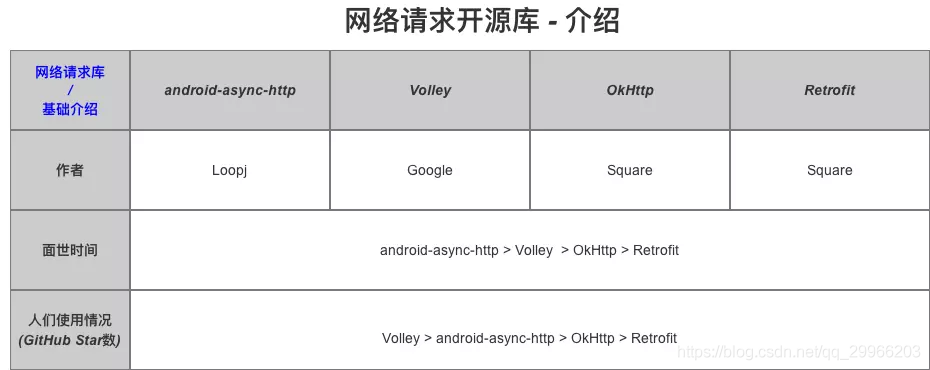
# **Android实现网络请求的主流方法**

# HttpURLConnection和HttpClient：这两种方式都支持HTTPS协议、以流的形式进行上传和下载、配置超时时间、IPv6、以及连接池等功能

**主流网络请求库**

网络请求开源库是一个将 网络请求+异步+数据处理 封装好的类库（网络请求是Android网络请求原生方法HttpClient或HttpURLConnection，异步包括多线程、线程池，数据处理包括序列化和反序列化）  
使用网络请求库后，实现网络请求的需求同时不需要考虑:异步请求、线程池、缓存等等；降低开发难度，缩短开发周期，使用方便





# **阻塞与非阻塞**

阻塞就是调用结果返回之前，该线程会被一直挂起，一直等待结果，不能继续，函数只有在得到结果之后才会返回

非阻塞则与阻塞概念相对，指在不能立即得到执行结果之前，该函数不会阻碍当前线程执行，而是会立即返回

**同步阻塞**

你在排队等奶茶的过程中，什么事情都不能做，只能干等着。就问你无聊不无聊，尴尬不尴尬。效率最为低下。

**同步非阻塞**

你在排队等奶茶的过程中，可以干其他事情，比如刷抖音，玩一把王者荣耀，但是你需要不断地看奶茶是否已经到你，你势必会分心导致输掉王者荣耀，成为一个坑货。注意排队等奶茶，玩王者荣耀是两件事情，你需要两件事情来回不断地切换，效率也不见得高到哪里去。

**异步阻塞**

你扫码拿号后，你不用在那里排队干等，你只需要等候服务员告诉你奶茶做好了去拿就可以了，但是在这个等的过程中，你啥事都不能干，只能干等着。很显然你已经被阻塞在这个等待服务员告诉你奶茶做好了的事情（消息通知）上面了。我们要注意是，并不是说异步就不能阻塞了，异步也是可以阻塞的，只不过它不是在处理消息时阻塞，而是在等待消息通知时被阻塞了。

**异步非阻塞**

你扫码拿号后，直接去边上玩王者荣耀了，中途你专心玩的王者荣耀，不需要分心去关注你的奶茶是否做好了，你只需要在那里等服务员告诉你奶茶做好了（消息通知）去拿就可以了。效率最高。

阻塞IO模型

应用程序发起一个系统调用（recvform），这个时候应用程序会一直阻塞下去，直到内核把数据准备好，并将其从内核复制到用户空间，复制完成后返回成功提示，这个时候应用程序才会继续处理数据

非阻塞IO模型

应用程序发起recvform系统调用，如果数据报没有准备会则会立即返回一个EWOULDBLOCK错误码，进程并不需要进行等待。进程收到该错误后，判断内核数据还没有准备好，它还可以继续发送 recvform，如果数据报已经准备好了，待数据从内核拷贝到用户空间返回成功指示后，进程则可以处理数据报了

**非阻塞IO模型需要应用进程不断地主动询问内核数据是否已准备好了，在此期间可以做别的事**。

# **okHttp**

## **简介**

## 支持http2/SPDY黑科技，共享同一个Socket来处理同一个服务器的所有请求（同一域名的所有请求stream共享同一个tcp连接），解决了HOL Blocking SPDY（读作“speedy”）是Google开发的基于TCP的应用层协议，用以最小化网络延迟，提升网络速度，优化用户的网络使用体验。SPDY是对HTTP协议的加强。新协议的功能包括数据流的多路复用、支持服务器推送技术、请求优先级、HTTP报头压缩以及强制使用SSL传输协议。 （2）socket自动选择最好路线，并支持自动重连 （3）拥有自动维护的socket连接池，减少握手次数，减少请求延时 （4）拥有队列线程池，轻松写并发 （5）拥有Interceptors（拦截器）轻松处理请求与响应（透明GZIP压缩，转换从而减少数据流量） （6）基于Headers的缓存策略减少重复的网络请求

## **2 使用步骤**

### **（1）添加okHttp和okIo**

(Okio是一款轻量级IO[框架](https://so.csdn.net/so/search?q=%E6%A1%86%E6%9E%B6&spm=1001.2101.3001.7020" \t "_blank)，是著名网络框架OkHttp的基石。Okio结合了java.io和java.nio，提供阻塞IO和非阻塞IO的功能，同时也对缓存等底层结构做了优化，能让你更轻快的获得、存储和处理数据。)

### **（2）创建OkHttpClient对象**

### **（3）get/post请求数据**

### **（4）get/post调用**

OkHttpClient client = new OkHttpClient.Builder().build();

Request request = new Request.Builder().

url("https://github.com/cozing").

build();

Call call = client.newCall(request);

try {

//1.同步请求调用的方法是call.execute()，内部采用的是线程阻塞（一直等待直到线程返回结果）方式直接将结果返回到Response

Response response = call.execute();

//2.异步请求调用的方法是call.enqueue(Callback callback)，该方法需要传入一个Callback等待结果回调的接口

call.enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) {

Log.w("cozing", "请求失败");

}

@Override

public void onResponse(Call call, Response response) throws IOException {

Log.w("cozing", "请求成功");

}

});

} catch (IOException e) {

e.printStackTrace();

}