

Maven配置

pom.xml

1. jdk版本设置

方式一

```
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
```

`maven.compiler.source` 和 `maven.compiler.target` 并不是一定有效的，如果要强制使用某个版本的jdk，可以使用 `executable` 设置方式：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.7.0</version>
  <configuration>
    <verbose>true</verbose>
    <fork>true</fork>
    <executable>!-- 关键-设置javac命令路径: path-to-javac --></executable>
    <compilerVersion>1.3</compilerVersion>
  </configuration>
</plugin>
```

方式二

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.1</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <showWarnings>true</showWarnings>
  </configuration>
</plugin>
```

2. javac参数

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.1</version>
  <configuration>
    <!-- 添加javac参数 -->
    <compilerArgument>-Xlint:unchecked</compilerArgument>

    <source>${java.version}</source>
    <target>${java.version}</target>
    <showWarnings>true</showWarnings>
  </configuration>
</plugin>
```

3. build

3.1 设置class输出目录

```
<build>
  <outputDirectory>target/classes</outputDirectory>
</build>
```

3.2 extensions

Extensions are a way to add libraries to [Core Classloader](#).

个人理解：Extensions是对maven的核心库的扩展，增强maven的功能；

经典应用： [Wagon providers](#) 和 引入生命周期增强插件

3.3 defaultGoal

```
<build>
  <defaultGoal>clean</defaultGoal>
</build>
```

执行 mvn 命令时，如果没有指定目标，指定使用的默认目标。如上配置：如果执行mvn命令，则相当于执行mvn clean

3.4 filtering

使用方式很简单，可以参考[示例](#)，但是，如果使用了spring-boot，不再使用 `${}` 来获取maven的属性，而是需要使用 `@...@` 获取（[参考](#)），springMvc暂时没测试；

3.5 filters

filters 是引入properties的作用，比如：通过filters引用的properties属性，可以在filtering功能中使用；

3.5 other

```
<!-- 构建文件输出目录 -->
<directory>target</directory>
<!-- 构建后的jar/war文件名 -->
<finalName>jar/war文件名称</finalName>
```

profiles

1. activation

1.1 jdk

```
<profile>
  <id>jdk_test</id>
  <activation>
    <!-- 只能是1.*之类的版本号，jdk1.8、jdk8、1.8.0_162都不行-->
    <jdk>1.6</jdk>
  </activation>
  <properties>
    <proj.server.port>81</proj.server.port>
  </properties>
</profile>
```

1.2 os

```
<profile>
  <id>jdk_test</id>
  <activation>
    <os>
      <!-- 需要与jvm中的os.name属性完全一致 -->
    </os>
  </activation>
</profile>
```

```

        <name>Windows</name>
        <!-- 需要与jvm中的os.version属性完全一致 -->
        <version>10.0</version>
        <!-- 需要与jvm中的os.arch属性完全一致 -->
        <arch>xamd64</arch>
    </os>
</activation>
<properties>
    <proj.server.port>81</proj.server.port>
</properties>
</profile>

```

plugins

1. plugin

1.1 executions

1.1.1 phase

绑定插件的goal和声明周期；如果没有配置phase，会默认使用插件的matedata中定义的默认phase（在插件jar包中的MATE-INFO > maven > plugin.xml 文件中有对goal和phase进行绑定）；

- <phase> 标签标示之后配置的 <goals> 标签中的goal何时执行
- 只有在 <goals> 中配置的goal才会执行；

1.1.2 inherited

是否将配置传播到子pom中；value类型实际上是boolean，但是xsd文件中定义的是String类型；

TODO: 与 <plugin> 标签下的 <inherited> 有何区别？

1.1.3 configuration

基础普及：每个插件（plugin）包含多个Mojo，每个Mojo里有多属性；

<configuration> 标签的子标签将对应Mojo中字段，子标签值将set到Mojo对应字段上：

```

public class MyQueryMojo extends AbstractMojo {
    private String url;
    private int timeout;
    private String[] options;

    public void execute() throws MojoExecutionException {
        ...
    }
}

```

```

<build>
  <plugins>
    <plugin>
      <artifactId>maven-myquery-plugin</artifactId>
      <version>1.0</version>
      <configuration>
        <url>http://www.foobar.com/query</url>
        <timeout>10</timeout>
        <options>
          <option>one</option>
          <option>two</option>
          <option>three</option>
        </options>
      </configuration>
    </plugin>
  </plugins>
</build>

```

有些插件的Mojo属性不是从pom中进行配置的，而是通过系统命令传入的，详细介绍请参考[文档](#)

setting.xml

1. usePluginRegistry

```
<!--Maven是否需要使用plugin-registry.xml文件来管理插件版本。如果需要让Maven使用文件${user.home}/.m2/plugin-registry.xml来管理插件版本，则设为true。默认为false。-->
<usePluginRegistry>false</usePluginRegistry>
```

2. pluginGroups

```
<!--当插件的组织Id (groupId) 没有显式提供时，供搜寻插件组织Id (groupId) 的列表。该元素包含一个pluginGroup元素列表，每个子元素包含了一个组织Id (groupId)。当我们使用某个插件，并且没有在命令行为其提供组织Id (groupId) 的时候，Maven就会使用该列表。默认情况下该列表包含了org.apache.maven.plugins和org.codehaus.mojo -->
<pluginGroups>
  <!--plugin的组织Id (groupId) -->
  <pluginGroup>org.codehaus.mojo</pluginGroup>
</pluginGroups>
```

3. servers

包含 `server` 列表，每个server一个id，在pom.xml中的

4. profiles

如果一个 `settings.xml` 中的 `profile` 被激活，它的值会覆盖任何其它定义在 `pom.xml` 中带有相同id的 `profile`：

4.1 激活方式

4.1.1 默认

指定`activeByDefault`为`true`的时候就表示当没有指定其他`profile`为激活状态时，该`profile`就默认会被激活

```
<profiles>
  <profile>
    <id>profileTest1</id>
    <properties>
      <hello>world</hello>
    </properties>
    <activation>
      <!-- 默认激活 -->
      <activeByDefault>true</activeByDefault>
    </activation>
  </profile>
</profiles>
```

4.1.2 命令行

```
mvn package -P profileA
```

`-P` 参数也可以抑制已经激活的profile

```
mvn package -P !profileTest1
```

4.1.3 settings.xml中指定

```
<activeProfiles>
  <!-- profileA可以在settings.xml中，也可以在pom.xml中 -->
```

```

    <activeProfile>profileTest1</activeProfile>
    <activeProfile>profileTest2</activeProfile>
</activeProfiles>

```

如果profile中有冲突的属性，maven会按照定义顺序，后面的profile覆盖前面的：

4.1.4 条件激活

在jdk为1.5版本系列的时候激活profileTest1

```

<profiles>
  <profile>
    <id>profileTest1</id>
    <jdk>1.5</jdk>
  </profile>
</profiles>

```

在jdk为1.4、1.5和1.6的时候激活profileTest1

```

<profiles>
  <profile>
    <id>profileTest1</id>
    <jdk>[1.4,1.7)</jdk>
  </profile>
</profiles>

```

根据操作系统来激活profile

```

<profiles>
  <profile>
    <id>profileTest1</id>
    <activation>
      <os>
        <name>Windows XP</name>
        <family>Windows</family>
        <arch>x86</arch>
        <version>5.1.2600</version>
      </os>
    </activation>
  </profile>
</profiles>

```

系统属性来激活profile，比如执行 `mvn package -Dhello=world`，将激活下面的profileTest1:

```

<profiles>
  <profile>
    <id>profileTest1</id>
    <activation>
      <property>
        <name>hello</name>
        <!-- 如果value标签没设置，则只要系统属性hello存在，无论如何，都会激活 -->
        <value>world</value>
      </property>
    </activation>
  </profile>
</profiles>

```

文件是否存在激活profile

```

<profiles>
  <profile>
    <id>profileTest1</id>
    <activation>
      <file>
        <!-- 子标签: exists/missing -->
        <missing>target</missing>
      </file>
    </activation>
  </profile>
</profiles>

```

```
    </activation>
  </profile>
</profiles>
```

5. repositories

maven调用仓库的顺序是：本地仓库 -> 中央仓库 -> 第三方仓库

```
<repositories>
  <repository>
    <layout>default</layout>
  </repository>
</repositories>
```

关于layout的解释可以参考[这里](#)

扩展应用

1. 打包后部署

[参考](#)