验证码

接入指引

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

接入指引
客户端接入
Web 客户端接入

最近更新时间：2025-11-10 16:25:41

> **注意：**
> 自2024年11月04日起，验证码2.0正式发布，启用验证码2.0 JS 地址：`https://turing.captcha.qcloud.com/TJCaptcha.js`，详情请参见天御验证码 2.0版发布公告。原有验证码1.0 JS 地址：`https://turing.captcha.qcloud.com/TCaptcha.js` 仍正常提供服务，用户可根据自身需求选择使用。

前提条件

客户端接入前，需完成新建验证，并在**验证列表**获取所需的 CaptchaAppId 以及 AppSecretKey。步骤如下：

1. 登录 验证码控制台，左侧导航栏选择**图形验证** > **验证管理**，进入验证管理页面。
2. 单击**新建验证**，根据业务场景需求，设置验证名称、客户端类型、验证方式等参数。
3. 单击**确定**，完成新建验证，即可在验证列表中查看验证码 CaptchaAppId 及 AppSecretKey。

代码示例

以下代码示例，单击**验证**，激活验证码，并弹窗展示验证结果。

> **注意**
> 该示例未展示调用票据校验 API 的逻辑。业务客户端完成验证码接入后，业务服务端需二次核查验证码票据结果（未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果），详情请参见：接入票据校验（Web 及 App）。

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Web 前端接入示例</title>
  <!-- 验证码程序依赖（必须）。请勿修改以下程序依赖，如通过其他手段规避加载，会导致验证码无法正常更新，对抗能力无法保证，甚至引起误拦截。 -->
```

```
  <script src="https://turing.captcha.qcloud.com/TJCaptcha.js"></script>
</head>

<body>
  <button id="CaptchaId" type="button">验证</button>
</body>

<script>

  // 定义回调函数
  function callback(res) {
    // 第一个参数传入回调结果，结果如下：
    // ret          Int         验证结果，0：验证成功。2：用户主动关闭验证码。
    // ticket       String      验证成功的票据，当且仅当 ret = 0 时 ticket 有值。
    // CaptchaAppId        String      验证码应用ID。
    // bizState     Any         自定义透传参数。
    // randstr      String      本次验证的随机串，后续票据校验时需传递该参数。
    // verifyDuration      Int     验证码校验接口耗时（ms）。
    // actionDuration      Int     操作校验成功耗时（用户动作+校验完成）(ms)。
    // sid      String    链路sid。

    console.log('callback:', res);


    // res（用户主动关闭验证码）= {ret: 2, ticket: null}
    // res（验证成功） = {ret: 0, ticket: "String", randstr: "String"}
    // res（请求验证码发生错误，验证码自动返回trerror_前缀的容灾票据） = {ret: 0,
ticket: "String", randstr: "String",  errorCode: Number, errorMessage:
"String"}
    // 此处代码仅为验证结果的展示示例，真实业务接入，建议基于ticket和errorCode情况
做不同的业务处理
    if (res.ret === 0) {
      // 复制结果至剪切板
      var str = '【randstr】->【' + res.randstr + '】      【ticket】->【'
+ res.ticket + '】';
      var ipt = document.createElement('input');
      ipt.value = str;
      document.body.appendChild(ipt);
      ipt.select();
      document.execCommand("Copy");
      document.body.removeChild(ipt);
      alert('1. 返回结果（randstr、ticket）已复制到剪切板，ctrl+v 查看。2. 打开
浏览器控制台，查看完整返回结果。');
```

```
    }
  }

  // 定义验证码js加载错误处理函数
  function loadErrorCallback() {
    var appid = '您的CaptchaAppId';
    // 生成容灾票据或自行做其它处理
    var ticket = 'trerror_1001_' + appid + '_' + Math.floor(new
Date().getTime() / 1000);
    callback({
      ret: 0,
      randstr: '@' + Math.random().toString(36).substr(2),
      ticket: ticket,
      errorCode: 1001,
      errorMessage: 'jsload_error'
    });
  }

  // 定义验证码触发事件
  window.onload = function () {
    document.getElementById('CaptchaId').onclick = function () {
      try {
        // 生成一个验证码对象
        // CaptchaAppId：登录验证码控制台，从【验证管理】页面进行查看。如果未创建
过验证，请先新建验证。注意：不可使用客户端类型为小程序的CaptchaAppId，会导致数据统计
错误。
        //callback：定义的回调函数
        var captcha = new TencentCaptcha('您的验证码CaptchaAppId',
callback, {
          userLanguage: 'zh-cn',
          showFn: (ret) => {
            const {
              duration, // 验证码渲染完成的耗时(ms)
              sid, // 链路sid
            } = ret;
          },
        });
        // 调用方法，显示验证码
        captcha.show();
      } catch (error) {
        // 加载异常，调用验证码js加载错误处理函数
        loadErrorCallback();
      }
```

```
    }
  }
</script>

</html>
```

接入说明

步骤1：动态引入验证码 JS

Web 页面需动态引入验证码 JS，在业务需要验证时，唤起验证码进行验证。

```
<!-- 动态引入验证码JS示例 -->
<script src="https://turing.captcha.qcloud.com/TJCaptcha.js"></script>
```

> **注意**
> 必须动态引入验证码 JS。如通过其他手段规避动态加载，会导致验证码无法正常更新，对抗能力无法保证，甚至引起误拦截。
> 如果使用验证码1.0，请将JS地址改为https://turing.captcha.qcloud.com/TCaptcha.js，其他无需变更。

步骤2：创建验证码对象

引入验证码 JS 后，验证码会在全局注册一个 `TencentCaptcha` 类，业务方可以使用这个类自行初始化验证码，并对验证码进行显示或者隐藏。

> **注意**
> 触发验证码的元素不要使用 `id="TencentCaptcha"` ，TencentCaptcha 属于系统默认 id，用来兼容验证码旧接入方式。

构造函数

```
new TencentCaptcha(CaptchaAppId, callback, options);
```

参数说明

参数名	值类型	说明
CaptchaAppId	String	验证码 CaptchaAppId：登录 验证码控制台 ，在验证管理页面进行查看。如果未创建过验证，请先新建验证。

		注意：不可使用客户端类型为小程序的 CaptchaAppId，会导致数据统计错误。
callback	Function	验证码回调函数，详情请参见 callback 回调函数 。
options	Object	验证码外观配置参数，详情请参见 options 外观配置参数 。

callback 回调函数

验证结束后，会调用业务传入的回调函数，并在第一个参数中传入回调结果。回调结果字段说明如下：

字段名	值类型	说明
ret	Int	验证结果，0：验证成功。2：用户主动关闭验证码。 **说明：** 容灾场景下验证结果返回：0，详情请参见 业务容灾方案（Web 及 App）。
ticket	String	验证成功的票据，当且仅当 ret = 0 时 ticket 有值。
appid	String	验证码应用 ID。
bizState	Any	自定义透传参数。
randstr	String	本次验证的随机串，后续票据校验时需传递该参数。
errorCode	Number	错误 code ，详情请参见 回调函数 errorCode 说明 。
errorMessage	String	错误信息。
verifyDuration	Number	验证码校验接口耗时（ms）
actionDuration	Number	用户操作校验成功耗时（ms）
sid	String	链路 sid

回调函数errorCode说明

errorCode	说明
1001	TJCaptcha.js 加载错误

1002	调用 show 方法超时
1003	中间 js 加载超时
1004	中间 js 加载错误
1005	中间 js 运行错误
1006	拉取验证码配置错误/超时（网络超时，欠费，CaptchaAppid 加密配置错误）
1007	iframe 加载超时
1008	iframe 加载错误
1009	jQuery 加载错误
1010	滑块 js 加载错误
1011	滑块 js 运行错误
1012	刷新连续错误3次
1013	验证网络连续错误3次
1085	无感验证超时/失败

options 外观配置参数

options 参数用于对验证码进行定制外观设置，默认可以设置为空。

⚠ 注意

- 验证码弹窗内部不支持调整样式大小，如果需要调整，可在弹窗最外层用 class=tcaptcha-transform 的元素设置 transform:scale();（更改大小可能会导致验证码图片失真，请谨慎修改）。举例如下：

```
.tcaptcha-transform{
transform: scale(0.9);
}
```

- 验证码服务更新可能会改变元素的 id、class 等属性，请勿依赖其他验证码元素属性值覆盖样式。
- 如果手机原生端有设置左右滑动手势，需在调用验证码 show 方法前禁用，验证完成后再打开，防止与验证码滑动事件冲突。

配置名	值类型	说明
bizState	Any	自定义透传参数，业务可用该字段传递少量数据，该字段的内容会被带入 callback 回调的对象中。
enableDarkMode	Boolean|String	开启自适应深夜模式或强制深夜模式。（**VTT 空间语义验证暂不支持该功能**） 1. 开启自适应深夜模式: {"enableDarkMode": true} 2. 强制深夜模式: {"enableDarkMode": 'force'}
sdkOpts	Object	示例 {"width": 140, "height": 140} 仅支持移动端原生 webview 调用时传入，用来设置验证码 loading 加载弹窗的大小（**注意，并非验证码弹窗大小**）。
ready	Function	验证码加载完成的回调，回调参数为验证码实际的宽高（单位：px）： {"sdkView": { "width": number, "height": number }} 该参数仅为查看验证码宽高使用，**请勿使用此参数直接设定宽高**。
needFeedBack	Boolean|String	隐藏帮助按钮或自定义帮助按钮链接。（**VTT 空间语义验证暂不支持自定义链接**） 隐藏帮助按钮: {"needFeedBack": false } 自定义帮助链接: {"needFeedBack": 'url地址' }
loading	Boolean	是否在验证码加载过程中显示loading框。不指定该参数时，默认显示loading框。 • 显示 loading 框: {"loading": true} • 不显示 loading 框: {"loading": false}（展示方式为嵌入式时不支持配置）
userLanguage	String	指定验证码提示文案的语言，优先级高于控制台配置。（**VTT 空间语义、文字点选验证暂不支持语言配置**） 支持传入值同 navigator.language 用户首选语言，大小写不敏感。 详情参见 userLanguage 配置参数 。
type	String	定义验证码展示方式。 • popup（默认）弹出式，以浮层形式居中弹出展示验证码。 • embed 嵌入式，以嵌入指定容器元素中的方式展示验证码。 详情参见 嵌入式验证码参数配置示例 。
aidEncrypted	String	CaptchaAppId 加密校验串，可选参数。详情见 CaptchaAppid 加密校验能力接入指引 。

showFn	Function	duration 渲染耗时 + sid 回调函数。

嵌入式验证码参数配置示例

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>验证码-内嵌接入</title>
    <script src="https://turing.captcha.qcloud.com/TJCaptcha.js">
</script>
  </head>
  <body>
    <!-- 验证码指定的内嵌容器 可自定义到任何位置 -->
    <div id="targetEmbed"></div>
    <script>
      function globalCallback(res) {
        console.log('captcha success', res);
      }
      function errorCallback(res) {
        console.log('errorCallback', res);
      }
      window.onload = function () {
        try {
          const captcha = new
TencentCaptcha(document.getElementById('targetEmbed'), '用户
CaptchaAppid', globalCallback, {
            type: 'embed',
          });
          captcha.show();
        } catch (error) {
          console.log('error', error);
          errorCallback(error);
        }
      };
    </script>
  </body>
</html>
```

userLanguage 配置参数（除滑动拼图验证外，其他验证形式只支持中英文）

参数名	说明
zh-cn	简体中文
zh-hk	繁体中文（中国香港）
zh-tw	繁体中文（中国台湾）
en	英文
ar	阿拉伯语
my	缅甸语
fr	法语
de	德语
he	希伯来语
hi	印地语
id	印尼语
it	意大利语
ja	日语
ko	朝鲜语/韩语
lo	老挝语
ms	马来语
pl	波兰语
pt	葡萄牙语
ru	俄语
es	西班牙语
th	泰语
tr	土耳其语
vi	越南语

fil	菲律宾语
ur	乌尔都语

步骤3：调用验证码实例方法

TencentCaptcha 的实例提供一些操作验证码的常用方法：

方法名	说明	传入参数	返回内容
show	显示验证码，可以反复调用。	无	无
destroy	隐藏验证码，可以反复调用。	无	无
getTicket	获取验证成功后的 ticket。	无	`Object: {"CaptchaAppId":"","ticket":""}`

步骤4：容灾处理

为保障验证码 Captcha 服务端异常时不阻塞客户网站正常业务流程，建议参考如下方式接入验证码。

1. 定义错误处理函数。

```
// 错误处理函数作用：在脚本加载或初始化错误时，保障事件流程正常
// 函数定义需在script加载前
function loadErrorCallback() {
 var appid = ''
 // 生成容灾票据或自行做其它处理
 var ticket = 'trerror_1001_' + appid + Math.floor(new
Date().getTime() / 1000);
 callback({
     ret: 0,
     randstr: '@'+ Math.random().toString(36).substr(2),
     ticket:ticket,
     errorCode: 1001,
     errorMessage: 'jsload_error',
 });
}
```

2. 验证码返回错误时，调用错误处理函数。

```
try {
 // 生成一个验证码对象
 var captcha = new TencentCaptcha('您的验证码CaptchaAppId', callback,
{});
 // 调用方法，显示验证码
 captcha.show();
} catch (error) {
 // 加载异常，调用验证码js加载错误处理函数
 loadErrorCallback();
}
```

3. 回调函数根据 ticket 和 errorCode（而非 ret）的情况做处理。

```
function callback(res) {
 // res（用户主动关闭验证码）= {ret: 2, ticket: null}
 // res（验证成功） = {ret: 0, ticket: "String", randstr: "String"}
 // res（请求错误，返回trerror_前缀的容灾票据） = {ret: 0, ticket:
"String", randstr: "String",  errorCode: Number, errorMessage:
"String"}
 if (res.ticket){
     //根据errorCode情况做特殊处理
     if(res.errorCode === xxxxx){
         //自定义容灾逻辑（例如跳过这次验证）
     }
   }
}
```

完整容灾方案，详情请见 业务容灾方案 。

> **注意**
> 业务客户端完成验证码接入后，服务端需二次核查验证码票据结果（未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果），详情请参见：接入票据校验（Web 及 App）。

React 架构接入示例

1. 在 HTML 模板文件中引入验证码 JS。

```
<!-- 验证码程序依赖（必须）。请勿修改以下程序依赖，如使用本地缓存，或通过其他手段规
避加载，会导致验证码无法正常更新，对抗能力无法保证，甚至引起误拦截。 -->
<script src="https://turing.captcha.qcloud.com/TJCaptcha.js"></script>
```

2. 调用验证码。

```
import React from 'react';

import logo from './logo.svg';

import './App.css';

interface ICaptchaResult {
  ret: number;

  ticket: string;

  randstr: string;

  CaptchaAppId?: string;

  bizState?: string;

  errorCode?: number;

  errorMessage?: string;
}

function App() {
  // 定义回调函数

  function callback(res: ICaptchaResult) {
    // 第一个参数传入回调结果，结果如下：

    // ret Int 验证结果，0：验证成功。2：用户主动关闭验证码。

    // ticket String 验证成功的票据，当且仅当 ret = 0 时 ticket 有值。

    // CaptchaAppId String 验证码应用ID。

    // bizState Any 自定义透传参数。

    // randstr String 本次验证的随机串，后续票据校验时需传递该参数。

    console.log('callback:', res);
```

```
    // res（用户主动关闭验证码）= {ret: 2, ticket: null}

    // res（验证成功） = {ret: 0, ticket: "String", randstr: "String"}

    // res（请求验证码发生错误，验证码自动返回trerror_前缀的容灾票据） = {ret: 0, ticket: "String", randstr: "String", errorCode: Number, errorMessage: "String"}

    // 此处代码仅为验证结果的展示示例，真实业务接入，建议基于ticket和errorCode情况做不同的业务处理

    if (res.ret === 0) {
      // 复制结果至剪切板

      var str = '【randstr】->【' + res.randstr + '】 【ticket】->【' + res.ticket + '】';

      var ipt = document.createElement('input');

      ipt.value = str;

      document.body.appendChild(ipt);

      ipt.select();

      document.execCommand('Copy');

      document.body.removeChild(ipt);

      alert('1. 返回结果（randstr、ticket）已复制到剪切板，ctrl+v 查看。 2. 打开浏览器控制台，查看完整返回结果。');
    }
  }

  // 定义验证码js加载错误处理函数

  function loadErrorCallback() {
    var appid = '您的CaptchaAppId';

    // 生成容灾票据或自行做其它处理

    var ticket = 'trerror_1001_' + appid + '_' + Math.floor(new Date().getTime() / 1000);
```

```
      callback({
        ret: 0,

        randstr: '@' + Math.random().toString(36).substr(2),

        ticket: ticket,

        errorCode: 1001,

        errorMessage: 'jsload_error',
      });
    }

    function onCaptchaShow() {
      try {
        // 生成一个验证码对象

        // CaptchaAppId：登录验证码控制台，从【验证管理】页面进行查看。如果未创建过验证，请先新建验证。注意：不可使用客户端类型为小程序的CaptchaAppId，会导致数据统计错误。

        // callback：定义的回调函数

        const captcha = new TencentCaptcha('您的CaptchaAppId', callback, {});

        // 调用方法，显示验证码

        captcha.show();
      } catch (error) {
        // 加载异常，调用验证码js加载错误处理函数

        loadErrorCallback();
      }
    }

    return (
      <div className='App'>
        <button className='captcha-btn' onClick={onCaptchaShow}>
          弹出验证码
        </button>
      </div>
```

```
    );
}

export default App;
```

CaptchaAppid 加密校验能力接入指引

通过前端传递加密符（非必选能力，可根据安全性需求选择性接入），可以有效防止因 CaptchaAppid 泄露而造成的资源盗刷。

加密规则

接口通过 aidEncrypted 参数（即加密后的字符串标识），支持采用加密模式传递验证码业务 CaptchaAppid 进行校验。
当控制台强制校验开启时，触发加密模式。由客户的服务端进行加密后下发对应加密字符串到客户的前端，由客户前端将加密后的字符串传参到验证码侧。
加密步骤如下：

1. 登录 验证码控制台，左侧导航栏选择**图形验证** > **验证管理**，进入验证管理页面。
2. 在验证管理页面，选择所需 AppSecretKey，作为密钥 key。当 key 小于32字节时，循环填充同一个 AppSecretKey 补齐到32字节作为密钥 key。

3. 随机生成16字节的 IV，结合步骤1中得到的密钥 Key，对业务数据对象进行 AES256加密，加密模式为 CBC/PKCS7Padding，加密的业务数据对象为验证码业务 CaptchaAppid&时间戳&密文过期时间，时间戳为秒级当前 unix 时间戳，不可为未来时间，密文过期时间单位为秒最大值为86400秒，得到加密后的串为 CaptchaAppidEncrypted。
4. 对步骤2中 IV 拼接 AES256加密得到的字节数组 CaptchaAppidEncrypted 进行 Base64编码，即 Base64（IV+CaptchaAppidEncrypted），中间无连接符，得到最终加密后的业务参数请求字符串即为 aidEncrypted。

加密结果示例

类型	示例值
Captchaappid	123456789
时间戳	1710144972
过期时间	86400秒
iv	0123456789012345
AppSecretKey	123456789101112131415151 6

循环填充的密钥 key	123456789101112131415151612345 67
加密的业务数据对象	123456789&1710144972&86400
加密后的最终字符串 aidEncrypted	MDEyMzQ1Njc4OTAxMjM0NWvZ11atw+1u zYmolyt5rAQVPyMK9ZDavskPw5hcayeT

代码示例

服务端加密

加密规则为：Base64(IV + AES256(CaptchaAppid&时间戳&密文过期时间，IV， Key))，即使用随机生成16字节的 IV、及根据 AppSecretKey 循环填充后得到32字节的加密 Key，使用 AES256算法加密模式 CBC/PKCS7Padding，对明文数据 **CaptchaAppid&时间戳&密文过期时间**进行加密。

```
 #!/usr/bin/env python
# -*- coding: utf-8 -*-
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import base64
import time


def encrypt(plaintext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv) # 创建一个新的AES cipher，CBC模
式
    ciphertext = cipher.encrypt(pad(plaintext.encode(),
AES.block_size))# 对数据进行填充，然后加密。pad(plaintext.encode(),
AES.block_size)将检查明文长度是否为16字节倍数，若非16字节倍数，将使用PKCS7填充方式
将明文填充到16字节倍数。
    ciphertextBase64 = base64.b64encode(iv + ciphertext).decode('utf-8')
# iv拼接加密后的数据，并进行Base64返回后进行传输。
    return ciphertextBase64


# 加密示例
AppSecretKey = b'1234567891011121314151516'    #客户从控制台获取对应验证码账号
下的AppSecretKey,25位
remainder = 32 % AppSecretKey.__len__()  # 计算需要补充的密钥长度
key = AppSecretKey + AppSecretKey[:remainder]       # 最终加密key，补充满32
位。

Captchaappid = "123456789" # 客户自身的验证码CaptchaAppid
```

```
curTime = 1710144972    # 获取当前的时间戳，示例暂设置成固定时间戳，客户应该设置成最新的时间戳，使用int(time.time())
expireTime = 86400 # 过期时间设置，这里暂设置成该值，客户根据自身需要设置

plaintext = Captchaappid + "&" + str(curTime) + "&" + str(expireTime)   # 拼接待加密的业务数据对象，CaptchaAppid&时间戳&密文过期时间

iv = "0123456789012345".encode() # 随机生成16字节的IV，这里暂设置成该值，客户使用时应该用随机生成的数据，使用os.urandom(16)

ciphertext = encrypt(plaintext, key, iv)    # 加密
print("Ciphertext (Base64):", ciphertext)   # 本示例数据将输出
MDEyMzQ1Njc4OTAxMjM0NWvZ11atw+1uzYmoIyt5rAQVPyMK9ZDavskPw5hcayeT
```

前端接入示例

```
const encryptAppid = async () => {
  /** 从后端获取加密后的 CaptchaAppid字符串 */
  const { aidEncrypted } = await fetch('/api/encryptAppid');
  /** 回调函数 */
  const callBack = (ret) => {
    console.log('ret', ret);
  };
  /** 错误回调函数 */
  const errorCb = (error) => {
    console.log('error', error);
  };
  try {
    /** 将获取的加密字符串传入aidEncrypted参数 */
    const captcha = new TencentCaptcha('123456789', callBack, {
aidEncrypted: aidEncrypted });
    captcha.show();
  } catch (error) {
    errorCb(error);
  }
};
```

常见问题

详情参见 接入相关问题 。

更多信息

您可以登录 验证码控制台 ，在页面右下角单击**快速咨询**，了解更多详细信息。

App 客户端接入
Android 接入

最近更新时间：2025-01-02 14:30:12

前提条件

客户端接入前，需完成新建验证，并在**验证列表**获取所需的 CaptchaAppId 以及 AppSecretKey。步骤如下：

1. 登录 验证码控制台，左侧导航栏选择**图形验证** > **验证管理**，进入验证管理页面。
2. 单击**新建验证**，根据业务场景需求，设置验证名称、客户端类型、验证方式等参数。
3. 单击**确定**，完成新建验证，即可在验证列表中查看验证码 CaptchaAppId 及 AppSecretKey。

接入步骤

> ⚠ **注意**
> App 客户端（Android/iOS/Harmony）当前仅支持通过 Webview 引入 H5页面进行接入。

Android 接入

Android 接入主要流程

1. 在 Android 端利用 WebView 引入 H5页面。H5 页面接入验证码，详情请参见 Web 客户端接入。
2. 在 H5 页面中，通过调用验证码 JS，渲染验证页面，并将 JS 返回的参数值传到 Android App 业务端。
3. Android App 业务端把相关参数（票据 ticket、随机数等）传入业务侧后端服务进行票据验证，详情请参见 接入票据校验(Web及App)。

Android 接入详细步骤

1. 在项目的工程中，新建一个 Activity 并导入 WebView 组件所需的包。

```
import android.webkit.WebView;
import android.webkit.WebSettings;
import android.webkit.WebViewClient;
import android.webkit.WebChromeClient;
```

2. 添加相关权限，如开启网络访问权限以及允许 App 进行非 HTTPS 请求等。

```
<uses-permission android:name="android.permission.INTERNET"/>
<application android:usesCleartextTraffic="true">...</application>
```

3. 在 Activity 的布局文件中，添加 WebView 组件。

```
<WebView
android:id="@+id/webview"
android:layout_height="match_parent"
android:layout_width="match_parent"
/>
```

4. 在项目的工程中，添加自定义 JavascriptInterface 文件，并定义一个方法用来获取相关数据。

```
import android.webkit.JavascriptInterface;
public class JsBridge {
@JavascriptInterface
public void getData(String data) {
System.out.println(data);
}
}
```

5. 在 Activity 文件中，加载相关 H5 业务页面。

```
public class MainActivity extends AppCompatActivity {
private WebView webview;
private WebSettings webSettings;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
initView();
}

private void initView() {
webview = (WebView) findViewById(R.id.webview);
webSettings = webview.getSettings();
webSettings.setUseWideViewPort(true);
webSettings.setLoadWithOverviewMode(true);
// 禁用缓存
webSettings.setCacheMode(WebSettings.LOAD_NO_CACHE);
webview.setWebViewClient(new WebViewClient(){
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
view.loadUrl(url);
```

```
return true;
}
});
// 开启js支持
webSettings.setJavaScriptEnabled(true);
webview.addJavascriptInterface(new JsBridge(), "jsBridge");
// 也可以加载本地html(webView.loadUrl("file:///android_asset/xxx.html"))
webview.loadUrl("https://x.x.x/x/");
}
}
```

6. 在 H5 业务页面中接入验证码，详情请参见 Web 客户端接入 文档，并使用 JSBridge 传回验证数据给具体业务端。

> **注意**
> 业务客户端完成验证码接入后，服务端需二次核查验证码票据结果（未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果），详情参见 接入票据校验（Web 及 App）。

常见问题

详情参见 接入相关问题 。

更多信息

您可以登录 验证码控制台，在页面右下角单击**快速咨询**，了解更多详细信息。

iOS 接入

最近更新时间：2025-08-13 17:50:52

前提条件

客户端接入前，需完成新建验证，并在**验证列表**获取所需的 CaptchaAppId 以及 AppSecretKey。步骤如下：

1. 登录 验证码控制台 ，左侧导航栏选择**图形验证** > **验证管理**，进入验证管理页面。
2. 单击**新建验证**，根据业务场景需求，设置验证名称、客户端类型、验证方式等参数。
3. 单击**确定**，完成新建验证，即可在验证列表中查看验证码 CaptchaAppId 及 AppSecretKey。

接入步骤

> **注意**
> App 客户端（Android/iOS/Harmony）当前仅支持通过 WebView 引入 H5页面进行接入。

iOS 接入主要流程

1. 在 iOS 中打开 WebView，通过 JSBridge 触发 HTML 页面 ，同时注入方法，供 HTML 调用传入验证结果。
2. 在 HTML 页面中接入验证码，详细请参见 Web 客户端接入 ，通过调用验证码 JS，渲染验证页面，并调用 iOS 注入的方法传入验证结果。
3. 通过 JSBridge 将验证结果返回到 iOS，并把相关参数（票据 ticket、随机数等）传入业务侧后端服务进行票据验证，详情请参见 接入票据校验(Web及App) 。

iOS 接入的详细操作步骤

1. 在控制器或 view 中导入 WebKit 库。

```
#import <WebKit/WebKit.h>
```

2. 创建 WebView 并渲染。

```
-(WKWebView *)webView{
if(_webView == nil){
 //创建网页配置对象
 WKWebViewConfiguration *config = [[WKWebViewConfiguration alloc]
init];
 // 创建设置对象
 WKPreferences *preference = [[WKPreferences alloc]init];
 //设置是否支持 javaScript 默认是支持的
```

```
 preference.javaScriptEnabled = YES;
 // 在 iOS 上默认为 NO，表示是否允许不经过用户交互由 javaScript 自动打开窗口
 preference.javaScriptCanOpenWindowsAutomatically = YES;
 config.preferences = preference;
 //这个类主要用来做 native 与 JavaScript 的交互管理
 WKUserContentController * wkUController = [[WKUserContentController
alloc] init];
 //注册一个name为jsToOcNoPrams的js方法 设置处理接收JS方法的对象
 [wkUController addScriptMessageHandler:self  name:@"jsToOcNoPrams"];
 [wkUController addScriptMessageHandler:self
name:@"jsToOcWithPrams"];
 config.userContentController = wkUController;
 _webView = [[WKWebView alloc] initWithFrame:CGRectMake(0, 0,
SCREEN_WIDTH, SCREEN_HEIGHT) configuration:config];
 // UI 代理
 _webView.UIDelegate = self;
 // 导航代理
 _webView.navigationDelegate = self;
 //此处即需要渲染的网页
 NSString *path = [[NSBundle mainBundle]
pathForResource:@"JStoOC.html" ofType:nil];
 NSString *htmlString = [[NSString alloc]initWithContentsOfFile:path
encoding:NSUTF8StringEncoding error:nil];
 [_webView loadHTMLString:htmlString baseURL:[NSURL fileURLWithPath:
[[NSBundle mainBundle] bundlePath]]];
}
return _webView;
}
[self.view addSubview:self.webView];
```

3. 代理方法，处理一些响应事件。

```
// 页面开始加载时调用
-(void)webView:(WKWebView *)webView didStartProvisionalNavigation:
(WKNavigation *)navigation {
}
// 页面加载失败时调用
-(void)webView:(WKWebView *)webView didFailProvisionalNavigation:
(null_unspecified WKNavigation *)navigation withError:(NSError *)error
{
[self.progressView setProgress:0.0f animated:NO];
}
// 当内容开始返回时调用
```

```
-(void)webView:(WKWebView *)webView didCommitNavigation:(WKNavigation
*)navigation {
}
// 页面加载完成之后调用
-(void)webView:(WKWebView *)webView didFinishNavigation:(WKNavigation
*)navigation {
[self getCookie];
}
//提交发生错误时调用
-(void)webView:(WKWebView *)webView didFailNavigation:(WKNavigation
*)navigation withError:(NSError *)error {
[self.progressView setProgress:0.0f animated:NO];
}
// 接收到服务器跳转请求即服务重定向时之后调用
-(void)webView:(WKWebView *)webView
didReceiveServerRedirectForProvisionalNavigation:(WKNavigation
*)navigation {
}
```

4. JS 将参数传给 OC。

```
<p style="text-align:center"> <button id="btn2" type = "button"
onclick = "jsToOcFunction()"> JS调用OC：带参数  </button> </p>
function jsToOcFunction()
{
window.webkit.messageHandlers.jsToOcWithPrams.postMessage({"params":"r
es.randstr"});
}
```

5. 将渲染好的 WebView 展示在视图上，调用验证码服务，将数据传给客户端。

```
-(void)userContentController:(WKUserContentController
*)userContentController didReceiveScriptMessage:(WKScriptMessage
*)message{
//此处message.body即传给客户端的json数据
//用message.body获得JS传出的参数体
NSDictionary * parameter = message.body;
//JS调用OC
if([message.name isEqualToString:@"jsToOcWithPrams"]){
//在此处客户端得到js透传数据 并对数据进行后续操作
parameter[@"params"]
}
```

```
}
```

> **注意**
> 业务客户端完成验证码接入后，服务端需二次核查验证码票据结果（未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果），详情请参见 接入票据校验（Web 及 App）。

常见问题

详情参见 接入相关问题 。

更多信息

您可以登录 验证码控制台 ，在页面右下角单击**快速咨询**，了解更多详细信息。

Harmony 接入

最近更新时间：2025-08-13 17:50:52

前提条件

客户端接入前，需完成新建验证，并在**验证列表**获取所需的 CaptchaAppId 以及 AppSecretKey。步骤如下：

1. 登录 验证码控制台 ，左侧导航栏选择**图形验证** > **验证管理**，进入验证管理页面。
2. 单击**新建验证**，根据业务场景需求，设置验证名称、客户端类型、验证方式等参数。
3. 单击**确定**，完成新建验证，即可在验证列表中查看验证码 CaptchaAppId 及 AppSecretKey。

接入步骤

> ⚠ **注意：**
> App 客户端（Android/iOS/Harmony）当前仅支持通过 WebView 引入 H5页面进行接入。

Harmony 接入主要流程

1. 在 Harmony 端利用 WebView 引入 H5页面。H5 页面接入验证码，详情请参见 Web 客户端接入 。
2. 在 H5 页面中，通过调用验证码 JS，渲染验证页面，并将 JS 返回的参数值传到 Harmony App 业务端。
3. Harmony App 业务端把相关参数（票据 ticket、随机数等）传入业务侧后端服务进行票据验证，详情请参见 接入票据校验(Web及App) 。
4. 体验完整接入示例 Harmony Next Demo 。

Harmony 接入详细步骤

1. 在项目的工程中，新建一个 View 视图。导入 WebView 组件所需的包并进行初始化。

```
// src/main/ets/view/Login.ets
import router from '@ohos.router';
import web_webview from '@ohos.web.webview';
import { CaptchaCbParams,CaptchaRetData } from
'../viewmodel/ParamsItem';
import Logger from '../common/utils/Logger';
import JSBridge from '../common/utils/JsBridge';

@Component
export struct LoginComponent {
  controller: web_webview.WebviewController = new
web_webview.WebviewController();
  aboutToAppear() {
    // 配置web开启调试模式
```

```
      web_webview.WebviewController.setWebDebuggingAccess(true);
    }
    ports: web_webview.WebMessagePort[] = [];
    // 初始化JS事件交互
    private jsBridge: JSBridge = new JSBridge(this.controller, this);

    // 在build方法之外定义关闭WebView的方法
    closeWebView(param: string) {
      Logger.info("接收的回调数据", param)
      const jsonObj: CaptchaCbParams = JSON.parse(param)
      const retObj: CaptchaRetData = JSON.parse(jsonObj.data)
      if (retObj.ret == 0) {
        // 验证成功有票据
        router.pushUrl({ url: CommonConstants.SUCCESS_PAGE_URL, params:
retObj })
      } else {
        // 验证失败或者主动关闭验证码无票据
        Logger.info("主动关闭验证码")
      }
      this.showWebView = false;
    }

    build() {
      Stack() {
        Web({
          src: $rawfile('captcha.html'), // 本地HTML文件路径
          controller: this.controller
        })
          .domStorageAccess(true)
          .javaScriptAccess(true)
          .javaScriptProxy(this.jsBridge.javaScriptProxy)
          .onPageBegin(() => {
            this.jsBridge.initJsBridge();
          })
          .width('360vp')
          .height('360vp')
          .backgroundColor(Color.Transparent)
          .alignSelf(ItemAlign.Center)
      }

    }
}
```

2. 添加验证码 HTML 页面文件，放置于 `src/main/resources/rawfile/captcha.html` 路径中，WebView 需要从这个路径加载文件。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <style>
    body {
      background-color: transparent;
      margin: 0;
      padding: 0;
    }
  </style>
  <title>Tencent Captcha</title>
  <script src="https://turing.captcha.qcloud.com/TCaptcha.js">
</script>
</head>
<body>
  <script type="text/javascript">
    function getAppIdParam() {
        let params = {};
        let queryString = window.location.search.slice(1); // 获取URL参
数字符串
        if (queryString) {
            queryString = queryString.split('&'); // 分割成参数数组
            queryString.forEach(item => {
                let arr = item.split('=');  // 分割每个参数键值对
                params[arr[0]] = decodeURIComponent(arr[1]);  //
decodeURIComponent用于解码URI
            });
        }
        return params['appid'] || null; // 返回'appid'参数，如果不存在则返
回null
    }
        function globalCallback(res) {
          // 与鸿蒙webview通信
          if (window.ohosCallNative &&
window.ohosCallNative.callNative) {
```

```
            window.ohosCallNative.callNative('postMessage',
JSON.stringify(res));
          }
        }
        // captcha js error callback
        function errorCallback() {
          globalCallback({
            ret: -1,
            randstr: '@' + Math.random().toString(36).substr(2),
            ticket: '',
            errorCode: 1001,
            errorMessage: 'jsload_error',
          });
        }
        window.onload = function () {
          try {
            const sdkOptions = {
              isMobile: true,
              needFeedBack: false,
              enableDarkMode: false,
              // 是否需要loading和mask蒙层
              loading: false,
              ready: function (size) {
                console.log('ready size:', JSON.stringify(size));
              }
            };
            const captcha = new TencentCaptcha('您的CatchaAppid',
globalCallback, sdkOptions);
            // 调用方法，显示验证码
            captcha.show();
          } catch (error) {
            errorCallback();
          }
        }
  </script>
</body>
</html>
```

3. 编写 WebView 页面与 App 端进行事件通信方法。

```
// src/main/ets/common/utils/JsBridge.ets
import WebView from '@ohos.web.webview';
import { code } from '../constants/CodeConstant';
```

```
import { ParamsItem } from '../../viewmodel/ParamsItem';
import { JavaScriptItem } from '../../viewmodel/JavaScriptItem';
import Logger from './Logger';
import { LoginComponent } from '../../view/LoginComponent';

/**
 * JS事件层 连接 WebView 和 ArkTS通信.
 */
export default class JsBridge {
  controller: WebView.WebviewController;
  private componentInstance: LoginComponent;
  constructor(controller: WebView.WebviewController,componentInstance:
LoginComponent) {
    this.controller = controller;
    this.componentInstance = componentInstance;
  }

  get javaScriptProxy(): JavaScriptItem {
    let result: JavaScriptItem = {
      object: {
        call: this.call
      },
      name: "JSBridgeHandle",
      methodList: ['call'],
      controller: this.controller
    }
    return result;
  }

  initJsBridge(): void {
    this.controller.runJavaScript(code);
  }


  postMessage = (params:string): Promise<string> => {
    // 可以在此将获取的票据传给页面，执行票据二次验证操作
    Logger.info("验证码回调数据,",params)
    this.componentInstance.closeWebView(params);
    return new Promise((resolve) => {
      resolve(params);
    })
  }
  call = (func: string, params: string): void => {
```

```
    const paramsObject: ParamsItem = JSON.parse(params);
    let result: Promise<string> = new Promise((resolve) =>
resolve(''));
    switch (func) {
      case 'postMessage':
        result = this.postMessage(params);
        break;
      default:
        break;
    }
    result.then((data: string) => {
      this.callback(paramsObject?.callID, data);
    })
  }
  callback = (id: number, data: string): void => {
    this.controller.runJavaScript(`JSBridgeCallback("${id}",
${JSON.stringify(data)})`);
  }
}
```

4. 在 H5 业务页面中接入验证码，详情请参见 Web 客户端接入 文档，并使用 JSBridge 传回验证数据给具体业务端。

> **注意**
> 业务客户端完成验证码接入后，服务端需二次核查验证码票据结果（未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果），详情参见 接入票据校验（Web 及 App）。

常见问题

详情参见 接入相关问题。

更多信息

您可以登录 验证码控制台，在页面右下角单击**快速咨询**，了解更多详细信息。

微信小程序接入

最近更新时间：2025-08-13 18:09:12

> **注意：**
> 为了提升验证码产品的性能和稳定性，腾讯验证码微信小程序插件已经升级至2.1.0版本。如果您有需求，可以按照以下方案进行接入。在新版本的插件中，我们将优化风险控制能力、用户体验和验证类型等功能。

前提条件

客户端接入前，需完成新建验证，并在**验证列表**获取所需的 CaptchaAppId 以及 AppSecretKey。步骤如下：

1. 登录 验证码控制台 ，左侧导航栏选择**图形验证** > **验证管理**，进入验证管理页面。
2. 单击**新建验证**，根据业务场景需求，设置验证名称、客户端类型、验证方式等参数。
3. 单击**确定**，完成新建验证，即可在验证列表中查看验证码 CaptchaAppId 及 AppSecretKey。

小程序原生语言接入

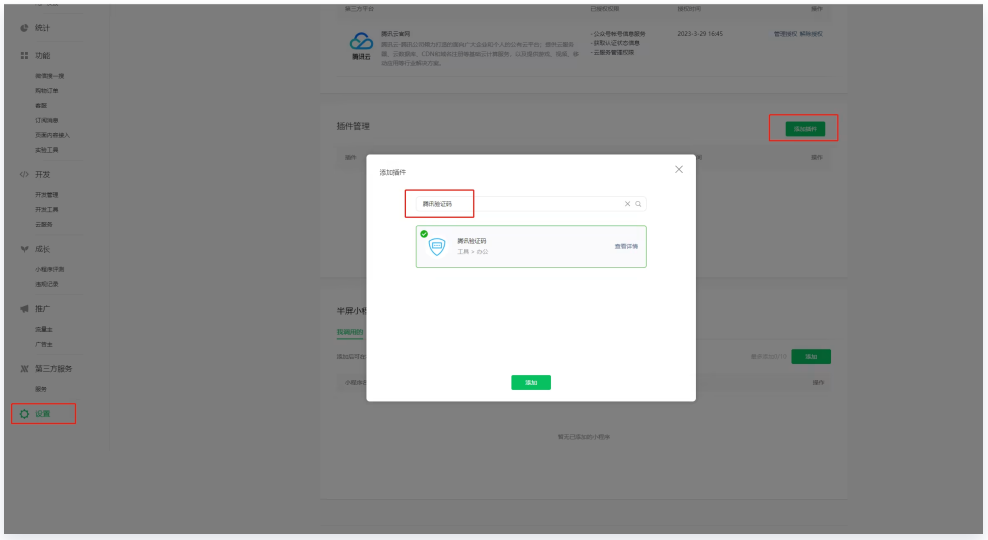
> **注意**
> - 请勿在“微信开发者工具”的“游客模式”下接入验证码。
> - 目前尚未支持 Skyline 渲染模式。

示例下载

代码示例：单击**验证**，激活验证码，并弹窗展示验证结果。
完整示例请下载： 小程序验证码接入示例 。

步骤1：添加插件

1. 用管理员身份登录 微信公众平台 ，且需使用接入小程序的相关账号。
2. 选择**设置** > **第三方设置** > **添加插件**，在搜索框内输入关键字“腾讯验证码”查找插件，并单击添加，如下图所示。



步骤2：集成插件

1. 引入验证码小程序插件。
   使用验证码插件前，需要在 `app.json` 中声明验证码小程序插件，如下：

```
{
    "plugins": {
        "captcha": {
            "version": "2.1.0", //请选择小程序插件最新版本
            "provider": "wx1fe8d9a3cb067a75"
        }
    }
}
```

2. 引入验证码小程序组件。
   在页面 `.json` 文件中需要引入自定义组件，js 代码如下：

```
{
    "usingComponents": {
      "t-captcha": "plugin://captcha/t-captcha"
    }
}
```

步骤3：使用小程序插件

1. 使用原生小程序语言接入时，需要在自定义的 `.wxml` 文件中，使用验证码插件，wxml 代码如下：

```
<!-- app-id：验证码CaptchaAppId，从腾讯云的验证码控制台中获取，在验证码控制台页面内【图形验证】>【验证列表】进行查看 -->
<t-captcha
      id="captcha"
      app-id="小程序插件验证码CaptchaAppId"
      bindverify="handlerVerify"
      bindready="handlerReady"
      bindclose="handlerClose"
      binderror="handlerError" />
<button bindtap='login'>登录</button>
```

- **组件参数说明：**

字段名	值类型	默认值	说明
CaptchaAppId	String	无	验证码应用 ID
lang	String	zh-CN	语言，可选 zh-CN、zh-TW、en
themeColor	String	#1A79FF	主题色

- **组件事件说明：**

事件名	参数	说明
ready	无	验证码准备就绪
verify	{ret, ticket}	验证码验证完成
close	{ret}	验证码弹框准备关闭
error	无	验证码配置失败

- **组件方法说明：**

方法名	说明
show	展示验证码
destroy	销毁验证码
refresh	重置验证码

2. 在自定义的 `.js` 文件中，监听事件，代码如下：

```
Page({
    data: {},
    login: function () {
        this.selectComponent('#captcha').show()
        // 进行业务逻辑，若出现错误需重置验证码，执行以下方法
        // if (error) {
        // this.selectComponent('#captcha').refresh()
        // }
    },
    // 验证码验证结果回调
    handlerVerify: function (ev) {
        // 如果使用了 mpvue，ev.detail 需要换成 ev.mp.detail
        if(ev.detail.ret === 0) {
            // 验证成功
            console.log('ticket:', ev.detail.ticket)
        } else {
            // 验证失败
            // 请不要在验证失败中调用refresh，验证码内部会进行相应处理
        }
    },
    // 验证码准备就绪
    handlerReady: function () {
        console.log('验证码准备就绪')
    },
    // 验证码弹框准备关闭
    handlerClose: function (ev) {
        // 如果使用了 mpvue，ev.detail 需要换成 ev.mp.detail,ret为0是验证
完成后自动关闭验证码弹窗，ret为2是用户主动点击了关闭按钮关闭验证码弹窗
        if(ev && ev.detail.ret && ev.detail.ret === 2){
            console.log('点击了关闭按钮，验证码弹框准备关闭');
        } else {
            console.log('验证完成，验证码弹框准备关闭');
        }
    },
    // 验证码出错
    handlerError: function (ev) {
        console.log(ev.detail.errMsg)
    }
})
```

注意

> 业务客户端完成验证码接入后，服务端需二次核查验证码票据结果（未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果），详情请参见 接入票据校验(微信小程序)。

CaptchaAppId 加密校验能力接入指引

通过前端传递加密符（非必选能力，可根据安全性需求选择性接入），可以有效防止因 CaptchaAppId 泄露而造成的资源盗刷。

加密规则

接口通过 aidEncrypted 参数（即加密后的字符串标识），支持采用加密模式传递验证码业务 CaptchaAppId 进行校验。
当控制台强制校验开启时，触发加密模式。由客户的服务端进行加密后下发对应加密字符串到客户的前端，由客户前端将加密后的字符串传参到验证码侧。
加密步骤如下：

1. 登录 验证码控制台，左侧导航栏选择**图形验证** > **验证管理**，进入验证管理页面。
2. 在验证管理页面，选择所需 AppSecretKey，作为密钥 key。当 key 小于32字节时，循环填充同一个 AppSecretKey 补齐到32字节作为密钥 key。

3. 随机生成16字节的 IV，结合步骤1中得到的密钥 Key，对业务数据对象进行 AES256加密，加密模式为 CBC/PKCS7Padding，加密的业务数据对象为验证码业务 CaptchaAppId&时间戳&密文过期时间，时间戳为秒级当前 unix 时间戳，不可为未来时间，密文过期时间单位为秒最大值为86400秒，得到加密后的串为 CaptchaAppIdEncrypted。
4. 对步骤2中 IV 拼接 AES256加密得到的字节数组 CaptchaAppIdEncrypted 进行 Base64编码，即 Base64（IV+CaptchaAppIdEncrypted），中间无连接符，得到最终加密后的业务参数请求字符串即为 aidEncrypted。

加密结果示例

类型	示例值
CaptchaAppId	123456789
时间戳	1710144972
过期时间	86400秒
iv	0123456789012345
AppSecretKey	123456789101112131415161 6
循环填充的密钥 key	1234567891011121314151516123456 7
加密的业务数据对象	123456789&1710144972&86400
加密后的最终字符串 aidEncrypted	MDEyMzQ1Njc4OTAxMjM0NWvZ11atw+1uzYmoIyt5rAQVPyMK9ZDavskPw5hcayeT

代码示例

服务端加密

加密规则为：Base64(IV + AES256(CaptchaAppId&时间戳&密文过期时间，IV， Key))，即使用随机生成16字节的 IV、及根据 AppSecretKey 循环填充后得到32字节的加密 Key，使用 AES256算法加密模式 CBC/PKCS7Padding，对明文数据 **CaptchaAppid&时间戳&密文过期时间**进行加密。

```
 #!/usr/bin/env python
# -*- coding: utf-8 -*-
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import base64
import time


def encrypt(plaintext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv) # 创建一个新的AES cipher，CBC模式
    ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block_size))# 对数据进行填充，然后加密。pad(plaintext.encode(), AES.block_size)将检查明文长度是否为16字节倍数，若非16字节倍数，将使用PKCS7填充方式将明文填充到16字节倍数。
    ciphertextBase64 = base64.b64encode(iv + ciphertext).decode('utf-8') # iv拼接加密后的数据，并进行Base64返回后进行传输。
    return ciphertextBase64
```

```
# 加密示例
AppSecretKey = b'12345678910111213141515l6'    #客户从控制台获取对应验证码账号下的AppSecretKey,25位
remainder = 32 % AppSecretKey.__len__()  # 计算需要补充的密钥长度
key = AppSecretKey + AppSecretKey[:remainder]       # 最终加密key，补充满32位。

Captchaappid = "123456789" # 客户自身的验证码CaptchaAppid
curTime = 1710144972    # 获取当前的时间戳，示例暂设置成固定时间戳，客户应该设置成最新的时间戳，使用int(time.time())
expireTime = 86400 # 过期时间设置，这里暂设置成该值，客户根据自身需要设置

plaintext = Captchaappid + "&" + str(curTime) + "&" + str(expireTime)   # 拼接待加密的业务数据对象，CaptchaAppid&时间戳&密文过期时间

iv = "0123456789012345".encode() # 随机生成16字节的IV，这里暂设置成该值，客户使用时应该用随机生成的数据，使用os.urandom(16)

ciphertext = encrypt(plaintext, key, iv)    # 加密
print("Ciphertext (Base64):", ciphertext)  # 本示例数据将输出
MDEyMzQ1Njc4OTAxMjM0NWvZ11atw+1uzYmoIyt5rAQVPyMK9ZDavskPw5hcayeT
```

前端接入示例

1. 确保插件版本为2.1.0。

```
{
    "plugins": {
        "captcha": {
            "version": "2.1.0",
            "provider": "wx1fe8d9a3cb067a75"
        }
    }
}
```

2. 组件添加 aidEncrypted 参数。

```
<t-captcha
```

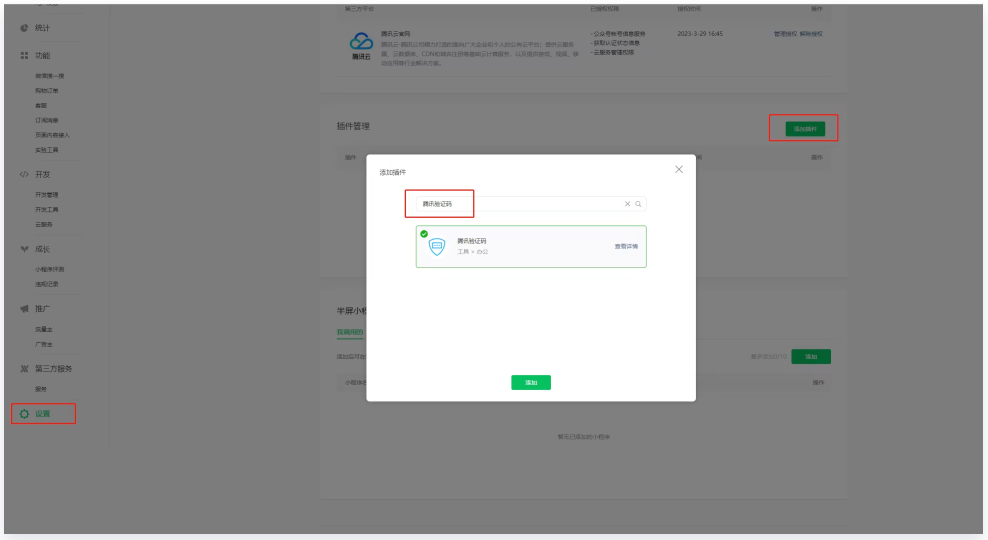
```
  id="captcha"
  app-id="{{CaptchaAppid}}"
  aid-encrypted="{{aidEncrypted}}"
  bindverify="handlerVerify"
  bindready="handlerReady"
  bindclose="handlerClose"
  binderror="handlerError"
/>
<button bindtap='login'>登录</button>


Page({
  data: {
    appId: "1234567",
    themeColor: "#1A79FF",
    aidEncrypted: ""
  },
  login() {
    this.selectComponent("#captcha").show()
  },
  onLoad() {
    /** 加载时获取加密的appid字符串 */
    this.getAidEncrypted()
  },
  getAidEncrypted() {
    wx.request({
      method: "post",
      url: "/api/aidEncrypted",
      data: {},
      success: (res) => {
        this.aidEncrypted = res.data.aidEncrypted
      }
    })
  }
})
```

uni-app 前端框架接入

步骤1：添加插件

1. 用管理员身份登录 微信公众平台 ，且需使用接入小程序的相关账号。

2. 选择**设置** > **第三方设置** > **添加插件**，在搜索框内输入关键字“腾讯验证码”查找插件，并单击添加，如下图所示。



步骤2：集成插件

1. 引入验证码小程序插件。
   使用验证码插件前，需要在 `app.json` 中声明验证码小程序插件，如下：

```
{
    "plugins": {
        "captcha": {
            "version": "2.1.0", //请选择小程序插件最新版本
            "provider": "wx1fe8d9a3cb067a75"
        }
    }
}
```

2. 引入验证码小程序组件。
   在页面 `.json` 文件中需要引入自定义组件，js 代码如下：

```
{
    "usingComponents": {
      "t-captcha": "plugin://captcha/t-captcha"
    }
}
```

步骤3：使用小程序插件

1. 使用 uni-app 框架接入时，需要在自定义的 `.vue` 中使用验证码插件，代码如下：

```
<!-- app-id：验证码CaptchaAppId，从腾讯云的验证码控制台中获取，在验证码控制台页面内【图形验证】>【验证列表】进行查看 -->
<t-captcha
      id="captcha"
      app-id="小程序插件验证码CaptchaAppId"
      @verify="handlerVerify"
      @ready="handlerReady"
      @close="handlerClose"
      @error="handlerError" />
<button @click="login">登录</button>
```

- **组件参数说明：**

字段名	值类型	默认值	说明
CaptchaAppId	String	无	验证码应用 ID
lang	String	zh-CN	语言，可选 zh-CN、zh-TW、en
themeColor	String	#1A79FF	主题色

- **组件事件说明：**

事件名	参数	说明
ready	无	验证码准备就绪
verify	{ret, ticket}	验证码验证完成
close	{ret}	验证码弹框准备关闭
error	无	验证码配置失败

- **组件方法说明：**

方法名	说明
show	展示验证码
destroy	销毁验证码
refresh	重置验证码

2. 在自定义的 `.vue` 文件中，监听事件，Vue2代码如下：

```
methods:{
      login: function () {
          this.selectComponent('#captcha').show()
          // 进行业务逻辑，若出现错误需重置验证码，执行以下方法
          // if (error) {
          // this.selectComponent('#captcha').refresh()
          // }
      },
      // 验证码验证结果回调
      handlerVerify: function (ev) {
          // 如果使用了 mpvue，ev.detail 需要换成 ev.mp.detail
          if(ev.detail.ret === 0) {
              // 验证成功
              console.log('ticket:', ev.detail.ticket)
          } else {
              // 验证失败
              // 请不要在验证失败中调用refresh，验证码内部会进行相应处理
          }
      },
      // 验证码准备就绪
      handlerReady: function () {
          console.log('验证码准备就绪')
      },
      // 验证码弹框准备关闭
      handlerClose: function (ev) {
          // 如果使用了 mpvue，ev.detail 需要换成 ev.mp.detail,ret为0是验证
完成后自动关闭验证码弹窗，ret为2是用户主动点击了关闭按钮关闭验证码弹窗
          if(ev && ev.detail.ret && ev.detail.ret === 2){
              console.log('点击了关闭按钮，验证码弹框准备关闭');
          } else {
              console.log('验证完成，验证码弹框准备关闭');
          }
      },
      // 验证码出错
      handlerError: function (ev) {
          console.log(ev.detail.errMsg)
      }
}
```

Vue3代码如下：

```
<script lang="ts" setup>
import { getCurrentInstance } from 'vue'
// 获取组件实例
const instance: any = getCurrentInstance()
const login = () => {
  instance.ctx.selectComponent('#captcha').show()
  // 进行业务逻辑，若出现错误需重置验证码，执行以下方法
  // if (error) {
  // instance.ctx.selectComponent('#captcha').refresh()
  // }
}
// 验证码验证结果回调
const handlerVerify = (ev) => {
  // 如果使用了 mpvue，ev.detail 需要换成 ev.mp.detail
  if (ev.detail.ret === 0) {
    // 验证成功
    console.log('ticket:', ev.detail.ticket)
  } else {
    // 验证失败
    // 请不要在验证失败中调用refresh，验证码内部会进行相应处理
  }
}
// 验证码准备就绪
const handlerReady = () => {
  console.log('验证码准备就绪')
}
// 验证码弹框准备关闭
const handlerClose = (ev) => {
  // 如果使用了 mpvue，ev.detail 需要换成 ev.mp.detail,ret为0是验证完成后自
动关闭验证码弹窗，ret为2是用户主动点击了关闭按钮关闭验证码弹窗
  if (ev && ev.detail.ret && ev.detail.ret === 2) {
    console.log('点击了关闭按钮，验证码弹框准备关闭')
  } else {
    console.log('验证完成，验证码弹框准备关闭')
  }
}
// 验证码出错
const handlerError = (ev) => {
  console.log(ev.detail.errMsg)
}
</script>
```

Taro 框架小程序插件接入示例

1. 在 app.config.ts 引入小程序插件。

```
{
  "plugins": {
    "captcha": {
      "version": "2.1.0",
      "provider": "wx1fe8d9a3cb067a75"
    }
  }
}
```

2. 在需要加载验证码的页面配置插件，如 page/index/index.config.ts。

```
{
  "usingComponents": {
    "t-captcha": "plugin://captcha/t-captcha"
  }
}
```

3. 在页面调用验证码，如 page/index/index.tsx。

```
import { getCurrentInstance, PageInstance } from '@tarojs/taro';

export default function Index() {
  // 获取页面实例

  const { page } = getCurrentInstance();

  // 弹出验证码

  const handlerCaptchaShow = () => {
    const pageInstance = page as PageInstance;

    const captcha: any = pageInstance.selectComponent &&
pageInstance?.selectComponent('#captcha');
```

```
    try {
      captcha?.show();
    } catch (error) {
      // 进行业务逻辑，若出现错误需重置验证码，执行以下方法

      captcha?.refresh();
    }
  };

  // 验证码验证结果回调

  const handlerVerify = (ev) => {
    console.log('ret:', ev.detail);

    // 如果使用了 mpvue，ev.detail 需要换成 ev.mp.detail

    if (ev.detail.ret === 0) {
      // 验证成功

      console.log('ticket:', ev.detail.ticket);
    } else {
      // 验证失败
      // 请不要在验证失败中调用refresh，验证码内部会进行相应处理
    }
  };

  return (
    <View className='index'>
      <t-captcha id='captcha' appId='小程序插件验证码CaptchaAppId'
onVerify={handlerVerify} />
      <Button onClick={handlerCaptchaShow}>弹出验证码</Button>
    </View>
  );
}
```

注意

业务客户端完成验证码接入后，服务端需二次核查验证码票据结果（未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果），详情请参见 接入票据校验(微信小程序)。

常见问题

详情参见 接入相关问题 。

更多信息

您可以登录 验证码控制台 ，在页面右下角单击**快速咨询**，了解更多详细信息。

Demo 下载及体验

最近更新时间：2025-10-24 15:20:22

为了让您更轻松地接入和使用验证码，我们提供了一些适用于常见开发环境的示例代码供您下载使用。

平台	Demo下载链接
Web/H5	• React 框架 • React-Native 框架 • Flutter 框架
iOS	• swift 开发语言 • Objective-C 开发语言
Android	Demo for Android
小程序	• 小程序原生语言 • Taro 跨端小程序 • Uniapp-Vue3
鸿蒙	• ArkTS9+for Harmony • ArkTS12+for Harmony Next

服务端接入
接入票据校验（Web 及 App）

最近更新时间：2024-12-02 11:15:32

服务端需调用票据校验 API，对客户端验证结果进行二次校验。

> **注意：**
> 未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果。

接入步骤

步骤1：安装对应语言 SDK

腾讯云提供多语言 SDK，您可以基于业务需求，安装对应语言的 SDK，详情请参见：SDK中心 。

步骤2：获取云 API 密钥

1. 登录 访问管理控制台 ，左侧导航栏选择**访问密钥** > **API 密钥管理**。
2. 在 API 密钥管理页面，如已创建密钥，在该页面查看即可；如未创建密钥，单击**新建密钥**，即可生成所需密钥。

> **注意：**
> 建议使用子账号密钥，并遵循最小权限控制的原则（仅赋予子账号“QcloudCaptchaFullAccess”权限），降低使用风险。子账号密钥获取可参见 子账号访问密钥管理 。

步骤3：调用 DescribeCaptchaResult 接口

建议使用 API Explorer，在线生成代码。API 详情参见 核查验证码票据结果（Web 及 APP） 。

> **注意：**
> 不同客户端需调用对应客户端的票据校验 API，不可混用，否则会导致接口报错。

常见问题

详情请参见 接入相关问题 。

更多信息

您可以登录 验证码控制台 ，在页面右下角单击**快速咨询**，了解更多详细信息。

接入票据校验（微信小程序）

最近更新时间：2024-12-02 11:15:32

服务端需调用票据校验 API，对客户端验证结果进行二次校验。

> **注意：**
> 未接入票据校验，会导致黑产轻易伪造验证结果，失去验证码人机对抗效果。

接入步骤

步骤1：安装对应语言 SDK

腾讯云提供多语言 SDK，您可以基于业务需求，安装对应语言的 SDK，详情请参见：SDK中心 。

步骤2：获取云 API 密钥

1. 登录 访问管理控制台 ，左侧导航栏选择**访问密钥** > **API 密钥管理**。
2. 在 API 密钥管理页面，如已创建密钥，在该页面查看即可；如未创建密钥，单击**新建密钥**，即可生成所需密钥。

> **注意：**
> 建议使用子账号密钥，并遵循最小权限控制的原则（仅赋予子账号“QcloudCaptchaFullAccess”权限），降低使用风险。子账号密钥获取可参见 子账号访问密钥管理 。

步骤3：调用 DescribeCaptchaMiniResult 接口

建议使用 API Explorer ，在线生成代码。API 详情参见：核查验证码票据结果(小程序插件) 。

> **注意：**
> 不同客户端需调用对应客户端的票据校验 API，不可混用，否则会导致接口报错。

常见问题

详情参见 接入相关问题 。

更多信息

您可以登录 验证码控制台 ，在页面右下角单击**快速咨询**，了解更多详细信息。