目录

1. 网络框架对比和源码分析

https://www.jianshu.com/p/4f8e49c38271

Volley:

特点：

1) 基于 HttpUrlConnection

2) 封装了 UIL 图片加载框架,支持图片加载

3) 有缓存

4) Activity 和生命周期的联动,Activity 结束时取消在此 Activity 中调用了所有网络请求

应用场景：

1) 适合传输数据量小,网络请求频繁的场景。

2) 不能进行大数据量的网络操作,比如下载及上传文件,原因如下: Volley 的 Request 和 Response 都是把数据放到 byte[]中,如果设计文件的上传及下载,byte[]就会变得很大,严重的消耗内存.比如下载一个大文件,不可能把整个文件一次性全部放到 byte[]中再写到本地文件。

OkHttp:

特点

A. **基于 NIO 和 Okio,请求处理速度更快.**

IO:阻塞式;NIO:非阻塞式;Okio:Square 基于 IO 和 NIO 做的更高效处理数据流的库

IO 和 NIO 的区别:

1. IO 是面向流(Stream)的,而 NIO 是面向缓冲区(Buffer)的:

面向流意味着 IO 读取流中 1 个或多个字节,他们没有被缓存在任何地方,此外它不能前后移动流中的数据;

面向缓冲区意味着先将数据读取到一个稍后处理的缓冲区,需要读取时可以在缓冲区中前后移动;

2. IO 是阻塞的,NIO 是非阻塞的

IO的各种流是阻塞的,意味着一个线程执行read()或write()时,该线程被阻塞,直到数据被读取或完全写入,期间不能做任何别的事情;

NIO,一个线程从 1 个通道读取数据,或者向 1 个通道写入数据,如果通道中暂时没有数据可以读取,或者写入数据没有完成,线程不会阻塞,可以去做别的事情.直到通道中出现了可以读取的数据或者可以继续写入数据,再继续之前的工作.NIO 情况下,一个线程可以处理多个通道的读取和写入,更充分的利用线程资源;

3. IO 和 NIO 的适用场景

IO 适合于链接数量不大,但是每个链接需要发送/接收的数据量很大,需要长时间连续处理;

NIO 更适合于同时存在海量链接,但是每个链接单次发送/接收的数据量较小的情形.比如聊天服务器.海量链接但是单个链接单次数据较小

B. 无缝支持 GZIP 来减少数据流量

GZIP 是网站压缩加速的一种技术,开启后可以加快客户端的打开速度.原理是响应数据先经过服务器压缩,客户端快速解压呈现内容,减少客户端接收的数据量

android 客户端在 Request 头加入"Accept-Encoding","gzip",告知服务器客户端接受 gzip 的数据;服务器支持的情况下,返回 gzip 后的 response body，同时加入以下 header:

1) Content-Encoding: gzip：表明 body 是 gzip 过的数据

2) Content-Length:117：表示 body gzip 压缩后的数据大小，便于客户端使用。

3) Transfer-Encoding: chunked：分块传输编码

OkHttp3 是支持 Gzip 解压缩的:它支持我们在发起请求的时候自动加入 header,Accept-Encoding:gzip,而我们的服务器返回的时候也需要 header 中有 Content-Encoding:gzip

应用场景

- 重量级网络交互场景:网络请求频繁,传输数据量大

Retrofit:

特点

- 基于 OkHttp
- 通过注解配置请求
- 性能最好,处理最快
- 解析数据需要使用统一的 Converter
- 易与其他框架 RxJava 联合使用

应用场景

- 任何场景下都优先使用,特别是项目中有使用 RxJava 或者后台 API 遵循 Restful 风格

2. HttpUrlConnection 和 okhttp 关系(源码)

我们最熟悉的肯定是 HttpUrlConnection，这是 google 官方提供的用来访问网络，但是 HttpUrlConnection 实现的比较简单，只支持 1.0/1.1，并没有上面讲的多路复用，如果碰到 app 大量网络请求的时候，性能比较差，而且 HttpUrlConnection 底层也是用 Socket 来实现的。

OkHttp 与 HttpUrlConnection 一样，实现了一个网络连接的过程(OkHttp 和 HttpUrlConnection 是一级的)。OkHttp 使用的是 sink 和 source，这两个是在 Okio 这个开源库里的，sink 相当于 outputStream，source 相当于是 inputStream。Sink 和 source 比 InputSrteam 和 OutputSrewam 更加强大（其子类有缓冲 BufferedSink、支持 Gzip 压缩 GzipSink、服务于 GzipSink 的 ForwardingSink 和 InflaterSink）

3. 谈谈对 Volley 的理解(源码)

https://blog.csdn.net/ysh06201418/article/details/46443235

1. Volley 简介

我们平时在开发 Android 应用的时候不可避免地都需要用到网络技术，而多数情况下应用程序都会使用 HTTP 协议来发送和接收网络数据。Android 系统中主要提供了两种方式来进行 HTTP 通信，HttpURLConnection 和 HttpClient，几乎在任何项目的代码中我们都能看到这两个类的身影，使用率非常高。

暂时不用

4. 描述一次网络请求的流程(源码)

https://blog.csdn.net/seu_calvin/article/details/53304406

经历了

1. 域名解析、
2. TCP 的三次握手、
3. 建立 TCP 连接后发起 HTTP 请求、
4. 服务器响应 HTTP 请求、
5. 解析从服务器传过来的数据，并使用

1. 域名解析

1) 搜索自身的 DNS 缓存。如果自身的缓存中存在对应的 IP 地址并且没有过期，则解析成功。

如果未找到，那么 Chrome 会搜索操作系统自身的 DNS 缓存。如果找到且没有过期则成功。

如果上述没找到，会找 TCP/IP 参数中设置的本地 DNS 服务器。如果要查询的域名包含在本地配置的区域资源中，则完成域名解析，否则根据本地 DNS 服务器会请求根 DNS 服务器。

2) 接着本地 DNS 会把请求发至 13 台根 DNS，根 DNS 服务器收到请求后会返回负责这个域名(.net)的服务器的一个 IP，本地 DNS 服务器使用该 IP 信息联系负责.net 域的这台服务器。

这台负责.net 域的服务器收到请求后，如果自己无法解析，会返回.net 域的下一级 DNS 服务器地址给本地 DNS 服务器。以此类推，直至找到。

2. TCP 的三次握手

这个部分正好之前整理过，可以参考 NetWork——关于 TCP 协议的三次握手和四次挥手。

3. 建立 TCP 连接后发起 HTTP 请求

TCP 三次握手建立连接成功后：

A. 客户端按照指定的格式开始向服务端发送 HTTP 请求。

B. 服务端接收请求后，解析 HTTP 请求，处理完业务逻辑。

C. 最后返回一个具有标准格式的 HTTP 响应给客户端。

3.1 HTTP 请求格式

HTTP 请求格式如下所示四部分组成，分别是请求行、请求头、空行、消息体，每部分内容占一行。

<request-line>

<general-headers>

<request-headers>

<entity-headers>

<empty-line>

[<message-body>]

1) **请求行：** 由三部分组成：分别是请求方法（GET/POST/DELETE/PUT/HEAD）、URI 路径、HTTP 版本号。

2) **请求头：**缓存相关信息(Cache-Control，If-Modified-Since)、客户端身份信息（User-Agent）、是否支持 gzip 压缩，等键值对信息。

3) **空行**。

4) **主体：**客户端发给服务端的请求数据，这部分数据并不是每个请求必须的。

常用的 GET、POST、PUT、DELETE 四种请求方式中：

（1）关于 GET 和 DELETE 将要处理的资源信息直接放在了 URL 中。通过"?<键值对>&<键值对>“的形式追加。HTTP RFC 规范中并没有规定 GET 请求的 URL 长度，只是说明如果 server 无法处理太长的 URI，可以通过返回 414 状态码。但是大多数浏览器会讲其限制在 2k-8k 之间。

（2）关于 POST 和 PUT 的请求参数存储在报文的主体中。每一个参数都以”--boundary 值“+"属性信息"+”空行“+"参数值"的数据结构存储。请求数据的最后以”--boundary 值--“的格式结尾。

3. 2　服务器响应 HTTP 请求

服务器接收处理完请求后返回一个 HTTP 响应消息给客户端。HTTP 响应消息的格式包括：状态行、响应头、空行、消息体。每部分内容占一行。

<status-line>

<general-headers>

<response-headers>

<entity-headers>

<empty-line>

[<message-body>]

(1) **状态行：**有 HTTP 协议版本号，状态码和状态说明三部分构成。

(2) **响应头：**用于说明数据的一些信息，比如数据类型、内容长度等键值对。

(3) **空行。**

(4) **消息体：**服务端返回给客户端的 HTML 文本内容。或者其他格式的数据，比如：视频流、图片或者音频数据。

4 解析从服务器传过来的数据，并使用

5. 自己去设计网络请求框架，怎么

做？

（随便套个开源框架的原理）

就套 okhttp 的，被 google 承认并使用的框架，准没错。

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

6. Android 代码中实现 WAP 方式联网（原理）

https://blog.csdn.net/asce1885/article/details/7844159

无论是移动、联通还是电信，都至少提供了两种类型的的 APN：WAP 方式和 NET 方式。

其中 NET 方式跟 WIFI 方式一样，无需任何设置，可自由访问所有类型网站

而 WAP 方式，需要手机先设置代理服务器和端口号等信息，并且只能访问 HTTP 协议类型的网站。

1） 移动的 WAP 名称是 CMWAP，NET 名称是 CMNET；

2） 联通的 WAP 名称是 UNIWAP，NET 名称是 UNINET；联通 3G 的 WAP 名称是 3GWAP，NET 名称是 3GNET；

3） 电信的 WAP 名称是 CTWAP，NET 名称是 CTNET；

其中，移动和联通的 WAP 代理服务器都是 10.0.0.172，端口号是 80；而电信的 WAP 代理服务器是 10.0.0.200，端口号是 80。

在安卓系统中，对于 APN 网络的 API 都是隐蔽的，因此获取手机的 APN 设置需要通过 ContentProvider 来进行数据库查询，查询的 URI 地址是：

取得全部的 APN 列表：content://telephony/carriers；

取得当前设置的 APN：content://telephony/carriers/preferapn；

取得 current=1 的 APN：content://telephony/carriers/current；

(APN：Access Point Name ，接入点，是手机端配置的一个参数，确定手机通过哪种方式（wap or net）上网)

1. 首先将当前设置的 APN 的 url 传进 ContentResolver 以获取响应的游标对象
2. 然后通过游标分别查询当前手机所设置的 APN、Proxy 和 Port（如果手机的 Proxy 没有设置，则需要根据 APN 来决定当前应该连接的代理服务器地址和端口号）（其中，移动和联通的 WAP 代

理服务器都是 10.0.0.172，端口号是 80；而电信的 WAP 代理服务器是 10.0.0.200，端口号是 80。）

3. 接着将代理服务器的 proxy 和端口值 port，通过这两个参数构造 HttpHost 实例，并将 HttpHost 实例设置为 ConnRouteParams.DEFAULT_PROXY 的值

7. okhttp 源码

https://www.jianshu.com/p/fe43449682d6

到时候看框架

8. 网络请求缓存处理，okhttp 如何处理网络缓存的

https://www.jianshu.com/p/2821000526df

三、浅谈HTTP知识

一般的响应头：

```
HTTP/1.1 200 OK
Server: openresty
Date: Mon, 24 Oct 2016 09:00:34 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Vary: Accept-Encoding
Cache-Control: private
X-Powered-By: PHP 5.4.28
Content-Encoding: gzip
```

咱们只关心倒数第三条：Cache-Control: private

Cache-control是由服务器返回的Response中添加的头信息，它的目的是告诉客户端是要从本地读取缓存还是直接从服务器摘取消息。它有不同的值，每一个值有不同的作用。

max-age：这个参数告诉浏览器将页面缓存多长时间，超过这个时间后才再次向服务器发起请求检查页面是否有更新。对于静态的页面，比如图片、CSS、Javascript，一般都不大变更，因此通常我们将存储这些内容的时间设置为较长的时间，这样浏览器会不会向浏览器反复发起请求，也不会去检查是否更新了。

no-cache：不做缓存。

max-stale：指示客户机可以接收超出超时期间的响应消息。如果指定 max-stale 消息的值，那么客户机可以接收超出超时期指定值之内的响应消息。

和客户端有关的缓存设置

在 OKHttp 开发中我们常见到的有下面几个：max-age、no-cache、max-stale 这三类

<h5>1.配置 okhttp 中的 Cache 文件目录 </h5>

```
OkHttpClient.Builder httpClientBuilder = new
OkHttpClient.Builder();
File cacheFile = new
File(content.getExternalCacheDir(),"tangnuer");
Cache cache = new Cache(cacheFile,1024*1024*50);
```

服务器支持缓存：如果服务器支持缓存，请求返回的 Response 会带有这样的 Header:Cache-Control, max-age=xxx,这种情况下我们只需要手动给 okhttp 设置缓存就可以让 okhttp 自动帮你缓存了（加入缓存不需要我们自己来实现，Okhttp 已经内置了缓存，默认是不使用的，如果想使用缓存我们需要手动设置。）。这里的 max-age 的值代表了缓存在你本地存放的时间，可以根据实际需要来设置其大小。

服务器不支持缓存：如果服务器不支持缓存就可能没有指定这个头部，或者指定的值是如 no-store 等，这种情况下我们就需要使用 Interceptor 来重写 Respose 的头部信息，从而让 okhttp 支持缓存。思路是在返回的信息的消息头中添加 Cache-Control:

首先实现 Interceptor 类，在该类中将 Respose 头部信息重写，加入先移除头消息中的 Cache-Control，然后再加上自定义的 Cache-Control 信息进去

接着将该 Interceptor 作为一个 NetworkInterceptor 加入到 okhttpClient 中：

设置好 Cache 我们就可以正常访问了。我们可以通过获取到的 Response 对象拿到它正常的消息和缓存的消息。

这样我们就可以在服务器不支持缓存的情况下使用缓存了。

解释几个概念

Response 的消息有两种类型，CacheResponse 和 NetworkResponse。

CacheResponse 代表从缓存取到的消息，NetworkResponse 代表直接从服务端返回的消息。

第一次访问的时候，Response 的消息是 NetworkResponse 消息，此时 CacheResponse 的值为 Null.而第二次访问的时候 Response 是 CahceResponse，而此时 NetworkResponse 为空。

所以咱们的思路是：通过拿到 NetworkResponse 网络数据，做缓存，刚才这个方法其实原理就是：定义一个拦截器，人为地添加 Response 中的消息头，然后再传递给用户，这样用户拿到的 Response 就有了我们理想当中的消息头 Headers，从而达到控制缓存的意图，正所谓移花接木。

缺点：

1. 网络访问请求的资源是文本信息，如新闻列表，这类信息经常变动，一天更新好几次，它们用的缓存时间应该就很短。
2. 网络访问请求的资源是图片或者视频，它们变动很少，或者是长期不变动，那么它们用的缓存时间就应该很长。

9. 从网络加载一个 10M 的图片，说下注意事项

https://www.jianshu.com/p/7c81d3742c38

https://www.jianshu.com/p/f850a23ab99c

我们首先获得目标 View 所需的大小，然后获得图片的大小，最后通过计算屏幕与图片的缩放比，按照缩放比来解析位图。

具体步骤如下：

1) 将 BitmapFactory.Options 的 inJustDecodeBounds 参数设为 true 并加载图片

2) 从 BitmapFactory.Options 中取出图片的原始宽高信息，他们对应于 outWidth 和 outHeight 参数

3) 根据采样率的规律并结合目标 View 的所需大小计算出采样率 inSampleSize

4) 将 BitmapFactory.Options 的 inJustDecodeBounds 参数设为 false,然后重新加载图片

两个方法比较重要，在这里我们进行解析：

options.inJustDecodeBounds：如果给它赋值 true，那么它就不会解析图片。使用它的目的是为了获得图片的一些信息，如图片高度和宽度，然后进行下一步工作，也就是计算缩放比。将 options.inJustDecodeBounds 设置为 false，将会加载图片

options.inSampleSize ：给图片赋予缩放比，当它的值大于 1 的时候，它就会按照缩放比返回一个小图片用来节省内存。

除了因为图片大小自身的原因之外，还有 Android 对图片解码的因素在内。

在 Android 中使用 ARGB 来展示颜色的，一般情况下使用的是 ARGB_8888，每个像素的大小约为 4byte。如果对质量不做太大要求，可以使用 ARGB_4444 或者 RGB_565，他们都是 2 个字节的。

如果图片涉及到放大功能，则也需要注意以下事项：

1. **图片分块加载：**

图片的分块加载在地图绘制的情况上最为明显，当想获取一张尺寸很大的图片的某一小块区域时，就用到了图片的分块加载，在 Android 中 BitmapRegionDecoder 类的功能就是加载一张图片的指定区域。BitmapRegionDecoder 类的使用非常简单，API 很少并且一目了然，如下：

1) 创建 BitmapRegionDecoder 实例

2) 获取图片宽高

3) 加载特定区域内的原始精度的 Bitmap 对象

4) 调用 BitmapRegionDecoder 类中的 recycle（）,回收释放 Native 层内存

```
// 创建实例
mDecoder = BitmapRegionDecoder.newInstance(mFile.getAbsolutePath(), false);

// 获取原图片宽高
mDecoder.getWidth();
mDecoder.getHeight();

// 加载(10, 10) - (80, 80) 区域内原始精度的Bitmap对象
Rect rect = new Rect(10, 10, 80, 80);
BitmapFactory.Options options = new BitmapFactory.Options();
options.inSampleSize = 1;

Bitmap bitmap = mDecoder.decodeRegion(rect, options);

// 回收释放Native层内存
mDecoder.recycle();
```

2. **使用 LruCache**

继承并使用 LruCache，利用其来缓存加载过的图片区域，需要重写一些方法，如 sizeOf()、entryRemoved（）等

其中 sizeOf()用于处理获取缓存对象的大小，比如缓存 Bitmap 对象时，可以使用 Bitmap 的字节数作为 Bitmap 大小的表示。

entryRemoved()用于回收某个对象时调用，这样当回收 Bitmap 对象时可以调用 Bitmap 对象的 recycle()方法主动释放 Bitmap 对象的内存。

3. **手势处理：**

主要用到两个手势处理类，分别是 ScaleGestureDetector 和 GestureDetector，前者用于处理缩放手势，后者用于处理其余手势，如移动，快速滑动，点击，双击，长按等。

ScaleGestureDetector 专门处理缩放手势，其比较重要的方法是 onScale(ScaleGestureDetector detector)，当缩放时会不停地回调这个方法，需要注意的一点是 detector.getScaleFactor()获取到的缩放比例是相对上一次的

10. TCP 的 3 次握手和四次挥手

https://www.jianshu.com/p/6b2e35fdaf2c

三次握手

（1）三次握手的详述

首先 Client 端发送连接请求报文，Server 段接受连接后回复 ACK 报文，并为这次连接分配资源。Client 端接收到 ACK 报文后也向 Server 段发生 ACK 报文，并分配资源，这样 TCP 连接就建立了。

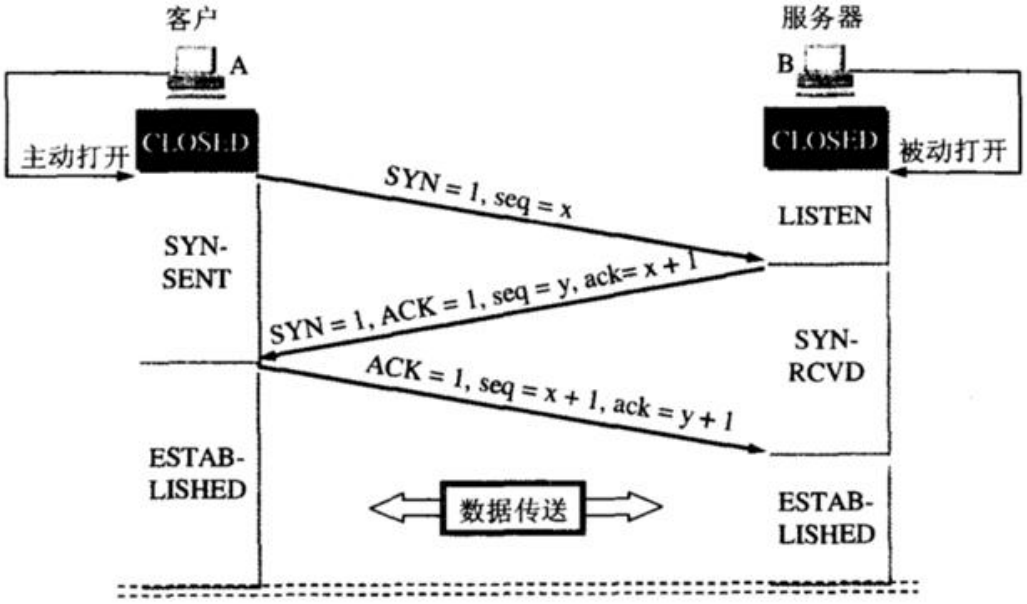


图 5-31　用三次握手建立 TCP 连接

最初两端的 TCP 进程都处于 CLOSED 关闭状态，客户端主动打开连接，而服务器被动打开连接。

1）第一次握手：TCP 客户端进程也是首先创建传输控制块 TCB，然后向 B 发出连接请求报文段，（首部的同步位 SYN=1，初始序号 seq=x），（SYN=1 的报文段不能携带数据）但要消耗掉一个序号，此时 TCP 客户进程进入 SYN-SENT（同步已发送）状态。

2）第二次握手：服务端收到连接请求报文段后，如同意建立连接，则向客户端发送确认，在确认报文段中（SYN=1，ACK=1，确认号 ack=x+1，初始序号 seq=y），测试 TCP 服务器进程进入 SYN-RCVD（同步收到）状态；

3）第三次握手：客户端进程收到 B 的确认后，要向服务端给出确认报文段（ACK=1，确认号 ack=y+1，序号 seq=x+1）（初始为 seq=x，第二个报文段所以要+1），ACK 报文段可以携带数据，不携带数据则不消耗序号。TCP 连接已经建立，客户端进入 ESTABLISHED（已建立连接）。

当服务端收到客户端的确认后，也进入 ESTABLISHED 状态。

为什么 A 还要发送一次确认呢？可以二次握手吗？

答：主要为了防止已失效的连接请求报文段突然又传送到了 B，因而产生错误。不采用三次握手，只要服务端发出确认，就建立新的连接了，此时客户端不理睬服务器的确认且不发送数据，则服务器一致等待客户端发送数据，浪费资源。

Server 端易受到 SYN 攻击？

服务器端的资源分配是在二次握手时分配的，而客户端的资源是在完成三次握手时分配的，所以服务器容易受到 SYN 洪泛攻击，SYN 攻击就是客户端 在短时间内伪造大量不存在的 IP 地址，并向 Server 不断地发送 SYN 包，Server 则回复确认包，并等待 Client 确认，由于源地址不存在，因此 Server 需要不断重发直至超时，这些伪造的 SYN 包将长时间占用未连接队列，导致正常的 SYN 请求因为队列满而被丢弃，从而引起网络拥塞甚至系统瘫痪。

防范 SYN 攻击措施：降低主机的等待时间使主机尽快的释放半连接的占用，短时间受到某 IP 的重复 SYN 则丢弃后续请求。

四次挥手

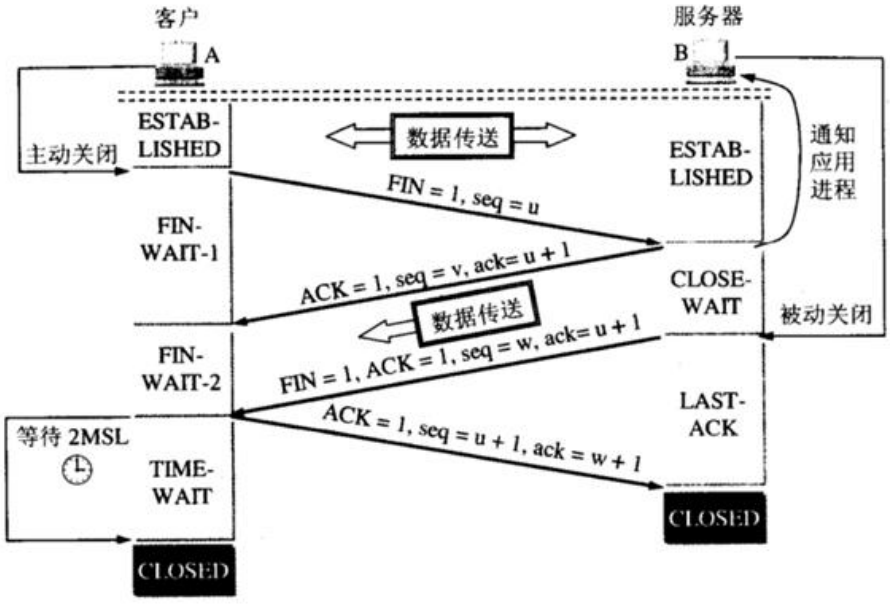


图 5-32　TCP 连接释放的过程

数据传输结束后，通信的双方都可释放连接，A 和 B 都处于 ESTABLISHED 状态。1）客户端的应用进程先向其 TCP 发出连接释放报文段（FIN=1，序号 seq=u），并停止再发送数据，主动关闭 TCP 连接，进入 FIN-WAIT-1（终止等待 1）状态，等待服务器的确认。

2）服务器收到连接释放报文段后即发出确认报文段，（ACK=1，确认号 ack=u+1，序号 seq=v），服务器进入 CLOSE-WAIT（关闭等待）状态，此时的 TCP 处于半关闭状态，客户端到服务器的连接释放。

3）客户端收到服务器的确认后，进入 FIN-WAIT-2（终止等待 2）状态，等待服务器发出的连接释放报文段。

4）服务器如果有数据要继续发送，则发送。若果没有要向客户端发出的数据，服务器发出连接释放报文段（FIN=1，ACK=1，序号 seq=w，确认号 ack=u+1），服务器进入 LAST-ACK（最后确认）状态，等待客户端的确认。

5）客户端收到服务端的连接释放报文段后，对此发出确认报文段（ACK=1，seq=u+1，ack=w+1），A 进入 TIME-WAIT（时间等待）状态。此时 TCP 未释放掉，需要经过时间等待计时器设置的时间 2MSL 后，客户端才进入 CLOSED 状态。

为什么 A 在 TIME-WAIT 状态必须等待 2MSL 的时间？

MSL 最长报文段寿命 Maximum Segment Lifetime，MSL=2

答：

两个理由：

1）保证客户端发送的最后一个 ACK 报文段能够到达服务器。

2）防止“已失效的连接请求报文段”出现在本连接中。

1）这个 ACK 报文段有可能丢失，使得处于 LAST-ACK 状态的服务器收不到对已发送的 FIN+ACK 报文段的确认，服务器超时重传 FIN+ACK 报文段，而客户端能在 2MSL 时间内收到这个重传的 FIN+ACK 报文段，接着客户端重传一次确认，重新启动 2MSL 计时器，最后客户端和服务器都进入到 CLOSED 状态，若客户端在 TIME-WAIT 状态不等待一段时间，而是发送完 ACK 报文段后立即释放连接，则无法收到服务器重传的 FIN+ACK 报文段，所以不会再发送一次确认报文段，则服务器无法正常进入到 CLOSED 状态。

2）客户端在发送完最后一个 ACK 报文段后，再经过 2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失，使下一个新的连接中不会出现这种旧的连接请求报文段。

为什么连接的时候是三次握手，关闭的时候确实却是四次握手？

答：因为当 Server 端收到 Client 端的 SYN 连接请求报文后，可以直接发送 SYN+ACK 报文。其中 ACK 报文是用来应答的，SYN 报文是用来同步的。但是关闭连接时，当 Server 端收到 FIN 报文时，很可能并不会立即关闭 SOCKET，所以只能先回复一个 ACK 报文，告诉

Client 端，"你发的 FIN 报文我收到了"。只有等到我 Server 端所有的报文都发送完了，我才能发送 FIN 报文，因此不能一起发送。故需要四步握手。

为什么 TIME_WAIT 状态需要经过 2MSL(最大报文段生存时间)才能返回到 CLOSE 状态？

答：虽然按道理，四个报文都发送完毕，我们可以直接进入 CLOSE 状态了，但是我们必须假象网络是不可靠的，有可以最后一个 ACK 丢失。所以 TIME_WAIT 状态就是用来重发可能丢失的 ACK 报文。

11. TCP 与 UDP 的区别

https://www.jianshu.com/p/9817444d64dd

TCP：TCP 是面向连接的， 可靠的流协议.

流就是指不间断的数据结构（你可以把它想象成排水管中的水流）。当应用程序采用 TCP 发送消息时， 虽然可以保证发送的顺序，但还是犹如没有任何间隔的数据流发送给接受端.

TCP 为提供可靠性传输， 实行顺序控制或重发控制机制. 此外还具有流控制(流量控制),拥塞控制,提高网络利用率等众多功能.

UDP：它是不具有可靠性的数据报协议. 细微的处理它会交给上层应用去完成.

在 UDP 的情况下，虽然可以确保发送消息的大小，却不能保证消息一定会到达，因此,应用有时会根据自己的需要进行重发处理.

UDP 主要用于那些对高速传输和实时性有较高要求的通信和广播通信. 我们举个通过 IP 电话进行通话的例子. 如果使用 TCP，数据在传输途中如果丢失会被重发，但这样无法流程地传输通话人是声音，会导致无法进行正常交流. 而采用 UDP，它会不进行重发处理. 从而也就不会有声音大幅度延迟到达的问题. 即使有部分数据丢失，也只是会影响某一小部分通话

小结 TCP 与 UDP 的区别：

1、基于连接与无连接；

2、对系统资源的要求（TCP 较多，UDP 少）；

3、UDP 程序结构较简单；

4、流模式与数据报模式 ；

5、TCP 保证数据正确性，UDP 可能丢包；

6、TCP 保证数据顺序，UDP 不保证。

12. TCP 与 UDP 的应用

TCP 是可靠的但传输速度慢 ，UDP 是不可靠的但传输速度快。因此在选用具体协议通信时，应该根据通信数据的要求而决定。

若通信数据完整性需让位与通信实时性，则应该选用 TCP 协议（如文件传输、重要状态的更新等）；反之，则使用 UDP 协议（如视频传输、实时通信等）。

13. HTTP 协议

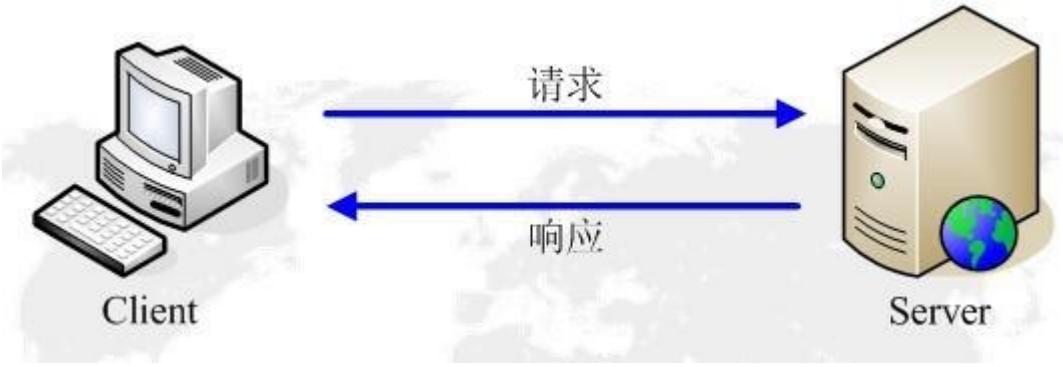
https://www.jianshu.com/p/80e25cb1d81a

https://juejin.im/post/5b2c4caff265da595534eab3

HTTP 简介

HTTP 用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。基于 TCP/IP 通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。同时属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

HTTP 客户端通过 URL 向 HTTP 服务端发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。



主要特点

1、简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

2、灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。

3.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

4.无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

5、支持 B/S 及 C/S 模式。

Http 几大属性

1) URL

2) 请求消息 Request

3) 响应消息 Response

4) 状态码

5) 请求方法

6) 工作原理

HTTP 之 URL

HTTP 使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。URL 是一种特殊类型的 URI（互联网上用来标识某一处资源的地址），包含了用于查找某个资源的足够的信息

如：http://www.aspxfans.com:8080/news/index.asp?boardID=5&ID=24618&page=1#name

一个完整的 URL 包括以下几部分：

1.协议部分：该 URL 的协议部分为“http：”，这代表使用的是 HTTP 协议。在 Internet 中可以使用多种协议，如 HTTP，FTP 等等。在"HTTP"后面的“//”为分隔符

2.域名部分：在互联网上注册的域名。一个 URL 中，也可以使用 IP 地址作为域名使用

3.端口部分：跟在域名后面的是端口，域名和端口之间使用“:”

作为分隔符。端口不是一个 URL 必须的部分，如果省略端口部分，将采用默认端口

4.虚拟目录部分：从域名后的第一个“/”开始到最后一个“/”为止，是虚拟目录部分。虚拟目录也不是一个 URL 必须的部分。

5.文件名部分：从域名后的最后一个“/”开始到“？”为止，是文件名部分，

如果没有“?”,则是从域名后的最后一个“/”开始到“#”为止，是文件部分，

如果没有“？”和“#”，那么从域名后的最后一个“/”开始到结束，都是文件名部分。文件名部分也不是一个 URL 必须的部分，如果省略该部分，则使用默认的文件名

6.锚部分：从“#”开始到最后，都是锚部分。锚部分也不是一个 URL 必须的部分

7.参数部分：从“？”开始到“#”为止之间的部分为参数部分，又称搜索部分、查询部分。参数可以允许有多个参数，参数与参数之间用“&”作为分隔符。

URI 和 URL 的区别

URI，是 uniform resource identifier，统一资源标识符，用来唯一的标识一个资源。

每种资源如文档、图像、视频片段、程序等都是一个来 URI 来定位的
URI 一般由三部组成：
①访问资源的命名机制
②存放资源的主机名
③资源自身的名称，由路径表示，着重强调于资源。

URL 是 uniform resource locator，统一资源定位器，它是一种具体的 URI，即 URL 可以用来标识一个资源，而且还指明了如何 locate 这个资源。

URL 是 Internet 上用来描述信息资源的字符串，主要用在各种 WWW 客户程序和服务器程序上。
采用 URL 可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。URL 一般由三部组成：
①协议(或称为服务方式)
②存有该资源的主机 IP 地址(有时也包括端口号)
③主机资源的具体地址。如目录和文件名等

HTTP 之请求消息 Request

客户端发送一个 HTTP 请求到服务器的请求消息包括以下格式：

请求行（request line）、请求头部（header）、空行和请求数据四个部分组成。

第一部分：请求行，用来说明请求类型,要访问的资源以及所使用的 HTTP 版本.

第二部分：请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息

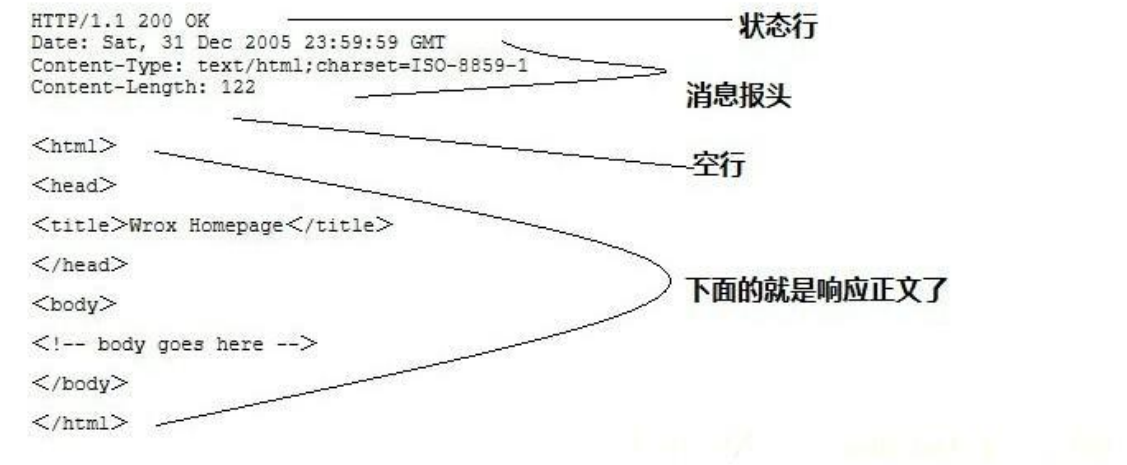
第三部分：空行，请求头部后面的空行是必须的

第四部分：请求数据也叫主体，可以添加任意的其他数据。（即使第四部分的请求数据为空，也必须有空行。）

HTTP 之响应消息 Response

一般情况下，服务器接收并处理客户端发过来的请求后会返回一个 HTTP 的响应消息。

HTTP 响应也由四个部分组成，分别是：**状态行、消息报头、空行和响应正文。**



第一部分：状态行，由 HTTP 协议版本号， 状态码， 状态消息 三部分组成。

第二部分：消息报头，用来说明客户端要使用的一些附加信息

第二行和第三行为消息报头，

Date:生成响应的日期和时间；

Content-Type:指定了 MIME 类型的 HTML(text/html),编码类型是 UTF-8

第三部分：空行，消息报头后面的空行是必须的

第四部分：响应正文，服务器返回给客户端的文本信息。

HTTP 之状态码

状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种

类别:

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

常见状态码：

200 //客户端请求成功

400 //客户端请求有语法错误，不能被服务器所理解

401 //请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用

403 //服务器收到请求，但是拒绝提供服务

404 //请求资源不存在，eg：输入了错误的 URL

500 //服务器发生不可预期的错误

503 //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

HTTP 请求方法

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD 方法。

HTTP1.1 新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

GET：请求指定的页面信息，并返回实体主体。

HEAD：类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头

POST：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。

PUT：从客户端向服务器传送的数据取代指定的文档的内容。

DELETE：请求服务器删除指定的页面。

CONNECT　HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
OPTIONS　允许客户端查看服务器的性能。
TRACE　　回显服务器收到的请求，主要用于测试或诊断。

HTTP 工作原理

以下是 HTTP 请求/响应的步骤：

1、客户端连接到服务器

2、发送 HTTP 请求：通过 TCP 套接字，客户端向服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据 4 部分组成。

3、服务器接受请求并返回 HTTP 响应：服务器将资源复本写到 TCP 套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据 4 部分组成。

4、释放连接 TCP 连接

若 connection 模式为 close，则服务器主动关闭 TCP 连接，客户端被动关闭连接，释放 TCP 连接;

若 connection 模式为 keepalive，则该连接会保持一段时间，

在该时间内可以继续接收请求;

5、客户端浏览器解析服务器传回来的内容

首先解析状态行，查看表明请求是否成功的状态代码。

然后解析每一个响应头，并使用

Http 协议定义了很多与服务器交互的方法，最基本的有 4 种，分别是 GET,POST,PUT,DELETE. 一个 URL 地址用于描述一个网络上的资源，而 HTTP 中的 GET, POST, PUT, DELETE 就对应着对这个资源的查，改，增，删 4 个操作。 我们最常见的就是 GET 和 POST 了。GET 一般用于获取/查询资源信息，而 POST 一般用于更新资源信息.

GET 方式提交数据，会带来安全问题，比如一个登录页面，通过 GET 方式提交数据时，用户名和密码将出现在 URL 上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

GET 和 POST 请求的区别

GET 请求

```
GET /books/?sex=man&name=Professional HTTP/1.1
Host: www.wrox.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
Gecko/20050225 Firefox/1.0.1
```

Connection: Keep-Alive

注意最后一行是空行

POST 请求

POST / HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 40

Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley

1）提交方式

GET 提交，请求的数据会附在 URL 之后（就是把数据放置在 HTTP 协议头中），以?分割 URL 和传输数据，多个参数用&连接

POST 提交：把提交的数据放置在是 HTTP 包的包体中。

因此，GET 提交的数据会在地址栏中显示出来，而 POST 提交，地址栏不会改变

2）提交的数据大小

GET:提交的数据大小有限制（因为浏览器对 URL 的长度有限制）

POST:提交的数据没有限制.

3）安全性

GET 方式提交数据，会带来安全问题（如登录时使用 Get 方式，会暴露用户名和密码）

POST 的安全性要比 GET 的安全性高。

总结：Http get,post,soap 协议都是在 http 上运行的

（1）get：请求参数是作为一个 key/value 对的序列（查询字符串）附加到 URL 上的查询字符串的长度受到自身和服务器的限制，不适合传输大型数据集同时，它很不安全

（2）post：请求参数是在 http 标题的一个不同部分（名为 entity body)传输的,这一部分用来传输表单信息,因此必须将 Content-type 设置为:application/x-www-form- urlencoded。

但是：它不支持复杂数据类型，因为 post 没有定义传输数据结构的语义和规则。

（3）soap：是 http post 的一个专用版本，遵循一种特殊的 xml 消息格式

Content-type 设置为: text/xml 任何数据都可以 xml 化。

14. HTTP1.0 与 2.0 的区别

https://www.jianshu.com/p/be29d679cbff

HTTP2.0 和 HTTP1.X 相比的新特性

1. **新的二进制格式（Binary Format）：**

HTTP1.x 的解析是基于文本。（文本的表现形式有多样性，要考虑的场景很多才能做到健壮性）

基于这种考虑 HTTP2.0 的协议解析决定采用二进制格式，实现方便且健壮。

2. **HTTP2.0 比 HTTP1.0 有路复用（MultiPlexing）：**即连接共享，即每一个 request 都是是用作连接共享机制的。一个 request 对应一个 id，这样一个连接上可以有多个 request，每个连接的 request 可以随机的混杂在一起，接收方可以根据 request 的 id 将 request 再归属到各自不同的服务端请求里面。

3. **header 压缩：**

HTTP1.x 的 header 带有大量信息，而且每次都要重复发送，

HTTP2.0 使用 encoder 来减少需要传输的 header 大小，通讯双方各自 cache 一份 header fields 表，既避免了重复 header 的传输，又减小了需要传输的大小。

4. **服务端推送（server push）：**HTTP2.0 也具有 server push 功能。

附注

HTTP2.0 的多路复用和 HTTP1.X 中的长连接复用有什么区别？

HTTP/1.* 一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接；

HTTP/1.1 Pipeling 解决方式为，若干个请求排队串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会，一旦有某请求超时等，后续请求只能被阻塞，毫无办法，也就是人们常说的线头阻塞；

HTTP/2 多个请求可同时在一个连接上并行执行。某个请求任务耗时严重，不会影响到其它连接的正常执行；

具体如图：

15. HTTP 报文结构

https://www.jianshu.com/p/0e6fc3e1a416

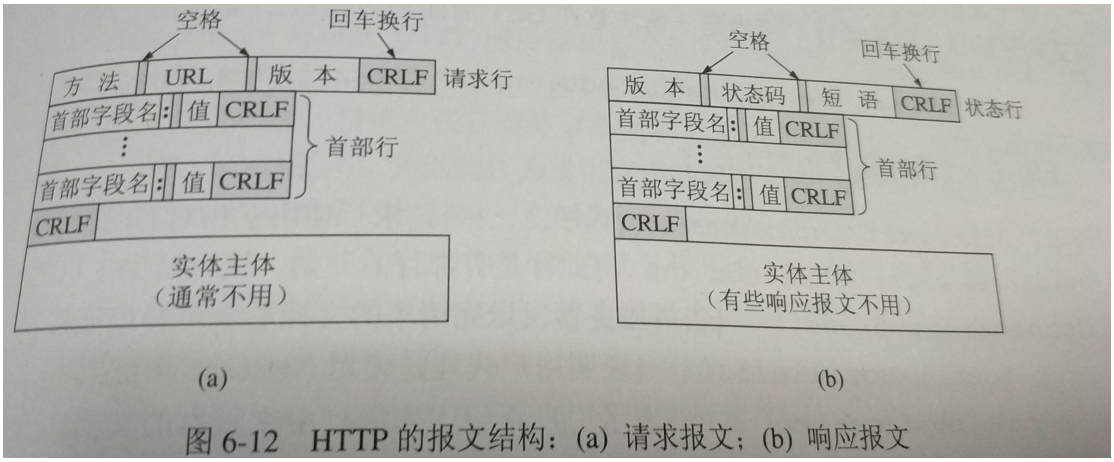


图 6-12 HTTP 的报文结构：(a) 请求报文；(b) 响应报文

HTTP 报头结构：请求报文和响应报文，请求报文是请求行，响应报文是状态行。

请求报文和响应报文都是由四部分组成的

1）报文头（initial line），例如 GET http://www.baidu.com/favicon.ico HTTP/1.1 表示请求 http://www.baidu.com/favicon.ico 这个文件，用的是 HTTP/1.1 协议

2）0 个或者多个请求头（header line），例如 Accept-language: en

3）空行

4）可选消息

请求报文：请求行、首部行，空行，信息主体

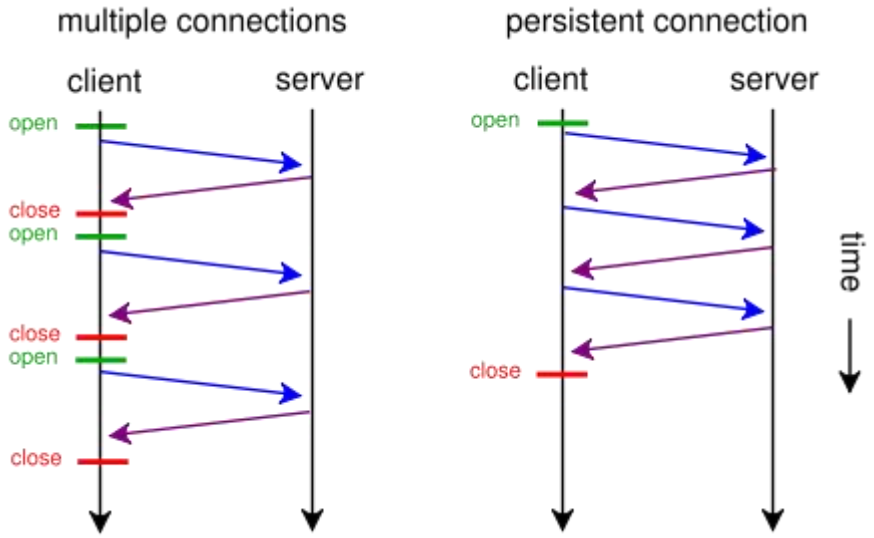
响应报文：状态行，首部行，空行，信息主体

http 使用的是无状态协议：

无状态协议：http 不要服务器保留（记忆）客户端任何状态信息。这样服务器端设计会变的简单。

Keep-Alive 模式：

我们知道 HTTP 协议采用“请求-应答”模式，当使用普通模式，即非 KeepAlive 模式时，每个请求/应答客户和服务器都要新建一个连接，完成 之后立即断开连接（HTTP 协议为无连接的协议）；当使用 Keep-Alive 模式（又称持久连接、连接重用）时，Keep-Alive 功能使客户端到服 务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive 功能避免了建立或者重新建立连接。



http 1.0 中默认是关闭的，需要在 http 头加入"Connection: Keep-Alive"，才能启用 Keep-Alive；

http 1.1 中默认启用 Keep-Alive，如果加入"Connection: close "，才关闭。目前大部分浏览器都是用 http1.1 协议，也就是说默认

都会发起 Keep-Alive 的连接请求了，所以是否能完成一个完整的 Keep- Alive 连接就看服务器设置情况。

决定请求报文的作用有五大因素：

请求报文的作用和类型由方法决定。

方法（操作）	作用
Get	请求url标志的文档。（请求url所对应的文档）
Post	向服务器发送数据
delete	删除url所对应的文档。
Put	在指明url下存储一个文档。
Head	请求url标志的文档的首部。
Option	请求一些选项的信息。
Trace	用来进行环回测试的请求报文。
Connect	用于代理服务器。

url 资源定位符：请求方法的操作对象

版本：http1.0 还是 http1.1 版本。

CRLF：回车空格。

信息主体：一般请求报文没有信息主体，只有在想服务器发送数据的时候，才会有信息主体。

响应报文格式：

（1）响应报文的第一行是状态行：http 版本，状态码，短语。

（2）状态行包括：http 的版本，状态码，短语。其中，状态码是对客户端的反馈。

（3）短语：用来解释状态码的。

状态码	说明	请求行例子
1xx	表示通知信息的，如请求收到了或正在进行处理	
2xx	表示成功	HTTP/1.1 200 OK
3xx	表示重定向	
4xx	表示客户端的差错，如请求链接为不存在	HTTP/1.1 404 Not Found
5xx	表示服务器的差错	

五种状态：

1）服务器给客户端的状态：我知道了，我正在处理

2）成功了

3）我还需要其他资源才能处理

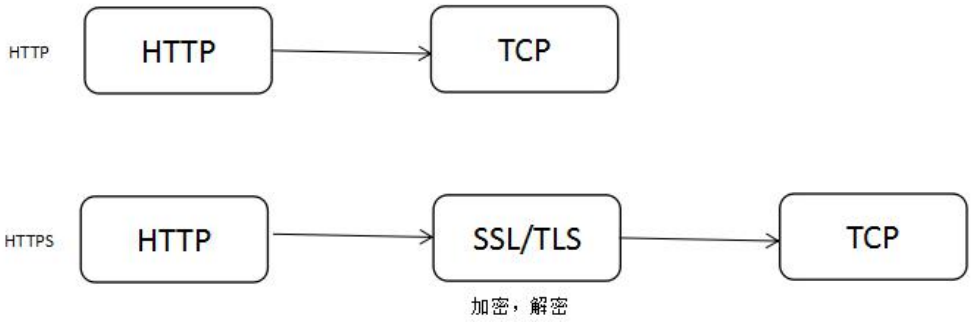
4）你发错了

5）我不能解决

16. HTTP 与 HTTPS 的区别以及如何实现安全性

1. **Url 开头：**HTTP 的 URL 以 HTTP:// 开头，而 HTTPS 的 URL 以 HTTPs:// 开头；

2. **安全性：**HTTP 是不安全的，而 HTTPS 是安全的。HTTP 协议运行在 TCP 之上，所有传输的内容都是明文，HTTPS 运行在 SSL/TLS 之上，SSL/TLS 运行在 TCP 之上，所有传输的内容都经过加密的。



3. **传输效率：**传输效率上 HTTP 要高于 HTTPS ，因为 HTTPS 需要经过加密过程，过程相比于 HTTP 要繁琐一点，效率上低一些也很正常；

4. **费用：**HTTP 无需证书，而 HTTPS 必需要认证证书；相比于 HTTP 不需要证书来说，使用 HTTPS 需要证书，申请证书是要费用的，HTTPS 这笔费用是无法避免的

5. **端口：**HTTP 和 HTTPS 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

6. **防劫持性：**HTTPS 可以有效的防止运营商劫持，解决了防劫持的一个大问题。

17. 如何验证证书的合法性?(暂无)

18. https 中哪里用了对称加密，哪里用了非对称加密，对加密算法（如 RSA）等是否有了解?

https://blog.csdn.net/tenfyguo/article/details/40922813

加密算法一般分为两种：对称加密和非对称加密。

1. 对称加密（Symmetric Key Algorithms）

对称加密算法使用的加密和解密的密钥一样，比如用秘钥 123 加密就需要用 123 解密。实际中秘钥都是普通数据在互联网传输的，这样秘钥可能会被中间人截取，导致加密被破解。

加密：将消息与秘钥通过加密算法加密出加密后的密文

解密：将加密后的密文与秘钥解密出消息

常用的对称加密算法包括：

（1）DES（Data Encryption Standard）：数据加密标准，速度较快，适用于加密大量数据的场合。

（2）3DES（Triple DES）：是基于 DES，对一块数据用三个不同的密钥进行三次加密，强度更高。

（3）AES（Advanced Encryption Standard）：高级加密标准，是下一代的加密算法标准，速度快，安全级别高；

对称加密算法的特点主要有：

（1）加密方和解密方使用同一个密钥；

（2）加解密的速度比较快，适合数据比较长时使用；

（3）密钥传输的过程不安全，且容易被破解，密钥管理也比较麻烦。

2. 非对称加密（Asymmetric Key Algorithms）

所谓非对称，是指该算法需要一对密钥，使用其中一个加密，则需要用另一个才能解密。把密钥分为公钥和私钥，公钥是公开的所有人都可以认领，私钥是保密的只有一个人知道。

非对称加密算法主要有：

（1）RSA：由 RSA 公司发明，是一个支持变长密钥的公共密钥算法，需要加密的文件块的长度也是可变的；

（2）DSA（Digital Signature Algorithm）：数字签名算法，是一种标准的 DSS（数字签名标准）；

（3）ECC（Elliptic Curves Cryptography）：椭圆曲线密码编码学。

假如发送方有一对密钥：私钥（KA）和公钥（KPA），接收方也生成一对密钥：私钥（KB）和公钥（KPB），其中（KPA）和（KPB）是公开的。

发送方用接收方的公钥对消息加密，将消息与接收方的公钥加密，生成加密后的密文。【　E=ENC（M，KPB）　】

接收方接收到密文后使用自己的私钥进行解密，将加密的密文与接收方的私钥解密，生成解密后的消息【　M=ENC（E，KB）　】

这样，即使密文被中间人截获，由于其不知道接收方的私钥，无法破解密文，所以消息仍然是安全的。

RSA 的安全性是基于极大整数因数分解的难度。

19. client 如何确定自己发送的消息被 server 收到?

HTTP 协议里，有请求就有响应，根据响应的状态吗就能知道。

20. 谈谈你对 WebSocket 的理解

https://www.jianshu.com/p/c08cc2b21496

WebSocket 介绍与原理

WebSocket 是一种双向通信协议，在建立连接后，WebSocket 服务器和 Browser/Client Agent 都能主动的向对方发送或接收数据，

就像 Socket 一样；

WebSocket 需要类似 TCP 的客户端和服务器端通过握手连接，连接成功后才能相互通信。

目的：即时通讯，替代轮询

连接过程 —— 握手过程

1. 浏览器、服务器建立 TCP 连接，三次握手。这是通信的基础，传输控制层，若失败后续都不执行。

2. TCP 连接成功后，浏览器通过 HTTP 协议向服务器传送 WebSocket 支持的版本号等信息。（开始前的 HTTP 握手）

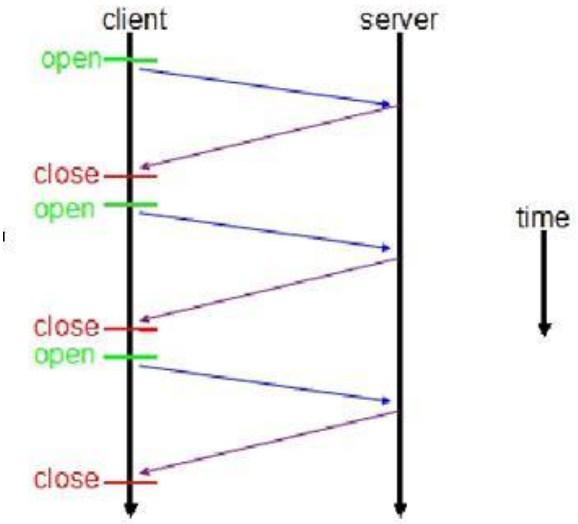
3. 服务器收到客户端的握手请求后，同样采用 HTTP 协议回馈数据。

4. 当收到了连接成功的消息后，通过 TCP 通道进行传输通信。

WebSocket 机制

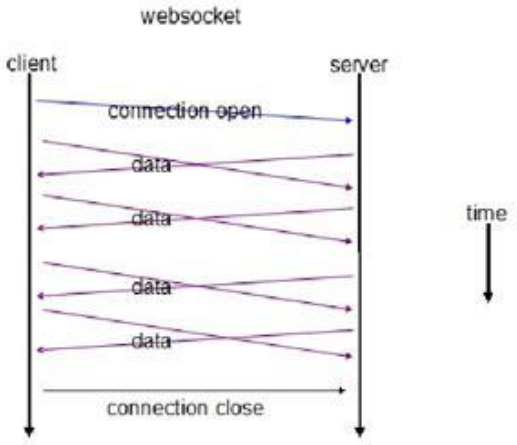
非 WebSocket 模式传统 HTTP 客户端与服务器的交互如下图所示：

图 1. 传统 HTTP 请求响应客户端服务器交互图



使用 WebSocket 模式客户端与服务器的交互如下图：

图 2.WebSocket 请求响应客户端服务器交互图



相对于传统 HTTP 每次请求-应答都需要客户端与服务端建立连接的模式，WebSocket 是类似 Socket 的 TCP 长连接的通讯模式，一旦 WebSocket 连接建立后，后续数据都以帧序列的形式传输。在客户端断开 WebSocket 连接或 Server 端断掉连接前，不需要客户端和服务端重新发起连接请求。

在海量并发及客户端与服务器交互负载流量大的情况下，极大的节省了网络带宽资源的消耗，有明显的性能优势，且客户端发送和接

受消息是在同一个持久连接上发起，实时性优势明显。

WebSocket 能更好的节省服务器资源和带宽并达到实时通讯，它建立在 TCP 之上，同 HTTP 一样通过 TCP 来传输数据。

WebSocket 和 HTTP 最大不同是：

HTTP 协议是非持久化的，单向的网络协议，在建立连接后只允许浏览器向服务器发出请求后，服务器才能返回相应的数据。这样的方法最明显的缺点就是需要不断的发送请求，而且通常 HTTP request 的 Header 是非常长的，为了传输一个很小的数据 需要付出巨大的代价，是很不合算的，占用了很多的宽带。会导致过多不必要的请求，浪费流量和服务器资源，每一次请求、应答，都浪费了一定流量在相同的头部信息上

然而 WebSocket 的出现可以弥补这一缺点。在 WebSocket 中，只需要服务器和浏览器通过 HTTP 协议进行一个握手的动作，然后单独建立一条 TCP 的通信通道进行数据的传送。

不同点

1. WebSocket 是双向通信协议，模拟 Socket 协议，可以双向发送或接受信息。HTTP 是单向的。

2. WebSocket 是需要握手进行建立连接的。

相同点

1. 都是一样基于 TCP 的，都是可靠性传输协议。

2. 都是应用层协议。

原理

WebSocket 同 HTTP 一样也是应用层的协议，但是它是一种双向通信协议，是建立在 TCP 之上的。

联系

WebSocket 在建立握手时，数据是通过 HTTP 传输的。但是建立之后，在真正传输时候是不需要 HTTP 协议的。

通过客户端和服务端交互的报文看一下 WebSocket 通讯与传统 HTTP 的不同：

在客户端，new WebSocket 实例化一个新的 WebSocket 客户端对象，连接类似 ws://yourdomain:port/path 的服务端 WebSocket URL，WebSocket 客户端对象会自动解析并识别为 WebSocket 请求，从而连接服务端端口，执行双方握手过程，客户端发送数据格式类似：

清单 1.WebSocket 客户端连接报文

<pre class="displaycode" style="margin-top: 0px; margin-bottom: 0px; white-space: pre-wrap; word-wrap: break-word; box-sizing: border-box;">GET /webfin/websocket/ HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://localhost:8080
Sec-WebSocket-Version: 13</pre>

可以看到，客户端发起的 WebSocket 连接报文类似传统 HTTP 报文，

”Upgrade：websocket”参数值表明这是 WebSocket 类型请求，
“Sec-WebSocket-Key”是 WebSocket 客户端发送的一个 base64 编码的密文，要求服务端必须返回一个对应加密的“Sec-WebSocket-Accept”应答，否则客户端会抛出“Error during WebSocket handshake”错误，并关闭连接。

服务端收到报文后返回的数据格式类似：

清单 2.WebSocket 服务端响应报文

<pre class="displaycode" style="margin-top: 0px; margin-bottom: 0px; white-space: pre-wrap; word-wrap: break-word; box-sizing: border-box;">HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=</pre>

“Sec-WebSocket-Accept”的值是服务端采用与客户端一致的密钥计算出来后返回客户端的,“HTTP/1.1 101 Switching Protocols”表示服务端接受 WebSocket 协议的客户端连接，经过这样的请求-响应处理后，客户端服务端的 WebSocket 连接握手成功, 后续就可以进行 TCP 通讯了。读者可以查阅 WebSocket 协议栈了解 WebSocket 客户端和服务端更详细的交互数据格式。

21. WebSocket 与 socket 的区别

https://blog.csdn.net/wwd0501/article/details/54582912

WebSocket 与 Socket 的关系

Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口（不是协议，为了方便使用 TCP 或 UDP 而抽象出来的一层，是位于应用层和传输控制层之间的一组接口）。在设计模式中，Socket 其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面。利用 TCP/IP 协议建立 TCP 连接。（TCP 连接则更依靠于底层的 IP 协议，IP 协议的连接则依赖于链路层等更低层次。）

WebSocket 则是一个典型的应用层协议。

区别

Socket 是传输控制层协议，WebSocket 是应用层协议。