AutomatedTesting2020

选题方向：经典自动化测试方向

算法流程

- 从外界接收到源代码路径，变更信息之后，程序会检验命令的合法性，并且做统一的预处理。
- 第一步会生成class级别和method级别的dot文件，读取.class文件采取递归的手段，接下来分析代码，生成method级别依赖图，该部分的实现基于wala的API接口。通过内置的itreator迭代器，结合点遍历，完成所有依赖信息的存取。生成class的依赖信息方法如下：基于前一步生成的method依赖文件，按照特定的结构，抽取出class名。
- 然后根据命令的参数不同，选出被选择的类/方法。寻找被影响的方法/类时，思路的灵感来自于利用队列生成杨辉三角，下一轮被找到的受影响项，均来自于上一轮被新加入集合的受影响项。

设计的类

- Influx：程序的入口，会在该类中调用其他类，来实现完整的业务逻辑
- WalaConnector：在这个类中，包含了所有对wala的调用。通过wala的API生成图，最后生成dot文件。包含class/method两种粒度，其中，class的dot文件由method的dot文件生成。主要由createMethodDotFile，createClassDotFile两个方法构成。
- TestSelector：在这个类中，完成了对测试用例的选择，包含class/method两种粒度。由createSelectClass，createSelectMethod两种方法构成。
- Util：工具类，包含：文件递归搜索方法getAllFile()，test类筛选方法ifTargetClass()， 路径预处理方法normalize()，变更信息读取方法readChangeInfo()，
- Property：仿配置文件类，存放所有静态信息和部分配置信息
- Autotest：结果检验脚本类。自动化测试程序结果和目标结果是否一致。考虑到并未遵循JavaTest类的标准设计模式，并未放入Test文件夹

巧妙构思

- 寻找被影响的方法/类时，最终的边界条件是不动点的确定，即新一轮的集合和上一轮的集合相同，用一个bool变量即可判断这一点。可以证明，最大迭代次数为依赖关系中最长的路径的长度。曾经考虑，完全使用队列的进出来完成这一点，每一次队列中的受影响项，都仅来自于上一轮。这样每一轮遍历所考察的点的数目都会变少。但是，该方法遇到代码依赖中有环的时候，将会陷入死循环。所以不动点的引入是必须的。不动点的使用至少有两种思路，第一，把维护的队列放任一个只进不出的set，包含历史上所有受影响项。第二，单独维护一个队列和一个包含历史上所有受影响项的集合。前者更为直观，空间复杂度略低。后者的时间复杂度略低。现详细证明两种方案的时间复杂度。

  两种方案的迭代次数都是相同的，每一次新加入集合的受影响项的数目也是相同的。不妨设，依赖表有m个条目，总共迭代n轮，最初集合中有a_0个项，每一轮新加入的项数目为$a_1, a_2 \dots a_i \dots, a_{n-1}, a_n = 0$，方案1的时间复杂度为 $(\Sigma_{i=0}^{n}(a_i \times i)) \times m$ 方案2的时间复杂度为$(\Sigma_{i=0}^{n} a_i) \times m + (\Sigma_{i=0}^{n}(a_i \times i))$

  可以看到，若依赖表中的数目较小，采用两种方案的时间不会相差太大，但如果m的数值普遍较大，采取第二种方案更为合理。考虑到，这次的示例中，m普遍较小。本次作业采取方案一。

- 考虑到本次作业中大量的文件操作，采取了单键模式，在减少内存开销的同时，防止对文件资源的多重占用。

- 为了提高程序的可演化能力，本次作业中，已尽可能的将职责分离，设计出不同的类来承担不同的职责。可以看到，常量或报错信息等，均用了property类来维护。虽然还未做到完全的由配置文件来管理程序，但也初具雏形。
- AutoTest类算是一个小彩蛋，作为对课程的致敬。这个类是我用来自动化测试我的程序结果和参考答案是否完全一致的自动化脚本类。