**来开源吧！发布开源组件到 MavenCentral 仓库超详细攻略**

**请点赞关注，你的支持对我意义重大。**

🔥 **Hi，我是小彭。本文已收录到**[**GitHub · AndroidFamily**](https://github.com/pengxurui/Android-NoteBook)**中。这里有 Android 进阶成长知识体系，有志同道合的朋友，关注公众号 [彭旭锐] 带你建立核心竞争力。**

小彭今天和群友讨论了一下学习方法的问题，觉得还挺感同身受的。有时候我们遇到不懂的地方，潜意识会产生厌恶和恐惧，大脑会驱使我们去学习和查看这个不懂的地方，结果有可能是陷入到另一个不懂的循环里，忘记了最初的目的。关于系统化学习和碎片化学习，你的想法是怎样的呢？评论区里告诉我吧。

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

**前言**

当一个开发者的水平提升到一定程度时，会有由内向外输出价值的需求，包括发布开源项目。而要发布开源组件，则需要将组件发布到公开的远程仓库，如 Jitpack、JenCenter 和 MavenCentral。其中，MavenCentral 是最流行的中央仓库，也是 Gradle 默认使用的仓库之一。

在这篇文章里，我将手把手带你发布组件到 MavenCentral 中央仓库。本文的示例程序使用小彭的开源项目 [**ModularEventBus**](https://github.com/pengxurui/ModularEventBus) 有用请给 Star，谢谢。

这不仅仅是一份攻略，还带着踩过一个个坑留下的泪和挠掉一根根落的宝贵发丝~~~

开发者系列文章：

* MavenCentral 仓库（本文）
* [**还在见招拆招？先看懂 APP 个人信息保护治理机制**](https://juejin.cn/post/7078940840009302052)
* [**使用 Markdown 高效率编写文档**](https://juejin.cn/post/6844903901548789773)
* [**技术周报 | 2021 年第 13 周**](https://juejin.cn/post/6947717736176812046)
* [**技术周报 | 2021 年第 11 周**](https://juejin.cn/post/6942139967917260836)
* [**技术周报 | 2021 年第 10 周**](https://juejin.cn/post/6939545355604197407)

**1. 概念梳理**

**1.1 什么是 POM？**

POM（Project Object Model）指项目对象模型，用于描述项目构件的基本信息。一个有效的 POM 节点中主要包含以下参数：

| **参数** | **描述** | **举例** |
| --- | --- | --- |
| groupId | 组织 / 公司名 | io.github.pengxurui |
| artifactId | 组件名 | modular-eventbus-annotation |
| version | 组件版本 | 1.0.0 |
| packaging | 格式 | jar |

**1.2 什么是仓库（repository）**

在项目中，我们会需要依赖各种各样的二方库或三方库，这些依赖一定会存放在某个位置（Place），这个 “位置” 就叫做仓库。使用仓库可以帮助我们管理项目构件，例如 jar、aar 等等。

**主流的构建工具都有 2 个层次的仓库概念：**

* **1、本地仓库：** 无论使用 Linux 还是 Window，计算机中会有一个目录用来存放从中央仓库或远程仓库下载的依赖文件；
* **2、远程仓库：** 包括中央仓库和私有仓库。中央仓库是开源社区提供的仓库，是绝大多数开源库的存放位置。比如 Maven 社区的中央仓库 [**Maven Central**](https://search.maven.org/)；私有仓库是公司或组织的自定义仓库，可以理解为二方库的存放位置。

**1.3 Sonatype、Nexus 和 Maven 的关系：**

* **Sonatype：** 完整名称是 Sonatype OSSRH（OSS Repository Hosting），为开源项目提供免费的中央存储仓库服务。其中需要用到 Nexus 作为仓库管理器；
* **Nexus：** 完整名称是 Sonatype Nexus Repository Manager，是 Sonatype 的另一款产品，用作提供仓库管理器。Sonatype 基于 Nexus 提供中央仓库，各个公司也可以使用 Nexus 搭建私有仓库；
* **Maven：** 完整名称是 Apache Maven，是一种构建系统。除了 Maven 之外，Apache Ant 和 Gradle 都可以发布组件。

**2. 新建 Sonatype 项目**

从这一节开始，我将带你一步步完成发布组件到中央仓库的操作（带你踩坑）。

**2.1 准备 Sonatype JIRA 账号**

进入 Sonatype 仪表盘界面，登录或注册新账号：[**https://issues.sonatype.org**](https://issues.sonatype.org/secure/Dashboard.jspa)：

图形用户界面, 文本, 应用程序

描述已自动生成

**2.2 新建工单**

点击仪表盘面板右上角的 **”新建“** 按钮，按照以下步骤向 Sonotype 提交新建项目的工单：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

填写方法总结如下：

* **项目：** 使用默认选项 Community Support - Open Source Project Repository Hosting (OSSRH)；
* **问题类型：** 使用默认选项 New Project；
* **概要：** 填写 Github 仓库相同的名称，以方便查找；
* **GroupId 组织名：** 填写发布组件时使用的 groupId，后续步骤中会检查你是否真实拥有该 groupId，所以不可以随便填写，有 2 种填写方式：
  + **使用 Github 账号：**按照 io.github.[Github 用户名] 的格式填写，后续步骤中 Sonatype 通过要求我们在个人 Github 仓库中新建指定名称的临时代码库的方式来做身份验证；
  + **使用个人域名：** 按照逆序域名的格式填写，例如个人域名为 oss.sonotype.org ，则填写 org.sonotype.oss 。
* **Project URL 项目地址：** 填写 Github 项目地址，例如： https://github.com/pengxurui/ModularEventBus ；
* **SCM url 版本控制地址：** 在 Github 项目地址后加 .git ，例如 https://github.com/pengxurui/ModularEventBus.git 。

**2.3 验证 GroupId 所有权**

点击弹出的消息进入工单详情页，刚新建的工单要等待 Sonotype 机器人回复，等待大概十几分钟后，在工单底部的评论区会告诉我们怎么操作。

图形用户界面

描述已自动生成图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

至此，Sonotype 项目准备完毕。

**3. 新建 GPG 密钥对**

**GPG（GNU Privacy Guard）** 是基于 OpenPGP 标准实现的加密软件，它提供了对文件的非对称加密和签名验证功能。所有发布到 Maven 仓库的文件都需要进行 GPG 签名，以验证文件的合法性。

**3.1 安装 GPG 软件**

安装 GPG 软件有两种方式：

* **方式 1 - 下载安装包：** 通过 [**GPG 官方**](https://gpgtools.org/) 下载安装包，这个我没试过；
* **方式 2 - 通过 Homebrew 安装：** 使用 Homebrew 执行以下命令：

命令行

*# 通过 Homebrew 安装 gpg*  
brew install gpg

如果本地没有 Homebrew 环境则需要先安装，这里也容易踩坑。小彭本地原本就有 Homebrew 环境，但是安装 gpg 的过程中各种报错，最后还是用了最暴力的解法才解决 —— 卸载重装 Homebrew：(

**参考资料：** [**MacOS下开发环境配置--homebrew的安装**](https://blog.csdn.net/Chaoggo/article/details/124932539)

**3.2 生成 GPG 密钥对**

使用 --generate-key 参数，按照指引填写相关信息和 passphrase 私钥口令。另外，使用 --list-keys 参数可以查看当前系统中生成过的密钥。

命令行

*# 密钥生成命令*  
gpg --generate-key  
  
*# 密钥查看命令*  
gpg --list-keys

命令行演示

文本

描述已自动生成

GPG 在生成密钥对时，会要求开发者做一些随机的举动，以给随机数加入足够多的扰动，稍等片刻就会生成完成了。完成后可以随时使用 —list-keys 参数查看密钥对信息：

命令行演示

文本

描述已自动生成

解释一下其中的信息：

/Users/pengxurui/.gnupg/pubring.kbx  
-----------------------------------  
pub   ed25519 2022-08-23 [SC] [expires: 2024-08-22]  
      D8BCD08568BE5D2D634DD99EFD4ECE3B54DE73AA  
uid           [ultimate] test <test@gmail.com>  
sub   cv25519 2022-08-23 [E] [expires: 2024-08-22]  
  
*# pubring.kbx：本地存储公钥的文件*  
*# 2022-08-23 [SC] [expires: 2024-08-22]：表示密钥对的创建时间和失效时间*  
*# test <test@gmail.com>：用户名和邮箱*  
*# ed25519：表示生成公钥的算法*  
*# cv25519：表示生成私钥的算法*  
*# D8BCD08568BE5D2D634DD99EFD4ECE3B54DE73AA：密钥指纹 / KeyId*

至此，你已经在本地生成一串新的密钥对，现在你手上有：

* **密钥指纹 / KeyId：** 密钥指纹是密钥对的唯一标识，即上面 D8BCD08568BE5D2D634DD99EFD4ECE3B54DE73AA 这一串。有时也可以使用较短的格式，取其最后 8 个字符，即 B54DE73AA 这一串；
* **公钥：** 该密钥指纹对应的公钥；
* **私钥：** 该密钥指纹对应的私钥；
* **passphrase 密钥口令：** 生成密钥对时输入的口令，私钥与密钥口令共同组成密钥对的私有信息。

**3.3 删除密钥对**

有时候需要删除密钥对，可以使用以下命令：

*# 先删除私钥后，才能删除公钥*  
  
*# 删除私钥*  
gpg --delete-secret-keys  [密钥指纹]  
*# 删除公钥*  
gpg --delete-keys  [密钥指纹]

**3.4 上传公钥**

密钥对中的公钥信息需要公开，其他人才能拿到公钥来验证你签名的数据，公开的方法就是上传到公钥服务器。公钥服务器是专门储存用户公钥的服务器，并且会用交换机制将数据同步给其它公钥服务器，因此你只要上传到其中一个服务器即可。我最后是上传到 hkp://keyserver.ubuntu.com 服务器的。以下服务器都可以尝试：

* pool.sks-keyservers.net
* keys.openpgp.org
* keyserver.ubuntu.com
* pgp.mit.edu

命令行

*// 上传公钥*  
gpg --keyserver 【服务器地址】:11371 --send-keys 【密钥指纹】  
*// 验证公钥*  
gpg --keyserver 【服务器地址】:11371 --recv-keys 【密钥指纹】

**3.5 导出密钥文件**

后文发布组件的时候需要用到密钥口令和私钥文件，可以使用以下参数导出

命令行

*# 默认导出到本地目录 /User/[用户名]/*  
  
*# 导出公钥*  
gpg --export 【密钥指纹】 > xiaopeng\_pub.gpg  
*# 导出私钥*  
gpg --export-secret-keys 【密钥指纹】 > xiaopeng\_pri.gpg

**3.6 踩坑：PGPException: unknown public key algorithm encountered**

我在发布组件时遇到 PGPException: unknown public key algorithm encountered 报错，最后排查下来是使用了 Gradle signing 插件不支持 EDDSA 算法，需要使用 RSA 算法。

可以看到上文 3.1 节生成的公钥，可以看到是 ed 开头的，表示使用的是 EDDSA 算法，应该是不同版本中的 --generate-key 参数使用的默认算法不一样。

3.1 节生成的公钥信息

pub   ed25519 2022-08-23 [SC] [expires: 2024-08-22]

解决方法是使用 --full-generate-key 参数选择使用 RSA 算法生成密钥对：

命令行演示

文本

描述已自动生成

至此，密钥对准备完毕。

**4. 配置发布脚本**

完成 Sonatype 项目和密钥对的准备工作后，现在着手配置项目的 Gradle 脚本了。Gradle 提供了两个 Maven 插件：

* **maven 插件：** 旧版发布插件，从 Gradle 7.0 开始无法使用；
* **maven-publish 插件：** 新版发布插件。

我最初的想法是分别整理出这两个插件的通用脚本，一开始是参考 ARouter 项目里的 publish.gradle 脚本，过程中也遇到各种问题，例如 Javadoc generation failed ，可能是因为 ARouter 是纯 Java 实现的，所以暴露的问题较少。耽搁了一周后，刚好这两天在看 LeakCanary 源码，果然在 LeakCanary 里发现宝藏 —— vanniktech 的发布插件！

报错

Execution failed for task ':eventbus\_api:androidJavadocs'.  
> Javadoc generation failed. Generated Javadoc options file (useful for troubleshooting): '/Users/pengxurui/workspace/public/ModularEventBus/eventbus\_api/build/tmp/androidJavadocs/javadoc.options'

**4.1 使用 maven 插件发布**

这块脚本是参考 ARouter 项目中 publish.gradle 脚本的，我在此基础上增加了注释和少量改动，如果遇到生成 Javadoc 出现问题，可以把 archives androidJavadocsJar 这一行注释掉。

maven\_sonatype.gradle

*// 在 ARouter 项目的 publish.gradle 上修改*  
  
apply plugin: 'maven'  
apply plugin: 'signing'  
  
version = VERSION\_NAME  
group = GROUP  
  
*// 是否 Release 发布（根据是否包含 SNAPSHOT 判断）*  
def isReleaseBuild() {  
    return VERSION\_NAME.contains("SNAPSHOT") == false  
}  
  
*// Central Repository: https://central.sonatype.org/publish/publish-guide/*  
*// Release 仓库地址（默认先发布到 staging 暂存库，需要手动发布到中央仓库）*  
def getReleaseRepositoryUrl() {  
    return hasProperty('RELEASE\_REPOSITORY\_URL') ? RELEASE\_REPOSITORY\_URL : "https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/"  
}  
  
*// Snapshot 仓库地址*  
def getSnapshotRepositoryUrl() {  
    return hasProperty('SNAPSHOT\_REPOSITORY\_URL') ? SNAPSHOT\_REPOSITORY\_URL : "https://s01.oss.sonatype.org/content/repositories/snapshots/"  
}  
  
*// 仓库账号*  
def getRepositoryUsername() {  
    return hasProperty('SONATYPE\_NEXUS\_USERNAME') ? SONATYPE\_NEXUS\_USERNAME : ""  
}  
  
*// 仓库密码*  
def getRepositoryPassword() {  
    return hasProperty('SONATYPE\_NEXUS\_PASSWORD') ? SONATYPE\_NEXUS\_PASSWORD : ""  
}  
  
*// 组件配置*  
def configurePom(pom) {  
    *// 组织名*  
    pom.groupId = GROUP  
    *// 组件名*  
    pom.artifactId = POM\_ARTIFACT\_ID  
    *// 组件版本*  
    pom.version = VERSION\_NAME  
  
    pom.project {  
        *// 名称*  
        name POM\_NAME  
        *// 发布格式*  
        packaging POM\_PACKAGING  
        *// 描述信息*  
        description POM\_DESCRIPTION  
        *// 主页*  
        url POM\_URL  
  
        scm {  
            url POM\_SCM\_URL  
            connection POM\_SCM\_CONNECTION  
            developerConnection POM\_SCM\_DEV\_CONNECTION  
        }  
  
        *// Licenses 信息*  
        licenses {  
            license {  
                name POM\_LICENCE\_NAME  
                url POM\_LICENCE\_URL  
                distribution POM\_LICENCE\_DIST  
            }  
        }  
  
        *// 开发者信息*  
        developers {  
            developer {  
                id POM\_DEVELOPER\_ID  
                name POM\_DEVELOPER\_NAME  
            }  
        }  
    }  
}  
  
afterEvaluate { project ->  
    *// 配置 Maven 插件的 uploadArchives 任务*  
    uploadArchives {  
        repositories {  
            mavenDeployer {  
                *// 配置发布前需要签名*  
                beforeDeployment { MavenDeployment deployment -> signing.signPom(deployment) }  
                *// 配置 Release 仓库地址与账号密码*  
                repository(url: getReleaseRepositoryUrl()) {  
                    authentication(userName: getRepositoryUsername(), password: getRepositoryPassword())  
                }  
                *// 配置 Snapshot 仓库地址与账号密码*  
                snapshotRepository(url: getSnapshotRepositoryUrl()) {  
                    authentication(userName: getRepositoryUsername(), password: getRepositoryPassword())  
                }  
                *// 配置 POM 信息*  
                configurePom(pom)  
            }  
        }  
    }  
    *// 配置 Maven 本地发布任务*  
    tasks.create("installLocally", Upload) {  
        configuration = configurations.archives  
  
        repositories {  
            mavenDeployer {  
                *// 本地仓库地址*  
                repository(url: "file://${rootProject.buildDir}/localMaven")  
                *// 配置 POM 信息*  
                configurePom(pom)  
            }  
        }  
    }  
  
    *// 配置签名参数，部分需要在 local.properties 中配置*  
    signing {  
        required { isReleaseBuild() && gradle.taskGraph.hasTask("uploadArchives") }  
        sign configurations.archives  
    }  
  
    if (project.getPlugins().hasPlugin('com.android.application') || project.getPlugins().hasPlugin('com.android.library')) {  
        *// Android 类型组件*  
        task install(type: Upload, dependsOn: assemble) { *// 依赖于 AGP assemble 任务*  
            repositories.mavenInstaller {  
                configuration = configurations.archives  
  
                configurePom(pom)  
            }  
        }  
  
        task androidJavadocs(type: Javadoc) {  
            source = android.sourceSets.main.java.source  
            classpath += project.files(android.getBootClasspath().join(File.pathSeparator))  
        }  
  
        task androidJavadocsJar(type: Jar, dependsOn: androidJavadocs) {  
            classifier = 'javadoc'  
            from androidJavadocs.destinationDir  
        }  
  
        *// 生成源码产物*  
        task androidSourcesJar(type: Jar) {  
            classifier = 'sources'  
            from android.sourceSets.main.java.source  
        }  
    } else {  
        *// 纯 Java / Kotlin 类型组件（如 Gradle 插件、APT 组件）*  
        install {  
            repositories.mavenInstaller {  
                configurePom(pom)  
            }  
        }  
  
        *// 生成源码产物*  
        task sourcesJar(type: Jar, dependsOn: classes) {  
            classifier = 'sources'  
            from sourceSets.main.allSource  
        }  
  
        *// 生成 javadoc 产物*  
        task javadocJar(type: Jar, dependsOn: javadoc) {  
            classifier = 'javadoc'  
            from javadoc.destinationDir  
        }  
    }  
  
    *// Java8 适配*  
    if (JavaVersion.current().isJava8Compatible()) {  
        allprojects {  
            tasks.withType(Javadoc) {  
                options.addStringOption('Xdoclint:none', '-quiet')  
            }  
        }  
    }  
  
    *// 配置源码和 Javadoc 发布产物*  
    if (!isReleaseBuild()) {  
        *// 快照版本跳过，提高效率*  
        artifacts {  
            if (project.getPlugins().hasPlugin('com.android.application') || project.getPlugins().hasPlugin('com.android.library')) {  
                *// Android 类型组件*  
                archives androidSourcesJar *// 源码*  
                archives androidJavadocsJar *// Javadoc，如果报错需要把这一行注释掉*  
            } else {  
                *// 纯 Java / Kotlin 类型组件（如 Gradle 插件、APT 组件）*  
                archives sourcesJar *// 源码*  
                archives javadocJar *// Javadoc*  
            }  
        }  
    }  
}

在需要发布的组件里应用这个脚本后，在 gradle.properties 里配置相关参数后就可以发布了。具体可以参考示例程序 [**ModularEventBus**](https://github.com/pengxurui/ModularEventBus/) 中被注释掉的参数，也可以参考 ARouter 项目，这里就不展开了，建议用 4.2 节 vanniktech 的发布插件。

项目级 gradle.properties

######################################################################  
# for maven\_sonatype.gradle  
######################################################################  
# GROUP=io.github.pengxurui  
#  
# POM\_URL=https:*//github.com/pengxurui/ModularEventBus/*  
# POM\_SCM\_URL=https:*//github.com/pengxurui/ModularEventBus/*  
# POM\_SCM\_CONNECTION=scm:git:git:github.com/pengxurui/ModularEventBus.git  
# POM\_SCM\_DEV\_CONNECTION=scm:git:ssh:*//git@github.com/pengxurui/ModularEventBus.git*  
#  
# POM\_LICENCE\_NAME=The Apache Software License, Version 2.0  
# POM\_LICENCE\_URL=http:*//www.apache.org/licenses/LICENSE-2.0.txt*  
# POM\_LICENCE\_DIST=repo  
#  
# POM\_DEVELOPER\_ID=pengxurui  
# POM\_DEVELOPER\_NAME=Peng Xurui  
#  
# SONATYPE\_NEXUS\_USERNAME=[provide your Sonatype user name]  
# SONATYPE\_NEXUS\_PASSWORD=[provide your Sonatype password]  
#  
# signing.keyId=[provide you gpg key]  
# signing.password=[provide you gpg passphrase]  
# signing.secretKeyRingFile=[provide you gpg secret file]

模块级 gradle.properties

######################################################################  
# for maven\_sonatype.gradle  
######################################################################  
# POM\_NAME=ModularEventBus Annotations  
# POM\_ARTIFACT\_ID=modular-eventbus-annotation  
# POM\_PACKAGING=jar  
# POM\_DESCRIPTION=The annotation used in ModularEventBus api  
# VERSION\_NAME=1.0.0

**4.2 使用 vanniktech 的发布插件（推荐）**

[**gradle-maven-publish-plugin**](https://github.com/vanniktech/gradle-maven-publish-plugin) \*\*\*\*\*\*\*\*是一个外国大佬 vanniktech 开源的 Gradle 插件，需要使用 Gradle 7.2.0 以上的 Gradle 环境。它会创建一个 publish Task，支持将 Java、Kotlin 或 Android 组件发布到任何 Maven 仓库，同时也支持发布携带 Java / Kotlin 代码的 Javadoc 产物和 Sources 产物。虽然目前（2022/08/24）这个项目的最新版本只是 0.21.0，不过既然已经在 LeakCanary 上验证过，大胆用起来吧。

以下为配置步骤：在项目级 build.gradle 中添加插件地址，在模块级 build.gradle 中应用插件：

项目级 build.gradle

buildscript {  
    repositories {  
        mavenCentral()  
    }  
    dependencies {  
        *// vanniktech 发布插件*  
        classpath 'com.vanniktech:gradle-maven-publish-plugin:0.18.0'  
        *// Kotlin Javadoc，非必须*  
        classpath "org.jetbrains.dokka:dokka-gradle-plugin:1.6.20"  
        *// 最新版 1.7.10 和 0.21.0 组合有问题，应该是没兼容好。上面两个版本组合我验证过是可以的。*  
    }  
}

模块级 build.gradle

apply plugin: "com.vanniktech.maven.publish"  
*// Kotlin Javadoc，非必须。如果有这个插件，发布时会生成 Javadoc，会延长发布时间。建议在 snapshot 阶段关闭*  
apply plugin: "org.jetbrains.dokka"

Sync 项目后，插件会为模块增加两个 Task 任务：

* **publish：** 发布到远程 Maven 仓库，默认是 Sonatype 中央仓库；
* **publishToMavenLocal：** 发布到当前机器的本地 Maven 仓库，即 ~/.m2/repository。

Gradle 面板

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**4.3 配置 vanniktech 插件的发布参数**

分别在项目级 gradle.properties 和模块级 gradle.properties 中配置以下参数：

项目级 gradle.properties

######################################################################  
# for vanniktech  
######################################################################  
# 服务器地址  
SONATYPE\_HOST=S01  
  
# 发布 release 组件时是否签名  
RELEASE\_SIGNING\_ENABLED=true  
  
# 组织名  
GROUP=io.github.pengxurui  
  
# 主页  
POM\_URL=https:*//github.com/pengxurui/ModularEventBus/*  
  
# 版本控制信息  
POM\_SCM\_URL=https:*//github.com/pengxurui/ModularEventBus/*  
POM\_SCM\_CONNECTION=scm:git:git:github.com/pengxurui/ModularEventBus.git  
POM\_SCM\_DEV\_CONNECTION=scm:git:ssh:*//git@github.com/pengxurui/ModularEventBus.git*  
  
# Licenses 信息  
POM\_LICENSE\_NAME=The Apache Software License, Version 2.0  
POM\_LICENSE\_URL=https:*//www.apache.org/licenses/LICENSE-2.0.txt*  
POM\_LICENSE\_DIST=repo  
  
# 开发者信息  
POM\_DEVELOPER\_ID=pengxurui  
POM\_DEVELOPER\_NAME=Peng Xurui  
POM\_DEVELOPER\_URL=https:*//github.com/pengxurui/*  
  
mavenCentralUsername=[填 Sonatype 账号名]  
mavenCentralPassword=[填 Sonatype 密码]  
  
signing.keyId=[密钥指纹，取后 8 位即可]  
signing.password=[passphrase 密钥口令]  
signing.secretKeyRingFile=[导出的私钥文件路径，如 /Users/pengxurui/xxx.gpg]

模块级 gradle.properties

POM\_NAME=ModularEventBus Annotations  
POM\_ARTIFACT\_ID=modular-eventbus-annotation  
POM\_PACKAGING=jar  
POM\_DESCRIPTION=The annotation used in ModularEventBus api  
VERSION\_NAME=1.0.0

**特别注意：私有信息不要提交到 git 版本管理中，可以写在 local.properties 中，等到要发布组件时再复制到 gradle.properties 中。而私钥文件也不要保存在当前工程的目录里，可以统一放到工程外的一个目录。**

至此，所有准备工作完成。

**4.4 浅尝一下 vanniktech 插件的源码**

毕竟发布逻辑都被人家封装在插件里了，有必要知道它背后的工作，浅尝一下。

* **支持的 Snoatype 服务器：**

SonatypeHost.kt

enum class SonatypeHost(  
    internal val rootUrl: String  
) {  
    DEFAULT("https://oss.sonatype.org"),  
    S01("https://s01.oss.sonatype.org"),  
}

* **支持 Dokka 插件，需要手动依赖：**

MavenPublishPlugin.kt

private fun Project.defaultJavaDocOption(): JavadocJar? {  
    return if (plugins.hasPlugin("org.jetbrains.dokka") || plugins.hasPlugin("org.jetbrains.dokka-android")) {  
        JavadocJar.Dokka(findDokkaTask())  
    } else {  
        null  
    }  
}

* **支持多种模块类型：**

MavenPublishPlugin.kt

afterEvaluate {  
    when {  
        plugins.hasPlugin("org.jetbrains.kotlin.multiplatform") -> {} *// Handled above.*  
        plugins.hasPlugin("com.android.library") -> {} *// Handled above.*  
        plugins.hasPlugin("java-gradle-plugin") ->  
            baseExtension.configure(GradlePlugin(defaultJavaDocOption() ?: javadoc()))  
        plugins.hasPlugin("org.jetbrains.kotlin.jvm") ->  
            baseExtension.configure(KotlinJvm(defaultJavaDocOption() ?: javadoc()))  
        plugins.hasPlugin("org.jetbrains.kotlin.js") ->  
            baseExtension.configure(KotlinJs(defaultJavaDocOption() ?: JavadocJar.Empty()))  
        plugins.hasPlugin("java-library") ->  
            baseExtension.configure(JavaLibrary(defaultJavaDocOption() ?: javadoc()))  
        plugins.hasPlugin("java") ->  
            baseExtension.configure(JavaLibrary(defaultJavaDocOption() ?: javadoc()))  
        else -> logger.warn("No compatible plugin found in project $name for publishing")  
    }  
}

**5. 发布组件到 MavenCentral 仓库**

终于终于，所有准备和配置工作都完成了！在发布之前，有必要先解释下 Sonatype 中用到的仓库地址：

**5.1 仓库地址**

如果你没有自定义发布的 Maven 仓库，vanniktech 插件默认会发布到 Sonatype 管理的中央仓库中。由于历史原因，Sonatype 中央仓库有 2 个域名：

* https://s01.oss.sonatype.org/
* https://oss.sonatype.org/

按照 [**官方的说法**](https://central.sonatype.org/publish/publish-guide/#deployment) ，oss.sonatype.org 是过时的，从 2021 年 2 月开始启用 s01.oss.sonatype.org/

截图

图形用户界面, 文本, 应用程序

描述已自动生成

官方也会提示目前最新的仓库地址：

截图

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**5.2 Staging 暂存库**

细心的朋友会发现官方提供的 snapshot 仓库和 release 仓库的格式不一样，为什么呢？—— 这是因为发布 release 组件是敏感操作，一旦组件发布 release 版本到中央仓库，就永远无法修改或删除这个版本的组件内容（这个规则是出于稳定性和可靠性考虑，如果可以修改，那些本地已经下载过组件的用户就得不到最新内容了）。所以 Sonatype 对发布 snapshot 组件和 release 组件采取了不同策略：

* **snapshot 组件：** 直接发布到 snapshot 中央仓库；
* **release 组件：** 使用 Staging 暂存策略，release 组件需要先发布到暂存库，经过测试验证通过后，再由开发者手动提升到 release 中央仓库。

中央 release 仓库："https://s01.oss.sonatype.org/content/repositories/releases"  
中央 snapshot 仓库："https://s01.oss.sonatype.org/content/repositories/snapshots"  
暂存库："https://s01.oss.sonatype.org/service/local/staging/deploy/maven2"

手机屏幕的截图

中度可信度描述已自动生成

vanniktech 插件默认也是按照 Sonatype 的策略走的，浅看一下源码：

MavenPublishBaseExtension.kt

*// 暂存库：*  
if (stagingRepositoryId != null) {  
    repo.setUrl("${host.rootUrl}/service/local/staging/deployByRepositoryId/$stagingRepositoryId/")  
} else {  
    repo.setUrl("${host.rootUrl}/service/local/staging/deploy/maven2/")  
}  
  
*// snapshot 库：*  
if (it.version.toString().endsWith("SNAPSHOT")) {  
    if (stagingRepositoryId != null) {  
        throw IllegalArgumentException("Staging repositories are not supported for SNAPSHOT versions.")  
    }  
    repo.setUrl("${host.rootUrl}/content/repositories/snapshots/")  
}

**5.3 发布 snapshot 组件**

版本号中带 SNAPSHOT 将被视为 snapshot 组件，会直接发布到 snapshot 中央仓库。经过小彭验证，确实在前端发布后，立马可以在 snapshot 中央仓库搜索到，例如 [**小彭的组件**](https://s01.oss.sonatype.org/content/repositories/snapshots/io/github/pengxurui/)。

验证截图

图形用户界面, 文本, 应用程序

描述已自动生成

**5.4 发布 release 组件到 Staging 暂存库**

版本号中未带 SNAPSHOT 将视为 release 组件，发布 release 组件后，进入 [**Nexus**](https://s01.oss.sonatype.org/#stagingRepositories) 面板查看暂存库（右上角 Log in 登录）：

操作截图

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**5.5 发布 release 组件到中央仓库**

确认要发布组件后，先点击 Close，再点击 Release 即可发布：

操作截图

图形用户界面, 应用程序

描述已自动生成

Close 的过程会对组件进行验证，验证失败的话就会报错了。你可以直接从 Activity 面板中查看报错提示，我遇到的几次问题都是参数缺失的小问题。

报错提示

图形用户界面, 网站

描述已自动生成

点击 Drop 按钮删除有问题的组件：

操作截图

图形用户界面, 应用程序

描述已自动生成

如果验证通过，Release 按钮就会高亮，点击按钮就终于终于发布了。

操作截图

图形用户界面, 应用程序

描述已自动生成

**5.6 查看已发布的 release 组件**

发布成功后，有 3 种方式查看自己的组件：

* **方法 1 - 在 Sonatype Nexus 面板上查看：**

操作截图

图形用户界面, 应用程序

描述已自动生成

* **方法 2 - 在 release 中央仓库的文件目录中查看，例如**[**小彭的 release 组件**](https://s01.oss.sonatype.org/content/repositories/releases/io/github/pengxurui/modular-eventbus-annotation/)**：**

操作截图

图形用户界面, 文本, 应用程序, 表格

描述已自动生成

* **方式 3 - 在**[**MavenCentral 搜索栏**](https://search.maven.org/)**查找：** 这是最正式的方式，缺点是不实时更新，大概有 的延迟，而前两种方式在发布后立即更新：

操作截图

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

按照 [**官方的说法**](https://central.sonatype.org/publish/publish-guide/) ，发布后的组件会在 30 分钟内同步到中央仓库，但搜索功能需要达到 4 个小时：

Upon release, your component will be published to Central:   
this typically occurs within 30 minutes,  
though updates to search can take up to four hours.

**5.6 依赖已发布的组件**

怎么依赖大家都懂。讲一下仓库吧，如果是已经发布到 release 中央仓库，你的工程只要包含 mavenCentral() 这个仓库地址就可以了。

示例程序

repositories {  
    *// 中央仓库（不包含 snapshot 中央仓库）*  
    mavenCentral()  
    *// release 中央仓库*  
    maven { url 'https://s01.oss.sonatype.org/content/repositories/releases'}  
    *// snapshot 中央仓库*  
    maven { url 'https://s01.oss.sonatype.org/content/repositories/snapshots/'}  
    *// 暂存库，用于验证*  
    maven { url "https://s01.oss.sonatype.org/service/local/staging/deploy/maven2"}    
}

**6. 报错记录**

* Sonatype 账号密码错误：

Failed to publish publication 'maven' to repository 'mavenCentral'  
   > Could not PUT 'https://s01.oss.sonatype.org/service/local/staging/deploy/maven2/io/github/pengxurui/modular-eventbus-annotation/1.0.2/modular-eventbus-annotation-1.0.2.jar'. Received status code 401 from server: Unauthorized

* GPG 密钥错误：

Execution failed for task ':eventbus\_annotation:signMavenPublication'.  
> Error while evaluating property 'signatory' of task ':eventbus\_annotation:signMavenPublication'  
   > org.bouncycastle.openpgp.PGPException: checksum mismatch at 0 of 20

* GPG 密钥算法错误：

Execution failed for task ':eventbus\_annotation:signMavenPublication'.  
> Error while evaluating property 'signatory' of task ':eventbus\_annotation:signMavenPublication'  
   > org.bouncycastle.openpgp.PGPException: unknown public key algorithm encountered

* Javadoc 生成报错：

Execution failed for task ':eventbus\_api:androidJavadocs'.  
> Javadoc generation failed. Generated Javadoc options file (useful for troubleshooting): '/Users/pengxurui/workspace/public/ModularEventBus/eventbus\_api/build/tmp/androidJavadocs/javadoc.options'

* vanniktech 插件与 Dokka 插件兼容问题：

Execution failed for task ':eventbus\_api:javaDocReleaseGeneration'.  
> 'void org.jetbrains.dokka.DokkaSourceSetImpl.<init>(java.lang.String, org.jetbrains.dokka.DokkaSourceSetID,  
...

* POM 验证错误：

图形用户界面, 文本, 应用程序

描述已自动生成

**7. 寻求 Sonatype 官方帮助**

如果你在使用 Sonatype 的过程中遇到任何问题，可以尝试向官方提问。我试过一次，10 分钟后就收到回复了，还是很 Nice 的。

操作截图

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

操作截图

图形用户界面, 文本, 应用程序

描述已自动生成

**8. 总结**

恭喜，到这里，我们已经能够实现发布开源项目到 MavenCentral 中央仓库。还没完，引出两个问题：

* **Github Action：** 每次发布都需要我们手动执行 upload 任务，Github 仓库中的 Releases 面板也不会同步显示手动发布的版本记录。 **我们期望的效果是在 Github 仓库上发布一个 Release 版本时，自动触发将该版本发布到 MavenCentral 中央仓库。** 这需要用到 Github 提供的 CI/CD 服务 —— Github Action；
* [**ModularEventBus**](https://github.com/pengxurui/ModularEventBus)**：** 本文的示例程序，它是做什么的呢？

关注我，带你了解更多。

**参考资料**

* [**Sonotype · 常见问题 Q&A**](https://central.sonatype.org/faq/) —— Sonotype 官方文档
* [**Sonatype · GPG**](https://central.sonatype.org/publish/requirements/gpg/) —— Sonatype 官方文档
* [**Sonatype · Gradle**](https://central.sonatype.org/publish/publish-gradle/) —— Sonatype 官方文档
* [**Sonatype · Managing Staging Repositories**](https://help.sonatype.com/repomanager2/staging-releases/managing-staging-repositories) —— Sonatype 官方文档
* [**Sonatype · Release**](https://central.sonatype.org/publish/release/) —— Sonatype 官方文档
* [**Github · 生成新 GPG 密钥**](https://docs.github.com/cn/authentication/managing-commit-signature-verification/generating-a-new-gpg-key) —— Github 官方文档
* [**Github · Adding a GPG key to your GitHub account**](https://docs.github.com/cn/authentication/managing-commit-signature-verification/adding-a-gpg-key-to-your-github-account) —— Github 官方文档
* [**Github · gradle-maven-publish-plugin**](https://github.com/vanniktech/gradle-maven-publish-plugin) —— vanniktech 著
* [**Dokka · Using the Gradle plugin**](https://kotlin.github.io/dokka/1.7.10/user_guide/gradle/usage/) —— Dokka 官方文档
* [**GPG 入门教程**](http://www.ruanyifeng.com/blog/2013/07/gpg.html) —— 阮一峰 著
* [**PGPException: unknown public key algorithm encountered 问题**](https://blog.csdn.net/ThomasChant/article/details/118940537) —— Java侠 著