**Gradle学习笔记**

Gradle与Maven类似，是一种构建工具,它可以帮你管理项目中的差异,依赖,编译,打包,部署，但又比maven更加灵活，因为maven的xml格式在项目复杂时很难维护和管理，并且不能自定义打包脚本

Gradle是基于Groovy语言的，当Groovy脚本在执行时，Groovy会将其编译成Java类字节码，然后通过JVM来执行该Java类

**Groovy与Java比较**

- Groovy完全兼容Java语法

- 句尾分号可选

- 类、方法默认为public

- 编译器给属性自动添加getter/setter

- 属性可以直接用点号获取

- 最后一个表达式的值会被作为返回值(可以不写return)

- ==等于equals()，并且不会抛出npe

- 弱类型

- 随时随地assert

- 方法括号可选(如println 1)

- 单引号：字符串

- 双引号：可以使用${}插入变量

- 三引号：可以换行

- range：range = 5..8 //5到8

- list添加元素：<<

- map添加元素：map.xxx=xxx (map['xxx']=xxx)

- 闭包

**这简直就是python啊**

**项目和任务**

在Gradle的配置文件中有项目和任务的概念

每个Gradle配置由一个项目构成，一个项目中由一个或多个任务构成

task代表着Gradle构建过程中可执行的最小单元。例如当构建一个组件时，可能需要先编译、打包、然后再生成文档或者发布等，这其中的每个步骤都可以定义成一个task

先看如下配置：

```

group 'com.gradle.study'

version '1.0-SNAPSHOT'

apply plugin: 'java'

apply plugin: 'war'

sourceCompatibility = 1.8

repositories {

mavenCentral()

}

dependencies {

testCompile group: 'junit', name: 'junit', version: '4.11'

}

```

- group version概念和maven类似

- apply是指引用哪些插件

- sourceCompatibility置顶编译版本

- repositories指定jar仓库

- dependencies：所依赖的包

可以看到上面所有概念都是基于项目的，并没有我们自定义的任务配置

当我们建立一个非web gradle时，是没有java src目录的，这时我们可以在配置中自己写一个任务来创建目录

```

def createDir = {

path ->

File dir = new File(path);

if(!dir.exists()){

dir.mkdirs();

}

}

task makeJavaDir() {

def paths = ['src/main/java','src/main/resources','src/test/java'];

doFirst{

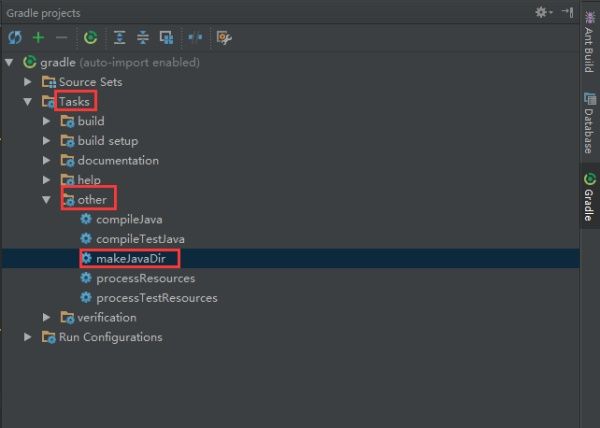
paths.forEach(createDir);

}

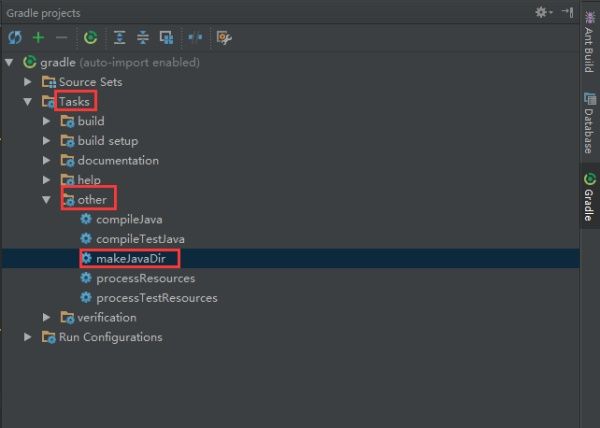
}

```

其中doFirst表示里面的行为放在当前task的前面运行，相对的还有doLast



然后在idea的右边运行我们写好的任务，之后可以看到目录中有了我们想要的结构



**构建生命周期**

**1.初始化阶段**

读取根工程中setting.gradle中的include信息，决定有哪几个project加入构建

**2.配置阶段**

执行所有工程的build.gradle脚本，配置project对象，并创建、配置task及相关信息

**3.运行阶段**

根据gradle命令传递过来的task名称，执行相关依赖任务

**依赖管理**

gradle通过group、name（id）、version唯一确定某jar包，这点和maven类似

可以看到上文中代码：

```

repositories {

mavenCentral()

}

dependencies {

testCompile group: 'junit', name: 'junit', version: '4.11'

}

```

repositories意味着仓库地址，这里用的是maven原仓库，也可以加上mavenLocal()本地仓库，当然也可以设置成阿里云仓库

dependencies代表所依赖的类，这里只有一个junit，并且只是在testCompile阶段才起作用，此处写明了group，name，version，也可以通过冒号相连，写成

testCompile 'junit:junit:4.11'

**解决版本冲突**

在maven中遇到版本冲突时，需要打印依赖树，然后自己手动去找哪个依赖中带有冲突的版本包，然后再通过移除解决冲突

在gradle构建中，默认依赖最高版本的jar包，这样就方便多了

如果我们想修改默认策略，首先需要设置当版本冲突时，构建失败，如下：

```

configurations.all {

resolutionStrategy {

failOnVersionConflict()

}

}

```

然后有两种方式修改依赖版本：

1.排除传递性依赖：exclude group:xxx module:xxx (module就是name)

2.强制版本：在上面的failOnVersionConflict方法下写上force版本，这里force也可以指定非当前依赖的版本

**多项目构建**

* 多项目集成：setting.gradle中include子项目名称
* 子项目依赖：在depencies中写上compile project(":项目名")
* 提取公共信息到根项目：在根配置中编写allprojects，如编译版本、java插件
* 统一所有项目版本：新建gradle.properties，设置版本

**发布**

```

apply plugin: 'maven-publish'

publishing {

publications {

myPublish(MavenPublication) {

from components.java

}

}

repositories {

maven {

name "名称"

url "要发布的地址"

}

}

}

```

然后执行发布任务就可以把jar或war发到我们想要的地址库了，url不填默认发布到本地仓库