Maven3 实战笔记 07 继承

刘岩

Email:suhuanzheng7784877@163.com

1. 继承

之前我们学习 Maven 的聚合机制遗留个问题 ,就是多个模块的 pom.xml 文件的内容出现了冗余、重复的内容，解决这个问题其实使用 Maven 的继承机制即可，就像 Java 的继承一样，父类就像一个模板，子类继承自父类，那么有些通用的方法、变量都不必在子类中再重复声明了，具体 Java 继承在内存中的表现形式可以参考

http://suhuanzheng7784877.iteye.com/blog/1000635

和 http://suhuanzheng7784877.iteye.com/blog/1000700 中的部分内容。Maven 的继承机制类似，在一个父级别的 Maven 的 pom 文件中定义了相关的常量、依赖、插件等等配置后，实际项目模块可以继承此父项目的 pom 文件，重复的项不必显示的再声明一遍了，相当于父 Maven 项目就是个模板，等着其他子模块去继承。不过父 Maven 项目要高度抽象，高度提取公共的部分（交集）。笔者使用了先前的聚合项目模块做的父模板 pom，实际上很多机构也是这么实施的。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.liuyan.account</groupId>
    <artifactId>MavenAccount-aggregator</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>

    <properties>
        <springversion>2.5.6</springversion>
        <junitversion>2.5.6</junitversion>
    </properties>
```

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${springversion}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>${springversion}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${springversion}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
        <version>${springversion}</version>
    </dependency>
    <dependency>
        <groupId>javax.mail</groupId>
        <artifactId>mail</artifactId>
        <version>1.4.1</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.7</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.icegreen</groupId>
        <artifactId>greenmail</artifactId>
        <version>1.3.1b</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <resources>
        <resource>
```

```
            <directory>src/main/resource</directory>
        </resource>
    </resources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <version>2.1.1</version>
            <executions>
                <execution>
                    <id>buildSource</id>
                    <goals>
                        <goal>jar-no-fork</goal>
                    </goals>
                    <inherited>false</inherited>
                    <configuration>
                    </configuration>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <target>1.5</target>
            </configuration>
        </plugin>
    </plugins>
</build>

<modules>
    <module>../MavenAccount-email</module>
    <module>../MavenAccount-persist</module>
</modules>
</project>
```

这个 pom 文件即描述了通用的依赖模板，也列举出了聚合的模块，放心 modules 不会被继承。下面我们来改造一下之前的两个模块

邮件模块 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

```
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <artifactId>MavenAccount-email</artifactId>
    <packaging>jar</packaging>

    <parent>
        <groupId>com.liuyan.account</groupId>
        <artifactId>MavenAccount-aggregator</artifactId>
        <version>0.0.1-SNAPSHOT</version>

    <relativePath>../MavenAccount-aggregator/pom.xml</relativePath>
    </parent>

</project>
```

注册模块 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <artifactId>MavenAccount-persist</artifactId>
    <packaging>jar</packaging>

    <parent>
        <groupId>com.liuyan.account</groupId>
        <artifactId>MavenAccount-aggregator</artifactId>
        <version>0.0.1-SNAPSHOT</version>

    <relativePath>../MavenAccount-aggregator/pom.xml</relativePath>
    </parent>

</project>
```

每一个模块节省了不少重复的配置项。每一个分模块少了版本号、groupId 信息，因为他们都被继承下来了，所以自然不用显示的写上，更代表了，他们是一个大项目的一个小螺丝钉。以下是可以继承的元素：groupId（项目组 id）、version（版本信息）、description（描述信息）、orrganization（项目组织信息）、inceptionYear（项目创始年份）、url（项目链接地址）、developer（开发人）、contributors（项目贡献者）、distributionManagement（项目部署配置）、issueManagement（项目问题跟踪信息）、ciManagement（项目持续

集成信息）、scm（项目版本控制信息）、maillingLists（邮件列表信息）、properties（自定义常量属性信息）、dependencies（项目依赖配置）、dependencyManagement（项目依赖管理配置）、repositories（项目仓库配置）、build（项目源码目录、输出目录、插件、插件管理配置）、reporting（项目的报告输出目录，报告插件配置）。其中红色的表示经常需要被继承的元素。

2. 继承下的依赖管理

假如有些第三方的包在某些模块中用到了，某些模块没有用到，而又不想让这些应用到的项目版本混乱，统一按照父模板中的配置进行下载。这就用到了 Maven 的依赖管理机制了，就是咱们刚刚提到的 dependencyManagement 标签。需要说明的就是在父 pom 文件中配置的 dependencyManagement 项，不会对子 pom 文件的依赖产生任何影响，它仅仅是将 dependencyManagement 项的内容毫无保留地继承了下来，对于子项目的依赖哪些东西，子项目的 pom 还是一无所知~。如果此时在子 pom 文件中显示的声明了父 dependencyManagement 标签中的某些依赖选项的话，那么才会依赖生效。首先我们来看父 pom 内容片断，将原先的依赖项修改如下

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${springversion}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${springversion}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${springversion}</version>
```

```
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-context-support</artifactId>
                <version>${springversion}</version>
            </dependency>
            <dependency>
                <groupId>javax.mail</groupId>
                <artifactId>mail</artifactId>
                <version>1.4.1</version>
            </dependency>
            <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.7</version>
                <scope>test</scope>
            </dependency>
            <dependency>
                <groupId>com.icegreen</groupId>
                <artifactId>greenmail</artifactId>
                <version>1.3.1b</version>
                <scope>test</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

下面修改我们的邮件模块 pom 文件片断如下

```
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>javax.mail</groupId>
            <artifactId>mail</artifactId>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.icegreen</groupId>
            <artifactId>greenmail</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

在这里不用显示的声明版 jar 包本号了，但是从 pom 文件的篇幅上来说并没有少多少内容，仅仅是版本号没有写上罢了。权威的人士建议这么做，虽然 pom 内容冗余，但是项目模块的使用依赖的版本做到了统一，至于效果怎么样，笔者觉得没有统一说法，各有利弊。

3. 继承下的插件管理

组件依赖机制已有的问题得到了解决，那么项目的插件依赖的问题与组件依赖类似

```
    <build>
        <resources>
            <resource>
                <directory>src/main/resource</directory>
            </resource>
        </resources>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-source-plugin</artifactId>
                    <version>2.1.1</version>
                    <executions>
                        <execution>
                            <id>buildSource</id>
                            <goals>
                                <goal>jar-no-fork</goal>
```

```
                        </goals>
                        <inherited>false</inherited>
                        <configuration>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <target>1.5</target>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
</build>
```

如果在子 pom 文件中让插件生效，必须在子 pom 内容中显示加入插件的配置，但是不必声明其使用的版本的代码就不赘述了。由此看之根据项目具体情况来设定具体的父 pom 文件的内容。

4. 聚合与继承

其实两者很有可能通过同一个 pom 文件来实现的，父 pom 文件、集合 pom 文件完全不冲突。只不过聚合是让聚合项目构建的时候知道应该具体找哪些小模块，而小模块可以不知道聚合项目的存在（当然了，一般项目的沟通都会让其知道彼此的存在）。而继承与聚合恰恰相反，是子项目一定要知道自己继承自哪个父项目，好利用它通用的一些配置项。而父项目不必知道具体哪些小模块项目使用了自己。还有一点就是 Maven 的继承机制和 Java 一样，是单继承机制，一个子项目不能同时继承多个父项目。