【协议生成工具的输入文件】

client\FireClient\Applications\modules.xml 定义协议模块，每个模块有哪些协议

client\FireClient\Applications\protocols\gsp.xml 定义gsp名字空间下的协议，引用其他需要包含的xml文件

【xml定义协议的格式】

<**protocol** 协议

name 协议名

type 协议号，全局唯一

maxsize 协议包的最大长度

prior 优先级

tolua 绑定lua，具体见下面**<protocol tolua的含义**

>

<variable 定义协议变量

name 变量名。这里可以使用基本类型和自定义类型。

type 变量类型，见下面"**<variable type定义**"

value 如果变量是模板容器类型，这里指名模板Value类型的名字。

key 如果变量是模板关联容器类型，这里指名模板Key类型的名字。

default 默认值

validator 取值范围限定，'['表示包含边界，'('表示不包含边界，会在发送前和接收后验证，下面**validator的定义**

/>

</protocol>

例：

prtocol: <protocol name="CAnswer" type="32532" prior="1" tolua="3" maxsize="128">

variable: <variable name="tasks" type="map" key="int" value="Veterantask"/>

<variable name="zhuanpinfo" type="ZhuanPanInfo" /> "ZhuanPanInfo"是自定义类型

<variable name="submit" type="byte" default="0"/>

<variable type定义

名字 c++ java

-----------------------------------------------------------

byte char byte

int int int

float float float

long int64\_t long

octets GNET::Octets com.goldhuman.Common.Octets

string GNET::Octets java.lang.String \*\*\* encode="UTF-16LE"

list std::list java.util.LinkedList

vector std::vector java.util.ArrayList

set std::set java.util.HashSet

map std::map java.util.HashMap

自定义类型

validator的定义

------------------------------------------------

byte value=[min,max]

int value=[min,max]

float value=[min,max]

long value=[min,max]

octets capacity=max

string capacity=max

list capacity=max \*value=[min,max]

vector capacity=max \*value=[min,max]

set capacity=max \*value=[min,max]

map capacity=max \*key=[min,max] \*value=[min,max]

<bean 自定义数据结构

name 自定义类型名字

>

<variable 变量，同protocol的变量

name 变量名

type 变量类型

validator 取值范围限定

>

<enum 枚举

name 名

value 值

>

</bean>

例：

bean: <bean name="ScenarioQuestInfo" tolua="1">

enum: <enum name="IQ" value="20"/>

<xi:include 引用

href 别的协议文件

>

例：

<xi:include href="task.xml" />

<protocol tolua的含义

【tolua=1】: C-->S 在c++和lua里都可以发送

xml定义：

<protocol name="CSendWatchBattle" type="7011" maxsize="65535" prior="1" tolua="1">

<variable name="roleid" type="long"/>

</protocol>

>> register in rpcgen0.cpp

相关的文件、代码：

>> client\tolua++-pkgs\FireClient\ProtocolDef\knight\gsp\battle\CSendWatchBattle.pkg 绑定lua用

>> client\FireClient\Application\protoluapkg\knight\gsp\battle\CSendWatchBattle.pkg 绑定lua用

>> client\FireClient\Application\ProtocolDef\rpcgen\knight\gsp\battle\\_depends\_CSendWatchBattle\_.hpp 需要引用的头文件

>> client\FireClient\Application\ProtocolDef\rpcgen\knight\gsp\battle\CSendWatchBattle.inc 类的具体实现

>> client\FireClient\Application\ProtocolDef\knight\gsp\battle\CSendWatchBattle.hpp 类的壳

lua里的试用：

local req = knight.gsp.battle.CSendWatchBattle(roleId);

GetNetConnection():send(req);

有几个S-->C的协议使用tolua="1"，但是生成后没有被使用过，

【tolua=2】: S-->C 在c++和lua里都可以接收处理，如果lua处理完后返回true，c++里会继续处理，看CMessageTask::Run()的实现

xml定义：

<protocol name="SGetBagInfo" type="1011" maxsize="65535" prior="101" tolua="2">

<variable name="bagid" type="int" validator="value=[1,)" />

<variable name="baginfo" type="knight.gsp.Bag" />

</protocol>

>> register in rpcgen3.cpp

相关的文件、代码：

>> client\tolua++-pkgs\FireClient\ProtocolDef\knight\gsp\item\SGetBagInfo.pkg

>> client\FireClient\Application\protoluapkg\knight\gsp\item\SGetBagInfo.pkg

>> client\FireClient\Application\ProtocolDef\rpcgen\knight\gsp\item\\_depends\_SGetBagInfo\_.hpp

>> client\FireClient\Application\ProtocolDef\rpcgen\knight\gsp\item\SGetBagInfo.inc

>> client\FireClient\Application\ProtocolDef\knight\gsp\item\SGetBagInfo.hpp

>> client\resource\artres\video\protocolhandler\knight\_gsp\_item.lua 这个文件应该是手写的

function knight\_gsp\_item.SGetBagInfo\_Lua\_Process(p) lua里的处理函数

>> protocolhandlermanager.lua

ProtocolHandlerManager.RegisterProtocolScriptHandler

SDXL.GetProtocolLuaFunManager():RegisterProtocolScriptHandler(787443, knight\_gsp\_item.SGetBagInfo\_Lua\_Process)

void knight::gsp::item::SGetBagInfo::Process(Manager \*, Manager::Session::ID ) c++里的处理函数

【tolua=3】:

C--S 只能从lua里发送

xml定义：

<protocol name="CDrawGift" type="32757" maxsize="65535" prior="1" tolua="3" >

相关的文件、代码：

>> client\resource\artres\video\protocoldef\knight\gsp\activity\fanfanle\cdrawgift.lua 协议的lua实现

>> client\resource\artres\video\protocoldef\protocols.lua 注册create方法，应该是只对S->C协议有用

function RegisterLuaProtocols()

m = require("protocoldef.knight.gsp.activity.fanfanle.cdrawgift")

LuaProtocolManager.getInstance():RegisterLuaProtocolCreator(819189, m.Create)

使用：

local req = require "protocoldef.knight.gsp.activity.fanfanle.cdrawgift".Create()

LuaProtocolManager.getInstance():send(req)

S-->C 只能在lua里处理

xml定义：

<protocol name="SVeteran" type="32541" prior="1" tolua="3" maxsize="2">

<variable name="status" type="byte"/>

</protocol>

相关的文件、代码：

>> client\resource\artres\video\protocoldef\knight\gsp\activity\veteran\sveteran.lua 协议的lua实现

>> client\resource\artres\video\protocoldef\protocols.lua 注册create方法，应该是只对S->C协议有用

function RegisterLuaProtocols()

m = require("protocoldef.knight.gsp.activity.veteran.sveteran")

LuaProtocolManager.getInstance():RegisterLuaProtocolCreator(818973, m.Create)

>> client\resource\artres\video\luaprotocolhandler\hnight\_gsp\_activity\_veteran.lua 这个文件应该是手写的，并手动加入相应协议的process函数

local p = require "protocoldef.knight.gsp.activity.veteran.sveteran"

function p:process() 实现协议的接收处理函数

【无tolua】: 生成c++文件，发送和接收都只在c++里完成

<bean tolua的含义

【tolua=1】: 可以在lua里使用该数据结构

xml定义：

<bean name="ScenarioQuestInfo" tolua="1">

<variable name="questid" type="int" validator="value=[0,)"/>

<variable name="queststatus" type="int" validator="value=[0,)"/>

<variable name="questvalue" type="int"/> 任务状态值

<variable name="round" type="int"/> 任务的环数

<variable name="dstnpckey" type="long" validator="value=[0,)"/>

</bean>

相关的文件、代码：

>> client\tolua++-pkgs\FireClient\ProtocolDef\rpcgen\knight\gsp\task\ScenarioQuestInfo.pkg

>> client\FireClient\Application\protoluapkg\rpcgen\knight\gsp\task\ScenarioQuestInfo.pkg

>> client\FireClient\Application\ProtocolDef\rpcgen\knight\gsp\task\ScenarioQuestInfo.hpp

lua里的使用

local quests = std.vector\_knight\_\_gsp\_\_task\_\_ScenarioQuestInfo\_() -- vector<ScenarioQuestInfo\*>

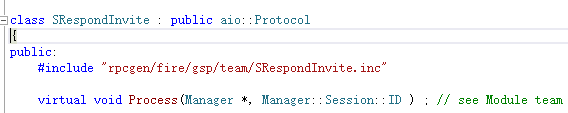
local questinfo = quests[1]

【关于生成代码里面Process的函数实现】

比如协议SRespondInvite，如果已经在程序写了SRespondInvite：：Process函数，则需要在client\FireClient\Application\modules.xml里添加进SRespondInvite



这样下次生成代码时就会只生成Process的声明



否则会生成函数的空定义



另外，生成代码前最后把相应的.hpp文件删除，否则有可能会被忽略而不生成新的