# 设计思路

基于消息的程序架构中，在发送一个request消息后可能会遇到这样的情况

1） 等到response消息，其中response分为肯定回答和否定回答。

2） 在规定时间内没有收到回复。

往往我们的业务中需要等到结果以后再进行其他业务，我提供了两种解决办法：

一种是同步方法，在发送request消息后挂起当前线程，当收到消息或者超时后激活当前线程，根据返回结果，继续处理剩下的业务；

另一种是异步方法，采用回调的方法调用响应函数，维护一个定时器，不停检查是否有超时消息产生，并调用对应回调函数。

# 同步消息等待的设计

class CSyncWait

{

public:

CSyncWait();

~CSyncWait();

bool WaitMsg(int msg1, int msg2 = INVALID\_WAIT\_MSG, int waitTime = INFINITE);

void MsgCome(int msg);

void Reset();

private:

int \_msg1;

int \_msg2;

Poco::Event \_event;

enum {

INVALID\_WAIT\_MSG = 0

};

}

如以上代码所示，主要设计一览无遗。主要通过\_event来控制对所期待消息的等待。

# 异步消息等待的设计

* 参考了磐视客户端的设计，需要用到boost的bind&function、shared\_ptr、lambda；需要使用到定时器，定时器默认间隔200ms。
* 维护了一个超时消息的列表；也维护每个消息对应的回调函数列表。

## 3.1 主要结构体

class CallbackItemBase{

public:

MsgCallback \_callback;

MsgTimeOutCallback \_timeOutCallback;

signed long \_timeout; // will be changed in RegMsgTimeOutCallback function.

};

class NormalCallbackItem :public CallbackItemBase{

public:

CallbackType \_type; // 是响应一次还是总是响应

int \_msg;

};

class MutiCallbackItem :public CallbackItemBase{

public:

std::vector<int> \_msgVec;

};

typedef std::vector<boost::shared\_ptr<CallbackItemBase> > CallbackItemVec;

typedef std::map<int, CallbackItemVec> CallbackMap;

CallbackMap \_callbackMap; // message dispatch map

typedef std::list<boost::shared\_ptr<CallbackItemBase> > TimeOutCallbackItemList;

TimeOutCallbackItemList \_timeoutCallbackList; // timeout callback function list

## 3.2 消息分发流程图



定时器响应函数执行的流程图



# TODO

* 同时引用了boost库和poco库。考虑修改成只用boost库，poco库中缺少boost中的lambda功能。
* 如以下代码，NormalCallbackItem和MutiCallbackItem的判断不在同一层次上，考虑过使用typeid来判断，但是增加不能减少判断的次数。可以考虑增加type关键字判断

boost::shared\_ptr<NormalCallbackItem> pCallbackItem = boost::dynamic\_pointer\_cast<NormalCallbackItem>(\*it);

if (pCallbackItem)

{

if (pCallbackItem->\_msg == msg)

{

\_timeoutCallbackList.erase(it);

break;

}

}

else

{

boost::shared\_ptr<MutiCallbackItem> pCallbackItem = boost::dynamic\_pointer\_cast<MutiCallbackItem>(\*it);

if (pCallbackItem)

{

vector<int>::iterator result =

find(pCallbackItem->\_msgVec.begin(), pCallbackItem->\_msgVec.end(), msg);

if (result != pCallbackItem->\_msgVec.end())

{

\_timeoutCallbackList.erase(it);

break;

}

}

}