**使用指南**

**使用M****aven仓库的优势**

**Q1：**为什么要使用maven远程仓库？

**A1：**有以下优势：

1. **方便管理。**便于对项目库进行保存和整合；团队可通过简单的依赖拉取共享代码；

1. **节省时间。**从仓库中拉取下来的库不需要编译，而直接导入代码则每次构建都需要；
2. **代码一致性。**相对于直接导入源码，maven拉取的方式避免了因个人失误而对库进行了修改，导致产生多份不同的代码；
3. **依赖聚合。**借助pom文件，多个版本库可以实现自动关联依赖，从而能够提供十分简洁的使用方式，如需要引入多个库，使用maven拉取只需要添加一条依赖即可；

**Q2：**maven仓库的使用门槛？

**A2：**要将项目库发布到maven仓库，需要对maven插件有一定的了解，并且不同版本的maven插件在使用上也有差异。但是现在，提供了足够简单的Gradle脚本来实现将自己的库发布到maven仓库。这些Gradle脚本适用于大部分场景，并且配置比较灵活。

**从maven仓库拉取依赖**

1. **为工程添加maven仓库信息**

由于Gradle 7.x版本开始，新增了对仓库的管理配置，因此，脚本中不集成仓库的添加代码，而是改为手动集成。

Gradle6.x或更低版本：

在根工程的build.gradle中添加：

allprojects {

repositories {

maven {

url 'maven仓库地址'

credentials {

username ='用户名称'

password = '用户密码'

}

//允许不受信任的http方式

allowInsecureProtocol true

}

}

}

Gradle7.x或更高版本：

在根工程的settings.gradle中添加：

dependencyResolutionManagement {

//如果是RepositoriesMode.PREFER\_PROJECT，应使用低版本Gradle的配置方法

repositoriesMode.set(RepositoriesMode.FAIL\_ON\_PROJECT\_REPOS)

repositories {

maven {

url 'maven仓库地址'

credentials {

username = '用户名称'

password = '用户密码''

}

allowInsecureProtocol true

}

}

}

1. **添加远程依赖**

在目标module的build.gradle中，添加需要拉取的依赖。示例：

dependencies {

implementation ‘x.x:x:1.2.3’

}

**发布到maven仓库**

1. **为工程添加maven仓库信息**

基本同[拉取依赖](#从maven仓库拉取依赖)步骤，也可以在build.gradle的发布参数配置中再单独指定这些配置。

1. **添加Gradle脚本**

添加仓库信息后，在需要打包发布的module的build.gradle中添加脚本。

但是，对于不同的library类型或编程语言，使用的脚本是不一样的。具体参考同级目录下“Gradle脚本”目录内README.md。

1. **发布到maven仓库**

具体参考”Gradle脚本“目录内的README.md。

**注意事项**

1. snapshots仓库要求版本名末尾必须加上"-SNAPSHOT"后缀，releases仓库则不需要；
2. 必须谨慎删除或者重复发布远程仓库中的版本，可能会导致无法拉取。如果出现此类问题，前往系统用户目录/.gradle/caches中，将这些从远程仓库拉取的版本的所有文件全部删除，然后再重新尝试拉取。一般地，releases仓库严禁删除版本库，防止已使用它们的项目无法拉取到依赖。