## 服务器代码规范

#### 1. 提交SVN提交时的注释规范

2.3.1.0|BUG #32653::【2.3.1.0】【竞技场】服务器卡时多次点击重置有误

说明:  
(1) 2.3.1.0 为版本号  
(2) |后面为具体的禅道BUG Id或具体的修改内容  
(3) 错误示例: 2.3.1.0|抽卡：bug fixed, 原因: 没有写清楚具体的修改内容  
(4) 正确示例: 2.3.1.0|BUG #32653::【0.3.0.11197测试】【竞技场】服务器卡时多次点击重置有误

#### 2. 函数注释规范, 统一采用以下格式

--[[  
-- 注释内容  
--]]

#### 3. 所有代码使用的结构, 必须在 \deploy\common\netimpl\common\ 目录下有定义; 如果在一个 lua 文件内使用一个结构(注意: 这个结构只有这个文件才使用到), 那么必须注释说明这个结构的内容

-- 下面注意很清晰看到 tbl\_cache 这个 table 中, activity\_type 作为 key,   
-- value 存放着一个结构, 结构里包含这两个字段, 分别是 activity\_id 和 activity\_name   
tbl\_cache = {} -- = { [activity\_type] = {{activity\_id = xx, activity\_name = xx}, ...} }

#### 4. 所有Log必须有模块名, 方法名, 并尽量加上 player\_id, 方便具体玩家问题的定位

-- 示例  
-- gw\_task\_oper 表示模块名  
-- submit 表示模块方法名  
-- player\_id, task\_id 相关参数  
-- cant found task id 为具体的错误信息  
log\_error("gw\_task\_oper|submit|player\_id(%d)|task\_id(%d)|cant found task id", player\_id, task\_id)

#### 5.函数与函数之前的空一行

-- 错误示例 - 1  
local function func1()  
end  
  
  
local function func2()  
end

-- 错误示例 - 2  
local function func1()  
end  
local function func2()  
end

-- 正确示例 - 1  
local function func1()  
end  
  
local function func2()  
end

#### 6. 不要在服务器代码中返回文字给客户端

1. 错误信息必须通过错误码通知客户端
2. 公告, 系统邮件 内容统一由策划配置在 server\_language\_cn 配置表中
3. server\_language\_cn 中一条记录包含 lang\_type, lang\_id 和 lang\_str 三个字段
4. lang\_type 表示模块名, lang\_id 表示模块内标识, lang\_str 表示具体的文字信息
5. server\_language\_cn 配置表的内容统一通过 language\_mgr 来访问

local language\_mgr = require("server\_common.language\_mgr")  
local mail\_subject = language\_mgr:fmt\_lang("ACTIVITY\_CLOUD\_BUY", "MAIL\_SUBJECT")

#### 7.除了枚举/全局变量, 所有变量及方法名一律小写，单词之间采用下划线作分隔

注: 常量的定义,及引入常量的变量大写

-- 正确示例  
local retcode = ERRCODE.GWERR\_NO\_GOLD  
  
-- 错误示例  
local retcode = errcode.GWERR\_NO\_GOLD  
  
-- 正确示例  
player:get\_player\_id()  
  
-- 错误示例  
player:GetPlayerID()

#### 8. 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

注:  
(1) 在 deploy\common\netimpl\common\ 目录下增加对应的结构定义文件 (2) 按以下格式写结构

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

错误示例:

function protocol.create\_response()  
 local response = {  
 activity\_id = 0,  
 player\_id = 0,  
 serial = response\_serial,  
 unserial = response\_unserial,  
 }  
  
 return response  
end

#### 9. 如果一个table是一个序列或hash table, 变量命令必须以tbl\_ 开头

-- 示例  
-- local tbl\_player = {}

#### 10. 变量名禁用 .. 字符串连接用法

-- 错误示例  
USE\_TYPE = {  
 [1] = 'GOLD',  
 [2] = 'DIAMOND',  
 [3] = 'STAMINA',  
 [4] = 'ITEM',  
 [5] = 'SOUL',  
 [6] = 'GUILD\_CONTRIBUTE',  
}  
resp.retcode = ERRCODE['GWERR\_NO\_' .. USE\_TYPE[cost\_type]]  
  
-- 正确示例  
resp.retcode = ERRCODE.GWERR\_NO\_SOUL

#### 11. Lua中字符串允许用单引号或双引号括起来, 服务器统一用双引号

-- 错误示例  
log\_debug('test')  
  
-- 正确示例  
log\_debug("test")

#### 12. 在业务逻辑的代码中不要使用协程

#### 13. 尽量避免在循环体内调用syncCall, syncSend

#### 14. 服务器与客户端统一采用load\_table接口来读取静态表

-- 配置表文件 equip 通过 load\_table("tb\_table\_equip") 来访问, 注意: 需要加上 tb\_table\_ 前缀  
local tb\_table\_equip = load\_table("tb\_table\_equip")

#### 15. 玩家登录不需要立即显示的内容不要在gw\_player\_online这条协议返回, 尽量做成打开界面请求的形式

注: gw\_player\_online 协议为客户端登录gateway成功后, 第一条返回给客户端的协议

#### 16. 具体的业务逻辑尽量不要放在dbmgr和loginmgr两个进程

#### 17. 协议Lua文件用进程名缩写字母开头, 但协议公用结构的Lua文件则不需要

#### 18. 下列函数，除了在main.lua里，其它文件不需要require，可以直接使用

\_ENV.log\_info = log\_info  
 \_ENV.log\_debug = log\_debug  
 \_ENV.log\_warning = log\_warning  
 \_ENV.log\_error = log\_error  
 \_ENV.log\_file = log\_file  
 \_ENV.NETDEFINE = NETDEFINE  
 \_ENV.daserver = daserver  
 \_ENV.create\_impl = create\_impl  
 \_ENV.ERRCODE = ERRCODE  
 \_ENV.UPDATEDEFINE = UPDATEDEFINE  
 \_ENV.ITEM\_OPER\_TYPE = ITEM\_OPER\_TYPE  
 \_ENV.MONEY\_OPER\_TYPE = MONEY\_OPER\_TYPE  
 \_ENV.load\_table = load\_table

#### 19. 多个条件分支语句尽量用 table 去实现

错误示例:

if request.update\_type == DEFINE.UPDATE\_TYPE.ADD then  
 ...  
elseif request.update\_type == DEFINE.UPDATE\_TYPE.DEL then  
 ...  
elseif request.update\_type == DEFINE.UPDATE\_TYPE.MOD then  
 ...  
end

正确示例:

local tbl\_func = {}  
tbl\_func[DEFINE.UPDATE\_TYPE.ADD] = function()  
 ...  
end  
  
tbl\_func[DEFINE.UPDATE\_TYPE.DEL] = function()  
 ...  
end  
  
tbl\_func[DEFINE.UPDATE\_TYPE.MOD] = function()  
 ...  
end  
  
-- 调用时  
local func = tbl\_func[request.update\_type]  
if not func then  
 -- 错误处理  
end  
  
func()

#### 20. 公用结构定义规范

参考 https://github.com/yaofei365/NEO/blob/master/1/1.%E6%9C%8D%E5%8A%A1%E5%99%A8%E5%8D%8F%E8%AE%AE%E6%8E%A5%E5%8F%A3%E8%AF%B4%E6%98%8E.md#%E7%BB%93%E6%9E%84%E5%AE%9A%E4%B9%89%E6%A0%BC%E5%BC%8F

#### 21. 创建结构的函数要写全

-- 错误示例:  
local create\_progress\_cache = require("netimpl.common.sys\_red\_packet\_rain\_progress").create\_response  
  
-- 正确示例:  
local create\_sys\_red\_packet\_rain\_progress = require("netimpl.common.sys\_red\_packet\_rain\_progress").create\_response

require 一个结构时, 通常采用create\_xxx的写法(xxx 为结构名, 通常也是文件名)

#### 22. 对于单向请求协议的逻辑处理, 不需要调用 daserver.response() 返回

-- 错误示例:  
local function execute(request, current, session\_id, ip, port)  
 local \_, resp = create\_impl("netimpl.globalmgr.gbm\_red\_packet\_rain\_progress\_sync")  
 log\_debug("gbm\_red\_packet\_rain\_progress\_sync")  
 sys\_red\_packet\_rain\_mgr:update(request)  
  
 resp.retcode = ERRCODE.SUCCESS  
 daserver.response(current, resp)  
 return  
end  
  
-- 正确示例:  
local function execute(request, current, session\_id, ip, port)  
 log\_debug("gbm\_red\_packet\_rain\_progress\_sync")  
 sys\_red\_packet\_rain\_mgr:update(request)  
  
 -- 单向请求, 不需要返回  
 -- resp.retcode = ERRCODE.SUCCESS  
 -- daserver.response(current, resp)  
 -- return  
end

#### 23. 发送协议给客户端的统一写法

-- 错误示例  
local create\_cl\_response = require("netimpl.client.cl\_sys\_red\_packet\_rain\_notice").create\_response  
.....  
local clresp = create\_cl\_response()  
clresp.retcode = ERRCODE.SUCCESS  
clresp.tbl\_red\_packet = tbl\_red\_packet  
globalmgr\_global.send\_to\_all\_player(NETDEFINE.CL\_SYS\_RED\_PACKET\_RAIN\_NOTICE, "netimpl.client.cl\_sys\_red\_packet\_rain\_notice", clresp)  
  
-- 正确示例  
local clresp = require("netimpl.client.cl\_sys\_red\_packet\_rain\_notice").create\_response()  
clresp.retcode = ERRCODE.SUCCESS  
clresp.tbl\_red\_packet = tbl\_red\_packet  
globalmgr\_global.send\_to\_all\_player(NETDEFINE.CL\_SYS\_RED\_PACKET\_RAIN\_NOTICE, "netimpl.client.cl\_sys\_red\_packet\_rain\_notice", clresp)

#### 24. 获取表变量禁止写在循环内部

-- 错误示例  
 for \_, progress in pairs(tbl\_progress) do  
 local red\_packet\_rain\_id = progress\_data.red\_packet\_rain\_id  
 local t = daodb.tb\_sys\_red\_packet\_rain\_progress()  
 ...  
 end  
  
-- 正确示例  
-- 注: 表变量在一个函数中只需要获取一次, 重复使用前 reset() 即可  
 local t = daodb.tb\_sys\_red\_packet\_rain\_progress()  
 for \_, progress in pairs(tbl\_progress) do  
 local red\_packet\_rain\_id = progress\_data.red\_packet\_rain\_id  
 t:reset()  
 ...  
 end