# HttpClient接入HTTPDNS SDK

**HttpClient接入HTTPDNS SDK**
- 关于HostNameResolver
- HttpClient替换URL接入HTTPDNS SDK
- 注意事项

以下代码片段摘自SDK使用Sample（HttpDnsSample目录），完整代码请参考使用Sample

## 关于HostNameResolver

HttpClient实际上提供了HostNameResolver接口用于注入DNS实现，不幸的是测试发现实现HostNameResolver接口并不会影响到HttpClient的域名解析，具体分析如下：

```
// NOTE: 以下是HttpClient Android实现的部分源代码

// HostNamResolver.java
public interface HostNameResolver {

    InetAddress resolve(String hostname) throws IOException;

}

// DefaultClientConnectionOperator.java

// ...

// non-javadoc, see interface ClientConnectionOperator
public void openConnection(OperatedClientConnection conn,
                           HttpHost target,
                           InetAddress local,
                           HttpContext context,
                           HttpParams params)
    throws IOException {

    //...

    // Step1: 通过LocalDNS完成域名解析
    InetAddress[] addresses =
InetAddress.getAllByName(target.getHostName());

    for (int i = 0; i < addresses.length; ++i) {
        Socket sock = plain_sf.createSocket();
        conn.opening(sock, target);
        try {
            // Step2: PlainSocketFactory的connectSocket方法中，会尝试通过
HostNameResolver进行域名解析
            // 而传入的hostname即addresses[i].getHostAddress()，已经是
LocalDNS解析得到的IP，即HostNameResolver的域名解析不会生效
            Socket connsock = plain_sf.connectSocket(sock,
                addresses[i].getHostAddress(),
                schm.resolvePort(target.getPort()),
                local, 0, params);
            if (sock != connsock) {
                sock = connsock;
                conn.opening(sock, target);
            }
            /*
             * prepareSocket is called on the just connected
             * socket before the creation of the layered socket to
             * ensure that desired socket options such as
             * TCP_NODELAY, SO_RCVTIMEO, SO_LINGER will be set
             * before any I/O is performed on the socket. This
             * happens in the common case as
             * SSLSocketFactory.createSocket performs hostname
             * verification which requires that SSL handshaking be
             * performed.
             */
            prepareSocket(sock, context, params);
            if (layered_sf != null) {
                // Step3: 对于HTTPS请求，通过LayeredSocketFactory的
createSocket方法，基于已经创建的Socket实例，创建出SSLSocket
                Socket layeredsock = layered_sf.createSocket(sock,
                    target.getHostName(),
                    schm.resolvePort(target.getPort()),
                    true);
                if (layeredsock != sock) {
                    conn.opening(layeredsock, target);
                }
                conn.openCompleted(sf.isSecure(layeredsock), params);
            } else {
                conn.openCompleted(sf.isSecure(sock), params);
            }
            break;
        // BEGIN android-changed
        //       catch SocketException to cover any kind of connect failure
        } catch (SocketException ex) {
            if (i == addresses.length - 1) {
                ConnectException cause = ex instanceof ConnectException
                    ? (ConnectException) ex : new
ConnectException(ex.getMessage(), ex);
                throw new HttpHostConnectException(target, cause);
            }
        // END android-changed
        } catch (ConnectTimeoutException ex) {
            if (i == addresses.length - 1) {
                throw ex;
            }
        }
    }
} // openConnection
```

因此，对于HttpClient，我们只能通过替换URL的方式接入HTTPDNS SDK

## HttpClient替换URL接入HTTPDNS SDK

如<<HTTPDNS Android接入文档>>所述，对于**需要使用到域名信息**的网络请求，我们需要针对HTTP和HTTPS请求分别进行兼容

示例代码如下，有关SNI扩展设置HostnameVerifier的更详细说明，可以参考<<HttpURLConnection接入HTTPDNS SDK>>：

```
// HttpClientHelper.kt
internal object HttpClientHelper {

    private const val HTTP_PROTOCOL = "http"
    private const val HTTP_PORT = 80
    private const val HTTPS_PROTOCOL = "https"
    private const val HTTPS_PORT = 443

    private val layeredSocketFactory4Dns by lazy {
        LayeredSocketFactory4Dns.getSocketFactory()
    }

    // Step1: 定制HttpClient
    val httpClient: HttpClient by lazy {
        val httpParams = BasicHttpParams()
        SchemeRegistry()
            .apply {
                register(Scheme(HTTP_PROTOCOL,
PlainSocketFactory.getSocketFactory(), HTTP_PORT))
                register(
                    Scheme(
                        HTTPS_PROTOCOL,
                        // NOTE: 如<<HTTPDNS Android接入文档>>所述, 对于HTTPS
协议, 需要使用到域名信息的网络请求, 我们需要通过SNI扩展来指明域名信息
                        // 这里我们通过定制LayeredSocketFactory的方式来指定SNI扩
展
                        // 指定SNI扩展实现方式参考OkHttp内部实现
                        if (DnsServiceWrapper.useHttpDns)
layeredSocketFactory4Dns
```

```
                            else SSLSocketFactory.getSocketFactory(),
                            HTTPS_PORT
                        )
                    )
                }
                .let { ThreadSafeClientConnManager(httpParams, it) }
                .let { DefaultHttpClient(it, httpParams) }
    }

    fun url2HttpGet(urlStr: String): HttpGet {
        val url = URL(urlStr)
        val hostname = url.host
        val newUrl = if (DnsServiceWrapper.useHttpDns) {
            // Step2: 替换URL
            DnsLog.d("HttpClientHelper lookup for $hostname")
            val inetAddrs = DnsServiceWrapper.getAddrsByName(hostname)
            if (inetAddrs.isEmpty()) urlStr
            else {
                val inetAddr = inetAddrs[0]
                if (HTTPS_PROTOCOL == url.protocol) {
                    // NOTE: 指定SNI扩展需要域名信息, 这里我们通过
hostnameContainer保存域名和IP的映射关系, 确保layeredSocketFactory4Dns可以获取到正
确的域名信息

 layeredSocketFactory4Dns.hostnameContainer.put(inetAddr.hostAddress,
hostname)
                }
                urlStr.replace(hostname, inetAddr.toUriFormat())
            }
        } else urlStr
        // NOTE: 如<<HTTPDNS Android接入文档>>所述, 对于需要使用到域名信息的网络请
求, 我们需要通过HOST字段来指明域名信息
        return HttpGet(newUrl).apply { setHeader(HTTP.TARGET_HOST,
hostname) }
    }
}

// HostnameContainer.kt
class HostnameContainer {

    // NOTE: IP和域名并非严格的一一对应关系, 因此我们不能通过Map一类的数据结构, 通过
IP来查找域名
    // 这里我们使用ThreadLocal来保存域名, 但需要确保是同步阻塞方式发起请求, 这样后续
通过ThreadLocal获取到的域名即是我们需要的域名信息
    private val hostname2IpPairContainer = ThreadLocal<Ip2HostnamePair>()

    fun put(ip: String, hostname: String) {
        DnsLog.d("HostnameContainer put $ip to $hostname")
        hostname2IpPairContainer.set(ip to hostname)
    }

    fun get(ip: String) =
        hostname2IpPairContainer.get()?.let {
            val (expectedIp, hostname) = it
            if (expectedIp == ip) hostname else null
        }
}

typealias Ip2HostnamePair = Pair<String, String>
```

```
// LayeredSocketFactory4Dns.java
// NOTE: LayeredSocketFactory4Dns拷贝自
org.apache.http.conn.ssl.SSLSocketFactory，并基于HTTPDNS需求做少量修改
// NOTE: BEGIN HTTPDNS-changed
// 修改类名，可见性由public改为default
final class LayeredSocketFactory4Dns implements LayeredSocketFactory {
// NOTE: BEGIN HTTPDNS-changed

    // ...

    // NOTE: BEGIN HTTPDNS-added
    public final HostnameContainer hostnameContainer = new
HostnameContainer();
    // NOTE: END HTTPDNS-added

    // NOTE: BEGIN HTTPDNS-added
    private static Method sSetHostnameMethod = null;
    private static final String SET_HOSTNAME_METHOD_NAME = "setHostname";

    private static Method getSetHostnameMethod(SSLSocket sslSocket) {
        if (null == sSetHostnameMethod) {
            // NOTE: 如果无法成功反射setHostname，则SNI扩展无法成功设置
            // 这里直接消化异常，由网络库决定如何处理网络访问失败情况
            try {
                sSetHostnameMethod =
sslSocket.getClass().getMethod(SET_HOSTNAME_METHOD_NAME, String.class);
            } catch (NoSuchMethodException ignored) {
            }
        }
        return sSetHostnameMethod;
    }
    // NOTE: END HTTPDNS-added

    // non-javadoc, see interface LayeredSocketFactory
    public Socket createSocket(
            final Socket socket,
            final String host,
            final int port,
            final boolean autoClose
    ) throws IOException, UnknownHostException {
        SSLSocket sslSocket = (SSLSocket) this.socketfactory.createSocket(
                socket,
                host,
                port,
                autoClose
        );
        // NOTE: BEGIN HTTPDNS-added
        DnsLog.INSTANCE.d("LayeredSocketFactory4Dns sslSocket: %s",
sslSocket);
        // NOTE: 这里的host是HTTPDNS解析得到的IP
        String realHostname = hostnameContainer.get(host);
        Method setHostnameMethod;
        if (null != realHostname && null != (setHostnameMethod =
getSetHostnameMethod(sslSocket))) {
            DnsLog.INSTANCE.d("LayeredSocketFactory4Dns try to setHostname:
%s", realHostname);
            try {
                // NOTE: 指定SNI扩展
                // SNI扩展需确保在开始TLS握手之前指定
                // 指定SNI扩展实现方式参考OkHttp内部实现
                setHostnameMethod.invoke(sslSocket, realHostname);
            } catch (Exception ignored) {
            }
        }
        // NOTE: END HTTPDNS-added

        // BEGIN android-added
        /*
         * Make sure we have started the handshake before verifying.
         * Otherwise when we go to the hostname verifier, it directly calls
         * SSLSocket#getSession() which swallows SSL handshake errors.
         */
        sslSocket.startHandshake();
        // END android-added
        // NOTE: BEGIN HTTPDNS-changed
        // NOTE: 校验证书时，也需要使用原来的域名进行校验
        hostnameVerifier.verify(null != realHostname ? realHostname : host,
sslSocket);
        // NOTE: END HTTPDNS-changed
        // verifyHostName() didn't blowup - good!
        return sslSocket;
    }

    // ...

}
```

## 注意事项

- 兼容方案基于HttpClient Android实现的源代码进行Hack，强耦合于HttpClient的具体实现，使用时请注意测试
- Google从Android 2.3之后就不再建议使用HttpClient，条件允许的话建议换用OkHttp进行HTTP相关的网络访问