# Captcha

# Integration Guide

# Product Documentation

# Contents

# Integration Guide
# Client Integration
# Web Integration

Last updated：2024-01-02 15:46:48

## Prerequisites

Before you can integrate TenDI Captcha into a client, you need to create a CAPTCHA and obtain its CaptchaAppId and AppSecretKey from the **CAPTCHA list**. Follow the steps below:

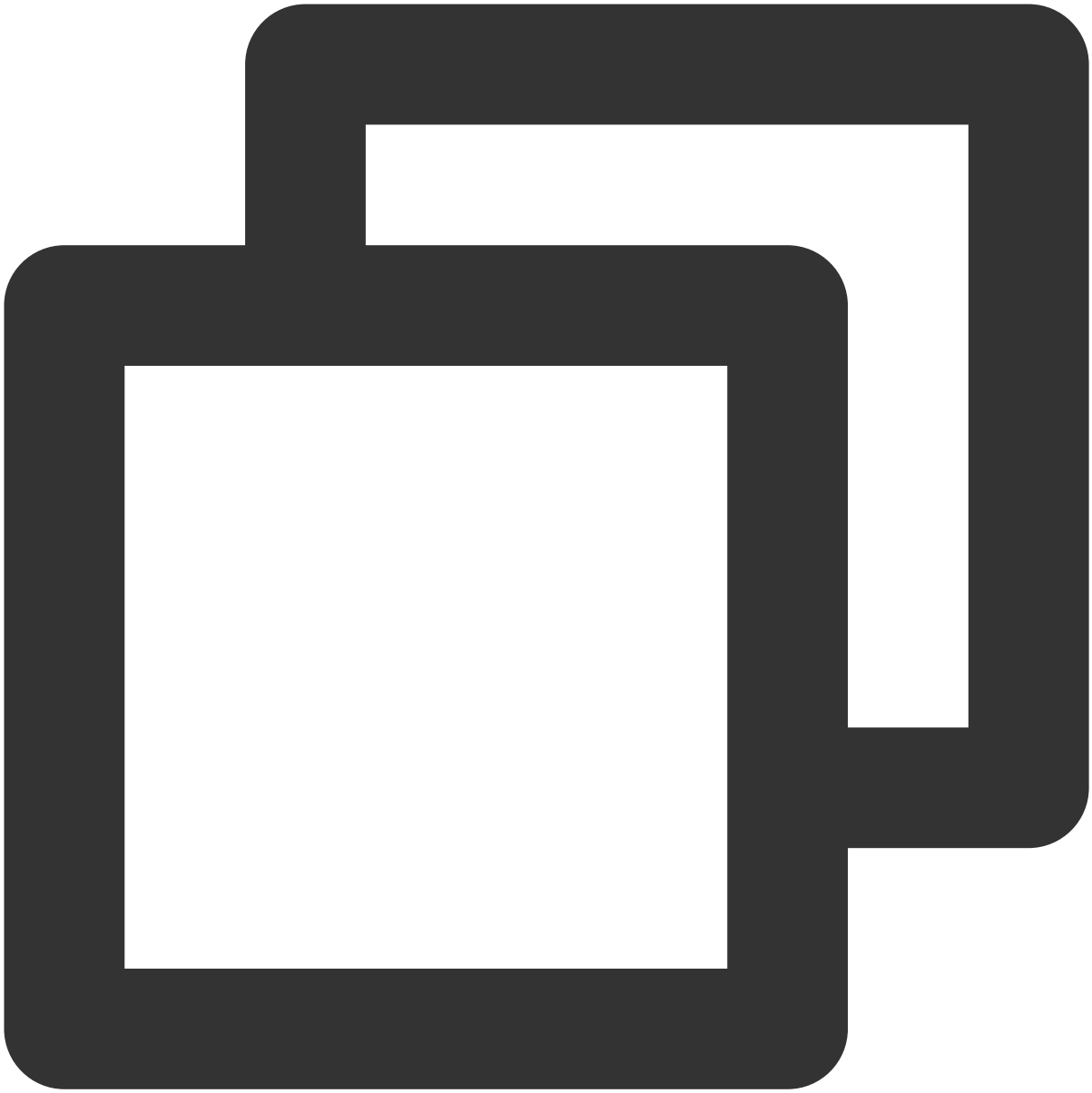
1. Log in to the Captcha console and select **CAPTCHA** -> **Manage CAPTCHA** in the left navigation pane to enter the CAPTCHA management page.
2. Click **Create CAPTCHA** and set parameters such as CAPTCHA name according to your business requirements.
3. Click **OK** to complete the creation. You can view its CaptchaAppId and AppSecretKey in the CAPTCHA list.

## Sample code

The following sample code demonstrates a page where **Verify** is clicked to activate the CAPTCHA, and the verification result is displayed in a pop-up window.

**Note**

This sample does not include the logic to call the ticket verification API. After TenDI Captcha is integrated into the business client, the business server needs to verify the CAPTCHA ticket (if ticket verification is not integrated, the black market can easily forge verification results, which defeats the purpose of human verification via CAPTCHAs). For more information, please see Integration to Ticket Verification (Web and 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 frontend access code sample</title>
  <!-- (Required) Dependency of the CAPTCHA program, which cannot be modified. If
  <script src="https://ca.turing.captcha.qcloud.com/TCaptcha-global.js">></script
```

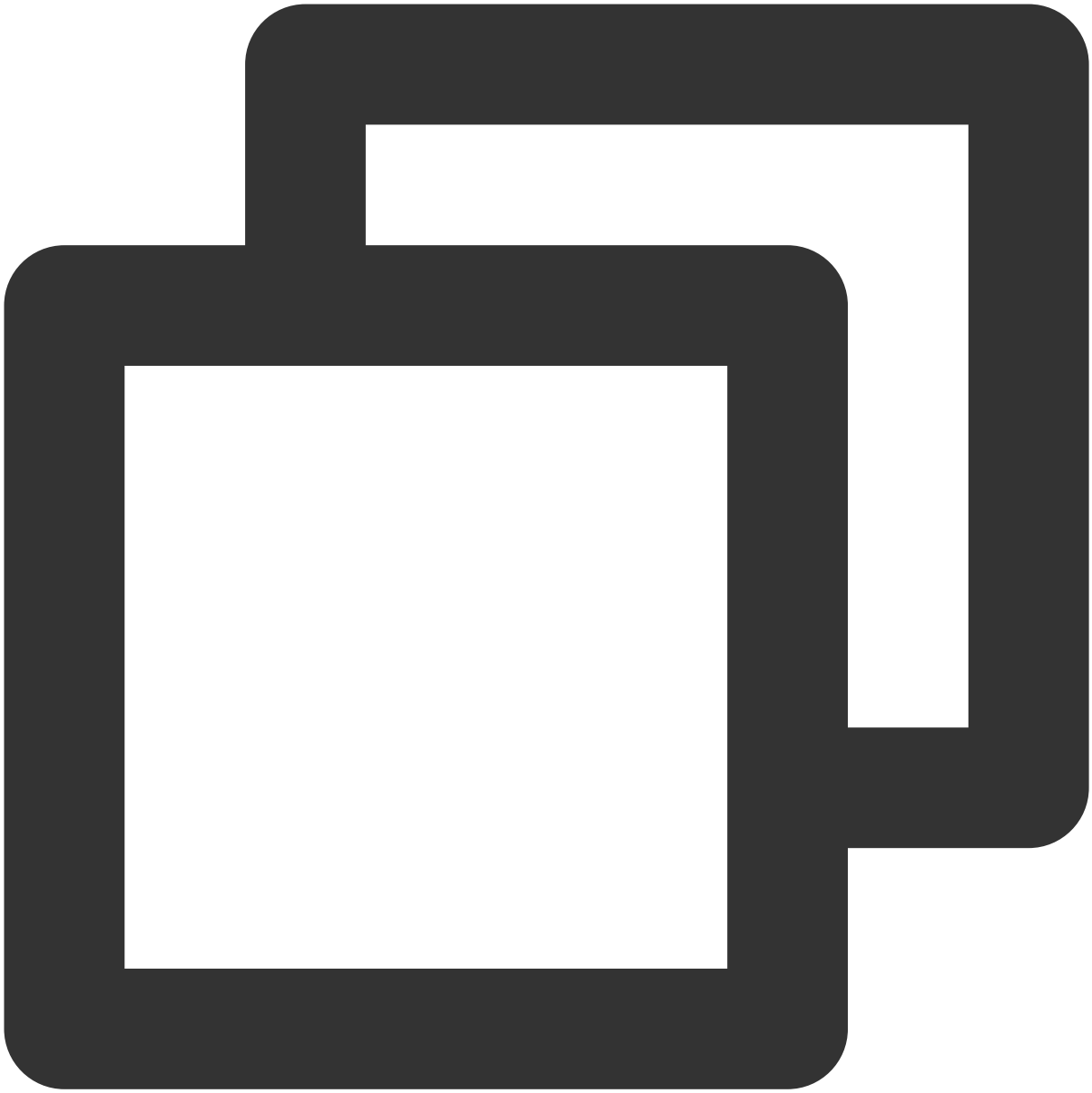
```
</head>

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

<script>

    // Define the callback function
    function callback(res) {
        // The callback output returned by the first passed parameter, which is a
        // ret         Int       Verification result. The value 0 indicates a suc
        // ticket      String    A verified ticket. The field value is not null o
        // CaptchaAppId       String    ID of the CAPTCHA.
        // bizState    Any       The defined transparent transmission parameter.
        // randstr     String    The random string verified, which is required fo
        console.log('callback:', res);


        // res (The user disables the CAPTCHA) = {ret: 2, ticket: null}
        // res (Verified)  = {ret: 0, ticket: "String", randstr: "String"}
        // res (Return a disaster recovery ticket prefixed with "error_" when the
        // Here is the code snippet of the verification result. Modify it based o
        if (res.ret === 0) {
          // Copy result to clipboard
          var str = '【randstr】-> 【' + res.randstr + '】        【ticket】-> 【' + res
          var ipt = document.createElement('input');
          ipt.value = str;
          document.body.appendChild(ipt);
          ipt.select();
          document.execCommand("Copy");
          document.body.removeChild(ipt);
          alert('1. Return result (randstr、ticket) Copied successfully. You can p
        }
    }

  // Define the function that handles TCaptcha-global.js loading errors
  function loadErrorCallback() {
    var appid = 'CaptchaAppId';
    // Generate a disaster recovery ticket or execute other operations
    var ticket = 'terror_1001_' + appid + '_' + Math.floor(new Date().getTime() /
    callback({
      ret: 0,
      randstr: '@'+ Math.random().toString(36).substr(2),
      ticket: ticket,
      errorCode: 1001,
      errorMessage: 'jsload_error',
```

```
      });
    }

      // Define the event that triggers CAPTCHA
      window.onload = function(){
        document.getElementById('CaptchaId').onclick = function(){
          try {
            // Generate a CAPTCHA object
            // CaptchaAppId：Log in to the Captcha console and enter the [Manage CA
            //callback：Defined callback function
            var captcha = new TencentCaptcha('CaptchaAppId', callback, {});
            // Call the method to display the CAPTCHA
            captcha.show();
          } catch (error) {
            // Load error. Please call the  load error handling function
            loadErrorCallback();
            }
          }
      }
  </script>

  </html>
```

# Integration steps

### Step 1: Dynamically introduce the CAPTCHA JS

You need to dynamically introduce the CAPTCHA JS into webpages to invoke CAPTCHAs when verification is needed.



```
<!-- Example of dynamically introducing the CAPTCHA JS -->
<script src="https://ca.turing.captcha.qcloud.com/TCaptcha-global.js"></script>
```

**Note**

You must dynamically introduce the CAPTCHA JS. If you use other methods to avoid dynamic loading, CAPTCHAs cannot be updated and consequently some legitimate requests rather than malicious requests might be blocked.

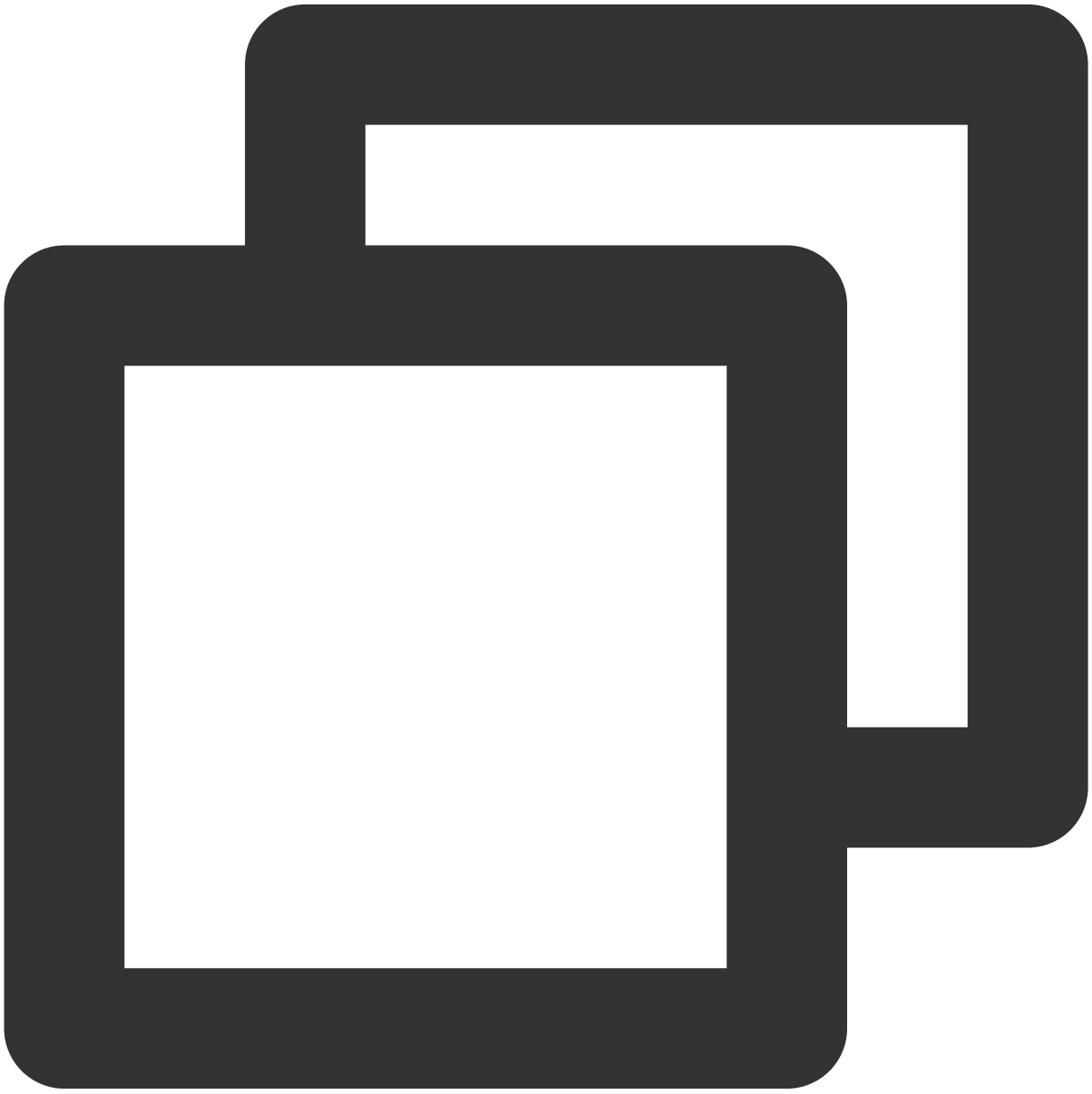
### Step 2: Create a CAPTCHA object

After the CAPTCHA JS is introduced, Captcha will globally register the class `TencentCaptcha`, which can be used to initialize CAPTCHAs, and show or hide CAPTCHAs.

**Note**

Do not use `id="TencentCaptcha"` to trigger the CAPTCHA. TencentCaptcha is a default system id and cannot be used.

**Constructor function**



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

**Parameters**

| Parameter | Value type | Description |
| --- | --- | --- |
| CaptchaAppId | String | The CAPTCHA's CaptchaAppId: Log in to the Captcha console to view it on the **Manage CAPTCHA** page. If no CAPTCHAs exist, create one first. |
| callback | Function | The CAPTCHA callback function. For more information, please see callback function. |
| options | Object | The CAPTCHA appearance configuration parameter. For more information, please see options appearance configuration parameter. |

**callback function**

After verification is finished, the callback function passed by the business is called, and the callback result is passed in the first parameter. The callback fields are as follows:

| Field | Value type | Description |
| --- | --- | --- |
| ret | Int | Verification result. The possible values are 0 (Verification is successful) and 2 (CAPTCHA is closed by the user). |
| ticket | String | The ticket of a successful verification. ticket has a value only when ret is 0. |
| CaptchaAppId | String | The CAPTCHA's app ID. |
| bizState | Any | The custom pass-through parameter. |
| randstr | String | The random string for this verification. This parameter is required for ticket verification. |
| errorCode | Number | Error code. For more information, please see Callback function errorCode values. |
| errorMessage | String | Error message. |

**Callback function errorCode values**

| errorCode | Description |
| --- | --- |
| 1001 | TCaptcha-global.js load error |
| 1002 | Timeout when calling the show method |

| | |
|---|---|
| 1003 | frame.js load timeout |
| 1004 | frame.js load error |
| 1005 | frame.js run error |
| 1006 | Error/Timeout when pulling the CAPTCHA configuration |
| 1007 | iframe load timeout |
| 1008 | iframe load error |
| 1009 | jquery load error |
| 1010 | dy-ele.js load error |
| 1011 | dy-ele.js run error |
| 1012 | 3 consecutive failed attempts to pull a CAPTCHA after refreshing |
| 1013 | 3 consecutive failed submissions for verification |

**options appearance configuration parameter**

The options parameter sets the CAPTCHA appearance. It can be left empty by default.

**Note**

You cannot change the style and size within the CAPTCHA pop-up window. Instead, you can apply `transform:scale();` to the element with `class=tcaptcha-transform` at the outermost layer of the pop-up window. CAPTCHA updates may change element attributes, such as id and class. Do not use other attribute values of the CAPTCHA element to override the style.

Any horizontal swipe gesture set in your mobile app must be disabled before you call the CAPTCHA show method to prevent conflicts with CAPTCHA sliding events. The gesture can be enabled after verification is finished.

| Parameter | Value type | Description |
|---|---|---|
| bizState | Any | The custom pass-through parameter, which can be used to pass a small amount of data. The parameter value will be included in the callback object. |
| enableDarkMode | Boolean/String | Enables the adaptive dark mode or force dark mode.<br>Adaptive dark mode: {"enableDarkMode": true}<br>Force dark mode: {"enableDarkMode": 'force'} |
| ready | Function | The CAPTCHA loading completion callback. The callback parameters are the actual CAPTCHA width and height:<br>{"sdkView": {<br>"width": number, |

| | | |
|---|---|---|
| | | "height": number<br>}}<br>This parameter only displays the CAPTCHA width and height, and **cannot be used to set the width and height directly**. |
| needFeedBack | String | Custom help link: {"needFeedBack": 'url' } |
| loading | Boolean | Indicates whether to display a loading overlay while the CAPTCHA is loading. If the parameter is not specified, a loading overlay is displayed by default.<br>Display a loading overlay: {"loading": true}<br>Don't display a loading overlay: {"loading": false} |
| userLanguage | String | Specifies the language of CAPTCHA prompts. This configuration takes priority over the language of the console.<br>You can pass the same value as the user's preferred language navigator.language. The value is non-case-sensitive.<br>For more information, please see userLanguage configuration parameters. |

**userLanguage configuration parameters**

| Parameter | Description |
|---|---|
| zh-cn | Simplified Chinese |
| zh-hk | Traditional Chinese (Hong Kong, China) |
| zh-tw | Traditional Chinese (Taiwan, China) |
| en | English |
| ar | Arabic |
| my | Burmese |
| fr | French |
| de | German |
| he | Hebrew |
| hi | Hindi |
| id | Indonesian |
| it | Italian |

| | |
|---|---|
| ja | Japanese |
| ko | Korean |
| lo | Lao |
| ms | Malay |
| pl | Polish |
| pt | Portuguese |
| ru | Russian |
| es | Spanish |
| th | Thai |
| tr | Turkish |
| vi | Vietnamese |

### Step 3: Call CAPTCHA instance methods

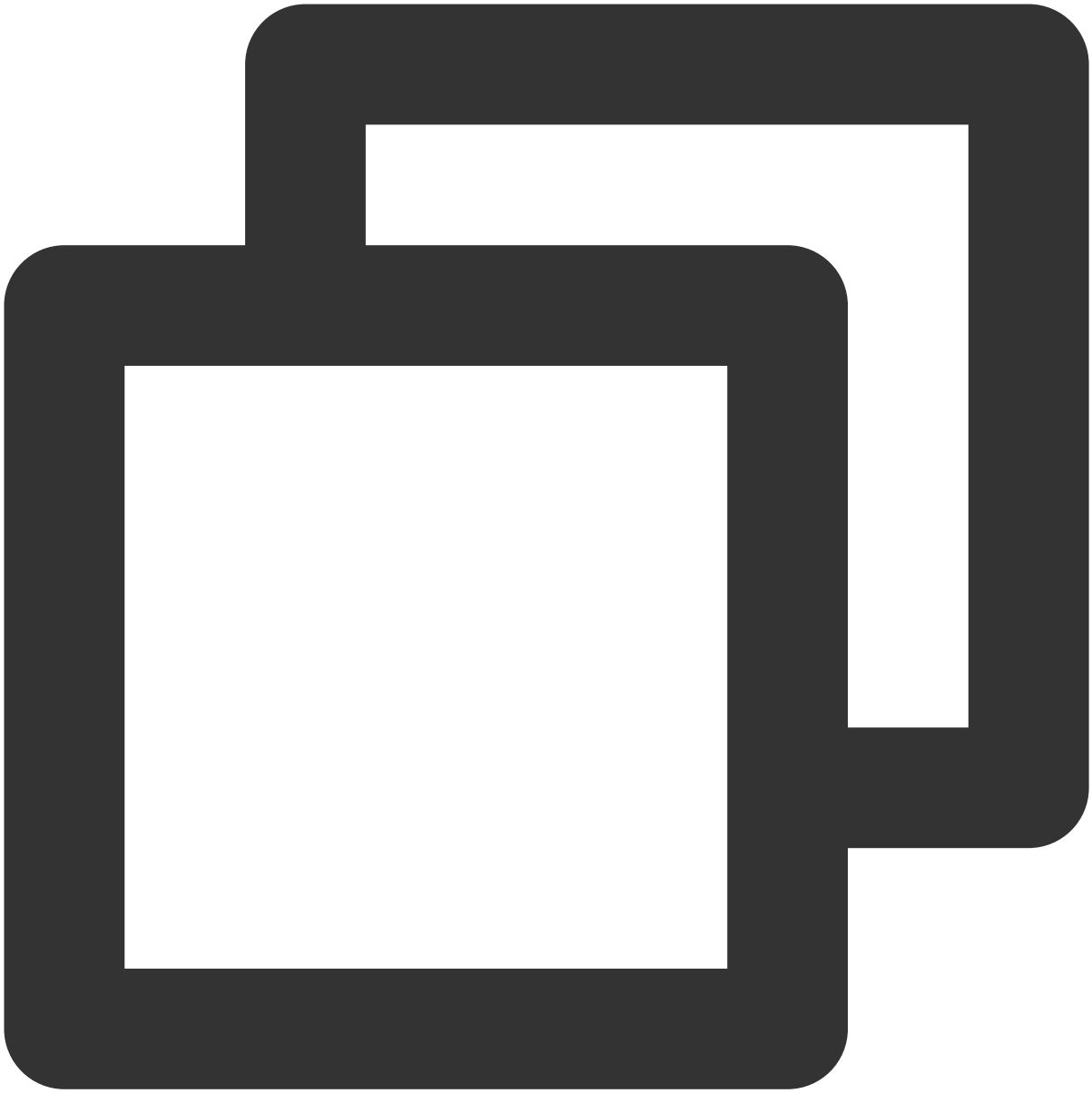
The TencentCaptcha instance provides some common methods for CAPTCHA operations:

| Method | Description | Input parameter | Response content |
|---|---|---|---|
| show | Shows the CAPTCHA. The method can be called multiple times. | None | None |
| destroy | Hides the CAPTCHA. The method can be called multiple times. | None | None |
| getTicket | Obtains the ticket of a successful verification. | None | `Object: {"CaptchaAppId":"","ticket":""}` |

### Step 4: Handle disaster recovery

To ensure that customers' websites run normally in case of an exception in the Captcha server, we recommend the following integration steps.

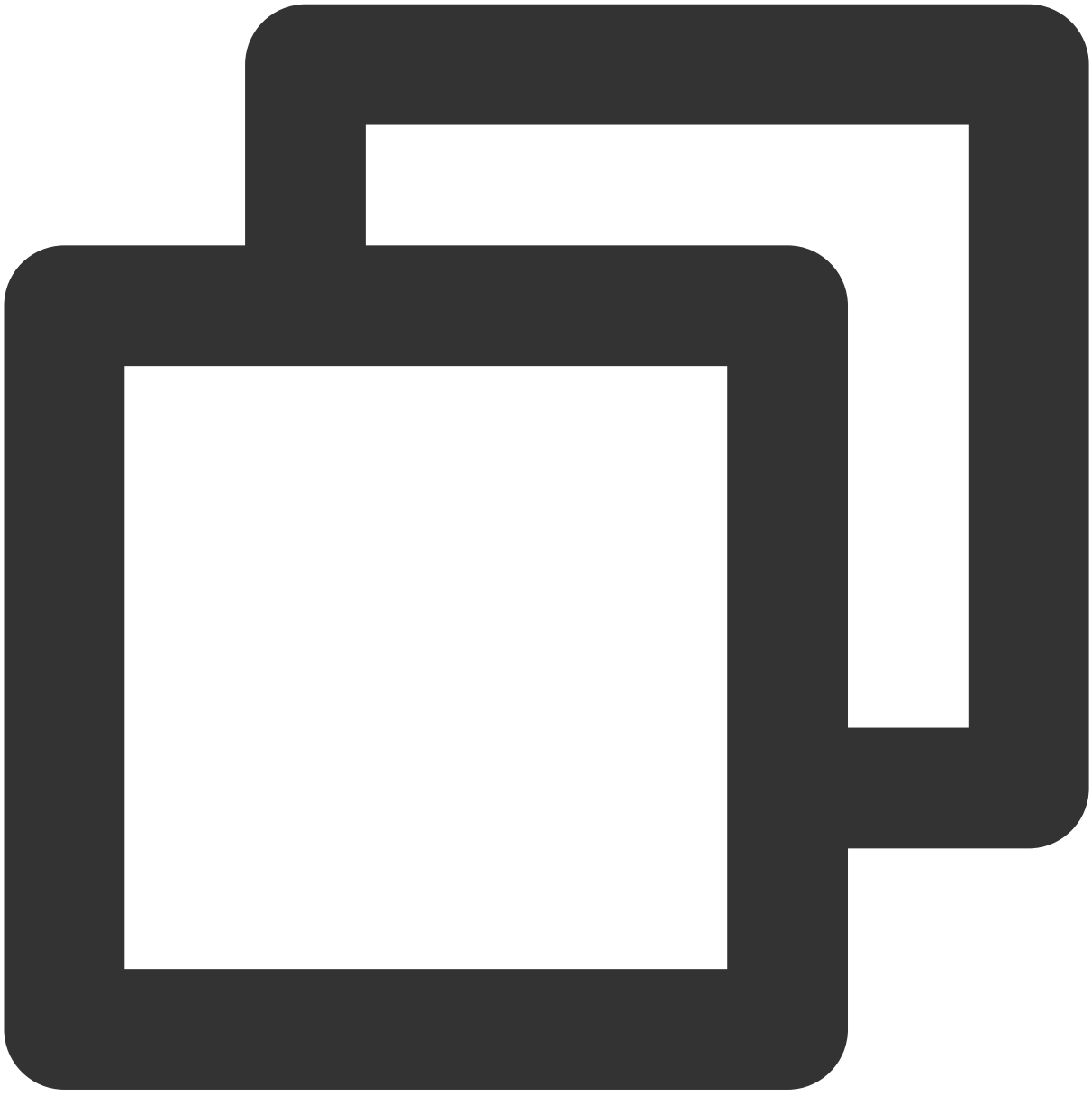
1. Define the JS load error handling function.



```
// The error handling function ensures that event processes run normally in case of
// Define the function before the script loads
function loadErrorCallback() {
 var appid = ''
 // Generate a disaster recovery ticket or use another handling technique
 var ticket = 'terror_1001_' + appid + Math.floor(new Date().getTime() / 1000);
 callback({
     ret: 0,
     randstr: '@'+ Math.random().toString(36).substr(2),
     ticket,
     errorCode: 1001,
```

```
      errorMessage: 'jsload_error',
  });
}
```

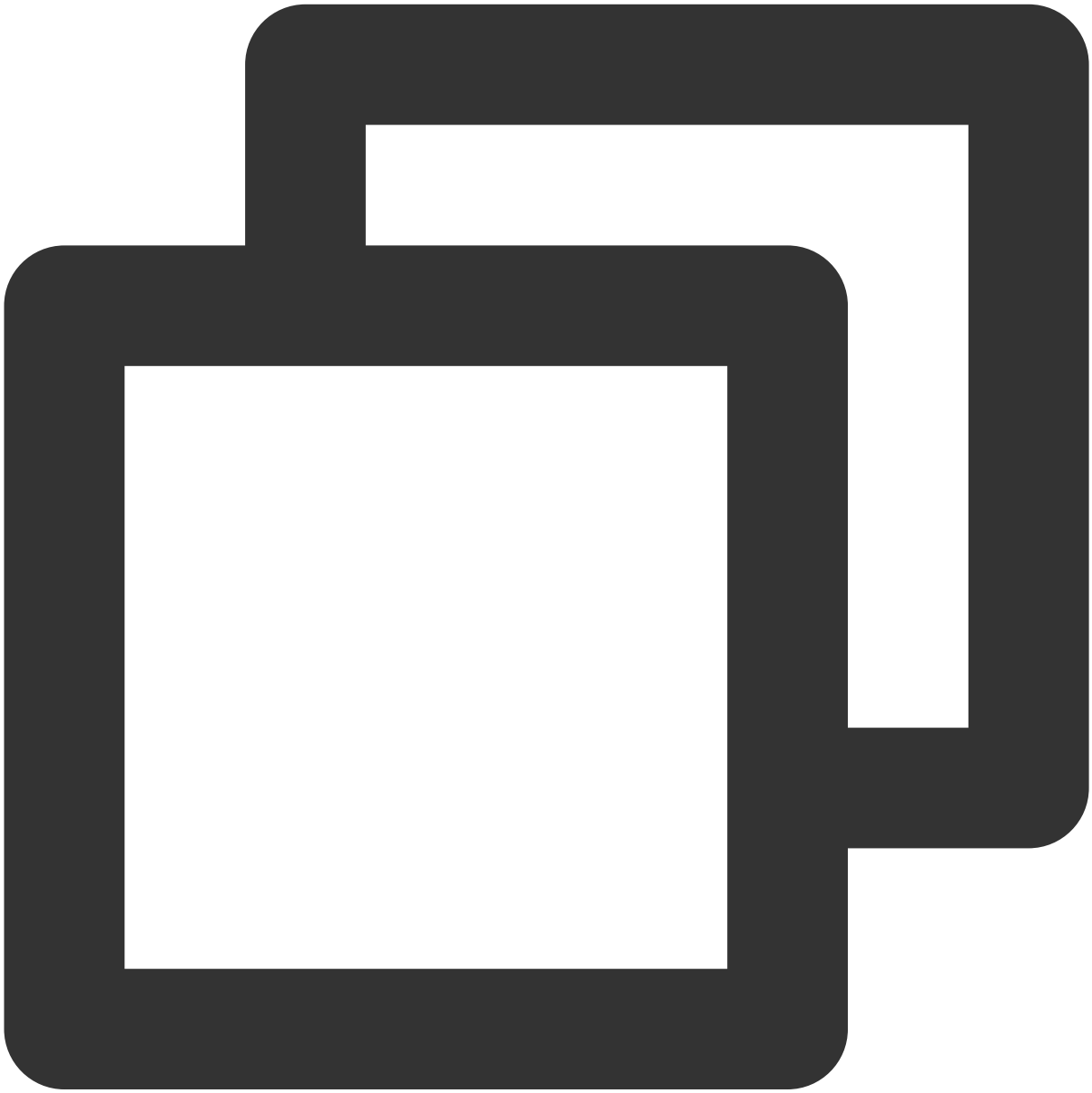
2. Call the JS load error handling function if an error is caught during the CAPTCHA instance call.



```
try {
  // Generate a CAPTCHA object
  var captcha = new TencentCaptcha('Your CAPTCHA's CaptchaAppId', callback, {});
  // Call the method to show the CAPTCHA
  captcha.show()
} catch (error) {
```

```
  // Load error. Call the CAPTCHA js load error handling function.
  loadErrorCallback();
}
```

3. Define the CAPTCHA callback function so the handling is based on ticket and errorCode (instead of ret). For errorCode definitions, please see "callback function errorCode values" in Step 2.



```
function callback(res) {
  // res (CAPTCHA is closed by the user) = {ret: 2, ticket: null}
  // res (Verification is successful) = {ret: 0, ticket: "String", randstr: "String"
  // res (Request error. A disaster recovery ticket with the prefix terror_ is retur
```

```
  if (res.ticket){
      // Handle based on errorCode
      if(res.errorCode === xxxxx){
         // Customize the disaster recovery logic (for example, skip this verificati
      }
    }
}
```

For more information on disaster recovery schemes, please see Business Disaster Recovery Scheme.

**Note**

After TenDI Captcha is integrated into the business client, the business server needs to verify the CAPTCHA ticket (if ticket verification is not integrated, the black market can easily forge verification results, which defeats the purpose of human verification via CAPTCHAs). For more information, please see Integration to Ticket Verification (Web and App).

# FAQs

For more information, please see FAQs About Integration.

# App Integration

Last updated：2024-01-02 15:46:48

## Prerequisites

Before you can integrate TenDI Captcha into a client, you need to create a CAPTCHA and obtain its CaptchaAppId and AppSecretKey from the **CAPTCHA list**. Follow the steps below:
1. Log in to the Captcha console and select **CAPTCHA** -> **Manage CAPTCHA** in the left navigation pane to enter the CAPTCHA management page.
2. Click **Create CAPTCHA** and set parameters such as CAPTCHA name according to your business requirements.
3. Click **OK** to complete the creation. You can view its CaptchaAppId and AppSecretKey in the CAPTCHA list.

## Integration steps

**Note:**
You can integrate TenDI Captcha into an (Android/iOS) app only by using a WebView to display H5 pages.

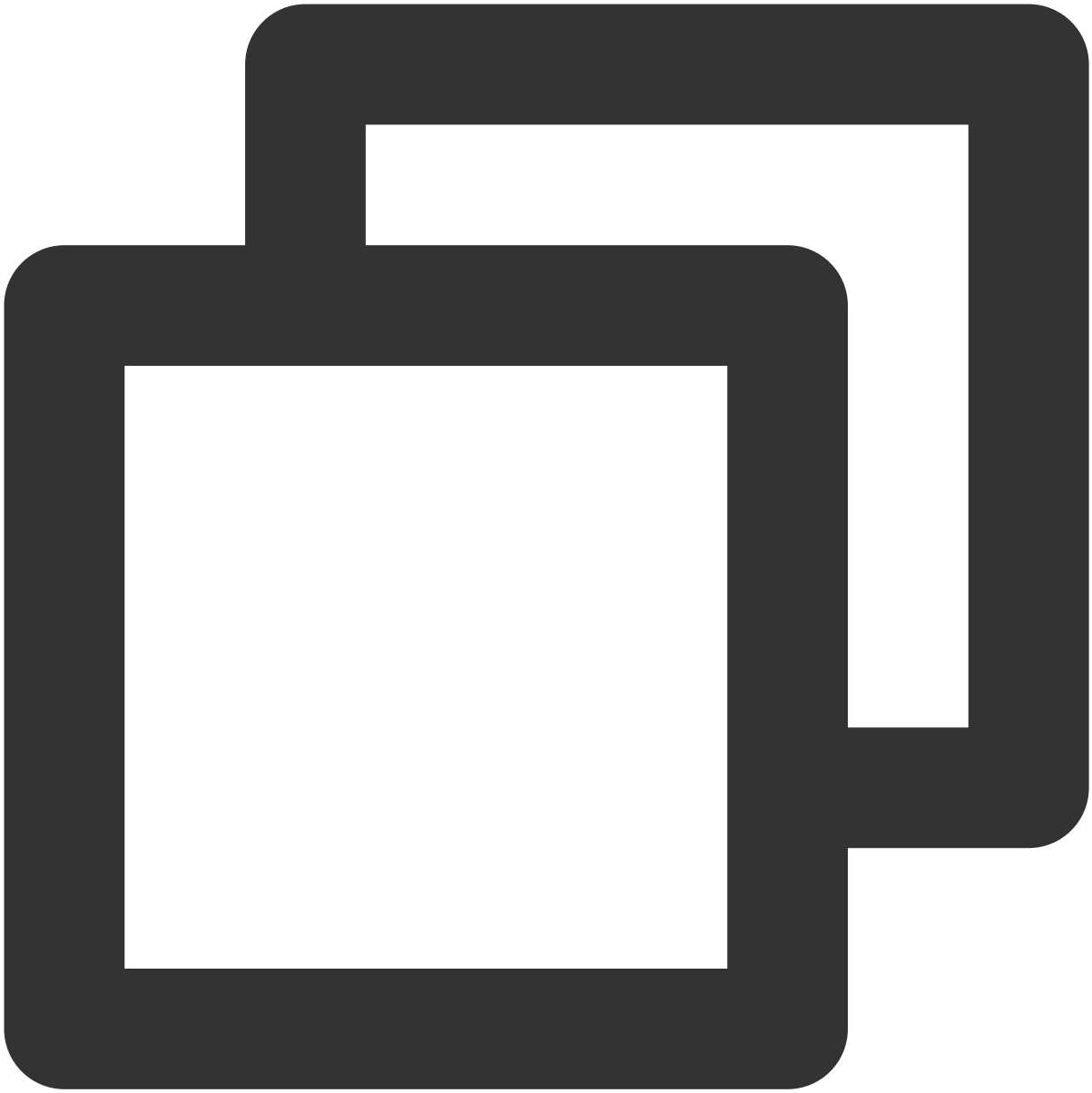
### Integration in Android

#### General process

1. Use a WebView to display H5 pages in an Android app. For more information on how to integrate TenDI Captcha into H5 pages, please see Web Integration.
2. Call the CAPTCHA JS to render the verification page in an H5 page. Then pass the parameter values returned in JS to the Android app's business client.
3. The Android app's business client passes parameters such as ticket and random number to the business server for ticket verification. For more information, please see Integration to Ticket Verification (Web and App).

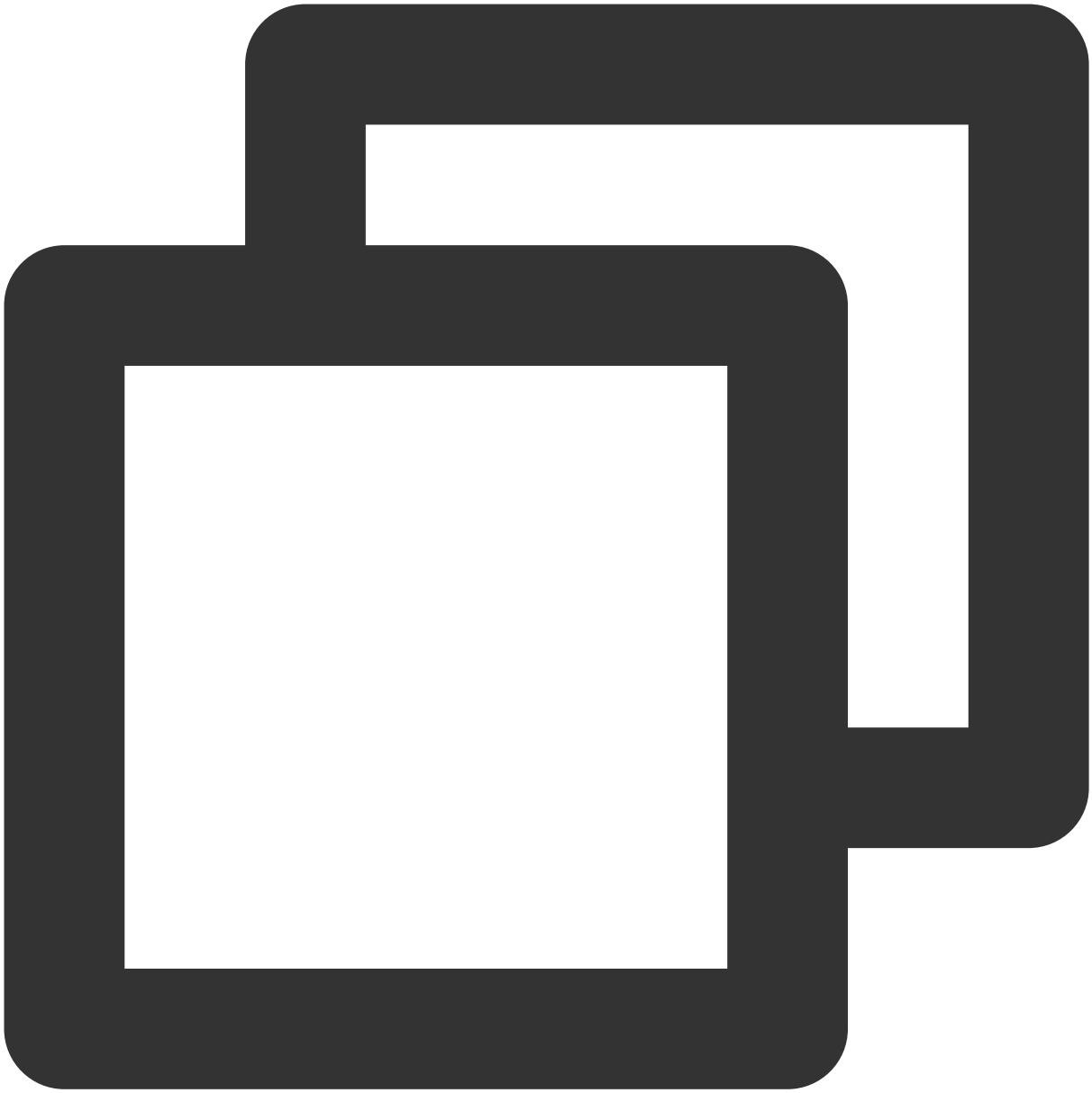
#### Detailed steps

1. In a project, create an activity and import the WebView packages required.



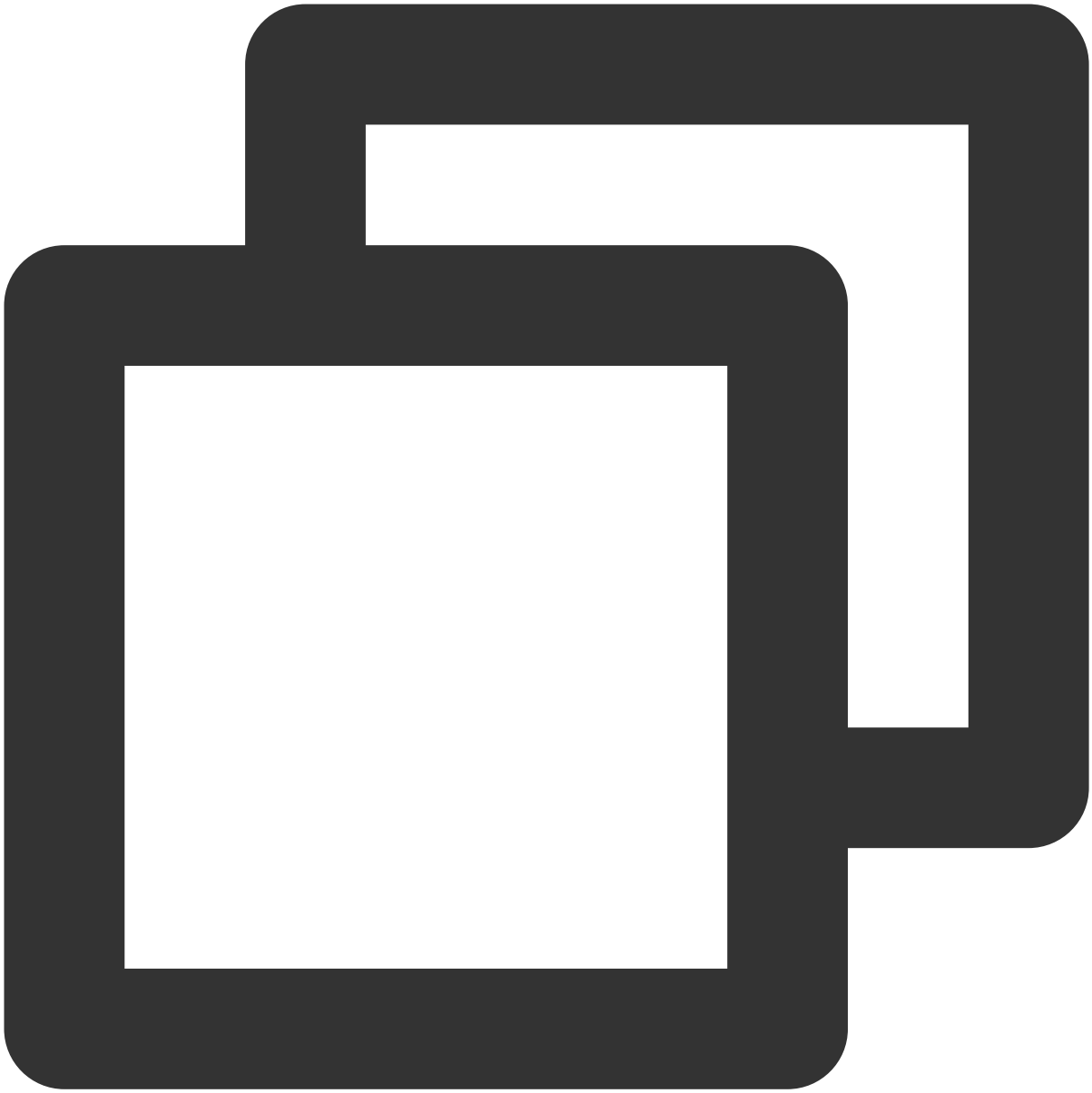
```
import android.webkit.WebView;
import android.webkit.WebSettings;
import android.webkit.WebViewClient;
import android.webkit.WebChromeClient;
```

2. Add permissions, such as "Enable network access" and "Allow the app to make non-HTTPS requests".



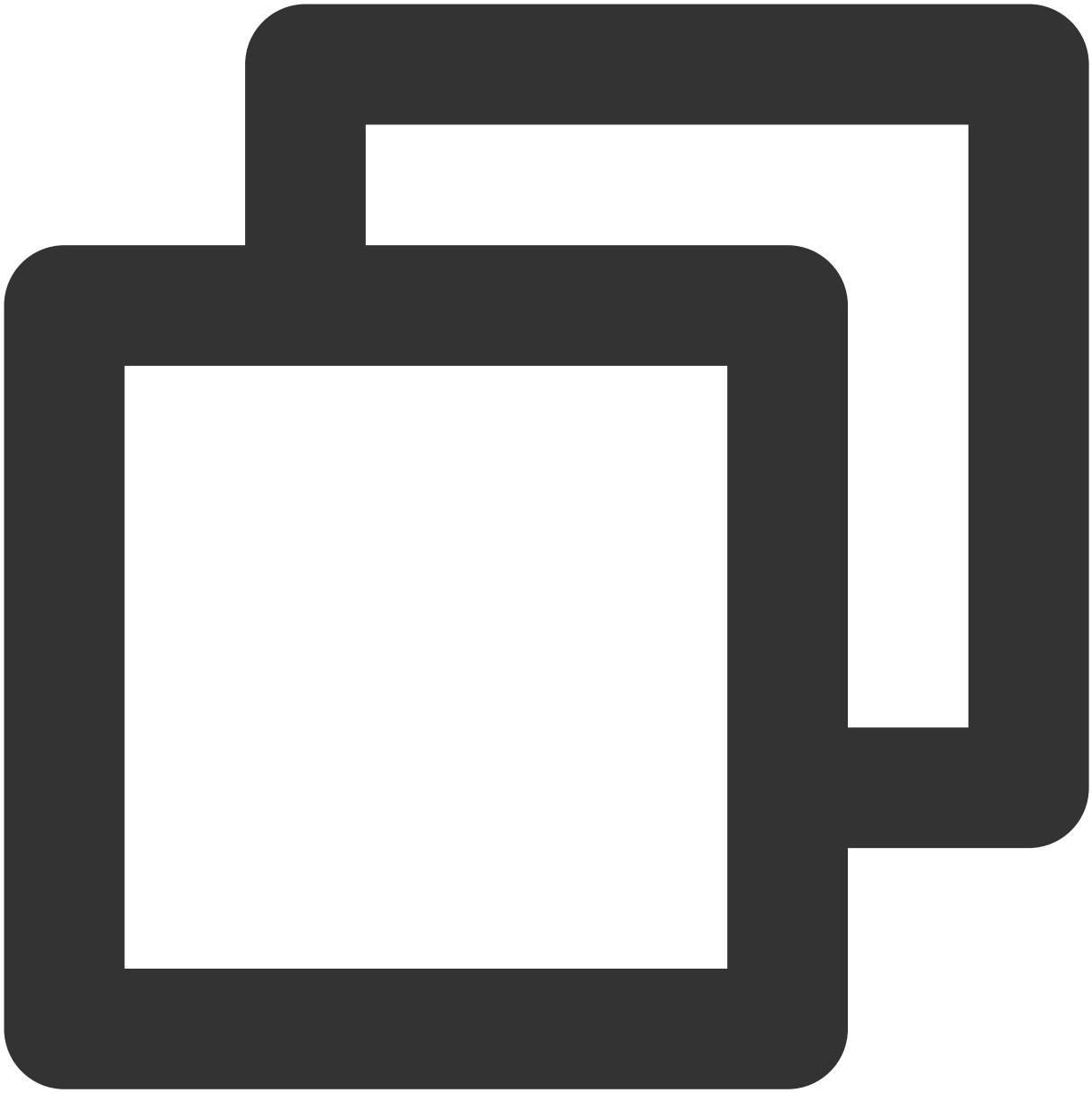
```
<uses-permission android:name="android.permission.INTERNET"/>
<application android:usesCleartextTraffic="true">...</application>
```

3. Add the WebView in the layout file of the activity.



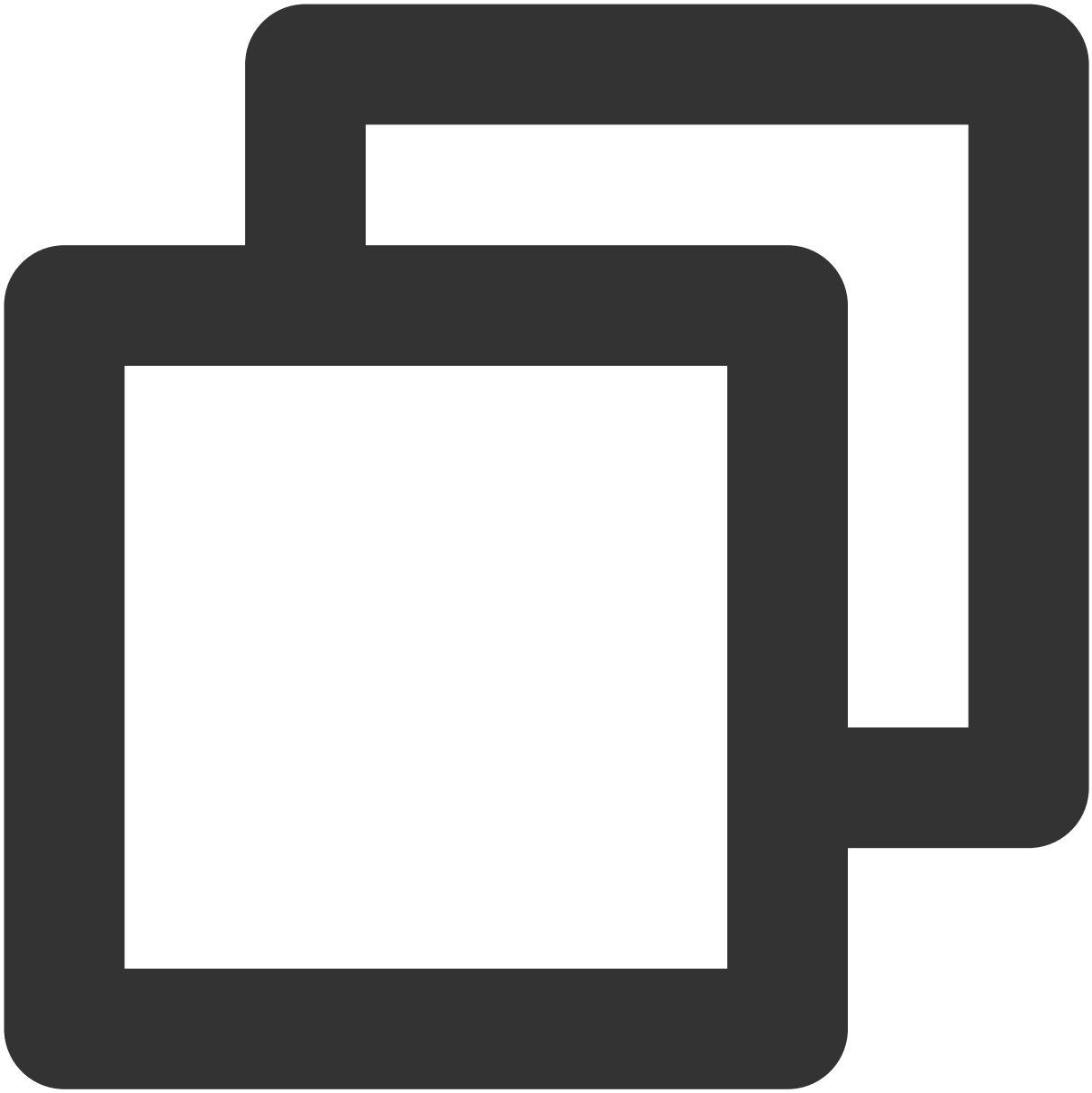
```
<WebView
android:id="@+id/webview"
android:layout_height="match_parent"
android:layout_width="match_parent"
/>
```

4. In the project, add a custom JavascriptInterface file and define a method to obtain data.



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

5. In the activity file, load the H5 business page.



```
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);
// Disable cache
webSettings.setCacheMode(WebSettings.LOAD_NO_CACHE);
webview.setWebViewClient(new WebViewClient(){
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
view.loadUrl(url);
return true;
}
});
// Enable js
webSettings.setJavaScriptEnabled(true);
webview.addJavascriptInterface(new JsBridge(), "jsBridge");
// Or load a local html file (webView.loadUrl("file:///android_asset/xxx.html"))
webview.loadUrl("https://x.x.x/x/");
}
}
```

6. Integrate TenDI Captcha in the H5 business page (for more information, please see Web Integration), and use JSBridge to pass the verification data to the business side.

**Note:**

After TenDI Captcha is integrated into the business client, the business server needs to verify the CAPTCHA ticket (if ticket verification is not integrated, the black market can easily forge verification results, which defeats the purpose of human verification via CAPTCHAs). For more information, please see Integration to Ticket Verification (Web and App).

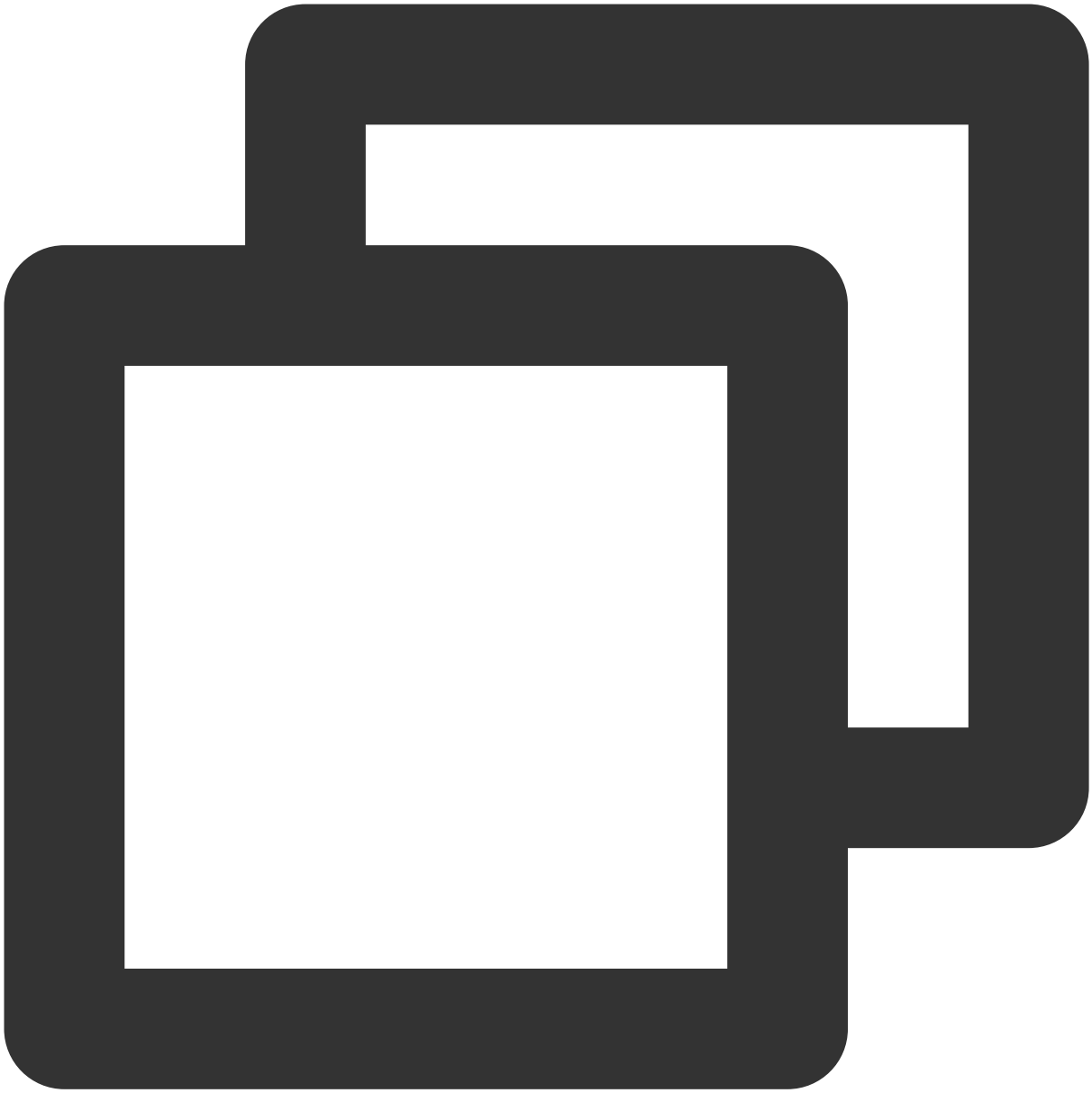
## Integration in iOS

### General process

1. Open a WebView in iOS, load an HTML page using JSBridge, and inject a method to be called in the HTML page to pass verification results.
2. Integrate TenDI Captcha in the HTML page, call the CAPTCHA JS to render the verification page, and then call the injected method to pass the verification result. For more information, please see Web Integration.
3. Return the verification result to iOS using JSBridge, and pass parameters such as ticket and random number to the business server for ticket verification. For more information, please see Integration to Ticket Verification (Web and App).

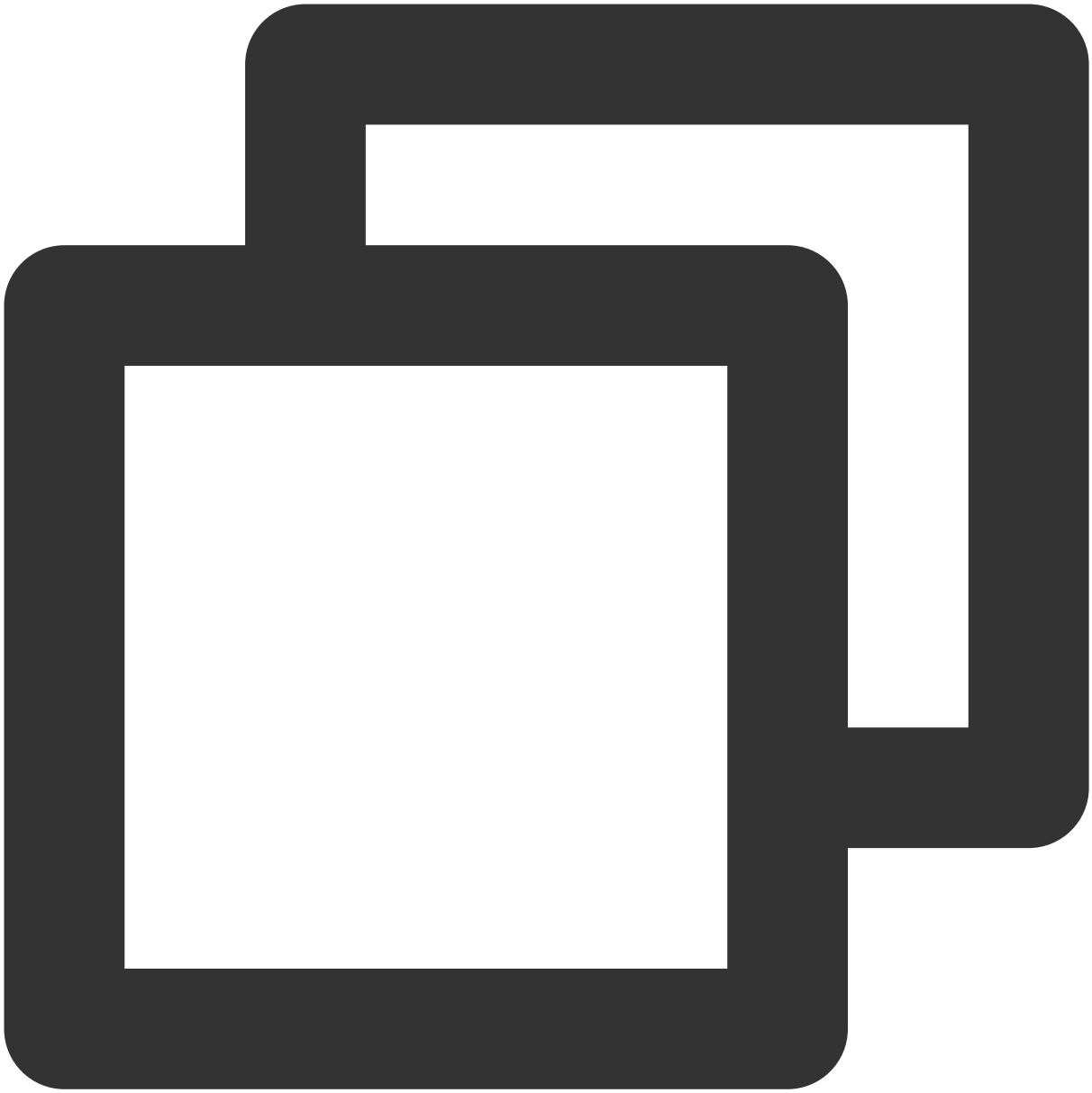
### Detailed steps

1. In a controller or view, import the WebKit library.



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

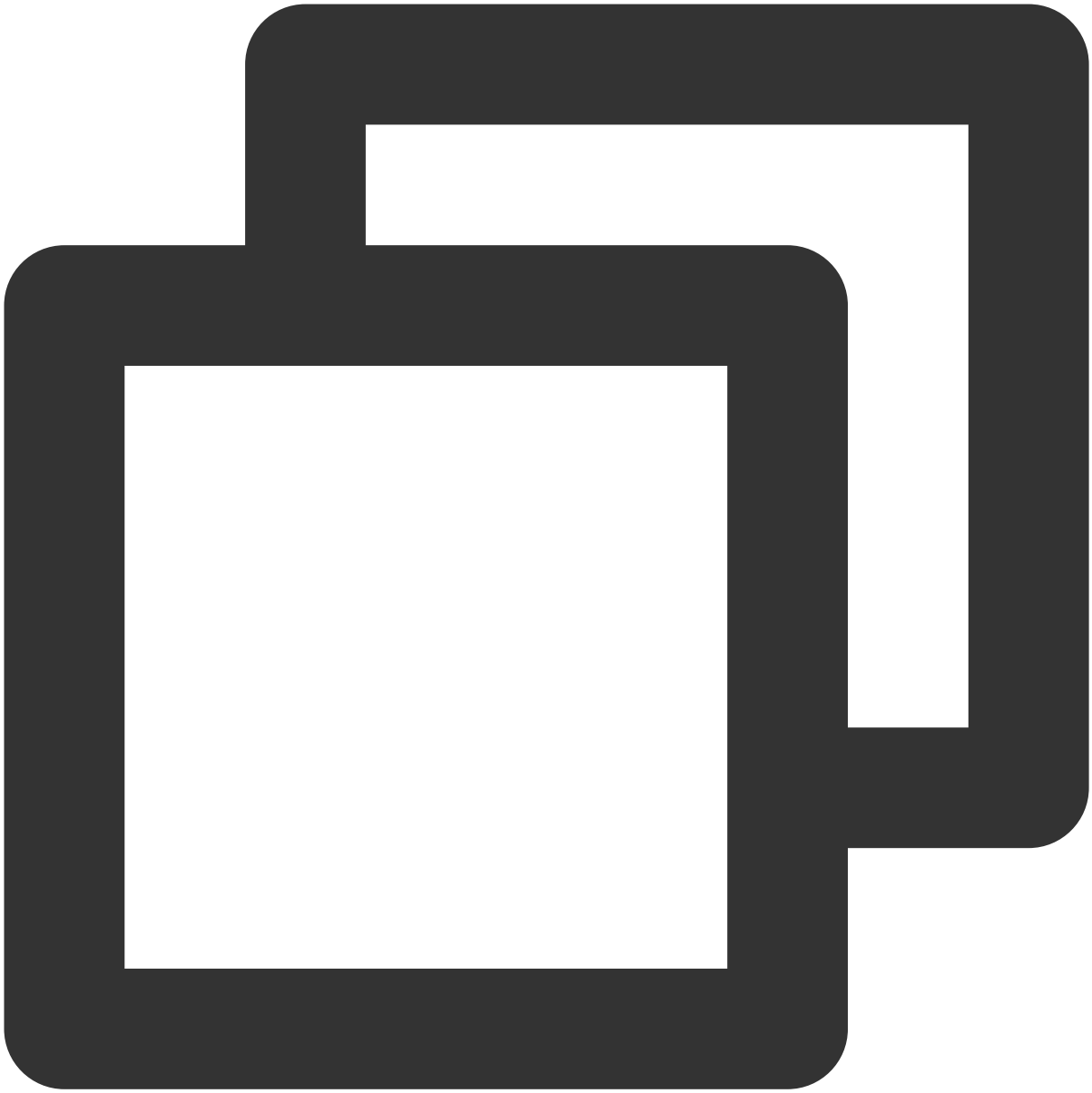
2. Create a WebView and render it.



```
-(WKWebView *)webView{
if(_webView == nil){
 // Create a webpage configuration object
 WKWebViewConfiguration *config = [[WKWebViewConfiguration alloc] init];
 // Create a settings object
 WKPreferences *preference = [[WKPreferences alloc]init];
 // Set "Whether javaScript is supported". The value is YES by default.
 preference.javaScriptEnabled = YES;
 // Set "Allow javaScript to open a window without user interaction". In iOS, the v
 preference.javaScriptCanOpenWindowsAutomatically = YES;
 config.preferences = preference;
```

```
    // This class is used to manage interactions between native and JavaScript.
    WKUserContentController * wkUController = [[WKUserContentController alloc] init];
    // Register a js method with the name jsToOcNoPrams. Set objects to handle and rec
    [wkUController addScriptMessageHandler:self  name:@"jsToOcNoPrams"];
    [wkUController addScriptMessageHandler:self  name:@"jsToOcWithPrams"];
    config.userContentController = wkUController;
    _webView = [[WKWebView alloc] initWithFrame:CGRectMake(0, 0, SCREEN_WIDTH, SCREEN_
    // UI delegate
    _webView.UIDelegate = self;
    // Navigation delegate
    _webView.navigationDelegate = self;
    // Webpage to be rendered
    NSString *path = [[NSBundle mainBundle] pathForResource:@"JStoOC.html" ofType:nil]
    NSString *htmlString = [[NSString alloc]initWithContentsOfFile:path encoding:NSUTF
    [_webView loadHTMLString:htmlString baseURL:[NSURL fileURLWithPath:[[NSBundle main
}
return _webView;
}
[self.view addSubview:self.webView];
```

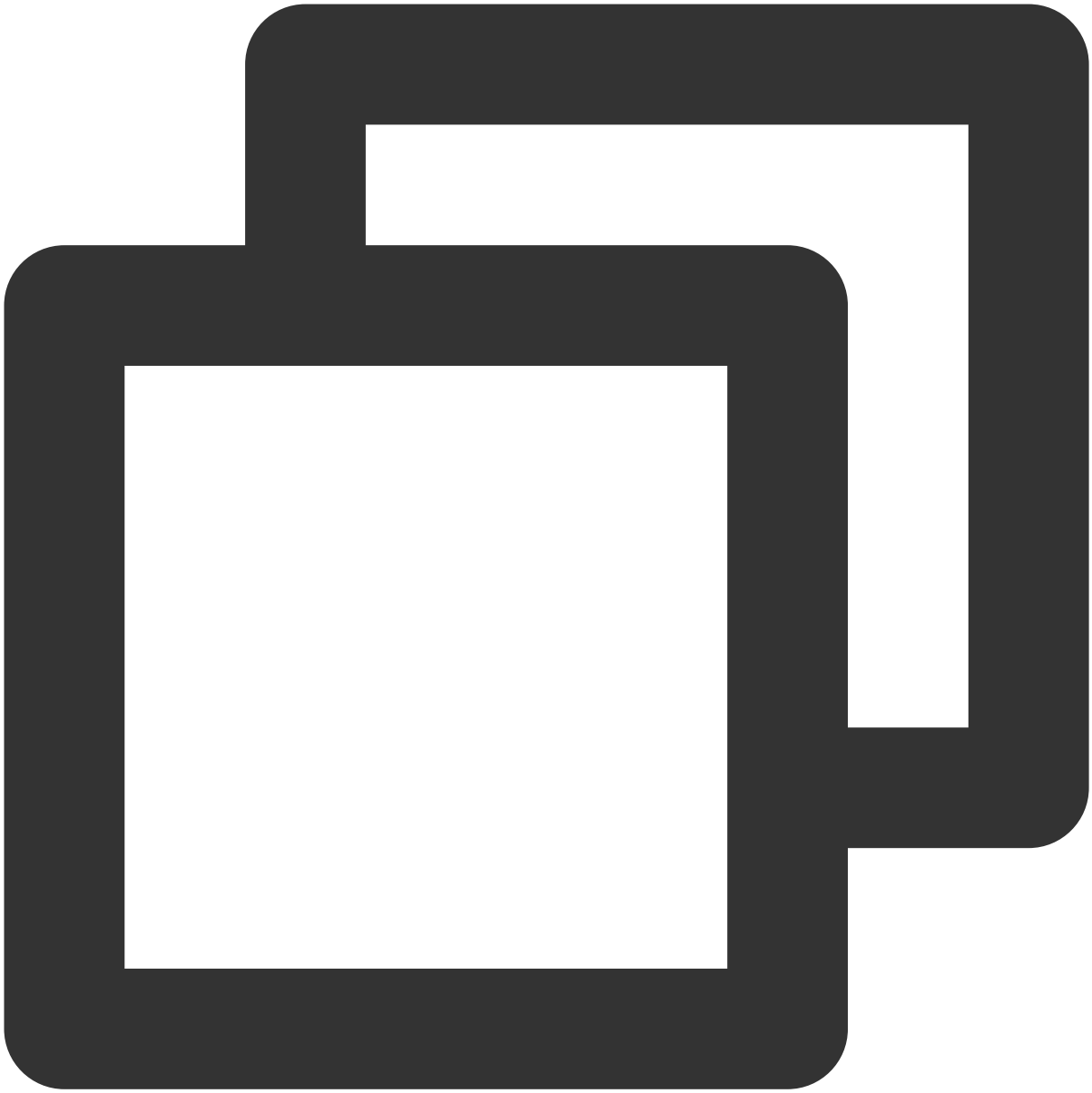
3. Set delegate methods to process some response events.



```
// Called when the page starts to load
-(void)webView:(WKWebView *)webView didStartProvisionalNavigation:(WKNavigation *)n
}
// Called when the page fails to load
-(void)webView:(WKWebView *)webView didFailProvisionalNavigation:(null_unspecified
[self.progressView setProgress:0.0f animated:NO];
}
// Called when content starts to be returned
-(void)webView:(WKWebView *)webView didCommitNavigation:(WKNavigation *)navigation
}
// Called when the page finishes loading
```

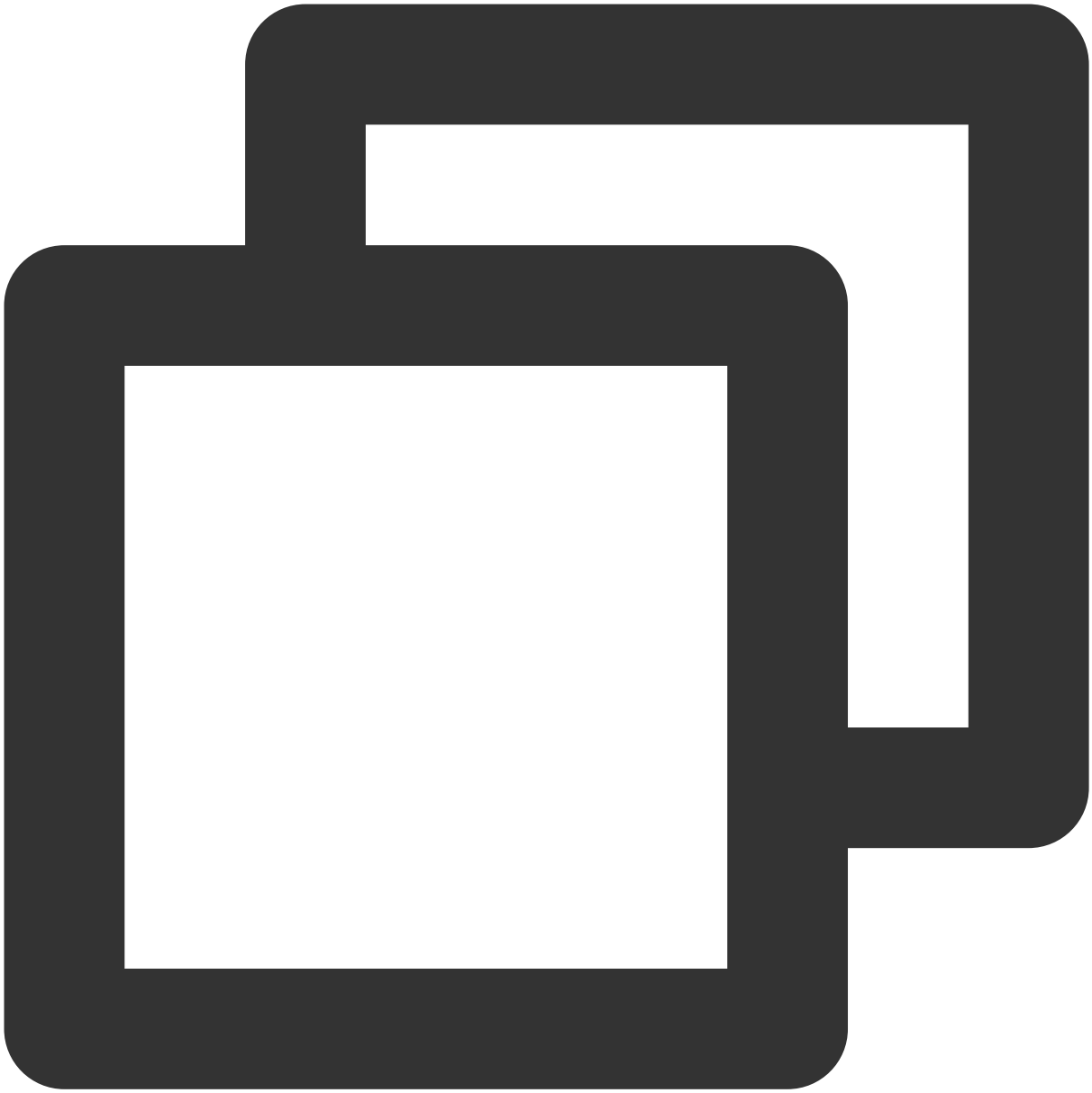
```
-(void)webView:(WKWebView *)webView didFinishNavigation:(WKNavigation *)navigation
[self getCookie];
}
// Called when a submission error occurs
-(void)webView:(WKWebView *)webView didFailNavigation:(WKNavigation *)navigation wi
[self.progressView setProgress:0.0f animated:NO];
}
// Called after receiving the server's redirect request, i.e., when the service is
-(void)webView:(WKWebView *)webView didReceiveServerRedirectForProvisionalNavigatio
}
```

4. Pass the parameters to OC with JS.



```
<p style="text-align:center"> <button id="btn2" type = "button" onclick = "jsToOcFu
function jsToOcFunction()
{
window.webkit.messageHandlers.jsToOcWithPrams.postMessage({"params":"res.randstr"})
}
```

5. Display the rendered WebView in the view, and then call the Captcha service and pass data to the client.



```
-(void)userContentController:(WKUserContentController *)userContentController didRe
// message.body is the json data passed to the client
// Use message.body to obtain the parameter body passed with JS
NSDictionary * parameter = message.body;
// Call OC with JS
if([message.name isEqualToString:@"jsToOcWithPrams"]){
// The client obtains the data passed with js and performs operations on the data
parameter[@"params"]
}
}
```

**Note:**

After TenDI Captcha is integrated into the business client, the business server needs to verify the CAPTCHA ticket (if ticket verification is not integrated, the black market can easily forge verification results, which defeats the purpose of human verification via CAPTCHAs). For more information, please see Integration to Ticket Verification (Web and App).

# FAQs

For more information, please see FAQs About Integration.

# Server Integration
# Integration to Ticket Verification (Web and App)

Last updated：2024-03-01 11:20:59

The server needs to call the ticket verification API to verify the client verification result.

**Note:**

If ticket verification is not integrated, the black market can easily forge verification results, which defeats the purpose of human verification via CAPTCHAs.

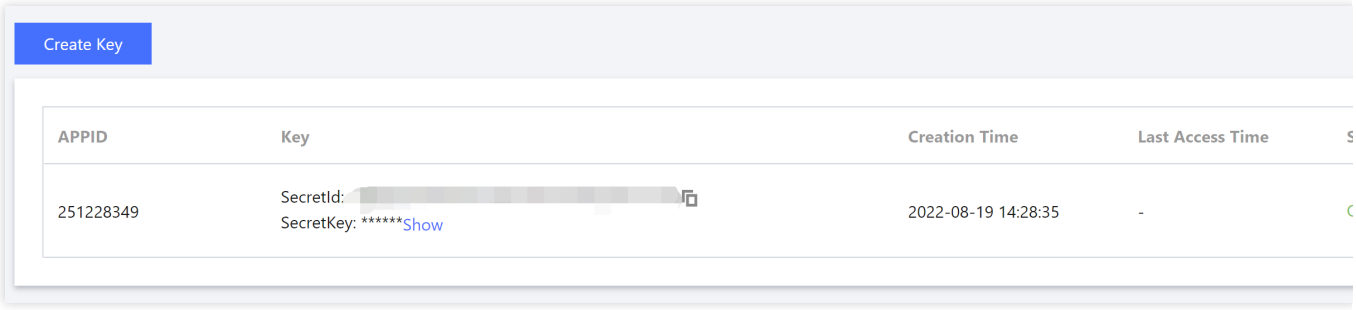
## Integration steps

### Step 1. Install the SDK in the corresponding language.

Tencent Cloud provides SDKs in multiple languages. You can install the SDK in any of the languages based on your business needs. For more information, please see SDK Center.

### Step 2. Obtain the TencentCloud API key.

1. Log in to the CAM console and select **Access Key** -> **API Key Management** on the left navigation pane.
2. On the API key management page, if the key has been created, you can view it on this page; if the key has not been created, click **Create Key** to generate the required key.




### Step 3. Call the `DescribeCaptchaResult` API.

It is recommended to use API Explorer to generate code online. For more information about the API, please see Verify CAPTCHA Ticket Result (Web and App).

# FAQs

For more information, please see Integration FAQs.