

FAQ

提供一些常见问题的回答，如有问题请联系作者：grx@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\jnp\/*)|(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\opendide\/*)|(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. **Exclusions**: `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

2020-11-16

1. 实例数据里面给出的依赖图的有向边表示被依赖（被调用）关系，我的有向边表示依赖（调用）关系可以么？

A: 可以，实现规定的功能即可。

2. 经典大作业需要交报告么？我看到示例仓库里面有 `/Report` 目录。

A: 经典大作业 `/Report` 目录下放置生成的报告图，好好阅读作业要求啊亲

3. 选取结果和给出的示例不同但是内容一致，有影响么？

A: 没有影响，对应依赖图校对选取结果即可。

4. 使用 `dot` 命令行生成调用图没有报错，但是生成的pdf没有内容，怎么回事？

A: 可能是PDF阅读器的问题，使用chrome打开尝试一下。

5. 如何判定一个方法为测试方法？

提示：字符串匹配是一种可行的方法，但是精度较差。可以借助Java语言的结构来辅助判定，如：注解。

2020-11-17

将rt.jar打包进测试选择jar包，使其不需要依赖机器上的JAVA_HOME

根据最近一段时间同学们的反馈和反复调试，发现 `wala.properties#java_runtime_dir` 指向的并非机器上的 `JAVA_HOME`，而是一个放置了java原生类的文件夹。java原生类通常打包在jar中，以java8为例：WALA需要依赖 `{JAVA_HOME/jre/lib}` 下的 `rt.jar`，其中放置了包括 `java.lang.*` 在内的许多基础类，比如 `java.lang.Object`，它是所有类型的父类。

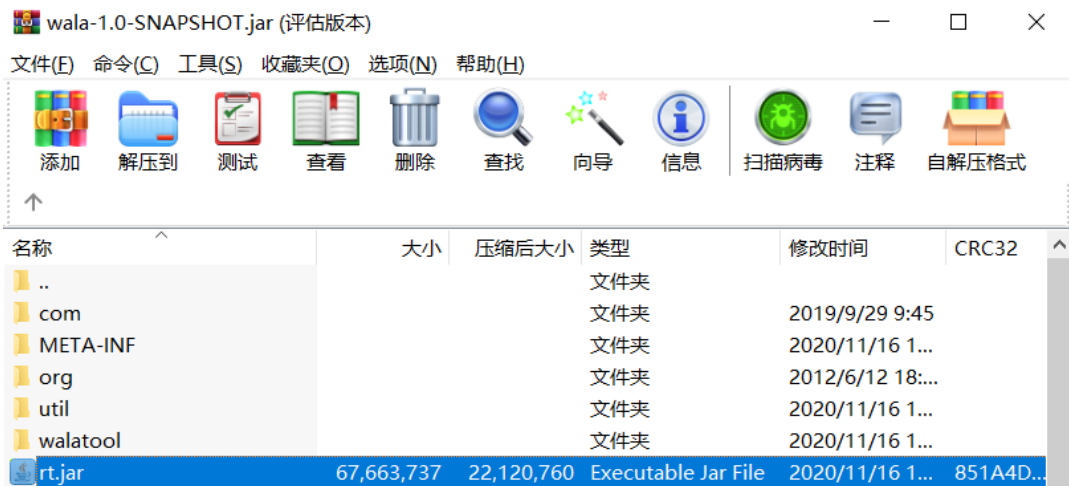
基于以上发现，为了方便我们后续调试和计分，现在决定让大家统一采用将 `rt.jar` 打包到 `testSelection.jar`（**注意**：这样就不用按照之前大作业的要求，将 `wala.properties#java_runtime_dir` 配置为 `C:/Program Files/Java/jdk1.8.0_231` 了），请大家按照以下步骤操作：

- 到 `{jdk8_dir}/jre/lib` 下找到 `rt.jar`，将其复制到 `{project_dir}/src/main/resources` 下；
- 修改 `wala.properties#java_runtime_dir = .`，让WALA可以从当前目录开始递归寻找java原生类；
- 在IDE内调试。保证项目可以在IDE中可以正常运行，且没有出现类似下面的警告：

```
WARNING: java_runtime_dir /opt/Java/jdk1.8.0_171/jre/lib in wala.properties
is invalid. Using boot class path instead.
```

如果没有出现警告且运行成功，则配置成功。在调试过程中如果出现其他问题，请及时反馈；

- 按照以下步骤进行打包调试：
 1. 检查 `testSelection.jar` 包内容：Windows可以使用WinRAR查看jar包内容。确保可以在根目录下找到 `rt.jar`，例如：



2. 将 `testSelection.jar` 包复制到一个空文件夹下。这一步是防止WALA递归地搜索 `rt.jar`，从而读入其他位置的 `rt.jar` 造成误判；
3. 运行jar包，保证jar包能够正常运行规定的命令行。

如果在调试过程中出现其他问题，请及时反馈；

PS：根据同学们反馈的情况，所给的项目在分析的时候只需要 `rt.jar` 包。上述打包流程也已经验证过，亲测可以实现 `testSelection.jar` 的跨平台运行。但是不排除其他同学在调试过程中出现依赖缺失的问题，这里提供解决思路：

- 查看wala报错。依赖缺失的格式通常类似：`failed to root <Primordial, Ljava/lang/Object;> of class hierarchy`。`Ljava/lang/Object;` 是缺失类的内部名称，这里对应 `java.lang.Object`；
- 搜索、查看缺失的类在哪个jar包下，如：`charsets.jar`、`jce.jar`、`dt.jar`；

- 到 {jdk8_dir}/jre/lib 下找到对应的jar包，将其复制到 {project_dir}/src/main/resources 下，重新调试。**如果还是有问题，请及时反馈。**