Maven

主讲：石小俊

1.介绍maven

maven是一款自动化构建工具

专注服务于java平台的项目的构建以及依赖管理

1-1 什么是构建

构建并不是创建一个工程

而是以java源文件、配置文件、资源文件等作为原料

去生产一个可以真正运行的一个项目的过程

1-2 构建的环节

- 清理
  - 删除以前的编译结果
  - 为重新编译做好准备
- 编译
  - 将java源文件编译成字节码文件
- 测试
  - 针对项目中的每一个关键点进行测试
  - 确保项目开发的正确性
- 报告
  - 在每一次测试完成之后在生成一个对应的测试报告来展现测试结果
- 打包
  - 将一个测试通过的工程封装成对应的压缩文件用于安装与部署
  - jar包对应的是java工程
  - war包对应的是web工程
- 安装
  - 在maven环境中将jar包或者war存放到对应的maven仓库中
  - 方便他人使用
- 部署
  - 将war包放入到服务器中

1-3 为什么用

- 获取第三方jar包
- jar包依赖
- jar包管理
- 项目规模

2.安装maven

2-1 windows

- 下载对应版本的maven
  - xxx.zip
- 将压缩文件解压到一个非中文无空格的目录中
- 配置环境变量
  - 自己的
    - M2_HOME：maven安装目录
  - 依赖的
    - JAVA_HOME
    - CLASSPATH
  - 系统的
    - path：原来的path+当前maven安装目录的bin目录
- 检查是否配置成功
  - mvn -version

2-2 linux

- 下载对应版本的maven
  - xxx.tar.gz
  - xxx.tgz
- 将压缩文件解压到一个非中文无空格的目录中
- 配置环境变量
  - 在soft01下找到一个隐藏文件:bashrc
  - 打开该文件，在该文件的末尾加上以下配置

```
export M2_HOME=maven安装目录
export PATH=$PATH:maven安装目录下的bin目录
```

- 执行source命令
  - source .bashrc
- 检查是否配置成功
  - mvn -version

3.开发一个maven工程

3-1 定义一个约定的目录结构

Hello----------根目录

|-----src----------源码目录

|-----|--------main----------存放主程序

|-----|--------|--------java----------存放主程序的源文件

|-----|--------|--------resource----------存放主程序的配置文件

|-----|--------test----------存放测试程序

|-----|--------|--------java----------存放测试程序的源文件

|-----|--------|--------resource----------存放测试程序的配置文件

|-----pom.xml----------maven核心配置文件

3-2 为什么用这个结构

maven负责该工程的自动化构建工具

但是maven是死的，是一个提前写好的程序

我们必须按照maven所定义的规则去开发

maven需要知道去哪边找到对应的文件

因此我们必须按照约定的目录结构来开发

3-3 开发

HelloMaven.java

```
package com.itany.maven;


public class HelloMaven{

    public String sayHello(){
        return "Hello Maven!";
    }

}
```

TestHello.java

```
package com.itany.maven;

import org.junit.Test;
import static org.junit.Assert.*;

public class TestHello{

    @Test
    public void test(){
        HelloMaven hm = new HelloMaven();
        String result = hm.sayHello();
        assertEquals("Hello Maven!",result);
    }

}
```

3-4.POM

Project Object Model

项目对象模型

pom.xml是整个maven的核心配置文件

pom.xml对于maven而言

相当于web工程中的web.xml

与maven构建相关的一切操作都在pom.xml中进行配置

```
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.itany.maven</groupId>
  <artifactId>HelloMaven</artifactId>
  <version>0.0.1-SNAPSHOT</version>


  <dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.0</version>
        <scope>test</scope>
    </dependency>
  </dependencies>
```

```
</project>
```

3-5.坐标

坐标表示指向某一个maven工程的具体位置

maven中提供了三个向量

- groupId：表示组织机构+工程名，即路径名
- artifactId：模块名称
- version：版本号

3-6 运行maven命令

在运行的时候，我们必须进入到maven工程所在的目录中执行

即与pom.xml平级

- mvn clean
  - 清理
- mvn compile
  - 编译主程序的代码
  - 生成一个classes目录，用于存放当前主程序的字节码文件
- mvn test-compile
  - 编译测试程序
  - 生成一个test-classes目录，用于存放当前测试程序的字节码文件
- mvn test
  - 执行测试
  - 生成surefire-reports目录，用于存放测试报告
- mvn package
  - 生成对应的jar包/war包
- mvn install
  - 执行安装命令
  - 将maven工程安装到仓库中

4.maven仓库

4-1 仓库分类

- 本地仓库
  - 在当前的计算机上部署一个仓库目录

    - 为当前计算机中的所有的maven工程服务
    - 如果本地仓库中没有插件，会在远程仓库中进行下载
    - 并且在下载完成后自动放入到对应的本地仓库中
- 远程仓库
    - 私服
        - 搭建在局域网内的仓库
        - 为局域网范围内的所有maven工程服务器
        - 一般用于公司内容
    - 中央仓库
        - 搭建在互联网上，为全世界所有的maven工程服务
    - 中央仓库镜像
        - 本质上的内容就是中央仓库中的内容
        - 只是为了分担服务器的压力
        - 提升用户的访问速度
        - 在各个区域建立了一些中央仓库的镜像

4-2 本地仓库的配置

找到maven安装目录下的conf目录中的settings.xml文件

找到localRepository标签配置

```
<!-- localRepository
  | The path to the local repository maven will use to stc
  |
  | Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->
<!-- 配置本地仓库 -->
<localRepository>E:\maven_repository</localRepository>
```

配置本地仓库，在localRepository标签体中输入本地仓库的位置

4-3 远程仓库的配置

找到maven安装目录下的conf目录中的settings.xml文件

找到mirrors标签配置

在mirrors中进行如下配置

```
<mirror>
    <id>aliyun</id>
    <mirrorOf>central</mirrorOf>
    <name>Nwxus aliyun</name>
    <!-- 阿里私服  -->
    <!-- <url>http://maven.aliyun.com/nexus/content/groups/public</url>-->
    <url>http://repo1.maven.org/maven2/</url>
</mirror>
```

5.第二个maven工程

通过命令直接自动创建maven工程

- 通过命令进入到存放创建maven工程的位置
- mvn archetype:generate
    - 执行命令之后生成了很多选项，对应的是不同类型的maven工程
    - 默认的为最简单的java工程

HelloFriend.java

```
package com.itany.maven;

import com.itany.maven.HelloMaven;

public class HelloFriend{

    public String sayHelloToFriend(String name){
        HelloMaven hm = new HelloMaven();
        String result = hm.sayHello()+"I am "+name;
        return result;
    }

}
```

TestFriend.java

```
package com.itany.maven;

import org.junit.Test;
import static org.junit.Assert.*;

public class TestFriend{

    @Test
    public void test(){
        HelloFriend hf = new HelloFriend();
        String result = hf.sayHelloToFriend("admin");
```

```
        assertEquals("Hello Maven!I am admin",result);
    }

}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.itany.maven</groupId>
  <artifactId>HelloFriend</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>HelloFriend</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

    <!-- 配置HelloMaven的依赖 -->
    <dependency>
      <groupId>com.itany.maven</groupId>
      <artifactId>HelloMaven</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <scope>compile</scope>
    </dependency>

  </dependencies>

  <build>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
```

```
      <plugins>
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.0.0</version>
        </plugin>
        <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_jar_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.7.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.20.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

6.依赖

6-1 介绍依赖

当使用maven进行开发的时候

对应的工程存在一些jar包的依赖

某些时候依赖的是第三方jar包

某些时候依赖的是另一个工程

此时需要对当前的maven工程进行依赖的配置

配置依赖时，我们必须保证我们所依赖的jar包必然是存在于对应的maven仓库中

在查找依赖的时候，优先查找本地仓库

如果本地仓库不存在，则去远程仓库中查找

找到之后会将其备份到本地仓库

每一个jar包都存在一个坐标

我们需要通过给定的坐标找到对应的资源包

6-2 依赖的配置

- dependencies标签
  - 配置所有的依赖信息
  - 可以存在无数个依赖配置
- dependency标签
  - 配置每一个依赖
  - groupId：对应资源的组织机构
  - artifactId：对应资源的模块名
  - version：对应资源的版本号
  - scope：依赖的范围，值有三种
    - test：测试，只作用于测试程序中
    - compile：默认值，作用于整个工程
    - provided：编写源代码时需要使用该资源，但是不参与打包

6-3 scope作用范围

scope	对于主程序是否有效	对于测试程序是否有效	是否参与打包	是否参与部署
test	无效	有效	不参与	不参与
compile	有效	有效	参与	参与
provided	有效	有效	不参与	不参与

6-4 依赖的特性

- 传递性
  - A工程依赖于B
  - B工程依赖于C
  - 则此时A工程中不仅存在了B，还存在了C
  - 此时依赖配置的前提是:scope作用范围为:compile
- 最短距离传递

    - A、B、C为三个maven工程
    - 其中A依赖B，B依赖C
    - C配置了某个资源的依赖，其版本为4.0
    - B同样配置了该资源的依赖，其版本为4.12
    - A中没有配置该资源的依赖
    - 但是此时A中存在该资源，其版本号为与A距离最近的一个maven工程的资源版本号
    - 即4.12

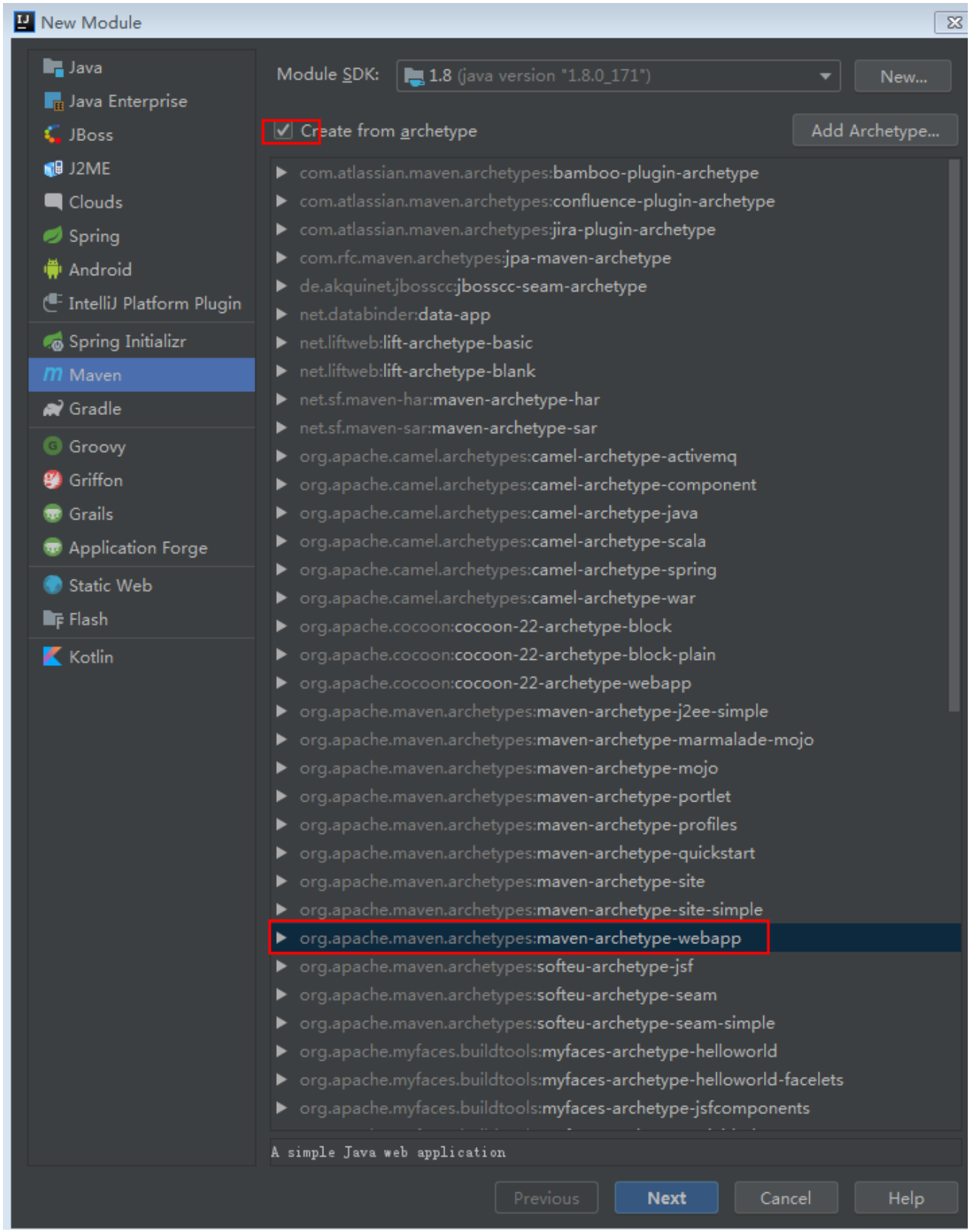
7.生命周期

maven存在三套完全独立的生命周期

- clean
    - 在真正构建之前执行清理工作
- default
    - 构建的核心部分
    - compile、test-compile、test、package、install
- site
    - 生成站点

8.web工程

8-1 idea创建web工程



当web工程创建完成之后

需要替换web.xml

替换成合适的带有schema约束的xml文件

8-2 web工程依赖配置

```
<?xml version="1.0" encoding="UTF-8"?>

<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.itany.maven</groupId>
  <artifactId>HelloWeb</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>


    <properties>
        <servlet.version>3.1.0</servlet.version>
        <jsp.version>2.2</jsp.version>
        <jstl.version>1.2</jstl.version>
        <mysql.version>5.1.46</mysql.version>
    </properties>


    <dependencies>
        <!-- J2EE begin -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>${servlet.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>${jsp.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>${jstl.version}</version>
            <scope>provided</scope>
        </dependency>

        <!-- J2EE end -->

        <!-- mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.version}</version>
```

```
        </dependency>

    </dependencies>


</project>
```

9.继承

package标签:

- jar：对应的是java工程，默认值
- war：对应的web工程
- pom：对应的是父工程

当在父工程中配置了依赖

则在子工程中会同步加载这些依赖的资源

子工程会继承父工程的坐标

如果不想使用父工程提供了，也可以由子工程本身去重写

10.聚合

一键安装

如果单个安装，首先需要根据依赖关系一个个的来安装

如果顺序出错，则安装失败

而通过聚合实现一键安装

只需要将你所需要安装的所有的Model指定之后

同时执行父工程的install

则会同步安装该父工程所指定的所有的子工程