X-P2P

接入指引

腾讯云

文档目录

接入指引
接入步骤

最近更新时间：2025-06-09 16:18:22

步骤1：线上申请

1. 登录腾讯云官网。如果没有账号，请参考 账号注册教程 。
2. 进入腾讯云 X-P2P 产品介绍页，单击**立即申请**进入 X-P2P 服务开通申请页 。
3. 填写申请信息，单击**提交申请**完成线上申请。

步骤2：域名准备和配置

1. **提供一个已通过 ICP 备案的域名。**
   可以到任意域名服务商申请一个经过 网站备案 的域名（供直播拉流使用），也可以使用备案过的域名新建的多级域名。

   > ⚠ **注意**
   > - 必须使用**申请者组织或个人**的身份备案，不能使用云厂商备案或非申请人身份备案的域名。
   > - 备案域名服务商推荐腾讯云的 DNSPOD。
   >   **例如：**备案域名 `a1.live.qq.com`，此域名与云直播的播放域名使用方法相似，主要用于拉取视频流。

2. **在云直播控制台添加一个播放域名。**
   进入**云直播控制台** > **域名管理**，单击**添加域名**，添加一个播放域名，具体操作请参见 添加自有域名 。**例如：**添加域名 `a1.live.qq.com` 为播放域名，云直播自动生成对应的 CNAME 域名 `liveplay.a1.live.qq.com.livecdn.liveplay.myqcloud.com` 。
3. 配置 CNAME **将播放域名映射到腾讯云直播自动生成的域名。**
   例如： `a1.live.qq.com`（您提供的域名）配置 CNAME（DNS 别名）映射到 `liveplay.a1.live.qq.com.livecdn.liveplay.myqcloud.com` 。

步骤3：准备 BIZID 和 APPID

获取腾讯云账号的 BIZID

BIZID 是开通直播业务后腾讯云会给每个用户分配的 ID，BIZID 默认为**云直播控制台** > **域名管理** > **推流域名**的四级主机名。
例如：推流域名 `99999.livepush.myqcloud.com` 中 `99999` 为 BIZID。

获取腾讯云账号的 APPID

APPID 是与账号 ID 有唯一对应关系的应用 ID，部分腾讯云产品会使用此 APPID。可在**账号信息** > **基本信息**中查看您的 APPID。

步骤4：按照对应文档接入并集成 SDK

客户端类型	适用 SDK 版本
H5 SDK 对接文档	v1.8.15 以后版本（包含）
iOS SDK 对接文档	1.1.0-lts 以后版本（包含）
Android SDK 对接文档	1.1.0-lts 以后版本（包含）
Windows SDK 对接文档	v2.2.0.7 以后版本（包含）

步骤5：联调测试

根据您 步骤1 填写的联系方式，腾讯云相关人员会主动联系您，根据您提供的信息制作和发送 SDK，并辅助您完成接入、调试、上线的全部过程。完成对接上线后腾讯云 X-P2P 项目组也会实时关注 P2P 服务的质量和状态，为您的业务保驾护航。

H5 SDK FLV 接入说明

最近更新时间：2024-08-23 15:04:31

概述

腾讯云 H5-P2P 直播解决方案，可帮助用户直接使用经过大规模验证的直播流媒体分发服务。用户可通过 SDK 中简洁的接口快速同自有应用集成，实现 H5 P2P 直播功能。

SDK 名称	XP2P H5 端 FLV SDK
版本号	V1.8.31
SDK 介绍	为直播、点播、下载等场景的内容分发网络提供 P2P 点到点对等网络内容共享加速
开发者	腾讯云计算（北京）有限责任公司
个人信息处理规则	XP2P SDK 隐私保护协议
下载 SDK	单击下载 H5 FLV SDK 压缩包

运行条件

sdk 运行需要浏览器支持以下特性:

- Media Source Extensions
- WEB RTC
- websocket

PC 浏览器版本要求	版本
chrome	55+
firefox	65+
safari	11+
edge	16+
IE	不支持

安卓浏览器	是否支持
UC	不支持
QQ	不支持
微信	支持
edge	支持
百度	支持

iOS 浏览器	是否支持
所有 iOS 浏览器	不支持

> **说明：**
> 您可点击此处查看浏览器是否支持以上特性

支持的流媒体格式

- http-flv

准备工作

在 提交 X-P2P 开通申请 后，再联系我们的研发工程师，确保 CDN 分发域名及 domain 白名单已完成配置。

集成工作

1. 加载sdk

QVBP2P 导出格式为 UMD

```
<script src='qvbp2p.js'></script>
```

2. 集成 sdk 到播放器

- 接入 flv.js
- 接入TCPlayer TCPlayer 已内置集成, 传入 xp2pConfig 参数可开启 P2P

3. 集成后验证

集成后需要验证 P2P 是否正常工作, 可以通过管理后台来确认是否有 P2P 流量

开发注意事项

- 需重点关注示例中 `todo` ,需要您来实现
- 如果 sdk 触发回退的话, 请自行控制回退后的播放不再使用 p2p sdk
- qvbp2p 实例一次性使用, `qvbp2p.loadSource()` 和 `qvbp2p.destroy()` 不能重复调用, 如需要再次播放, 请重新创建 qvbp2p 实例
- 每次使用 qvbp2p 实例播放前, 请确保上一个 `qvbp2p实例` 已经调用 `qvbp2p.destroy()` (如下场景需要: 切换线路,切换清晰度,回退).

API 文档

最近更新时间：2024-12-23 15:41:22

QVBP2P

QVBP2P 是 sdk lib 名称, 可以访问到一些常量和生成 qvbp2p 实例, 如下 `qvbp2p` 表示 sdk 实例。

QVBP2P.create(config: Config): QVBP2P

创建 SDK 的实例

> ⚠ **注意：**
> 创建实例前请销毁上一个实例

参数

```
type Config = {
  tencentCloudAppId: number; // 必填，腾讯云AppId
  bizId: string; // 必填，邮件给出
  xp2pAppId: string; // 必填，邮件给出
  xp2pPackage: string; // 必填，邮件给出
  xp2pPlayDomain: string; // 必填，拉流域名，邮件给出或按需填写
  xp2pAppKey: string; // 必填，邮件给出
  authMode: 'none' | 'local' | 'remote'; // 必填，鉴权模式．请根据如下表格自行选择
  localSecKey?: string; // 可选，邮件给出
  debug?: boolean; // debug开关
}
```

`authMode` 参数含义

- 'none'：使用原有的拉流鉴权参数, 因为 SDK 运行中可能会多次重新拉流，因此使用此选项请注意您的鉴权过期时间。鉴权过期后会导致 P2P 拉流失败; 如果您的拉流域名没有防盗链，也需使用此项配置。
- 'local'：使用腾讯云的鉴权. 此选项需要您设置 `localSecKey` 字段, 值为 xp2pPlayDomain 域名对应的鉴权 key (控制台-云直播-域名管理-访问控制-Key鉴权)；SDK 内部会使用这个 key 实时生成防盗链地址。
- 'remote'：远程鉴权, 使用 XP2P 的鉴权方式, 需要您传入 xp2pAppKey, 并且由我们来配置域名的鉴权。

返回值

```
QVBP2P // sdk 实例
```

常量

QVBP2P.version: string

查看当前 sdk 版本

QVBP2P.ComEvents: ComEvents

SDK 会抛出的事件，客户端需监听处理。

定义

```
const ComEvents = {
  STATE_CHANGE: 'tp2pStateChange', // 目前sdk仅抛出了这一个事件， 客户需要监听这个事件,并处理对应的消息,消息码见 QVBP2P.ComCode
}
```

QVBP2P.ComCodes: ComCode

QVBP2P.ComEvents.STATE_CHANGE 事件对对应多种消息, 通过此处的消息码区分处理。

定义

```
const ComCode = {
  ROLLBACK: -2, // sdk回退
  RECEIVE_BUFFER: 1, // 接收flv数据chunk
  END_OF_STREAM: 3, // 流结束
  STREAM_NOT_FOUND: 4, // 启动时拉流404
};
```

- 回退说明: 当 sdk 拉流失败,播放长时间卡顿等异常情况时候，会抛出此事件，客户播放器此时需要使用原有方式重新尝试拉流，不使用 P2P。
- QVBP2P.ComCodes 和 QVBP2P.ComEvents 用法见示例

方法

QVBP2P.isSupported(): boolean

静态方法, 判断当前浏览器是否支持 sdk。

返回值

`boolean` true 表示支持 sdk

例子

```
if (QVBP2P.isSupported()) {
  // 支持sdk,可以使用
}
```

qvbp2p.listen(event: P2PEvent, callback: P2PCallback): void

实例方法, 向 sdk 注册监听的事件。

参数

```
type P2PEvent = QVBP2P.ComEvents.STATE_CHANGE;
type P2PCallback = (event: QVBP2P.ComEvents.STATE_CHANGE, data:
CallbackData) => void;

type ReceiveBufferCallbackData = {
    code: ComCodes.RECEIVE_BUFFER, // 接收数据
    payload: ArrayBuffer; // flv chunk数据
}

type RollbackCallbackData = {
    code: ComCodes.ROLLBACK
    rollbackReason: string;
};

type CallbackData = ReceiveBufferCallbackData | RollbackCallbackData;
```

例子

请参考 接入 flv.js

qvbp2p.setMediaElement(videoElement: HTMLVideoElement): void

实例方法，给 sdk 传入当前的 videoElement，sdk 使用 videoElement 进行事件监控，buffer 查询请在 `loadSource()` 调用前设置。

例子

```
const videoEl = document.getElementById('your-video-id')
```

```
qvbp2p.setMediaElement(videoEl);
```

qvbp2p.loadSource(loadConfig: LoadConfig): void

实例方法, 开始启动 sdk 拉流功能。

> ⚠ **注意:**
> **此方法不能重复调用, 只能调用一次**

参数

```
type LoadConfig = {
  src: string; // 原始flv流的url. sdk会进行解析, 然后拼出sdk内部使用的url进行拉流
  xresid?: string; // 可选, 如果不同拉流域名会使用相同的流id, 则可以自行设置不同xresid, 避免P2P串流
}
```

例子

```
qvbp2p.loadSource({
  src: 'https://xxxx.xxx.xxx/live/teststrea.flv?xxxx'
})
```

qvbp2p.destroy()

实例方法, 中断拉流, 销毁 sdk(不能重复调用)。销毁后的 sdk 也不能继续使用，需要重新创建 sdk 实例。

语法

```
qvbp2p.destroy();
```

接入 flv.js

最近更新时间：2024-12-23 15:41:22

此文档介绍了如何将 sdk 集成到 flv.js.

步骤1 根据 flv.js 的规范编写 loader

loader 在 flv.js 中是下载 flv 数据的模块, 我们需要实现一个类似的 loader, 从 p2p sdk 中接收数据。

代码如下, 代码中 `todo` 需要您来实现。

```
/**
   * XP2P loader, 对接flv.js
   */
class QVBP2PLoader extends flvjs.BaseLoader {
  /**
     * 确定当前环境是否支持sdk
     * @returns {boolean}
     */
  static isSupported() {
    return window.QVBP2P && window.QVBP2P.isSupported();
  }

  constructor(seekHander, config) {
    super();
    this._qvbp2p = null;
    // flv.js成员,非sdk必须
    this._receivedLength = 0;
    this._config = config;
  }

  /**
     * @public
     */
  destroy() {
    this._destroyQVBP2P();
    super.destroy();
  }

  /**
     * 通过p2p sdk播放一个url
     * @public
```

```
     * @param {object} dataSource
     * @param {string} dataSource.url
     */
  open(dataSource) {
    // 初始化sdk实例
    this._createQVBP2P();

    // 监听sdk事件
    this._qvbp2p.listen(
      window.QVBP2P.ComEvents.STATE_CHANGE,
      this._onQVBP2PStateChange.bind(this),
    );

    // 绑定video元素到sdk, 需要替换为客户实际的videoEl
    const videoEl = document.getElementById(this._config.videoId);
    this._qvbp2p.setMediaElement(videoEl);

    // 开始拉流, 注意对于一个sdk实例,loadSource不能重复调用
    const config = {
      src: dataSource.url,
    };
    this._qvbp2p.loadSource(config);
  }

  /**
     * @public
     */
  abort() {
    this._destroyQVBP2P();
  }

  /**
     * 接收sdk抛出的事件,针对不同事件类型有不同的处理
     *
     * 详细事件名称见 `QVBP2P.ComCodes`
     * @param {string} event 事件名称
     * @param {*} data
     */
  _onQVBP2PStateChange(event, data) {
    const { ComCodes } = window.QVBP2P;
    const { code } = data;
    switch (code) {
      case ComCodes.RECEIVE_BUFFER:
```

```
        this._receiveBuffer_need_to_implement(data.payload);
        break;
      case ComCodes.ROLLBACK:
        this._rollback_need_to_implement();
        break;
      default:
        break;
    }
  }

  /**
   * 接收sdk下载回来的flv chunk数据，送给播放器
   *
   * @todo 如果您使用的不是flv.js，则此处需要客户根据播放器情况自行实现
   *
   * @param {ArrayBuffer} chunk flv chunk
   */
  _receiveBuffer_need_to_implement(chunk) {
    const byteStart = this._receivedLength;
    this._receivedLength += chunk.byteLength;
    if (this._onDataArrival) {
      this._onDataArrival(chunk, byteStart, this._receivedLength);
    }
  }

  /**
   * 当sdk发生错误或不能继续播放时，会抛出此事件，需要客户使用原来的flv流程重新播
放
   *
   * @todo 此处需要客户根据播放器情况自行实现
   */
  _rollback_need_to_implement() {
    window.player.loadWithoutQVBP2P();
  }

  /**
   * 创建sdk实例
   */
  _createQVBP2P() {
    if (this._qvbp2p) {
      this._destroyQVBP2P();
    }
    this._qvbp2p = new window.QVBP2P.create(this._config.xp2pConfig);
```

```
    window.qvbp2p = this._qvbp2p;
  }

  /**
   * 销毁sdk实例
   */
  _destroyQVBP2P() {
    if (this._qvbp2p) {
      this._qvbp2p.destroy();
      this._qvbp2p = null;
      window.qvbp2p = null;
    }
  }
}
```

步骤2 在flv.js中使用P2P

此步骤需要您根据播放器实际情况来实现，此处为示例。

```
/**
 * 封装了flv.js的调用，根据配置选择是否开启XP2P
 */
class Player {
  constructor() {
    this.flvjsPlayer = null;
  }

  config({ useP2P, url, xp2pConfig }) {
    this.flvjsConfig = {
      mediaDataSource: {
        type: 'flv',
        url,
        isLive: true,
      },
      optionalConfig: {
        enableWorker: false,
        videoId: 'videoId', // 必填，<video>标签的id
        xp2pConfig,
      },
    };
    // 此处关键步骤，如果需要使用P2P，则需要替换
`flvjsConfig.optionalConfig.customLoader`为我们步骤1中实现的p2p loader
    this.flvjsConfig.optionalConfig.customLoader = useP2P
```

```
        ? QVBP2PLoader
        : undefined;
    }

    /**
     * 关闭P2P, 并且下次播放不再使用P2P
     */
    loadWithoutQVBP2P() {
      this.dispose();
      // 这里将customLoader置为空, 则控制flv.js不使用P2P loader
      this.flvjsConfig.optionalConfig.customLoader = undefined;
      this.play();
    }

    play() {
      if (this.flvjsPlayer) {
        this.dispose();
      }
      this.flvjsPlayer = flvjs.createPlayer(
        this.flvjsConfig.mediaDataSource,
        this.flvjsConfig.optionalConfig,
      );
      const element =
document.getElementById(this.flvjsConfig.optionalConfig.videoId);
      this.flvjsPlayer.attachMediaElement(element);
      this.flvjsPlayer.load();
    }

    dispose() {
      this.flvjsPlayer.pause();
      this.flvjsPlayer.unload();
      this.flvjsPlayer.detachMediaElement();
      this.flvjsPlayer.destroy();
      this.flvjsPlayer = null;
    }
}

// 创建
const player = new Player();
// 配置
player.config({
    useP2P: true,
    url: '需要自行替换为flv url',
```

```
    xp2pConfig: {
      // 调试 start
      debug: false,
      // 调试 end
      tencentCloudAppId: 按邮件填写,
      bizId: '按邮件填写',
      xp2pAppId: '按邮件填写',
      xp2pPackage: '按邮件填写',
      xp2pPlayDomain: '按邮件填写或参考文档',
      xp2pAppKey: '按邮件填写或参考文档',
      authMode: '按邮件填写或参考文档',
      localSecKey: '按邮件填写或参考文档',
    },
});
// 启动
window.player.play();
```

HLS
接入说明

最近更新时间：2024-12-23 15:41:22

介绍

腾讯云 H5-P2P HLS 直播点播解决方案，可帮助用户直接使用经过大规模验证的直播流媒体分发服务。用户可通过 SDK 中简洁的接口快速同自有应用集成，实现 H5 P2P 直播功能。

接入原理：将播放器的 ts 请求导向 p2p sdk, sdk 内部会调度使用 p2p 网络下载或者直接从 ts url 下载, 然后返回播放器。

SDK 名称	XP2P H5 端 HLS SDK
版本号	V1.6.37
SDK 介绍	为直播、点播、下载等场景的内容分发网络提供 P2P 点到点对等网络内容共享加速
开发者	腾讯云计算（北京）有限责任公司
个人信息处理规则	XP2P SDK 隐私保护协议
下载 SDK	单击下载 H5 HLS SDK 压缩包

运行条件

sdk 运行需要浏览器支持以下特性

- WEB RTC
- websocket
- service worker(IOS)

浏览器版本要求	桌面
Chrome	55+
Firefox	65+
Safari	11+
Edge	16+

IE	不支持

浏览器版本要求	移动端
iOS safari	支持 service worker 接入
iOS 微信	不支持
Android 微信	支持

> **说明：**
> 其他浏览器和详细信息，您可点击此处 测试浏览器是否支持以上特性。

接入前准备

准备账号

在 提交 X-P2P 开通申请 后，再联系我们工程师创建账号和ID，
您需要提供如下资料，我们创建完成后会邮件同步您相关接入资料。

- 您在腾讯云账号的 APPID（控制台查看路径：账号中心-账号信息-基本信息-APPID）

配置 CDN

对于直播 HLS 业务，需要您的 hls 域名支持 ts range 请求，配置详见 RANGE CORS 配置 。

接入中：如何接入

接入的步骤为

1. 根据我们提供的参数，创建 sdk 实例。
2. 根据播放器接入指南对接
3. sdk 生命周期管理. 播放器销毁/更换m3u8/sdk错误 时候销毁 sdk 实例

具体接口使用方式可以参照 API 文档
针对常见的播放器，我们已经进行了接入，可以根据详细文档对接。

- 接入 hls.js
- 接入 video.js(vhs)
- 接入 service worker(iOS)
- 接入TCPlayer，通过 xp2pConfig 开启。
- 接入其他播放器

接入后：对接入结果进行确认

- 启动是否正常

- 销毁是否正常
- 确认是否有 P2P 分享

您可以监听 sdk 的日志接口, 获取相关日志。

直播 HLS CORS 配置

直播场景下, sdk 需要 cdn 对 ts 请求支持 range, 因此除正常 cors 配置外, ts 请求需要支持如下配置以支持 range 请求，如何配置可以咨询对应的 cdn 厂商。

配置要求

1. ts 的请求需要支持OPTIONS请求, 并且返回的状态码是: 2xx。
2. ts 的 OPTIONS 请求响应头中需要携带:
   - Access-Control-Allow-Headers: Range
   - Access-Control-Allow-Methods: GET,OPTIONS.
3. ts 的请求支持 HTTP range
   可以使用我们部署的页面来确认是否配置成功 在线HLS range支持测试工具, 需要注意的是, 如果您的 HLS 拉流域名限制了 referer, 则会测试报错, 需要进一步单独测试。

配置含义(参考 MDN)

- OPTIONS request

> https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request

- Access-Control-Allow-Methods

> https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Methods

- Access-Control-Allow-Headers

> https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Headers

- Access-Control-Request-Headers

> https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Request-Headers

FAQ

资源 ID 生成规则

资源 ID 用来唯一标识一个HLS视频, 通常由 m3u8 url 生成. 对于资源 ID 相同的视频，SDK 内部会进行 P2P 数据分享. 因此为了避免串流, 请务必确保只有相同的具有完全相同 ts 文件的视频, 才会生成相同的资源 ID。

> **说明:**
> 如果您播放器播放的 m3u8 是多码率视频, 那么可以直接使用多码率 m3u8 生成资源 ID, SDK 内部会通过 P2P 获取当前正在播放的码率。

主动传入

您可以通过设置参数 `channelId` 字段, 主动为当前视频指定一个资源ID。

默认生成

如果您没有传入 `channelId` 字段, sdk会默认为每一个url生成一个资源ID, channelId生成规则如下:
例如: `https://a.b.com/p1/p2/p3.m3u8?m=1&n=2`

- **(默认)** 截取 host 和 path 部分生成 MD5, 即 资源 ID = MD5("a.b.com/p1/p2/p3.m3u8")
- **(可选, 默认为 true)** 通过传入 `channelIdWithHost` 参数, 可以包含host部分, 即 资源 ID = MD5("a.b.com/p1/p2/p3.m3u8")
- **(可选, 默认为 false)** 通过传入 `channelIdWithSearch` 参数, 可以包含search部分, 即 资源 ID = MD5("/p1/p2/p3.m3u8?m=1&n=2")

可以通过组合上述参数来生成适合的资源 ID

API 文档

最近更新时间：2024-12-23 15:41:22

HLSP2P 主线程 SDK

SDK API

HLSP2P.uuid

- 返回 `{string}` 用户的uuid, 同一浏览器, 同一域名页面, UUID是固定的。

HLSP2P.version

- 返回 `{string}` version sdk 版本号
  查询 sdk 的版本

HLSP2P.isSupported(): boolean

- 返回 `{boolean}` } isSupported 当前浏览器环境是否支持 sdk

检查当前浏览器环境是否支持 sdk, 请务必在创建 sdk 实例之前需要使用此接口, 如果返回 `false` ,则不能初始化 sdk。

HLSP2P.create(hlsjs, options): HLSP2P

创建自动集成到hls.js的sdk实例, 参数说明和返回值如下。

- 参数 `{HLSJS}` `hlsjs` hls.js 的实例
- 参数 `{Options}` `options` 必填参数
- 返回 `{HLSP2P}` `hlsp2p` sdk 实例

options 参数详细如下

```
type Options = {
  videoId: string; // `<video>` 标签的 id, `必填`
  videoType: 'VOD' | 'LIVE'; // `必填`, 根据实际情况选择直播还是点播
  url: string; // m3u8 的 url, `必填`
  domain: string; // 随邮件 `必填`
  xp2pAppId: string; // 随邮件 `必填`
  cloudAppId: number; // 您在腾讯云的appid `必填`
  logLevel?: LogLevel; // 可选, 设置log事件的level. 如设置为warn, 则仅抛出
warn和error. 默认为error
  channelId?: string; // 可选. 您可以主动为当前资源生成一个ID, 如果不填, 则默认
内部自动生成, 生成规则见 概览->FAQ->资源ID生成规则
```

```
  channelIdWithHost?: boolean; // 可选. 默认为true. 生成的资源ID包含m3u8
host部分, 不同host的url不会互相分享. 详见概览->FAQ
  channelIdWithSearch?: boolean; // 可选. 默认为false. 生成的资源ID不包含m3u8
search部分. 详见概览->FAQ
  enableServiceWorker?: boolean; // 可选, 默认为false. 是否开启支持service
worker功能. 使用前请与我们确认
}

type LogLevel = 'log' | 'info' | 'warn' | 'error';
```

HLSP2P.createCommon(options): HLSP2P

创建通用的sdk实例

- **参数** `{Options}` `options` **必填参数**

options 参数详细如下

```
type Options = {
  videoId: string; // `<video>` 标签的 id, `必填`
  videoType: 'VOD' | 'LIVE'; // `必填`, 根据实际情况选择直播还是点播
  url: string; // m3u8 的 url, `必填`
  domain: string; // 随邮件 `必填`
  xp2pAppId: string; // 随邮件 `必填`
  cloudAppId: number; // 您在腾讯云的appid `必填`
  logLevel?: LogLevel; // 可选, 设置log事件的level. 如设置为warn, 则仅抛出
warn和error. 默认为error
  channelId?: string; // 可选, 您可以主动为当前资源生成一个ID, 如果不填, 则默认
内部自动生成, 生成规则见 概览->FAQ->资源ID生成规则
  channelIdWithHost?: boolean; // 可选, 默认为true. 生成的资源ID包含m3u8
host部分, 不同host的url不会互相分享. 详见概览->FAQ->资源ID生成规则
  channelIdWithSearch?: boolean; // 可选, 默认为false. 生成的资源ID不包含m3u8
search部分. 详见概览->FAQ->资源ID生成规则
  enableServiceWorker?: boolean; // 可选, 默认为false. 是否开启支持service
worker功能. 使用前请与我们确认
}
```

hlsp2p.on(HLSP2P.Events.Rollback, (msg: RollbackMsg) => void)

监听 hlsp2p 抛出的回退事件, 回退表示不能继续使用 SDK。

```
type RollbackMsg = {
    reason: 'unknown_domain' | 'config_rollback';
```

```
}
```

value	含义
unknown_domain	内网模式生效，域名未在后台配置
config_rollback	主动配置回退

hlsp2p.destroy()

关闭 sdk 实例 hlsp2p, 释放资源, 停止代理播放器请求。

hlsp2p.attachVhsPlayer(vhs): boolean

用于对接 video.js的http-streaming 模块, 调用此方法后, hlsp2p 会拦截 video.js 的 ts 请求, 并通过 hlsp2p 内部进行下载(从 cdn 或从 p2p)

- 参数 `{Vhs}` `vhs` video.js http-streaming的实例, 获取方式为 `player.tech().vhs`
- 返回 `{Boolean}` 返回 true 表示对接成功, false 表示对接失败. 对接失败的情况下, 不会影响 video.js 的正常使用。

Service Worker内部的XP2P SDK

此 SDK 用于在 iOS 上通过 ServiceWorker 功能来支持 P2P, 在 service worker 内部运行，此 SDK 需要配合主线程 HLSP2P SDK 使用。

SDK API

XP2PSW.XP2PSWLib

XP2P Service worker 内部 sdk 的 lib 名称, 通过这个接口来操作 sdk。

static XP2PSW.XP2PSWLib.version

静态属性, 获取 XP2PSW SDK 的版本号。

- 返回 `{string}`

static XP2PSW.XP2PSWLib.create()

静态方法, 创建 Service Worker 内部 sdk 的实例, 单例模式, 多次调用都会返回同一个 sdk 实例。

```
const xp2pSWSDKInstance = XP2PSW.XP2PSWLib.create();
```

- 返回 `{XP2PSW.XP2PSWLib}`

xp2pSWSDKInstance.enableLog(param: LogParam)

实例方法, 调用后可以开启日志功能，可以多次调用。

```
type LogParam = {
  enableLog: boolean, // true则使用console.log打印日志
  enableUpload: boolean, // true则, 开启日志上传到P2P后台用于分析, 请不要在线上环境开启
  logLevel: 'log' | 'info' | 'warn' | 'error', // 日志等级, 可选值有 log, info, warn, error, 默认是error级别
}
```

- 参数 `{LogParam}` param 传入参数来控制日志的功能
- 返回 无

xp2pSWSDKInstance.detectXP2PRequest(event)

实例方法, 用于检测 Service Worker 内拦截的 fetch 事件, 是否是 XP2P 内部的事件. 如果返回 true, 则这个 event 请务必使用 sdk 来处理(`xp2pSWSDKInstance.match(event)`)

- 参数 `{FetchEvent}` event
- 返回 `{boolean}`

xp2pSWSDKInstance.match(event)

- 参数 `{FetchEvent}` event
- 返回 `Promise<Response>`

通过 xp2p service worker sdk 来代理这个请求，适用的 fetch event 包括如下三种。

- detectXP2PRequest(event) 返回 true
- event.request.url 是 ts url (需要和 HLSP2P 初始化 m3u8 里边的 ts 一致)
- event.request.url 是 m3u8 url (需要和 HLSP2P 初始化 m3u8 一致)

当 XP2P sdk 代理请求返回成功的时候, 会返回 Promise<Response>; 当代理请求失败的时候, 需要对返回 promise 进行异常处理，示例如下。

```
// service worker内部
self.xp2pSWSDKInstance = XP2PSW.XP2PSWLib.create();

self.addEventListener('fetch',
  /**
   * @param {FetchEvent} event
   */
  function (event) {
    console.log('sw: fetch event', event.request.url);
```

```
    if (self.xp2pSWSDKInstance &&
(self.xp2pSWSDKInstance.detectXP2PRequest(event) ||
isTsOrM3u8Url(event.request.url))) {
      console.log('sw: fetch event XP2P sdk处理', event.request.url);
      return event.respondWith(
        self.xp2pSWSDKInstance.match(event)
        .then(function (response) {
          console.warn('sw: fetch event resp接收成功', event.request.url,
response);
          // 这里不要删除
          if (self.xp2pSWSDKInstance.detectXP2PRequest(event)) {
            // 此处需要直接返回，客户不能做任何修改
            return response;
          }
          // 这里不要删除
          if (new URL(event.request.url).searchParams.get('xfrom')) {
            // 此处需要直接返回，客户不能做任何修改
            return response;
          }
          return response;
        })
        .catch(function (e) {
          console.warn(`sw: fetch event [XP2P] match error: ${e}，异常兜底
请求 ${event.request.url}`);
          // 注意：safari会增加pragma header,导致发出不必要的cors preflight请
求，会导致请求失败．这里删除掉
          const r = new Request(event.request);
          r.headers.delete('pragma');
          const response = fetch(r);
          // 这里不要删除
          if (self.xp2pSWSDKInstance.detectXP2PRequest(event)) {
            // 此处需要直接返回，客户不能做任何修改
            return response;
          }
          // 这里不要删除
          if (new URL(event.request.url).searchParams.get('xfrom')) {
            // 此处需要直接返回，客户不能做任何修改
            return response;
          }
          return response;
        })
      );
    }
```

```
});
```

xp2pSWSDKInstance.destroy()

销毁 XP2P Service Worker SDK 实例

接入 hls.js

最近更新时间：2024-08-23 15:04:32

P2P 的 SDK 适配了 hls.js 接口, 如果您使用hls.js播放器, 请按照此文档接入, (hls.js 的版本>=0.9.1)

对接准备

请参考接入说明

> **注意：**
> 请注意当前 sdk 不支持多实例

对接示例

此代码对接了 hls.js

```
const video = document.getElementById("video");
if (Hls.isSupported()) {
  // 创建hls.js实例
  const hls = new Hls();
  hls.loadSource(url);
  hls.attachMedia(video);
  hls.on(Hls.Events.MANIFEST_PARSED, function () {
    video.play();
  });

  // 如下初始化P2P, 并继承hls.js
  if (HLSP2P.isSupported()) {
    // 首先确定是否支持sdk
    const hlsp2p = HLSP2P.create(hls, {
      videoId: "这里需要根据实际情况修改为<video>的id",
      videoType: "VOD", // 或者 'LIVE', 根据实际情况选择
      url: "根据实际情况填写hls视频的m3u8 url",
      domain: "见邮件",
      xp2pAppId: "见邮件",
      xp2pAppKey: "见邮件",
      cloudAppId: "您在腾讯云的appid", // number类型
    }); // 创建sdk实例
    hlsp2p.on(HLSP2P.Events.Rollback, () => {
      // 监听sdk抛出的异常
      hlsp2p.destroy(); // 销毁sdk
    });
```

```
    }
}
```

```
// 销毁sdk示例
window.hlsp2p.destroy();
window.hlsp2p = null;
```

对接注意事项(重要)

1. 当关闭播放器的时候, 需要同时销毁 P2P sdk
2. 当更换视频播放的时候, 需要销毁 sdk 并重新初始化 sdk

接入 video.js(vhs)

最近更新时间：2024-08-23 15:04:32

此文档对接了 video.js http-streaming 模块, 简称 vhs

对接准备

请参考接入说明

> ⚠ **注意：**
> 请注意当前 sdk 不支持多实例

对接示例

```
  <video id="my_video_2" class="video-js" controls preload="auto"
width="640" height="268">
    <source src="https://test-
streams.mux.dev/x36xhzz/url_0/193039199_mp4_h264_aac_hd_7.m3u8">
  </video>
  <button onclick="startP2P">创建p2p sdk</button>
  <button onclick="destroyP2P">销毁p2p sdk</button>
```

```
    var videoId = 'my_video_2';
    var player = videojs(videoId, {}, function () {
      startP2P(player);
    });

    function startP2P(player) {
      // 获取videojs内置的vhs插件
      var vhs = window.player.tech().vhs;
      if (vhs && HLSP2P.isSupported() && !window.hlsp2p) {
        // 创建sdk实例
        const hlsp2p = HLSP2P.createCommon({
          videoId: videoId + '_html5_api', // 特别注意是<video>的id, 对于
videojs来说, 传入的id拼接 _html5_api
          videoType: 'VOD', // 根据实际情况填写, 见API
          url: '这里要替换为实际的m3u8 url',
          domain: '这里要根据邮件填写',
          xp2pAppId: '这里要根据邮件填写',
          xp2pAppKey: '这里需要根据邮件填写',
```

```
        cloudAppId: 需要替换为您在腾讯云的AppId
      });
      window.hlsp2p = hlsp2p;

      // 监听SDK播放失败事件
      hlsp2p.on(HLSP2P.Events.Rollback, () => {
        // 需销毁sdk
        hlsp2p.destroy();
      });

      // 将vhs绑定到p2p sdk, 从此vhs的下载的ts都会经由p2p sdk
      const result = hlsp2p.attachVhsPlayer(vhs);
      if (result) {
        console.log('vhs绑定p2p sdk成功');
      } else {
        console.log('vhs绑定p2p sdk失败');
      }
    }
  }

  // 注意,
  // 1. 当关闭播放器的时候, 需要同步关闭P2P sdk!
  // 2. 当更换视频的时候, 需要销毁sdk并重新初始化sdk
  function destroyP2P() {
    if (window.hlsp2p) {
      window.hlsp2p.destroy();
      window.hlsp2p = null;
    }
  }

  /**
   * 关闭播放器同步关闭P2P sdk!
   */
  function closePlayer() {
    player.dispose();
    destroyP2P();
  }
```

对接注意事项(重要)

1. 当关闭播放器的时候, 需要同步销毁 P2P sdk
2. 当更换视频播放的时候, 需要销毁 sdk 并重新初始化 sdk

接入 service worker(iOS)

最近更新时间：2025-05-21 16:46:21

本文档适用于 iOS Safari 浏览器的 HLS 播放，提供 P2P 功能 XP2P 提供了两个 SDK：XP2P_MAIN_SDK 和 XP2P_SW_SDK 。

- XP2P_MAIN_SDK 用于主线程。
- XP2P_SW_SDK 用于 service worker 内部。

接入原理

交互示意

safari 播放器 <---http----> service worker(XP2P_SW_SDK) <---http/message 通道----> XP2P_MAIN_SDK

交互描述

1. 播放器发出的 m3u8 请求
   - service worker 拦截播放器发出的 m3u8 请求
   - service worker 直接请求 m3u8 并返回给播放器 (XP2P_SW_SDK 已实现)
2. 播放器发出的 ts 请求
   - service worker 拦截播放器的 ts 请求
   - worker 内部的 XP2P_SW_SDK 向主线程 XP2P_MAIN_SDK 查询这个 ts (XP2P_SW_SDK 已实现)
   - XP2P_MAIN_SDK 通过 P2P 或者 http 获取到 ts, 返回给 service worker (XP2P_SW_SDK 已实现)
   - service worker 将 ts 返回给播放器

demo

参见 我们的接入邮件 demo/sw 目录。

实际接入

接入前准备

1. 按照接入说明 中的接入前准备

2. 忽略 message 事件中 XP2P 的消息

xp2p XP2P_MAIN_SDK 和 XP2P_SW_SDK 之间的通信，依赖于 service worker 的 message 通信机制，因此如果您使用到了 message 事件或者设置了 onmessage。
请忽略 XP2P 传递的消息，忽略方法如下。

```
/**
 * 主线程内，如果使用message事件接收service worker消息
 * @param {MessageEvent} event
 */
self.navigator.serviceWorker.addEventListener('message', (event) => {
  // 判断xp2p消息的方法，如果是XP2P消息，请务必忽略，不要处理或修改
  if (!Object.prototype.hasOwnProperty.call(event.data, 'xp2pType')) {
    return;
  }
});

/**
 *或者 如果您在主线程使用onmessage接收service worker消息
 * @param {MessageEvent} event
 */
navigator.serviceWorker.onmessage = (event) => {
  // 判断xp2p消息的方法，如果是XP2P消息，请务必忽略，不要处理或修改
  if (!Object.prototype.hasOwnProperty.call(event.data, 'xp2pType')) {
    return;
  }
};

/**
 * 在service worker内如果使用message事件接收主线程消息
 * @param {MessageEvent} event
 */
self.addEventListener('message', (event) => {
  // 判断xp2p消息的方法，如果是XP2P消息，请务必忽略，不要处理或修改
  if (!Object.prototype.hasOwnProperty.call(event.data, 'xp2pType')) {
    return;
  }
});
/**
 * 或者 如果您在service worker内部使用onmessage接受主线程消息
 * @param {MessageEvent} event
 */
self.onmessage = (event) => {
  // 判断xp2p消息的方法，如果是XP2P消息，请务必忽略，不要处理或修改
  if (!Object.prototype.hasOwnProperty.call(event.data, 'xp2pType')) {
    return;
  }
};
```

3. 设置 service worker 文件的拦截作用域

这一步的目的是确保 sw.js 可以拦截到播放器的 ts 和 m3u8 请求。

> **说明：**
> 参考 https://www.w3.org/TR/service-workers/#service-worker-allowed 。

(1) 设置 sw.js 的 http response header

```
// 增加一个 response header, 设置sw.js的作用域是跟路径
Service-Worker-Allowed : "/"
```

(2) service worker 代码内设置作用域

```
// 提升sw.js的拦截作用域，虽然sw.js的路径是在/js/目录下，但是可以拦截根路径的请求
navigator.serviceWorker.register("/js/sw.js", {scope: "/"}).then(() => {
  console.log("Install succeeded as the max allowed scope was overriden
to '/'.");
});
```

SDK 代码接入

> **说明：**
> 相关的 SDK API 请参考 API 文档 。

主线程 XP2P 接入代码(XP2P_MAIN_SDK)

1. 创建 sdk 实例。
2. 播放器播放。

```


// 判断当前浏览器环境是否支持XP2P
if (!HLSP2P.isSupportedInServiceWorker()) {
  alert('浏览器不支持');
  return;
}

// TODO 配置的详细内容请参考API文档
const config = {
  // 如下为必传参数
```

```
  enableServiceWorker: true, // 仅在service worker环境中开启，不传默认为
false
  // 如下为必传参数
  videoId: 'video_id', // 特别注意是<video>的id，这里需要跟据实际情况修改
  url: '实际播放的m3u8 url',
  domain: '这里要根据邮件填写',
  xp2pAppId: '这里要根据邮件填写',
  cloudAppId: 您的腾讯云APPID,
  videoType: 'LIVE', // 重要!! 直播写LIVE，点播写VOD,
};

// 创建SDK实例
const hlsp2p = HLSP2P.createCommon(config);

// 监听SDK播放失败事件
hlsp2p.on(HLSP2P.Events.Rollback, ({reason}) => {
  console.warn(`p2p 回退 destroy, reason: ${reason}`);
  // 需销毁sdk
  hlsp2p.destroy();
});
window.hlsp2p = hlsp2p;

/**
 * 当停止播放时调用销毁SDK
 */
function destroySDK() {
  if (window.hlsp2p) {
    window.hlsp2p.destroy();
    window.hlsp2p = null;
  }
}
```

service worker 内部接入代码(XP2P_SW_SDK)

1. 加载 XP2P_SW_SDK。
2. 创建 XP2P_SW_SDK 实例。
3. 监听 fetch event 并拦截必要的请求：包括 xp2p 内部请求，m3u8 请求，ts 请求。

```
// 在service worker内部加载XP2P_SW_SDK
var xp2pswUrl = '这里写XP2P_SW_SDK的url'
importScripts(xp2pswUrl);
```

```
// 创建sw内的XP2P SDK实例，并绑定在self作用域
// 此处create函数返回单例
self.xp2pIns = XP2PSW.XP2PSWLib.create();

/**
 * TODO 这里您可以自行实现，请仅拦截视频正片的ts和m3u8
 * @param url
 * @return {boolean}
 */
function isTsOrM3u8Url(url) {
  return url.indexOf(".ts") > -1 || url.indexOf(".m3u8") > -1;
}

// 在fetch事件中，使用XP2P处理必要的请求
self.addEventListener('fetch',
  /**
   * @param {FetchEvent} event
   */
  (event) => {
    // TODO XP2P 拦截两种请求：xp2p自身的请求和 HLS请求
    // TODO 您可以根据自己的灰度策略，选择只开启部分资源走P2P
    if (self.xp2pIns && (self.xp2pIns.detectXP2PRequest(event) ||
isTsOrM3u8Url(event.request.url))) {
      return event.respondWith(
        // 调用XP2P的match方法来处理fetch event，尝试请求
        self.xp2pIns.match(event)
          .then(response => {
            // 通过XP2P SDK请求成功，返回response给fetch请求
            // TODO 如果需要对respone进行处理，可以在这里做
            return response;
          })
          .catch((e) => {
            // TODO 重要！当XP2P请求失败或者XP2P关闭了service worker功能的时候
会走到这里，需要在这里兜底请求一次
            // 注意：safari会增加pragma header,导致发出不必要的cors preflight
请求，会导致请求失败。这里删除掉
            const r = new Request(event.request);
            r.headers.delete('pragma');
             // TODO 如果需要对respone进行处理，可以在这里做
            return fetch(r);
          })
      );
```

```
    }
    // TODO 对于不需要XP2P处理的其他请求，您可以在此处自行处理
  });
```

测试

如何确认 XP2P 已经启动

观察网络面板，XP2P_MAIN_SDK 启动一段时间后，会发出 `https://xw.xp2p/hello` 请求，如果响应如下则表示启动成功。

```
statusCode: 200
body: {
  code: 0
}
```

如何确认是否成功收到 P2P 数据

1. XP2P_MAIN_SDK 提供了数据统计接口，包含了 P2P 和 cdn 字节数，可以参考 API 文档 。
2. 可以观察浏览器网络面板，对于一个ts请求例如： 1.ts。
   - 如果没有通过 P2P 接收到数据，则会在 network 面板上观察到两个 1.ts 请求，分别由播放器和 XP2P_MAIN_SDK 发出。
   - 如果通过 P2P 接收到数据，则在 network 面板上只会看到一个 1.ts 请求，由播放器发出。

接入其他播放器

最近更新时间：2024-08-23 15:04:32

sdk 提供了通用的sdk接口，您可以自行接入播放器

> ⚠ **注意：**
> 请注意当前 sdk 不支持多实例

对接准备

请参考接入说明

对接示例

```
if (HLSP2P.isSupported()) { // 首先确定是否支持sdk
  // 创建sdk实例; 请注意这里创建接口为createCommon, 并且不能传入hls.js实例
  // 详细参数见如下接口API
  const hlsp2p = HLSP2P.createCommon({
    videoId: '这里需要根据实际情况修改为<video>的id',
    videoType: 'VOD', // 根据实际情况填写, 见API
    url: '根据实际情况填写hls视频的m3u8 url',
    domain: '见邮件',
    xp2pAppId: '见邮件',
    xp2pAppKey: '见邮件',
    cloudAppId: 您在腾讯云的appid, // number类型
  });
  hlsp2p.on(HLSP2P.Events.Rollback, () => { // 监听SDK播放失败事件
    hlsp2p.destroy(); // 需销毁sdk
  });

  // 请注意, 切换线路, 切换清晰度或不使用sdk后,都需主动调用destroy()


  // 当需要下载ts文件的时候, 如下为下载ts示例
  // 请求的ts参数
  const requestContext = {
    url: 'ts_download_url' // 完整的ts文件url
  };

  type SuccessData = ArrayBuffer;
```

```
    type SuccessStats = {
        loading: {
            start: number; // 请求的开始时间
            first: number; // 收到首包的时间
            end: number; // 请求结束的时间
        }
    }

    // 为请求设置回调方法
    const requestCallbacks = {
      onSuccess: (tsData: SuccessData, stats: SuccessStats) => {
        // 下载成功后会调用. tsData即为下载完成的ts数据(Arraybuffer)
      },
      onError: () => {
        // 下载失败后会调用
      },
      onTimeout: () => {
        // 下载超时后会调用这个回调.
      },
    }

    // 获取ts下载器
    const commonLoader= hlsp2p.commonLoader;
    // 传入刚才的参数, 通过p2p sdk ts下载器下载ts文件
    commonLoader.loadTs(requestContext, requestCallbacks);
}
```

对接注意事项(重要)

1. 当关闭播放器的时候, 需要同步销毁 P2P sdk
2. 当更换视频播放的时候, 需要销毁 sdk 并重新初始化 sdk

iOS SDK
接入说明

最近更新时间：2025-05-22 17:06:52

腾讯云 X-P2P 解决方案，可帮助用户直接使用经过大规模验证的直播、点播、文件分发服务，通过经商用验证的 P2P 服务大幅节省带宽成本，提供更优质的用户体验。开发者可通过 SDK 中简洁的接口快速同自有应用集成，实现 iOS 设备上的 P2P 加速功能。
传统 CDN 加速服务中，客户端向 CDN 发送 HTTP 请求获取数据。在腾讯云 X-P2P 服务中，SDK 可以视为下载代理模块，客户端应用将 HTTP 请求发送至 SDK，SDK 从 CDN 或其他 P2P 节点获取数据，并将数据返回至上层应用。SDK 通过互相分享数据降低 CDN 流量，并通过 CDN 的参与，确保了下载的可靠性。

SDK 名称	XP2P iOS 端 SDK
版本号	V1.3.5
SDK 介绍	为直播、点播、下载等场景的内容分发网络提供P2P点到点对等网络内容共享加速
开发者	腾讯云计算（北京）有限责任公司
个人信息处理规则	SDK个人信息保护规则
合规使用指南	SDK合规使用指南
下载 SDK	单击下载 iOS SDK压缩包

说明
SDK 支持多实例，即支持同时开启多个直播 P2P。

支持的架构

当前 iOS SDK 支持如下 ABI 架构：

- armv7
- armv7s
- arm64
- x86
- x86_64

集成 SDK

通过 CocoaPods 集成

1. 在项目根目录创建 Podfile 文件。

```
platform :ios, '9.0'
target 'RTMPiOSDemo' do #这里需要修改成自己的target工程
  use_frameworks!
  pod 'xnet', :git => 'https://github.com/tencentyun/xnet-ios-
sdk.git', :tag => "release-lastest"
end
```

2. 在 Terminal 中执行以下命令。

```
pod install
```

如果您是更新版本请执行：

```
pod update
```

3. 打开 workspace：

```
open RTMPiOSDemo.xcworkspace #您需要打开自己的xcworkspace
```

通过 framework 集成

1. 拷贝 X-P2P 团队提供的 `xnet.framework` 。
2. 直接通过 `Xode -> General -> "Framework, Libraries, and Embedded Content"` 添加 `xnet.framework` 。

配置应用

腾讯云对接人员会提供 iOS 项目的 Bundle identifier，并索取 `App ID` 、`App Key` 、`App Secret Key` ，如以下形式：

```
Bundle identifier: com.qcloud.helloworld
NSString *appID = @"your_appID";
NSString *key = @"your_key";
NSString *secret = @"your_secret";
```

接入步骤

1. 解压 `TencentXP2P.zip` 文件得到 `TencentXP2P.framework`，并在项目中引用。
2. 在 App 启动时初始化 `XP2PModule`，并随后初始化 P2P SDK。

```
    // Example: 程序的入口AppDelegate.m
    #import "AppDelegate.h"
    #import <TencentXP2P/TencentXP2P.h>
    @implementation AppDelegate
    - (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
    {
        // do something other...

        //下面的接口调用可以开启SDK log的打印，不调用该接口，log打印默认不开启
        [XNet enableDebug]

        NSString *appID = @"your app id";
        NSString *key = @"your app key";
        NSString *secret = @"your app secret";
        bool ok = [XNet initWith:appID appKey:key
appSecretKey:secret];

        return YES;
    }

    - (void)onLogPrint:(NSString*)msg {
        //这里能够收到SDK输出的日志
    }

    @end
```

3. 加载一个频道。

```
 - (int)startPlay:(NSString*)url
 {
     // 将resource之前的 域名/路径 内容替换为 [XP2PModule host] 即可，例如
     // 原先是，http://example.cdn.com/live/streamid.flv?token=xxx
     // 转换成，
http://127.0.0.1:16080/live.p2p.com/example.cdn.com/live/streamid.flv?
token=xxx
     NSString* p2pUrl = [url
stringByReplacingOccurrencesOfString:@"http://" withString:
                            [XNet proxyOf:@"live.p2p.com"]];
```

```
    NSString *codecID = @"_h265";
    NSString *stremaid = [[url lastPathComponent]
stringByDeletingPathExtension];
    NSString *xresid = [NSString stringWithFormat:@"%@_%@", stremaid,
codecID];
    NSString* param = [NSString stringWithFormat:@"?xresid=%@",
xresid];
    p2pUrl = [p2pUrl stringByAppendingString:param];
    // 直接播放此url即可
    [_player startPlay:p2pUrl];
    return EXIT_SUCCESS;
 }
```

卸载一个频道。

```
- (int)stopPlay
{
    // 播放器链接断开以后 SDK内部会自动释放频道相关的资源，直接关闭播放器即可
    [_player stopPlay];
    return EXIT_SUCCESS;
}
```

4. 恢复前台显示时，必须调用 resume。否则在播放器暂停阶段 iOS 会关闭链接，此时向代理发起请求，会收到类似502的错误。

```
bool ok = [XNet resume];
```

注意事项

以上是一个播放器使用 P2P 的基本调用，完成上述调用后您就可以使用 P2P 为播放器提供服务了，但一个更加完善的客户端还需要做以下工作：

- 统计流量，获取 SDK 运行过程中的 P2P 流量和 CDN 流量。
- 监听网络变化，在移动网络显示 P2P 上传，WIFI 网络开启 P2P 上传（默认开启）。

这些设置您需要以 HTTP 请求的方式告知 SDK 代理服务，调用 API 请参见 API 说明 。

API 说明

最近更新时间：2025-06-09 16:18:22

SDK 接口除初始化接口，其余接口均由 HTTP 实现，请求格式为：

```
http:://${host}/${func}?${param}
```

其中 `${host}` 为本地代理服务器，通过 `XNet.HTTP_PROXY` 获取。

统计

接口说明

- 描述：请求对应频道的统计数据
- 方法：GET
- 路径：`/stat?xresid=${resource}`

请求参数

参数名称	必选	类型	说明
xresid	是	string	默认为 URL 中的 resource，否则为频道请求中的 xresid 值

返回参数

返回码	说明
200	查询成功
404	查询失败，频道不存在

其中，返回码 200 返回的 JSON 内容，格式详细说明如下：

参数名称	类型	说明
flow.p2pBytes	num	对应频道 P2P 流量
flow.cdnBytes	num	对应频道 CDN 流量

示例

- **请求示例：**

```
http://127.0.0.1:16080/live.p2p.com/stat?xresid=${yourURL}
```

> **注意**
> xresid 即 `http://127.0.0.1:16080/live.p2p.com/resoruce.ext` 中的 resource。

- **返回示例：**

```
{"flow":{"p2pBytes":0,"cdnBytes":0}}
```

设置上下行

接口说明

- 描述：请求设置 P2P 上行与下行，0为关闭，1为开启。
- 方法：GET
- 路径：`/feature?download=${0or1}&upload=${0or1}`

请求参数

参数名称	必选	类型	说明
download	是	num	0为关闭；1为开启（默认值）
upload	是	num	0为关闭；1为开启（默认值）

返回参数

返回的 JSON 内容，格式说明如下：

参数名称	必选	类型	说明
ret	是	num	0为正常
msg	是	string	相关信息，调试使用
upload	是	bool	1为开启，0为关闭
download	是	bool	1为开启，0为关闭

示例

- **请求示例：**

```
http://127.0.0.1:16080/live.p2p.com/feature?download=1&upload=0
```

> **说明**
> 一般情况下移动网络需要关闭上传。

- **返回示例：**

```
{"ret":0, "msg":"ok", "download":0,"upload":0}
```

Android SDK 接入说明

最近更新时间：2025-05-22 17:06:52

腾讯云 X-P2P 解决方案，可帮助用户直接使用经过大规模验证的直播、点播、文件分发服务，通过经商用验证的 P2P 服务大幅节省带宽成本，提供更优质的用户体验。开发者可通过 SDK 中简洁的接口快速同自有应用集成，实现 Android 设备上的 P2P 加速功能。
传统 CDN 加速服务中，客户端向 CDN 发送 HTTP 请求获取数据。在腾讯云 X-P2P 服务中，SDK 可以视为下载代理模块，客户端应用将 HTTP 请求发送至 SDK，SDK 从 CDN 或其他 P2P 节点获取数据，并将数据返回至上层应用。SDK 通过互相分享数据降低 CDN 流量，并通过 CDN 的参与，确保了下载的可靠性。

SDK 名称	XP2P Android 端 SDK
版本号	V1.3.5
SDK 介绍	为直播、点播、下载等场景的内容分发网络提供P2P点到点对等网络内容共享加速
开发者	腾讯云计算（北京）有限责任公司
个人信息处理规则	SDK个人信息保护规则
合规使用指南	SDK合规使用指南
下载 SDK	单击下载 Android SDK压缩包

> **说明**
> SDK 支持多实例，即支持同时开启多个不同资源的直播 P2P。

支持的架构

当前 Android SDK 支持如下 ABI 架构：

- armeabi-v7a
- arm64-v8a
- x86
- x86_64

添加依赖

通过 bintray 获取最新的 gradle 组件包：

```
implementation 'com.tencent.qcloud:xnet:release-lastest'
```

混淆配置

由于 native 层代码需要反射调回 java，需要确保 SDK 内的代码都不被混淆，请在 proguard 中添加以下配置：

```
-keep public class com.tencent.qcloud.** {
    *;
}
```

接入步骤

步骤1：初始化

在 App 启动时初始化 `XP2PModule`，并随后初始化 P2P SDK。

```
// 初始化appId等关键客户信息
final String APP_ID = "$your_app_id";
final String APP_KEY = "$your_app_key";
final String APP_SECRET = "$your_app_secret";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // 返回值ok可用于统计p2p成功与失败
    // 无论成功与失败，都可以使用XNet.proxyOf与XNet.proxyOf
    // 失败时，XNet.proxyOf是""
    // 成功时，XNet.proxyOf是127.0.0.1:16080/${domain}
    bool ok = XNet.create(this, APP_ID, APP_KEY, APP_SECRET);

    //开启调试日志(发布可关闭)
    XNet.enableDebug();
}
```

步骤2：接入直播 Live

直播播放控制 P2P（start/stop），操作如下：

1. 启动直播 P2P：

```
// 例如：http://domain/path/to/resource.flv?params=xxx
// 变成：http://{XNet.proxyOf(domain)}/path/to/resource.flv?params=xxx
// 要改的仅仅是在://后插入XNET.proxyOf
    String codecID = "_h265";
    String streamID = url.substring(url.lastIndexOf(("/") + 1));
    streamID = streamID.substring(0, streamID.indexOf("."));

    String xResID = streamID + codecID;
    String param =   "?xresid=" + xResID; //该参数标识P2P资源

    String p2pURl = url.replace("http://",
XNet.proxyOf("live.p2p.com"));
    p2pURl += param;

// 通过获取的url，播放器直接执行请求
    mediaPlayer.setDataSource(context, Uri.parse(p2pUrl), headers);
    mediaPlayer.prepareAsync();

    // 相关资源随着http请求关闭直接释放，无需再做任何处理
    //注意：该http请求可能会返回302
```

2. 由于后台监听限制的问题，resume 的情况下需要调用 resume 接口。

```
 // 返回值ok代表着是否恢复成功
 bool ok = XNet.resume();
```

API 说明

最近更新时间：2025-06-09 16:18:22

SDK 接口除初始化接口，其余接口均由 HTTP 实现，请求格式为：

```
http:://${XNet.proxyOf}${func}?${param}
```

> **注意**
> 在初始化返回值为 false 或 `XNet.proxyOf` 时，意味着没有成功使用 P2P 播放视频，不应再向其发起 HTTP 请求，否则会遇到502等错误，也可作为没有成功使用 P2P 播放器的标志。

统计

接口描述

- 描述：请求对应频道的统计数据
- 方法：GET
- 路径：`/stat?xresid=${resource}`

请求参数

参数名称	必选	类型	说明
xresid	是	string	默认为 URL 中的 resource，否则为频道请求中的 xresid 值

返回参数

返回码	说明
200	查询成功
404	查询失败，频道不存在

其中，返回码 200 返回的 JSON 内容，格式详细说明如下：

参数名称	类型	说明
flow.p2pBytes	num	对应频道 P2P 流量
flow.cdn	num	对应频道 CDN 流量

Bytes		

示例

- **请求示例**

```
http://127.0.0.1:16080/live.p2p.com/stat?xresid=${yourURL}
```

> **说明**
> xresid 默认为 `http://127.0.0.1:16080/live.p2p.com/streamId.flv` 中的 `streamId`，或用户在播放时主动指定直播流资源唯一的 ID。

- **返回示例**

```
{"flow":{"p2pBytes":0,"cdnBytes":0}}
```

设置上下行

接口描述

- 描述：请求设置 P2P 上行与下行，0为关闭，1为开启。
- 方法：GET
- 路径：`/feature?download=${0or1}&upload=${0or1}`

请求参数

参数名称	必选	类型	说明
download	是	num	0为关闭，1为开启（默认值）
upload	是	num	0为关闭，1为开启（默认值）

返回参数

返回的 JSON 内容，格式说明如下：

参数名称	必选	类型	说明
ret	是	num	0为正常
msg	是	string	相关信息，调试使用

upload	是	bool	1为开启，0为关闭
download	是	bool	1为开启，0为关闭

示例

- **请求样例：**

```
http://127.0.0.1:16080/live.p2p.com/feature?download=1&upload=0
```

> **说明**
> 一般情况下移动网络需要关闭上传。

- **返回样例：**

```
{"ret":0, "msg":"ok", "download":0,"upload":0}
```

Windows SDK 接入说明

最近更新时间：2024-08-23 17:03:41

腾讯云 X-P2P 直播解决方案，可帮助用户直接使用经过大规模验证的直播流媒体分发服务。用户可通过 SDK 中简洁的接口快速同自有应用集成，实现 Windows 客户端的 P2P 直播功能。

SDK 名称	XP2P Windows 端 SDK
版本号	V1.2.4
SDK 介绍	为直播、点播、下载等场景的内容分发网络提供P2P点到点对等网络内容共享加速
开发者	腾讯云计算（北京）有限责任公司
个人信息处理规则	XP2P SDK 隐私保护协议
下载 SDK	单击下载 Windows SDK 压缩包

准备工作

- 在 提交 X-P2P 开通申请 后，再联系我们的研发工程师，确保 CDN 分发域名及 domain 白名单已完成配置。
- 确保您已在开发者中心上注册账号并创建应用，创建应用时要写对包名。应用创建成功后获得一对有效的 `Access ID`、`Key`、`Secret`。

> **说明**
> 如果我们已向您提供，可忽略该步骤。

集成工作

1. 初始化日志（可选）：

```
XP2PService::enableFileLog("log", 1024 * 1024, 5);
```

2. 然后设置一些 SDK 的属性，如包名、版本、appId 等，初始化 SDK。

```
XP2PService::init(appId, appKey, secretKey, package);
```

3. 构造并播放 p2pUrl。
   - 原始 URL：`http://domain/path/to/resource.flv?params=xxx`
   - 构造 URL：

```
XP2PService::host()/live.p2p.com/domain/path/to/resource.flv?
params=xxx&xresid=xxx&xhost=xxx
```

> ⚠ **注意**
> - 由于客户情况不一样，可以咨询我们的工程师，了解如何构造 p2pUrl。
> - 请保持和其他客户端一致，否则各端之间无法进行 P2P。

示例：

```
std::string url = "http://domain/path/to/resource.flv?params=xxx";

std::string streamID = url.substr(url.find_last_of("/")+1);
streamID = streamID.substr(0, streamID.find("."));

std::string xResID = streamID; //该参数标识P2P资源，需要保证唯一
std::string param = "&xresid=" + xResID + "&xhost=txtest-
xp2p.p2p.huya.com";

std::string p2pURl = url.replace(0,
url.find("://")+std::string("://").length(),
XP2PService::host()+"/live.p2p.com/");
p2pURl += param;

// 使用您的播放器播放此url
player.play(p2pUrl);

// 相关资源随着http请求关闭直接释放，无需再做任何处理
// 注意：该http请求可能会返回302
```

API 说明

最近更新时间：2025-06-09 16:18:22

SDK 接口除初始化接口，其余接口均由 HTTP 实现，请求格式为：

```
http:://${host}:${port}/${func}?${param}
```

其中 `http://${host}:${port}` 为本地代理服务器（即 `XP2PService::host()` ）。

统计

接口描述

- 描述：请求对应频道的统计数据
- 方法：GET
- 路径：`/stat?xresid=${yourUrl}`

请求参数

参数名称	必选	类型	说明
xresid	是	string	默认为 URL 中的 resource，如果有 xresid，则用 xresid，否则为频道的 encode url

返回参数

返回码	说明
200	查询成功
404	查询失败，频道不存在

其中，返回码 200 返回的 JSON 内容，格式详细说明如下：

参数名称	类型	说明
flow.p2pBytes	num	对应频道 P2P 流量

flow.cdnBytes	num	对应频道 CDN 流量

示例

- **请求示例：**

```
http://127.0.0.1:16080/live.p2p.com/stat?xresid=${yourURL}
```

> **说明**
> xresid 即 `http://127.0.0.1:16080/live.p2p.com/resoruce.ext` 中的 resource。

- **返回示例：**

```
{"flow":{"p2pBytes":0,"cdnBytes":0}}
```

设置上下行

接口描述

- 描述：请求设置 P2P 上行与下行，0为开启，1为关闭。
- 方法：GET
- 路径：`/feature?download=${0or1}&upload=${0or1}`

请求参数

参数名称	必选	类型	说明
download	是	num	0为关闭；1为开启（默认值）
upload	是	num	0为关闭；1为开启（默认值）

返回参数

返回的 JSON 内容，格式说明如下：

参数名称	必选	类型	说明

ret	是	num	0为正常
msg	是	string	相关信息, 调试使用
upload	是	bool	true 为开启，false 为关闭
download	是	bool	true 为开启，false 为关闭

示例

- **请求示例：**

```
http://127.0.0.1:16080/live.p2p.com/feature?download=1&upload=0
```

> **说明**
> 一般情况下移动网络可以关闭上传。

- **返回示例：**

```
{"ret":0, "msg":"ok", "download":0,"upload":0}
```