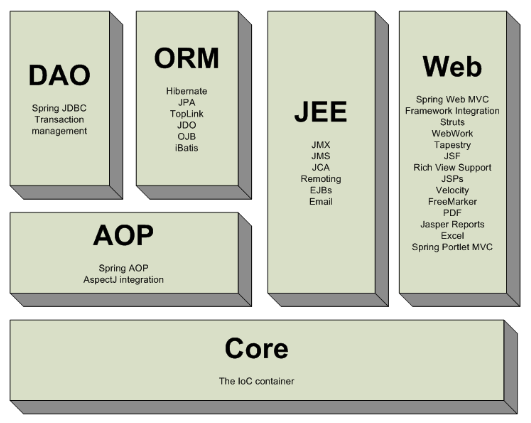
**SpringMVC第一天**

**框架课程**

# Spring入门

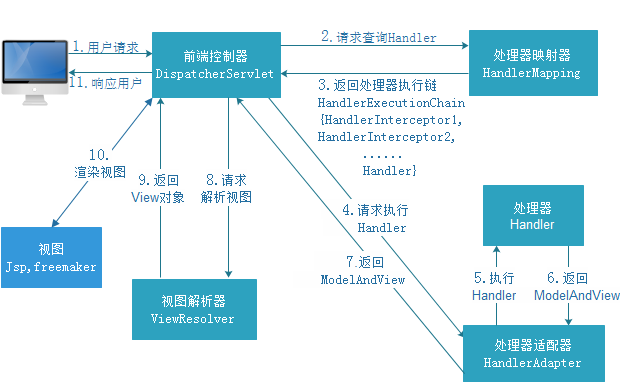
## Springmvc是什么

Spring web mvc和Struts2都属于表现层的框架,**它是Spring框架的一部分**,我们可以从Spring的整体结构中看得出来,如下图：



# Springmvc架构

## 框架结构



## 架构流程

1. 用户发送请求至前端控制器DispatcherServlet
2. DispatcherServlet收到请求调用HandlerMapping处理器映射器。
3. 处理器映射器**根据请求url找到具体的处理器**，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。
4. DispatcherServlet通过HandlerAdapter处理器适配器调用处理器
5. **执行处理器(Handler，也叫后端控制器)。**
6. **Controller执行完成返回ModelAndView**
7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet
8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器
9. ViewReslover解析后返回具体View
10. DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。
11. DispatcherServlet响应用户

## 组件说明

|  |
| --- |
| 说明：在springmvc的各个组件中**，处理器映射器、处理器适配器、视图解析器称为springmvc的三大组件。** |

以下组件通常使用框架提供实现：

* DispatcherServlet：前端控制器

用户请求到达前端控制器，dispatcherServlet是整个流程控制的中心，由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

* HandlerMapping：处理器映射器

HandlerMapping负责根据用户请求url找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，**注解方式**等。

* Handler：处理器

Handler 是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。

由于Handler涉及到具体的用户业务请求，所以一般情况需要程序员根据业务需求开发Handler。

* HandlAdapter：处理器适配器

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

下图是许多不同的适配器，最终都可以使用usb接口连接



* ViewResolver：视图解析器

View Resolver负责将处理结果生成View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。

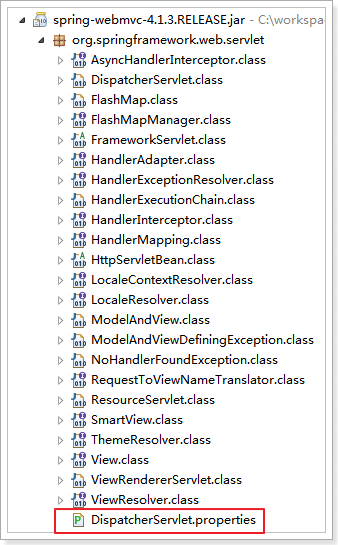
* View：视图

springmvc框架提供了很多的View视图类型的支持，包括：jstlView、freemarkerView、pdfView等。我们最常用的视图就是jsp。

## 默认加载的组件

我们没有做任何配置，就可以使用这些组件

因为框架已经默认加载这些组件了，配置文件位置如下图：



## 注解映射器和适配器

### 配置处理器映射器

从spring3.1版本开始，**废除了****DefaultAnnotationHandlerMapping的使用，推荐使用****RequestMappingHandlerMapping完成注解式处理器映射。**

注解式处理器映射器，对类中标记了@ResquestMapping的方法进行映射。根据@ResquestMapping定义的url匹配@ResquestMapping标记的方法，匹配成功返回**HandlerMethod对象**给前端控制器。

**HandlerMethod对象中封装url对应的方法Method**。

在springmvc.xml配置文件中配置如下：

<!-- 配置处理器映射器 -->

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*/>

### 配置处理器适配器

从spring3.1版本开始，废除了AnnotationMethodHandlerAdapter的使用，推荐使用**RequestMappingHandlerAdapter**完成注解式处理器适配。

**注解式处理器适配器，对标记@ResquestMapping的方法进行适配。**

在springmvc.xml配置文件中配置如下：

<!-- 配置处理器适配器 -->

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*/>

### 注解驱动

直接配置处理器映射器和处理器适配器比较麻烦，可以使用注解驱动来加载。

SpringMVC使用<mvc:annotation-driven>自动加载RequestMappingHandlerMapping和RequestMappingHandlerAdapter

可以在springmvc.xml配置文件中使用<mvc:annotation-driven>替代注解处理器和适配器的配置。

<!-- 注解驱动 -->

<mvc:annotation-driven />

## 视图解析器

视图解析器使用SpringMVC框架默认的InternalResourceViewResolver，这个视图解析器支持JSP视图解析。

在springmvc.xml配置文件中配置如下：

<!-- 配置视图解析器 -->

<beanclass=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 配置逻辑视图的前缀 -->

<propertyname=*"prefix"*value=*"/jsp/"*/>

<!-- 配置逻辑视图的后缀 -->

<propertyname=*"suffix"*value=*".jsp"*/>

</bean>

**我们配置视图解析器的目的：就是节约点代码**

比如逻辑视图名为ItemList，则最终返回的jsp视图地址:

“jsp/itemList.jsp”

# 整合mybatis

## 整合思路

Dao层：

1. SqlMapConfig.xml，**可以保留别名配置**。
2. applicationContext-dao.xml
   1. 数据库连接池
   2. SqlSessionFactory对象，需要spring和mybatis整合包下的。
   3. 配置mapper文件扫描器。

Service层：

1. applicationContext-service.xml包扫描器，扫描@service注解的类。
2. applicationContext-trans.xml配置事务。

Controller层：

1. Springmvc.xml
   1. 包扫描器，扫描@Controller注解的类。
   2. 配置注解驱动
   3. 配置视图解析器

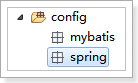
Web.xml文件：

1. 配置spring
2. 配置前端控制器。

## 加入配置文件

创建资源文件夹config

在其下创建mybatis和spring文件夹，用来存放配置文件，如下图：



### sqlMapConfig.xml

**使用逆向工程来生成Mapper相关代码，不需要配置别名（逆向工程生成的自带全路径类名）。**

### applicationContext-dao.xml

**配置数据源、配置SqlSessionFactory、mapper扫描器。**

### applicationContext-service.xml

**配置mapper扫描器**

<!--扫描注解（这里主要是service层） -->

<context:component-scanbase-package=*"cn.itcast.ssm.service"*/>

</beans>

### applicationContext-trans.xml

<!-- 事务管理器 -->

<beanid=*"transactionManager"* class=*"org.springframework.jdbc.datasource.****DataSourceTransactionManager****"*>

<!-- 数据源 -->

<propertyname=*"dataSource"*ref=*"dataSource"*/>

</bean>

<!-- 通知 -->

<tx:adviceid=*"txAdvice"*transaction-manager=*"transactionManager"*>

<tx:attributes>

<!-- 传播行为 -->

<tx:methodname=*"save\*"*propagation=*"REQUIRED"*/>

<tx:methodname=*"insert\*"*propagation=*"REQUIRED"*/>

<tx:methodname=*"delete\*"*propagation=*"REQUIRED"*/>

<tx:methodname=*"update\*"*propagation=*"REQUIRED"*/>

<tx:methodname=*"find\*"*propagation=*"SUPPORTS"*read-only=*"true"*/>

<tx:methodname=*"get\*"*propagation=*"SUPPORTS"*read-only=*"true"*/>

<tx:methodname=*"query\*"*propagation=*"SUPPORTS"*read-only=*"true"*/>

</tx:attributes>

</tx:advice>

<!-- 切面 -->

<aop:config>

<aop:advisoradvice-ref=*"txAdvice"*

pointcut=*"execution(\* cn.itcast.ssm.service.\*.\*(..))"* />

</aop:config>

</beans>

### springmvc.xml

<!-- 扫描controller层注解 -->

<context:component-scanbase-package=*"cn.itcast.ssm.controller"*/>

<!-- 注解驱动 -->

<mvc:annotation-driven/>

<!-- 配置视图解析器 -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 配置逻辑视图的前缀 -->

<propertyname=*"prefix"*value=*"/WEB-INF/jsp/"*/>

<!-- 配置逻辑视图的后缀 -->

<propertyname=*"suffix"*value=*".jsp"*/>

</bean>

</beans>

### web.xml

<!-- 配置spring -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext\*.xml</param-value>

</context-param>

<!-- 使用监听器加载Spring配置文件 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 配置SrpingMVC的前端控制器 -->

<servlet>

<servlet-name>springmvc-web</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/springmvc.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springmvc-web</servlet-name>

<!-- 配置所有以action结尾的请求进入SpringMVC -->

<url-pattern>\*.action</url-pattern>

</servlet-mapping>

</web-app>

# 参数绑定

## 默认支持的参数类型

|  |
| --- |
| 1. HttpServletRequest 2. HttpServletResponse 3. HttpSession 4. Model/ModelMap  Model 除了ModelAndView以外，还可以**使用Model来向页面传递数据**，  Model是一个接口，在参数里直接声明model即可。  如果使用Model则可以不使用ModelAndView对象，**Model对象可以向页面传递数据，View对象则可以使用String返回值替代**。  不管是Model还是ModelAndView，其**本质都是使用httpServletRequest对象向jsp传递数据**。  代码实现：  /\*\*  \* 根据id查询商品,使用Model  \*/  @RequestMapping("/itemEdit")  **public**StringqueryItemById(Integer id, Model model) {  // 根据id查询商品数据  Item item = itemService.queryItemById(id);  // 把商品数据放在模型中  model.addAttribute("item", item);  **return**"itemEdit";  } 4.2 ModelMap **ModelMap是Model接口的实现类**，也可以通过ModelMap向页面传递数据  使用Model和ModelMap的效果一样，如果直接使用Model，springmvc会实例化ModelMap。 |

## 绑定简单类型

当请求的参数名称和处理器形参**名称一致**时会将请求参数与形参进行绑定。

### 支持的数据类型

**参数类型推荐使用包装数据类型**，因为基础数据类型不可以为null

整形：**Integer**、int

字符串：String

单精度：Float、float

双精度：Double、double

布尔型：Boolean、boolean

|  |
| --- |
| 栗子：  请求url：  http://localhost:8080/xxx.action?**id=2&status=false**  处理器方法：  public String editItem(Model model,**Integer id,Boolean status**) |

### @RequestParam

使用@RequestParam常用于处理简单类型的绑定。

value：参数名字，即入参的请求参数名字，如value=“itemId”表示请求的参数 区中的名字为itemId的参数的值将传入

required：是否必须，默认是true(不能为null)，表示请求中一定要有相应的参数，否则将报错：

TTP Status 400 - Required Integer parameter 'XXXX' is not present

defaultValue：默认值，表示如果请求中没有同名参数时的默认值

定义如下：

@RequestMapping("/itemEdit")

**public** String queryItemById(@RequestParam(value = "itemId", required = **false**, defaultValue = "1") Integer id,

ModelMap modelMap) {

。。。。**表示页面的name为“itemId”，接收时用id接收。（适用于接收参数不一致时）**

}

## 绑定pojo类型

**要求：pojo对象中的属性名和表单中input的name属性一致。**

请求的参数名称和pojo的属性名称一致，会自动将请求参数赋值给pojo的属性。（和beanutils一个道理）

注意：

**提交的表单中不要有日期类型的数据，否则会报400错误。如果想提交日期类型的数据需要用到后面的自定义参数绑定的内容。**

### 解决post乱码问题

在web.xml中加入：

<!-- 解决post乱码问题 -->

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.**CharacterEncodingFilter**</filter-class>

<!—**指定转换后的编码格式是：UTF8** -->

**<init-param>**

**<param-name>encoding</param-name>**

**<param-value>UTF-8</param-value>**

**</init-param>**

</filter>

<filter-mapping>

<filter-name>encoding</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：

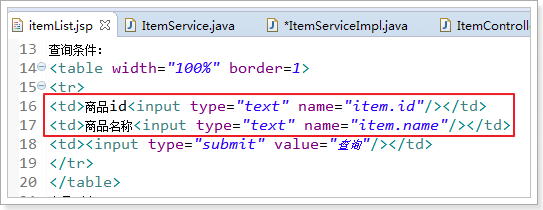
String userName new

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

## 绑定pojo中的pojo

页面定义如下图：**（重点在这：name=）**



### 接收查询条件

// 绑定包装数据类型

@RequestMapping("/queryItem")

**public** String queryItem(QueryVo queryVo) {

System.***out***.println(queryVo.getItem().getId());

System.***out***.println(queryVo.getItem().getName());

**return**"success";

}

## 自定义参数绑定

由于日期数据有很多种格式，springmvc没办法把字符串转换成日期类型。所以需要自定义参数绑定。

前端控制器接收到请求后，找到注解形式的处理器适配器，对RequestMapping标记的方法进行适配，并对方法中的形参进行参数绑定。可以**在springmvc处理器适配器上自定义转换器Converter进行参数绑定**。

一般使用<mvc:annotation-driven/>注解驱动加载处理器适配器，可以在此标签上进行配置。

### 自定义Converter

//Converter<S, T>

//S:source,需要转换的源的类型

//T:target,转换后的类型

**Public class** DateConverter **implements** Converter<String, Date> {

@Override

**public** Date convert(String source) {

**try** {

// 把字符串转换为日期类型

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

Date date = simpleDateFormat.parse(source);

**return**date;

} **catch** (ParseException e) {

e.printStackTrace();

}

// 如果转换异常则返回空

**Returnnull**;

}

}

### 配置Converter

我们同时可以配置多个的转换器。

类似下图的usb设备，可以接入多个usb设备



<!--配置注解驱动-->

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

<!--转换器配置 -->

<bean id=*"conversionService"* class=*"org.springframework.format.support.****FormattingConversionServiceFactoryBean****"*>

<property name=*"****converters****"*>

<set>（用set、list、array等等都可以）

<bean class=*"cn.itcast.springmvc.converter.DateConverter"*/>

……….可以配很多自定义的转换器

</set>

</property>

</bean>

### 配置方式2（了解）

<!--注解适配器 -->

<beanclass=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>

<propertyname=*"webBindingInitializer"*ref=*"customBinder"*></property>

</bean>

<!--自定义webBinder -->

<beanid=*"customBinder"*class=*"org.springframework.web.bind.support.ConfigurableWebBindingInitializer"*>

<propertyname=*"conversionService"*ref=*"conversionService"*/>

</bean>

<!--转换器配置 -->

<beanid=*"conversionService"*class=*"org.springframework.format.support.FormattingConversionServiceFactoryBean"*>

<propertyname=*"converters"*>

<set>

<beanclass=*"cn.itcast.springmvc.convert.DateConverter"*/>

</set>

</property>

</bean>

注意：此方法需要独立配置处理器映射器、适配器，

不再使用<mvc:annotation-driven/>

# springmvc与struts2不同

1. springmvc的入口是一个servlet即前端控制器，而struts2入口是一个filter过滤器。
2. springmvc是**基于方法开发**(**一个url对应一个方法**)（多线程，用的方法的副本），请求参数传递到方法的形参。**struts2是基于类开发（多例的，一个url对应一个action类）**，传递参数是通过类的属性，只能设计为多例。
3. Struts2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。