**扫描某个包下所有的类，输出所有使用了特定注解的类的注解值**

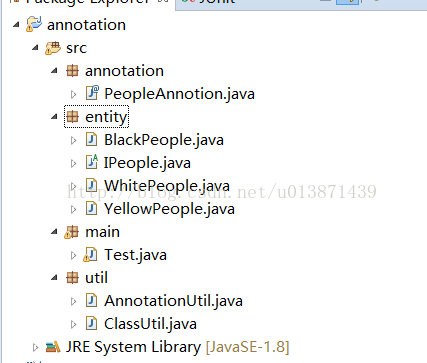
原创 2017年04月18日 18:36:17

* 标签：
* [注解](http://so.csdn.net/so/search/s.do?q=%E6%B3%A8%E8%A7%A3&t=blog)
* 1343

最近学习了注解的使用，今天做一个小结，有需要改进的地方欢迎指正。

代码功能：扫描某个包下所有的类，输出所有使用了特定注解的类的注解值：

工程代码结构：



1、注解：PeopleAnnotion

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** annotation;
3. **import** java.lang.annotation.Documented;
4. **import** java.lang.annotation.ElementType;
5. **import** java.lang.annotation.Retention;
6. **import** java.lang.annotation.RetentionPolicy;
7. **import** java.lang.annotation.Target;
9. @Target(ElementType.METHOD)
10. @Retention(RetentionPolicy.RUNTIME)
11. @Documented
12. **public** **@interface** PeopleAnnotion {
14. **public** String say();
16. }

定义注解使用在方法之上，

2、被注解的类：IPeople、BlackPeople、WhitePeople、YellowPeople

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** entity;
3. **import** annotation.PeopleAnnotion;
5. **public** **abstract** **class** IPeople {
6. @PeopleAnnotion(say="NoColor")
7. **public** **abstract** **void** say();
9. **public** **void** walk(){
10. System.out.println("I can Walk");
11. }
13. }

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** entity;
3. **import** annotation.PeopleAnnotion;
5. **public** **class** WhitePeople **extends** IPeople {
7. @Override
8. @PeopleAnnotion(say = "White")
9. **public** **void** say() {
10. System.out.println("I am White");
11. }
13. }

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** entity;
3. **import** annotation.PeopleAnnotion;
5. **public** **class** YellowPeople **extends** IPeople{
7. @Override
8. @PeopleAnnotion(say="Yellow")
9. **public** **void** say() {
10. System.out.println("I am Yellow");
11. }
13. }

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** entity;
3. **import** annotation.PeopleAnnotion;
5. **public** **class** BlackPeople **extends** IPeople{
7. @Override
8. @PeopleAnnotion(say="Black")
9. **public** **void** say() {
10. System.out.println("I am Black");
11. }
13. }

3、获取包内所有类：ClassUtil

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** util;
3. **import** java.io.File;
4. **import** java.io.FileFilter;
5. **import** java.io.IOException;
6. **import** java.net.JarURLConnection;
7. **import** java.net.URL;
8. **import** java.net.URLDecoder;
9. **import** java.util.ArrayList;
10. **import** java.util.Enumeration;
11. **import** java.util.List;
12. **import** java.util.jar.JarEntry;
13. **import** java.util.jar.JarFile;
15. **public** **class** ClassUtil {
17. /\*\*
18. \* 通过包名获取包内所有类
19. \*
20. \* @param pkg
21. \* @return
22. \*/
23. **public** **static** List<Class<?>> getAllClassByPackageName(Package pkg) {
24. String packageName = pkg.getName();
25. // 获取当前包下以及子包下所以的类
26. List<Class<?>> returnClassList = getClasses(packageName);
27. **return** returnClassList;
28. }
30. /\*\*
31. \* 通过接口名取得某个接口下所有实现这个接口的类
32. \*/
33. **public** **static** List<Class<?>> getAllClassByInterface(Class<?> c) {
34. List<Class<?>> returnClassList = **null**;
36. **if** (c.isInterface()) {
37. // 获取当前的包名
38. String packageName = c.getPackage().getName();
39. // 获取当前包下以及子包下所以的类
40. List<Class<?>> allClass = getClasses(packageName);
41. **if** (allClass != **null**) {
42. returnClassList = **new** ArrayList<Class<?>>();
43. **for** (Class<?> cls : allClass) {
44. // 判断是否是同一个接口
45. **if** (c.isAssignableFrom(cls)) {
46. // 本身不加入进去
47. **if** (!c.equals(cls)) {
48. returnClassList.add(cls);
49. }
50. }
51. }
52. }
53. }
55. **return** returnClassList;
56. }
58. /\*\*
59. \* 取得某一类所在包的所有类名 不含迭代
60. \*/
61. **public** **static** String[] getPackageAllClassName(String classLocation, String packageName) {
62. // 将packageName分解
63. String[] packagePathSplit = packageName.split("[.]");
64. String realClassLocation = classLocation;
65. **int** packageLength = packagePathSplit.length;
66. **for** (**int** i = 0; i < packageLength; i++) {
67. realClassLocation = realClassLocation + File.separator + packagePathSplit[i];
68. }
69. File packeageDir = **new** File(realClassLocation);
70. **if** (packeageDir.isDirectory()) {
71. String[] allClassName = packeageDir.list();
72. **return** allClassName;
73. }
74. **return** **null**;
75. }
77. /\*\*
78. \* 从包package中获取所有的Class
79. \*
80. \* @param pack
81. \* @return
82. \*/
83. **private** **static** List<Class<?>> getClasses(String packageName) {
85. // 第一个class类的集合
86. List<Class<?>> classes = **new** ArrayList<Class<?>>();
87. // 是否循环迭代
88. **boolean** recursive = **true**;
89. // 获取包的名字 并进行替换
90. String packageDirName = packageName.replace('.', '/');
91. // 定义一个枚举的集合 并进行循环来处理这个目录下的things
92. Enumeration<URL> dirs;
93. **try** {
94. dirs = Thread.currentThread().getContextClassLoader().getResources(packageDirName);
95. // 循环迭代下去
96. **while** (dirs.hasMoreElements()) {
97. // 获取下一个元素
98. URL url = dirs.nextElement();
99. // 得到协议的名称
100. String protocol = url.getProtocol();
101. // 如果是以文件的形式保存在服务器上
102. **if** ("file".equals(protocol)) {
103. // 获取包的物理路径
104. String filePath = URLDecoder.decode(url.getFile(), "UTF-8");
105. // 以文件的方式扫描整个包下的文件 并添加到集合中
106. findAndAddClassesInPackageByFile(packageName, filePath, recursive, classes);
107. } **else** **if** ("jar".equals(protocol)) {
108. // 如果是jar包文件
109. // 定义一个JarFile
110. JarFile jar;
111. **try** {
112. // 获取jar
113. jar = ((JarURLConnection) url.openConnection()).getJarFile();
114. // 从此jar包 得到一个枚举类
115. Enumeration<JarEntry> entries = jar.entries();
116. // 同样的进行循环迭代
117. **while** (entries.hasMoreElements()) {
118. // 获取jar里的一个实体 可以是目录 和一些jar包里的其他文件 如META-INF等文件
119. JarEntry entry = entries.nextElement();
120. String name = entry.getName();
121. // 如果是以/开头的
122. **if** (name.charAt(0) == '/') {
123. // 获取后面的字符串
124. name = name.substring(1);
125. }
126. // 如果前半部分和定义的包名相同
127. **if** (name.startsWith(packageDirName)) {
128. **int** idx = name.lastIndexOf('/');
129. // 如果以"/"结尾 是一个包
130. **if** (idx != -1) {
131. // 获取包名 把"/"替换成"."
132. packageName = name.substring(0, idx).replace('/', '.');
133. }
134. // 如果可以迭代下去 并且是一个包
135. **if** ((idx != -1) || recursive) {
136. // 如果是一个.class文件 而且不是目录
137. **if** (name.endsWith(".class") && !entry.isDirectory()) {
138. // 去掉后面的".class" 获取真正的类名
139. String className = name.substring(packageName.length() + 1, name.length() - 6);
140. **try** {
141. // 添加到classes
142. classes.add(Class.forName(packageName + '.' + className));
143. } **catch** (ClassNotFoundException e) {
144. e.printStackTrace();
145. }
146. }
147. }
148. }
149. }
150. } **catch** (IOException e) {
151. e.printStackTrace();
152. }
153. }
154. }
155. } **catch** (IOException e) {
156. e.printStackTrace();
157. }
159. **return** classes;
160. }
162. /\*\*
163. \* 以文件的形式来获取包下的所有Class
164. \*
165. \* @param packageName
166. \* @param packagePath
167. \* @param recursive
168. \* @param classes
169. \*/
170. **private** **static** **void** findAndAddClassesInPackageByFile(String packageName, String packagePath, **final** **boolean** recursive, List<Class<?>> classes) {
171. // 获取此包的目录 建立一个File
172. File dir = **new** File(packagePath);
173. // 如果不存在或者 也不是目录就直接返回
174. **if** (!dir.exists() || !dir.isDirectory()) {
175. **return**;
176. }
177. // 如果存在 就获取包下的所有文件 包括目录
178. File[] dirfiles = dir.listFiles(**new** FileFilter() {
179. // 自定义过滤规则 如果可以循环(包含子目录) 或则是以.class结尾的文件(编译好的java类文件)
180. **public** **boolean** accept(File file) {
181. **return** (recursive && file.isDirectory()) || (file.getName().endsWith(".class"));
182. }
183. });
184. // 循环所有文件
185. **for** (File file : dirfiles) {
186. // 如果是目录 则继续扫描
187. **if** (file.isDirectory()) {
188. findAndAddClassesInPackageByFile(packageName + "." + file.getName(), file.getAbsolutePath(), recursive, classes);
189. } **else** {
190. // 如果是java类文件 去掉后面的.class 只留下类名
191. String className = file.getName().substring(0, file.getName().length() - 6);
192. **try** {
193. // 添加到集合中去
194. classes.add(Class.forName(packageName + '.' + className));
195. } **catch** (ClassNotFoundException e) {
196. e.printStackTrace();
197. }
198. }
199. }
200. }
201. }

4、获取特定包内注解方法的注解值：AnnotationUtil

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** util;
3. **import** java.lang.reflect.Method;
4. **import** java.util.List;
6. **import** annotation.PeopleAnnotion;
8. **public** **class** AnnotationUtil {
10. **public** **static** **void** validAnnotation(List<Class<?>> clsList){
11. **if** (clsList != **null** && clsList.size() > 0) {
12. **for** (Class<?> cls : clsList) {
13. //获取类中的所有的方法
14. Method[] methods = cls.getDeclaredMethods();
15. **if** (methods != **null** && methods.length > 0) {
16. **for** (Method method : methods) {
17. PeopleAnnotion peopleAnnotion = (PeopleAnnotion) method.getAnnotation(PeopleAnnotion.**class**);
18. **if** (peopleAnnotion != **null**) {
19. //可以做权限验证
20. System.out.println(peopleAnnotion.say());
21. }
22. }
23. }
24. }
25. }
26. }
28. }

 5、测试类：Test

**[java]** [view plain](http://blog.csdn.net/u013871439/article/details/70231288) [copy](http://blog.csdn.net/u013871439/article/details/70231288)

1. **package** main;
3. **import** java.util.List;
5. **import** javax.crypto.KeyGenerator;
6. **import** javax.crypto.Mac;
7. **import** javax.crypto.SecretKey;
8. **import** javax.crypto.spec.SecretKeySpec;
10. **import** entity.BlackPeople;
11. **import** util.AnnotationUtil;
12. **import** util.ClassUtil;
14. **public** **class** Test {
15. **public** **static** **void** main(String[] args) **throws** InstantiationException, IllegalAccessException {
16. // 获取特定包下所有的类(包括接口和类)
17. List<Class<?>> clsList = ClassUtil.getAllClassByPackageName(BlackPeople.**class**.getPackage());
18. //输出所有使用了特定注解的类的注解值
19. AnnotationUtil.validAnnotation(clsList);
20. }
22. }

版权声明：本文为博主原创文章，未经博主允许不得转载。