<http://blog.csdn.net/qq924862077/article/details/53946043>

版权声明：http://blog.csdn.net/qq924862077/

@SessionAttribute作用于处理器类上，用于在多个请求之间传递参数，类似于Session的Attribute，但不完全一样，一般来说@SessionAttribute设置的参数只用于暂时的传递，而不是长期的保存，长期保存的数据还是要放到Session中。

通过@SessionAttribute注解设置的参数有3类用法：

（1）在视图中通过request.getAttribute或session.getAttribute获取

（2）在后面请求返回的视图中通过session.getAttribute或者从model中获取

（3）自动将参数设置到后面请求所对应处理器的Model类型参数或者有@ModelAttribute注释的参数里面。

将一个参数设置到SessionAttribute中需要满足两个条件：

（1）在@SessionAttribute注解中设置了参数的名字或者类型

（2）在处理器中将参数设置到了model中。

@SessionAttribute用户后可以调用SessionStatus.setComplete来清除，这个方法只是清除SessionAttribute里的参数，而不会应用Session中的参数。

示例如下：注解@SessionAttribute中设置book、description和types={Double}，这样值会被放到@SessionAttribute中，但Redirect跳转时就可以重新获得这些数据了，接下来操作sessionStatus.setComplete()，则会清除掉所有的数据，这样再次跳转时就无法获取数据了。

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. @Controller
2. @RequestMapping("/book")
3. @SessionAttributes(value ={"book","description"},types={Double.**class**})
4. **public** **class** RedirectController {
6. @RequestMapping("/index")
7. **public** String index(Model model){
8. model.addAttribute("book", "金刚经");
9. model.addAttribute("description","不擦擦擦擦擦擦擦车");
10. model.addAttribute("price", **new** Double("1000.00"));
11. //跳转之前将数据保存到book、description和price中，因为注解@SessionAttribute中有这几个参数
12. **return** "redirect:get.action";
13. }
15. @RequestMapping("/get")
16. **public** String get(@ModelAttribute ("book") String book,ModelMap model,
17. SessionStatus sessionStatus){
18. //可以获得book、description和price的参数
19. System.out.println(model.get("book")+";"+model.get("description")+";"+model.get("price"));
20. sessionStatus.setComplete();
21. **return** "redirect:complete.action";
22. }
24. @RequestMapping("/complete")
25. **public** String complete(ModelMap modelMap){
26. //已经被清除，无法获取book的值
27. System.out.println(modelMap.get("book"));
28. modelMap.addAttribute("book", "妹纸");
29. **return** "sessionAttribute";
30. }
32. }

接下来我们分析一下@SessionAttribute的实现机制

第一步：我们首先要获取注解@SessionAttribute的值的情况，在RequestMappingHandlerAdapter中的getModelFactory中处理。

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. **protected** ModelAndView invokeHandlerMethod(HttpServletRequest request,
2. HttpServletResponse response, HandlerMethod handlerMethod) **throws** Exception {
4. //这里面对注解的@SessionAttribute的处理器做处理
5. ModelFactory modelFactory = getModelFactory(handlerMethod, binderFactory);
7. ........
8. //会对@SessionAttribute操作的值进行处理
9. modelFactory.initModel(webRequest, mavContainer, invocableMethod);
10. ........
12. **return** getModelAndView(mavContainer, modelFactory, webRequest);
13. }

在getModelFactory中会创建@SessionAttribute的处理器SessionAttributeHandler

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. **private** ModelFactory getModelFactory(HandlerMethod handlerMethod, WebDataBinderFactory binderFactory) {
2. //创建SessionAttribute处理器
3. SessionAttributesHandler sessionAttrHandler = getSessionAttributesHandler(handlerMethod);
5. **return** **new** ModelFactory(attrMethods, binderFactory, sessionAttrHandler);
6. }

getSessionAttributesHandler的操作就是获取或者初始化SessionAttributesHandler

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. //已经获取过的SessionAttribute放到Map中，如果没有则需要初始化
2. **private** SessionAttributesHandler getSessionAttributesHandler(HandlerMethod handlerMethod) {
3. Class<?> handlerType = handlerMethod.getBeanType();
4. SessionAttributesHandler sessionAttrHandler = **this**.sessionAttributesHandlerCache.get(handlerType);
5. **if** (sessionAttrHandler == **null**) {
6. **synchronized** (**this**.sessionAttributesHandlerCache) {
7. sessionAttrHandler = **this**.sessionAttributesHandlerCache.get(handlerType);
8. **if** (sessionAttrHandler == **null**) {
9. //初始化sessionAttrHandler,并放到map中
10. sessionAttrHandler = **new** SessionAttributesHandler(handlerType, sessionAttributeStore);
11. **this**.sessionAttributesHandlerCache.put(handlerType, sessionAttrHandler);
12. }
13. }
14. }
15. **return** sessionAttrHandler;
16. }

SessionAttributesHandler的构造函数中的操作如下，其实就是解析被@SessionAttribute注解的处理器，这样就完成了@SessionAttribute注解中设置的key的解析。

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. **public** SessionAttributesHandler(Class<?> handlerType, SessionAttributeStore sessionAttributeStore) {
2. Assert.notNull(sessionAttributeStore, "SessionAttributeStore may not be null.");
3. **this**.sessionAttributeStore = sessionAttributeStore;
4. //解析被@SessionAttribute注解的处理器
5. SessionAttributes annotation = AnnotationUtils.findAnnotation(handlerType, SessionAttributes.**class**);
6. **if** (annotation != **null**) {
7. **this**.attributeNames.addAll(Arrays.asList(annotation.names()));
8. **this**.attributeTypes.addAll(Arrays.asList(annotation.types()));
9. }
11. **for** (String attributeName : **this**.attributeNames) {
12. **this**.knownAttributeNames.add(attributeName);
13. }
14. }

接下来我们看看springMVC对@SessionAttribute的处理操作，在ModelFactory.initModel会对@SessionAttribute的注解进行处理操作。

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. **protected** ModelAndView invokeHandlerMethod(HttpServletRequest request,
2. HttpServletResponse response, HandlerMethod handlerMethod) **throws** Exception {
4. //这里面对注解的@SessionAttribute的处理器做处理
5. ModelFactory modelFactory = getModelFactory(handlerMethod, binderFactory);
7. ........
8. //会对@SessionAttribute操作的值进行处理
9. modelFactory.initModel(webRequest, mavContainer, invocableMethod);
10. ........
12. **return** getModelAndView(mavContainer, modelFactory, webRequest);
13. }

initModel其实做了两步操作，一是：获取上一次请求保存在SessionAttributeHandler中值，给这次请求的值用，二是将这次请求的处理结果可能会对上次的@SessionAttribute中的值进行改变的值进行保存给下一次请求使用。

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. **public** **void** initModel(NativeWebRequest request, ModelAndViewContainer mavContainer, HandlerMethod handlerMethod)
2. **throws** Exception {
3. //获取所有的@SessionAttribute注解设置的key中值
4. Map<String, ?> sessionAttributes = **this**.sessionAttributesHandler.retrieveAttributes(request);
5. //将获取的值传递给下一个请求使用
6. mavContainer.mergeAttributes(sessionAttributes);
8. invokeModelAttributeMethods(request, mavContainer);
10. //请求访问完之后将修改的值重新放到@SessionAttributeStore设置的key中
11. **for** (String name : findSessionAttributeArguments(handlerMethod)) {
12. **if** (!mavContainer.containsAttribute(name)) {
13. Object value = **this**.sessionAttributesHandler.retrieveAttribute(request, name);
14. **if** (value == **null**) {
15. **throw** **new** HttpSessionRequiredException("Expected session attribute '" + name + "'");
16. }
17. mavContainer.addAttribute(name, value);
18. }
19. }
20. }

sessionAttributesHandler.retrieveAttributes的操作是将request中值，按照注解@SessionAttribute中key取值处理，然后保存到attribute中，作为这次请求的传递值使用。

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. **public** Map<String, Object> retrieveAttributes(WebRequest request) {
2. Map<String, Object> attributes = **new** HashMap<String, Object>();
3. //获取注解@SessionAttribute中设置的key
4. **for** (String name : **this**.knownAttributeNames) {
5. //如果设置的key有值则把它保存到attribute中，给跳转之后的请求使用
6. Object value = **this**.sessionAttributeStore.retrieveAttribute(request, name);
7. **if** (value != **null**) {
8. attributes.put(name, value);
9. }
10. }
11. **return** attributes;
12. }

这样就完成了将值保存在SessionAttributeHandler中，这样下一次请求过来时依然可以从SessionAttributeHandler中获取上次的结果，完成了类似Session的实现机制，但明显感觉和Session不一样，所有的请求其值是保存在一个同一个SessionAttributeHandler中。

SessionAttributeHandler其实维持了一个Map结构来存取数据，功能主要有解析@SessionAttribute注解，get和set相关值，源码如下：

**[java]** [view plain](http://blog.csdn.net/qq924862077/article/details/53946043) [copy](http://blog.csdn.net/qq924862077/article/details/53946043)

[print?](http://blog.csdn.net/qq924862077/article/details/53946043)

1. **public** **class** SessionAttributesHandler {
3. **private** **final** Set<String> attributeNames = **new** HashSet<String>();
5. **private** **final** Set<Class<?>> attributeTypes = **new** HashSet<Class<?>>();
7. **private** **final** Set<String> knownAttributeNames =
8. Collections.newSetFromMap(**new** ConcurrentHashMap<String, Boolean>(4));
10. **private** **final** SessionAttributeStore sessionAttributeStore;

13. //构造函数，解析@SessionAttribute注解，将其设置额key等信息保存
14. **public** SessionAttributesHandler(Class<?> handlerType, SessionAttributeStore sessionAttributeStore) {
15. Assert.notNull(sessionAttributeStore, "SessionAttributeStore may not be null.");
16. **this**.sessionAttributeStore = sessionAttributeStore;
18. SessionAttributes annotation = AnnotationUtils.findAnnotation(handlerType, SessionAttributes.**class**);
19. **if** (annotation != **null**) {
20. **this**.attributeNames.addAll(Arrays.asList(annotation.names()));
21. **this**.attributeTypes.addAll(Arrays.asList(annotation.types()));
22. }
24. **for** (String attributeName : **this**.attributeNames) {
25. **this**.knownAttributeNames.add(attributeName);
26. }
27. }

30. **public** **boolean** hasSessionAttributes() {
31. **return** ((**this**.attributeNames.size() > 0) || (**this**.attributeTypes.size() > 0));
32. }
34. //判断类型
35. **public** **boolean** isHandlerSessionAttribute(String attributeName, Class<?> attributeType) {
36. Assert.notNull(attributeName, "Attribute name must not be null");
37. **if** (**this**.attributeNames.contains(attributeName) || **this**.attributeTypes.contains(attributeType)) {
38. **this**.knownAttributeNames.add(attributeName);
39. **return** **true**;
40. }
41. **else** {
42. **return** **false**;
43. }
44. }
46. //保存
47. **public** **void** storeAttributes(WebRequest request, Map<String, ?> attributes) {
48. **for** (String name : attributes.keySet()) {
49. Object value = attributes.get(name);
50. Class<?> attrType = (value != **null**) ? value.getClass() : **null**;
52. **if** (isHandlerSessionAttribute(name, attrType)) {
53. **this**.sessionAttributeStore.storeAttribute(request, name, value);
54. }
55. }
56. }
58. //获取
59. **public** Map<String, Object> retrieveAttributes(WebRequest request) {
60. Map<String, Object> attributes = **new** HashMap<String, Object>();
61. **for** (String name : **this**.knownAttributeNames) {
62. Object value = **this**.sessionAttributeStore.retrieveAttribute(request, name);
63. **if** (value != **null**) {
64. attributes.put(name, value);
65. }
66. }
67. **return** attributes;
68. }
70. //清除所有内容
71. **public** **void** cleanupAttributes(WebRequest request) {
72. **for** (String attributeName : **this**.knownAttributeNames) {
73. **this**.sessionAttributeStore.cleanupAttribute(request, attributeName);
74. }
75. }
77. //获取所有值
78. Object retrieveAttribute(WebRequest request, String attributeName) {
79. **return** **this**.sessionAttributeStore.retrieveAttribute(request, attributeName);
80. }
82. }