PUBLISHED",
"commentDate": "2023-04-27 18:54:18.018", "commentText"
: "优秀的应用程序！",
"likeCounter": 1, "dislikeCounter": 0,
"updatedAt": "2023-04-27 18:55:30.030", "appVersionName": "1.3
", "edited": false }, { "packageName"
: "ru.voonsh.push", "appId": 227169215, "commentId"
: 237681343, "userName": "伊布拉欣",
"appRating": 5, "commentStatus": "PUBLISHED",
"commentDate": "2022-09-15 17:26:46.046", "commentText"
: "提高DAU(日活跃用户数)",
"likeCounter": 1, "dislikeCounter": 1,
"updatedAt": "2022-09-16 18:06:39.039", "appVersionName": null
, "edited": false }],"timestamp":"2023-06-15T07:32:55.5
05979576+03:00"}

用于显示特定评论的示例

```
curl -location
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comme
nt?id=1981699775'
<br>-header'Content-Type:application/json'<br>-header'Public-Token:eyJlbm
MiOiJBMjU2R0NNIiwiYWxnIjoiUlNBLU9BRVAtMjU2In0.h635qF_TZc43287jXQVImu_-o4e
VFQwrQYe2WnjbywC_KUC4oX6W3ssyPWzlAugd2RELbVCTk1wiDdKwPIbkOJC_HdF0yAmnPg0P
RxwyfCHblRuccuuEg_l6sKY1Fqrh6kH3D5N2i_HnDei-hTusAvHR333ZstAK73dc-4Ecn24jb
1XyBsdg0_KddKaRpEjTMSudIV6rdpBNMlQRUyQufrP2RMXK5Kc_0gY0iA-tazQoOJmK4xstHm
uFbSBx3J6oN5QIlYonx0LZ6ABf2fD0O1E7LFsVUMd2bOdLYg5id5bTRXKd238iB5snmPhGJJN
3d6v8xdoV5TdOMGPvjO0A5A.-mSbEMAvEglyEOJu.eKxraHkohwEcn3cG4glUBnwypjnapol4
WnwhQGyKe-TDq9TGNj9CO4hnXGh4UPSRI55w73pJwSCuDe7LfsQ8zqBVjirT_HXypowHsWBwv
BG-6rwSRvhZsad2YY8wHTZeXOddVynWnESoKAnXIdImafEbIWN1Hik.VJ-b1KxDxMrgJTz_Vu
ul2Q'
```

示例回复

{"code":"OK","message":null,"body":[{"packageName":"ru.voonsh.push","appI
d":227169215,"commentId":1981699775,"userName":"格里高里",
"appRating":5,"commentStatus":"PUBLISHED","commentDate":"2023-04-2718:54:
18.018","commentText":"优秀的应用程序！",
"likeCounter":1,"dislikeCounter":0,"updatedAt":"2023-04-2718:55:30.030","
appVersionName":"1.3","edited":false}],"timestamp":"2023-06-15T07:13:16.3
09841987+03:00"}

获取.csv格式的评论。

此方法允许在特定时间内以.csv格式获取所有评论。

限制：

- 选择评论的最短时间段 – 1天；

- 选择评论的最长时间段 – 92天。

交互参数

GET
https://public-api.rustore.ru/public/v1/application/{packageName}/comment/export?from={date_from}&to={date_to}

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	RuStore API访问令牌	
packageName	string	是	path	应用程序包名称。最大长度为50个字符	com.myapp.example
from	date	是	query	需要开始下载评论的日期，格式为YYYY-MM-DD	2023-06-01
to	date	是	query	需要结束下载评论的日期（包括该日），格式为YYYY-MM-DD	2023-06-05

响应参数

.csv格式文件

user comment for application com.publicapi.publicapi from 2023-05-22 to 2023-05-23 (1)

Package Name	User App Version	Review Submit Date and Time	Review Last Update Date and Time	Star Rating	Review Text	User name	Comment Id	Like counter	Dislike counter	edited
com.publicapi.publicapi		2023-05-22 16:32:08.446	2023-05-22 17:02:22.019	4	Клевые, но можно лучше	Irina	697535	0	0	false
com.publicapi.publicapi		2023-05-22 16:28:25.699	2023-05-22 17:02:59.293	2	Ну не знаю, мне не нравится	Айша	697279	0	0	false

文件内容

列标题	描述
Package Name	应用程序包的名称
User App Version	应用程序版本
Review Submit Date and Time	评论留下的日期
Review Last Update Date and Time	评论通过审核并在系统中发布的日期
Star Rating	评论评分
Review Text	评论文本
User name	评论作者名称
Comment Id	评论ID

Like counter	RuStore展示平台上其他用户对该评论的点赞数
Dislike counter	RuStore展示平台上其他用户对该评论的点踩数
edited	评论编辑标志

示例

```
curl-location’https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comment/export?from=2023-03-03&to=2023-04-28’<br>-header’Public-Token:eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBLU9BRVAtMjU2In0.h635qF_TZc43287jXQVImu_-o4eVFQwrQYe2WnjbywC_KUC4oX6W3ssyPWzlAugd2RELbVCTk1wiDdKwPIbkOJC_HdF0yAmnPg0PRxwyfCHblRuccuuEg_l6sKY1Fqrh6kH3D5N2i_HnDei-hTusAvHR333ZstAK73dc-4Ecn24jb1XyBsdg0_KddKaRpEjTMSudIV6rdpBNMlQRUyQufrP2RMXK5Kc_0gY0iA-tazQoOJmK4xstHmuFbSBx3J6oN5QIlYonx0LZ6ABf2fD0O1E7LFsVUMd2bOdLYg5id5bTRXKd238iB5snmPhGJJN3d6v8xdoV5TdOMGPvjO0A5A.-mSbEMAvEglyEOJu.eKxraHkohwEcn3cG4glUBnwypjnapol4WnwhQGyKe-TDq9TGNj9CO4hnXGh4UPSRI55w73pJwSCuDe7LfsQ8zqBVjirT_HXypowHsWBwvBG-6rwSRvhZsad2YY8wHTZeXOddVynWnESoKAnXIdImafEbIWN1Hik.VJ-b1KxDxMrgJTz_Vuul2Q’
```

对评论进行回复

此方法允许对评论进行回复。

交互参数

POST
https://public-api.rustore.ru/public/v1/application/{packageName}/feedback?commentId={commentId}

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	RuStore API访问令牌	
packageName	string	是	path	应用程序包名称。最大长度为50个字符	com.myapp.example
commentId	number	是	query	用户评论ID	748223
message	string	是	body	回复评论的文本	感谢您的评论。这对我们非常重要

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	

timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
body {}	object	是		

body

属性	类型	必须	描述	示例
id	number	是	回应评论的id	748479

示例

```
curl-location’https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback?commentId=2142370751’<br>-header’Content-Type:application/json’<br>-header’Public-Token:eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBLU9BRVAtMjU2In0.ziOR4J-_3A36M55IMdJqxck4Lktk9GFe6vvdRFBkIICA6z4WLLUxdEyHNdMWYomBs9MkJecCaCStOQ5YtmHvFIRK8aoj4c386WwUBXmTDXN_BJg2puwuLivMJWuAhgveZpC7afZCwM6m5RgLi538BAjFV_gE8XvbSUKlhWhkEvIzgrK2zk211SRUVXaAdrWEz5NNSSrQhyEv1fiMgQNmV9Sehp8fxP7G_9HkAWVtfNvgEiTbFHMTO-qpWtdh5Ts440Du9MC7PL59IUmXts1Khx6xbuUWLQe3WHPQCBmKezpxtI-l9Ms4F-iopZy-bXzUaUMFsaQu8Jh4kFvztFenYg.UPvgd5jKQcFO-MRE.zHHu-P5GSwUTvEMae-bu337jxpq25TbftTC7oF8r0APCfUqdx55CbO-PuIeGdjN08K4IC8GsWhLmKr9mqEeV9L-Dh5QSqA6M7GsLH8LAqrR1UX0Z849pyhrOt_Pz4SJ3YeHdu-ITfTL5Ysr0kECMXWIMcE24X8U.zHXeYuxDJf-Wtl9Vn0betw’<br>-data’{“message”:“Thankyou”}’
```

示例回复

```
{“code”:“OK”,“message”:null,“body”:{“id”:2149775551},“timestamp”:“2023-06-02T16:36:57.847391009+03:00”}
```

获取对评论回复的状态

此方法允许获取对评论回复的审核状态。

交互参数

GET
https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	RuStore API访问令牌	
packageName	string	是	path	应用程序包的名称。最大长度为50个字符	com.myapp.example
feedbackId	number	否	path	特定回应评论的id，在希望获取特定评论	743103

回应信息时设置

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
body{}	object	是		

body

属性	类型	必须	描述	示例
id	number	是	回应评论的id	748479
commentId	number	是	评论id	748223
text	string	是	回应文本	这是对用户评论的回应
status	string	是	回应评论的状态	PUBLISHED — 已发布，一切正常；MODERATION — 正在审核中；REJECTED — 审核未通过；DELETED — 已被开发者删除或更改。
date	string	是	最后修改的日期和时间	2023-06-01 18:10:43.043

示例1

```
curl -location
‘https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback/2177559743’ <br>-header ‘Public-Token:
eyJlbmMiO.....ELthVcQVmNyw’{    “code”: “OK”,    “message”:
null,    “body”: [        {            “id”:
“2177559743”,            “commentId”: “2142370751”,            “text”:
“Thank you”,            “status”: “PUBLISHED”,            “date”:
“2023-06-22T09:46:18.115+00:00”        }    ],    “timestamp”:
“2023-06-22T12:46:30.193419294+03:00”
```

示例回复

```
{    “code”: “OK”,    “message”: null,    “body”:
[        {            “id”: “2177559743”,            “commentId”:
“2142370751”,            “text”: “Thank you”,            “status”:
“PUBLISHED”,            “date”:
“2023-06-22T09:46:18.115+00:00”        }    ],    “timestamp”:
“2023-06-22T12:46:30.193419294+03:00”
```

获取对多个评论回复的状态

该方法允许获取评论回复的审核状态或获取特定评论回复的信息。

限制：

- 默认每页显示20条评论。您可以通过设置size参数来显示多达100条评论；
- 不能同时使用分页和过滤参数来仅显示1条评论回复（要么用page和size参数进行分页，要么使用id）。

交互参数

GET
https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/?id={id}&page={number}&size={size}

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	RuStore API 访问令牌	
packageName	string	是	path	应用程序包名称。最大长度为50个字符	com.myapp.example
feedbackId	number	否	path	特定回应评论的id，在希望获取特定评论回应信息时设置	743103
id	number	否	query	评论的具体回复的id，如果需要获取特定的评论时设置	743103
page	number	否	query	页面编号。从0开始	0
size	number	否	query	页面上的评论数量。默认为20，最多为100	100

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
body{}	object	是		

body

属性	类型	必须	描述	示例
id	number	是	回应评论的id	748479
commentId	number	是	评论id	748223
text	string	是	回应文本	这是对用户评论的回应
status	string	是	回应评论的状态	PUBLISHED — 已发布，一切正常；MODERATION — 审核中；REJECTED — 审核拒绝；DELETED — 开发者删除或已更改。
date	string	是	最后修改的日期和时间	2023-06-01 18:10:43.043

示例1

```
curl -location
‘https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback?id=2177559743’ <br>-header ‘Public-Token: eyJlb.....VcQVmNyw’
```

回复1

```
{    “code”: “OK”,    “message”: null,    “body”: [        {            “id”: “2177559743”,            “commentId”: “2142370751”,            “text”: “Thank you”,            “status”: “PUBLISHED”,            “date”: “2023-06-22T09:46:18.115+00:00”        }    ],    “timestamp”: “2023-06-22T12:46:30.193419294+03:00”
```

更改对评论的回复

此方法允许更改对评论的回复文本。

交互参数

POST
https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	Header	RuStore API 访问令牌	
packageName	string	是	path	应用程序包名称。最大长度为50个字符	com.myapp.example
feedbackId	number	否	path	需要修改的评论回复的id	743103
message	string	是	body	修改后的评论回复文本	这是修改后的回复，之前有个打字错误

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
body{}	string	是		

body

属性	类型	必须	描述	示例
id	number	是	新的评论回复id。前一个id的状态变为 — DELETED。在更改后，进一步的工作将继续使用更新的id。	748479

示例

```
curl-location’https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback/2149775551’<br>-header’Content-Type:application/json’<br>-header’Public-Token:eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBLU9BRVAtMjU2In0.ziOR4J-_3A36M55IMdJqxck4Lktk9GFe6vvdRFBkIICA6z4WLLUxdEyHNdMWYomBs9MkJecCaCStOQ5YtmHvFIRK8aoj4c386WwUBXmTDXN_BJg2puwuLivMJWuAhgveZpC7afZCwM6m5RgLi538BAjFV_gE8XvbSUKlhWhkEvIzgrK2zk211SRUVXaAdrWEz5NNSSrQhyEv1fiMgQNmV9Sehp
```

```
8fxP7G_9HkAWVtfNvgEiTbFHMTO-qpWtdh5Ts440Du9MC7PL59IUmXts1Khx6xbuUWLQe3WHP
QCBmKezpxtI-l9Ms4F-iopZy-bXzUaUMFsaQu8Jh4kFvztFenYg.UPvgd5jKQcFO-MRE.zHHu
-P5GSwUTvEMae-bu337jxpq25TbftTC7oF8r0APCfUqdx55CbO-PuIeGdjN08K4IC8GsWhLmK
r9mqEeV9L-Dh5QSqA6M7GsLH8LAqrR1UX0Z849pyhrOt_Pz4SJ3YeHdu-ITfTL5Ysr0kECMXW
IMcE24X8U.zHXeYuxDJf-Wtl9Vn0betw'<br>-data'{"message":"Thankyouverymuch!"
}'
```

示例回复

```
{"code":"OK","message":null,"body":{"id":2149807039},"timestamp":"2023-06
-15T08:26:25.355326578+03:00"}
```

删除对评论的回复

此方法允许删除对评论的回复。

交互参数

DELETE
https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	RuStore API访问令牌	
packageName	string	是	path	应用程序包名称。最大长度为50个字符	com.myapp.example
feedbackId	number	否	path	需要删除的评论回复的id	743103

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00

示例

```
curl-location-requestDELETE'https://public-api.rustore.ru/public/v1/appli
cation/ru.voonsh.push/feedback/2149807039'<br>-header'Content-Type:applic
ation/json'<br>-header'Public-Token:eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBL
U9BRVAtMjU2In0.1r3cxOdxuNpypJSWXMQ4oAVYhqh6_3RlqKfItkhthhTzisiRYnboOkZpw_
r5J9w0S5G8u-BexQeganyoG3MbCJ5QP2X6945wQMxlPkl81UKewkZuFrjsH36USk6dnnMbjT8
Yw8nA4Yr0n8Oinspj3zkw66kZd-57EJvoMfneCEyTBY1mYEoc2DnfUa99syX1kIgX7Jfipn4y
Rm3pxWad5aesCK3eQFlP57CBirm8qGecDzkKcV1DeBx-qXK1S72FaXId11zN-rbe14U1z8jCC
iEHhYrTIT9ci7OKF7OqF2kgRRVdCoO3eRWl4JWF-JfGAeYcv7rEpNLC32pEm5FOCA.khXZSeT
```

```
odz2mVoNd.fKVjmkHUEM9AW7S_mYf-LFO4T26Lqf4RNSyjyNmFfsxZybDKahZgnaJ4lXYq-MP
VN-o39eg1jISmoJcBonqS-0rIFe1P3CAM5cbNiSsTCX1r-cVdf4ei998KKGMg8bZL24-uLfxg
cJSBBTgmUkyvf_KqH_dcxmQ.DwbK_08RLgHibat3h5dvkQ'
```

示例回复

```
{"code":"OK","message":null,"body":null,"timestamp":"2023-05-30T20:07:22.
365025849+03:00"}
```

获取应用程序评分

此方法允许获取应用程序的评分。

交互参数

POST
https://public-api.rustore.ru/public/v1/application/{packageName}/comment/statistic

属性	类型	必须	布局	描述	填写示例
Public-Token	string	是	header	RuStore API 访问令牌	
packageName	string	是	path	应用程序包名称。最大长度为50个字符	com.myapp.example

响应参数

属性	类型	必须	描述	填写示例
code	string	是	响应代码	error/OK
message	string	否	响应代码解释	
timestamp	timestamptz	是	响应时间	2022-07-08T13:24:41.8328711+03:00
body{}	object	是		

body

属性	类型	必须	描述	示例
ratings	object	是		
averageUserRating	number	是	应用程序的平均用户评分	4
totalRatings	number	是	总评分数量	0
totalResponses	number	是	未回复的总评论数量	0
ratingsNoComments	number	是	未评论的总评分数量	0

raitings

#### 属性	#### 类型	#### 必需	#### 描述	#### 示例
amountFive	number	是	数量为«5»的评分	0
amountFour	number	是	数量为«4»的评分	3
amountThree	number	是	数量为«3»的评分	0
amountTwo	number	是	数量为«2»的评分	0
amountOne	number	是	数量为«1»的评分	0

示例

```
curl-location’https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comment/statistic’<br>-header’Public-Token:eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBLU9BRVAtMjU2In0.1r3cxOdxuNpypJSWXMQ4oAVYhqh6_3RlqKfItkhthhTzisiRYnboOkZpw_r5J9w0S5G8u-BexQeganyoG3MbCJ5QP2X6945wQMxlPkl81UKewkZuFrjsH36USk6dnnMbjT8Yw8nA4Yr0n8Oinspj3zkw66kZd-57EJvoMfneCEyTBY1mYEoc2DnfUa99syX1kIgX7Jfipn4yRm3pxWad5aesCK3eQFlP57CBirm8qGecDzkKcV1DeBx-qXK1S72FaXId11zN-rbe14U1z8jCCiEHhYrTIT9ci7OKF7OqF2kgRRVdCoO3eRWl4JWF-JfGAeYcv7rEpNLC32pEm5FOCA.khXZSeTodz2mVoNd.fKVjmkHUEM9AW7S_mYf-LFO4T26Lqf4RNSyjyNmFfsxZybDKahZgnaJ4lXYq-MPVN-o39eg1jISmoJcBonqS-0rIFe1P3CAM5cbNiSsTCX1r-cVdf4ei998KKGMg8bZL24-uLfxgcJSBBTgmUkyvf_KqH_dcxmQ.DwbK_08RLgHibat3h5dvkQ’
```

示例回复

```
{“code”:“OK”,“message”:null,“body”:{“ratings”:{“amountFive”:13,“amountFour”:2,“amountThree”:0,“amountTwo”:0,“amountOne”:0},“averageUserRating”:4.87,“totalRatings”:15,“totalResponses”:9,“ratingsNoComments”:4},“timestamp”:“2023-06-15T08:35:06.412194896+03:00”}
```