http 是无状态的

RESTful 接口中，利用 HTTP 协议的 method 字段来描述要对资源操作的方式，比如 GET 表示获取资源，POST 表示新增一个资源，PUT 表示更新资源,DELETE 表示删除资源等等。

1. http 请求和响应的消息结构:

两者区别: 请求消息有请求行, 响应消息有状态行

url 在请求行, cookie 在请求头

一个HTTP请求一般由四部分组成：

1. **HTTP请求的方法或动作，比如是GET还是POST请求**
2. **正在请求的URL，总得知道请求的地址是什么吧**
3. **请求头，包含一些客户端环境信息，身份验证信息等**
4. **请求体，也就是请求正文，请求正文中可以包含客户提交的查询字符串信息，表单信息等等**

- **一个HTTP响应一般由三部分组成：**

1. **一个数字和文字组成的状态码，用来显示请求是成功还是失败**
2. **响应头，响应头也和请求头一样包含许多有用的信息，例如服务器类型、日期时间、内容类型和长度等**
3. **响应体，也就是响应正文**

2.HTTP 请求过程步骤

一个完整的HTTP请求过程，通常有下面7个步骤：

1. **建立TCP连接**
2. **Web浏览器向Web服务器发送请求命令**
3. **Web浏览器发送请求头信息**
4. **Web服务器应答**
5. **Web服务器发送应答头信息**
6. **Web服务器向浏览器发送数据**
7. **Web服务器关闭TCP连接**

3. http 请求头和响应头有哪些字段

说一说常见的请求头和相应头都有什么呢？

1)请求(客户端->服务端[request])

GET(请求的方式) /newcoder/hello.html(请求的目标资源) HTTP/1.1(请求采用的协议和版本号)
Accept: */*(客户端能接收的资源类型)
Accept-Language: en-us(客户端接收的语言类型)
Connection: Keep-Alive(维护客户端和服务端的连接关系)
Host: localhost:8080(连接的目标主机和端口号)
Referer: http://localhost/links.asp(告诉服务器我来自于哪里)
User-Agent: Mozilla/4.0(客户端版本号的名字)
Accept-Encoding: gzip, deflate(客户端能接收的压缩数据的类型)
If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT(缓存时间)
Cookie(客户端暂存服务端的信息)
Date: Tue, 11 Jul 2000 18:23:51 GMT(客户端请求服务端的时间)

2)响应(服务端->客户端[response])

HTTP/1.1(响应采用的协议和版本号) 200(状态码) OK(描述信息)
Location: http://www.baidu.com(服务端需要客户端访问的页面路径)
Server:apache tomcat(服务端的 Web 服务端名)
Content-Encoding: gzip(服务端能够发送压缩编码类型)

Content-Length: 80(服务端发送的压缩数据的长度)
Content-Language: zh-cn(服务端发送的语言类型)
Content-Type: text/html; charset=GB2312(服务端发送的类型及采用的编码方式)
Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT(服务端对该资源最后修改的时间)
Refresh: 1;url=http://www.it315.org(服务端要求客户端 1 秒钟后，刷新，然后访问指定的页面路径)
Content-Disposition: attachment; filename=aaa.zip(服务端要求客户端以下载文件的方式打开该文件)
Transfer-Encoding: chunked(分块传递数据到客户端)
Set-Cookie:SS=Q0=5Lb_nQ; path=/search(服务端发送到客户端的暂存数据)
Expires: -1//3 种(服务端禁止客户端缓存页面数据)
Cache-Control: no-cache(服务端禁止客户端缓存页面数据)
Pragma: no-cache(服务端禁止客户端缓存页面数据)
Connection: close(1.0)/(1.1)Keep-Alive(维护客户端和服务端的连接关系)
Date: Tue, 11 Jul 2000 18:23:51 GMT(服务端响应客户端的时间)
在服务器响应客户端的时候，带上 Access-Control-Allow-Origin 头信息，解决跨域的一种方法。

答:
http 请求和 http 响应都使用头发送有关 http 消息的信息. 头由一系列行组成, 每行都包含名称, 然后依次是冒号, 空格, 值. 字段可按任何顺序排列. 某些头字段既可以用于请求头也可以用于响应头, 而另一些字段只能使用其中之一.
许多请求字段都允许客户端在值部分指定多个可接收的选项, 有时甚至可以对这些选项的首选项进行排名. 多个项以逗号分隔. 例如, 客户端可以发送包含"Content-Encoding: gzip, compress"的请求头, 表示可以接受各种压缩类型. 如果服务器的响应正文使用 gzip 编码,其响应头中将包含"Content-Encoding: gzip".
有些字段可以在单个头中出现多次, 例如, 头可以有多个"Warning"字段.
下表列出了 http1.1 头字段. 注意, 有些头字段是 mime 字段. mime 字段在 ietf 文档 rfc2045 中进行了定义, 但也可用于 http1.1 协议.
一般头字段: 一般头字段可用于请求消息和响应消息
Ps：与缓存相关的头

Cache-Control	指定请求和响应遵循的缓存机制. 请求消息或响应消息中设置 Cache-Control 并不会修改另一个消息处理过程只能怪的缓存处理过程	"max-age=10" or "no-cache" or "no-store"
Connection	处理完这次请求后是否断开连接还是继续保持连接	"close" or "Keep-Alive"
Date	表示消息发送的时间. 时间的描述格式由 rfc822 定义	"Tue, 11 Jul 2000 18:23:51 GMT"
Pragma	用来包含实现特殊的指令 知道"no-cache"值表示服务器必须返回一个刷新后的文档, 即使它是代理服务器而且已经有了页面的本地拷贝	"no-cache"(与设置 Cache-Control: no-cache 相同)
Trailer		"Date"
Transfer-Encoding		"chuncked"
Upgrade	向服务器指定某种传输协议以便服务器进行转换(如果支持)	"SHTTP/1.3"
Via	通知中间网关或代理服务器地址, 通信协议	"HTTP/1.1 Proxy1, HTTP/1.1 Proxy2"
Warning	关于实体消息的警告细心	"112 Disconnected Operation"

请求字段头:
请求消息的第一行格式为:

1	Method SP Request-URI SP HTTP-Version CRLF

Method表示对Request-URI完成的方法，这个字段是大小写敏感的，包括 OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE
GET 方法取回 Request-URI 标识的信息,
HEAD 方法也是取回由 Request-URI 标识的信息，只是可以在响应时不返回消息体,
POST 方法可以请求服务器接收包含在请求中的实体消息，可以用于提交表单，向新闻组, BBS，邮件群组和数据库发送消息.
SP 表示空格
Request-URI 遵循 URI 格式,在此字段为*时，说明请求并不用于某个特定的资源地址，而是用于服务器本身.
HTTP-Version 表示支持的 http 版本，例如 HTTP/1.1
CRLF 表示换行符
请求头域：允许客户端向服务器传递关于请求或者关于客户机的附加信息.

Accept	浏览器能够处理的内容类型.	"text/html, iamge/*"
Accept-Charset	告诉服务器，客户端采用的编码格式/字符集	"iso8859-5"
Accept-Encoding	告诉服务器，客户机支持的数据压缩格式	"gzip, compress"
Accept-Language	客户机的语言环境	"en, fr"
Authorization	授权信息.，通常出现在对服务器发送的 WWW-Authenticate 头的应答中	[credentials]
Content-Encoding		"gzip"
Expect		"100-continue"
From	请求发送者的 email 地址，由一些特殊的 web 客户程序使用，浏览器不会用到	"user@microsoft.com"
Host	客户机想访问的主机名. 即指定资源的 internet 主机和端口号. 必须表示请求 url 的原始服务器或网关的位置, http/1.1 请求必须包含主机头域，否则系统会以 400 状态码返回	"www.microsoft.com"
If-Match		"entity_tag001"
If-Modified-Since	资源的缓存时间. 只有当所请求的内容在指定的日期之后又经过修改才返回它，否则返回 304 "Not Modified" 应答	"Tue, 11 Jul 2000 18:12:12 GMT"
If-None=Match		"entity_tag001"
If-Range		"entity_tag001" or "Tue, 11 Jul 2000 18:12:12 GMT"
If-Unmodified-Since		"Tue, 11 Jul 2000 18:12:12 GMT"
Max-Forwards		"3"
Proxy-Authorization		[credentials]
Range	请求实体的一个或者多个子范围. 如示例即表示请求 100-599 个字节.	"bytes=100-599"

	但是服务器可以忽略此请求头，如果无条件 get 包含 range 请求头，响应会以状态码 206 返回而不是 200	
Referer	告诉服务器，客户端是从哪个资源来访问服务器的(防盗链)	"http://www.microsoft.com/resources.asp"
TE	客户端愿意接受的传输编码，并通知服务器接受接受尾加头信息	"trailers"
User-Agent	客户机的软件环境，浏览器类型	"Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)"

响应头字段:

响应消息的第一行为下面的格式

```
1 HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

Accecpt-Ranges	表面服务器是否支持指定范围请求及哪种类型的分段请求	"none"
Age	从原始服务器到代理缓存形成的估算时间(以秒记，非负)	"2147483645(2^31)"
ETag	缓存相关的头	"b38b9-17dd-367c5dcd"
Last-Modified	请求资源的最后修改时间	"Tue, 11 Jul 2000 18:23:51 GMT"
Location	配合 302 状态码使用，用于告诉客户找谁	"http://localhost/redirecttarget.asp"
Proxy-Authenticate	指出认证方案和可应用到代理的该 url 上的参数	[challenge] Proxy-Authenticate: Basic
Retry-After		"Tue, 11 Jul 2000 18:23:51 GMT" or "60"
Server	服务器通过这个告诉浏览器数据的服务器的类型	"Microsoft-IIS/5.0"
Vary		"Date"
WWW-Authenticate		[challenge]

实体头字段

请求消息和响应消息都可以包含实体信息．实体信息一般是由实体头域和实体组成．实体头域包含关于实体的原信息.实体可以是一个经过编码的字节流．其编码方式由 Content-Encoding 和 content-Type 定义．长度由 Content-Length 或 Content-Range 定义.

实体头字段:实体头字段可以用于请求消息或响应消息．实体头字段中包含消息实体正文的有关信息，如使用的编码格式

Allow		"GET, HEAD"
Content-Encoding	数据的压缩格式	"gzip"
Content-Language		"en"
Content-Length	请求消息正文的长度	"8445"
Content-Location		"http://localhost/page.asp"
Content-MD5		[md5-digest]
Content-	用于指定整个实体中的一部分的插入位置，也指示了整个实体	"bytes 2543-4532/7898"

Range	的长度. 在服务器向客户返回一个部分响应,它必须描述响应覆盖的范围和整个实体长度	
Content-Type	数据的类型. 指定 head 方法送到接收方的实体介质类型, 或 get 方法发送的请求介质类型 Content-Range 实体头	"text/html"
Expires	告诉浏览器把回送的资源缓存多长时间 -1 或 0 则是不缓存. 即应该在什么时候认为文档已经过期, 从而不再缓存它	"Tue, 11 Jul 2000 18:12:12 GMT"
Last-Modified	当前资源的最后缓存时间. 即服务器上保存内容的最后修订时间. 客户可以通过 If-Modified-Since 请求头提供一个日期, 该请求将被视为一个条件 get, 只有改动时间迟于指定时间的文档才会返回, 否则返回一个 304(Not Modified)状态	"Tue, 11 Jul 2000 18:12:12 GMT"

实体内容:
代表服务器向客户端回送的数据
请求头实例:

GET /articles/news/today.asp HTTP/1.1 Accept: */* Accept-Language: en-us Connection: Keep-Alive Host: localhost Referer: http://localhost/links.asp User-Agent: Mozilla/4.0 (compatible; MSIE 3.5; Windows NT 5.0) Accept-Encoding: gzip, deflate

该请求具有请求行, 其中包括方法(GET), 资源路径(/articles/news/today.asp)和 http 版本(http/1.1). 由于该请求没有正文, 故所有请求行后面的内容都是头的一部分. 紧接着头之后是一个空行, 表示头已结束.
响应头实例
Web 服务器可以通过多种方式响应前一个请求, 假设文件是可以访问的, 并且用户具有查看该文件的权限, 则响应类似于:

HTTP/1.1 200 OK Server: Microsoft-IIS/5.0 Date: Thu, 13 Jul 2000 05:34:32 GMT Content-Length: 2291 Content-Type: text/html Set-Cookie: ASPSESSIONIDQQGGGNCG=LKLDFFKCINFLDMFHCBCBMFLJ; path=/ Cache-control: private <HTML> <BODY> ...

响应的第一行称为状态行. 它包含响应所用的 http 版本, 状态编码(200)和原因短语. 示例中包含一个头, 其中具有五个字段, 接着是一个空行(回车和换行符), 然后是响应正文的头两行.
参考: HTTP 头参考 HTTP 协议---HTTP 请求中的常用请求字段和 HTTP 的响应状态码及响应头 HTTP 头字段
HTTP 响应头和请求头信息对照表
请求头与请求体

4. http 响应常见状态码

100-199 : 表示成功接收请求, 要求客户端继续提交下一次请求才能完成整个处理过程
200-299: 表示成果接收请求并已完成整个处理过程. 常用 200
300-399: 为完成请求, 客户需进一步细化需求: 例如: 请求的资源已经移动一个新地址, 常用 302(重定向), 307 和 304(拿缓存)
400-499: 客户端的请求有错误, 包含语法错误或者不能正确执行. 常用 404(请求的资源在 web 服务器中没有) 403(服务器拒绝访问, 权限不够)
500-599: 服务器端出现错误

常用:
200 正常 表示一切正常, 返回的是正常请求结果
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 未修改 表示客户机缓存的版本是最新的, 客户机应该继续使用它
403 禁止 服务器理解客户端请求, 但拒绝处理它, 通常用于服务器上文件或目录的权限设置所致
404 找不到 服务器上不存在客户机所请求的资源

500　　服务器内部错误　　服务器端的 cgi, asp, jsp 等程序发生错误

304 缓存的原理
服务器首先产生 ETag，服务器可在稍后使用它来判断页面是否已经被修改。本质上，客户端通过将该记号传回服务器要求服务器验证其（客户端）缓存。
304 是 HTTP 状态码，服务器用来标识这个文件没修改，不返回内容，浏览器在接收到个状态码后，会使用浏览器已缓存的文件
客户端请求一个页面（A）。 服务器返回页面 A，并在给 A 加上一个 ETag。 客户端展现该页面，并将页面连同 ETag 一起缓存。 客户再次请求页面 A，并将上次请求时服务器返回的 ETag 一起传递给服务器。 服务器检查该 ETag，并判断出该页面自上次客户端请求之后还未被修改，直接返回响应 304（未修改——Not Modified）和一个空的响应体。

5. 简述 http 1.1 与 http 1.0 的区别
答:
http 1.0 对于每个连接都得建立一次连接，一次只能传送一个请求和响应，请求就会关闭, http1.0 没有 Host 字段
而 http1.1 在同一个连接中可以传送多个请求和响应，多个请求可以重叠和同时进行, http1.1 必须有 host 字段
http1.1 中引入了 ETag 头，它的值 entity tag 可以用来唯一的描述一个资源. 请求消息中可以使用 If-None-Match 头域来匹配资源的 entitytag 是否有变化
http1.1 新增了 Cache-Control 头域(消息请求和响应请求都可以使用)，它支持一个可扩展的指令子集
http1.0 中只定义了 16 个状态响应码，对错误或警告的提示不够具体. http1.1 引入了一个 Warning 头域，增加对错误或警告信息的描述. 且新增了 24 个状态响应码
参考: HTTP/1.1 与 HTTP/1.0 的区别

6. 请列举三种禁止浏览器缓存的头字段，并写出相应的设置值
答:
Expires: 告诉浏览器把回送的资源缓存多长时间　-1 或 0 则是不缓存
Cache-Control: no-cache
Pragma: no-cache

7. 和客户端浏览器缓存相关的 http 头
Expires: +过期时间
表示在指定时间后浏览器缓存失效
这里的过期时间必须是 http 格式的日期时间，其他都会被解析成当前时间"之前"，缓存会马上过期. http 的日期时间必须是格林威治时间(GMT)，而不是本地时间
e.g.　Fri, 30 Oct 2009 13:13:13
使用 Expires 过期必须要求服务器的时间是正确的,否则发送的 http 头就会出问题，在 windows 服务下可以设置时间服务器来同步时间
Cache-control: 缓存控制
控制缓存
值分为:
max-age=[秒]: 执行缓存被认为是最新的最长时间. 类似于过期时间，这个参数是基于请求时间的相对时间间隔，而不是绝对过期时间, [秒]是一个数组，单位是秒: 从请求时间到过期时间之间的秒数
s-maxage=[秒]: 类似于 max-age 属性，除了他应用于共享(如: 代理服务器)换粗
public: 仅体现在响应头. 通知浏览器可以无条件地缓存该响应. 标记认证内容也可以被缓存. 一般来说，经过 http 认证才能访问的内容，输出是自动不可以缓存的
private: 仅体现在响应头，通知浏览器只针对单个用户缓存响应，且可以具体指定某个字段，如 private-"username"
no-cache: 强制每次请求之间发送给源服务器，而不经过本地缓存版本的校验. 这对于需要确认认证应用很有用(可以和 public 结合使用)，或者严格要求使用最新数据的应用(不惜牺牲使用缓存的所有好处)
请求头中: 告诉浏览器回去服务器取数据，并验证你的缓存(如果有的话)

响应头中：告诉浏览器，一定要回服务器校验，不管有没有缓存数据．如果确定没有修改，可以使用缓存中的数据
no-store：告诉浏览器任何情况下都不要被缓存
must-revalidate：告诉浏览器必须遵循所有你给予副本的新鲜度的，http 允许缓存在某些特定情况下返回过期数据，指定了这个属性，你告诉缓存，你希望严格的遵循你的规则
proxy-revalidate：和 must0revalidate 类似，除了他只对缓存代理服务器起作用
e.g.　Cache-Control: max-age=3600, must-revalidate
Last-Modifield / If-Modified-Since(/If-Unmodified-Since)
一对
Last-Modified 表示某个地址的最近更新时间，是服务器端响应给客户端的
If-(Un)Modified-Since 是客户端浏览器发送给服务器的，告诉 web 服务器客户端有一个最后更改时间为什么时间的缓存，服务器接收到 If-Modified-Since 头后则判断客户端缓存的这份 url 地址的缓存是否是最新的，如果是最新的则服务器直接给客户端返回 HttpStatus 304，如果服务器发现 url 的最后更新时间比 If-Modified-Since 的值要新，则会输出新的内容
ETag/If-None-Match
ETag 和 Last-Modified 类似，不过他发送的是一个字符串来标识 url 的版本，如果 url 变了则此标识也跟着变化，在浏览器发送 If\-Modified-Match 时告诉浏览器内容已经变了，或者没变可以使用缓存
list 会自动给静态文件加上 Etag，在文件发生改变时重新生成一个 ETag，这样对于一个网站中的 n 多个静态文件，如样式表，小图片等，客户端只需要下载一次就够了，可以减轻负载
Pragma: no-cache
是 http1.0 中的常规头，作用同 http1.1 的 Cache-Control: no-cache
关于以上缓存机制的优先级:
Cache-Control > Expires：前者设置更详细
Cache-Control/Expires > Last-Modified/ETag：本地副本根据 Cache-Control/Expires 还在有效期内，则不会在此发送请求去服务器询问修改时间或实体标识了
即最前面的最重要，前面的生效后，后面的基本就失效了
ETag 默认是需要要源网站确认的，因为要确认是否匹配．而 Last-Modified 默认是不向源服务器确认的
在大型多 web 集群时，使用 ETag 有问题．因为多服务器时 INode 不一样，所以不同的服务器生成的 ETag 不一样，所以用户有可能重复下载(这时 ETag 就会不准).
如果服务器端同时设置了 ETag 和 Expires 时，ETag 原理同样，即与 Last-Modified/ETag 对应的 HttpRequest Header: If-Modified-Since 和 If-None-Match．则在完全匹配 If-Modified-Since 和 If-None-Match 即检查完修改时间和 ETag 后，服务器才能返回 304;
如果服务器又设置了 Cache-Control:max-age 和 Expires 时，也是同时使用.(到底是同时使用还是如上所述前者大于后者?)
一般情况下，使用 Cache-Control/ Expires 会配合 Last-Modified/ETag 一起使用，因为即使在服务器设置缓存时间，当用户点击"刷新"按钮时，浏览器会忽略缓存继续向服务器发送请求，这是后者就可以很好利用 304，从而减少响应开销
ETag 是服务器自动生成或者或者由开发者生成的对应资源在服务器端的唯一标识符，能够更加准确的控制缓存．Last-Modified 和 ETag 是可以一起使用的，服务器会优先验证 ETag，一致的情况下，才会继续比对 Last-Modified，最后才决定返回 304.
用户操作和缓存的关系

用户操作	Cache-Control/Expires	Last-Modified/ETag
地址栏回车	有效	有效
页面链接调整	有效	有效
新开窗口	有效	有效
前进后退	有效	有效
F5 刷新	无效	有效
Ctrl+F5	无效	无效

参考: 有关客户端浏览器缓存的 Http 头介绍　HTTP 缓存相关的概念 http 请求头信息 http 响应头信息　expires 与

etag 控制页面缓存的优先级

8. Get 和 post 的区别

答:
get 请求一般用于向服务器查询某些信息，post 请求通常用于向服务器发送应该被保存的数据. 即: get 是从服务器上获取数据，post 是向服务器传送数据
get 请求可以将查询字符串参数追加到 url 的末尾; post 请求应该把数据作为请求的主体提交. 其请求主体可以包含非常多的数据，而且格式不限
因为 get 请求提交的数据直接加载 url 末尾,所以其大小有限制; 理论来讲, post 是没有大小限制的.
post 安全性比 get 要高
对于 get 方式，服务器端用 Request.QueryString 获取变量的值，对于 post 方式，服务器端用 Request.Form 获取提交的数据

get 是把参数数据队列加到提交表单的 ACTION 属性所指的 URL 中，值和表单内各个字段一一对应，在 URL 中可以看到。post 是通过 HTTP post 机制，将表单内各个字段与其内容放置在 HTML HEADER 内一起传送到 ACTION 属性所指的 URL 地址。用户看不到这个过程
get 形式的 url 对搜索引擎更加友好，可以提高搜索引擎排名。Post 使用的 url 有时候会阻止爬虫和搜索引擎的访问。其他网站和用户可以链接到 get 形式的 url，无论用户的访问，还是搜索引擎的收录而相应提高了页面排名，能够直接或间接提高网站浏览。同时，get 形式的 url 这种表示法是可以缓存的，显著提升了客户端和服务端的性能。而不安全操作，如确定订购、下订单、达成协议和删除页面等，应该通过 post 执行，避免没有显式用户请求和同一的情况下发生意外的操作。例如搜索引擎删除整个页面，只因为抓取了一个链接。很多不希望用户浏览器遵循页面链接的各种完整，这些情况下，应该要求用户登录并且足够的权限才能执行某些危险操作。

若符合下列任一情况，则用 POST 方法：
* 请求的结果有持续性的副作用，例如，数据库内添加新的数据行。
* 若使用 GET 方法，则表单上收集的数据可能让 URL 过长。
* 要传送的数据不是采用 7 位的 ASCII 编码。
若符合下列任一情况，则用 GET 方法：
* 请求是为了查找资源，HTML 表单数据仅用来帮助搜索。
* 请求结果无持续性的副作用。
* 收集的数据及 HTML 表单内的输入字段名称的总长不超过 1024 个字符。

9. 三次握手、四次挥手

答: 建立 TCP 连接需要三次握手，而断开连接需要执行四次挥手.
三次握手:
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。
第一步: 客户机的 TCP 先向服务器的 TCP 发送一个连接请求报文. 这个特殊的报文中不含应用层数据，其首部中的 SYN 标志位被置 1. 另外，客户机会随机选择一个起始序号 seq=x(连接请求报文不携带数据,但要消耗掉一个序号)
第二步: 服务器端的 TCP 收到连接请求报文后，若同意建立连接，就向客户机发送请求，并为该 TCP 连接分配 TCP 缓存和变量. 在确认报文中,SYN 和 ACK 位都被置为 1，确认号字段的值为 x+1，并且服务器随机产生起始序号 seq=y(确认报文不携带数据，但也要消耗掉一个序号). 确认报文同样不包含应用层数据.
第三步: 当客户机收到确认报文后，还要向服务器给出确认，并且也要给该连接分配缓存和变量. 这个报文的 ACK 标志位被置为 1，序号字段为 x+1，确认号字段为 y+1
四次挥手
第一步: 客户机打算关闭连接,就向其 TCP 发送一个连接释放报文,并停止再发送数据,主动关闭 TCP 连接，该报文的 FIN 标志位被置 1, seq=u，它等于前面已经传送过的数据的最后一个字节的序号加 1(FIN 报文即使不携带数据,也要消耗掉一个序号)
第二步: 服务器接收连接释放报文后即发出确认，确认号是 ack=u+1，这个报文自己的序号是 v，等于它前面已传送

过的数据的最后一个自己的序号加 1. 此时，从客户机到服务器这个方向的连接就释放了，TCP 连接处于半关闭状态. 但服务器若发送数据，客户机仍要接收，即从服务器到客户机的连接仍未关闭.
第三步: 若服务器已经没有了要向客户机发送的数据，就通知 TCP 释放连接，此时其发出 FIN=1 的连接释放报文
第四步: 客户机收到连接释放报文后，必须发出确认. 在确认报文中，ACK 字段被置为 1，确认号 ack=w+1，序号 seq=u+1. 此时，TCP 连接还没有释放掉，必须经过等待计时器设置的时间 2MSL 后，A 才进入到连接关闭状态.

10. tcp/ip / http 对应哪一层 七层模型

TCP/IP 五层协议: 应用层、传输层、网络层、数据链路层、物理层. http 对应应用层
ISO 七层模型
物理层：通过媒介传输比特,确定机械及电气规范（比特 Bit）
数据链路层：将比特组装成帧和点到点的传递（帧 Frame）
网络层：负责数据包从源到宿的传递和网际互连（包 PackeT） （IP）
传输层：提供端到端的可靠报文传递和错误恢复（段 Segment） （TCP 和 UDP）
会话层：建立、管理和终止会话（会话协议数据单元 SPDU）
表示层：对数据进行翻译、加密和压缩（表示协议数据单元 PPDU）
应用层：允许访问 OSI 环境的手段（应用协议数据单元 APDU）（HTTP、FTP、SMTP、DNS）
各种协议
ICMP 协议： 因特网控制报文协议。它是 TCP/IP 协议族的一个子协议，用于在 IP 主机、路由器之间传递控制消息。
TFTP 协议： 是 TCP/IP 协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。
HTTP 协议： 超文本传输协议，是一个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。
DHCP 协议： 动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。

OSI 层	功能	TCP/IP 协议
应用层(Application layer)	文件传输，电子邮件，文件服务，虚拟终端	TFTP，HTTP，SNMP，FTP，SMTP，DNS，Telnet
表示层(Presentation layer)	数据格式化，代码转换，数据加密	没有协议
会话层(Session layer)	解除或建立与其他接点的联系	没有协议
传输层(Transport layer)	提供端对端的接口	TCP，UDP
网络层(Network layer)	为数据包选择路由	IP，ICMP，RIP，OSPF，BGP，IGMP
数据链路层(Data link layer)	传输有地址的帧，错误检测功能	SLIP，CSLIP，PPP，ARP，RARP，MTU
物理层(Physical layer)	以二进制数据形式在物理媒体上传输数据	ISO2110，IEEE802，IEEE802.2

TCP 传输慢但是可靠，UDP 传输快但是不可靠。
Socket 可以使用 TCP/IP 协议或 UDP 协议。
TCP/IP 协议
TCP/IP 协议是目前应用最为广泛的协议，是构成 Internet 国际互联网协议的最为基础的协议,由 TCP 和 IP 协议组成:
TCP 协议:面向连接的、可靠的、基于字节流的传输层通信协议，负责数据的可靠性传输的问题。
IP 协议:用于报文交换网络的一种面向数据的协议，主要负责给每台网络设备一个网络地址，保证数据传输到正确的目的地。
UDP 协议
UDP 特点：无连接、不可靠、基于报文的传输层协议，优点是发送后不用管，速度比 TCP 快。

TCP/IP 层	网络设备
应用层	
传输层	四层交换机、也有工作在四层的路由器
网络层	路由器、三层交换机
数据链路层	网桥（现已很少使用）、以太网交换机（二层交换机）、网卡（其实网卡是一半工作在物理层、一半工作在数据链路层）
物理层	中继器、集线器、还有我们通常说的双绞线也工作在物理层

11. 浏览器中输入网址后到页面展现的过程

1）用户输入网址

2）浏览器通过 DNS 获取网站的 IP 地址。客户端先检查本地是否有对应的 IP 地址，若找到则返回响应的 IP 地址。若没找到则请求上级 DNS 服务器，直至找到或到根节点。

DNS 查找 IP 地址的顺序: 浏览器缓存、系统缓存、互联网服务提供商（ISP）的 DNS 缓存、递归搜索（从浏览器缓存开始，如果没找到就继续往下一个找。）找到后，浏览器会获得一个 IP 地址。

3）浏览器客户端发送 http 请求

HTTP 请求包括请求报头和请求主体两个部分，其中请求报头包含了至关重要的信息，包括请求的方法（GET / POST）、目标 url、遵循的协议（http / https / ftp…），返回的信息是否需要缓存，以及客户端是否发送 cookie 等。

4）传输层 TCP 传输报文。TCP 协议通过“三次握手”等方法保证传输的安全可靠。

5）网络层 IP 协议查询 MAC 地址

IP 协议的作用是把 TCP 分割好的各种数据包传送给接收方。而要保证确实能传到接收方还需要接收方的 MAC 地址，也就是物理地址。IP 地址和 MAC 地址是一一对应的关系，一个网络设备的 IP 地址可以更换，但是 MAC 地址一般是固定不变的。ARP 协议可以将 IP 地址解析成对应的 MAC 地址。当通信的双方不在同一个局域网时，需要多次中转才能到达最终的目标，在中转的过程中需要通过下一个中转站的 MAC 地址来搜索下一个中转目标。

7）数据到达数据链路层

在找到对方的 MAC 地址后，就将数据发送到数据链路层传输。这时，客户端发送请求的阶段结束

8）服务器接收数据

接收端的服务器在链路层接收到数据包，再层层向上直到应用层。这过程中包括在运输层通过 TCP 协议将分段的数据包重新组成原来的 HTTP 请求报文。

9）服务器响应请求

服务接收到客户端发送的 HTTP 请求后，查找客户端请求的资源，并返回响应报文，响应报文中包括一个重要的信息——状态码。状态码由三位数字组成，其中比较常见的是 200 OK 表示请求成功。301 表示永久重定向，即请求的资源已经永久转移到新的位置。在返回 301 状态码的同时，响应报文也会附带重定向的 url，客户端接收到后将 http 请求的 url 做相应的改变再重新发送。404 not found 表示客户端请求的资源找不到。

10）服务器返回响应文件

请求成功后，服务器会返回相应的 HTML 文件。接下来就到了页面的渲染阶段了。

11） 页面渲染： 解析 HTML 以构建 DOM 树 –> 构建渲染树 –> 布局渲染树 –> 绘制渲染树。

关于页面渲染过程：

1）解析 HTML 代码，生成一棵 DOM 树

2）解析 CSS 文件

3）生成渲染树（受样式影响，包含不可见元素）

4）渲染树中的节点

12、TCP 和 UDP 的区别

TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对

方建立可靠的连接。一个 TCP 连接必须要经过三次“对话”才能建立起来
UDP（User Data Protocol，用户数据报协议）是与 TCP 相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！ UDP 适用于一次只传送少量数据、对可靠性要求不高的应用环境。

13.HTTP 和 HTTPS
HTTP 协议通常承载于 TCP 协议之上，在 HTTP 和 TCP 之间添加一个安全协议层（SSL 或 TSL），这个时候，就成了我们常说的 HTTPS。
默认 HTTP 的端口号为 80，HTTPS 的端口号为 443。
为什么 HTTPS 安全
因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用 HTTPS，密钥在你和终点站才有。https 之所以比 http 安全，是因为他利用 ssl/tls 协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer 传递等。保障了传输过程的安全性

14.Http 2.0
HTTP/2 引入了“服务端推（server push）”的概念，它允许服务端在客户端需要数据之前就主动地将数据发送到客户端缓存中，从而提高性能。
HTTP/2 提供更多的加密支持
HTTP/2 使用多路技术，允许多个消息在一个连接上同时交差。
它增加了头压缩（header compression），因此即使非常小的请求，其请求和响应的 header 都只会占用很小比例的带宽。