内存共享方案简述



图1

lua-shaedata.c ： C层管理共享数据

corelib.lua : 封装C层接口，提供接口

sharedatad : 主机用于创建更新，唯一操作共享数据角色

sharedata: 用于查询数据的接口

游戏中配置（导表）一般是excel转成lua table格式，而通常数体积都会比较大，如果每个Actor都独立使用一份导表，如果50M内存消耗，10个Actor则需要500M内存，显然这种消耗并不是我们想看见的。

Sharedata是基于数据共享的想法设计，其将需要共享的内存数据用自己定义的结构struct table存储，并只提供指向struct table的指针，lua透过指针便可访问数据。为了方便访问，透过构造一个元表，用户可以像访问普通table一样访问共享数据。

lua-sharedata.c



图2

struct table

创建一个共享结构，应用需要提供一个lua的table结构，然后将table中的数据（key-value）转换成自定义的struct table结构。为了方便处理，将table中的数据和哈希分开，定义数字key<#table长度的为数组，其余均为hash结构。struct table会共享同一个lua\_State。

union value

定义table中的value，如果value是table则是指向struct table的指针。对于字符串，为了节省内存，这里并没有使用char\*,而是指向字符串在栈中的位置。上图中，lua\_state栈中，栈顶是一个table，栈底是userdate（实质就是指向struct table的指针），中间则是字符串，而union value的number\_key string字段就是指定了目标字符串位于栈中的位置，如果number\_key= 3 ，则其值为str2。这个栈中字符串构造是先通过把所有字符串放入table中，构造一个已字符串为key，number\_key为value的table = {[str1] = key1,[str2]=key2 …[strn} = keyn]。最后再放入栈中即可（参考convert\_stringmap函数处理）。

数组结构

数组处理非常简单，初始化时，数组长度L1,L2 = #table，会为array和array\_type分配L1个单位空间。取值只需要根据下标取array[index]，然后根据array\_type[index]的数据类型进行相对的类型转换即可获得数据。

哈希结构

初始化先计算需要哈希个数，然后分配空间，计算hashkey来确定位置，若没数据则设置数据，若有冲突，则稍后处理（见fillcolliding函数）。查找时，则根据hashkey来查找，若key值不相等且有冲突，则向next下一个继续查询下一个元素。

主机创建与更新



图3

创建

上图中，唯一服务sharedatad先要创建共享对象。

当用户通过query方法从sharedata获取共享数据，这个时候如果本地没有该数据，则需要向sharedatad获取，sharedatad返回共享对象（userdata），最后sharedata通过传递共享对象调用box获得访问数据的table返回给用户。

Sharedatad中的共享对象是根据引用来判断是否释放对象的，学过Python的应该很清楚。在sharedatad返回对象时，会引用+1，所以在后面需要手动发送confirm来使引用数-1,。

最后会fork 一个协程monitor来关注更新

更新：

 a. 当fork协程monitor时，会调用sharedatad的monitor，并且挂起等待返回。而sharedatad把请求者加入watch列表等待更新。

 b. 当某个服务调用sharedatad的update来更新后，底层不会修改原来的共享对象，而是重新创建一个新的替换，并唤醒监听的协程，通知更新。

C. 唤醒sharedata的minoter后，回调用corlib的update方法，并且删除缓存

d. 在更新时，不会直接更新，而是等待下次查询到来才检查，这种为惰性策略。

释放：

当sharedatad需要删除共享对象时，由于其他服务还可能在使用该共享对象，所以只能减引用。并且定时检查引用数<0的删除。

 查询

使用sharedata.query时，会返回一个table，访问这个table即可访问到该共享对象的数据。该查询技术使用了元表，table只保存了共享对象，当我们访问table时，触发了\_\_index方法，然后通过访问获得该值，如果该值是userdata，则证明是struct table 的指针，再构造一个元表，否则就直接返回。



备注:

sharedata单元测试

只需要lua-sharedata.c,atomic.h,corelib.lua放在一个目录

编译 gcc -o skynet\_sharedata\_core.so lua-sharedata.c --share -fpic

Corelib.lua修改载入库名local core = require "skynet\_sharedata\_core"

测试代码



 移植分析：

项目会启动一个root的Actor，然后root再启动其他功能Actor，将root作为shareadatad，并且在启动其他Actor后将共享对象call给Actor，这里需要注意的时，这个启动必须有严格的启动流程，必须是创建后第一事是设置共享对象，然后再执行其他流程，比如某些系统初始化。

更新时，由于没有skynet的监控和通知，所以这里直接在root广播给所有Actor即可。

总体来说基于Actor模型移植可行性比较高。