一、常用的类

1、ClasspathHelper



通过上面的测试代码可以看出ClasspathHelper这个类提供的上面的三个方法是获取资源的路径。

2、Reflections

public class Test01 {

    private static String **packNameList** = "com.jeff.reflection.bean,"+

                                                                  +"com.jeff.reflection.test,com.jeff.anotation";  
      
    public static void main(String[] args) {  
        FilterBuilder filterBuilder = new FilterBuilder();

        for (String packName : packNameList.split(",")) {  
            filterBuilder = filterBuilder.includePackage(packName);//定义要扫描的包  
        }  
        Predicate<String> filter = filterBuilder;//过滤器

        //里面放的是这些包所在的资源路径  
        Collection<URL> urlTotals = new ArrayList<URL>();  
        for (String packName : packNameList.split(",")) {  
            Set<URL> urls = ClasspathHelper.forPackage(packName);  
            urlTotals.addAll(urls);  
        }

        /\*\*  
         \* 定义Reflections对象，指明"包过滤器"，以及扫描器的类型，主要把是扫描器的类型  
         \* 细分之后，得到对应的数据  
         \*/  
        Reflections reflections = new Reflections(new ConfigurationBuilder()

                .filterInputsBy(filter)  
                .setScanners(

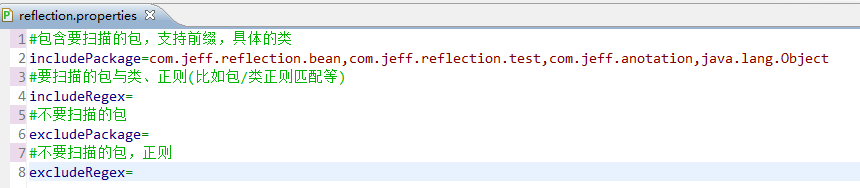
**new SubTypesScanner().filterResultsBy(filter),  
                        new TypeAnnotationsScanner().filterResultsBy(filter),  
                        new FieldAnnotationsScanner().filterResultsBy(filter),  
                        new MethodAnnotationsScanner().filterResultsBy(filter),  
                        new MethodParameterScanner().filterResultsBy(filter)**

                ).setUrls(urlTotals));  
          
        //获取方法上待MyAnotation注解的所有的方法  
        Set<Method> methods = reflections.getMethodsAnnotatedWith(MyAnotation.class);  
        for(Method m : methods){  
            System.out.println(m.getName());  
        }  
    }  
}

注意：从上面的代码可以看出，我们在使用的时候要先定义过滤器，把你要扫描的包加入。然后获取这个资源路径。最后定义reflections对象，然后拿reflections获取对应的Method、Field等等对象。

二、封装工具

1、在配置中定义扫描类包



2、SacnUtils类

public class ScanUtils {

    private static final String  SPLIT\_STR = ",";//通过","切割包  
      
    private static Properties pro = null;  
      
    private static Reflections reflections = null;  
      
    static{  
        InputStream in = ScanUtils.class.getClassLoader()

                                    .getResourceAsStream("reflection.properties");  
        pro = new Properties();  
        try {  
            pro.load(in);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
      
    /\*\*  
     \* 初始化  
     \* [@return](https://my.oschina.net/u/556800)  
     \*/  
    public static Reflections init(){  
        //定义过滤器集合  
        FilterBuilder filterBuilder = new FilterBuilder();  
        Collection<URL> urlTotals = new ArrayList<URL>();  
        //初始化过滤器集合  
        **initPackage(filterBuilder,urlTotals);**  
        Predicate<String> filter = filterBuilder;  
          
        reflections = new Reflections(new ConfigurationBuilder().filterInputsBy(filter)  
                .setScanners(new SubTypesScanner().filterResultsBy(filter),  
                        new TypeAnnotationsScanner()  
                                .filterResultsBy(filter),  
                        new FieldAnnotationsScanner()  
                                .filterResultsBy(filter),  
                        new MethodAnnotationsScanner()  
                                .filterResultsBy(filter),  
                        new MethodParameterScanner().filterResultsBy(filter)).setUrls(urlTotals));  
          
        return reflections;  
    }  
      
    /\*\*  
     \* 获取Reflections对象  
     \* [@return](https://my.oschina.net/u/556800)  
     \*/  
    public static Reflections getReflections() {  
        return reflections;  
    }  
      
    /\*\*  
     \* 初始化配置文件的包 reflection.properties  
     \* [@param](https://my.oschina.net/u/2303379) filterBuilder  
     \*/  
    private static void initPackage(FilterBuilder filterBuilder,Collection<URL> urlTotals){  
          
        String includePackage = pro.getProperty("includePackage");  
        if(StringUtils.isNotEmpty(includePackage)){  
            for (String packName : includePackage.split(SPLIT\_STR)) {  
                filterBuilder = filterBuilder.includePackage(packName);  
                  
                /\*\*  
                 \* 通过ClasspathHelper（ClassLoader）类加载去去加载该包路径下的所有资源(URL),  
                 \* 最终通过URL资源（类、方法，注解，修饰符等等）, 结合Filter可以得到Reflections  
                 \* 对象。  
                 \*/  
                Set<URL> urls = ClasspathHelper.forPackage(packName);  
                urlTotals.addAll(urls);  
            }  
        }  
        String includeRegex = pro.getProperty("includeRegex");  
        if(StringUtils.isNotEmpty(includeRegex)){  
            for (String packName : includeRegex.split(SPLIT\_STR)) {  
                filterBuilder = filterBuilder.include(packName);  
                  
                Set<URL> urls = ClasspathHelper.forPackage(packName);  
                urlTotals.addAll(urls);  
            }  
        }  
          
        String excludePackage = pro.getProperty("excludePackage");  
        if(StringUtils.isNotEmpty(excludePackage)){  
            for (String packName : excludePackage.split(SPLIT\_STR)) {  
                filterBuilder = filterBuilder.excludePackage(packName);  
                  
                Set<URL> urls = ClasspathHelper.forPackage(packName);  
                urlTotals.addAll(urls);  
            }  
        }  
          
        String excludeRegex = pro.getProperty("excludeRegex");  
        if(StringUtils.isNotEmpty(excludeRegex)){  
            for (String packName : excludeRegex.split(SPLIT\_STR)) {  
                filterBuilder = filterBuilder.includePackage(packName);  
                  
                Set<URL> urls = ClasspathHelper.forPackage(packName);  
                urlTotals.addAll(urls);  
            }  
        }  
    }  
}

3、测试类路径的资源

public class Test02 {

    @SuppressWarnings("rawtypes")  
    public static void main(String[] args) {  
        Reflections reflections = ScanUtils.init();  
        /\*\*  
         \* 注意要获取拥有某个注解的方法，注意这个注解所在的包一定要配置到我们的includePackage  
         \* 或者includeRegex下，否则获取不到对应的方法  
         \*/  
        Set<Method> methods = reflections.getMethodsAnnotatedWith(MyAnotation.class);  
        for(Method m : methods){  
               System.out.println(m.getName());  
        }  
        /\*\*  
         \*获取Object的子类，注意：Object这个类或者它的包，要配置在includePackage下，  
         \*另外得到的结果也一定是在“配置包”路径下面的类：includePackage=com.jeff.reflection.bean,  
         \*com.jeff.reflection.test,com.jeff.anotation,java.lang.Object  
         \*/  
        Set sub = reflections.getSubTypesOf(Object.class);  
        System.out.println(sub);  
          
        /\*\*  
         \* 得到String为返回值的所有方法  
         \*/  
        methods = reflections.getMethodsReturn(String.class);  
        for(Method m : methods){  
               System.out.println(m.getName());  
        }  
          
        /\*\*  
         \* 获取构造方法中参数带ParamAnotation.class注解的所有构造方法  
         \*/  
        Set<Constructor> cons = reflections.

                            getConstructorsWithAnyParamAnnotated(ParamAnotation.class);  
        for(Constructor c : cons){  
            System.out.println(c.getModifiers() + "," + c.getName());//1,com.jeff.reflection.bean.User  
        }  
          
        /\*\*  
         \* 获取该注解上面的修饰的注解类型（该注解一定是在我们配置的包路径下）  
         \* 由于@Documented注解不在我们配置的包路径下，下面的方法获取不了它的  
         \* 注解  
         \* @Documented  
         \* @ParamAnotation(name="")  
         \* public @interface MyAnotation {  
         \* }  
         \*/  
        Set anot = reflections.getTypesAnnotatedWith(ParamAnotation.class,true);  
        System.out.println(anot);//[interface com.jeff.anotation.MyAnotation]  
          
        /\*\*  
         \* 获取资源：从reflections晒选出符合条件（正则）的资源，所以在includePackage  
         \* 配置这个下面资源的路径，要不筛选不出来。（查看Test03.java）  
         \*/  
        Set<String> res = reflections.getResources(Pattern.compile(".\*\\.properties"));  
        for(Iterator<String> ite = res.iterator();ite.hasNext();){  
            String str = ite.next();  
            System.out.println(str);  
        }  
    }  
}

4、获取资源

     /\*\*  
     \* 首先要通过过滤器指定包/类/文件资源，因为ClassLoader要依靠指定的过滤器去找出  
     \* 符合条件的资源。然后通过api：reflections.getResources(regex)找出符合条件  
     \* 的资源。  
     \*/  
    public static void main(String[] args) {  
        Predicate<String> filter = new FilterBuilder().include(".\*\\.properties")

                                                                    .exclude(".\*testModel-reflections\\.xml");  
        Reflections reflections = new Reflections(new ConfigurationBuilder()  
                .filterInputsBy(filter)  
                .setScanners(new ResourcesScanner())  
                .setUrls(ClasspathHelper.forPackage("com.jeff.")));  
          
        Set<String> str = reflections.getResources(Pattern.compile(".\*\\.properties"));  
        for(String s : str){  
            System.out.println(s);  
        }  
    }