HttpDNS(Android)接入说明文档

写在前面

3.1.0分支上，提供了一个新版本的SDK实现，主要包括了以下特性：

- 优化域名解析实现
- 提供预解析以及异步解析能力
- 提供监控解析情况的相关接口
- 提供设置SDK内部使用线程池的接口

此外，在3.1.0分支上，还提供了以下信息，使用master分支版本的业务也可以进行参考：

- 对于SDK接入验证，以及HTTP代理的使用，提供了详细的说明
- 对于几大主流网络库的兼容处理，提供了相应的原理说明
- 提供使用Sample，演示了SDK如何和几种主流网络库结合使用

原理介绍

HttpDNS服务的详细介绍可以参见文章全局精确流量调度新思路-HttpDNS服务详解。 总的来说，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

HttpDNS的Android SDK，主要提供了基于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HttpDNS服务得到域名解析结果，极端情况下如果HttpDNS服务不可用，则使用LocalDNS解析结果
- 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HttpDNS解析结果的缓存管理

接入

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" />

<!-- 灯塔 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

网络安全配置兼容

App targetSdkVersion >= 28(Android 9.0)情况下，系统默认不允许HTTP网络请求，详细信息参见 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：

- 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">182.254.116.117</domain>
        <domain includeSubdomains="false">119.29.29.29</domain>
    </domain-config>
</network-security-config>
```

接入HttpDNS

- 将HttpDNSLibs/HttpDNS_xxxx.jar拷贝至应用libs相应位置
- 将HttpDNSLibs/*/libhttpdns.so拷贝至应用jniLibs相应位置

接入灯塔

- 将HttpDNSLibs/beacon-android-xxxx.jar拷贝至应用libs相应位置
    - **注意**：已经接入了腾讯灯塔(beacon)组件的应用忽略此步
    - 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

反混淆配置

```
# HttpDNS
-keep public class com.tencent.msdk.dns.MSDKDnsResolver {*;}
-keep public class com.tencent.msdk.dns.base.jni.Jni{*;}
-keep public class
com.tencent.msdk.dns.HttpDnsCache$ConnectivityChangeReceiver {*;}
-keep public class com.tencent.msdk.dns.base.log.ILogNode {*;}

# 灯塔
-keep class com.tencent.beacon.** {*;}
```

接口调用

```
// 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.getApplicationContext());
} catch (Exception e) {
    Log.e(TAG, Init beacon failed", e);
}

/**
 * 初始化HttpDNS：如果接入了MSDK，建议初始化MSDK后再初始化HttpDNS
 *
 * @param context 应用上下文，最好传入ApplicationContext
 * @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 debug 是否开启debug日志，true为打开，false为关闭，建议测试阶段打开，正式上线时关闭
 * @param timeout dns请求超时时间，单位ms，建议设置1000
 */
MSDKDnsResolver.getInstance().init(MainActivity.this, appkey, debug,
timeout);

/**
 * 设置OpenId，已接入MSDK业务直接传MSDK OpenId，其它业务传“NULL”
 *
 * @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
 * 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地址（解析失败填“0”）
```

```
     *
     * @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);
```

接入验证

init接口中debug参数传入true，过滤TAG为“WGGetHostByName”的日志。查看到LocalDns（日志上为ldns_ip）和HttpDNS（日志上为hdns_ip）相关日志，则可以确认接入无误

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URL替换和HOST头设置
    String ip = ipArr[0];
    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，假设你要访问www.qq.com，通过HttpDNS解析出来的IP为192.168.0.111，那么可以这么访问：

```
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
}
```

实践场景

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（Https/Https + 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接口意味着所有经由当前OkHttpClient实例处理的网络请求都会经过HttpDNS。如果业务只有少部分域名是需要通过HttpDNS进行解析的，建议在调用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

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，然后调用HttpDNS解析该host，通过得到的IP组成新的URL来进行网络请求。

```
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();
            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") || 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());
                    String[] ipArr = ips.split(";");
                    if (2 == ipArr.length && !"0".equals(ipArr[0])) { // 通
过HttpDNS获取IP成功，进行URL替换和HOST头设置
                        String ip = ipArr[0];
                        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());
                    }
                    Log.d(TAG, "contentType:" +
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();
            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") || 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());
                    String[] ipArr = ips.split(";");
                    if (2 == ipArr.length && !"0".equals(ipArr[0])) { // 通
过HttpDNS获取IP成功，进行URL替换和HOST头设置
                        String ip = ipArr[0];
                        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());
                    }
                    Log.d(TAG, "contentType:" +
connection.getContentType());
                    return new WebResourceResponse("text/css", "UTF-8",
connection.getInputStream());
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                }
```

```
            }
        }
        return null;
    }});
// 加载web资源
mWebView.loadUrl(targetUrl);
```

HttpURLConnection

- Https 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",
session);
    }
});
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",
session);
        }
    };
    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, UnknownHostException {
        return null;
    }

    @Override
    public Socket createSocket(String host, int port, InetAddress
localHost, int localPort) throws IOException, 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) throws 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 IOException {
        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)
SSLCertificateSocketFactory.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(TAG, "Setting SNI hostname");
            sslSocketFactory.setHostname(ssl, realHost);
        } else {
            Log.d(TAG, "No documented SNI support on Android < 4.2,
trying with reflection");
            try {
                Method setHostnameMethod =
ssl.getClass().getMethod("setHostname", String.class);
                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.getDefaultHostnameVerifier();
        if (!hostnameVerifier.verify(realHost, session)) {
            throw new SSLPeerUnverifiedException("Cannot verify
hostname: " + realHost);
        }
        Log.i(TAG, "Established " + session.getProtocol() + "
connection with " + session.getPeerHost() + " using " +
session.getCipherSuite());
        return ssl;
    }
  }
```

Unity

- 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。示例如下：

```
private static AndroidJavaObject sHttpDnsObj;
public static void Init() {
    AndroidJavaClass unityPlayerClass = new
AndroidJavaClass("com.unity3d.player.UnityPlayer");
    if (unityPlayerClass == null) {
        return;
    }
    AndroidJavaObject activityObj =
unityPlayerClass.GetStatic<AndroidJavaObject>("currentActivity");
    if (activityObj == null) {
        return;
    }
    AndroidJavaObject contextObj = activityObj.Call<AndroidJavaObject>
("getApplicationContext");
    // 初始化HttpDNS
    AndroidJavaObject httpDnsClass = new
AndroidJavaObject("com.tencent.msdk.dns.MSDKDnsResolver");
    if (httpDnsClass == null) {
        return;
    }
    sHttpDnsObj = httpDnsClass.CallStatic<AndroidJavaObject>
("getInstance");
    if (sHttpDnsObj == null) {
        return;
    }
    sHttpDnsObj.Call("init", contextObj, appkey, debug, timeout);
}
```

- 调用getAddrByName接口解析域名 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;
}
```