# 1 整体框架

主要以消息控制流程归纳助手内部各模块之间的控制与调用关系。

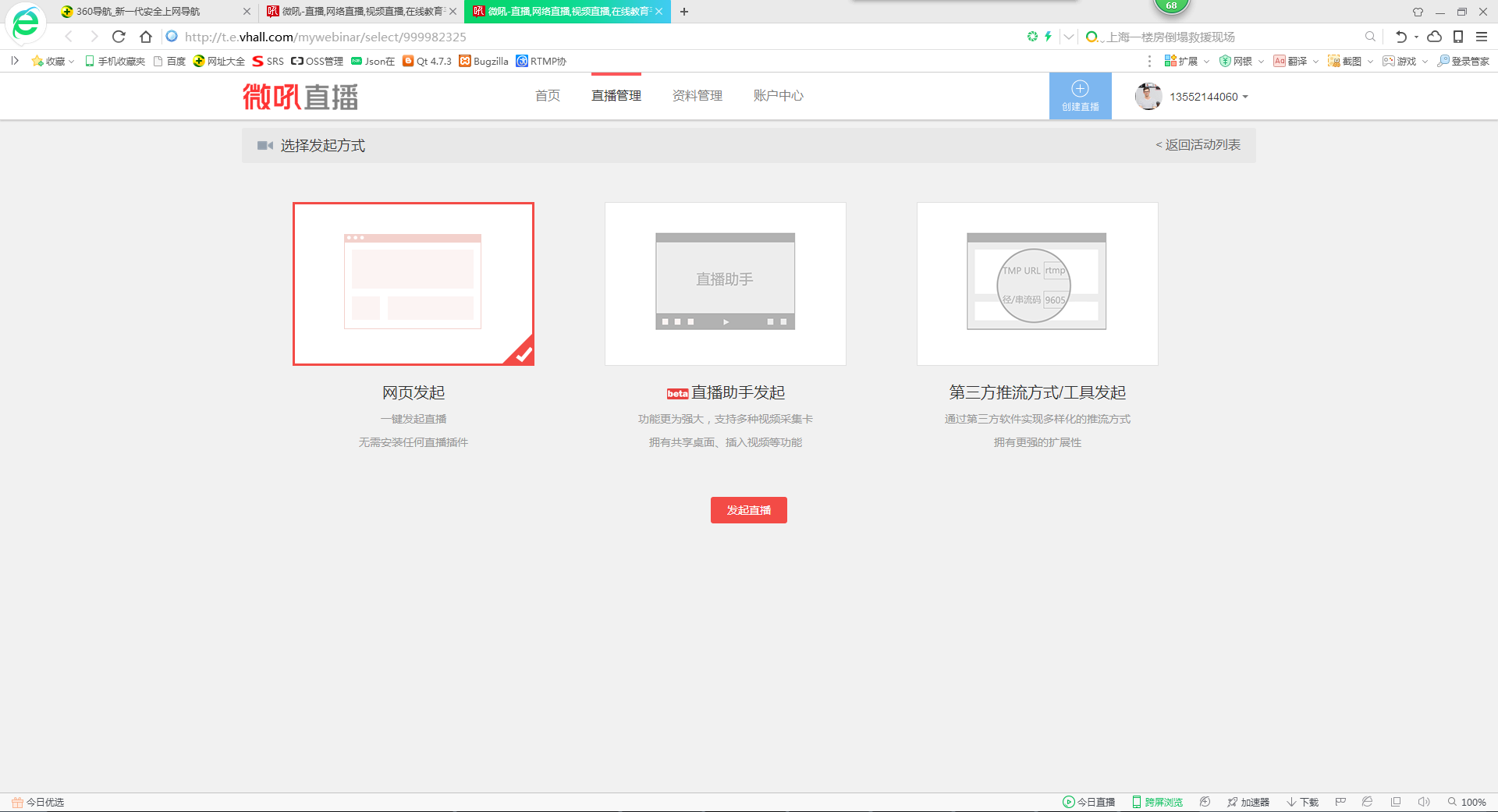


* CRuntimeInstance：控制应用程序的启动、初始化、停止；
* MessageDispatcher：PostMessageThread线程控制消息的分发；
* CommonToolKit：目前主要进行http请求处理，CheckMessageThread线程负责事件消息分发。
* MainUI：所有主界面控制逻辑在此完成，CRuntimeInstance::updateClock()负责事件消息分发；
* OBSControl：进行OBS推流控制、设备等处理，CheckMessageThread线程负责分发消息；
* VhallRightExtraWidget：聊天与成员列表功能实现，CRuntimeInstance::updateClock()负责事件消息分发；
* OBSWraped：包括采集、渲染、推流等模块；
* AudioEngine：音频采集线程、混音线程，不同音频数据设备实例化处理。
* VhallMediaCore：底层推流编码处理；
* InteractionSDK：socketio、websocket第三方库的集成；
* InterActivitySDK：腾讯云第三方SDK集成；
* DEF\_GET\_INTERFACE\_PTR(SingletonOBSControlIns, PID\_IMainUI, IID\_IMainUILogic, pMainUILogic);

# 2 VhallLive

1. CRuntimeInstance运行实力为单例模式对象，控制应用程序的启动、初始化控制与消息分发处理。
2. 检查启动模式（**flash/桌面/私有协议**）；
3. 初始实例：插件管理器初始化，消息派发事件注册等；
4. 加载并创建基础插件，注册控制消息；
5. 版本检查、与VhallService的连接；
6. 启动主界面。
7. 控制CRuntimeUIThreadManager定时分发消息。
8. CRuntimePlugManager控制插件对象的注册、创建、消息事件与对象的绑定操作。

## 启动方式比较



|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **桌面启动** | **网页启动**  **（flash）** | **直播助手发起**  **（私有协议）** |
| 启动流程 | C:\Users\vhall\AppData\Roaming\Tencent\Users\415657636\QQ\WinTemp\RichOle\FI6L0Q]B_}MP8A{JQ%S_Y)5.png | C:\Users\vhall\AppData\Roaming\Tencent\Users\415657636\QQ\WinTemp\RichOle\K[Y(ER@1K27)]UYJ4)P80RF.png |  |
| 交互方式 | 桌面双击快捷方式 | 通过flash与VhallService后台服务交互，VhallService通过向VhallLive传参调用启动。 | 私有协议的注册，是在助手目录下\install\VhallLive.iss，打包脚本打包时注册的。 |
| 启动参数 | E:\vhall\_vls\_develop\_hd\_2.2.0.1\Debug\VhallLive.exe | char \*targv[]={  "VhallLive",  "",//userName  "[{\"alias\":\"自动测试\",\"url\":\"rtmp://rs.live.t.vhou.net/vhall\"},{\"alias\":\"华北区测试\",\"url\":\"rtmp://rs.live.t.vhou.net/vhall\"},{\"alias\":\"华东区测试\",\"url\":\"rtmp://rs.live.t.vhou.net/vhall\"},{\"alias\":\"华南区测试\",\"url\":\"rtmp://rs.live.t.vhou.net/vhall\"}]",//streamUrls  "18ED71CDt245DB",//stremtoken  "516617411",//streamName  "0",//hideLogo  "0",//bMengzhu  "0",//channelId  "764128",//userid  "host",//role  "2",//webinar\_type 1普通活动 2切换嘉宾活动  "vhall",//accesstoken  "//mss.t.vhall.com:8090"  }; | vhalllive://vhalllive?**stream\_name**=100282948&**stream\_token**=...&**msg\_token**=...&hide\_logo=0 |
| 请求参数 | http://t.e.vhall.com/api/vhallassistant/auth/login?account=13552144060&password=0112qwaszx | \_\_\_\_\_\_ | http://t.e.vhall.com/ /api/client/v1/clientapi/ init/ ?token=……… |
| 响应参数  （主要部分） | "data":{"**plugins\_url**":"....",  "**list\_u**rl":"...",  "**token**":"...",  "**avatar**":"...",  "**nick\_name**":"13552144060"}} | \_\_\_\_\_\_ | "address": [  {  "name": "自动",  "srv": "rtmp://rs.live.t.vhou.net/vhall"  },  ….],  "user": {  "avatar": "//static01.t.vhall.com/static/img/head50.jpg",  "join\_uid": "157716",  "nick\_name": "18519139623",  "role": "host",  "user\_id": "612"} |
| 活动状态 | VHGetRet(url,8000,ba) 如果非flash方式启动的助手，需要调用后台接口，将活动置为开始状态，如果接口调用失败则再UI上提示网络异常，需要重新开始活动 | 不需要请求此接口，因为已经从网页端控制了活动状态 | 同桌面启动 |

## 版本更新

**GetURL:**

<http://e.vhall.com/api/client/v1/util/current-version>

**Response:**

{

"code":200,

"msg":"success",

"data":

{"version":"1.7.3.3",

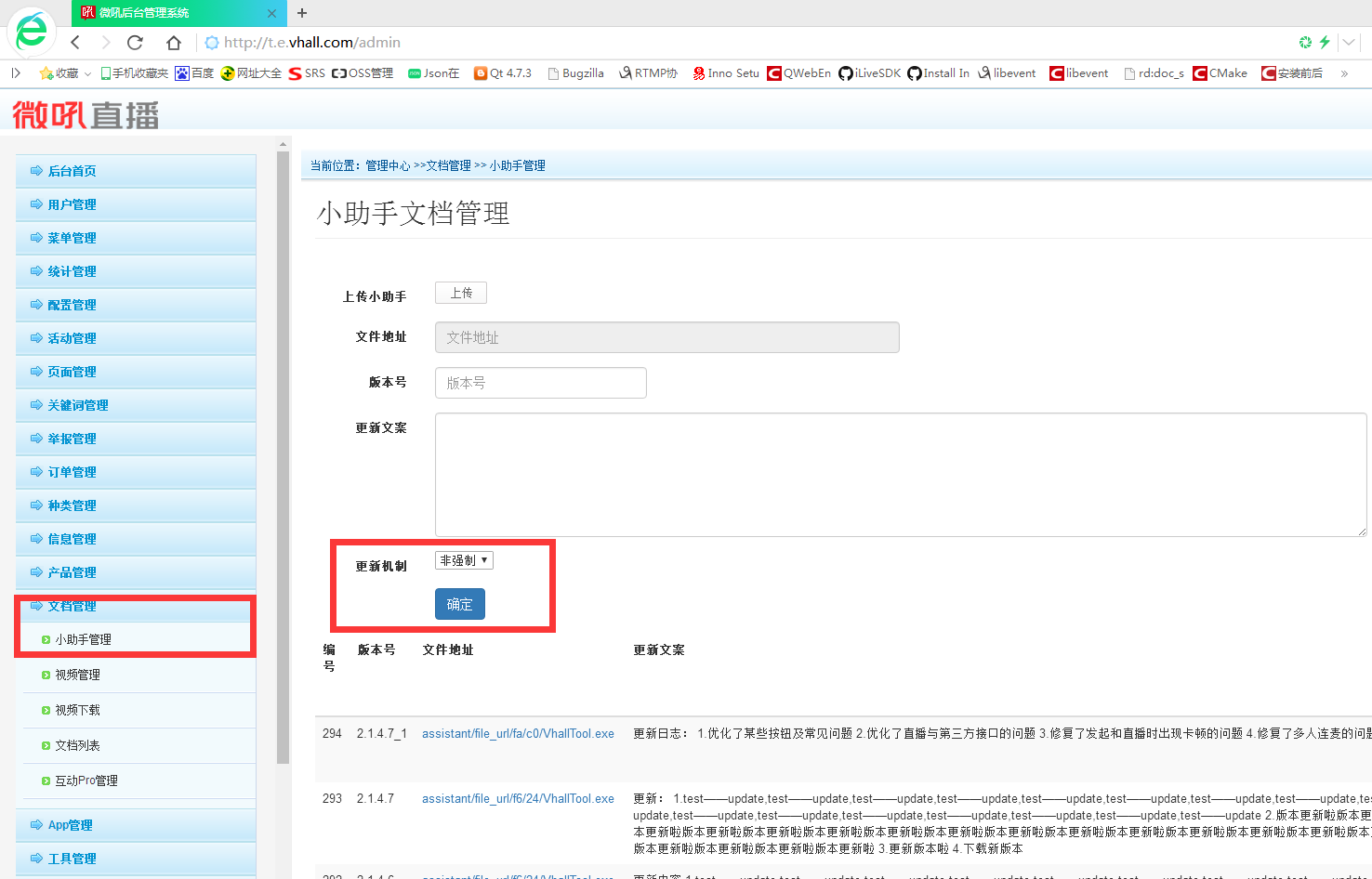
"download\_link":"\/\/cnstatic01.e.vhall.com\/upload\/assistant\/file\_url\/cd\/a9\/VhallTool.exe",

"description":"1\uff0c\u4fee\u590d\u4e86\u5728\u7ebf\u4eba\u6570\u4e0d\u51c6\u786e\u7684bug\uff1b\r\n2\uff0c\u5176\u4ed6\u5f71\u54cd\u4f7f\u7528\u4f53\u9a8c\u7684bug\uff1b",

“force\_upgrade”:”1”}

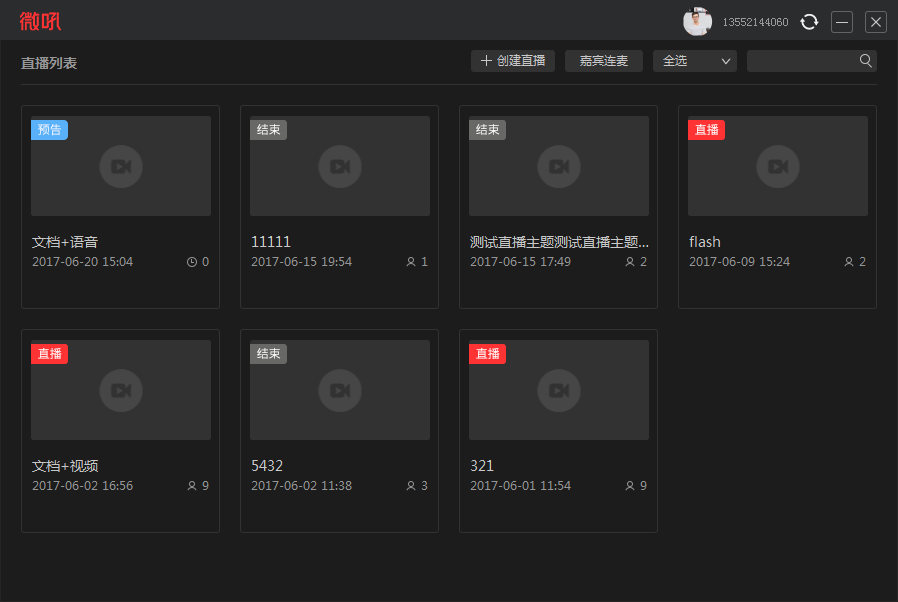
}

与本地安装版本比较，如果获取的版本大于运行版本，则提示更新。其中需要注意，如果force\_upgrade为1表示为强制升级。这个强制升级的配置如图所示，如果强制升级就把更新机制勾选成强制即可。

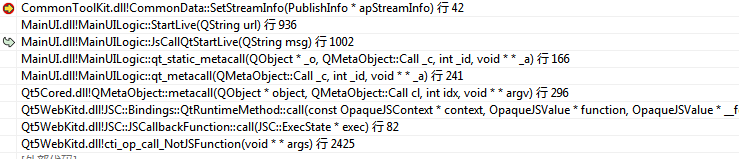


## 2.3 选取活动

桌面启动之后，根据返回的**“list\_url”**加载直播列表。选中某一个活动后，



点击开播，js回回调Qt返回当前活动的参数。



启动JS返回的参数为：vhalllive://vhalllive?stream\_name=999982325&stream\_token=….&msg\_token=….&hide\_logo=0

返回值：

{"code":200,

"msg":"success",

"data":

{"accesstoken":"vhall",

"ismix":1,

"switchDomain":"//t-mss.e.vhall.com/",

"user":{

"user\_id":"16417401",

"join\_uid":"922103",

"nick\_name":"13552144060",

"avatar":"//……/b5fa73691edecde3fa41be2a017a76c8.jpg",

"role":"host"

},

"address":[{"name":"\u81ea\u52a8\u6d4b\u8bd5","srv":"rtmp:\/\/rs.live.t.vhou.net\/vhall"},

………],

"forbidchat":0,

"filterurl":"http://t.e.vhall.com/webinar/chatfilter/211079534?vc=eyJpdiI6IjlJRWFMTEU2a3NlQzVhcDErY1wvbnFBPT0iLCJ2YWx1ZSI6IjZaQThCc2hLZVhPc2JJQjZYdlRCY01QMDJyYlBXMENVYzc1ZDRaZUYweDQ9IiwibWFjIjoiYmM1OGQ5M2QxODk4ZTM2NDAwMDMxNjE4MDE3YTJiYzg0OTczZDYwMzAxZWM5YzU5ZTI1NzNkMmE1OGU3Yzk1NyJ9 ",//

"msg\_srv":"t-msg01.e.vhall.com",//socketio

"msg\_port":"",

"chat\_srv":"t-chat01.e.vhall.com",//websockect recv msg

"chat\_port":"",

"chat\_url":"//t-api.e.vhall.com\/pub\/index",//send msg

"is\_webinar\_plug":1,

"webinar\_name":"\u6587\u6863+\u89c6\u9891",

"isGuestChange":"1",//0代表不可以使用互动功能，1可以使用互动功能。

"isPrivilege":"1", //0表示当前活动不是多嘉宾活动，1表示是多嘉宾活动。

"isVoice":"1", //0表示用户没有使用实时字幕功能，1表示支持实时字幕功能

"3rdstream\_url":"t-relay.e.vhall.com:5000\/api\/3rdstream\/",

"plugins\_url":"http://tzs.e.vhall.com\/plugins.html?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjE2NDE3NDAxIiwicGhvbmUiOiIxMzU1MjE0NDA2MCIsIm5pY2tfbmFtZSI6IjEzNTUyMTQ0MDYwIn0.sTcRjovPJ98T0zxo2pViVMGfBxjbQyTr6Blbapd98Rw&apiUrl=http:\/\/t.e.vhall.com"}

}



通过参数isGuestChange和isPrivilege如果都是1，则支持嘉宾连麦活动。

# 直播

## 3.1 插件DLL加载

vhallLive插件（dll动态库）采用的是显示的加载方式，即程序中需要哪些插件动态载入，关于插件加载说明如下：

### 3.1.1动态载入 DLL

动态载入方式是指在编译之前并不知道将会调用哪些 DLL 函数， 完全是在运行过程中根据需要决定应调用哪些函数。

方法是：用 LoadLibrary 函数加载动态链接库到内存，用 GetProcAddress函数动态获得 DLL 函数的入口地址。当一个 DLL 文件用 LoadLibrary 显式加载后，在任何时刻均可以通过调用 FreeLibrary 函数显式地从内存中把它给卸载。

动态调用使用的 Windows API 函数主要有 3 个， 分别是 LoadLibrary、 GetProcAddress 和FreeLibrary。

我们分别详细介绍这三个函数的功能，因为无论学习编程还是逆向这是三个函数都是非常常用滴。

1. **LoadLibrary 函数**

注：Delphi 中还提供了 SafeLoadLibrary 函数，它封装了 Loadlibrary 函数，可以装载由 Filename 参数指定的 WindowsDLL或 Linux 共享对象。它简化了DLL的装载并且使装载更加安全。  
  
[格式]：

function LoadLibrary(LibFileName : PChar): Thandle;

[功能]：加载由参数 LibFileName 指定的 DLL 文件。

[说明]：参数 LibFileName 指定了要装载的 DLL 文件名，如果 LibFileName 没有包含一个路径，系统将按照：当前目录、Windows 目录、Windows 系统目录、包含当前任务可执行文件的目录、列在 PATH 环境变量中的目录等顺序查找文件。如果函数操作成功，将返回装载 DLL 库模块的实例句柄，否则，将返回一个错误代码，错误代码的定义如下表所示。

|  |  |
| --- | --- |
| **错误代码** | **含义** |
| 0 | 系统内存不够，可执行文件被破坏或调用非法 |
| 2 | 文件没有被发现 |
| 3 | 路径没有被发现 |
| 5 | 企图动态链接一个任务错误或者有一个共享或网络保护错误 |
| 6 | 库需要为每个任务建立分离的数据段 |
| 8 | 没有足够的内存启动应用程序 |
| 10 | Windows  版本不正确 |
| 11 | 可执行文件非法或不是Windows  应用程序，或在.  EXE映像中有错误 |
| 12 | 应用程序为一个不同的操作系统设计（如  OS/2） |
| 13 | 应用程序为  MS  DOS   4. 0  设计 |
| 14 | 可执行文件的类型不知道 |
| 15 | 试图装载一个实模式应用程序（为早期Windows  版本设计） |
| 16 | 试图装载包含可写的多个数据段的可执行文件的第二个实例 |
| 19 | 试图装载一个压缩的可执行文件（文件必须被解压后才能被装载） |
| 20 | DLL  文件非法 |
| 21 | 应用程序需要  32  位扩展 |

假如在应用程序中用 LoadLibrary 函数装入某一个 DLL 前， 其他应用程序已把该 DLL 装入内存中了，则系统将不再装入该 DLL 的另一个实例，而是使该 DLL 的“引用计数”加 1 。

**（2）GetProcAddress 函数**

[格式]：

function GetProcAddress(Module:Thandle; ProcName:PChar): TfarProc;

[功能]：返回参数 Module 指定的模块中，由参数 ProcName 指定的过程或函数的入口地址。

[说明]：参数 Module 包含被调用函数的 DLL 句柄，这个值由 LoadLibrary 返回， ProcName

是指向含有函数名的以 nil 结尾的字符串指针，或者可以是函数的次序值，但大多数情况下，用函数名是一种更稳妥的选择。如果该函数执行成功，则返回 DLL 中由参数 ProcName 指定的过程或函数的入口地址，否则返回 nil 。

**（3）FreeLibrary 函数**  
[格式]：

procedure  FreeLibrary(Module: Thandle);

[说明]：将由参数 Module 指定的 DLL 文件从内存中卸载 1 次。

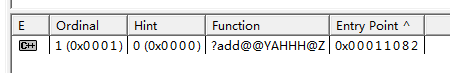
[说明]：Module 为 DLL 库的句柄。这个值由 LoadLibrary 返回。由于 DLL 在内存中只装载一次，因此调用 FreeLibrary 首先使 DLL 的引用计数减 1，如果计数减为 0 则卸载该 DLL。[注意]：每调用一次 LoadLibrary 函数就应调用一次 FreeLibrary 函数，以保证不会有多余的库模块在应用程序结束后仍留在内存中，否则导致内存泄漏。

### 3.1.2[使用def文件简化dll导出](http://www.cnblogs.com/TianFang/archive/2013/05/04/3059073.html)

在C++中，我们可以通过 [\_\_declspec(dllexport)](http://msdn2.microsoft.com/zh-cn/library/a90k134d(VS.80).aspx) 将函数导出为Dll中供其它程序使用，例如：

     \_declspec(dllexport) int add(int a, int b);

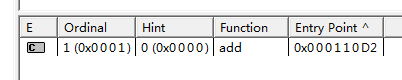
在这种方式下，如果调用该dll的是一个c++程序（同一个编译器的版本）是没有问题的。但是，如果调用该dll是一个其它语言的程序（如C#、VB），则会出错。究其原因，是因为在C++中存在函数的重载，允许函数重名，因此在编译器生成dll的时候，为了区别重名的程序，会将其进行一定算法进行[名称转换](http://www.cppblog.com/FateNo13/archive/2009/08/03/92052.html)。例如，对于前面的add函数，实际的函数名称是如下形式。



因此，我们直接通过函数名add是无法找到该函数的，从而导致调用失败。为了解决这一问题，我们往往在函数前面再加一个extern "C"，使用C方式的函数命名规则。

     extern "C" \_declspec(dllexport) int add(int a, int b);

这样函数的名称就成add了。



这样，我们就需要在每一个函数签名加上"extern "C" \_declspec(dllexport)"这一长串声明。如果需要导出的函数较多则显得非常繁琐，也非常难看。为了简化这一过程，MS引入了[def文件](http://msdn.microsoft.com/zh-cn/library/28d6s79h(v=vs.80).aspx)方便我们操作。

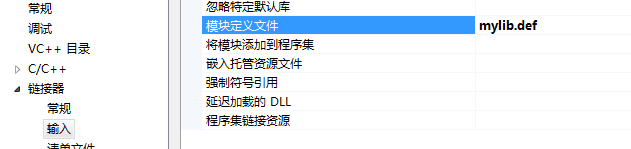
使用Def文件比较简单，只需要在项目中添加一个def文件，然后把我们要导出的函数放在def文件中即可。

http://images.cnitblog.com/blog/12132/201305/04111910-81f15098e7c44a179107366a41b5c12d.png

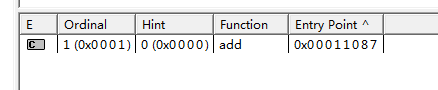
Def文件的简单示例如下：

    LIBRARY    EXPORTS        add

最后记得在链接器选项中选中使用的def文件（默认情况下，添加def文件时会自动加上该选项，无需手动更改）。



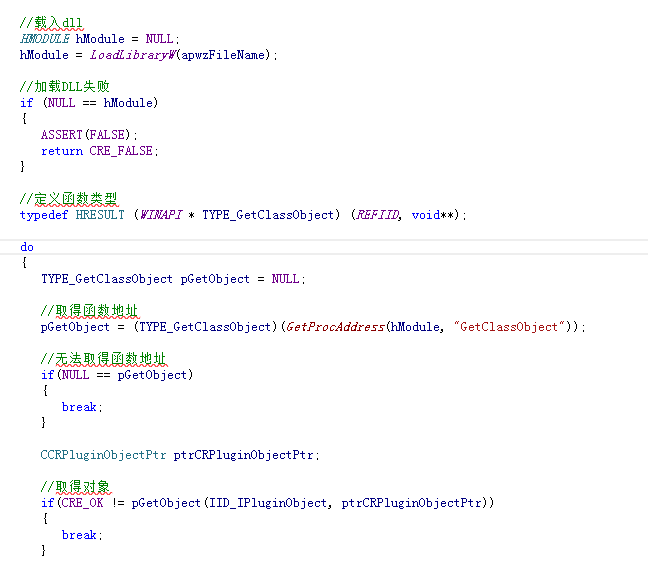
这样，我们的函数无需加那一堆前缀，仍然可以使用默认的int add(int a, int b);形式，但导出后的方式依然是C形式的函数定义。



最后指得一提的是，一般C/C++默认的调用方式是\_\_cdecl,这种方式下需要调用方对函数清栈。如果对外提供api共其它非C++程序使用时，调用方会无法清栈而出错（C#会直接报函数声明不匹配的错误）。因此，对外提供api时还应该将接口声明为\_\_stdcall，让api函数自己清栈。这也是Windows API前面都加上了一个WINAPI的宏的原因。

def文件还有许多其它的高级用法，要进一步了解的话，可以参看一下MS的官方文档：<http://msdn.microsoft.com/zh-cn/library/28d6s79h(v=vs.80).aspx>

### 3.1.3 vHallLive加载插件



//载入dll

*HMODULE* hModule = NULL;

hModule = *LoadLibraryW*(apwzFileName);

//取得函数地址,为了获取插件单例对象。

pGetObject = (TYPE\_GetClassObject)(*GetProcAddress*(hModule, "GetClassObject"));

各个插件在模块定义文件中对GetClassObject进行了导出。

mainui模块的GetClassObject定义:

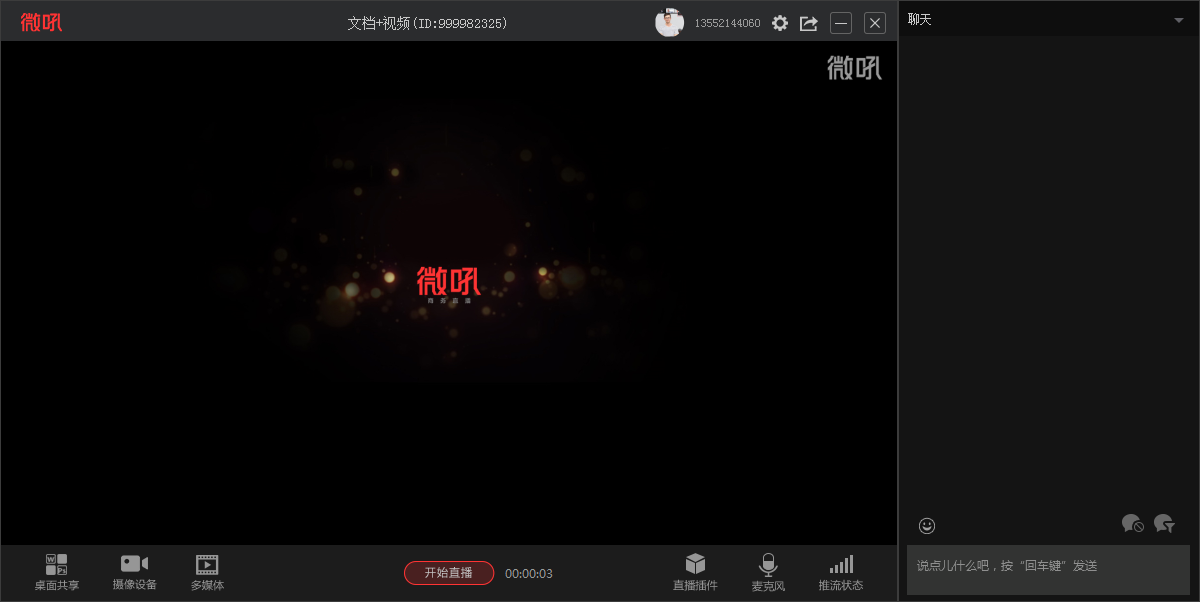
//模块导出函数

extern "C" HRESULT *WINAPI* GetClassObject(*REFIID* riid, void \*\* appvObj) {

return SingletonOBSControlIns::Instance().QueryInterface(riid, appvObj);

}

之后保存插件的各种信息到CPluginManager，与插件管理器进行关联，并通知加载了插件。



## 3.2场景源添加

MainUI界面控制不同场景的添加，最终调用均是AddNewSource接口。



## 3.3多媒体插入



MediaVideoSource实现了插播场景的实现，内部使用了vlc开源库。



插播视频的处理通过vlc实现，封装在MediaReader工程中。



在开始播放中添加了回调。

//音频回调

*libvlc\_audio\_set\_format\_callbacks*(mp, audioSetupCallbackProxy, audioCleanupCallbackProxy);

*libvlc\_audio\_set\_callbacks*(mp, audioPlayCallbackProxy, nullptr, nullptr, nullptr, nullptr, this);

//视频回调

*libvlc\_video\_set\_callbacks*(mp, videoLock, videoUnlock, display, this);

*libvlc\_video\_set\_format\_callbacks*(mp, VideoSetupFormatCallback, VideoSetupFormatCleanup);

*libvlc\_media\_player\_play*(mp);

mMediaPlayer = mp;

## 3.4视频采集

目前渲染线程主要渲染内容包括BitmapImageSource（图片）、MediaVideoSource（插播视频）、TextOutputSource（文字）、DeviceSource（摄像头）、DesktopImageSource （桌面共享）、DeckLinkVideoSource（外接输入）等场景源。渲染部分均在OBS:: MainCaptureLoop线程中实现。如图展示了DeviceSource渲染流程。所有场景源都保存在List<SceneItem\*> Scene::sceneItems中。



## 3.5音频采集

助手的音频输入包括插播视频音量、麦克风音量、扬声器音量，如果描述了麦克风采样的过程。



### 3.5.1 Core Audio音频采集

    首先我们来了解下面几个概念：

    1. IMMDevice ： 创建音频设备终端，我们可以把它简单的理解为设备对象

    2. IAudioClient ： 创建一个用来管理音频数据流的对象，应用程序通过这个对象可以获取的音频设备里的数据，我们可以把它想象成一个大水池，里面都是一些数据

    3.  IAudioCaptureClient ： 很明显，专用于获取采集数据的对象，它还有个兄弟IAudioRenderClient

 下面直接上官网示例代码，解释下代码大家就清楚怎么做了

//首先枚举你的音频设备，你可以在这个时候获取到你机器上所有可用的设备，并指定你需要用到的那个设置

hr = CoCreateInstance(CLSID\_MMDeviceEnumerator, NULL,

CLSCTX\_ALL, IID\_IMMDeviceEnumerator,

(void\*\*)&pEnumerator);

EXIT\_ON\_ERROR(hr)

hr = pEnumerator->GetDefaultAudioEndpoint(eCapture, eConsole, &pDevice);

EXIT\_ON\_ERROR(hr)

// 创建一个管理对象，通过它可以获取到你需要的一切数据

hr = pDevice->Activate(IID\_IAudioClient, CLSCTX\_ALL,NULL, (void\*\*)&pAudioClient);

EXIT\_ON\_ERROR(hr)

hr = pAudioClient->GetMixFormat(&pwfx);

EXIT\_ON\_ERROR(hr)

//初始化管理对象，在这里，你可以指定它的最大缓冲区长度，这个很重要，应用程序控制数据块的大小以及延时长短都靠这里的初始化，具体参数大家看看文档解释

hr = pAudioClient->Initialize(AUDCLNT\_SHAREMODE\_SHARED,

0,

hnsRequestedDuration,

0,

pwfx,

NULL);

EXIT\_ON\_ERROR(hr)

//这个buffersize，指的是缓冲区最多可以存放多少帧的数据量

hr = pAudioClient->GetBufferSize(&bufferFrameCount);

EXIT\_ON\_ERROR(hr)

//创建采集管理接口，这个接口很简单，没什么重要的东西

hr = pAudioClient->GetService(IID\_IAudioCaptureClient, (void\*\*)&pCaptureClient);

EXIT\_ON\_ERROR(hr)

// Notify the audio sink which format to use.

hr = pMySink->SetFormat(pwfx);

EXIT\_ON\_ERROR(hr)

// Calculate the actual duration of the allocated buffer.

音频数据量的计算：数据量（字节/秒）= (采样频率（Hz）\*采样位数（bit）\*声道数)/ 8

hnsActualDuration = (double)REFTIMES\_PER\_SEC \* bufferFrameCount / pwfx->nSamplesPerSec;

hr = pAudioClient->Start(); // Start recording.

EXIT\_ON\_ERROR(hr)

// Each loop fills about half of the shared buffer.

while (bDone == FALSE)

{

//让程序暂停运行一段时间，缓冲区里在这段时间会被填充数据

Sleep(hnsActualDuration/REFTIMES\_PER\_MILLISEC/2);

hr = pCaptureClient->GetNextPacketSize(&packetLength);

EXIT\_ON\_ERROR(hr)

while (packetLength != 0)

{

//锁定缓冲区，获取数据

hr = pCaptureClient->GetBuffer(&pData,&numFramesAvailable,&flags, NULL, NULL);

EXIT\_ON\_ERROR(hr)

if (flags & AUDCLNT\_BUFFERFLAGS\_SILENT){

pData = NULL;

}

hr = pMySink->CopyData(pData, numFramesAvailable, &bDone);

EXIT\_ON\_ERROR(hr)

hr = pCaptureClient->ReleaseBuffer(numFramesAvailable);

EXIT\_ON\_ERROR(hr)

hr = pCaptureClient->GetNextPacketSize(&packetLength);

EXIT\_ON\_ERROR(hr)

}

}

hr = pAudioClient->Stop();

EXIT\_ON\_ERROR(hr)

## 开始/停止推流业务

开始、结束直播时需要上报开始、结束状态。

* 开始推流

通过http请求，上报推流状态，token是在点击开播时返回的url中携带的stream\_token。

[http://t.e.vhall.com/api/client/v1/clientapi/**startwebinar**?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjkyMjEwMyIsIm5hbWUiOiIxMzU1MjE0NDA2MCIsInJvb20iOiI5OTk5ODIzMjUiLCJyb2xlIjoiaG9zdCIsInVpZCI6IjE2NDE3NDAxIiwidHJhY2tJZCI6IjEwMjM3MDUifQ.olDyqWmN1f82sSIYmt2ZyjcBt1iDiwx0oUCzGjv\_xng](http://t.e.vhall.com/api/client/v1/clientapi/startwebinar?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjkyMjEwMyIsIm5hbWUiOiIxMzU1MjE0NDA2MCIsInJvb20iOiI5OTk5ODIzMjUiLCJyb2xlIjoiaG9zdCIsInVpZCI6IjE2NDE3NDAxIiwidHJhY2tJZCI6IjEwMjM3MDUifQ.olDyqWmN1f82sSIYmt2ZyjcBt1iDiwx0oUCzGjv_xng)

* 停止推流

[http://t.e.vhall.com/api/client/v1/clientapi/**stopwebinar**?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjkyMjEwMyIsIm5hbWUiOiIxMzU1MjE0NDA2MCIsInJvb20iOiI5OTk5ODIzMjUiLCJyb2xlIjoiaG9zdCIsInVpZCI6IjE2NDE3NDAxIiwidHJhY2tJZCI6IjEwMjM3MDUifQ.olDyqWmN1f82sSIYmt2ZyjcBt1iDiwx0oUCzGjv\_xng](http://t.e.vhall.com/api/client/v1/clientapi/stopwebinar?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjkyMjEwMyIsIm5hbWUiOiIxMzU1MjE0NDA2MCIsInJvb20iOiI5OTk5ODIzMjUiLCJyb2xlIjoiaG9zdCIsInVpZCI6IjE2NDE3NDAxIiwidHJhY2tJZCI6IjEwMjM3MDUifQ.olDyqWmN1f82sSIYmt2ZyjcBt1iDiwx0oUCzGjv_xng)

OBSControl模块负责处理开始和停止推流在OBSControlLogic中的DealStartPublish(void\* apData, DWORD adwLen)接口处理。控制消息是MSG\_OBSCONTROL\_PUBLISH消息。在此方法内开始推流调用StartStream方法，停止推流调用StopStream方法。

MainUILogic::PublishStream()

VhallRightExtraWidgetLogic::StartWebnair()；

VhallRightExtraWidgetLogic::StopWebnair()；

## 数据源控制



对于场景源控制类型定义在包括：

#define OP\_TYPE\_FULL\_SCREEN 0 全屏  
#define OP\_TYPE\_MOVE\_TOP 1 置顶  
#define OP\_TYPE\_MOVE\_BOTTOM 2 置底  
#define OP\_TYPE\_MODIFY 3 修改  
#define OP\_TYPE\_DELETE 4 删除

主窗口定义在MainUI的RenderWnd.h中。鼠标如果在渲染窗口中可以进行拖拽等处理。

比如移动处理在virtual void mouseMoveEvent(QMouseEvent \*mouseEvent);中，

实现为：

void RenderWnd::mouseMoveEvent(QMouseEvent \*mouseEvent) {

*POINTS* pos = { mouseEvent->x(), mouseEvent->y() };

if (NULL != m\_pOBSControlLogic) {

**m\_pOBSControlLogic->OnMouseEvent(*WM\_MOUSEMOVE*, pos);**

}

QWidget::mouseMoveEvent(mouseEvent);

}

## 聊天列表与在线列表

目前助手消息页面嵌入的是一个QWebEngineView，创建聊天窗体时需向JS进行注册主窗体。其中具体的注册方法是为QWebEngineView创建一个QWebChannel。

m\_pWebChannel->registerObject("MainWindow", this);

m\_pWebEngineView->page()->setWebChannel(m\_pWebChannel);

m\_pWebEngineView->page()->load(QUrl("qrc:/chat/chat\_html/chat.html"));

* 发送消息：

发送消息采用http协议。通过HttpReqWork从队列中轮询的方式读取发送消息任务。

* 接收消息：

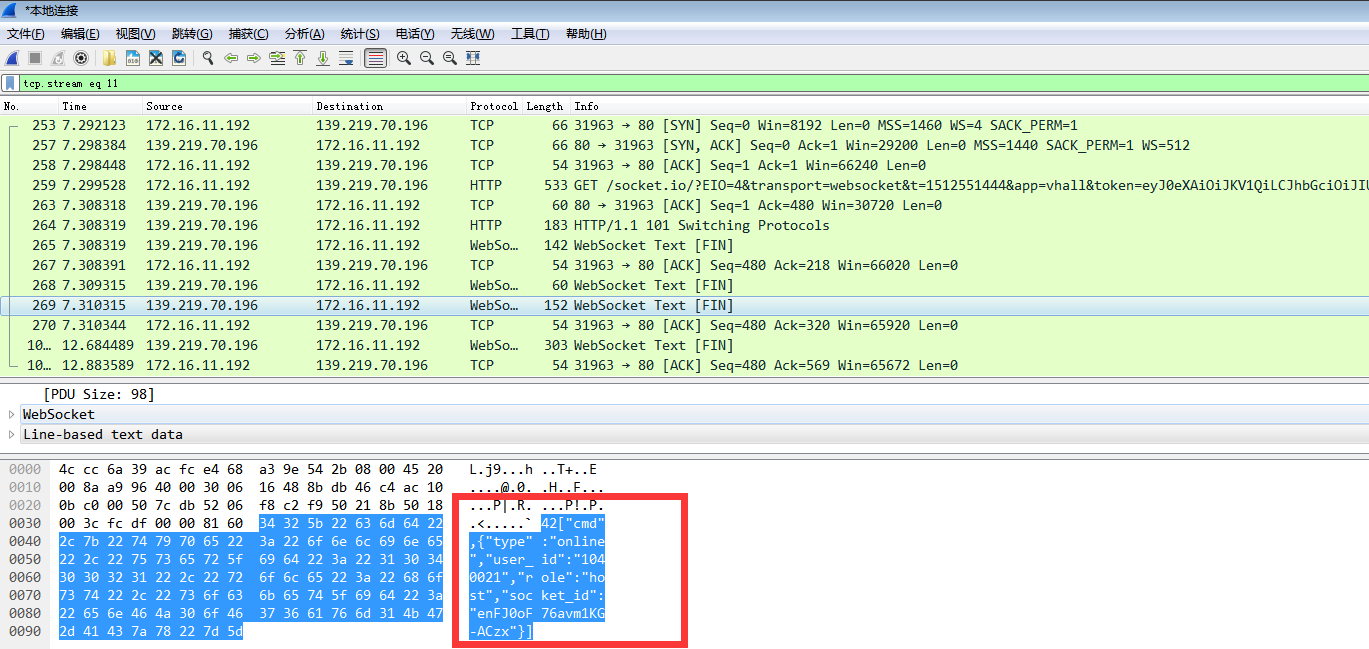
接收文本消息、在线信息、公告等采用Http轮询方式。

如果使用http代理，在进行Http请求时需要进行用户名、密码的认证。

### 在线列表、公告

VhallRightExtraWidget的主要功能是，完成了助手的用户列表、公告消息的处理。若flash方式直接启动助手，则不加载此模块。

目前由于平安证券客户在使用助手时，均是通过HTTP代理服务器，对于websocket消息进行过滤所以收不到在线信息与禁言cmd等。如果将socketio的链接msg直接替换成HTTP，服务器会将协议重新解析为WebSocket，接收在线信息等。这样仍不能满足用户需求。



所以目前将SocketIO统一处理成http-polling方式。其中注意的一点是要间隔性的发送POST消息，发送内容为“1:2”进行链接保活。

具体处理接口为InteractionClient::SocketIOHttpReqWork()。

### 聊天

WebSocket在初始化时创建了一个线程：

BOOL InteractionClient::ConnectWebSocket() {

CBaseThread loSocketIOThread;

loSocketIOThread.BeginThread(InteractionClient::WebSocketThread, this);

return TRUE;

}

线程中主要是对聊天消息，上下线消息的处理。采用HTTP轮询的方式接收。

## 3.9 设置

### 3.9.1 系统设置



#### 3.9.1.1 线路

对于推流线路的获取分为几种情况：

1：读取配置推流配置文件CustomStreamURL.txt，这个配置文件是存放在exe同级目录下吗。

[

{

"displayName": "AESTP1.0自动",

"url": "aestp://rs.edge.live.vhou.net/vhall"

}

,

{

"displayName": "AESTP1.0北京",

"url": "aestp://bjrs01.edge.live.vhou.net/vhall"

},

{

"displayName": "AESTP1.0上海",

"url": "aestp://shrs01.edge.live.vhou.net/vhall"

}

]

2：开启活动时通过接口

<http://t.e.vhall.com/api/client/v1/clientapi/init?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjEwNDAwMjEiLCJuYW1lIjoiMTM1NTIxNDQwNjAiLCJyb29tIjoiODY4MzIzNzExIiwicm9sZSI6Imhvc3QiLCJ1aWQiOiIxNjQxNzQwMSIsInRyYWNrSWQiOiIxMTU2NDk5In0.Zpg8fQDg0-C4DmH9CBdC9sakUyi-cWKbI_m6r8WPCmA>获取到的RTMP推流地址。

[

{

"alias": "自动测试",

"url": "rtmp://rs.live.t.vhou.net/vhall"

},

{

"alias": "华北区测试",

"url": "rtmp://rs.live.t.vhou.net/vhall"

},

{

"alias": "华东区测试",

"url": "rtmp://rs.live.t.vhou.net/vhall"

},

{

"alias": "华南区测试",

"url": "rtmp://rs.live.t.vhou.net/vhall"

}

]

对于http推流地址就是将RTMP队列地址改成HTTP推流地址，同时将端口改成443。

[

{

"alias": "网络代理线路-自动测试",

"url": "http://rs.live.t.vhou.net:443/vhall"

},

{

"alias": "网络代理线路-华北区测试",

"url": " http://rs.live.t.vhou.net:443/vhall"

},

{

"alias": "网络代理线路-华东区测试",

"url": " http://rs.live.t.vhou.net:443/vhall"

},

{

"alias": "网络代理线路-华南区测试",

"url": " http://rs.live.t.vhou.net:443/vhall"

}

]

#### 3.9.1.2 分辨率

在配置文件中配置了视频分辨率级别，下面列出了对应分辨率。此参数是设置推流时的分辨率。

quality=0 对应 640\*360 360p

quality=1 对应 960\*540 540p

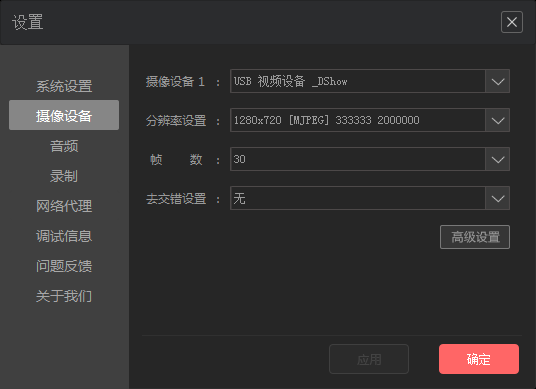
quality=2 对应 1280\*720 1280p

quality=3 对应 1920\*1080 1080p

quality=4 对应 1920\*1080 原始比例

### 3.9.2 摄像设备

主要是分辨率、帧率设置。



### 3.9.3 音频设置

麦克风、降噪控制。



### 3.9.4 网络代理

网络代理需要配置地址、端口，如果代理服务器需要对用户名、密码进行验证需要添加用户名和密码。



## 3.10白板插件

<http://wiki.vhall.com/index.php?id=rd:doc_stream_sync>

<http://redmine.vhall.com/issues/3741> //libEvent

当助手开启直播插件之后，进行白板或文档处理，需要对flash发出的HTTP消息进行处理。其原则是当已经开始直播且直播成功，则将消息直接抛送给MediaCore进行处理。如果没有推流或推流失败，则接收到的消息需要本地存储在数据库中。

### 3.10.1 Web Server

助手中Web Server是有PluginWebServer工程提供的，集成了第三方库libEvent进行http请求的处理。对外提供开始、停止接口与数据回调。

### 3.10.2 流程图

1.开始推流之后，处理流程



2．接收HTTP消息处理。



## 3.11 语音转换显示

语音文字转换功能是用于直播时，通过语音转换将文字显示在预览窗口内的功能。主要功能实现如图。

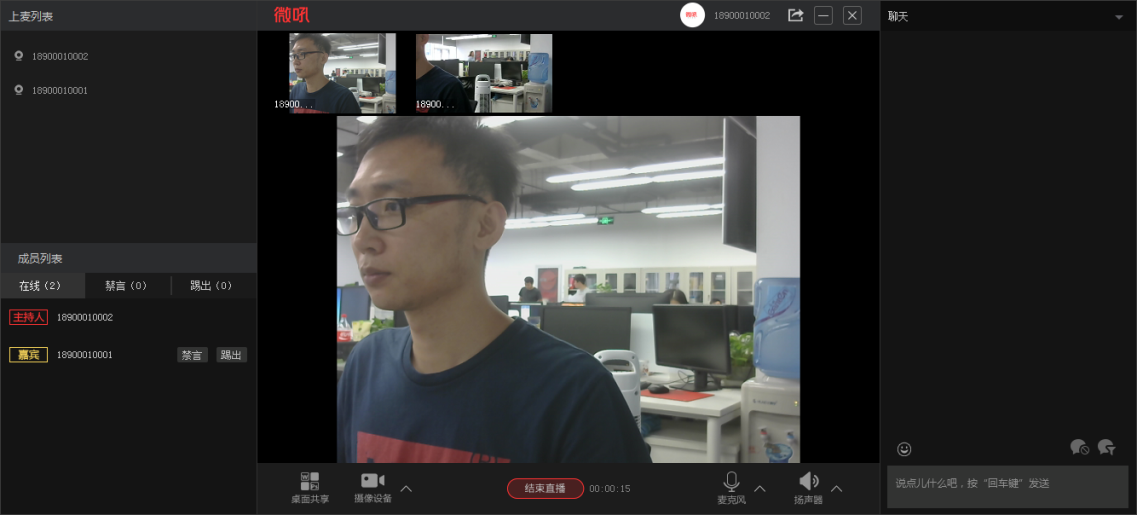


# 4 多人连麦

InteractiveLive对腾讯SDK进行了必要接口封装，例如登陆、登出，创建房间、加入房间等。MainUI中三个模块是在MainUILogic中实例化的成员变量，RoomManger主要处理IM控制消息、上麦成员管理等，线程处理IM控制消息，通过信号与槽的方式向主窗体上报状态。VhallInteractive为多人连麦主界面。VhallRightExtraWidgetLogic主要用户推流状态上报，聊天页面，成员列表状态上报等处理。



**虚线表示回调处理；橙色实现接口调用；蓝色表示信号与槽**



## 4.1登陆

参加活动返回的Msg\_token调用接口获取到腾讯SDK登陆需要的userSig

* 请求：

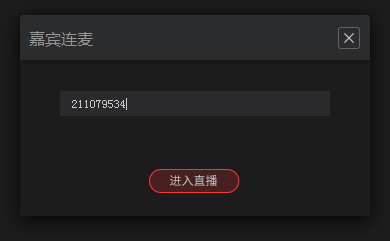
<http://t.e.vhall.com/api/client/v1/clientapi/gen-sig?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjkyMjEwMyIsIm5hbWUiOiIxMzU1MjE0NDA2MCIsInJvb20iOiI5OTk5ODIzMjUiLCJyb2xlIjoiaG9zdCIsInVpZCI6IjE2NDE3NDAxIiwidHJhY2tJZCI6IjEwMjQyNjEifQ.U9kO8rrBRuIXespT39L_Oqdt_HvyupoWxhg847JkwSg>

* 响应：

{"code":200,"msg":"success","data":"eJxlj1FPgzAUhd-5FQ2vGmnLCmiyhzKGEd2WMSfRlwZpp2WzdFAX1PjfVVxiE8-r9917cj4cAIB7e7M6K6uqeVWGmTctXHABXOie-kGtJWelYX7L-0HRa9kKVm6MaAeICCEYQtuRXCgjN-JonGOMoG-xjm-ZUPL7YPR9jUkUIluRTwOcTZeTqySj61jlz7u7vc5RgZty0tE5fCdtsu2zB14lay\*aeuryHlNJVbc4\*H1MFtH1rN9lJsjIKi3qZVp5NX3MT7Sp45Ts50VMx2Or0sgXcVwEIRmFYRBY9CDaTjZqEDBEBGEf-sR1Pp0vIw9bSw\_\_"}

### 4.1.2嘉宾加入

嘉宾加入需要向后台请求当前房间主持人是否已经进行，如果没有进入不允许“进入直播”。允许之后才能调用腾讯SDK直播。



## 4.2上麦处理

* 主持人主动邀请嘉宾上麦处理。目前上麦人数最多为4人。



* 嘉宾主动上麦



## 4.3下麦处理

下麦流程与上麦流程基本上是相反的方向。



## 4.4 屏幕共享

屏幕共享iLiveSDK目前仅支持一个房间只能一个人进行屏幕共享。所以在共享时需要进行逻辑判断。

#### 4.4.1主持人屏幕共享

主持人屏幕共享分为两种情况。1是SDK调用时成功，则表面当前没有用户正在屏幕共享。只需要等待回调之后，请求视频流即可。如图所示。



图2.4

由于屏幕共享在同一个房间只能一人共享，如果SDK返回1008错误码则说明目前正在有人使用屏幕共享，需要主持人通知嘉宾关闭屏幕共享，此时主持人等待回应，如果超时没有收到响应提示共享屏幕失败。流程如图所示



图2.5

#### 4.4.2嘉宾屏幕共享

嘉宾屏幕共享调用判断返回值，如果为1008则不再发起，如果返回值为0，则正常发起。

## 4.5 设备控制

嘉宾的摄像头麦克风可以由主持人控制关闭和打开，下图以关闭麦克风为例，展示业务处理流程。



## 4.6 混流

### 4.6.1腾讯云混流例子



### 4.6.2助手屏混屏处理





input\_type：输入源标识，0表示输入源为流，3表示输入源为画布。

mix\_stream\_template\_id：输入模版ID。目前两输入源支持10、20、30、40；三输入源支持310、390、391；四输入源支持410；五输入源支持510、590；六输入源支持610。不填默认为0。

对于屏幕共享时，没有幕布生成。

<https://www.qcloud.com/document/product/267/8832>

{

"eventId": 1501555251,

"interface": {

"interfaceName": "Mix\_StreamV2",

"para": {

"app\_id": 1253248467,

"input\_stream\_list": [

{

"layout\_params": {

"color": "0x000000",

"image\_height": 1080,

"image\_layer": 1,

"image\_width": 1920,

"input\_type": 3

}

},

{

"input\_stream\_id": "8136\_3761378052f3111798824fd768aac033",

"layout\_params": {

"image\_layer": 2

}

},

{

"input\_stream\_id": "8136\_1eb98b0335034f2885711fec2639dacf",

"layout\_params": {

"image\_layer": 3

}

},

{

"input\_stream\_id": "8136\_dc32fab709042686ac75944961a4c152",

"layout\_params": {

"image\_layer": 4

}

},

//四分屏时将最后一个输入按照画布放置。

{

"input\_stream\_id": "8136\_dc32fab709042686ac75944961a4c152",

"layout\_params": {

"color": "0x000000",

"image\_height": 360,

"image\_layer": 5,

"image\_width": 640,

"input\_type": 3//输入源标识，0表示输入源为流，3表示输入源为画布

}

}

],

"interface": "mix\_streamv2.start\_mix\_stream\_advanced",

"mix\_stream\_session\_id": "211079534",

"mix\_stream\_template\_id": 590,//四分屏

"output\_stream\_id": "211079534\_helper",

"output\_stream\_type": 1

}

},

"timestamp": 1501555251

}

## 4.7 开始、停止推流

* http://wiki.vhall.com/index.php?id=rd:3rdparty\_stream:第三方推拉流
* VhallInteractive::Slot\_BtnStreamClicked()接口实现开始、停止推流状态的判断。

首先需要调用VhallRightExtraWidgetLogic::StartWebnair()通知Saas平台活动状态，改成直播状态。之后通知Vhall媒体服务器开始混流，再通知腾讯服务器混流。

在助手连麦界面，点击开始直播，拼接开始推流URL，通知Vhall媒体服务器。

"rtmp://"+ QString::number(m\_bizid)+".liveplay.myqcloud.com/live/" + mStreamId +STREAM\_TAIL;

void VhallInteractive::ThirdPullStart(QString streamId,QString url) {

//拼接内容

QJsonObject obj;

obj["sessionid"]="vhall\_inter\_"+streamId;

obj["command"]="start";

obj["vhallid"]=streamId;

obj["3rdstream"]=url;

obj["type"]="pull";

obj["ishelper"]="1";

}

void VhallInteractive::ThirdPullStop(QString streamId) {

//拼接内容

obj["sessionid"]="vhall\_inter\_"+streamId;

obj["command"]="stop";

}

# 5 打包发布

通常助手打包在打包服务器上进行打包，每个版本拉取一个项目保存下载，用于后续问题分析。

打包服务器地址是：172.16.11.191

用户名是：Administrator

密码是：vhall.098

## 5.1 工程拉取

1、拉取工程文件

git clone [git@gitlab.vhall.com:vhall\_client/vhall\_vls.git](mailto:git@gitlab.vhall.com:vhall_client/vhall_vls.git) vhall\_vls\_2.0.0

2、进入工程目录

cd vhall\_vls\_2.0.0

3、切换分支

git checkout develop

4、更新子模块

git submodule update --init --recursive

5、编译vhall\_media\_core

1.从vhall\_mdeia\_core ，git的master分支拉取代码，注意子模块也是从master分支拉取代码；将3rdparty中srslibrtmp、libjingle切换到master,并git pull拉取

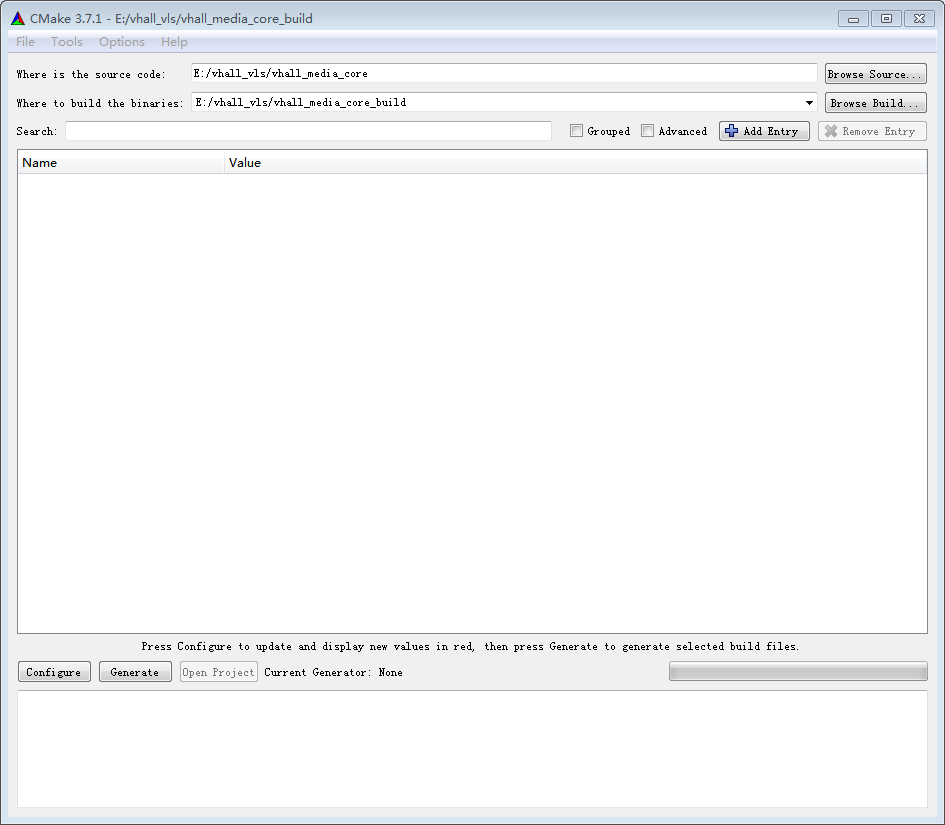
2.重新cmake media\_core;

3.在VhallMediaCore文件夹下添加noise\_suppression,rate\_control,vhall\_audio\_process 3个项目；

4，在项目MeidaCore->属性->链接器->输入->附加依赖项中添加 winmm.lib,noise\_suppression.lib,vhall\_audio\_process.lib,rate\_control.lib 4个库；

5.将vhall\_media\_core->build->windows->libsx86->x264->bin下的libx264-148.dll文件copy到生成目录替换原有的libx264-148.dll文件；

运行：cmake 在第一项中找到vhall\_media\_core路径，第二项为保存路径



点击configure编译出工程文件，之后点击Generate编译出media\_core对应的vs2013工程项目。

## 5.2 修改版本号

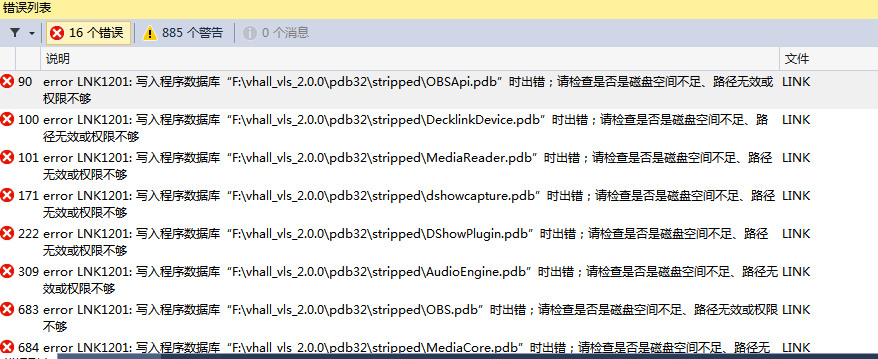
修改版本号：  
修改如下文件 助手目录\VhallLive\project\VhallLive.rc  
一般使用记事本打开

每次只修改版本号部分

IDI\_ICON1 ICON DISCARDABLE "Vhall.ico"   
#if defined(UNDER\_CE)  
#include <winbase.h>  
#else  
#include <winver.h>  
#endif

VS\_VERSION\_INFO VERSIONINFO  
FILEVERSION 2,0,0,28  
PRODUCTVERSION 2,0,0,28  
FILEFLAGSMASK 0x3fL  
#ifdef DEBUGFILEFLAGS VS\_FF\_DEBUG#elseFILEFLAGS 0x0L#endifFILEOS VOS\_WINDOWS32   
FILESUBTYPE 0x0L  
BEGIN  
BLOCK "StringFileInfo"   
BEGIN  
BLOCK "080004b0"   
BEGIN  
VALUE "CompanyName", "Vhall Corporation"   
VALUE "FileDescription", "VhallLive"   
VALUE "FileVersion", "2.0.0.28"   
VALUE "InternalName", "Vhall"   
VALUE "LegalCopyright", "Copyright 2017"   
VALUE "OriginalFilename", "VhallLive.exe"   
VALUE "ProductName", "VhallLive"   
VALUE "ProductVersion", "2.0.0.28"   
END  
END   
END

修改版本好以后，打开vHallLive.sln工程，编译程序的release版本。注意在调试时需要在开始菜单，右键点击vs2013应用，使用管理员权限打开。然后进行编译全部应用。



在工程pdb32目录下建立stripped重新编译即可。

## 5.3 工程打包

### 5.3.1 工具安装

1.安装 isetup-5.5.5.exe ，默认选项，一直下一步到完成安装  
2.ChineseSimp.isl 和 Chinese.isl 复制到 C:\Program Files (x86)\Inno Setup 5\Languages 目录下  
3.C:\Program Files (x86)\Inno Setup 5\Compil32.exe 对此文件执行：右键->属性->兼容性->勾选 以管理员身份运行此程序  
至此，完成inno的安装

### 5.3.2 签名与打包

#### 5.3.2.1 配置文件修改

修改配置文件Config.ini配置文件地址环境配置。

测试环境地址是：t.e.vhall.com  
线上环境地址是：e.vhall.com //发版时使用此环境

#### 5.3.2.2 隐藏logo（可选项）

1:添加了配置文件vhall\_tool\_config.ini，具体内容如下：

[default]  
vhall\_live=微吼直播 //用于读取显示助手窗体默认名称  
vhall\_helper=微吼助手 //应用名称  
vhall\_logo\_hide=0 //是否隐藏logo,0：不隐藏 1：隐藏  
vhall\_app\_update\_enable=1 //是否支持在线升级，0：不支持 1：支持

2：当隐藏logos时，加载页面加载的配件图片是从安装目录直接读取的图片：vhalllogoNoVhall.png、VhallLiveBGNoVhall.png、splash.png

3：在打包文件中，将File中添加：  
Source: "..\Release\splash.png"; DestDir: "{app}"; Flags: ignoreversion  
Source: "..\Release\VhallLiveBGNoVhall.png"; DestDir: "{app}"; Flags: ignoreversion  
Source: "..\Release\vhalllogoNoVhall.png"; DestDir: "{app}"; Flags: ignoreversion

4：后续如果有其他配置，可以从该配置文件中进行添加。

#### 5.3.2.3 打包

进入到 助手目录/install 目录下  
打开cmd窗口也进入到 助手目录/install目录下

1.对动态库进行签名  
在cmd窗口里执行：VhallSign\_little2.bat 对助手编译出的动态库和可执行程序进行签名

2.打包  
双击 VhallLive.iss 之后自动打开inno工具  
在第一行 #define EXE\_VER "2.0.0.28" 处，修改版本号  
点击inno工具里面的Compile图标开始编译生成安装包，生成的安装包位于 助手目录\install\setup

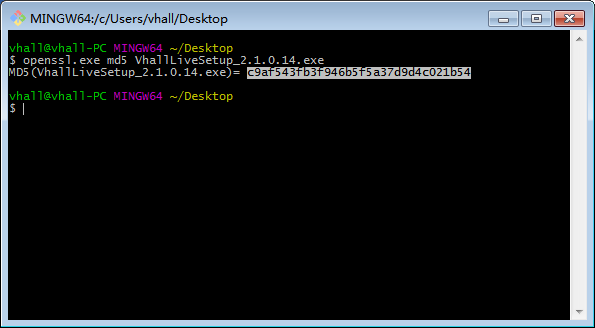
\*.iss语法可以参照：

http://redmine.vhall.com/issues/4393

3.对安装包进行签名  
使用记事本编辑签名脚本VhallLast2.bat  
signtool sign /f ./wosign/vhall20160522.pfx /p 15110073312Vhall /t <http://timestamp.wosign.com/> ./setup/VhallLiveSetup\_2.0.0.28.exe  
修改上面的版本号  
之后保存，在cmd窗口内执行VhallLast2.bat对安装包进行签名

### 5.3.3 提测

提测时需要更加安装包生成md5，在git命令行窗口输入命令后如图所示，生成MD5值。上传redmin之后，将md5信息附带上。





## 5.4 创建分支

先拉取工程代码，更新子模块。

进入develop执行 git branch shidaiguanghua2

然后git checkout 到shidaiguanghua2便进入分支。

git push origin shidaiguanghua2进行提交

## 5.5 冲突

冲突处理。

先git commit –a提交

然后git pull 拉取后对比修改。

## 5.6 分支合并

<http://blog.csdn.net/hudashi/article/details/7664382>

将hudong分支内容合并到develop.

* 首先下载develop分支代码：git checkout develop
* 执行：git pull ; git submodle update –init --recursive
* 其次：git merge hudong，将hudong分支代码合并到develop
* 最后：git push origin develop，将合并的工程提交。

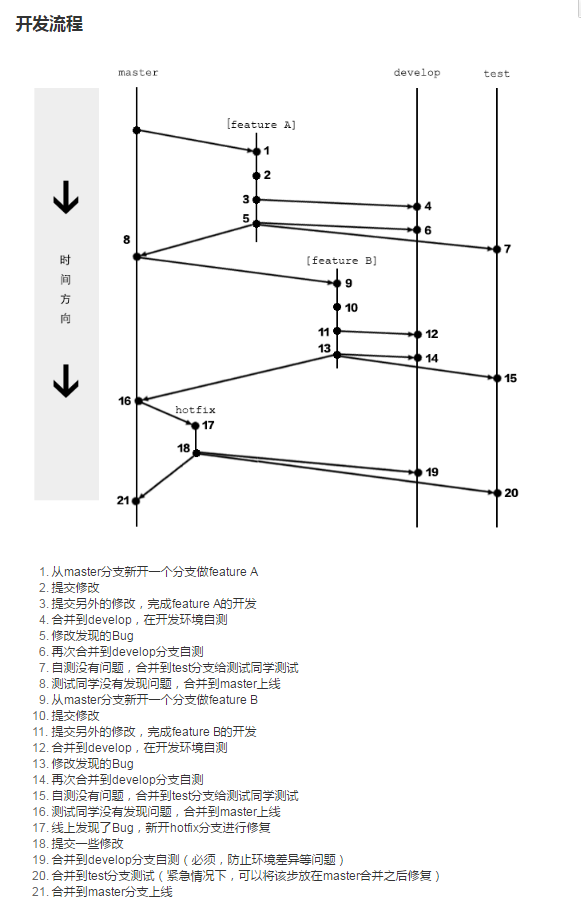
## 5.7 打标签

1、查看tag: git tag；

2、创建tag: git tag -a v2.0.5.3

3、提交tag: git push origin v2.1.1.23

## 5.8 Git管理规范



<http://wiki.vhall.com/index.php?id=rd:smb:git>

## 5.9 助手QT工程依赖文件

**助手QT工程依赖库的查找**  
a: 先将应用程序拷贝到临时目录。 E:\1\VhallLive.exe  
b: 进入目录C:\Qt\Qt5.7.1\5.7\msvc2013\bin；  
c: 在cmd控制台输入C:\Qt\Qt5.7.1\5.7\msvc2013\bin> windeployqt.exe E:\1\VhallLive.exe  
d：之后会看到所依赖的QT文件。

## 5.10 删除分支

1：git branch -r -d origin/QT\_Updata

2：git push origin :QT\_Updata

使用 git rm 命令即可，有两种选择,

一种是 git rm --cached "文件路径"，不删除物理文件，仅将该文件从缓存中删除；

一种是 git rm --f "文件路径"，不仅将该文件从缓存中删除，还会将物理文件删除（不会回收到垃圾桶）。

## 5.11 新工程打包

1：安装Qt\_5.7.1 qt-vs-addin-1.2.5.exe  
2：配置QTDIR  
3：拉取代码后，切换到分支git checkout Qt\_5\_7\_1  
4：更新子模块git submodule update --init –recursive  
5：vhall\_media\_core切换到master，执行git submodule update --init –recursive更新子模块。

6：vhall\_media\_core/3rdparty/audiomixing切换到develop,此模块用于音频混音处理  
7：VhallRightExtraWidget\Resources\chat\_html切到qt\_5\_7，修改了QT与js的注册方法。  
8：Release\Debug切到develop，更新了QT5.7.1的dll

9：打包使用的脚本 install/VhallLive\_Qt\_5.7.1.iss

# 6 腾讯后台管理

登陆地址：https://console.qcloud.com/

腾讯云测试账号：2425825420

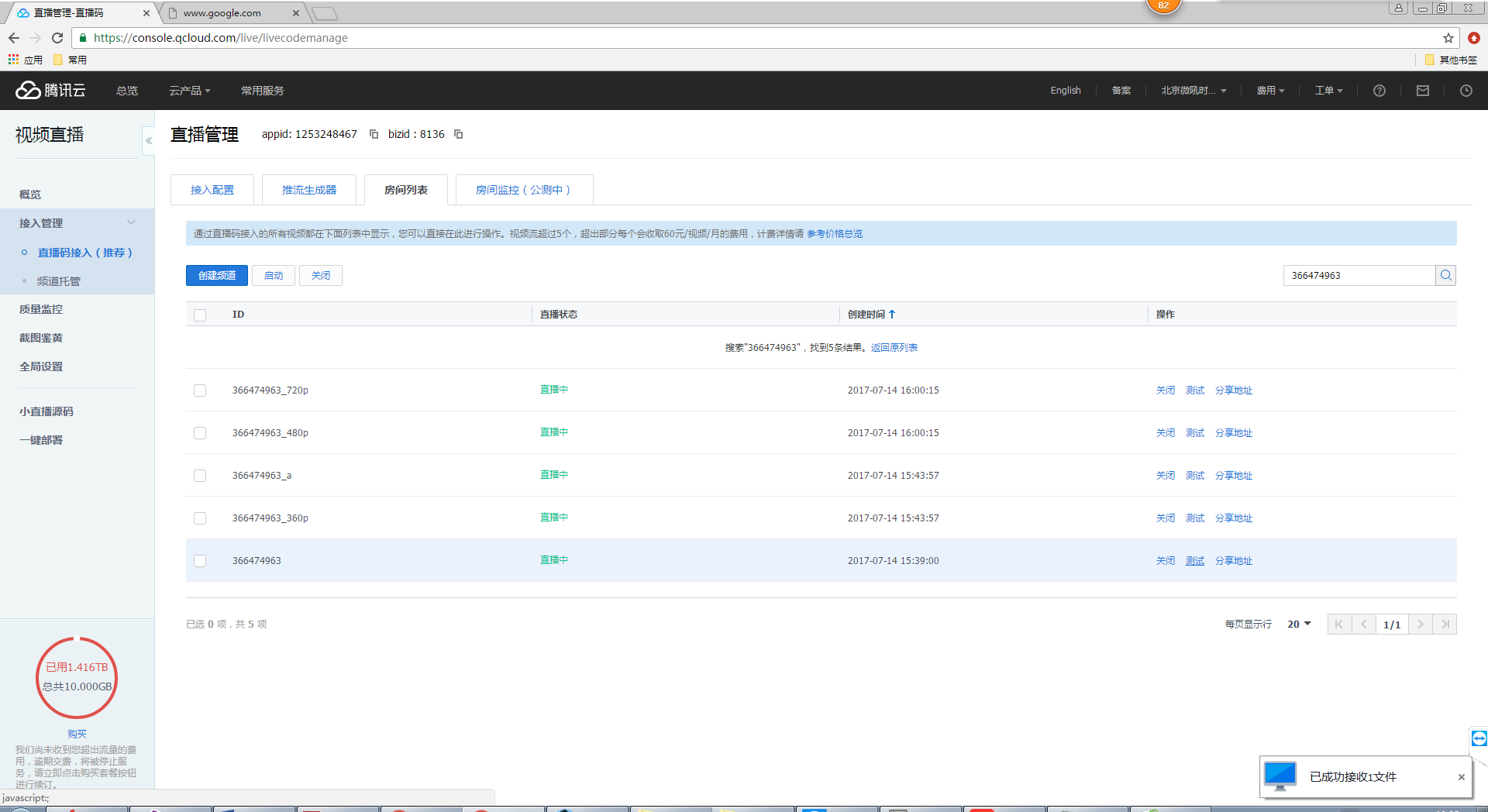
密码：vhall2017

## 6.1直播管理



直播主要查看流信息，互动直播主要看房间信息；

下图可以查看流信息，这部分流是通过付壮端推到腾讯云的视频。



开始直播是通知付壮开始拉流，拉流后推至腾讯云。

付壮端去腾讯云拉流地址：

rtmp://8136.liveplay.myqcloud.com/live/366474963\_helper

付壮端推流地址，推至腾讯CDN：

rtmp://8136.liveplay.myqcloud.com/live/366474963

用于测试使用，观看流状态工具。

<http://www.cutv.com/demo/live_test.swf>



## 6.2命令行停止拉流

在git里面输入：

curl -X POST -H "Content-Type: application/json" -d '{"sessionid":"vhall\_inter\_211079534", "command":"stop","vhallid":"211079534","3rdstream":"rtmp://cnrtmplive02.t.vhall.com/vhall/123","type":"pull","ishelper":"1"}' "http://t-relay.e.vhall.com:5000/api/3rdstream/"

curl -X POST -H "Content-Type: application/json" -d '{"sessionid":"vhall\_inter\_366474963", "command":"stop","vhallid":"366474963","3rdstream":"rtmp://cnrtmplive02.t.vhall.com/vhall/123","type":"pull","ishelper":"1"}' "http://t-relay.e.vhall.com:5000/api/3rdstream/"

curl -X POST -H "Content-Type: application/json" -d '{"sessionid":"vhall\_inter\_999982325", "command":"stop","vhallid":"999982325","3rdstream":"rtmp://cnrtmplive02.t.vhall.com/vhall/123","type":"pull","ishelper":"1"}' <http://t-relay.e.vhall.com:5000/api/3rdstream/>

curl -X POST -H "Content-Type: application/json" -d '{"sessionid":"vhall\_inter\_ 569377616", "command":"stop","vhallid":" 569377616","3rdstream":"rtmp://cnrtmplive02.t.vhall.com/vhall/123","type":"pull","ishelper":"1"}' <http://t-relay.e.vhall.com:5000/api/3rdstream/>

curl -X POST -H "Content-Type: application/json" -d '{"sessionid":"vhall\_inter\_569377616", "command":"stop","vhallid":"569377616","3rdstream":"rtmp://cnrtmplive02.t.vhall.com/vhall/123","type":"pull","ishelper":"1"}' <http://e-relay.e.vhall.com:5000/api/3rdstream/>

# 7 QT应用总结

## 1. Qt自动生成moc文件

有些情况，如果创建的工程不是QT工程，很难进行信号与槽的处理。这时就要手动添加Moc\_XXXX文件。

针对助手开发使用的VS2013为例，具体操作如下。

在VS的解决方案管理器里面，右键需要生成moc文件的头文件-打开属性窗口;

常规-项类型，选择自定义生成工具；

关闭属性窗口重新打开,就出现了自定义生成工具选项；

自定义生成工具-常规：

命令行："$(QTDIR)\bin\moc.exe"  "%(FullPath)" -o ".\GeneratedFiles\$(ConfigurationName)\moc\_%(Filename).cpp"

输出：.\GeneratedFiles\$(ConfigurationName)\moc\_%(Filename).cpp

如果编译还有问题，在解决方案资源管理器里阿敏，右键生成的moc文件打开属性窗口；

配置属性-常规：设置下在debug和release下面分别设置下是否在项目中排除的属性;

# 8 日志上报

在每个需要日志上报的模块中，XXXins实例内都封装了static void reportLog(const eLogRePortK ekey, const QJsonObject body);

## 8.1实现逻辑和思路：

1.在需要上报日志的地方调用该静态函数，将需要上报的日志消息MSG\_MAINUI\_LOG通过post方式post出；

2.消息MSG\_MAINUI\_LOG将在MainUILogic中被包装处理,并以同样的方式将处理好的消息MSG\_HTTPCEMYER\_HTTP\_LOG\_RQpost出；

3.消息MSG\_HTTPCEMYER\_HTTP\_LOG\_RQ将被CommonToolKitIns添加到任务队列（HttpReqLogic实例）m\_oHttpReqLogLogic中；

4.m\_oHttpReqLogLogic内任务线程不停轮循执行HTTP请求任务，将日志真正的get到服务器；

## 8.2日志格式描述：

日志数据内容分json和token两部分：json的值是明码可以直接读取，token部分通过base64转换先反向转换后才能查看；例如开始推流日志上报格式如下：

[http://pingback.e.vhall.com/zs.gif?id=%ws&s=%u&key=stream\_start&stream\_add=%ws&%ws&b\_agent=%ws&b\_record=%ws&b\_dispatch=%ws&k=%d&streamid=%ws&token=%ws](http://pingback.e.vhall.com/zs.gif?id=%25ws&s=%25u&key=stream_start&stream_add=%25ws&%25ws&b_agent=%25ws&b_record=%25ws&b_dispatch=%25ws&k=%25d&streamid=%25ws&token=%25ws)

## 8.3日志上报的内容和字段含义：

**8.3.1上报信息分类**：

详见：<http://wiki.vhall.com/index.php?id=rd:media_monitor:monitor_vhall_helper>

**8.3.2 上报信息中字段含义：**

详见：<http://wiki.vhall.com/index.php?id=rd:media_monitor:monitor_home_page>

## 8.4 日志查看

1.登录微吼直播故障查询系统 l.vhallops.com；

2.点击左侧 “小助手日志查询”，根据活动ID可查看basic\_info\init\_info\stream\_info\stream\_start\stream\_end\stream\_error六类日志；

3.其它日志可以通过如下开源日志查看系统查看：

http://elk.vhallops.com/app/kibana#/discover?\_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:now-15m,mode:quick,to:now))&\_a=(columns:!(\_source),index:'web-vhall-\*',interval:auto,query:(query\_string:(analyze\_wildcard:!t,query:'\*')),sort:!('@timestamp',desc))

# 9 Dmp查看

## 9.1 Dmp 下载

下载连接：<http://helpdump.t.vhall.com:5588/dumpfile/>

查看事例：<http://helpdump.t.vhall.com:5588/dumpfile/2017/12/05/>

下载后可以查看和调试dmp

## 9.2 Dmp 上传

<http://helpdump.t.vhall.com:5587/upload>?

http://helpdump.t.vhall.com/upload?

streamID=&version=2.1.1.0&filename=C:\Users\xuke\Documents\VHallHelper\1512112506016.dmp&crashTime=2017-12-01 15:15&MengZhu=0&token=3041839952

上传端口：80\5587