**在线更新：**

**小规模改动采用在线更新，不需重启，譬如更新service.world.playerobj.lua文件（可一次性更新多个文件），两种途径：**

1、GM指令：$update\_code “service.world.playerobj”

2、连上dictator后门，执行update\_code service.world.playerobj

**需要在线更新的模块编写规则：**

项目目前使用：

存在require和import两种包管理，require由lua vm进行管理，一

般用于不期望更新的模块，一般为基础模块；import 由 上层管理，主要

用于期望更新的模块，适用于大量业务模块。

在线更新原理和取舍：

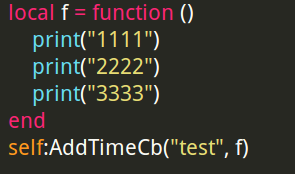
对应代码reload.lua。reload的流程：譬如更新A模块(A1->A2)，会递归遍历A1，对非table类型取A2对应值覆盖，table类型则取A2对应table递归更新A1对应table。按该模式进行更新，保证所有table的内存地址不变，也保证了A2的\_ENV和A1的\_ENV为同一对象。为保证更新速度，我们不采取遍历lua vm去进行新老对象的替换，新函数所携带upvalues也不会替换为老函数的upvalues，所以在写法上希望有所注意：

1、

避免外部对模块方法或者对象的长期持有，譬如：

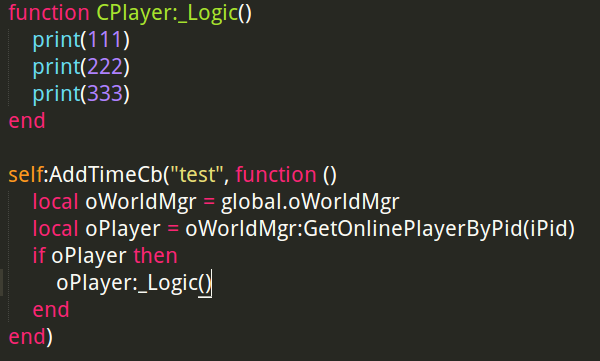


问题：reload不会更新self.m\_Func持有的老对象。写各种回调引用的闭包尤其注意这个问题，譬如：



这种情况下，如果reload，之前已经AddTimeCb的闭包f还是老对象， 逻辑并没更新，只有reload之后AddTimeCb的闭包f才是新对象新逻

辑，这不是我们希望的结果，我们希望两种情况都是跑新逻辑，推荐写法：



也即将逻辑也即期望更新的部分移到可被更新的对象方法或者模块方

法，闭包函数只调用这些可被在线更新的函数。

2、

模块内顶层域慎用local，除了模块或各种静态数据或不期望更新的本

地函数，所有其他都不需要local，正常范例如下：



不正常如下：

譬如A文件：

老：

local m = {}

function f()

m[2] = 2

end

新：

local m = {}

function f()

m[3] = 3

end

问题：在线更新之后，f操作的m不是同个对象，因为upvalue不会更新，这

存在很大隐患，特别m是一个动态数据结构。

3、在线更新三个级别，另外还有客户端内容更新流程两个级别

a. update\_res：适用于在线更新导表数据，用法：修改生成data.lua后后门或者客户端执行update\_res；

b. update\_code：适用于替换更新替换类方法、类变量、模块级别变量、函数，用法：修改代码文件后，通过后门或者客户端指令执行update\_code，传入文件路径列表，同时带有是否同时更新协议proto的标记，注意在线更新协议一般用于紧急情况，只做增加字段，不删除原来字段，不改动原来字段名，不修改或者打乱原来字段的序号；

c. update\_fix：适用于线上修复对象的属性，用法：修改dictator/fixbug.lua文件，添加修复函数，然后后门或者客户端执行update\_fix，传入修复函数名。

d.客户端代码在线更新：

1.客户端给在线更新代码给服务端，服务端程序将代码放入public.version文件的CLIENT\_UPDATE\_CODE变量中

2.登录后门，执行client\_code

3.每周注意清理CLIENT\_UPDATE\_CODE变量

e.客户端更新导表文件

1.客户端通知服务器需要更新导表文件的名字，服务端通过脚本执行生成对应lua文件的打包文件，每一个文件都有一个版本号，版本号为脚本执行时的时间戳，修改version的CLIENT\_UPDATE\_RES值，连接后台执行client\_res，则可以更新对应客户端导表文件

2.客户端登录gs服务器之前，会发C2GSQueryLogin请求导表文件数据，服务端一定要回复相关协议，客户端才会进行下一步。服务器根据当前的导表数据和客户端发送过来请求数据进行对应，告知客户端导表的增加，修改，删除内容

3.服务端进行客户端导表在线更新时，会通知全服在线玩家，发送当前的导表文件和版本号给客户端，客户端根据自己的导表和服务端发送的导表文件信息进行比对，根据比对内容进行导表数据的请求

4.每周维护需要定时清理version的CLIENT\_UPDATE\_RES和gamedata/server/client\_daobiao内容