[JAVA-获取实现了指定接口类的所有实现类或继承了指定类的所有子类](http://blog.csdn.net/u010156024/article/details/44875459)

标签： [指定接口的所有实现类](http://www.csdn.net/tag/%e6%8c%87%e5%ae%9a%e6%8e%a5%e5%8f%a3%e7%9a%84%e6%89%80%e6%9c%89%e5%ae%9e%e7%8e%b0%e7%b1%bb)[指定父类的所有子类](http://www.csdn.net/tag/%e6%8c%87%e5%ae%9a%e7%88%b6%e7%b1%bb%e7%9a%84%e6%89%80%e6%9c%89%e5%ad%90%e7%b1%bb)[接口类](http://www.csdn.net/tag/%e6%8e%a5%e5%8f%a3%e7%b1%bb)[父类](http://www.csdn.net/tag/%e7%88%b6%e7%b1%bb)

2015-04-04 19:54 1530人阅读 [评论](http://blog.csdn.net/u010156024/article/details/44875459#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/u010156024/article/details/44875459#report)

 分类：



java（22）



版权声明：本文为博主【龙吟在天】原创文章，未经博主允许不得转载。

实际编程过程中，我们可能遇到这样的问题，就是获取实现了指定接口类的所有实现类。   
本工具类就提供了这样的功能。下面是工具类的详细解析：

/\*\*

\* 查找指定路径下面实现指定接口的全部类

\* @author longyin

\* @author 博客地址：http://blog.csdn.net/u010156024

\* 如果大家有什么问题或疑问，欢迎留言或评论，谢谢!!

\*/

public class ClassUtil {

@SuppressWarnings({ "rawtypes", "unchecked" })

public static ArrayList<Class> getAllClassByInterface(Class clazz){

ArrayList<Class> list = new ArrayList<>();

//判断是否是一个接口

if (clazz.isInterface()) {

try {

ArrayList<Class> allClass = getAllClass(clazz.getPackage().getName());

/\*\*

\* 循环判断路径下的所有类是否实现了指定的接口

\* 并且排除接口类自己

\*/

for (int i = 0; i < allClass.size(); i++) {

/\*\*

\* 判断是不是同一个接口

\* 该方法的解析，请参考博客：

\* http://blog.csdn.net/u010156024/article/details/44875195

\*/

if (clazz.isAssignableFrom(allClass.get(i))) {

if (!clazz.equals(allClass.get(i))) {//自身并不加进去

list.add(allClass.get(i));

}else {

}

}

}

} catch (Exception e) {

System.out.println("出现异常");

}

}else {

//如果不是接口不作处理

}

return list;

}

/\*\*

\* 从一个指定路径下查找所有的类

\* @param name

\*/

@SuppressWarnings("rawtypes")

private static ArrayList<Class> getAllClass(String packagename) {

ArrayList<Class> list = new ArrayList<>();

ClassLoader classLoader = Thread.currentThread().getContextClassLoader();

String path = packagename.replace('.', '/');

try {

ArrayList<File> fileList = new ArrayList<>();

/\*\*

\* 这里面的路径使用的是相对路径

\* 如果大家在测试的时候获取不到，请理清目前工程所在的路径

\* 使用相对路径更加稳定！

\* 另外，路径中切不可包含空格、特殊字符等！

\* 本人在测试过程中由于空格，吃了大亏！！！

\*/

Enumeration<URL> enumeration = classLoader.getResources("../bin/"+path);

while (enumeration.hasMoreElements()) {

URL url = enumeration.nextElement();

fileList.add(new File(url.getFile()));

}

for (int i = 0; i < fileList.size(); i++) {

list.addAll(findClass(fileList.get(i),packagename));

}

} catch (IOException e) {

e.printStackTrace();

}

return list;

}

/\*\*

\* 如果file是文件夹，则递归调用findClass方法，或者文件夹下的类

\* 如果file本身是类文件，则加入list中进行保存，并返回

\* @param file

\* @param packagename

\* @return

\*/

@SuppressWarnings("rawtypes")

private static ArrayList<Class> findClass(File file,String packagename) {

ArrayList<Class> list = new ArrayList<>();

if (!file.exists()) {

return list;

}

File[] files = file.listFiles();

for (File file2 : files) {

if (file2.isDirectory()) {

assert !file2.getName().contains(".");//添加断言用于判断

ArrayList<Class> arrayList = findClass(file2, packagename+"."+file2.getName());

list.addAll(arrayList);

}else if(file2.getName().endsWith(".class")){

try {

//保存的类文件不需要后缀.class

list.add(Class.forName(packagename + '.' + file2.getName().substring(0,

file2.getName().length()-6)));

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

}

return list;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106

上面方法，大家在运行测试的时候一定记得更改路径。一定以当前功能文件所在的路径下使用相对路径进行更新，是路径执行可执行的.class文件。   
上面的类中的public方法是提供给外界进行调用的，该方法里面有一个isInterface的判断，加入了该判断的话，就只能获取所有实现了指定接口的类，那么要获取继承了指定类的所有类怎么做呢？   
非常简单，不加这个判断就可以了。   
所有上面的public方法更改如下：

@SuppressWarnings({ "rawtypes", "unchecked" })

public static ArrayList<Class> getAllClassByInterface(Class clazz){

ArrayList<Class> list = new ArrayList<>();

//获取指定接口的实现类

if (clazz.isInterface()) {

try {

ArrayList<Class> allClass = getAllClass(clazz.getPackage().getName());

/\*\*

\* 循环判断路径下的所有类是否实现了指定的接口

\* 并且排除接口类自己

\*/

for (int i = 0; i < allClass.size(); i++) {

/\*\*

\* 判断是不是同一个接口

\* isAssignableFrom该方法的解析，请参考博客：

\* http://blog.csdn.net/u010156024/article/details/44875195

\*/

if (clazz.isAssignableFrom(allClass.get(i))) {

if (!clazz.equals(allClass.get(i))) {//自身并不加进去

list.add(allClass.get(i));

}else {

}

}

}

} catch (Exception e) {

System.out.println("出现异常");

}

//如果不是接口，则获取它的所有子类

}else{

try {

ArrayList<Class> allClass = getAllClass(clazz.getPackage().getName());

/\*\*

\* 循环判断路径下的所有类是否继承了指定类

\* 并且排除父类自己

\*/

for (int i = 0; i < allClass.size(); i++) {

/\*\*

\* isAssignableFrom该方法的解析，请参考博客：

\* http://blog.csdn.net/u010156024/article/details/44875195

\*/

if (clazz.isAssignableFrom(allClass.get(i))) {

if (!clazz.equals(allClass.get(i))) {//自身并不加进去

list.add(allClass.get(i));

}else {

}

}

}

} catch (Exception e) {

System.out.println("出现异常");

}

}

return list;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55



* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55

在else中可以获取到所有继承了指定类的所有子类！！！

本人测试完成，完全满足需求！！！   
如果有疑问，请留言或评论！！