

# 腾讯云微信小程序开发者工具

SDK 文档

产品文档



腾讯云

## 【版权声明】

©2013-2017 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

## 【商标声明】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

## 【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

## 文档目录

文档声明.....	2
SDK 文档.....	4
小程序端 SDK 文档 .....	4
Node.js SDK 文档 .....	12
PHP SDK 文档 .....	32

SDK 文档

## 小程序端 SDK 文档

本项目是 Wafer 的组成部分，为小程序客户端开发提供 SDK 支持会话服务和信道服务。

## SDK 安装

解决方案 [wafer2-quickstart](#) 已经集成并使用最新版的 SDK，需要快速了解的可以从 Demo 开始。

如果需要单独开始，本 SDK 已经发布为 npm 模块，可以直接安装到小程序目录中。

```
npm install wafer2-client-sdk
```

安装之后，就可以使用

```
require
```

引用 SDK 模块：

```
var qcloud = require('./node_modules/wafer2-client-sdk/index.js');
```

## 会话服务

会话服务让小程序拥有会话管理能力。

### 登录

登录可以在小程序和服务器之间建立会话，服务器由此可以获取到用户的标识和信息。

```
var qcloud = require('./node_modules/wafer2-client-sdk/index.js');
```

```
// 设置登录地址
```

```
qcloud.setLoginUrl('https://199447.qcloud.la/weapp/login');
qcloud.login({
  success: function (userInfo) {
    console.log('登录成功', userInfo);
  },
  fail: function (err) {
    console.log('登录失败', err);
  }
});
```

本 SDK 需要配合云端 SDK 才能提供完整会话服务。通过 [setLoginUrl](#) 设置登录地址，云服务器在该地址上使用云端 SDK 处理登录请求。

### setLoginUrl

方法设置登录地址之后会一直有效，因此你可以在微信小程序启动时设置。

登录成功后，可以获取到当前微信用户的基本信息。

### 请求

如果希望小程序的网络请求包含会话，登录之后使用 [request](#) 方法进行网络请求即可。

```
qcloud.request({
  url: 'http://199447.qcloud.la/user',
  success: function (response) {
    console.log(response);
  },
  fail: function (err) {
    console.log(err);
  }
});
```

```
}  
});
```

如果调用

request

之前还没有登录，则请求不会带有会话。

request

方法也支持

login

参数支持在请求之前自动登录。

```
// 使用 login 参数之前，需要设置登录地址  
qcloud.setLoginUrl('https://199447.qcloud.la/login');  
qcloud.request({  
  login: true,  
  url: 'http://199447.qcloud.la/user',  
  success: function (response) {  
    console.log(response);  
  },  
  fail: function (err) {  
    console.log(err);  
  }  
});
```

关于会话服务详细技术说明，请参考 [Wiki](#)。

## 信道服务

[信道服务](#)小程序支持利用腾讯云的信道资源使用 WebSocket 服务。

```
// 创建信道，需要给定后台服务地址
var tunnel = this.tunnel = new qcloud.Tunnel('https://199447.qcloud.la/tunnel');

// 监听信道内置消息，包括 connect/close/reconnecting/reconnect/error
tunnel.on('connect', () => console.log('WebSocket 信道已连接'));
tunnel.on('close', () => console.log('WebSocket 信道已断开'));
tunnel.on('reconnecting', () => console.log('WebSocket 信道正在重连...'));
tunnel.on('reconnect', () => console.log('WebSocket 信道重连成功'));
tunnel.on('error', error => console.error('信道发生错误：', error));

// 监听自定义消息（服务器进行推送）
tunnel.on('speak', speak => console.log('收到 speak 消息：', speak));

// 打开信道
tunnel.open();
// 发送消息
tunnel.emit('speak', { word: "hello", who: { nickName: "techird" }});
// 关闭信道
tunnel.close();
```

信道服务同样需要业务服务器配合云端 SDK 支持，构造信道实例的时候需要提供业务服务器提供的信道服务地址。通过监听信道消息以及自定义消息来通过信道实现业务。

关于信道使用的更完整实例，建议参考客户端 Demo 中的[三木聊天室应用源码](#)。

关于信道服务详细技术说明，请参考 [Wiki](#)。

## API

## setLoginUrl

设置会话服务登录地址。

语法

```
qcloud.setLoginUrl(loginUrl);
```

参数

参数	类型	说明
loginUrl	string	会话服务登录地址

## login

登录，建立微信小程序会话。

语法

```
qcloud.login(options);
```

参数

参数	类型	说明
options	PlainObject	会话服务登录地址
options.success	() => void	登录成功的回调
options.error	(error) => void	登录失败的回调

## request

进行带会话的请求。

语法



```
qcloud.request(options);
```

### 参数

参数	类型	说明
options	PlainObject	会话服务登录地址
options.login	bool	是否自动登录以获取会话，默认为 false
options.url	string	必填，要请求的地址
options.header	PlainObject	请求头设置，不允许设置 Referer
options.method	string	请求的方法，默认为 GET
options.success	(response) => void	登录成功的回调。  <ul style="list-style-type: none"> <li>response.statusCode : 请求返回的状态码</li> <li>response.data : 请求返回的数据</li> </ul>
options.error	(error) => void	登录失败的回调
options.complete	() => void	登录完成后回调，无论成功还是失败

## Tunnel

表示一个信道。由于小程序的限制，同一时间只能有一个打开的信道。

### constructor

#### 语法

```
var tunnel = new Tunnel(tunnelUrl);
```

### 参数

参数	类型	说明
tunnelUrl	String	会话服务登录地址

### on

监听信道上的事件。信道上事件包括系统事件和服务器推送消息。

### 语法

```
tunnel.on(type, listener);
```

### 参数

参数	类型	说明
type	string	监听的事件类型
listener	(message?: any) => void	监听器，具体类型的事件发生时调用监听器。如果是消息，则会有消息内容。

### 事件

事件	说明
connect	信道连接成功后回调
close	信道关闭后回调
reconnecting	信道发生重连时回调
reconnected	信道重连成功后回调
error	信道发生错误后回调
[message]	信道服务器推送过来的消息类型，如果消息类型和上面内置的时间类型冲突，需要在监听的时候在消息类型前加 @
*	监听所有事件和消息，监听器第一个参数接收到时间或消息类型

## open

打开信道，建立连接。由于小程序的限制，同一时间只能有一个打开的信道。

### 语法

```
tunnel.open();
```

## emit

向信道推送消息。

### 语法

```
tunnel.emit(type, content);
```

### 参数

参数	类型	说明
type	string	要推送的消息的类型
content	any	要推送的消息的内容

## close

关闭信道

### 语法

```
tunnel.close();
```

## Node.js SDK 文档

### 介绍

Wafer 服务端 SDK 是腾讯云为微信小程序开发者提供的快速开发库，SDK 封装了以下功能供小程序开发者快速调用：

- 用户登录与验证
- 信道服务
- 图片上传
- 数据库
- 客服消息

开发者只需要根据文档对 SDK

进行初始化配置，就可以获得以上能力。你还可以直

接到[腾讯云小程序控制台](#)

购买小程序解决方案，可以得到运行本示例所需的资源和服务，其中包括已部署好的相关程序、示例代码及自动下发的 SDK 配置文件。

### 安装

```
npm install wafer-node-sdk --save
```

### SDK 初始化配置

```
require('qcloud-weapp-server-sdk')(options)
```

该方法用于初始化 SDK 需要使用的各种配置项，需先于其他 API 调用。

参数

- appId

- 可选。微信小程序的 App id

- appSecret

- 可选。微信小程序的 App secret

- useQcloudLogin

- 必填。是否使用腾讯云代理登录小程序。会话登录需要使用小程序的 App id 和 App secret 来解密用户信息，腾讯云提供使用腾讯云 App id 和 App secret 代理请求微信进行解密。如果该项为

false

，则需填写微信小程序 App id 和 App secret。默认为

true

- mysql

- 选填。MySQL 配置。不填则使用小程序解决方案分配机器中默认的 MySQL，若使用自行部署的 MySQL 数据库，则需提供一个类型为

object

的配置，具体配置项如下：

- host

- 必填。MySQL 主机名

- user

- 必填。MySQL 用户名

- db

- 必填。MySQL 数据库名

- pass

- 必填。MySQL 密码，若使用了腾讯云微信小程序解决方案，开发环境下，MySQL 的初始密码为您的微信小程序 appid

- port

- 选填。MySQL 端口 ( 默认 : 3306 )

- char

- 选填。MySQL 编码

- COS

- 必填。腾讯云对象存储配置信息，用于上传模块使用。

- region

- 必填。COS 的地域

- fileBucket

- 必填。COS 的 bucket 名

- uploadFolder

- 必填。COS 上传文件夹名

- maxSize

- 选填。COS 上传最大大小，默认 5M (单位：M)

- field

- 选填。COS 上传是 field 名称，默认为

- file

- serverHost

- 必填。当前服务器的 hostname

- tunnelServerUrl

- 必填。信道服务器地址

- tunnelSignatureKey

- 必填。信道服务签名密钥

- qcloudAppId

- 必填。腾讯云 AppId

- qcloudSecretId

- 必填。腾讯云 SecretId

- qcloudSecretKey

- 必填。腾讯云 SecretKey

- wxMessageToken

- 必填。微信客服消息通知 token

- wxLoginExpires

- 可选。微信登录态有效期，默认 7200 秒（单位：秒）

如果购买了腾讯云小程序解决方案，配置项中

serverHost

,

tunnelServerUrl



,

tunnelSignatureKey

,

qcloudAppId

,

qcloudSecretId

,

qcloudSecretKey

,

wxMessageToken

由腾讯云自动下发到您的服务器上。

自动下发的 SDK 配置文件地址：

/data/release/sdk.config

返回值

初始化之后会返回一个 SDK 的实例，提供以下所有的 API。

## 用户登录和校验

qcloud.auth.authorization(req)

登录授权接口，该接口会返回登录状态和用户信息，并将用户信息和登录态储存到 MySQL 里

参数

- req

-

http.IncomingMessage

实例化对象

返回值

Promise

对象，

.then

中可以得到登录状态和用户信息

返回数据格式

```
{
  loginState: number, // 1表示登录成功，0表示登录失败
  userinfo: object
}
```

调用示例

```
// express
module.exports = (req, res) => {
  qcloud.auth.authorization(req).then(result => {
    // result : {
```

```
// loginState: 0 // 1表示登录成功，0表示登录失败
// userinfo: { // 用户信息 }
// }
})
}
```

qcloud.auth.validation(req)

登录态校验接口，该接口会从 MySQL 取出用户信息，校验登录态，返回登录状态和用户信息

参数

- req

-

http.IncomingMessage

实例化对象

返回值

Promise

对象，

.then

中可以得到登录状态和用户信息

返回数据格式

```
{
  loginState: number, // 1表示登录成功，0表示登录失败
```

```
userinfo: object  
}
```

调用示例

```
// express  
module.exports = (req, res) => {  
  qcloud.auth.validation(req).then(result => {  
    // result : {  
    // loginState: 0 // 1表示登录成功，0表示登录失败  
    // userinfo: { // 用户信息 }  
    // }  
  })  
}
```

```
qcloud.auth.authorizationMiddleware(ctx[, next])
```

用户登录的 Koa 中间件，登录信息将会被写进

```
ctx.state.$wxInfo
```

参数

- ctx

-

Koa Context

Koa 上下文

- next

### 调用示例

```
const { auth: { authorizationMiddleware } } = qcloud

// 颁发登录态
router.get('/login', authorizationMiddleware, ctx => {
  console.log(ctx.state.$wxInfo)
  // {
  //   loginState: 0 // 1表示登录成功, 0表示登录失败
  //   userinfo: { // 用户信息 }
  // }
})
```

```
qcloud.auth.validationMiddleware(ctx[, next])
```

用户登录态校验的 Koa 中间件，登录信息将会被写进

```
ctx.state.$wxInfo
```

### 参数

- ctx

-

Koa Context

Koa 上下文

- next

调用示例

```
const { auth: { validationMiddleware } } = qcloud
```

```
// 校验登录态
```

```
router.get('/user', validationMiddleware, ctx => {  
  console.log(ctx.state.$wxInfo)  
  // {  
  // loginState: 0 // 1表示登录成功，0表示登录失败  
  // userinfo: { // 用户信息 }  
  // }  
})
```

## 信道服务

```
qcloud.tunnel.getTunnelUrl(req)
```

获取信道地址接口。调用这个接口需要用户已经登录，请查看客服端示例

参数

- req

-

http.IncomingMessage

实例化对象

返回值

Promise

对象 ,

.then

中可以获得信道服务地址

返回数据格式

```
{
  tunnel: {
    tunnelId: string, // 信道ID
    connectUrl: string // 信道连接地址
  },
  userinfo: object // 用户信息
}
```

调用示例

```
// express
module.exports = (req, res) => {
  qcloud.tunnel.getTunnelUrl(req).then(result => {
    // {
    // tunnel: {
    // tunnelId: string, // 信道ID
    // connectUrl: string // 信道连接地址
    // },
    // userinfo: object // 用户信息
    // }
    // 你需要在本地维护一个信道 ID 和用户信息的对应关系
  })
}
```

qcloud.tunnel.onTunnelMessage(body)

解析请求体获取信道服务 post 过来的信息

参数

- req

-

http.IncomingMessage

实例化对象

返回值

Promise

对象 ,

.then

中可以获得信道服务地址

返回数据格式

```
{
  type: string, // 包类型 , 有 connect, message, close 三种
  tunnelId: string // 信道 ID
  content: {
    messageType: string, // 消息类型
    messageContent: string // 消息内容
  }
}
```



调用示例

```
// express
module.exports = (req, res) => {
  qcloud.tunnel.onTunnelMessage(req.body).then(packet => {
    // {
    // type: string, // 包类型 , 有 connect, message, close 三种
    // tunnelId: string // 信道 ID
    // content: {
    // messageType: string, // 消息类型
    // messageContent: string // 消息内容
    // }
    // }
  })
}

qcloud.tunnel.broadcast(tunnelIds, messageType, messageContent)
```

向指定的多个

tunnelId

广播信息

参数

- tunnelIds

- 要广播的信道 ID 的列表数组

- messageType

- 信息类型

- messageContent

- 信息内容

返回数据

Promise

对象

调用示例

```
const { tunnel: { broadcast } } = qcloud
```

```
tunnel.broadcast(['abcdefghijk'], 'speak', 'hello world')  
.then(result => {  
  const invalidTunnelIds = result.data && result.data.invalidTunnelIds || []  
  // { invalidTunnelIds: [] } // 会返回无效的信道ID  
})
```

```
qcloud.tunnel.closeTunnel(tunnelId)
```

关闭指定的信道

参数

- tunnelId

- 要关闭的信道 ID

返回数据

Promise

对象

调用示例

```
tunnel.closeTunnel('abcdefghijkl')
```

## 图片上传

```
qcloud.uploader(req)
```

图片上传接口，直接从请求中读取图片文件，并上传到指定的 COS 中。完全兼容直接接入微信小程序端的

wx.upload

接口。

参数

- req

-

http.IncomingMessage

实例化对象

返回数据

Promise

对象

返回数据格式

```
{
  imgUrl: string, // 图片访问地址
  size: number, // 图片大小
  mimeType: string, // 图片 MIME 类型
  name: string // 图片名称
}
```

调用示例

```
// express
module.exports = (req, res) => {
  qcloud.uploader(req).then(data => {
    console.log(data)
    // {
    // imgUrl: 'http://test-121000000.cosgz.myqcloud.com/abcdef.jpg',
    // size: 1048576,
    // mimeType: 'image/jpeg',
    // name: 'abcdef.jpg'
    // }
  })
}
```

## 数据库

由于 SDK 内部使用 [Knex.js](#) 连接数据库，SDK 暴露的 MySQL 实例就是 Knex.js 连接实例，具体使用方法可以查看 [Knex.js 文档](#)

## 客服消息

qcloud.message.checkSignature(signature, timestamp, nonce)

客服消息签名校验接口，具体文档可以参考[微信接入指引](#)

### 参数

- signature

- 请求的

query

中的

signature

- timestamp

- 请求的

query

中的

timestamp

- nonce

- 请求的

query

中的

nonce

返回数据

boolean

- 签名是否有效

调用示例

```
const { message: { checkSignature } } = qcloud
```

```
/**
 * 响应 GET 请求（响应微信配置时的签名检查请求）
 */
router.get('/message', ctx => {
  const { signature, timestamp, nonce, echostr } = ctx.query
  if (checkSignature(signature, timestamp, nonce)) ctx.body = echostr
  else ctx.body = 'ERR_WHEN_CHECK_SIGNATURE'
})

// POST 请求的路由用来接收消息
router.post('/message', (ctx, next) {
  // 检查签名，确认是微信发出的请求
  const { signature, timestamp, nonce } = ctx.query
  if (!checkSignature(signature, timestamp, nonce)) ctx.body = 'ERR_WHEN_CHECK_SIGNATURE'

  /**
   * 解析微信发送过来的请求体
   * 可查看微信文档：https://mp.weixin.qq.com/debug/wxadoc/dev/api/custommsg/receive.html#接收消息和事件
   */
  const body = ctx.request.body

  ctx.body = 'success'
})
```



## PHP SDK 文档

### SDK 配置

在使用本 SDK 提供的其他 API 之前，需调用以下和配置项相关的 API 进行初始化。

#### 命名空间

QCloud\_WeApp\_SDK

#### API

Conf::setup(array \$config)

可以使用本方法批量设置所有配置。

#### 参数

- appId
  - 可选。微信小程序的 AppID
- appSecret
  - 可选。微信小程序的 AppSecret
- useQcloudLogin
  - 必填。是否使用腾讯云代理登录小程序。会话登录需要使用小程序的 AppID 和 AppSecret 来解密用户信息，腾讯云提供使用腾讯云 AppID 和 AppSecret 代理请求微信进行解密。如果该项为



false

, 则需填写微信小程序 AppID 和 AppSecret。默认为

true

- mysql

- 必填。MySQL 配置。不填则使用小程序解决方案分配机器中默认的 MySQL，若使用自行部署的 MySQL 数据库，则需提供一个类型为

object

的配置，具体配置项如下：

- host

- 必填。MySQL 主机名

- user

- 必填。MySQL 用户名

- db

- 必填。MySQL 数据库名

- pass

- 必填。MySQL 密码，若使用了腾讯云微信小程序解决方案，开发环境下，MySQL

的初始密码为您的微信小程序 appid

- port

- 选填。MySQL 端口 ( 默认 : 3306 )

- char

- 选填。MySQL 编码

- COS

- 必填。腾讯云对象存储配置信息，用于上传模块使用。

- region

- 必填。COS 的地域

- fileBucket

- 必填。COS 的 bucket 名

- uploadFolder

- 必填。COS 上传文件夹名

- maxSize

- 选填。COS 上传最大大小，默认 5M (单位：M)

- field

- 选填。COS 上传是 field 名称，默认为

file

- serverHost

- 必填。当前服务器的 hostname

- tunnelServerUrl

- 必填。信道服务器地址

- tunnelSignatureKey

- 必填。信道服务签名密钥

- qcloudAppId

- 必填。腾讯云 AppId

- qcloudSecretId

- 必填。腾讯云 SecretId

- qcloudSecretKey

- 必填。腾讯云 SecretKey

- wxMessageToken

- 必填。微信客服消息通知 token

- wxLoginExpires

- 可选。微信登录态有效期，默认 7200 秒（单位：秒）

如果购买了腾讯云小程序解决方案，配置项中

serverHost

,

tunnelServerUrl

,

tunnelSignatureKey

,

qcloudAppId

,

qcloudSecretId

,

qcloudSecretKey

,

wxMessageToken

由腾讯云自动下发到您的服务器上。

自动下发的 SDK 配置文件地址：

/data/release/sdk.config.json

返回值

void

## 会话服务

命名空间

QCloud\_WeApp\_SDK\Auth

API

LoginService::login()

该静态方法用于处理用户登录。

参数

无

返回值

登录成功时，返回：

```
array(  
  'loginState' => 1,  
  'userinfo' => array(  
    // 第三方 key  
    'skey' => 'fy89ayri3h2ifs'  
    // 微信用户信息  
    'userinfo' => array(...),  
  ),  
)
```

登录失败时，返回：

```
array(  
  'loginState' => 0,  
  'error' => '失败原因'  
)
```

LoginService::check()

该静态方法用于校验登录态。

参数

无

返回值

校验登录态成功时，返回：

```
array(  
    'loginState' => 1,  
    'userinfo' => array(  
        // 微信用户信息  
    ),  
)
```

校验登录态失败时，返回：

```
array(  
    'loginState' => 0,  
    'error' => '失败原因'  
)
```

## 信道服务

### 命名空间

QCloud\_WeApp\_SDK\Tunnel

### API

#### ITunnelHandler

处理信道请求需实现该接口，接口定义和描述如下：

```
interface ITunnelHandler {  
    /*-----  
    * 初始化时传入用户信息
```

```
*-----  
* @param array $userinfo 用户信息  
*-----  
*/  
public function __construct ($userinfo);  
  
/*-----  
* 在客户端请求 WebSocket 信道连接之后会调用该方法  
* 此时可以把信道 ID 和用户信息关联起来  
*-----  
* @param string $tunnelId 信道 ID  
* @param string $tunnelUrl 信道地址  
*-----  
*/  
public function onRequest($tunnelId, $tunnelUrl);  
  
/*-----  
* 在客户端成功连接 WebSocket 信道服务之后会调用该方法  
* 此时可以通知所有其它在线的用户当前总人数以及刚加入的用户是谁  
*-----  
* @param string $tunnelId 信道 ID  
*-----  
*/  
public function onConnect($tunnelId);  
  
/*-----  
* 客户端推送消息到 WebSocket 信道服务器上后会调用该方法  
* 此时可以处理信道的消息  
*-----  
* @param string $tunnelId 信道 ID  
* @param string $type 消息类型  
* @param mixed $content 消息内容  
*-----  
*/
```



```
public function onMessage($tunnelId, $type, $content);

/*-----
 * 客户端关闭 WebSocket 信道或者被信道服务器判断为已断开后会调用该方法
 * 此时可以进行清理及通知操作
 *-----
 * @param string $tunnelId 信道 ID
 *-----
 */
public function onClose($tunnelId);
}
```

TunnelService::handle(ITunnelHandler \$handler[, array \$options])

该静态方法用于处理信道请求。

参数

- \$handler

- 该参数须实现接口

ITunnelHandler

(必填)

- \$options

- 该参数支持的可选选项如下：

- checkLogin

- 是否校验登录态 ( 默认为

FALSE

)

返回值

void

当

checkLogin

为

FALSE

时, 传递给

ITunnelHandler->onRequest

的参数

\$userInfo

值为

NULL

。

TunnelService::broadcast(array \$tunnelIds, string \$messageType, mixed \$messageContent)

该静态方法用于广播消息到多个信道。

参数

- \$tunnelIds
  - 要广播消息的信道 ID 列表 (必填)
- \$messageType
  - 要广播消息的消息类型 (必填)
- \$messageContent
  - 要广播消息的消息内容 (必填)

返回值

消息广播成功时，返回：

```
array(  
  'code' => 0,  
  'message' => 'OK',  
  'data' => array(  
    // 广播消息时无效的信道 IDs  
    'invalidTunnelIds' => array(...),  
  ),  
)
```

消息广播失败时，返回：

```
array(  
  'code' => '失败错误码 (非0)',  
  'message' => '失败原因',  
)
```

TunnelService::emit(string \$tunnelId, string \$messageType, mixed \$messageContent)

该静态方法用于发送消息到指定信道。

参数

- \$tunnelId
  - 要发送消息的信道 ID (必填)
- \$messageType
  - 要发送消息的消息类型 (必填)
- \$messageContent
  - 要发送消息的消息内容 (必填)

返回值

消息发送成功时，返回：

```
array(  
  'code' => 0,  
  'message' => 'OK',  
)
```

消息发送失败时，返回：

```
array(  
  'code' => '失败错误码 (非0)',  
  'message' => '失败原因',  
)
```

TunnelService::closeTunnel(string \$tunnelId)

该静态方法用于关闭指定信道。

参数

- \$tunnelId  
- 要关闭的信道 ID (必填)

返回值

信道关闭成功时，返回：

```
array(  
  'code' => 0,  
  'message' => 'OK',  
)
```

信道关闭失败时，返回：

```
array(  
  'code' => '失败错误码 (非0)',  
  'message' => '失败原因',  
)
```

## MySQL

### 命名空间

QCloud\_WeApp\_SDK\Mysql

### API

MySQL::getInstance()

获取 SDK 连接数据库实例，这个是个 [PDO 连接实例](#)。

#### 参数

无

#### 返回值

返回 PDO Instance

Mysql::insert(\$tableName, \$data)

向数据库中插入数据

#### 参数

- \$tableName
  - 要操作的数据表名（必填）
- \$data
  - 要插入的数据（key-value 的 array 类型）

返回值

受影响的行数（数值类型）。

示例

```
use QCloud_WeApp_SDK\Mysql\Mysql as DB;
```

```
DB::insert('tableName', [  
    'nickname' => 'Jason',  
    'age' => 21  
]);
```

```
Mysql::select($tableName[, $columns = ['*'], $conditions = '', $operator = 'and', $suffix = ''])
```

从数据库中查询多条数据

参数

- `$tableName`
  - 要操作的数据表名（必填）
- `$columns`
  - 查询出来的列名
- `$conditions`
  - 查询条件，支持 string、array 和 key-value array 类型

- \$operator

- 条件之间的操作符

- \$suffix

- SQL 语句的后缀，可以用来插入 order、limit 等

返回值

返回一个包含结果集中所有行的数组。

示例

```
use QCloud_WeApp_SDK\Mysql\Mysql as DB;
```

```
// 条件为字符串
```

```
$rows = DB::select('tableName', ['*'], 'nickname = "Jason"');
```

```
// 条件为数组
```

```
$rows = DB::select('tableName', ['*'], ['nickname = "Jason"']);
```

```
// 条件为 key-value 数组
```

```
$rows = DB::select('tableName', ['*'], ['nickname' => 'Jason']);
```

```
// 查询结果
```

```
// $rows > [['nickname' => 'Jason', 'age' => 21]]
```

```
Mysql::row($tableName[, $columns = ['*'], $conditions = "", $operator = 'and', $suffix = ''])
```

从数据库中查询单条数据

参数



- \$tableName
  - 要操作的数据表名（必填）
- \$columns
  - 查询出来的列名
- \$conditions
  - 查询条件，支持 string、array 和 key-value array 类型
- \$operator
  - 条件之间的操作符
- \$suffix
  - SQL 语句的后缀，可以用来插入 order、limit 等

#### 返回值

返回一个包含结果集中所有行的第一行。

#### 示例

```
use QCloud_WeApp_SDK\Mysql\Mysql as DB;  
  
// 条件为字符串  
$rows = DB::row('tableName', ['*'], 'nickname = "Jason"');
```

```
// 条件为数组
```

```
$rows = DB::row('tableName', ['*'], ['nickname = "Jason"']);
```

```
// 条件为 key-value 数组
```

```
$rows = DB::row('tableName', ['*'], ['nickname' => 'Jason']);
```

```
// 查询结果
```

```
// $rows > ['nickname' => 'Jason','age' => 21]
```

```
Mysql::update($tableName, $updates[, $conditions = "", $operator = 'and', $suffix = ''])
```

从数据库中查询单条数据

参数

- \$tableName
  - 要操作的数据表名（必填）
- \$updates
  - 更新的数据对象
- \$conditions
  - 查询条件，支持 string、array 和 key-value array 类型
- \$operator
  - 条件之间的操作符

- \$suffix

- SQL 语句的后缀，可以用来插入 order、limit 等

返回值

受影响的行数（数值类型）。

示例

```
use QCloud_WeApp_SDK\Mysql\Mysql as DB;
```

```
// 条件为字符串
```

```
$rows = DB::update('tableName', ['age' => 22], 'nickname = "Jason"');
```

```
// 条件为数组
```

```
$rows = DB::update('tableName', ['age' => 22], ['nickname = "Jason"']);
```

```
// 条件为 key-value 数组
```

```
$rows = DB::update('tableName', ['age' => 22], ['nickname' => 'Jason']);
```

```
// 查询结果
```

```
// $rows > 1
```

```
Mysql::delete($tableName, $conditions[, $operator = 'and', $suffix = ''])
```

从数据库中删除数据

参数

- \$tableName

- 要操作的数据表名（必填）

- \$conditions
  - 查询条件，支持 string、array 和 key-value array 类型
- \$operator
  - 条件之间的操作符
- \$suffix
  - SQL 语句的后缀，可以用来插入 order、limit 等

返回值

受影响的行数（数值类型）。

示例

```
use QCloud_WeApp_SDK\Mysql\Mysql as DB;

// 条件为字符串
$rows = DB::delete('tableName', 'nickname = "Jason"');

// 条件为数组
$rows = DB::delete('tableName', ['nickname = "Jason"']);

// 条件为 key-value 数组
$rows = DB::delete('tableName', ['nickname' => 'Jason']);

// 查询结果
// $rows > 1
```

## COS 对象存储 SDK

### 命名空间

QCloud\_WeApp\_SDK\Cos

### API

CosAPI::getInstance()

获取 COS 初始化实例

参数

无

示例

```
use \QCloud_WeApp_SDK\Cos\CosAPI as Cos;
```

```
$cosClient = Cos::getInstance();
```

```
$cosClient->upload('mybucket', 'test.txt', 'Hello World')->toArray();
```

更多关于

Cos::getInstance()

返回 COS 实例的 API，可以查看 [COS PHP SDK V5 文档](#)。