springmvc 第一天 springmvc 的基础知识

课程安排：

第一天：springmvc 的基础知识
什么是 springmvc？
springmvc 框架原理（掌握）
　　前端控制器、处理器映射器、处理器适配器、视图解析器
springmvc 入门程序
　　目的：对前端控制器、**处理器映射器、处理器适配器**、视图解析器学习
　　非注解的处理器映射器、处理器适配器
　　注解的处理器映射器、处理器适配器（掌握）
springmvc 和 mybatis 整合（掌握）

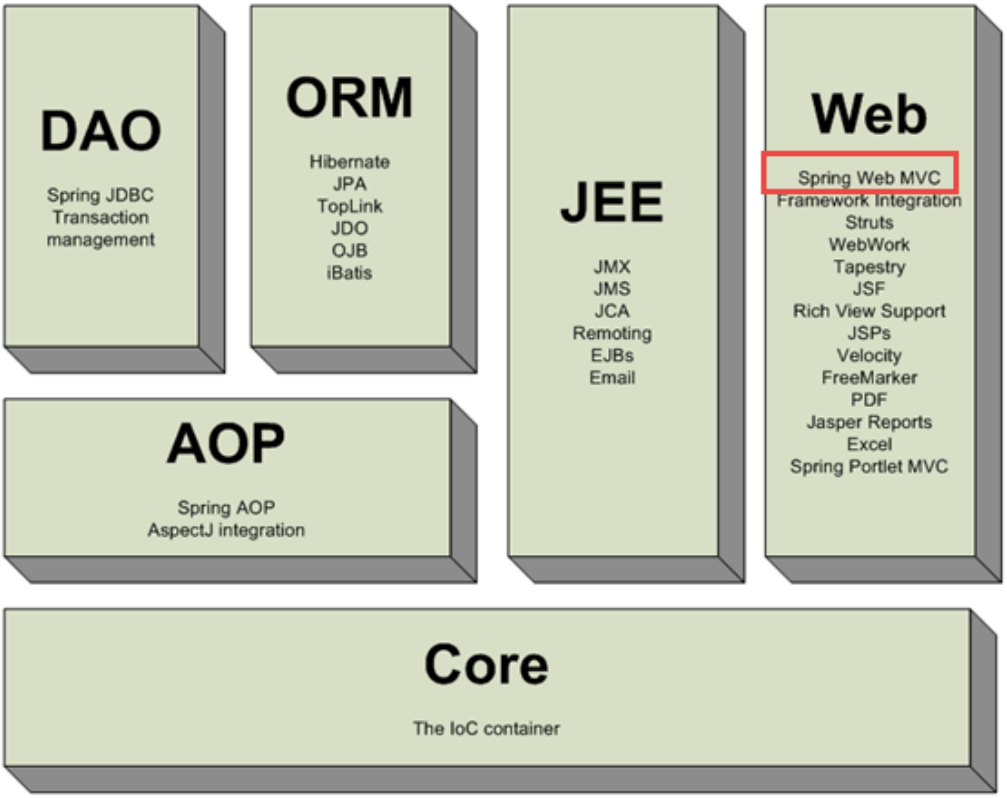
springmvc 注解开发：（掌握）
　　常用的注解学习
　　参数绑定（简单类型、pojo、集合类型（明天讲））
　　自定义参数绑定（掌握）
springmvc 和 struts2 区别

第二天：springmvc 的高级应用
　　参数绑定（集合类型）
　　数据回显
　　上传图片
　　json 数据交互
　　RESTful 支持
　　拦截器

1 springmvc框架

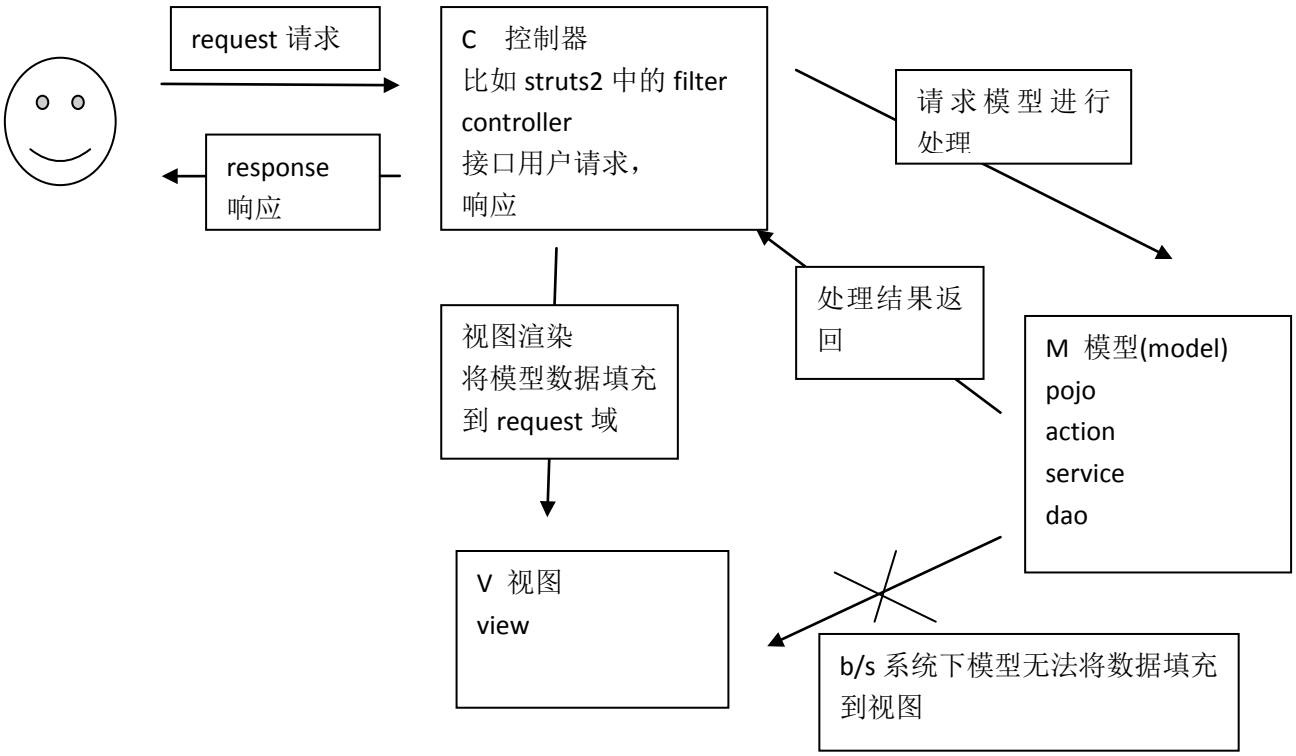
1.1 什么是springmvc

springmvc 是 spring 框架的一个模块，springmvc 和 spring 无需通过中间整合层进行整合。
springmvc 是一个基于 mvc 的 web 框架。

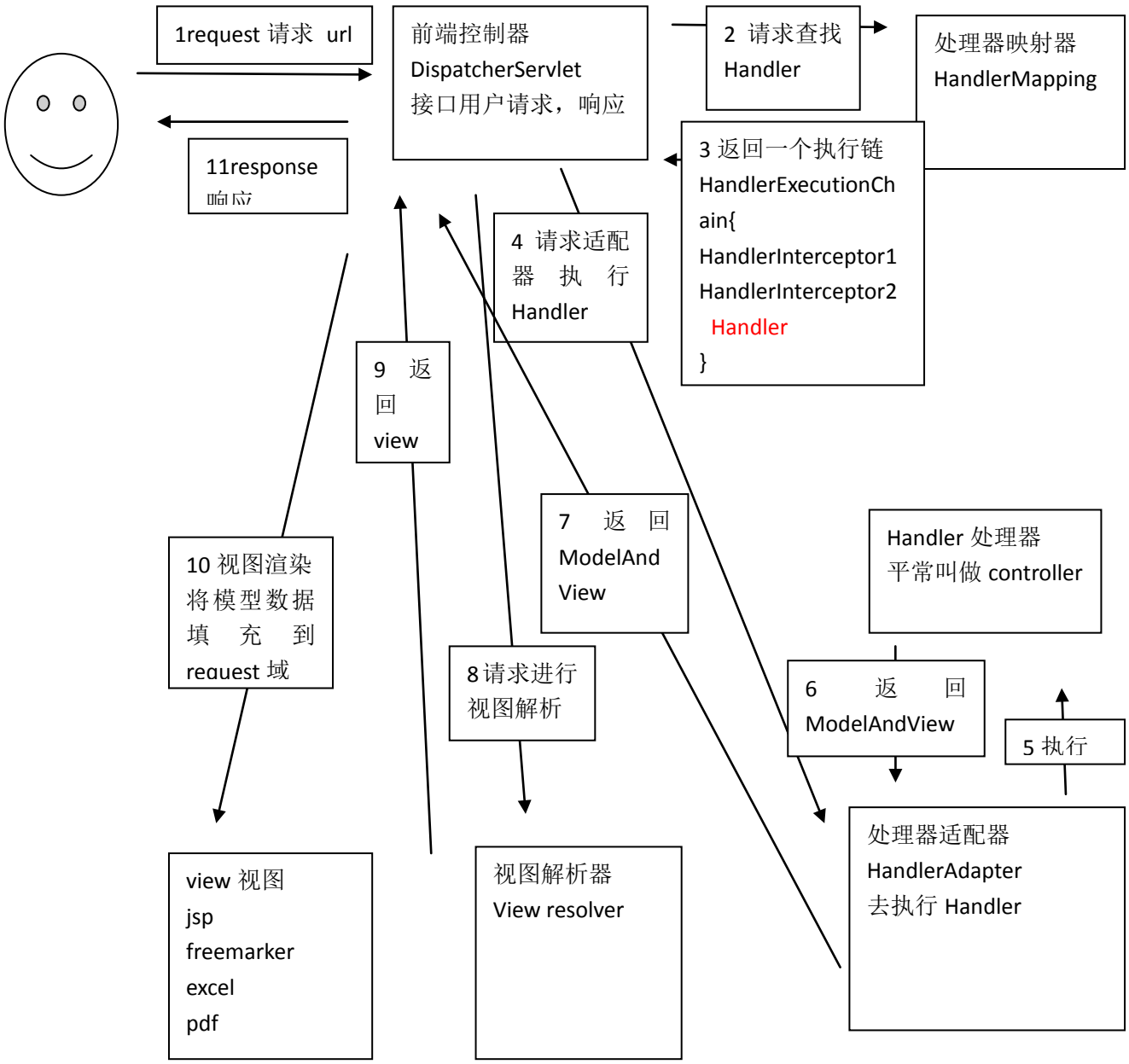


1.2 mvc在b/s系统 下的应用

mvc 是一个设计模式，mvc 在 b/s 系统 下的应用：



1.3 springmvc框架



第一步：发起请求到前端控制器(DispatcherServlet)

第二步：前端控制器请求 HandlerMapping 查找 Handler

可以根据 xml 配置、注解进行查找

第三步：处理器映射器 HandlerMapping 向前端控制器返回 Handler

第四步：前端控制器调用处理器适配器去执行 Handler

第五步：处理器适配器去执行 Handler

第六步：Handler 执行完成给适配器返回 ModelAndView
第七步：处理器适配器向前端控制器返回 ModelAndView
　　ModelAndView 是 springmvc 框架的一个底层对象，包括 Model 和 view
第八步：前端控制器请求视图解析器去进行视图解析
　　根据逻辑视图名解析成真正的视图(jsp)
第九步：视图解析器向前端控制器返回 View
第十步：前端控制器进行视图渲染
　　视图渲染将模型数据(在 ModelAndView 对象中)填充到 request 域
第十一步：前端控制器向用户响应结果

组件：
1、前端控制器 DispatcherServlet（不需要程序员开发）
作用接收请求，响应结果，相当于转发器，中央处理器。
有了 DispatcherServlet 减少了其它组件之间的耦合度。

2、处理器映射器 HandlerMapping(不需要程序员开发)
作用：根据请求的 url 查找 Handler

3、处理器适配器 HandlerAdapter
作用：按照特定规则（HandlerAdapter 要求的规则）去执行 Handler

4、处理器 Handler(需要程序员开发)
注意：编写 Handler 时按照 HandlerAdapter 的要求去做，这样适配器才可以去正确执行 Handler

5、视图解析器 View resolver(不需要程序员开发)
作用：进行视图解析，根据逻辑视图名解析成真正的视图（view）

6、视图 View(需要程序员开发 jsp)
View 是一个接口，实现类支持不同的 View 类型（jsp、freemarker、pdf...）

2 入门程序

2.1 需求

以案例作为驱动。
springmvc 和 mybaits 使用一个案例（商品订单管理）。

功能需求：商品列表查询

2.2 环境准备

数据库环境：mysql5.1

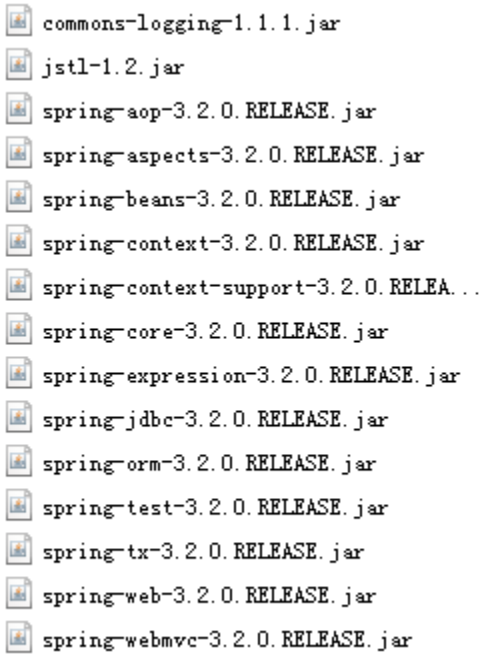
java 环境：

jdk1.7.0_72

eclipse indigo

springmvc 版本：spring3.2

需要 spring3.2 所有 jar（一定包括 spring-webmvc-3.2.0.RELEASE.jar）



2.3 配置前端控制器

在 web.xml 中配置前端控制器。

```
<!-- 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>
  <!--
  第一种：*.action，访问以.action结尾 由DispatcherServlet进行解析
  第二种：/，所以访问的地址都由DispatcherServlet进行解析，对于静态文件的解析需要配置不让DispatcherServlet进行解析
  使用此种方式可以实现 RESTful风格的url
  第三种：/*，这样配置不对，使用这种配置，最终要转发到一个jsp页面时，
  仍然会由DispatcherServlet解析jsp地址，不能根据jsp页面找到handler，会报错。

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

2.4 配置处理器适配器

在 classpath 下的 springmvc.xml 中配置处理器适配器

```
<!-- 处理器适配器
所有处理器适配器都实现 HandlerAdapter接口
-->
<bean
    class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter" />
```

通过查看原代码：

```
public class SimpleControllerHandlerAdapter implements HandlerAdapter {

    public boolean supports(Object handler) {
        return (handler instanceof Controller);
    }
```

此适配器能执行实现 `Controller` 接口的 `Handler`。

```java
public interface Controller {

    /**
     * Process the request and return a ModelAndView object which the DispatcherServlet
     * will render. A <code>null</code> return value is not an error: It indicates that
     * this object completed request processing itself, thus there is no ModelAndView
     * to render.
     * @param request current HTTP request
     * @param response current HTTP response
     * @return a ModelAndView to render, or <code>null</code> if handled directly
     * @throws Exception in case of errors
     */
    ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception;

}
```

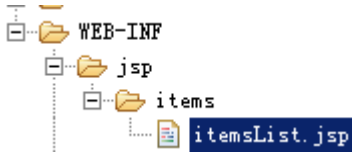
2.5 开发Handler

需要实现 controller接口，才能由org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter适配器执行。

```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;
```

```
    }
}
```

2.6 视图编写



2.7 配置Handler

将编写 Handler 在 spring 容器加载。

```
<!-- 配置Handler -->
<bean name="/queryItems.action" class="cn.itcast.ssm.controller.ItemsController1"/>
```

2.8 配置处理器映射器

在 classpath 下的 springmvc.xml 中配置处理器映射器

```
<!-- 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）
 -->
<bean
    class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />
```

2.9 配置视图解析器

需要配置解析 jsp 的视图解析器。

```
<!-- 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"></bean>
```

2.10部署调试

访问地址：http://localhost:8080/springmvcfirst1208/queryItems.action

处理器映射器根据 url 找不到 Handler，报下边的错误。说明 url 错误。

处理器映射器根据 url 找到了 Handler，转发的 jsp 页面找到，报下边的错误，说明 jsp 页面地址错误了。

3 非注解的处理器映射器和适配器

3.1 非注解的处理器映射器

处理器映射器：

org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping

另一个映射器：
org.springframework.web.servlet.handler.SimpleUrlHandlerMapping

```
<!--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>
        </props>
    </property>
</bean>
```

多个映射器可以并存，前端控制器判断 url 能让哪些映射器映射，就让正确的映射器处理。

3.2 非注解的处理器适配器

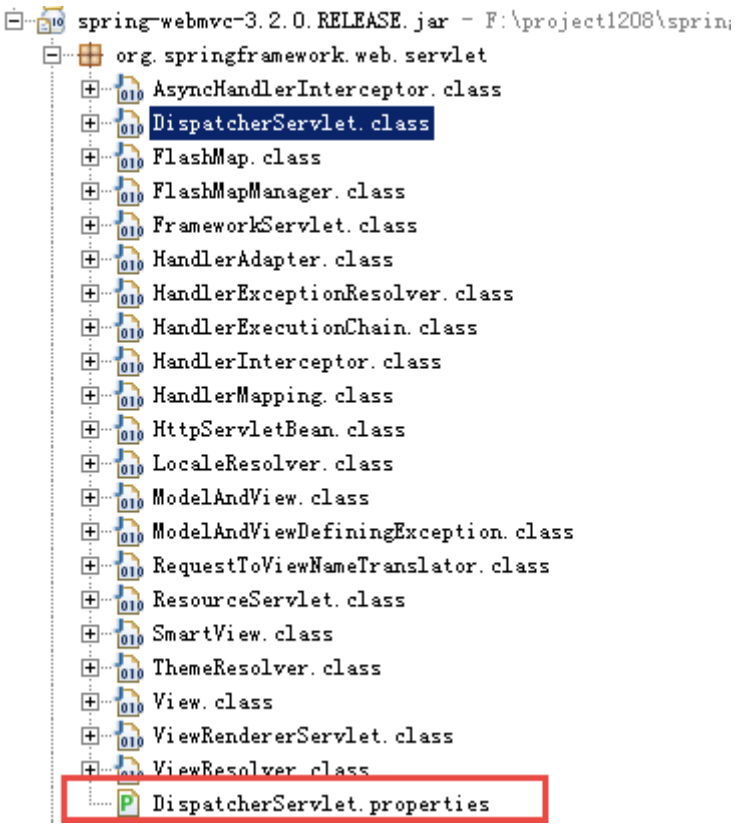
org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter
要求编写的 Handler 实现 Controller 接口。

org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter
要求编写的 Handler 实现 `HttpRequestHandler` 接口。

```
@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);

}
```

```
        //使用此方法可以通过修改response，设置响应的数据格式，比如响应json数据

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

4 DispatcherSerlvet.properties



前端控制器从上边的文件中加载处理映射器、适配器、视图解析器等组件，如果不在 springmvc.xml 中配置，使用默认加载的。

5 注解的处理器映射器和适配器

在 spring3.1 之前使用 org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping 注解映射器。

在 spring3.1 之后使用 org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping 注解

映射器。

在 spring3.1 之前使用 org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter 注解适配器。

在 spring3.1 之后使用 org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter 注解适配器。

5.1 配置注解映射器和适配器。

```
<!--注解映射器 -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>
<!--注解适配器 -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>
```

```
<!-- 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RequestMappingHandlerAdapter
    实际开发时使用mvc:annotation-driven
     -->
    <!-- <mvc:annotation-driven></mvc:annotation-driven> -->
```

5.2 开发注解Handler

使用注解的映射器和注解的适配器。（注解的映射器和注解的适配器必须配对使用）

```
//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
        modelAndView.setViewName("/WEB-INF/jsp/items/itemsList.jsp");

        return modelAndView;

    }
```

5.3 在spring容器中加载Handler

```
    <!-- 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.itcast.ssm.controller"></context:component-scan>
```

5.4 部署调试

访问：http://localhost:8080/springmvcfirst1208/queryItems.action

6 源码分析（了解）

通过前端控制器源码分析 springmvc 的执行过程。

第一步：前端控制器接收请求

调用 doDiapatch

```
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws
    HttpServletRequest processedRequest = request;
    HandlerExecutionChain mappedHandler = null;
    int interceptorIndex = -1;
```

第二步：前端控制器调用处理器映射器查找 Handler

```
// Determine handler for the current request.
mappedHandler = getHandler(processedRequest, false);
```

```
protected HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception {
    for (HandlerMapping hm : this.handlerMappings) {
        if (logger.isTraceEnabled()) {
            logger.trace(
                    "Testing handler map [" + hm + "] in DispatcherServlet with name '" + getServlet
        }
        HandlerExecutionChain handler = hm.getHandler(request);
        if (handler != null) {
            return handler;
        }
    }
    return null;
}
```

第三步：调用处理器适配器执行 Handler，得到执行结果 ModelAndView

```
// Actually invoke the handler.
mv = ha.handle(processedRequest, response, mappedHandler.getHandler());
```

第四步：视图渲染，将 model 数据填充到 request 域。

视图解析，得到 view:

```
// We need to resolve the view name.
view = resolveViewName(mv.getViewName(), mv.getModelInternal(), locale, request);
```

调用 view 的渲染方法，将 model 数据填充到 request 域

渲染方法：

```
view.render(mv.getModelInternal(), request, response);
```

```
protected void exposeModelAsRequestAttributes(Map<String, Object> model, HttpServletRequest request) throws Excep
    for (Map.Entry<String, Object> entry : model.entrySet()) {
        String modelName = entry.getKey();
        Object modelValue = entry.getValue();
        if (modelValue != null) {
            request.setAttribute(modelName, modelValue);
            if (logger.isDebugEnabled()) {
                logger.debug("Added model object '" + modelName + "' of type [" + modelValue.getClass().getName()
                        "] to request in view with name '" + getBeanName() + "'");
            }
        }
        else {
            request.removeAttribute(modelName);
            if (logger.isDebugEnabled()) {
                logger.debug("Removed model object '" + modelName +
                        "' from request in view with name '" + getBeanName() + "'");
            }
        }
    }
}
```

7 入门程序小结

通过入门程序理解 springmvc 前端控制器、**处理器映射器、处理器适配器**、视图解析器用法。

前端控制器配置：

第一种：*.action，访问以.action结尾 由DispatcherServlet进行解析

第二种：/，所以访问的地址都由DispatcherServlet进行解析，对于静态文件的解析需要配置不让DispatcherServlet进行解析

使用此种方式可以实现 RESTful风格的url

处理器映射器：

非注解处理器映射器（了解）

注解的处理器映射器（掌握）

对标记@Controller类中标识有@RequestMapping的方法进行映射。在@RequestMapping里边定义映射的url。使用注解的映射器不用在xml中配置url和Handler的映射关系。

处理器适配器：

非注解处理器适配器（了解）

注解的处理器适配器（掌握）

注解处理器适配器和注解的处理器映射器是配对使用。理解为不能使用非注解映射器进行映射。

<mvc:annotation-driven></mvc:annotation-driven>可以代替下边的配置：

```
    <!--注解映射器 -->
    <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHan
```

```
dlerMapping"/>
    <!--注解适配器 -->
    <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHan
dlerAdapter"/>
```

实际开发使用：mvc:annotation-driven

视图解析器配置前缀和后缀：

```
<!-- 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>
```

程序中不用指定前缀和后缀：

```
//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");
```

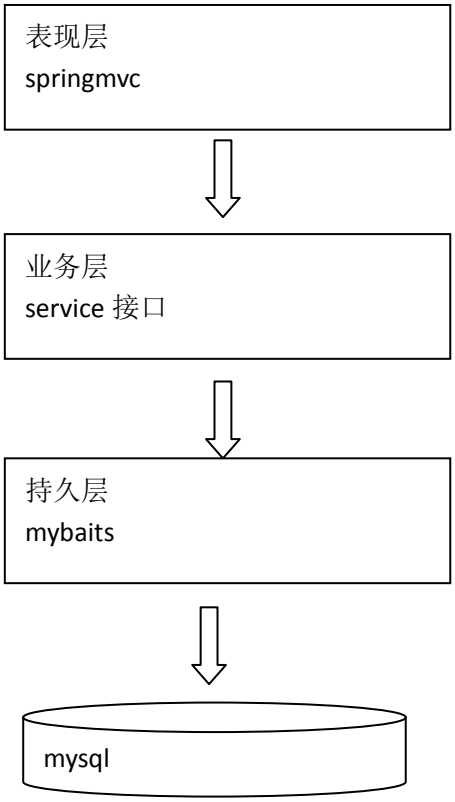
8 springmvc和mybatis整合

8.1 需求

使用 springmvc 和 mybatis 完成商品列表查询。

8.2 整合思路

springmvc+mybaits 的系统架构：



spring 将各层进行整合
通过 spring 管理持久层的 mapper(相当于 dao 接口)
通过 spring 管理业务层 service，service 中可以调用 mapper 接口。
spring 进行事务控制。

通过 spring 管理表现层 Handler，Handler 中可以调用 service 接口。

mapper、service、Handler 都是 javabean。

第一步：整合 dao 层

mybatis 和 spring 整合，通过 spring 管理 mapper 接口。

使用 mapper 的扫描器自动扫描 mapper 接口在 spring 中进行注册。

第二步：整合 service 层

通过 spring 管理 service 接口。

使用配置方式将 service 接口配置在 spring 配置文件中。

实现事务控制。

第三步：整合 springmvc

由于 springmvc 是 spring 的模块，不需要整合。

8.3 准备环境

数据库环境：mysql5.1

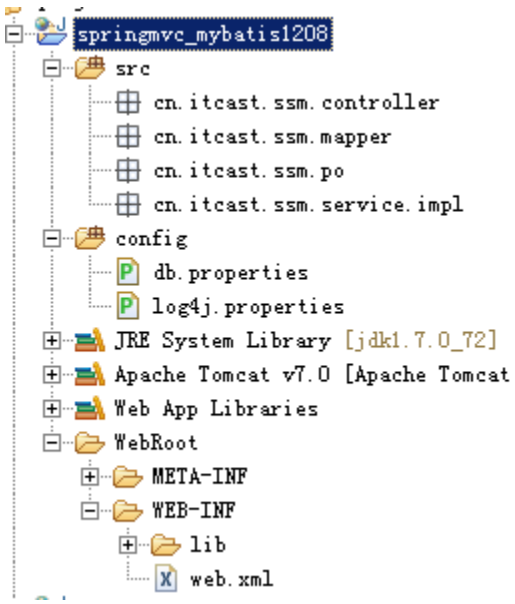
java 环境：
jdk1.7.0_72
eclipse indigo

springmvc 版本：spring3.2

所需要的 jar 包：
数据库驱动包：mysql5.1
mybatis 的 jar 包
mybatis 和 spring 整合包
log4j 包
dbcp 数据库连接池包
spring3.2 所有 jar 包
jstl 包

参考： 

工程结构：



8.4 整合dao

mybatis 和 spring 进行整合。

8.4.1 sqlMapConfig.xml

mybatis 自己的配置文件。

sqlMapConfig.xml

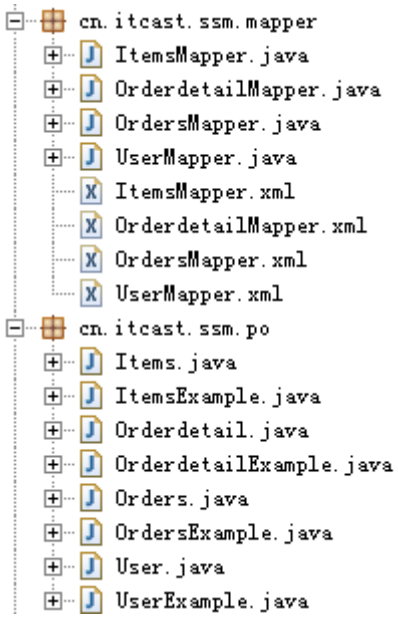
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <!-- 全局setting配置，根据需要添加 -->

    <!-- 配置别名 -->
    <typeAliases>
        <!-- 批量扫描别名 -->
        <package name="cn.itcast.ssm.po"/>
    </typeAliases>

    <!-- 配置mapper
    由于使用spring和mybatis的整合包进行mapper扫描，这里不需要配置了。
    必须遵循：mapper.xml和mapper.java文件同名且在一个目录
     -->

    <!-- <mappers>

    </mappers> -->
</configuration>
```

8.4.2 applicationContext-dao.xml

配置：
数据源
SqlSessionFactory
mapper 扫描器

```xml
<!-- 加载db.properties文件中的内容，db.properties文件中key命名要有一定的特殊规则 -->
<context:property-placeholder location="classpath:db.properties" />
<!-- 配置数据源，dbcp -->

<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="30" />
    <property name="maxIdle" value="5" />
</bean>
<!-- sqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!-- 数据库连接池 -->
    <property name="dataSource" ref="dataSource" />
    <!-- 加载mybatis的全局配置文件 -->
    <property name="configLocation" value="classpath:mybatis/sqlMapConfig.xml" />
</bean>
<!-- mapper扫描器 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <!-- 扫描包路径，如果需要扫描多个包，中间使用半角逗号隔开 -->
    <property name="basePackage" value="cn.itcast.ssm.mapper"></property>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
</bean>
```

8.4.3 逆向工程生成po类及mapper(单表增删改查)



将生成的文件拷贝至工程 中。

8.4.4 手动定义商品查询mapper

针对综合查询 mapper，一般情况会有关联查询，建议自定义 mapper

8.4.4.1 ItemsMapperCustom.xml

sql 语句：

SELECT * FROM items WHERE items.name LIKE '%笔记本%'

```
<mapper namespace="cn.itcast.ssm.mapper.ItemsMapperCustom" >

    <!-- 定义商品查询的sql片段，就是商品查询条件 -->
    <sql id="query_items_where">
     <!-- 使用动态sql，通过if判断，满足条件进行sql拼接 -->
     <!-- 商品查询条件通过ItemsQueryVo包装对象 中itemsCustom属性传递 -->
         <if test="itemsCustom!=null">
             <if test="itemsCustom.name!=null and itemsCustom.name!=''">
                 items.name LIKE '%${itemsCustom.name}%'
             </if>
         </if>

    </sql>

     <!-- 商品列表查询 -->
     <!-- parameterType传入包装对象(包装了查询条件)
         resultType建议使用扩展对象
      -->
     <select id="findItemsList" parameterType="cn.itcast.ssm.po.ItemsQueryVo"
          resultType="cn.itcast.ssm.po.ItemsCustom">
         SELECT items.* FROM items
         <where>
             <include refid="query_items_where"></include>
         </where>
     </select>

</mapper>
```

8.4.4.2 ItemsMapperCustom.java

```java
public interface ItemsMapperCustom {
    //商品查询列表
    public List<ItemsCustom> findItemsList(ItemsQueryVo itemsQueryVo)throws Exception;
}
```

8.5 整合service

让 spring 管理 service 接口。

8.5.1 定义service接口

```java
public interface ItemsService {

    //商品查询列表
    public List<ItemsCustom> findItemsList(ItemsQueryVo itemsQueryVo) throws Exception;

}
```

```java
public class ItemsServiceImpl implements ItemsService{

    @Autowired
    private ItemsMapperCustom itemsMapperCustom;

    @Override
    public List<ItemsCustom> findItemsList(ItemsQueryVo itemsQueryVo)
            throws Exception {
        //通过ItemsMapperCustom查询数据库
        return itemsMapperCustom.findItemsList(itemsQueryVo);
    }

}
```

8.5.2 在spring容器配置service(applicationContext-service.xml)

创建 applicationContext-service.xml，文件中配置 service。

```xml
<!-- 商品管理的service -->
<bean id="itemsService" class="cn.itcast.ssm.service.impl.ItemsServiceImpl"/>
```

8.5.3 事务控制(applicationContext-transaction.xml)

在 applicationContext-transaction.xml 中使用 spring 声明式事务控制方法。

```
<!-- 事务管理器
    对mybatis操作数据库事务控制，spring使用jdbc的事务控制类
-->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <!-- 数据源
    dataSource在applicationContext-dao.xml中配置了
     -->
    <property name="dataSource" ref="dataSource"/>
</bean>

<!-- 通知 -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <!-- 传播行为 -->
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="insert*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="select*" propagation="SUPPORTS" read-only="true"/>
    </tx:attributes>
</tx:advice>
<!-- aop -->
<aop:config>
    <aop:advisor advice-ref="txAdvice" pointcut="execution(* cn.itcast.ssm.service.impl.*.*(..))"/>
</aop:config>
```

8.6 整合springmvc

8.6.1 springmvc.xml

创建 springmvc.xml 文件，配置处理器映射器、适配器、视图解析器。

```
<!-- 可以扫描controller、service、...
    这里让扫描controller，指定controller的包
     -->
    <context:component-scan
base-package="cn.itcast.ssm.controller"></context:component-scan>


    <!--注解映射器 -->
    <!-- <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHan
dlerMapping"/> -->
    <!--注解适配器 -->
    <!-- <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHan
```

```
dlerAdapter"/> -->

    <!-- 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RequestMappingHandlerAdapter
    实际开发时使用mvc:annotation-driven
     -->
    <mvc:annotation-driven></mvc:annotation-driven>


    <!-- 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>
```

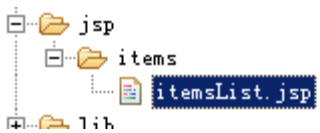
8.6.2 配置前端控制器

参考入门程序。

8.6.3 编写Controller(就是Handler)

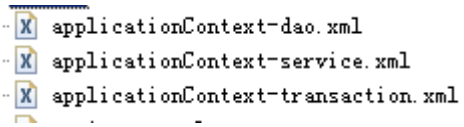
```java
@Controller
public class ItemsController {

    @Autowired
    private ItemsService itemsService;

    // 商品查询
    @RequestMapping("/queryItems")
    public ModelAndView queryItems() throws Exception {

        // 调用service查找 数据库，查询商品列表
        List<ItemsCustom> itemsList = itemsService.findItemsList(null);

        // 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;

    }
```

8.6.4 编写jsp



8.7 加载spring容器

将 mapper、service、controller 加载到 spring 容器中。



建议使用通配符加载上边的配置文件。

在 web.xml 中，添加 spring 容器监听器，加载 spring 容器。

```
<!-- 加载spring容器 -->
  <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/classes/spring/applicationContext-*.xml</param-value>
  </context-param>
  <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
```

9 商品修改功能开发

9.1 需求

操作流程：

1、进入商品查询列表页面

2、点击修改，进入商品修改页面，页面中显示了要修改的商品（从数据库查询）

要修改的商品从数据库查询，根据商品 id(主键)查询商品信息

3、在商品修改页面，修改商品信息，修改后，点击提交

9.2 开发mapper

mapper：

根据 id 查询商品信息

根据 id 更新 Items 表的数据

不用开发了，使用逆向工程生成的代码。

9.3 开发service

接口功能：

根据 id 查询商品信息

修改商品信息

```java
//根据id查询商品信息
/**
 *
 * <p>Title: findItemsById</p>
 * <p>Description: </p>
 * @param id 查询商品的id
 * @return
 * @throws Exception
 */
public ItemsCustom findItemsById(Integer id) throws Exception;

//修改商品信息
/**
 *
 * <p>Title: updateItems</p>
 * <p>Description: </p>
 * @param id 修改商品的id
 * @param itemsCustom 修改的商品信息
 * @throws Exception
 */
public void updateItems(Integer id,ItemsCustom itemsCustom) throws Exception;
```

9.4 开发controller

方法：

商品信息修改页面显示

商品信息修改提交

10 @RequestMapping

- url 映射

定义 controller 方法对应的 url，进行处理器映射使用。

- 窄化请求映射

```java
@Controller
//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 {
```

- 限制 http 请求方法

出于安全性考虑，对 http 的链接进行方法限制。
如果限制请求为 post 方法，进行 get 请求，报错：


```
//限制http请求方法，可以post和get
@RequestMapping(value="/editItems",method={RequestMethod.POST,RequestMethod.GET})
```

11 controller方法的返回值

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

■ 返回 string
如果 controller 方法返回 string，

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";
}
```

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";
```

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";
```

■ 返回 void

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")

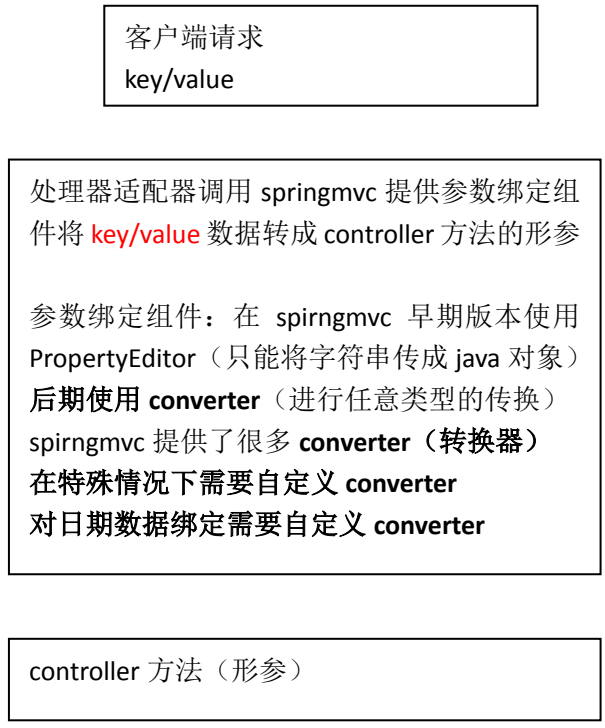
3、也可以通过 response 指定响应结果，例如响应 json 数据如下：
```
response.setCharacterEncoding("utf-8");
response.setContentType("application/json;charset=utf-8");
response.getWriter().write("json 串");
```

12 参数绑定

12.1spring参数绑定过程

从客户端请求 key/value 数据，经过参数绑定，将 key/value 数据绑定到 controller 方法的形参上。

springmvc 中，接收页面提交的数据是通过方法形参来接收。而不是在 controller 类定义成员变更接收!!!!



12.2默认支持的类型

直接在 controller 方法形参上定义下边类型的对象，就可以使用这些对象。在参数绑定过程中，如果遇到下边类型直接进行绑定。

1.2.1.1HttpServletRequest

通过 request 对象获取请求信息

1.2.1.2HttpServletResponse

通过 response 处理响应信息

1.2.1.3HttpSession

通过 session 对象得到 session 中存放的对象

1.2.1.4Model/ModelMap

model 是一个接口，modelMap 是一个接口实现 。
作用：将 model 数据填充到 request 域。

12.3简单类型

通过@RequestParam 对简单类型的参数进行绑定。
如果不使用@RequestParam，要求 request 传入参数名称和 controller 方法的形参名称一致，方可绑定成功。

如果使用@RequestParam，不用限制 request 传入参数名称和 controller 方法的形参名称一致。

通过 required 属性指定参数是否必须要传入，如果设置为 true，没有传入参数，报下边错误：

HTTP Status 400 - Required Integer parameter 'id' is not present

---

```
//@RequestParam里边指定request传入参数名称和形参进行绑定。
//通过required属性指定参数是否必须要传入
//通过defaultValue可以设置默认值，如果id参数没有传入，将默认值和形参绑定。
public String editItems(Model model,@RequestParam(value="id",required=true) Integer items_id)throws Exception
```

参考教案 对其它简单类型绑定进行测试。

12.4pojo绑定

页面中 input 的 name 和 controller 的 pojo 形参中的属性名称一致，将页面中数据绑定到 pojo。

页面定义：

```
<table width="100%" border=1>
<tr>
    <td>商品名称</td>
    <td><input type="text" name="name" value="${itemsCustom.name }"/></td>
</tr>
<tr>
    <td>商品价格</td>
    <td><input type="text" name="price" value="${itemsCustom.price }"/></td>
</tr>
```

controller 的 pojo 形参的定义：

```
public class Items {
    private Integer id;

    private String name;

    private Float price;

    private String pic;
```

12.5自定义参数绑定实现日期类型绑定

对于 controller 形参中 pojo 对象，如果属性中有日期类型，需要自定义参数绑定。
将请求日期数据串传成 日期类型，要转换的日期类型和 pojo 中日期属性的类型保持一致。

```
public class Items {
    private Integer id;

    private String name;

    private Float price;

    private String pic;

    private Date createtime;
    java.util.Date
    private
```

所以自定义参数绑定将日期串转成 java.util.Date 类型。

需要向处理器适配器中注入自定义的参数绑定组件。

12.5.1自定义日期类型绑定

```
public class CustomDateConverter implements Converter<String,Date>{

    @Override
    public Date convert(String source) {

        //实现 将日期串转成日期类型(格式是yyyy-MM-dd HH:mm:ss)

        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        try {
            //转成直接返回
            return simpleDateFormat.parse(source);
        } catch (ParseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //如果参数绑定失败返回null
        return null;
    }

}
```

12.5.2配置方式

```
<mvc:annotation-driven conversion-service="conversionService"></mvc:annotation-driven>


<!-- 自定义参数绑定 -->
<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <!-- 转换器 -->
    <property name="converters">
        <list>
            <!-- 日期类型转换 -->
            <bean class="cn.itcast.ssm.controller.converter.CustomDateConverter"/>
        </list>
    </property>

</bean>
```

13 springmvc和struts2 的区别

1、springmvc 基于方法开发的，struts2 基于类开发的。

springmvc 将 url 和 controller 方法映射。映射成功后 springmvc 生成一个 Handler 对象，对象中只包括了一个 method。
方法执行结束，形参数据销毁。
springmvc 的 controller 开发类似 service 开发。

2、springmvc 可以进行单例开发，并且建议使用单例开发，struts2 通过类的成员变量接收参数，无法使用单例，只能使用多例。

3、经过实际测试，struts2 速度慢，在于使用 struts 标签，如果使用 struts 建议使用 jstl。

14 问题

14.1post乱码

在 web.xml 添加 post 乱码 filter

在 web.xml 中加入：

```
<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>
```

以上可以解决 post 请求乱码问题。
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