* 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

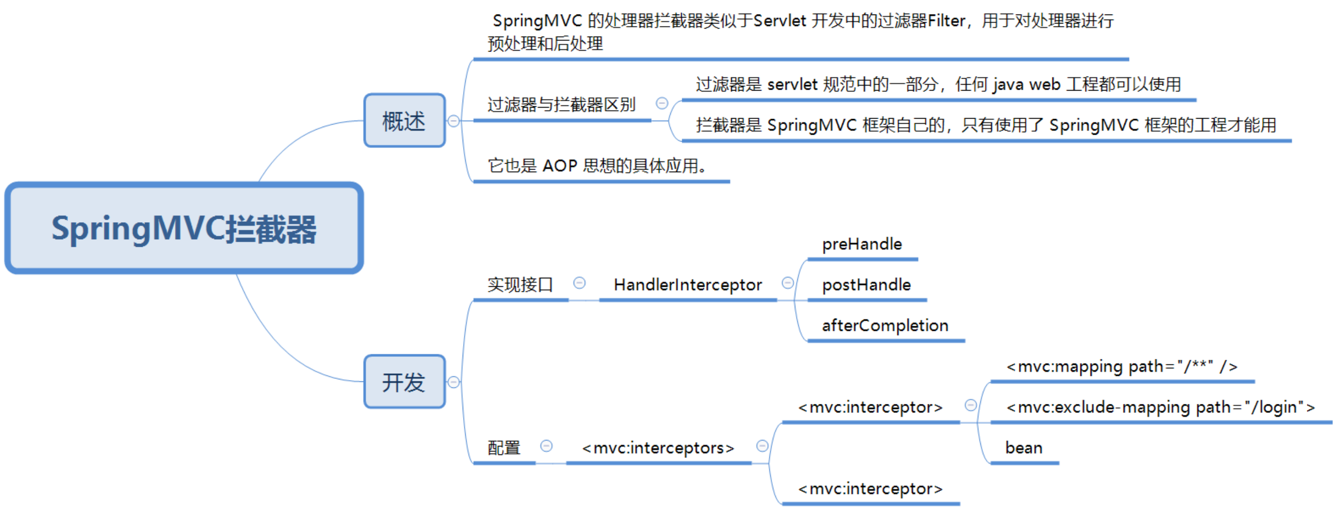
 * 能够掌握SSM的整合

----

 * 回顾

* 能够掌握SpringMVC的拦截器

  * 复习过滤器：[07-Fitler&Listener]()


SpringMVC拦截器
概述
SpringMVC 的处理器拦截器类似于Servlet 开发中的过滤器Filter，用于对处理器进行预处理和后处理
过滤器与拦截器区别
过滤器是 servlet 规范中的一部分，任何 java web 工程都可以使用
拦截器是 SpringMVC 框架自己的，只有使用了 SpringMVC 框架的工程才能用
它也是 AOP 思想的具体应用。
开发
实现接口
HandlerInterceptor
preHandle
postHandle
afterCompletion
配置
<mvc:interceptors>
<mvc:interceptor>
<mvc:mapping path="/**" />
<mvc:exclude-mapping path="/login">
bean
<mvc:interceptor>

```
* 案例一:(一个拦截器)
public class LgHandlerInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        // 返回值:返回false，不执行handler，返回true执行handler
        System.out.println("LgHandlerInterceptor--preHandle");
        return true;
    }
    /**
     * 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
     */
    @Override
    public void postHandle(HttpServletRequest request,
       HttpServletResponse response,
       Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("LgHandlerInterceptor--postHandle");
    }
    /**
     * 在 DispatcherServlet 完全处理完请求后被调用
     * 可以在该方法中进行一些资源清理的操作。
     */
    @Override
    public void afterCompletion(HttpServletRequest request,
          HttpServletResponse response,
         Object handler, Exception ex) throws Exception {
        System.out.println("LgHandlerInterceptor--afterCompletion");
    }
}
* 配置
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/**"/>
    <bean id="lgHandlerInterceptor" class="com.lg.interceptor.LgHandlerIntercep
  </mvc:interceptor>
</mvc:interceptors>
* 单元测试
 @Test
public void test1() throws Exception {
  mockMvc.perform(post("/user/testsa"));
}
* 结果
 * preHandle 返回值为true的执行结果
LgHandlerInterceptor--preHandle
handler执行...
LgHandlerInterceptor--postHandle
LgHandlerInterceptor--afterCompletion
```

```
 * preHandle返回值为false的执行结果（不会执行handler）
LgHandlerInterceptor--preHandle

* 案例二：两个拦截器
 * 在上面基础上再编写拦截器并配置
public class LgHandlerInterceptor2 implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler) throws Exception {
        // 返回值:返回false，不执行handler，返回true执行handler
        System.out.println("LgHandlerInterceptor2--preHandle");
        return true;
    }
    /**
     * 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
     */
    @Override
    public void postHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler,
    ModelAndView modelAndView) throws Exception {
        System.out.println("LgHandlerInterceptor2--postHandle");
    }

    /**
     * 在 DispatcherServlet 完全处理完请求后被调用
     * 可以在该方法中进行一些资源清理的操作。
     */
    @Override
    public void afterCompletion(HttpServletRequest
    request, HttpServletResponse response,
    Object handler, Exception ex) throws Exception {
        System.out.println("LgHandlerInterceptor2--afterCompletion");
    }
}
* 配置
 <mvc:interceptors>
   <mvc:interceptor>
     <mvc:mapping path="/**"/>
     <bean id="lgHandlerInterceptor" class="com.lg.interceptor.LgHandlerInterce
   </mvc:interceptor>
```

```
    <mvc:interceptor>
      <mvc:mapping path="/**"/>
      <bean id="lgHandlerInterceptor2" class="com.lg.interceptor.LgHandlerInterc
    </mvc:interceptor>
</mvc:interceptors>
* 单元测试
* 测试结果
 * 当拦截器1和2的preHandle返回值为true
   * 拦截器1preHandle-->拦截器2preHandle-->handler
     -->拦截器2postHandle-->拦截器1postHandle
     -->拦截器2afterCompletion-->拦截器1afterCompletion
LgHandlerInterceptor--preHandle
LgHandlerInterceptor2--preHandle
handler执行...
LgHandlerInterceptor2--postHandle
LgHandlerInterceptor--postHandle
LgHandlerInterceptor2--afterCompletion
LgHandlerInterceptor--afterCompletion
 * 当拦截器1的preHandle返回值为false
 LgHandlerInterceptor--preHandle
 * 拦截器1的preHandle返回值为true和拦截器2的preHandle返回值为false
  * 拦截器1preHandle-->拦截器2preHandle-->handler
   -->拦截器1afterCompletion
 LgHandlerInterceptor--preHandle
 LgHandlerInterceptor2--preHandle
 LgHandlerInterceptor--afterCompletion

* 案例三：
  * 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。
  * 开发思路：
   * 登录的页面（url:login）不需要拦截
   * 其他的任何的url都应该由拦截器拦截。
     * 拦截器中的逻辑：（preHandle） 方法中做验证
      * 获取URL判断下：如果 是login就放行。
      * 如果不是登录的URL，
       * 判断：session是否有用户信息，
         * 如果有，已经登录，验证通过。
         * 如果没有，重定向到登录的页面
 * 前期准备
  * copy之前登录小米页面
```

```
   * 改成jsp放在/WEB-INF/jsps目录
   * 注意修改路径
  * 假如访问看不到css样式，图片，js之类
  !-- 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 -->
    <mvc:resources location="/css/" mapping="/css/**"/>
    <mvc:resources location="/image/" mapping="/image/**"/>
    <mvc:resources location="/js/" mapping="/js/**"/>
  * 测试http://localhost:8080/lgspringmvc/user/login
* 代码
@RequestMapping("/login")
public String login(){
   return "login";
 }
@RequestMapping(value = "/loginsubmit",method = RequestMethod.POST)
public String loginSubmit(HttpSession session,String username,String password){
        session.setAttribute("loginStatus","success");
        System.out.println(username+":"+password);
        System.out.println("登录成功...");
        return "redirect:/index.jsp";
}
public class LoginHandlerInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response,
            Object handler) throws Exception {
        System.out.println("启动登录拦截器...");
        Object loginStatus=request.getSession().getAttribute("loginStatus");
        //如果不是登录的URL，
        //判断：session是否有用户信息，
        //如果有，已经登录，验证通过。
        //如果没有重定向到登录的页面
        if(loginStatus!=null){
            return true;
        }else {
            response.sendRedirect(request.getContextPath()+"/user/login");
            return false;
        }
    }
}
* 配置
```

```
  <mvc:interceptor>
      <mvc:mapping path="/**"/>
      <mvc:exclude-mapping path="/user/login"/>
      <mvc:exclude-mapping path="/user/loginsubmit"/>
      <mvc:exclude-mapping path="/css/**"/>
      <mvc:exclude-mapping path="/image/**"/>
      <mvc:exclude-mapping path="/js/**"/>
      <bean id="loginHandlerInterceptor" class="com.lg.interceptor.LoginHandler
  </mvc:interceptor>

* 测试
```

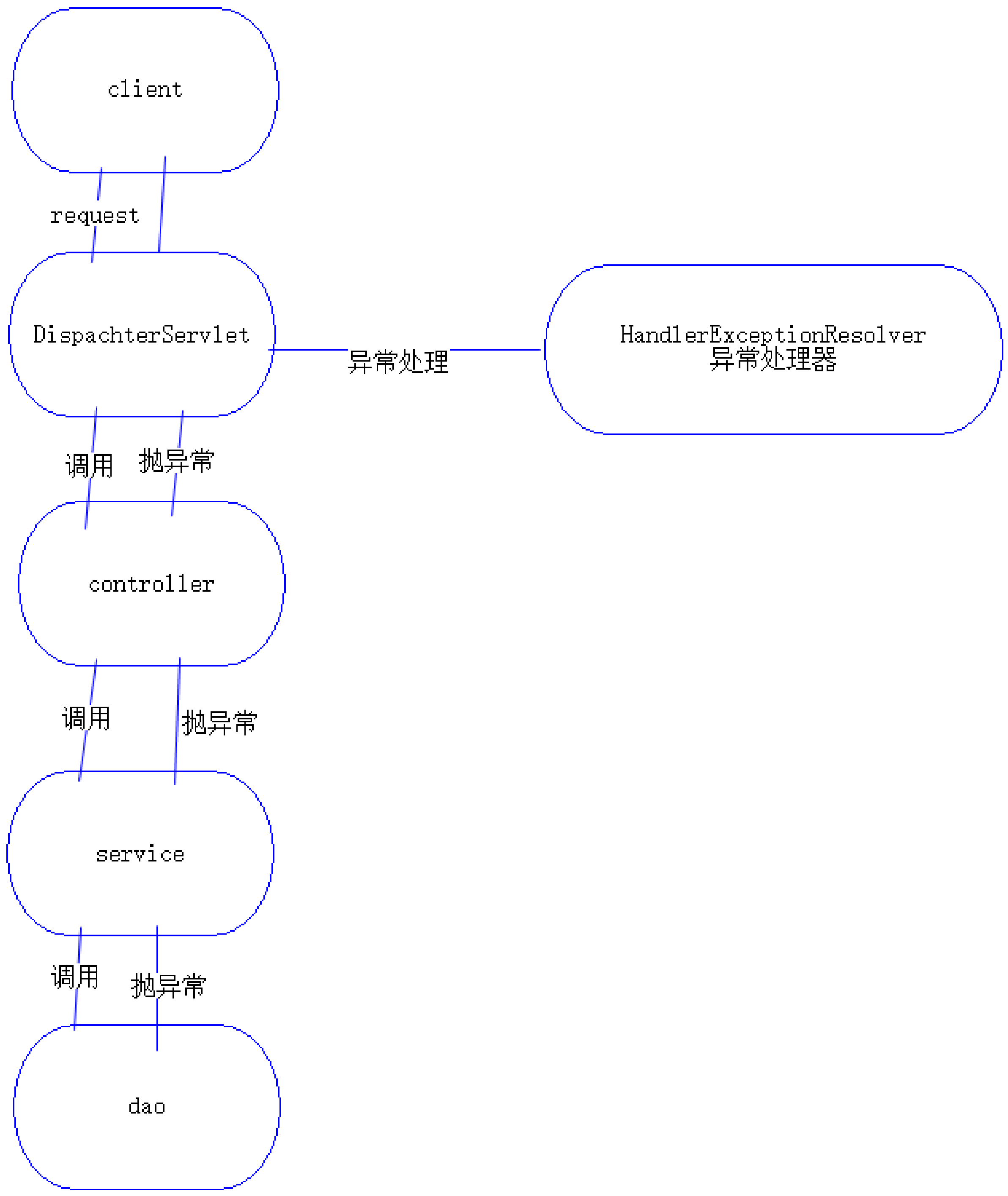
* 能够掌握SpringMVC异常处理

  * 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，

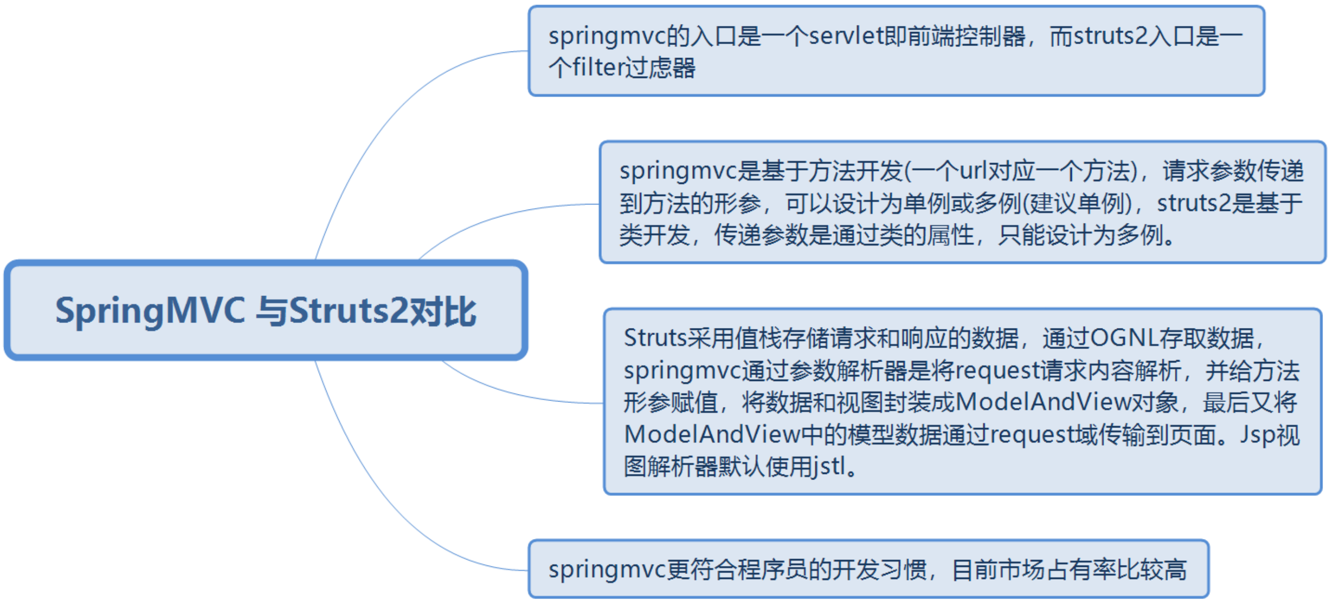
     最后由springmvc前端控制器交由异常处理器进行异常处理



```
* 案例一：（XML配置）
* 自定义异常代码
public class CustomException extends Exception {
    private String message;
    public CustomException(String message) {
        this.message = message;
    }
    public String getMessage() {
        return message;
    }
}
public class CustomHandlerExceptionResolver implements HandlerExceptionResolver
    @Override
    public ModelAndView resolveException(HttpServletRequest request, HttpServle
                                         Object handler, Exception ex) {
        ex.printStackTrace();
        CustomException customException = null;
        //如果抛出的是系统自定义异常则直接转换
        if (ex instanceof CustomException) {
            customException = (CustomException) ex;
        } else {
            //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。
            customException = new CustomException("系统错误，请与系统管理 员联系!
        }
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("message", customException.getMessage());
        modelAndView.setViewName("error");
        return modelAndView;
    }
}
* 测试代码
@RequestMapping("testError1")
public void testError1(){
    int i=100/0;
}
@RequestMapping("testError2")
public void testError2() throws CustomException {
    throw new CustomException("用户不存在");
}
* 配置
<bean id="customHandlerExceptionResolver" class="com.lg.exception.CustomHandler
* 界面
 * 在/WEB-INF/jsps/ 新建error.jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="fal
<html>
<head>
    <title>错误提示页面</title>
</head>
<body>
    <h1>${message}</h1>
    <img width="85%" height="85%" src="${pageContext.request.contextPath}/image
</body>
</html>
 * 在wepapp 下新建testError.jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="fal
<html>
<head>
    <title>测试</title>
</head>
<body>
<a href="${pageContext.request.contextPath}/user/testError1">错误1</a>
<hr/>
<a href="${pageContext.request.contextPath}/user/testError2">错误2</a>
<hr/>
</body>
</html>
* 测试
 * http://localhost:8080/lgspringmvc/testError.jsp

* 案例二：(注解形式)
@ControllerAdvice
public class CustomHandlerExceptionResolver2 {

    @ExceptionHandler
    public ModelAndView resolveException(Exception ex) {
        ex.printStackTrace();
        CustomException customException = null;
        //如果抛出的是系统自定义异常则直接转换
        if (ex instanceof CustomException) {
            customException = (CustomException) ex;
        } else {
            //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。
            customException = new CustomException("系统错误，请与系统管理 员联系!
        }
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("message", customException.getMessage());
        modelAndView.setViewName("error");
        return modelAndView;
    }
}
```

```

```

* 能够了解SpringMVC与Struts2区别


SpringMVC 与Struts2对比

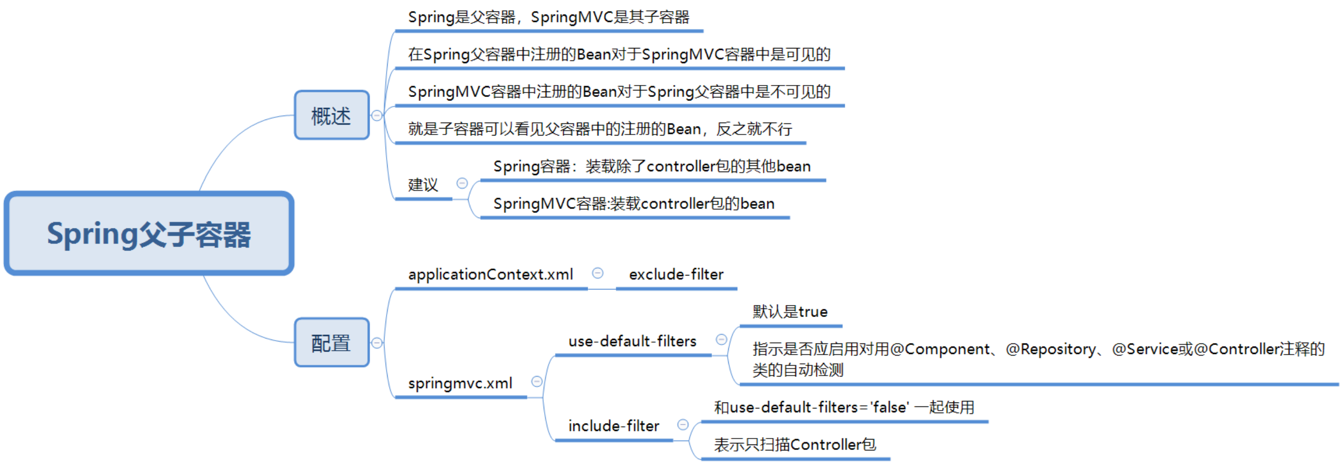
springmvc的入口是一个servlet即前端控制器，而struts2入口是一个filter过虑器

springmvc是基于方法开发(一个url对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。

Struts采用值栈存储请求和响应的数据，通过OGNL存取数据，springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过request域传输到页面。Jsp视图解析器默认使用jstl。

springmvc更符合程序员的开发习惯，目前市场占有率比较高

* 能够掌握Spring父子容器


Spring父子容器

概述
- Spring是父容器，SpringMVC是其子容器
- 在Spring父容器中注册的Bean对于SpringMVC容器中是可见的
- SpringMVC容器中注册的Bean对于Spring父容器中是不可见的
- 就是子容器可以看见父容器中的注册的Bean，反之就不行
- 建议
  - Spring容器：装载除了controller包的其他bean
  - SpringMVC容器:装载controller包的bean

配置
- applicationContext.xml
  - exclude-filter
- springmvc.xml
  - use-default-filters
    - 默认是true
    - 指示是否应启用对用@Component、@Repository、@Service或@Controller注释的类的自动检测
  - include-filter
    - 和use-default-filters='false' 一起使用
    - 表示只扫描Controller包

* 能够掌握SSM的整合

* 复习Spring 集成MyBatis：03-spring03

```
* 前期准备
* 新建webapp的maven工程
* 复制之前Spring 集成MyBatis相关内容
* 复制之前SpringMVC相关内容
* 重点关注Spring集成SpringMVC
* 案例一（xml+注解形式）
* 添加依赖
<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.12</version>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <version>1.18.10</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context</artifactId>
   <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-test</artifactId>
   <version>5.2.2.RELEASE</version>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-aspects</artifactId>
   <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjrt</artifactId>
   <version>1.9.5</version>
</dependency>
<dependency>
   <groupId>aopalliance</groupId>
```

```xml
    <artifactId>aopalliance</artifactId>
    <version>1.0</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.16</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.3.0</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
          <!--slf4j到log4j-->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.25</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.2</version>
</dependency>
<dependency>
```

```
   <groupId>com.alibaba</groupId>
   <artifactId>druid</artifactId>
   <version>1.1.21</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
</dependency>
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.3</version>
</dependency>
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-client</artifactId>
  <version>1.19.4</version>
</dependency>

* 配置
 * applicationContext.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">
    <import resource="spring-mybatis.xml"/>
    <context:component-scan base-package="com.lg">
        <context:exclude-filter type="annotation"
        expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>
</beans>
 * db.properties
lg.driver=com.mysql.jdbc.Driver
lg.url=jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-8
lg.username=root
lg.password=root
 * spring-mybatis.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd">
      <context:property-placeholder location="classpath:db.properties"/>
      <bean id="druid" class="com.alibaba.druid.pool.DruidDataSource">
          <property name="driverClassName" value="${lg.driver}"/>
          <property name="url" value="${lg.url}"/>
          <property name="username" value="${lg.username}"/>
          <property name="password" value="${lg.password}"/>
      </bean>
    <!--构建SqlSessionFactory:由于这里是与Spring结合的选择SqlSessionFactoryBean
    （里面包含SqlSessionFactory）-->
     <bean id="ssf" class="org.mybatis.spring.SqlSessionFactoryBean">
```

```
         <property name="dataSource" ref="druid"/>
         <property name="mapperLocations" value="classpath:com/lg/dao/*.xml"/>
         <property name="typeAliasesPackage" value="com.lg.bean"/>
     </bean>

    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.lg.dao"/>
        <property name="sqlSessionFactoryBeanName" value="ssf"/>
    </bean>
    <bean id="transactionManager" class="org.springframework.jdbc.
        datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="druid"/>
    </bean>
    <tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
 * springmvc.xml
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"
>
    <context:component-scan base-package="com.lg" use-default-filters="false">
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResol
        <property name="prefix" value="/WEB-INF/jsps/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
    <bean id="conversionService1" class="org.springframework.context.support.
        ConversionServiceFactoryBean">
        <property name="converters">
```

```
            <array>
                <bean class="com.lg.converter.StringToDateConverter"></bean>
            </array>
        </property>
    </bean>
    <mvc:annotation-driven conversion-service="conversionService1"/>
    <bean id="multipartResolver"
        class="org.springframework.web.multipart.commons.CommonsMultipartResolv
        <!-- 设置上传文件的最大尺寸为5MB -->
        <property name="maxUploadSize">
            <value>5242880</value>
        </property>
    </bean>
    <!-- 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 -
    <mvc:resources location="/css/" mapping="/css/**"/>
    <mvc:resources location="/image/" mapping="/image/**"/>
    <mvc:resources location="/js/" mapping="/js/**"/>
</beans>
* log4j.properties
log4j.rootLogger=DEBUG, A1
# A1 is set to be a ConsoleAppender which outputs to System.out.
log4j.appender.A1=org.apache.log4j.ConsoleAppender
# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
# The conversion pattern uses format specifiers. You might want to
# change the pattern an watch the output format change.
#log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n
log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %20c - %m%n
* web.xml
<!DOCTYPE web-app PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>亮哥教育</display-name>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
  </context-param>
  <filter>
```

```
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filte
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
  </filter>
  <filter>
    <filter-name>hiddenHttpMethodFilter</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter
  </filter>
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>hiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</list
  </listener>
  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-c
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>

*/ 转班案例
@Controller
@RequestMapping("/emp")
public class TransferEmployeeController {
    @Autowired
    private TransferEmployeeService transferEmployeeService;
    @RequestMapping("/transfer")
    public String transfer(String empno,String job,Integer odid,Integer ndid){
        Map<String,Object> params=new HashMap<>();
        params.put("empno",empno);
        params.put("job",job);
        params.put("odid",odid);
        params.put("ndid",ndid);
        transferEmployeeService.transferEmployee(params);
        return "transferSuccess";
    }
}

* 单元测试
 @Test
public void test17() throws Exception {
   String result = mockMvc.perform(delete("/emp/transfer")
                .param("empno","LG008")
                .param("job","前端开发工程师")
                .param("odid","1")
                .param("ndid","2")).
   andExpect(status().isOk()).
                andDo(print()).andReturn().getResponse().getContentAsString();
   System.out.println(result);
}

* 父子容器测试
@RequestMapping("/test14")
 public void test14(HttpServletRequest request, HttpServletResponse response) {
        // 父容器
        WebApplicationContext parentContext = WebApplicationContextUtils.
                getRequiredWebApplicationContext(request.getServletContext());
        int beanCount = parentContext.getBeanDefinitionCount();
        System.out.println("parent-beanCount:" + beanCount);
        String[] beanDefinitionNames = parentContext.getBeanDefinitionNames();
        for (String beanName : beanDefinitionNames) {
            System.out.println("parent:" + beanName);
        }

        //子容器
        WebApplicationContext childContext = (WebApplicationContext) request.
                getAttribute(DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUT
        beanCount = childContext.getBeanDefinitionCount();
        System.out.println("child-beanCount:" + beanCount);
        beanDefinitionNames = childContext.getBeanDefinitionNames();
        for (String beanName : beanDefinitionNames) {
            System.out.println("child:" + beanName);
        }
    }
* 用浏览器测试
 * http://localhost:8080/lgssm/test/test14

* 作业（零XML配置）
  * 可以参考百度
* SSM整合--智慧汽车项目（两天时间）：零XML配置
```