Maven

1.Maven基础（已学习）

2.Maven高级

```
maven 基础知识回顾
maven 的依赖传递
maven 聚合工程
```

2.1 maven基础知识回顾

2.1.1 maven介绍

maven 是一个项目管理工具，主要作用是在项目开发阶段对Java项目进行依赖管理和项目构建。

依赖管理：就是对jar包的管理。通过导入maven坐标，就相当于将仓库中的jar包导入了当前项目中。

项目构建：通过maven的一个命令就可以完成项目从清理、编译、测试、报告、打包，部署整个过程。

2.1.2 maven的仓库类型

1.本地仓库

2.远程仓库

①maven中央仓库（地址：http://repo2.maven.org/maven2/）

②maven私服（公司局域网内的仓库，需要自己搭建）

③其他公共远程仓库（例如apache提供的远程仓库，地址：http://repo.maven.apache.org/maven2/）

本地仓库---》maven私服---》maven中央仓库

2.1.3 maven常用命令

clean：　清理

compile：编译

test：　测试

package：打包

install：　安装

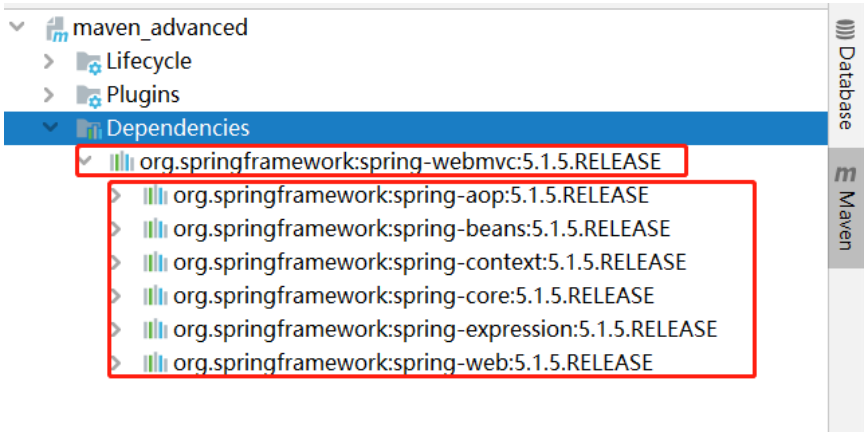
2.1.4 maven坐标书写规范

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.32</version>
</dependency>
```

2.2 maven的依赖传递

2.2.1 什么是依赖传递

在maven中，依赖是可以传递的，假设存在三个项目，分别是项目A，项目B以及项目C。假设C依赖B，B依赖A，那么我们可以根据maven项目依赖的特征不难推出项目C也依赖A。

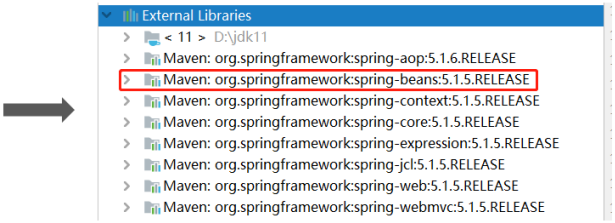


通过上面的图可以看到，我们的web项目直接依赖了spring-webmvc，而spring-webmvc依赖了sping-aop、spring-beans等。最终的结果就是在我们的web项目中间接依赖了spring-aop、spring-beans等。

依赖冲突

由于依赖传递现象的存在， spring-webmvc 依赖 spirng-beans-5.1.5，spring-aop 依赖 spring-beans-5.1.6，但是发现 spirng-beans-5.1.5 加入到了工程中，而我们希望 spring-beans-5.1.6 加入工程。这就造成了依赖冲突。

```xml
<dependencies>
    <!--springMVC坐标-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.1.5.RELEASE</version>
    </dependency>

    <!--springAOP坐标-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>5.1.6.RELEASE</version>
    </dependency>
</dependencies>
```



2.2.2 如何解决依赖冲突

1.使用maven提供的依赖调解原则

第一声明者优先原则

路径近者优先原则

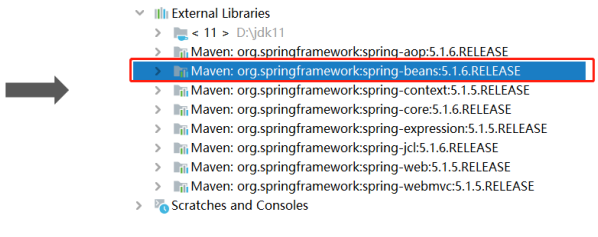
2.排除依赖

3.锁定版本

2.2.3 依赖调节原则——第一声明者优先原则

在 pom 文件中定义依赖，以先声明的依赖为准。其实就是根据坐标导入的顺序来确定最终使用哪个传递过来的依赖。

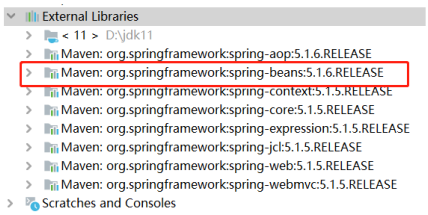
```xml
<!--springAOP坐标-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>5.1.6.RELEASE</version>
</dependency>
<!--springMVC坐标-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.5.RELEASE</version>
</dependency>
```



结论：通过上图可以看到，spring-aop和spring-webmvc都传递过来了spring-beans，但是因为spring-aop在前面，所以最终使用的spring-beans是由spring-aop传递过来的，而spring-webmvc传递过来的spring-beans则被忽略了。

2.2.4 依赖调节原则——路径近者优先原则

```xml
<dependencies>
    <!--spring mvc-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.1.5.RELEASE</version>
    </dependency>

    <!--spring aop-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>5.1.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>5.1.6.RELEASE</version>
    </dependency>
```

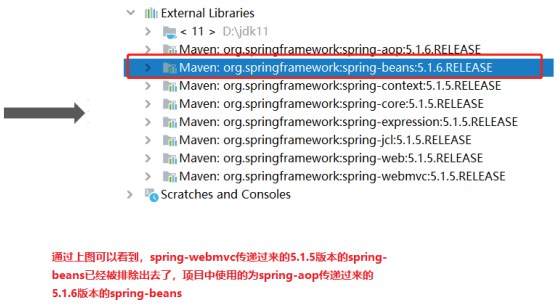


总结：直接依赖大于依赖传递

2.2.5 排除依赖

可以使用exclusions标签将传递过来的依赖排除出去。

```xml
<!--springMVC坐标-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.5.RELEASE</version>
    <!--排除依赖-->
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<!--springAOP坐标-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>5.1.6.RELEASE</version>
</dependency>
```



通过上图可以看到，spring-webmvc传递过来的5.1.5版本的spring-beans已经被排除出去了，项目中使用的为spring-aop传递过来的5.1.6版本的spring-beans

2.2.6 版本锁定

采用直接锁定版本的方法确定依赖jar包的版本，版本锁定后则不考虑依赖的声明顺序或依赖的路径，以锁定的版本为准添加到工程中，此方法在企业开发中经常使用。

版本锁定的使用方式：

第一步：在dependencyManagement标签中锁定依赖的版本

第二步：在dependencies标签中声明需要导入的maven坐标

①在dependencyManagement标签中锁定依赖的版本

```
<!--依赖jar包版本锁定-->
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>5.1.6.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.6.RELEASE</version>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
```

注意：pom文件中使用dependencyManagement标签进行依赖jar的版本锁定，并不会真正将jar包导入到项目中，只是对这些jar的版本进行锁定。项目中使用那些jar包，还需要在dependencies标签中进行声明。

②在dependencies标签中声明需要导入的maven坐标

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.1.5.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>

</dependencies>
```

注意：由于前面已经在dependencyManagement标签中锁定了依赖jar包的版本，后面需要导入依赖时只需要指定groupId和artifactId，无须再指定version。

2.2.7 properties标签的使用

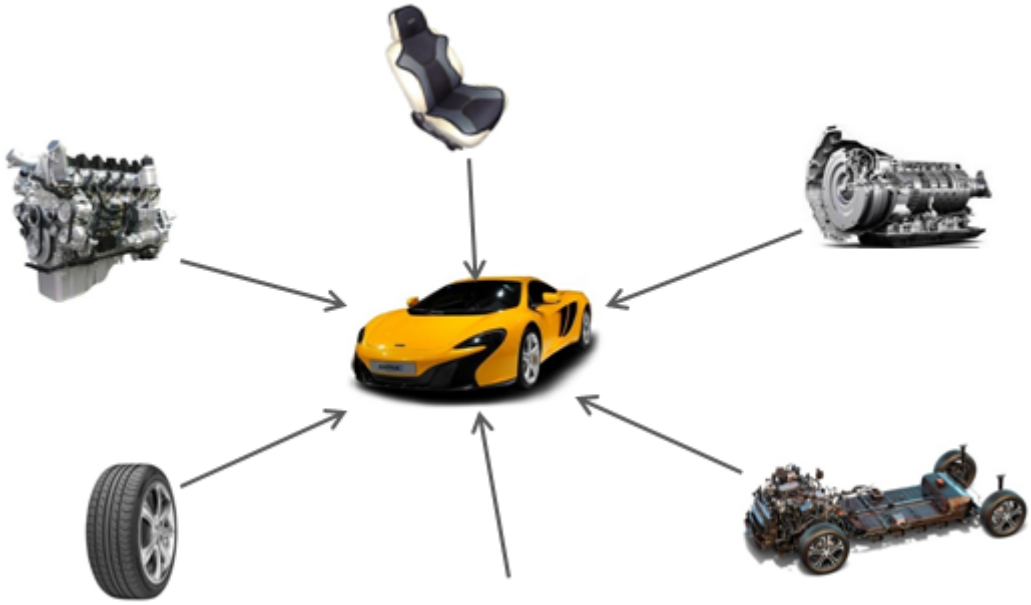
```
    <properties>
        <spring.version>5.1.5.RELEASE</spring.version>
        <springmvc.version>5.1.5.RELEASE</springmvc.version>
        <mybatis.version>3.5.1</mybatis.version>
    </properties>

  <!--锁定jar版本-->
    <dependencyManagement>
        <dependencies>
            <!-- Mybatis -->
            <dependency>
                <groupId>org.mybatis</groupId>
                <artifactId>mybatis</artifactId>
                <version>${mybatis.version}</version>
            </dependency>
            <!-- springMVC -->
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-webmvc</artifactId>
                <version>${springmvc.version}</version>
            </dependency>
            <!-- spring -->
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-context</artifactId>
                <version>${spring.version}</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
```

```
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-expression</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring.version}</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

2.3 maven聚合工程（分模块）

概念:

在现实生活中，汽车厂家进行汽车生产时，由于整个生产过程非常复杂和繁琐，工作量非常大，所以厂家都会将整个汽车的部件分开生产，最终再将生产好的部件进行组装，形成一台完整的汽车。



2.3.1 分模块构建maven工程分析

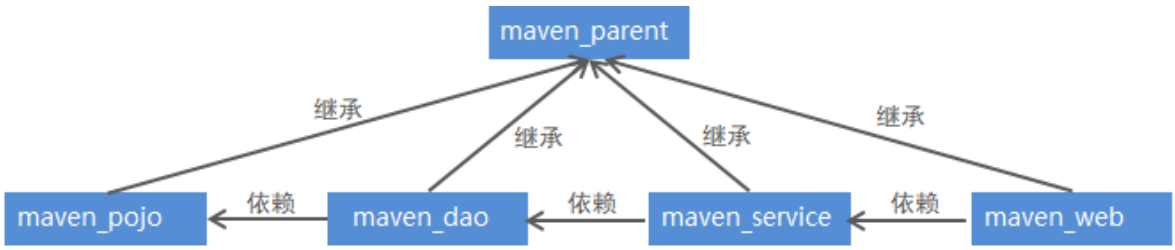
在企业项目开发中，由于项目规模大，业务复杂，参与的人员比较多，一般会通过合理的模块拆分将一个大型的项目拆分为N多个小模块，分别进行开发。而且拆分出的模块可以非常容易的被其他模块复用

常见的拆分方式有两种：

第一种：按照**业务模块**进行拆分，每个模块拆分成一个maven工程，例如将一个项目分为用户模块，订单模块，购物车模块等，每个模块对应就是一个maven工程

第二种：按照**层**进行拆分，例如持久层、业务层、表现层等，每个层对应就是一个maven工程

不管上面那种拆分方式，通常都会提供一个父工程，将一些公共的代码和配置提取到父工程中进行统一管理和配置。



2.3.2 maven工程的继承

在Java语言中，类之间是可以继承的，通过继承，子类就可以引用父类中非private的属性和方法。同样，在maven工程之间也可以继承，子工程继承父工程后，就可以使用在父工程中引入的依赖。继承的目的是为了消除重复代码。

在Java语言中，类之间是可以继承的，通过继承，子类就可以引用父类中非private的属性和方法。同样，在maven工程之间也可以继承，子工程继承父工程后，就可以使用在父工程中引入的依赖。继承的目的是为了消除重复代码。

被继承的Maven项目中的POM的部分定义是

```
<groupId>com.company</groupId>
<artifactId>company-project-parent</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>pom</packaging>
```

被继承的maven工程通常称为父工程，父工程的打包方式必须为pom，所以我们区分某个maven工程是否为父工程就看这个工程的打包方式是否为pom

继承的Maven项目中的POM的关键部分就是

```
<parent>
<groupId>com.taotao</groupId>
<artifactId>company-project-parent</artifactId>
<version>0.0.1-SNAPSHOT</version>
</parent>
<artifactId>company-project-children</artifactId>
```

继承其他maven父工程的工程通常称为子工程，在pom.xml文件中通过parent标签进行父工程的继承

2.3.3 maven工程的聚合

在maven工程的pom.xml文件中可以使用标签将其他maven工程聚合到一起，聚合的目的是为了进行统一操作。

例如拆分后的maven工程有多个，如果要进行打包，就需要针对每个工程分别执行打包命令，操作起来非常繁琐。这时就可以使用标签将这些工程统一聚合到maven父工程中，需要打包的时候，只需要在此工程中执行一次打包命令，其下被聚合的工程就都会被打包了。

```
<groupId>com.lagou</groupId>
<artifactId>lagou_edu_home_parent</artifactId>
<version>1.0-SNAPSHOT</version>
<modules>
    <module>ssm_domain</module>
    <module>ssm_utils</module>
    <module>ssm_dao</module>
    <module>ssm_service</module>
    <module>ssm_web</module>
</modules>
```

五个工程被聚合到一个工程

2.3.3 maven聚合工程_搭建拉勾教育后台管理系统

工程整体结构如下：

1）lagou_edu_home_parent为父工程，其余工程为子工程，都继承父工程lagou_edu_home_parent

2）lagou_edu_home_parent工程将其子工程都进行了聚合

3）子工程之间存在依赖关系：

ssm_domain依赖ssm_utils

ssm_dao依赖ssm_domain

ssm_service依赖ssm_dao

ssm_web依赖ssm_service