[通过接口获取所有实现类以及通过注解获取实现类的思路](http://blog.csdn.net/yeluosc/article/details/47315751)

2015-08-06 14:30 532人阅读 [评论](http://blog.csdn.net/yeluosc/article/details/47315751#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/yeluosc/article/details/47315751#report)

 分类：



java（23）



版权声明：本文为博主原创文章，未经博主允许不得转载。

近日，工作上被安排修改项目，别人写的一套内外网同步系统，修改成一套上报下发系统，处理很复杂，在原先那人的基础上修改了不少源代码，比较痛苦的是修改他的源代码，必须要看懂全部代码，改的才能顺利进行下去，其实这是违反面向对象的一项重要原则——开闭原则，像这样的代码，换个人写，时间成本会很高，而如果做好了面向对象的设计的话，在修改的时候，只需实现预留好的接口就可以实现，所以最近一直在构思如何重构整个项目，而且这个系统不像常见的业务系统那样：很多业务都已经确定了，完全没有抽象的必要。考虑到需要一些灵活性，比如未来某个程序猿，需要扩展我预留的接口，我需要调用其实现类，这就导致我要知道他的实现类名称，以获得其beanId，通过[**spring**](http://lib.csdn.net/base/javaee) IoC再进行获取调用，但是beanId如何获取呢？通过xml或者配置文件感觉又太麻烦，干脆使用一个注解，通过注解获取类的名字，自动转换成相应的beanId，强转后调用接口方法即可完成，当然这样的话有一些弊端，比如自定义的beanId如何处理，暂时还未想好。想法有了，还需要找一些资料弥补知识，下面直接贴上代码：

**[java]** [view plain](http://blog.csdn.net/yeluosc/article/details/47315751) [copy](http://blog.csdn.net/yeluosc/article/details/47315751)

1. **package** com.tjhq.synch2.test;
3. **import** java.io.File;
4. **import** java.io.IOException;
5. **import** java.net.URL;
6. **import** java.util.ArrayList;
7. **import** java.util.Enumeration;
8. **import** java.util.List;
10. **public** **class** ClassUtil {
12. /\*\*
13. \* @Description: 根据一个接口返回该接口的所有类
14. \* @param c 接口
15. \* @return List<Class>    实现接口的所有类
16. \*/
17. @SuppressWarnings("unchecked")
18. **public** **static** List<Class> getAllClassByInterface(Class c){
19. List returnClassList = **new** ArrayList<Class>();
20. //判断是不是接口,不是接口不作处理
21. **if**(c.isInterface()){
22. String packageName = c.getPackage().getName();  //获得当前包名
23. **try** {
24. List<Class> allClass = getClasses(packageName);//获得当前包以及子包下的所有类
26. //判断是否是一个接口
27. **for**(**int** i = 0; i < allClass.size(); i++){
28. **if**(c.isAssignableFrom(allClass.get(i))){
29. **if**(!c.equals(allClass.get(i))){
30. returnClassList.add(allClass.get(i));
31. }
32. }
33. }
34. } **catch** (Exception e) {
35. // TODO: handle exception
36. }
37. }
38. **return** returnClassList;
39. }
41. /\*\*
42. \*
43. \* @Description: 根据包名获得该包以及子包下的所有类不查找jar包中的
44. \* @param pageName 包名
45. \* @return List<Class>    包下所有类
46. \*/
47. **private** **static** List<Class> getClasses(String packageName) **throws** ClassNotFoundException,IOException{
48. ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
49. String path = packageName.replace(".", "/");
50. Enumeration<URL> resources = classLoader.getResources(path);
51. List<File> dirs = **new** ArrayList<File>();
52. **while**(resources.hasMoreElements()){
53. URL resource = resources.nextElement();
54. String newPath = resource.getFile().replace("%20", " ");
55. dirs.add(**new** File(newPath));
56. }
57. ArrayList<Class> classes = **new** ArrayList<Class>();
58. **for**(File directory:dirs){
59. classes.addAll(findClass(directory, packageName));
60. }
61. **return** classes;
62. }
64. **private** **static**  List<Class> findClass(File directory, String packageName)
65. **throws** ClassNotFoundException{
66. List<Class> classes = **new** ArrayList<Class>();
67. **if**(!directory.exists()){
68. **return** classes;
69. }
70. File[] files = directory.listFiles();
71. **for**(File file:files){
72. **if**(file.isDirectory()){
73. **assert** !file.getName().contains(".");
74. classes.addAll(findClass(file, packageName+"."+file.getName()));
75. }**else** **if**(file.getName().endsWith(".class")){
76. classes.add(Class.forName(packageName+"."+file.getName().substring(0,file.getName().length()-6)));
77. }
78. }
79. **return** classes;
80. }
82. @SuppressWarnings("unchecked")
83. **public** **static** List<Class> getAllClassByAnnotation(Class annotationClass){
84. List returnClassList = **new** ArrayList<Class>();
85. //判断是不是注解
86. **if**(annotationClass.isAnnotation()){
87. String packageName = annotationClass.getPackage().getName();    //获得当前包名
88. **try** {
89. List<Class> allClass = getClasses(packageName);//获得当前包以及子包下的所有类
91. **for**(**int** i = 0; i < allClass.size(); i++){
92. **if**(allClass.get(i).isAnnotationPresent(annotationClass)){
93. returnClassList.add(allClass.get(i));
94. }
95. }
96. } **catch** (Exception e) {
97. // TODO: handle exception
98. }
99. }
100. **return** returnClassList;
101. }
102. }

前3个方法是通过接口找实现类的方法，网上随便就搜到了，我这里把物理路径中的空格20%又转换成了空格，否则找不到路径，报错；最后一个方法是照猫画虎，通过注解找其所有实现类的方法。传入注解的Class对象，即可完成；

**[java]** [view plain](http://blog.csdn.net/yeluosc/article/details/47315751) [copy](http://blog.csdn.net/yeluosc/article/details/47315751)

1. **package** com.tjhq.synch2.test;
3. **import** java.util.List;
5. **public** **class** TestClassUtil {
7. /\*\*
8. \* @param args
9. \* @throws IllegalAccessException
10. \* @throws InstantiationException
11. \*/
12. @SuppressWarnings("unchecked")
13. **public** **static** **void** main(String[] args) **throws** InstantiationException, IllegalAccessException {
14. //      List<Class> list = ClassUtil.getAllClassByInterface(SignInterface.class);
15. //      for(Class c : list){
16. //          String name = c.getAnnotation(SignAnnotation.class).getClass().
17. //          System.out.println(name);
18. //          SignInterface sign = (SignInterface)c.newInstance();
19. //          System.out.println(sign.returnStr());
20. //      }
21. List<Class> list = ClassUtil.getAllClassByAnnotation(SignAnnotation.**class**);
22. **for**(Class c : list){
23. //SignInterface sign = (SignInterface)c.newInstance();
24. System.out.println(c.getSimpleName());
25. SignAnnotation annot = (SignAnnotation) c.getAnnotation(SignAnnotation.**class**);
26. System.out.println(annot.TypeName());
27. //System.out.println(sign.returnStr());
28. }

31. }
33. }

下面贴上注解的代码：

**[java]** [view plain](http://blog.csdn.net/yeluosc/article/details/47315751) [copy](http://blog.csdn.net/yeluosc/article/details/47315751)

1. **package** com.tjhq.synch2.test;
3. **import** java.lang.annotation.ElementType;
4. **import** java.lang.annotation.Retention;
5. **import** java.lang.annotation.RetentionPolicy;
6. **import** java.lang.annotation.Target;
8. @Target(ElementType.TYPE)
9. @Retention(RetentionPolicy.RUNTIME)
10. **public** **@interface** SignAnnotation {
11. **public** TYPE TypeName() **default** TYPE.DEFAULTTYPE;
12. **public** **enum** TYPE{HANDLER,DATATYPE,DEFAULTTYPE};
13. }



因为需要使用反射，所以Retention的值必须设置为RUNTIME,否则可能会取不到Class对象（没试），接口的代码不贴了，就一个空接口，标记而已！