学学课堂 致力于培养卓越的程序员

OkHttp源码分析

请求流程-分发器与拦截器

1.okhttp请求流程

2.高并发请求分发器与线程池

3.责任链模式请求与响应拦截器

Lance

前爱奇艺高级工程师

12月22日 20:00

VIP课程

讲师简介

享学课堂 -Lance老师

某游戏公司主程，前爱奇艺高级工程师。多年移动平台开发经验，涉猎广泛，热爱技术与研究。主要对NDK、架构与性能优化拥有深入的理解及开发经验。授课严谨负责。

- **QQ：2260035406**

OkHttp介绍

https://square.github.io/okhttp/

由Square公司贡献的一个处理网络请求的开源项目，是目前Android使用最广泛的网络框架。从Android4.4开始HttpURLConnection的底层实现采用的是OkHttp。

- 支持HTTP/2并允许对同一主机的所有请求共享一个套接字
- 通过连接池,减少了请求延迟
- 默认通过GZip压缩数据
- 响应缓存，避免了重复请求的网络
- 请求失败自动重试主机的其他ip，自动重定向

目 录

CONTENTS

使用方法

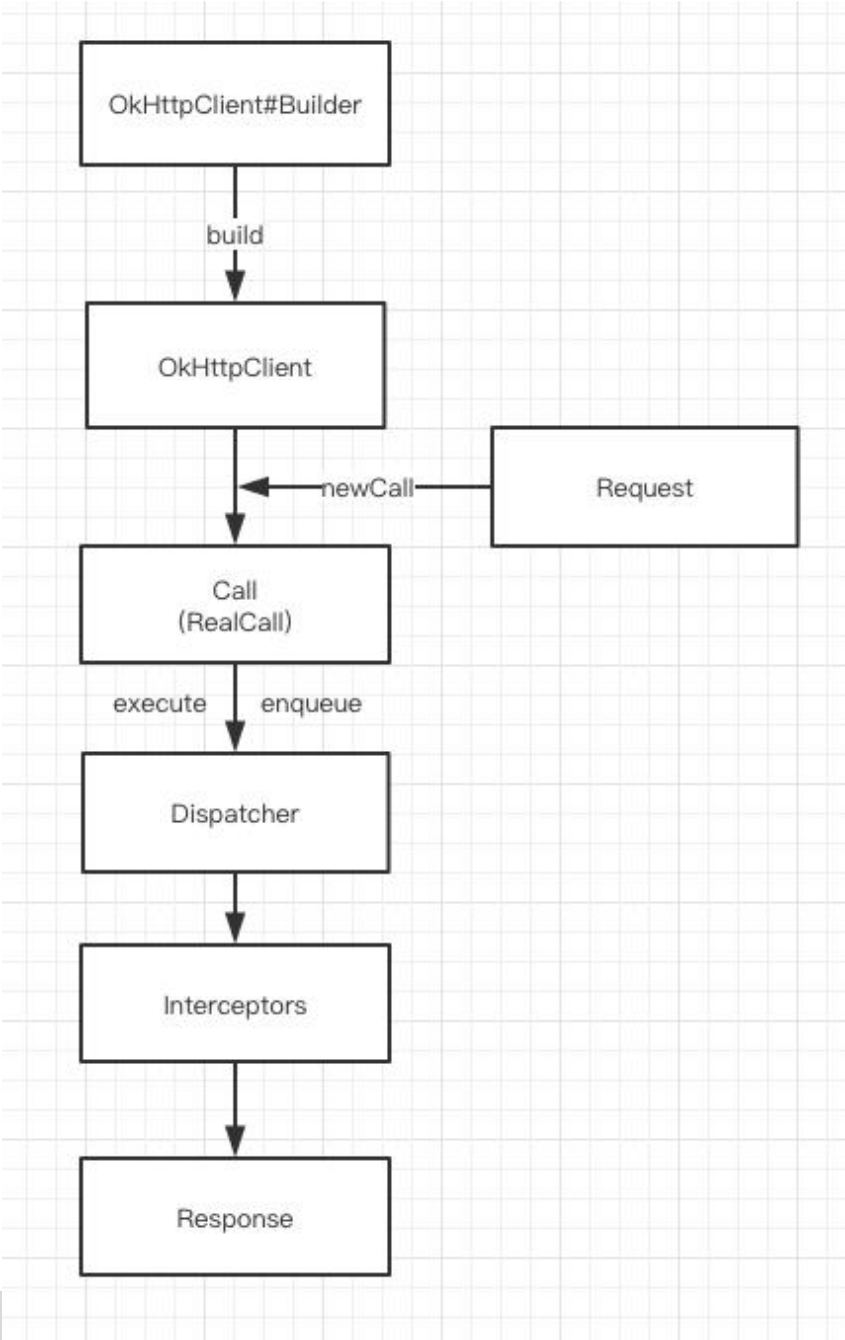
```
OkHttpClient client = new OkHttpClient();

void get(String url) throws IOException {
    Request request = new Request.Builder()
            .url(url)
            .build();
    //执行同步请求
    Call call = client.newCall(request);
    Response response = call.execute();

    //获得响应
    ResponseBody body = response.body();
    System.out.println(body.string());
}

//application/x-www-form-urlencoded"
void post(String url) throws IOException {
    RequestBody requestBody = new FormBody.Builder().add( name: "city",  value: "长沙").add( name: "key",
            value: "13cb58f5884f9749287abbead9c658f2").build();
    Request request = new Request.Builder()
            .url(url)
            .post(requestBody).build();

    //执行同步请求
    Call call = client.newCall(request);
    Response response = call.execute();
    //获得响应
    ResponseBody body = response.body();
    System.out.println(body.string());
}
```

调用流程

OkHttp请求过程中最少只需要接触OkHttpClient、Request、Call、Response，但是框架内部进行大量的逻辑处理。

所有的逻辑大部分集中在拦截器中，但是在进入拦截器之前还需要依靠分发器来调配请求任务。

- 分发器：内部维护队列与线程池，完成请求调配；
- 拦截器：五大默认拦截器完成整个请求过程。



目 录

CONTENTS

使用方法

调用流程

分发器

高并发任务分发

线程池排队

拦截器

责任链模式

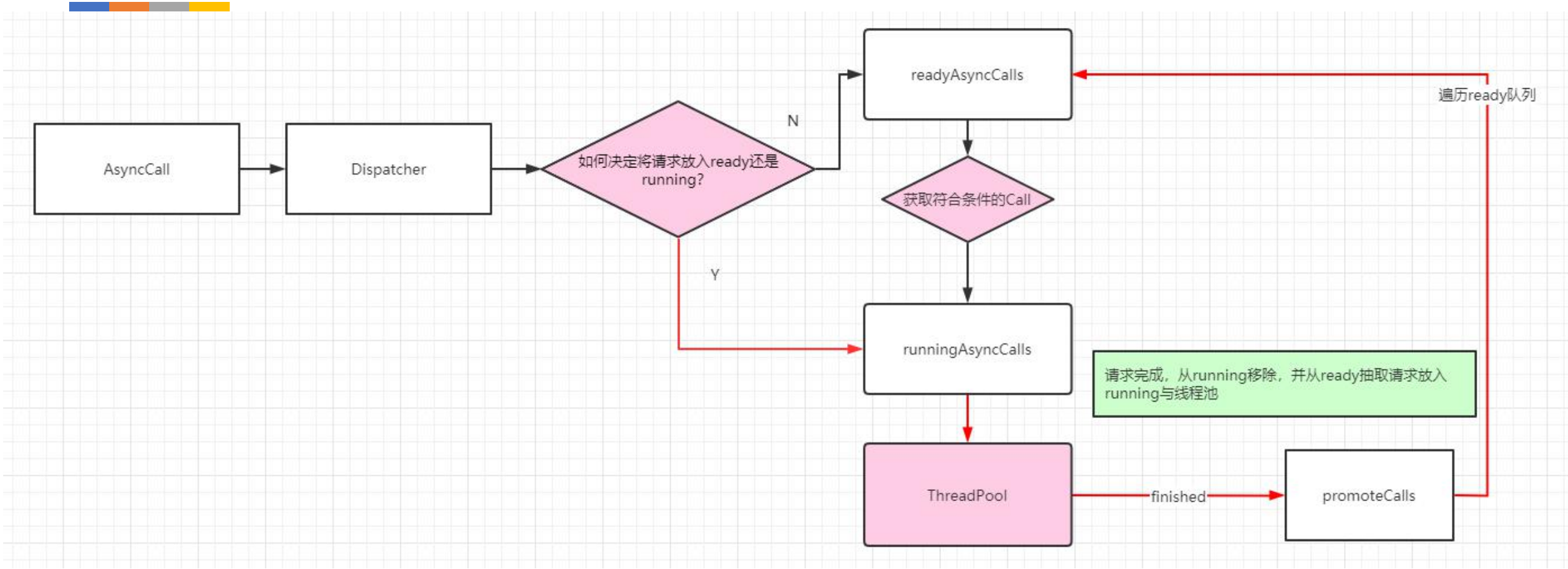
五大拦截器

课程总结

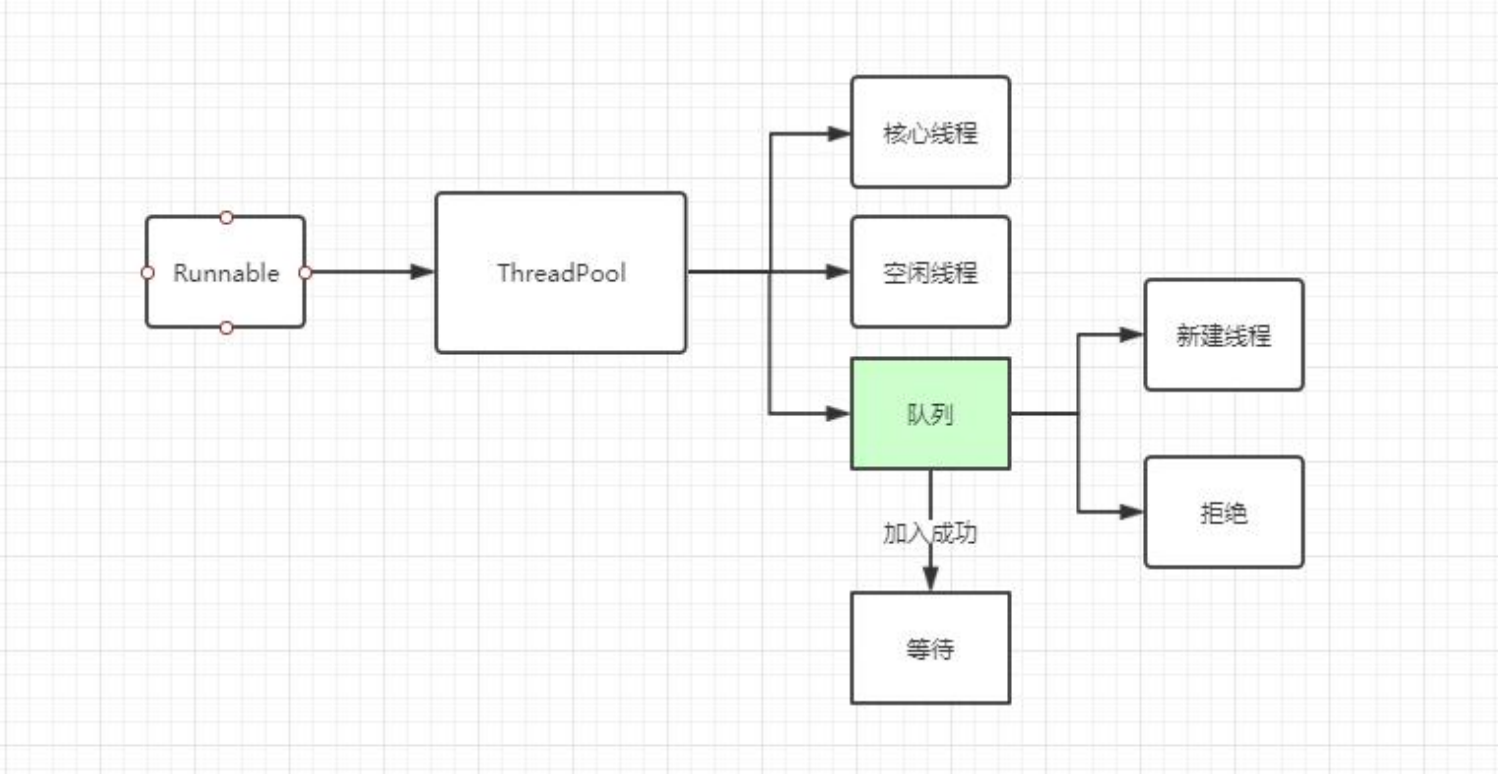
课程技术总结

交流互动

分发器：异步请求工作流程



线程池



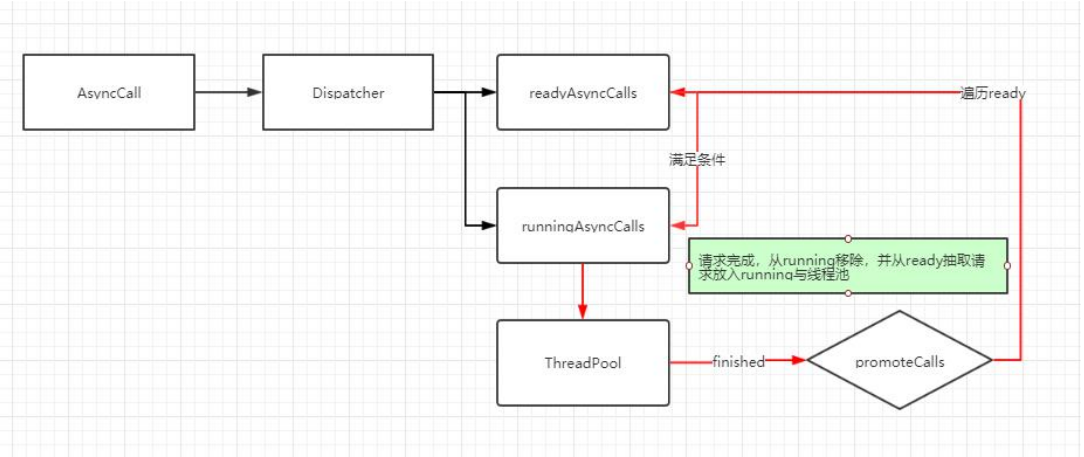
当一个任务通过execute(Runnable)方法添加到线程池时：

- 线程数量小于corePoolSize，新建线程(核心)来处理被添加的任务；
- 线程数量大于等于 corePoolSize，存在空闲线程，使用空闲线程执行新任务；
- 线程数量大于等于 corePoolSize，不存在空闲线程，新任务被添加到等待队列，添加成功则等待空闲线程，添加失败：

  线程数量小于maximumPoolSize，新建线程执行新任务；
  线程数量等于maximumPoolSize，拒绝此任务。

分发器总结

对于同步请求，分发器只记录请求，用于判断IdleRunnable是否需要执行
对于异步请求，向分发器中提交请求：



- Q：如何决定将请求放入ready还是running？
  A：如果当前正在请求数不小于64放入ready；如果小于64，但是已经存在同一域名主机的请求5个放入ready

- Q：从ready移动running的条件是什么？
  A：每个请求执行完成就会从running移除，同时进行第一步相同逻辑的判断，决定是否移动！

- Q：分发器线程池的工作行为？
  A：无等待，最大并发

目 录

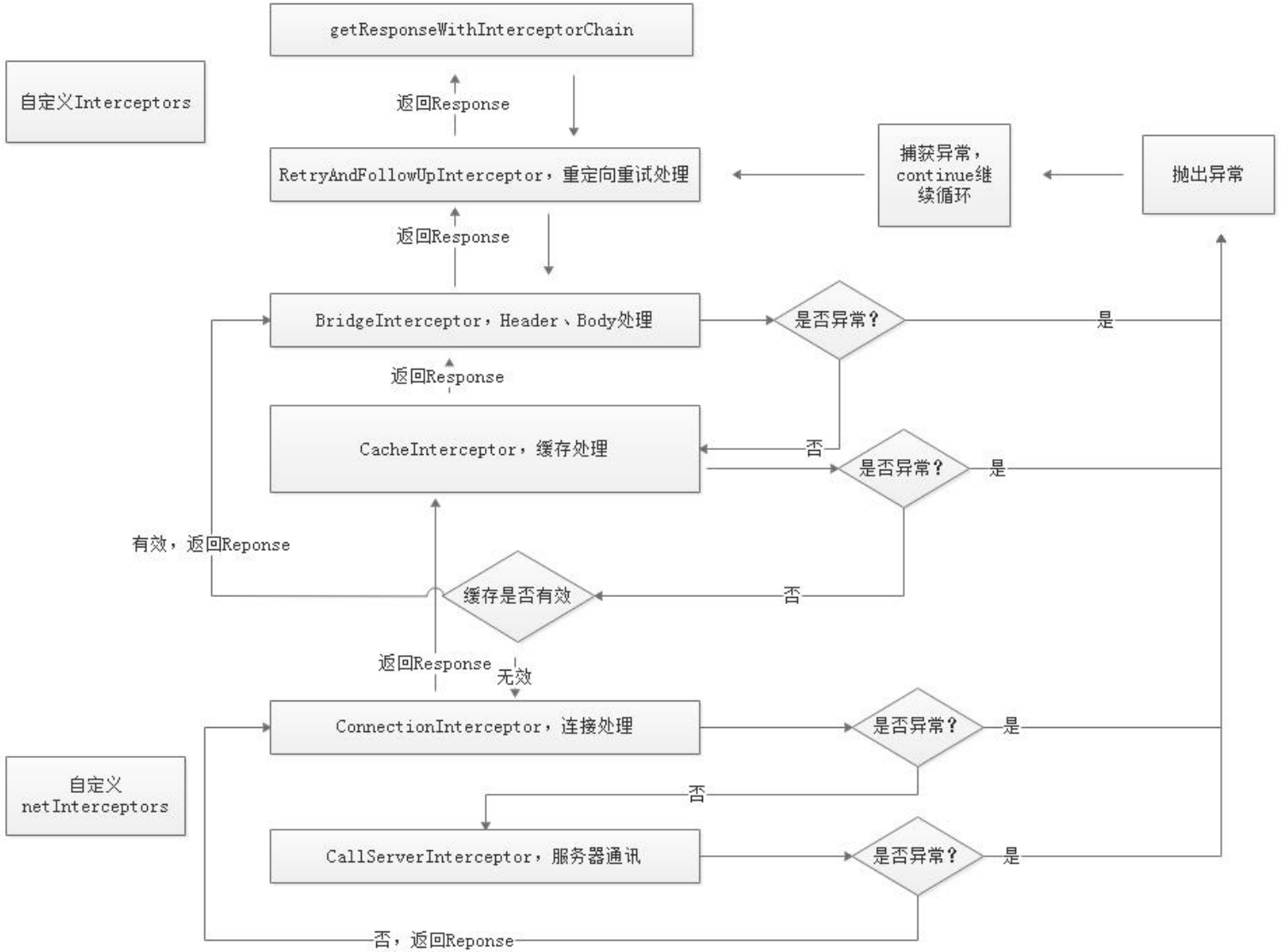
CONTENTS

获得响应

在请求需要执行时，通过 getResponseWithInterceptorChain 获得请求的结果：Response

```
Response getResponseWithInterceptorChain() throws IOException {
    // Build a full stack of interceptors.
    List<Interceptor> interceptors = new ArrayList<>();
    interceptors.addAll(client.interceptors());
    interceptors.add(retryAndFollowUpInterceptor);
    interceptors.add(new BridgeInterceptor(client.cookieJar()));
    interceptors.add(new CacheInterceptor(client.internalCache()));
    interceptors.add(new ConnectInterceptor(client));
    if (!forWebSocket) {
        interceptors.addAll(client.networkInterceptors());
    }
    interceptors.add(new CallServerInterceptor(forWebSocket));

    Interceptor.Chain chain = new RealInterceptorChain(interceptors, null, null, null, 0,
            originalRequest, this, eventListener, client.connectTimeoutMillis(),
            client.readTimeoutMillis(), client.writeTimeoutMillis());

    return chain.proceed(originalRequest);
}
```

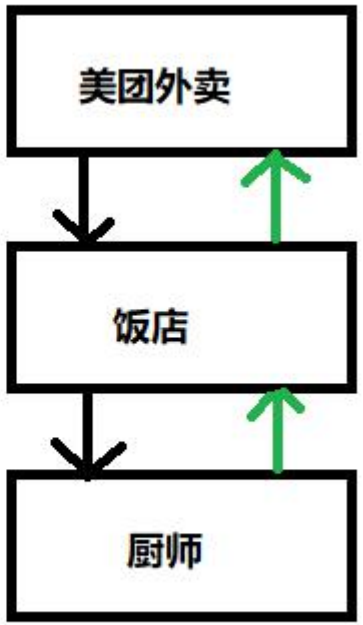
执行流程


getResponseWithInterceptorChain
自定义Interceptors
返回Response
RetryAndFollowUpInterceptor，重定向重试处理
捕获异常，continue继续循环
抛出异常
返回Response
BridgeInterceptor，Header、Body处理
是否异常？
是
返回Response
CacheInterceptor，缓存处理
否
是否异常？
是
有效，返回Reponse
缓存是否有效
否
返回Response
无效
ConnectionInterceptor，连接处理
是否异常？
是
自定义netInterceptors
否
CallServerInterceptor，服务器通讯
是否异常？
是
否，返回Reponse

责任链模式

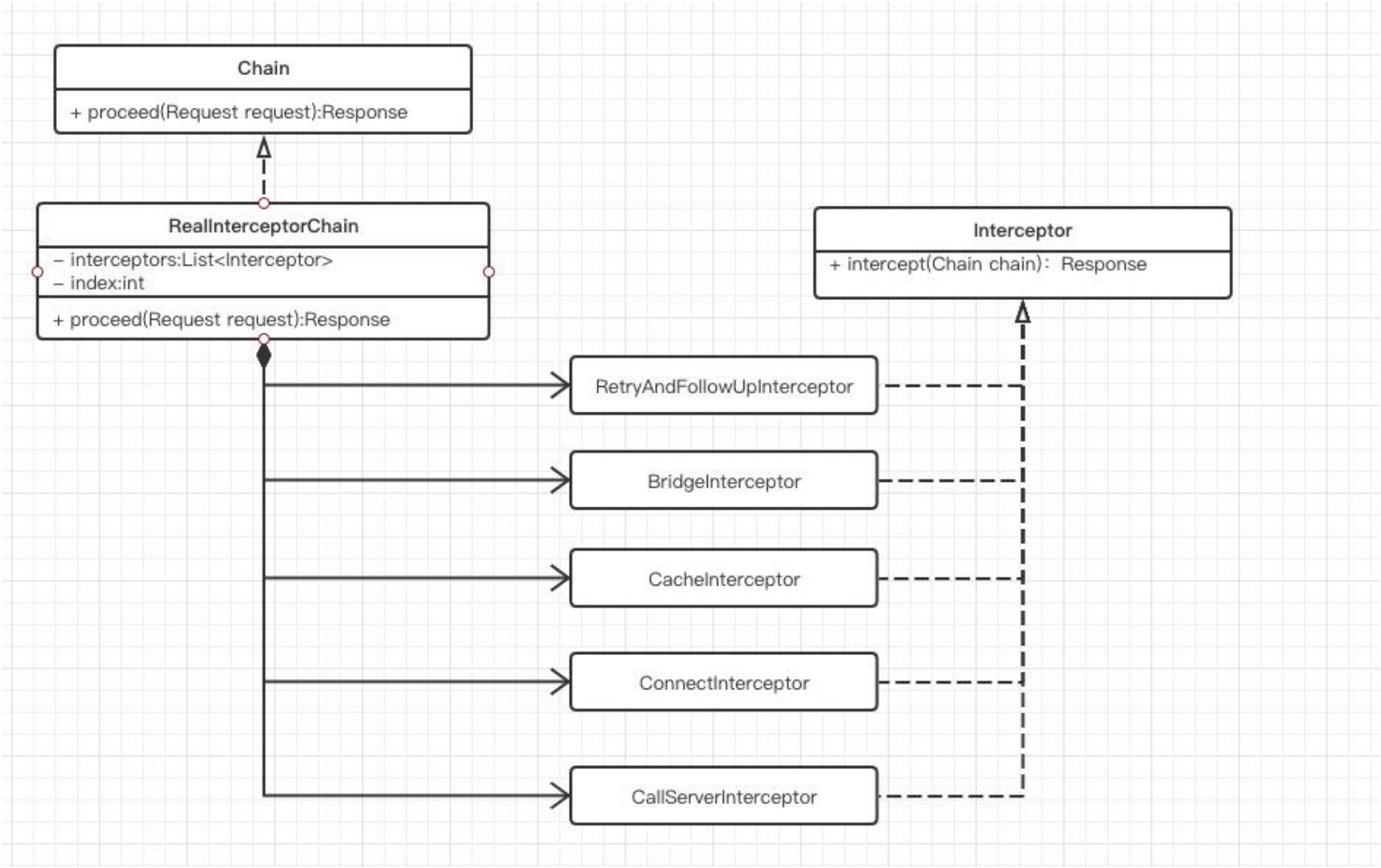
对象行为型模式，为请求创建了一个接收者对象的链，在处理请求的时候执行过滤(各司其职)。

责任链上的处理者负责处理请求，客户只需要将请求发送到责任链即可，无须关心请求的处理细节和请求的传递，所以职责链将请求的发送者和请求的处理者解耦了。



拦截器责任链

请求任务交给Chain即可



拦截器

1、重试拦截器在交出(交给下一个拦截器)之前，负责判断用户是否取消了请求；在获得了结果之后，会根据响应码判断是否需要重定向，如果满足条件那么就会重启执行所有拦截器。

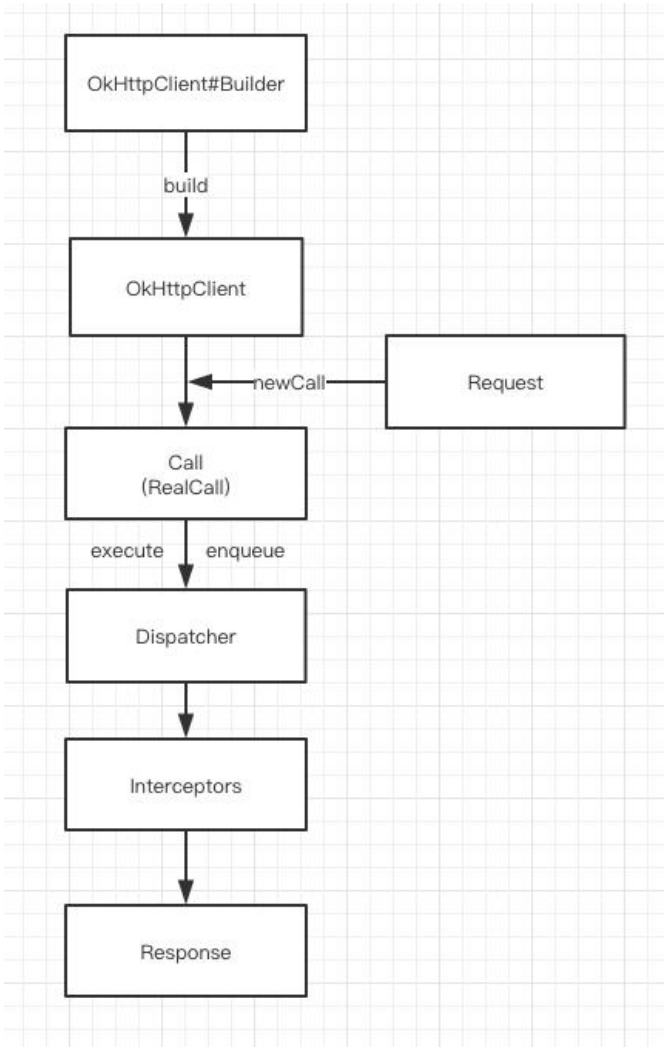
2、桥接拦截器在交出之前，负责将HTTP协议必备的请求头加入其中(如：Host)并添加一些默认的行为(如：GZIP压缩)；在获得了结果后，调用保存cookie接口并解析GZIP数据。

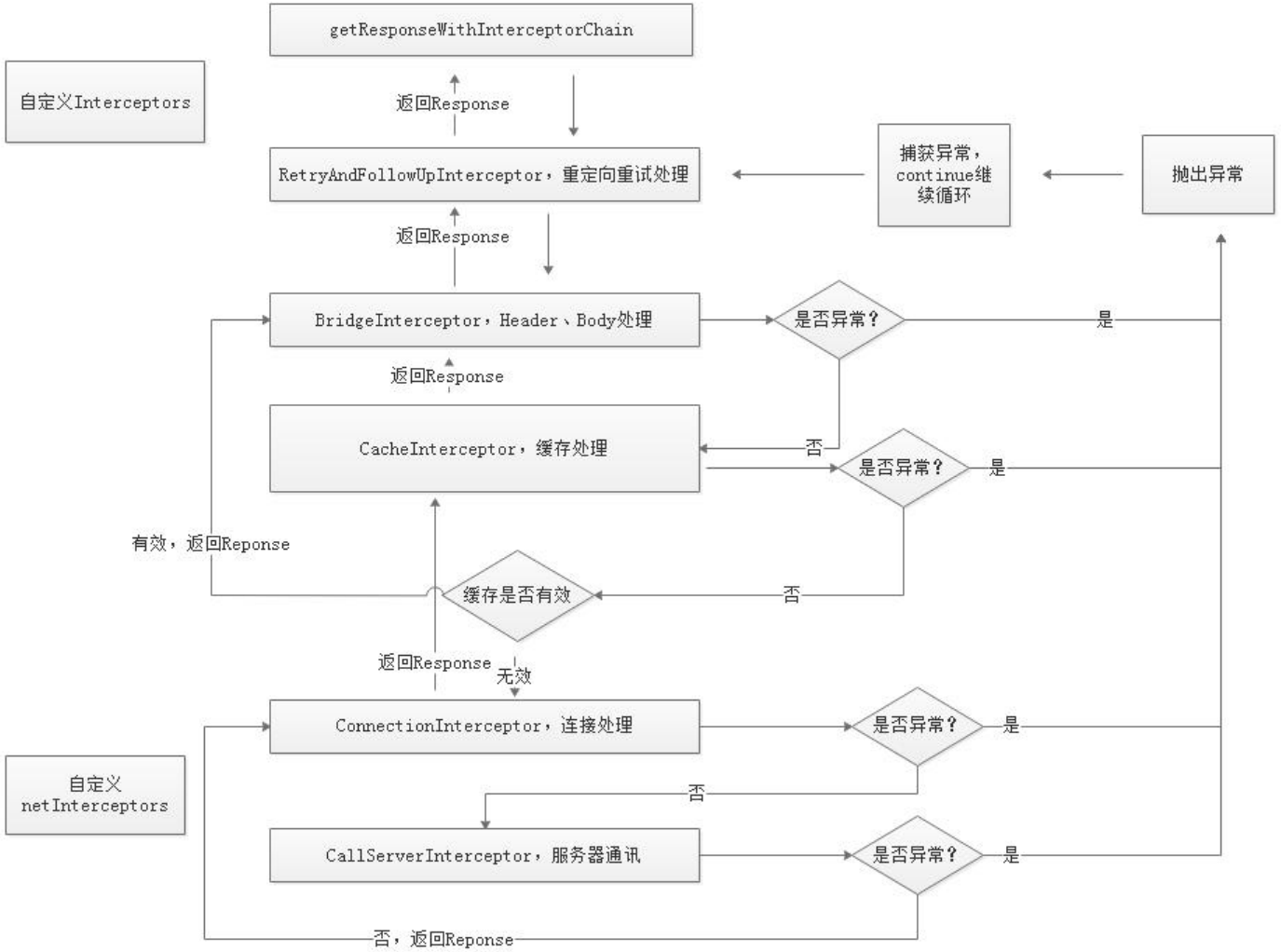
3、缓存拦截器顾名思义，交出之前读取并判断是否使用缓存；获得结果后判断是否缓存。

4、连接拦截器在交出之前，负责找到或者新建一个连接，并获得对应的socket流；在获得结果后不进行额外的处理。

5、请求服务器拦截器进行真正的与服务器的通信，向服务器发送数据，解析读取的响应数据。

总结


OkHttpClient#Builder
build
OkHttpClient
newCall
Request
Call
(RealCall)
execute
enqueue
Dispatcher
Interceptors
Response


getResponseWithInterceptorChain
自定义Interceptors
返回Response
RetryAndFollowUpInterceptor，重定向重试处理
捕获异常，continue继续循环
抛出异常
返回Response
BridgeInterceptor，Header、Body处理
是否异常？
是
返回Response
CacheInterceptor，缓存处理
否
是否异常？
是
有效，返回Reponse
缓存是否有效
否
返回Response
无效
ConnectionInterceptor，连接处理
是否异常？
是
自定义netInterceptors
否
CallServerInterceptor，服务器通讯
是否异常？
是
否，返回Reponse