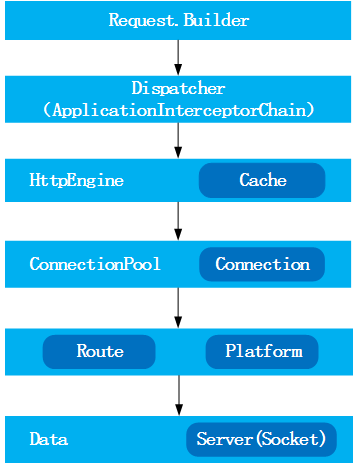
# OKHttp

<http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0326/2643.html>



主要是通过Diapatcher不断从RequestQueue中取出请求（Call），根据是否已缓存调用Cache或 Network这两类数据获取接口之一，从内存缓存或是服务器取得请求的数据。该引擎有同步和异步请求，同步请求通过Call.execute()直接返 回当前的Response，而异步请求会把当前的请求Call.enqueue添加（AsyncCall）到请求队列中，并通过回调（Callback） 的方式来获取最后结果。

先创建OKHttpClient和Request，然后client.newCall(request)

如果同步则执行execute()返回 Response

异步则执行enqueue（callback）

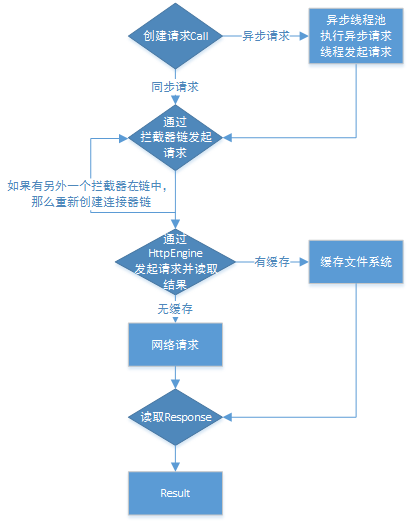
Request request=new Request.Builder().url(...).build();

这是典型的build模式，可以将复杂的对象的创建与表示相分离

核心：Interceptor拦截器

Interceptor本身的文档解释：观察，修改以及可能短路的请求输出和响应请求的回来。通常情况下拦截器用来添加，移除或者转换请求或者回应的头部信息。  
拦截器接口中有intercept(Chain chain)方法，同时返回Response。所谓拦截器更像是AOP设计的一种实现。下面来看一个okhttp源码中的一个引导例子来说明拦截器的作用。

请求流程图



外部网络请求实现：

外部网络请求的入口都是通过Transport接口来完成。该类采用了桥接模式将HttpEngine和HttpConnection来连接起来。 因为HttpEngine只是一个逻辑处理器，同时它也充当了请求配置的提供引擎，而HttpConnection是对底层处理Connection的封 装。

# Retrofit

封装了OKHttp，动态代理模式

Retrofit使用的关键一步就是Retrofit.create函数创建接口动态代理的实例，为接口的每个method创建了一个对应的ServiceMethod,并使用这个ServiceMethod对象创建OkHttpCall,并使用ServiceMethod实例的callAdapter来调用okhttpCall并返回结果

具体是下面三步

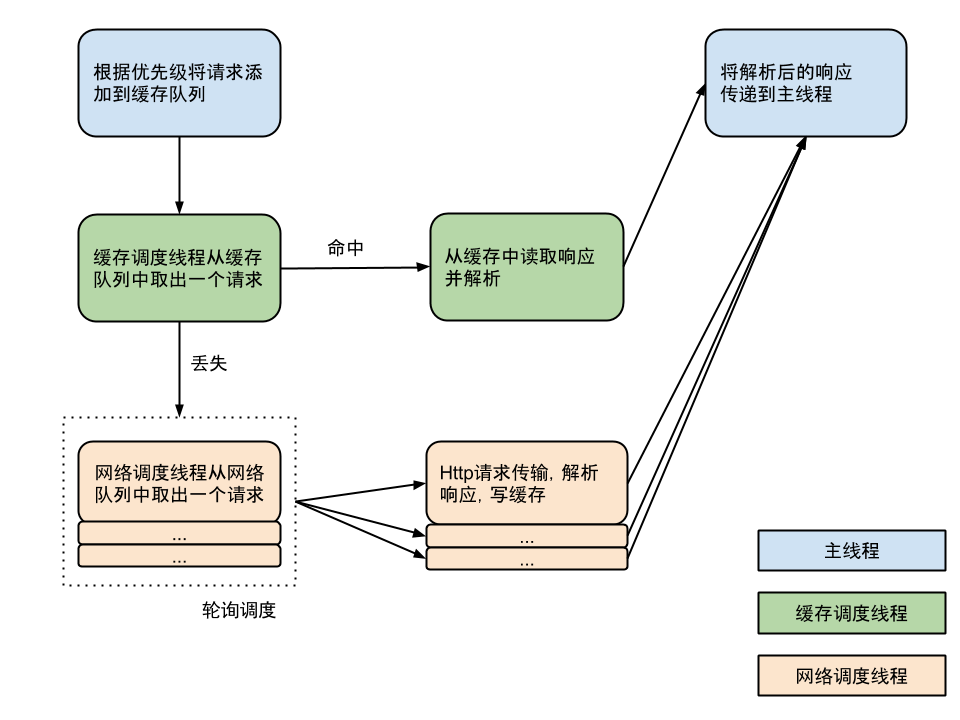
通过上面代码可以看到调用关键的就是三步：

* 1 加载对应method的ServiceMethod实例
* 2 使用ServiceMethod实例和方法调用参数创建OkHttpCall
* 3 调用serviceMethod.callAdapter.adapt(okHttpCall)来产生method所定义的返

# Volley

<http://blog.csdn.net/ht_android/article/details/44960495>

打开两个线程，分别是缓存和网络



1.传入Context时，调用newRequestQueue(context, null)，然后根据DEFAULT\_CACHE\_DIR定义缓存文件名。

2.然后根据系统版本号，sdk大于9时使用HurlStack( HttpUrlConnection )，否则使用HttpClientStack( HttpClient ),然后将Stack传入Network，Network可以调用Stack处理请求，然后将结果转换成NetworkResponse

3.然后将定义的Network和缓存目录传入RequestQueue(Cache cache, Network network)去新建请求队列。

4.最后开启请求队列，执行start()方法，该方法会产生根据线程池大小产生对应数量的网络调度线程NetworkDispatcher和一个缓存调度线程CacheDispatcher(此处是用于异步处理请求的)

5.每次调用只需要将自定义的Request通过add()方法传入，请求将会被添加到mCurrentRequests，该Set用于存储当前正在处理的请求，接着根据请求的shouldCache()方法判断是否需要进行缓存，如果不需要则直接放入网络队列，否则根据Url将请求放入缓存队列。

6.缓存调度器CacheDispatcher会从缓存队列中取出请求，然后根据CacheKey获取缓存，如果获取失败则放入网络队列中去处理，否则进行验证有效期等，最后将处理结果交由ResponseDelivery，他是返回结果分发接口，用于分发返回的结果

7.网络调度器NetworkDispatcher会从网络队列中取出请求，然后处理完成交由ResponseDelivery分发处理结果。

# ButterKnife

通过java注解机制，性能比反射机制好，不能将变量设为private或static，会使其变为反射

扫描注解-》生成ViewBinder类-》调用bind方法加载ViewBinder类

## Java Annotation Processing

是javac中用于编译时扫描和解析Java注解的工具

当你编译你的Android工程时，ButterKnife工程中ButterKnifeProcessor类的process()方法会执行以下操作：

* 开始它会扫描Java代码中所有的ButterKnife注解@Bind、@OnClick、@OnItemClicked等
* 当它发现一个类中含有任何一个注解时，ButterKnifeProcessor会帮你生成一个Java类，名字类似<className>$$ViewBinder，这个新生成的类实现了ViewBinder<T>接口
* 这个ViewBinder类中包含了所有对应的代码，比如@Bind注解对应findViewById(), @OnClick对应了view.setOnClickListener()等等
* 最后当Activity启动ButterKnife.bind(this)执行时，ButterKnife会去加载对应的ViewBinder类调用它们的bind()方法