移动解析 HttpDNS

SDK 文档

产品文档

腾讯云

文档目录

SDK 文档
iOS SDK

最近更新时间：2018-09-19 20:27:00

功能介绍

HttpDNS 的主要功能是为了有效地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。原理是使用 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：

demo 内容	说明
MSDKDns.framework	适用于“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“Build Setting->C++ Language Dialect”和“Build Setting->C++ Standard Library”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

已接入灯塔（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。

未接入灯塔（Beacon）的业务

1. 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）
2. 引入系统库：
   - libz.tdb

- libsqlite3.tdb
- libstdc++.tdb
- libstdc++.6.0.9.tdb
- libc++.tdb
- Foundation.framework
- CoreTelephony.framework
- SystemConfiguration.framework
- CoreGraphics.framework
- Security.framework

3. 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，由腾讯云官网注册获取";
[BeaconBaseInterface setAppKey:appkey];
[BeaconBaseInterface enableAnalytics:@"" gatewayIP:nil];
//******************************
```

> **注意：**
> 请在 Other linker flag 里加入 -ObjC 标志。

API 及使用示例

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。返回格式的详细说明如下：

- [IPv4, 0]：一般业务使用的情景中，绝大部分均会返回这种格式的结果，即不存在 IPv6 地址，仅返回 IPv4 地址给业务；
- [IPv4, IPv6]：发生在 IPv6 环境下，IPv6 及 IPv4 地址均会返回给业务；
- [0, 0]：在极其少数的情况下，会返回该格式给业务，此时 HttpDNS 与 LocalDNS 请求均超时，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]/***** ****

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。

获取 IP

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;
```

接口调用示例代码：

```
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
}
}
```

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
}
```

您可根据自身需求，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。

设置业务基本信息

```
/**
设置业务基本信息（腾讯云业务使用）

@param appkey 业务appkey，腾讯云官网（https://console.cloud.tencent.com/httpdns）申请获得，用于上报
@param dnsid dns解析id，即授权id，腾讯云官网（https://console.cloud.tencent.com/httpdns）申请获得，用于域名解析鉴权
@param dnsKey dns解析key，即授权id对应的key，腾讯云官网（https://console.cloud.tencent.com/httpdns）申请获得，用于域名解析鉴权
@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

@return YES:设置成功 NO:设置失败
*/
- (BOOL) WGSetDnsAppKey:(NSString *) appkey DnsID:(int)dnsid DnsKey:(NSString *)dnsKey Debug:(BOOL)debug TimeOut:(int)timeout;
```

```
[[MSDKDns sharedInstance] WGSetDnsAppKey: @"业务appkey，由腾讯云官网申请获得" DnsID:dns解析id DnsKey:@"dns解析key" Debug:YES TimeOut:2000];
```

注意事项

1. 如果客户端的业务已与 Host 绑定，例如绑定了 Host 的 HTTP 服务或 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 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);
}
```

2. 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*)kC
FNetworkProxiesHTTPProxy);
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*)kC
FNetworkProxiesHTTPSProxy);
NSString* proxy = (__bridge NSString *)proxyCFstr;
if (proxy) {
return YES;
} else {
return NO;
}
}
```

实践场景

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。
   - 若使用同步接口 **HttpDns.GetAddrByName**，直接调用接口即可；
   - 若使用异步接口 **HttpDns.GetAddrByNameAsync**，还需设置回调函数 **onDnsNotify(string ipString)**，函数名可自定义。并建议添加下面的 代码。
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：

```
//
//  MSDKDnsUnityManager.h
//  MSDKDns
//
//  Created by fu chunhui on 16/7/25.
//  Copyright © 2016年 Tencent. All rights reserved.
//

#ifndef MSDKDnsUnityManager_h
#define MSDKDnsUnityManager_h

#define UnityReceiverObject "ClickObject"

// 如引入framework为MSDKDns时，引入头文件<MSDKDns/MSDKDns.h>
#import <MSDKDns/MSDKDns.h>
// 如引入framework为MSDKDns_C11，则改为引入头文件<MSDKDns_C11/MSDKDns.h>
// #import <MSDKDns_C11/MSDKDns.h>

@interface MSDKDnsUnityManager : NSObject

- (NSString*)GetHostByName:(const char*) domain;
+ (id) sharedInstance;

@end

#endif /* MSDKDnsUnityManager_h */
```

如使用异步解析接口，需要关注：
“ClickObject”为对应的GameObject名称

如只使用同步解析接口无需关注

```
void WGGetHostByNameAsync(const char* domain){
    [[MSDKDns sharedInstance] WGGetHostByNameAsync:[NSString stringWithUTF8String:domain] returnIps:^(NSArray *
        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);
        }
    }];
}
```

对应回调函数名称

5. 按照所需接口调用即可。

建议添加的代码：

```
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: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);
}
}
```

普通 HTTPS 场景（非 SNI）

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：

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];
}
}
```

2. 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);
}
```

3. 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，将下述代码：

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

   修改为：

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

HTTPS SNI（单 IP 多 HTTPS 证书）场景

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：

需调用以下接口来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。

Android SDK

最近更新时间：2018-06-05 18:14:14

概述

目前域名解析普遍存在域名劫持的情况，这给我们的运维和安全带来不少挑战，为解决这个问题，我们引入 HttpDNS 功能。为保证能获取 DNS，该功能共分为两个层级：HttpDNS、LocalDNS。HttpDNS 获取成功，直接返回 HttpDNS 的值；HttpDNS 获取失败时，则返回 LocalDNS 的值；若两种情况都获取失败，则返回空值。
您可以通过以下方式获取智营解析 Android SDK：
从 Github 获取最新版本SDK >>

接入

步骤 1：AndroidMainfest 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. CHANGE_WIFI_STATE" />
<uses-permission android:name="android. permission. READ_PHONE_STATE" />
<uses-permission android:name="android. permission. INTERNET" />
<! -- DNS接收网络切换广播 -->
<receiver
android:name="com.tencent.msdk.dns. HttpDnsCache$ConnectivityChangeReceiver"
android: label="NetworkConnection" >
<intent-filter>
<action android:name=  “android.net. conn. CONNECTIVITY_CHANGE" />
</intent-filter>
</receiver>
```

步骤 2：接入 HttpDns 库

将 `HttpDnsLibs\httpdns_xxxx.jar` 库文件拷贝至工程 libs 相应的位置，将 `HttpDnsLibs\dnsconfig.ini` 配置文件拷贝到应用 `Android\assets` 目录下；

> **注意：**
> 拷贝 `dnsconfig.ini` 文件前，先修改此文件里的相关配置，但不要改变文件原有的编码格式。
> 具体修改方法如下：

修改项	修改字段	修改方法
外部厂商和内部厂商开关	IS_COOPERATOR	true
外部厂商测试开关	IS_COOPERATOR_TEST	测试环境填"true"，使用正式环境填"false"，正式上线时必须为"false"
厂商上报 appID	COOPERATOR_APPID	云官网注册获得
HttpDns SDK 日志开关	IS_DEBUG	true 为打开日志开关，false 为关闭开关
服务端分配的 ID	DNS_ID	腾讯云官网注册获得
服务端分配的 KEY	DNS_KEY	腾讯云官网注册获得

步骤 3：接入依赖库

注意：
已经接入了腾讯灯塔（beacon）组件的应用请忽略此步骤。

将 HttpDnsLibs\ beacon_android_vxxxx.jar 灯塔库拷贝至游戏 libs 相应的位置。

步骤 4：HttpDns Java 接口调用

调用命令如下：

```
// 初始化灯塔：如果已经接入MSDK或者IMSDK或者单独接入了腾讯灯塔(Beacon)则不需再初始化该接口
try {
// ***注意：这里业务需要输入自己的灯塔AppKey
UserAction.setAppKey("0I000LT6GW1YGCP7");
UserAction.initUserAction(MainActivity.this);
} catch (Exception e) {
Logger.e("init beacon error:" + e.getMessage());
}

/**
* 初始化HttpDns：如果接入了MSDK，需要初始化完MSDK后再初始化HttpDns
* @param Activity 传入当前Activity
*/
MSDKDnsResolver.getInstance(). init (MainActivity.this);
```

```
/**
* 设置OpenId，已接入MSDK业务直接传MSDK OpenId，其它业务传“NULL”
* 注意：该接口返回值是布尔型，在Unity或者Cocos下调用请注意处理返回类型
* @param String openId
*/
MSDKDnsResolver.getInstance().WGSetDnsOpenId("10000");


/**
* 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，若解析失败返回空对象
* @param domain 域名(如www.qq.com)，注意：domain只能传入域名不能传入IP，返回结果需要做非空
* 判断
* @return 域名对应的解析IP结果集合
*/
String ips = MSDKDnsResolver.getInstance(). getAddrByName(domain);
```

注意事项

- 建议调用 HttpDns 同步接口时最好在子线程调用 getAddrByName(domain) 接口。
- 如果客户端的业务已与 Host 绑定，例如绑定了 Host 的 HTTP 服务或 CDN 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 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());
if (ips != null) { // 通过HTTPDNS获取IP成功，进行URL替换和Host头设置
String ip;
if (ips.contains(";")) {
ip = ips.substring(0, ips.indexOf(";"));
} else {
ip = ips;
}
String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
connection = (HttpURLConnection) new URL(newUrl).openConnection(); // 设置HTTP请求头Host
域名
connection.setRequestProperty("Host", oldUrl.getHost());
}
```

- 以 curl 为例：
  假设您要访问 www.qq.com 通过 HTTPDNS 解析出来的 IP 为 192.168.0.111，那么通过这个方式来调用即可：

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

实践场景

Unity 接入说明

1. 初始化 HttpDns 和灯塔接口。

> **注意：**
> 若已接入了腾讯灯塔则不需要初始化灯塔。

```
private static AndroidJavaObject m_dnsJo;
private static AndroidJavaClass sGSDKPlatformClass;
public static void Init() {
AndroidJavaClass jc = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
if (jc == null)
return;

AndroidJavaObject joactivety = jc.GetStatic<AndroidJavaObject>("currentActivity");
if (joactivety == null)
return;
AndroidJavaObject context = joactivety.Call<AndroidJavaObject>("getApplicationContext");
// 初始化HttpDns
AndroidJavaObject joDnsClass = new AndroidJavaObject("com.tencent.msdk.dns.MSDKDnsResolver"
);
Debug.Log(" WGGetHostByName ==========" + joDnsClass);
if (joDnsClass == null)
return;
m_dnsJo = joDnsClass.CallStatic<AndroidJavaObject>("getInstance");
Debug.Log(" WGGetHostByName ==========" + m_dnsJo);
if (m_dnsJo == null)
return;
m_dnsJo.Call("init", context);
}
```

2. 调用 HttpDns 接口解析域名。

```
// 该操作建议在子线程中处理
public static string GetHttpDnsIP( string strUrl ) {
string strIp = string.Empty;
// 解析得到IP配置集合
strIp = m_dnsJo.Call<string>("getAddrByName", strUrl);
Debug.Log( strIp );
if( strIp != null )
{
// 取第一个
string[] strIps = strIp.Split(';');
strIp = strIps[0];
}
return strIp;
}
```

H5 页面内元素 HTTP_DNS 加载

原理：

Android 原生系统提供了系统 API 以实现 WebView 中的网络请求拦截与自定义逻辑注入，我们可以通过上述拦截 WebView 的各类网络请求，截取 URL 请求的 host，然后调用 HttpDns 解析该 host，通过得到的 IP 组成新的 URL 来请求网络地址。

实现方法：

```
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, WebResourceRequest request)
{
if (request != null && request.getUrl() != null && request.getMethod().equalsIgnoreCase("get")) {
String scheme = request.getUrl().getScheme().trim();
String url = request.getUrl().toString();
Logger.d("url a: " + 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") || url
.endsWith(".gif"))) {
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.getHost());
if (ips != null) { // 通过HTTPDNS获取IP成功，进行URL替换和HOST头设置
Logger.d("HttpDns ips are: " + ips + " for host: " + oldUrl.getHost());
String ip;
if (ips.contains(";")) {
ip = ips.substring(0, ips.indexOf(";"));
} else {
ip = ips;
}
String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
Logger.d("newUrl a is: " + newUrl);
connection = (HttpURLConnection) new URL(newUrl).openConnection(); // 设置HTTP请求头Host域
connection.setRequestProperty("Host", oldUrl.getHost());
}
Logger.d("ContentType a: " + connection.getContentType());
return new WebResourceResponse("text/css", "UTF-8", connection.getInputStream());
} catch (MalformedURLException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
}
return null;
}
// API 11至 API 20 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();
Logger.d("url b: " + 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") || url
.endsWith(".gif"))) {
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.getHost());
if (ips != null) {
// 通过HTTPDNS获取IP成功，进行URL替换和HOST头设置
Logger.d("HttpDns ips are: " + ips + " for host: " + oldUrl.getHost());
String ip;
if (ips.contains(";")) {
ip = ips.substring(0, ips.indexOf(";"));
} else {
ip = ips;
}
String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
Logger.d("newUrl b is: " + newUrl);
connection = (HttpURLConnection) new URL(newUrl).openConnection();
// 设置HTTP请求头Host域
connection.setRequestProperty("Host", oldUrl.getHost());
}
Logger.d("ContentType b: " + connection.getContentType());
return new WebResourceResponse("text/css", "UTF-8", connection.getInputStream());
} catch (MalformedURLException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
}
return null;
}
});
// 加载Web资源
mWebView.loadUrl(targetUrl);
}
```

OkHttp + HttpDns 场景

```
// 方案仅做参考
String url = "http://www.qq.com";
Uri uri = Uri.parse(url);
String ip = MSDKDnsResolver.getInstance().getAddrByName("www.qq.com"); // HttpDns解析域名
if (ip != null) {
if (ip.contains(";")) {
ip = ip.substring(0, ip.indexOf(";"));
}
} else {
```

```
return;
}
String mURL = uri.toString().replaceFirst(uri.getHost(), ip);

// OkHttp GET Method
try {
String mGetURL = mURL; // 根据业务自己的服务器地址来封装
Request request = new Request.Builder().url(mGetURL).build();
OkHttpClient client = new OkHttpClient();
Response response = client.newCall(request).execute();
String mGetResult = response.body().string();
System.out.println("OkHttp Get Method result:" + mGetResult);
} catch (Exception e) {
e.printStackTrace();
}

// OkHttp POST Method
try {
String mPostURL = mURL; // 根据业务自己的服务器地址来封装
MediaType mType = MediaType.parse("application/json; charset=utf-8");
// 封装请求参数，根据业务实际情况封装
JSONObject jsonData = new JSONObject();
jsonData.put("os", "Android");
jsonData.put("api_version", 23);
jsonData.put("version", "0.0.1");
.....
String data = jsonData.toString();
RequestBody body = RequestBody.create(mType, data);
Request request = new Request.Builder().url(mPostURL).post(body).build();
OkHttpClient client = new OkHttpClient();
Response response = client.newCall(request).execute();
String mPostResult = response.body().string();
System.out.println("OkHttp POST Method result:" + mPostResult);
} catch (Exception e) {
e.printStackTrace();
}
```

HTTPS 场景

普通 HTTPS 场景

```
String url = "https://httpdns域名解析得到的IP/d?dn=&clientip=1&ttl=1&id=128"; // 业务自己的请求连接
HttpsURLConnection connection = (HttpsURLConnection) new URL(url).openConnection();
connection.setRequestProperty("Host", "原解析的域名");
```

```
connection.setHostnameVerifier(new HostnameVerifier() {
@Override
public boolean verify(String hostname, SSLSession session) {
return HttpsURLConnection.getDefaultHostnameVerifier().verify("原解析的域名", session);
}
});
connection.setConnectTimeout(mTimeOut); // 设置连接超时
connection.setReadTimeout(mTimeOut); // 设置读流超时
connection.connect();
```

HTTPS SNI（单 IP 多 HTTPS 证书）场景

```
String url = "https://" + ip + "/pghead/xxxxxxx/140"; // 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", "原解析的域名");
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", session);
}
};
sniConn.setHostnameVerifier(hostnameVerifier);
int code = sniConn.getResponseCode();// Network block
if (code >= 300 && code < 400) {
// 临时重定向和永久重定向location的大小写有区分
String location = sniConn.getHeaderField("Location");
if (location == null) {
location = sniConn.getHeaderField("location");
}
if (!(location.startsWith("http://") || location.startsWith("https://"))) {
URL originalUrl = new URL(url);
location = originalUrl.getProtocol() + "://" + "原解析的域名" + location;
}
showSniHttpsDemo(location, "原解析的域名");
} else {
```

```
// 业务自己处理服务端响应结果
DataInputStream dis = new DataInputStream(sniConn.getInputStream());
int len;
byte[] buff = new byte[4096];
StringBuilder response = new StringBuilder();
while ((len = dis.read(buff)) != -1) {
response.append(new String(buff, 0, len));
}
dis.close();
Log.d("WGGetHostByName", "response: " + response.toString());
}
} catch (Exception e) {
Log.w("WGGetHostByName", e.getMessage());
} finally {
if (sniConn != null) {
sniConn.disconnect();
}
}
```

定制SSLSocketFactory：

```
class SniSSLSocketFactory extends SSLSocketFactory {
private HttpsURLConnection conn;

public SniSSLSocketFactory(HttpsURLConnection conn) {
this.conn = conn;
}

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

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

@Override
public Socket createSocket(String host, int port, InetAddress localHost, int localPort) throws IOExcep
tion, UnknownHostException {
return null;
}

@Override
public Socket createSocket(InetAddress host, int port) throws IOException {
```

```
return null;
}

@Override
public Socket createSocket(InetAddress address, int port, InetAddress localAddress, int localPort) thr
ows IOException {
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 autoClose) throws IOExcepti
on {
String mHost = this.conn.getRequestProperty("Host");
if (mHost == null) {
mHost = host;
}
Log.i("WGGetHostByName", "customized createSocket host is: " + mHost);
InetAddress address = socket.getInetAddress();
if (autoClose) {
socket.close();
}
SSLCertificateSocketFactory sslSocketFactory = (SSLCertificateSocketFactory) SSLCertificateSocketFact
ory.getDefault(0);
SSLSocket ssl = (SSLSocket) sslSocketFactory.createSocket(address, port);
ssl.setEnabledProtocols(ssl.getSupportedProtocols());

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR1) {
Log.i("WGGetHostByName", "Setting SNI hostname");
sslSocketFactory.setHostname(ssl, mHost);
} else {
Log.d("WGGetHostByName", "No documented SNI support on Android <4.2, trying with reflection");
try {
java.lang.reflect.Method setHostnameMethod = ssl.getClass().getMethod("setHostname", String.clas
s);
setHostnameMethod.invoke(ssl, mHost);
```

```
} catch (Exception e) {
Log.w("WGGetHostByName", "SNI not useable", e);
}
}
// verify hostname and certificate
SSLSession session = ssl.getSession();
HostnameVerifier mHostnameVerifier = HttpsURLConnection.getDefaultHostnameVerifier();
if (!mHostnameVerifier.verify(mHost, session))
throw new SSLPeerUnverifiedException("Cannot verify hostname: " + mHost);
Log.i("WGGetHostByName",
"Established " + session.getProtocol() + " connection with " + session.getPeerHost() + " using " + sess
ion.getCipherSuite());
return ssl;
}
}
```

代理使用场景

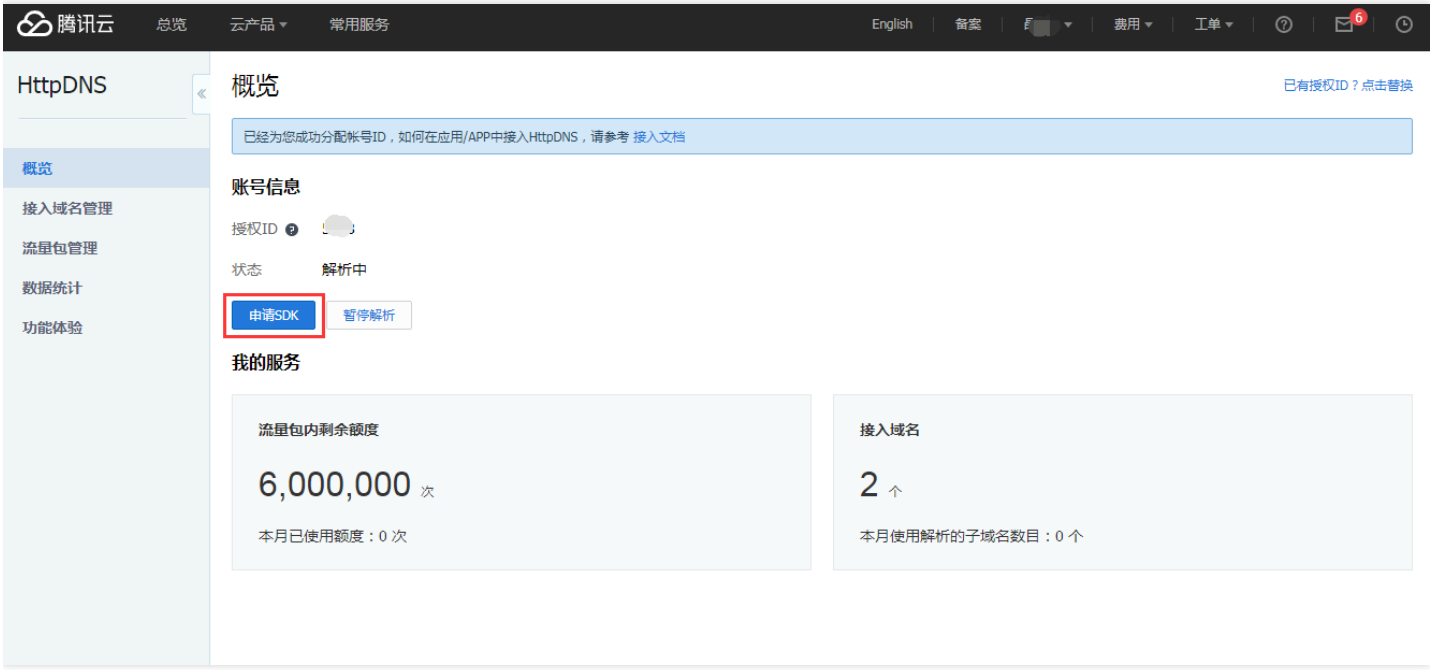
请检测本地是否使用了 HTTP 代理，如果使用了 HTTP 代理，建议不要使用 HTTPDNS 做域名解析。

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

SDK 开通流程

最近更新时间：2018-08-07 17:02:20

1.HttpDNS 服务开通后，可以在控制台提交接入 SDK 的申请。



2.填写接入 SDK 的 App 应用信息。

3.提交申请后，邮箱会收到接入 SDK 必须的 appid（Android 和 iOS 各一个），参考文档进行配置，可以快速接入 SDK 。

特殊场景-Https

最近更新时间：2017-09-26 11:54:54

该文档说明HttpDNS 的 Https 场景

iOS部分代码

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

以NSURLConnection接口为例

```
#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, (__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];
```

```
}
}
```

特殊场景-Unity接入

最近更新时间：2017-09-26 11:55:07

该文档说明Unity如何接入HttpDNS

Android部分代码

先初始化HttpDns和灯塔接口：

> 注意：若已接入msdk或者单独接入了腾讯灯塔则不用初始化灯塔。

```
private static AndroidJavaObject m_dnsJo;
private static AndroidJavaClass sGSDKPlatformClass;
public static void Init() {
AndroidJavaClass jc = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
if (jc == null)
return;
AndroidJavaObject joactivety = jc.GetStatic("currentActivity");
if (joactivety == null)
return;
AndroidJavaObject context = joactivety.Call("getApplicationContext");
// 初始化HttpDns
AndroidJavaObject joDnsClass = new AndroidJavaObject("com.tencent.msdk.dns.MSDKDnsResolver"
);
Debug.Log(" WGGetHostByName ==========" + joDnsClass);
if (joDnsClass == null)
return;
m_dnsJo = joDnsClass.CallStatic("getInstance");
Debug.Log(" WGGetHostByName ==========" + m_dnsJo);
if (m_dnsJo == null)
return;
m_dnsJo.Call("init", context);
// 初始化灯塔
AndroidJavaObject joBeaconClass = new AndroidJavaObject("com.tencent.beacon.event.UserAction"
);
if (joBeaconClass == null)
return;
m_dnsJo.Call("initUserAction", context);
}
```

再调用HttpDns接口解析域名：

```
// 该操作建议在子线程中处理
public static string GetHttpDnsIP( string strUrl ) {
string strIp = string.Empty;
// 解析得到IP配置集合
strIp = m_dnsJo.Call("getAddrByName", strUrl);
Debug.Log( strIp );
if( strIp != null )
{
// 取第一个
string[] strIps = strIp.Split(';');
strIp = strIps[0];
}
return strIp;
}
```

iOS部分代码

在cs文件中进行接口声明：

```
#if UNITY_IOS
[DllImport("__Internal")]
private static extern string WGGetHostByName(string domain);
#endif
```

在需要进行域名解析的部分，调用WGGetHostByName(string domain)方法，并建议进行如下处理：

```
string ips = WGGetHostByName(domainStr);
string[] sArray=ips.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地址进行连接，注意格式，ipv6需加方框号[ ]进行处理，例如：http://[64:ff9b::b6fe:7475]/
} else {
//使用ipv4地址进行连接
}
}
```

将unity工程打包为xcode工程，并按如上说明，引入依赖库等操作即可。

特殊场景-H5页面

最近更新时间：2018-05-28 17:58:07

该文档说明 H5 页面内元素 HTTP_DNS 的加载

Android 部分代码

原理

Android 原生系统提供了系统 API 以实现 WebView 中的网络请求拦截与自定义逻辑注入，我们可以通过上述拦截 WebView 的各类网络请求，截取 URL 请求的 host，然后调用 HttpDns 解析该 host，通过得到的 IP 组成新的 URL 来请求网络地址。

实现方法

```
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, WebResourceRequest request)
{
if (request != null && request.getUrl() != null && request.getMethod().equalsIgnoreCase("get")) {
String scheme = request.getUrl().getScheme().trim();
String url = request.getUrl().toString();
Logger.d("url a: " + 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") || url .endsWith(".jif"))) {
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.getHost());
if (ips != null) { // 通过 HTTPDNS 获取 IP 成功，进行 URL 替换和 HOST 头设置
Logger.d("HttpDns ips are: " + ips + " for host: " + oldUrl.getHost());
```

```
String ip;
if (ips.contains(";")) {
ip = ips.substring(0, ips.indexOf(";"));
} else {
ip = ips;
}
String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
Logger.d("newUrl a is: " + newUrl);
connection = (HttpURLConnection) new URL(newUrl).openConnection(); // 设置 HTTP 请求头 Host 域
connection.setRequestProperty("Host", oldUrl.getHost());
}
Logger.d("ContentType a: " + connection.getContentType());
return new WebResourceResponse("text/css", "UTF-8", connection.getInputStream());
} 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();
Logger.d("url b: " + 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") || url
.endsWith(".jif"))) {
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.getHost());
if (ips != null) {
// 通过 HTTPDNS 获取 IP 成功，进行 URL 替换和 HOST 头设置
Logger.d("HttpDns ips are: " + ips + " for host: " + oldUrl.getHost());
String ip;
if (ips.contains(";")) {
ip = ips.substring(0, ips.indexOf(";"));
} else {
ip = ips;
}
```

```
String newUrl = url.replaceFirst(oldUrl.getHost(), ip);
Logger.d("newUrl b is: " + newUrl);
connection = (HttpURLConnection) new URL(newUrl).openConnection();
// 设置 HTTP 请求头 Host 域
connection.setRequestProperty("Host", oldUrl.getHost());
}
Logger.d("ContentType b: " + connection.getContentType());
return new WebResourceResponse("text/css", "UTF-8", connection.getInputStream());
} catch (MalformedURLException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
}
return null;
}
});
// 加载 web 资源
mWebView.loadUrl(targetUrl);
}
```

iOS部分代码

原理

拦截网络请求：使用 iOS 原生的 NSURLProtocol，拦截 webview 的网络请求。然后根据网络请求 URL 的文件名后缀进行过滤。拿到过滤后的 URL 以后，截取 URL 的的域名，然后进行 HTTP_DNS 请求。然后用结果的 IP 地址拼接处原有的文件网络请求。

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:@".jpg"] || [fileExtension con
tainsString:@".css"])
```

```
{
// 修改了请求的头部信息，同步/异步请求
newRequest = [[H5ContentURLProtocol convertToNewRequest:self.request isSynchronous:NO] mutab
leCopy];
} else {
newRequest = [self.request mutableCopy];
}

[NSURLProtocol setProperty:@YES forKey:@"MyURLProtocolHandledKey" inRequest:newRequest];

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