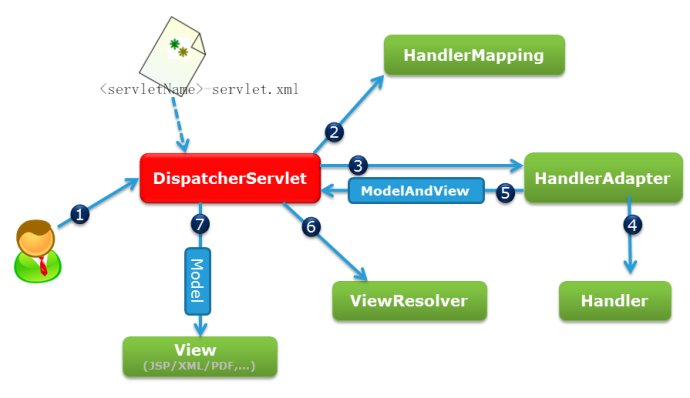
## [springMVC 的工作原理和机制](http://www.cnblogs.com/zbf1214/p/5265117.html)

2016-03-11 12:05 by 孤鸿子, 1967 阅读, 0 评论, [收藏](http://www.cnblogs.com/zbf1214/p/5265117.html), [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=5265117)



工作原理  
上面的是springMVC的工作原理图：

1、客户端发出一个http请求给web服务器，web服务器对http请求进行解析，如果匹配DispatcherServlet的请求映射路径（在web.xml中指定），web容器将请求转交给DispatcherServlet.

2、DipatcherServlet接收到这个请求之后将根据请求的信息（包括URL、Http方法、请求报文头和请求参数Cookie等）以及HandlerMapping的配置找到处理请求的处理器（Handler）。

3-4、DispatcherServlet根据HandlerMapping找到对应的Handler,将处理权交给Handler（Handler将具体的处理进行封装），再由具体的HandlerAdapter对Handler进行具体的调用。

5、Handler对数据处理完成以后将返回一个ModelAndView()对象给DispatcherServlet。

6、Handler返回的ModelAndView()只是一个逻辑视图并不是一个正式的视图，DispatcherSevlet通过ViewResolver将逻辑视图转化为真正的视图View。

7、Dispatcher通过model解析出ModelAndView()中的参数进行解析最终展现出完整的view并返回给客户端。

工作机制是什么

### **Control的调用（续）**

接着对于（二）的补充：主要是小结下Control的处理逻辑的关键操作；

对于control的处理关键就是：DispatcherServlet的handlerMappings集合中根据请求的URL匹配每一个handlerMapping对象中的某个handler,匹配成功之后将会返回这个handler的处理连接handlerExecutionChain对象。而这个handlerExecutionChain对象中将会包含用户自定义的多个handlerInterceptor对象。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | /\*\*       \* Return the HandlerExecutionChain for this request.       \* <p>Tries all handler mappings in order.       \* @param request current HTTP request       \* @return the HandlerExecutionChain, or <code>null</code> if no handler could be found       \*/      protected HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception {          for (HandlerMapping hm : this.handlerMappings) {              if (logger.isTraceEnabled()) {                  logger.trace(                          "Testing handler map [" + hm + "] in DispatcherServlet with name '" + getServletName() + "'");              }              HandlerExecutionChain handler = hm.getHandler(request);              if (handler != null) {                  return handler;              }          }          return null;      } |

而对于handlerInterceptor接口中定义的三个方法中，preHandler和postHandler分别在handler的执行前和执行后执行，afterCompletion在view渲染完成、在DispatcherServlet返回之前执行。

PS:这么我们需要注意的是：当preHandler返回false时，当前的请求将在执行完afterCompletion后直接返回,handler也将不会执行。

在类HandlerExecutionChain中的getHandler()方法是返回object对象的；

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | /\*\*       \* Return the handler object to execute.       \* @return the handler object       \*/      public Object getHandler() {          return this.handler;      } |

这里的handler是没有类型的，handler的类型是由handlerAdapter决定的。dispatcherServlet会根据handler对象在其handlerAdapters集合中匹配哪个HandlerAdapter实例支持该对象。接下来去执行handler对象的相应方法了，如果该handler对象的相应方法返回一个ModelAndView对象接下来就是去执行View渲染了。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | /\*\*       \* Return the handler object to execute.       \* @return the handler object       \*/      public Object getHandler() {          return this.handler;      } |

---------------------------------------邪恶的分割线---------------------------------------------

## **Model设计**

如果handler兑现返回了ModelAndView对象，那么说明Handler需要传一个Model实例给view去渲染模版。除了渲染页面需要model实例，在业务逻辑层通常也有Model实例。

ModelAndView对象是连接业务逻辑层与view展示层的桥梁，对spring MVC来说它也是连接Handler与view的桥梁。ModelAndView对象顾名思义会持有一个ModelMap对象和一个View对象或者View的名称。ModelMap对象就是执行模版渲染时候所需要的变量对应的实例，如jsp的通过request.getAttribute(String)获取的JSTL标签名对应的对象。velocity中context.get(String)获取$foo对应的变量实例。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public class ModelAndView {    /\*\* View instance or view name String \*/      private Object view;        /\*\* Model Map \*/      private ModelMap model;        /\*\* Indicates whether or not this instance has been cleared with a call to {@link #clear()} \*/      private boolean cleared = false;    .....    } |

ModelMap其实也是一个Map,Handler中将模版中需要的对象存在这个Map中，然后传递到view对应的ViewResolver中。

|  |  |
| --- | --- |
| 1  2  3  4 | public interface ViewResolver {      View resolveViewName(String viewName, Locale locale) throws Exception;    } |

不同的ViewResolver会对这个Map中的对象有不同的处理方式；

* velocity中将这个Map保存到VelocityContext中。
* JSP中将每一个ModelMap中的元素分别设置到request.setAttribute(modelName,modelValue);

-----------------------邪恶的分割线-----------------------------------------------

## **view设计**

在spring MVC中，view模块需要两个组件来支持：RequestToViewNameTranslator和ViewResolver

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public interface RequestToViewNameTranslator {        /\*\*       \* Translate the given {@link HttpServletRequest} into a view name.       \* @param request the incoming {@link HttpServletRequest} providing       \* the context from which a view name is to be resolved       \* @return the view name (or <code>null</code> if no default found)       \* @throws Exception if view name translation fails       \*/      String getViewName(HttpServletRequest request) throws Exception;    } |

对于 ViewResolver，前面有写到了，就不写了；

-----------------------邪恶的分割线-------------------------------------------------

RequestToViewNameTranslator：主要支持用户自定义对viewName的解析，如将请求的ViewName加上前缀或者后缀，或者替换成特定的字符串等。

ViewResolver：主要是根据用户请求的viewName创建适合的模版引擎来渲染最终的页面，ViewResolver会根据viewName创建一个view对象，调用view对象的Void render方法渲染出页面；

|  |  |
| --- | --- |
| 1  2  3 | public interface View {  void render(Map<String, ?> model, HttpServletRequest request, HttpServletResponse response) throws Exception;  } |

下面来总结下 Spring MVC解析View的逻辑：

* dispatcherServlet方法调用getDefaultViewName（）方法；

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /\*\*       \* Translate the supplied request into a default view name.       \* @param request current HTTP servlet request       \* @return the view name (or <code>null</code> if no default found)       \* @throws Exception if view name translation failed       \*/      protected String getDefaultViewName(HttpServletRequest request) throws Exception {          return this.viewNameTranslator.getViewName(request);      } |

* 调用了RequestToViewNameTranslator的getViewName方法；

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public interface RequestToViewNameTranslator {        /\*\*       \* Translate the given {@link HttpServletRequest} into a view name.       \* @param request the incoming {@link HttpServletRequest} providing       \* the context from which a view name is to be resolved       \* @return the view name (or <code>null</code> if no default found)       \* @throws Exception if view name translation fails       \*/      String getViewName(HttpServletRequest request) throws Exception;    } |

* 调用LocaleResolver接口的resolveLocale方法；

|  |  |
| --- | --- |
| 1 | Locale resolveLocale(HttpServletRequest request); |

* 调用ViewResolver接口的resolveViewName方法，返回view对象

|  |  |
| --- | --- |
| 1 | View resolveViewName(String viewName, Locale locale) throws Exception; |

* 调用render方法渲染出页面