[Java Gradle入门指南之依赖管理（添加依赖、仓库、版本冲突）](https://www.cnblogs.com/gzdaijie/p/5296624.html)

#### 目录

[**1.添加依赖包名**](http://www.cnblogs.com/gzdaijie/p/5296624.html#_caption_0)  
1.1 依赖类型  
1.2 声明依赖  
1.3 添加java依赖  
1.4 查找依赖包名  
1.5 完整的例子  
[**2.添加依赖仓库**](http://www.cnblogs.com/gzdaijie/p/5296624.html#_caption_1)  
[**3.依赖常见问题**](http://www.cnblogs.com/gzdaijie/p/5296624.html#_caption_2)  
3.1 依赖传递性  
3.2 版本冲突  
3.3 动态依赖  
3.4 更多设置

博客逐步迁移至 [极客兔兔的小站](http://geektutu.com/)

    开发任何软件，如何管理依赖是一道绕不过去的坎，软件开发过程中，我们往往会使用这样那样的第三方库，这个时候，一个好的依赖管理就显得尤为重要了。作为一个自动构建工作，Gradle对依赖管理有着很好的支持。  
    通常我们使用IDE（Eclipse、IDEA、Android Studio）开发Java项目，IDE自动为我们创建了Gradle文件，添加依赖也不过简单的几行代码，这篇随笔将从逐步解释Gradle的依赖管理方法，希望对大家有所帮助。  
    如有错误，请不吝指出，非常感谢！如果本文对你有帮助，右下角点个**推荐**吧~

## 1.添加依赖包名

### 1.1 依赖类型

* 常见的依赖包含两种类型。

(1) 一类是项目中所需要的库，包括本地/仓库中的文件和其他项目文件（例如一个多项目工程，一个项目依赖于另一个项目） 

(2) 一类是实现自动化编译、部署等所需的库，包含Gradle的API和Groovy编写的Task、Plugin等，这一类依赖在前2篇随笔有提到和使用

### 1.2 声明依赖

* 声明依赖使用下面的闭包

dependencies {

<configuration name> <dependencies>

}

### 1.3 添加java依赖

* 在这里，我们以构建java项目为例，构建java项目首先需要应用java插件，插件的使用可参考上一篇随笔[Java Gradle之插件管理](http://www.cnblogs.com/gzdaijie/p/5290157.html)
* java插件针对不同操作，将依赖分为10类（详见 [java plugin 45.5](https://docs.gradle.org/current/userguide/java_plugin.html)），下面介绍常用的5类

(1) **compile**：源代码（src/main/java）编译时的依赖，最常用  
(2) **runtime**：源代码（src/main/java）执行时依赖  
(3) **testCompile**：测试代码（src/main/test）编译时的依赖  
(4) **testRuntime**：测试代码（src/main/java）执行时的依赖

(5) **archives**：项目打包（e.g.jar）时的依赖

* 通常，一个JAR依赖需包含JAR文件组（group/命名空间）、JAR文件名（name）、JAR文件版本（version），特殊情况下还可指定JDK版本。添加依赖可以有以下方式：

/\* 单个依赖 \*/

compile group:'log4j', name:'log4j', version:'1.2.17'

// 简写 => compile 'log4j:log4j:1.2.17'

/\* 以数组形式添加多个依赖\*/

compile 'joda-time:joda-time:2.9.2', 'log4j:log4j:1.2.17'

/\* 闭包形式，以添加额外配置\*/

compile (group:'log4j', name:'log4j', version:'1.2.17'){

// ... 额外配置

}

/\* 等价于 \*/

compile ('log4j:log4j:1.2.17'){

// ... 额外配置

}

### 1.4 查找依赖包名

* 点击[maven网站](http://mvnrepository.com/)
* 搜索需要导入的包，例如gson，点击对应版本，例如2.6.2
* 选择gradle，将会出现'com.google.code.gson:gson:2.6.2'
* 复制 & 粘贴

### 1.5 完整的例子

// sourceCompatibility = 1.8为java版本，默认为当前JVM版本

apply plugin: 'java'

sourceCompatibility = 1.8

version = '1.0'

repositories {

mavenCentral()

}

dependencies {

compile 'org.springframework:spring-context:4.2.1.RELEASE'

compile 'log4j:log4j:1.2.17'

}

* repositories{ ... } 是放置这些包的仓库，接下来介绍
* sourceCompatibility、version只是java plugin的部分属性，更多请看 [java plugin 45.8](https://docs.gradle.org/current/userguide/java_plugin.html)

## 2.添加依赖仓库

    你可能会疑惑，声明了这些依赖，这些依赖是在哪里找到的呢？repositories定义了下载依赖的仓库

/\* Maven Central respoitory \*/

repositories {

mavenCentral()

}

/\* Maven JCenter respoitory \*/

repositories {

jcenter()

}

/\* Maven local respoitory \*/

/\* 本地仓库是之前下载的依赖，缓存在本地磁盘\*/

repositories {

mavenLocal()

}

* 不需要记住仓库的地址，直接使用即可，多个仓库可以同时用，通常我们会将远程仓库与本地仓库一起使用，因为缓存在本地磁盘上的文件速度更快，不需要重复下载。
* 关于jcenter和 mavenCentral的区别，推荐看[stackoverflow的回答](http://stackoverflow.com/questions/24852219/android-buildscript-repositories-jcenter-vs-mavencentral)
* 当然，国外的仓库在国内使用速度可能会比较慢，Gradle支持自定义地址，例如公司的仓库地址、国内仓库镜像地址等。

repositories {

mavenLocal()

/\* 指定本地仓库地址 \*/

maven { url "file://E:/githubrepo/releases" }

/\* 指定的远程仓库 \*/

maven { url "http://mvnrepository.com/" }

/\*

\* 公司仓库，可能需要验证

\* 不推荐直接将用户名密码写在build.gradle中

\* 可以写在~/.gradle/gradle.properties中，再使用

\*/

maven {

url "<you\_company\_resp\_url>"

credentials {

username 'your\_username'

password 'your\_password'

}

}

// 支持ivy仓库

ivy { url "<ivy\_resp\_url>" }

}

* 有时，我们需要调用自己编译实现的\*.jar，我们也可以将包含这些文件的文件夹（不同于mavenLocal）映射为一个仓库，但一般不推荐这样做

repositories {

flatDir { dirs 'libs1/java/...','libs2' }

}

## 3.依赖常见问题

### 3.1 依赖传递性

* 很多库依赖于其他库，例如a.jar依赖b.jar，在Gradle中，只需添加a.jar即可，Gradle将自动把a依赖的所有库全部下载。
* 但是，有时你并不想让Gradle自动去做这件事情，比如你希望明明白白地知道添加哪些库，可以配置transitive实现，编译时报错，你就可以知道进一步需添加哪些库。

dependencies {

// transitive 属性默认为 true

compile group:'log4j',name:'log4j',version:'1.2.17',transitive:false

}

* 另一种情况是，依赖传递可能会导致版本冲突，即依赖传递下载的库可能与项目依赖的另一个库版本冲突，这种情况下可以排除一些库，而下载其他所有的依赖库，即选择性排除。

dependencies {

compile ('commons-httpclient:commons-httpclient:3.1'){

exclude group:'commons-codec' //排除该group的依赖

// exclude group:'commons-codec',module:'commons-codec'

// group是必选项，module可选

}

}

### 3.2 版本冲突

* 版本冲突时十分常见的，比如下面的例子

// 库 a 传递性依赖库 b-1.2，与添加的b-1.1冲突

dependencies {

compile 'a:a:1.0'

compile 'b:b:1.1'

}

* Gradle解决冲突有以下几种方式

(1) 最近版本策略（默认）：上例将忽略b-1.1，而下载b-1.2   
(2) 冲突失败策略：发生冲突时，编译失败（有些新版本库并不兼容之前的，因此这个库可以让开发者主动作出选择）

(3) 强制指定版本策略：发生冲突时，使用开发者指定的版本

/\* 冲突失败策略设置\*/

configurations.all {

resolutionStrategy { failOnVersionConflict() }

}

/\* 强制指定版本策略设置\*/

dependencies {

compile group:'b',name:'b',version:'1.1',force:true

}

### 3.3 动态依赖

* 动态依赖增加了库版本依赖时的灵活性

dependencies {

/\* 选择1以上任意一个版本，这使发生版本冲突的几率变小\*/

compile group:'b',name:'b',version:'1.+'

/\* 选择最新的版本，避免直接指定版本号 \*/

compile group:'a',name:'a',version:'latest.integration'

}

### 3.4 更多设置

* 指定库文件类型

// ext 默认jar，可选属性为war、zip

compile group:'b',name:'b',version:'1.1',ext:'war'

* 使用分类器（classifiers）

// 例如提供了2种包，a-1.0-dev.war, a-1.0-dev.jar

compile group:'b',name:'b',version:'1.1',classifier:'dev',ext:'war'

* 替换传递依赖的版本

compile group:'a',name:'a',version:'l.0' {

dependencies 'b:b:1.1'

}

* 常用命令

(1) 查看所有依赖库

gradle dependencies

(2) 查看指定配置（详见 1.3）的依赖库

gradle dependencies -configuration <configuration>

例 gradle dependencies -configuration compile => 查看编译时依赖

例 gradle dependencies -configuration runtime => 查看运行时依赖