# Spring boot基础（版本是1.5.10正式版）

## 官网

http://spring.io/projects/spring-boot

springboot发布时间在2014年

## Spring boot API

https://docs.spring.io/spring-boot/docs/current/api/

## springboot优秀文章的网址

http://www.ityouknow.com/spring-boot

https://github.com/xuwujing/springBoot-study

http://www.ityouknow.com/springboot/2015/12/30/springboot-collect.html

## 概述

1、随着动态语言的流行（Ruby、Groovy、Scala、Node.js），Java的开发显得格外的笨重，繁多的配置、低下的开发效率、复杂的部署流程以及第三方技术集成难度大。

2、Spring boot使用“习惯优于配置”（项目中存在大量的配置，此外还内置一个习惯性的配置，让你无须手动进行配置）的理念让你的项目快速运行起来，使用Spring boot很容易创建一个独立运行（运行jar，内嵌servlet容器）、准生产级别的基于Spring框架的项目，使用Spring boot你可以不用或者只需要很少的Spring配置。

3、Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程，该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。

4、这里没有配置，没有web.xml，没有构建说明，甚至没有应用服务器，但这就是整个应用程序了。SpringBoot会搞定执行应用程序所需的各种后勤工作，你只要搞定应用程序的代码就好。

## 优点

（1）纯JAVA的配置方式，简单方便。

（2）配合各种starter使用，基本上可以做到自动化配置。

（3） 配合Maven或Gradle等构建工具打成jar包后，java -jar 进行部署运行。

## 缺点

（1）集成度较高，使用过程中不太容易了解底层。

（2）从原来的xml配置方式转换到JAVA配置方式变化有点大，可能一下子转换不过来。

（3）坑有些多, 文档略少, 特别是一些JavaConfig相关的东西。

## 特性

（1）使用 Spring 项目引导页面可以在几秒构建一个项目。

（2）方便对外输出各种形式的服务，如 REST API、WebSocket、Web、Streaming、Tasks。

（3）非常简洁的安全策略集成。

（4）支持关系型数据库和非关系型数据库。

（5）支持运行期内嵌容器，如Tomcat、Jetty。

（6）强大的开发包、支持热启动。

（7）自动管理依赖。

（8）自带应用监控。

（9）支持各种IDE，如 IntelliJ IDEA 、NetBeans

## 核心功能

### （1）独立运行Spring项目

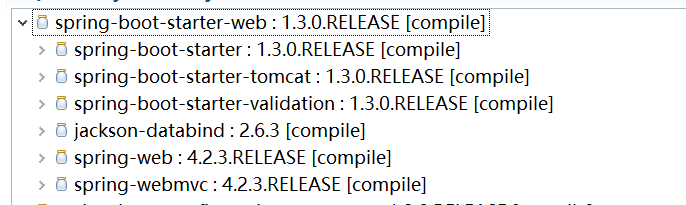
Spring boot可以以jar包的形式独立运行，运行一个Spring boot项目只需要通过java –jar xx.jar来运行。

### （2）内嵌Servlet容器

Spring boot可选择内嵌Tomcat、Jetty或者Undertow，这样我们无须以war包形式部署项目。

### （3）提供starter简化maven配置

Springboot提供了一系列的starter pom来简化maven的依赖加载。例如当你使用了spring-boot-starter-web时会自动的加载其他的依赖包，如图：



### （4）自动配置Spring

Spring boot会根据在类路径中的jar包、类，为jar包里的类自动配置Bean，这样会极大的减少我们要使用的配置。当然如果在实际开发中我们需要自动配置Bean，而Spring boot没有提供支持，则可以自定义自动配置。

### （5）准生产的应用监控

Spring boot提供基于http、ssh、telnet对运行时的项目进行监控。

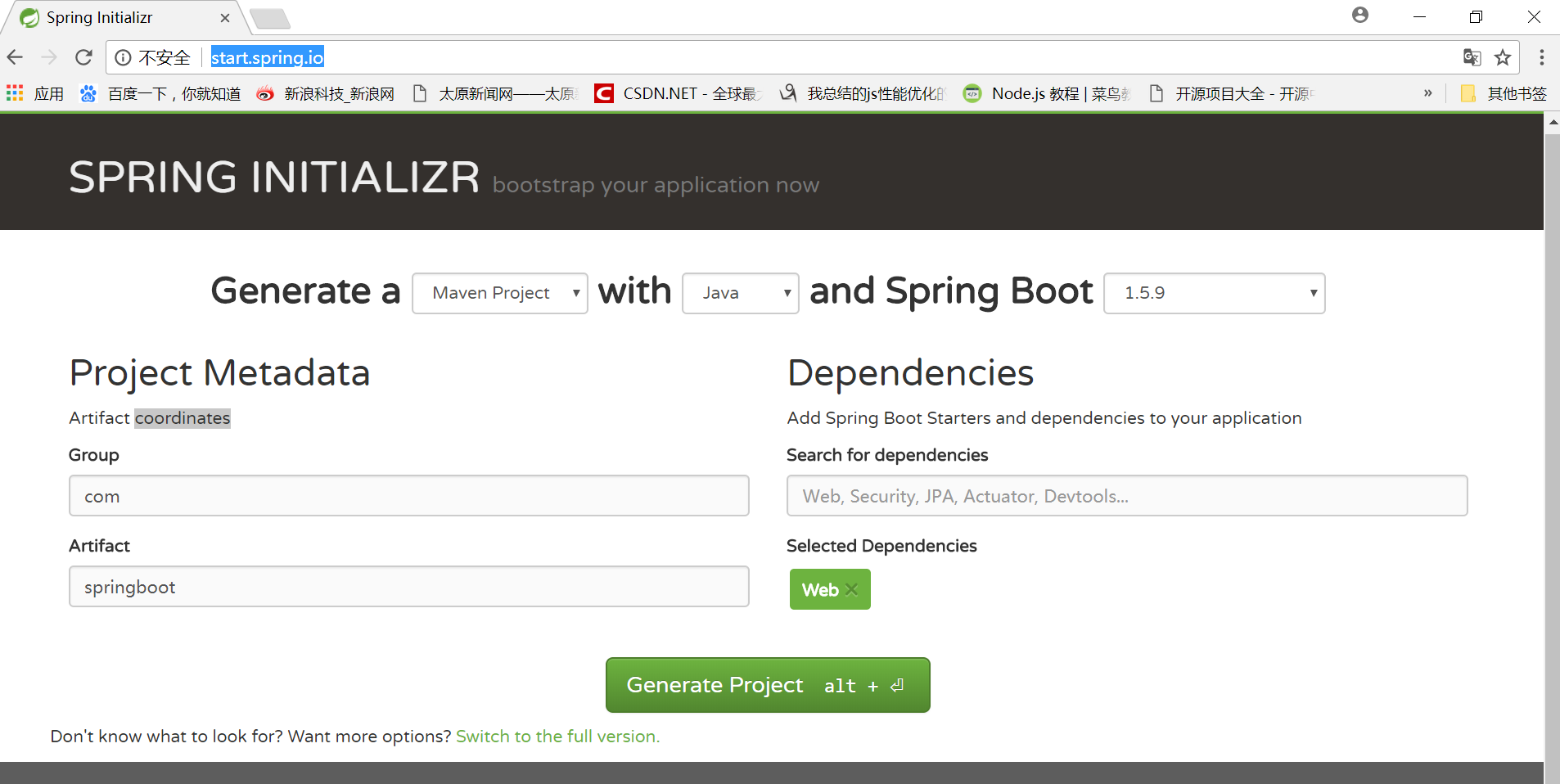
### （6）无代码生成和xml配置

Spring boot神奇的不是借助于代码生成来实现的，而是通过条件注解来实现的，这是spring4.x提供的新特性，spring4.x提倡使用java配置和注解配置组合，而spring boot不需要任何xml配置即可实现Spring的所有配置。

## Spring boot快速搭建（重点）

### 使用网站的方式搭建

1. 打开http://start.spring.io/网址
2. 填写项目信息



1. 选择项目的技术
2. 下载代码
3. 这是一个Maven项目，然后导入到开发工具中。

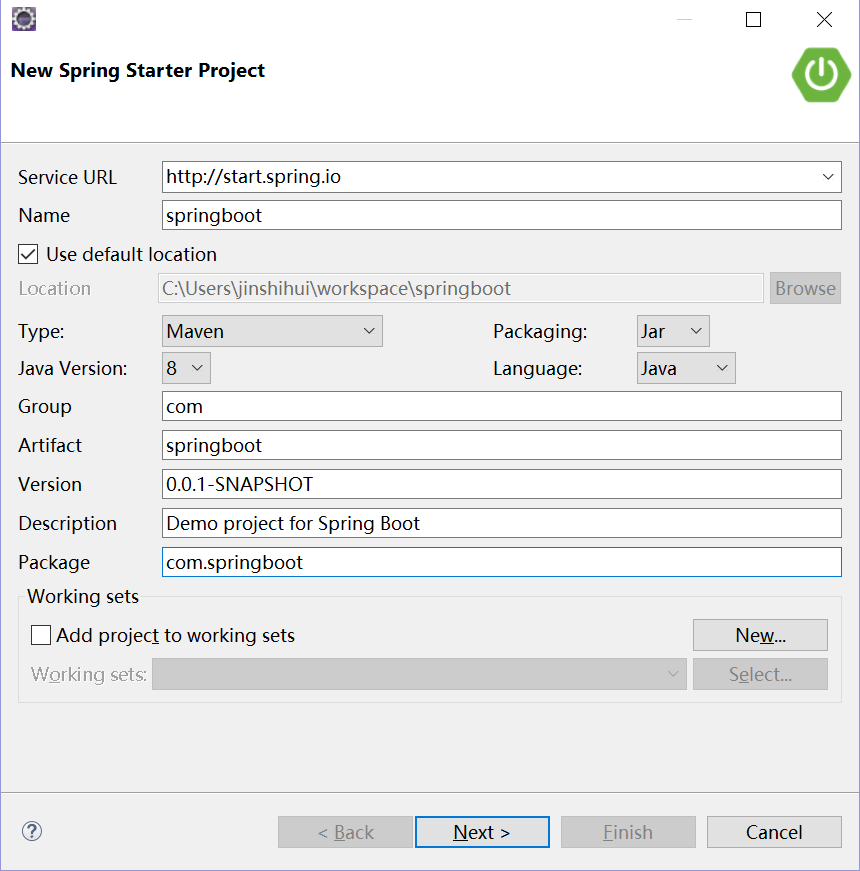
说明：选择的每一项技术都是spring boot的starter pom，我们选中的web，就是在maven里依赖spring-boot-starter-web。

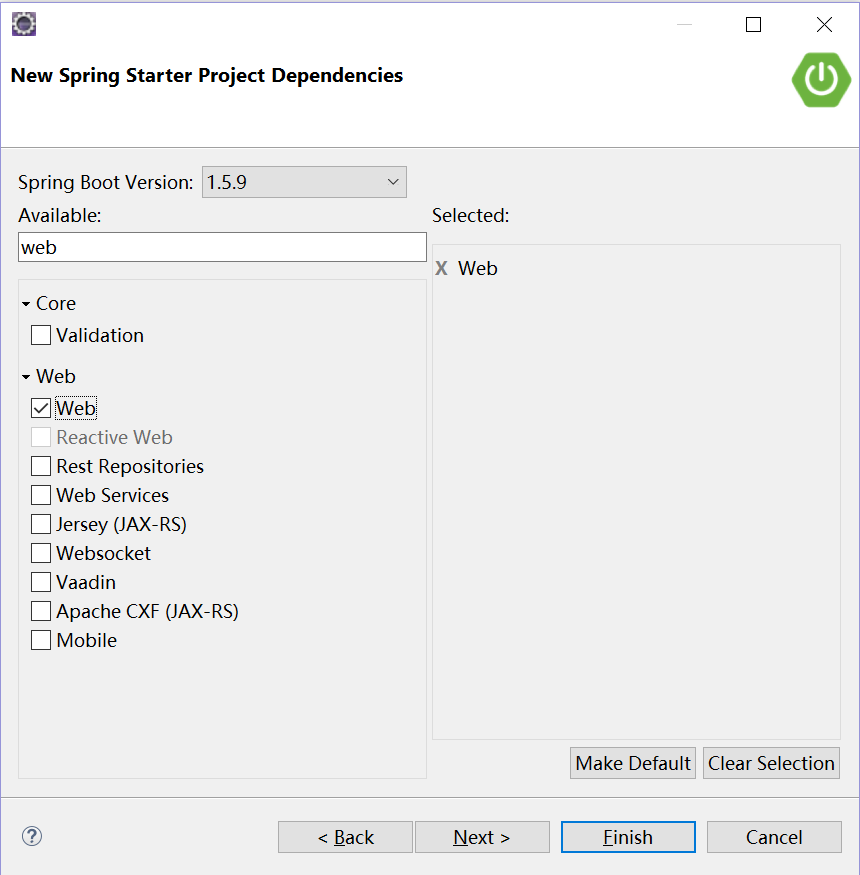
### 使用spring Tool Suite搭建（推荐）

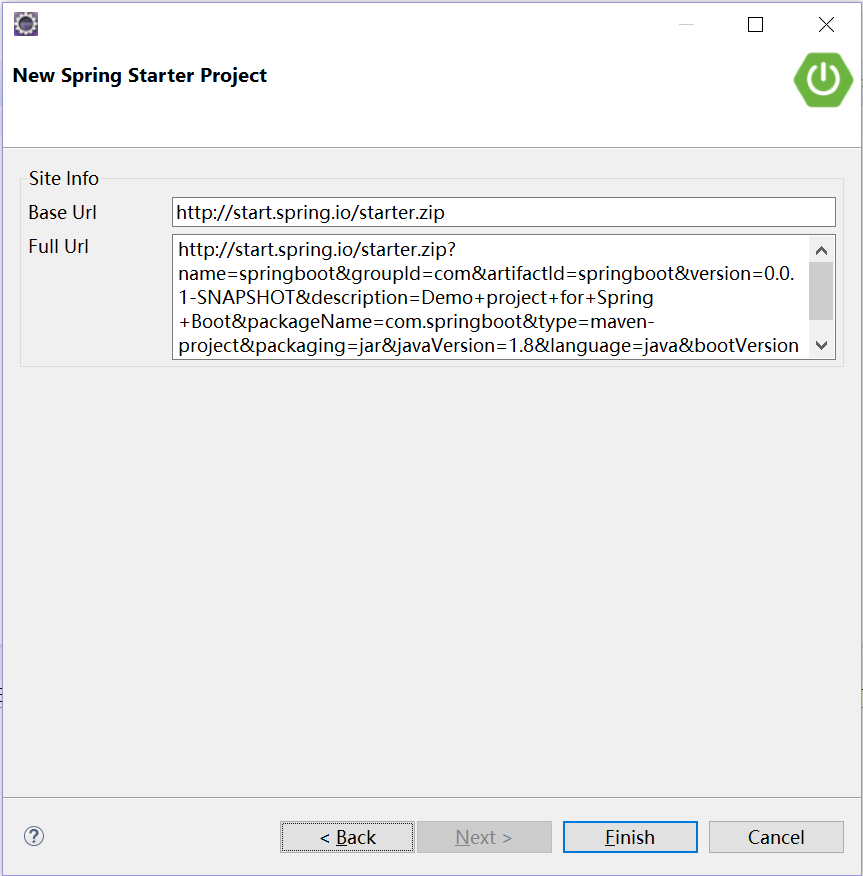
#### （1）新建spring starter project

选择新建spring starter project项目

#### （2）填写相关信息

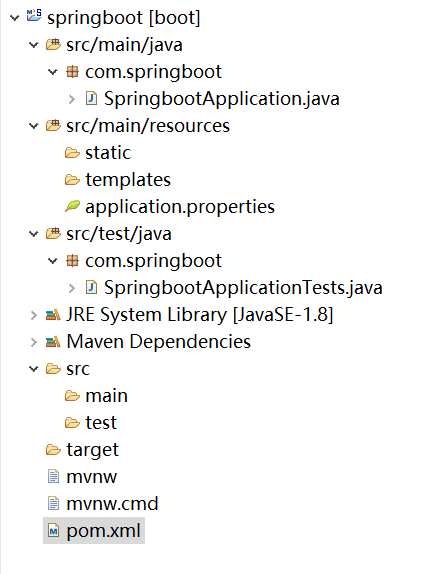




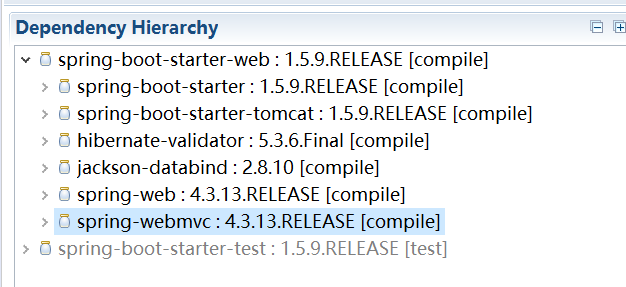


点击finish即可。

#### （3）项目结构如图



依赖树：



### 手工创建springboot项目（重点）

#### （1）创建Maven项目

#### （2）修改pom.xml文件

##### 添加spring-boot-starter-parent依赖

添加spring boot父级依赖，这样当前的项目就是spring boot项目了，spring-boot-starter-parent是一个特殊的starter，它用来提供相关的maven默认依赖，使用它之后，常用的包依赖可以省去version标签。

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.9.RELEASE</version>

<relativePath />

</parent>

##### 添加spring-boot-starter-web依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

##### 添加spring-boot-maven-plugin插件

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

###### 作用

在添加了该插件之后，当运行“mvn package”进行打包时，会打包成一个可以直接运行的 JAR 文件，使用“[java](http://lib.csdn.net/base/javase) -jar”命令就可以直接运行。这在很大程度上简化了应用的部署，只需要安装了 JRE 就可以运行。

Spring Boot的Maven插件（Spring Boot Maven plugin）能够以Maven的方式为应用提供Spring Boot的支持，即为Spring Boot应用提供了执行Maven操作的可能。

Spring Boot Maven plugin能够将Spring Boot应用打包为可执行的jar或war文件，然后以通常的方式运行Spring Boot应用。

###### 五个Goals

（1）spring-boot:repackage，默认goal。在mvn package之后，再次打包可执行的jar/war，同时保留mvn package生成的jar/war为.original

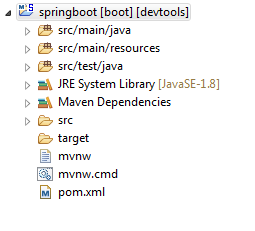
（2）spring-boot:run，运行Spring Boot应用

（3）spring-boot:start，在mvn integration-test阶段，进行Spring Boot应用生命周期的管理

（4）spring-boot:stop，在mvn integration-test阶段，进行Spring Boot应用生命周期的管理

（5）spring-boot:build-info，生成Actuator使用的构建信息文件build-info.properties

#### 项目结构



Spring Boot的基础结构共三个文件：

（1）src/main/java 程序开发以及主程序入口

（2）src/main/resources 配置文件

（3）src/test/java 测试程序

另外spingboot建议的目录结构如下

root package结构：com.example.myproject



1、Application.java 建议放到根目录下面,主要用于做一些框架配置

2、domain目录主要用于实体（Entity）与数据访问层（Repository）

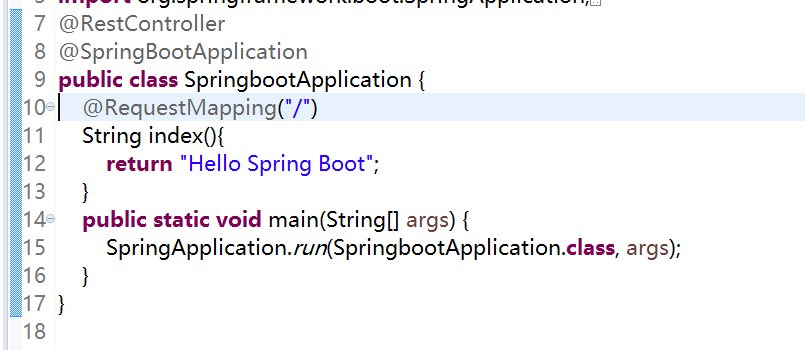
3、service 层主要是业务类代码

4、controller 层负责页面访问控制

采用默认配置可以省去很多配置，当然也可以根据自己的喜欢来进行更改，最后启动Application main方法，至此一个java项目搭建好了。

### 运行Spring boot项目

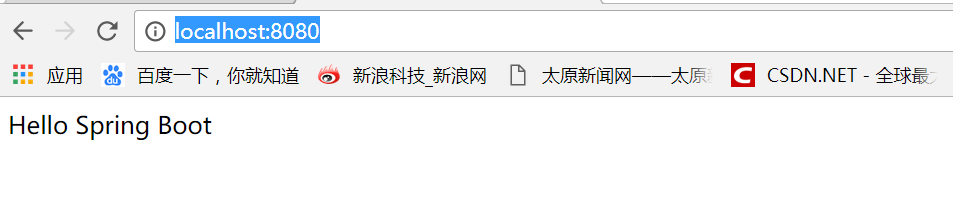
1. 新建spring boot项目
2. 在入口类中编辑，这里以SpringbootApplication类为例



说明：@SpringBootApplication是Spring Boot项目的核心注解，主要目的是开启自动配置。main方法主要作用是作为项目启动的入口。入口类中的方法不能加static，否则启动服务器会报空指针异常。

1. 在SpringbootApplication类中右键，选择Run as-Spring boot APP运行项目。
2. 等项目启动之后，访问http://localhost:8080/

如图：



说明：修改SpringbootApplication类中方法的内容时，需要重启服务器才能生效。springboot1.5.9这个版本启动有点问题，不要使用这个版本。

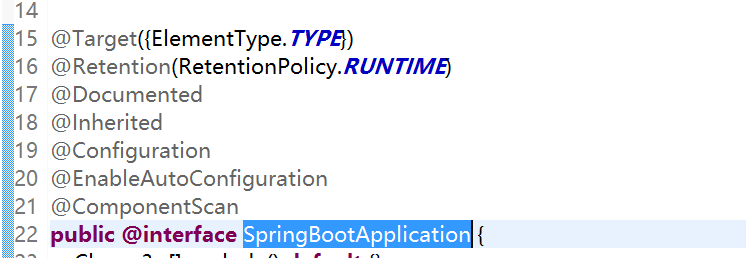
# Spring boot核心（重点 版本是1.5.9正式版）

## 基本配置

### 入口类和@SpringBootApplication注解

SpringBoot通常有一个名为\*Application的入口类，入口类里有一个main方法，这个main方法其实就是标准的Java应用的入口方法，在main方法中使用SpringApplication.run(类名.class,args)来启动Spring boot应用程序。

@SpringBootApplication是Spring boot的核心注解，它是一个组合注解。源码如下：



@SpringBootApplication注解主要组合了@Configuration、@EnableAutoConfiguration、@ComponentScan，若不使用@SpringBootApplication注解，则可以在入口类直接使用@Configuration、@EnableAutoConfiguration、@ComponentScan。

其中@EnableAutoConfiguration让Spring Boot根据类路径中的jar包依赖为当前项目进行自动配置。

Spring Boot会自动扫描@SpringBootApplication所在类的同级包以及下级包里的Bean（若为JPA项目还可以扫描标注@Entity的实体类）。建议入口类放置的位置在groupId和artifactId组合的包名下。

### 关闭特定的自动配置

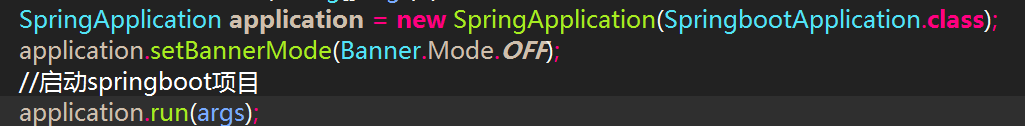
关闭特定的自动配置应该使用@SpringBootApplication注解的exclude参数。例如：

@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})

### 关闭Banner

main方法中修改

1.5.9版本中的方法：



### Spring Boot配置文件

#### 基本概述

Springboot使用一个全局的配置文件application.properties或者application.yml，放置在src/main/resources目录或者类路径的/config下。

Springboot不仅支持常规的properties配置文件，还支持yaml语言的配置文件，yaml是以数据为中心的语言，在配置数据的时候具有面向对象的特征。Springboot的全局配置文件的作用是对一些默认配置的配置值进行修改。

#### 修改示例

将tomcat的默认端口号修改为9090，将默认访问路径”/”修改为”/helloboot”，可以在application.properties中添加：

server.port=9090

server.context-path=/helloboot

说明：在springboot2.0中修改为server.servlet.context-path=/springboot

application.properties配置文件具有代码自动提示的功能，在日常开发中我们习惯用properties文件来配置，所以目前推荐使用properties进行配置。

#### yaml文件

##### 概念

YAML（Yet Another Markup Language），一种基于Unicode容易阅读，容易和脚本语言交互的，用来表达资料序列的编程语言。

##### 适应场景

（1）脚本语言：由于实现简单，解析成本很低，YAML 特别适合在脚本语言中使用。

（2）序列化：YAML是由宿主语言数据类型直转的比较适合做序列化。

（3）配置文件：写 YAML 要比写 XML 快得多(无需关注标签或引号)，并且比 INI 文档功能更强。由于兼容性问题，不同语言间的数据流转建议不要用 YAML。

##### 语言优点

（1）YAML易于人们阅读

（2）YAML数据在编程语言之间是可移植的

（3）YAML匹配敏捷语言的本机数据结构

（4）YAML具有一致的模型来支持通用工具

（5）YAML支持单程处理

（6）YAML具有表现力和可扩展性

（7）YAML易于实现和使用

##### 语法（重点）

（1）使用空格 Space 缩进表示分层，不同层次之间的缩进可以使用不同的空格数目，但是同层元素一定左对齐，即前面空格数目相同（不能使用 Tab，各个系统 Tab对应的 Space 数目可能不同，导致层次混乱）

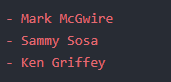
（2）‘#’表示注释，只能单行注释，从#开始处到行尾

（3）破折号后面跟一个空格（a dash and space）表示列表

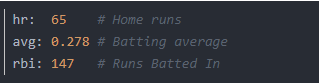
（4）用**冒号和空格**表示键值对 key: value

（5）简单数据（scalars，标量数据）可以不使用引号括起来，包括字符串数据。用单引号或者双引号括起来的被当作字符串数据，在单引号或双引号中使用C风格的转义字符

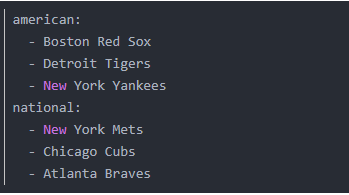
###### 简单数据列表



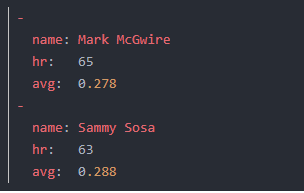
###### 简单数据键值对以及注释



###### 简单数据列表键值对

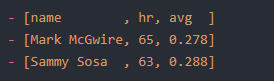


###### 键值对列表

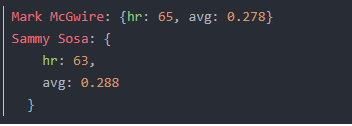


YAML 还支持流类型，用中括号括起来表示列表，用逗号分隔元素；用大括号括起来表示键值对，用逗号分隔元素

###### 列表的列表



###### 键值对的键值对



### starter pom

#### 基本概念

Springboot为我们提供了简化企业级开发绝大多数场景的starter pom，只要使用了应用场景所需要的starter pom，相关的技术配置就会消除，就可以得到Springboot为我们提供的自动配置的Bean。

starter pom：springboot 通过starter依赖为项目的依赖管理提供帮助，starter依赖其实就是特殊的maven依赖,利用了传递依赖解析,把常用库聚合在一起,组成了几个为特定功能而定制的依赖。springboot starter 旨在为具有依赖关系的一组jar包提供统一的条目框架。它们通常应该从项目描述符（pom.xml）继承，以便获得配置版本的所有父依赖关系。

#### 官方提供的starter pom

##### spring-boot-starter（重点）

//spring boot核心的starter，包含自动配置、日志、yaml配置文件的支持。

##### spring-boot-starter-aop

//使用spring-aop和AspectJ支持面向切面编程

##### spring-boot-starter-data-jpa（重点）

//对JPA的支持，包含spring-data-jpa、spring-orm和Hiberbate。

##### spring-boot-starter-jdbc

//对JDBC数据库的支持

##### spring-boot-starter-mail

//对javax.mail的支持

##### spring-boot-starter-security

//对spring-security的支持

##### spring-boot-starter-web（重点）

//对Web项目开发的支持，包含Tomcat和spring-webmvc

##### spring-boot-starter-Tomcat

//Spring boot默认的servlet容器Tomcat

##### spring-boot-starter-Jetty

//使用Jetty作为Servlet容器替换Tomcat

##### spring-boot-starter-logging

//springboot默认的日志框架Logback

##### spring-boot-starter-log4j

//支持使用Log4j日志框架

##### spring-boot-starter-websocket

//对Websocket开发的支持

##### spring-boot-starter-ws

//对Spring Web Service的支持

#### 第三方starter pom

除了官方的starter pom外，还有第三方为Spring boot所写的starter pom。

##### Spring Batch(高级用法)

//地址：https://github.com/codecentric/spring-boot-starter-batch-web

其他上网查询就可以。

### 使用xml配置

Spring boot推荐零配置，即无xml配置，但是在实际项目开发中，可能有一些特殊要求你必须使用xml配置，这时我们可以通过Spring提供的@ImportResource注解来加载xml配置文件。

示例：

@ImportResource（{“classpath:some-context.xml”,”classpath:another-context.xml”}）

## 外部配置

Springboot允许使用properties文件、yaml文件或者命令行参数作为外部配置。

### 命令行参数配置（不是重点）

Springboot可以是基于jar包运行，打成jar包的程序可以直接通过下面的命令去运行：

java –jar xx.jar

可以通过以下命令修改tomcat的端口号

java –jar xx.jar –server.port=9090

### 常规属性配置（重点）

#### 基本概念

在springboot里我们只需要在application.properties文件中定义属性，直接使用@Value注入就可以了。

#### @Value

为了简化读取properties文件中的配置值，Spring支持@Value注解的方式来获取配置值，这种方式大大简化了项目配置，提高业务中的灵活性。

两种使用方法：

（1）@Value("#{configProperties['key']}")

（2）@Value("${key}")

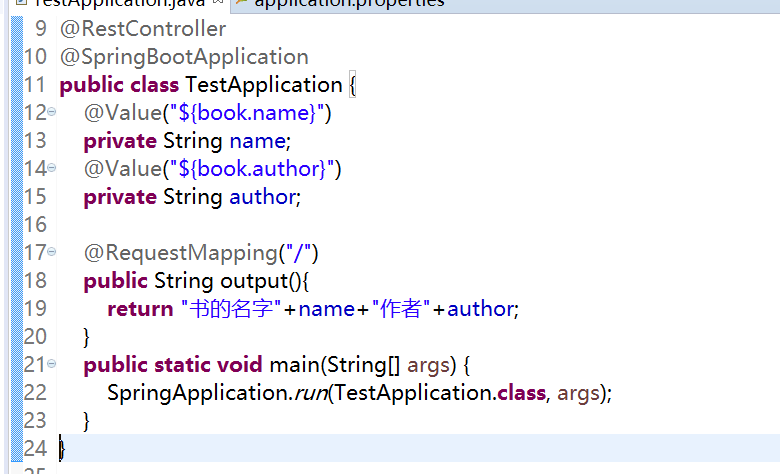
#### 示例

1、编写application.properties文件，增加属性：

book.name = springboot

book.author = jinshihui

2、修改入口类



3、运行访问



说明：这种情况下pom.xml文件中只能配置spring-boot-starter-web，不能配置其他。

### 类型安全的配置（基于properties文件重点）

#### 基本概念

Springboot还提供了基于类型安全的配置方式，通过@ConfigurationProperties将properties属性文件和一个Bean及其属性关联，从而实现类型安全的配置。

#### @ConfigurationProperties

外部化配置的注释，如果您想绑定和验证一些外部属性，可以将其添加到@Configuration类中的类定义或@Bean方法中。注意，与@Value相反，SpEL表达式没有计算，因为属性值是外部化的。

#### 示例

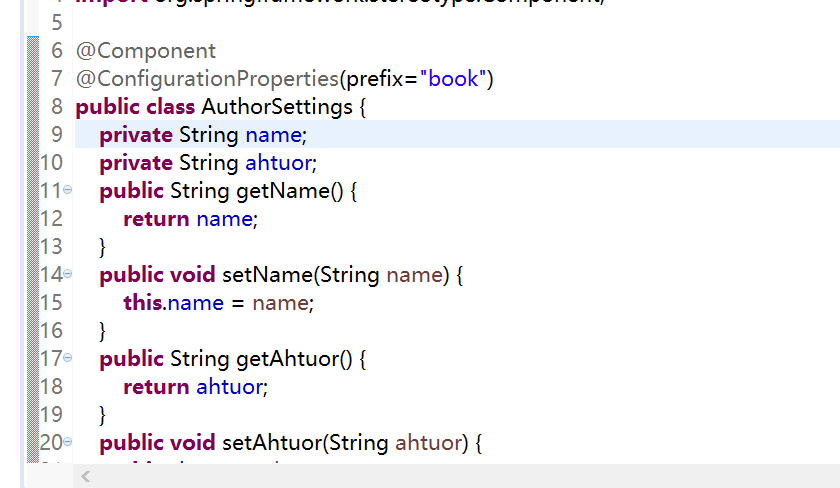
（1）在配置文件中添加配置

在application.properties文件里添加：

book.name = springboot

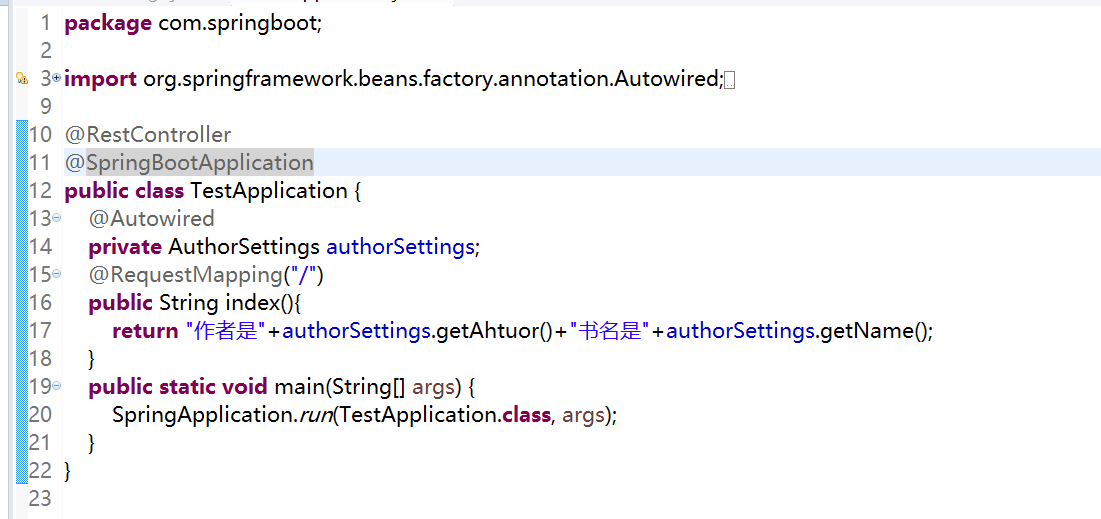
book.author = jinshihui

（2）类型安全的Bean



说明：通过@ConfigurationProperties注解加载properties文件内的配置，通过prefix属性指定properties配置的前缀。

（3）检验代码



说明：可以用@Autowired直接注入该配置Bean。

（4）运行



说明：这种情况下pom.xml文件中只能配置spring-boot-starter-web，不能配置其他。

## 日志配置

### 基本概念

Springboot支持Java Util Logging、Log4J、Log4J2和Logback作为日志框架，无论使用哪种日志框架，Spring boot已为当前使用日志框架的控制台输出及文件输出做好了配置。默认情况下Springboot使用Logback作为日志框架。

### 配置日志级别

格式为logging.level.包名=级别

示例：

logging.level.org.springframework.web=DEBUG

### 配置日志文件的地址

logging.file=D:/mylog/log.log

logging.path=/user/local/log

说明：这两种方法都可以。

//如果要把日志文件输出到项目根目录下，如下配置

logging.file=log.log

## Profile配置

### 基本概念

Profile是Spring用来针对不同的环境对不同的配置提供支持，全局的Profile配置使用application-{profile}.properties。例如：application-prod.properties

通过在application.properties中设置spring.profile.active=prod来指定获得的Profile。

### 示例

我们分为生产和开发环境，生产环境下端口号是80，开发环境下端口号是8888。

1、分别在src/main/resource目录下面新建application-dev.properties和application-prod.properties文件，内容如下：

application-dev.properties：

server.port=80

application-prod.properties:

server.port=8888

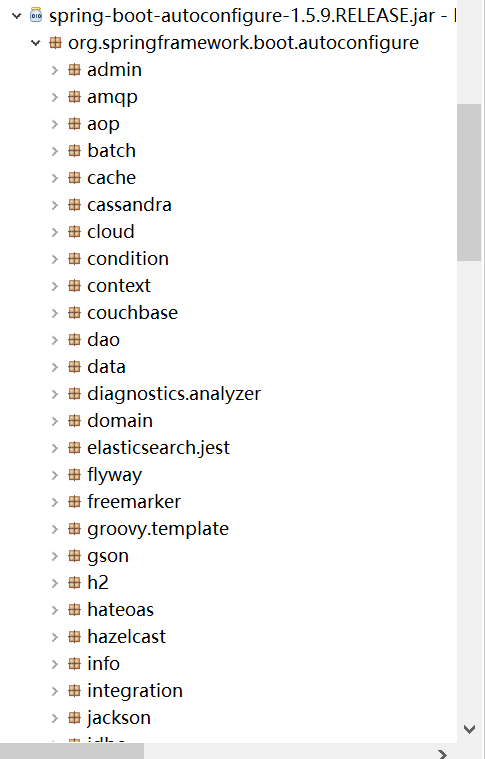
2、在application.properties文件中增加：spring.profiles.active=dev

3、运行项目，就会看到端口号是80开始的。

4、修改spring.profiles.active=prod，就会看到端口号是8888开始的。

## Springboot运行原理（重点）

Springboot关于自动配置的源码在spring-boot-autoconfigure-xxx.jar内，主要包含如下：



### 查看自动配置的报告

如果想知道Springboot为我们做了哪些自动配置，可以查看这里的源码。可以通过下面三种方式查看当前项目中已启用和未启用的自动配置的报告。

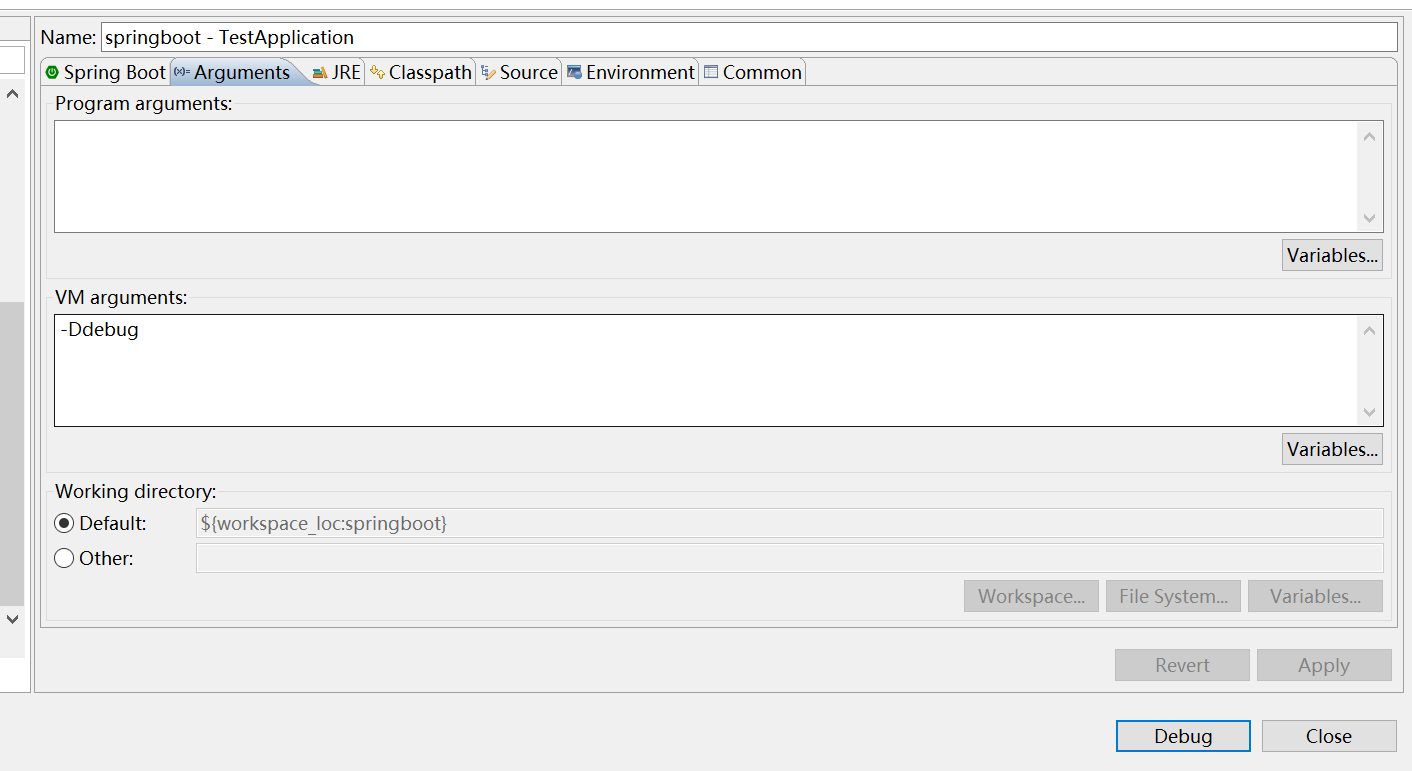
#### （1）运行jar时增加--debug参数

java –jar xx.jar --debug

#### （2）在application.properties中设置属性

debug=true

#### （3）在STS中设置-Ddebug

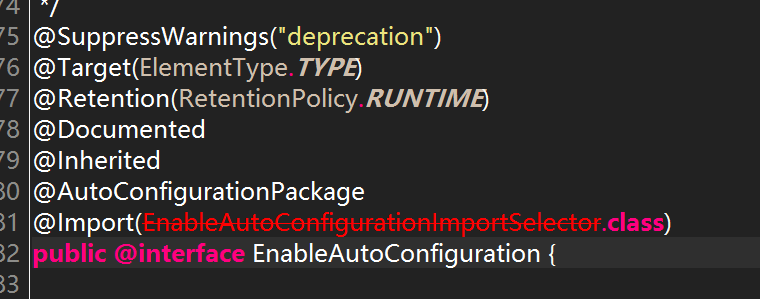


此时启动可在控制台中输出，已启用的自动配置是positive matches，未启用的自动配置是：negative matches。

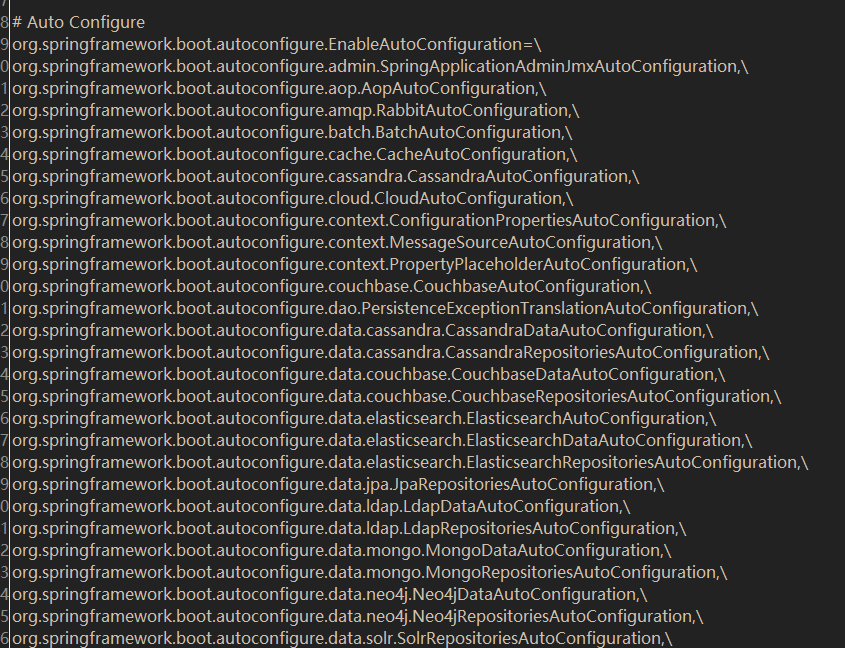
### 运作原理

关于Springboot的运作原理，我们还是回归到@SpringBootApplication注解上来，这个注解是一个组合注解，它的核心功能是由@EnableAutoConfiguration注解提供的。

#### @EnableAutoConfiguration注解的源码



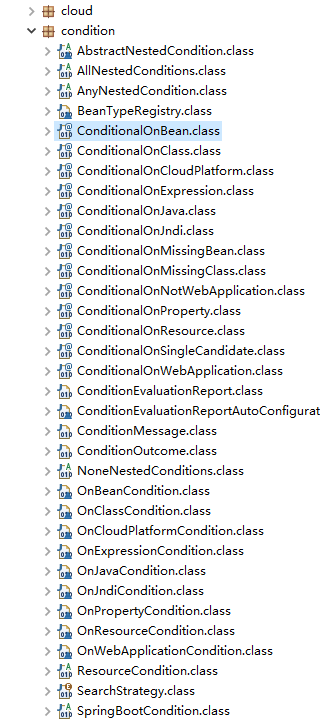
这里的关键功能是@Import注解导入的配置功能，EnableAutoConfigurationImportSelector类使用SpringFactoriesLoader.loadFactoryNames方法来扫描具有META-INF/spring.factories文件的jar包，而我们的spring-boot-autoconfigure.jar里就有一个spring.factories文件，此文件中声明了自动配置。



### 核心注解

#### @Conditional

@Conditional，其他注解都组合了这个注解（@Conditional），在spring-boot-autoconfigure.jar的org.springframework.boot.autoconfigure.condition包下，条件注解如下：



这些注解都是组合了@Conditional元注解，只是使用了不同的条件。

## Springboot常用的注解

### @SpringBootApplication

这个注解是Springboot项目的核心注解，目的是开启自动配置。这是一个便利的注解，等价于声明@Configuration @EnableAutoConfiguration @ComponentScan这三个注解。

属性：

exclude：排除自动配置的类

excludeName：排除自动配置的类名

scanBasePackages：用于扫描带注释组件的基础包

### @Configuration

表示一个类声明了一个或多个@Bean的方法，并且可以由Spring容器处理，以便在运行时为这些bean生成bean定义和服务请求。

### @EnableAutoConfiguration

启用Spring Application Context的自动配置，尝试猜测和配置您可能需要的bean，自动配置类通常基于您的类路径和您定义的bean来应用。

### @ComponentScan

配置组件扫描指令以与@Configuration类一起使用。提供与Spring XML <context：component-scan>元素并行的支持。

### @ImportResource

指示包含要导入的bean定义的一个或多个资源，主要是引入spring配置文件.xml

### @Value

字段或方法/构造函数参数级别的注释，指示受影响参数的默认值表达式。通常用于表达式驱动的依赖注入，还支持动态解析处理程序方法参数，例如在Spring MVC中。

这个注解用于字段、方法、参数、注解类上。

### @ConfigurationProperties

外部化配置的注释。如果要绑定并验证某些外部属性（例如，来自.properties文件），请将其添加到@Configuration类中的类定义或@Bean方法上。

这个注解用在类或者方法上。

属性：

prefix：有效绑定到此对象的属性的名称前缀

### @Import

表示要导入一个或多个带有@Configuration的类，提供的功能是和在Spring xml文件中的<import />元素的相等的。

这个注解用在类上。

### @Conditional

表示组件仅在所有指定条件匹配时才有资格进行注册。条件是可以在bean定义注册之前以编程方式确定的任何状态。

这个注解用在类、方法上。

### @EnableWebMvc

将注解添加到@Configuration类可从WebMvcConfigurationSupport导入Spring MVC配置。

### @SpringBootTest

可以在运行基于Spring Boot的测试的测试类上指定的注释。在常规Spring TestContext框架之上提供以下功能：

（1）当没有定义特定的ContextConfiguration＃loader（）时，使用SpringBootContextLoader作为默认的ContextLoader。

（2）在未使用嵌套的@Configuration时自动搜索SpringBootConfiguration，并且未指定显式类。

### @EnableCaching

### @MapperScan

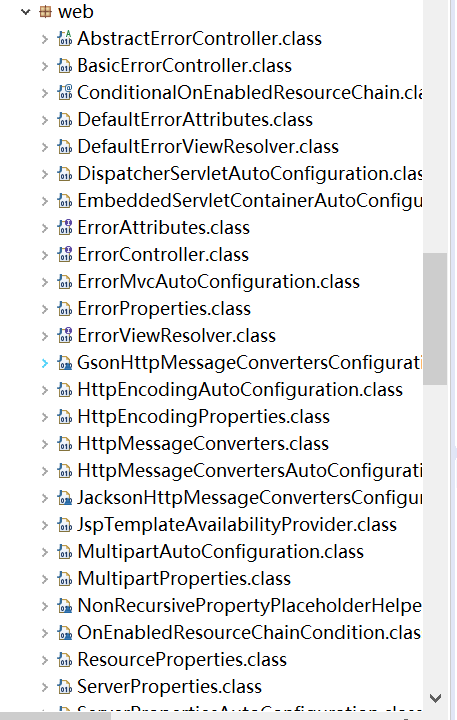
### @AliasFor

支持属性别名的Spring注释，从Spring Framework 4.2开始，核心Spring中的几个注释已更新为使用@AliasFor来配置其内部属性别名

# Springboot Web开发（重点）

## Springboot的Web开发支持

Springboot提供了spring-boot-starter-web为Web开发予以支持，spring-boot-starter-web为我们提供了嵌入的Tomcat以及Spring MVC的依赖，而Web相关的自动配置存储在spring-boot-autoconfigure.jar的org.springframework.boot.autoconfigure.web包下。



从这些文件名可以看出：

ServerPropertiesAutoConfiguration和ServerProperties自动配置内嵌的Server容器。

HttpEncodingAutoConfiguration和HttpEncodingProperties用来自动配置http的编码。

MultipartAutoConfiguration和MultipartProperties用来自动配置上传文件的属性。

JacksonHttpMessageConvertersConfiguration用来自动配置mappingJackson2HttpMessageConverter和mappingJackson2XmlHttpMessageConverter

WebMvcAutoConfiguration和WebMvcProperties配置Spring MVC

## Thymeleaf模板引擎

Springboot提供了大量的模板引擎，Springboot推荐使用Thymeleaf作为模板引擎，因为Thymeleaf提供了完美的Spring MVC的支持。最好不要使用JSP，因为JSP在内嵌的Servlet容器中运行有一些问题，内嵌的Tomcat、Jetty不支持已jar形式运行JSP。

### 基础知识

Thymeleaf是一个Java类库，它是一个xhtml、xml和html5的模板引擎，可以作为MVC的Web应用的View层。Thymeleaf提供了额外的模块与Spring MVC集成，所以我们可以使用Thymeleaf完全替代JSP。

#### 优点

1、简洁漂亮、容易理解

2、完美支持HTML5 使用浏览器直接打开页面

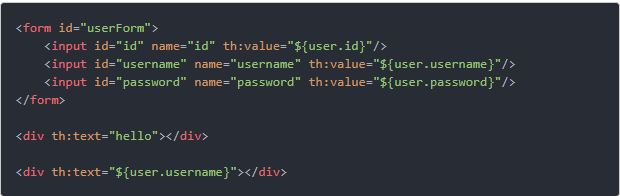
3、不新增标签 只需增强属性

Thymeleaf 最为显著的特征是增强属性，任何属性都可以通过th:xx 来完成交互，例如th:value最终会覆盖value属性。

#### 基础语法

##### 变量表达式${}

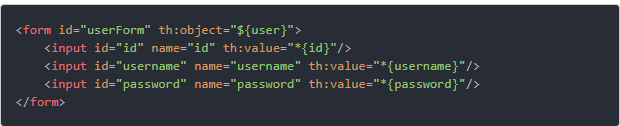
使用方法：直接使用th:xx = "${}" 获取对象属性 。例如：



##### 选择变量表达式 \*{}

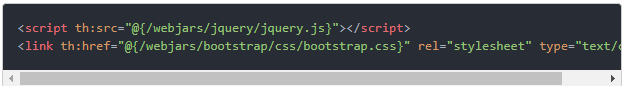
使用方法：首先通过th:object 获取对象，然后使用th:xx = "\*{}"获取对象属性。

这种简写风格极为清爽，推荐大家在实际项目中使用。 例如：



##### 链接表达式 @{}

使用方法：通过链接表达式@{}直接拿到应用路径，然后拼接静态资源路径。例如：



##### 片段表达式 ~{}

片段表达式是Thymeleaf的特色之一，细粒度可以达到标签级别，这是JSP无法做到的。

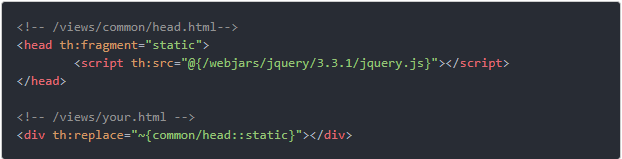
片段表达式拥有三种语法：

（1）~{ viewName } 表示引入完整页面

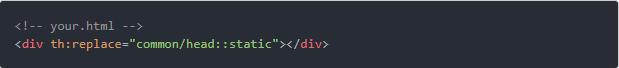
（2）~{ viewName ::selector} 表示在指定页面寻找片段 其中selector可为片段名、jquery选择器等

（3）~{ ::selector} 表示在当前页寻找

使用方法：首先通过th:fragment定制片段 ，然后通过th:replace 填写片段路径和片段名。例如：



在实际使用中，我们往往使用更简洁的表达，去掉表达式外壳直接填写片段名。例如：



值得注意的是，使用替换路径th:replace 开头请勿添加斜杠，避免部署运行的时候出现路径报错。（因为默认拼接的路径为spring.thymeleaf.prefix = classpath:/templates/）

##### 消息表达式#{}

即通常的国际化属性：#{msg} 用于获取国际化语言翻译值。例如：



##### 其他表达式

在基础语法中，默认支持字符串连接、数学运算、布尔逻辑和三目运算等。例如：



#### 内置对象

##### 七大基础对象

${#ctx} 上下文对象，可用于获取其它内置对象

${#vars}: 上下文变量

${#locale}：上下文区域设置

${#request}: HttpServletRequest对象

${#response}: HttpServletResponse对象

${#session}: HttpSession对象

${#servletContext}: ServletContext对象

##### 常用的工具类

#strings：字符串工具类

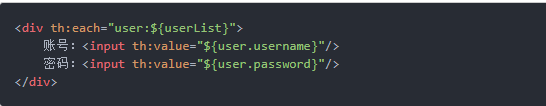
#lists：List 工具类

#arrays：数组工具类

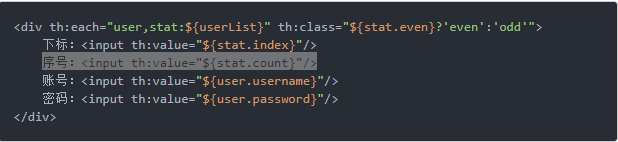
#sets：Set 工具类

#### 迭代循环

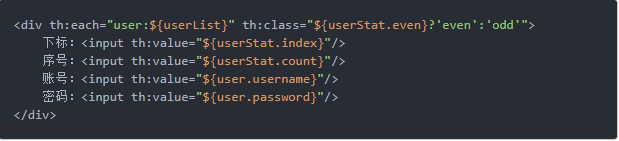
想要遍历List集合很简单，配合th:each 即可快速完成迭代。例如遍历用户列表：



在集合的迭代过程还可以获取状态变量，只需在变量后面指定状态变量名即可，状态变量可用于获取集合的下标/序号、总数、是否为单数/偶数行、是否为第一个/最后一个。例如：



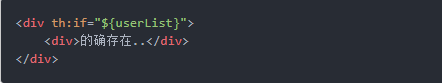
如果缺省状态变量名，则迭代器会默认帮我们生成以变量名开头的状态变量 xxStat， 例如：



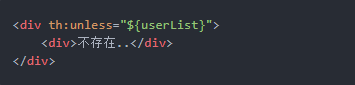
#### 条件判断

条件判断通常用于动态页面的初始化，使用th:if

例如：



如果想取反则使用unless ，使用th:unless，例如：



#### 日期格式化

使用默认的日期格式(toString方法) 并不是我们预期的格式：Mon Dec 03 23:16:50 CST 2018



此时可以通过时间工具类#dates来对日期进行格式化：2018-12-03 23:16:50



#### 引入Thymeleaf

引入Thymeleaf：

<html xmlns:th=“http://www.thymeleaf.org”>

//通过xmlns:th= http://www.thymeleaf.org命名空间，将镜头页面转换为动态的视图，需要进行动态处理的元素将使用”th:”为前缀。

#### 引入css文件

语法如下：

<link th:href="@{bootstrap/css/bootstrap.min.css}" rel="stylesheet"/>

<link th:href="@{bootstrap/css/bootstrap-theme.min.css}" rel="stylesheet"/>

#### 引入js文件

语法如下：

<script type="text/javascript" th:src="@{jquery-1.10.2.min.js}"></script>

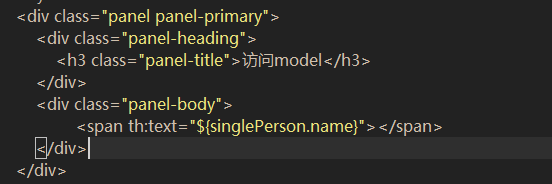
<script type="text/javascript" th:src="@{bootstrap/js/bootstrap.min.js}"></script>

#### 引入静态资源文件的方法

通过@{}引入web静态资源文件，比如css、js文件。

#### 访问model中的数据

通过${}访问model中的属性，这和JSP是很像的。

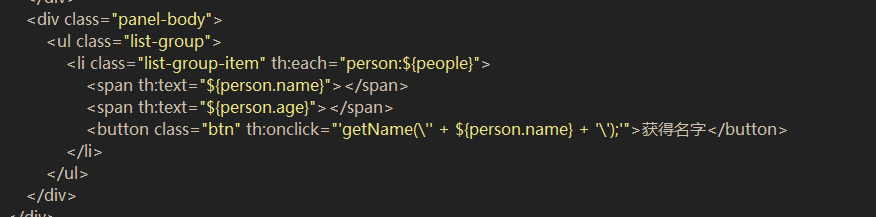


说明：使用<span th:text=”${singlePerson.name}”></span>访问model中的singlePerson的name属性。注意：需要处理的动态内容需要加上”th:”前缀。

#### model中的数据迭代

使用th:each来做循环（th:each=”person:${people}”）,person作为迭代元素来使用，然后访问迭代元素中的属性就可以。

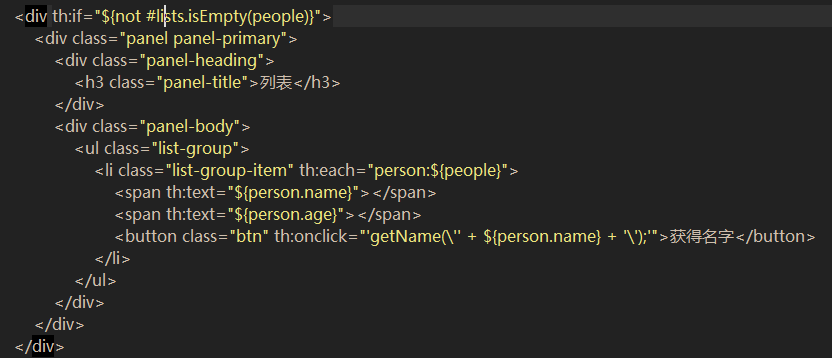
示例代码：



#### 数据判断

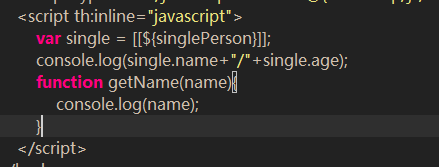
通过th:if=”${not #lists.isEmpty(people)}”表达式判断people是否为空，Thymeleaf支持>、<、>=、<=、 ==、!=作为比较条件，同时也支持将SpringEL表达式语言用于条件中。

示例代码



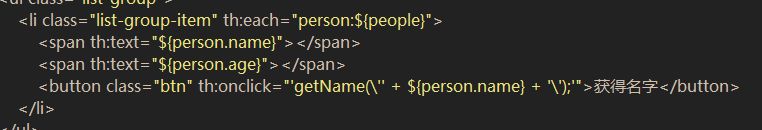
#### 在JS中访问model

通过th:inline=“javascript”添加到script标签，这样js代码即可访问model中的属性。通过”[[${}]]”格式获得实际的值（Bean对象）。



#### 在html中访问model的属性

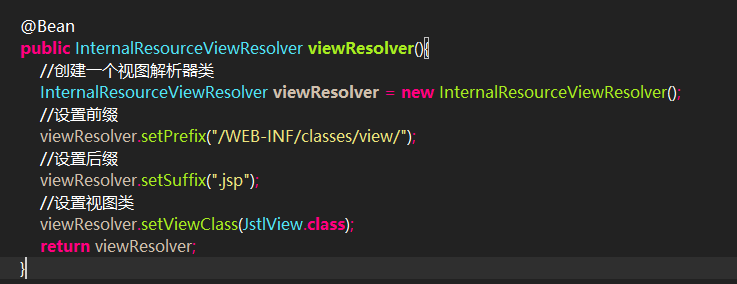
我们需要在列表后单击每一行后面的按钮获得model中的值，代码如下：



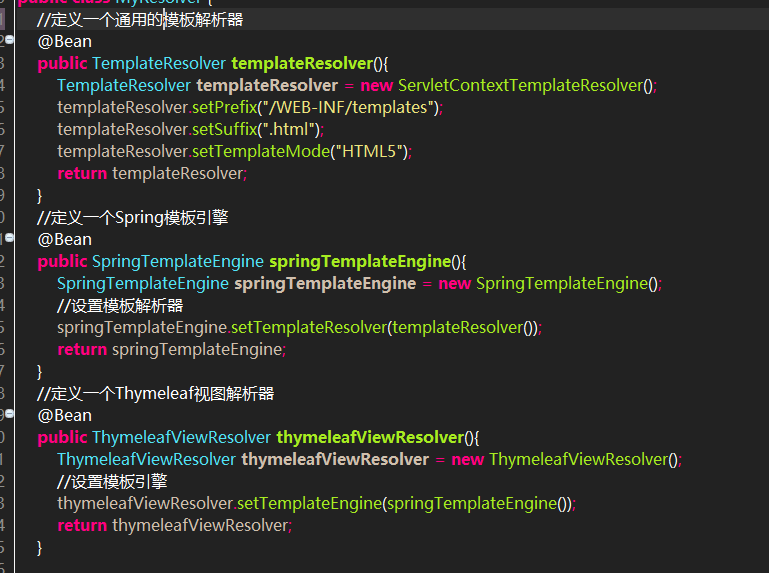
说明：注意格式th:onclick=”’getName(\”+${person.name}+’\’);’”

### 与Spring MVC集成（原理）

在Spring MVC中，如果我们需要集成一个模板引擎的话，需要定义ViewResolver，而ViewResolver需要定义一个View，代码示例：

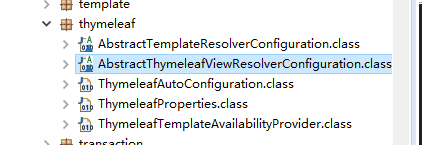


我们使用了JstiView定义了一个InternalResourceViewResolver，因而使用Thymeleaf作为我们的模板引擎也应该做类似的定义，庆幸的是Thymeleaf为我们定义好了org.thymeleaf.spring4.view.ThymeleafView和org.thymeleaf.spring4.view.ThymeleafViewResolver（默认使用ThymeleafView作为View）。Thymeleaf为我们提供了一个SpringTemplateEngine类，用来驱动在Spring MVC下使用Thymeleaf模板引擎，另外还提供了一个TemplateResolver用来设置通用的模板引擎（包含前缀、后缀），这使我们在Spring MVC中集成Thymeleaf引擎变得十分简单。代码如下：



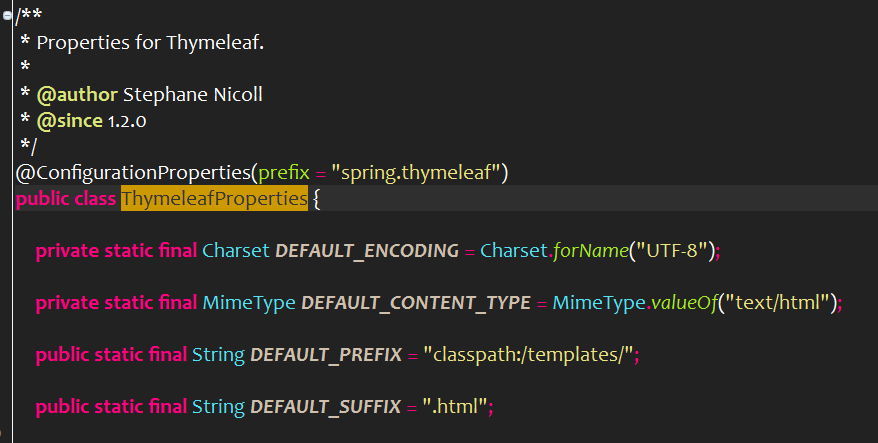
### Springboot Thymeleaf的支持（重点）

在Springboot中这一切（配置视图解析器）都是不需要的，Springboot通过org.springframework.boot.autoconfigure.thymeleaf包对Thymeleaf进行了自动配置，如下：



通过ThymeleafAutoConfiguration类对集成所需要的Bean进行自动配置，包括templateResolver、templateEngine、thymeleafViewResolver的配置。

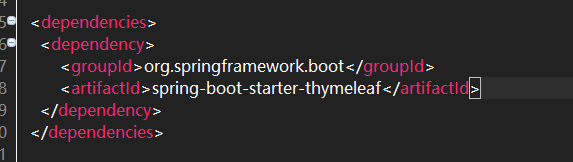
通过ThymeleafProperties类来配置Thymeleaf，在applicaiotn.properties中，以spring.thymeleaf开头来配置，通过查看ThymeleafProperties类的主要源码，我们可以看出如何设置属性以及默认配置，源码如下：



### 实战（重点）

#### （1）在pom.xml文件中编辑

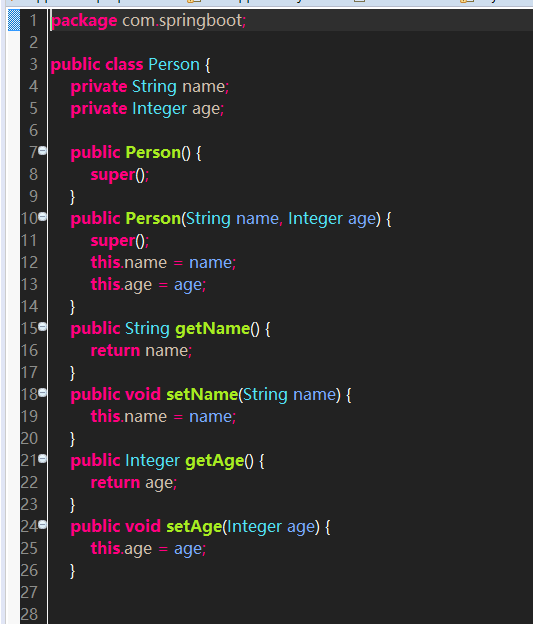
添加如下依赖：



说明：spring-boot-starter-thymeleaf会自动包含spring-boot-starter-web。

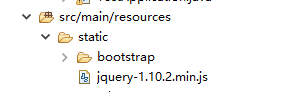
#### （2）编写JavaBean

用于在模板页面中展示数据使用



#### （3）添加脚本样式静态文件

根据默认原则，脚本样式、图片等静态文件应放在src/main/resources/static下，这里引入了Bootstrap和jQuery。

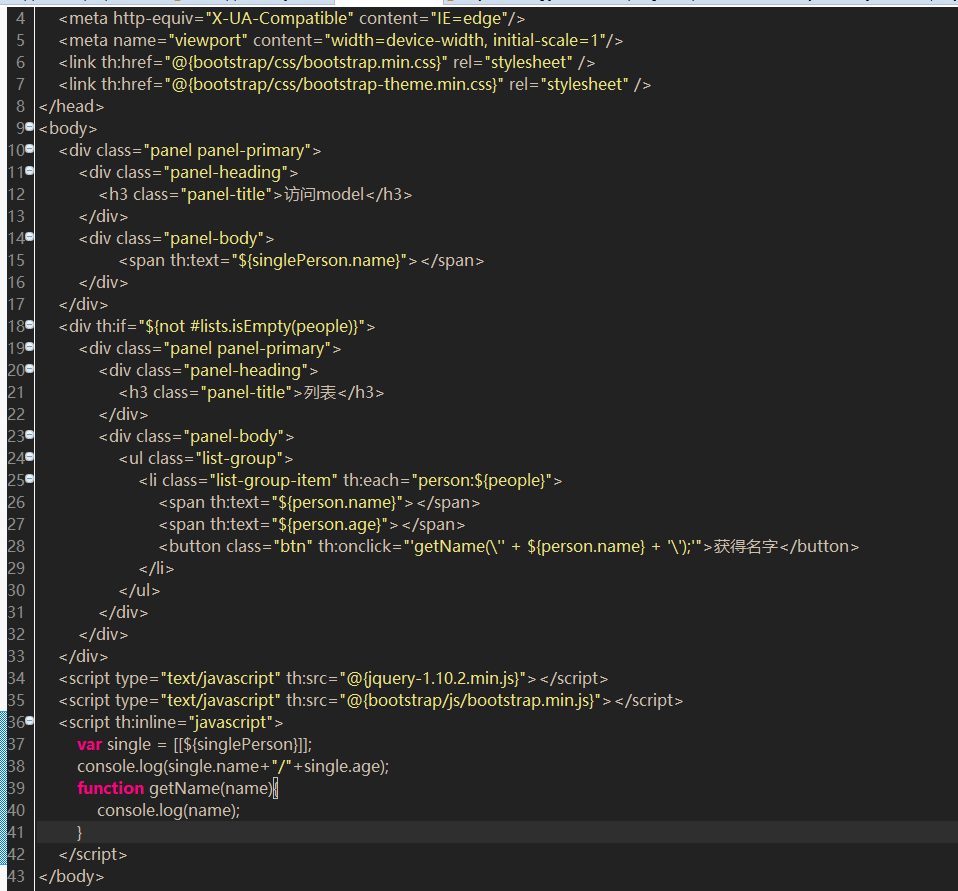


#### （4）编写页面

根据默认规则，页面应该放置在src/main/resources/templates下，在src/main/resources/templates下新建index.html文件，如图：

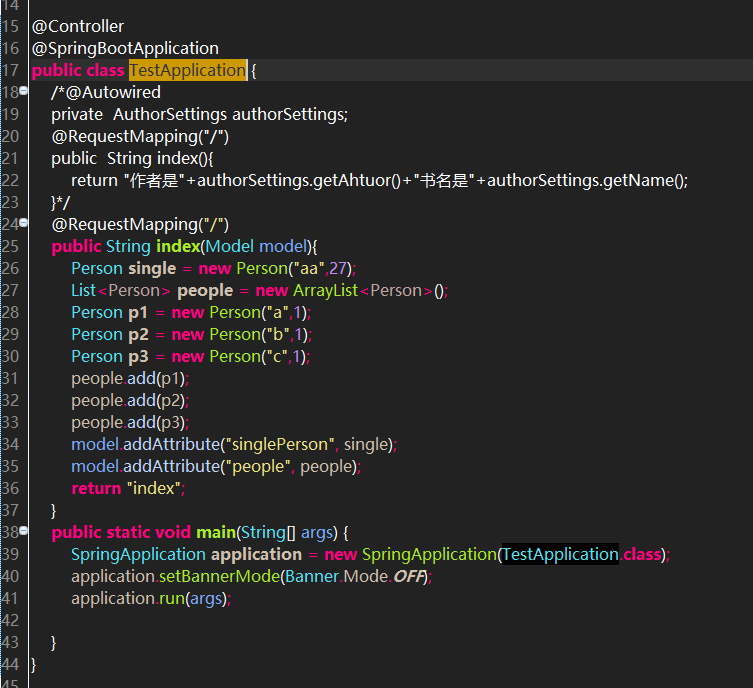


代码如下：



#### （5）准备数据

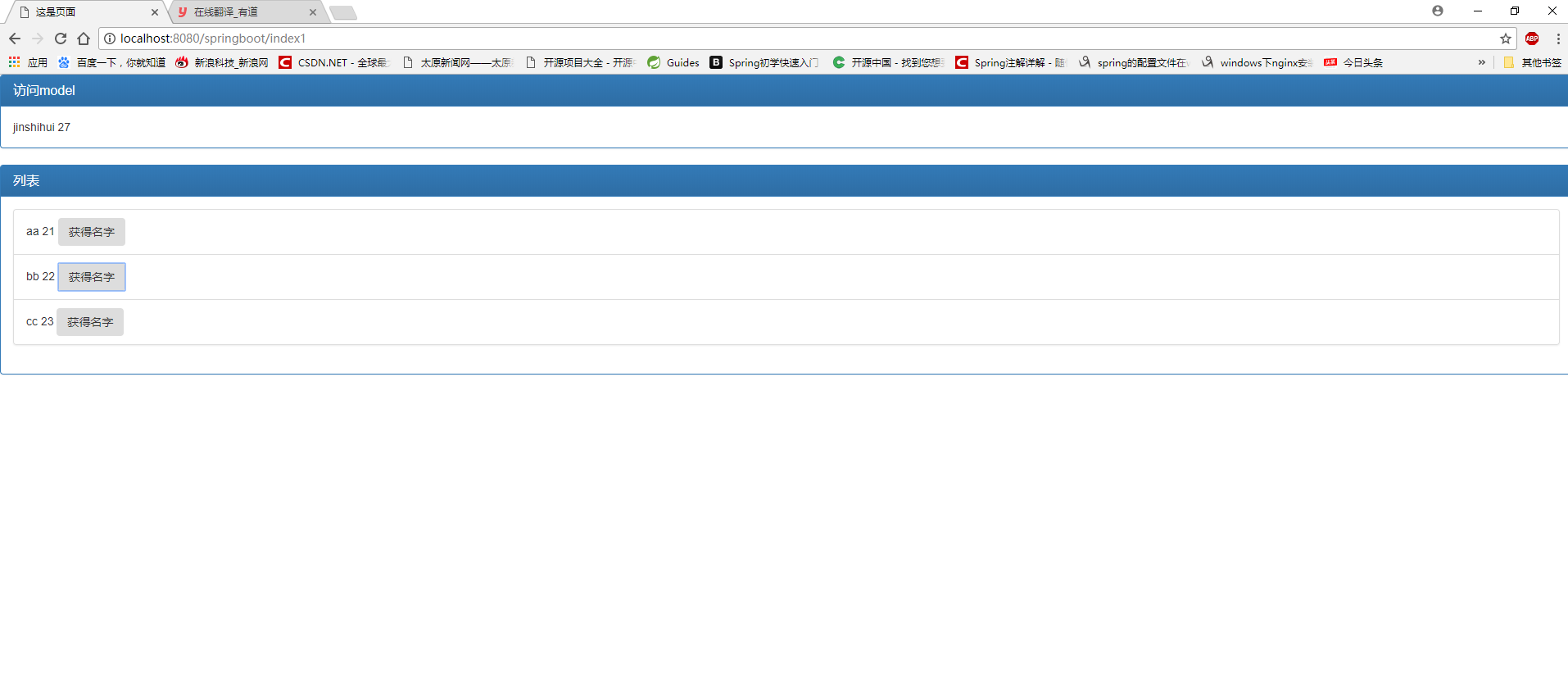
在Springboot的入口类TestApplication中进行编辑，代码如下：



说明：也可以在一个Controller类中编写这个方法，不用在Springboot的入口类中编写。

#### （6）运行

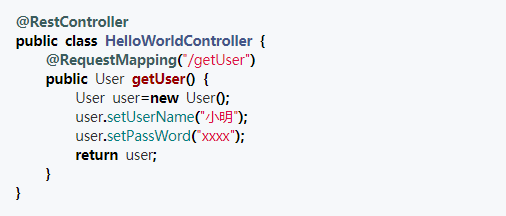
访问http://localhost:8080/springboot/index,如图：



说明：修改了html页面的内容不需要重启服务器。

## JSON接口开发

在Springboot项目中，只需要在类上添加  @RestController  即可，默认类中的方法都会以json的格式返回。



如果我们需要使用页面开发只能使用 @Controller，结合模板去使用。

## Web相关配置（重点）

### Springboot提供的自动配置

通过查看WebMvcAutoConfiguration类和WebMvcProperties类的源码，可以发现Springboot为我们提供了如下的配置：

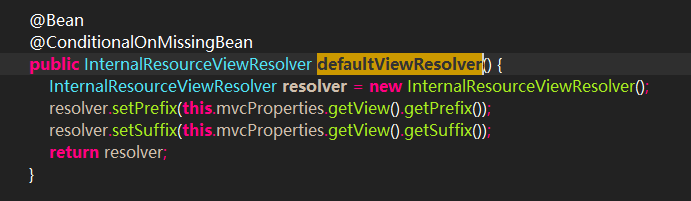
#### 自动配置ViewResolver

##### （1）BeanNameViewResolver

当控制器（@Controller）中的一个方法的返回值是字符串（视图名）会根据BeanNameViewResolver类去查找Bean的名称为返回字符串的View来渲染视图。

##### （2）InternalResourceViewResolver

这是一个极为常用的ViewResolver，主要是通过设置前缀、后缀，以及控制器中的方法来返回视图名的字符串，已得到实际页面。Springboot源码如下（源码在WebMvcAutoConfiguration类中）：



#### 自动配置的静态资源

##### （1）类路径文件

把类路径下的/static、/public、/resources和/META-INF/resources文件夹下的静态文件直接映射为/\*\*，可以通过http://localhost:8080/\*\*来访问。

示例：

我在static路径下面存放一个jquery.js文件，我可以直接通过http://localhost:8080/springboot/jquery.js来访问这个js文件。



##### （2）webjar

webjar就是将我们常用的脚本框架封装在jar包中的jar包（前端资源的jar包形式），把webjar的/META-INF/resources/webjars/下的静态文件映射为/webjar/\*\*

#### 自动配置的Formatter和Converter

只要我们定义了Converter接口、GenericConverter接口和Formatter接口的实现类的Bean，这些Bean就会自动注册到Spring MVC中。

#### 自动配置的HttpMessageConverters

在Springboot中，如果要新增定义的HttpMessageConverter，则只需定义一个你自己的HttpMessageConverters类的Bean，然后在此Bean中注册自定义HttpMessageConverter即可。示例：

@Bean

public HttpMessageConverters customConverters(){

HttpMessageConverter<?> customConverter1 = new CustomConverter1();

HttpMessageConverter<?> customConverter2 = new CustomConverter2();

return new HttpMessageonverters(customConverter1, customConverter2);

}

#### 静态首页的支持

把静态首页index.html文件放置在如下目录：

classpath:/META-INF/resources/index.html

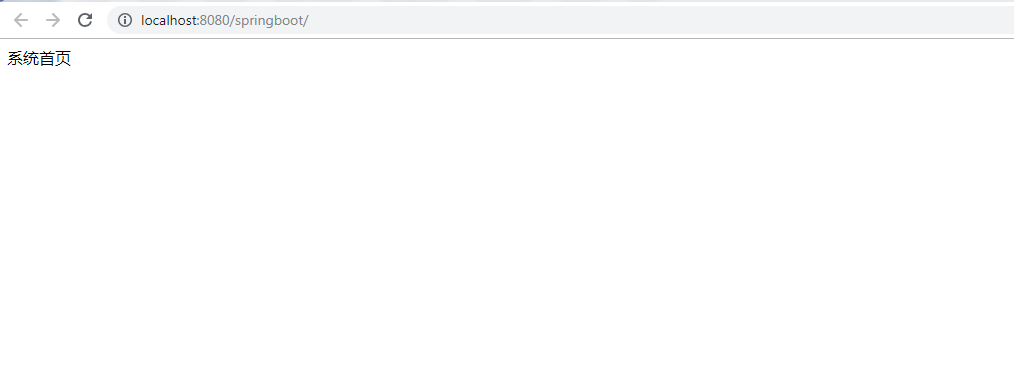
classpath:/resources/index.html

classpath:/static/index.html

classpath:/public/index.html

当我们访问应用程序根目录http://localhost:8080/时，会直接映射。

示例：

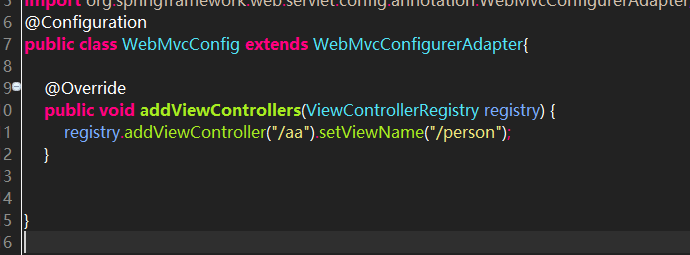


### 接管Springboot的Web配置

如果Springboot提供的SpringMVC默认配置不符合你的需求，则可以通过一个配置类（注解有@Configuration）加上@EnableWebMvc注解来实现完全自己控制的MVC配置。

在你既要保留Springboot提供的便利，又需要增加自己额外配置的时候，可以定义一个配置类并继承WebMvcConfigurerAdapter类，无须使用@EnableWebMvc注解。

示例：



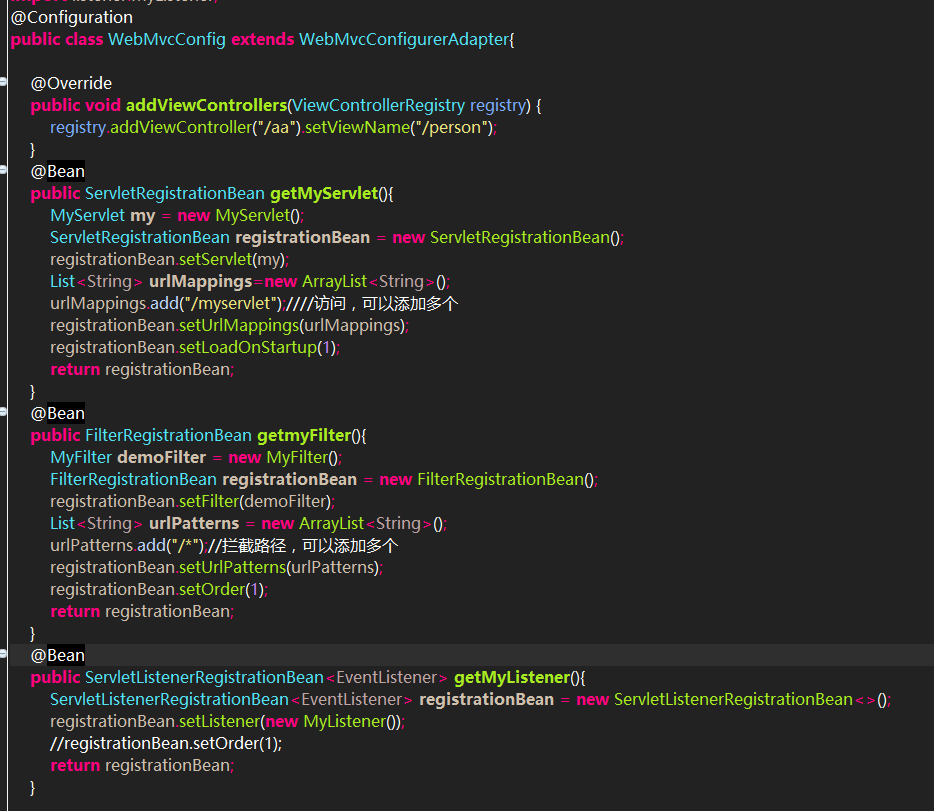
我们这里重写的addViewControllers方法，并不会覆盖WebMvcAutoConfiguration中的addViewControllers，这也就意味着我们自己的配置和Spring boot的自动配置同时有效，这也是我们推荐添加自己的MVC配置的方式。

### 注册Servlet、Filter、Listener

#### 基本概述

Spring Boot提供了 ServletRegistrationBean，FilterRegistrationBean，ServletListenerRegistrationBean这3个类来进行配置Servlet、Filter、Listener，当使用嵌入式的Servlet容器（Tomcat、jetty）时，我们通过注册Servlet、Filter、Listener声明为Spring Bean而达到注册的效果。

示例：



#### RemoteIpFilter

这个Servlet过滤器的大部分设计都是mod\_remoteip端口，这个Servlet过滤器用代理或负载均衡器通过请求头提供的IP地址列表替换请求的客户端远程IP地址和主机名。

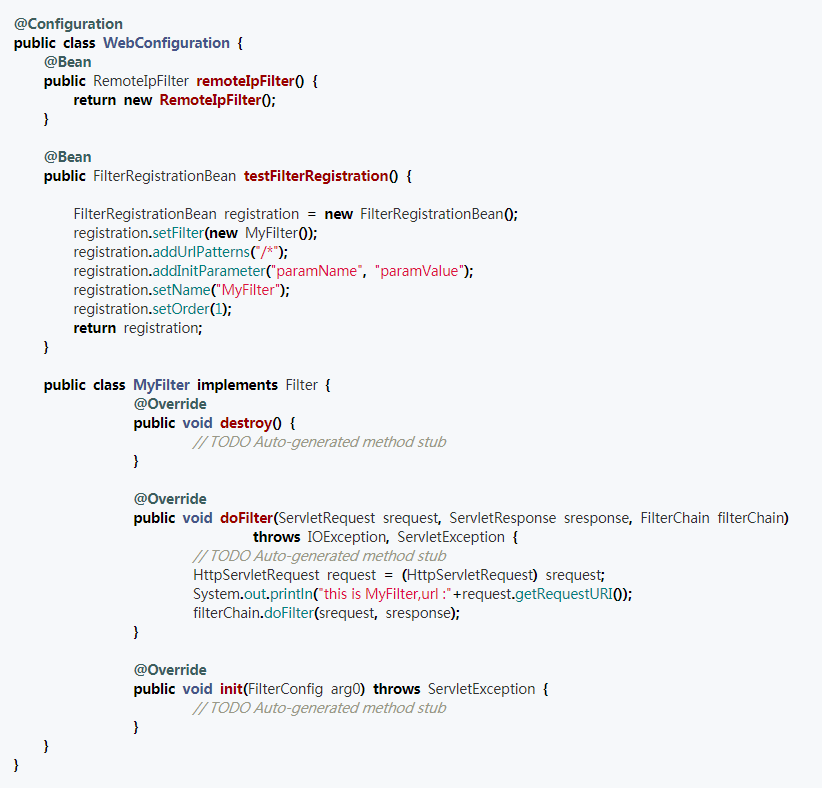
#### 自定义Filter

我们常常在项目中会使用filters用于调用日志、排除有XSS威胁的字符、执行权限验证等等。Spring Boot自动添加了OrderedCharacterEncodingFilter和HiddenHttpMethodFilter，并且我们可以自定义Filter。

两个步骤：

（1）定义一个类并实现Filter接口，实现doFilter方法

（2）添加@Configuration注解，将自定义Filter加入过滤链。



## Tomcat配置

### 配置Tomcat

关于Tomcat的所有属性都在org.springframework.boot.autoconfigure.web.ServerProperties配置类中做了定义，我们只需要在application.properties文件中配置属性就可以。通用的servlet容器配置都以server作为前缀，下面举例：

#### 配置端口号

server.port=需要配置的端口号，默认为8080

#### 配置session过期的时间

--用户session过期的时间，单位是秒

server.servlet.session.timeout=10

#### 配置访问的路径

--配置访问路径。默认是/

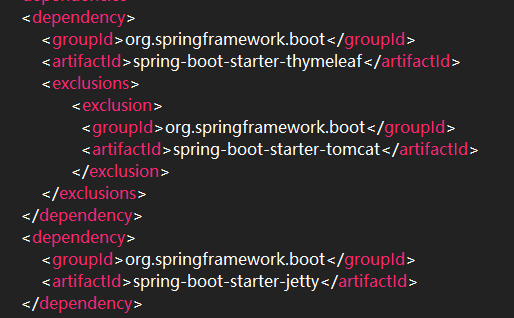
server.servlet.context-path=配置的访问路径

### 替换Tomcat

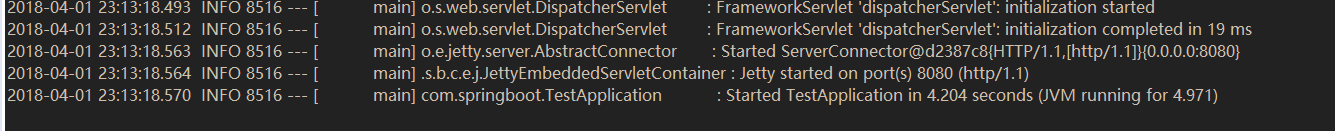
Springboot默认使用Tomcat作为内嵌的Servlet容器，如果要使用Jetty或者Undertow为Servlet容器，只需要修改spring-boot-starter-web的依赖就可以。

#### 替换为Jetty

在pom.xml文件中，将spring-boot-starter-web的依赖由spring-boot-starter-tomcat替换为spring-boot-starter-jetty，示例：



启动截图：



## SSL配置

### 基本概念

SSL是为网络通信提供安全及数据完整性的一种安全协议，SSL在网络传输层对网络连接进行加密。SSL协议位于TCP/IP协议与各种应用层协议之间，为数据通信提供安全支持。在基于B/S的Web应用中是通过HTTPS来实现SSL的。HTTPS是以安全为目的的HTTP通道，简单讲是HTTP的安全版，即在HTTP下加入SSL层，HTTPS的安全基础是SSL。我们在用springboot做SSL配置的时候需要做以下操作。

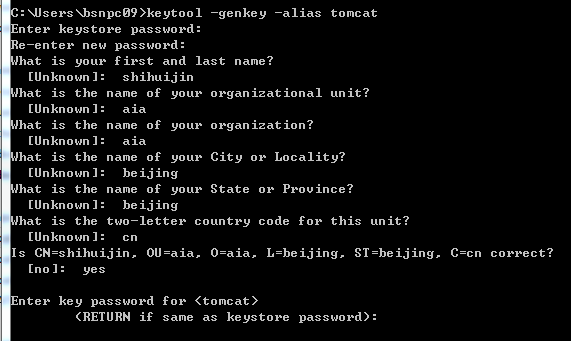
### 生成证书

使用SSL首先需要一个证书。证书既可以是自签名的，也可以是从SSL证书授权中心获得的。在这里我们使用自授权证书的生成。

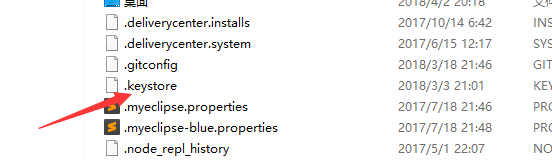
每一个JDK或者JRE里都有一个工具叫keytool，它是一个证书管理工具，可以用来生成自签名的证书，这个工具在jdk的bin目录下面。

在控制台输入如下命令，然后按照提示操作就可以了。

keytool –genkey –alias tomcat

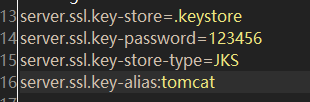


这时候我们在当前目录下生成了一个.keystore文件，这就是我们要用的证书文件。

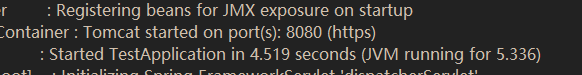


### Spring boot配置SSL

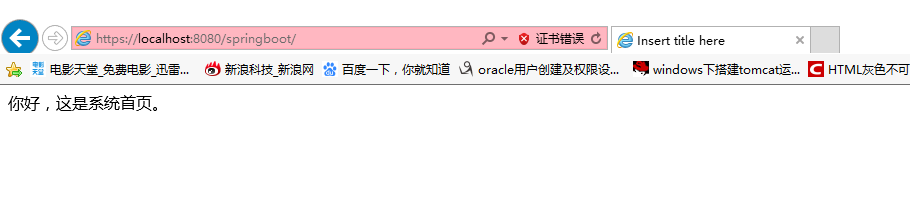
添加一个Index.html文件到src/main/resources/static下作为测试。将.keystore文件复制到项目的根目录，然后在application.properties中做如下的SSL的配置。



然后启动Tomcat，控制台输出效果：

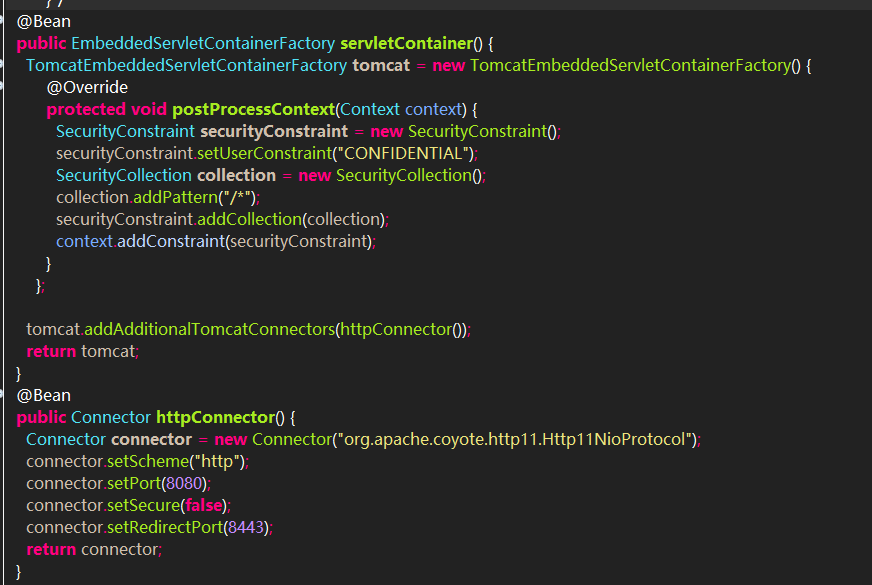


此时访问https://localhost:8080/springboot/，如下：

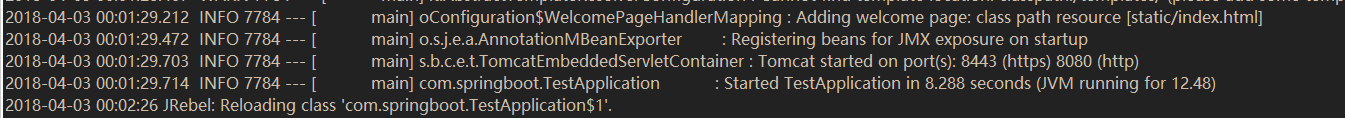


### http转https

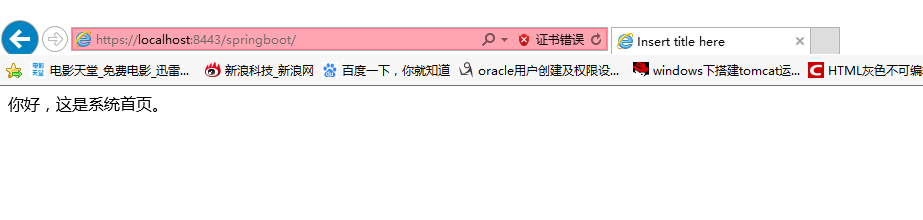
要实现这个功能，我们需要配置TomcatEmbeddedServletContainerFactory类，并且添加Tomcat的connector来实现。我们需要在springboot启动类中添加如下代码：



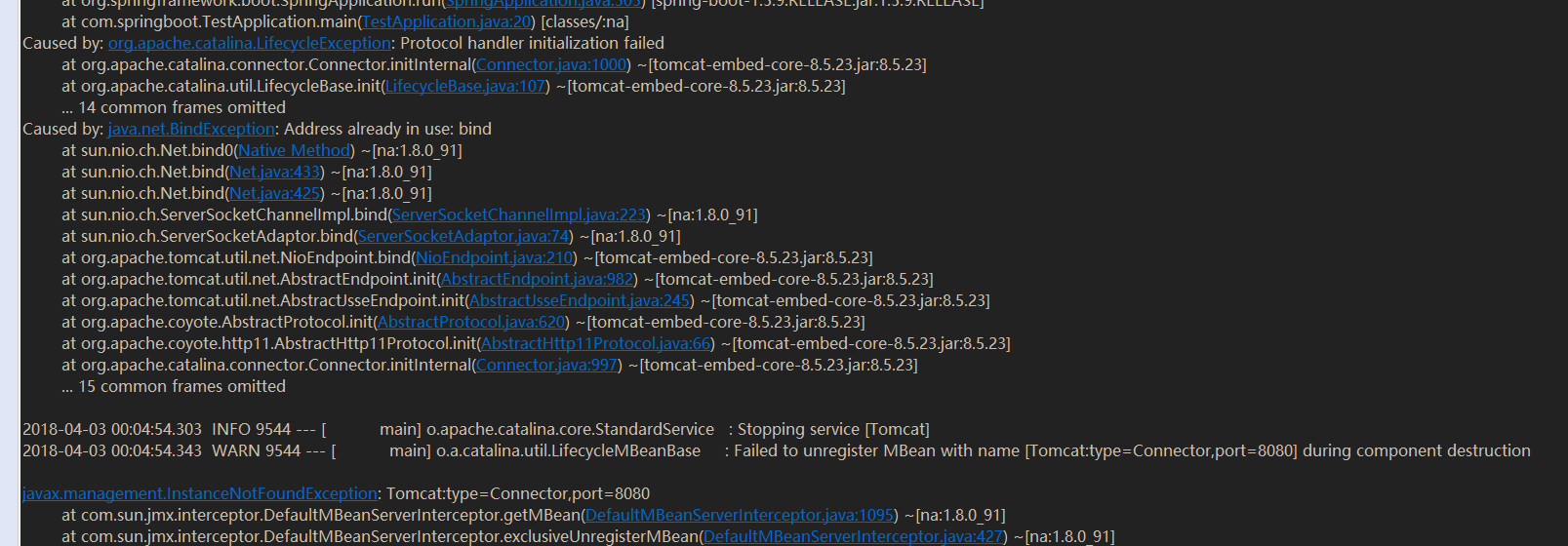
此时启动springboot时，控制台会输出如下信息：



我们访问http://localhost:8080/springboot/，就会跳到https://localhost:8443/springboot/这个网址：



说明：如果控制台启动报错，报错如下：

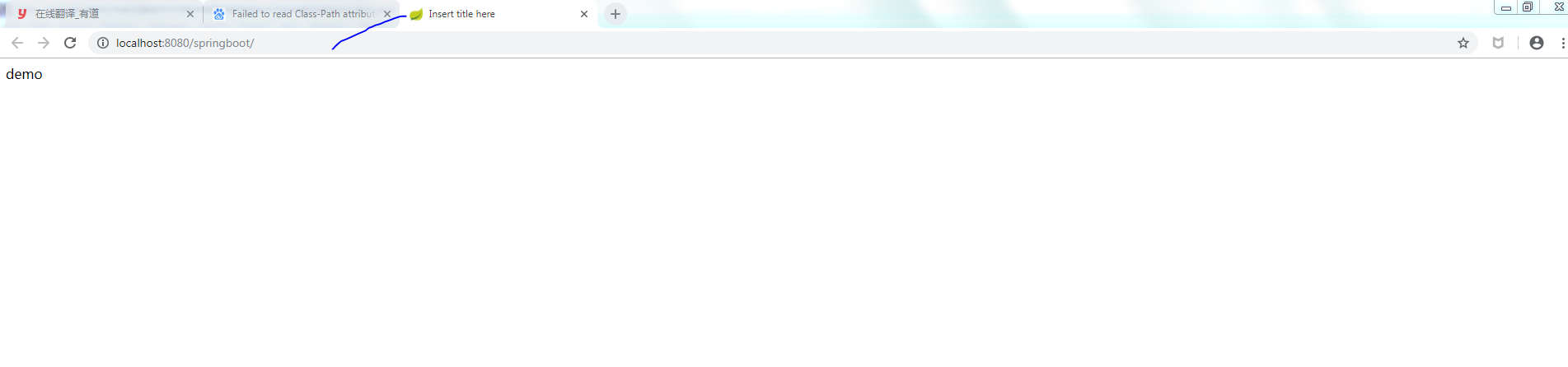


解决方法是要在application.properties配置文件中添加omcat的端口号server.port=8443，这里的端口号要和启动类中配置的重定向端口号要相同。

## Favicon配置

### 默认的Favicon

Springboot提供了一个默认的Favicon，每次访问应用的时候都能看到，如图：



### 关闭Favicon

我们可以在application.properties中设置关闭Favicon，默认是开启的。

### 设置自己的Favicon

若要设置自己的Favicon，只需将自己的favicon.ico（文件名不能修改）文件放置在类路径根目录、类路径META-INF/resources/下、类路径resources/下、类路径static/下或类路径public/下，我们将图片放置在src/main/resources/static下，如图：

# Springboot的数据访问

## Spring Data概述

Spring Data项目是Spring用来解决数据访问问题的一揽子解决方案，Spring Data是一个伞形项目，包含了大量关系型数据库及非关系型数据库的数据访问解决方案，Spring Data使我们可以快速且简单的使用普通的数据访问技术及新的数据访问技术。

Spring Data的使命是为数据访问提供熟悉且一致的基于Spring的编程模型，同时仍保留底层数据存储的特殊性。它使得使用数据访问技术，关系数据库和非关系数据库，map-reduce框架和基于云的数据服务变得容易。这是一个伞形项目，其中包含许多特定于给定数据库的子项目。

## 特点

（1）强大的存储库和自定义对象映射抽象

（2）从存储库方法名称派生动态查询

（3）实现域基类提供基本属性

（4）支持透明审核（创建，最后更改）

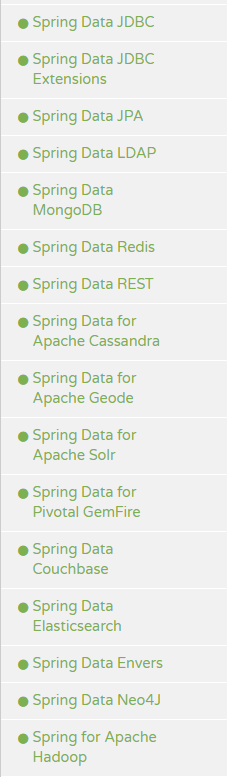
（5）可以集成自定义存储库代码

（6）通过JavaConfig和自定义XML命名空间轻松实现Spring集成

（7）与Spring MVC控制器的高级集成

（8）跨存储持久性的实验支持

## Spring Data包含的子项目



Spring Data为我们使用统一的API来对上述的数据存储技术进行数据访问操作提供了支持。这是Spring通过提供Spring Data Commons项目来实现的，它是上述各种Spring Data项目的依赖，Spring Data Commons让我们在使用关系型或非关系型数据访问技术时都使用基于Spring的统一标准，该标准包含CRUD（创建、获取、更新、删除）、查询、排序和分页的相关操作。

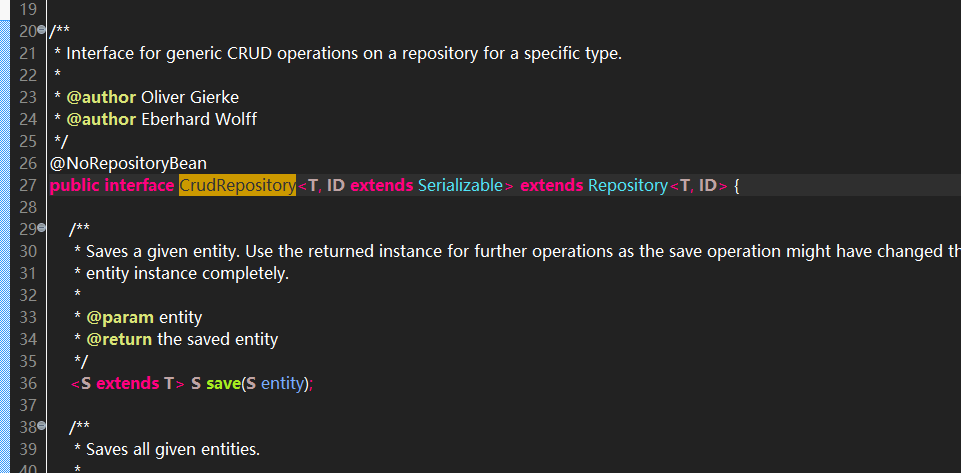
### Spring Data Commons项目概述

Spring Data Commons的一个重要概念：Spring Data Repository抽象，使用Spring Data Repository可以极大的减少数据访问层的代码。Spring Data Repository抽象的根接口是Repository接口，如下：

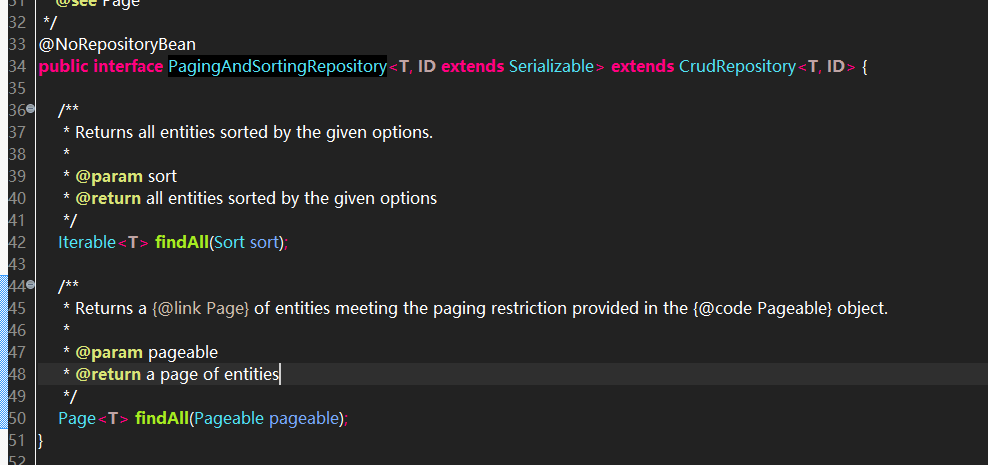
public interface Repository<T,ID extends Serializable>{

}

从源码可以看出，这个接口接受领域类（JPA为实体类）和领域类的id类型作为类型参数，它的子接口CrudRepository定义了和CRUD操作相关的内容,源码如下：



CrudRepository的子接口PagingAndSortingRepository定义了与分页和排序操作相关的内容，源码如下：



不同的数据访问技术也提供了不同的Repository，如Spring Data JPA有JpaRepository、Spring Data MongoDB有MongoRepository。

Spring Data项目还给我们提供了一个激动人心的功能，可以根据属性名进行计数、删除、查询方法等操作。

## Spring Data JPA（重点）

### 概念

JPA是一个基于O/R映射的标准规范，规范是只定义标准规则（如注解、接口），不提供实现，软件提供商可以按照标准规范来实现，而使用者只需按照规范中定义的方法来使用，而不用和软件提供商的实现打交道，JPA的主要实现有Hibernate、OpenJPA等，这也就意味着我们只要使用JPA来开发，无论是哪一个开发方式都是一样的。

Spring Data JPA是更大的Spring Data系列的一部分，可以轻松实现基于JPA的存储库。此模块处理对基于JPA的数据访问层的增强支持，它使构建使用数据访问技术的Spring驱动应用程序变得更加容易。在相当长的一段时间内，实现应用程序的数据访问层一直很麻烦，必须编写太多样板代码来执行简单查询以及执行分页和审计。Spring Data JPA旨在通过减少实际需要的工作量来显著改善数据访问层的实现。作为开发人员，您编写存储库接口，包括自定义查找器方法，Spring将自动提供实现。

### 特性

（1）基于Spring和JPA构建存储库的复杂支持

（2）支持Querydsl谓词，从而支持类型安全的JPA查询

（3）透明审核domain类

（4）分页支持，动态查询执行，集成自定义数据访问代码的能力

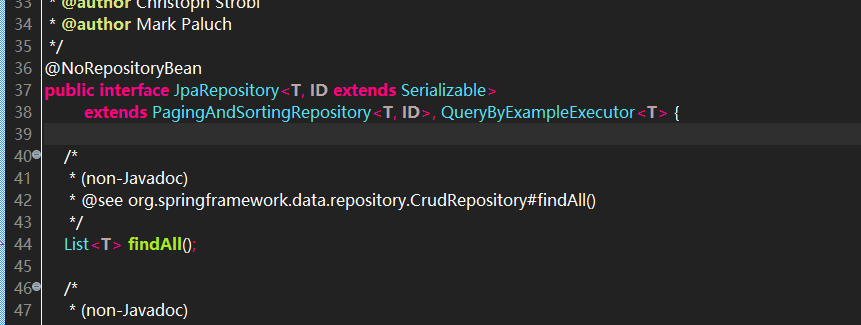
（5）在引导时验证@Query带注释的查询

（6）支持基于XML的实体映射

（7）基于JavaConfig的存储库配置介绍@EnableJpaRepositories

### 定义数据访问层

使用Spring Data JPA建立数据访问层十分简单，只需定义一个继承JpaRepository的接口就可以了，JpaRepository的源码如下：



这个接口已经给我们提供了CRUD的方法。

### Repository<T, ID>接口

中央存储库标记接口，捕获要管理的域类型以及域类型的id类型。 一般用途是保存类型信息，以及能够在类路径扫描期间发现扩展此类信息的接口，以便轻松创建Spring bean。

扩展此接口的域存储库可以通过简单地声明与CrudRepository中声明的签名相同的签名方法来有选择地公开CRUD方法。

参数：

T 存储库管理的域类型

ID 存储库管理的实体的id的类型

### JpaRepository接口

JPA特定的存储库扩展

#### 方法

##### List<T> findAll()

##### List<T> findAll(Sort sort);

##### List<T> findAllById(Iterable<ID> ids);

##### <S extends T> List<S> saveAll(Iterable<S> entities);

##### <S extends T> S saveAndFlush(S entity);

//保存实体并立即刷新更改

##### void deleteInBatch(Iterable<T> entities);

##### void deleteAllInBatch();

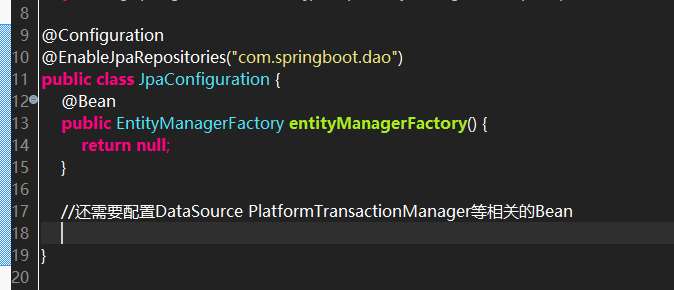
##### T getOne(ID id);

##### <S extends T> List<S> findAll(Example<S> example);

##### <S extends T> List<S> findAll(Example<S> example, Sort sort);

### 配置使用Spring Data Jpa（重点）

在Spring环境中，使用Spring Data Jpa可以通过@EnableJpaRepositories注解开启Spring Data JPA的支持，@EnableJpaRepositories接收的value参数用来扫描数据访问层所在包下的数据访问的接口定义。如下：



#### @EnableJpaRepositories

用于启用JPA存储库的注解。默认情况下将扫描Spring Data存储库的带注释配置类的包。

属性：

value：#basePackages（）属性的别名。 允许更简洁的注释声明。

例如：@ EnableJpaRepositories（“ororg.my.pkg”）而不是@EnableJpaRepositories（basePackages =“org.my.pkg”）

### 定义查询方法

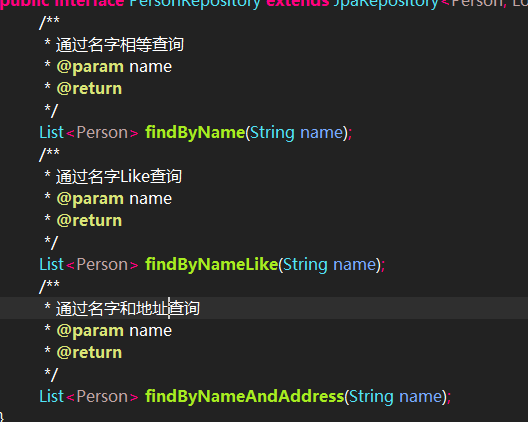
假设我们已经有一张数据表叫PERSON，有ID（Long）、NAME(Varchar2) AGE(Number) ADDRESS(Varchar2)几个字段，对应的实体类叫Person，也有这几个字段。

#### 根据属性名查询

Spring Data Jpa支持通过定义在Repository接口中的方法名来定义查询，而方法名是根据实体类的属性名来确定的。

##### 常规查询

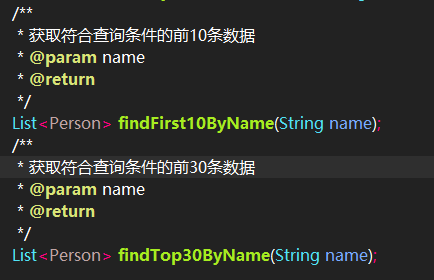
根据属性名来定义查询方法。如下：



从代码可以看出，这里使用了findBy Like And这样的关键字，其中findBy可以用find read readBy queryBy get getBy来代替。

##### 限制结果数量

结果数量是用top和first关键字来实现的，如下：



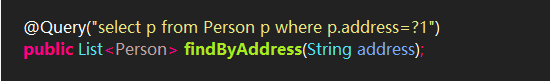
#### 使用JPA的NamedQuery查询

Spring Data JPA支持用JPA的NameQuery来定义查询方法，即一个名字映射一个查询语句，如下：

#### 使用@Query查询

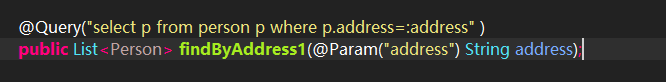
##### 使用参数索引

Spring Data JPA支持用@Qquery注解在接口的方法上实现查询，例如：



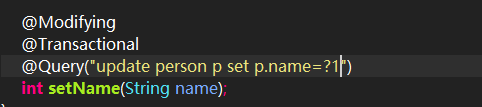
##### 使用命名参数

在Spring Data JPA里还支持在语句里用名称来匹配查询参数，例如：



##### 更新查询

Spring Data JPA支持@Modifying和@Query注解组合来更新查询，例如：



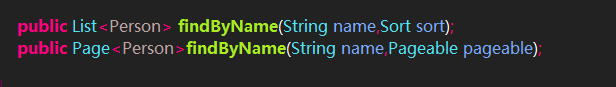
其中返回值int表示更新语句影响的行数。

#### 排序与分页

Spring Data JPA充分考虑了在实际开发中所必须的排序和分页的场景，为我们提供了Sort类以及Page接口和Pageable接口。

##### 1、定义

在接口中定义如下两个方法：



##### 2、使用排序

List<Person> people = personRepository.findByName("xx",new Sort(Direction.ASC,"age"));

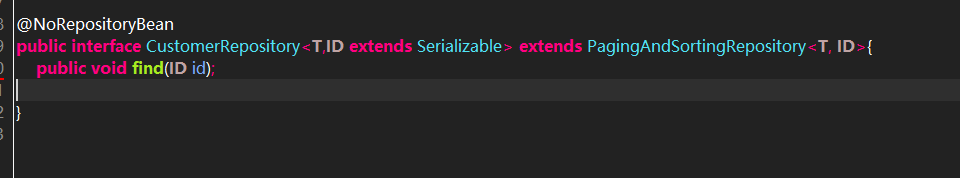
##### 3、使用分页

Page<Person> people2 = personRepository.findByName("xx",new PageRequest(0,10));

其中Page接口可以获得当前页面的记录、总页数、总记录数、是否有上一页或下一页等。

### 自定义Repository的实现

#### 1、自定义Repository接口



#### 2、定义接口实现

#### 3、自定义RepositoryFactoryBean

### Spring Data JPA常用的注解

## SpringBoot的支持

### 基本概述

Spring Framework为使用SQL数据库提供了广泛的支持，从使用JdbcTemplate的直接JDBC访问到完成“对象关系映射”技术（如Hibernate）。 Spring Data提供了更多级别的功能：直接从接口创建存储库实现，并使用约定从方法名称生成查询。

### 配置数据源

Java的javax.sql.DataSource接口提供了一种使用数据库连接的标准方法。 传统上，“DataSource”使用URL和一些凭据来建立数据库连接。

#### 通过Java配置的方式创建数据源

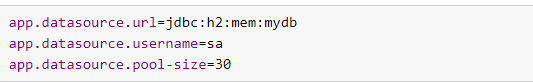
要配置自己的DataSource，请在配置类中定义该类型的@Bean。 Spring Boot在任何需要的地方重用您的DataSource，包括数据库初始化。 如果需要外部化某些设置，可以将DataSource绑定到环境中。

##### 代码示例

###### 定义一个类，在类中配置数据源



###### 在配置文件中定义数据源的配置



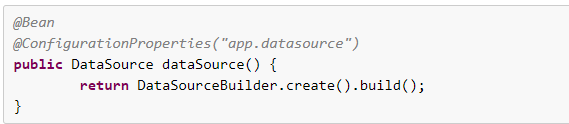
假设您的FancyDataSource具有URL的常规JavaBean属性，用户名和池大小，这些设置在DataSource可供其他组件使用之前自动绑定。常规数据库初始化也会发生（因此spring.datasource。\*的相关子集仍然可以与您的自定义配置一起使用）

#### 通过DataSourceBuilder创建数据源

Spring Boot还提供了一个名为DataSourceBuilder的实用程序构建器类，可用于创建其中一个标准数据源（如果它位于类路径中）。 构建器可以根据类路径上的可用内容检测要使用的那个，它还会根据JDBC URL自动检测驱动程序。

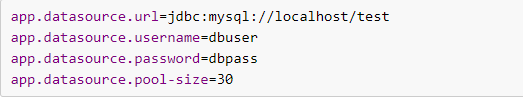
##### 代码示例

###### 在类中配置数据源

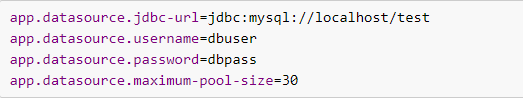


要使用该DataSource运行应用程序，您只需要连接信息。 还可以提供特定于池的设置。 检查将在运行时使用的实现以获取更多详细信息。

###### 在配置文件中定义一个JDBC数据源属性

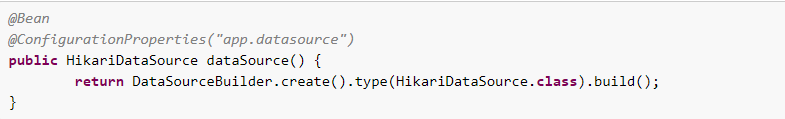


然而，有一个问题。 由于未公开连接池的实际类型，因此在自定义DataSource的元数据中不会生成任何键，并且IDE中没有可用的完成（因为DataSource接口不公开任何属性）。 此外，如果你碰巧在类路径上有Hikari，这个基本设置不起作用，因为Hikari没有url属性（但是有一个jdbcUrl属性）。 在这种情况下，您必须按如下方式重写配置：

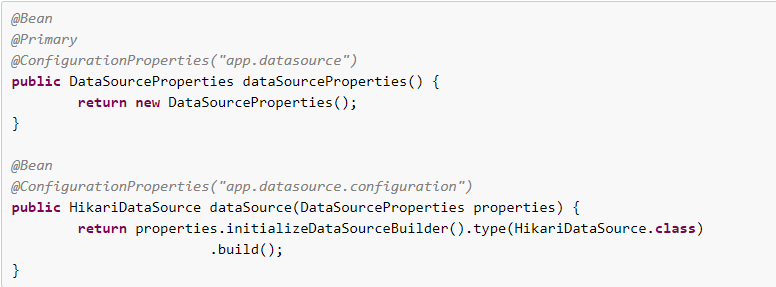


您可以通过强制连接池使用并返回专用实现而不是DataSource来解决此问题。 您无法在运行时更改实现，但选项列表将是显式的。

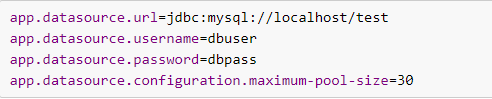
使用DataSourceBuilder创建一个HikariDataSource数据源



您甚至可以通过利用DataSourceProperties为您做的事情来进一步发展 - 也就是说，如果没有提供URL，则通过提供具有合理用户名和密码的默认嵌入式数据库。 您可以从任何DataSourceProperties对象的状态轻松初始化DataSourceBuilder，因此您也可以注入Spring Boot自动创建的DataSource。 但是，这会将您的配置拆分为两个名称空间：spring.datasource上的url，username，password，type和driver，以及自定义名称空间（app.datasource）上的其余名称空间。 为避免这种情况，您可以在自定义命名空间上重新定义自定义DataSourceProperties，如以下示例所示：



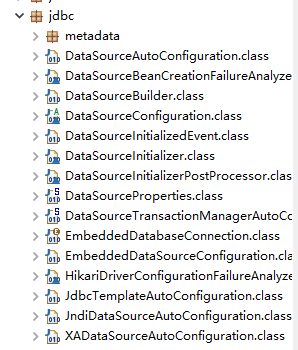
默认情况下，此设置使您与Spring Boot为您执行的操作保持同步，除了选择了专用连接池（在代码中）并且其设置在app.datasource.configuration子命名空间中公开。 因为DataSourceProperties正在为您处理url / jdbcUrl转换，所以您可以按如下方式对其进行配置：



Spring Boot会将Hikari特定的设置暴露给spring.datasource.hikari。 此示例使用更通用的配置子命名空间，因为该示例不支持多个数据源实现。由于您的自定义配置选择使用Hikari，因此app.datasource.type无效。 在实践中，构建器初始化为您可能在那里设置的任何值，然后通过调用.type（）来覆盖

### JDBC的自动配置

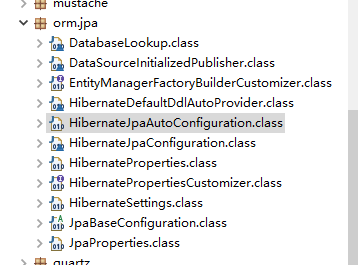
spring-boot-starter-data-jpa依赖于spring-boot-starter-jdbc，而Spring Boot对JDBC做了一些自动配置，源码在org.springframework.boot.autoconfigure.jdbc包下，如图：



从源码分析可以看出，我们通过"spring.datasource"为前缀的属性自动配置dataSource，Springboot自动开启了注解事务的支持（@EnableTransactionManagerment），还配置了一个jdbcTemplate。Springboot还提供了一个初始化数据的功能，放置在类路径下的schema.sql文件会自动用来初始化表结构，放置在类路径下的data.sql文件会自动用来填充表数据。

### 对JPA的自动配置

Springboot对JPA的自动配置放置在org.springframework.boot.autoconfigure.orm.jpa下，如图：



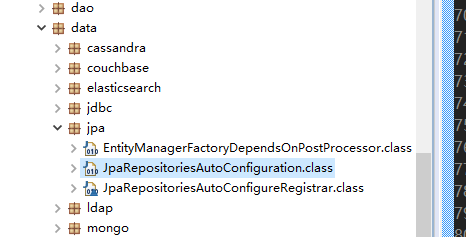
Springboot默认JPA的实现者是Hibernate，HibernateJpaAutoConfiguration依赖于DataSourceAutoConfiguration。

从JpaProperties的源码可以看出，配置JPA可以使用spring.jpa为前缀的属性在application.properties中配置。

从JpaBaseConfiguration的源码可以看出，Springboot为我们配置了transactionManager jpaVendorAdapter、entityManagerFactory等Bean。JpaBaseConfiguration还有一个getPackagesToScan方法，可以自动扫描注解有@Entity的实体类。

### 对Spring Data JPA的自动配置

Springboot对Spring Data JPA的自动配置放置在org.springframework.boot.autoconfigure.data.jpa下，如图：



从JpaRepositoriesAutoConfiguration和JpaRepositoriesAutoConfigureRegistrar源码可以看出，JpaRepositoriesAutoConfiguration是依赖于HibernateJpaAutoConfiguration配置的，且Springboot自动开启了对Spring Data JPA的支持，我们无须在配置类显示声明@EnableJpaRepositories。

### Springboot下的Spring Data JPA

我们在Springboot下使用Spring Data JPA，在项目的maven依赖里添加spring-boot-starter-data-jpa，然后只需定义DataSource、实体类和数据访问层，并在需要使用数据访问的地方注入数据访问层的Bean即可，无须任何额外配置。

## 数据缓存Cache

一个程序的瓶颈在于数据库，我们知道内存的速度是大大快于硬盘的速度的，当我们需要重复的获取相同的数据的时候，我们一次次的请求数据库或远程服务，导致大量的时间耗费在数据库查询或者远程方法调用上，导致程序性能的恶化，这便是数据缓存要解决的问题。

### Spring缓存支持

Spring定义了org.springframework.cache.CacheManager和org.springframework.cache.Cache接口用来统一不同的缓存的技术。其中，CacheManager是Spring提供的各种缓存技术抽象接口，Cache接口包含缓存的各种操作（增加、删除、获得缓存，我们一般不会直接和此接口打交道）

#### Spring支持的CacheManager

针对不同的缓存技术，需要实现不同的CacheManager，Spring定义了如下CacheManager的实现。

SimpleCacheManager：使用简单的Collection来存储缓存，主要用来测试用途。

ConcurrentMapCacheManager：使用ConcurrentMap来存储缓存

NoOpCacheManager：仅测试用途，不会实际存储缓存

EhCacheCacheManager：使用EhCache作为缓存技术

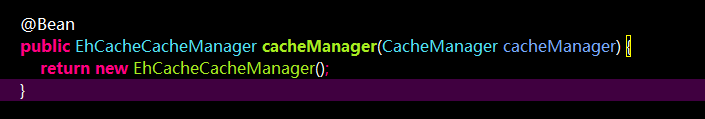
GuavaCacheManager：使用Google Guava的GuavaCache作为缓存技术

HazelcastCacheManager：使用Hazelcast作为缓存技术

JcacheCacheManager：使用Jcache标准的实现作为缓存技术，如Apache Commons JCS

RedisCacheManager：使用Redis作为缓存技术

在我们使用任意一个实现的CacheManager的时候，需要注册实现的CacheManager的Bean，例如：



当然每种缓存技术都有很多的额外配置，但是配置cacheManager是必不可少的。

#### 声明式缓存注解

Spring提供了4个注解来声明缓存规则（又是使用注解式AOP的一个生动例子），四个注解如下：

@Cacheable 在方法执行前Spring先查看缓存中是否有数据，如果有数据，则直接返回缓存数据，若没有数据，调用方法并将方法返回值放进缓存。

@CachePut 无论怎样，都会将方法的返回值放到缓存中，@CachePut的属性与@Cacheable保持一致。

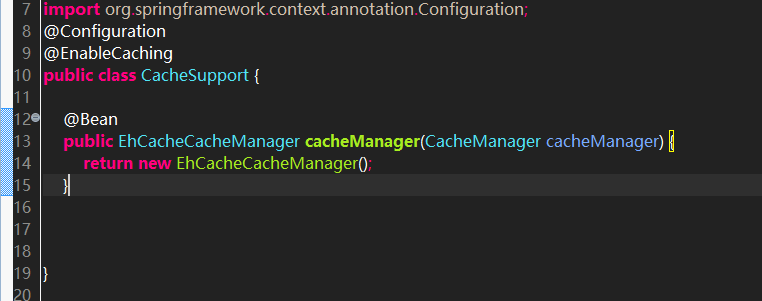
@CacheEvict 将一条或多条数据从缓存中删除

@Caching 可以通过@Caching注解组合多个注解策略在一个方法上

@Cacheable @CachePut @CacheEvict都有value属性，指定的是要使用的缓存名称，key属性指定的是数据在缓存中的存储的键。

#### 开启声明式缓存支持

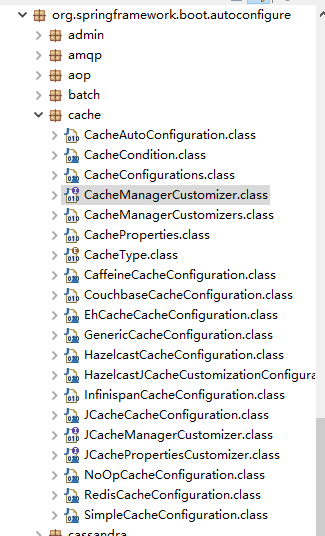
开启声明式缓存支持十分简单，只需要在配置类上使用@EnableCaching注解即可，例如：



### Springboot的支持

在Spring中使用缓存技术的关键是配置CachaManager，而Springboot为我们自动配置了多个CacheManager的实现。

Springboot的CacheManager的自动配置放置在org.springframework.boot.autoconfigure.cache包中，如图：



在不做任何额外配置的情况下，默认使用的是SimpleCacheConfiguration，即使用ConcurrentMapCacheManager，Springboot支持以"spring.cache"为前缀的属性来配置缓存。

在Springboot环境下，使用缓存技术只需在项目中导入相关缓存技术的依赖包，并在配置类使用@EnableCaching开启缓存支持即可。

### 代码示例

将以Springboot默认的ConcurrentMapCacheManager作为缓存技术，演示@Cacheable @CachePut @CacheEvit

#### 1、添加依赖

# Spring boot开发部署与测试

## 开发的热部署

### （1）模板热部署

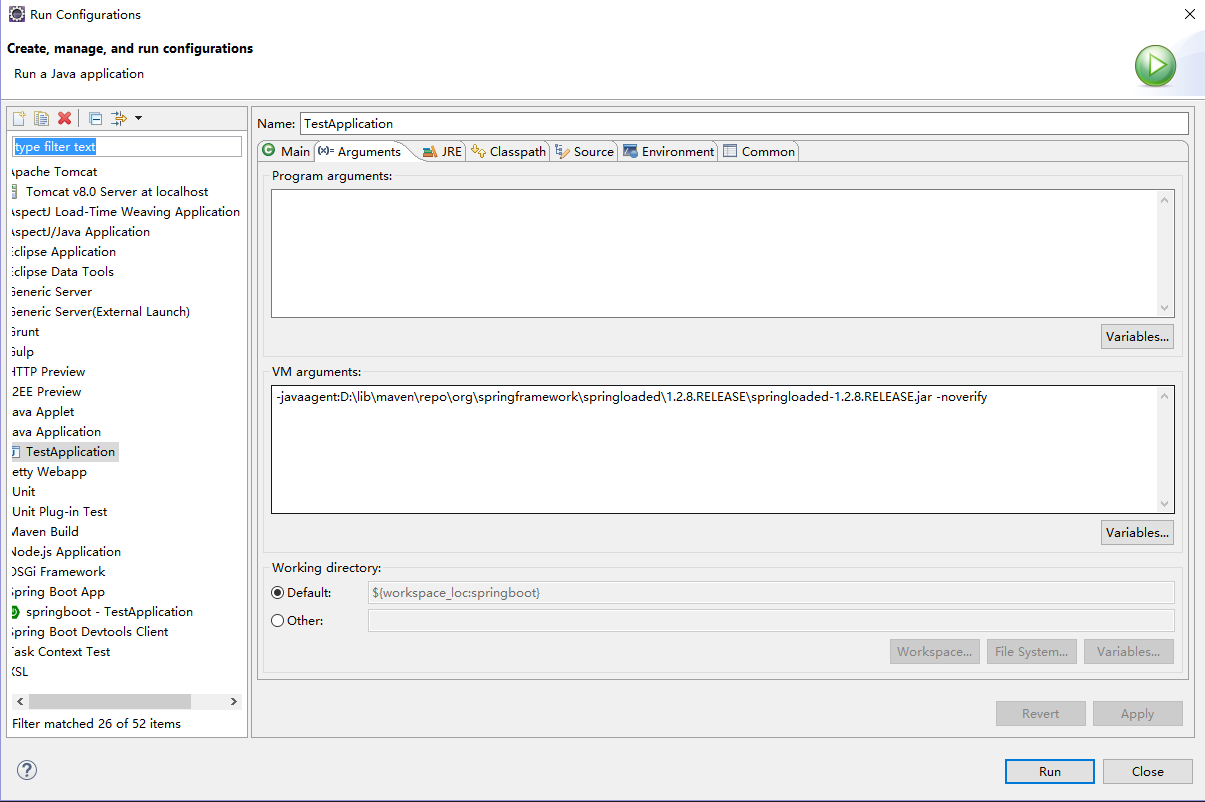
在Springboot里，模板引擎的页面默认是开启缓存的，如果修改了页面的内容，则刷新页面是得不到修改后的页面的，我们可以在application.properties中关闭模板引擎的缓存，例如：

Thymeleaf的配置：

spring.thymeleaf.cache=false

### （2）Spring Loaded

Spring loaded可实现修改类文件的热部署，下载地址在maven仓库中搜索就可以。安装单击：run Configurations…，在arguments标签页的vm arguments中填入如下内容，注意下面指定的springloaded的路径。



### （3）spring-boot-devtools（推荐这个）

在Springboot项目中添加spring-boot-devtools依赖即可实现页面和代码的热部署。

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

//<optional>true</optional> 依赖不会传递，该项目依赖devtools，之后依赖boot项目的项目如果想要使用devtools，需要重新引入。

</dependency>

说明：直接加上这个依赖就可以了，其他的什么都不用修改，该模块在完整的打包环境下运行的时候会被禁用。如果你使用java -jar启动应用或者用一个特定的classloader启动，它会认为这是一个“生产环境”。

devtools可以实现页面热部署，即页面修改后会立即生效，这个可以直接在application.properties文件中配置spring.thymeleaf.cache=false来实现。实现类文件热部署（类文件修改后不会立即生效），实现对属性文件的热部署。

devtools会监听classpath下的文件变动，并且会立即重启应用。因为采用了虚拟机机制，该项重启是很快的。重启的时候只是加载了在开发的class，没有加载第三方的jar包。

### （4）Jrebel（不采用）

Jrebel是Java开发热部署的最佳工具，对springboot也提供了极佳的支持。Jrebel是收费软件，试用期14天。使用方法：

1、安装，在eclipse marketplace中搜索安装。（不截图了）

2、配合使用，注册使用。（不截图了）

3、选择springboot项目，右键，选择Jrebel-Add Jrebel Nature，此时我们增加了一个rebel.xml文件，用来配置热部署内容，这个xml文件中会做说明。

4、首次启动会询问是否以Jrebel启动程序，当启动时出现和Jrebel相关的信息时，表明配置成功。

## 常规部署

### jar形式

#### （1）打包

若我们在新建springboot项目的时候，选择打包方式(packaging)是jar时，则我们需要用：mvn package。

#### （2）运行

在命令行中执行命令：

java –jar xx.jar

### war形式

#### 1、打包方式为war时

新建springboot项目时可选择打包方式是war的形式，或者修改pom.xml文件：

<packaging>war</packaging>

打包的方式和jar包一致：

mvn package

最后将生成的war文件放在你喜欢的Servlet容器上运行就可以。

#### 2、打包方式为jar时

如果我们新建springboot项目时选择打包方式选择的是jar，部署时我们又想要用war包形式部署，那么怎么将jar形式修改为war形式呢？只需要修改pom.xml文件即可。

<packaging>war</packaging>

增加下面依赖覆盖默认内嵌的Tomcat依赖：

<!-- 覆盖内置的tomcat容器 -->

<dependency>

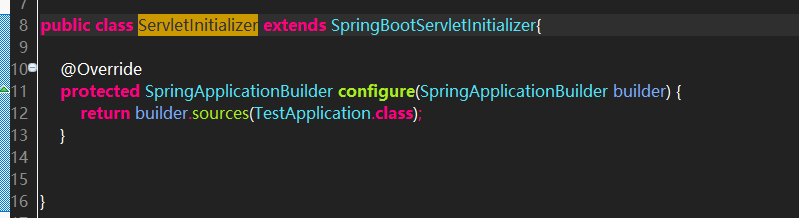
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>

增加ServletInitializer类，内容如下：



说明：TestApplication类是Springboot的启动类。

## Spring boot测试

Springboot测试和SpringMVC测试类似，Springboot为我们提供了一个@SpringApplicationConfiguration来替代@ContextConfiguration，用来配置ApplicationContext。在Springboot中，每次新建项目都会自动加上spring-boot-starter-test的依赖，这样我们就没有必要测试时再添加额外的jar包。Springboot还会新建一个当前项目的测试类，位于src/test/java的根包下。

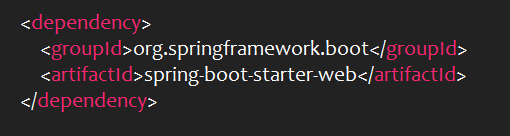
在新版本的springboot框架中，@SpringApplicationConfiguration这个注解已经取消了，替换为@SpringBootTest这个注解。

打开src/test/java下的测试入口，编写简单的http请求来测试；使用mockmvc进行，利用MockMvcResultHandlers.print()打印出执行结果。

# Springboot开发邮件功能

https://www.cnblogs.com/studyDetail/p/7007979.html

## 引入依赖



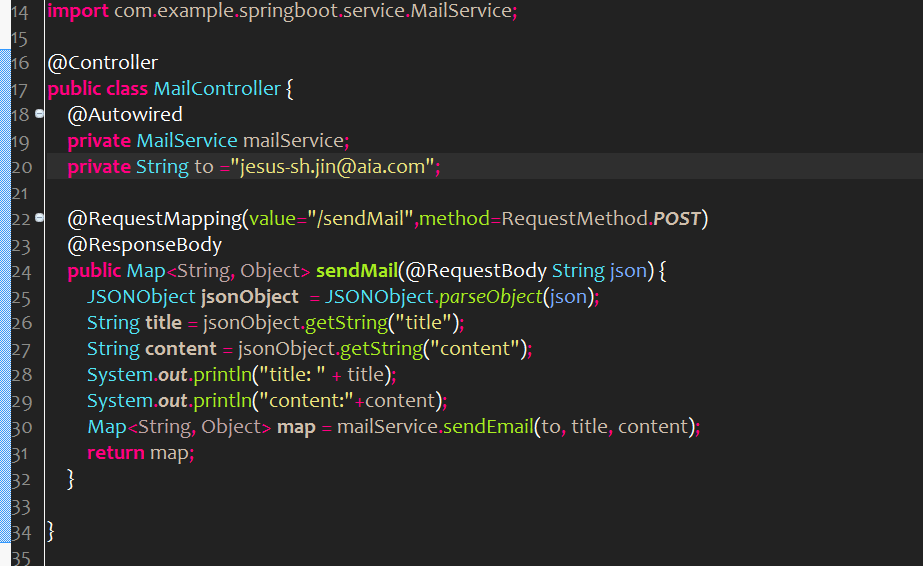
## 参数配置

在application.properties中配置邮件相关的参数

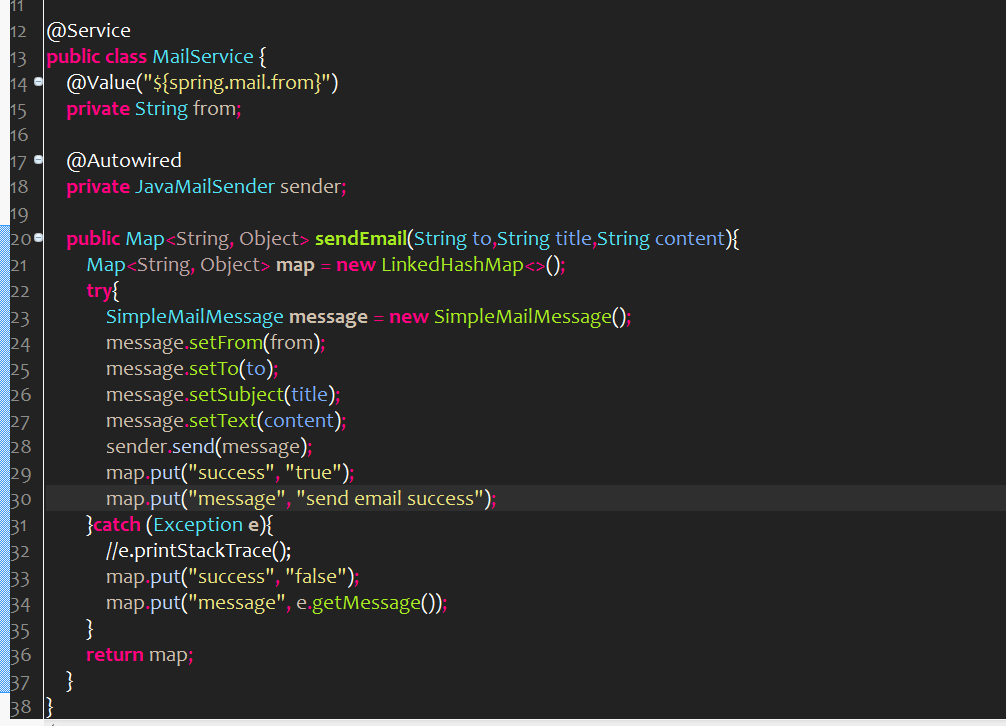
spring.mail.host=smtpsg-int.aia.biz //配置邮件的host

spring.mail.from=ann@aia.com //配置邮件的sender

## 编写Controller



## 编写Service



## 测试

打开postman，输入接口地址：

http://localhost:8082/springboot/sendMail

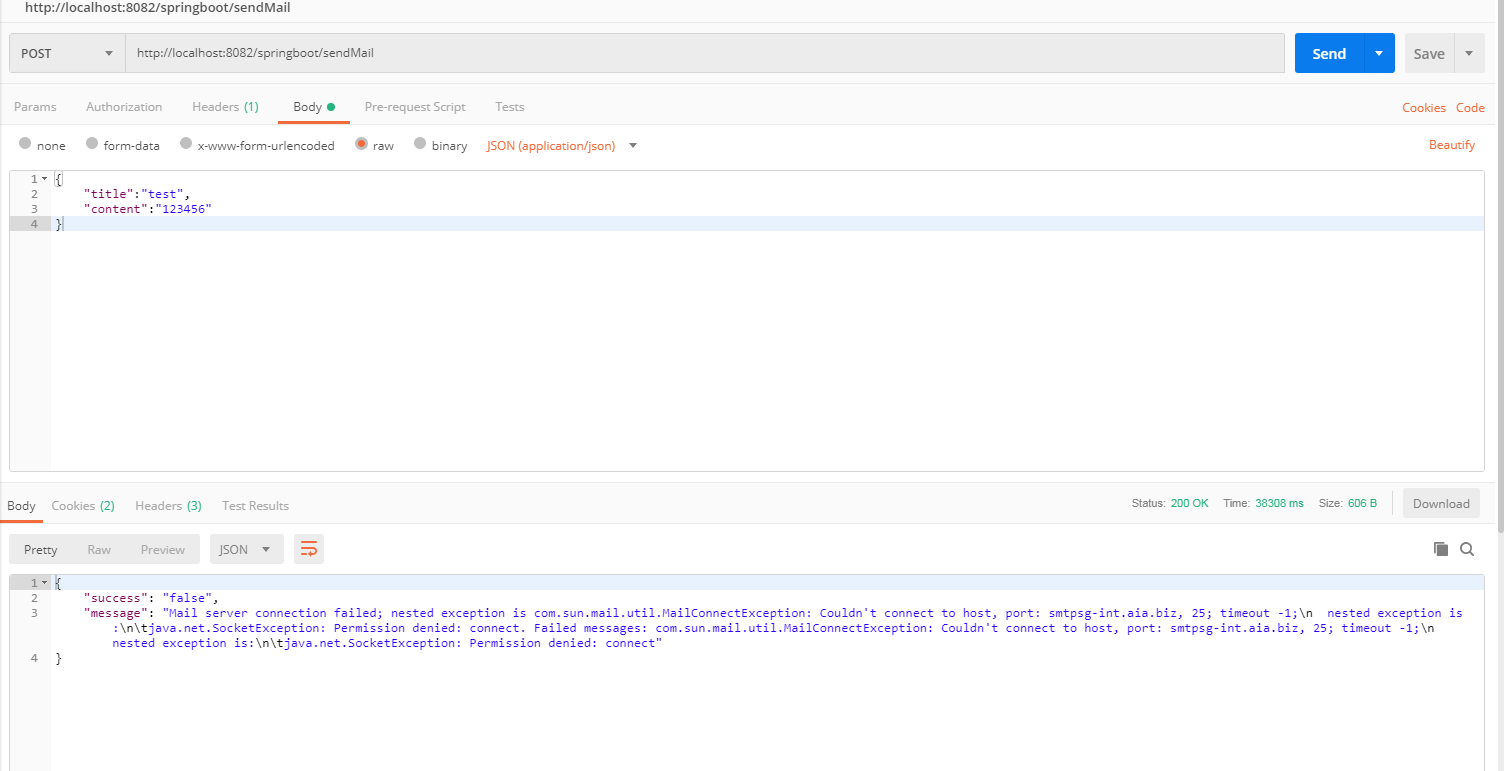
json串：

{

"title":"test",

"content":"123456"

}



## SimpleMailMessage类

为简单邮件消息建模，包括from，to，cc，subject和text等字段。考虑JavaMailSender和JavaMail MimeMessages来创建更复杂的消息，例如带有附件的消息，特殊字符编码或伴随邮件地址的个人名称。

## JavaMailSender接口

扩展了用于JavaMail的org.springframework.mail.MailSender接口，支持MIME消息作为直接参数和准备回调。 通常与MimeMessageHelper类一起使用，以便于创建JavaMail MimeMessages，包括附件等。

### 方法

#### void send(MimeMessage mimeMessage)

发送给定的JavaMail MIME消息。

参数：要发送的mimeMessage对象

# .md文件

## 基本概念

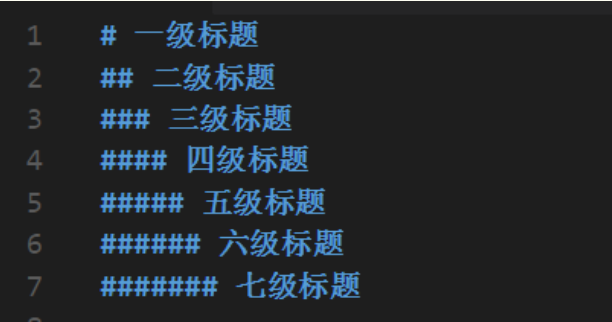
Markdown 是一种轻量级标记语言，创始人为约翰·格鲁伯（John Gruber）。它允许人们“使用易读易写的纯文本格式编写文档，然后转换成有效的XHTML(或者HTML)文档”。这种语言吸收了很多在电子邮件中已有的纯文本标记的特性。

Markdown 的目标是实现「易读易写」。可读性，无论如何，都是最重要的。一份使用 Markdown 格式撰写的文件应该可以直接以纯文本发布，并且看起来不会像是由许多标签或是格式指令所构成。Markdown 语法受到一些既有 text-to-HTML 格式的影响，包括Setext、atx、Textile、reStructuredText、Grutatext 和 EtText，而最大灵感来源其实是纯文本电子邮件的格式。总之， Markdown 的语法全由一些符号所组成，这些符号经过精挑细选，其作用一目了然。比如：在文字两旁加上星号，看起来就像\*强调\*。Markdown 的列表看起来，嗯，就是列表。Markdown 的区块引用看起来就真的像是引用一段文字，就像你曾在电子邮件中见过的那样。

Markdown 语法的目标是：成为一种适用于网络的书写语言。Markdown 不是想要取代 HTML，甚至也没有要和它相近，它的语法种类很少，只对应 HTML 标记的一小部分。Markdown 的构想不是要使得 HTML 文档更容易书写。在我看来， HTML 已经很容易写了。Markdown 的理念是，能让文档更容易读、写和随意改。HTML 是一种发布的格式，Markdown 是一种书写的格式。就这样，Markdown 的格式语法只涵盖纯文本可以涵盖的范围。

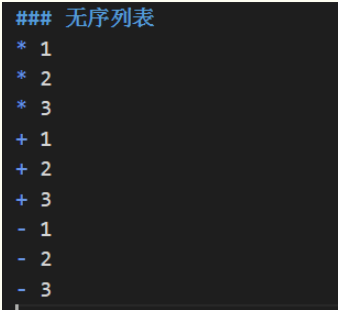
## 语法

### 标题的写法

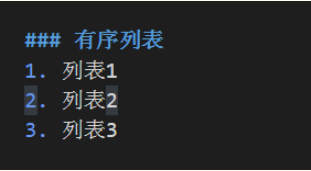


前面带#号，后面带文字，分别表示h1-h6,上图可以看出，注意，#号后面有空格

### 列表



可以看到，无序列表可以用\* ， + ， — 来创建，用在线编辑器看，实际上是转换成了ul>li ，所以使用哪个都可以，推荐使用\*吧



# 遇到的问题

## 1、The method onStartup(ServletContext) of type Web must override a superclass method

解决方法：

修改jdk的版本，window ->preferences... -> java -> Compiler

Compiler compliance level : 6.0，项目右键->build path->configure build path->java Compiler(左边那排中) ->在右边的Compiler compliance level 修改版本为6.0。（修改jdk的版本根据你本机安装的jdk版本为参考）

## 2、Unable to find a single main class from the following candidates

解决方法：

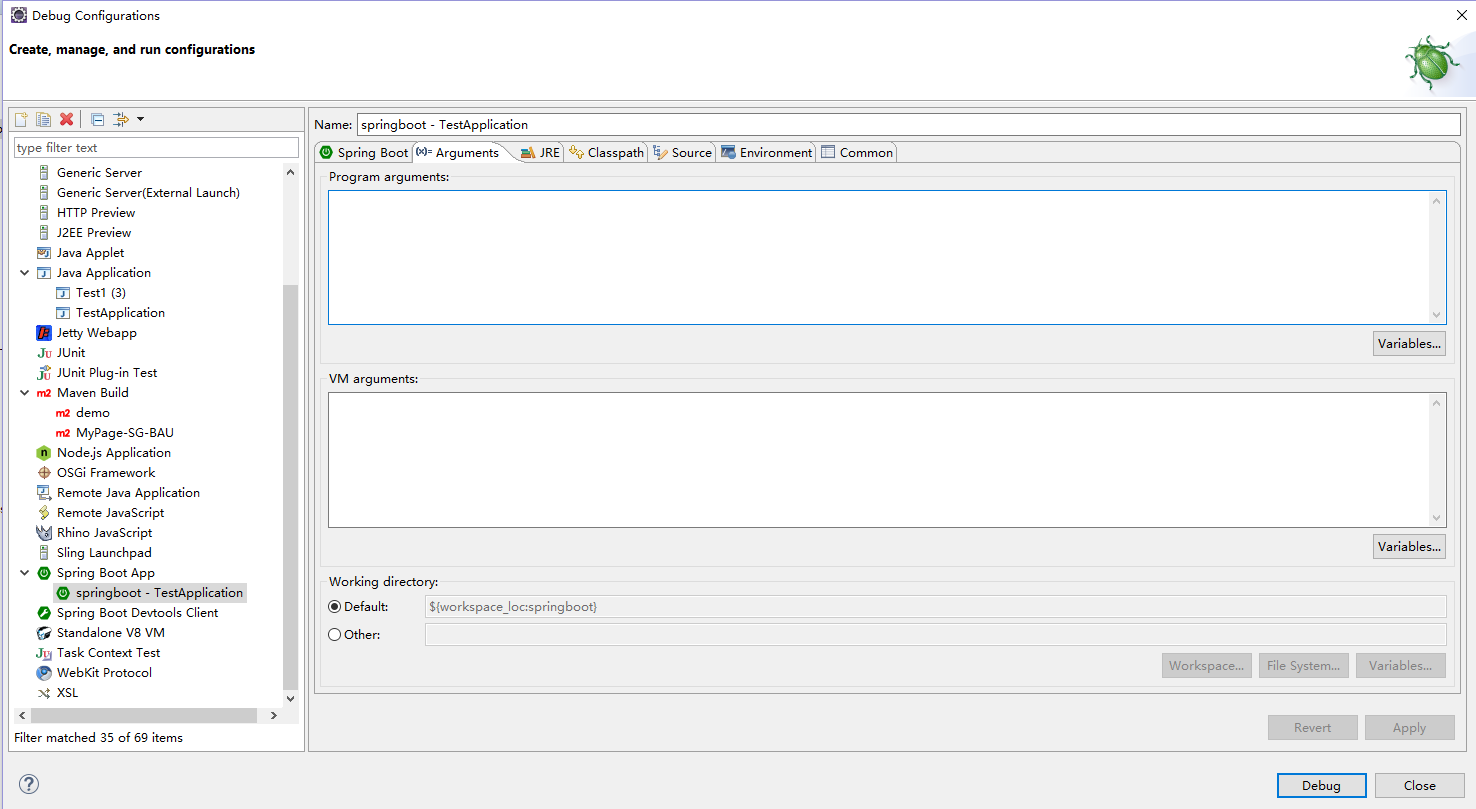
这个问题是springboot不能找到唯一的入口类，项目中有多个入口类，把其他多余的入口类删除，留下一个唯一的入口类就可以。

## 3、错误: 找不到或无法加载主类

解决方法：更新一下maven项目就可以

## 4、Reference to undefined variable jrebel\_args

在springboot项目的配置中，VM arguments中删除jrebel\_args变量即可。



## 5、[Cannot determine embedded database driver class for database type NONE](https://www.cnblogs.com/yql1986/p/6819097.html)

这是因为spring boot默认会加载org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration类，DataSourceAutoConfiguration类使用了@Configuration注解向spring注入了dataSource bean。因为工程中没有关于dataSource相关的配置信息，当spring创建dataSource bean因缺少相关的信息就会报错。因为我仅仅只是使用spring boot来写一些很简单的例子来学习它，在Application类上增加@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

阻止spring boot自动注入dataSource bean。

## 6、Consider revisiting the conditions above or defining a bean of type 'javax.sql.DataSource' in your configuration.

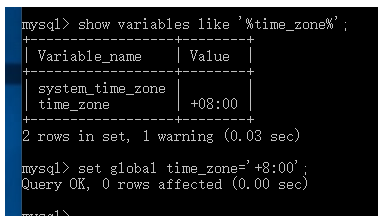
看了网上的添加@EnableAutoConfiguration(exclude = {DataSourceAutoConfiguration.class})解决方法都没有用，然后在pom.xml文件中把和数据库相关的依赖删除了就可以了。

## 7、The server time zone value 'ÖÐ¹ú±ê×¼Ê±¼ä' is unrecognized or represents more than one time zone.

解决方法：执行下面的命令即可解决。

SHOW VARIABLES LIKE '%time\_zone%'

SET GLOBAL time\_zone='+8:00'



## 8、Whitelabel Error Page 404

解决方法：可能的原因是请求路径不对，解析不对，注解忘记引入等。另外，对于springboot项目，\*Application这个类要放在groupId对应的包下，其他自己新建的包都应该是它的一个子包。注意，目录结构的关联。

## 9、Unregistering JMX-exposed beans on shutdown

这个问题是在使用了springboot1.5.9版本启动时报的错，导致无法启动springboot应用程序，我们需要修改springboot的版本为1.5.10即可。