本文主要讲述如何使用 network 模块提供的接口编写通讯协议文件, 本文假设读者已经熟读Lua 参考手册

## 协议文件格式

1. 协议文件必须返回一个 table, 返回的 table 必须包含 create\_request() 和 create\_response() 两个函数
2. create\_request() 和 create\_response() 必须分别返回一个 table, 返回的 table 必须包含 serial() 和 unserial() 两个函数
3. serial()函数用于处理协议的序列化; unserial()函数用于处理协议的反序列化;
4. create\_response() 返回的 table 中必须包含一个 retcode 表示错误码

一个最简单的协议文件格式如下所示:

local network = require("network")  
  
local function request\_serial(self, buffer)  
end  
  
local function request\_unserial(self, buffer)  
end  
  
local function response\_serial(self, buffer)  
 network.writenumber(buffer, self.retcode)  
end  
  
local function response\_unserial(self, buffer)  
 self.retcode = network.readnumber(buffer)  
end  
  
local protocol = {}  
function protocol.create\_request()  
 local request = {}  
  
 request.serial = request\_serial  
 request.unserial = request\_unserial  
 return request  
end  
  
function protocol.create\_response()  
 local response = {}  
 response.retcode = 0x00000000  
  
 response.serial = response\_serial  
 response.unserial = response\_unserial  
 return response  
end  
  
return protocol

## 结构定义格式

1. 如果需要在多于一个 lua 文件中传递一个 结构, 那么这个 结构 需要定义
2. 以下的示例中, 定义了一个结构, 这个结构包含两个字段, 分别是 player\_id 和 activity\_id

local network = require("network")  
  
local function response\_serial(self, buffer)  
 network.writenumber(buffer, self.player\_id)  
 network.writenumber(buffer, self.active\_id)  
end  
  
local function response\_unserial(self, buffer)  
 self.player\_id = network.readnumber(buffer)  
 self.active\_id = network.readnumber(buffer)  
end  
  
local protocol = {}  
function protocol.create\_request()  
 error("not implemented")  
end  
  
function protocol.create\_response()  
 local response = {}  
 response.player\_id = 0 -- 玩家 ID  
 response.activity\_id = 0 -- 活动 ID  
  
 response.serial = response\_serial  
 response.unserial = response\_unserial  
 return response  
end  
  
return protocol

## 协议序列化接口

通用规则: 1. network.read 前缀的函数通常用于 unserial() 函数中 2. network.write 前缀的函数通常用于 serial() 函数中

1. network.readnumber() 和 network.writenumber() 用于 序列化 和 反序列化 一个数字

network.writenumber(buffer, player\_id)  
local player\_id = network.readnumber(buffer)

1. network.readstring() 和 network.writestring() 用于 序列化 和 反序列化 一个字符串

network.writestring(buffer, nickname)  
local nickname = network.readstring(buffer)

1. network.readarray() 和 network.writearray() 用于 序列化 和 反序列化 一个序列  
   使用这组函数, 序列化 的数据必须满足以下两个要求
2. 序列化 的数据必须是一个序列 (详见 lua参考手册中对于序列的定义)
3. 序列里的每一个元素必须是一个结构 (即必须提供 serial 和 unserial两个 function )

network.writearray(self.tbl\_activity, buffer)  
  
-- create\_activity 用于创建结构的函数  
local create\_activity = require("netimpl.common.activity").create\_response  
network.readarray(self.tbl\_activity, buffer, create\_activity)

1. network.readnumberarray() 和 network.writenumberarray() 用于 序列化 和 反序列化 一个 number 序列

network.writearray(self.tbl\_activity\_id, buffer)  
  
network.readarray(self.tbl\_activity\_id, buffer)

1. network.readstringarray() 和 network.writestringarray() 用于 序列化 和 反序列化 一个 string 序列
2. network.readtable() 和 network.writetable() 用于 序列化 和 反序列化 一个 table 使用这组函数, 序列化 的数据必须满足以下两个要求
3. 序列化 的数据必须是一个table
4. table里的每一个key必须是number, string其中一种类型, value必须是一个结构 (即必须提供 serial 和 unserial两个 function )

local create\_activity = require("netimpl.common.activity").create\_response  
  
-- create\_activity 用于创建结构的函数  
-- 序列化一个 tbl\_activity 的 table, table 里的 key 是结构里的 activity\_id 字段  
network.readtable(tbl\_activity, buffer, create\_activity, "activity\_id")  
  
network.writetable(tbl\_activity, buffer)

1. daserver 网络其它接口
2. daserver.response(current, response)  
   current 为 execute()　方法传递过来的参数  
   response 为　回复请求, 一般情况在[协议序列化lua]已定义
3. daserver.send(session\_id, servant\_name, request)  
   session\_id 为execute() 方法传递过来的参数  
   servant\_name 为协议号  
   request 为主动发送的请求, 一般情况在[协议序列化lua]已定义  
   此接口常用于服务器主动向客户端发送协议
4. daserver.call(servant\_name, request, end\_point, session\_id, time\_out)  
   servant\_name 为协议号  
   request 为发送请求, 一般情况在[协议序列化lua]已定义  
   end\_point 可选,用于在连接多个相同协议服务器时进行指定其中一台  
   session\_id 可选,用于模拟客户端请求  
   time\_out 可选,超时,默认10秒  
   此接口常用于向服务器发送请求, 但获取其返回信息
5. daserver.syncCall(servant\_name, request,response,end\_point, session\_id, time\_out)  
   此接口参数与 daserver.call 类似, 但此接口会等待服务器回复后, 接口返回,　并将服务器的回复信息置于 response 中, response 为回复, 一般情况在“协议序列化 lua”已定义
6. daserver.closeSession(session\_id)  
   根据session\_id 关闭连接,常用于主动断开与客户端的连接