maven的使用

本节目标

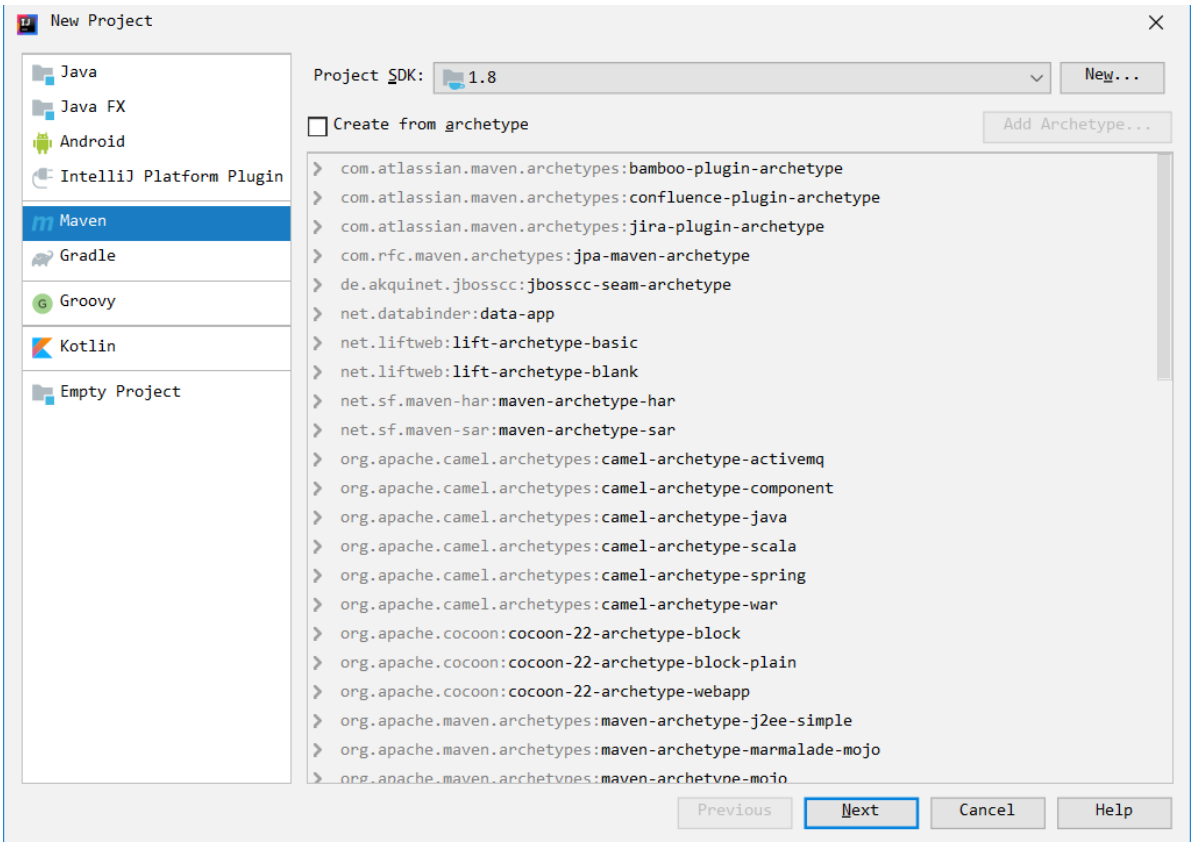
> - 了解 jar 包的基本作用和使用方式
> - 理解 maven 的作用
> - 掌握 maven 的使用方式
> - 理解类的加载是什么

1 maven 是什么

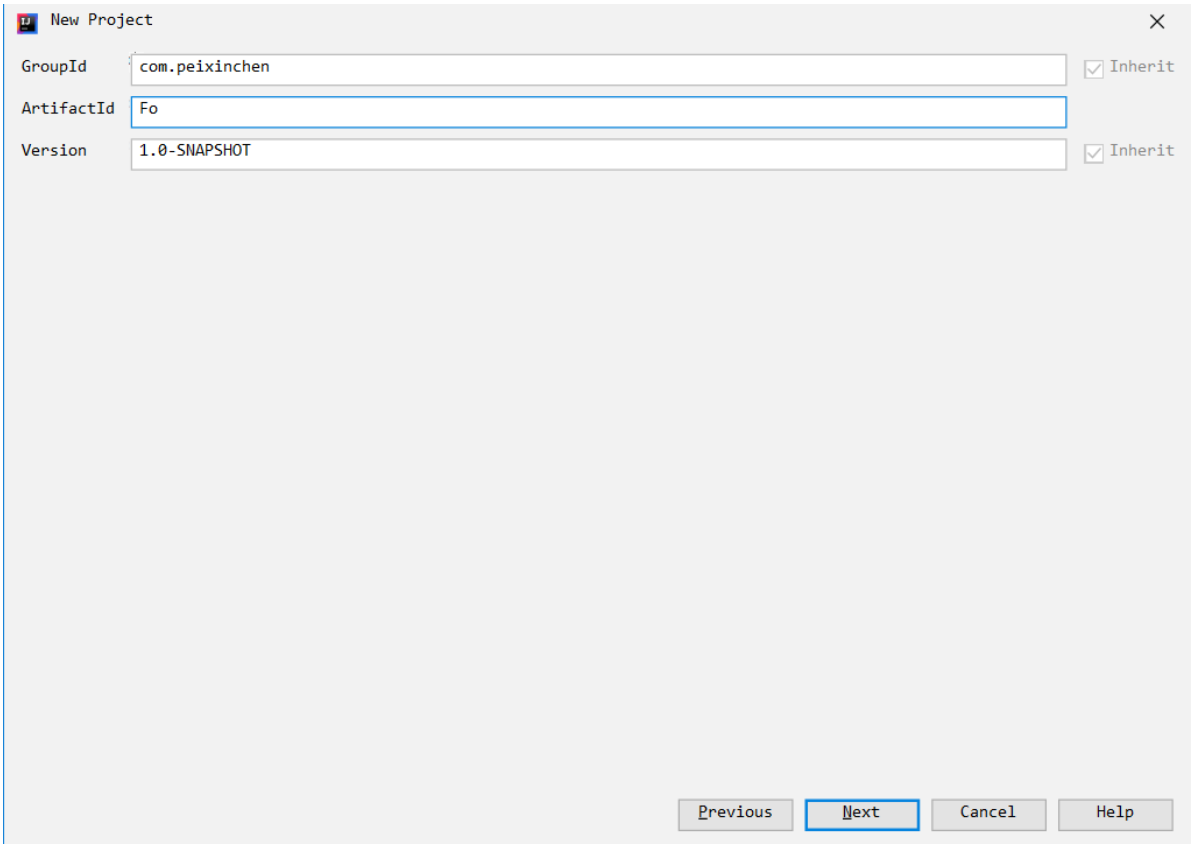
Apache Maven 是一种用于软件项目管理工具，基于 Project Object Model（POM），用来管理项目的构建，汇报及文档生成等功能。

2 示例: 通过 IDEA 创建 maven 项目

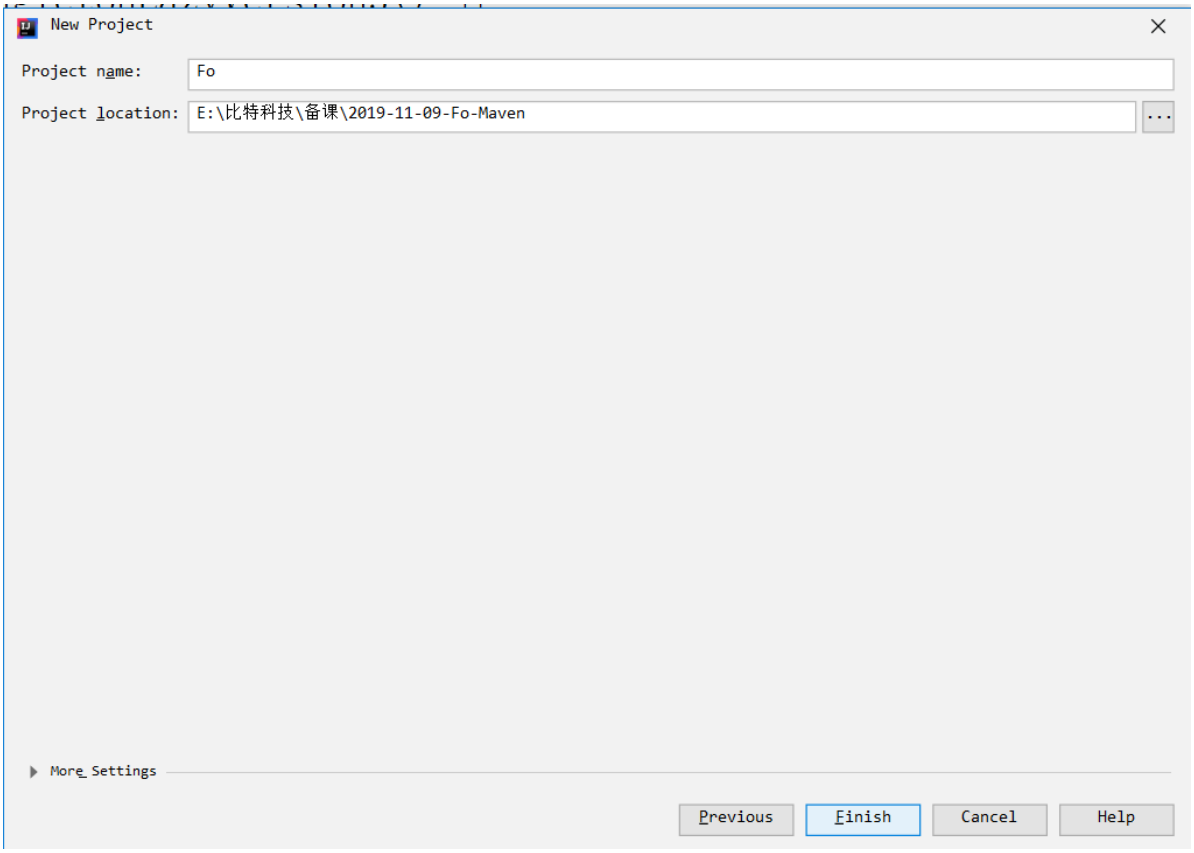
新建 maven 类型的项目



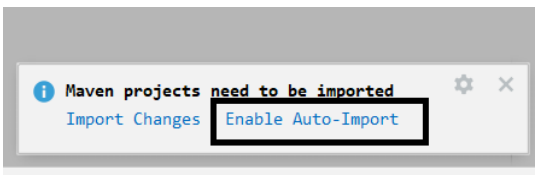
输入合适的 GroupId 和 ArtifactId，一般 GroupId 代表的是机构名称，我们自己使用可以使用 com.自己名字 代替，ArtifactId 描述这个项目的名字，给出合适的名字即可。



选择项目路径



开启自动导入功能



至此项目新建完成。

观察项目的文件夹结构

```
Fo\
    src\
        main\
            java\
            resources\
        test\
            java\
    pom.xml
```

其中

我们的代码一般在 src\main\java 文件夹下，跟着我们的包名即可。

src\main\resources 下一般是用于同时部署的一些资源文件，例如图片、音频、视频等

src\test\java 一般用来放一些测试代码

pom.xml 为 maven 最重要的文件，是 maven 的 配置描述文件。

3 maven 的配置文件——pom.xml

pom.xml 文件后缀名表示这个文件是用 XML 格式进行组织。

什么是 XML 文件呢，简单的去理解，是一种类似我们学习过的 HTML 格式的文件，全程 Extensible Markup Language，Java 语言中经常会用 XML 用来做配置管理。

具体的标签大家可以去 POM 进行详细了解。

下面我把我的 pom.xml 给出，并做适当的讲解，以后大家在使用过程中可以直接从之前的项目中复制粘贴内容即可。

```
<?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">
    <!-- 上面的内容完全不用管，使用自动生成的就行，是用于一些校验规则指定的 -->

    <!-- 这里指定的是 POM 的版本，也不需要动 -->
    <modelVersion>4.0.0</modelVersion>

    <!-- 这里是项目的描述信息，是新建项目时我们填入的内容 -->
    <groupId>com.peixinchen</groupId>
    <artifactId>Fo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <!-- 一般把我们需要的内容附加在这下面 -->

    <!-- 可以配置一些参数 -->
    <properties>
        <!-- 默认情况下，maven 会使用 1.5 版本进行代码检查，一般我们都修改为 1.8 -->
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

</project>
```

XML 中可以用 `<!-- 这里是注释 -->` 的方式来进行注释

4 依赖管理

什么是 maven 仓库(maven repository)

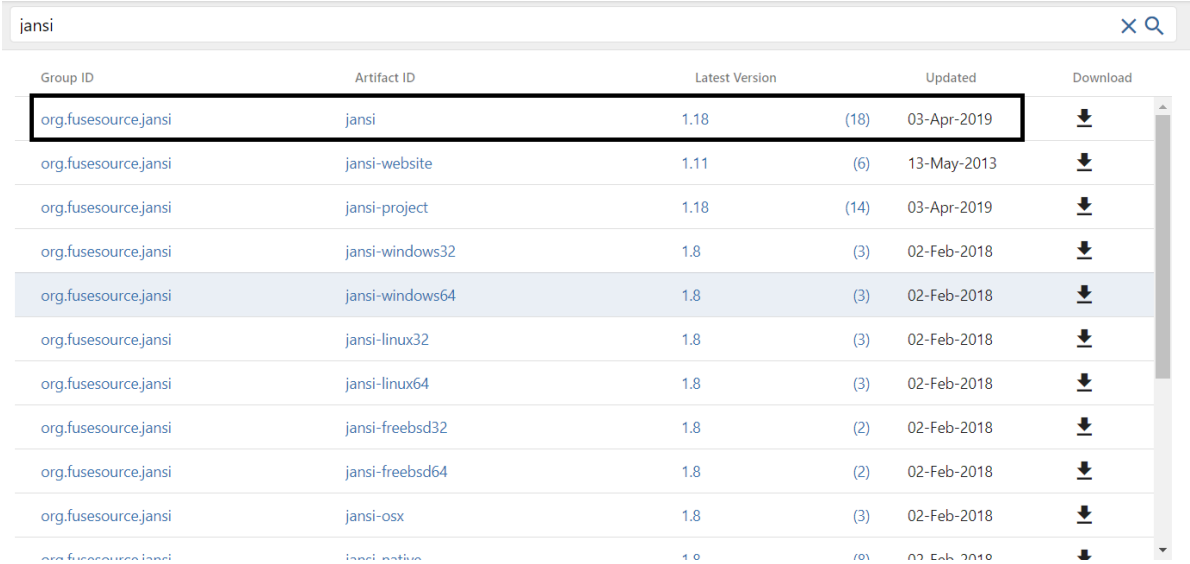
maven 仓库是一个类似手机上的 App Store 的东西，上面会有全世界的人上传的各种第三方的 jar 包供我们使用，当我们的项目需要用到其中的内容，可以像手机安装 app 一样方便的进行查找和按照。

maven 仓库

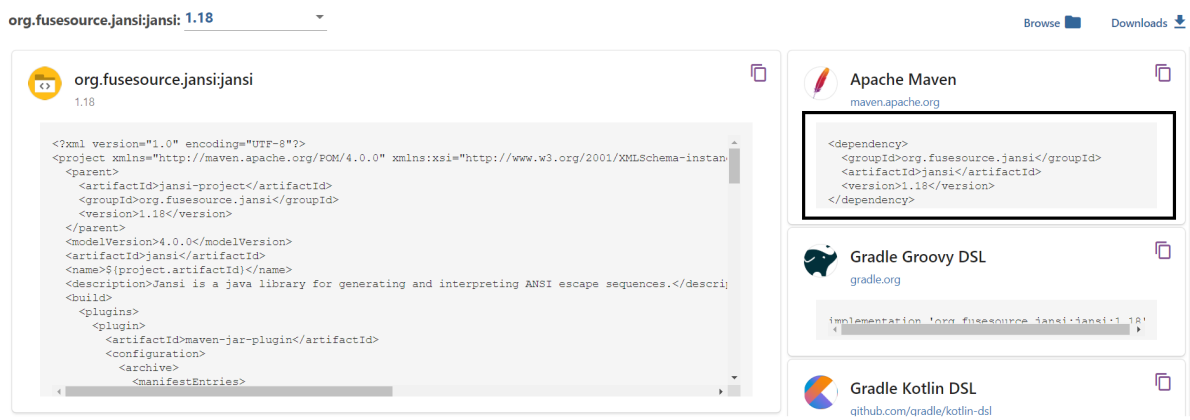
其中我们的项目用到了另一个项目，叫做依赖关系。

而一个项目中往往需要很多的依赖，所以诞生了依赖管理的概念。

搜索我们的 jansi 依赖



点击具体的版本号后，可以看到这个类库的详情页面



左侧可以进行具体的版本号选择，至于如何选择版本号，还是一个比较复杂的话题，目前我们保持紧跟课程的版本即可。

右边框出的内容，就是我们要添加到 pom.xml 上的依赖配置，可以直接使用右侧的复制图标进行内容复制，完成后我们的 pom.xml 变成

```
<?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">
    <!-- 上面的内容完全不用管，使用自动生成的就行，是用于一些校验规则指定的 -->

    <!-- 这里指定的是 POM 的版本，也不需要动 -->
    <modelVersion>4.0.0</modelVersion>

    <!-- 这里是项目的描述信息，是新建项目时我们填入的内容 -->
    <groupId>com.peixinchen</groupId>
    <artifactId>Fo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <!-- 一般把我们需要的内容附加在这下面 -->

    <!-- 可以配置一些参数 -->
    <properties>
        <!-- 默认情况下，maven 会使用 1.5 版本进行代码检查，一般我们都修改为 1.8 -->
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <!-- 这个标签中指定所有的依赖项 -->
    <dependencies>
        <!-- 这里指定了我们需要的依赖 -->
        <dependency>
            <groupId>org.fusesource.jansi</groupId>
            <artifactId>jansi</artifactId>
            <version>1.18</version>
        </dependency>
    </dependencies>

</project>
```

添加完成后，IDEA 中的 maven 会自动帮我们进行依赖 jar 包的下载，所以这个时候我们需要保证**网络连接是可用的。**

如果需要手动下载，大家可以在文件的空白处，点击鼠标右键，选择 maven -> reimport 进行重新导入。

至此，我们的依赖管理就配置完成了。

依赖管理，可以说是目前阶段，我们使用 maven 的最重要的目的了，所以大家有问题一定要解决。

课堂练习： 大家可以尝试把我们之前 JDBC 依赖的 jar 包，使用 maven 的方式，加入到我们项目中，搜索的名称是 `mysql-connector-java`

> 另外，仓库还有个含义是本地仓库，主要指的是我们下载下的依赖存放的位置，通常我们不需要关注。

5 完成代码编写

在 src\main\java 文件夹下创建 Main.java

```
import org.fusesource.jansi.AnsiConsole;

import java.util.Scanner;

import static org.fusesource.jansi.Ansi.*;
```

```
import static org.fusesource.jansi.Ansi.Color.*;

public class Main {
    private static final String[] fo = {
        "                     _ooOoo",
        "                    o8888888o",
        "                    88\" . \"88",
        "                    (| -_- |)",
        "                     O\\ = /O",
        "                 ____/`---'\\\",
        "               .   ' \\\\| |// `.",
        "                / \\\\||| : ||// \\\",
        "              / _||||| -:- |||||- \\\",
        "                | | \\\\\\ - /// | |",
        "              | \\_| ''\\---/'' | |",
        "               \\ .-\\__ `-` ___/-. /",
        "            ___`. .' /--.--\\ `. . __",
        "         .\"\" '< `.___\\_<|>_/___.' >'\"\".",
        "        | | : `- \\`.;`\\ _ /`;.`/ - ` : | |",
        "          \\ \\ `-. \\_ __\\ /__ _/ .-` / /",
        "======`-.____`-.___\\_____/___.-`____.-'======",
        "                     `=---='"
    };

    private static void printFO(Color color) {
        System.out.println(ansi().eraseScreen());
        for (String line : fo) {
            System.out.println(ansi().fg(color).a(line).reset());
        }
        Scanner scanner = new Scanner(System.in);
        scanner.nextLine();
    }

    public static void main(String[] args) {
        AnsiConsole.systemInstall();
        printFO(RED);
        printFO(BLUE);
        printFO(YELLOW);
        printFO(GREEN);
        printFO(CYAN);
        printFO(WHITE);
        AnsiConsole.systemUninstall();
    }
}
```

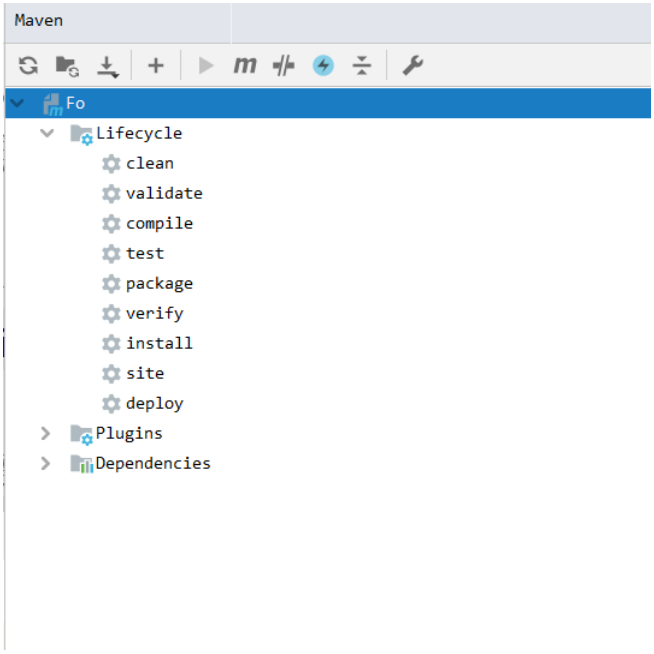
5.1 构建生命周期

通常，我们理解的工程构建，可以被区分成不同的生命周期（Lifecycle）和阶段（Phase）。

其中 maven 把各个阶段都做了各自的映射。

我们重点了解以下阶段即可。

1. compile 编译阶段
2. test 测试阶段
3. package 打包阶段
4. deploy 部署阶段



我们可以点击 package 进行打包，成功后，项目的 target 文件夹下会生成 Fo-1.0-SNAPSHOT.jar 包。

但这个 jar 包是不带 Main-Class 的 jar 包，即无法直接运行。

依赖管理时，可以指定一个依赖被用于哪个阶段，例如 junit 作为一种著名的单元测试框架，用于测试阶段，后面的阶段就不再需要了。

5.2 插件

maven 同时还提供了开放的插件开发功能，可以提供给大牛们进行构建过程中方便功能的开发，这里我们针对性的使用其中一种插件，可以打包带 Main-Class 的 jar 包。

修改 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">
    <!-- 上面的内容完全不用管，使用自动生成的就行，是用于一些校验规则指定的 -->

    <!-- 这里指定的是 POM 的版本，也不需要动 -->
    <modelVersion>4.0.0</modelVersion>

    <!-- 这里是项目的描述信息，是新建项目时我们填入的内容 -->
    <groupId>com.peixinchen</groupId>
    <artifactId>Fo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <!-- 一般把我们需要的内容附加在这下面 -->

    <!-- 可以配置一些参数 -->
    <properties>
        <!-- 默认情况下，maven 会使用 1.5 版本进行代码检查，一般我们都修改为 1.8 -->
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <!-- 这个标签中指定所有的依赖项 -->
```

```
    <dependencies>
        <!-- 这里指定了我们需要的依赖 -->
        <dependency>
            <groupId>org.fusesource.jansi</groupId>
            <artifactId>jansi</artifactId>
            <version>1.18</version>
        </dependency>
    </dependencies>

    <!-- 一般我们把构建相关的配置放这里 -->
    <build>
        <!-- 使用各种插件 -->
        <plugins>
            <!-- 这个插件的目的是帮我们把依赖复制到 target\lib 文件夹下，用于一会打 jar 包
使用 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>3.1.1</version>
                <executions>
                    <execution>
                        <id>copy-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>copy-dependencies</goal>
                        </goals>
                        <configuration>

 <outputDirectory>${project.build.directory}/lib</outputDirectory>
                            <includeScope>runtime</includeScope>
                        </configuration>
                    </execution>
                </executions>
            </plugin>

            <!--  这个插件是用于打 jar 包的 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <!-- 这里指定 Main-Class -->
                            <mainClass>Main</mainClass>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>lib/</classpathPrefix>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

完成后再次进行 package 打包。

这次生成的 jar 包就可以直接运行了。

5.3 maven 的作用

maven 的目标是完成项目构建解决的一切繁琐事宜。我们具体关注它的以下功能：

1. 提供一个标准的项目工程目录
2. 提供项目描述
3. 提供强大的版本管理工具
4. 可以分阶段的进行构建过程
5. 提供了丰富的插件库使用

5.4 实践：配置更快速的 maven 仓库

通常，默认的仓库因为网络原因，下载都比较慢，大家可以把自己的仓库更新为阿里的版本。