## Java Spring Boot学习笔记

1. 创建工程的三种方式：
2. 去spring官网，quickstart，一系列设置后下载得到一个zip，解压用idea打开即可
3. 使用IDEA（专业版）创建，选择Spring Initializer……. 跟官网上没什么区别
4. 使用IDEA创建maven项目，但这种方法创建的不是springboot项目，需要进一步配置
5. 项目结构
6. POM文件

* 继承



* 依赖（所有SpringBoot依赖都是以spring-boot-starter开头的）



* 插件

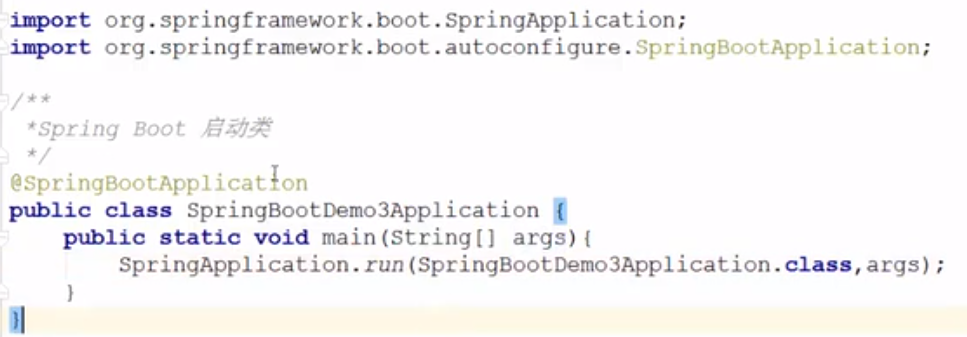


1. 启动类

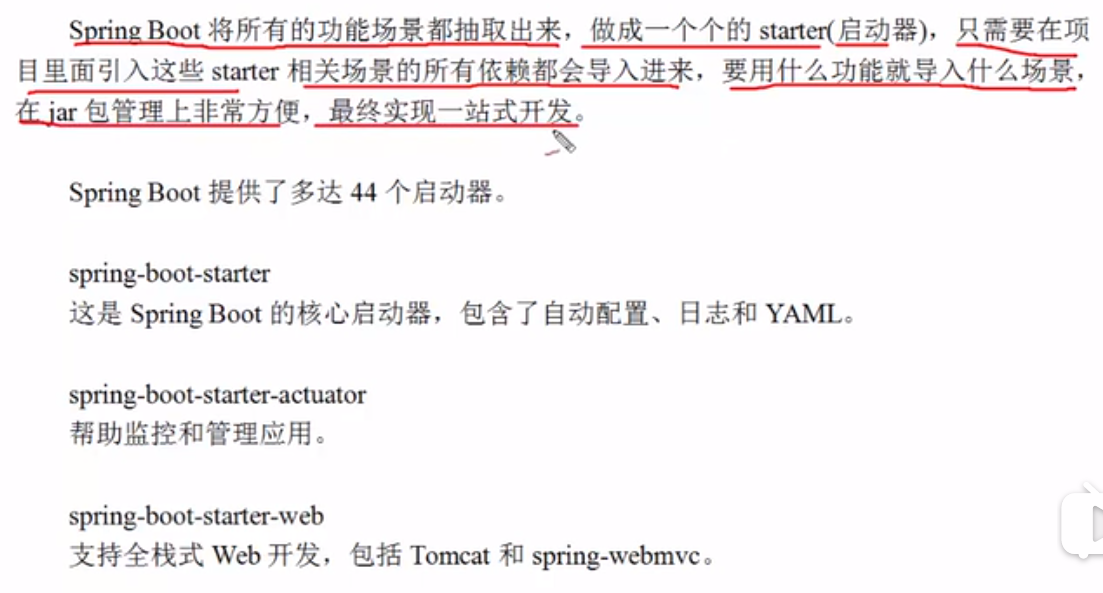
Springboot内嵌了服务器容器（位于spring-boot-starter-web依赖中），因此启动时不再是启动服务器容器的方法启动项目，而是启动springboot框架，实现方法就是启动类。



* 启动类与启动器（starter）的区别
* 启动类表示项目的启动入口，本质上是一个class
* 启动器表示jar包的 坐标
* 创建启动类（通过官网或spring脚手架创建的话会自动补全这个启动类，而maven创建法就需要自己写）



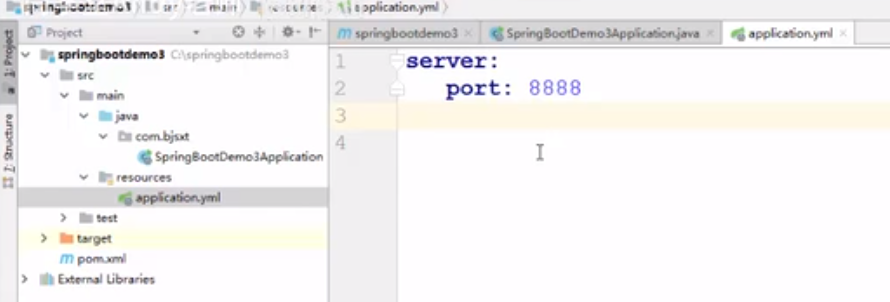
1. 启动器



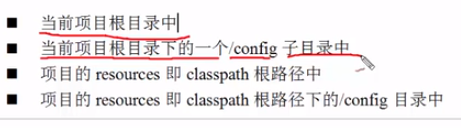
1. 配置文件

提供了名为application的全局配置文件，在main/resources/文件夹下，格式为properties或yml（没有的话可以自行创建），可用配置tomcat监听端口等信息。

常用yml格式：大小写敏感，用缩进代表层级关系。



* 配置文件存放位置



* 配置文件加载顺序

优先读取properties格式的文件

同一个配置属性在多个配置文件中都读取到了，默认使用第一个读取到的。

* 不同位置的配置文件读取的优先级：

按照上图四个位置，排序：2 1 4 3（即第二个优先级最高，第三个优先级最低）

1. 配置文件中的占位符

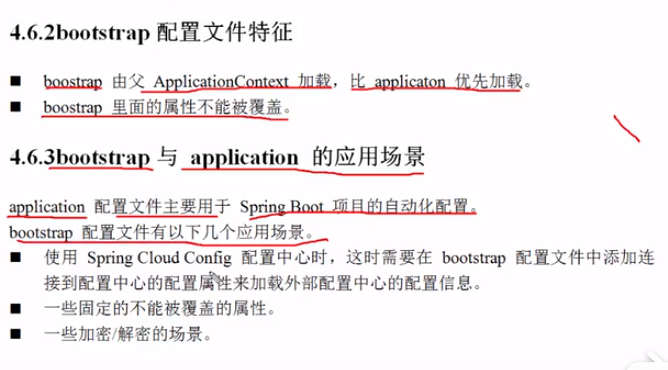
语法 ${ }



1. Bootstrap配置文件
2. Springboot提供两种配置文件：application和bootstrap，分别服务application上下文对象和bootstrap上下文对象。
3. 介绍：主要用于外部属性、外部资源的配置。



1. 应用场景



1. Bootstrap配置文件与application配置文件除了名字之外，格式和路径都相同。
2. 面试：谈谈对springboot的理解：

两个方面：自动装配和run方法做了什么（四步）

Pom.xml

* Spring-boot-dependencies : 核心依赖在父工程中
* 我们在写或引入Springboot依赖的时候，不需要指定版本，是因为有这些版本仓库。

启动器：

* 说白了，就是springboot的启动场景
* 比如spring-boot-web，他就会帮我们自动导入web环境所有的依赖。
* Springspringoboot会将所有的功能场景变成一个个的的启动器。
* 。要使用什么功能只需要找到对应的启动器就可以了。

主程序：

* 注解：标注这个类是springboot的应用
* SpringApplication.run方法将应用启动，使用反射加载当前启动类的对象

注解：

* SpringBootConfiguration springboot的配置
* Configuration spring配置类
* Component：说明这也是一个spring的组件
* EnableAutoConfiguration：自动配置
* AutoConfigurationPackage：自动配置包
* @Import( AutoConfigurationPackages.Register.class):自动配置包注册

1. 配置文件

配置文件名称固定：application.properties 或 application.yml

前者语法结构是：key=value 后者语法结构是：key:空格value

配置文件的作用：修改springboot自动配置的默认值，因为springboot在底层都给我们配置好了。

**补充：Springboot给某个类的属性赋值的几种方式**：<https://blog.csdn.net/qq_44906471/article/details/100518484>

**结论：**

* **配置yml和properties都可以获取到值，推荐yml**
* **如果只需要获取配置文件中的某个值，使用@value**
* **如果写了一个JavaBean来和配置文件进行映射，直接使用@configurationProperties**

1. Springboot核心注解









**@Component** 用在某个类上，表示该类是一个组件（一般是Bean规范的）

然后可以在测试类中或工程中其他位置的某个类使用@Autowired（加在类的属性上，比如private Person person；就可以加上一个@Autowired来导入Person这个Component）来导入这个Component。

@Controller

@Value

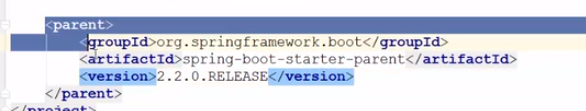
@Servlet

@Service

@Bean

@MapperScan

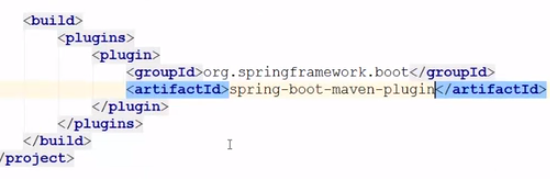
1. 编写helloworld
2. 创建项目
3. 创建maven项目
4. 变成springboot项目



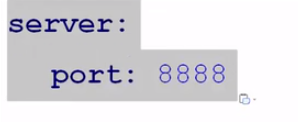
1. 添加依赖



1. 添加打包插件

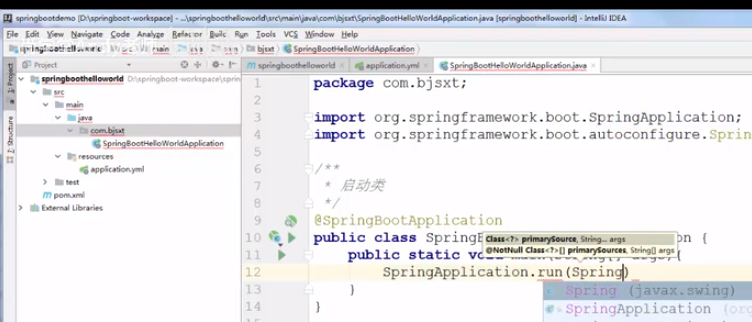


1. 修改pom文件（如上）
2. 修改Tomcat端口
3. 默认是8080
4. 在resources目录下创建配置文件application.yml

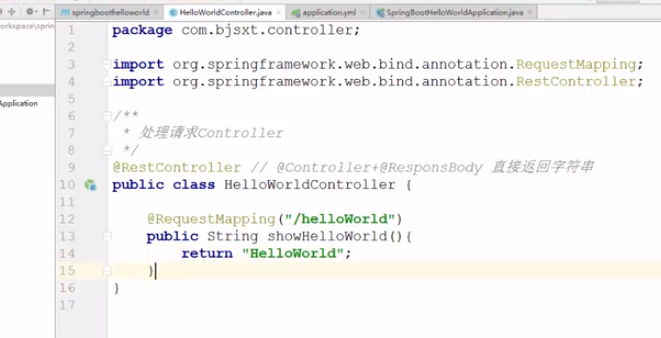


1. 创建启动类

Main/java下创建一个自定义的包，包下创建启动类（参照上文）



1. 创建Controller



1. 找到启动类去启动main方法即可。
2. 一般在com.xxx.下需要建立以下几个文件夹：
3. Controller
4. Pojo
5. Dao（也就是mapper接口文件）
6. Service
7. Springboot在controller中的常用注解



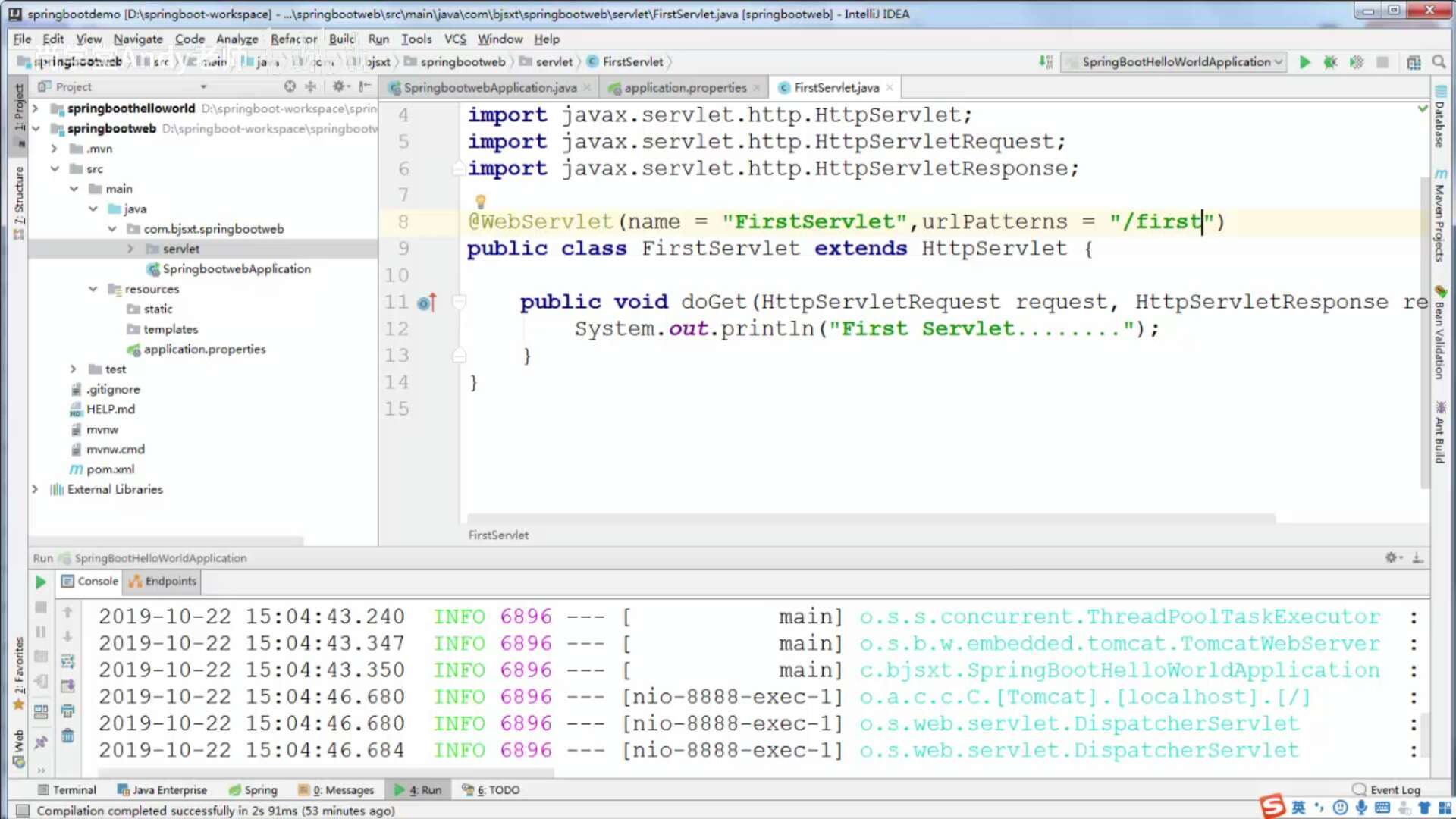


1. 整合web层技术

上文已经提到了整合springMVC中的controller

1. 整合servlet
2. 方式一

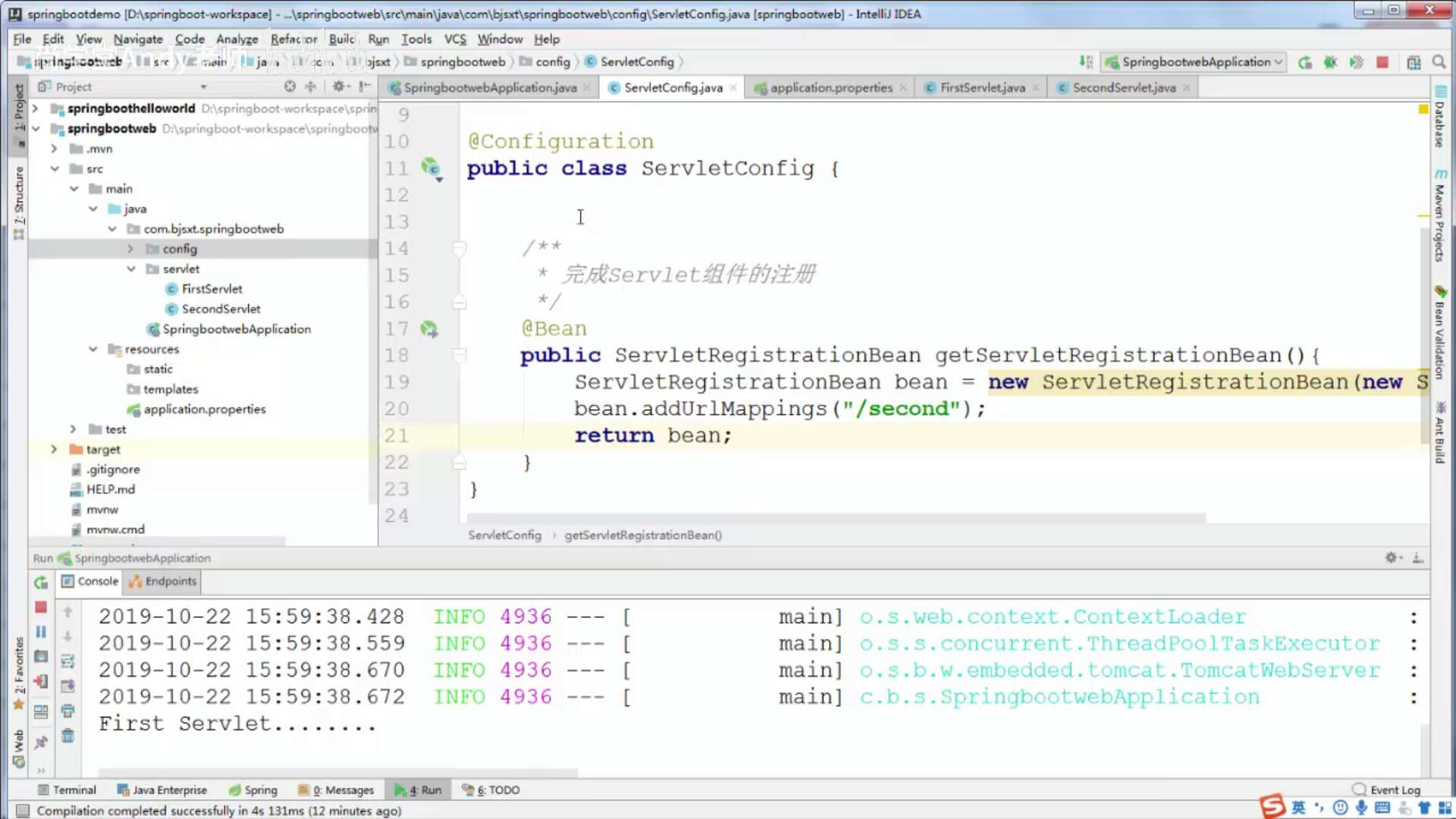
* 概述：通过注解完成servlet组件的注册
* 创建文件夹servlet，存放servlet类
* 创建servlet类，继承HttpServlet类
* 重写doGet、doPost方法



* 修改启动类：在启动类上多加一个注解@ServletComponentScan，在启动时会自动扫描@WebServlet注解，并将该类实例化

1. 方式二：不方便，

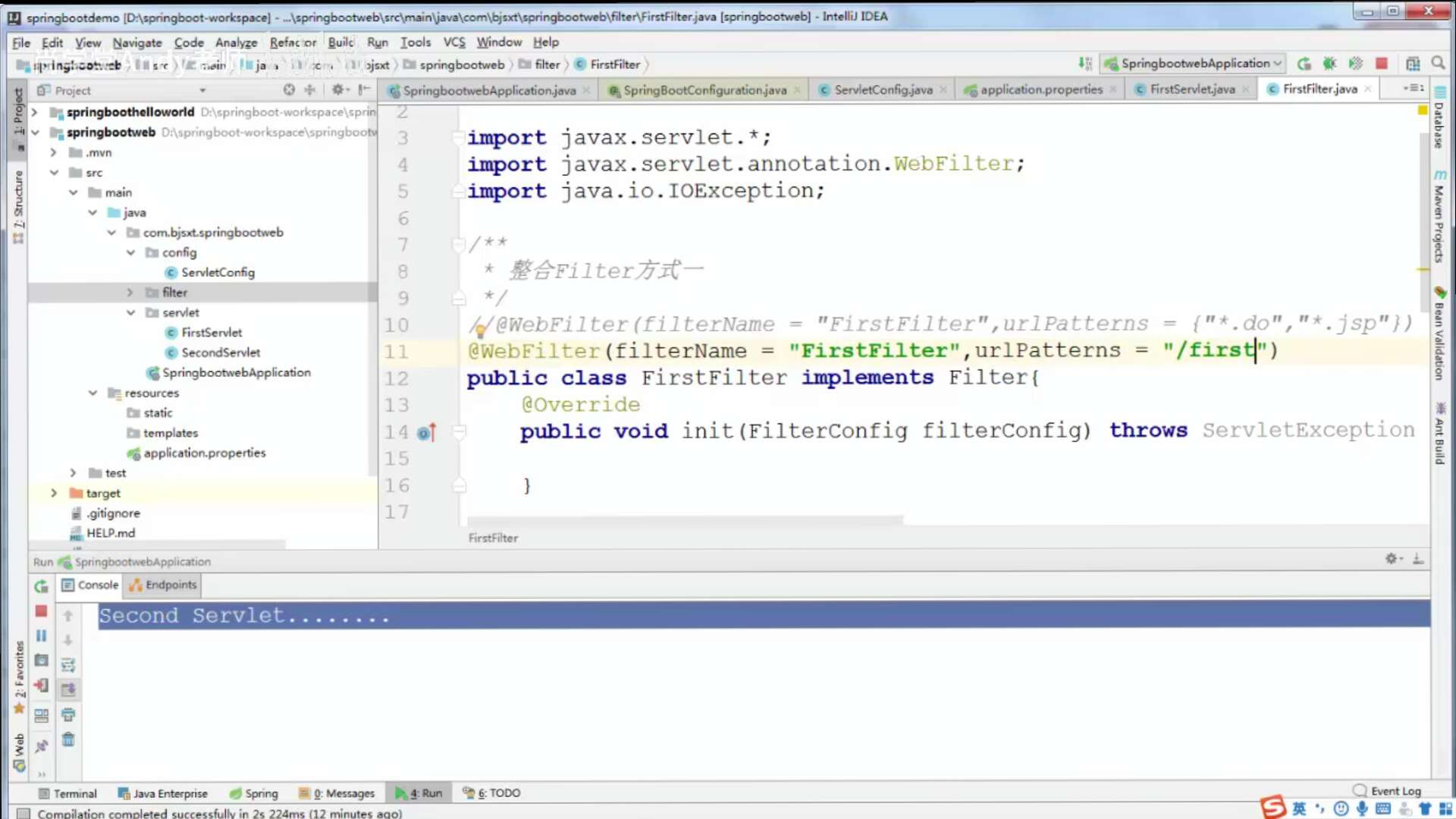
* 概述：通过方法完成servlet组件的注册，而不是通过注解
* 与方式一一样创建servlet类，但不必加WebServlet注解
* 然后照下图写就完事了。



1. 整合filter
2. 方式一：注解扫描完成filter组件注册

* 创建Filter
* 与创建servlet类似
* 创建一个包存放filter类，创建filter类实现Filter接口，加注解@WebFilter
* 重写init, doFilter, destroy三个方法（主要是doFilter方法）
* 修改启动类

只要加了@ServletComponentScan注解，还可以扫描filter，除了可以扫描servlet。

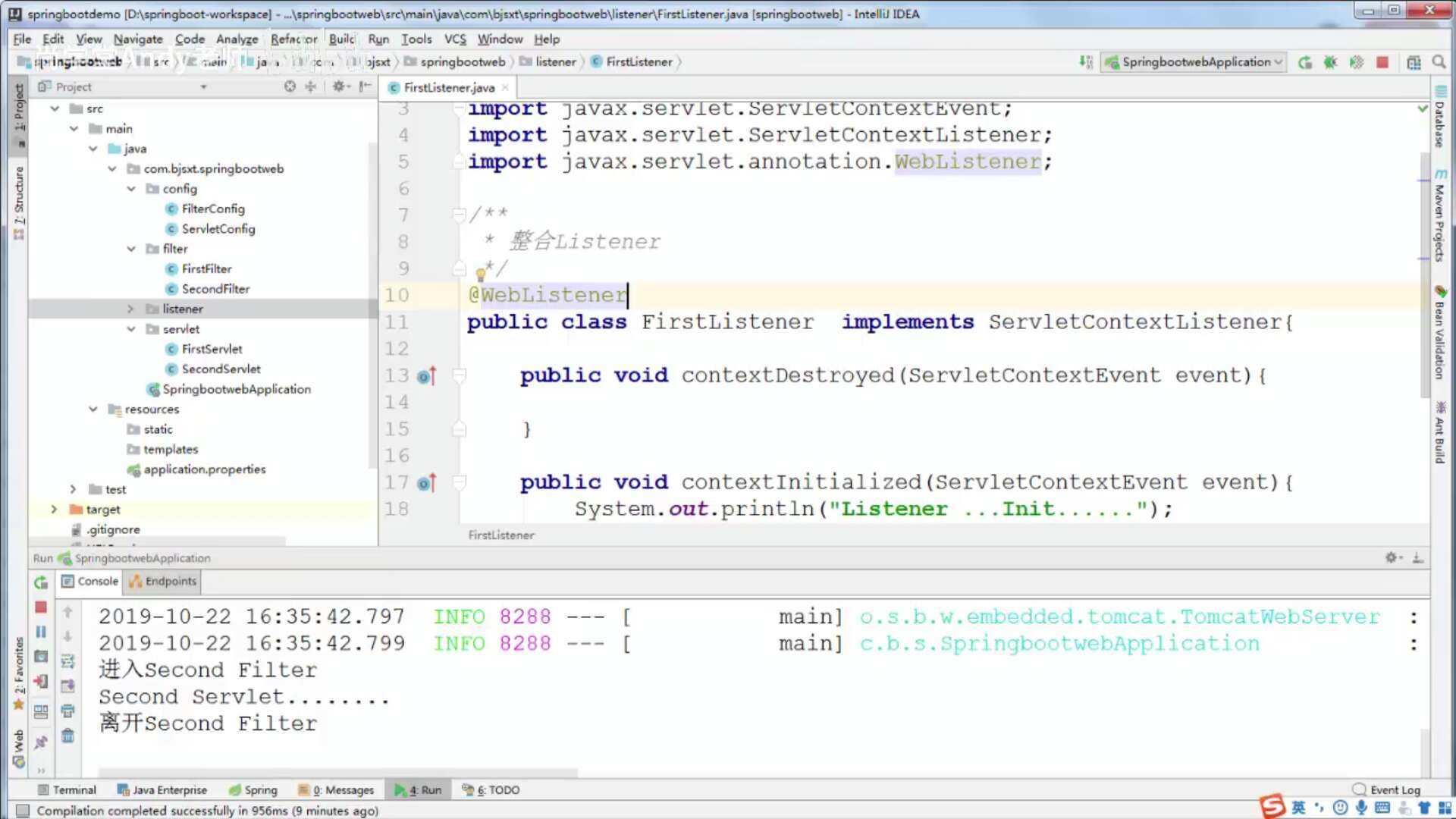




1. 方式二：通过方法完成filter组件注册
2. 整合listener
3. 方式一：注解扫描

Servlet中有八个监听器（listener）

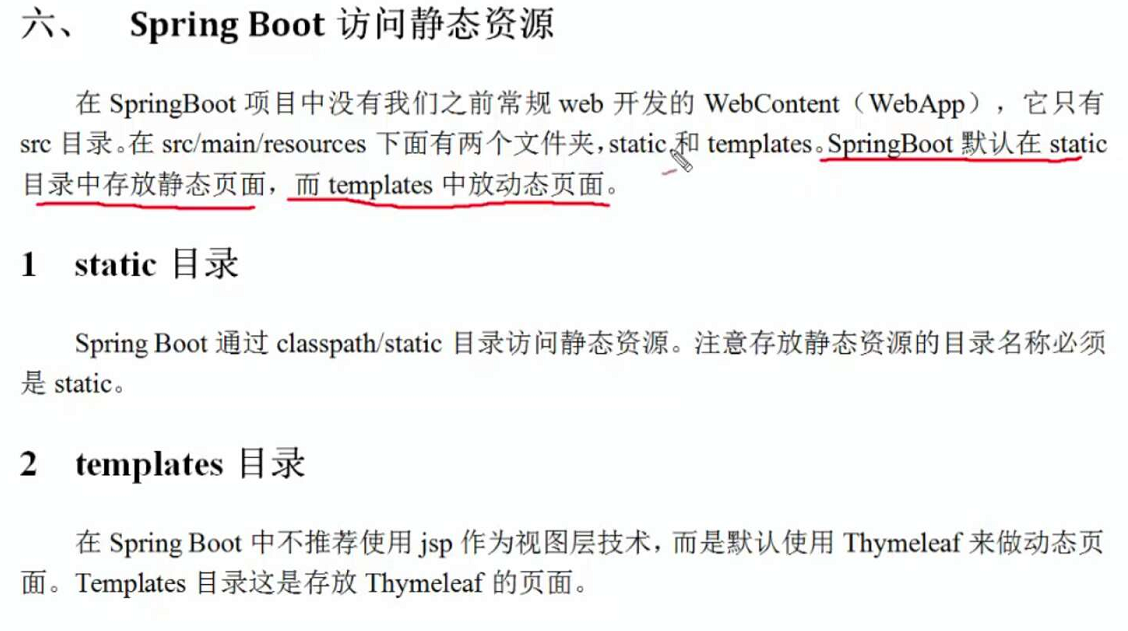
* 创建包存放listener类
* 创建listener类，实现接口（具体实现什么接口由这个监听器的功能决定）
* 重写接口方法
* 加注解@WebListener

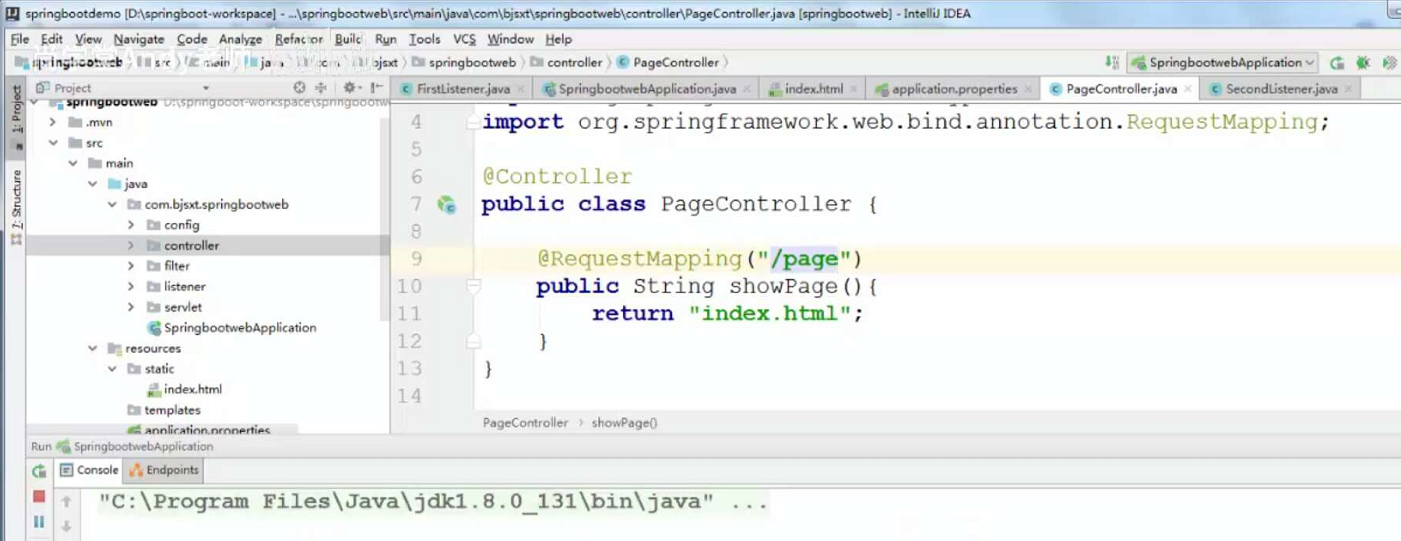


* 修改启动类：在启动类上加注解@ServletComponentScan （也就是说这个注解可以同时完成对servlet，filter，listener（只要加了注解）的扫描）

1. 方式二略
2. Spring boot 访问静态资源

Static目录和templates目录，static目录放静态资源，（图片，html，css，js等）





* Static是默认位置，访问里面的资源时不必加路径前缀，直接写资源名字即可。（如上图返回页面 index.html）

1. 静态资源存放其他位置
2. Spring boot默认访问静态资源的位置：（默认会逐一查找这几个目录，顺序就是如下所示）
3. Classpath：/META-INF/resources/
4. Calsspath:/resources/
5. Classpath:/static/ 上文所示就是这个目录
6. Classpath:/public/
7. 注意：Classpath指的是src/main/resources/目录
8. 注意：上述四个目录是默认的，当访问其中的资源时不用指明资源的路径。
9. 自定义静态资源存放位置：修改application.properties 或.yml 配置文件即可

在配置文件中添加：spring.resources.static-locations=classpath:/whatever/

**注意**：一旦自定义配置，之前的关于静态资源位置的默认配置（上述4个默认位置）就失效了

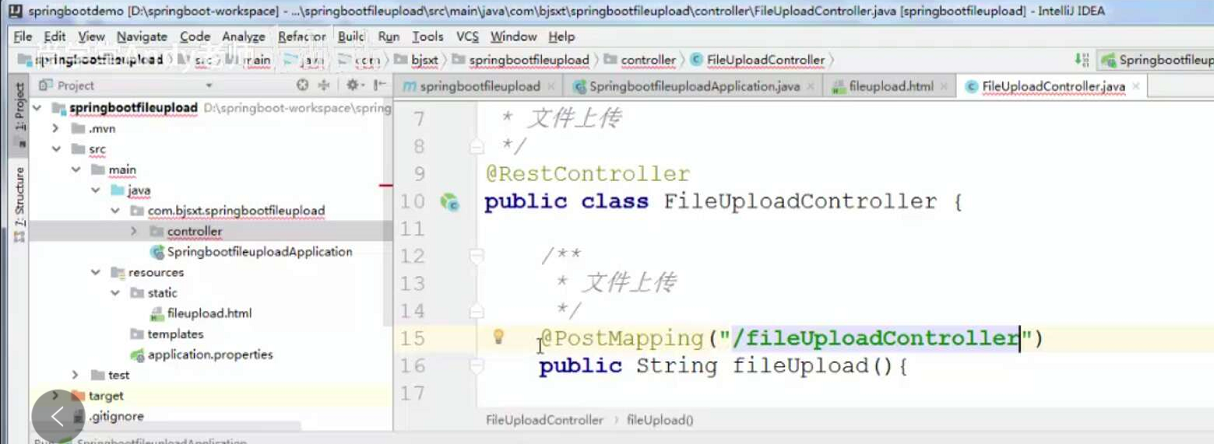
1. 文件上传
2. 编写上传页面

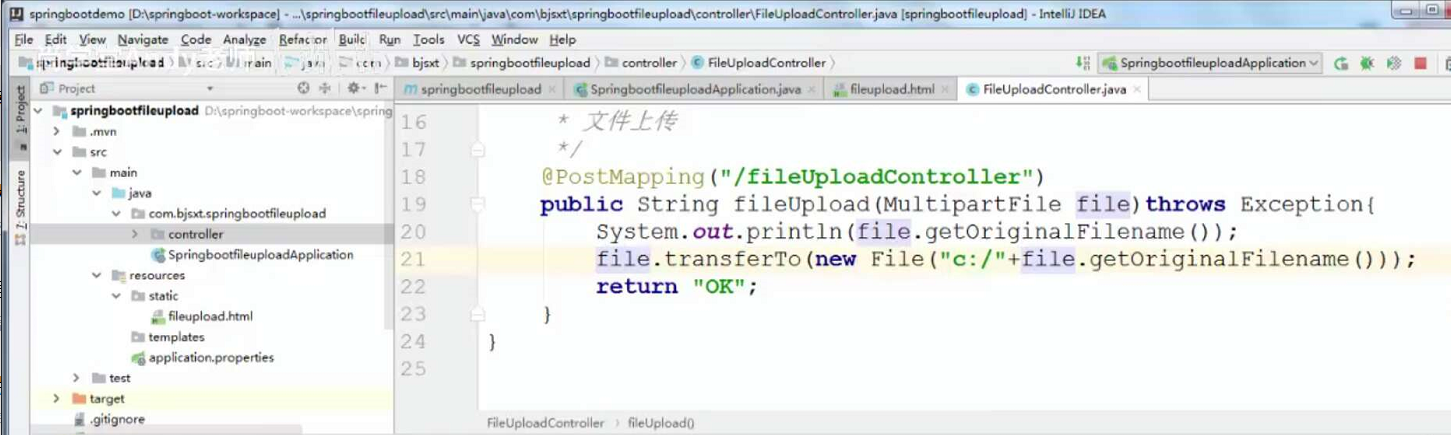
* 在main/resources/static中创建html页面



1. 编写controller

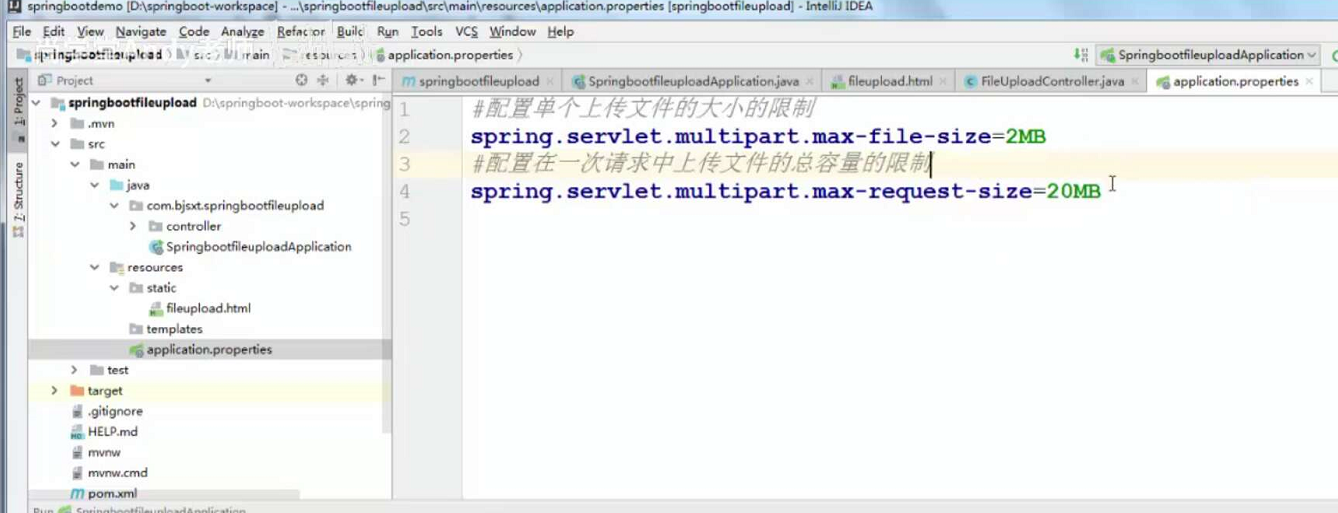
创建包controller，写一个Controller类，加注解





1. 文件大小限制配置（默认最大1Mb）

在application配置文件中修改：spring.servlet.multipart.max-file-size=1MB (单个文件上传大小限制)



1. Spring boot 整合视图层技术
2. 整合JSP
3. 修改pom文件，添加JSP引擎和JSTL标签库

Springboot内嵌的tomcat容器没有JSP引擎，所以需要添加



1. 创建webapp目录

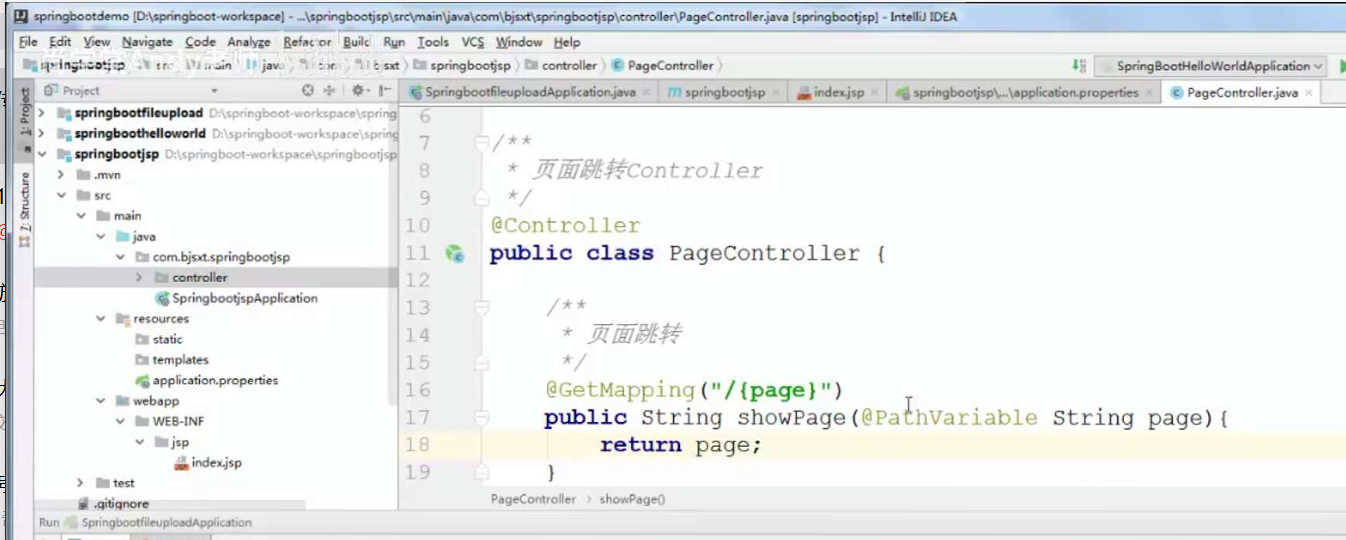
* 创建在main目录下，与java和resources同级。
* webapp下创建WEB-INF目录，在WEB-INF下面建立jsp目录存放jsp
* 此时，右击webapp还不能创建jsp文件，要将webapp目录标记为支持web技术的目录就可以了。

1. 标记为web目录
2. 创建JSP
3. 修改application配置文件，配置视图解析器

spring.mvc.view.prefix=/WEB-INF/jsp/

spring.mvc.view.suffix=.jsp

1. 创建controller实现页面跳转



1. 整合Freemarker

Freemarker与jsp，Thymeleaf一样都是视图层技术，区别在于jsp是在服务器端进行页面和数据的渲染，而其他二者都是在客户端进行渲染，所以jsp的效率更低。

1. 整合Thymeleaf

## Java Spring框架学习笔记

1. 共四天：
2. Spring框架概述及spring中基于XML的IOC配置
3. Spring中的基于注解的IOC和ioc的案例
4. Spring中的AOP和基于XML及注解的AOP配置
5. Spring中的JdbcTemplate以及spring事务控制
6. Spring概述----------------------------------------
7. 是什么

全栈轻量级框架

1. 两大核心

IoC（反转控制）和AOP（面向切面编程）

1. 发展历程和优势

方便解耦 简化开发

AOP编程的支持

声明式事务的支持（通过声明方式进行事务管理）

方便程序测试

方便集成各种优秀框架

降低JavaEE API的使用难度

Java源码的经典学习范例

1. 体系结构
2. 程序的 耦合 解耦
3. 曾经案例中问题
4. 工厂模式解耦
5. IOC概念和spring中的IOC
6. Spring中基于XML的IOC环境搭建
7. 依赖注入（Dependently Injection）
8. 作业：