<http://blog.csdn.net/tongxinxiao/article/details/39502519>

**【写在前面】**：很久没写博客了，从毕业到web开发快1个月了，接触的比较多的当然是jsp、js和Spring。其中对ModelAndView印象深刻，使用频繁，特辑之。

**0、【前言】**：使用ModelAndView类用来存储处理完后的结果数据，以及显示该数据的视图。从名字上看ModelAndView中的Model代表模型，View代表视图，这个名字就很好地解释了该类的作用。业务处理器调用模型层处理完用户请求后，把结果数据存储在该类的model属性中，把要返回的视图信息存储在该类的view属性中，然后让该ModelAndView返回该Spring MVC框架。框架通过调用配置文件中定义的视图解析器，对该对象进行解析，最后把结果数据显示在指定的页面上。

**1、【其源码】**：熟悉一个类的用法，最好从其源码入手。

**[java]** [view plain](http://blog.csdn.net/tongxinxiao/article/details/39502519) [copy](http://blog.csdn.net/tongxinxiao/article/details/39502519)

1. **public** **class** ModelAndView {
3. /\*\* View instance or view name String \*/
4. **private** Object view;//<span style="color: rgb(0, 130, 0); font-family: Consolas, 'Courier New', Courier, mono, serif; line-height: 18px;">该属性用来存储返回的视图信息</span>

**[java]** [view plain](http://blog.csdn.net/tongxinxiao/article/details/39502519) [copy](http://blog.csdn.net/tongxinxiao/article/details/39502519)

1. /\*\* Model Map \*/
2. **private** ModelMap model;//<span style="color: rgb(0, 130, 0); font-family: Consolas, 'Courier New', Courier, mono, serif; line-height: 18px;">该属性用来存储处理后的结果数据</span>
4. /\*\*
5. \* Indicates whether or not this instance has been cleared with a call to {@link #clear()}.
6. \*/
7. **private** **boolean** cleared = **false**;

10. /\*\*
11. \* Default constructor for bean-style usage: populating bean
12. \* properties instead of passing in constructor arguments.
13. \* @see #setView(View)
14. \* @see #setViewName(String)
15. \*/
16. **public** ModelAndView() {
17. }
19. /\*\*
20. \* Convenient constructor when there is no model data to expose.
21. \* Can also be used in conjunction with <code>addObject</code>.
22. \* @param viewName name of the View to render, to be resolved
23. \* by the DispatcherServlet's ViewResolver
24. \* @see #addObject
25. \*/
26. **public** ModelAndView(String viewName) {
27. **this**.view = viewName;
28. }
30. /\*\*
31. \* Convenient constructor when there is no model data to expose.
32. \* Can also be used in conjunction with <code>addObject</code>.
33. \* @param view View object to render
34. \* @see #addObject
35. \*/
36. **public** ModelAndView(View view) {
37. **this**.view = view;
38. }
40. /\*\*
41. \* Creates new ModelAndView given a view name and a model.
42. \* @param viewName name of the View to render, to be resolved
43. \* by the DispatcherServlet's ViewResolver
44. \* @param model Map of model names (Strings) to model objects
45. \* (Objects). Model entries may not be <code>null</code>, but the
46. \* model Map may be <code>null</code> if there is no model data.
47. \*/
48. **public** ModelAndView(String viewName, Map<String, ?> model) {
49. **this**.view = viewName;
50. **if** (model != **null**) {
51. getModelMap().addAllAttributes(model);
52. }
53. }
55. /\*\*
56. \* Creates new ModelAndView given a View object and a model.
57. \* <emphasis>Note: the supplied model data is copied into the internal
58. \* storage of this class. You should not consider to modify the supplied
59. \* Map after supplying it to this class</emphasis>
60. \* @param view View object to render
61. \* @param model Map of model names (Strings) to model objects
62. \* (Objects). Model entries may not be <code>null</code>, but the
63. \* model Map may be <code>null</code> if there is no model data.
64. \*/
65. **public** ModelAndView(View view, Map<String, ?> model) {
66. **this**.view = view;
67. **if** (model != **null**) {
68. getModelMap().addAllAttributes(model);
69. }
70. }
72. /\*\*
73. \* Convenient constructor to take a single model object.
74. \* @param viewName name of the View to render, to be resolved
75. \* by the DispatcherServlet's ViewResolver
76. \* @param modelName name of the single entry in the model
77. \* @param modelObject the single model object
78. \*/
79. **public** ModelAndView(String viewName, String modelName, Object modelObject) {
80. **this**.view = viewName;
81. addObject(modelName, modelObject);
82. }
84. /\*\*
85. \* Convenient constructor to take a single model object.
86. \* @param view View object to render
87. \* @param modelName name of the single entry in the model
88. \* @param modelObject the single model object
89. \*/
90. **public** ModelAndView(View view, String modelName, Object modelObject) {
91. **this**.view = view;
92. addObject(modelName, modelObject);
93. }

96. /\*\*
97. \* Set a view name for this ModelAndView, to be resolved by the
98. \* DispatcherServlet via a ViewResolver. Will override any
99. \* pre-existing view name or View.
100. \*/
101. **public** **void** setViewName(String viewName) {
102. **this**.view = viewName;
103. }
105. /\*\*
106. \* Return the view name to be resolved by the DispatcherServlet
107. \* via a ViewResolver, or <code>null</code> if we are using a View object.
108. \*/
109. **public** String getViewName() {
110. **return** (**this**.view **instanceof** String ? (String) **this**.view : **null**);
111. }
113. /\*\*
114. \* Set a View object for this ModelAndView. Will override any
115. \* pre-existing view name or View.
116. \*/
117. **public** **void** setView(View view) {
118. **this**.view = view;
119. }
121. /\*\*
122. \* Return the View object, or <code>null</code> if we are using a view name
123. \* to be resolved by the DispatcherServlet via a ViewResolver.
124. \*/
125. **public** View getView() {
126. **return** (**this**.view **instanceof** View ? (View) **this**.view : **null**);
127. }
129. /\*\*
130. \* Indicate whether or not this <code>ModelAndView</code> has a view, either
131. \* as a view name or as a direct {@link View} instance.
132. \*/
133. **public** **boolean** hasView() {
134. **return** (**this**.view != **null**);
135. }
137. /\*\*
138. \* Return whether we use a view reference, i.e. <code>true</code>
139. \* if the view has been specified via a name to be resolved by the
140. \* DispatcherServlet via a ViewResolver.
141. \*/
142. **public** **boolean** isReference() {
143. **return** (**this**.view **instanceof** String);
144. }
146. /\*\*
147. \* Return the model map. May return <code>null</code>.
148. \* Called by DispatcherServlet for evaluation of the model.
149. \*/
150. **protected** Map<String, Object> getModelInternal() {
151. **return** **this**.model;
152. }
154. /\*\*
155. \* Return the underlying <code>ModelMap</code> instance (never <code>null</code>).
156. \*/
157. **public** ModelMap getModelMap() {
158. **if** (**this**.model == **null**) {
159. **this**.model = **new** ModelMap();
160. }
161. **return** **this**.model;
162. }
164. /\*\*
165. \* Return the model map. Never returns <code>null</code>.
166. \* To be called by application code for modifying the model.
167. \*/
168. **public** Map<String, Object> getModel() {
169. **return** getModelMap();
170. }

173. /\*\*
174. \* Add an attribute to the model.
175. \* @param attributeName name of the object to add to the model
176. \* @param attributeValue object to add to the model (never <code>null</code>)
177. \* @see ModelMap#addAttribute(String, Object)
178. \* @see #getModelMap()
179. \*/
180. **public** ModelAndView addObject(String attributeName, Object attributeValue) {
181. getModelMap().addAttribute(attributeName, attributeValue);
182. **return** **this**;
183. }
185. /\*\*
186. \* Add an attribute to the model using parameter name generation.
187. \* @param attributeValue the object to add to the model (never <code>null</code>)
188. \* @see ModelMap#addAttribute(Object)
189. \* @see #getModelMap()
190. \*/
191. **public** ModelAndView addObject(Object attributeValue) {
192. getModelMap().addAttribute(attributeValue);
193. **return** **this**;
194. }
196. /\*\*
197. \* Add all attributes contained in the provided Map to the model.
198. \* @param modelMap a Map of attributeName -> attributeValue pairs
199. \* @see ModelMap#addAllAttributes(Map)
200. \* @see #getModelMap()
201. \*/
202. **public** ModelAndView addAllObjects(Map<String, ?> modelMap) {
203. getModelMap().addAllAttributes(modelMap);
204. **return** **this**;
205. }

208. /\*\*
209. \* Clear the state of this ModelAndView object.
210. \* The object will be empty afterwards.
211. \* <p>Can be used to suppress rendering of a given ModelAndView object
212. \* in the <code>postHandle</code> method of a HandlerInterceptor.
213. \* @see #isEmpty()
214. \* @see HandlerInterceptor#postHandle
215. \*/
216. **public** **void** clear() {
217. **this**.view = **null**;
218. **this**.model = **null**;
219. **this**.cleared = **true**;
220. }
222. /\*\*
223. \* Return whether this ModelAndView object is empty,
224. \* i.e. whether it does not hold any view and does not contain a model.
225. \*/
226. **public** **boolean** isEmpty() {
227. **return** (**this**.view == **null** && CollectionUtils.isEmpty(**this**.model));
228. }
230. /\*\*
231. \* Return whether this ModelAndView object is empty as a result of a call to {@link #clear}
232. \* i.e. whether it does not hold any view and does not contain a model.
233. \* <p>Returns <code>false</code> if any additional state was added to the instance
234. \* <strong>after</strong> the call to {@link #clear}.
235. \* @see #clear()
236. \*/
237. **public** **boolean** wasCleared() {
238. **return** (**this**.cleared && isEmpty());
239. }

242. /\*\*
243. \* Return diagnostic information about this model and view.
244. \*/
245. @Override
246. **public** String toString() {
247. StringBuilder sb = **new** StringBuilder("ModelAndView: ");
248. **if** (isReference()) {
249. sb.append("reference to view with name '").append(**this**.view).append("'");
250. }
251. **else** {
252. sb.append("materialized View is [").append(**this**.view).append(']');
253. }
254. sb.append("; model is ").append(**this**.model);
255. **return** sb.toString();
256. }

    在源码中有7个构造函数，如何用？是一个重点。构造ModelAndView对象当控制器处理完请求时，通常会将包含视图名称或视图对象以及一些模型属性的ModelAndView对象返回到DispatcherServlet。因此，经常需要在控制器中构造ModelAndView对象。ModelAndView类提供了几个重载的构造器和一些方便的方法，让你可以根据自己的喜好来构造ModelAndView对象。这些构造器和方法以类似的方式支持视图名称和视图对象。通过ModelAndView构造方法可以指定返回的页面名称，也可以通过setViewName()方法跳转到指定的页面 , 使用addObject()设置需要返回的值，addObject()有几个不同参数的方法，可以默认和指定返回对象的名字。

（1）当你只有一个模型属性要返回时，可以在构造器中指定该属性来构造ModelAndView对象：

**[java]** [view plain](http://blog.csdn.net/tongxinxiao/article/details/39502519) [copy](http://blog.csdn.net/tongxinxiao/article/details/39502519)

1. **package** com.apress.springrecipes.court.web;
2. ...
3. **import** org.springframework.web.servlet.ModelAndView;
4. **import** org.springframework.web.servlet.mvc.AbstractController;
5. **public** **class** WelcomeController **extends** AbstractController{
6. **public** ModelAndView handleRequestInternal(HttpServletRequest request,
7. HttpServletResponse response)**throws** Exception{
8. Date today = **new** Date();
9. **return** **new** ModelAndView("welcome","today",today);
10. }
11. }

        （2）如果有不止一个属性要返回，可以先将它们传递到一个Map中再来构造ModelAndView对象。

**[java]** [view plain](http://blog.csdn.net/tongxinxiao/article/details/39502519) [copy](http://blog.csdn.net/tongxinxiao/article/details/39502519)

1. **package** com.apress.springrecipes.court.web;
2. ...
3. **import** org.springframework.web.servlet.ModelAndView;
4. **import** org. springframework.web.servlet.mvc.AbstractController;
5. **public** **class** ReservationQueryController **extends** AbstractController{
6. ...
7. **public** ModelAndView handleRequestInternal(HttpServletRequest request,
8. HttpServletResponse response)**throws** Exception{
9. ...
10. Map<String,Object> model = **new** HashMap<String,Object>();
11. **if**(courtName != **null**){
12. model.put("courtName",courtName);
13. model.put("reservations",reservationService.query(courtName));
14. }
15. **return** **new** ModelAndView("reservationQuery",model);
16. }
17. }

Spring也提供了ModelMap,这是java.util.Map实现，可以根据模型属性的具体类型自动生成模型属性的名称。

**[java]** [view plain](http://blog.csdn.net/tongxinxiao/article/details/39502519) [copy](http://blog.csdn.net/tongxinxiao/article/details/39502519)

1. **package** com.apress.springrecipes.court.web;
2. ...
3. **import** org.springframework.ui.ModelMap;
4. **import** org.springframework.web.servlet.ModelAndView;
5. **import** org.springframework.web.servlet.mvc.AbstractController;
6. **public** **class** ReservationQueryController **extends** AbstractController{
7. ...
8. **public** ModelAndView handleRequestInternal(HttpServletRequest request,
9. HttpServletResponse response)**throws** Exception{
10. ...
11. ModelMap model = **new** ModelMap();
12. **if**(courtName != **null**){
13. model.addAttribute("courtName",courtName);
14. model.addAttribute("reservations",reservationService.query(courtName));
15. }
16. **return** **new** ModelAndView("reservationQuery",model);
17. }
18. }

这里，我又想多说一句：ModelMap对象主要用于传递控制方法处理数据到结果页面，也就是说我们把结果页面上需要的数据放到ModelMap对象中即可，他的作用类似于request对象的setAttribute方法的作用，用来在一个请求过程中传递处理的数据。通过以下方法向页面传递参数：   
addAttribute(String key,Object value); 在页面上可以通过el变量方式$key或者bboss的一系列数据展示标签获取并展示modelmap中的数据。   
modelmap本身不能设置页面跳转的url地址别名或者物理跳转地址，那么我们可以通过控制器方法的返回值来设置跳转url地址别名或者物理跳转地址。 比如：

**[java]** [view plain](http://blog.csdn.net/tongxinxiao/article/details/39502519) [copy](http://blog.csdn.net/tongxinxiao/article/details/39502519)

1. **public** String xxxxmethod(String someparam,ModelMap model)
2. {
3. //省略方法处理逻辑若干
4. //将数据放置到ModelMap对象model中,第二个参数可以是任何java类型
5. model.addAttribute("key",someparam);
6. ......
7. //返回跳转地址
8. **return** "path:handleok";
9. }

               在这些构造函数中最简单的ModelAndView是持有View的名称返回，之后View名称被view resolver，也就是实作org.springframework.web.servlet.View接口的实例解析，例如 InternalResourceView或JstlView等等：ModelAndView(String viewName)；如果您要返回Model对象，则可以使用Map来收集这些Model对象，然后设定给ModelAndView，使用下面这个版本的 ModelAndView：ModelAndView(String viewName, Map model)，Map对象中设定好key与value值，之后可以在视图中取出，如果您只是要返回一个Model对象，则可以使用下面这个 ModelAndView版本：ModelAndView(String viewName, String modelName, Object modelObject)，其中modelName，您可以在视图中取出Model并显示。

ModelAndView类别提供实作View接口的对象来作View的参数：

ModelAndView(View view)

ModelAndView(View view, Map model)

ModelAndView(View view, String modelName, Object modelObject)

**2【方法使用】**：给ModelAndView实例设置view的方法有两个：setViewName(String viewName) 和 setView(View view)。前者是使用viewName，后者是使用预先构造好的View对象。其中前者比较常用。事实上View是一个接口，而不是一个可以构造的具体类，我们只能通过其他途径来获取View的实例。对于viewName，它既可以是jsp的名字，也可以是tiles定义的名字，取决于使用的ViewNameResolver如何理解这个view name。如何获取View的实例以后再研究。  
而对应如何给ModelAndView实例设置model则比较复杂。有三个方法可以使用：  
addObject(Object modelObject);

addObject(String modelName, Object modelObject);  
addAllObjects(Map modelMap);

**3【作用简介】：**

ModelAndView对象有两个作用：   
作用一 设置转向地址,如下所示（这也是ModelAndView和ModelMap的主要区别）   
ModelAndView view = new ModelAndView("path:ok");   
  
作用二 用于传递控制方法处理结果数据到结果页面，也就是说我们把需要在结果页面上需要的数据放到ModelAndView对象中即可，他的作用类似于request对象的setAttribute方法的作用，用来在一个请求过程中传递处理的数据。通过以下方法向页面传递参数：   
addObject(String key,Object value);

==================================================个人总结，希望监督=================================================================  
【参考博文】：

1）http://fhd001.iteye.com/blog/1131639

2）http://yin-bp.iteye.com/blog/1312176#

3）http://itroop.iteye.com/blog/263845

版权声明：本文为博主原创文章，未经博主允许不得转载。