

Mull简易入门指南

Mull是一个实用的开源C/C++变异测试工具，具有以下特性：

- 支持多种类型的变异算子
- 支持多种格式的测试结果
- 支持并行执行测试用例
- 支持增量变异测试

本文将以libxml2为例简要介绍Mull的使用流程，并分享笔者在使用Mull的过程中遇到的坑。

1.快速上手

安装

Mull的安装配置非常简单，参考官方文档即可。

生成变异程序

编写配置文件（可选）

使用Mull指定的编译选项对目标程序进行编译时，Mull默认在当前目录下寻找名为mull.yml的配置文件，若未找到，则会递归地在父级目录搜索，直到文件系统的根目录为止。此外，也可以设置环境变量**MULL_CONFIG**来指定配置文件的路径。

一个配置文件示例如下：

```
mutators: # 使用的变异算子，参考官方文档"Supported Mutation Operators"页面
- cxx_add_to_sub
- cxx_logical
excludePaths: # 被指定的路径下所有文件不会产生变异体，支持正则表达式，也可以直接写需要排除的代码文件的路径+文件名。参考官方文档Tutorials/Keeping #mutants under control/File Path Filters
- gtest
- gmock
timeout: # 设置每个变异体的超时时间，默认单位为毫秒
- 10000 # 10 seconds
quiet: false # 静默模式开关，若设为true，则编译时不会输出编译日志
```

编译

本文默认读者已经掌握了C/C++编译以及构建工具使用的相关知识。使用Mull编译源代码生成变异体程序的步骤非常简单，以ubuntu18.04下编译libxml2为例：

```
git clone https://gitlab.gnome.org/GNOME/libxml2.git
cd libxml2

export CC=clang-10 # 必须使用clang编译, 且clang版本必须对应Mull版本, 例如Mull-10对应clang-10
export CFLAGS="-O1 -fexperimental-new-pass-manager -fpass-plugin=/usr/lib/mull-ir-frontend-10 -g -grecord-command-line" # Mull的编译插桩
./autogen.sh
./configure --without-python --with-threads=no --with-zlib=no --with-lzma=no --prefix="构建产物存储路径"
make clean
make install
```

总结一下, 使用Mull生成变异体程序只需满足两点要求:

(1) 使用与Mull版本对应的clang作为编译器进行编译, 若使用构建系统如Autotool或Cmake, 只需提前设置环境变量CC或CXX

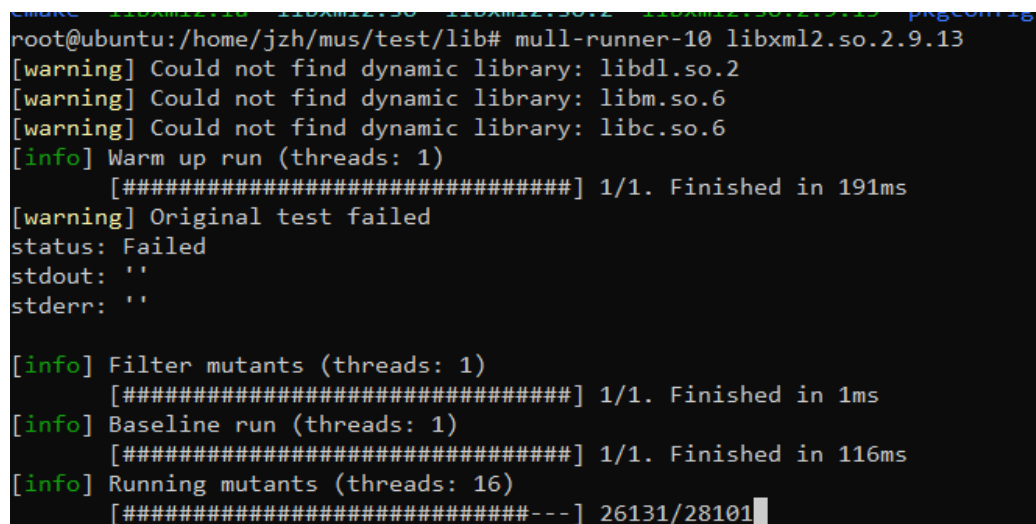
(2) 使用Mull指定的编译参数, 若使用构建系统如Autotool或Cmake, 只需提前设置环境变量CFLAGS或CXXFLAGS

运行

Mull生成的变异体程序必须使用Mull的伴生工具mull-runner运行, mull-runner执行变异测试的流程如下: 为mull-runner指定一个二进制程序作为变异测试入口-mull-runner探测该可执行程序中包含的变异体-多轮循环运行该二进制程序, 每次运行激活一个变异体-生成变异测试报告。

以mull-10为例, 使用mull-runner执行变异测试的命令如下:

```
mull-runner-10 target_binary [options]
# 我们可以使用该指令验证libxml2的变异插桩编译是否成功
mull-runner-10 $INSTALL_PREFIX/lib/libxml2.so.2.9.13
# 运行截图如下
```



```
root@ubuntu:/home/jzh/mus/test/lib# mull-runner-10 libxml2.so.2.9.13
[warning] Could not find dynamic library: libdl.so.2
[warning] Could not find dynamic library: libm.so.6
[warning] Could not find dynamic library: libc.so.6
[info] Warm up run (threads: 1)
[#####] 1/1. Finished in 191ms
[warning] Original test failed
status: Failed
stdout: ''
stderr: ''

[info] Filter mutants (threads: 1)
[#####] 1/1. Finished in 1ms
[info] Baseline run (threads: 1)
[#####] 1/1. Finished in 116ms
[info] Running mutants (threads: 16)
[#####---] 26131/28101
```

所有options可参考: [mull-runner — Mull 0.18.0 documentation](https://mull-project.org/docs/mull-runner.html)

下面总结一些常用options:

--workers *number*: mull-runner使用的线程数

--timeout *number*: 每轮测试循环的超时时间, 默认单位为毫秒

--reporters *reporter*: 测试报告类型，常用的有IDE（直接输出至标准输出）和SQLite（以SQLite数据库文件存储）

--report-name *filename*: 测试报告文件名

--report-dir *directory*: 测试报告存储路径

--no-output: 屏蔽二进制程序本身的标准输出和标准错误

2.踩坑总结

1.**Mull版本差异**：使用Mull-10和Mull-12生成变异体时两者的优化等级存在差别，Mull-10必须使用-O1，Mull-12必须使用-O0，否则会报错。

Note

For Clang 9, 10, and 11 also pass `-O1`, otherwise the plugin won't be called.

2.**-grecord-command-line的使用**：如果在一条指令中同时编译多个文件（当然最好避免这样的做法），请移除编译参数-grecord-command-line

Note

`-grecord-command-line` doesn't currently work if you compile several files in one go, e.g.
`clang a.c b.c c.c`. In this case, please remove the flag.

3.**Docker内使用mull-runner**：在Docker内使用mull-runner会严重降低变异测试运行速度，请尽量避免在Docker内运行变异测试。

4.**SQLite文件大小**：SQLite数据库文件会记录每轮测试循环中二进制程序的标准输出和标准错误，当标准输出和标准错误内容较多时会导致该数据库文件过大，可以使用--no-output选项屏蔽二进制程序的标准输出和标准错误。