云风的 BLOG

思绪来得快去得也快，偶尔会在这里停留

« 怎样在运行时插入运行一段 Lua 代码 | 返回首页 | skynet 1.0 发布计划 »

在线调试 Lua 代码

一直有人问，如果调试 skynet 构件的服务。

我的简单答案是，仔细 review 代码，加 log 输出。长一点的答案是，尽量熟悉 skynet 的构造，充分利用预留的监控接口，自己编写工具辅助调试。

之前的好多年，我也写过很多 lua 的调试器，这里就不一一翻旧帖了。今天要说的是，我最终还是计划加入 1.0 正式版的调试控制台。

也就是单步跟踪调试单个 lua coroutine 的能力。这对许多新手来说是个学走路的拐杖，虽然有人一辈子都扔不掉。

一开始我想实现两种模式：冻结住整个服务，慢慢跟踪调试；以及不破坏服务处理其它消息的能力，单步调试单条消息的处理流程。

冻结模式的好处是，在调试过程中，服务的整个 lua 虚拟机是挂起的，所以其内部状态是不变的。而如果运行在调试过程还可以处理其它消息，那么内部状态可能就变来变去了。

但坏处也是很明显的，如果在线上环境，如果是关键服务，可能很快就让整个系统过载（处理消息的速度远远低于正常水平，做高并发状态下，冻结一两秒都是致命的）。

我最后放弃了冻结模式。因为如果是开发器，你的系统同时可能就处理几条消息，你也只会调试专心编写的部分。所以状态改变的影响是很小的。如果有，也可以多加一些 log 来提示。而不打断对外服务能力的调试方式看起来要舒服的多。

当然，为实现这个调试器需要做的一些 C 层的基础设施还是按可以满足两种需求来做的。万一有一天需要冻结模式调试，也方便加上。

底层主要是实现一个额外的通讯管道，不走 skynet 的 message queue 。这样才方便绕开 skynet 的服务间通讯机制来调试服务本身。

为了可以单步跟踪，我们还需要稍微改造一下 lua 自带的 debug hook ，让其可以在 hook 中 yield 出来。lua 的 C debug api 本身是支持的，但是 lua 版 api 屏蔽了这个特性。（这里有个坑，可能是 lua 实现的 bug ，后面我会谈谈）

有了基础设施后，我们就可以用 lua 愉快的搭建调试工具了。

用户的入口可以是 debug_console 。但每次需要调试一个 lua 编写的服务时，可以单独启动一个调试服务利用前面所述的通讯管道和被调试服务通讯。被调试服务可以在每个消息进来时，处理它之前留个钩子，一旦发现正被调试，就取出用户的操作指令运行。

由于 skynet 的服务在没有消息处理时是完全挂起的，所以一旦想调试一个服务，还必须给它安一个定时器定期唤醒检查调试管道（因为调试管道的消息不经过 skynet 的消息调度）。我暂时设置的是一秒检查 100 次。通过一个调试指令动态开关。

由于 skynet 的服务就是消息驱动的，所以我加了一条叫 watch 的指令，可以监控某一类消息，并可以附加一个条件函数，在条件满足时才中断下来。

利用前两天编写的 代码注入模块 ，我们可以在消息处理流程中断下来后，观察和改变它的环境。

在 skynet 的 lua53 分支上，我已经提交了调试器的代码。有兴趣的同学可以玩玩。目前还只提供了一些基本特性，以后可能加上更多东西。

使用方法大致是，在启动了 debug console 服务时（默认的 example config 配置启动了它），使用 telnet 连接上调试端口（通常是 127.0.0.1:8000）。

使用 debug address 来 attach 入要调试的服务，address 是服务地址。

这个时候，会出现命令行提示符。当没有 watch 任何消息时，提示符是当前服务的地址。

任何使用输入 cont 都会脱离调试状态。

此刻，上下文在这个服务的主线程中。你可以随意输入一些合法的 lua 表达式或 lua 指令运行。也可以调用 watch(proto, cond) 函数来加一个断点。

watch 的第一个参数是一个字符串，表示你想关注的协议名，一般是 "lua" 。

第二个参数是一个可选参数，它是一个函数，参数会传入当前消息的参数。如果你返回 true 表示关注这条消息。不写第二个参数表示只要协议类型匹配上即可。

每次 watch 只对一条消息有效。

如果匹配到关注的消息，消息处理流程会被挂起。提示符会变成停下来的源文件名以及行号。

除了可以输入 lua 表达式以及 lua 指令外（输入的语句会放在当前位置执行），还可以用三条特殊指令：

c 表示继续处理这条消息，离开关注状态。

s 表示单步运行一行，如果是函数调用，会跟踪进去。

n 表示单步运行一行，如果有函数调用，不会跟踪进去。

例如，你可以用 ./skynet example/config 启动一个简单的 skynet 进程。如果你没有修改过配置，这个使用会启动一个叫 simpledb 的服务。

接下来，你可以使用 nc 127.0.0.1 8000 接入调试控制台。正确接入的话，会看到

Welcome to skynet console

这行字。

如果你 list 的话，可以看到所有服务：

```
:01000004       snlua cmaster
:01000005       snlua cslave
:01000007       snlua datacenterd
:01000008       snlua service_mgr
:0100000a       snlua console
:0100000b       snlua debug_console 8000
:0100000c       snlua simpledb
:0100000d       snlua watchdog
:0100000e       snlua gate
```

我们可以用 simpledb 这个服务做实验。注意：目前仅限于调试同一进程内的服务。（这个限制是因为实现者特别懒）

输入 debug c 或 debug :0100000c 可以 attach 进 simpledb ，然后你会看到 :0100000c> 这样的提示符。

你可以输入 ... 来检查当前消息是什么。通常你会看到这样的信息（表示当前是一个 timeout 消息）。因为这个时候 simpledb 在不停的调用 timer 保持和你的交互。

```
1       userdata: (nil) 0       226     0
```

当然，你也可以运行你想运行的任何 lua 代码。

调试器在这里只提供了一个叫 watch 的函数，让我们下一个条件断点，并跟踪运行它。

```
:0100000c>watch("lua", function(_,_,cmd) return cmd=="get" end)
```

这时，启动一下测试客户端，并输入 get hello 。

```
./3rd/lua/lua examples/client.lua
```

我们会看到，在输入 get hello 后，调试控制台的提示符会变成 ./examples/simpledb.lua(18) 表示停在了 simpledb.lua 的 18 行。接下来可以用 ... 检查这个函数的参数。用 n 继续一行行运行，直到消息处理完毕。

```
:0100000c>./examples/simpledb.lua(18)>...
get hello
./examples/simpledb.lua(18)>n
```

```
./examples/simpledb.lua(19)>n
./examples/simpledb.lua(23)>n
:0100000c>
```

如果用 s 的话还会跟踪进入子函数内部。为了方便调试，调试器不会进入定义在 skynet.lua 的函数里（通常你不需要关心 skynet 本身的实现）。

另外，调试器还提供了一个叫 _CO 的变量，保存在正在调试的协程对象。如果你想使用 debug api ，这个变量可能有用。例如，可以用 debug.traceback(_CO) 查看调用栈：

```
:0100000c>watch "lua"

:0100000c>./examples/simpledb.lua(18)>_CO
thread: 0x7fe7f9811dc8
./examples/simpledb.lua(18)>debug.traceback(_CO)
stack traceback:
        ./examples/simpledb.lua:18: in upvalue 'dispatch'
        ./lualib/skynet/remotedebug.lua:150: in upvalue 'f'
        ./lualib/skynet.lua:111: in function <./lualib/skynet.lua:105>
./examples/simpledb.lua(18)>s
./examples/simpledb.lua(19)>s
./examples/simpledb.lua(10)>s
./examples/simpledb.lua(11)>debug.traceback(_CO)
stack traceback:
        ./examples/simpledb.lua:11: in local 'f'
        ./examples/simpledb.lua:20: in upvalue 'dispatch'
        ./lualib/skynet/remotedebug.lua:150: in upvalue 'f'
        ./lualib/skynet.lua:111: in function <./lualib/skynet.lua:105>
./examples/simpledb.lua(11)>c
```

最后谈谈 lua 的一个问题。

根据 lua 5.3 的文档，在 debug hook 里是可以调用 yield 让出线程的。只要满足两个条件：1. 不传入任何值，2. 只在 line 和 count 模式下调用。

这给实现单行运行指定 coroutine 提供了方便。你可以给指定的 coroutine 挂上 debug hook ，每运行一行就 yield 出来。这样不会影响其它 coroutine 的处理，还可以在主线程中去观察它。

但是，目前的 lua 5.3 （包括 lua 5.2），如果在 hook 中 yield 后，调用 getlocal 去观察挂起线程的局部变量时，进程会 crash 掉。

我花了一晚上寻找原因。

似乎是因为，挂起的线程，lua 在 callinfo 结构中调整了 func 的值，用于保护调用栈上的临时变量。（这样用 api 去访问挂起线程时，是看不到那些临时变量的）。可以理解为 callinfo 的 func 就是当前栈帧的底。但通常，对于 lua 函数，这个底同时也指向函数对象。而 debug api 则需要从这个对象中获得调试信息。

所以一旦访问挂起线程顶部 lua 函数的调试信息，很可能访问到一个非函数对象，结果就挂掉了。

我们平时用 coroutine.yield 来让出 coroutine 则不会有问题。这是因为，coroutine.yield 是一个 C 函数，也就是栈顶并非一个 lua 函数，也没有局部变量这样的调试信息可以获取。

只有从 debug hook 中 yield 的线程才有这个问题。它从直接从一个 lua 函数中让出，而栈顶的信息对于调试 api 来说是错误的。

我尝试打了个 patch (见 skynet 上的提交）绕过这个问题。一旦发现从一个 yielded 的 coroutine 的顶部取lua 函数的调试信息，则从 extra 域而不是 func 域读去函数对象。

bug 已经提交到 lua mailling list 中，不知道 lua 开发团队是否有更好的解决方案。

云风 提交于 February 11, 2015 03:05 PM | 固定链接

COMMENTS

请教个问题 debug 和 watch 的执行流程是怎样的？list能找到执行流 debug 和 watch 不知怎么执行的

Posted by: lxl | (7) April 13, 2015 11:58 PM

虽然平时lua不用 debug 但还是 MARK 一下

Posted by: 红色男爵 | (6) March 18, 2015 10:31 AM

sadf asdf 撒的个的说法个的说法个的法个人打手犯规 http://www.aseaa.com/377.html啊色法撒旦法撒旦法阿斯蒂芬分撒的分委托人阿斯蒂芬三等份

Posted by: 小莉爱美 | (5) March 12, 2015 12:21 PM

sadf asdf 撒的个的说法个的说法个的法个人打手犯规 http://www.aseaa.com/377.html啊色法撒旦法撒旦法阿斯蒂芬分撒的分委托人阿斯蒂芬三等份

Posted by: 小莉爱美 | (4) March 12, 2015 12:21 PM

vs2012本身就能调试lua

Posted by: egirlasm | (3) March 4, 2015 03:17 PM

我做过一个基于websocket的调试工具，用于一个基于coroutine的web服务器，如果websocket连接了，那么每个新传入的连接对应的coroutine都会被set一个line hook，每执行一行都会给调试器发送一下状态；也可以给代码在运行时设置断点（输入行号），每个断点把upvalue和local的状态传到调试器，但是因为不能yield across C-call boundary，只能在hoaok的callback里面调用debug.debug弄出个repl。感觉lua的multi-thread和debug能力对于线上调试帮助蛮大的。

Posted by: zyxwvu | (2) February 18, 2015 12:29 AM

看不大明白 不过还是要留个爪

Posted by: hannah | (1) February 12, 2015 09:39 AM

POST A COMMENT

非这个主题相关的留言请到：留言本

名字：

Email 地址：

为了验证您是人类，请将六加一的结果（阿拉伯数字七）填写在下面：

URL:

记住我的信息？

留言：
（不欢迎在留言中粘贴程序代码）

提交