

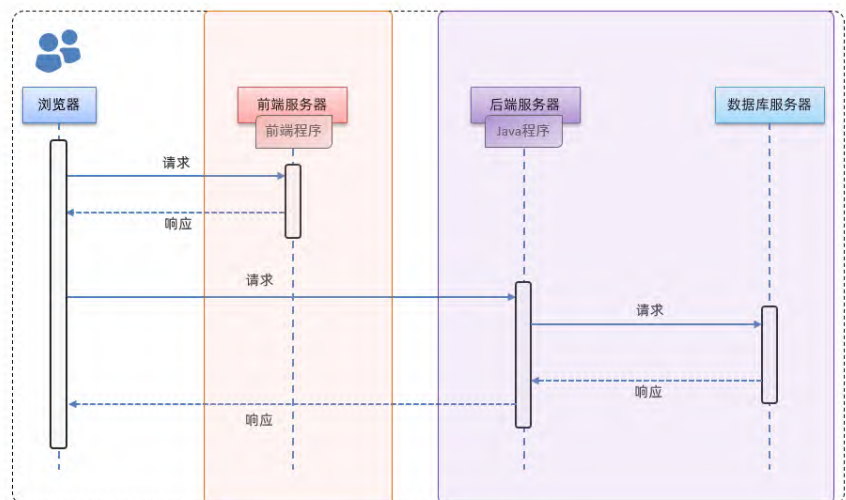
SpringBoot基础

课程内容

1. SpringBootWeb入门
2. HTTP协议
3. Web服务器-Tomcat

前言

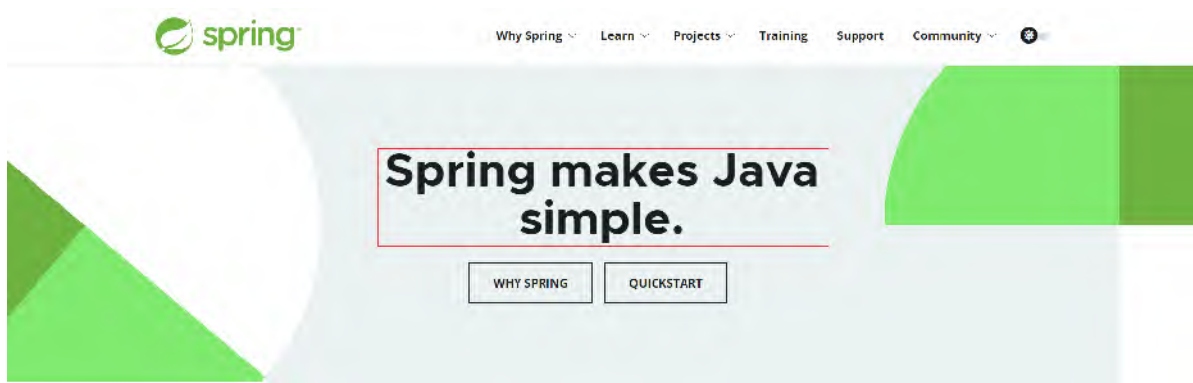
Web开发课程安排



下面我们将进入SpringBoot基础阶段的学习。

在没有正式的学习SpringBoot之前，我们要先来了解下什么是Spring。

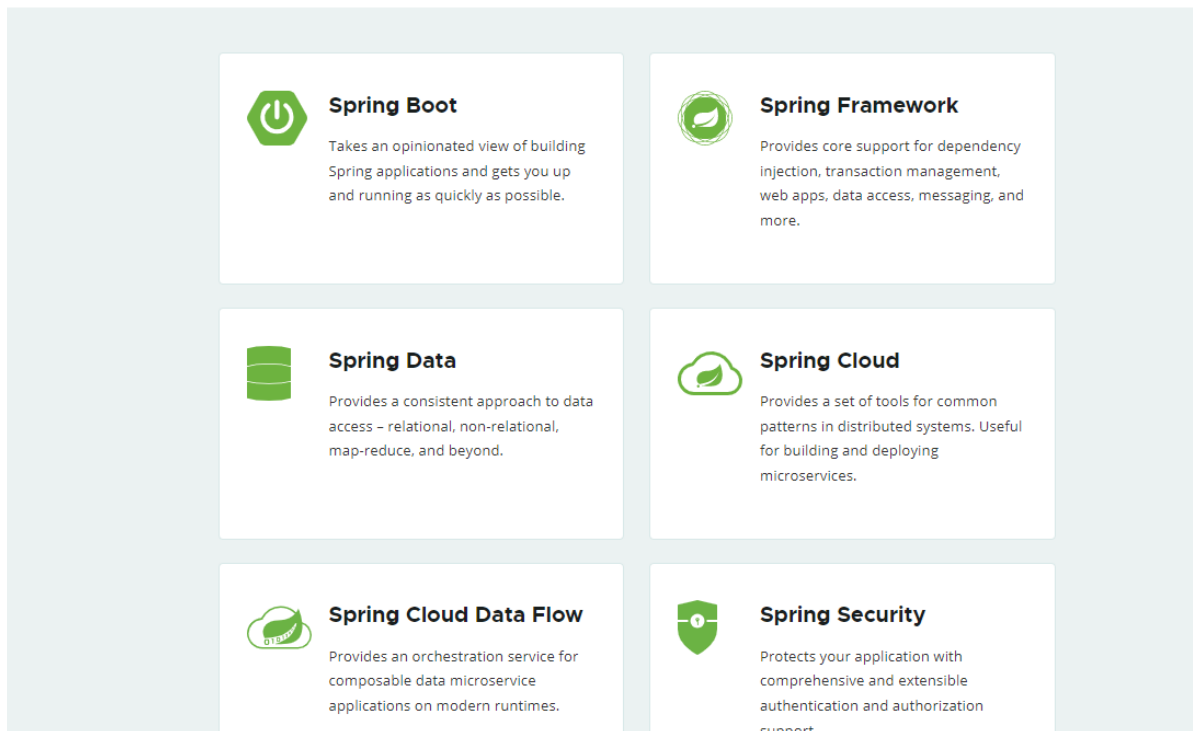
我们可以打开Spring的官网 (<https://spring.io>)，去看一下Spring的简介：Spring makes Java simple。



Spring的官方提供很多开源的项目，我们可以点击上面的projects，看到spring家族旗下的项目，按照流行程度排序为：

Projects

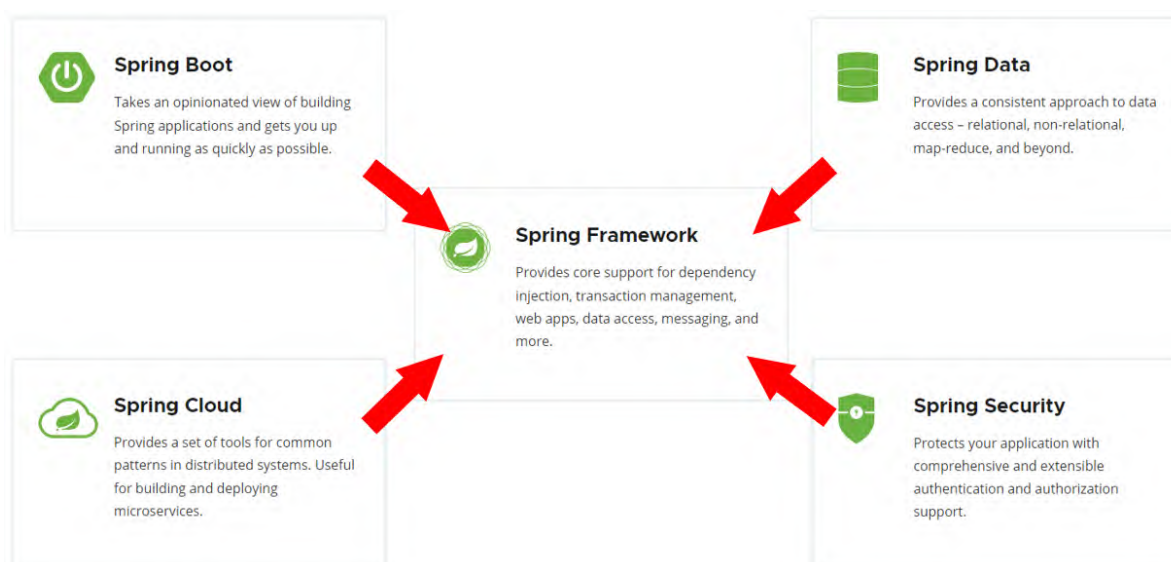
From configuration to security, web apps to big data—whatever the infrastructure needs of your application may be, there is a Spring Project to help you build it. Start small and use just what you need—Spring is modular by design.



Spring发展到今天已经形成了一种开发生态圈，Spring提供了若干个子项目，每个项目用于完成特定的功能。而我们在项目开发时，一般会偏向于选择这一套spring家族的技术，来解决对应领域的问题，那我们称这一套技术为spring全家桶。



而Spring家族旗下这么多的技术，最基础、最核心的是 SpringFramework。其他的spring家族的技术，都是基于SpringFramework的，SpringFramework中提供很多实用功能，如：依赖注入、事务管理、web开发支持、数据访问、消息服务等等。



而如果我们在项目中，直接基于SpringFramework进行开发，存在两个问题：配置繁琐、入门难度大。



所以基于此呢，spring官方推荐我们从另外一个项目开始学习，那就是目前最火爆的SpringBoot。

通过springboot就可以快速的帮我们构建应用程序，所以springboot呢，最大的特点有两个：

- 简化配置
- 快速开发

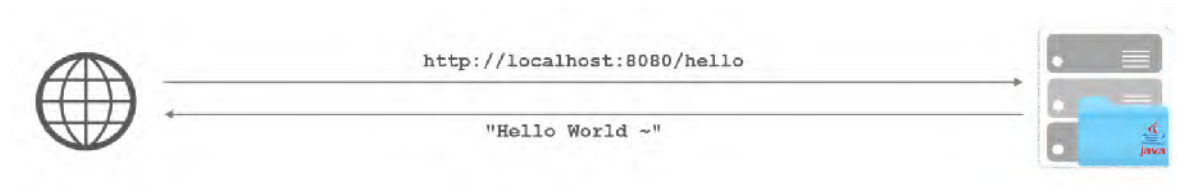
Spring Boot 可以帮助我们非常快速的构建应用程序、简化开发、提高效率 。

接下来，我们就直接通过一个SpringBoot的web入门程序，让大家快速感受一下，基于SpringBoot进行Web开发的便捷性。

1. SpringBootWeb快速入门

1.1 需求

需求：基于SpringBoot的方式开发一个web应用，浏览器发起请求/hello后，给浏览器返回字符串"Hello World ~"。



1.2 开发步骤

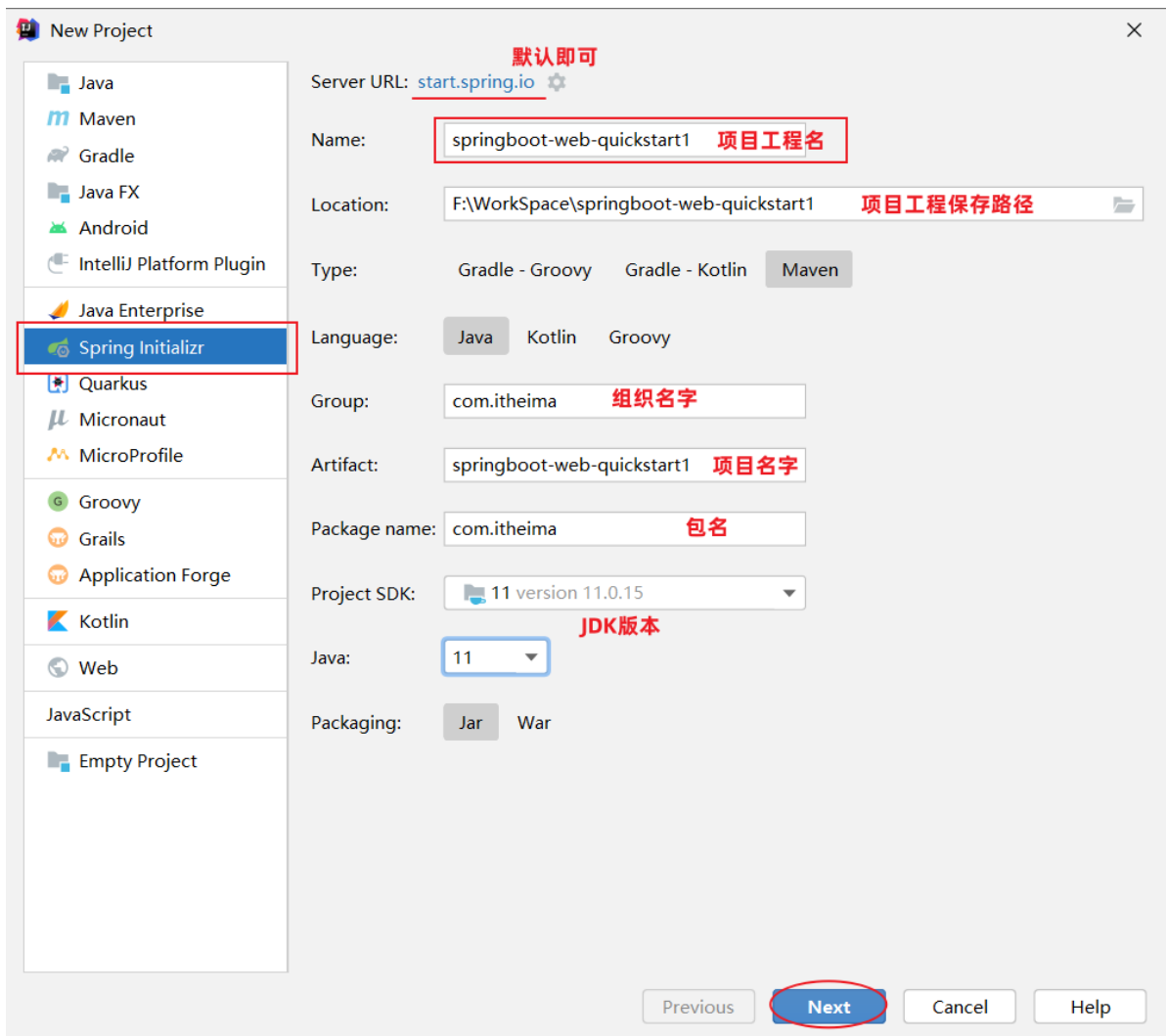
第1步：创建SpringBoot工程项目

第2步：定义HelloController类，添加方法hello，并添加注解

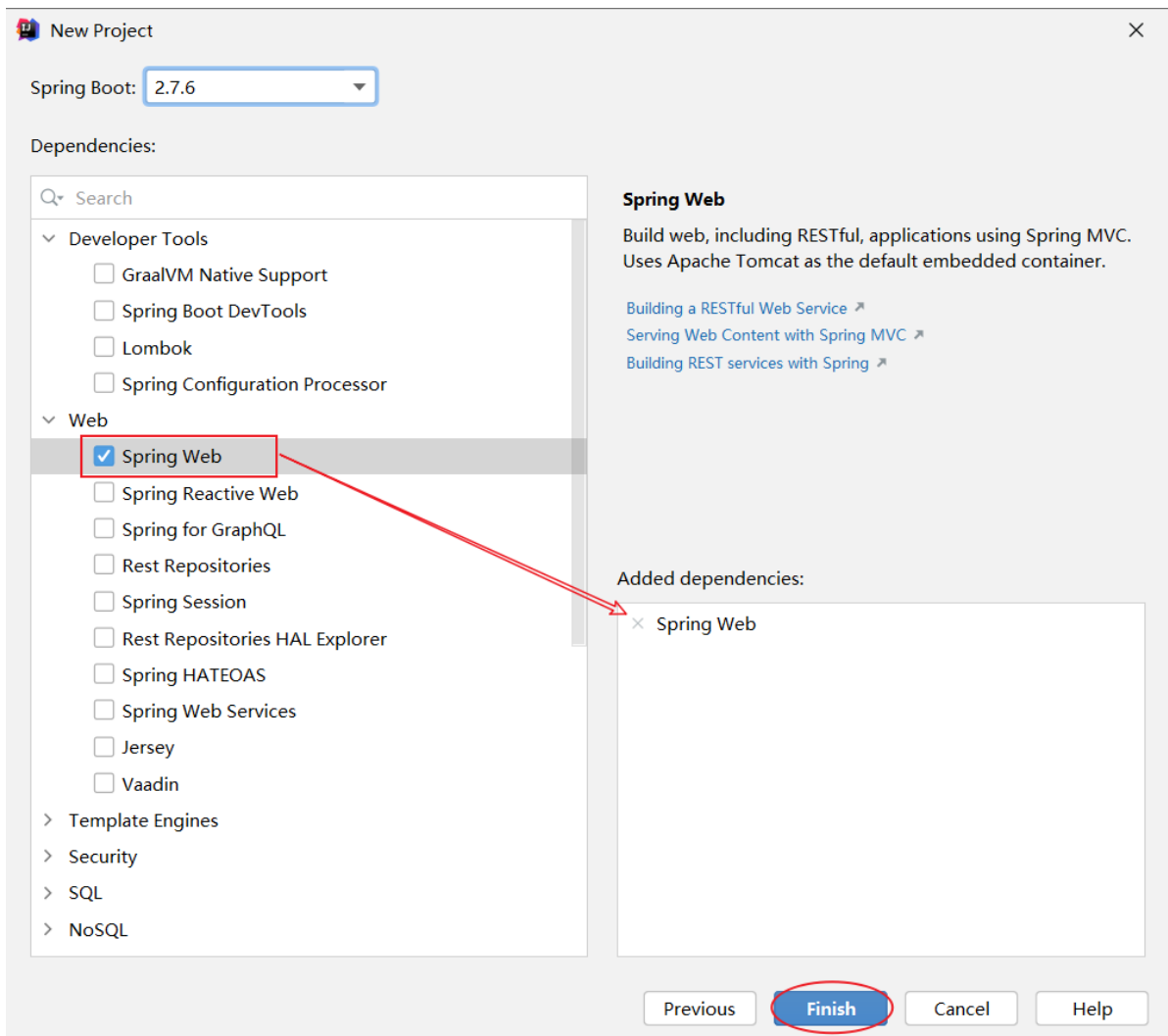
第3步：测试运行

1.2.1 创建SpringBoot工程（需要联网）

基于Spring官方骨架，创建SpringBoot工程。



基本信息描述完毕之后，勾选web开发相关依赖。



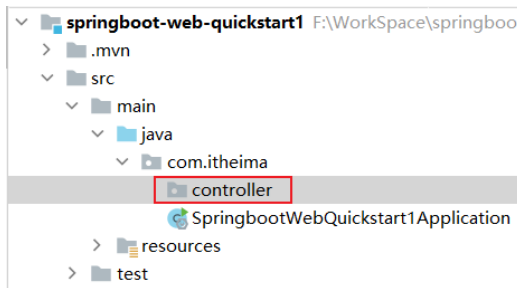
点击Finish之后，就会联网创建这个SpringBoot工程，创建好之后，结构如下：

- 注意：在联网创建过程中，会下载相关资源（请耐心等待）

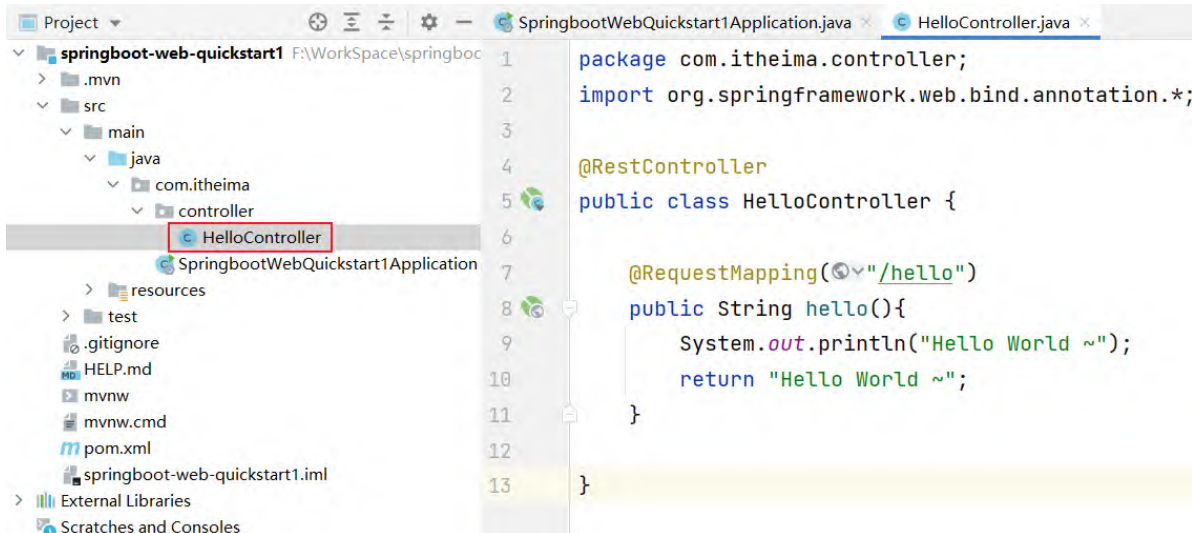


1.2.2 定义请求处理类

在com.itheima这个包下创建一个子包controller



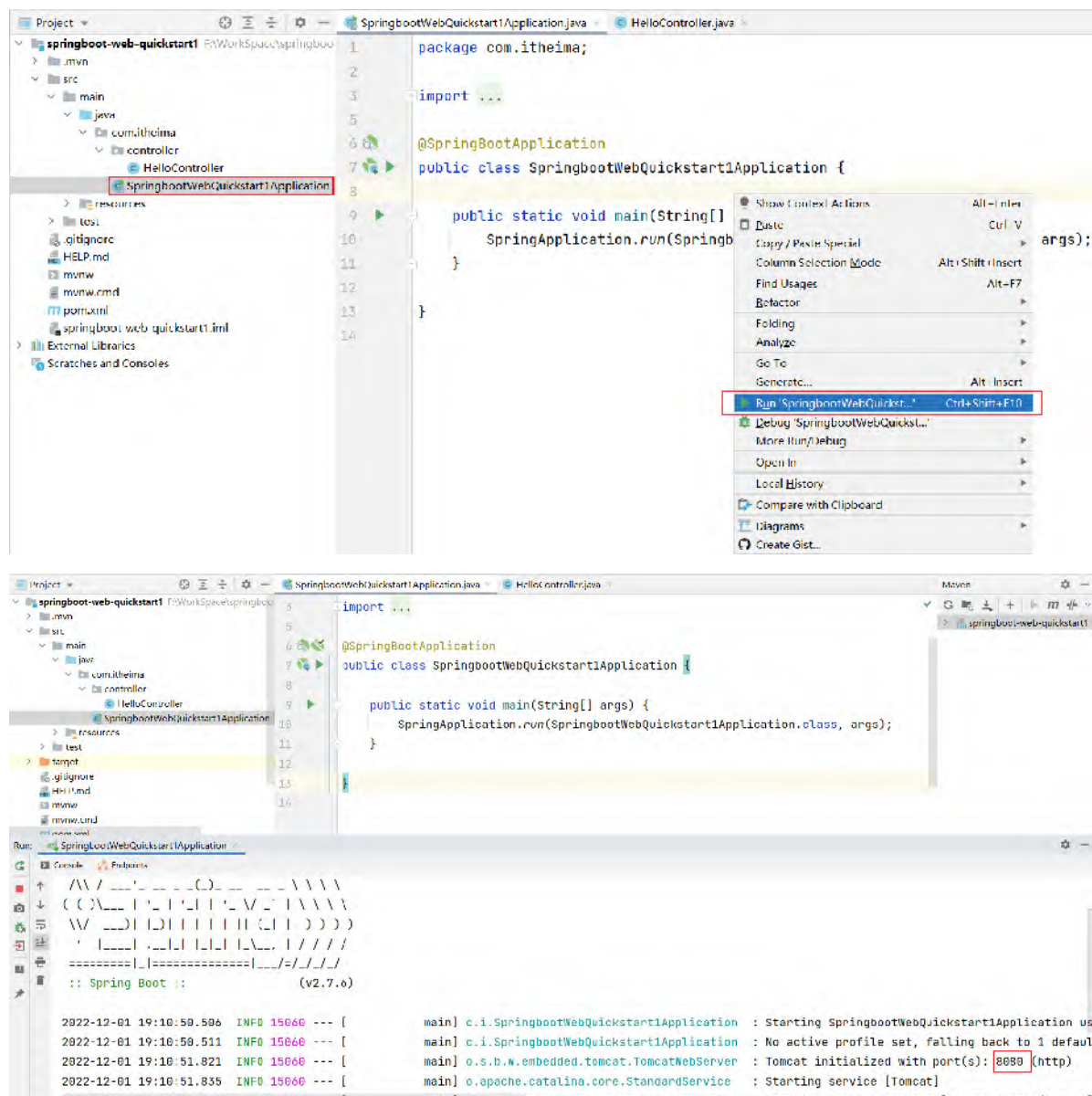
然后在controller包下新建一个类：HelloController



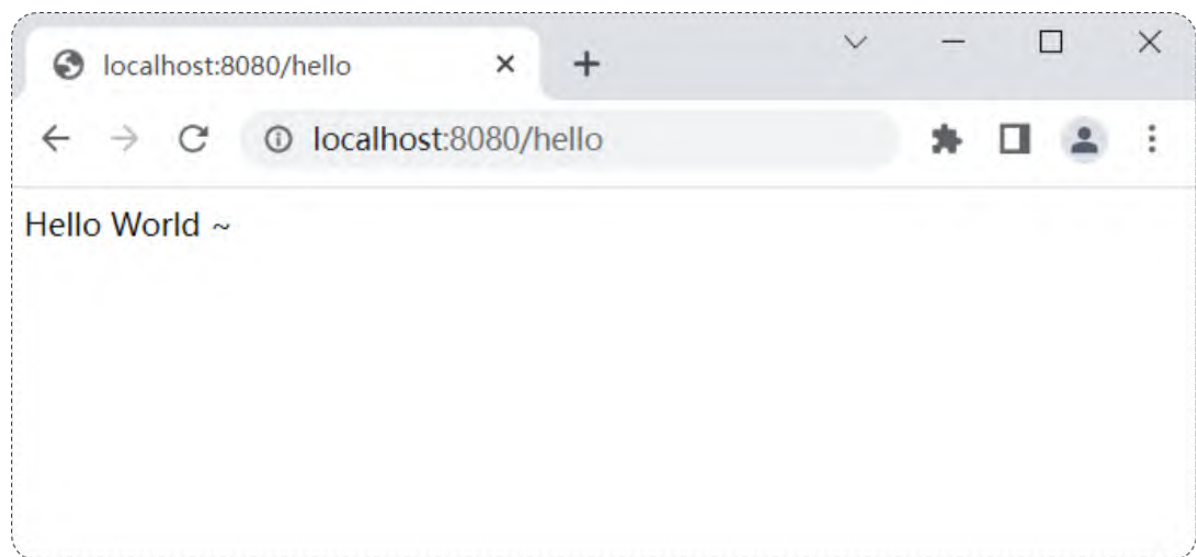
```
1 package com.itheima.controller;
2 import org.springframework.web.bind.annotation.*;
3
4 @RestController
5 public class HelloController {
6
7     @RequestMapping("/hello")
8     public String hello(){
9         System.out.println("Hello World ~");
10        return "Hello World ~";
11    }
12
13 }
```

1.2.3 运行测试

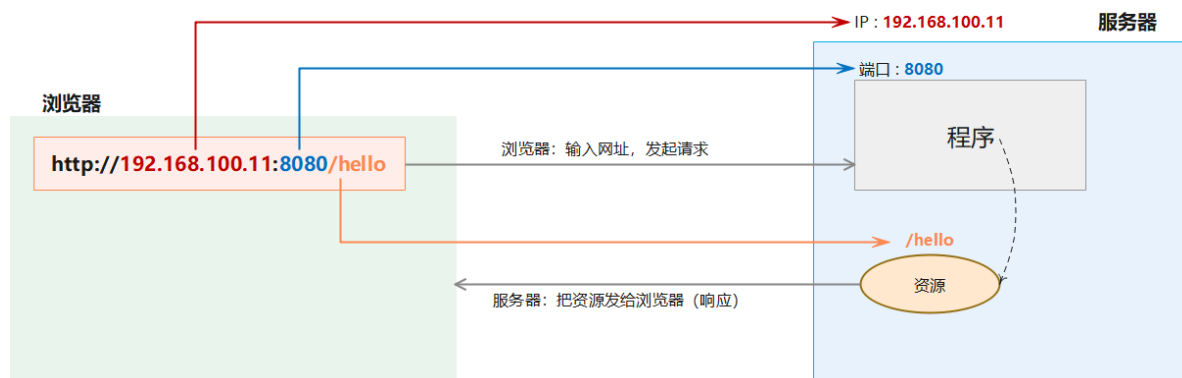
运行SpringBoot自动生成的引导类



打开浏览器，输入 <http://localhost:8080/hello>



1.3 Web分析



浏览器:

- 输入网址: `http://192.168.100.11:8080/hello`
 - 通过IP地址192.168.100.11定位到网络上的一台计算机

我们之前在浏览器中输入的localhost, 就是127.0.0.1 (本机)

- 通过端口号8080找到计算机上运行的程序

`localhost:8080` , 意思是在本地计算机中找到正在运行的8080端口的程序

- `/hello`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)

`localhost:8080/hello` , 意思是向本地计算机中的8080端口程序, 获取资源位置是/hello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: (可以理解为ServerSocket)

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

我们在JavaSE阶段学习网络编程时, 有讲过网络三要素:

- IP : 网络中计算机的唯一标识
- 端口 : 计算机中运行程序的唯一标识
- 协议 : 网络中计算机之间交互的规则

问题: 浏览器和服务器两端进行数据交互, 使用什么协议?

2. HTTP协议

2.1 HTTP-概述

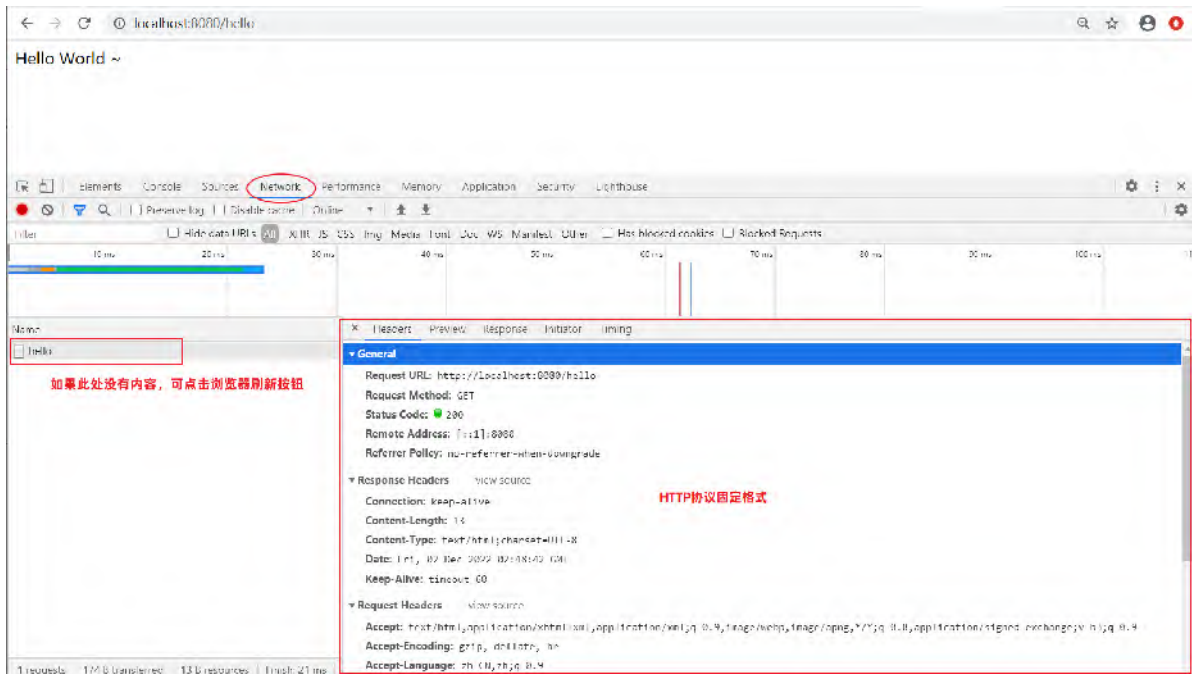
2.1.1 介绍



HTTP: Hyper Text Transfer Protocol (超文本传输协议), 规定了浏览器与服务器之间数据传输的规则。

- http是互联网上应用最为广泛的一种网络协议
- http协议要求: 浏览器在向服务器发送请求数据时, 或是服务器在向浏览器发送响应数据时, 都必须按照固定的格式进行数据传输

如果想知道http协议的数据传输格式有哪些, 可以打开浏览器, 点击 **F12** 打开开发者工具, 点击 **Network** 来查看



浏览器向服务器进行请求时：

- 服务器按照固定的格式进行解析



服务器向浏览器进行响应时：

- 浏览器按照固定的格式进行解析



所以，我们学习HTTP主要就是学习请求和响应数据的具体格式内容。

2.2.2 特点

我们刚才初步认识了HTTP协议，那么我们在看看HTTP协议有哪些特点：

- 基于TCP协议： 面向连接，安全

TCP是一种面向连接的(建立连接之前是需要经过三次握手)、可靠的、基于字节流的传输层通信协议,在数据传输方面更安全

- **基于请求-响应模型:** 一次请求对应一次响应(先请求后响应)

请求和响应是一一对应关系,没有请求,就没有响应

- **HTTP协议是无状态协议:** 对于数据没有记忆能力。每次请求-响应都是独立的

无状态指的是客户端发送HTTP请求给服务端之后,服务端根据请求响应数据,响应完后,不会记录任何信息。

- 缺点: 多次请求间不能共享数据
- 优点: 速度快

请求之间无法共享数据会引发的问题:

- 如: 京东购物。加入购物车和去购物车结算是两次请求
- 由于HTTP协议的无状态特性,加入购物车请求响应结束后,并未记录加入购物车是何商品
- 发起去购物车结算的请求后,因为无法获取哪些商品加入了购物车,会导致此次请求无法正确展示数据

具体使用的时候,我们发现京东是可以正常展示数据的,原因是Java早已考虑到这个问题,并提出了使用会话技术(Cookie、Session)来解决这个问题。具体如何做,我们后面课程中会讲到。

刚才提到HTTP协议是规定了请求和响应数据的格式,那具体的格式是什么呢?

2.2 HTTP-请求协议

浏览器和服务器是按照HTTP协议进行数据通信的。

HTTP协议又分为: 请求协议和响应协议

- 请求协议: 浏览器将数据以请求格式发送到服务器
 - 包括: **请求行**、**请求头**、**请求体**
- 响应协议: 服务器将数据以响应格式返回给浏览器
 - 包括: **响应行**、**响应头**、**响应体**

在HTTP1.1版本中，浏览器访问服务器的几种方式：

请求方式	请求说明
GET	获取资源。 向特定的资源发出请求。例： http://www.baidu.com/s?wd=itheima
POST	传输实体主体。 向指定资源提交数据进行处理请求（例：上传文件），数据被包含在请求体中。
OPTIONS	返回服务器针对特定资源所支持的HTTP请求方式。 因为并不是所有的服务器都支持规定的方法，为了安全有些服务器可能会禁止掉一些方法，例如：DELETE、PUT等。那么OPTIONS就是用来询问服务器支持的方法。
HEAD	获得报文首部。 HEAD方法类似GET方法，但是不同的是HEAD方法不要求返回数据。通常用于确认URI的有效性及资源更新时间等。
PUT	传输文件。 PUT方法用来传输文件。类似FTP协议，文件内容包含在请求报文的实体中，然后请求保存到URL指定的服务器位置。
DELETE	删除文件。 请求服务器删除Request-URI所标识的资源
TRACE	追踪路径。 回显服务器收到的请求，主要用于测试或诊断
CONNECT	要求用隧道协议连接代理。 HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器

在我们实际应用中常用的也就是 ： GET、POST

GET方式的请求协议：

```
GET /brand/findAll?name=OPPO&status=1 HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/...
```

- 请求行：HTTP请求中的第一行数据。由：请求方式、资源路径、协议/版本 组成（之间使用空格分隔）
 - 请求方式：GET
 - 资源路径：/brand/findAll?name=OPPO&status=1
 - 请求路径：/brand/findAll
 - 请求参数：name=OPPO&status=1
 - 请求参数是以key=value形式出现
 - 多个请求参数之间使用 & 连接
 - 请求路径和请求参数之间使用 ? 连接
 - 协议/版本：HTTP/1.1
- 请求头：第二行开始，上图黄色部分内容就是请求头。格式为key: value形式
 - http是个无状态的协议，所以在请求头设置浏览器的一些自身信息和想要响应的形式。这样服务器在收到信息后，就可以知道是谁，想干什么了

常见的HTTP请求头有：

- 1 Host：表示请求的主机名
- 2
- 3 User-Agent：浏览器版本。 例如：Chrome浏览器的标识类似Mozilla/5.0 ...Chrome/79 ， IE浏览器的标识类似Mozilla/5.0 (Windows NT ...)like Gecko
- 4
- 5 Accept：表示浏览器能接收的资源类型，如text/*， image/*或者*/ *表示所有；
- 6
- 7 Accept-Language：表示浏览器偏好的语言，服务器可以据此返回不同语言的网页；
- 8
- 9 Accept-Encoding：表示浏览器可以支持的压缩类型，例如gzip， deflate等。
- 10
- 11 Content-Type：请求主体的数据类型
- 12
- 13 Content-Length：数据主体的大小（单位：字节）

举例说明：服务端可以根据请求头中的内容来获取客户端的相关信息，有了这些信息服务端就可以处理不同的业务需求。

比如：

- 不同浏览器解析HTML和CSS标签的结果会有不一致，所以就会导致相同的代码在不同的浏览器会出现不同的效果

- 服务端根据客户端请求头中的数据获取到客户端的浏览器类型，就可以根据不同的浏览器设置不同的代码来达到一致的效果（这就是我们常说的浏览器兼容问题）

- 请求体：存储请求参数
 - GET请求的请求参数在请求行中，故不需要设置请求体

POST方式的请求协议：

```
POST /brand HTTP/1.1
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Content-Length: 161
Content-Type: application/json;charset=UTF-8
Cookie: Idea-8296eb32=841b16f0-0cfe-495a-9cc9-d5aaa71501a6; JSESSIONID=0FDE4E430876BD9C5C955F061207386F
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/...

{"status":1,"brandName":"黑马","companyName":"黑马程序员","id":"","description":"黑马程序员"}
```

- 请求行 (以上图中红色部分)：包含请求方式、资源路径、协议/版本
 - 请求方式：POST
 - 资源路径：/brand
 - 协议/版本：HTTP/1.1
- 请求头 (以上图中黄色部分)
- 请求体 (以上图中绿色部分)：存储请求参数
 - 请求体和请求头之间是有一个空行隔开（作用：用于标记请求头结束）

GET请求和POST请求的区别：

区别方式	GET请求	POST请求
请求参数	请求参数在请求行中。 例：/brand/findAll?name=OPPO&status=1	请求参数在请求体中
请求参数长度	请求参数长度有限制（浏览器不同限制也不同）	请求参数长度没有限制
安全性	安全性低。原因：请求参数暴露在浏览器地址栏中。	安全性相对高

2.3 HTTP-响应协议

2.3.1 格式介绍

与HTTP的请求一样，HTTP响应的数据也分为3部分：**响应行**、**响应头**、**响应体**



- 响应行 (以上图中红色部分)：响应数据的第一行。响应行由 协议及版本、响应状态码、状态码描述 组成
 - 协议/版本：HTTP/1.1
 - 响应状态码：200
 - 状态码描述：OK
- 响应头 (以上图中黄色部分)：响应数据的第二行开始。格式为key: value形式
 - http是个无状态的协议，所以可以在请求头和响应头中设置一些信息和想要执行的动作，这样，对方在收到信息后，就可以知道你是谁，你想干什么

常见的HTTP响应头有：

- 1 Content-Type: 表示该响应内容的类型，例如text/html, image/jpeg ；
- 2
- 3 Content-Length: 表示该响应内容的长度（字节数）；
- 4
- 5 Content-Encoding: 表示该响应压缩算法，例如gzip ；
- 6
- 7 Cache-Control: 指示客户端应如何缓存，例如max-age=300表示可以最多缓存300秒 ；
- 8
- 9 Set-Cookie: 告诉浏览器为当前页面所在的域设置cookie ；

- 响应体 (以上图中绿色部分)： 响应数据的最后一部分。存储响应的数据
 - 响应体和响应头之间有一个空行隔开（作用：用于标记响应头结束）

2.3.2 响应状态码

状态码分类	说明
1xx	响应中 --- 临时状态码。表示请求已经接受，告诉客户端应该继续请求或者如果已经完成则忽略
2xx	成功 --- 表示请求已经被成功接收，处理已完成
3xx	重定向 --- 重定向到其它地方，让客户端再发起一个请求以完成整个处理
4xx	客户端错误 --- 处理发生错误，责任在客户端，如：客户端的请求一个不存在的资源，客户端未被授权，禁止访问等
5xx	服务器端错误 --- 处理发生错误，责任在服务端，如：服务端抛出异常，路由出错，HTTP版本不支持等

参考：资料/SpringbootWeb/响应状态码.md

关于响应状态码，我们先主要认识三个状态码，其余的等后期用到了再去掌握：

- 200 ok 客户端请求成功
- 404 Not Found 请求资源不存在
- 500 Internal Server Error 服务端发生不可预期的错误

2.4 HTTP-协议解析

将资料中准备好的Demo工程，导入到我们的IDEA中，有一个Server.java类，这里面就是自定义的一个服务器代码，主要使用的是 `ServerSocket` 和 `Socket`

说明：以下代码大家不需要自己写，我们主要是通过代码，让大家了解到服务器针对HTTP协议的解析机制

```
1 package com.itheima;
2
3 import java.io.*;
4 import java.net.ServerSocket;
5 import java.net.Socket;
6 import java.nio.charset.StandardCharsets;
7
8 /*
```

```
9      * 自定义web服务器
10     */
11     public class Server {
12         public static void main(String[] args) throws IOException {
13             ServerSocket ss = new ServerSocket(8080); // 监听指定端口
14             System.out.println("server is running...");
15
16             while (true){
17                 Socket sock = ss.accept();
18                 System.out.println("connected from " +
sock.getRemoteSocketAddress());
19                 Thread t = new Handler(sock);
20                 t.start();
21             }
22         }
23     }
24
25     class Handler extends Thread {
26         Socket sock;
27
28         public Handler(Socket sock) {
29             this.sock = sock;
30         }
31
32         public void run() {
33             try (InputStream input = this.sock.getInputStream();
34                 OutputStream output = this.sock.getOutputStream()) {
35                 handle(input, output);
36             } catch (Exception e) {
37                 try {
38                     this.sock.close();
39                 } catch (IOException ioe) {
40                 }
41                 System.out.println("client disconnected.");
42             }
43         }
44
45         private void handle(InputStream input, OutputStream output)
throws IOException {
46             BufferedReader reader = new BufferedReader(new
InputStreamReader(input, StandardCharsets.UTF_8));
47             BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(output, StandardCharsets.UTF_8));
48             // 读取HTTP请求:
```

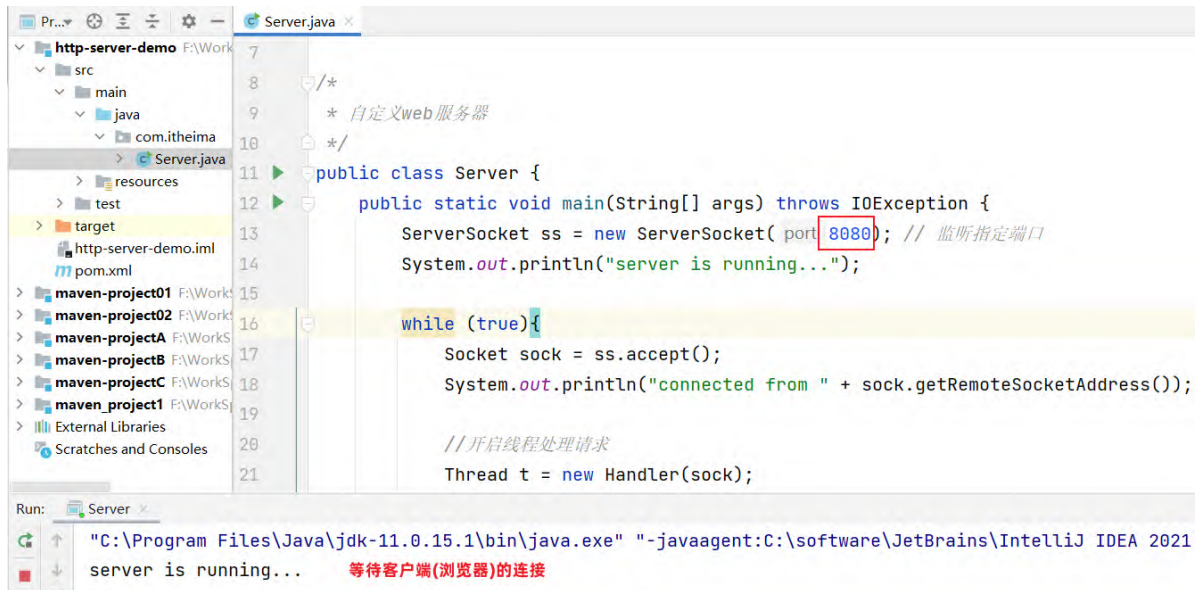
```

49         boolean requestOk = false;
50         String first = reader.readLine();
51         if (first.startsWith("GET / HTTP/1.")) {
52             requestOk = true;
53         }
54         for (;;) {
55             String header = reader.readLine();
56             if (header.isEmpty()) { // 读取到空行时, HTTP Header读取完
毕
57                 break;
58             }
59             System.out.println(header);
60         }
61         System.out.println(requestOk ? "Response OK" : "Response
Error");
62
63         if (!requestOk) { // 发送错误响应:
64             writer.write("HTTP/1.0 404 Not Found\r\n");
65             writer.write("Content-Length: 0\r\n");
66             writer.write("\r\n");
67             writer.flush();
68         } else { // 发送成功响应:
69             //读取html文件, 转换为字符串
70             InputStream is =
Server.class.getClassLoader().getResourceAsStream("html/a.html");
71             BufferedReader br = new BufferedReader(new
InputStreamReader(is));
72             StringBuilder data = new StringBuilder();
73             String line = null;
74             while ((line = br.readLine()) != null) {
75                 data.append(line);
76             }
77             br.close();
78             int length =
data.toString().getBytes(StandardCharsets.UTF_8).length;
79
80             writer.write("HTTP/1.1 200 OK\r\n");
81             writer.write("Connection: keep-alive\r\n");
82             writer.write("Content-Type: text/html\r\n");
83             writer.write("Content-Length: " + length + "\r\n");
84             writer.write("\r\n"); // 空行标识Header和Body的分隔
85             writer.write(data.toString());
86             writer.flush();
87         }

```

```
88     }
89 }
90
```

启动ServerSocket程序：



浏览器输入：`http://localhost:8080` 就会访问到ServerSocket程序

- ServerSocket程序，会读取服务器上 `html/a.html` 文件，并把文件数据发送给浏览器
- 浏览器接收到[a.html](#)文件中的数据后进行解析，显示以下内容

序号	品牌名称	企业名称
010	三只松鼠	三只松鼠
009	优衣库	优衣库
008	小米	小米科技有限公司

现在大家知道了服务器是可以使用java完成编写，是可以接受页面发送的请求和响应数据给前端浏览器的，而在开发中真正用到的Web服务器，我们不会自己写的，都是使用目前比较流行的web服务器。

如：Tomcat



3. WEB服务器-Tomcat

3.1 简介

3.1.1 服务器概述

服务器硬件

- 指的也是计算机，只不过服务器要比我们日常使用的计算机大很多。



服务器，也称伺服器。是提供计算服务的设备。由于服务器需要响应服务请求，并进行处理，因此一般来说服务器应具备承担服务并且保障服务的能力。

服务器的构成包括处理器、硬盘、内存、系统总线等，和通用的计算机架构类似，但是由于需要提供高可靠的服务，因此在处理能力、稳定性、可靠性、安全性、可扩展性、可管理性等方面要求较高。

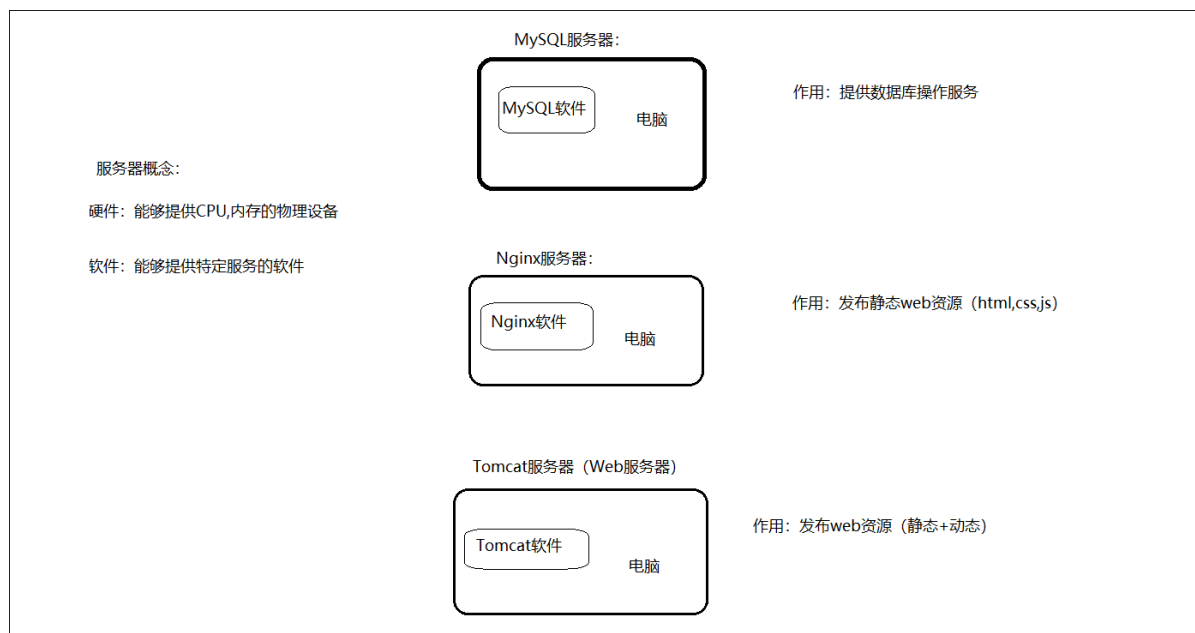
在网络环境下，根据服务器提供的服务类型不同，可分为：文件服务器，数据库服务器，应用程序服务器，WEB服务器等。

服务器只是一台设备，必须安装服务器软件才能提供相应的服务。

服务器软件

服务器软件：基于ServerSocket编写的程序

- 服务器软件本质是一个运行在服务器设备上的应用程序
- 能够接收客户端请求，并根据请求给客户端响应数据



3.1.2 Web服务器

Web服务器是一个应用程序(软件)，对HTTP协议的操作进行封装，使得程序员不必直接对协议进行操作(不用程序员自己写代码去解析http协议规则)，让Web开发更加便捷。主要功能是"提供网上信息浏览服务"。



Web服务器是安装在服务器端的一款软件，将来我们把自己写的Web项目部署到Tomcat服务器软件中，当Web服务器软件启动后，部署在Web服务器软件中的页面就可以直接通过浏览器来访问了。

Web服务器软件使用步骤

- 准备静态资源
- 下载安装Web服务器软件
- 将静态资源部署到Web服务器上
- 启动Web服务器使用浏览器访问对应的资源

第1步：准备静态资源

- 在提供的资料中找到静态资源文件

资料 > SpringbootWeb > 02. web服务器-tomcat					搜索"02. web服务器-tomc..."
名称	修改日期	类型	大小		
部署项目	2022/10/16 20:34	文件夹			
apache-tomcat-9.0.27-windows-x64....	2022/7/28 17:35	WinRAR ZIP 压缩...	12,626 KB		

第2步：下载安装Web服务器软件

< SpringbootWeb > 02. web服务器-tomcat					搜索"02. web服务器-tomc..."
名称	修改日期	类型	大小	名称	修改日期
部署项目	2022/10/16 20:34	文件夹		apache-maven-3.6.3	2022/11/25 11:05
apache-tomcat-9.0.27-windows-x64....	2022/7/28 17:35	WinRAR ZIP 压缩...	12,626 KB	apache-tomcat-8.5.31	2018/4/27 21:26
				apache-tomcat-9.0.27	2019/10/7 10:59

解压即安装

第3步：将静态资源部署到Web服务器上

> 02. web服务器-tomcat > 部署项目					搜索"部署项目"
名称	修改日期	类型	大小	名称	修改日期
demo	2022/10/16 20:34	文件夹		demo	2022/10/7 16:05
				docs	2019/10/7 10:57
				examples	2019/10/7 10:58
				host-manager	2019/10/7 10:58
				manager	2019/10/7 10:58
				ROOT	2019/10/7 10:57

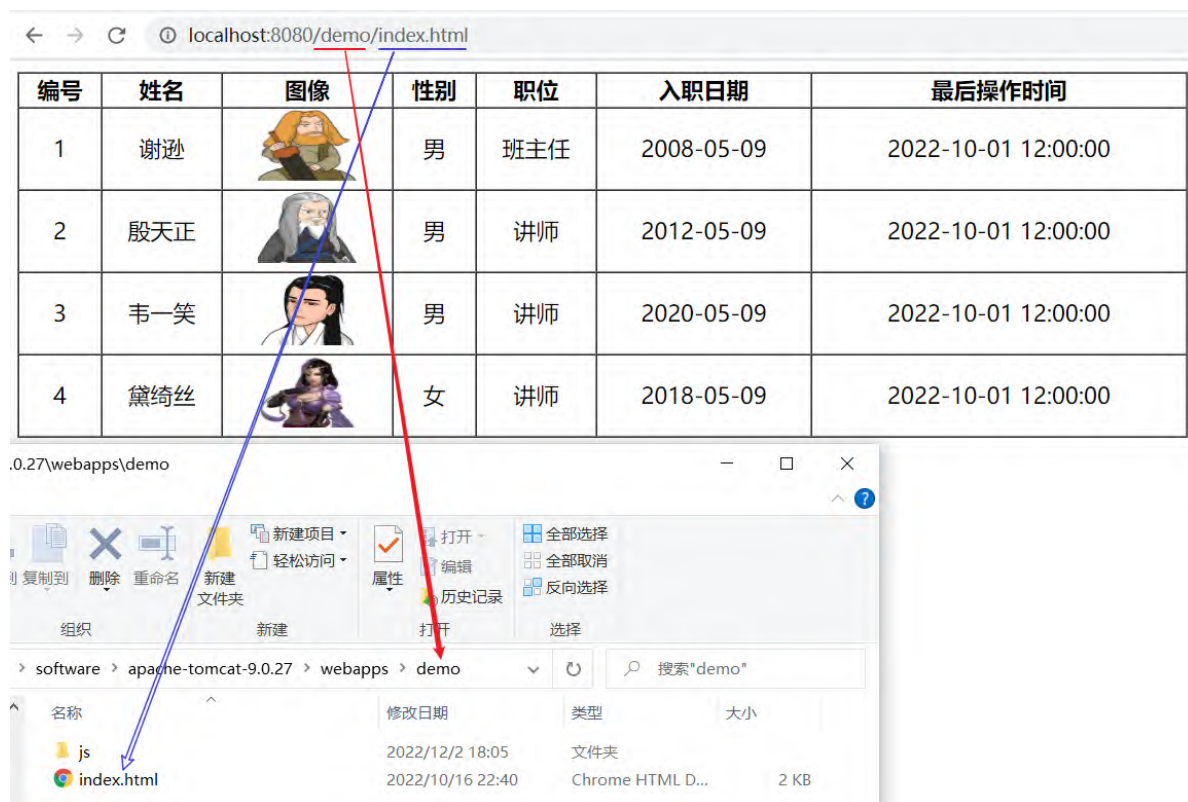
复制到Tomcat安装路径下的webapps目录内

第4步：启动Web服务器使用浏览器访问对应的资源

> software > apache-tomcat-9.0.27 > bin					搜索"bin"
名称	修改日期	类型	大小		
digest.bat	2019/10/7 10:57	Windows 批处理...	3 KB		
digest.sh	2019/10/7 10:58	Shell Script	2 KB		
makebase.bat	2019/10/7 10:57	Windows 批处理...	4 KB		
makebase.sh	2019/10/7 10:58	Shell Script	4 KB		
service.bat	2019/10/7 10:57	Windows 批处理...	9 KB		
setclasspath.bat	2019/10/7 10:57	Windows 批处理...	4 KB		
setclasspath.sh	2019/10/7 10:58	Shell Script	4 KB		
shutdown.bat	2019/10/7 10:57	Windows 批处理...	2 KB		
shutdown.sh	2019/10/7 10:58	Shell Script	2 KB		
startup.bat	2019/10/7 10:57	Windows 批处理...	2 KB		
startup.sh	2019/10/7 10:58	Shell Script	2 KB		
tcnative-1.dll	2019/10/7 10:58	应用程序扩展	2,532 KB		
tomcat9.exe	2019/10/7 10:58	应用程序	122 KB		
tomcat9w.exe	2019/10/7 10:58	应用程序	119 KB		
tomcat-juli.jar	2019/10/7 10:57	Executable Jar File	47 KB		
tomcat-native.tar.gz	2019/10/7 10:58	WinRAR 压缩文件	410 KB		
tool-wrapper.bat	2019/10/7 10:57	Windows 批处理...	5 KB		
tool-wrapper.sh	2019/10/7 10:58	Shell Script	6 KB		
version.bat	2019/10/7 10:57	Windows 批处理...	2 KB		
version.sh	2019/10/7 10:58	Shell Script	2 KB		

双击启动Tomcat

浏览器输入：<http://localhost:8080/demo/index.html>



上述内容在演示的时候，使用的是Apache下的Tomcat软件，至于Tomcat软件如何使用，后面会详细的讲到。而对于Web服务器来说，实现的方案有很多，Tomcat只是其中的一种，而除了Tomcat以外，还有很多优秀的Web服务器，比如：



Tomcat就是一款软件，我们主要是以学习如何去使用为主。具体我们会从以下这些方向去学习：

1. 简介：初步认识下Tomcat
2. 基本使用：安装、卸载、启动、关闭、配置和项目部署，这些都是对Tomcat的基本操作
3. IDEA中如何创建Maven Web项目
4. IDEA中如何使用Tomcat，后面这两个都是我们以后开发经常会用到的方式

首选我们来认识下Tomcat。

3.1.3 Tomcat

Tomcat服务器软件是一个免费的开源的web应用服务器。是Apache软件基金会的一个核心项目。由Apache, Sun和其他一些公司及个人共同开发而成。

由于Tomcat只支持Servlet/JSP少量JavaEE规范, 所以是一个开源免费的轻量级Web服务器。

JavaEE规范: JavaEE => Java Enterprise Edition (Java企业版)

JavaEE规范就是指Java企业级开发的技术规范总和。包含13项技术规范: JDBC、JNDI、EJB、RMI、JSP、Servlet、XML、JMS、Java IDL、JTS、JTA、JavaMail、JAF

因为Tomcat支持Servlet/JSP规范, 所以Tomcat也被称为Web容器、Servlet容器。JavaWeb程序需要依赖Tomcat才能运行。

Tomcat的官网: <https://tomcat.apache.org/>



3.2 基本使用

3.2.1 下载

直接从官方网站下载: <https://tomcat.apache.org/download-90.cgi>

← → ↻ tomcat.apache.org/download-90.cgi

Tomcat 10.1 (beta)
Tomcat 10.0
Tomcat 9.0
Tomcat 8.5
Tomcat Connectors
Tomcat Native 2
Tomcat Native 1.2
Wiki
Migration Guide
Presentations
Specifications

Problems?
Security Reports
Find help
FAQ
Mailing Lists
Bug Database
IRC

Get Involved
Overview
Source code
Buildbot
Translations
Tools

Media
Twitter
YouTube
Blog

You are currently using **https://dlcdn.apache.org/**. If you encounter a problem with mirrors list) that should be available.

Other mirrors:

9.0.65

Please see the [README](#) file for packaging information. It explains what every distribution contains.

Binary Distributions

- Core:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
 - [32-bit Windows zip \(pgp, sha512\)](#)
 - [64-bit Windows zip \(pgp, sha512\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
 - [tar.gz \(pgp, sha512\)](#)
- Deployer:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
- Embedded:
 - [tar.gz \(pgp, sha512\)](#)
 - [zip \(pgp, sha512\)](#)

Source Code Distributions

- [tar.gz \(pgp, sha512\)](#)
- [zip \(pgp, sha512\)](#)

Tomcat软件类型说明：

- tar.gz文件，是linux和mac操作系统下的压缩版本
- zip文件，是window操作系统下压缩版本（我们选择zip文件）

大家可以自行下载，也可以使用资料中已经下载好的资源，

Tomcat的软件程序 ： /资料/SpringbootWeb/apache-tomcat-9.0.27-windows-x64.zip

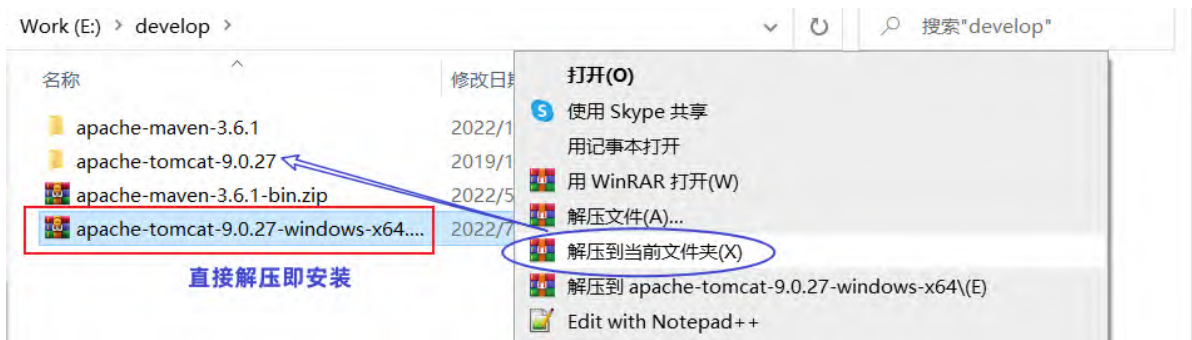


apache-tomcat-9.0.27-windows-x64.zip

3.2.2 安装与卸载

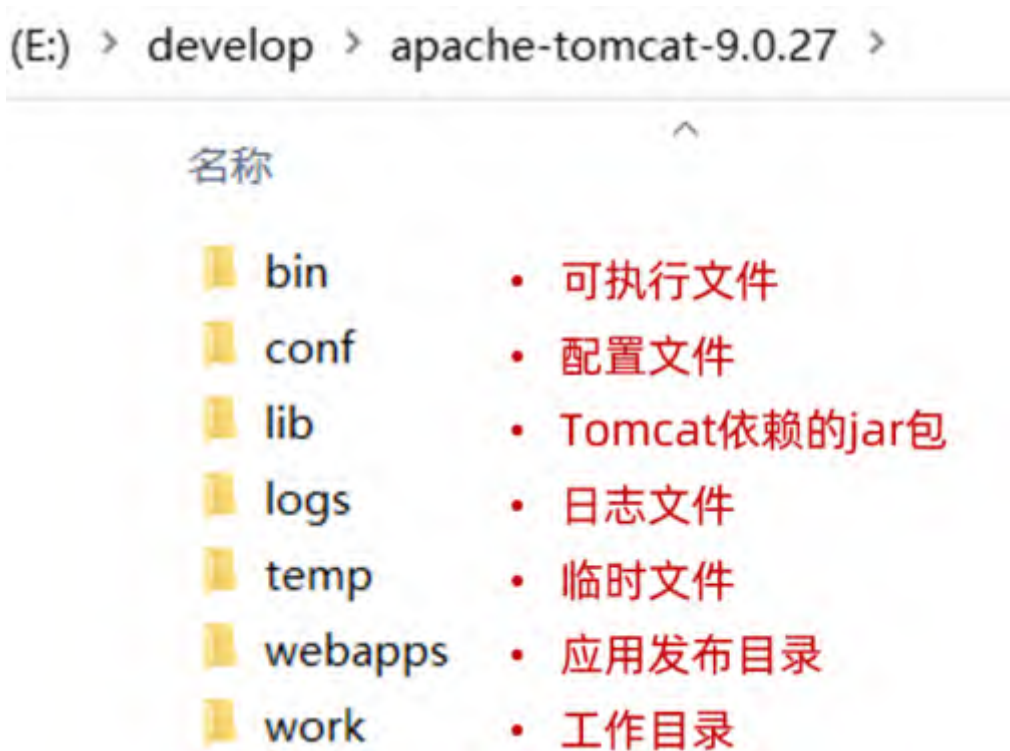
安装： Tomcat是绿色版，直接解压即安装

在E盘的develop目录下，将 `apache-tomcat-9.0.27-windows-x64.zip` 进行解压缩，会得到一个 `apache-tomcat-9.0.27` 的目录，Tomcat就已经安装成功。



注意，Tomcat在解压缩的时候，解压所在的目录可以任意，但最好解压到一个不包含中文和空格的目录，因为后期在部署项目的时候，如果路径有中文或者空格可能会导致程序部署失败。

打开 `apache-tomcat-9.0.27` 目录就能看到如下目录结构，每个目录中包含的内容需要认识下



bin：目录下有两类文件，一种是以 `.bat` 结尾的，是Windows系统的可执行文件，一种是以 `.sh` 结尾的，是Linux系统的可执行文件。

webapps：就是以后项目部署的目录

卸载：卸载比较简单，可以直接删除目录即可

3.2.3 启动与关闭

启动Tomcat

- 双击tomcat解压目录/bin/startup.bat文件即可启动tomcat

Work (E:) > develop > apache-tomcat-9.0.27 > bin					搜索"bin"
名称	修改日期	类型	大小		
shutdown.sh	2019/10/7 10:58	Shell Script	2 KB		
startup.bat	2019/10/7 10:57	Windows 批处理...	2 KB		
startup.sh	2019/10/7 10:58	Shell Script	2 KB		
tcnative-1.dll	2019/10/7 10:58	应用程序扩展	2,532 KB		
tomcat9.exe	2019/10/7 10:58	应用程序	122 KB		

注意：tomcat服务器启动后，黑窗口不会关闭，只要黑窗口不关闭，就证明tomcat服务器正在运行

```
Tomcat
02-Dec-2022 18:33:33.232 警告 [main] org.apache.catalina.util.SessionIdGeneratorBase.createSecureRandom Creation of SecureRandom instance f
or session ID generation using [SHA1PRNG] took [219] milliseconds.
02-Dec-2022 18:33:33.264 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [E:\dev
elop\apache-tomcat-9.0.27\webapps\docs] has finished in [652] ms
02-Dec-2022 18:33:33.265 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [E:\develop\apache-tom
cat-9.0.27\webapps\examples]
02-Dec-2022 18:33:33.774 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [E:\dev
elop\apache-tomcat-9.0.27\webapps\examples] has finished in [505] ms
02-Dec-2022 18:33:33.775 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [E:\develop\apache tom
cat-9.0.27\webapps\host-manager]
02-Dec-2022 18:33:33.843 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [E:\dev
elop\apache-tomcat-9.0.27\webapps\host-manager] has finished in [68] ms
02-Dec-2022 18:33:33.844 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [E:\develop\apache-tom
cat-9.0.27\webapps\manager]
02-Dec-2022 18:33:33.900 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [E:\dev
elop\apache-tomcat-9.0.27\webapps\manager] has finished in [57] ms
02-Dec-2022 18:33:33.901 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory 把web 应用程序部署到目录 [E:\develop\apache-tom
cat-9.0.27\webapps\ROOT]
02-Dec-2022 18:33:33.946 信息 [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [E:\dev
elop\apache tomcat 9.0.27\webapps\ROOT] has finished in [45] ms
02-Dec-2022 18:33:33.957 信息 [main] org.apache.coyote.AbstractProtocol.start 开始协议处理句柄["http-nio-8080"]
02-Dec-2022 18:33:33.976 信息 [main] org.apache.coyote.AbstractProtocol.start 开始协议处理句柄["ajp-nio-8009"]
```

Tomcat的默认端口为8080，所以在浏览器的地址栏输入：<http://127.0.0.1:8080> 即可访问tomcat服务器


127.0.0.1 也可以使用localhost代替。如：<http://localhost:8080>

localhost:8080

Home Documentation Configuration Examples Wiki Mailing Lists Find Help

Apache Tomcat/9.0.27

If you're seeing this, you've successfully installed Tomcat. Congratulations!



Recommended Reading:

[Security Considerations How-To](#)
[Manager Application How-To](#)
[Clustering/Session Replication How-To](#)

Server Status

Manager App

Host Manager

Developer Quick Start

[Tomcat Setup](#)
[First Web Application](#)

[Realms & AAA](#)
[JDBC DataSources](#)

[Examples](#)

[Servlet Specifications](#)
[Tomcat Versions](#)

Managing Tomcat

For security, access to the manager webapp is restricted. Users are defined in:
`$CATALINA_HOME/conf/tomcat-users.xml`
In Tomcat 9.0 access to the manager application is split between different users.
[Read more...](#)

[Release Notes](#)
[Changelog](#)
[Migration Guide](#)

Documentation

[Tomcat 9.0 Documentation](#)
[Tomcat 9.0 Configuration](#)
[Tomcat Wiki](#)
Find additional important configuration information in:
`$CATALINA_HOME/BUILD.txt`
Developers may be interested in:
[Tomcat 9.0 Bug Database](#)
[Tomcat 9.0 JavaDocs](#)

Getting Help

[FAQ and Mailing Lists](#)
The following mailing lists are available:
[tomcat-announce](#)
Important announcements, releases, security vulnerability notifications. (Low volume).
[tomcat-users](#)
User support and discussion
[taglibs-user](#)
User support and discussion for [Apache Taglibs](#)
[tomcat-dev](#)
Development mailing list, including commit messages

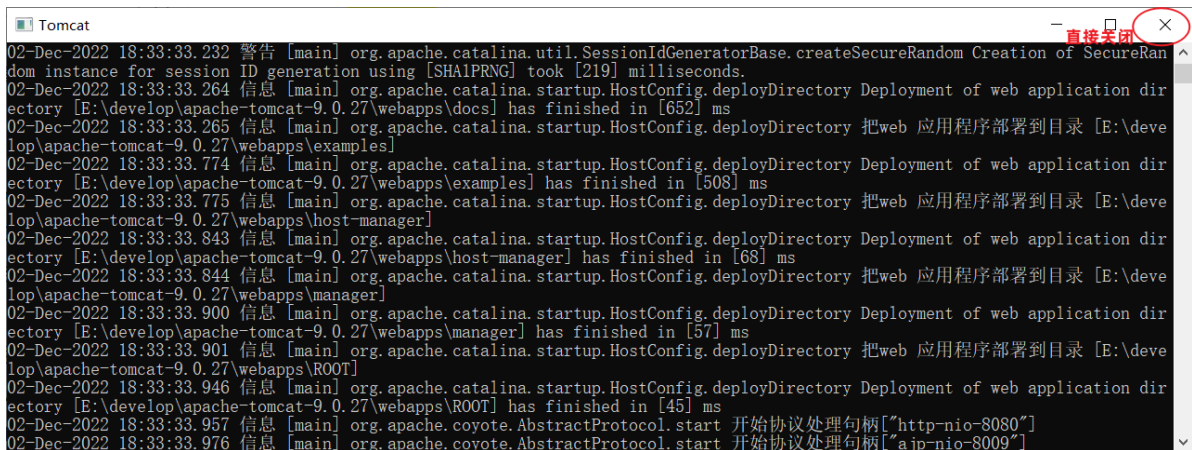
- 能看到以上图片中Apache Tomcat的内容就说明Tomcat已经启动成功

注意事项 : Tomcat启动的过程中, 遇到控制台有中文乱码时, 可以通常修改
conf/logging.properties文件解决

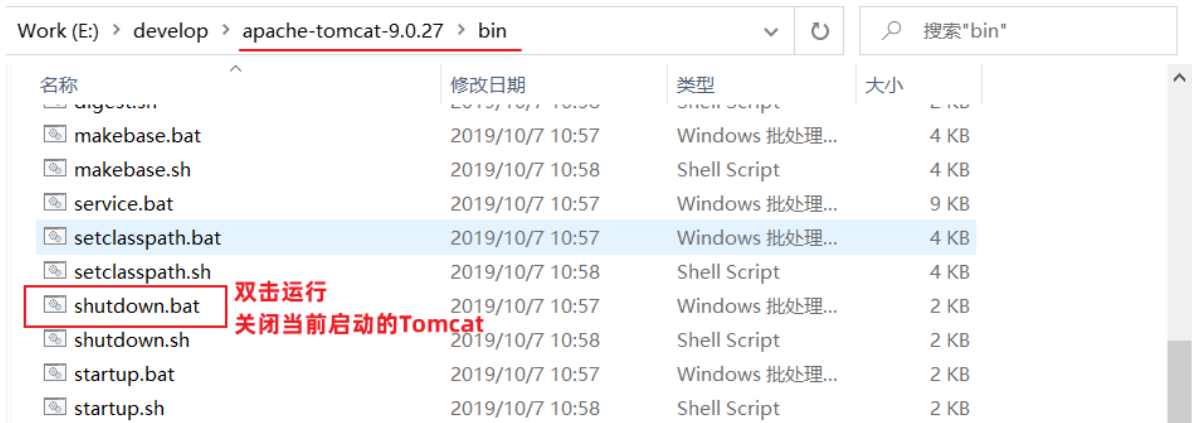
```
49 java.util.logging.ConsoleHandler.level = FINE
50 java.util.logging.ConsoleHandler.formatter = org.apache.juli.OneLineFormatter
51 java.util.logging.ConsoleHandler.encoding = GBK
```

关闭: 关闭有三种方式

1、强制关闭: 直接x掉Tomcat窗口 (不建议)



2、正常关闭: bin\shutdown.bat



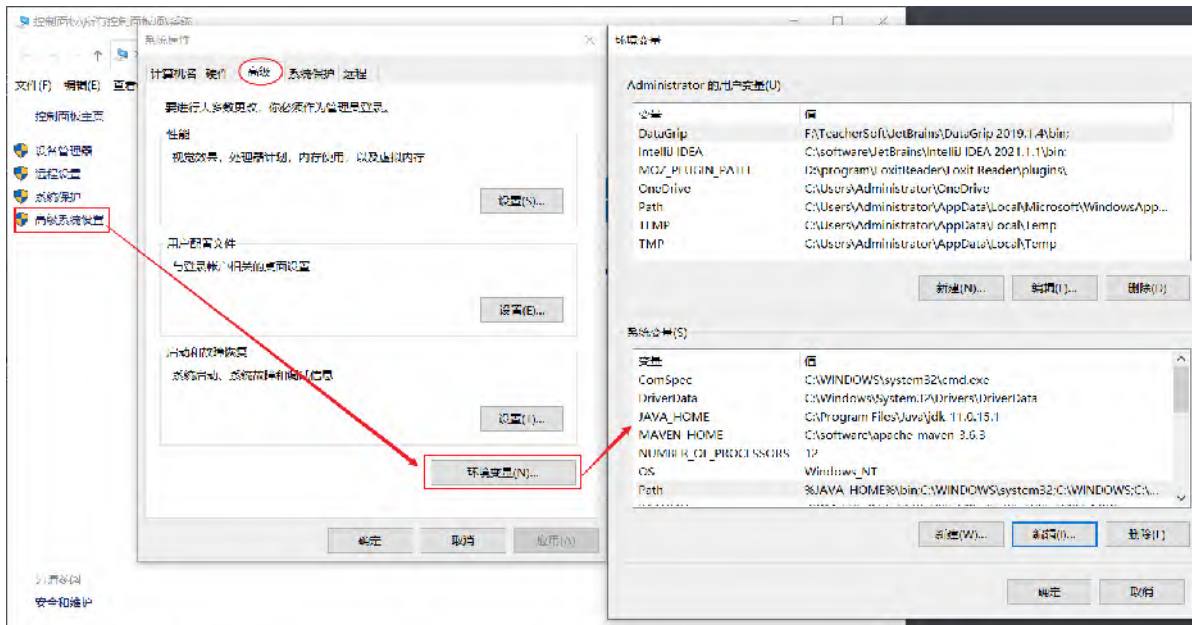
3、正常关闭: 在Tomcat启动窗口中按下 Ctrl+C

- 说明: 如果按下Ctrl+C没有反映, 可以多按几次

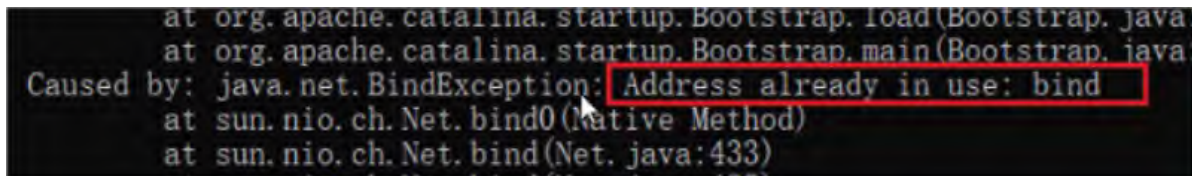
3.2.4 常见问题

问题1: Tomcat启动时, 窗口一闪而过

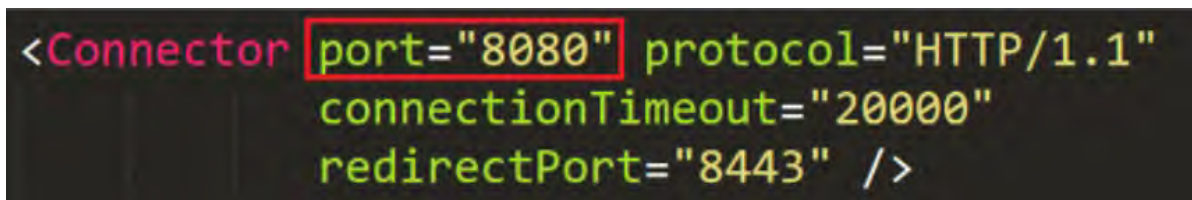
- 检查JAVA_HOME环境变量是否正确配置



问题2：端口号冲突



- 发生问题的原因：Tomcat使用的端口被占用了。
- 解决方案：换Tomcat端口号
 - 要想修改Tomcat启动的端口号，需要修改 conf/server.xml文件



注：HTTP协议默认端口号为80，如果将Tomcat端口号改为80，则将来访问Tomcat时，将不用输入端口号。

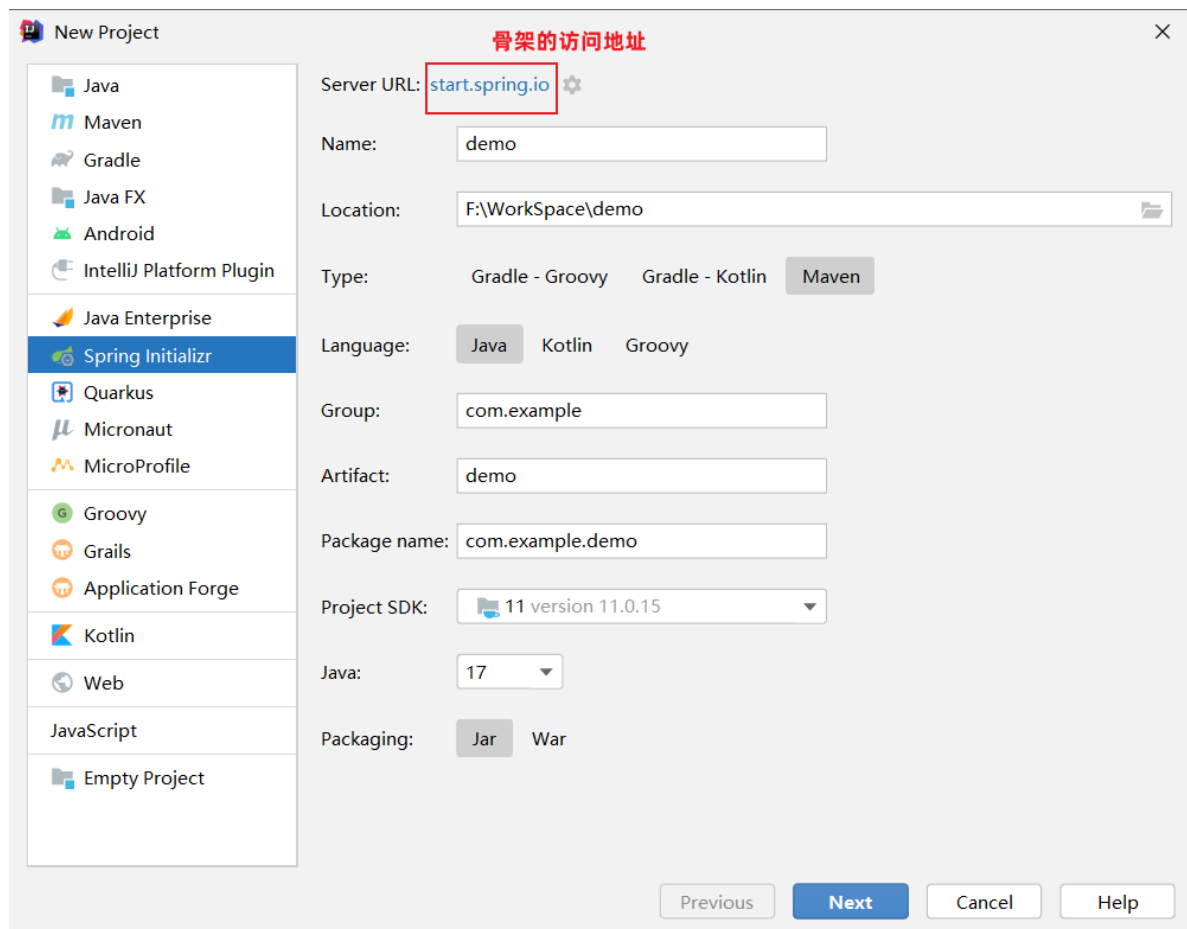
3.3 入门程序解析

关于web开发的基础知识，我们可以告一段落了。下面呢，我们在基于今天的核心技术点SpringBoot快速入门案例进行分析。

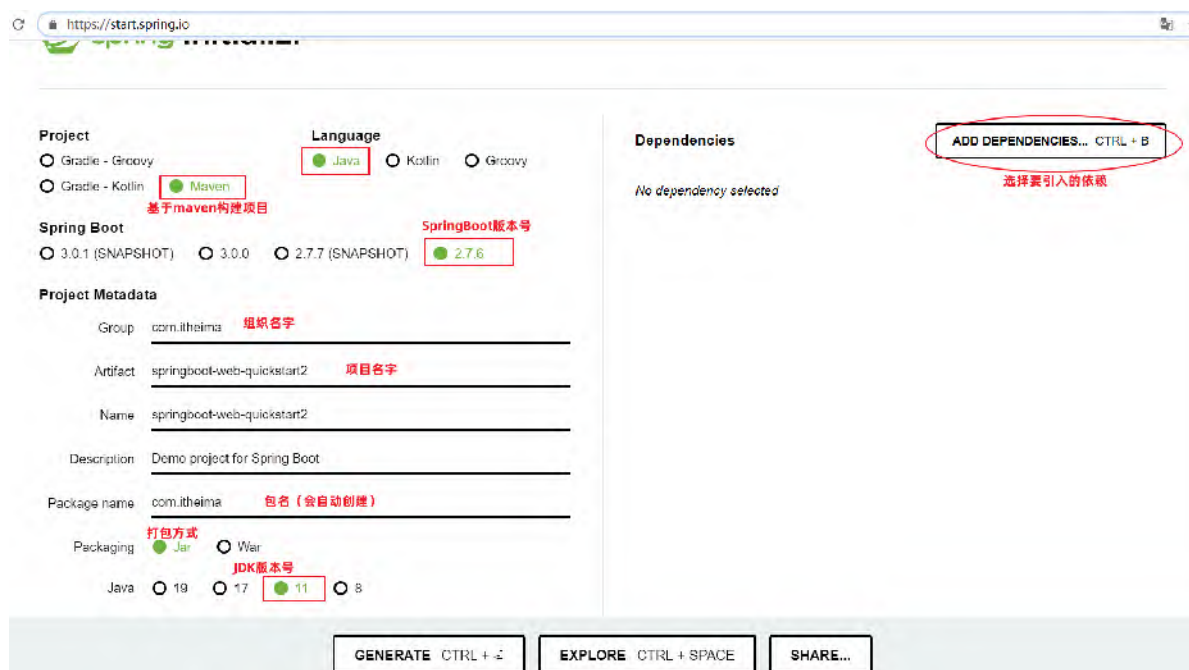
3.3.1 Spring官方骨架

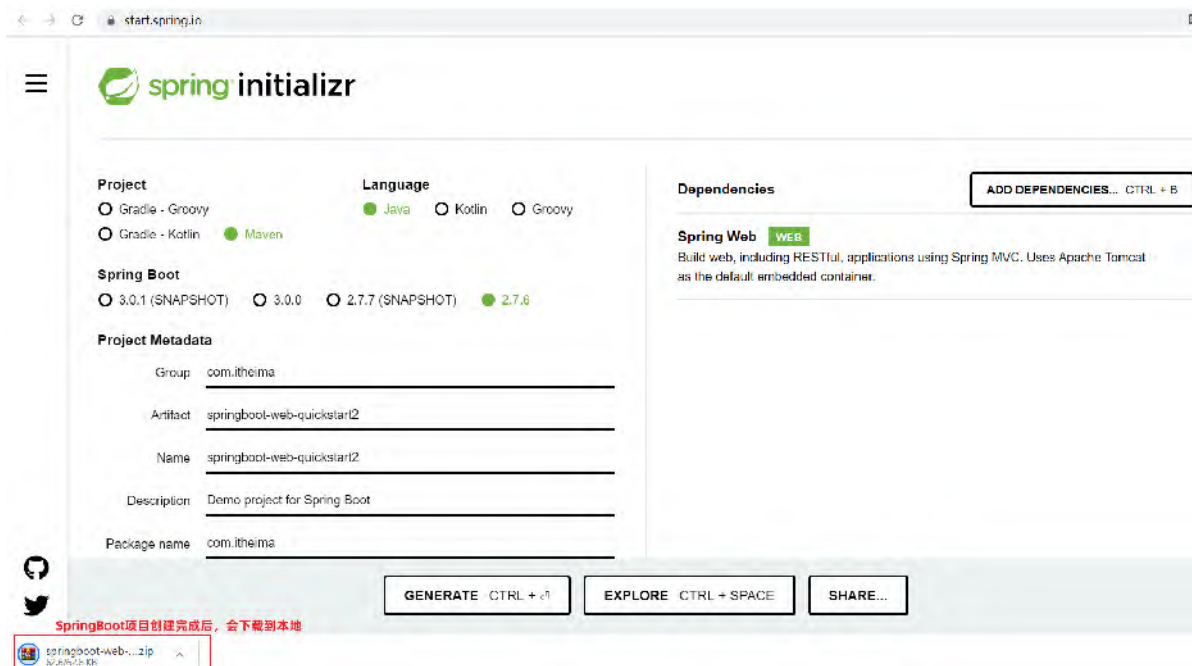
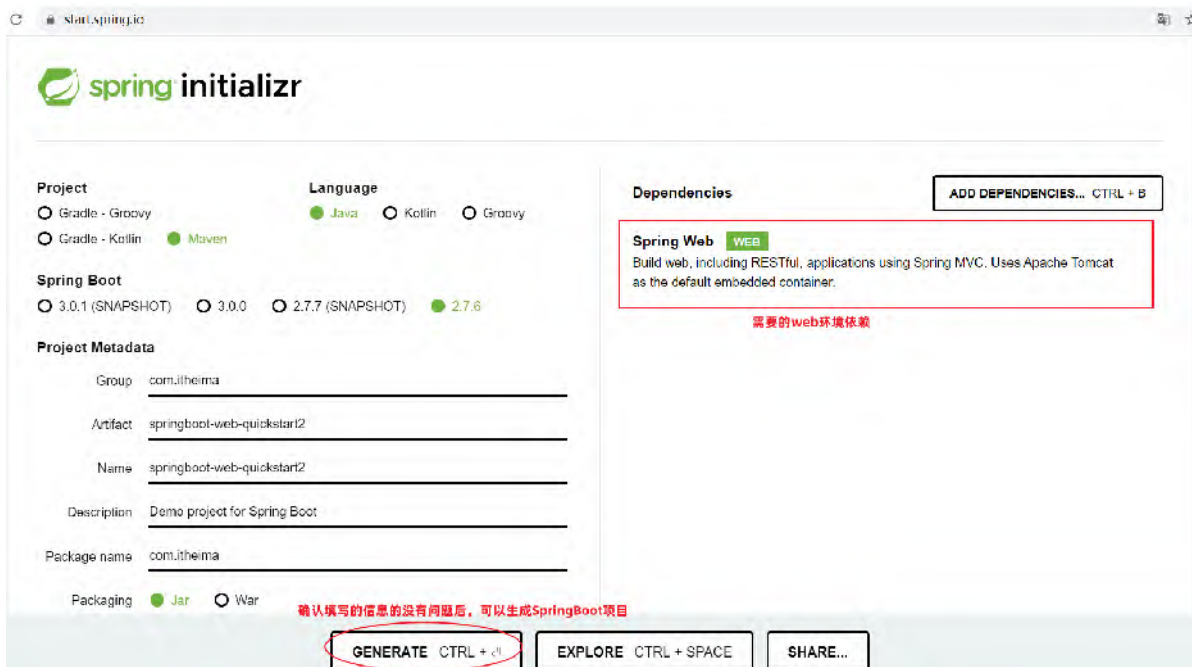
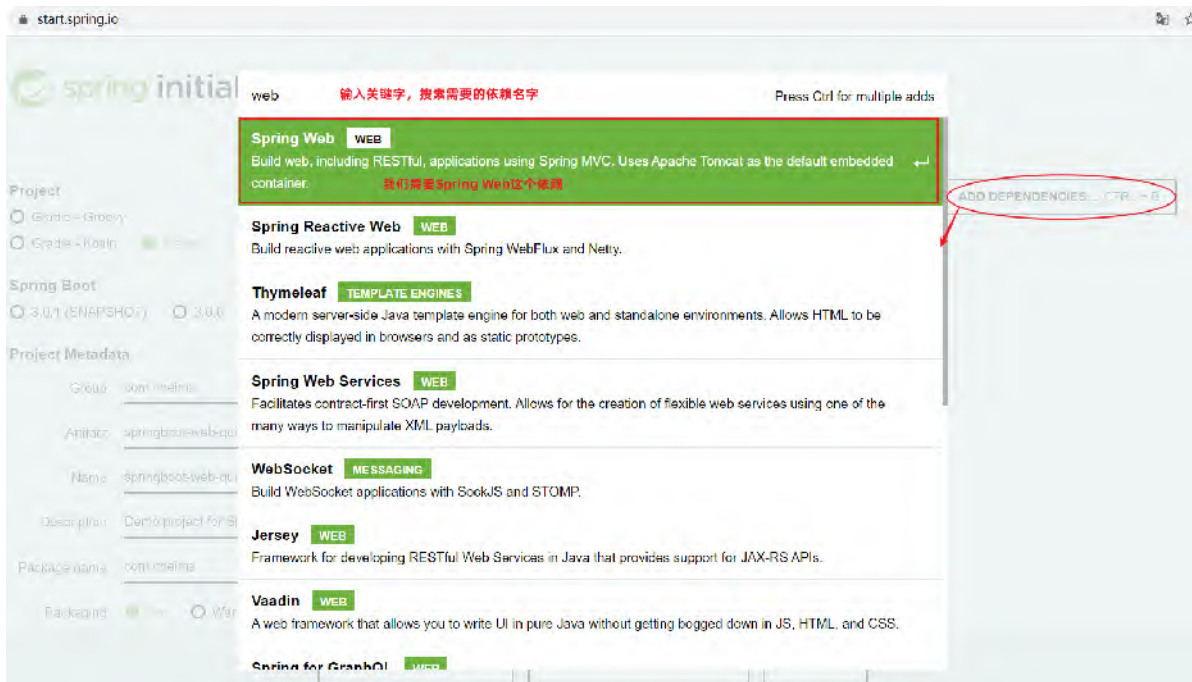
之前我们创建的SpringBoot入门案例，是基于Spring官方提供的骨架实现的。

Spring官方骨架，可以理解为Spring官方为程序员提供一个搭建项目的模板。



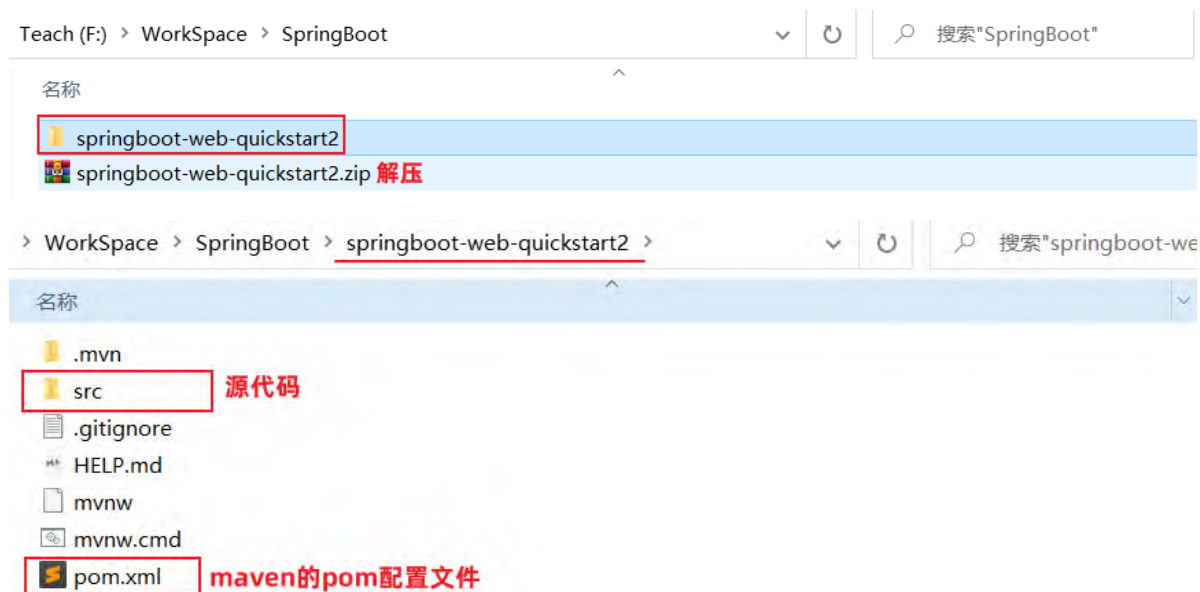
我们可以通过访问：<https://start.spring.io/>，进入到官方骨架页面



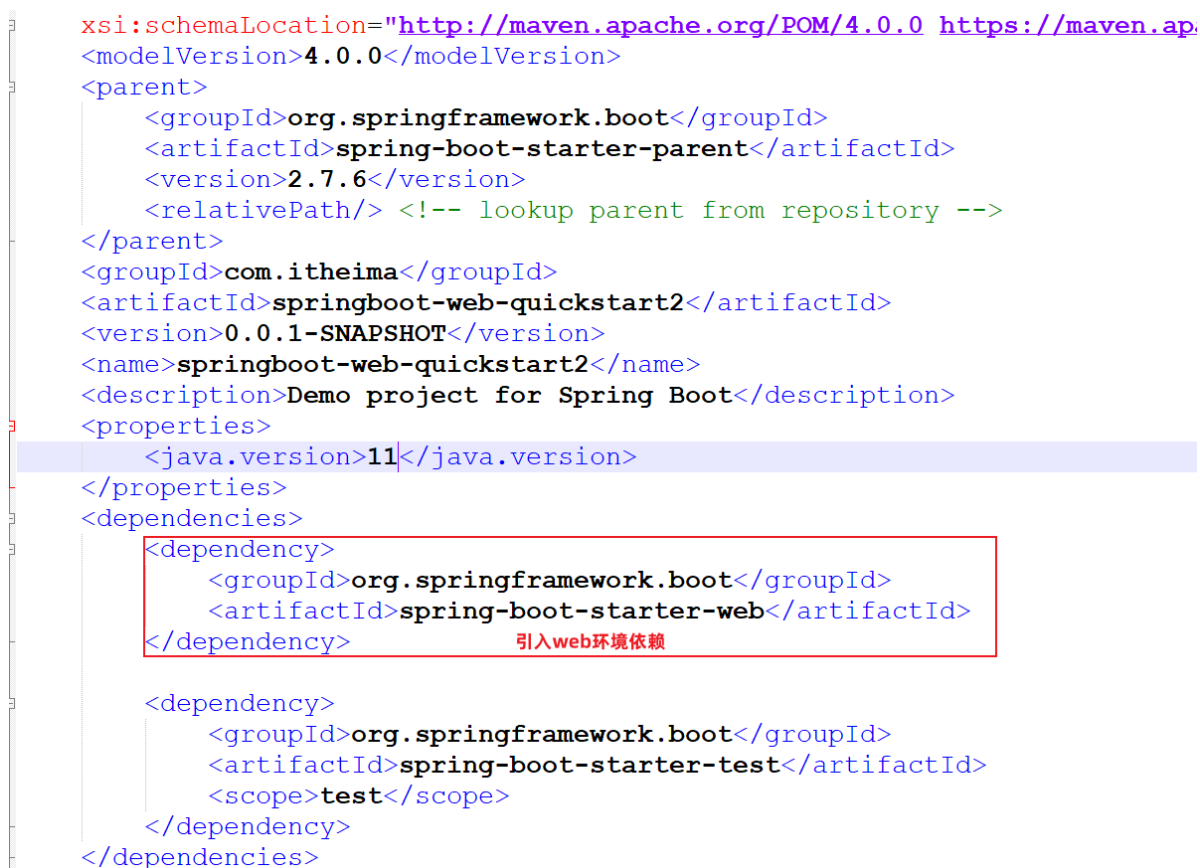


Spring官方生成的SpringBoot项目，怎么使用呢？

- 解压缩后，就会得到一个SpringBoot项目工程



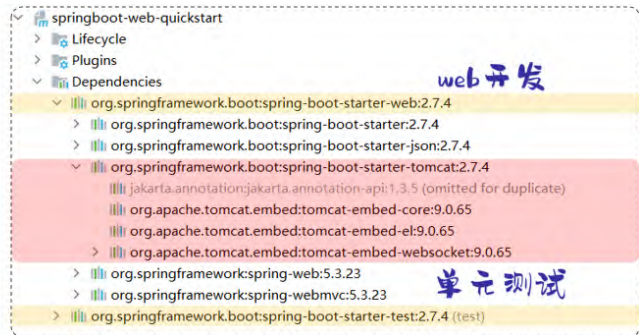
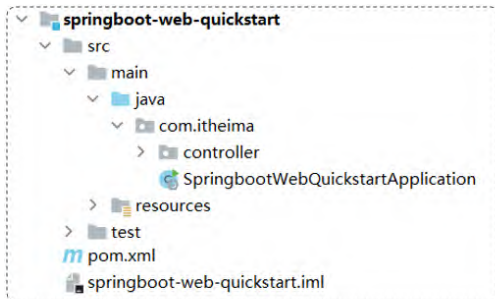
打开pom.xml文件，我们可以看到springboot项目中引入了web依赖和test依赖



结论：不论使用IDEA创建SpringBoot项目，还是直接在官方网站利用骨架生成SpringBoot项目，项目的结构和pom.xml文件中内容是相似的。

3.3.2 起步依赖

在我们之前讲解的SpringBoot快速入门案例中，同样也引用了：web依赖和test依赖



spring-boot-starter-web和spring-boot-starter-test，在SpringBoot中又称为：起步依赖

而在SpringBoot的项目中，有很多的起步依赖，他们有一个共同的特征：就是以 `spring-boot-starter-` 作为开头。在以后大家遇到spring-boot-starter-xxx这类的依赖，都为起步依赖。

起步依赖有什么特殊之处呢，这里我们以入门案例中引入的起步依赖做为讲解：

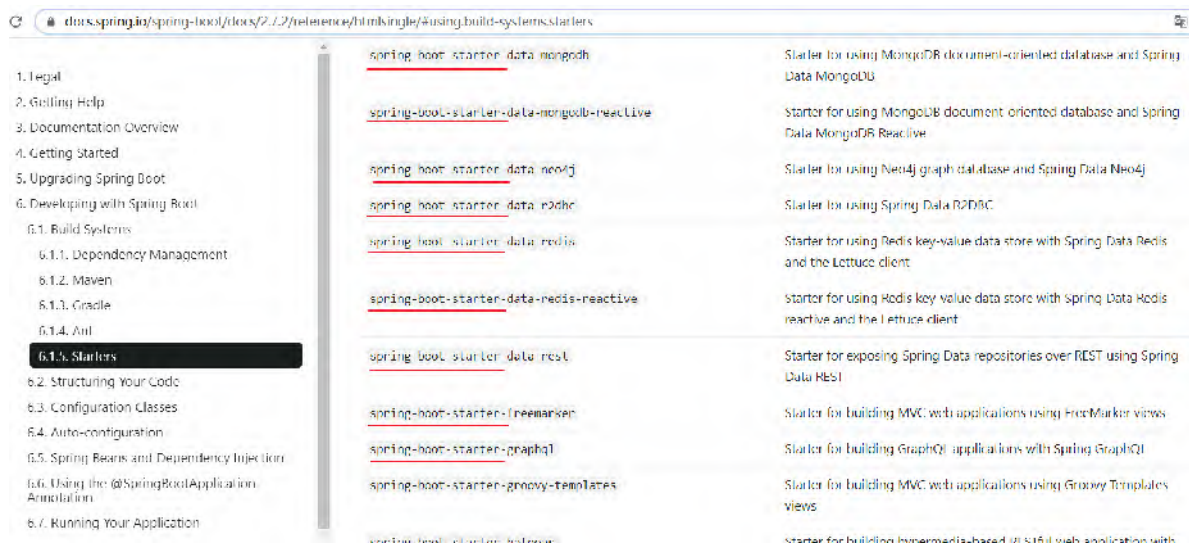
- spring-boot-starter-web：包含了web应用开发所需要的常见依赖
- spring-boot-starter-test：包含了单元测试所需要的常见依赖

spring-boot-starter-web内部把关于Web开发所有的依赖都已经导入并且指定了版本，只需引入 `spring-boot-starter-web` 依赖就可以实现Web开发的需要的功能



Spring的官方提供了很多现成的starter(起步依赖)，我们在开发相关应用时，只需要引入对应的starter即可。

官方地址：<https://docs.spring.io/spring-boot/docs/2.7.2/reference/htmlsingle/#using.build-systems.starters>



The screenshot shows the Spring Boot documentation page for starters. The left sidebar contains a navigation menu with sections 1 through 6.7, with '6.1.5. Starters' highlighted. The main content area is a table listing various starters and their descriptions.

Starter	Description
spring-boot-starter-data-mongodb	Starter for using MongoDB document-oriented database and Spring Data MongoDB
spring-boot-starter-data-mongodb-reactive	Starter for using MongoDB document-oriented database and Spring Data MongoDB Reactive
spring-boot-starter-data-neo4j	Starter for using Neo4j graph database and Spring Data Neo4j
spring-boot-starter-data-r2dbc	Starter for using Spring Data R2DBC
spring-boot-starter-data-redis	Starter for using Redis key-value data store with Spring Data Redis and the Lettuce client
spring-boot-starter-data-redis-reactive	Starter for using Redis key-value data store with Spring Data Redis reactive and the Lettuce client
spring-boot-starter-data-rest	Starter for exposing Spring Data repositories over REST using Spring Data REST
spring-boot-starter-freemarker	Starter for building MVC web applications using FreeMarker views
spring-boot-starter-graphql	Starter for building GraphQL applications with Spring GraphQL
spring-boot-starter-groovy-templates	Starter for building MVC web applications using Groovy Templates views
spring-boot-starter-h2database	Starter for building H2 database-based or standalone application with

每一个起步依赖，都用于开发一个特定的功能。

举例：当我们开发中需要使用redis数据库时，只需要在SpringBoot项目中，引入：`spring-boot-starter-redis`，即可导入redis开发所需要的依赖。

3.3.2 SpringBoot父工程

在我们之前开发的SpringBoot入门案例中，我们通过maven引入的依赖，是没有指定具体的依赖版本号的。


```
pom.xml (springboot-web-quickstart1) x
10 </parent>
11 <groupId>com.itheima</groupId>
12 <artifactId>springboot-web-quickstart1</artifactId>
13 <version>0.0.1-SNAPSHOT</version>
14 <name>springboot-web-quickstart1</name>
15 <description>Demo project for Spring Boot</description>
16 <properties>
17   <java.version>11</java.version>
18 </properties>
19 <dependencies>
20   <dependency>
21     <groupId>org.springframework.boot</groupId>
22     <artifactId>spring-boot-starter-web</artifactId>
23   </dependency> 没有依赖版本号
24
25   <dependency>
26     <groupId>org.springframework.boot</groupId>
27     <artifactId>spring-boot-starter-test</artifactId>
28     <scope>test</scope> 没有依赖版本号
29   </dependency>
30 </dependencies>
```

为什么没有指定版本号，可以正常使用呢？

- 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。

```
pom.xml (springboot-web-quickstart1) x
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/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion> 父工程
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.7.6</version> 父工程指定了版本号后，就会自动
9     <relativePath/> <!-- lookup parent from repository --> 引入和所指定版本对应的起步依赖
10  </parent>
11  <groupId>com.itheima</groupId>
12  <artifactId>springboot-web-quickstart1</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>springboot-web-quickstart1</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17    <java.version>11</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-web</artifactId>
23    </dependency>
```

创建的SpringBoot工程，会继承SpringBoot父工程

