六星教育 WEB 前端学院面试圣经

应用层重点

HTTP 协议

行叫做请求行，后面的行叫做首部行，首部行后还可以跟一个实体主体。请求首部之后有一个空行，这 个空行不能省略，它用来划分首部与实体。

请求行包含三个字段：方法字段、URL 字段和 HTTP 版本字段。

方法字段可以取几种不同的值，一般有 GET、POST、HEAD、PUT 和 DELETE。一般 GET 方法只被用于向服务器获取数据。 POST 方法用于将实体提交到指定的资源，通常会造成服务器资源的修改。HEAD 方法与 GET 方法类似，但是在返回的响应 中，不包含请求对象。PUT 方法用于上传文件到服务器，DELETE 方法用于删除服务器上的对象。虽然请求的方法很多，但 更多表达的是一种语义上的区别，并不是说 POST 能做的事情，GET 就不能做了，主要看我们如何选择。更多的方法可以参 看文档。

HTTP 响应报文

HTTP 报文有两种，一种是请求报文，一种是响应报文。

HTTP 响应报文的格式如下：

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 137582
Expires: Thu, 05 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 5 August 1996 15:55:28 GMT
Server: Apache 0.84
```

```
<html>
  <body>Hello World</body>
</html>
```

HTTP 响应报文的第一行叫做状态行，后面的行是首部行，最后是实体主体。

状态行包含了三个字段：协议版本字段、状态码和相应的状态信息。

实体部分是报文的主要部分，它包含了所请求的对象。

常见的状态有

200-请求成功、202-服务器端已经收到请求消息，但是尚未进行处理 301-永久移动、302-临时移动、304-所请求的资源未修改、 400-客户端请求的语法错误、404-请求的资源不存在 500-服务器内部错误。

一般 1XX 代表服务器接收到请求、2XX 代表成功、3XX 代表重定向、4XX 代表客户端错误、5XX 代表服务器端错误。

更多关于状态码的可以查看：

《HTTP 状态码》

首部行

首部可以分为四种首部，请求首部、响应首部、通用首部和实体首部。通用首部和实体首部在请求报文和响应报文中都可以设 置，区别在于请求首部和响应首部。

常见的请求首部有 Accept 可接收媒体资源的类型、Accept-Charset 可接收的字符集、Host 请求的主机名。

常见的响应首部有 ETag 资源的匹配信息，Location 客户端重定向的 URI。

常见的通用首部有 Cache-Control 控制缓存策略、Connection 管理持久连接。

常见的实体首部有 Content-Length 实体主体的大小、Expires 实体主体的过期时间、Last-Modified 资源的最后修 改时间。

更多关于首部的资料可以查看：

《HTTP 首部字段详细介绍》

《图解 HTTP》

HTTP/1.1 协议缺点

HTTP/1.1 默认使用了持久连接，多个请求可以复用同一个 TCP 连接，但是在同一个 TCP 连接里面，数据请求的通信次序 是固定的。服务器只有处理完一个请求的响应后，才会进行下一个请求的处理，如果前面请求的响应特别慢的话，就会造成许 多请求排队等待的情况，这种情况被称为“队头堵塞”。队头阻塞会导致持久连接在达到最大数量时，剩余的资源需要等待其他 资源请求完成后才能发起请求。

为了避免这个问题，一个是减少请求数，一个是同时打开多个持久连接。这就是我们对网站优化时，使用雪碧图、合并脚本的 原因。

HTTP/2 协议

2009 年，谷歌公开了自行研发的 SPDY 协议，主要解决 HTTP/1.1 效率不高的问题。这个协议在 Chrome 浏览器上证明 可行以后，就被当作 HTTP/2 的基础，主要特性都在 HTTP/2 之中得到继承。2015 年，HTTP/2 发布。

HTTP/2 主要有以下新的特性：

二进制协议

HTTP/2 是一个二进制协议。在 HTTP/1.1 版中，报文的头信息必须是文本（ASCII 编码），数据体可以是文本，也可以是 二进制。HTTP/2 则是一个彻底的二进制协议，头信息和数据体都是二进制，并且统称为”帧”，可以分为头信息帧和数据帧。 帧的概念是它实现多路复用的基础。

多路复用

HTTP/2 实现了多路复用，HTTP/2 仍然复用 TCP 连接，但是在一个连接里，客户端和服务器都可以同时发送多个请求或回 应，而且不用按照顺序一一发送，这样就避免了”队头堵塞”的问题。

数据流

HTTP/2 使用了数据流的概念，因为 HTTP/2 的数据包是不按顺序发送的，同一个连接里面连续的数据包，可能属于不同的 请求。因此，必须要对数据包做标记，指出它属于哪个请求。HTTP/2 将每个请求或回应的所有数据包，称为一个数据流。每 个数据流都有一个独一无二的编号。数据包发送的时候，都必须标记数据流 ID ，用来区分它属于哪个数据流。

头信息压缩

HTTP/2 实现了头信息压缩，由于 HTTP 1.1 协议不带有状态，每次请求都必须附上所有信息。所以，请求的很多字段都是 重复的，比如 Cookie 和 User Agent ，一模一样的内容，每次请求都必须附带，这会浪费很多带宽，也影响速度。

HTTP/2 对这一点做了优化，引入了头信息压缩机制。一方面，头信息使用 gzip 或 compress 压缩后再发送；另一方面， 客户端和服务器同时维护一张头信息表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引 号，这样就能提高速度了。

服务器推送

HTTP/2 允许服务器未经请求，主动向客户端发送资源，这叫做服务器推送。使用服务器推送，提前给客户端推送必要的资源 ，这样就可以相对减少一些延迟时间。这里需要注意的是 http2 下服务器主动推送的是静态资源，和 WebSocket 以及使用 SSE 等方式向客户端发送即时数据的推送是不同的。

详细的资料可以参考：《HTTP 协议入门》《HTTP/2 服务器推送（Server Push）教程》

HTTP/2 协议缺点

因为 HTTP/2 使用了多路复用，一般来说同一域名下只需要使用一个 TCP 连接。由于多个数据流使用同一个 TCP 连接，遵 守同一个流量状态控制和拥塞控制。只要一个数据流遭遇到拥塞，剩下的数据流就没法发出去，这样就导致了后面的所有数据都 会被阻

塞。HTTP/2 出现的这个问题是由于其使用 TCP 协议的问题，与它本身的实现其实并没有多大关系。

HTTP/3 协议

由于 TCP 本身存在的一些限制，Google 就开发了一个基于 UDP 协议的 QUIC 协议，并且使用在了 HTTP/3 上。 QUIC 协议在 UDP 协议上实现了多路复用、有序交付、重传等等功能

HTTPS 协议

HTTP 存在的问题

1.

HTTP 报文使用明文方式发送，可能被第三方窃听。

2.

3.

HTTP 报文可能被第三方截取后修改通信内容，接收方没有办法发现报文内容的修改。

4.

5.

HTTP 还存在认证的问题，第三方可以冒充他人参与通信。

6.

六星教育 WEB 前端学院面试圣经

HTTPS 简介

HTTPS 指的是超文本传输安全协议，HTTPS 是基于 HTTP 协议的，不过它会使用 TLS/SSL 来对数据加密。使用 TLS/ SSL 协议，所有的信息都是加密的，第三方没有办法窃听。并且它提供了一种校验机制，信息一旦被篡改，通信的双方会立 刻发现。它还配备了身份证书，防止身份被冒充的情况出现。

TLS 握手过程

1.

   第一步，客户端向服务器发起请求，请求中包含使用的协议版本号、生成的一个随机数、以及客户端支持的加密方法。

2.

3.

   第二步，服务器端接收到请求后，确认双方使用的加密方法、并给出服务器的证书、以及一个服务器生成的随机数。

4.

5.

   第三步，客户端确认服务器证书有效后，生成一个新的随机数，并使用数字证书中的公钥，加密这个随机数，然后发给服 务器。并且还会提供一个前面所有内容的 hash 的值，用来供服务器检验。

6.

7.

第四步，服务器使用自己的私钥，来解密客户端发送过来的随机数。并提供前面所有内容的 hash 值来供客户端检验。

8.

9.

第五步，客户端和服务器端根据约定的加密方法使用前面的三个随机数，生成对话秘钥，以后的对话过程都使用这个秘钥 来加密信息。

10.

实现原理

TLS 的握手过程主要用到了三个方法来保证传输的安全。

首先是对称加密的方法 ,对称加密的方法是 ,双方使用同一个秘钥对数据进行加密和解密。但是对称加密的存在一个问题，就 是如何保证秘钥传输的安全性，因为秘钥还是会通过网络传输的，一旦秘钥被其他人获取到，那么整个加密过程就毫无作用了。 这就要用到非对称加密的方法。

非对称加密的方法是，我们拥有两个秘钥，一个是公钥，一个是私钥。公钥是公开的，私钥是保密的。用私钥加密的数据，只 有对应的公钥才能解密，用公钥加密的数据，只有对应的私钥才能解密。我们可以将公钥公布出去，任何想和我们通信的客户， 都可以使用我们提供的公钥对数据进行加密，这样我们就可以使用私钥进行解密，这样就能保证数据的安全了。但是非对称加 密有一个缺点就是加密的过程很慢，因此如果每次通信都使用非对称加密的方式的话，反而会造成等待时间过长的问题。

因此我们可以使用对称加密和非对称加密结合的方式，因为对称加密的方式的缺点是无法保证秘钥的安全传输，因此我们可以 非对称加密的方式来对对称加密的秘钥进行传输，然后以后的通信使用对称加密的方式来加密，这样就解决了两个方法各自存 在的问题。

但是现在的方法也不一定是安全的，因为我们没有办法确定我们得到的公钥就一定是安全的公钥。可能存在一个中间人，截取 了对方发给我们的公钥，然后将他自己的公钥发送给我们，当我们使用他的公钥加密后发送的信息，就可以被他用自己的私钥 解密。然后他伪装成我们以同样的方法向对方发送信息，这样我们的信息就被窃取了，然而我们自己还不知道。

为了解决这样的问题，我们可以使用数字证书的方式，首先我们使用一种 Hash 算法来对我们的公钥和其他信息进行加密生成 一个信息摘要，然后让有公信力的认证中心（简称 CA ）用它的私钥对消息摘要加密，形成签名。最后将原始的信息和签名合 在一起，称为数字证书。当接收方收到数字证书的时候，先根据原始信息使用同样的 Hash 算法生成一个摘要，然后使用公证 处的公钥来对数字证书中的摘要进行解密，最后将解密的摘要和我们生成的摘要进行对比，就能发现我们得到的信息是否被更改 了。这个方法最要的是认证中心的可靠性，一般浏览器里会内置一些顶层的认证中心的证书，相当于我们自动信任了他们，只有 这样我们才能保证数据的安全。

DNS 协议

概 况

六星教育 WEB 前端学院面试圣经

DNS 协议提供的是一种主机名到 IP 地址的转换服务，就是我们常说的域名系统。它是一个由分层的 DNS 服务器组成的分 布式数据库，是定义了主机如何查询这个分布式数据库的方式的应用层协议。DNS 协议运行在 UDP 协议之上，使用 53 号 端口。

域名的层级结构

域名的层级结构可以如下

```
主机名.次级域名.顶级域名.根域名

# 即

host.sld.tld.root
```

根据域名的层级结构，管理不同层级域名的服务器，可以分为根域名服务器、顶级域名服务器和权威域名服务器。

查询过程

DNS 的查询过程一般为 我们首先将 DNS 请求发送到本地 DNS 服务器 ,由本地 DNS 服务器来代为请求。

1. 从”根域名服务器”查到”顶级域名服务器”的 NS 记录和 A 记录（ IP 地址）。
2. 从”顶级域名服务器”查到”次级域名服务器”的 NS 记录和 A 记录（ IP 地址）。
3. 从”次级域名服务器”查出”主机名”的 IP 地址。

比如我们如果想要查询 www.baidu.com 的 IP 地址，我们首先会将请求发送到本地的 DNS 服务器中，本地 DNS 服务 器会判断是否存在该域名的缓存，如果不存在，则向根域名服务器发送一个请求，根域名服务器返回负责 .com 的顶级域名 服务器的 IP 地址的列表。然后本地 DNS 服务器再向其中一个负责 .com 的顶级域名服务器发送一个请求，负责 .com 的顶级域名服务器返回负责 .baidu 的权威域名服务器的 IP 地址列表。然后本地 DNS 服务器再向其中一个权威域名服 务器发送一个请求，最后权威域名服务器返回一个对应的主机名的 IP 地址列表。

DNS 记录和报文

DNS 服务器中以资源记录的形式存储信息，每一个 DNS 响应报文一般包含多条资源记录。一条资源记录的具体的格式为

（Name，Value，Type，TTL）

其中 TTL 是资源记录的生存时间，它定义了资源记录能够被其他的 DNS 服务器缓存多长时间。

常用的一共有四种 Type 的值，分别是 A、NS、CNAME 和 MX ，不同 Type 的值，对应资源记录代表的意义不同。

1.

如果 Type = A，则 Name 是主机名，Value 是主机名对应的 IP 地址。因此一条记录为 A 的资源记录，提供了标 准的主机名到 IP 地址的映射。

2.

3.

如果 Type = NS ,则 Name 是个域名 ,Value 是负责该域名的 DNS 服务器的主机名。这个记录主要用于 DNS 链式 查询时，返回下一级需要查询的 DNS 服务器的信息。

4.

5.

如果 Type = CNAME，则 Name 为别名，Value 为该主机的规范主机名。该条记录用于向查询的主机返回一个主机名 对应的规范主机名，从而告诉查询主机去查询这个主机名的 IP 地址。主机别名主要是为了通过给一些复杂的主机名提供 一个便于记忆的简单的别名。

6.

7.

如果 Type = MX，则 Name 为一个邮件服务器的别名，Value 为邮件服务器的规范主机名。它的作用和 CNAME 是一 样的，都是为了解决规范主机名不利于记忆的缺点。

8.

递归查询和迭代查询

递归查询指的是查询请求发出后，域名服务器代为向下一级域名服务器发出请求，最后向用户返回查询的最终结果。使用递归 查询，用户只需要发出一次查询请求。

迭代查询指的是查询请求后，域名服务器返回单次查询的结果。下一级的查询由用户自己请求。使用迭代查询，用户需要发出 多次的查询请求。

一般我们向本地 DNS 服务器发送请求的方式就是递归查询，因为我们只需要发出一次请求，然后本地 DNS 服务器返回给我 们最终的请求结果。而本地 DNS 服务器向其他

域名服务器请求的过程是迭代查询的过程，因为每一次域名服务器只返回单次 查询的结果，下一级的查询由本地 DNS 服务器自己进行。

DNS 缓存

DNS 缓存的原理非常简单，在一个请求链中，当某个 DNS 服务器接收到一个 DNS 回答后，它能够将回答中的信息缓存在本 地存储器中。返回的资源记录中的 TTL 代表了该条记录的缓存的时间。

DNS 实现负载平衡

DNS 可以用于在冗余的服务器上实现负载平衡。因为现在一般的大型网站使用多台服务器提供服务，因此一个域名可能会对应 多个服务器地址。当用户发起网站域名的 DNS 请求的时候，DNS 服务器返回这个域名所对应的服务器 IP 地址的集合，但在 每个回答中，会循环这些 IP 地址的顺序，用户一般会选择排在前面的地址发送请求。以此将用户的请求均衡的分配到各个不 同的服务器上，这样来实现负载均衡。

HTTP 的几种请求方法用途

HTTP 的几种请求方法用途

GET 方法

发送一个请求来取得服务器上的某一资源

POST 方法

- 向 URL 指定的资源提交数据或附加新的数据

PUT 方法

- 跟 POST 方法很像，也是想服务器提交数据。但是，它们之间有不同。 PUT 指定了资源在服务器上的位置，而 POST 没有

HEAD 方法

- 只请求 面的首部

DELETE 方法

- 删除服务器上的某资源

OPTIONS 方法

- 它用于获取当前 URL 所支持的方法。如果请求成功，会有一个 Allow 的头包含类似 “GET,POST” 这样的信息

TRACE 方法

- TRACE 方法被用于激发一个远程的，应用层的请求消息回路

CONNECT 方法

- 把请求连接转换到透明的 TCP/IP 通道

HTTP 状态码及其含义

1XX ：信息状态码

- 100 Continue 继续，一般在发送 post 请求时，已发送了 http header 之后服务端
- 将返回此信息，表示确认，之后发送具体参数信息

2XX ：成功状态码

- 200 OK 正常返回信息
- 201 Created 请求成功并且服务器创建了新的资源
- 202 Accepted 服务器已接受请求，但尚未处理

3XX ：重定向

- 301 Moved Permanently 请求的网 已永久移动到新位置。
- 302 Found 临时性重定向。
- 303 See Other 临时性重定向，且总是使用 GET 请求新的 URI 。
- 304 Not Modified 自从上次请求后，请求的网 未修改过。

4XX ：客户端错误

- 400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内
- 容发起请求。
- 401 Unauthorized 请求未授权。
- 403 Forbidden 禁止访问。
- 404 Not Found 找不到如何与 URI 相匹配的资源。

5XX: 服务器错误

- 500 Internal Server Error 最常 的服务器端错误。

- 503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

HTTP request 报文结构是怎样的

1. 首行是 Request-Line 包括：请求方法，请求 URI，协议版本，CRLF
2. 首行之后是若干行请求头，包括 general-header，request-header 或者 entity-header， 每个一行以 CRLF 结束
3. 请求头和消息实体之间有一个 CRLF 分隔
4. 根据实际请求需要可能包含一个消息实体 一个请求报文例子如下：
5.
6.

```
GET /Protocols/rfc2616/rfc2616-sec5.html HTTP/1.1
Host: www.w3.org
Connection: keep-alive
Cache-Control: max-age=0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML,
Referer: https://www.google.com.hk/
```

```
8.
9.      Accept-Encoding: gzip,deflate,sdch
10.     Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
11.     Cookie: authorstyle=yes
12.     If-None-Match: "2cc8-3e3073913b100"
13.     If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT
14.     name=qiu&age=25
```

HTTP response 报文结构是怎样的

首行是状态行包括：HTTP 版本，状态码，状态描述，后面跟一个 CRLF 首行之后是若干行响应头，包括：通用头部，响应头部，实体头部 响应头部和响应实体之间用一个 CRLF 空行分隔 最后是一个可能的消息实体 响应报文例子如下：

```
.       HTTP/1.1 200 OK
.       Date: Tue, 08 Jul 2014 05:28:43 GMT
.       Server: Apache/2
.       Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
.       ETag: "40d7-3e3073913b100"
.       Accept-Ranges: bytes
.       Content-Length: 16599
.       Cache-Control: max-age=21600
.       Expires: Tue, 08 Jul 2014 11:28:43 GMT
.       P3P: policyref="http://www.w3.org/2001/05/P3P/p3p.xml"
```

```
Content-Type: text/html; charset=iso-8859-1
{"name": "qiu", "age": 25}
```

Ajax 原理

Ajax 的原理简单来说是在用户和服务器之间加了一个中间层(AJAX 引擎)，通过 XmlHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 javascrip t 来操作 DOM 而更新 面。使用户操作与服务器响应异步化。这其中最关键的一步就是从服务器获得请求数据

Ajax 的过程只涉及 JavaScript 、 XMLHttpRequest 和 DOM 。 XMLHttpRequest 是 aja x 的核心机制

```
/** 1. 创建连接 **/
var xhr = null;
xhr = new XMLHttpRequest()
/** 2. 连接服务器 **/
xhr.open('get', url, true)
/** 3. 发送请求 **/
xhr.send(null);
/** 4. 接受请求 **/
xhr.onreadystatechange = function(){
```

```
10.             if(xhr.readyState == 4){
11.                 if(xhr.status == 200){
12.                     success(xhr.responseText);
13.                 } else {
14.                     /** false **/
15.                     fail && fail(xhr.status);
16.                 }
17.             }
18.             }
```

ajax 有那些优缺点?

ajax 有那些优缺点?

优点：

通过异步模式，提升了用户体验.

优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占用.

Ajax 在客户端运行，承担了一部分本来由服务器承担的工作，减少了大用户量下的服务器负载。

Ajax 可以实现动态不刷新（局部刷新）

缺点：

- 安全问题 AJAX 暴露了与服务器交互的细节。
- 对搜索引擎的支持比较弱。

- 不容易调试。
-

谈一下 WebSocket

由于 http 存在一个明显的弊端（消息只能有客户端推送到服务器端，而服 务器端不能主动推送到客户端），导致如果服务器如果有连续的变化，这时只 能使用轮询，而轮询效率过低，并不适合。于是 WebSocket 被发明出来

相比与 http 具有以下有点

支持双向通信，实时性更强；

可以发送文本，也可以二进制文件；

协议标识符是 ws ，加密后是 wss ；

较少的控制开销。连接创建后， ws 客户端、服务端进行数据交换时，协议控制的数据包头部较小。在不包含头部的情况下，服务端到客户端的包头只有 2~10 字节（取决于数据包 度），客户端到服务端的的话，需要加上额外的 4 字节的掩码。而 HTTP 协议每次通信都需要携带完整的头部；

支持扩展。ws 协议定义了扩展，用户可以扩展协议，或者实现自定义的子协议。（比如支

持自定义压缩算法等）

无跨域问题。

实现比较简单，服务端库如 socket.io 、 ws ，可以很好的帮助我们入 。

而客户端也只需要参照 api 实现即可

ajax、axios、fetch 区别

jQuery ajax

```
$.ajax({
    type: 'POST',
    url: url,
    data: data,
    dataType: dataType,
    success: function () {},
    error: function () {}
});
```

优缺点：

本身是针对 MVC 的编程,不符合现在前端 MVVM 的浪潮

基于原生的 XHR 开发， XHR 本身的架构不清晰，已经有了 fetch 的替代方案

JQuery 整个项目太大，单纯使用 ajax 却要引入整个 JQuery 非常的不合理（采取个性化打包的方案又不能享受 CDN 服务

axios

```
axios({
    method: 'post',
    url: '/user/12345',
```

```
    data: {
        firstName: 'Fred',
        lastName: 'Flintstone'
    }
})
.then(function (response) {
    console.log(response);
})
.catch(function (error) {
    console.log(error);
});
```

优缺点：

- 从浏览器中创建 XMLHttpRequest
- 从 node.js 发出 http 请求
- 支持 Promise API
- 拦截请求和响应
- 转换请求和响应数据
- 取消请求
- 自动转换 JSON 数据
- 客户端支持防止 CSRF/XSRF

fetch

```
try {
    let response = await fetch(url);
    let data = response.json();
    console.log(data);
} catch(e) {
    console.log("Oops, error", e);
}
```

优缺点：

- fetcht 只对网络请求报错，对 400 ， 500 都当做成功的请求，需要封装去处理
- fetch 默认不会带 cookie ，需要添加配置项
- fetch 不支持 abort ，不支持超时控制，使用 setTimeout 及 Promise.reject 的实现的超时控制并不能阻止请求过程继续在后台运行，造成了量的浪费
- fetch 没有办法原生监测请求的进度，而XHR 可以

AJAX 面试题

JavaScript Ajax 读取一个 xml 文档并进行解析的实例

```
var xhr = new XMLHttpRequest;//->在 IE7 以下浏览器中是不兼容的
xhr=new ActiveXObject("Microsoft.XMLHTTP");
xhr=new ActiveXObject("Msxml2.XMLHTTP");
xhr=new ActiveXObject("Msxml3.XMLHTTP");
//->惰性思想
var getXHR = (function () {
```

```
//->存放我们需要的几个获取 Ajax 对象的方法
var ajaxAry = [
    function () {
        return new XMLHttpRequest;
    },
    function () {
        return new ActiveXObject("Microsoft.XMLHTTP");
    },
    function () {
        return new ActiveXObject("Msxml2.XMLHTTP");
    },
    function () {
        return new ActiveXObject("Msxml3.XMLHTTP");
    }
];

//->循环数组,把四个方法依次的执行
var xhr = null;
for (var i = 0; i < ajaxAry.length; i++) {
    //-> 标准浏览器:i=0,获取的是第一个函数 function(){return new XMLHttpRequest;}(A1),执行的时候没有报错,xhr 是它的返回值也是我们的 Ajax 对象,没有报错不会走 catch,执行 getXHR = A1,这样把外面的 getXHR 重写了,遇到 break 循环结束
```

```
        //->IE6 浏览器:i=0,获取第一个函数执行,IE6 不支持 XMLHttpRequest,所以会报错,执行 catch 中的 continue 继续下一次的循环,i=1,获取第二个函数 function(){return new ActiveXObject("Microsoft.XMLHTTP");}(A2),执行没有报错,那么开始执行 getXHR = A2,遇到 break 结束整个循环,此时外面的 getXHR = A2

        var tempFn = ajaxAry[i];
        try {
            xhr = tempFn();
        } catch (e) {
            continue;
        }
        getXHR = tempFn;
        break;
    }
    if (!xhr) {
        throw new Error("你的浏览器版本也太 LOW 了吧,还能不能愉快的玩耍~~");
    }
    return getXHR;
})();

var xhr = getXHR();
```

```
xhr.open("get", "test.txt?_=" + Math.random(), true);
xhr.onreadystatechange = function () {
  if (xhr.readyState === 4 && /^2\d{2}$/.test(xhr.status)) {
    var val = xhr.responseText;
    console.log(val);
  }
};
xhr.send(null);
```

避免返回的数据是乱码的

->前端页面是 UTF-8 编码,如果我们从服务器请求回来的数据不是 UTF-8 编码格式,那么获取到的内容中,中文汉字会出现乱码

->需要我们使用”UTF-8 到底的原则”:前端页面、JS、CSS、后台代码、数据库、请求传递的数据统一都采用一个编码 UTF-8

->[RESPONSE] Content-Type:text/plain(纯文本)、application/json(JSON 格式的)...设定响应主体中内容的格式

前端：设置请求头,获取响应头

->在前端的 JS 中我们可以使用 xhr.setRequestHeader([name],[value]) 设置请求头的信息；

可以使用 `xhr.getResponseHeader([name])/xhr.getAllResponseHeaders()` 获取响应头信息；

服务器端：获取请求头,设置响应头

->在 NODE 中我们可以使用 `response.writeHead(200, {'content-type': 'application/json'})`; 设置响应头信息

->在 NODE 中我们可以使用 request 这个对象获取到请求信息(起始行、首部、主体) 中的都可以获取到

JavaScript Ajax 是什么？ajax 的交互模型？

1) ajax 是什么？

1. 通过异步模式，提升了用户体验
2. 优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占用
3. Ajax 在客户端运行，承担了一部分本来由服务器承担的工作，减少了大用户量下的服务器负载。

2.） Ajax 的最大的特点是什么。

1. Ajax 可以实现动态不刷新（局部刷新）
2. readyState 属性 状态 有 5 个可取值： 0=未初始化 ，1=启动 2=发送，3=接收，4=完成

经 典 总 结

ajax 的全称：Asynchronous Javascript And XML。

异步传输 + js + xml。

异步，在这里简单地解释就是：向服务器发送请求的时候，我们不必等待结果，而是可以同时做其他的事情，等到有了结果它自己会根据设定进行后续操作，与此同时，页面是不会发生整页刷新的，提高了用户体验。

(1)创建 XMLHttpRequest 对象,也就是创建一个异步调用对象

(2)创建一个新的 HTTP 请求,并指定该 HTTP 请求的方法、URL 及验证信息

(3)设置响应 HTTP 请求状态变化的函数

(4)发送 HTTP 请求

(5)获取异步调用返回的数据

(6)使用 JavaScript 和 DOM 实现局部刷新

jQuery 如何创建一个 Ajax

Jquery 的 ajax 方法；

```
//->Ajax
$.ajax({
    url: "test.txt?_=" + Math.random(),
    type: "get",
    dataType: "json",//->text 获取的是一个字符串 json 获取的是一个对象
    success: function (data) {
        console.log(data);
```

```
            //->解析数据和实现数据绑定都可以继续在这个方法中去操作了
        }
    });

    //->JSONP
    $.ajax({
        url:
"http://matchweb.sports.qq.com/kbs/list?columnId=100000&startTime=2016-02-
28&endTime=2016-03-05&_=" + Math.random(),
        type: "get",
        dataType: "jsonp",
        success: function (data) {
            console.log(data);
        }
    });
```

使用 jQuery 的 JSONP 请求,我们不需要自己在 URL 后面加 callback,也不需要自己事先写一个方法传给后台,因为 jQuery 把这些事情都帮我们做了,它默认在 URL 的末尾增加一个 callback=,会自己随机创建一个函数

jQuery111302697077137418091 3_1456989781792,把这个函数传递给后台,后台也会把这个函数给执行了

success 对应的那个匿名函数其实可以理解为就是随机创建的这个函数,data 其实就是我们想要获取的数据

```
//http://matchweb.sports.qq.com/kbs/list?columnId=100000&startTime=2016-02-28&endTime=2016-03-05&_=1456989781793&callback=jQuery11130269707771374180913_1456989781792
$.ajax({
  url: "http://matchweb.sports.qq.com/kbs/list?columnId=100000&startTime=2016-02-28&endTime=2016-03-05&_=" + Math.random(),
  type: "get",
  dataType: "jsonp",
  jsonp: "cb",//->我们把 callback 这个名字也可以修改为自己想要的
  jsonpCallback: "wangHaoYu",//->自己把 callback 后面传递的函数的名字修改为自己想要的名字,不使用自己随机生成的那个长名字
  success: function (data) {
    console.log(data);
  }
});

//->....&cb=wangHaoYu
$.ajax({
  url: "test.txt?_=" + Math.random(),
```

```
        type: "get",
        dataType: "text",
        async: true,//默认就是 true
        success: function (data) {//->请求成功执行的方法,data 就是服务器返回给前端的数据
            console.log(data);
        },
        error: function (message) {//->请求失败执行的方法,message 就是错误信息
        }
    });
```

简述 JavaScript AJAX 的原理

ajax 是很多种技术的集合体。其中包括

浏览器的 xmlHTTPRequest 对象，他是负责为你开通另一条连接通道，可以传递信息。

javascript：他是负责调用 XMLHTTPRequest 对象进行与后台交互的媒介。

xml 是一种数据格式，用于服务器应答传递信息的格式。除了 xml 外，还可以使用任何的文本格式，包括 text，html，json 等。

ajax 并非一种新的技术，而是几种原有技术的结合体。它由下列技术组合而成。

1.

使用 CSS 和 XHTML 来表示。

2.

3.

使用 DOM 模型来交互和动态显示。

4.

5.

使用 XMLHttpRequest 来和服务器进行异步通信。

6.

7.

使用 javascript 来绑定和调用。

8.

在上面几中技术中，除了 XmlHttpRequest 对象以外，其它所有的技术都是基于 web 标准并且已经得到了广泛使用的，XMLHttpRequest 虽然目前还没有被 W3C 所采纳，但是它已经是一个事实的标准，因为目前几乎所有的主流浏览器都支持它。

XMLHttpRequest 是 ajax 的核心机制，它是在 IE5 中首先引入的，是一种支持异步请求的技术。简单的说，也就是 javascript 可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

所以我们先从 XMLHttpRequest 讲起，来看看它的工作原理。

XMLHttpRequest 的属性

它的属性有：

onreadystatechange 每次状态改变所触发事件的事件处理程序。

responseText 从服务器进程返回数据的字符串形式。

responseXML 从服务器进程返回的 DOM 兼容的文档数据对象。

status 从服务器返回的数字代码，比如常见的 404（未找到）和 200（已就绪）

status Text 伴随状态码的字符串信息

readyState 对象状态值

- 0 (未初始化) 对象已建立，但是尚未初始化（尚未调用 open 方法）
- 1 (初始化) 对象已建立，尚未调用 send 方法
- 2 (发送数据) send 方法已调用，但是当前的状态及 http 头未知
- 3 (数据传送中) 已接收部分数据，因为响应及 http 头不全，这时通过 responseBody 和 responseText 获取部分数据会出现错误，
- 4 (完成) 数据接收完毕,此时可以通过通过 responseXml 和 responseText 获取完整的回应数据

但是，由于各浏览器之间存在差异，所以创建一个 XMLHttpRequest 对象可能需要不同的方法。

这个差异主要体现在 IE 和其它浏览器之间。下面是一个比较标准的创建 XMLHttpRequest 对象的方法。

```
function CreateXmlHttp () {
  //非 IE 浏览器创建 XmlHttpRequest 对象
  if (window.XmlHttpRequest) {
```

```
        xmlhttp = new XmlHttpRequest();
    }
    //IE 浏览器创建 XmlHttpRequest 对象
    if (window.ActiveXObject) {
        try {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        catch (e) {
            try {
                xmlhttp = new ActiveXObject("msxml2.XMLHTTP");
            }
            catch (ex) { }
        }
    }
}

function Ustbwuyi () {
    var data = document.getElementById("username").value;
    CreateXmlHttp();
    if (!xmlhttp) {
        alert("创建 xmlhttp 对象异常！");
        return false;
```

```
    }
    xmlhttp.open("POST", url, false);
    xmlhttp.onreadystatechange = function () {
      if (xmlhttp.readyState == 4) {
        document.getElementById("user1").innerHTML = "数据正在加载...";
        if (xmlhttp.status == 200) {
          document.write(xmlhttp.responseText);
        }
      }
    }
    xmlhttp.send();
  }
```

如上所示，函数首先检查 XMLHttpRequest 的整体状态并且保证它已经完成（readyStatus=4），即数据已经发送完毕。

然后根据服务器的设定询问请求状态，如果一切已经就绪（status=200），那么就执行下面需要的操作。

对于 XmlHttpRequest 的两个方法，open 和 send，其中 open 方法指定了：

- a、向服务器提交数据的类型，即 post 还是 get。
-
-

b、请求的 url 地址和传递的参数。

-

-

c、传输方式，false 为同步，true 为异步。默认为 true。如果是异步通信方式(true)，客户机就不等待服务器的响应；如果是同步方式(false)，客户机就要等到服务器返回消息后才去执行其他操作。我们需要根据实际需要来指定同步方式，在某些页面中，可能会发出多个请求，甚至是有组织有计划有队形大规模的高强度的 request，而后一个是会覆盖前一个的，这个时候当然要指定同步方式。

-

Send 方法用来发送请求

知道了 XMLHttpRequest 的工作流程，我们可以看出，XMLHttpRequest 是完全用来向服务器发出一个请求的，它的作用也局限于此，但它的作用是整个 ajax 实现的关键，因为 ajax 无非是两个过程，发出请求和响应请求。

并且它完全是一种客户端的技术。而 XMLHttpRequest 正是处理了服务器端和客户端通信的问题所以才会如此的重要。

现在，我们对 ajax 的原理大概可以有一个了解了。我们可以把服务器端看成一个数据接口，它返回的是一个纯文本流，当然，这个文本流可以是 XML 格式，可以是 Html，可以是 Javascript 代码，也可以只是一个字符串。

这时候，XMLHttpRequest 向服务器端请求这个页面，服务器端将文本的结果写入页面，这和普通的 web 开发流程是一样的，不同的是，客户端在异步获取这个结果后，不是直

接显示在页面，而是先由 javascript 来处理，然后再显示在页面。至于现在流行的很多 ajax 控件，比如 magicajax 等，可以返回 DataSet 等其它数据类型，只是将这个过程封装了的结果，本质上他们并没有什么太大的区别

JavaScript AJAX 的优点

Ajax 的给我们带来的好处大家基本上都深有体会，在这里我只简单的讲几点：

1、最大的一点是页面无刷新，在页面内与服务器通信，给用户的体验非常好。

2、使用异步方式与服务器通信，不需要打断用户的操作，具有更加迅速的响应能力。

3、可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。并且减轻服务器的负担，ajax 的原则是“按需取数据”，可以最大程度的减少冗余请求，和响应对服务器造成的负担。

4、基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

JavaScript AJAX 的缺点

下面我着重讲一讲 ajax 的缺陷，因为平时我们大多注意的都是 ajax 给我们所带来的好处诸如用户体验的提升。而对 ajax 所带来的缺陷有所忽视。

下面所阐述的 ajax 的缺陷都是它先天所产生的。

1、ajax 干掉了 back 按钮，即对浏览器后退机制的破坏。

后退按钮是一个标准的 web 站点的重要功能，但是它没法和 js 进行很好的合作。这是 ajax 所带来的一个比较严重的问题，因为用户往往是希望能够通过后退来取消前一次操作的。

那么对于这个问题有没有办法？答案是肯定的，用过 Gmail 的知道，Gmail 下面采用的 ajax 技术解决了这个问题，在 Gmail 下面是可以后退的，但是，它也并不能改变 ajax 的机制，它只是采用的一个比较笨但是有效的办法，即用户单击后退按钮访问历史记录时，通过创建或使用一个隐藏的 IFRAME 来重现页面上的变更。（例如，当用户在 Google Maps 中单击后退时，它在一个隐藏的 IFRAME 中进行搜索，然后将搜索结果反映到 Ajax 元素上，以便将应用程序状态恢复到当时的状态。）

但是，虽然说这个问题是可以解决的，但是它所带来的开发成本是非常高的，和 ajax 框架所要求的快速开发是相背离的。这是 ajax 所带来的一个非常严重的问题。

2、安全问题

技术同时也对 IT 企业带来了新的安全威胁，ajax 技术就如同对企业数据建立了一个直接通道。这使得开发者在不经意间会暴露比以前更多的数据和服务器逻辑。ajax 的逻辑可以对客户端的安全扫描技术隐藏起来，允许黑客从远端服务器上建立新的攻击。还有 ajax 也难以避免一些已知的安全弱点，诸如跨站点脚步攻击、SQL 注入攻击和基于 credentials 的安全漏洞等。

3、对搜索引擎的支持比较弱。

4、破坏了程序的异常机制。

至少从目前看来，像 ajax.dll，ajaxpro.dll 这些 ajax 框架是会破坏程序的异常机制的。关于这个问题，我曾经在开发过程中遇到过，但是查了一下网上几乎没有相关的介绍。后来我自己做了一次试验，分别采用 ajax 和传统的 form 提交的模式来删除一条数据……给我们的调试带来了很大的困难。

5、另外，像其他方面的一些问题，比如说违背了 url 和资源定位的初衷。例如，我给你一个 url 地址，如果采用了 ajax 技术，也许你在该 url 地址下面看到的和我在这个 url 地址下看到的内容是不同的。这个和资源定位的初衷是相背离的。

6、一些手持设备（如手机、PDA 等）现在还不能很好的支持 ajax，比如说我们在手机的浏览器上打开采用 ajax 技术的网站时，它目前是不支持的，当然，这个问题和我们没太多关系。

JavaScript AJAX 的几种框架

jQuery

jQuery 是一个快速、简洁的 JavaScript 框架

jQuery 是对原生 XHR 的封装，另外还增加了 jsonp 的支持，让 ajax 请求可以支持跨域请求，

但是要注意的是：jsonp 请求本质不是 XHR 异步请求，就是请求了一个 js 文件，因此在浏览器的 network 面板中的 xhr 标签下看不到 jsonp 的跨域请求，但是在 js 标签下能看见，因为它利用 src 特性请求任何一个网站的资源。

Axios

特性

从浏览器中创建 XMLHttpRequest

从 node.js 发出 http 请求

支持 Promise API

拦截请求和响应

转换请求和响应数据

取消请求

自动转换 JSON 数据

客户端支持防止 CSRF/XSRF

fetch

Fetch API 是近年来被提及将要取代 XHR 的技术新标准，是一个 HTML5 的 API。

Fetch 并不是 XHR 的升级版本，而是从一个全新的角度来思考的一种设计。

Fetch 是基于 Promise 语法结构，而且它的设计足够低阶，这表示它可以在实际需求中进行更多的弹性设计。对于 XHR 所提供的能力来说，Fetch 已经足够取代 XHR，并且提供了更多拓展的可能性。

JavaScript AJAX 的过程是怎么样的

(1)创建 XMLHttpRequest 对象,也就是创建一个异步调用对象.

(2)创建一个新的 HTTP 请求,并指定该 HTTP 请求的方法、URL 及验证信息.

(3)设置响应 HTTP 请求状态变化的函数.

(4)发送 HTTP 请求.

(5)获取异步调用返回的数据.

(6)使用 JavaScript 和 DOM 实现局部刷新.

JavaScript 简述 Ajax 异步机制，Ajax 有哪些的好处和弊端，介绍下 Ajax 异步请求的原理和过程

ajax 的缺点

1、ajax 不支持浏览器 back 按钮。

2、安全问题 AJAX 暴露了与服务器交互的细节。

3、对搜索引擎的支持比较弱。

4、破坏了程序的异常机制。

5、不容易调试。

JavaScript XMLHttpRequest 是什么、怎样完整地执行一次 GET 请求、怎样检测错误

XMLHttpRequest 是 ajax 的核心机制，它是在 IE5 中首先引入的，是一种支持异步请求的技术。简单的说，也就是 javascript 可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

所以我们先从 XMLHttpRequest 讲起，来看看它的工作原理。

XMLHttpRequest 属性

它的属性有：

onreadystatechange 每次状态改变所触发事件的事件处理程序。

responseText 从服务器进程返回数据的字符串形式。

responseXML 从服务器进程返回的 DOM 兼容的文档数据对象。

status 从服务器返回的数字代码，比如常见的 404（未找到）和 200（已就绪）

status Text 伴随状态码的字符串信息

readyState 对象状态值

- 0 (未初始化) 对象已建立，但是尚未初始化（尚未调用 open 方法）
-
-

1 (初始化) 对象已建立，尚未调用 send 方法

2 (发送数据) send 方法已调用，但是当前的状态及 http 头未知

3 (数据传送中) 已接收部分数据，因为响应及 http 头不全，这时通过 responseBody 和 responseText 获取部分数据会出现错误，

4 (完成) 数据接收完毕,此时可以通过通过 responseXml 和 responseText 获取完整的回应数据 但是，由于各浏览器之间存在差异，所以创建一个 XMLHttpRequest 对象可能需要不同的方法。这个差异主要体现在 IE 和其它浏览器之间。下面是一个比较标准的创建 XMLHttpRequest

```
var xmlhttp;
function loadXMLDoc (url) {
xmlhttp = null;
if (windows.XMLHttpRequest) {
xmlhttp = new XMLHttpRequest();
```

```
6.          } else if (window.ActiveXObject) {
7.          //For IE5 IE6
8.          xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
9.          }
10.
11.         if (xmlhttp != null) {
12.         xmlhttp.onreadystatechange = stateChange;
13.         xmlhttp.open("GET", url, true)
14.         xmlhttp.send(null);
15.         } else {
16.         alert("不支持 XMLHTTP")
17.         }
18.         }
19.
20.         function stateChange () {
21.         // 4 'load'
22.         if (xmlhttp.readState == 4) {
23.         // 200 'load'
24.         if (xmlhttp.status == 200) {
25.         //...
26.         } else {
27.         alert("problem retrieving XML data")
```

```
28.                 }
29.                 }
30.                 }
```

JavaScript flash，ajax 各自的优缺点。在使用中如何取舍

Flash ajax 对比

Flash 适合处理多媒体、矢量图形、访问机器；对 CSS、处理文本上不足，不容易被搜索。

Ajax 对 CSS、文本支持很好，支持搜索；多媒体、矢量图形、机器访问不足。

共同点：与服务器的无刷新传递消息、用户离线和在线状态、操作 DOM

JavaScript 如何得到 HTTP 的请求头信息和返回的头信息

Javascript 中跟 response header 有关的就两个方法：

getResponseHeader 从响应信息中获取指定的 http 头 语法

```
strValue = oXMLHttpRequest.getResponseHeader(bstrHeader);
```

getAllResponseHeaders 获取响应的所有 http 头 语法

```
strValue = oXMLHttpRequest.getAllResponseHeaders();
```

需要注意的是，通常，在 IE 下不能完整的获取 header 报头数据，只能取到如下 header 数据：

. X-Powered-By:

- X-UA-Compatible:
- Keep-Alive:
- Transfer-Encoding:
- Content-Type:

比如你要获取时间戳，在 IE 下必须做些特殊处理，需要在后端设置一下，关闭缓存：

```
header( 'Cache-Control: no-store');
var req = new XMLHttpRequest();
req.open('GET', document.location, false);
req.send(null);
var header = req.getAllResponseHeaders().toLowerCase();
console.log(Headers);
```

JavaScript 页面编码和被请求的资源编码如果不一致如何处理

比如：http://www.yyy.com/a.html 中嵌入了一个 http://www.xxx.com/test.js

a.html 的编码是 gbk 或 gb2312 的。 而引入的 js 编码为 utf-8 的 ，那就需要在引入的时候

```
<script src="http://www.xxx.com/test.js" charset="utf-8"></script>
```

同理，如果你的页面是 utf-8 的，引入的 js 是 gbk 的，那么就需要加上 `charset="gbk"`.

JavaScript AJAX 同步和异步的区别

同步：提交请求->等待服务器处理->处理完毕返回，这个期间客户端浏览器不能干任何事情；

异步：请求通过事件触发->服务器处理（这时浏览器仍然可以作其他事情）->处理完毕

Ajax.open 方法中，第三个参数是设置同步或者异步的；

同步和异步的区别；

同步：浏览器访问服务器请求，用户看得到页面刷新，重新发请求,等请求完，页面刷新，新内容出现，用户看到新内容,进行下一步操作。

异步：浏览器访问服务器请求，用户正常操作，浏览器后端进行请求。等请求完，页面不刷新，新内容也会出现，用户看到新内容。

JavaScript GRT 和 POET 区别？何时使用 post？

GET：一般用于信息获取，使用 URL 传递参数，对所发送信息的数量也有限制，一般在 2000 个字符

POST：一般用于修改服务器上的资源，对所发送的信息没有限制。

GET 方式需要使用 `Request.QueryString` 来取得变量的值，而 POST 方式通过 `Request.Form` 来获取变量的值，

也就是说 Get 是通过地址栏来传值，而 Post 是通过提交表单来传值。

然而，在以下情况中，请使用 POST 请求：

无法使用缓存文件（更新服务器上的文件或数据库）

向服务器发送大量数据（POST 没有数据量限制）

发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

JavaScript get 为什么比 post 性能好，php/node 都用一个东西接收 post 和 get 请求，怎么解释 get 比 post 性能好

所以如果你希望

请求中的 URL 可以被手动输入

请求中的 URL 可以被存在书签里，或者历史里，或者快速拨号里面，或者分享给别人。

请求中的 URL 是可以被搜索引擎收录的。

带云压缩的浏览器，比如 Opera mini/Turbo 2，只有 GET 才能在服务器端被预取的。

请求中的 URL 可以被缓存。

请使用 GET.

HTTP 请求方式

- get 从服务器获取
- post 向服务器发送

- 注册页面:用户填完信息,有很多的信息,我们把这些信息都获取到,然后通过 Ajax 中的 POST 请求,在请求主体中把这些信息都传递给后台服务器,服务器把这些信息存储到数据库中(“表单提交”)

- put 增加
- delete 删除
- head、options、trace、connection、track...

get 系的特征(get、delete、head)

1、会把给服务器发送的数据放到 url 后面

->http://matchweb.sports.qq.com/kbs/hotMatchList?callback=getHotMatchList&_=1456903174906 “?后面写的内容都是同过 get 请求传递给服务器的”

2、有大小限制

- 因为 get 系把数据都放到 url 里面，而浏览器会对 url 的长度有大小限制， 所以造成了 get 系对传输的数据大小有限制
- ie->2K
- firefox->7k
- chrome->8k

3、会被缓存(服务器和浏览器自带的缓存)

- 我们一般不用缓存,因为自带的缓存很多情况导致,第二次请求的内容一直和第一次一样,最新的内容请求不回来
- 如果想要快,我们做的所有的缓存都是自己单独用程序设计出来的而不是浏览器自带的缓存机制

4、不安全,因为明文发送

- 在浏览器的控制台中我们可以查看到 URL 后面的参数值,所以不安全

post 系(post、 put)

- 1、会把给服务器发送的数据放到请求主体里
- 2、不会有大小限制

  因为 post 系把数据放到请求主体里，而请求主体是没有大小限制的
- 3、永远不会被缓存
- 4、相对于 get 系安全许多,因为不是明文发送

为什么 get 系会被缓存而 post 系不会被缓存?

1、因为 get 系设计的初衷是用来从服务器拉取数据的。含有两个特点：

- 数据可能很大，数据可能会重复。正因为有这两个特点，所以造成了 get 系会被缓存
- post 系设计的初衷是往服务器发送数据的,所以不需要缓存

HEAD、DELETE 这两个方法比较特殊：他们成功只返回 202，不会返回 200

WEB 应用从服务器主动推送 Data 到客户端有哪些方式

html5 websoket

WebSocket 通过 Flash

XHR 长时间连接

XHR Multipart Streaming

不可见的 Iframe

<script>标签的长时间连接(可跨域)

JavaScript 跨域怎么实现?如何解决跨域问题

jsonp、jsonp 的原理是动态插入 script 标签

document.domain+iframe、

window.name、window.postMessage、

服务器上设置代理页面;

具体的如下；

javascript 跨域有两种情况：

- 1、基于同一父域的子域之间，如：a.c.com 和 b.c.com
- 2、基于不同的父域之间，如：www.a.com 和 www.b.com
- 3、端口的不同，如：www.a.com:8080 和 www.a.com:8088
- 4、协议不同，如：http://www.a.com 和 https://www.a.com

对于情况 3 和 4，需要通过后台 proxy 来解决，具体方式如下：

- a、在发起方的域下创建 proxy 程序
- b、发起方的 js 调用本域下的 proxy 程序
- c、proxy 将请求发送给接收方并获取相应数据

- d、proxy 将获得的数据返回给发起方的 js

代码和 ajax 调用一致，其实这种方式就是通过 ajax 进行调用的

而情况 1 和 2 除了通过后台 proxy 这种方式外，还可以有多种办法来解决：

1、document.domain+iframe（只能解决情况 1）：

- a、在发起方页面和接收方页面设置 document.domain，并将值设为父域的主域名(window.location.hostname)
- b、在发起方页面创建一个隐藏的 iframe，iframe 的源是接收方页面
- c、根据浏览器的不同，通过 iframe.contentDocument || iframe.contentWindow.document 来获得接收方页面的内容
- d、通过获得的接收方页面的内容来与接收方进行交互

这种方法有个缺点，就是当一个域被攻击时，另一个域会有安全漏洞出现。

```
//发起方
document.domain = 'a.com';
var src = document.getElementById('txtSrc').value;
var ifr = document.createElement('iframe');
ifr.src = src;
ifr.style.display = 'none';
document.body.appendChild(ifr);
function GetDataFromDomain () {
  var doc = ifr.contentDocument || ifr.contentWindow.document;
```

```
    console.log(doc.getElementById('data').value)
}

//接收方
document.domain = 'a.com'
```

```
//发起方
document.domain = 'a.com';
var src = document.getElementById('txtSrc').value;
var ifr = document.createElement('iframe');
ifr.src = src;
ifr.style.display = 'none';
document.body.appendChild(ifr);
function GetDataFromDomain() {
    var doc = ifr.contentDocument || ifr.contentWindow.document;
    alert(doc.getElementById("data").value);
}
//接收方
document.domain = 'a.com';
```

2、 动态创建 script(也就是 jsonp)

a、在发起方页面动态加载一个 script ,script 的 URL 指向接收方的一个处理地址(后台),该地址返回的 javascript 方法会被执行，另外 URL 中可以传入一些参数，该方法只支持 GET 方式提交参数。

b、加载的 script 可以在调用跨域 js 方法后再做一些自己的处理

```
//发起方
function load_script (callback) {
var head = document.getElementsByTagName('head')[0];
```

```
var script = document.createElement('script');
var src = document.getElementById('txtSrc').value;

script.type = 'text/javascript';
script.src = src;

// 借鉴 jquery 的 script 跨域方法
script.onload = script.onreadystatechange = function () {
if (
  !this.readyState ||
  this.readyState === 'loaded' ||
  this.readyState == 'complete'
) {
  callback && callback();
  script.onload = script.onreadystatechange = null;
  if (head && script.parentNode) {
    head.removeChild(script);
  }
}
}
head.insertBefore(script, head.firstChild);
}
```

```
//发起方
function load_script(callback){
    var head = document.getElementsByTagName('head')[0];
    var script = document.createElement('script');
    var src = document.getElementById('txtSrc').value;
    script.type = 'text/javascript';
    script.src = src;
    //借鉴了jQuery的script跨域方法
    script.onload = script.onreadystatechange = function(){
    if((!this.readyState||this.readyState === "loaded"||this.readyState === "complete")){
            callback && callback();
            // Handle memory leak in IE
            script.onload = script.onreadystatechange = null;
            if (head && script.parentNode) {
                head.removeChild( script );
            }
        }
    }
    // Use insertBefore instead of appendChild to circumvent an IE6 bug.
    head.insertBefore(script,head.firstChild);
}
//接收方
protected void Page_Load(object sender, EventArgs e)
{
    Response.Clear();
    Response.ContentType = "application/x-javascript";
    Response.Write(String.Format(@"alert('{0}');", DateTime.Now));
    Response.End();
}
```

3、location.hash+iframe：

- a、发起方创建一个隐藏的 iframe，iframe 的源指向接收方的页面，并通过接收方页面的 hash 值来传送数据
- b、发起方创建一个定时器，定时检查自己的 location.hash 并作相应的处理
- c、接收方创建一个隐藏的 iframe，iframe 的源指向发起方所在域的一个代理页面，并将接收方根据发起方传入的数据而处理后的数据通过代理页面的 hash 值来传送
- d、接收方创建一个定时器，定时检查自己的 location.hash 并作相应的处理
- e、代理页面创建一个定时器，定时检查自己的 location.hash 并同步更新发起方页面的 hash 值 www.a.com/a.html#aaa，其中#aaa 就是 location.hash 值

六星教育 WEB 前端学院面试圣经

```
//发起方
function addHash() {
    var src = document.getElementById('txtSrc').value;
    if (src.length > 0) {
        changeHash(src);
    }
}
function changeHash(src) {
    if (document.getElementById('ifr1')) {
        var ifr = document.getElementById('ifr1');
        ifr.src = 'http://www.b.com/Test/HashTest2.htm#' + src;
    }else {
        var ifr = document.createElement('iframe');
        ifr.setAttribute('id', 'ifr1');
        ifr.src = 'http://www.b.com/Test/HashTest2.htm#' + src;
        ifr.style.display = 'none';
        document.body.appendChild(ifr);
    }
}
function checkHash() {
    if (location.hash && location.hash.length > 1) {
        changeHash(location.hash.substring(1));
    }
}
setInterval(checkHash, 2000);
//发起方代理
function checkHash() {
    if (parent && parent.parent && parent.parent.location && self.location.hash.length > 1) {
        parent.parent.location.hash = self.location.hash.substring(1);
    }
}
setInterval(checkHash, 500);
```

```
//接收方
function checkHash() {
    if (location.hash && location.hash.length > 1) {
        var hashData = location.hash.substring(1);
        var ifr = null;
        if (document.getElementById('ifr2')) {
            ifr = document.getElementById('ifr2');
        }else {
            ifr = document.createElement('iframe');
            ifr.setAttribute('id', 'ifr2');
            ifr.style.display = 'none';
            document.body.appendChild(ifr);
        }
        switch (hashData) {
            case '1':
                alert('One');
                if (ifr) {
                    ifr.src = 'http://www.a.com/test/HashTest3.htm#2';
                }
            break;
            case '2':
                alert('Two');
                if (ifr) {
                    ifr.src = 'http://www.a.com/test/HashTest3.htm#1';
                }
            break;
            default: break;
        }
    }
}
setInterval(checkHash, 2000);
```

4、window.name：

- a、发起方页面创建一个隐藏的 iframe，并且源指向接收方页面
- b、接收方在自己页面通过 script 将需要传送的数据放入 window.name 里
- c、发起方在 iframe 的 onload 方法里将 iframe 的源改为和自己在同一个域下的代理页面(因为只能是同一个域下才能访问 window.name 的值)
- d、获取 window.name 的值(虽然 iframe 的源改变了，但是 window.name 的值不会变)
- window.name 的值差不多可以有 2MB 大小

```
//发起方
var ischanged = false;
function changeSrc() {
    if (document.getElementById('ifr1')) {
        var ifr = document.getElementById('ifr1');
        if (!ischanged) {
            ischanged = true;
            ifr.contentWindow.location = 'http://www.a.com/Test/NameTest3.htm';
        }else {
            var data = ifr.contentWindow.name;
            alert(data);
        }
    }else {
        var ifr = document.createElement('iframe');
        ifr.setAttribute('id', 'ifr1');
        ifr.src = 'http://www.b.com/Test/NameTest2.htm';
        ifr.style.display = 'none';
        document.body.appendChild(ifr);
    }
}
function getData() {
    setInterval(changeSrc, 2000);
}
//接收方
window.name = 'NameTest2';
```

5、HTML5 的 postMessage

-

a、receiverWindow.postMessage(msg, targetOrigin)，receiverWindow 就是对接收消息的 window 的引用，可以是 iframe 的 contentWindow/window.open 的返回值/window.frames 中的一个；msg 就是要发送的消息，string 类型；targetOrigin 用于限制 receiverWindow 的 URI，包括主域名和端口，使用“*”表示无限制，但是为了安全起见还是需要设置下，以防把消息发送给恶意的网站，如果 targetOrigin 的 URI 和 receiverWindow 的不符，则放弃发送消息。

b、接收方通过 message 事件来获得消息，并且通过 event.origin 的属性来验证发送方并通过 event.data 来获得传送的消息内容 ,event.source 来获得发送方的 window 对象

```
<iframe id="ifr" style="display:none" src="http://www.b.com/Test/PostMessageTest2.htm"></iframe
//发起方
function getData() {
    var ifr = document.getElementById('ifr');
    var targetOrigin = 'http://www.b.com';
    if (ifr.contentWindow.postMessage) {
        ifr.contentWindow.postMessage('PostMessageTest2', targetOrigin);
    }
}
//接收方
window.addEventListener('message', function(event) {
    if (event.origin == 'http://www.a.com') {
        alert(event.data);
        alert(event.source);
    }
}, false);
```

6、window.opener

适用于 IE6、7，也就是 operner hack 方法，不过貌似现在已经不管用了，只要打过微软的安全补丁.kb2497640 就不能用了

- a、发起方页面创建一个隐藏的 iframe，并且源指向接收方页面
- b、发起方页面通过 iframe.content `Window.opener = {a: function(params){...}, b: function(params){...} ...}`来定义可被接收方调用的方法
- c、接收方页面通过 `window.opener.a/window.opener.b` 来调用发起方定义的方法
- d、接收方页面通过 `parent.opener = {c: function(params){...}, d: function(params){...} ...}`来定义可被发起方调用的方法
- e、发起方页面通过 `opener.c/opener.d` 来调用接收方定义的方法

其实原理就是重置 opener 对象

```
//发起方
<iframe id="ifr" src="http://www.b.com/test/OpenerTest2.htm" style="display:none"></iframe>
var ifr = document.getElementById('ifr');
ifr.contentWindow.opener = { a: function(msg) { alert('我调用了a方法获得了消息：' + msg); } }
//接收方
window.opener.a('aaa');
```

7、window.navigator

适用于 IE6、7，貌似现在还能用，还没被补丁掉

- a、发起方页面创建一个隐藏的 iframe，并且源指向接收方页面
- b、发起方页面通过 window.navigator.a = function(params){ … }; window.navigator.b = function(params){…}; 来定义被接 收方调用的方法
- c、接收方页面通过 window.navigator.a(params); window.navigator.b(params); 来调用发起方定义的方法
- d、接收方页面通过 window.navigator.c = function(params){ … }; window.navigator.d = function(params){…}; 来定义被发起方调用的方法
- e、发起方页面通过 window.navigator.c(params); window.navigator.d(params); 来调用接收方定义的方法

```
//发起方
<iframe id="ifr" src="http://www.b.com/test/NavigatorTest2.htm" style="display:none"></iframe>
window.navigator.a = function(msg) { alert('我调用了a方法获得了消息: ' + msg); }
window.navigator.b = function(msg) { alert('我调用了b方法获得了消息: ' + msg); }
setInterval(function() { window.navigator.c('ccc'); }, 2000);
setInterval(function() { window.navigator.d('ddd'); }, 2000);
//接收方
window.navigator.c = function(msg) { alert('我调用了c方法获得了消息: ' + msg); }
window.navigator.d = function(msg) { alert('我调用了d方法获得了消息: ' + msg); }
setInterval(function() { window.navigator.a('aaa'); }, 2000);
setInterval(function() { window.navigator.b('bbb'); }, 2000);
```

JavaScript jsonP 有哪几种方式？

JSONP 原理及实现跨域方式?

后台给了个接口：https://a.a.com/a/a.json,我页面的上线地址是：http://b.b.com。显而易见，因为浏览器同源策略的限制，通过 ajax 无法无法取得 json 的数据。

同源策略，它是由 Netscape 提出的一个著名的安全策略。现在所有支 JavaScript 的浏览器都会使用这个策略。所谓同源是指，域名，协议，端口相同。

同源策略限制了我们无法通过原生的 `XMLHttpRequest()` 对象获取到 json 数据。为了突破这个限制，我们的前辈们想出了一个解决方案：jsonp。

jsonp 并非新的数据格式，而是解决 JSON 跨域获取的解决方案。通过 JSONP 获取到得数据已经不是 JSON 了，而是 JS 类型的数据（大部分是对象）。

上网找过很多讲 jsonp 的文章，大部分都是讲的模模糊糊的。jsonp 的原理其实不复杂：

1、浏览器的同源策略把跨域请求都禁止了；

2、HTML 的 `<script>` 标签是例外，可以突破同源策略从其他来源获取数据；

3、由上可得，我们可以通过 `<script>` 标签引入 jsonp 文件，然后通过一系列 JS 操作获取数据。

不仅如此，我们还发现凡是拥有" src" 这个属性的标签都拥有跨域的能力，比如 `<script>`、`<img>`、`<iframe>`

上面三点便是 JSONP 实现跨域的原理。

原理我们知道了，该怎么实现这些操作呢？

接下来轮到 jQuery 登场！JQ 已经帮我们封装好了

```
$.ajax({
  dataType:'jsonp',
  jsonp:'jsonp_callback',
```

```
        url:'http://www.baidu.com/xxx.jsonp',
        success:function(){
            //dosomthing
        }
    });
```

严格的 jquery 代码；

```
    jQuery(document).ready(function(){
        $.ajax({
            type: "get",
            async: false,
            url: "http://flightQuery.com/jsonp/flightResult.aspx?code=CA1998",
            dataType: "jsonp",
            jsonp: "callback",//传递给请求处理程序或页面的 ,用以获得 jsonp 回调函数名的参数名(一般默认为:callback)
            jsonpCallback:"flightHandler",//自定义的 jsonp 回调函数名称，默认为 jQuery 自动生成的随机函数名，也可以写"?"，jQuery 会自动为你处理数据
            success: function(json){
                alert('您查询到航班信息：票价： ' + json.price + ' 元，余票： ' + json.tickets + ' 张。');
            },
            error: function(){
                alert('fail');
```

```
        }
    });
});
```

原生 JS demo：

```
function jsonHandle (url) {
  var script = document.createElement("script");
  script.setAttribute("src", url);
  document.getElementsByTagName("body")[0].appendChild(script);
}
//JS 插入之后就可以处理数据了
```

Ajax 和 jsonp 的区别

1、ajax 和 jsonp 这两种技术在调用方式上“看起来”很像，目的也一样，都是请求一个 url，然后把服务器返回的数据进行处理，因此 jquery 和 ext 等框架都把 jsonp 作为 ajax 的一种形式进行了封装；

2、但 ajax 和 jsonp 其实本质上是不同的东西。ajax 的核心是通过 XmlHttpRequest 获取非本页内容，而 jsonp 的核心则是动态添加`<script>`标签来调用服务器提供的 js 脚本。

3、所以说，其实 ajax 与 jsonp 的区别不在于是否跨域，ajax 通过服务端代理一样可以实现跨域，jsonp 本身也不排斥同域的数据的获取。

4、还有就是，jsonp 是一种方式或者说非强制性协议，如同 ajax 一样，它也不一定非要用 json 格式来传递数据，如果你愿意，字符串都行，只不过这样不利于用 jsonp 提供公开服务。

总而言之，jsonp 不是 ajax 的一个特例，哪怕 jquery 等巨头把 jsonp 封装进了 ajax，也不能改变这一点！；

http://www.XXXX.com:80/

1、协议、域名/IP、端口号 2、同源策略：上面的三个完全一致是同源

我们当前页面的地址(http://www.aaa.com/)和我们要向服务器请求数据的地址(http://www.XXX.com/data/userInfo?name=aaa&age=13)在同源下,这就是同源策略,项目中我们在同源下请求数据使用 JS 中的 Ajax 技术

非同源(跨域)策略：上面的三个有一个不一样就是非同源

我们当前页面的地址和请求数据的服务器地址不是同一个源,

例如:当前地址是 http://www.XXX.com/,请求数据的地址 http://www.baidu.com/data/videoInfo,此时我们是跨域请求(非同源策略),在项目中目前最常用的解决跨域的方式是 JS 中的 JSONP 技术

具体的目录文件和 URL 后面的参数值

后面的都是传递给当前页面的参数值(问号传参),如果需要传递多个参数,中间中&隔开,#是当前页面的锚点定位

六星教育 WEB 前端学院面试圣经

https://www.baidu.com:80/user/index.html?name=aaa&age=17&sex=1#userName

(#userName 锚点定位,定位到当前页面中 ID 为 userName 这个标签的位置)

JavaScript 用 a.com 引用了网页 b.com 的 js；js 是否能读 a.com。能够读 b.com

1)如果当前页面是 A 页面,那么我们可以在当前 A 页面中通过 iframe 把需要展示的 B 页面嵌入进来

2)我们还可以在 A 页面中操作 B 页面中的内容

AJAX 的状态码有哪些

readyState 属性 状态 有 5 个可取值

- 0=未初始化
- 1=启动
- 2=发送
- 3=接收
- 4=完成

HTTP 状态码有哪些？分别代表什么意思？

100-199 用于指定客户端应相应的某些动作。

200-299 用于表示请求成功。

300-399 用于已经移动的文件并且常被包含在定位头信息中指定新的地址信息。

400-499 用于指出客户端的错误。400 1、语义有误，当前请求无法被服务器理解。401 当前请求需要用户验证 403 服务器已经理解请求，但是拒绝执行它。

500-599 用于支持服务器错误。 503 – 服务不可用;

常用的

- 100 Continue 继续，一般在发送 post 请求时，已发送了 http header 之后服务端将返回此信息，表示确认，之后发送具体参数信息
- 200 OK 正常返回信息
- 201 Created 请求成功并且服务器创建了新的资源
- 202 Accepted 服务器已接受请求，但尚未处理
- 301 Moved Permanently 请求的网页已永久移动到新位置。
- 302 Found 临时性重定向。
- 303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。
- 304 Not Modified 自从上次请求后，请求的网页未修改过。
- 400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。
- 401 Unauthorized 请求未授权。
- 403 Forbidden 禁止访问。
- 404 Not Found 找不到如何与 URI 相匹配的资源。
- 500 Internal Server Error 最常见的服务器端错误。
- 503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）

http 状态吗 100,200,300,400,500 分别代表什么意思？

100 Continue 继续，一般在发送 post 请求时，已发送了 http header 之后服务端将返回此信息，表示确认，之后发送具体参数信息

200 OK 正常返回信息

400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

500 Internal Server Error 最常见的服务器端错误。

聊聊 cache-control

网页的缓存是由 HTTP 消息头中的“Cache-control”来控制的，常见的取值 有 private、no-cache、max-age、must-revalidate 等，默认为 private。

Expires 头部字段提供一个日期和时间，响应在该日期和时间后被认为失效。允许客户端在这个时间之前不去检查（发请求），等同 max-age 的效果。但是如果同时存在，则被 Cache-Control 的 max-age 覆盖。

```
Expires = "Expires" ":" HTTP-date
```

例如

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```
（必须是 GMT 格式）

如果把它设置为-1，则表示立即过期

Expires 和 max-age 都可以用来指定文档的过期时间，但是二者有一些细微差别

1.Expires 在 HTTP/1.0 中已经定义，Cache-Control:max-age 在 HTTP/1.1 中才有定义，为了向下兼容，仅使用 max-age 不够；

2.Expires 指定一个绝对的过期时间(GMT 格式),这么做会导致至少 2 个问题 :1)客户端和服务器时间不同步导致 Expires 的配置出现问题。 2) 很容易在配置后忘记具体的过期时间，导致过期来临出现浪涌现象；

3.max-age 指定的是从文档被访问后的存活时间，这个时间是个相对值(比如:3600s),相对的是文档第一次被请求时服务器记录的 Request_time(请求时间)

4.Expires 指定的时间可以是相对文件的最后访问时间(Atime)或者修改时间(MTime),而 max-age 相对对的是文档的请求时间(Atime)

5.如果值为 no-cache,那么每次都会访问服务器。如果值为 max-age,则在过期之前不会重复访问服务器。

实现一个页面操作不会整页刷新的网站，并且能在浏览器前进、后退时正确响应。给出你的技术实现方案？

相较于不同页面的跳转，AJAX 可以说大大提高了用户的浏览体验，不用看到页面切换之间的白屏是件很惬意的事情。但是很多早先的 AJAX 应用是不支持浏览器的前进后退的，这导致了用户不管在网站里浏览到何处，一旦刷新就会立刻回到起初的位置，并且用户也无法通过浏览器的前进后退按钮来实现浏览历史的切换。

对于第一个问题，解决还算容易，只要用 cookie 或者 localStorage 来记录应用的状态即可，刷新页面时读取一下这个状态，然后发送相应 ajax 请求来改变页面即可。但是第二个问题就很麻烦了，先说下现代浏览器的解决方案。

六星教育 WEB 前端学院面试圣经

JavaScript 使用 AJax Post 方式向页面 check.do 发送请求；请求的数据是 userID=“admin”，password=“ABC”，假设服务器返回”OK”是成功，客户端踏出”验证通过提示框”

```
$.post{
"check.do",
{'UserId':"Admin","Password":"ABC"},
function(data){
if(data=="OK"){
alert("验证通过");
}
},
"text"
}
```

传输层重点

多路复用与多路分解

多路复用与多路分解

将传输层报文段中的数据交付到正确的套接字的工作被称为多路分解。

在源主机上从不同的套接字中收集数据，封装头信息生成报文段后，将报文段传递到网络层，这个过程被称为多路复用。

无连接的多路复用和多路分解指的是 UDP 套接字的分配过程，一个 UDP 套接字由一个二元组来标识，这个二元组包含了一 个目的地址和一个目的端口号。因此不同源地址和端口号的 UDP 报文段到达主机后，如果它们拥有相同的目的地址和目的端 口号，那么不同的报文段将会转交到同一个 UDP 套接字中。

面向连接的多路复用和多路分解指的是 TCP 套接字的分配过程，一个 TCP 套接字由一个四元组来标识，这个四元组包含了 源 IP 地址、源端口号、目的地址和目的端口号。因此，一个 TCP 报文段从网络中到达一台主机上时，该主机使用全部 4 个 值来将报文段定向到相应的套接字。

UDP 协议

UDP 是一种无连接的，不可靠的传输层协议。它只提供了传输层需要实现的最低限度的功能 ,除了复用/分解功能和少量的差 错检测外 ,它几乎没有对 IP 增加其他的东西。UDP 协议适用于对实时性要求高的应用场景。

特点：

1.

使用 UDP 时，在发送报文段之前，通信双方没有握手的过程，因此 UDP 被称为是无连接的传输层协议。因为没有握手 过程，相对于 TCP 来说，没有建立连接的时延。因为没有连接，所以不需要在端系统中保存连接的状态。

2.

3.

UDP 提供尽力而为的交付服务，也就是说 UDP 协议不保证数据的可靠交付。

4.

5.

UDP 没有拥塞控制和流量控制的机制，所以 UDP 报文段的发送速率没有限制。

6.

7.

因为一个 UDP 套接字只使用目的地址和目的端口来标识，所以 UDP 可以支持一对一、一对多、多对一和多对多的交互 通信。

8.

9.

UDP 首部小，只有 8 个字节。

10.

UDP 报文段结构

UDP 报文段由首部和应用数据组成。报文段首部包含四个字段，分别是源端口号、目的端口号、长度和检验和，每个字段的长 度为两个字节。长度字段指的是整个报文段的长度，包含了首部和应用数据的大小。校验和是 UDP 提供的一种差错校验机制。 虽然提供了差错校验的机制，但是 UDP 对于差错的恢复无能为力。

TCP 协议

TCP 协议是面向连接的，提供可靠数据传输服务的传输层协议。

特点：

1.

TCP 协议是面向连接的，在通信双方进行通信前，需要通过三次握手建立连接。它需要在端系统中维护双方连接的状态信息。

2.

3.

TCP 协议通过序号、确认号、定时重传、检验和等机制，来提供可靠的数据传输服务。

4.

5.

TCP 协议提供的是点对点的服务，即它是在单个发送方和单个接收方之间的连接。

6.

7.

TCP 协议提供的是全双工的服务，也就是说连接的双方的能够向对方发送和接收数据。

8.

9.

TCP 提供了拥塞控制机制，在网络拥塞的时候会控制发送数据的速率，有助于减少数据包的丢失和减轻网络中的拥塞程度。

10.

11.

TCP 提供了流量控制机制，保证了通信双方的发送和接收速率相同。如果接收方可接收的缓存很小时，发送方会降低发送 速率，避免因为缓存填满而造成的数据包的丢失。

12.

TCP 报文段结构

TCP 报文段由首部和数据组成，它的首部一般为 20 个字节。

源端口和目的端口号用于报文段的多路复用和分解。

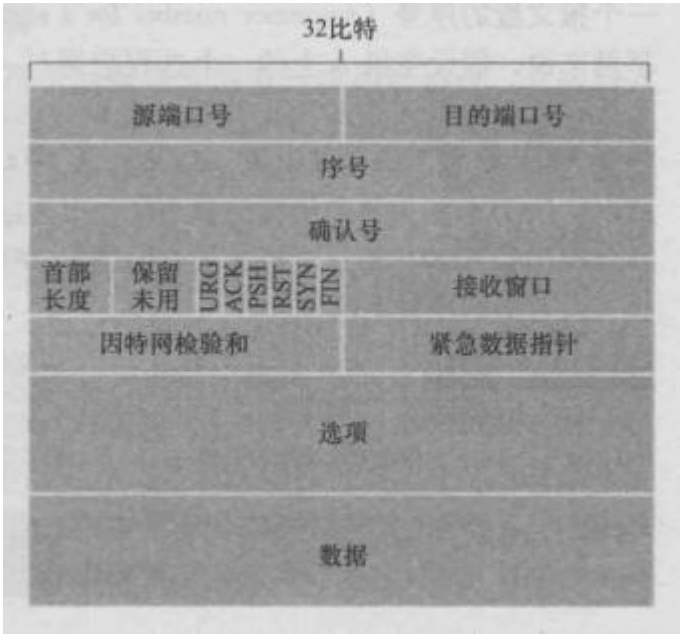
32 比特的序号和 32 比特的确认号，用与实现可靠数据运输服务。

16 比特的接收窗口字段用于实现流量控制，该字段表示接收方愿意接收的字节的数量。

4 比特的首部长度字段，该字段指示了以 32 比特的字为单位的 TCP 首部的长度。

6 比特的标志字段，ACK 字段用于指示确认序号的值是有效的，RST、SYN 和 FIN 比特用于连接建立和拆除。设置 PSH 字 段指示接收方应该立即将数据交给上层，URG 字段用来指示报文段里存在紧急的数据。

校验和提供了对数据的差错检测。



TCP 三次握手的过程

第一次握手，客户端向服务器发送一个 SYN 连接请求报文段，报文段的首部中 SYN 标志位置为 1，序号字段是一个任选的 随机数。它代表的是客户端数据的初始序号。

第二次握手，服务器端接收到客户端发送的 SYN 连接请求报文段后，服务器首先会为该连接分配 TCP 缓存和变量，然后向 客户端发送 SYN ACK 报文段，报文段的首部中 SYN 和 ACK 标志位都被置为 1，代表这是一个对 SYN 连接请求的确认， 同时序号字段是服务器端产生的一个任选的随机数，它代表的是服务器端数据的初始序号。确认号字段为客户端发送的序号加 一。

第三次握手，客户端接收到服务器的肯定应答后，它也会为这次 TCP 连接分配缓存和变量，同时向服务器端发送一个对服务 器端的报文段的确认。第三次握手可以在报文段中携带数据。

在我看来，TCP 三次握手的建立连接的过程就是相互确认初始序号的过程，告诉对方，什么样序号的报文段能够被正确接收。 第三次握手的作用是客户端对服务器端的初始序号的确认。如果只使用两次握手，那么服务器就没有办法知道自己的序号是否 已被确认。同时这样也是为了防止失效的请求报文段被服务器接收，而出现错误的情况。

详细资料可以参考：《TCP 为什么是三次握手，而不是两次或四次？》《TCP 的三次握手与四次挥手》

TCP 四次挥手的过程

因为 TCP 连接是全双工的，也就是说通信的双方都可以向对方发送和接收消息，所以断开连接需要双方的确认。

第一次挥手，客户端认为没有数据要再发送给服务器端，它就向服务器发送一个 FIN 报文段，申请断开客户端到服务器端的 连接。发送后客户端进入 FIN_WAIT_1 状态。

第二次挥手，服务器端接收到客户端释放连接的请求后，向客户端发送一个确认报文段，表示已经接收到了客户端释放连接的 请求，以后不再接收客户端发送过来的数据。但是因为连接是全双工的，所以此时，服务器端还可以向客户端发送数据。服务 器端进入 CLOSE_WAIT 状态。客户端收到确认后，进入 FIN_WAIT_2 状态。

第三次挥手，服务器端发送完所有数据后，向客户端发送 FIN 报文段，申请断开服务器端到客户端的连接。发送后进入 LAS T_ACK 状态。

第四次挥手，客户端接收到 FIN 请求后，向服务器端发送一个确认应答，并进入 TIME_WAIT 阶段。该阶段会持续一段时间， 这个时间为报文段在网络中的最大生存时间，如果该时间内服务端没有重发请求的话，客户端进入 CLOSED 的状态。如果收到 服

务器的重发请求就重新发送确认报文段。服务器端收到客户端的确认报文段后就进入 CLOSED 状态，这样全双工的连接就被 释放了。

TCP 使用四次挥手的原因是因为 TCP 的连接是全双工的，所以需要双方分别释放到对方的连接，单独一方的连接释放，只代 表不能再向对方发送数据，连接处于的是半释放的状态。

最后一次挥手中，客户端会等待一段时间再关闭的原因，是为了防止发送给服务器的确认报文段丢失或者出错，从而导致服务器 端不能正常关闭。

详细资料可以参考：

《前端面试之道》

状 态 转 化 图

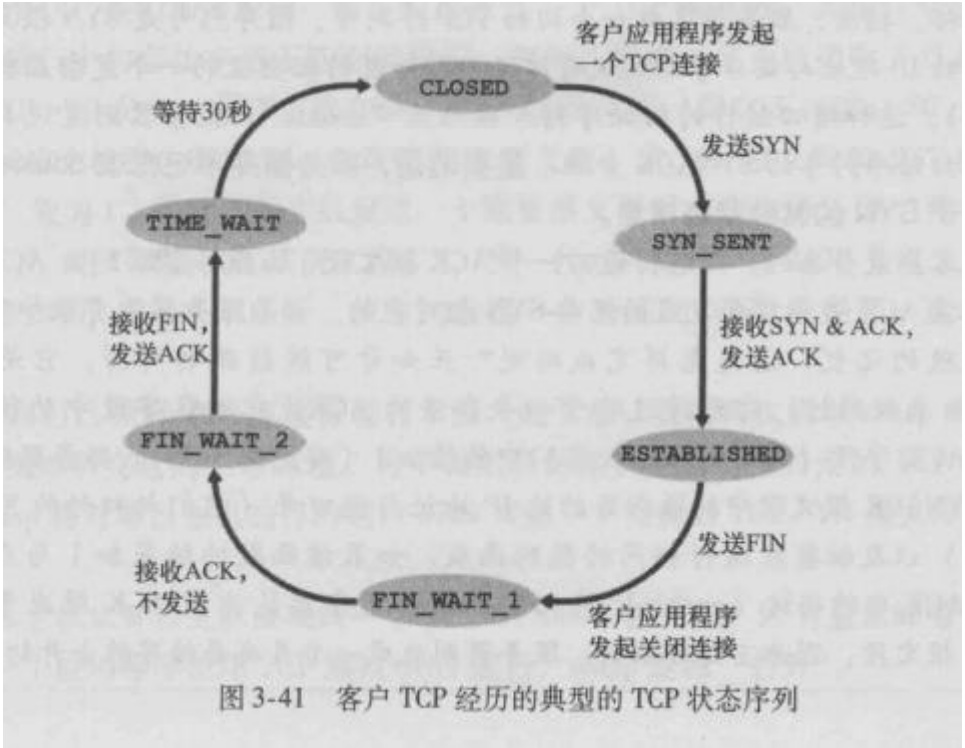


图 3-41 客户 TCP 经历的典型的 TCP 状态序列

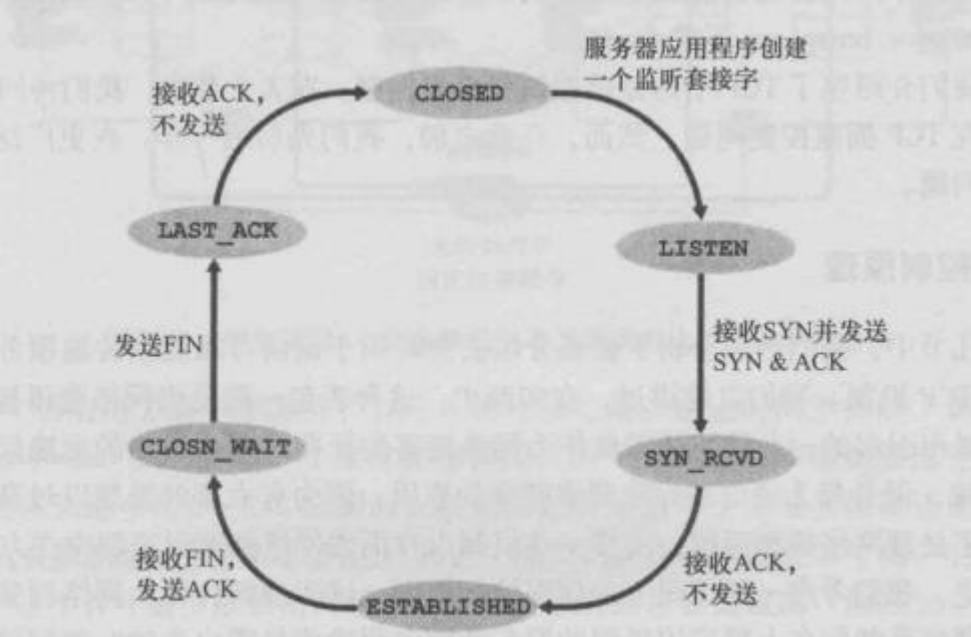


图 3-42　服务器端 TCP 经历的典型的 TCP 状态序列

ARQ 协议

ARQ 协议指的是自动重传请求 ,它通过超时和重传来保证数据的可靠交付 ,它是 TCP 协议实现可靠数据传输的一个很重要的 机制。

它分为停止等待 ARQ 协议和连续 ARQ 协议。

一、停止等待 ARQ 协议

停止等待 ARQ 协议的基本原理是，对于发送方来说发送方每发送一个分组，就为这个分组设置一个定时器。当发送分组的确认 回答返回了，则清除定时器，发送下一个分组。如果在规定的时间内没有收到已发送分组的肯定回答，则重新发送上一个分组。

对于接受方来说，每次接受到一个分组，就返回对这个分组的肯定应答，当收到冗余的分组时，就直接丢弃，并返回一个对冗余 分组的确认。当收到分组损坏的情况的时候，直接丢弃。

使用停止等待 ARQ 协议的缺点是每次发送分组必须等到分组确认后才能发送下一个分组，这样会造成信道的利用率过低。

二、连续 ARQ 协议

连续 ARQ 协议是为了解决停止等待 ARQ 协议对于信道的利用率过低的问题。它通过连续发送一组分组，然后再等待对分组的 确认回答，对于如何处理分组中可能出现的差错恢复情况，一般可以使用滑动窗口协议和选择重传协议来实现。

1. 滑动窗口协议

使用滑动窗口协议，在发送方维持了一个发送窗口，发送窗口以前的分组是已经发送并确认了的分组，发送窗口中包含了已经发 送但未确认的分组和允许发送但还未发送的分组，发送窗口以后的分组是缓存中还不允许发送的分组。当发送方向接收方发送分 组时，会依次发送窗口内的所有分组，并且设置一个定时器，这个定时器可以理解为是最早发送但未收到确认的分组。如果在定 时器的时间内收到某一个分组的确认回答，则滑动窗口，将窗口的首部移动到确认分组的后一个位置，此时如果还有已发送但没 有确认的分组，则重新设置定时器，如果没有了则关闭定时器。如果定时器超时，则重新发送所有已经发送但还未收到确认的分 组。

接收方使用的是累计确认的机制，对于所有按序到达的分组，接收方返回一个分组的肯定回答。如果收到了一个乱序的分组，那 么接方会直接丢弃，并返回一个最近的按序到达的分组的肯定回答。使用累计确认保证了确认号以前的分组都已经按序到达了， 所以发送窗口可以移动到已确认分组的后面。

滑动窗口协议的缺点是因为使用了累计确认的机制，如果出现了只是窗口中的第一个分组丢失，而后面的分组都按序到达的情况 的话，那么滑动窗口协议会重新发送所有的分组，这样就造成了大量不必要分组的丢弃和重传。

1. 选择重传协议

因为滑动窗口使用累计确认的方式，所以会造成很多不必要分组的重传。使用选择重传协议可以解决这个问题。

选择重传协议在发送方维护了一个发送窗口。发送窗口的以前是已经发送并确认的分组，窗口内包含了已发送但未被确认的分组， 已确认的乱序分组，和允许发送但还未发送的分组，发送窗口以后的是缓存中还不允许发送的分组。选择重传协议与滑动窗口协 议最大的不同是，发送方发送分组时，为一个分组都创建了一个定时器。当发送方接受到一个分组的确认应答后，取消该分组的 定时器，并判断接受该分组后，是否存在由窗口首部为首的连续的确认分组，如果有则向后移动窗口的位置，如果没有则将该分 组标识为已接收的乱序分组。当某一个分组定时器到时后，则重新传递这个分组。

在接收方，它会确认每一个正确接收的分组，不管这个分组是按序的还是乱序的，乱序的分组将被缓存下来，直到所有的乱序分 组都到达形成一个有序序列后，再将这一段分组交付给上层。对于不能被正确接收的分组，接收方直接忽略该分组。

详细资料可以参考：《TCP 连续 ARQ 协议和滑动窗口协议》

TCP 的可靠运输机制

TCP 的可靠运输机制是基于连续 ARQ 协议和滑动窗口协议的。

TCP 协议在发送方维持了一个发送窗口，发送窗口以前的报文段是已经发送并确认了的报文段，发送窗口中包含了已经发送但 未确认的报文段和允许发送但还未发送的报文段，发送窗口以后的报文段是缓存中还不允许发送的报文段。当发送方向接收方发 送报文时，会依次发送窗口内的所有报文段，并且设置一个定时器，这个定时器可以理解为是最早发送但未收到确认的报文段。 如果在定时器的时间内收到某一个报文段的确认回答，则滑动窗口，将窗口的首部向后滑动到确认报文段的后一个位置，此时如 果还有已发送但没有确认的报文段，则重新设置定时器，如果没有了则关闭定时器。如果定时器超时，则重新发送所有已经发送 但还未收到确认的报文段，并将超时的间隔设置为以前的两倍。当发送方收到接收方的三个冗余的确认应答后，这是一种指示， 说明该报文段以后的报文段很有可能发生丢失了，那么发送方会启用快速重传的机制，就是当前定时器结束前，发送所有的已发 送但确认的报文段。

接收方使用的是累计确认的机制，对于所有按序到达的报文段，接收方返回一个报文段的肯定回答。如果收到了一个乱序的报文 段，那么接方会直接丢弃，并返回一个最近的按序到达的报文段的肯定回答。使用累计确认保证了返回的确认号之前的报文段都 已经按序到达了，所以发送窗口可以移动到已确认报文段的后面。

发送窗口的大小是变化的，它是由接收窗口剩余大小和网络中拥塞程度来决定的，TCP 就是通过控制发送窗口的长度来控制报文 段的发送速率。

但是 TCP 协议并不完全和滑动窗口协议相同，因为许多的 TCP 实现会将失序的报文段给缓存起来，并且发生重传时，只会重 传一个报文段，因此 TCP 协议的可靠传输机制更像是窗口滑动协议和选择重传协议的一个混合体。

TCP 的流量控制机制

TCP 提供了流量控制的服务，这个服务的主要目的是控制发送方的发送速率，保证接收方来得及接收。因为一旦发送的速率大 于接收方所能接收的速率，就会造成报文段的丢失。接收方主要是通过接收窗口来告诉发送方自己所能接收的大小，发送方根据 接收方的接收窗口的大小来调整发送窗口的大小，以此来达到控制发送速率的目的。

TCP 的拥塞控制机制

TCP 的拥塞控制主要是根据网络中的拥塞情况来控制发送方数据的发送速率，如果网络处于拥塞的状态，发送方就减小发送的 速率，这样一方面是为了避免继续增加网络中的拥塞程度，另一方面也是为了避免网络拥塞可能造成的报文段丢失。

TCP 的拥塞控制主要使用了四个机制，分别是慢启动、拥塞避免、快速重传和快速恢复。

慢启动的基本思想是 ,因为在发送方刚开始发送数据的时候 ,并不知道网络中的拥塞程度，所以先以较低的速率发送，进行试探 ，每次收到一个确认报文，就将发动窗口的长度加一，这样每个 RTT 时间后，发送窗口的长度就会加倍。当发送窗口的大小达 到一个阈值的时候就进入拥塞避免算法。

拥塞避免算法是为了避免可能发生的拥塞，将发送窗口的大小由每过一个 RTT 增长一倍，变为每过一个 RTT ，长度只加一。 这样将窗口的增长速率由指数增长，变为加法线性增长。

快速重传指的是，当发送方收到三个冗余的确认应答时，因为 TCP 使用的是累计确认的机制，所以很有可能是发生了报文段的 丢失，因此采用立即重传的机制，在定时器结束前发送所有已发送但还未接收到确认应答的报文段。

快速恢复是对快速重传的后续处理，因为网络中可能已经出现了拥塞情况，所以会将慢启动的阀值减小为原来的一半，然后将拥 塞窗口的值置为减半后的阀值，然后开始执行拥塞避免算法，使得拥塞窗口缓慢地加性增大。简单来理解就是，乘性减，加性增。

TCP 认为网络拥塞的主要依据是报文段的重传次数，它会根据网络中的拥塞程度，通过调整慢启动的阀值，然后交替使用上面四 种机制来达到拥塞控制的目的

网络层

网络层协议主要实现了不同主机间的逻辑通信功能。网络层协议一共包含两个主要的组件，一个 IP 网际协议，一个是路由选 择协议。

IP 网际协议规定了网络层的编址和转发方式，比如说我们接入网络的主机都会被分配一个 IP 地址，常用的比如 IPV4 使用 32 位来分配地址，还有 IPv6 使用 128 位来分配地址。

路由选择协议决定了数据报从源到目的地所流经的路径，常见的比如距离向量路由选择算法等。

数据链路层

提供的服务是如何将数据报通过单一通信链路从一个结点移动到相邻节点。每一台主机都有一个唯一的 MAC 地址， 这是由网络适配器决定的，在全世界都是独一无二的。

物理层

物理层提供的服务是尽可能的屏蔽掉组成网络的物理设备和传输介质间的差异 ,使数据链路层不需要考虑网络的具体传输介质 是什么。

常考面试题

Post 和 Get 的区别？

Post 和 Get 是 HTTP 请求的两种方法。

（1）从应用场景上来说，GET 请求是一个幂等的请求，一般 Get 请求用于对服务器资源不会产生影响的场景，比如说请求一个网页。而 Post 不是一个幂等的请求，一般用于对服务器资源会产生影响的情景。比如注册用户这一类的操作。

（2）因为不同的应用场景，所以浏览器一般会对 Get 请求缓存，但很少对 Post 请求缓存。

（3）从发送的报文格式来说，Get 请求的报文中实体部分为空，Post 请求的报文中实体部分一般为向服务器发送的数据。

（4）但是 Get 请求也可以将请求的参数放入 url 中向服务器发送，这样的做法相对于 Post 请求来说，一个方面是不太安全，因为请求的 url 会被保留在历史记录中。并且浏览器由于对 url 有一个长度上的限制，所以会影响 get 请求发送数据时的长度。这个限制是浏览器规定的，并不是 RFC 规定的。还有就是 post 的参数传递支持更多的数据类型。

TLS/SSL 中什么一定要用三个随机数，来生成”会话密钥” ？

客户端和服务器都需要生成随机数，以此来保证每次生成的秘钥都不相同。

使用三个随机数，是因为 SSL 的协议默认不信任每个主机都能产生完全随机的数，如果只使用一个伪随机的数来生成秘钥，就很容易被破解。

通过使用三个随机数的方式，增加了自由度，一个伪随机可能被破解，但是三个伪随机就很接近于随机了，因此可以使用这种方法来保持生成秘钥的随机性和安全性。

SSL 连接断开后如何恢复？

一共有两种方法来恢复断开的 SSL 连接，一种是使用 session ID，一种是 session ticket。

使用 session ID 的方式，每一次的会话都有一个编号，当对话中断后，下一次重新连接时，只要客户端给出这个编号，服务器如果有这个编号的记录，那么双方就可以继续使用以前的秘钥，而不用重新生成一把。

目前所有的浏览器都支持这一种方法。

但是这种方法有一个缺点是，session ID 只能够存在一台服务器上，如果我们的请求通过负载平衡被转移到了其他的服务器上，那么就无法恢复对话。

另一种方式是 session ticket 的方式 ,session ticket 是服务器在上一次对话中发送给客户的，这个 ticket 是加密的，只有服务器能够解密，里面包含了本次会话的信息，比如对话秘钥和加密方法等。

这样不管我们的请求是否转移到其他的服务器上，当服务器将 ticket 解密以后，就能够获取上次对话的信息，就不用重新生成对话秘钥了

RSA 算法的安全性保障？

对极大整数做因数分解的难度决定了 RSA 算法的可靠性。

换言之，对一极大整数做因数分解愈困难，RSA 算法愈可靠。

现在 1024 位的 RSA 密钥基本安全，2048 位的密钥极其安全。

DNS 为什么使用 UDP 协议作为传输层协议？

DNS 使用 UDP 协议作为传输层协议的主要原因是为了避免使用 TCP 协议时造成的连接时延。

因为为了得到一个域名的 IP 地址，往往会向多个域名服务器查询，如果使用 TCP 协议，那么每次请求都会存在连接时延，这样使 DNS 服务变得很慢，因为大多数的地址查询请求，都是浏览器请求页面时发出的，这样会造成网页的等待时间过长。

使用 UDP 协议作为 DNS 协议会有一个问题，由于历史原因，物理链路的最小 MTU = 576，所以为了限制报文长度不超过 576，UDP 的报文段的长度被限制在 512 个字节以内，这样一旦 DNS 的查询或者应答报文，超过了 512 字节，那么基于 UDP 的 DNS 协议就会被截断为 512 字节，那么有可能用户得到的 DNS 应答就是不完整的。这里 DNS 报文的长度一旦超过限制，并不会像 TCP 协议那样被拆分成多个报文段传输，因为 UDP 协议不会维护连接状态，所以我们没有办法确定那几个报文段属于同一个数据，UDP 只会将多余的数据给截取掉。

为了解决这个问题，我们可以使用 TCP 协议去请求报文。

DNS 还存在的一个问题是安全问题，就是我们没有办法确定我们得到的应答，一定是一个安全的应答，因为应答可以被他人伪造，所以现在有了 DNS over HTTPS 来解决这个问题。

当你在浏览器中输入 Google.com 并且按下回车之后发生了什么？

（1）首先会对 URL 进行解析，分析所需要使用的传输协议和请求的资源的路径。如果输入的 URL 中的协议或者主机名不合法，将会把地址栏中输入的内容传递给搜索引擎。如果没有问题，浏览器会检查 URL 中是否出现了非法字符，如果存在非法字 符，则对非法字符进行转义后再进行下一过程。

（2）浏览器会判断所请求的资源是否在缓存里，如果请求的资源在缓存里并且没有失效，那么就直接使用，否则向服务器发起新 的请求。

（3）下一步我们首先需要获取的是输入的 URL 中的域名的 IP 地址，首先会判断本地是否有该域名的 IP 地址的缓存，如果 有则使用，如果没有则向本地 DNS 服务器发起请求。本地 DNS 服务器也会先检查是否存在缓存，如果没有就会先向根域 名服务器发起请求，获得负责的顶级域名服务器的地址后，再向顶级域名服务器请求，然后获得负责的权威域名服务器的地 址后，再向权威域名服务器发起请求，最终获得域名的 IP 地址后，本地 DNS 服务器再将这个 IP 地址返回给请求的用 户。用户向本地 DNS 服务器发起请求属于递归请求，本地 DNS 服务器向各级域名服务器发起请求属于迭代请求。

（4）当浏览器得到 IP 地址后，数据传输还需要知道目的主机 MAC 地址，因为应用层下发数据给传输层，TCP 协议会指定源 端口号和目的端口号，然后下发给网络层。网络层会将本机地址作为源地址，获取的 IP 地址作为目的地址。然后将下发给 数据链路层，

数据链路层的发送需要加入通信双方的 MAC 地址，我们本机的 MAC 地址作为源 MAC 地址，目的 MAC 地 址需要分情况处理，通过将 IP 地址与我们本机的子网掩码相与，我们可以判断我们是否与请求主机在同一个子网里，如果 在同一个子网里，我们可以使用 APR 协议获取到目的主机的 MAC 地址，如果我们不在一个子网里，那么我们的请求应该 转发给我们的网关，由它代为转发，此时同样可以通过 ARP 协议来获取网关的 MAC 地址，此时目的主机的 MAC 地址应 该为网关的地址。

（5）下面是 TCP 建立连接的三次握手的过程，首先客户端向服务器发送一个 SYN 连接请求报文段和一个随机序号，服务端接 收到请求后向服务器端发送一个 SYN ACK 报文段，确认连接请求，并且也向客户端发送一个随机序号。客户端接收服务器的 确认应答后，进入连接建立的状态，同时向服务器也发送一个 ACK 确认报文段，服务器端接收到确认后，也进入连接建立 状态，此时双方的连接就建立起来了。

（6）如果使用的是 HTTPS 协议，在通信前还存在 TLS 的一个四次握手的过程。首先由客户端向服务器端发送使用的协议的版 本号、一个随机数和可以使用的加密方法。服务器端收到后，确认加密的方法，也向客户端发送一个随机数和自己的数字证 书。客户端收到后，首先检查数字证书是否有效，如果有效，则再生成一个随机数，并使用证书中的公钥对随机数加密，然后 发送给服务器端，并且还会提供一个前面所有内容的 hash 值供服务器端检验。服务器端接收后，使用自己的私钥对数据解 密，同时向客户端发送一个前面所有内容的 hash 值供客户端检验。这个时候双方都有了三个随机数，按照之前所约定的加 密方法，使用这三个随机数生成一把秘钥，以后双方通信前，就使用这个秘钥对数据进行加密后再传输。

（7）当页面请求发送到服务器端后，服务器端会返回一个 html 文件作为响应，浏览器接收到响应后，开始对 html 文件进行 解析，开始页面的渲染过程。

（8）浏览器首先会根据 html 文件构建 DOM 树，根据解析到的 css 文件构建 CSSOM 树，如果遇到 script 标签，则判端 是否含有 defer 或者 async 属性，要不然 script 的加载和执行会造成页面的渲染的阻塞。当 DOM 树和 CSSOM 树建 立好后，根据它们来构建渲染树。渲染树构建好后，会根据渲染树来进行布局。布局完成后，最后使用浏览器的 UI 接口对页 面进行绘制。这个时候整个页面就显示出来了。

（9）最后一步是 TCP 断开连接的四次挥手过程。

谈谈 CDN 服务？

CDN 是一个内容分发网络，通过对源网站资源的缓存，利用本身多台位于不同地域、不同运营商的服务器，向用户提供资就近访问的 功能。也就是说，用户的请求并不是直接发送给源网站，而是发送给 CDN 服务器，由 CND 服务器将请求定位到最近的含有该资源 的服务器上去请求。这样有利于提高网站的访问速度，同时通过这种方式也减轻了源服务器的访问压力。

什么是正向代理和反向代理？

我们常说的代理也就是指正向代理，正向代理的过程，它隐藏了真实的请求客户端，服务端不知道真实的客户端是谁，客户端请求的 服务都被代理服务器代替来请求。

反向代理隐藏了真实的服务端，当我们请求一个网站的时候，背后可能有成千上万台服务器为我们服务，但具体是哪一台，我们不知 道，也不需要知道，我们只需要知道反向代

理服务器是谁就好了，反向代理服务器会帮我们把请求转发到真实的服务器那里去。反向代理器一般用来实现负载平衡。

负载平衡的两种实现方式？

一种是使用反向代理的方式，用户的请求都发送到反向代理服务上，然后由反向代理服务器来转发请求到真实的服务器上，以此来实 现集群的负载平衡。

另一种是 DNS 的方式，DNS 可以用于在冗余的服务器上实现负载平衡。因为现在一般的大型网站使用多台服务器提供服务，因此一 个域名可能会对应多个服务器地址。当用户向网站域名请求的时候，DNS 服务器返回这个域名所对应的服务器 IP 地址的集合，但在 每个回答中，会循环这些 IP 地址的顺序，用户一般会选择排在前面的地址发送请求。以此将用户的请求均衡的分配到各个不同的服 务器上，这样来实现负载均衡。这种方式有一个缺点就是，由于 DNS 服务器中存在缓存，所以有可能一个服务器出现故障后，域名解 析仍然返回的是那个 IP 地址，就会造成访问的问题。

http 请求方法 options 方法有什么用？

OPTIONS 请求与 HEAD 类似，一般也是用于客户端查看服务器的性能。这个方法会请求服务器返回该资源所支持的所有 HTTP 请 求方法，该方法会用’ * ‘来代替资源名称，向服务器发送 OPTIONS 请求，可以测试服务器功能是否正常。JS 的 XMLHttpRequest 对象进行 CORS 跨域资源共享时，对于复杂请求，就是使用 OPTIONS 方法发送嗅探请求，以判断是否有对指定资源的访问权限。

http1.1 和 http1.0 之间有哪些区别？

http1.1 相对于 http1.0 有这样几个区别：

（1）连接方面的区别，http1.1 默认使用持久连接，而 http1.0 默认使用非持久连接。http1.1 通过使用持久连接来使多个 http 请求复用同一个 TCP 连接，以此来避免使用非持久连接时每次需要建立连接的时延。

（2）资源请求方面的区别，在 http1.0 中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，http1.1 则在请求头引入了 range 头域，它允许只请求资源的某个部分，即返回码是 206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

（3）缓存方面的区别，在 http1.0 中主要使用 header 里的 If-Modified-Since,Expires 来做为缓存判断的标准，http1.1 则引入了更多的缓存控制策略例如 Etag、If-Unmodified-Since、If-Match、If-None-Match 等更多可供选择的缓存头来控制缓存策略。

（4）http1.1 中还新增了 host 字段，用来指定服务器的域名。http1.0 中认为每台服务器都绑定一个唯一的 IP 地址，因此，请求消息中的 URL 并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机，并且它们共享一个 IP 地址。因此有了 host 字段，就可以将请求发往同一台服务器上的不同网站。

（5）http1.1 相对于 http1.0 还新增了很多方法，如 PUT、HEAD、OPTIONS 等

即时通讯的实现，短轮询、长轮询、SSE 和 WebSocket 间的区别？

短轮询和长轮询的目的都是用于实现客户端和服务器端的一个即时通讯。

短轮询的基本思路就是浏览器每隔一段时间向浏览器发送 http 请求，服务器端在收到请求后，不论是否有数据更新，都直接进行 响应。这种方式实现的即时通信，本质上还是浏览器发送请求，服务器接受请求的一个过程，通过让客户端不断的进行请求，使得客 户端能够模拟实时地收到服务器端的数据的变化。这种方式的优点是比较简单，易于理解。缺点是这种方式由于需要不断的建立 ht tp 连接，严重浪费了服务器端和客户端的资源。当用户增加时，服务器端的压力就会变大，这是很不合理的。

长轮询的基本思路是，首先由客户端向服务器发起请求，当服务器收到客户端发来的请求后，服务器端不会直接进行响应，而是先将 这个请求挂起，然后判断服务器端数据是否有更新。如果有更新，则进行响应，如果一直没有数据，则到达一定的时间限制才返回。客户端 JavaScript 响应处理函数会在处理完服务器返回的信息后，再次发出请求，重新建立连接。长轮询和短轮询比起来，它的 优点是明显减少了很多不必要的 http 请求次数，相比之下节约了资源。长轮询的缺点在于，连接挂起也会导致资源的浪费。

SSE 的基本思想是，服务器使用流信息向服务器推送信息。严格地说，http 协议无法做到服务器主动推送信息。但是，有一种变通 方法，就是服务器向客户端声明，接下来要发送的是流信息。也就是说，发送的不是一次性的数据包，而是一个数据流，会连续不断地发送过来。这时，客户端不会关闭连接，会一直等着服务器发过来的新的数据流，视频播放就是这样的例子。SSE 就是利用这种机 制，使用流信息向浏览器推送信息。它基于 http 协议，目前除了 IE/Edge，其他浏览器都支持。它相对于前面两种方式来说，不 需要建立过多的 http 请求，相比之下节约了资源。

上面三种方式本质上都是基于 http 协议的，我们还可以使用 WebSocket 协议来实现。WebSocket 是 Html5 定义的一个新协 议，与传统的 http 协议不同，该协议允许由服务器主动的向客户端推送信息。使用 WebSocket 协议的缺点是在服务器端的配置 比较复杂。WebSocket 是一个全双工的协议，也就是通信双方是平等的，可以相互发送消息，而 SSE 的方式是单向通信的，只能 由服务器端向客户端推送信息，如果客户端需要发送信息就是属于下一个 http 请求了。

怎么实现多个网站之间共享登录状态

在多个网站之间共享登录状态指的就是单点登录。多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。

我认为单点登录可以这样来实现，首先将用户信息的验证中心独立出来，作为一个单独的认证中心，该认证中心的作用是判断客户端发 送的账号密码的正确性，然后向客户端返回对应的用户信息，并且返回一个由服务器端秘钥加密的登录信息的 token 给客户端，该 token 具有一定的有效时限。当一个应用系统跳转到另一个应用系统时，通过 url 参数的方式来传递 token ,然后转移到的应用站 点发送给认证中心 ,认证中心对 token 进行解密后验证，如果用户信息没有失效，则向客户端返回对应的用户信息，如果失效了则将 页面重定向会单点登录页面。

六星教育 WEB 前端学院面试圣经