### Http接口请求常用注解

1、GET请求

1、单一参数@RequestMapping(path = "/{id}", method = RequestMethod.GET)

1) public String getUser(@PathVariable String id ) {}

1. @RequestMapping(path="/{depid}/{userid}",method= equestMethod.GET) 可以同时指定多个提交方法

getUser(@PathVariable("depid")String departmentID,@PathVariable("userid") String userid)

3）一个顶俩

@GetMapping = @RequestMapping(method = RequestMethod.GET)

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

@PutMapping = @RequestMapping(method = RequestMethod.PUT)

@DeleteMapping = @RequestMapping(method = RequestMethod.DELETE)

4）@RequestParam(value = "name", required = true)

可以设置默认值，比如分页

4)@RequestBody 请求体映射实体类

需要指定http头为 content-type为application/json charset=utf-8

5）@RequestHeader 请求头，比如鉴权

@RequestHeader("access\_token") String accessToken

1. HttpServletRequest request自动注入获取参数

### 常用json框架介绍和Jackson返回结果处理

简介：介绍常用json框架和注解的使用，自定义返回json结构和格式

1、常用框架 阿里 fastjson,谷歌gson等

JavaBean序列化为Json，性能：Jackson > FastJson > Gson > Json-lib 同个结构

Jackson、FastJson、Gson类库各有优点，各有自己的专长

空间换时间，时间换空间

2、jackson处理相关自动

指定字段不返回：@JsonIgnore

指定日期格式：

@JsonFormat(pattern="yyyy-MM-dd hh:mm:ss",locale="zh",timezone="GMT+8")

空字段不返回：@JsonInclude(Include.NON\_NUll)

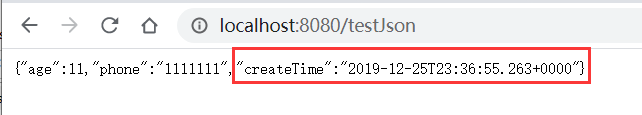
指定别名：@JsonProperty

举例：

|  |
| --- |
| **package** com.example.lchtest.springbootdemo1.domain;   **import** com.fasterxml.jackson.annotation.JsonIgnore; **import** lombok.Getter; **import** lombok.Setter;  **import** java.util.Date;  @Getter @Setter **public class** User {  **private int age**;  *// @JsonIgnore指定的字段不返回* @JsonIgnore  **private** String **pwd**;  **private** String **phone**;  **private** Date **createTime**;   **public** User(){  **super**();  }  **public** User(**int** age, String pwd, String phone, Date createTime) {  **this**.**age** = age;  **this**.**pwd** = pwd;  **this**.**phone** = phone;  **this**.**createTime** = createTime;  } } |
| **package** com.example.lchtest.springbootdemo1.controller;  **import** com.example.lchtest.springbootdemo1.domain.User; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RestController;  **import** java.util.Date; **import** java.util.Map;  @RestController **public class** SampleController {  @RequestMapping(**"/testJson"**)  **public** Object testJson(){  **return new** User(11,**"abc"**,**"1111111"**,**new** Date());  } } |

接口请求结果如下：

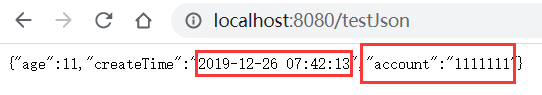
可以看到加了 @JsonIgnore注解的字段没有返回，但是时间的格式不是我想要的格式，



给phone指定一个序列化的别名，给**createTime指定返回的时间格式：**

|  |
| --- |
| @JsonProperty(**"account"**) **private** String **phone**; @JsonFormat(pattern = **"yyyy-MM-dd hh:mm:ss"**,timezone = **"GMT+8"**) **private** Date **createTime**; |

再次请求：



### SpringBoot2.x目录文件结构

##### 1.SpringBoot目录文件结构和官方推荐的目录规范

1、目录讲解

src/main/java：存放代码

src/main/resources

static: 存放静态文件，比如 css、js、image, （访问方式 http://localhost:8080/js/main.js）

templates:存放静态页面jsp,html,tpl

config:存放配置文件,application.properties

resources:

2、引入依赖 Thymeleaf

<dependency>

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

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

</dependency>

注意：如果不引人这个依赖包，html文件应该放在默认加载文件夹里面，

比如resources、static、public这个几个文件夹，才可以访问

3、同个文件的加载顺序,静态资源文件

Spring Boot 默认会挨个从

META/resources > resources > static > public 里面找是否存在相应的资源，如果有则直接返回。

Why? 这是因为源码中就是这么定义的，参考

<https://blog.csdn.net/qq_34797335/article/details/80194137>

4、默认配置

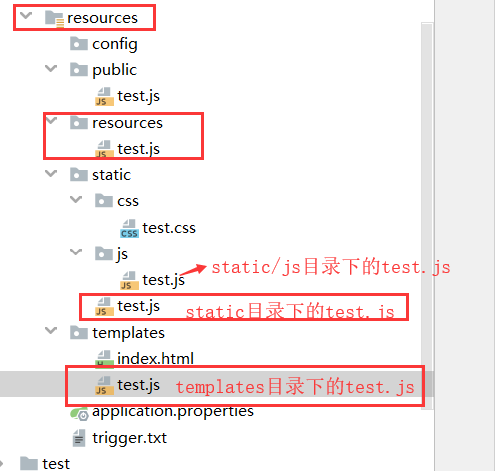
1)官网地址：

https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-developing-web-applications.html#boot-features-spring-mvc-static-content

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

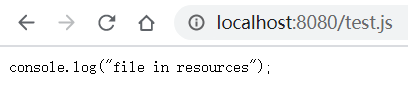
##### 2.资源文件加载目录顺序测试

在src/main/resources目录下创建resources、public目录，分别创建test.js文件，打印所属目录

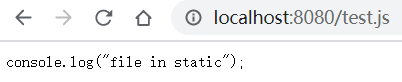


启动项目访问<http://localhost:8080/test.js> 进行测试：

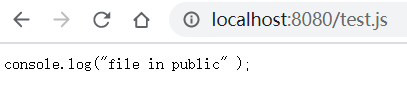
第一次访问：获取到的是src/main/resources/resources目录下的js文件，将 这个文件改名，



第二次访问：获取到的是src/main/resources/static目录下的js文件，将这个文件改名



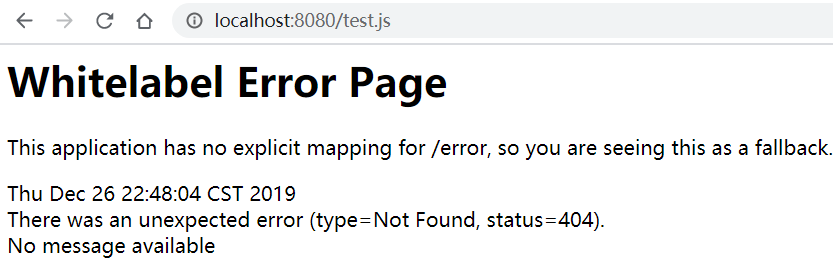
第三次访问：获取到的是src/main/resources/pbulic目录下的js文件



这说明了同名的资源文件，Spring Boot 加载的默认顺序是 resources > static > public

##### 3.访问自定义资源目录下的文件

在前述步骤，将public目录的test.js改名，重新访问：<http://localhost:8080/test.js>，结果如下，发现无法找到文件



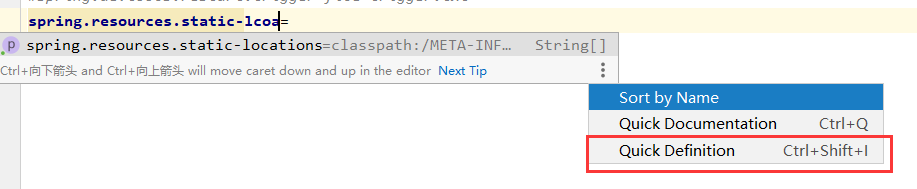
说明访问resources 、static 、public 目录之外的资源，需要手动去配置资源加载路径，在application.properties中添加如下配置，加上**classpath:/templates/**：

|  |
| --- |
| **spring.resources.static-locations**=**classpath:/META-INF/resources/, classpath:/resources/, classpath:/static/, classpath:/public/,classpath:/templates/** |

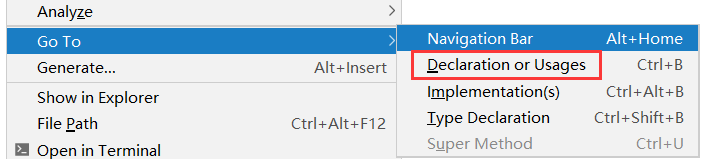
然后再去访问：



Tips：发现一个小技巧，输入配置项值时，停顿一些，会有一个提示，点击三个点，出现Quick Definition,然后在弹出框的界面上，会显示出这个配置项所在的类名，点进去可以查看反编译出来的源码：



或者直接鼠标右键，goto Declartion or usages，就可以打开所在的类ResourceProperties（这是一个配置类）

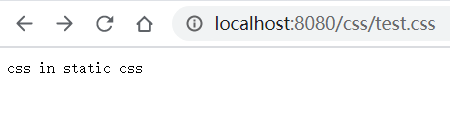


可以看到它里面定义了默认的资源加载路径：

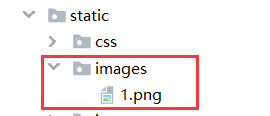
|  |
| --- |
| **private static final** String[] CLASSPATH\_RESOURCE\_LOCATIONS = **new** String[]{**"classpath:/META-INF/resources/"**, **"classpath:/resources/"**, **"classpath:/static/"**, **"classpath:/public/"**}; |

##### 4.访问css/图片

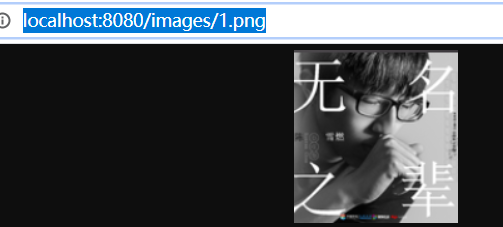
访问css文件： src/main/resources/目录下的resources 和static目录下的文件，访问时，不需要再带上resources static，默认从这两个目录下查找



访问静态图片：在static目录下创建一个images目录（名字任意），访问该目录下的图片：



<http://localhost:8080/images/1.png>



##### 5.访问html页面

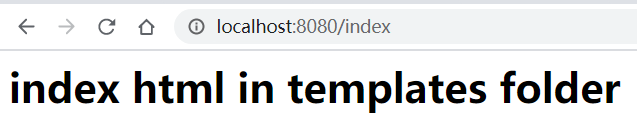
如果直接访问<http://localhost:8080/index.html> ，则会报404的错误，这时在pom中 添加模板引擎thymeleaf依赖，刷新maven依赖，

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**> </**dependency**> |

添加一个controller：

|  |
| --- |
| @RequestMapping(**"/index"**) **public** Object idnex(){  **return "index"**; } |

访问结果如下，也就是说，依赖了模板引擎，springboot会自动把idnex这个逻辑视图名映射到src/main/resources/templates目录下的index.html中去



如果不想用thymeleaf，那么静态页面必须放到src/main/resources/resources 或者src/main/static目录下。

### Springboot文件上传

1. 在staitc目录下创建upload.html（可以直接访问，不用controller跳转）

|  |
| --- |
| <!DOCTYPE **html**> <**html**>  <**head**>  <**title**>uploadimg.html</**title**>  <**meta name="keywords" content="keyword1,keyword2,keyword3"**></**meta**>  <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  <**script src="/js/test.js" type="text/javascript"**></**script**>  </**head**>  <**body**>  <**form enctype="multipart/form-data" method="post" action="/upload"**>  文件:<**input type="file" name="head\_img"**/>  姓名:<**input type="text" name="name"**/>  <**input type="submit" value="上传"**/>  </**form**>  </**body**> </**html**> |

1. 配置文件上传大小和目录:application.properties

|  |
| --- |
| *#文件上传路径* **file.upload.path**=**D:/opt/temp/** *# 设置上传文件大小 spring-boot-autoconfigure-2.2.2.RELEASE.jar/MET-INF目录下找到spring.factories,搜索multi,找到 MultipartAutoConfiguration # 然后点进去这个类，上面有个注解@EnableConfigurationProperties({MultipartProperties.class})， # MultipartProperties这个类 就是上传文件的配置类：ConfigurationProperties注解指定了前缀 @ConfigurationProperties( # prefix = "spring.servlet.multipart", # ignoreUnknownFields = false #)* **spring.servlet.multipart.max-file-size**=**1000MB spring.servlet.multipart.max-request-size**=**1500MB** |

**spring.servlet.multipart.max-file-size  
spring.servlet.multipart.max-request-size**

**两个参数如果不设置，springboot默认只能 上传10M以内的文件，超过会报错**

1. Controller编写：

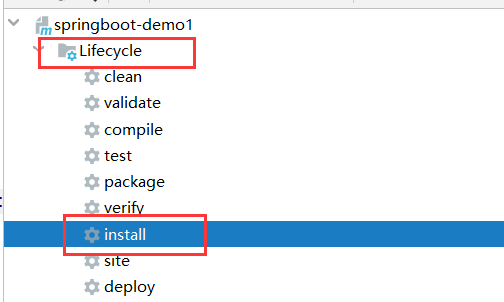
|  |
| --- |
| **package** com.example.lchtest.springbootdemo1.controller;  **import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RequestParam; **import** org.springframework.web.bind.annotation.ResponseBody; **import** org.springframework.web.multipart.MultipartFile;  **import** javax.servlet.http.HttpServletRequest; **import** java.io.File; **import** java.io.IOException; **import** java.util.\*;  @Controller **public class** UploadController {  */\*\*  \* 可以上传的文件类型  \*/* **private static final** List<String> ***UPLOAD\_SUFFIX\_ALLOWED*** = Arrays.*asList*(**".jpg"**, **".png"**, **".jpeg"**, **".apk"**, **".zip"**);   */\*\*  \* 获取文件上传路径  \*/* @Value(**"${file.upload.path}"**)  **private** String **uploadPath**;   */\*\*  \* http://localhost:8080/upload.html (访问 static/upload.html)  \** ***@param file*** *上传的文件  \** ***@param request*** *http请求，获取其他表单字段  \** ***@return*** *\*/* @RequestMapping(**"/upload"**)  @ResponseBody  **public** Object upload(@RequestParam(**"head\_img"**) MultipartFile file, HttpServletRequest request) {  String name = request.getParameter(**"name"**);  System.***out***.println(**"name="** + name);   Map<String, Object> result = **new** HashMap<>();  String originalFilename = file.getOriginalFilename();   String suffix = originalFilename.substring(originalFilename.lastIndexOf(**"."**)).toLowerCase();  *// 文件校验：后缀名，文件头，文件路径，如果是zip文件，要校验文件个数，个数，边解压边校验单个文件大小以及解压后的文件总大小* **if** (!***UPLOAD\_SUFFIX\_ALLOWED***.contains(suffix)) {  result.put(**"retCode"**, 500);  result.put(**"Msg"**, **"File not allow upload"**);  **return** result;  }   *// 文件重命名* String fileName = UUID.*randomUUID*().toString().replaceAll(**"-"**, **""**) + suffix;  File dest = **new** File(**uploadPath** + fileName);   **try**{  *// 文件路径归一化校验* **if**(!dest.getAbsolutePath().equals(dest.getCanonicalPath())){  *// 文件保存路径异常* }  file.transferTo(dest);  result.put(**"retCode"**, 200);  result.put(**"fileName"**, fileName);  } **catch** (Exception e){  e.printStackTrace();  result.put(**"retCode"**, 500);  result.put(**"Msg"**, **"upload inner error"**);  }   **return** result;  } } |

### Springboot jar包运行web项目

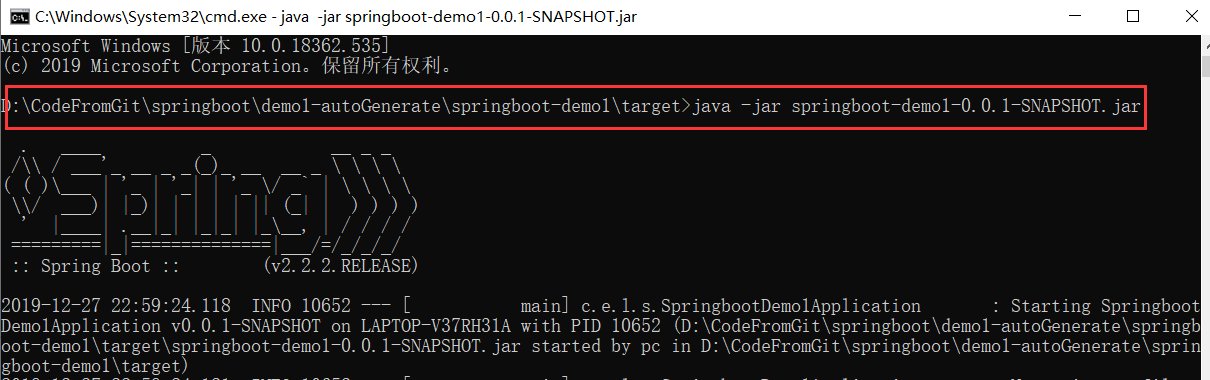
1. 添加打包依赖：

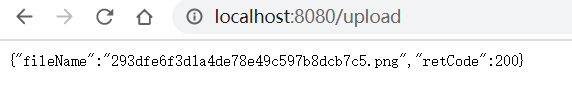
|  |
| --- |
| *<!-- 打包依赖，如果不加，执行maven打包，运行后会报错，no main manifest attribute -->* <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**> </**build**> |

1. 打包：执行Lifecycle下的install即可，它会在target目录下生成打出来的jar包，并拷贝一份jar包到maven配置的本地仓库中



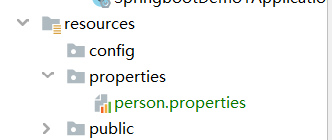
1. 运行并测试





### Springboot读取自定义配置文件

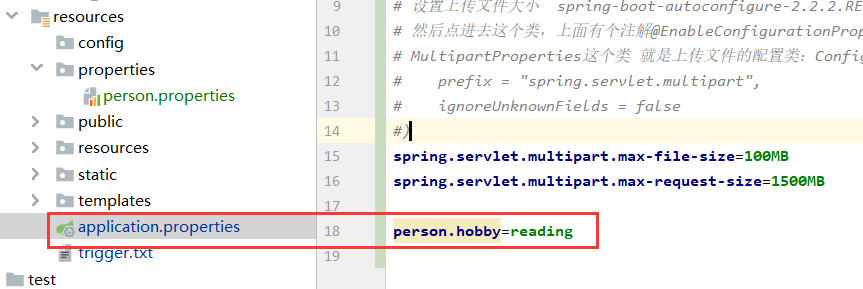
1. 在resources目录下新增properties目录 ，新增自定义配置文件person.properties



内容 如下：

|  |
| --- |
| *#自定义配置文件* **person.name**=**jack person.age**=**20 person.phone**=**13613623366** |

同时，在application.properties中添加一个配置项



1. 新增配置类，就是一个javabean,但是这个javabean只要加上三个注解，就变成了一个配置类！！！

|  |
| --- |
| **package** com.example.lchtest.springbootdemo1.domain;  **import** org.springframework.boot.context.properties.ConfigurationProperties; **import** org.springframework.context.annotation.PropertySource; **import** org.springframework.stereotype.Component;  **import** java.io.Serializable;  */\*\*  \* 配置读取类  \* 要求：1. 属性名要和配置项中的名称一致 2.要有getter setter  \*/ // 声明这是spring的一个组件* @Component *// 要读取的配置文件的位置* @PropertySource(**"classpath:properties/person.properties"**) *// 配置项的前缀* @ConfigurationProperties(prefix = **"person"**) **public class** Person {  **private** String **name**;  **private int age**;  **private** String **phone**;   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public int** getAge() {  **return age**;  }   **public void** setAge(**int** age) {  **this**.**age** = age;  }   **public** String getPhone() {  **return phone**;  }   **public void** setPhone(String phone) {  **this**.**phone** = phone;  } } |

1. Controller中使用配置项

对于自定义的配置文件，通过配置类加上注解，使配置项的值自动映射到配置类对应的属性上，在使用的地方，只需要通过@Autowired 注解自动注入这个配置类，就可以使用了，这种方式适合有大量的配置项需要读取的情况。而定义在application.properties中的配置项**person.hobby，**可以通过spring的@Value注解来读取它的值

|  |
| --- |
| @Autowired **private** Person **person**;  @Value(**"${person.hobby}"**) **private** String **hobby**; @RequestMapping(**"/testproperties"**) @ResponseBody **public** Object test(){  System.***out***.println(**"read properties person.hobby from application.properties:"** + **hobby**);  **return person**; } |

测试：

