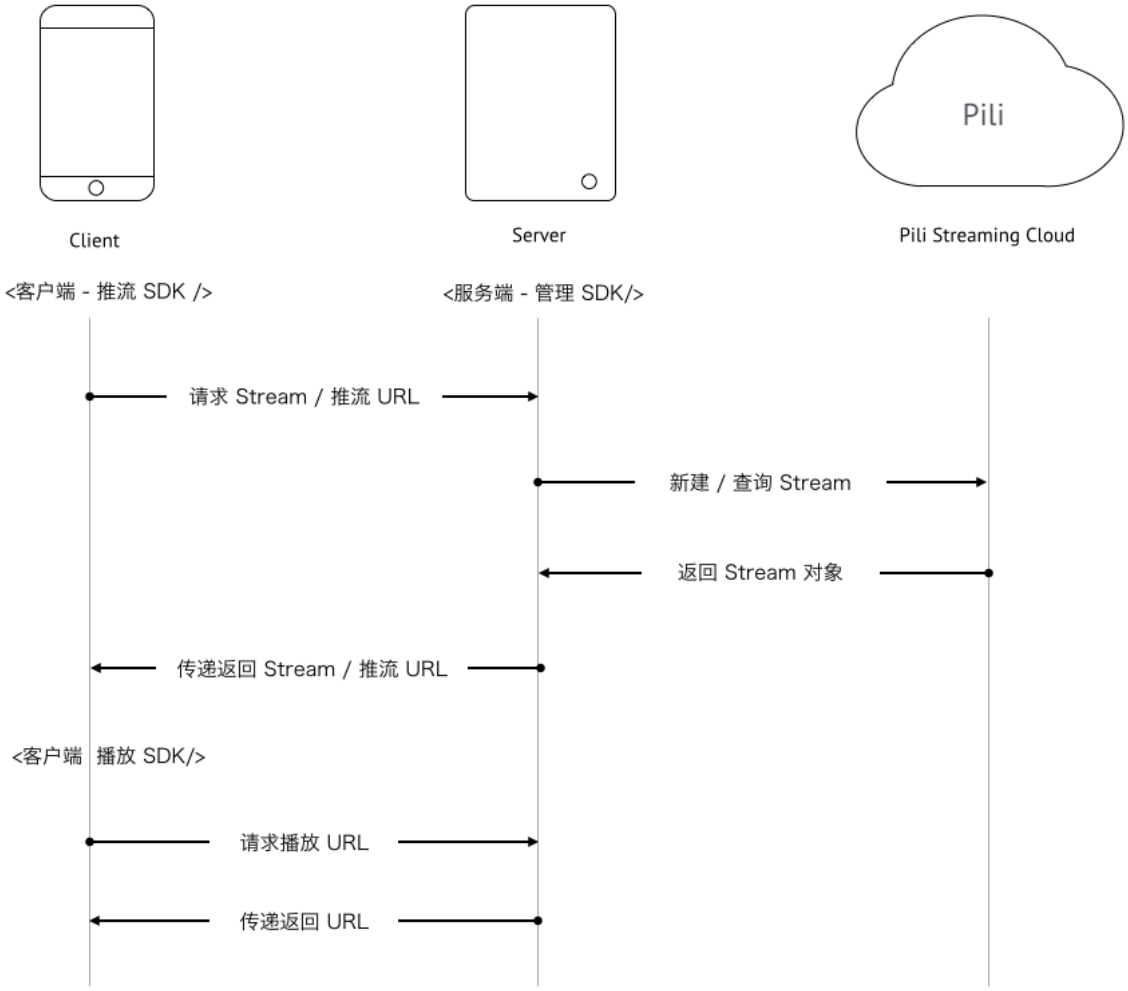
Pili Streaming Cloud 接入指南

直播工作流模型

1. Client (iOS/Android/PC/Camera) 向 Server (业务逻辑服务器) 请求推流授权
2. Server 颁发带授权信息的 Stream 给 Client
3. Client 通过 RTMP 推流 给 Pili Streaming Cloud
4. Client 向 Server 请求播放授权
5. Server 向 Client 颁发播放地址
6. Client 调用 播放器 SDK 打开播放地址进行播放

三方交互的业务逻辑可以借鉴如下示意图进行理解：



服务端 SDK 功能介绍

目前提供的服务端 SDK 有：Go, NodeJS, Ruby, Python, PHP, Java

- https://github.com/pili-engineering/pili-sdk-go
- https://github.com/pili-engineering/pili-sdk-nodejs
- https://github.com/pili-engineering/pili-sdk-ruby
- https://github.com/pili-engineering/pili-sdk-python
- https://github.com/pili-engineering/pili-sdk-php
- https://github.com/pili-engineering/pili-sdk-java

服务端 SDK 用于对直播流（Live Streaming）进行增删查改等管理，以及颁发带授权凭证的 RTMP 推流地址，RTMP/HTTP-FLV/HLS 的播放地址给客户端。服务端 SDK 功能包括但不限于：

```
- hub.create_stream()         #创建流
- hub.get_stream()            #查询流
- hub.list_streams()          #流列表
- stream.update()             #更新流
- stream.delete()             #删除流
- stream.disable()            #禁止流（禁推、禁播）
- stream.enable()             #激活流
- stream.rtmp_publish_url()   #生成带授权凭证的 RTMP 推流地址
- stream.rtmp_live_urls()     #生成 RTMP 直播播放地址
- stream.hls_live_urls()      #生成 HLS 直播播放地址
- stream.http_flv_live_urls() #生成 HTTP-FLV 直播播放地址
- stream.hls_playback_urls()  #生成 HLS 回看在线点播地址
- stream.status()             #查询主播推流实时状态信息
- stream.segments()           #查询直播推流和断流的详细记录
- stream.snapshot()           #直播抽帧截图
- stream.save_as()            #直播录制转码（例如转存mp4）
- stream.to_json()            #将 Stream 对象转成 JSON 下发给客户端 SDK
```

服务端 SDK 使用样例

拿 PHP SDK 举例，接入步骤如下：

第1步，集成SDK

```
// SDK 下载地址：https://github.com/pili-engineering/pili-sdk-php
require_once '/path/to/pili-sdk-php/lib/Pili.php';

// 配置企业开发者密钥
// 密钥使用七牛账号登录 https://portal.qiniu.com/setting/key 获取
define('ACCESS_KEY', 'Your_Qiniu_AccessKey');
define('SECRET_KEY', 'Your_Qiniu_SecretKey');

// 已创建好的直播应用名字，账号必须先开通直播权限才可使用
// 如未开通，发送邮件至 pili@qiniu.com 告知您的七牛账号和直播场景和业务需求
define('HUB', 'Your_Pili_Hub_Name');

// 初始化证书授权
$credentials = new \Qiniu\Credentials(ACCESS_KEY, SECRET_KEY);
$hub = new \Pili\Hub($credentials, HUB);
```

第2步，获取 Stream

```
// 首次，新建流
$title = '流名';           // 一般对应主播房间号
$publishKey = NULL;       // 流密钥，用于生成推流鉴权凭证
$publishSecurity = NULL; // 推流鉴权策略，一般为"static"，针对安全要求较高的UGC推流建议用"dynamic"
$stream = $hub->createStream($title, $publishKey, $publishSecurity);
// 之后，可以从已创建的流里边复用
$stream = $hub->getStream($stream["id"]);
```

第3步，获取 rtmp 推流地址

```
// 生成 rtmp 推流地址
$publishUrl = $stream->rtmpPublishUrl();
echo $publishUrl;
// => rtmp://publish.example.com/hubName/流名

// 手机直播，可将 Stream 对象转成 JSON 下发给客户端推流 SDK
$streamJson = $stream->toJSONString();
```

第4步，生成直播播放地址

```
// 生成 rtmp 直播播放地址
$urls = $stream->rtmpLiveUrls();
var_export($urls);
/*
array (
  'ORIGIN' => 'rtmp://live-rtmp.example.com/hubName/流名',
  '720p'   => 'rtmp://live-rtmp.example.com/hubName/流名@720p',
  '480p'   => 'rtmp://live-rtmp.example.com/hubName/流名@480p'
)
*/

// 生成 http-flv 直播播放地址
$urls = $stream->hlsLiveUrls();
var_export($urls);
/*
array (
  'ORIGIN' => 'http://live-hdl.example.com/hubName/流名.flv',
  '720p'   => 'http://live-hdl.example.com/hubName/流名@720p.flv',
  '480p'   => 'http://live-hdl.example.com/hubName/流名@480p.flv'
)
*/

// 生成 hls 直播播放地址
$urls = $stream->hlsLiveUrls();
var_export($urls);
/*
array (
  'ORIGIN' => 'http://live-hls.example.com/hubName/流名.m3u8',
  '720p'   => 'http://live-hls.example.com/hubName/流名@720p.m3u8',
  '480p'   => 'http://live-hls.example.com/hubName/流名@480p.m3u8'
)
*/
```

这样边推流就可以边播放。如果是使用 OBS 桌面直播推流软件推流，假设生成的 `static` 推流 URL如下:

- `rtmp://{domain}/{hub_name}/{stream_title}?key={publishKey}`

OBS 串流地址请按如下格式填写:

- URL: `rtmp://{domain}/{hub_name}`
- Stream: `{stream_title}?key={publishKey}`

第5步, 主播管理

```
// 查询主播实时Qos状态
$result = $stream->status();
var_export($result);
/*
array (
  'addr' => '222.73.202.226:2572',
  'status' => 'connected',
  'bytesPerSecond' => 16870.200000000001,
  'framesPerSecond' =>
  array (
    'audio' => 42.200000000000003,
    'video' => 14.733333333333333,
    'data' => 0.066666666666666666,
  ),
)
*/

// 禁推，禁播
$stream = $stream->disable();

// 重新激活
$stream = $stream->enable();
```

第6步，查询主播推流断流记录

```
$start = NULL;
$end = NULL;
$limit = NULL;
$result = $stream->segments($start, $end, $limit);
var_export($result);
/*
{
    "start": 1440196065,
    "end": 1440196405,
    "duration": 400, // 内容实际长度
    "segments": [
        {
            "start": 1440196065,
            "end": 1440196105
        },
        {
            "start": 1440196205,
            "end": 1440196405
        }
    ]
}
*/
```

第7步，在线回放点播

```
// 生成 hls 回看点播地址
$start     = 1440196065;  // optional, in second, unix timestamp
$end       = 1440196105;  // optional, in second, unix timestamp
// 如果不给出 start 和 end，生成的 url 使用 start=-1&end=-1 拉取所有回放
$urls = $stream->hlsPlaybackUrls($start, $end);
var_export($urls);
/*
array (
  'ORIGIN' => 'http://playback.example.com/hubName/流名.m3u8?start=1440196065&end=1440196105',
  '720p'   => 'http://playback.example.com/hubName/流名@720p.m3u8?start=1440196065&end=1440196105',
  '480p'   => 'http://playback.example.com/hubName/流名@480p.m3u8?start=1440196065&end=1440196105'
)
*/
```

第8步，直播点播抽帧截图

```
$name      = 'imageName.jpg'; // required
$format    = 'jpg';           // required
$time      = 1440196100;      // optional, in second, unix timestamp, default is now.
$notifyUrl = NULL;            // optional, callback server endpoint
$result    = $stream->snapshot($name, $format, $time, $notifyUrl); # => Array
var_export($result);
/*
array (
  'targetUrl' => 'http://static.example.com/snapshots/z1.hubName.55d7a219e3ba5723280000b5/imageName.jpg',
  'persistentId' => 'z1.55d7a6e77823de5a49a8899a',
)
*/
```

第9步，转存直播录像为音视频文件进行下载

```
$name      = 'videoName.mp4'; // required
$format    = 'mp4';           // required
$start     = 1440196065;      // optional, in second, unix timestamp
$end       = 1440196105;      // optional, in second, unix timestamp
$notifyUrl = NULL;            // optional, callback server endpoint
$result    = $stream->saveAs($name, $format, $start, $end, $notifyUrl);
var_export($result);
/*
array (
  'url' => 'http://vod.example.com/recordings/z1.hubName.55d7a219e3ba5723280000b5/videoName.m3u8',
  'targetUrl' => 'http://vod.example.com/recordings/z1.hubName.55d7a219e3ba5723280000b5/videoName.mp4',
  'persistentId' => 'z1.55d7a6e77823de5a49a8899b',
)
*/
```

可以请求 `curl -D GET http://api.qiniu.com/status/get/prefop?id={PersistentId}` 查询转存进度。参考:
http://developer.qiniu.com/docs/v6/api/overview/fop/persistent-fop.html#pfop-status

客户端 SDK

手机直播，拿 iOS 举例，您可以非常方便地集成手机直播 SDK 来赋予 APP 直播功能。App 端除了 UI 的定制外，主要是集成 PLCameraStreamingKit 来直接对接 Pili 直播云。

只需要在 Podfile 中添加

```
pod 'PLCameraStreamingKit'
```

然后

```
$ pod install
```

或者

```
$ pod update
```

就集成完毕了。

之后具体的调用可以参见 PLCameraStreamingKit 的 README 文档

其中核心的类是 `PLCameraStreamingSession`，整个直播推流过程中基本都是对这个类的实例做的操作。

One more thing，`PLCameraStreamingKit` 提供了一种切换推流质量的方法，也就是说，在网络环境变更或者其他任何你认为需要提高或者降低推流质量的情况下，你都可以直接操作做切换，并且会有方法回调告知发送队列的状态。

[iOS] 播放器 SDK

- https://github.com/pili-engineering/PLPlayerKit

简介

> *PLPlayerKit 是一个适用于 iOS 的音视频播放器 SDK，可高度定制化和二次开发，特色是支持 RTMP 和 HLS 直播流媒体播放，并且支持常见音视频文件例如 MP4/M4A 的播放。*

功能特性

1. RTMP 直播流播放
2. HLS 播放
3. 常见音视频文件播放
4. 高可定制
5. 渲染逐行扫描支持
6. 缓存时长可定制
7. 支持本地及线上音视频文件直接播放

[Android] 播放器 SDK

- https://github.com/pili-engineering/PLDroidPlayer

简介

> *PLDroidPlayer 是一个适用于 Android 的音视频播放器 SDK，可高度定制化和二次开发，特色是支持 RTMP 和 HLS 直播流媒体、以及常见音视频文件（如 MP4、M4A ）播放。*

功能特性

1. 基于 ijkplayer (based on ffplay)
2. Android Min API 9
3. 支持 RTMP, HLS 协议
4. 支持 ARMv7a
5. 支持 MediaCodec 硬解码
6. 提供 VideoView 控件
7. 可定制化的 MediaController
8. 支持 seekTo()
9. 支持获取当前播放时长 getDuration()
10. 支持获取当前播放的位置 getCurrentPosition()

11. 支持音量控制 setVolume()
12. 提供如下接口：
    - OnPreparedListener
    - OnCompletionListener
    - OnErrorListener
    - OnInfoListener

[Web] HTML5 播放器样例

- https://github.com/pili-engineering/pili-html5-player-example

> *基于开源播放器 `MediaElement.js` 实现的样例，缺省先检查浏览器是否支持 HTML5 的 `<video>` 标签来播放 HLS，如果浏览器不支持 `<video>` 标签播 HLS，则降格使用 Flash 来兼容播放 HLS。`Video.js` 和 `MediaElement.js` 都有采用开源的 Flashls 插件来支持 HLS 播放。除了 Safari 支持直接播放 HLS 以外，IE、Chrome、Firefox、Opera 等浏览器都支持使用基于 Flashls 插件的播放器（例如 Video.js、MediaElement.js）来播放 HLS。在 iOS平台上，只要 iOS 版本 >= 3.0，都支持 HLS 播放，包括一切嵌入 WebKit 的 App 例如微信朋友圈。在 Android 平台上，4.x 系列的版本，也都支持 HLS 播放，包括一切潜入 WebKit 的 App 也都支持，个别第三方非系统浏览器可能不支持（如能搭配 Flashls 也能支持）。网页播放器选型，推荐 http://flashls.org 所覆盖的播放器清单。*

[iOS] RTMP 直播推流 SDK

- https://github.com/pili-engineering/PLCameraStreamingKit

简介

> *PLCameraStreamingKit 是一个适用于 iOS 的 RTMP 直播推流 SDK，可高度定制化和二次开发。特色是支持 iOS Camera 画面捕获并进行 H.264 硬编码，以及支持 iOS 麦克风音频采样并进行 AAC 硬编码；同时，还根据移动网络环境的多变性，实现了一套可供开发者灵活选择的编码参数集合。借助 PLCameraStreamingKit，开发者可以快速构建一款类似 Meerkat 或 Periscope 的 iOS 直播应用。*

功能特性

1. 硬件编解码
2. 多码率可选
3. H.264 视频编码
4. AAC 音频编码
5. 支持前后摄像头
6. 自动对焦支持
7. 手动调整对焦点支持
8. 闪光灯开关
9. 多分辨率编码支持
10. HeaderDoc 文档支持
11. 内置生成安全的 RTMP 推流地址
12. ARM64 支持
13. 支持 RTMP 协议直播推流

[iOS] AAC 音频 RTMP 直播推流 SDK

- https://github.com/pili-engineering/PLAudioStreamingKit

简介

> *PLAudioStreamingKit 是一个适用于 iOS 的 RTMP 直播推流 SDK，可高度定制化和二次开发。特色是支持 iOS 麦克风音频采样并进行 AAC 硬编码；同时，还支持后台持续推流。*

功能特性

1. 硬件编解码
2. 多码率可选
3. AAC 音频编码
4. HeaderDoc 文档支持
5. 内置生成安全的 RTMP 推流地址
6. ARM64 支持
7. 支持 RTMP 协议直播推流
8. 支持后台推流

[Android] RTMP 直播推流 SDK

- https://github.com/pili-engineering/PLDroidCameraStreaming

简介

> *PLDroidCameraStreaming 是一个适用于 Android 的 RTMP 直播推流 SDK，可高度定制化和二次开发。特色是支持 Android Camera 画面捕获并进行 H.264 硬编码，以及支持 Android 麦克风音频采样并进行 AAC 硬编码；同时，还实现了一套可供开发者选择的编码参数集合，以便灵活调节相应的分辨率和码率。借助 PLDroidCameraStreaming ，开发者可以快速构建一款类似 Meerkat 或 Periscope 的 Android 直播应用。*

功能特性

1. 支持 MediaCodec 硬编码
2. 支持 AAC 音频编码
3. 支持 H264 视频编码
4. 内置生成安全的 RTMP 推流地址
5. 支持 RTMP 协议推流
6. 支持 ARMv7a
7. Android Min API 18
8. 支持前后置摄像头，以及动态切换
9. 支持自动对焦
10. 支持闪光灯操作
11. 支持纯音频推流，以及后台运行

[IPCamera] 视频 H.264 音频 AAC 编码 RTMP 直播推流 SDK

- https://github.com/pili-engineering/pili-camera-sdk

简介

> *pili-camera-sdk 是一个适用于 IPCamera 的 RTMP 直播推流 SDK，开发者可以直接写入 h264 NAL 数据，无需做视频内容的封装，小巧、简洁。使用该 SDK ，可快速开发 Dropcam 类产品。*

功能特性

1. h264 NAL 数据无需开发者封装
2. 低内存占用
3. 简单握手支持

[go-socket.io] Go 语言版本的 Socket.IO 实现

- https://github.com/pili-engineering/go-socket.io

简介

> *go-socket.io 是一个完全由 Golang 编写的高并发 Socket.IO 实现，一个支持 WebSocket 的协议用于实时通信、跨平台的开源框架。Socket.IO 除了支持 WebSocket 通讯协议外，还支持许多种轮询（Polling）机制以及其它实时通信方式，并封装成了通用的接口，并且在服务端实现了这些实时机制的相应代码。Socket.IO 实现的 Polling 通信机制包括 Adobe Flash Socket、AJAX 长轮询、AJAX multipart streaming、持久 Iframe、JSONP 轮询等。Socket.IO 能够根据浏览器对通讯机制的支持情况自动地选择最佳的方式来实现网络实时应用。 常见应用场景例如 弹幕、聊天室。*

安全篇

服务端 SDK 请求 API 交互鉴权算法

如下是一个典型的 HTTP API Request 结构：

```
<Method> <PathWithRawQuery>
<Header1>: <Value1>
<Header2>: <Value2>
...
Host: <Host>
...
Content-Type: <ContentType>
...
Authorization: Qiniu <AK>:<Sign>
...

<Body>
```

对于上面这样一个请求，我们构造如下这个待签名的：

```
<Method> <PathWithRawQuery>
Host: <Host>
Content-Type: <ContentType>

[<Body>] #这里的 <Body> 只有在 <ContentType> 存在且不为 application/octet-stream 时才签进去。
```

有了 <Data>，就可以计算对应的 <Sign>，如下：

```
<Sign> = base64.URLEncoding.EncodeToString(hmac_sha1(<SK>, <Data>))
```

一个正常的待签名的 <Data> 长如下样子：

```
POST /v1/streams
Host: pili.qiniuapi.com
Content-Type: application/json

{
    "title": 'test',
    "hub": 'live',
    "publishKey": 'aaaa-bbbb-cccc-dddd',
    "publishSecurity": 'dynamic'
}
```

如下是一段 Go 语言 API 请求数字签名示例：

```
func incBody(req *http.Request, ctType string) bool {

    return req.Body != nil && ctType != "" && ctType != "application/octet-stream"
}

func (mac *Mac) SignRequest(req *http.Request) (token string, err error) {

    h := hmac.New(sha1.New, []byte(mac.SecretKey))

    u := req.URL
    data := req.Method + " " + u.Path
    if u.RawQuery != "" {
        data += "?" + u.RawQuery
    }
    io.WriteString(h, data+"\nHost: "+req.Host)

    ctType := req.Header.Get("Content-Type")
    if ctType != "" {
        io.WriteString(h, "\nContent-Type: "+ctType)
    }
    io.WriteString(h, "\n\n")

    if incBody(req, ctType) {
        s2, err2 := seekable.New(req)
        if err2 != nil {
            return "", err2
        }
        h.Write(s2.Bytes())
    }

    sign := base64.URLEncoding.EncodeToString(h.Sum(nil))
    token = mac.AccessKey + ":" + sign
    return
}
```

代码出处：https://github.com/pili-engineering/pili-sdk-go/blob/master/pili/auth.go

RTMP 推流地址鉴权算法说明

RTMP Publish URL 有两种格式，一种是 `dynamic`，另一种是 `static`，由 Stream 对象的 `publishSecurity` 字段控制。

当 `stream.publishSecurity="dynamic"` 时，推流地址格式如下：

```
rtmp://{rtmp_publish_host}/{hub_name}/{stream_title}?nonce={nonce++}&token={publishToken}

publishToken = hmac_sha1(secret, data)
secret = stream.publishKey
data = rtmp://{rtmp_publish_host}/{hub_name}/{stream_title}?nonce={nonce++}
```

`dynamic` 格式的推流地址一次性有效，每次连接都需要递增 `nonce` 参数并重新生成 `publishToken`，以保证推流过程的安全性。

当 `stream.publishSecurity="static"` 时，推流地址格式如下：

```
rtmp://{rtmp_publish_host}/{hub_name}/{stream_title}?key={publishKey}
```

`static` 格式的推流地址永久有效，但是把 `publishKey` 作为明文传输了，故没有 `dynamic` 格式的推流地址安全。

`dynamic` 适合 UGC 直播推流场景，`static` 适合用直播软件推流。`dynamic` 格式的推流地址需要校验 `publishToken`，`static` 格式的推流地址没有校验凭证的机制。`dynamic` 相对于 `static` 更能保证推流内容的安全。

联系我们

如需开通测试，或者有任何疑问，可以邮件联系 pili@qiniu.com 。