* 简介

单例模式在编写和理解上都有很大的优势，但是对应的也存在一些劣势如下：

1. Protobuf的C模块部分因为不支持多luastate共享，在单例模式下，每一个agent都需要重复的启动一个相同的C模块
2. Lua不同于erlang这种共享模块的虚拟机，每一个luastate都会加载和初始化自己的不同状态的模块，也就意味着将来如果需要热更，则需要遍历所有的已启动，或者受代码更改部分影响的luastate，让这些luastate都独自完成热更才可以
3. 对于skynet来说，一个luastate或者理解成一个服务是支持不同任务类型并行的，并不是erlang那种对于一个进程中的消息队列是绝对串行执行完毕的，也就是启动多个agent对于skynet的这种架构，显得有点多余。用queue也可以达到每个角色的处理逻辑对内串行，从而减轻考虑重入状态的负担

* 实现细节

共享agent与单例模式比起来，最大的不同是一个agent是为多个角色服务，也就是再也不能直接调用role\_data.get\_xx\_info()这种无参调用，所有的接口必须带上关键key值

从登陆说起，登陆的时候，关键值只有fd，因为当时账号还没有完成登陆，能用来做识别的只有当前的fd，之前我们已经将所有的账号登陆前的处理整合到了role\_account模块，所以可以做成对于role\_account模块处理的消息，都带上关键key值fd，用fd来识别和管理当前链接，对应的数据有fd对应的 state，检查心跳和挂机超时使用的数据结构，用来校验ccd的ccd\_spec。这些数据除了state统一由agent\_lib管理，对外也暴露了必要的操作接口。

也就是说，在一个角色完全登陆成功之前，所有的都是以fd作为索引来在同一个agent完成不同客户端的同时处理，这里在agent处理消息的时候，针对每个fd实现了一个queue，每个fd的消息都是在自己的处理队列里串行处理，对于同一个fd的消息，可以做到保序处理，不用担心重入的问题。

另外对消息处理框架CS\_HANDLE进行了再次包装，加入了key参数，key参数在CS\_HANDLE初始化的时候被传入，后续的调用中，都会作为第一个参数，不需要额外传参，同样对于role\_account模块来说这个key值只能是fd，即使角色登陆已经完成，后续也仍使用fd来做，而对于其他role模块，一般传入的是roleid。

关于数据保存，所有角色的数据都保存在由role\_data模块管理的\_dict中，\_dict中的每个元素都是以角色的roleid为key值，对于一个data\_set，而data\_set和之前单例模式中的很像，区别在与data\_set在发现某个字段未初始化而调用对应模块的初始化函数的时候，必须加入这个角色的roleid，所以data\_set有一个保留字段\_roleid用来存储这个值，当data\_set初始化的时候这个值被初始化，供后续的data\_set中逻辑数据部分需要初始化的时候调用，对应这种修改，数据落盘的时候也是以\_dict为单位，每次处理整个\_dict中未落盘的数据，而不是按照角色分批循环处理(虽然本质上其实也是按照以一个玩家的数据为单位以此落盘的)。

* 注意点

1. 编写lib库的时候尽量编写无副作用的函数，即函数所有需要的数据都会作为参数传进来，或者函数在任何服务下都可以执行，不要再引入role\_data模块
2. 虽然使用queue来做了单个fd内的串行，但是不代表不会出现重入，对于同一个agent可能会在处理过程中阻塞，然后执行其他任务后切换回来，所以千万不要依赖本地环境，所有的计算过程使用函数体内的局部变量和入参来完成，内存中的临时变量是不可靠的
3. 如果将来新增了time\_wheel调用用来刷新所有agent内玩家的某些数据的时候，对于单个玩家的操作一定要用CATCH来保护，避免某个玩家操作执行失败从而导致所有其他玩家无法执行，当然必须确保执行不会抛出异常
4. 模块尽量干练，如果差异大就另起模块重新编写，不要一个模块完成很多种功能，容易造成循环依赖，一般来说lib类的库只会依赖一些基础的模块，应该避免这种lib库对功能逻辑模块的依赖。