第一次阅览本系列文章，请参见导读，更多文章请参见文章目录。
在Android的网络开发过程中，我们通常会使用像Okhttp、Retrofit这种高度封装的网络库，它们完全屏蔽了相关技术细节。但是掌握其中的原理对我们来 说是很重要的，要知其然，也要知其所以然，只要掌握了这些原理，你才能更好的理解Okhttp等网络库的源码实现。
网络编程通常会涉及以下几个角色：

- HTTP/HTTPS
- TCP/IP
- 客户端/服务端

怎么去理解它们的关系呢？

例如我们是双十一从马老板家买了部手机，这个时候我们就是客户端，马老板就是服务端。手机要通过快递公司的汽车运送到我们手中。TCP/IP就相当于汽车，但是光有汽车是不够的，还要对汽车 进行分类，不然都是一样的汽车就乱套了，而完成分类的就是HTTP/HTTPS了，HTTP/HTTPS会告诉这些汽车，你是负责送货的（GET），你是负责退货的（POST）等等。

注：文章中部分图片来源于网络，这次就偷个懒，有些流程图就不画了。

— TCP/IP

TCP（传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，

TCP协议是HTTP/HTTPS、WebSocket等协议的基础，我们首先来看看它们的报文格式。

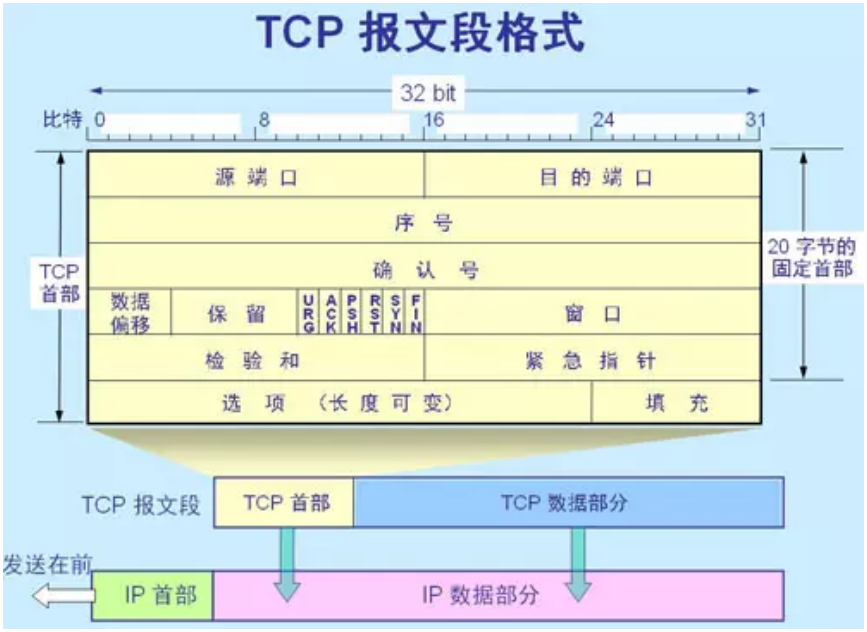
1.1 IP数据报与TCP报文

关于IP数据报与TCP报文你只需要理解它的结构就行，不用去记它，等到使用的时候不记得了，查一下就好了。
IP数据报

1. 版本——占 4 bit，指IP协议的版本. 目前的 IP 协议版本号为 4 (即 IPv4)
2. 首部长度——占 4 bit，可表示的最大数值是 15 个单位(一个单位为 4 字节)因此 IP 的首部长度的最大值是60字节。
3. 总长度——占 16 bit，指首部和数据之和的长度，单位为字节，因此数据报的最大长度为 65535 字节。总长度必须不超过最大传送单元 MTU。
4. 标识(identification) 占 16 bit，它是一个计数器，用来产生数据报的标识。当数据报需要分片时，此标识表示同一个数据报的分片。
5. 标志(flag)：3 bit，D0：MF，D1：DF，D2保留， DF位用来表示数据报是否允许分片，DF=1不分片；MF位表示是否有后续分片，MF=0表示是最后一片。
6. 片偏移(13 bit)指出：较长的分组在分片后某分片在原分组中的相对位置。片偏移以 8 个字节为偏移单位。
7. 生存时间(8 bit)记为 TTL (Time To Live)表示数据报在网络中的寿命，其单位为秒。在目前的实际应用中，常以“跳”为单位。
8. 协议(8 bit)字段指出此数据报携带的数据使用何种协议(如TCP/UDP等)以便目的主机的 IP 层将数据部分上交给哪个处理过程
9. 首部检验和(16 bit)字段只检验数据报的首部不包括数据部分。这里不采用 CRC 检验码而采用简单的“反码算术求和”计算方法。
10. 源地址和目的地址都各占 4 字节，32bit 的IP地址
11. 可选字段的长度是 可变的，1～40 字节，用于增加IP数据报的控制功能。

12. 填充字段保证IP首部长度是 4 字节的整倍数

TCP报文

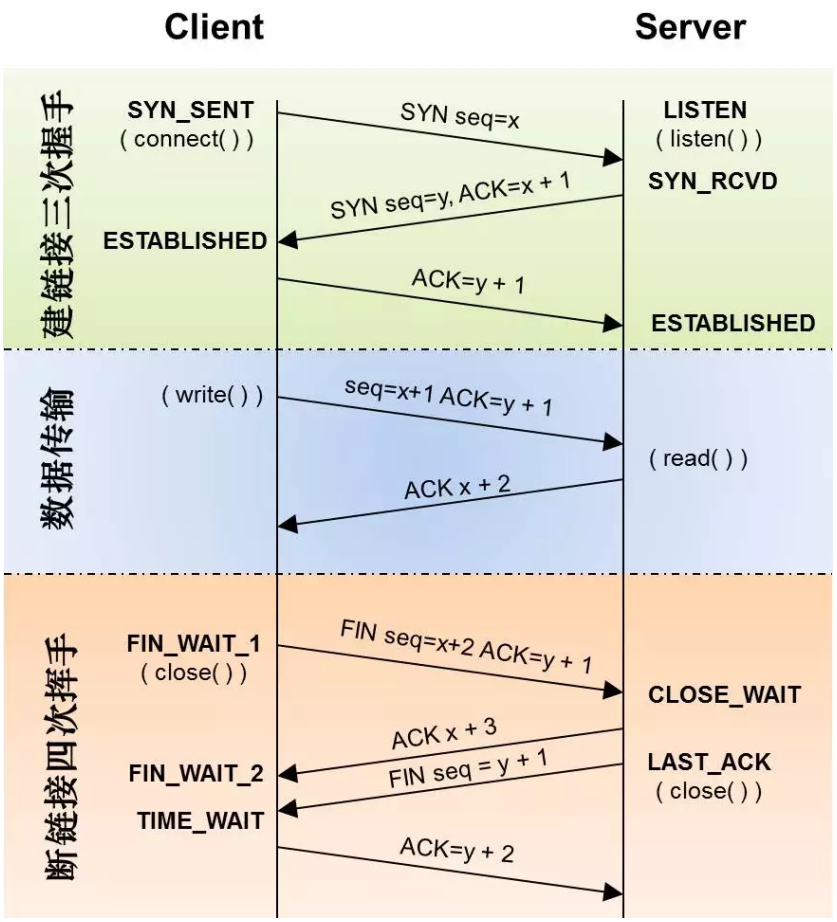


1. 源端口和目的端口字段——各占 2 字节。端口是传输层与应用层的服务接口。传输层的复用和分用功能都要通过端口才能实现。
2. 序号字段——占 4 字节。TCP 连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。
3. 确认号字段——占 4 字节，是期望收到对方的下一个报文段的数据的第一个字节的序号。
4. 数据偏移——占 4 bit，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。“数据偏移”的单位不是字节而是 32 bit 字（4 字节为计算单位）。
5. 保留字段——占 6 bit，保留为今后使用，但目前应置为 0。
6. 紧急比特 URG —— 当 URG=1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。
7. 确认比特 ACK —— 只有当 ACK=1 时确认号字段才有效。当 ACK=0 时，确认号无效。
8. 推送比特 PSH(Push)接收方 TCP 收到推送比特置1的报文段，就尽快地交付给接收应用进程，而不再等到整个缓存都填满了后再向上交付.
9. 复位比特 RST (Reset) —— 当 RST=1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。
10. 同步比特 SYN —— 同步比特 SYN 置为 1，就表示这是一个连接请求或连接接受报文。
11. 终止比特 FIN (FINal) —— 用来释放一个连接。当FIN=1 时，表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。
12. 窗口字段 —— 占 2 字节。窗口字段用来控制对方发送的数据量，单位为字节。TCP 连接的一端根据设置的缓存空间大小确定自己的接收窗口大小，然后通知对方以确定对方的发送窗口的上限。
13. 检验和 —— 占 2 字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部。
14. 紧急指针字段 —— 占 16 bit。紧急指针指出在本报文段中的紧急数据的最后一个字节的序号。
15. 选项字段 —— 长度可变。TCP 首部可以有多达40字节的可选信息，用于把附加信息传递给终点，或用来对齐其它选项。

1.2 三次握手与四次分手

TCP用三次握手（three-way handshake）过程创建一个连接，使用四次分手 关闭一个连接。

三次握手与四次分手的流程如下所示：



三次握手

- 第一次握手：建立连接。客户端发送连接请求报文段，将SYN位置为1，Sequence Number为x；然后，客户端进入SYN_SEND状态，等待服务器的确认；
- 第二次握手：服务器收到SYN报文段。服务器收到客户端的SYN报文段，需要对这个SYN报文段进行确认，设置Acknowledgment Number为x+1(Sequence Number+1)；同时，自己自己还要发送SYN请求信息，将SYN位置为1，Sequence Number为y；服务器端将上述所有信息放到一个报文段（即SYN+ACK报文段）中，一并发送给客户端，此时服务器进入SYN_RECV状态；
- 第三次握手：客户端收到服务器的SYN+ACK报文段。然后将Acknowledgment Number设置为y+1，向服务器发送ACK报文段，这个报文段发送完毕以后，客户端和服务器端都进入ESTABLISHED状态，完成TCP三次握手。 完成了三次握手，客户端和服务器端就可以开始传送数据。以上就是TCP三次握手的总体介绍。

四次分手

- 第一次分手：主机1（可以使客户端，也可以是服务器端），设置Sequence Number和Acknowledgment Number，向主机2发送一个FIN报文段；此时，主机1进入FIN_WAIT_1状态；这表示主机1没有数据要发送给主机2了；
- 第二次分手：主机2收到了主机1发送的FIN报文段，向主机1回一个ACK报文段，Acknowledgment Number为Sequence Number加1；主机1进入FIN_WAIT_2状态；主机2告诉主机1，我“同意”你的关闭请求；
- 第三次分手：主机2向主机1发送FIN报文段，请求关闭连接，同时主机2进入LAST_ACK状态；
- 第四次分手：主机1收到主机2发送的FIN报文段，向主机2发送ACK报文段，然后主机1进入TIME_WAIT状态；主机2收到主机1的ACK报文段以后，就关闭连接；此时，主机1等待2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，主机1也可以关闭连接了。

三次握手与四次分手也是个老生常谈的概念，举个简单的例子说明一下。

三次握手

例如你小时候出去玩，经常玩忘了回家吃饭。你妈妈也经常过来喊你。如果你没有走远，在门口的小土堆上玩泥巴，你妈妈会喊："小新，回家吃饭了"。你听到后会回应："知道了，一会就回去"。妈妈听 到你的回应后又说："快点回来，饭要凉了"。这样你妈妈和你就完成了三次握手的过程。😁说到这里你也可以理解三次握手的必要性，少了其中一个环节，另一方就会陷入等待之中。

三次握手的目的是为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误.

四次分手

例如偶像言情剧干净利落的分手，女主对男主说：我们分手吧😢，男主说：分就分吧😰。女主说：你果然是不爱我了，你

只知道让我多喝热水😢。男主说：事到如今也没什么好说的了，祝你幸福🙃。四次分手完成。说到这里你可以理解 了四次分手的必要性，第一次是女方（客户端）提出分手，第二次是男主（服务端）同意女主分手，第三次是女主确定男主不再爱她，也同意男主分手。第四次两人彻底拜拜（断开连接）。

因为TCP是全双工模式，所以四次分手的目的就是为了可靠地关闭连接。

二 HTTP/HTTPS

HTTP（HyperText Transfer Protocol）是一种用于分布式、协作式和超媒体信息系统的应用层协议[1]。HTTP是万维网的数据通信的基础。

HTTP是最常见的应用层协议，我们日常开发中基本上接触到的都是这个协议。

2.1 HTTP报文

HTTP应用程序是通过相互发送报文工作的，报文是HTTP应用程序之间发送的数据块，报文通常分为请求报文和响应报文两种，请求报文向服务器请求一个动作，响应报文将请求结果返回给客户端。

HTTP请求报文分为三部分：请求行、请求首部、请求实体.

- 请求行由方法字段、URL 字段 和HTTP 协议版本字段 3 个部分组成，他们之间使用空格隔开。
- 请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号":"分隔。请求头部通知服务器有关于客户端请求的信息。
- 请求实体不在 GET 方法中使用，而是在POST 方法中使用。POST 方法适用于需要客户填写表单的场合。与请求包体相关的最常使用的是包体类型 Content-Type 和包体长度 Content-Length。

请求报文

```
GET /his?wd=&from=pc_web&rf=3&hisdata=&json=1&p=3&sid=20740_20742_1424_18280_20417_17001_15840_11910_20744_20705&c
&cb=jQuery110206488567241711853_1469936513370&_=1469936513371 HTTP/1.1
Host: www.baidu.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/javascript, application/javascript, application/ecmascript, application/x-ecmascript, */*; q=0.01
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
X-Requested-With: XMLHttpRequest
Referer: https://www.baidu.com/
Cookie: BAIDUID=DB24D5F4AB36694CF00C4877ADA56562:FG=1; BIDUPSID=DB24D5F4AB36694CF00C4877ADA56562; PSTM=1469936050;
BDRCVFR[gltLrB7qNCt]=mk3SLVN4HKm; BD_CK_SAM=1; H_PS_PSSID=20740_20742_1424_18280_20417_17001_15840_11910_20744_207
BD_UPN=133252; H_PS_645EC=96a0XJobAseSCdbn9%2FviULLD7KreCHN4V4HzQtcGacKF8tGu13Nzd6j9PoB2SPPVj1d5;
 BD_HOME=0; __bsi=11860814506529643127_00_0_I_R_25_0303_C02F_N_I_I_0
Connection: keep-alive
```

响应报文

HTTP响应报文分为三部分：状态行、响应首部、响应实体。

- 状态行由 HTTP 协议版本字段、状态码和状态码的描述文本 3 个部分组成，他们之间使用空格隔开。
- 响应首部由关键字/值对组成，每行一对，关键字和值用英文冒号":"分隔。请求首部部通知客户端有关于服务端响应的信息。
- 响应实体是服务器返回给客户端的文本信息。


```
HTTP/1.1 200 OK
Server: bfe/1.0.8.14
Date: Sun, 31 Jul 2016 03:41:53 GMT
Content-Type: baiduApp/json; v6.27.2.14; charset=UTF-8
Content-Length: 95
Connection: keep-alive
Cache-Control: private
Expires: Sun, 31 Jul 2016 04:41:53 GMT
Set-Cookie: __bsi=12018325985460509248_00_0_I_R_4_0303_C02F_N_I_I_0; expires=Sun, 31-Jul-16 03:41:58 GMT; domain=w
```

报文通常由以下部分组成：

- 方法（method）：客户端希望服务器对资源执行的动作。例如：GET
- 请求URL（request url）：客户端要访问的资源URL。例如：www.google.com
- 版本（version）：报文所使用的HTTP版本。例如：HTTP/1.1
- 状态码（status code）：描述请求过程中发生的状况。例如：200
- 原因短语（reason phrase）：状态码的解释。例如：OK
- 首部（header）向请求报文或者响应报文中添加一些附加信息。例如：Content-Type: text/html
- 实体（body）：包含一个由任意数据组成的数据块。例如：

方法

方法（method）：客户端希望服务器对资源执行的动作。

- GET 请求指定url的数据,请求体为空(例如打开网页)。
- POST 请求指定url的数据，同时传递参数(在请求体中)。
- HEAD 类似于get请求，只不过返回的响应体为空，用于获取响应头。
- 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。

我们主要讨论我们最常用的两个GET/POST。

- GET方法通常用于请求服务器发送某个资源。
- POST方法通常用于向服务器提交数据。

GET与POST在本质上都是TCP连接，只是GET直接把参数写在请求行中，而POST把参数放在请求体中。关于这两个方法，要注意以下两点：

- HTTP协议本身并没有规定GET请求行的长度显示，但是浏览器和服务端有这个限制，浏览器支持的URL场地一般都2kb，服务器一般为64kb（可以设置）。
- GET中如果包含中文，需要进行编码URLEncoder.encode(params, "gbk")。

状态码

- 1xx：信息提示
- 2xx：成功
- 3xx：重定向
- 4xx：客户端错误
- 5xx：服务端错误

更多关于状态码的细节可以参见HTTP状态码。

首部

首部通常和方法配合工作，共同决定了客户端和服务器能做什么事情。

- 通用首部：客户端和服务端都可以使用的首部，提供一些通用的功能。例如：Date: Sat, 11 Jan 2003 02:44:04 GMT 提供了构建报文的时间。
- 请求首部：请求报文特有，它们为服务器提供一些额外的信息。例如：Accept: / 告知服务器会接收与其请求相符的任意媒体类型。
- 响应首部：响应报文特有，它们为客户端提供一些额外的信息。例如：Server: GWS/2.0 告知客户端与其通信的服务端是GWS/2.0。
- 实体首部：实体中特有，为实体提供一些说明。例如：Content-Type: text/html 告知实体中的内容类型是text/html。
- 扩展首部：非标准首部，可以由开发者创建。

常见的通用首部

- Date 发送该消息的日期和时间(按照 RFC 7231 中定义的"超文本传输协议日期"格式来发送) Date: Tue, 15 Nov 1994 08:12:31 GMT
- Host 服务器的域名(用于虚拟主机)，以及服务器所监听的传输控制协议端口号。如果所请求的端口是对应的服务的标准端口，则端口号可被省略。

常见的请求首部

- Accept 能够接受的回应内容类型（Content-Types）。Accept: text/plain
- Accept-Charset 能够接受的字符集 Accept-Charset: utf-8
- Accept-Encoding 能够接受的编码方式列表。 Accept-Encoding: gzip, deflate
- Accept-Language 能够接受的回应内容的自然语言列表。 Accept-Language: en-US
- Accept-Datetime 能够接受的按照时间来表示的版本 Accept-Datetime: Thu, 31 May 2007 20:35:00 GMT
- Authorization 用于超文本传输协议的认证的认证信息 Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
- Cookie 之前由服务器通过 Set- Cookie （下文详述）发送的一个 超文本传输协议Cookie Cookie: $Version=1; Skin=new;
- User-Agent 浏览器的浏览器身份标识字符串 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/21.0
- Upgrade 要求服务器升级到另一个协议。 Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11

常见的响应首部

- Server 服务器的名字 Server: Apache/2.4.1 (Unix)
- Status 用来说明当前这个超文本传输协议回应的 状态。Status: 200 OK
- Upgrade 要求客户端升级到另一个协议。 Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
- Age 这个对象在代理缓存中存在的时间，以秒为单位 Age: 12
- Cache-Control 向从服务器直到客户端在内的所有缓存机制告知，它们是否可以缓存这个对象。其单位为秒 Cache-Control: max-age=3600 常设
- Connection 针对该连接所预期的选项 Connection: close
- Location 用来进行重定向，或者在创建了某个新资源时使用。 Location: http://www.w3.org/pub/WWW/People.html

常见的实体首部

- Content-Type 当前内容的MIME类型 Content-Type: text/html; charset=utf-8
- Content-Length 回应消息体的长度，以 字节 （8位为一字节）为单位 Content-Length: 348
- Content-Encoding 在数据上使用的编码类型。 Content-Encoding: gzip
- Content-Language 内容所使用的语言
- Content-Language 内容所使用的语言
- Content-Location 所返回的数据的一个候选位置 Content-Location: /index.htm
- Content-MD5 回应内容的二进制 MD5 散列，以 Base64 方式编码 Content-MD5: Q2hlY2sgSW50ZWdyaXR5IQ==
- Content-Range 这条部分消息是属于某条完整消息的哪个部分 Content-Range: bytes 21010-47021/47022

更多关于首部的细节可以参见HTTP首部。

2.2 HTTP与HTTPS

HTTPS是一种通过计算机网络进行安全通信的传输协议。HTTPS经由HTTP进行通信，但利用SSL/TLS来加密数据包。HTTPS开发的主要目的，是提供对网站服务器的身份 认证，保护交换数据的隐私与完整性。

如下图所示，可以很明显的看出两个的区别：

注：TLS是SSL的升级替代版，具体发展历史可以参考传输层安全性协议。
HTTP与HTTPS在写法上的区别也是前缀的不同，客户端处理的方式也不同，具体说来：

- 如果URL的协议是HTTP，则客户端会打开一条到服务端端口80（默认）的连接，并向其发送老的HTTP请求。
- 如果URL的协议是HTTPS，则客户端会打开一条到服务端端口443（默认）的连接，然后与服务器握手，以二进制格式与服务器交换一些SSL的安全参数，附上加密的 HTTP请求。

所以你可以看到，HTTPS比HTTP多了一层与SSL的连接，这也就是客户端与服务端SSL握手的过程，整个过程主要完成以下工作：

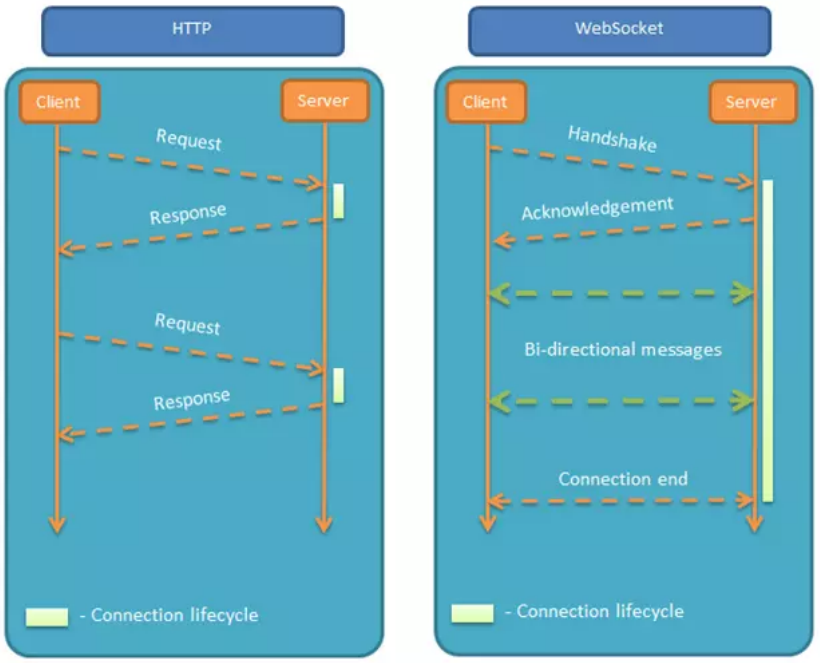
- 交换协议版本号
- 选择一个两端都了解的密码
- 对两端的身份进行认证
- 生成临时的会话密钥，以便加密信道。

SSL握手是一个相对比较复杂的过程，更多关于SSL握手的过程细节可以参考TLS/SSL握手过程
SSL/TSL的常见开源实现是OpenSSL，OpenSSL是一个开放源代码的软件库包，应用程序可以使用这个包来进行安全通信，避免窃听，同时确认另一端连接者的身份。这个包广泛被应用在互联网的网页服务器上。 更多源于OpenSSL的技术细节可以参考OpenSSL。

三 WebSocket

> WebSocket是一种在单个TCP连接上进行全双工通讯的协议，它使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握 手，两者之间就直接可以创建持久性的连接，并进行双向数据传输。

为什么需要WebSocket，因为 HTTP 协议有一个缺陷：通信只能由客户端发起。而WebSocket可以实现双向通信。一般来说WebSocket是用来实现双工通信的长连接的。HTTP想要达到 这种效果，一般会通过轮询或者long poll来实现，这样比较占用资源且非常被动。



一个典型的WebSocket请求与响应
客户端请求

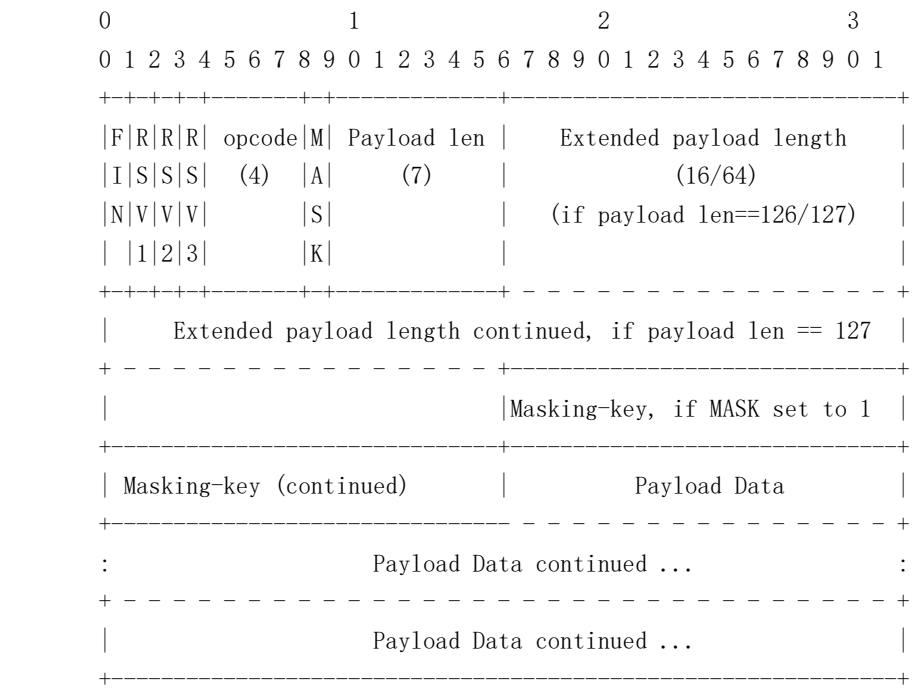
```
GET / HTTP/1.1
```

```
Upgrade: websocket
Connection: Upgrade
Host: example.com
Origin: http://example.com
Sec-WebSocket-Key: sN9cRrP/n9NdMgdcy2VJFQ==
Sec-WebSocket-Version: 13
```

服务器响应

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: fFBooB7FAkL1XgRSz0BT3v4hq5s=
Sec-WebSocket-Location: ws://example.com/
```

这里会使用Upgrade: websocket Connection: Upgrade 提示当前发起的是WebSocket协议，注意升级协议。
注：Okhttp已支持WebSocket。
我们同样也来看看WebSocket的报文结构。



- FIN：1位，标识消息是否是最后一帧。1个消息由1个或者多个数据帧组成，若消息由1帧构成，则起始帧就是结束帧。
- RSV1，RSV2，RSV3：1位，预留位，用于自定义扩展。如果没有扩展，则各位为0；如果定义扩展，即各位非0，但扩展中没有该值的定义，则关闭连接。
- opcode：4位，标识帧类型，帧类型微分控制帧与非控制帧，如果接收到未知帧，接收端就必须关闭连接。
- MASK：1位，掩码位，标识帧里的数据是否经过加密，如果帧经过掩码加密处理，该位为1，Masking-key帧的数据就是掩码密钥，用于解码Payload。 WebSocket协议规定

已定义的帧类型：

- %x0 denotes a continuation frame
- %x1 denotes a text frame
- %x2 denotes a binary frame
- %x3-7 are reserved for further non-control frames
- %x8 denotes a connection close
- %x9 denotes a ping
- %xA denotes a pong
- %xB-F are reserved for further control frames

WebSocket协议的控制帧有3种：

- 关闭帧：用于关闭连接，客户端可以发送关闭帧给服务端，服务端也可以发送关闭帧给客户端。
- Ping/Pong帧：用于心跳检测，服务端向客户端发送Ping帧，客户端回复Pong帧，表示连接还存在，可以继续通信。

前面我们讲完了几种常用的协议，最后我们再看看和编码相关的知识，这在日常的业务开发中也经常用到。

四 实体与编码

前面我们已经提到过与内容编码相关的实体首部：

- Content-Type 当前内容的MIME类型 Content-Type: text/html; charset=utf-8
- Content-Length 回应消息体的长度，以字节 （8位为一字节）为单位 Content-Length: 348
- Content-Encoding 在数据上使用的编码类型。 Content-Encoding: gzip
- Content-Language 内容所使用的语言
- Content-Location 所返回的数据的一个候选位置 Content-Location: /index.htm
- Content-MD5 回应内容的二进制 MD5 散列，以 Base64 方式编码 Content-MD5: Q2hlY2sgSW50ZWdyaXR5IQ==
- Content-Range 这条部分消息是属于某条完整消息的哪个部分 Content-Range: bytes 21010-47021/47022

对于我们来说，比较常见到的也需要重点关注的是Content-Type、Content-Encoding这两个。
Content-Type描述的是当前内容的MIME类型，关于MIME类型：

MIME：表示一种主要的对象类型和一个特定的子类型。

它主要有以下几种类型：

- Text：用于标准化地表示的文本信息，文本消息可以是多种字符集和或者多种格式的；
- Multipart：用于连接消息体的多个部分构成一个消息，这些部分可以是不同类型的数据；
- Application：用于传输应用程序数据或者二进制数据；
- Message：用于包装一个E-mail消息；
- Image：用于传输静态图片数据；
- Audio：用于传输音频或者音声数据；
- Video：用于传输动态影像数据，可以是与音频编辑在一起的视频数据格式。

Content-Encoding描述的是编码类型，它的意义在于告诉服务端当前客户端支持的编码方式，这样服务端就会根据该编码方式来编码数据。如果没有该首部，则默认认为 客户端支持所有的编码方式。

Accept-Encoding 能够接受的编码方式列表。 Accept-Encoding: gzip:q=1.0, deflate:q=0.5, *:q=0

另外Accept-Encoding还可以用q来表示编码优先级，1.0表示最希望使用的编码，0表示不想接受该编码。
常见的编码类型有：

- compress – UNIX的“compress”程序的方法（历史性，不推荐大多数应用使用，应该使用gzip或deflate）
- deflate – 基于deflate算法（定义于RFC 1951）的压缩，使用zlib数据格式（RFC 1950）封装
- exi – W3C高效XML交换
- gzip – GNU zip格式（定义于RFC 1952）。此方法截至2011年3月，是应用程序支持最广泛的方法。
- identity – 不转换内容。这是内容编码的默认值。
- pack200-gzip – 传输Java存档文件的网络传输格式

当然我们常用的就是gzip，Okhttp里面可以利用Okio进行gzip压缩，这里我们也贴下代码。

```
/** This interceptor compresses the HTTP request body. Many webservers can't handle this! */
final class GzipRequestInterceptor implements Interceptor {
  @Override public Response intercept(Interceptor.Chain chain) throws IOException {
    Request originalRequest = chain.request();
    if (originalRequest.body() == null || originalRequest.header("Content-Encoding") != null) {
      return chain.proceed(originalRequest);
    }

    Request compressedRequest = originalRequest.newBuilder()
        .header("Content-Encoding", "gzip")
        .method(originalRequest.method(), gzip(originalRequest.body()))
        .build();
    return chain.proceed(compressedRequest);
  }

  private RequestBody gzip(final RequestBody body) {
    return new RequestBody() {
      @Override public MediaType contentType() {
```

```
        return body.contentType();
      }

      @Override public long contentLength() {
        return -1; // We don't know the compressed length in advance!
      }

      @Override public void writeTo(BufferedSink sink) throws IOException {
        BufferedSink gzipSink = Okio.buffer(new GzipSink(sink));
        body.writeTo(gzipSink);
        gzipSink.close();
      }
    };
  }
}
```