互动直播

重要概念

产品文档

腾讯云

文档目录

重要概念
帐号登录集成
帐号登录集成说明

最近更新时间：2019-05-10 14:29:38

腾讯登录服务（Tencent Login Service，TLS）是腾讯为开发者快速完成帐号集成接入音视频或即时通讯云服务（后面简称云服务）而提供的一套通用帐号登录组件。

帐号体系集成位于云服务接入流程的第四步。在帐号体系集成前，您需要首先注册腾讯云账号，开通服务和创建应用，具体方法请单击『应用接入指引』。 本文后续将重点讲解如何进行帐号集成。另外我们针对这里提到的内容录制了视频，可以在帐号 集成示例 进行观看。

两种模式介绍

1. 如果您的应用已经发布，用户较多，独立模式 可以帮您快速集成云服务（**温馨提醒：帐号的独立模式并不影响资料、关系链和群组的托管**）；
2. 如果您不想维护复杂的用户帐号系统，托管模式 可以帮您方便快速地搭建安全的应用自有帐号体系。

附录

集成模式

根据开发者使用场景，我们为开发者提供了两种不同的帐号集成方式，开发者可根据自身情况选择。

- 独立模式：用户帐号信息由开发者保存，用户身份验证（比如注册与验密）也由开发者负责；
- 托管模式：由腾讯为开发者提供帐号的密码注册、存储和密码验证，以及第三方 `openid` 和 `token` 的托管验证服务。

> ⚠ **注意：**

帐号集成模式只能选择一次，后续无法变更，请谨慎选择。如果想改变集成模式，请创建新的应用重新走帐号集成流程。

公私钥

公私钥是密码学中非对称加密算法中的概念。非对称加密算法中，公钥与私钥是成对出现的，私钥需要保密，公钥是公开的。用公钥加密的数据，只有私钥才能解开；用私钥加密的数据，也只有公钥才能解开。在帐号集成中，公私钥用于鉴别开发者用户的身份合法性，使用场景如下：

- 独立模式：私钥由开发者保存，公钥由腾讯保存。开发者使用私钥生成用户签名 UserSig，腾讯使用公钥对签名 UserSig 进行校验。
- 托管模式：私钥由腾讯保存，公钥由开发者保存。腾讯使用私钥生成用户签名 UserSig，开发者可以使用公钥对签名 UserSig 进行校验，注意，对于第三方开放帐号，此时不需要公私钥。

对于独立模式，为了方便开发者快速开发，开发者进行帐号集成后，可以在应用列表的应用配置中 下载公私钥。同时我们也提供了工具供开发者调试使用，详情请参见 TLS 后台 API 开发指引。

说明：

关于非对称算法，请访问 公开密钥加密 进一步了解。

如何生成公私钥

详见『TLS 后台 API 开发指引』中的『工具使用』章节。

App 管理员

对外的开放接口中，有些操作是管理员身份才能调用的，例如解散群，给用户发 C2C 消息等。因此在开发者的帐号体系中，可以标识一些帐号为管理员，在腾讯验证管理员身份后，管理员可以对自己的业务进行一些特殊操作。
App 管理员是对 App 具有最高管理权限的角色，与普通帐号相比，其区别如下。

- 读取权限更高。例如获取 App 内部的所有群组、获取任意群组的任意资料。
- 操作权限更高。例如给任意用户发消息、在任意群组中增删成员。

App 在调用 REST API 进行服务端集成时，应当使用 App 管理员帐号作为 `identifier` 进行调用。App 可以在帐号体系中将数个帐号设置为 App 管理员，设置方法参见 设置 App 管理员 文档。

短信验证码内容

开发者选择托管模式集成自有帐号后，由腾讯为开发者提供帐号的注册和登录服务。在开发注册和功能时，开发者可以选择让用户通过 短信验证 的方式进行注册和登录。验证码短信格式如下。

1234（XXXX短信验证码），请勿转发，否则会导致帐号被盗。【腾讯云】

- 以上内容开发者只能更改『XXXX』部分，可填写公司名称或者 App 名称，长度 30 个字符以内。
- 『1234』为随机的四位数字，实际使用时会随机生成。
- 【腾讯云】为默认短信签名（应运营商要求，下发短信必须要有短信签名）。如果您的月短信量超过 1 万条，可以联系客服给您定制短信签名。
- 如果开发者没有填写『XXXX』部分，则使用默认的短信验证码，格式如下。

```
1234（短信验证码），请勿转发，否则会导致帐号被盗。【腾讯云】
```

identifier 格式说明

`identifier` 即用户帐号标识，以下是对此参数的格式说明。

- **独立模式下**，`identifier` 长度建议不超过 32 字节 。
- **托管模式下**，字符串类型的 `identifier` 长度为4 - 24个字节，请使用英文字符和下划线，不能全为数字，大小写不敏感。

联系我们

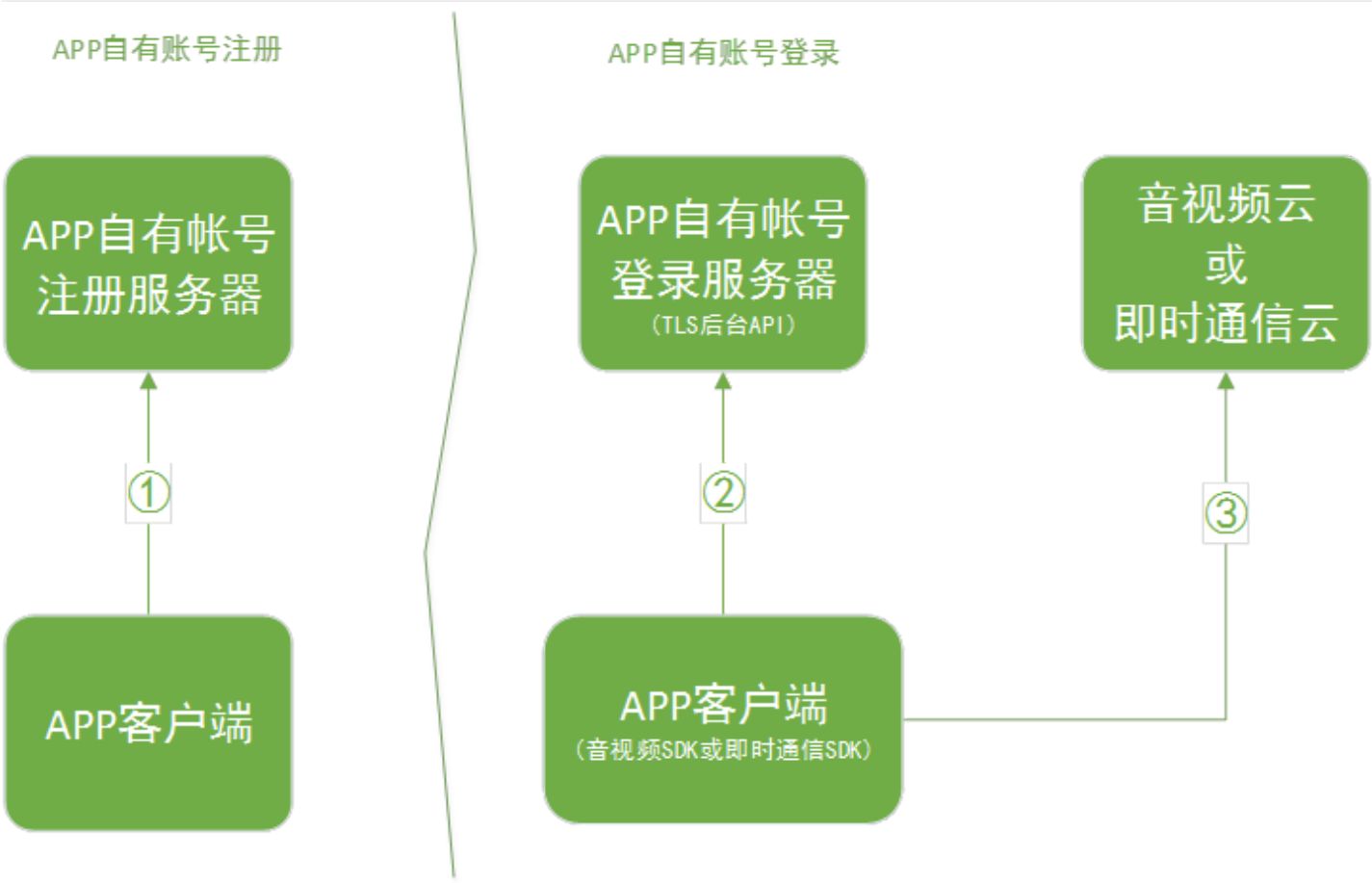
如果遇到问题您可以先到 腾讯云论坛 中查找解决方法，如仍需支持，请 @TLS 帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

独立模式

最近更新时间：2019-02-26 16:26:41

独立模式是指用户注册和身份验证由开发者负责，开发者和腾讯之间通过签名验证建立信任关系。开发者在申请接入时，直接 下载公私钥 用于开发，而后用私钥加密指定数据生成签名交由腾讯服务器验证合法性。

App 自有帐号



注册说明

帐号注册在 App 自有注册服务器完成，帐号密码无需同步到腾讯。

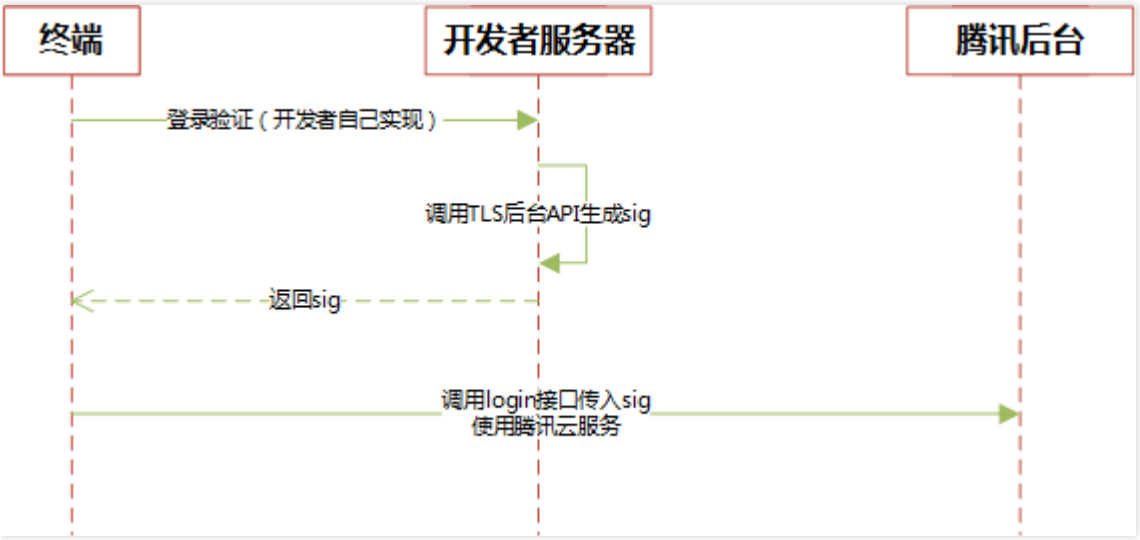
登录说明

用户在 App 客户端输入帐号密码后到 App 自有帐号登录服务器验证，验证成功后，App 自有帐号登录服务器使用私钥派发签名（ `UserSig` ）给客户端。App 客户端使用用户帐号和私钥签名，调用音视频 SDK 或即时通讯 SDK 的接口进行验证，验证成功后即可使用音视频云或即时通讯云服务。

使用说明

开发者需要保证私钥安全，腾讯完全信赖私钥签名。TLS 后台 API 默认接口生成的签名有效期为 180 天，开发者可以使用含有有效期参数的接口自行设定有效期，开发者需要在签名过期前到开发者后台获取新的签名，TLS 后台 API 详见 TLS后台API使用手册。

典型流程



第三方开放帐号

开发者集成第三方开放帐号，流程上与自有帐号的集成方式一致，在腾讯这一侧使用的都是 `identifier`（用户 ID）和 `UserSig`。

联系我们

如果遇到问题您可以先到 腾讯云论坛 查找解决方法，如仍需支持，请 @TLS 帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

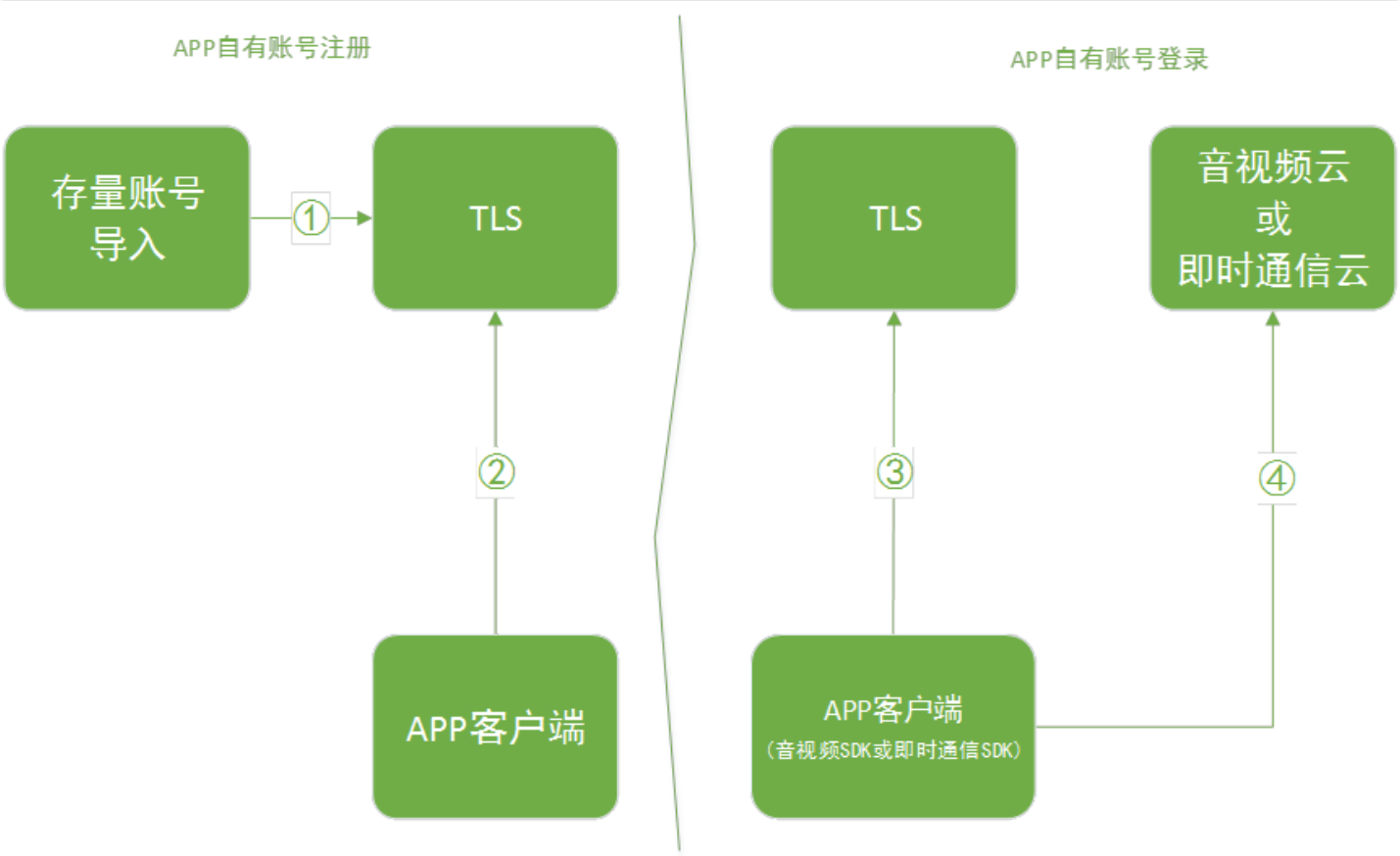
托管模式

最近更新时间：2019-03-05 10:49:21

托管模式是指，由 TLS 为开发者提供 App 帐号的密码注册、存储和密码验证。帐号验证成功后，派发私钥加密生成的签名到客户端，App 业务服务器可以通过 下载的公钥 解密签名进行验证。

> 注：托管模式暂仅支持 Android 和 iOS 平台。如有 PC 和 Web 平台需求，请使用独立模式。

App 自有帐号



注册说明

1. App 自有帐号注册有 2 个接口。
2. （可选）提供 接口 支持存量帐号导入。
3. 新帐号注册使用 TLS SDK 完成。

登录说明

1. 用户在客户端输入帐号密码后，到 TLS 验证。
2. 登录后可以直接使用音视频或者即时通讯云服务。
3. （可选）App 应用服务器可以使用 TLS 后台 API 对签名进行验证，用以确定用户的合法性。

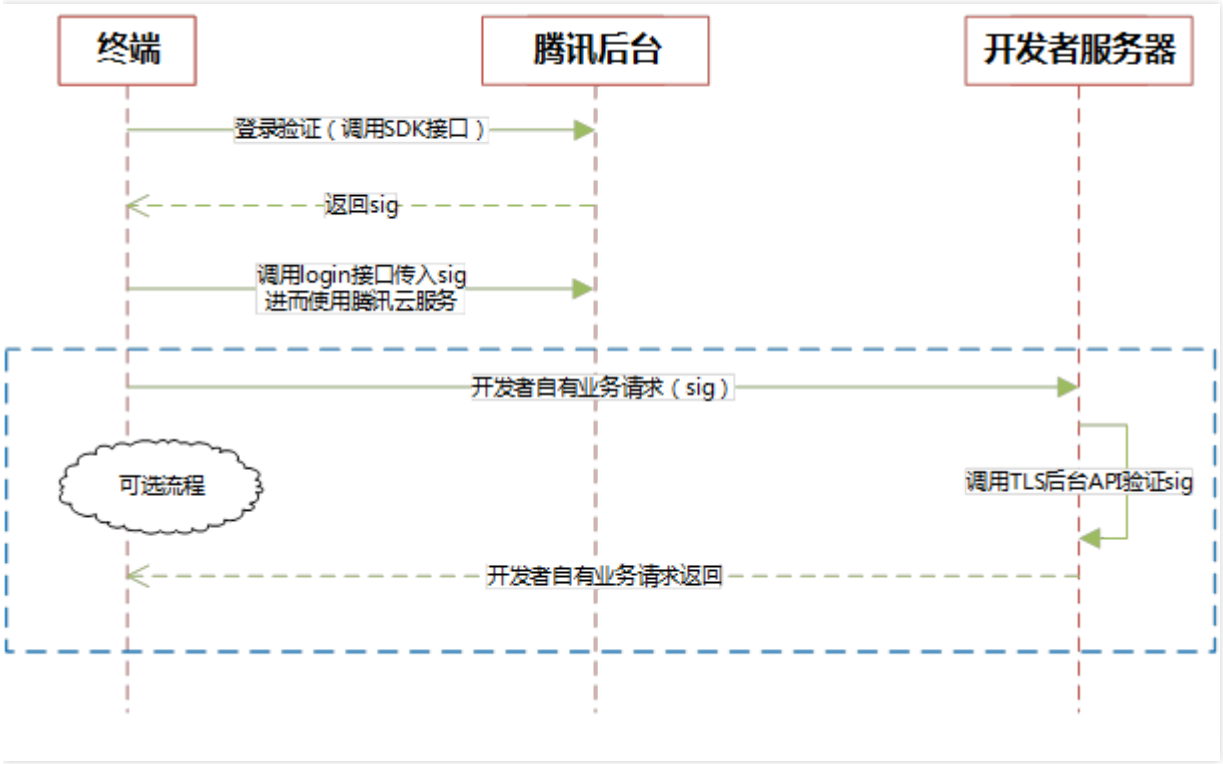
使用说明

1. 直接使用 TLS SDK 即可快速完成注册和登录能力集成。
2. 已上线的 App 也可以通过存量帐号导入使用托管模式。

安全性说明

帐号密码存储安全级别与 QQ 帐号相同，能够保证拖库后帐号安全。

典型流程



联系我们

如果遇到问题您可以先到 腾讯云论坛 中查找解决方法，如仍需支持，请 @TLS 帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

TLS 后台 API

最近更新时间：2019-05-15 09:57:33

开发者可以使用 TLS 后台 API 及相关工具，生成公私钥、生成 `UserSig` 和校验 `UserSig`。TLS 后台 API 我们提供了6个包供开发者 下载，内容分别是 Windows 下64位预编译文件包、Windows 下32位预编译文件包、Linux 下64位预编译文件包、Linux 下32位预编译文件包、zip 格式的源代码文件和 tar.gz 格式的源代码文件。

> ⚠ **注意：**
>
> 在控制台上下载的公私钥文件名分别为 `private_key` 和 `public_key`，分别对应下面的 `ec_key.pem` 和 `public.pem`。请在使用公私钥时注意区分。

Linux 平台

工具使用

> ⚠ **注意：**
>
> 这里讲解的是工具的使用说明，实际应用中需要开发者后台调用 TLS 的后台 API 接口生成 sig。

Linux 下生成 sig 和校验 sig

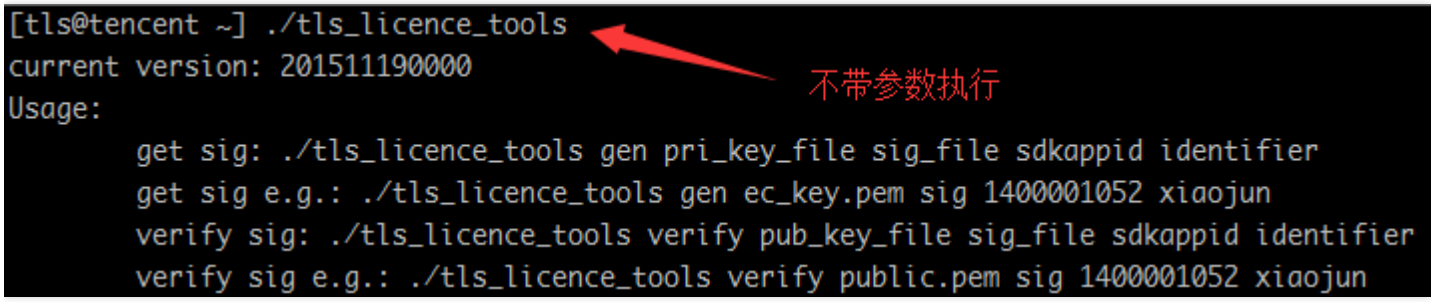
首先不带参数执行 `tls_licence_tools`，即执行下面的命令：

```
$ ./tls_licence_tools
```

输出：

```
current version: 201511190000
Usage:
get sig: ./tls_licence_tools gen pri_key_file sig_file sdkappid identifier
get sig e.g.:./tls_licence_tools gen ec_key.pem sig 1400001052 xiaojun
verify sig:./tls_licence_tools verify pub_key_file sig_file sdkappid identifier
verify sig e.g.: ./tls_licence_tools verify public.pem sig 1400001052 xiaojun
```

输出实际上是参数模板和示例。下面是演示截图：



执行类似于下面的命令可以生成 sig：

> ./tls_licence_tools gen ec_key.pem **sig** 1400001052 xiaojun

对应的参数解释是：

> ./tls_licence_tools gen 私钥文件路径 sig将要存放的路径 sdkappid 用户**id**

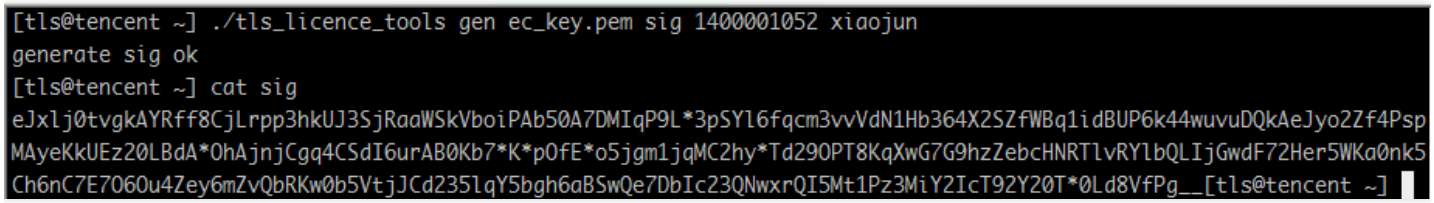
执行类似于下面的命令可以校验 sig：

> ./tls_licence_tools verify **public**.pem sig 1400001052 xiaojun

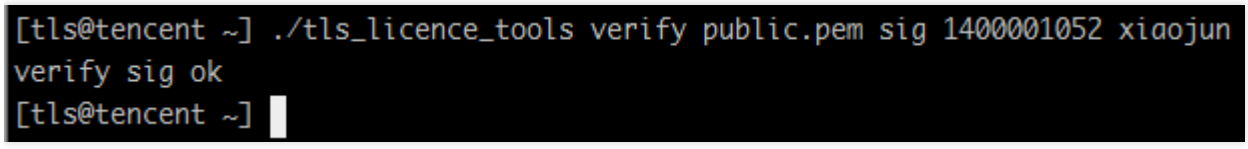
对应参数的解释是：

> ./tls_licence_tools verify 公钥文件路径 sig的存放路径 sdkappid 用户**id**

以下为生成 sig 演示：



以下为校验 sig演示：



以下为参数模板中各个参数的意义：

gen 和 verify 分别表示生成 **sig** 和校验 **sig** 的命令
pri_key_file：私钥文件的路径
pub_key_file：公钥文件的路径
sig_file：**sig** 文件的路径，如果是生成 **sig**，那么会将 **sig** 写入这个文件，如果是校验 **sig**，那么会从这个文件读取 **sig** 的内容
sdkappid：创建应用时页面上分配的 sdkappid
identifier：用户标识，即用户 id

> ⚠ **注意：**
>
> 生成的 sig 有效期为180天，开发者需要在 sig 过期前，重新生成 sig。

C++ 接口

首先包含 `include/tls_sig_api` 目录下的 `tls_signature.h` 。头文件中包含的接口， `tls_gen_signature_ex2` 和 `tls_check_signature_ex2` ，前者是生成 sig 的接口，后者是校验 sig 的接口，详细的参数和返回值说明请参考头文件 `tls_signature.h` 。

然后是链接静态库，在 lib 目录下有下列目录：

```
├── jni
├── jsoncpp
├── openssl
└── tls_sig_api
```

需要链接的静态库是 libjsoncpp.a、openssl 目录下的 libcrypto.a 和 libtlsignature.a。另外还需要链接系统的 -ldl 和 -lz ，详细可以查看 example/cpp/Makefile，由于 libtlsignature.a 引用了 openssl 和 json 的开发库，所以链接时 libtlsignature.a 出现在命令的最前面。

下图是在我们的开发环境编译 tls_licence_tools 的命令行，实际开发时，链接库的路径可以按照开发者自己的情况给出。

```
[tls@tencent ~] make
g++ -g -I../../include -I../../include/tls_sig_api -Wall -fPIC -c tls_licence_tools.cpp
g++ tls_licence_tools.o ../../src/libtlsignature.a -o -Wall tls_licence_tools -g -I../../include -I../../include/tls_sig_api -Wall -fPIC -L../../linux/64/lib/jsoncpp ../../openssl-dynamic/lib/libcrypto.a -ljsoncpp -ldl -lz
```

> ⚠ **注意：**
>
> 如果程序有多线程调用 TLS 后台 API 的用法，请在程序初始化时和结束时分别调用下面的接口：

```
int multi_thread_setup(void);
void multi_thread_cleanup(void);
```

Java 接口

目前 Java 接口使用 JNI 的方式实现。Java 目录下 tls_sigcheck.class，是由 tls_sigcheck.java 编译得到的，如果有 jdk 兼容性问题，开发者可自行重新编译此文件，编译命令为：

```
javac -encoding utf-8 tls_sigcheck.java
```

请注意接口的包路径为 `com.tls.sigcheck` ，典型的使用方法是 `example` 目录下 Java 版本 Demo 的组织方式。

```
├── com
│   └── tls
│   └── sigcheck
│   └── tls_sigcheck.class
├── Demo.class
├── Demo.java
├── ec_key.pem
├── public.pem
└── README
```

之前提到 Java 接口目前使用的 JNI 的方式，所以 `Demo.java` 调用了载入 so 的语句，开发者根据自己的存放 `jnisigcheck.so` 实际路径进行修改，在二进制包中预编译的 `jnisigcheck.so` 存放在 `lib/jni` 目录下。Demo 的使

用方式请参考 example/java/README 。下面是演示截图：

说明：

如果在 Java 代码中使用了**多线程**的方式生成 usersig ，可以参阅 腾讯云论坛 相关介绍。

Java 原生接口

Java 原生接口依赖于5个 jar 包。在 tls_sig_api/java_native/lib 目录下：

- **bcpkix-jdk15on-152**.jar
- **bcprov-jdk15on-152**.jar
- **commons-codec-1**.10.jar
- **gson-2**.3.1.jar
- **json**.jar
- **tls_signature**.jar

注意：

从控制台界面 下载 的公私钥，将公钥内容赋值给接口中的 publicBase64Key 参数，私钥内容赋值给接口中的 privateBase64Key 参数。

PHP 接口

PHP 实现的方式较为简单，就是调用命令行工具生成 sig，工具是 `bin/signature.exe` ，PHP 的调用方式如下：

> **说明：**
>
> 开发者请注意命令执行的路径和可执行权限，如果出现问题请尝试打印出 `command` 变量的内容进行定位。

```
function signature($identifier, $sdkappid, $private_key_path)
{
# 这里需要写绝对路径，开发者根据自己的路径进行调整
$command = '/home/signature'
. ' ' . escapeshellarg($private_key_path)
. ' ' . escapeshellarg($sdkappid)
. ' ' . escapeshellarg($identifier);
$ret = exec($command, $out, $status);
if ($status == -1)
{
return null;
}
return $out;
}
```

PHP原生接口

在源码包和二进制包中都带有 `php/TLSSig.php` 文件，生成 sig 接口 genSig 和校验 sig 接口 verifySig 均在其中，注意 PHP 环境需要带 openssl 扩展，否则接口使用会报错，另外只支持 PHP 5.3及以上的版本。

> **说明：**
>
> 如果上述实现 PHP 环境无法满足要求，如使用了红帽系（fedora、centos 和 rel 等）的操作系统，可以参考腾讯论坛 中另一种与 openssl 和系统无关的实现。

Windows 平台

工具使用

> **注意：**

这里讲解的是工具的使用说明，实际应用中需要开发者后台调用 TLS 的后台 API 接口生成 sig。

Windows 下生成 sig 和校验 sig

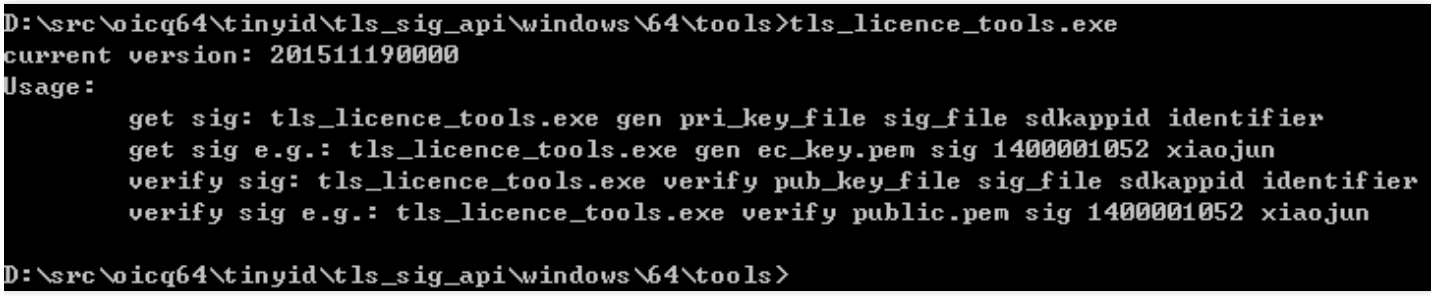
首先不带参数执行 tls_licence_tools.exe ，即执行下面的命令：

```
tls_licence_tools.exe
```

输出：

```
current version: 201511190000
Usage:
get sig: tls_licence_tools.exe gen pri_key_file sig_file sdkappid identifier
get sig e.g.: tls_licence_tools.exe gen ec_key.pem sig 1400001052 xiaojun
verify sig: tls_licence_tools.exe verify pub_key_file sig_file sdkappid identifier
verify sig e.g.: tls_licence_tools.exe verify public.pem sig 1400001052 xiaojun
```

输出实际上是参数模板和示例。下面是演示截图：



执行类似于下面的命令可以生成 sig：

```
tls_licence_tools.exe gen ec_key.pem sig 1400001052 xiaojun
```

对应的参数解释是：

```
tls_licence_tools gen 私钥文件路径 sig 将要存放的路径 sdkappid 用户id
```

执行类似于下面的命令可以校验 sig：

```
tls_licence_tools.exe verify public.pem sig 1400001052 xiaojun
```

对应参数的解释是：

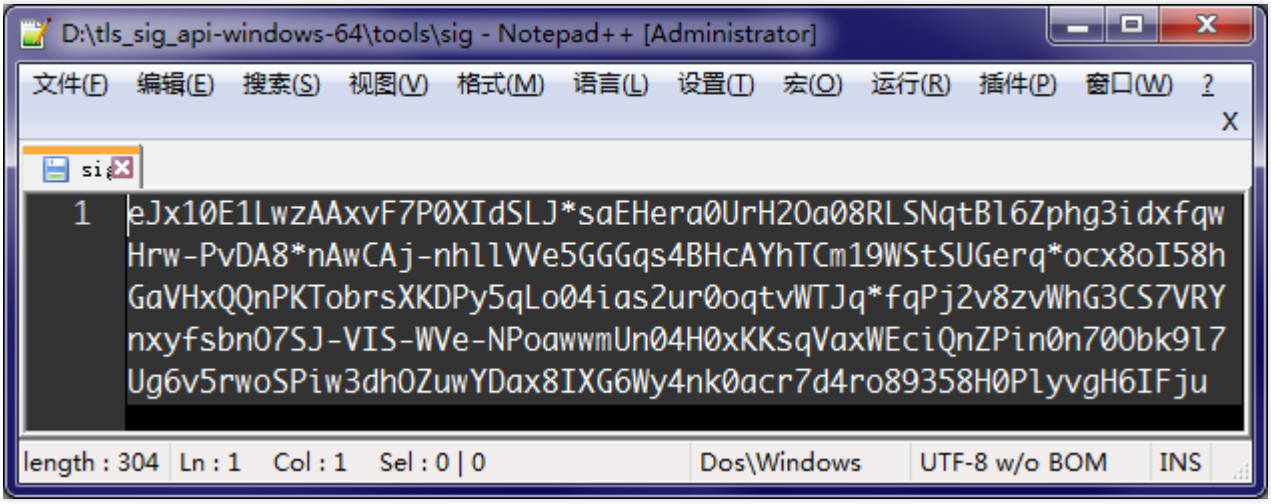
```
tls_licence_tools verify 公钥文件路径 sig的存放路径 sdkappid 用户id
```

下面演示截图：

```
D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>tls_licence_tools.exe gen ec_key.pem sig 14000010
generate sig ok

D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>
```

sig 文件的内容如下图：



校验 sig 演示截图：

```
D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>tls_licence_tools.exe verify public.pem sig 14000
verify sig ok

D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>
```

下面解释下参数模板中参数的意义：

> ⚠ **注意：**
>
> 生成的 sig 有效期为 180 天，开发者需要在 sig 过期前，重新生成 sig。

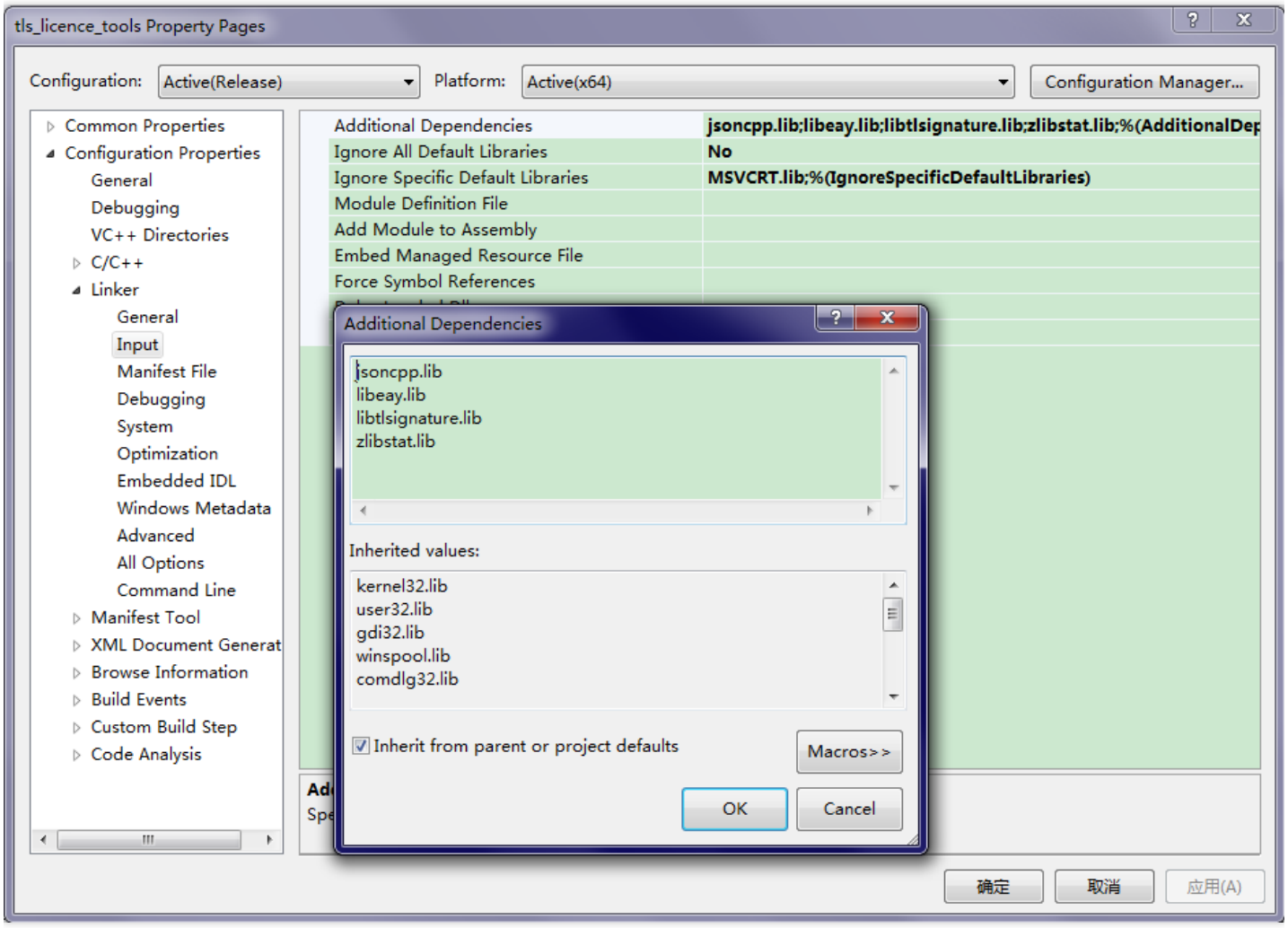
gen 和 verify:分别表示生成 **sig** 和校验 **sig** 的命令
pri_key_file：私钥文件的路径
pub_key_file：公钥文件的路径
sig_file：**sig** 文件的路径，如果是生成 **sig**，那么会将 **sig** 写入这个文件，如果是校验 **sig**，那么会从这个文件读取 **sig** 的内容
sdkappid：创建应用时页面上分配的 sdkappid
identifier：用户标识，即用户 id

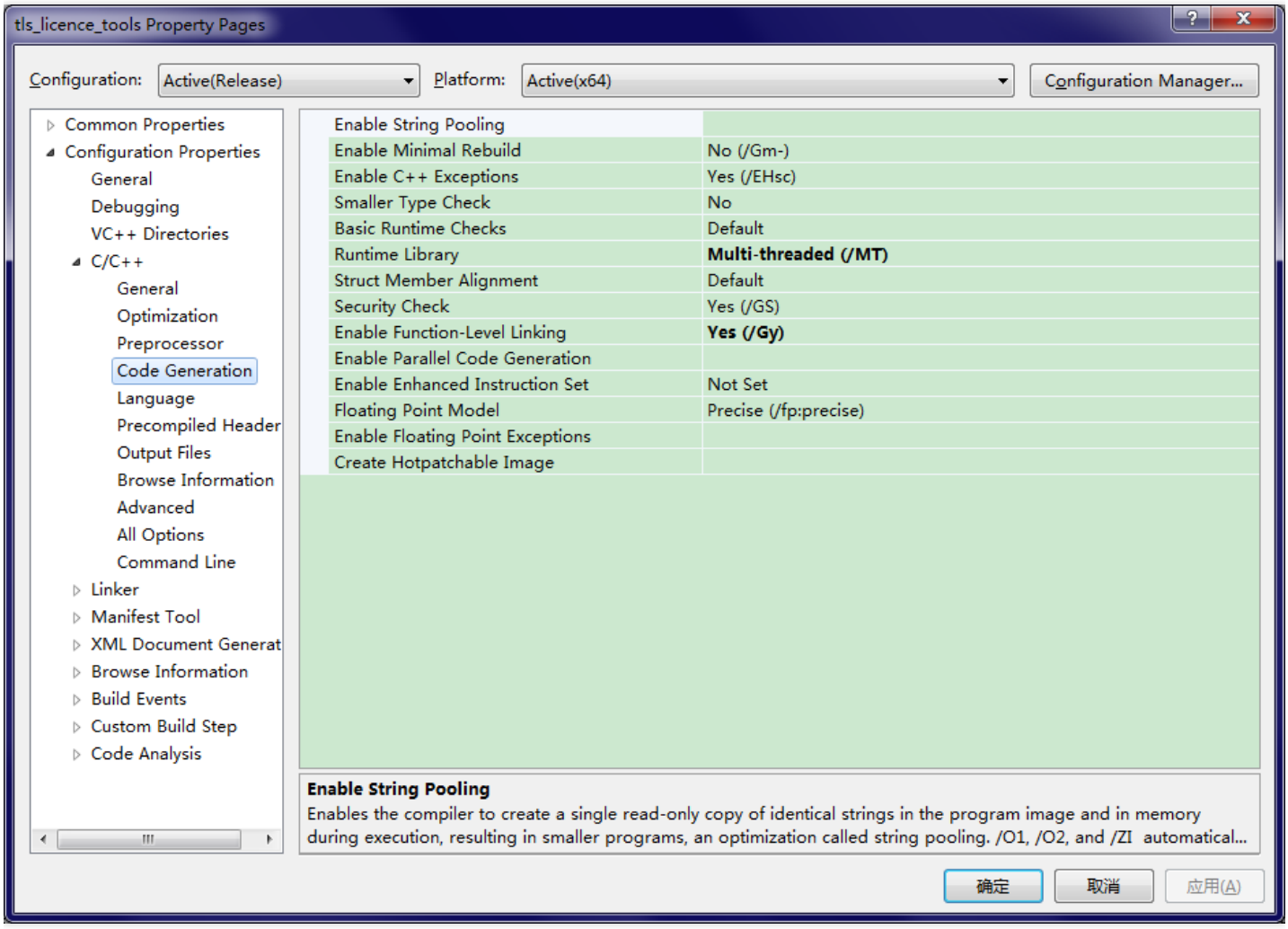
C++ 接口

Windows 下 C++ 接口的使用方式我们采用 vs2012 来举例。首先包含 `include\tls_sig_api` 目录下的 `tls_signature.h` 。头文件中包含的接口， `tls_gen_signature_ex2` 和 `tls_check_signature_ex2` ，前者是生成 sig 的接口，后者是校验 sig 的接口，详细的参数和返回值说明请参考头文件 `tls_signature.h` 。
然后是链接静态库，在 `lib` 目录下有下列目录：

```
├── jni
├── jsoncpp
├── libsigcheck
├── openssl
├── tls_sig_api
└── zlib
```

需要链接的静态库是 `jsoncpp.lib` 、 `openssl` 目录下的 `libeay.lib` 、 `libtlsignature.lib` 和 `zlib` 目录下的 `zlibstat.lib` ，典型的编译配置如下：


tls_licence_tools Property Pages
Configuration: Active(Release)
Platform: Active(x64)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Manifest Tool
XML Document Generat
Browse Information
Build Events
Custom Build Step
Code Analysis
Additional Dependencies
jsoncpp.lib;libeay.lib;libtlsignature.lib;zlibstat.lib;%(AdditionalDep
Ignore All Default Libraries
No
Ignore Specific Default Libraries
MSVCRT.lib;%(IgnoreSpecificDefaultLibraries)
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Additional Dependencies
jsoncpp.lib
libeay.lib
libtlsignature.lib
zlibstat.lib
Inherited values:
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
Inherit from parent or project defaults
Macros>>
OK
Cancel
确定
取消
应用(A)



> ⚠ **注意：**
>
> 如果程序是多线程调用 TLS 后台 API，请在程序初始化时和结束时分别调用下面的接口：

```
int multi_thread_setup(void);
void multi_thread_cleanup(void);
```

Java 接口

目前 Java 接口使用 JNI 的方式实现。Java 目录下 tls_sigcheck.class，是由 tls_sigcheck.java 编译得到的，如果有 JDK 兼容性问题，开发者可自行重新编译此文件，编译命令为：

```
javac -encoding utf-8 tls_sigcheck.java
```

请注意接口的包路径为 `com.tls.sigcheck` ，典型的使用方法是 `example` 目录下 Java 版本 Demo 的组织方式：

```
├── com
│   └── tls
│   └── sigcheck
│   └── tls_sigcheck.class
├── Demo.class
├── Demo.java
├── ec_key.pem
├── public.pem
└── README
```

之前提到 Java 接口使用的 JNI 的方式，所以 Demo.java 调用了载入 dll 的语句，开发者根据自己的存放 jnisigcheck.dll 实际路径进行修改，预编译的 jnisigcheck.dll 存放在 lib\jni 目录下。Demo 的使用方式请参考 example\java\README 。下面是演示截图：

```
6  // 使用的编译命令是
7  // javac -encoding utf-8 Demo.java
8  // 使用的运行命令是
9  // java Demo
10
11 public class Demo {
12
13     public static void main(String args[]) throws Exception {
14
15         tls_sigcheck demo = new tls_sigcheck();
16
17         // 使用前请修改动态库的加载路径
18         demo.loadJniLib("D:\\tls_sig_api-windows-64\\lib\\jni\\jnisigcheck.dll");
19
20         File priKeyFile = new File("ec_key.pem");
21         StringBuilder strBuilder = new StringBuilder();
22         String s = "";
```

demo 源代码中载入 jni dll

下面是运行结果：

```
D:\src\oicq64\tinyid\tls_sig_api\example\java>java Demo
sig:
eJxlj0FPgzAAhe-8CtKrxrXQzmHiYc6ZoQPnxM1wIQilKYttLcV1Mf53M1xiE9-1*5L33pfn*z7Ils8XZVXJXpjCHBQF-pUPIDj-g0rxuihNEer6H6RWcU2LsjFUDzAg
fo610xlAwcYXhMML50Fc4GmMxfZvHTbVbZ-IPtEx0J3ahosU3XkqHdlqj5Z1puDnWXv95bdtOG*5il49UDZJKIVGRi2uVGxbMWnz0mbytM13cLmU76kV2OpKiunUrD3*
EUEIRcfZwPv2fgBww17Q
--
verify ok -- expire time 2592000 -- init time 1448544453
```

⚠ 注意：

如果在 Java 代码中使用了多线程的方式生成 usersig ，可以参考腾讯云论坛 中的相关介绍。

Java 原生接口

Java 原生接口依赖于 5 个 jar 包。在 `tls_sig_api/java_native/lib` 目录下：

```
├── bcpkix-jdk15on-152.jar
├── bcprov-jdk15on-152.jar
├── commons-codec-1.10.jar
├── gson-2.3.1.jar
├── json.jar
└── tls_signature.jar
```

> ⚠ **注意：**
>
> 从控制台界面 下载 的公私钥，将公钥内容赋值给接口中的 `publicBase64Key` 参数，私钥内容赋值给接口中的 `privateBase64Key` 参数。

C# 接口

以非托管的方式调用 dll 实现，调用的 dll 为 `lib\libsigcheck\sigcheck.dll` ，C 样式接口的参数与返回值说明参见 `include\sigcheck.h` 头文件，接口的转换方式如下：

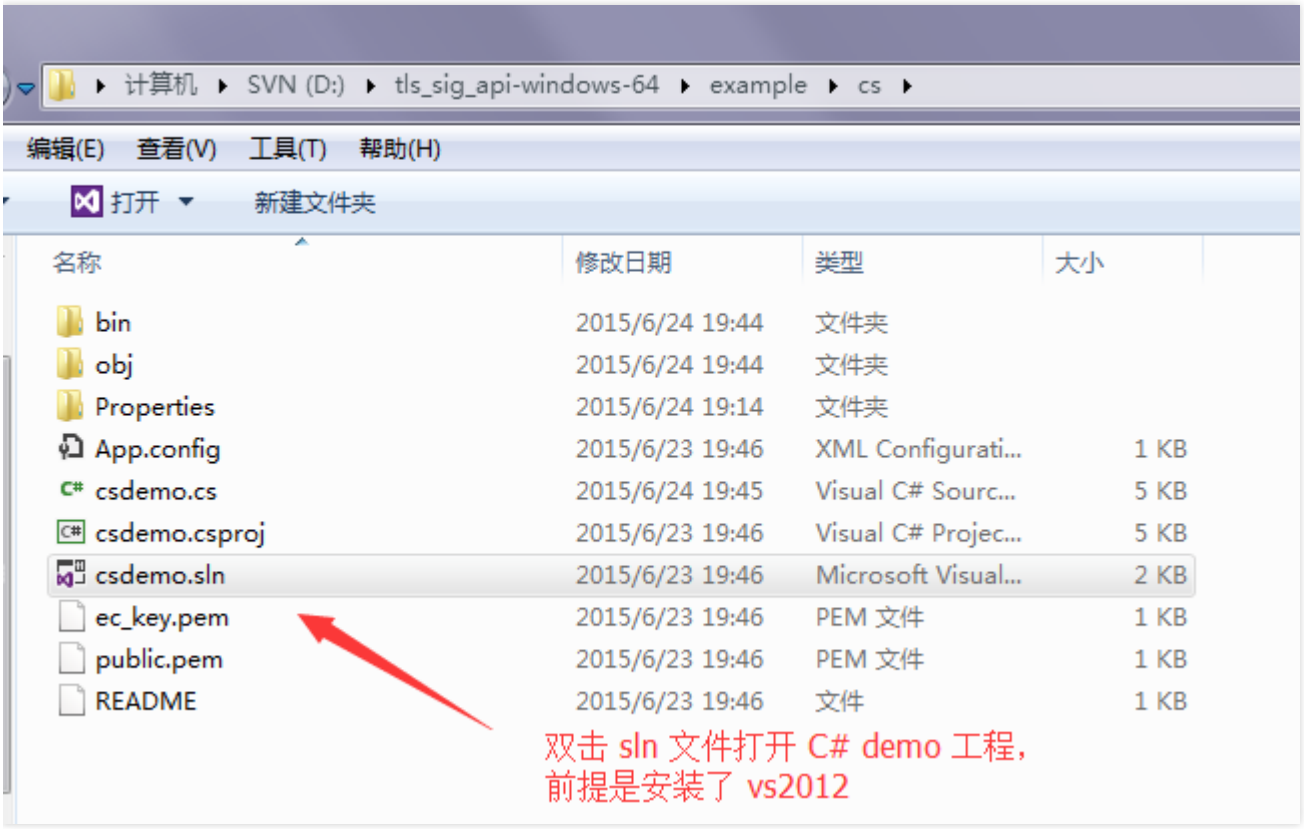
```
class sigcheck
{
[DllImport(dllpath.DllPath, EntryPoint = "tls_gen_sig_ex", CharSet = CharSet.Ansi, CallingConventi
on = CallingConvention.Cdecl)]
public extern static int tls_gen_sig_ex(
UInt32 sdkappid,
string identifier,
StringBuilder sig,
UInt32 sig_buff_len,
string pri_key,
UInt32 pri_key_len,
StringBuilder err_msg,
UInt32 err_msg_buff_len
);

[DllImport(dllpath.DllPath, EntryPoint = "tls_vri_sig_ex", CharSet = CharSet.Ansi, CallingConventi
on = CallingConvention.Cdecl)]
public extern static int tls_vri_sig_ex(
string sig,
string pub_key,
UInt32 pub_key_len,
UInt32 sdkappid,
string identifier,
```

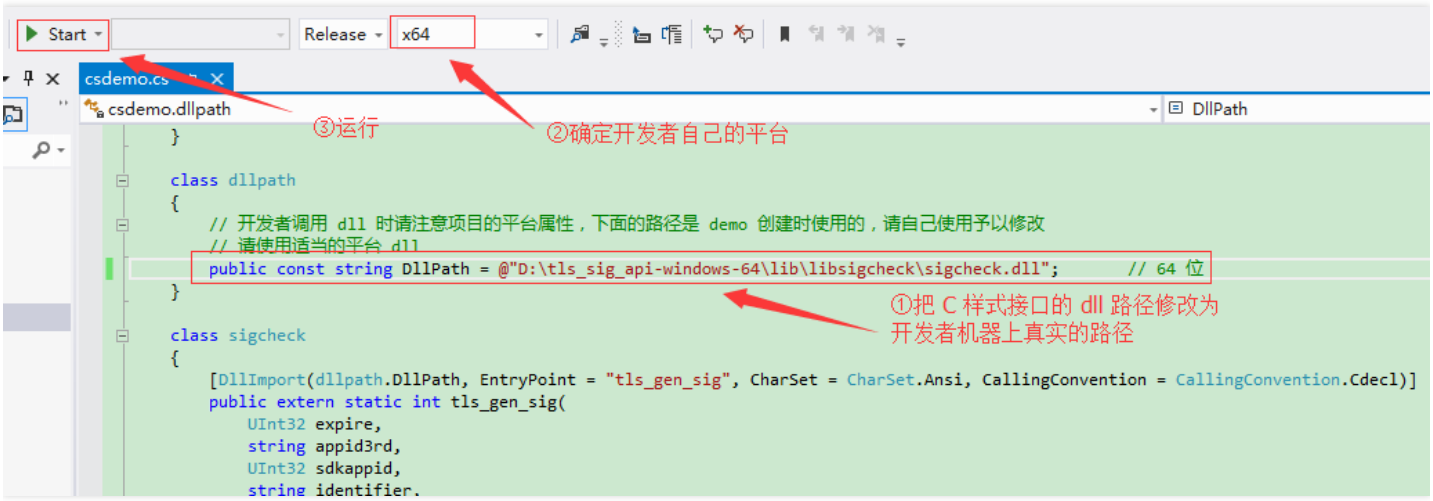
```
    ref UInt32 expire_time,
    ref UInt32 init_time,
    StringBuilder err_msg,
    UInt32 err_msg_buff_len
    );
    }
```

其中 dllpath.DllPath 指明了 dll 的路径，详细请参见 example\cs\csdemo.cs 。关于 Demo 的使用方法参见 example\cs\README 。下面是演示截图：

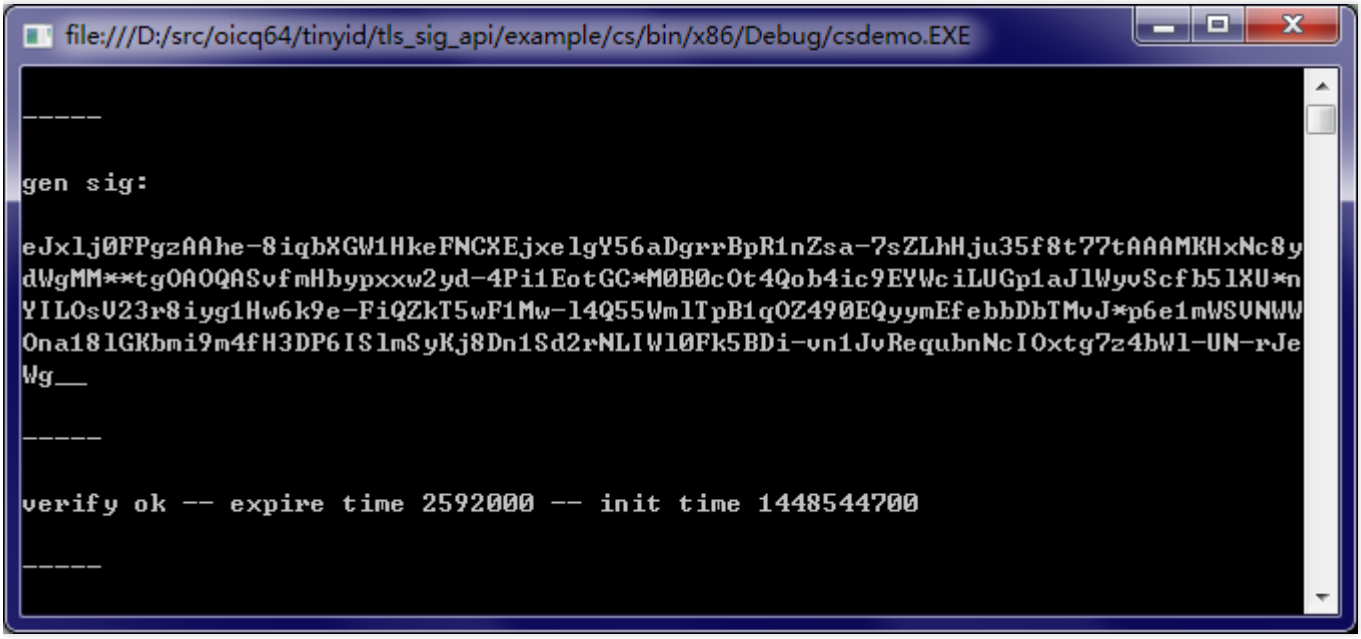
> ⚠ **注意：**
>
> 如果选择 Any CPU 平台，请默认加载 32 位 dll。


计算机 ▸ SVN (D:) ▸ tls_sig_api-windows-64 ▸ example ▸ cs ▸
编辑(E) 查看(V) 工具(T) 帮助(H)
打开 新建文件夹
名称 修改日期 类型 大小
bin 2015/6/24 19:44 文件夹
obj 2015/6/24 19:44 文件夹
Properties 2015/6/24 19:14 文件夹
App.config 2015/6/23 19:46 XML Configurati... 1 KB
csdemo.cs 2015/6/24 19:45 Visual C# Sourc... 5 KB
csdemo.csproj 2015/6/23 19:46 Visual C# Projec... 5 KB
csdemo.sln 2015/6/23 19:46 Microsoft Visual... 2 KB
ec_key.pem 2015/6/23 19:46 PEM 文件 1 KB
public.pem 2015/6/23 19:46 PEM 文件 1 KB
README 2015/6/23 19:46 文件 1 KB
双击 sln 文件打开 C# demo 工程，
前提是安装了 vs2012


using System.Runtime.InteropServices;
// 此文件为 C# demo，使用的非托管模式的 C++ 接口
// 项目中使用 C# 调用 dllsigcheck.dll 时请注意平台，64 位和 32 位平台的 dll 我们均有预编译
namespace csdemo
{
class Program
{
// 下面的公私钥路径都是绝对路径，请开发者自行修改
const string pri_key_path = @"D:\tls_sig_api-windows-64\example\cs\ec_key.pem";
const string pub_key_path = @"D:\tls_sig_api-windows-64\example\cs\public.pem";
static void Main(string[] args)
{
// 生成 sig 文件
FileStream f = new FileStream(pri_key_path, FileMode.Open, FileAccess.Read);
BinaryReader reader = new BinaryReader(f);
byte[] b = new byte[f.Length];
reader.Read(b, 0, b.Length);
string pri_key = Encoding.Default.GetString(b);
修改公私钥路径



下面是运行结果：



PHP 接口

PHP 实现的方式较为简单，就是调用命令行工具生成 sig，工具是 `bin\signature.exe` ，PHP 的调用方式如下：

> **说明：**
>
> 开发者请注意命令执行的路径，如果出现问题请尝试打印出 `command` 变量的内容进行定位。

```
function signature($identifier, $sdkappid, $private_key_path)
{
# 这里需要写绝对路径，开发者根据自己的路径进行调整
```

```
$command = 'D:\\signature.exe'
. ' ' . escapeshellarg($private_key_path)
. ' ' . escapeshellarg($sdkappid)
. ' ' . escapeshellarg($identifier);
$ret = exec($command, $out, $status);
if ($status == -1)
{
return null;
}
return $out;
}
```

PHP 原生接口

在源码包和二进制包中都带有 `php/TLSSig.php` 文件，生成 sig 接口 `genSig` 和校验 sig 接口 `verifySig` 均在其中，注意 PHP 环境需要带 openssl 扩展，否则接口使用会报错，另外只支持 PHP 5.3 及以上的版本。

说明：

若上述实现 PHP 环境无法满足要求，如使用了红帽系（fedora、centos 和 rel 等）的操作系统，可以参考腾讯云论坛 另一种与 openssl 和系统无关的实现。

其他平台接口

- JavaScript
- Python
- Go
- Java
- PHP

TLS 后台 API 下载

下载 TLS 后台 API 。

联系我们

腾讯云论坛 的一些信息可能对您有帮助，如需支持，请 @TLS 帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

秘钥和签名相关

最近更新时间：2019-05-15 09:37:04

下载公私钥

进入 互动直播控制台 ，单击【应用列表】>【App基础设置】。

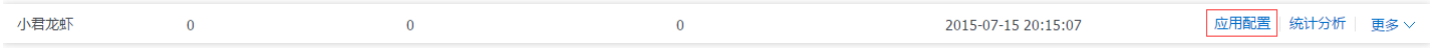
在【帐号体系集成】项目下面单击【下载公私钥】，即可获取到系统生成的公私钥。

设置 App 管理员

仅集成自有帐号，才能设置管理员，否则设置管理员没有意义。在输入框中填写管理员帐号，如果有多个请添加多行。

添加短信签名

打开应用列表，单击相应应用的【应用配置】。

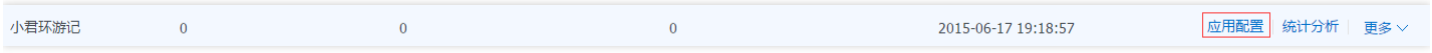


在【帐号体系集成】项目后面单击【编辑】。

在【验证短信签名】后面的文本框内按规范填写短信签名即可。

上传公钥

打开应用列表，单击相应应用的【应用配置】。

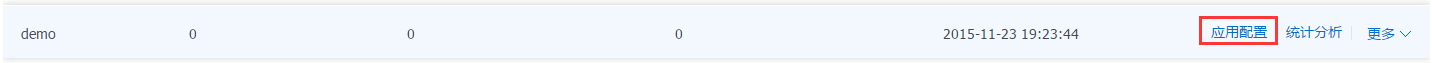


在【帐号体系集成】项目后面单击【编辑】。

然后单击【上传公钥】即可。

下载公钥

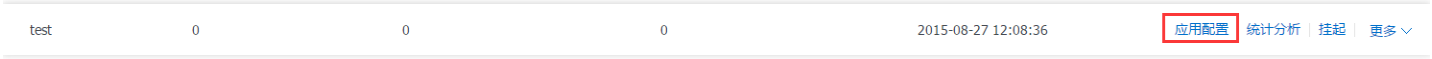
打开应用列表，单击相应应用的【应用配置】。



在【帐号体系集成】项目下面单击【下载公钥】，即可获取到系统生成的公钥。

下载 UserSig

打开应用列表，单击相应应用的【应用配置】。



在【帐号体系集成】项目下面的【下载用户凭证】栏单击【下载】，即可获取到系统生成的 UserSig。

帐号体系集成 编辑

集成自有帐号体系

帐号名称	test
accountType	940
集成模式	托管模式
验证方式	系统生成公私钥 什么是公私钥 下载公钥 私钥由腾讯云保存，公钥由开发者保存。腾讯云使用私钥生成用户签名UserSig，开发者可以使用公钥对签名UserSig进行校验。
账号管理员	test 什么是账号管理员
下载用户凭证	下载 什么是用户凭证
主域名	什么是主域名
验证短信签名	hello 什么是验证短信签名

DC、OC 机房的分配

最近更新时间：2019-05-10 14:42:30

相关概念说明

- **DC（Data Center）：**核心机房，用于互动直播业务中需要上行音视频数据或实时交互的用户角色（如主播、讲师、参与实时互动的角色等）接入。
- **OC（Outer Center）：**边缘节点，用于互动直播业务中不需要上行音视频数据、仅观看的用户角色（如普通观众、不需要与老师互动的学生等）接入。

二者的计费价格是不同的。关于价格详情和分配策略参见 基础网络费用。

关于 DC 和 OC 的分配原则

对于一个 App 的用户来说，什么情况下会接入 DC、什么情况下会接入 OC 呢？

代码层面需要关心的分配原则简单来说只有一句话：有上行音视频数据权限的实例会分配 DC、没有上行音视频数据权限的实例分配 OC。具体地，在调用 SDK 进入房间接口 `ILiveRoomManager.getInstance().createRoom()` 的时候，其参数 `ILiveRoomOption.authBits()` 用于设置该实例在房间内的权限，具体权限字段如下：

字段	值	说明
AUTH_BITS_DEFUALT	0xFFFFFFFFFFFFFFFF	缺省值。拥有所有权限。
AUTH_BITS_CREATE_ROOM	0x00000001	创建房间权限。
AUTH_BITS_JOIN_ROOM	0x00000002	加入房间的权限。
AUTH_BITS_SEND_AUDIO	0x00000004	发送语音的权限。
AUTH_BITS_RECV_AUDIO	0x00000008	接收语音的权限。
AUTH_BITS_SEND_VIDEO	0x00000010	发送视频的权限。
AUTH_BITS_RECV_VIDEO	0x00000020	接收视频的权限。
AUTH_BITS_SEND_SUB	0x00000040	发送辅路视频的权限，暂不支持辅路。
AUTH_BITS_RECV_SUB	0x00000080	接收辅路视频的权限，暂不支持辅路。

AVRoomMulti.auth_bits 成员变量是权限位的明文形式。AVRoomMulti.auth_bits 将 `AUTH_BITS_SEND_AUDIO` / `AUTH_BITS_SEND_VEDIO` / `AUTH_BITS_SEND_SUB` 中的任意一个置为 1，则实例会被分配接入 DC；反之则该实例被分配接入 OC。

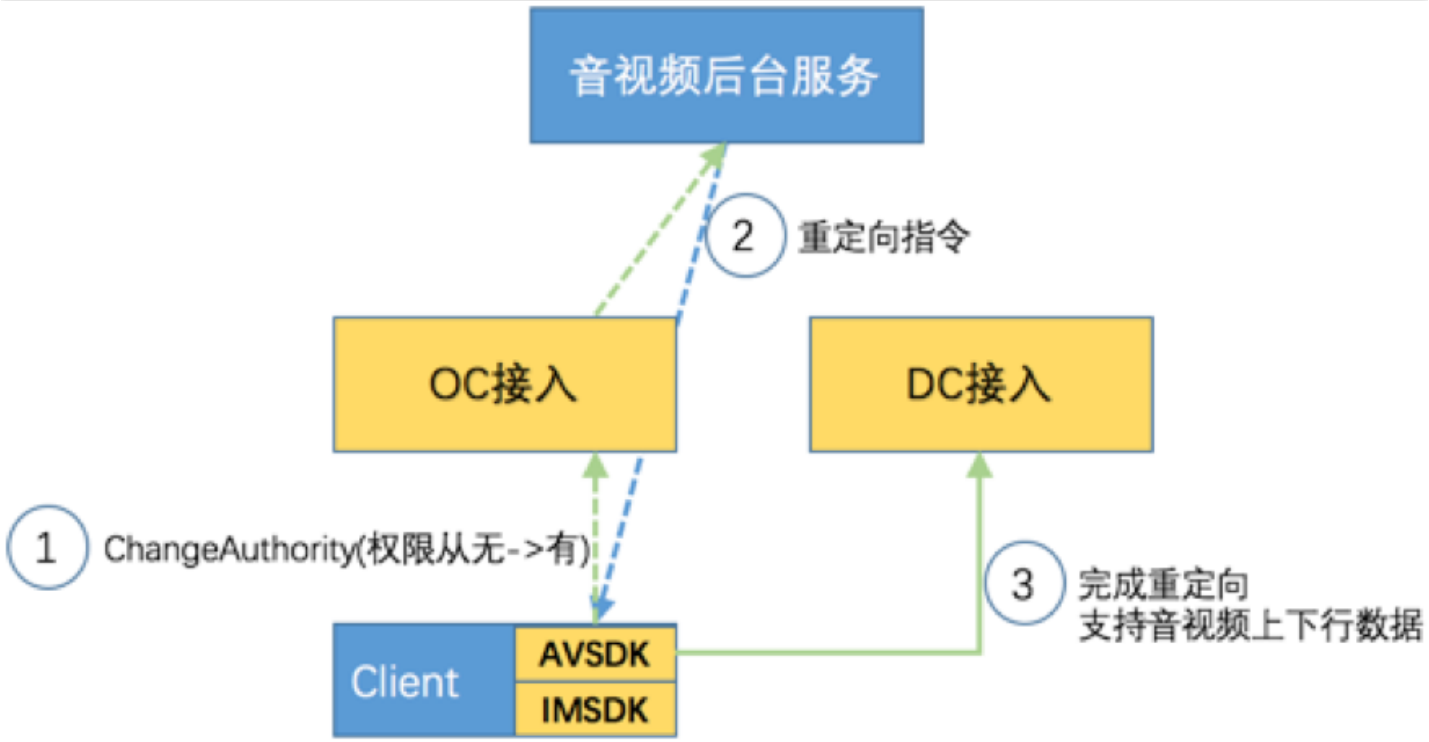
说明：

后台对单个房间接入 DC 的用户数量有一个上限保护。例如，如果某个业务设置每个用户都有上行权限，那么后台对于每一个房间前1000个用户分配 DC，其他用户分配 OC。该限制为后台保护策略，后面可以根据需要进行调整。

关于 DC/OC 之间的切换策略

当用户通过进入房间的 `ILiveRoomManager.getInstance().createRoom()` 流程被分配到 DC/OC 之后，有时也会有需要在 DC/OC 之间进行切换的需求。例如，教育场景下，一个原本没有上行权限的学生（被分配接入 OC）被老师点中回答问题时，需要从不能上行音视频数据 OC 切换到 DC。DC/OC 的切换依然是以权限变化为依据的，相对应的接口为 ILiveRoomManager 类中 changeAuthAndRole 接口。下面将分几种情况来分别讨论：

- 用户位于 DC，权限从有到无（将 `AUTH_BITS_SEND_AUDIO` / `AUTH_BITS_SEND_VEDIO` / `AUTH_BITS_SEND_SUB` 全设置为0），在此情况下，因为 OC 比 DC 有时延，为了避免学生切回 OC 后看到之前的画面，学生依然会留在 DC 直到退出房间。
- 用户位于 DC，权限从无到有：不存在这种情况。
- 用户位于 OC，权限从有到无：在此情况下 SDK 不会有任何动作。
- 用户位于 OC，权限从无到有（将 `AUTH_BITS_SEND_AUDIO` / `AUTH_BITS_SEND_VEDIO` / `AUTH_BITS_SEND_SUB` 其中一个置为非0），在此情况下，音视频后台会下发重定向指令，将终端实例重定向到 DC。


音视频后台服务
2
重定向指令
OC接入
DC接入
1
ChangeAuthority(权限从无->有)
3
完成重定向
支持音视频上下行数据
Client
AVSDK
IMSDK

角色

最近更新时间：2019-03-11 10:11:29

互动直播引入角色的概念，用于实现在单一平台上也可以配置不同的参数。可以将角色理解为终端进入房间的配置集。

配置角色

用户可以在 互动直播控制台 单击【应用列表】>【SPEAR引擎配置】，配置角色。

定制角色

用户可以根据自己的需求定制自己的角色。

使用角色

用户可以在进房间的 option 中配置要使用的角色。

Android：

```
ILVLiveRoomOption hostOption = new ILVLiveRoomOption(hostId)
.controlRole("LiveMaster"); // 使用 LiveMaster 角色
```

iOS：

```
//TILLiveSDK ( 直播 SDK )
TILLiveRoomOption *hostOption = [TILLiveRoomOption defaultHostLiveOption];
hostOption.controlRole = @"LiveMaster"; // 使用 LiveMaster 角色

//TILCallSDK ( 通话 SDK )
TILCallSponsorConfig *sponsorConfig = [[TILCallSponsorConfig alloc] init];
sponsorConfig.controlRole = @"LiveMaster"; // 使用 LiveMaster 角色
```

MacOS :

```
ILiveRoomOption *option = [ILiveRoomOption defaultHostLiveOption];
option.controlRole = @"LiveMaster";//角色字符串来自腾讯云控制台 spear 配置
[[ILiveRoomManager getInstance] createRoom:(int)_item.info.roomnum option:option succ:^{
NSLog(@"create room succ");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"createRoom fail,module=%@,code=%d,msg=%@",module,errId,errMsg);
}];
```

PC :

```
iLiveRoomOption roomOption;
roomOption.controlRole = "LiveMaster"; // 使用 LiveMaster 角色
```

IE :

```
sdk.createRoom(roomid, "LiveMaster", // 使用 LiveMaster 角色
function () {
}, function (errMsg) {
}, false);
```

切换角色

用户在进进入房间后，仍然可以根据需求调整角色。

Android :

```
// 切换角色为 LiveGuest
ILiveRoomManager.getInstance().changeRole("LiveGuest", new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
//...
```

```
}

@Override
public void onError(String module, int errCode, String errMsg) {
//...
}
});
```

iOS：

```
// 切换角色为 LiveGuest
ILiveRoomManager *manager = [ILiveRoomManager getInstance];
[manager changeRole:@"LiveGuest" succ:^ {
NSLog(@"角色改变成功");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"角色改变失败");
}];
```

MacOS：

```
// 切换角色为 LiveGuest
NSString *role = @"LiveGuest";
[[ILiveRoomAVManager getInstance] changeRole:role succ:^{
NSLog(@"change role succ");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"change role fail");
}];
```

PC：

```
// 切换角色为 LiveGuest
void Live::OnChangeRoleSuc( void* data )
{
//切换角色成功
}
void Live::OnChangeRoleErr( int code, const char *desc, void* data )
{
//切换角色失败
}
GetILive()->changeRole("LiveGuest", OnChangeRoleSuc, OnChangeRoleErr, NULL);
```

IE：

```
// 切换角色为 LiveGuest
sdk.changeRole("LiveGuest", function() {
```

```
//切换角色成功
},
function(err){
//切换角色失败
}
);
```

角色的高阶应用

根据网络状态调整动态当前的视频质量：用户可以在腾讯云针对主播，观众分别配置多个角色(高清，标清，流畅)，用户可以检测当前的网络状态，动态调整当前使用的角色，以达到动态修改视频质量的效果。

> ⚠ **注意：**
>
> 可以参照 随心播。