扩展从输入URL到页面展示，发生了什么?

1、首先从本地查找域名，有的话直接用hosts文件里的ip地址，否则查询DNS，得到ip地址
2、建立TCP连接——进行“三次握手”
3、客户端发送http请求
4、服务端处理，并返回结果给客户端
5、关闭TCP连接——需要“四次挥手”

6、浏览器收到结果，开始解析资源（JS、CSS、HTML），解析HTML生成的dom树，和同时解析css生成的cssom树结合生成渲染树
7、根据渲染树渲染页面

从浏览器地址栏输入 url 到显示页面的步骤(以 HTTP 为例)

1. 在浏览器地址栏输入 URL
2. 浏览器查看

缓存

1. 如果资源未缓存，发起新请求
2. 如果已缓存，检验是否足够新鲜，足够新鲜直接提供给客户端，否则与服务器进行验证。
3. 检验新鲜通常有两个 HTTP 头进行控制
   Expires
   Cache-Control

- HTTP1.0 提供 Expires，值为一个绝对时间表示缓存新鲜日期
- HTTP1.1 增加了 Cache-Control: max-age=,值为以秒为单位的最大新鲜时间

3. 浏览器**解析 URL**获取协议，主机，端口，path
4. 浏览器**组装一个 HTTP（GET）请求报文**
5. 浏览器获取主机 ip 地址，过程如下：
    1. 浏览器缓存
    2. 本机缓存
    3. hosts 文件
    4. 路由器缓存
    5. ISP DNS 缓存
    6. DNS 递归查询（可能存在负载均衡导致每次 IP 不一样）
6. 打开一个 socket 与目标 IP 地址，端口建立 TCP 链接，三次握手如下：
    1. 客户端发送一个 TCP 的**SYN=1，Seq=X**的包到服务器端口
    2. 服务器发回**SYN=1， ACK=X+1， Seq=Y**的响应包
    3. 客户端发送**ACK=Y+1， Seq=Z**
7. TCP 链接建立后**发送 HTTP 请求**
8. 服务器接受请求并解析，将请求转发到服务程序，如虚拟主机使用 HTTP Host 头部判断请求的服务程序
9. 服务器检查**HTTP 请求头是否包含缓存验证信息**如果验证缓存新鲜，返回**304**等对应状态码
10. 处理程序读取完整请求并准备 HTTP 响应，可能需要查询数据库等操作
11. 服务器将**响应报文通过 TCP 连接发送回浏览器**
12. 浏览器接收 HTTP 响应，然后根据情况选择
    关闭 TCP 连接或者保留重用，关闭 TCP 连接的四次握手如

    1. 主动方发送**Fin=1， Ack=Z， Seq= X**报文
    2. 被动方发送**ACK=X+1， Seq=Z**报文
    3. 被动方发送**Fin=1， ACK=X， Seq=Y**报文
    4. 主动方发送**ACK=Y， Seq=X**报文
13. 浏览器检查响应状态吗：是否为 1XX，3XX， 4XX， 5XX，这些情况处理与 2XX 不同
14. 如果资源可缓存，**进行缓存**
15. 对响应进行**解码**（例如 gzip 压缩）
16. 根据资源类型决定如何处理（假设资源为 HTML 文档）
17. **解析 HTML 文档，构件 DOM 树，下载资源，构造 CSSOM 树，执行 js 脚本**，这些操作没有严格的先后顺序，以下分别解释
18. 构建 DOM 树
    1. **Tokenizing**：根据 HTML 规范将字符流解析为标记
    2. **Lexing**：词法分析将标记转换为对象并定义属性和规则
    3. **DOM construction**：根据 HTML 标记关系将对象组成 DOM 树
19. 解析过程中遇到图片、样式表、js 文件，**启动下载**
20. 构建
    CSSOM 树
    1. **Tokenizing**：字符流转换为标记流
    2. **Node**：根据标记创建节点
    3. **CSSOM**：节点创建 CSSOM 树
21. 根据 DOM 树和 CSSOM 树构建渲染树
    1. 从 DOM 树的根节点遍历所有**可见节点**，不可见节点包括：1）`script`,`meta`这样本身不可见的标签。2)被 css 隐藏的节点，如`display: none`
    2. 对每一个可见节点，找到恰当的 CSSOM 规则并应用
    3. 发布可视节点的内容和计算样式
22. js 解析如下
    ：
    1. 浏览器创建 Document 对象并解析 HTML，将解析到的元素和文本节点添加到文档中，此时**document.readystate 为 loading**
    2. HTML 解析器遇到**没有 async 和 defer 的 script 时**，将他们添加到文档中，然后执行行内或外部脚本。这些脚本会同步执行，并且在脚本下载和执行时解析器会暂停。这样就可以用 document.write()把文本插入到输入流中。**同步脚本经常简单定义函数和注册事件处理程序，他们可以遍历和操作 script 和他们之前的文档内容**
    3. 当解析器遇到设置了**async**属性的 script 时，开始下载脚本并继续解析文档。脚本会在它**下载完成后尽快执行**，但是**解析器不会停下来等它下载。**异步脚本**禁止使用 document.write()**，它们可以访问自己 script 和之前的文档元素
    4. 当文档完成解析，document.readState 变成 interactive
    5. 所有**defer**脚本会**按照在文档出现的顺序执行**，延迟脚本**能访问完整文档树**，禁止使用 document.write()
    6. 浏览器**在 Document 对象上触发 DOMContentLoaded 事件**
    7. 此时文档完全解析完成，浏览器可能还在等待如图片等内容加载，等这些**内容完成载入并且所有异步脚本完成载入和执行**，document.readState 变为 complete,window 触发 load 事件
23. **显示页面**（HTML 解析过程中会逐步显示页面）

简单版

1.解析URL来请求指定的服务器中指定服务
2、建立TCP连接--进行“三次握手”
3、客户端发送http请求
4、服务端处理，并返回结果给客户端
5、关闭TCP连接--需要“四次挥手”
6、浏览器收到结果，开始解析资源（JS、CSS、HTML），解析HTML生成的dom树，和同时解析css生成的cssom树结合生成渲染树
7、根据渲染树渲染页面

1、输入url

2、查看浏览器缓存，看是否有缓存，如果有缓存，继续查看缓存是否过期，如果没有过期，直接返回缓存页面，如果没有缓存或者缓存过期，发送一个请求。

3、浏览器解析url地址，获取协议、主机名、端口号和路径。

//4、获取主机ip地址过程

（1）浏览器缓存

（2）主机缓存

（3）hosts文件

（4）路由器缓存

（5）DNS缓存

（6）DNS递归查询

5、浏览器发起和服务器的TCP连接，执行三次握手（略）

6、三次握手连接后，浏览器发送一个http请求（这部分是重点，请查询相关资料，详细了解http协议关于请求格式和重要的几个请求头字段含义）。

7、服务器收到请求，转到相关的服务程序，期间可能需要连接并操作数据库（主要分get和post请求）。

8、服务器看是否需要缓存，服务器处理完请求，发出一个响应（这部分也是重点，请查询资料了解http响应头各个字段的含义）

9、服务器并根据请求头包含信息决定是否需要关闭TCP连接（如需关闭，则需要四次挥手过程）

10、浏览器对接收到的响应进行解码

11、浏览器解析收到的响应并根据响应的内容（假如是HTML文件）进行构建DOM树，构建render树，渲染render树等过程

什么是http:应用层协议

HTTP协议，即超文本传输协议,规范了浏览器和万维网服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。

协议:协议是指计算机通信网络中两台计算机之间进行通信所必须共同遵守的规定或规则;
超文本传输协议(HTTP)是一种通信协议，它允许将超文本标记语言(HTML)文档从Web服务器传送到客户端的浏览器。

特点:
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。
无状态：HTTP协议是无状态协议。
无状态是指客户端和服务端之间不需要建立持久的连接 ， 客户端发起一个请求 ， 服务器端返回响应 ， 这个连接就会被关闭 ， 在服务器端不会保留该请求的有关信息 .

请求报文由请求行、请求头部、空行 和 请求包体(请求正文) 4 个部分组成

1.请求行：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。
求方法字段、URL字段和HTTP协议版本
例如：GET /index.html HTTP/1.1
get方法将数据拼接在url后面，传递参数受限
请求方法：
GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT
2.请求头部：
- 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;

3.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方法中，会把数据以key value形式发送请求
4.空行
发送回车符和换行符，通知服务器以下不再有请求头

GET方式的请求一般不包含”请求内容”部分，请求数据以地址的形式表现在请求行。

```
GET /search?hl=zh-CN&source=hp&q=domety&aq=f&oq= HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, application/x-silverlight, application/x-shockwave-flash, */*

Referer: <a href="http://www.google.cn/">http://www.google.cn/</a>
Accept-Language: zh-cn
```

```
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
2.0.50727; TheWorld)
Host: <a href="http://www.google.cn">www.google.cn</a>
Connection: Keep-Alive
Cookie:
PREF=ID=80a06da87be9ae3c:U=f7167333e2c3b714:NW=1:TM=1261551909:LM=1261551917:S=y
bYcq2wpfefs4V9g;
NID=31=ojj8d-IygaEtSxLgaJmqSjVhCspkviJrB6omjamNrSm8lZhKy_yMfO2M4QMRKcH1g0iQv9u-
2hfBW7bUFwVh7pGaRUb0RnHcJU37y-
FxlRugatx63JLv7CWMD6UB_O_r
```

HTTP响应报文由状态行、响应头部、空行 和 响应包体 4 个部分组成

```
1.状态行
    由3部分组成，分别为：协议版本，状态码，状态码描述，之间由空格分隔
2.响应头部:与请求头部类似，为响应报文添加了一些附加信息
    Server 服务器应用程序软件的名称和版本
    Content-Type 响应正文的类型（是图片还是二进制字符串）
    Content-Length 响应正文长度
    Content-Charset 响应正文使用的编码
    Content-Encoding 响应正文使用的数据压缩格式
    Content-Language 响应正文使用的语言
```

TCP

在Internet中所有的传输都是通过TCP/IP进行的。HTTP协议作为TCP/IP模型中应用层的协议也不例外。HTTP协议通常承载于TCP协议之上，有时也承载于TLS或SSL协议层之上，这个时候，就成了我们常说的HTTPS。如下图所示：

```
在TCP/IP的模型图中 ， HTTP协议位于最上层的应用层 ， 它是互联网上应用最为广泛的一种网络协议 ， 所有 www 文件都必须遵守这个协议 .

 HTTP 是一个由请求和响应组成的 ， 标准的客户端/服务端模型(B/S结构) . HTTP 协议永远是由客户端发起请求 ， 服务端给与响应 ，
```

Http与Https的区别

- HTTP 的URL 以http:// 开头，而HTTPS 的URL 以https:// 开头
- HTTP 是不安全的，而 HTTPS 是安全的
- HTTP 标准端口是80 ，而 HTTPS 的标准端口是443
- 在OSI 网络模型中，HTTP工作于应用层，而HTTPS 的安全传输机制工作在传输层
- HTTP 无法加密，而HTTPS 对传输的数据进行加密
- HTTP无需证书，而HTTPS 需要CA机构wosign的颁发的SSL证书

HTTP协议是无状态的和Connection: keep-alive的区别：

从HTTP/1.1起，默认都开启了Keep-Alive，保持连接特性，简单地说，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。
Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。

cookie 和 Session解决http无状态

cookie是web浏览器存储的少量数据，最早设计为服务器端使用，作为HTTP协议的扩展实现。cookie数据会自动在浏览器和服务器之间传输。
通过读写cookie检测是否支持
cookie属性有名，值，max-age，path，domain，secure；
cookie默认有效期为浏览器会话，一旦用户关闭浏览器，数据就丢失，通过设置max-age=seconds属性告诉浏览器cookie有效期
cookie作用域通过文档源和文档路径来确定，通过path和domain进行配置，web页面同目录或子目录文档都可访问
通过cookie保存数据的方法为：为document.cookie设置一个符合目标的字符串如下
读取document.cookie获得'; '分隔的字符串，key=value,解析得到结果

```
document.cookie = 'name=qiu; max-age=9999; path=/; domain=domain; secure';
document.cookie = 'name=aaa; path=/; domain=domain; secure';
// 要改变cookie的值，需要使用相同的名字、路径和域，新的值
// 来设置cookie，同样的方法可以用来改变有效期
// 设置max-age为0可以删除指定cookie
//读取cookie，访问document.cookie返回键值对组成的字符串，
//不同键值对之间用'; '分隔。通过解析获得需要的值
```

Session机制:服务器端使用的一种记录客户端状态的机制

传到前端的是SessionId 用cookie 或是Url重写存起来,下次再请求 如果session没过期 根据id可以获取到session里存储信息。

Session对浏览器的要求:
虽然Session保存在服务器，对客户端是透明的，它的正常运行仍然需要客户端浏览器的支持。这是因为Session需要使用Cookie作为识别标志。HTTP协议是无状态的，Session不能依据HTTP连接来判断是否为同一客户，因此服务器向客户端浏览器发送一个名为JSESSIONID的Cookie，它的值为该Session的id（也就是HttpSession.getId()的返回值）。Session依据该Cookie来识别是否为同一用户

注意：
新开的浏览器窗口会生成新的Session，但子窗口除外。子窗口会共用父窗口的Session。例如，在链接上右击，在弹出的快捷菜单中选择“在新窗口中打开”时，子窗口便可以访问父窗口的Session。

Session 和cookie 区别:

1、cookie数据存放在客户的浏览器上，session数据放在服务器上.

登录一个网站的时候，如果web服务器端使用的是session,那么所有的数据都保存在服务器上面，客户端每次请求服务器的时候会发送 当前会话的session_id，服务器根据当前session_id判断相应的用户数据标志，以确定用户是否登录，或具有某种权限。
SessionID这一数据则是保存到客户端，用Cookie保存的，用户提交页面时，会将这一 SessionID提交到服务器端，来存取Session数据。这一过程，是不用开发人员干预的。所以一旦客户端禁用Cookie，那么Session也会失效。

2.设置cookie时间可以使cookie过期。但是使用session-destory（），我们将会销毁会话
3.session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用cookie。
4.单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。(Session对象没有对存储的数据量的限制，其中可以保存更为复杂的数据类型)
注意:

session很容易失效,用户体验很差;
虽然cookie不安全,但是可以加密 ;
cookie也分为永久和暂时存在的;
浏览器 有禁止cookie功能 ,但一般用户都不会设置;
定要设置失效时间,要不然浏览器关闭就消失了;

例如:
记住密码功能就是使用永久cookie写在客户端电脑，下次登录时，自动将cookie信息附加发送给服务端。

HTTP常见的响应头:Response header

- Allow：服务器支持哪些请求方法（如GET、POST等）。

- Date：表示消息发送的时间，时间的描述格式由rfc822定义。例如，

- Date:Mon,31Dec200104:25:57GMT。Date描述的时间表示世界标准时，换算成本地时间，需要知道用户所在的时区。你可以用setDateHeader来设置这个头以避免转换时间格式的麻烦

- Expires：指明应该在什么时候认为文档已经过期，从而不再缓存它，重新从服务器获取，会更新缓存。过期之前使用本地缓存。HTTP1.1的客户端和缓存会将非法的日期格式（包括0）看作已经过期。eg：为了让浏览器不要缓存页面，我们也可以将Expires实体报头域，设置为0。

- Set-Cookie：非常重要的header，用于把cookie发送到客户端浏览器，每一个写入cookie都会生成一个Set-Cookie。Set-Cookie: sc=4c31523a; path=/; domain=.acookie.taobao.com

- Content-Type：WEB服务器告诉浏览器自己响应的对象的类型和字符集。Servlet默认为text/plain，但通常需要显式地指定为text/html。由于经常要设置Content-Type，因此HttpServletResponse提供了一个专用的方法setContentType。可在web.xml文件中配置扩展名和MIME类型的对应关系。

  Content-Type:text/html;charset=GB2312

  Content-Type: image/jpeg

http报文头部有哪些字段? 有什么意义 ?
这个就很多了，cookie cache-control user-agent expires host refer 等等 挑你会的常用的说，面试官也不会要求你都说全的

HTTP常见的请求头:request header

-Accept 请求头用来告知客户端可以处理的内容类型，这种内容类型用MIME类型来表示。Accept: text/xml;代表客户端希望接受的数据类型是xml类型

- Cache-Control：指定请求和响应遵循的缓存机制。缓存指令是单向的（响应中出现的缓存指令在请求中未必会出现），且是独立的（在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中的缓存处理过程）。请求时的缓存指令包括no-cache、no-store、max-age、max-stale、min-fresh、only-if-cached，响应消息中的指令包括public、private、no-cache、no-store、no-transform、must-revalidate、proxy-revalidate、max-age、s-maxage。

  Cache-Control:Public 可以被任何缓存所缓存

  Cache-Control:Private 内容只缓存到私有缓存中

  Cache-Control:no-cache 所有内容都不会被缓存

- Accept：浏览器端可以接受的MIME类型。例如：Accept: text/html 代表浏览器可以接受服务器回发的类型为 text/html 也就是我们常说的html文档，如果服务器无法返回text/html类型的数据，服务器应该返回一个406错误(non acceptable)。通配符 * 代表任意类型，例如 Accept: */* 代表浏览器可以处理所有类型，(一般浏览器发给服务器都是发这个)。

- Accept-Encoding：浏览器申明自己可接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate）;Servlet能够向支持gzip的浏览器返回经gzip编码的HTML页面。许多情形下这可以减少5到10倍的下载时间。例如： Accept-Encoding: gzip, deflate。如果请求消息中没有设置这个域，服务器假定客户端对各种内容编码都可以接受。

- Cookie：最重要的请求头之一，将cookie的值发送给HTTP服务器。

- Content-Length：表示请求消息正文的长度。例如：Content-Length: 38。

- Authorization：授权信息，通常出现在对服务器发送的WWW-Authenticate头的应答中。主要用于证明客户端有权查看某个资源。当浏览器访问一个页面时，如果收到服务器的响应代码为401（未授权），可以发送一个包含Authorization请求报头域的请求，要求服务器对其进行验证。

状态码

1(信息类)**：表示接收到请求并且继续处理
2(响应成功)**：表示动作被成功接收、理解和接受

3(重定向类)**：为了完成指定的动作，必须接受进一步处理
300——请求的资源可在多处得到
301——本网页被永久性转移到另一个URL
302——请求的网页被转移到一个新的地址，但客户访问仍继续通过原始URL地址，重定向，新的URL会在response中的Location中返回，浏览器将会使用新的URL发出新的Request。
303——建议客户访问其他URL或访问方式
304——自从上次请求后，请求的网页未修改过，服务器返回此响应时，不会返回网页内容，代表上次的文档已经被缓存了，还可以继续使用
305——请求的资源必须从服务器指定的地址得到
306——前一版本HTTP中使用的代码，现行版本中不再使用
307——申明请求的资源临时性删除

4(客户端错误类)**：请求包含错误语法或不能正确执行
404——一个404错误表明可连接服务器，但服务器无法取得所请求的网页，请求资源不存在。eg：输入了错误的URL
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
HTTP 401.5 - 未授权：ISAPI 或 CGI 授权失败
402——保留有效ChargeTo头响应
403——禁止访问，服务器收到请求，但是拒绝提供服务

5(服务端错误类)**：服务器不能正确执行一个正确的请求
HTTP 500 - 服务器遇到错误，无法完成请求
HTTP 500.100 - 内部服务器错误 - ASP 错误
HTTP 500-11 服务器关闭
HTTP 500-12 应用程序重新启动
HTTP 500-13 - 服务器太忙
HTTP 500-14 - 应用程序无效
HTTP 500-15 - 不允许请求 global.asa
Error 501 - 未实现
HTTP 502 - 网关错误
HTTP 503：由于超载或停机维护，服务器目前无法使用，一段时间后可能恢复正常

三次握手（Three-way Handshake）其实就是指建立一个TCP连接时，需要客户端和服务器总共发送3个包。进行三次握手的主要作用就是为了确认双方的接收能力和发送能力是否正常、指定自己的初始化序列号为后面的可靠性传送做准备。实质上其实就是连接服务器指定端口，建立TCP连接，并同步连接双方的序列号和确认号，交换`TCP窗口大小`信息。

比较简单：

1）No1：浏览器（192.168.1.6）向服务器（115.239.210.36）发出连接请求。此为TCP三次握手第一步，此时从图中可以看出，为SYN，seq:X （x=0）；
2）No2：服务器（115.239.210.36）回应了浏览器（192.168.1.6）的请求，并要求确认，此时为：SYN，ACK，此时seq：y（y为0），ACK：x+1（为1）。此为三次握手的第二步；
3）No3：浏览器（192.168.1.6）回应了服务器（115.239.210.36）的确认，连接成功。为：ACK，此时seq：x+1（为1），ACK：y+1（为1）。此为三次握手的第三步；
//在TCP三次握手之后，建立了TCP连接，此时HTTP就可以进行传输了。
4）No4：浏览器（192.168.1.6）发出一个页面HTTP请求；
5）No5：服务器（115.239.210.36）确认；
6）No6：服务器（115.239.210.36）发送数据；
7）No8：客户端浏览器（192.168.1.6）确认；
8）No81：客户端（192.168.1.6）发出一个图片HTTP请求；
9）No202：服务器（115.239.210.36）发送状态响应码200 OK。

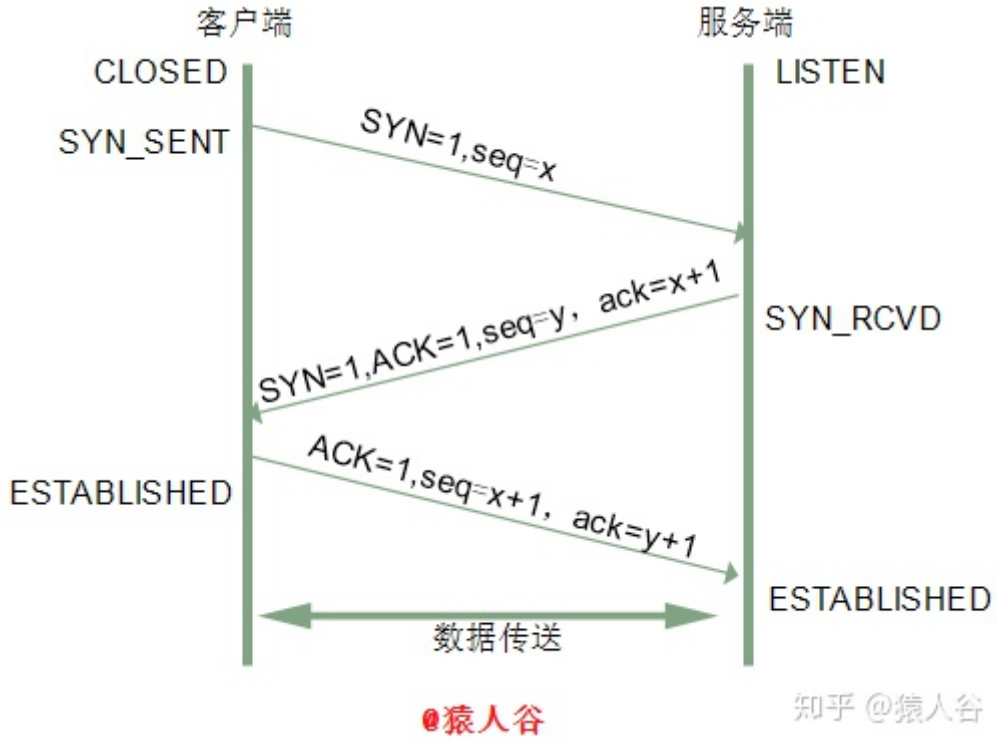
面试时越简单的问题，一般就是隐藏着比较大的坑，一般都是需要将问题扩展的。上面求职者的回答不对吗？当然对，但距离面试官的期望可能还有点距离。
希望大家能带着如下问题进行阅读，收获会更大。 1. 请画出三次握手和四次挥手的示意图 2. 为什么连接的时候是三次握手？ 3. 什么是半连接队列？ 4. ISN(Initial Sequence Number)是固定的吗？ 5. 三次握手过程中可以携带数据吗？ 6. 如果第三次握手丢失了，客户端服务端会如何处理？ 7. SYN攻击是什么？ 8. 挥手为什么需要四次？ 9. 四次挥手释放连接时，等待2MSL的意义？

刚开始客户端处于 Closed 的状态，服务端处于 Listen 状态。 进行三次握手：

- 第一次握手：客户端给服务端发一个 SYN 报文，并指明客户端的初始化序列号 ISN(c)。此时客户端处于 `SYN_SEND` 状态。
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。

- 第二次握手：服务器收到客户端的 SYN 报文之后，会以自己的 SYN 报文作为应答，并且也是指定了自己的初始化序列号 ISN(s)。同时会把客户端的 ISN + 1 作为ACK 的值，表示自己已经收到了客户端的 SYN，此时服务器处于 `SYN_REVD` 的状态。
在确认报文段中SYN=1，ACK=1，确认号ack=x+1，初始序号seq=y。

- 第三次握手：客户端收到 SYN 报文之后，会发送一个 ACK 报文，当然，也是一样把服务器的 ISN + 1 作为 ACK 的值，表示已经收到了服务端的 SYN 报文，此时客户端处于 `ESTABLISHED` 状态。服务器收到 ACK 报文之后，也处于 `ESTABLISHED` 状态，此时，双方已建立起了连接。
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。
发送第一个SYN的一端将执行主动打开（active open），接收这个SYN并发回下一个SYN的另一端执行被动打开（passive open）。
在socket编程中，客户端执行connect()时，将触发三次握手。



1.1 为什么需要三次握手，两次不行吗?

弄清这个问题，我们需要先弄明白三次握手的目的是什么，能不能只用两次握手来达到同样的目的。

- 第一次握手：客户端发送网络包，服务端收到了。 这样服务端就能得出结论：客户端的发送能力、服务端的接收能力是正常的。
- 第二次握手：服务端发包，客户端收到了。 这样客户端就能得出结论：发送能力，客户端的接收是正常的。不过此时服务器并不能确认客户端的接收能力是否正常。
- 第三次握手：客户端发包，服务端收到了。 这样服务端就能得出结论：客户端的接收、发送能力正常，服务器自己的发送、接收能力也正常。

因此，需要三次握手才能确认双方的接收与发送能力是否正常。

试想如果是用两次握手，则会出现下面这种情况：

> 如客户端发出连接请求，但因连接请求报文丢失而未收到确认，于是客户端再重传一次连接请求。后来收到了确认，建立了连接。数据传输完毕后，就释放了连接，客户端共发出了两个连接请求报文段，其中第一个丢失，第二个到达了服务端，但是第一个丢失的报文段只是在**某些网络结点长时间滞留了，延误到连接释放以后的某个时间才到达服务端**，此时服务端误认为客户端又发出一次新的连接请求，于是就向客户端发出确认报文段，同意建立连接，不采用三次握手，只要服务端发出确认，就建立新的连接了，此时客户端忽略服务端发来的确认，也不发送数据，则服务端一致等待客户端发送数据，浪费资源。

1.4 三次握手过程中可以携带数据吗?

其实第三次握手的时候，是可以携带数据的。但是，第一次、第二次握手不可以携带数据
为什么这样呢?大家可以想一个问题，假如第一次握手可以携带数据的话，如果有人要恶意攻击服务器，那他每次都在第一次握手中的 SYN 报文中放入大量的数据。因为攻击者根本就不理服务器的接收、发送能力是否正常，然后疯狂着重复发 SYN 报文的话，这会让服务器花费很多时间、内存空间来接收这些报文。

也就是说，第一次握手不可以放数据，其中一个简单的原因就是会让服务器更加容易受到攻击了。而对于第三次的话，此时客户端已经处于 ESTABLISHED 状态。对于客户端来说，他已经建立起连接了，并且也已经知道服务器的接收、发送能力是正常的了，所以能携带数据也没啥毛病。

1.5 SYN攻击是什么?

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Client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。SYN 攻击是一种典型的 DoS/DDoS 攻击。

检测 SYN 攻击非常的方便，当你在服务器上看到大量的半连接状态时，特别是源IP地址是随机的，基本上可以断定这是一次SYN攻击。在 Linux/Unix 上可以使用系统自带的 netstats 命令来检测 SYN 攻击。

2. 四次挥手

建立一个连接需要三次握手，而终止一个连接要经过四次挥手（也有将四次挥手叫做四次握手的）。这由TCP的**半关闭**（half-close）造成的。所谓的半关闭，其实就是TCP提供了连接的一端在结束它的发送后还能接收来自另一端数据的能力。

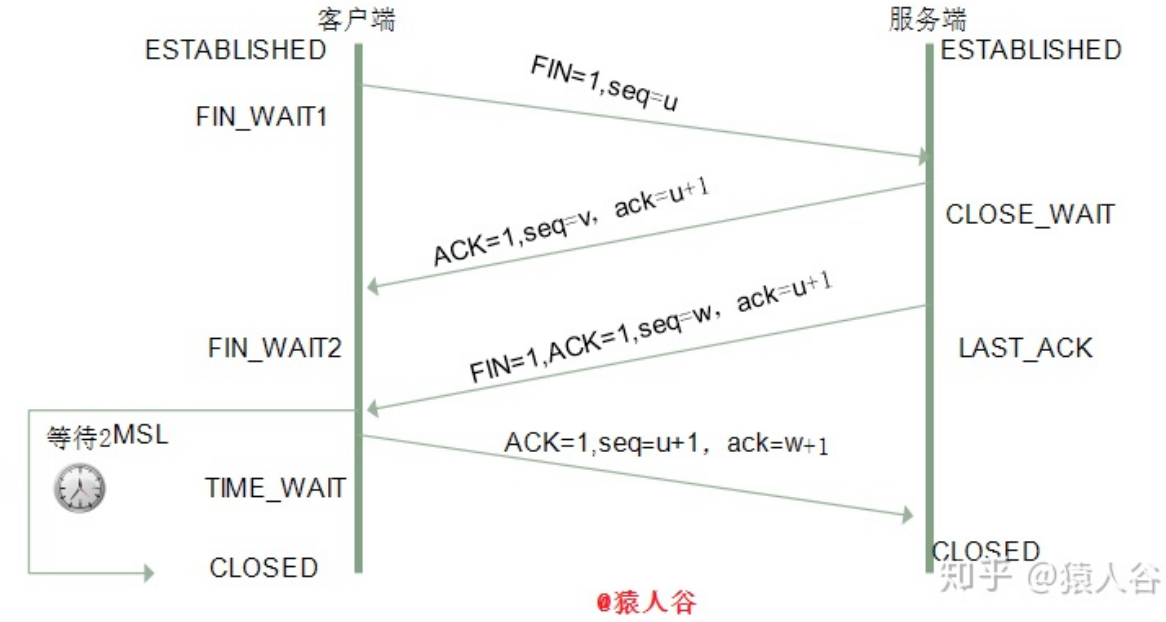
TCP 的连接的拆除需要发送四个包，因此称为四次挥手(Four-way handshake)，客户端或服务器均可主动发起挥手动作。

刚开始双方都处于 ESTABLISHED 状态，假如是客户端先发起关闭请求。四次挥手的过程如下：

- 第一次挥手：客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于 `FIN_WAIT1` 状态。 即发出**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_WAIT1（终止等待1）状态，等待服务端的确认。
- 第二次挥手：服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 +1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 `CLOSE_WAIT` 状态。 即服务端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），服务端进入CLOSE_WAIT（关闭等待）状态，此时的TCP处于半关闭状态，客户端到服务端的连接释放。客户端收到服务端的确认后，进入FIN_WAIT2（终止等待2）状态，等待服务端发出的连接释放报文段。
- 第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 `LAST_ACK` 的状态。 即服务端没有要向客户端发出的数据，服务端发出**连接释放报文段**（FIN=1，ACK=1，序号seq=w，确认号ack=u+1），服务端进入LAST_ACK（最后确认）状态，等待客户端的确认。
- 第四次挥手：客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 +1 作为自己 ACK 报文的序列号值，此时客户端处于 `TIME_WAIT` 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态，服务端收到 ACK 报文之后，就处于关闭连接了，处于 `CLOSED` 状态。 即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），客户端进入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。

收到一个FIN只意味着在这一方向上没有数据流动。客户端执行主动关闭并进入TIME_WAIT是正常的，服务端通常执行被动关闭，不会进入TIME_WAIT状态。

在socket编程中，任何一方执行close()操作即可产生挥手操作。



2.1 挥手为什么需要四次?

因为当服务端收到客户端的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，当服务端收到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，告诉客户端，"你发的FIN报文我收到了"。只有等到我服务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。故需要四次挥手。

题目

关于 HTTP 协议，下面说话错误的是哪个一个：

A. 看到网页有乱码，则很有可能是某个请求的 `Content-Type` 响应头丢失或者是值设置不当造成的
B. 即便是不需要发送请求体的 `GET` 请求，请求头区域下方也必须留一个空行（CRLF）
C. 服务端可以根据客户端发送的 `Accept-Encoding` 请求头来分别返回不同压缩格式（Gzip、Brotli）的文件
D. 服务端返回的 `Date` 响应头表示服务器上的系统时间，除给人读外没有实际用途
E. HTTP 是无状态的，网站是通过 `Cookie` 请求头来识别出两个请求是不是来自同一个浏览器的
F. `Access-Control-Allow-Origin` 响应头只支持配置单个的域名或者是 `*` ，不支持配置多个特定的域名

参考答案：D
D 为错误选项，`Date` 响应头有参与缓存时长的计算，不仅仅是给人看看服务器时间。

http基于 TCP/IP协议族

TCP/IP 协议族重要一点就是分层：

1.应用层
2.传输层
3.网络层
4.数据链路层

TCP/IP 通信传输流

利用 TCP/IP 协议族进行网络通信时，会通过分层顺序与对方进行通 信。发送端从应用层往下走，接收端则往应用层往上走。
实例：

> 我们用 HTTP 举例来说明，服务端：
> 1.首先作为发送端的客户端在应用层 （HTTP 协议）发出一个想看某个 Web 页面的 HTTP 请求。
> 2.接着，为了传输方便，在传输层（TCP 协议）把从应用层处收到的数 据（HTTP 请求报文）进行分割，
> 3.并在各个报文上打上标记序号及端口号后转发给网络层
> 4.在网络层（IP 协议），增加作为通信目的地的 MAC 地址后转发给链 路层。这样一来，发往网络的通信请求就准备齐全了。
>
> 服务端：
> 接收端的服务器在链路层接收到数据，按序往上层发送，一直到应用 层。当传输到应用层，才能算真正接收到由客户端发送过来的 HTTP 请求。

解析：
发送端在层与层之间传输数据时，每经过一层时必定会被打上一个该 层所属的首部信息。反之，接收端在层与层传输数据时，每经过一层 时会把对应的首部消去。
这种把数据信息包装起来的做法称为封装（encapsulate）

负责传输的 IP 协议

> 按层次分，IP（Internet Protocol）网际协议位于网络层。Internet Protocol 这个名称可能听起来有点夸张，但事实正是如此，因为几乎 所有使用网络的系统都会用到 IP 协议。TCP/IP 协议族中的 IP 指的就 是网际协议，协议名称中占据了一半位置，其重要性可见一斑。可能 有人会把“IP”和“IP 地址”搞混，“IP”其实是一种协议的名称。

IP 协议的作用是把各种数据包传送给对方。而要保证确实传送到对方 那里，则需要满足各类条件。其中两个重要的条件是 IP 地址和 MAC 地址。
IP 地址指明了节点被分配到的地址，MAC 地址是指网卡所属的固定 地址。IP 地址可以和 MAC 地址进行配对。IP 地址可变换，但 MAC 地址基本上不会更改。
###使用 ARP 协议凭借 MAC 地址进行通信。

> IP 间的通信依赖 MAC 地址。在网络上，通信的双方在同一局域网 （LAN）内的情况是很少的，通常是经过多台计算机和网络设备中转 才能连接到对方。而在进行中转时，会利用下一站中转设备的 MAC 地址来搜索下一个中转目标。这时，会采用 ARP 协议（Address Resolution Protocol）。ARP 是一种用以解析地址的协议，根据通信方 的 IP 地址就可以反查出对应的 MAC 地址。

确保可靠性的TCP协议

TCP位于传输层，提供可靠的字节流服务。
所谓的字节流服务：为方便传输，将大块数据分割成以报文段 为单位的数据包进行管理。
可靠的传输服务：能够把数据准确可靠传给对方。

> 总结：
> tcp协议为方便传输，将大块数据分割成以报文段 为单位的数据包进行管理,而且tcp协议能够确认数据最终能否送到对方

确保数据达到目标：tcp采用三次握手策略

tcp发送后向对方确认是否成功送达：

```
SYN
ACK
```

负责域名解析的DNS服务

DNS服务是和http协议在应用层的协议，dns提供域名到ip地址之间的解析服务

用户通常通过使用主机名或域名来访问对方的计算机，而不是直接通过IP地址访问。所有需要dns协议提供通过域名查找ip地址，或逆向从ip地址查找域名服务。

1.应用层

应用层决定了向用户提供应用服务时通信的活动。||　　应用层负责处理特定的应用程序细节。
　　TCP/IP 协议族内预存了各类通用的应用服务。比如，FTP（File Transfer Protocol，文件传输协议）和 DNS（Domain Name System，域 名系统）服务就是其中两类。 HTTP 协议也处于该层

2.传输层

传输层对上层应用层，提供处于网络连接中的两台计算机之间的数据 传输。
在传输层有两个性质不同的协议：TCP（Transmission Control Protocol，传输控制协议）和UDP（User Data Protocol，用户数据报 协议）。

　　主要为两台主机上的应用程序提供端到端的通信。在TCP/IP协议族中，有两个互不相同的传输协议：TCP（传输控制协议）和UDP（用户数据报协议）。
　　TCP为两台主机提供高可靠性的数据通信。它所做的工作包括把应用程序交给它的数据分成合适的小块交给下面的网络层，确认接收到的分组，设置发送最后确认分组的超时时钟等。由于运输层提供了高可靠性的端到端的通信，因此应用层可以忽略所有这些细节。为了提供可靠的服务，TCP采用了超时重传、发送和接收端到端的确认分组等机制。

　　UDP则为应用层提供一种非常简单的服务。它只是把称作数据报的分组从一台主机发送到另一台主机，但并不保证该数据报能到达另一端。一个数据报是指从发送方传输到接收方的一个信息单元（例如，发送方指定的一定字节数的信息）。UDP协议任何必需的可靠性必须由应用层来提供。

3.网络层

网络层用来处理在网络上流动的数据包。数据包是网络传输的最小数 据单位。该层规定了通过怎样的路径（所谓的传输路线）到达对方计 算机，并把数据包传送给对方。

与对方计算机之间通过多台计算机或网络设备进行传输时，网络层所 起的作用就是在众多的选项内选择一条传输路线。

也称作互联网层（在第一个图中为网际层），处理分组在网络中的活动，例如分组的选路。在TCP/IP协议族中，网络层协议包括IP协议（网际协议），ICMP协议（Internet互联网控制报文协议），以及IGMP协议（Internet组管理协议）

IP是一种网络层协议，提供的是一种不可靠的服务，它只是尽可能快地把分组从源结点送到目的结点，但是并不提供任何可靠性保证。同时被TCP和UDP使用。TCP和UDP的每组数据都通过端系统和每个中间路由器中的IP层在互联网中进行传输。

ICMP是IP协议的附属协议。IP层用它来与其他主机或路由器交换错误报文和其他重要信息。

IGMP是Internet组管理协议。它用来把一个UDP数据报多播到多个主机。

4.链路层

用来处理连接网络的硬件部分。包括控制操作系统、硬件的设备驱 动、NIC（Network Interface Card，网络适配器，即网卡），及光纤等 物理可见部分（还包括连接器等一切传输媒介）。硬件上的范畴均在 链路层的作用范围之内

也称作数据链路层或网络接口层（在第一个图中为网络接口层和硬件层），通常包括操作系统中的设备驱动程序和计算机中对应的网络接口卡。它们一起处理与电缆（或其他任何传输媒介）的物理接口细节。ARP（地址解析协议）和RARP（逆地址解析协议）是某些网络接口（如以太网和令牌环网）使用的特殊协议，用来转换IP层和网络接口层使用的地址。

05*五章*与http协作的web服务器

通信数据转发程序：代理，网关，隧道

http通信时候，除客户端和服务器以外，还有一些用于通信数据转发的应用程序，例如代理，网关，隧道。

代理:中间人
网关：可以由http转化为其他通信
隧道：确保安全通信，不会解析http请求，隧道在双方断开连接中断

六章*安全*https

HTTP+ 加密 + 认证 + 完整性保护 =HTTPS

如果在 HTTP 协议通信过程中使用未经加密的明文，比如在 Web 页面中输入信用卡号，如果这条通信线路遭到窃听，那么信用卡号就暴露了。 另外，对于 HTTP 来说，服务器也好，客户端也好，都是没有办法确认通信方的。因为很有可能并不是和原本预想的通信方在实际通信。并且还需要考虑到接收到的报文在通信途中已经遭到篡改这一可能性。 为了统一解决上述这些问题，需要在 HTTP 上再加入加密处理和认证等机制。我们把添加了加密及认证机制的 HTTP 称为 HTTPS（HTTP Secure）

经常会在 Web 的登录页面和购物结算界面等使用 HTTPS 通 信。使用 HTTPS 通信时，不再用 http://，而是改用 https://。另外，当浏览器访问 HTTPS 通信有效的 Web 网站时，浏览器的地址栏内会出现一个带锁的标记。对 HTTPS 的显示方式会因浏览器的不同而有所改变。

1.HTTPS 是身披 SSL 外壳的 HTTP

HTTPS 并非是应用层的一种新协议。只是 HTTP 通信接口部分用 SSL（Secure Socket Layer）和 TLS（Transport Layer Security）协议代替而已。 通常，HTTP 直接和 TCP 通信。当使用 SSL 时，则演变成先和 SSL 通信，再由 SSL 和 TCP 通信了。简言之，所谓 HTTPS，其实就是身披 SSL 协议这层外壳的 HTTP。

在采用 SSL 后，HTTP 就拥有了 HTTPS 的加密、证书和完整性保护这些功能。 SSL 是独立于 HTTP 的协议，所以不光是 HTTP 协议，其他运行在应用层的 SMTP 和 Telnet 等协议均可配合 SSL 协议使用。可以说 SSL 是当今世界上应用最为广泛的网络安全技术。

确认访问用户身份的认证

session 管理及cookie应用：使用cookie来管理session,弥补http协议中不存在的状态管理功能

1.客户端：发送密码登录

2.服务端向用户发送sesstion

然后通过验证session Id 判定对方身份