HTTP 协议

一、HTTP 超文本传输协议

HTTP 的设计目的是保证客户机与服务器之间的通信。

HTTP 的工作方式是客户机与服务器之间的请求-应答协议。

1、HTTP 请求方式：

```
HTTP 请求方式：
GET - 从指定的资源请求数据。
POST - 向指定的资源提交要被处理的数据。
HEAD - 与 GET 相同，但只返回 HTTP 报头，不返回文档主体。
PUT - 上传指定的 URI 表示。
DELETE - 删除指定资源。
OPTIONS - 返回服务器支持的 HTTP 方法。
CONNECT - 把请求连接转换到透明的 TCP/IP 通道。
```

重点是 post和get

2、GET 和 POST比较

下面的表格比较了两种 HTTP 方法：GET 和 POST。

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
书签	可收藏为书签	不可收藏为书签
缓存	能被缓存	不能缓存
编码类型	application/x-www-form-urlencoded	application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。
历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。 在发送密码或其他敏感信息时绝不要使用 GET ！	POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。

GET：当客户端要从服务器中读取某个资源时，使用GET 方法。GET 方法要求服务器将URL 定位的资源放在响应报文的 数据 部分，回 送给 客户端，即向服务器请求某个资源。使用GET 方法时，请求参数和对应的值附加在 URL 后面，利用一个问号(“?”)代表URL 的结尾与请求参数的开始，传递参数长度受限制。例如，/index.jsp?id=100&op=bind。

● POST：当客户端给服务器提供信息较多时可以使用POST 方法，POST 方法向服务器提交数据，比如完成表单数据的提交，将数据提交给服务器处理。GET 一般用于获取/查询资源信息，POST 会附带用户数据，一般用于更新资源信息。POST 方法将请求参数 封装 在HTTP 请求数据中，以名称/值的形式出现，可以传输大量数据;

3、请求头部：

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，典型的请求头有：

● User-Agent：产生请求的浏览器类型;

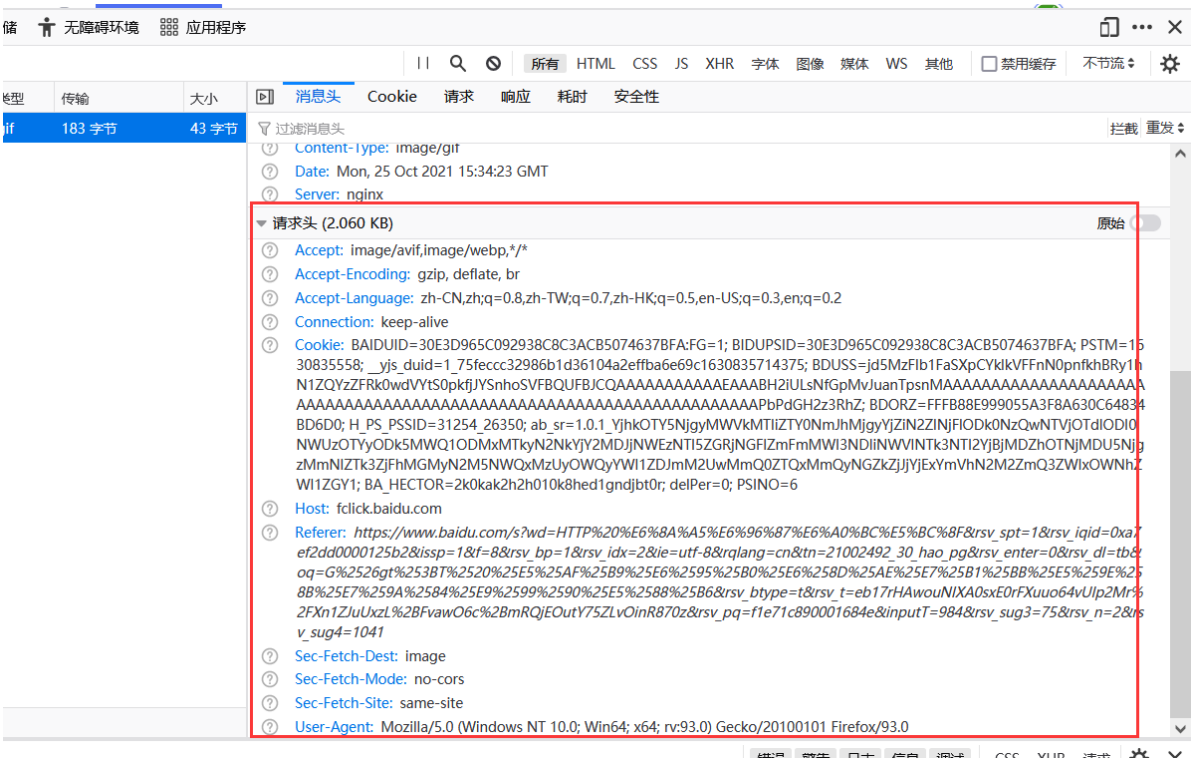
● Accept：客户端可识别的响应内容类型列表;星号 “ * ” 用于按范围将类型分组，用 “ / ” 指示可接受全部类型，用“ type/* ”指示可接受 type 类型的所有子类型;

● Accept-Language：客户端可接受的自然语言;

● Accept-Encoding：客户端可接受的编码压缩格式;

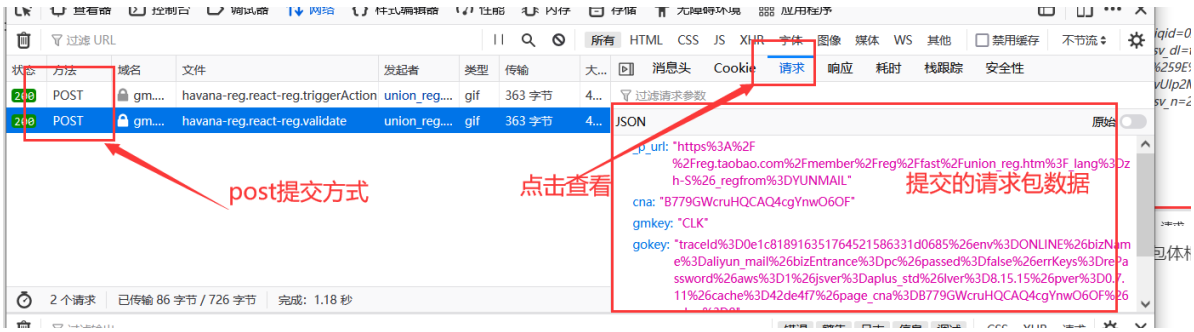
- Accept-Charset：可接受的应答的字符集;
- Host：请求的主机名，允许多个[域名]同处一个IP 地址，即虚拟主机;
- connection：连接方式(close 或 keepalive);
- Cookie：存储于客户端扩展字段，向同一域名的服务端发送属于该域的cookie;

空行：最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头;



请求包体：请求包体不在 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;

如：https://mailsso.mxhichina.com/aliyun/register?lang=zh_CN 提交数据后：



3、HTTP 报文格式：

由从客户机到服务器的请求和从服务器到客户机的响应构成。

```
请求报文格式如下：
请求行 — 通用信息头 — 请求头 — 实体头 — 报文主体

应答报文格式如下：
状态行 — 通用信息头 — 响应头 — 实体头 — 报文主体
```

HTTP请求报文由三部分组成

HTTP 请求报文

请求行由（123）组成：1是请求方法；2是url地址，它和报文头的Host属性组成完整的请求URL；3是协议名称和版本号。

请求头（4）：是HTTP的报文头，报文头包含若干个属性，格式均为"属性名：属性值"，服务端由此获得客户端的信息。与缓存有关的信息都放在头部（header）。（key:value的形式，一个key对应一个value，一个key对应多个value，但是其实一个key也可以对应多个value。结果区别就是aa：bb和aa：bb，cc）

请求体（5）：它将一个页面表单中的组件值通过param1=value1&param2=value2的键值对形式编码成一个格式化串，它承载着多个请求参数的数据。

```
HTTP请求报文属性：

Accept ：请求报文通过Accept属性告诉服务器，大哥我就只能接受这个类型的响应（比如纯文本）,你发图片我就GG了!(当然了，Accept属性的值可以为一个或者多个MIME类型的值，就可以接受几种响应啦)

cookie：缓存信息。

Cache-Control：对缓存进行控制。比如你希望禁止缓存，或者缓存一年，或者一月，这些都是通过设置这个属性。
```

HTTP响应报文也由三部分组成

响应行（12） 组成：1是报文协议及版本，2是状态码及描述。

响应头（3）：和请求头一样，由属性组成。

响应体（4）：是服务器返回给客户端的文本信息

```
 HTTP响应报文属性

Cache-Control ：响应输出到客户端后，服务端通过该报文头属告诉客户端如何控制响应内容的缓存。常见的有：（默认        为private）

        private:                客户端可以缓存
        public:                 客户端和代理服务器都可缓存（前端的同学，可以认为public和private是一样的）
        max-age=xxx:    缓存的内容将在 xxx 秒后失效
        no-cache:               需要使用对比缓存来验证缓存数据
        no-store:               所有内容都不会缓存

Location：当我们想要页面重定向redirect的时候，设置Location的属性值（地址）跳转到该地址

Cookie：缓存信息
```

响应状态码,5大类

```
1xx 消息，一般是告诉客户端，请求已经收到了，正在处理，别急...
2xx 处理成功，一般表示：请求收悉、我明白你要的、请求已受理、已经处理完成等信息.
3xx 重定向到其它地方。它让客户端再发起一个请求以完成整个处理。
4xx 处理发生错误，责任在客户端，如客户端的请求一个不存在的资源，客户端未被授权，禁止访问等。
5xx 处理发生错误，责任在服务端，如服务端抛出异常，路由出错，HTTP版本不支持等。
```

常见状态码：

◆200 (OK): 找到了该资源，并且一切正常。

◆302/307：临时重定向，指出请求的文档已被临时移动到别处, 此文档的新的url在location响应头中给出

◆304 (NOT MODIFIED): 该资源在上次请求之后没有任何修改。这通常用于浏览器的缓存机制。

◆401 (UNAUTHORIZED): 客户端无权访问该资源。这通常会使得浏览器要求用户输入用户名和密码，以登录到服务器。

◆403 (FORBIDDEN): 客户端未能获得授权。这通常是在401之后输入了不正确的用户名或密码。

◆404 (NOT FOUND): 在指定的位置不存在所申请的资源。

◆500 Internal Server Error：看到这个错误，你就应该查查服务端的日志了，肯定抛出了一堆异常，别睡了，起来改BUG去吧！

5、HTTP 协议版本：

```
1、 HTTP/0.9  已过时。只接受 GET 一种请求方法，没有在通讯中指定版本号，且不支持请求头。由于该版本不支持 POST 方法，所以客户端无法向服务器传递太多信息。
2、 HTTP/1.0  这是第一个在通讯中指定版本号的HTTP 协议版本，至今仍被广泛采用，特别是在代理服务器中。
3、 HTTP/1.1  当前版本。持久连接被默认采用，并能很好地配合代理服务器工作。还支持以管道方式同时发送多个请求，以便降低线路负载，提高传输速度。
```

6、HTTP/1.1 HTTP/1.0 协议区别：

```
缓存处理
带宽优化及网络连接的使用
错误通知的管理
消息在网络中的发送
互联网地址的维护
安全性及完整性
```

二、HTTPS(Secure Hypertext Transfer Protocol)安全超文本传输协议

它是一个安全通信通道，它基于HTTP开发，用于在客户计算机和服务器之间交换信息。它使用安全套接字层(SSL)进行信息交换，简单来说它是HTTP的安全版。 它是由Netscape开发并内置于其浏览器中，用于对数据进行压缩和解压操作，并返回网络上传送回的结果。

HTTPS实际上应用了Netscape的安 全全套接字层(SSL)作为HTTP应用层的子层。()SSL使 用40 位关键字作为RC4流加密算法，这对于商业信息的加密是合适的。HTTPS和SSL支持使用X.509数字认证，如果需要的话用户可以确认发送者是谁。

1、HTTPS和HTTP的区别：

```
HTTPS 协议需要有ca申请证书。
HTTP 是超文本传输协议，信息是明文传输。
HTTPS 则是具有安全性的ssl加密传输协议。
HTTPS 使用端口443。
HTTP 那样使用端口80来和TCP/IP进行通信。HTTP 效率更高，HTTPS 安全性更高。
```

https 首页打开会比 http 消耗一点CPU ； 相对会慢一点。当建立稳定链接之后就和HTTP一样了。

2、HTTPS首次请求速度影响：

```
HTTPS 对速度有什么影响。影响主要来自两方面：
协议交互所增加的网络 RTT(round trip time)。
加解密相关的计算耗时。
```

3、HTTPS 网络耗时原因：

由于 HTTP 和 HTTPS 都需要 DNS 解析，并且大部分情况下使用了 DNS 缓存，为了突出对比效果，忽略主域名的 DNS 解析时间。

然后都进行三次握手协议。但是 HTTPS 的访问过程，相比 HTTP 要复杂很多，在部分场景下，使用 HTTPS 访问有可能增加 7 个 RTT。(RTT定义是第一个发送端有数据发送开始到接收到确认信号经历的时间)

```
HTTPS 首次请求需要的网络耗时：

1、三次握手建立 TCP 连接。耗时一个 RTT。

2、使用 HTTP 发起 GET 请求，服务端返回 302 跳转到 https://www.baidu.com。需要一个 RTT 以及 302 跳转延时。

    大部分情况下用户不会手动输入 https://www.baidu.com 来访问 HTTPS，服务端只能返回 302 强制浏览器跳转到 https。

    浏览器处理 302 跳转也需要耗时。

3、三次握手重新建立 TCP 连接。耗时一个 RTT。

    302 跳转到 HTTPS 服务器之后，由于端口和服务器不同，需要重新完成三次握手，建立 TCP 连接。

4、TLS 完全握手阶段一。耗时至少一个 RTT。

    这个阶段主要是完成加密套件的协商和证书的身份认证。

    服务端和浏览器会协商出相同的密钥交换算法、对称加密算法、内容一致性校验算法、证书签名算法、椭圆曲线（非 ECC 算法不需要）等。

    浏览器获取到证书后需要校验证书的有效性，比如是否过期，是否撤销。

5、解析 CA 站点的 DNS。耗时一个 RTT。

    浏览器获取到证书后，有可能需要发起 OCSP 或者 CRL 请求，查询证书状态。

    浏览器首先获取证书里的 CA 域名。

    如果没有命中缓存，浏览器需要解析 CA 域名的 DNS。

6、三次握手建立 CA 站点的 TCP 连接。耗时一个 RTT。

     DNS 解析到 IP 后，需要完成三次握手建立 TCP 连接。

7、发起 OCSP 请求，获取响应。耗时一个 RTT。

8、完全握手阶段二，耗时一个 RTT 及计算时间。

    完全握手阶段二主要是密钥协商。

9、 完全握手结束后，浏览器和服务器之间进行应用层（也就是 HTTP）数据传输。
```

请求增加RTT情况

也并不是每个请求都需要增加 7 个 RTT 才能完成 HTTPS 首次请求交互。

```
、必须是首次请求。即建立 TCP 连接后发起的第一个请求，该连接上的后续请求都不需要再发生上述行为。
、必须要发生完全握手，而正常情况下 80% 的请求能实现简化握手。
、浏览器需要开启 OCSP 或者 CRL 功能。Chrome 默认关闭了 ocsp 功能，firefox 和 IE 都默认开启。
、浏览器没有命中 OCSP 缓存。Ocsp 一般的更新周期是 7 天，firefox 的查询周期也是 7 天，也就说是 7 天中才会发生一次 ocsp 的查询。
、浏览器没有命中 CA 站点的 DNS 缓存。只有没命中 DNS 缓存的情况下才会解析 CA 的 DNS。
```

4、**三次握手：**

```
第一次握手：建立连接时，客户端发送syn包（syn=j）到服务器，并进入SYN_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。
第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，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包，向服务器发送确认包ACK(ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。
```

第一次握手:（Client向Server发送联机请求）

SYN=1（Client向Server发送联机请求）

Client想要与Server进行TCP通信,首先他需要向Server发送一个**SYN=1的同步序列编号(syncsynchronized squsequence number)用来表示建立连接**,并且随机产生一个数Seq number = X的数据包到Server，Server由于SYN=1知道，Client要求建立联机，到这里第一次握手就结束了

第二次握手:（Server向Client回复联机并确认联机信息）

SYN=1（Server接受Client的联机请求）

ACK=1(确认信息)

这是对第一次握手信息的确认，表示Server收到了Client的第一次握手信息

Ack=X+1(确认回复)

同时Server回复Client一个确认码**Ack**表示你的联机请求我已经收到，而且数据没有丢失，怎么验证数据没有丢失呢？即Ack的值等于Client发过来Seq的值加1，即**Ack = X+1。**因为我都知道你发过来的Seq的值，所以这个数据包没有丢失。

Seq = Y（第二次握手的数据包序列号）

Server给Client的数据包序列号，为了数据包在到达Client之后的验证，所以这次从Server到Client的数据包中同样也会随机产生一个Seq number = Y，

第三次握手

ACK=1（对第二次握手的确认）

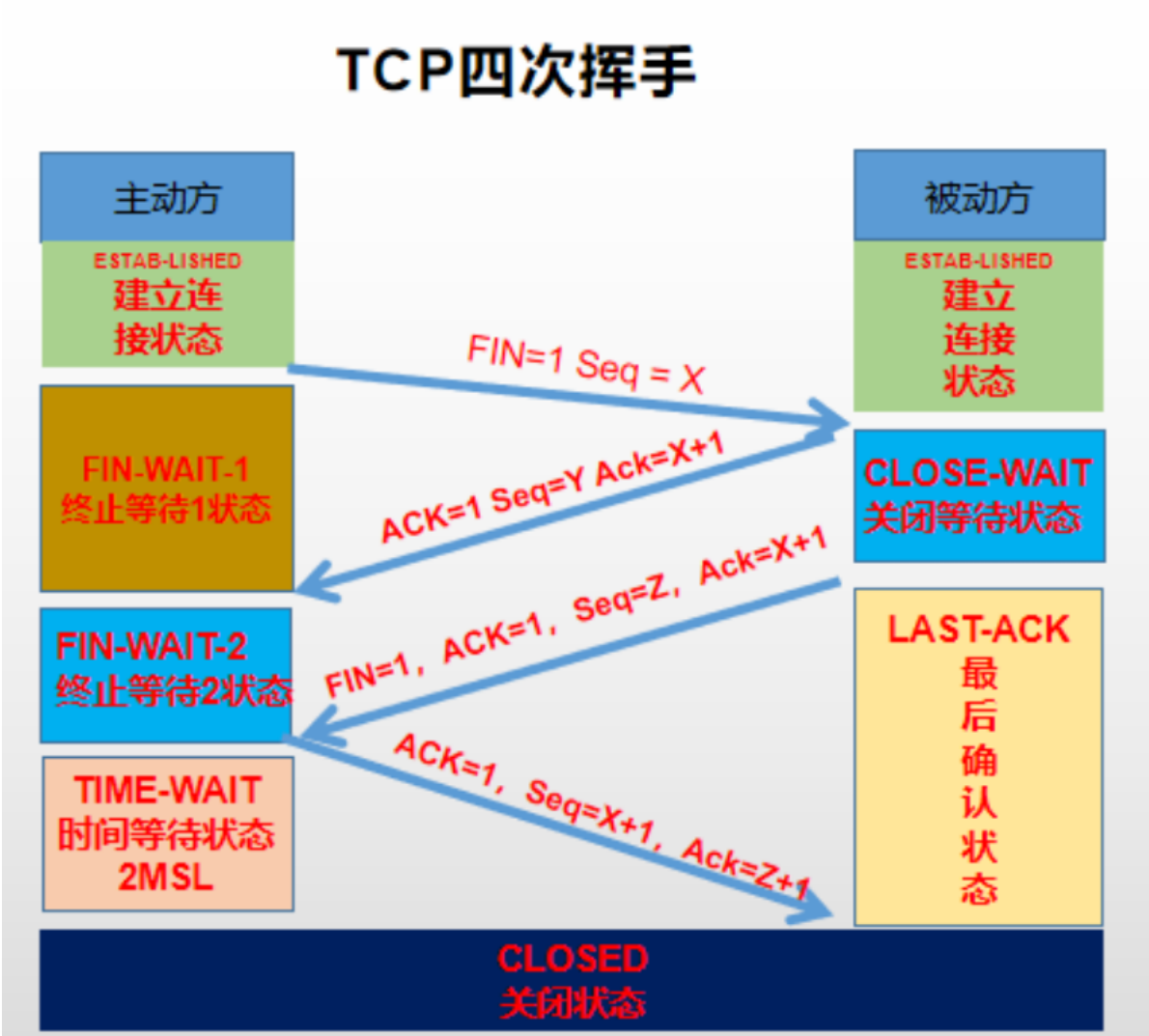
首先Client会打开Server发送过来的**Ack**验证一下是否正确为Seq+1，即第一次发送的seq number+1，确认无误后，Client仍然需要给Server再次回复确认即**ACK=1**

Seq=Z（第三次握手的数据包序列号）

Ack=Y+1

Client告诉Server，你给我回复的信息我也收到了，怎么确定我收到了你的信息呢？就是通过Ack等于第二次握手传递过来的Seq值+1。到此为止三次握手结束进入ESTABLISHED状态,开始进行数据传输。

4、TCP四次挥手



第一次挥手发送FIN请求,第一次挥手结束。

第二次挥手开始,被动方向主动方发送ACK确认码,到这里第二次挥手结束。

第三次握手开始被动方向主动方发送FIN号结束。

第四次挥手开始主动方向被动方发送ACK确认,等待**2MSL**后断开TCP连接。