项目管理结构说明

目录结构说明

顶级目录

放置 Gradle 的配置文件以及 wrapper 等，projects 目录用来存放所有研发项目，deps 目录用于存放 projects中项目依赖的项目，一般是外部的git源项目。

projects 目录

实际项目存放的地点。

这些项目也可能是来自不同的仓库，例如 git、subversion 等，不同的项目组可能会下载不同的项目于依赖模块，这时候只需要配置一下顶级目录中的 settings.gradle，管理好 include 即可。

内置插件

Gradle 自带插件

- 所有子项目
    - **eclipse**： 用于生成 eclipse 配置
    - **idea**： 用于生成 intelliJ IDEA 配置
    - **java**:
- Web 子项目
    - **war**： 用于打 war 包，注意：这儿做了一些处理，war默认打的包带有 SNAPSHOT 后缀，如果想打发布版的 war 包，可采用 gradle release
    - **eclipse-wtp**： 用于生成 wtp 配置，建议配合 eclipse 的 Gradle 插件使用，很方便地将项目依赖的 jar 作为 eclipse 的 libraries，保持 WEB-INF/lib 目录干净清爽

引用的外部插件

- versions： 用于检测jar包是否有新版本，命令为 gradle dependencyUpdates
- gradle.templates： 用于生成项目目录结构，命令较多，可采用 gradle tasks 查看。顶级目录下有个 templates 目录，里面可以自己定义一些模板来代替这个工具的默认模板

子项目jar包调用方式

jar 包定义外移

将所有的 jar 的定义放置到了 dependencyDefinitions.gradle 中，然后在顶级目录的 build.gradle 中引入：

> *apply from: 'dependencyDefinitions.gradle'*

该文件内容为：

```
 ext.versions = [
     spring: '3.2.5.RELEASE',
     security: '3.1.4.RELEASE',
     jackson: '1.9.13',
     logback: '1.0.13',
     slf4j: '1.7.5',
     httpclient: '4.3.1',
 ]



ext.libraries = [
    "servlet-api": "javax.servlet:servlet-api:2.5",
    "jsp-api": "javax.servlet:jsp-api:2.0",

    "ebean": "org.avaje.ebeanorm:avaje-ebeanorm:3.2.4",
    "persistence-api": "javax.persistence:persistence-api:1.0",

    "aspectjweaver": "org.aspectj:aspectjweaver:1.7.4",
    ...
]
```

在各个子项目的 build.gradle 中就只需要写 libraries.'ebean' 即可，将来升级的时候方便。

善用 gradle dependencies

这个命令可以很方便地检查 jar 的依赖情况，显示的是简单的树形结构，因为在命令行下会显示太多行，所以可以将其输出到一个文本中再看：

> *gradle dependencies > depend.log*

通过这个命令极大简化了 ext.libraries 定义，那些自动会依赖的 jar 包就不需要再写了。

java 编译时候报编码错误

这是个经典的问题，当然，用了IDE 之后已经见不到此问题，所以在 gradle 编译的时候再次发现此问题的时候都有点故人复见的感觉了。

想当年初学 java 的时候采用命令行编译的诸位应该都会有这个感觉吧。

解决办法很简单，增加一行设置即可：

> *[compileJava, compileTestJava]*.options*.encoding = 'UTF-8'*

不要定义 group，除非你知道自己在做什么

忽略掉 .gradle 目录

在顶级目录下有个 .gradle 文件夹，这个是 gradle 自动生成的，可以在 git 的 .gitignore 文件中增加一条忽略掉：

> *.gradle/*

Maven 库中没有的 jar 该怎么管理？

顶级目录libs 文件夹里面的 jar 是对所有项目都起作用的。

如果是某个项目自用的，则可以在该项目的 source 下面创建个 libs，具体实现是在顶级目录下的 build.gradle 中：

```
ext.jarTree = fileTree(dir: 'libs', include: '**/*.jar')

ext.rootProjectLibs = new File(rootProject.rootDir,
'libs').getAbsolutePath()

ext.jarTree += fileTree(dir: rootProjectLibs, include: '**/*.jar')



compile jarTree
```

开发环境、生产环境设置

Gradle 有几种办法可以用来管理这些设置，例如将配置文件按环境写在 build 文件的不同块中。

- resources： 通用配置放在这儿
- resources-dev： 开发环境配置
- resources-prod： 生产环境配置

在用 gradle eclipse 命令生成 IDE 配置的时候，会默认将前两个目录包含在 eclipse 中，只有通过 gradle release 生成 war 包的时候才会忽略 dev，使用 prod。

打包命令

打 war 包时候有两种方式：

- gradle war： 测试用的 war 包，文件后缀增加 **SNAPSHOT**，采用的是 resources-dev。
- gradle release： 生产用的 war 包，采用的是 resources-prod。