# 1 xml 或 注解 怎么选择？

应用基本配置：数据源/资源文件🡪XML配置

DI注入：注解

spring 3.x ：使用java配置取代xml

## 1.1 spring java配置方式

### 1.1.1 spring的java配置方式主要通过@Configuration 和 @Bean 这两个注解实现：

@Configuration 作用于类上，相当于一个xml配置文件

@Bean 作用于方法上，相当于xml配置中的<bean>

其他注解：

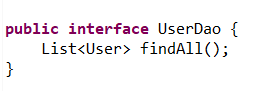
@ComponentScan 组件扫描

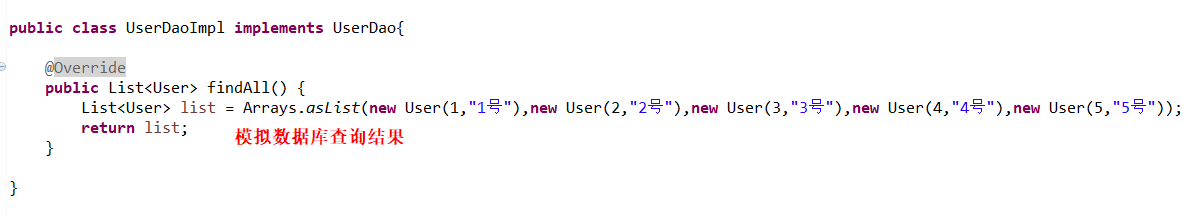
@PropertySource 引入外部资源文件

@Value 赋值外部资源文件的值

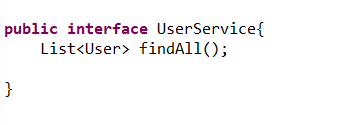
### 1.1.2 示例

#### 1.1.2.1 dao





#### 1.1.2.2 service

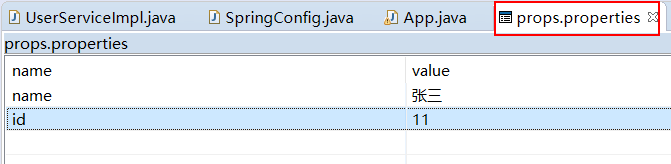




#### 1.1.2.3 配置代码

#### 

外部属性文件内容：



#### 1.1.2.4 测试文件



# 2 springboot

## 2.1 什么是springboot ？

### 2.1.1 静态语言和动态语言？

静态语言需要先编译，在运行。如：java

动态语言，不需要编译，直接运行。如：JavaScript/Node.js

### 2.1.2 什么是springboot ？

随着动态语言的流行，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。

1 “习惯优于配置”理念，springboot中内置了一些习惯性的配置，可以直接使用，从而简化配置、使项目更快的运行起来。

2 springboot内嵌了一个servlet容器（tomcat），可以很容易的创建一个独立运行的项目。

3 总之，使用springboot可以不用或者只需要很少的spring配置。

### 2.1.3 springboot优缺点：

1、简化配置，可以快速构建项目，极大提高项目的开发效率、部署效率；

2、由于内置servlet容器，所以项目可独立运行，无须依赖外部servlet容器。

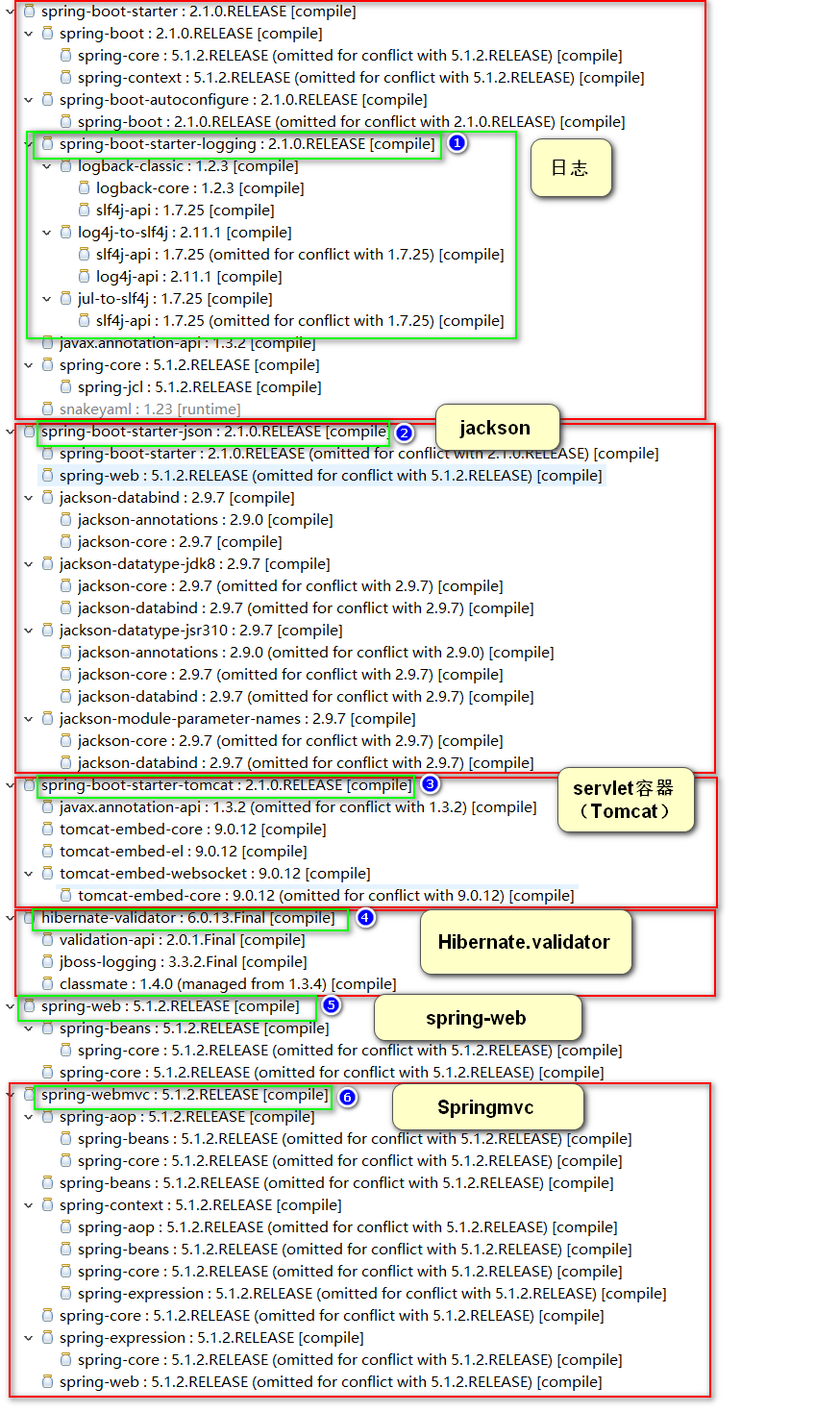
3、对主流开发框架能无配置集成。

4、提供了运行时的应用监控。

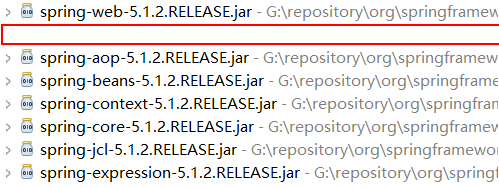
5、与云计算的天然集成。

## 2.2 spring-boot 的pom.xml配置

### 2.2.1 spring-boot 已经集成了的东西，不需要再配置了。



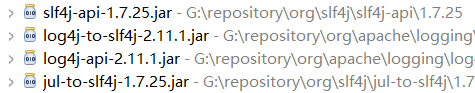
1 spring 基础包



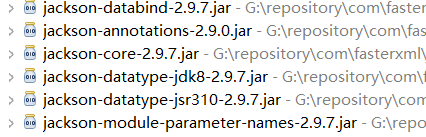
2 springmvc

3 servlet容器（Tomcat）

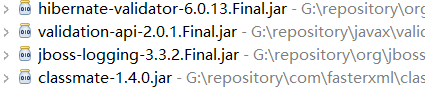
4 log4j 2.11



5 Jackson



6数据验证（Hibernate.validator）



### 2.2.2 需要配置的东西：

必须配置：

spring-boot-starter-parent(它的版本号可以确定其他文件的版本号，所以不需要再给别的依赖添加版本号)

spring-boot-starter-web

可选配置：

spring-boot-maven-plugin

### 2.2.3 简单spring-boot配置示例

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<!-- spring-boot-starter-parent -->

<parent>

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

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

<version>2.1.0.RELEASE</version>

</parent>

<groupId>cn</groupId>

<artifactId>springboot</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>springboot Maven Webapp</name>

<url>http://maven.apache.org</url>

<dependencies>

<!-- spring-boot-starter-web 支持 -->

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

</dependency>

</dependencies>

<build>

<finalName>springboot</finalName>

<plugins>

<!-- 资源文件拷贝插件 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-resources-plugin</artifactId>

<configuration>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<!-- java编译插件 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<!-- spring-boot 插件 -->

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

## 2.3 spring-boot Hello Spring-boot示例代码

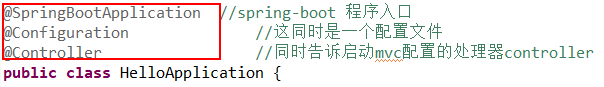
### 2.3.1配置文件：见2.2.3

### 2.3.2 示例代码



## 2.4 springboot核心内容

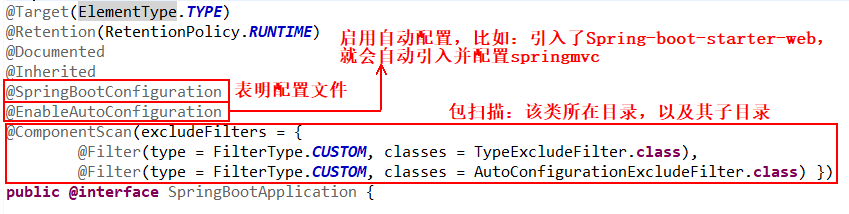
### 2.4.1 入口类



重要的注解：

#### 2.4.1.1 @SpringBootApplication

该注解是一个组合注解：

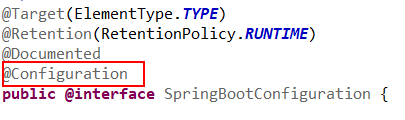


#### 2.4.1.2着重关注

SpringBootConfiguration 、 EnableAutoConfiguration 、ComponentSacn

##### 1 SpringBootConfiguration

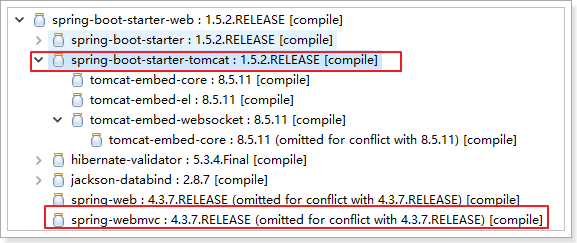
也是一个组合注解，其中包括了Configuration.官方推荐使用SpringBootConfiguration代替Configuration



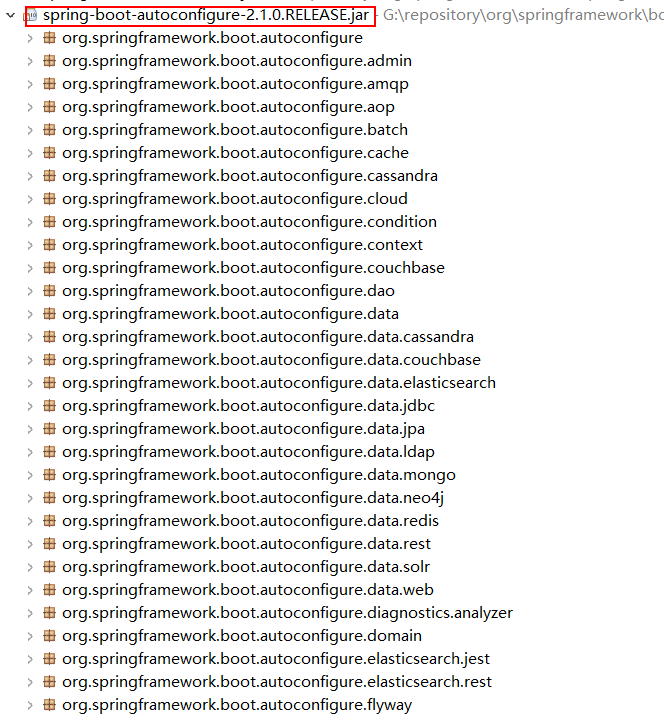
##### 2 EnableAutoConfiguration

###### 2.1 启用自动配置：

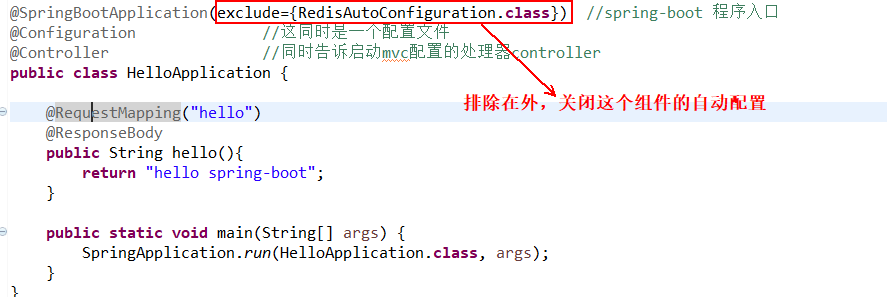
该注解会使Spring Boot根据项目中依赖的jar包自动配置项目的配置项。

如：我们添加了spring-boot-starter-web的依赖，项目中也就会引入SpringMVC的依赖，Spring Boot就会自动配置tomcat和SpringMVC。  


如下，引入相关包后就会自动配置：



###### 2.2关闭自动配置



##### 3 ComponentSacn

包扫描，也可以自己配置。

### 2.4.2 全局配置

#### 2.4.2.1 配置文件

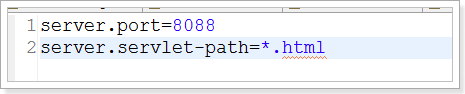
springboot 项目使用一个全局的配置文件，叫做application.properties 或者application.yml。放置在resources目录下，或者类路径下的/config下，一般放在resources下。

#### 2.4.2.2 文件内容示例

1 修改tomcat 的端口为8088

server.port=8088

2 修改springmvc路径后缀为.html（貌似有问题）



@Bean

**public** ServletRegistrationBean dispatcherRegistration(DispatcherServlet dispatcherServlet) {

ServletRegistrationBean reg = **new** ServletRegistrationBean(dispatcherServlet);

reg.getUrlMappings().clear();

reg.addUrlMappings("\*.html");

reg.addUrlMappings("\*.do");

**return** reg;

}

#### 2.4.2.3 详细配置内容如下：

[连接](application.properties.docx)

### 2.4.3 xml配置



### 2.4.4 日志

Spring Boot对各种日志框架都做了支持，我们可以通过配置来修改默认的日志的配置：

#设置日志级别

logging.level.org.springframework=DEBUG

格式：

logging.level.\*= *# Log levels severity mapping. For instance `logging.level.org.springframework=DEBUG`*

# 3 springboot自动配置原理

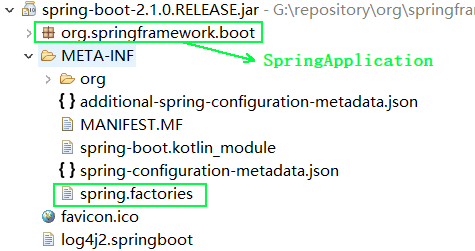
3.1maven获取源码

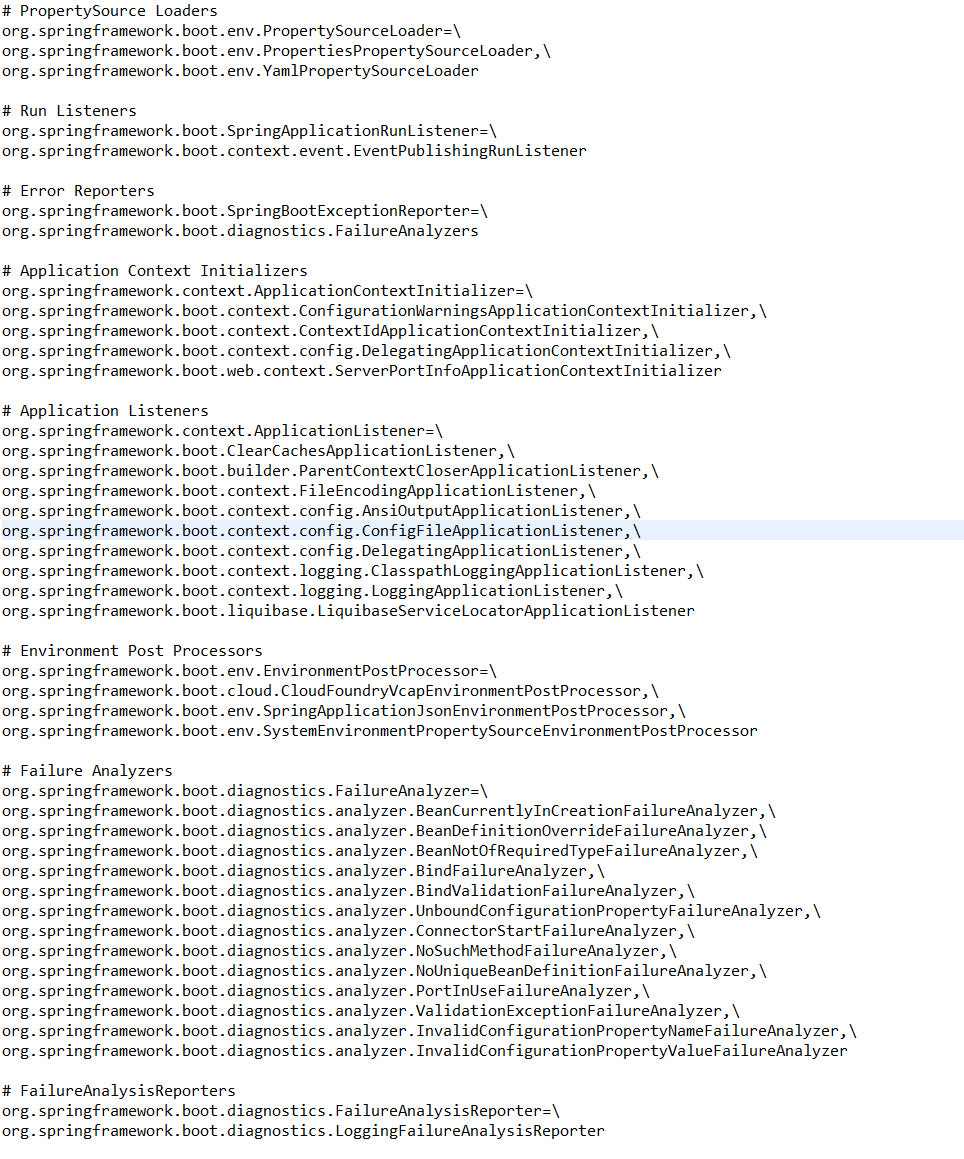
通过dependency:sources 可以下载所有依赖的源码

3.2 源码分析：

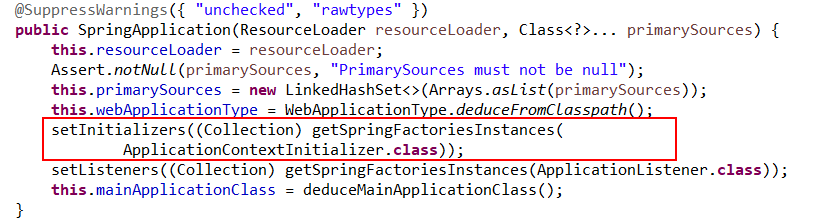
关键类：SpringApplication SpringFactoriesLoader

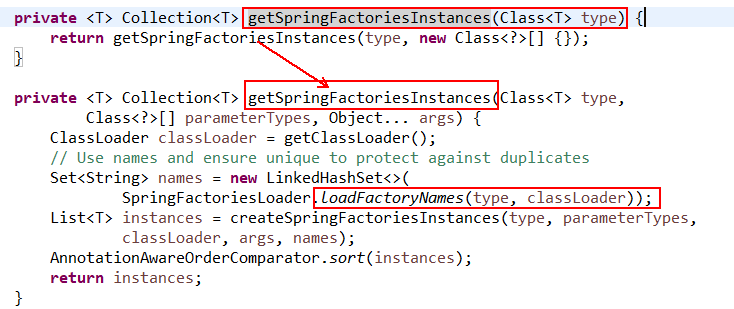
关键文件：spring.factories 该文件中有很多的类全限名



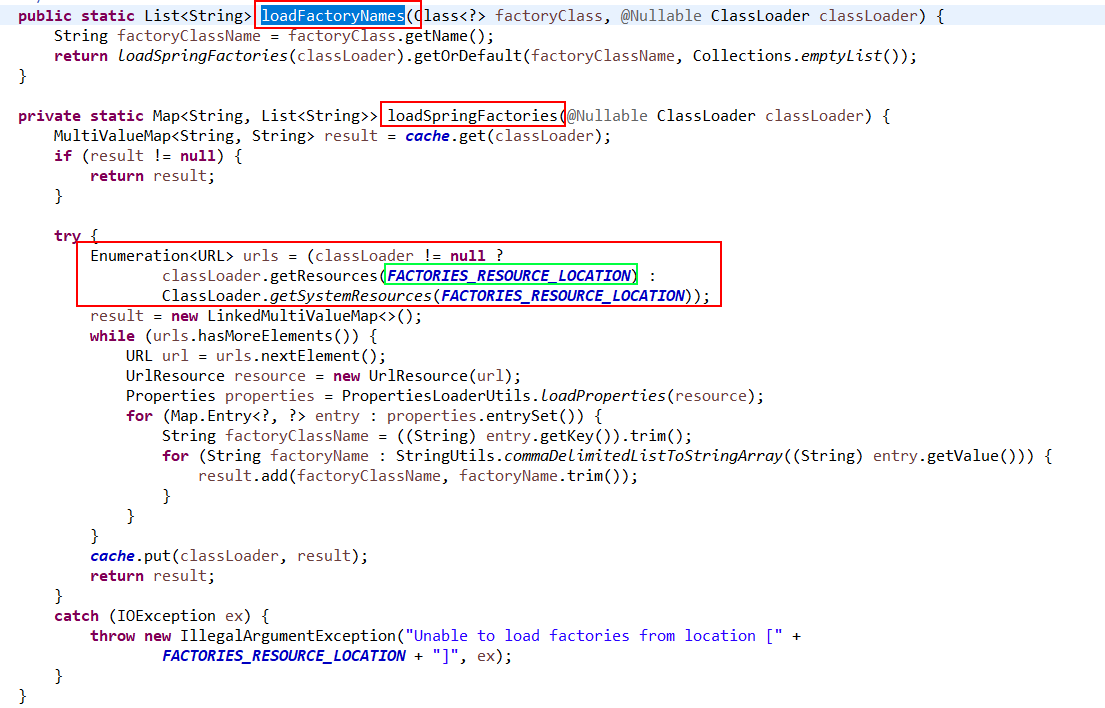


### 3.2.1 SpringApplication初始化构造方法中

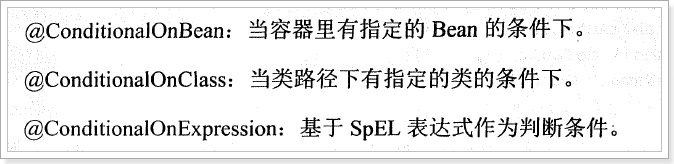


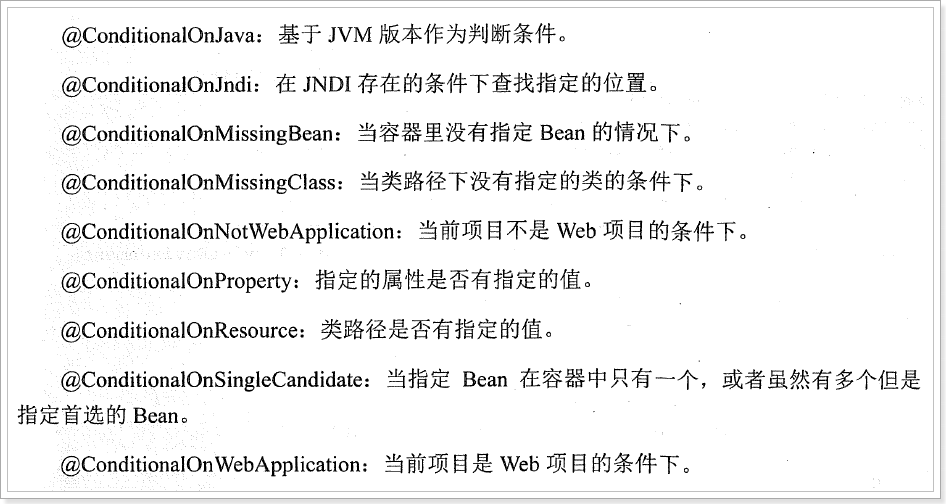


### 3.2.2 SpringFactoriesLoader



### 3.2.3 条件注解





# 4 springboot的 web开发

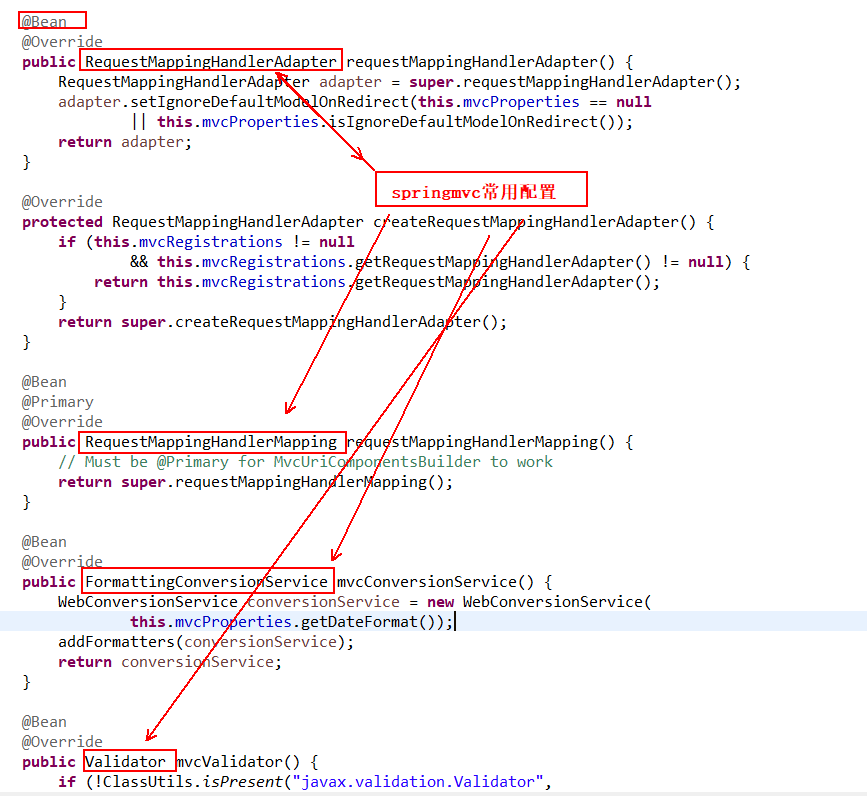
Web开发的自动配置类：org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration

## 4.1 springmvc的自动配置

### 4.1.1 WebMvcAutoConfiguration

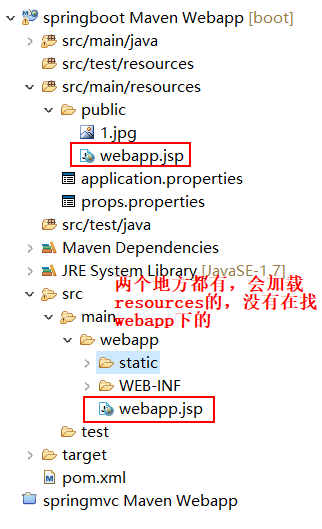
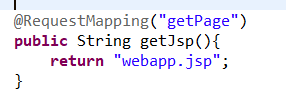
主要使用的是spring基于Java的配置；关键注解@Configuration 和 @Bean





### 4.1.2 springmvc 视图Resolver自动配置示例

不加后缀jsp，则无法找到文件。



## 4.2 静态资源的加载路径

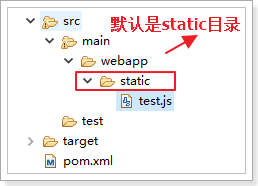
4.2.1如果进入SpringMVC的规则为/时，Spring Boot的默认静态资源的路径为：

spring.resources.static-locations= classpath:/META-INF/resources/,

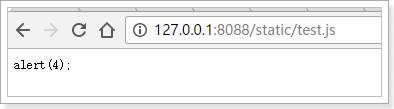
classpath:/resources/,classpath:/static/,classpath:/public/

4.2.2进入规则为\*.xxx 或者 不指定静态文件路径时

将静态资源放置到webapp下的static目录中即可通过地址访问：



测试：



## 4.3 消息转换器

默认是utf-8,所以不需要该配置。仅为一个示例，别的转换器也是这么配置的。



## 4.4 四大器

### 4.4.1 拦截器

### 4.4.2 异常处理器

### 4.4.3 转换器

### 4.4.4 校验器

**package** springboot\_first.test;

**import** java.lang.reflect.Field;

**import** java.util.List;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.springframework.boot.SpringBootConfiguration;

**import** org.springframework.format.FormatterRegistry;

**import** org.springframework.web.servlet.HandlerExceptionResolver;

**import** org.springframework.web.servlet.HandlerInterceptor;

**import** org.springframework.web.servlet.ModelAndView;

**import** org.springframework.web.servlet.config.annotation.InterceptorRegistry;

**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@SpringBootConfiguration

**public** **class** MyWebMvcAutoConfigAdapter **implements** WebMvcConfigurer{

/\* 拦截器 \*/

@Override

**public** **void** addInterceptors(InterceptorRegistry registry) {

//匿名内部类

HandlerInterceptor interceptor = **new** HandlerInterceptor() {

//前置拦截

**public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

**throws** Exception {

System.***out***.println("前置拦截===============================");

**return** HandlerInterceptor.**super**.preHandle(request, response, handler);

}

//后置拦截

**public** **void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,

ModelAndView modelAndView) **throws** Exception {

System.***out***.println("后置拦截===============================");

HandlerInterceptor.**super**.postHandle(request, response, handler, modelAndView);

}

//最终异常拦截

**public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,

Exception ex) **throws** Exception {

System.***out***.println("最终拦截===============================");

HandlerInterceptor.**super**.afterCompletion(request, response, handler, ex);

}

};

//最关键的一步，一定要将自定义的拦截器进行注册，并设置拦截路径

registry.addInterceptor(interceptor).addPathPatterns("/\*\*");

WebMvcConfigurer.**super**.addInterceptors(registry);

}

/\* 异常处理器 \*/

**public** **void** configureHandlerExceptionResolvers(List<HandlerExceptionResolver> resolvers) {

//使用了匿名内部类

HandlerExceptionResolver exception = **new** HandlerExceptionResolver() {

**public** ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,

Exception ex) {

ModelAndView mav = **new** ModelAndView();

mav.addObject("exception", ex.getMessage());

mav.setViewName("exception.jsp");

**return** mav;

}

};

//最关键的一步,一定要将自定义的异常处理器进行注册

resolvers.add(exception);

WebMvcConfigurer.**super**.configureHandlerExceptionResolvers(resolvers);

}

/\* 数据验证器 \*/

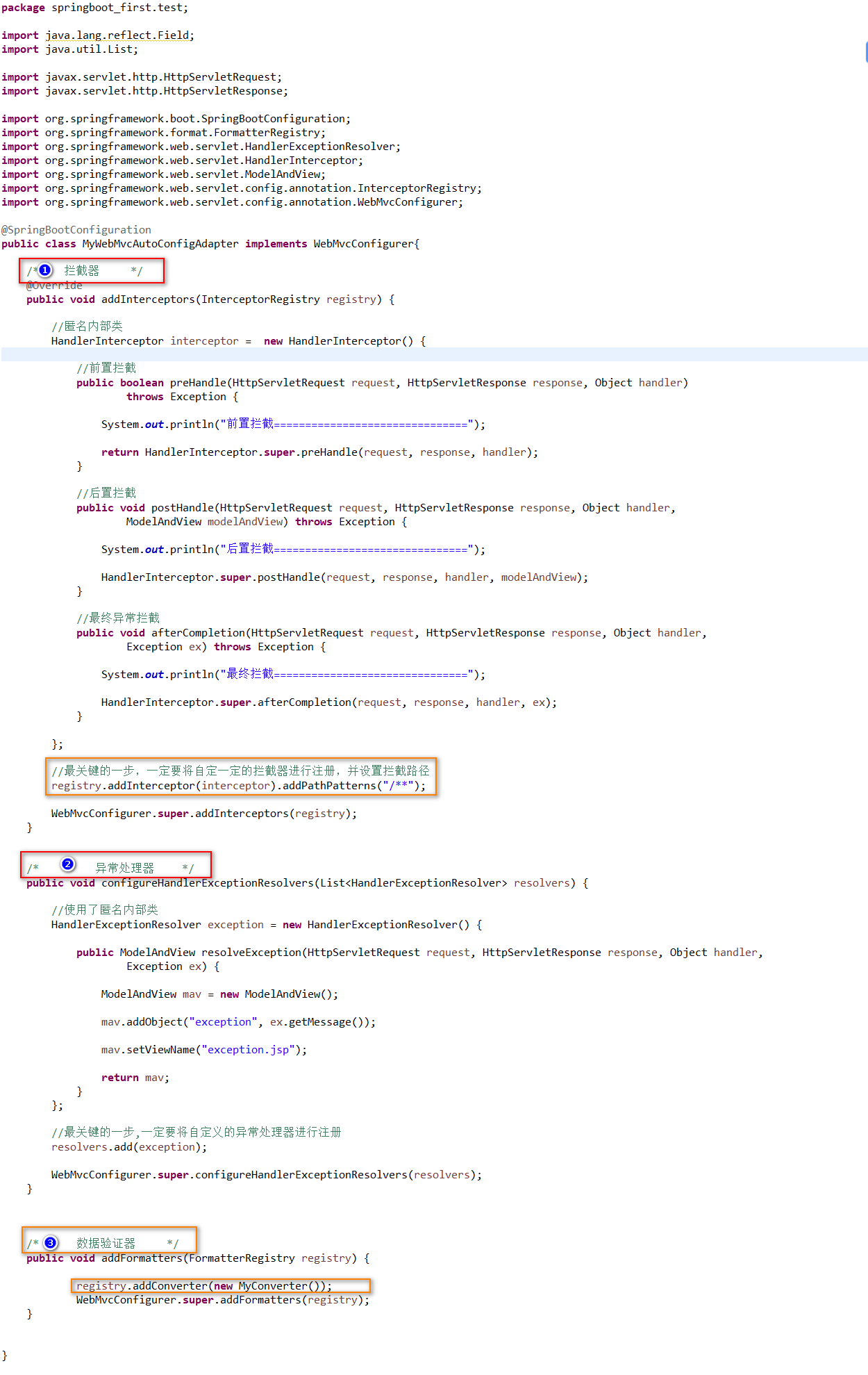
**public** **void** addFormatters(FormatterRegistry registry) {

registry.addConverter(**new** MyConverter());

WebMvcConfigurer.**super**.addFormatters(registry);

}

}



自定义转换器：

**package** springboot\_first.test;

**import** java.lang.reflect.Field;

**import** org.springframework.core.convert.converter.Converter;

**public** **class** MyConverter **implements** Converter<String, User>{

User user = **null**;

@Override

**public** User convert(String source) {

**if**(source != **null** && source.length()>0){

**try** {

String[] strs = source.split("-");

//要在循环外创建实例对象，否则会在每次循环创建一次

user = User.**class**.newInstance();

**for**(**int** i = 0;i<strs.length;i++){

String[] props = strs[i].split(":"); //id:11 name:张三

Field field = User.**class**.getDeclaredField(props[0]);

//因为person中设置属性为private，所以在此设置setAccessible 为true,可访问

field.setAccessible(**true**);

Class<?> type = field.getType();

**if**(type.equals(Integer.**class**)){//判断如果属性时integer类型，则需要对11这个字符串进行转换为integer类型

//不直接使用person.setId() 是因为，如果有多个属性类型为int，就会出现问题，所以使用反射赋值更方便

field.set(user, Integer.*parseInt*(props[1]));

**continue**;

}

field.set(user, props[1]);

}

} **catch** (NoSuchFieldException e) {

e.printStackTrace();

} **catch** (SecurityException e) {

e.printStackTrace();

} **catch** (InstantiationException e) {

e.printStackTrace();

} **catch** (IllegalAccessException e) {

e.printStackTrace();

}

}

**return** user;

}

}

注册方法一：就是在WebMvcConfiger的实现类里注册；

注册方法二：

package springboot\_first.test;

import javax.annotation.PostConstruct;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.convert.support.GenericConversionService;

import org.springframework.web.bind.support.ConfigurableWebBindingInitializer;

import org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter;

@Configuration //必须添加注解作为配置文件使用

public class SpringToUser {

@Autowired

private RequestMappingHandlerAdapter requestMappingHandlerAdapter;

/\*\*

\* 增加字符串转换为user对象

\*/

@PostConstruct

public void addConversionConfig() {

ConfigurableWebBindingInitializer initializer = (ConfigurableWebBindingInitializer) requestMappingHandlerAdapter.getWebBindingInitializer();

if (initializer.getConversionService() != null) {

GenericConversionService genericConversionService = (GenericConversionService)initializer.getConversionService();

//添加字符串转换为User的转换机器

genericConversionService.addConverter(new MyConverter());

}

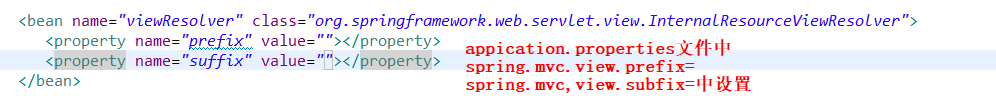
}

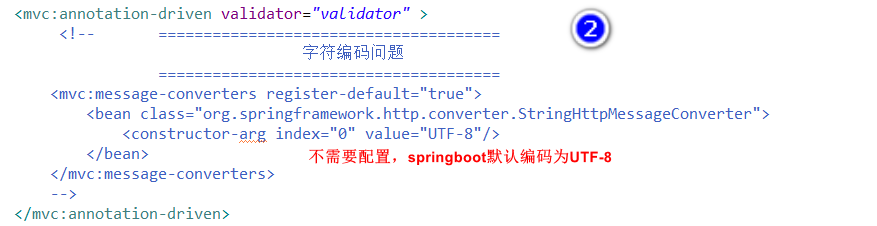
}

# 5 springboot整合springmvc

## 5.1 springmvc原始需要的配置：







四大器：拦截器/异常处理器、校验器/转换器，可以通过在WebMvcConfigurer的实现类中进行配置。



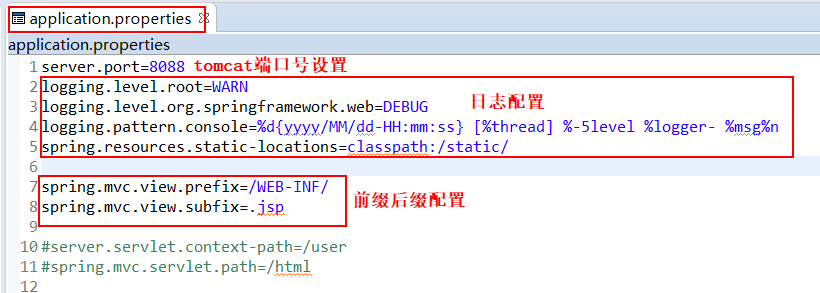


## 5.2 springboot中相关配置

### 5.2.1 扫描、驱动、编码不需要配置

### 5.2.2 前缀后缀配置

#### 5.2.2.1 application.properties全局配置文件，放置在类路径下



前缀不要加webapp

学习spring boot 2.x时，使用jsp作为前端页面。在application.properties配置了jsp所在位置

spring.mvc.view.prefix:/WEB-INF/views/  
spring.mvc.view.suffix:.jsp

启动spring boot时报错，找不到页面

Whitelabel Error Page

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

Sun Mar 10 21:37:47 CST 2019

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

No message available

服务器端日志输出为：

o.s.w.s.r.ResourceHttpRequestHandler     : Path with "WEB-INF" or "META-INF": [WEB-INF/views/home.jsp]

原因是spring boot默认不使用jsp，需要在pom中引用jsp的编译程序：

<!--jsp页面使用jstl标签 -->  
<dependency>  
<groupId>javax.servlet</groupId>  
<artifactId>jstl</artifactId>  
</dependency>

<!--用于编译jsp -->  
<dependency>  
<groupId>org.apache.tomcat.embed</groupId>  
<artifactId>tomcat-embed-jasper</artifactId>  
<scope>provided</scope>  
</dependency>

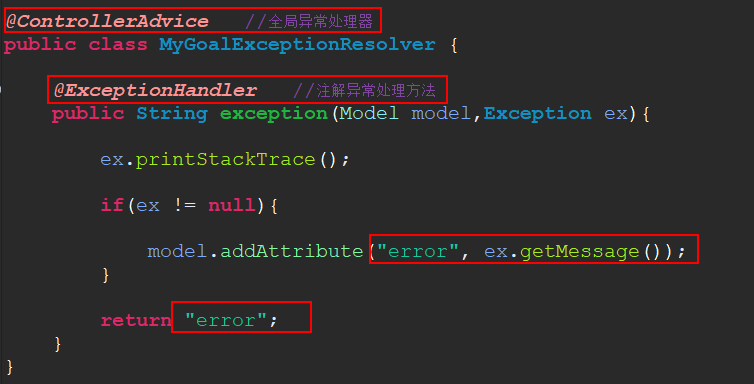
#### 5.2.2.2 编码配置（虽然不太需要，还是了解一下）

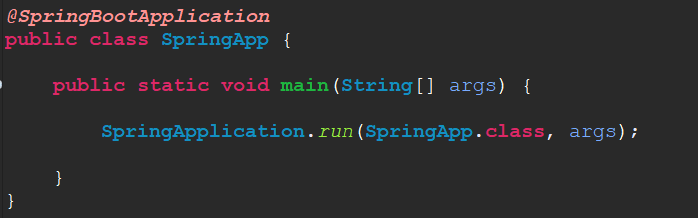
在springboot入口类中配置：

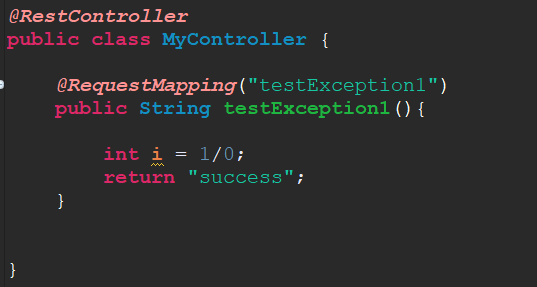


#### 5.2.2.3 [四大器配置](#_4.4_四大器)

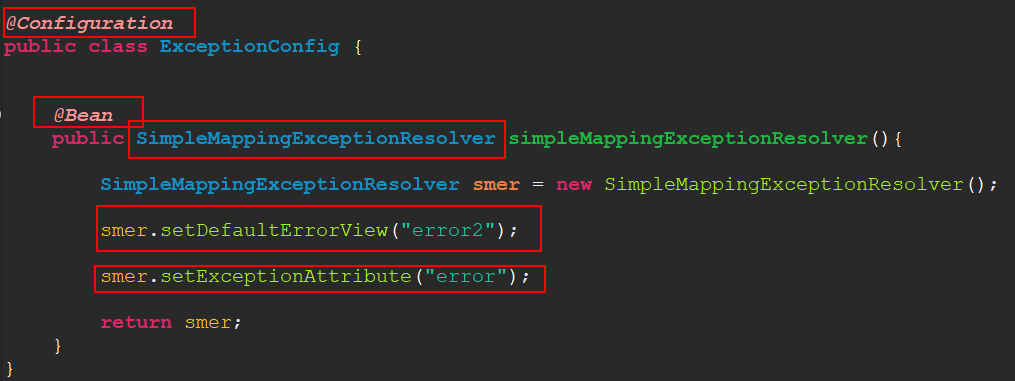
##### @ExceptionHandler @ControllerAdvice

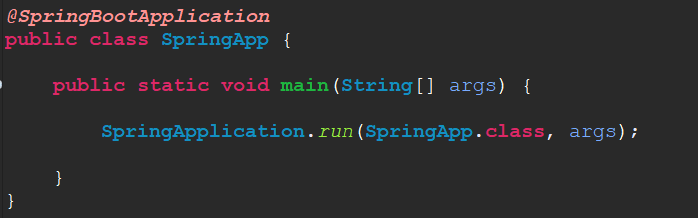


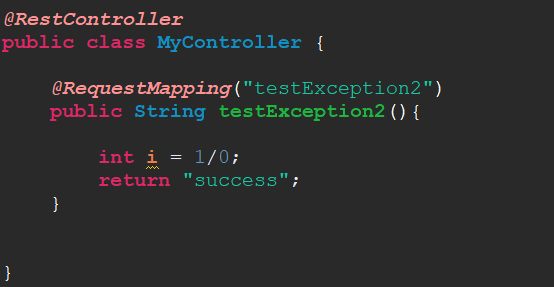




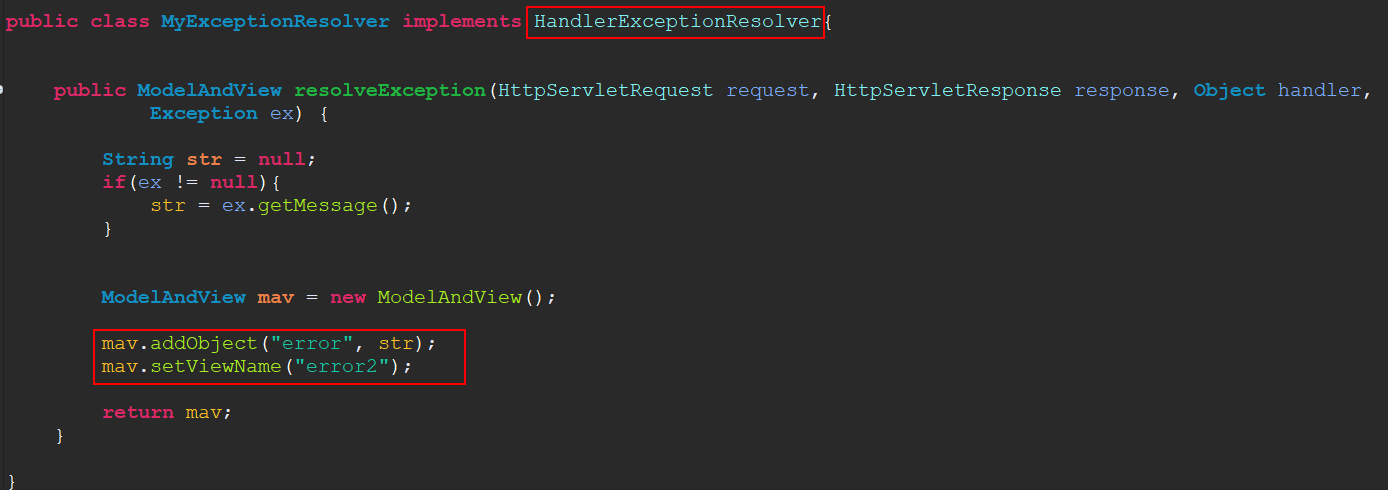
##### SimpleMappingExceptionResolver

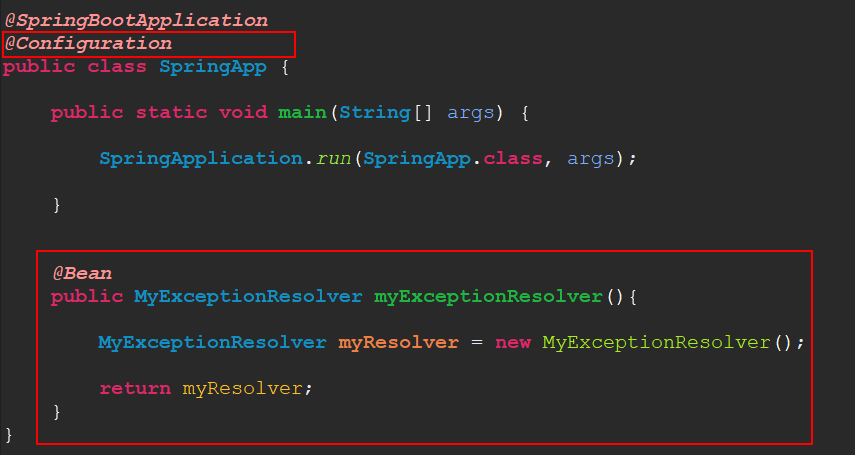


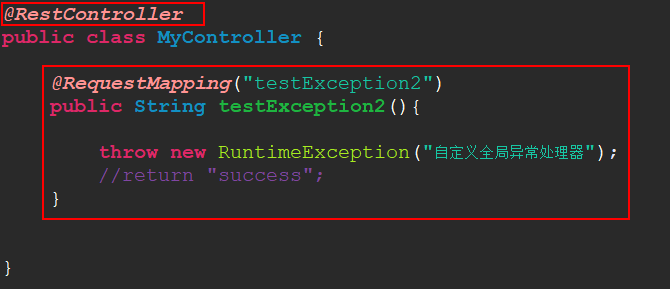




##### MyExceptionResolver







# 6 springboot整合mybatis

整合方式：

一： 使用springboot提供的开发包：MyBatis-Spring-Boot-Starter

二：传统的mybatis-spring 方式整合。

## 6.1 mybatis的原始xml配置

6.1.1 DriverManagerDataSource 数据源配置

6.1.2 SqlSessionFactory 配置

6.1.3 DataSourceTransactionManager 事务管理器

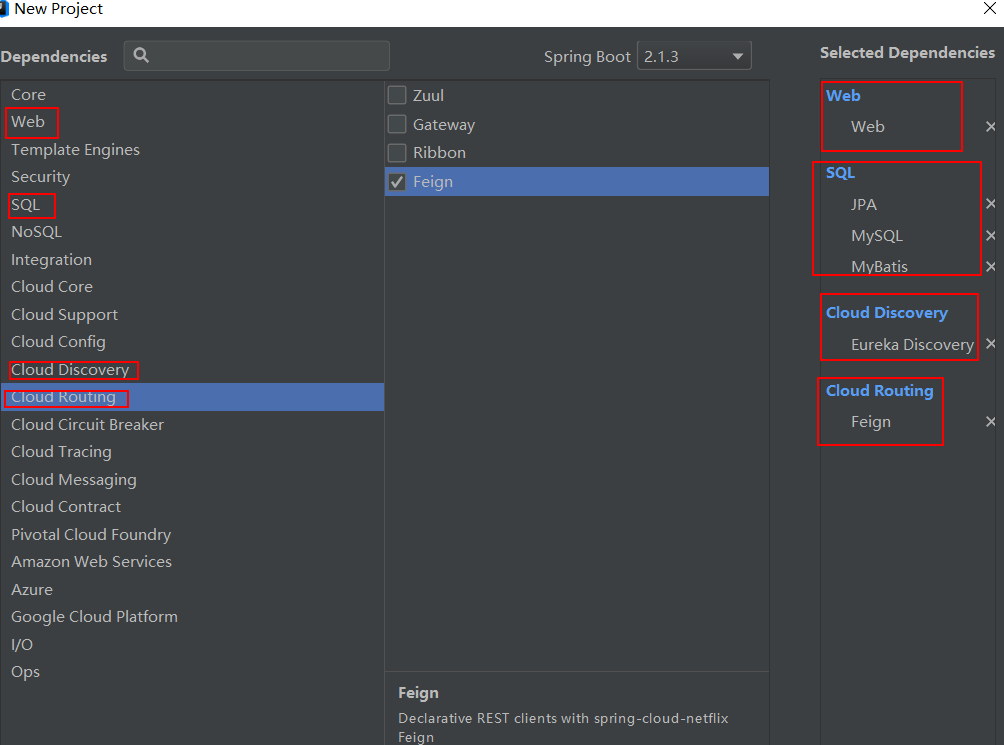
6.1.4 MapperScannerConfigurer服务接口扫描

6.1.5包别名识别

## 6.2 springboot中的mybatis配置

### 实测可用

#### 创建项目：



#### pom依赖：

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.1.3.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>cn.qianzui</groupId>  
 <artifactId>lcn\_store</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>lcn\_store</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Greenwich.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <!-- jpa -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>2.0.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
 </dependency>  
 <!-- mysql链接 -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <!-- 阿里巴巴数据源链接池 -->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.0.31</version>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

#### 配置文件：

server.port=9010  
spring.application.name=lcn-store  
eureka.client.service-url.defaultZone=http://localhost:8089/eureka/  
#默认使用配置  
spring.profiles.active=dev  
#公共配置与profiles选择无关 mapperLocations指的路径是src/main/resources  
mybatis.type-aliases-package=cn.qianzui.lcn\_store.pojo  
#mapperLocations指的路径是src/main/resources 也就是说要把mapper.xml文件放在resource资源文件夹下中  
mybatis.mapper-locations=classpath\*:LcnStoreMapper.xml   
mybatis.config-location=classpath:SqlMapConfig.xml  
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/lcntest?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=utf8&autoReconnect=true  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.username=root  
spring.datasource.password=123456  
  
# 使用druid数据源  
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

#### 注意事项：

1 mapper.xml 文件要放在resource目录下，而且其namespace要和接口的路径相同

2 dao接口要添加注解@Repository （不然IDEA中，无法通过编译进行注入）

3 启动类添加dao接口扫描注解 @MapperScan(basePackages = "cn.qianzui.lcn\_store.mapper")

6.2.1 数据源 配置类 🡪配置数据源

6.2.2 MybatisConfig 配置类 🡪配置sqlSession + 包别名识别

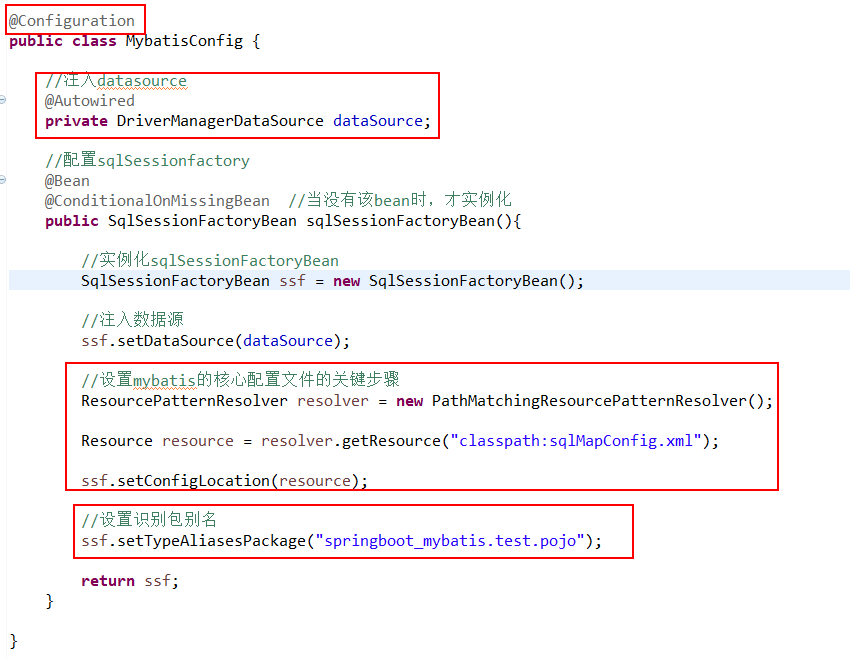
6.2.3 @Transactional 事务管理 🡪当引入jdbc依赖后，springboot会自动配置DataSourceTransactionManager，只需注解即可

6.2.4 MapperSannerConfig 配置类 🡪配置扫描包

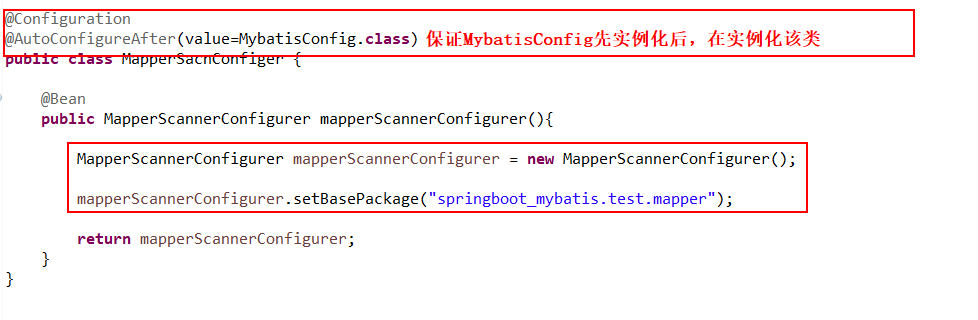
### 数据源配置类：



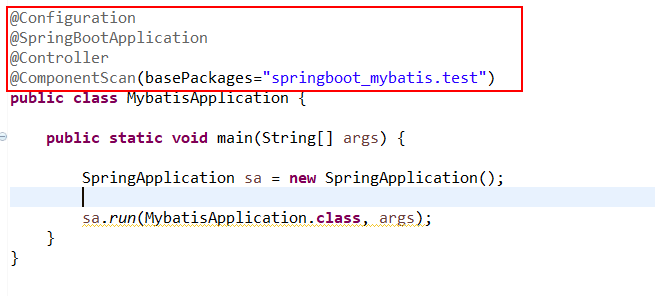
### MybatisConfig主配置文件：



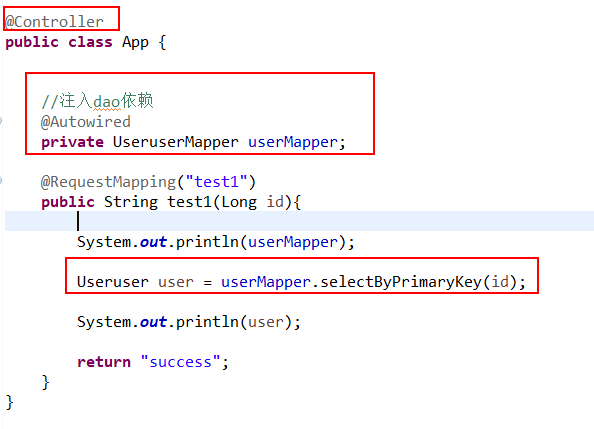
### 包扫描类：



### springboot入口类：



### 测试文件：



### mybatis的映射文件和服务接口，由逆向工程生成。

# SpringBoot整合Session共享

## 版本：springboot2.1.0

pom.xml依赖

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>cn.htb</groupId>  <artifactId>session</artifactId>  <version>0.0.1-SNAPSHOT</version>      <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.2.RELEASE</version>  </parent>    <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency>  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.session</groupId>  <artifactId>spring-session-data-redis</artifactId>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Greenwich.SR1</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

## application.properties设置

注意不要有多余的空格

|  |
| --- |
| server.port=8081  spring.mvc.view.prefix=/WEB-INF/  spring.mvc.view.suffix=.jsp  spring.redis.host=192.168.153.136  spring.redis.port=6379  spring.redis.password=  spring.redis.timeout=2000  spring.session.store-type=redis |

## SpringAPP启动类

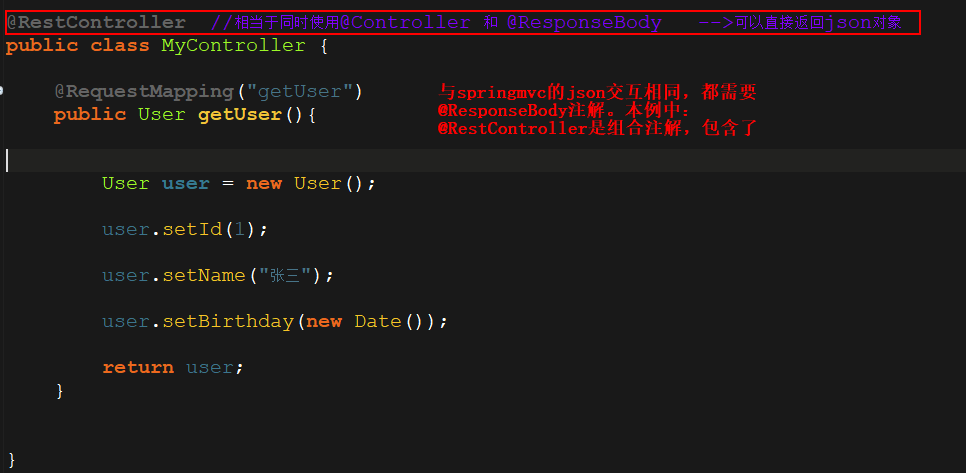
|  |
| --- |
| **package** session;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;  @SpringBootApplication  @EnableRedisHttpSession(maxInactiveIntervalInSeconds= 1800)//开启redis session支持,并配置session过期时间  **public** **class** SpringApp {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringApp.**class**, args);    }  } |

## Controller测试类

|  |
| --- |
| **package** session;  **import** java.util.HashMap;  **import** java.util.Map;  **import** javax.servlet.http.HttpServletRequest;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping(value = "/admin/v1")  **public** **class** QuickRun {  @RequestMapping(value = "/first", method = RequestMethod.***GET***)  **public** Map<String, Object> firstResp (HttpServletRequest request){  Map<String, Object> map = **new** HashMap<>();  request.getSession().setAttribute("request Url", request.getRequestURL());  map.put("request Url", request.getRequestURL());  **return** map;  }    @RequestMapping(value = "/sessions", method = RequestMethod.***GET***)  **public** Object sessions (HttpServletRequest request){  Map<String, Object> map = **new** HashMap<>();  map.put("sessionId", request.getSession().getId());  map.put("message", request.getSession().getAttribute("request Url"));  **return** map;  }  } |

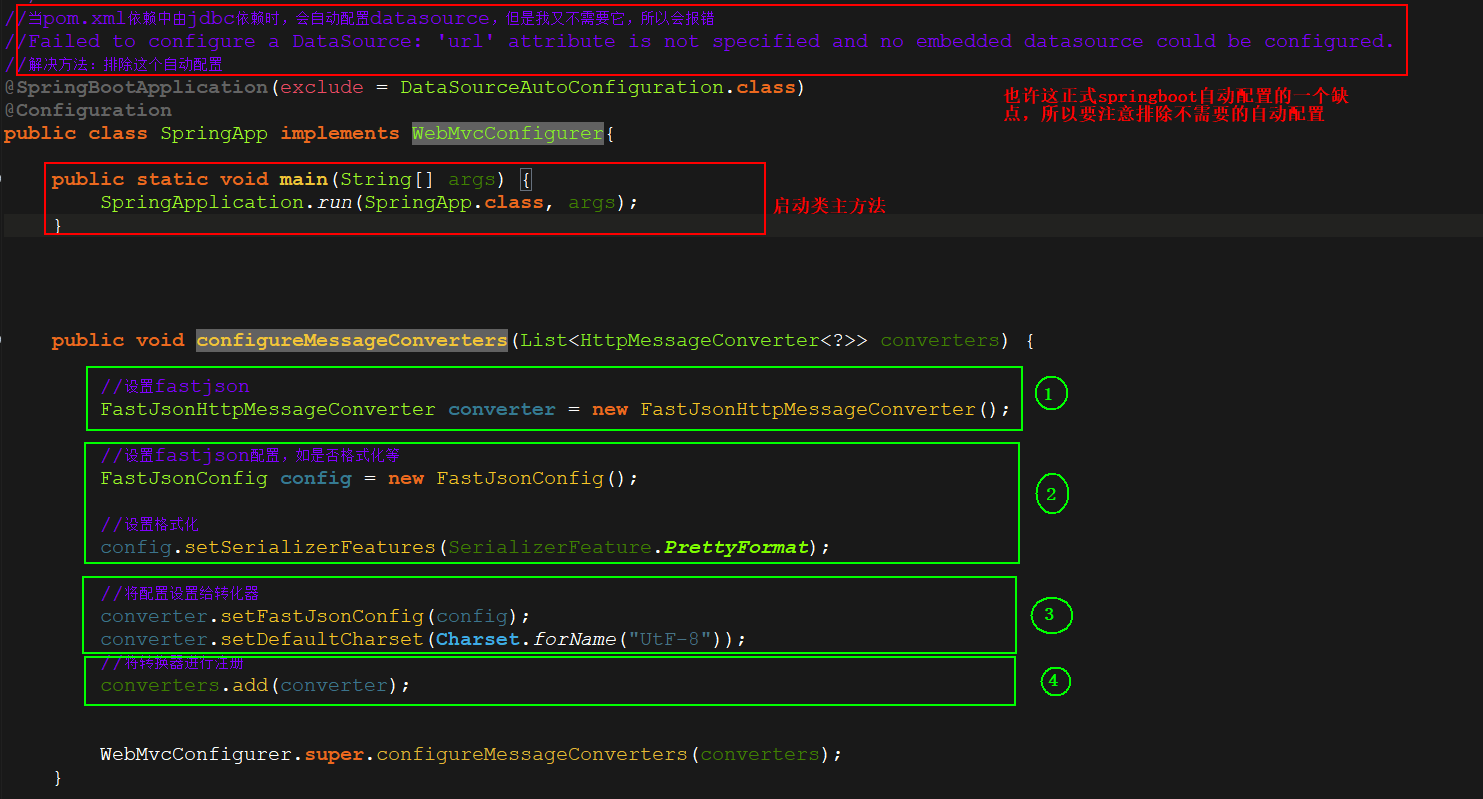
# 7 springboot 的json交互

## 7.1 返回json对象

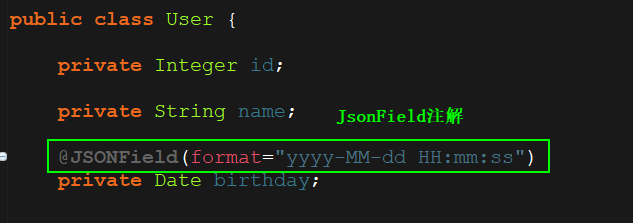


## 7.2 FastJson解析json对象数据

### 7.2.1 方式一：实现WebMvcConfigurer,重写其configureMessageConverters方法



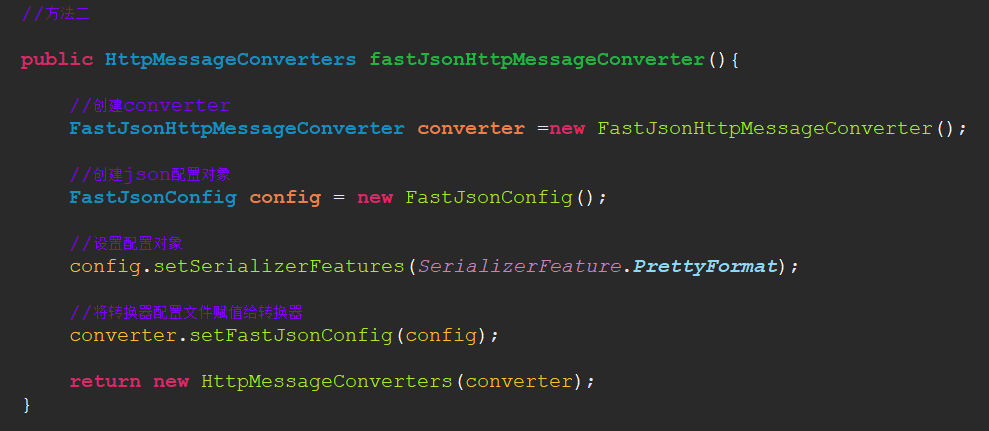
pojo类：



效果：



### 7.2.2 方法二：创建bean对象HttpMessageConvrters



# 8 springboot热部署

## 8.1 springboot的springload热部署

在pom.xml文件添加依赖包：

<plugin>

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

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

<dependencies>

<!--springloaded hot deploy -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.4.RELEASE</version>

</dependency>

</dependencies>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

<configuration>

<classifier>exec</classifier>

</configuration>

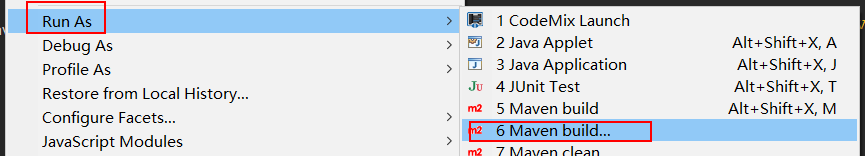
</execution>

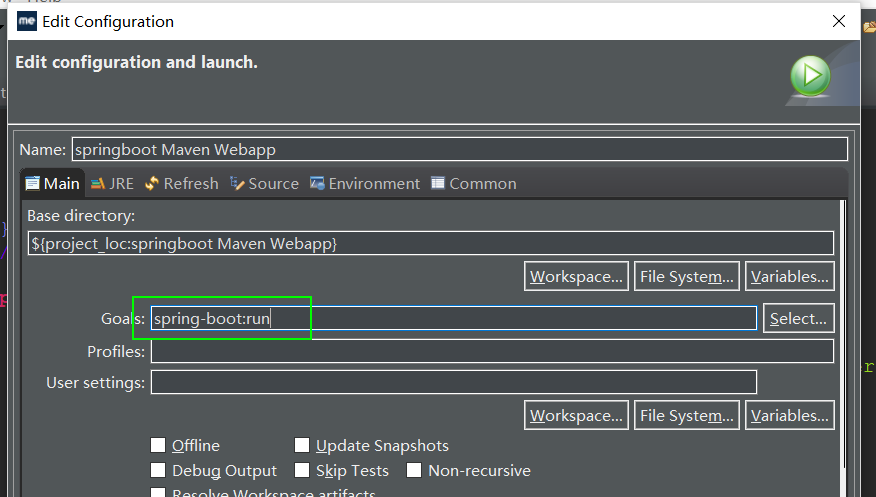
</executions>

</plugin>

### 8.1.1 启动方法

#### 8.1.1.1方法一：

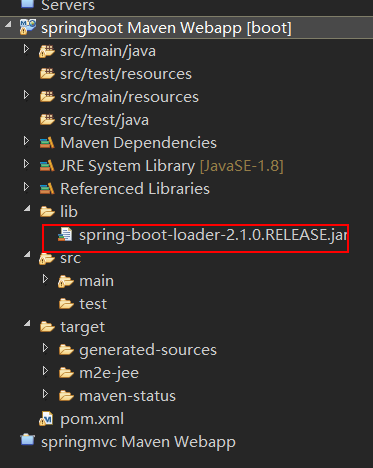


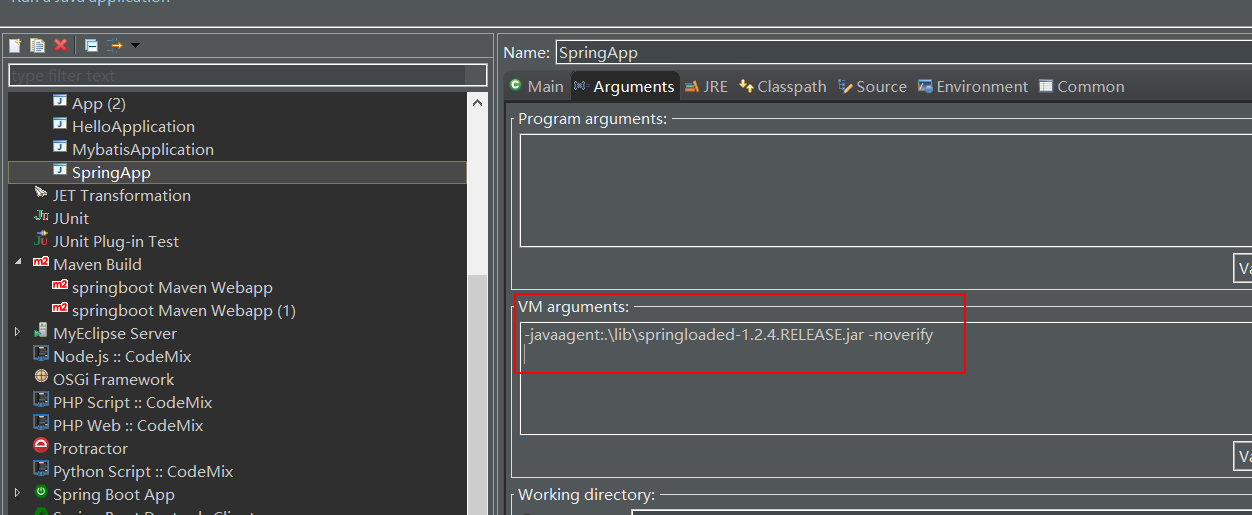


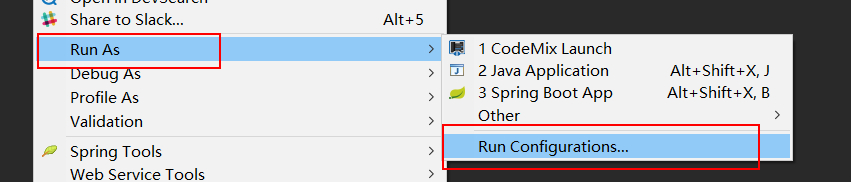
#### 8.1.1.2方法二：

添加spring-loader-1.2.4.RELEASE.jar到项目lib文件下

* 然后把IDEA的run参数里VM参数设置为：
* -javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify
* 然后启动就可以了，这样在run as的时候，也能进行热部署







## 8.2 （重要）spring-boot-devtools热部署

* spring-boot-devtools 是一个为开发者服务的一个模块，其中最重要的功能就是自动应用代码更改到最新的App上面去。原理是在发现代码有更改之后，重新启动应用，但是速度比手动停止后再启动还要更快，更快指的不是节省出来的手工操作的时间。
* 其深层原理是使用了两个ClassLoader，一个Classloader加载那些不会改变的类（第三方Jar包），另一个ClassLoader加载会更改的类，称为 restart ClassLoader
* ,这样在有代码更改的时候，原来的restart ClassLoader 被丢弃，重新创建一个restart ClassLoader，由于需要加载的类相比较少，所以实现了较快的重启时间（5秒以内）。

### 8.2.1 使用方法

#### 8.2.1.1加入依赖：spring-boot-devtools

<**dependency**>

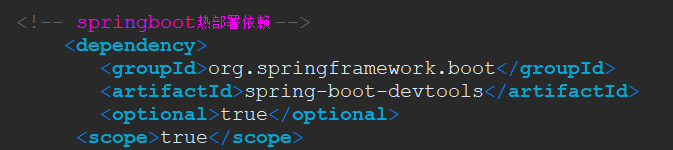
<**groupId**>org.springframework.boot</**groupId**>

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

<**optional**>true</**optional**>

<**scope**>true</**scope**>

</**dependency**>



#### 8.2.1.2 插件修改：

<**build**>

<**plugins**>

<**plugin**>

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

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

<**configuration**>

<!--fork : 如果没有该项配置，devtools不会起作用，即应用不会restart -->

<**fork**>true</**fork**>

</**configuration**>

</**plugin**>

</**plugins**>

</**build**>



### 8.2.2 该热部署原理

1. devtools会监听classpath下的文件变动，并且会立即重启应用（发生在保存时机），注意：因为其采用的虚拟机机制，该项重启是很快的。

2. 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(这里注意不同的模板配置不一样)

### 8.2.3 不能正常使用分析

* 对应的spring-boot版本是否正确，这里使用的是1.4.1版本；
* 是否加入plugin以及属性<fork>true</fork>
* Eclipse Project 是否开启了Build Automatically（我自己就在这里栽了坑，不知道为什么我的工具什么时候关闭了自动编译的功能）。
* 如果设置SpringApplication.setRegisterShutdownHook(false)，则自动重启将不起作用。

8.2.4 可触发热部署场景

>修改方法内容

>重新创建方法

>重新创建一个class

>修改配置文件

>修改页面（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）

# 9 数据持久化操作相关

## 9.1 springboot-jpa-hibernate-springdata

9.1.1 引入依赖：

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

</**dependency**>

<!-- spring-data-jpa -->

<**dependency**>

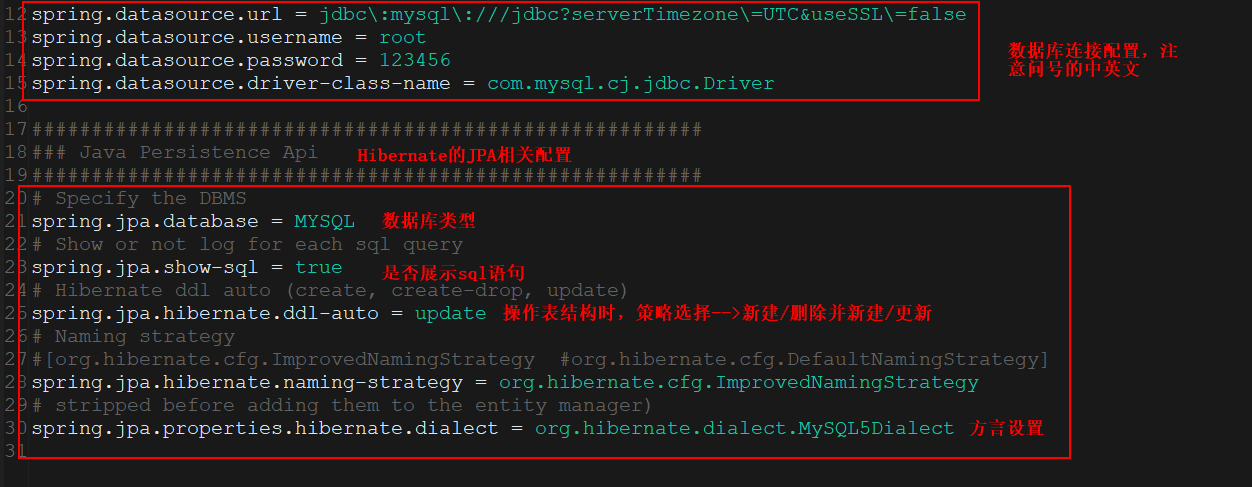
<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-data-jpa</**artifactId**>

</**dependency**>

9.1.2 代码示例：

#### 9.1.2.1 application.properties数据库配置



spring.datasource.url = jdbc\:mysql\:///jdbc?serverTimezone\=UTC&useSSL\=false

spring.datasource.username = root

spring.datasource.password = 123456

spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver

########################################################

### Java Persistence Api

########################################################

# Specify the DBMS

spring.jpa.database = MYSQL

# Show or not log for each sql query

spring.jpa.show-sql = true

# Hibernate ddl auto (create, create-drop, update)

spring.jpa.hibernate.ddl-auto = update

# Naming strategy

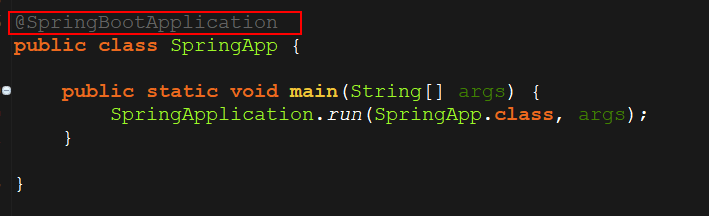
#[org.hibernate.cfg.ImprovedNamingStrategy #org.hibernate.cfg.DefaultNamingStrategy]

spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy

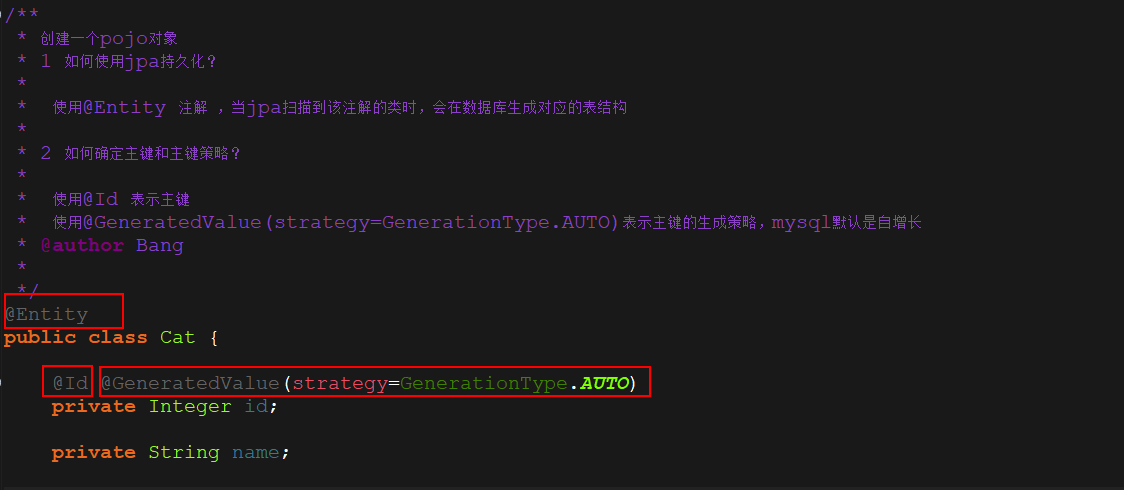
# stripped before adding them to the entity manager)

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

#### 9.1.2.2 springboot启动类创建



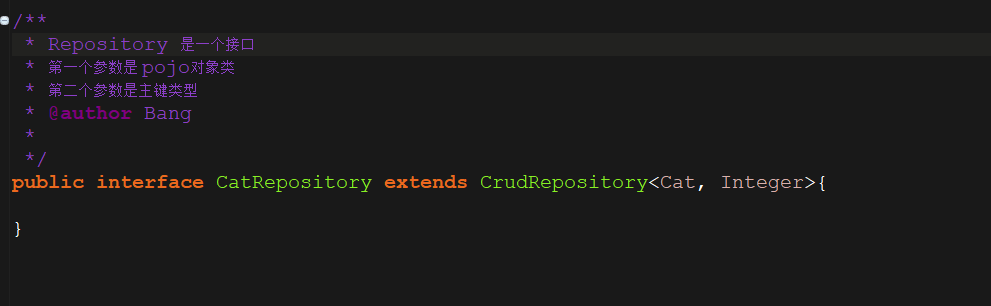
#### 9.1.2.3 pojo创建



#### 9.1.2.4 Repository创建

可以查看实现的Crud Repository是Repository的一个实现类，再Repository中有相应的增/删/改/查的方法。

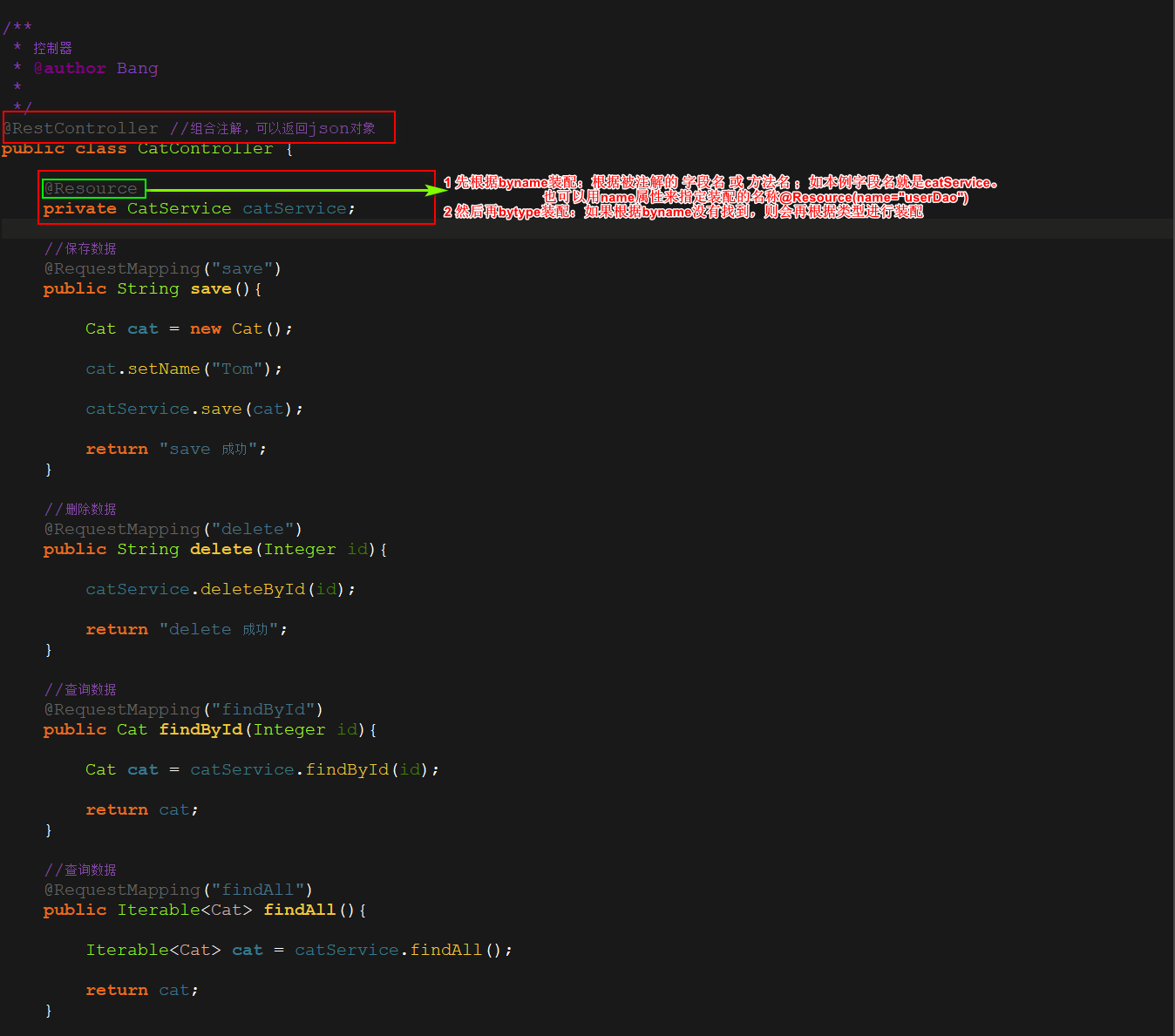
仓库的save方法，类似saveorupdate 方法。



#### 9.1.2.5 Service创建



#### 9.1.2.6 controller创建



## 9.2 springboot-srpingdata-jpa深入分析

### 9.2.1 仓库Repository 接口

***@Indexed***

**public** **interface** Repository<**T**, **ID**> {}

Repository 接口是 Spring Data 的一个核心接口，它不提供任何方法，开发者需要在自己定义的接口中声明需要的方法 。

有这么几点需要强调下：

1. Repository是一个空接口，即是一个标记接口；

2. 若我们定义的接口继承了Repository，则该接口会被IOC容器识别为一个Repository Bean纳入到IOC容器中，进而可以在该接口中定义满足一定规范的方法。

3. 实际上也可以通过@RepositoryDefinition,注解来替代继承Repository接口。

4. 查询方法以find | read | get开头；

5. 涉及查询条件时，条件的属性用条件关键字连接，要注意的是条件属性以首字母大写。如：findByUserName(String userName)

6.使用@Query注解可以自定义JPQL语句实现更灵活的查询。(JPQL语句和HQL语句类似)

### 9.2.2 CrudRepository接口

CrudRepository 接口提供了最基本的对实体类的添删改查操作

* --T save(T entity);//保存单个实体
* --Iterable<T> save(Iterable<? extends T> entities);//保存集合
* --T findById(ID id);//根据id查找实体
* --boolean exists(ID id);//根据id判断实体是否存在
* --Iterable<T> findAll();//查询所有实体,不用或慎用!
* --long count();//查询实体数量
* --void deleteById(ID id);//根据Id删除实体
* --void delete(T entity);//删除一个实体
* --void delete(Iterable<? extends T> entities);//删除一个实体的集合
* --void deleteAll();//删除所有实体,不用或慎用!

### 9.2.3 PagingAndSortingRepository接口

该接口提供了分页与排序功能

--Iterable<T> findAll(Sort sort); //排序

--Page<T> findAll(Pageable pageable); //分页查询（含排序功能）

#### 9.2.3.1 注意问题：

可能会抛出异常：classNotFoundException:com.querydsl.core.types.OrderSpecifier

解决方法：

添加依赖：

<**dependency**>

<**groupId**>com.querydsl</**groupId**>

<**artifactId**>querydsl-jpa</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>com.querydsl</**groupId**>

<**artifactId**>querydsl-apt</**artifactId**>

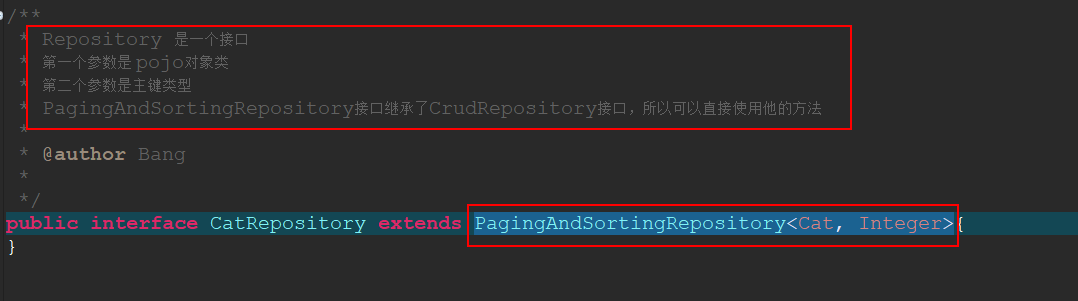
<**scope**>provided</**scope**>

</**dependency**>

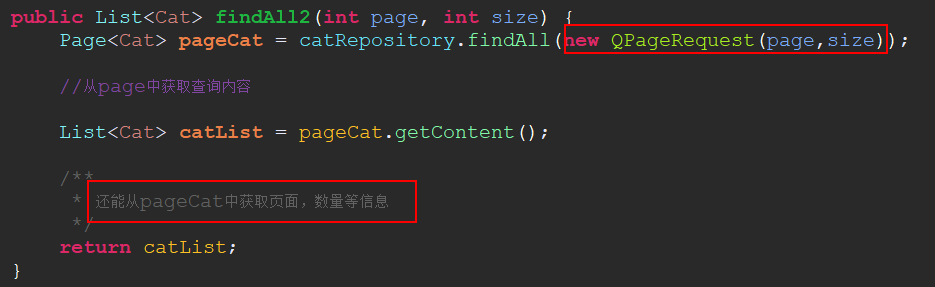
</**dependencies**>

#### 9.2.3.2 使用示例：

仓库接口：



service代码：



controller代码



### 9.2.4 其他接口

**JpaRepository：**查找所有实体，排序、查找所有实体，执行缓存与数据库同步

**JpaSpecificationExecutor：**不属于Repository体系，实现一组 JPA Criteria 查询相关的方法，封装 JPA Criteria 查询条件。通常使用匿名内部类的方式来创建该接口的对象。

**自定义 Repository：可以自己定义一个MyRepository接口。**

### 9.2.5 自定义仓库接口



## 9.3 springboot-jdbcTemplate使用

### 9.3.1 依赖

<dependency>

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

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

</dependency>

如果在JPA已经加入的话，则可以不用引入以上的配置。

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

### 9.3.2 dao

**package** springboot\_jpa\_springdata.dao;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** javax.annotation.Resource;

**import** org.springframework.jdbc.core.BeanPropertyRowMapper;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.jdbc.core.RowMapper;

**import** org.springframework.stereotype.Repository;

**import** springboot\_jpa\_springdata.pojo.Cat;

***@Repository***

**public** **class** **CatDao** {

***@Resource***

**private** **JdbcTemplate** jdbcTemplate;

/\*\*

\* 使用jdbcTemplate 查询复杂对象：方法一

\* **@param** id

\* **@return**

\*/

**public** **Cat** **findById**(**Integer** id){

**String** **sql** = "select id,name from cat where id = ?";

/\*\*

\* queryForObject(sql, requiredType, args)

\* 第二个参数requiredType 只能是返回基本数据类型时使用，而且返回结果时一列。 如：String.class,Integer.class

\* 如果传参其他对象类型。如：Cat.class 会抛出异常Incorrect column count: expected 1, actual 2

\* 所以，查询复杂对象时，要使用RowMapper，创建器实现类。

\*

\*/

**Cat** **cat** = jdbcTemplate.queryForObject(sql, **new** MyRowMapper(), id);

**return** cat;

}

/\*\*

\* 使用jdbcTemplate 查询复杂对象：方法二

\* **@param** id

\* **@return**

\*/

**public** **Cat** **findById2**(**Integer** id){

**String** **sql** = "select id,name from cat where id = ?";

/\*\*

\* queryForObject(sql, requiredType, args)

\* 第二个参数requiredType 只能是返回基本数据类型时使用，而且返回结果时一列。 如：String.class,Integer.class

\* 如果传参其他对象类型。如：Cat.class 会抛出异常Incorrect column count: expected 1, actual 2

\* 所以，查询复杂对象时，要使用RowMapper，创建器实现类。

\*

\*/

**Cat** **cat** = jdbcTemplate.queryForObject(sql, **new** BeanPropertyRowMapper<Cat>(**Cat**.**class**), id);

**return** cat;

}

}

**class** **MyRowMapper** **implements** RowMapper<Cat>{

***@Override***

**public** **Cat** **mapRow**(ResultSet rs, **int** rowNum) **throws** **SQLException** {

**Cat** **cat** = **new** Cat();

cat.setId(rs.getInt("id"));

cat.setName(rs.getString("name"));

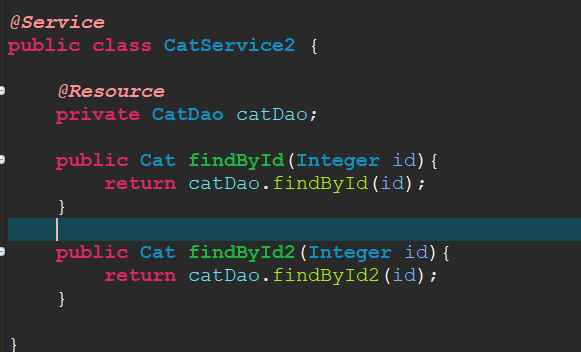
**return** cat;

}

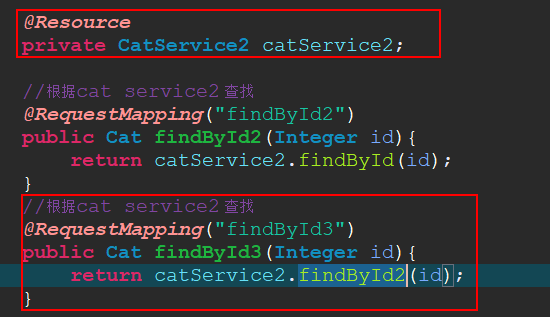
}



### 9.3.3 service



### 9.3.4 controller



## 9.4整合redis

### pom.xml依赖

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.example</groupId>  
 <artifactId>demo</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>demo</name>  
 <description>Demo project for Spring Boot</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.8.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
  
</project>

### RedisConfig

package com.imooc.activiti.springboot;  
  
import com.fasterxml.jackson.annotation.JsonAutoDetect;  
import com.fasterxml.jackson.annotation.PropertyAccessor;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.redis.connection.RedisConnectionFactory;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.data.redis.core.StringRedisTemplate;  
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;  
  
@Configuration  
public class RedisConfig {  
 @Bean  
 public RedisTemplate<String, String> redisTemplate(RedisConnectionFactory factory) {  
 StringRedisTemplate template = new StringRedisTemplate(factory);  
 Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);  
 ObjectMapper objectMapper = new ObjectMapper();  
 objectMapper.setVisibility(PropertyAccessor.*ALL*, JsonAutoDetect.Visibility.*ANY*);  
 objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.*NON\_FINAL*);  
 jackson2JsonRedisSerializer.setObjectMapper(objectMapper);  
 template.setValueSerializer(jackson2JsonRedisSerializer);  
 template.afterPropertiesSet();  
 return template;  
 }  
}

### application.yml

**spring:  
 application:  
 name:** redis-demo  
 **redis:  
 database:** 0  
 **host:** 192.168.153.136  
 **port:** 6379  
 **timeout:** 0 *# 连接超时时间（毫秒）* **pool:  
 max-active:** 20 *# 连接池最大连接数（使用负值表示没有限制）* **max-idle:** 20 *# 连接池中的最大空闲连接* **max-wait:** -1 *# 连接池最大阻塞等待时间（使用负值表示没有限制）* **min-idle:** 0 *# 连接池中的最小空闲连接***server:  
 port:** 8080  
 **servlet:  
 context-path:** /redis-demo

### 启动类：

package com.imooc.activiti.springboot;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication()  
public class SpringbootApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(SpringbootApplication.class, args);  
 }  
  
}

### 测试类：

package com.imooc.activiti.springboot;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
import java.util.Date;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest()  
public class SpringbootApplicationTests {  
  
 @Autowired  
 private PrimaryKeyService primaryKeyService;  
  
 @Test  
 public void contextLoads() {  
 long startMillis = System.*currentTimeMillis*();  
 String orderIdPrefix = primaryKeyService.getOrderIdPrefix(new Date());  
 for (int i = 0; i < 10; i++) {  
 Long aLong = primaryKeyService.orderId(orderIdPrefix);  
 System.*out*.println(aLong);  
 }  
 long endMillis = System.*currentTimeMillis*();  
 System.*out*.println("生成速度:"+(endMillis-startMillis)+",单位毫秒");  
  
 }  
  
}

## 9.5 整合mybatis

### 依赖：

|  |
| --- |
| <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.1</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.11</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.9</version>  </dependency>s |

### 配置：

|  |
| --- |
| server.port=10101  spring.application.name=user-service  eureka.client.service-url.defaultZone=http://localhost:10010/eureka/  ribbon.eureka.enabled=true  ribbon.http.client.enabled=true  feign.hystrix.enabled=true  #thymeleaf  spring.thymeleaf.prefix=classpath:/templates/  spring.thymeleaf.suffix=.html  spring.thymeleaf.cache=false  spring.thymeleaf.content-type=text/html  spring.thymeleaf.enabled=true  spring.thymeleaf.encoding=UTF-8  spring.thymeleaf.mode=HTML5  # mybatis  mybatis.type-aliases-package=com.qianzui.pojo.user  mybatis.configuration.map-underscore-to-camel-case=true  mybatis.configuration.default-fetch-size=100  mybatis.configuration.default-statement-timeout=3000  mybatis.mapperLocations = classpath:com/qianzui/service/user/dao/\*.xml  # druid  spring.datasource.url=jdbc:mysql://127.0.0.1:3306/taotao?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=utf8&autoReconnect=true  spring.datasource.username=root  spring.datasource.password=123456  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  spring.datasource.filters=stat  spring.datasource.maxActive=1000  spring.datasource.initialSize=100  spring.datasource.maxWait=60000  spring.datasource.minIdle=500  spring.datasource.timeBetweenEvictionRunsMillis=60000  spring.datasource.minEvictableIdleTimeMillis=300000  spring.datasource.validationQuery=select 'x'  spring.datasource.testWhileIdle=true  spring.datasource.testOnBorrow=false  spring.datasource.testOnReturn=false  spring.datasource.poolPreparedStatements=true  spring.datasource.maxOpenPreparedStatements=20 |

### 启动类：

|  |
| --- |
| **package** com.qianzui.service.user;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  /\*\*  \* **@author** 浅醉  \* **@email** 1795683509@qq.com  \*/  @SpringBootApplication  @EnableEurekaClient  @MapperScan("com.qianzui.service.user.dao") //掃描dao接口  **public** **class** QianZuiUserService {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(QianZuiUserService.**class**, args);  }  } |

### dao接口： 注意添加注解@Mapper

或者启动类添加：@MapperScan("com.qianzui.service.user.dao") //掃描dao接口

|  |
| --- |
| **package** com.qianzui.service.user.dao;  **import** com.qianzui.pojo.uer.TbUser;  **import** com.qianzui.pojo.uer.TbUserExample;  **import** java.util.List;  **import** org.apache.ibatis.annotations.Mapper;  **import** org.apache.ibatis.annotations.Param;  @Mapper  **public** **interface** TbUserMapper {  **int** countByExample(TbUserExample example);  **int** deleteByExample(TbUserExample example);  **int** deleteByPrimaryKey(Long id);  **int** insert(TbUser record);  **int** insertSelective(TbUser record);  List<TbUser> selectByExample(TbUserExample example);  TbUser selectByPrimaryKey(Long id);  **int** updateByExampleSelective(@Param("record") TbUser record, @Param("example") TbUserExample example);  **int** updateByExample(@Param("record") TbUser record, @Param("example") TbUserExample example);  **int** updateByPrimaryKeySelective(TbUser record);  **int** updateByPrimaryKey(TbUser record);  } |

# 10 扩展 模板引擎

## 10.1 thymeleaf 模板引擎

### 10.1.1 添加依赖：

<dependency>

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

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

</dependency>

### 10.1.2 application.properties

########################################################

###THYMELEAF (ThymeleafAutoConfiguration)

########################################################

#spring.thymeleaf.prefix=classpath:/templates/

#spring.thymeleaf.suffix=.html

#spring.thymeleaf.mode=HTML5

#spring.thymeleaf.encoding=UTF-8

# ;charset=<encoding> is added

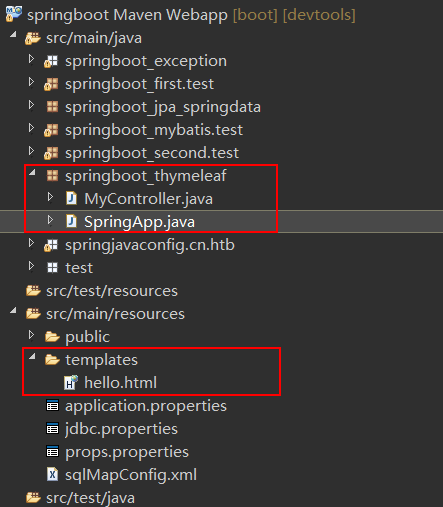
#spring.thymeleaf.content-type=text/html

# set to false for hot refresh

spring.thymeleaf.cache=false

### 10.1.3 简单示例

#### 10.1.3.1 目录结构：



10.1.3.2 controller

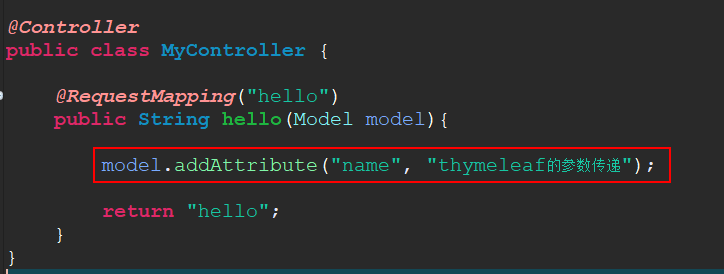


10.1.3.3 测试结果：



### 10.1.4 参数传递

#### 10.1.4.1 controller



#### 10.1.4.2 html页面

<!**DOCTYPE** html>

<**html** xmlns=*"http://www.w3.org/1999/xhtml"*

xmlns:th=*"http://www.thymeleaf.org"*>

<**head**>

<**title**>hello.html</**title**>

<**meta** name=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<**meta** name=*"description"* content=*"this is my page"*>

<**meta** name=*"content-type"* content=*"text/html; charset=UTF-8"*>

<!--<link rel="stylesheet" type="text/css" href="./styles.css">-->

</**head**>

<**body**>

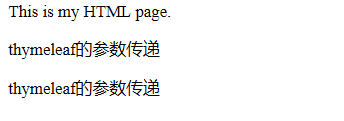
This is my HTML page. <**br**>

<**p** th:text=*"${name}"*></**p**>

</**body**>

</**html**>

#### 10.1.4.3 测试结果



## 10.2 thymeleaf [基本语法](thymeleaf语法1.oxps)

th:text=”${name}” 🡪 获取参数信息

th:text=”#{name}” 🡪 国际化

th:href=”@{/hello}” 🡪 url

## 10.3 freemarker 模板引擎

### 10.3.1 依赖

<dependency>

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

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

</dependency>

### 10.3.2 application.properties 配置信息

########################################################

###FREEMARKER (FreeMarkerAutoConfiguration)

########################################################

spring.freemarker.allow-request-override=false

spring.freemarker.cache=true

spring.freemarker.check-template-location=true

spring.freemarker.charset=UTF-8

spring.freemarker.content-type=text/html

spring.freemarker.expose-request-attributes=false

spring.freemarker.expose-session-attributes=false

spring.freemarker.expose-spring-macro-helpers=false

#与thymeleaf 不同freemarker的前缀不能是classpath:/templates/，否则会报错

# 如 Error resolving template [hello2], template might not exist or might not be accessible by any of the configured Template Resolvers

#如果果改成 / ，则不会报错

#spring.freemarker.prefix=

#spring.freemarker.request-context-attribute=

#spring.freemarker.settings.\*=

#spring.freemarker.suffix=.ftl

#spring.freemarker.template-loader-path=classpath:/templates/ #comma-separated list

#spring.freemarker.view-names= # whitelist of view names that can be resolved

### 10.3.3 controller代码

@RequestMapping("hello2")

public String hello2(Model model){

model.addAttribute("name", "freemarker的参数传递");

return "hello2";

}

### 10.3.4 hello2.ftl 代码

<!DOCTYPE html>

<html>

<head>

<title>hello.html</title>

<meta charset=UTF-8">

</head>

<body>

This is my HTML page. <br>

${name}

</body>

</html>

## 10.4 thymeleaf 和 freemarker

10.4.1 两者可以共存

一个项目中可以同时存在这两个模板引擎。

10.4.2 freemarker的页面类型可以是.html

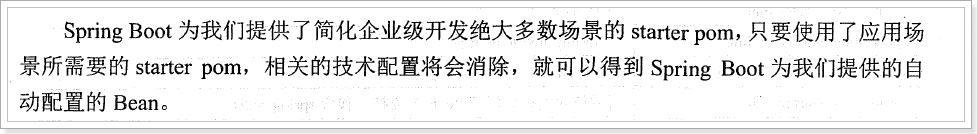
freemarker的页面类型可以是html，但是要在配置文件中的后缀进行修改。

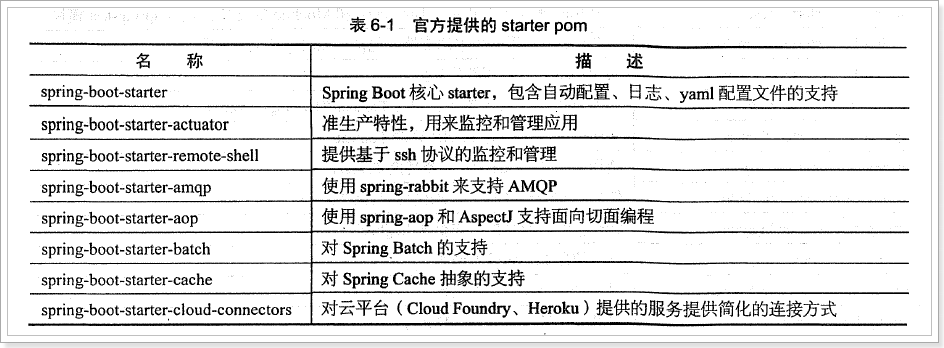
104.3 不同点：两者的前缀配置略有不同：

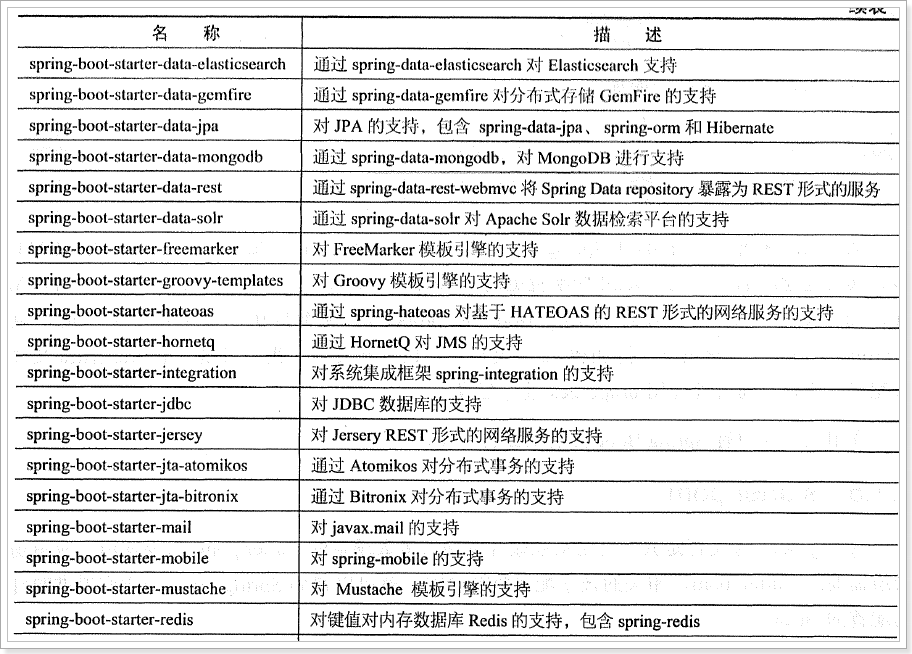
thymeleaf : spring.thymeleaf.prefix=classpath:/templates/ 可以开启，也可以不开启

freemarker: freemarker的前缀不能是classpath:/templates/，否则会报错. 但可以不开启，也可以是/

# Starter.pom









# 问题解决：

## 1 SLF4J: Failed to load class “org.slf4j.impl.StaticLoggerBinder”

添加依赖包：

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.25</version>

</dependency>

## 2 某些类无法加载，可以删除仓库包重新加载

## 3 不能加载SpringContextServletServiceBean

是因为没有添加注解@SpringBootApplication

# Junit测试

在test包下

package com.imooc.activiti.springboot;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.test.context.junit4.SpringRunner;  
  
import java.util.Date;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest()  
public class SpringbootApplicationTests {  
  
 @Autowired  
 private PrimaryKeyService primaryKeyService;  
  
 @Test  
 public void contextLoads() {  
 long startMillis = System.*currentTimeMillis*();  
 String orderIdPrefix = primaryKeyService.getOrderIdPrefix(new Date());  
 for (int i = 0; i < 10; i++) {  
 Long aLong = primaryKeyService.orderId(orderIdPrefix);  
 System.*out*.println(aLong);  
 }  
 long endMillis = System.*currentTimeMillis*();  
 System.*out*.println("生成速度:"+(endMillis-startMillis)+",单位毫秒");  
  
 }  
  
 @Autowired  
 private RedisTemplate redisTemplate;  
 @Test  
 public void test(){  
  
 }  
  
  
}