HTTPDNS iOS客户端接入文档

（腾讯内部业务专用）

[1. 功能介绍 2](#_Toc524014043)

[2. 安装包结构 2](#_Toc524014044)

[3. 接入步骤 2](#_Toc524014045)

[3.1 已接入MSDK或灯塔（Beacon）的业务 2](#_Toc524014046)

[3.2 未接入MSDK且未接入灯塔（Beacon）的业务 2](#_Toc524014047)

[4. API及使用示例 3](#_Toc524014048)

[4.1 设置业务基本信息 3](#_Toc524014049)

[4.2 设置用户Openid: WGSetDnsOpenId 4](#_Toc524014050)

[4.3 域名解析接口 4](#_Toc524014051)

[5. 注意事项 7](#_Toc524014052)

[6. 实践场景 8](#_Toc524014053)

[6.1 Unity工程接入 8](#_Toc524014054)

[6.2 Https场景下（非SNI）使用HttpDns解析结果 10](#_Toc524014055)

[6.3 SNI（单IP多HTTPS证书）场景下使用HttpDns解析结果 15](#_Toc524014056)

## 功能介绍

HttpDns的主要功能是为了有效的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。

## 安装包结构

压缩文件中包含demo工程，其中包含：

|  |  |
| --- | --- |
| MSDKDns.framework | 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\_C11.framework | 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。 |

## 接入步骤

### 3.1 已接入MSDK或灯塔（Beacon）的业务

仅需引入位于HTTPDNSLibs目录下的MSDKDns.framework（或MSDKDns\_C11.framework，根据工程配置选其一）即可。

### 3.2 未接入MSDK且未接入灯塔（Beacon）的业务

灯塔(beacon)SDK是腾讯灯塔团队开发的用于移动应用统计分析的SDK, HttpDNS SDK使用灯塔(beacon)SDK收集域名解析质量数据, 辅助定位问题

* 引入依赖库（位于HTTPDNSLibs目录下）：
  + BeaconAPI\_Base.framework
  + MSDKDns.framework（或MSDKDns\_C11.framework，根据工程配置选其一）
* 引入系统库：
  + libz.tdb
  + libsqlite3.tdb
  + libc++.tdb
  + Foundation.framework
  + CoreTelephony.framework
  + SystemConfiguration.framework
  + CoreGraphics.framework
  + Security.framework
* 并在application:didFinishLaunchingWithOptions:加入注册灯塔代码：

//已正常接入灯塔的业务无需关注以下代码，未接入灯塔的业务调用以下代码注册灯塔

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NSString \* appkey = @"业务的灯塔appkey，由灯塔官网注册获取";

[BeaconBaseInterface setAppKey:appkey];

[BeaconBaseInterface enableAnalytics:@"" gatewayIP:nil];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**注意：需要在Other linker flag里加入-ObjC标志。**

## API及使用示例

### 设置业务基本信息

/\*\*

设置业务基本信息（腾讯内部及代理业务使用）

@param appkey 业务appid，同手Q APPID

@param 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

@param timeout 超时时间，单位ms，如设置0，则设置为默认值2000ms

@param useHttp 是否使用http路解析，YES：使用http路解析，NO：使用https路解析，强烈建议使用http路解析，解析速度更快

@return YES:设置成功 NO:设置失败

\*/

- (BOOL) WGSetDnsAppKey:(NSString \*) appkey Debug:(BOOL)debug TimeOut:(int)timeout UseHttp:(BOOL)useHttp;

示例代码，接口调用示例：

BOOL result = [[MSDKDns sharedInstance] WGSetDnsAppKey:@"手Q APPID" Debug:YES TimeOut:1000 UseHttp:YES];

### 设置用户Openid: WGSetDnsOpenId

/\*\*

设置用户Openid，腾讯内部及代理业务关注

@param dnsOpenId 用户openid

@return YES:成功 NO:失败

\*/

- (BOOL) WGSetDnsOpenId:(NSString \*) dnsOpenId;

示例代码，接口调用示例：

// 腾讯内部及代理业务需在拿到openid成功后调用此接口

BOOL result = [[MSDKDns sharedInstance] WGSetDnsOpenId:@"xxxxxxxxxxxxxxx"];

### 域名解析接口

获取IP共有两个接口，同步接口**WGGetHostByName**，异步接口**WGGetHostByNameAsync**，引入头文件，调用相应接口即可。

返回的地址格式为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。

注意：使用ipv6地址进行URL请求时，需加方框号[ ]进行处理，例如：

[http://[64:ff9b::b6fe:7475]/\*\*\*\*\*\*\*\*\*](http://[64:ff9b::b6fe:7475]/*********)

**使用建议：**

1、ipv6为0，直接使用ipv4地址连接

2、ipv6地址不为0，优先使用ipv6连接，如果ipv6连接失败，再使用ipv4地址进行连接

#### 4.3.1 同步接口: WGGetHostByName

/\*\*

\* 同步接口

\* @param domain 域名

\* @return 查询到的IP数组，超时（1s）或者未未查询到返回[0,0]数组

\*/

- (NSArray\*) WGGetHostByName:(NSString\*) domain;

示例代码，接口调用示例：

NSArray\* ipsArray = [[MSDKDns sharedInstance] WGGetHostByName: @"www.qq.com"];

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地址进行连接，注意格式，ipv6需加方框号[ ]进行处理，例如：http://[64:ff9b::b6fe:7475]/

} 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

}

}

#### 4.3.2 异步接口: WGGetHostByNameAsync

/\*\*

\* 异步接口

\* @param domain 域名

\* @return 查询到的IP数组，超时（1s）或者未未查询到返回[0,0]数组

\*/

- (void) WGGetHostByNameAsync:(NSString\*) domain returnIps:(void (^)(NSArray\* ipsArray))handler;

示例代码：

**接口调用示例1**：等待完整解析过程结束后，拿到结果，进行连接操作

[[MSDKDns sharedInstance] WGGetHostByNameAsync:domain returnIps:^(NSArray \*ipsArray) {

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：http://[64:ff9b::b6fe:7475]/

} 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

}

}

}];

**接口调用示例2**：无需等待，可直接拿到缓存结果，如无缓存，则result为nil

\_\_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

}

**注意**：业务可根据自身需求，任选一种调用方式：

**示例1**，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。

**示例2**，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。

## 注意事项

1. 如果客户端的业务是与host绑定的，比如是绑定了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。

* 以NSURLConnection为例：

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 cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval: timeOut];

[mutableReq setValue:@"原域名" forHTTPHeaderField:@"host"];

NSURLConnection\* connection = [[NSURLConnection alloc] initWithRequest:mutableReq delegate:self];

[connection start];

* 以curl为例：

假设你要访问www.qq.com，通过HTTPDNS解析出来的IP为192.168.0.111，那么通过这个方式来调用即可：

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

* 以Unity的WWW接口为例：

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);

}

1. 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。

* 检测是否使用了HTTP代理：

- (BOOL)isUseHTTPProxy {

CFDictionaryRef dicRef = CFNetworkCopySystemProxySettings();

const CFStringRef proxyCFstr = (const CFStringRef)CFDictionaryGetValue(dicRef, (const void\*)kCFNetworkProxiesHTTPProxy);

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, (const void\*)kCFNetworkProxiesHTTPSProxy);

NSString\* proxy = (\_\_bridge NSString \*)proxyCFstr;

if (proxy) {

return YES;

} else {

return NO;

}

}

## 实践场景

### 6.1 Unity工程接入

* 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**；**
     1. 如使用同步接口**HttpDns.GetAddrByName**，直接调用接口即可；
     2. 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。

并建议进行如下处理：

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:ff9b::b6fe:7475]/

} else if(!sArray [0].Equals ("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

HttpDns. GetAddrByName (domainStr);

}

}

* 1. 将unity工程打包为xcode工程后，引入所需依赖库；
  2. 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：





* 1. 按照所需接口调用即可。

### 6.2 Https场景下（非SNI）使用HttpDns解析结果

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。

1. 以NSURLConnection接口为例，实现以下两个方法：

- (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, (\_\_bridge CFStringRef)domain)];

} 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serverTrust可以被验证通过

\* <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 == kSecTrustResultProceed);

}

-(void)connection:(NSURLConnection\*)connection willSendRequestForAuthenticationChallenge:(NSURLAuthenticationChallenge \*)challenge

{

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模式下会进行该身份验证流程），

\* 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:NSURLAuthenticationMethodServerTrust])

{

if([self evaluateServerTrust:challenge.protectionSpace.serverTrust

forDomain:host]) {

/\*

\* 验证完以后，需要构造一个NSURLCredential发送给发起方

\*/

NSURLCredential \*credential = [NSURLCredential

credentialForTrust:challenge.protectionSpace.serverTrust];

[[challenge sender] useCredential:credential forAuthenticationChallenge:challenge];

} 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:challenge];

}

}

1. 以NSURLSession接口为例，实现以下两个方法：

- (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, (\_\_bridge CFStringRef)domain)];

} 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serverTrust可以被验证通过

\* <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 == kSecTrustResultProceed);

}

- (void)URLSession:(NSURLSession \*)session task:(NSURLSessionTask \*)task

didReceiveChallenge:(NSURLAuthenticationChallenge \*)challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition disposition, NSURLCredential \* \_\_nullable credential))completionHandler

{

if (!challenge) {

return;

}

NSURLSessionAuthChallengeDisposition disposition = NSURLSessionAuthChallengePerformDefaultHandling;

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:NSURLAuthenticationMethodServerTrust]) {

if ([self evaluateServerTrust:challenge.protectionSpace.serverTrust

forDomain:host]) {

disposition = NSURLSessionAuthChallengeUseCredential;

credential = [NSURLCredential

credentialForTrust:challenge.protectionSpace.serverTrust];

} 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);

}

1. 以Unity的WWW接口为例：

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";

为：

const char\* WWWDelegateClassName = "UnityWWWConnectionSelfSignedCertDelegate";

//const char\* WWWDelegateClassName = "UnityWWWConnectionDelegate";

### 6.3 SNI（单IP多HTTPS证书）场景下使用HttpDns解析结果

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。它的工作原理如下：

* 在连接到服务器建立SSL链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：

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。

具体示例参见Demo，部分代码如下：

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:hostFirstRange withString:ip];

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 defaultSessionConfiguration];

NSArray \*protocolArray = @[ [MSDKDnsHttpMessageTools class] ];

configuration.protocolClasses = protocolArray;

NSURLSession \*session = [NSURLSession sessionWithConfiguration:configuration delegate:self delegateQueue:[NSOperationQueue mainQueue]];

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", @"val2"];

// [\_request addValue:postStr forHTTPHeaderField:@"originalBody"];

// \_request.HTTPMethod = @"POST";

// NSURLSessionConfiguration \*configuration = [NSURLSessionConfiguration defaultSessionConfiguration];

// NSArray \*protocolArray = @[ [CFHttpMessageURLProtocol class] ];

// configuration.protocolClasses = protocolArray;

// NSURLSession \*session = [NSURLSession sessionWithConfiguration:configuration delegate:self delegateQueue:[NSOperationQueue mainQueue]];

// 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，其它域名不做处理；
* 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；

**建议使用WGSetHijackDomainArray仅拦截SNI场景下的域名，避免拦截其它场景下的域名。**