题目目录

---

1. 常见的HTTP首部

- 通用首部
    - Connection：连接相关的选项
    - Date：报文的创建时间
    - Transfer-Encoding：报文的编码方式
    - Via：显示报文经过的中间节点（网关、代理）
    - Cache-Control：用于随报文传送缓存指示
- 请求
    - 关于客户端信息性首部：
        - From：用户的email地址
        - User-Agent：用户的浏览器软件（名称，版本，通常还包含操作系统的信息）
        - Referer：用于描述用户的请求是从哪个页面上的链接跳转过来的（可以用于理解用户的浏览行为，用户的兴趣）
    - Accept首部：将客户端的喜好和能力告知服务器的方式
        - Accept：支持的媒体类型
        - Accept-Charset：支持的字符集

        - Accept-Encoding：支持的编码方式
        - Accept-Language：支持的语言
    - 条件请求首部：
        - Expect：允许客户端列出某一请求 所要求 服务端的行为
        - If-Match, If-None-Match
        - If-Modified-Since，If-Unmodified-Since
    - 安全请求头部：
        - Authorization：包含客户端提供给服务器，以便对其自身进行认证的数据
        - Cookie：客户端用它向服务器传送一个令牌
- 响应首部
    - Set-Cookie：在客户端设置一个令牌，以便服务器对客户端进行标识
    - WWW-Authenticate：来自服务器的对客户端的质询列表
- 实体首部：用来描述HTTP 报文的负荷
    - 内容首部：描述了Content的类型，长度等信息
        - Content-Encoding
        - Content-Length
    - 缓存首部
        - Expires：缓存失效时间

2. 常见的HTTP请求方法和状态码

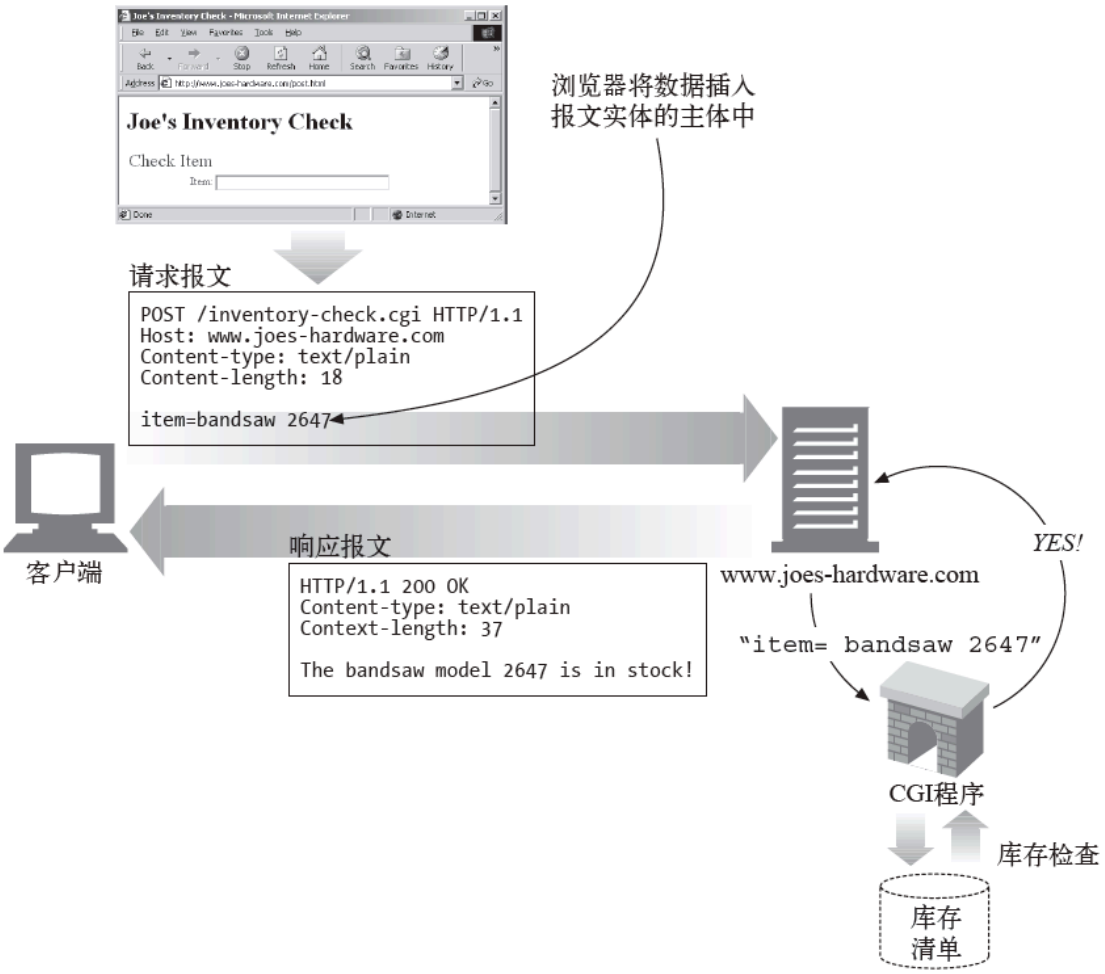
2.1 请求方法

请求方法	描述
GET	向服务端请求某一资源
POST	将客户端的数据发送到服务器网关应用程序
PUT	将客户端的数据存储到服务器资源中去
DELETE	从服务器中删除命名资源
HEAD	仅请求指定资源响应的首部

- 安全方法：GET和HEAD请求不会产生任何动作（在服务器端不会产生任何结果）
    - GET：用于请求服务器发送某个资源
    - HEAD：和GET类似，但是服务器在响应中只需要返回头部，而不返回实体，使用HEAD可以：
        - 不获取资源的情况下了解资源情况
        - 通过查看响应的状态码，查看资源是否存在
        - 通过查看首部，测试资源是否被修改

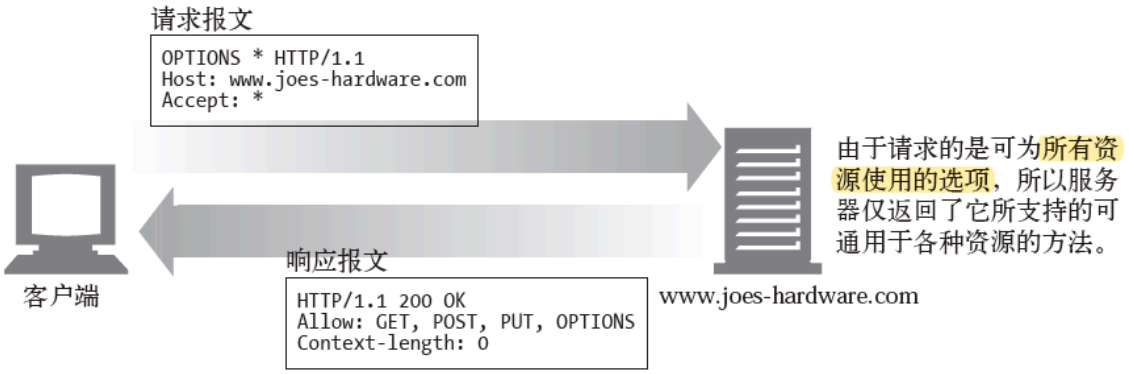
```
> telnet www.baidu.com 80
请求：
HEAD / HTTP/1.1 # 获取根路径默认主页

# 响应：
HTTP/1.1 302 Found
Connection: keep-alive
Content-Length: 17931
Content-Type: text/html
Date: Fri, 10 Jan 2020 08:32:34 GMT
Etag: "54d97487-460b"
Server: bfe/1.0.8.18
```

- PUT：向服务器写入文档（存储）
  - 一些发布系统允许用户创建web页面，并使用PUT将页面安装到web服务器上
  - PUT的语义就是：让服务器用请求的**主体部分**创建一个由**请求URL**命名的新文档
- POST：用来向服务器输入数据

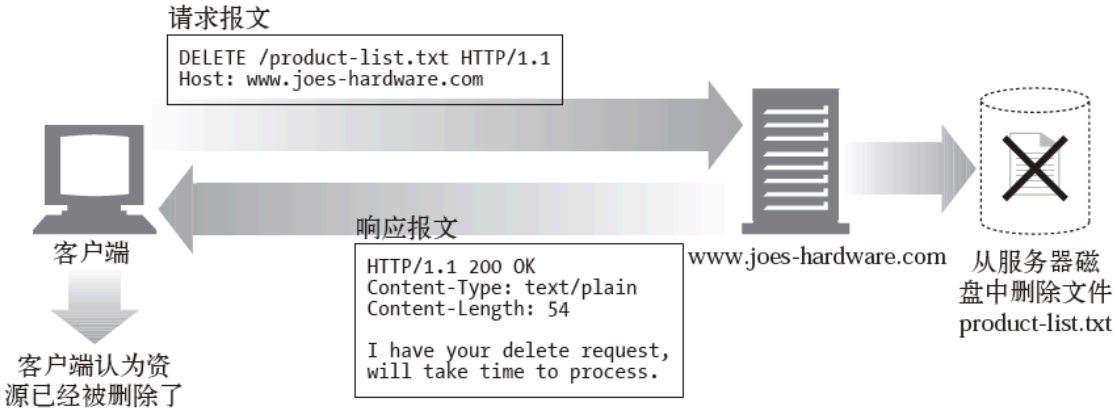


- TRACE：TRACE方法主要用于**诊断**
  - 客户端发起一个请求时，这个请求可能要穿过**防火墙、代理、网关或其他一些应用程序**。每个中间节点都可能会**修改原始的HTTP 请求**。TRACE 方法允许客户端在最终将请求发送给服务器时，看看它变成了什么样子
  - TRACE到达服务端后，服务端会回传一条Trace响应，在响应主体中携带它收到的原始请求报

文。
    - 客户端收到响应之后可以检查报文实体以诊断原始报文是否损坏或者被修改过
- OPTIONS：请求WEB服务器告知其支持的各种功能
    - 可以询问服务器通常支持哪些方法，或者对某些特殊资源支持哪些方法



- DELETE：请求服务器删除URL指定的资源，但是服务端无法保证删除一定会被执行。HTTP规范允许在不通知客户端的情况下撤销请求



2.2 状态码

- 1XX：信息提示
    - 100 当客户端向服务端发送一个较大的实体，需要在头部中添加值为100 Continue的Expect首部，服务器收到请求之后，如何支持会回复100 Continue
- 2XX：成功
    - 200 OK，请求资源正确返回，实体的主题部分包含请求的资源
    - 201 Created，用于PUT请求
- 3XX：重定向，
    - 要么告知客户端使用**替代位置**来访问他们感兴趣的资源，要么提供一个替代的响应，而**不是资源的内容**
    - 如果资源被移走，可以发送一个重定向码和可选的**Location首部**告知客户端资源被移走，以及在哪里可以找到它。因此，浏览器就可以在不打扰使用者的情况下，透明的转入新位置。
    - 301 Move Permanently，在请求的URL已经被移除时使用。响应的Location首部包含资源当前所处的URL
    - 307 Temporary Redirect，和状态码301类似，用来表示资源被移走，响应的Location首部给出的URL临时定位资源，将来的请求还是使用老的URL
    - 302 Found，重定向，到其他地方请求资源，和307的描述一样（一开始还以为看错了），主要是在HTTP1.0中使用，如果服务端接收到客户端发送来的一个POST请求，但是服务端想让

客户端重定向到其他URL，并向重定向的URL发送一个GET请求，那么就会使用302状态码
    - 303 see other，是HTTP 1.1用来实现和HTTP 1.0 中的302相同的功能
- 4XX：客户端错误
    - 400 Bad Request，客户端发送了一个错误的请求
    - 403 Forbidden，服务端拒绝了客户端的请求
    - 404 Not Found，未找到指定资源
    - 401：Unauthorized，未授权，需要用户提供用户名和密码
    - 407：代理认证时的未授权码
- 5XX：服务器错误
    - 500 Internal Server Error，服务器遇到一个妨碍它为请求提供服务的错误
    - 502 Bad Gateway，作为代理或者网关的服务器从请求链的下一条链路收到一条伪响应
    - 504 Gateway Timeout：响应来自网关或者代理，用来描述在等待另一台服务器对其进行响应时超时了。

2.3 GET 和 POST 区别

- 参考文献
- 报文格式不同
    - GET方法，发送的数据会作为URL的一部分进行请求，但是POST通过请求体来传送数据
    - 因此GET方法能发送的数据长度受限于URL的长度

2.4 HTTP 状态码4XX和5XX的区别

- 4XX是客户端错误时，服务器给客户端传送的状态码
    - 有时候客户端会给服务端发送一些服务器没法处理的东西，比如格式错误的报文，或者请求一个不存在的URL
- 而5XX是服务端自身出错时给客户端传送的状态码
    - 比如客户端遇到一个妨碍它为客户端请求提供服务的错误（500）或者代理（网关）从请求链的下一链路收到一条伪响应

3. HTTP 如何保持长连接

3.1 从HTTP如何使用TCP连接说起

- 在输入一个URL之后，浏览器首先会将域名对应的IP以及端口号解析出来
- 然后，建立一条到指定IP和端口的TCP连接
- 第三步，在该TCP连接上发送一条请求报文
- 第四步，读取响应
- 第五步，关闭连接

3.2 关于HTTP的性能的考量

- HTTP时延：HTTP的时延主要是TCP连接建立，关闭连接和传送请求和相应的时延，而HTTP事务的处理时延往往是相当短的
- 对时延影响比较大的几个点：
    - TCP连接建立握手：每次连接都要三次握手，如果只传输少量的数据，握手过程会严重降低HTTP的性能

- 延迟确认机制：在接收方回送确认时，因为确认报文很小，通常接收方会等待一段捎带数据。但是HTTP的请求应答模式降低了捎带信息的可能。比如对相应的ACK
- TCP的慢启动拥塞控制：TCP连接在一开始时会限制发送端的传输速度，在发送成功之后才会慢慢提升传输速度。由于TCP的该特性，新连接的传输速度会比那些已经交换过一些数据的连接的速度要慢。
- Nagle算法：会要求发送方尽可能的积累一定量的数据再进行发送，或者ACK到达才发送下一个segment（以提高TCP的传送效率，发送大量包含少量数据的报文会严重降低TCP的性能）
  - 但是Nagle算法会引起几种HTTP的性能问题：
  - 小的HTTP报文可能没法填满一个分组，可能会因为要等待永远不会到来的额外数据而产生时延。
  - Nagle算法要求收到确认报文才能继续发送分组，但是确认报文的发送又收到延迟确认的限制，因此，两边都拉长了报文的传输时延。

```
if there is new data to send
  if the window size >= MSS and available data is >= MSS
    send complete MSS segment now
  else
    if there is unconfirmed data still in the pipe
      enqueue data in the buffer until an acknowledge is received
    else
      send data immediately
    end if
  end if
end if
```

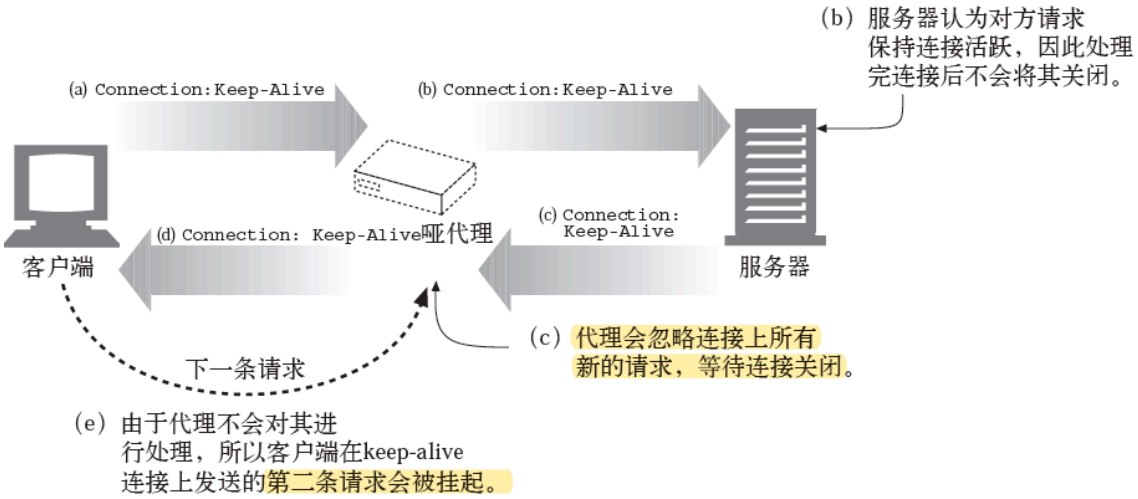
- TIME_WAIT时延和端口耗尽（主要存在于benchMark测试中，客户端的主机很少的情况下）

3.3 HTTP连接的性能优化

- 串行连接的弊端：
  - 例如请求一个包含三张嵌入图片的web页面，总共需要4次串行的请求，串行的请求大大增加了时延
- 关于HTTP连接的优化：
  - 并行连接：允许客户端打开多条连接，并行的执行多个HTTP事务
    - 但是要连接数并不是越多越好，一个是连接是需要消耗资源的，另一个是带宽就只有这么多，连接再多也没用
    - 每个事务建立一条连接会耗费更多的时间（TCP连接建立）
    - 由于TCP的慢启动特性，新连接的传输速度比已建立一段时间的连接要慢
  - 持久连接：HTTP/1.1允许HTTP 设备在事务处理结束之后将TCP连接保持在打开状态，以便为未来的HTTP 请求重用现存的连接
    - 解决了每个HTTP事务都要建立TCP连接的时延
    - 同时也避免了TCP的慢启动带来的弊端
  - 管道化连接
  - 复用连接

3.3.1 HTTP/1.0 + Keep-Alive连接

- Keep-Alive操作：
  - 客户端在发送请求时，可以在请求首部的Connection: Keep-Alive来请求将连接保持在打开状态（由于在1.0中，Keep-Alive是可选项）
  - 如果服务器支持持久连接，会在响应首部中添加相同的首部，如果响应首部中没有Connection: Keep-Alive，说明服务端不支持持久连接，在发回响应之后会关闭连接
- Keep-Alive选项：可以用于调节keep-alive的行为，只有再响应中包含Connection: Keep-Alive才能使用该选项
  - timeout参数：服务器希望保持连接的时间
  - max：服务器还能为多少个事务保持该连接
  - 例如：`Keep-Alive: timeout=120, max=5`
- Keep-Alive存在的问题：
  - Connection首部是单跳首部，只对单个TCP连接起效，对于盲转（不识别keep-alive的代理）的代理，可能会造成如下的问题：



  - 以上的问题是由于代理盲目转发Connection头部造成的，因此现代的代理都不应该转发该头部。网景公司采用了一个Proxy-Connection来解决单跳代理的问题

3.3.2 HTTP/1.1 的持久连接

- 在HTTP/1.1中主键停止了对keep-alive的支持，用持久连接的方式取代了它
- HTTP/1.1的持久连接默认是激活的，想要在事务处理结束关闭连接，必须在报文中添加：`Connection: close`首部
- 持久连接的规则：
  - 如果客户端没有更多请求时，应该在请求中附带`Connection: close`首部，然后客户端也就无法在该连接上发送更多的请求
  - 代理必须能分别管理和客户端、服务器的持久连接（持久连接是单跳的）

3.3.3 管道化连接-相对于持久化连接的又一性能优化

- 在响应到达之前，可以将更多的请求放入队列中，顺序的发送请求，可以降低网络的环回时间，提高性能。
- 限制：

- 只能在持久化连接上使用管道化请求
- 必须按照和请求相同的顺序进行响应
- 客户端必须做好连接会在任意时刻关闭的准备

3.3.4 连接的关闭

实现正常关闭的应用程序首先应该关闭它们的输出信道，然后等待连接另一端的对等实体关闭它的输出信道。当两端都告诉对方它们不会再发送任何数据（比如关闭输出信道）之后，连接就会被完全关闭，而不会有重置的危险

4. session 和 cookie

4.1 Cookie简介

- Cookie的简介可以在HTTP权威指南中查看，它和session本质上都是为了实现**客户端的识别**功能
- 服务端可以通过Cookie来获得客户端的状态信息

4.2 session 和 cookie 的区别

- 在回答该问题时，最好把cookie，session是什么，什么用途串起来
- 参考文献
- 本质上都是用于实现客户端的识别功能
  - cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，是实现session机制的一部分
  - session是在服务端保存的一个数据结构，用来跟踪用户的状态
  - cookie：将客户端的识别信息保存在本地，每次请求附带在请求的首部送给服务端
  - 而session：则是将客户端的状态信息保存在服务端，客户端每次请求只需要携带客户端到该session的映射即可（即cookie）

4.3 cookie 在传输中放在 HTTP 报文的哪里

- 保存在请求的首部里。
- 如果是响应中的set-cookie则保存在set-cookie首部中

4.4 cookie被禁的解决方案

- URL重写，将cookie中的k-v对直接添加到URL路径的后面
- 表单隐藏字段

5. HTTPS

- 最主要的参考文献是：HTTP权威指南和图解HTTP
- 参考文献

5.1 HTTPS如何建立通信

1. 首先，客户端会建立起与服务端443端口的TCP连接
2. TCP连接建立起来之后，客户端与服务端会进行SSL握手操作，协商使用的加密方式，秘钥等
   - 第一步：客户端会向服务端发送可供选择（支持的）的密码套件，并请求服务端的证书
   - 第二步：服务端会发送选中的密码套件，以及证书（数字证书中包含服务器所属机构的信息，而且证书包含了权威机构的签名（以权威机构的私钥进行了加密），客户端可以根据签

名来鉴别真伪--> 这里是因为加密函数和解密函数会相互抵消）
    - 第三步：客户端使用非对称加密将秘钥传输给服务端
    - 第四步：客户端和服务端互相告知，开始加密过程

5.2 为什么要用对称加密 key 加密信息，而不用公钥直接加密信息

- 因为对称加密的效率高，速度快
- 但是对称加密双方需要事先协商好加密的秘钥

5.3 http和https的区别

- 参考文献（很全很乱）

Https在http和tcp之间增加了一个SSL层来保证传输的安全性，防止报文被窃听、篡改和伪装。他们之间的区别主要体现在：

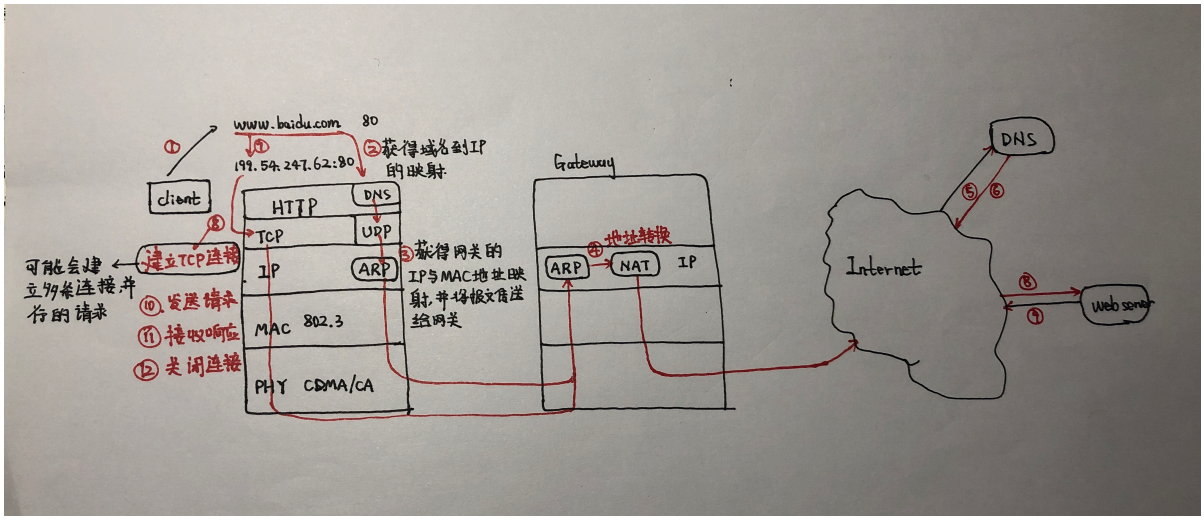
1. HTTPS相比HTTP，多一个SSL握手的过程用于协商加密参数和服务端的认证
2. HTTPS的报文都是加密的，能够防止被窃听和篡改，而HTTP则明文传输，因此HTTPS比HTTP安全
3. HTTPS在http与传输层之间增加了一个传输安全层，服务端使用的熟知端口号为443端口，而HTTP一般是以80端口为熟知端口

6. HTTP应用过程中TCP/IP四层中都应用了哪些协议

1. 应用层的DNS
2. 传输层：由于DNS一般是用的UDP，所以有用到UDP；HTTP报文的传输使用的是TCP
3. 网络层：使用IP协议，ARP协议
4. 链路层：使用多址接入协议，CSMA/CA

7. 点击 URL 的流转过程

1. 将URL的域名和端口号解析出来，如果端口号没有指定，那么http默认为80端口，https默认为443端口
2. 使用DNS协议访问DNS服务器域名对应的IP地址
3. 建立起与指定的IP和端口的TCP连接
4. 发送请求并接受响应
5. 最终将接收到的数据显示在浏览器上



8. HTTP Request Header 中有哪些内容

HTTP请求报文包含：

- 请求行
  - 格式为：请求方法 请求URL 协议版本
  - 例如：`GET /chapter1 HTTP/1.1`
- 请求头
  - 报文头包含若干个属性，格式为“属性名:属性值”，服务端据此获取客户端的信息

```
accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.9
accept-encoding: gzip, deflate, br
accept-language: zh,en-US;q=0.9,en;q=0.8,zh-CN;q=0.7
cache-control: max-age=0
user-agent: Mozilla/5.0
referer: https://www.google.com/
cookie: key=value; key1=value1;
```

- 请求体
  - 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，它承载多个请求参数的数据。不但报文体可以传递请求参数，请求URL也可以通过类似于`/chapter15/user.html? param1=value1&param2=value2`的方式传递请求参数。
- 常见的请求头解析
  - Accept：请求报文可通过一个“Accept”报文头属性告诉服务端 客户端接受什么类型的响应。
  - Cookie：服务端可以通过cookie得知客户端的连接属于哪一个Session呢
  - Referer：表示这个请求是从哪个URL过来的。比如说在google中搜索到一个广告，点击该广告，就会跳转到广告页面，在对广告页面进行请求时，就会添加Refer属性：www.google.com
  - Cache-Control：对缓存进行控制
  - Content-Length：报文长度

9. HTTP 1.1比1.0多了什么

- 优雅的谈论HTTP协议
- 参考文献2
- 参考文献3
- HTTP/2.0 强烈推荐
- HTTP1.0 Vs HTTP1.1
  - 短暂连接 Vs 持久连接
    - HTTP1.0：规定浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求
    - HTTP 1.1支持持久连接，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟
      - 一个包含有许多图像的网页文件的多个请求和应答可以在一个连接中传输，但每个

单独的网页文件的请求和应答仍然需要使用各自的连接。
        - HTTP 1.1还允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，但服务器端必须按照接收到客户端请求的**先后顺序**依次回送响应结果，以保证客户端能够区分出每次请求的响应内容，这样也显著地减少了整个下载过程所需要的时间。
    - HTTP 1.1通过增加更多的请求头和响应头来改进和扩充HTTP 1.0的功能。
        - HTTP 1.0不支持Host请求头字段，WEB浏览器无法使用主机头名来明确表示要访问服务器上的哪个WEB站点，这样就无法使用WEB服务器在同一个IP地址和端口号上配置多个虚拟WEB站点。
        - 在HTTP 1.1中增加Host请求头字段后，WEB浏览器可以使用主机头名来明确表示要访问服务器上的哪个WEB站点，这才实现了在一台WEB服务器上可以在同一个IP地址和端口号上使用不同的主机名来创建多个虚拟WEB站点。
        - HTTP 1.1还提供了与身份认证、状态管理和Cache缓存等机制相关的请求头和响应头。
        - HTTP/1.0不支持文件断点续传，HTTP/1.0每次传送文件都是从文件头开始，即0字节处开始
        - `RANGE:bytes`是HTTP/1.1新增内容，`RANGE:bytes=XXXX`表示要求服务器从文件XXXX字节处开始传送，这就是我们平时所说的断点续传！

10. 如何判断是否是长连接

- 参考文献
- 根据响应头中的：`connection:Keep-Alive`

11. HTTP/2.0新增功能

https://www.zhihu.com/question/34074946

- 连接的复用，可以在一条连接上同时进行多个请求和响应
- 通过报文的压缩来减少报文头部传输带来的开销
- 对服务器推送进行了支持
- 支持请求的优先级

> 为了实现这些功能，HTTP2.0在协议上进行多方面的增强，例如新的流控、错误处理和升级机制等。但是并没有对HTTP应用上的语义进行修改，HTTP中的核心概念：如HTTP的请求方法、HTTP状态码、资源定位符和HTTP报文头，都没有改变。
>
> 在协议上的增强体现在：HTTP2.0改变了客户端和服务端数据的成帧和传输方式。但是这些都被封装到一个新的成帧层，因此对现有的应用不会产生任何的影响。
>
> 除非你是在原生的TCP socket上进行开发，否则HTTP2.0和HTTP1.0在开发上没有任何差别，你能看到的差别只有更高的性能和一些fancy的新功能