**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

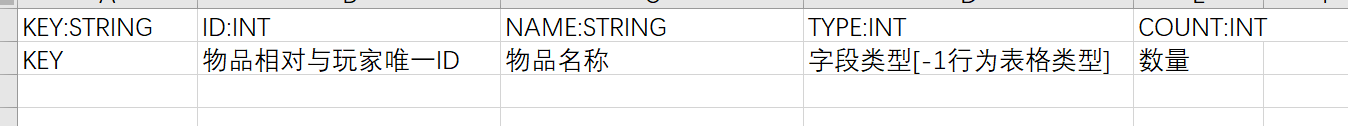
2 使用端保存时, 默认不发送字段数据, NoSQL保存时, 如果字段数据不存在, 则异步询求字段数据, 保存字段数据后, 再保存记录数据. 记录数据只保存数据部分及字段数据的哈希

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