网络框架解析

CONTENTS

- 为什么用OKHTTP
- 如何发起HTTP请求
- HTTP请求的背后
- 如何优雅地发起HTTP请求

OkHttp

Retrofit

- **OKHTTP的优越性**
- 全新的API, 简单直观
- 能很好地解决我们遇到的问题
- 最流行的Android网络请求库

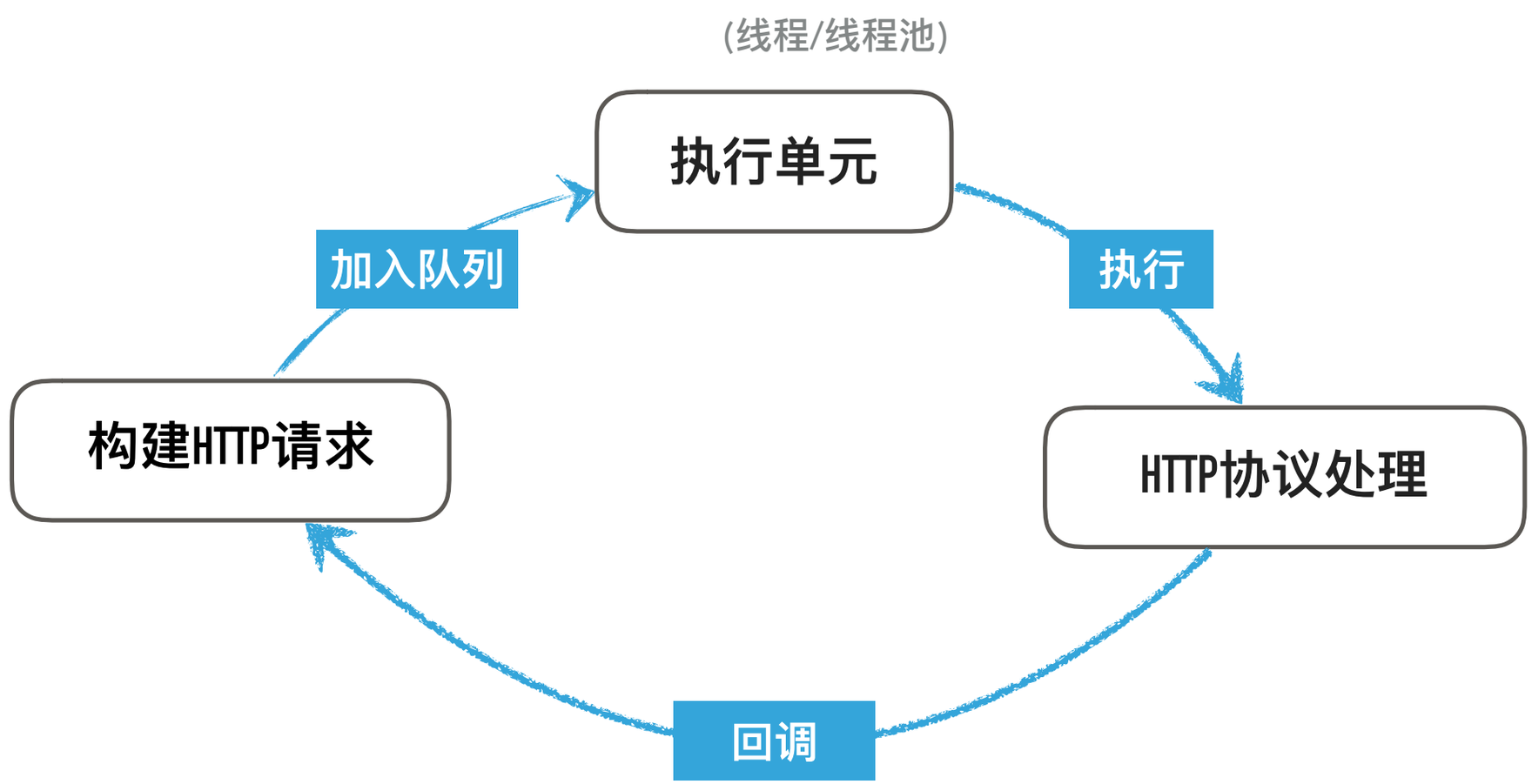
有多流行?

Android network libraries

Sort by: Installs, Apps Marketshare in: Overall, New, Top 500

OkHttp	okHttp	OkHttp is an efficient HTTP client providing features such as SPDY, connection pooling, transparent...	18.47% of apps 12.20% of installs
Retrofit	Retrofit	A type-safe REST client for Android and Java	17.07% of apps 7.17% of installs
Apache Commons	Apache HttpMime API	This module provides support for MIME multipart encoded entities.	8.43% of apps 4.76% of installs
Apache Commons	Apache Http Auth	The API for client-side HTTP authentication against a server.	5.42% of apps 2.72% of installs
	Android Asynchronous Http Client	An asynchronous callback-based Http client for Android built on top of Apache's HttpClient librarie...	6.83% of apps 1.93% of installs

HTTP请求流程



OKHTTP

HttpURLConnection ->

```
new Thread(new Runnable() {
    @Override
    public void run() {
        HttpURLConnection connection = null;
        String response = null;
        try {
            URL url = new URL("http://open.play.cn/api/v2/egame/host.json");
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.setConnectTimeout(10000);
            connection.setReadTimeout(10000);
            connection.connect();
            InputStream inputStream = connection.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
            StringBuilder result = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                result.append(line);
            }
            response = result.toString();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (connection != null) {
                connection.disconnect();
            }
        }
    }
}).start();
```

HttpClient ->

```
new Thread(new Runnable() {
    @Override
    public void run() {
        String response = null;
        HttpParams params = new BasicHttpParams();
        HttpConnectionParams.setConnectionTimeout(params, 10000);
        HttpConnectionParams.setSoTimeout(params, 10000);
        HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
        HttpProtocolParams.setContentCharset(params, HTTP.UTF_8);
        HttpClient client = new DefaultHttpClient(params);
        try {
            HttpGet request = new HttpGet("http://open.play.cn/api/v2/egame/host.json");
            HttpResponse response = client.execute(request);
            InputStream inputStream = response.getEntity();
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
            StringBuilder result = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                result.append(line);
            }
            response = result.toString();
        } catch (Exception e) {
        }
    }
}).start();
```

OkHttp ->

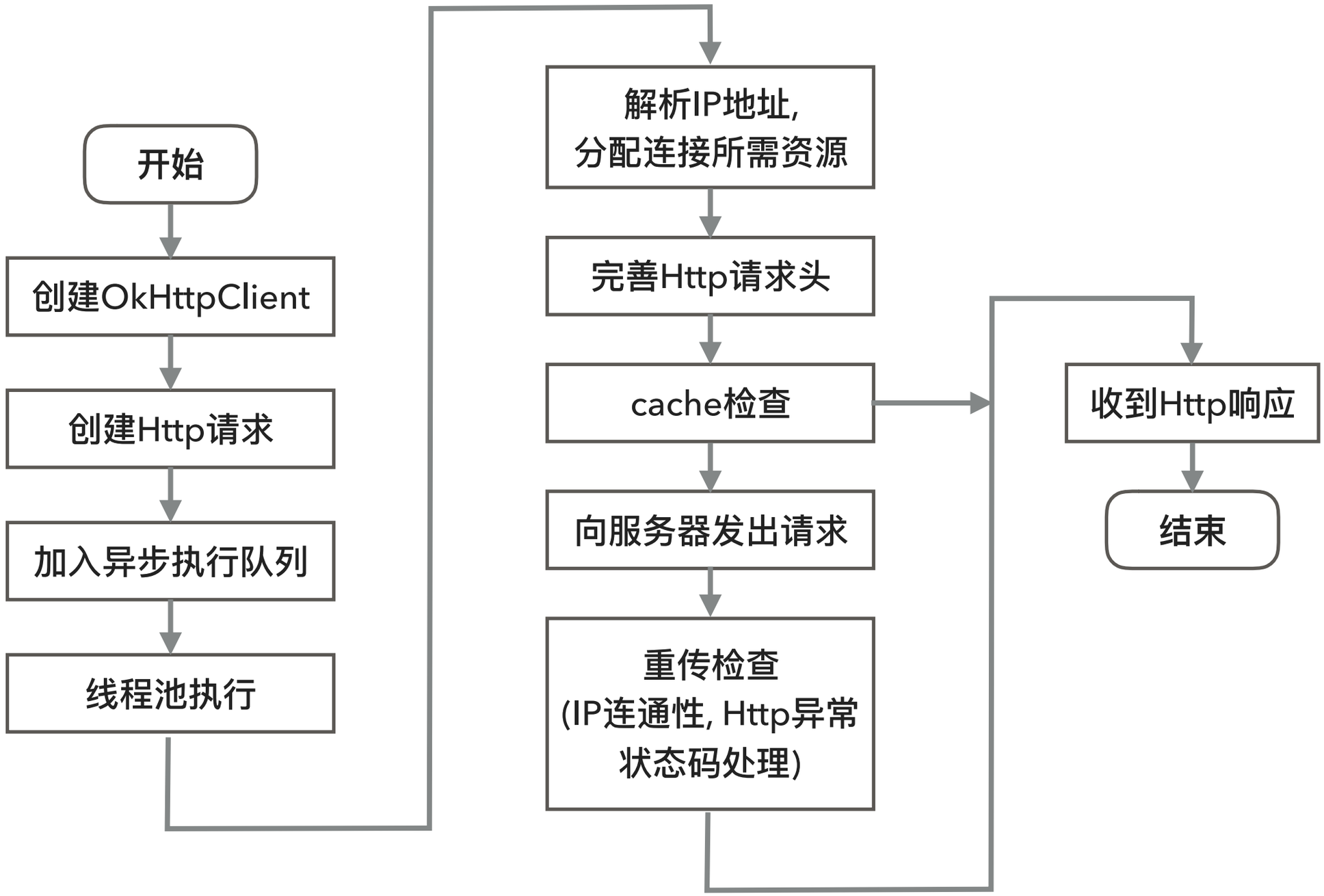
```
OkHttpClient client = new OkHttpClient.Builder().build();
Request request = new Request.Builder().url("http://open.play.cn/api/v2/egame/
host.json").build();
Call call = client.newCall(request);
call.enqueue(new Callback() {
    @Override
    public void onResponse(Call call, Response response) throws IOException {
        String response = response.body().string();
    }

    @Override
    public void onFailure(Call call, IOException e) {
    }
});
```

- 相比于HttpClient的“谜之API”, OkHttp显得尤为清爽, 更符合人类的思维.
- 当然, OkHttp的性能也是十分优越的. 从此发起Http请求变得惬意.

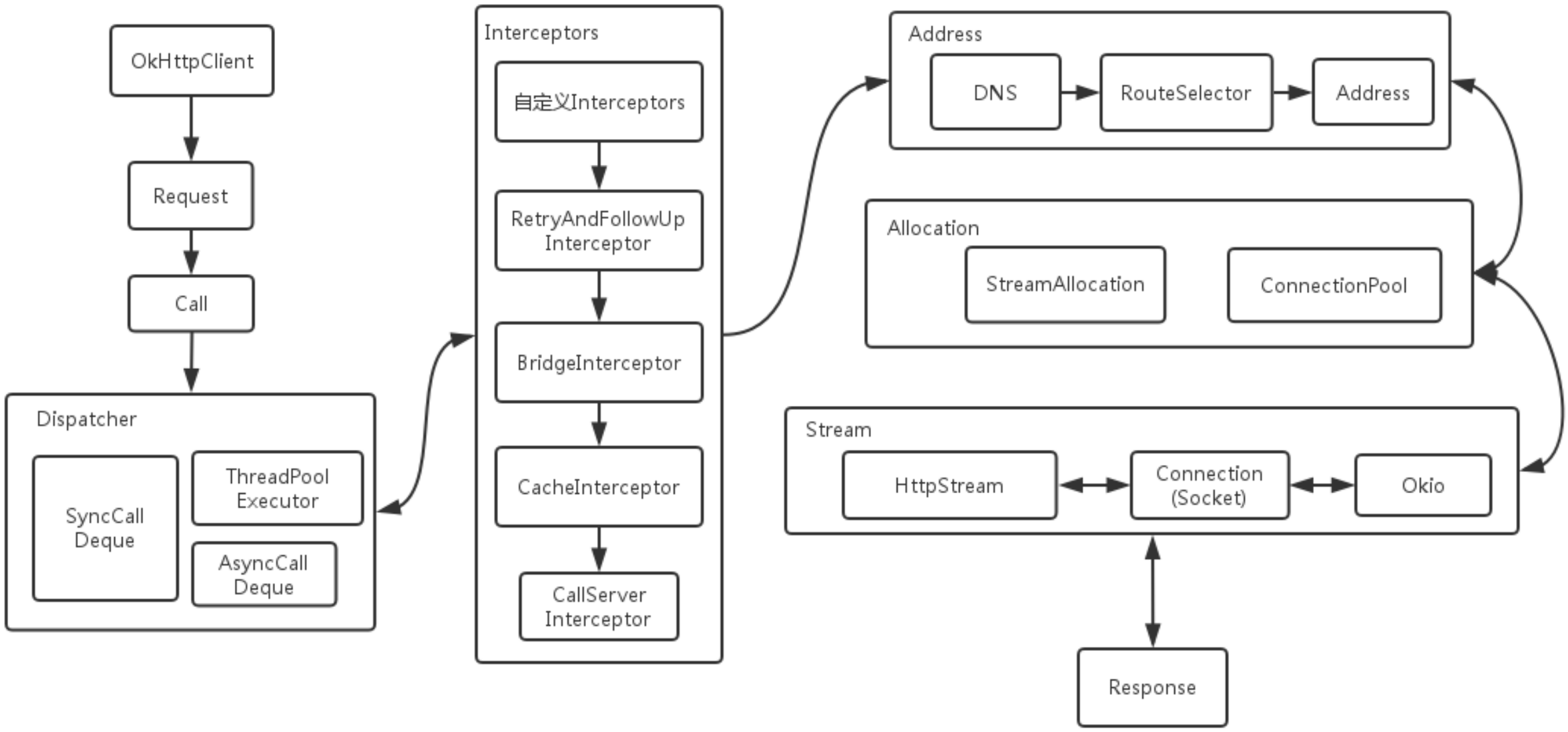
现有HTTPCLIENT网络库遇到的问题

- 线程优化
  - 每个请求都会开启新的线程 -> 线程复用 -> 线程池
- 连接数优化
  - 每个请求都会发起一个连接 -> 连接复用 -> 连接池
- 重传机制优化
  - 为了切换IP手动处理异常响应 -> Http协议栈自动处理
- API复杂, 不够抽象
- Cache机制
- DNS域名劫持
- Https证书认证
- ...

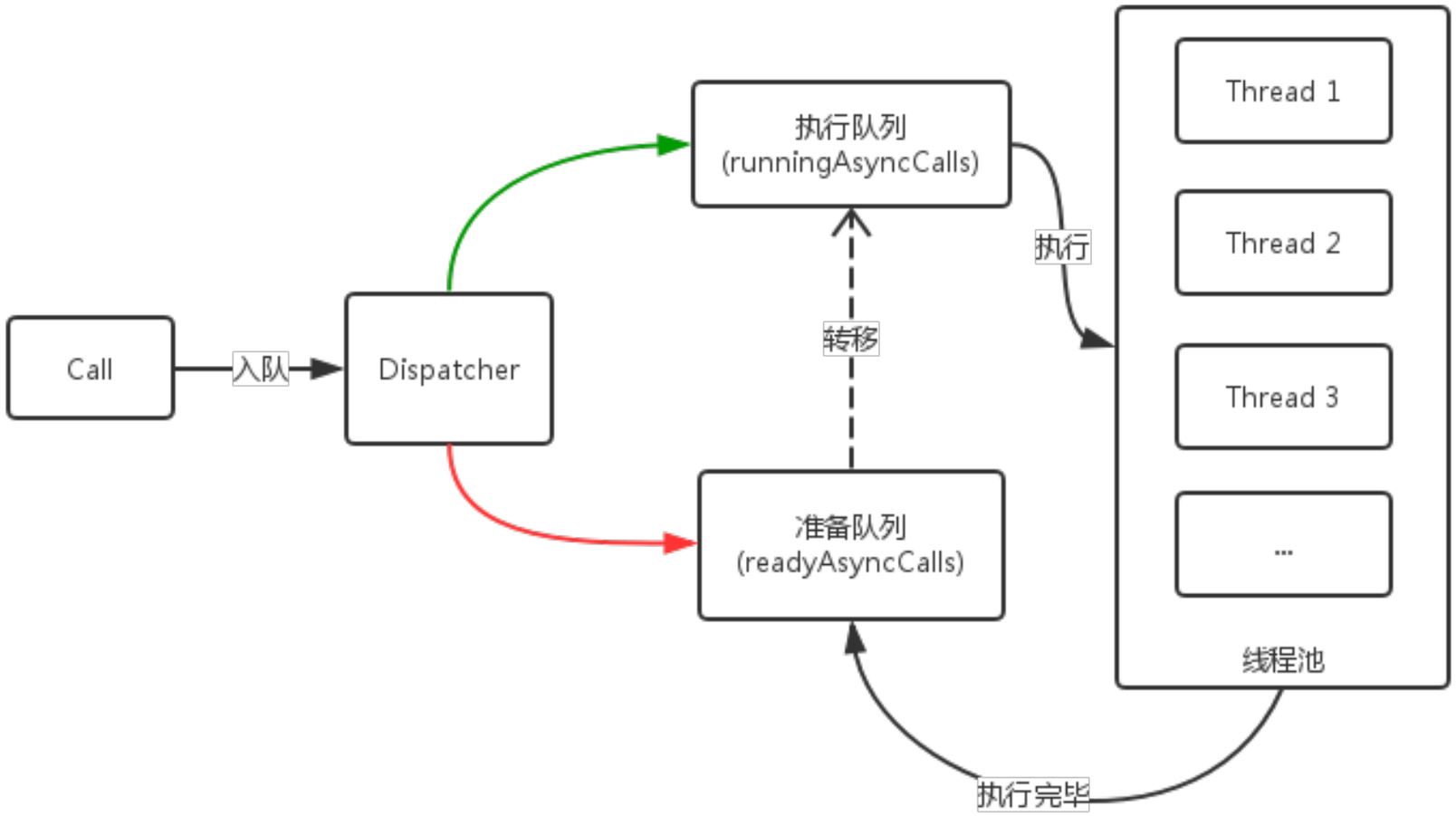
OKHTTP处理流程 (HTTP, 无代理)



OKHTTP模块图


OkHttpClient
Request
Call
Dispatcher
SyncCall Deque
ThreadPool Executor
AsyncCall Deque
Interceptors
自定义Interceptors
RetryAndFollowUp Interceptor
BridgeInterceptor
CacheInterceptor
CallServer Interceptor
Address
DNS
RouteSelector
Address
Allocation
StreamAllocation
ConnectionPool
Stream
HttpStream
Connection (Socket)
Okio
Response

CASE1: 并发请求

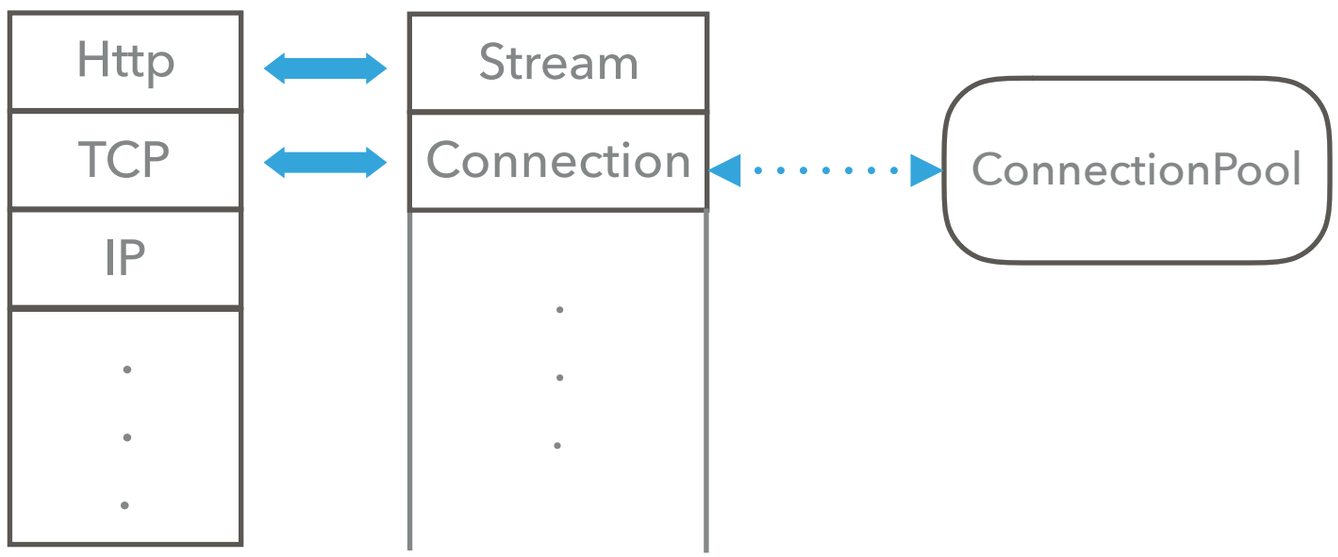



```
new ThreadPoolExecutor(
    0, Integer.MAX_VALUE, 60,
    TimeUnit.SECONDS,
    new SynchronousQueue<Runnable>());
}
```

它不保留任何最小线程数，随时创建更多的线程数，当线程空闲时只能活60秒.

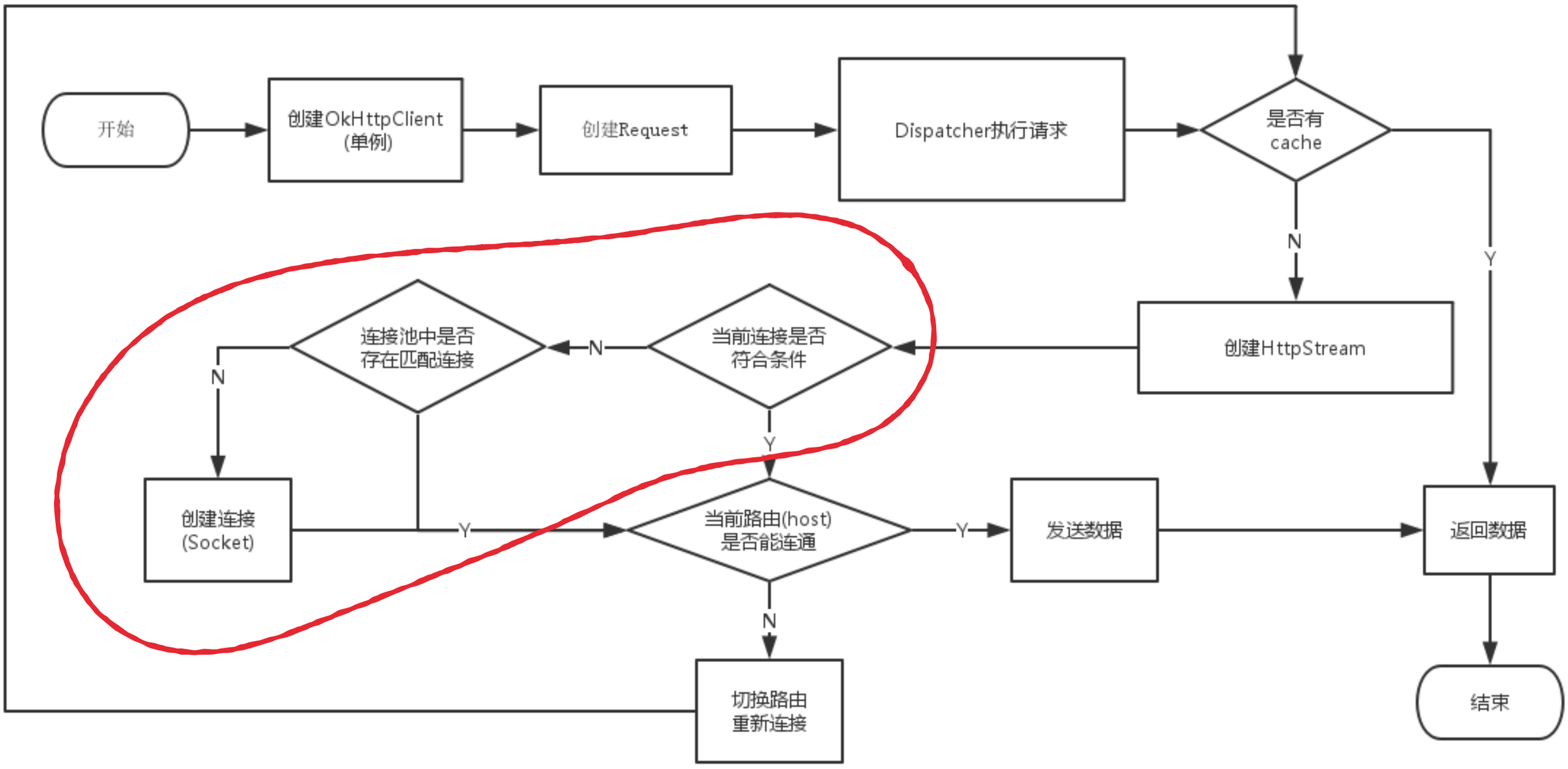
策略:
最大并发请求数为64 (可配)
每个Host最大请求数为5 (可配)
当超过上述限制时, 就加入至准备队列

CASE2: HTTP长连接

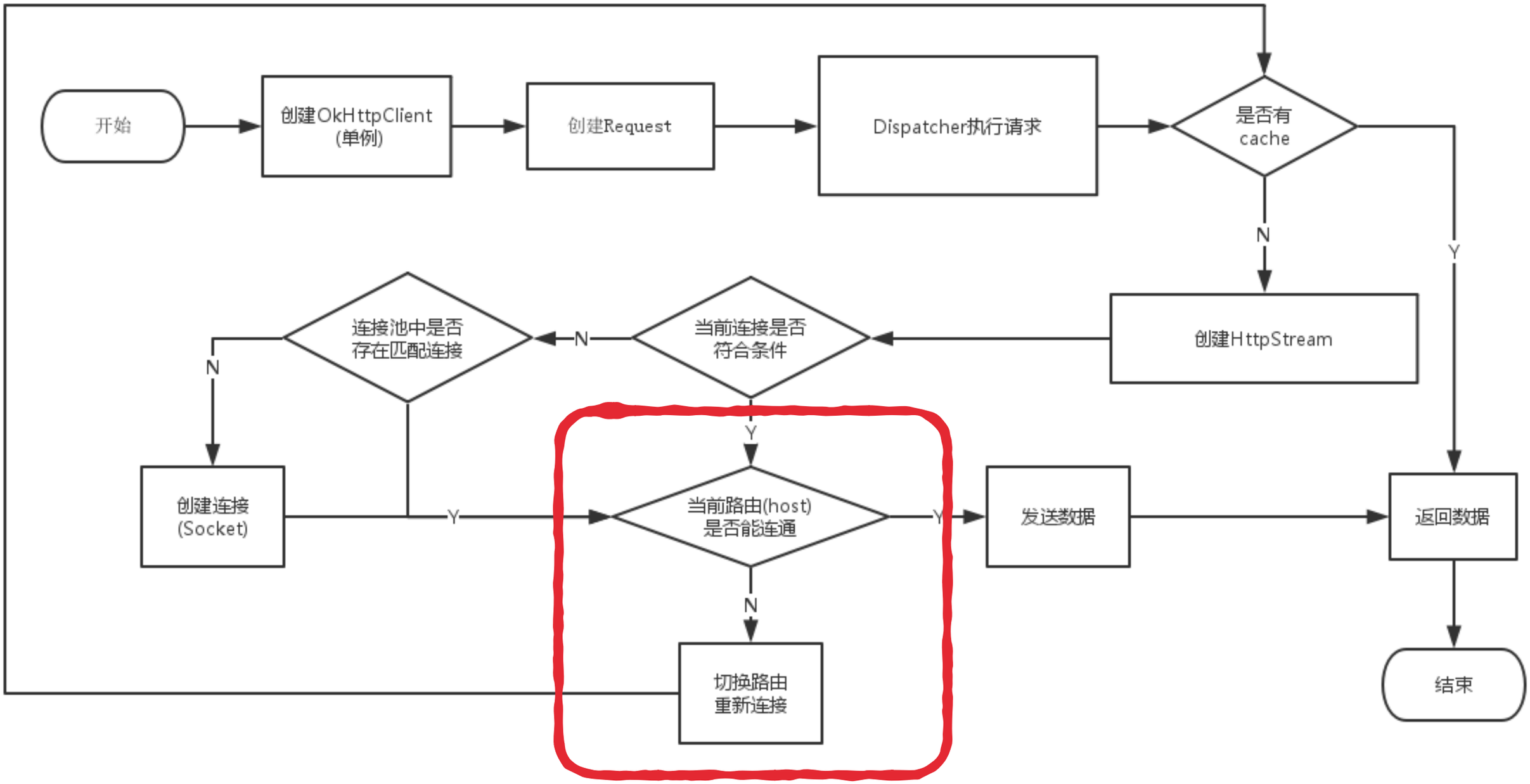


Http1.1 一个连接对应一个流
Http2 一个连接可以有多个流

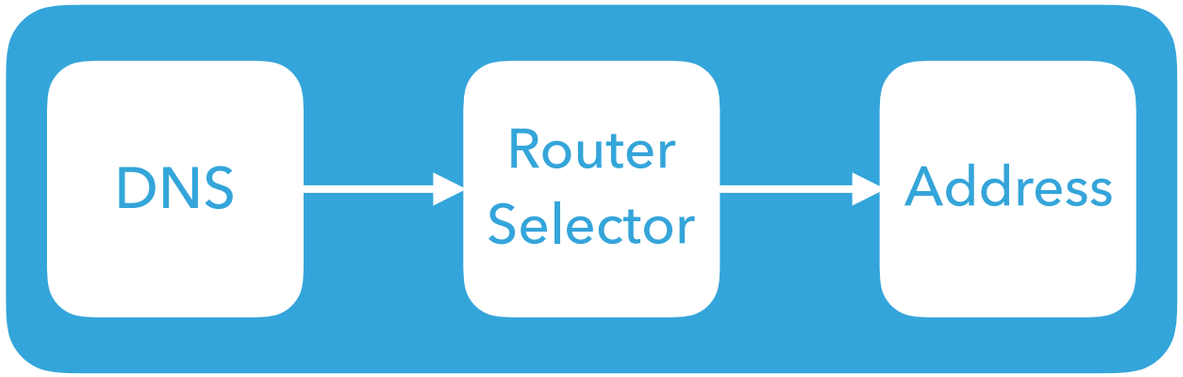
CASE2: HTTP长连接



CASE3: 路由


开始
创建OkHttpClient
(单例)
创建Request
Dispatcher执行请求
是否有
cache
N
Y
创建HttpStream
当前连接是否
符合条件
N
连接池中是否
存在匹配连接
N
Y
创建连接
(Socket)
Y
当前路由(host)
是否能连通
Y
N
切换路由
重新连接
发送数据
返回数据
结束

CASE3: 路由 (DNS)



1. 重传 -> 重设IP
2. Https证书

```
public class OkHttpDns implements Dns {
    private List<String> mAddresses;
    private String mUrl;
    private OkHttpDns(List<String> addr, String url) {
        mAddresses = addr;
        mUrl = url;
    }
    @Override
    public List<InetAddress> lookup(String hostname) throws UnknownHostException {
        List<InetAddress> inetAddresses = new ArrayList<InetAddress>();
        // 如果是预先配置的host请求,则使用自定义的host列表; 否则使用系统dns解析
        // 这是为了防止手机设置代理而导致的问题
        if (mUrl != null && mUrl.contains(hostname)
                && mAddresses != null && !mAddresses.isEmpty()) {
            for (String address : mAddresses) {
                InetAddress inetAddress = InetAddress.getByName(address);
                inetAddresses.add(inetAddress);
            }
            return inetAddresses;
        }
        inetAddresses = Dns.SYSTEM.lookup(hostname);
        return inetAddresses;
    }
}
```

RETROFIT是什么

- 类型安全的Restful Http client
- OkHttp的封装
- 强大的解耦与扩展能力

获取IP地址列表

http://open.play.cn/api/v2/egame/host.json
{"code":0,"text":"success","ext":{"host_url":["http://202.102.39.23/","http://180.96.49.15/","http://180.96.49.16/"],"app_key":"","cdn_url":[]}}

http://open.play.cn/api/v2/egame/host.json

```
{
  code: 0,
  text: "success",
  ext: {
    host_url: [
      "http://202.102.39.23/",
      "http://180.96.49.15/",
      "http://180.96.49.16/"
    ],
    app_key: "",
    cdn_url: [ ]
  }
}
```

获取IP地址列表

http://open.play.cn/api/v2/egame/host.json
{"code":0,"text":"success","ext":{"host_url":["http://202.102.39.23/","http://180.96.49.15/","http://180.96.49.16/"],"app_key":"","cdn_url":[]}}

```
// 1. 创建model
public class HostModel {
    public int code;
    public String text;
    public ExtBean ext;
    public static class ExtBean {
        public List<String> host_url;
        }
    }
}
```

```
// 2. 定义Restful api
public interface HostApi {
    String BASE_URL = “http://open.play.cn";
    @GET ("api/v2/egame/host.json")
    Call<HostModel> hostList();
}
```

```
// 3. 发起, 处理请求
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("http://open.play.cn")
        .addConverterFactory(GsonConverterFactory.create())
        .build();
HostApi hostApi = retrofit.create(HostApi.class);
Call<HostModel> call = hostApi.hostList();
call.enqueue(new Callback<HostModel>() {
    @Override
    public void onResponse(Call<HostModel> call, Response<HostModel> response) {
        HostModel model = response.body();
        List<String> urls = model.ext.host_url;
        Log.e("MY_RETRO", urls.toString());
    }

    @Override
    public void onFailure(Call<HostModel> call, Throwable t) {

    }
});
```

RETROFIT：类型安全的Http客户端

```
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("http://open.play.cn")
        .addConverterFactory(GsonConverterFactory.create())
        .build();
HostApi hostApi = retrofit.create(HostApi.class);
Call<HostModel> call = hostApi.hostList();
call.enqueue(new Callback<HostModel>() {
    @Override
    public void onResponse(Call<HostModel> call,
Response<HostModel> response) {
        HostModel model = response.body();
        List<String> urls = model.ext.host_url;
        Log.e("MY_RETRO", urls.toString());
    }

    @Override
    public void onFailure(Call<HostModel> call, Throwable t) {

    }
});
```

OKHTTP

```
OkHttpClient client = new OkHttpClient.Builder().build();
Request request = new Request.Builder().url("http://
open.play.cn/api/v2/egame/host.json").build();
Call call = client.newCall(request);
call.enqueue(new Callback() {
    @Override
    public void onResponse(Call call, Response response)
throws IOException {
        String response = response.body().string();
        ...
    }

    @Override
    public void onFailure(Call call, IOException e) {
    }
});
```

OKHTTP网络库的不足

- 需要处理线程切换
- 手动解析JSON文本

OKHTTP网络库的不足

- 需要处理线程切换
  - ✦ CallAdapter
- 手动解析JSON文本
  - ✦ Converter

CALLADAPTER

```
// 将Call适配成T
public interface CallAdapter<T> {
  <R> T adapt(Call<R> call);
  abstract class Factory {...}
}
```

```
public class SimpleCallAdapterFactory extends
                                CallAdapter.Factory{
   @Override
   public CallAdapter<Call<?>> get(...) {
      return new CallAdapter<Call<?>>() {
         @Override
         public <R> Call<?> adapt(Call<R> call) {
            return new SimpleCall<>(call);
         }
         ...
      };
   }
}
```

```
public static class SimpleCall<T> implements Call<T> {
   final Call<T> delegate;
   final Handler handler = new Handler(Looper.getMainLooper());
   public SimpleCall(Call<T> call) {
      delegate = call;
   }

   @Override
   public void enqueue(final Callback<T> callback) {
      delegate.enqueue(new Callback<T>() {
         @Override
         public void onResponse(final Call<T> call, final
Response<T> response) {
            handler.post(() -> {
                  callback.onResponse(call, response);
            });
         }
      });
   }
}
```

CONVERTER

- 默认解析成OkHttp的ResponseBody
- 支持自定义解析器
  - Gson
  - Jackson
  - FastJson
  - Protobuf
  - ...

TODO

- Https
- HttpTunnel
- Http2
- RxJava + Retrofit

结语

- 与时俱进, 不断学习是程序员的基本修养.
- 多看优秀的框架, 收获会很大.
- 看源代码要有抽象技能, 不要陷入细节的泥潭
- 关于网络库, 我们需要的可能只是一个定制化的OkHttpClient, 其余的交给Retrofit就好.

参考文档

- AppBrain
- OkHttp官博
- Retrofit官博
- OKHttp源码解析
- 拆轮子系列：拆 OkHttp
- OkHttp3源码分析[任务队列]
- Retrofit by Future Studio
- Retrofit分析-经典设计模式案例
- Retrofit分析-漂亮的解耦套路