[如何用 Valgrind 检测使用 LuaJIT FFI 过程中的内存泄漏](https://segmentfault.com/a/1190000008226235)

* [valgrind](https://segmentfault.com/t/valgrind/blogs)

* [luajit](https://segmentfault.com/t/luajit/blogs)

* [ffi](https://segmentfault.com/t/ffi/blogs)

[**spacewander**](https://segmentfault.com/u/spacewander) 1月29日发布

* 推荐 **0** 推荐
* 收藏 **0** 收藏，46 浏览

什么情况下可能会有内存泄漏

给带 GC 的语言写 C binding 一向是件让人迷糊的事。到底应该在 C 手工释放资源呢，还是依靠 GC 来回收？  
还好 LuaJIT FFI 提供了很好用的 ffi.gc 方法。该方法允许给 cdata 对象注册在 gc 时调用的回调，它能让你在 Lua 领域里完成 C 手工释放资源的事。

C++ 提倡用一种叫 RAII 的方式管理你的资源。简单地说，就是创建对象时获取，销毁对象时释放。我们可以在 LuaJIT FFI 里借鉴同样的做法，在调用 resource = ffi.C.xx\_create 等申请资源的函数之后，立即补上一行 ffi.gc(resource, ...) 来注册释放资源的函数。尽量避免尝试手动释放资源！即使不考虑 error 对执行路径的影响，在每个出口都补上一模一样的逻辑会够你受的（用 goto 也差不多，只是稍稍好一点）。

有些时候，ffi.C.xx\_create 返回的不是具体的 cdata，而是整型的 handle。这会儿需要用 ffi.metatype 把 ffi.gc 包装一下：

local resource\_type = ffi.metatype("struct {int handle;}", {

\_\_gc = free\_resource

})

local **function** free\_resource(handle)

...

end

resource = ffi.new(resource\_type)

resource.handle = ffi.C.xx\_create()

回到小标题，如果你没能把申请资源和释放资源的步骤放一起，那么内存泄露多半会在前方等你。写代码的时候切记这一点。

在单元测试中检查内存泄漏

当然要想保障代码里不存在内存泄露，严格按照 RAII 规范编写代码并不够。毕竟圣人千虑，必有一失；何况你我凡胎？显而易见，我们需要一个侦测内存泄漏的工具。在这方面首选 Valgrind。

Valgrind 只能检查程序运行路径上的内存问题。所以要想最大化 Valgrind 检查的覆盖面，最好结合单元测试一起跑。这样单元测试覆盖到的地方，内存检查也能覆盖到。

鉴于 OpenResty 在这方面提供了一套工具集，而且我写这篇文章也是为了解决 OpenResty 应用开发中的一些问题，所以请允许我先以 OpenResty 应用为例，说说如何预防内存泄漏。

TEST\_NGINX\_USE\_VALGRIND=1

OpenResty 官方的测试框架 test-nginx 内置了对 Valgrind 的支持。你所需的，不过是加个 TEST\_NGINX\_USE\_VALGRIND=1 环境变量。测试框架看到该环境变量的存在后，会在启动 Nginx 的时候，前面加上 valgrind --leak-check 等选项。这样 Valgrind 就会去检查 Nginx 内部的内存分配。一旦 FFI 调用中存在内存泄漏，Valgrind 便会报告出来。效果与用 Valgrind 运行一个普通的二进制程序无异。

$opts = "--tool=memcheck --leak-check=full --show-possibly-lost=no";

**if** (-f 'valgrind.suppress') {

# 如果 valgrind.suppress 存在，用它来消除警告

$cmd = "valgrind --num-callers=100 -q $opts --gen-suppressions=all --suppressions=valgrind.suppress $cmd";

} **else** {

$cmd = "valgrind --num-callers=100 -q $opts --gen-suppressions=all $cmd";

}

由于 Valgrind 会显著拖慢托管程序的运行速度，你通常还需要另一个环境变量 TEST\_NGINX\_SLEEP 设置 test-nginx 测试框架的超时时间，以免遭遇各种奇怪的错误。最后完整可用的运行方式如下：

TEST\_NGINX\_USE\_VALGRIND=1 TEST\_NGINX\_SLEEP=1 prove -r t

实际运行一下，你会发现输出来的“错误”非常多，甚至可能会出现尴尬的内容：

==10898== More than 1000 different errors detected. I'm not reporting any more.

==10898== Final error counts will be inaccurate. Go fix your program!

不用担心！大部分都是 faise positive（假阳性）。你只需弄一个 valgrind.suppress 来消除错误。由于我们只关注内存泄漏问题，这里简单粗暴地关闭其他错误输出：

{

<insert\_a\_suppression\_name\_here>

Memcheck:Cond

obj:\*

}

...

还有一类 Nginx 或 LuaJIT 相关的内存泄漏报告，我们可以把它们也一并消除掉：

{

<insert\_a\_suppression\_name\_here>

Memcheck:Leak

fun:malloc

fun:ngx\_alloc

}

...

现在再跑一次测试，如果还有报错，应该就是你的 FFI 代码问题了。背景噪音消除了，问题排查就清晰多了。

注意默认情况下 Valgrind 的检测结果不会影响退出码，所以为了跟 CI 配合，需要 grep 一下具体的报错：

TEST\_NGINX\_USE\_VALGRIND=1 TEST\_NGINX\_SLEEP=1 prove -r t 2>&1 | grep -B 3 -A 20 "match-leak-kinds: definite"

test $? && exit 1

这样一旦 Valgrind 报告中出现了 "match-leak-kinds: definite" 字眼，测试就会失败。

非 test-nginx 下的内存泄漏检测

如果用的不是 test-nginx 那一套，又该怎么检测内存泄漏呢？

我们可以照搬 test-nginx 的原理，加塞 Valgrind 参数进去。比如，如果测试集只依赖 LuaJIT 本身，你可以这么运行：

**opts**="--tool=memcheck --leak-check=full --show-possibly-lost=no --error-exitcode=42"

**valgrind** --num-callers=100 -q $opts --gen-suppressions=all [--suppressions=valgrind.suppress] luajit ...

不像 test-nginx，这里不再需要 grep 一下。通过指定 --error-exitcode，一旦 Valgrind 发现了错误，会以指定的错误码退出。

如果测试集基于 resty 命令行工具驱动，可以用 resty 的 --valgrind 选项。

如果测试集基于 busted 测试框架，可以改造下调用方式。

首先，创建一个 test\_valgrind.lua 文件，绕过 luajit -e 无法传参的缺陷。

**require** "busted.runner"({ **standalone** = false })

然后用 Valgrind 运行 luajit：

valgrind --error-exitcode=42 --tool=memcheck \

--gen-suppressions=all --suppressions=valgrind.suppress \

luajit test\_valgrind.lua .