FAQ

提供一些常见问题的回答，如有问题请联系作者：qrx@smail.nju.edu.cn

2020-11-11

1. WALA仓库的代码clone下来在本地构建失败，如何解决？

   A：这里建议使用编译好的WALA jar包来做，也就是从Maven中心库下载jar包在本地作为lib运行。WALA仓库中的代码可以作为参考。因为某些环境原因，WALA代码在国内的构建经常会出问题。

2. 加载类的时候出现"Bad File"报错，怎么回事？该如何解决？

   A：可能是代码无法正确加载到类文件，尝试以下操作：

   - 确保作为参数类加载器代码所在的类，如：

     ```
     class SomeClass {

         public void loadClass() {
             ClassLoader classloader = SomeClass.class.getClassLoader();
             /*
                 Other code...
             */
             AnalysisScope scope = AnalysisScopeReader.readJavaScope(
                     "scope.txt", /*Path to scope file*/
                     exclusionFile, /*Path to exclusion file*/
                     classLoader
             );
             scope.addClassFileToScope(ClassLoaderReference.Application, file
     /*Class file instance.*/);
         }

     }
     ```

   - 构造类文件实例的时使用**绝对路径。**

3. 有些生产方法看起来没有和某些测试方法无关，为何这部分测试方法也被选中了？

   A： 方法之间可能存在**间接依赖。**例如有测试方法`t()`，生产方法`m1()`和`m2()`。其中：t调用了m1，而m1（隐式的）调用了m2，如果此时m2发生变更，t也会被选中。所以在实现测试选择之前需要对调用图进行处理，例如：**变更传递**，或者生成**生产代码**与**测试代码**映射关系。

4. 如何获取被调用者（Callee）和调用者（Caller）？

   A：具体的实现方法有很多。根据WALA的官方文档描述，可以通过`getCallSite`（大概是这个方法）获取调用点，同学们可以自行探索。我个人实现的方法是使用`CallGraph`接口定义的`getPredNodes`或`getSuccNodes`，通过获取前驱和后继的方式获取某个方法的调用者和被调用者。

5. 如何生成.dot文件？需要使用WALA仓库给出的`DotUtil`么？

A：.dot文件就是一种用于生成图的标记型语言，借助它可以更好的调试生成的图结构，其本身和WALA没有什么直接联系。`DotUtil`只是WALA开发者的实现。所以，只要能够生成符合.dot语法的文件，再使用命令行生成pdf就可以满足作业要求。推荐大家参照`0-CMD`中给出的dot语法来自行实现。

6. 听说把README里面的代码扒下来作业就完成了一半了？

   A：README里面给出的代码部分能够用于构建调用图，提供了处理WALA调用图的一种思路。在这个基础上，实现测试选择需要进一步裁剪调用图，以细化获取到的信息。本次大作业的主要目的是实现作业描述里面提到的功能，所有可行的实现都可以。但要注意：**代码风格**（规范程度、原创性）是本次大作业的重要评分标准。

2020-11-15

1. 为何使用了`exclusion.txt`文件生成的之后，调用图还是很大？

   - `exclusion.txt`文件的作用：`exclusion.txt`用于排除一些与本次程序分析无关的java类。很多java原生类（一般通过Primordial加载器加载）和我们分析的目标关系不大，例如`sun.swing.*`和`sun.awt.*`，但是又很容易在加载原生类的时候加载进来，所以需要使用`exclusion.txt`文件将其排除在外。针对java的分析是不可能完全脱离原生类进行的，比如`java.lang.*`提供了很多java语言的基本特性，是不能被排除在外的。
   - `wala.properties#java_runtime_dir`的含义：`java_runtime_dir`的主要作用是为WALA指明一个能够找到java原生类的路径。他会递归地遍历所给的文件夹路径，并把目录下所有的.class和.jar加载进来，作为本次分析的原生类。
   - **供参考的优化方式**：根据多位同学的反馈，发现本次大作业的程序分析部分其实仅依赖rt.jar文件中的原生类，大家可以解压看一下这个jar包中的内容。因此一个可能的优化方式是：
     - 找到rt.jar（一般在`{JAVA_HOME}/jre/lib`下），将其复制到resources目录（和wala.properties同一级目录）中
     - 配置`wala.properties#java_runtime_dir`的值为`java_runtime_dir = ./src/main/resources`
     - 重新运行代码，查看是否运行正常

     上述做法通常可以削减调用图的规模。推荐大家都尝试一下，因为后面可能会要求大家统一这样做（方便我们给大作业打分）。

2. 一些自调试手段

   调试手段以控制台输出为主要方法，同学们也可以打断点进行view。

   - 输出分析域实例，查看需要的类是否正确加载

     ```
     AnalysisScope scope = /* Some code to get analysis scope */
     System.out.println(scope);
     ```

     样例输出：

```
Primordial
 JarFileModule:C:\Users\QRX\IdeaProjects\test-selection-demo-
wala\src\main\resources\rt.jar
Extension
Application
 ClassFileModule:C:\Users\QRX\IdeaProjects\test-selection-demo-
wala\material\0-CMD\target\classes\net\mooctest\CMD.class
 ClassFileModule:C:\Users\QRX\IdeaProjects\test-selection-demo-
wala\material\0-CMD\target\test-classes\net\mooctest\CMDTest.class
 ClassFileModule:C:\Users\QRX\IdeaProjects\test-selection-demo-
wala\material\0-CMD\target\test-classes\net\mooctest\CMDTest1.class
 ClassFileModule:C:\Users\QRX\IdeaProjects\test-selection-demo-
wala\material\0-CMD\target\test-classes\net\mooctest\CMDTest2.class
 ClassFileModule:C:\Users\QRX\IdeaProjects\test-selection-demo-
wala\material\0-CMD\target\test-classes\net\mooctest\CMDTest3.class
Synthetic
Exclusions: (apple\/.*)|(com\/apple\/.*)|(com\/ibm\/.*)|
(com\/oracle\/.*)|(com\/sun\/.*)|(dalvik\/.*)|(java\/beans\/.*)|
(java\/io\/ObjectStreamClass*)|(java\/rmi\/.*)|(java\/text\/.*)|
(java\/time\/.*)|(javafx\/.*)|(javafx\/beans\/.*)|
(javafx\/collections\/.*)|(javafx\/scene\/.*)|
(javax\/accessibility\/.*)|(javax\/activation\/.*)|
(javax\/activity\/.*)|(javax\/annotation\/.*)|(javax\/crypto\/.*)|
(javax\/imageio\/.*)|(javax\/jnlp\/.*)|(javax\/jws\/.*)|
(javax\/management\/.*)|(javax\/net\/.*)|(javax\/print\/.*)|
(javax\/rmi\/.*)|(javax\/script\/.*)|(javax\/smartcardio\/.*)|
(javax\/sound\/.*)|(javax\/sql\/.*)|(javax\/tools\/.*)|(jdk\/.*)|
(netscape\/.*)|(oracle\/jrockit\/.*)|(org\/apache\/xerces\/.*)|
(org\/ietf\/.*)|(org\/jcp\/.*)|(org\/netbeans\/.*)|(org\/omg\/.*)|
(org\/openide\/.*)|(sun\/.*)|(sun\/awt\/.*)|(sun\/swing\/.*)
```

说明

1. `Primordial`：加载到的Java原生类（所在的jar包）
2. `Application`：和本次分析有关的java类文件
3. `Exclutions`：`exclusions.txt`文件中记录的、需要排除在外的类（支持通配符）

- 遍历类层级实例，查看待分析的类是否正确加载

```
AnalysisScope scope = WalaUtil.getDynamicScope(
    classDir, exPath, FunctionTest.class.getClassLoader()
);
ClassHierarchy cha = ClassHierarchyFactory.makeWithRoot(scope);
for (IClass iClass : cha) {
    if(iClass.toString().contains("Application"))
        System.out.println(iClass);
}
```

样例输出：

```
<Application,Lnet/mooctest/CMDTest>
<Application,Lnet/mooctest/CMDTest1>
<Primordial,Ljava/lang/ApplicationShutdownHooks>
<Application,Lnet/mooctest/CMD>
<Application,Lnet/mooctest/CMDTest2>
<Primordial,Ljava/lang/ApplicationShutdownHooks$1>
<Application,Lnet/mooctest/CMDTest3>
<Primordial,Ljava/awt/ModalEventFilter$ApplicationModalEventFilter>
```

可以看到：和分析相关`CMDTest*`以及`CMD`都被正确加载

⚠PS：`WalaUtil`是我自定义的一个工厂类，不是WALA自带的内容

- 利用`CallGraphStats`输出图的统计信息，查看图的大小

```
CHACallGraph chaCG = makeCHACGFromScope(scope);
// Test CallGraphStats
String stats = CallGraphStats.getStats(chaCG);
System.out.println(stats);
```

样例输出

```
Call graph stats:
  Nodes: 29171
  Edges: 362330
  Methods: 30665
  Bytecode Bytes: 1743572
```

输出结果可以从一定程度上反映图的大小