资料来源

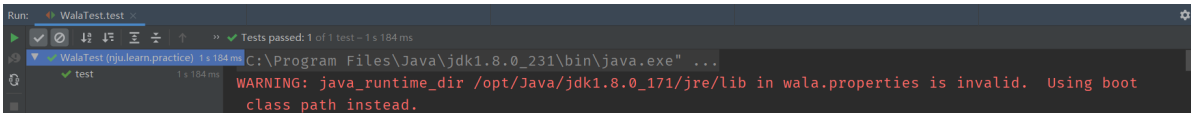
- github仓库：https://github.com/wala/WALA
- 旧版Wiki：http://wala.sourceforge.net/wiki/index.php/Main_Page
- 指南：`PLDI_WALA_Tutorial.pdf`

配置文件

主要的配置文件有三个：`wala.properties`、`scope.txt`、`exclusions.txt`。以maven项目结构为例：`wala.properties`**必须**放在`/src/main/resources`下，`scope.txt`和`exclusion.txt`推荐放在`/src/main/resources`下。配置文件的样例在`配置文件样例/`目录下。

`wala.properties`

- wala属性文件，记录了运行wala所需要的一些环境配置。其中最重要的属性是`java_runtime_dir`。请同学们将该属性修改成自己的jdk文件夹路径
- ⚠**注意**：在IDEA开发环境中运行时，有可能会出现`java_runtime_dir`配置错误、但是程序依然能够运行的情况。这个时候控制台一般会抛出WARNING，但是打成jar包后可能不能正常运行。如下图所示，请同学们注意



`scope.txt`

- 域配置文件，与静态分析的范围`AnalysisScope`有关。WALA提供了**读取配置文件**和**直接添加类文件进入分析域**两种构建分析域的手段，`scope.txt`就和第一种方式相关，详细教程见：http://wala.sourceforge.net/wiki/index.php/UserGuide:AnalysisScope
- WALA的开发团队虽然声称能够支持源代码的程序分析，但实际应用中基本上都是使用字节码文件（.class）作为静态分析的材料，这和Java语言本身的特性有关。本项目要求同学们以字节码为分析材料
- 推荐结合两种方式构建分析域，文档的后面会进一步说明

`exclusion.txt`

- 排除配置文件，与静态分析的范围`AnalysisScope`有关。改文件记录了一些静态分析中**不关心**的类。文件每行一条记录，支持正则表达式，表示在本次静态分析中排除在外的类
- ⚠**注意**：静态分析的计算复杂度较高，通过`exclusion.txt`可以有效地缩减分析域、提高分析效率。但排除过多的类可能会丢失一些分析必须的类，导致抛出异常。因此推荐同学使用`makeWithRoot`的方式构建类层次对象，文档的后面会进一步说明

操作指南

该部分主要讲解一些推荐的WALA使用方式，该部分使用的wala API版本为1.5.4

生成分析域

推荐同时使用两种方式综合构建`AnalysisScope`对象，涉及到两个方法两个步骤：

1. `AnalysisScopeReader.readJavaScope(scopePath, new File(exPath), classLoader);`

   保持配置样例中`scope.txt`的内容不变，该方法能够返回一个只包含Java原生类的分析域，并排除一些不常用的原生类（如：`sun.awt.*`）

2. `scope.addClassFileToScope(ClassLoaderReference.Application, clazz);`

   `scope`是一个`AnalysisScope`对象，`clazz`是一个我们想要加入分析域的类文件对象。这行代码能够将我们想要分析的类动态地加入到分析域中。

生成类层次

类层次生成方法由工厂类`ClassHierarchyFactory`提供，以分析域对象为构建原料，这里推荐使用`makeWithRoot`方法构建类层次对象：

```
ClassHierarchy cha = ClassHierarchyFactory.makeWithRoot(scope);
```

在缺失了某些分析所需的类时，`makeWithRoot`方法会尽可能地为依赖这些缺失类的方法添加“Root”，即认为`java.lang.Object`为这些类的父类。

确定进入点

进入点和我们分析的兴趣的有关，以分析域和类层次对象为构建原料。WALA内置了若干种进入点集合，如：

- 针对主程序生成进入点：`Util.makeMainEntryPoints`
- 针对所有Application类（非原生类）生成进入点：`new AllApplicationEntrypoints(scope, cha)`

此外，还支持自定义进入点，可以通过实现自己的进入点子类实现。为了降低实现难度，这里推荐大家使用`AllApplicationEntrypoints`构建进入点。

构建调用图

本项目要求完成两种粒度的依赖分析（同时这也是实现两种粒度的测试选择的基础），而依赖分析通常和构建调用图有关，这里推荐两种构建调用图的方法：

1. `CHACallGraph`：使用类层次分析（Class Hierarchy Analysis）算法构建调用图。该方法构建的调用图精度较低，但是速度较快，构建例子如下：

   ```
   CHACallGraph chaCG = new CHACallGraph(cha);
   chaCG.init(new AllApplicationEntrypoints(scope, cha));
   ```

2. 使用0-CFA算法，通过上下文无关的方式构建调用图。相比于第一种方法构建的调用图，该方法构建的调用图精度更高，但同时速度更慢。构建例子如下：

```
ClassHierarchy cha = ClassHierarchyFactory.makeWithRoot(scope);
AllApplicationEntrypoints entrypoints = new AllApplicationEntrypoints(scope,
cha);
AnalysisOptions option = new AnalysisOptions(scope, entrypoints);
SSAPropagationCallGraphBuilder builder = Util.makeZeroCFABuilder(
    Language.JAVA, option, new AnalysisCacheImpl(), cha, scope
);
```

此外，WALA还支持有很多种其他调用图构建算法，如RTA、1-CFA等。同学们也可以查阅资料学习后自行实现。

联系我们

如有未尽事宜，请联系我：钱瑞祥 qrx@smail.nju.edu.cn