**OspSipDemo文档说明**

# Demo功能

Demo分为Client和Server，基本功能仅包括注册（Reg），注销（UnReg），发送消息（SendMsg），订阅（Subscribe），广播消息（Broadcast），以及后来添加的invite，message in dialog。

其中，Client不允许同名用户登录，登录后由Server分配一个唯一SessionID，后续操作用SessionID完成。

只有注册成功后才能后续操作。只有订阅后才收到服务器的广播消息。

## Client设计

Client只有一个OspApp，该App使用了OspExt库封装过。Client采用了OspExt的事务模式以及OspSip库的Sip接口。

Client只模拟单个客户端，所以利用CInstance中的简单状态模型。

## Server功能

同Client。

不过，由于Server要同时处理多个Client事务，采用更加严格的事务模式。

思考：为什么不能采用CInstance提供的状态模型？

***因为CInstance的状态模型是针对所有Inst的，形成了统一的模板，而我们要处理多种事务，每种事务都有自己的状态机。因此不能统一化。每种事务都要根据自身业务特性，细化状态模型，这也是对所有开发者的要求。***

## 公共消息定义

在新的OspSip模型中，所有的消息都变成请求应答模式，因此公共消息体有两个基类。请求消息继承自CEventReq和应答消息继承自CEventRsp。

## 消息段分类

所有的消息号分为三类：

1，模块之间的消息。属于用户自定义业务消息。

2，OspSip和上层模块之间的通信消息，例如OSP\_SIP\_DISCONNECT

3，OspExt库内部消息，以及各个模块内部的内部消息。注意：各个模块内部消息可能相同，因此严禁内部消息在模块之间流通。

## 异常流程处理

1，Client（UA）初始化后，注册到Proxy成功，然后Proxy断链

2，Proxy重新运行，Client自动重新注册Proxy。

2，Client发送一条消息给Server，Server直接丢弃

3，Client设置Server的心跳检测，Server断链

# OspSip框架

Sip协议的细节比较繁琐，本质上就是一个通信协议，Sip头相当于CMonMsg，也有From，To（CMonMsg中叫做SrcID和DstID）。而Sip定义了会话概念，某一次通信就是一次会话。SIP协议本身没有定义消息体（除了SDP描述媒体格式）。对于我们来说，消息体就是一段xml字符串。

OspSip库都是以UA身份存在的。

## Sip概念

《[这里打开一个SIP协议范本](file:///D:\z文档\SIP\sip协议头范本(rfc%203261).txt)》

From，To：SIP协议中用From，和To不仅用来记录源和目的，而且还用来唯一标识一次会话。

会话：SIP中Invite和Subscribe是长期的会话，有发起和结束，中间所有的消息都在此会话中。

事务：一次简单的Message请求和应答，就是一次事务。事务也可以是会话中的一次请求应答。

UA：UserAgent，在SIP中代表终端。例如一个CU是一个UA，一个CMU也是UA。

PROXY：处理消息转发的核心SIP模块。在平台2.0中，它不仅承担了消息投递功能，还承担了消息路由功能。

### SIP所有的方法

这里列举所有的SIP方法，并指明哪些方法是会话内消息。《[kdsip.h](file:///E:\phoenix\10-Common\include\cbb\sip)》

## OspSip初始化

1，OspSipInit

需要给OspSip提供本机的IP和Port以及Proxy的IP和Port。目前我们使用的SIP协议栈都是基于UDP的，由于没有连接，所以连带本机地址也要提供给Proxy。

需要提供本UA的URI，OspSip会使用它向Proxy注册。注册成功后Proxy才会向本UA发送消息。

需要设置dwDefaultOspIID。

我们采用OspSip库通信，而Osp此次本身不承担任何消息通信，因此OspApp收到的消息都是从底层Sip协议栈接收过来，但是SIP与Osp没有绑定关系。绑定关系是在OspSip库完成的。

当我们发送一条消息并期望Rsp消息时，OspSip库就保存了这条消息是由哪个OspApp的那个Inst发送的，以便收到Rsp消息后再转发给该Inst。

当我们收到一条新的Req请求时，OspSip库根据保存的dwDefaultOspIID直接转发给该 Inst。

2，SetXmlCB(&GetSipXmlMsgType);

OspSip在收到一条Sip消息后，为了填充COspSipMsg消息头中的event字段，需要先解析出eventid，才能给用户的OspApp发送消息。

GetSipXmlMsgType回调函数中会在xml中分析“event”字段，并转为eventid

3，InitEventDesc();

**注意！！**我们要求在main函数中注册所有的消息。不仅在GetSipXmlMsgType函数中会将strEvent转为wEvent，而且在公共消息体中也会用到。这全部依赖于消息注册。

4，OspSipRegProxy(240);

注册到Proxy之后，Proxy才会向本UA发送消息。

OspSip会自动续航注册的，用户不需要关心。

同时OspSip会自动向Proxy发送心跳保活，如果Proxy挂掉后，OspSip会向上层发送通知消息。当Proxy再上来后，OspSip会自动重新注册，并通知上层。

## 与OspApp的绑定

请求应答

发送请求消息

发送应答消息

Node的HeartBeat

# OspExt框架

## 事务模型

## 表驱动

## 一个例子，要继承CTask需要完成以下步骤：

1，定义一个自己的CMyTask : public CTask

//子类需要重写的方法

virtual const char\* GetObjName() const

virtual const char\* GetStateName(u32 dwState);

virtual void InitStateMachine();

2，定义自己的状态枚举

3，实现自己的状态函数

状态函数包括ProcMsg类和TimerPoll类，如果对TimerPoll没有要求，可直接使用CTask中的TimerPollSkip函数。

4，/\*在构造函数中调用InitStateMachine()\*/ 用户只需要实现纯虚函数InitStateMachine，直接由ProcMsg()调用InitStateMachine，用户无需关心

5，如果有其他资源需要在任务结束时释放，重载TerminateTask函数

## OspApp中提供的CTask管理