HTTPDNS

SDK Documentation

Product Documentation

Contents

SDK Documentation
SDK Activation Process

Last updated：2022-06-22 15:20:24

Overview

This document describes how to submit an application for connection to the SDK in the console after you activate HTTPDNS.

Prerequisites

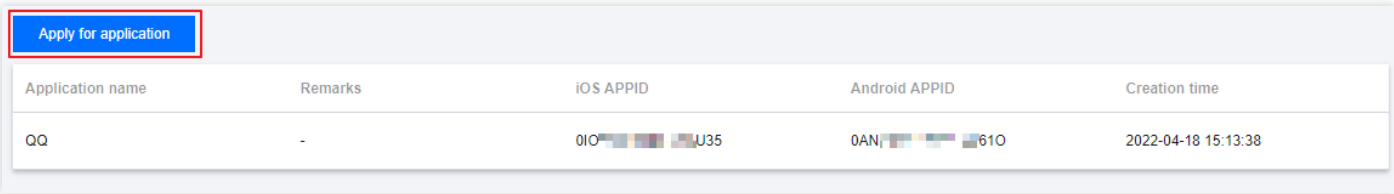
You have activated HTTPDNS.

Directions

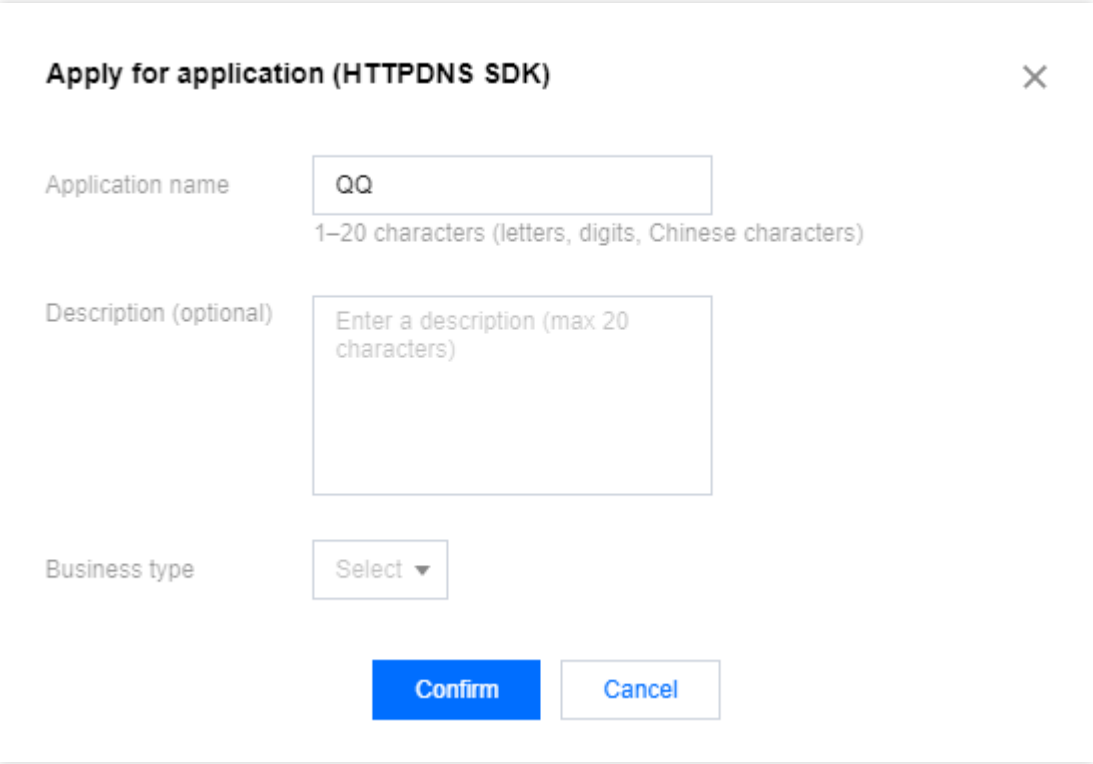
1. Log in to the HTTPDNS console.

2. On the left sidebar, select **Development Configuration**.

3. On the **Development Configuration** management page, click **Apply for application**.



4. In the **Apply for application (HTTPDNS SDK)** pop-up window, enter the required information.



- **Application name**: Enter the name of your application.
- **Description (optional)**: Enter the description of your application.
- **Business type**: Select an option based on your actual business conditions.

1. Click **Confirm**. `APPKey` values (one for Android and one for iOS) required for connection to the SDK will be sent to your Tencent Cloud verification email address. You can also view them on the **Development configuration** page.

Tencent Cloud

Your application for HTTPDNS SDK has been approved

Dear User,

The application for the HTTPDNS SDK submitted under your Tencent Cloud account (account ID: xxxxx, name: xxxxxx) has been approved. The AppID required for connecting to the SDK has been issued, which can be viewed in the console.

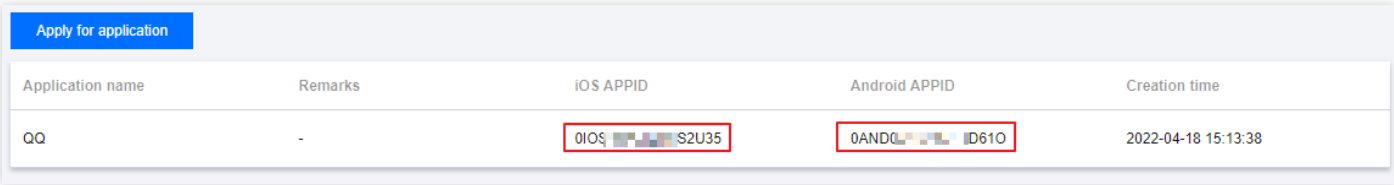
Note: To use the SDK, you need to replace the information with your account ID and AppID. For details, see the above documentation.

Console

Thank you!

Tencent Cloud

This is a system-generated message and please do not reply. If you don't want to receive these emails in the future, please **unsubscribe**.



SDK for iOS
Change History

Last updated：2023-09-05 09:53:07

v1.8.1 (August 28, 2023)

Optimized features.

v1.8.0 (July 5, 2023)

Added support for scheduling dnsip (HTTPDNS service IP) inside the SDK, without user configuration required.
Discontinued the Beacon data reporting service.
Optimized the package size.
Optimized features.

v1.7.0 (May 23, 2023)

Added support for data reporting and statistical analysis, which can be used via the **Query monitoring** feature provided by the console. The original data reporting service (Beacon) will be discontinued. Please switch the service as soon as possible.
Added an SDK dedicated for Tencent Cloud International.
Replaced WCDB with SQLite3 to achieve local cache.
Optimized features.

v1.6.0 (September 6, 2022)

Added the feature of local persistent cache.
Added support for returning the IP addresses in expired DNS records of a domain.

v1.5.0 (August 12, 2022)

Added the IP address ranking feature.

v1.4.0 (July 11, 2022)

Added the cache update logic.

v1.3.5 (March 14, 2022)

Optimized the multi-threading logic.

v1.3.4 (March 10, 2022)

Fixed bugs.

v1.3.3 (February 25, 2022)

Added support for DNS prefetch of domains.

Added support for returning all IPs.

Added support for specifying the type of IP to return.

v1.3.2 (February 8, 2022)

Optimized the multi-threading logic.

Integration

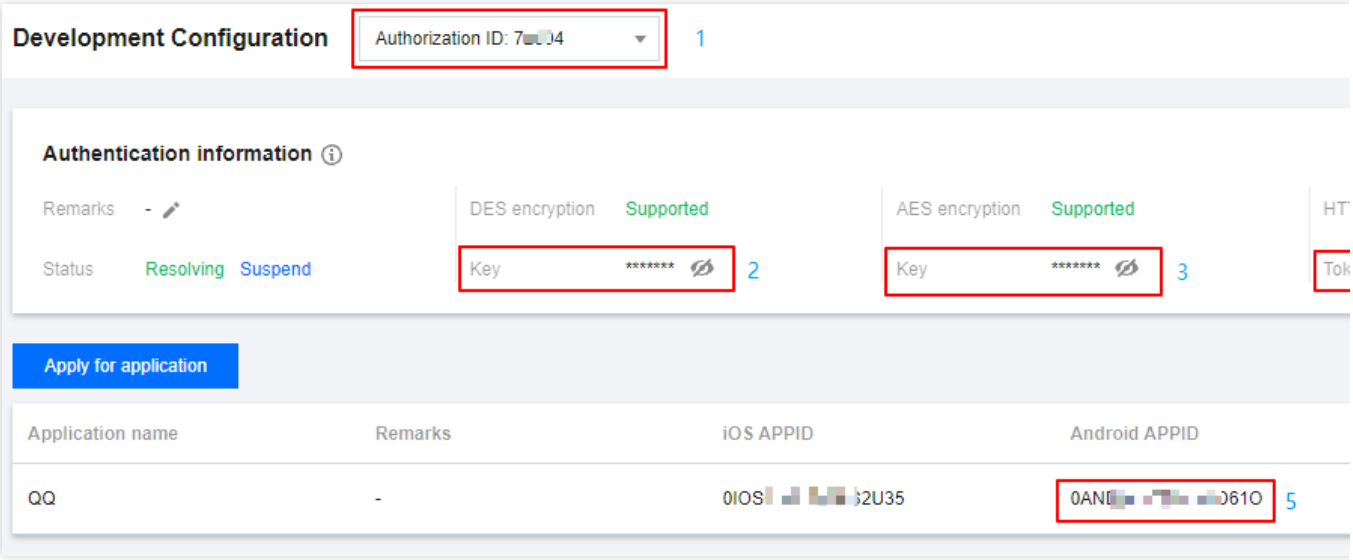
Last updated：2023-10-18 10:41:16

Overview

HTTPDNS helps avoid the failure to access the optimal access point during the traditional local DNS of ISPs. It works by replacing the traditional DNS protocol with the HTTP encryption protocol, without using domains throughout the process, which greatly reduces the possibility of DNS hijacking.

Preparations

1. You have activated HTTPDNS as instructed in Activating HTTPDNS.
2. You have added domains to be resolved in the HTTPDNS console as instructed in Adding a Domain.
3. You have applied in the HTTPDNS console for SDK integration as instructed in SDK Activation Process.
4. After activating the service, you have been assigned the configuration information such as authorization ID, AES and DES encryption keys, and HTTPS token. You can also view them on the Development Configuration page.



Configuration information required for using the SDK for iOS:

Authorization ID: The unique ID of a development configuration used in HTTPDNS, namely, the `dnsId` parameter in the SDK used for DNS authentication.

DES encryption key: The `dnsKey` parameter in the SDK, which you need to pass in when using the DES encryption method.

AES encryption key: The `dnsKey` parameter in the SDK, which you need to pass in when using the AES encryption method.
HTTPS encryption token: The `token` parameter in the SDK, which you need to pass in when using the HTTPS encryption method.
iOS APPID (optional): Obtained from the HTTPDNS console for data reporting.

Installation package structure

Download the latest version of the SDK package here.
Get the open-source repository of the SDK here.

Name	Applicable Scope
MSDKDns.xcframework	Applicable to projects with "Build Setting->C Language Dialect" configured as **"GNU98" **and "Build Setting->C++ Standard Library" as** "libstdc++(GNU C++ standard library)"**.
MSDKDns_intl.xcframework	MSDKDns.xcframework version for Tencent Cloud International
MSDKDns_C11.xcframework	Applicable to projects with "Build Setting->C Language Dialect" and "Build Setting->C Standard Library" configured as **"GNU++11"** and **"libc++(LLVM C** standard library with C11 support)", respectively.
MSDKDns_C11_intl.xcframework	MSDKDns_C11.xcframework version for Tencent Cloud International

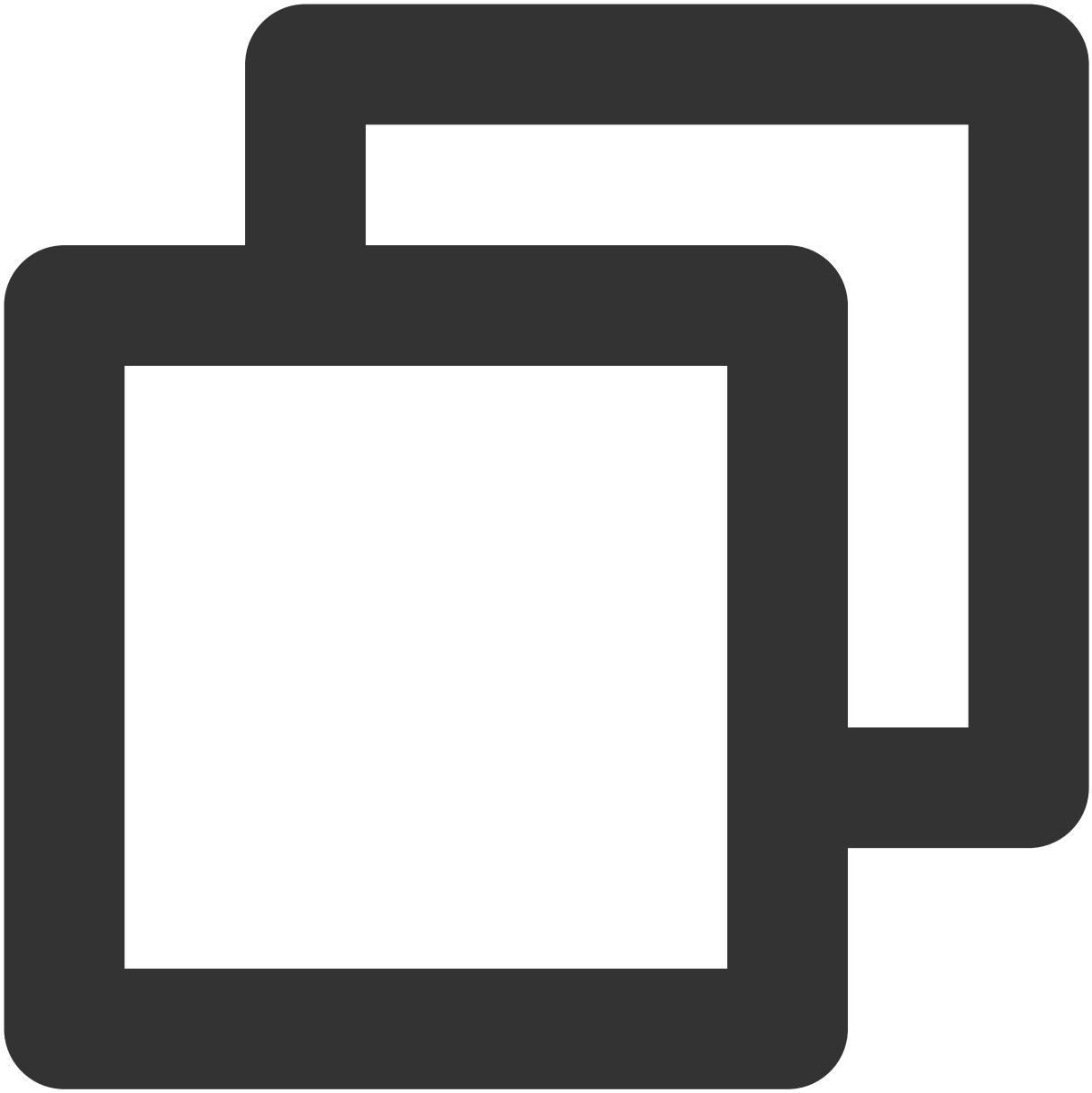
Note
If you want to use the version for Tencent Cloud International, obtain the initialization configuration from the HTTPDNS **international** console.
See Access Documentation for details.

SDK integration

HTTPDNS provides two integration methods for iOS:
Integration using CocoaPods
Manual integration

Integration using CocoaPods

1. Add the following code in the `Podfile` of the project:



```
# Applicable to projects with "Build Setting->C++ Language Dialect" configured as *
  pod 'MSDKDns_intl'
# Applicable to projects with "Build Setting->C++ Language Dialect" and "Build Sett
# pod 'MSDKDns_C11_intl'
```

2. Save, run `pod install` , and open the project with a file with the `.xcworkspace` extension.

Note

For more information on `CocoaPods` , see CocoaPods.

Manual integration

For the directions of manual integration, see the following demos:

Download the Objective-C demo here.

Download the Swift demo here.

Perform the following steps:

1. Import dependency libraries (in the `HTTPDNSLibs` directory):

`MSDKDns_C11_intl.framework` (or `MSDKDns_intl.framework` , depending on the project configuration)

2. Import system libraries:

libz.tbd

libsqlite3.tbd

libc++.tbd

Foundation.framework

CoreTelephony.framework

SystemConfiguration.framework

CoreGraphics.framework

Security.framework

Note

Add the `-ObjC` flag in `Other Linker Flags` .

SDK initialization

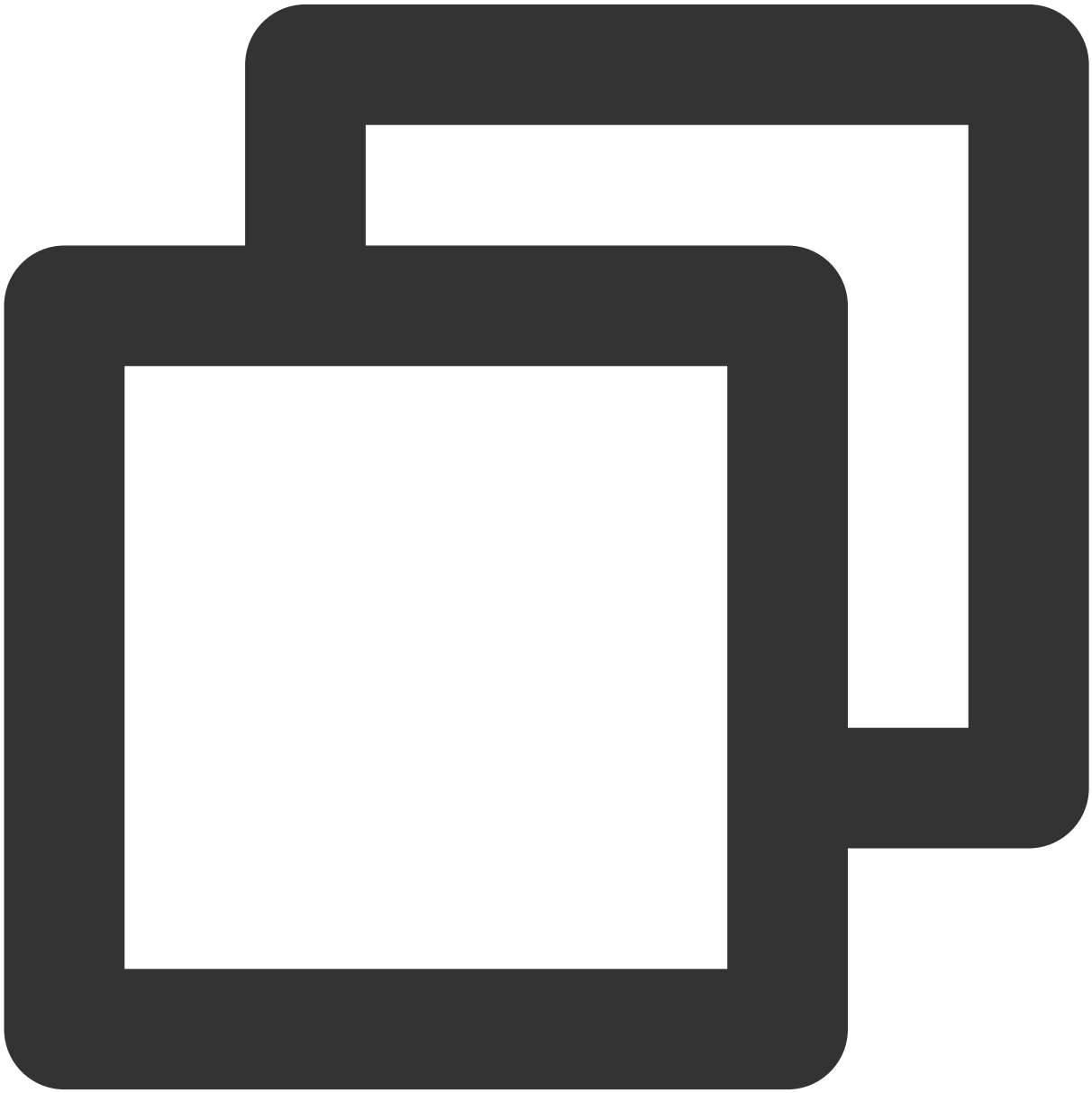
Sample API call:

In an Objective-C project:

```
// Use the following method for a .m file:
DnsConfig config = {
      .dnsId = DNS authorization ID, // Obtain the authorization ID from the HTTP
      .dnsKey = @"Encryption key",
      .encryptType = HttpDnsEncryptTypeDES,
      .debug = YES,
      .timeout = 2000,
};
[[MSDKDns sharedInstance] initConfig: &config];

// Use the following method for a .mm file:
```

```
DnsConfig *config = new DnsConfig();
config->dnsId = DNS authorization ID; // Obtain the authorization ID from the HTT
config->dnsKey = @"Encryption key";
config->encryptType = HttpDnsEncryptTypeDES;
config->debug = YES;
config->timeout = 2000;
[[MSDKDns sharedInstance] initConfig: config];
```

In a Swift project:



```
let msdkDns = MSDKDns.sharedInstance() as? MSDKDns;
msdkDns?.initConfig(with: [
```

```
        "dnsId": "DNS authorization ID", // Obtain the authorization ID from the HTTP
        "dnsKey": "Encryption key",
        "encryptType": 0, // 0: DES; 1: AES; 2: HTTPS
 ]);
```

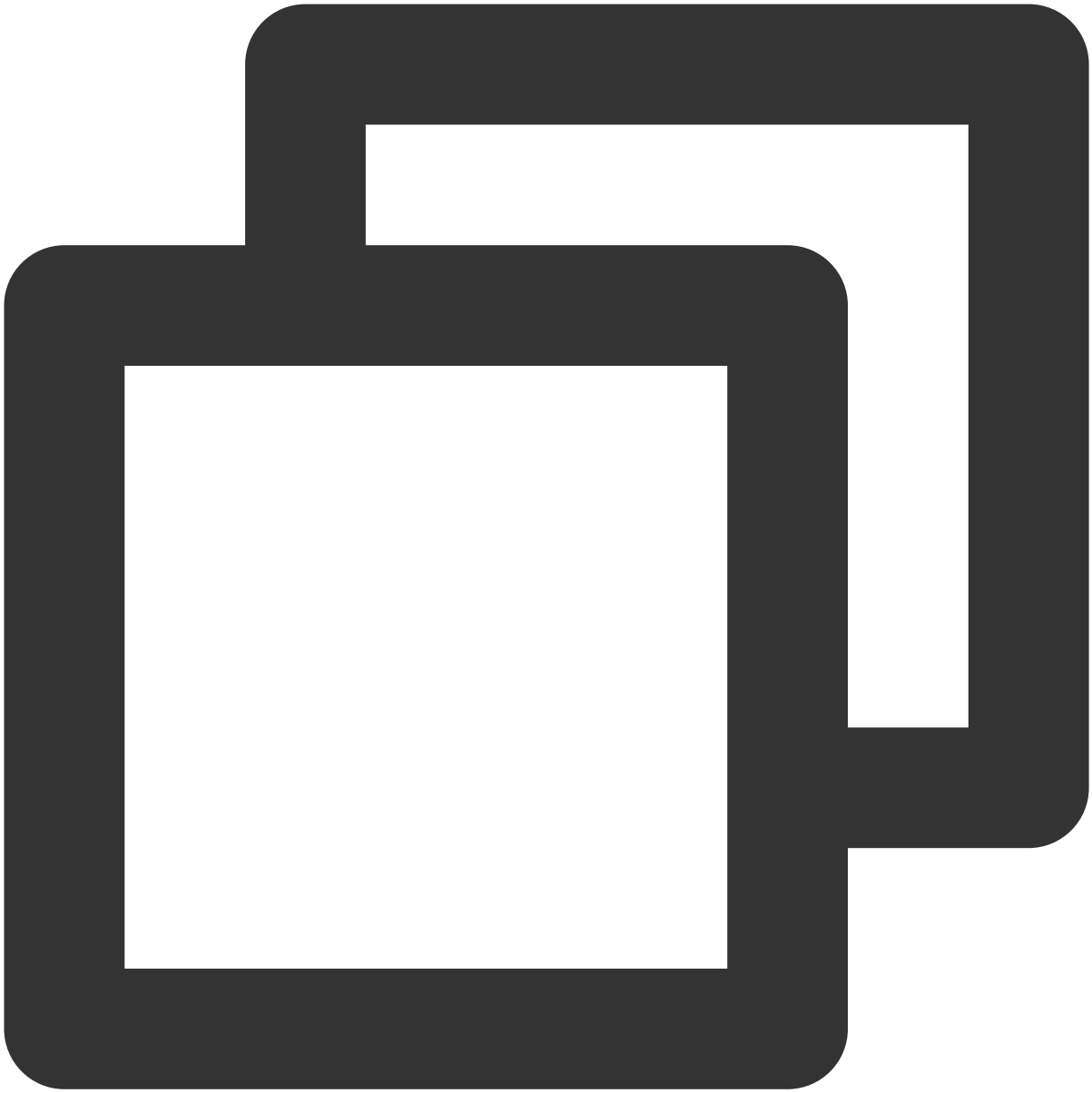
Integration verification

Log verification

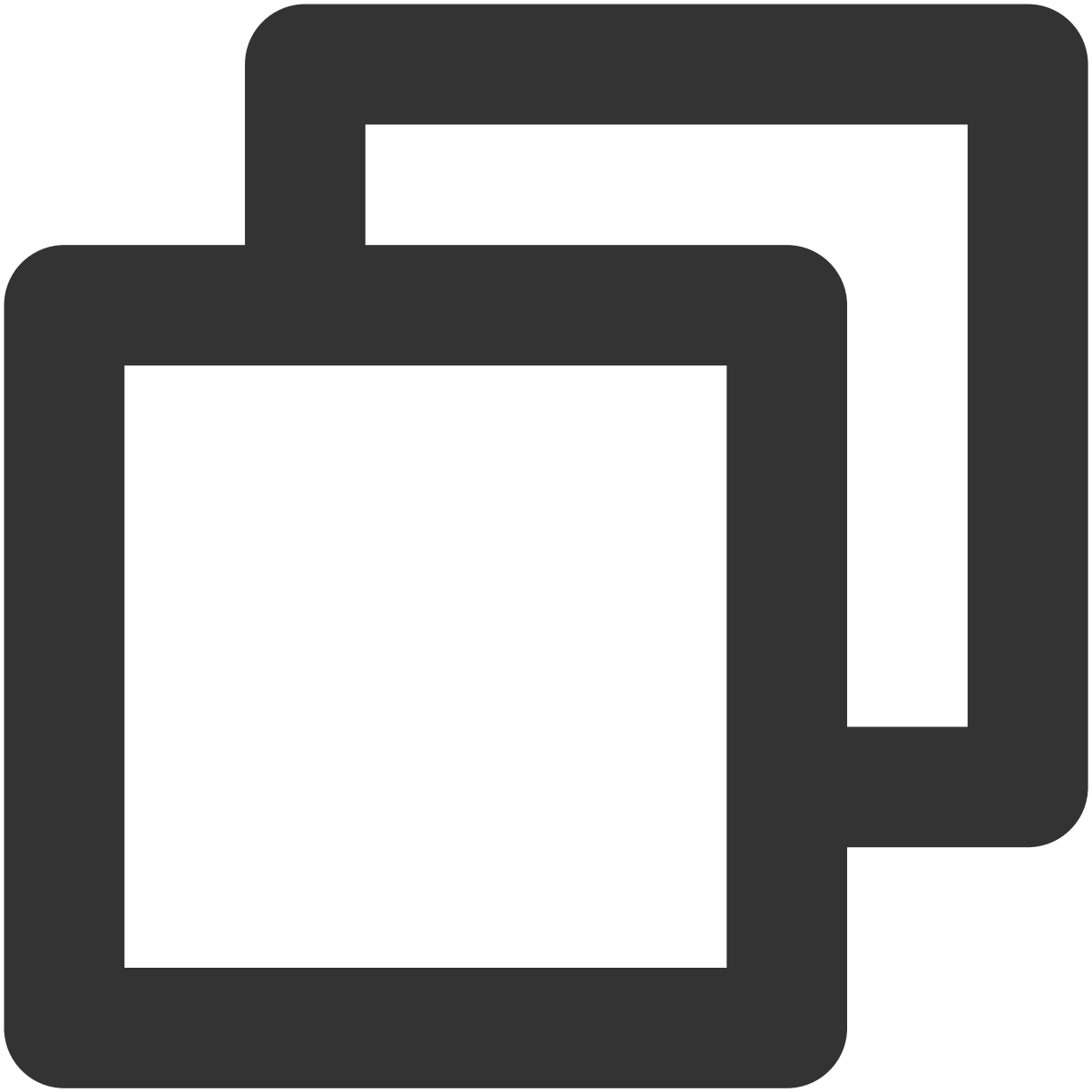
Enable the SDK DEBUG logging (set **debug** in **DnsConfig** to **YES**), find the printed `api name:HDNSGetHostByName, data: { ... }` log, and check the log info of the local DNS (**ldns_ip** in the log) and HTTPDNS (**hdns_ip** in the log) to determine whether the connection is successful.
The **hdns_ip** key indicates the DNS query results of A records provided by HTTPDNS.
The **hdns_4a_ips** key indicates the DNS query results of AAAA records provided by HTTPDNS.
If **hdns_ip** or **hdns_4a_ips** is not empty, the connection is successful.
The **ldns_ip** key indicates the DNS query results provided by the local DNS.

Notes

If the service on the client is bound to the host, such as the HTTP service or the CDN service of the host, you need to specify the host field of the HTTP header after replacing the domain in the URL with the IP returned by HTTPDNS.
Example:
NSURLConnection
NSURLSession
curl
The `WWW` API of Unity



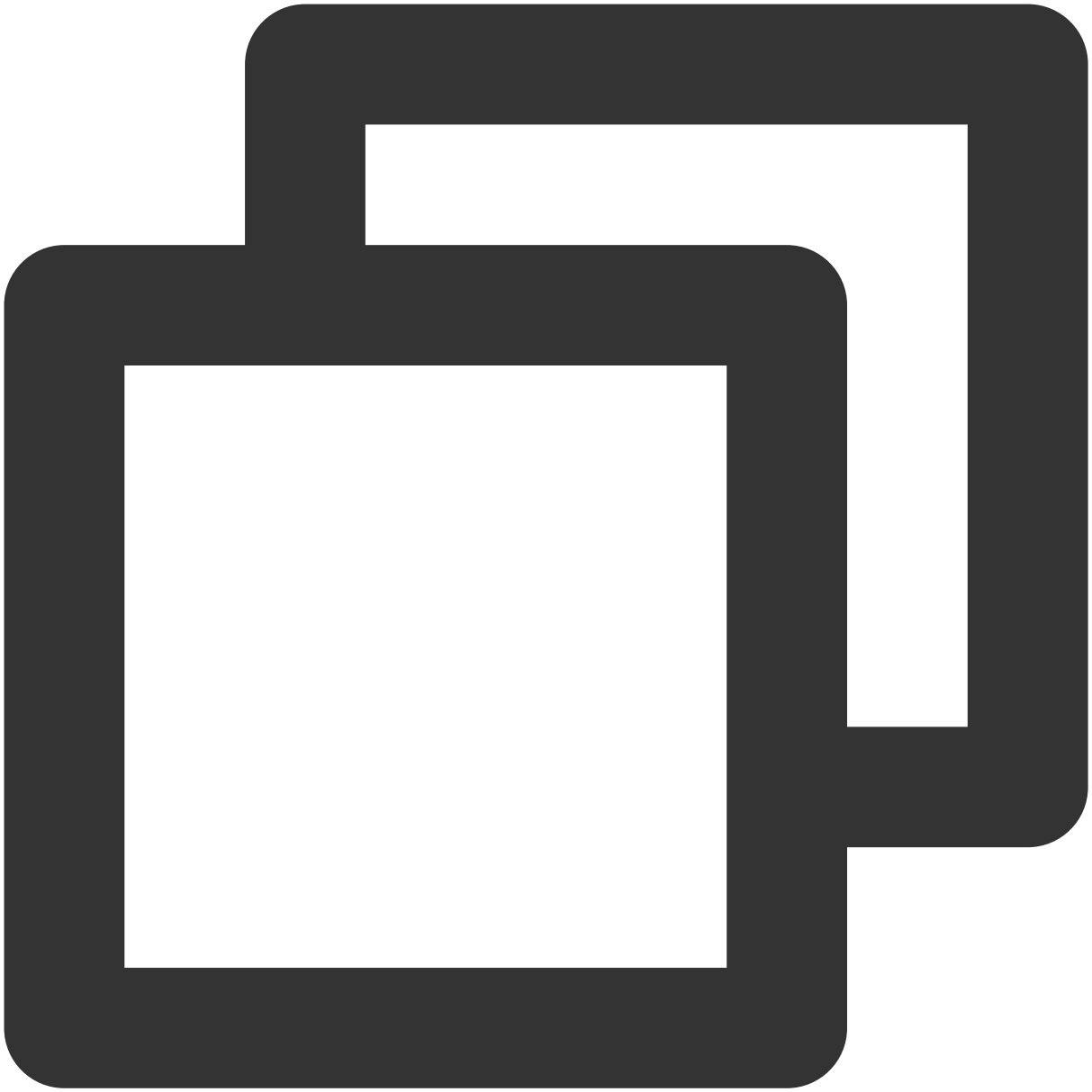
```
NSURL *httpDnsURL = [NSURL URLWithString:@"URL obtained by concatenating the resolv
float timeOut = The set timeout period;
NSMutableURLRequest *mutableReq = [NSMutableURLRequest requestWithURL:httpDnsURL ca
[mutableReq setValue:@"original domain" forHTTPHeaderField:@"host"];
NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest:mutableReq d
[connection start];
```



```
NSURL *httpDnsURL = [NSURL URLWithString:@"URL obtained by concatenating the resolv
float timeOut = The set timeout period;
NSMutableURLRequest *mutableReq = [NSMutableURLRequest requestWithURL:httpDnsURL ca
[mutableReq setValue:@"original domain" forHTTPHeaderField:@"host"];
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSessio
NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration delega
NSURLSessionTask *task = [session dataTaskWithRequest:mutableReq];
[task resume];
```

If you want to access `www.qq.com` and the IP obtained by HTTPDNS is `192.168.0.111`, you can perform a call as follows:

```
curl -H "host:www.qq.com" http://192.168.0.111/aaa.txt.
```



```
string httpDnsURL = "URL obtained by concatenating the resolved IP";
Dictionary<string, string> headers = new Dictionary<string, string> ();
headers["host"] = "Original domain";
WWW conn = new WWW (url, null, headers);
yield return conn;
if (conn.error != null) {
    print("error is happened:"+ conn.error);
} else {
    print("request ok" + conn.text);
}
```

Check whether an HTTP proxy is used locally. If an HTTP proxy is used, we recommend you **not use HTTPDNS** to resolve domains.

Check whether an HTTP proxy is used:

```
- (BOOL)isUseHTTPProxy {
    CFDictionaryRef dicRef = CFNetworkCopySystemProxySettings();
    const CFStringRef proxyCFstr = (const CFStringRef)CFDictionaryGetValue(dicRef,
    NSString *proxy = (__bridge NSString *)proxyCFstr;
    if (proxy) {
        return YES;
    } else {
```

```
        return NO;
    }
}
```

Check whether an HTTPS proxy is used:

```
- (BOOL)isUseHTTPSProxy {
    CFDictionaryRef dicRef = CFNetworkCopySystemProxySettings();
    const CFStringRef proxyCFstr = (const CFStringRef)CFDictionaryGetValue(dicRef,
    NSString *proxy = (__bridge NSString *)proxyCFstr;
    if (proxy) {
        return YES;
```

```
    } else {
        return NO;
    }
}
```

APIs

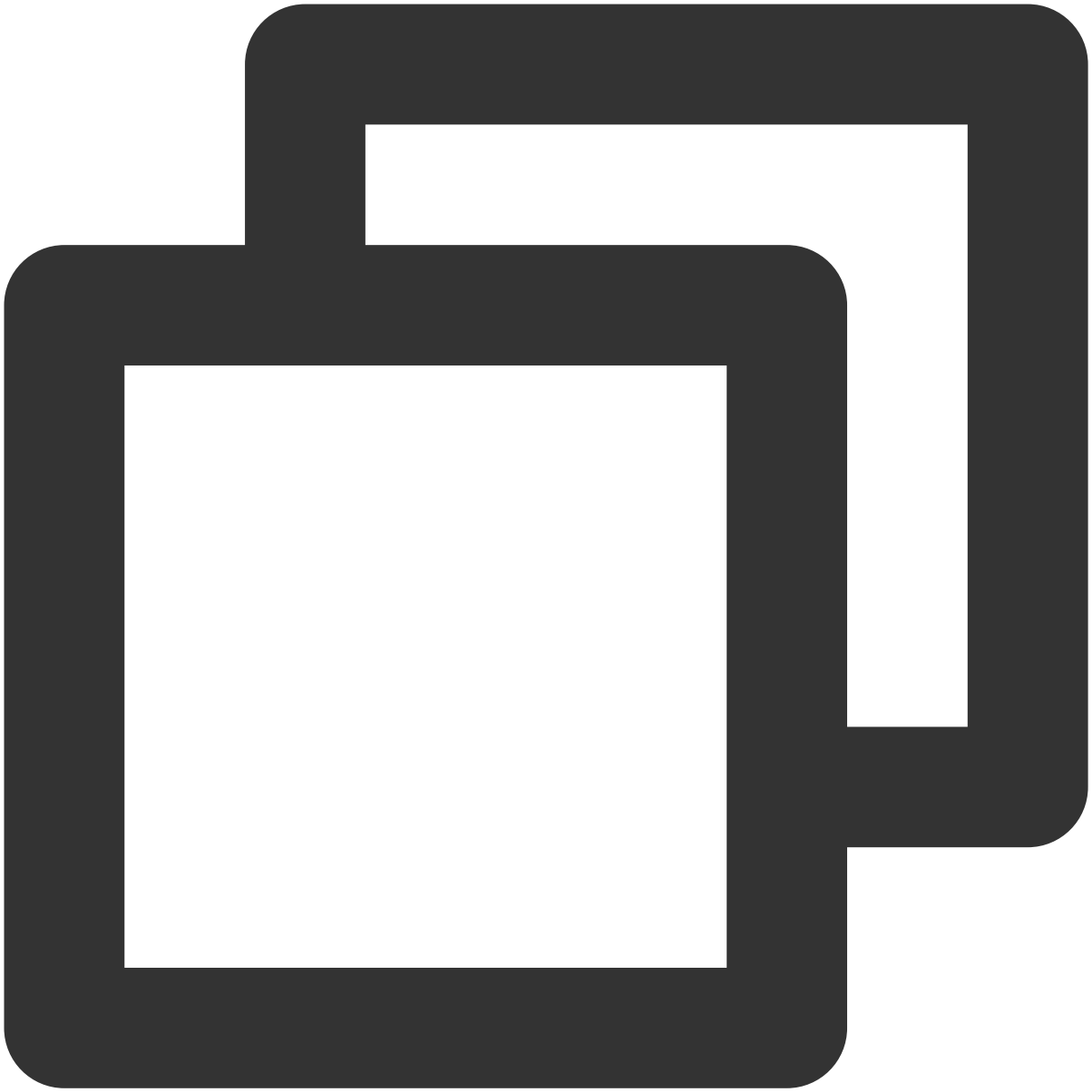
Last updated：2023-07-11 10:51:39

Setting basic business information

Type definition

Note

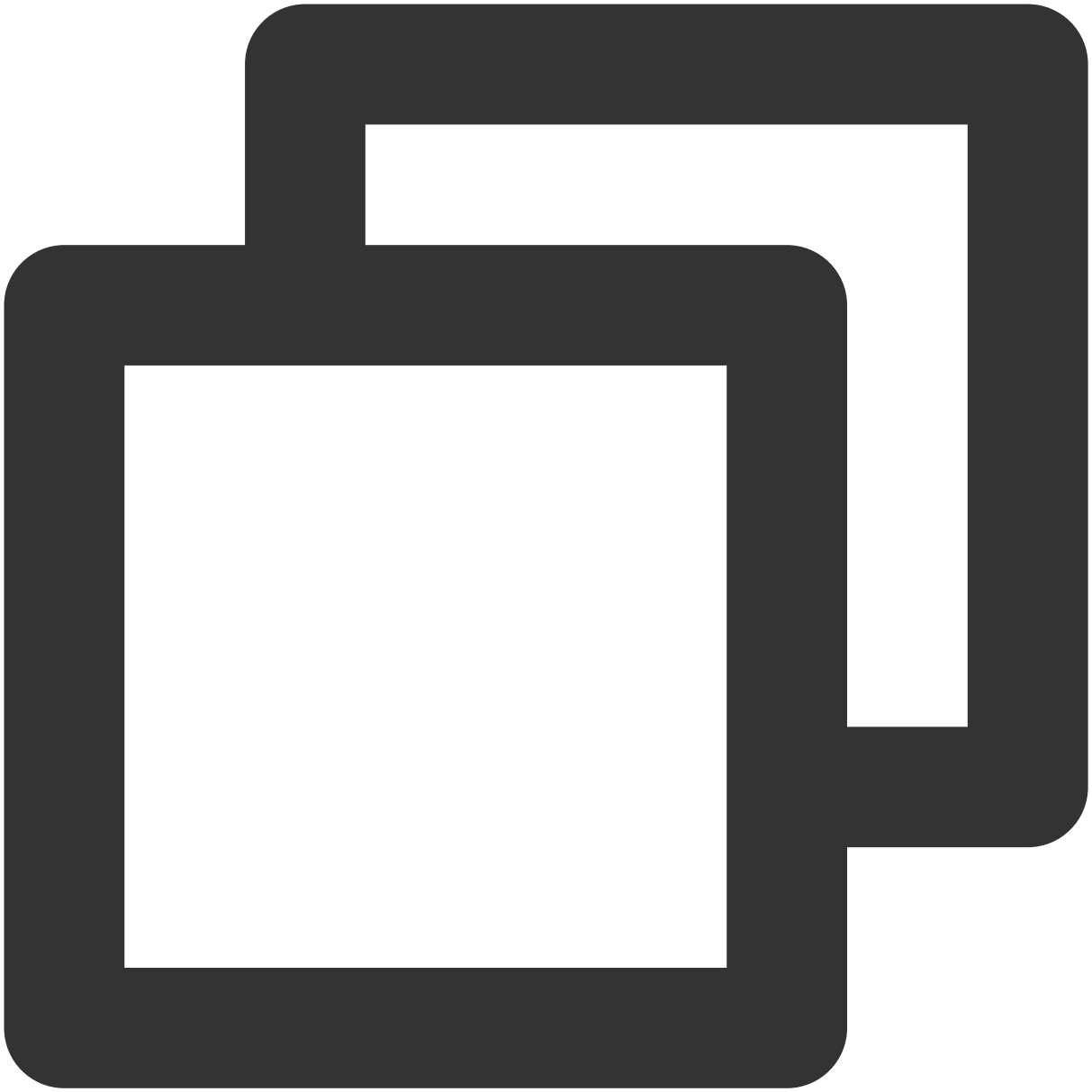
[Disused from v1.7.0] The SDK log reporting capability is configured in the console.



```
/**
    Encryption method
**/
typedef enum {
    HttpDnsEncryptTypeDES = 0, // DES encryption
    HttpDnsEncryptTypeAES = 1, // AES encryption
    HttpDnsEncryptTypeHTTPS = 2 // HTTPS encryption
} HttpDnsEncryptType;

/**
     IP address type
```

```
**/
typedef enum {
    HttpDnsAddressTypeAuto = 0, // Automatically detected by the SDK
    HttpDnsAddressTypeIPv4 = 1, // Only IPv4 is supported
    HttpDnsAddressTypeIPv6 = 2, // Only IPv6 is supported
    HttpDnsAddressTypeDual = 3, // Both protocols are supported
} HttpDnsAddressType;

/**
    Configuration structure
    You can get the following authentication information after activating the servi
**/
typedef struct DnsConfigStruct {
    NSString* appId; // Application ID, which is optional and can be obtained after
    int dnsId; // Authorization ID, which you can directly view in the HTTPDNS cons
    NSString* dnsKey; // Encryption key, which you must pass in for the AES or DES
    NSString* token; // Encryption token, which you must pass in for the HTTPS encr
    NSString* dnsIp; // [For v1.8.0 and later, the IP is scheduled inside the SDK a
    BOOL debug; // Specify whether to enable debug logging. `YES`: Enable; `NO`: Di
    int timeout; // Timeout period in milliseconds, which is optional. If you set i
    HttpDnsEncryptType encryptType; // Set the encryption method
    HttpDnsAddressType addressType; // Specify the type of the returned IP address,
    NSString* routeIp; // The ECS (EDNS-Client-Subnet) value of the DNS request. By
    BOOL httpOnly;// This parameter is optional. It specifies whether to return onl
    NSUInteger retryTimesBeforeSwitchServer; // Number of retries before the IP swi
    NSUInteger minutesBeforeSwitchToMain; // Interval for switching back to the pri
    BOOL enableReport; // [Disused from v1.7.0] The SDK log reporting capability is
} DnsConfig;
```

API declaration



```
/**
 Set the basic business information (for Tencent Cloud services)

 @param config Business configuration structure
 @return YES: Set successfully. NO: Failed to set.
 */
- (BOOL) initConfig:(DnsConfig *)config;

/**
 * It is configured through `Dictionary` as for the `DnsConfig` structure. It is us
 *
```

```
 * @param config Configuration
 * @return YES: Set successfully. NO: Failed to set.
 */
- (BOOL) initConfigWithDictionary:(NSDictionary *)config;

/**
 // Domains for DNS prefetch. You can configure up to eight domains. DNS prefetch i
 * Sample code: [[MSDKDns sharedInstance] WGSetPreResolvedDomains:@[@"dnspod.com",
 * @param domains  Domain array
 */
- (void) WGSetPreResolvedDomains:(NSArray *)domains;

/**
 * Domains for automatic cache update. You can configure up to eight domains.
 * Sample code: [[MSDKDns sharedInstance] WGSetKeepAliveDomains:@[@"dnspod.com", @"
 * @param domains  Domain array
 */
- (void) WGSetKeepAliveDomains:(NSArray *)domains;


/**
 * Set whether to enable IP address ranking. To enable it, configure the port numbe
 */
- (void) WGSetIPRankData:(NSDictionary *)IPRankData;

/**
 * Set whether to return the IP addresses in expired DNS records of a domain. This
 * When this parameter is set to `true`, the SDK directly returns the cached DNS re
 */
- (void) WGSetExpiredIPEnabled:(BOOL)enable;

/**
 * Set whether to enable the feature of local persistent cache. It is disabled by d
 */
- (void) WGSetPersistCacheIPEnabled:(BOOL)enable;
```

Note

The HTTPDNS SDK provides multiple DNS optimization options, which can be combined for optimal DNS performance.

You can configure `(void) WGSetExpiredIPEnabled:(true)enable;` and `(void) WGSetPersistCacheIPEnabled:(true)enable;` to achieve optimistic DNS cache.

The persistent cache feature is used to improve cache hit rate and shorten the time to the first meaningful paint. This feature stores the last DNS result locally and preferentially reads the local cache when an app is launched.

If the cached DNS result has expired (TTL expired), since optimistic DNS cache allows the return of IP addresses in expired DNS results, the SDK returns the IP addresses in the expired DNS result (assuming that the IP addresses are

available in most cases) and initiates an async request to update the cache.

When optimistic DNS cache is enabled, cache cannot be hit (there is no cache) for the first DNS request of a domain, so `0;0` is returned, and an async DNS request is initiated to update the cache. Therefore, after enabling optimistic DNS cache, you need to ensure that local DNS will work if no cache is hit. For important domains, we recommend you enable DNS prefetch via `(void) WGSetPreResolvedDomains:(NSArray *)domains;`.

If the server IP addresses are changed frequently, be sure to enable automatic cache update via `(void) WGSetKeepAliveDomains:(NSArray *)domains;` and DNS prefetch via `(void) WGSetPreResolvedDomains:(NSArray *)domains;` to ensure the accuracy of DNS results.

Sample code

Sample API call:

In an Objective-C project:

```
DnsConfig *config = new DnsConfig();
config->dnsIp = @"HTTPDNS server IP";
config->dnsId = DNS authorization ID;
config->dnsKey = @"Encryption key";
config->encryptType = HttpDnsEncryptTypeDES;
config->debug = YES;
config->timeout = 2000;
[[MSDKDns sharedInstance] initConfig: config];
```

In a Swift project:

```
let msdkDns = MSDKDns.sharedInstance() as? MSDKDns;
msdkDns?.initConfig(with: [
      "dnsIp": "HTTPDNS server IP",
      "dnsId": "DNS authorization ID",
      "dnsKey": "Encryption key",
      "encryptType": 0, // 0: DES; 1: AES; 2: HTTPS
]);
```

DNS APIs

The following APIs are used to get IPs, and you only need to import the header file to call the corresponding API.

Sync APIs

Single query API: **WGGetHostByName:**

Batch query API (with one IP returned): **WGGetHostsByNames:**

Batch query API (with all IPs returned): **WGGetAllHostsByNames:**

Note

For a sync API, execution will block. Therefore, we recommended you call a sync API in a **child thread** or use an **async API**.

Async APIs

Single query API: **WGGetHostByNameAsync:returnIps:**

Batch query API (with one IP returned): **WGGetHostsByNamesAsync:returnIps:**

Batch query API (with all IPs returned): **WGGetAllHostsByNamesAsync:returnIps:**

The returned address is in the following format:

Single query: The single query API will return `NSArray` with a fixed length of 2, where the first value is the IPv4 address, and the second value is the IPv6 address. The response format is as described below:

Under IPv4, only the IPv4 address will be returned; that is, the response format will be `[ipv4, 0]` .

Under IPv6, only the IPv6 address will be returned; that is, the response format will be `[0, ipv6]` .

Under a dual stack network, the resolved IPv4 and IPv6 addresses (if they exist) will be returned; that is, the response format will be `[ipv4, ipv6]` .

If DNS query fails, `[0, 0]` will be returned; in this case, you only need to call the `WGGetHostByName` API again.

Batch query (with one IP returned): The batch query API will return an `NSDictionary` , where `key` is the queried domain, and `value` is `NSArray` with a fixed length of 2. The first value is the IPv4 address, and the second value is the IPv6 address. The response format is as described below:

Under IPv4, only the IPv4 address will be returned; that is, the response format will be `{"queryDomain" : [ipv4, 0]}` .

Under IPv6, only the IPv6 address will be returned; that is, the response format will be `{"queryDomain" : [0, ipv6]}` .

Under a dual stack network, the resolved IPv4 and IPv6 addresses (if they exist) will be returned; that is, the response format will be `{"queryDomain" : [ipv4, ipv6]}` .

If DNS query fails, `{"queryDomain" : [0, 0]}` will be returned; in this case, you only need to call the `WGGetHostByNames` API again.

Batch query (with all IPs returned): The batch query API will return an `NSDictionary` , where `key` is the queried domain, and `value` is an `NSDictionary` containing two keys (ipv4, ipv6) with corresponding

`NSArray` objects as values indicating all queried IPv4/IPv6 result IPs. The response format is as follows:
{"queryDomain" : { "ipv4": [], "ipv6": []}}

How to improve IPv6 usage

When you make a URL request by using an IPv6 address, you need to enclose the IPv6 address in square brackets, such as `http://[64:ff9b::b6fe:7475]/` .

If the IPv6 address is 0, you can directly use the IPv4 address for connection.

If the IPv4 address is 0, you can directly use the IPv6 address for connection.

If neither the IPv4 address nor the IPv6 address is 0, the client can determine which address to use first for connection; if the client fails to connect to the preferred address, it should switch to the other one.

When you integrate HTTPDNS through the SDK, if HTTPDNS does not find a DNS query result, the domain will be resolved by local DNS, and the result provided by local DNS will be returned.

Sync DNS APIs

API name

WGGetHostByName and WGGetHostsByNames

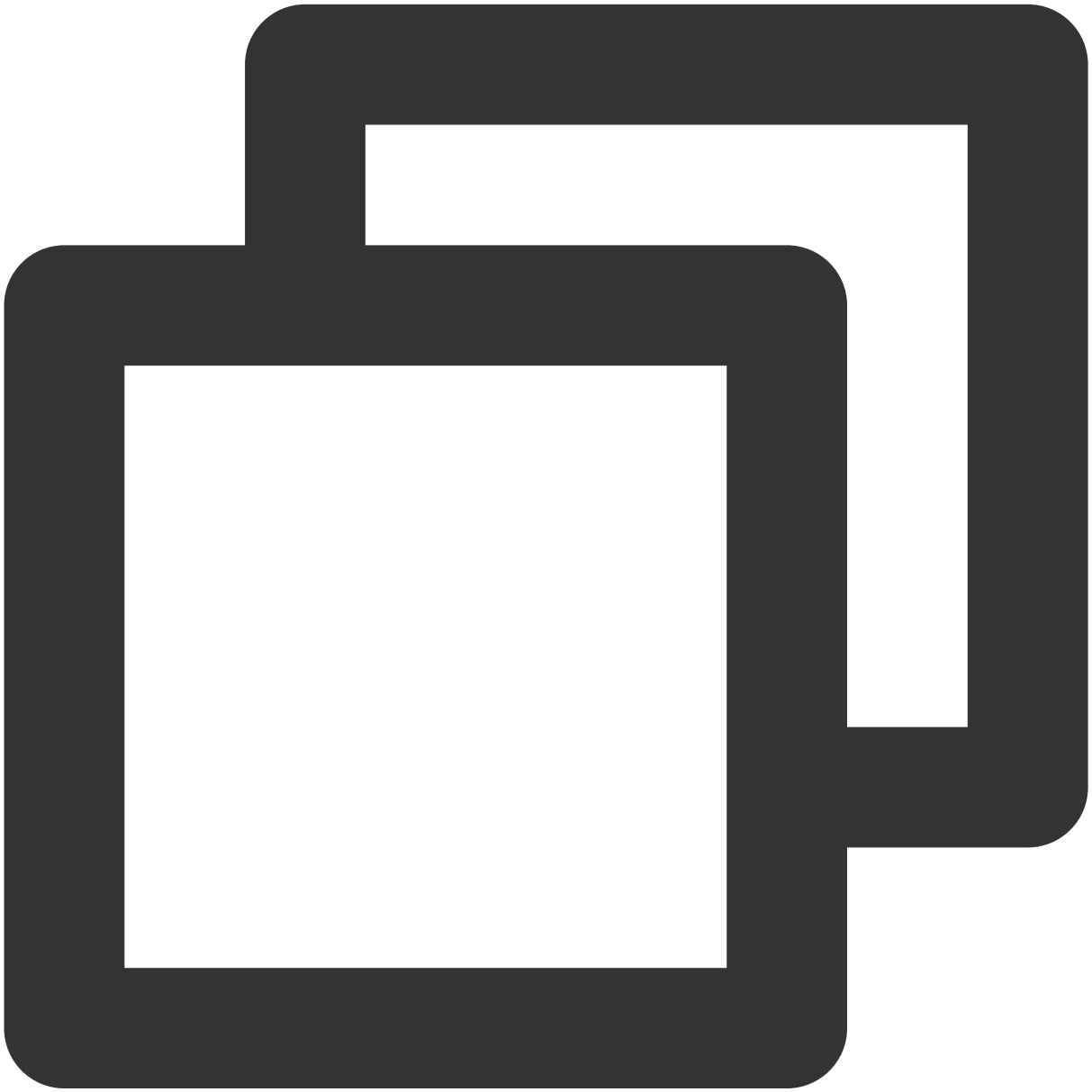
API declaration

```
/**
 Sync DNS API (general)
 @param domain Domain
 @return Array of found IPs. The `[0,0]` array will be returned if timeout (1s) occ
*/
- (NSArray *) WGGetHostByName:(NSString *) domain;

/**
 Batch sync DNS API (general)
 @param domains Domain array
 @return Dictionary of found IPs
```

```
 */
- (NSDictionary *) WGGetHostsByNames:(NSArray *) domains;
```

Sample code

Sample API call:



```
// Query a single domain
NSArray *ipsArray = [[MSDKDns sharedInstance] WGGetHostByName: @"qq.com"];
if (ipsArray && ipsArray.count > 1) {
    NSString *ipv4 = ipsArray[0];
```

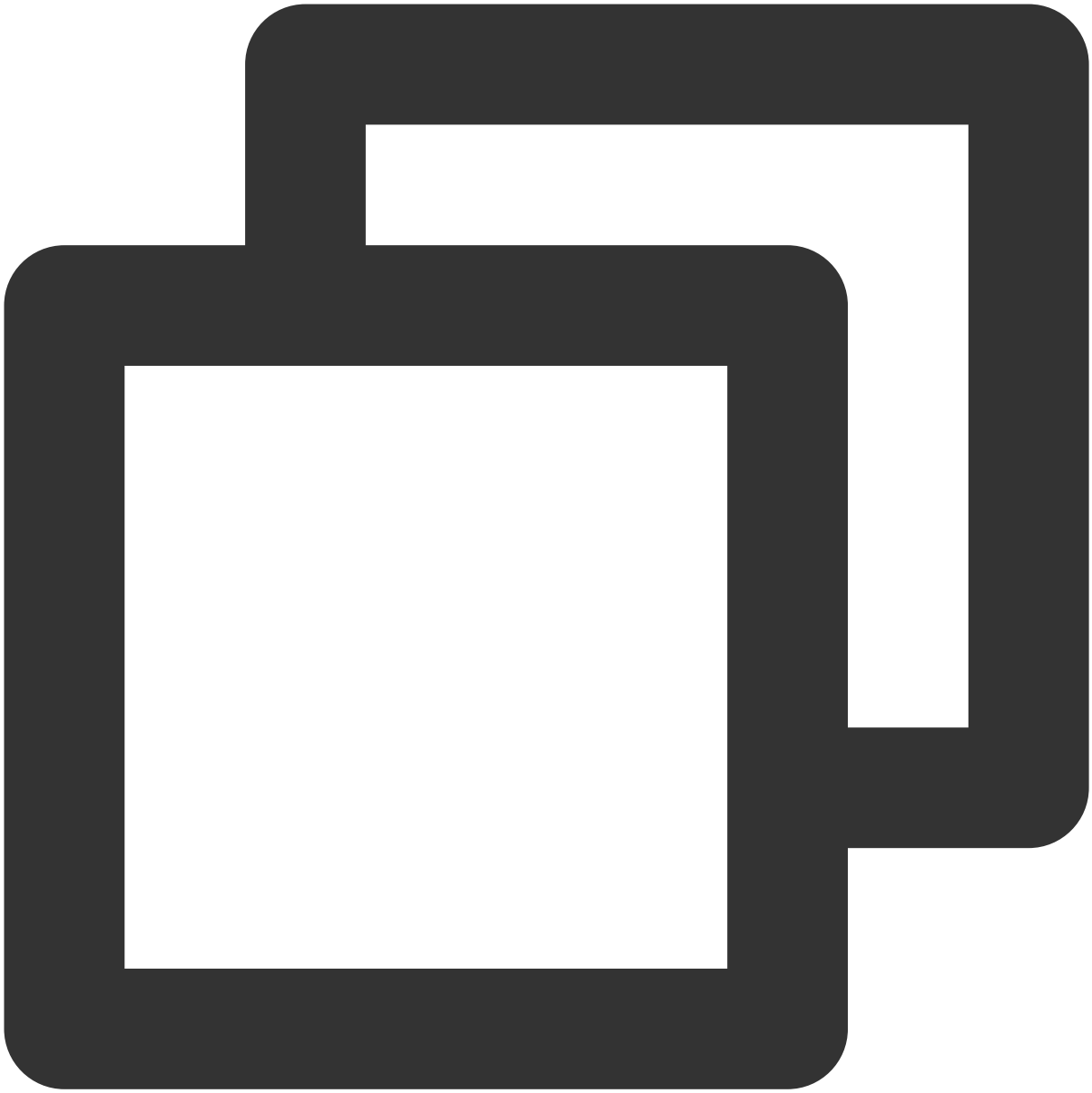
```
    NSString *ipv6 = ipsArray[1];
    if (![ipv6 isEqualToString:@"0"]) {
        // TODO: When making a URL connection by using an IPv6 address, be sure to
    } else if (![ipv4 isEqualToString:@"0"]){
        // Connect by using the IPv4 address
    } else {
        // `0,0` will be returned if an exception occurs. In this case, we recommen
    }
}

// Batch query domains
NSDictionary *ipsDict = [[MSDKDns sharedInstance] WGGetHostsByNames: @[@"qq.com", @
NSArray *ips = [ipsDict objectForKey: @"qq.com"];
if (ips && ips.count > 1) {
    NSString *ipv4 = ips[0];
    NSString *ipv6 = ips[1];
    if (![ipv6 isEqualToString:@"0"]) {
        // TODO: When making a URL connection by using an IPv6 address, be sure to
    } else if (![ipv4 isEqualToString:@"0"]){
        // Connect by using the IPv4 address
    } else {
        // `0,0` will be returned if an exception occurs. In this case, we recommen
    }
}
```

Async DNS APIs

API name

WGGetHostByNameAsync and WGGetHostsByNamesAsync

API declaration



```
/**
 Async DNS API (general)
 @param domain Domain
 @param handler Array of returned IPs. The `[0,0]` array will be returned if timeou
 */
 - (void) WGGetHostByNameAsync:(NSString *) domain returnIps:(void (^)(NSArray *ips

 /**
 Batch async DNS API (general)

 @param domains Domain array
```

```
 @param handler Dictionary of returned IPs. {"queryDomain" : [0, 0] ...} will be re
 */
- (void) WGGetHostsByNamesAsync:(NSArray *) domains returnIps:(void (^)(NSDictionar
```

Sample code

Note

You can select any call method based on your business needs.

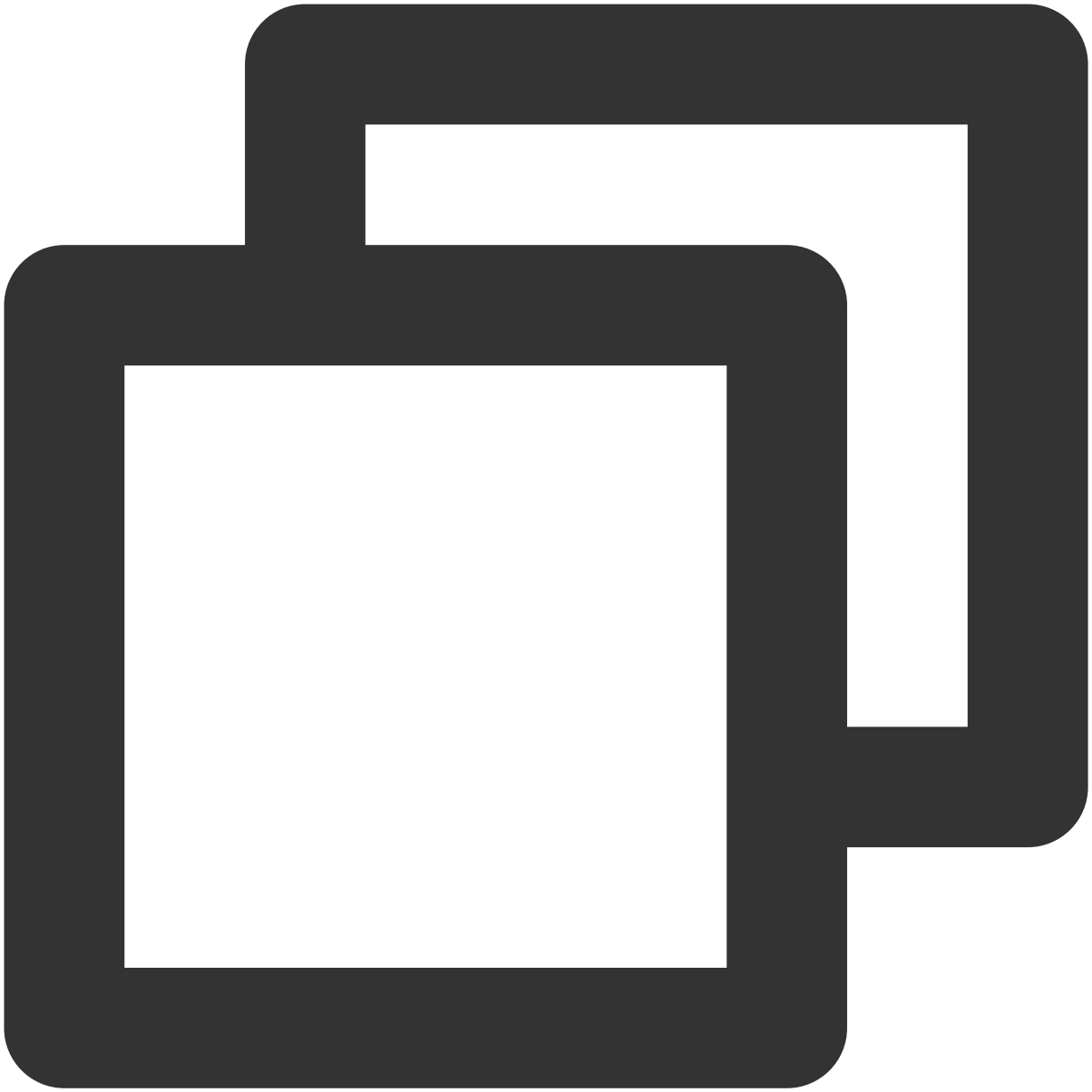
Example 1

Example 2

Wait for the entire DNS query process to complete, get the result, and make a connection.

Strengths: It is guaranteed that each request can get the returned result for subsequent connection operations.

Shortcomings: Processing through the async API is slightly more complex than that through the sync API.



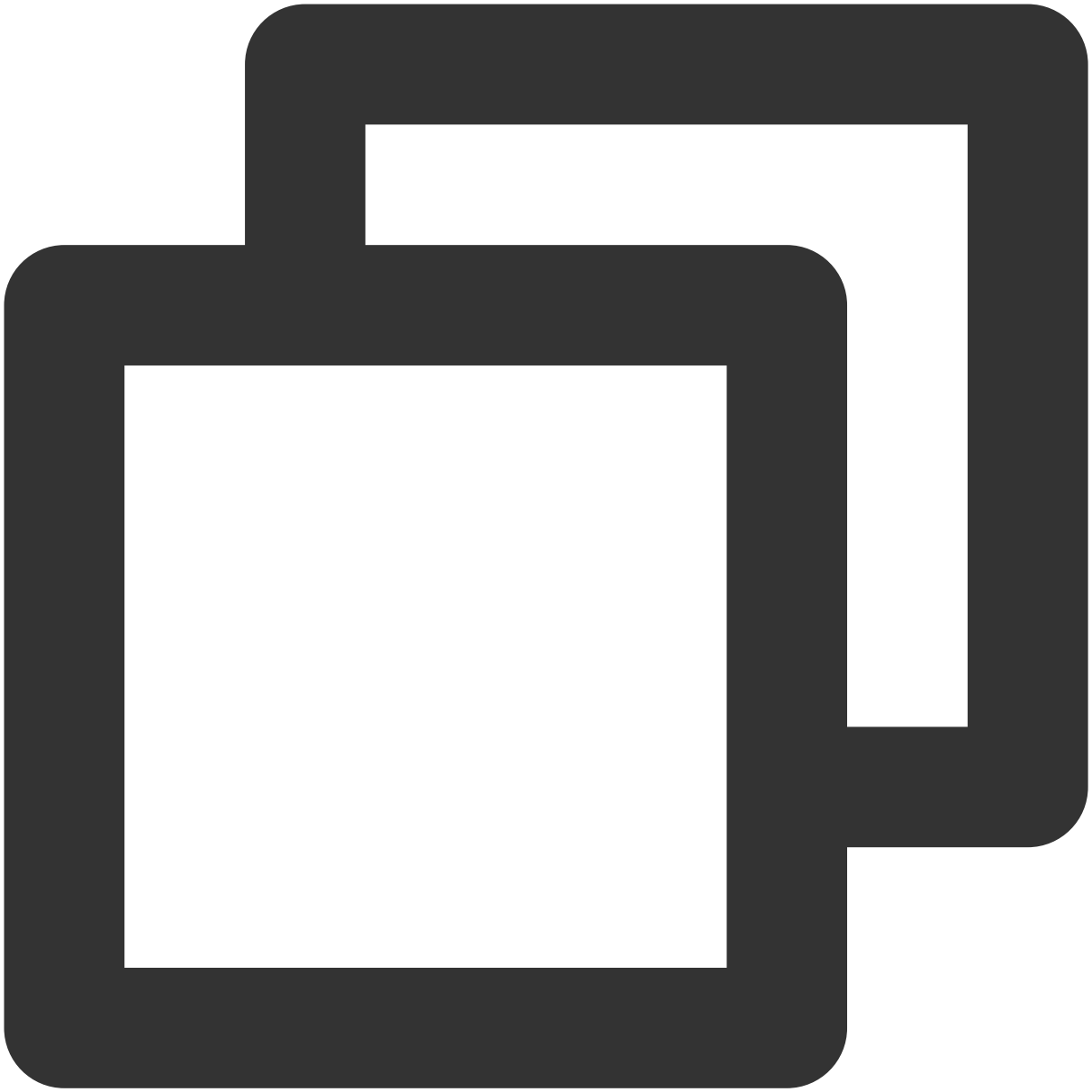
```
// Query a single domain
[[MSDKDns sharedInstance] WGGetHostByNameAsync:@"qq.com" returnIps:^(NSArray *ipsAr
    // Wait for the end of the complete DNS query process, get the result, and make
    if (ipsArray && ipsArray.count > 1) {
        NSString *ipv4 = ipsArray[0];
        NSString *ipv6 = ipsArray[1];
        if (![ipv6 isEqualToString:@"0"]) {
            // Suggestion: Use an IPv6 address first if it exists
            // TODO: When making a URL connection by using an IPv6 address, be sure
        } else if (![ipv4 isEqualToString:@"0"]){
            // Connect by using the IPv4 address
```

```
        } else {
            // `0,0` will be returned if an exception occurs. In this case, we reco
        }
    }
}];
// Batch query domains
[[MSDKDns sharedInstance] WGGetHostsByNamesAsync:@[@"qq.com", @"dnspod.cn"] returnI
    // Wait for the end of the complete DNS query process, get the result, and make
    NSArray *ips = [ipsDict objectForKey: @"qq.com"];
    if (ips && ips.count > 1) {
        NSString *ipv4 = ips[0];
        NSString *ipv6 = ips[1];
        if (![ipv6 isEqualToString:@"0"]) {
            // Suggestion: Use an IPv6 address first if it exists
            // TODO: When making a URL connection by using an IPv6 address, be sure
        } else if (![ipv4 isEqualToString:@"0"]){
            // Connect by using the IPv4 address
        } else {
            // `0,0` will be returned if an exception occurs. In this case, we reco
        }
    }
}];
```

You can directly get the cached result with no need to wait. If there is no cache, `result` will be `nil`

Strengths: Businesses that have demanding requirements for the DNS time don't need to wait and can directly get the cached result to perform subsequent connection operations, unlike the sync API where the DNS query may take more than 100 milliseconds.

Shortcomings: The result of the first request will be `nil`, and you need to add the processing logic.



```
__block NSArray* result;
[[MSDKDns sharedInstance] WGGetHostByNameAsync:domain returnIps:^(NSArray *ipsArray
    result = ipsArray;
}];
// You can directly get the cached result with no need to wait. If there is no cach
if (result) {
    // Get the cached result and make a connection
} else {
    // There is no cache for this request, and you can follow the original logic
}
```

Best Practices
HTTPS (Non-SNI) Scenario

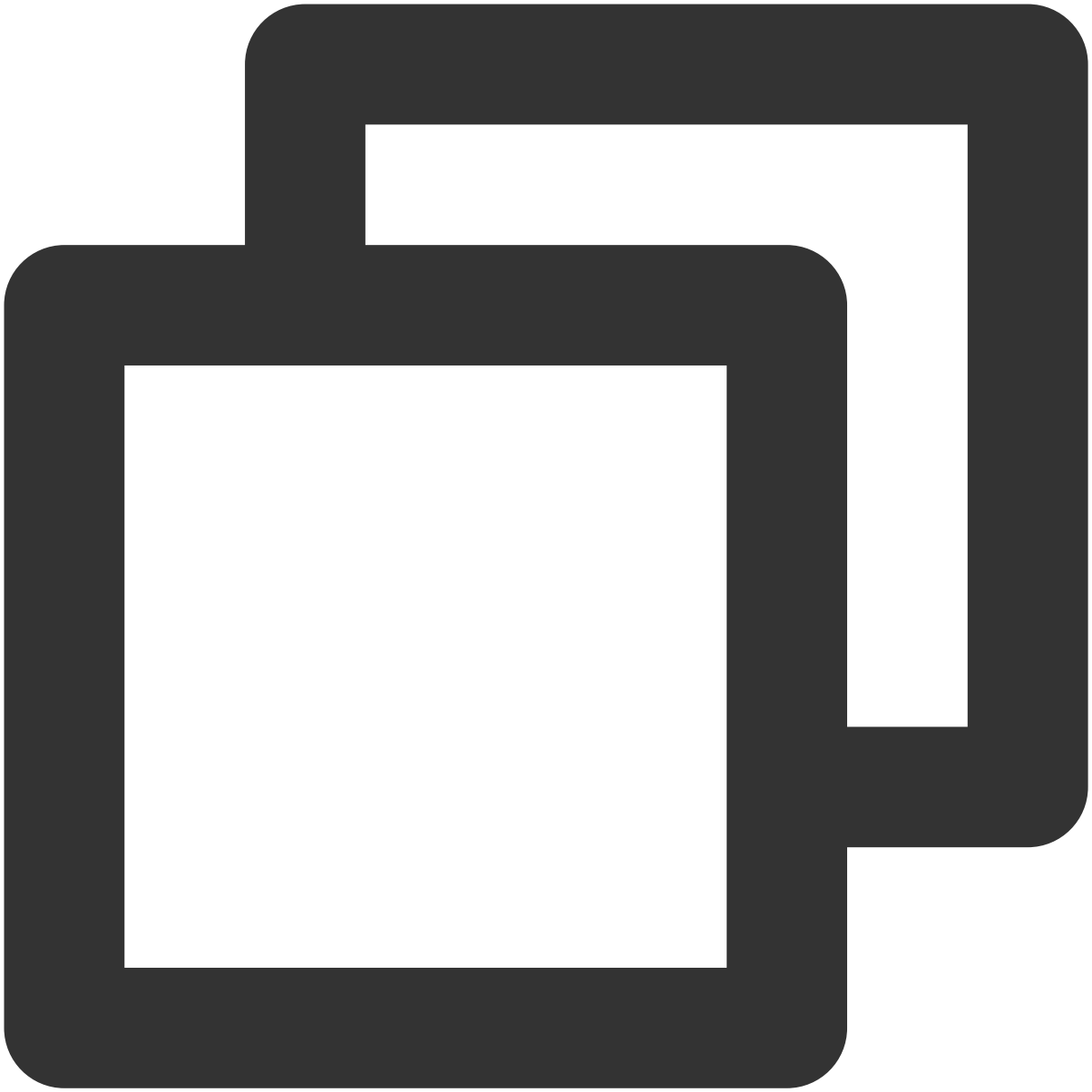
Last updated：2023-06-12 14:50:21

How It Works

Replace the IP with the original domain before verifying a certificate.

Demo

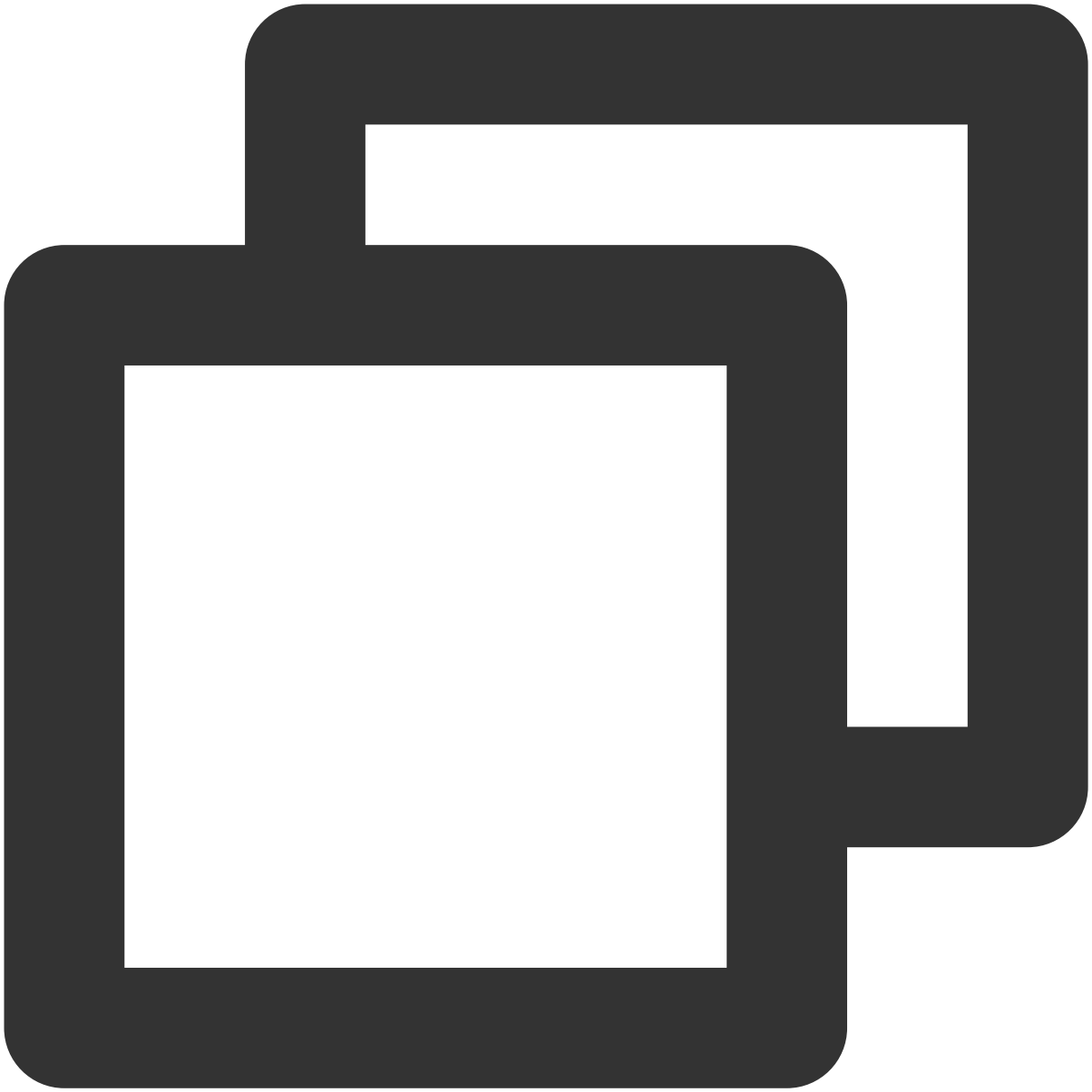
For `NSURLConnection` API



```
#pragma mark - NSURLConnectionDelegate
- (BOOL)evaluateServerTrust:(SecTrustRef)serverTrust forDomain:(NSString *)domain {

    // Create a certificate verification policy
    NSMutableArray *policies = [NSMutableArray array];
    if (domain) {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateSSL(true, (__bridg
    } else {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateBasicX509()];
    }
```

```
    // Bind the verification policy to the certificate on the server
    SecTrustSetPolicies(serverTrust, (__bridge CFArrayRef)policies);

    // Evaluate whether the current `serverTrust` is trustworthy
    // We recommend that only when `result` is `kSecTrustResultUnspecified` or `kSe
    //https://developer.apple.com/library/ios/technotes/tn2232/_index.html
    // For more information on `SecTrustResultType`, see `SecTrust.h`
    SecTrustResultType result;
    SecTrustEvaluate(serverTrust, &result);
    return (result == kSecTrustResultUnspecified || result == kSecTrustResultProcee
}

- (void)connection:(NSURLConnection *)connection willSendRequestForAuthenticationCh
    if (!challenge) {
        return;
    }

    // When HTTPDNS is used, the host in the URL is set to the IP, and you can get
    NSString *host = [[self.request allHTTPHeaderFields] objectForKey:@"host"];
    if (!host) {
        host = self.request.URL.host;
    }

    // Determine whether the challenge authentication method is `NSURLAuthenticatio
    // The default network request process will be performed if no authentication m
    if ([challenge.protectionSpace.authenticationMethod isEqualToString:NSURLAuthen
        if ([self evaluateServerTrust:challenge.protectionSpace.serverTrust forDoma

            // After authentication, you need to construct an `NSURLCredential` and
            NSURLCredential *credential = [NSURLCredential credentialForTrust:chall
            [[challenge sender] useCredential:credential forAuthenticationChallenge
        } else {
            // If authentication fails, the authentication process will be canceled
            [[challenge sender] cancelAuthenticationChallenge:challenge];
        }
    } else {

        // For other authentication methods, directly proceed with the process
        [[challenge sender] continueWithoutCredentialForAuthenticationChallenge:cha
    }
}
```

For `NSURLSession` API

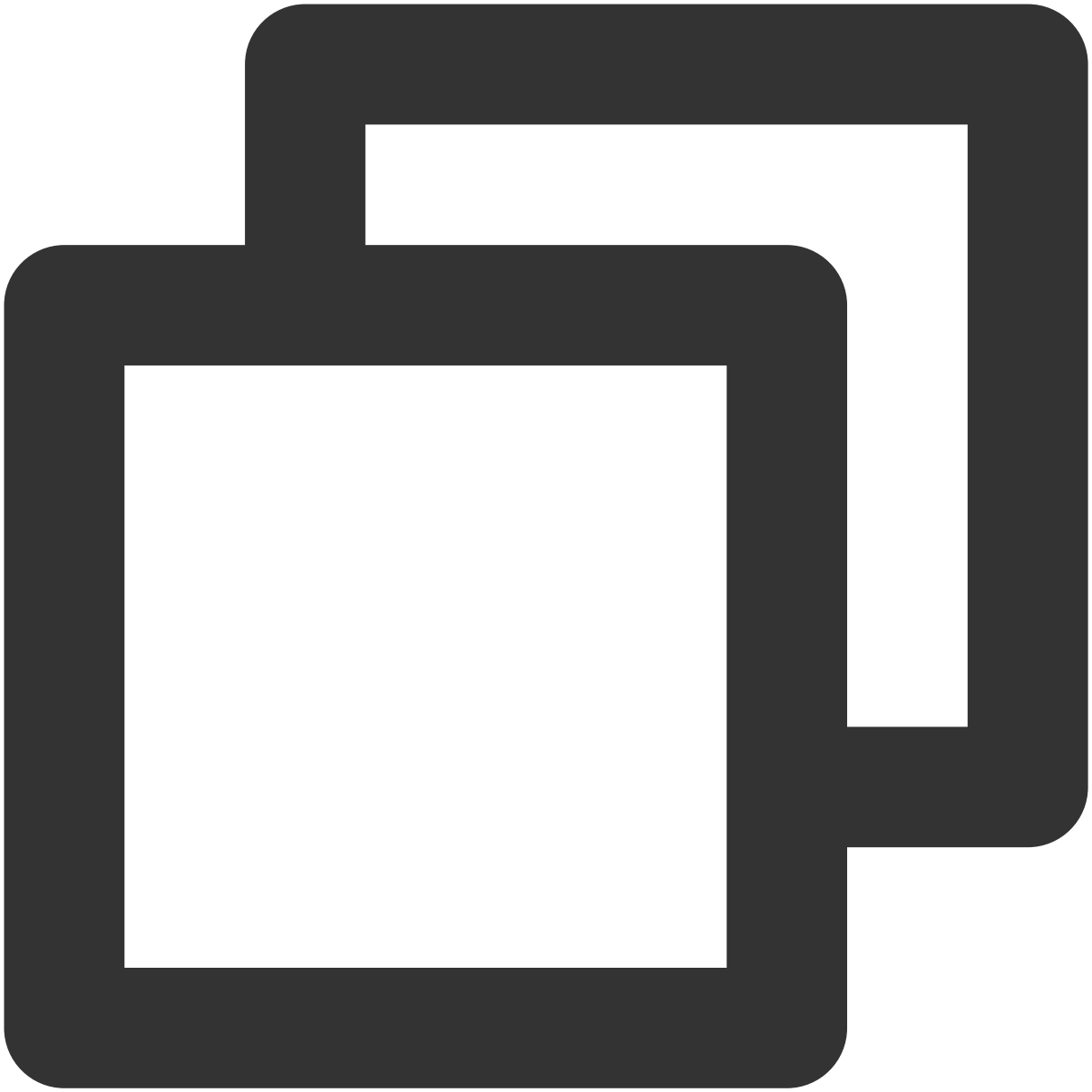


```
#pragma mark - NSURLSessionDelegate
- (BOOL)evaluateServerTrust:(SecTrustRef)serverTrust forDomain:(NSString *)domain {

    // Create a certificate verification policy
    NSMutableArray *policies = [NSMutableArray array];
    if (domain) {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateSSL(true, (__bridg
    } else {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateBasicX509()];
    }
```

```
    // Bind the verification policy to the certificate on the server
    SecTrustSetPolicies(serverTrust, (__bridge CFArrayRef)policies);

    // Evaluate whether the current `serverTrust` is trustworthy
    // We recommend that only when `result` is `kSecTrustResultUnspecified` or `kSe
    //https://developer.apple.com/library/ios/technotes/tn2232/_index.html
    // For more information on `SecTrustResultType`, see `SecTrust.h`
    SecTrustResultType result;
    SecTrustEvaluate(serverTrust, &result);

    return (result == kSecTrustResultUnspecified || result == kSecTrustResultProcee
}

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task didReceive
    if (!challenge) {
        return;
    }

    NSURLSessionAuthChallengeDisposition disposition = NSURLSessionAuthChallengePer
    NSURLCredential *credential = nil;

    // Get the original domain information
    NSString *host = [[self.request allHTTPHeaderFields] objectForKey:@"host"];
    if (!host) {
        host = self.request.URL.host;
    }
    if ([challenge.protectionSpace.authenticationMethod  isEqualToString:NSURLAuthe
        if ([self evaluateServerTrust:challenge.protectionSpace.serverTrust forDoma
            disposition = NSURLSessionAuthChallengeUseCredential;
            credential = [NSURLCredential credentialForTrust:challenge.protectionSp
        } else {
            disposition = NSURLSessionAuthChallengePerformDefaultHandling;
        }
    } else {
        disposition = NSURLSessionAuthChallengePerformDefaultHandling;
    }

    // For other challenges, directly use the default authentication scheme
    completionHandler(disposition,credential);
}
```

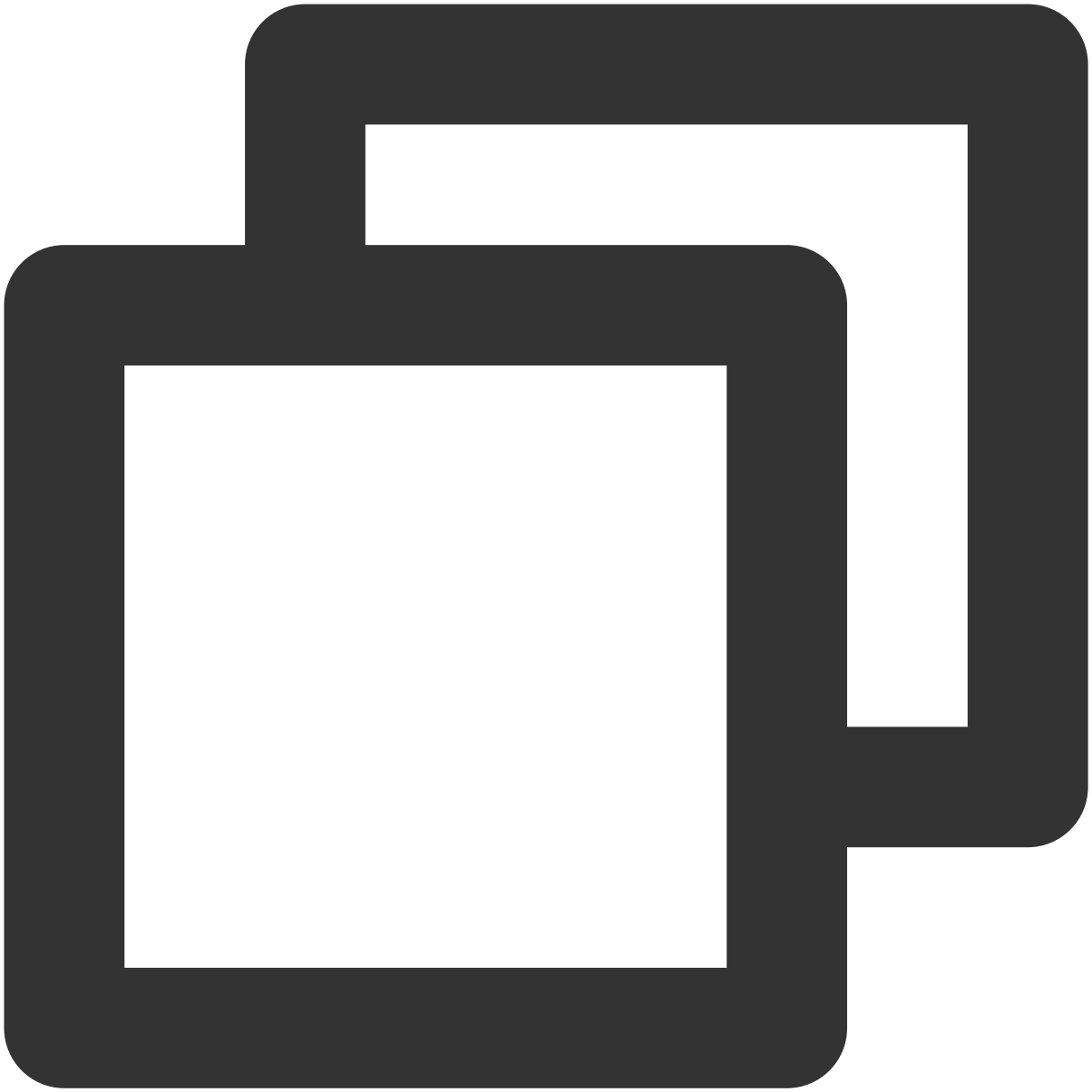
For `WWW` API of Unity

After importing the Unity project as an Xcode project, open the `Classes/Unity/` **`WWWConnection.mm`** file and modify the following code:



```
//const char* WWWDelegateClassName = "UnityWWWConnectionSelfSignedCertDelegate";
const char* WWWDelegateClassName = "UnityWWWConnectionDelegate";
```

To:



```
const char* WWWDelegateClassName = "UnityWWWConnectionSelfSignedCertDelegate";
//const char* WWWDelegateClassName = "UnityWWWConnectionDelegate";
```

HTTPS (SNI) Scenario

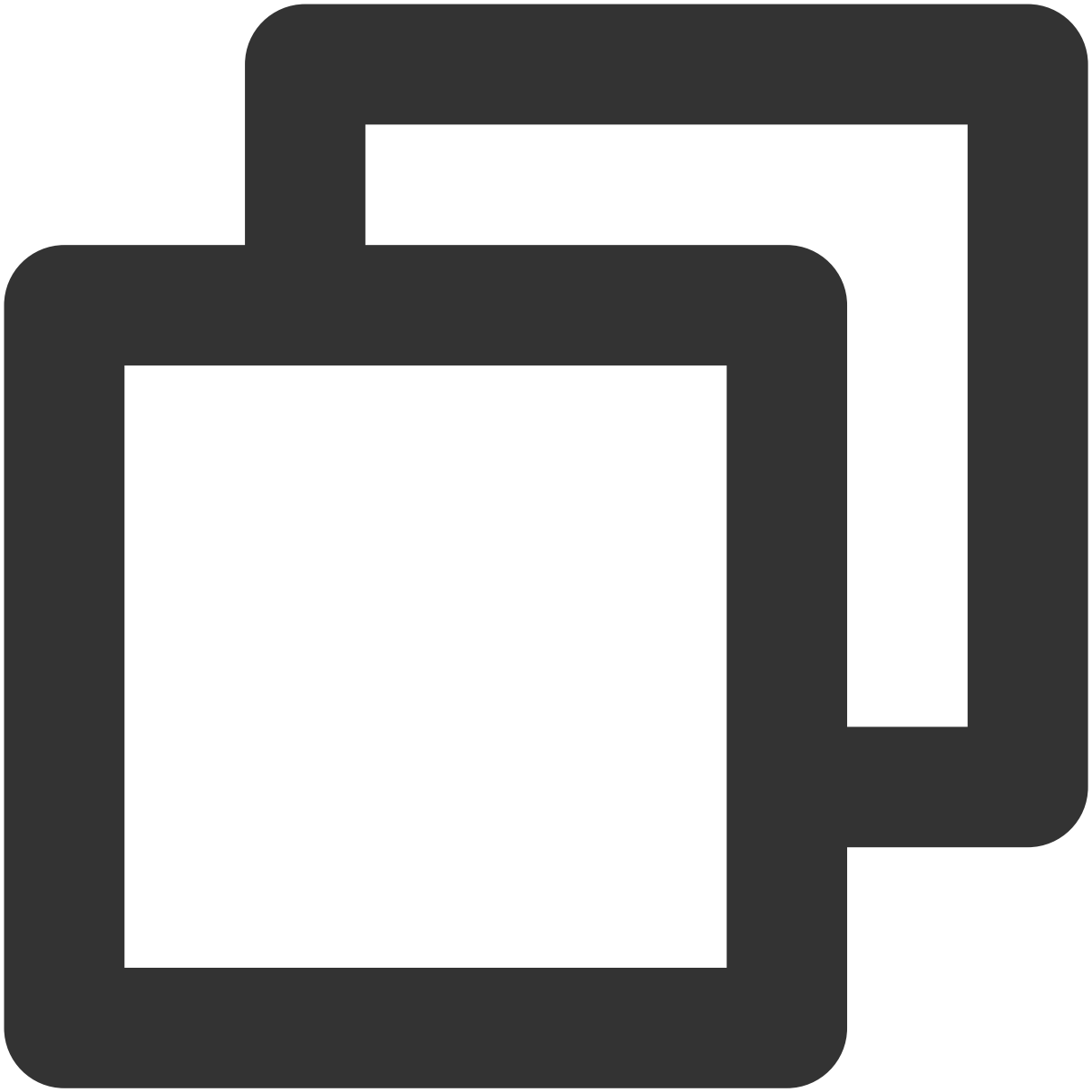
Last updated：2023-06-12 14:45:53

How It Works

Server Name Indication (SNI) is an extension to SSL/TLS that allows a server to use multiple domains and certificates. It works as follows:
Send the domain (hostname) of the site to be accessed before connecting to the server to establish an SSL connection.
The server returns an appropriate certificate according to the domain.
In the above process, when the client uses HTTPDNS to resolve a domain, the host in the request URL will be replaced with the IP resolved by HTTPDNS, so that the domain obtained by the server will be the resolved IP, and the client cannot find a matching certificate and can return only the default certificate or no certificate. As a result, an SSL/TLS handshake failure will occur.
As iOS' upper level network libraries of NSURLConnection and NSURLSession don't provide an API to configure the SNI field, you can consider using `NSURLProtocol` to intercept network requests, using `CFHTTPMessageRef` to create an `NSInputStream` instance for socket communication, and setting the value of its `kCFStreamSSLPeerName`.
Note that when you use `NSURLProtocol` to intercept a POST request made by `NSURLSession`, `HTTPBody` will be empty. There are two solutions to this:
Send a POST request by using `NSURLConnection`.
Place `HTTPBody` in the HTTP header field first and then get it from `NSURLProtocol`.

Demo

Register the `NSURLProtocol` subclass in `SNIViewController.m` of the demo before sending network requests.



```
// Register `NSURLProtocol` to intercept requests
[NSURLProtocol registerClass:[MSDKDnsHttpMessageTools class]];

// Set the SNI URL
NSString *originalUrl = @"your url";
NSURL *url = [NSURL URLWithString:originalUrl];
NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url];
NSArray *result = [[MSDKDns sharedInstance] WGGetHostByName:url.host];
NSString *ip = nil;
if (result && result.count > 1) {
    if (![result[1] isEqualToString:@"0"]) {
```

```
        ip = result[1];
    } else {
        ip = result[0];
    }
}
// Get the IP successfully through HTTPDNS, replace the URL, and set the host heade
if (ip) {
    NSRange hostFirstRange = [originalUrl rangeOfString:url.host];
    if (NSNotFound != hostFirstRange.location) {
        NSString *newUrl = [originalUrl stringByReplacingCharactersInRange:hostFirs
        request.URL = [NSURL URLWithString:newUrl];
        [request setValue:url.host forHTTPHeaderField:@"host"];
    }
}

// Sample `NSURLConnection`
self.connection = [[NSURLConnection alloc] initWithRequest:request delegate:self];
[self.connection start];

// Sample `NSURLSession`
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSessio
NSArray *protocolArray = @[ [MSDKDnsHttpMessageTools class] ];
configuration.protocolClasses = protocolArray;
NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration delega
self.task = [session dataTaskWithRequest:request];
[self.task resume];

// Note: When you use `NSURLProtocol` to intercept a POST request made by `NSURLSes
// There are two solutions:
// 1. Use `NSURLConnection` to send POST requests.
// 2. Place `HTTPBody` in the HTTP header field first and then get it from `NSURLPr
// The following mainly demonstrates the second solution
// NSString *postStr = [NSString stringWithFormat:@"param1=%@&param2=%@", @"val1",
// [_request addValue:postStr forHTTPHeaderField:@"originalBody"];
// _request.HTTPMethod = @"POST";
// NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSes
// NSArray *protocolArray = @[ [CFHttpMessageURLProtocol class] ];
// configuration.protocolClasses = protocolArray;
// NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration del
// NSURLSessionTask *task = [session dataTaskWithRequest:_request];
// [task resume];
```

Notes

You need to call the following API to set the domains to be or not to be intercepted:

```
#pragma mark - In SNI scenarios, call it only once
/**
 Set the list of domains to be intercepted in SNI scenarios
 We recommend you call the API to intercept domains only in SNI scenarios but not o

 @param hijackDomainArray List of domains to be intercepted
 */
- (void) WGSetHijackDomainArray:(NSArray *)hijackDomainArray;
```

```
/**
 Set the list of domains not to be intercepted in SNI scenarios

 @param noHijackDomainArray List of domains not to be intercepted
 */
 - (void) WGSetNoHijackDomainArray:(NSArray *)noHijackDomainArray;
```

If you set the list of domains to be intercepted, only HTTPS requests in the list will be intercepted and processed, while other domains will not.

If you set the list of domains not to be intercepted, HTTPS requests in the list will not be intercepted and processed.

Note

We recommend you use `WGSetHijackDomainArray` to intercept domains only in SNI scenarios but not other scenarios.

Unity

Last updated：2023-06-12 14:57:35

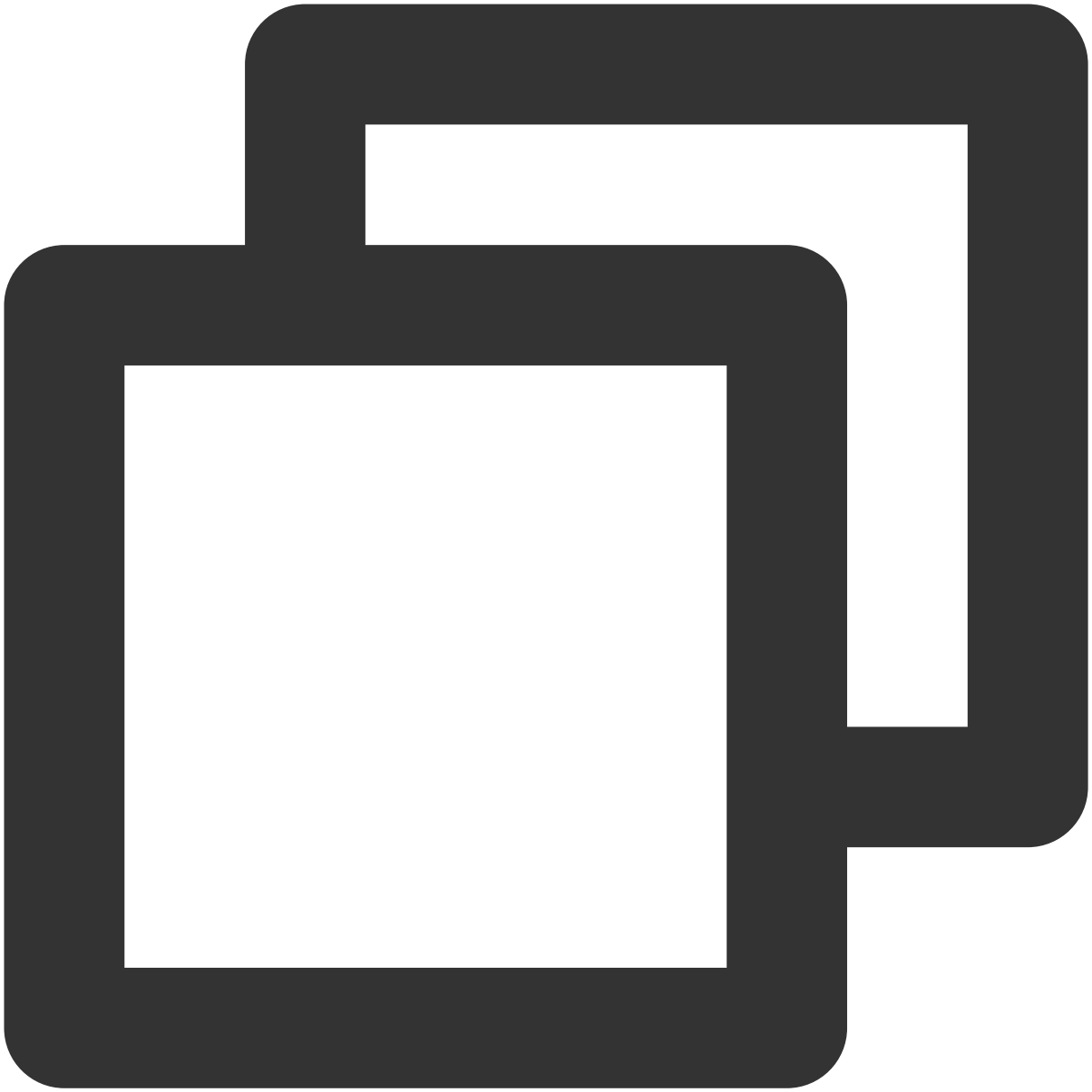
Directions

1. Copy the `HttpDns.cs` file in `HTTPDNSUnityDemo/Assets/Plugins/Scripts` to the corresponding `Assets/Plugins/Scripts` path of Unity.
2. For the part that requires DNS query, call the `HttpDns.GetAddrByName(string domain)` or `HttpDns.GetAddrByNameAsync(string domain)` method.

If you use the sync API `HttpDns.GetAddrByName`, directly call it.

If you use the async API `HttpDns.GetAddrByNameAsync`, you need to set the callback function `onDnsNotify(string ipString)`, which you can rename.

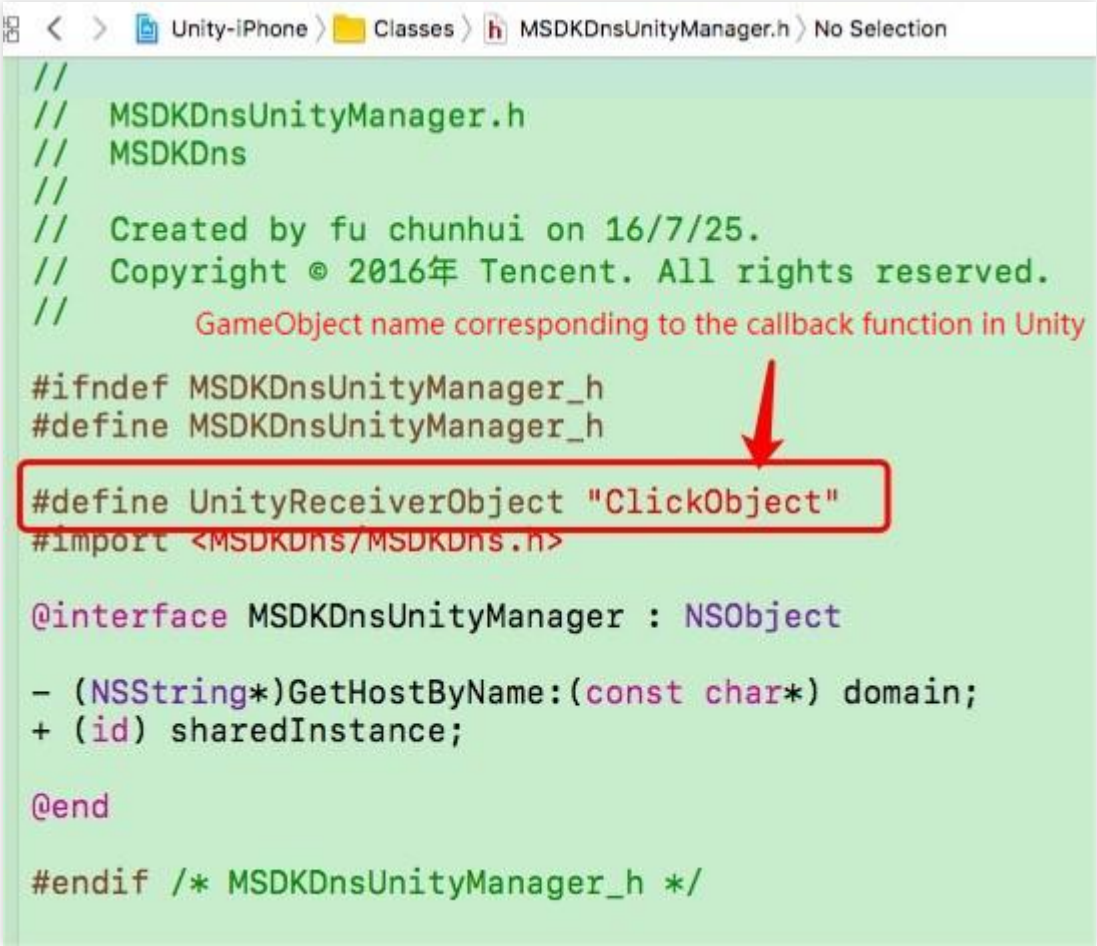
We also recommend you add the following processing code:



```
string[] sArray=ipString.Split(new char[] {';'});
if (sArray != null && sArray.Length > 1) {
 if (!sArray[1].Equals("0")) {
  // Suggestion: Use an IPv6 address first if it exists
  // TODO: When making a URL connection by using an IPv6 address, be sure to enclos
 } else if(!sArray [0].Equals ("0")) {
  // Connect by using the IPv4 address
 } else {
  // `0,0` will be returned if an exception occurs. In this case, we recommend you
  HttpDns.GetAddrByName(domainStr);
 }
```

```
}
```

3. After packaging the Unity project into an Xcode project, import the required dependency libraries.
4. Import the `MSDKDnsUnityManager.h` and `MSDKDnsUnityManager.mm` files in `HTTPDNSUnityDemo` into the project. Note that the following parts need to be the same as the corresponding `GameObject` name and callback function name in Unity, as shown below:



```
void WGGetHostByNameAsync(const char* domain){
    [[MSDKDns sharedInstance] WGGetHostByNameAsync:[NSString stringWithUTF8String:domain] r
        ipsArray) {
        if (ipsArray && ipsArray.count > 1) {
            NSString* result = [NSString stringWithFormat:@"%@;%@",ipsArray[0], ipsArray[1]
            char* resultChar = MakeStringCopy([result UTF8String]);
            UnitySendMessage(UnityReceiverObject, "onDnsNotify", resultChar);
        } else {
            char* resultChar = (char*)"0;0";
            UnitySendMessage(UnityReceiverObject, "onDnsNotify", resultChar);
        }
    }];
}
```

Name of the corresponding callback f

WebView

Last updated：2023-06-12 14:58:27

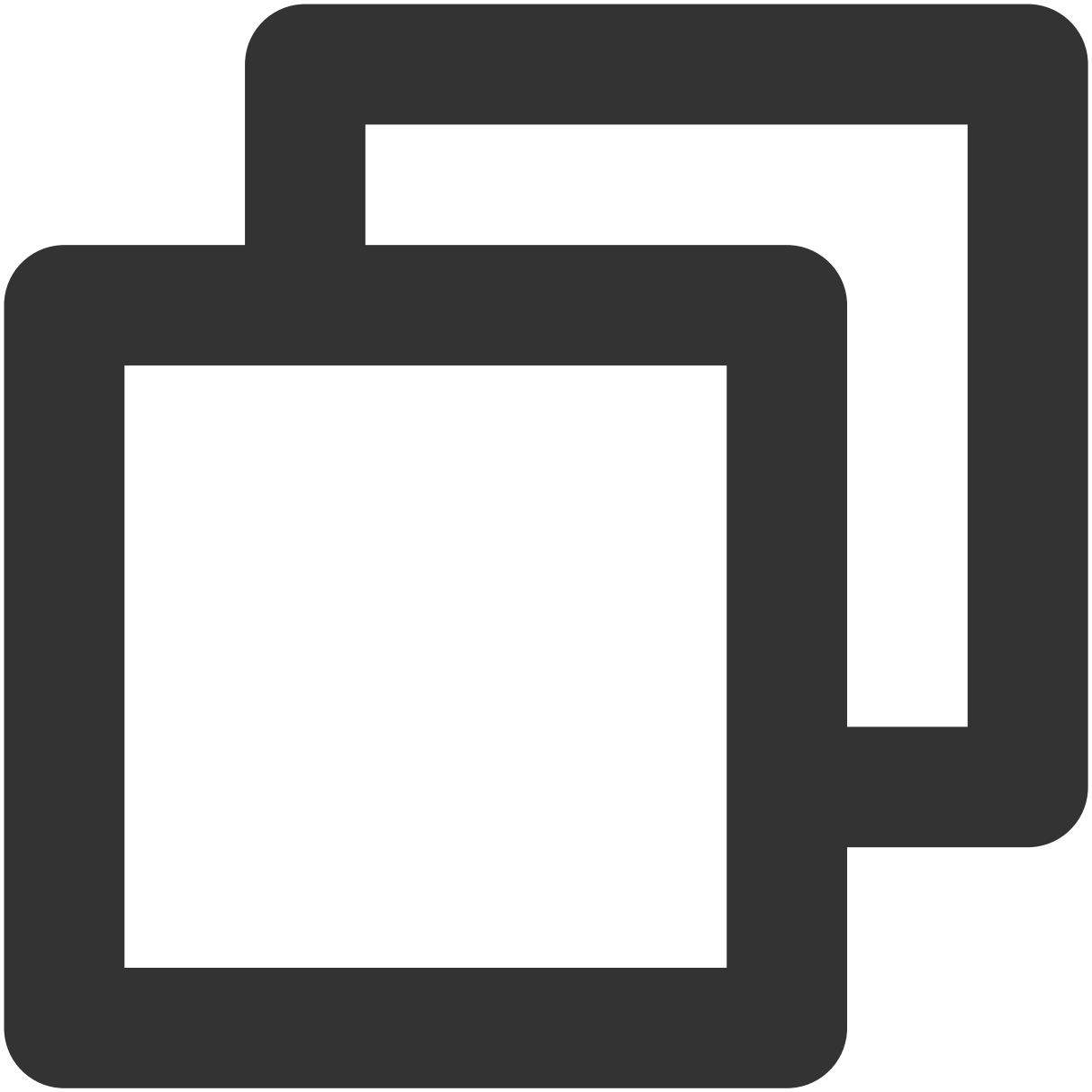
This document describes how to load elements on HTML5 pages by using HTTPDNS.

How It Works

Network request interception: Intercept a network request from WebView by using the iOS' native NSURLProtocol, filter by the filename extension of the network request URL, extract the domain from the obtained URL, make an HTTPDNS request, and finally concatenate the returned IP address into the URL of the original file network request.

Implementation Method

Perform HTTPDNS query in the `startLoading` method of the `NSURLProtocol` abstract class and replace the domain with the IP for `URLConnection` .



```
/**
 * Run the intercepted request, perform HTTPDNS query, and replace the domain with
 */
- (void)startLoading
{
      NSMutableURLRequest *newRequest;
    NSString *fileExtension = [[self.request URL] absoluteString];

      // Resolve domains for URLs of PNG, JPG, or CSS files based on your business
      if ([fileExtension containsString:@".png"] || [fileExtension containsString:@
      {
```

```
            // Modify the header information of the sync/async request
            newRequest = [[H5ContentURLProtocol convertToNewRequest:self.request isSy
        } else {
            newRequest = [self.request mutableCopy];
        }

    [NSURLProtocol setProperty:@YES forKey:@"MyURLProtocolHandledKey" inRequest:new

        self.connection = [NSURLConnection connectionWithRequest:newRequest delegate:
}
```

SDK for Android
Change History

Last updated：2023-09-05 09:53:54

v4.6.0 (August 28, 2023)

Added support for persistent connections by HTTP DNS requests.
Optimized batch resolution.
Optimized features.

v4.5.0 (July 6, 2023)

Added support for scheduling dnsip (HTTPDNS service IP) inside the SDK, without user configuration required.
Supported ECS IP configuration.
Optimized the package size.
Discontinued the Beacon data reporting service.

v4.4.0 (May 23, 2023)

Added support for data reporting and statistical analysis, which can be used via the **Query monitoring** feature provided by the console. The original data reporting service (Beacon) will be discontinued. Please switch the service as soon as possible.
Added an SDK dedicated for Tencent Cloud International.
Optimized features.

v4.3.0 (September 6, 2022)

Adjusted the resolution logic to allow the use of expired cache.
Added the local storage capability.
Optimized features.

v4.2.0 (August 15, 2022)

Added the IP address ranking feature.
Optimized features.

v4.1.0 (July 12, 2022)

Added the automatic cache update feature.
Implemented latency optimization.

v4.0.1 (May 10, 2022)

Optimized the code.

v4.0.0 (March 18, 2022)

Added parameter options for customizing the network stack (IPv4 or IPv6) for resolution.
Provided the DNS prefetch capability and the feature of disabling the return of the local DNS result.

v3.9.0 (February 23, 2022)

Optimized the code.

v3.8.0 (December 29, 2021)

Added the log service.
Optimized the redundancy logic.

v3.7.0 (June 25, 2021)

Provided APIs for asynchronously resolving domain names to IPv4 and IPv6 addresses.
Removed SSID information collection.

v3.6.0 (May 25, 2021)

Added support for batch queries and multiple encryption methods (AES, HTTPS, and DES).

v3.5.0 (April 16, 2021)

Optimized features.

v3.3.0 (January 5, 2021)

Optimized the code.

v3.2.5 (October 22, 2020)

Fixed bugs.

v3.2.4 (June 22, 2020)

Fixed bugs.

Integration

Last updated：2023-09-05 09:54:12

Overview

As a general solution to DNS optimization in the mobile internet era, HTTPDNS mainly addresses the following problems:

Local DNS hijacking/failures

Inaccurate local DNS scheduling

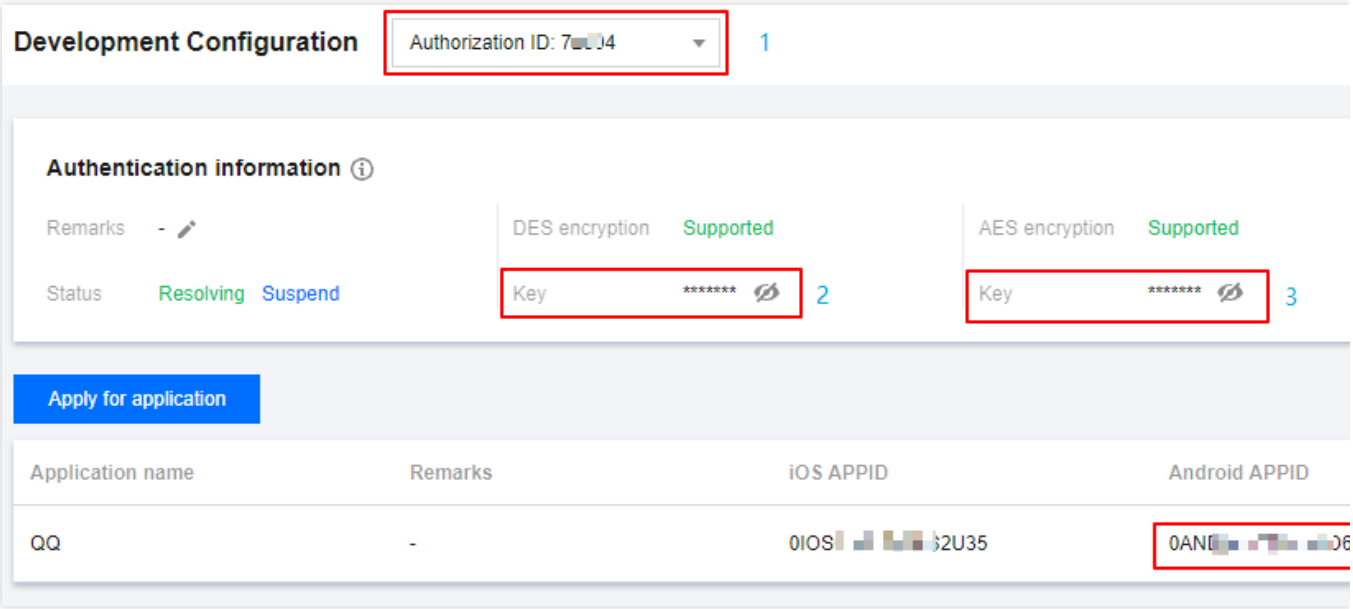
The HTTPDNS SDK for Android mainly provides DNS and cache management capabilities based on HTTPDNS:

When the SDK resolves a domain, it first uses HTTPDNS to get the DNS query result. In extreme cases, when HTTPDNS is unavailable, the result provided by local DNS will be used.

The DNS query result returned by HTTPDNS will carry the TTL information, which the SDK will use to manage the cache storing the result provided by HTTPDNS.

Preparations

1. You have activated HTTPDNS as instructed in Activating HTTPDNS.
2. You have added domains to be resolved in the HTTPDNS console as instructed in Adding a Domain.
3. You have applied in the HTTPDNS console for SDK integration as instructed in SDK Activation Process.
4. After activating the service, you have been assigned the configuration information such as authorization ID, AES and DES encryption keys, and HTTPS token. You can also view them on the Development Configuration page.



Configuration information required for using the SDK for Android:

Authorization ID: The unique ID of a development configuration used in HTTPDNS, namely, the `dnsId` parameter in the SDK used for DNS authentication.

DES encryption key: The `dnsKey` parameter in the SDK, which you need to pass in when using the DES encryption method.

AES encryption key: The `dnsKey` parameter in the SDK, which you need to pass in when using the AES encryption method.

HTTPS encryption token: The `token` parameter in the SDK, which you need to pass in when using the HTTPS encryption method.

Android APPID: The `appkey` of the SDK for Android used for authentication.

SDK Integration

HTTPDNS SDK integration

Importing the AAR package

1. Obtain the HTTPDNS SDK for Android.
2. Copy `HttpDNSLibs\\HTTPDNS_ANDROID_xxxx.aar` to the corresponding location in the application's `libs` folder.
3. Add the following to `build.gradle` in the app module:

Note

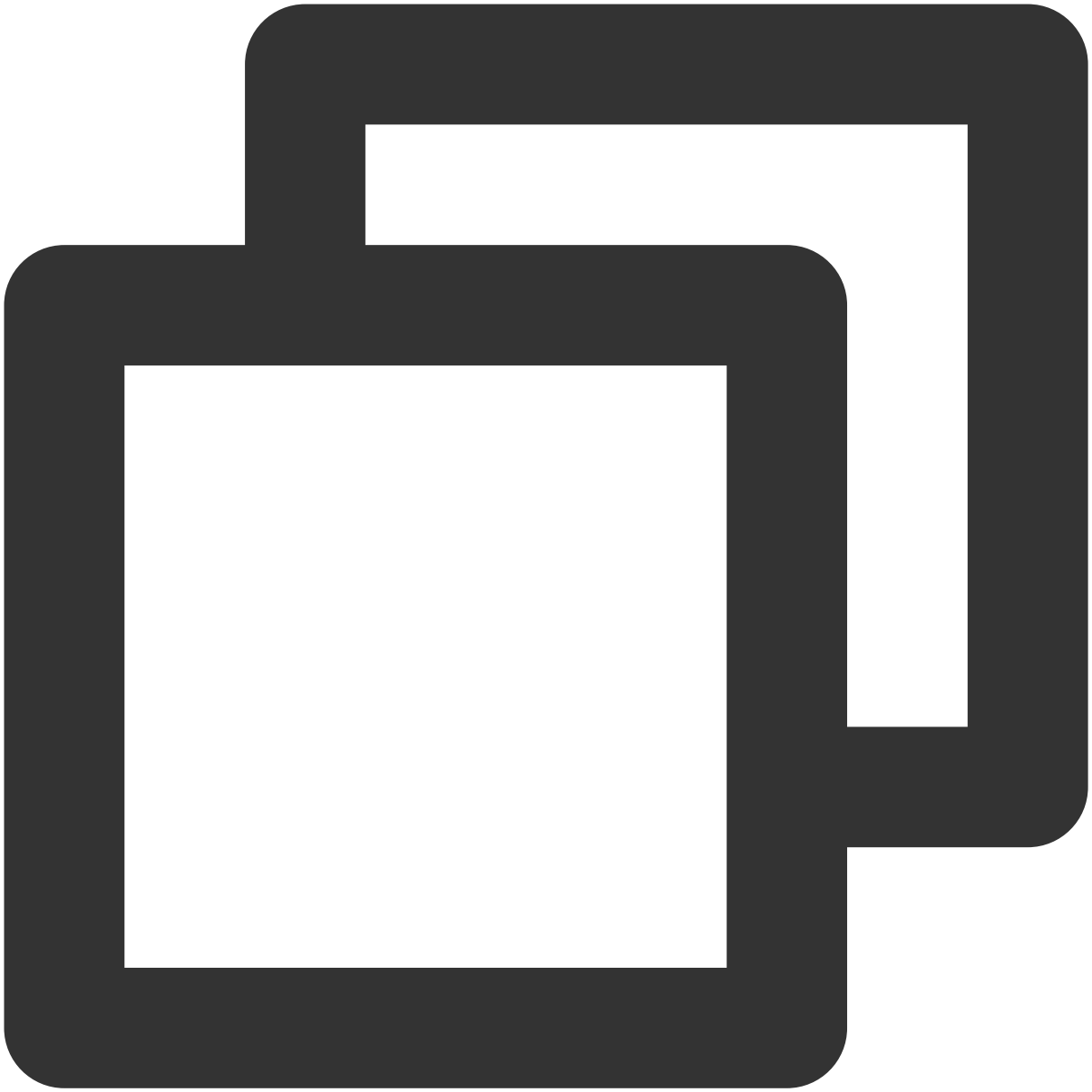
In v4.3.0, local data storage is added. You also need to add a dependency on Room. For more information, see here.

```
android {

 // ...

 repositories {
     flatDir {
         dirs 'libs'
     }
 }
}
```

```
dependencies {

 // ...

 implementation(name: 'HTTPDNS_Android_xxxx', ext: 'aar')

 // In v4.3.0, local data storage is added. You also need to add a dependency on Ro
 implementation "androidx.room:room-rxjava2:2.2.0"
}
```

Downloading the Maven repository

1. Import dependencies via a POM file.



```
<dependency>
  <groupId>io.github.dnspod</groupId>
  <artifactId>httpdns-sdk</artifactId>
  <version>4.4.0</version>
  <type>aar</type>
</dependency>

<!-- SDK for Tencent Cloud International -->
<dependency>
  <groupId>io.github.dnspod</groupId>
  <artifactId>httpdns-sdk</artifactId>
```

```
    <version>4.4.0-intl</version>
    <type>aar</type>
  </dependency>
```

Note

HTTPDNS SDK v4.4.0 supports a version dedicated for Tencent Cloud International. If you want to use the version for Tencent Cloud International, obtain the initialization configuration from the HTTPDNS international console. See Access Documentation for details.

Permission configuration

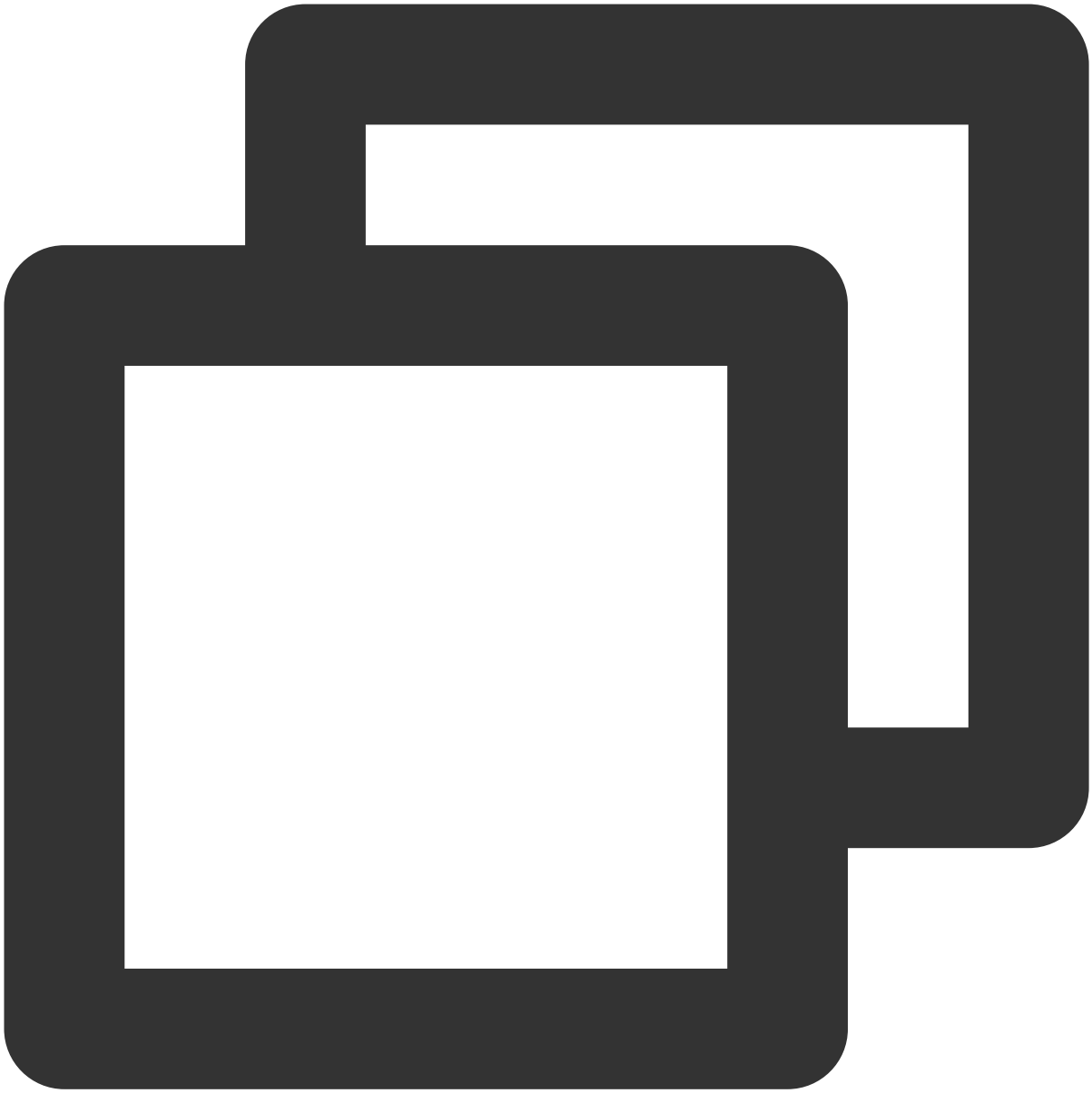
```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />

<!-- Optional and used to get the mobile phone's IMEI for data reporting -->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Network security configuration

If targetSdkVersion of the application is 28 (Android 9.0) or later, HTTP network requests are not allowed by default. For more information, see Opt out of cleartext traffic.

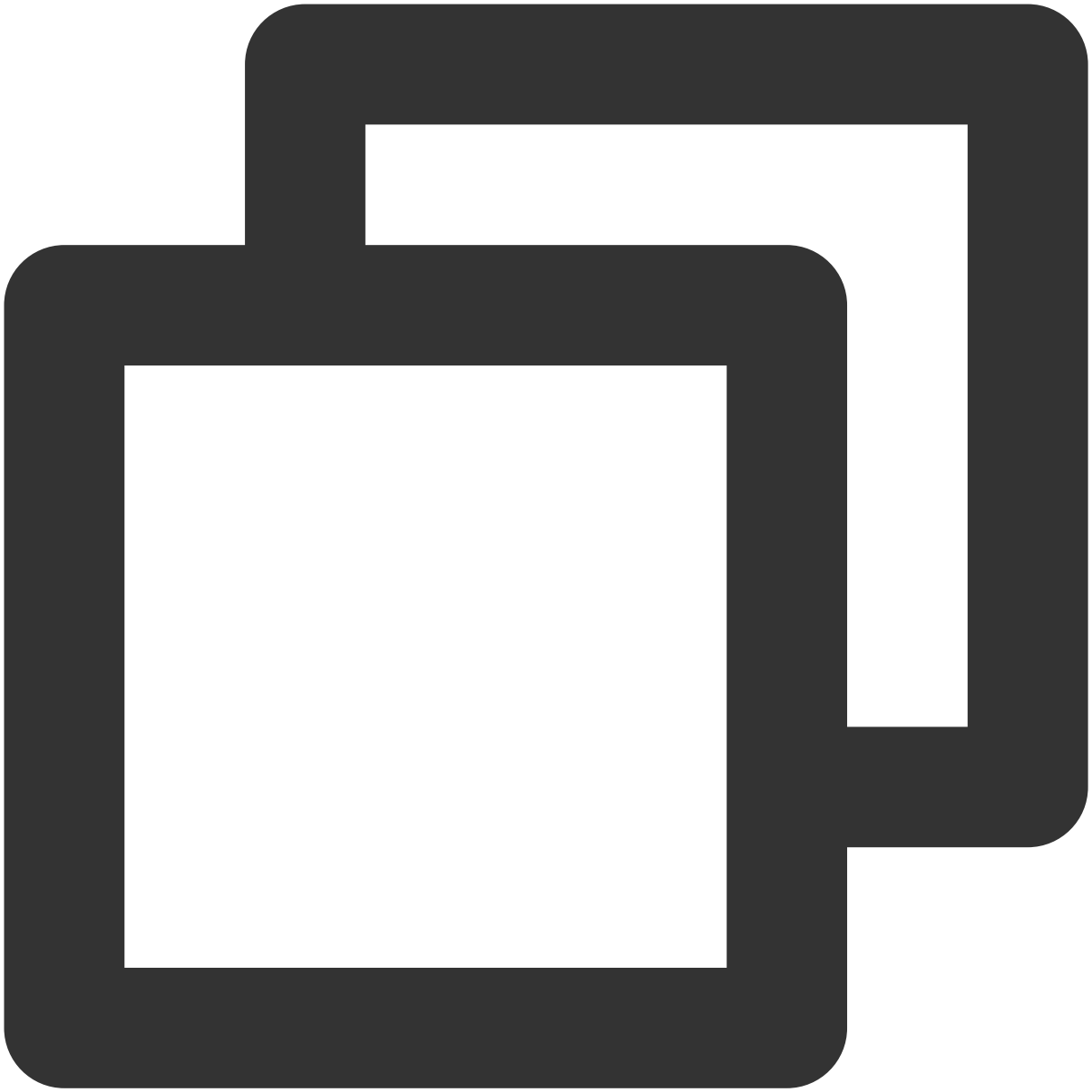
In this case, you need to add the IP used by the HTTPDNS request to the domain allowlist. The configuration is as follows:

Configure in the `AndroidManifest` file.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:networkSecurityConfig="@xml/network_security_config"
                 ... >
      ...
  </application>
</manifest>
```

Add the `network_security_config.xml` configuration file in the XML directory.



```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="true">
      <domain includeSubdomains="false">119.29.29.98</domain>
      <domain includeSubdomains="false">119.28.28.98</domain>
  </domain-config>
</network-security-config>
```

SDK initialization

Initialization configuration (supported starting from v4.0.0)

Note

The HTTPDNS SDK provides multiple DNS optimization options, which can be combined for optimal DNS performance.

You can configure `setUseExpiredIpEnable(true)` and `setCachedIpEnable(true)` to achieve optimistic DNS cache.

The persistent cache feature is used to improve cache hit rate and shorten the time to the first meaningful paint. This feature stores the last DNS result locally and preferentially reads the local cache when an app is launched.

If the cached DNS result has expired (TTL expired), since optimistic DNS cache allows the return of IP addresses in expired DNS results, the SDK returns the IP addresses in the expired DNS result (assuming that the IP addresses are available in most cases) and initiates an async request to update the cache.

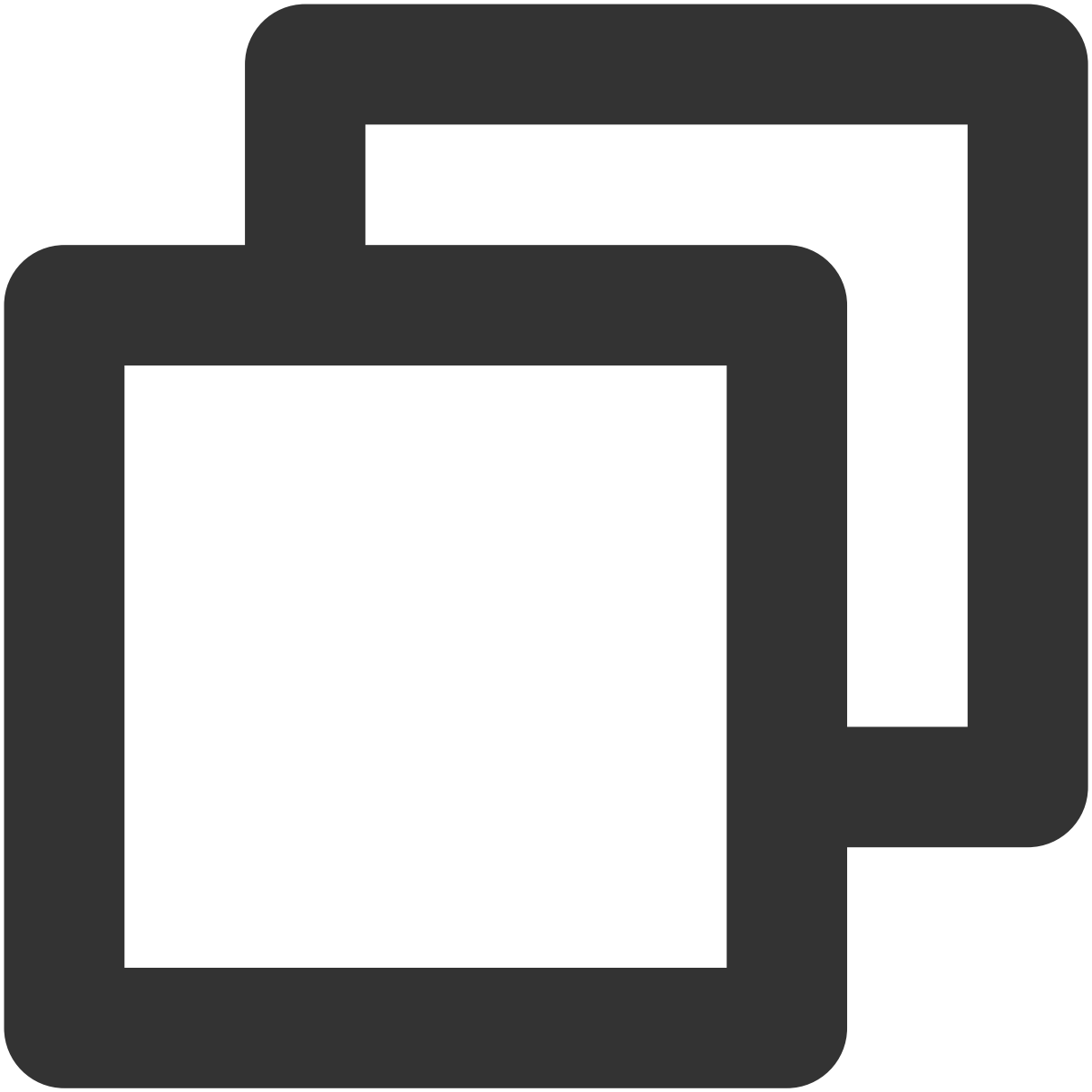
When optimistic DNS cache is enabled, cache cannot be hit (there is no cache) for the first DNS request of a domain, so `0;0` is returned, and an async DNS request is initiated to update the cache. Therefore, after enabling optimistic DNS cache, you need to ensure that local DNS will work if no cache is hit. For important domains, we recommend you enable DNS prefetch via `preLookupDomains(String... domainList)`.

If the server IP addresses are changed frequently, be sure to enable automatic cache update via `persistentCacheDomains(String... domainList)` and DNS prefetch via `preLookupDomains(String... domainList)` to ensure the accuracy of DNS results.

Before getting the service instance, you can set some service attributes by initializing the configuration, which can be passed in as configuration items when the SDK is initialized.

Note:

[Disused from v4.4.0] The SDK log reporting capability is configured in the console. For more information, see here.



```
DnsConfig dnsConfigBuilder = DnsConfig.Builder()
    // (Required) DNS ID, i.e., authorization ID, which can be applied for and obta
    .dnsId("xxx")
    // (Required) DNS key, i.e., the key corresponding to the authorization ID, whi
    .dnsKey("xxx")
    // [Disused from v4.5.0] Internally scheduled by the SDK
    // (Required) Use `119.29.29.98` (default value) if `channel` is `desHttp()` or
    .dnsIp("xxx")
    // (Optional) Channel configuration: The default value is `desHttp()` (DES encr
    .desHttp()
    // (Optional and needs to be set when the HTTPS channel is selected) Token, whi
```

```
    .token("xxx")
    // (Optional) Log granularity. If DEBUG printing is enabled, pass in "Log.VERBO
    .logLevel(Log.VERBOSE)
    // (Optional) Domains for DNS prefetch, in the format of `"baidu.com", "qq.com"
    .preLookupDomains("baidu.com", "qq.com")
    (Optional) Domains for automatic cache update, in the format of `"baidu.com", "
    .persistentCacheDomains("baidu.com", "qq.com")
    // (Optional) Items for IP address ranking, configured as a list of `IpRankItem
    .ipRankItems(ipRankItemList)
    // (Optional) Manually specify the support of network stack. Pass in 1 for IPv4
    .setCustomNetStack(3)
    // (Optional) Set whether to allow the use of expired cache. The default value
    // (Optional) Set whether to allow the use of expired cache. When this paramete
    .setUseExpiredIpEnable(true)
    // (Optional) Set whether to enable the local cache (Room). The default value i
    .setCachedIpEnable(true)
    // (Optional) Set the DNS request timeout period, which is 2000 ms by default
    .timeoutMills(2000)
    // (Optional) The ECS (EDNS-Client-Subnet) value of the DNS request. By default
    .routeIp("XXX")
    // (Optional) [Disused from v4.4.0] The SDK log reporting capability is configu
    .enableReport(true)
    // End with `build()`
    .build();

MSDKDnsResolver.getInstance().init(this, dnsConfigBuilder);
```

Initialization for earlier versions (upgrade is recommended)

The service addresses for the HTTP and HTTPS protocols are `119.29.29.98` and `119.29.29.99` respectively (use the `119.29.29.99` IP only when you select an encryption method on your own and `channel` is `Https`).

The new version of the APIs now can be accessed at `119.29.29.99/98` , and the original HTTPDNS service address `119.29.29.29` is for development and debugging purposes only without an SLA guarantee, so it is not recommended for business purposes. Migrate your business to `119.29.29.99/98` as soon as possible.

The IP is as provided in API Description.

When you integrate HTTPDNS through the SDK, if HTTPDNS does not find a DNS query result, the domain will be resolved by local DNS, and the result provided by local DNS will be returned.

SDK integration methods

There are two methods to integrate the DNS capability of the HTTPDNS SDK into the HTTP (HTTPS) network access process:

Method 1: URL replacement

Replace the host part in the URL to get a new URL for accessing the network.

In this method, the URL discards the domain information; therefore, for network requests that require domain information, there is a lot of work to do to ensure compatibility.

Method 2: DNS replacement

Incorporate the DNS capability of HTTPDNS in the network access process to replace local DNS in the original process.

In this method, you don't need to modify the URLs of the requests one by one, so there is no need to conduct extra work to ensure compatibility, but the network library used on the business side must support DNS replacement.

DNS replacement can be implemented by hooking the system's DNS function, but the function is already used in the HTTPDNS SDK, so hooking the function again may cause recursive calls or even stack overflow.

For more information on how to connect to different network libraries, see the corresponding connection documents (in the current directory) and the samples (in the `HttpDnsSample` directory).

Compatibility for URL replacement

As described above, for network requests that require domain information (usually when multiple domains are mapped to the same IP), you need to conduct extra work to ensure compatibility. The following describes how to ensure compatibility from the perspective of protocols, and the specific implementation method is subject to the implementation of the network library.

HTTP compatibility

For HTTP requests, you need to notify the server of the domain information by specifying the host field in the header. For more information on the host field, see Host.

HTTPS compatibility

HTTPS is a version of HTTP over TLS; therefore, for HTTPS requests, you also need to set the host field.

In HTTPS requests, you should perform a TLS handshake first. During the TLS handshake, the server will send its own digital certificate to you for identity verification; therefore, you also need to notify the server of the domain information during the TLS handshake. In the TLS protocol, you can specify the domain information by using the SNI extension as detailed in Server Name Indication.

Using HTTP proxy locally

Note

If an HTTP proxy is used locally, we recommend you **not use HTTPDNS** to resolve domains.

The following provides a detailed analysis for both integration methods:

URL replacement

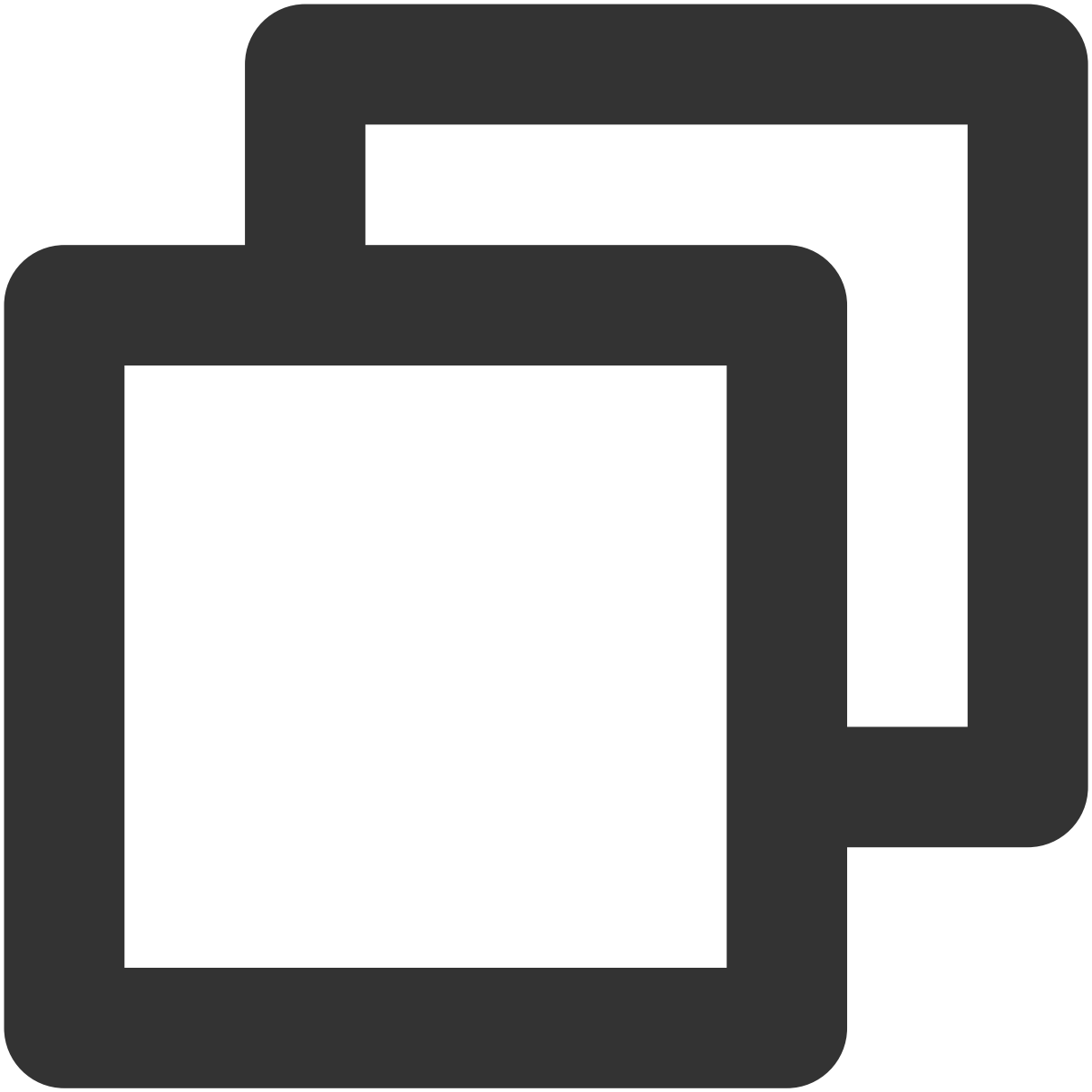
According to the HTTP/1.1 protocol, when an HTTP proxy is used, the request line will carry complete server address information on the client. For more information, see origin-form.

In this case (i.e., an HTTP proxy is used locally, the business has been connected to the HTTPDNS SDK by replacing the URL, and the host field has been set correctly), the HTTP request received by the HTTP proxy will contain the server IP information (in the request line) and domain information (in the host field). However, how the HTTP proxy will send the HTTP request to the real destination server depends on the proxy implementation, and the proxy may directly discard the set host field, causing the network request to fail.

DNS replacement

Taking the OkHttp network library as an example, after the HTTP proxy is enabled locally, OkHttp will resolve only the host of the configured HTTP proxy but not the host field in the HTTP request. In this case, it is useless to enable HTTPDNS.

Check whether an HTTP proxy is used locally with the following code:



```
val host = System.getProperty("http.proxyHost")
val port = System.getProperty("http.proxyPort")
if (null != host && null != port) {
    // An HTTP proxy is used locally
}
```

Integration verification

Log verification

When you pass in `true` for the `debug` parameter in the `init` API to filter logs with the `HTTPDNS` tag, if you see logs related to the local DNS (`ldns_ip`) and HTTPDNS (`hdns_ip`), the connection is correct.

The `ldns_ip` key indicates the DNS query results provided by the local DNS.

The `hdns_ip` key indicates the DNS query results of A records provided by HTTPDNS.

The `hdns_4a_ips` key indicates the DNS query results of AAAA records provided by HTTPDNS.

The `a_ips` key indicates the set of IPv4 addresses returned by the DNS API.

The `4a_ips` key indicates the set of IPv6 addresses returned by the DNS API.

Simulating local DNS hijacking

When you simulate hijacking local DNS, if the application works properly, HTTPDNS has been integrated successfully.

Note

Due to the caching mechanism of local DNS, when simulating local DNS for integration verification, make sure that the cache of local DNS has been cleared. You can clear the cache by restarting the server or switching the network.

During verification, be sure to compare the effects of local DNS and HTTPDNS.

Modify the hosts file on your server.

Local DNS gets the DNS query result by reading the hosts file on your server first.

You can simulate hijacking local DNS by modifying the hosts file to point the domain to an incorrect IP.

You can directly modify the hosts file on a root server.

Modify the DNS server configuration.

You can simulate hijacking local DNS by modifying the configuration of the DNS server to point it to an unavailable IP (such as another IP on the LAN).

When your server is connected to Wi-Fi, you can modify the DNS server settings by changing the IP settings to **Static** in the **Advanced Settings** option of the Wi-Fi (this operation may vary slightly by server).

Use a DNS modifier to modify the DNS server configuration (usually by tampering with the DNS packet through VPN).

Packet capture verification

The following takes accessing a network over HTTP as an example:

Note

Common mobile HTTP/HTTPS packet capture tools (such as Charles and Fiddler) capture packets through an HTTP proxy, so they are not suitable for verifying whether HTTPDNS works. For more information, see Using the HTTP proxy locally.

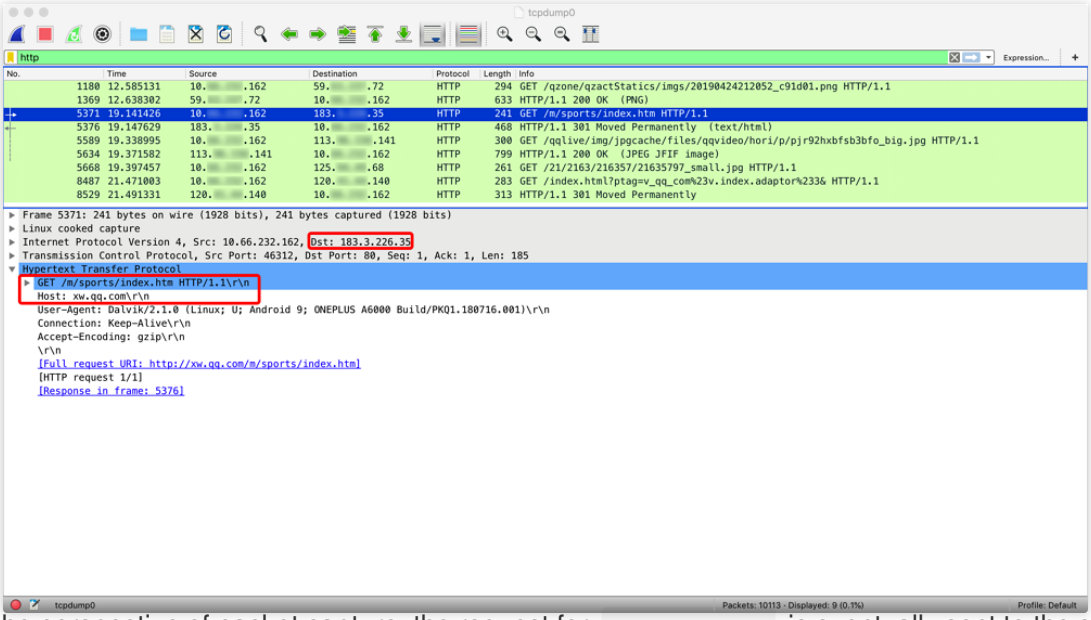
Use **tcpdump** to capture packets.

On a root server, you can use the `tcpdump` command to capture packets.

On a non-root server, the system may have a built-in debugging tool that can be used to get the packet capture result (the enabling method varies by server).

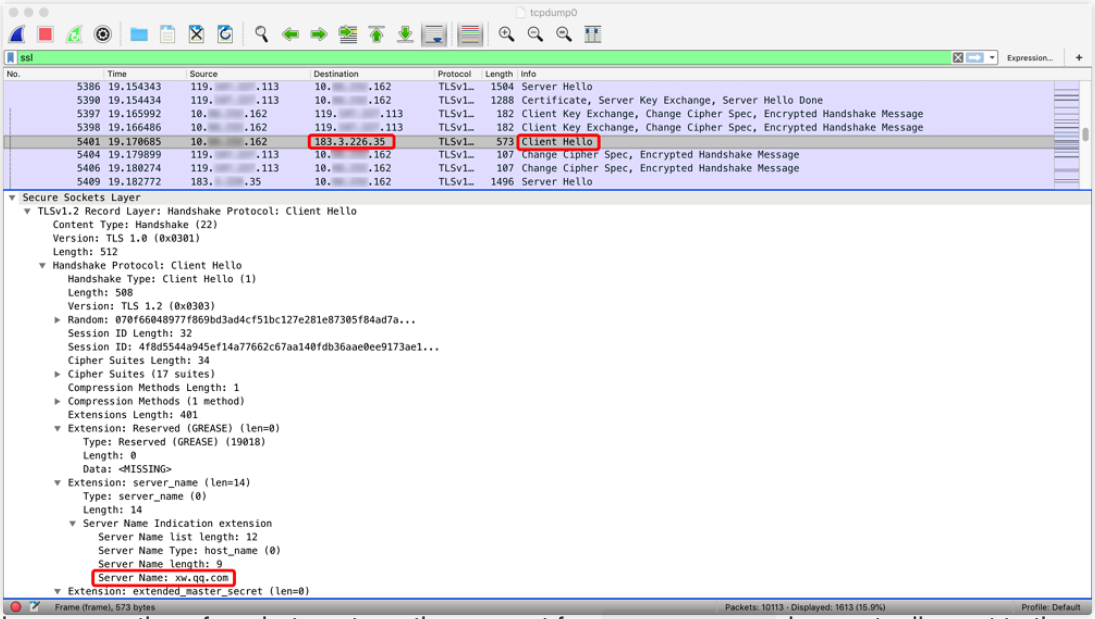
Observe the packet capture result with **WireShark**.

For HTTP requests, you can observe plaintext information and confirm whether the IP used during request sending is the one returned by the SDK by comparing the log and the specific packet capture record. The details are as shown below:



From the perspective of packet capture, the request for `xw.qq.com` is eventually sent to the server at the IP address `183.3.226.35`.

For HTTPS requests, a TLS handshake packet is actually a plaintext packet. After setting the SNI extension (see HTTPS compatibility), you can verify whether the IP used during request sending is the one returned by the SDK by comparing the log and the specific packet capture record. The details are as shown below:



From the perspective of packet capture, the request for `xw.qq.com` is eventually sent to the server at the IP address `183.3.226.35`.

Notes

`getAddrByName` is a time-consuming sync API, so it should be called on a child thread.

If the business on the client is bound to the host, such as the HTTP service of the host or the CDN service, you need to specify the host field of the HTTP header after replacing the domain in the URL with the IP returned by HTTPDNS. Take `URLConnection` as an example:

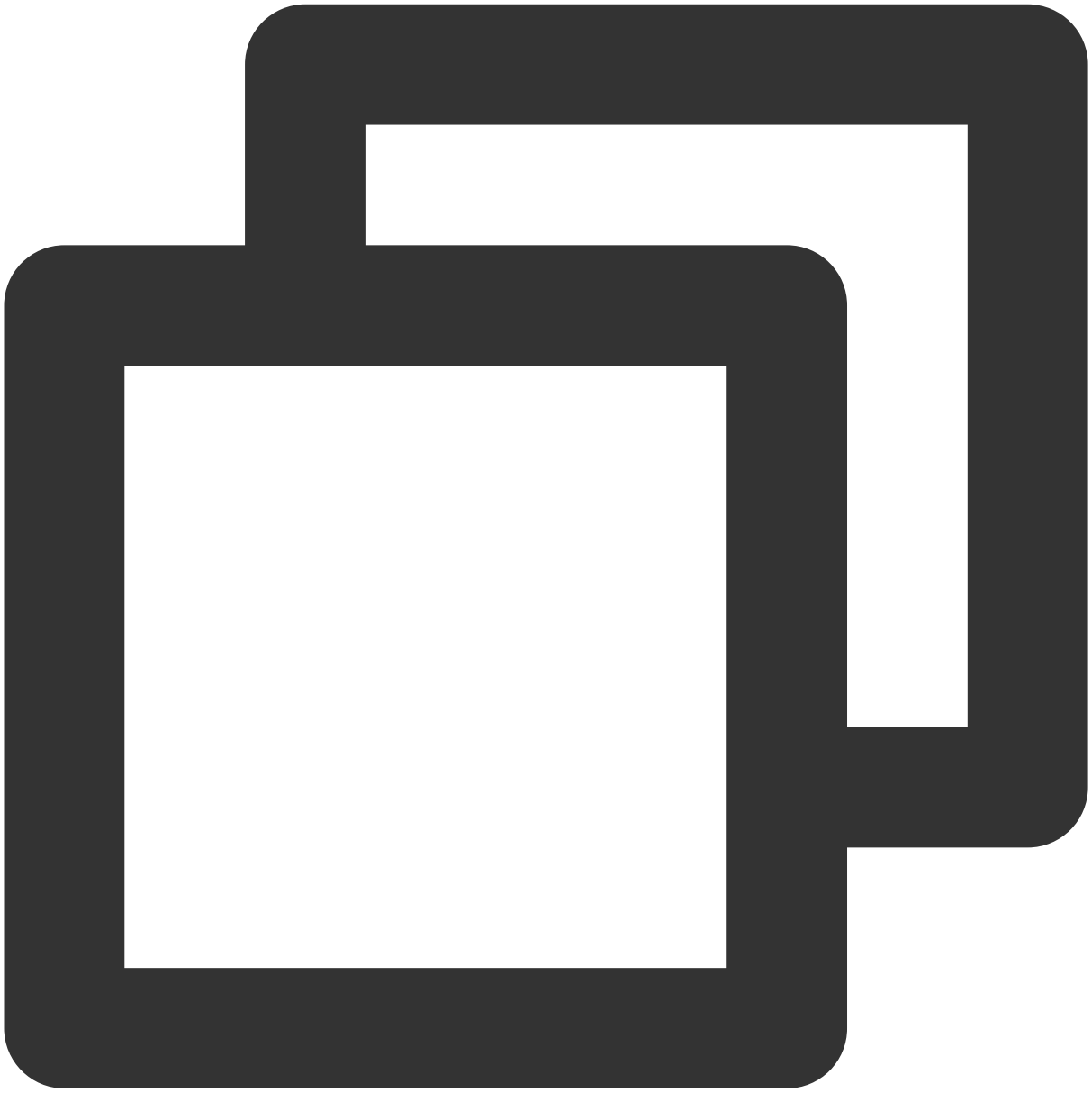
```
URL oldUrl = new URL(url);
URLConnection connection = oldUrl.openConnection();
// Get the HTTPDNS query result
String ips = MSDKDnsResolver.getInstance().getAddrByName(oldUrl.getHost());
```

```
String[] ipArr = ips.split(";");
if (2 == ipArr.length && !"0".equals(ipArr[0])) { // Get the IP successfully throug
 String ip = ipArr[0];
 String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
 connection = (HttpURLConnection) new URL(newUrl).openConnection(); // Set the host
 connection.setRequestProperty("Host", oldUrl.getHost());
}
```

Taking curl as an example, if you want to access `www.qq.com` , and the IP obtained by HTTPDNS is `192.168.0.111` , then you can access it as follows:

```
curl -H "Host:www.qq.com" http://192.168.0.111/aaa.txt
```

Check whether an HTTP proxy is used locally. If an HTTP proxy is used, we recommend you **not use HTTPDNS** to resolve domains. Example:

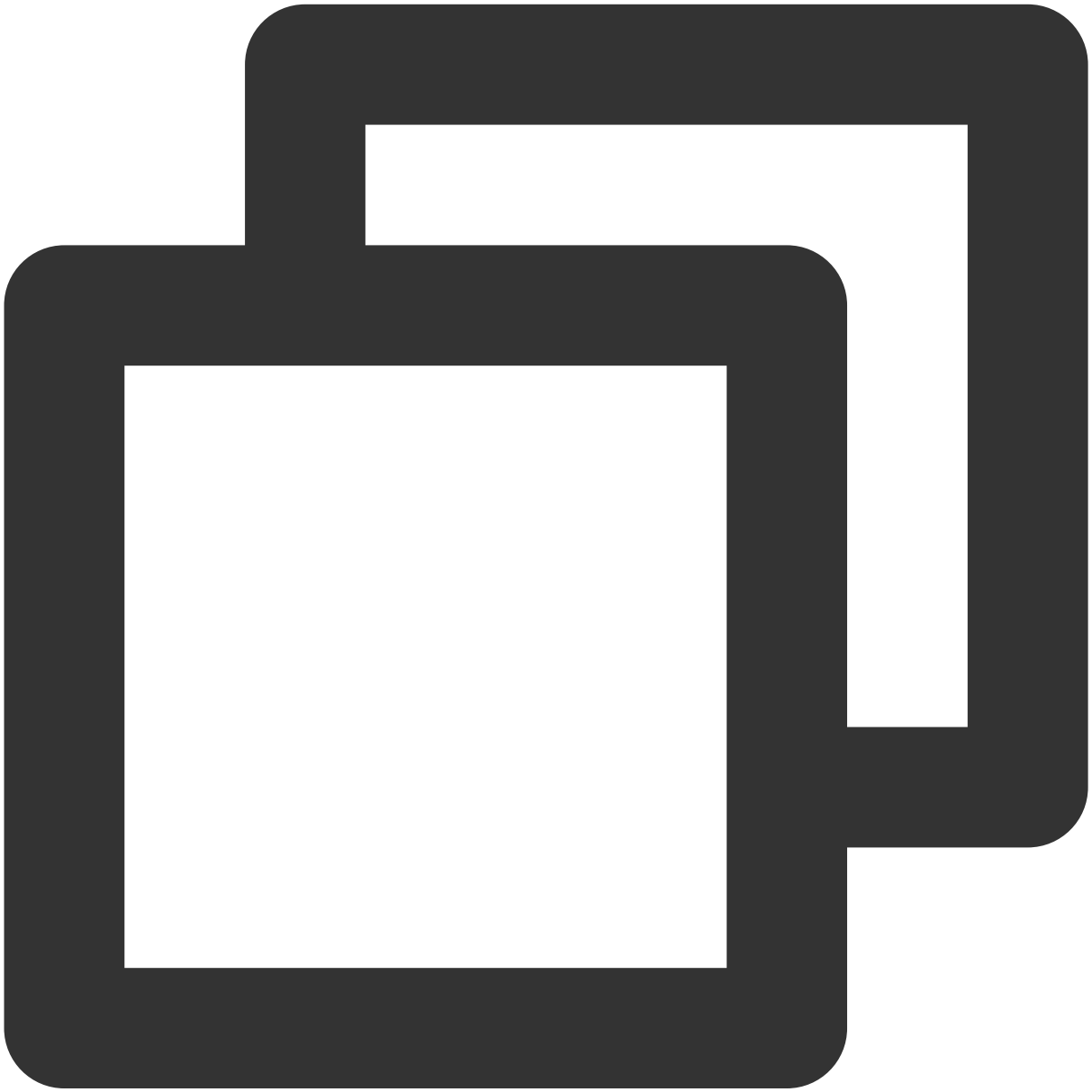


```
String host = System.getProperty("http.proxyHost");
String port= System.getProperty("http.proxyPort");
if (null != host && null != port) {
  // Local proxy mode is used
}
```

APIs

Last updated：2023-06-12 14:45:52

Sync DNS APIs



```
/**
 * HTTPDNS' sync DNS API
 * The cache is queried first. If the cache is hit, the result will be returned; ot
 * The latest DNS query result will be returned after the query is completed
```

```
 * The returned value string will be separated by semicolon, with the resolved IPv4
 * Sample response: 121.14.77.221;2402:4e00:1020:1404:0:9227:71a3:83d2
 * @param domain Domain (such as www.qq.com)
 * @return Set of resolved IP results that correspond to the domain
 */
String ips = MSDKDnsResolver.getInstance().getAddrByName(domain);

/**
 * HTTPDNS' batch sync DNS API
 * The cache is queried first. If the cache is hit, the result will be returned; ot
 * The latest DNS query result will be returned after the query is completed
 * The returned value `ipSet` is the set of resolved IP addresses
 * `ipSet.v4Ips` is the set of resolved IPv4 addresses, which may be `null`
 * `ipSet.v6Ips` is the set of resolved IPv6 addresses, which may be `null`
 * Sample response for a single domain: IpSet{v4Ips=[121.14.77.201, 121.14.77.221],
 * Sample response for multiple domains: IpSet{v4Ips=[www.baidu.com:14.215.177.39,
 * @param domain Multiple domains can be separated by comma, such as `qq.com,baidu.
 * @return Set of resolved IP results that correspond to the domain
 */
Ipset ips = MSDKDnsResolver.getInstance().getAddrsByName(domain);
```

Async DNS APIs

```
// Async callback. Note that all async requests need to be used together with async
MSDKDnsResolver.getInstance().setHttpDnsResponseObserver(new HttpDnsResponseObserve
    @Override
    public void onHttpDnsResponse(String tag, String domain, Object ipResultSemicol
        long elapse = (System.currentTimeMillis() - Long.parseLong(tag));
        String lookedUpResult = "[[getAddrByNameAsync]]:ASYNC:::" + ipResultSemicol
                ", domain:" + domain +  ", tag:" + tag +
                ", elapse:" + elapse;
    }
});
```

```
/**
 * HTTPDNS' async DNS API (to be used together with async callback)
 * The cache is queried first. If the cache is hit, the result will be returned; ot
 * The latest DNS query result will be returned asynchronously after the query is c
 * @param domain Domain (such as www.qq.com)
 */
MSDKDnsResolver.getInstance()
    .getAddrByNameAsync(hostname, String.valueOf(System.currentTimeMillis()))

/**
 * HTTPDNS' batch async DNS API (to be used together with async callbacks)
 * The cache is queried first. If the cache is hit, the result will be returned; ot
 * The latest DNS query result will be returned asynchronously after the query is c
 * @param domain Multiple domains can be separated by comma, such as `qq.com,baidu.
 */
MSDKDnsResolver.getInstance()
    .getAddrsByNameAsync(hostname, String.valueOf(System.currentTimeMillis()))
```

How to improve IPv6 usage

As described above, the result returned for resolving a single domain is in the format of `IPv4;IPv6` and that for multiple domains is `IpSet{v4Ips=[], v6Ips=[], ips=[]}`. You can get IPv6 addresses to make URL requests as needed.

When you make a URL request by using an IPv6 address, you need to enclose the IPv6 address in square brackets, such as `http://[64:ff9b::b6fe:7475]/`.

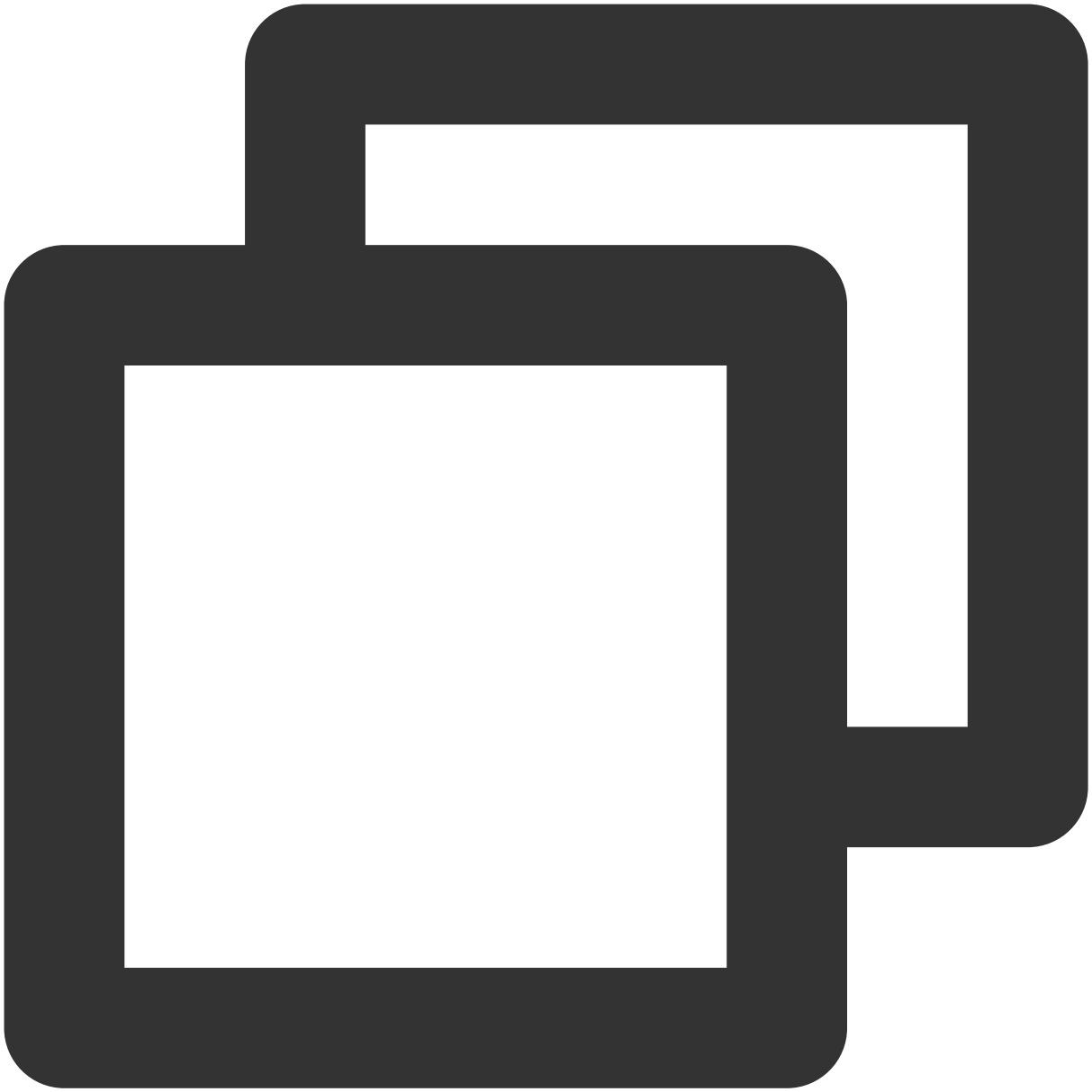
Best Practices

OkHttp

Last updated：2023-06-12 14:47:38

OkHttp

OkHttp provides a DNS API to inject a DNS implementation into OkHttp. Thanks to the excellent design of OkHttp, when OkHttp is used to access the network, you can connect to HTTPDNS to resolve domains by implementing the DNS API with no extra work required even in complex scenarios (HTTP/HTTPS/WebSocket + SNI), which causes little intrusion. Below is a sample:



```
mOkHttpClient =
    new OkHttpClient.Builder()
        .dns(new Dns() {
            @NonNull
            @Override
            public List<InetAddress> lookup(String hostname) {
                Utils.checkNotNull(hostname, "hostname can not be null");
                String ips = MSDKDnsResolver.getInstance().getAddrByName(hostname);
                String[] ipArr = ips.split(";");
                if (0 == ipArr.length) {
                    return Collections.emptyList();
```

```
                }
                List<InetAddress> inetAddressList = new ArrayList<>(ipArr.length);
                for (String ip : ipArr) {
                    if ("0".equals(ip)) {
                        continue;
                    }
                    try {
                        InetAddress inetAddress = InetAddress.getByName(ip);
                        inetAddressList.add(inetAddress);
                    } catch (UnknownHostException ignored) {
                    }
                }
                return inetAddressList;
            }
        })
        .build();
```

Note

Implementing the DNS API means that all network requests processed by the current OkHttpClient instance will go through HTTPDNS. If you have only a small number of domains to be resolved by HTTPDNS, we recommend you filter them before calling the DNS API of HTTPDNS.

Retrofit + OkHttp

Retrofit is actually a library that adds a layer of encapsulation bridging to the API based on OkHttp. Therefore, you can connect to HTTPDNS simply by customizing the OkHttpClient in Retrofit like in the OkHttp connection method as shown below:

```
mRetrofit =
    new Retrofit.Builder()
        .client(mOkHttpClient)
        .baseUrl(baseUrl)
        .build();
```

WebView

Last updated：2023-06-12 14:47:58

The Android system provides APIs to intercept network requests and inject custom logic in WebView. You can get the host of a URL request by intercepting various types of network requests from WebView, call HTTPDNS to resolve the host, and then request the network address by using the new URL formed with the obtained IP.

Note

Because the `WebResourceRequest` provided in `shouldInterceptRequest(WebView view, WebResourceRequest request)` does not contain request body information, the SDK can only successfully intercept GET requests, but not POST requests.

```
WebSettings webSettings = mWebView.getSettings();
// Adopt the default cache policy, i.e., the cache will always be used before it ex
webSettings.setCacheMode(WebSettings.LOAD_DEFAULT);
// Load web image resources
webSettings.setBlockNetworkImage(false);
// Support JavaScript scripts
webSettings.setJavaScriptEnabled(true);
// Support zoom
webSettings.setSupportZoom(true);
mWebView.setWebViewClient(new WebViewClient() {
    // Use this method for API 21 or later
```

```
    @SuppressLint("NewApi")
    @Override
    public WebResourceResponse shouldInterceptRequest(WebView view, WebResourceRequ
        if (request != null && request.getUrl() != null && request.getMethod().equa
            String scheme = request.getUrl().getScheme().trim();
            String url = request.getUrl().toString();
            Log.d(TAG, "url:" + url);
            // HTTPDNS resolves the network and image requests of CSS files
            if ((scheme.equalsIgnoreCase("http") || scheme.equalsIgnoreCase("https"
            && (url.contains(".css") || url.endsWith(".png") || url.endsWith(".jpg"
                try {
                    URL oldUrl = new URL(url);
                    URLConnection connection = oldUrl.openConnection();
                    // Get the HTTPDNS query result
                    String ips = MSDKDnsResolver.getInstance().getAddrByName(oldUrl
                    String[] ipArr = ips.split(";");
                    if (2 == ipArr.length && !"0".equals(ipArr[0])) { // Get the IP
                        String ip = ipArr[0];
                        String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
                        connection = (HttpURLConnection) new URL(newUrl).openConnec
                        connection.setRequestProperty("Host", oldUrl.getHost());
                    }
                    Log.d(TAG, "contentType:" + connection.getContentType());
                    return new WebResourceResponse("text/css", "UTF-8", connection.
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return null;
    }

    // Use this method for API 11-20
    public WebResourceResponse shouldInterceptRequest(WebView view, String url) {
        if (!TextUtils.isEmpty(url) && Uri.parse(url).getScheme() != null) {
            String scheme = Uri.parse(url).getScheme().trim();
            Log.d(TAG, "url:" + url);
            // HTTPDNS resolves the network and image requests of CSS files
            if ((scheme.equalsIgnoreCase("http") || scheme.equalsIgnoreCase("https"
            && (url.contains(".css") || url.endsWith(".png") || url.endsWith(".jpg"
                try {
                    URL oldUrl = new URL(url);
                    URLConnection connection = oldUrl.openConnection();
                    // Get the HTTPDNS query result
                    String ips = MSDKDnsResolver.getInstance().getAddrByName(oldUrl
```

```
                    String[] ipArr = ips.split(";");
                    if (2 == ipArr.length && !"0".equals(ipArr[0])) { // Get the IP
                        String ip = ipArr[0];
                        String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
                        connection = (HttpURLConnection) new URL(newUrl).openConnec
                        connection.setRequestProperty("Host", oldUrl.getHost());
                    }
                    Log.d(TAG, "contentType:" + connection.getContentType());
                    return new WebResourceResponse("text/css", "UTF-8", connection.
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                }
            }
        }
        return null;
    }});
// Load web resources
mWebView.loadUrl(targetUrl);
```

HttpURLConnection

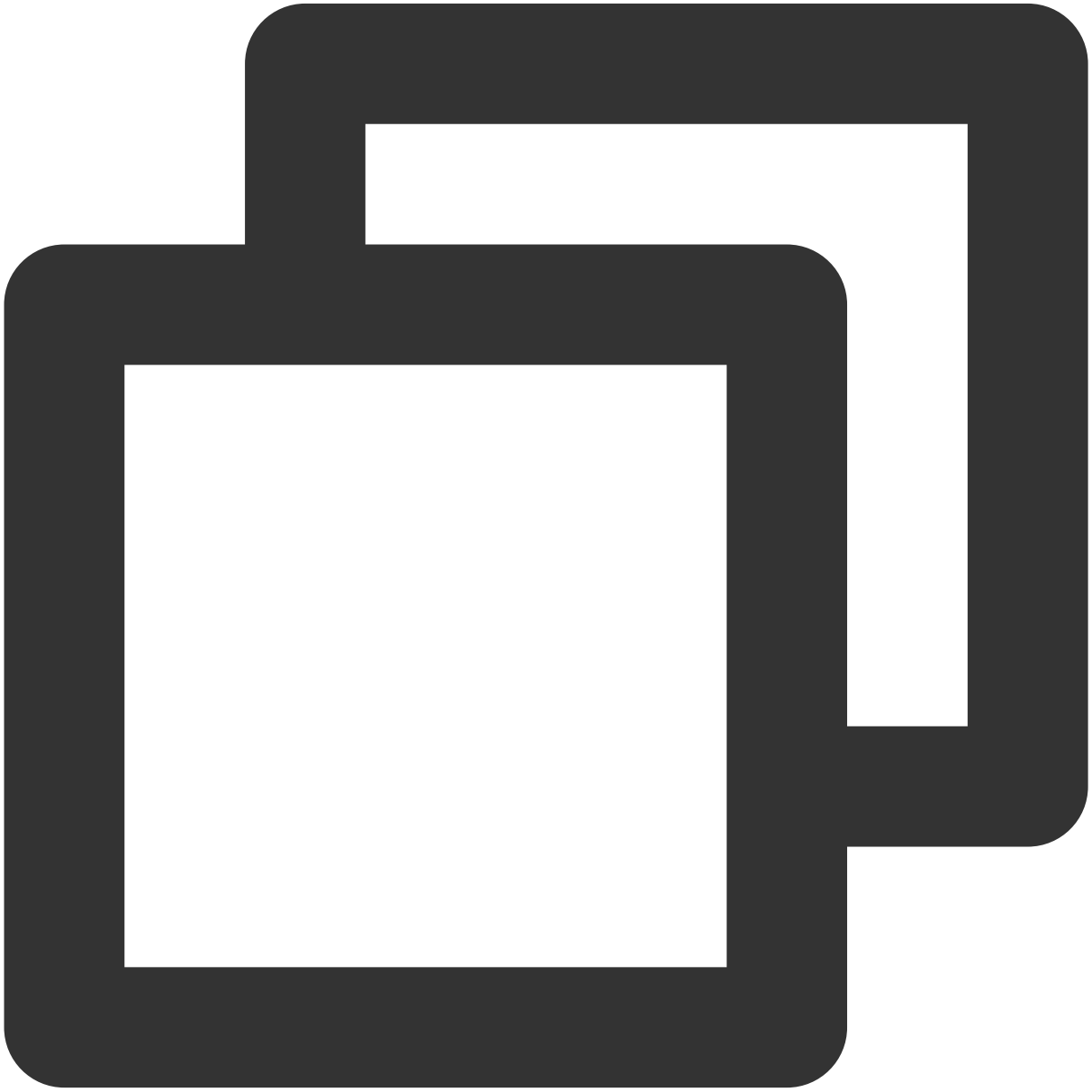
Last updated：2023-06-12 14:48:18

The sample for HTTPS certificate validation is as follows:

```
 // Take the domain `www.qq.com` and the IP `192.168.0.1` obtained by HTTPDNS as an
String url = "https://192.168.0.1/"; // Your own request connection
 HttpsURLConnection connection = (HttpsURLConnection) new URL(url).openConnection()
 connection.setRequestProperty("Host", "www.qq.com");
 connection.setHostnameVerifier(new HostnameVerifier() {
```

```
    @Override
    public boolean verify(String hostname, SSLSession session) {
        return HttpsURLConnection.getDefaultHostnameVerifier().verify("www.qq.com",
    }
});
connection.setConnectTimeout(mTimeOut); // Set the connection timeout period
connection.setReadTimeout(mTimeOut); // Set the stream read timeout period
connection.connect();
```

The sample for HTTPS + SNI certificate validation is as follows:



```
// Take the domain `www.qq.com` and the IP `192.168.0.1` obtained by HTTPDNS as an
```

```
String url = "https://192.168.0.1/"; // Encapsulate the business' request URL with
HttpsURLConnection sniConn = null;
try {
  sniConn = (HttpsURLConnection) new URL(url).openConnection();
  // Set the host field of the HTTP request header
  sniConn.setRequestProperty("Host", "www.qq.com");
  sniConn.setConnectTimeout(3000);
  sniConn.setReadTimeout(3000);
  sniConn.setInstanceFollowRedirects(false);
  // Customize SSLSocketFactory to carry the requested domain ***(key step)
  SniSSLSocketFactory sslSocketFactory = new SniSSLSocketFactory(sniConn);
  sniConn.setSSLSocketFactory(sslSocketFactory);
  // Verify whether the hostname and the server authentication scheme match
  HostnameVerifier hostnameVerifier = new HostnameVerifier() {
      @Override
      public boolean verify(String hostname, SSLSession session) {
          return HttpsURLConnection.getDefaultHostnameVerifier().verify("originall
      }
  };
  sniConn.setHostnameVerifier(hostnameVerifier);
  ...
} catch (Exception e) {
  Log.w(TAG, "Request failed", e);
} finally {
  if (sniConn != null) {
      sniConn.disconnect();
  }
}

class SniSSLSocketFactory extends SSLSocketFactory {

  private HttpsURLConnection mConn;

  public SniSSLSocketFactory(HttpsURLConnection conn) {
      mConn = conn;
  }

  @Override
  public Socket createSocket() throws IOException {
      return null;
  }

  @Override
  public Socket createSocket(String host, int port) throws IOException, UnknownHos
      return null;
  }
```

```
    @Override
    public Socket createSocket(String host, int port, InetAddress localHost, int loc
        return null;
    }

    @Override
    public Socket createSocket(InetAddress host, int port) throws IOException {
        return null;
    }

    @Override
    public Socket createSocket(InetAddress address, int port, InetAddress localAddre
        return null;
    }

    @Override
    public String[] getDefaultCipherSuites() {
        return new String[0];
    }

    @Override
    public String[] getSupportedCipherSuites() {
        return new String[0];
    }

    @Override
    public Socket createSocket(Socket socket, String host, int port, boolean autoClo
        String realHost = mConn.getRequestProperty("Host");
        if (realHost == null) {
            realHost = host;
        }
        Log.i(TAG, "customized createSocket host is: " + realHost);
        InetAddress address = socket.getInetAddress();
        if (autoClose) {
            socket.close();
        }
        SSLCertificateSocketFactory sslSocketFactory = (SSLCertificateSocketFactory)
        SSLSocket ssl = (SSLSocket) sslSocketFactory.createSocket(address, port);
        ssl.setEnabledProtocols(ssl.getSupportedProtocols());
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR1) {
            Log.i(TAG, "Setting SNI hostname");
            sslSocketFactory.setHostname(ssl, realHost);
        } else {
            Log.d(TAG, "No documented SNI support on Android < 4.2, trying with refl
            try {
                Method setHostnameMethod = ssl.getClass().getMethod("setHostname", S
                setHostnameMethod.invoke(ssl, realHost);
```

```
            } catch (Exception e) {
                Log.w(TAG, "SNI not useable", e);
            }
        }
        // Verify hostname and certificate
        SSLSession session = ssl.getSession();
        HostnameVerifier hostnameVerifier = HttpsURLConnection.getDefaultHostnameVer
        if (!hostnameVerifier.verify(realHost, session)) {
            throw new SSLPeerUnverifiedException("Cannot verify hostname: " + realHo
        }
        Log.i(TAG, "Established " + session.getProtocol() + " connection with " + se
        return ssl;
    }
}
```

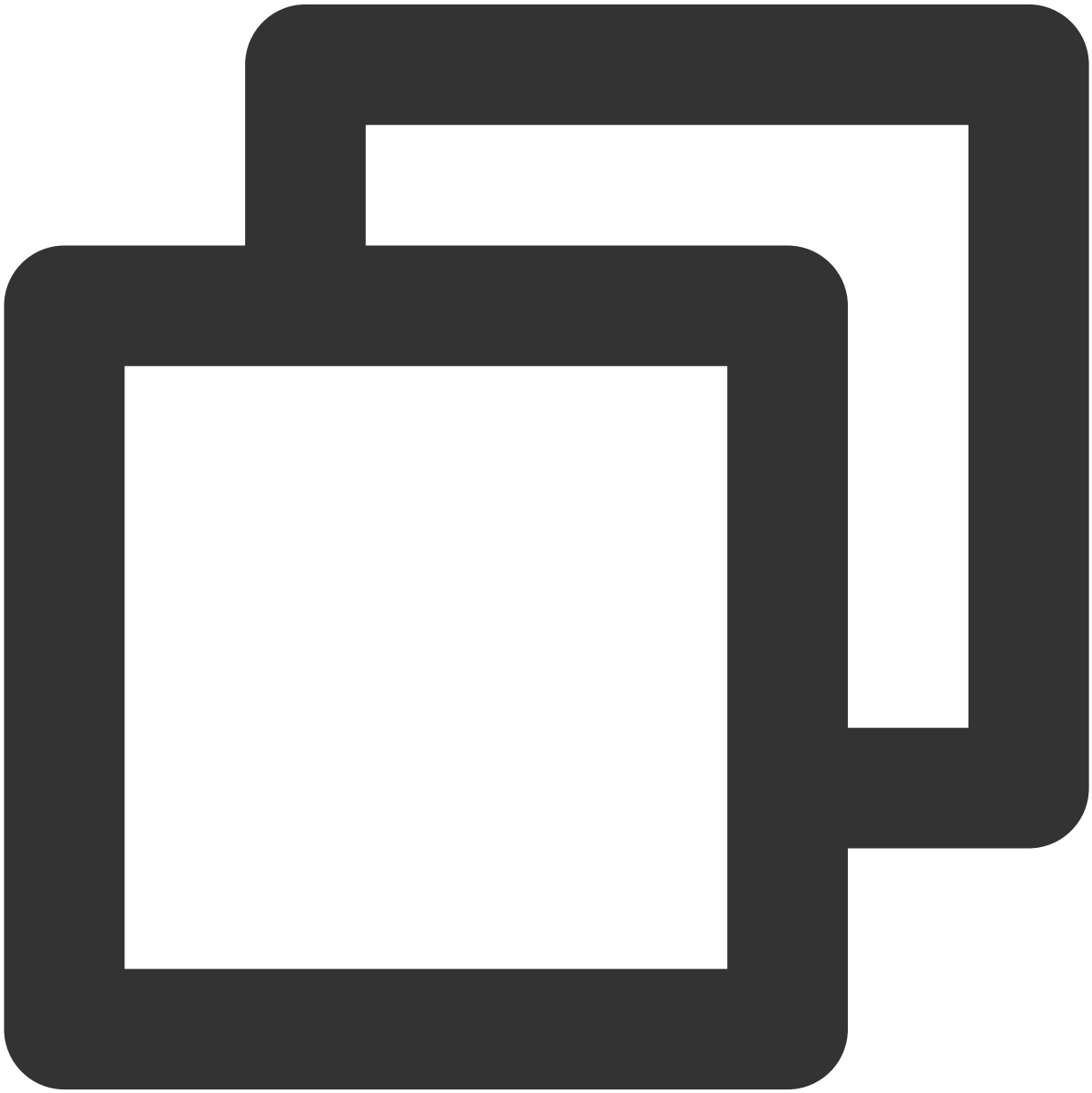
Unity

Last updated：2023-06-12 14:45:51

Prepare for the integration.

Obtain the SDK file and copy it to the `Assets/Plugins/Android` directory of the Unity project.

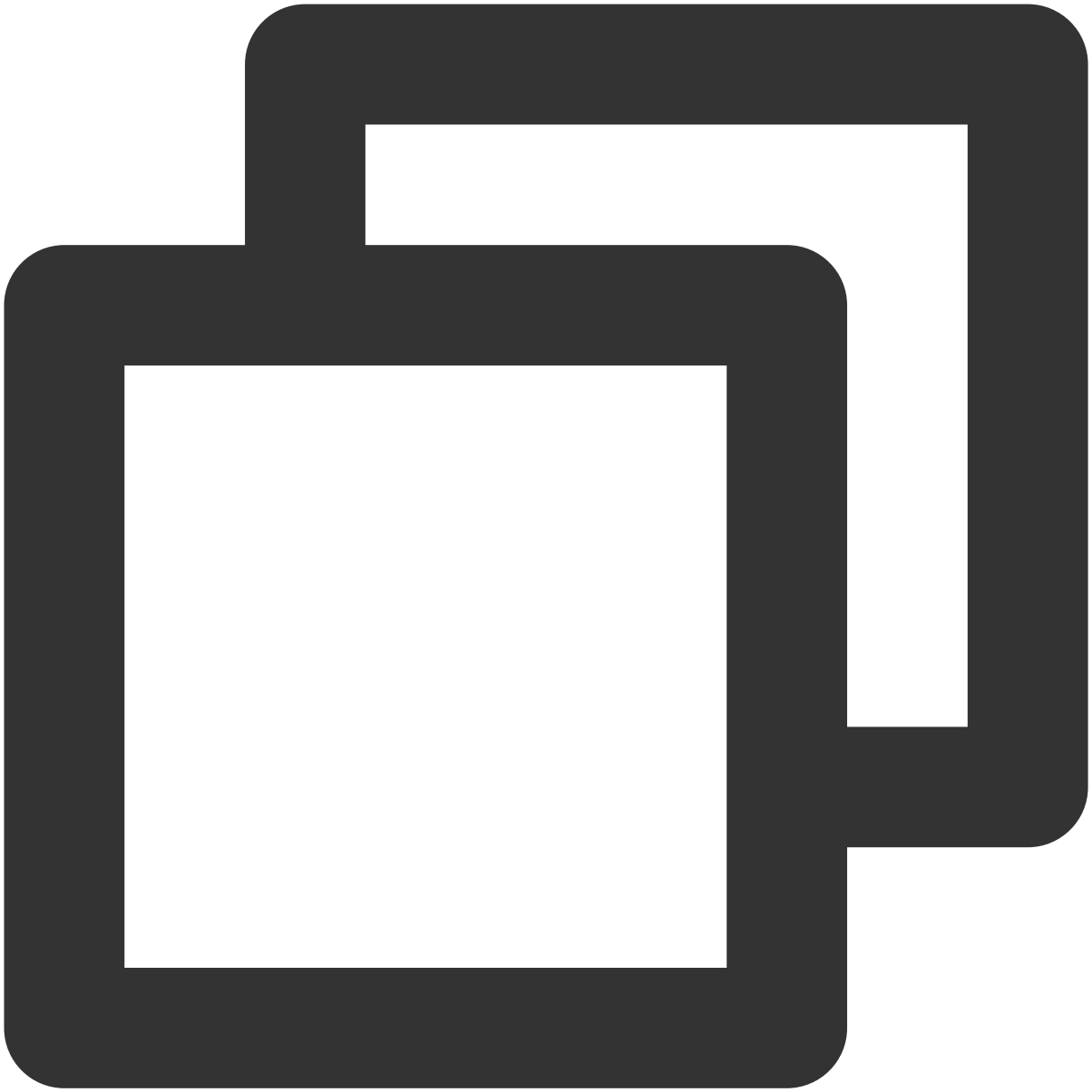
Initialize HTTPDNS. Sample code:



```
private static AndroidJavaObject sHttpDnsObj;
public static void Init() {
```

```
    AndroidJavaClass unityPlayerClass = new AndroidJavaClass("com.unity3d.player.U
    if (unityPlayerClass == null) {
        return;
    }
    AndroidJavaObject activityObj = unityPlayerClass.GetStatic<AndroidJavaObject>(
    if (activityObj == null) {
        return;
    }
    AndroidJavaObject contextObj = activityObj.Call<AndroidJavaObject>("getApplica
    // Initialize HTTPDNS
    AndroidJavaObject httpDnsClass = new AndroidJavaObject("com.tencent.msdk.dns.M
    if (httpDnsClass == null) {
        return;
    }
    sHttpDnsObj = httpDnsClass.CallStatic<AndroidJavaObject>("getInstance");
    if (sHttpDnsObj == null) {
        return;
    }

    // For v4.0.0 or later (recommended). For more information, see Android SDK
    AndroidJavaObject dnsConfigBuilder = new AndroidJavaObject("com.tencent.msdk.d
    dnsConfigBuilder.Call<AndroidJavaObject>("dnsId", "XXX");
    dnsConfigBuilder.Call<AndroidJavaObject>("dnsIp", "XXX");
    dnsConfigBuilder.Call<AndroidJavaObject>("dnsKey", "XXX");
    // Other configuration information
    AndroidJavaObject dnsConfig = dnsConfigBuilder.Call<AndroidJavaObject>("build"

    sHttpDnsObj.Call("init", contextObj, dnsConfig);

    // For a version earlier than v4.0.0
    sHttpDnsObj.Call("init", contextObj, appkey, dnsid, dnskey, dnsIp, debug, time
}
```

Call the `getAddrByName` API to resolve a domain. Sample code:



```
// We recommend you perform this operation on a child thread or with Coroutine
// Note that you need to call the `AttachCurrentThread` and `DetachCurrentThread` f
public static string GetHttpDnsIP(string url) {
    string ip = string.Empty;
    AndroidJNI.AttachCurrentThread(); // You need to add this when calling the `get
    // Get the IP configuration set
    string ips = sHttpDnsObj.Call<string>("getAddrByName", url);
    AndroidJNI.DetachCurrentThread(); // You need to add this when calling the `get
    if (null != ips) {
        string[] ipArr = ips.Split(';');
        if (2 == ipArr.Length && !"0".Equals(ipArr[0]))
```

```
        ip = ipArr[0];
    }
    return ip;
}
```

For the information about HTTPS certificate validation in a direct IP connection scenario, see here. Sample code:

```
UnityWebRequest www = UnityWebRequest.Get(url);
www.certificateHandler = new CertHandler();
yield return www.SendWebRequest();

// CertHandler is a custom certificate handler class that overrides the ValidateCe
```

```
public class CertHandler : CertificateHandler
{
    protected override bool ValidateCertificate(byte[] certificateData)
    {
        X509Certificate2 certificate = new X509Certificate2(certificateData);
        X509Certificate2Collection collection = new X509Certificate2Collection();
        collection.Import(Application.dataPath + "/Certificates/server.cer");
        return certificate.Issuer == collection[0].Issuer;
    }
}
```