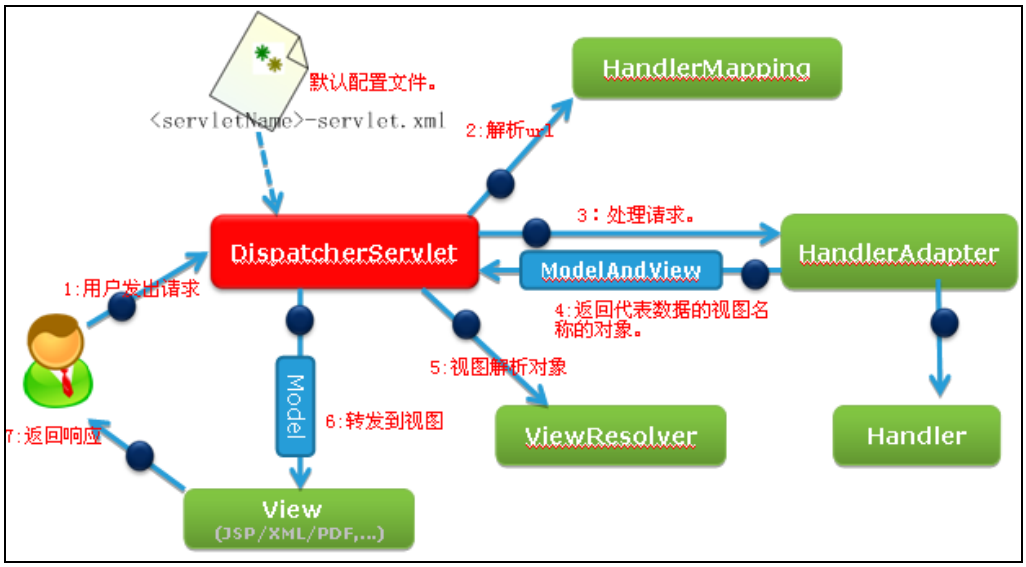
SpringMVC

王健（北京传智播客）

1、SpringMVC 简介

- Spring MVC 的核心组件：DispatcherServlet，它的工作是将不同的请求分发到不同的处理器。
- Spring 的 Web 框架包括可以配置的：处理器(Handler)映射，视图(Viewer)解析，本地(Local)化解析及主题和对文件上传的支持。
- SpringWeb 框架中，默认的控制器为 Controller 接口。它只包含 ModelAndView handlerRequest(request,response)方法，可以通过这个接口，实现自己的控制器。此接口同时拥有一些其他的子接口。
- Spring 允许使用任何对象做为命令对象和表单对象。
- 以下是 DispatcherServlet 的工作过程：



1.1、DispatcherServlet 详解

- DispatcherServlet 是 SpringMVC 的核心类，它就是一个 Servlet. 此类默认读取 WEB－INF/[servlet-name]-servlet.xml 配置文件。
- 重要说明：DispatcherServlet 可以多次配置到 web.xml 中，它们将会读取自己的配置文件，但不会共享 Bean.如果希望设置一些共享的 Bean 应该配置 ContextLoaderListener 来读取 Spring 的配置文件。
- DispatcherServlet 的配置如下：

1.2、HandlerMapping 介绍

HandlerMapping 用于在用户的请求(request)与控制器(Controller)之间定义一个路径(mapping)，即定义一种方式的方式。

在一个项目中，可以配置多个 HandlerMapping。

经常使用的它的两个实现子类为：

org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping :此类根据 Bean 的名称执行某个 Controller。

批注 [W1]: Spring 的核心组件。

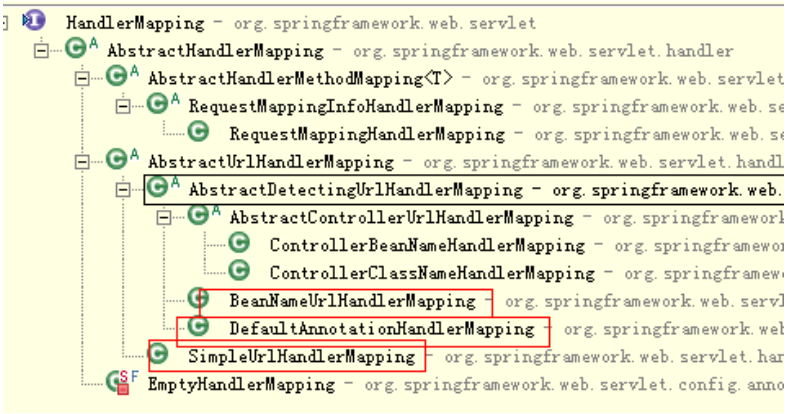
批注 [W2]: Spring 的控制接器接

批注 [W3]: Servlet-name 即配置的<servlet-name>元素的值。

批注 [W4]: 可以配置多个。

org.springframework.web.servlet.handler.SimpleUrlHandlerMapping： 此类接收一个 map 类型的属性，需要开发者一一指定某个 url 对应的控制器(Controller)。

下图展示了 HandlerMapping 的继承关系：最上层是 HandlerMapping 接口：



1.3、Controller 介绍

Spring 中的 Controller 类似于 Struts2 中的 Action。负责接收用户的请求，封装数据，返回响应。

以下是官方 API：

Base Controller interface, representing a component that receives HttpServletRequest and HttpServletResponse instances just like a HttpServlet but is able to participate in an MVC workflow. Controllers are comparable to the notion of a Struts Action.

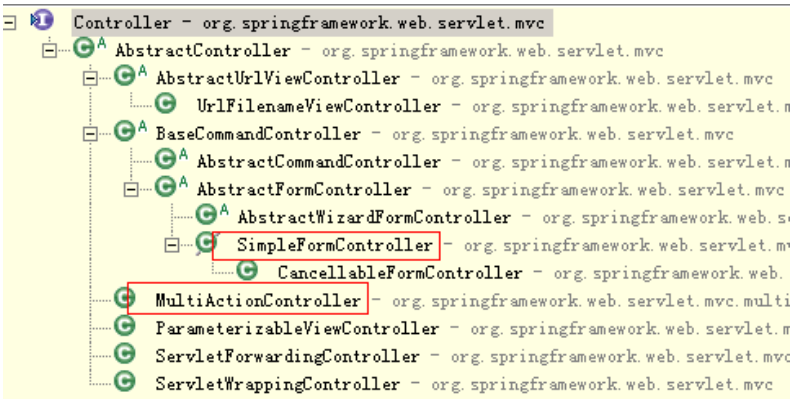
此类只有一个方法，它的源代码如下：

```
public interface Controller {
    ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
    throws Exception;
}
```

用配置文件的方式实现的控制器，必须是Controller的子类。

批注 [W5]: 必须是 Controller 的子类。

Controller的继承关系如下：

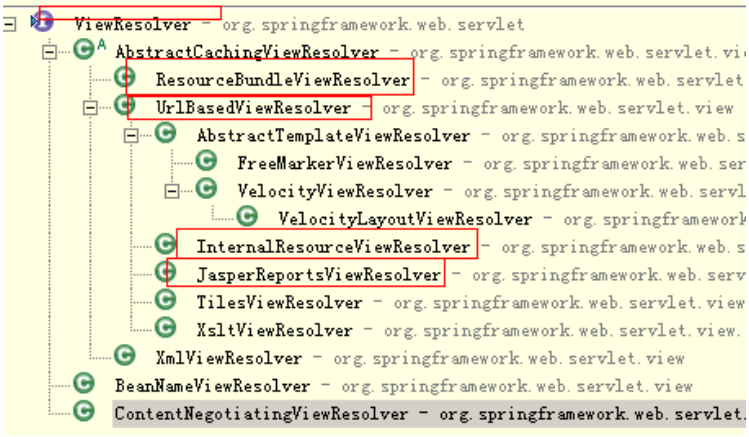


1、子类 MultiActionController

此类可以处理用户不同的请求。

1.4、ViewResolver – 视图解析器介绍

视图解析器的作用就是根据一个名称，解析需要显示的jsp或是pdf或是其他视图。



1.5、ModelAndView 介绍

ViewResolver就是根据ModelAndView中的name值，决定显示哪一个具体资源。ModelAndView有两个作用：

1：用于设置资源的名称。

ModelAndView mv = new ModelAndView(“someName”);

2：用于封装数据到Request中去。

ModelAndView mv = new ModelAndView(“someName”,”someKey”,Object someValue);

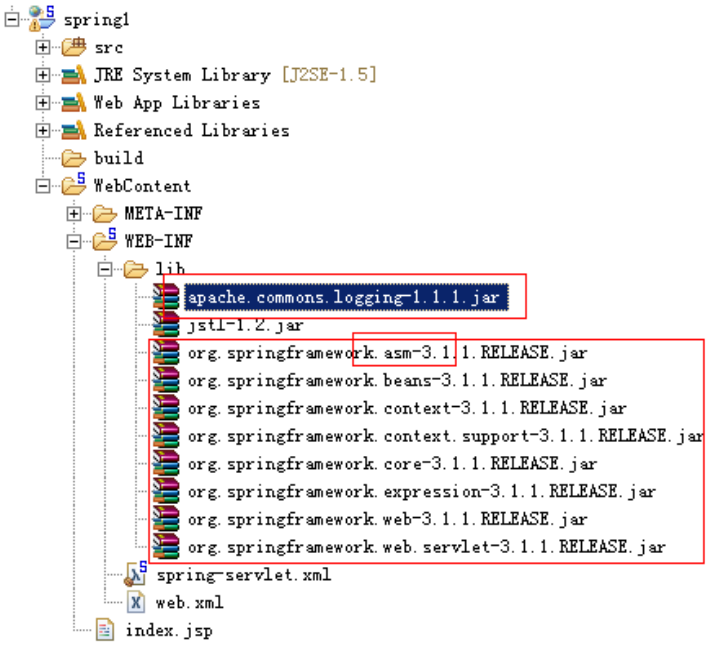
2、快速入门

在开发 SpringMVC 之前，我建议使用 Spring 的 STS 开发 Spring 的应用：

第一步：创建一个 web 项目，并导入以下 Spring 的 jar 文件

Commons-logging.jar 是 spring 依赖的第三方 jar 包我们也需要导入。

Asm 包是字节码操作包



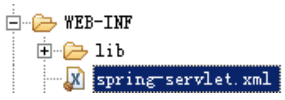
第二步：在 web.xml 中配置 DispatcherServlet 类如下

```
<servlet>
     <servlet-name>spring</servlet-name>
     <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
     </servlet-class>
     <init-param>
       <param-name>namespace</param-name>
       <!-- 默认即是下面的名称 -->
       <param-value>spring-servlet</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.sp</url-pattern>
</servlet-mapping>
```

批注 [W6]: 可选的配置 namespac

批注 [W7]: 设置访问 Spring 的后为.sp。 可以是任意值。这个值会响到后面的 bean 的定义。

第三步：在 WEB-INF 目录下添加 spring-servlet.xml 文件



内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
    <!-- 空文件 -->
</beans>
```

[至此可以发布一下项目，如果可以正常访问到主页，则可以继续下一步。]

第四步：创建一个控制器类，实现 Controller 接口

```
package cn.itcast.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
/**
 * 控制器类
 */
public class OneController implements Controller{
    public ModelAndView handleRequest(HttpServletRequest req,
            HttpServletResponse resp) throws Exception {
        resp.setContentType("text/html;charset=UTF-8");
        resp.getWriter().print("你好，这是第一个控制器");
        return null;
    }
}
```

批注 [W8]: 实现 Controller 接口。

批注 [W9]: 我还没有设置返回值。

第五步：在 spring-servlet.xml 中配置这个控制器

```
    <bean id="oneController" class="cn.itcast.controller.OneController"/>
```

第六步：在 spring-servlet.xml 中配置 url 解析对象-HandlerMapping

HandlerMapping 主要的工作就是如何让用户的请求找到上面的控制器，最基本的一个 HandlerMapping 就是 SimpleUrlHandlerMapping:

我们先配置 SimpleUrlHandlerMapping，此类可以配置 urlMap(AntPathMatcher 类型)和 mapping（经典类型）的两个 url 映射配置：

在 spring-servlet.xml 中配置 SimpleUrlHandlerMapping 如下：（以下配置多种 url 都指向同一个 Controller。）

```
<beans>
    <bean id="handlerMapping"
        class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <!-- 是否使用全路径，即contextPath,默认值为false,可以省略 -->
        <property name="alwaysUseFullPath" value="true"/>
        <!-- 路径的url编码方式 ,可以省略-->
        <property name="urlDecode" value="true"></property>
        <!-- Ant-Path风格的路径配置,接收一个map类型
            ant风格的可以使用通配符:
            ? : 表示一个任意的字符 /one?.sp可以是:onea.sp,oneb.sp等等
            * :  表示0或是一个任何字符/one*.sp，可以是one.sp,onea.sp
```

批注 [W10]: 声明 HandlerMap;ing 可以省略。

```
            **  ：表示任意多个目录
        -->
        <property name="urlMap">
            <map>
                <entry key="/one.sp" value-ref="oneController"></entry>
                <entry key="/a?.sp" value-ref="oneController"></entry>
                <entry key="/b*.sp" value-ref="oneController"></entry>
                <entry key="/**/c.sp" value-ref="oneController"></entry>
            </map>
        </property>
        <!-- 经典风格的配置，接收一个property类型 -->
        <property name="mappings">
            <props>
                <prop key="/two.sp">oneController</prop>
            </props>
        </property>
    </bean>
    <bean id="oneController" class="cn.itcast.controller.OneController"/>
</beans>
```

批注 [W11]: 设置 antpath 风格的路径。优选这种方式。

批注 [W12]: 设置经典风格的 url。支持通配符。

第七步：测试，同时测试通配符

代码到此，我可通过以下 url 测试访问：
http://server:port/project/one.sp
http://server:port/project/ax.sp - x 可以修改成任意的值
http://server:port/project/b.sp - b 后面可以为 0~N 个任意字符
http://server:port/project/xxxx/c.sp - xxx 可是 0~N 个任意字符。
http://server:port/project/two.sp

第八步：配置 ViewResolver（视图解析）

在上面的 OneController 当中，我们返回的是 null。而 ModelAndView 可以是一个具体的值，所以，我们可以返回 ModelAndView 的实例，而 ModelAndView 需要一个 ViewResolver 解析器。

最基本的 ViewResolver，就是 Inter..，它根据配置的路径前缀，的路径后缀，找到需要显示的 view 视图。

```
    <!-- 配置视图解析器 -->
    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/jsps/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
```

批注 [W13]: 路径的前缀

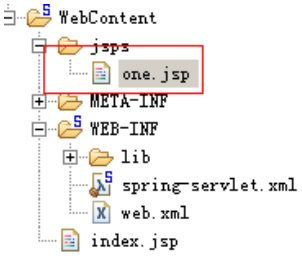
批注 [W14]: 路径的后缀。

第九步：修改 OneController 的代码如下

```
public class OneController implements Controller{
    public ModelAndView handleRequest(HttpServletRequest req,
            HttpServletResponse resp) throws Exception {
```

```
        System.err.println("OneController执行..."+this);
        //声明ModelAndView
        ModelAndView mv = new ModelAndView("one");
        //封装数据的ModelAndView,name是key,jack是value会放到request范围中
        mv = new ModelAndView("one","name","Jack");
        return mv;
    }
}
```

第十步：创建存放视图资源的目录，并创建一个名称为 one.jsp 的文件



One.jsp 的内容如下：

```
<body>
    <p>One.jsp</p>
    <hr/>
    <p>封装的数据是:<%=request%></p>
    ${requestScope.name}
</body>
```

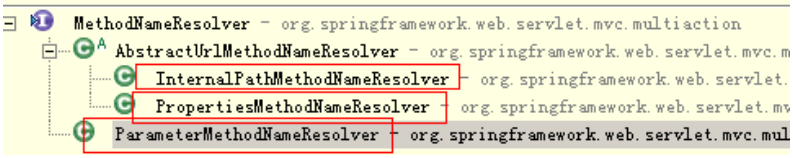
[到此运行测试就可以了]。

3、MultiActionController-处理多请求类型

此类用于处理多个不同的方法。它必须需要接收一个 MethodNameResolver。
此类是 Controller 的子类，通过 `ModelAndView handleRequestInternal()` 反射调用目标方法。
请查看它的源代码，你可以知道，此类是通过反射调用的。且方法中传递的参数可以是 4 个。

1、 MethodNameResolver 及其子类与 MultiActionController 协同工作

此类的某些子类，用于在运行时，给 MultiActionController 指定要调用的方法。



2、示例-ParameterMethodNameResolver

第一步：在配置文件中声明 ParameterMethodNameResolver

此方法用于在执行时确定方法。

```
<!-- 配置MethodNameResolver -->
    <bean
    id="paramterMethodNameResolver"
    class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
        <property name="defaultMethodName" value="execute"/>
        <!-- 配置好以后，通过?cmd=someMethod决定执行的方法 -->
        <property name="paramName" value="cmd"></property>
    </bean>
```

批注 [W15]: 默认执行的方法。

批注 [W16]: 决定执行的方法。

第二步：声明一个 Controller 继承 MultiActionController

```
package cn.itcast.multi;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.multiaction.MultiActionController;
public class OneMultiAction extends MultiActionController{
    /**
     * 实现第一个最基本的方法
     */
    public ModelAndView execute(HttpServletRequest req,HttpServletResponse resp)
                throws Exception{
        System.err.println("Hello.Multi..execute...");
        return new ModelAndView("two");
    }
}
```

第三步：配置此 Controller

```
<!-- 声明多方法处理类 -->
    <bean id="oneMulti" class="cn.itcast.multi.OneMultiAction">
        <property name="methodNameResolver" ref="paramterMethodNameResolver"/>
    </bean>
```

在 SimpleHandlerMapping 中配置这个 Controller

```
<property name="mappings">
            <props>
                <prop key="/two.sp">oneController</prop>
                <prop key="/onemulti.sp">oneMulti</prop>
            </props>
        </property>
```

第四步：测试运行

在地址栏输入以下测试：
http://localhost:8080/project/onemulti.sp

3、示例-PropertiesMethodNameResolver

PropertiesMethodNameResolver 通过配置的方式，执行不同的方法。
在 Spring 的配置文件中实现如下配置：

```
<!-- 以下演示ParameterMethodNameResolver处理不同的请求的访求 -->
<bean id="propertiesMethodNameResolver"
        class=
          "org.springframework.web.servlet.mvc.
          multiaction.PropertiesMethodNameResolver">
      <property name="mappings">
         <props>
            <!-- 与urlMap中的url对应，执行配置的方法 -->
            <prop key="/oneprop.sp">one</prop>
            <prop key="/twoprop.sp">two</prop>
         </props>
      </property>
   </bean>
   <!-- 声明一个与之对应的MultiActionController -->
   <bean id="propController" class="cn.itcast.controller.PropController">
      <property name="methodNameResolver" ref="propertiesMethodNameResolver"/>
   </bean>
```

配置对应的 url 访问，可选的通过 SimpleUrlHandlerMapping：

```
   <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
      <property name="urlMap">
         <map>
            <entry key="/tt.sp" value-ref="twoController"/>
            <entry key="/*prop.sp" value-ref="propController"></entry>
         </map>
      </property>
   </bean>
```

批注 [W17]: 用户访问的 url.前面对应 oneprop.sp。one 对应 one 方

propController 的源代码如下：

```
package cn.itcast.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.mvc.multiaction.MultiActionController;
public class PropController extends MultiActionController {
   public void one(HttpServletRequest req,HttpServletResponse resp) throws Exception{
      System.err.println("one....");
   }
   public void two(HttpServletRequest req,HttpServletResponse resp) throws Exception{
```

批注 [W18]: 这个方法对应 oneprop.sp 中的 one。

```
        System.err.println("two....");
    }
}
```

4、InternalPathMethodNameResolver

此类用于解析在路径中存在的参数做为方法名。一般情况下，参数名出现的 url 的最后部分，不包含 Servlet 的 url-pattern。
配置 InternalPathMethodNameResolver 如下：

```
<!-- 声明internalMethodNameResolver -->
    <bean id="internalPathMethodNameResolver"
         class=
              "org.springframework.web.servlet.mvc.
              multiaction.InternalPathMethodNameResolver">
         <!-- 这儿不需要声明任何参数 -->
    </bean>
    <!-- 声明自己的控制器类，并引用internalPathMethodNameResolver -->
    <bean id="interController" class="cn.itcast.controller.InterController">
       <property name="methodNameResolver" ref="internalPathMethodNameResolver"/>
    </bean>
```

在 SimpleUrlHandlerMapping 中配置 interController：

```
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="urlMap">
           <map>

              <entry key="/inter/*.sp" value-ref="interController"/>
           </map>
        </property>
    </bean>
```

批注 [W19]: 这个星就是需要动态用的方法名。我个人还是喜欢这种式。

在地址中输入这样的地址：
http://localhost:8080/project/inter/one.sp 将会调用 interController 类的 one 方法。
http://localhost:8080/project/inter/two.sp 将会调用 interController 类的 two 方法。

5、MultiActionController 方法说明

MultiActionController 子类的方法可以是以下的方法：（请参见此类的 API 文档。）

Controller implementation that allows multiple request types to be handled by the same class. Subclasses of this class can handle several different types of request with methods of the form

```
public (ModelAndView | Map | String | void) actionName(HttpServletRequest
request, HttpServletResponse response, [,HttpSession] [,AnyObject]);
```

批注 [W20]: 可选的这些返回值。

批注 [W21]: 可动态添加的方法。

1、添加 HttpSession 和 Bean 参数

```
    /**
     * 给方法设置更多的参数
     * 第一个参数必须是request
     * 第二个参数必须是request
     * 第三个参数可以是HttpSession或是JavaBean
     * 第四个参数如果有，必须是JavaBean
     */
    public ModelAndView
            one(HttpServletRequest req,HttpServletResponse resp,HttpSession s2,User user)
            throws Exception{
        System.err.println("s2 is "+s2+","+user);
        return new ModelAndView("two");
    }
```

批注 [W22]: 任意的方法名，接收个参数。

批注 [W23]: 后面的两个参数，可是 Session 或是 JavaBean。

使用以下访问方式：
http://localhost:8080/project/some.spring?cmd=one&beanProperty=value..

2、返回 void

返回 View 将不会转向任何页面。

3、返回 Map

返回一个 Map 时，刚会将整个 Map 放到 request 中再转发到默认页面。Spring 在这儿遵循了默认值原则：

```
/**
     * 使用internalPathMethodNameResolver
     * 并配置成/return/*.sp=returnController
     * 如果请求地址为:http://localhost:8080/project/return/one.sp
     * 则会请求one方法，默认转发到以下目录：
     * http://localhost:8080/project/prefix/return/one.sufix
     * 即：
     * http://localhost:8080/project/jsps/return/one.jsp
     */
    public Map<String,Object> one(HttpServletRequest req,HttpServletResponse resp)
             throws Exception{
        System.err.println("返回map。。。。。");
        Map<String,Object> mm = new HashMap<String, Object>();
        mm.put("name","Jack");
        mm.put("addr","中国北京");
        return mm;
    }
```

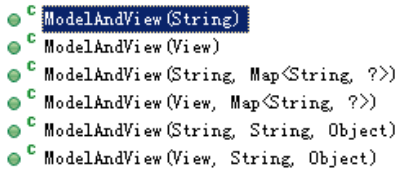
4、返回 String

返回 String 时，这个字符串，将指向 HandlerMapping 配置的资源：

```
/**
 * 如果配置了SimpleUrlHandlerMapping
 * 且配置了prefix = /jsps/
 * sufix = .jsp
 * 则默认会转向/jsps/two.jsp这个文件
 */
public String two(HttpServletRequest req,HttpServletResponse resp)
          throws Exception{
    System.err.println("返回String。。。。。");
    Map<String,Object> mm = new HashMap<String, Object>();
    mm.put("name","Rose");
    mm.put("addr","山东济南");
    req.setAttribute("mm",mm);
    return "two";
}
```

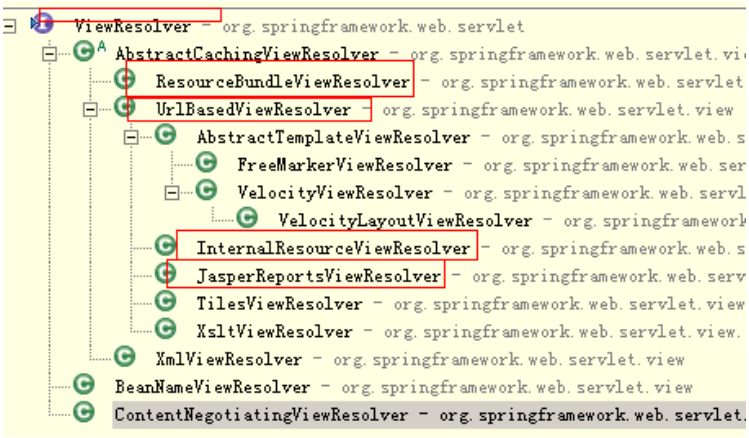
5、返回 ModelAndView

ModelAndView 的构造方法：



String ： 表示视图的名称。
Map ： 将自动被封装到 request 中。
String,Object ： 第二个 String 为 request 中的 key,Object 则为 request 中的 value。
View ： 确定为某一种视图，不适用。

4、ViewResolver 解析视图资源



比较常用的两个视图解析器为：

InternalResourceViewResolver

ResourceBundleViewResolver

1、InternalResourceViewResolver

此类提供路径的前缀和路径的后缀，根据返回的字符串视图名称，确定显示的资源。

此类是 UrlBaseViewResolver 类的子类。UrlBaseViewResolver 类中提供了 prefix 和 sufix 属性。

```
<bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="order" value="1"></property>
      <property name="prefix" value="/jsps/"/>
      <property name="suffix" value=".jsp"/>
   </bean>
```

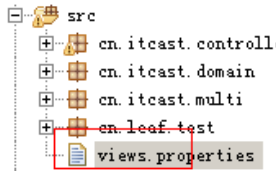
批注 [W24]: 配置前缀和后缀。

在配置好以后，当 handlerRequest 方法返回 one 时，将会显示/jsps/one.jsp 视图资源。

2、ResourceBoundViewResolver 从资源文件中加载视图资源

此类默认从一个资源文件加载字符串对应的资源名称。默认的资源文件为 classpath:views.properties。此资源文件也可以使用国际类型的资源文件。如 views_zh_CN.properties。

第一步：在 classpath 下创建一个 views.properties 文件




```
default.(class)=org.springframework.web.servlet.view.JstlView
default.contentType=text/html;charset=UTF-8
default.url=/jsps/default.jsp
```

```
other.url=/jsps/other.jsp
#设置它的重定向
three.(class)=org.springframework.web.servlet.view.RedirectView
#设置从上下方对象中开始找
three.contextRelative=true
three.url=/jsps/other.jsp
```

批注 [W25]: 如果 other 只设置 url，其他属性将从 default 中继承。

批注 [W26]: 设置 three 的重定向。

批注 [W27]: 从项目的根下开始。

第二步：在 Spring 文件中配置 ResourceBundleViewResolver

建议将它的 order 值调整的最高：

```
<bean id="resourceBoundleViewResolver"
        class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
      <property name="order" value="1"></property>
      <property name="basenames">
          <list>
              <value>views</value>
          </list>
      </property>
      <!-- 配置默认的父视图 -->
      <property name="defaultParentView" value="default"></property>
  </bean>
```

批注 [W28]: 默认在 classpath 下的件 views.properties。

第三步：在类中返回所定义的资源视图的 key 值即可

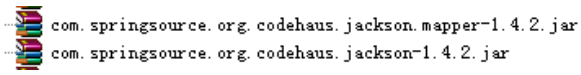
以下两个方法都返回 three 字符串，将会从 views.properties 文件中确定名称为 three 的 url。并向它重定向。

```
public String two(HttpServletRequest req,HttpServletResponse resp)
          throws Exception{
    System.err.println("返回String。。。。。");
    Map<String,Object> mm = new HashMap<String, Object>();
    mm.put("name","Rose");
    mm.put("addr","山东济南");
    req.setAttribute("mm",mm);
    return "three";
}

public ModelAndView four(HttpServletRequest req,HttpServletResponse resp)
        throws Exception{
    System.err.println("返回rrrrrr....。。。。");
    return new ModelAndView("three");
}
```

3、返回 JSON-MappingJacksonJsonView

MappingJacksonJsonView 默认用将 request 中的所有对象转成 json 字符串。
如果需要返回 json 我们需要添加两个 jar 包如下：



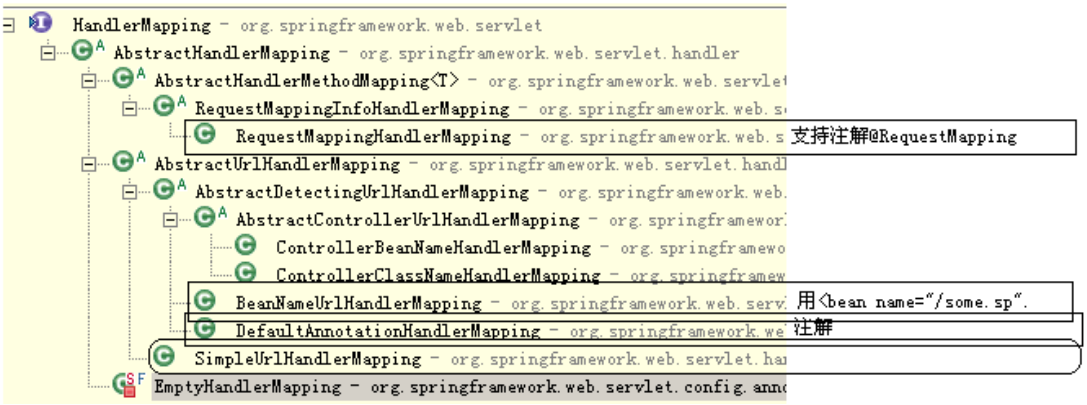
然后在 views.properties 文件中添加 MappingJacksonJsonView 的配置如下：

```
json.(class)=org.springframework.web.servlet.view.json.MappingJacksonJsonView
#设置读取request中的哪个属性
json.modelKey=mm
#此类不需要配置 url 或其他信息
```

将信息封装到 request 中然后返回：

```
public ModelAndView json(HttpServletRequest req, HttpServletResponse resp)
          throws Exception {
       System.err.println("json... method...");
       //secu为InternalResourceViewResolver配置的路径
       List<String> ll = new ArrayList<String>();
       ll.add("Jack");
       Map<String,String> mm = new HashMap<String, String>();
       mm.put("name","Jack");
       return new ModelAndView("json","mm",mm);
   }
```

5、HandlerMapping



1、SimpleUrlHandlerMapping

所有请求的资源 url 必须要通过 urlMap 或是 mappings 属性进行配置。使用不够灵活。对于某些特别重要的安全组件，可以这样配置。

```
<bean id="handlerMapping"
        class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
       <!-- 是否使用全路径，即contextPath,默认值为false,可以省略 -->
      <property name="alwaysUseFullPath" value="true"/>
      <!-- 路径的url编码方式 ,可以省略-->
      <property name="urlDecode" value="true"></property>
      <!-- Ant-Path风格的路径配置,接收一个map类型
```

```
            ant风格的可以使用通配符：
            ? ：表示一个任意的字符 /one?.sp可以是:onea.sp,oneb.sp等等
            * ：  表示0或是一个任何字符/one*.sp，可以是one.sp,onea.sp
            ** ：表示任意多个目录
        -->
        <property name="urlMap">
            <map>
                <entry key="/one.sp" value-ref="oneController"></entry>
                <entry key="/a?.sp" value-ref="oneController"></entry>
                <entry key="/b*.sp" value-ref="oneController"></entry>
                <entry key="/**/c.sp" value-ref="oneController"></entry>
            </map>
        </property>
        <!-- 经典风格的配置，接收一个property类型 -->
        <property name="mappings">
            <props>
                <prop key="/two.sp">oneController</prop>
                <prop key="/onemulti.sp">oneMulti</prop>
            </props>
        </property>
    </bean>
```

2、BeanNameUrlHandlerMapping

根据 bean 的名称，执行某个控制器。BeanNameUrlHandlerMapping 可以与 MultiActionController，ParameterMethodNameResolver，InternalPathMethodNameResolver 很好的整合。

使用非常灵活，但不能设置拦截器拦截某些特别的 url,还必须要借助于过虑器实现。

第一步：配置 BeanNameUrlHandlerMapping：

```
    <!-- 配置根据name解析Controller -->
    <bean name="beanNameUrlHandlerMapping"
          class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
        <property name="order" value="1"></property>
    </bean>
```

批注 [W29]: 配置它的优先级为 1

第二步：配置一个 Controller 接口的子类，此类实现 Controller 接口，

调用时将会执行 handleRequest(req,resp)方法

```
<!-- 配置第二个控制器 -->
    <bean name="/name1.sp" class="cn.itcast.name.Name1Controller"/>
```

调用 name1.sp 的 url 为：Http://localhost:8080/project/name1.sp

第三步：配置 MultiActionController 类的子类：（以下分别使用了两种方法解析技术，都是可以的）

```
<!-- 配置多请求控制器,以下使用的是internalPathMethodNameResolver
    访问abc方法的url：http://<local>:<port>/project/name2/abc.sp
    访问hello方法的url:http://<local>:<port>/project/name2/hello.sp
-->
<bean name="/name2/*.sp" class="cn.itcast.name.Name2Controller">
    <property name="methodNameResolver" ref="internalPathMethodNameResolver"/>
</bean>
<!-- 配置同一个控制器,使用parameterMethodNameRosolver
    访问abc方法的url：http://<local>:<port>/project/name3.sp?cmd=abc
    访问hello方法的url:http://<local>:<port>/project/name3.sp?cmd=hello
-->
<bean name="/name3.sp" class="cn.itcast.name.Name2Controller">
    <property name="methodNameResolver" ref="paramterMethodNameResolver"/>
</bean>
```

批注 [W30]: *必须要在/后面。且
立存在。

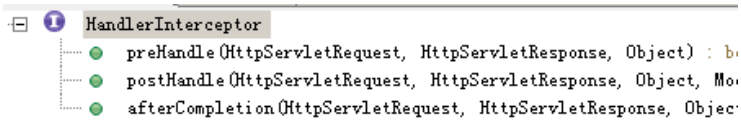
批注 [W31]: 通过
internalPathMethodNameResolver
析方法，我个人比较喜欢这种方

批注 [W32]: 通过 parameter..解析
用的方法，浪潮使用的这一种。

[测试即可]

6、拦截器（略）

在 Spring 中拦截器也是用于拦截目标请求的。但由于在 Spring 中已经有了 AOP，所以，拦截器作用被淡化了。拦截器类的最高接口为：



preHandler(..):boolean ： 请求之前拦截，如果返回 true 则放行，否则拒绝访问。
postHandle(..):void ： 目标方法执行完成以后拦截。
afterCompletion(..):void ： 所有请求都完成以后执行的方法。包括 Spring 的代码。
示例：

第一步：开发一个拦截器

```
public class OneInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        System.err.println("前拦截......");
        // 返回true|false决定是否向下执行
        return true;
    }
    public void postHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
        System.err.println("目标方法执行完成以后拦截...");
```

```
    }
    public void afterCompletion(HttpServletRequest request,
            HttpServletResponse response, Object handler, Exception ex)
            throws Exception {
        System.err.println("全部都执行完成以后。。。。");
    }
}
```

第二步：在某个 HandlerMapping 中引用这个拦截器：

```
<bean name="beanNameUrlHandlerMapping"
          class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
        <property name="order" value="1"></property>
        <property name="interceptors">
            <list>
                <bean class="cn.itcast.interceptor.OneInterceptor"></bean>
            </list>
        </property>
    </bean>
```

第三步：执行测试

(略)

7、文件上传

在 Spring 中处理文件上传。SpringMVC 为文件上传提供了直接的支持，这种支持是通过即插即用的 MultiPartResolver（接口）实现的。Spring 使用 Jakarta Commons FileUplad 技术实现了一个 MultipartResolver 类：CommonsMultipartResolver。

SpringMVC 默认情况下，并没有装配 MultipartResolver。因此默认情况下，不支持文件上传。如果希望处理文件上传，需要先配置 MultipartResolver。

如果使用继承 Controller 的方式实现文件上传则可以使用在 web 中学习的文件上传知识。

如果配置了 MultipartResolver 则应该使用注解方式实现文件上传。（后面讲）。

8、最佳组合

使用 InternalResourceViewResolver 处理/WEB-INF/下的资源。

使用 ResourceBundleViewResolver 处理外部资源。

使用 BeanNameUrlHandlerMapping 处理 url 请求。

使用 ParameterMethodNameResolver 处理动态方法调用，并配置默认方法。

使用 ContextLoaderListener 配置 spring 的配置文件。

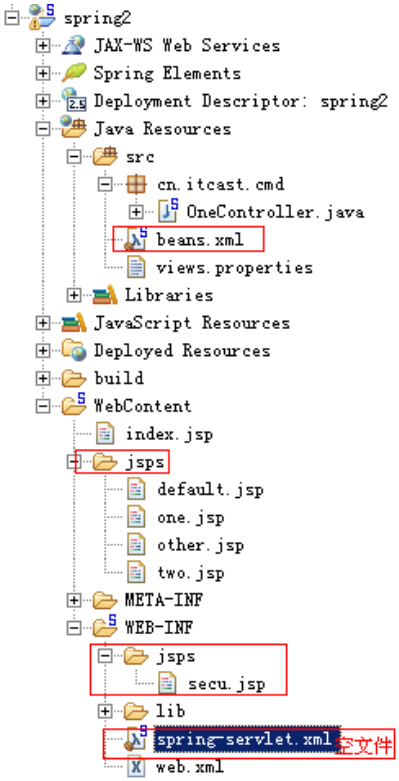
批注 [W33]: 两个资源解析。

批注 [W34]: 一个路径解析。

批注 [W35]: 一个参数名解析。

所有 Controller 都继承 MultiActionController。

第一步：创建以下项目结构



保持 spring-servlet.xml 是一个空文件，所有的配置在 classpath:beans.xml 中配置。

第二步：在 web.xml 中配置

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:beans.xml
    </param-value>
</context-param>
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>namespace</param-name>
        <param-value>spring-servlet</param-value>
    </init-param>
```

批注 [W36]: 配置需要加载的资源件。

批注 [W37]: 加载配置文件的监听

批注 [W38]: 空文件。

```
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/spring/*</url-pattern>
    </servlet-mapping>
```

第三步：在 classpath:beans.xml 中配置以下内容

```
    <!-- 配置内部路径解析 -->
    <bean id="interalResourceViewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- 先执行order=1的再执行order=2的 -->
        <property name="order" value="2"/>
        <property name="prefix" value="/WEB-INF/jsps/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
    <!-- 配置外部资源 -->
    <bean id="resourceBundleViewResolver"
          class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
        <property name="order" value="1"></property>
        <property name="basenames">
            <list>
                <value>views</value>
            </list>
        </property>
        <property name="defaultParentView" value="default"/>
    </bean>
    <!-- 配置路径解析 -->
    <bean id="beanNameUrlHandlerMapping"
          class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
        <property name="order" value="1"></property>
    </bean>
    <!-- 配置参数方法名解析 -->
    <bean id="parameterMethodNameResolver"
                  class="org.springframework.web.servlet.mvc
                         .multiaction.ParameterMethodNameResolver">
        <property name="defaultMethodName" value="execute"/>
        <property name="paramName" value="cmd"/>
    </bean>

    <!-- 配置一个测试类 -->
    <bean name="/one" class="cn.itcast.cmd.OneController">
        <property name="methodNameResolver" ref="parameterMethodNameResolver"/>
    </bean>
```

第四步：开发测试类

```
public class OneController extends MultiActionController {
    public ModelAndView one(HttpServletRequest req, HttpServletResponse resp)
```

批注 [W39]: 配置第一个视图资源解析器。

批注 [W40]: 配置第二个视图资源解析器。

批注 [W41]: url 解析就一个。

批注 [W42]: 配置一个控制器。

```
            throws Exception {
        System.err.println("One method...");
        //defalt为ResourceBundleViewResolver中配置的路径
        return new ModelAndView("default");
    }
    public String two(HttpServletRequest req, HttpServletResponse resp)
            throws Exception {
        System.err.println("Two method...");
        //secu为InternalResourceViewResolver配置的路径
        return "secu";
    }
}
```

测试：

http://localhost:8080/proj/spring/one?cmd=one

http://localhost:8080/proj/spring/one?cmd=two

(同上)

批注 [W43]: 这是 Controller。

批注 [W44]: 这是 Controller。

批注 [W45]: 决定需要执行的方法

9、用注解实现 SpringMVC

Spring 的注解，可以实现将 POJO 转成 Controller 的功能。除@Controller 注解以外，其他注解位于包：org.springframework.web.bind.annotation：

org.springframework.web.bind.annotation

Interfaces

ValueConstants

Enums

RequestMethod

Annotation Types

CookieValue
ExceptionHandler
InitBinder
Mapping
ModelAttribute
PathVariable
RequestBody
RequestHeader
RequestMapping
RequestParam
RequestPart
ResponseBody
ResponseStatus
SessionAttributes

类/注解	声明位置	作用
DefaultAnnotationHandlerMapping	Beans.xml	用于识别@Controller,@RequestMapping 等注解。
<context:component-scan/>	Beans.xml	设计读取哪些包中的注解
@Controller	POJO/Controller	标明某个类是控制器类
@RequestMapping	Method/Type	标明某个方法的请求 url 及请求方法,可以注解在方法上或是类上。
@RequestParam	参数上	参数绑定
@PathVariable	参数上	路径中的参数绑定

@CookieValue	参数	获取 cookie 的值
@RequestHeader	参数	获取请求头值

1、在 Spring 配置文件中加入 context 的命名空间

```
http://www.springframework.org/schema/context/spring-context-3.1.xsd
声明完成命名空间以后，再配置读取哪些包中的注解：
<!-- 配置读取哪些包 -->
<context:component-scan base-package="cn.itcast.anno"/>
```

2、在 spring 配置文件中声明 DefaultAnnotationHandlerMapping

```
<bean id="annotationHandlerMapper"
        class="org.springframework.web.servlet
               .mvc.annotation.DefaultAnnotationHandlerMapping">
        <property name="order" value="1"/>
</bean>
```

批注 [W46]: 可选配置。

3、使用 Controller 注解与@RequestMapping 注解声明 url

```
package cn.itcast.anno;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
@Scope(value="prototype")
@Controller
public class AnnoAController {
    @RequestMapping(value="/abc",method={RequestMethod.POST,RequestMethod.GET})
    public String abc(){
        System.err.println("Hello.....");
        return "three";
    }
}
```

批注 [W47]: 声明本 POJO 是一个
制器。

批注 [W48]: 声明 url.

批注 [W49]: 返回到 ResourceBun
所配置的资源名称上去。

在地址栏中访问：http://localhost:8080/project/spring/abc

批注 [W50]: 上面配置的路径。

4、通过@RequestParam 接收参数

@RequestParam 用于接收 get 和 post 类型的参数

```
//通过get/post方式请求：
//http://localhost:8080/proj/spring/a1?name=Jack&age=90
@RequestMapping(value="/a1")
public String a1(@RequestParam(value="name")String name,
```

```
                @RequestParam(value="age")Integer age){
        System.err.println("name is:"+name+",age is:"+age);
        return "default";
    }
```

批注 [W51]: 通过 RequestParam 接参数。

5、声明任意多个参数

以下方法都是合法的：

```
@RequestMapping(value="/a1")
public String a1(@RequestParam(value="name")String name,
                @RequestParam(value="age")Integer age){
        System.err.println("name is:"+name+",age is:"+age);
        return "default";
}
@RequestMapping(value="/a2")
public String a2(HttpServletRequest req){
        System.err.println("a2......................."+req);
        return "default";
}
@RequestMapping(value="/a3")
public String a3(HttpServletRequest req,HttpServletResponse resp){
        System.err.println("a3."+req+","+resp);
        return "default";
}
@RequestMapping(value="/a4")
public String a4(HttpServletRequest req,HttpServletResponse resp,HttpSession s){
        System.err.println("a4.."+req+","+resp+","+s);
        return "default";
}
@RequestMapping(value="/a5")
public String a5(HttpServletRequest req,HttpServletResponse resp,HttpSession s,User user){
        System.err.println("a5."+req+","+user);
        return "default";
}
@RequestMapping(value="/a6")
public String a6(User user){
        System.err.println("a56."+user);
        return "default";
}
//访问 url 为:http://localhost:8080/project/spring/a6?name=Jack&pwd=1234,即可以自动封装参数
```

批注 [W52]: 接收任意多个参数。

批注 [W53]: 只接收 request.

批注 [W54]: 接收的也是最全的。

批注 [W55]: 只接收 javaBean.

6、@PathVariable 路径中的参数-RESTful 类型参数绑定

http://localhost:8080/spring2/spring/a7/Jack/990

```
@RequestMapping(value="/a7/{name}/{age}")
public String a7(
```

批注 [W56]: 只能绑定基本类型的参数。不可以绑定到 JavaBean.

```
            @PathVariable("name")String name,
            @PathVariable("age")Integer age){
    System.err.println("a7:"+name+","+age);
    return "default";
}
```

7、返回 ModelAndView

```
/**
* 返回modelAndView对象
*/
@RequestMapping(value="/a8")
public ModelAndView a8(){
        System.err.println("a8:"+this);
        return new ModelAndView("two","mm","Value");
}
```

8、返回 void

按 Spring 的默认方式处理，如访问 http://localhost:8080/proj/spring/a9,则会去显示 proj/jsps/a9.jsp 页面。

9、接收 Model 对象

```
@RequestMapping(value="/a9")
public String a9(Model model){
    System.err.println("a9:"+model);
    model.addAttribute("mm","Some某些数据");//相当于将数据放到req中
    return "two";
}
```

10、读取 Cookie 和请求头信息

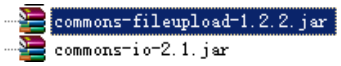
```
@RequestMapping(value="/a10")
public String a10(@CookieValue("JSESSIONID")String sid,
                  @RequestHeader(value="accept")String accept){
    System.err.println("a10:"+sid+","+accept);
    return "two";
}
```

批注 [W57]: 读取 cookie 中的某个值。

批注 [W58]: 读取请求头信息。

11、文件上传参数

文件上传需要导入 commons-fileuplad 包。



第一步：先在 beans.xml 中配置 MultipartResolver：

```
<!-- 配置对文件上传的支持 -->
    <bean id="multipartResolver"
          class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="defaultEncoding" value="UTF-8"/>
        <property name="maxUploadSize" value="#{1024*1024*100}"/>
    </bean>
```

批注 [W59]: Spring3.0 可以直接在 儿参与运算

第二步：声明方法接收 MultipartFile

```
    //声明访问的url
    @RequestMapping(value="/a11")
    public String a11(@RequestParam("img")MultipartFile file,
                      HttpSession s,
                      Model model) throws Exception{
        System.err.println("a11:"+file);
        String fileName = file.getOriginalFilename();
        fileName = fileName.substring(fileName.lastIndexOf("\\")+1);
        String path = s.getServletContext().getRealPath("/up");
        //保存文件到指定的目录
        file.transferTo(new File(path+"/"+fileName));
        //放到req中
        model.addAttribute("mm","文件名:"+fileName);
        return "two";
    }
```

批注 [W60]: 从参数中接收 img 这 字段的值。

12、上传多个文件

```
    @RequestMapping(value = "/a12")
    public String a12(@RequestParam("img") MultipartFile[] file, HttpSession s,
            Model model) throws Exception {
        String ns= "";
        for (MultipartFile f : file) {
            System.err.println("a11:" + file);
            String fileName = f.getOriginalFilename();
            fileName = fileName.substring(fileName.lastIndexOf("\\") + 1);
            String path = s.getServletContext().getRealPath("/up");
            // 保存文件到指定的目录
            f.transferTo(new File(path + "/" + fileName));
            ns+=fileName+",";
        }
```

批注 [W61]: 声明一个数组对象即 可。

```
        // 放到req中
        model.addAttribute("mm", "文件名:" + ns);
        return "two";
    }
```

表单：

```
    <form action="<c:url value='/spring/a12'/>" method="post" enctype="multipart/form-data">
        Img:<input type="file" name="img"/><br/>
        Img:<input type="file" name="img"/><br/>
        <input type="submit"/>
    </form>
```

10、Spring 提供的工具类

Spring 提供的工具类，如 CharacterSetEncodingFilter，Log4j….等：

1、org.springframework.web.filter.CharacterEncodingFilter

```
    <filter>
        <filter-name>char</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>char</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

2、WebApplicationContextUtils

当 Web 应用集成 Spring 容器后，代表 Spring 容器的 WebApplicationContext 对象将以 WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE 为键存放在 ServletContext 属性列表中。您当然可以直接通过以下语句获取 WebApplicationContext：

```
WebApplicationContext wac = (WebApplicationContext)servletContext.
            getAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE);
```

但通过位于 org.springframework.web.context.support 包中的 WebApplicationContextUtils 工具类获取 WebApplicationContext 更方便：

```
WebApplicationContext wac =WebApplicationContextUtils.
            getWebApplicationContext(servletContext);
```

11、深入 DispatcherServlet 内部的逻辑

以下是 DispatcherServlet 的部分源代码：

```
protected void initStrategies(ApplicationContext context) {
    initMultipartResolver(context);
    initLocaleResolver(context);
    initThemeResolver(context);
    initHandlerMappings(context);
    initHandlerAdapters(context);
    initHandlerExceptionResolvers(context);
    initRequestToViewNameTranslator(context);
    initViewResolvers(context);
    initFlashMapManager(context);
}
```