skynet lua 与 C 交互

从 skynet 核心模块来看，它只认得 C 服务，每个服务被编译为动态库，在需要时由 skynet 加载。skynet 提供发送消息和注册回调函数的接口，并保证消息的正确到达，并调用目标服务回调函数。其它东西，如消息调度，线程池等，对于用户来说都是透明的。

skynet 服务可以由 lua 编写，因此 skynet 将 C 模块核心接口通过 skynet/lualib-src/lua-skynet.c 导出为 skynet.so 提供给 lua 使用。在 lua 层，通过 skynet/lualib/skynet.lua 加载 C 模块(`require "skynet.core"`)完成对 C API 的封装。主要涉及 lua 服务的加载和退出，消息的发送，回调函数的注册等。用户定义的 lua 服务通过`require "skynet"`的接口即可完成服务的注册，启动和退出等。关于 skynet lua api 可以参见 skynet wiki。

lua 服务如何关联到 C 核心层

下面主要提一下 skynet 是如何在这套 C 框架上承载 lua 服务的。

skynet 预置了一个 C 服务，叫 snlua(位于 skynet/service-src/skynet_snlua.c)，这个服务的主要任务就是承载 lua 服务。一个 snlua 服务可以承载一个 lua 服务，可以启动任意份 snlua 服务。我们直接从 snlua 这个 C 服务开始，介绍一个 lua 服务是如何融合到 C 框架中的。当需要加载一个名为”console.lua”的服务时，我们将启动一个参数为”console”的 snlua 服务。主要流程：

1. 调用 skynet.launch(“sunlua”, “console”)
2. skynet.launch 对应 C 中的 cmd_launch，它通过 skynet_context_new 加载 snlua 服务：

   a. 创建服务对应的 skynet_context

   b. 加载 snlua.so 模块，并调用模块导出的 snlua_create 创建 snlua 服务，snlua_create 会创建一个 lua_State，这样每个 lua 服务拥有自己的 lua_State。

   c. 创建服务消息队列，并为 skynet_context 绑定唯一 handle，将消息队列放入全局消息队列中

   d. 调用 snlua_init 初始化服务，在 snlua_init 中，完成对 snlua 回调函数的注册。并且构造一条消息，将要加载的 lua 服务名(“console”)发给自己。

1. 在 snlua 服务的消息回调函数中，先注销回调函数。然后通过加载并运行一个叫 loader.lua 的脚本，并解析收到的数据(“console”)来完成实际服务的加载。

2. loader.lua 在指定路径(可配置)下找到 console.lua 脚本，并执行 console.lua 脚本
3. 此时回调函数就返回了。由于之前已经注销了 snlua 的回调函数。此时 snlua 看似”报废”了。而事实在重点在 console.lua 当中：

每个 skynet lua 服务都需要有一个启动函数,通过调用 `skynet.start(function ... end )`来启动 lua 服务。在 skynet.start 中：

```
c = require "skynet.core"

function skynet.start(start_func)

        c.callback(dispatch_message)

        skynet.timeout(0, function()

        init_service(start_func)

end)

end
```

通过 c.callback 注册了 lua 服务的回调函数 dispatch_message，c.callback 由 skynet.so 导出，它最**终调用 skynet_callback 这个函数完成对本服务(当前是 snlua)的回调函数注册**。dispatch_message 也定义于 skynet.lua 中，它**主要的功能是根据消息类型(C 层定义于 skynet.h 中，lua 层定义于 skynet.lua)将消息分发到 lua 服务指定的回调函数**，前面提到过 **skynet.dispatch 可以注册特定类型的处理函数。c.callback 将 dispatch_message 注册为 snlua 新的回调函数。此时 snlua 这个服务就承载了 lua 服务，因为它收到的消息将通过 dispath_message 转发到 lua 服务注册的回调函数中。**

那么 c.callback 如何将一个 lua 函数(dispatch_message)注册为一个 C 服务(snlua)的回调函数的呢？在 skynet/lualib-src/lua-skynet.c 中，c.callback 对应的 C 函数实现如下：

```
static int_callback(lua_State *L) {

        struct skynet_context * context = lua_touserdata(L, lua_upvalueindex
(1));
```

```
        int forward = lua_toboolean(L, 2);

        luaL_checktype(L,1,LUA_TFUNCTION);

        lua_settop(L,1);

        lua_rawsetp(L, LUA_REGISTRYINDEX, _cb);

        lua_rawgeti(L, LUA_REGISTRYINDEX, LUA_RIDX_MAINTHREAD);

        lua_State *gL = lua_tothread(L,-1);

        if (forward) {

                skynet_callback(context, gL, forward_cb);

        } else {

        skynet_callback(context, gL, _cb);

}       return 0;}
```

_callback 将 lua 服务消息回调 dispatch_message 以_cb 函数地址为 key 保存到 lua 注册表中。再将_cb 函数作为 lua 服务的”代理回调函数”注册到 C 核心框架中。这样真正的回调函数_cb 就能够满足 C 服务回调函数形式。这里 C 中的_cb 和 lua 中的\dispatch_message 都是预先定义好的，可以通过 lua 全局注册表做一一映射。

当消息到达 snlua 时，在_cb 中，通过 `lua_rawgetp(L, LUA_REGISTRYINDEX, _cb);` 从 lua 注册表中取出 lua 服务的真正回调函数 dispatch_message，压入消息参数。然后调用 dispatch_message。dispatch_message 根据消息类型将消息分到到 lua 服务注册的回调函数中。

总结一下，snlua 帮 lua 服务做了如下工作：

- 创建服务上下文 skynet_context
- 创建 lua_State
- 分配并绑定唯一 handle
- 创建服务消息队列

执行指定 lua 服务脚本

在最后一步中，lua 服务脚本会通过 skynet.start 启动服务，后者通过 c.callback 完成回调函数的替换。之后 **snlua 便成功代理了 lua 服务，它收到的消息都会转发给 lua 层的 dispatch_message**。

launcher 服务

skynet 中所有的 lua 服务都是通过 snlua 来承载的，skynet 提供了一个 lua 服务 launcher.lua(skynet/service/下)专门用来启动其它 lua 服务，launcer 服务本身通过 skynet.launch(“snlua”, “launcher”)来创建，而其它的 lua 服务则更推荐使用 skynet.newservice(“console”)来启动:

```
function skynet.newservice(name, ...)

        return skynet.call(".launcher", "lua" , "LAUNCH", "snlua", name, ...)end
```

根据前面 skynet.call 的原型，skynet.call 向名为”.launcher”的服务发送一条类型为”lua”的消息，之后的参数便是消息数据，一般来说，消息的第一个字段代表命令，如这里向”.launcher”服务发送了一个”LAUNCH”命令。