### TCP建立连接的三次握手过程

第一次握手（SYN，seq，SYN-SENT）：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（SYN,ACK,判断ack,seq，SYN-RCVD）：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(判断ack，ack)：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。

### cdn原理

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应

### tcp三次握手过程

第一次握手（3）：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（5）：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（3）：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。

### 说一下用户从输入url到显示页面这个过程发生了什么（6个步骤）

DNS解析（域名系统解析）

TCP连接

发送HTTP请求

服务器处理请求并返回HTTP报文

浏览器解析渲染页面

连接结束

### HTTP的头部包含哪些内容。常见的请求方法（我为什么要说后面的options，head，connect）

常见的请求方法有get,post，get用来请求数据，post用来提交数据，form表单使用get时数据会以querystring形式存在url中，因而不够安全也存在数据大小限制，而post不会，post将数据存放在http报文体中，获取数据应该用get，提交数据用post

### 请求方法head特性

Head只请求页面的首部，head方法和get方法相同，只不过服务器响应时不会返回消息体。一个head请求的响应中，http头中包含的元信息应该和一个get请求的响应消息**相同**，这种方法可以用来获取请求中隐含的元信息，而不用传输实体本身，这个也经常用来测试超链接的有效性和可用性，

Head请求有以下特点：

只请求资源的**首部**，

检查**超链接的有效性**

检查网页**是否被修改**

用于自动搜索机器人获取网页的标志信息，获取rss种子信息，或者传递安全认证信息等

### HTTP状态码，301和302有什么具体区别，200和304 的区别，

状态码可以按照第一个数字分类，1表示**信息**，2表示**成功**，3表示**重定向**，4表示**客户端错误**，5表示**服务器错误**

常见的状态码有101切换协议，200成功，301永久重定向，302临时重定向，304未修改

301和302的区别：301：永久移动，请求的网页已永久移动到新的位置，服务器返回此响应，会自动将请求者转到新位置，302：历史移动，服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来继续以后的请求

200和304：

200表示成功，服务器已成功处理了请求，通常表示为服务器提供了请求的网页。

304表示未修改，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

### OSI七层模型

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

### TCP和UDP的区别，为什么三次握手四次挥手

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。

1、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。

2、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。

3、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。

4、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。

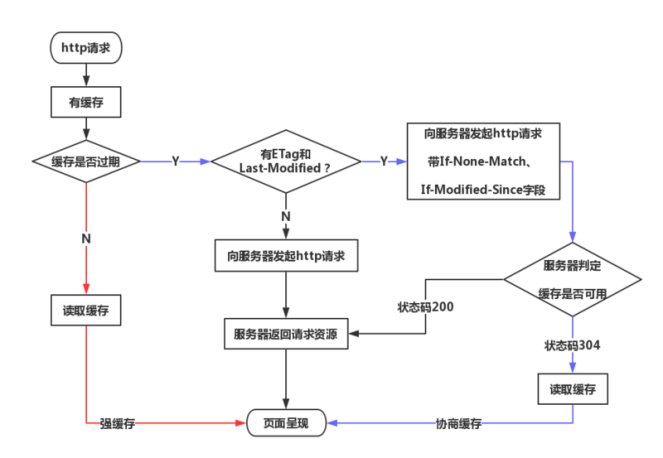
当然TCP也并不是100%可靠的。

### 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200：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304：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，



### 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](https://link.zhihu.com/?target=http:/Socket.IO" \t "https://www.nowcoder.com/tutorial/96/_blank)、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

### TCP/IP的网络模型

TCP/IP模型是一系列网络协议的总称，这些协议的目的是使得计算机之间可以进行信息交换，

TCP/IP模型四层架构从下到上分别是链路层，网络层，传输层，应用层

链路层的作用是负责**建立电路连接**，是整个**网络的物理基础**，典型的协议包括**以太网，ADSL**等，

网络层负责**分配地址和传送二进制数据**，主要协议是IP协议，

传输层负责**传送文本数据**，主要协议是TCP

应用层负责传送各种最终形态的数据，是直接**与用户信息打交道的层**，**主要协议是http，ftp**等

### 知道什么跨域方式吗，jsonp具体流程是什么，如何实现原生Jsonp封装，优化，对于CORS，服务器怎么判断它该不该跨域呢

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XMLHTTPRequest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（访问源控制） 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

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，**

此外还有四中不常用的方式，也可了解下：

Document.domain

location.hash

Window.name

postMessage

### 怎么生成token，怎么传递？

接口特点汇总：

1、因为是非开放性的，所以**所有的接口都是封闭的，只对公司内部的产品有效**；

2、因为是非开放性的，所以OAuth那套协议是行不通的，因为**没有中间用户的授权过程**；

3、有些接口**需要用户登录才能访问**；

4、有些接口**不需要用户登录就可访问**；

针对以上特点，移动端与服务端的通信就需要2把钥匙，即2个token。

第一个token是针对接口的（api\_token）；

第二个token是针对用户的（user\_token）；

先说第一个token（api\_token）

它的职责是保持接口访问的**隐蔽性和有效性**，保证接口只能给自家人用，怎么做到？参考思路如下：

现在的接口基本是mvc模式，URL基本是restful风格，URL大体格式如下：

[http://blog.snsgou.com/](http://blog.snsgou.com/" \t "https://www.nowcoder.com/tutorial/96/_blank)模块名/控制器名/方法名?参数名1=参数值1&参数名2=参数值2&参数名3=参数值3

接口token生成规则参考如下：

api\_token = md5 ('模块名' + '控制器名' + '方法名' + '2017-07-18' + '加密密钥') = 770fed4ca2aabd20ae9a5dd774711de2

其中的

1、 '2013-12-18' 为当天时间，

2、'加密密钥' 为私有的加密密钥，手机端需要在服务端注册一个“接口使用者”账号后，系统会分配一个账号及密码，数据表设计参考如下：



服务端接口校验，PHP实现流程如下：

<?php

1. 获取GET参数值

$module = $\_GET['mod'];

$controller = $\_GET['ctl'];

$action = $\_GET['act'];

$client\_id = $\_GET['client\_id'];

$api\_token = $\_GET['api\_token'];

2. 根据客户端传过来的client\_id，查询数据库，获取对应的client\_secret。

$client\_secret = getclientSecretById($client\_id);

3. 服务器重新生成一份api\_token

$api\_token\_server = md5($module.$controller.$action.date('Y-m-d', time()).$client\_secret);

4. 客户端传过来的api\_token与服务器生成的api\_token进行校对，如果不相等，则表示验证失败。

if($api\_token != $api\_token\_server){

exit('access deny');

}

5. 验证通过，返回数据到客户端。

再说第二个token（user\_token），它的职责是**保护用户的用户名及密码多次提交，以防密码泄露。**

如果接口需要用户登录，其访问流程如下：

1、用户提交“用户名”和“密码”，实现登录（条件允许，这一步最好走https）；

2、登录成功后，服务端返回一个 user\_token，生成规则参考如下：

服务端用数据表维护user\_token的状态，表设计如下：



服务端生成user\_token 后，返回给客户端（自己存储），客户端每次接口请求时，如果接口需要用户登录才能访问，则需要把 user\_id 与 user\_token 传回给服务端，服务端接受到这2个参数后，需要做以下几步：

1、检测 api\_token的有效性；

2、删除过期的 user\_token 表记录；

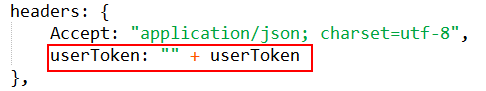
3、根据 user\_id，user\_token 获取表记录，如果表记录不存在，直接返回错误，如果记录存在，则进行下一步；

4、更新 user\_token 的过期时间（延期，保证其有效期内连续操作不掉线）；

5、返回接口数据。

那么token如何传递呢，ajax中传递token有以下几种方式：

1、放在**请求头**中：



2、使用**beforeSend**方法设置请求头

