移动解析 HTTPDNS

SDK 文档

产品文档

腾讯云

文档目录

SDK 文档
SDK 开通流程

最近更新时间：2022-06-22 15:20:24

概述

本文档将指导您开通移动解析 HTTPDNS 服务后，如何在控制台提交接入 SDK 的申请。

前提条件

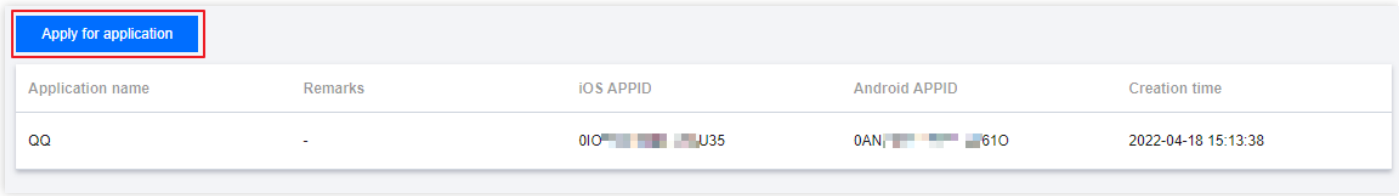
已 开通移动解析 HTTPDNS 服务。

操作步骤

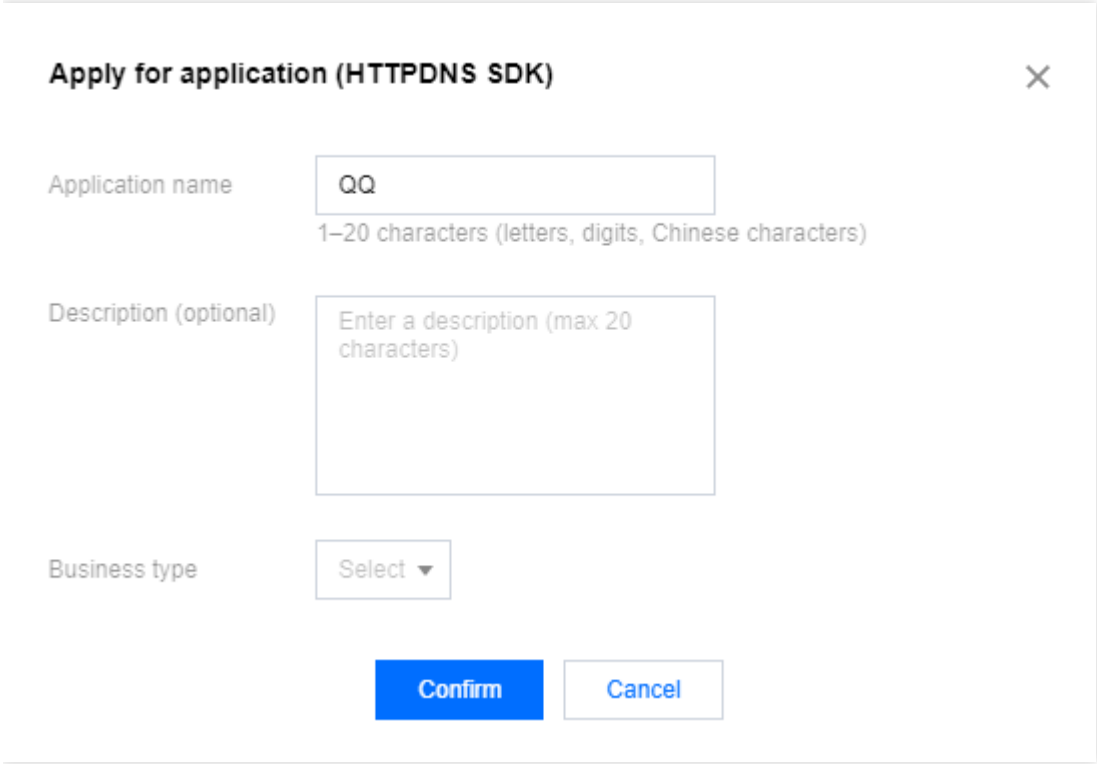
1. 登录 移动解析 HTTPDNS 管理控制台。

2. 在左侧菜单栏中选择**开发配置**，即可进入“开发配置” 管理页面。

3. 在 “开发配置” 管理页面，单击**申请应用**。如下图所示：



4. 在弹出的 “申请应用（HTTPDNS SDK）” 窗口中，填写相关信息。如下图所示:



- **应用名称**：输入您的应用名称。
- **描述（选填）**：输入您的应用描述。
- **业务类型**：请根据您的实际业务情况进行选择。

1. 单击**保存**，即可成功申请。您的腾讯云安全邮箱将会收到接入 SDK 必须的 APPKey（Android 和 iOS 各一个）。如下图所示：或您也可以在**开发配置**页面进行查看。如下图所示：

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.

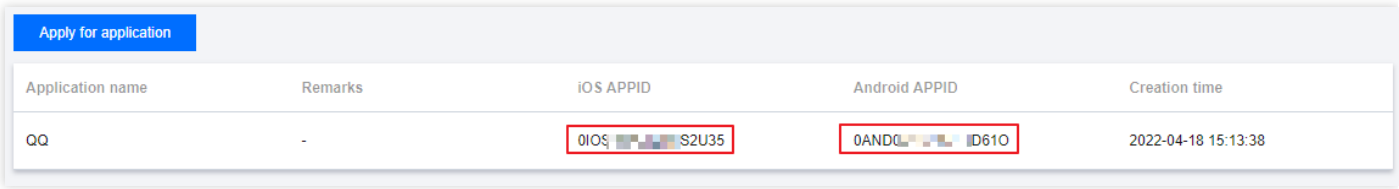
Note: To use the SDK, you need to replace the information with your account ID and AppID. For details, see the above documentation.

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**.


Apply for application
Application name
Remarks
iOS APPID
Android APPID
Creation time
QQ
-
0IOS S2U35
0AND ID61O
2022-04-18 15:13:38

IOS SDK 文档
IOS SDK 发布动态

最近更新时间：2023-09-05 09:53:17

v1.8.1（2023年08月28日）

功能优化。

v1.8.0（2023年07月05日）

dnsip（HTTPDNS服务IP） SDK内部调度，无需用户配置。
原灯塔上报服务（Beacon）已正式下线。
包体积优化。
功能优化。

v1.7.0（2023年05月23日）

SDK支持数据上报统计分析，配合控制台**解析监控**使用。原灯塔上报服务（Beacon）将下线，建议尽快切换。
新增独立国际站 SDK。
使用 sqlite3 替换 WCDB 实现本地缓存。
功能优化。

v1.6.0（2022年09月06日）

新增本地持久化缓存功能。
新增允许返回 TTL 过期域名的 IP。

v1.5.0（2022年08月12日）

新增 IP 优选功能。

v1.4.0（2022年07月11日）

增加缓存刷新逻辑。

v1.3.5（2022年03月14日）

优化多线程逻辑。

v1.3.4（2022年03月10日）

修复 bug。

v1.3.3（2022年02月25日）

支持域名预解析

支持返回所有 IP

支持指定返回的 IP 类型

v1.3.2（2022年02月08日）

优化多线程

IOS SDK 接入

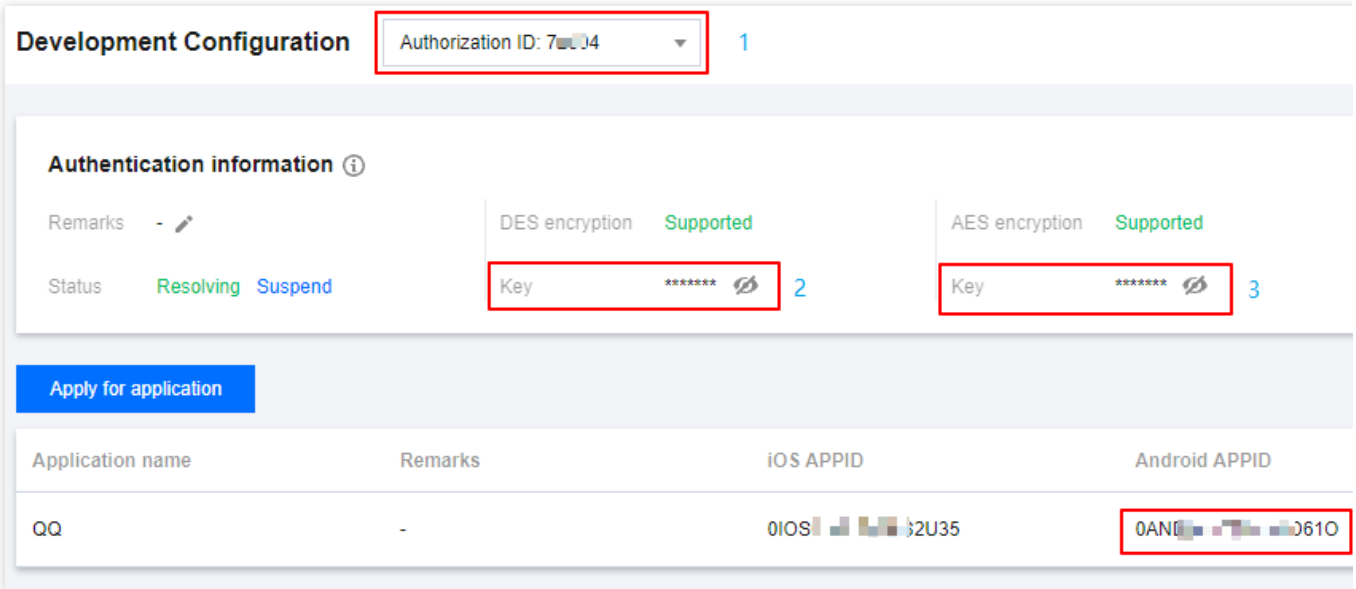
最近更新时间：2023-10-18 10:41:34

概述

移动解析 HTTPDNS 的主要功能是为了有效避免由于运营商传统 LocalDNS 解析导致的无法访问最佳接入点的方案。原理为使用 HTTP 加密协议替代传统的 DNS 协议，整个过程不使用域名，大大减少劫持的可能性。

前期准备

1. 开通移动解析 HTTPDNS 服务，详情请参见 开通移动解析 HTTPDNS。
2. 服务开通后，您需在移动解析 HTTPDNS 控制台添加解析域名才可正常使用，详情请参见 添加域名。
3. 在移动解析 HTTPDNS 控制台申请接入 SDK，详情请参见 开通 SDK。
4. SDK 开通后，移动解析 HTTPDNS 将为您分配授权 ID、AES 和 DES 加密密钥及 HTTPS Token 等配置信息。您可前往 开发配置 页面查看，如下图所示：



使用 iOS SDK 需求获取的配置如下：

授权 ID：使用移动解析 HTTPDNS 服务中，开发配置的唯一标识。SDK 中 `dnsId` 参数，用于域名解析鉴权。

DES加密密钥：SDK 中 `dnsKey` 参数，加密方式为 DES 时传入此项。

AES加密密钥：SDK 中 `dnsKey` 参数，加密方式为 AES 时传入此项。

HTTPS加密Token：SDK 中 `token` 参数，加密方式为 HTTPS 时传入此项。

IOS APPID（可选）：腾讯云控制台申请获得，用于数据上报。

安装包结构

SDK 最新版本包下载地址。
SDK 开源仓库地址。

名称	适用说明
MSDKDns.xcframework	适用 “Build Setting->C++ Language Dialect” 配置为 **“GNU++98”**，“Build Setting->C++ Standard Library” 为 **“libstdc++(GNU C++ standard library)”** 的工程。
MSDKDns_intl.xcframework	MSDKDns.xcframework 的国际站版本
MSDKDns_C11.xcframework	适用于该两项配置分别为 “GNU++11”和“libc++(LLVM C++ standard library with C++11 support)”的工程。
MSDKDns_C11_intl.xcframework	MSDKDns_C11.xcframework 的国际站版本

注意

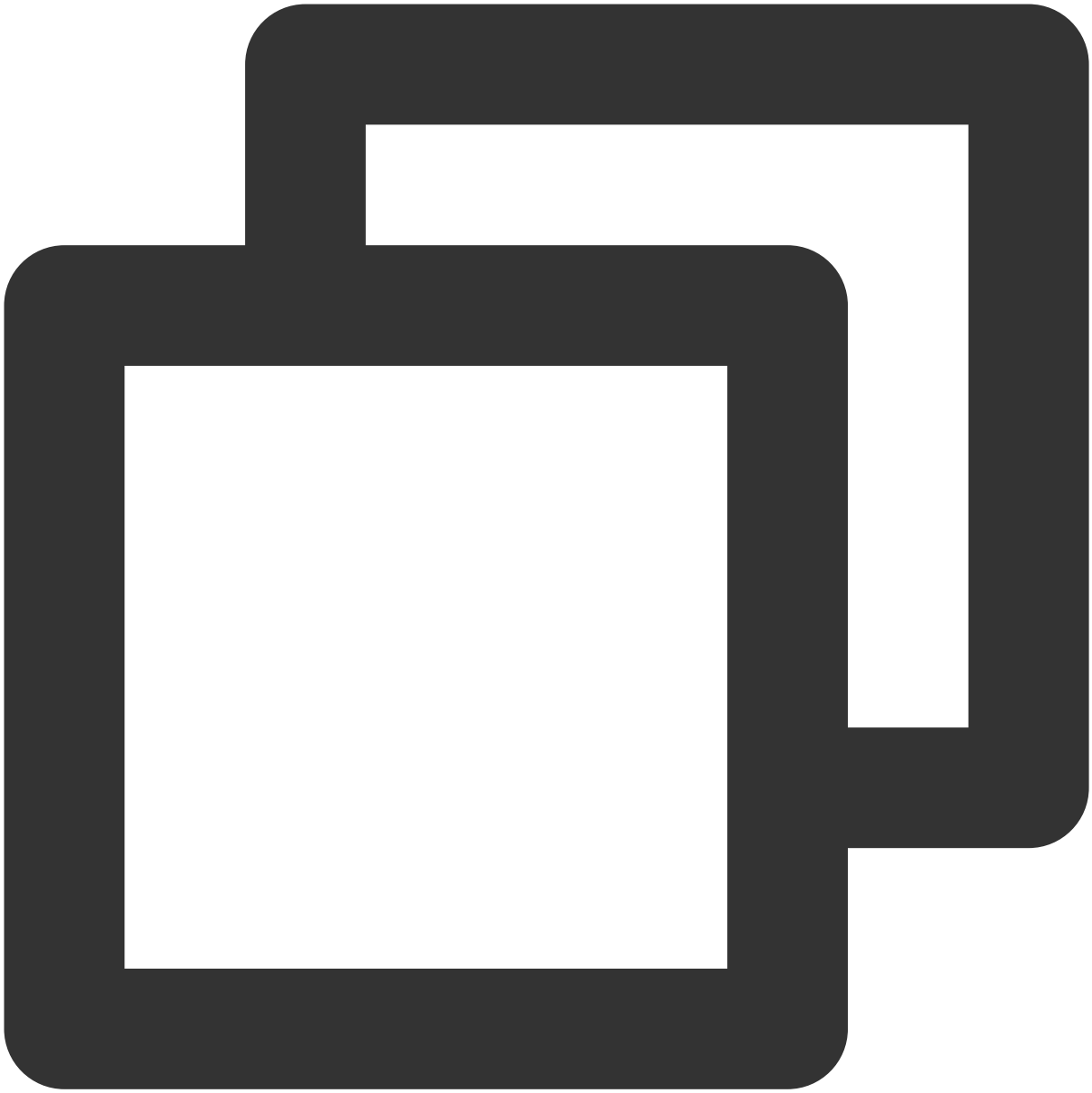
在使用**国际站**版本时，初始化配置需要前往HTTPDNS**国际站**控制台中获取。详情请参见 国际站接入文档。

SDK 集成

移动解析 HTTPDNS 提供以下两种集成方式供 IOS 开发者选择：
通过 CocoaPods 集成。
手动集成。

通过 CocoaPods 集成

1. 在工程的 Podfile 里面添加以下代码：



```
# 适用“Build Setting->C++ Language Dialect”配置为**“GNU++98”**, “Build Setting->C++ S
  pod 'MSDKDns_intl'
# 适用于该两项配置分别为**“GNU++11”**和**“libc++(LLVM C++ standard library with C++11 s
# pod 'MSDKDns_C11_intl'
```

2. 保存并执行 `pod install`，再使用后缀为 `.xcworkspace` 的文件打开工程。

说明

关于 `CocoaPods` 的更多信息，请查看 CocoaPods 官方网站。

手动集成

手动集成可以参考以下案例：

Objective-C Demo 下载地址。

Swift Demo 下载地址。

执行以下步骤：

1. 引入依赖库（位于 HTTPDNSLibs 目录下）：

MSDKDns_C11_intl.framework（或 MSDKDns_intl.framework，根据工程配置选其一）

2. 引入系统库：

libz.tbd

libsqlite3.tbd

libc++.tbd

Foundation.framework

CoreTelephony.framework

SystemConfiguration.framework

CoreGraphics.framework

Security.framework

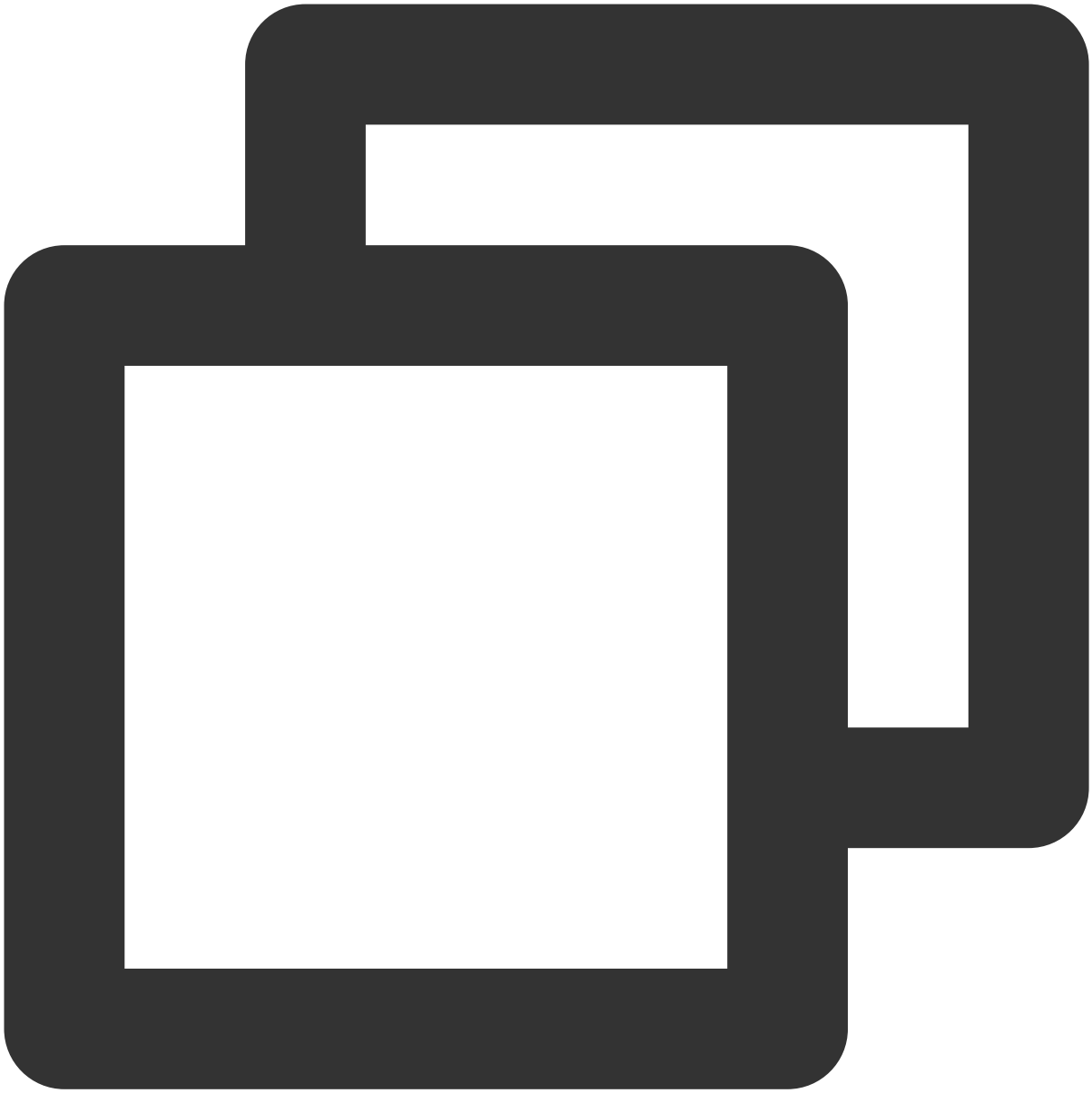
注意

请在 Other linker flag 里加入 -ObjC 标志。

SDK 初始化

接口调用示例：

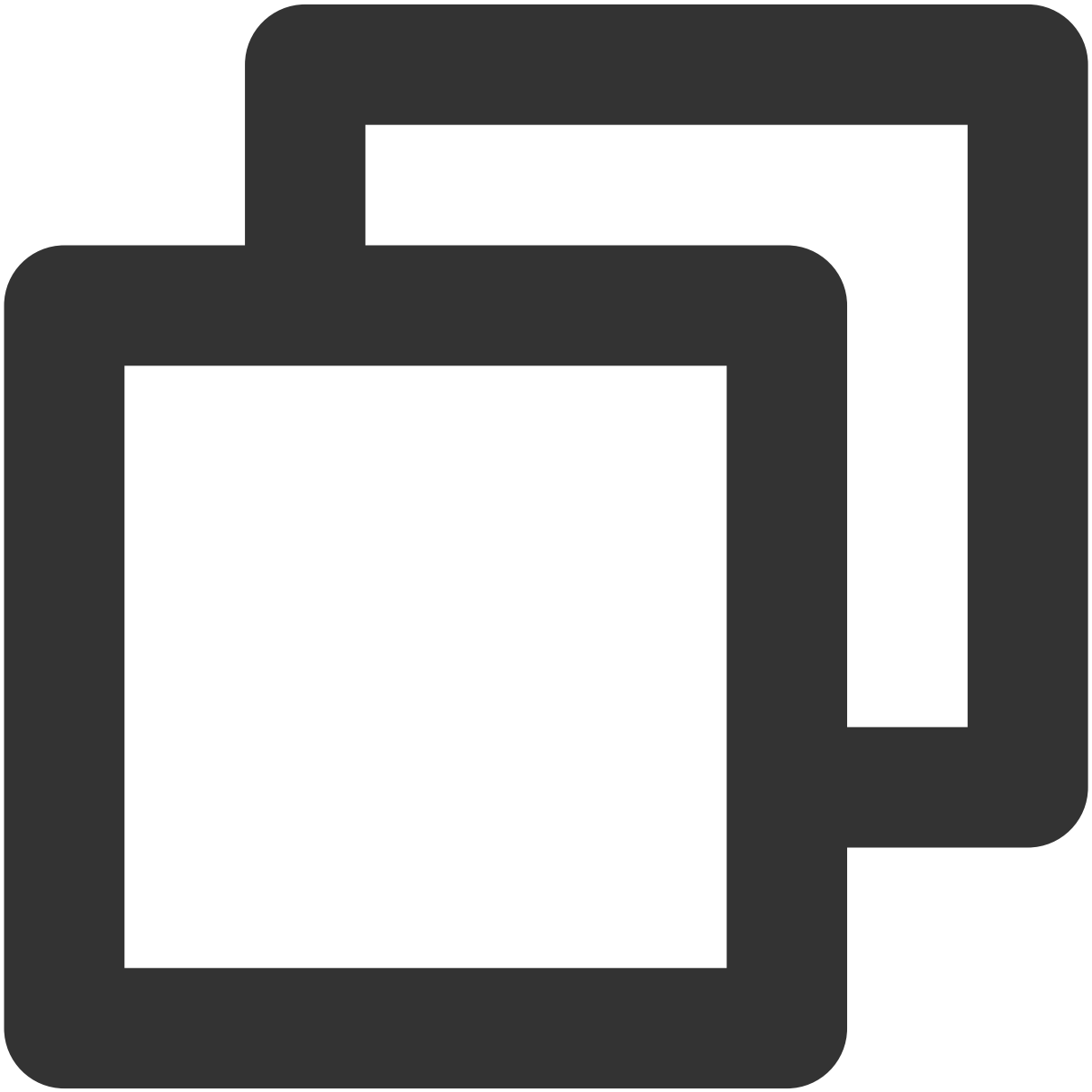
在 Objective-C 项目中。



```
//.m文件使用如下方式：
DnsConfig config = {
      .dnsId = dns授权id, // 从移动解析腾讯云控制台中获取，可以参考上文“前期准备”的截图，“授
      .dnsKey = @"加密密钥",
      .encryptType = HttpDnsEncryptTypeDES,
      .debug = YES,
      .timeout = 2000,
};
[[MSDKDns sharedInstance] initConfig: &config];

// .mm文件可以使用如下方式：
```

```
    DnsConfig *config = new DnsConfig();
    config->dnsId = dns授权id; // 从移动解析腾讯云控制台中获取，可以参考上文“前期准备”的截图，“
    config->dnsKey = @"加密密钥";
    config->encryptType = HttpDnsEncryptTypeDES;
    config->debug = YES;
    config->timeout = 2000;
    [[MSDKDns sharedInstance] initConfig: config];
```

在 Swift 项目中。



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

```
        "dnsId": "dns授权id", // 从移动解析腾讯云控制台中获取，可以参考上文“前期准备”的截图，“授
        "dnsKey": "加密密钥",
        "encryptType": 0, // 0 -> des, 1 -> aes, 2 -> https
    ]);
```

接入验证

日志验证

开启 SDK 调试日志（设置 DnsConfig 中 debug 为 YES），找到打印的 `api name:HDNSGetHostByName, data: { ... }` 日志，并检查 HTTPDNS（日志上为 **hdns_ip**）和 LocalDns（日志上为 **ldns_ip**）相关日志，可以确认接入是否成功。

key 为 **hdns_ip** 的是 HTTPDNS A 记录的解析结果。

key 为 **hdns_4a_ips** 的是 HTTPDNS AAAA 记录的解析结果。

如果 **hdns_ip** 或 **hdns_4a_ips** 不为空，则说明接入成功。

key 为 **ldns_ip** 的是 LocalDNS 的解析结果。

注意事项

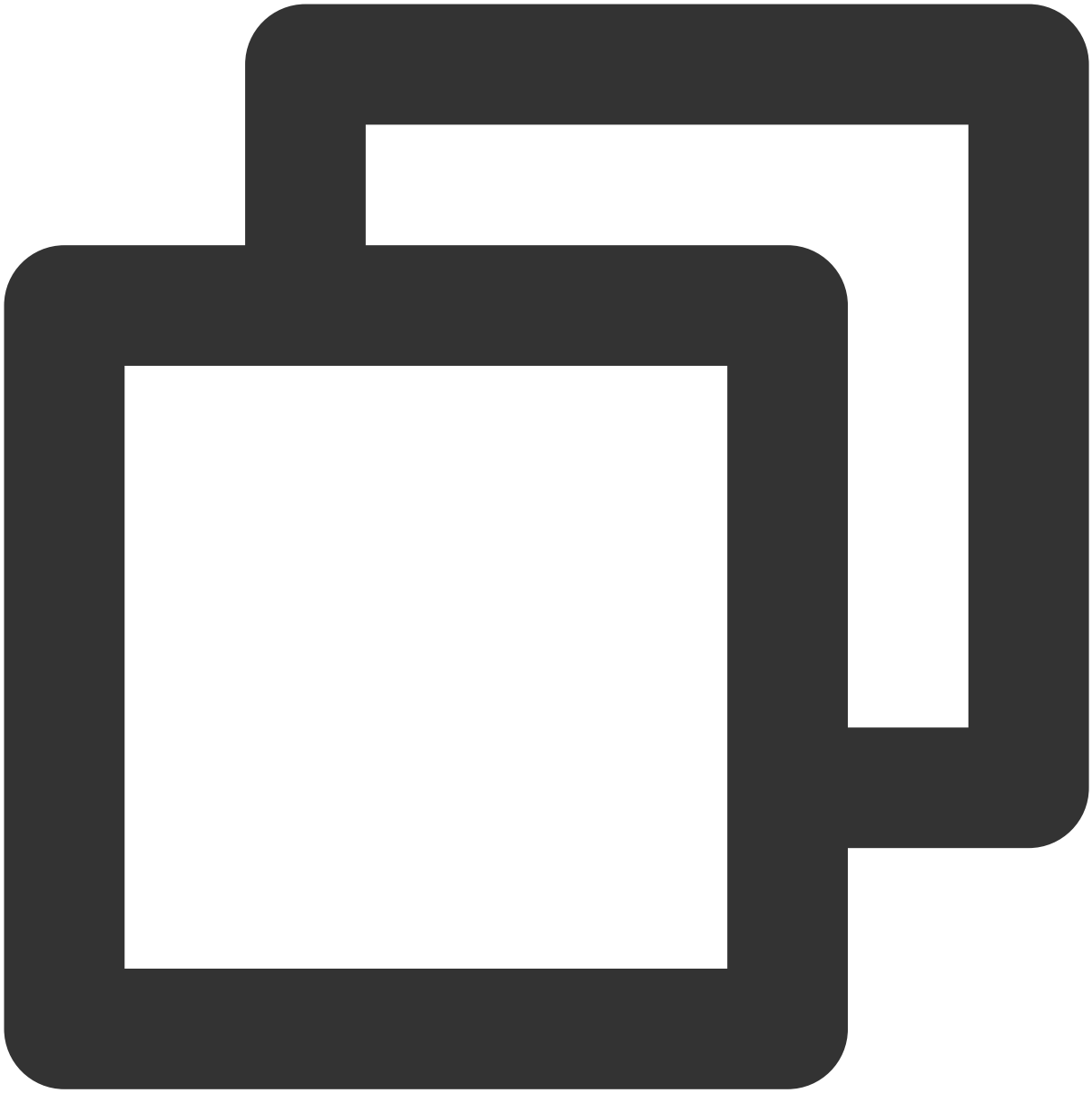
如客户端的业务与 host 绑定，例如绑定了 host 的 HTTP 服务或者是 cdn 的服务，那么在用 HTTPDNS 返回的 IP 替换掉 URL 中的域名以后，还需要指定下 HTTP 头的 host 字段。示例如下：

NSURLConnection

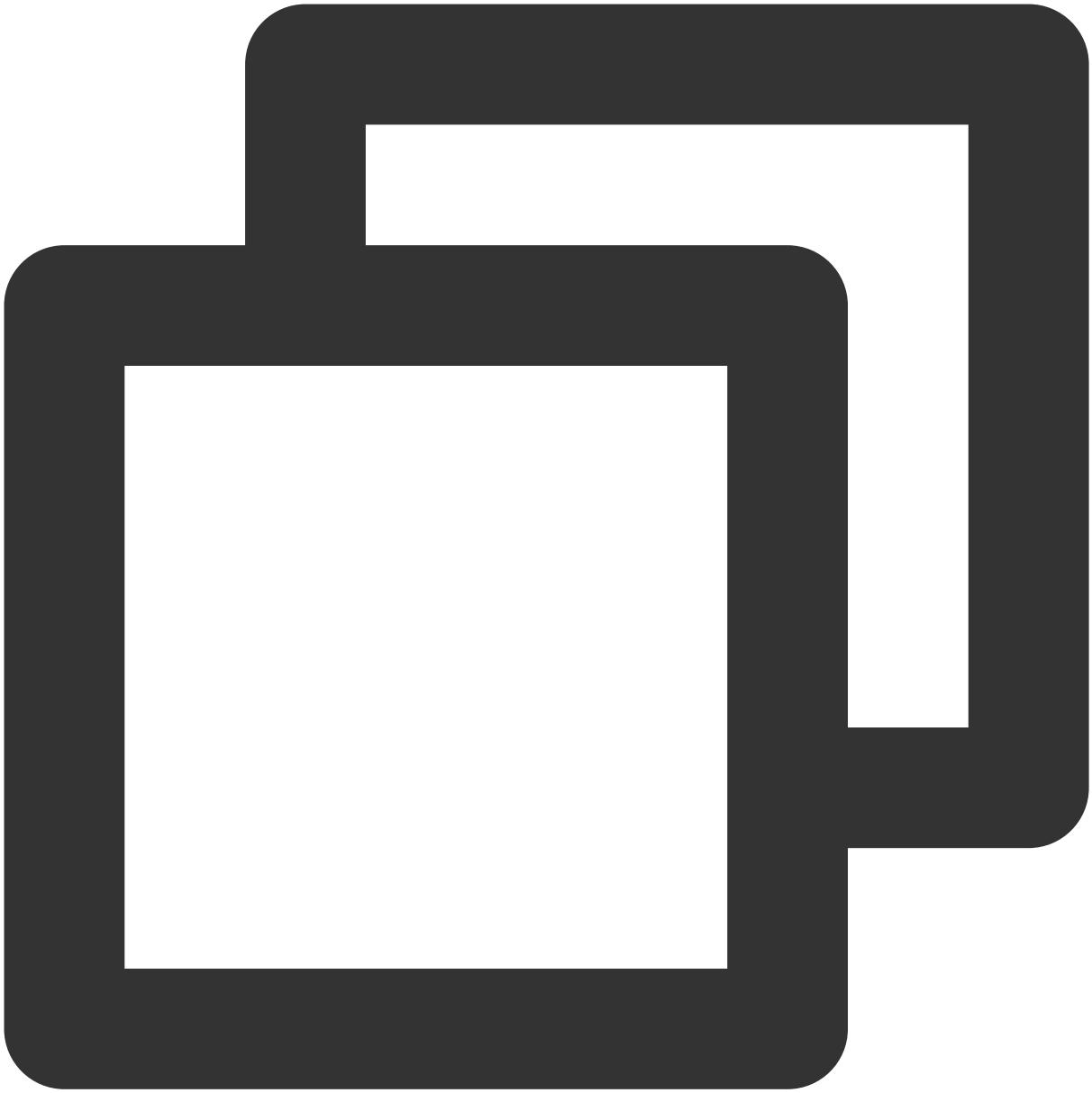
NSURLSession

curl

Unity 的 WWW 接口

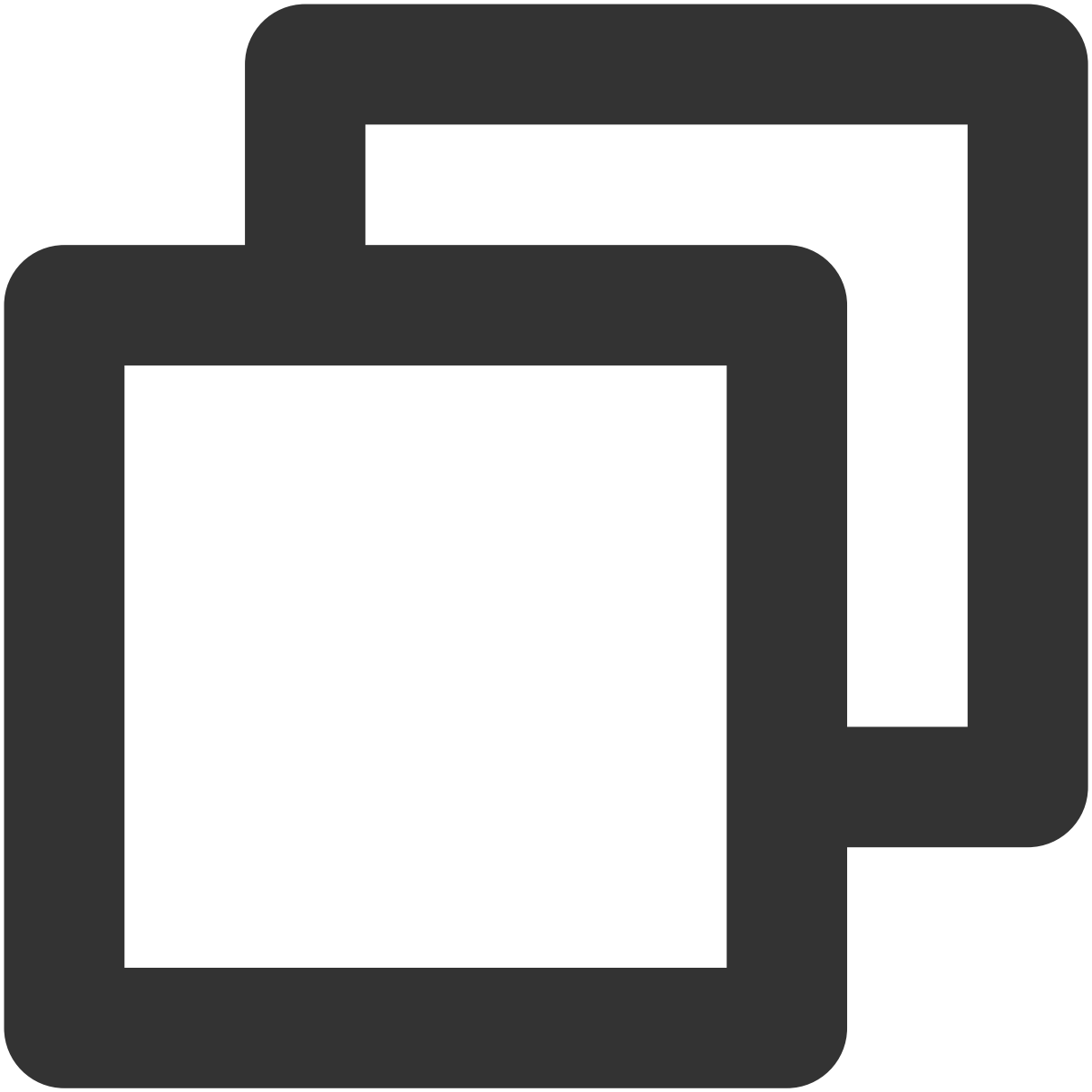


```
NSURL *httpDnsURL = [NSURL URLWithString:@"使用解析结果ip拼接的URL"];
float timeOut = 设置的超时时间;
NSMutableURLRequest *mutableReq = [NSMutableURLRequest requestWithURL:httpDnsURL ca
[mutableReq setValue:@"原域名" forHTTPHeaderField:@"host"];
NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest:mutableReq d
[connection start];
```

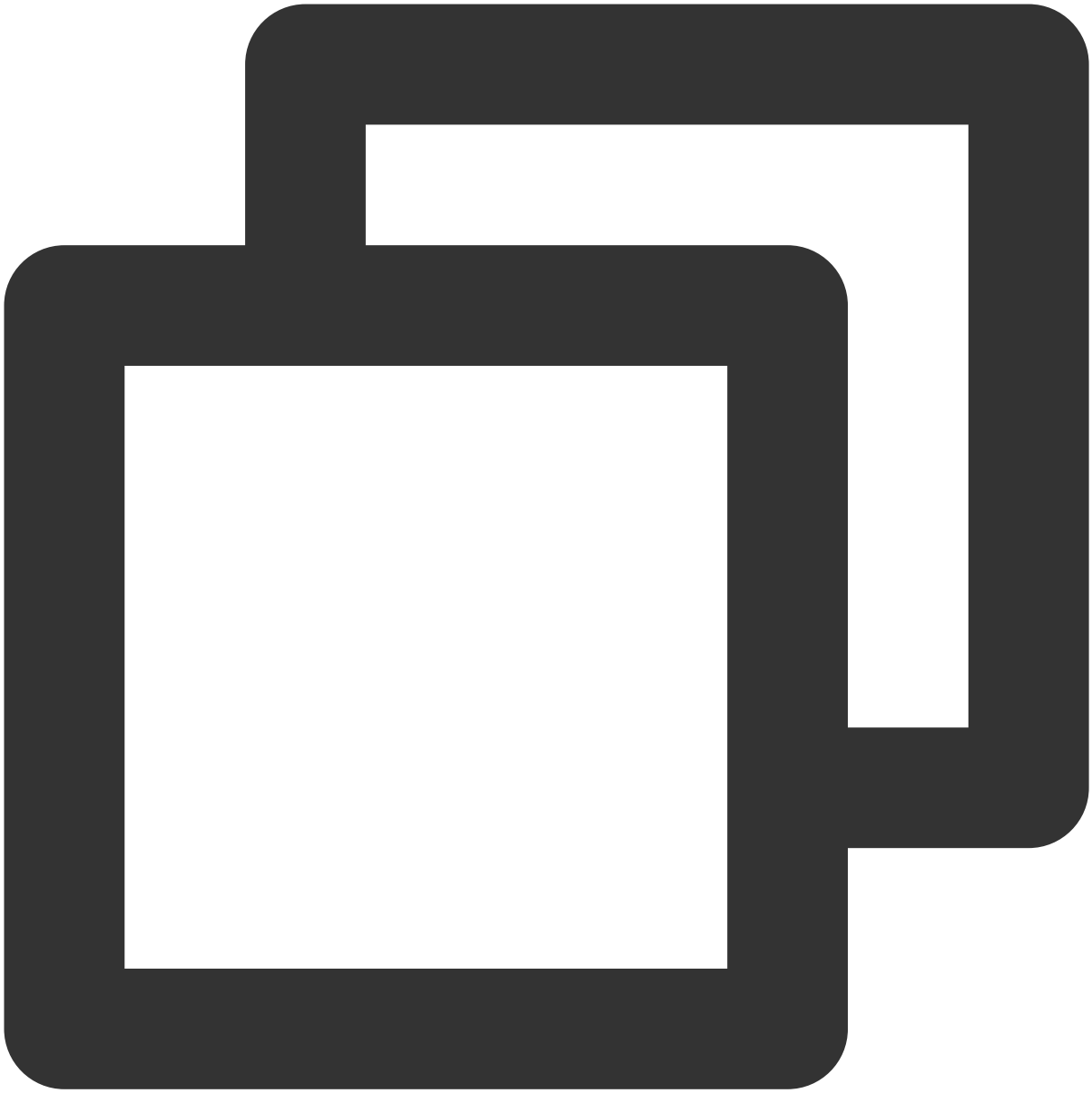


```
NSURL *httpDnsURL = [NSURL URLWithString:@"使用解析结果ip拼接的URL"];
float timeOut = 设置的超时时间;
NSMutableURLRequest *mutableReq = [NSMutableURLRequest requestWithURL:httpDnsURL ca
[mutableReq setValue:@"原域名" forHTTPHeaderField:@"host"];
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSessio
NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration delega
NSURLSessionTask *task = [session dataTaskWithRequest:mutableReq];
[task resume];
```

假设您要访问 www.qq.com，通过 HTTPDNS 解析出来的 IP 为192.168.0.111，那么通过以下方式调用即可：

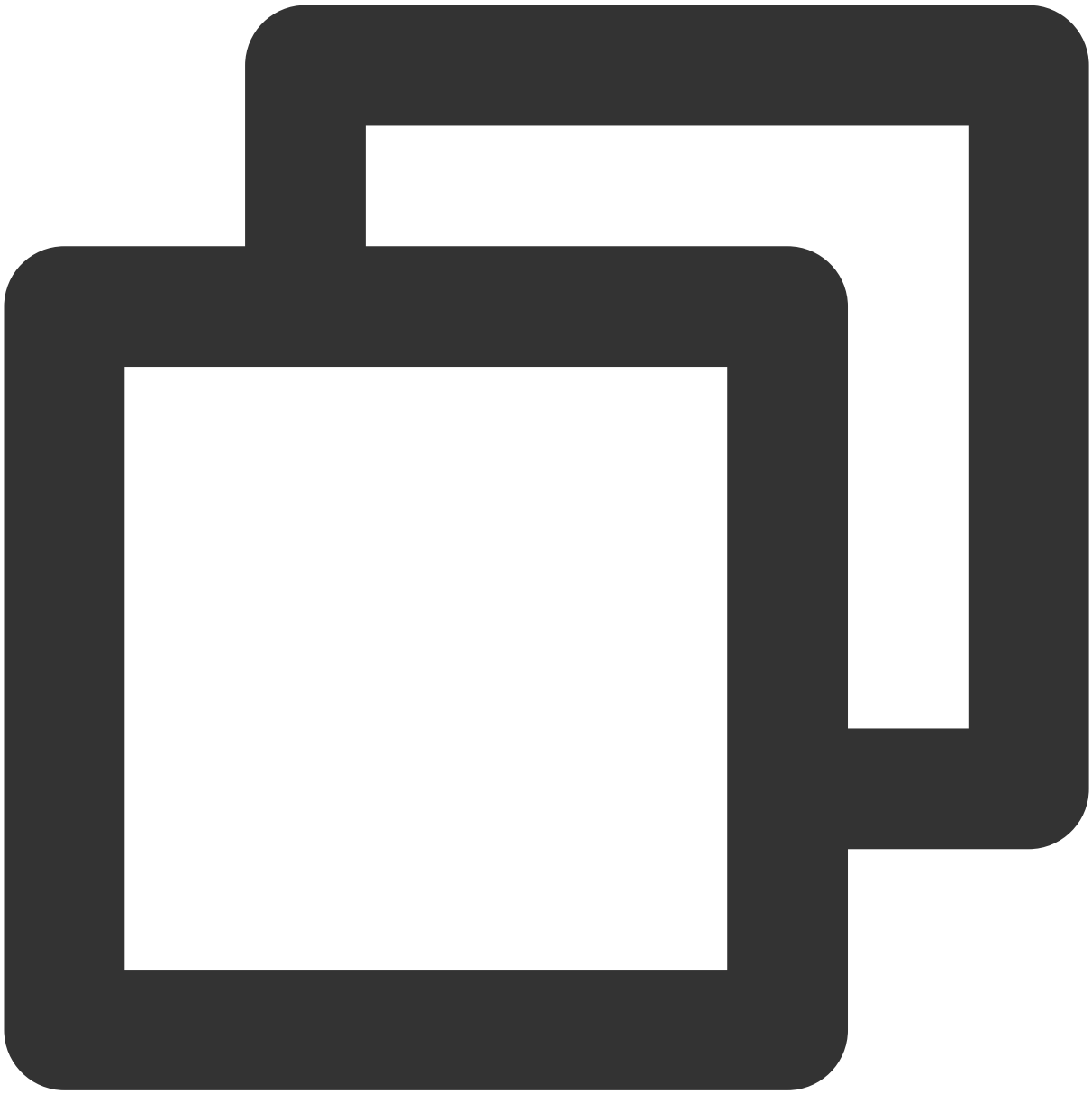


```
curl -H "host:www.qq.com" http://192.168.0.111/aaa.txt.
```



```
string httpDnsURL = "使用解析结果ip拼接的URL";
Dictionary<string, string> headers = new Dictionary<string, string> ();
headers["host"] = "原域名";
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);
}
```

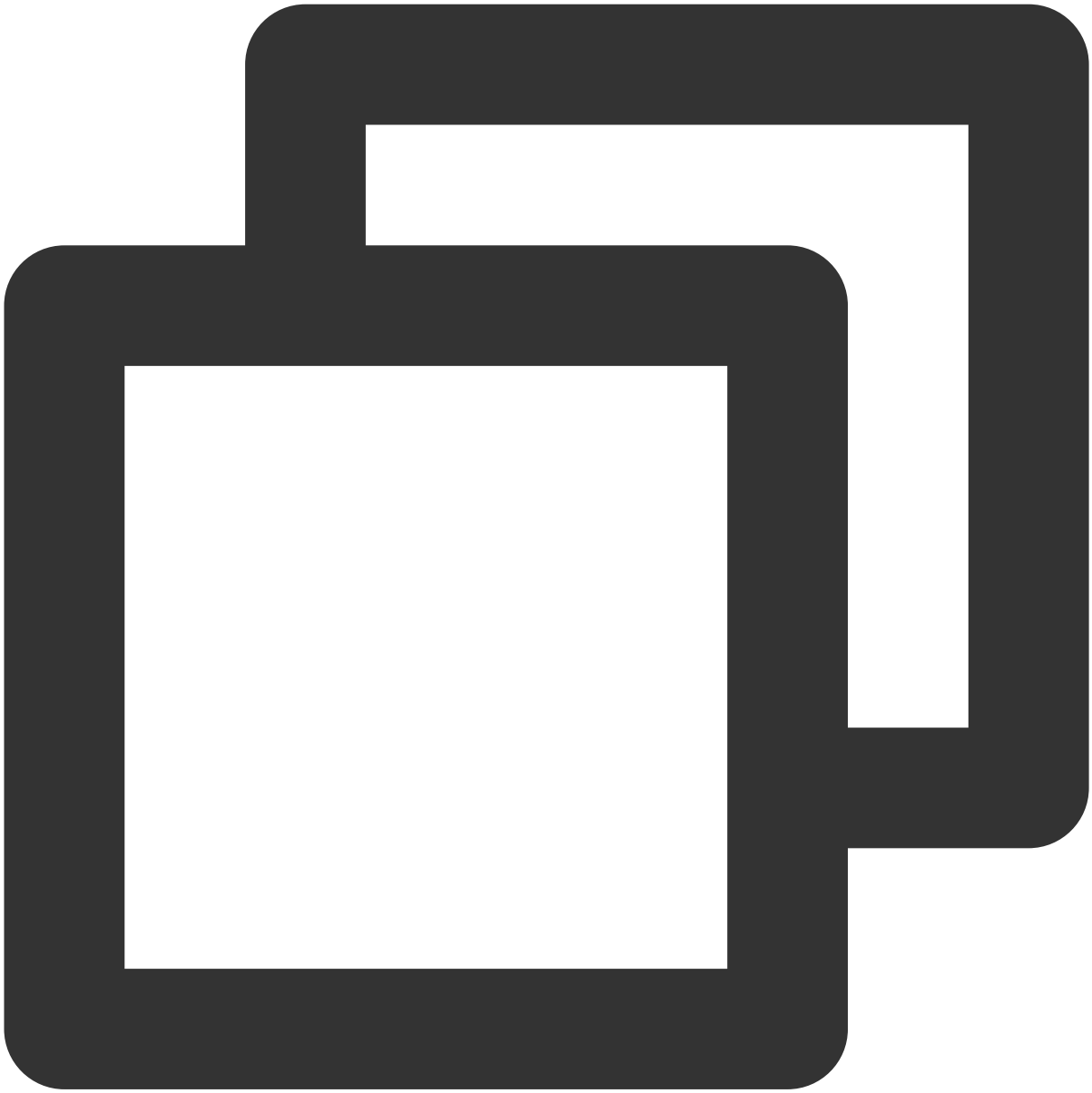
检测本地是否使用了 HTTP 代理。如使用了 HTTP 代理，建议不要使用 HTTPDNS 做域名解析。
检测是否使用了 HTTP 代理：



```
- (BOOL)isUseHTTPProxy {
    CFDictionaryRef dicRef = CFNetworkCopySystemProxySettings();
    const CFStringRef proxyCFstr = (const CFStringRef)CFDictionaryGetValue(dicRef,
    NSString *proxy = (__bridge NSString *)proxyCFstr;
    if (proxy) {
        return YES;
    } else {
        return NO;
```

```
    }
}
```

检测是否使用了 HTTPS 代理：



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

```
            return NO;
        }
    }
```

IOS SDK API 接口

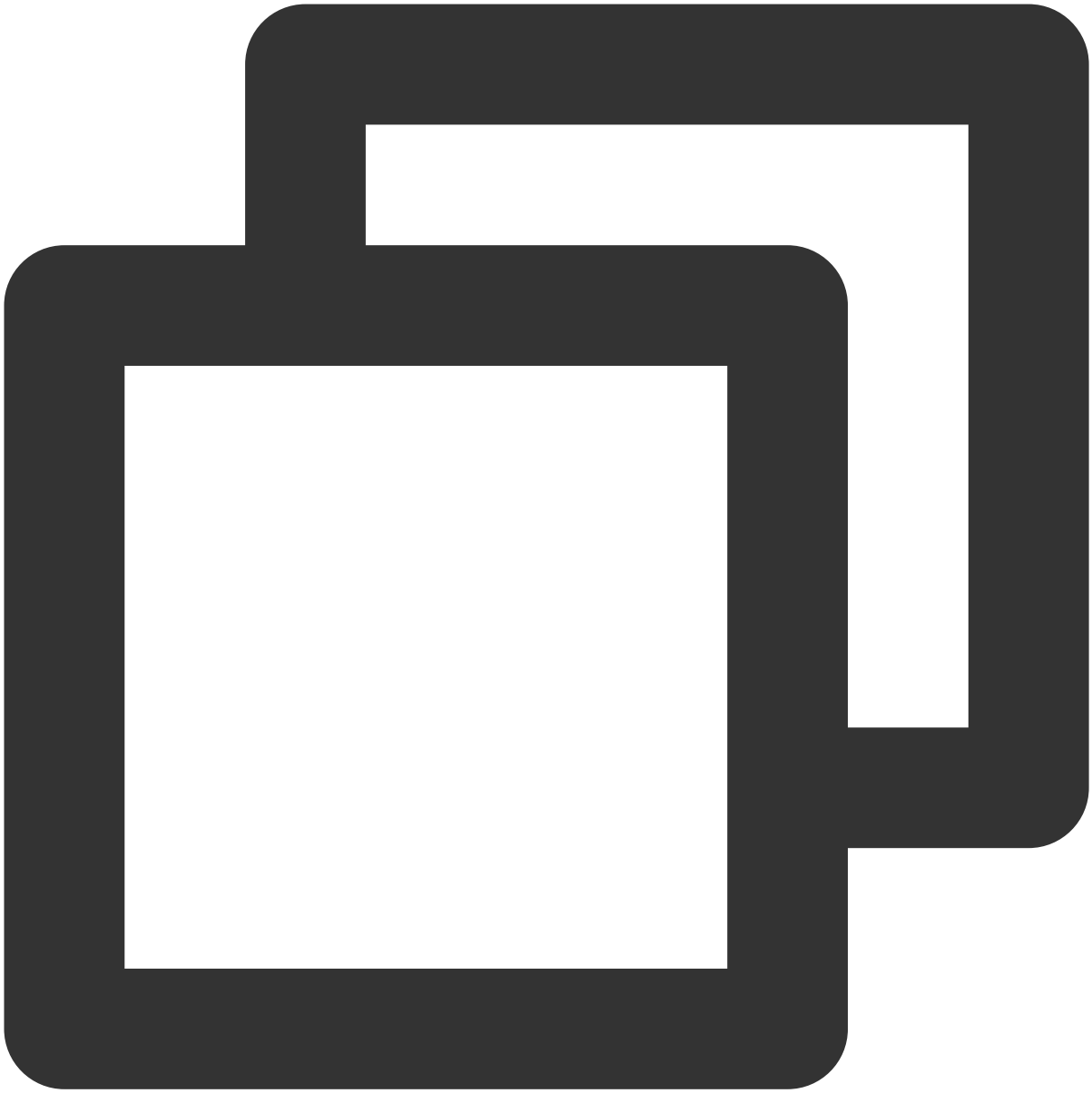
最近更新时间：2023-07-11 10:51:58

设置业务基本信息

类型定义

说明：

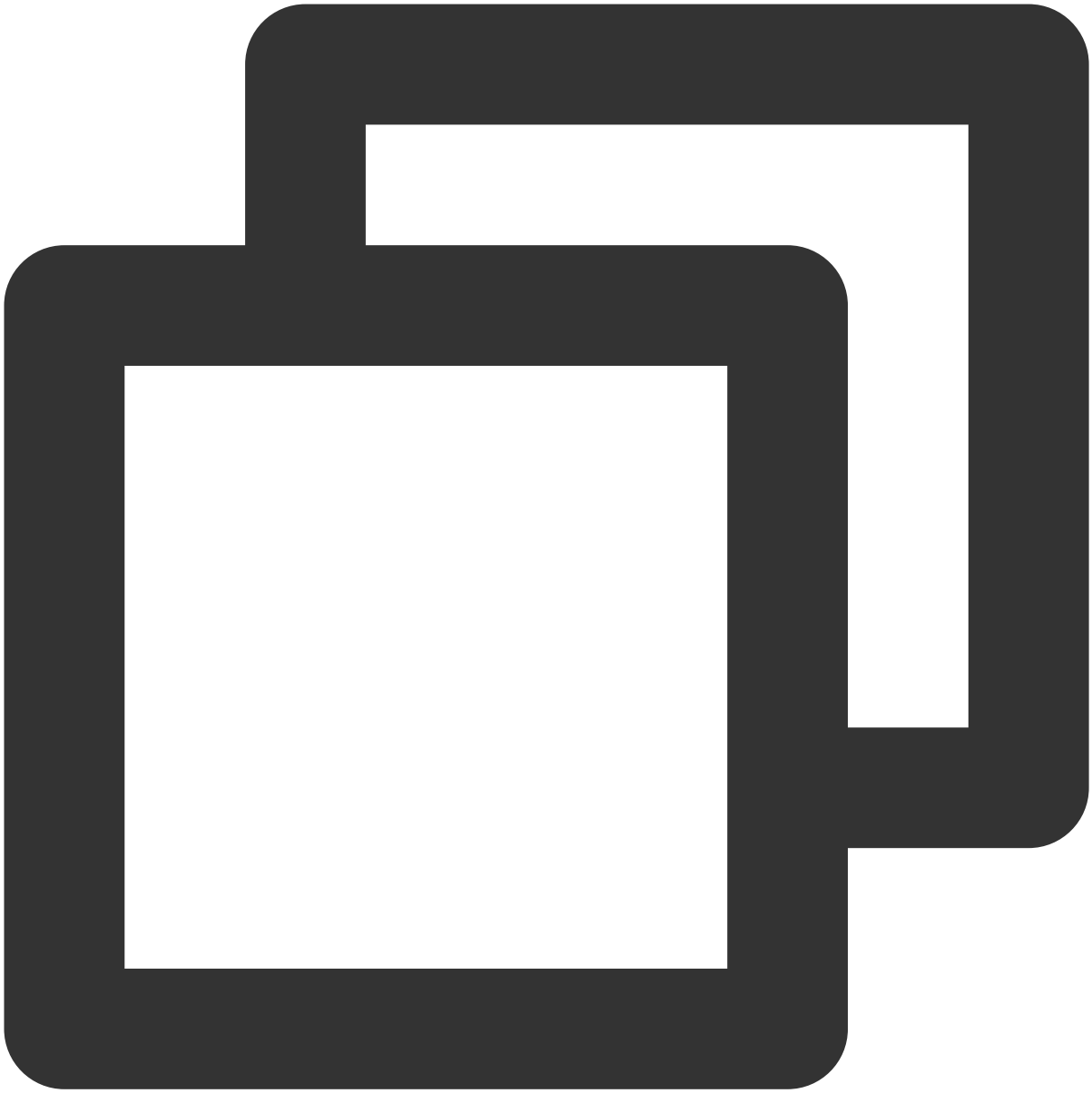
【V1.7.0 废弃】sdk日志上报能力由控制台控制。



```
/**
    加密方式
**/
typedef enum {
    HttpDnsEncryptTypeDES = 0, // DES 加密
    HttpDnsEncryptTypeAES = 1, // AES 加密
    HttpDnsEncryptTypeHTTPS = 2 // HTTPS 加密
} HttpDnsEncryptType;

/**
     IP地址类型
```

```
**/
typedef enum {
    HttpDnsAddressTypeAuto = 0, // sdk自动检测
    HttpDnsAddressTypeIPv4 = 1, // 只支持ipv4
    HttpDnsAddressTypeIPv6 = 2, // 只支持ipv6
    HttpDnsAddressTypeDual = 3, // 支持双协议栈
} HttpDnsAddressType;

/**
    配置结构体
    以下鉴权信息可在腾讯云控制台（https://console.tencentcloud.com/httpdns/configure）开通
**/
typedef struct DnsConfigStruct {
    NSString* appId; // 可选，应用ID，腾讯云控制台申请获得，用于数据上报
    int dnsId; // 授权ID，腾讯云控制台申请后可直接在控制台查看
    NSString* dnsKey; // 加密密钥，加密方式为 AES、DES 时必传。腾讯云控制台申请后可直接在控制
    NSString* token; // 加密 token，加密方式为 HTTPS 时必传
    NSString* dnsIp; //【v1.8.0及以上SDK内部调度，无需设置】HTTPDNS 服务器 IP。HTTP 协议服
    BOOL debug; // 是否开启Debug日志，YES：开启，NO：关闭。建议联调阶段开启，正式上线前关闭
    int timeout; // 可选，超时时间，单位ms，如设置0，则使用默认值2000ms
    HttpDnsEncryptType encryptType; // 控制加密方式
    HttpDnsAddressType addressType; // 指定返回的ip地址类型，默认为 HttpDnsAddressTypeAu
    NSString* routeIp; // 可选，DNS 请求的 ECS（EDNS-Client-Subnet）值，默认情况下 HTTPD
    BOOL httpOnly;// 可选，是否仅返回 httpDns 解析结果。默认 false，即当 httpDns 解析失败时
    NSUInteger retryTimesBeforeSwitchServer; // 可选，切换ip之前重试次数，默认3次
    NSUInteger minutesBeforeSwitchToMain; // 可选，设置切回主ip间隔时长，默认10分钟
    BOOL enableReport; //【V1.7.0 废弃】sdk日志上报能力由控制台控制
} DnsConfig;
```

接口声明



```
/**
 设置业务基本信息（腾讯云业务使用）

 @param config 业务配置结构体
 @return YES:设置成功 NO:设置失败
 */
- (BOOL) initConfig:(DnsConfig *)config;

/**
 * 通过 Dictionary 配置，字段参考 DnsConfig 结构，用于兼容 swift 项目，解决 swift 项目中无法
 *
```

```
 * @param config  配置
 * @return YES：设置成功 NO：设置失败
 */
- (BOOL) initConfigWithDictionary:(NSDictionary *)config;

/**
 * 预解析域名。建议不要设置太多预解析域名，当前限制为最多 8 个域名。仅在初始化时触发。
 * 示例代码：[[MSDKDns sharedInstance] WGSetPreResolvedDomains:@[@"dnspod.com", @"dns
 * @param domains  域名数组
 */
- (void) WGSetPreResolvedDomains:(NSArray *)domains;

/**
 * 解析缓存自动刷新，以数组形式进行配置。当前限制为最多 8 个域名。
 * 示例代码：[[MSDKDns sharedInstance] WGSetKeepAliveDomains:@[@"dnspod.com", @"dnspo
 * @param domains  域名数组
 */
- (void) WGSetKeepAliveDomains:(NSArray *)domains;


/**
 * 启用IP优选，设置域名对应的端口号，对域名解析返回的IP列表进行IP探测，对返回的列表进行动态排序，以
 */
- (void) WGSetIPRankData:(NSDictionary *)IPRankData;

/**
 * 设置是否允许返回TTL过期域名的IP，默认关闭
 * 设置为true时，会直接返回缓存的解析结果，没有缓存则返回0。且在无缓存结果或缓存已过期时，会异步发
 */
- (void) WGSetExpiredIPEnabled:(BOOL)enable;

/**
 * 设置是否启用本地持久化缓存功能，默认关闭
 */
- (void) WGSetPersistCacheIPEnabled:(BOOL)enable;
```

注意

HTTPDNS SDK 提供多重解析优化策略，建议根据实际情况选配，也可以组合使用，可使得解析成功率达到最优效果。

可以通过配置 `(void) WGSetExpiredIPEnabled:(true)enable;` 和 `(void) WGSetPersistCacheIPEnabled:(true)enable;` 来实现乐观 DNS 缓存。

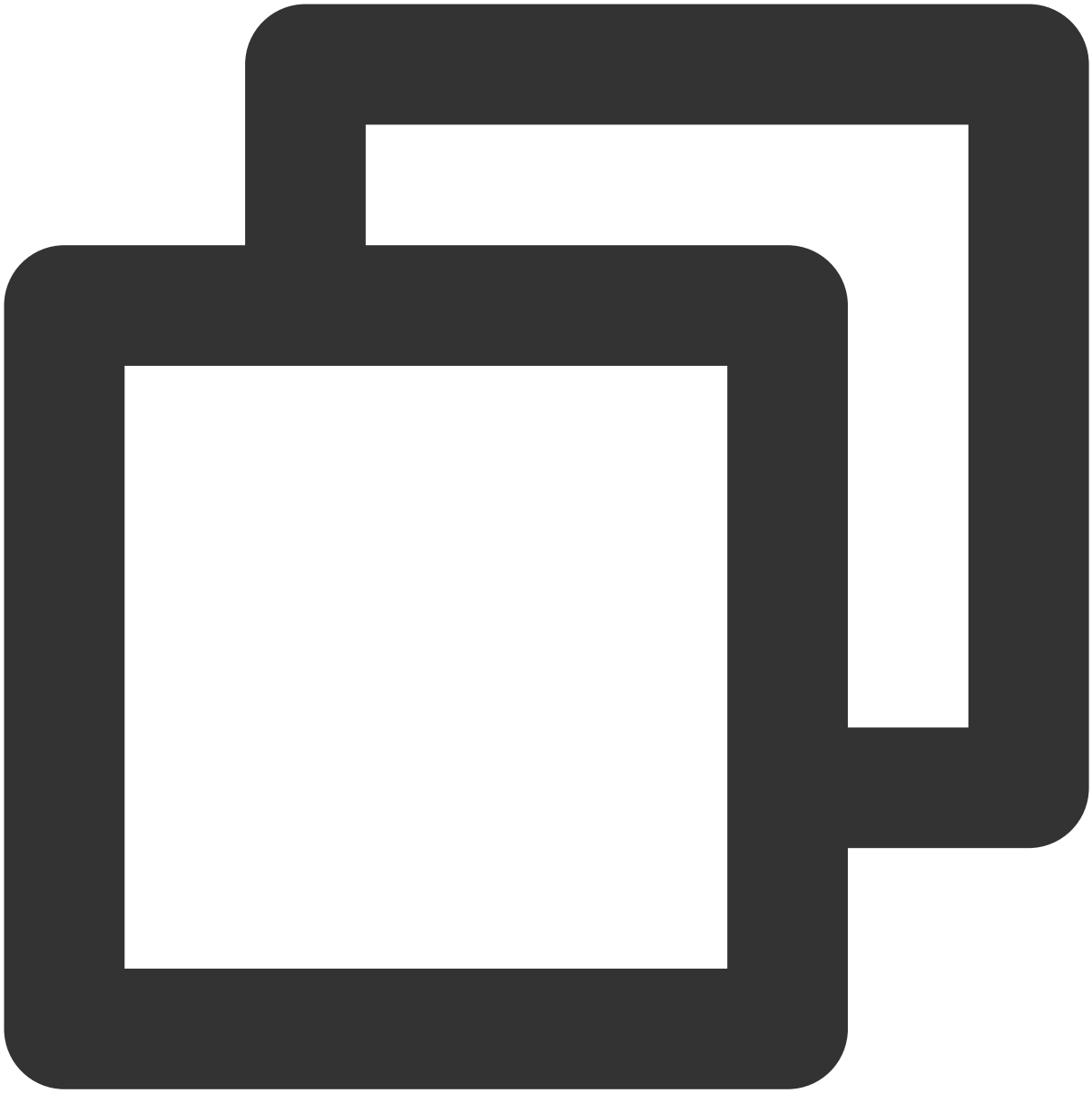
该功能旨在提升缓存命中率和首屏加载速度。持久化缓存会将上一次解析结果保持在本地，在 App 启动时，会优先读取到本地缓存解析结果。

存在使用缓存 IP 时为过期 IP（TTL 过期），该功能启用了允许使用过期 IP，乐观的推定 TTL 过期，大多数情况下该 IP 仍能正常使用。优先返回缓存的过期结果，同时异步发起解析服务，更新缓存。

乐观 DNS 缓存在首次解析域名（无缓存）时，无法命中缓存，返回0;0，同时也会异步发起解析服务，更新缓存。
在启用该功能后需自行 LocalDNS 兜底。核心域名建议配置预解析服务 `(void) WGSetPreResolvedDomains:(NSArray *)domains;` 。
如果业务服务器 IP 变化比较频繁，务必启用缓存自动刷新 `(void) WGSetKeepAliveDomains:(NSArray *)domains;` 、预解析能力 `(void) WGSetPreResolvedDomains:(NSArray *)domains;` ，以确保解析结果的准确性。

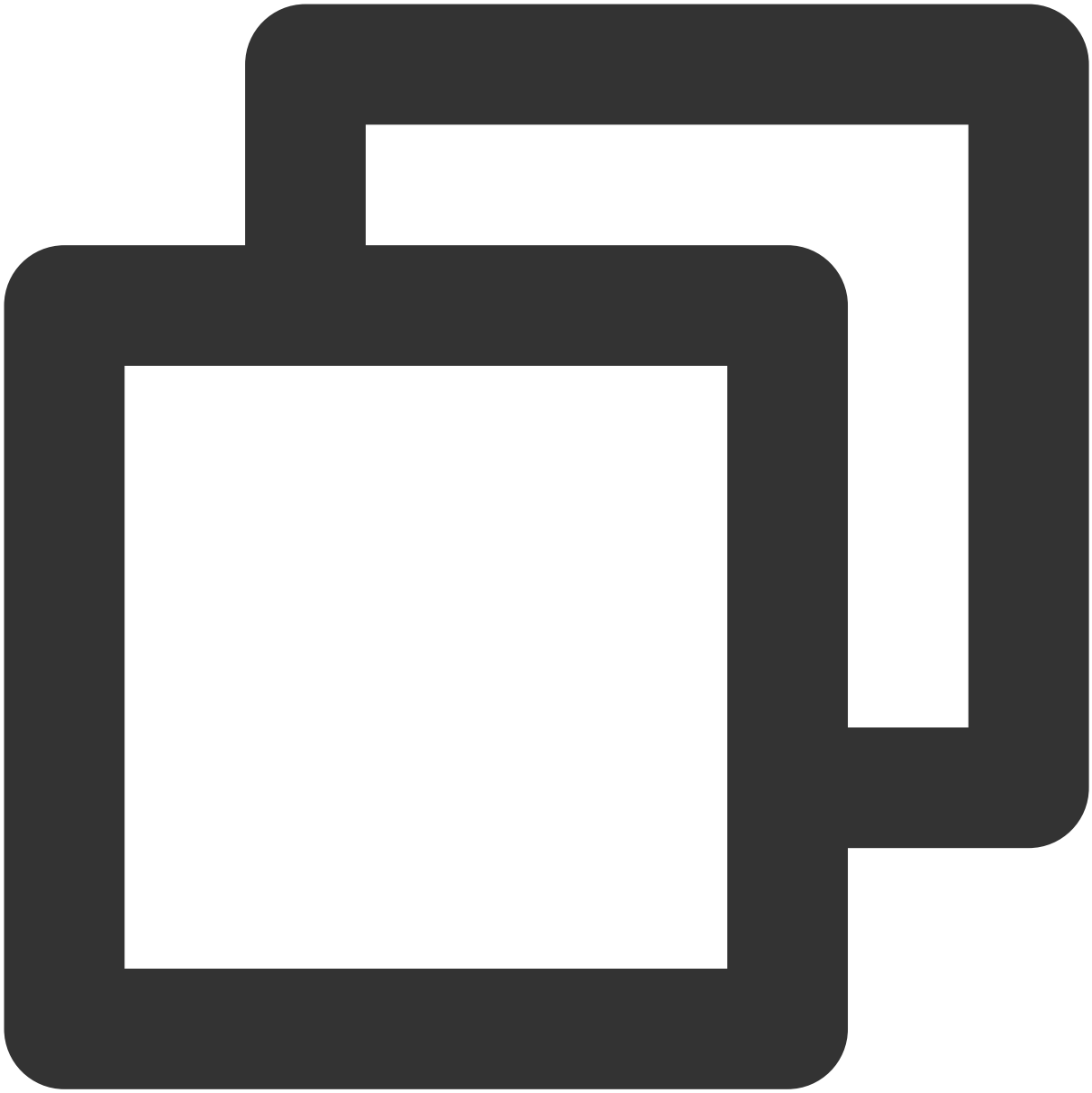
示例代码

接口调用示例：
在 Objective-C 项目中。



```
DnsConfig *config = new DnsConfig();
config->dnsIp = @"HTTPDNS 服务器IP";
config->dnsId = dns授权id;
config->dnsKey = @"加密密钥";
config->encryptType = HttpDnsEncryptTypeDES;
config->debug = YES;
config->timeout = 2000;
[[MSDKDns sharedInstance] initConfig: config];
```

在 Swift 项目中。



```
let msdkDns = MSDKDns.sharedInstance() as? MSDKDns;
msdkDns?.initConfig(with: [
      "dnsIp": "HTTPDNS 服务器IP",
      "dnsId": "dns授权id",
      "dnsKey": "加密密钥",
      "encryptType": 0, // 0 -> des, 1 -> aes, 2 -> https
]);
```

域名解析接口

获取 IP 共有以下四个接口，引入头文件，调用相应接口即可。

同步接口

单个查询 **WGGetHostByName:**；

批量查询（返回单个 IP）**WGGetHostsByNames:**；

批量查询（返回所有 IP）**WGGetAllHostsByNames:**；

注意

同步接口会阻塞，建议在**子线程中调用**或者切换为**异步接口**。

异步接口

单个查询 **WGGetHostByNameAsync:returnIps:**；

批量查询 (返回单个 IP）**WGGetHostsByNamesAsync:returnIps:**；

批量查询（返回所有 IP）**WGGetAllHostsByNamesAsync:returnIps:**；

返回的地址格式如下：

单个查询：单个查询接口返回 NSArray，固定长度为2，其中第一个值为 IPv4 地址，第二个值为 IPv6 地址。以下为返回格式的详细说明：

IPv4 下，仅返回 IPv4 地址，即返回格式为：[ipv4, 0]。

IPv6 下，仅返回 IPv6 地址，即返回格式为：[0, ipv6]。

双栈网络下，返回解析到 IPv4&IPv6（如果存在）地址，即返回格式为：[ipv4, ipv6]。

解析失败，返回[0, 0]，业务重新调用 WGGetHostByName 接口即可。

批量查询（返回单个 IP）：批量查询接口返回 NSDictionary，key 为查询的域名，value 为 NSArray，固定长度为2，其他第一个值为 IPv4 地址，第二个值为 IPv6 地址。以下为返回格式的详细说明：

IPv4 下，仅返回 IPv4 地址，即返回格式为：{"queryDomain" : [ipv4, 0]}。

IPv6 下，仅返回 IPv6 地址，即返回格式为：{"queryDomain" : [0, ipv6]}。

双栈网络下，返回解析到 IPv4&IPv6（如果存在）地址，即返回格式为：{"queryDomain" : [ipv4, ipv6]}。

解析失败，返回{"queryDomain" : [0, 0]}，业务重新调用 WGGetHostsByNames 接口即可。

批量查询（返回所有 IP）：批量查询接口返回 NSDictionary，key 为查询的域名，value 为 NSDictionary，包含两个 key（ipv4、ipv6），对应的 value 为 NSArray 对象，表示所有的ipv4/ipv6 解析结果 IP。以下为返回格式的详细说明：

返回格式为：{"queryDomain" : { "ipv4": [], "ipv6": []}}。

如何提高IPv6使用率

使用 IPv6 地址进行 URL 请求时，需添加方框号[ ]进行处理，例如：`http://[64:ff9b::b6fe:7475]/`。

如 IPv6 地址为0，则直接使用 IPv4 地址连接。

如 IPv4 地址为0，则直接使用 IPv6 地址连接。

如 IPv4 和 IPv6 地址都不为0，则由客户端决定优先使用哪个地址进行连接，但优先地址连接失败时应切换为另一个地址。

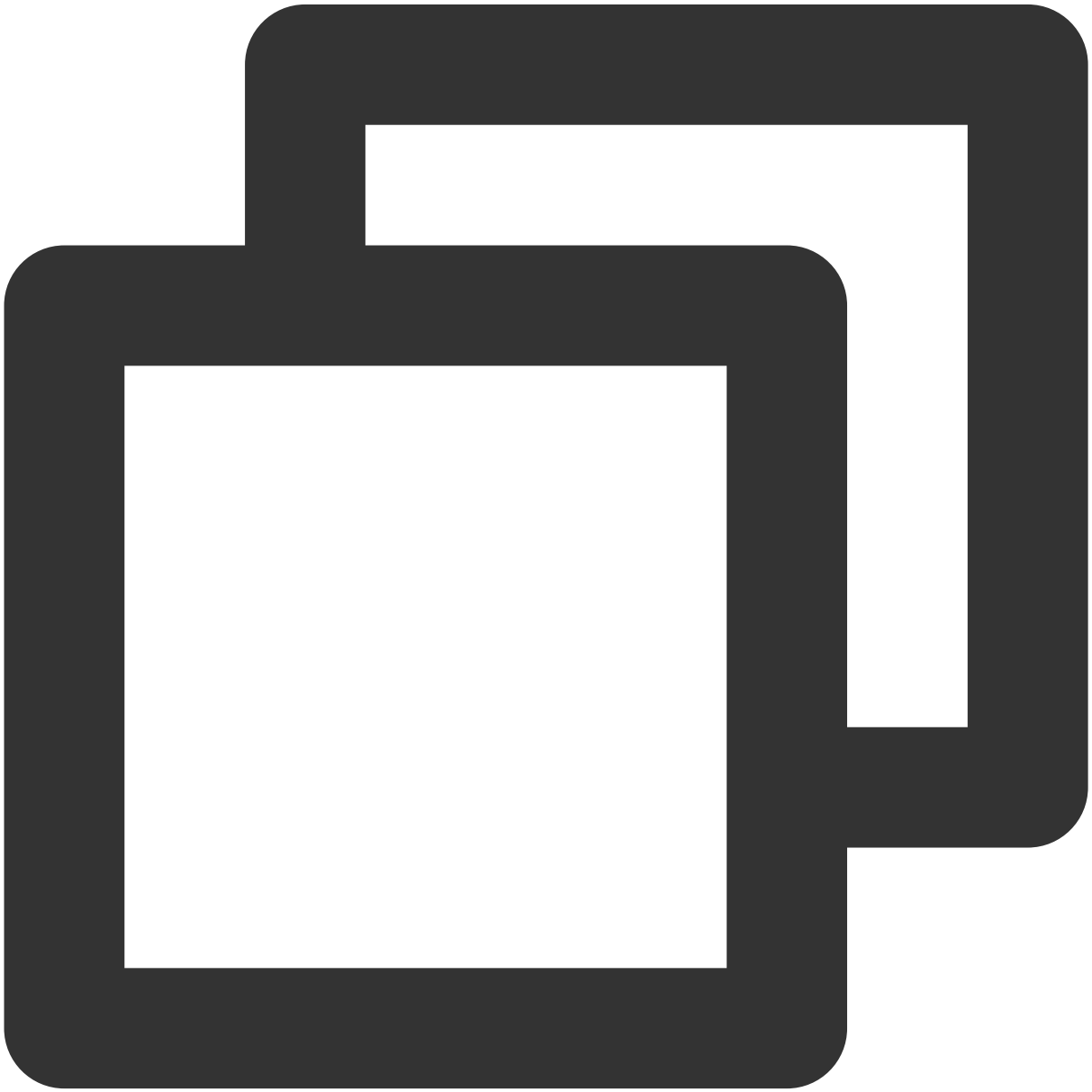
使用 SDK 方式接入 HTTPDNS，若 HTTPDNS 未查询到解析结果，则通过 LocalDNS 进行域名解析，返回 LocalDNS 的解析结果。

同步解析接口

接口名称

WGGetHostByName、WGGetHostsByNames

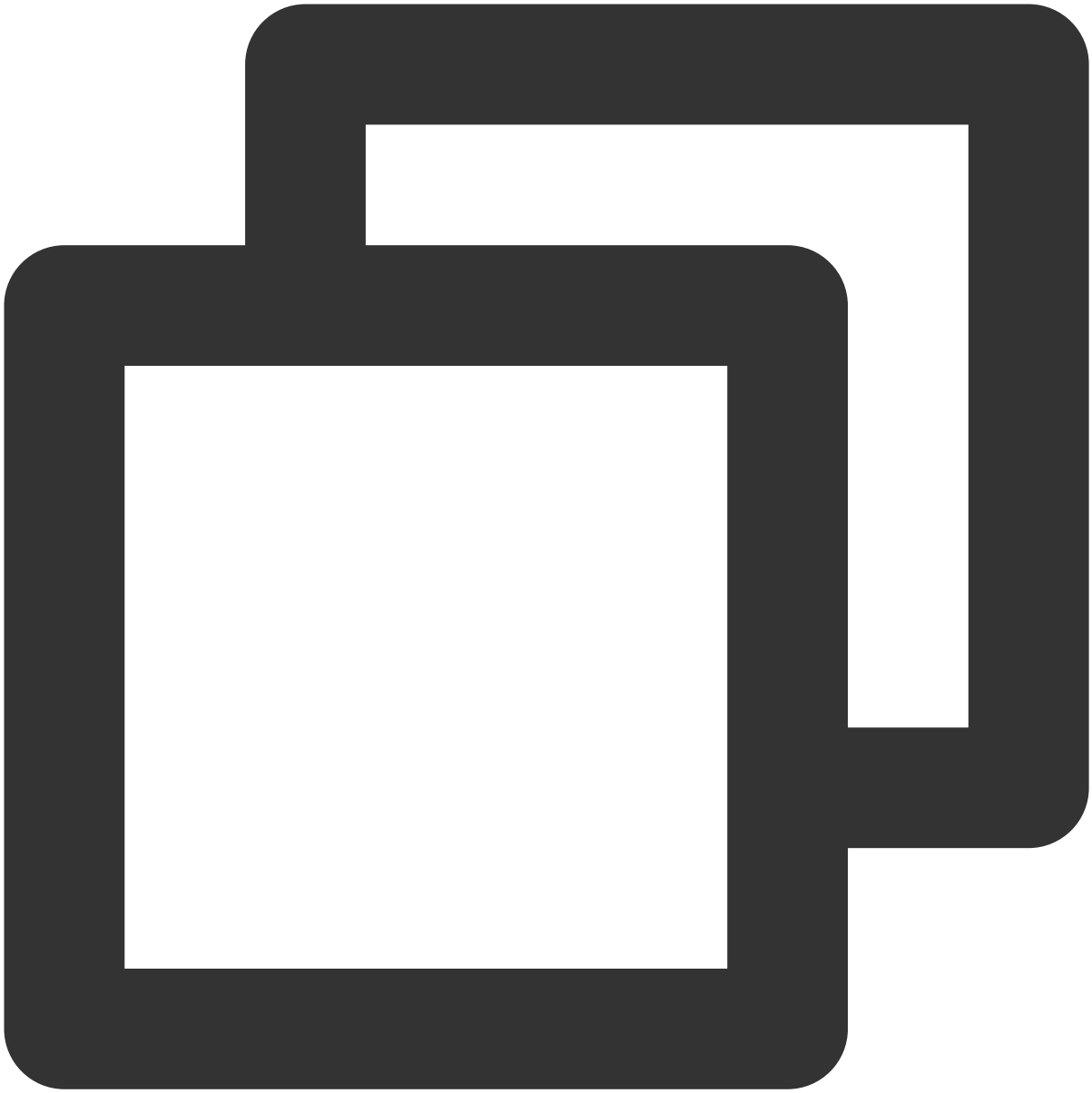
接口声明



```
/**
 域名同步解析（通用接口）
 @param domain 域名
 @return 查询到的 IP 数组，超时（1s）或者未查询到返回[0,0]数组
*/
- (NSArray *) WGGetHostByName:(NSString *) domain;

/**
 域名批量同步解析（通用接口）
 @param domains 域名数组
 @return 查询到的 IP 字典
 */
- (NSDictionary *) WGGetHostsByNames:(NSArray *) domains;
```

示例代码

接口调用示例：



```
// 单个域名查询
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 使用 IPv6 地址进行 URL 连接时，注意格式，IPv6 需加方框号[]进行处理，例如：http
    } else if (![ipv4 isEqualToString:@"0"]){
        //使用 IPv4 地址进行连接
    } else {
        //异常情况返回为0,0，建议重试一次
```

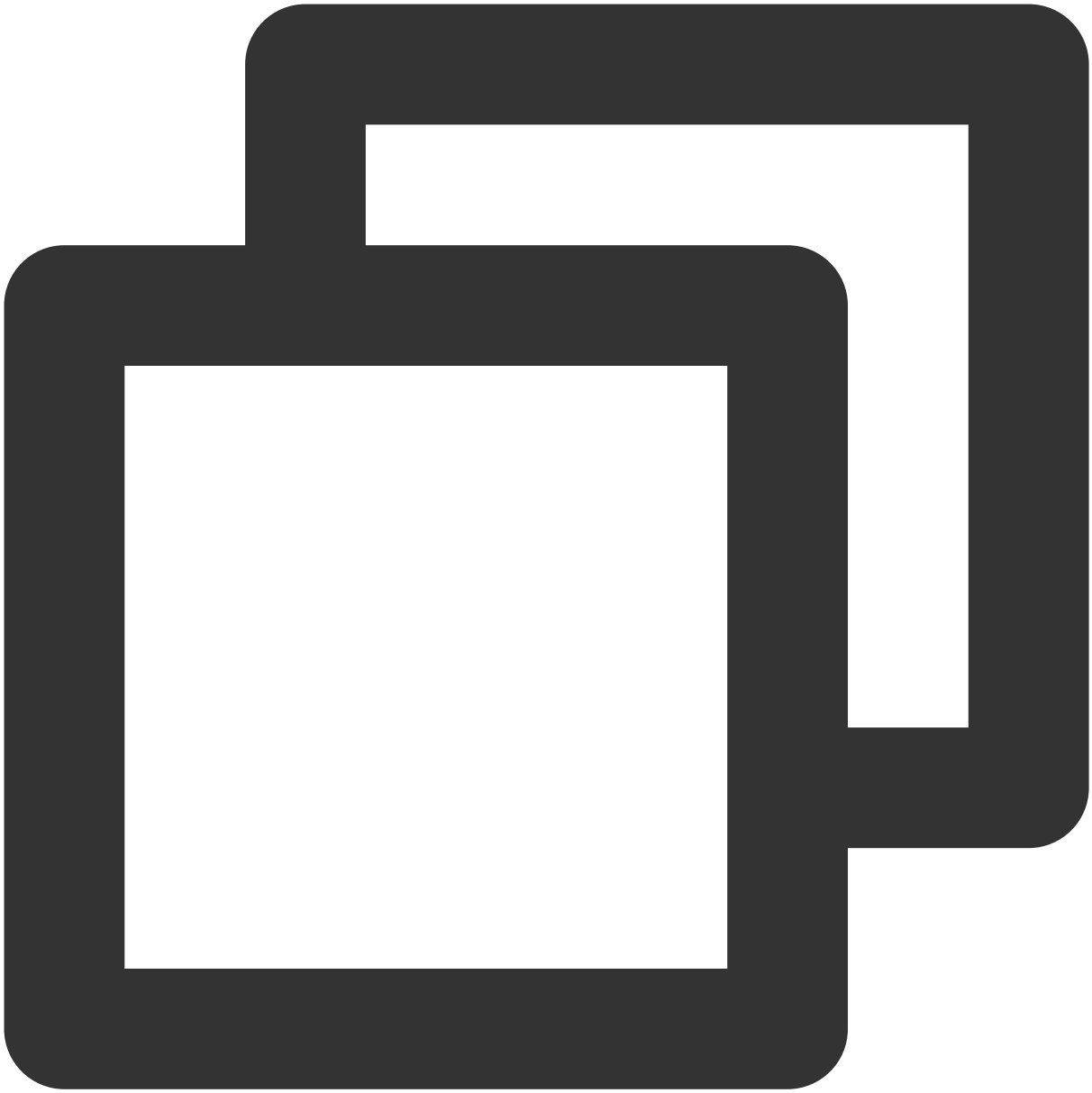
```
    }
}

// 批量域名查询
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 使用 IPv6 地址进行 URL 连接时，注意格式，IPv6 需加方框号[]进行处理，例如：htt
    } else if (![ipv4 isEqualToString:@"0"]){
        //使用 IPv4 地址进行连接
    } else {
        //异常情况返回为0,0，建议重试一次
    }
}
```

异步解析接口

接口名称

WGGetHostByNameAsync、WGGetHostsByNamesAsync

接口声明



```
/**
 域名异步解析（通用接口）
 @param domain  域名
 @param handler 返回查询到的 IP 数组，超时（1s）或者未查询到返回[0,0]数组
 */
 - (void) WGGetHostByNameAsync:(NSString *) domain returnIps:(void (^)(NSArray *ips

 /**
 域名批量异步解析（通用接口）

 @param domains  域名数组
```

```
 @param handler 返回查询到的IP字典，超时（1s）或者未查询到返回 {"queryDomain" : [0, 0] ...
 */
- (void) WGGetHostsByNamesAsync:(NSArray *) domains returnIps:(void (^)(NSDictionar
```

示例代码

注意

业务可根据自身需求，任选一种调用方式。

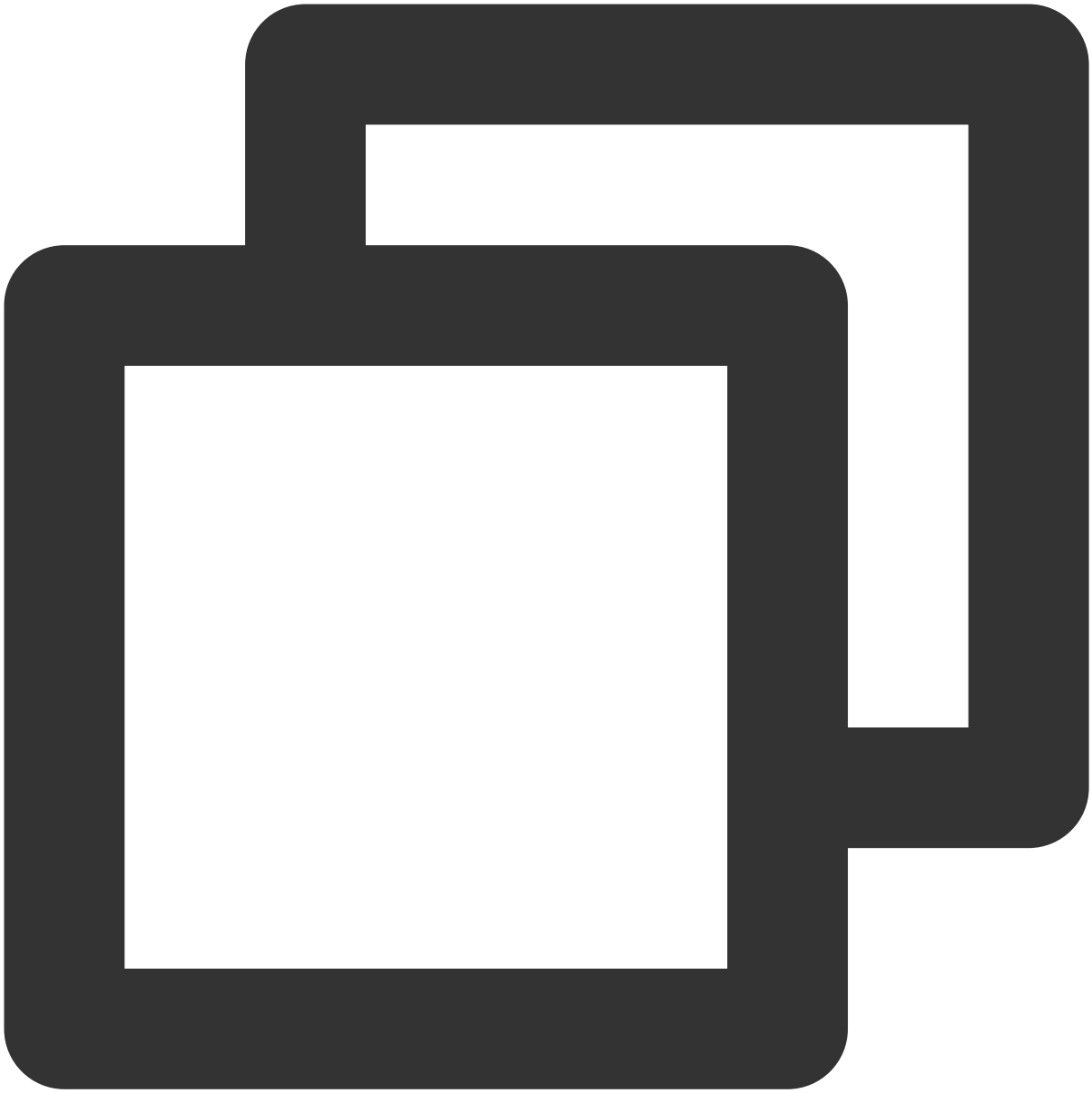
示例1

示例2

等待完整解析过程结束后，拿到结果，进行连接操作。

优点：可保证每次请求都能拿到返回结果进行接下来的连接操作。

缺点：异步接口的处理较同步接口稍显复杂。



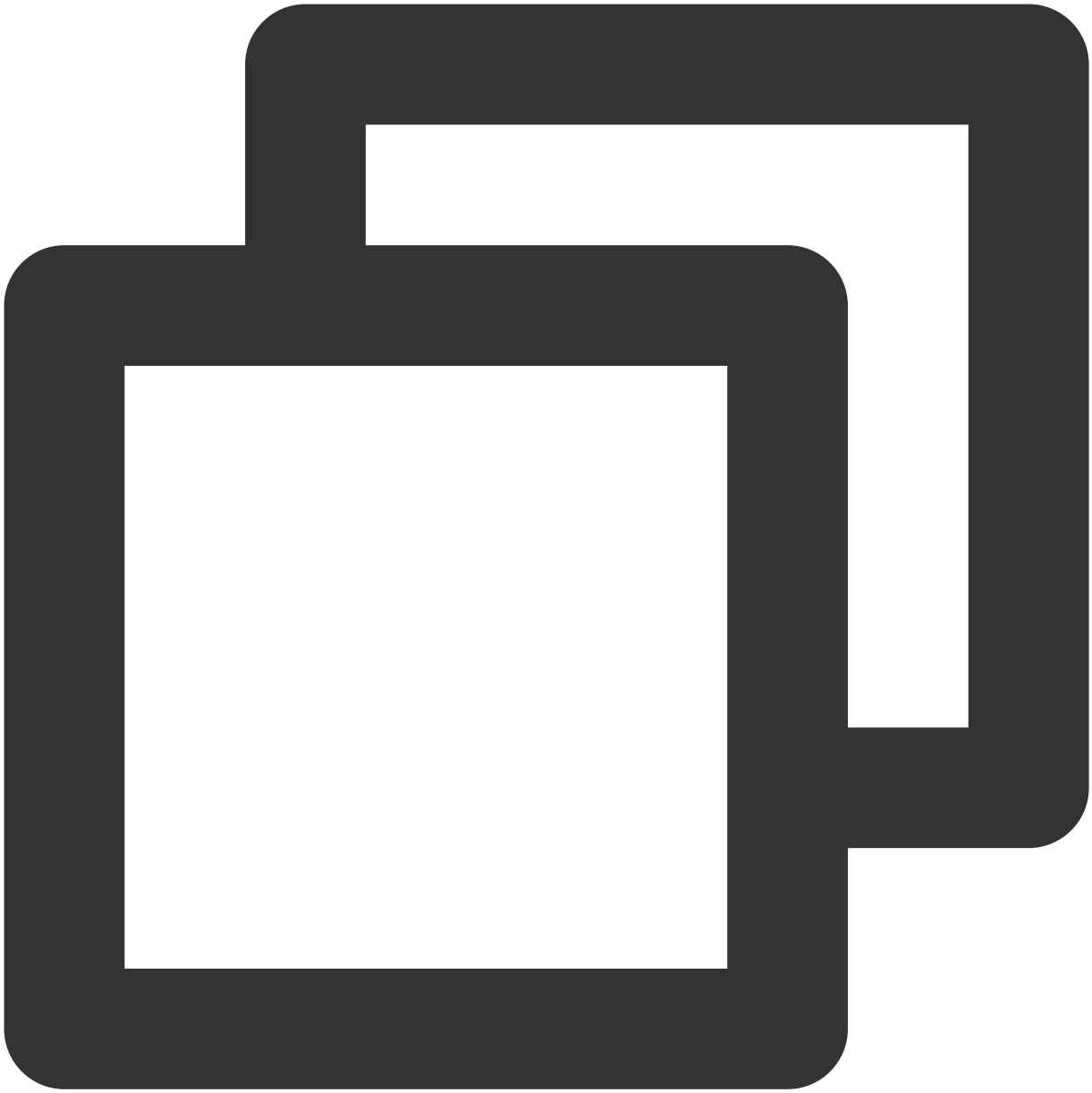
```
// 单个域名查询
[[MSDKDns sharedInstance] WGGetHostByNameAsync:@"qq.com" returnIps:^(NSArray *ipsAr
    //等待完整解析过程结束后，拿到结果，进行连接操作
    if (ipsArray && ipsArray.count > 1) {
        NSString *ipv4 = ipsArray[0];
        NSString *ipv6 = ipsArray[1];
        if (![ipv6 isEqualToString:@"0"]) {
            //使用建议：当 IPv6 地址存在时，优先使用ipv6地址
            //TODO 使用 IPv6 地址进行 URL 连接时，注意格式，IPv6 需加方框号[]进行处理，例如：
        } else if (![ipv4 isEqualToString:@"0"]){
            //使用 IPv4 地址进行连接
```

```
        } else {
            //异常情况返回为0,0，建议重试一次
        }
    }
}];
// 批量域名查询
[[MSDKDns sharedInstance] WGGetHostsByNamesAsync:@[@"qq.com", @"dnspod.cn"] returnI
    //等待完整解析过程结束后，拿到结果，进行连接操作
    NSArray *ips = [ipsDict objectForKey: @"qq.com"];
    if (ips && ips.count > 1) {
        NSString *ipv4 = ips[0];
        NSString *ipv6 = ips[1];
        if (![ipv6 isEqualToString:@"0"]) {
            //使用建议：当ipv6地址存在时，优先使用ipv6地址
            //TODO 使用 IPv6 地址进行 URL 连接时，注意格式，IPv6 需加方框号[]进行处理，例如：
        } else if (![ipv4 isEqualToString:@"0"]){
            //使用 IPv4 地址进行连接
        } else {
            //异常情况返回为0,0，建议重试一次
        }
    }
}];
```

无需等待，可直接拿到缓存结果，如无缓存，则 result 为 nil。

优点：对于解析时间有严格要求的业务，使用本示例，可无需等待，直接拿到缓存结果进行后续的连接操作，完全避免了同步接口中解析耗时可能会超过 100ms 的情况。

缺点：第一次请求时，result 一定会 nil，需业务增加处理逻辑。



```
__block NSArray* result;
[[MSDKDns sharedInstance] WGGetHostByNameAsync:domain returnIps:^(NSArray *ipsArray
    result = ipsArray;
}];
//无需等待，可直接拿到缓存结果，如无缓存，则 result 为 nil
if (result) {
    //拿到缓存结果，进行连接操作
} else {
    //本次请求无缓存，业务可走原始逻辑
}
```

IOS SDK 最佳实践
HTTPS（非 SNI）场景

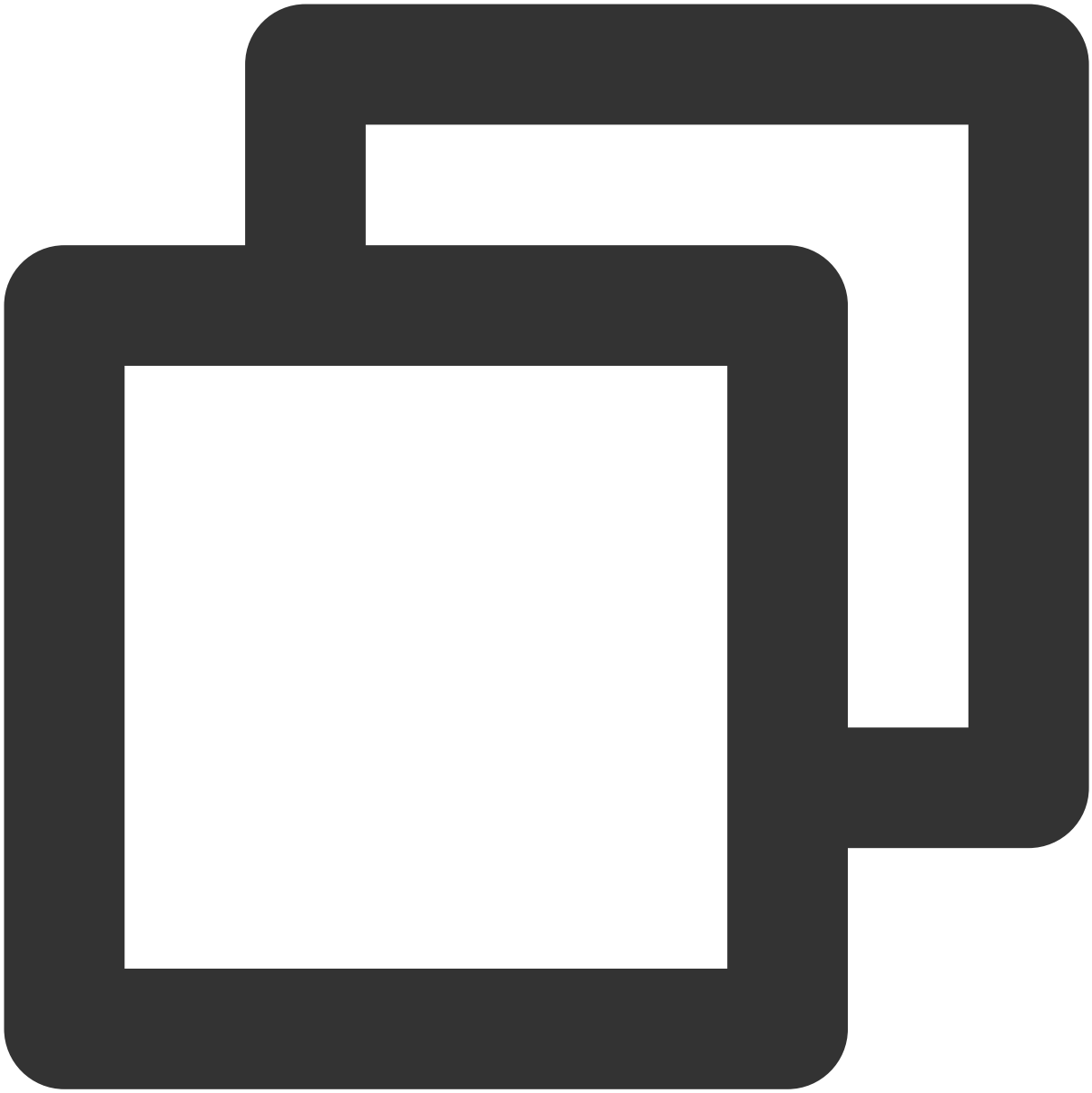
最近更新时间：2023-06-12 14:49:11

原理

在进行证书校验时，将 IP 替换成原来的域名，再进行证书验证。

Demo 示例

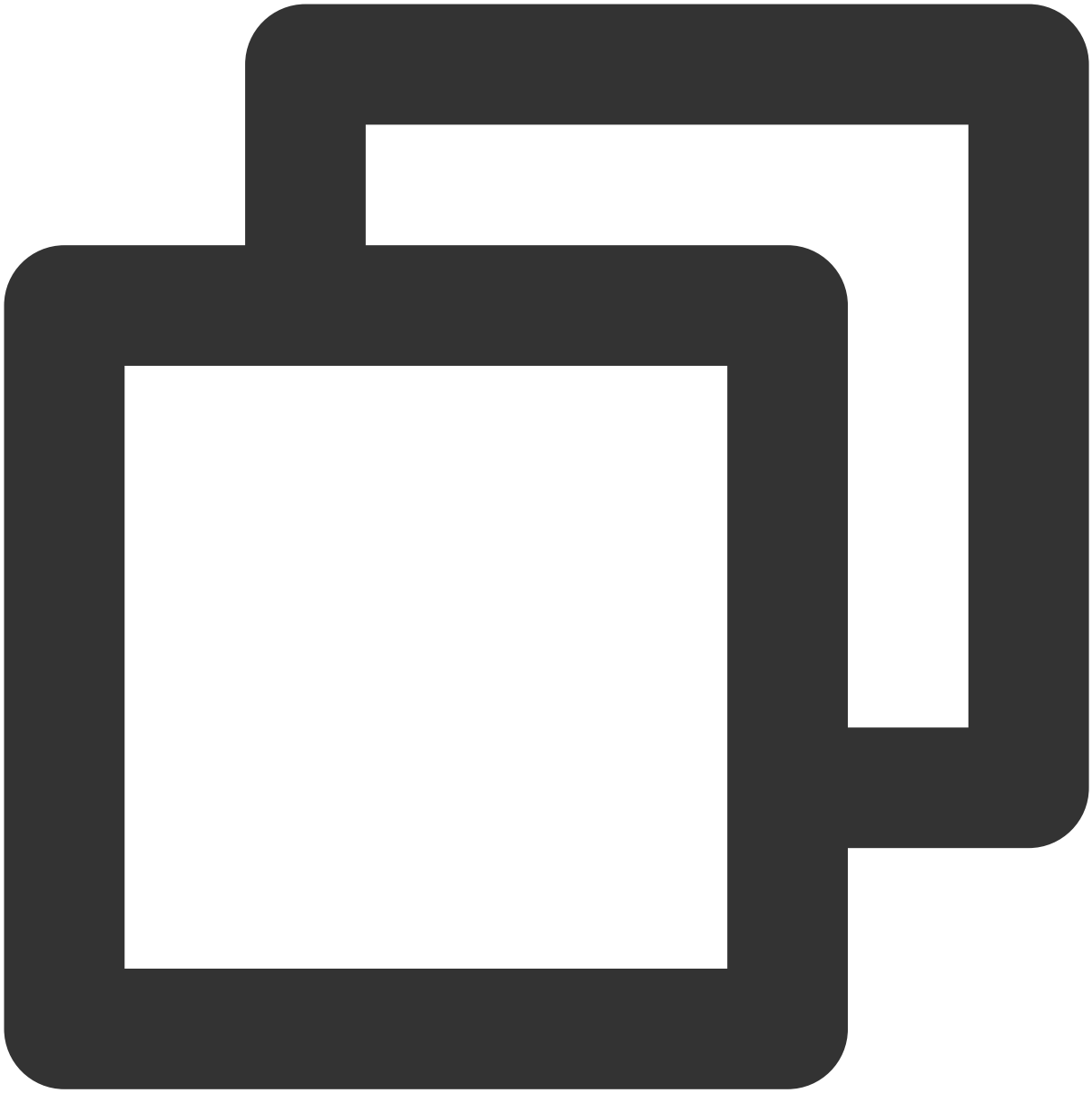
NSURLConnection 接口示例



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

    //创建证书校验策略
    NSMutableArray *policies = [NSMutableArray array];
    if (domain) {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateSSL(true, (__bridg
    } else {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateBasicX509()];
    }
```

```
    //绑定校验策略到服务端的证书上
    SecTrustSetPolicies(serverTrust, (__bridge CFArrayRef)policies);

    //评估当前 serverTrust 是否可信任,
    //官方建议在 result = kSecTrustResultUnspecified 或 kSecTrustResultProceed 的情况下
    //https://developer.apple.com/library/ios/technotes/tn2232/_index.html
    //关于 SecTrustResultType 的详细信息请参考 SecTrust.h
    SecTrustResultType result;
    SecTrustEvaluate(serverTrust, &result);
    return (result == kSecTrustResultUnspecified || result == kSecTrustResultProcee
}

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

    //URL 里面的 host 在使用 HTTPDNS 的情况下被设置成了 IP，此处从 HTTP Header 中获取真实域名
    NSString *host = [[self.request allHTTPHeaderFields] objectForKey:@"host"];
    if (!host) {
        host = self.request.URL.host;
    }

    //判断 challenge 的身份验证方法是否是 NSURLAuthenticationMethodServerTrust（HTTPS 模式
    //在没有配置身份验证方法的情况下进行默认的网络请求流程。
    if ([challenge.protectionSpace.authenticationMethod isEqualToString:NSURLAuthen
        if ([self evaluateServerTrust:challenge.protectionSpace.serverTrust forDoma

            //验证完以后，需要构造一个 NSURLCredential 发送给发起方
            NSURLCredential *credential = [NSURLCredential credentialForTrust:chall
            [[challenge sender] useCredential:credential forAuthenticationChallenge
        } else {
            //验证失败，取消这次验证流程
            [[challenge sender] cancelAuthenticationChallenge:challenge];
        }
    } else {

        //对于其他验证方法直接进行处理流程
        [[challenge sender] continueWithoutCredentialForAuthenticationChallenge:cha
    }
}
```

NSURLSession 接口示例

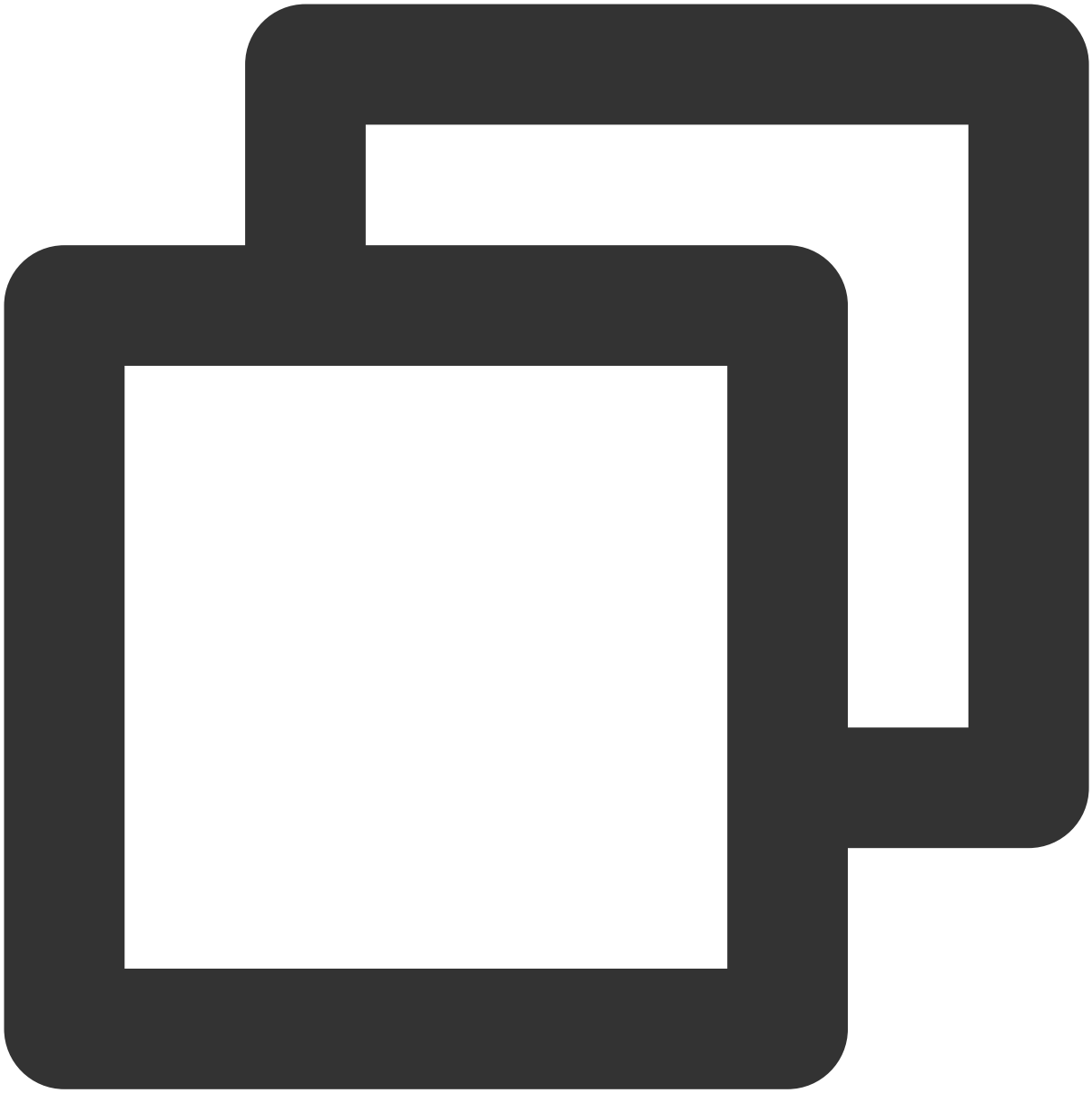


```
 #pragma mark - NSURLSessionDelegate
- (BOOL)evaluateServerTrust:(SecTrustRef)serverTrust forDomain:(NSString *)domain {

    //创建证书校验策略
    NSMutableArray *policies = [NSMutableArray array];
    if (domain) {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateSSL(true, (__bridg
    } else {
        [policies addObject:(__bridge_transfer id)SecPolicyCreateBasicX509()];
    }
```

```
    //绑定校验策略到服务端的证书上
    SecTrustSetPolicies(serverTrust, (__bridge CFArrayRef)policies);

    //评估当前 serverTrust 是否可信任,
    //官方建议在 result = kSecTrustResultUnspecified 或 kSecTrustResultProceed 的情况下
    //https://developer.apple.com/library/ios/technotes/tn2232/_index.html
    //关于SecTrustResultType的详细信息请参考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;

    //获取原始域名信息
    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;
    }

    // 对于其他的 challenges 直接使用默认的验证方案
    completionHandler(disposition,credential);
}
```

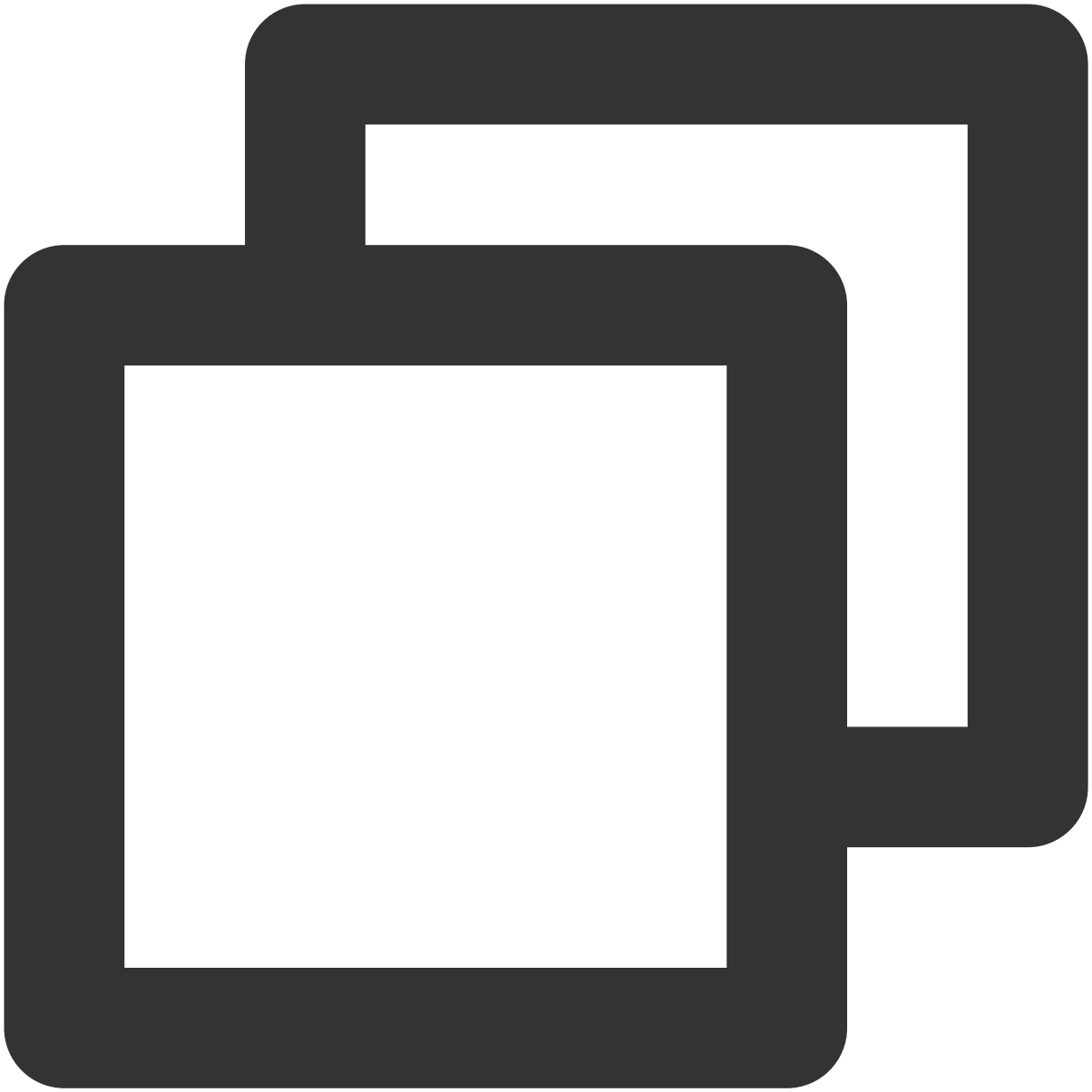
Unity 的 WWW 接口示例

将 Unity 工程导为 Xcode 工程后，打开 Classes/Unity/**WWWConnection.mm** 文件，修改下述代码：



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

调整为：



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

HTTPS（SNI）场景

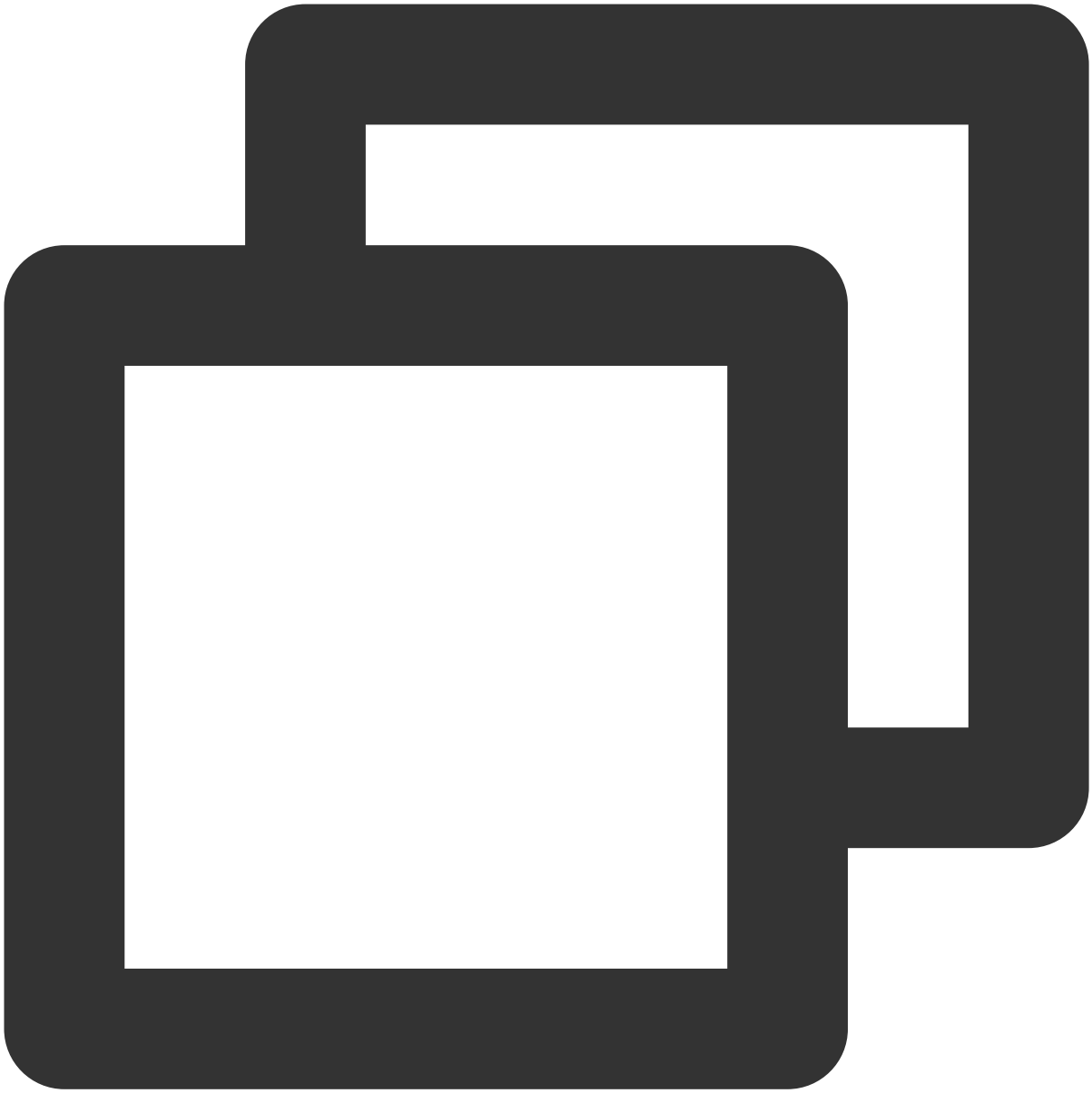
最近更新时间：2023-06-06 10:50:29

原理

SNI（Server Name Indication）是为了解决一个服务器使用多个域名和证书的 SSL/TLS 扩展。工作原理如下：
在连接到服务器建立 SSL 连接之前先发送要访问站点的域名（Hostname）。
服务器根据这个域名返回一个合适的证书。
上述过程中，当客户端使用 HTTPDNS 解析域名时，请求 URL 中的 host 会被替换成 HTTPDNS 解析出来的 IP，导致服务器获取到的域名为解析后的 IP，无法找到匹配的证书，只能返回默认的证书或者不返回，所以会出现 SSL/TLS 握手不成功的错误。
由于 iOS 上层网络库 NSURLConnection/NSURLSession 没有提供接口进行 SNI 字段的配置，因此可以考虑使用 NSURLProtocol 拦截网络请求，然后使用 CFHTTPMessageRef 创建 NSInputStream 实例进行 Socket 通信，并设置其 kCFStreamSSLPeerName 的值。
需要注意的是，使用 NSURLProtocol 拦截 NSURLSession 发起的 POST 请求时，HTTPBody 为空。解决方案有两个：
使用 NSURLConnection 发 POST 请求。
先将 HTTPBody 放入 HTTP Header field 中，然后在 NSURLProtocol 中再取出来。

Demo 示例

在网络请求前注册 NSURLProtocol 子类，在示例的 SNIViewController.m 中。

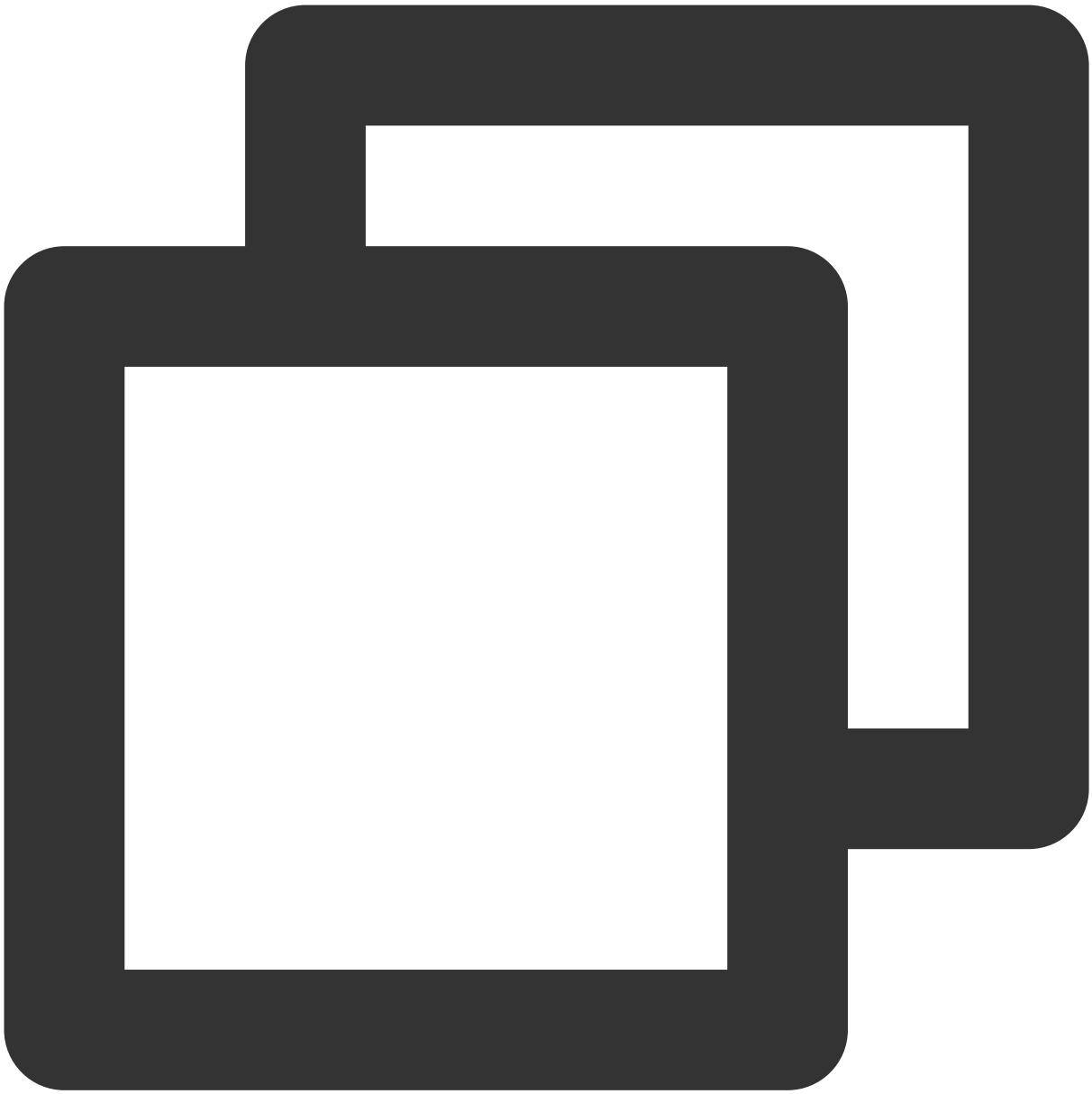


```
// 注册拦截请求的 NSURLProtocol
[NSURLProtocol registerClass:[MSDKDnsHttpMessageTools class]];

// 需要设置 SNI 的 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];
    }
}
// 通过 HTTPDNS 获取 IP 成功，进行 URL 替换和 HOST 头设置
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"];
    }
}

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

// NSURLSession 例子
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];

// 注*：使用 NSURLProtocol 拦截 NSURLSession 发起的 POST 请求时，HTTPBody 为空。
// 解决方案有两个：
// 1. 使用 NSURLConnection 发 POST 请求。
// 2. 先将 HTTPBody 放入 HTTP Header field 中，然后在 NSURLProtocol 中再取出来。
// 下面主要演示第二种解决方案
// 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];
```

使用说明

需调用以下接口设置需要拦截域名或无需拦截的域名：



```
#pragma mark - SNI 场景，仅调用一次即可，请勿多次调用
/**
 SNI 场景下设置需要拦截的域名列表
 建议使用该接口设置，仅拦截 SNI 场景下的域名，避免拦截其它场景下的域名

 @param hijackDomainArray 需要拦截的域名列表
 */
- (void) WGSetHijackDomainArray:(NSArray *)hijackDomainArray;
```

```
/**
 SNI 场景下设置不需要拦截的域名列表

 @param noHijackDomainArray 不需要拦截的域名列表
 */
- (void) WGSetNoHijackDomainArray:(NSArray *)noHijackDomainArray;
```

如设置了需要拦截的域名列表，则仅会拦截处理该域名列表中的 HTTPS 请求，其他域名不做处理。

如设置了不需要拦截的域名列表，则不会拦截处理该域名列表中的 HTTPS 请求。

注意

建议使用 WGSetHijackDomainArray 仅拦截 SNI 场景下的域名，避免拦截其他场景下的域名。

Unity 工程接入

最近更新时间：2023-06-12 14:57:53

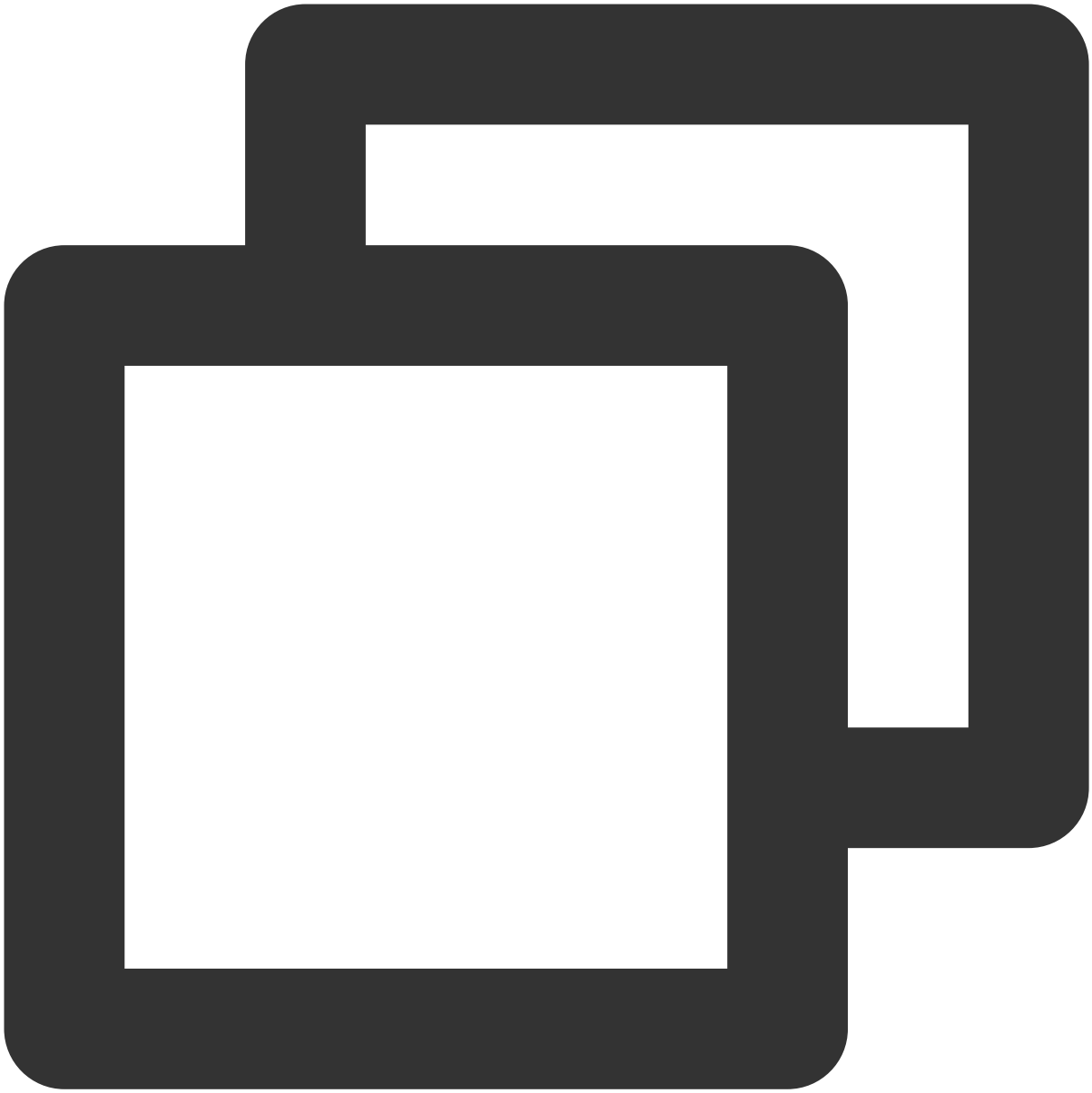
操作步骤

1. 将 `HTTPDNSUnityDemo/Assets/Plugins/Scripts` 下的 `HttpDns.cs` 文件拷贝到 Unity 对应 `Assets/Plugins/Scripts` 路径下。
2. 在需要进行域名解析的部分，调用 `HttpDns.GetAddrByName(string domain)` 或者 `HttpDns.GetAddrByNameAsync(string domain)` 方法。

如使用同步接口 `HttpDns.GetAddrByName`，直接调用接口即可。

如果使用异步接口 `HttpDns.GetAddrByNameAsync`，还需设置回调函数 `onDnsNotify(string ipString)`，函数名可自定义。

并建议添加如下处理代码：

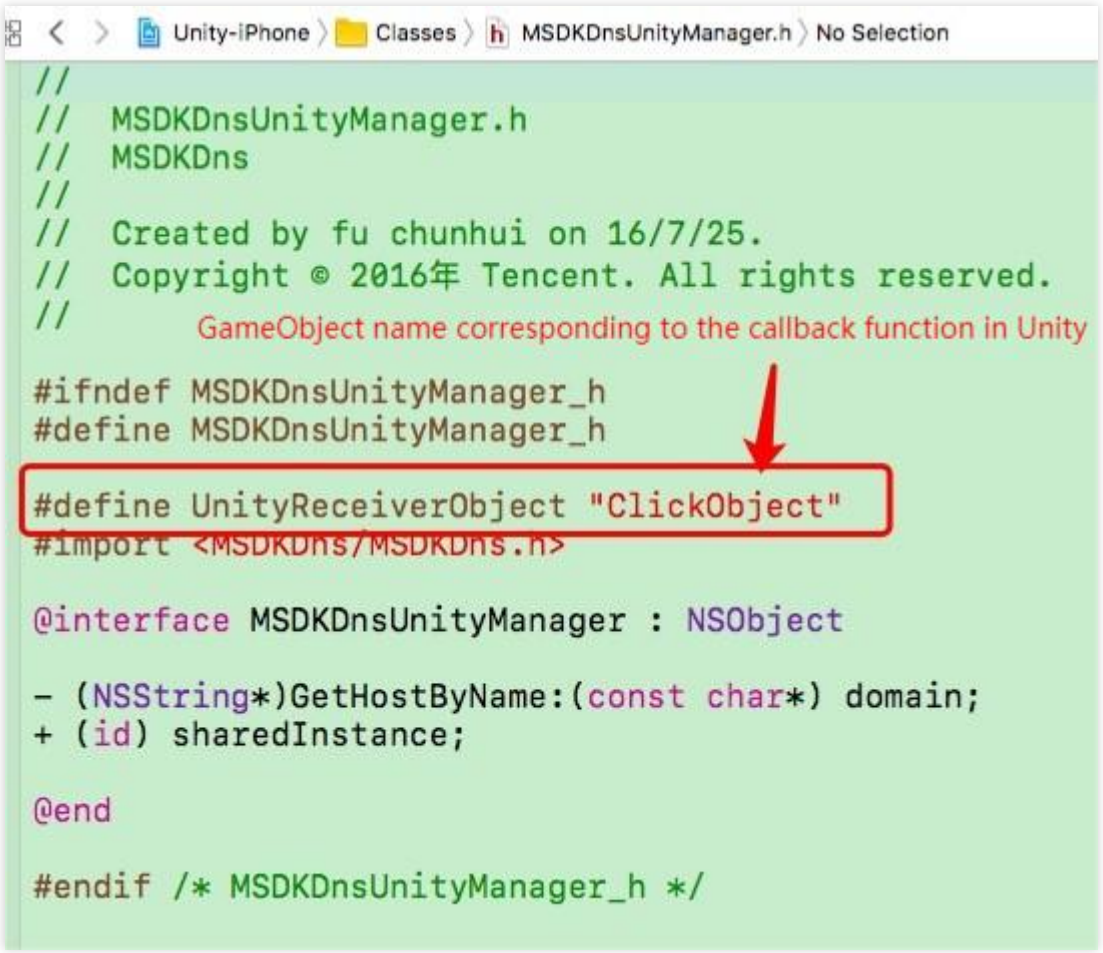


```
string[] sArray=ipString.Split(new char[] {';'});
if (sArray != null && sArray.Length > 1) {
 if (!sArray[1].Equals("0")) {
  //使用建议：当 IPv6 地址存在时，优先使用 IPv6 地址
  //TODO 使用 IPv6 地址进行 URL 连接时，注意格式，需加方框号[ ]进行处理，例如：http://[64:ff
 } else if(!sArray [0].Equals ("0")) {
  //使 IPv4 地址进行连接
 } else {
  //异常情况返回为0,0，建议重试一次
  HttpDns.GetAddrByName(domainStr);
 }
```

```
}
```

3. 将 unity 工程打包为 xcode 工程后，引入所需依赖库。

4. 将 `HTTPDNSUnityDemo` 下的 `MSDKDnsUnityManager.h` 及 `MSDKDnsUnityManager.mm` 文件导入到工程中，注意以下地方需要与 Unity 中对应的 `GameObject` 名称及回调函数名称一致。如下图所示：



```
void WGGetHostByNameAsync(const char* domain){
    [[MSDKDns sharedInstance] WGGetHostByNameAsync:[NSString stringWithUTF8St
        ipsArray) {
        if (ipsArray && ipsArray.count > 1) {
            NSString* result = [NSString stringWithFormat:@"%@;%@",ipsArray[0
            char* resultChar = MakeStringCopy([result UTF8String]);
            UnitySendMessage(UnityReceiverObject, "onDnsNotify", resultChar);
        } else {
            char* resultChar = (char*)"0;0";
            UnitySendMessage(UnityReceiverObject, "onDnsNotify", resultChar);
        }
    }];
}
```

Name of the cor

WebView 实践

最近更新时间：2023-06-12 14:58:44

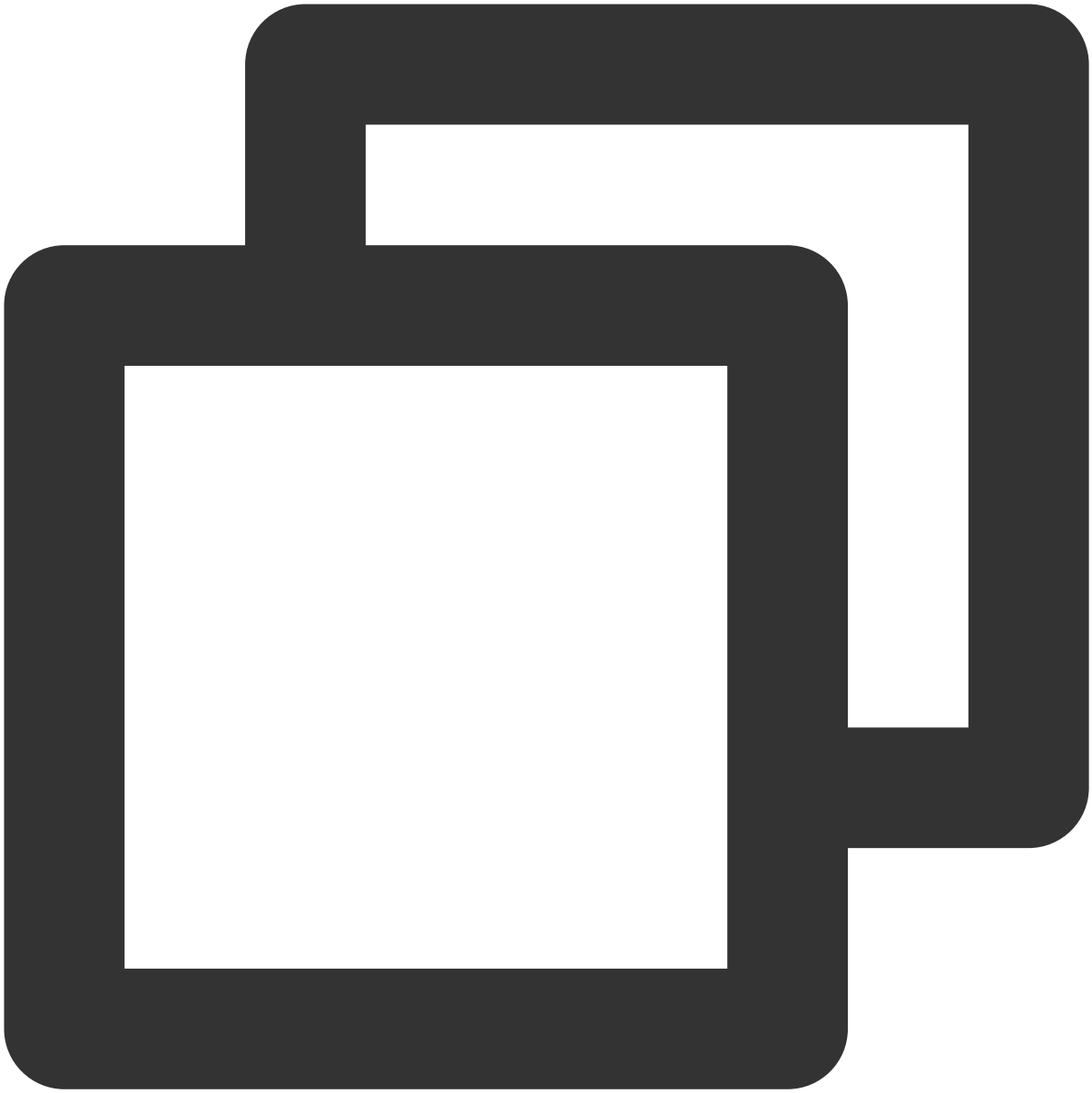
该文档说明 H5 页面内元素 HTTP_DNS 的加载。

原理

拦截网络请求：使用 IOS 原生的 NSURLProtocol，拦截 webview 的网络请求，并根据网络请求 URL 的文件名后缀进行过滤，拿到过滤后的 URL 以后，截取 URL 的域名，然后进行 HTTP_DNS 请求，最后用返回的 IP 地址拼接原文件网络请求的 URL。

实现方法

在 NSURLProtocol 抽象类方法 startLoading 中进行 HTTPDNS 解析，将域名替换成 IP 后进行 URLConnection。



```
/**
 *  让被拦截的请求执行，在此处进行 HTTPDNS 解析，将域名替换成 IP 后进行 URLConnection
 */
- (void)startLoading
{
      NSMutableURLRequest *newRequest;
    NSString *fileExtension = [[self.request URL] absoluteString];

      //根据业务需求，进行 png，jpg，css 等格式的 URL 域名解析
      if ([fileExtension containsString:@".png"] || [fileExtension containsString:@
      {
```

```
            // 修改了请求的头部信息，同步/异步请求
            newRequest = [[H5ContentURLProtocol convertToNewRequest:self.request isSy
        } else {
            newRequest = [self.request mutableCopy];
        }

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

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

Android SDK 文档
Android SDK 发布动态

最近更新时间：2023-09-05 09:53:47

v4.6.0 (2023年08月28日)

http 解析请求支持长链接。
批量解析重构优化。
功能优化。

v4.5.0 (2023年07月06日)

dnsip（HTTPDNS服务IP） SDK内部调度，无需用户配置。
支持ECS IP配置。
包体积优化。
灯塔（beacon）下线。

v4.4.0 (2023年05月23日)

SDK 支持数据上报统计分析，配合控制台**解析监控**使用。原灯塔上报服务（beacon）将下线，建议尽快切换。
新增独立国际版 SDK。
功能优化。

v4.3.0 (2022年09月06日)

新增允许使用过期缓存配置，调整对应解析逻辑。
新增本地存储能力。
功能优化。

v4.2.0 (2022年08月15日)

新增 IP 优选功能。
功能优化。

v4.1.0 (2022年07月12日)

新增缓存自动刷新功能。
解析时延优化。

v4.0.1 (2022年05月10日)

代码优化。

v4.0.0 (2022年03月18日)

增加参数配置选项、自定义解析栈（IPV4，IPV6）。
开放预解析能力、开放关闭 localdns 兜底解析能力等。

v3.9.0 (2022年02月23日)

代码优化。

v3.8.0 (2021年12月29日)

增加日志服务。
容灾逻辑优化。

v3.7.0 (2021年06月25日)

提供 IPv4、IPv6 异步解析接口。
去除 SSID 信息采集。

v3.6.0 (2021年05月25日)

支持批量查询以及多种加密方式（AES、HTTPS、DES）。

v3.5.0 (2021年04月16日)

功能优化。

v3.3.0 (2021年01月05日)

代码优化。

v3.2.5 (2020年10月22日)

修复 Bug。

v3.2.4 (2020年06月22日)

修复 Bug。

Android SDK 接入

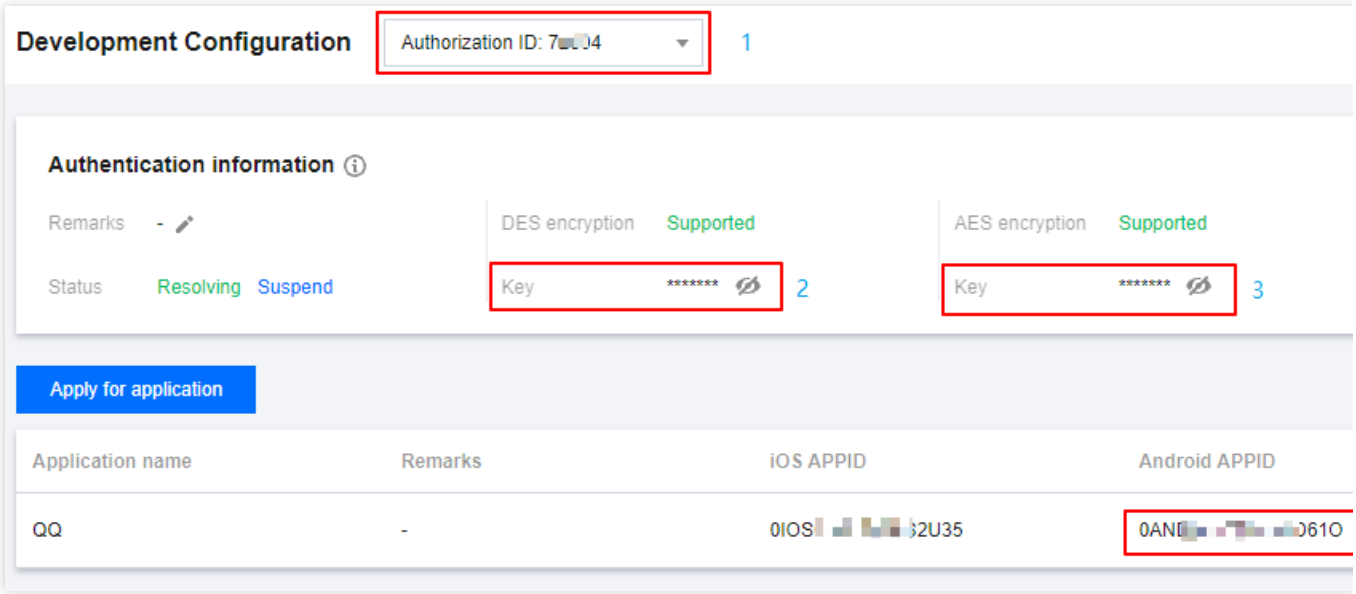
最近更新时间：2023-09-05 09:54:22

概述

移动解析 HTTPDNS 作为移动互联网时代 DNS 优化的一个通用解决方案，主要解决了以下几类问题：
LocalDNS 劫持/故障
LocalDNS 调度不准确
移动解析 HTTPDNS 的 Android SDK，主要提供了基于移动解析 HTTPDNS 服务的域名解析和缓存管理能力：
SDK 在进行域名解析时，优先通过移动解析 HTTPDNS 服务得到域名解析结果，极端情况下如果移动解析 HTTPDNS 服务不可用，则使用 LocalDNS 解析结果。
移动解析 HTTPDNS 服务返回的域名解析结果会携带相关的 TTL 信息，SDK 会使用该信息进行移动解析 HTTPDNS 解析结果的缓存管理。

前期准备

1. 开通移动解析 HTTPDNS 服务，详情请参见 开通移动解析 HTTPDNS。
2. 服务开通后，您需在移动解析 HTTPDNS 控制台添加解析域名后才可正常使用，详情请参见 添加域名。
3. 在移动解析 HTTPDNS 控制台申请接入 SDK，详情请参见 开通 SDK。
4. SDK 开通后，移动解析 HTTPDNS 将为您分配授权 ID、AES 和 DES 加密密钥及 HTTPS Token 等配置信息。您可前往 开发配置 页面查看，如下图所示：



使用 Android SDK 需求获取的配置如下：

授权 ID：使用移动解析 HTTPDNS 服务中，开发配置的唯一标识。SDK中 `dnsId` 参数，用于域名解析鉴权。
DES加密密钥：SDK中 `dnsKey` 参数，加密方式为 DES 时传入此项。
AES 加密密钥：SDK中 `dnsKey` 参数，加密方式为 AES 时传入此项。
HTTPS 加密 Token：SDK中 `token` 参数，加密方式为 HTTPS 时传入此项。
Android APPID： Android 端 SDK 的 `appkey` 鉴权信息。

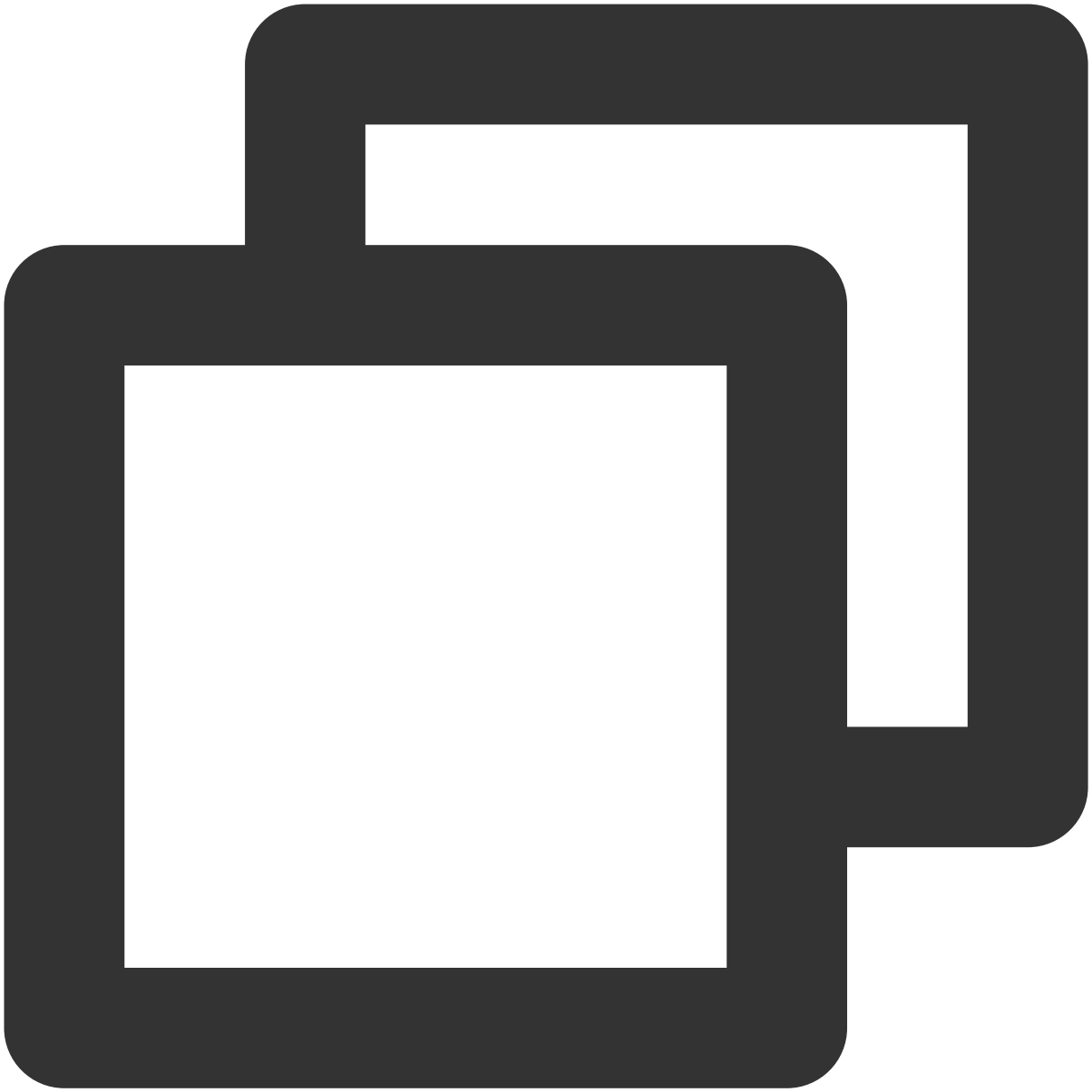
SDK 接入

接入 HTTPDNS SDK

直接引入aar包

1. 获取 移动解析 Android SDK。
2. aar 包引入，将 `HttpDNSLibs\\HTTPDNS_ANDROID_xxxx.aar` 拷贝至应用 libs 相应位置。
3. 在 App module的 `build.gradle` 文件中，添加如下配置：

说明：
V4.3.0版本开始增加了本地数据存储，需增加room依赖引入。请参见 官方指引。

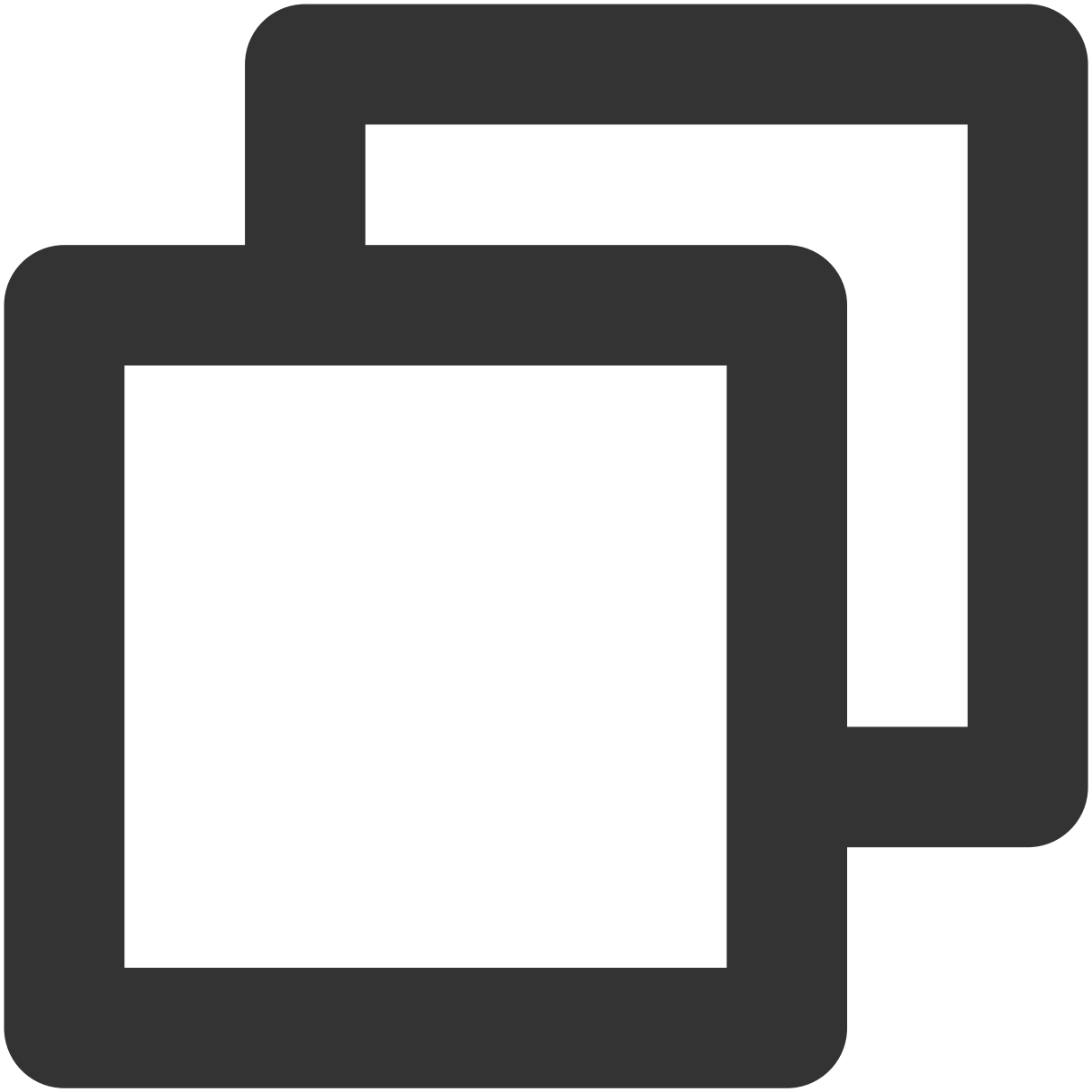


```
android {

 // ...

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

```
dependencies {

 // ...

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

 // V4.3.0版本开始增加了本地数据存储，需增加room依赖引入。
 implementation "androidx.room:room-rxjava2:2.2.0"
}
```

maven资源库下载

1. 直接导入pom文件依赖



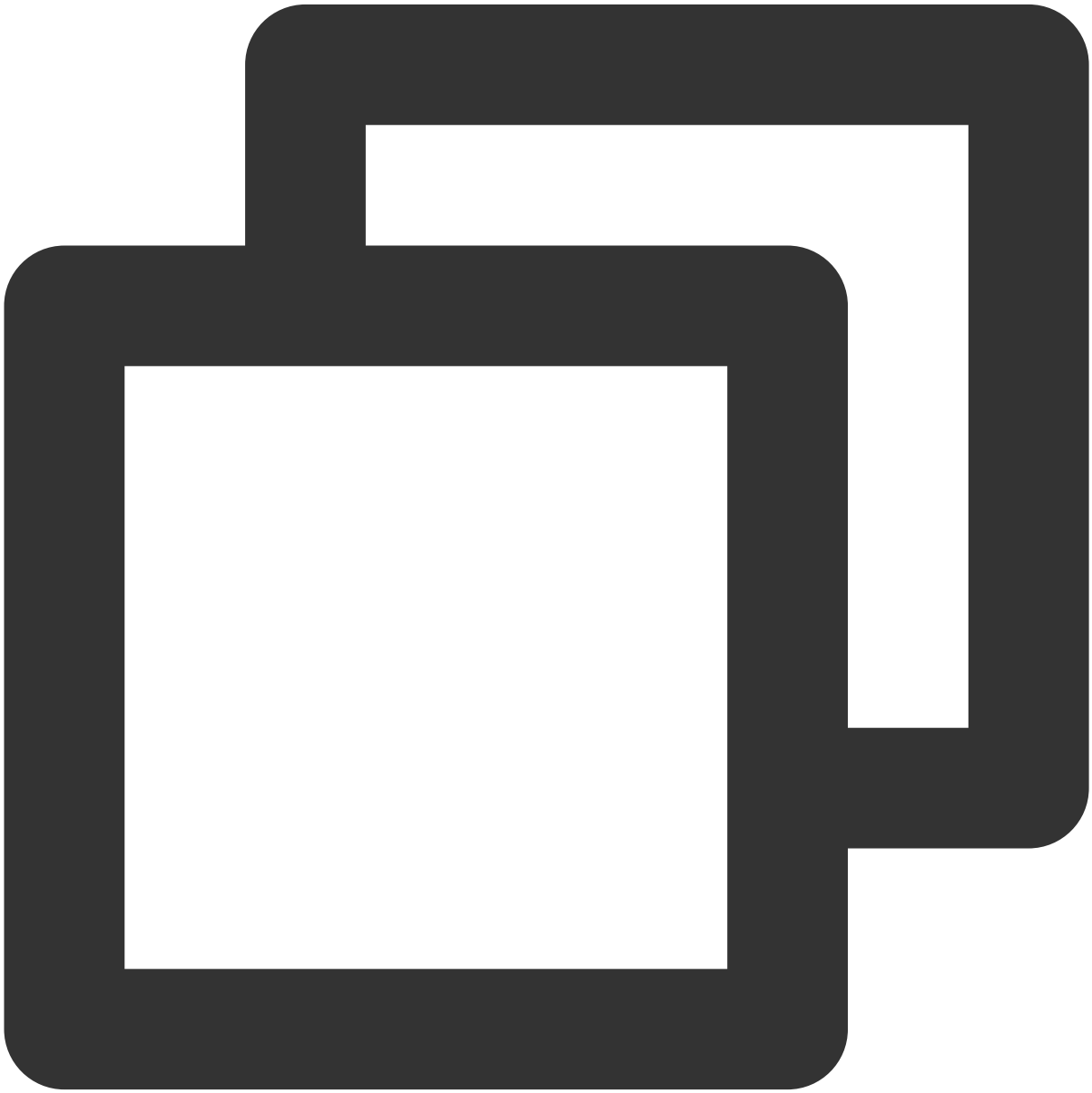
```
<dependency>
  <groupId>io.github.dnspod</groupId>
  <artifactId>httpdns-sdk</artifactId>
  <version>4.4.0</version>
  <type>aar</type>
</dependency>

<!-- 国际站sdk -->
<dependency>
  <groupId>io.github.dnspod</groupId>
  <artifactId>httpdns-sdk</artifactId>
```

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

注意

HTTPDNS SDK V4.4.0支持国际站独立SDK，使用时初始化配置信息需在HTTPDNS国际站控制台中获取。详情请参见国际站接入文档。

权限配置



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

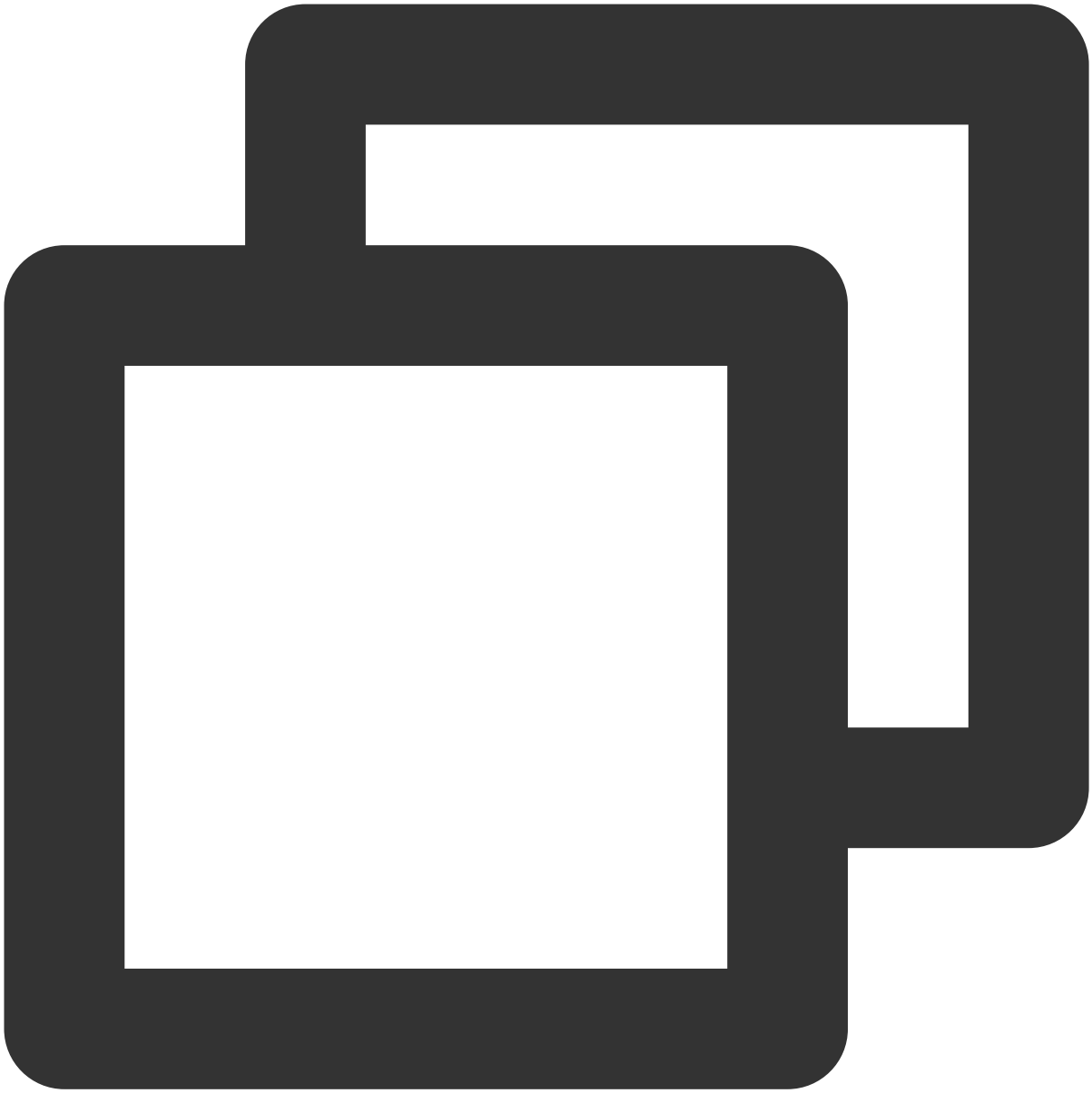
```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />

<!-- 用于获取手机imei码进行数据上报，非必须 -->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

网络安全配置兼容

App targetSdkVersion ≥ 28(Android 9.0)的情况下，系统默认不允许 HTTP 网络请求，详情请参见 Opt out of cleartext traffic。

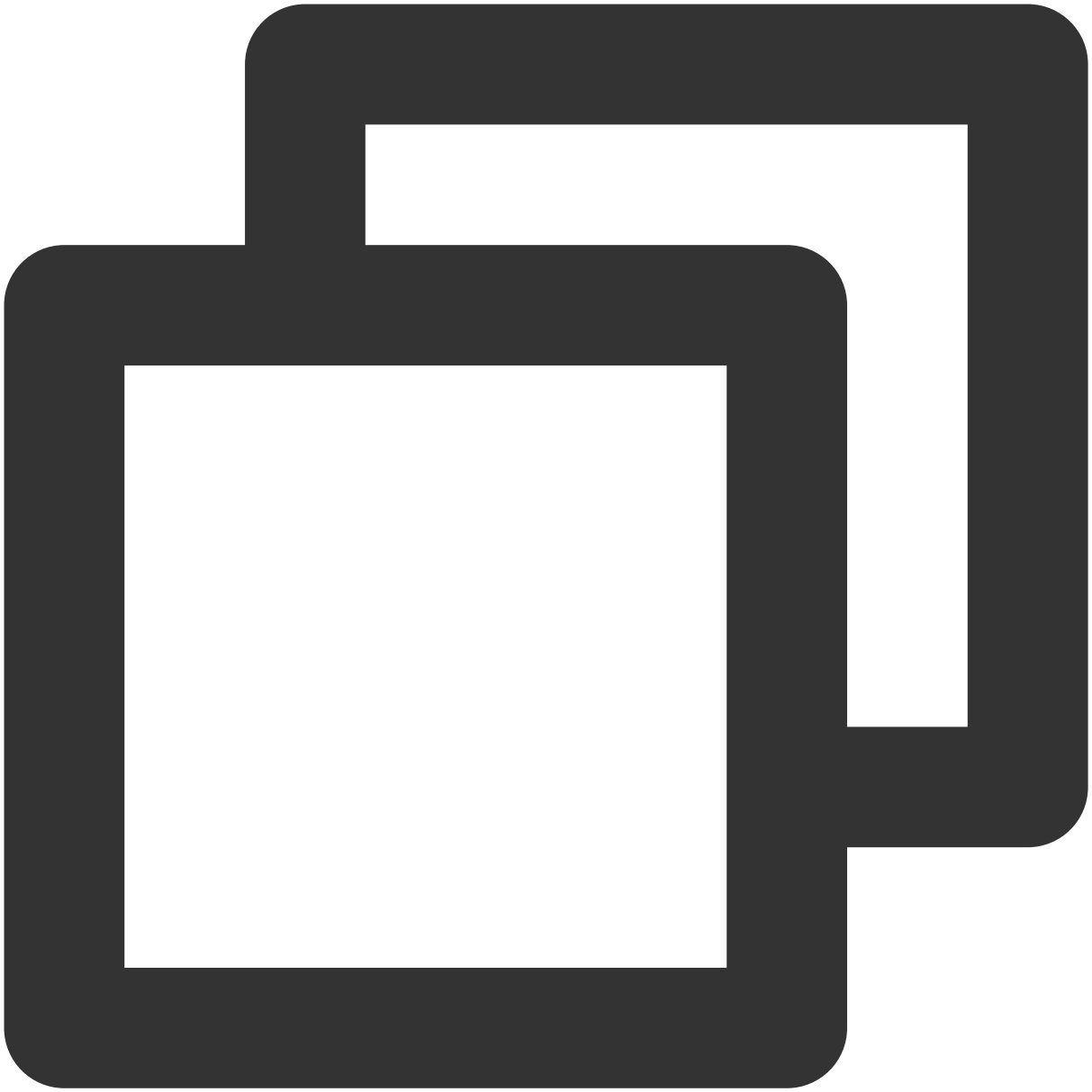
这种情况下，业务侧需要将 HTTPDNS 请求使用的 IP 配置到域名白名单中。配置如下：

AndroidManifest 文件中配置。



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

XML 目录下添加 network_security_config.xml 配置文件。



```
<?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 初始化

初始化配置服务（4.0.0版本开始支持）

注意

HTTPDNS SDK 提供多重解析优化策略，建议根据实际情况选配，也可以组合使用，可使得解析成功率达到最优效果。

可以通过配置 `setUseExpiredIpEnable(true)` 和 `setCachedIpEnable(true)` 来实现乐观 DNS 缓存。

该功能旨在提升缓存命中率和首屏加载速度。持久化缓存会将上一次解析结果保持在本地，在 App 启动时，会优先读取到本地缓存解析结果。

存在使用缓存 IP 时为过期 IP（TTL 过期），该功能启用了允许使用过期 IP，乐观的推定 TTL 过期，大多数情况下该 IP 仍能正常使用。优先返回缓存的过期结果，同时异步发起解析服务，更新缓存。

乐观 DNS 缓存在首次解析域名（无缓存）时，无法命中缓存，返回0;0，同时也会异步发起解析服务，更新缓存。

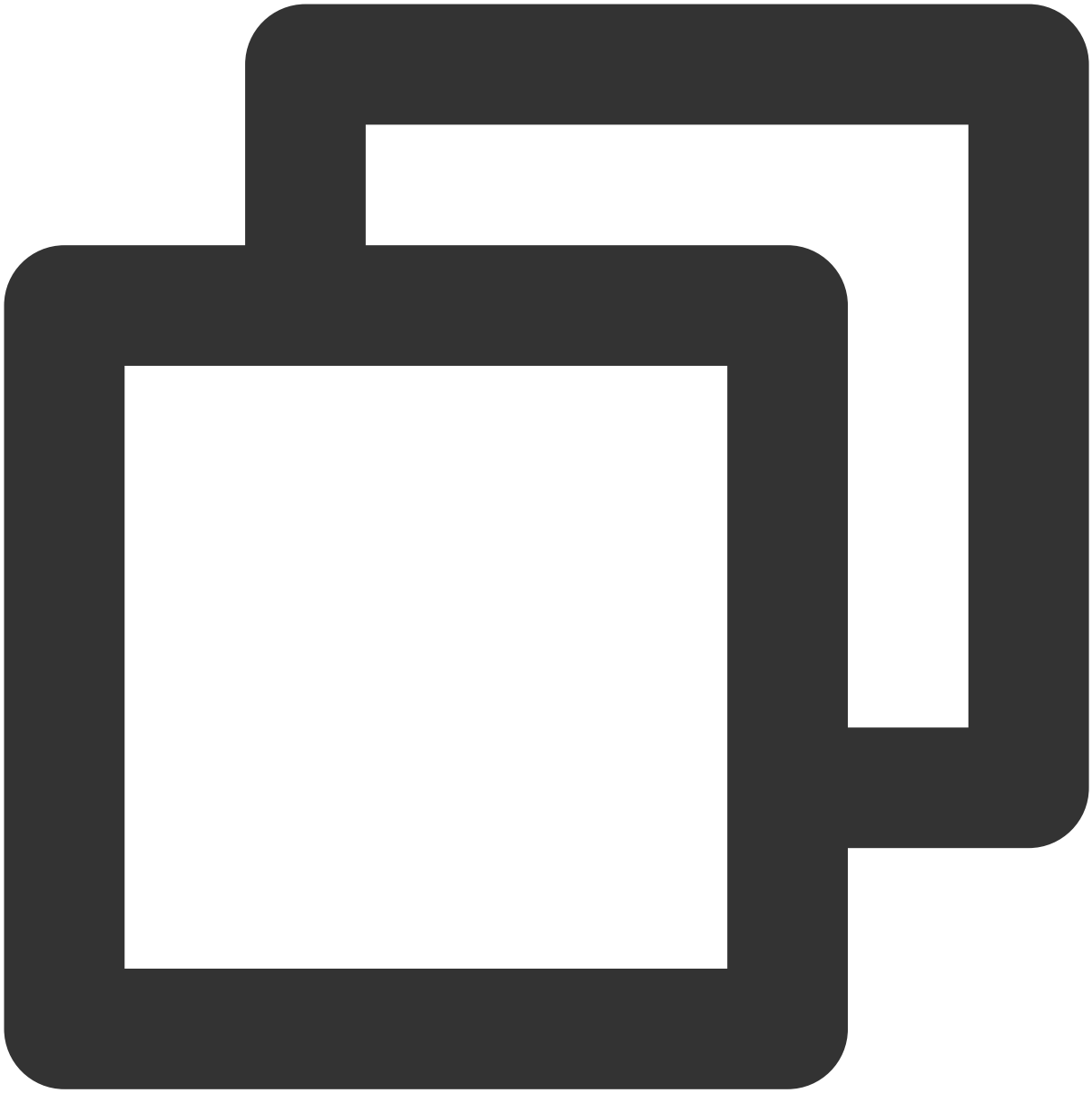
在启用该功能后需自行 LocalDNS 兜底。核心域名建议配置预解析服务 `preLookupDomains(String... domainList)` 。

如果业务服务器 IP 变化比较频繁，务必启用缓存自动刷新 `persistentCacheDomains(String... domainList)` 、预解析能力 `preLookupDomains(String... domainList)` ，以确保解析结果的准确性。

在获取服务实例之前，可通过初始化配置，设置服务的一些属性在 SDK 初始化时进行配置项传入。

说明：

【V4.4.0 废弃】 sdk日志上报能力由控制台控制，请查看具体指引。



```
DnsConfig dnsConfigBuilder = DnsConfig.Builder()
    //（必填）dns 解析 id，即授权 id，腾讯云官网（https://console.tencentcloud.com/httpdr
    .dnsId("xxx")
    //（必填）dns 解析 key，即授权 id 对应的密钥，在开发配置（https://console.tencentcloud
    .dnsKey("xxx")
    //  【V4.5.0版本废弃】由sdk内部调度
    //（必填）Channel为desHttp()或aesHttp()时使用 119.29.29.98（默认填写这个就行），channe
    .dnsIp("xxx")
    //（可选）channel配置：基于 HTTP 请求的 DES 加密形式，默认为 desHttp()，另有 aesHttp()、
    .desHttp()
    //（可选，选择 https channel 时进行设置）腾讯云官网（https://console.tencentcloud.com
```

```
        .token("xxx")
        //（可选）日志粒度，如开启Debug打印则传入"Log.VERBOSE"
        .logLevel(Log.VERBOSE)
        //（可选）预解析域名，填写形式："baidu.com", "qq.com"，建议不要设置太多预解析域名，当前限制
        .preLookupDomains("baidu.com", "qq.com")
        //（可选）解析缓存自动刷新，以域名形式进行配置，填写形式："baidu.com", "qq.com"。配置的域名
        .persistentCacheDomains("baidu.com", "qq.com")
        // （可选）IP 优选，以 IpRankItem(hostname, port) 组成的 List 配置，port（可选）默认值
        .ipRankItems(ipRankItemList)
        //（可选）手动指定网络栈支持情况，仅进行 IPv4 解析传 1，仅进行 IPv6 解析传 2，进行 IPv4、I
        .setCustomNetStack(3)
        //（可选）设置是否允许使用过期缓存，默认false，解析时先取未过期的缓存结果，不满足则等待解析请求
        // 设置为true时，会直接返回缓存的解析结果，没有缓存则返回0;0，用户可使用localdns（InetAddr
        .setUseExpiredIpEnable(true)
        //（可选）设置是否启用本地缓存（Room），默认false
        .setCachedIpEnable(true)
        //（可选）设置域名解析请求超时时间，默认为2000ms
        .timeoutMills(2000)
        // （可选）DNS 请求的 ECS（EDNS-Client-Subnet）值，默认情况下 HTTPDNS 服务器会查询客户端
        .routeIp("XXX")
        //（可选）【V4.4.0 废弃】 sdk日志上报能力由控制台控制
        .enableReport(true)
        // 以build()结束
        .build();

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

旧版本初始化（建议尽快升级）

HTTP 协议服务地址为 `119.29.29.98`，HTTPS 协议服务地址为 `119.29.29.99`（仅当采用自选加密方式并且 `channel` 为 `Https` 时使用 `99` 的 IP）。
新版本 API 更新为使用 `119.29.29.99/98` 接入，同时原移动解析 HTTPDNS 服务地址 `119.29.29.29` 仅供开发调试使用，无 SLA 保障，不建议用于正式业务，请您尽快将正式业务迁移至 `119.29.29.99/98`。
具体以 API 说明 提供的 IP 为准。
使用 SDK 方式接入 HTTPDNS，若 HTTPDNS 未查询到解析结果，则通过 LocalDNS 进行域名解析，返回 LocalDNS 的解析结果。

SDK 接入业务方式

将 HTTPDNS SDK 的域名解析能力接入到业务的 HTTP（HTTPS）网络访问流程中，总的来说可以分为以下两种方式：

方式1：替换 URL

替换 URL 中的 Host 部分得到新的 URL，并使用新的 URL 进行网络访问。

这种实现方案下，URL 丢掉了域名的信息，对于需要使用到域名信息的网络请求，需进行较多的兼容性工作。

方式2：替换 DNS

将 HTTPDNS 的域名解析能力注入到网络访问流程中，替换掉原本网络访问流程中的 LocalDNS 来实现。

这种实现方案下，不需要逐个对请求的 URL 进行修改，同时由于没有修改 URL，无需进行额外的兼容性工作，但需要业务侧使用的网络库支持 DNS 实现替换。

DNS 替换也可以通过 Hook 系统域名解析函数的方式来实现，但 HTTPDNS SDK 内部已经使用系统的域名解析函数，Hook 系统域名解析函数可能会造成递归调用直到栈溢出。

不同网络库具体的接入方式，可以参见对应的接入文档（当前目录下）及参考使用 Sample（HttpDnsSample 目录）。

替换 URL 接入方式兼容

如前文所述，对于需要使用到域名信息的网络请求（一般是多个域名映射到同一个 IP 的情况），需要进行额外兼容。以下从协议层面阐述具体的兼容方式，具体的实现方式需要视网络库的实现而定。

HTTP 兼容

对于 HTTP 请求，需要通过指定报文头中的 Host 字段来告知服务器域名信息。Host 字段详细介绍参见 Host。

HTTPS 兼容

HTTPS 是基于 TLS 协议之上的 HTTP 协议的统称。对于 HTTPS 请求，同样需要设置 Host 字段。

在 HTTPS 请求中，需要先进行 TLS 的握手。TLS 握手过程中，服务器会将自己的数字证书发给我们用于身份认证，因此，在 TLS 握手过程中，也需要告知服务器相关的域名信息。在 TLS 协议中，通过 SNI 扩展来指明域名信息。SNI 扩展的详细介绍参见 Server Name Indication。

本地使用 HTTP 代理

说明

本地使用 HTTP 代理情况下，建议**不要使用 HTTPDNS** 进行域名解析。

以下区分两种接入方式并进行分析：

替换 URL 接入

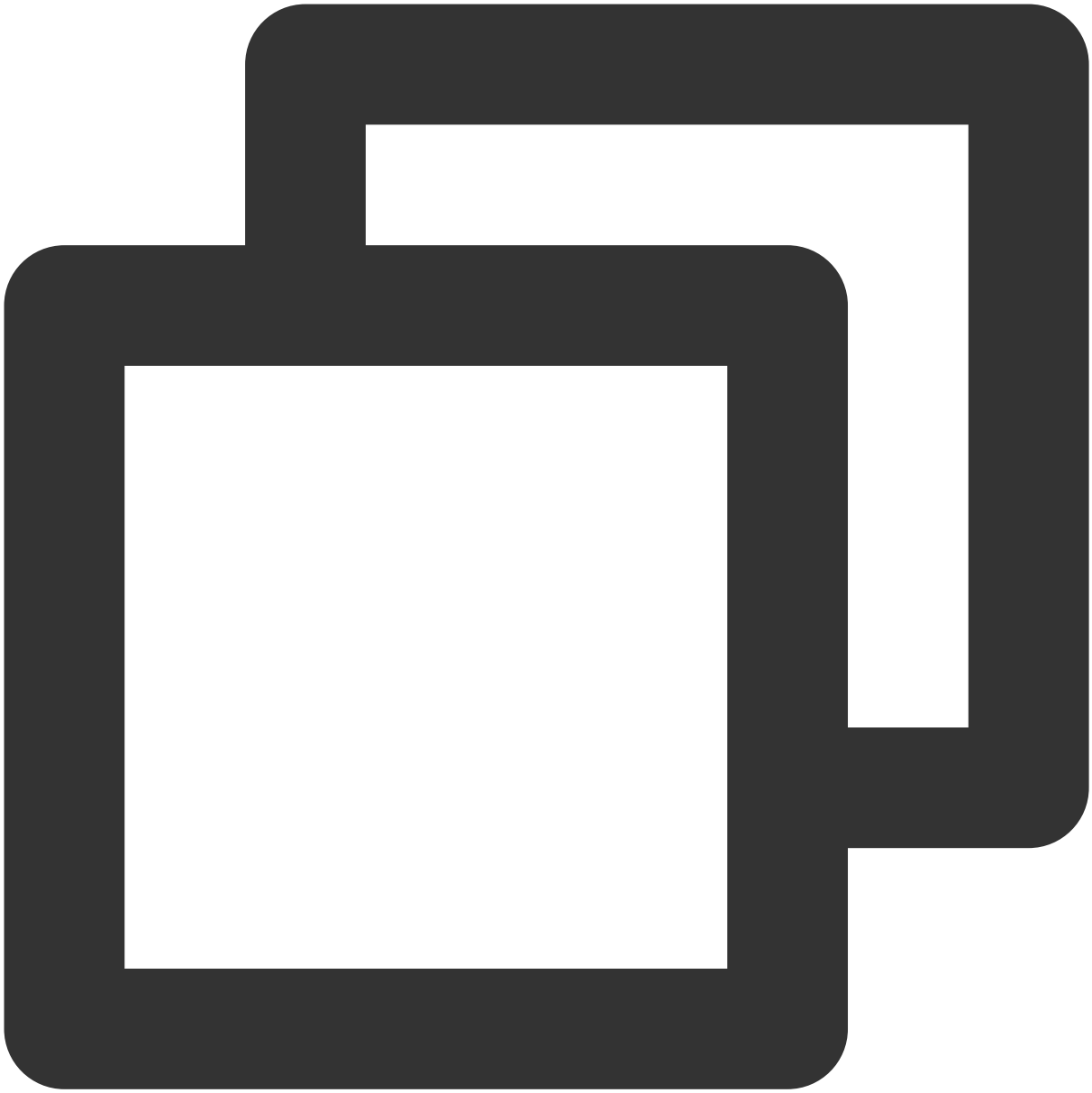
根据 HTTP/1.1 协议规定，在使用 HTTP 代理情况下，客户端侧将在请求行中带上完整的服务器地址信息。详细介绍可以参见 origin-form。

这种情况下（本地使用了 HTTP 代理，业务侧使用替换 URL 方式接入了 HTTPDNS SDK，且已经正确设置了 Host 字段），HTTP 代理接收到的 HTTP 请求中会包含服务器的 IP 信息（请求行中）以及域名信息（Host 字段中），但具体 HTTP 代理会如何向真正的目标服务器发起 HTTP 请求，则取决于 HTTP 代理的实现，可能会直接丢掉我们设置的 Host 字段使得网络请求失败。

替换 DNS 实现

以 OkHttp 网络库为例，在本地启用 HTTP 代理情况下，OkHttp 网络库不会对一个 HTTP 请求 URL 中的 Host 字段进行域名解析，而只会对设置的 HTTP 代理的 Host 进行域名解析。在这种情况下，启用 HTTPDNS 没有意义。

您可通过以下代码，**判断本地是否使用 HTTP 代理**：



```
val host = System.getProperty("http.proxyHost")
val port = System.getProperty("http.proxyPort")
if (null != host && null != port) {
    // 本地使用了 HTTP 代理
}
```

接入验证

日志验证

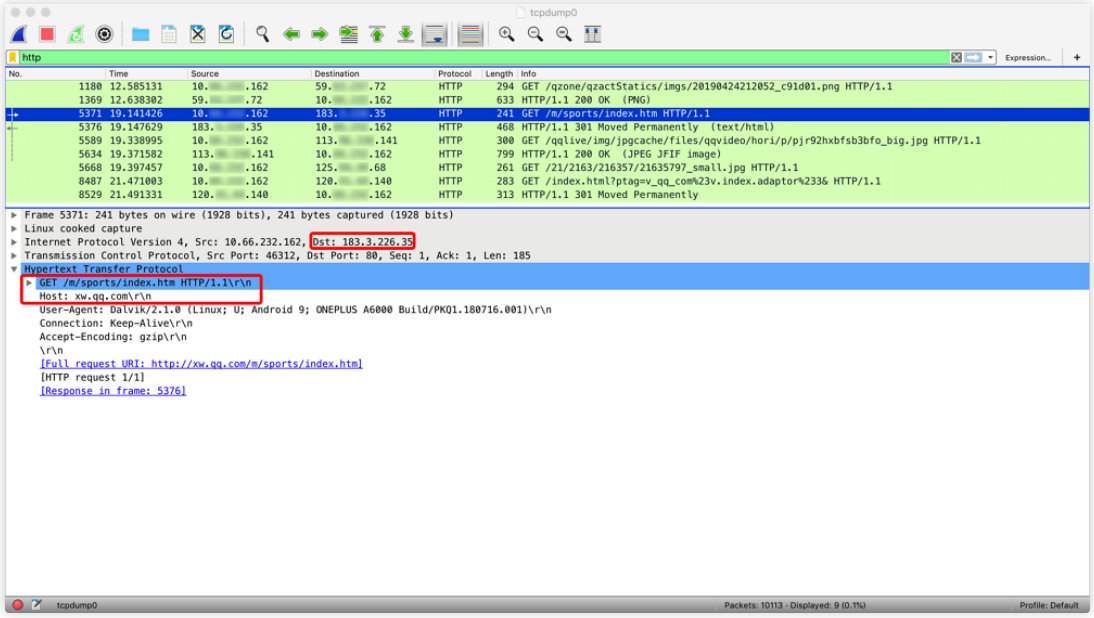
当 init 接口中 debug 参数传入 true，过滤 TAG 为 “HTTPDNS” 的日志，并查看到 LocalDns（日志上为 ldns_ip）和 HTTPDNS（日志上为 hdns_ip）相关日志时，可以确认接入无误。
key 为 ldns_ip 的是 LocalDNS 的解析结果。
key 为 hdns_ip 的是 HTTPDNS A 记录的解析结果。
key 为 hdns_4a_ips 的是 HTTPDNS AAAA 记录的解析结果。
key 为 a_ips 的是域名解析接口返回的 IPv4 集合。
key 为 4a_ips 的是域名解析接口返回的 IPv6 集合。

模拟 LocalDNS 劫持

模拟 LocalDNS 劫持情况下，若 App 能够正常工作，可以证明 HTTPDNS 已经成功接入。
注意
由于 LocalDNS 存在缓存机制，模拟 LocalDNS 进行接入验证时，请尽量保证 LocalDNS 的缓存已经被清理。您可以通过重启机器，切换网络等方式，尽量清除 LocalDNS 的解析缓存。验证时，请注意对照启用 LocalDNS 和启用 HTTPDNS 的效果。
修改机器 Hosts 文件。
LocalDNS 优先通过读取机器 Hosts 文件方式获取解析结果。
通过修改 Hosts 文件，将对应域名指向错误的 IP，可以模拟 LocalDNS 劫持。
Root 机器可以直接修改机器 Hosts 文件。
修改 DNS 服务器配置。
通过修改 DNS 服务器配置，将 DNS 服务器指向一个不可用的 IP（如局域网内的另一个 IP），可以模拟 LocalDNS 劫持。
机器连接 Wi-Fi 情况下，在当前连接的 Wi-Fi 的高级设置选项中修改 IP 设置为静态设置，可以修改 DNS 服务器设置（不同机器具体的操作路径可能略有不同）。
借助修改 DNS 的 App 来修改 DNS 服务器配置（通常是通过 VPN 篡改 DNS 包的方式来修改 DNS 服务器配置）。

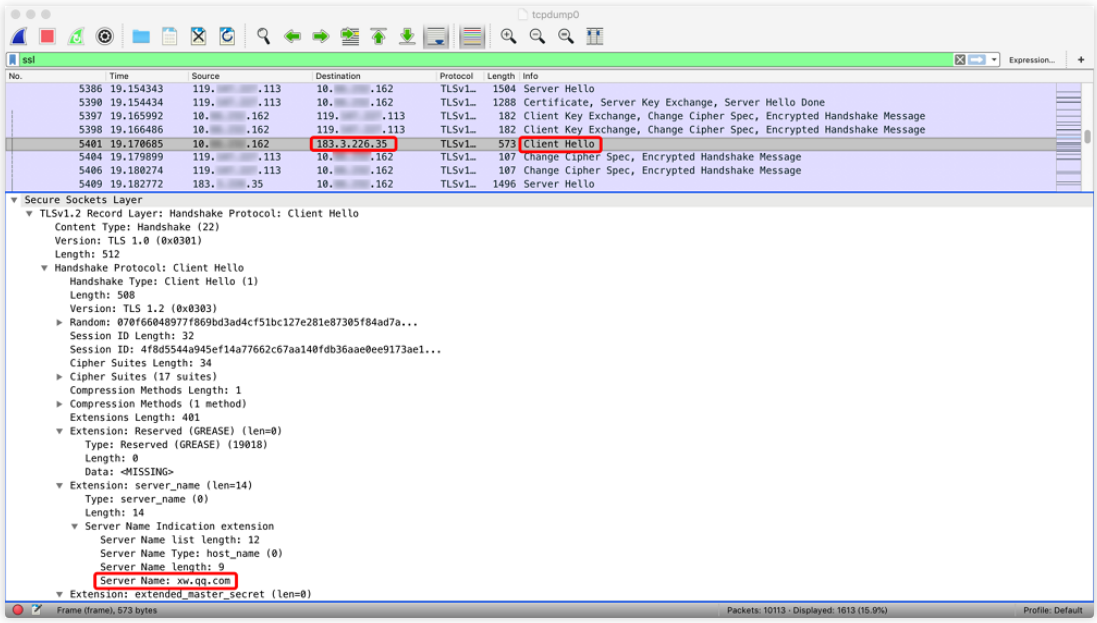
抓包验证

以下以接入 HTTP 网络访问为例进行说明：
注意
常用的移动端 HTTP/HTTPS 抓包工具（例如 Charles/Fiddler），是通过 HTTP 代理方式进行抓包，不适用于抓包验证 HTTPDNS 服务是否生效，相关说明请参见 本地使用 HTTP 代理https://cloud.tencent.com/document/product/379/78134#local。
使用 **tcpdump** 进行抓包。
Root 机器可以通过 tcpdump 命令抓包。
非 Root 机器上，系统可能内置有相关的调试工具，可以获取抓包结果（不同机器具体的启用方式不同）。
通过 **WireShark** 观察抓包结果。
对于 HTTP 请求，我们可以观察到明文信息，通过对照日志和具体的抓包记录，可以确认最终发起请求时使用的 IP 是否和 SDK 返回的一致。如下图所示：



从抓包上看，`xw.qq.com` 的请求最终发往了 IP 为 `183.3.226.35` 的服务器。

对于 HTTPS 请求，TLS 的握手包实际上是明文包，在设置了 SNI 扩展（请参见 HTTPS 兼容）情况下，通过对照日志和具体的抓包记录，可以确认最终发起请求时使用的 IP 是否和 SDK 返回的一致。如下图所示：

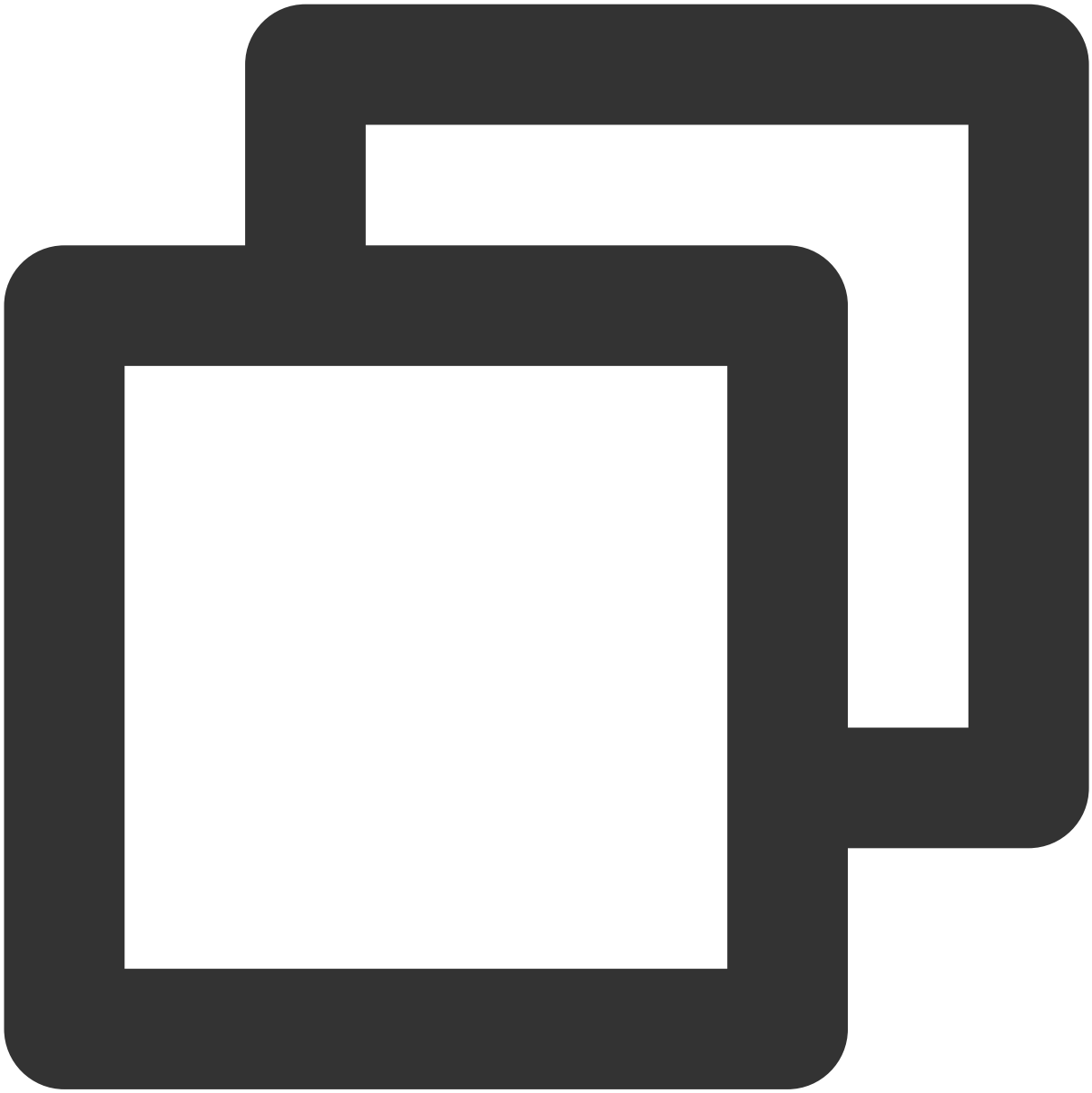


从抓包上看，`xw.qq.com` 的请求最终发往了 IP 为 `183.3.226.35` 的服务器。

注意事项

getAddrByName 是耗时同步接口，应当在子线程调用。

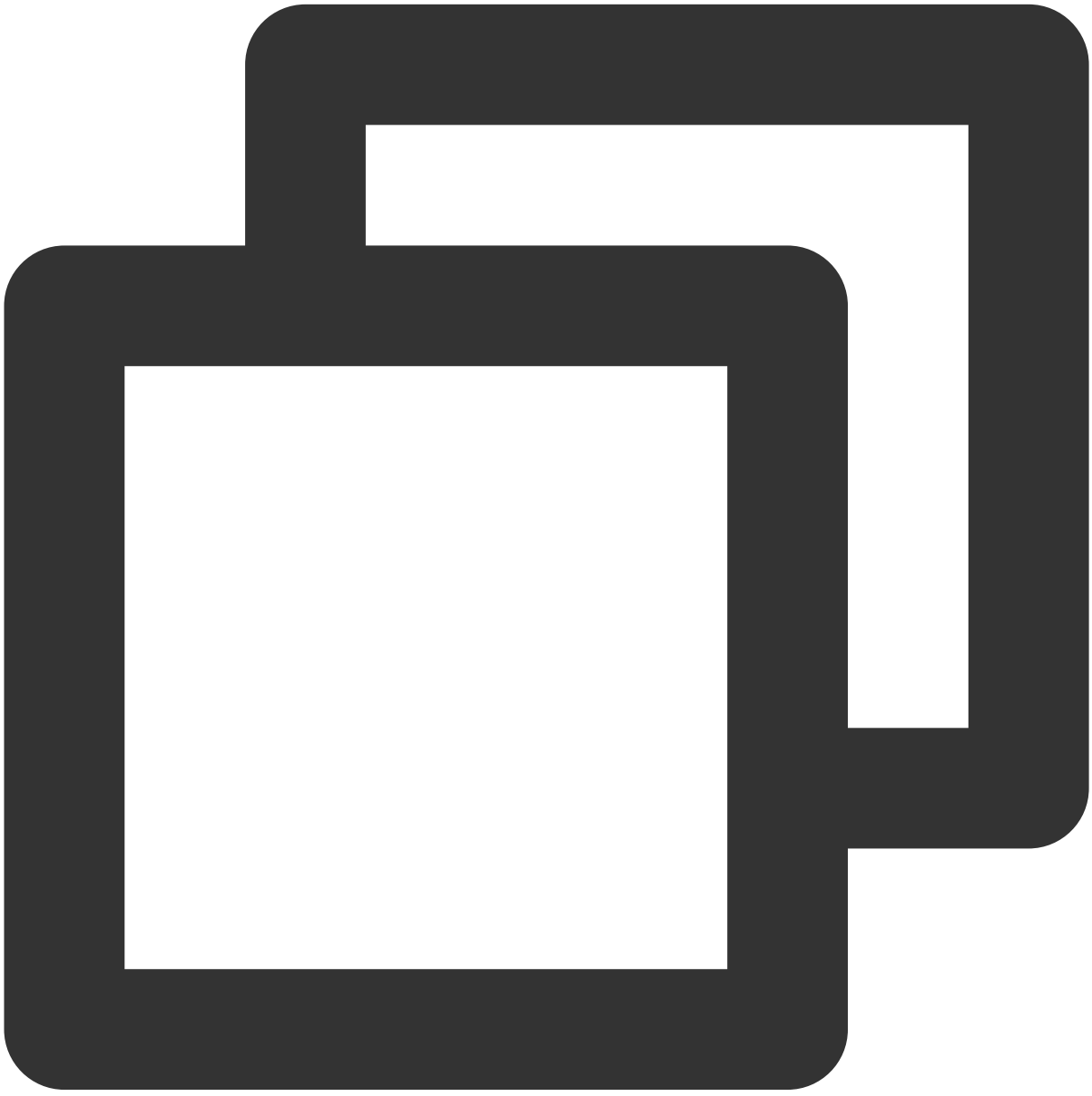
如果客户端的业务与 HOST 绑定，例如客户端的业务绑定了 HOST 的 HTTP 服务或者是 CDN 的服务，那么您将 URL 中的域名替换成 HTTPDNS 返回的 IP 之后，还需要指定下 HTTP 头的 HOST 字段。
以 URLConnection 为例：



```
URL oldUrl = new URL(url);
URLConnection connection = oldUrl.openConnection();
// 获取HTTPDNS域名解析结果
String ips = MSDKDnsResolver.getInstance().getAddrByName(oldUrl.getHost());
String[] ipArr = ips.split(";");
if (2 == ipArr.length && !"0".equals(ipArr[0])) { // 通过 HTTPDNS 获取 IP 成功，进行 U
 String ip = ipArr[0];
```

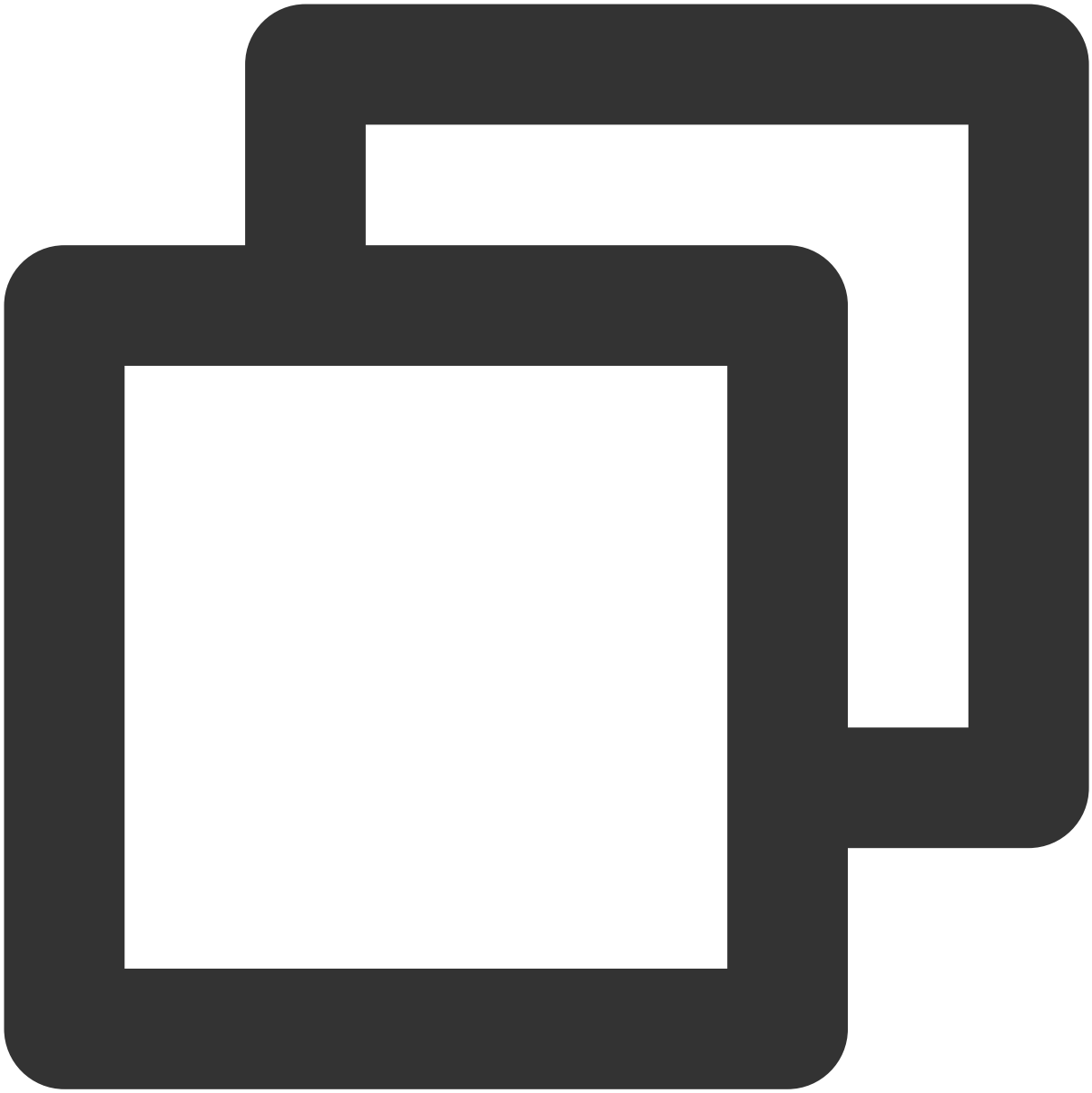
```
  String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
  connection = (HttpURLConnection) new URL(newUrl).openConnection(); // 设置 HTTP 请求
  connection.setRequestProperty("Host", oldUrl.getHost());
}
```

以 curl 为例，假设您想要访问 `www.qq.com`，通过 HTTPDNS 解析出来的 IP 为 `192.168.0.111`，则访问方式如下：



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

检测本地是否使用了 HTTP 代理。如果使用了 HTTP 代理，建议**不要使用 HTTPDNS** 做域名解析。示例如下：

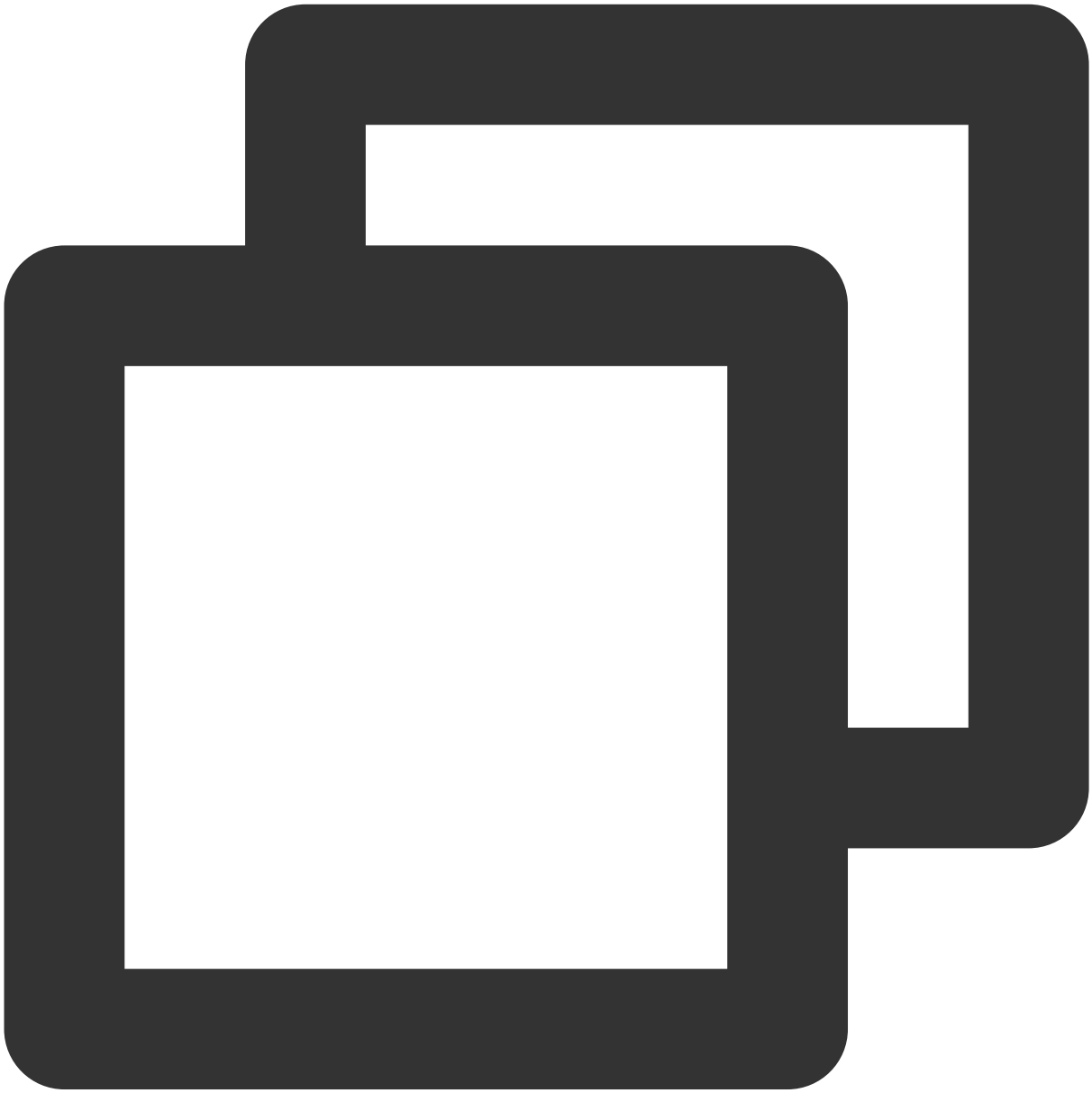


```
String host = System.getProperty("http.proxyHost");
String port= System.getProperty("http.proxyPort");
if (null != host && null != port) {
  // 使用了本地代理模式
}
```

Android SDK API 接口

最近更新时间：2023-06-06 10:57:54

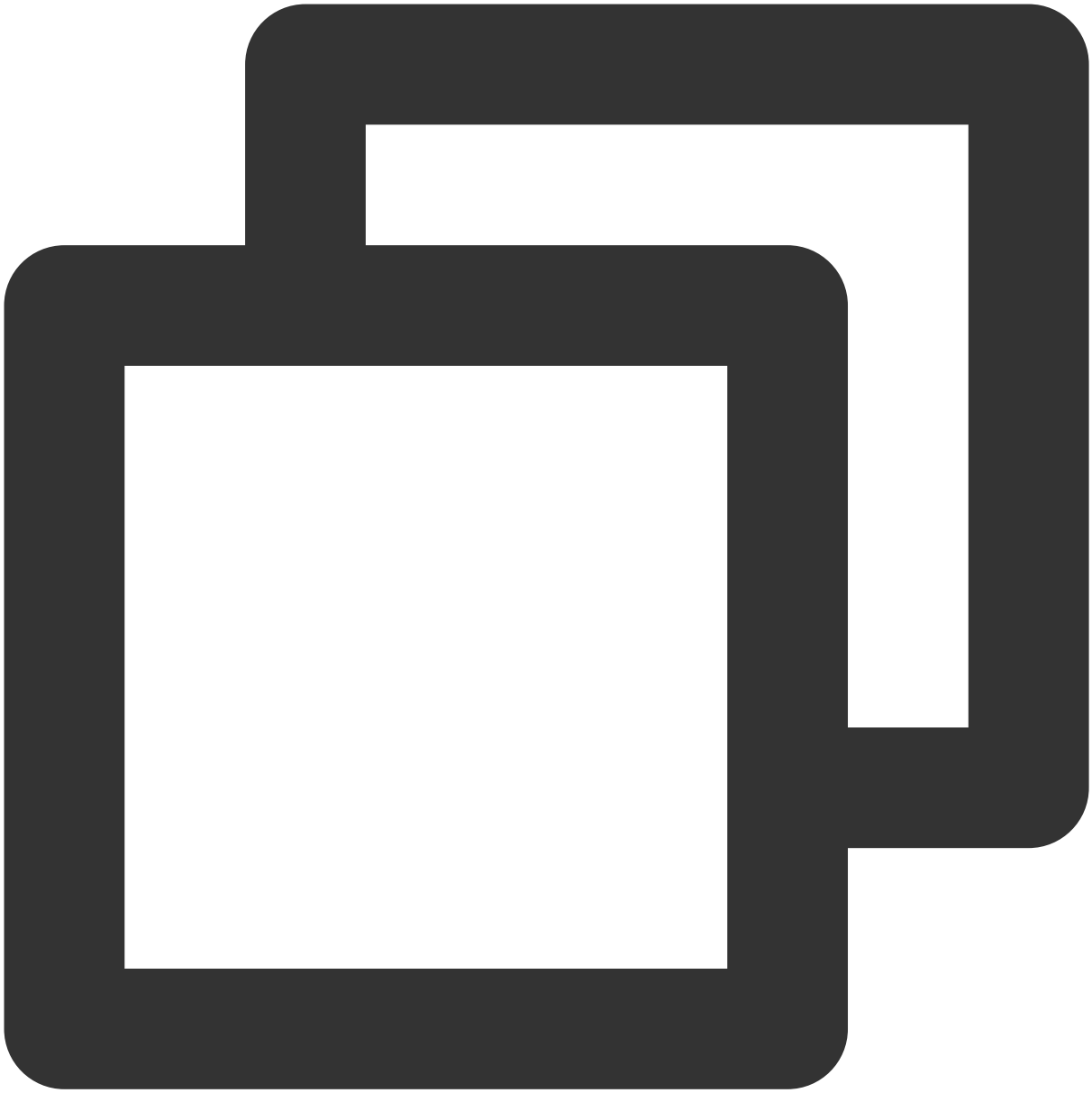
同步解析接口



```
/**
 * HTTPDNS 同步解析接口
 * 首先查询缓存，若存在则返回结果，若不存在则进行同步域名解析请求
 * 解析完成返回最新解析结果
```

```
 * 返回值字符串以“;”分隔，“;”前为解析得到的 IPv4 地址（解析失败填“0”），“;”后为解析得到的 IPv6
 * 返回示例：121.14.77.221;2402:4e00:1020:1404:0:9227:71a3:83d2
 * @param domain 域名（如www.qq.com）
 * @return 域名对应的解析 IP 结果集合
 */
String ips = MSDKDnsResolver.getInstance().getAddrByName(domain);

/**
 * HTTPDNS 同步解析接口（批量查询）
 * 首先查询缓存，若存在则返回结果，若不存在则进行同步域名解析请求
 * 解析完成返回最新解析结果
 * 返回值 ipSet 即解析得到的 IP 集合
 * ipSet.v4Ips 为解析得到 IPv4 集合，可能为 null
 * ipSet.v6Ips 为解析得到 IPv6 集合，可能为 null
 * 单独域名返回结果示例：IpSet{v4Ips=[121.14.77.201, 121.14.77.221], v6Ips=[2402:4e00:1
 * 多域名返回结果示例：IpSet{v4Ips=[www.baidu.com:14.215.177.39, www.baidu.com:14.215.1
 * @param domain 支持多域名，域名以“,”分割，例如：qq.com,baidu.com
 * @return 域名对应的解析 IP 结果集合
 */
Ipset ips = MSDKDnsResolver.getInstance().getAddrsByName(domain);
```

异步解析接口



```
//  异步回调，注意所有异步请求需配合异步回调使用
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 异步解析接口（需配合异步回调使用）
 * 首先查询缓存，若存在则返回结果，若不存在则进行异步域名解析请求
 * 解析完成会在异步回调返回最新解析结果
 * @param domain 域名（如www.qq.com）
 */
MSDKDnsResolver.getInstance()
    .getAddrByNameAsync(hostname, String.valueOf(System.currentTimeMillis()))

/**
 * HTTPDNS 异步解析接口（批量查询，需配合异步回调使用）
 * 首先查询缓存，若存在则返回结果，若不存在则进行同步域名解析请求
 * 解析完成会在异步回调返回最新解析结果
 * @param domain 支持多域名，域名以“,”分割，例如：qq.com,baidu.com
 */
MSDKDnsResolver.getInstance()
    .getAddrsByNameAsync(hostname, String.valueOf(System.currentTimeMillis()))
```

如何提升IPv6使用率

如上所示，单个域名解析时，返回固定格式为 IPv4、IPv6，批量域名解析时，返回格式为 `IpSet{v4Ips=[], v6Ips=[], ips=[]}` 。业务可按需获取 IPv6 地址进行 URL 请求。

使用 IPv6 地址进行 URL 请求时，需添加方框号 `[ ]` 进行处理，例如： `http://[64:ff9b::b6fe:7475]/` 。

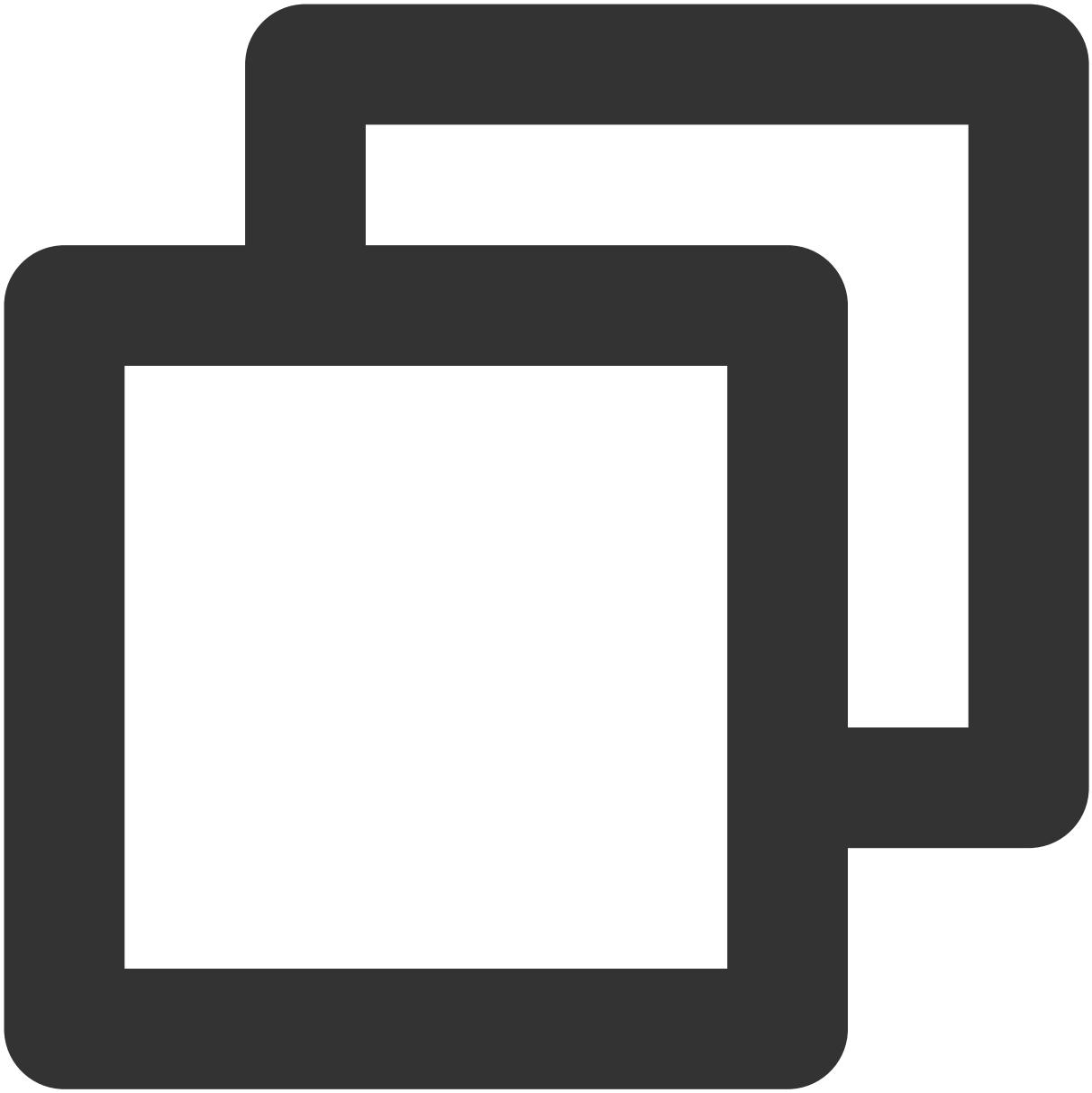
Android SDK 最佳实践

OkHttp 接入

最近更新时间：2023-06-12 15:00:34

OkHttp

OkHttp 提供了 DNS 接口，用于向 OkHttp 注入 DNS 实现。得益于 OkHttp 的良好设计，使用 OkHttp 进行网络访问时，实现 DNS 接口即可接入 HTTPDNS 进行域名解析，在较复杂场景（HTTP/HTTPS/WebSocket + SNI）下也不需要做额外处理，侵入性极小。示例如下：



```
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();
```

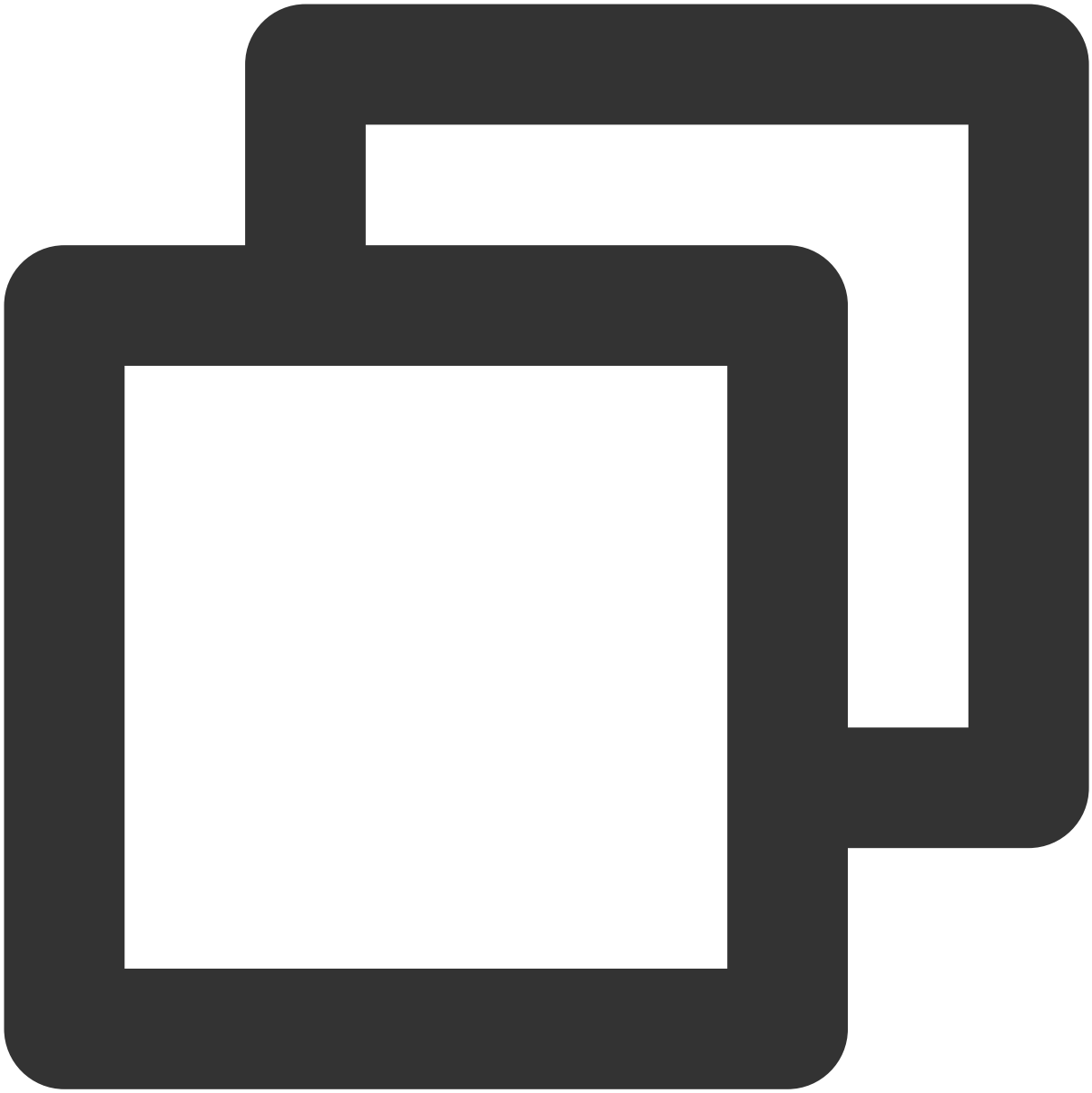
```
                }
                List<InetAddress> inetAddressList = new ArrayList<>(ipArr.length);
                for (String ip : ipArr) {
                    if ("0".equals(ip)) {
                        continue;
                    }
                    try {
                        InetAddress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

注意

实现 DNS 接口，即表示所有经由当前 OkHttpClient 实例处理的网络请求都会经过 HTTPDNS。如果您只有少部分域名是需要通过 HTTPDNS 进行解析，建议您在调用 HTTPDNS 域名解析接口之前先进行过滤。

Retrofit + OkHttp

Retrofit 实际上是一个基于 OkHttp，对接口做了一层封装桥接的 lib。因此只需要仿 OkHttp 的接入方式，定制 Retrofit 中的 OkHttpClient，即可方便地接入 HTTPDNS。示例如下：



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

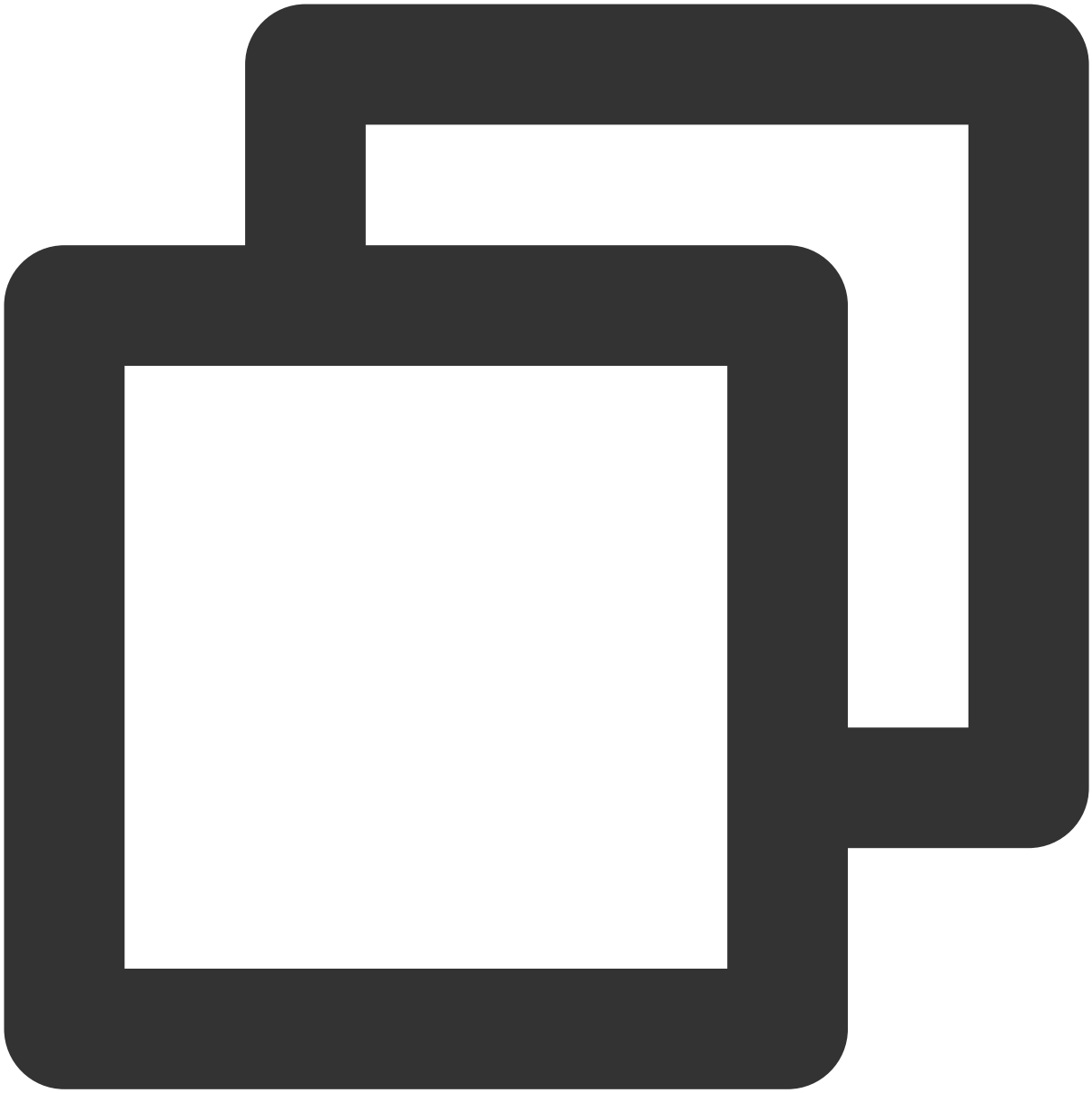
WebView 实践

最近更新时间：2023-06-12 15:00:53

Android 系统提供了 API 以实现 WebView 中的网络请求拦截与自定义逻辑注入。我们可以通过该 API 拦截 WebView 的各类网络请求，截取 URL 请求的 HOST，调用 HTTPDNS 解析该 HOST，通过得到的 IP 组成新的 URL 来进行网络请求。示例如下：

注意

由于 shouldInterceptRequest(WebView view, WebResourceRequest request) 中 WebResourceRequest 没有提供请求 body 信息，所以只能成功拦截 get 请求，无法拦截 post 请求。



```
WebSettings webSettings = mWebView.getSettings();
// 使用默认的缓存策略, cache 没有过期就用 cache
webSettings.setCacheMode(WebSettings.LOAD_DEFAULT);
// 加载网页图片资源
webSettings.setBlockNetworkImage(false);
// 支持 JavaScript 脚本
webSettings.setJavaScriptEnabled(true);
// 支持缩放
webSettings.setSupportZoom(true);
mWebView.setWebViewClient(new WebViewClient() {
    // API 21及之后使用此方法
```

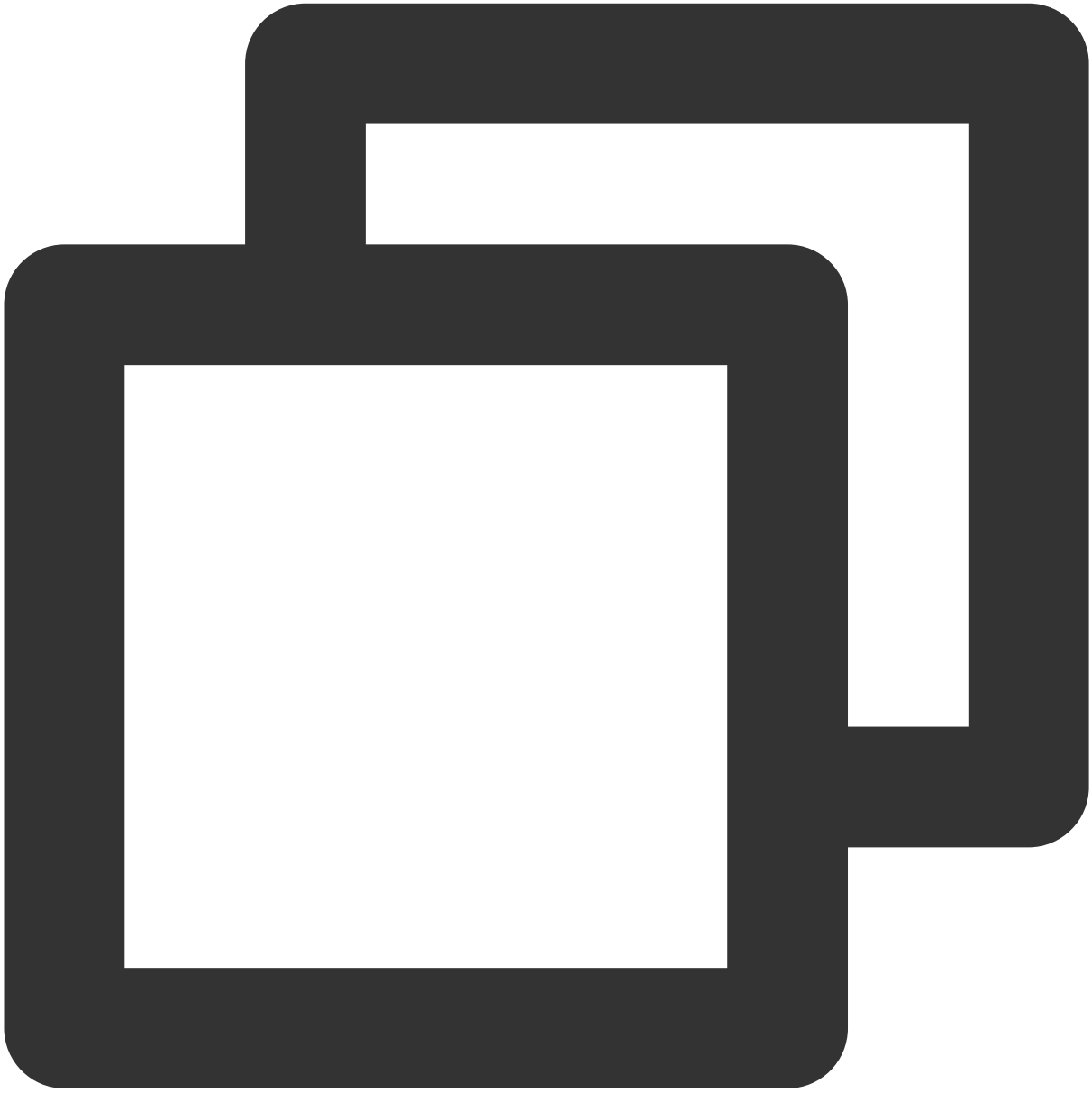
```
    @SuppressLint("NewApi")
    @Override
    public WebResourceResponse shouldInterceptRequest(WebView view, WebResourceRequ
        if (request != null && request.getUrl() != null && request.getMethod().equa
            String scheme = request.getUrl().getScheme().trim();
            String url = request.getUrl().toString();
            Log.d(TAG, "url:" + url);
            // HTTPDNS 解析 css 文件的网络请求及图片请求
            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();
                    // 获取 HTTPDNS 域名解析结果
                    String ips = MSDKDnsResolver.getInstance().getAddrByName(oldUrl
                    String[] ipArr = ips.split(";");
                    if (2 == ipArr.length && !"0".equals(ipArr[0])) { // 通过 HTTPDN
                        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;
    }

    // API 11至 API20使用此方法
    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 解析 css 文件的网络请求及图片请求
            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();
                    // 获取 HTTPDNS 域名解析结果
                    String ips = MSDKDnsResolver.getInstance().getAddrByName(oldUrl
```

```
                    String[] ipArr = ips.split(";");
                    if (2 == ipArr.length && !"0".equals(ipArr[0])) { // 通过 HTTPDN
                        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;
    }});
// 加载web资源
mWebView.loadUrl(targetUrl);
```

HttpURLConnection 接入

最近更新时间：2023-06-12 15:01:08

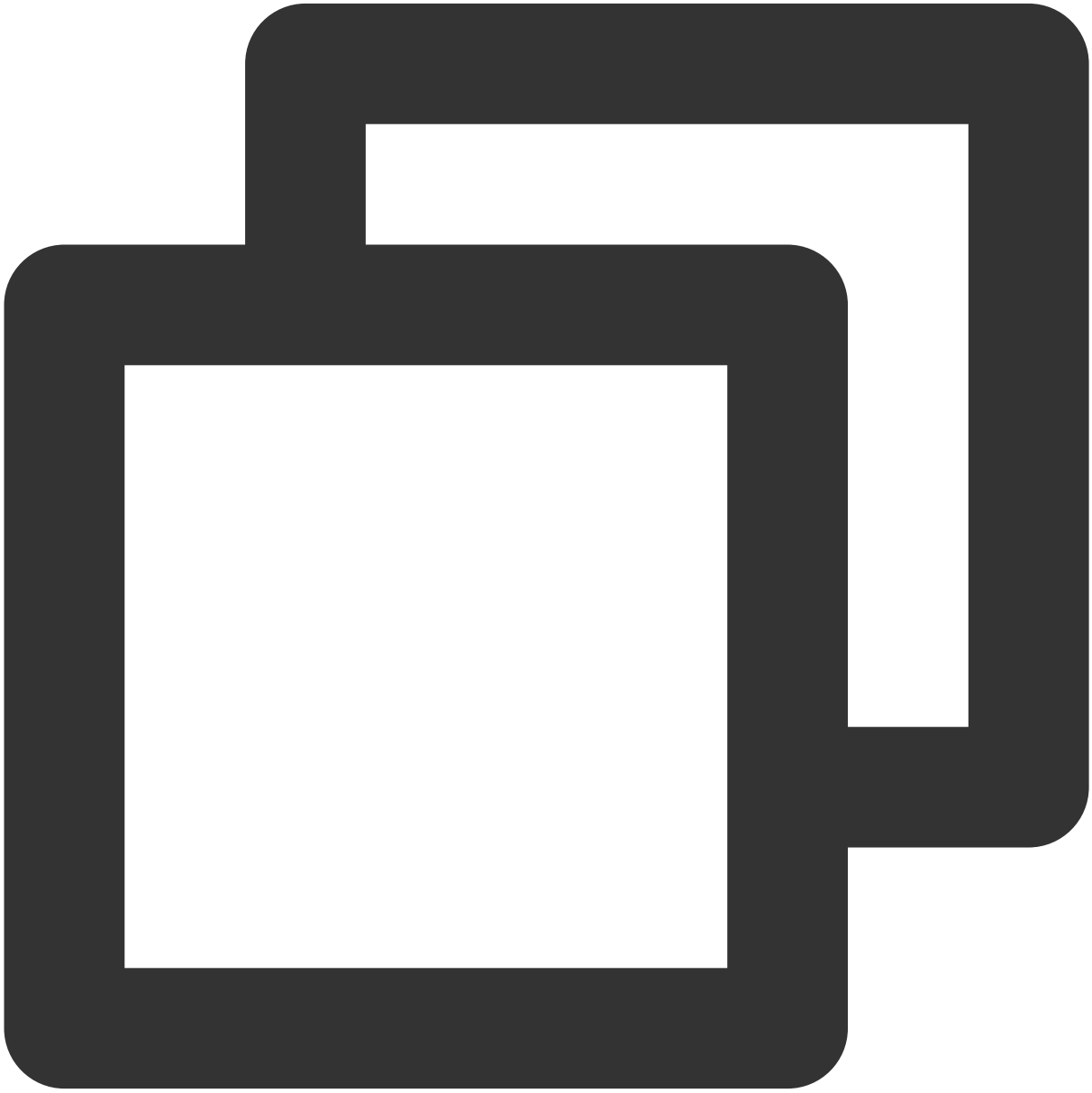
HTTPS 证书校验示例如下：



```
 // 以域名为 www.qq.com, HTTPDNS 解析得到的 IP 为192.168.0.1为例
String url = "https://192.168.0.1/"; // 业务自己的请求连接
 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); // 设置连接超时
connection.setReadTimeout(mTimeOut); // 设置读流超时
connection.connect();
```

HTTPS + SNI 示例如下：



```
// 以域名为 www.qq.com, HttpDNS 解析得到的 IP 为192.168.0.1为例
```

```
String url = "https://192.168.0.1/"; // 用 HTTPDNS 解析得到的 IP 封装业务的请求 URL
HttpsURLConnection sniConn = null;
try {
  sniConn = (HttpsURLConnection) new URL(url).openConnection();
  // 设置HTTP请求头Host域
  sniConn.setRequestProperty("Host", "www.qq.com");
  sniConn.setConnectTimeout(3000);
  sniConn.setReadTimeout(3000);
  sniConn.setInstanceFollowRedirects(false);
  // 定制SSLSocketFactory来带上请求域名 ***关键步骤
  SniSSLSocketFactory sslSocketFactory = new SniSSLSocketFactory(sniConn);
  sniConn.setSSLSocketFactory(sslSocketFactory);
  // 验证主机名和服务器验证方案是否匹配
  HostnameVerifier hostnameVerifier = new HostnameVerifier() {
      @Override
      public boolean verify(String hostname, SSLSession session) {
          return HttpsURLConnection.getDefaultHostnameVerifier().verify("原解析的域名
      }
  };
  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;
    }
}
```

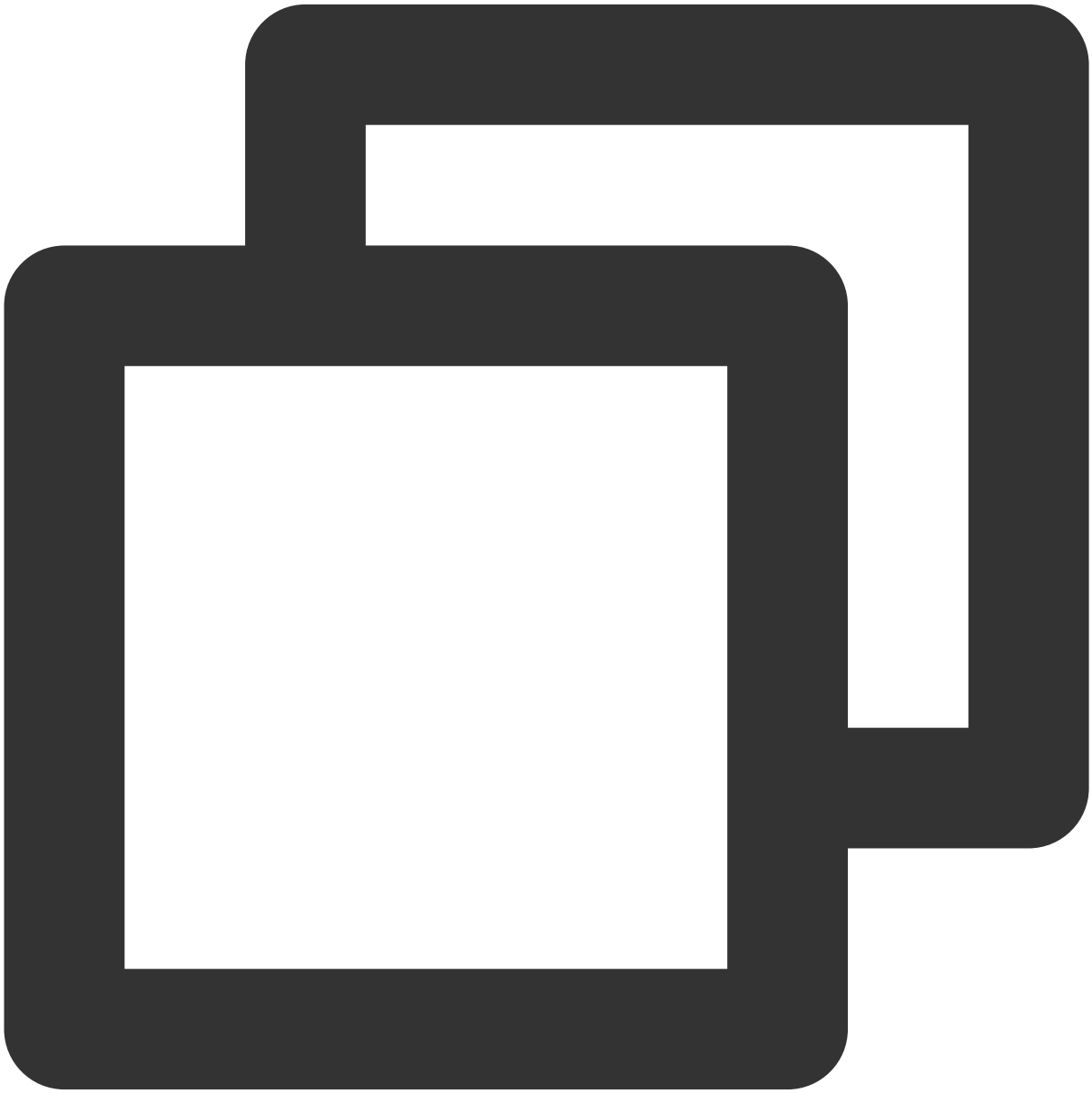
Unity 接入

最近更新时间：2023-06-08 14:46:48

接入准备

获取 SDK 文件 并复制到 Unity 项目的 Assets/Plugins/Android目录中

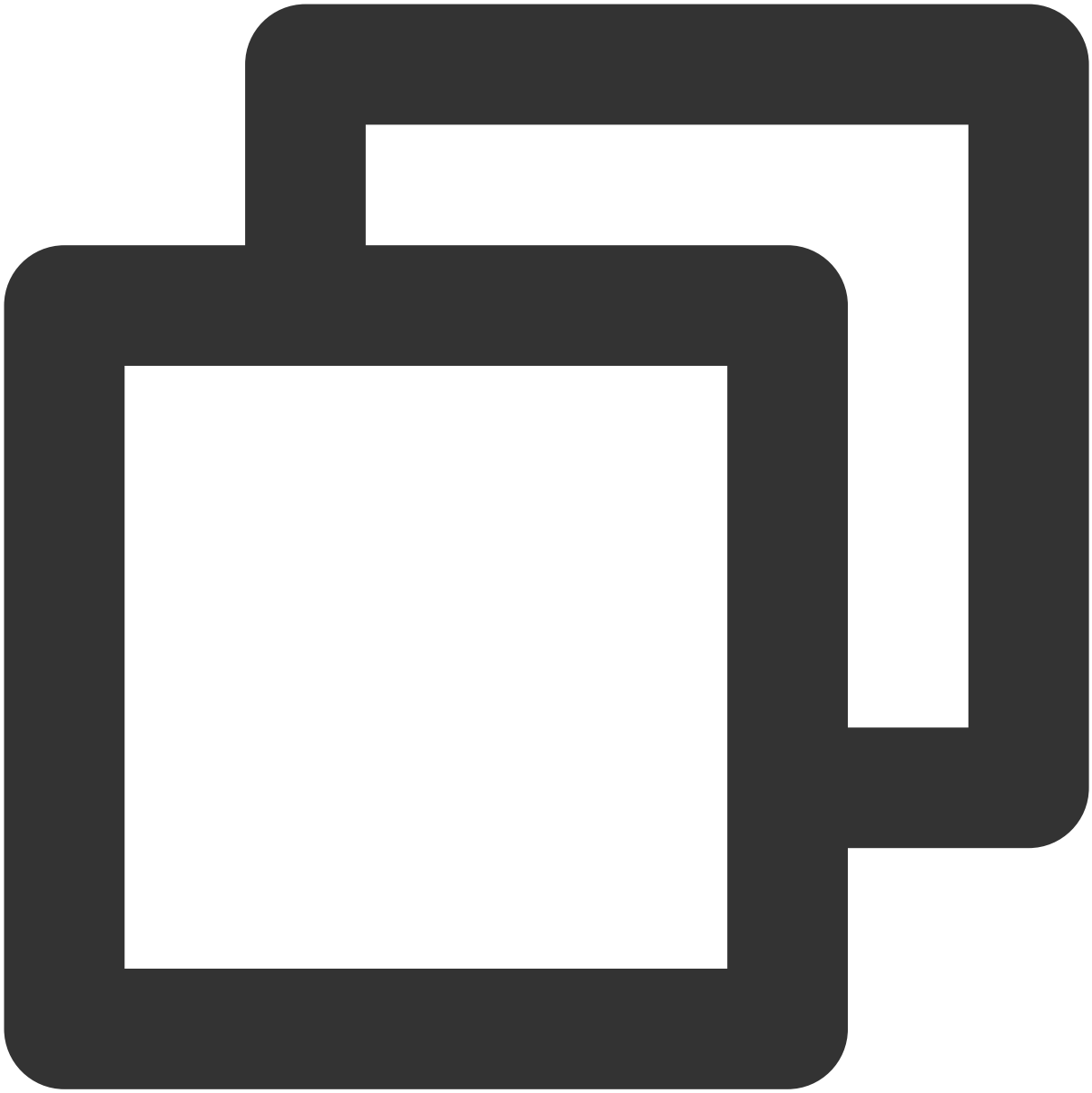
初始化 HTTPDNS。代码示例如下：



```
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
        // 初始化 HTTPDNS
        AndroidJavaObject httpDnsClass = new AndroidJavaObject("com.tencent.msdk.dns.M
        if (httpDnsClass == null) {
            return;
        }
        sHttpDnsObj = httpDnsClass.CallStatic<AndroidJavaObject>("getInstance");
        if (sHttpDnsObj == null) {
            return;
        }

        // V4.0.0开始（推荐使用），具体可查看Android SDK接入
        AndroidJavaObject dnsConfigBuilder = new AndroidJavaObject("com.tencent.msdk.d
        dnsConfigBuilder.Call<AndroidJavaObject>("dnsId", "XXX");
        dnsConfigBuilder.Call<AndroidJavaObject>("dnsIp", "XXX");
        dnsConfigBuilder.Call<AndroidJavaObject>("dnsKey", "XXX");
        // 其他配置信息...
        AndroidJavaObject dnsConfig = dnsConfigBuilder.Call<AndroidJavaObject>("build"

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

        // V4.0.0之前
        sHttpDnsObj.Call("init", contextObj, appkey, dnsid, dnskey, dnsIp, debug, time
    }
```

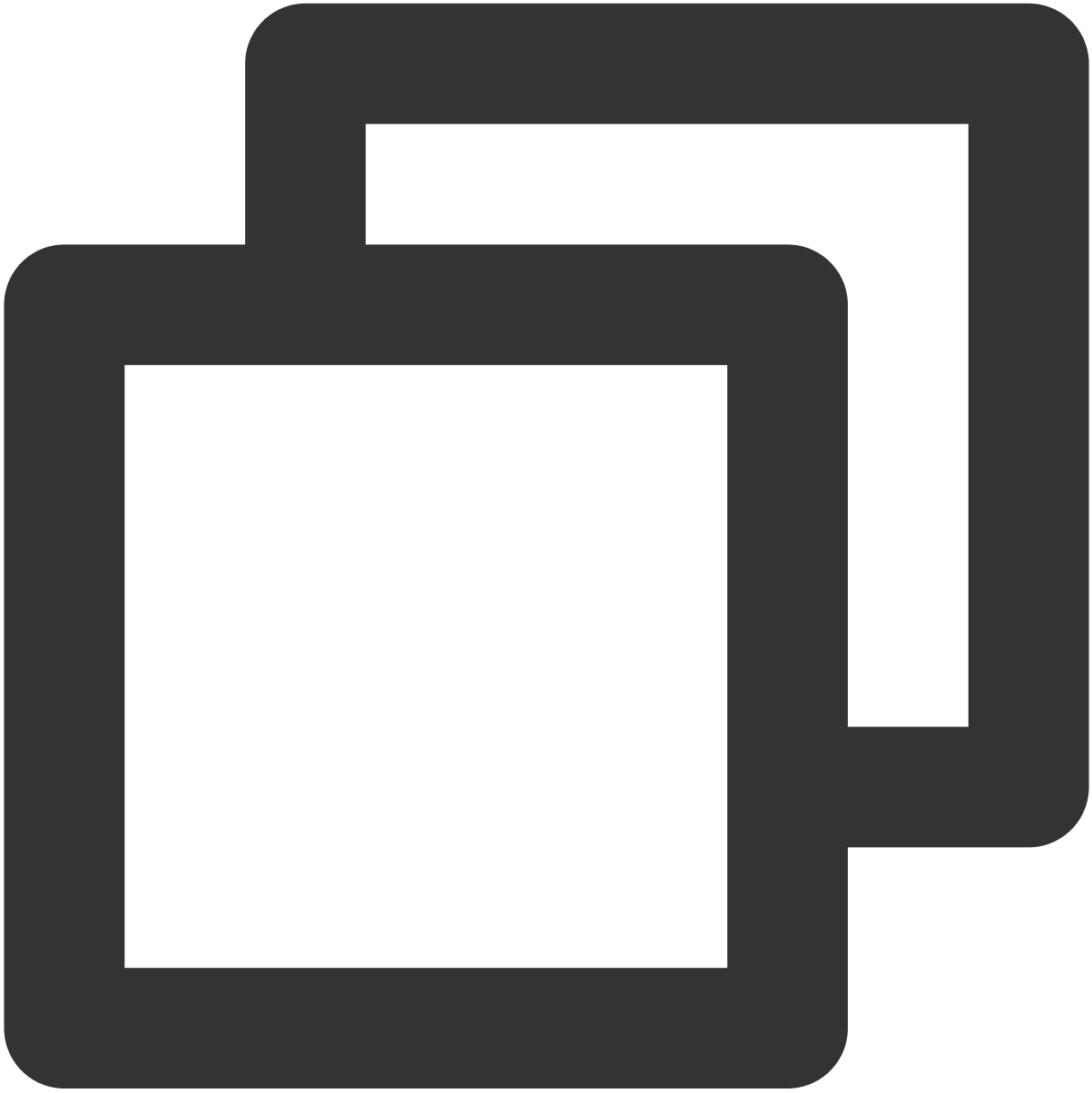
调用 getAddrByName 接口解析域名。代码示例如下：



```
// 该操作建议在子线程中或使用 Coroutine 处理
// 注意在子线程中调用需要在调用前后做 AttachCurrentThread 和 DetachCurrentThread 处理
public static string GetHttpDnsIP(string url) {
    string ip = string.Empty;
    AndroidJNI.AttachCurrentThread(); // 子线程中调用需要加上
    // 解析得到 IP 配置集合
    string ips = sHttpDnsObj.Call<string>("getAddrByName", url);
    AndroidJNI.DetachCurrentThread(); // 子线程中调用需要加上
    if (null != ips) {
        string[] ipArr = ips.Split(';');
        if (2 == ipArr.Length && !"0".Equals(ipArr[0]))
```

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

IP 直连 HTPPS 证书校验，请参见 unity官方指引。代码示例如下：



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

// CertHandler是一个自定义的证书处理器类，重写ValidateCertificate方法，实现自定义的证书验证逻
```

```
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;
    }
}
```