

前言

认识SpringMVC

- 一、什么是SpringMVC?
- 二、为什么要使用SpringMVC?
- 三、回顾Struts2开发
- 四、Struts2的工作流程
- 五、SpringMVC快速入门
 - 5.1 导入开发包
 - 5.2 编写Action
 - 5.3 注册核心控制器
 - 5.4 创建SpringMVC控制器
 - 5.5 访问
- 六、SpringMVC工作流程
 - 6.1 映射器
 - 6.2 适配器
 - 6.3 视图解析器
- 七、AbstractCommandController
 - 7.1 实体
 - 7.2 提交参数的JSP
 - 7.3 配置Action处理请求
 - 7.4 Action接收参数
 - 7.5 测试效果:
- 八、小总结

参数绑定、数据回显、文件上传

- 一、参数绑定
 - 1.1 默认支持的参数类型
 - 1.2 参数的绑定过程
 - 1.3 自定义绑定参数【老方式、全部Action均可使用】
 - 1.4 配置转换器
 - 1.5 自定义参数转换器【新方式、推崇方式】
 - 1.6 配置转换器
 - 1.7 @RequestParam注解
 - 1.8 Controller方法返回值
- 二、数据回显
 - 2.1 @ModelAttribute 注解
- 三、SpringMVC文件上传
 - 3.1 配置虚拟目录
 - 3.2 快速入门
- 四、总结

拦截器、统一处理异常、RESTful、拦截器

- 一、Validation
 - 1.1 快速入门
 - 1.2 分组校验
- 二、统一异常处理
 - 2.1 定义统一异常处理器类
 - 2.2 配置统一异常处理器
- 三、RESTful支持

- 3.1url的RESTful实现
- 3.2更改DispatcherServlet的配置
- 四、SpringMVC拦截器
 - 4.1测试执行顺序
 - 4.2拦截器应用-身份认证
- 五、总结

前言

这个文档的内容**纯手打**，如果想要看更多的干货文章，关注我的公众号：**Java3y**。有更多的原创技术文章和干货！

目前疯狂处于**疯狂更新PDF**中，只要是Java后端的知识，都会有！**欢迎来我公众号催更**！微信搜索：**Java3y**

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！公众号有我的**联系方式**



- 🔥Java精美脑图
- 🔥Java学习路线
- 🔥开发常用工具
- 🔥精美原创电子书

在公众号下回复「**888**」即可获得！！

学习不能盲目，跟着我，会让你事半功倍

文档允许随意传播，但不能修改任何内容。

下面的文章都有对应的原创精美PDF，在持续更新中，可以来找我催更~

- [92页的Mybatis](#)
- [129页的多线程](#)
- [141页的Servlet](#)
- [158页的JSP](#)
- [76页的集合](#)
- [64页的JDBC](#)
- [105页的数据结构和算法](#)
- [142页的Spring](#)
- [58页的过滤器和监听器](#)
- [30页的HTTP](#)
- Hibernate
- AJAX
- Redis
-

电子书的整理也是挺不容易，如果你觉得有帮助，想要打赏作者，那么可以通过这个收款码打赏我，金额不重要，心意最重要。主要是我可以通过这个打赏情况来预计大家对这本电子书的评价，嘻嘻

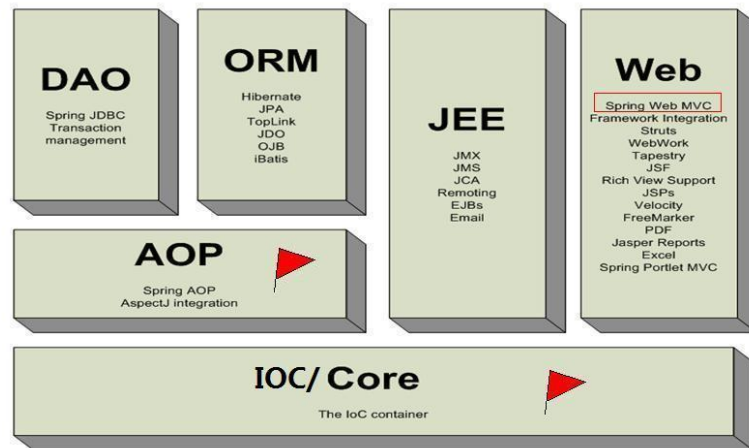


认识SpringMVC

一、什么是SpringMVC?

SpringMVC是Spring家族的一员，Spring是将现在开发中流行的组件进行组合而成的一个框架！它用在基于MVC的表现层开发，类似于struts2框架

声明：spring web mvc 等价于 springmvc



http://blog.csdn.net/hon_3y

二、为什么要使用SpringMVC?

我们在之前已经学过了Struts2这么一个基于MVC的框架....那么我们已经学会了Struts2，为啥要学习SpringMVC呢???

下面我们来看一下Struts2不足之处：

- 有漏洞【详细可以去搜索】
- 运行速度较慢【比SpringMVC要慢】
- 配置的内容较多【需要使用Struts.xml文件】
- 比较重量级

基于这么一些原因，并且业内现在SpringMVC已经逐渐把Struts2给替代了...因此我们学习SpringMVC一方面能够让我们跟上业界的潮流框架，一方面SpringMVC确实是非常好用！

可以这么说，Struts2能做的东西，SpringMVC也能够做....

三、回顾Struts2开发

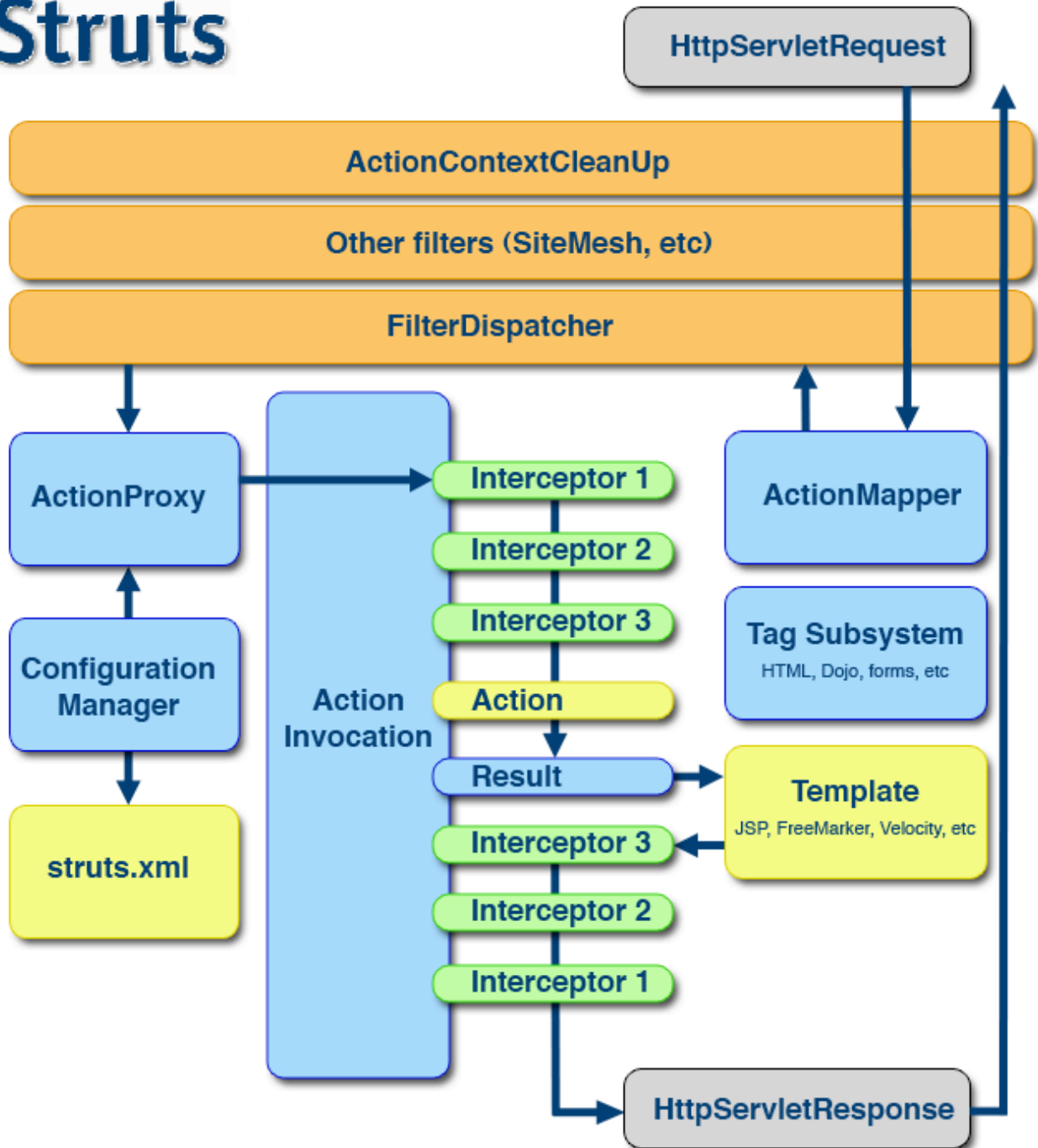
如果没接触过Struts2的，这里可以跳过。Struts2可以不学

在Struts2中，我们的开发特点是这样的：

- Action类继承着ActionSupport类【如果要使用Struts2提供的额外功能，就要继承它】
- Action业务方法总是返回一个字符串，再由Struts2内部通过我们手写的Struts.xml配置文件去跳转到对应的view
- Action类是多例的，接收Web传递过来的参数需要使用实例变量来记住，通常我们都会写上set和get方法

四、Struts2的工作流程

Struts



Key:

Servlet Filters Struts Core Interceptors User created

- Struts2接收到request请求
- 将请求转向我们的过滤分批器进行过滤
- 读取Struts2对应的配置文件
- 经过默认的拦截器之后创建对应的Action **【多例】**
- 执行完业务方法就返回给response对象

五、SpringMVC快速入门

5.1导入开发包

如果用Maven的，那导入Maven依赖即可

前6个是Spring的核心功能包【IOC】，第7个是关于web的包，第8个是SpringMVC包

- org.springframework.context-3.0.5.RELEASE.jar
- org.springframework.expression-3.0.5.RELEASE.jar
- org.springframework.core-3.0.5.RELEASE.jar
- org.springframework.beans-3.0.5.RELEASE.jar
- org.springframework.asm-3.0.5.RELEASE.jar
- commons-logging.jar
- org.springframework.web-3.0.5.RELEASE.jar
- org.springframework.web.servlet-3.0.5.RELEASE.jar

5.2编写Action

Action实现Controller接口

```
public class HelloAction implements Controller {
    @Override
    public ModelAndView handleRequest(javax.servlet.http.HttpServletRequest
    httpServletRequest, javax.servlet.http.HttpServletResponse
    httpServletResponse) throws Exception {
        return null;
    }
}
```

我们只要实现handleRequest方法即可，该方法已经说了request和response对象给我们用了。这是我们非常熟悉的request和response对象。然而该方法返回的是ModelAndView这么一个对象，这是和Struts2不同的。Struts2返回的是字符串，而SpringMVC返回的是ModelAndView

ModelAndView其实他就是将我们的视图路径和数据封装起来而已【我们想要跳转到哪，把什么数据存到request域中，设置这个对象的属性就行了】。

```
public class HelloAction implements Controller {
    @Override
    public ModelAndView handleRequest(javax.servlet.http.HttpServletRequest
    httpServletRequest, javax.servlet.http.HttpServletResponse
    httpServletResponse) throws Exception {

        ModelAndView modelAndView = new ModelAndView();

        //跳转到hello.jsp页面。
        modelAndView.setViewName("/hello.jsp");
        return modelAndView;
    }
}
```

5.3注册核心控制器

在Struts2中，我们想要使用Struts2的功能，那么就得在web.xml文件中配置过滤器。而我们使用SpringMVC的话，我们是在web.xml中配置核心控制器

```
<!-- 注册springmvc框架核心控制器 -->
<servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <!--到类目录下寻找我们的配置文件-->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:hello.xml</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <!--映射的路径为.action-->
    <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

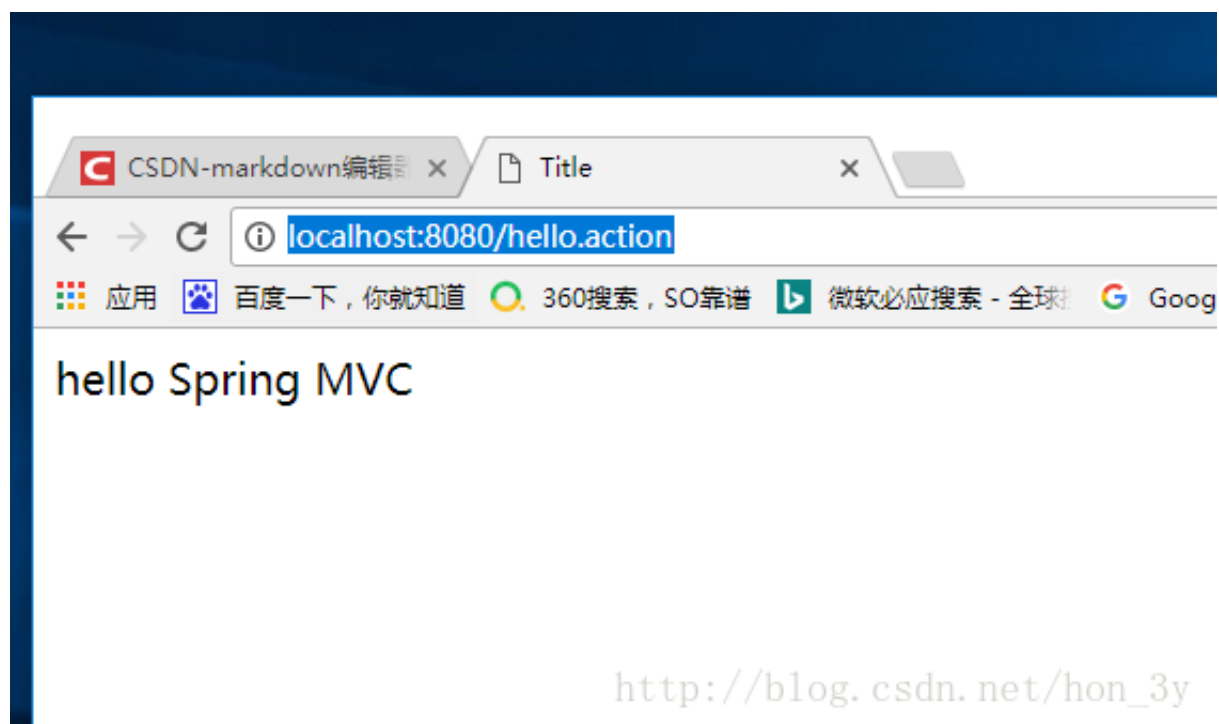
5.4创建SpringMVC控制器

我们在hello.xml配置文件中把SpringMVC的控制器创建出来

```
<!--
    注册控制器
    name属性的值表示的是请求的路径【也就是说，当用户请求到/helloAction时，就交由
HelloAction类进行处理】
-->
<bean class="HelloAction" name="/hello.action"></bean>
```

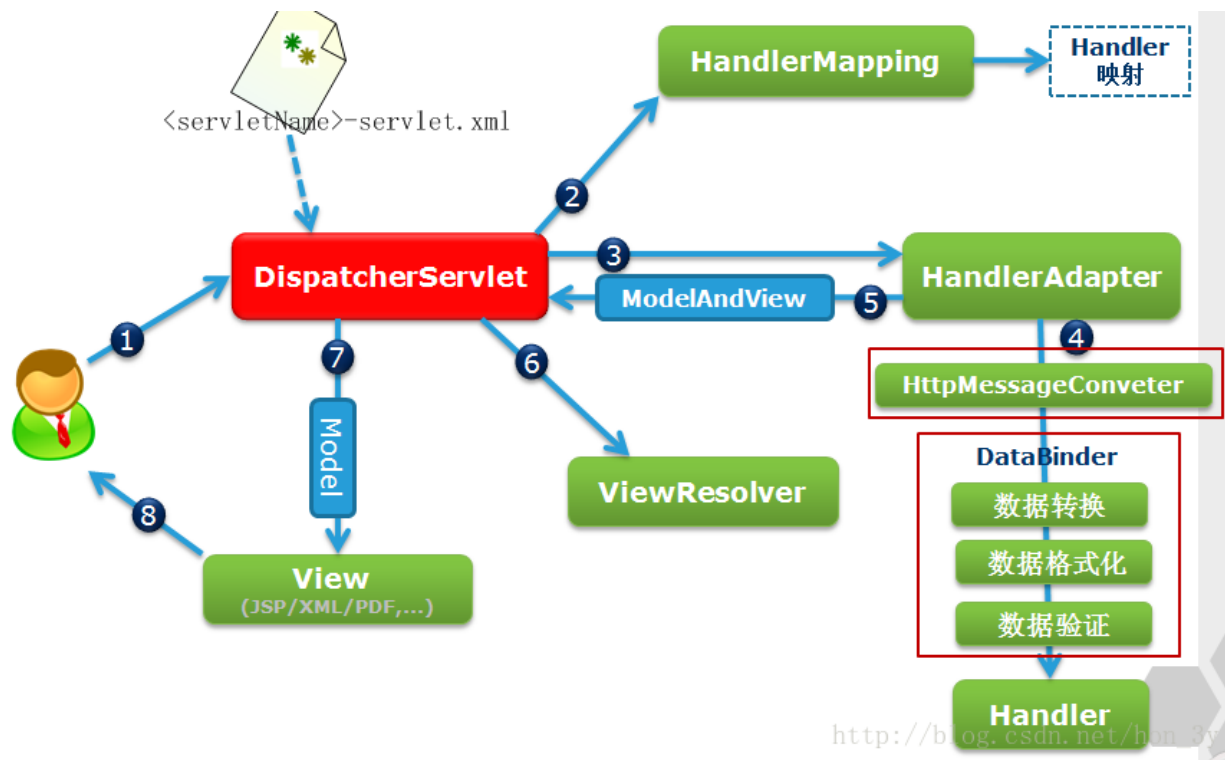
5.5访问

当我们在浏览器访问 `http://localhost:8080/hello.action` 的时候，Spring会读取到我们的访问路径，然后对比一下我们的配置文件中是否有配置 `/hello.action`，如果有。那么就交由对应的Action类来进行处理。Action类的业务方法将其请求输出到hello.jsp页面上。



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

六、SpringMVC工作流程



- 用户发送请求
- 请求交由核心控制器处理
- 核心控制器找到映射器，映射器看看请求路径是什么
- 核心控制器再找到适配器，看看有哪些类实现了Controller接口或者对应的bean对象
- 将带过来的数据进行转换，格式化等等操作
- 找到我们的控制器Action，处理完业务之后返回一个ModelAndView对象
- 最后通过视图解析器来对ModelAndView进行解析
- 跳转到对应的JSP/html页面

上面的工作流程中，我们是没有讲过映射器，适配器，视图解析器这样的东西的。但是SpringMVC的环境还是被我们搭建起来了。

下面就由我来一个一个来介绍他们是有什么用的！

6.1 映射器

我们在web.xml中配置规定只要是.action为后缀的请求都是会经过SpringMVC的核心Servlet。

当我们接收到请求的时候，我们发现是hello.action，是会经过我们的核心Servlet的，那么核心Servlet就会去找有没有专门的Action类来处理hello.action请求的。

也就是说：映射器就是用于处理“什么样的请求提交给Action”处理。【默认可省略的】.....

其实我们在快速入门的例子已经配置了：**name**属性就是规定了**hello.action**到**HelloAction**控制器中处理！

```

<!--
    注册控制器
    name属性的值表示的是请求的路径【也就是说，当用户请求到/helloAction时，就交由
    HelloAction类进行处理】
-->
<bean class="HelloAction" name="/hello.action"></bean>

```

映射器默认的值是这样的：

```

<!-- 注册映射器(handler包)(框架)【可省略】 -->
<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
</bean>

```

当然了，上面我们在创建控制器的时候【也就是HelloAction】可以不使用name属性来指定路径，可以使用我们的映射器来配置。如以下的代码：

```

<bean class="HelloAction" id="helloAction"></bean>

<!-- 注册映射器(handler包)(框架) -->
<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/hello.action">helloAction</prop>
        </props>
    </property>
</bean>

```

当我们需要多个请求路径都交由helloAction控制器来处理的话，我们只要添加prop标签就行了！

```

<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/hello.action">helloAction</prop>
            <prop key="/bye.action">helloAction</prop>
        </props>
    </property>
</bean>

```



6.2 适配器

当我们映射器找到对应的Action来处理请求的时候，核心控制器会让适配器去找该类是否实现了**Controller**接口。【默认可省略的】

也就是说：适配器就是去找实现了**Controller**接口的类

```
<!-- 适配器【可省略】 -->
<bean
class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter">
</bean>
```

6.3 视图解析器

我们把结果封装到ModelAndView以后，SpringMVC会使用视图解析器来对ModelAndView进行解析。【默认可省略的】

也有一种情况是不能省略的。我们在快速入门的例子中，将结果封装到**ModelAndView**中，用的是**绝对真实路径**！如果我们用的是逻辑路径，那么就必须对其配置，否则SpringMVC是找不到对应的路径的。

那什么是逻辑路径呢？？我们在**Struts2**中，返回的是**"success"**这样的字符串，从而跳转到**success.jsp**这样的页面上。我们就可以把**"success"**称作为逻辑路径。

在Action中返回hello，**hello**是一个逻辑路径。需要我们使用视图解析器把逻辑路基补全

```

public ModelAndView handleRequest(javax.servlet.http.HttpServletRequest
httpServletRequest, javax.servlet.http.HttpServletResponse
httpServletResponse) throws Exception {

    ModelAndView modelAndView = new ModelAndView();

    //跳转到hello.jsp页面。
    modelAndView.setViewName("hello");
    return modelAndView;
}

```

如果不使用视图解析器的话，那么就会找不到页面：



因此，我们需要配置视图解析器

```

<!--
如果Action中书写的是视图逻辑名称，那么视图解析器就必须配置
如果Action中书写的是视图真实名称，那么视图解析器就可选配置
-->
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- 路径前缀 -->
    <property name="prefix" value="/" />
    <!-- 路径后缀 -->
    <property name="suffix" value=".jsp" />
    <!-- 前缀+视图逻辑名+后缀=真实路径 -->
</bean>

```

七、AbstractCommandController

到目前为止，我们都没有将SpringMVC是怎么接收web端传递过来的参数的。

我们在Struts2中，只要在Action类上写对应的成员变量，给出对应的set和get方法。那么Struts2就会帮我们把参数封装到对应的成员变量中，是非常方便的。

那么我们在SpringMVC中是怎么获取参数的呢？？？？我们是**将Action继承AbstractCommandController**这么一个类的。

```
public class HelloAction extends AbstractCommandController {

    @Override
    protected ModelAndView handle(HttpServletRequest httpServletRequest,
        HttpServletResponse httpServletResponse, Object o, BindException e) throws
        Exception {

        return null;
    }
}
```

在讲解该控制器之前，首先我们要明白SpringMVC控制器一个与Struts2不同的地方：**SpringMVC的控制器是单例的，Struts2的控制器是多例的！**

也就是说：**Struts2**收集变量是定义成员变量来进行接收，而**SpringMVC**作为单例的，是不可能使用成员变量来进行接收的【因为会有多个用户访问，就会出现数据不合理性】！

那么SpringMVC作为单例的，他只能通过方法的参数来进行接收对应的参数！只有方法才能保证不同的用户对应不同的数据！

7.1 实体

实体的属性要和web页面上的name提交过来的名称是一致的。这和Struts2是一样的！

```
public class User {

    private String id;
    private String username;

    public User() {
    }

    public User(String id, String username) {
        this.id = id;
        this.username = username;
    }

    public String getId() {
        return id;
    }
}
```

```

    }

    public void setId(String id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @Override
    public String toString() {
        return "User{" +
            "id='" + id + '\'' +
            ", username='" + username + '\'' +
            '}';
    }
}

```

7.2提交参数的JSP

```

<form action="${pageContext.request.contextPath}/hello.action" method="post">
    <table align="center">
        <tr>
            <td>用户名: </td>
            <td><input type="text" name="username"></td>
        </tr>
        <tr>
            <td>编号</td>
            <td><input type="text" name="id"></td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" value="提交">
            </td>
        </tr>
    </table>

</form>

```

7.3 配置Action处理请求

```

<bean class="HelloAction" id="helloAction"></bean>

<!-- 注册映射器(handler包)(框架) -->
<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/hello.action">helloAction</prop>
        </props>
    </property>
</bean>

```

7.4 Action接收参数

```

public class HelloAction extends AbstractCommandController {

    /*设置无参构造器，里边调用setCommandClass方法，传入要封装的对象*/
    public HelloAction() {
        this.setCommandClass(User.class);
    }

    /**
     *
     * @param httpServletRequest
     * @param httpServletResponse
     * @param o 这里的对象就表示已经封装好的了User对象了。!
     * @param e
     * @return
     * @throws Exception
     */
    @Override
    protected ModelAndView handle(HttpServletRequest httpServletRequest,
        HttpServletResponse httpServletResponse, Object o, BindException e) throws
        Exception {

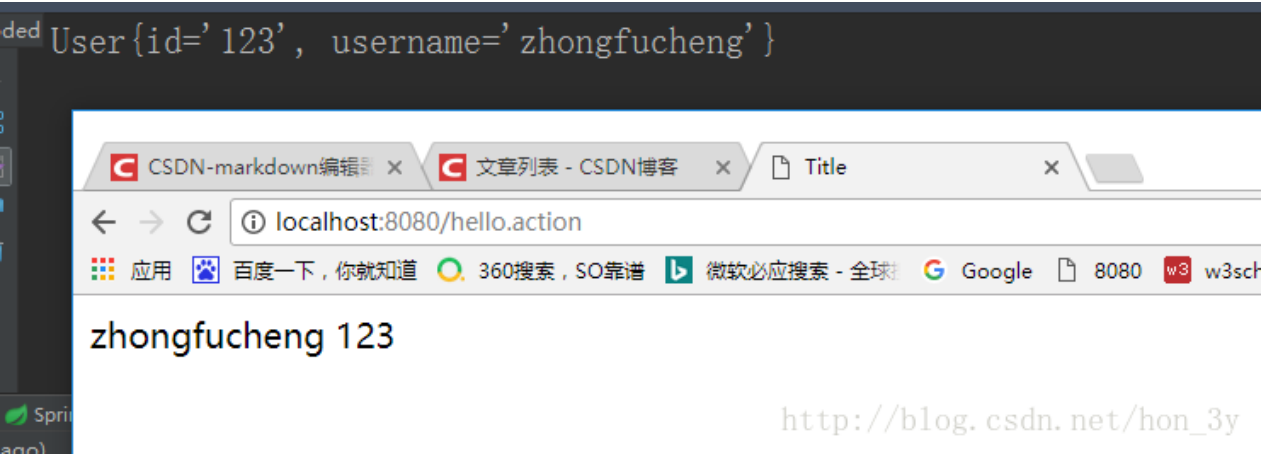
        User user = (User) o;

        System.out.println(user);

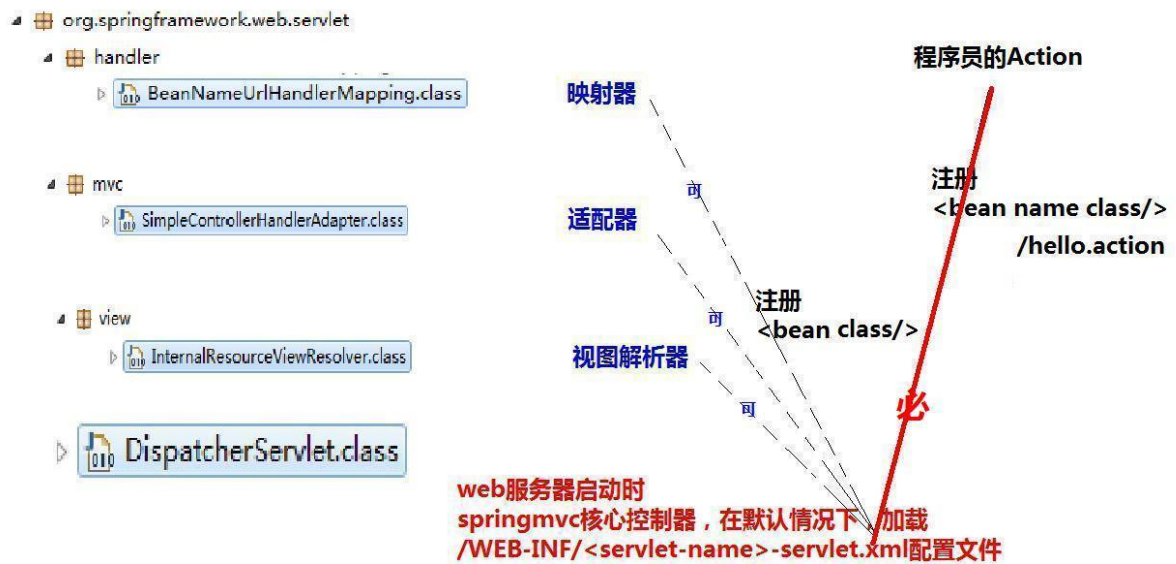
        ModelAndView modelAndView = new ModelAndView();
        //跳转到ok.jsp
        modelAndView.setViewName("/WEB-INF/ok.jsp");
        //将数据封装到ModelAndView中
        modelAndView.addObject("USER", user);
        return modelAndView;
    }
}

```

7.5测试效果：



八、小总结



映射器： web.servlet.handler.BeanNameUrlHandlerMapping （将bean标签的name属性值作为请求的url） web.servlet.handler.SimpleUrlHandlerMapping （多个请求，对应着一个Action）	BeanNameUrlHandlerMapping 【可省】	框架
适配器： 实现接口 web.servlet.mvc.SimpleControllerHandlerAdapter （只会找实现了Controller接口的后端控制器）	SimpleControllerHandlerAdapter 【可省】	框架
控制器： 继承类 web.servlet.mvc.ParameterizableViewController （根据请求，直接转到jsp页面，不经过action） web.servlet.mvc.AbstractCommandController （根据请求，调用处理action，以模型方式传入参数值）	AbstractCommandController 【可省】	框架
视图器： org.springframework.web.servlet.view.InternalResourceViewResolver （根据逻辑名，找到真实名）	InternalResourceViewResolver 【如果视图是真实路径，可省】	框架

程序员写的Action，必须注册到springioc容器中，不可省

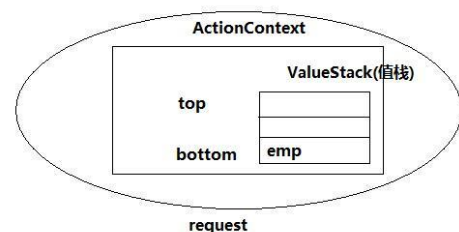
http://blog.csdn.net/hon_3y

Struts2和SpringMVC存值的区别：

struts2

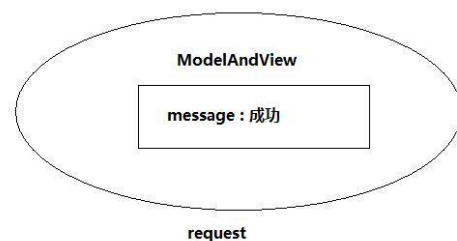
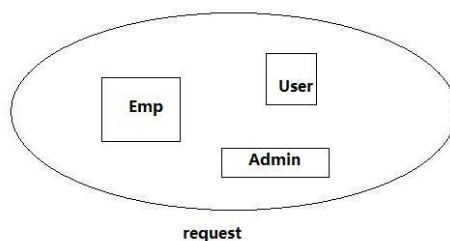
EmpAction
private Emp
set/get

取值：
1)struts2标签 + OGNL
2)JSTL + EL



springmvc

取值：
1)springEL
2)JSTL+EL



http://blog.csdn.net/hon_3y

- SpringMVC的工作流程:
 - 用户发送HTTP请求，SpringMVC核心控制器接收到请求
 - 找到映射器看该请求是否交由对应的Action类进行处理
 - 找到适配器看有无该Action类
 - Action类处理完结果封装到ModelAndView中
 - 通过视图解析器把数据解析，跳转到对应的JSP页面
- 控制器
 - AbstractCommandController
 - 可以实现对参数数据的封装

加油~



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

参数绑定、数据回显、文件上传

本文主要讲解的知识点如下：

- 参数绑定
- 数据回显
- 文件上传

一、参数绑定

我们在Controller使用方法参数接收值，就是把web端的值给接收到Controller中处理,这个过程就叫做参数绑定...

1.1默认支持的参数类型

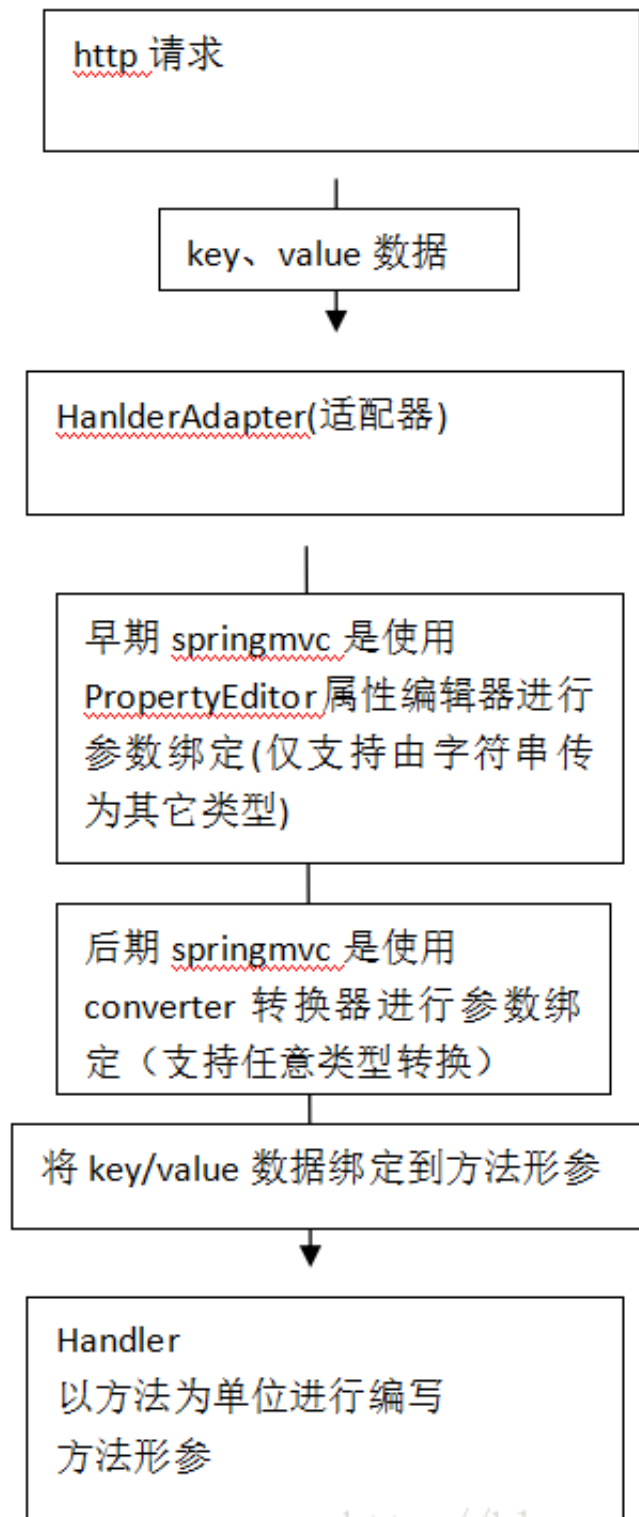
从上面的用法我们可以发现，我们可以使用request对象、Model对象等等，其实是不是可以随便把参数写上去都行？？？其实并不是的...

Controller方法默认支持的参数类型有4个，这4个足以支撑我们的日常开发了

- **HttpServletRequest**
- **HttpServletResponse**
- **HttpSession**
- **Model**

1.2参数的绑定过程

一般地，我们要用到自定义的参数绑定就是上面所讲的时间类型转换以及一些特殊的需求....对于平常的参数绑定，我们是无需使用转换器的，SpringMVC就已经帮我们干了这个活了...



http://blog.csdn.net/hon_3y

1.3自定义绑定参数【老方式、全部Action均可使用】

在上一篇我们已经简单介绍了怎么把字符串转换成日期类型了【使用的是WebDataBinder方式】...其实那是一个比较老的方法，我们可以使用SpringMVC更推荐的方式...

在上次把字符串转换成日期类型，如果使用的是WebDataBinder方式的话，那么该转换仅仅只能在当前Controller使用...如果想要全部的Controller都能够使用，那么我们可以使用WebBindingInitializer方式

如果想多个controller需要共同注册相同的属性编辑器，可以实现PropertyEditorRegistrar接口，并注入webBindingInitializer中。

实现接口

```
public class CustomPropertyEditor implements PropertyEditorRegistrar {

    @Override
    public void registerCustomEditors(PropertyEditorRegistry binder) {
        binder.registerCustomEditor(Date.class, new CustomDateEditor(
            new SimpleDateFormat("yyyy-MM-dd HH-mm-ss"), true));
    }

}
```

1.4配置转换器

注入到webBindingInitializer中

```
<!-- 注册属性编辑器 -->
<bean id="customPropertyEditor"
class="cn.itcast.ssm.controller.propertyeditor.CustomPropertyEditor"></bean>

<!-- 自定义webBinder -->
<bean id="customBinder"

class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer"
>

    <!-- propertyEditorRegistrars用于属性编辑器 -->
    <property name="propertyEditorRegistrars">
        <list>
            <ref bean="customPropertyEditor" />
        </list>
    </property>
</bean>

<!-- 注解适配器 -->
<bean

class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHan
dlerAdapter">
    <!-- 在webBindingInitializer中注入自定义属性编辑器、自定义转换器 -->
```

```
<property name="webBindingInitializer" ref="customBinder"></property>
</bean>
```

1.5 自定义参数转换器【新方式、推崇方式】

上面的方式是对象较老的，现在我们一般都是实现Converter接口来实现自定义参数转换...我们就来看看实现Converter比上面有什么好

配置日期转换器

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

    @Override
    public Date convert(String source) {
        try {
            //进行日期转换
            return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse(source);

        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

配置去除字符串转换器

```
public class StringTrimConverter implements Converter<String, String> {

    @Override
    public String convert(String source) {
        try {
            //去掉字符串两边空格，如果去除后为空设置为null
            if(source!=null){
                source = source.trim();
                if(source.equals("")){
                    return null;
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return source;
    }

}
```

```
}
```

从上面可以得出，我们想要转换什么内容，就直接实现接口，该接口又是支持泛型的，阅读起来就非常方便了...

1.6 配置转换器

```
<!-- 转换器 -->
<bean id="conversionService"

class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="converters">
        <list>
            <bean class="cn.itcast.ssm.controller.converter.CustomDateConverter"/>
            <bean class="cn.itcast.ssm.controller.converter.StringTrimConverter"/>
        </list>
    </property>
</bean>

<!-- 自定义webBinder -->
<bean id="customBinder"

class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer"
>
    <!-- 使用converter进行参数转 -->
    <property name="conversionService" ref="conversionService" />
</bean>

<!-- 注解适配器 -->
<bean

class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
    <!-- 在webBindingInitializer中注入自定义属性编辑器、自定义转换器 -->
    <property name="webBindingInitializer" ref="customBinder"></property>
</bean>
```

如果是基于 `<mvc:annotation-driven>` 的话，我们是这样配置的

```

<mvc:annotation-driven conversion-service="conversionService">
</mvc:annotation-driven>
<!-- conversionService -->
    <bean id="conversionService"

class="org.springframework.format.support.FormattingConversionServiceFactoryBe
an">
    <!-- 转换器 -->
    <property name="converters">
        <list>
            <bean class="cn.itcast.ssm.controller.converter.CustomDateConverter"/>
            <bean class="cn.itcast.ssm.controller.converter.StringTrimConverter"/>
        </list>
    </property>
</bean>

```

1.7 @RequestParam注解

我们一般使用的参数绑定都有遵循的规则：方法参数名要与传递过来的name属性名相同。

在默认的情况下，只有名字相同，SpringMVC才会帮我们进行参数绑定...

如果我们使用 @RequestParam注解 的话，我们就可以使方法参数名与传递过来的name属性名不同...

该注解有三个变量

- value 【指定name属性的名称是什么】
- required 【是否必须要有该参数】
- defaultValue设置默认值

例子：我们的方法参数叫id，而页面带过来的name属性名字叫item_id，一定需要该参数

```

public String editItem(@RequestParam(value="item_id",required=true) String id)
{

}

```

1.8 Controller方法返回值

Controller方法的返回值其实就几种类型，我们来总结一下....

- void
- String
- ModelAndView
- redirect重定向
- forward转发



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜Java3y公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

二、数据回显

其实数据回显我们现代的话就一点也不陌生了...我们刚使用EL表达式的时候就已经学会了数据回显了，做SSH项目的时候也有三圈问题的数据回显...

在页面上数据回显本质上就是获取request域的值..

而在我们SpringMVC中，我们是使用Model来把数据绑定request域对象中的

一般地我们都是使用model.addAttribute()的方式把数据绑定到request域对象中...其实SpringMVC还支持注解的方式

2.1 @ModelAttribute 注解

我们可以将请求的参数放到Model中，回显到页面上

```
public String editItemSubmit(Model model,Integer id,  
    @ModelAttribute(value="itemsCustom") ItemsCustom itemsCustom)throws Exception{  
    ...  
}
```

上面这种用法和model.addAttribute()的方式是没啥区别的，也体现不了注解的方便性...

而如果我们要回显的数据是公共的话，那么我们就能够体会到注解的方便性了，我们把公共需要显示的属性抽取成方法，将返回值返回就行了。

```
//单独将商品类型的方法提出来，将方法返回值填充到request，在页面显示
@ModelAttribute("itemsType")
public Map<String, String> getItemsType() throws Exception{

    HashMap<String, String> itemsType = new HashMap<String,String>();
    itemsType.put("001", "数码");
    itemsType.put("002", "服装");
    return itemsType;

}
```

http://blog.csdn.net/hon_3y

那我们就不用再在每一个controller方法通过Model将数据传到页面。

三、SpringMVC文件上传

我们使用Struts2的时候，觉得Struts2的文件上传方式比传统的文件上传方式好用多了...

既然我们正在学习SpringMVC，那么我们也看一下SpringMVC究竟是怎么上传文件的...

3.1配置虚拟目录

在这次，我们并不是把图片上传到我们的工程目录中...

那为啥不将图片直接上传到我们的工程目录中呢？？？我们仔细想想，按照我们之前的做法，直接把文件上传到工程目录，而我们的工程目录是我们写代码的地方...往往我们需要备份我们的工程目录。

如果把图片都上传到工程目录中，那么就非常难以处理图片了...

因此，我们需要配置Tomcat的虚拟目录来解决，把上传的文件放在虚拟目录上...

又值得注意的是，Idea使用的Tomcat并不能使用传统的配置方式，也就是修改server.xml方式来配置虚拟目录，在Idea下好像不支持这种做法...

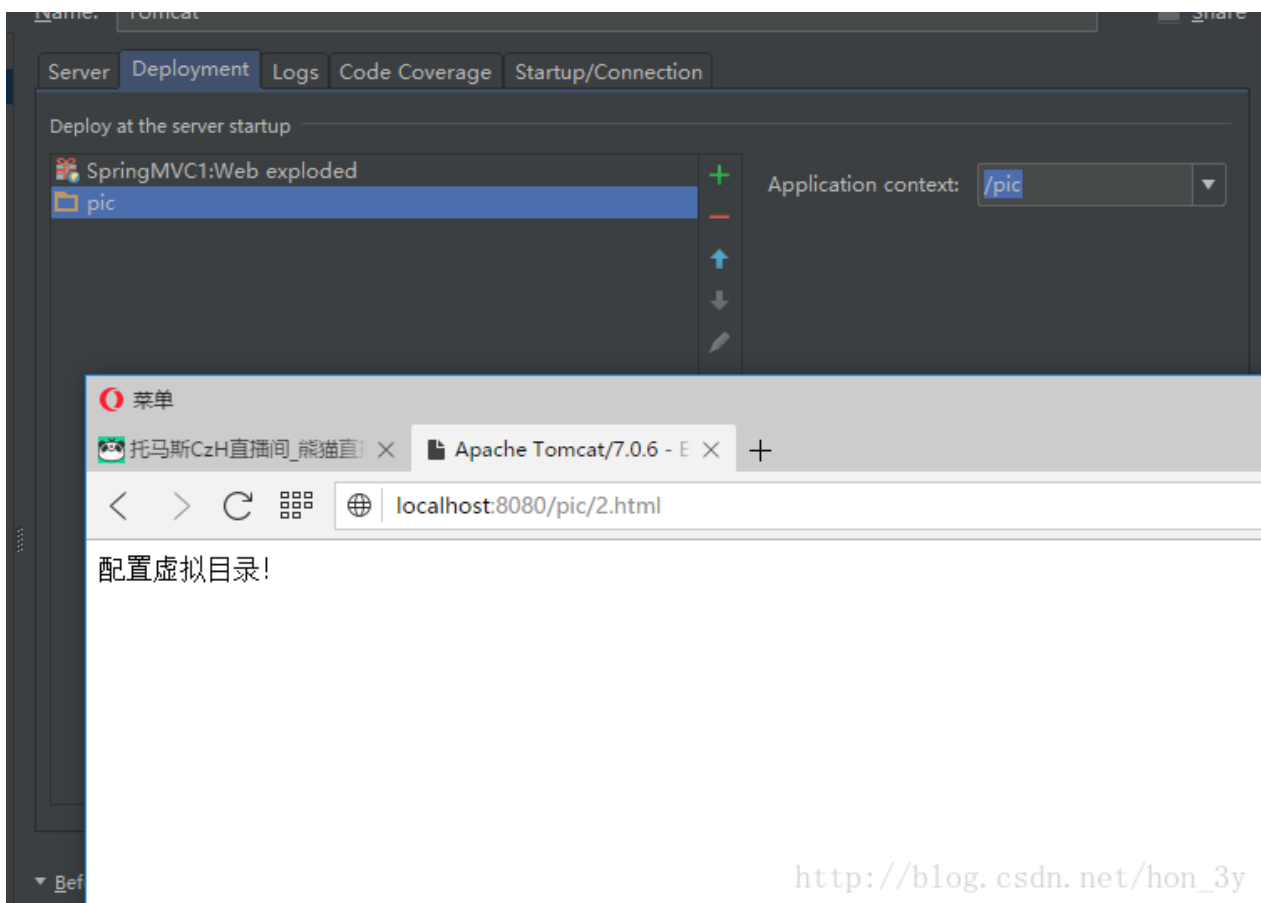
有兴趣的同学可以去测试一下：

http://blog.csdn.net/hon_3y/article/details/54412484

那么我在网上已经找到了对应的解决办法，就是如果在idea上配置虚拟目录

<http://blog.csdn.net/LABLENET/article/details/51160828>

检测是否配置成功：



http://blog.csdn.net/hon_3y

3.2快速入门

在SpringMVC中文件上传需要用到的jar包

如果用Maven的同学，引入pom就好了

- commons-fileupload-1.2.2.jar
- commons-io-2.4.jar

配置文件上传解析器

```
<!-- 文件上传 -->
<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <!-- 设置上传文件的最大尺寸为5MB -->
    <property name="maxUploadSize">
        <value>5242880</value>
    </property>
</bean>
```

测试的JSP

```
<%--
Created by IntelliJ IDEA.
User: ozc
```

```

Date: 2017/8/11
Time: 9:56
To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>测试文件上传</title>
</head>
<body>

<form action="${pageContext.request.contextPath}/upload.action" method="post"
enctype="multipart/form-data" >
    <input type="file" name="picture">
    <input type="submit" value="submit">
</form>

</body>
</html>

```

值得注意的是，在JSP的name属性写的是picture，那么在Controller方法参数的名称也是要写picture的，否则是获取不到对应的文件的..

```

@Controller
public class UploadController {
    @RequestMapping("/upload")
    //MultipartFile该对象就是封装了图片文件
    public void upload(MultipartFile picture) throws Exception {
        System.out.println(picture.getOriginalFilename());
    }
}

```

```

Connected to server
[2017-08-11 10:10:51,667] Artifact SpringMVC1:Web exploded: Artifact is being deployed, please w
[2017-08-11 10:10:51,668] Artifact pic: Artifact is being deployed, please wait...
[2017-08-11 10:11:02,028] Artifact SpringMVC1:Web exploded: Artifact is deployed successfully
[2017-08-11 10:11:02,029] Artifact SpringMVC1:Web exploded: Deploy took 10,361 milliseconds
[2017-08-11 10:11:02,082] Artifact pic: Artifact is deployed successfully
[2017-08-11 10:11:02,082] Artifact pic: Deploy took 10,414 milliseconds
log4j:WARN No appenders could be found for logger (org.springframework.web.servlet.DispatcherSer
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2017-08-11_094703.png http://blog.csdn.net/hon_3y

```

四、总结

- 在SpringMVC中的业务方法默认支持的参数有四种
 - request
 - response

- session
 - model
- 我们的参数绑定(自动封装参数)是由我们的转换器来进行绑定的。现在用的一般都是Converter转换器
- 在上一章中我们使用WebDataBinder方式来实现对日期格式的转化，当时仅仅是可用于当前Action的。我们想要让全部Action都可以使用的话，有两种方式：
 - 实现PropertyEditorRegistrar(比较老的方式)
 - 实现Converter(新的方式)
- 参数绑定都有遵循的规则：方法参数名要与传递过来的name属性名相同
 - 我们可以使用@RequestParam注解来具体指定对应的name属性名称，这样也是可以实现参数绑定的。
 - 还能够配置该参数是否是必须的。
- Controller方法的返回值有5种：
 - void
 - String
 - ModelAndView
 - redirect重定向
 - forward转发
- Model内部就是将数据绑定到request域对象中的。
- @ModelAttribute注解能够将数据绑定到model中(也就是request中)，如果经常需要绑定到model中的数据，抽取成方法来使用这个注解还是不错的。
- idea配置虚拟目录其实就是加多一个deployment，然后配置它的应用路径
- SpringMVC的文件上传就是配置一个上传解析器，使用MultipartFile来接收带过来的文件。





如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜Java3y公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

拦截器、统一处理异常、RESTful、拦截器

本博文主要讲解的知识点如下：

- 校验器
- 统一处理异常
- RESTful
- 拦截器

一、Validation




在我们的Struts2中，我们是继承ActionSupport来实现校验的...它有两种方式来实现校验的功能

- 手写代码
- XML配置
 - 这两种方式也是可以特定处理方法或者整个Action的

而SpringMVC使用JSR-303（javaEE6规范的一部分）校验规范，springmvc使用的是Hibernate Validator（和Hibernate的ORM无关）

1.1快速入门

导入jar包

 hibernate-validator-4.3.0.Final.jar
 jboss-logging-3.1.0.CR2.jar
 validation-api-1.0.0.GA.jar

配置校验器

```
<!-- 校验器 -->
<bean id="validator"

    class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
    <!-- 校验器 -->
    <property name="providerClass"
value="org.hibernate.validator.HibernateValidator" />
    <!-- 指定校验使用的资源文件，如果不指定则默认使用classpath下的
ValidationMessages.properties -->
    <property name="validationMessageSource" ref="messageSource" />
</bean>
```

错误信息的校验文件配置

```
<!-- 校验错误信息配置文件 -->
<bean id="messageSource"

class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <!-- 资源文件名 -->
    <property name="basenames">
        <list>
            <value>classpath:CustomValidationMessages</value>
        </list>
    </property>
    <!-- 资源文件编码格式 -->
    <property name="fileEncodings" value="utf-8" />
    <!-- 对资源文件内容缓存时间，单位秒 -->
    <property name="cacheSeconds" value="120" />
</bean>
```

添加到自定义参数绑定的WebBindingInitializer中

```

<!-- 自定义webBinder -->
<bean id="customBinder"

class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer"
>
    <!-- 配置validator -->
    <property name="validator" ref="validator" />
</bean>

```

最终添加到适配器中

```

<!-- 注解适配器 -->
<bean

class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHan
dlerAdapter">
    <!-- 在webBindingInitializer中注入自定义属性编辑器、自定义转换器 -->
    <property name="webBindingInitializer" ref="customBinder"></property>
</bean>

```

创建CustomValidationMessages配置文件

```

#校验提示信息，items.name.length.error要写在java代码中
items.name.length.error=商品名称的长度请限制在1到30个字符
items.createtime.is.notnull=请输入商品生产日期

```

定义规则

```

package entity;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.util.Date;

public class Items {
    private Integer id;

    //商品名称的长度请限制在1到30个字符
    @Size(min=1,max=30,message="{items.name.length.error}")
    private String name;

    private Float price;

    private String pic;

    //请输入商品生产日期

```



```
@NotNull(message="{items.createtime.is.notnull}")
private Date createtime;

private String detail;

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name == null ? null : name.trim();
}

public Float getPrice() {
    return price;
}

public void setPrice(Float price) {
    this.price = price;
}

public String getPic() {
    return pic;
}

public void setPic(String pic) {
    this.pic = pic == null ? null : pic.trim();
}

public Date getCreatetime() {
    return createtime;
}

public void setCreatetime(Date createtime) {
    this.createtime = createtime;
}

public String getDetail() {
    return detail;
}
```

```

    public void setDetail(String detail) {
        this.detail = detail == null ? null : detail.trim();
    }
}

```

测试:

```

<%--
    Created by IntelliJ IDEA.
    User: ozc
    Date: 2017/8/11
    Time: 9:56
    To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>测试文件上传</title>
</head>
<body>

<form action="${pageContext.request.contextPath}/validation.action"
method="post" >
    名称: <input type="text" name="name">
    日期: <input type="text" name="createtime">
    <input type="submit" value="submit">
</form>

</body>
</html>

```

Controller需要在校验的参数上添加@Validation注解...拿到BindingResult对象...

```

@RequestMapping("/validation")
public void validation(@Validated Items items, BindingResult bindingResult) {

    List<ObjectError> allErrors = bindingResult.getAllErrors();
    for (ObjectError allError : allErrors) {
        System.out.println(allError.getDefaultMessage());
    }
}

```

由于我在测试的时候，已经把日期转换器关掉了，因此提示了字符串不能转换成日期，但是名称的校验已经是出来了...

```
log4j:WARN No appenders could be found for logger (org.springframework.web.servlet.DispatcherServlet).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
进来了?
Failed to convert property value of type 'java.lang.String' to required type 'java.util.Date' for property 'createtime'
商品名称的长度请限制在1到30个字符
http://blog.csdn.net/hon_3y
```

1.2分组校验

分组校验其实就是为了我们的校验更加灵活，有的时候，我们并不需要把我们当前配置的属性都进行校验，而需要的是当前的方法仅仅校验某些的属性。那么此时，我们就可以用到分组校验了...

步骤：

- 定义分组的接口【主要是标识】
- 定于校验规则属于哪一各组
- 在Controller方法中定义使用校验分组

```
/**
 *
 * <p>Title: ValidGroup1</p>
 * <p>Description: 校验分组，用于商品修改的校验 </p>
 * <p>Company: www.itcast.com</p>
 * @author 传智.燕青
 * @date 2015-3-22下午2:35:34
 * @version 1.0
 */
public interface ValidGroup1 {
    //接口不定义方法，就是只标识 哪些校验 规则属于该 ValidGroup1分组
}
http://blog.csdn.net/hon_3y
```

```
//请输入商品生产日期
//通过groups指定此校验属于哪个分组，可以指定多个分组
@NotNull(message="{items.createtime.is.notnull}",groups={ValidGroup1.class})
private Date createtime;

//在@Validated中定义使用ValidGroup1组下边的校验
@RequestMapping("/editItemSubmit")
// public String editItemSubmit(Integer id,ItemsCustom itemsCustom,
// ItemsQueryVo itemsQueryVo)throws Exception{
public String editItemSubmit(Model model,Integer id,
    @Validated(value={ValidGroup1.class}) @ModelAttribute(value="itemsCustom") ItemsCustom itemsCustom,
    BindingResult bindingResult,
    // 返回结果
http://blog.csdn.net/hon_3y
```

二、统一异常处理

在我们之前SSH，使用Struts2的时候也配置过统一处理异常...

当时是这么干的：

- 在service层中自定义异常
- 在action层也自定义异常

- 对于Dao层的异常我们先不管【因为我们管不着，dao层的异常太致命了】
- service层抛出异常，Action把service层的异常接住，通过service抛出的异常来判断是否让请求通过
- 如果不通过，那么接着抛出Action异常
- 在Struts的配置文件中定义全局视图，页面显示错误信息

详情可看：http://blog.csdn.net/hon_3y/article/details/72772559

那么我们这次的统一处理异常的方案是什么呢？？？

我们知道Java中的异常可以分为两类

- 编译时期异常
- 运行期异常

对于运行期异常我们是无法掌控的，只能通过代码质量、在系统测试时详细测试等排除运行时异常

而对于编译时期的异常，我们可以在代码手动处理异常可以try/catch捕获，可以向上抛出。

我们可以换个思路，自定义一个模块化的异常信息，比如：商品类别的异常

```
public class CustomException extends Exception {

    //异常信息
    private String message;

    public CustomException(String message){
        super(message);
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```

我们在查看Spring源码的时候发现：前端控制器DispatcherServlet在进行HandlerMapping、调用HandlerAdapter执行Handler过程中，如果遇到异常，在系统中自定义统一的异常处理器，写系统自己的异常处理代码。。

```

protected ModelAndView processHandlerException(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
    ModelAndView exMv = null;
    Iterator i$ = this.handlerExceptionResolvers.iterator();

    while(i$.hasNext()) {
        HandlerExceptionResolver handlerExceptionResolver = (HandlerExceptionResolver)i$.next();
        exMv = handlerExceptionResolver.resolveException(request, response, handler, ex);
        if(exMv != null) {
            break;
        }
    }

    if(exMv != null) {

```

http://blog.csdn.net/hon_3y

```

private void processDispatchResult(HttpServletRequest request, HttpServletResponse response, HandlerExecutionChain mappedHandler, ModelAndView mv, Exception exception) throws Exception {
    boolean errorView = false;
    if(exception != null) {
        if(exception instanceof ModelAndViewDefiningException) {
            this.logger.debug("ModelAndViewDefiningException encountered", exception);
            mv = ((ModelAndViewDefiningException)exception).getModelAndView();
        } else {
            Object handler = mappedHandler != null ? mappedHandler.getHandler() : null;
            mv = this.processHandlerException(request, response, handler, exception);
            errorView = mv != null;
        }
    }
}

```

http://blog.csdn.net/hon_3y

我们也可以学着点，定义一个统一的处理器类来处理异常...

2.1 定义统一异常处理器类

```

public class CustomExceptionHandler implements HandlerExceptionResolver {

    //前端控制器DispatcherServlet在进行HandlerMapping、调用HandlerAdapter执行Handler过程中，如果遇到异常就会执行此方法
    //handler最终要执行的Handler，它的真实身份是HandlerMethod
    //Exception ex就是接收到异常信息
    @Override
    public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) {
        //输出异常
        ex.printStackTrace();

        //统一异常处理代码
        //针对系统自定义的CustomException异常，就可以直接从异常类中获取异常信息，将异常处理在错误页面展示
        //异常信息
        String message = null;
        CustomException customException = null;
        //如果ex是系统 自定义的异常，直接取出异常信息
        if(ex instanceof CustomException){
            customException = (CustomException)ex;
        }else{

```

```

        //针对非CustomException异常, 对这类重新构造一个CustomException, 异常信息为“未知错误”
        customException = new CustomException("未知错误");
    }

    //错误 信息
    message = customException.getMessage();

    request.setAttribute("message", message);

    try {
        //转向到错误 页面
        request.getRequestDispatcher("/WEB-INF/jsp/error.jsp").forward(request, response);
    } catch (ServletException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return new ModelAndView();
}
}

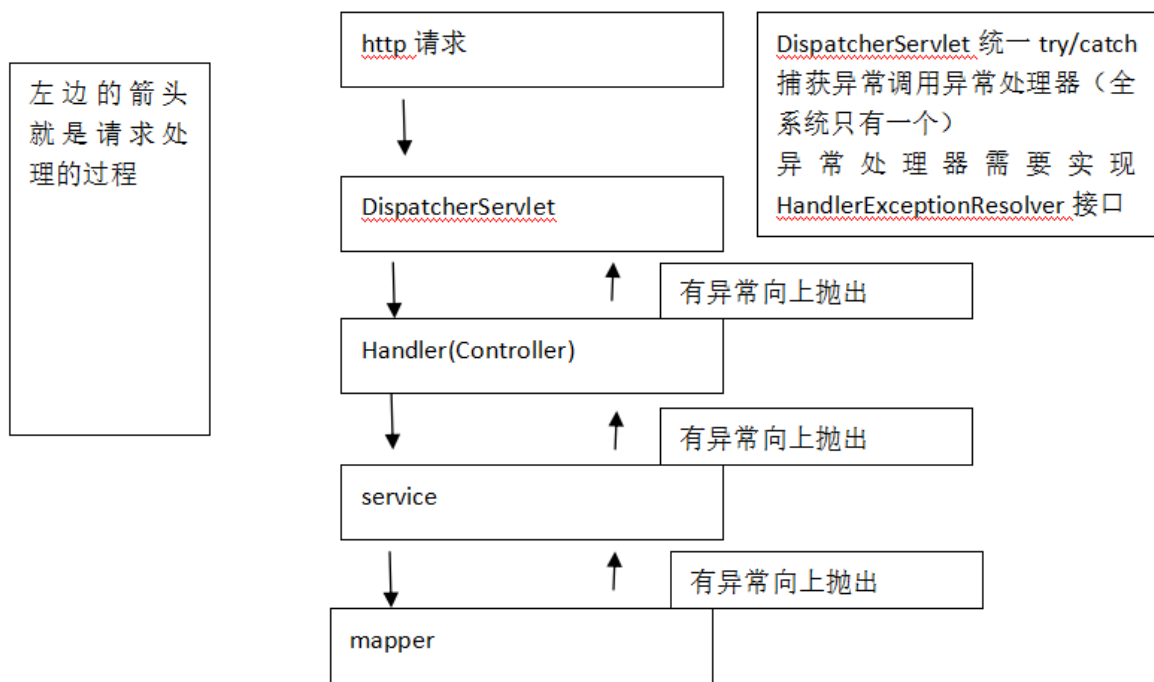
```

2.2配置统一异常处理器

```

<!-- 定义统一异常处理器 -->
<bean class="cn.itcast.ssm.exception.CustomExceptionResolver"></bean>

```



要求 `dao`、`service`、`controller` 遇到异常全部向上抛出异常，方法向上抛出异常 `throws Exception`

http://blog.csdn.net/hon_3y





如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜Java3y公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

三、RESTful支持

我们在学习webservice的时候可能就听过RESTful这么一个名词，当时与SOAP进行对比的...那么RESTful究竟是什么东东呢???

RESTful(Representational State Transfer)软件开发理念，RESTful对http进行非常好的诠释。

如果一个架构支持RESTful，那么就称它为RESTful架构...

以下的文章供我们了解：

<http://www.ruanyifeng.com/blog/2011/09/restful>

综合上面的解释，我们总结一下什么是RESTful架构：

- (1) 每一个URI代表一种资源；
- (2) 客户端和服务端之间，传递这种资源的某种表现层；
- (3) 客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

关于RESTful幂等性的理解：<http://www.oschina.net/translate/put-or-post>

简单来说，如果对象在请求的过程中会发生变化(以Java为例子，属性被修改了)，那么此是非幂等的。多次重复请求，结果还是不变的话，那么就是幂等的。

PUT用于幂等请求，因此在更新的时候把所有的属性都写完整，那么多次请求后，我们其他属性是不会变的

在上边的文章中，幂等被翻译成“状态统一性”。这就更好地理解了。

其实一般的架构并不能完全支持RESTful的，因此，只要我们的系统支持RESTful的某些功能，我们一般就称作为支持RESTful架构...

3.1url的RESTful实现

非RESTful的http的url: <http://localhost:8080/items/editItems.action?id=1&...>

RESTful的url是简洁的: <http://localhost:8080/items/editItems/1>

3.2更改DispatcherServlet的配置

从上面我们可以发现, url并没有.action后缀的, 因此我们要修改核心分配器的配置

```
<!-- restful的配置 -->
<servlet>
    <servlet-name>springmvc_rest</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <!-- 加载springmvc配置 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <!-- 配置文件的地址 如果不配置contextConfigLocation, 默认查找的配置文件名称
classpath下的: servlet名称+"-servlet.xml"即: springmvc-servlet.xml -->
        <param-value>classpath:spring/springmvc.xml</param-value>
    </init-param>

</servlet>
<servlet-mapping>
    <servlet-name>springmvc_rest</servlet-name>
    <!-- rest方式配置为/ -->
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

在Controller上使用PathVariable注解来绑定对应的参数

```
//根据商品id查看商品信息rest接口
//@RequestMapping中指定restful方式的url中的参数, 参数需要用{}包起来
//@PathVariable将url中的{}包起参数和形参进行绑定
@RequestMapping("/viewItems/{id}")
public @ResponseBody ItemsCustom viewItems(@PathVariable("id") Integer id)
throws Exception{
    //调用 service查询商品信息
    ItemsCustom itemsCustom = itemsService.findItemsById(id);

    return itemsCustom;
}
```

当DispatcherServlet拦截/开头的请求，对静态资源的访问就报错：我们需要配置对静态资源的解析

```
<!-- 静态资源 解析 -->
<mvc:resources location="/js/" mapping="/js/**" />
<mvc:resources location="/img/" mapping="/img/**" />
```

/** 就表示不管有多少层，都对其进行解析，/* 代表的是当前层的所有资源..

四、SpringMVC拦截器

在Struts2中拦截器就是我们当时的核心，原来在SpringMVC中也是有拦截器的

用户请求到DispatcherServlet中，DispatcherServlet调用HandlerMapping查找Handler，HandlerMapping返回一个拦截的链儿（多个拦截），springmvc中的拦截器是通过HandlerMapping发起的。

实现拦截器的接口：

```
public class HandlerInterceptor1 implements HandlerInterceptor {

    //在执行handler之前来执行的
    //用于用户认证校验、用户权限校验
    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {

        System.out.println("HandlerInterceptor1...preHandle");

        //如果返回false表示拦截不继续执行handler，如果返回true表示放行
        return false;
    }

    //在执行handler返回modelAndView之前来执行
    //如果需要向页面提供一些公用 的数据或配置一些视图信息，使用此方法实现 从modelAndView入手
    @Override
    public void postHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler,
        ModelAndView modelAndView) throws Exception {
        System.out.println("HandlerInterceptor1...postHandle");
    }

    //执行handler之后执行此方法
    //作系统 统一异常处理，进行方法执行性能监控，在preHandle中设置一个时间点，在
    afterCompletion设置一个时间，两个时间点的差就是执行时长
    //实现 系统 统一日志记录
```

```

@Override
public void afterCompletion(HttpServletRequest request,
    HttpServletResponse response, Object handler, Exception ex)
    throws Exception {
    System.out.println("HandlerInterceptor1...afterCompletion");
}

}

```

配置拦截器

```

<!--拦截器 -->
<mvc:interceptors>
    <!--多个拦截器,顺序执行 -->
    <!-- <mvc:interceptor>
        <mvc:mapping path="/**" />
        <bean class="cn.itcast.ssm.controller.interceptor.HandlerInterceptor1">
    </bean>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <bean class="cn.itcast.ssm.controller.interceptor.HandlerInterceptor2">
    </bean>
    </mvc:interceptor> -->

    <mvc:interceptor>
        <!-- /**可以拦截路径不管多少层 -->
        <mvc:mapping path="/**" />
        <bean class="cn.itcast.ssm.controller.interceptor.LoginInterceptor">
    </bean>
    </mvc:interceptor>
</mvc:interceptors>

```

4.1测试执行顺序

如果两个拦截器都放行

测试结果：

```
HandlerInterceptor1...preHandle  
HandlerInterceptor2...preHandle
```

```
HandlerInterceptor2...postHandle  
HandlerInterceptor1...postHandle
```

```
HandlerInterceptor2...afterCompletion  
HandlerInterceptor1...afterCompletion
```

总结：

执行preHandle是顺序执行。

执行postHandle、afterCompletion是倒序执行

1 号放行和2号不放行

测试结果：

```
HandlerInterceptor1...preHandle  
HandlerInterceptor2...preHandle  
HandlerInterceptor1...afterCompletion
```

总结：

如果preHandle不放行，postHandle、afterCompletion都不执行。

只要有一个拦截器不放行，controller不能执行完成

1 号不放行和2号不放行

测试结果：

```
HandlerInterceptor1...preHandle
```

总结：

只有前边的拦截器preHandle方法放行，下边的拦截器的preHandle才执行。

日志拦截器或异常拦截器要求

- 将日志拦截器或异常拦截器放在拦截器链儿中第一个位置，且preHandle方法放行

4.2拦截器应用-身份认证

拦截器拦截

```
public class LoginInterceptor implements HandlerInterceptor {  
  
    //在执行handler之前来执行的  
    //用于用户认证校验、用户权限校验  
    @Override  
    public boolean preHandle(HttpServletRequest request,  
        HttpServletResponse response, Object handler) throws Exception {
```

```

//得到请求的url
String url = request.getRequestURI();

//判断是否是公开 地址
//实际开发中需要公开 地址配置在配置文件中
//...
if(url.indexOf("login.action")>=0){
    //如果是公开 地址则放行
    return true;
}

//判断用户身份在session中是否存在
HttpSession session = request.getSession();
String usercode = (String) session.getAttribute("usercode");
//如果用户身份在session中存在放行
if(usercode!=null){
    return true;
}
//执行到这里拦截, 跳转到登陆页面, 用户进行身份认证
request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request,
response);

//如果返回false表示拦截不继续执行handler, 如果返回true表示放行
return false;
}

//在执行handler返回modelAndView之前来执行
//如果需要向页面提供一些公用 的数据或配置一些视图信息, 使用此方法实现 从modelAndView入手
@Override
public void postHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler,
    ModelAndView modelAndView) throws Exception {
    System.out.println("HandlerInterceptor1...postHandle");
}

//执行handler之后执行此方法
//作系统 统一异常处理, 进行方法执行性能监控, 在preHandle中设置一个时间点, 在
afterCompletion设置一个时间, 两个时间点的差就是执行时长
//实现 系统 统一日志记录
@Override
public void afterCompletion(HttpServletRequest request,
    HttpServletResponse response, Object handler, Exception ex)
    throws Exception {
    System.out.println("HandlerInterceptor1...afterCompletion");
}
}

```

```

@Controller
public class LoginController {

    //用户登陆提交方法
    @RequestMapping("/login")
    public String login(HttpSession session, String usercode,String
password)throws Exception{

        //调用service校验用户账号和密码的正确性
        //..

        //如果service校验通过, 将用户身份记录到session
        session.setAttribute("usercode", usercode);
        //重定向到商品查询页面
        return "redirect:/items/queryItems.action";
    }

    //用户退出
    @RequestMapping("/logout")
    public String logout(HttpSession session)throws Exception{

        //session失效
        session.invalidate();
        //重定向到商品查询页面
        return "redirect:/items/queryItems.action";

    }

}

```

五、总结

- 使用Spring的校验方式就是将要校验的属性前边加上注解声明。
- 在Controller中的方法参数上加上@Validation注解。那么SpringMVC内部就会帮我们对其进行处理(创建对应的bean, 加载配置文件)
- BindingResult可以拿到我们校验错误的提示
- 分组校验就是将让我们的校验更加灵活:某方法需要校验这个属性, 而某方法不用校验该属性。我们就可以使用分组校验了。
- 对于处理异常, SpringMVC是用一个统一的异常处理器类的。实现了HandlerExceptionResolver接口。
- 对模块细分多个异常类, 都交由我们的统一异常处理器类进行处理。

- 对于RESTful规范，我们可以使用SpringMVC简单地支持的。将SpringMVC的拦截.action改成是任意的。同时，如果是静态的资源文件，我们应该设置不拦截。
- 对于url上的参数，我们可以使用@PathVariable将url中的{}包起参数和形参进行绑定
- SpringMVC的拦截器和Struts2的拦截器差不多。不过SpringMVC的拦截器配置起来比Struts2的要简单。
 - 至于他们的拦截器链的调用顺序，和Filter的是没有差别的。



加油



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>