SFUCLIENT概要设计

需求

- 负责与SFUMASTER的连接管理
- 提供接口,供CMU创建SFU会议
- 提供接口,供PAS控制SFU会议
- 相关异常处理
- 考虑混合云场景

相关文档

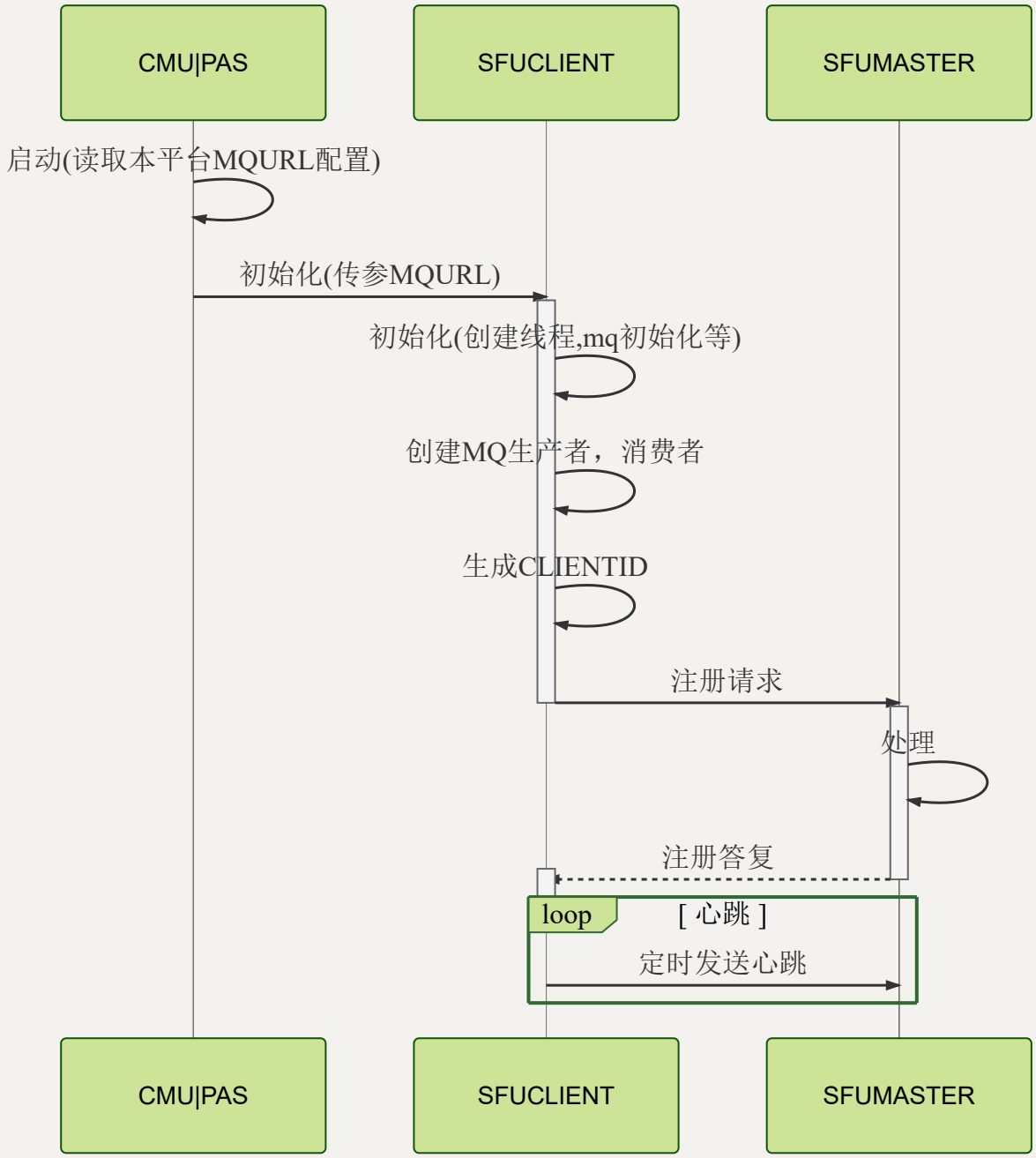
- 阅读本文档前请先阅读SFU系统设计详见confluence:
  云平台产品部/综合业务/ 6.0 SP4 SFU设计文档/SFU系统设计文档
- SFUCLIENT与SFUMASTER间消息定义详见confluence:
  云平台产品部/综合业务/6.0 SP4 SFU设计文档/SFU消息定义
- SFUCLIENT概要设计文档名词字典详见confluence:
  云平台产品部/友商MCU/SFU/平台名词字典
- 了解SFUMASTER模块详见<<SFUMASTER概要设计.md>>
- 了解SFUWORKER模块详见<<SFUWORKER概要设计.md>>

设计原则

- 暂不考虑高可用性
  即CMU/PAS崩溃，则其包含的SFUCLIENT库崩溃前占用相关资源将被释放.ROOM,ROUTER,ENDPOINT,TRANSPORT,PRODUCER,CONSUMER都需重新创建

与SFUMASTER基础连接管理

启动连接时序图



- CMU|PAS启动读取到MQ的地址之后就构造SFUCLIENT对象初始传入连接参数，无需手动调用注册,也不感知是否注册成功
  CLIENT的注册机制限定为内部的资源管理模式，业务不感知
- CLIENTID作用：用于SFUMASTER,SFUWORKER标识资源归属
  CMU/PAS异常情况,MASTER/WORKER需要根据清除对应CLIENT的数据
- 混合云情况,CMU/PAS重新构造一个CLIENT连接HCM

接口说明

SFUCLIENT提供两种粒度的接口使用方式

- 第一种由业务自己初始化管理SFUCLIENT的缓存
- 第二种由SFUCLIENT自己管理缓存,业务只调接口

接口

```
//第一种使用方式接口
namespace SFUCLIENT
{
    //业务自己维护client缓存
    typedef void (*CB_NTF)(EMMsgNtf event, const void* content);
    typedef void (*CB_RPC)(EMMsgRpc event, const void* content, const uintptr_t context);

    class CSFUClient
    {
    public:
        //SFUCLIENT宿主信息, 业务填写自身信息
        class CHost
        {
            ...
            private:
                string m_MOID;
                string m_roomMOID;
                string m_servicedomainMOID;
        };

        class CServerInfo
        {
            ...
            private:
```

```
                string m_mqUrl;
                string m_MOID;
                string m_roomMOID;
        };

    public:
        CSFUClient(const CHost& host,
                   const CServerInfo& serverInfo,
                   CB_RPC cbRPC,
                   CB_NTF cbNtf);
        virtual ~CSFUClient();
    };
}
```

```
//第二种使用方式接口
namespace SFUCLIENT
{
    //由CLIENT初始化缓存
    class CSFUClientInitParam
    {
        ...
        private:
            CServerInfo m_serverInfo;
            CB_RPC m_cbRPC;
            CB_NTF m_cbNTF;
    };
    u32 InitSFUClient(const CSFUClient::CHost& cHost,
                      const set<CSFUClientInitParam>& initParams);
    //返回值CSFUClient*的生命周期由SFUCLIENT负责,业务不可以释放
    //初始化时填入的MQURL会唯一标识并生成一个和进程生命周期相同的SFUCLIENT
    CSFUClient* GetSFUClient(const string& mqUrl);
}
```

基础回调数据结构

基础回调数据结构说明

基于第一次SFUCLIENT概要设计会议评审决议,SFUCLIENT回调给业务的数据统一以数据结构的方式返回

SFUCLIENT定义CRoom类,用于抽象整个会议使用的ROOM.按照操作返回CRoom中某一层级的数据.所返回的CRoom层级的数据中只含单次操作相关信息,不包含全部的信息,
用于业务定位是哪次操作的回调
业务不应在回调中做超时操作.最好收到回调消息就直接抛消息给自己的处理线程
SFUCLIENT会提供工具函数,用于将回调数据结构序列化成json,业务抛消息到
自己线程之后,再反序列化成回调的数据结构,方便使用

基础回调数据结构定义

```
namespace SFUCLINET
{
    /*数据结构层级
    ROOM
        -> ROUTER(6.0sp4只包含一个)
            -> ENDPOINT
                -> TRANSPORT
                    -> STREAM
                    -> SUBSCRIBER(绑定一个STREAM)(业务一般不感知)
    example:
        业务发布一条流
        回调返回CTransport
        业务感知返回的流ID可以使用以下方式获取
        string streamID =
        pcTransport().begin()->GetStreams().begin()->GetID();
        SFUCLIENT保证必定返回以上信息,所以业务可以放心使用以下接口
        不必担心没有数据(不必对每个返回判断是否为空)
    */
    class CSubscriber
    {
        ...
        private:
            string m_ID;
            CStream m_stream;
    };

    class CStream
    {
        enum EMStreamStatus
        {
            EmStreamStatus_OK,
```

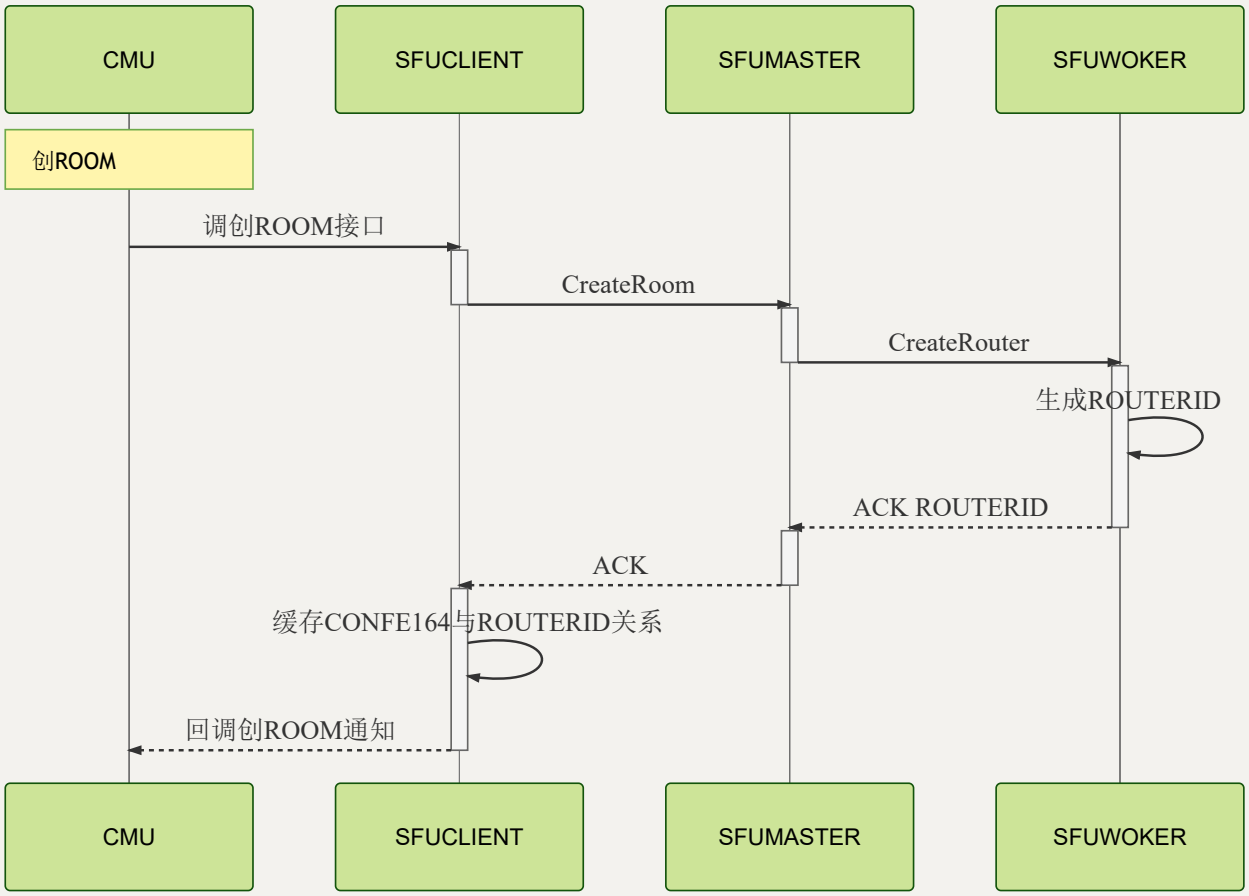
```
        EmStreamStatus_PAUSE
    };
    ...
    private:
        string m_ID;
        EMStreamStatus m_status;
        string m_offerSDP;
        string m_answerSDP;
};

class CTransport
{
    ...
    privte:
        string m_ID;
        map<TypeStreamID, CStream> m_streams;
        map<TypeSubscriberID, CSubscriber> m_subscribers;
};

class CEndpoint
{
    ...
    private:
        string m_ID;
        map<TypeTransportID, CTransport> m_transports;
};

class CRouter
{
    ...
    private:
        string m_ID;
        map<TypeEndpointID, CEndpoint> m_endpoints;
};

class CRoom
{
    ...
    private:
        string m_confE164;
        map<TypeRouterID, CRouter> m_routers;
};
```

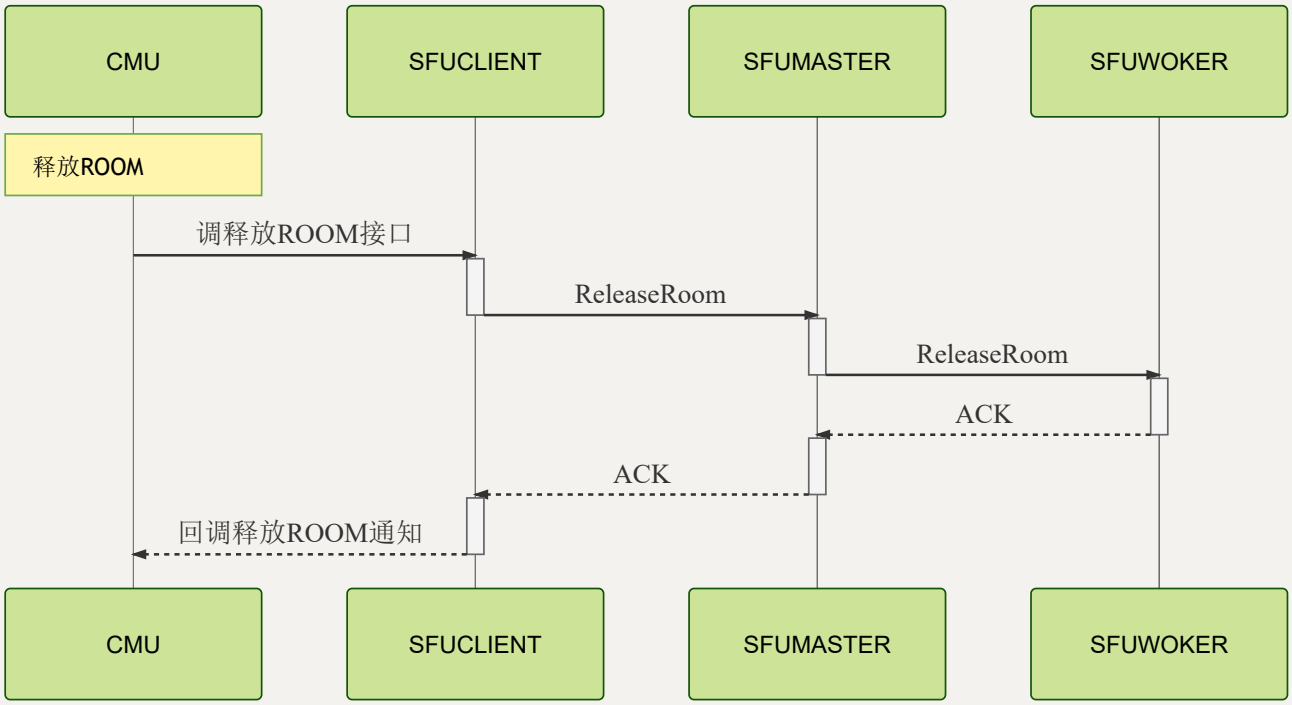
```
    //以上为基础数据结构,在此结构基础上,会附加其他回调信息

    //业务回调函数在SFUCLIENT线程执行.回调函数建议只做收到数据,然后
    //序列化成json,抛消息到业务的处理线程
    //处理线程再反序列化成数据结构使用
}
```

CMU相关

创建/释放SFU ROOM

创建/释放SFU ROOM时序图





- 释放ROOM操作可能回复NACK，原因可能是连接异常，内部系统错误等
  业务感知释放ROOM异常只用于打印。底层将保证SFUCLIENT,SFUMASTER
  及SFUWORKER ROOM相关资源会最终得到释放(文末异常处理模块详述)

CMU接口

```
namespace SFUCLIENT
{
    class CSFUClient
    {
        public:
            const u32 CreateRoom(const s8* confE164,
            const CCreateRoomParam& cCreateRoomParam,
            const uintptr_t context);

            //SFUCLENT所有RELEASE操作一定保证成功
            const u32 ReleaseRoom(const s8* confE164,
            const uintptr_t context);
    };

    //parameter
    class CCreateRoomParam
```

```
    {
        ...
        private:
        //创建ROOM暂时无需参数
    };
}
```

```
namespace SFUCLIENT
{
    enum EMMsgNtf
    {
        EmMsgNtf_ROOM_RELEASED_NTF,
    };
    /*回调消息类型对应的结构体
    CRoom
    */

    enum EMMsgRpc
    {
        EmMsgRpc_CREATE_ROOM_ACK,
        /*回调消息体
        CRoom
        */
        EmMsgRpc_CREATE_ROOM_NACK,
        /*
        TypeErrCode
        */

        EmMsgRpc_RELEASE_ROOM_ACK,
        /*
        CRoom
        */
        EmMsgRpc_RELEASE_ROOM_NACK,
        /*
        pair<CRoom, TypeErrCode>
        */
    };
}
```

创建ROOM客户端数据缓存

```
namespace SFUCLINET
{
    map<TypeConfE164, CRoom> m_confs;
}
```

- 客户端段数据缓存为SFUCLIENT实现所需私有数据,CMU/PAS不感知
- 用于通过会议E164号找到ROUTERID
- 用于保证释放ROOM操作一定成功
  释放ROOM操作出现错误，后续通过拉取MASTER/WORKER信息
  实现资源同步机制

PAS相关

SFU会议控制

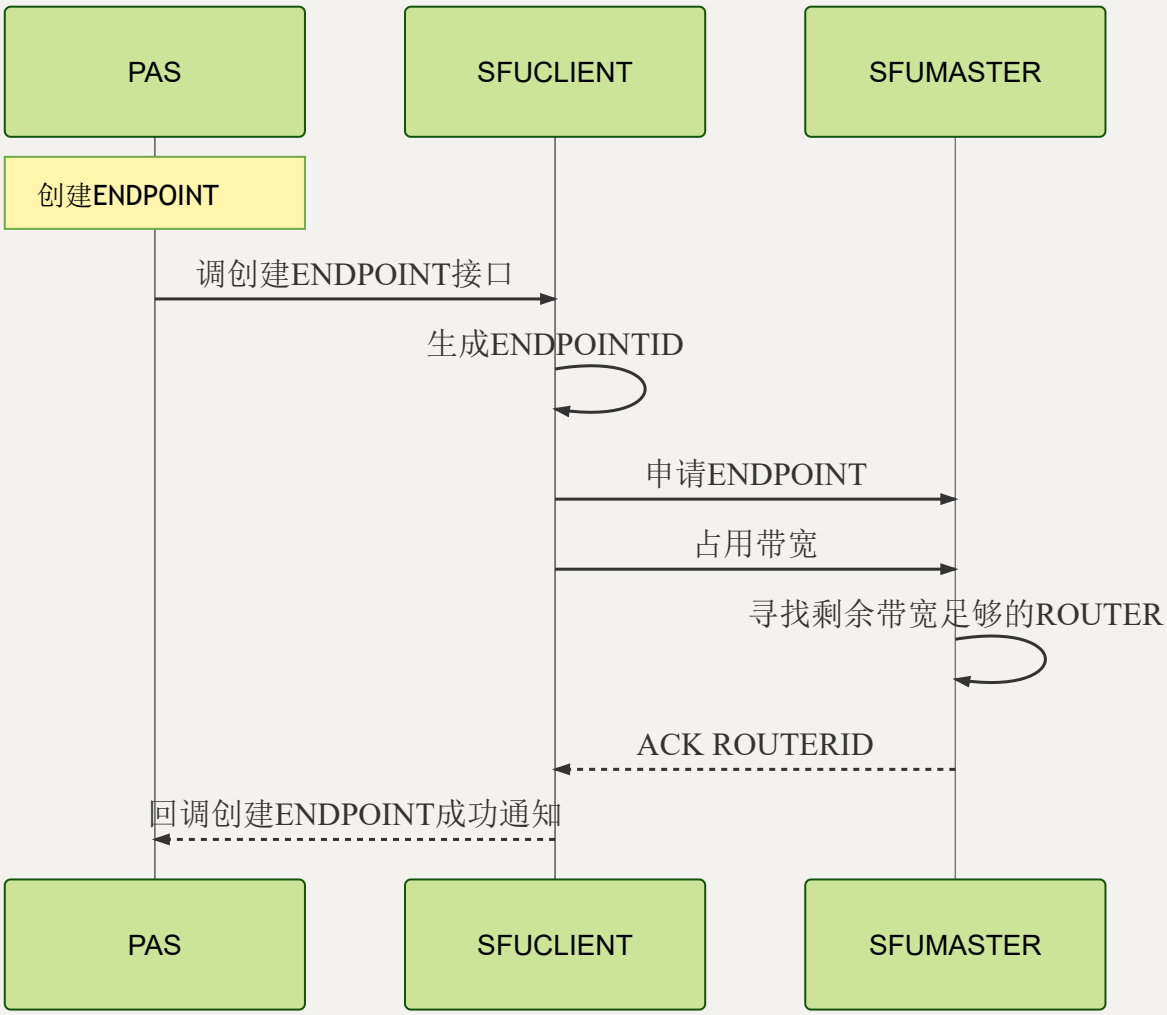
新增ENDPOINT概念

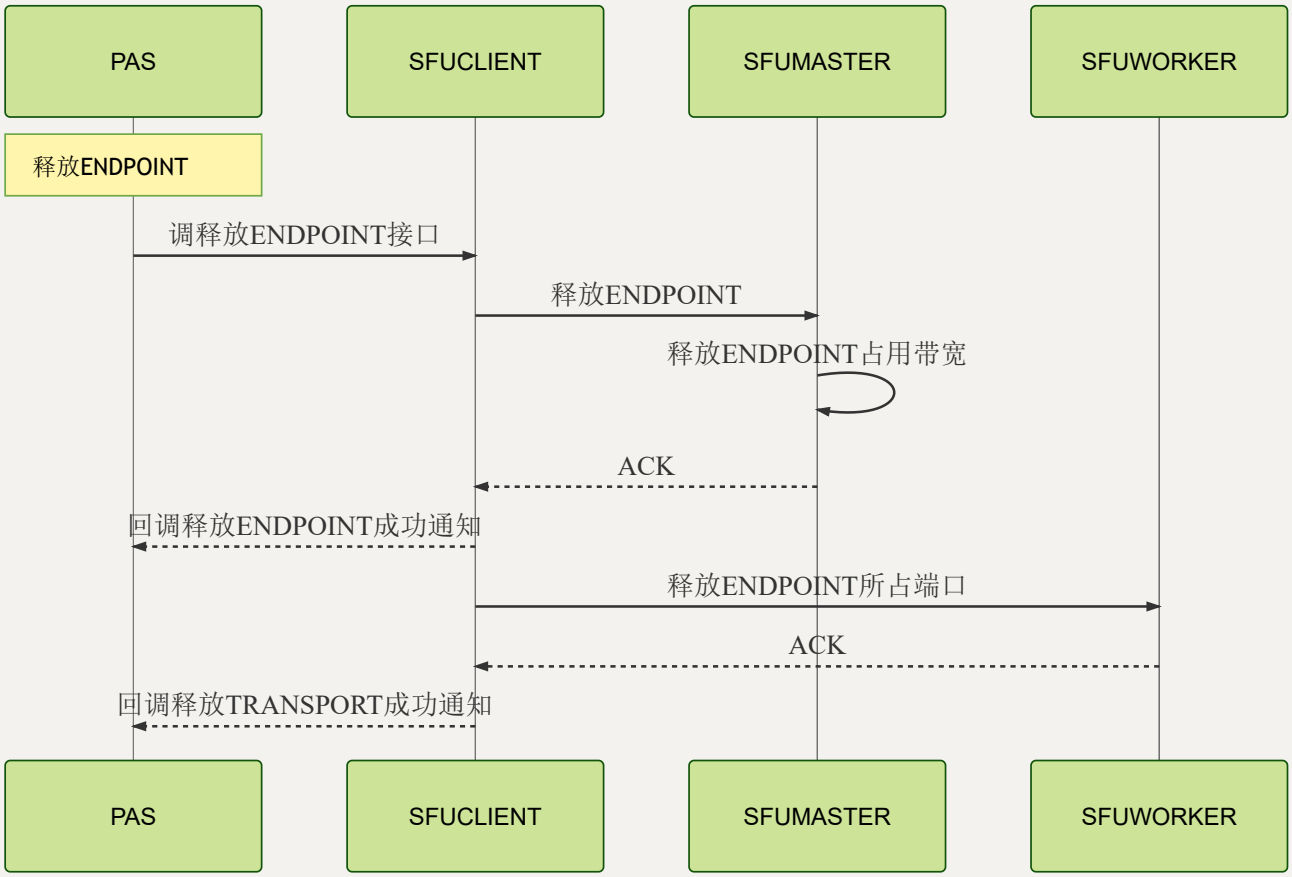
什么是ENDPOINT

ENDPOINT是对具备收发码流能力实体的抽象，包括会议终端，混音器，画面合成器等
ENDPOINT在SFUCLIENT中用于以一个实体的概念去占用一个ROUTER
假设ROUTER剩余的端口/带宽不够一个ENDPOINT使用，则会使用一个新的ROUTER,以保证ENDPOINT始终整个地存在一个ROUTER上

ENDPOINT管理

管理ENDPOINT时序图


PAS
SFUCLIENT
SFUMASTER
创建ENDPOINT
调创建ENDPOINT接口
生成ENDPOINTID
申请ENDPOINT
占用带宽
寻找剩余带宽足够的ROUTER
ACK ROUTERID
回调创建ENDPOINT成功通知
PAS
SFUCLIENT
SFUMASTER



- 释放ENDPOINT可能返回NACK,业务感知NACK只用于打印
  SFUCLIENT将保证后续ENDPOINT及属于ENDPOINT的相关资源将得到释放
- 由于当前系统无统一的ENDPOINT标识。则由CLIENT统一生成ENDPOINT ID.
  业务需要针对各自的ENDPOINT标识和返回的ENDPOINT标识做映射

ENDPOINT接口

```
//interface
namespace SFUCLIENT
{
    class CSFUClient
    {
        public:
            u32 CreateEndpoint(const s8* confE164,
            const CCreateEndpointParam& cParam,
            const uintptr_t context);

            u32 ReleaseEndpoint(const s8* confE164,
            const s8* endpointID
```

```
            const uintptr_t context);
    };

    class CCreateEndpointParam
    {
        ...
        private:
            //被叫ENDPOINT运营商
            //用于后续选择合适的ROUTER
            string m_calledOperator;
            u32 m_dwInitOutBitrate;
            u32 m_dwMinOutBitrate;
    };
}
```

```
namespace SFUCLINET
{
    enum EMMsgNtf
    {
        EmMsgNtf_ENDPOINT_RELEASED_NTF
        /*回调消息类型对应的结构体
        CEndpoint
        */
    };

    enum EMMsgRpc
    {
        EmMsgRpc_CREATE_ENDPOINT_ACK,
        /*
        CEndpoint
        */

        EmMsgRpc_CREATE_ENDPOINT_NACK,
        /*
        TypeErrCode
        */

        EmMsgRpc_RELEASE_ENDPOINT_ACK,
        /*
        CEndpoint
        */
        EmMsgRpc_RELEASE_ENDPOINT_NACK,
```

```
        /*
        pair<CEndpoint, TypeErrCode>
        */
    };
}
```

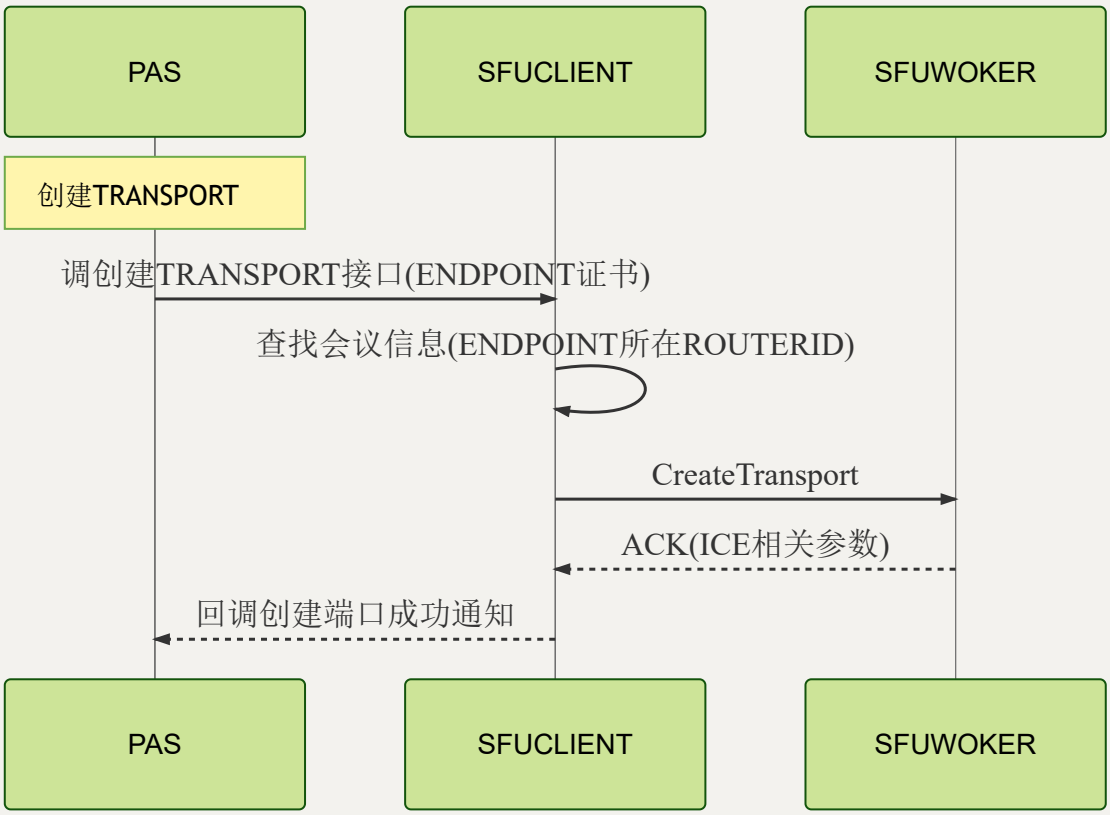
ENDPOINT数据缓存

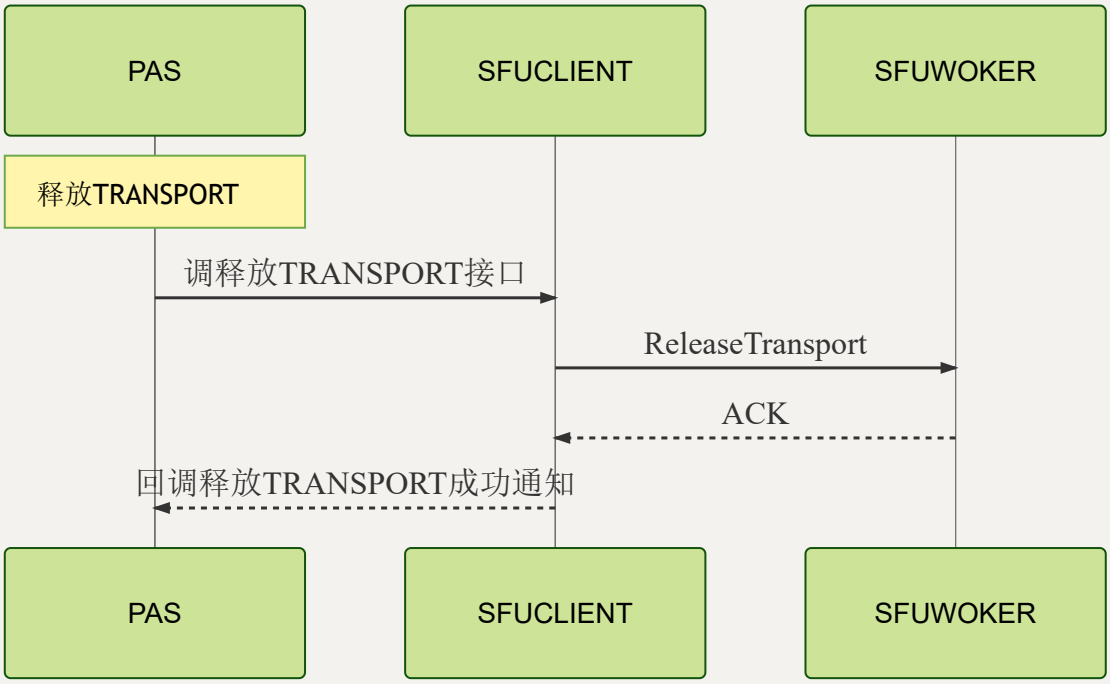
```
namespace SFUCLINET
{
    //将缓存进m_confs单例数据缓存中
    map<TypeConfE164, CRoom> m_confs;
}
```

- 用于后续创建TRANSPORT可以使用之前创建ENDPOINT时的参数
  从而无需业务每次创建TRANSPORT都要再传入相关参数，如：运营商信息
- 用于释放ENDPOINT出现错误，后续同步机制比对CLIENT缓存及MASTER REDIS数据去释放MASTER上ENDPOINT相关资源

TRANSPORT管理

管理TRANSPORT时序图





TRANSPORT接口

```
//interface
namespace SFUCLIENT
```

```
{
    class CSFUClient
    {
        public:
            u32 CreateICETransport(
            const s8* confE164,
            const s8* endpointID,
            const CICETransportCreateParam& cCreateParam,
            const uintptr_t context);

            u32 ReleaseICETransport(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const uintptr_t context);
    };

    //parameter
    namespace SFUCLIENT
    {
        //interface parameter
        class CICETransportCreateParam
        {
            ...
            private:
                string m_strTransportID;
                CFingPrints m_cFingPrints;
        };

        class CFingerPrint
        {
            ...
            private:
                string m_strAlgorithm;
                string m_strValue;
        };
    }
```

```
namespace SFUCLINET
{
    enum EMMsgNtf
    {
```

```
    EmMsgNtf_TRANSPORT_RELEASED_NTF
    /*回调消息对应消息体数据结构
    CEndpoint
    */
};

enum EMMsgRpc
{
    // 创建ICETransport
    EmMsgRpc_CREATE_ICETRANSPORT_ACK,
    /*
    CTransport
    其中CTransport中补充以下数据
    class CTransport
    {
        ...
        private:
            CICEInfo m_ICEInfo;
    };

    class CICEInfo
    {
        class CICEParameter
        {
            ...
            private:
                string m_usernameFragment;
                string m_pwd;
                string m_iceLite;
        };

        class CICECandicate
        {
            ...
            private:
                string  m_foundation;
                string m_priority;
                string m_ip;
                u32 m_port;
        };

        ...
```

```
            private:
                string m_role; //写死为controlled
                CICEParameter m_ICEParameter;
                vector<CICECandicate> m_ICECandicates;
                map<TypeSSRCID, TypeSubscriberID> m_SSRCSubscriberID;
        };
        */
        EmMsgRpc_CREATE_ICETRANSPORT_NACK,
        /*
        TypeErrCode
        */

        // 删除ICETransport
        EmMsgRpc_REMOVE_ICETRANSPORT_ACK,
        /*
        CTransport
        */
        EmMsgRpc_REMOVE_ICETRANSPORT_NACK,
        /*
        pair<CTransport, TypeErrCode>
        */
    };
}
```

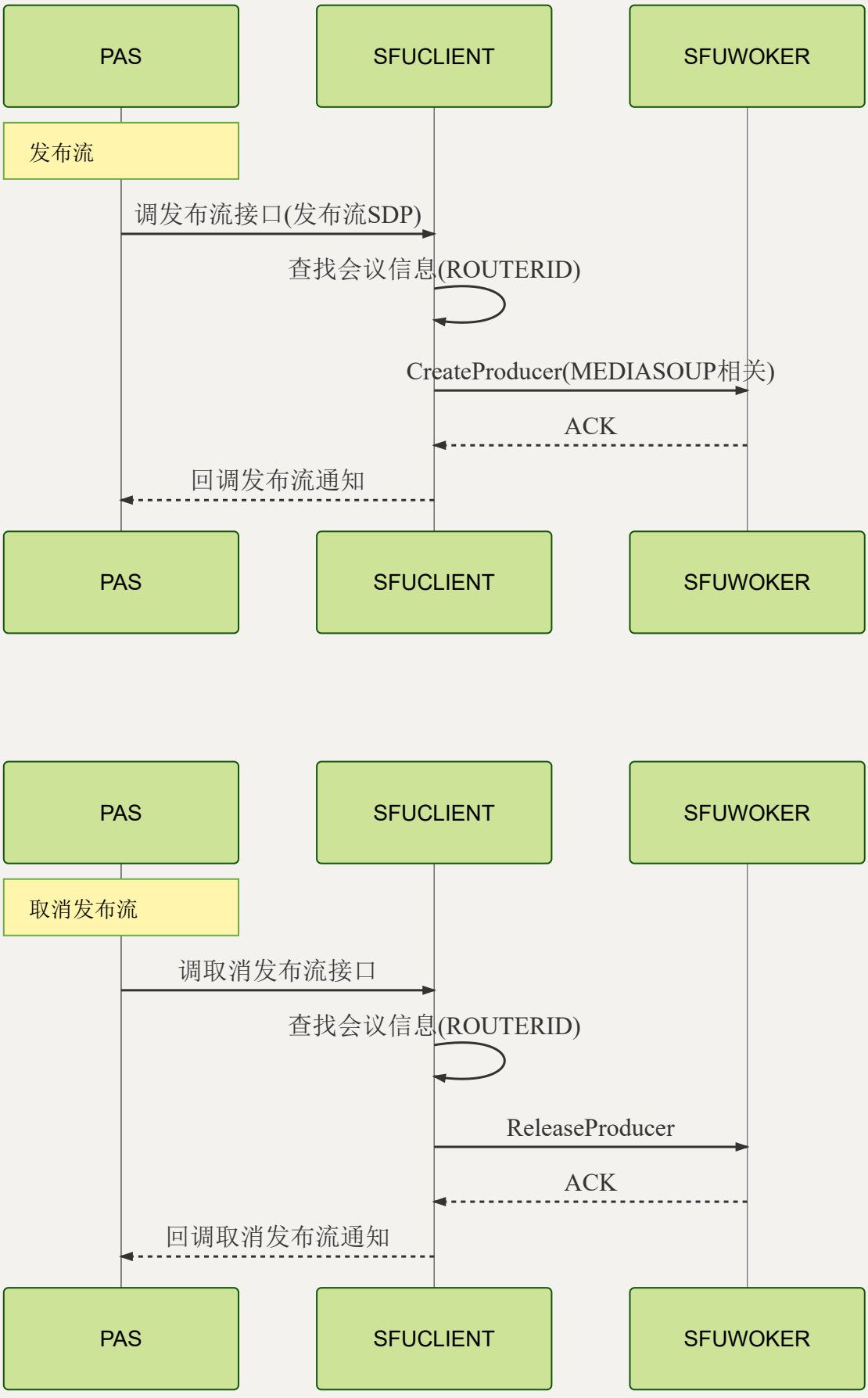
TRANSPORT数据缓存

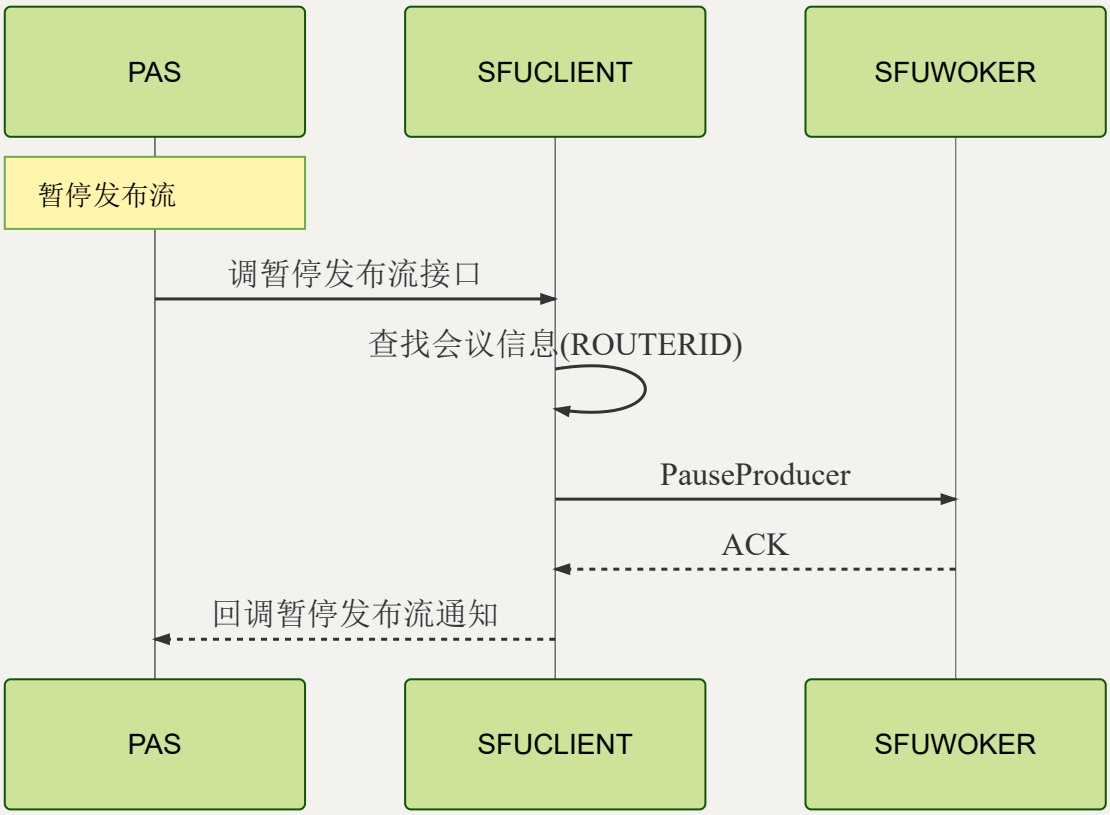
```
namespace SFUCLINET
{
    //将缓存进m_confs单例数据缓存中
    map<TypeConfE164, CRoom> m_confs;
}
```

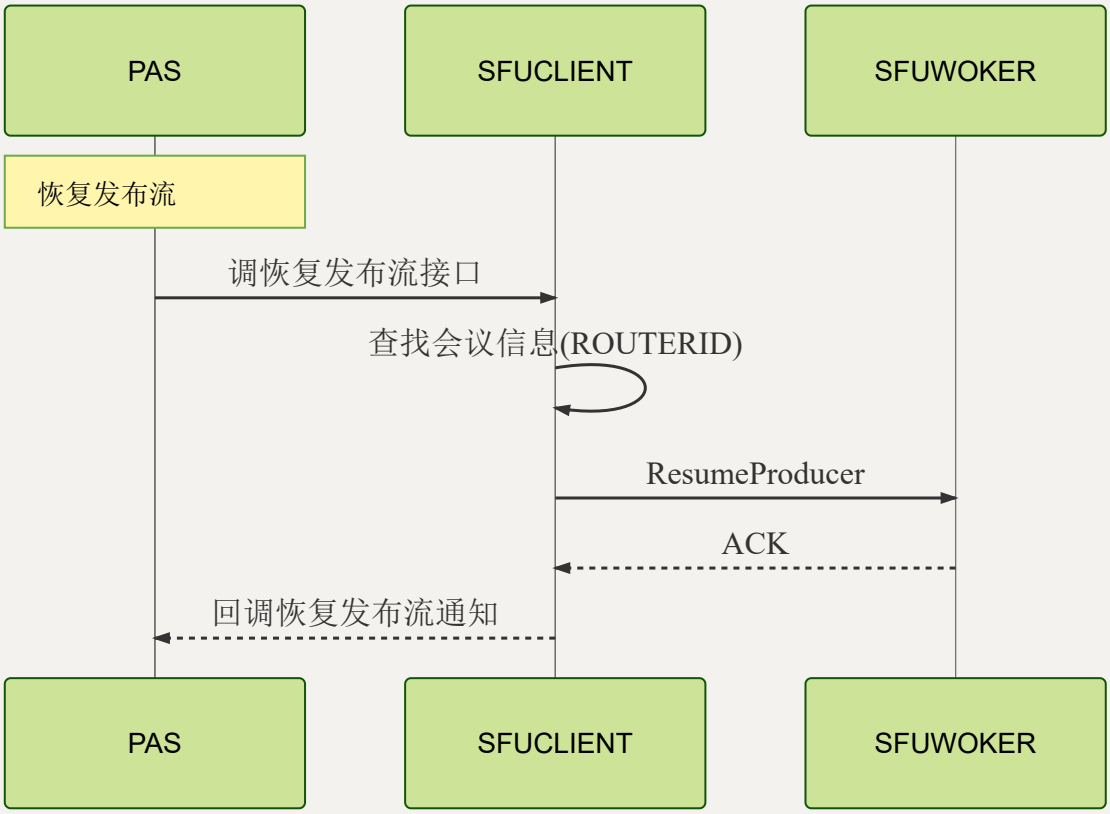
- 用于释放TRANSPORT出现错误，后续同步机制比对CLIENT缓存及WORKER数据去释放WORKER上TRANSPORT相关资源

发布流管理

发布流管理时序


PAS
SFUCLIENT
SFUWOKER
发布流
调发布流接口(发布流SDP)
查找会议信息(ROUTERID)
CreateProducer(MEDIASOUP相关)
ACK
回调发布流通知
PAS
SFUCLIENT
SFUWOKER
PAS
SFUCLIENT
SFUWOKER
取消发布流
调取消发布流接口
查找会议信息(ROUTERID)
ReleaseProducer
ACK
回调取消发布流通知
PAS
SFUCLIENT
SFUWOKER


PAS
SFUCLIENT
SFUWOKER
暂停发布流
调暂停发布流接口
查找会议信息(ROUTERID)
PauseProducer
ACK
回调暂停发布流通知
PAS
SFUCLIENT
SFUWOKER


PAS
SFUCLIENT
SFUWOKER
恢复发布流
调恢复发布流接口
查找会议信息(ROUTERID)
ResumeProducer
ACK
回调恢复发布流通知
PAS
SFUCLIENT
SFUWOKER

SFU会议发布流接口

```
namespace SFUCLIENT
{
    class CSFUClient
    {
        public:
            u32 Publish(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const CStreamParam& streamParam,
            const uintptr_t context);

            u32 Unpublish(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const uintptr_t context);

            u32 PausePublish(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const uintptr_t context);

            u32 ResumePublish(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const uintptr_t context);
    };

    class CStreamParam
    {
        ...
        private:
            string m_SDP;
            u32 m_bitrate; //发布带宽
    };
}
```

```
namespace SFUCLIENT
{
    enum EMMsgNtf
    {
        EmMsgNtf_STREAM_UNPUBLISHED_NTF,
        /*
        CEndpoint
        */
    };

    enum EMMsgRpc
    {
        EmMsgRpc_PUBLISH_ACK,
        /*
        CStream
        CStream需要增加该流offer返回的answer信息
        class CStream
        {
            ...
            private:
                string m_answerSDP;
        };
        */

        EmMsgRpc_PUBLISH_NACK,
        /*
        TypeErrCode
        */

        EmMsgRpc_UNPUBLISH_ACK,
        /*
        CStream
        */

        EmMsgRpc_UNPUBLISH_NACK,
        /*
        pair<CStream, TypeErrCode>
        */

        EmMsgRpc_PAUSE_PUBLISH_ACK,
        /*
```

```
        CStream
        */

        EmMsgRpc_PAUSE_PUBLISH_NACK,
        /*
        pair<CStream, TypeErrCode>
        */

        EmMsgRpc_RESUME_PUBLISH_ACK,
        /*
        CStream
        */

        EmMsgRpc_RESUME_PUBLISH_NACK,
        /*
        pair<CStream, TypeErrCode>
        */
    };
}
```

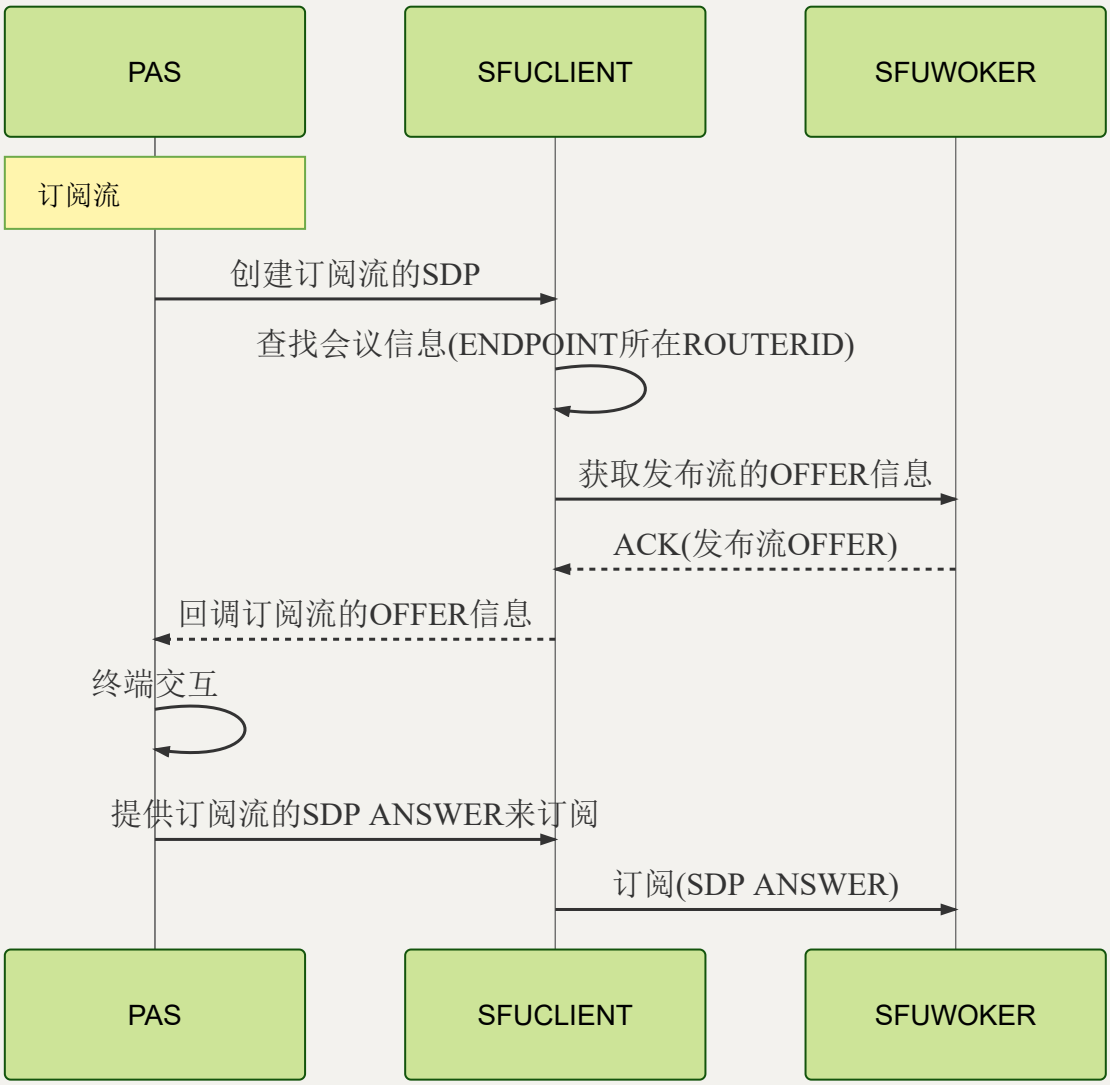
发布流数据缓存

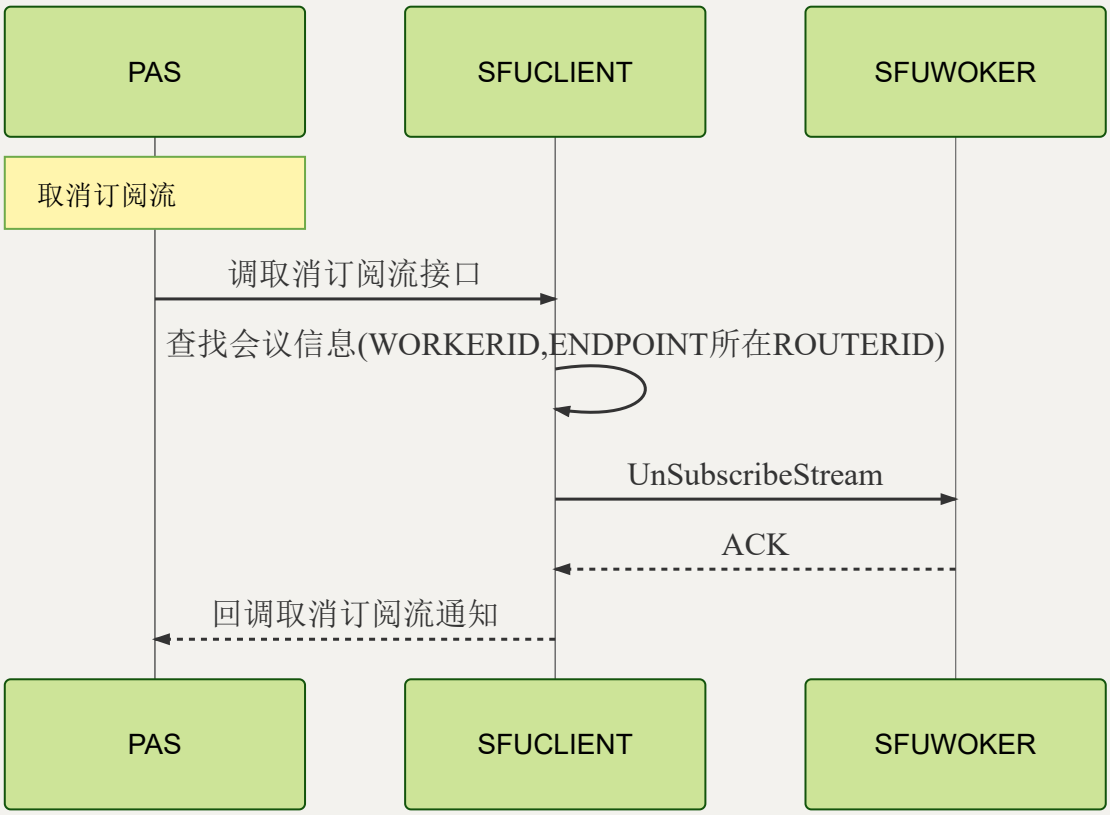
```
namespace SFUCLINET
{
    //将缓存进m_confs单例数据缓存中
    map<TypeConfE164, CRoom> m_confs;
}
```

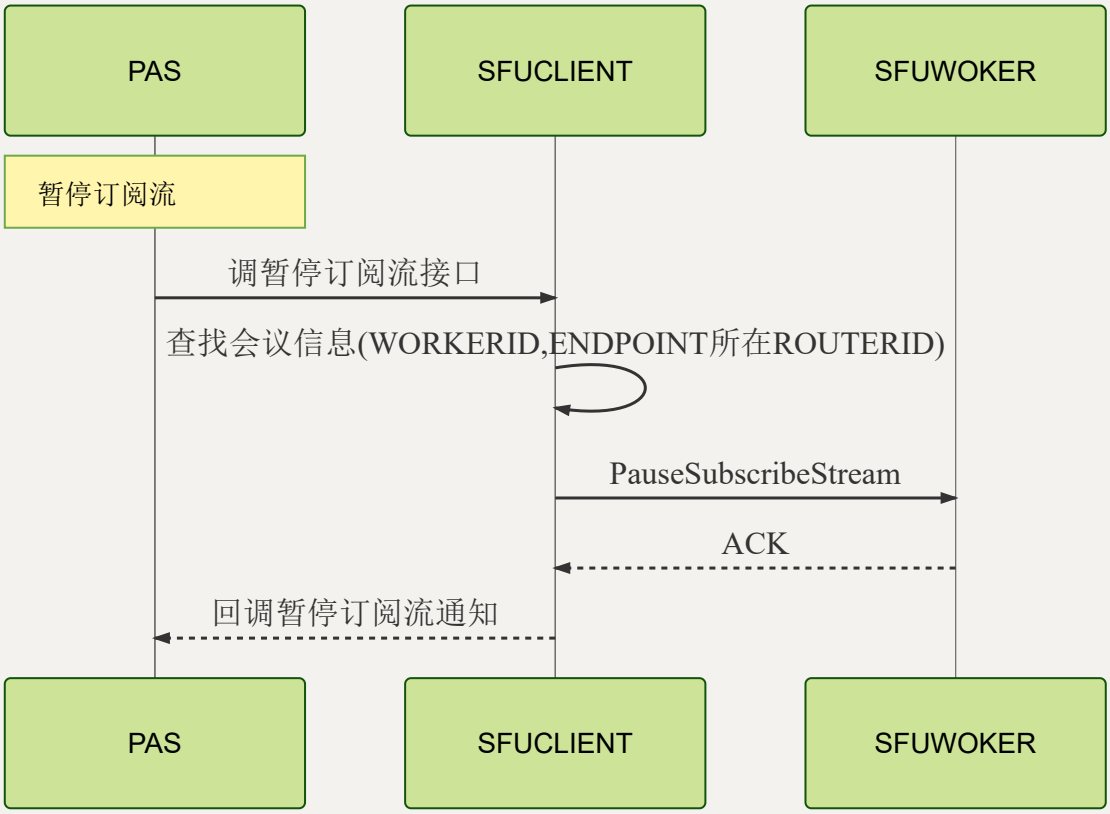
- 用于取消发布出现错误，后续同步机制比对CLIENT缓存及WORKER数据去释放WORKER上STREAM相关资源

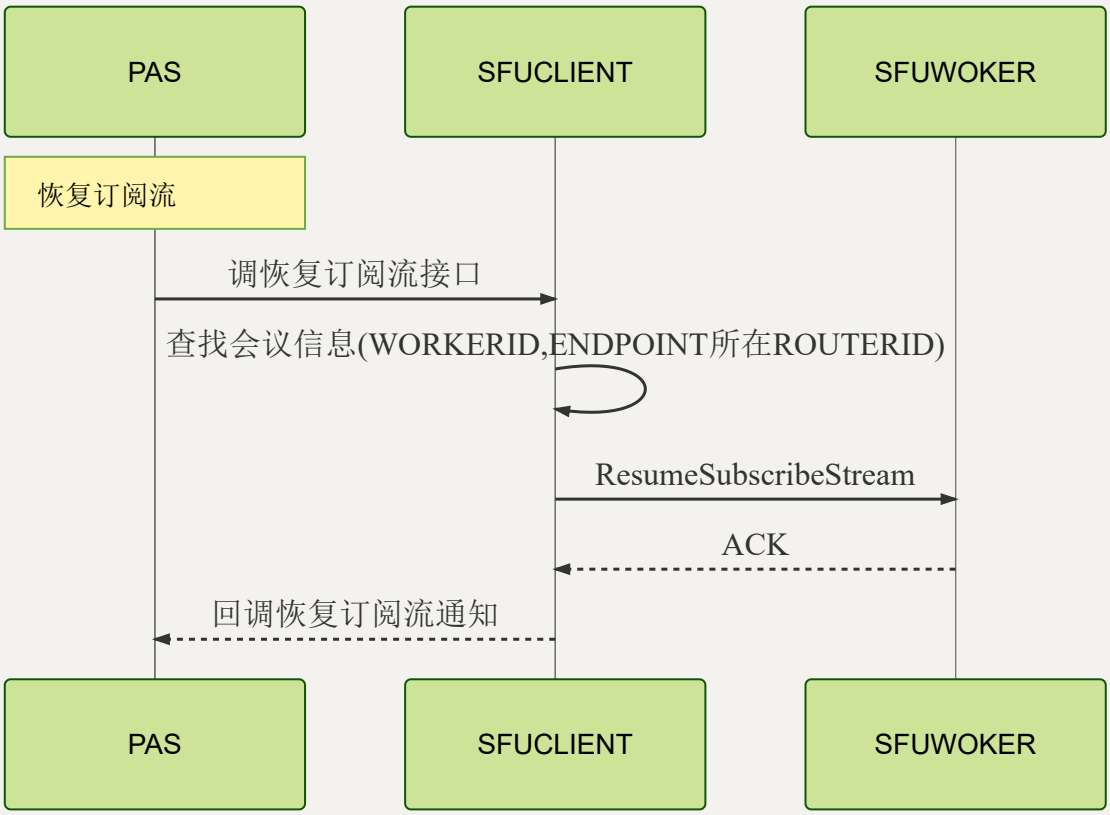
订阅流管理

订阅流管理时序


PAS
SFUCLIENT
SFUWOKER
订阅流
创建订阅流的SDP
查找会议信息(ENDPOINT所在ROUTERID)
获取发布流的OFFER信息
ACK(发布流OFFER)
回调订阅流的OFFER信息
终端交互
提供订阅流的SDP ANSWER来订阅
订阅(SDP ANSWER)
PAS
SFUCLIENT
SFUWOKER


PAS
SFUCLIENT
SFUWOKER
取消订阅流
调取消订阅流接口
查找会议信息(WORKERID,ENDPOINT所在ROUTERID)
UnSubscribeStream
ACK
回调取消订阅流通知
PAS
SFUCLIENT
SFUWOKER


PAS
SFUCLIENT
SFUWOKER
暂停订阅流
调暂停订阅流接口
查找会议信息(WORKERID,ENDPOINT所在ROUTERID)
PauseSubscribeStream
ACK
回调暂停订阅流通知
PAS
SFUCLIENT
SFUWOKER



SFU会议订阅流接口

```
namespace SFUCLIENT
{
    class CSFUClient
    {
        public:
            u32 CreatStreamOffer(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID);

            //调用Subscribe前,需要获取流的OFFER,然后带上ANSWER来订阅
            u32 Subscribe(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* answerSDP,
            const uintptr_t context);

            u32 Unsubscribe(
```

```
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const uintptr_t context);

            u32 PauseSubscribe(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const uintptr_t context);

            u32 ResumeSubscribe(
            const s8* confE164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const uintptr_t context);
    };
}
```

```
namespace SFUCLIENT
{
    enum EMMsgNtf
    {
        EmMsgNtf_STREAM_UNSUBSCRIBE_NTF
        /*
        CEndpoint
        pcEndpoint->GetStreams().begin()->GetID()
        */
    };

    enum EMMsgRpc
    {
        EmMsgRpc_CREATE_STREAM_OFFER_ACK,
        /*
        CStream
        */
        EmMsgRpc_CREATE_STREAM_OFFER_NACK,
        /*
        TypeErrCode
```

```
        */
        EmMsgRpc_SUBSCRIBE_ACK
        /*
        CStream
        */
        EmMsgRpc_SUBSCRIBE_NACK,
        /*
        pair<CStream, TypeErrCode>
        */

        EmMsgRpc_UNSUBSCRIBE_ACK,
        /*
        CStream
        */
        EmMsgRpc_UNSUBSCRIBE_NACK,
        /*
        pair<CStream, TypeErrCode>
        */

        EmMsgRpc_PAUSE_SUBSCRIBE_ACK,
        /*
        CStream
        */
        EmMsgRpc_PAUSE_SUBSCRIBE_NACK,
        /*
        pair<CStream, TypeErrCode>
        */

        EmMsgRpc_PAUSE_SUBSCRIBE_ACK,
        /*
        CStream
        */
        EmMsgRpc_PAUSE_SUBSCRIBE_NACK,
        /*
        pair<CStream, TypeErrCode>
        */
    };
}
```

订阅流数据缓存

```
namespace SFUCLINET
{
    //将缓存进m_confs单例数据缓存中
    map<TypeConfE164, CRoom> m_confs;
}
```

- 用于取消订阅出现错误，后续同步机制比对CLIENT缓存及WORKER数据去释放WORKER上CONSUMER相关资源

其他控制接口

```
namespace SFUCLIENT
{
    class CSFUCLIENT
    {
        public:
            //other control
            u32 RequestKeyFrame(
            const s8* confe164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const uintptr_t context);

            u32 SetPreferLayer(
            const s8* confe164,
            const s8* endpointID,
            const s8* transportID,
            const s8* streamID,
            const u32 length,
            const u32 width,
            const uintptr_t context);

            u32 SetMaxIncomingBitrate(
            const s8* confe164,
            const s8* endpointID,
            const s8* transportID,
            const u32 bitrate,
            const uintptr_t context);
```

```
            /*next edition
            u32 CreatePipeTransport(s8* confe164, s8* carrier, s8* remoteIp, u16 rtpPort, uir
            u32 ConnectPipeTransport(s8* confe164, s8* transport_id, s8* remoteIp, u16 rtpPor
            u32 CreateRTPTransport(s8* confe164, s8* carrier, s8* remoteIp, u16 rtpPort, u16
            u32 ConnectRTPTransport(s8* confe164, s8* transport_id, s8* remoteIp, u16 rtpPort
            */
    }
}
```

```
//message
namespace SFUCLINET
{
    enum EMRpcMsg
    {

        // 设置最大上行带宽
        EmRpcMsg_SET_MAXINCOMINGBITRATE_ACK,
        /*
        CTrarnsport
        */
        EmRpcMsg_ET_MAXINCOMINGBITRATE_NACK,
        /*
        CTrarnsport
        */

        // 请求关键帧
        EmRpcMsg_REQUEST_KEYFRAME_ACK,
        /*
        CStream
        */
        EmRpcMsg_REQUEST_KEYFRAME_NACK,
        /*
        CStream
        */

        // 设置接收层
        EmRpcMsg_SET_PREFER_LAYER_ACK,
        /*
        CStream
        */
        EmRpcMsg_SET_PREFER_LAYER_NACK
```

```
        /*
        CStream
        */

        /*next edition
        // 创建RTPTransport
        EmRpcMsg_CREATE_RTPTRANSPORT_ACK,
        EmRpcMsg_CREATE_RTPTRANSPORT_NACK,
        // 连接RTPTransport
        EmRpcMsg_CONNECT_RTPTRANSPORT_ACK,
        EmRpcMsg_CONNECT_RTPTRANSPORT_NACK,
        // 删除RTPTransport
        EmRpcMsg_REMOVE_RTPTRANSPORT_ACK,
        EmRpcMsg_REMOVE_RTPTRANSPORT_NACK,
        // 创建PIPETransport
        EmRpcMsg_CREATE_PIPETRANSPORT_ACK,
        EmRpcMsg_CREATE_PIPETRANSPORT_NACK,
        // 连接PIPETransport
        EmRpcMsg_CONNECT_PIPETRANSPORT_ACK,
        EmRpcMsg_CONNECT_PIPETRANSPORT_NACK,
        // 删除PIPETransport
        EmRpcMsg_REMOVE_PIPETRANSPORT_ACK,
        EmRpcMsg_REMOVE_PIPETRANSPORT_NACK,
        */
    };
}
```

异常处理

接口统一超时处理

SFUCLIENT任何一次接口调用都是异步操作
SFUCLIENT会为每次操作设定超时时间
规定时间内未获得操作通知，则会回调通知业务操作超时
暂定单次操作超时时间为 ***10S***

倘若操作超时，但是实际操作是成功的。则CLIENT和MASTER/WORKER间的状态
以CLIENT为准(判定为不成功)。后续走同步状态处理

CLIENT异常崩溃

当前数据都保存在程序内部.
异常崩溃,数据自动丢失.重启之后构造新的CLIENTID进行注册保活
针对CLIENT快速重启情况：

- MASTER将检测到超时的旧CLIENT的数据进行清除
- 同时广播WORKER最新CLIENT列表

MASTER崩溃

CLIENT不感知MASTER崩溃
后续创建ROOM/ENDPOINT则自动超时回调失败

WORKER崩溃

SFUCLIENT所记录会议资源信息都有归属于某个WORKER
SFUMASTER定时推送所有有效WORKER列表，广播给所有CLIENT
CLIENT检测到失效的WORKER将自动清除对应缓存，并回调通知上层

SFUCLIENT快速可用策略

每次操作，不判断是否已经注册上MASTER
都发给SFUMASTER。

接口错误码

```
namespace SFU
{
    enum EMErrorCode
    {
        EmError_Success, //成功
        EmError_Internal, //内部错误
    }
}
```