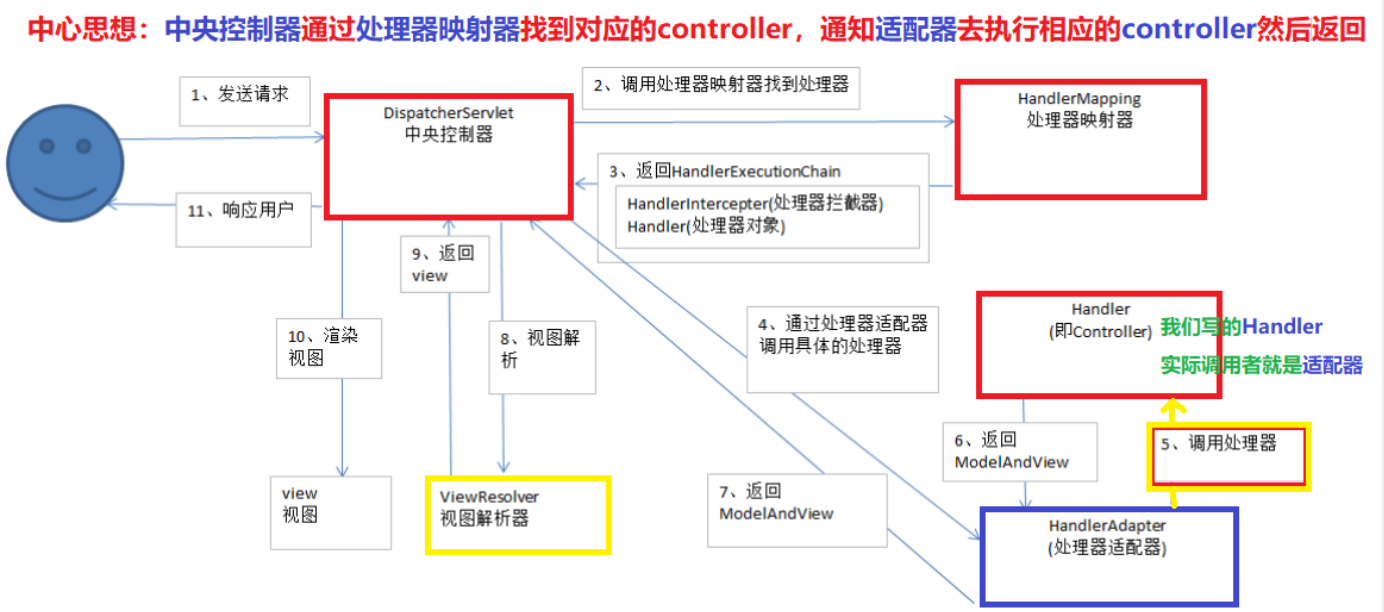
SpringMVC学习笔记

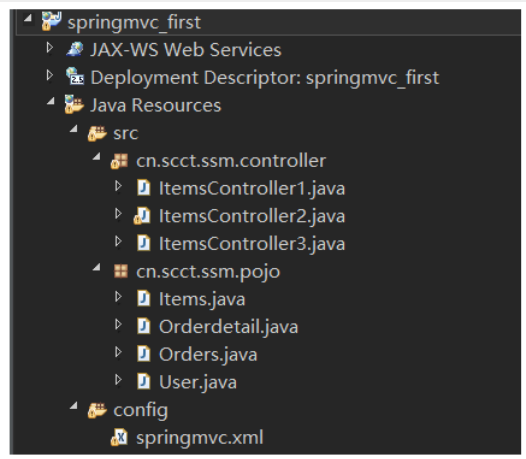
一、SpringMVC介绍

二、SpringMVC架构流程图

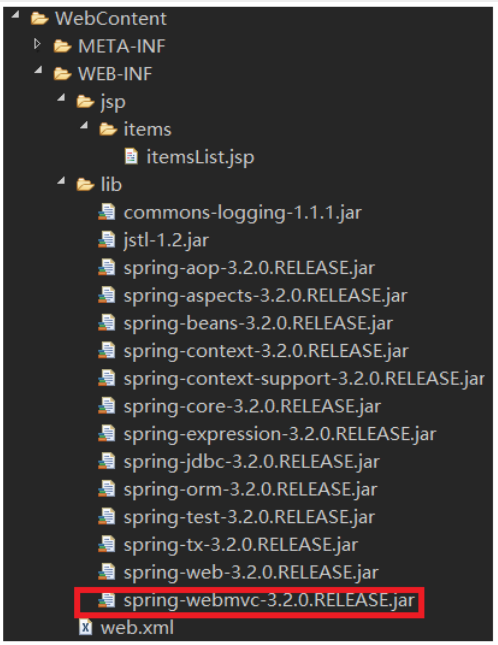


三、SpringMVC入门

3.1 项目结构



springmvc只是spring的一套组件



3.2 配置前端控制器DispatcherServlet

```
<!-- springmvc前端控制器 -->
<servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <!-- contextConfigLocation配置springmvc加载的配置文件（配置处理器映射器、适配器等等）
  如果不配置contextConfigLocation，默认加载的是/WEB-INF/servlet名称-serlvet.xml（springmvc-servlet.xml）
   -->
  <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <!--
  <!--
      1. /*  拦截所有 jsp  js png .css  真的全拦截 建议不使用
      2. *.action *.do 拦截以do action 结尾的请求  肯定能使用 ERP
      3. /  拦截所有（不包括jsp）（包含.js .png.css）  强烈建议使用  前台 面向消费者 www.jd.com/search    /对静态资源放行
   -->

   -->
  <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

1.配置前端控制器 具体完整类名要会找呀

2.配置包含其余“器”的配置文件的位置

3.配置访问后缀名

3.3 处理器映射器HandlerMapping配置及其详解

3.3.1 非注解方式配置

```
<!-- 处理器映射器 将bean的name作为url进行查找，需要在配置Handler时指定beanname（就是url）
    所有的映射器都实现 HandlerMapping接口 -->
非注解
<!-- 配置处理器映射器
    class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />

<!--简单url映射 -->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <!-- 对itemsController1进行url映射，url是/queryItems1.action -->
            <prop key="/queryItems1.action">itemsController1</prop>
            <prop key="/queryItems2.action">itemsController1</prop>
            <prop key="/queryItems3.action">itemsController2</prop>
        </props>
    </property>
</bean>

<!-- 配置Handler -->
<bean id="itemsController1" name="/queryItems_test.action"
    class="cn.scct.ssm.controller.ItemsController1" />
<!-- 配置另外一个Handler -->
<bean id="itemsController2" class="cn.scct.ssm.controller.ItemsController2" />
```

方法一：用这个对应Handler中的name属性，也就是访问的url

方法二：将没有Handler name属性(表示url访问路径)的Handler配置其url

也就是说，允许配置多个访问路径

配置handler

3.3.2 注解方式配置

```
<!--注解映射器 --> //这个会与适配器合体统一注解配置
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>
```

3.4 处理器Handler配置及其详解

3.4.1 非注解方式配置

```
<!-- 配置Handler --> 其实和普通的bean差不多  只是这里的name是个url
    <bean id="itemsController1" name="/queryItems_test.action"
        class="cn.scct.ssm.controller.ItemsController1" />          Handler配置方法一
<!-- 配置另外一个Handler -->
    <bean id="itemsController2" class="cn.scct.ssm.controller.ItemsController2" />
```

3.4.2 注解方式配置

```
<!-- 对于注解的Handler可以单个配置
    实际开发中建议使用组件扫描
     -->
    <!-- <bean class="cn.itcast.ssm.controller.ItemsController3" /> -->
    <!-- 可以扫描controller、service、...
    这里让扫描controller，指定controller的包
     -->     提醒：在spring学习中提到过spring注解开发时用的是这个标签
    <context:component-scan base-package="cn.scct.ssm.controller"></context:component-scan>
```

3.5 处理器适配器配置及其详解

3.5.1 非注解方式配置

```
                                                    对应的controller
<!-- 处理器适配器 所有处理器适配器都实现 HandlerAdapter接口 -->
<!-- 要求的handler实现了controller接口 -->   两个非注解的方式的适配器，分别需要实现两个不同的接口
    <bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter" />
<!-- 另一个非注解的适配器 要求编写的Handler实现 HttpRequestHandler接口-->
    <bean class="org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter"/>
```

3.5.2 注解方式配置

```
<!--注解适配器 -->                                          //注解的适配器
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>
                    当然也可以和映射器合体，之后再讲
```

3.6 视图解析器ViewResolver及其详解

```
前缀+逻辑视图名+后缀，逻辑视图名需要在controller中返回ModelAndView指定，比如逻辑视图名为hello，则最
终返回的jsp视图地址 “WEB-INF/jsp/hello.jsp”
```

```
<!-- 视图解析器
解析jsp解析 默认使用jstl标签，classpath下的得有jstl的包
 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- 配置jsp路径的前缀 -->
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <!-- 配置jsp路径的后缀 -->
    <property name="suffix" value=".jsp"/>

</bean>
```

3.7 书写Handler

```jsp
<c:forEach items="${itemsList }" var="item">
<tr>
    <td>${item.name }</td>
    <td>${item.price }</td>
    <td><fmt:formatDate value="${item.createtime}" pattern="yyyy-MM-dd HH:mm:ss"/></td>
    <td>${item.detail }</td>

    <td><a href="${pageContext.request.contextPath }/item/editItem.action?id=${item.id}">修改</a></td>

</tr>
```

JSTL

转发的页面

3.7.1 对应于非注解方式之实现controller接口

```java
public class ItemsController1 implements Controller {

    @Override
    public ModelAndView handleRequest(HttpServletRequest request,
            HttpServletResponse response) throws Exception {

        //调用service查找 数据库，查询商品列表，这里使用静态数据模拟
        List<Items> itemsList = new ArrayList<Items>();
        //向list中填充静态数据

        Items items_1 = new Items();
        items_1.setName("联想笔记本");
        items_1.setPrice(6000f);
        items_1.setDetail("ThinkPad T430联想笔记本电脑！");

        Items items_2 = new Items();
        items_2.setName("苹果手机");
        items_2.setPrice(5000f);
        items_2.setDetail("iphone6苹果手机！");

        itemsList.add(items_1);
        itemsList.add(items_2);

        //返回ModelAndView
        ModelAndView modelAndView =  new ModelAndView();
        //相当于request的setAttribut，在jsp页面中通过itemsList取数据
        modelAndView.addObject("itemsList", itemsList);
        //指定视图
        modelAndView.setViewName("/WEB-INF/jsp/items/itemsList.jsp");

        return modelAndView;
    }
}
```

```xml
<!-- 配置Handler -->
<bean id="itemsController1" name="/queryItems_test.action"
    class="cn.scct.ssm.controller.ItemsController1" />
```

原始的非注解的配置方式，这是url

3.7.2 对应于非注解方式之实现HttpRequestHandler接口

```java
public class ItemsController2 implements HttpRequestHandler {

    @Override
    public void handleRequest(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {

        //调用service查找 数据库，查询商品列表，这里使用静态数据模拟
        List<Items> itemsList = new ArrayList<Items>();
        //向list中填充静态数据

        Items items_1 = new Items();
        items_1.setName("联想笔记本");
        items_1.setPrice(6000f);
        items_1.setDetail("ThinkPad T430 联想笔记本电脑！");

        Items items_2 = new Items();
        items_2.setName("苹果手机");
        items_2.setPrice(5000f);
        items_2.setDetail("iphone6苹果手机！");

        itemsList.add(items_1);
        itemsList.add(items_2);
        //设置模型数据
        request.setAttribute("itemsList", itemsList);
        //设置转发的视图
        request.getRequestDispatcher("/WEB-INF/jsp/items/itemsList.jsp").forward(request, response);
    }


}
```

//提醒一下，这个controller的映射路径是这个

```
<prop key="/queryItems3.action">itemsController2</prop>
```

设置模型

视图地址

3.7.3 对应于注解方式的Handler

```java
//使用Controller标识 它是一个控制器
@Controller
public class ItemsController3 {

    //商品查询列表
    //@RequestMapping实现 对queryItems方法和url进行映射，一个方法对应一个url
    //一般建议将url和方法写成一样
    @RequestMapping("/queryItems")
    public ModelAndView queryItems()throws Exception{

        //调用service查找 数据库，查询商品列表，这里使用静态数据模拟
        List<Items> itemsList = new ArrayList<Items>();
        //向list中填充静态数据

        Items items_1 = new Items();
        items_1.setName("联想笔记本");
        items_1.setPrice(6000f);
        items_1.setDetail("ThinkPad T430 联想笔记本电脑！");

        Items items_2 = new Items();
        items_2.setName("苹果手机");
        items_2.setPrice(5000f);
        items_2.setDetail("iphone6苹果手机！");

        itemsList.add(items_1);
        itemsList.add(items_2);

        //返回ModelAndView
        ModelAndView modelAndView =  new ModelAndView();
        //相当 于request的setAttribut，在jsp页面中通过itemsList取数据
        modelAndView.addObject("itemsList", itemsList);

        //指定视图
        //下边的路径，如果在视图解析器中配置jsp路径的前缀和jsp路径的后缀，修改为
//      modelAndView.setViewName("/WEB-INF/jsp/items/itemsList.jsp");
        //上边的路径配置可以不在程序中指定jsp路径的前缀和jsp路径的后缀
        modelAndView.setViewName("items/itemsList");

        return modelAndView;
    }
```

其实在spring中有关注解的方式已经说明了

这个controller注解

```
@Component("user")
//相当于<bean name="user" class="cn.itcast.bean.User"
//@Service("user") // service层
//@Controller("user") // web层
//@Repository("user")// dao层
```

//配置的url路径，在前端控制器中已经配置了.action结尾，这里可以写.action，也可以不写，但访问的时候必须写.action

已经指明了前缀和后缀，在视图解析器中

3.8 测试

略，自然很容易懂在做啥

3.9 简化开发方式

```
<!-- 对于注解的Handler可以单个配置
    实际开发中建议使用组件扫描
     -->
    <!-- <bean class="cn.itcast.ssm.controller.ItemsController3" /> -->
    <!-- 可以扫描controller、service、...
    这里让扫描controller，指定controller的包
     -->
    <context:component-scan base-package="cn.scct.ssm.controller"></context:component-scan>
```

注解的Handler

```
<!-- 使用mvc:annotation-driven代替上边注解映射器和注解适配器配置
mvc:annotation-driven默认加载很多的参数绑定方法，
比如json转换解析器就默认加载了，如果使用mvc:annotation-driven不用配置上边的RequestMappingHandlerMapping和RequestMappingHa
实际开发时使用mvc:annotation-driven
 -->
<mvc:annotation-driven></mvc:annotation-driven>
```

注解的映射器和适配器

```
88      <!-- 视图解析器
89      解析jsp解析，默认使用jstl标签，classpath下的得有jstl的包
90       -->
91      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
92          <!-- 配置jsp路径的前缀 -->
93          <property name="prefix" value="/WEB-INF/jsp/"/>
94          <!-- 配置jsp路径的后缀 -->
95          <property name="suffix" value=".jsp"/>
96
97      </bean>
```

//配置视图解析器

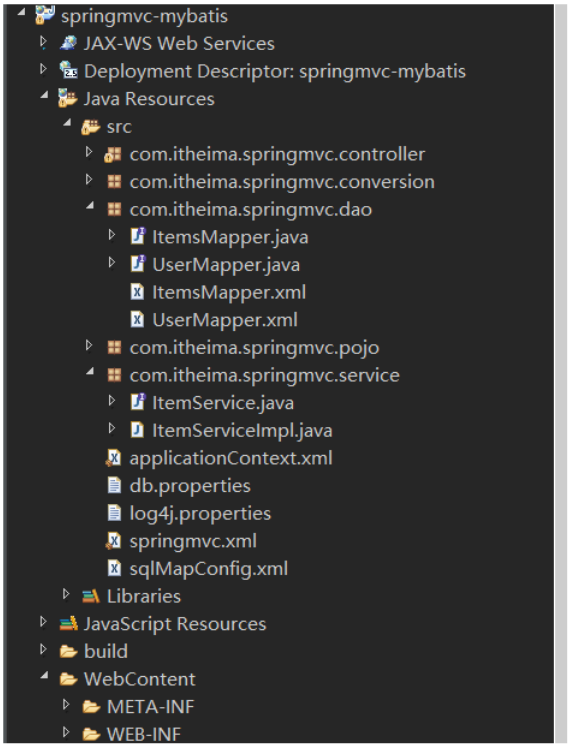
Handler开发就用对应的注解开发方式，对应于3.7.3

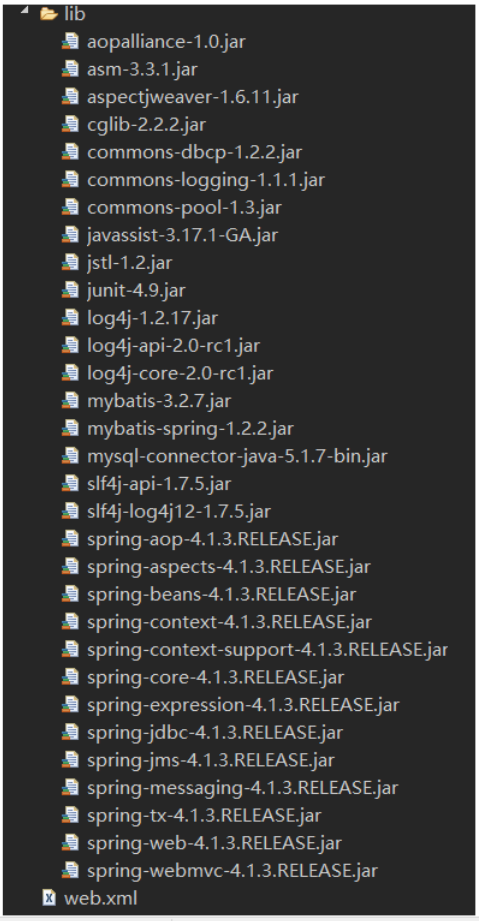
四、SpringMVC三大组件详解

对应于第三节中的介绍，不谈。

五、SpringMVC整合Mybatis

5.1 目录结构和导包





5.2 Spring及SpringMVC方面

5.2.1 Spring容器需要随web启动

```
<!-- 让spring随web启动而创建的监听器 -->
<listener>                                    注意不要忘记，不然手动去加载spring配置文件吗
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- 配置spring配置文件位置参数 -->                  在web.xml中配置
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:applicationContext.xml</param-value>
</context-param>
```

5.2.2 SpringMVC配置文件springmvc.xml

5.2.3 在web.xml配置SpringMVC的前端控制器

```
<!-- 前端控制器 -->
<servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <!-- 默认找/WEB-INF/[servlet的名称]-servlet.xml -->
  <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <!--
      1. /*  拦截所有 jsp  js png .css  真的全拦截 建议不使用
      2. *.action *.do 拦截以do action 结尾的请求  肯定能使用 ERP
      3. /  拦截所有（不包括jsp）（包含.js .png.css）  强烈建议使用  前台面向消费者 www.jd.com/search    /对静态资源放行
   -->
  <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

这是自然，这说明Spring与SpringMVC是无缝整合，其实SpringMVC是Spring的一个插件，根本不需要整合包

5.3 Mybatis方面

5.3.1 sqlMapConfig.xml文件配置

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 设置别名 -->
    <typeAliases>
        <!-- 2. 指定扫描包，会把包内所有的类都设置别名，别名的名称就是类名，大小写不敏感 -->
        <package name="com.itheima.springmvc.pojo" />
    </typeAliases>

</configuration>
```

所以特别简单

在applicationContext.xml中

这里没有配置mapper属性，是因为，spring与mybatis整合后有一个扫描基本包的配置，前提是mapper类与mapper文件名称相同，一一对应

5.3.2 各种mapper类及其映射文件

5.3.3 mybatis相关对象托付给spring管理

```
<context:property-placeholder location="classpath:db.properties"/>

<!-- 数据库连接池 -->          数据源
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="${jdbc.driver}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
    <property name="maxActive" value="10" />
    <property name="maxIdle" value="5" />
</bean>
```

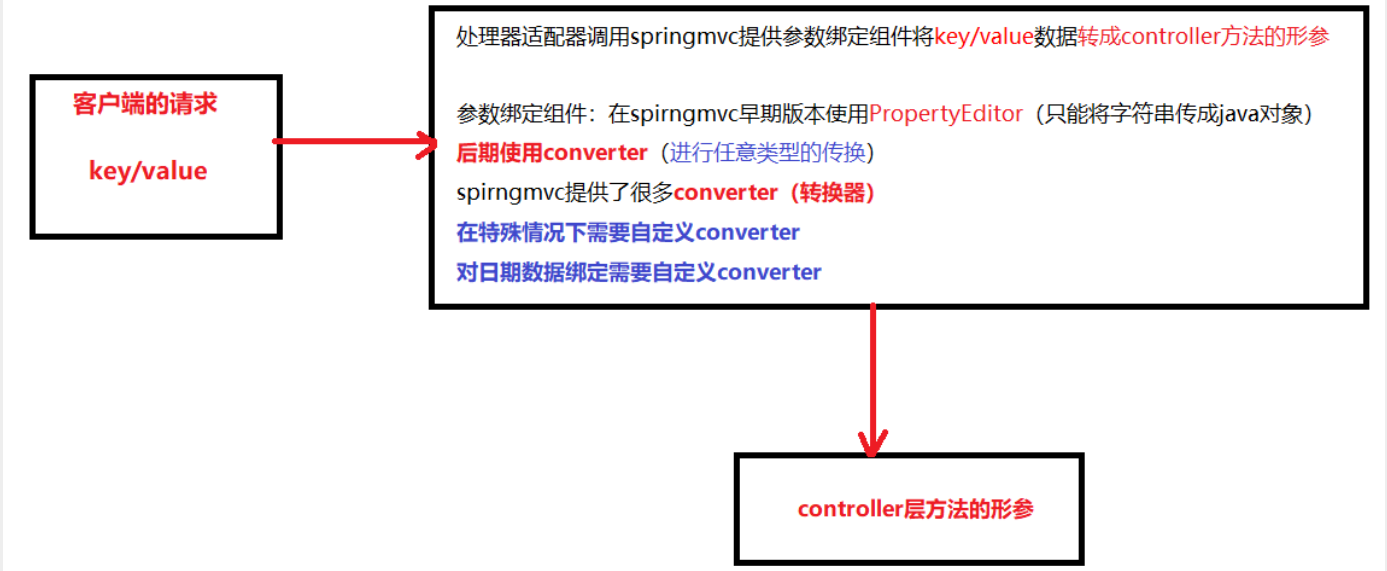
```
<!-- Mybatis的工厂 -->                    sqlSessionFactory工厂
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>          //需要数据源
    <!-- 核心配置文件的位置 -->
    <property name="configLocation" value="classpath:sqlMapConfig.xml"/>          //全局配置文件的位置
</bean>

<!-- Mapper动态代理开发 扫描 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <!-- 基本包 -->
    <property name="basePackage" value="com.itheima.springmvc.dao"/>          //都能扫到并交付给spring管理
</bean>
```

5.4 其他方面

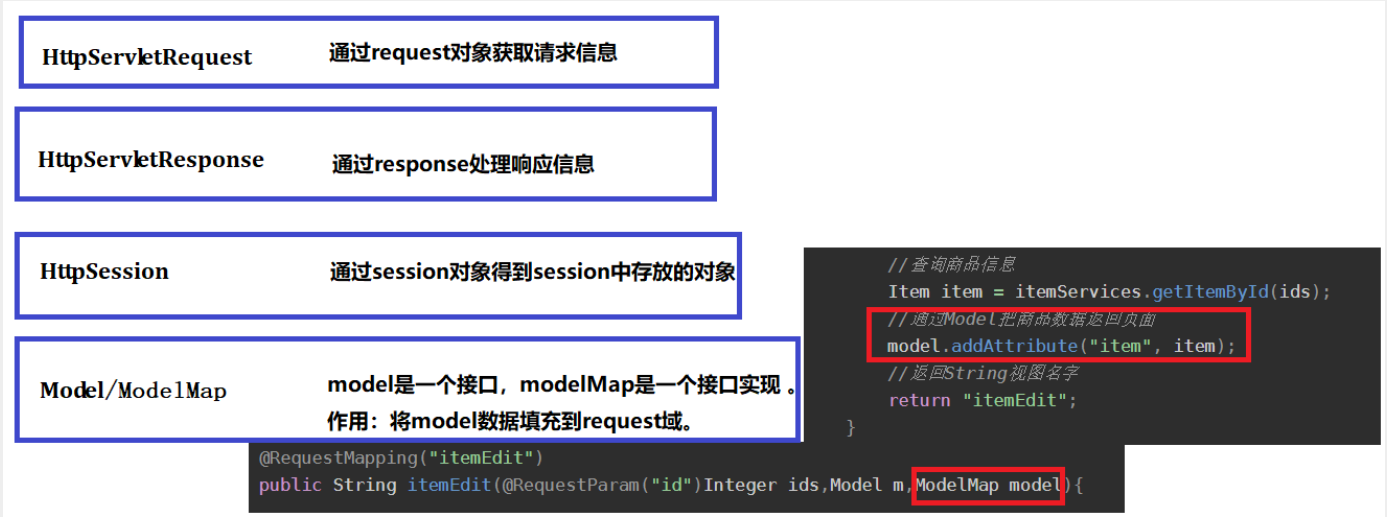
比如开启事务，自己配就是，没啥好说的

六、controller层参数绑定

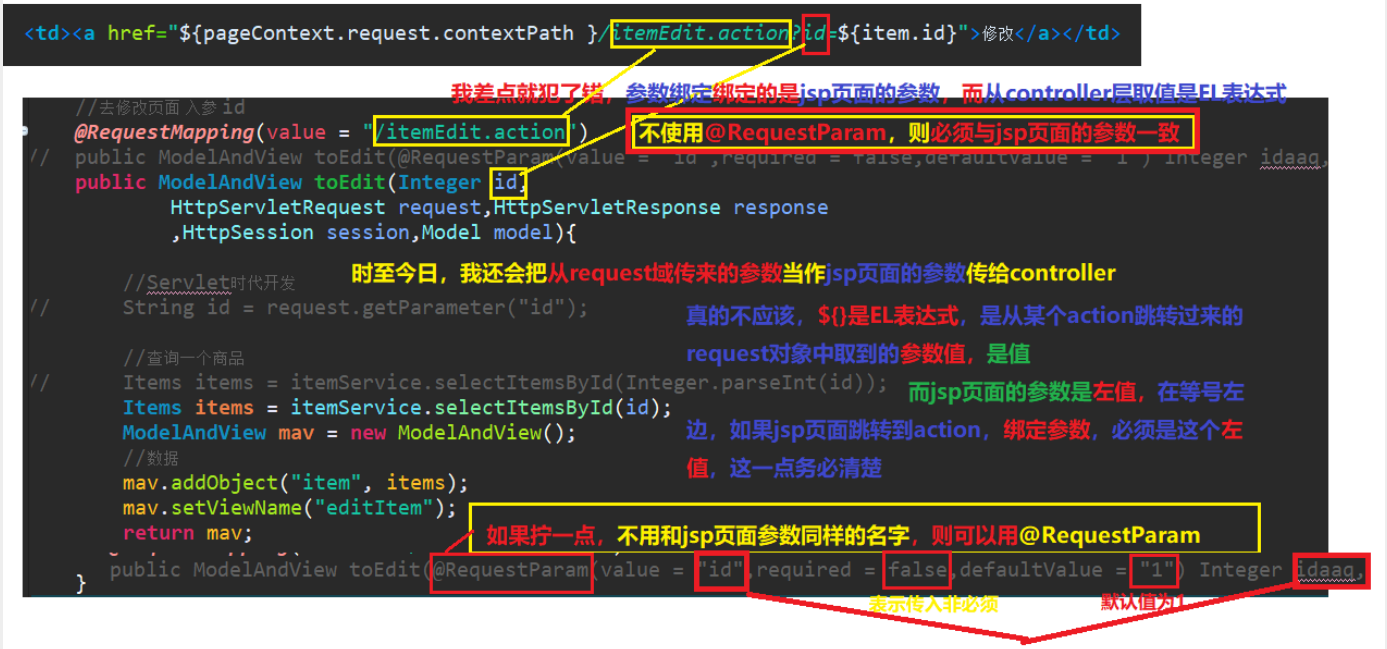
6.1 参数绑定过程



6.2 默认支持的类型



6.3 简单类型



6.4 pojo类型

6.5 包装类型pojo

6.6 集合类型

6.6.1 数组类型

6.6.2 list绑定

6.6.3 map绑定

在包装类中定义 Map 对象，并添加 get/set 方法，action 使用包装对象接收。
包装类中定义 Map 对象如下：

```
Public class QueryVo {
private Map<String, Object> itemInfo = new HashMap<String, Object>();
  //get/set 方法..
}
```

```
<tr>
<td>学生信息：</td>
<td>
姓名：<inputtype="text"name="itemInfo['name']"/>
年龄：<inputtype="text"name="itemInfo['price']"/>
.. .. ..
</td>
</tr>
```

```
public String useraddsubmit(Model model,QueryVo queryVo)throws Exception{
System.out.println(queryVo.getStudentinfo());
}
```

七、controller层返回值类型

7.1 ModelAndView

1.ModelAndView 无敌的 带着数据 返回视图路径 不建议使用

■ 返回 ModelAndView

需要方法结束时，定义 ModelAndView，将 model 和 view 分别进行设置。

7.2 String类型

1、表示返回逻辑视图名。
真正视图(jsp 路径)=前缀+逻辑视图名+后缀

```
@RequestMapping(value="/editItems",method={RequestMethod.POST,RequestMethod.GET})
public String editItems(Model model)throws Exception {

    //调用service根据商品id查询商品信息
    ItemsCustom itemsCustom = itemsService.findItemsById(1);

    //通过形参中的model将model数据传到页面
    //相当于modelAndView.addObject方法
    model.addAttribute("itemsCustom", itemsCustom);

    return "items/editItems";
}
```

3、forward 页面转发
通过 forward 进行页面转发，浏览器地址栏 url 不变，request 可以共享。

```
//页面转发
return "forward:queryItems.action";
```

2、redirect 重定向
商品修改提交后，重定向到商品查询列表。
redirect 重定向特点：浏览器地址栏中的 url 会变化。修改提交的 request 数据无法传到重定向的地址。因为重定向后重新进行 request（request 无法共享）

```
//重定向到商品查询列表
return "redirect:queryItems.action";
```

7.3 void类型

在 controller 方法形参上可以定义 request 和 response，使用 request 或 response 指定响应结果：

1、使用 request 转向页面 如下：

```
request.getRequestDispatcher("页面路径").forward(request, response);
```

2、也可以通过 response 页面重定向

```
response.sendRedirect("url")
```

3、也可以通过 response 指定响应结果，例如响应 json 数据如下：

```
response.setCharacterEncoding("utf-8");
response.setContentType("application/json;charset=utf-8");
response.getWriter().write("json 串");
```

八、@RequestMapping

8.1 基础用法

```
// 为了对url进行分类管理，可以在这里定义根路径，最终访问url是根路径+子路径
// 比如：商品列表：/items/queryItems.action
@RequestMapping("/items")
public class ItemsController {
```

```
    // 商品查询
    @RequestMapping("/queryItems")
    public ModelAndView queryItems(HttpServletRequest request,
            ItemsQueryVo itemsQueryVo) throws Exception {
        // 测试forward后request是否可以共享
```

8.2 处理多个url

```
@RequestMapping(value = {"/item/itemlist.action","/item/itemlisthaha.action"})
```

8.3 与@RequestParam配合使用

```
@RequestMapping(value = "/id")
String getIdByValue(@RequestParam("id") String personId) {  }
```

8.4 指定http请求的各种方法

```
//修改
@RequestMapping(value = "/updates.action",method = {RequestMethod.POST,RequestMethod.GET})
```

8.5 处理生产和消费者对象(待续)

8.6 处理请求参数(待续)

8.7 处理动态url(待续)

九、SpringMVC其他说明

9.1 乱码解决

```xml
<!-- post乱码过虑器 -->
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

对于 get 请求中文参数出现乱码解决方法有两个：

修改 tomcat 配置文件添加编码与工程编码一致，如下：

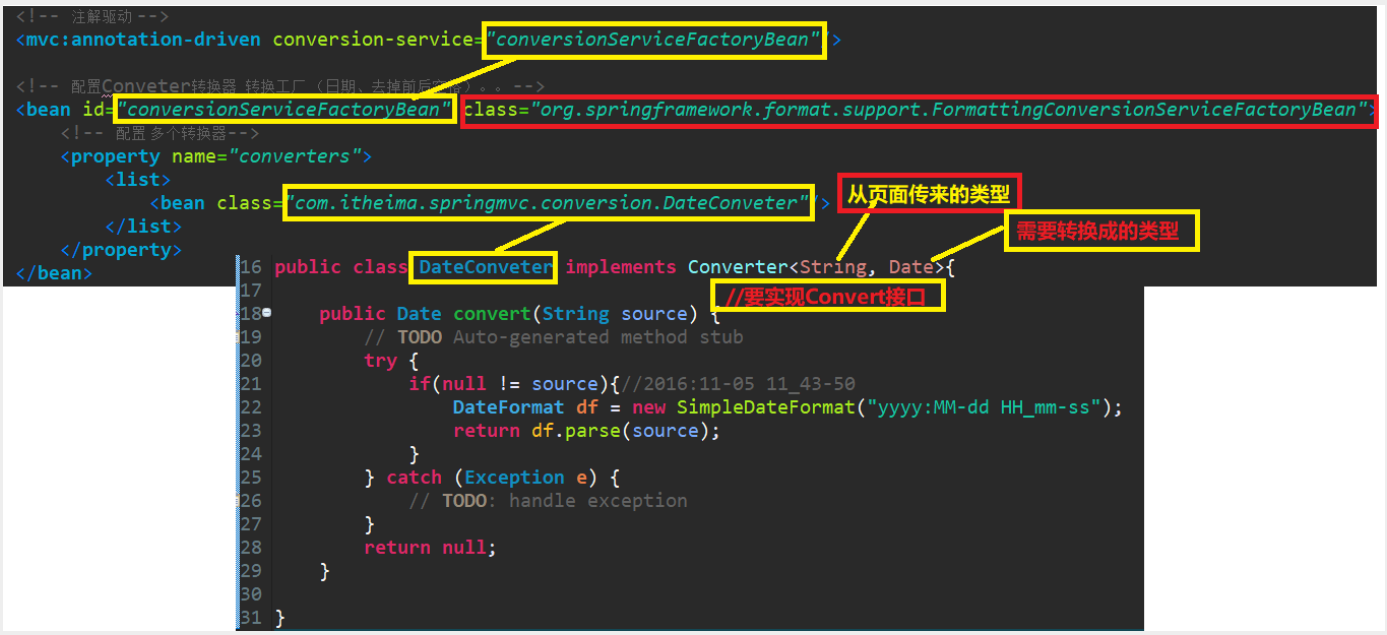
```
<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>
```

另外一种方法对参数进行重新编码：

```
String userName new
String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")
```

ISO8859-1 是 tomcat 默认编码，需要将 tomcat 编码后的内容按 utf-8 编码

9.2 自定义参数绑定(以日期为例)

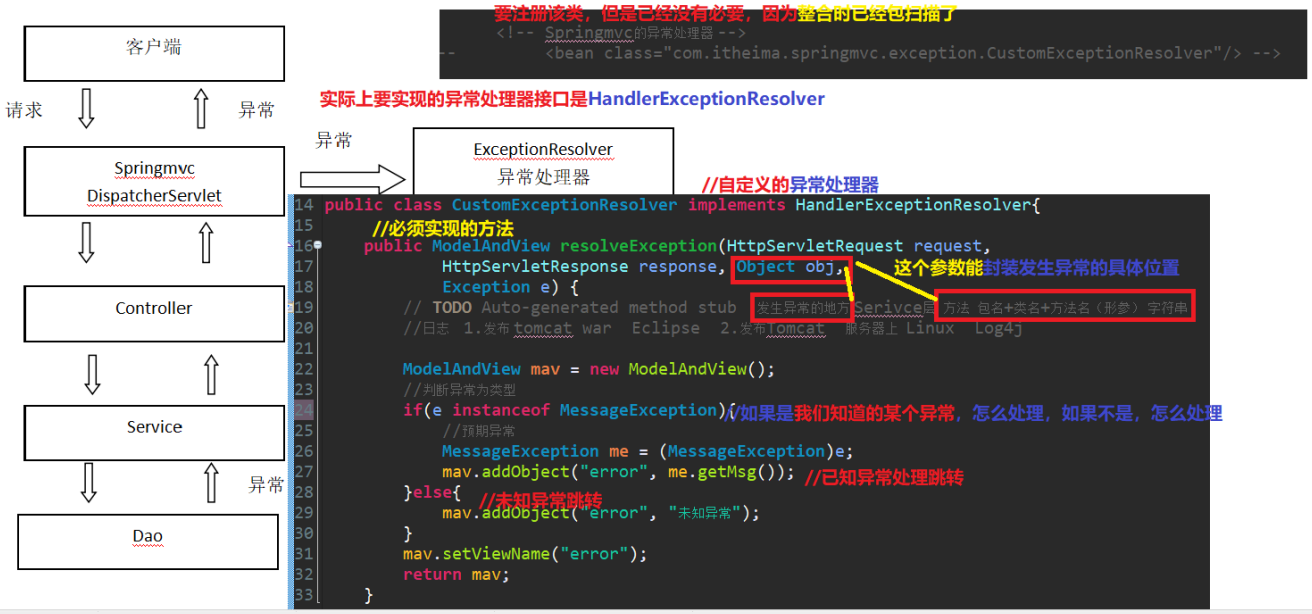



```xml
<!-- 注解驱动 -->
<mvc:annotation-driven conversion-service="conversionServiceFactoryBean" />

<!-- 配置Conveter转换器 转换工厂（日期、去掉前后空格）。。。 -->
<bean id="conversionServiceFactoryBean" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <!-- 配置多个转换器-->
    <property name="converters">
        <list>
            <bean class="com.itheima.springmvc.conversion.DateConveter" />
        </list>
    </property>
</bean>
```

```java
public class DateConveter implements Converter<String, Date>{

    public Date convert(String source) {
        // TODO Auto-generated method stub
        try {
            if(null != source){//2016:11-05 11_43-50
                DateFormat df = new SimpleDateFormat("yyyy:MM-dd HH_mm-ss");
                return df.parse(source);
            }
        } catch (Exception e) {
            // TODO: handle exception
        }
        return null;
    }

}
```

9.3 SpringMVC异常处理

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，如下图：



9.4 上传图片

9.5 json数据交互

9.6 Restful风格开发

9.7 拦截器

9.8 Springmvc校验

9.9 SpringMVC数据回显

十、SpringMVC与Struts2的区别