(6) 计算机基础

一 计算机网络

1 TCP建立连接的3次握手过程

第一次握手：起初两端都处于CLOSED关闭状态，Client将标志位SYN置为1，随机产生一个值seq=x，并将该数据包发送给Server，Client进入SYN-SENT状态，等待Server确认；

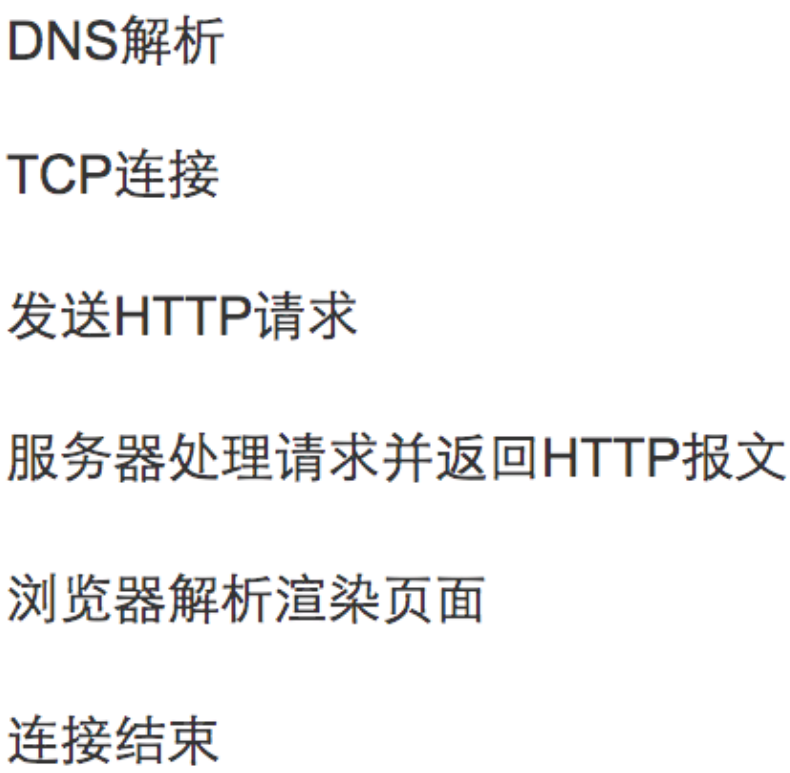
第二次握手：Server收到数据包后由标志位SYN=1得知Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=x+1，随机产生一个值seq=y，并将该数据包发送给Client以确认连接请求，Server进入SYN-RCVD状态，此时操作系统为该TCP连接分配TCP缓存和变量；

第三次握手：Client收到确认后，检查ack是否为x+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=y+1，并且此时操作系统为该TCP连接分配TCP缓存和变量，并将该数据包发送给Server，Server检查ack是否为y+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client和Server就可以开始传输数据。

2 CDN（Content Delivery Network）原理【2大核心：缓存 + 回源！！】

CDN的全称是Content Delivery Network，即内容分发网络。CDN的基本原理是广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响

3 用户输入URL到显示页面这个过程发生了什么？？



4 OSI七层模型？？【应用层、表示层、会话层、传输层、网络层、数据链路层、物理层】

从上到下分别是：

应用层：文件传输，常用协议HTTP，snmp,FTP，

表示层：数据格式化，代码转换，数据加密，

会话层：建立，解除会话

传输层：提供端对端的接口，tcp,udp

网络层：为数据包选择路由，IP，icmp

数据链路层：传输有地址的帧

物理层：二进制的数据形式在物理媒体上传输数据

物理层在最底下！！！

5 TCP和UDP的区别？？ 为什么要 3次握手 4次挥手？？

TCP和UDP之间的区别 OSI 和TCP/IP 模型在传输层定义两种传输协议：TCP（或传输控制协议）和UDP（或用户数据报协议）。 UDP 与TCP 的主要区别在于UDP 不一定提供可靠的数据传输。 事实上，该协议不能保证数据准确无误地到达目的地。

为什么TCP要进行四次挥手呢?

因为是双方彼此都建立了连接，因此双方都要释放自己的连接，A向B发出一个释放连接请求，他要释放链接表明不再向B发送数据了，此时B收到了A发送的释放链接请求之后，给A发送一个确认，A不能再向B发送数据了，它处于FIN-WAIT-2的状态，但是此时B还可以向A进行数据的传送。此时B向A 发送一个断开连接的请求，A收到之后给B发送一个确认。此时B关闭连接。A也关闭连接。

为什么要有TIME-WAIT这个状态呢，这是因为有可能最后一次确认丢失，如果B此时继续向A发送一个我要断开连接的请求等待A发送确认，但此时A已经关闭连接了，那么B永远也关不掉了，所以我们要有TIME-WAIT这个状态。

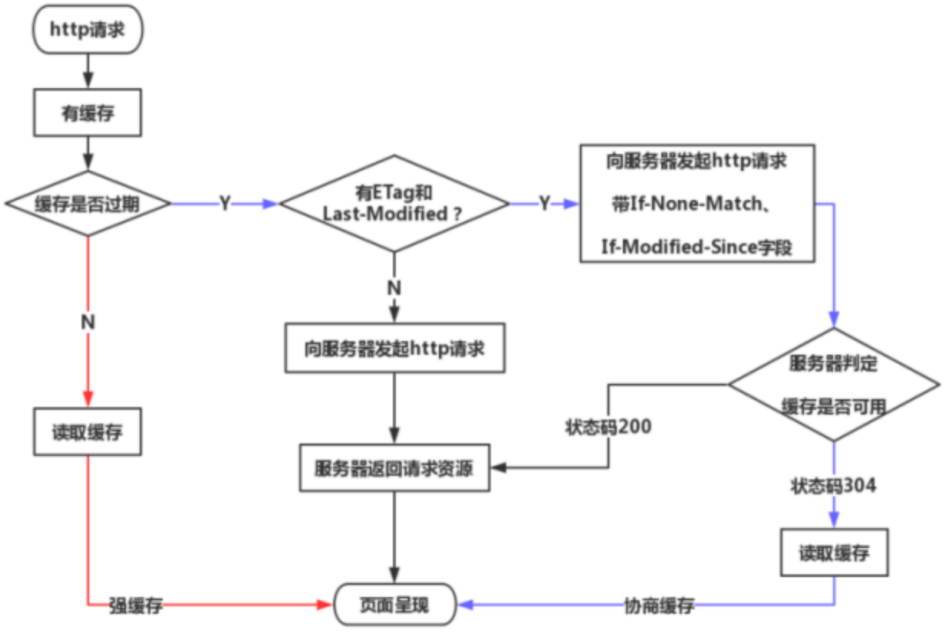
当然TCP也并不是100%可靠的。

6 HTTP缓存机制

HTTP缓存即是浏览器第一次想一个服务器发起HTTP请求后，服务器会返回请求的资源，并且在响应头中添加一些有关缓存的字段如：cache-control，expires,last-modifed，ETag,Date，等，之后浏览器再向该服务器请求资源就可以视情况使用强缓存和协商缓存，

强缓存：浏览器直接从本地缓存中获取数据，不与服务器进行交互，

协商缓存：浏览器发送请求到服务器，服务器判断是否可使用本地缓存，



7 websocket 和 ajax的区别是什么？？ websocket的应用场景有哪些？？

WebSocket的诞生本质上就是为了解决HTTP协议本身的单向性问题：请求必须由客户端向服务端发起，然后服务端进行响应。这个Request-Response的关系是无法改变的。对于一般的网页浏览和访问当然没问题，一旦我们需要服务端主动向客户端发送消息时就麻烦了，因为此前的TCP连接已经释放，根本找不到客户端在哪。

为了能及时从服务器获取数据，程序员们煞费苦心研究出来的各种解决方案其实都是在HTTP框架下做的妥协，没法子，浏览器这东西只支持HTTP，我们有什么办法。所以大家要么定时去轮询，要么就靠长连接——客户端发起请求，服务端把这个连接攥在手里不回复，等有消息了再回，如果超时了客户端就再请求一次——其实大家也懂，这只是个减少了请求次数、实时性更好的轮询，本质没变。

WebSocket就是从技术根本上解决这个问题的：看名字就知道，它借用了Web的端口和消息头来创建连接，后续的数据传输又和基于TCP的Socket几乎完全一样，但封装了好多原本在Socket开发时需要我们手动去做的功能。比如原生支持wss安全访问（跟https共用端口和证书）、创建连接时的校验、从数据帧中自动拆分消息包等等。

换句话说，原本我们在浏览器里只能使用HTTP协议，现在有了Socket，还是个更好用的Socket。

了解了WebSocket的背景和特性之后，就可以回答它能不能取代AJAX这个问题了：

对于服务器与客户端的双向通信，WebSocket简直是不二之选。如果不是还有少数旧版浏览器尚在服役的话，所有的轮询、长连接等方式早就该废弃掉。那些整合多种双向推送消息方式的库（如http://Socket.IO、SignalR）当初最大的卖点就是兼容所有浏览器版本，自动识别旧版浏览器并采取不同的连接方式，现在也渐渐失去了优势——所有新版浏览器都兼容WebSocket，直接用原生的就行了。

说句题外话，这点很像jQuery，在原生js难用时迅速崛起，当其他库和原生js都吸收了它的很多优势时，慢慢就不那么重要了。

但是，很大一部分AJAX的使用场景仍然是传统的请求-响应形式，比如获取json数据、post表单之类。这些功能虽然靠WebSocket也能实现，但就像在原本传输数据流的TCP之上定义了基于请求的HTTP协议一样，我们也要在WebSocket之上重新定义一种新的协议，最少也要加个request id用来区分每次响应数据对应的请求吧。

……但是，何苦一层叠一层地造个新轮子呢？直接使用AJAX不是更简单、更成熟吗？

另外还有一种情况，也就是传输大文件、图片、媒体流的时候，最好还是老老实实用HTTP来传。如果一定要用WebSocket的话，至少也专门为这些数据专门开辟个新通道，而别去占用那条用于推送消息、对实时性要求很强的连接。否则会把串行的WebSocket彻底堵死的。

所以说，WebSocket在用于双向传输、推送消息方面能够做到灵活、简便、高效，但在普通的Request-Response过程中并没有太大用武之地，比起普通的HTTP请求来反倒麻烦了许多，甚至更为低效。

每项技术都有自身的优缺点，在适合它的地方能发挥出最大长处，而看到它的几个优点就不分场合地全方位推广的话，可能会适得其反。

我们自己在开发能与手机通信的互联网机器人时就使用了WebSocket，效果很好。但并不是用它取代HTTP，而是取代了原先用于通信的基于TCP的Socket。

优点是：

原先在Socket连接后还要进行一些复杂的身份验证，同时要阻止未验证的连接发送控制指令。现在不需要了，在建立WebSocket连接的url里就能携带身份验证参数，验证不通过可以直接拒绝，不用设置状态；

原先自己实现了一套类似SSL的非对称加密机制，现在完全不需要了，直接通过wss加密，还能顺便保证证书的可信性；

原先要自己定义Socket数据格式，设置长度与标志，处理粘包、分包等问题，现在WebSocket收到的直接就是完整的数据包，完全不用自己处理；

前端的nginx可以直接进行转发与负载均衡，部署简单多了

8 常见的跨域解决方案？？ 【至少7种】

8.1 常见的跨域解决方案：

常见的跨域方式大概有七种，大致可分为iframe、api跨域

1、JSONP，全称为json with padding，解决老版本浏览器跨域数据访问问题，原理是web页面调用JS文件不受浏览器同源策略限制，所以通过script标签可以进行跨域请求，流程如下：

首先前端设置好回调参数，并将其作为URL的参数

服务器端收到请求后，通过该参数获取到回调函数名，并将数据放在参数中返回

收到结果后因为是script标签，所以浏览器当做脚本运行，

2、cors，全称是跨域资源共享，允许浏览器向跨源服务器发出XMLHTTP Request请求，从而克服了ajax只能同源使用的策略，实现cors的关键是服务器，只要服务器实现了cros接口，就可以跨域通信

前端逻辑很简单，正常发起ajax请求即可，成功的关键在于服务器 Access-Control-Allow-Origin 是否包含请求页面的域名，如果不包含的话，浏览器将认为这是一次失败的异步请求，将会调用 xhr.onerror 中的函数。

Cros使用简单，支持POST方式，但是存在兼容问题

浏览器将cors请求分为两类，简单请求和非简单请求，对于简单请求，浏览器直接发出cors请求，就是在头信息之中增加一个origin字段，用于说明本次请求来自哪个协议+域名+端口，服务器根据这个值，决定是否同意本次请求，如果服务器同意本次请求，返回的响应中会多出几个头信息字段：

Access-Control-Allow-Orign：返回origin的字段或者*

Access-Control-Allow-Credentials,该字段可选，是一个bool值，表示是否允许发送cookie，

Access-Control-Expose-Headers

参考：http://www.ruanyifeng.com/blog/2016/04/cors.html

3、服务器代理：

即当你有跨域的请求操作时发给后端，让后端帮你代为请求，

8.2 四种不常用的跨域解决方案：

此外还有四中不常用的方式，也可了解下：

location.hash：

Window.name

postMessage

参考：https://juejin.im/entry/59feae9df265da43094488f6

二 操作系统

1 进程 和 线程 的区别？？

1.1 线程 是 进程的一部分，一个没有线程的进程 可以视为【“退化为”】 但线程。
线程有时又被称为 轻权进程、轻量级进程，是 CPU 调调的基本单位！！

1.2 进程，是并发执行的程序 在执行过程中分配和管理资源 的基本单位，是一个动态的概念，竞争计算机系统资源的基本单位！！

进程，是并发执行的程序在执行过程中分配和管理资源的基本单位，是一个动态概念，竞争计算机系统资源的基本单位。

线程，是进程的一部分，一个没有线程的进程可以被看作是单线程的。线程有时又被称为轻权进程或轻量级进程，也是 CPU 调度的一个基本单位。

2 线程的哪些资源共享 、不共享？？

2.1 线程中 共享的资源【堆是放 “复杂数据类型的”、”较大，所以需要进行共享？？！“】

a. 堆 由于堆是在进程空间中开辟出来的，所以它是理所当然地被共享的；因此new出来的都是共享的（16位平台上分全局堆和局部堆，局部堆是独享的）

b. 全局变量 它是与具体某一函数无关的，所以也与特定线程无关；因此也是共享的

c. 静态变量虽然对于局部变量来说，它在代码中是“放”在某一函数中的，但是其存放位置和全局变量一样，存于堆中开辟的.bss和.data段，是共享的

d. 文件等公用资源 这个是共享的，使用这些公共资源的线程必须同步。Win32 提供了几种同步资源的方式，包括信号、临界区、事件和互斥体。

2.2 线程中 不共享的资源【栈 放“基本数据类型、较小”，所以不该被共享？？！】

a. 栈 栈是独享的

b. 寄存器 这个可能会误解，因为电脑的寄存器是物理的，每个线程去取值难道不一样吗？其实线程里存放的是副本，包括程序计数器PC

3 进程间通信的 8种方式？？

3.1

1、无名管道：半双工的通信方式，数据只能单向流动且只能在具有亲缘关系的进程间使用

2、高级管道：将另一个程序当作一个新的进程在当前程序进程中启动，则这个进程算是当前程序的子进程，

3、有名管道，：也是半双工的通信方式，但是允许没有亲缘进程之间的通信

4、消息队列：消息队列是有消息的链表，存放在内核中，并由消息队列标识符标识，消息队列克服了信号传递信息少，管道只能承载无格式字节流以及缓冲区大小受限的缺点

5、信号量：信号量是一个计数器，可以用来控制多个进程对共享资源的访问，它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源，

6、信号：用于通知接受进程某个事件已经发生

7、共享内存：共享内存就是映射一段能被其他进程所访问的内存。这段共享内存由一个进程创建，但是多个进程可以访问，共享内存是最快的IPC 方式，往往与其他通信机制配合使用

8、套接字：可用于不同机器之间的进程通信

三 数据库

1 Redis 和 mysql 的区别？？？

1.1 类型上

mysql是关系型数据库； redis是缓存数据库！！

1.2 作用上

mysql用于持久化的存储数据到硬盘，功能强大、但是速度慢【毕竟放到了 硬盘上！！】

redis用于存储使用 较为频繁的数据 到缓存中、速度较快【毕竟在缓存中！！】

1.3 需求上

一般来说，2者需求点不同、基本都是结合者使用！！

（1）类型上

从类型上来说，mysql是关系型数据库，redis是缓存数据库

（2）作用上

mysql用于持久化的存储数据到硬盘，功能强大，但是速度较慢

redis用于存储使用较为频繁的数据到缓存中，读取速度快

（3）需求上

mysql和redis因为需求的不同，一般都是配合使用。

完