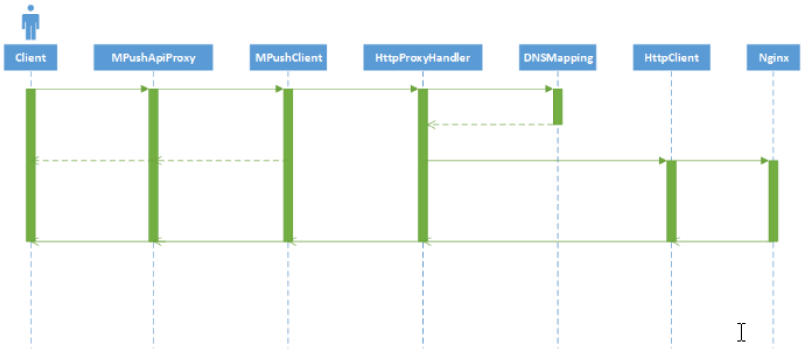
接收客户端的HTTP代理请求

```
1 //ConnectionServer#init()
2 messageDispatcher.register(Command.HTTP_PROXY, () -> new HttpProxyHandler(mPushServer), CC.mp.http.proxy_en
abled);
```

Mpush实现HTTP代理的描述：《3 HTTP代理.note》

流程分析



说明

1. Client代表App业务比如查询用户信息的接口
2. MPushApiProxy是一个工具类用于负责处理当前请求是使用普通的HTTP还是使用MPush长链接通道，这个类在SDK中说不存在的，是我们公司内部的业务，实现起来也很简单，建议Android工程中增加这么一个角色，而不是到处直接去依赖Mpush的代码，方便以后解耦。
3. MPushClient这个SDK已经提供，用于把Http协议打包成mpush协议。
4. HttpProxyHandler**包括后面的几个组件都是服务端业务组件。**用于接收客户端传过来的请求并反解为Http协议，然后通过DNSMapping找到域名对应的局域网IP，再通过内置的HttpClient，把请求转发给业务WEB服务，并把业务服务的返回值(HttpResponse)打包成MPush协议发送到客户端。
5. DNSMapping负责通过域名解析成局域网IP，并具有负载均衡以及简单的健康检查功能(针对所配置的WEB服务)
6. HttpClient目前使用的是用Netty实现的全异步的一个HttpClient，负责通过http的方式请求业务服务。
7. Nginx是业务服务，也可以是Tomcat，**特别需要建议的是链接超时时间配置长一些。**

mpush server做客户端的http请求代理流程

* 客户端，用mpush sdk，发送HTTP请求给Mpush server（ConnectionServer）；
* ConnectionServer接入服务接收到Command.HTTP_PROXY消息，包装成HTTP request，将请求转发给HTTP server端；
* HTTP Server端返回数据(HttpResponse)到mpush server，mpush将response打包成mpush协议发送给客户端

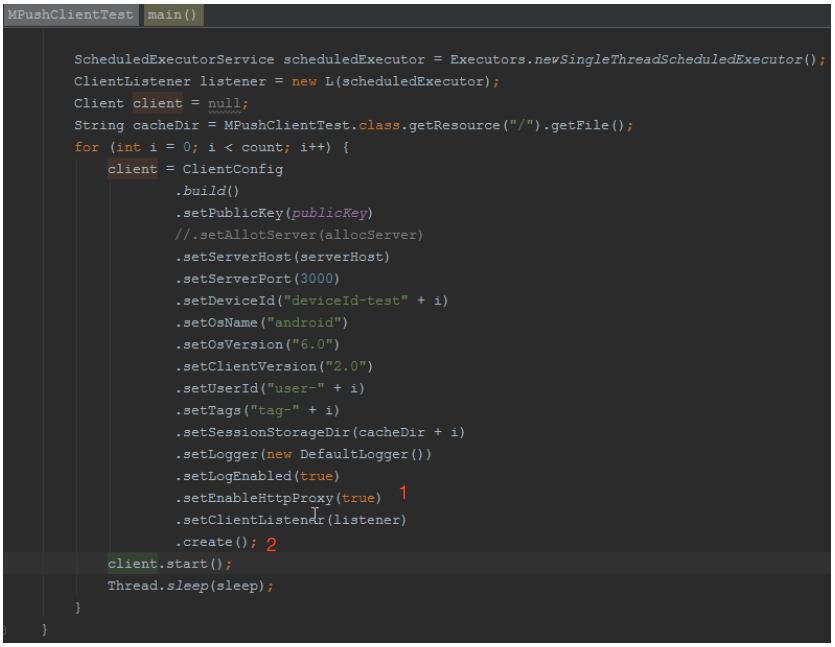
客户端- 用SDK发送HTTP请求

客户端 MPushClient 提供了一个叫sendHttp的方法，该方法用于把客户端原本要通过HTTP方式发送的请求，全部通过PUSH通道转发，实现整个链路的长链接化；通过这种方式应用大大减少Http短链接频繁的创建，不仅仅节省电量，经过测试证明请求时间比原来至少缩短一倍，而且MPush提供的还有数据压缩功能，对于比较大的数据还能大大节省流量(压缩率4-10倍)，更重要的是所有通过代理的数据都是加密后传输的，大大提高了安全性！

1、设置ClientConfig.setEnableHttpProxy(true)来启用客户端代理。
2、通过Client.sendHttp(HttpRequest request)方法来发送请求。
AndroidSDK通过com.mpush.android.MPush#sendHttpProxy(HttpRequest request)来发送比较合适。

启动客户端代理

mpush-client-java工程，com/mpush/client/MPushClient.java
1、设置ClientConfig.setEnableHttpProxy(true)



ClientConfig#create()方法，初始化MPushClient实例

```
public Client create() {
 return new MPushClient(this);
}
```

MPushClient() 构造方法，具体实现细节，参考mpush-client-java下的《1 启动-建立连接.note》章节

```
MPushClient(ClientConfig config) {
 this.config = config;
 this.logger = config.getLogger();
 //初始化消息接收处理器（各种类型）
 MessageDispatcher receiver = new MessageDispatcher();
 //如果启用了代理，注册一个处理HTTP代理请求的处理器类 HttpProxyHandler
 if (config.isEnableHttpProxy()) {
 //HTTP 请求超时处理
 this.httpRequestMgr = HttpRequestMgr.I();
  receiver.register(Command.HTTP_PROXY, new HttpProxyHandler());
  }
  //ACK 超时处理
  this.ackRequestMgr = AckRequestMgr.I();
  //客户端conn连接管理
  this.connection = new TcpConnection(this, receiver);
  this.ackRequestMgr.setConnection(this.connection);
 }
```

MPushClient#start()方法

```
@Override
public void start() {
 if (clientState.compareAndSet(State.Shutdown, State.Started)) {
 connection.setAutoConnect(true);
 connection.connect();
 logger.w("do start client ...");
 }
}
```

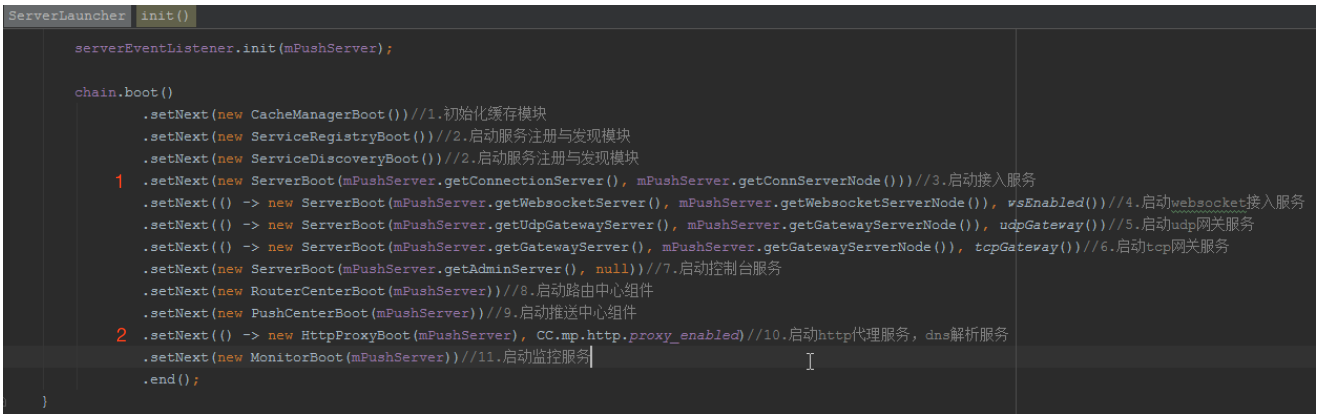
发送请求

```
@Override
public Future<HttpResponse> sendHttp(HttpRequest request) {
 if (connection.getSessionContext().handshakeOk()) {
```

```
HttpRequestMessage message = new HttpRequestMessage(connection);
message.method = request.getMethod();
message.uri = request.getUri();
message.headers = request.getHeaders();
message.body = request.getBody();
message.send();
 logger.d("<<< send http proxy, request=%s", request);
 return httpRequestMgr.add(message.getSessionId(), request);
 }
 return null;
}
```

MPush服务端- 接收and转发HTTP代理请求

Mpush server服务端，分2个服务来处理HTTP 代理请求，一个负责接收客户端HTTP代理请求，一个负责转发HTTP代理请求；



1、接收HTTP代理请求服务（ConnectionServer）

客户端 -> Mpush server（ConnectionServer）

2、转发HTTP代理请求服务(NettyHttpClient)

Mpush server(NettyHttpClient) -> HTTP Server

接收HTTP代理请求服务（ConnectionServer）

ConnectionServer启动时，注册 Command.HTTP_PROXY 的处理类 HttpProxyHandler

```
messageDispatcher.register(Command.HTTP_PROXY, () -> new HttpProxyHandler(mPushServer), CC.mp.http.proxy_en
abled);
```

接收到客户端的HTTP代理请求，将数据打包成HTTP请求，发送给HTTP Server

```
public class HttpProxyHandler extends BaseMessageHandler<HttpRequestMessage> {
 private static final Logger LOGGER = LoggerFactory.getLogger(HttpProxyHandler.class);
 private final DnsMappingManager dnsMappingManager = DnsMappingManager.create();
 private final HttpClient httpClient;

 public HttpProxyHandler(MPushServer mPushServer) {
 // 获取NettyHttpClient 服务实例，用于和Http server端进行交互(request/response)
 this.httpClient = mPushServer.getHttpClient();
 }
 @Override
 public HttpRequestMessage decode(Packet packet, Connection connection) {
 return new HttpRequestMessage(packet, connection);
 }
 @Override
 public void handle(HttpRequestMessage message) {
 try {
 //1.参数校验
```

```
18  String method = message.getMethod();
19  String uri = message.uri;
20  if (Strings.isNullOrEmpty(uri)) {
21  HttpResponseMessage
22  .from(message)
23  .setStatusCode(400)
24  .setReasonPhrase("Bad Request")
25  .sendRaw();
26  Logs.HTTP.warn("receive bad request url is empty, request={}", message);
27  }
28
29  //2.url转换
30  uri = doDnsMapping(uri);
31
32  Profiler.enter("time cost on [create FullHttpRequest]");
33  //3.包装成HTTP request
34  FullHttpRequest request = new DefaultFullHttpRequest(HTTP_1_1, HttpMethod.valueOf(method), uri, getBody(m
essage));
35  setHeaders(request, message);//处理header
36
37  Profiler.enter("time cost on [HttpClient.request]");
38  //4.发送请求
39  httpClient.request(new RequestContext(request, new DefaultHttpCallback(message)));
40  } catch (Exception e) {
41  HttpResponseMessage
42  .from(message)
43  .setStatusCode(502)
44  .setReasonPhrase("Bad Gateway")
45  .sendRaw();
46  LOGGER.error("send request ex, message=" + message, e);
47  Logs.HTTP.error("send proxy request ex, request={}, error={}", message, e.getMessage());
48  } finally {
49  Profiler.release();
50  }
51  }
52  private static class DefaultHttpCallback implements HttpCallback {
53  private final HttpRequestMessage request;
54  private int redirectCount;
55
56  private DefaultHttpCallback(HttpRequestMessage request) {
57  this.request = request;
58  }
59  @Override
60  public void onResponse(HttpResponse httpResponse) {
61  HttpResponseMessage response = HttpResponseMessage
62  .from(request)
63  .setStatusCode(httpResponse.status().code())
64  .setReasonPhrase(httpResponse.status().reasonPhrase());
65  for (Map.Entry<String, String> entry : httpResponse.headers()) {
66  response.addHeader(entry.getKey(), entry.getValue());
67  }
68  if (httpResponse instanceof FullHttpResponse) {
69  ByteBuf content = ((FullHttpResponse) httpResponse).content();
70  if (content != null && content.readableBytes() > 0) {
71  byte[] body = new byte[content.readableBytes()];
72  content.readBytes(body);
73  response.body = body;
```

```
74   response.addHeader(CONTENT_LENGTH.toString(), Integer.toString(response.body.length));
75   }
76   }
77   response.send();
78   Logs.HTTP.info("send proxy request success end request={}, response={}", request, response);
79   }
80   @Override
81   public void onFailure(int statusCode, String reasonPhrase) {
82   HttpResponseMessage
83   .from(request)
84   .setStatusCode(statusCode)
85   .setReasonPhrase(reasonPhrase)
86   .sendRaw();
87   Logs.HTTP.warn("send proxy request failure end request={}, response={}", request, statusCode + ":" + reas
onPhrase);
88   }
89   @Override
90   public void onException(Throwable throwable) {
91   HttpResponseMessage
92   .from(request)
93   .setStatusCode(502)
94   .setReasonPhrase("Bad Gateway")
95   .sendRaw();
96
97   LOGGER.error("send proxy request ex end request={}, response={}", request, 502, throwable);
98   Logs.HTTP.error("send proxy request ex end request={}, response={}, error={}", request, 502, throwable.ge
tMessage());
99   }
100  @Override
101  public void onTimeout() {
102  HttpResponseMessage
103  .from(request)
104  .setStatusCode(408)
105  .setReasonPhrase("Request Timeout")
106  .sendRaw();
107  Logs.HTTP.warn("send proxy request timeout end request={}, response={}", request, 408);
108  }
109
110  @Override
111  public boolean onRedirect(HttpResponse response) {
112  return redirectCount++ < 5;
113  }
114  }
115  private void setHeaders(FullHttpRequest request, HttpRequestMessage message) {
116  Map<String, String> headers = message.headers;
117  if (headers != null) {
118  HttpHeaders httpHeaders = request.headers();
119  for (Map.Entry<String, String> entry : headers.entrySet()) {
120  httpHeaders.add(entry.getKey(), entry.getValue());
121  }
122  }
123
124  if (message.body != null && message.body.length > 0) {
125  request.headers().add(CONTENT_LENGTH, Integer.toString(message.body.length));
126  }
127
```

```
128   InetSocketAddress remoteAddress = (InetSocketAddress)
message.getConnection().getChannel().remoteAddress();
129   String remoteIp = remoteAddress.getAddress().getHostAddress();//这个要小心，不要使用getHostName,不然会耗时
比较大
130   request.headers().add("x-forwarded-for", remoteIp);
131   request.headers().add("x-forwarded-port", Integer.toString(remoteAddress.getPort()));
132   }
133   private ByteBuf getBody(HttpRequestMessage message) {
134   return message.body == null ? Unpooled.EMPTY_BUFFER : Unpooled.wrappedBuffer(message.body);
135   }
136   private String doDnsMapping(String url) {
137   URL uri = null;
138   try {
139   uri = new URL(url);
140   } catch (MalformedURLException e) {
141   //ignore e
142   }
143   if (uri == null) {
144   return url;
145   }
146   String host = uri.getHost();
147   DnsMapping mapping = dnsMappingManager.lookup(host);
148   if (mapping == null) {
149   return url;
150   }
151   return mapping.translate(uri);
152   }
153 }
```

1、参数校验

2、URL转换（DNS）

HttpProxyDnsMappingManager.java

3、包装成HTTP request

4、发送请求给HTTP Server

转发HTTP代理请求服务(NettyHttpClient)

通过HttpProxyBoot启动请求转发服务，会分别调用NettyHttpClient、HttpProxyDnsMappingManager的start();

NettyHttpClient初始化和启动：

初始化HTTP代理服务、DNS解析服务，见《3初始化和启动-10-HTTP和DNS服务.note》；

用于和Http server端进行交互；

调用NettyHttpClient#doStart() 启动服务，用于转发请求给http server、接收http response;

HttpClientHandler用于处理http server的response;

1、HttpProxyHandler#handle()处理客户端发来的HTTP请求，并调用NettyHttpClient#request()转发给Http server端：

```
1 //4.发送请求
2 httpClient.request(new RequestContext(request, new DefaultHttpCallback(message)));
```

2、NettyHttpClient#request()转发请求给http server

```
1 @Override
2 public void request(RequestContext context) throws Exception {
3  URI uri = new URI(context.request.uri());
4  String host = context.host = uri.getHost();
5  int port = uri.getPort() == -1 ? 80 : uri.getPort();
6  //1.设置请求头
7  context.request.headers().set(HOST, host);//映射后的host
```

```
 context.request.headers().set(CONNECTION, KEEP_ALIVE);//保存长链接

 //2.添加请求超时检测队列
 timer.newTimeout(context, context.readTimeout, TimeUnit.MILLISECONDS);

 //3.先尝试从连接池里取可用链接，去取不到就创建新链接。
 Channel channel = pool.tryAcquire(host);
 if (channel == null) {
 final long startCreate = System.currentTimeMillis();
 LOGGER.debug("create new channel, host={}", host);
 ChannelFuture f = b.connect(host, port);
 f.addListener((ChannelFutureListener) future -> {
 LOGGER.debug("create new channel cost={}", (System.currentTimeMillis() - startCreate));
 if (future.isSuccess()) {//3.1.把请求写到http server
 writeRequest(future.channel(), context);
 } else {//3.2如果链接创建失败，直接返回客户端网关超时
 context.tryDone();
 context.onFailure(504, "Gateway Timeout");
 LOGGER.warn("create new channel failure, request={}", context);
 }
 });
 } else {
 //3.1.把请求写到http server
 writeRequest(channel, context);
 }
}

private void writeRequest(Channel channel, RequestContext context) {
 channel.attr(requestKey).set(context);
 pool.attachHost(context.host, channel);
 channel.writeAndFlush(context.request).addListener((ChannelFutureListener) future -> {
 if (!future.isSuccess()) {
 RequestContext info = future.channel().attr(requestKey).getAndSet(null);
 info.tryDone();
 info.onFailure(503, "Service Unavailable");
 LOGGER.debug("request failure request={}", info);
 pool.tryRelease(future.channel());
 }
 });
}
```

3、接收Http server端的response，并处理

```
@ChannelHandler.Sharable
class HttpClientHandler extends ChannelInboundHandlerAdapter {
 private static final Logger LOGGER = LoggerFactory.getLogger(NettyHttpClient.class);
 private final NettyHttpClient client;

 public HttpClientHandler(NettyHttpClient client) {
 this.client = client;
 }
 @Override
 public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {
 RequestContext context = ctx.channel().attr(client.requestKey).getAndSet(null);
 try {
 if (context != null && context.tryDone()) {
 //调用DefaultHttpCallback#onException()方法，响应502错误消息给客户端
```

```
context.onException(cause);
}
} finally {
//回收连接到连接池，多余的丢弃
client.pool.tryRelease(ctx.channel());
}
LOGGER.error("http client caught an ex, info={}", context, cause);
}
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
RequestContext context = ctx.channel().attr(client.requestKey).getAndSet(null);
try {
if (context != null && context.tryDone()) {
LOGGER.debug("receive server response, request={}, response={}", context, msg);
HttpResponse response = (HttpResponse) msg;
//判断返回状态码是否是3xx重定向
if (isRedirect(response)) {
//重定向次数 <5
if (context.onRedirect(response)) {
//拿到response headers中的location值(重定向的URL)
String location = getRedirectLocation(context.request, response);
if (location != null && location.length() > 0) {
context.cancelled.set(false);
context.request.setUri(location);
//重新发送请求
client.request(context);
return;
}
}
}
//正常响应，调用DefaultHttpCallback#onResponse()方法，将内容转发给客户端
context.onResponse(response);
} else {
LOGGER.warn("receive server response but timeout, request={}, response={}", context, msg);
}
} finally {
//回收连接到连接池，多余的丢弃
client.pool.tryRelease(ctx.channel());
//释放对象
ReferenceCountUtil.release(msg);
}
}
private boolean isRedirect(HttpResponse response) {
HttpResponseStatus status = response.status();
switch (status.code()) {
case 300:
case 301:
case 302:
case 303:
case 305:
case 307:
return true;
default:
return false;
}
}
```

```
71  private String getRedirectLocation(HttpRequest request, HttpResponse response) throws Exception {
72  String hdr = URLDecoder.decode(response.headers().get(HttpHeaderNames.LOCATION), "UTF-8");
73  if (hdr != null) {
74  if (hdr.toLowerCase().startsWith("http://") || hdr.toLowerCase().startsWith("https://")) {
75  return hdr;
76  } else {
77  URL orig = new URL(request.uri());
78  String pth = orig.getPath() == null ? "/" : URLDecoder.decode(orig.getPath(), "UTF-8");
79  if (hdr.startsWith("/")) {
80  pth = hdr;
81  } else if (pth.endsWith("/")) {
82  pth += hdr;
83  } else {
84  pth += "/" + hdr;
85  }
86  StringBuilder sb = new StringBuilder(orig.getProtocol());
87  sb.append("://").append(orig.getHost());
88  if (orig.getPort() > 0) {
89  sb.append(":").append(orig.getPort());
90  }
91  if (pth.charAt(0) != '/') {
92  sb.append('/');
93  }
94  sb.append(pth);
95  return sb.toString();
96  }
97  }
98  return null;
99  }
100  @SuppressWarnings("unused")
101  private HttpRequest copy(String uri, HttpRequest request) {
102  HttpRequest nue = request;
103  if (request instanceof DefaultFullHttpRequest) {
104  DefaultFullHttpRequest dfr = (DefaultFullHttpRequest) request;
105  FullHttpRequest rq;
106  try {
107  rq = dfr.copy();
108  } catch (IllegalReferenceCountException e) { // Empty byteBuf
109  rq = dfr;
110  }
111  rq.setUri(uri);
112  } else {
113  DefaultHttpRequest dfr = new DefaultHttpRequest(request.protocolVersion(), request.method(), uri);
114  dfr.headers().set(request.headers());
115  nue = dfr;
116  }
117  return nue;
118  }
119 }
```