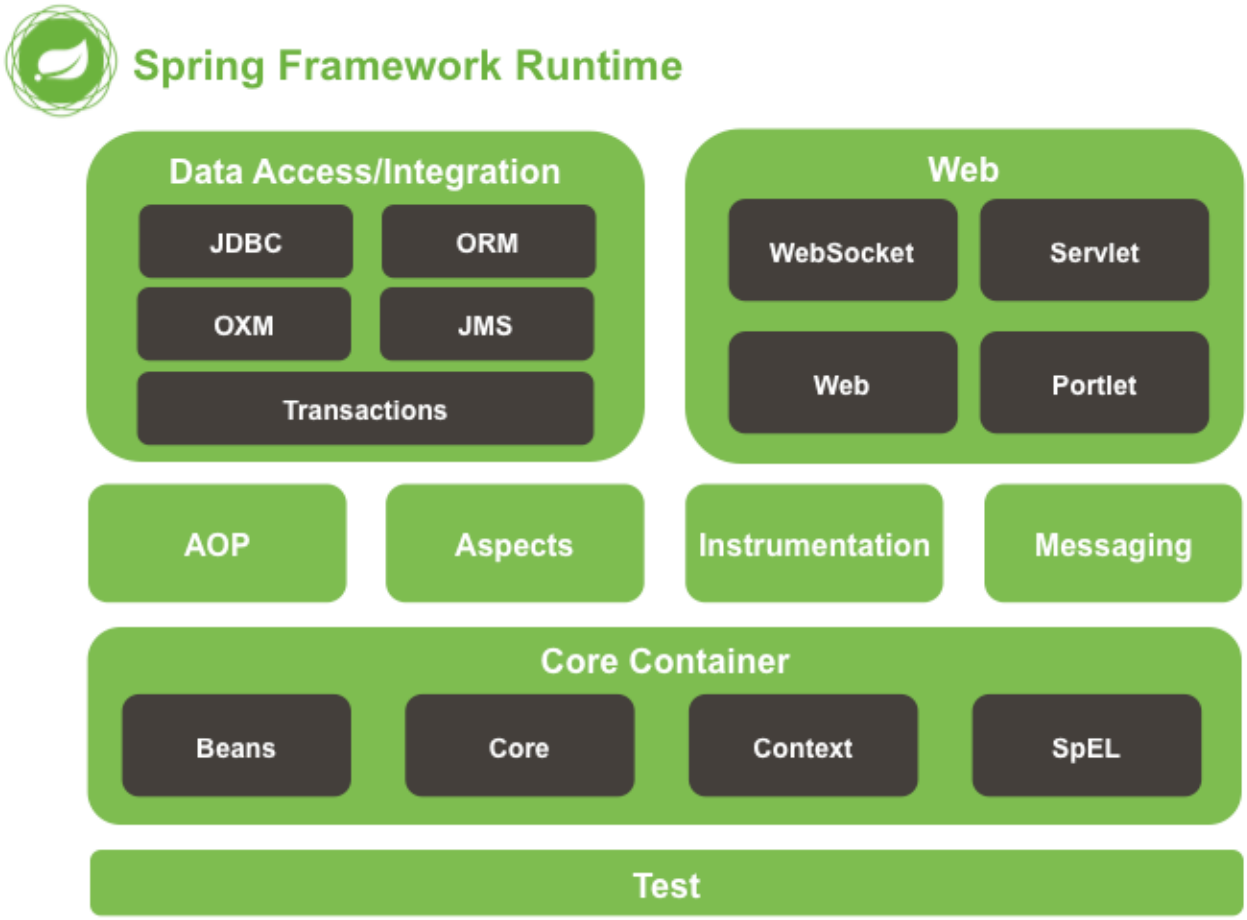
1.springMVC介绍

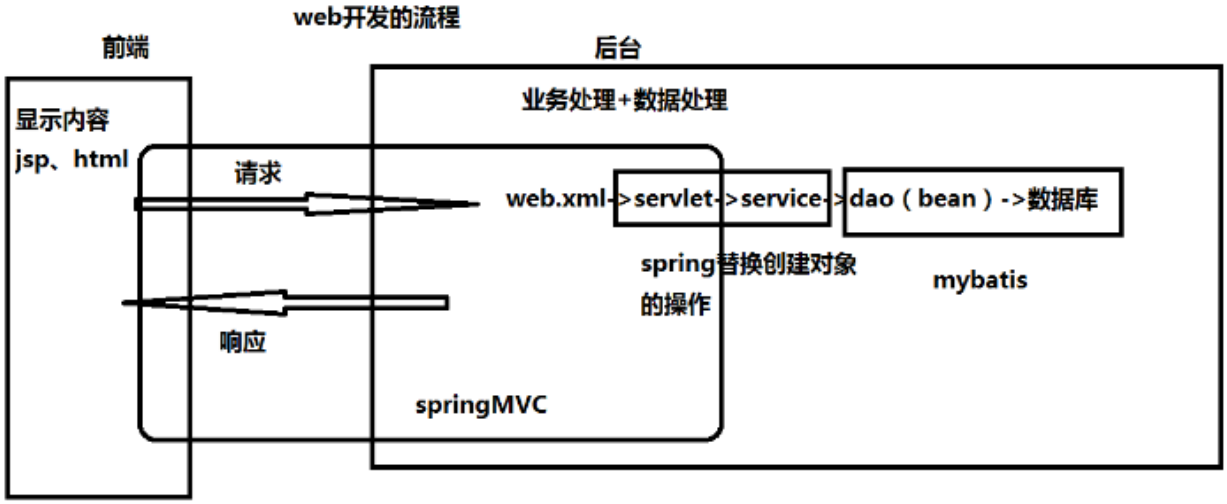


Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。 使用 Spring 可插入的 MVC 架构，可以选择是使用内置的 Spring Web 框架还是 Struts 这样的 Web 框架。通过策略接口，**Spring 框架是高度可配置的，而且包含多种视图技术**，例如 JavaServer Pages（JSP）技术、Velocity、Tiles、iText 和 POI。Spring MVC 框架并不知道使用的视图，所以不会强迫您只使用 JSP 技术。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

mvc架构的应用

webwork,struts1,struts2,springmvc...

2.web请求过程



3. springMVC组件介绍

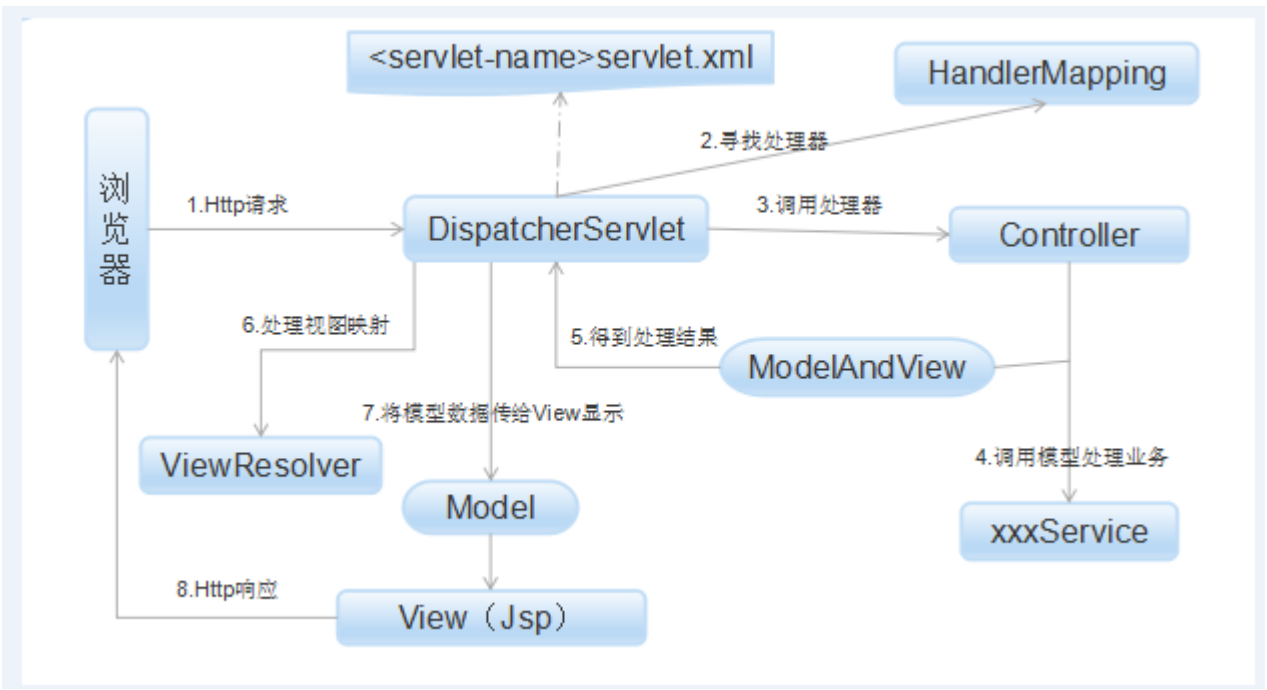
DispatcherServlet：作为前端控制器，整个流程控制的中心，控制其它组件执行，统一调度，降低组件之间的耦合性，提高每个组件的扩展性。

HandlerMapping：通过扩展处理器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。

HandlAdapter：通过扩展处理器适配器，支持更多类型的处理器,调用处理器传递参数等工作!

ViewResolver：通过扩展视图解析器，支持更多类型的视图解析，例如：jsp、freemarker、pdf、excel等。

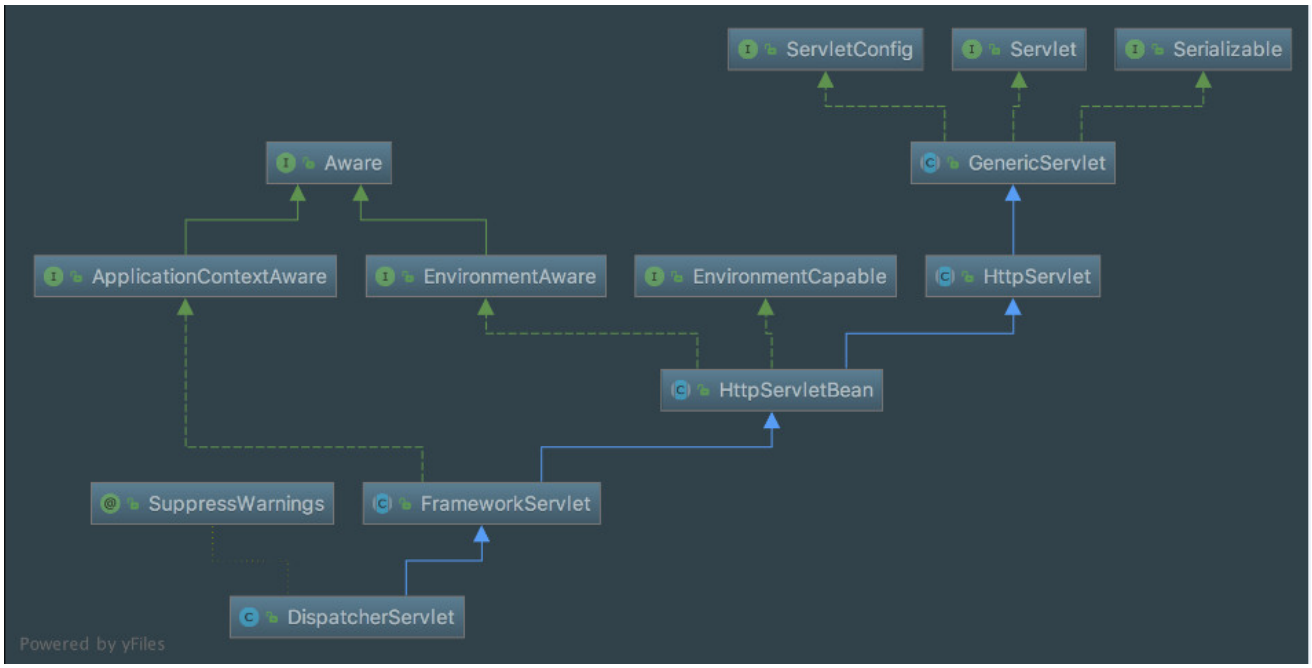
4.MVC执行过程



4.1Dispatcher介绍

DispatcherServlet主要用作职责调度工作，本身主要用于控制流程，主要职责如下：

1. 文件上传解析，如果请求类型是multipart将通过MultipartResolver进行文件上传解析；
2. 通过HandlerMapping，将请求映射到处理器（返回一个HandlerExecutionChain，它包括一个处理器、多个HandlerInterceptor拦截器）；
3. 通过HandlerAdapter支持多种类型的处理器(HandlerExecutionChain中的处理器)；
4. 通过ViewResolver解析逻辑视图名到具体视图实现；
5. 本地化解析；
6. 渲染具体的视图等；
7. 如果执行过程中遇到异常将交给HandlerExceptionResolver来解析。



4.2 DispatcherServlet辅助类

spring中的DispatcherServlet使用一些特殊的bean来处理request请求和渲染合适的视图。

bean类型	说明
Controller	处理器/页面控制器，做的是MVC中的C的事情，但控制逻辑转移到前端控制器了，用于对请求进行处理
HandlerMapping	请求到处理器的映射，如果映射成功返回一个HandlerExecutionChain对象（包含一个Handler处理器（页面控制器）对象、多个HandlerInterceptor拦截器）对象；如BeanNameUrlHandlerMapping将URL与Bean名字映射，映射成功的Bean就是此处的处理器
HandlerAdapter	HandlerAdapter将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；如SimpleControllerHandlerAdapter将对实现了Controller接口的Bean进行适配，并且掉处理器的handleRequest方法进行功能处理
HandlerExceptionResolver处理器异常解析器	处理器异常解析，可以将异常映射到相应的统一错误界面，从而显示用户友好的界面（而不是给用户看到具体的错误信息）
ViewResolver视图解析器	ViewResolver将把逻辑视图名解析为具体的View，通过这种策略模式，很容易更换其他视图技术；如InternalResourceViewResolver将逻辑视图名映射为jsp视图
LocaleResolver & LocaleContextResolver地区解析器和地区Context解析器	解析客户端中使用的地区和时区，用来提供不同的国际化的view视图。
ThemeResolver	主题解析器,解析web应用中能够使用的主题，比如提供个性化的网页布局。
MultipartResolver	多部件解析器,主要处理multi-part(多部件)request请求，例如：在HTML表格中处理文件上传。
FlashMapManager	FlashMap管理器储存并检索在"input"和"output"的FlashMap中可以在request请求间(通常是通过重定向)传递属性的FlashMap

5.springMVC搭建

(1)添加jar包

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
```

```
    <version>5.0.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>5.0.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>5.0.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

(2)修改web.xml

```
<servlet>
    <servlet-name>springMVC</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring.xml</param-value>
    </init-param>
 </servlet>
 <servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <url-pattern>/</url-pattern>
 </servlet-mapping>
```

2.1 /和/*的区别

< url-pattern > / </ url-pattern > 不会匹配到*.jsp，即：*.jsp不会进入spring的 DispatcherServlet类 。

< url-pattern > /* </ url-pattern > 会匹配*.jsp，会出现返回jsp视图时再次进入spring的DispatcherServlet 类，导致找不到对应的controller所以报404错。

可以配置/ ，此工程 所有请求全部由springmvc解析，此种方式可以实现 RESTful方式，需要特殊处理对静态文件的解析不能由springmvc解析

可以配置.*do或*.action，所有请求的url扩展名为.do或.action由springmvc解析，此种方法常用

不可以/*，如果配置/*，返回jsp也由springmvc解析，这是不对的。

2.2 url-pattern有5种配置模式

（1）/xxx:完全匹配/xxx的路径

（2）/xxx/*:匹配以/xxx开头的路径，请求中必须包含xxx。

（3）/*：匹配/下的所有路径,请求可以进入到action或controller，但是转发jsp时再次被拦截，不能访问jsp界面。

（4）.xx:匹配以xx结尾的路径，所有请求必须以.xx结尾，但不会影响访问静态文件。

（5）/:默认模式，未被匹配的路径都将映射到刺servlet，对jpg，js，css等静态文件也将被拦截，不能访问。

(3)修改spring配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd"
>
<!-- 扫描controller-->
<context:component-scan base-package="com.yhp.controller"/>
     <!-- 视图解析器 -->
     <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- jsp所在的位置-->
        <property name="prefix" value="/" />
        <!-- jsp文件的后缀名-->
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

(4)创建控制器类

1.@Controller

2.@RequestMapping("请求地址")

* 加在类上: 给模块添加根路径

* 加载方法: 方法具体的路径

- 设置@RequestMapping method属性

@RequestMapping(method=RequestMethod.GET,value="请求名")

(5)测试

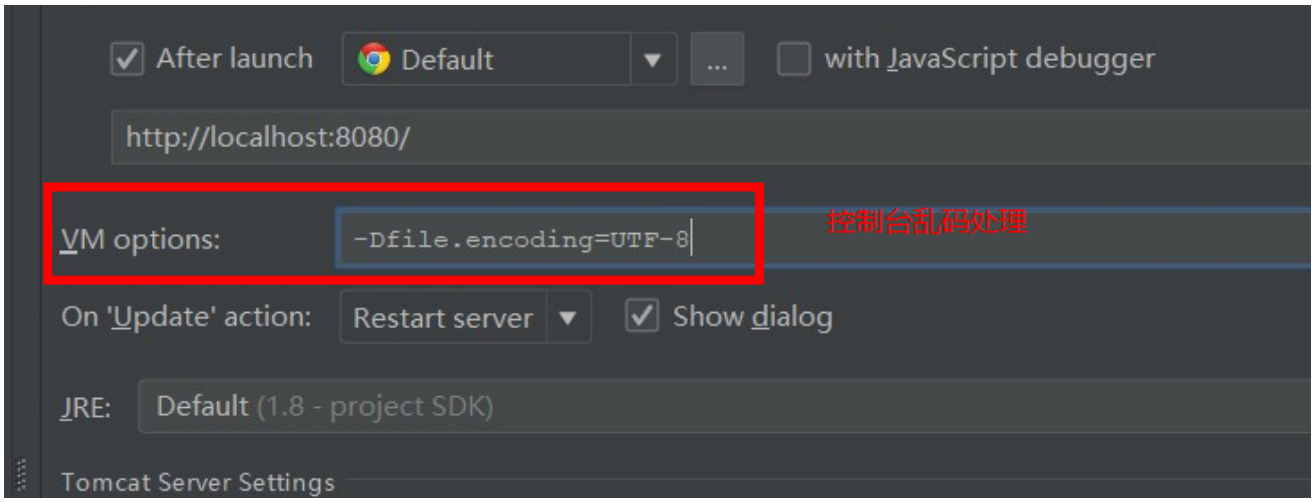
6.接参

接收方式:

(1)HttpServletRequest

(2)页面传值时的key=处理请求的方法的参数名

(3)使用控件名和对象的属性名一致的方式进行接收

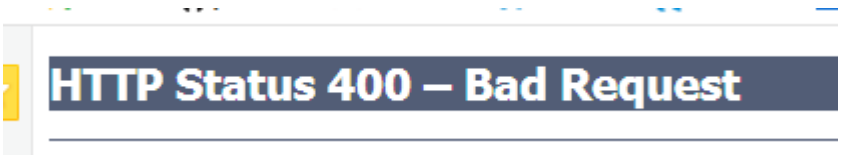


方法的参数名与传参的name值不同

示例:

```
public  String login(@RequestParam(value = "name") String username, String password){}
//设置默认值
public  String list(@RequestParam(defaultValue = "1") Integer currentPage)
```

常见错误:



错误原因：给定的数据无法由框架转换成目标类型

springmvc框架默认支持转换得日期格式:yyyy/MM/dd

解决日期问题方式:

（1）使用string接受日期，接受后，再转换： SimpleDataFormate

（2）使用工具类处理日期

日期处理:

```
<dependency>
     <groupId>joda-time</groupId>
     <artifactId>joda-time</artifactId>
     <version>2.9.9</version>
</dependency>
```

```
配置文件:<mvc:annotation-driven/>
```

```
public String test1(@DateTimeFormat(pattern = "yyyy-MM-dd")Date birthday){}
```

补充:参数类型使用引用数据类型，基本数据类型使用包装类

7.返参

修改web.xml文件版本,用来支持jsp操作EL表达式

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
```

(1)HttpServletRequest

(2)ModelMap map ,默认作用域request

(3)ModelAndView 对象需要new,同时作为返回值类型

(4)Model类保存数据

8.session存值

（1）使用HttpSession： request.getSession();

（2）使用@sessionAttributes("key值")//写的是ModelMap中定义的key值

注：该注解和ModelMap结合使用,当使用ModelMap存值时,会在session中同时存储一份数据

@SessionAttributes()的小括号中如果是一个值，不要加{}

示例:

@SessionAttributes("key")

@SessionAttributes({"key1","key2"})

清除注解session:SessionStatus类

status.setComplete();

9.弹窗响应

输出流的问题（返回值必须是void）

```
    @RequestMapping("delete")
    public void  delete(HttpServletResponse response) throws IOException{
        System.out.println("删除成功");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter pw=response.getWriter();
        pw.print("<script type='text/javascript'>alert('删除成
功');location.href='MyJsp.jsp'</script>");
    }
```

补充：如何设置get请求乱码？tomcat8已经默认配置

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

post处理乱码

注意:这里只能处理接受后数据的编码，如果是其他位置的编码问题还需要再考虑处理方案

```
<!-- 处理post乱码 -->
  <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>*.do</url-pattern>
  </filter-mapping>
```

10.转发和重定向

默认转发跳转

```
@RequestMapping("/forwardView")
public String forwardView(){
    return "forward:/WEB_INF/pages/success.jsp";
}
```

重定向：

```
return "redirect:a.jsp" 或者:redirect:findall
```

注意：重定向时地址栏会发生拼接modelmap中值的问题

11.异常处理

方法1:在web.xml响应状态码配置一个对应页面

```
<error-page>
   <error>404</error>
   <location>/404.html</location>
</error-page>
```

方法2:

```
@RequestMapping("/login9")
public String login9(){
   String a=null;
   System.out.println(a.charAt(0));
   return "login9.jsp";
}

//配置异常结果界面
@ExceptionHandler(NullPointerException.class)
public String exeception Result(){
    return "exception";
}
```

全局异常:@ControllerAdvice

使一个Contoller成为全局的异常处理类，类中用@ExceptionHandler方法注解的方法可以处理所有Controller发生的异常

12.Cookie操作

```
@CookieValue注解可以获取请求中的cookie
    public String testCookie(@CookieValue("JSESSIONID")String cookie)
    {
        System.out.println("cookie:"+cookie);
        return "result";
    }
```

13.获得头信息

```
@RequestHeader
@RequestHeader注解可以获取请求头中的数据!!
    public String testHeader(@RequestHeader("User-Agent")String header)
```

14.RestFul风格

REST:即Representational State Transfer ,(资源)表现层状态转化,是目前最流行的一种互联网软件架构。

具体说，就是HTTP协议里面,四个表示操作方式的动词:

GET POST PUT DELETE

它们分别代表着四种基本操作:

- GET用来获取资源

- POST用来创建新资源

- PUT用来更新资源

- DELETE用来删除资源

示例:

order?method=insert&id=1

order?method=delete&id=1

order?method=update&id=1

order?method=select&id=1

- /order/1 HTTP GET :得到id = 1 的 order

- /order/1 HTTP DELETE: 删除 id=1 的order

- /order/1 HTTP PUT : 更新id = 1的 order

- /order/1 HTTP POST : 新增 order

通过修改http的状态值来标记请求的目的

Spring中实现RESTful风格

HiddenHttpMethodFilter:浏览器form表单只支持GET和POST,不支持DELETE和PUT请求,

Spring添加了一个过滤器,可以将这些请求转换为标准的http方法,支持GET,POST,DELETE,PUT请求!

实现步骤:

(1) web.xml添加HiddenHttpMethodFilter配置

```
<filter>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <filter-class>
    org.springframework.web.filter.HiddenHttpMethodFilter
    </filter-class>
 </filter>
 <filter-mapping>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
 </filter-mapping>
```

(2) 实现查,改,删 框架

```
@RequestMapping(value = "/list",method = RequestMethod.GET)
@RequestMapping(value = "/{id}",method = RequestMethod.DELETE)
@RequestMapping(value = "/{id}",method = RequestMethod.PUT)
```

(3) Jsp代码:

```
  <a href="javascript:void(0)" onclick="deleteById()">删除</a>
  <form action="/order/1" method="post" id="deleteForm">
      <input type="hidden" name="_method" value="DELETE" />
  </form>
```

```
  <a href="javascript:void(0)" onclick="updateById()">修改</a>
     <form action="/order/1" method="post" id="updateForm">
        <input type="hidden" name="_method" value="PUT" />
    </form>
```

```
 <script>
          function updateById() {
              var form = document.getElementById("updateForm");
              form.submit();
          }
         function deleteById() {
             var form = document.getElementById("deleteForm");
             form.submit();
         }
</script>
```

需要注意: 由于doFilterInternal方法只对method为post的表单进行过滤，所以在页面中必须如下设置:

```
  <form action="..." method="post">
       <input type="hidden" name="_method" value="put" />
  </form>
```

代表post请求,但是HiddenHttpMethodFilter将把本次请求转化成标准的put请求方式! name="_method"固定写法!

(4) controller

@PathVariable获取路径参数

```
@RequestMapping("/user/list/{id}")
public  String getData(@PathVariable(value = "id") Integer id){
        System.out.println("id = " + id);
        return "list" ;
}
```

其他请求:

```
@RequestMapping(value = "/order",method = RequestMethod.POST)
@RequestMapping(value = "/order/{id}",method = RequestMethod.DELETE)
@RequestMapping(value = "/order/{id}",method = RequestMethod.PUT)
@RequestMapping(value = "/order",method = RequestMethod.GET)
```

注意:如果访问put和delete请求的时候，报405：method not allowed。处理方式是将过滤器的请求地址改成/，而不是/*

15.静态资源访问

需要注意一种,DispatcherServlet拦截资源设置成了 / 避免了死循环,但是 / 不拦截jsp资源,但是它会拦截其他静态资源,例如 html , js , css,image等等, 那么我们在使用jsp内部添加 静态资源就无法成功,所以,我们需要单独处理下静态资源!

处理方案: 在springmvc的配置文件中添加mvc命名空间下的标签!

1. 修改Spring MVC对应配置文件,添加mvc命名空间和约束

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

2. 添加处理标签

```
 <mvc:annotation-driven /> <!--注解驱动-->
 <mvc:resources mapping="/img/**" location="/images/" ></mvc:resources>
```

方式2:

```
<mvc:default-servlet-handler></mvc:default-servlet-handler>
```

16.Json处理

(1)添加jar包

```
<dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.9.5</version>
  </dependency>
```

注意:maven引入jackson-databind会连带引入 core和annotations

非maven项目需要引入这三个包

(2)实现代码:@ResponseBody

注意:需要在配置文件添加 <mvc:annotation-driven/>

17.SpringMVC拦截器

(1)创建拦截器类:实现HandlerInterceptor接口

preHandle() 拦截器开始

postHandle() 拦截器结束

afterCompletion 最后执行

(2)配置拦截器

拦截所有请求（测试:拦截内容包含jsp吗？）

```
  <mvc:interceptors>
          <bean id="my" class="util.MyInterceptor"/>
    </mvc:interceptors
```

拦截指定请求:

```
  <mvc:interceptors>
       <mvc:interceptor >
           <mvc:mapping path="/请求名" />
           <mvc:mapping path="/请求名" />
           <bean id="my" class="util.MyInterceptor"/>
       </mvc:interceptor>
    </mvc:interceptors>
```

补充:

springMVC拦截器使用场景

1、日志记录： 记录请求信息的日志 2、权限检查，如登录检查 3、性能检测：检测方法的执行时间

SpringMVC的拦截器（Interceptor）和过滤器（Filter）的区别与联系

一 简介

（1）过滤器：

依赖于servlet容器。在实现上基于函数回调，可以对几乎所有请求进行过滤，但是缺点是一个过滤器实例只能在容器初始化时调用一次。使用过滤器的目的是用来做一些过滤操作，获取我们想要获取的数据，比如：在过滤器中修改字符编码；在过滤器中修改HttpServletRequest的一些参数，包括：过滤低俗文字、危险字符等

（2）拦截器：

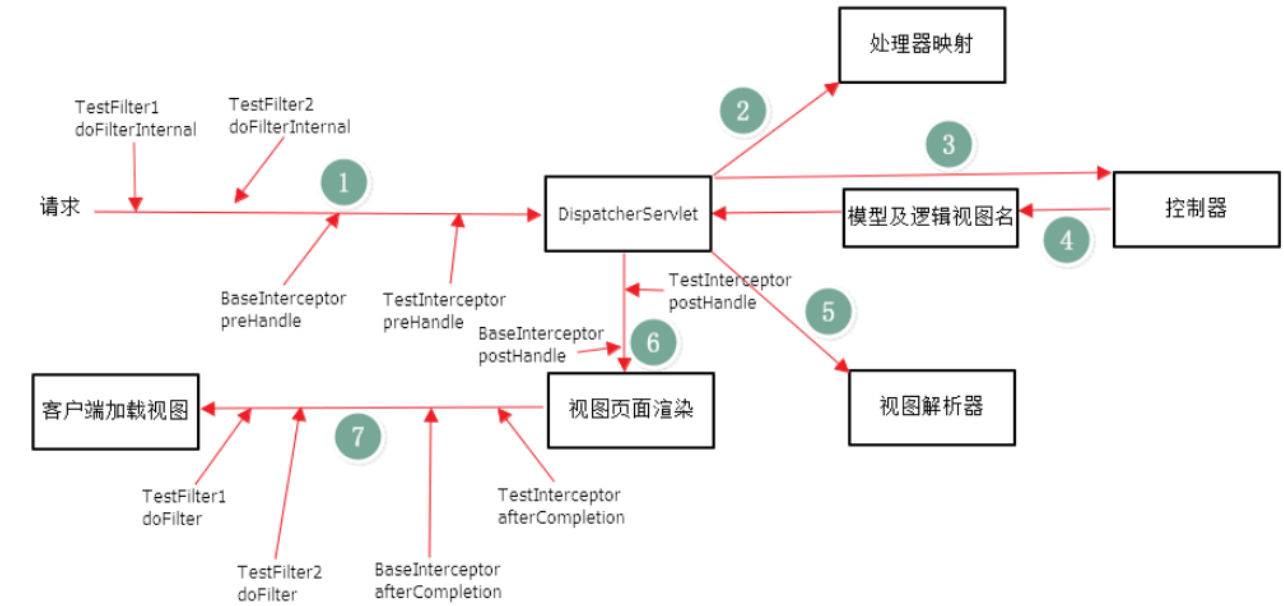
依赖于web框架，在SpringMVC中就是依赖于SpringMVC框架。在实现上基于Java的反射机制，属于面向切面编程（AOP）的一种运用。由于拦截器是基于web框架的调用，因此可以使用Spring的依赖注入（DI）进行一些业务操作，同时一个拦截器实例在一个controller生命周期之内可以多次调用。但是缺点是只能对controller请求进行拦截，对其他的一些比如直接访问静态资源的请求则没办法进行拦截处理

二 多个过滤器与拦截器的代码执行顺序

(1)过滤器的运行是依赖于servlet容器的，跟springmvc等框架并没有关系。并且，多个过滤器的执行顺序跟xml文件中定义的先后关系有关

(2)对于多个拦截器它们之间的执行顺序跟在SpringMVC的配置文件中定义的先后顺序有关

最终顺序:



18.文件上传下载

Spring MVC为文件上传提供了直接支持,这种支持是通过即插即用的MultipartResolver实现.

Spring使用Jakarta Commons FileUpload技术实现了一个MultipartResolver实现类:CommonsMultipartResolver。

在SpringMVC上下文中默认没有装配MultipartResolver,因此默认情况下不能处理文件上传工作。

如果想使用Spring的文件上传功能,则需要先在上下文中配置MultipartResolver。

fileUpload

文件上传的步骤:

（1）添加jar包

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>
```

(2)配置MultipartResolver:

```
  <mvc:annotation-driven/>
    <bean id="multipartResolver"
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
      p:defaultEncoding="UTF-8"
      p:maxUploadSize="5242880"
    />
```

（3）页面表单,提交方式必须是post

```
enctype="multipart/form-data"
```

（4）配置java代码(注意要创建文件夹保存上传之后的文件)

```
@RequestMapping("/upload")
 public  String saveFile(@RequestParam("name") String name , @RequestParam("file")MultipartFile
file) throws IOException {
    //接收表单提交的数据,包含文件
    System.out.println("name = " + name);
    String path=request.getRealPath("/");
    if (!file.isEmpty())
     {
       file.transferTo(new File(path+"upload/"+file.getOriginalFilename()));
     }
   return "success";
 }
```

方法名称	方法解释
byte [] getBytes()	获取文件数据
String getContentType()	获取文件MIMETYPE类型,如image/jpeg,text/plain等
InputStream getInputStream()	获取文件输入流
String getName()	获取表单中文件组件的名称 name值
String getOriginalFilename()	获取文件上传的原名
long getSize()	获取文件的字节大小,单位为byte
boolean isEmpty()	是否有长传的文件
void transferTo(File dest)	可以将上传的文件保存到指定的文件中

文件下载步骤:

(1）添加jar包

```
 <dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
  </dependency>
```

(2)配置处理类方法

```
   @RequestMapping("down")
        public ResponseEntity<byte[]> test(String imgname,HttpServletRequest request) throws
IOException{
                String serverpath= request.getRealPath("/img");
                serverpath=serverpath+"/"+imgname;
                //创建http头信息的对象
                HttpHeaders header=new HttpHeaders();
                //标记以流的方式做出响应
                header.setContentType(MediaType.APPLICATION_OCTET_STREAM);
            //设置以弹窗的方式提示用户下载
//attachment 表示以附件的形式响应给客户端
header.setContentDispositionFormData("attachment",URLEncoder.encode(imgname,"utf-8"));
                File f=new File(serverpath);
                ResponseEntity<byte[]> resp=
                                new ResponseEntity<byte[]>
(FileUtils.readFileToByteArray(f), header, HttpStatus.CREATED);
                return resp;
        }
```