我们上一讲搭建了一个工程，而且重点是什么做了两件事情

第一件事情：交给了大家如何在一个项目中去设置企业级的坐标

第二件事情：在pom.xml里面加入了各种各样的依赖，搭建了一个spring+mybatis框架整合的这么一个模块，在模块里面也开发了一些功能，测试没有问题，打包没有问题

我们来重点讲解一下依赖这个事情，maven到底是如何去管理依赖的，依赖管理的机制是较为复杂的，这里面都有哪些学问，声明了那些<dependency>元素之后，maven干了哪些事情

1、用<dependency>可以引用任何你需要的依赖

<dependency>

<groupId></groupId>

<artifactId></artifactId>

<version></version>

<type></type>

<scope></scope>

<optional></optional>

</dependency>

在哪里找到你需要的依赖？

比如你感觉自己可能需要一个什么东西，你怎么知道要引入哪些依赖呢？

分两块来说，第一块，如果你是要用某个比较重要的技术，比如elasticsearch，spring，mybatis，等等，开源项目，直接到官网里面去，去找里面会告诉你如何在java里面引入maven依赖

另外一块，是这样的，可能在你开发的过程中，也许你还需要一些小公司，比如log4j，slf4j，jackson等等这种东西，连官网都用去了，mysql connector驱动。直接上百度

百度：maven + 你需要的依赖名称，比如log4j

下面两个方式，走国外的网站去搜索，一般反正我比较少用，速度较慢

sonatype nexus：http://repository.sonatype.org

mvnbrowser：http://www.mvnbrowser.com

2、依赖声明的三要素，坐标

groupId、artifactId、version，就ok了，type很少用

声明了三要素，一个坐标唯一定位了一个依赖的某个版本的jar包，maven会自动到远程的中央仓库里面去，给我们去找，下载到本地来，在打包的时候，就会自动使用，编译的时候

3、依赖范围

<scope></scope>

maven有三套classpath，classpath，就是项目中用到的各种依赖的类，jvm在运行的时候需要去classpath下面加载对应的类

maven在编译源代码的时候有一套classpath；在编译测试代码以及执行测试代码的时候，有一套classpath；运行项目的时候，有一套classpath；依赖范围就是用来控制依赖包与这三种classpath的关系的。

简单来说，不同的依赖范围，会导致那个依赖包可能在编译、测试或者打包运行的时候，有时候可以使用，有时候不能够使用

compile：默认，对编译、测试和运行的classpath都有效。一般都是用这种scope

test：仅仅对于运行测试代码的classpath有效，编译或者运行主代码的时候无效，仅仅测试代码需要用的依赖一般都会设置为这个范围，比如junit。一些测试框架，或者只有在测试代码中才会使用的一些依赖，会设置为test，这个的好处在于说，打包的时候这种test scope的依是不会放到最终的发布包里去的。减少发布包的体积。

provided：编译和测试的时候有效，但是在运行的时候无效，因为可能环境已经提供了，比如servlet-api，一般就是这个范围，在运行的时候，servlet容器会提供依赖。servlet-api是用来开发java web项目的，可能你在开发代码和执行单元测试的时候，需要在pom.xml里面声明这个servlet-api的依赖，因为要写代码和测试代码。但是最终打完包之后，放到tomcat容器里面去跑的时候，是不需要将这个servlet-api的依赖包打入发布包中的，因为tomcat容器本身就会给你提供servlet-api的包。

runtime：测试和运行classpath有效，但是编译代码时无效，比如jdbc的驱动实现类，比如mysql驱动。因为写代码的时候是基于javax.sql包下的标准接口去写代码的。然后在测试的时候需要用这个包，在实际运行的时候才需要用这个包的，但是编译的时候只要javax.sql接口就可以了，不需要mysql驱动类。一般我们声明mysql驱动的时候，不会设置为runtime，因为也许你开发代码的时候会用到mysql驱动特定的api接口，不仅仅只是用javax.sql。

3、传递性依赖

每个依赖可能又有其他的依赖，其他依赖又有其他的依赖，循环往复

纯手工时期，可能就是先加入第一层依赖，然后报错；接着加入第二层依赖，再报错，继续加入第三层依赖；循环往复，极其麻烦

在最初期，10年以前，做java项目的时候，用类似maven的工具相对较少，一般很多国内的项目，都是手动直接放一大堆jar包。先找一堆spring的jar包，然后尝试运行，发现报错，说缺失某某类，再去找那个项目的jar包，加进来，又说缺失某某类，循环往复。

maven的传递性依赖，就是说会自动递归解析所有的依赖，然后负责将依赖下载下来，接着所有层级的依赖，都会成为我们的项目的依赖，不需要我们手工干预。所有需要的依赖全部下载下来，不管有多少层级。这个就是maven的传递性依赖机制，自动给我递归依赖链条下载所有依赖的这么一个特性。

好比说我们依赖了junit，junit依赖了A，A又依赖了B

我们对junit的依赖范围是test，junit对A的依赖范围是compile，那么我们对A的依赖范围是什么呢？ -> test

比如说，我们依赖于A，是compile；A依赖于B，是test；我们对B的依赖范围是空，就是我们不会去依赖B，因为你自己想想都知道，A只有在测试的时候才会使用B。我们依赖A是生产用的，我们去依赖B干嘛？B是给A测试的。

传递性依赖机制对依赖范围也是有影响的，比如下面的表格，第一列是一级依赖，第一行是二级依赖，传递性依赖会导致多级依赖的依赖范围交叉在一起，会有影响：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | compile | test | provided | runtime |
| compile | compile |  |  | runtime |
| test | test |  |  | test |
| provided | provided |  | provided | provided |
| runtime | runtime |  |  | runtime |

有可能依赖是不会传递的，就是可能有些多层级的依赖，是不会成为我们项目的依赖的

4、依赖调解

传递性依赖深入去讲解的，既然说maven会自动解析所有层级的依赖，给我们自动下载所有的依赖，但是可能会出现依赖冲突的问题

比如A->B->C->X(1.0)，A->D->X(2.0)，A有两个传递性依赖X，不同的版本

就产生了依赖冲突的问题，maven如何解决呢？依赖调解的机制

此时就会依赖调解，就近原则，离A最近的选用，就是X的2.0版本

如果A->B->X(1.0)和A->D->X(2.0)，路径等长呢？那么会选择第一声明原则，哪个依赖在pom.xml里先声明，就用哪个

5、可选依赖

<optional>true</optional>

此时依赖传递失效，不会向上传递

如果A依赖于B，B依赖于C，B对C的依赖是optional，那么A就不会依赖于C。反之，如果没有optional，根据传递性依赖机制，A会依赖于C。

比如项目B是一个通用的数据访问平台，可以接elasticsearch，可以接hbase，可以接redis，有几个依赖，但是这些依赖可能是不可以同时使用的，此时就可以声明为optional不要向上传递

那么如果项目A依赖项目B，此时就需要根据自己的需求，引入elasticsearch/hbase/redis的依赖，让B可以跑起来。比如说，A就依赖于使用B来访问elasticsearch，那么A自己就引入elasticsearch的依赖即可。B对elasticsearch，hbase，redis的依赖是不会自动传递给A的，需要A自己去声明。

这个很少很少用

总结一下

（1）声明依赖

（2）找依赖，两个方案去找到你需要的依赖的maven坐标

（3）找到依赖以后配置依赖，三要素，坐标

（4）依赖范围，根据对依赖的使用场景，比如仅仅在test环境下使用，或者在运行时由运行环境来提供，可以给依赖设置不同的依赖范围

（5）传递性依赖，自动解析所有层级的依赖

（6）传递性依赖的多层级的依赖范围，交叉影响的结果

（7）依赖调解，碰到一个项目多个版本，启动根据两大就近原则来选择一个最合适的版本来使用

（8）可选依赖，不向上传递

853769620