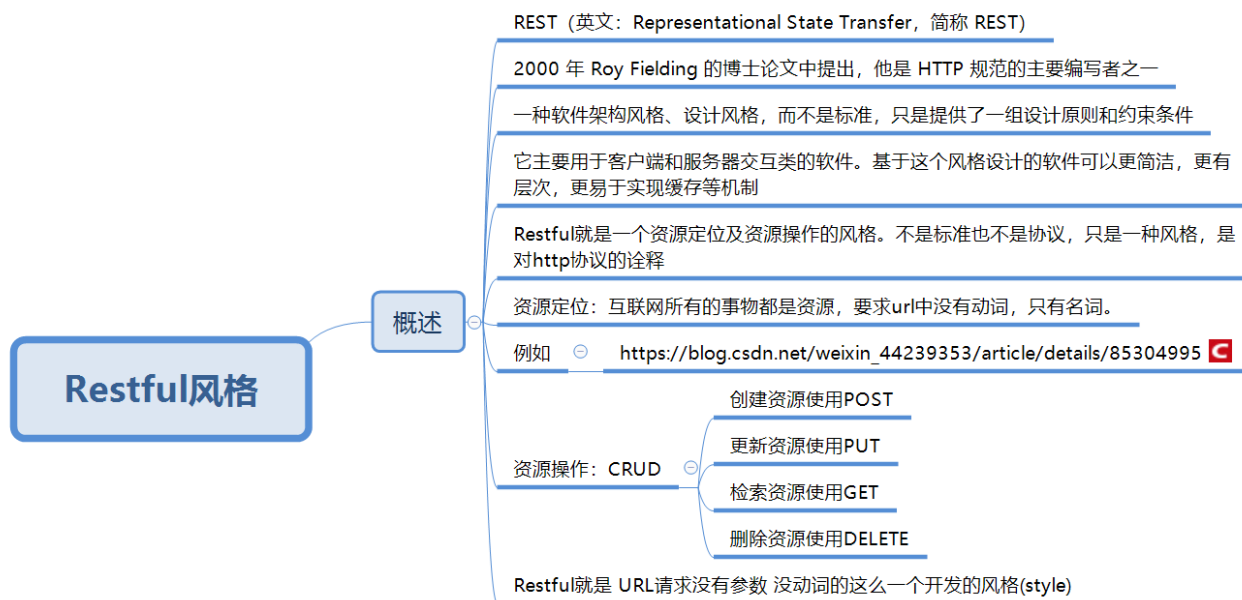


- * 学习目标
- * 能够掌握SpringMVC实现restful风格
- * 能够掌握SpringMVC响应数据和结果视图
- * 能够掌握SpringMVC文件上传
- * 能够掌握SpringMVC的拦截器
- * 能够掌握SpringMVC异常处理
- * 能够了解SpringMVC与Struts2区别

-
-
- * 回顾
 - * ModelAttribute
 - * 能够掌握SpringMVC实现restful风格



- * 案例
- @Controller

```

3 @RequestMapping("/test1")
4 public class RestFulController {
5
6     @PostMapping("/insert")
7     public String testPost(User user){
8         System.out.println(user);
9         return "success";
10    }
11    @GetMapping("/get/{id}")
12    public String testGet(@PathVariable("id") Integer id){
13        System.out.println("Get:"+id);
14        return "success";
15    }
16
17    @PutMapping("/update/{id}")
18    public String testPut(@PathVariable("id") Integer id,User user){
19        System.out.println(id+": "+user);
20        return "success";
21    }
22    @DeleteMapping("/delete/{id}")
23    public String testDelete(@PathVariable("id") Integer id){
24        System.out.println("Delete:"+id);
25        return "success";
26    }
27 }
28
29 * 单元测试
30 @Test
31 public void test13() throws Exception {
32     String result = mockMvc.perform(post("/test1/insert")
33         .param("id","2")
34         .param("username","xiaohei")
35         .param("psw","123")
36         .param("sex","男")).
37         andExpect(status().isOk()).
38         andDo(print()).andReturn().getResponse().getContentAsString();
39     System.out.println(result);
40 }
41
42 @Test

```

```

43 public void test14() throws Exception {
44     String result = mockMvc.perform(get("/test1/get/2")).
45         andExpect(status().isOk()).
46         andDo(print()).andReturn().getResponse().getContentAsString();
47     System.out.println(result);
48 }
49
50 @Test
51 public void test15() throws Exception {
52     String result = mockMvc.perform(put("/test1/update/2")
53         .param("id", "2")
54         .param("username", "xiaohei")
55         .param("psw", "123")
56         .param("sex", "男")).
57         andExpect(status().isOk()).
58         andDo(print()).andReturn().getResponse().getContentAsString();
59     System.out.println(result);
60 }
61
62 @Test
63 public void test16() throws Exception {
64     String result = mockMvc.perform(delete("/test1/delete/8")).
65         andExpect(status().isOk()).
66         andDo(print()).andReturn().getResponse().getContentAsString();
67     System.out.println(result);
68 }

```

69 * 测试结果

70 * 例如/test1/get/2，需要获取2的参数值，需要加上注解@PathVariable，否则获取参数值为

72 * 再通过postman测试

73 * 发现测试put的时候，值设置的不完全

74 * 原因：form 表单只支持 GET 与 POST 请求，而 DELETE、PUT 等 method 并不支持，Spring
75 加了一个过滤器，可以将浏览器请求改为指定的请求方式，发送给我们的控制器方法，使得支持
76 与 DELETE 请求

77 * PUT和DELETE没办法跳转视图但是返回json是可以的

79 * 添加过滤器

80 <filter>

81 <filter-name>hiddenHttpMethodFilter</filter-name>

82 <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter

```

83 </filter>
84 <filter-mapping>
85     <filter-name>hiddenHttpMethodFilter</filter-name>
86     <url-pattern>/*</url-pattern>
87 </filter-mapping>
88 * 查看过滤器的源码
89 * 发现post请求，添加_method=PUT|DELETE
90

```

http://localhost:8080/lgspringmvc/test1/update/2

POST ▼ Params Send Save

Authorization Headers (1) **Body** Pre-request Script Tests Cookies Code

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> id	2			
<input checked="" type="checkbox"/> username	xiaohei			
<input checked="" type="checkbox"/> psw	123			
<input checked="" type="checkbox"/> sex	男			
<input checked="" type="checkbox"/> _method	PUT			

http://localhost:8080/lgspringmvc/test1/delete/2

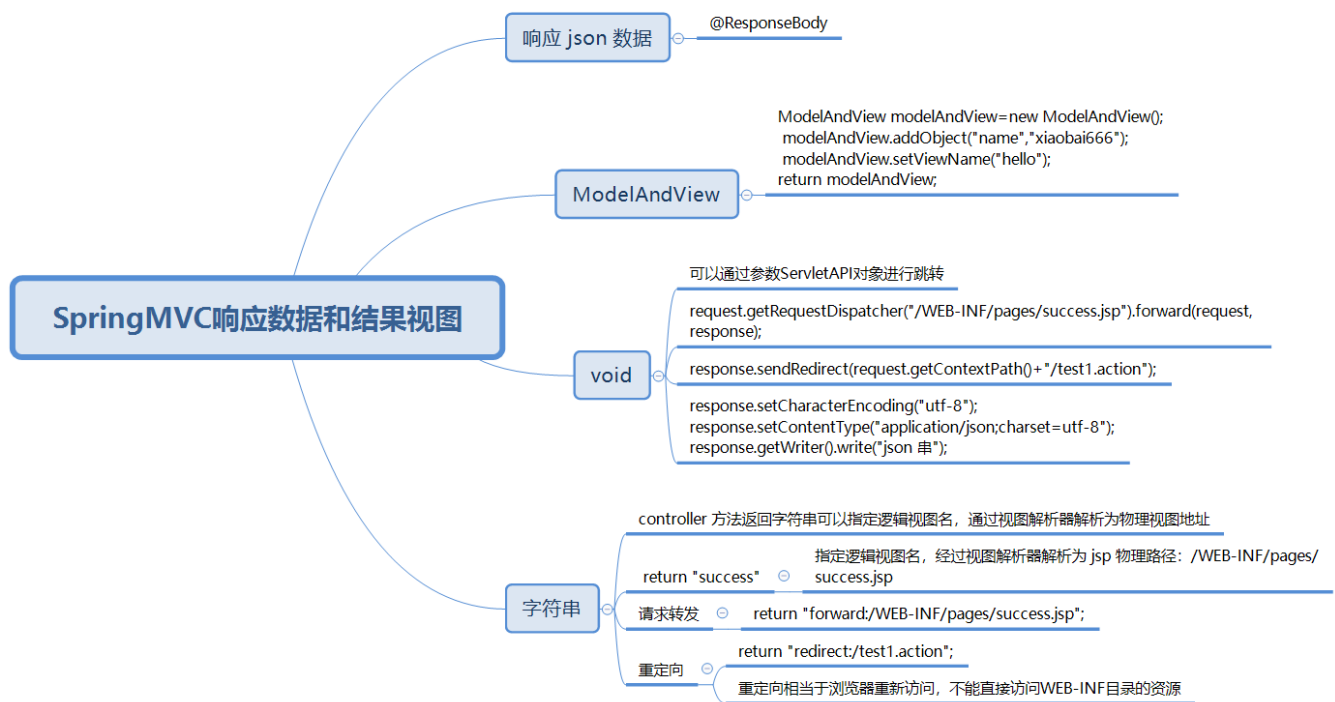
POST ▼ Params Send Save

Authorization Headers (1) **Body** Pre-request Script Tests Cookies Code

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> id	2			
<input checked="" type="checkbox"/> username	xiaohei			
<input checked="" type="checkbox"/> psw	123			
<input checked="" type="checkbox"/> sex	男			
<input checked="" type="checkbox"/> _method	DELETE			

* 能够掌握SpringMVC响应数据和结果视图



```

1 @RequestMapping("/test10")
2 public void test10(HttpServletRequest request, HttpServletResponse response)
3     throws ServletException, IOException {
4     request.getRequestDispatcher("/WEB-INF/jsp/hello.jsp").forward(request, resp
5 }
6 @RequestMapping("/test11")
7 public String test11(){
8     return "forward:/WEB-INF/jsp/hello.jsp";
9 }
10 @RequestMapping("/test12")
11 public void test12(HttpServletRequest request, HttpServletResponse response)
12     throws ServletException, IOException {
13     response.sendRedirect(request.getContextPath()+"/test1.action");
14 }
15 @RequestMapping("/test13")
16 public String test13(){
17     return "redirect:/test1.action";
18 }
19
20 * 浏览器进行测试
  
```

* 能够掌握SpringMVC文件上传

* 复习文件上传：[10-文件上传](#)

```
1 * 案例一：（传到应用服务器）
2 * 添加依赖
3 <dependency>
4     <groupId>commons-fileupload</groupId>
5     <artifactId>commons-fileupload</artifactId>
6     <version>1.3.3</version>
7 </dependency>
8 * 页面
9 <%@ page contentType="text/html; charset=UTF-8" language="java" isELIgnored="fal
10 <html>
11 <head>
12     <title>测试</title>
13 </head>
14 <body>
15 <form action="${pageContext.request.contextPath}/fileUpload"
16     method="post" enctype="multipart/form-data">
17     图片: <input type="file" name="uploadFile"/><br/>
18     <input type="submit" value="上传"/>
19 </form>
20 </body>
21 </html>
22
23 <%@ page contentType="text/html; charset=UTF-8" language="java" isELIgnored="fal
24 <html>
25 <body>
26 <h2>文件${result}</h2>
27 </body>
28 </html>
29
30 * 代码
31 @Controller
32 public class UploadController {
33     @RequestMapping("/fileUpload")
34     public String fileUpload(MultipartFile uploadFile, Model model){
35         try {
36             //生成唯一的文件名(UUID)
37             String filename = UUID.randomUUID().toString();
38             System.out.println(filename);
39             //获取原文件名: b.jpg
```

```

40     String originalFilename = uploadFile.getOriginalFilename();
41     System.out.println(originalFilename);
42     //获取带.的文件后缀名
43     String extName = originalFilename.substring(originalFilename.lastIndexOf(
44     System.out.println(extName);
45     //上传
46     //指定上传的图片的路径
47     File file= new File("D:\\work666\\upload\\"+filename+extName);
48     uploadFile.transferTo(file);
49     model.addAttribute("result","上传成功");
50     } catch (Exception e) {
51         e.printStackTrace();
52         model.addAttribute("result","上传失败");
53     }
54     return "uploadsuccess";
55 }
56 }
57
58 * 配置
59 <bean id="multipartResolver" class="
60     org.springframework.web.multipart.commons.CommonsMultipartResolver">
61     <!-- 设置上传文件的最大尺寸为5MB -->
62     <property name="maxUploadSize">
63         <value>5242880</value>
64     </property>
65 </bean>
66
67 * 案例二：（传到文件服务器）
68 * 构建文件服务项目lgimage
69     * 构建不同tomcat：端口为8082，其他的端口对应改一下
70     * 在lgimage下webapp下面
71     * 构建uploads文件夹
72     * 温馨提醒：里面新建个文件空白文件，不然有时候打包没有打包uploads目前过去
73 * tomcat默认是只读，修改tomcat的web.xml文件
74 * 添加
75     <init-param>
76         <param-name>readonly</param-name>
77         <param-value>false</param-value>
78     </init-param>
79

```

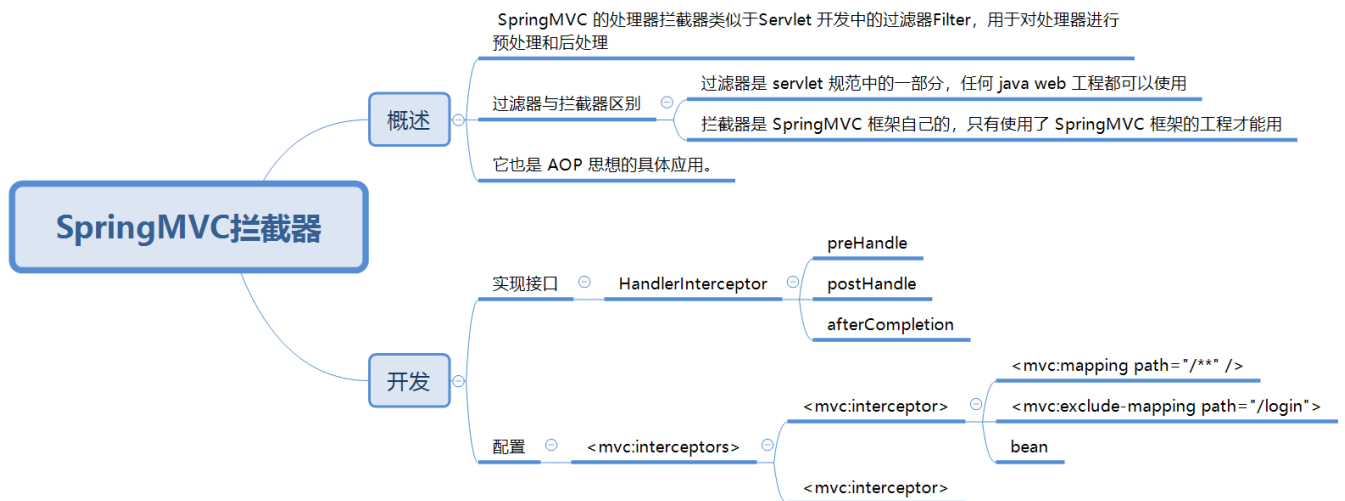
```

80 * 在应用服务中
81 * 添加依赖
82 <dependency>
83     <groupId>com.sun.jersey</groupId>
84     <artifactId>jersey-client</artifactId>
85     <version>1.19.4</version>
86 </dependency>
87 * 代码
88 public static final String FILE_SERVER_URL = "http://localhost:8082/lgimage/";
89 @RequestMapping("/fileUpload")
90 public String fileUpload2(MultipartFile uploadFile, Model model){
91     try {
92         //生成唯一的文件名(UUID)
93         String filename = UUID.randomUUID().toString();
94         System.out.println(filename);
95         //获取原文件名
96         String originalFilename = uploadFile.getOriginalFilename();
97         System.out.println(originalFilename);
98         //获取带.的文件后缀名
99         String extName = originalFilename.substring(originalFilename.lastIndexOf("."));
100         System.out.println(extName);
101         //上传
102         // 创建 sun 公司提供的 jersey 包中的 Client 对象
103         Client client = Client.create();
104         // 指定上传文件的地址，该地址是 web 路径
105         WebResource resource = client.resource(FILE_SERVER_URL+filename+extName);
106         // 实现上传
107         resource.put(uploadFile.getBytes());
108         model.addAttribute("result", "上传成功");
109     } catch (Exception e) {
110         e.printStackTrace();
111         model.addAttribute("result", "上传失败");
112     }
113     return "uploadsucces";
114 }
115 * 访问进行测试:
116 * http://localhost:8080/lgspringmvc/upload.jsp

```

* 能够掌握SpringMVC的拦截器

* 复习过滤器：[07-Filter&Listener](#)



```
1 * 案例一:(一个拦截器)
2 public class LgHandlerInterceptor implements HandlerInterceptor {
3     @Override
4     public boolean preHandle(HttpServletRequest request,
5         HttpServletResponse response, Object handler) throws Exception {
6         // 返回值:返回false, 不执行handler, 返回true执行handler
7         System.out.println("LgHandlerInterceptor--preHandle");
8         return true;
9     }
10    /**
11     * 在业务处理器处理完请求后, 但是 DispatcherServlet 向客户端返回响应前被调用
12     */
13    @Override
14    public void postHandle(HttpServletRequest request,
15        HttpServletResponse response,
16        Object handler, ModelAndView modelAndView) throws Exception {
17        System.out.println("LgHandlerInterceptor--postHandle");
18    }
19    /**
20     * 在 DispatcherServlet 完全处理完请求后被调用
21     * 可以在该方法中进行一些资源清理的操作。
22     */
23    @Override
24    public void afterCompletion(HttpServletRequest request,
```

```

25         HttpServletResponse response,
26         Object handler, Exception ex) throws Exception {
27             System.out.println("LgHandlerInterceptor--afterCompletion");
28         }
29     }
30     * 配置
31     <mvc:interceptors>
32         <mvc:interceptor>
33             <mvc:mapping path="/*" />
34             <bean id="lgHandlerInterceptor" class="com.lg.interceptor.LgHandlerIntercep
35         </mvc:interceptor>
36     </mvc:interceptors>
37     * 单元测试
38     @Test
39     public void test1() throws Exception {
40         mockMvc.perform(post("/user/testsa"));
41     }
42     * 结果
43     * preHandle 返回值为true的执行结果
44     LgHandlerInterceptor--preHandle
45     handler执行...
46     LgHandlerInterceptor--postHandle
47     LgHandlerInterceptor--afterCompletion
48     * preHandle返回值为false的执行结果（不会执行handler）
49     LgHandlerInterceptor--preHandle
50
51     * 案例二：两个拦截器
52     * 在上面基础上再编写拦截器并配置
53     public class LgHandlerInterceptor2 implements HandlerInterceptor {
54         @Override
55         public boolean preHandle(HttpServletRequest request,
56         HttpServletResponse response, Object handler) throws Exception {
57             // 返回值:返回false, 不执行handler, 返回true执行handler
58             System.out.println("LgHandlerInterceptor2--preHandle");
59             return true;
60         }
61         /**
62          * 在业务处理器处理完请求后,但是 DispatcherServlet 向客户端返回响应前被调用
63          */
64         @Override

```

```

65     public void postHandle(HttpServletRequest request,
66         HttpServletResponse response, Object handler,
67         ModelAndView modelAndView) throws Exception {
68         System.out.println("LgHandlerInterceptor2--postHandle");
69     }
70
71     /**
72      * 在 DispatcherServlet 完全处理完请求后被调用
73      * 可以在该方法中进行一些资源清理的操作。
74      */
75     @Override
76     public void afterCompletion(HttpServletRequest request,
77         HttpServletResponse response,
78         Object handler, Exception ex) throws Exception {
79         System.out.println("LgHandlerInterceptor2--afterCompletion");
80     }
81 }
82 * 配置
83 <mvc:interceptors>
84     <mvc:interceptor>
85         <mvc:mapping path="/**"/>
86         <bean id="lgHandlerInterceptor" class="com.lg.interceptor.LgHandlerInterce
87     </mvc:interceptor>
88     <mvc:interceptor>
89         <mvc:mapping path="/**"/>
90         <bean id="lgHandlerInterceptor2" class="com.lg.interceptor.LgHandlerInterce
91     </mvc:interceptor>
92 </mvc:interceptors>
93 * 单元测试
94 * 测试结果
95 * 当拦截器1和2的preHandle返回值为true
96     * 拦截器1preHandle-->拦截器2preHandle-->handler
97         -->拦截器2postHandle-->拦截器1postHandle
98         -->拦截器2afterCompletion-->拦截器1afterCompletion
99 LgHandlerInterceptor--preHandle
100 LgHandlerInterceptor2--preHandle
101 handler执行...
102 LgHandlerInterceptor2--postHandle
103 LgHandlerInterceptor--postHandle
104 LgHandlerInterceptor2--afterCompletion

```

```

105 LgHandlerInterceptor--afterCompletion
106 * 当拦截器1的preHandle返回值为false
107 LgHandlerInterceptor--preHandle
108 * 拦截器1的preHandle返回值为true和拦截器2的preHandle返回值为false
109 * 拦截器1preHandle-->拦截器2preHandle-->handler
110 -->拦截器1afterCompletion
111 LgHandlerInterceptor--preHandle
112 LgHandlerInterceptor2--preHandle
113 LgHandlerInterceptor--afterCompletion
114
115 * 案例三：
116 * 如何用户的登录过就放行访问，如果不没有登录过，跳转到的登录页面。
117 * 开发思路：
118 * 登录的页面（url:login）不需要拦截
119 * 其他的任何的url都应该由拦截器拦截。
120 * 拦截器中的逻辑：（preHandle）方法中做验证
121 * 获取URL判断下：如果 是login就放行。
122 * 如果不是登录的URL，
123 * 判断：session是否有用户信息，
124 * 如果有，已经登录，验证通过。
125 * 如果没有，重定向到登录的页面
126 * 前期准备
127 * copy之前登录小米页面
128 * 改成jsp放在/WEB-INF/jsp目录
129 * 注意修改路径
130 * 假如访问看不到css样式，图片，js之类
131 |-- location 表示路径，mapping 表示文件，**表示该目录下的文件以及子目录的文件 -->
132 <mvc:resources location="/css/" mapping="/css/**"/>
133 <mvc:resources location="/image/" mapping="/image/**"/>
134 <mvc:resources location="/js/" mapping="/js/**"/>
135 * 测试http://localhost:8080/lgspringmvc/user/login
136 * 代码
137 @RequestMapping("/login")
138 public String login(){
139     return "login";
140 }
141 @RequestMapping(value = "/loginsubmit",method = RequestMethod.POST)
142 public String loginSubmit(HttpSession session,String username,String password){
143     session.setAttribute("loginStatus","success");
144     System.out.println(username+": "+password);

```

```

145     System.out.println("登录成功...");
146     return "redirect:/index.jsp";
147 }
148 public class LoginHandlerInterceptor implements HandlerInterceptor {
149     @Override
150     public boolean preHandle(HttpServletRequest request,
151         HttpServletResponse response,
152         Object handler) throws Exception {
153         System.out.println("启动登录拦截器...");
154         Object loginStatus=request.getSession().getAttribute("loginStatus");
155         //如果不是登录的URL,
156         //判断: session是否有用户信息,
157         //如果有, 已经登录, 验证通过。
158         //如果没有重定向到登录的页面
159         if(loginStatus!=null){
160             return true;
161         }else {
162             response.sendRedirect(request.getContextPath()+"/user/login");
163             return false;
164         }
165     }
166 }
167 * 配置
168 <mvc:interceptor>
169     <mvc:mapping path="/**"/>
170     <mvc:exclude-mapping path="/user/login"/>
171     <mvc:exclude-mapping path="/user/loginsubmit"/>
172     <mvc:exclude-mapping path="/css/**"/>
173     <mvc:exclude-mapping path="/image/**"/>
174     <mvc:exclude-mapping path="/js/**"/>
175     <bean id="loginHandlerInterceptor" class="com.lg.interceptor.LoginHandler
176 </mvc:interceptor>
177
178 * 测试(测试Controller)

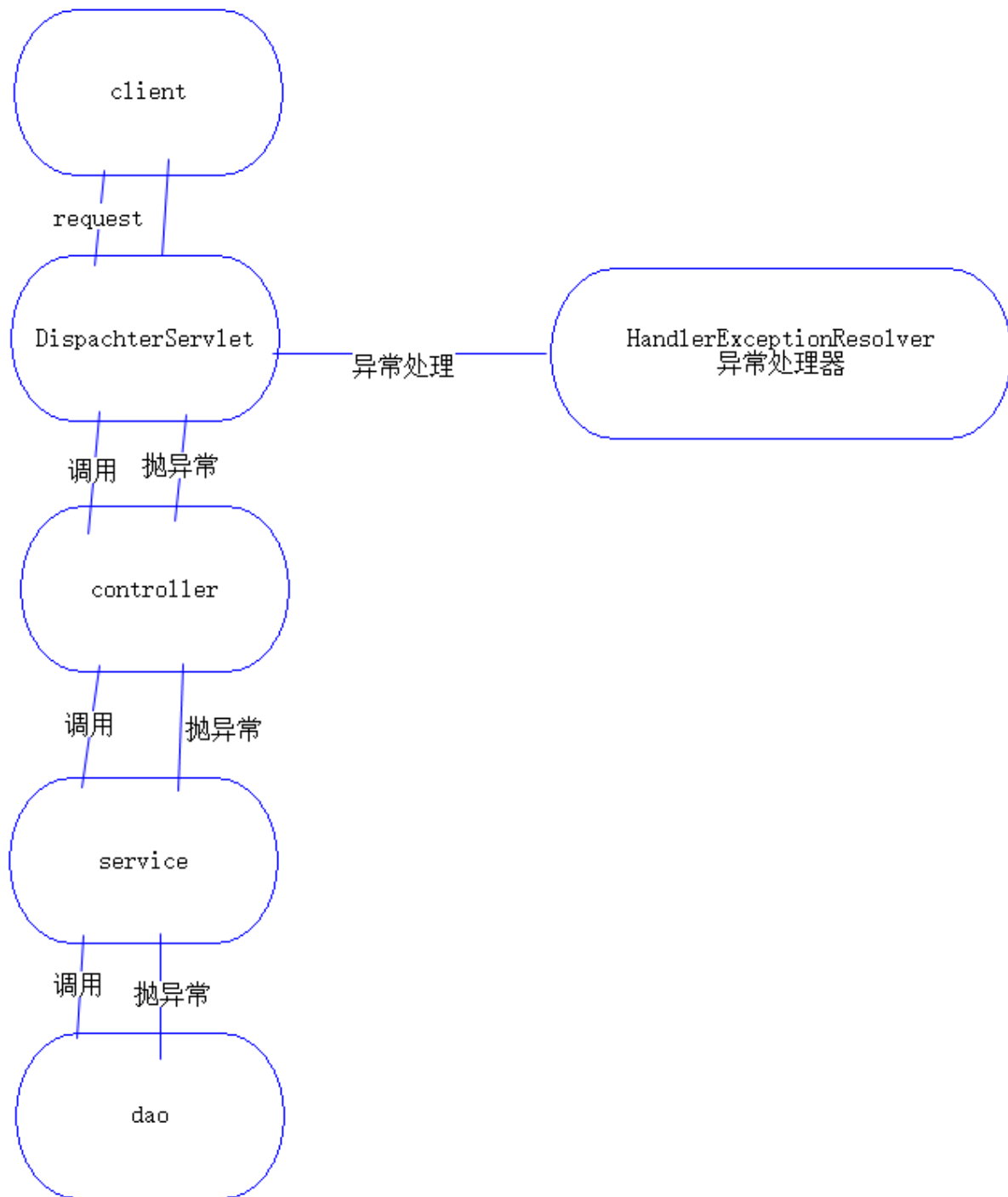
```

* 能够掌握SpringMVC异常处理

* 复习异常：[02-异常处理&Log4j](#)

* 异常处理思路

* 系统的dao、service、controller出现都通过throws Exception向上抛出，
最后由springmvc前端控制器交由异常处理器进行异常处理



```
1 * 案例一：（XML配置）
2 * 自定义异常代码
3 public class CustomException extends Exception {
4     private String message;
5     public CustomException(String message) {
6         this.message = message;
7     }
}
```

```

8     public String getMessage() {
9         return message;
10    }
11 }
12 public class CustomHandlerExceptionResolver implements HandlerExceptionResolver
13     @Override
14     public ModelAndView resolveException(HttpServletRequest request, HttpServlet
15                                         Object handler, Exception ex) {
16         ex.printStackTrace();
17         CustomException customException = null;
18         //如果抛出的是系统自定义异常则直接转换
19         if (ex instanceof CustomException) {
20             customException = (CustomException) ex;
21         } else {
22             //如果抛出的不是系统自定义异常则重新构造一个系统错误异常。
23             customException = new CustomException("系统错误，请与系统管理 员联系！
24         }
25         ModelAndView modelAndView = new ModelAndView();
26         modelAndView.addObject("message", customException.getMessage());
27         modelAndView.setViewName("error");
28         return modelAndView;
29     }
30 }
31 * 测试代码
32 @RequestMapping("testError1")
33 public void testError1(){
34     int i=100/0;
35 }
36 @RequestMapping("testError2")
37 public void testError2() throws CustomException {
38     throw new CustomException("用户不存在");
39 }
40 * 配置
41 <bean id="customHandlerExceptionResolver" class="com.lg.exception.CustomHandler
42 * 界面
43 * 在/WEB-INF/jsp/ 新建error.jsp
44 <%@ page contentType="text/html; charset=UTF-8" language="java" isELIgnored="fal
45 <html>
46 <head>
47     <title>错误提示页面</title>

```

```
48 </head>
49 <body>
50     <h1>${message}</h1>
51     
57 <head>
58     <title>测试</title>
59 </head>
60 <body>
61 <a href="${pageContext.request.contextPath}/user/testError1">错误1</a>
62 <hr/>
63 <a href="${pageContext.request.contextPath}/user/testError2">错误2</a>
64 <hr/>
65 </body>
66 </html>
67 * 测试
68 * http://localhost:8080/lgspringmvc/testError.jsp
69
70 * 案例二：（注解形式）
71 @ControllerAdvice
72 public class CustomHandlerExceptionHandler2 {
73
74     @ExceptionHandler
75     public ModelAndView resolveException(Exception ex) {
76         ex.printStackTrace();
77         CustomException customException = null;
78         //如果抛出的是系统自定义异常则直接转换
79         if (ex instanceof CustomException) {
80             customException = (CustomException) ex;
81         } else {
82             //如果抛出的不是系统自定义异常则重新构造一个系统错误异常。
83             customException = new CustomException("系统错误，请与系统管理 员联系！
84         }
85         ModelAndView modelAndView = new ModelAndView();
86         modelAndView.addObject("message", customException.getMessage());
87         modelAndView.setViewName("error");
```



```
88         return modelAndView;  
89     }  
90 }  
91
```

* 能够了解SpringMVC与Struts2区别

