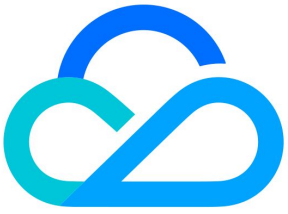
物联网开发平台

应用端开发指南



腾讯云

文档目录

应用端开发指南
腾讯连连小程序自定义 H5 开发
概述

最近更新时间：2024-12-26 18:07:33

腾讯连连 H5 自定义开发是指设备制造商、方案商，将智能化产品通过“腾讯连连”小程序统一管理控制，只需根据腾讯连连 H5 自定义开发规范进行厂商个性化设备操控面板的交互开发，即可直接使用“腾讯连连”提供的所有能力，从而减少用户开发完整独立应用的开发成本。
微信小程序以免下载、即扫即用降低消费者使用智能化产品的门槛，腾讯云物联网提供“腾讯连连”官方小程序，让消费者实现不同品牌、不同品类、不同通信方式设备的互联互通，满足厂商个性化和共享腾讯连连生态的需求。

开发使用参考

- 快速入门：按文档指引流程部署并体验 H5 自定义面板 Demo。
- 开发调试调用
- H5 自定义开发 SDK

快速入门

最近更新时间：2024-11-05 15:02:52

腾讯云物联网开发平台为开发者提供一个 H5 自定义面板 Demo，开发者可以按本文档的指引流程部署并体验 H5 自定义面板 Demo。关于 Demo 的更多信息，请参见 H5 面板 Demo 。

前提条件

注册腾讯云 账号，并完成 实名认证 。

步骤1：创建产品及设备

本步骤指导您通过物联网开发平台的快速入门功能，创建一个智能灯产品及设备，并将模拟设备连接平台进行数据上报。如需自行创建产品和设备，请参见 产品开发 。

1. 登录腾讯云 物联网开发平台控制台 ，选择**公共实例**或您购买的**标准企业实例**，进入**产品开发**。
2. 参考 入门指引 创建产品和设备，并将设备成功接入物联网开发平台和上报数据。

步骤2：配置自定义 H5 面板

1. 下载 H5 面板的 JS 文件 和 CSS 文件 。
2. 登录腾讯云 物联网开发平台控制台 ，选择 步骤1 中创建的项目，进入项目详情页面。
3. 从产品列表中选择 步骤1 中创建的产品，进入产品详情页面。
4. 选择**交互开发** > **面板配置**，单击**配置**，进入面板配置页面。

5. 在面板类型中，选择“H5自定义面板”。

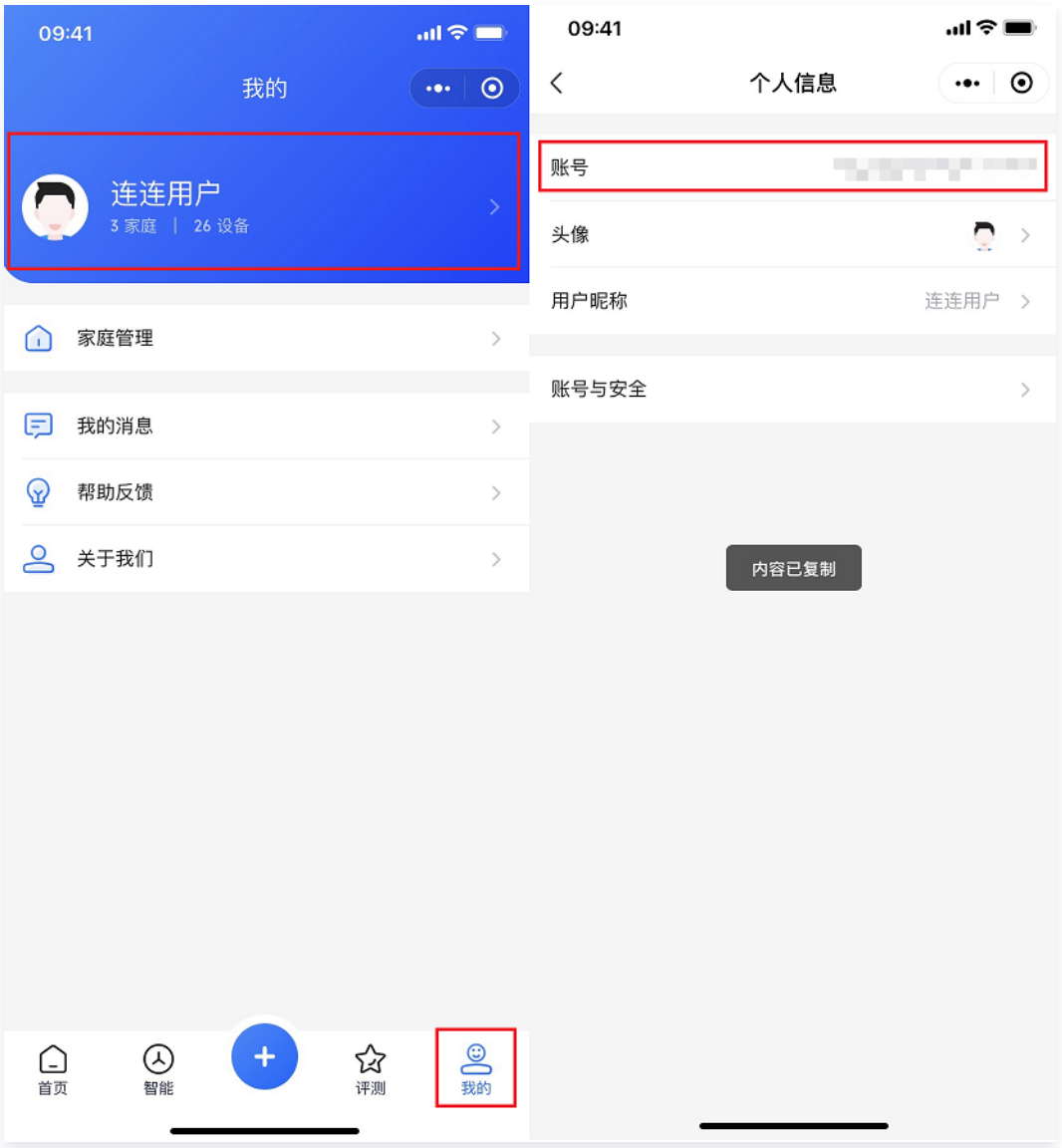
6. 在“上传 JS”表单中，单击**上传**，选择 H5 面板的 JS 文件。上传完成后，页面显示 JS 文件的名称、大小和 MD5 信息。
7. 在“上传 CSS”表单中，单击**上传**，选择 H5 面板的 CSS 文件。上传完成后，页面显示 CSS 文件的名称、大小和 MD5 信息。

步骤3：添加连连用户到 H5 面板访问白名单

开发阶段，需要设置访问白名单才可使用腾讯连连调试 H5 自定义面板。设置访问白名单的操作步骤如下：

1. 打开腾讯连连小程序。
2. 单击**我的**，然后单击页面顶部的个人信息，进入个人信息页面。

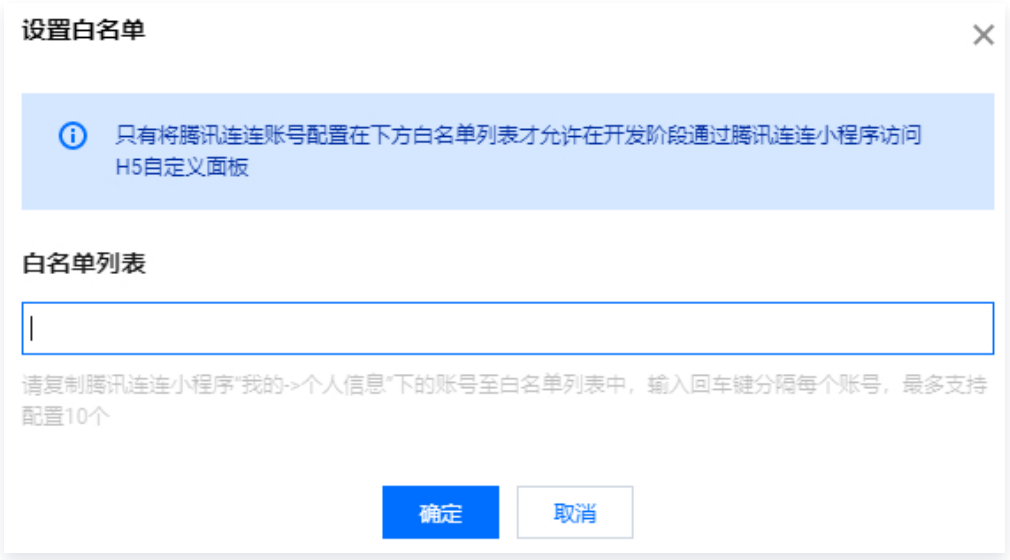
3. 长按**账号**以复制腾讯连连用户 ID，复制成功后页面提示内容已复制。



4. 登录 物联网开发平台控制台，选择对应的项目中及产品。
5. 单击产品中的**交互开发** > **配置小程序** > **面板配置** > **配置**进入面板配置页。

6. 将“面板类型”选择为“H5自定义面板”，单击**设置白名单**，进入设置白名单页面。

7. 在白名单列表中，输入上述步骤复制的腾讯连连用户 ID。



8. 单击**确定**，页面提示保存成功。
9. 关闭“设置白名单”的弹窗后，单击配置页面的**保存**即可。

步骤4：在小程序绑定设备

虚拟设备

1. 在产品详情页面，选择**设备调试** > **虚拟设备调试**，进入虚拟设备调试页面，页面将展示虚拟设备调试的二维码。

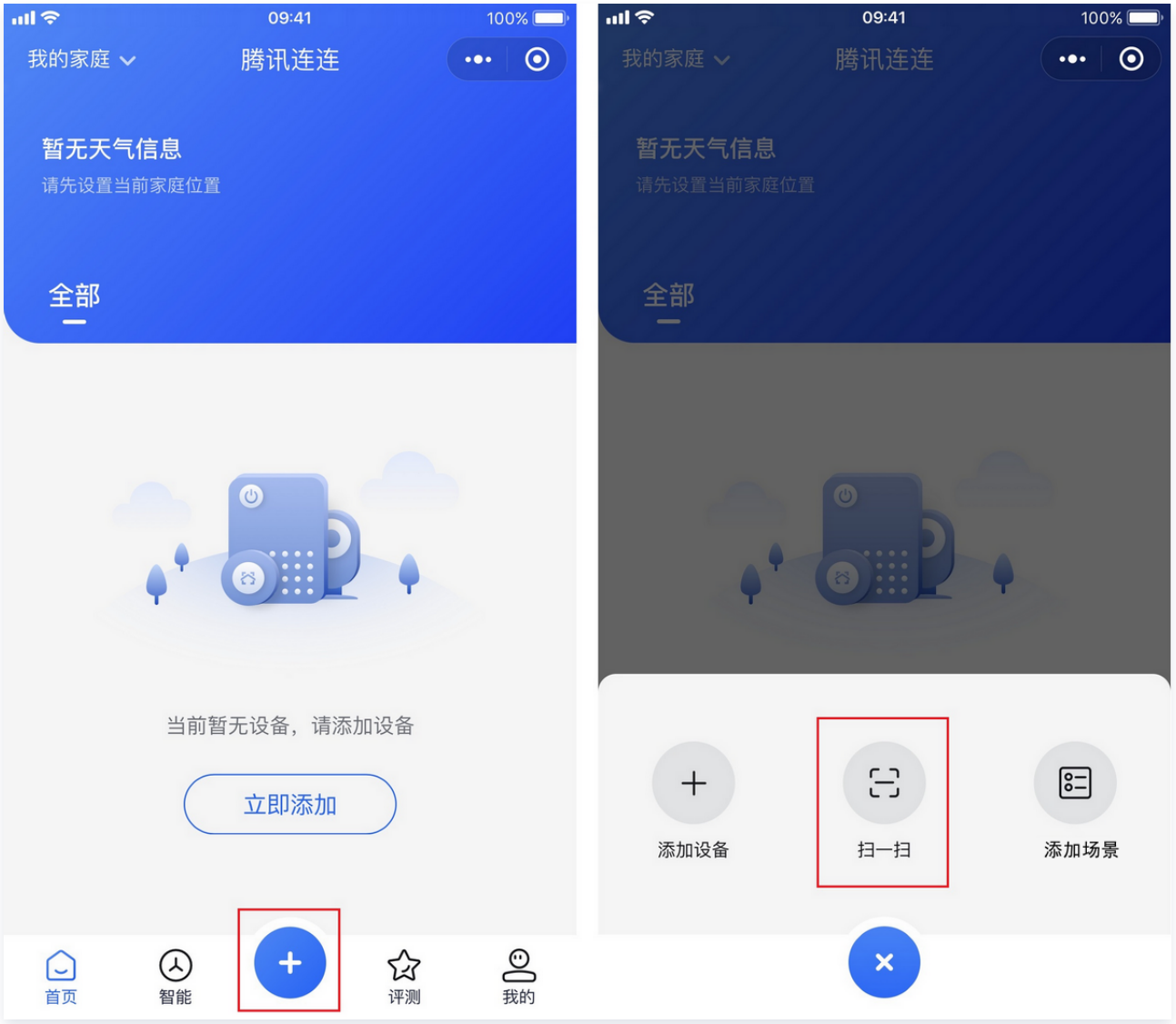
2. 使用微信扫描该二维码，微信即跳转到腾讯连连小程序。授权完成后，腾讯连连小程序自动绑定该设备，页面提示添加设备成功。

真实设备

1. 在产品详情页面，单击**设备调试**，进入设备调试页面，页面展示设备列表。
2. 在设备列表中，选择 步骤1 中创建的设备，单击**二维码**，页面展示设备绑定二维码。

3. 使用微信扫描下方的小程序码，打开腾讯连连小程序。

4. 选择“+” > **扫一扫**，扫描控制台页面展示的二维码。扫描成功后，腾讯连连小程序自动绑定该设备，页面提示添加设备成功。



步骤4：体验自定义 H5 面板

在腾讯连连小程序中，选择 步骤3 中绑定的设备，进入设备控制页面。页面展示自定义 H5 面板，可在该面板进行设备控制等操作。


09:41
100%
智能灯
开关：开
开关
开
颜色
Red
亮度
60%

开发指南
开发调试

最近更新时间：2024-11-26 15:00:53

H5 面板脚手架

我们可以使用以下命令初始化一个 H5 面板项目：

```
npx --package caz@1.0.0 caz tencentyun/iotexplorer-h5-panel-template my-h5-panel
```

如果您用过 vue-cli 等工具来初始化项目，那么这个命令和它很像。这个命令会拉取 iotexplorer-h5-panel-template 作为模板，然后会进行一些特性的选择，例如版本号，选择使用的框架等，如下图所示，最后会初始化一个 H5 面板项目。

```
✔ Project name … my-h5-panel
✔ Project version … 0.1.0
✔ Project description … Awesome h5-panel apps.
✔ 请选一个框架 › vue 3
✔ Choose the features you need ›
✔ Install dependencies … no
Created a new panel project in my-h5-panel by the tencentyun/iotexplorer-h5-panel-template template.

Getting Started:
  $ cd my-h5-panel
  $ npm install # or yarn
  $ npm run dev -- --productId=your_product_id --deviceName=your_devicename # deviceName 参数可选
```

创建项目完成后，开发者可以参考项目目录下的 README.md。在安装完依赖之后，通过以下命令将将项目运行起来：

```
npm run dev -- --productId=your_productId
```

其中 `your_productId` 为 H5 面板对应产品的 productId。

H5 面板 Demo

物联网开发平台提供一个 H5 面板 Demo 供开发者参考，H5 面板 Demo 以开源的方式向开发者开放，便于开发者在开发 H5 面板时进行参考。
H5 面板 Demo 项目，采用 React 框架编写，详情请参见 iotexplorer-h5-panel-demo 。
我们也提供了一个脚手架工具，支持创建 React 或 Vue 框架的 H5 面板 Demo，详情请参见 iotexplorer-h5-panel-template 。

启动调试

H5 面板开发调试时，需要执行以下命令。
1. 安装依赖（下载 Demo 后执行一次即可）。

```
npm install
```

2. 实时编译 H5 面板（开发过程中需要保持运行）。

```
npm run dev
```

该命令执行时，H5 面板的 JS 文件可通过 `http://127.0.0.1:9000/index.js` 访问，CSS 文件可通过 `http://127.0.0.1:9000/index.css` 访问。当 H5 面板源代码文件更改后，将自动重新编译 JS 文件和 CSS 文件，刷新 H5 面板页面即可加载到更新后的 JS 文件和 CSS 文件。

编译构建

在 H5 面板开发完成后，需要执行以下命令构建 H5 面板的 JS 文件和 CSS 文件。

```
npm run release
```

构建完成后，JS 文件和 CSS 文件将会分别输出到 `dist/index.js` 文件和 `dist/index.css` 文件。开发者需要在 物联网开发平台控制台 将这些文件上传到产品的交互开发配置项中。上传操作请参见 上传 H5 面板 。

H5 面板调试

使用浏览器或微信开发者工具的 公众号网页调试 功能访问指定的 URL，可进入 H5 面板的开发模式，对 H5 面板进行调试。
开发模式下，H5 面板框架将分别从以下两个 URL 加载 JS 文件和 CSS 文件。您需要配置代理，使得浏览器或微信开发者工具可通过这两个 URL 加载 H5 面板的 JS 文件和 CSS 文件，从而进行 H5 面板的调试。

- JS 文件 URL： `https://developing.script/developing.js`
- CSS 文件 URL： `https://developing.style/developing.css`

> **说明：**
> 开发模式需要用到您的腾讯云登录态，必须使用对该产品有操作权限的腾讯云账号登录，才可以进入相应产品的 H5 面板开发模式。

本地调试

本地调试将会注册一个与您腾讯云账号关联的腾讯连连用户作为调试用户，并创建一个家庭和一个用于调试的设备，让您无需关注创建家庭、设备绑定等逻辑，只需专注于 H5 面板的开发。
访问以下 URL 可进行 H5 面板的本地调试，请将 ABCDEFG 替换为实际的产品 ID。

```
https://iot.cloud.tencent.com/h5panel/developing?productId=ABCDEFG
```

下面以微信开发者工具进行 H5 面板 Demo 的本地调试为例，描述具体的操作步骤。

步骤1：下载 Demo 项目

1. 拉取 Demo 项目源代码。

```
git clone https://github.com/tencentyun/iotexplorer-h5-panel-demo.git
```

2. 安装依赖。

```
cd iotexplorer-h5-panel-demo
```

```
npm install
```

步骤2：安装并配置 whistle

如果您对 whistle 不熟悉，也可以尝试更简单的 免代理模式，从而跳过步骤2。

1. 安装 whistle，执行如下命令：

```
npm install -g whistle
```

2. 启动 whistle，执行如下命令：

```
w2 start
```

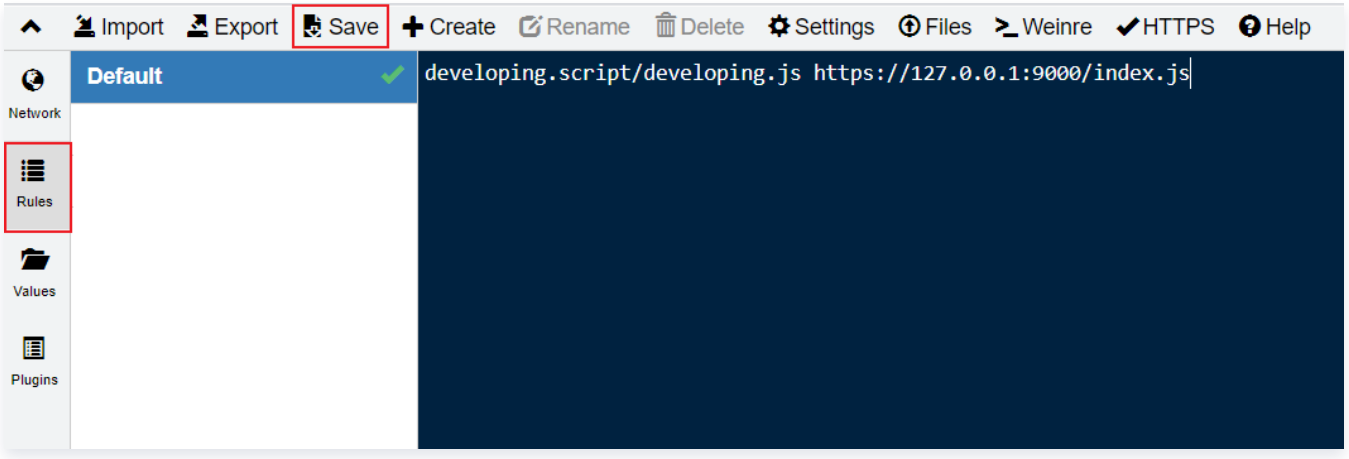
whistle 启动成功后，命令行有如下输出：

```
[i] whistle@2.5.7 started
[i] 1. use your device to visit the following URL list, gets the IP of the URL
you can access:
    http://127.0.0.1:8899/
...（其后输出省略）
```

3. 通过浏览器打开 `http://127.0.0.1:8899/`，进入 whistle 的用户界面。
4. 选择 **Rules** > **Default**，进入规则配置页面。在输入框中填入以下代理规则：

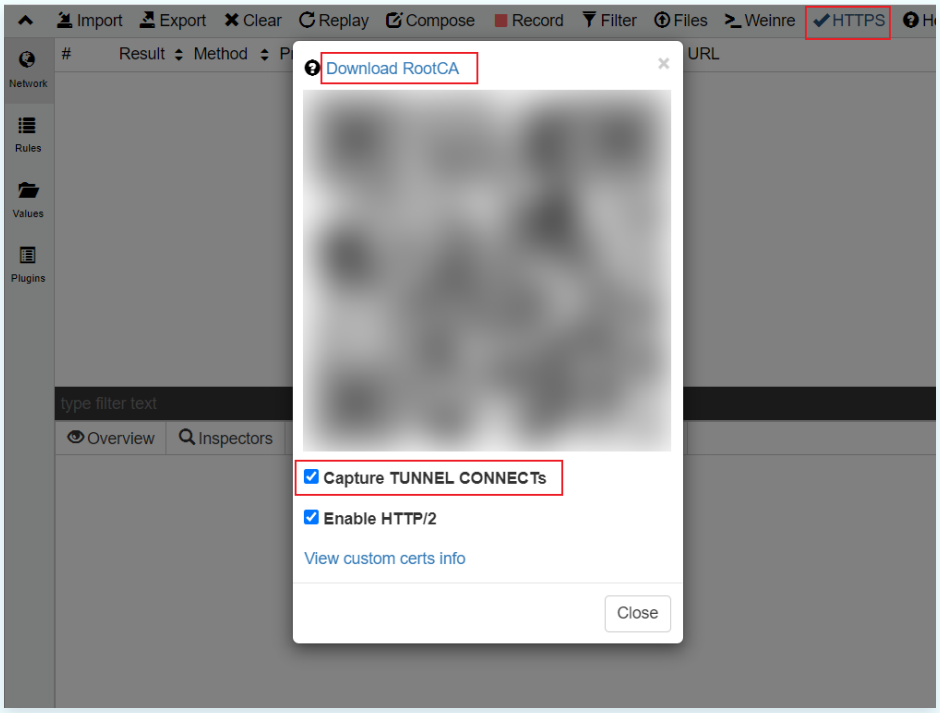
```
developing.script/developing.js http://127.0.0.1:9000/index.js
developing.style/developing.css http://127.0.0.1:9000/index.css
```

单击 **Save**，保存更改内容。



5. 单击菜单栏的 **HTTPS**，进入 HTTPS 配置页面。勾选 “**Capture TUNNEL CONNECTs**”。
6. 单击 **Download RootCA**，下载根证书到本地。

> **注意：**
> 此步骤无需关注二维码信息，单击 **Download RootCA** 并勾选 “**Capture TUNNEL CONNECTs**” 即可。



7. 安装根证书，具体步骤请参见 安装根证书 。

步骤3：启动 Demo 项目的实时编译

在 Demo 项目目录下，执行如下命令：

```
npm run dev
```

编译成功后命令行有如下输出：

```
i ｢wds｣: Project is running at https://localhost:9000/
...（中间输出省略）
i ｢wdm｣: Compiled successfully.
```

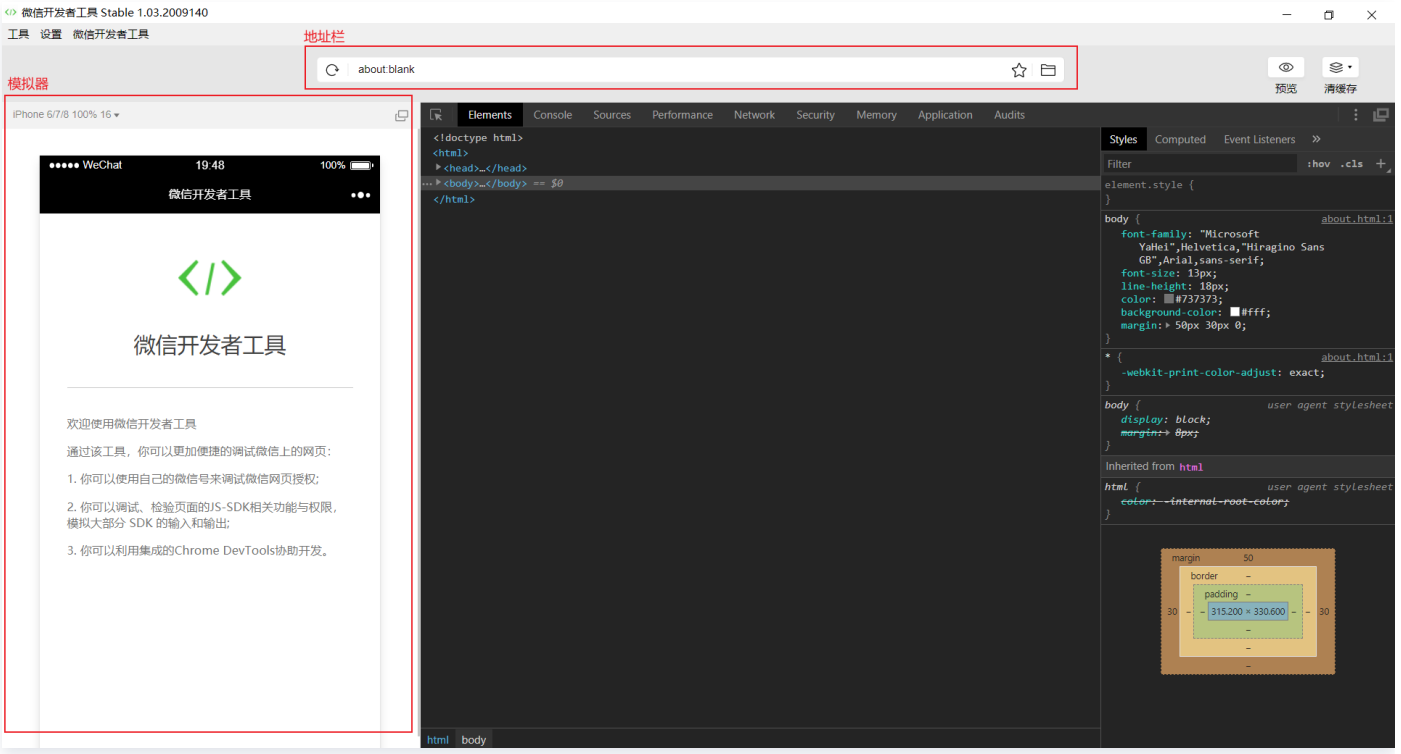
步骤4：配置微信开发者工具

1. 打开微信开发者工具，选择菜单栏的**设置** > **代理** > **代理设置**。

2. 选择**手动设置代理**，代理地址填写 `127.0.0.1` ，端口填写 `8899` 。

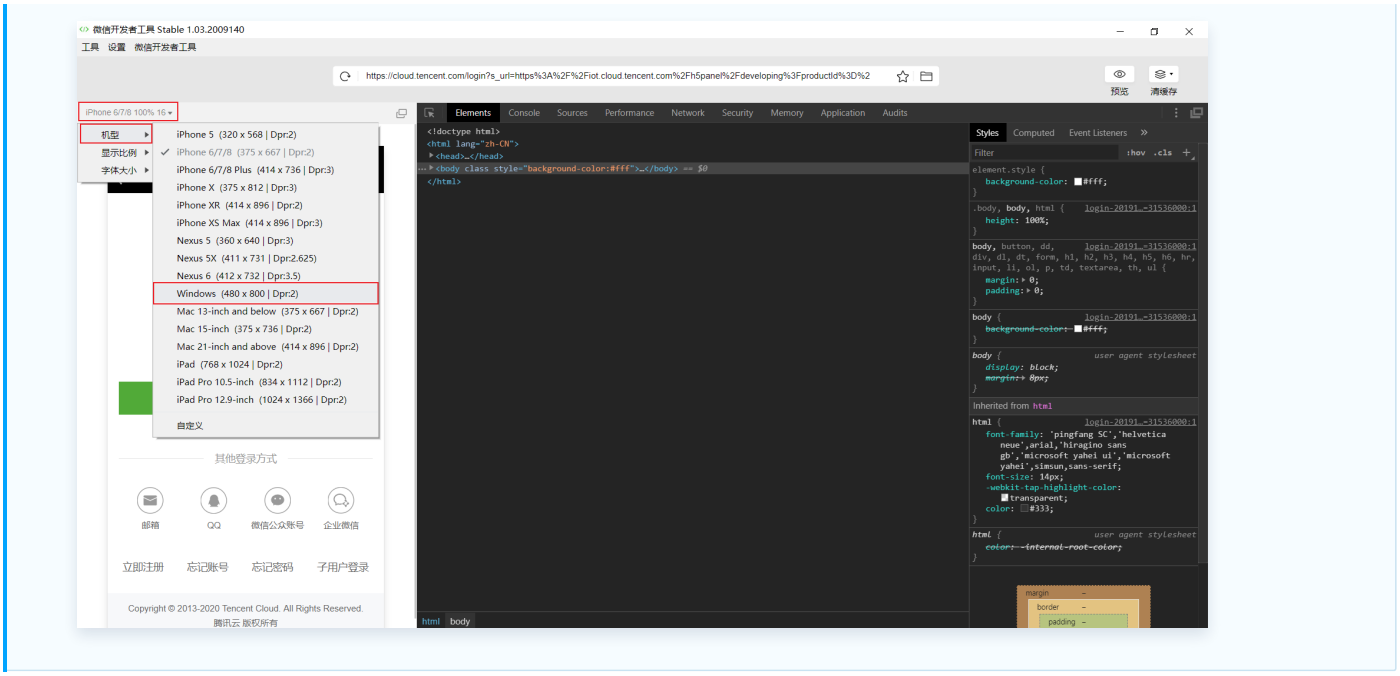
步骤5：微信开发者工具打开 H5 面板

1. 打开微信开发者工具，在新建项目页面，单击公众号网页，进入公众号调试页面。



2. 验证 H5 面板 JS 文件是否可以访问。
   在地址栏输入 `https://developing.script/developing.js` ，按回车键。若模拟器中可加载并显示 JS 文件代码，则表明 H5 面板 JS 文件可以访问。
3. 在地址栏输入 **本地调试** URL，按回车键，模拟器将跳转到腾讯云控制台的登录页面。
4. 在模拟器菜单中选择**机型** > **“Windows (480×800 | Dpr:2)”** 。

> **说明：**
> 若模拟器模拟的是手机环境，则不能正常进行微信或 QQ 登录，此处先更改为 Windows 机型，登录完成后可改回原来的机型设定。



5. 在模拟器页面中登录腾讯云控制台，登录成功后，模拟器将跳转至 H5 面板，可进行 H5 面板调试。

免代理模式

开启免代理模式的方式也很简单，就是在开发面板的链接后面加入一个参数：`&sw=true`，假设我们要开发一个设备，它的 productId 为 ABCDEFG, deviceName 为 dev1，那么通过以下链接即可开启免代理模式

```
- https://iot.cloud.tencent.com/h5panel/developing?productId=ABCDEFG&deviceName=dev1
+ https://iot.cloud.tencent.com/h5panel/developing?
productId=ABCDEFG&deviceName=dev1&sw=true
```

通过加入 `sw=true`，页面会开启一个 service-worker, 通过 service-worker 对请求的拦截和转发，将 `developing.script`，`developing.style` 的请求代理到 localhost 开发服务器。这种方式对于后面的真实设备调试，真机调试同样有效。

免代理模式原理简述

如前文所述，开发模式下，H5 面板框架将分别从以下两个 URL 加载 JS 文件和 CSS 文件：

- JS 文件 URL：`https://developing.script/developing.js`
- CSS 文件 URL：`https://developing.style/developing.css`

而开发服务器打包出来的文件通过 localhost 来访问的，因此我们需要在两者之间进行一个代理，上述的 whistle 就是起着这样的作用。如果不借助 whistle，我们也可以借助浏览器自带的 service-worker，通过监听 `fetch` 事件，进而拦截和修改请求来完成这样的工作，这就是免代理模式。也就是说，所谓免代理模式并不是真正的免代理，只是避免了安装 whistle 这样的工具，而通过浏览器自身的能力(service-worker)实现了请求的代理。

真实设备调试

通过真实设备调试，您可以对指定的设备进行 H5 面板调试。请将 ABCDEFG 替换为实际的产品 ID，abcdefg 替换为实际的设备名称。如需调试物联网开发平台提供的虚拟设备，请将 abcdefg 替换为 `~virtualDev`。

```
https://iot.cloud.tencent.com/h5panel/developing?
productId=ABCDEFG&deviceName=abcdefg
```

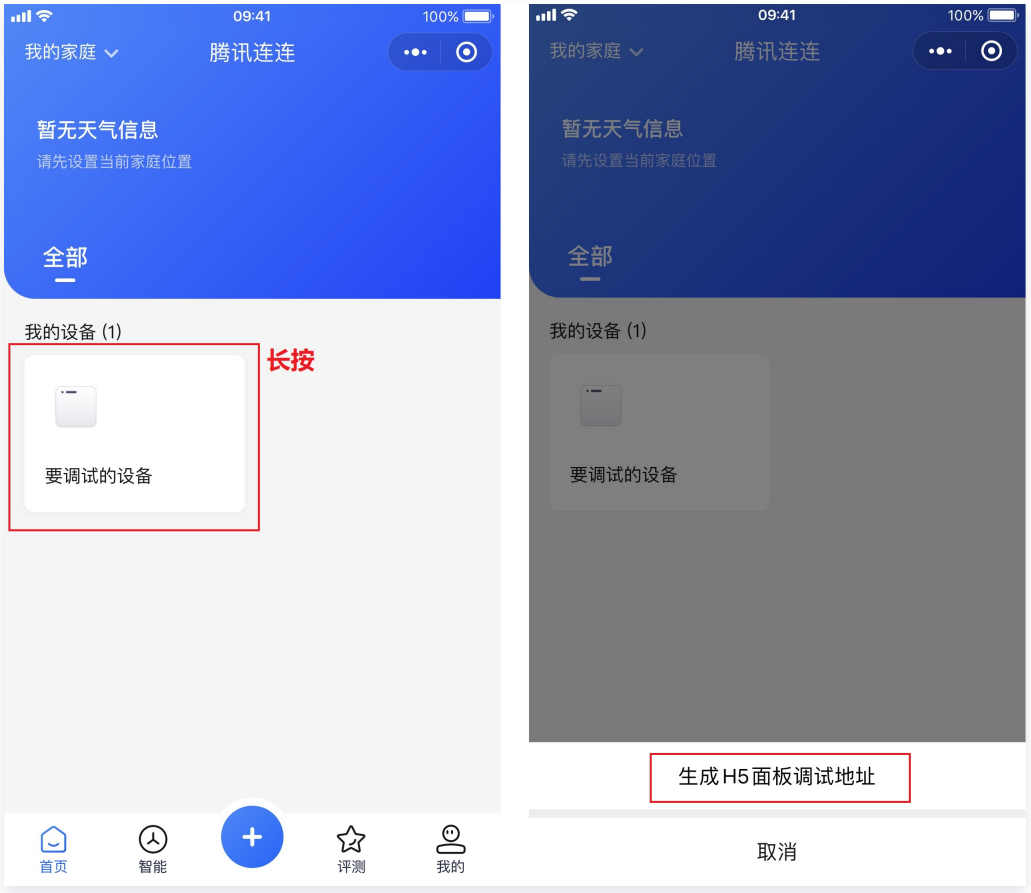
> **注意：**
> 对指定设备进行调试时，会自动将该设备绑定到您的调试用户的家庭下，同时会解除该设备原来的绑定关系。
> “已发布”状态的产品不支持真实设备调试，因为存在将真实用户正在使用的设备解除绑定关系的风险。

真机调试

对于已发布的产品，不开放真实设备调试，且在小程序中调试 H5 面板又非常困难，为了解决这一问题，平台提供真机调试方案。真机调试可以在保持设备原有绑定关系的情况下，对 H5 面板进行调试。在开始之前，需要先使用腾讯连连小程序，将待调试的设备进行绑定。具体操作步骤如下。

1. 打开腾讯连连小程序，选择**我的 > 关于我们**，进入关于我们页面。
2. 长按腾讯连连 Logo，在弹出菜单中选择**进入开发模式**。页面提示进入开发模式成功。

3. 返回腾讯连连小程序首页，在设备列表中，长按需要调试的设备，在弹出菜单中选择“**生成H5面板调试地址**”，之后该调试地址将被复制到剪贴板。

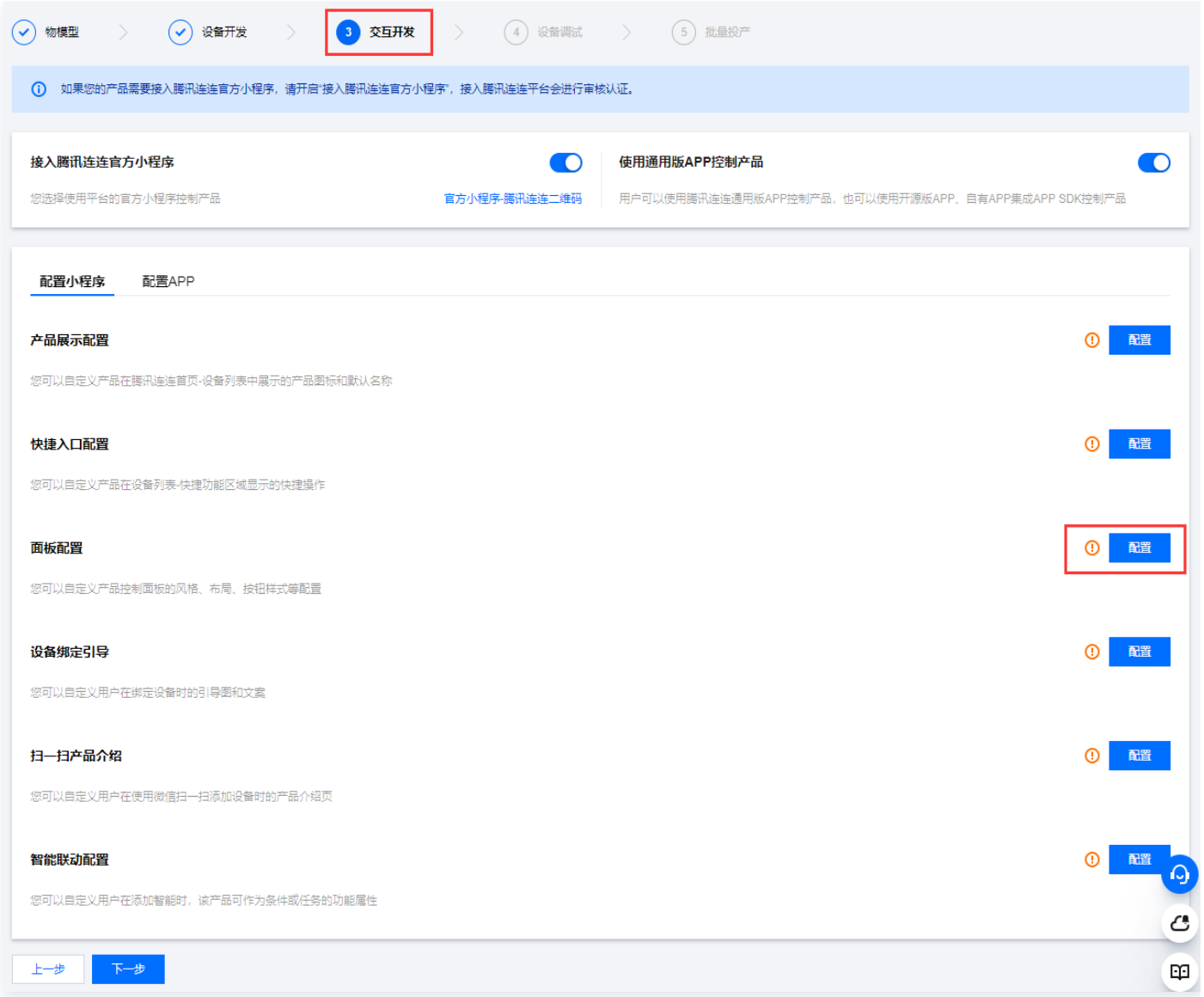


4. 通过浏览器访问该调试地址，可以进行 H5 面板真机调试。

上传 H5 面板

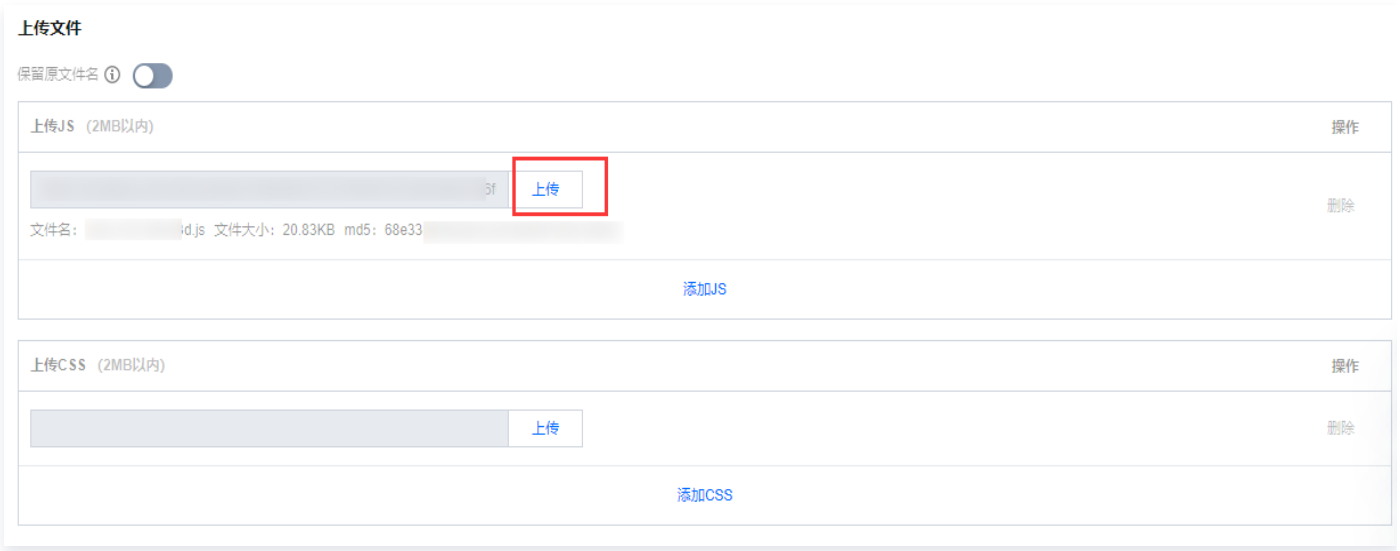
在 H5 面板开发完成以后，需要在控制台上传 H5 面板的 JS 文件和 CSS 文件，具体操作如下：

1. 登录腾讯云 物联网开发平台控制台，选择项目与产品，进入产品详情页面。
2. 选择**交互开发** > **面板配置**，单击**配置**，进入面板配置页面。



3. 在面板类型中选择 **H5自定义面板**。

4. 在**上传 JS**表单中，单击**上传**，选择 H5 面板的 JS 文件。上传完成后，页面显示 JS 文件的名称、大小和 MD5 信息。



5. 如果需要上传 CSS 文件，在“**上传 CSS**”表单中，单击**上传**，选择 H5 面板的 CSS 文件。上传完成后，页面显示 CSS 文件的名称、大小和 MD5 信息。
6. 单击**保存**，页面提示保存成功，上传的 JS 文件与 CSS 文件保存生效。

> **说明：**
> 如果面板中包含小图片，可以通过构建工具内联到代码中，如果图片比较大，建议上传到 COS 后，通过链接引用。

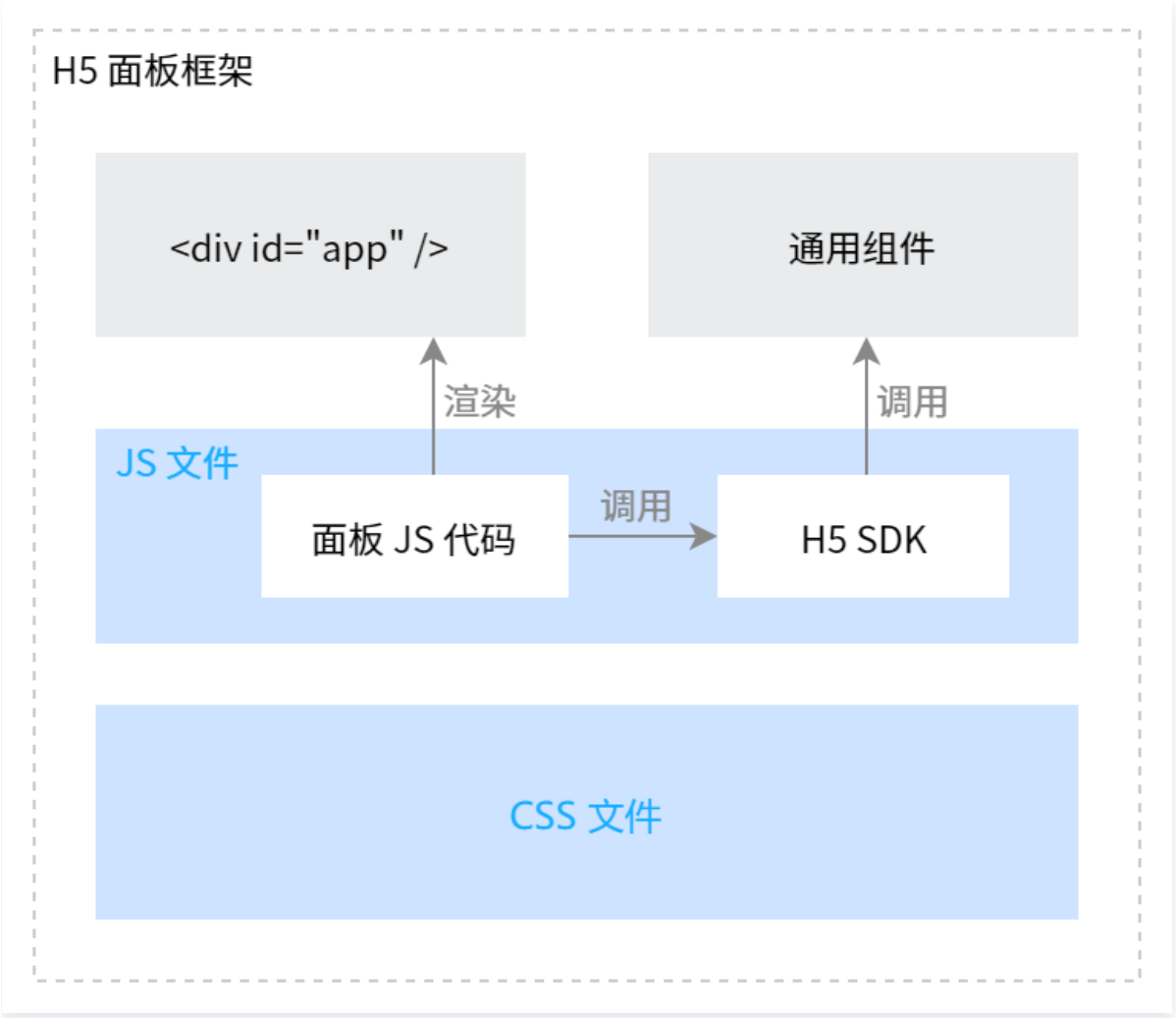
H5 面板访问白名单

开发阶段，需要设置访问白名单才可使用腾讯连连调试 H5 自定义面板。设置访问白名单的操作步骤，详情可参见 快速入门 。

H5 面板原理

最近更新时间：2022-02-15 14:39:37

H5 面板框架组成如下：



- `<div id="app" >` 节点，H5 面板需要渲染到该 DOM 节点下。
- H5 面板的通用组件（例如设备详情页面、模态提示框、顶部提示等），可通过 H5 SDK 调用。
- JS 文件与 CSS 文件由开发者上传。

开发者需要在 JS 文件中实现将 H5 面板渲染到 `<div id="app" >` 节点的逻辑，可以自行选择使用任何技术方案进行前端渲染，最终仅需输出一个 JS 文件和一个 CSS 文件（CSS 文件可选），提供给 H5 面板加载即可。
通过 H5 SDK 可获取 H5 面板所需的参数、调用 H5 面板的标准组件以及平台提供的能力。

调用 H5 SDK 能力

最近更新时间：2025-06-05 14:30:22

引入 H5 SDK

直接通过 Window 访问。

```
window.h5PanelSdk;
```

或 webpack 配置 externals。

```
// webpack.config.js
module.exports = {
  externals: {
    'qcloud-iotexplorer-h5-panel-sdk': 'h5PanelSdk',
  },
};
```

获取基本参数

H5 SDK 提供产品信息、设备数据、用户信息与家庭信息等基本参数供 H5 面板使用，可通过 `sdk.属性名` 取得。具体的参数列表，请参见 基本参数 。

```
console.log(sdk.deviceId);
```

调用应用端 API

关于应用端 API 的信息，请参见 应用端 API 简介 。
H5 SDK 对应用端 API 的调用过程已进行封装，发送请求时将会自动带上公共参数 `AccessToken` 与 `RequestId` 。以下示例代码以调用 获取设备当前状态 应用端 API 为例。

```
const action = 'AppGetDeviceStatuses'; // 请求应用端 API 的 Action 名
const params = { // 请求应用端 API 的参数
  DeviceIds: [sdk.deviceId]
};
sdk.requestTokenApi(action, params).then(data = >  {
  // 请求成功
  console.log(data);
}).catch(err = >  {
  // 请求失败
  console.error(err);
});
```

控制设备

调用 H5 SDK 的 控制设备属性 接口可对指定设备发起控制操作。

```
// 指定要控制的设备的属性数据
const deviceData = {
  light_switch: 0
};

// 控制 H5 面板当前设备
sdk.controlDeviceData(deviceData).then(data => {
  // 请求成功
  console.log(data);
}).catch(err => {
  // 请求失败
  console.error(err);
});

// 控制指定设备
const deviceId = '要控制的设备 ID';
sdk.controlDeviceData(deviceData, deviceId).then(data => {
  // 请求成功
  console.log(data);
}).catch(err => {
  // 请求失败
  console.error(err);
});
```

固件升级

SDK 封装了检查固件更新的接口，以及一个默认样式的固件升级提示。开发者通过调用 H5 SDK 提供的接口，可以检查设备当前是否有可升级固件，以及进行固件升级操作。

检查固件升级

调用 H5 SDK 的 检查设备是否有可升级固件 接口，可以检查指定设备是否有可升级固件。开发者可以直接使用 H5 SDK 提供的默认样式固件升级提示，也可以利用该接口的返回值来实现自定义的固件升级提示，详情如下。

- **方式一：使用 H5 SDK 提供的默认样式固件升级提示**
  执行以下命令调用 `sdk.checkFirmwareUpgrade` 。

  ```
  sdk.checkFirmwareUpgrade();
  ```

若当前设备有可升级固件，且设备在线，则弹出默认样式的固件升级提示。

- **方式二：使用自定义的固件升级提示**
  调用 `sdk.checkFirmwareUpgrade` 时传入 `silent` 参数值为 `true`，则不会弹出上述的默认样式固件升级提示。开发者可根据接口返回的固件升级信息，展示自定义的固件升级提示。其中 `CurrentVersion` 为设备当前固件版本，`DstVersion` 为固件可升级版本。

```
sdk.checkFirmwareUpgrade({
  silent: true // silent 参数为 true 则不弹出 H5 SDK 的默认固件升级提示
}).then(({ CurrentVersion, DstVersion }) => {
  if (DstVersion && CurrentVersion !== DstVersion) {
    // 开发者需要在此处实现自定义的固件升级提示的展示
  }
});
```

进行固件升级

调用 H5 SDK 的 进行固件升级 接口，可对指定设备进行固件升级。调用该接口后，会跳转至小程序的固件升级页面，进行固件升级操作。

```
// 对当前设备进行固件升级
sdk.goFirmwareUpgradePage();

// 对指定设备进行固件升级
sdk.goFirmwareUpgradePage({
  deviceId: '要进行固件升级的设备 ID'
});
```

长连接通信

H5 SDK 初始化时默认自动连接 WebSocket 服务端，并订阅当前设备的上报数据以及状态信息。

```
// 监听设备上报数据推送
sdk.on('wsReport', ({ deviceId, deviceData }) => {
  console.log('websocket device report', deviceId, deviceData);
});

// 监听设备在线状态变更推送
```

```
sdk.on('wsStatusChange', ({ deviceId, deviceStatus }) => {
  console.log('websocket device status change', deviceId, deviceStatus);
});

// 监听设备控制推送
sdk.on('wsControl', ({ deviceId, deviceData }) => {
  console.log('websocket device control', deviceId, deviceData);
});
```

设备详情

H5 SDK 为开发者提供两种设备详情页面，开发者可以根据需要选择使用。

- 标准设备详情：跳转到小程序标准面板的设备详情页面。
- H5 自定义设备详情：在 H5 面板内展示设备详情页面，可在标准设备详情的基础上增加自定义的菜单和按钮。

调用标准设备详情

执行以下代码后，将跳转到小程序标准面板的设备详情页面。

```
sdk.goDeviceDetailPage();
```

调用 H5 自定义设备详情

执行以下代码后，将在 H5 面板内展示设备详情页面。关于 H5 自定义设备详情的接口参数描述，详见 展示 H5 自定义设备详情视图 。

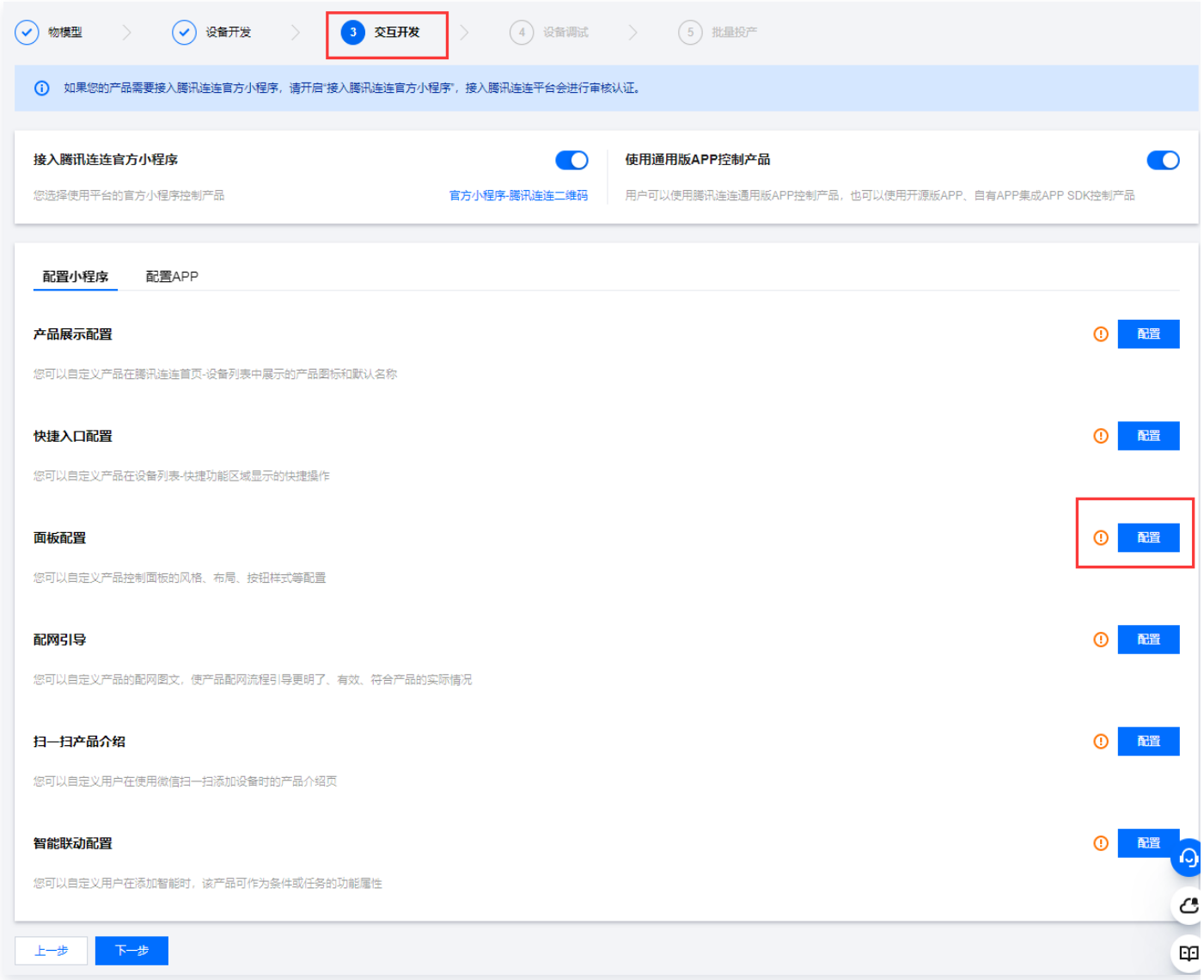
```
sdk.showDeviceDetail({
  labelWidth: 120, // 设备详情的label宽度
  marginTop: 8, // 设备详情的上间距
  shareParams: { // 设备分享时设置自定义分享参数
    a: 'a',
    b: 'b',
  },
  extendItems: [
    {
      labelIcon:
'https://main.qcloudimg.com/raw/be1d876d55ec2479d384e17c94****e69.svg',
      label: '自定义菜单',
      content: '自定义菜单内容（可选）',
      onClick: () => console.log('点击自定义菜单'),
    },
  ],
  extendButtons: [
    {
      text: '自定义按钮',
      type: 'warning',
      onClick: () => console.log('点击自定义按钮'),
```

```
    },
    {
      text: '自定义按钮2',
      type: 'primary',
      onClick: () => console.log('点击自定义按钮2'),
    },
    {
      text: '获取自定义分享参数',
      onClick: async () => {
        const shareParams = await sdk.getShareParams();
        console.log('自定义分享参数: ', shareParams);
      }
    },
    {
      text: '关闭设备详情',
      type: '',
      onClick: () => sdk.hideDeviceDetail(),
    },
  ],
});
```

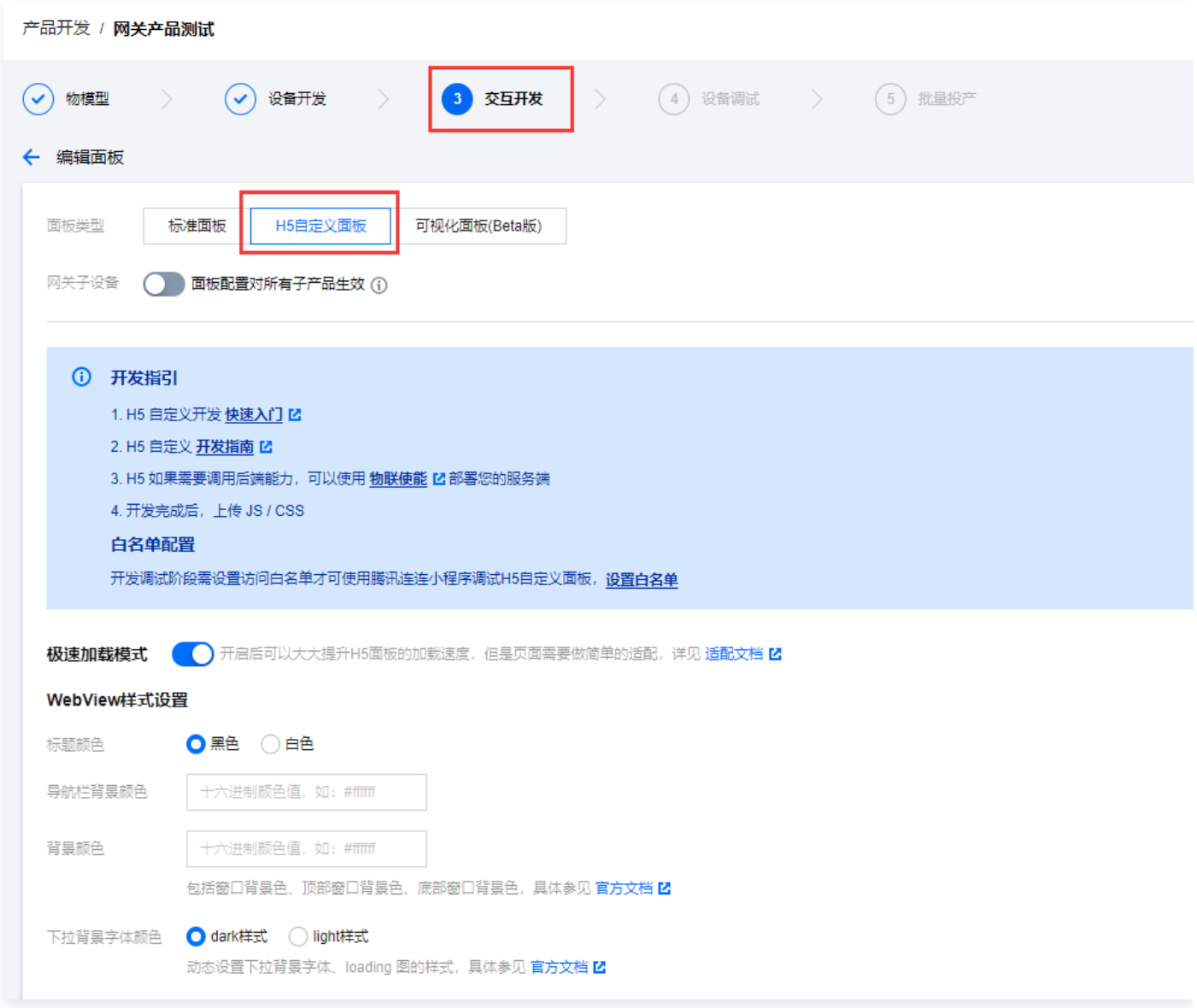
子设备使用网关面板

平台支持设定网关的面板配置对所有子产品生效，开启后，网关下的子产品将使用网关的面板配置。配置步骤如下：

1. 登录腾讯云 物联网开发平台控制台，选择项目与网关产品，进入网关产品的产品详情页面。
2. 选择**交互开发** > **面板配置**，单击**配置**，进入面板配置页面。



3. 在面板类型中选择“**H5自定义面板**”。



4. 开启网关子设备中**面板配置对所有子产品生效**的开关。

监听页面显示、隐藏事件

H5 SDK 提供页面显示、隐藏事件回调。当 H5 页面或小程序切换到前台或后台时，会触发相应的事件。开发者可以通过监听这些事件，在对应事件触发时执行自己的业务逻辑。

```
// 监听 H5 页面切前台
sdk.on('pageShow', () => {
  console.log('on pageShow');
});

// 监听 H5 页面切后台
sdk.on('pageHide', () => {
  console.log('on pageHide');
});

// 监听小程序切前台
sdk.on('appShow', () => {
  console.log('on appShow');
});

// 监听小程序切后台
sdk.on('appHide', () => {
  console.log('on appHide');
});

// H5 页面切前台时，刷新页面标题的设备名称
sdk.on('pageShow', async () => {
  const deviceInfo = await sdk.getDeviceInfo();

  // 设备展示名称
```

```
  const deviceDisplayName = deviceInfo.AliasName || sdk.productInfo.Name;
  // 更新页面标题
  window.document.title = deviceDisplayName;
});
```

蓝牙设备通信

H5 面板内无法直接调用小程序的蓝牙接口，H5 SDK 中已封装 H5 面板与小程序间的蓝牙协议通信机制。要在 H5 面板使用蓝牙通信能力，必须使用 H5 SDK 提供的蓝牙模块。关于蓝牙设备的 H5 面板开发，请参见 蓝牙设备文档 。

自定义蓝牙协议 H5 面板开发

最近更新时间：2024-12-26 18:07:33

功能概述

通信方式为“BLE”，并且设备开发方式为“基于自定义蓝牙协议开发”的产品，搜索设备与设备面板仅支持自定义 h5，需要自行实现。

页面介绍

需要实现两个页面：设备搜索页 、设备面板页。
目前，这两个页面共用 h5 免开发面板 JS 文件，厂商需要自行根据路由处理 JS 中渲染的页面，两个页面的**路由**如下：
调试模式：

- 搜索页：`/h5panel/developing/live/bluetooth-search`
- 面板页：`/h5panel/developing/live`

生产环境：

- 搜索页：`/h5panel/bluetooth-search`
- 面板页：`/h5panel`

渲染页面可以参考 iotexplorer-h5-panel-demo 里 `app.jsx` 中的路由实现。

设备搜索页

主要功能

负责处理蓝牙搜索，过滤出厂商自己的设备，并将设备绑定到用户家庭下。
由于各蓝牙设备协议和广播内容不同，需要厂商自行实现 设备适配器(DeviceAdapter) ，并 添加设备适配器 。设备搜索页可参考 demo 中 BluetoothDemo/SearchPage 页面。

页面入口

详情请见 自定义蓝牙协议设备-搜索设备 H5 页面入口 。

开发流程

1. 腾讯连连进入开发模式（参考 H5 面板调试-真机调试 ）。
2. 打开添加设备页，长按搜索蓝牙设备区域，弹窗中选择 "生成H5蓝牙搜索页调试地址"。
3. 通过手机微信或其他途径，将链接发送到电脑，通过电脑浏览器访问调试链接。链接形式如下，开发者需要自行在链接中填入需要开发的产品 ID：

```
https://iot.cloud.tencent.com/h5panel/developing/live/bluetooth-search?
h5PanelDevId=xxx&productId=
```

4. 同 H5 面板调试，搜索页默认会加载 //developing.script/developing.js 与 //developing.style/developing.css 文件，您需要配置代理转发，将这两个地址转发到您本地开发的 JS 与 CSS 上。

设备面板页

主要功能

与其他设备的 H5 面板一样，开发者可以自定义设备面板页，通过 h5sdk 对设备进行控制。

页面入口

在搜索页成功绑定到家庭后，可以在腾讯连连首页，点击对应设备进入到设备面板页。

开发流程

1. 首先需要完成搜索页的开发，当设备成功绑定到家庭后，才能够进行 H5 面板页的开发。
2. 按照 H5 面板开发 即可。

搜索设备 H5 页面入口

蓝牙搜索 H5 页面入口分为“产品未发布”和“产品已发布”两种情况，具体说明如下。

产品未发布

方式一

1. 登录 控制台 选择对应的项目，单击**产品开发**进入产品列表页面。
2. 选择产品状态为“开发中”的某一产品进入产品详情页，单击**交互开发** > “扫一扫产品介绍”右侧的**配置**进入配置页面。
3. 上传“产品实体图”、“产品备注”等相关信息，单击**保存**后，您可扫描页面右下角提供的二维码进行体验。

方式二

1. 如 真机调试 阐述，进入腾讯连连开发模式。
2. 单击“+” > **添加设备**，长按页面上方的“扫描附近的蓝牙设备”模块，在弹出框内“输入前往 H5 蓝牙搜索页的产品 ID”。

产品已发布

方式一

1. 登录 控制台 选择对应的项目，单击**产品开发**进入产品列表页面。
2. 选择产品状态为“已发布”的某一产品进入产品详情页，单击**批量投产**，您即可扫描页面下方提供的二维码进行体验。

> **注意：**
> 如果扫码打开的不是自定义蓝牙搜索页，而是添加设备的页面，请检查是否已经完成产品的配置。

方式二

您可 提交申请 并通过腾讯连连测试认证流程，成为推荐产品后；产品将会在**添加设备** > **产品分类列表**中展示供您选择。


电工照明
安防报警
家用电器
网络设备
厨房电器
运动健康
影音办公
推荐
遥控大师空调伴侣
华来小方智能彩灯带STP01
窗帘
五路排插
灯
插座
开关面板
插排
五路灯
排插

极速模式

最近更新时间：2022-03-04 18:08:13

功能概述

H5 加载时，需要在服务端拉取面板所需要的所有数据，导致加载速度较慢。为了解决这个问题，我们推出了极速模式，其原理是加载时服务端不去拉取数据，而是由 H5 打开后由小程序客户端将本地的数据通过我们特有的链路推送到 H5，可以大大加速加载时间，但是也由于从原本的直出时同步准备好所有数据改成了页面渲染后异步加载数据，所以渲染时依赖直出数据的面板需要做一定的适配才能够正常使用。

适配方法

仅需要渲染页面之前先等待一个 sdkReady 的 Promise 即可，具体实现可参考以下 React 实现的示例：

```
function Page() {
    const [sdkReady, setSdkReady] = useState(false);
    useEffect(() => {
        h5PanelSdk.sdkReady().then(() => setSdkReady(true));
    }, []);

    return !sdkReady ? <div >loading...</div > : (
        <div >
            {* 正常渲染您的页面 *}
        </div >
    );
}
```

开启说明

由于极速模式是针对产品级生效，存量产品请务必全量发布适配版本后，再开启极速模式，否则可能会影响到老版本用户的正常使用。

H5 自定义开发 SDK
基本参数

最近更新时间：2024-10-04 11:34:21

基本参数

H5 SDK 提供产品信息、设备数据、用户信息与家庭信息等基本参数供 H5 面板使用，可通过 `sdk.属性名` 取得。

属性名	类型	描述
deviceId	string	设备 ID，由 `{productId}/{deviceName}` 组成。
productId	string	产品 ID。
deviceName	string	设备名称
deviceInfo	DeviceList | ShareDevices	设备信息，数据结构同 DeviceList；如果是分享设备，则数据结构同 ShareDevices 。
roomList	RoomList	当前家庭的房间列表。
roomName	string	当前设备的房间名称。
dataTemplate	string	设备所在产品的物模型，与**数据模板** > **查看JSON** 中展示的物模型定义一致。
deviceStatus	number	设备在线状态。 • 在线：1。 • 非在线：0。
deviceDisplayName	string	设备展示名称，会依次取：AliasName > productInfo；name > deviceName 来展示。
isShareDevice	boolean	是否是分享设备。
familyId	string	设备所在家庭 ID，如果是分享设备则无此值。
roomId	string	设备所在房间 ID，如果是分享设备则无此值。
familyInfo	FamilyList	设备所在家庭详情，如果是分享设备则无此值。
isFamilyOwner	boolean	用户是否是当前家庭的所有者。
userInfo	object	用户信息。 • Avatar：string；头像 URL。 • CountryCode：string；国家代码。 • Email：string；邮箱地址。 • NickName：string；昵称。 • PhoneNumber：string；手机号码。

		• UserID：string；用户 ID。

设备管理

最近更新时间：2024-11-26 15:00:53

获取产品信息

- **接口定义**

```
sdk.getProductInfo({ productId?: string }) => Promise<{
    ProductId: string,
    Name: string,
    Description: string,
    DataTemplate: string,
    NetType: string,
    CategoryId: number,
    ProductType: number,
    UpdateTime: number,
}>
```

- **参数说明**

参数名	参数描述	类型	必填
productId	可选，不传则使用当前设备的产品 ID。	string	否

- **返回值**

返回一个 Promise，输出参数如下：

参数名	参数描述	类型
ProductId	产品ID。	string
Name	产品名称。	string
Description	产品描述。	string
DataTemplate	产品数据模板。	string
NetType	通信方式（子设备为接入网关协议）。	string
CategoryId	产品分类 ID。	number
ProductType	产品类型（0：普通产品；5：网关产品）。	number
UpdateTime	最后更新的 Unix 时间戳（秒级）。	number

获取设备信息

- **接口定义**

```
sdk.getDeviceInfo({ deviceId?: string }) => Promise<{
    ProductId: string,
    DeviceName: string,
    DeviceId: string,
    IconUrl: string,
    AliasName: string,
    UserId: string,
    RoomId: string,
    CreateTime: number,
    UpdateTime: number
} | {
    ProductId: string,
    DeviceName: string,
    DeviceId: string,
    IconUrl: string,
    AliasName: string,
    CreateTime: string
}>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

  返回一个 Promise，输出参数请参见 DeviceList（非分享设备）和 ShareDevices（分享设备）。

获取绑定到网关下的子设备列表

这个接口只能对网关设备使用。

- **接口定义**

```
sdk.getSubDeviceList() => Promise<{
    subDeviceList: Device[],
    syncFailList: Device[],
}>
```

- **返回值**

  返回一个 Promise， `subDeviceList` 为该网关下已绑定到家庭的子设备列表， `syncFailList` 为网关下没有绑定到家庭的子设备列表, Device 的数据结构详见 deviceList。

控制设备属性

- **接口定义**

```
sdk.controlDeviceData(data, deviceId?: string) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
data	要控制的设备属性数据。	object	是
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise，输出参数请参见 用户控制设备 。

调用设备行为

- **接口定义**

```
sdk.callDeviceAction(actionPayload: ActionPayload, actionId: string, deviceId?:
string) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
actionPayload	物模型中定义的行为调用入参。	object	是
actionId	在物模型中定义的该行为的标志符。	string	是
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise，输出参数请参见 同步调用设备行为 。

获取设备物模型数据

- **接口定义**

```
sdk.getDeviceData({ deviceId?: string }) => Promise<object>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise，输出参数为设备的物模型数据。

获取设备物模型历史数据

- **接口定义**

```
sdk.getDeviceDataHistory({
    FieldName: string,
    MaxTime: number,
    MinTime: number,
    Context?: string,
    Limit: number
}) => Promise<{
    RequestId: string,
    Context: string,
    FieldName: string,
    Listover: boolean,
    Results: DataHistoryItem[]
}>
```

- **参数说明**

参数名	参数描述	类型	必填
FieldName	查询的属性名称。	string	是
MaxTime	结束时间，毫秒时间戳。	number	是
MinTime	开始时间，毫秒时间戳。	number	是
Context	翻页游标，首次查询时可不传。	string	否
Limit	单页数据量。	number	是

- **返回值**

  返回一个 Promise，输出参数请参见 获取设备物模型历史数据 。

获取设备当前状态

- **接口定义**

```
sdk.getDeviceStatus({ deviceId?: string }) => Promise<0 | 1>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

  返回一个 Promise，输出结果取值如下：
  - 0：设备离线。
  - 1：设备在线。

删除设备

- **接口定义**

```
sdk.deleteDevice({ deviceId?: string }) => Promise<void>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise。

获取自定义分享参数

若该设备是分享设备，且分享方设置了自定义分享参数，则被分享方在接受分享后可通过该接口获取自定义分享参数。

- **接口定义**

```
sdk.getShareParams({ deviceId?: string }) => Promise<any>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise，输出参数为自定义分享参数。

检查设备是否有可升级固件

检查指定设备是否有可升级固件，若有可升级固件，且设备在线，则弹出固件升级提示。

- **接口定义**

```
sdk.checkFirmwareUpgrade({
    deviceId?: string,
    silent?: boolean
}) => Promise<{
    CurrentVersion: string,
    DstVersion: string,
}>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

silent	可选，默认为 false。 • true：只检查固件升级，不弹出固件升级提示。 • false：检查固件升级，若有可升级固件，且设备在线，则弹出固件升级提示。	boolean	否

- **返回值**

  返回一个 Promise，输出参数请参见 查询设备固件是否升级 。

获取最新固件信息

获取控制台推送的固件信息。

- **接口定义**

```
sdk.getDeviceLatestOTAInfo({ deviceId?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

  返回一个 Promise，输出参数如下：

参数名	参数描述	类型
TargetVersion	目标固件版本。	string
FirmwareURL	固件下载地址。	string

下载固件

下载控制台推送的固件

- **接口定义**

```
sdk.downLoadDeviceOTA({ deviceId?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

  返回一个 Promise，输出参数如下：

参数名	参数描述	类型

otaData	ArrayBuffer格式的固件。	ArrayBuffer
deviceId	设备 ID。	string

固件版本上报

上报设备端固件版本

- **接口定义**

```
sdk.reportFirmwareVersion({ deviceId?: string, version: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否
version	当前设备的固件版本。	string	是

固件升级状态上报

上报设备端固件升级状态

- **接口定义**

```
sdk.reportOTAStatus({ deviceId?: string, state: string, persent: string, version:
string, resultCode: number, resultMsg?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否
state	• downloading：表示下载固件中。 • burning：表示烧录中。 • done：表示完成 • fail：表示失败 。	string	是
persent	固件升级进度 0 - 100。	number	是
version	当前设备的固件版本。	string	是
resultCode	错误码。 • -1：下载超时。 • -2：文件不存在。 • -3：签名过期。 • -4：MD5不匹配。 • -5：更新固件失败。	number	是

resultMsg	错误消息，如果失败时必填，错误原因。	string	否

固件升级信息查询

查询设备的固件升级信息

- **接口定义**

```
sdk.checkFirmwareUpdate({ ProductId: string, DeviceName: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
ProductId	产品id	string	是
DeviceName	设备名称	string	是

- **返回值**

返回一个 Promise，输出参数如下：

参数名	参数描述	类型
CurrentVersion	设备当前版本	string
DstVersion	下发的版本	string

进行固件升级

跳转到小程序的固件升级页面，进行固件升级。

- **接口定义**

```
sdk.goFirmwareUpgradePage({ deviceId?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID	string	否

- **返回值**

返回一个Promise

用户管理

最近更新时间：2022-02-15 14:40:08

获取用户信息

- **接口定义**

```
sdk.getUserInfo() => Promise<{
    Avatar: string,
    CountryCode: string,
    Email: string,
    NickName: string,
    PhoneNumber: string,
    UserID: string
}>
```

- **返回值**

返回一个 Promise，输出参数如下：

参数名	参数描述	类型
Avatar	头像 URL。	string
CountryCode	国家代码。	string
Email	邮箱地址。	string
NickName	昵称。	string
PhoneNumber	手机号码。	string
UserID	用户 ID。	string

界面组件

最近更新时间：2024-12-26 18:07:33

tips 组件

tips 组件，样式和风格与腾讯连连小程序一致。

基础用法

- **接口定义**

```
sdk.tips.show(message, options) => Promise
```

- **参数说明**

参数名	类型	必填	参数描述
message	string	是	提示文本。
options.type	• 'info' • 'danger' • 'loading' • 'success'	否	tips 类型。
options.waitForHide	boolean	否	若为 true，则 show 方法返回一个 Promise，并且当关闭后才会触发 resolve。
options.duration	number	否	展示提示的时间，单位毫秒，默认1500。
options.delayDuration	number	否	默认为0，单位毫秒，提示会在该延时后展示。
options.canClickClose	boolean	否	默认为 true，点击 mask 是否能够关闭提示。
options.canBeReplace	boolean	否	默认为 false，为 false 时上一个提示未关闭前，再次调用 tips.show 会被忽略。

关闭提示

接口定义

```
sdk.tips.hide() => Promise
```

显示加载提示

封装后的 `tips.show` 方法，等价于：

```
sdk.tips.show(message, {
```

```
  type: 'loading',
  canBeReplace: true,
  duration: 0,
  delayDuration: 200,
  canClickClose: false,
  ...options,
});
```

- **接口定义**

```
sdk.tips.showLoading(message, options) => Promise
```

- **参数说明**

  请参见 展示 tips 参数说明 。

关闭加载提示

展示 loading tips 后必须主动调用本接口，否则 tips 将会一直保留。

接口定义

```
sdk.tips.hideLoading() => Promise
```

显示成功提示

封装后的 `tips.show` 方法，等价于：

```
sdk.tips.show(message, { type: 'success', ...options });
```

- **接口定义**

```
sdk.tips.showSuccess(message, options) => Promise
```

- **参数说明**

  请参见 展示 tips 参数说明 。

显示一般提示

封装后的 `tips.show` 方法，等价于：

```
sdk.tips.show(message, { type: 'info', ...options });
```

- **接口定义**

```
sdk.tips.showInfo(message, options) => Promise
```

- **参数说明**

请参见 展示 tips 参数说明 。

显示错误提示

先标准化处理错误展示信息，然后展示错误 tips。等价于：

```
sdk.tips.show(getErrorMsg(error), { type: 'error', ...options });
```

- 接口定义

```
sdk.tips.showError(error, options) => Promise
```

- 参数说明

参数名	类型	必填	参数描述
error	• Error • { code, msg } • string	是	错误对象或错误信息。
options	object	否	请参见 展示 tips 参数说明 。

模态对话框

基础用法

展示一个弹窗，参数、功能、样式同小程序原生 showModal 基本一致。

- 接口定义

```
sdk.tips.showModal({
      title?: string,
      content?: string,
      showCancel?: boolean,
      cancelText?: string,
      cancelColor?: string,
      confirmText?: string,
      confirmColor?: string,
}) => Promise<boolean>
```

- 参数说明

参数名	类型	必填	参数描述
title	string	否	弹窗标题。
content	string	否	弹窗内容。
showCancel	boolean	否	是否展示取消按钮，默认为 true。

cancelText	string	否	取消按钮文案，默认为“取消”。
cancelColor	string	否	取消按钮颜色，默认为“#6C7078”。
confirmText	string	否	确认按钮文案，默认为“确定”。
confirmColor	string	否	确认按钮颜色，默认为“#0066FF”。

- **返回值**

  返回一个 `Promise<boolean>`，输出结果取值如下：
  - true：用户点击确认
  - false：用户点击取消

确认模态对话框

基于 `sdk.tips.showModal` 封装，用于向用户进行二次确认操作时使用。

- **接口定义**

```
sdk.tips.confirm(title, content, options) => Promise<boolean>
```

- **参数说明**

参数名	类型	必填	参数描述
title	string	否	弹窗标题。
content	string	否	弹窗内容。
options	object	否	请参见 展示模态对话框 。

- **示例代码**

```
const isConfirm = await sdk.tips.confirm('确认删除该设备吗？');
if (isConfirm) {
  // 用户确认，执行后续流程
}
```

提示模态对话框

基于 `sdk.tips.showModal` 封装，用于向用户进行消息提示操作时使用。

- **接口定义**

```
sdk.tips.alert(content, options) => Promise<boolean>
```

- **参数说明**

参数名	类型	必填	参数描述

content	string	否	弹窗内容。
options	object	否	请参见 展示模态对话框 。

- **示例代码**

  向用户进行消息提示。

```
await sdk.tips.alert('该功能暂时无法使用，请稍后再试');
```

设备离线提示组件

设备离线提示组件，样式和风格与腾讯连连小程序一致。

基础用法

- **接口定义**

```
// 调用方式1
sdk.offlineTip.show() => void
// 调用方式2
sdk.showOfflineTip() => void
```

关闭提示

- **接口定义**

```
// 调用方式1
sdk.offlineTip.hide() => void
// 调用方式2
sdk.hideOfflineTip() => void
```

H5 自定义设备详情视图

显示设备详情视图

在当前 H5 展示一个铺满全屏的设备详情视图，支持增加自定义菜单项及按钮。

- **接口定义**

```
sdk.showDeviceDetail({
    deviceInfo?: object,
    labelWidth?: number,
    marginTop?: number,
    shareParams?: object,
    extendItems?: ExtendItemConfig[],
    extendButtons?: ExtendButtonConfig[],
    containerClassName?: string
}) => void
```

- **参数说明**

参数名	类型	必填	参数描述
extendItems[].label	string	是	自定义菜单项的标题。
extendButtons[].text	string	是	自定义按钮文案。
extendButtons[].onClick	function	是	自定义按钮点击后触发的回调。
deviceInfo	object	否	展示详情的设备信息，不传则使用当前设备信息。
labelWidth	number	否	设备详情的 label 宽度，默认110，单位 px。
marginTop	number	否	设备详情的上间距，默认10，单位 px。
shareParams	• object • string	否	自定义分享参数。
extendItems	ExtendItemConfig[]	否	自定义菜单配置。
extendItems[].labelIcon	string	否	展示在 label 前的 icon 地址。
extendItems[].content	string	否	自定义菜单项的内容。
extendItems[].className	string	否	自定义菜单项的样式类名。
extendItems[].onClick	function	否	点击自定义菜单项后触发的回调。
extendButtons	ExtendButtonConfig[]	否	自定义按钮配置。
extendButtons[].className	string	否	自定义按钮的样式类名。
extendButtons[].type	• 'danger' • 'primary' • 'warning'	否	自定义按钮的风格。
containerClassName	string	否	容器的样式类名。

关闭设备详情视图

接口定义

```
sdk.hideDeviceDetail() => void
```

调用小程序能力

最近更新时间：2024-10-08 18:37:11

跳转小程序的标准设备详情页面

- **接口定义**

```
sdk.goDeviceDetailPage({
    reload?: boolean,
    deviceId?: string,
    isShareDevice?: boolean,
    shareParams?: object | string,
}) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
reload	• 如果为 true，则进入详情页后会重新拉取一次该设备的数据。 • 如果为 false，则进入详情页后会使用缓存的设备数据。	boolean	否
deviceId	可选，不传则使用当前设备的设备 ID。	string	否
isShareDevice	可选，设备是否为分享设备，不传则使用当前的 sdk.isShareDevice。	boolean	否
shareParams	可选，设备自定义分享参数。	• object • string	否

- **返回值**

返回一个 Promise。

跳转小程序的网关添加子设备页面

- **接口定义**

```
sdk.goGatewayAddSubDevicePage(gatewayDeviceId: string);
```

- **参数说明**

参数名	参数描述	类型	必填
gatewayDeviceId	网关设备 ID。	string	是

- **返回值**

返回一个 Promise。

跳转小程序的反馈页面

- **接口定义**

```
sdk.goFeedBackPage() => Promise
```

- **返回值**

返回一个 Promise。

跳转小程序的设备信息页面

- **接口定义**

```
sdk.goDeviceInfoPage({ deviceId?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise。

跳转小程序的修改设备名称页面

- **接口定义**

```
sdk.goEditDeviceNamePage({ deviceId?: string, name?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否
name	可选，不传则使用当前设备的 aliasName。	string	否

- **返回值**

返回一个 Promise。

跳转其他设备的面板页面

- **接口定义**

```
sdk.goDevicePanelPage(deviceId: string, customParams?: { passThroughParams:
Record<string, string> }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	要跳转面板的设备 ID。	string	是
passThroughParams	会被解析成查询字符串后，拼接到待跳转的面板的地址上。	Record<string, string>	否

- **返回值**

返回一个 Promise。

跳转小程序的房间设置页面

- **接口定义**

```
sdk.goRoomSettingPage({ deviceId?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise。

跳转小程序的设备分享页面

- **接口定义**

```
sdk.goShareDevicePage({ deviceId?: string }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否

- **返回值**

返回一个 Promise。

小程序刷新数据

要求小程序在当前 H5 面板关闭后进行一次数据刷新。

- **接口定义**

```
sdk.reloadAfterUnmount() => Promise
```

- **返回值**

  返回一个 Promise。

跳转到其他小程序

跳转到另外一个小程序,适用于面板内购买增值服务等场景，入参与说明详见 wx.navigateToMiniprogram 。

> **注意：**
> 只有将要目标小程序的 appid，加入白名单后才能调用此接口，如有需求请需联系客服开通。

- **接口定义**

```
sdk.navigateToMiniprogram({ appid, extraData, envVersion }) => Promise
```

- **返回值**

  返回一个 Promise。

返回小程序的上一级页面

可用于主动关闭 H5 面板。

- **接口定义**

```
sdk.navBack() => Promise
```

- **返回值**

  返回一个 Promise。

设置当前页面的分享内容

设置当前页面的分享内容，通过 wx.miniProgram.postMessage 向小程序推送分享信息，具体参考 小程序页面分享文档 。

- **接口定义**

```
sdk.setShareConfig({ title: string, imgUrl: string? }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
title	分享的标题。	string	是
imgUrl	分享图片的地址，默认会取当前页面截图。	string	否

- **返回值**

  返回一个 Promise。

跳转云端定时

- **接口定义**

```
sdk.goTimingProjectPage({ deviceId, isShareDevice, featureId } = {})
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	可选，不传则使用当前设备的设备 ID。	string	否
isShareDevice	可选，设备是否为分享设备，不传则使用当前的 sdk.isShareDevice。	boolean	否
featureId	物模型标识符，传入参数后只会筛选出对应标识符的定时任务。	string	否

- **返回值**

无返回值。

通知小程序触发震动

- **接口定义**

```
sdk.triggerVibrateShort(type)
```

- **参数说明**

参数名	参数描述	类型	必填
type	震动强度类型，有效值为：heavy、medium、light。	string	是

- **返回值**

无返回值。

跳转到video设备的面板页面

- **接口定义**

```
sdk.goVideoPanelPage({ deviceId, passThroughParams, redirect })
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	要跳转面板的设备ID，不传取当前设备ID。	string	否
passThroughParams	会被解析成查询字符串后，拼接到待跳转的面板的地址上。	Record	否
redirect	跳转的方式， true 时为 redirectTo，false 时 navigateTo。	boolean	否

- **返回值**

  无返回值。

应用端 API

最近更新时间：2024-12-26 18:07:33

在开发 H5 面板的过程中，我们需要获取产品信息，设备信息，或者获取设备在线状态，控制设备，这些能力都通过应用端 API 来提供，完整的 API 列表可以参考 应用端 API 。H5 自定义开发 SDK 对调用应用端 API 的过程进行了封装，从而方便开发者进行调用。

调用应用端 API

H5 SDK 对应用端 API 的调用过程进行了封装，发送请求时会自动带上公共参数 `AccessToken` 与 `RequestId` 。

- **接口定义**

```
sdk.requestTokenApi(action, data, options) => Promise
```

- **参数说明**

参数名	类型	必填	参数描述
action	string	是	具体应用端 API Action 名称，如：`AppGetDeviceStatuses` 。
data	object	否	接口调用参数。
options	object	否	请参见 wx.request 。

- **返回值**
  - 请求成功（code=0）

    返回一个 resolved 的 Promise，其值为应用端 API 响应中的 `Response` 部分数据。
  - 请求失败

    返回一个 rejected 的 Promise，其值的数据结构为：`{ code, msg, ...detail }` 。

事件监听

最近更新时间：2024-12-26 18:07:33

监听事件

- **接口定义**

```
sdk.on(type: string, listener: (...args) => void) => void
```

- **参数说明**

参数名	类型	参数描述
type	string	要监听的事件。
listener	(...args) => void	事件触发时的回调函数。

取消监听事件

- **接口定义**

```
sdk.off(type: string, listener: (...args) => void) => void
```

- **参数说明**

参数名	类型	参数描述
type	string	要取消监听的事件。
listener	(...args) => void	要取消监听的事件的回调函数，不传则清除该事件的所有回调函数。

WebSocket 事件

wsClose 事件

WebSocket 的 `close` 事件。

参数名	类型	参数描述
code	number	服务器发送的关闭码。
reason	string	服务器关闭连接的原因。

wsError 事件

WebSocket 的错误事件。

wsControl 事件

当 WebSocket 收到 `control` 指令后触发。

参数名	类型	参数描述
deviceId	string	设备 ID
deviceData	object	设备数据

wsReport 事件

当 WebSocket 收到 `report` 指令后触发。

参数名	类型	参数描述
deviceId	string	设备 ID
deviceData	object	设备数据

wsStatusChange 事件

当 WebSocket 收到 `wsStatusChange` 指令后触发。

参数名	类型	参数描述
deviceId	string	设备 ID
deviceStatus	number	设备在线状态，0离线，1在线。

wsEventReport 事件

当 WebSocket 收到 `wsEventReport` 指令后触发。

参数名	类型	参数描述
deviceId	string	设备 ID。
Payload	object	设备回复详情。

wsActionPush 事件

当 WebSocket 收到 `wsActionPush` 指令后触发。

参数名	类型	参数描述
deviceId	string	设备 ID
Payload	object	下发的 Action 指定详情。

wsActionReport 事件

当 WebSocket 收到 `wsActionReport` 指令后触发。

参数名	类型	参数描述

deviceId	string	设备 ID
Payload	object	设备回复的 Action 响应详情。

前后台切换事件

appShow 事件

当小程序 App.onShow 执行后触发。

appHide 事件

当小程序 App.onHide 执行后触发。

pageShow 事件

当小程序 Page.onShow 执行后触发。

pageHide 事件

当小程序 Page.onHide 执行后触发。

蓝牙模块

最近更新时间：2024-11-14 15:47:32

名词解释

名词	含义
serviceId	服务 id，蓝牙服务的 uuid，搜索设备时主要通过 `serviceId` 来过滤我们需要的设备。
deviceId	小程序 API 搜索出来的设备的标识，连接设备时主要通过 `deviceId` 来标识需要连接的设备。
explorerDeviceId	物联网开发平台侧定义的设备 ID，查询设备数据和上报设备数据时以设备 ID 作为设备标识。
BlueToothAdapter 蓝牙适配器	全局单例，实例上声明了蓝牙搜索、连接等方法。
DeviceAdapter 设备适配器	真正用来连接设备以及跟设备进行通信的模块，每一个设备连接对应一个设备适配器实例，设备适配器会在连接设备后实例化，并在设备断开连接后销毁。根据不同的 `serviceId` 来区别不同类型设备的适配器构造函数。

蓝牙适配器

添加设备适配器

添加一个设备适配器。使用蓝牙模块时需要根据具体设备情况创建一个设备适配器，并调用本接口将其构造函数添加到蓝牙适配器中。H5 SDK 默认添加了一个支持 LLSync 蓝牙协议的设备适配器。

- **接口定义**

```
sdk.blueToothAdapter.addAdapter(deviceAdapter: DeviceAdapterConstructor) => void
```

- **参数说明**

参数名	参数描述	类型	必填
deviceAdapter	要添加的设备适配器的构造函数。	DeviceAdapterConstructor	是

- **示例代码**

```
class DemoDeviceAdapter extends DeviceAdapter {
    static serviceId = '0000FFF0-0000-1000-8000-00805F9B34CC';
    static deviceFilter(deviceInfo) {
        if (deviceInfo.advertisServiceUUIDs) {
            const matchedServiceId = deviceInfo.advertisServiceUUIDs.find(id =>
id === DemoDeviceAdapter.serviceId);
            if (matchedServiceId && deviceInfo.advertisData) {
```

```
                try {
                    const macArr = deviceInfo.advertisData.slice(2);
                    const mac = macArr.join(':');
                    return {
                        ...deviceInfo,
                        deviceName: mac,
                        serviceId: matchedServiceId,
                    };
                } catch (err) {
                    console.error('parse mac error', err);
                }
            }
        }
    }
    handleBLEMessage(hex) {
        return {
            type: 'unknown',
            data: hex,
        };
    }
}
sdk.blueToothAdapter.addAdapter(DemoDeviceAdapter);
```

初始化蓝牙模块

初始化蓝牙模块，包括初始化蓝牙模块、打通小程序间蓝牙通信以及注册全局回调等。本接口可重复调用，可在每次使用蓝牙模块前调用。

- **接口定义**

```
sdk.blueToothAdapter.init() => Promise<void>
```

- **返回值**
  返回一个带缓存的 Promise，初始化成功后 resolve。若初始化未完成或已初始化成功，则多次调用后返回同一个 Promise。若初始化失败，则该缓存的 Promise 在 reject 之后会被释放，再次调用则将重新初始化。
- **示例代码**

```
sdk.blueToothAdapter.init().then(() => {
  // 调用蓝牙模块能力
});
```

开始搜索蓝牙设备

开始搜索蓝牙设备。（将会调用 wx.startBluetoothDevicesDiscovery，比较耗费系统资源，务必在不需要搜索后调用 停止搜索蓝牙设备，如离开页面后）。

- **接口定义**

```
sdk.blueToothAdapter.startSearch({
```

```
    serviceId?: string,
    serviceIds?: string[],
    ignoreDeviceIds?: string[],
    onSearch: (DeviceInfo[]) => void,
    onError: (Error) => void,
    timeout: number
}) => Promise<void>
```

- **参数说明**

参数名	参数描述	类型	必填
serviceId	指定需要搜索的 serviceId，不传的话会使用当前支持的所有 DeviceAdapter 的 serviceId 来匹配。	string	否
serviceIds	参数描述同 serviceId。	string[]	否
ignoreDeviceIds	可选，需要过滤掉的 deviceId 列表（例如刚添加完的设备），搜索结果中将不会出现这些设备。	string[]	否
onSearch	当搜索结果更新后调用，返回搜索到的设备列表。	(DeviceInfo[]) => void	是
onError	当搜索过程中发生错误后调用，触发后设备搜索将会中止。	(Error) => void	是
timeout	可选，默认20000，单位毫秒，超过指定时长没有搜索到设备后将会触发超时错误。	number	是

- **返回值**

  返回一个 Promise。

停止搜索蓝牙设备

接口定义

```
sdk.blueToothAdapter.stopSearch() => void
```

搜索单个蓝牙设备

开始搜索蓝牙设备，并在搜索到第一个满足条件的设备后停止搜索。

- **接口定义**

```
sdk.blueToothAdapter.searchDevice({
    deviceName: string,
    serviceId?: string,
    serviceIds?: string[],
    ignoreDeviceIds?: string[]
}) => Promise<DeviceInfo>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceName	指定需要搜索的设备 deviceName。	string	是
serviceId	指定需要搜索的 serviceId，不传的话会使用当前支持的所有 DeviceAdapter 的 serviceId 来匹配。	string	否
serviceIds	参数描述同 serviceId。	string[]	否
ignoreDeviceIds	可选，需要过滤掉的 deviceId 列表（例如刚添加完的设备），搜索结果中将不会出现这些设备。	string[]	否

- **返回值**

  返回一个 Promise，在找到第一个满足条件的设备后 resolve。

连接蓝牙设备

连接指定蓝牙设备。

- **接口定义**

```
blueToothAdapter.connectDevice(deviceInfo: DeviceInfo, options?: { autoNotify?: boolean }) => Promise<DeviceAdapter>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceInfo	传入 开始搜索蓝牙设备 或 搜索单个蓝牙设备 接口搜索出来的 DeviceInfo。	DeviceInfo	是
options.autoNotify	可选，默认为 true。指定为 true 时，在连接设备后，会自动去拉取服务列表，以及主服务下的特征值列表，并会自动订阅第一个 notifyId 或 indicateId 特征值的 notify。若设备含有多个服务或多个 notify 特征值，请传 false，并自行通过 getBLEDeviceServices、getBLEDeviceCharacteristics、notifyBLECharacteristicValueChange 等方法获取及订阅特征值。	boolean	否

- **返回值**

  返回一个 Promise，连接成功后返回设备适配器。

获取设备适配器实例

根据 `deviceId` 查询对应的设备适配器实例。

- **接口定义**

```
sdk.blueToothAdapter.getDeviceAdapter({ explorerDeviceId: string }) => DeviceAdapter
```

- **参数说明**

参数名	参数描述	类型	必填
explorerDeviceId	设备适配器实例的 explorerDeviceId,可以通过 `sdk.deviceId` 获得。	string	是

- **返回值**

返回与传入 `explorerDeviceId` 相匹配的设备适配器实例，如果找不到，则返回 `undefined` 。

上报设备信息

蓝牙设备不能通过mqtt直接上报设备的mac地址等信息，所以需要H5端进行上报，对应的是设备详情里面的设备信息

> **注意：**
> 图片里面厂家名称和产品型号是在设备量产时在控制台填写的，mac 地址，固件版本等由 H5 端进行上报。

- **接口定义**

```
sdk.blueToothAdapter.reportDeviceInfo({ productId: string, deviceName: string,
deviceInfo: any }) => Promise;
```

- **参数说明**

参数名	参数描述	类型
productId	产品 ID。	string
deviceName	设备名称。	string
deviceInfo	设备信息。	any，详情见下面示例

```
deviceInfo: {
```

```
    "module_hardinfo": "模组具体硬件型号 N10",
    "module_softinfo": "模组软件版本",
    "fw_ver": "mcu 固件版本",
    "imei": "设备 imei 号，可选上报",
    "mac": "设备 mac 地址，可选上报",
    "device_label": {
    "append_info": "设备商自定义的产品附加信息"
    }
}
```

监听蓝牙适配器事件

监听蓝牙适配器事件。

- **接口定义**

```
sdk.blueToothAdapter.on(type: string, listener: (...args) => void) => void
```

- **参数说明**

参数名	参数描述	类型	必填
type	要监听的事件。	string	是
listener	事件触发时的回调函数。	(...args) => void	是

取消监听蓝牙适配器事件

取消监听蓝牙适配器事件。

- **接口定义**

```
sdk.blueToothAdapter.off(type: string, listener: (...args) => void) => void
```

- **参数说明**

参数名	参数描述	类型	必填
type	要取消监听的事件。	string	是
listener	要取消监听的事件的回调函数，不传则清除该事件的所有回调函数。	(...args) => void	否

蓝牙适配器事件

adapterStateChange 事件：当适配器状态变化时触发。

参数名	参数描述	类型
available	蓝牙适配器是否可用	boolean

discovering	蓝牙适配器是否处于搜索状态	boolean

设备适配器

自定义设备适配器

自定义设备适配器类需要继承 `DeviceAdapter`，并补充以下实现。

- **serviceId**：自定义设备适配器类需要设置该属性，代表该设备的主服务 ID。
- **deviceFilter**：自定义设备适配器类需要实现该静态方法，在搜索蓝牙设备时会将每个搜出的设备信息传入该方法，如果判断是本产品的设备，则需在除入参 deviceInfo 之外返回设备唯一标识 deviceName 及 serviceId，否则返回空。

```
DeviceAdapter.deviceFilter: (deviceInfo: DeviceInfo) => {
    deviceName: string,
    serviceId: string,
    ...deviceInfo
}
```

- **handleBLEMessage**：自定义设备适配器类需要实现该方法，用于处理收到 `onBLECharacteristicValueChange` 回调后的协议解析。

```
DeviceAdapter.handleBLEMessage: (hexString, { serviceId, characteristicId }) => {
    reportData?: any,
    ...any
}
```

返回值中如果返回 `reportData`，则会将该部分数据上报到云端（注意需与产品定义物模型匹配），其他字段则会透传到 `message` 事件的 payload 中。

设备适配器属性

属性名	属性描述	类型
explorerDeviceId	只读，设备的 explorerDeviceId。	string
isConnected	只读，当前是否已连接设备。	boolean
deviceId	只读，设备的 deviceId。	string
serviceId	只读，设备的主服务 ID，与构造函数上的静态属性 DeviceAdapter.serviceId 一致。	string
originName	只读，设备的原始名称，即小程序接口搜索出来时的 name 字段。	string

断开设备连接

接口定义

```
deviceAdapter.disconnectDevice() => void
```

获取设备服务列表

- **接口定义**

```
deviceAdapter.getBLEDeviceServices() => Promise<ServiceList>
```

- **返回值**

  返回一个 Promise，`ServiceList` 数据结构请参见 wx.getBLEDeviceServices 。

获取设备指定服务的特征值列表

获取设备指定服务中的特征值列表，并将特征值存放在 `deviceAdapter` 实例上。

- **接口定义**

```
deviceAdapter.getBLEDeviceCharacteristics({ serviceId: string }) =>
Promise<CharacteristicsList>
```

- **参数说明**

参数名	参数描述	类型	必填
serviceId	可选，指定要获取特征值列表的服务 ID，默认为主服务 ID。	string	否

- **返回值**

  返回一个 Promise，`CharacteristicsList` 数据结构请参见 wx.getBLEDeviceCharacteristics 。

- **说明**

  将获取到的特征值按照如下数据结构存放在 `deviceAdapter` 实例上。

```
deviceAdapter.characteristicsMap[serviceId] = {
    notifyIds: string[],
    indicateIds: string[],
    writeIds: string[],
    readIds: string[]
}
```

协商设置蓝牙最大传输单元

本接口仅支持在 Android 系统下调用，iOS 因系统限制不支持。

- **接口定义**

```
deviceAdapter.setBLEMTU({ mtu: number }) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
mtu	最大传输单元。设置范围为 (22,512) 区间内，单位为 bytes。	number	是

- **返回值**

  返回一个 Promise。

读取指定特征值的二进制数据

- **接口定义**

```
deviceAdapter.readBLECharacteristicValue({
    serviceId?: string,
    characteristicId: string
}) => Promise<void>
```

- **参数说明**

参数名	参数描述	类型	必填
serviceId	可选，默认为主服务 ID。	string	否
characteristicId	需要读取的特征值 ID，默认会取主服务下的第一个 read 特征值。	string	是

- **返回值**

  返回一个 Promise。接口读取到的信息需要在 `onBLECharacteristicValueChange` 方法注册的回调中获取，具体参见 wx.readBLECharacteristicValue 。

获取蓝牙设备信号强度

- **接口定义**

```
deviceAdapter.getBLEDeviceRSSI() => Promise
```

- **返回值**

  返回一个 Promise，具体数据结构请参见 wx.getBLEDeviceRSSI 。

启用特征值变化时的 notify 功能

- **接口定义**

```
deviceAdapter.notifyBLECharacteristicValueChange({
    characteristicId?: string,
    serviceId?: string,
    state?: boolean
}) => Promise<void>
```

- **参数说明**

参数名	参数描述	类型	必填
serviceId	需要订阅的服务 ID，默认会取主服务 ID。	string	否
characteristicId	需要订阅的特征值 ID，默认会取主服务下的第一个 notify 或 indicate 特征值。	string	否
state	是否启用 notify，默认为 true。	boolean	否

- **返回值**

  返回一个 Promise。

写入二进制数据到指定特征值中

- **接口定义**

```
deviceAdapter.write(hexString: string, options?: {
    writeId?: string,
    serviceId?: string
}) => Promise
```

- **参数说明**

参数名	参数描述	类型	必填
hexString	需要写给蓝牙设备的十六进制字符串。	string	是
options.writeId	可选，需要写入的特征值 ID，默认会取主服务下的第一个 writeId。	string	否
options.serviceId	可选，需要写入的服务 ID，默认会取主服务 ID。	string	否

监听事件

可以使用 `deviceAdapter.on` 来 监听设备适配器事件 。

- **接口定义**

```
deviceAdapter.on(type: string, listener: (...args) => void) => void
```

- **参数说明**

参数名	参数描述	类型	必填
type	要监听的事件。	string	是
listener	事件触发时的回调函数。	(...args) => void	是

取消监听事件

取消监听设备适配器事件。

- **接口定义**

```
deviceAdapter.off(type: string, listener: (...args) => void) => void
```

- **参数说明**

参数名	参数描述	类型	必填
type	要取消监听的事件。	string	是
listener	要取消监听的事件的回调函数，不传则清除该事件的所有回调函数。	(...args) => void	否

设备适配器事件

- **connect 事件**：设备连接后触发。
- **disconnect 事件**：设备断开后触发。
- **message 事件**：当收到 `onBLECharacteristicValueChange` 回调，并经过 `handleBLEMessage` 处理后触发。

参数名	参数描述	类型
timestamp	收到设备消息的时间戳，单位毫秒。	number
dataReported	收到设备的消息是否已上报云端。	boolean
（其他）	handleBLEMessage 函数返回的其他参数将会透传到 message 事件中。	any

- **bLEConnectionStateChange 事件**：当 `onBleConnectionStateChange` 触发时触发，若 `connected` 为 true，则接下来会触发 `connect` 事件，否则会触发 `disconnect` 事件。

参数名	参数描述	类型
connected	设备是否连接。	boolean

标准蓝牙协议

LLSync 标准蓝牙协议设备的绑定流程在小程序中进行，开发者只需在自定义 H5 面板中关注设备的连接、控制操作与事件监听。自定义 H5 面板已默认添加 LLSync 标准蓝牙协议的设备适配器（StandardDeviceAdapter），支持通过 LLSync 标准蓝牙协议与设备通信。H5 面板 Demo 中提供了 标准蓝牙协议 demo 供开发者参考。

搜索并连接设备

与标准蓝牙协议设备建立连接的过程分为三个步骤：搜索设备、连接设备、连接鉴权。示例代码如下：

```
import { StandardDeviceAdapter } from 'qcloud-iotexplorer-h5-panel-sdk';

// 初始化蓝牙适配器
await sdk.blueToothAdapter.init();
```

```
// 搜索设备
const deviceInfo = await sdk.blueToothAdapter.searchDevice({
    deviceName: sdk.deviceName,
    serviceId: StandardDeviceAdapter.serviceId,
    productId: sdk.productId,
    disableCache: true,
});

if (!deviceInfo) {
    throw new Error('未搜索到设备');
}

// 连接设备
const deviceAdapter = await sdk.blueToothAdapter.connectDevice({
    ...device,
    productId: sdk.productId,
});

// 连接鉴权
if (!deviceAdapter.authorized) {
    await deviceAdapter.authenticateConnection({
        deviceName: sdk.deviceName,
    });
}
```

设备通信

通过上述流程与设备建立连接后，H5 面板即可控制设备，接收设备的属性、事件上报。

- **控制设备**

  接口定义请参见 控制设备属性，示例代码如下：

  ```
  sdk.controlDeviceData({
      // 要控制的设备属性
      power_switch: 1
  });
  ```

- **监听设备事件**

  接口定义请参见 事件监听，示例代码如下：

  ```
  // 监听设备属性上报
  sdk.on('wsReport', ({ deviceId, deviceData }) => {
      console.log('device', deviceId, 'report_property', deviceData);
  });

  // 监听设备事件上报
  sdk.on('wsEventReport', ({ deviceId, Payload }) => {
      console.log('device', deviceId, 'report_event', Payload);
  ```

```
});
```

设备适配器属性

属性名	属性描述	类型
explorerDeviceId	只读，设备在物联网开发平台的 DeviceId。	string
isConnected	只读，当前是否已连接设备。	boolean
deviceId	只读，小程序蓝牙接口提供的 deviceId。	string
serviceId	只读，设备的主服务 ID，与构造函数上的静态属性 DeviceAdapter.serviceId 一致。	string
originName	只读，设备的原始名称，即小程序接口搜索出来时的 name 字段。	string
bleVersion	只读，LLSync 协议版本号。	number
mtu	只读，设备要求设置的最大传输单元 (MTU) 大小。	number
otaVersion	只读，设备固件版本号。	string
extendInfo.macStr	只读，设备的 mac 地址。	string
authorized	只读，是否与设备完成鉴权。	boolean

BLE + WIFI 双路通信

针对 Wi-Fi + BLE Combo 模组，提供设备端 SDK 和 H5 SDK，支持设备 Wi- Fi 离线状态下，小程序通过 LLSync 标准蓝牙协议与设备通信，为用户提供 Wi-Fi 断网下的更佳体验。设备端 SDK 请参考 开发指引 。
对于自定义 H5 面板，配网流程无需开发者关注，开发者需要在面板中关注：

1. 监听设备在线状态，wifi连接是否正常。
2. 当设备离线时启用蓝牙进行通信。下面分开说明。

监听设备在线状态

我们可以通过监听 WebSocket 事件 中的 `wsStatusChange` 事件来感知设备的在线离线状态。

```
sdk.on('wsStatusChange', ({deviceId, deviceStatus}) => {
    if (deviceStatus === 0) {
        // 设备已离线，开始启用蓝牙连接进行通信，见第二步
    }
})
```

启用蓝牙通信

我们提供了 双路通信 demo 供参考。

1. 添加 BleComboDualModeDeviceAdapter4H5

   BleComboDualModeDeviceAdapter4H5 是一个设备适配器实例，如果您还不了解设备适配器，请阅读 设备适配器部分 。

```
import { BleComboDualModeDeviceAdapter4H5 } from 'qcloud-iotexplorer-appdev-plugin-wificonf-blecombo/lib/protocols/BleComboDualMode';

const sdk = window.h5PanelSdk;
BleComboDualModeDeviceAdapter4H5.injectOptions({ appDevSdk: sdk.appDevSdk });


sdk.blueToothAdapter.addAdapter(BleComboDualModeDeviceAdapter4H5, true);
```

2. 搜索设备

   searchDevice 的参数详见 蓝牙适配器 搜索单个蓝牙设备的参数说明部分。

```
await blueToothAdapter.init();
console.log('开始搜索设备', sdk.deviceName);
const deviceInfo = await blueToothAdapter.searchDevice({
    deviceName: sdk.deviceName,
    serviceId: BleComboDualModeDeviceAdapter4H5.serviceId,
    productId: sdk.productId,
    disableCache: true,
});
```

3. 连接设备

   使用上一步获取到的 deviceInfo, 我们可以调用 connectDevice 连接设备以获得 deviceAdapter，传入参数定义详见 连接蓝牙设备 部分。

```
const deviceAdapter = await blueToothAdapter.connectDevice({
    ...device,
    productId: sdk.productId,
});
console.log('deviceAdapter:', deviceAdapter);
// authorized之后，才能向设备发送控制数据
if (!deviceAdapter.authorized) {
    await deviceAdapter.authenticateConnection({
        deviceName: sdk.deviceName,
    });
}
```

   当小程序和设备间的蓝牙连接成功后，面板就可以对设备进行控制了，sdk会将控制数据通过蓝牙发送到设备。例如进行开关的控制：

```
sdk.controlDeviceData({ power_switch: 1 });
```

ASR 语音识别

最近更新时间：2024-12-26 18:07:33

语音识别

目前支持两种场景，分别为“录音文件”与“一句话识别”，两种场景下入参会有所不同。
由于识别过程是异步，因此接口“voiceRecognition”并不会立即返回识别结果，可以理解为该接口是创建了一个“异步任务”，当这个成功被创建的“异步任务”执行完后，会通过 websocket 将结果推送到所有订阅该 deviceId 的终端上，下文为您详细介绍 ASR 语音识别。

- **接口定义**

```
sdk.voiceRecognition({...}) => Promise<{...}>
```

- **参数说明**
  - 关于“录音文件”场景支持的音频类型、大小限制以及相关字段的详细介绍，详情请参见 录音文件识别 。
  - 关于“一句话识别”场景支持的音频类型、大小限制以及相关字段的详细介绍，详情请参见 一句话识别 。
- **公共参数**

参数名	类型	必填	参数描述
DeviceId	string	否	默认使用当前设备的设备 ID。
AudioType	string	是	识别场景。 • “录音文件”取值“file”。 • “一句话识别”取值“sentence”。
Data	• Blob • File	是	音频文件。
ResourceName	string	否	文件名称，如果 Data 类型是 File，则取其“name”作为默认值。
EngineType	string	否	引擎模型类型，默认值为“16k_zh”。
FilterDirty	number	否	是否过滤脏词。
FilterModal	number	否	是否过滤语气词。
FilterPunc	number	否	是否过滤标点符号。
ConvertNumMode	number	否	是否进行阿拉伯数字智能转换。

- **录音文件额外参数**

参数名	类型	必填	参数描述
ChannelNum	number	否	语音声道数，默认为1。

SpeakerDiarization	number	否	是否开启话者分离。
SpeakerNumber	number	否	话者分离人数。

- **返回值**

参数名	类型	参数描述
ResourceToken	string	某个设备下，音频文件的唯一标示。

监听识别结果

对于“录音文件”场景，如果音频文件过大，后台可能会对音频文件进行分片识别，每个分片识别完成后，都将推送一条 websocket 消息，但推送的消息不保证顺序（例如有可能分片2的结果先到达）。
对于“asrResponse”事件，实际是基于“wsControl”事件进行二次封装；当然，您也可以通过监听“wsControl”事件获取识别结果。

- **接口定义**

```
sdk.on('asrResponse', ({ deviceId, data }) => void)
```

- **返回值说明**

参数名	类型	参数描述
deviceId	string	设备 ID。
data	object	识别结果数据。
data.resource_token	string	某个设备下，音频文件的唯一标示。
data.result_code	number	状态码，0代表成功。
data.total_num	number	分片总数。
data.seq	number	当前分片序号。
data.res_text	string	当前分片识别结果，对于“录音文件”场景，识别结果会包含分段时间戳。

获取语音文件下载链接

> **注意：**
> 仅限“录音文件”场景。

- **接口定义**

```
sdk.getAsrDownloadUrl({...}) => Promise<{...}>
```

- **参数说明**

参数名	类型	必填	参数描述
DeviceId	string	是	设备 ID。
ResourceToken	string	是	调用 voiceRecognition 返回的 ResourceToken。

- **返回值说明**

参数名	类型	参数描述
ResourceURL	string	cos 访问链接。

TRTC 音视频

最近更新时间：2024-12-26 18:07:33

接口描述

腾讯云物联网开发平台实时音视频服务，调用该接口从 h5 跳转至腾讯连连小程序 TRTC 页面。

接口定义

```
sdk.TRTCManager.goTRTCPage()
```

参数说明

参数名	类型	必填	参数描述
callType	string	是	呼叫类型： • `video` • `audio`

云存服务

最近更新时间：2024-05-31 15:02:41

云存服务

在 Explorer 平台增加 video 的云存能力，可以将事件云存与物模型中事件关联，方便用户在进行日志查询时查询到对应的云存图片等。

接口请求方式

通过应用端 API sdk.requestTokenApi(action, data, options) => Promise 对接口进行请求调用。

使用说明请参见 应用端 API 。

拉取云存事件列表

- **接口定义**

```
sdk.requestTokenApi("IotVideoDescribeCloudStorageEvents", {
    ProductId: string,
    DeviceName: string,
    StartTime?: number,
    EndTime?: number,
    Context?: string,
    Size?: number,
    EventId?: string
}) => Promise;
```

- **参数说明**

  请参见 拉取云存事件列表 。

拉取云存事件缩略图

- **接口定义**

```
sdk.requestTokenApi("IotVideoDescribeCloudStorageThumbnail", {
    ProductId: string,
    DeviceName: string,
    Thumbnail: string
}) => Promise;
```

- **参数说明**

  请参见 拉取云存事件缩略图 。

获取具有云存的日期

- **接口定义**

```
sdk.requestTokenApi("IotVideoDescribeCloudStorageDate", {
    ProductId: string,
```

```
    DeviceName: string
}) => Promise;
```

- **参数说明**

  请参见 获取具有云存的日期 。

获取某一天云存时间轴

- **接口定义**

```
sdk.requestTokenApi("IotVideoDescribeCloudStorageTime", {
    ProductId: string,
    DeviceName: string,
    Date: string,
    StartTime?: number,
    EndTime?: number
}) => Promise;
```

- **参数说明**

  请参见 获取某一天云存时间轴 。

获取视频防盗链播放 URL

- **接口定义**

```
sdk.requestTokenApi("IotVideoGenerateSignedVideoURL", {
    ProductId: string,
    VideoURL: string,
    ExpireTime: number
}) => Promise;
```

- **参数说明**

  先拼 VideoURL，使用上一接口的返回，加入开始时间和结束时间：

  ```
  https://zylcb.iotvideo.tencentcs.com/timeshift/live/xxxx/timeshift.m3u8?starttime_epoch={StartTime}&endtime_epoch={EndTime}
  ```

  例如：

  ```
  https://zylcb.iotvideo.tencentcs.com/timeshift/live/xxxx/timeshift.m3u8?starttime_epoch=1631849895&endtime_epoch=1631849907
  ```

  其它参数请参见 获取视频防盗链播放 URL 。

拉取图片流数据

- **接口定义**

```
sdk.requestTokenApi("IotVideoDescribeCloudStorageStreamData", {
    DeviceName: string,
    ProductId: string,
```

```
    StartTime: string
}) => Promise;
```

- **参数说明**

参数名	描述	类型	必填
DeviceName	设备名称。	String	是
ProductId	产品 ID。	String	是
StartTime	起始时间。	String	是

- **返回值**

参数名	描述	类型
RequestId	发送请求的 ID。	String
VideoStream	图片流视频地址。	String
AudioStream	图片流音频地址。	String

音乐服务

最近更新时间：2024-12-02 10:58:11

腾讯连连自定义 H5 SDK 提供 QQ 音乐接口能力，包括获取 QQ 音乐歌曲基本信息、播放链接、歌手信息、歌单信息、个性化推荐歌曲、最近播放、最新歌曲等，可用于支持 H5 面板内进行音乐内容的点播。

接口调用方式

```
window.h5PanelSdk.qqMusic.<接口名>
```

例如：

```
window.h5PanelSdk.qqMusic.goAuthPage()
```

> **注意：**
> 除“登录授权”以及“免登录接口”以外，其他接口均需要完成登录授权后才能调用。您可以调用 跳转 QQ 音乐登录授权页面 完成登录授权。
> 若在未登录状态下调用需登录的接口，会报错 `InvalidParameterValue.OAuthClientNotExist` 。

登录授权

登录授权完成后，H5 SDK 及设备可以使用所登录 QQ 音乐账号的听歌数据与会员权益。

查询 QQ 音乐授权状态

查询当前用户是否已完成 QQ 音乐登录授权。

```
getAuthStatus(): Promise<boolean>
```

输入参数

无

输出参数

```
boolean
```

跳转 QQ 音乐登录授权页面

调用该接口后会跳转到腾讯连连小程序完成 QQ 音乐登录授权。

```
goAuthPage(): void
```

输入参数

无

输出参数

无

歌曲

批量获取歌曲详情信息

通过歌曲 id 或歌曲 mid 获取一首或多首歌曲的详情信息。`SongIds` 和 `SongMids` 同时传入时优先使用 `SongMids` 。

```
describeSongInfoBatch(params: object): Promise<QQMusicSongInfo[]>
```

输入参数

参数名称	参数类型	参数描述
params.SongIds	number[]	歌曲 id 数组
params.SongMids	string[]	歌曲 mid 数组

输出参数

```
Promise<QQMusicSongInfo[]>
```

获取歌曲歌词

通过歌曲 id 或歌曲 mid 获取一首歌曲的歌词。`SongId` 和 `SongMid` 同时传入时优先使用 `SongMid` 。

```
describeLyric(params: object): Promise<DescribeLyricResp>
```

输入参数

参数名称	参数类型	参数描述
params.SongId	number	歌曲 id
params.SongMid	string	歌曲 mid

输出参数

```
Promise<DescribeLyricResp>
```

获取最近运营新歌

```
describeNewTrack(params: object): Promise<QQMusicNewTrackSongItem>
```

输入参数

参数名称	参数类型	参数描述
params.Tag	number	类别（12：内地；9：韩国；13：港台；3：欧美；8：日本；1：最新）

输出参数

```
Promise<QQMusicNewTrackSongItem>
```

设备控制

构造 IoT 设备播放列表

将 QQ 音乐歌曲信息转换为下发到 IoT 设备的播放列表数据格式。

```
pickSongInfoForSync(list, options): QQMusicSyncSongEntry[]
```

输入参数

参数名称	参数类型	参数描述
list.SongId	number	歌曲ID
list.SongName	string	歌曲名称
list.SingerName	string	歌手名称
list.SongPlayUrlDolby	string	杜比品质播放链接
list.SongPlayUrlXq	string	Hi-Res 品质播放链接
list.SongPlayUrlSq	string	超品质播放链接
list.SongPlayUrlHq	string	高品质播放链接
list.SongPlayUrlStandard	string	标准品质播放链接
list.SongPlayUrl	string	流畅品质播放链接
list.Try30SUrl	string	试听播放链接
list.TryBegin	number	试听开始时间（毫秒）
list.TryEnd	number	试听结束时间（毫秒）
list.UnplayableCode	number	不可播放状态码
list.UnplayableMsg	string	不可播放信提示信息

list.SongPlayTime	number	歌曲时长（秒）
options.quality	QQMusicQualityType	首选音质
options.necessarySongId	number	可选参数，若 id 对应的歌曲不能播放（UnplayableCode != 0），则抛出异常

输出参数

```
QQMusicSyncSongEntry[]
```

下发播放列表到 IoT 设备

```
syncSongList(params: object): Promise<void>
```

输入参数

参数名称	参数类型	参数描述
params.Id	string	播放列表 ID
params.Name	string	播放列表标题
params.Type	QQMusicPlayListType	播放列表类型
params.Quality	QQMusicQualityType	当前选择的音质
params.Total	number	当前播放列表的歌曲总数
params.Page	number	当前播放列表页码
params.PageSize	number	页长
params.SongList	QQMusicSyncSongEntry[]	播放列表（可以调用 pickSongInfoForSync 生成）
params.PlaySongId	number	开始播放的歌曲 ID

输出参数

```
Promise<void>
```

歌单

获取个人歌单目录

```
describeSelfSongList(): Promise<QQMusicSelfSongList[]>
```

输入参数

无

输出参数

```
Promise<QQMusicSelfSongList[]>
```

获取歌单中歌曲列表

```
describeSongList(params: object): Promise<DescribeSongListResp>
```

输入参数

参数名称	参数类型	参数描述
params.DissId	number	歌单 ID
params.Page	number	页码（从 0 开始）
params.PageSize	number	页长（最大为 30）

输出参数

```
Promise<DescribeSongListResp>
```

获取、收藏、取消收藏歌单

对歌单进行收藏、取消收藏操作。

```
operateSquareSongList(params: object): Promise<OperateSquareSongListResp>
```

输入参数

参数名称	参数类型	参数描述
params.DissId	number	操作的歌单 id
params.Op	number	进行的操作类型（1：收藏，2：取消收藏，3：获取收藏歌单）

输出参数

```
Promise<OperateSquareSongListResp>
```

MV

获取最新或最热 MV

```
describeMVList(params: object): Promise<QQMusicMVItem[]>
```

输入参数

参数名称	参数类型	参数描述
params.MVType	number	MV 类型（0：获取最新 MV；1：获取最热 MV）
params.MVTag	number	MV 标签（0：推荐；12：内地；9：韩国；13：港台；3：欧美；8：日本）

输出参数

```
Promise<QQMusicMVItem[]>
```

批量获取 MV 详情

通过 MV id 或 MV vid 获取一个或多个 MV 的详情信息。支持 `MVIds`，`MVVids` 两种 id 参数进行查询；同时传入时优先使用 `MVVids`。

```
describeMV(params: object): Promise<QQMusicMVInfo[]>
```

输入参数

参数名称	参数类型	参数描述
params.MVIds	number[]	歌曲 id 数组
params.MVVids	string[]	歌曲 mid 数组

输出参数

```
Promise<QQMusicMVInfo[]>
```

搜索

搜索歌曲、专辑、MV

根据输入的关键词搜索歌曲、专辑、MV或电台。

```
searchMusic(params: object): Promise<SearchMusicResp>
```

输入参数

参数名称	参数类型	参数描述
params.KeyWord	string	搜索关键字

params.SearchType	number	搜索类型（0：单曲 8：专辑 12：MV 15：电台）
params.Page	number	页码（从 1 开始，最大 4 页）
params.Num	number	页长（最大 50）

输出参数

```
Promise<SearchMusicResp>
```

歌手

搜索歌手列表

根据输入的关键词搜索歌手。

```
searchSingerList(params: object): Promise<QQMusicSearchSingerItem>
```

输入参数

参数名称	参数类型	参数描述
params.Keyword	string	搜索关键字
params.Page	number	页码（从 1 开始，最大 2 页）
params.PageSize	number	页长（最大 20）

输出参数

```
Promise<QQMusicSearchSingerItem>
```

获取热门歌手列表

按照分类索引（支持地区，性别，流派），拉取相应分类下的热门歌手id列表。

```
describeSingerList(params: object): Promise<QQMusicPopularSingerItem[]>
```

输入参数

参数名称	参数类型	参数描述
params.Area	number	歌手所在地区索引（-100：全部 200：内地 2：港台 3：韩国 4：日本 5：欧美）
params.Type	number	歌手性别索引（-100：全部 0：男 1：女 2：组合）

params.Genre	number	歌手所属流派索引（-100：全部 1：流行 2：摇滚 3：民谣 4：电子 5：爵士 6：嘻哈 8：R&B 9：轻音乐 10：民歌 14：古典 19：国风 20：蓝调 25：乡村）

输出参数

```
Promise<QQMusicPopularSingerItem[]>
```

获取歌手歌曲信息

通过歌手 id，获取歌手下的歌曲信息。

```
describeSinger(params: object): Promise<QQMusicSingerInfo>
```

输入参数

参数名称	参数类型	参数描述
params.SingerId	number	歌手 id
params.Page	number	页码（从 0 开始）
params.PageSize	number	页长（最大 50）
params.Order	number	歌曲排序方式（0：按时间，1：按热度）
params.Wiki	number	是否需要歌手wiki信息（0：不需要，1：需要）

输出参数

```
Promise<QQMusicSingerInfo>
```

最近播放

获取最近播放列表

获取用户最近播放列表，最大支持最近 100 条记录（全部 tab 下不包括歌曲历史播放的明细，需要通过歌曲 tab 查看明细）。

```
describeRecentPlay(params: object): Promise<DescribeRecentPlayResp>
```

输入参数

参数名称	参数类型	参数描述
params.Type	number	tab 类型（1：全部，2：歌曲，3：专辑，4：歌单）
params.UpdateT	numb	最近更新时间，由接口下发（见返回数据），客户端传入以增量获取数据，减少返回

ime	er	数据量，初始可传 0

输出参数

```
Promise<DescribeRecentPlayResp>
```

上报最近播放

上报用户的最近播放记录。

```
reportRecentPlay(params: object): Promise<void>
```

输入参数

参数名称	参数类型	参数描述
params.ResourceId	number	资源id
params.Type	number	资源类型（2：歌曲，3：专辑，4：歌单）

输出参数

```
Promise<void>
```

推荐

获取首页推荐

根据输入要获取的推荐内容类型获取内容。

```
describeHomepageSongList(params: object): Promise<DescribeHomepageSongListResp>
```

输入参数

参数名称	参数类型	参数描述
params.SN	string	机器码/序列号/唯一标识
params.Type	string	要获取的内容类型（200：单曲；500：歌单；可多选，如：200,500）

输出参数

```
Promise<DescribeHomepageSongListResp>
```

获取个性化推荐歌曲

获取个性化算法所推荐的歌曲，即个性电台。

```
describeIndividualRadio(): Promise<QQMusicIndividualRadioSongItem[]>
```

输入参数

无

输出参数

```
Promise<QQMusicIndividualRadioSongItem[]>
```

获取每日30首推荐歌曲

```
describeDailySongs(): Promise<QQMusicDailySongItem[]>
```

输入参数

无

输出参数

```
Promise<QQMusicDailySongItem[]>
```

榜单

获取排行榜榜单

获取 QQ 音乐的排行榜的歌单。

```
describeTopList(): Promise<QQMusicTopListGroupItem[]>
```

输入参数

无

输出参数

```
Promise<QQMusicTopListGroupItem[]>
```

获取排行榜榜单详情

获取 QQ 音乐的排行榜的歌曲。

```
describeTopListInfo(params: object): Promise<DescribeTopListInfoResp>
```

输入参数

参数名称	参数类型	参数描述
params.TopId	number	排行榜榜单id
params.Page	number	页码（从 0 开始）
params.PageSize	number	页长（最大 50）

输出参数

```
Promise<DescribeTopListInfoResp>
```

免登录专区

未登录用户获取播放列表

可以在用户未登录时候获取可播放歌曲列表，该接口不下发播放链接。

```
describeFreeSongList(): Promise<QQMusicFreeSongGroupItem>
```

输入参数

无

输出参数

```
Promise<QQMusicFreeSongGroupItem>
```

未登录用户获取播放链接

可以在用户未登录时候获取可播放歌曲的播放链接。

```
describeFreeSongInfo(params: object): Promise<QQMusicFreeSongInfo>
```

输入参数

参数名称	参数类型	参数描述
params.SongId	number	歌曲 id
params.SongToken	string	歌曲 token（可调用 describeFreeSongList 获取）

输出参数

```
Promise<QQMusicFreeSongInfo>
```

数据结构

QQMusicSongInfo

```
interface QQMusicSongInfo {
  SongId: number;
  SongMid: string;
  SongName: string;
  SongTitle: string;
  IsOnly: number;
  Vip: number;
  Language: string;
  Genre: string;
  PublicTime: string;
  SongPlayTime: number;
  IsDigitalAlbum: number;
  Copyright: number;
  Hot: number;
  Playable: number;
  SongH5Url: string;
  MvId: number;
  MvVid: string;
  KSongId: number;
  KSongMid: string;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerTitle: string;
  SingerPic: string;
  SingerPic150X150: string;
  SingerPic300X300: string;
  SingerPic500X500: string;
  OtherSingerList: {
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    SingerTitle: string;
    SingerPic: string;
  }[];
  AlbumId: number;
  AlbumMid: string;
  AlbumName: string;
  AlbumTitle: string;
  AlbumPic: string;
  AlbumPic150X150: string;
  AlbumPic300X300: string;
  AlbumPic500X500: string;
  SongPlayUrl: string;
  SongPlayUrlStandard: string;
  SongPlayUrlHq: string;
  SongPlayUrlSq: string;
```

```
  SongSize: number;
  SongSizeStandard: number;
  SongSizeHq: number;
  SongSizeSq: number;
  TryPlayable: number;
  TryBegin: number;
  TryEnd: number;
  Try30SUrl: string;
  TryFileSize: number;
  UnplayableCode: number;
  UnplayableMsg: string;
  UserOwnRule: number;
  SongVersion: number;
  WeightPlayCnt: number;
  FNote: number;
  EditAllow: number;
  SongPlayUrlDolby: string;
  SongSizeDolby: number;
  Action: {
    Switch: number;
    Switch2: number;
    PlayAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    DownloadAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    CacheAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
```

```
      ATMOS: number;
    };
  };
  SongEkeyStandard: string;
  SongEkeyHq: string;
  SongEkeySq: string;
  SongEkeyDolby: string;
  Extra: Record<string, unknown>;
  SongEkeyXq: string;
  SongPlayUrlXq: string;
  SongSizeXq: number;
  SongTypeDolby: string;
  Bpm: number;
  LimitFree: number;
  ShouldPay: number;
  PayPrice: number;
  VolumeGain: number;
  VolumePeak: number;
  VolumeLra: number;
  PayStatus: number;
  SongPlayUrlPq: string;
  SongEkeyPq: string;
  SongSizePq: number;
  SongTypePq: string;
}
```

DescribeLyricResp

```
interface DescribeLyricResp {
  SongId: number;
  SongLyric: string;
  SongTranslateEncLyric: string;
  SongRomeEncLyric: string;
  SongQrcEncLyric: string;
}
```

QQMusicNewTrackSongItem

```
interface QQMusicNewTrackSongItem {
  SongId: number;
  SongMid: string;
  SongName: string;
  SongTitle: string;
  IsOnly: number;
  Vip: number;
  Language: string;
  Genre: string;
  PublicTime: string;
```

```
  SongPlayTime: number;
  IsDigitalAlbum: number;
  Copyright: number;
  Hot: number;
  Playable: number;
  SongH5Url: string;
  MvId: number;
  MvVid: string;
  KSongId: number;
  KSongMid: string;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerTitle: string;
  SingerPic: string;
  SingerPic150X150: string;
  SingerPic300X300: string;
  SingerPic500X500: string;
  OtherSingerList: {
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    SingerTitle: string;
    SingerPic: string;
  }[];
  AlbumId: number;
  AlbumMid: string;
  AlbumName: string;
  AlbumTitle: string;
  AlbumPic: string;
  AlbumPic150X150: string;
  AlbumPic300X300: string;
  AlbumPic500X500: string;
  SongPlayUrl: string;
  SongPlayUrlStandard: string;
  SongPlayUrlHq: string;
  SongPlayUrlSq: string;
  SongSize: number;
  SongSizeStandard: number;
  SongSizeHq: number;
  SongSizeSq: number;
  TryPlayable: number;
  TryBegin: number;
  TryEnd: number;
  Try30SUrl: string;
  TryFileSize: number;
  UnplayableCode: number;
  UnplayableMsg: string;
  UserOwnRule: number;
```

```
  SongVersion: number;
  WeightPlayCnt: number;
  FNote: number;
  EditAllow: number;
  SongPlayUrlDolby: string;
  SongSizeDolby: number;
  Action: {
    Switch: number;
    Switch2: number;
    PlayAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    DownloadAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    CacheAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
  };
  SongEkeyStandard: string;
  SongEkeyHq: string;
  SongEkeySq: string;
  SongEkeyDolby: string;
  Extra: Record<string, unknown>;
  SongEkeyXq: string;
  SongPlayUrlXq: string;
  SongSizeXq: number;
  SongTypeDolby: string;
```

```
  Bpm: number;
  LimitFree: number;
  ShouldPay: number;
  PayPrice: number;
  VolumeGain: number;
  VolumePeak: number;
  VolumeLra: number;
  PayStatus: number;
  SongPlayUrlPq: string;
  SongEkeyPq: string;
  SongSizePq: number;
  SongTypePq: string;
}
```

QQMusicSyncSongEntry

```
interface QQMusicSyncSongEntry {
  UserOwnRule: number;
  SongId: number;
  SongPlayTime: number;
  SongURL: string;
  SongName: string;
  SingerName: string;
}
```

QQMusicSelfSongList

```
interface QQMusicSelfSongList {
  /** 歌单修改时间 */
  UpdateTime: number;
  /** 歌单创建时间 */
  CreateTime: number;
  /** 歌单ID */
  DissId: number;
  /** 歌单名 */
  DissName: string;
  /** 歌单封面 */
  DissPic: string;
  /** 歌单歌曲数量 */
  SongNum: number;
  /** 歌单收听数量 */
  ListenNum: number;
}
```

DescribeSongListResp

```
interface DescribeSongListResp {
```

```
  DissId: number;
  Hot: number;
  DissTitle: string;
  PicUrl: string;
  TotalNum: number;
  OwnerFlag: number;
  SongList: QQMusicSongInDiss[];
}
```

QQMusicSongInDiss

```
interface QQMusicSongInDiss {
  SongId: number;
  SongName: string;
  SongTitle: string;
  SongMid: string;
  AlbumId: number;
  AlbumMid: string;
  AlbumName: string;
  SongPlayTime: number;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  QQMusicFlag: number;
  UserOwnRule: number;
  OpiPlayFlag: number;
  SongType: number;
}
```

OperateSquareSongListResp

```
interface OperateSquareSongListResp {
  Data: QQMusicSelfSongList[];
}
```

QQMusicMVItem

```
interface QQMusicMVItem {
  Id: number;
  Vid: string;
  Title: string;
  Subtitle: string;
  PicURL: string;
  Duration: number;
  PlayCnt: number;
  CommentCnt: number;
```

```
}
```

QQMusicMVInfo

```
interface QQMusicMVInfo {
  /** 蓝光流媒体大小 */
  MvBlueRaySize: number;
  /** 蓝光流媒体url */
  MvBlueRayURL: string;
  /** 高清流媒体大小 */
  MvHQSize: number;
  /** 高清流媒体url */
  MvHQUrl: string;
  /** 低品质流媒体大小 */
  MvLQSize: number;
  /** 低品质流媒体url */
  MvLQUrl: string;
  /** 超清流媒体大小 */
  MvSQSize: number;
  /** 超清流媒体url */
  MvSQUrl: string;
  /** Mv的文件id */
  MVFileid: string;
  /** Mv的id */
  MvId: number;
  /** 播放时长 */
  MvPlayTime: number;
  /** Mv的标题 */
  MvTitle: string;
  /** Mv的vid */
  MvVid: string;
  /** Mv图片url */
  PicUrl: string;
  /** 播放权限 */
  Playable: number;
  /** 发布时间 */
  PublicTime: string;
  /** 歌手id */
  SingerId: number;
  /** 歌手mid */
  SingerMid: string;
  /** 歌手名 */
  SingerName: string;
  /** Mv的歌手集合 */
  Singers: {
    /** 歌手id */
    Id: number;
    /** 歌手mid */
```

```
    Mid: string;
    /** 歌手名 */
    Name: string;
  }[];
  /** 不能播放权限阻断码 */
  UnplayableCode: number;
}
```

SearchMusicResp

```
interface SearchMusicResp {
  /** 当前返回个数 */
  CurNum: number;
  /** 当前页码 */
  CurPage: number;
  /** 该搜索词可以搜到的总结果数 */
  TotalNum: number;
  /** 搜索词 */
  Keyword: string;
  /** 单曲列表 */
  List: QQMusicSearchMusicItem[];
  /**  */
  DirectInfo: QQMusicDirectInfo;
}
```

QQMusicSearchSingerItem

```
interface QQMusicSearchSingerItem {
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerPic: string;
  AlbumNum: string;
  SongNum: string;
}
```

QQMusicPopularSingerItem

```
interface QQMusicPopularSingerItem {
  Country: string;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerTranslatorName: string;
}
```

QQMusicSingerInfo

```
interface QQMusicSingerInfo {
  Area: string;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerPic: string;
  SingerTranslatorName: string;
  SongSum: number;
  SongList: {
    UserOwnRule: number;
    AlbumId: number;
    AlbumMid: string;
    AlbumName: string;
    AlbumPic: string;
    Genre: string;
    IsOnly: number;
    KSongId: number;
    KSongMid: string;
    Language: string;
    Playable: number;
    PublicTime: string;
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    SingerPic: string;
    SizeTry: number;
    SongH5Url: string;
    SongId: number;
    SongMid: string;
    SongName: string;
    SongPlayTime: number;
    SongPlayUrl: string;
    SongPlayUrlHq: string;
    SongPlayUrlSq: string;
    SongPlayUrlStandard: string;
    SongSize: number;
    SongSizeHq: number;
    SongSizeSq: number;
    SongSizeStandard: number;
    TryBegin: number;
    TryEnd: number;
    IsDigitalAlbum: number;
  }[];
}
```

DescribeRecentPlayResp

```
interface DescribeRecentPlayResp {
  Data: {
    All: Array<{
      Type: number;
      TypeName: string;
      LastTime: number;
      Detail: {
        Comm: {
          Title: string;
          Pic: string;
          Count: number;
        };
        Album: {
          Title: string;
          Mid: string;
          Singer: string;
          Count: number;
          Pic: string;
          Pic150X150: string;
          Pic300X300: string;
          Pic500X500: string;
          Id: number;
          LastTime: number;
        };
        Playlist: {
          Title: string;
          Id: number;
          Creator: string;
          Count: number;
          Pic: string;
          LastTime: number;
        };
      };
    }>;
    Song: Array<{
      Title: string;
      Mid: string;
      Singer: string[];
      AlbumTitle: string;
      IsOnly: number;
      Size320Mp3: number;
      SizeFlac: number;
      ListenCount: number;
      SubTitle: string;
      Id: number;
      Vip: number;
      LastTime: number;
    }>;
```

```
    Album: Array<{
      Title: string;
      Mid: string;
      Singer: string;
      Count: number;
      Pic: string;
      Pic150X150: string;
      Pic300X300: string;
      Pic500X500: string;
      Id: number;
      LastTime: number;
    }>;
    Playlist: Array<{
      Title: string;
      Id: number;
      Creator: string;
      Count: number;
      Pic: string;
      LastTime: number;
    }>;
  };
  UpdateTime: number;
}
```

DescribeHomepageSongListResp

```
interface DescribeHomepageSongListResp {
  ShelfArr: {
    Title: string;
    CardArr: {
      Type: number;
      Id: string;
      Title: string;
      Subtitle: string;
      Cover: string;
      Cnt: number;
      SubId: string;
      FavCnt: number;
    }[];
  }[];
  RcItemArr: {
    DirId: number;
    DirName: string;
    DissId: number;
    AlbumPicUrl: string;
  }[];
}
```

QQMusicIndividualRadioSongItem

```
interface QQMusicIndividualRadioSongItem {
  SongId: number;
  SongMid: string;
  SongName: string;
  SongTitle: string;
  Isonly: number;
  Vip: number;
  Language: string;
  Genre: string;
  PublicTime: string;
  SongPlayTime: number;
  IsdigitalAlbum: number;
  Copyright: number;
  Hot: number;
  Playable: number;
  SongH5Url: string;
  MvId: number;
  MvVid: string;
  KSongId: number;
  KSongMid: string;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerTitle: string;
  SingerPic: string;
  SingerPic150X150: string;
  SingerPic300X300: string;
  SingerPic500X500: string;
  OtherSingerList: {
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    SingerTitle: string;
    SingerPic: string;
  }[];
  AlbumId: number;
  AlbumMid: string;
  AlbumName: string;
  AlbumTitle: string;
  AlbumPic: string;
  AlbumPic150X150: string;
  AlbumPic300X300: string;
  AlbumPic500X500: string;
  SongPlayUrl: string;
  SongPlayUrlStandard: string;
  SongPlayUrlHq: string;
  SongPlayUrlSq: string;
```

```
  SongSize: number;
  SongSizeStandard: number;
  SongSizeHq: number;
  SongSizeSq: number;
  TryPlayable: number;
  TryBegin: number;
  TryEnd: number;
  Try30SUrl: string;
  TryFileSize: number;
  UnplayableCode: number;
  UnplayableMsg: string;
  UserOwnRule: number;
  SongVersion: number;
  WeightPlayCnt: number;
  Fnote: number;
  EditAllow: number;
  SongPlayUrlDolby: string;
  SongSizeDolby: number;
  Action: {
    Switch: number;
    Switch2: number;
  };
  SongEkeyStandard: string;
  SongEkeyHq: string;
  SongEkeySq: string;
  SongEkeyDolby: string;
  Extra: Record<string, unknown>;
}
```

QQMusicDailySongItem

```
interface QQMusicDailySongItem {
  SongId: number;
  SongMid: string;
  SongName: string;
  SongTitle: string;
  Isonly: number;
  Vip: number;
  Language: string;
  Genre: string;
  PublicTime: string;
  SongPlayTime: number;
  IsdigitalAlbum: number;
  Copyright: number;
  Hot: number;
  Playable: number;
  SongH5Url: string;
  MvId: number;
```

```
  MvVid: string;
  KSongId: number;
  KSongMid: string;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerTitle: string;
  SingerPic: string;
  SingerPic150X150: string;
  SingerPic300X300: string;
  SingerPic500X500: string;
  OtherSingerList: {
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    SingerTitle: string;
    SingerPic: string;
  }[];
  AlbumId: number;
  AlbumMid: string;
  AlbumName: string;
  AlbumTitle: string;
  AlbumPic: string;
  AlbumPic150X150: string;
  AlbumPic300X300: string;
  AlbumPic500X500: string;
  SongPlayUrl: string;
  SongPlayUrlStandard: string;
  SongPlayUrlHq: string;
  SongPlayUrlSq: string;
  SongSize: number;
  SongSizeStandard: number;
  SongSizeHq: number;
  SongSizeSq: number;
  TryPlayable: number;
  TryBegin: number;
  TryEnd: number;
  Try30SUrl: string;
  TryFileSize: number;
  UnplayableCode: number;
  UnplayableMsg: string;
  UserOwnRule: number;
  SongVersion: number;
  WeightPlayCnt: number;
  Fnote: number;
  EditAllow: number;
  SongPlayUrlDolby: string;
  SongSizeDolby: number;
  Action: {
```

```
    Switch: number;
    Switch2: number;
    PlayAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    DownloadAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    CacheAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
  };
  SongEkeyStandard: string;
  SongEkeyHq: string;
  SongEkeySq: string;
  SongEkeyDolby: string;
  Extra: {
    Abt: string;
    Cmd: string;
    Tf: string;
    Trace: string;
    UniqueTraceId: string;
  };
  SongEkeyXq: string;
  SongPlayUrlXq: string;
  SongSizeXq: number;
  SongTypeDolby: string;
  Bpm: number;
```

```
  LimitFree: number;
  ShouldPay: number;
  PayPrice: number;
  VolumeGain: number;
  VolumePeak: number;
  VolumeLra: number;
  PayStatus: number;
  SongPlayUrlPq: string;
  SongEkeyPq: string;
  SongSizePq: string;
  SongTypePq: string;
}
```

QQMusicTopListGroupItem

```
interface QQMusicTopListGroupItem {
  GroupId: number;
  GroupName: string;
  GroupTopList: {
    ListenNum: number;
    ShowTime: string;
    SongList: {
      Rank: number;
      SingerId: number;
      SingerMid: string;
      SingerName: string;
      SongId: number;
      SongMid: string;
      SongName: string;
    }[];
    TopBannerPic: string;
    TopHeaderPic: string;
    TopId: number;
    TopName: string;
    TopType: number;
    TotalNum: number;
  }[];
  GroupType: number;
}
```

DescribeTopListInfoResp

```
interface DescribeTopListInfoResp {
  ListenNum: number;
  SongList: {
    UserOwnRule: number;
    AlbumId: number;
    AlbumMid: string;
```

```
    AlbumName: string;
    AlbumPic: string;
    Genre: string;
    Hot: number;
    IsOnly: number;
    KSongId: number;
    KSongMid: string;
    Language: string;
    Playable: number;
    PublicTime: string;
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    SingerPic: string;
    SongH5Url: string;
    SongId: number;
    SongMid: string;
    SongName: string;
    SongPlayTime: number;
    SongPlayUrl: string;
    SongPlayUrlHq: string;
    SongPlayUrlSq: string;
    SongPlayUrlStandard: string;
    SongSize: number;
    SongSizeHq: number;
    SongSizeSq: number;
    SongSizeStandard: number;
    TopRankIncrease: number;
    TryBegin: number;
    TryEnd: number;
    IsDigitalAlbum: number;
  }[];
  TopBannerPic: string;
  TopDesc: string;
  TopHeaderPic: string;
  TopId: number;
  TopName: string;
  TopType: number;
}
```

QQMusicFreeSongGroupItem

```
interface QQMusicFreeSongGroupItem {
  SongList: {
    SongId: number;
    SongMid: string;
    SongTitle: string;
    Vip: number;
```

```
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    TryPlayable: number;
    SongPlayTime: number;
    Playable: number;
    UnplayableCode: number;
    AlbumId: number;
    AlbumMid: string;
    AlbumTitle: string;
    AlbumPic: string;
    AlbumPic300X300: string;
    AlbumPic500X500: string;
    SongToken: string;
  }[];
  Number: number;
  Name: string;
  Pic: string;
  BannerPic: string;
}
```

QQMusicFreeSongInfo

```
interface QQMusicFreeSongInfo {
  SongId: number;
  SongMid: string;
  SongName: string;
  SongTitle: string;
  IsOnly: number;
  Vip: number;
  Language: string;
  Genre: string;
  PublicTime: string;
  SongPlayTime: number;
  IsDigitalAlbum: number;
  Copyright: number;
  Hot: number;
  Playable: number;
  SongH5Url: string;
  MvId: number;
  MvVid: string;
  KSongId: number;
  KSongMid: string;
  SingerId: number;
  SingerMid: string;
  SingerName: string;
  SingerTitle: string;
  SingerPic: string;
```

```
  SingerPic150X150: string;
  SingerPic300X300: string;
  SingerPic500X500: string;
  OtherSingerList: {
    SingerId: number;
    SingerMid: string;
    SingerName: string;
    SingerTitle: string;
    SingerPic: string;
  }[];
  AlbumId: number;
  AlbumMid: string;
  AlbumName: string;
  AlbumTitle: string;
  AlbumPic: string;
  AlbumPic150X150: string;
  AlbumPic300X300: string;
  AlbumPic500X500: string;
  SongPlayUrl: string;
  SongPlayUrlStandard: string;
  SongPlayUrlHq: string;
  SongPlayUrlSq: string;
  SongSize: number;
  SongSizeStandard: number;
  SongSizeHq: number;
  SongSizeSq: number;
  TryPlayable: number;
  TryBegin: number;
  TryEnd: number;
  Try30SUrl: string;
  TryFileSize: number;
  UnplayableCode: number;
  UnplayableMsg: string;
  UserOwnRule: number;
  SongVersion: number;
  WeightPlayCnt: number;
  FNote: number;
  EditAllow: number;
  SongPlayUrlDolby: string;
  SongSizeDolby: number;
  Action: {
    Switch: number;
    Switch2: number;
    PlayAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
```

```
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    DownloadAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
    CacheAccess: {
      FQ: number;
      STANDARD: number;
      HQ: number;
      SQ: number;
      XQ: number;
      DOLBY: number;
      FLYRA: number;
      ATMOS: number;
    };
  };
  SongEkeyStandard: string;
  SongEkeyHq: string;
  SongEkeySq: string;
  SongEkeyDolby: string;
  Extra: Record<string, unknown>;
  SongEkeyXq: string;
  SongPlayUrlXq: string;
  SongSizeXq: number;
  SongTypeDolby: string;
  Bpm: number;
}
```

QQMusicQualityType

```
const enum QQMusicQualityType {
  /** 流畅 */
  FQ = 1,
  /** 标准 */
  STANDARD = 2,
  /** 高品质 */
  HQ = 4,
  /** 超品质 */
  SQ = 8,
```

```
  /** Hi-Res */
  XQ = 16,
  /** 杜比 */
  DOLBY = 32,
  /** 母带 */
  FLYRA = 64,
  /** 臻品全景声 */
  ATMOS = 128,
}
```

QQMusicPlayListType

```
const enum QQMusicPlayListType {
  /** 歌单 */
  SongList = 5,
  /** 歌曲 */
  Song = 6,
  /** 电台 */
  Radio = 7,
  /** 榜单 */
  Rank = 8,
  /** 专辑 */
  Album = 9,
  /** 歌手 */
  Singer = 10,
  /** 推荐 */
  Recommend = 12,
  /** 免登录专区 */
  NoLoginArea = 14,
}
```

底层 SDK 能力

最近更新时间：2023-05-29 18:02:34

应用开发 SDK

H5 SDK 底层依赖应用开发小程序端 SDK。通过以下代码可以获取应用开发 SDK 的实例，更多调用能力请参见 应用开发小程序端 SDK 文档。

接口定义

```
sdk.appDevSdk
```

微信 JS SDK

通过以下代码可以获取微信 JSSDK 实例，具体用法请参见 小程序 web-view 文档 。使用前需要先调用 **初始化 JS SDK**。

接口定义

```
sdk.wx
```

初始化 JS SDK

- **接口定义**

  ```
  sdk.wxSdkReady() => Promise
  ```

- **返回值**

  返回一个带缓存的 Promise，初始化成功后 resolve。若初始化未完成或已初始化成功，则多次调用后返回同一个 Promise。若初始化失败，则该缓存的 Promise 在 reject 之后会被释放，再次调用则将重新初始化。

- **示例代码**

  ```
  sdk.wxSdkReady().then(() => wx.miniProgram.navigateBack());
  ```

自主品牌小程序和 App 开发

最近更新时间：2025-04-08 11:09:52

概述

本文档用于指引设备厂商开发自主品牌的小程序和 App。

接入指南

一站式开发

基于腾讯云物联网开发平台的**应用端 API**，开发自主品牌的小程序和 App，您无需开发自己的业务后台。

1. 参考 应用开发 指南，在物联网开发平台创建应用，获取 App Key 与 App Secret。
2. 设备上行数据，小程序和 App 可参考 长连接通信 订阅接收，或通过 数据查询 接口获取。
3. 小程序和 App 可通过 设备控制 接口，向设备下发控制指令。

> **说明：**
> 如需开发音视频通话功能，请先与商务沟通，购买音视频激活码。届时，会有专群支持您对接使用。

云云对接开发

基于腾讯云物联网开发平台的**云 API**，开发自主品牌的小程序和 App，您需开发自己的业务后台。

1. 小程序/App 获取设备上行数据
   1.1 设备上报的数据，通过 规则引擎 转发到客户业务后台。
   1.2 客户业务后台收到设备上报的数据，可进行保存，用于小程序/App 查询。
   1.3 小程序/App 通过客户业务后台提供的接口，请求到客户业务后台，获取设备的最新数据或历史数据。
2. 小程序/App 控制设备
   2.1 小程序/App 把设备的控制指令，通过客户业务后台提供的接口，请求到客户业务后台。
   2.2 客户业务后台通过 远程控制设备 的云 API，向设备下发控制指令。

> **说明：**
> 如需开发音视频通话功能，请先与商务沟通，购买音视频激活码。届时，会有专群支持您对接使用。