

# 3. SpringBoot 快速上手

## 本节目标

1. 了解Maven,并配置国内源
2. 使用SpringBoot创建一个项目, 输出HelloWorld

## 1. 环境准备

自检Idea版本:

社区版: 2021.1 -2022.1.4

专业版: 无要求

如果个人电脑安装的idea不在这个范围, 需要卸载重新安装.

Idea 卸载参考: [https://blog.csdn.net/qq\\_19072921/article/details/126408402](https://blog.csdn.net/qq_19072921/article/details/126408402) (一定要删除注册表)

## 2. Maven

### 2.1 什么是Maven

官方对于Maven的描述:

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

引用来自: <https://maven.apache.org/index.html>

翻译过来就是:

Maven是一个项目管理工具。基于POM(Project Object Model,项目对象模型)的概念，Maven可以通过一小段描述信息来管理项目的构建，报告和文档的项目管理工具软件。

大白话: Maven是一个项目管理工具, 通过pom.xml文件的配置获取jar包，而不用手动去添加jar包

### 2.2 为什么要学Maven

一句话: 简单, 方便, 提高我们的开发效率, 减少我们的开发Bug.

Maven提供的功能非常多, Maven在咱们课程中的体现主要是以下两个方面:

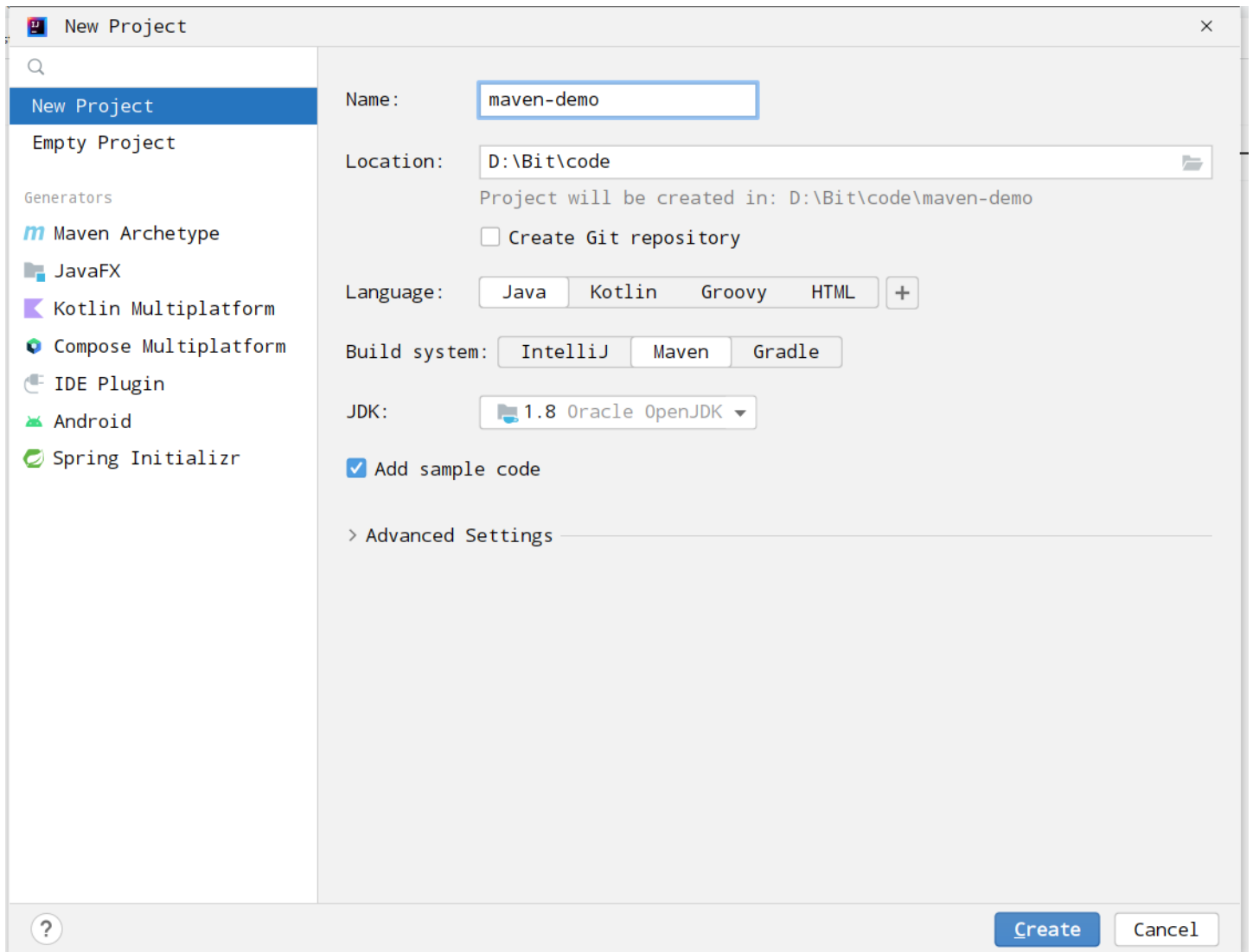
1. 项目构建
2. 管理依赖

## 2.3 创建一个Maven项目

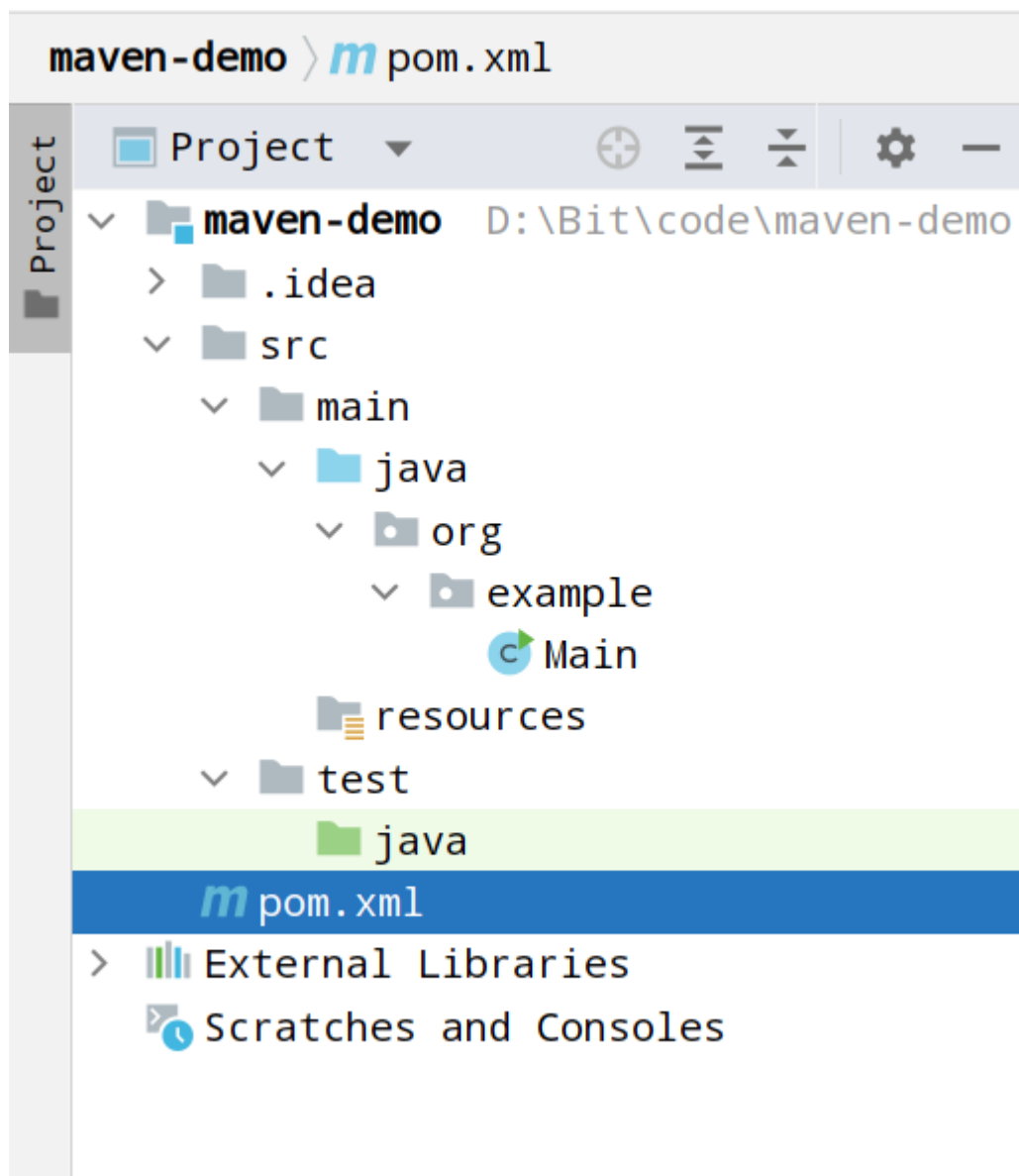
IDEA本身已经集成了Maven, 我们可以直接使用, 无需安装

以下截图的idea版本为: 2022.1.4, 不同版本的idea界面展示会有所不同

File -> New -> Project



点击 Create, 就创建好了一个Maven项目



## 2.4 Maven 核心功能

接下来, 我们结合项目, 介绍Maven在项目开发中的作用.

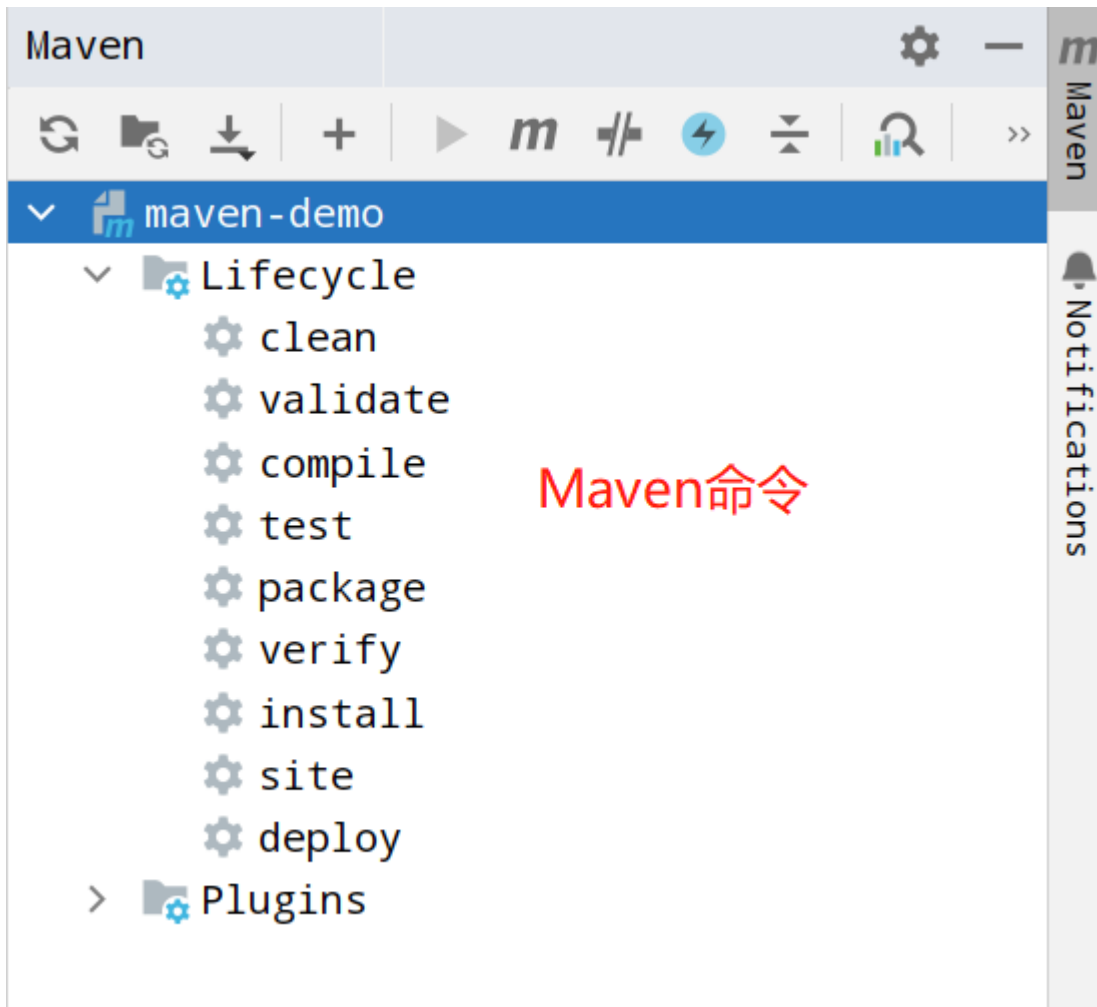
主要体现在两个方面:

1. 项目构建
2. 管理依赖

### 2.4.1 项目构建

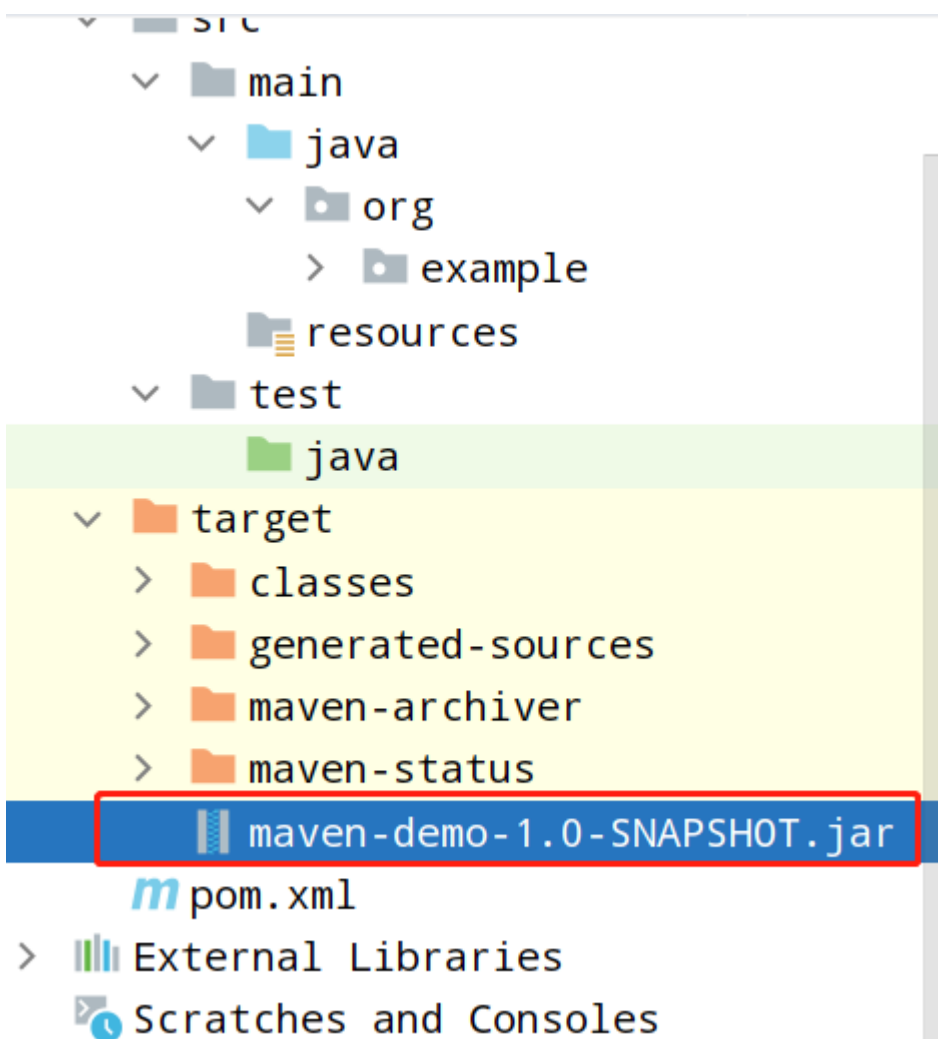
Maven 提供了标准的,跨平台(Linux, Windows, MacOS等)的自动化项目构建方式

当我们开发了一个项目之后, 代码需要经过编译, 测试, 打包, 发布等流程, 每次代码的修改, 都需要经过这些流程, 如果代码反复调试修改, 这个流程就需要反复进行, 就显得特别麻烦,, 而Maven 给我们提供了一套简单的命令来完成项目的构建.



比如, 点击package, 就可以完成项目的打包操作

```
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ maven-demo ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ maven-demo ---
[INFO] Building jar: D:\Bit\code\maven-demo\target\maven-demo-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.394 s
[INFO] Finished at: 2023-06-12T18:55:31+08:00
[INFO] -----
```



打包就是把所有的class文件, 全部放在一起, 打成jar包或者war包

jar包和war包都是一种压缩文件

jar包就是把开发人员已经写好的一些代码进行打包. 打好的jar包就可以引入到其他项目中, 也可以直接使用这些jar包中的类和属性. 另外也可以打成可执行jar包, 这样的包就可以通过java -jar命令来执行

war包可以理解为是一个web项目, 里面是项目的所有东西, 通常用于网站.

## 2.4.2 依赖管理

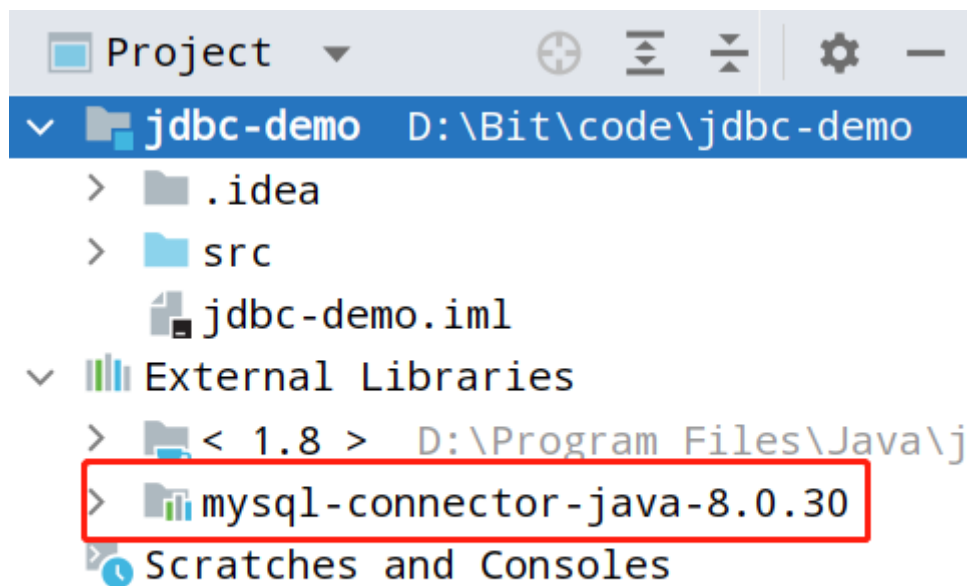
上面说到, Maven是一个项目管理工具, 通过pom.xml文件的配置获取jar包, 而不用手动去添加jar包. 获取的jar包, 其实就是依赖.

pom.xml 就是maven 的配置文件, 用以描述项目的各种信息

### 依赖配置

依赖: 指当前项目运行所需要的jar包.

比如前面学习JDBC时, 我们需要手动下载mysql-connector-java的包. 并且添加到项目中.



如果使用Maven, 我们只需要在pom.xml中引入mysql-connector-java的依赖就可以了

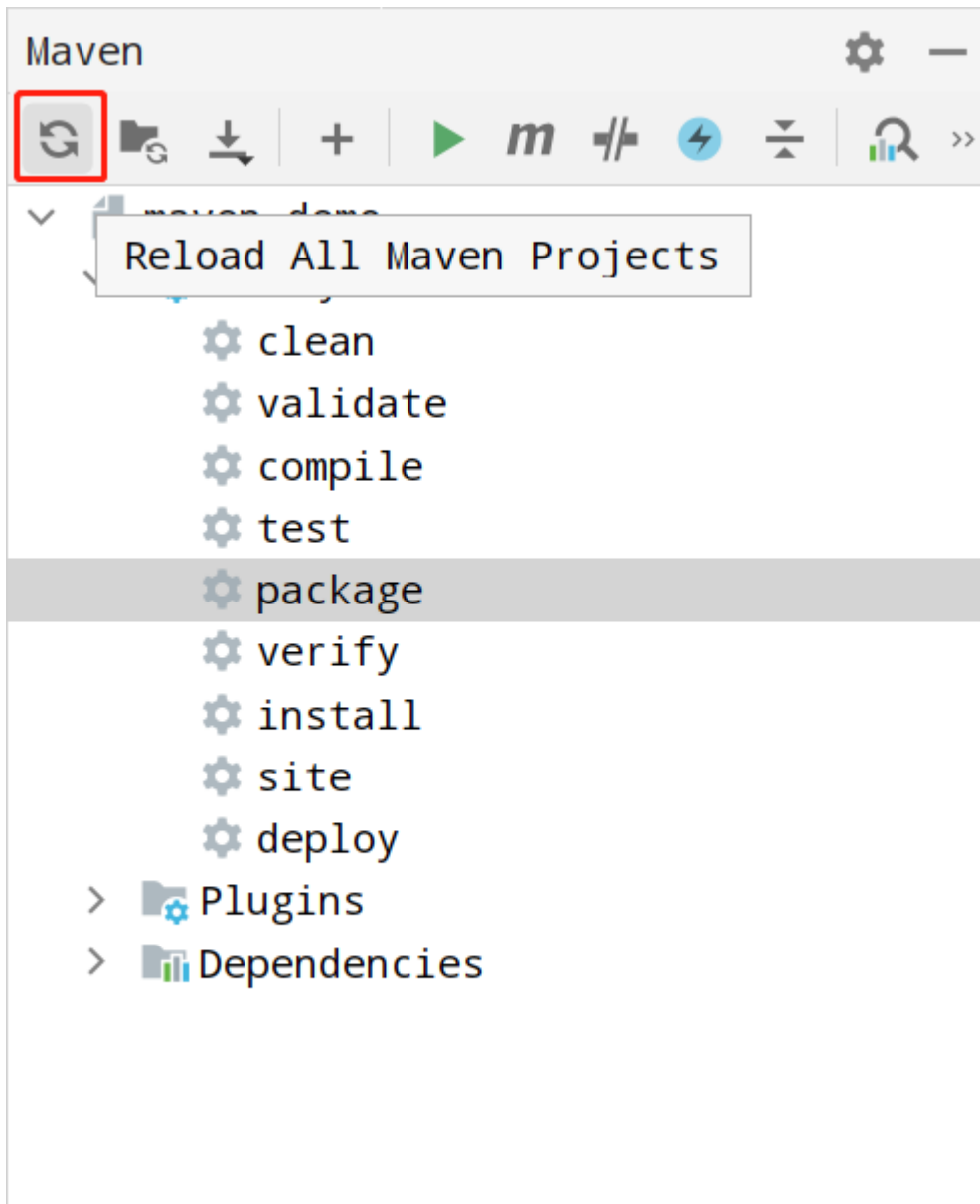
```
1 <dependencies>
2     <!--里面放置项目的依赖坐标, 可为多个 -->
3 </dependencies>
```

1. 在pom文件 `<dependencies>` 标签内, 添加依赖坐标

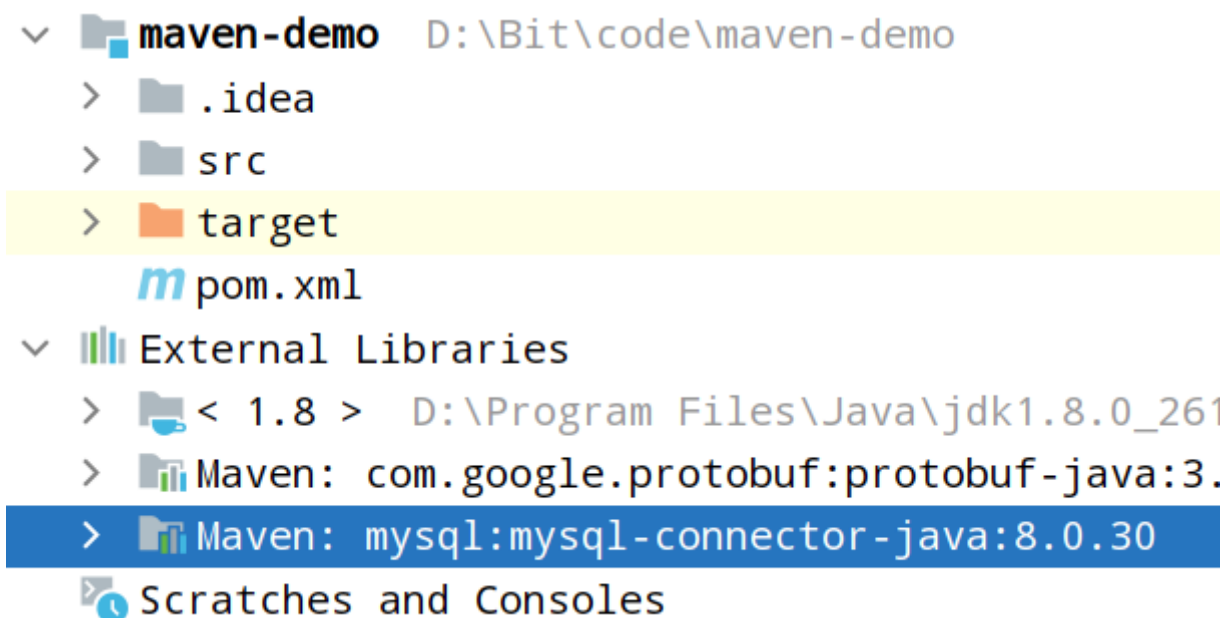
```
1 <dependency>
2     <groupId>mysql</groupId>
3     <artifactId>mysql-connector-java</artifactId>
4     <version>8.0.30</version>
5 </dependency>
```

2. 点击刷新按钮, 引入新加入的依赖jar包

后续有添加新的jar包, 或者修改jar包版本, 都需要通过该方式在项目中添加依赖.



3. 刷新完之后, 就可以在项目中看到新加入的jar包



依赖传递

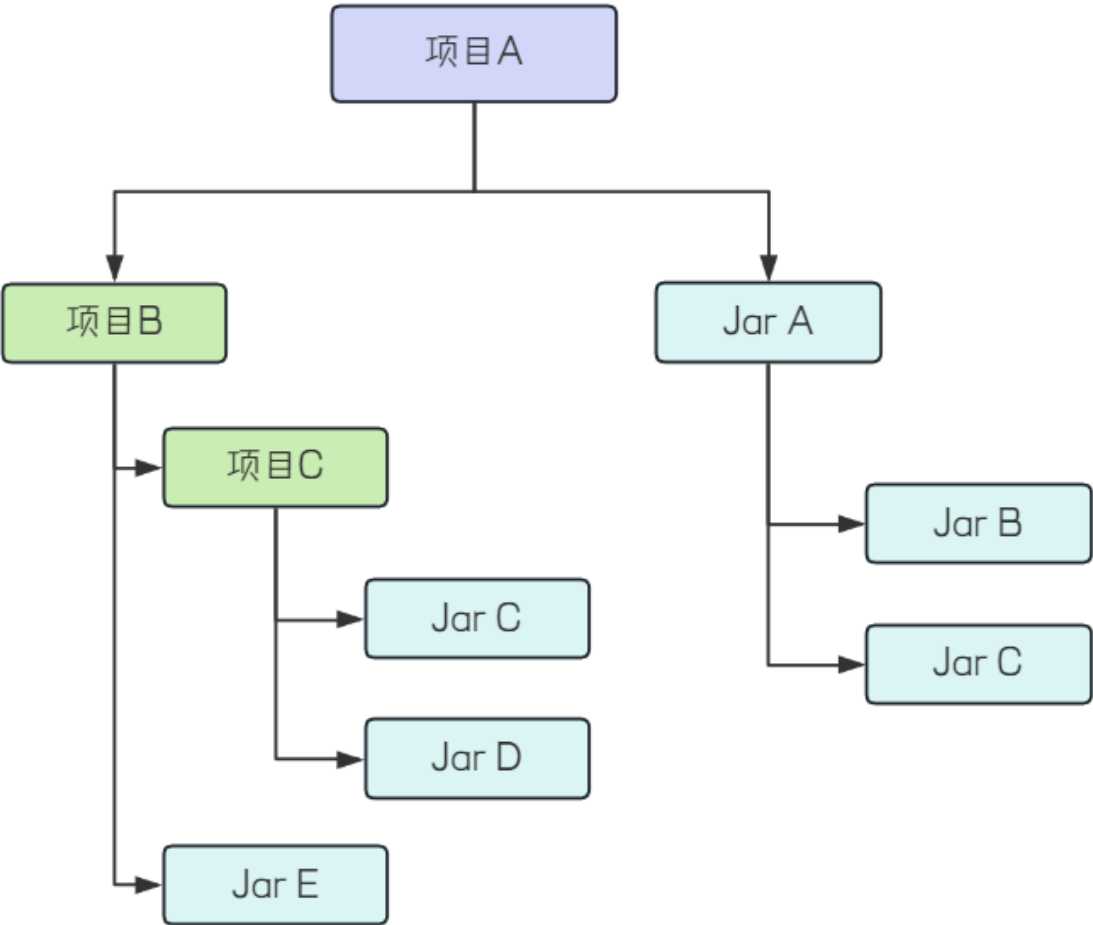
早期我们没有使用maven时, 向项目中添加依赖的jar包, 需要把所有的jar包都复制到项目工程下.

比如 A 依赖B, B依赖C, 那么 A项目引入B 的同时, 也需要引入C, 如果我们手动管理这个依赖, 这个过程就会比较麻烦, 我们需要知道每个库都依赖哪些库, 以及这些依赖之间的版本是如何关联的

比如我们要吃火锅, 需要有锅, 有调料, 有食材, 以及确认什么样的锅, 什么样的食材.  
比如去医院看病, 需要带上以往的病历, 检查结果, 处方等, 并且要确认带的资料是正确的, 如果日期错了, 或者患者错了, 带少了, 就需要回去重新拿.

但使用maven的话, 就可以避免管理所需依赖的关系。我们只需要在pom文件中, 定义直接依赖就可以了, 由于maven的依赖具有传递性, 所以会自动把所依赖的其他jar包也一起导入

比如吃火锅, 现在我们可以点一个海底捞外卖, 直接就把所有食材都送过来了, 包括什么锅, 配什么菜.  
比如去医院看病, 借助"互联网", 实现了信息共享, 只需要带上身份证, 以往的病历和检查结果就都可以看到了.



如上图, 项目A通过Maven 引入 Jar A 时, 会自动引入 Jar B 和Jar C.

Jar A 和项目B就是项目A的直接依赖.

Jar B, Jar C是间接依赖.

直接依赖：在当前项目中通过依赖配置建立的依赖关系



间接依赖：被依赖的资源如果依赖其他资源，当前项目间接依赖其他资源

## 依赖排除

当前阶段我们需要依赖的库并不多,但随着项目的越来越复杂,库之间的依赖关系也会变得越来越复杂.

如上图中,如果项目A不需要Jar B,也可以通过排除依赖的方式来实现.

排除依赖:

指主动断开依赖的资源。（被排除的资源无需指定版本）

比如,我下了一个快递单子,默认会有一个服务,就是快递小哥会上门取件,但是我刚好要出门,而且顺路经过站点,也可以选择自己送过去

```
1 <dependency>
2   <groupId>org.springframework</groupId>
3   <artifactId>spring-core</artifactId>
4   <version>6.0.6</version>
5
6   <!--排除依赖-->
7   <exclusions>
8     <exclusion>
9       <artifactId>spring-jcl</artifactId>
10      <groupId>org.springframework</groupId>
11    </exclusion>
12  </exclusions>
13 </dependency>
```

maven还有一些功能是依赖调解,可选依赖等

依赖调解:

当项目中的依赖存在依赖冲突时,例如 存在这样的依赖:

A->B->C->X(1.0)

A->D->X(2.0)

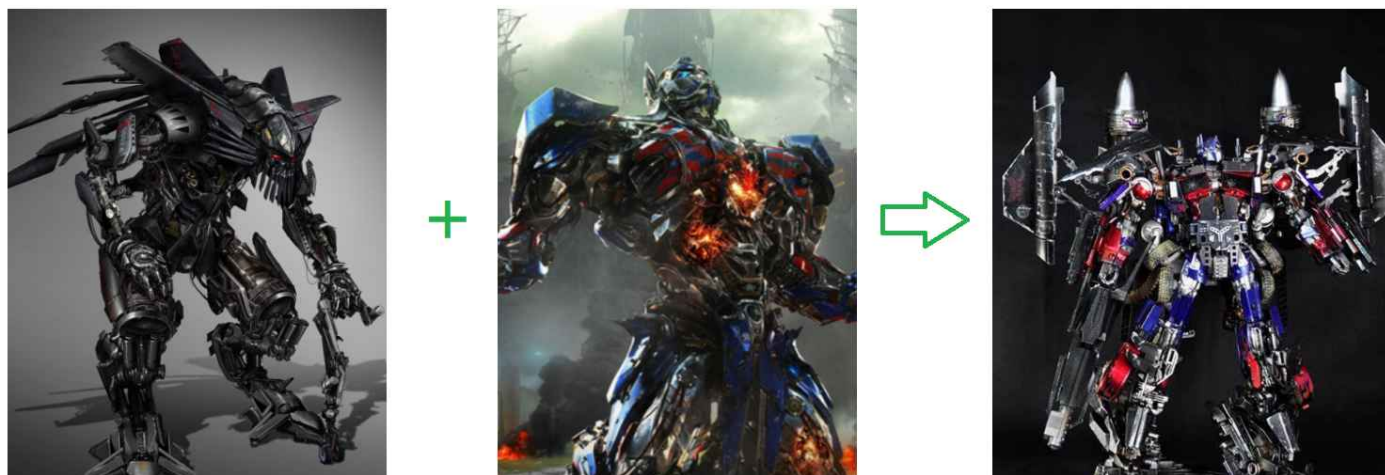
Maven会采用最短路径优先的原则去选择依赖,这里2的依赖路径更短,所以会选择X(2.0),当然我们也可以选择指定某个依赖的版本

当然也可以指定X的版本

## 1.4.3. Maven Help插件

当项目比较复杂时, 我们就会有Jar包冲突的问题, 这时候就需要去解决依赖冲突. 解决冲突之前, 需要先找到冲突, 我们可以使用Maven Help插件来观察包和包之间的依赖关系.

理解 "插件" (plugin)



天火 + 擎天柱 => 会飞的擎天柱.

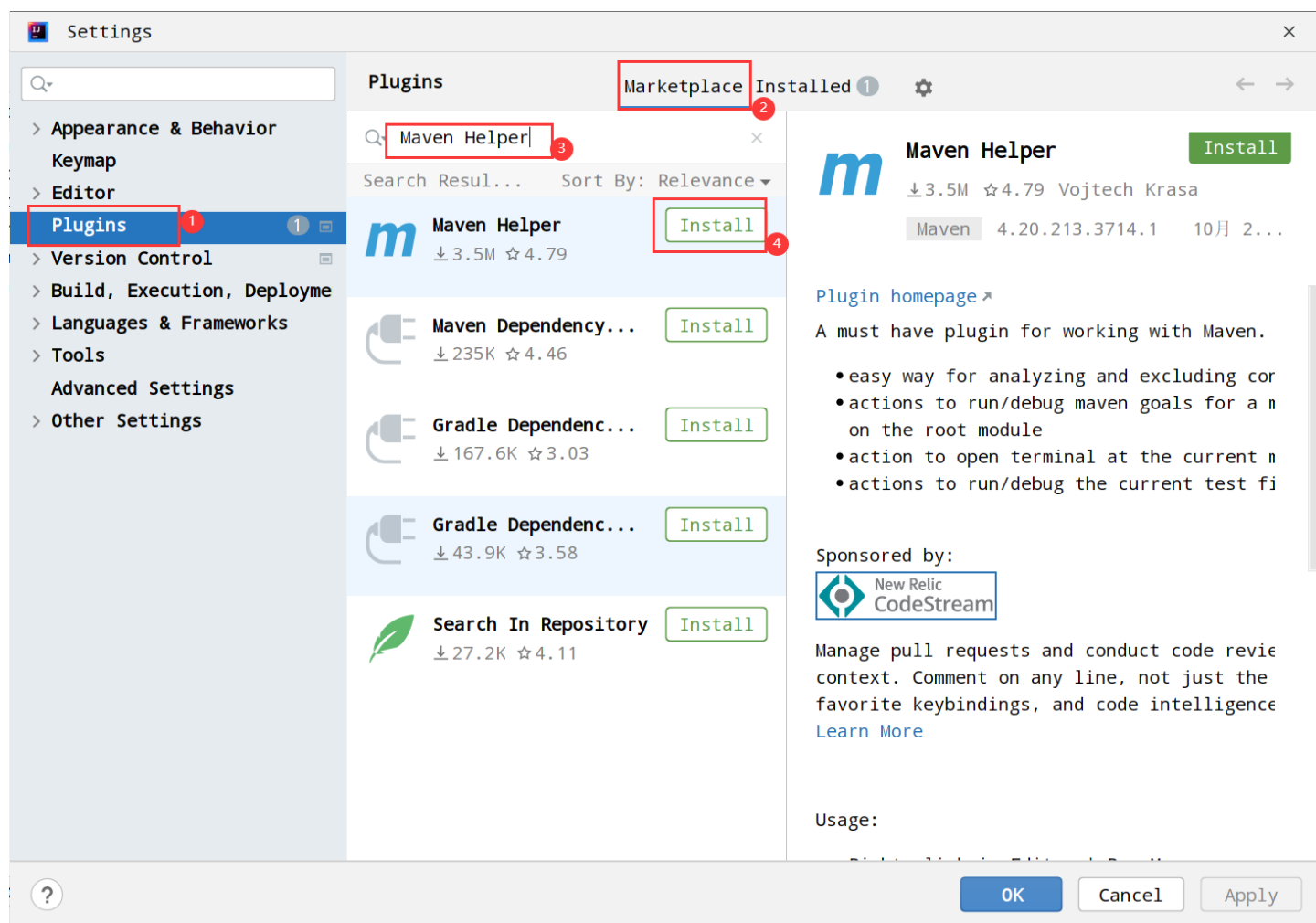
天火在牺牲之前把自己变成了擎天柱的 "飞行插件". 在擎天柱需要起飞的时候就变成翅膀装在擎天柱身上. 不需要起飞的时候就卸下来放到擎天柱的集装箱里.

程序开发的时候也经常如此.

像 IDEA 这样的程序虽然功能强大, 但是也无法面面俱到. 对于一些特殊场景的功能, 开发者就可以开发一些 "插件". 如果需要这个插件, 就单独安装.

**插件就是对程序的一些特定场景, 做出一些特定的功能的扩展.**

安装插件: File -> Settings -> Plugins -> 搜索'Maven Help' -> 找到对应插件, 点击Install 安装即可, 安装后需要重启下idea, 才能生效



安装之后, 打开pom文件, 可以看到Jar包之间的依赖关系

m pom.xml (maven-demo) x

Refresh UI Reimport

☐ Conflicts  ☐ Filter

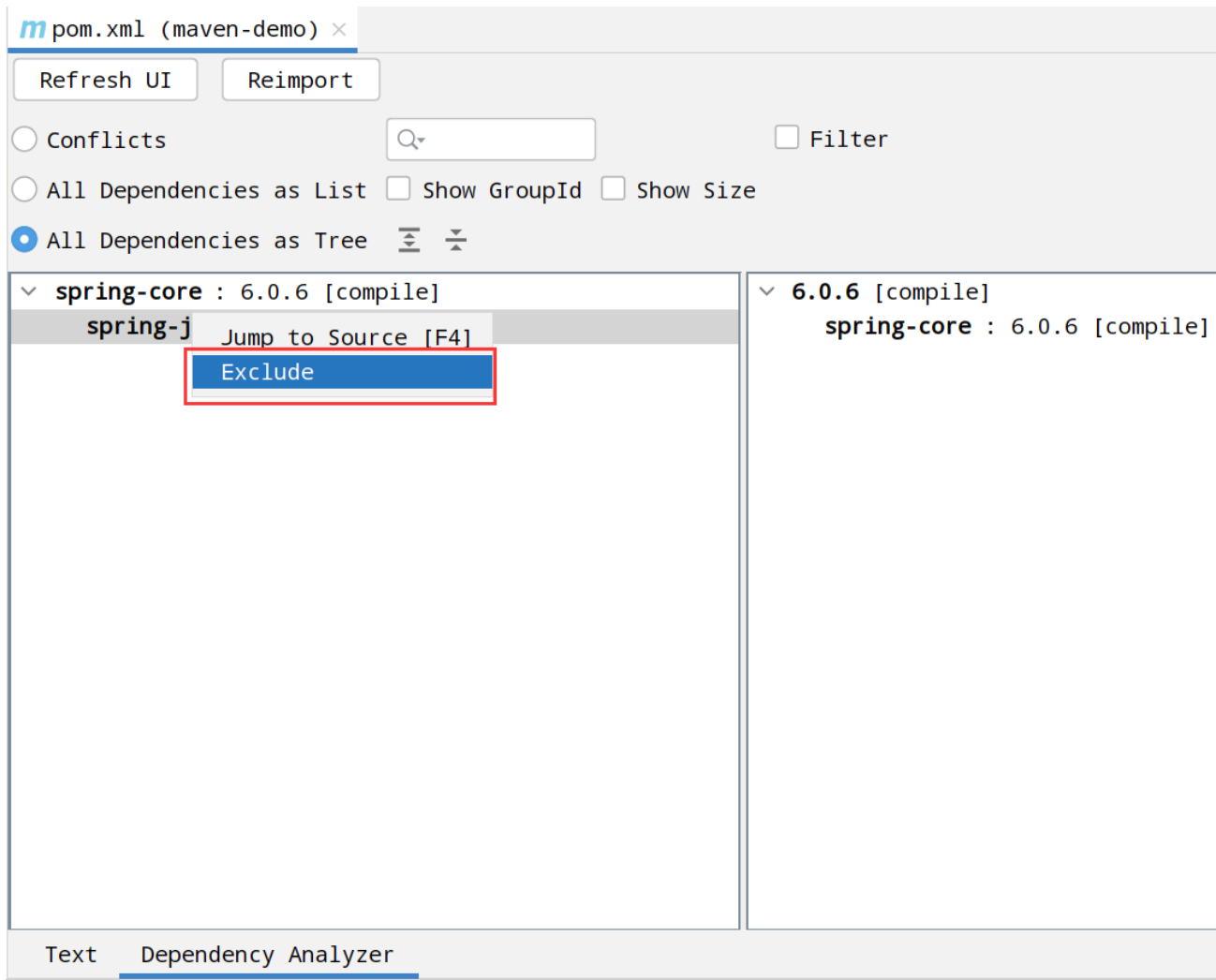
☐ All Dependencies as List ☐ Show GroupId ☐ Show Size

☒ All Dependencies as Tree ☐ ☐

spring-core : 6.0.6 [compile]  
spring-jcl : 6.0.6 [compile]

Text Dependency Analyzer

也可以右键排除掉一些依赖



## 2.5 Maven 仓库

我们通过短短几行代码, 就把依赖jar包放在了项目里, 具体是如何做的呢?

```
1 <dependency>
2   <groupId>mysql</groupId>
3   <artifactId>mysql-connector-java</artifactId>
4   <version>8.0.30</version>
5 </dependency>
```

这个代码, 我们称之为 "坐标", 也就是唯一的.

在Maven中, 根据 groupId、artifactId、version 的配置, 来唯一识别一个 jar 包, 缺一不可.

当我们在pom文件中配置完依赖之后, 点击刷新, Maven会根据坐标的配置, 去仓库里寻找Jar包, 并把他下载下来, 添加到项目中. 这个Jar包下载的地方就称为仓库.

仓库: 用于存储资源, 管理各种jar包



Maven仓库的本质就是一个目录(文件夹)，这个目录被用来存储开发中所有依赖(jar包, 插件等).



# springframework

---

<a href="#">../</a>	-	-
<a href="#">spring/</a>	-	-
<a href="#">spring-aop/</a>	-	-
<a href="#">spring-beans/</a>	-	-
<a href="#">spring-context/</a>	-	-
<a href="#">spring-core/</a>	-	-
<a href="#">spring-dao/</a>	-	-
<a href="#">spring-full/</a>	-	-
<a href="#">spring-hibernate/</a>	-	-
<a href="#">spring-jdbc/</a>	-	-
<a href="#">spring-mock/</a>	-	-
<a href="#">spring-orm/</a>	-	-
<a href="#">spring-parent/</a>	-	-
<a href="#">spring-remoting/</a>	-	-
<a href="#">spring-support/</a>	-	-
<a href="#">spring-web/</a>	-	-
<a href="#">spring-webmvc/</a>	-	-

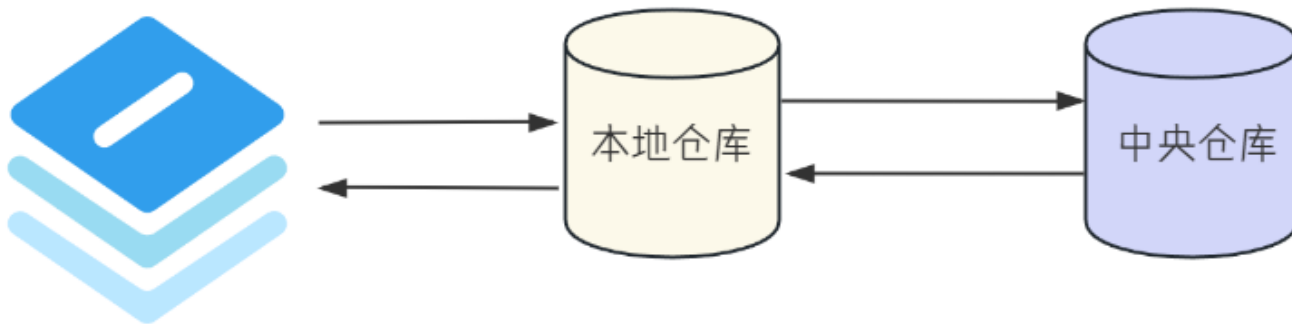
Maven仓库分为两大类: 本地仓库和远程仓库. 其中远程仓库又分为中央仓库, 私服 和其他公共库

## 2.5.1 本地仓库

本地仓库: 自己计算机上的一个目录(用来存储jar包)

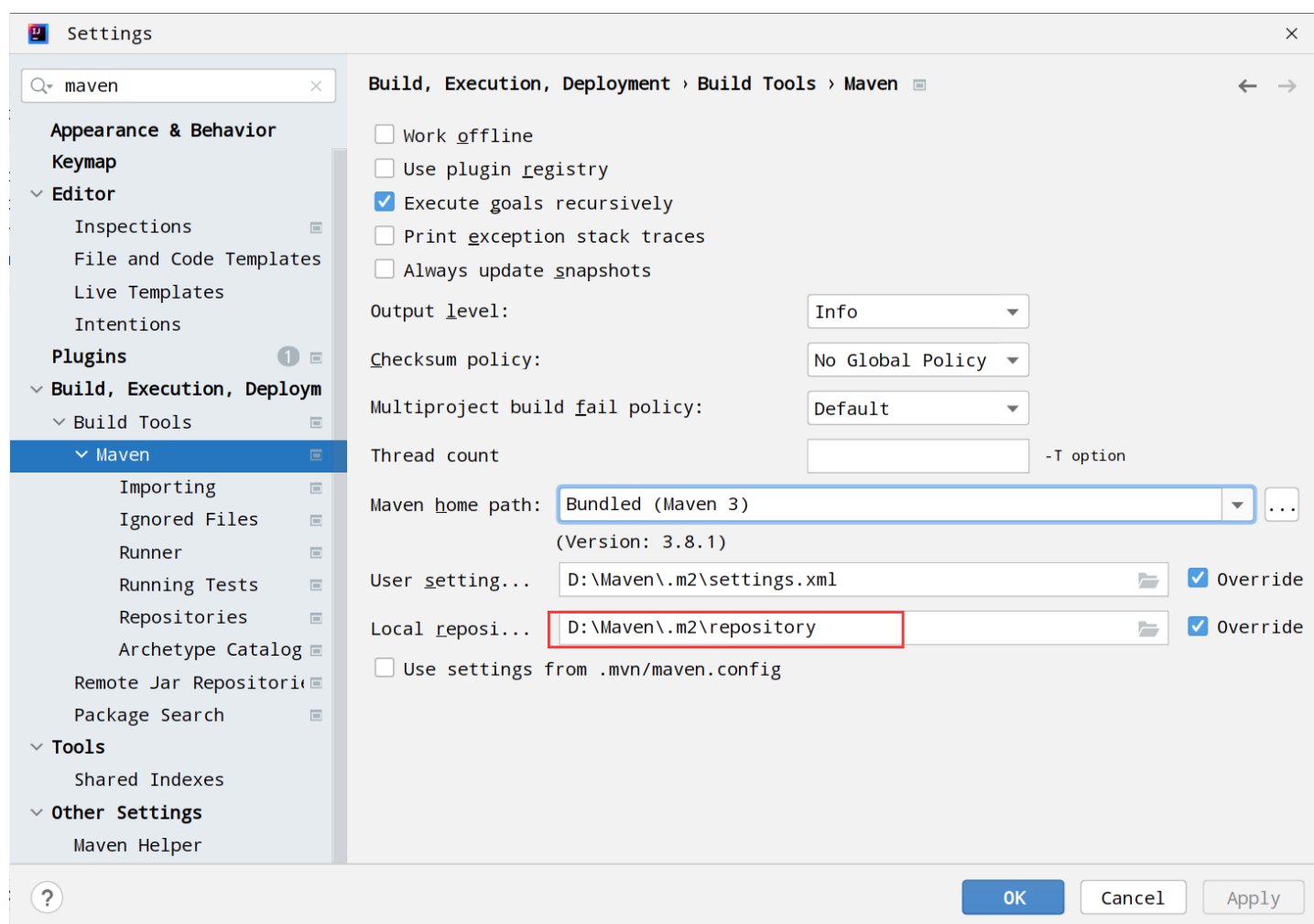
当项目中引入对应依赖jar包后, 首先会查找本地仓库中是否有对应的jar包

- 如果有, 则在项目直接引用
- 如果没有, 则去中央仓库中下载对应的jar包到本地仓库



本地仓库地址可以通过Maven配置查看:

File -> Settings



查看该仓库目录, 可以看到该目录下有很多的jar(最开始是空的, 随着Maven的使用, 该仓库下文件会越来越多)



<div> <div> <div>←</div> <div>→</div> <div>⌵</div> <div>⬆</div> </div> <div> <div>📁</div> <div>&gt; 此电脑 &gt; Data (D:) &gt; Maven &gt; .m2 &gt; repository &gt;</div> <div>⌵</div> </div> </div>				
快速访问	名称	修改日期	类型	大小
OneDrive	antlr	2023/3/24 19:34	文件夹	
此电脑	aopalliance	2023/3/24 19:34	文件夹	
3D 对象	asm	2023/3/24 19:34	文件夹	
视频	avalon-framework	2023/3/24 19:34	文件夹	
图片	backport-util-concurrent	2023/3/24 19:34	文件夹	
文档	ch	2023/3/24 19:34	文件夹	
下载	classworlds	2023/3/24 19:34	文件夹	
音乐	cn	2023/5/25 15:32	文件夹	
桌面	com	2023/7/24 22:49	文件夹	
Windows (C:)	commons-beanutils	2023/3/24 19:34	文件夹	
Data (D:)	commons-chain	2023/3/24 19:34	文件夹	
网络	commons-cli	2023/3/24 19:34	文件夹	
	commons-codec	2023/3/24 19:34	文件夹	
	commons-collections	2023/3/24 19:34	文件夹	
	commons-digester	2023/3/24 19:34	文件夹	
	commons-io	2023/3/24 19:34	文件夹	
	commons-lang	2023/3/24 19:34	文件夹	
	commons-logging	2023/3/24 19:34	文件夹	
	commons-validator	2023/3/24 19:34	文件夹	
	dom4j	2023/3/24 19:34	文件夹	
	error	2023/6/13 15:07	文件夹	
	io	2023/6/26 17:13	文件夹	
	jakarta	2023/8/30 15:54	文件夹	
	javax	2023/3/24 19:34	文件夹	
	junit	2023/3/24 19:34	文件夹	

## 2.5.2 中央仓库

中央仓库: maven 软件中内置一个远程仓库地址, 就是中央仓库, 服务于整个互联网. 由 Maven 团队维护, 全球唯一.

仓库地址: <https://repo1.maven.org/maven2/>

可以通过<https://mvnrepository.com> 这个网站来查询并下载

我们可以把自己写好的Jar包上传到中央仓库(具备一定的要求), 也可以从中央仓库下载Jar包

### 查找Jar的坐标

1. 访问 <https://mvnrepository.com/>
2. 进行查找, 比如mysql

← → ↺ ⌂

https://mvnrepository.com/search?q=mysql

MVN REPOSITORY

mysql

输入查找Jar包关键词

Search

Repository

Central989

Sonatype197

Spring Plugins95

Spring Lib M85

JCenter46

Clojars33

OpenHAB24

XWiki Releases20

Group

com.github107

io.github85

org.apache61

online-repo.223

org.anyline18

org.xwiki15

Found 1239 results

Sort: **relevance** | popular | newest

MySQL

1. **MySQL Connector/J**

com.mysql » mysql-connector-j

JDBC Type 4 driver for MySQL.

Last Release on Jul 18, 2023

317 usages

从列表中选择自己要添加的依赖jar  
点击进去

MySQL

2. **MySQL Connector Java**

mysql » mysql-connector-java

MySQL Connector/J is a JDBC Type 4 driver, which means that it is pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries. This driver supports auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for large update counts, support for local and offset date-time variants from the java.time package, support for JDBC-4.x XML processing, support for per connection client information and support for the NCHAR, NVARCHAR ...

Last Release on Apr 18, 2023

7,263 usages

3. **Testcontainers :: JDBC :: MySQL**

org.testcontainers » mysql

274 usages

MIT

3. 选择要添加的Jar包版本

Home » com.mysql » mysql-connector-j

MySQL

**MySQL Connector/J**

JDBC Type 4 driver for MySQL.

Categories

JDBC Drivers

Tags

database | sql | jdbc | driver | connector | mysql

Ranking

#1465 in MvnRepository (See Top Artifacts)  
#12 in JDBC Drivers

Used By

317 artifacts

Central (4)

	Version	Vulnerabilities	Repository	Usages	Date
8.1.x	8.1.0		Central	69	Jul 18, 2023
	8.0.33		Central	165	Apr 18, 2023
8.0.x	8.0.32		Central	129	Jan 18, 2023
	8.0.31		Central	101	Oct 14, 2022

4. 查看Jar包对应坐标

Categories	JDBC Drivers
Tags	database sql jdbc driver connector mysql
Organization	Oracle Corporation
HomePage	<a href="http://dev.mysql.com/doc/connector-j/en/">http://dev.mysql.com/doc/connector-j/en/</a>
Date	Apr 18, 2023
Files	<a href="#">pom (3 KB)</a> <a href="#">jar (2.4 MB)</a> <a href="#">View All</a>
Repositories	Central
Ranking	#1465 in MvnRepository (See Top Artifacts) #12 in JDBC Drivers
Used By	317 artifacts

**Note:** There is a new version for this artifact

New Version	8.1.0
-------------	-------

[Maven](#)
[Gradle](#)
[Gradle \(Short\)](#)
[Gradle \(Kotlin\)](#)
[SBT](#)
[Ivy](#)
[Grape](#)
[Leiningen](#)
[Buildr](#)

```

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>8.0.33</version>
</dependency>

```

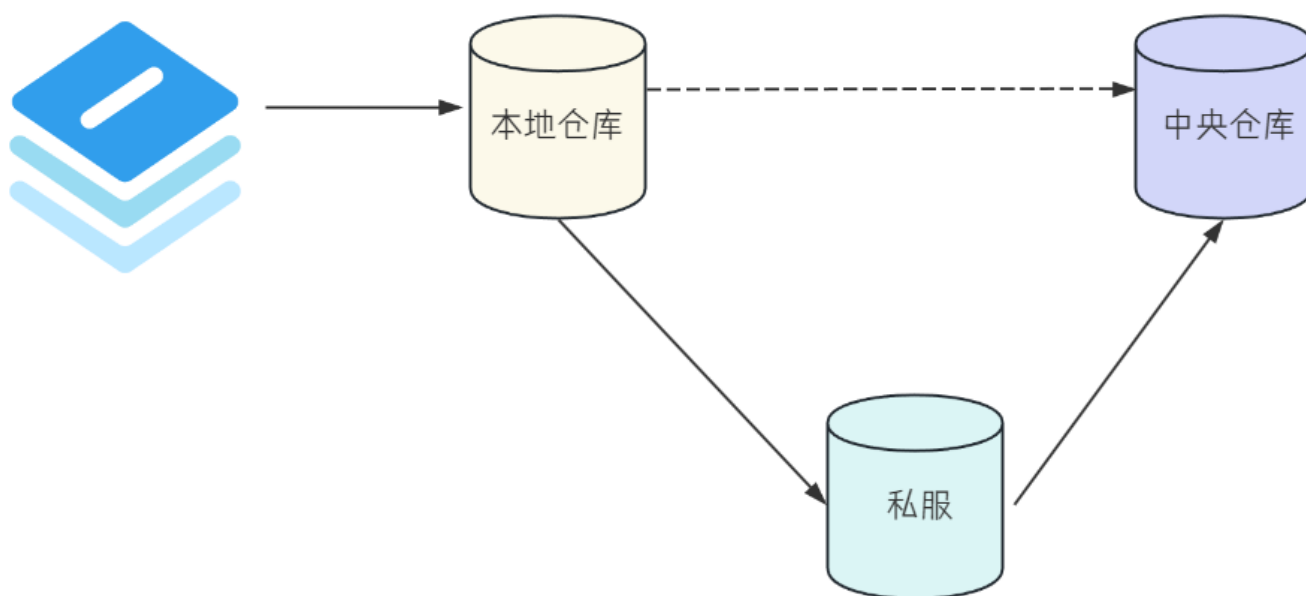
Jar包对应的坐标

### 2.5.3 私有服务器, 也称为私服

私服: 一般由公司团队搭建的私有仓库.

私服属于某个公司, 或者某个部门, 往往需要一定权限.

有了私服之后, Maven依赖下载的顺序又发生了变化



当Maven需要下载资源的时候

1. 先从本地仓库获取, 本地仓库存在, 则直接返回
2. 如果本地仓库没有, 就从私服请求, 私服存在该资源, 就直接返回
3. 如果私服上不存在该资源, 则从中央仓库下载, 中央仓库不存在, 就报错了...
4. 如果中央仓库中存在, 就先缓存在私服上之后, 再缓存到本地仓库里, 再为Maven的下载请求提供服务

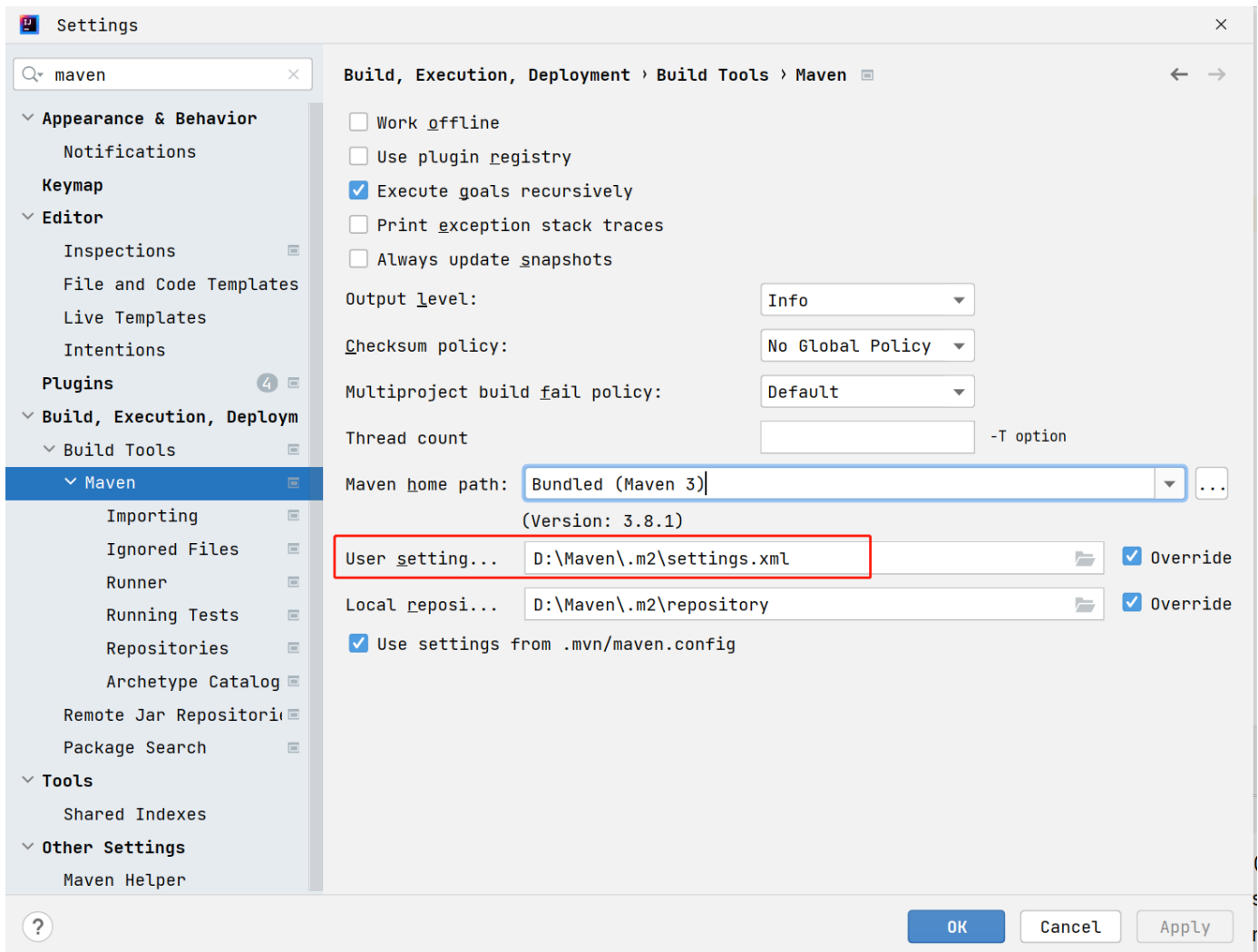
私服是很多人在使用的, 所以只需要第一个使用者下载一次就可以了

## 2.6 Maven 设置国内源

因为中央仓库在国外, 所以下载起来会比较慢, 所以咱们选择借助国内一些公开的远程仓库来下载资源  
接下来介绍, 如何设置国内源

### 2.6.1 配置当前项目setting

File -> Settings



1. 查看配置文件的地址, 如上图所示, Maven配置文件地址为: `D:\Maven\.m2\settings.xml`

不同电脑设置的Maven路径不同

settings和repository 可以修改设置为其他路径, 两个路径不要有中文

## 2. 配置国内源

Maven 仓库默认在国外<https://mvnrepository.com/>，国内使用时会比较慢，我们可以更换为阿里云的仓库

也可以选择别的仓库, 参考: [Maven 镜像地址大全](#)

打开settings.xml, 在 mirrors 节点上, 添加内容如下:

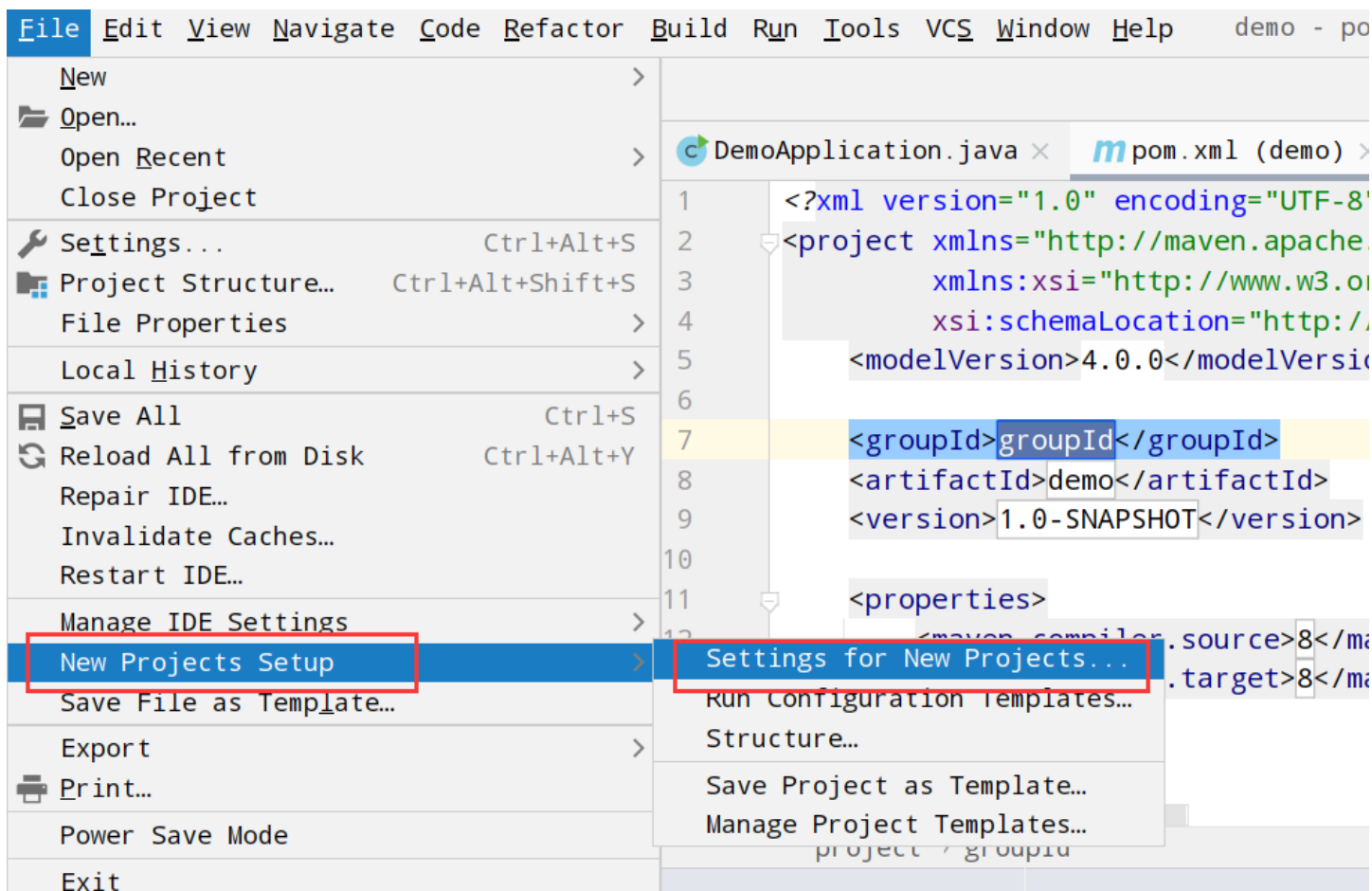
```
1  <mirror>
2      <id>aliyunmaven</id>
3      <mirrorOf>central</mirrorOf>
4      <name>阿里云公共仓库</name>
5      <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
6  </mirror>
```

如果上述地址不存在settings文件, 则直接复制课件中提供的文件粘贴过去即可.

### 2.6.2 设置新项目的setting

上述配置的内容, 只对当前项目生效, 为了让后续新建的项目也生效, 需要重新设置一下新项目的 Settings

当前项目和新项目共用一个settings文件即可, 所以新项目的设置, 只需要确认一下settings文件的路径即可.

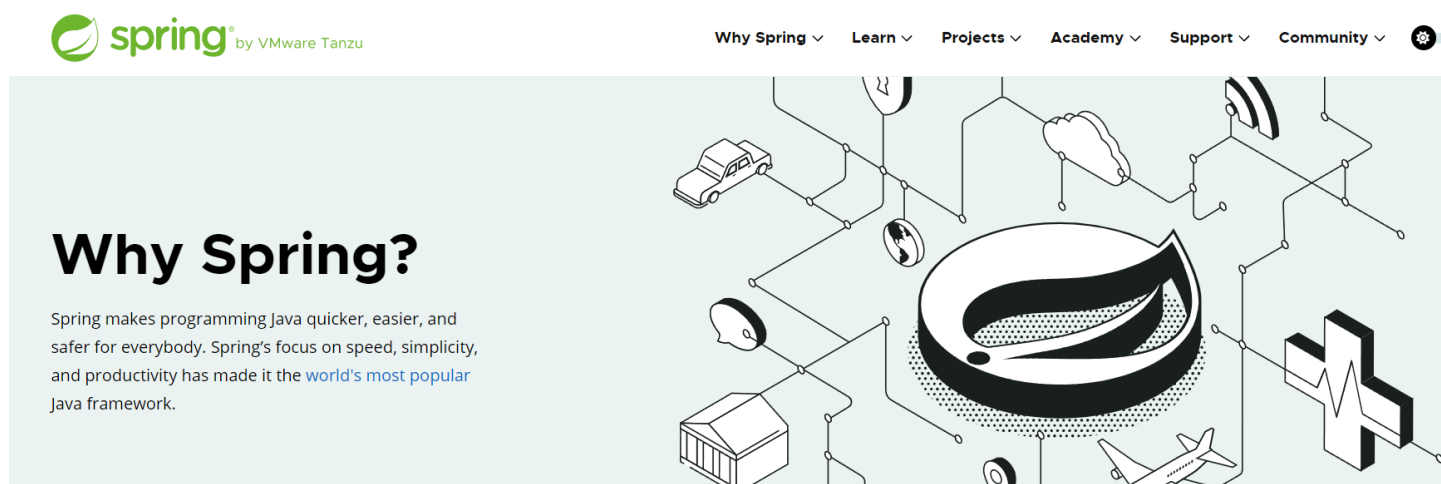


## 3. 第一个SpringBoot程序

### 3.1 Spring Boot介绍

在学习SpringBoot之前, 我们先来认识一下Spring

我们看下Spring官方(<https://spring.io/>)的介绍

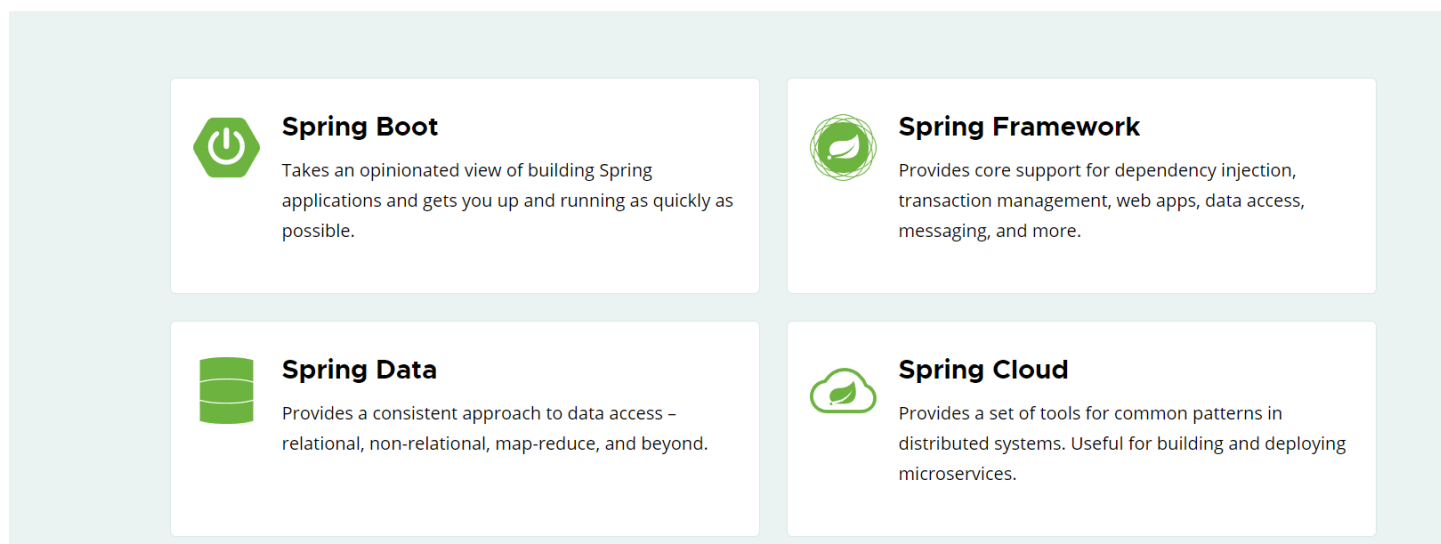


可以看到, Spring让Java程序更加快速, 简单和安全. Spring对于速度、简单性和生产力的关注使其成为世界上最流行的Java框架。

Spring官方提供了很多开源的项目, 覆盖范围从Web开发到大数据, Spring发展到了今天, 已经形成了自己的生态圈. 我们在开发时, 也倾向于使用Spring官方提供的技术, 来解决对应的问题.

## Projects

From configuration to security, web apps to big data—whatever the infrastructure needs of your application may be, there is a Spring Project to help you build it. Start small and use just what you need—Spring is modular by design.



这些项目都是基于Spring Framework来进行开发的, 但是Spring Framework存在配置多, 入门难的问题, Spring 也意识到了这个问题, 为了简化开发者的使用, 从而创造性的推出了SpringBoot.



接下来我们看下什么是Spring Boot.

**Spring Boot 的诞生是为了简化 Spring 程序开发的。**

Spring Boot 翻译一下就是 Spring 脚手架，什么是脚手架呢？如下图所示：



盖房子的这个架子就是脚手架，脚手架的作用是砌筑砖墙，浇筑混凝土、方便墙面抹灰，装饰和粉刷的，简单来说，就是使用脚手架可以更快速的盖房子。

而 Spring Boot 就是 Spring 框架的脚手架，它是为了快速开发 Spring 框架而诞生的。

以前铺路是这样的：



改造之后的效率是这样的：



或者是这样的：





可以看到，每次技术的诞生和改进相比于之前的效率会有一个质的提升，而 Spring Boot 相比于 Spring 也是如此。

## 3.2 Spring Boot 项目创建

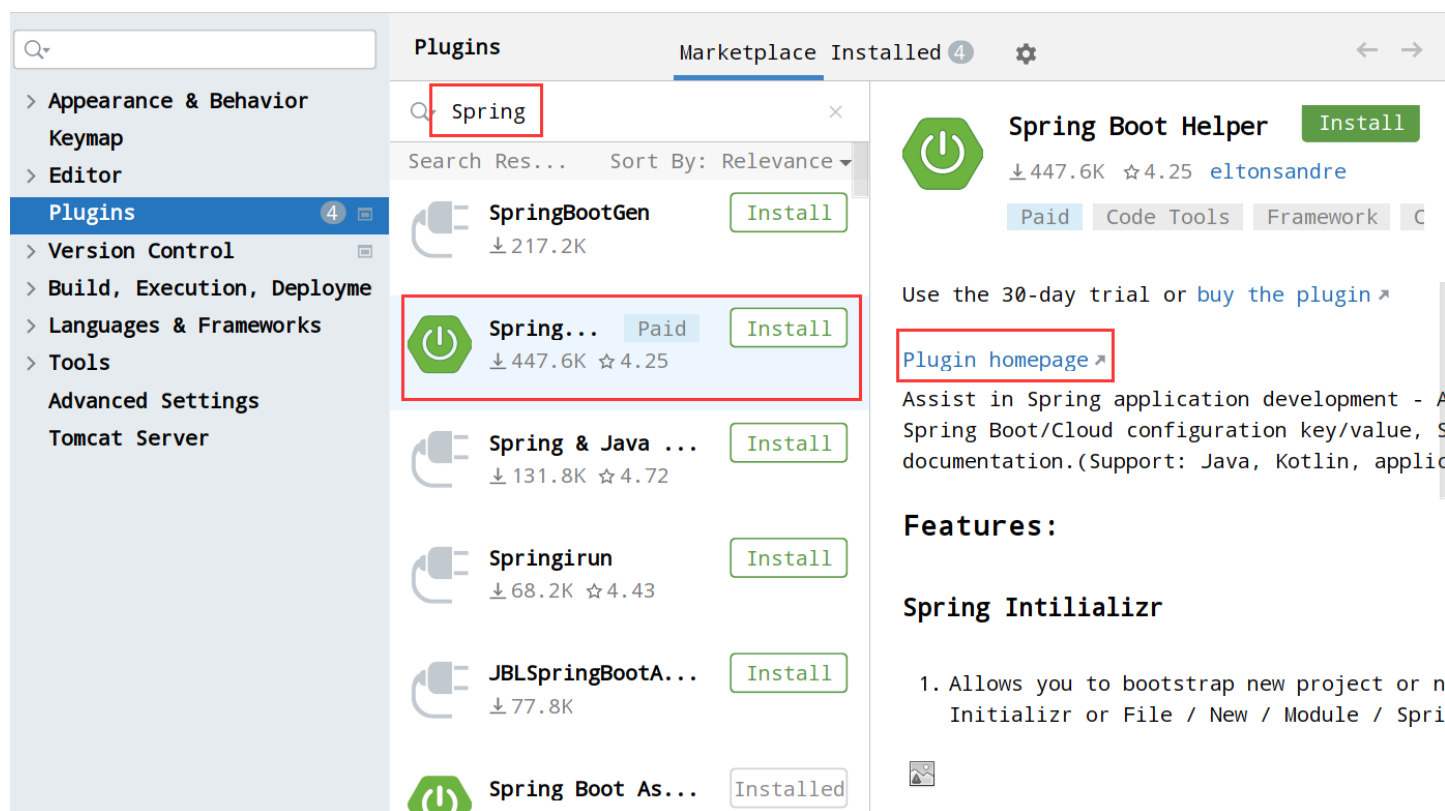
### 3.2.1 使用 Idea 创建

因为我们用的 Idea 社区版（其他版本也同样适用），所以先要安装 Spring Boot Helper 插件才能创建 Spring Boot 项目。

#### 安装 Spring Boot Help 插件

使用专业版 Idea 的同学不需要安装插件, Idea 已经集成了

#### 1. 查找插件并下载



点击 [Plugin homepage](#) ,进入网页下载插件

此处不要直接点击 Install

直接点击Install, 安装的是收费版(土豪请随意)

插件地址: <https://plugins.jetbrains.com/plugin/18622-spring-boot-helper/versions>

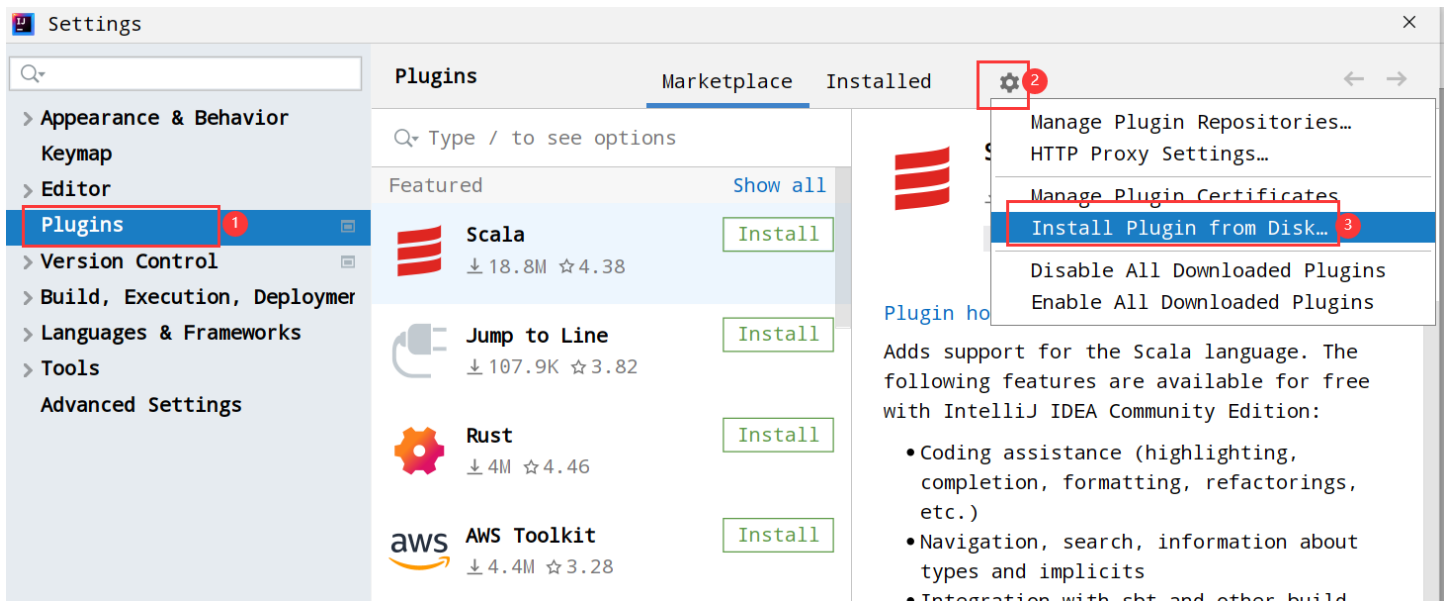
Free versions	对应Idea版本		点击下载
<a href="#">2022.1.2</a>	<a href="#">2021.1 — 2022.1.4</a>	Jul 15, 2022	<a href="#">Download</a>
<a href="#">2022.1.1</a>	2021.1 — 2022.1.4	Jun 30, 2022	<a href="#">Download</a>
<a href="#">2022.1.0</a>	2021.1 — 2022.1.4	Jun 28, 2022	<a href="#">Download</a>

插件对Idea版本有要求, Idea版本需要在2021.1 -2022.1.4范围内, 不在这个范围内需要重新卸载安装

卸载要删除注册表的内容

## 2. 安装插件

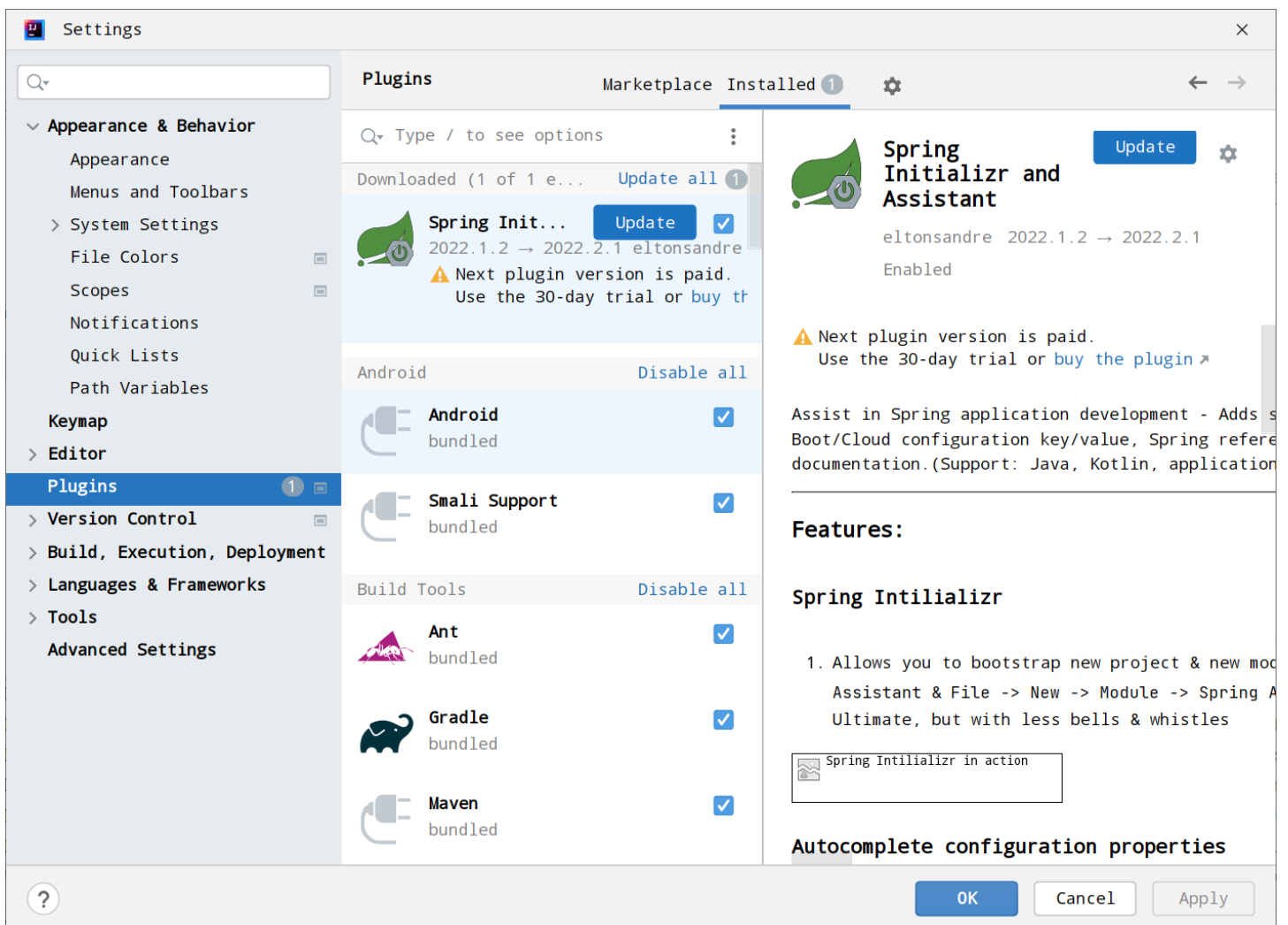
按下图序号操作, 逐步安装插件



选择刚才下载的插件, 安装, 重启Idea即可

此时查看已安装插件

安装好之后, 它的名字就变成了 Spring Initializr and Assistant, 如下图所示：

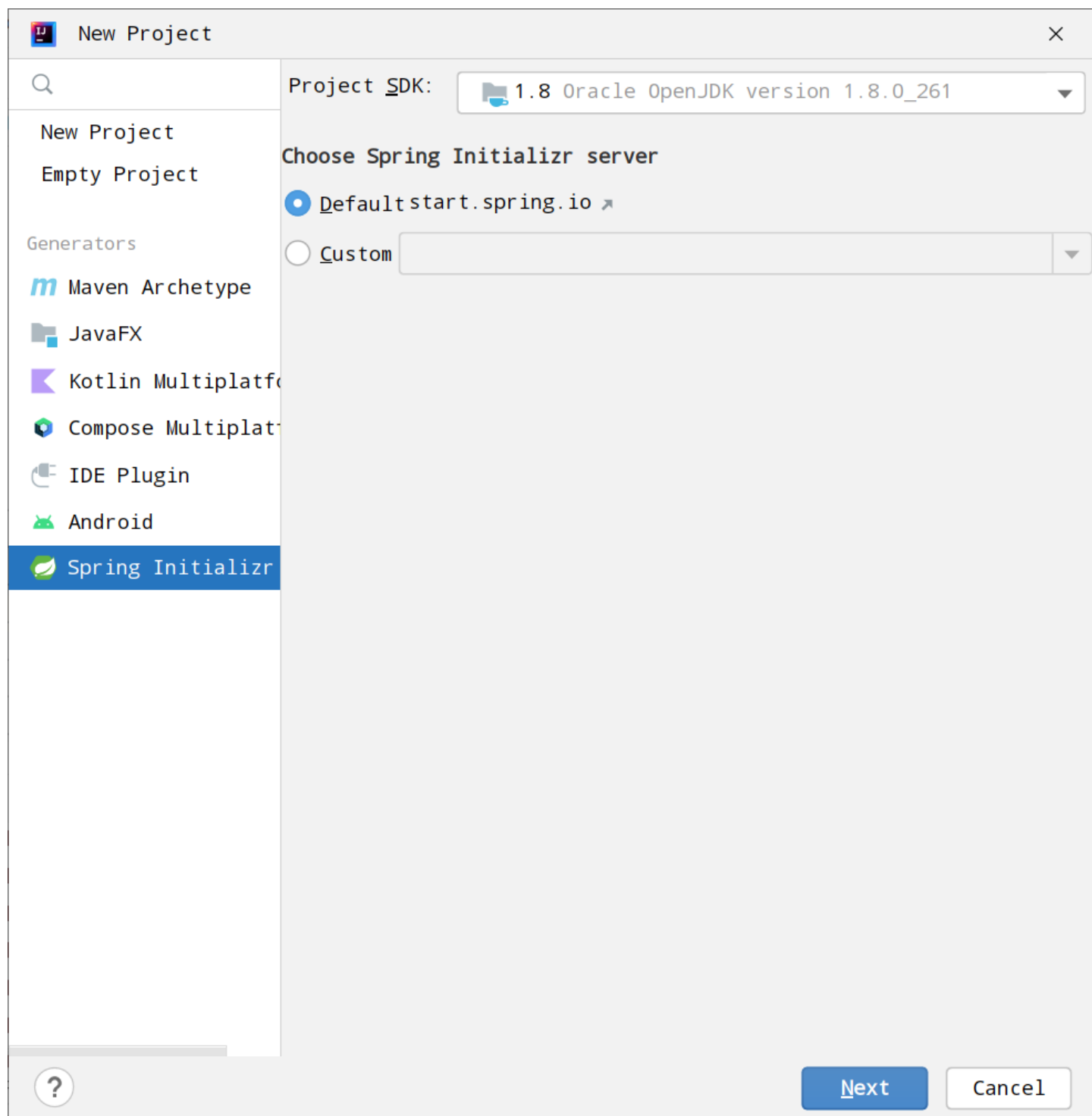



不要点击 update!!!

创建SpringBoot项目

接下来我们来创建 Spring Boot 项目：

File -> New ->Project



 New Project

×

Project properties

Group Id

com.example

组织id

Artifact Id

demo

项目id

Version

0.0.1-SNAPSHOT

项目版本

Project type

Maven

maven项目

Language

Java

开发语言

Packaging

Jar

打包类型

Java version

8

Java版本(推荐使用Java8)

Project name

demo

项目名称

Project description

Demo project for Spring Boot

项目描述

Package name

com.example.demo

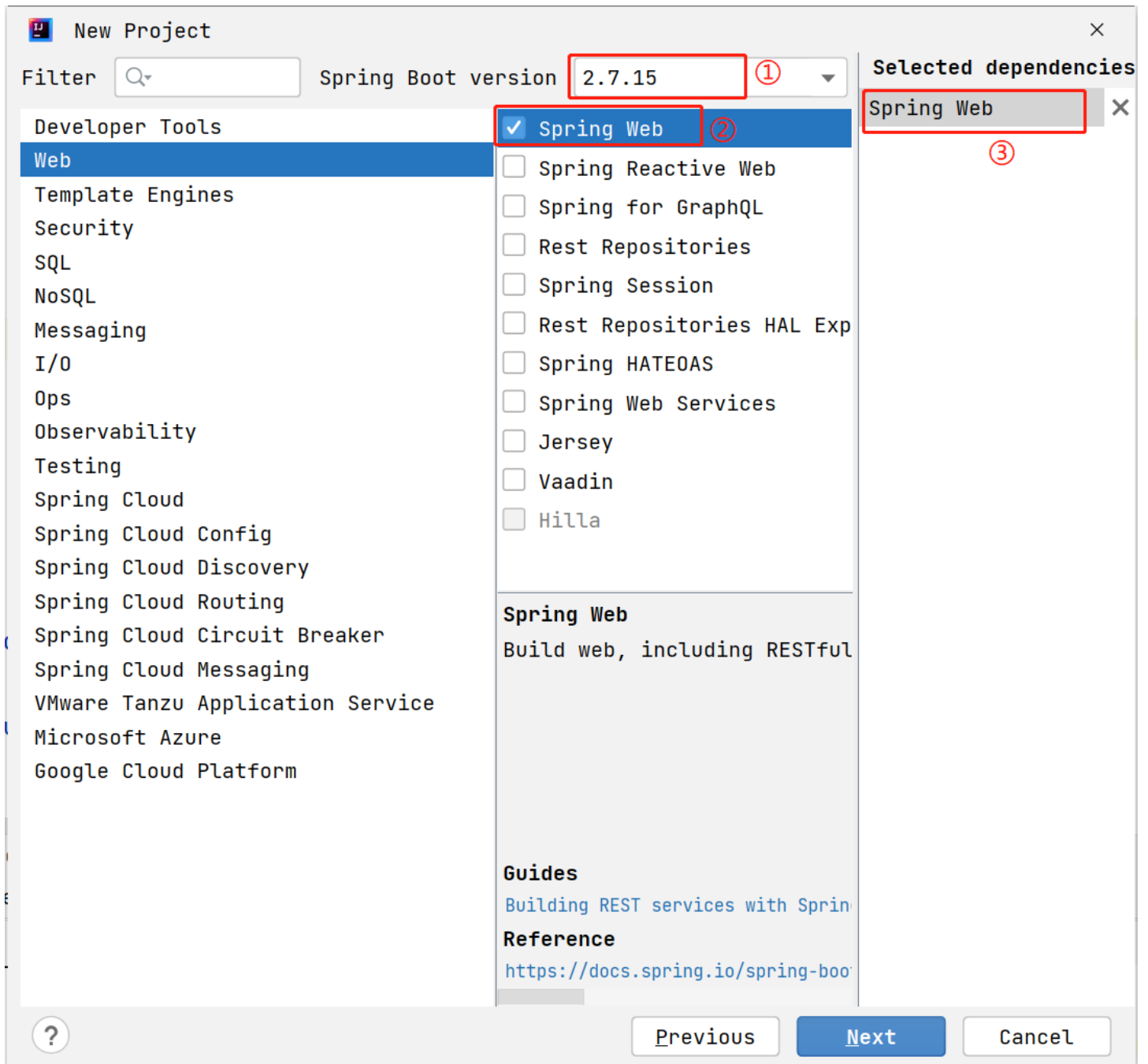
项目包名

?

Previous

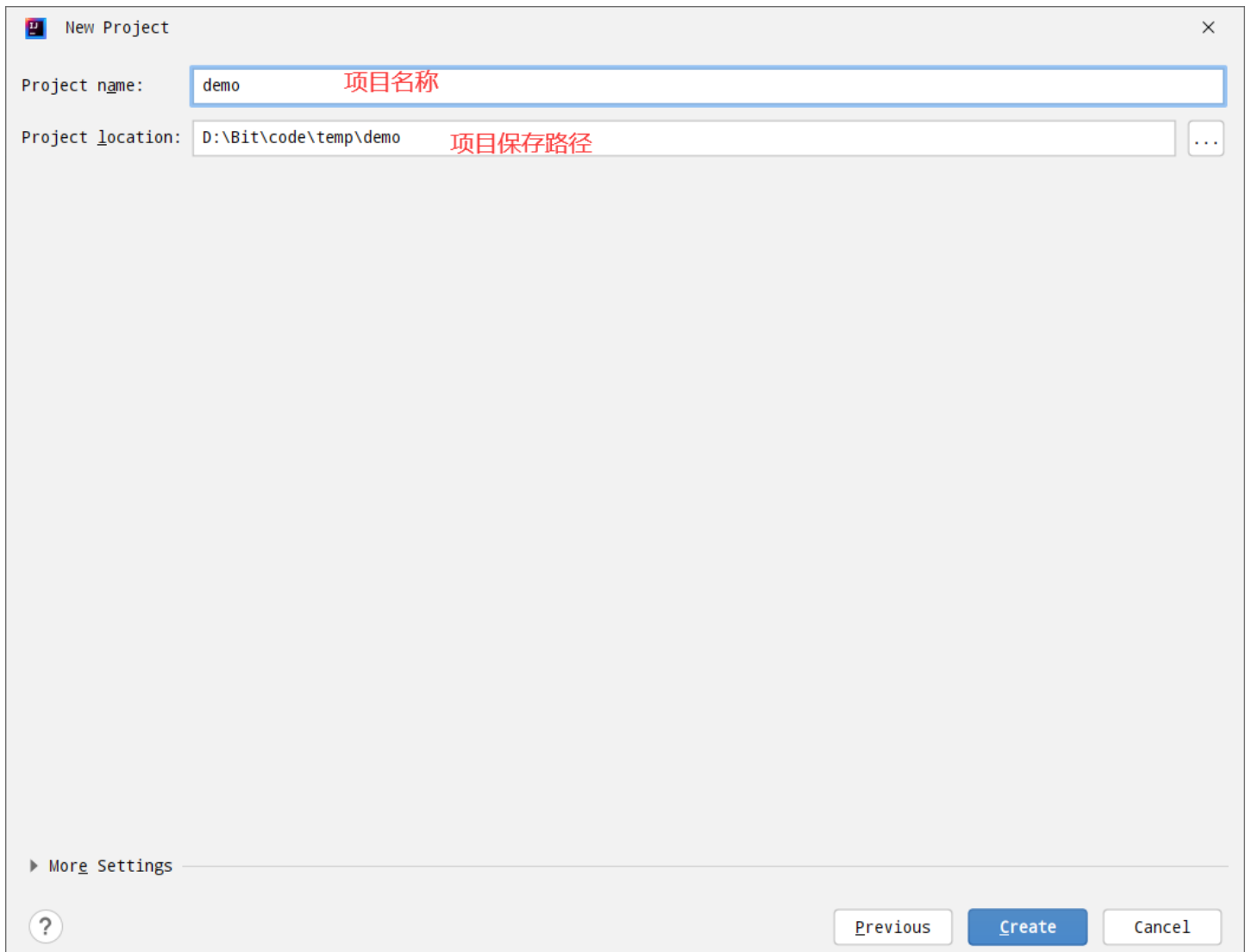
Next

Cancel



### 上图说明

- ① SpringBoot版本 , 选择任意2.X的, 这个版本会随着SpringBoot的升级而发生变化, 不固定  
3.X版本使用的是jdk17
- ② 打勾表示创建项目之初, 就引入的第三方依赖(框架, 插件, 组件等...后面再细讲)
- ③ 所有引用的第三方框架



点击 Create 就完成 Spring Boot 的项目创建了。

### 注意事项

第一次打开 Spring Boot 项目需要加载很久，因为当前 Spring Boot 框架并没有在自己的本地仓库。为了加速 Spring Boot 项目的下载，在打开项目之前，请先确认自己的 Maven 已经配置为国内源

### 3.2.2 网页版创建（了解）

不使用 Idea 也可以创建 Spring Boot 项目，我们可以使用 Spring 官方提供的网页版来创建 Spring Boot 项目。

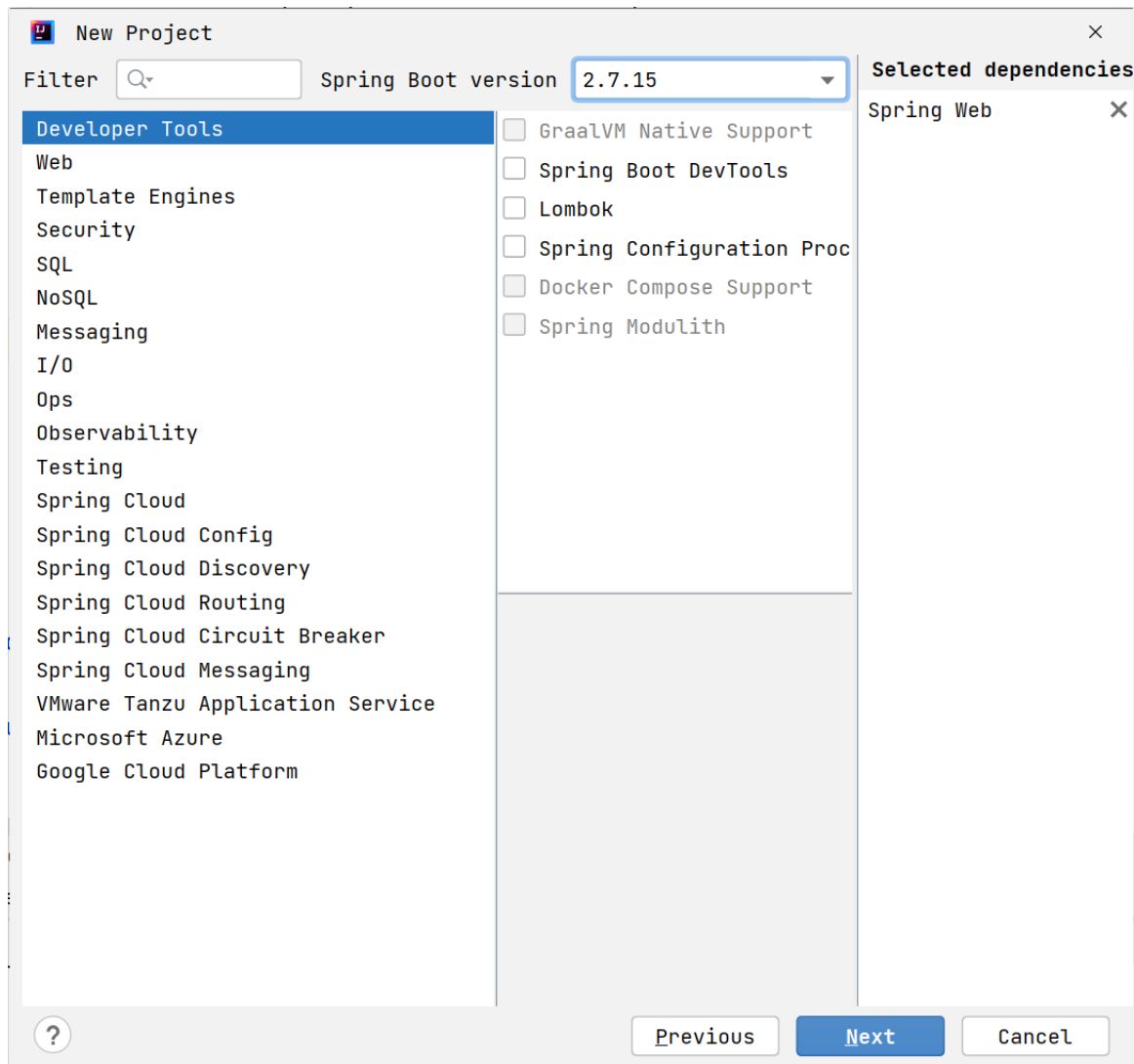
网页版创建项目先访问：<https://start.spring.io>，如下图所示：

# Project Metadata

Group	com.example	项目组织标识
Artifact	demo	项目标识
Name	demo	项目名称
Description	Demo project for Spring Boot	项目简介
Package name	com.example.demo	项目包名
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War	打包类型
Java	<input type="radio"/> 17 <input type="radio"/> 11 <input checked="" type="radio"/> 8	使用 Java 版本

Idea创建项目时, 显示的界面, 就是来自于这个网站, 所以网络不好的情况下, 这个界面也会打不开





点击生成按钮会下载一个 Spring Boot 的 zip 包，解压 zip 之后目录如下：

Windows (C:) > 用户 > lucf > 下载 > demo > demo >

名称	修改日期	类型	大小
.mvn	2023/4/10 18:13	文件夹	
src	2023/4/10 18:13	文件夹	
.gitignore	2023/4/10 18:13	文本文档	1 KB
HELP.md	2023/4/10 18:13	Markdown docu...	1 KB
mvnw	2023/4/10 18:13	文件	11 KB
mvnw.cmd	2023/4/10 18:13	Windows 命令脚本	7 KB
pom.xml	2023/4/10 18:13	XML 文档	2 KB

然后再使用 Idea 打开之后，Spring Boot 项目就算创建成功了。

### 3.3 项目代码和目录介绍

观察pom文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.or
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.7.15</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>demo</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17    <java.version>1.8</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-web</artifactId>
23    </dependency>
24
25    <dependency>
26      <groupId>org.springframework.boot</groupId>
27      <artifactId>spring-boot-starter-test</artifactId>
28      <scope>test</scope>
29    </dependency>
30  </dependencies>
31
32  <build>
33    <plugins>
34      <plugin>
35        <groupId>org.springframework.boot</groupId>
36        <artifactId>spring-boot-maven-plugin</artifactId>
37      </plugin>
38    </plugins>
39  </build>
40
41 </project>
```

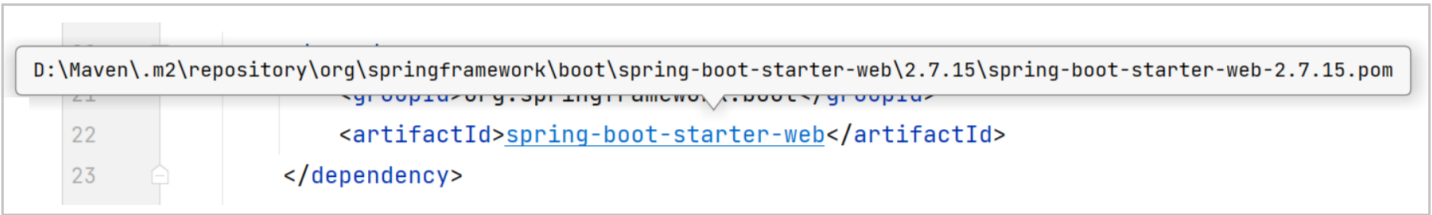
pom.xml文件里的具体信息, 不需要过度关注, 我们主要看 `<dependencies>` 标签里面的内容

pom文件中, 引入了两个依赖

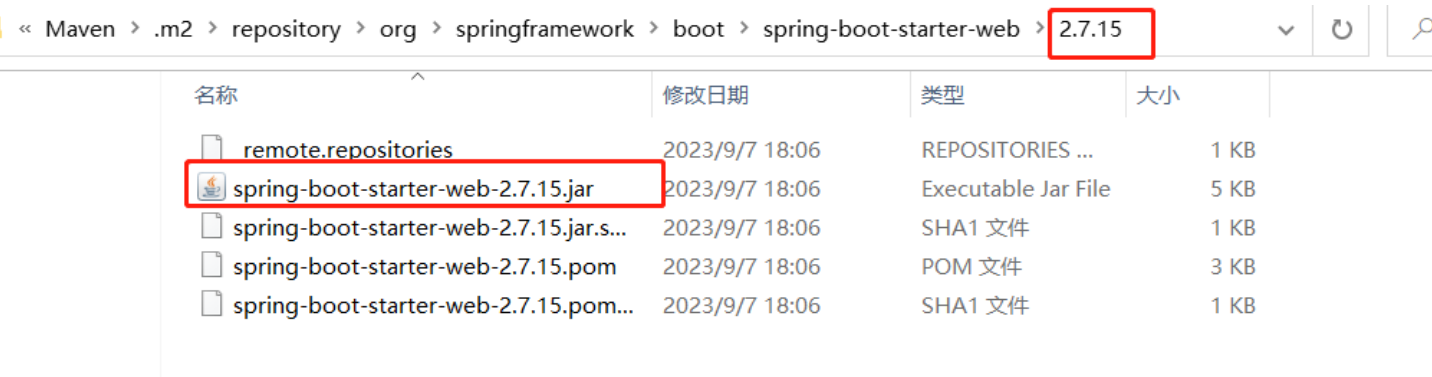
- spring-boot-starter-web: 包含了web应用开发所需要的常见依赖
- spring-boot-starter-test: 包含了单元测试所需要的常见依赖

父工程

但是我们发现, 这个依赖没有指定具体的版本号, 但我们的jar包也正常引入到项目中了.并且查看的话, 是有版本号的.



我们来看看本地仓库的的jar包是否下载下来了呢?



也下载下来了.

为什么没有指定 <version> , 程序也可以正常运行呢?

因为每一个SpringBoot工程, 都有一个父工程. 依赖的版本号, 在父工程中统一管理.

父工程指定版本号后, 就会自动引入和所指定版本对应的依赖, 创建的SpringBoot项目, 会继承SpringBoot父工程.

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.15</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo</name>
  <description>Demo project for Spring Boot</description>

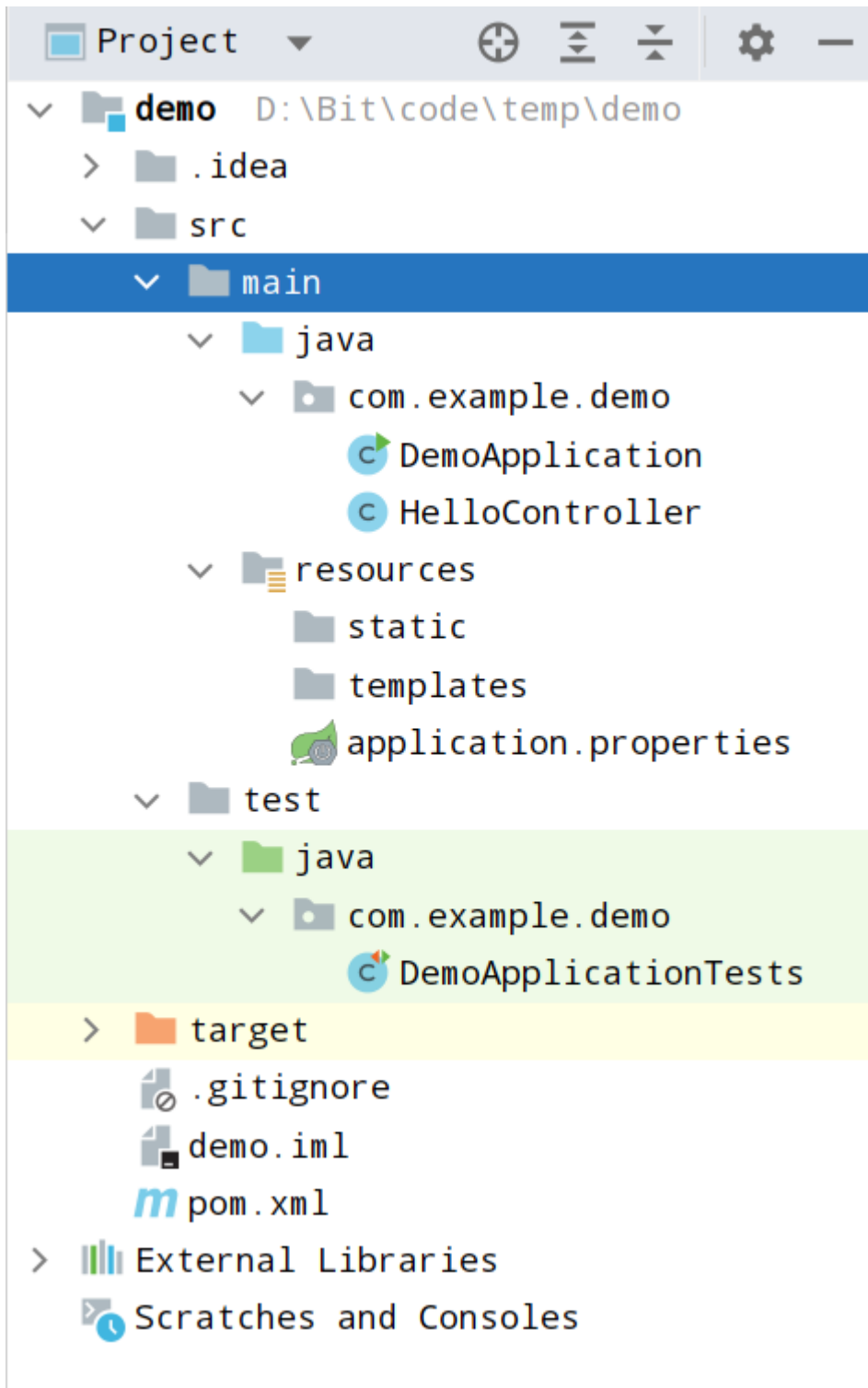
```

```

pom.xml (demo) x spring-boot-starter-parent-2.7.15.pom x spring-boot-dependencies-2.7.15.pom x
26 <url>https://github.com/spring-projects/spring-boot</url> 1034 ✓
27 </scm>
28 <properties>
29   <activemq.version>5.16.6</activemq.version>
30   <antlr2.version>2.7.7</antlr2.version>
31   <appengine-sdk.version>1.9.98</appengine-sdk.version>
32   <artemis.version>2.19.1</artemis.version>
33   <aspectj.version>1.9.7</aspectj.version>
34   <assertj.version>3.22.0</assertj.version>
35   <atomikos.version>4.0.6</atomikos.version>
36   <awaitility.version>4.2.0</awaitility.version>
37   <build-helper-maven-plugin.version>3.3.0</build-helper-maven-plugin.version>
38   <byte-buddy.version>1.12.23</byte-buddy.version>
39   <cache2k.version>2.6.1.Final</cache2k.version>
40   <caffeine.version>2.9.3</caffeine.version>
41   <cassandra-driver.version>4.14.1</cassandra-driver.version>
42   <classmate.version>1.5.1</classmate.version>
43   <commons-codec.version>1.15</commons-codec.version>
44   <commons-dbcp2.version>2.9.0</commons-dbcp2.version>
45   <commons-lang3.version>3.12.0</commons-lang3.version>
46   <commons-pool.version>1.6</commons-pool.version>
47   <commons-pool2.version>2.11.1</commons-pool2.version>

```

## 目录介绍



Spring Boot 项目有两个主要的目录：

- src/main/java: Java 源代码
- src/main/resources: 为静态资源或配置文件：
  - /static: 静态资源文件夹, 比如前期学的js, css, html等静态文件, 不需要服务器数据进行绑定的页面
  - /templates: 模版资源文件夹, 主要存放动态模板文件, 比如JSP, Freemarker, Thymeleaf等需要服务器动态渲染数据的文件
- src/test/java: 测试代码源代码

- target: 编译后的文件路径
- pom.xml: maven 配置文件

## 3.4 运行项目

```
1 usage
  @SpringBootApplication  SpringBoot 启动类注解
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

点击启动类的 main 方法就可以运行 Spring Boot 项目了，启动成功如下图所示：

```
2023-09-08 10:59:39.405 INFO 35908 --- [main] com.example.demo.DemoApplication : No active profile set, falling back to 1 default profile: "default"
2023-09-08 10:59:40.226 INFO 35908 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-09-08 10:59:40.234 INFO 35908 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-09-08 10:59:40.234 INFO 35908 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.79]
2023-09-08 10:59:40.407 INFO 35908 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-09-08 10:59:40.407 INFO 35908 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 954 ms
2023-09-08 10:59:40.655 INFO 35908 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-09-08 10:59:40.661 INFO 35908 --- [main] com.example.demo.DemoApplication : Started DemoApplication in 1.614 seconds (JVM running for 2.029s)
```

## 3.5 输出 Hello world

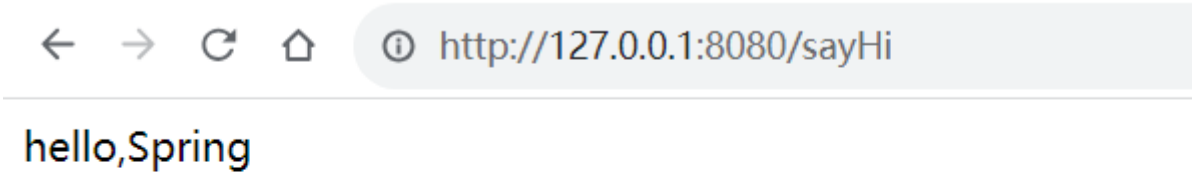
我们的Java EE课程, 更多是围绕着如何使用Java来进行web开发.

咱们前面的课程中, 讲的是Java基础, 是没办法直接和浏览器进行互动的. 所以接下来我们要用 Spring Boot 来实现和浏览器及用户的交互。

在创建的项目包路径下创建 UserController 文件，实现代码如下：

```
1 import org.springframework.stereotype.Controller;
2 import org.springframework.web.bind.annotation.RequestMapping;
3 import org.springframework.web.bind.annotation.RestController;
4
5 @RestController
6 public class UserController {
7
8     @RequestMapping("/sayHi")
9     public String sayHi(){
10         return "hello, Spring";
11     }
12 }
```

重新启动项目，访问 <http://127.0.0.1:8080/sayHi> 最终效果如下：



### 3.6 Web服务器

浏览器和服务器两端进行数据交互, 使用的就是HTTP协议



前面我们已经学习了 HTTP 协议, 知道了 HTTP 协议就是 HTTP 客户端和 HTTP 服务器之间的交互数据的格式.

Web 服务器就是对HTTP协议进行封装, 程序员不需要直接对协议进行操作(自己写代码去解析http协议规则), 让Web开发更加便捷, 所以Web服务器也被称为WWW服务器, HTTP服务器, 主要功能是提供网上信息浏览服务.

常见的Web服务器有: Apache, Nginx, IIS, **Tomcat**, Jboss等

SpringBoot 内置了Tomcat服务器, 无需配置即可直接运行

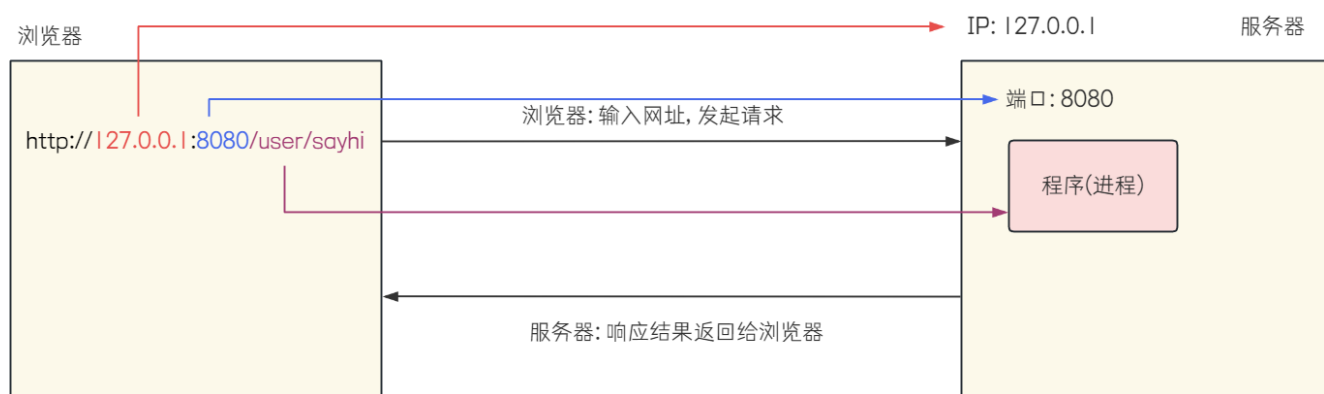
```
2023-08-06 18:14:10.918 INFO 1836 --- [main] com.example.demo.DemoApplication : Starting DemoApplication using Java 1.8.0_261 on LUCF with PID 1836
2023-08-06 18:14:10.922 INFO 1836 --- [main] com.example.demo.DemoApplication : No active profile set, falling back to 1 default profile: "default"
2023-08-06 18:14:11.464 INFO 1836 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules found, entering strict repository confi
2023-08-06 18:14:11.467 INFO 1836 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data Redis repositories in DEFAULT mode.
2023-08-06 18:14:11.493 INFO 1836 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 9 ms. Found 0 Redis repo
2023-08-06 18:14:12.167 INFO 1836 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-08-06 18:14:12.175 INFO 1836 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-08-06 18:14:12.175 INFO 1836 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.78]
2023-08-06 18:14:12.328 INFO 1836 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-08-06 18:14:12.328 INFO 1836 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1367 ms
```

Tocmat默认端口号是8080, 所以我们程序访问时的端口号也是8080

## 3.7 请求响应流程分析

浏览器输入URL之后, 发起请求, 就和服务器之间建立了连接

服务器:



浏览器:

- 输入网址: <http://127.0.0.1:8080/hello>
  - 通过IP地址127.0.0.1定位到网络上的一台计算机, 127.0.0.1就是本机
  - 通过端口号8080找到计算机上对应的进程, 也就是在本地计算机中找到正在运行的8080端口的程序
  - /user/sayhi是请求资源位置
    - 资源: 对计算机而言资源就是数据
      - web资源: 通过网络可以访问到的资源 (通常是指存放在服务器上的数据)

<http://127.0.0.1:8080/user/sayhi>, 就是向本地计算机中的8080端口程序, 获取资源位置是/user/sayhi的数据

8080端口程序, 在服务器找/hello位置的资源数据, 发给浏览器

服务器:

- 接收到浏览器发送的信息 (如: /user/sayhi)
- 在服务器上找到/user/sayhi的资源
- 把资源发送给浏览器

## 3.8 访问出错怎么办

### 3.8.1 404

404 表示用户访问的资源不存在. 大概率是 URL 的路径写的不正确.



## 错误实例1: url单词拼错

← → ↻ 🏠 ⓘ http://127.0.0.1:8080/sayHi1

# Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Aug 08 20:58:32 CST 2023

There was an unexpected error (type=Not Found, status=404).

## 错误实例2: 注解写错

```
1 @Controller
2 public class UserController {
3
4     @RequestMapping("/sayHi")
5     public String sayHi(){
6         return "hello, Spring";
7     }
8 }
```

访问时也会报错404

← → ↻ 🏠 ⓘ http://127.0.0.1:8080/sayHi

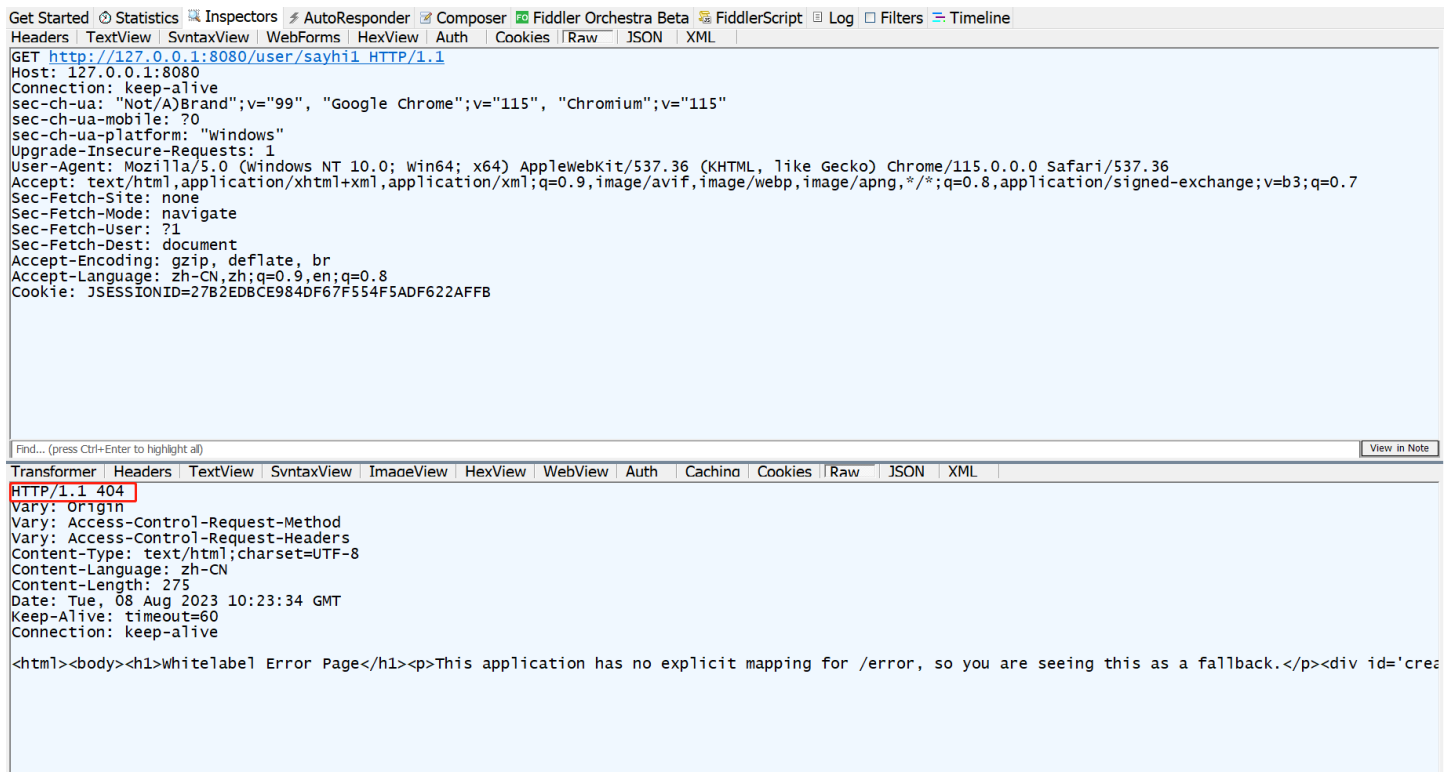
# Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Aug 08 20:59:33 CST 2023

There was an unexpected error (type=Not Found, status=404).

## 通过Fiddler观察Http请求



## 3.8.2 500

服务器出现内部错误. 一般是服务器的代码执行过程中遇到了一些特殊情况(服务器异常崩溃)会产生这个状态码.

**错误实例:**

```
1 @RestController
2 public class UserController {
3
4     @RequestMapping("/sayHi")
5     public String sayHi(){
6         int res = 10/0;
7         return "hello, Spring";
8     }
9 }
```

重启 Tomcat 服务器.

重新访问页面, 可以看到:

# Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Aug 08 21:00:57 CST 2023

There was an unexpected error (type=Internal Server Error, status=500).

此时, 程序后端控制台已经打印了具体的异常调用栈

```
java.lang.ArithmeticException: Create breakpoint : / by zero
    at com.example.demo.UserController.sayHi(UserController.java:11) ~[classes/:na] <4 internal lines>
    at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:205) ~[spring-web-5.3.29.jar:5.3.29]
    at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:150) ~[spring-web-5.3.29.jar:5.3.29]
    at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:117) ~[spring-webmvc-5.3.29.jar:5.3.29]
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:895) ~[spring-webmvc-5.3.29.jar:5.3.29]
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:808) ~[spring-webmvc-5.3.29.jar:5.3.29]
    at org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87) ~[spring-webmvc-5.3.29.jar:5.3.29]
```

异常信息里已经提示了出现异常的代码是UserController.java的第 11 行。

错误原因是算术异常: 除数为0

按照异常提示, 去解决对应的问题即可。

## 3.8.3 无法访问此网站

一般是 Tomcat 启动失败了。



### 无法访问此网站

127.0.0.1 拒绝了我们的连接请求。

请试试以下办法：

- 检查网络连接
- 检查代理服务器和防火墙

ERR\_CONNECTION\_REFUSED

重新加载

详情

打开Fiddler的话, 界面如下:

← → ↻ 🏠 ⓘ http://127.0.0.1:8080/sayHi

[Fiddler] The connection to '127.0.0.1' failed.

Error: ConnectionRefused (0x274d).

System.Net.Sockets.SocketException 由于目标计算机积极拒绝, 无法连接。 127.0.0.1:8080

这种情况一般是服务未成功启动, 也就是tomcat未启动,或者IP/端口写错

← → ↻ 🏠 ⓘ http://127.0.0.1:8081/sayHi

[Fiddler] The connection to '127.0.0.1' failed.

Error: ConnectionRefused (0x274d).

System.Net.Sockets.SocketException 由于目标计算机积极拒绝, 无法连接。 127.0.0.1:8081

### 3.8.4 小结

最开始学习Spring的时候, 会遇到很多问题, 更多是环境相关的问题.

我们不仅要学习 Spring 代码的基本写法, 更重要的是学习**排查错误的思路**.

程序猿调试 BUG 如同医生诊病.

一个有经验的程序猿和一个新手程序猿相比, 最大的优势往往不是代码写的多好, 而是调试效率有多高. 同一个问题可能新手花了几天花都无法解决的, 但是有经验的程序猿可能几分钟就搞定了.

熟悉 HTTP 协议能让我们调试问题事半功倍.

- 4xx 的状态码表示路径不存在, 往往需要检查 URL 是否正确, 和代码中设定的 Context Path 以及 Servlet Path 是否一致.
- 5xx 的状态码表示服务器出现错误, 往往需要观察页面提示的内容和 Tomcat 自身的日志, 观察是否存在报错.
- 出现连接失败往往意味着服务没有正确启动, 也需要观察 服务的自身日志是否有错误提示.

观察日志是调试程序的重要途径. 系统日志往往很多, 需要同学们耐心阅读, 经常阅读, 熟练了就能更快速的找到问题了.

## 4. 总结

Spring Boot 是为了快速开发 Spring 而诞生的, Spring Boot 具备:

- Spring Boot 提供了启动添加依赖的功能, 可以快速集成框架.
- 内置web服务器, 无需配置 Tomcat 等 Web服务器, 直接运行和部署程序.
- 可以完全抛弃繁琐的 XML, 使用注解和配置的方式进行开发.
- 支持更多的监控的指标, 可以更好的了解项目的运行情况等特点.

Spring Boot 可使用 Idea 或网页创建, 它的设计思想是约定大于配置, 类上标注 `@SpringBootApplication` 就可以启动 Spring Boot 项目了.

