# 新架构ivr交互查询

## 1 需求描述

新架构ivr交互查询兼容老架构98,99查询被叫协议，并在此基础上新增了三种协议，分别是95,96,97。

通用的修改需求：

1. 新架构企业，通过ccgeid获取企业信息
2. 新架构企业，通过CR-Web提供的查询响应接口返回数据
3. 新架构企业，确定用户请求和用户响应格式

### 1.1 95(交互收键模式)

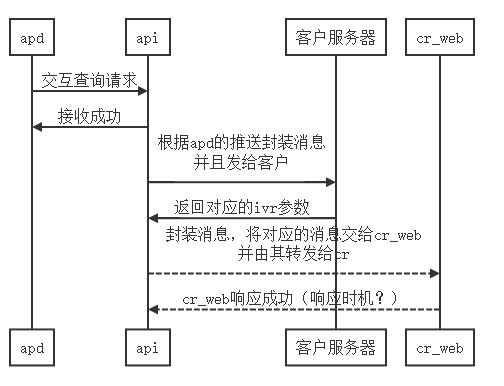
### 1.2 96(被叫查询模式)

### 1.3 97(通用交互模式)

### 1.4 98(AMNB或者AMXNB模式)

### 1.5 99(AMB或者AMXB模式)

## 2 交互流程



## 3 消息定义

### 3.1 apd->api

#### 3.1.1 95(交互收键模式)

{  
 "type" : 95,  
 "ccgeid" : 1576,  
 "callId" : "apixxxxxxxxxxxx",  
 "ccNumber" : "13260278209conf0\_1519439781636",  
 "callType" : 0,  
 "ivrFlowId" : 245,  
 "ivrQueryId" : 1,  
 "caller" : "13260278209",  
 "callerType" : 1,  
 "switchNumber" : "02566699794",  
 "called" : "1001",  
 "calledType" : 2,  
 "timestamp" : "1519439787",  
 "userQueryId" : "id\_0000001",  
 "inputKeys" : "1000",  
 "variables" : [  
 {"id\_number" : "110108198703127621" },  
 {:"name" :"" },  
 {"address":"" }  
 ]  
}

#### 3.1.2 96(被叫查询模式)

{  
 "type" : 96,  
 "ccgeid" : 1576,  
 "callId" : "apixxxxxxxxxxxx",  
 "ccNumber" : "13260278209conf0\_1519439781636",  
 "callType" : 0,  
 "ivrFlowId" : 245,  
 "ivrQueryId" : 1,  
 "caller" : "13260278209",  
 "callerType" : 1,  
 "switchNumber" : "02566699794",  
 "called" : "1001",  
 "calledType" : 2,  
 "timestamp" : "1519439787",  
 "userQueryId" : "id\_0000001",  
 "inputKeys" : "1000",  
 "variables" : [  
 {"id\_number" : "110108198703127621" },  
 {:"name" :"" },  
 {"address":"" }  
 ]  
}

#### 3.1.3 97(通用交互模式)

{  
 "type" : 97,  
 "ccgeid" : 1576,  
 "callId" : "apixxxxxxxxxxxx",  
 "ccNumber" : "13260278209conf0\_1519439781636",  
 "callType" : 0,  
 "ivrFlowId" : 245,  
 "ivrQueryId" : 1,  
 "caller" : "13260278209",  
 "callerType" : 1,  
 "switchNumber" : "02566699794",  
 "called" : "1001",  
 "calledType" : 2,  
 "timestamp" : "1519439787",  
 "userQueryId" : "id\_0000001",  
 "inputKeys" : "1000",  
 "variables" : [  
 {"id\_number" : "110108198703127621" },  
 {:"name" :"" },  
 {"address":"" }  
 ]  
}

#### 3.1.4 98(AMNB或者AMXNB模式)

{  
 "type" : 98,  
 "ccgeid" : 1576,  
 "ccNumber" : "13260278209conf0\_1519439781636",  
 "ivrFlowId" : 34,  
 "ivrQueryId" : 35,  
 "switchNumber" : "02566699794",  
 "useNumber" : "02566699794",  
 "caller" : "13260278209",  
 "callerType" : 1,  
 "timestamp" : "1519439787",  
 "path" : "1000"  
}

#### 3.1.5 99(AMB或者AMXB模式)

{  
 "type" : 98,  
 "ccgeid" : 1576,  
 "ccNumber" : "13260278209conf0\_1519439781636",  
 "ivrFlowId" : 34,  
 "ivrQueryId" : 35,  
 "switchNumber" : "02566699794",  
 "useNumber" : "02566699794",  
 "caller" : "13260278209",  
 "callerType" : 1,  
 "timestamp" : "1519439787",  
 "path" : "1000"  
}

### 3.2 api内部消息转换

根据前面的[需求描述](#header-n4)

1. 我们需要在某些结构体中新增ccgeid字段

新增ccgeid这个参数的意图应该是利用ccgeid代替enterpriseId和provinceId做hash运算确定处理线程，因为provinceId字段的获取需要查找数据库。但是事实是，入库日志数据库时目前enterpriseId和provinceId是必须要填写的。这个操作在apache模块进行，同样的还有通知给回查模块的消息。可能需要在日志数据库中新增一个索引字段ccgeid，或者调整这些操作的位置。

然后我们就可以定义结构体信息：

//日志数据库内容，利用ccgeid代替enterpriseId和provinceId入库
  
typedef struct \_emicalldev\_db\_callpushpost\_data\_
  
{
  
 unsigned long id;
  
 BOOL isCommonEnterprise;
  
 unsigned long app\_id;
  
 unsigned long provinceId;
  
 unsigned long enterpriseId;
  
 unsigned long ccgeid; //新架构新增ccgeid字段，用以标识企业
  
 unsigned long callLogId;
  
 int type;
  
 unsigned char callId[CALL\_RECORD\_CALLID\_MAX\_LEN];
  
 unsigned char switchNumber[PHONE\_NUMBER\_MAX\_LEN];
  
 DB\_CALLPUSH\_POSTDATA\_STATUS status;
  
 unsigned char post\_data[HTTP\_CONTENT\_BUFFER\_SIZE];
  
 unsigned long createTime;
  
 unsigned long updateTime;
  
} EMICALLDEV\_DB\_CALLPUSH\_POST\_DATA;

//推送内容，利用ccgeid代替enterpriseId和provinceId查询企业的详细信息
  
typedef struct \_\_push\_post\_data\_\_
  
{
  
 unsigned char callId[CALL\_RECORD\_CALLID\_MAX\_LEN];
  
 unsigned char ccNumber[CALL\_RECORD\_CALLID\_MAX\_LEN];
  
 unsigned char caller[PHONE\_NUMBER\_MAX\_LEN];
  
 unsigned char called[PHONE\_NUMBER\_MAX\_LEN];
  
 int xferTimes;
  
 BOOL extCaller;
  
 BOOL extCalled;
  
 BOOL isCaller;
  
 DB\_CALL\_RECORD\_CALL\_TYPE type;
  
 DB\_CALL\_RECORD\_STATUS status;
  
 unsigned char useNumber[PHONE\_NUMBER\_MAX\_LEN];
  
 unsigned char switchNumber[PHONE\_NUMBER\_MAX\_LEN];
  
 unsigned char subNumber[PHONE\_NUMBER\_MAX\_LEN];
  
 unsigned char virtNumber[PHONE\_NUMBER\_MAX\_LEN];
  
 unsigned long enterpriseId;
  
 unsigned long ccgeid; //新架构新增ccgeid字段，用以标识企业
  
 unsigned long ringTime;
  
 unsigned long startTime;
  
 unsigned long endTime;
  
 unsigned long timestamp;
  
 unsigned long duration;
  
 unsigned long reason;
  
 unsigned long gid;
  
 unsigned long pbxCallLogId;
  
 char feedback[CALL\_RECORD\_FEEDBACK\_MAX\_LEN];
  
   
 BOOL realtimeData;
  
 BOOL isCheckData;
  
 BOOL isCommonEnterprise;
  
 BOOL hangup2calling;
  
 int index;
  
 int lwpid;
  
 int mes\_type;
  
 int failed\_delay\_time;
  
 unsigned char path[CALL\_RECORD\_FEEDBACK\_MAX\_LEN]; //99协议 ，按键 (可能包含二级按键，比如:2-9)
  
   
 unsigned long app\_id;
  
 unsigned long provinceId;
  
 char number[USER\_NUMBER\_MAX\_LEN];
  
 char mobile[PHONE\_NUMBER\_MAX\_LEN];
  
 char destNumber[USER\_NUMBER\_MAX\_LEN];
  
 unsigned long ngnReason; //201708-N02细化通话失败和挂断原因
  
 char batchCallId[CALL\_RECORD\_CALLID\_MAX\_LEN];
  
 char batchCallUserData[USER\_DATA\_MAX\_LEN];
  
 char batchCallTaskId[BATCH\_TASK\_ID\_MAX\_LEN];
  
} ModCallPushPostData;

1. 新架构的通用ivr交互新增了部分通用字段，这些字段api必须转发给客户。需要新增一个结构体用来存储这些信息

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 通话类型 | IVR流程 | callType | caller | callerType | called | calledType |
| 呼入 | 正常流程 | 5 | 客户号码 | 1 | - | - |
| 呼入 | 子IVR流程 | 5 | 客户号码 | 1 | 坐席话机号 | 1/2 |
| 呼出 | 子IVR流程 | 0或7 | 坐席话机号 | 1/2 | 客户号码 | 1 |
| 语音通知 | 语音通知流程 | 3 | 总机号码 | 1 | 客户号码 | 1 |

typedef enum  
{  
 IVR\_CALLER\_AND\_CALLLED\_TYPE\_OUTLINE = 1,  
 IVR\_CALLER\_AND\_CALLLED\_TYPE\_INLINE = 2，  
}IVRCallerAndCalledType;  
  
typedef struct \_\_push\_post\_data\_ivr\_moudle\_\_  
{  
 IVRCallerAndCalledType callerType; //标记主叫是内线还是外线  
 IVRCallerAndCalledType calledType; //标记被叫是内线还是外线  
 DB\_CALL\_RECORD\_CALL\_TYPE calltype; //联合calltype callertype calledtype，可以得知在此次ivr请求中，哪个是客户，哪个是坐席  
 /\*  
 | 通话类型 | IVR流程 | callType | caller | callerType | called | calledType |  
 | -------- | ------------ | -------- | ---------- | ---------- | ---------- | ---------- |  
 | 呼入 | 正常流程 | 5 | 客户号码 | 1 | - | - |  
 | 呼入 | 子IVR流程 | 5 | 客户号码 | 1 | 坐席话机号 | 1/2 |  
 | 呼出 | 子IVR流程 | 0或7 | 坐席话机号 | 1/2 | 客户号码 | 1 |  
 | 语音通知 | 语音通知流程 | 3 | 总机号码 | 1 | 客户号码 | 1 |  
 \*/  
 unsigned long ivrFlowId; //ivr流程id  
 unsigned long ivrQueryId; //ivr查询节点id  
 char userQueryId[64]; //用于向客户服务器确定id对应下一步的操作是什么  
 char inputKeys[64]; //推送上次输入的按键  
 char \*variables;  
 /\*  
 用户自定义查询请求变量值集合，是{"variables" : [ {"id\_number" : "110108198703127621"}, {:"name" :""}, {"address":""} ]}，  
 存入对空间中，用完释放  
 \*/  
}ModCallPushPostDataIVRMoudle;  
  
typedef struct \_\_push\_post\_data\_\_  
{  
 unsigned char callId[CALL\_RECORD\_CALLID\_MAX\_LEN];  
 unsigned char ccNumber[CALL\_RECORD\_CALLID\_MAX\_LEN];  
 unsigned char caller[PHONE\_NUMBER\_MAX\_LEN];  
 unsigned char called[PHONE\_NUMBER\_MAX\_LEN];  
 int xferTimes;  
 BOOL extCaller;  
 BOOL extCalled;  
 BOOL isCaller;  
 DB\_CALL\_RECORD\_CALL\_TYPE type;  
 DB\_CALL\_RECORD\_STATUS status;  
 unsigned char useNumber[PHONE\_NUMBER\_MAX\_LEN];  
 unsigned char switchNumber[PHONE\_NUMBER\_MAX\_LEN];  
 unsigned char subNumber[PHONE\_NUMBER\_MAX\_LEN];  
 unsigned char virtNumber[PHONE\_NUMBER\_MAX\_LEN];  
 unsigned long enterpriseId;  
 unsigned long ccgeid;  
 unsigned long ringTime;  
 unsigned long startTime;  
 unsigned long endTime;  
 unsigned long timestamp;  
 unsigned long duration;  
 unsigned long reason;  
 unsigned long gid;  
 unsigned long pbxCallLogId;  
 char feedback[CALL\_RECORD\_FEEDBACK\_MAX\_LEN];  
 ModCallPushPostDataIVRMoudle \*ivr\_argv; //新架构通用ivr扩展字段  
  
 BOOL realtimeData;  
 BOOL isCheckData;  
 BOOL isCommonEnterprise;  
 BOOL hangup2calling;  
 int index;  
 int lwpid;  
 int mes\_type;  
 int failed\_delay\_time;  
 unsigned char path[CALL\_RECORD\_FEEDBACK\_MAX\_LEN]; //99协议 ，按键 (可能包含二级按键，比如:2-9)  
  
 unsigned long app\_id;  
 unsigned long provinceId;  
 char number[USER\_NUMBER\_MAX\_LEN];  
 char mobile[PHONE\_NUMBER\_MAX\_LEN];  
 char destNumber[USER\_NUMBER\_MAX\_LEN];  
 unsigned long ngnReason; //201708-N02细化通话失败和挂断原因  
 char batchCallId[CALL\_RECORD\_CALLID\_MAX\_LEN];  
 char batchCallUserData[USER\_DATA\_MAX\_LEN];  
 char batchCallTaskId[BATCH\_TASK\_ID\_MAX\_LEN];  
} ModCallPushPostData;

### 3.2 api->用户服务器

#### 3.2.1 请求客户

1. 99/98 99和98向客户的请求信息保持不变

{  
 'appId':'b23abb6d451346efa13370172d1921ef',  
 'callId':'api1234059445aDbbJxIdbT',  
 'accountSid':'c5dc4b87f33ef2ef37c8e974793ad8e5',  
 'caller':'18769874345',  
 'path':'1-2',  
 'callType':99,  
 'type':0,  
 'useNumber':'02566687987',  
 'switchNumber' : "02566699794",  
 'userData':FE87D3  
}

1. 95,96和97消息需要使用新架构新的消息格式

{  
 "type" : 95/96/97,  
 "ccgeid" : 1576,  
 "callId" : "apixxxxxxxxxxxx",  
 "ccNumber" : "13260278209conf0\_1519439781636",  
 "callType" : 0,  
 "ivrFlowId" : 245,  
 "ivrQueryId" : 1,  
 "caller" : "13260278209",  
 "callerType" : 1,  
 "switchNumber" : "02566699794",  
 "called" : "1001",  
 "calledType" : 2,  
 "timestamp" : "1519439787",  
 "userQueryId" : "id\_0000001",  
 "inputKeys" : "1000",  
 "variables" : [  
 {"id\_number" : "110108198703127621" },  
 {"name" :"" },  
 {"address":"" }  
 ]  
}  
说明：在上例中，有3个全局变量：id\_number、name和address，id\_number已经赋值，name和address未赋值，需要用户服务器返回，并在IVR其它节点中引用。

#### 3.2.2 客户响应

actionType

typedef enum  
{  
 0, invalid  
 1, 放音响应  
 2, 放音按键响应  
 3, 转技能组响应  
 4, 转坐席响应  
 5, 转外线响应  
 6, 转其他IVR流程响应  
 7, 流程结束响应  
}IvrActionType;

##### 3.2.2.1 放音响应

json to cr

{  
 "rspCode" : 0,  
 “ccgeid” : “123”,  
 “ccNumber” : :21212”,  
 "userQueryId" : "id\_0000001",  
 "variables" : [  
 { "id\_number" : "110108198703127621" },  
 { "name" :"张三" },  
 { "address":"江苏省南京市江宁区" }  
 ]  
 "nextAction" : {  
 "action" : 1,  
 "paras" : {  
 "voiceId" : "播放语音文件id",  
 "voiceName" : "播放语音文件唯一名称",  
 "allowBreak" : "是否允许打断: 0-不允许 1-允许"  
 }  
 }  
 "reason" : "test",  
 "userdata" : "test"  
}

##### 3.2.2.2 放音收键响应

json to cr

{  
 "rspCode" : 0,  
 “ccgeid” : “123”,  
 “ccNumber” : :21212”,  
 "userQueryId" : "id\_0000001",  
 "variables" : [  
 { "id\_number" : "110108198703127621" },  
 { "name" :"张三" },  
 { "address":"江苏省南京市江宁区" }  
 ]  
 "nextAction" : {  
 "action" : 2,  
 "paras" : {  
 "voiceId" : "播放语音文件id",  
 "voiceName" : "播放语音文件唯一名称",  
 "allowBreak" : "是否允许打断: 0-不允许 1-允许",  
 "getKeyNumber" : "获取按键位数",  
 "getKeyTimeout" : "收键超时时间",  
 "endWithHashKey" : "是否以#号键结束, 0-不是, 1-是"  
 }  
 }  
 "reason" : "test",  
 "userdata" : "test"  
}

##### 3.2.2.3 转技能组响应

json to cr

{
  
 "rspCode" : 0,
  
 “ccgeid” : “123”,
  
 “ccNumber” : :21212”,
  
 "userQueryId" : "id\_0000001",
  
 "variables" : [
  
 { "id\_number" : "110108198703127621" },
  
 { "name" :"张三" },
  
 { "address":"江苏省南京市江宁区" }
  
 ]
  
 "nextAction" : {
  
 "action" : 3,
  
 "paras" : {
  
 "acdId" : "技能组id",
  
 "acdName" : "技能组名称",
  
 "useAcdValue" : "0-不使用技能组配置 1-使用技能组配置",
  
 "queueTime" : "排队超时时长",
  
 "switchTimes" : "坐席流转次数",
  
 "ringTimeout" : "坐席振铃超时时长",
  
 "customerMemory" : "0-不记忆 1-优先熟客记忆 2-强制熟客记忆"
  
 }
  
 }
  
 "reason" : "test",
  
 "userdata" : "test"
  
}

##### 3.2.2.4 转座席响应

json to cr

{  
 "rspCode" : 0,  
 “ccgeid” : “123”,  
 “ccNumber” : :21212”,  
 "userQueryId" : "id\_0000001",  
 "variables" : [  
 { "id\_number" : "110108198703127621" },  
 { "name" :"张三" },  
 { "address":"江苏省南京市江宁区" }  
 ]  
 "nextAction" : {  
 "action" : 4,  
 "paras" : {  
 "workNumber" : "1001,1002,1003",  
 "number" : "1001,1002,1003",  
 "queueTime" : "坐席忙时排队时长",  
 "ringTimeout" : "多坐席情况下，坐席振铃超时时长"  
 }  
 }  
 "reason" : "test",  
 "userdata" : "test"  
}

##### 3.2.2.5 转外线响应

json to cr

{  
 "rspCode" : 0,  
 “ccgeid” : “123”,  
 "ccNumber" : :"21212”,  
 "userQueryId" : "id\_0000001",  
 "variables" : [  
 { "id\_number" : "110108198703127621" },  
 { "name" :"张三" },  
 { "address":"江苏省南京市江宁区" }  
 ]  
 "nextAction" : {  
 "action" : 5,  
 "paras" : {  
 "called" : "外线被叫号码",  
 "outNumber" : "呼出总机号码"  
 }  
 }  
 "reason" : "test",  
 "userdata" : "test"  
}

##### 3.2.2.6 转其他IVR流程响应

json to cr

{  
 "rspCode" : 0,  
 “ccgeid” : “123”,  
 “ccNumber” : "21212”,  
 "userQueryId" : "id\_0000001",  
 "variables" : [  
 { "id\_number" : "110108198703127621" },  
 { "name" :"张三" },  
 { "address":"江苏省南京市江宁区" }  
 ]  
 "nextAction" : {  
 "action" : 6,  
 "paras" : {  
 "ivrFlowId" : "IVR流程id",  
 "ivrFlowName" : "IVR流程名称"  
 }  
 }  
 "reason" : "test",  
 "userdata" : "test"  
}

##### 3.2.2.7 流程结束响应

{  
 "rspCode" : 0,  
 “ccgeid” : “123”,  
 “ccNumber” : “21212”,  
 "userQueryId" : "id\_0000001",  
 "variables" : [  
 { "id\_number" : "110108198703127621" },  
 { "name" :"张三" },  
 { "address":"江苏省南京市江宁区" }  
 ]  
 "nextAction" : {  
 "action" : 7  
 }  
 "reason" : "test",  
 "userdata" : "test"  
}

### 3.3 api->cr\_web

api到cr*web，只需要将json数据外包装一个头，然后透传给cr*web就可以了

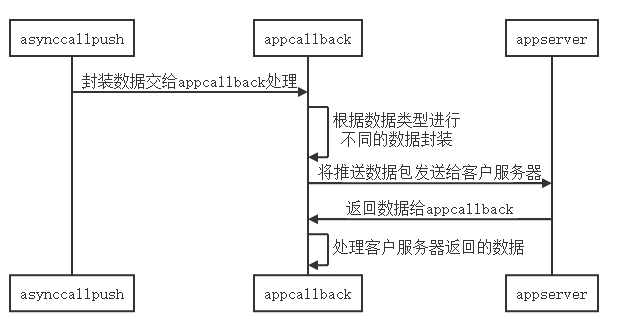
## 4 处理流程

cr给api的推送目前分为通话推送和坐席状态推送。

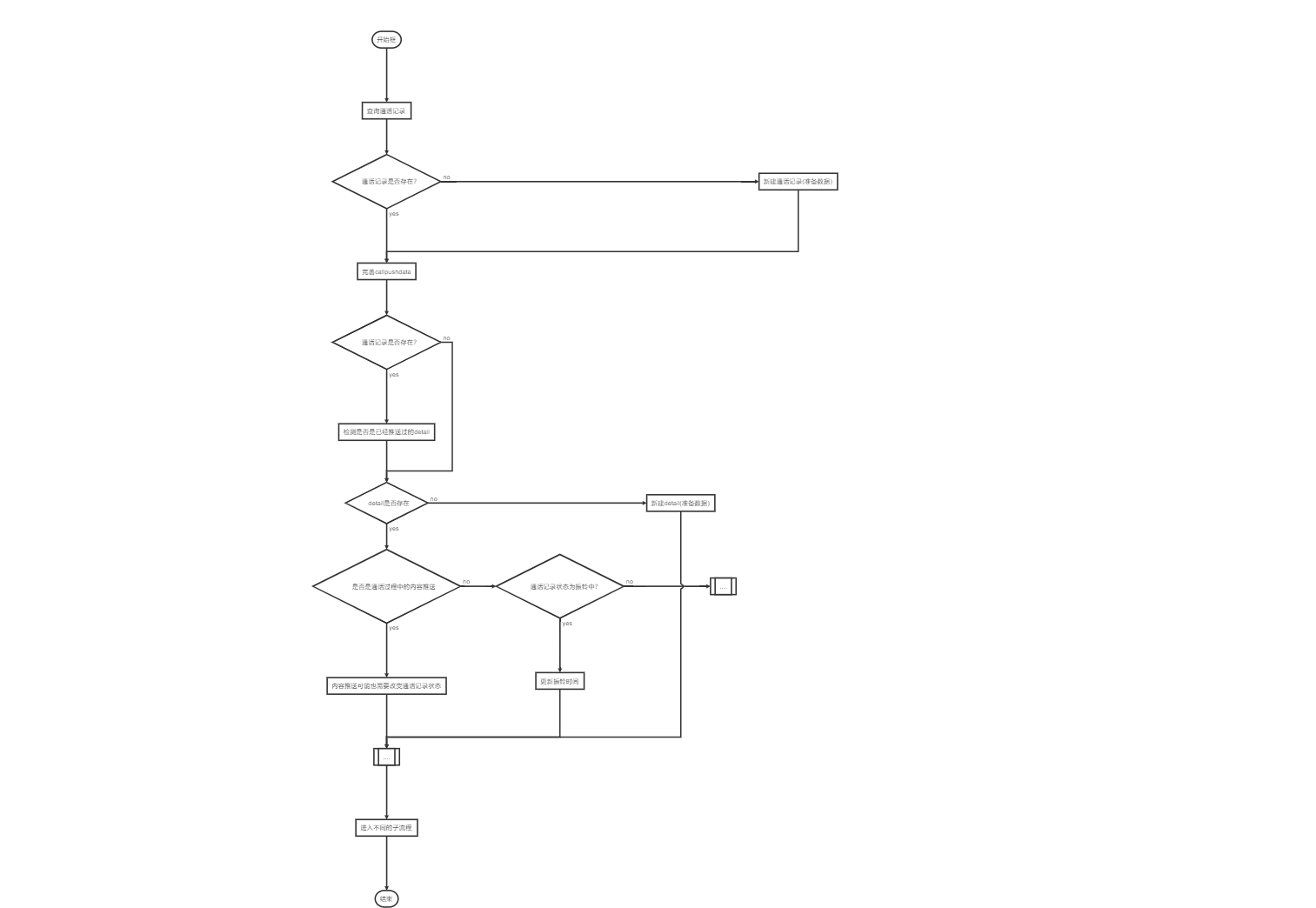
查询被叫默认按照通话推送进行处理，并且按照振铃推送的处理逻辑进行处理。

之前的查询被叫协议只会出现在呼入场景中，而现在的交互式ivr呼入呼出都有涉及，不仅仅只是查询被叫，同样的会有和通话无关的查询请求，首先考察是否会对通话记录产生影响

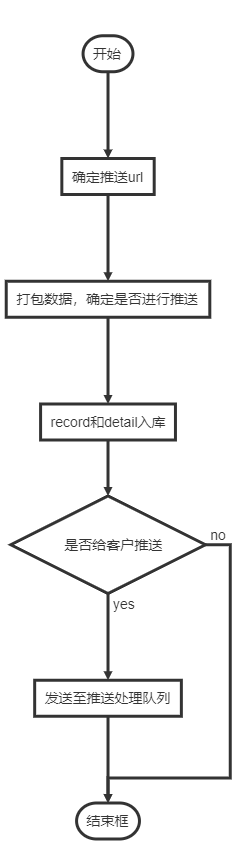
* api callpush模块时序图



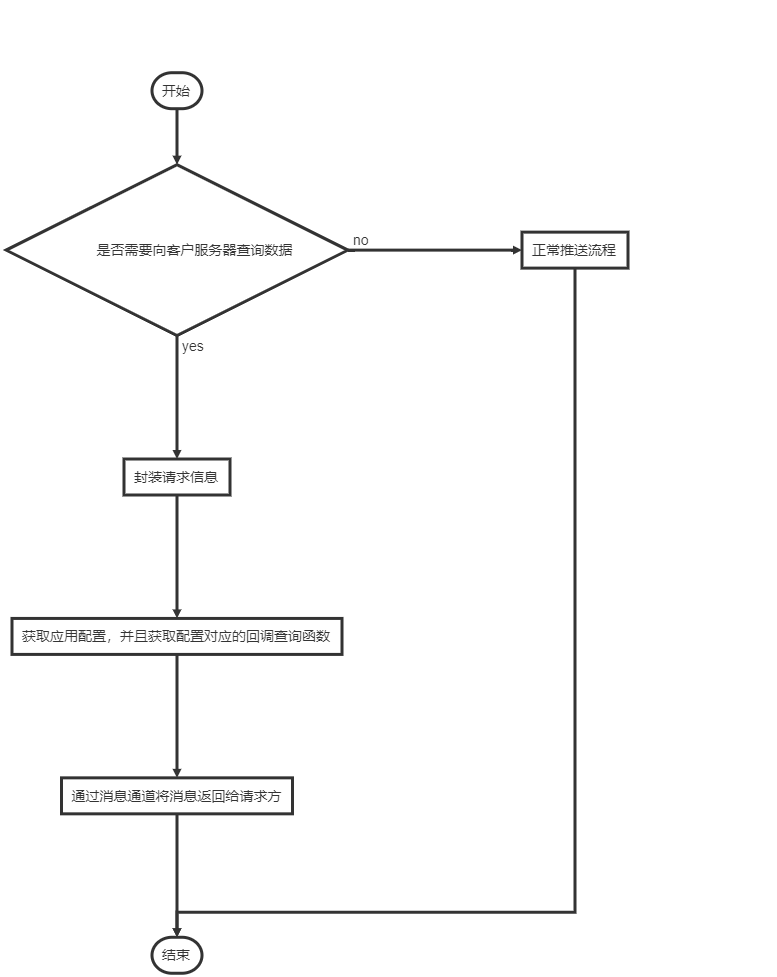
* asynccallpush流程图(status为0的分支)



* 通话振铃推送子流程图



* appcallback请求消息处理流程



经过考察我们主要的修改是：

1. 发向appcallback队列的数据会进行扩展，存储ivr扩展信息
2. 回调接口扩展，新建新的协议和对应的回调函数
3. 查询结果返回

### 4.1 95(交互收键模式)

### 4.2 96(被叫查询模式)

### 4.3 97(通用交互模式)

### 4.4 98(AMNB或者AMXNB模式)

被叫查询api会直接根据消息数据入库，和之前的处理方式保持一致即可。

### 4.5 99(AMB或者AMXB模式)

被叫查询api会直接根据消息数据入库，和之前的处理方式保持一致即可。

## 5 涉及代码

if( data->type == DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BASE ||  
 data->type == DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BY\_PBX ||  
 data->type == DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BY\_USENUMBER ||  
 data->type == DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BY\_VIRTNUMBER )  
{  
 strcpy(call\_record->useNumber, data->useNumber);  
}  
else if(data->type == DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BY\_SUBNUMBER)  
 strcpy(call\_record->subNumber, data->subNumber);  
else if(data->type == DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BY\_VIRTNUMBER)  
 strcpy(call\_record->virtNumber, data->virtNumber);

else if(IsOnlineCallPushType(data->type))  
{  
 if(call\_record->status < DB\_CALL\_RECORD\_STATUS\_CALL\_ESTABLISHED)  
 data->status = call\_record->status; //如果是查被叫的话重新改为振铃  
 else  
 data->status = DB\_CALL\_RECORD\_STATUS\_CALL\_ESTABLISHED;  
}

else if(NULL != callreqUrl ||  
 ( call\_record->type >= DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BASE &&  
 DB\_APPLICATION\_CALLBACK\_COMMON\_PROTOCOL == appInfo->protocol) )

if(call\_record->type >= DB\_CALL\_RECORD\_TYPE\_QUERY\_CALLED\_BASE)  
{