云风的 BLOG
思绪来得快去得也快，偶尔会在这里停留

« Skynet 新的 socket.channel 模式 | 返回首页 | 内存安全的 Lua api 调用 »

在不同的 lua vm 间共享 Proto

在 skynet 这种应用中，同一个系统进程里很轻易的就会创建数千个 lua 虚拟机。lua 虚拟机本身的开销很小，在不加载任何库（包括基础库）时，仅几百字节。但是，实际应用时，还需要加载各种库。

在 lua 虚拟机中加载 C 语言编写的库，同一进程中只会存在一份 C 函数原型。但 lua 编写的库则需要在每个虚拟机中创建一份拷贝。当有几千个虚拟机运行着同一份脚本时，这个浪费是巨大的。

我们知道，lua 里的 function 是 first-class 类型的。lua 把函数称为 closure ，它其实是函数原型 proto 和绑定在上面的 upvalue 的复合体。对于 Lua 实现的函数，即使没有绑定 upvalue ，我们在语言层面看到的 function 依然是一个 closure ，只不过其 upvalue 数量为 0 罢了。

btw, 用 C 编写的 function 不同：不绑定 upvalue 的 C function 被称为 light C function ，可视为只有原型的函数。

如果函数的实现是一致的，那么函数原型就也是一致的。无论你的进程中开启了多少个 lua 虚拟机，它们只要跑着一样的代码，那么用到的函数原型也应该是一样的。只不过用 C 编写的函数原型可以在进程的代码段只存在一份，而 Lua 编写的函数原型由于种种原因必须逐个复制到独立的虚拟机数据空间中。

这些限制有哪些呢？

函数原型包含了三类数据：字节码、常量表、调试信息（包括字节码对应的行号、函数名、局部变量名等等）。这些数据都是只读的，理论上是可以被共享的。

但是函数原型(proto) 也是 lua 的基础类型（但没有暴露到语言层），依然是被 Lua 虚拟机管理的 gcobject ，它需要参于垃圾收集的过程。Lua 在实现时并没有考虑将多个虚拟机共享数据。

如果我们需要共享，第一步就是要改变 proto 类型的生命期管理。不能再由单个 lua 虚拟机的 gc 扫描流程决定是否要释放一个不再被引用的 proto 。

一个完备的方案是对 proto 做一个线程安全的引用计数，但我们也可以简单粗暴的直接在内存中保留所有的 proto 对象，无论是否有人引用它。

保留所有用过的函数在内存中这种做法是广泛存在的，如果你对比看 C 层次的函数，即使 C 函数存在于动态库中，我们也不能轻易卸载动态库，这有让其它模块保留过动态库中函数指针变得无效。另外，由于调试信息的存在，引用计数的方案会对 lua 实现做相当大的改变。

第二步，我们需要考虑常量表。对于常量字符串，往往是不可以被多个 lua 虚拟机共享的。尤其是短字符串，lua 会对短字符串做唯一化 (string interning) 处理，同样的短字符串在同一个 lua 虚拟机中只有一份。不同的 lua 虚拟机中的短字符串一定会被判定为不同的。如果对常量表中的字符串也做共享处理，那么除了需要给 lua 实现增加一种字符串类型（不被 gc 管理的字符串）外，还会降低字符串处理速度（目前 lua 在做短字符串比较时，直接比较对象指针，可以达到 O(1) 的处理速度；而如果常量字符串在不同的虚拟机中的话，比较会变成 O(n) 的复杂度）。

第三步，每个 proto 对象中带有一个 closure cache 。绑定同样 upvalue 的 proto 生成的 closure 可以被复用。但如果 proto 是跨虚拟机的，这个 cache 就很难正常工作了。

第四步，调试信息中也有大量的字符串。考察一下 Lua 实现可以发现，Lua 的 api 仅将这些字符串用内部字符串对象储存参与 gc 管理，但并不会把这些字符串对象传递到别的地方。所有 api 都是返回这些字符串对象的 C string 指针的。

针对这些问题，我们可以开始对 lua 的实现做改造了。

我们可以将 proto 数据结构拆分成可共享和不可共享两部分。不可共享的有常量表和 cache ，其它都可以共享。不可共享部分继承原有的 proto 结构，再用一个指针指向共享部分即可。我们需要在共享的数据结构中保留一个它实际存在于的 lua 虚拟机的指针。只有这个虚拟机才有权利回收它所占的内存。而其它引用它的 lua 虚拟机在 gc 时，可以检查这个指针来决定是否要标记清除它。

lua 提供了一个 api `lua_topointer` 可以返回一个 lua 函数对象中的原型指针（注：这是 undocument 的）。我们只需要再添加一个 api 把函数原型还原成 closure 即可。

这里引入了一个新 api 叫 lua_clonefunction 它能复制一份函数原型的常量表到当前的 lua 虚拟机中，并创建其它需要的数据结构。

我给 lua 5.2.3 打好了 patch 支持这个特性 。并将它合并到 skynet 的主干上了。

为了更好的利用这个特性，我在 skynet 中，改写了 luaL_loadfilex 。这个 patch 版的文件加载函数是线程安全的。它为每个文件名对应的函数（lua 中加载一个源文件，就生成一个函数）创建一份独立的 lua 虚拟机，并将生成好的函数原型指针记录下来。之后同名文件的加载就不再有文件 IO ，不必再次解析文件，直接用 lua_clonefunction 复制一份出来。

为了 skynet 服务器可以热更新 lua 脚本，还增加了 clear cache 的方法（skynet.cache.clear），可以将 cache 重置。当然，之前加载过的代码其实是没有从内存中清理掉的，这一定程度上会带来一些内存泄露。但考虑到这个 patch 可以给系统节约的内存，不是过于频繁的热更新是可以接受的。

这个 patch 可以带来的好处：

1. 对于我们的项目（陌陌争霸 ），每个在线用户大约可以少消耗 1M 内存。
2. 为一个在线用户初始化 lua 虚拟机的时间加快了 4 倍。
3. 由于字节码占用的内存更为集中，提高了 cpu 内的 cache 利用率。

但是，这个 patch 也增加了热更新的复杂度。需要主动清理 cache ，并考虑历史上的过期版本的代码占据内存不能回收的问题。如果不想在 skynet 中使用这个 patch ，可以在 makefile 中调整 lua 库的链接，指向官版的 lua 即可。

云风 提交于 March 27, 2014 10:44 AM | 固定链接

COMMENTS

为什么每个用户的lua虚拟机会有1m之大？。。。

Posted by: aohajin | (8) December 22, 2014 10:27 PM

为什么每个用户的lua虚拟机会有1m之大？。。。

Posted by: aohajin | (7) December 22, 2014 10:27 PM

风哥，我很明白你的想法，其实我也一直困惑这个问题，每个玩家就一份一模一样的lua上下文内容，无疑是增加着一些内存的开销，但能把其共享和非共享剥离出来真不简单，我打了你的patch，目前还没明白怎么弄出来的！但我知道你的初衷

Posted by: lelong | (6) September 25, 2014 11:38 PM

今早陌陌争霸挂了，挂了很久了。。。

Posted by: L | (5) April 4, 2014 01:23 PM

好久没来大哥博客了

Posted by: 宁波 | (4) March 30, 2014 12:27 AM

skynet是用的erlang的思想, actor并发模型. 我估计也应该支持独立玩家重载脚本吧

Posted by: 战魂小筑 | (3) March 28, 2014 11:16 AM

每个玩家一个lua_state不浪费吗？lua不只做逻辑还保持状态？ 默默争霸里面 打仗的流程服务器会做验证吗？

Posted by: 竿子 | (2) March 27, 2014 01:34 PM

为什么不做成异步单线程呢

Posted by: T | (1) March 27, 2014 12:04 PM

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