**一 逻辑流程**

1 使用FollowCancvns

1. 与 或 非 三个逻辑接点, 输入为两个端点被执行表示为真
2. 导出为流程文件, 实现C++ Lua C# TS 支持节点运行
3. 服务器和客户端对应设计, 不存在的节点可动态抽象创建, 实现先设计后实现

**二 C++ Actor模型**

1 支持组件

2 支持DB

DB两种应用场景模型

1. 明确数据结构,使用DBActor, 优点:方便数据维护
2. 动态产生各种数据结构, 如智能功能, 优点:NoSQL方式

3 支持异步消息

4 支持组件异步消息

5 代码实现

1. mActorFactory->RegisterActorMsg(#RQ, &Actor::OnMsg<ActorClass, RQ, RS>);

pActorMgr->RegisterActorComMsg (#RQ, &Actor::OnComponentMsg<ComponentClass, RQ, RS>);

1. Async::ProcessPacket->Actor::OnReceiveProcess
2. 接收处理

bool NetCloud::Actor::OnReceiveProcess(NodePacket \*pNodePacket)

{

Auto<TransferPacket> pak = pNodePacket;

if (pak)

{

Hand<AsyncNode> pNetNode = GetNetNode();

if (pNetNode == NULL)

{

ERROR\_LOG("Actor %s Node is NULL, May be not append node", GetID().dump().c\_str());

return AutoNice();

}

switch (pak->mMsgType)

{

case eActorMsg\_response:

{

Auto< AsyncProtocol> protocol = pNetNode->mNodeNet->GetNetProtocol();

auto waitResp = protocol->FindWaitResponse(pak->mRequestID);

if (waitResp)

{

waitResp->mResponsePacket = pak;

RESUME(waitResp->mWaitCoroID);

}

else

ERROR\_LOG("No find wait request %u", pak->mRequestID);

return true;

break;

}

case eActorMsg\_Reqeust:

{

int nCompValue = 0;

AString msgName;

pak->mData.seek(0);

pak->mData.read(nCompValue);

if (!pak->mData.readString(msgName))

{

ERROR\_LOG("Read msg name fail");

return true;

}

auto fun = mActorFactory->mOnMsgFunctionList.*find*(msgName);

if (fun==NULL)

fun = GetMgr()->mOnMsgFunctionList.*find*(msgName);

if (fun != NULL)

{

CoroutineTool::AsyncCall([=]()

{

Auto< AsyncProtocol> protocol = pNetNode->GetNet()->GetNetProtocol();

Auto<TransferPacket> respPacket = protocol->CreatePacket(eNGN\_TransferMsg);

respPacket->mSenderID = pak->mSenderID;

(\*fun)(this, (DataStream\*)&pak->mData, respPacket.getPtr(), nCompValue);

respPacket->mSenderID = GetID();

respPacket->mTargetID = pak->mSenderID;

respPacket->mRequestID = pak->mRequestID;

respPacket->mMsgType = eActorMsg\_response;

if (!SendTo(respPacket.getPtr()))

ERROR\_LOG("Response send fail");

});

}

else

ERROR\_LOG("No register process request function : %s", msgName.c\_str());

}

break;

case eActorMsg\_Notify:

{

int nCompValue = 0;

AString msgName;

pak->mData.seek(0);

pak->mData.read(nCompValue);

if (!pak->mData.readString(msgName))

{

ERROR\_LOG("Read msg name fail");

return true;

}

auto fun = mActorFactory->mOnNotifyMsgFunctionList.find(msgName);

if (fun == NULL)

fun = GetMgr()->mOnNotifyMsgFunctionList.find(msgName);

if (fun != NULL)

{

CoroutineTool::AsyncCall([=]()

{

(\*fun)(this, (DataStream\*)&pak->mData, pak->mSenderID, nCompValue);

});

}

else

ERROR\_LOG("No register process request function : %s", msgName.c\_str());

}

break;

default:

ERROR\_LOG("Can not process %d", pak->mMsgType);

}

**三 简化的网络处理**

1 包执行, 调用对应工厂的ProcessPacket

2 包工厂内聚合了tNetProcess 指针, 直接执行tNetProcess泛化的函数On

3 消息使用协议文本进行生成对应的消息代码类 ExpportBase.h > GenerateProtocol ()

// 保存

SQL\_SaveNoSQLData

{

int mFieldHash;

string mKey;

data mData;

}

// 请求提供字段数据

SQL\_RequestFieldData

{

int mFieldHash;

string mKey;

}

SQL\_ResponseFieldData

{

int mFieldHash;

data mData;

}

// 调取数据

SQL\_LoadNoSQLData

{

string mKey;

bool mbNeedField;

}

// 回复数据

SQL\_ResponseNoSQLData

{

data mData;

data mFieldData;

}

**四 动态DB表结构数据库**

1 NoSQL Actor 由字段表与记录数据表组成

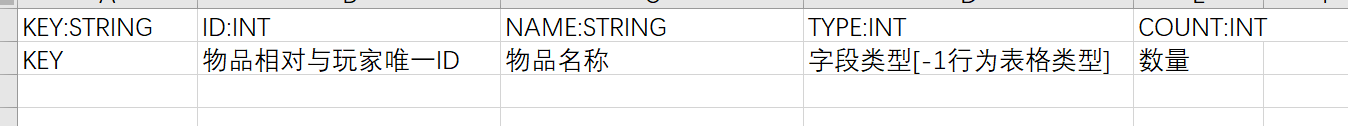
2 使用端保存时, 默认不发送字段数据, NoSQL保存时, 如果字段数据不存在, 则异步询求字段数据, 保存字段数据后, 再保存记录数据. 记录数据只保存数据部分及字段数据的哈希

3 使用端获取时, 可选择是否获取字段数据, 然后根据字段信息进行恢复记录

4 多记录保存, 如物品, 邮件

1. 使用数组列表的原理, 删除记录即把最后位置记录移到到删除的记录位置
2. 只保存多记录的数量, 遍历获取

5 记录结构可以使用协议文件定义,由代码自动生成user端代码 GenerateNoSQLUser



**五 同台电脑内部短接网络**

1 实现共享内存网络

1. 每个连接节点分配一块接收数据的共享内存
2. 连接索引到Gate
3. 发送时, 互斥锁共享内存, 然后再将数据保存到对方接收的数据缓存 (先判断对方缓存是否满足大小, 不满足, 解锁后继续等待后重试)
4. 使用异步发送

2 独立的Gate

3 发送前, 先检查是否在共享内存Gate中

**六 简化的网关网络**

1 Node直接使用MeshNet 的ClientSetNet连接Gate

2 Gate 直接使用ClientSetNet连接其他Gate

**七 客户端Actor与服务器SCActor消息发送与处理**

**1 服务器Login需要创建LoginActor : LoginNetActor, 用于接受客户端连接及消息中转功能**

**2 在LoginActor 的LoginNetComponect 中将连接的客户端创建ClientActor且与Connect绑定**

**3 ClientActor接收服务器Actor的消息并中转给客户端**

**4 客户端请求服务器消息**

1. 服务器逻辑Actor继承SCActor
2. 注册接受客户端消息

// 直通客户端消息

void On(CS\_RequestTest &msg, MSG\_Test &resp, UnitID, int)

{

NOTE\_LOG("Test msg : \r\n%s", msg.dump().c\_str());

resp.mTest = "ok : 888";

CoroutineTool::AsyncCall([=]()

{

tTimer::AWaitTime(5000);

GN\_NotifyNodeInfo testReq;

testReq.mNodeKey = 991111;

MSG\_Test respTestMsg;

AwaitClient(UnitID(1, 111), testReq, respTestMsg, 16000);

NOTE\_LOG(" -------@@@@ %s", respTestMsg.dump().c\_str());

});

}

void Notify(MSG\_Test &msg, UnitID sender, int)

{

NOTE\_LOG("\*\*\*\*\*\*\*\*\*\*\*Client notify %s : %s", sender.dump().c\_str(), msg.dump().c\_str());

tTimer::AWaitTime(3000);

SC\_ResponseTest xx;

xx.mInfo = "=== $$$$$$$$$$$ yyyyyy ===";

SendClientMsg(1, UnitID(1, 111), xx);

}

virtual void RegisterMsg(ActorManager \*pActorMgr) override

{

//pActorMgr->RegisterActorMsg("RQ\_FirstRmbReward", &Actor::OnMsg<TestActor, RQ\_FirstRmbReward, RS\_FirstRmbReward>);

REG\_ACTOR\_MSG(WorkerActor, RQ\_CreateDBTable, RS\_CreateDBTable);

REG\_ACTOR\_MSG(WorkerActor, RQ\_CreatePlayerActor, RS\_CreatePlayerActor);

REG\_ACTOR\_MSG(WorkerActor, CS\_RequestTest, MSG\_Test);

REG\_NOTIFY\_MSG(WorkerActor, MSG\_Test);

SCActor::RegisterMsg(pActorMgr);

}

**(3)客户端发送请求**

public async Task<NiceData> AsyncRequestMsg(UnitID targetActorID, BasePacket requestMsg, int overMilSecond = 10000)  
{  
 var response = await MainStart.mNet.AsyncRequest(new UnitID(104, 1), requestMsg.MsgName(), requestMsg, overMilSecond);  
 if (response != null)  
 response.dump("ok=========");  
 else  
 {  
 LOG.log("Request fail, response null");  
 }  
  
 return response;  
}  
  
public async Task<NiceData> AsyncRequestMsg(UnitID targetActorID, string msgName, NiceData requestMsgData, int overMilSecond = 10000)  
{  
 var response = await MainStart.mNet.AsyncRequest(new UnitID(104, 1), msgName, requestMsgData, overMilSecond);  
 if (response != null)  
 response.dump("ok=========");  
 else  
 {  
 LOG.log("Request fail, response null");  
 }  
  
 return response;  
}

**5 服务器Acotr请求客户端Actor消息**

1. **客户端注册消息处理**

static public async Task<NiceData> On(TestActor actor, GN\_NotifyNodeInfo req)   
{  
 *//var req = new GN\_NotifyNodeInfo();  
 //req.mMsgData = reqData;* LOG.log("888888@@@@@@@@@@@@ " + req.GetType());  
 *//(actor as TestActor).On(req);* req.mMsgData.dump("===========");  
 MSG\_Test t = new MSG\_Test();  
 t.mTest = "pppp+++9999";  
 return t.mMsgData;  
}  
  
static public void Notify(TestActor actor, SC\_ResponseTest req)  
{  
 req.mMsgData.dump("---------------\*\*\*\*\*\*\*\*\*\*");  
}  
  
public override void RegisterMsg()  
{  
 RegisterMsg<TestActor, GN\_NotifyNodeInfo>(On);  
 RegisterMsg<TestActor, SC\_ResponseTest>(Notify);  
}

static public async Task<NiceData> On(TestComponent comp, GN\_NotifyNodeInfo req)   
{  
 *//var req = new GN\_NotifyNodeInfo();  
 //req.mMsgData = reqData;* LOG.log( comp.GetType().Name+ " 6666888888@@@@@@@@@@@@ " + req.GetType());  
 *//(actor as TestActor).On(req);* req.mMsgData.dump("===========");  
 MSG\_Test t = new MSG\_Test();  
 t.mTest = "~~~~~~~~~~~~~~kkkkkpppp+++9999";  
 return t.mMsgData;  
}  
  
static public void Notify(TestComponent actor, SC\_ResponseTest req)  
{  
 req.mMsgData.dump("444444444&&&&&&&&&&\*\*\*\*\*\*\*\*\*\*");  
}  
  
public override void RegisterMsg(ActorManager mgr)  
{  
 mgr.RegisterMsg<TestComponent, GN\_NotifyNodeInfo>(On);  
 mgr.RegisterMsg<TestComponent, SC\_ResponseTest>(Notify);  
}

**(2)服务器发送请求**

template<typename RespMsg>

bool AwaitClient(Int64 clientID, UnitID clientActorID, tBaseMsg &requestMsg, RespMsg &responseMsg, int overMilSecond)

{

//重用优化, 发送时, 直接保存到中转消息中, 等待后不可再使用

static thread\_local SC\_ActorRequestClientMsg msg;

msg.mClientActorID = clientActorID;

msg.mRequestMsgName = requestMsg.GetMsgName();

if (!msg.mRequestMsgData)

msg.mRequestMsgData = MEM\_NEW DataBuffer();

else

msg.mRequestMsgData->clear(false);

requestMsg.serialize(msg.mRequestMsgData.getPtr());

// 因为异步执行, 不可重用

CS\_ResponceServerActorMsg resp;

if (Await(UnitID(Actor\_Client, clientID), msg, resp, overMilSecond) && resp.mResponseMsgData)

{

resp.mResponseMsgData->seek(0);

responseMsg.restore(resp.mResponseMsgData.getPtr());

return true;

}

else

ERROR\_LOG("AwaitClient fail");

return false;

}

bool SendClientMsg(Int64 clientID, UnitID clientActorID, tBaseMsg &notifyMsg )

{

static thread\_local SCS\_NotifyMsg msg;

msg.mActorID = clientActorID;

msg.mMsgName = notifyMsg.GetMsgName();

if (!msg.mNotifyMsgData)

msg.mNotifyMsgData = MEM\_NEW DataBuffer();

else

msg.mNotifyMsgData->clear(false);

if (!notifyMsg.serialize(msg.mNotifyMsgData.getPtr()))

{

ERROR\_LOG("Save notify msg fail : %s", notifyMsg.GetMsgName());

return false;

}

return SendMsg(msg, UnitID(Actor\_Client, clientID));

}