**开发注意事项说明：**

1. **逻辑层开发尽量使用Send和CallBack回调，少使用skynet.call().具体实现可以参考interactive**
2. **开启服务时如果需要使用res导表，需要在文件头require res,然后在需要使用的地方require。**

**3. 变量命名规范:整形变量i开始，字符串变量s开始，对象o开始等等，协议统一用小写：避免这种写法。存盘变量也统一小写**

**4. 游戏公用代码库在lualib**

**5. 时间库中尽量使用get\_time()函数（以秒为单位，精确到10毫秒），如果有需求可以考虑其他更精确的接口**

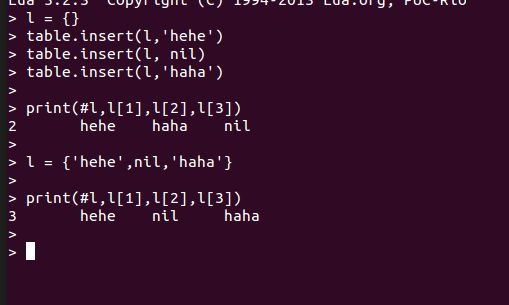
**6. RO,RW模块玩家可以直接调用，不需要回调，玩家在线不会卸载。模块中的数据可以离线操作**

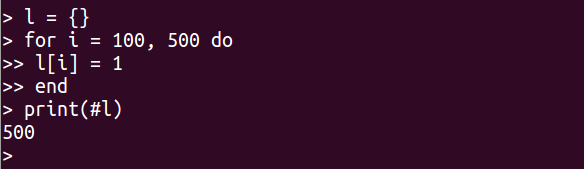
**7. 尽量减少模块的互相引用，使用mgr管理**

**8. 定时器使用：先删除，再添加**

**9. table数据存取接口注意：bson存储默认是：如果是数组key是存int类型，如果是字典key是以字符串格式存储。加载数据要注意类型转换。建议统一使用db\_key将key值进行转换后再存盘**

**10.**





**注意以上lua的坑！**

**11.**

**在线更新模块需要注意写法，在线更新无法更新闭包**

**12.**

**GM指令编写，如果希望策划或者测试人员使用的请随手加上Helper编写，如果只是服务端自己使用那就不要写**

**13.**

**涉及使用loadfile的，使用自定制的loadfile\_ex，原因：skynet底层为了省内存这块做了共享内存，不定制的话每次loadfile都会是第一次的内容**

**14.**

**一般业务服务启动都需要调用dictator服务(业务服务管理/后门)的Register接口，便于后期进行后门注入或者进行在线更新等基础功能**

**15.**

**开发期内，服务端小规模改动，可采取在线更新，而不需要重启服务器，两个途径：**

**a、GM指令。$update\_code service.world.playerobj**

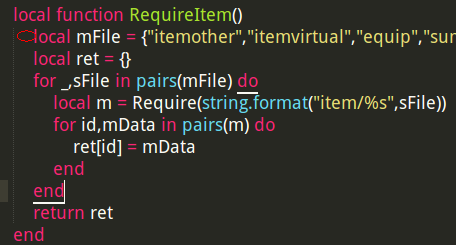
**b、后门。update\_code service.world.playerobj**

**16.**

**字段命名问题：配置表字段，proto字段及存盘字段的命名统一格式 xxx\_yyy，开发期遇到驼峰格式xxxYyy的要上技术群和客户端确认，然后修改过来，注意要同步修改服务端相关的逻辑。**

**17.**

**代码格式问题。对于代码空格对齐问题请用4个空格**



**18****.**

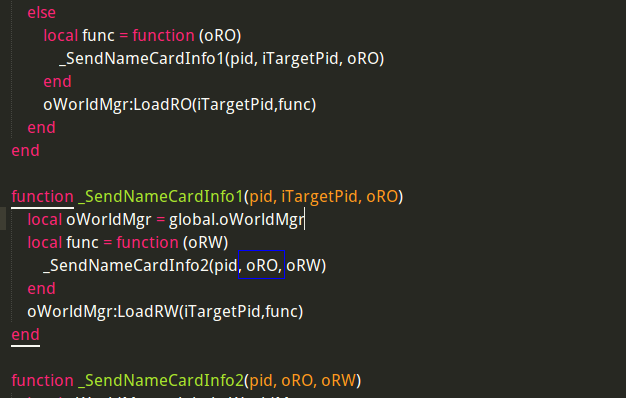
**所有继承CBaseObject的对象要注意Release，有部分对象有自己的Release需求可重写但记得调用基类Release（譬如对象某些字段引用了需要Release的对象）**

**1****9.**

**所有涉及存盘的对象，需要考虑好定时存盘和关服存盘**

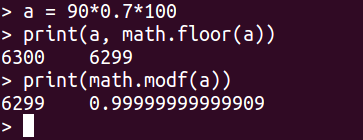
**20.**

**所有的回调避免直接引用内存的上层对象，尽量使用id索引，因为内存对象在回调时可能已经消失或者被修改了。图中RO在LoadRW回调前就消失了**



**21.**

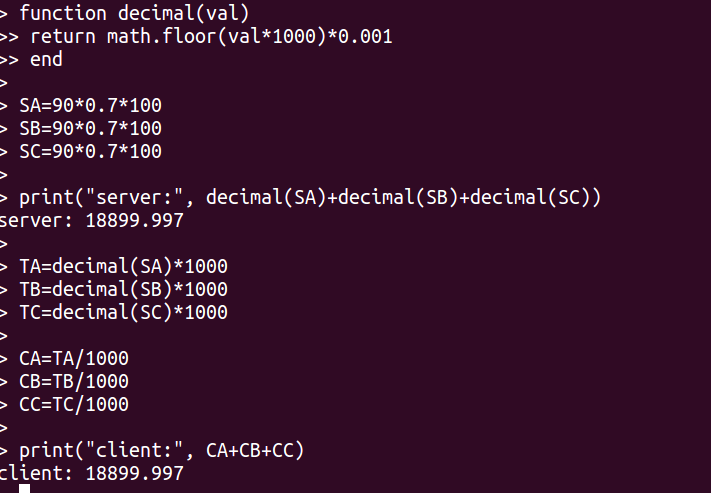
**浮点数问题．**



**业务可能遇到问题：**

**1)　结果显示问题．如上譬如ａ的结果（无限逼近6300），假设是向下取整后发协议则会取6299显示．这里取决于策划，最终显示可以统一取四舍五入，不影响任何计算流程**

**２)　计算问题．部分业务存在CS各自算的情况，这种情况要保证两边计算流程一致，类似以下：**



**22.**

**netcmd和logiccmd等命令接收处理文件里面不要添加除收包函数以外的其他函数**

**23.**

**写功能模块尽量使用单例类来划分，不是为了好看，是有实在的好处：**

**１）系统内所有对象或者功能的获取都有稳定的root，那就是global，不需要import require之类的来掺杂**

**２）写代码，数据自然而然的就放在类这个安全环境内，减少写出不适合在线更新的代码**

**24.**

**卡循环问题：**

**1）目前我们lua源码是改版支持了打断机制，详情可看dictator服务\_CheckEndless，对于执行过久的逻辑（超过13秒）会进行error。出现类似上图recv a signal 0报错，查清楚接下来的栈报错，定位问题**

**2）对于一切有遍历的行为要谨慎再谨慎，特别是预期会被多次调用的接口内，或者类似道具邮件等可能元素较多的集合，尽量减少无效遍历**

**25.**

**服务器任何元素存储和传输，一定要有硬上限，策划设计有问题要提出，自己得估计时空度**

**26.**

**事件机制**

**事件机制用起来挺舒服，不过在实质上的运行时，往往会被滥用，经常会成为效率热点，主要是跟这种写法本身诱导性有关。**

**譬如A事件发生，引起了B C D事件，本来最简单写法就是：if A then B C D end，引入事件机制之后成了：Trigger(A)**

**好处：**

**1、解耦**

**坏处：**

**1、用搓了，导致侦听者数量运行时不可控，这个代码肉眼难发现**

**2、侦听门槛 不等于 触发条件，经常一个事件发生了导致一堆的无效遍历**

**对这东西，建议是必要时再用，且使用前估量一下事件发布者的最多订阅者数量是多少，譬如玩家或者战士对象应该作为发布者还是挺合适的，订阅者不过就是最多几十个功能系统。不鼓励广泛使用，客户端代码是很喜欢事件机制的，但他们运行时对象是很少的，但毕竟我们是服务器，宁愿丑得安全都不愿意好看不中用**

**27.**

**重要数据变动处务必加上行为日志，做到能通过日志比较完整的重现玩家行为。**

**规范说明：修改和添加玩家行为日志，需要修改导表中log文件夹下对应系统类型的字段说明，保证记入数据库的字段和导表说明一致。**

**28.**

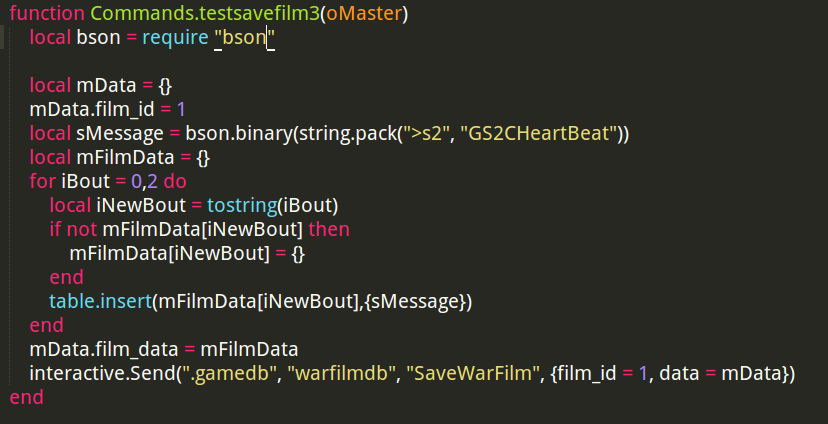
**mongodb的upsert特性．建议尽量不用upsert，而是采取显式的create（在应用层保证索引key逻辑唯一，并进行创建insert）．另外，如果确定使用upsert的集合，那么就不要同时有使用insert的可能性．原因：对于\_id的生成，我们mongo driver的设计：insert的objectid在mongo client driver这层本地生成，而upsert是由mongo server自行判断存在与否来生成，存在冲突可能性．**

**29.**

**mongodb采取bson编码格式，除了整型/浮点型/字符串等常见类型可交由C层自动判断，而对于一些譬如 二进制类型/正则表达式/定制时间类型，需在上层手动调用bson相应编码接口，保证bson格式，譬如：**

**30.**

**在定时器函数回调内，尤其是使用servicetimer服务内定时器，不要使用协程挂起操作（一般业务逻辑性的service拒绝使用协程序，使用interactive回调机制即可）**

****

**31.table中不要出现对类的应用，在线更新有问题，改为引用字符串，如下图，左边的是错误的写法，右边是正确的**

