文档

2017年4月26日 19:05

1. 简介
2. 编译
3. 编写一个echo用例
4. 程序的运行时解析
5. 其他

简介

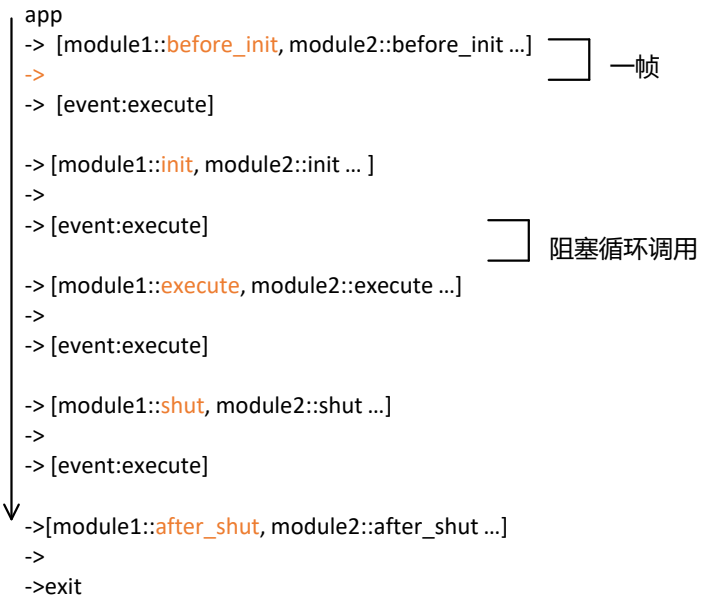
这是一个组件式的游戏服务器框架，其设计意图是帮助团队能够尽可能容易的写出耦合度低的代码。

特性

1. 所有的module都完全独立，只能通过event和其他module产生交互行为，由application驱动。
2. 提供args ，可以像动态语言一样方便的传参
3. 支持动态的创建和销毁module
4. 支持lua module，让c++和lua 更好的结合，可以通过构建多个lua module来水平扩展逻辑。
5. 支持轻量的集群（待实现

程序的运行示意

1. 静态



进程的状态

帧

一次服务器进程循环可以认为是一帧（默认20ms）

1. 在每帧中会先处理当前服务器状态的函数调用
2. 如果遇到动态创建Module的情况，则会在状态的函数调用完成后，进行垫帧
3. 在完成上面两项操作后，会驱动一次事件处理。完成listen , dispatch 中真正的函数处理。

事件处理

垫帧

由于所有模块的执行都是依据，befor_init、 init、 execute

等阶段顺序执行的，并且服务也保证了每个阶段必然是在不同的处理帧上的。所以在遇到动态创建Module的情形时，app会将新创建的module的 阶段保存在临时的帧队列中，在接下来的帧运行过程中每帧弹出。

核心接口

1. eid::get_module
    a. 通过module的名称获得module的id，只能用于静态申请的module。
2. eid::new_dynamic_module
    a. 通过module的名称动态创建一个对象（需要在c++中声明好对象类型
3. eid::delete_dynamic_module
    a. 通过module的id在运行时删除这个module

1. event::listen(module_id, event_id, args, func)

a.

module_id	侦听所有往这个module发送的事件，一般使用module自己的id
event_id	事件id，可以在event_list 中查找
args	传出的参数列表，需要保证压入和弹出的顺序。
func(args)	事件处理函数

2. event::dispatch(module_id, event_id, args, callback)

a.

module_id	发往目标的module_id
event_id	
args	
callback	有些发出事件可以被立即处理，会调用callback回调函数

lua层相关

由于整个框架是基于事件的，在lua_script中已经默认完成了事件和lua层相关的绑定，
所以在使用lua的过程中只需通过事件id就能进行模块之间的访问，只需在c++层做好创建，重载，销毁 lua_state 等接口。无需各种功能向的绑定

1. 使用lua模块
    a. 在c++层创建一个lua_proxy的module，用于管理lua状态机的创建，重载，销毁等。
    b. 在lua_proxy中向LuaScriptModule（ lua层的绑定器 ）发送事件创建一个正真的lua状态机。
    c. 如果这个lua_state module 被用于动态创建的话则需显示声明 REGISTER_CLASS，后通过new_dynamic_module事件进行自动创建。
    d. 如果这个lua_state module 是静态的（推荐方式），则只需通过app.regist_module注册。

```
module = {
        before_init = function() end,
        init = function() end,
        execute = function() end,
        shut = function() end,
        after_shut = function() end,
}
```

lua中module对象的声明方式，其运行机制和c++ module类似。

预设的功能模块（都是由module实现

1. acceptor
2. connector
3. log
4. timer
5. lua_binder
6. dbproxy

可以通过宏WATCH_PERF 开启模块性能监控

可以通过宏**WATCH_PERF** 开启模块性能监控