目录

[SpringBoot常用注解 4](#_Toc11159)

[SpringMVC中注解 5](#_Toc32346)

[Springboot 支持 多环境测试 Profile 7](#_Toc3562)

[SpringBoot默认支持的数据校验 8](#_Toc15520)

[SpringBoot 一些常用接口 9](#_Toc31099)

[初始化相关 9](#_Toc1126)

[CommandLineRunner 9](#_Toc2530)

[WebMvcConfigurer 9](#_Toc1696)

[Springboot 与 日志框架的选择 9](#_Toc10496)

[Springboot 与 web 模块 11](#_Toc30597)

[拦截器 12](#_Toc11448)

[配置druid 数据库链接池 14](#_Toc9673)

[https访问控制 14](#_Toc16329)

[SpringBoot项目的 打包，发布 15](#_Toc13475)

[SpringBoot异常处理 16](#_Toc22070)

[SpringBoot配置错误页面 17](#_Toc22281)

[SpringBoot 打成war包 部署到tomcat 17](#_Toc12823)

[文件上传 19](#_Toc7627)

[Springboot 和 thymeleaf 20](#_Toc25546)

[Restful风格 22](#_Toc14187)

[SpringBoot整合Restful 架构 23](#_Toc18655)

[配置Swagger 服务 25](#_Toc14317)

[AbstractBaseController的作用 27](#_Toc26422)

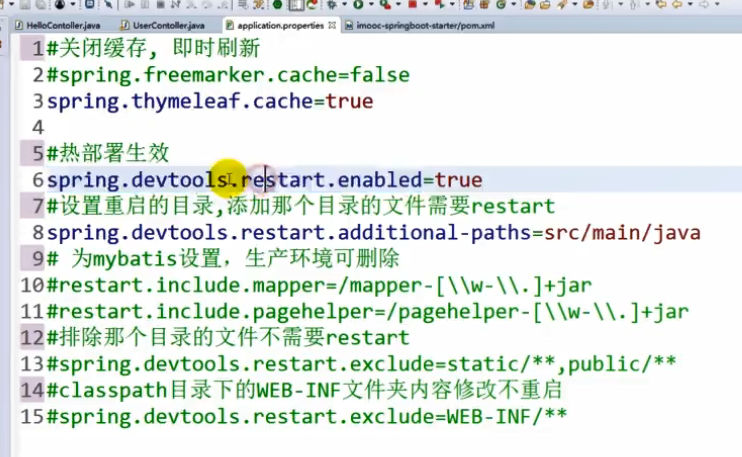
[SpringBoot整合Redis 27](#_Toc28659)

[嵌入式Servlet容器配置修改 29](#_Toc11808)

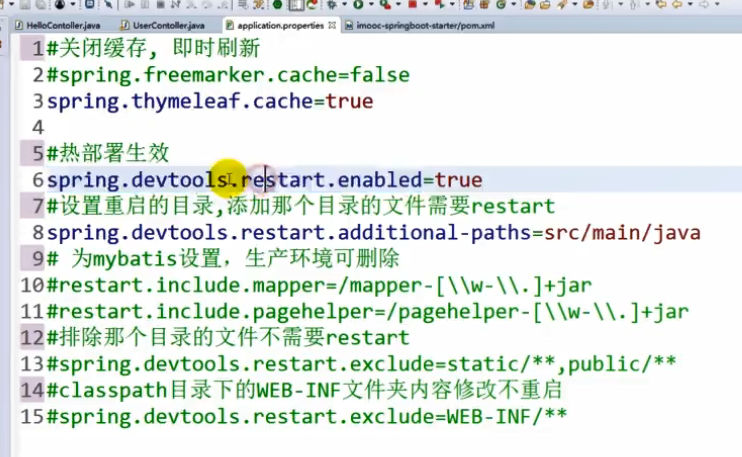
@RestController 相当于 @Controller +@ResponseBody注解组合

但是 用@RestController注解 视图解析器就不会起作用 会返回一个json格式的数据

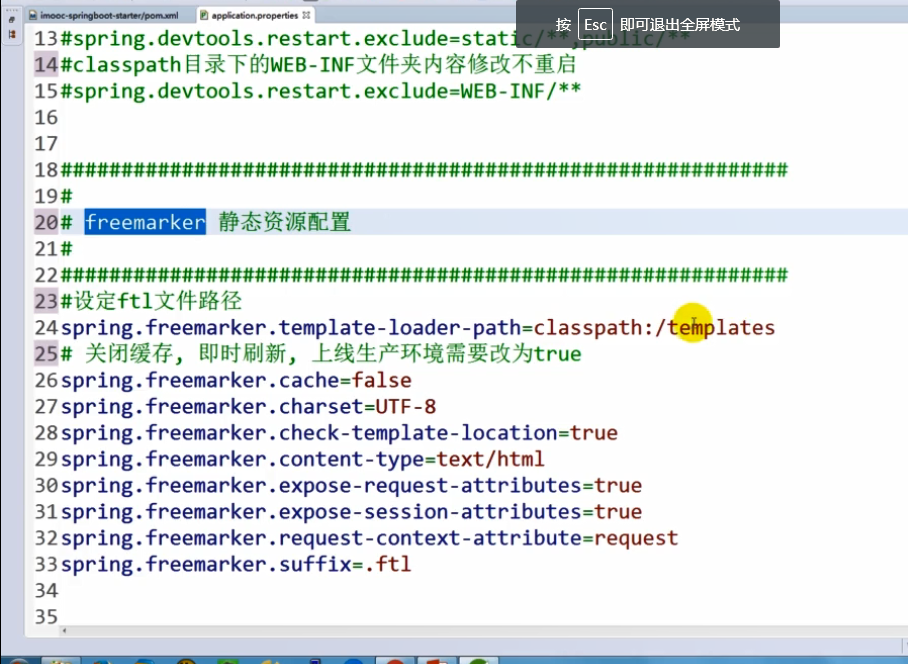
Springboot 相关配置



SpringBoot 热部署相关



Springboot 整合 freemaker 相关配置



Springboot 整合 thymeleaf 的相关配置



## SpringBoot常用注解

**测试启动 注解**

@RunWith(SpringRunner.class)

@SpringBootTest

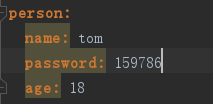
**@ConfigurationProperties**

此注解只能 获取 springBoot 全局配置文件中的 值（application.yml/properties）

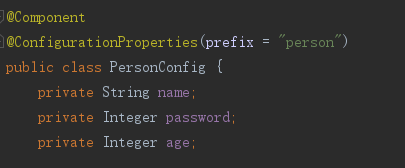
该注解的属性 prefix 支持前缀获取

将配置文件 yml .properties 等的值注入的到 实体属性

比如 yml配置文件 有



将 值 注入到 对应的属性中



该注解 支持JSR303校验 类上加@Validated

我们可以导入配置文件处理器，以后编写配置就有提示了



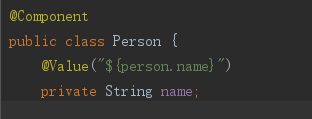
**@Value**

将配置文件中值注入到属性 该注解不支持数据校验

支持 1 ${} 将配置文件中的值注入 （比如获取类路径下 properties的配置文件的内容）

将 配置文件 person.name 的值 注入

注意 ： @Value注解 也可以 用在方法 参数上， 将 配置文件中的值 注入到参数中



2 #{} SpEL 表达式注入

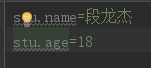
3 直接写字符串

复杂类型封装 不支持 eg:Map类型

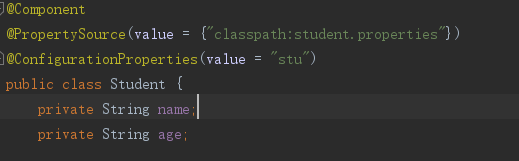
**@PropertySource(value=”clsspath:...”)**

加载指定 路径下的配置文件 可以将配置中的值赋值到bean里.

这个注解只是 指定配置文件的位置，要实现赋值还得用 @ConfigurationProperties 注解



// 这 @ConfigurationProperties 的 value 属性和 prefix 属性的区别



@ImportResource(locations={“”}) 再 启动类上 使用 (springboot不推荐这样配置)

指定 spring 配置文件（xml文件）所在路径, 这样 配置文件才能生效

**@Configuration**

指定类为配置类 (springboot 推荐用注解方式 配置)

**@Bean**

修饰一个 方法 返回值 就是 bean的类型 ,方法名 就是 id

也可以通过 @Bean注解 的属性 指定 bean的Id

@ResponseBody

在Restful 风格中 该注解 返回 json 或者字符串

### SpringMVC中注解

**@RequestMapping**

@RequestMapping 的 consumes produces headers 属性用法尚不清楚

value 是默认属性 可以省略

method = RequestMethod.POST :指定该方法 处理哪些Http请求 也可以同时指定多个请求

没有指定请求方式的话 可以 处理任意的http请求

(地址栏访问是 get请求，提交表单 可以指定请求方式)

超链接 是get请求

params 属性: 请求中 必须包含指定参数时，该方法才处理

eg: http://localhost:8080/pra?username=admin&password=123 只有这个请求 该方法才处理

@RequestMapping(value = "pra",params = {"username=admin","password=123"})

SpringMVC的请求方法参数

请求方法可以出现的类型参数 很多

eg HttpServletRequest request,Model model 等等

每次调用请求方法时(@RequestMapping标注的方法) ，springMVC都会 创建其对象

**Model ModelMap(前两个差不多) ModelAndView**

**@RequestParam 将指定的请求参数赋值给 方法形参 defaultValue 默认取值**

@RequestParam(value = "username",defaultValue = "tom",required = true) String name

**@PathVariable 获取请求URL 中的动态参数 如：{name}**

**value 如果省略 默认绑定 同名 动态参数**

@RequestMapping("login/{name}")

public String login(**@PathVariable**(value = "name") String name){

return name; //动态跳转页面

}

@RequestHeader 将请求头信息数据 传递到方法形参上 eg:

@RequestHeader("User-Agent") String userAgent,

@RequestHeader(value = "Accept") String[] accepts

**@CookieValue**

@CookieValue(value = "JSESSIONID",defaultValue = "") String sessionId

自动将JSESSIONID的值设置到 sessionId 形参上

defaultValue 表示 Cookie中没有 JSESSIONID时 默认为空

**@ModelAttribute**

此注解注释的方法 会在 该 Controller 中的每个方法执行前执行

(1)

这相当于 model.addAttribute("user",user);

value值 相当于 model的key 方法返回值 就是 model的value

@ModelAttribute(value = "user")

public User m1(){

User user=new User();

user.setPassword("123");

user.setUsername("tom");

return user;

}

(2)

在这个 Controller 中 无论访问哪个 方法

都会先调用 @ModelAttribute 注解修饰的方法

所以 也会接收 其他方法 能接收到的 页面传入的参数

@ModelAttribute

public voidme2(User user, Model model){

model.addAttribute("user",user);

}

@RequestMapping("me3")

public String me3(Model model){

// 获取 Model 已经 add 过的 obj

User user =(User) model.asMap().get("user");

return "html/login";

}

**@SessionAttributes**: 指定Model 中的哪些属性 转存到HttpSession中 此注解只能声明在类上

**@RequestBody** （作用在 javaBean 前面）获取 ajax 传入的json数据。将的json数据 封装到相应的 javaBean中

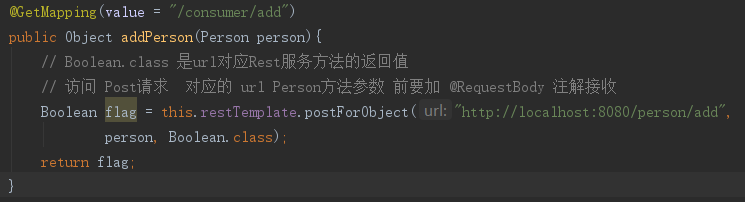
**@ResponseBody** (作用在方法前面) 将 javaBean 或者是 集合数据 转换为json格式 返回客户端

**@RequestBody** 可以直接将 对象作为参数发送，不需要 url路径恩地拼接，或者通过表单的方式

举个例子 :

public void add(@RequestBody Person person){}

// 下面是 springboot Rest远程 服务调用的一个例子



## Springboot 支持 多环境测试 Profile

配置文件名: application-{profile}.yml/properties

application.yml/properties 是 主配置文件(默认配置文件)

profile 名字随便取 : 比如 dev 开发环境,prod 生产环境

**这些不同环境下配置文件的激活:**

1

如果 时 properties 配置文件,配置 多个文件时,在application.properties

中 使用 spring.profiles.active=dev .. 方式激活 （值为 profile名称）

2

如果 是yml文件 ,可以用 文档块 --- 来分割文档块



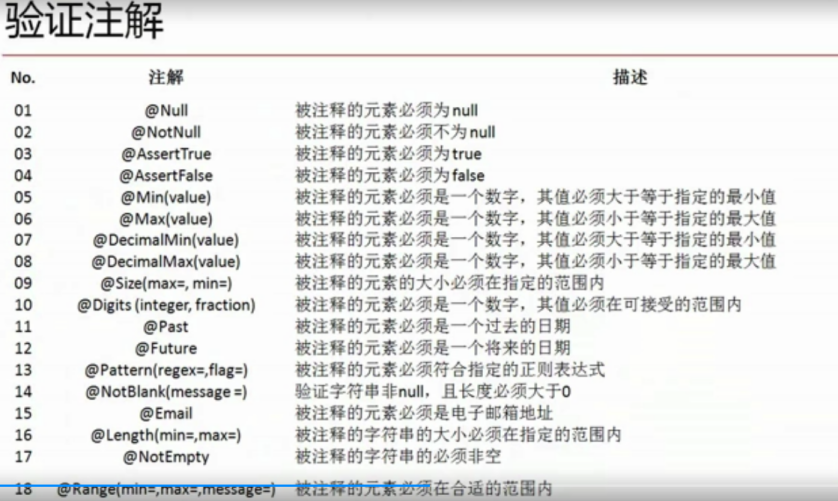
3 命令行 激活 动态切换profile （生产环境下 ，项目已经打包，发布）

比如 在 配置文件中 active=dev

执行 命令 java -jar \*.jar --spring.profiles.active=prod ：以prod 配置启动

## SpringBoot默认支持的数据校验

这些是hibernate的校验注解



注解作用在属性上

错误的 提示信息统一放到 一个 properties文件中 ，用注解的messge属性 引用

需要校验的 实体参数 要用 @Valid 注解修饰



这种 错误校验 太过繁琐，实际上可以用拦截器 校验

## SpringBoot 一些常用接口

### 初始化相关

### CommandLineRunner

在使用SpringBoot构建项目时，我们通常有一些预先数据的加载。 接口CommandLineRunner 实现改功能

### WebMvcConfigurer

扩展 springMVC 功能 （拦截器啊 等等） 保留了spring boot的自动配置

@EnableWebMvc 全面接管 MVC 的配置, spring boot MVC 模块的一些自动配置 将 失效

该注解 不推荐使用



#### Springboot 主配置文件的加载位置

file/config/ file/ classpath:/config/ classpath::/ 加载的优先级 从高到低

## Springboot 与 日志框架的选择

(日志 记录系统的 执行行为 )

**Springboot 选用的 是SLF4j 门面(抽象层,接口),和 logback 具体日志框架的实现**

首先导入 SLF4J 的jar 和logback 实现jar

调用 SLF4J 接口的 方法 , 用的其实是 logback 的 实现

SLF4J 的实现框架很多 , logback 只是其中之一



历史 遗留问题

很多框架 (mybatis ，hibernate等)用的 不是 SLF4J 日志, 要 同一 日志框架,需要适配

1 将其他框架排除

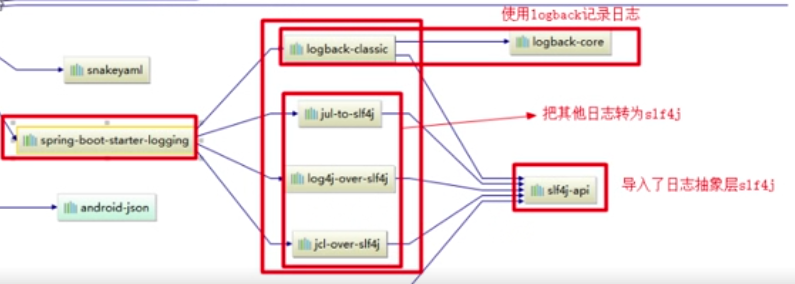
2 用中间包 替换原来的 日志框架 (偷梁换柱包)

3 导入SLF4J 其他的实现 把 其他日志框架排除

Springboot 考虑到了 日志 不统一的问题 该依赖解决日志不统一问题



也可以使用 log4j2



**Logback的使用**

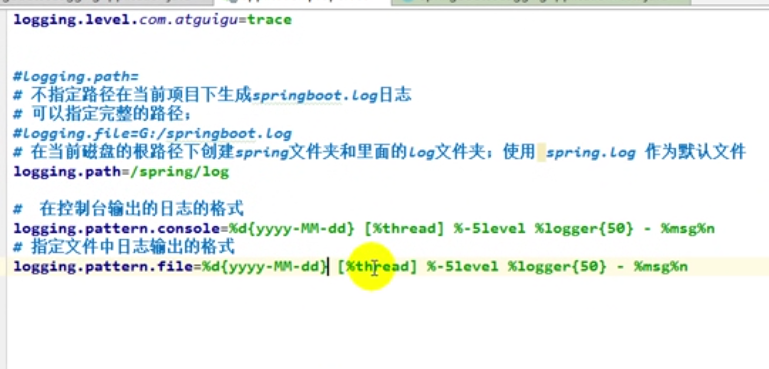


Logback 在主配置文件中的相关配置

com.atguigu 是包名 ，意味着 该包下的所有日志级别 是 trace级别

可以指定 多个包下的日志级别。

只有 该级别 及以上级别的日志 才会打印



在框架中 可以指定 自己的 日志配置文件

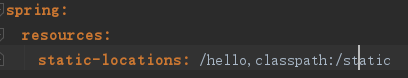
## Springboot 与 web 模块

<https://www.webjars.org/> springboot 静态资源 maven依赖

**静态资源存放路径 (springboot默认)**



修改 静态资源路径

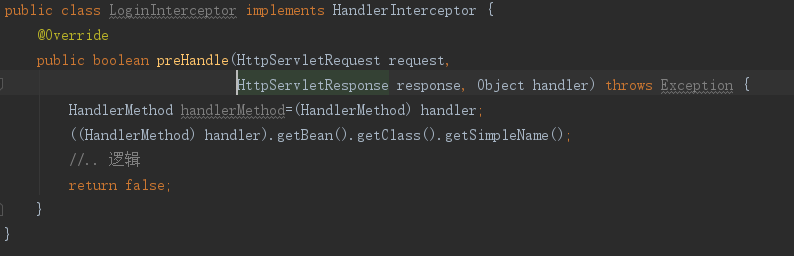


**favicon.ico: 页面的图标 把这个图标放到静态资源路径下**

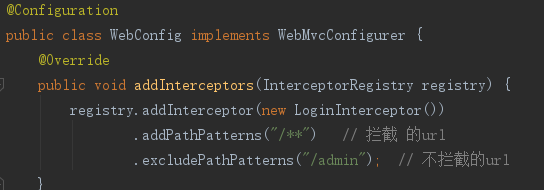


### 拦截器

拦截器 实现 HandlerInterceptor 接口 （新版的 方法 是默认方法 可以不用全部实现 java8 新特性）



拦截注册



Aop 依赖

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

</dependency>

切面

@Component

@Aspect

public class ServiceAspect {

//切点， 每一个controller请求方法

@Pointcut("execution(public \* com.zycz.myproject.controller..\*.\*(..))")

public void pointCut(){ }

/\*\*

\* before advice 前置增强处理

\* @param joinPoint 连接点

\*/

@Before(value = "pointCut()")

public void doBefore(JoinPoint joinPoint) { }

/\*\*\*

\* after returning advice 后置返回增强处理

\* @param joinPoint 连接点

\* @param returnMsg return返回的信息(就是response)

\*/

@AfterReturning(pointcut = "pointCut()",returning = "returnMsg")

public void doAfterReturn(JoinPoint joinPoint,Object returnMsg) { }

/\*\*\*

\* 后置异常增强处理

\* @param joinPoint 连接点

\* @param exception 目标方法抛出的异常与增强处理的异常一致才执行，否则不执行，

\* 如果是throwing对应的是Throwable类型将匹配任何类型异常

\*/

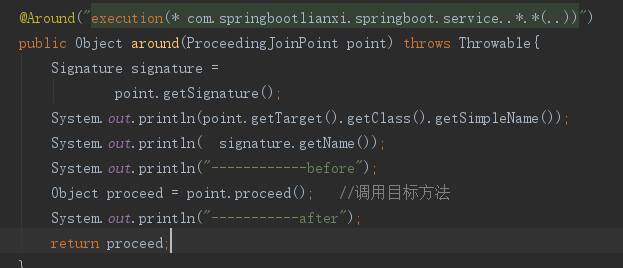
@AfterThrowing(pointcut="pointCut()",throwing = "exception")

public void doAfterThrow(JoinPoint joinPoint, Throwable exception) {

}

}

环绕通知 可以实现 service 的 日志处理等



魔乐科技 看完 第 8 章

### 配置druid 数据库链接池

### https访问控制

用CA 认证

1 使用keytools 创建证书

打开cmd 输入下面的命令

keytool -genkey -alias tomcat -keyalg RSA -keystore ./server.keystore

然后按照提示操作

可在用户 目录下生成keystore文件

2

将keystore 文件 放到项目根目录

在application.yml 配置文件中，配置ssl安全访问

server.port=443 https 默认端口443

server.ssl.key-store=server.keystore

server.ssl.key-alias=tomcat

server.ssl.enabled=true

server.ssl.key-store-password=11111

server.ssl.key-store-type=JKS

3 http访问 自动转 https访问 将一个配置类 注入下面两个bean

@Bean

public Connector connector(){

Connector connector=new Connector("org.apache.coyote.http11.Http11NioProtocol");

connector.setScheme("http");

connector.setPort(80);

connector.setSecure(false);

connector.setRedirectPort(443);

return connector;

}

@Bean

public TomcatServletWebServerFactory tomcatServletWebServerFactory(Connector connector){

TomcatServletWebServerFactory tomcat=new TomcatServletWebServerFactory(){

@Override

protected void postProcessContext(Context context) {

SecurityConstraint securityConstraint=new SecurityConstraint();

securityConstraint.setUserConstraint("CONFIDENTIAL")；

SecurityCollection collection=new SecurityCollection();

collection.addPattern("/\*")；

securityConstraint.addCollection(collection);

context.addConstraint(securityConstraint);

}

};

tomcat.addAdditionalTomcatConnectors(connector);

return tomcat;

}

### SpringBoot项目的 打包，发布

下面的插件 实现 打包发布

<packaging>jar</packaging> pom文件jar包就行

<plugin>

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

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

</plugin>

mvn package : 将springboot 项目达成 jar包 （taget目录下）

这个jar包 随便放 哪个目录下都行 （linux系统下也可以）

用 java -jar \*.jar 命令就可以启动 springboot项目

因为内置了 web容器， 所以不需要 打成war包 部署到，tomcat的 webapps目录了

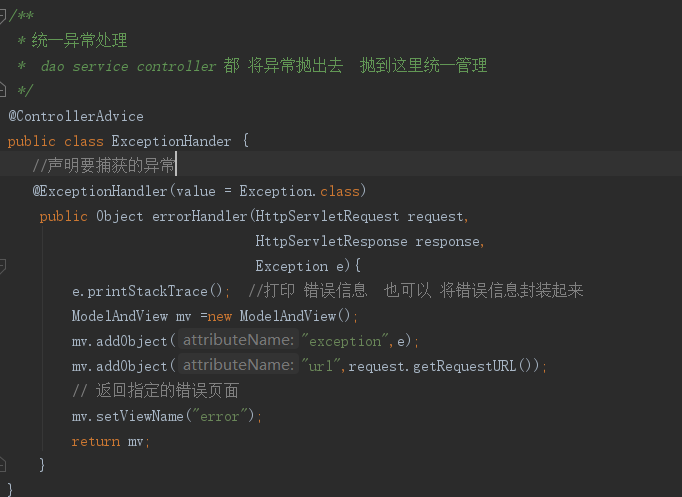
如果不是springboot 项目，而是用 maven 搭建的 web项目，要实现打包发布

要先 用 mvn package 打成war包 ，再将war包部署到 tomcat的 webapps文件夹下

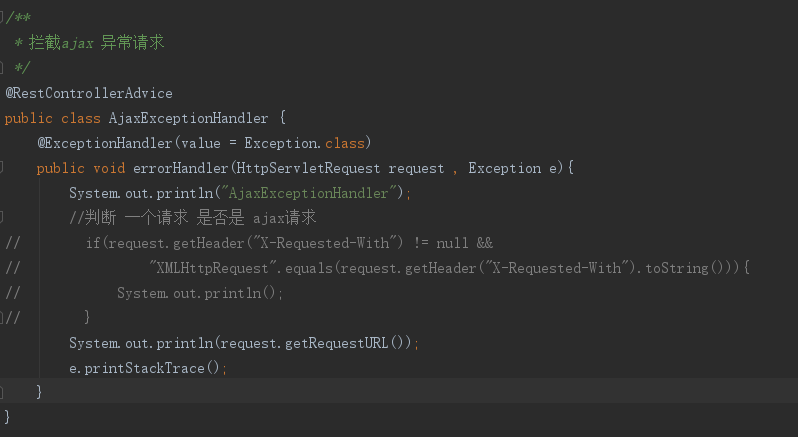
在windows 命令行 ： startup 启动tomcat

### SpringBoot异常处理

捕获异常的 类



Ajax 异常处理类



### SpringBoot配置错误页面

404 页面找不到等

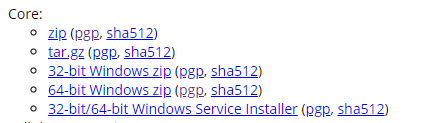
根据http状态码，跳转到指定的错误路径



### SpringBoot 打成war包 部署到tomcat

Tomcat 在windows环境下的 安装和配置 ：

<https://tomcat.apache.org/download-80.cgi> 下载tomcat



选择 最后一个 （最后一个可以在 windows 下 后台启动。因为bin 目录下有service.bat服务）

配置 ：

添加 系统变量 ：

TIM截图20181014130149

在path 系统变量 追加 %CATALINA\_HOME%\bin;%CATALINA\_HOME%\lib;

配置 完成 ： 在 cmd 下 运行 startup 验证 tomcat 配置成功！

SpringBoot默认达成jar包， 使用内置的Tomcat。但考虑到项目需要集群部署或者进行优化时，就需要打成war包部署到外部的Tomcat服务器中

1 : pom.xml <packaging>war</packaging>

2 : 添加依赖

<dependency>

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

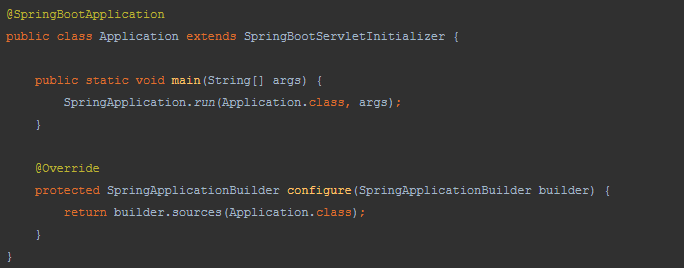
<artifactId>spring-boot-starter-tomcat</artifactId>

<!--打包的时候可以不用包进去，别的设施会提供。事实上该依赖理论上可以参与编译，测试，运行等周期。 相当于compile，但是打包阶段做了exclude操作-->

<scope>provided</scope>

</dependency>

3 启动类 继承SpringBootServletInitializer，实现configure方法



为什么继承该类？

如果您正在构建WAR文件并部署它，则需要WebApplicationInitializer。

注意 ： 使用外部tomcat 部署访问的时候 （将项目打成 war包），

application.yml中的

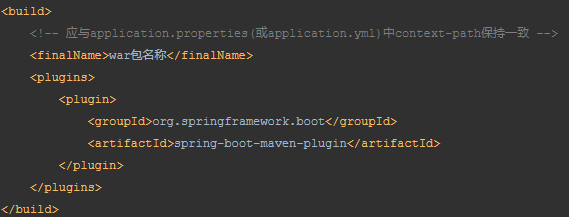
server.port=

server.servlet.context-path= 将失效

使用 tomcat 端口，**webapps下的项目名称进行访问**

为了防止应用上下文所导致的项目访问资源加载不到的问题，

建议pom.xml文件中<build></build>标签下添加<finalName></finalName>标签：



mvn clean package : 打成war包

将 war包 考到 webapps下后，运行：${Tomcat\_home}/bin/目录下执行startup.bat(windows)或者startup.sh(linux) 启动 或者 windows下 startup 启动

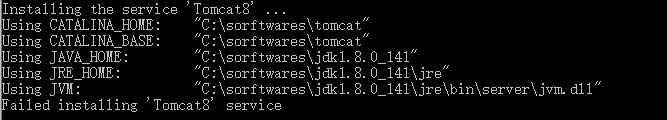
starup 启动属于 前台启动，有个启动界面

**Tomcat的 后台启动方式：**

cmd 下运行 service.bat install

然后在 此电脑 -> 管理 -> 服务和应用程序-> 服务 ->找到Apache Tomcat 手动启动即可

如果 执行 service.bat install 出现 一下 错误



执行 service.bat remove 再执行 service.bat install 即可

### 文件上传

springBoot 已经引入 fileupload 文件上传包

form表单的 method 需是 POST，enctype 属性值为 multipart/form-data

<form action="/file" method="post" enctype="multipart/form-data">

<input type="file" name="file01"/>

</form>

处理文件上传的方法 处理POST请求，用MultipartFile 类型接收

equired = false 此参数 表明此参数 不是 必须的

@RequestMapping(value = "file",method = RequestMethod.POST)

public void fileHandler(@RequestParam(value = "file01",required = false)

MultipartFile file){

System.out.println(file.getName());

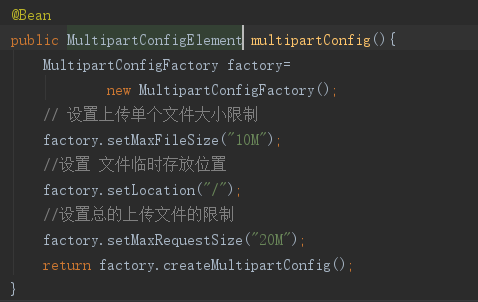
System.out.println(file.getOriginalFilename());

}

**上传限制**

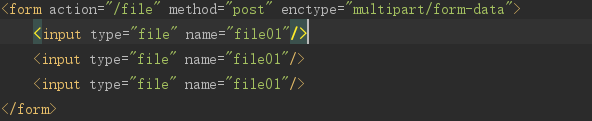
上传 文件的大小不允许 无限大 ，springBoot默认设置的有限制

上传 文件 限制 的配置类 在yml 中配置文件上传限制 过时了

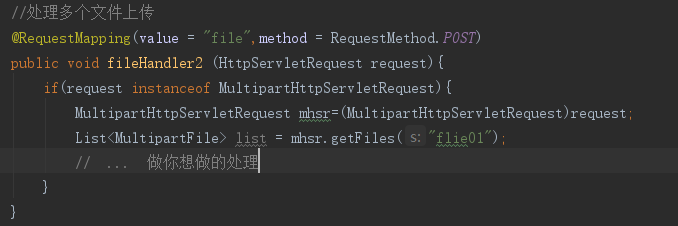


**上传多个文件**

首先表单 有多个上传文件



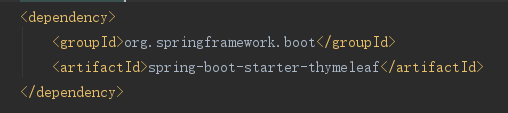
控制器接收参数 为 request



图片服务器 第七章有讲

## Springboot 和 thymeleaf

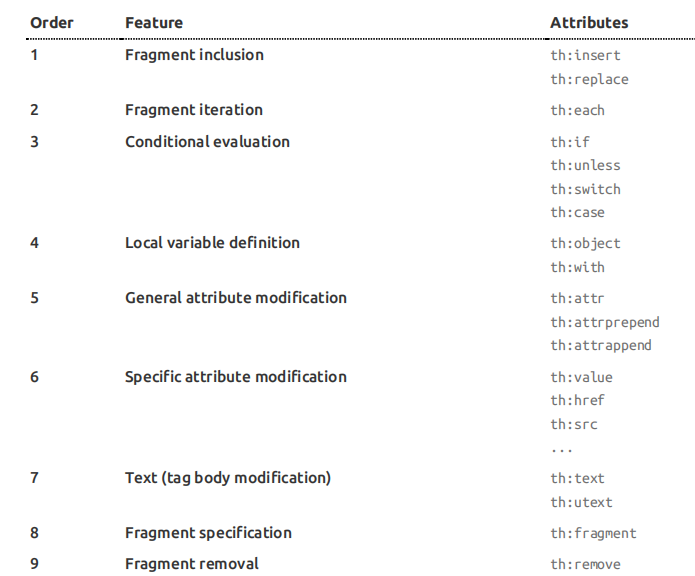
SpringBoot 整合 thymeleaf依赖





只要把 页面 放到 classpath:/templates ,thymeleaf 就能自动渲染

Thmeleaf 属性的优先级



行内写法 .可以获取 放到域中的值

[[${name}]]

**Thymeleaf 的模板 复用 (using thymeleaf 的 模块8 Template Layout)**

1 首先 定义 被引用的 公共模板 (如 左侧的 导航栏)

在模板的标签上 写 th:fragment="copy">

在 别的页面 用标签 footer: 是模板所在的 页面的名称除去 扩展名 (与 前缀 后缀有关)

copy: 是定义的 fragment 的名称

TIM截图20180927150315

也可以用 选择器的形式 (和 css 选择器一致)

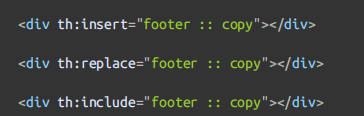
如 <div id=”tem”></div>

引用可以这样 <div th:insert=”~{footer : : #tem}””></div>

Insert: 是在 div基础上 插入 模板

replace : 模板 替换div

Include div 包含 模板的内容:



因此,可以将公共 模板 单独 抽取出来 放到一个页面,共其他页面引用

SpringMVC 的扩展: (扩展 拦截器 , viewController等等等)

**WebMvcConfigurer**

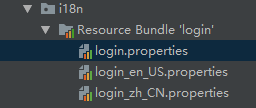
实现 WebMvcConfigurer 接口 ,配置类用 @Configuration 修饰

若使用 @EnableWebMvc 注解修饰 ,springboot 的一些 默认 springmvc 配置将 失效

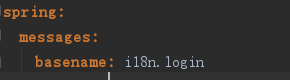
(自己全面接管 springMVC 的配置)

**国际化: (支持 浏览器 不同文字)**

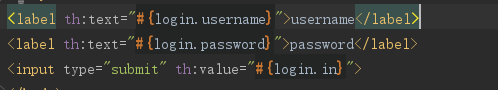
1: 配置 国际化 相关文件 (spring boot 已经做好了相关自动配置)



2 spring.message.basename=i18n.login 配置



3 页面 用 #{} 对国际化配置进行引用



也可以 自定义 处理 国际化 LocalResolvor

### Restful风格

Restful 是一种软件架构风格，是面向资源的

@PostMapping Restful 风格 == @RequestMapping(method=RequestMethod.POST)

Restful 风格 还有 @GetMapping @PutMapping @DeleteMapping

RESTful Web 服务天然具备如下的属性:

1 客户端-服务器: UI 与数据存储分离 。。。。。

2 无状态 ：每个请求会包含服务器所需的足够信息，无需维护状态，就你进行操作

3 可缓存 ：

4 统一接口：URL会唯一识别资源，能够通过超链接发现API

5 分层：API的每个资源都提供了合理程度的细节

Richardson 成熟度模型

Lenard Richardson 定义了 4 个等级 0-3级，它们描述了Web API的RESTful程度。

0 级： HTTP 只需让资源在网络上通过http协议获取即可

1 级 ： 资源。资源是模型中某个元素的唯一标识，借助http，资源会与一个统一资源标识符URL 进行关联。

如 ： /users 所有用户的列表 。 /user/12 ： 包含特定用户

2 级： HTTP动作。 此级别用HTTP动作，来识别资源的行为，能够很好地描述API的功能 。 主要的动作 ：

GET ：在一个特定的URL上读取数据

DELETE ： 删除某个资源

PUT ：更新或创建资源

POST ： 更新或创建资源

PATCH ： 部分更新资源

OPTIONS ：返回服务端针对特定资源所支持方法列表

3 级：超媒体控制

API 版本化 ：

有用的HTTP代码

RESTful API还有一个很重要的方面，就是合理的使用HTTP 编码



## SpringBoot整合Restful 架构

Spring与Rest的整合是微架构的核心。Rest是整个微架构之中进行通信的基础模式。

Rest 的核心是用 json交换 数据。**Spring提供RestTemplate模板实现**，Rest的JSON数据与各种对象的自动转换。Spring 默认用 jackson 实现 json和对象间的对象转换。

**使用RestTemplate模板实现Rest的服务调用**

**（所谓的Rest服务，就是Controller的一个 方法）**

由于Rest属于分布式的项目开发环境。所以本次进行项目建立3 个子模块，进行实验

1 springboot-restful-api:作为公共的类定义，例如将所有的VO类定义在此项目之中

2springboot-restful-provider :作为服务提供者，本次练习 服务提供者提供2个服务

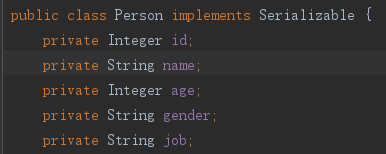
（获得对象，增加对象）

3 springboot-restful-consumer : 服务的消费者，消费者就是利用RestTemplate实现Rest

服务调用以及对象的转换

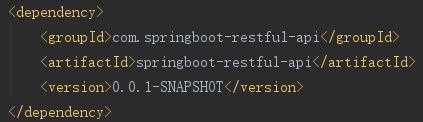
Rest 服务调用 简单的代码 示例 （三个模块都是 springboot项目）

springboot-restful-api 模块: 定义一个全局的VO类 Person



springboot-restful-provider 模块 （端口设置为 8080）

Pom 文件 引入 restful-api 模块



一个简单 Rest 服务



Springboot-restful-consumer 模块 ： （端口设置为 80）

同样引入 restful-api 模块

利用 RestTemplate 远程调用 Provider 模块的Rest 服务



Rest的访问流程



## 配置Swagger 服务

**认识 Swagger**

Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。总体目标是使客户端和文件系统作为服务器以同样的速度来更新。文件的方法，参数和模型紧密集成到服务器端的代码，允许API来始终保持同步。

**Swagger作用**

1 接口的文档在线自动生成。

2 功能测试。

**Swagger使用的注解及其说明**

@Api：用在类上，说明该类的作用。

@ApiOperation：注解来给API增加方法说明。

@ApiImplicitParams : 用在方法上包含一组参数说明。

@ApiImplicitParam：用来注解来给方法入参增加说明。

@ApiResponses：用于表示一组响应

@ApiResponse：用在@ApiResponses中，一般用于表达一个错误的响应信息

l code：数字，例如400

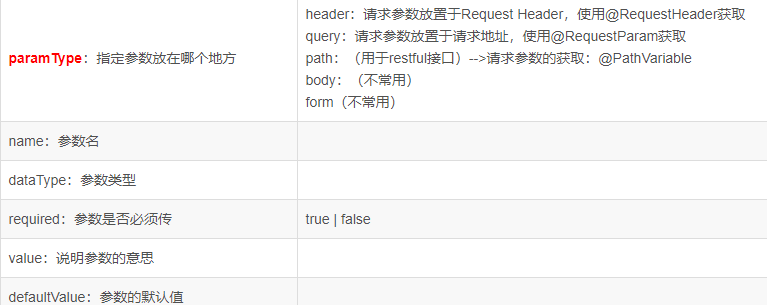
l message：信息，例如"请求参数没填好"

l response：抛出异常的类

@ApiModel：描述一个Model的信息（一般用在请求参数无法使用@ApiImplicitParam注解进行描述的时候）

l @ApiModelProperty：描述一个model的属性

**@ApiImplicitParam的参数说明**

****

**建立一个swagger的小案例**

1 新建有一个 swagger 模块

1 引入 swagger2 的 依赖

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.8.0</version>

</dependency>

Swagger-ui 依赖

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.7.0</version>

</dependency>

2 定义 swagger2 的配置类

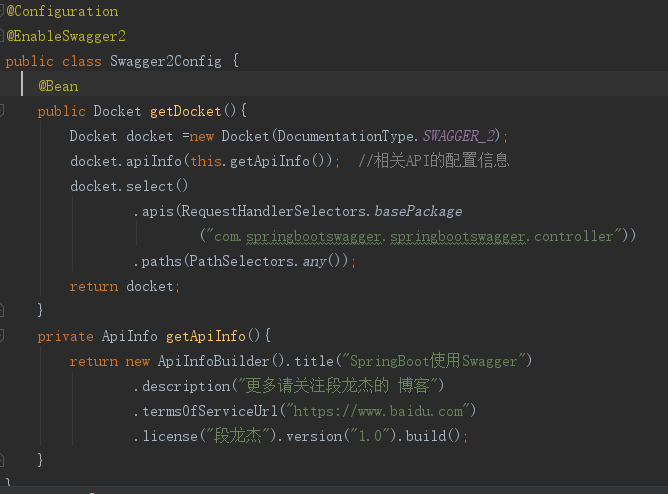
核心配置

此类是整个Swagger 的配置项，需要这个类指派扫描包

select().apis(RequestHandlerSelectors.basePackage

("com.springbootswagger.springbootswagger.controller"))

此 代码 就是 指定 Rest服务所在的包名



Controller 实列

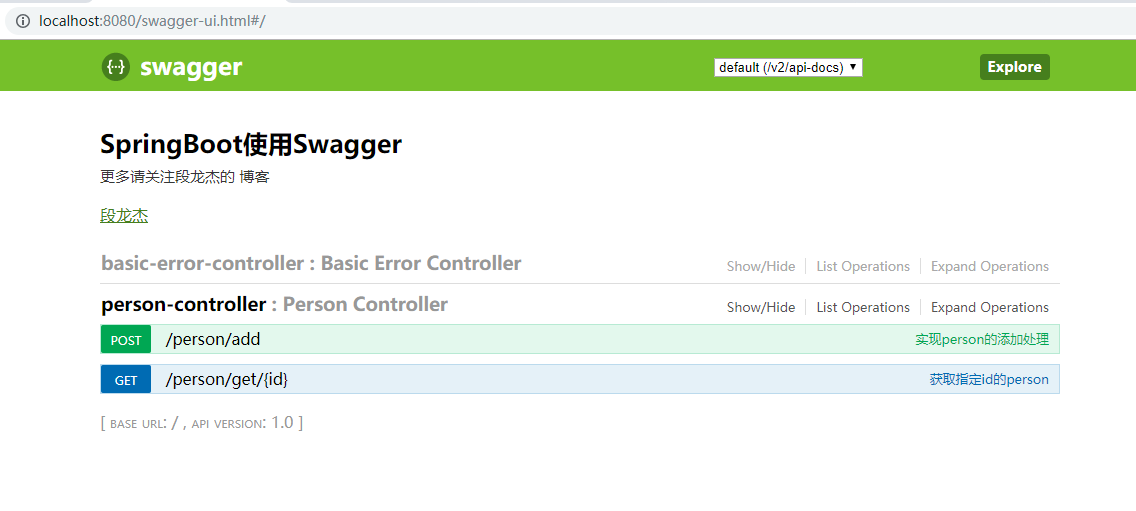


启动之后

地址栏 访问

<http://localhost:8080/swagger-ui.html>

出现 类似 下面 的 页面 （完成！）



## AbstractBaseController的作用

开发技巧:

开打期间,模板引擎页面修改后 要实时生效

1 禁用 模板缓存 spring.thymeleaf.cache=false 2 页面修改 完成后 ctrl + f9

从 等路页面到 别的页面,用 重定向可以 防止表单重复提交

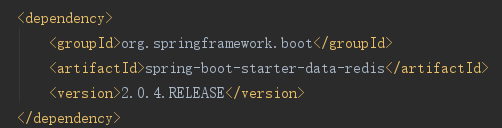
Spring boot 也可以 注册 web 3 大组件(servlet , filter,Listener)

## SpringBoot整合Redis

Redis数据库的整合java里面提供官方工具包 jedis。

RedisTemplate 操作

1 添加 redis依赖



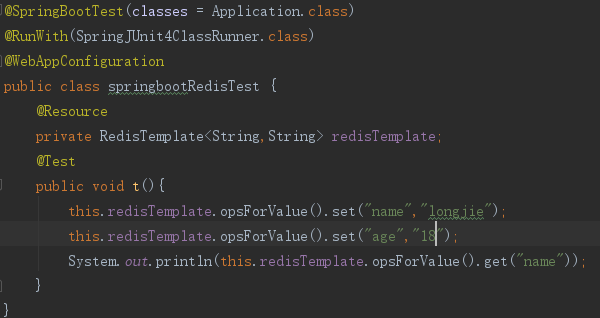
2 要想进行 redis操作，要修改application.yml 配置文件。

下图是 springBoot 关于redis 的简单配置



3 可以通过程序来利用RedisTemplate 模板来进行数据处理了

下面是简单的代码测试 。 向redis数据库 插入记录

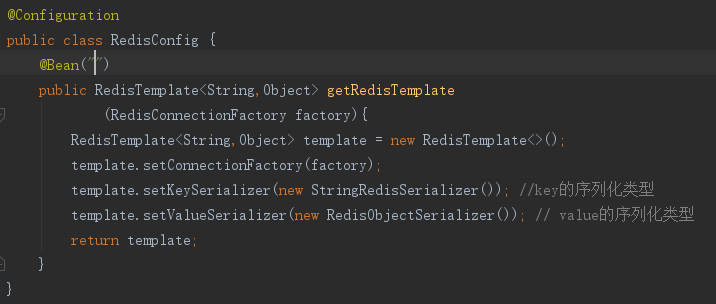


Redis 序列化处理

对象的序列化保存,将对象持久化

1 首先准备一个序列化处理程序类。这个类的实现方式：

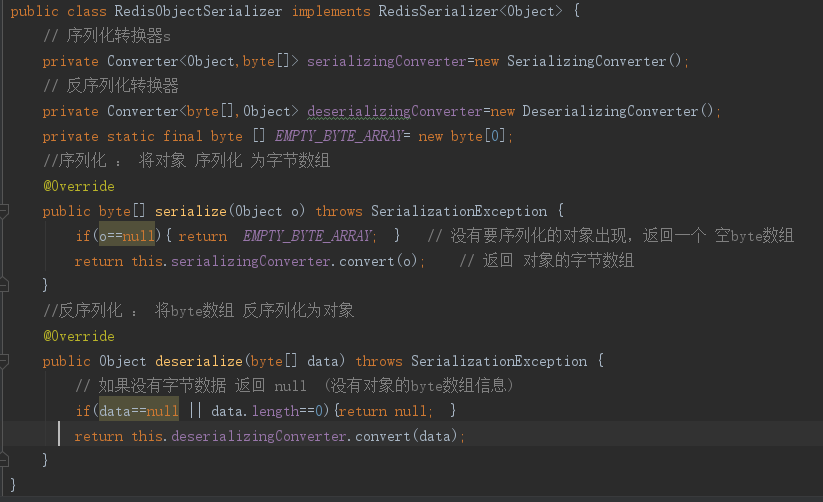
下图是一个 序列化程序类的示例

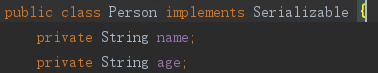


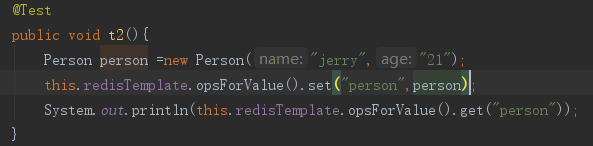
泛型 Object表示，可以序列化所有类型的对象。

当然 要序列化的类，要实现Serializable 接口

为了方便进行对象与字节数组的转换，所以应该先准备2 个转换器





****

**配置多个Redis 连接 （有机会在看，没太懂、）**

配置多个RedisTemplate

需求：由于在项目的实际开发过程中Redis的使用会非常频繁，那么 就需要连接多个Redis数据库。SpringBoot里面针对Redis 的连接配置本质上只提供一个连接配置项。

要想有 多个Redis连接配置，就需要自己来进行Redis的创建管理

## 嵌入式Servlet容器配置修改

SpringBoot 默认使用Tomcat作为嵌入式的Servelt容器

1 修改和server有关的配置 （在主配置文件 application.yml/properties）

server.xxx