Maven3 实战笔记 13Maven Profile 定制化构建

刘岩

Email:suhuanzheng7784877@163.com

1. 前言

有时候我们开发的软件有不同的环境，开发阶段是开发环境，也就是我们这些研发人员平时使用的环境，大多数人开发还是在 Windows 下面吧，少数人连开发环境都需要在 Linux 或者 Unix 下进行 ,因为 Java 自身的跨平台性可能在哪个操作系统下开发差别不大，如果 Java 调用 C/C++执行特定服务，就需要 C/C++人员开发时最好和生产环境一样在 Linux（而且版本一致）下进行研发，省得到时候测试的时候还得放到 Linux 下重新编译一遍。开发到了一定阶段后往往就是测试那边 Team 的加入了，功能测试、压力测试等等需要一套测试环境，而往往测试环境也分很多种情况，功能测试环境比较接近于开发环境，而压力测试环境往往接近于生产环境，甚至就是生产环境。面对这么多环境，以前我们可能都是手工进行配置，在不同环境下赋予不同的配置值。而手工配置就面临着可能出现错误，效率比较低等问题。

使用 Maven 的 Profile 配置，可以帮助我们灵活的进行构建项目，使之通过统一的定制化 pom.xml 配置就能让它适应不同的环境，如此一来显得软件更有应变性和适应性。

2. 屏蔽构建差异

笔者就以之前那个持久层模块作为基础，假如数据库在开发环境和测试环境有差异，怎么办，一般我们将数据库信息写到了配置文件 properties 中。

db.properties 内容如下

```
db.Driver=${db.Driver}
db.url=${db.url}
db.user=${db.user}
db.password=${db.password}
```

大家有可能奇怪，这里应该是具体的配置值了，是啊，配置值在 pom.xml 中

pom.xml 片段如下

```
<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>
    <name>MavenAccount-persist</name>

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

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

    <dependencies>

    ..................省略依赖

    </dependencies>

    <build>
        <resources>
            <resource>

    <directory>${project.basedir}/src/main/resource</directory>
                <filtering>true</filtering>
            </resource>
        </resources>
    </build>

    <profiles>
        <profile>
            <id>test1</id>
            <activation>
                <property>
                    <name>env</name>
                    <value>dev1</value>
                </property>
```

```
            </activation>
            <properties>
                <db.Driver>org.gjt.mm.mysql.Driver</db.Driver>
                <db.url>jdbc:mysql://localhost:3306/uxian99</db.url>
                <db.user>liuyan</db.user>
                <db.password>111111</db.password>
            </properties>
        </profile>
        <profile>
            <id>test2</id>
            <activation>
                <property>
                    <name>env</name>
                    <value>dev2</value>
                </property>
            </activation>
            <properties>
                <db.Driver>com.mysql.jdbc.Driver</db.Driver>

    <db.url>jdbc:mysql://192.168.1.109:3306/uxian99</db.url>
                <db.user>dba</db.user>
                <db.password>dba</db.password>
            </properties>
        </profile>
    </profiles>
</project>
```

红色配置部分代表过滤资源配置文件，所有的资源文件在执行 Maven 构建命令时，pom 中种种信息都和配置文件中的内容息息相关，需要让配置文件感应到 pom 内容。

蓝色部分就是定制化的 prifile 属性信息 ,其中配置了 2 个个性化配置信息。test1 和 test2。activation 标签代表激活该特性的配置，这里所谓激活，其实就是让该个性化配置生效的意思。

下面我们写一个 Java 类简单读取配置文件

```
package util;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
```

```
public class ReadConfig {

    public static String read() throws IOException {
        InputStream in =
ClassLoader.getSystemResourceAsStream("db.properties");
        Properties p = new Properties();
        p.load(in);
        String dbDriver = (String) p.get("db.Driver");
        String dburl = (String) p.get("db.url");
        String dbuser = (String) p.get("db.user");
        String dbpassword = (String) p.get("db.password");

        System.out.println(dbDriver);
        System.out.println(dburl);
        System.out.println(dbuser);
        System.out.println(dbpassword);
        return dbpassword;

    }

}
```

之后为其写单元测试

```
package util;

import static org.junit.Assert.assertTrue;

import java.io.IOException;

import org.junit.Test;

public class ReadConfigTest {

    @Test
    public void testGenerateText() throws IOException {
        String password = ReadConfig.read();
        System.out.println(password);
        assertTrue(password.equals("dba"));
    }

}
```

代码很简单，下面我们在控制台输入如下命令看看

```
mvn clean package –Ptest2
```

控制台输出结果如下

```
-------------------------------------------------------

 T E S T S

-------------------------------------------------------

Running util.ReadConfigTest

com.mysql.jdbc.Driver

jdbc:mysql://192.168.1.109:3306/uxian99

dba

dba

dba

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.078 sec



Results :



Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

可以看到单元测试用例结果是 test2 的配置信息，而构建后的配置文件内容如下

db.properties

```
db.Driver=com.mysql.jdbc.Driver

db.url=jdbc:mysql://192.168.1.109:3306/uxian99

db.user=dba

db.password=dba
```

可以看到，构建后的配置文件内容不再是${db.Driver}这种临时变量信息，而是被 pom 文件替换后的实际配置值。

不更改任何代码，在控制台输入如下

```
mvn clean package -Denv=dev1
```

结果如下

```
-------------------------------------------------------

 T E S T S

-------------------------------------------------------

Running util.ReadConfigTest

org.gjt.mm.mysql.Driver

jdbc:mysql://localhost:3306/uxian99

liuyan

111111

111111

Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.062 sec <<< FA

ILURE!



Results :



Failed tests:

   testGenerateText(util.ReadConfigTest)
```

```
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
```

表示 profile 的 test1 特性被激活了，测试结果与预期预料不符合。激活 profile 有以上两种方式，显示指定配置参数方式、系统属性=某些具体值的时候。还有在用户级 setting.xml 或者全局级 setting.xml 文件中配置默认信息的。不过这种方式不是很赞成。因为配置了此信息意味着只能在自己本机环境中得到正确的相应，项目打包，发布后并不会保存到自身的 pom 文件中，那么别人构建的时候也许会报错。所以尽量在项目级别的 pom.xml 中配置相关定制化信息。

3. 总结

Profile 是轮廓;外形;外观;形象; 侧面(像),侧影的意思，我觉得叫做个性化配置更好。个性化配置还可以过滤 web 项目资源、集成测试。因为网上有相关的资源，就不总结了。多数应用还是屏蔽个性化构建的差异。