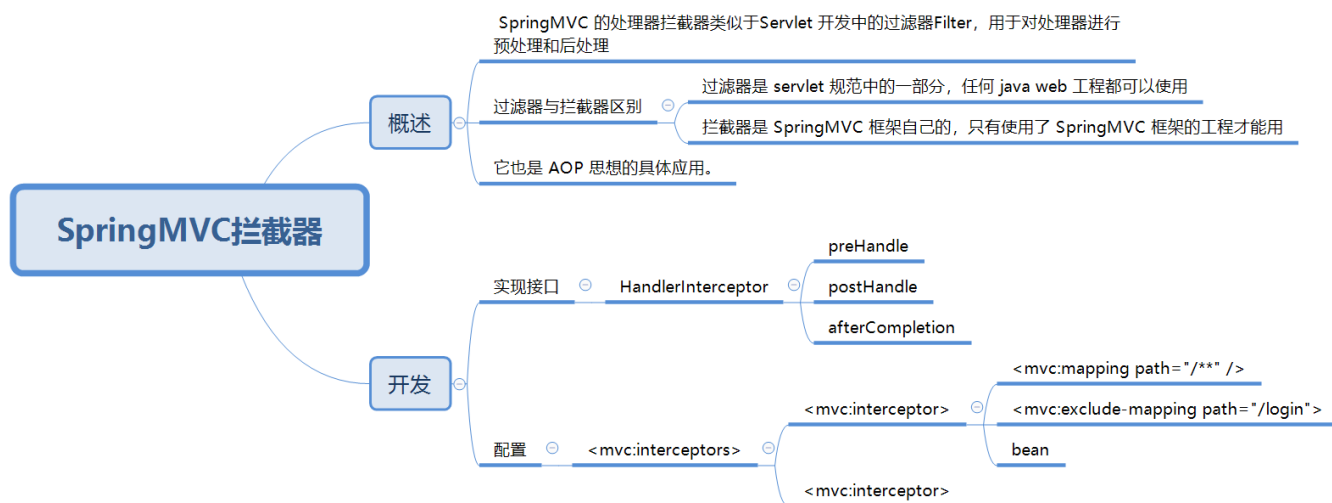


- \* 学习目标
- \* 能够掌握SpringMVC的拦截器
- \* 能够掌握SpringMVC异常处理
- \* 能够了解SpringMVC与Struts2区别
- \* 能够掌握Spring父子容器
- \* 能够掌握SSM的整合

- 
- \* 回顾
  - \* 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, Object handler, Exception ex) throws Exception {
78         System.out.println("LgHandlerInterceptor2--afterCompletion");
79     }
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     * 假如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

```

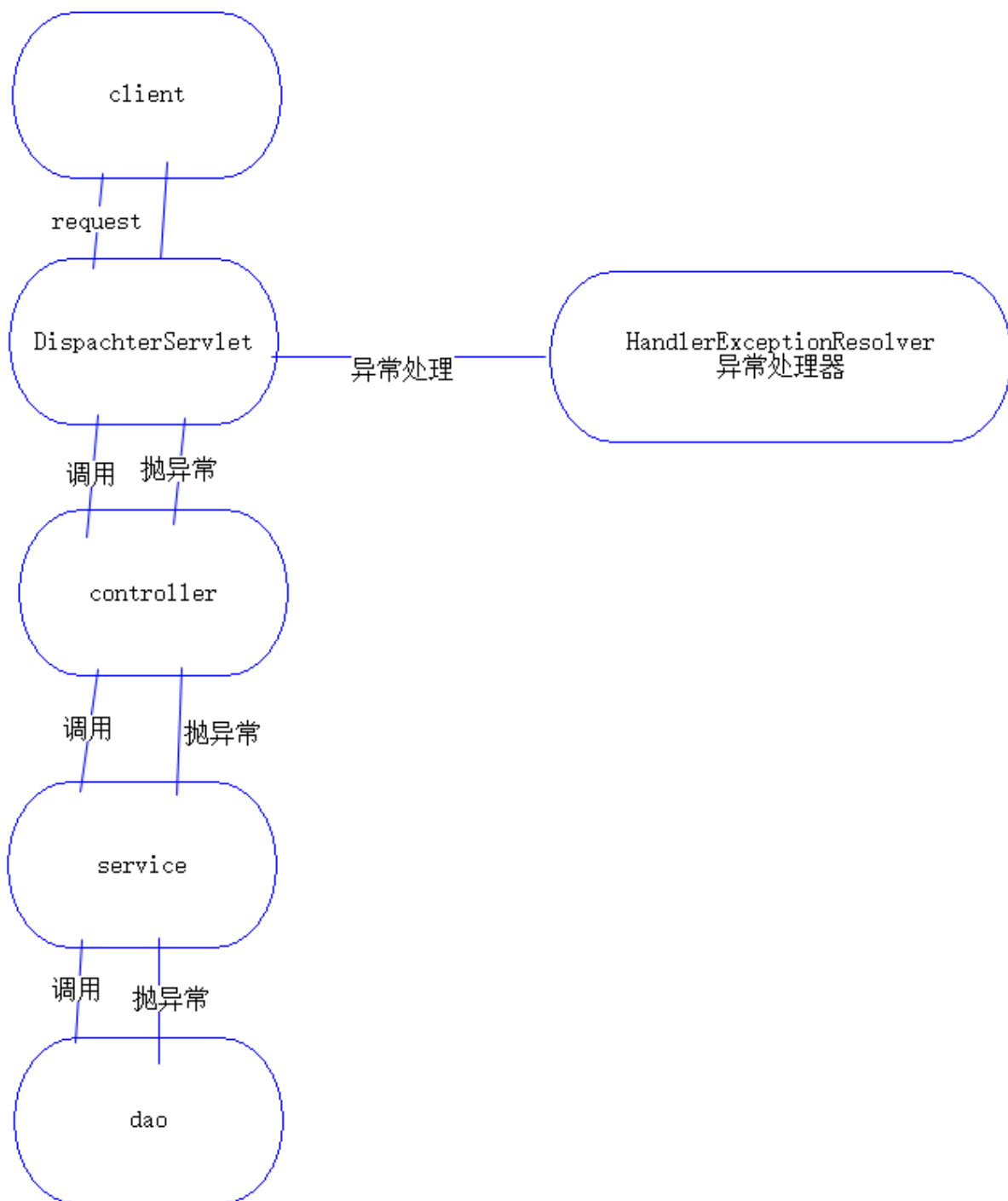
```
168 <mvc:interceptor>
169     <mvc:mapping path="/**"/>
170     <mvc:exclude-mapping path="/user/login"/>
171     <mvc:exclude-mapping path="/user/loginsubmit"/>
172     <mvc:exclude-mapping path="/css/**"/>
173     <mvc:exclude-mapping path="/image/**"/>
174     <mvc:exclude-mapping path="/js/**"/>
175     <bean id="loginHandlerInterceptor" class="com.lg.interceptor.LoginHandler
176 </mvc:interceptor>
177
178 * 测试
```

\* 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    }
```

```

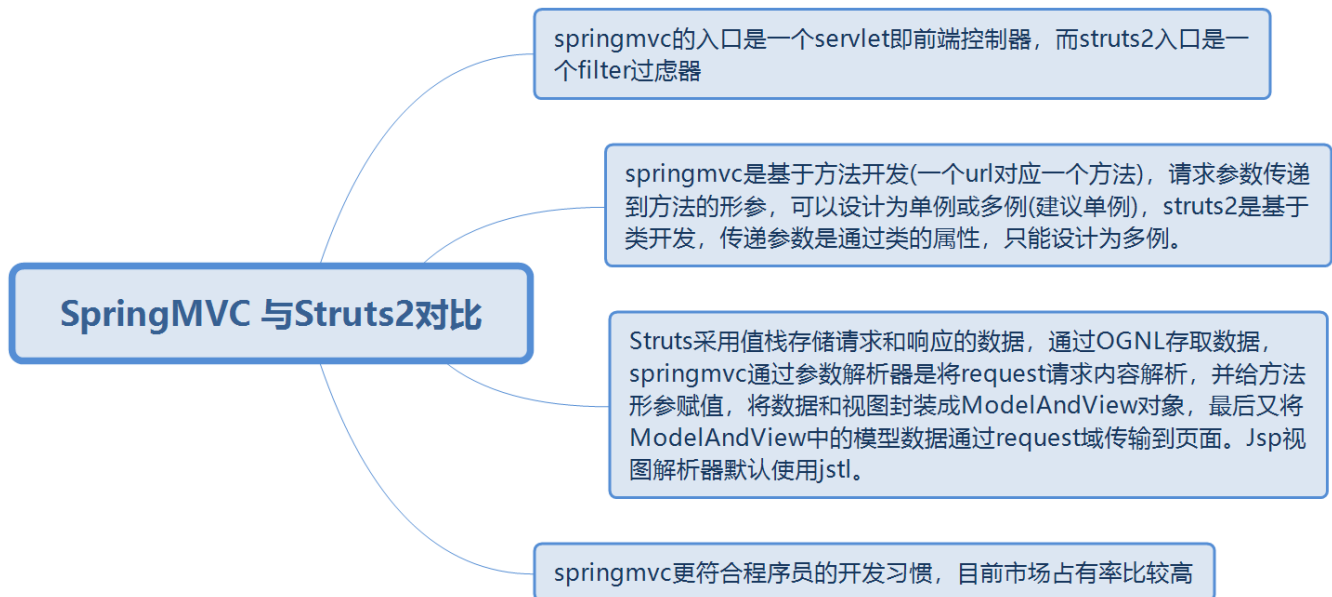
11 }
12 public class CustomHandlerExceptionResolver implements HandlerExceptionResolver
13     @Override
14     public ModelAndView resolveException(HttpServletRequest request, HttpServlet
15                                     Object handler, Exception ex) {
16         ex.printStackTrace();
17         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>

```

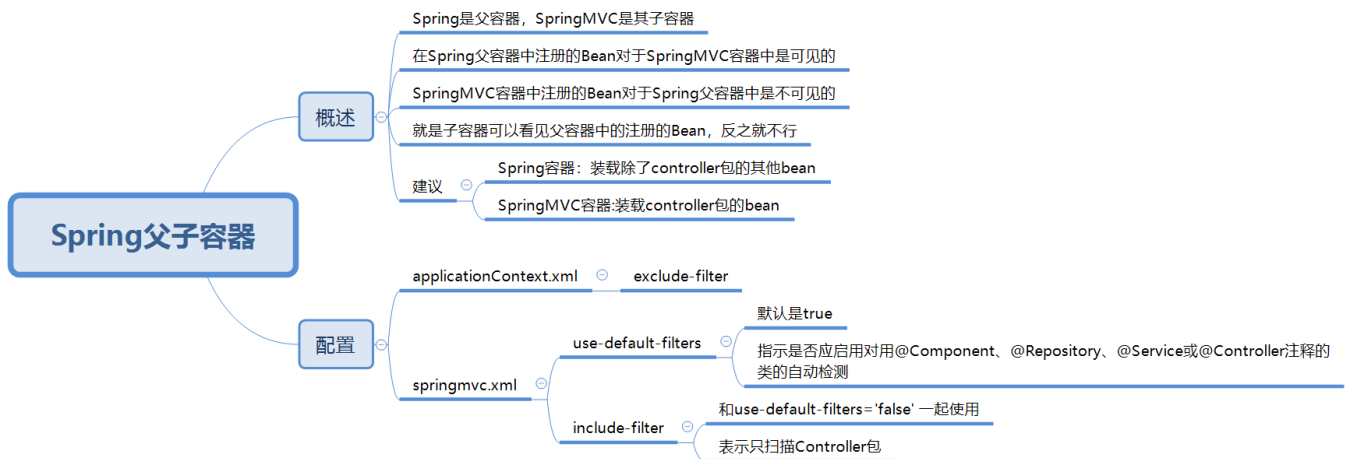


```
51     
57 <head>
58     <title>测试</title>
59 </head>
60 <body>
61 <a href="${pageContext.request.contextPath}/user/testError1">错误1</a>
62 <hr/>
63 <a href="${pageContext.request.contextPath}/user/testError2">错误2</a>
64 <hr/>
65 </body>
66 </html>
67 * 测试
68 * http://localhost:8080/lgspringmvc/testError.jsp
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 }
```

## \* 能够了解SpringMVC与Struts2区别



## \* 能够掌握Spring父子容器



## \* 能够掌握SSM的整合

### \* 复习Spring 集成MyBatis：[03-spring03](#)

- 1 \* 前期准备
- 2 \* 新建webapp的maven工程

```
3 * 复制之前Spring 集成MyBatis相关内容
4 * 复制之前SpringMVC相关内容
5 * 重点关注Spring集成SpringMVC
6 * 案例一（xml+注解形式）
7 * 添加依赖
8 <dependency>
9     <groupId>junit</groupId>
10    <artifactId>junit</artifactId>
11    <version>4.12</version>
12    <scope>test</scope>
13 </dependency>
14 <dependency>
15     <groupId>org.projectlombok</groupId>
16     <artifactId>lombok</artifactId>
17     <version>1.18.10</version>
18     <scope>provided</scope>
19 </dependency>
20 <dependency>
21     <groupId>org.springframework</groupId>
22     <artifactId>spring-context</artifactId>
23     <version>5.2.2.RELEASE</version>
24 </dependency>
25 <dependency>
26     <groupId>org.springframework</groupId>
27     <artifactId>spring-test</artifactId>
28     <version>5.2.2.RELEASE</version>
29     <scope>test</scope>
30 </dependency>
31 <dependency>
32     <groupId>org.springframework</groupId>
33     <artifactId>spring-aspects</artifactId>
34     <version>5.2.2.RELEASE</version>
35 </dependency>
36 <dependency>
37     <groupId>org.aspectj</groupId>
38     <artifactId>aspectjrt</artifactId>
39     <version>1.9.5</version>
40 </dependency>
41 <dependency>
42     <groupId>aopalliance</groupId>
```

```
43     <artifactId>aopalliance</artifactId>
44     <version>1.0</version>
45 </dependency>
46 <dependency>
47     <groupId>mysql</groupId>
48     <artifactId>mysql-connector-java</artifactId>
49     <version>5.1.16</version>
50 </dependency>
51 <dependency>
52     <groupId>org.mybatis</groupId>
53     <artifactId>mybatis</artifactId>
54     <version>3.3.0</version>
55 </dependency>
56 <dependency>
57     <groupId>org.springframework</groupId>
58     <artifactId>spring-tx</artifactId>
59     <version>5.2.2.RELEASE</version>
60 </dependency>
61 <dependency>
62     <groupId>org.slf4j</groupId>
63     <artifactId>slf4j-api</artifactId>
64     <version>1.7.25</version>
65 </dependency>
66 <dependency>
67     <groupId>log4j</groupId>
68     <artifactId>log4j</artifactId>
69     <version>1.2.17</version>
70 </dependency>
71     <!--slf4j到log4j-->
72 <dependency>
73     <groupId>org.slf4j</groupId>
74     <artifactId>slf4j-log4j12</artifactId>
75     <version>1.7.25</version>
76 </dependency>
77 <dependency>
78     <groupId>org.mybatis</groupId>
79     <artifactId>mybatis-spring</artifactId>
80     <version>1.3.2</version>
81 </dependency>
82 <dependency>
```

```
83     <groupId>com.alibaba</groupId>
84     <artifactId>druid</artifactId>
85     <version>1.1.21</version>
86 </dependency>
87 <dependency>
88     <groupId>org.springframework</groupId>
89     <artifactId>spring-webmvc</artifactId>
90     <version>5.2.2.RELEASE</version>
91 </dependency>
92 <dependency>
93     <groupId>javax.servlet</groupId>
94     <artifactId>javax.servlet-api</artifactId>
95     <version>3.1.0</version>
96     <scope>provided</scope>
97 </dependency>
98 <dependency>
99     <groupId>javax.servlet</groupId>
100    <artifactId>jstl</artifactId>
101    <version>1.2</version>
102 </dependency>
103 <dependency>
104    <groupId>commons-fileupload</groupId>
105    <artifactId>commons-fileupload</artifactId>
106    <version>1.3.3</version>
107 </dependency>
108 <dependency>
109    <groupId>com.sun.jersey</groupId>
110    <artifactId>jersey-client</artifactId>
111    <version>1.19.4</version>
112 </dependency>
113
114 * 配置
115 * applicationContext.xml
116 <?xml version="1.0" encoding="UTF-8"?>
117 <beans xmlns="http://www.springframework.org/schema/beans"
118        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
119        xmlns:context="http://www.springframework.org/schema/context"
120        xmlns:aop="http://www.springframework.org/schema/aop"
121        xsi:schemaLocation="http://www.springframework.org/schema/beans
122        http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
123     http://www.springframework.org/schema/context
124     http://www.springframework.org/schema/context/spring-context.xsd
125     http://www.springframework.org/schema/aop
126     http://www.springframework.org/schema/aop/spring-aop.xsd">
127     <import resource="spring-mybatis.xml"/>
128     <context:component-scan base-package="com.lg">
129         <context:exclude-filter type="annotation"
130             expression="org.springframework.stereotype.Controller"/>
131     </context:component-scan>
132 </beans>
133 * db.properties
134 lg.driver=com.mysql.jdbc.Driver
135 lg.url=jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-8
136 lg.username=root
137 lg.password=root
138 * spring-mybatis.xml
139 <?xml version="1.0" encoding="UTF-8"?>
140 <beans xmlns="http://www.springframework.org/schema/beans"
141     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
142     xmlns:context="http://www.springframework.org/schema/context"
143     xmlns:aop="http://www.springframework.org/schema/aop"
144     xmlns:tx="http://www.springframework.org/schema/tx"
145     xsi:schemaLocation="http://www.springframework.org/schema/beans
146         http://www.springframework.org/schema/beans/spring-beans.xsd
147         http://www.springframework.org/schema/context
148         http://www.springframework.org/schema/context/spring-context.xsd
149         http://www.springframework.org/schema/aop
150         http://www.springframework.org/schema/aop/spring-aop.xsd
151         http://www.springframework.org/schema/tx
152         http://www.springframework.org/schema/tx/spring-tx.xsd">
153     <context:property-placeholder location="classpath:db.properties"/>
154     <bean id="druid" class="com.alibaba.druid.pool.DruidDataSource">
155         <property name="driverClassName" value="${lg.driver}"/>
156         <property name="url" value="${lg.url}"/>
157         <property name="username" value="${lg.username}"/>
158         <property name="password" value="${lg.password}"/>
159     </bean>
160     <!--构建SqlSessionFactory:由于这里是与Spring结合的选择SqlSessionFactoryBean
161         (里面包含SqlSessionFactory) -->
162     <bean id="ssf" class="org.mybatis.spring.SqlSessionFactoryBean">
```

```

163     <property name="dataSource" ref="druid"/>
164     <property name="mapperLocations" value="classpath:com/lg/dao/*.xml"/>
165     <property name="typeAliasesPackage" value="com.lg.bean"/>
166 </bean>
167
168 <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
169     <property name="basePackage" value="com.lg.dao"/>
170     <property name="sqlSessionFactoryBeanName" value="ssf"/>
171 </bean>
172 <bean id="transactionManager" class="org.springframework.jdbc.
173     datasource.DataSourceTransactionManager">
174     <property name="dataSource" ref="druid"/>
175 </bean>
176 <tx:annotation-driven transaction-manager="transactionManager"/>
177 </beans>
178 * springmvc.xml
179 <?xml version="1.0" encoding="UTF-8"?>
180 <beans xmlns="http://www.springframework.org/schema/beans"
181     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
182     xmlns:context="http://www.springframework.org/schema/context"
183     xmlns:aop="http://www.springframework.org/schema/aop"
184     xmlns:mvc="http://www.springframework.org/schema/mvc"
185     xsi:schemaLocation="http://www.springframework.org/schema/beans
186     http://www.springframework.org/schema/beans/spring-beans.xsd
187     http://www.springframework.org/schema/context
188     http://www.springframework.org/schema/context/spring-context.xsd
189     http://www.springframework.org/schema/mvc
190     http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"
191 >
192     <context:component-scan base-package="com.lg" use-default-filters="false">
193         <context:include-filter type="annotation"
194             expression="org.springframework.stereotype.Controller"/>
195     </context:component-scan>
196     <bean class="org.springframework.web.servlet.view.InternalResourceViewResol
197         <property name="prefix" value="/WEB-INF/jsp/" />
198         <property name="suffix" value=".jsp" />
199     </bean>
200     <bean id="conversionService1" class="org.springframework.context.support.
201         ConversionServiceFactoryBean">
202         <property name="converters">

```

```

203         <array>
204             <bean class="com.lg.converter.StringToDateConverter"></bean>
205         </array>
206     </property>
207 </bean>
208 <mvc:annotation-driven conversion-service="conversionService1"/>
209 <bean id="multipartResolver"
210     class="org.springframework.web.multipart.commons.CommonsMultipartResol
211 <!-- 设置上传文件的最大尺寸为5MB -->
212     <property name="maxUploadSize">
213         <value>5242880</value>
214     </property>
215 </bean>
216 <!-- 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
217 <mvc:resources location="/css/" mapping="/css/**"/>
218 <mvc:resources location="/image/" mapping="/image/**"/>
219 <mvc:resources location="/js/" mapping="/js/**"/>
220 </beans>
221 * log4j.properties
222 log4j.rootLogger=DEBUG, A1
223 # A1 is set to be a ConsoleAppender which outputs to System.out.
224 log4j.appender.A1=org.apache.log4j.ConsoleAppender
225 # A1 uses PatternLayout.
226 log4j.appender.A1.layout=org.apache.log4j.PatternLayout
227 # The conversion pattern uses format specifiers. You might want to
228 # change the pattern and watch the output format change.
229 #log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n
230 log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %20c - %m%n
231 * web.xml
232 <!DOCTYPE web-app PUBLIC
233     "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
234     "http://java.sun.com/dtd/web-app_2_3.dtd" >
235
236 <web-app>
237     <display-name>亮哥教育</display-name>
238     <context-param>
239         <param-name>contextConfigLocation</param-name>
240         <param-value>classpath:applicationContext.xml</param-value>
241     </context-param>
242     <filter>

```



```

243     <filter-name>CharacterEncodingFilter</filter-name>
244     <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-
245     <init-param>
246         <param-name>encoding</param-name>
247         <param-value>UTF-8</param-value>
248     </init-param>
249 </filter>
250 <filter>
251     <filter-name>hiddenHttpMethodFilter</filter-name>
252     <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-
253 </filter>
254 <filter-mapping>
255     <filter-name>CharacterEncodingFilter</filter-name>
256     <url-pattern>/*</url-pattern>
257 </filter-mapping>
258 <filter-mapping>
259     <filter-name>hiddenHttpMethodFilter</filter-name>
260     <url-pattern>/*</url-pattern>
261 </filter-mapping>
262 <listener>
263     <listener-class>org.springframework.web.context.ContextLoaderListener</list
264 </listener>
265 <servlet>
266     <servlet-name>springmvc</servlet-name>
267     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-c
268     <init-param>
269         <param-name>contextConfigLocation</param-name>
270         <param-value>classpath:springmvc.xml</param-value>
271     </init-param>
272 </servlet>
273 <servlet-mapping>
274     <servlet-name>springmvc</servlet-name>
275     <url-pattern>/</url-pattern>
276 </servlet-mapping>
277 </web-app>
278
279 */ 转班案例
280 @Controller
281 @RequestMapping("/emp")
282 public class TransferEmployeeController {

```

```

283     @Autowired
284     private TransferEmployeeService transferEmployeeService;
285     @RequestMapping("/transfer")
286     public String transfer(String empno,String job,Integer odid,Integer ndid){
287         Map<String,Object> params=new HashMap<>();
288         params.put("empno",empno);
289         params.put("job",job);
290         params.put("odid",odid);
291         params.put("ndid",ndid);
292         transferEmployeeService.transferEmployee(params);
293         return "transferSuccess";
294     }
295 }
296
297 * 单元测试
298 @Test
299 public void test17() throws Exception {
300     String result = mockMvc.perform(delete("/emp/transfer")
301         .param("empno","LG008")
302         .param("job","前端开发工程师")
303         .param("odid","1")
304         .param("ndid","2")).
305     andExpect(status().isOk()).
306         andDo(print()).andReturn().getResponse().getContentAsString();
307     System.out.println(result);
308 }
309
310 * 父子容器测试
311 @RequestMapping("/test14")
312 public void test14(HttpServletRequest request, HttpServletResponse response) {
313     // 父容器
314     WebApplicationContext parentContext = WebApplicationContextUtils.
315         getRequiredWebApplicationContext(request.getServletContext());
316     int beanCount = parentContext.getBeanDefinitionCount();
317     System.out.println("parent-beanCount:" + beanCount);
318     String[] beanDefinitionNames = parentContext.getBeanDefinitionNames();
319     for (String beanName : beanDefinitionNames) {
320         System.out.println("parent:" + beanName);
321     }
322 }

```

```
323      //子容器
324      WebApplicationContext childContext = (WebApplicationContext) request.
325          getAttribute(DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE);
326      beanCount = childContext.getBeanDefinitionCount();
327      System.out.println("child-beanCount:" + beanCount);
328      beanDefinitionNames = childContext.getBeanDefinitionNames();
329      for (String beanName : beanDefinitionNames) {
330          System.out.println("child:" + beanName);
331      }
332  }
333  * 用浏览器测试
334  * http://localhost:8080/lgssm/test/test14
335
336  * 作业（零XML配置）
337  * 可以参考百度
338  * SSM整合--智慧汽车项目（两天时间）：零XML配置
```