CODE

拥抱 Android Studio 之四：Maven 仓库使用与私有仓库搭建

2016-01-27 #ANDROID #EMBRACEANDROIDSTUDIO #GRADLE #MAVEN

目录

使用、创造和分享

笔者曾经不思量力的思考过 『是什么推动了互联网技术的快速发展？』 这种伟大的命题。结论是，除了摩尔定律之外， 技术经验的快速积累和广泛分享 ，也是重要的原因。

下一篇
Bugtags Web 2016-01-28 更新内容

上一篇
Bugtags Web 使用小技巧

有人戏称，『写 Java，首先要学会选包』，在这里不好评论对错。不过这句话里面，至少包含两层意思：首先 Java 有大量的现成的依赖包，不必要自己造轮子；其次，Java 的包存放较为集中，集成方式也方便。

笔者从事 Android 和 Java 开发以来，经历了几个阶段：

> 闭门造轮子 > 使用别人的轮子 > 开门造轮子 > 分享轮子

在使用、创造、分享轮子的过程中，maven 仓库的使用可谓必备技能。

相信各位使用 Android Studio，对于 `jcenter()` 、 `mavenCentral()` 等概念应该是司空见惯了。程序员要知其然，知其所以然。本篇将按照如下脉络介绍在 Android Studio 中 Maven 仓库相关的概念和应用。

- Maven 包
- Maven 仓库
- 发布包到本地仓库

- 发布包到 Bintray Jcenter 远程仓库
- 发布包到 Sonatype MavenCentral 远程仓库
- 搭建私有 Sonatype 仓库
- 搭建私有 Artifacotory 仓库

Maven 包（Package）

至于 Maven 是什么，请参考 Apache Maven。

对于 Android 开发者而言，只需要知道 Maven 是一种构建工具，Maven 包是由所谓 POM（Project Object Model）所定义的文件包格式即可。

Gradle 可以使用 Maven 包，而且大部分的 Android 能够使用的远程依赖包都是 Maven 包。

先来看一个托管在某仓库上的 Maven 包：Bugtags-Android-Lib 所包含的内容：

```
1  bugtags-lib-1.1.0-javadoc.jar//javadoc 文件
2  bugtags-lib-1.1.0-javadoc.jar.asc//javadoc 文件的签名
3  bugtags-lib-1.1.0-sources.jar//源码文件
4  bugtags-lib-1.1.0-sources.jar.asc//源码文件的签名
5  bugtags-lib-1.1.0.aar//Android Library 的主文件包
```

INTRO
BUGTAGS 远程配置功能介绍
2016-10-20

分类

- case (5)
- code (9)
- intro (10)
- news (3)
- skill (6)
- update (20)

归档

- February 2017 (1)
- November 2016 (2)
- October 2016 (3)
- September 2016 (1)
- August 2016 (2)

```
6  bugtags-lib-1.1.0.aar.asc//主文件包的签名
7  bugtags-lib-1.1.0.pom//包描述文件
8  bugtags-lib-1.1.0.pom.asc//描述文件的签名
```

对于一个合符规范的 Maven Package， pom 文件、aar（或者 jar） 文件 是必须的。

而 javadoc 文件、源码文件、签名文件都不是必要的，但是 某些公开仓库 （如 mavenCentral ）有此要求。

使用这个包的方式，相信大家已经很熟悉了：

```
1  dependencies {
2      compile 'com.bugtags.library:bugtags-lib:1.1.0'
3  }
```

POM 文件

一个 Maven Package，最重要就是 POM（Project Object Model）文件，这其实是一个 XML 文件，这里截取 Bugtags-Android-Lib POM 主要内容如下：

```
1  <?xml version="1.0" encoding="UTF-8"?>
```


- July 2016 (4)
- June 2016 (3)
- May 2016 (2)
- April 2016 (3)
- March 2016 (2)
- February 2016 (4)
- January 2016 (12)
- December 2015 (10)
- November 2015 (5)

标签

- Android (5)
- BTGLog (1)
- Beta (1)
- Bugtags (45)
- CSDN 大奖 (1)
- EmbraceAndroidStudio (5)

```
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 htt
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.bugtags.library</groupId>
  <artifactId>bugtags-lib</artifactId>
  <version>1.1.0</version>
  <packaging>aar</packaging>
  <dependencies>
    <dependency>
      <groupId>com.android.support</groupId>
      <artifactId>support-v4</artifactId>
      <version>19.0.0</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

- modelVersion: 从 mvn 2.x 起，这个数值都是4.0.0
- packaging：打包方式，aar 是 Android Libirary 的打包方式，常用的还有 jar
- dependency：声明依赖列表
-

包的唯一标示：

```
<!--包组 id，通常是发布者拥有的域名的反向，以免跟别人的重复-->
<groupId>com.bugtags.library</groupId>
```

```
<!--包 artifactId，不好意思我也不知道如何准确翻译，其实就是组以下应该有一个更
<artifactId>bugtags-lib</artifactId>
<!--包版本-->
<version>1.1.0</version>
```

其中三个字段与 Gradle 的依赖格式 `'com.bugtags.library:bugtags-lib:1.1.0'` 冒号分割的三段一一对应。这就解释了所谓的 `Gradle 兼容 Maven 包` 。

Maven 仓库

Maven 包集中存放的地方，就是 Maven 仓库。这些仓库，可以是放在本地，也可以放在某个远程服务器上。 可以是私有仓库，也可以是公开的。下面是笔者日常开发用的库列表：

```
mavenCentral();
jcenter()
maven {
    url 'file:///Users/my-user-name/Documents/Android/repo/'
}
maven {
    url 'http://192.168.99.100:8081/content/repositories/releases/
}
```

Android Studio Gradle 主要支持两个 Maven 中央库： `mavenCentral` 和 `jcenter` 。

- mavenCentral 是最早的 maven 中央仓库
- jcenter 是 Android Studio 0.8 版本起的默认 maven 中央仓库
- 第三个是笔者的本机的仓库
- 第四个是笔者部署在内网服务器的私有仓库

读者可能会发现两个问题：

- 为什么有了 mavenCentral，谷歌还切换到了 jcenter？
- `maven { url : xxx}` ，这种格式可以配置任何一个存在的仓库？

解释如下：

jcenter VS. mavenCentral

根据这篇博客，jcenter 具有如下优胜特点，使得谷歌进行切换：

- jcenter 是一个 mavenCentral 的超集，jcenter 还包含了其他 maven 包
- jcenter 具有更好的 cdn，默认支持 https，这个对于谷歌有巨大吸引力
- bintray（jcenter 的服务提供商）表示 jcenter 具有更好的性能

- 瑞士军刀 (1)
- 用例导出 (1)
- 用户反馈 (15)
- 知乎 (1)
- 移动测试 (43)
- 第三方转发 (1)
- 红包 (1)
- 自定义数据 (1)
- 视频介绍 (2)
- 远程配置 (2)
- 重复标签 (1)

标签云

Android BTGLog Beta Bugtags CSDN 大奖 EmbraceAndroidStudio Gradle Groovy Live Maven PHP android canary docs gitbook ndk plugin sendFeedback setUserData 产品测试 产品经理 企业版 使用手册 使用技巧 关键词 发展历程 后端 在线修复 在线标注 实时跟踪 小技巧 插件 日志工具 春节 更新 最具成长潜力奖 活动 测试任务 测试用例 热修复 瑞士军刀 用例导出

- 有数据表明 bintray jcenter 占用更少的本地缓存空间
- 更好的交互界面，可以从 jcenter 向 mavenCentral 同步包（下面两个平台的使用教程将会证实这一点）

> *笔者亲测，在 bintray 上发布包到 jcenter 在易用性上的确比 在 sonatype 发布到到 mavenCentral 要好得多。*

使用符合规范的 maven 仓库

没错，你可以通过 `maven { url : xxx }` 使用任何一个符合 maven 规范的仓库。

-

存在本地的

```
maven {
    url 'file:///Users/my-user-name/Documents/Android/repo/'
}
```

-

存在内网服务器的

```
maven {
    url 'http://192.168.99.100:8081/content/repositories/releases/
}
```

-

存在某个远程服务器的

```
maven {
        url 'https://raw.githubusercontent.com/liaohuqiu/umeng-libs
}
```

> *此仓库由 liaohuqiu 同学为方便大家使用友盟开发者工具，把相应的包做成了符合规范的 Maven 包，托管在 github 项目中。*

发布 Maven 包

使用 maven 包相信已经很清楚了，让我们更进一步。

当我们在日常开发实践中，积累了一些公共库，想要固定下来，被自己或者别人方便的使用，就需要发布 maven 包。

一个符合规范的 maven 包至少包含 pom 文件和主文件包。难道这些都要手动编写和创建么？

答案是：有了 gradle 插件，你只需要干很少的事儿。

全局设定

下面以发布这系列包为示例：

- groupId: com.as-gradle.demo //改成你的 groupId
- artifcatId: x //artifactId 会有些变化，这里先用 `x` 代替，下面会修改。
- version: 1.0.0

也就是 `'com.as-gradle.demo:x:1.0.0'`

> *读者要进行练习的时候，最好改一下你的 `groupId` ，否则可能会发布失败*

下面使用到的示例工程已经放在了 github 上。

为了后面使用方便，首先在工程的项目 `gradle.properties` 中定义一些属性，**这些属性，主要是用生成 POM 文件，将会在通篇文章中用到**：

```
# 包信息
PROJ_GROUP=com.as-gradle.demo
PROJ_VERSION=1.0.0

# 项目的描述
PROJ_WEBSITEURL=https://bugtags.com
PROJ_ISSUETRACKERURL=https://github.com/bugtags/Bugtags-Android/is
PROJ_VCSURL=https://github.com/bugtags/Bugtags-Android.git
PROJ_DESCRIPTION=Simple and effective bug & crash reporting tool

# Licence信息
PROJ_LICENCE_NAME=The Apache Software License, Version 2.0
PROJ_LICENCE_URL=http://www.apache.org/licenses/LICENSE-2.0.txt
PROJ_LICENCE_DEST=repo

# Developer 信息
DEVELOPER_ID=your-dev-id
DEVELOPER_NAME=your-dev-name
DEVELOPER_EMAIL=your-email@your-mailbox.com
```

发布包到本地仓库

- 创建一个 module： `localrepo`

- 将本地某个路径设置为仓库根目录：
`/Users/your-user-name/Documents/Android/repo/` （ Mac 下 ）

这里使用了一个叫做 `your-user-name` 的用户下的某个目录，请读者**自行替换**成自己的登录用户名。

- 为了优雅，在 localrepo 这个 module 的 `gradle.properties` 定义属性：

```
PROJ_NAME=localrepo
PROJ_ARTIFACTID=localrepo
PROJ_POM_NAME=Local Repository

LOCAL_REPO_URL=file:///Users/your-user-name/Documents/Android/repo/
```

- 在 module 中应用和配置 maven plugin：

```
apply plugin: 'maven'
```

```
2
3  uploadArchives {
4          repositories.mavenDeployer {
5          repository(url: LOCAL_REPO_URL)
6          pom.groupId = PROJ_GROUP
7          pom.artifactId = PROJ_ARTIFACTID
8          pom.version = PROJ_VERSION      }
9  }
```

-

在控制台运行：

```
1  $ ./gradlew -p localrepo clean build uploadArchives --info
```

-

一切顺利的话，你的第一个本地包已经发布到设定的目录的本地仓库了：

```
1  |       ├── com
2  |   ├── as-gradle
3  |   |   └── demo
4  |   |       └── localrepo
5  |   |           ├── 1.0.0
6  |   |           |   ├── localrepo-1.0.0.aar
```

```
7  |   |           |   ├── localrepo-1.0.0.aar.md5
8  |   |           |   ├── localrepo-1.0.0.aar.sha1
9  |   |           |   ├── localrepo-1.0.0.pom
10 |   |           |   ├── localrepo-1.0.0.pom.md5
11 |   |           |   └── localrepo-1.0.0.pom.sha1
12 |   |           ├── maven-metadata.xml
13 |   |           ├── maven-metadata.xml.md5
14 |   |           └── maven-metadata.xml.sha1
```

使用本地包（两个疑问向读者请教）

-

要使用这个包，首先在项目的 build.gradle 中添加这个本地仓库：

```
1  allprojects {
2      repositories {
3          jcenter()
4
5          maven{
6              url 'file:///Users/your-user-name/Documents/Android/repo
7          }
8      }
9  }
```

在某个 module（如 demo 项目中的 app）的 build.gradle 中添加依赖：

```
1  compile 'com.as-gradle.demo:localrepo:1.0.0@aar'
```

> *这里有两个奇怪的地方，笔者也没有深入研究，初步猜测是 Android Studio 的 Bug，知道答案的读者请到我博客文章下留言赐教：*

* 依赖末尾一般都需要加一个`@aar`，在某些版本的 Android Studio，又不需要，这是为
* 另外，如果本地包本身使用了了远程的依赖，也需要在使用本地包的时候，一并加上，否则会

> *想要让更多的人使用到你的劳动成果，你就需要把 Maven 包放在一个别人有权访问的远程仓库上，而不是本机，接下来要介绍发布 Maven 到 jcenter 仓库和 mavenCentral 仓库。因为前者的使用简单，本着『先易后难，快速出成效』的原则，我先介绍 jcenter 的上传。*

发布包到 Bintray jcenter 远程仓库

简介

jcenter 是由 bintray 提供的 maven 中央库托管服务，bintray 又是 jfrog 公司的一款产品。jfrog 是一个商业公司，通过提供高级服务盈利，又为普通开发者提供了足够用的免费基础功能（截止至2016-01-24），笔者较为推崇这种开发者服务的商业模式。

> *引用一张图来表述 bintray 的工作方式*

how-bintray-works

how-bintray-works

图片来源，http://inthecheesefactory.com/

使用 jcenter 需要在 bintray 上注册账号，在本地进行加密签名配置，下面开始介绍。

1. 注册账号

- 登陆 jcenter 首页
- sigin -> signup，填写表单，注意 username 这一栏是后面的 bintray 私有库的后缀，要慎重选择。

2. 创建 GPG 签名

> *前方高能预警：比较繁琐，切勿半途放弃*

前面介绍过，可以把 bintray 的包同步到 mavenCentral，而后者需要对包文件进行签名，签名和验证过程需要用到一个叫做 GPG 的工具产生的公钥和私钥。这里有适合多个平台的 GPG 程序，下面只介绍 OSX 平台。

> *这种工具大概的意义是产生公钥和私钥，把公钥发送到公开的服务器，私钥用来产生包文件签名。包的使用者在拿到包文件之后，通过公钥来验证文件的有效性，运行具体原理参考这里。*

- 下载 gpgtool，安装
-

检测安装成功，在命令行运行

```
1  $ gpg --version
2  gpg (GnuPG/MacGPG2) 2.0.28
3  libgcrypt 1.6.3
```

有类似的输出，就是正常安装了

-

产生证书，运行命令，按照提示输入

```
1  $ gpg --gen-key
```

-

检查结果

```
1  $ gpg --list-keys
```

找到刚才创建证书的时候，输入的相关信息那三行，例如：

```
1  pub   2048R/2E686B39 2015-06-02
2  uid       [ultimate] Your Name <your-email@your-mailbox.com>
3  sub   2048R/your-sub-key-id 2015-06-02
```

-

上传公钥到服务器，找到你的 pub 的那一行，2048R/后的那一串八位字符串，如上面的：2E686B39，就是公钥 ID

```
1  $ gpg --keyserver hkp://pool.sks-keyservers.net --send-keys your-pu
```

- 输出公钥和私钥成文件

```
$ gpg -a --export your-email@your-mailbox.com > public_key_sender.a
$ gpg -a --export-secret-key your-email@your-mailbox.com > private_
```

- 配置本地 gradle 运行环境的属性，位于 `~/.gradle/gradle.properties` ，添加内容：

```
signing.keyId=your-public-key-id
signing.password=your-gpg-password
signing.secretKeyRingFile=/Users/your-user-name/.gnupg/secring.gpg
```

- bintray 本身可以通过在 profile->GPG Sining 中配置 public key 和 private key 来自动对上传的文件进行签名，在下图中，对应填入 `public_key_sender.asc`

与 `private_key_sender.asc` 的内容即可。

gpg-signing

gpg-signing

-

设置 bintray maven 包自动签名
选取 maven 仓库首页，进入 edit：

auto-sign

auto-sign

最下面有两个选项：

```
1  GPG sign uploaded files using Bintray's public/private key pair.
2  GPG Sign uploaded files automatically
```

因为咱们是希望使用自己的 key，所以勾选第二个。

3. 创建 bintray 项目

首页-> maven -> add new package，填入对应的信息，其中 name 是在下面 bintray gradle 插件上传的时候，使用的项目名称，例如：bintryaar，这是要上传一个 Android Library，上传纯 Java 包的方式有 点点不一样 ，下面有介绍。

create-package

create-package

4. 配置插件

bintray 官方在 github 上托管了 bintray-examples，方便用户使用 gradle 上传包。

因为这里要上传的是 aar 格式的包，所以，具体是参考 gradle-aar-example 例子，然而例子有一些地方没有更新，请注意下面的描述。

- 在项目中创建 local.properties 来配置 bintray 登陆信息以及 gpg 证书密码

```
1 bintray.user=your-bintray-user
2 bintray.apikey=your-bintray-apikey
3 bintray.gpg.password=your-gpg-password
```

其中 your-bintray-user 就是 bintray 右上角显示的用户名称，your-bintray-apikey 在 profile->API Key 可以找到，your-gpg-password 则是创建 gpg 证书的时候的密码

- 在项目的 build.gradle 配置 buildscript 的 classpath

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.3.0'
        //下面两个包是用于上传的插件
        classpath 'com.jfrog.bintray.gradle:gradle-bintray-plugin
        classpath 'com.github.dcendents:android-maven-gradle-plug
    }
}
```

-

在 module 的 gradle.properties 文件中定义属性

```
PROJ_NAME=bintrayaar
PROJ_ARTIFACTID=bintrayaar
PROJ_POM_NAME=Bintray Aar Repository
```

- 在 module 的 build.gradle 中使用插件

```
apply plugin: 'com.github.dcendents.android-maven'
apply plugin: 'com.jfrog.bintray'
```

- 为了build.gradle 文件干净，笔者创建了一个名为 bintray.gradle 的文件配置插件信息，请参考这个文件。
关键点：

```
artifacts {
    archives javadocJar
    archives sourcesJar
}
```

是为了同时生成 javadoc.jar 和 sources.jar 文件

-

build，上传

```
1  $ ./gradlew -p bintrayrepo/ clean build bintrayUpload --info
```

如果一切顺利，你会在控制台看到多个文件上传成功的标输出

-

踩坑实录

-

  HTTP/1.1 401 Unauthorized

  apikey 或者 user 填写错误

-

  HTTP/1.1 409 Conflict

  包的该版本已经存在，需要在 bintray 管理界面上删除该版本后才可以再次上传

- 想让 sources.jar 或者 javadoc.jar 为空

```
task sourcesJar(type: Jar) {
                        classifier = 'sources'
                from sourceSets.main.java.srcDirs
                exclude '**'
}
```

5. 上传 Jar 包

- 在上传 jar 的时候，使用的插件有些区别

```
apply plugin: 'maven-publish'
apply plugin: 'com.jfrog.bintray'
```

-

在生成符合规定的 pom 文件的时候，要调用 groovy 的API，具体请参考这个文件

6. 通过私有仓库的方式引用

至此，刚才上传的两个库，已经可以通过如下方式引用了

```
allprojects {
    repositories {
            maven {
                url 'https://dl.bintray.com/freefaces/maven'//这个址
                }
    }
}
```

```
compile 'com.as-gradle.demo:bintrayaar:1.0.0'
compile 'com.as-gradle.demo:bintrayjar:1.0.0'
```

但是你也发现了，包的用户需要添加一个额外的 maven 仓库。作为一个以用户价值为先的库的开发者，那当然不希望用户麻烦的。那就需要把这个私有的库，推送到 jcenter 上去。

7. 推送到 jcenter

在 bintray 的 maven 库的界面，有 `add to jcenter`

add-to-jcenter

add-to-jcenter

点击之后，会进入一个消息页面，写或者不写都可以。提交，等待回复即可。

记住,包必须满足如下条件：

- 包含 sources.jar 和 javadoc.jar 文件
- 必须是 maven 包

> *bintray 的消息系统有些 bug，假设你的包提交申请被驳回，你修改之后再提交申请，可能没有人回复你。请**不要傻等**。直接找页面右侧的 Feedback，这个他们是很快有人回答的。*

成功了之后，会出现如下的标记：

inclusion-succeed

inclusion-succeed

你可以在 jcenter 服务器上看到你的包了

8. 推送到 mavenCentral

在包管理页面，可以找到推送到 mavenCentral 功能，
sync-mavencentral

sync-mavencentral

一个包要从 bintray 推送到 jcenter，有几个前提：

- 已经推送到了 jcenter[已完成]
- 每个文件都有对应的签名[已完成]
- 有 sonatye 账户[未完成]
- maven 仓库经过审核[未完成]

点击 Sync 之后，一段时间之后，右边的 Sync Status 会反馈出结果。

当然了，现在咱还干不了这个，因为还有两个条件没准备好。那咱们就进入 mavenCentral 的条件准备。

发布包到 Sonatype MavenCentral 远程仓库

1. 注册 sonatye 账户

进入 sonatype issue 页面，注册账号。

2. 创建 issue

登陆之后，顶部有按钮，Created，下面是关键的条目

- Project: `Community Support - Open Source Project Repository Hosting (OSSRH)`
- Issue Type: New Project
- Group Id：就是你的包的 groupId

其他的都认真填写。确认之后，大概两个工作日， Issue 会变成 resolved 状态，就可以发布你的包了。有了这两部，其实就可以从 bintray 上直接反向推到 mavenCentral ，而不需要走下面的步骤了，但是我还是简略介绍一下下面的步骤。如果很感兴趣详情，可以参考 trinea 的介绍。

3. 上传包

也有方便的 gradle 插件帮助我们进行传送，可以参考 chrisbanes/gradle-mvn-push 项目。配置好插件，上传。

4. 发布包

登陆 oss sonatype，登陆，选择左边栏里的 `Staging Repositories`，然后点 Close 按钮，sonatype 会做响应的 validation，通过的话，就可以点 Release 发布啦，如果不通过，就检查问题，先 Drop，并重新做 Staging 发布。

5. 检查包

在 https://oss.sonatype.org/content/repositories/releases 可以看到你发布的包。

6. 为何如此简略

因为这个过程真的很繁琐，ui 也不友好，在体验了 bintray 的便捷和友好，并发现 bintray 可以反向推送到 mavenCentral 之后，我就再也不想使用 sonatype 了。无奈，贪嗔痴是人类天性。

搭建私服

由于"你懂得"的原因，在国内访问 jcenter，总是偶尔不稳定，经常会出现诸如 peer not found 这种错误。为了保证用户的稳定使用库，那就要考虑搭建放

在自己服务器上的私有仓库。

Sonatype 和 bintray 都提供了可供自己部署的 maven 库管理软件。Sonatype 提供了免费的 sonatype/nexus，bintray 提供了免费的 artifactory-oss。

为了部署简便，笔者使用了 docker 进行这两个私服搭建。对于 docker 是什么，怎么用，并不是系列文章的重点。有兴趣可以自行学习。入门文档在此。

搭建私有 Sonatype 仓库

-

下载安装 docker 镜像

```
1  $ docker pull sonatype/nexus
```

-

运行镜像

```
$ docker run -d -p 8081:8081 --name nexus sonatype/nexus:oss
```

- 访问服务器

因为的 docker-machine ip 是： `192.168.99.100` ，于是可以通过在浏览器访问 `http://192.168.99.100:8081/` 这个 URL 来访问 sonatype 私服。

你会发现这个界面跟你在 `https://oss.sonatype.org/` 看到的几乎一样。

默认账户密码是：

```
admin
admin123
```

- 设置仓库

点击左侧 repository，会出现 repository 的列表，把其中的 `Releases` 的 `Configutation->Access Setting-> Deploy Polocy` 设置成 `Allow Redeploy` 使得可以重复发布包。

- 配置和使用插件

我还是使用了 chrisbanes/gradle-mvn-push 插件，稍微改动了一下字段的值，主要改动是环境配置，如账号密码，repository URL 等，具体请参考这里 mvn-push。

- 关键设置

要在 `gradle.properties` 中设置：

```
1  PROJ_NAME=sonatyaar
2  PROJ_ARTIFACTID=sonatyaar
3  PROJ_POM_NAME=Sonatye Aar Repository
4  POM_PACKAGING=aar
5  RELEASE_REPOSITORY_URL=http://192.168.99.100:8081/content/repositor
6  SNAPSHOT_REPOSITORY_URL=http://192.168.99.100:8081/content/reposit
```

- 查看

上传成功之后，就可以在浏览器的 `http://192.168.99.100:8081/content/repositories/releases` 看到这个包。并可引用了。

-

错误

上传的时候，返回400，可能是 `Configutation->Access Setting-> Deploy Polocy` 没设置好；返回401，可能是账号密码错误。

搭建私有 Artifactory 仓库

bintray 其实提供了多个私有部署仓库的版本，分别是：

```
1  Artifactory OSS
2  Artifactory Pro
3  Artifactory Pro Registry
```

按名字来看，后两者可能是收费的，这里就只介绍 `Artifactory OSS` ，依然是使用 docker 进行部署运行。更详细的使用手册，参考 Running with Docker。

- 下载镜像（截止至2016-01-27，最新版本是4.4.1）

```
$ docker pull jfrog-docker-reg2.bintray.io/jfrog/artifactory-oss:4
```

- 运行镜像

```
$ docker run -d -p 8080:8081 jfrog-docker-reg2.bintray.io/jfrog/ar
```

- 在浏览器中访问 http://192.168.99.100:8080/ ，默认账号密码是：

```
admin
password
```

笔者写到这，发现这个篇幅已经太长了。现在的读者，其实也没有太多耐心看长篇大论，所以考虑将更详细的私服的部署，放在一篇独立的博客中讲解。

kevinho/gradle-maven-complete

为了方便读者使用 gradle 将 aar、jar包推送到 jcenter 和 mavenCentral，笔者决定将本文所使用的 sample 项目，分离出一个独立的 github 项目：kevinho/gradle-maven-complete，里面包含如下范例：

- localrepo：本地仓库推送
- bintrayaar：bintray 的 aar 包推送
- bintrayjar：bintray 的 jar 包推送
- sonatypearr：mavenCentral 的 aar 包推送

基本上覆盖到了主流的场景了，希望我这个小轮子，也能帮助大家，喜欢记得 star 哦！

总结

这一篇，笔者结合实例讲解了 maven 仓库相关的知识，以及将 maven 包通过 gradle 插件上传本地仓库、bintray jcenter、sonatype mavenCentral，还简要介绍了 sonatype 和 artifactory 私服的 docker 搭建。或许你已经蠢蠢欲动了，那就赶紧打开你的电脑，把你的轮子，用 maven 武装起来吧！下一篇会介绍 gradle 插件的编写以及发布使用，敬请期待！

参考文献

- inthecheesefactory.com
- trinea.cn
- wikipedia
- apache-maven
- sonatype-central
- bintray-manual

系列导读

本文是笔者《拥抱 Android Studio》系列第四篇，其他篇请点击：

拥抱 Android Studio 之一：从 ADT 到 Android Studio

拥抱 Android Studio 之二：Android Studio 与 Gradle 深入

拥抱 Android Studio 之三：溯源，Groovy 与 Gradle 基础

拥抱 Android Studio 之四：Maven 公共仓库使用与私有仓库搭建

拥抱 Android Studio 之五：Gradle 插件使用与开发

> *有问题？在文章下留言或者加 qq 群：453503476，希望能帮到你。*

番外

笔者 kvh 在开发和运营 bugtags.com，这是一款移动时代首选的 bug 管理系统，能够极大的提升 app 开发者的测试效率，欢迎使用、转发推荐。

笔者目前关注点在于移动 SDK 研发，后端服务设计和实现。

我们团队长期求 PHP 后端研发，有兴趣请加下面公众号勾搭：

bugtags-logo

bugtags-logo

相关文章推荐

- 拥抱 Android Studio 之五：Gradle 插件开发
- 拥抱 Android Studio 之三：溯源，Groovy 与 Gradle 基础
- 拥抱 Android Studio 之二：Android Studio 与 Gradle 深入