# 前端面试知识点汇总

## HTML5

1. Doctype作用？严格模式与混杂模式如何区分？它们有何意义?

答：

（1）、<!DOCTYPE>声明位于HTML文档中的第一行，处于 <html> 标签之前。告知浏览器的解析器用什么文档标准解析这个文档。DOCTYPE不存在或格式不正确会导致文档以兼容模式呈现。

（2）、标准模式的排版 和JS运作模式都是以该浏览器支持的最高标准运行。在兼容模式中，页面以宽松的向后兼容的方式显示,模拟老式浏览器的行为以防止站点无法工作。

1. H5的新特性有哪些？移除了哪些？如何与 HTML 区分？

答：新增的主要是关于图像、位置、存储、多任务等功能。

Canvas、audio、video、localStorage、sessionStorage

语义化更好的标签：article/footer/header/nav/session

表单控件：calendar/date/time/email/url/search

新的技术：webworker/websocket/geolocation

移除的元素主要有：

纯表现的元素：basefont/big/center/font/s

对可用性产生负面影响的元素：frame/frameset/noframes

IE8/IE7/IE6支持通过document.createElement方法产生的标签，可以利用这一特性让这些浏览器支持HTML5新标签。

如何区分HTML5： DOCTYPE声明\新增的结构元素\功能元素。

1. 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？

答：CSS规范规定，每个元素都有display属性，且都有默认值。

（1）行内元素有：a label span img input select strong（强调的语气）

（2）块级元素有：div ul ol li dl dt dd h1 h2 h3 h4…p

（3）常见的空元素：

<br> <hr> <img> <input> <link> <meta>

鲜为人知的是：

<area> <base> <col> <command> <embed> <keygen> <param> <source> <track> <wbr>

不同浏览器（版本）、HTML4（5）、CSS2等实际略有差异

1. iframe有那些缺点？

答：

阻塞主页面的onload事件；

搜索引擎的检索程序无法解读这种页面，不利于SEO；

和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载；

如果需要使用iframe，最好是通过javascript动态给iframe添加src属性值，这样可以绕开以上两个问题。

1. 如何实现浏览器内多个标签页之间的通信? (阿里)

答：

WebSocket、ShardWorker

Localstorage、cookies等本地存储方式

Localstorage的原理: 它在另一个浏览上下文中被添加、修改或删除时，它都会触发一个事件。通过监听事件，控制它的值来进行页面通信。

## CSS3

1. 介绍一下标准的CSS的盒子模型？低版本IE的盒子模型有什么不同的？

答：IE 和 W3C 标准盒模型。主要有content/padding/margin/border

区别：IE的content部分把border和padding计算了进去。

1. Div 垂直居中的方法

答：绝对定位；绝对定位+transform；flex；

1. position的值relative和absolute定位原点是？

答：absolute：第一个不为static定位的父元素；relative：相对正常位置定位；fixed：相对于浏览器窗口定位；static：没有定位，正常文档流，会忽略 top/left/right/bottom/z-index 属性。

1. 经常遇到的浏览器的兼容性有哪些？原因，解决方法是什么，常用hack的技巧 ？

答：浏览器默认的margin和padding不同。解决方案是加一个全局的\*{margin:0;padding:0;}来统一。IE6双边距bug，使用“\9”、“+”、“\_”区分IE678浏览器；IE下获取获取自定义属性的办法，getAttribute；

事件：e = event || window.event;

坐标：IE下，event对象有x,y，而没有pageX、pageY，火狐下则相反，mX = event.x ? event.x : event.pageX;

触发事件的DOM：在IE中是window.event.srcElement，在Firefox中是event.target，而在Opera中则两者都支持。

IE和火狐下的兼容性：

使用obj.parentNode/document.formName.item(“itemname”)，使用[]获取集合类对象；

1. 对BFC规范(块级格式化上下文：block formatting context)的理解？

答：W3C CSS2.1 规范的一个概念，它是一个独立容器，决定了元素如何对其内容进行定位，以及与其他元素的关系和相互作用。

一个页面是由很多个 Box 组成的，元素的类型和 display 属性，决定了 Box的类型。

不同类型的 Box，会参与不同的 Formatting Context ，因此Box内的元素会以不同的方式渲染，也就是说 **BFC 内部的元素和外部的元素不会相互影响。**

BFC是block formatting context，也就是块级格式化上下文，是用于布局块级盒子的一块渲染区域

满足下列条件之一就可触发BFC

　　【1】根元素，即HTML元素

　　【2】float的值不为none

　　【3】overflow的值不为visible

　　【4】display的值为inline-block、table-cell、table-caption

　　【5】position的值为absolute或fixed

作用

【1】可以阻止元素被浮动元素覆盖

【2】可以包含浮动元素

【3】属于同一个BFC的两个相邻块级子元素的上下margin会发生重叠

1. 浏览器是怎样解析CSS选择器的？

答：样式系统从关键选择器开始匹配，然后左移查找规则选择器的祖先元素。

只要选择器的子树一直在工作，样式系统就会持续左移，直到和规则匹配，或者是因为不匹配而放弃该规则。

1. rem布局的优缺点
2. CSS3 动画——transform/transition/animate
3. Flex 布局，及适用场景

答：可以把列表放在同一个方向（从上到下排列，从左到右），并让列表能延伸到占用可用的空间。它的所有子元素自动成为容器成员，称为Flex项目（flex item），简称"项目"。常规布局是基于块和内联流方向，而Flex布局是基于flex-flow流可以很方便的用来做局中，能对不同屏幕大小自适应。在布局上有了比以前更加灵活的空间。

1. 为什么要清除浮动？清除方法有哪些？

答：浮动元素的父容器高度塌陷；

解决办法：

* 父元素定义高度；
* 父元素设置 overflow：hidden；
* 父元素浮动；
* .clearfix 类：

.clearfix::before,.clearfix:after{

Display:block;

Content: “.”;

Visibility:hidden;

Height:0;

Clear:both;

\*zoom:1; //触发 IE layout

}

## JavaScript

1. String 对象的 api
2. Array 对象的 api
3. Object 对象的 api
4. 介绍JavaScript的基本数据类型。

答：7种基本类型：Undefined、Null、String、Number、Boolean、Object、Symbol

1. JavaScript原型，原型链 ? 有什么特点？

答：每个构造函数都有一个原型对象，原型对象都包含一个指向构造函数的指针，而实例都包含一个指向原型对象的内部指针。当访问一个对象属性时，如果这个对象内部找不到，就会去prototype里找这个属性，而prototype里会有自己的prototype属性，于是一直找下去就是我们所说的原型链。

特点：JavaScript 对象是通过引用来传递的，我们创建的每个新对象实例中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象实例也会继承这一改变。

1. JavaScript如何实现继承？如何使用构造函数实现？

答：使用原型链实现继承。

构造函数也可实现：在子类的构造函数中调用超类的构造函数。

构造函数继承的五种方法：

1. 构造函数

function super(){}

function sub(){

super.apply(this,arguments);

}

1. Prototype模式

Sub.prototype = new super();

Sub.prototype.constructor = sub;

1. 直接继承prototype

Sub.prototype = super.prototype;

Sub.prototype.constructor = sub;

1. 利用空对象作中介

function F(){}

F.prototype = Super.prototype;

Sub.prototype = new F();

Sub.prototype.constructor = sub;

空对象不占内存。

1. 拷贝继承

实现拷贝函数

非构造函数实现继承：

1）Object（）方法

function object(o){

function F(){}

F.prototype = o;

return new F();

}

2）浅拷贝

1. 什么是闭包（closure），为什么要用它？

答：闭包就是有权访问另一个函数作用域中的变量的函数。创建闭包最常用的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量，利用闭包可以突破作用域链，将函数内部的变量和方法传递到外部。

闭包的特性：

* 函数内再嵌套函数；
* 内部函数可以引用外部函数的变量和参数；
* 参数和变量不会被垃圾回收机制回收。

1. 写一个通用的事件侦听器函数(机试题)。

答：见 EventUtils.js

1. 关于事件，IE与火狐的事件机制有什么区别？ 如何阻止冒泡？

答：用户操作可以被 JavaScript 侦测到的行为。

事件处理机制：IE是事件冒泡；Firefox 同时支持捕获和冒泡。

阻止冒泡：e.stopPropagation() || e.cancelBubble = false;

1. Ajax 是什么? 如何创建一个Ajax？

答：见 Ajax.js

1. 请介绍一下JS之事件节流（throttle）？

答：函数节流：一个函数执行一次后，只有大于设定的执行周期后才会执行第二次。

函数节流的应用场景是需要间隔一定时间触发回调来控制函数调用频率，如下：

* DOM 元素的拖拽功能实现（mousemove）
* 搜索联想（keyup）
* 计算鼠标移动的距离（mousemove）
* Canvas 模拟画板功能（mousemove）
* 射击游戏的 mousedown/keydown 事件（单位时间只能发射一颗子弹）

例如：监听滚动事件判断是否到页面底部自动加载更多：给 scroll 加了 debounce 后，只有用户停止滚动后，才会判断是否到了页面底部；如果是 throttle 的话，只要页面滚动就会间隔一段时间判断一次。

Throttle 函数示例：见 ThrottleAndDebounce.js

1. 什么是JS的函数防抖（debounce）？

答：防抖函数：一个需要频繁触发的函数，在规定时间内，只让最后一次生效，前面的不生效。

函数防抖的应用场景为，对于连续的事件响应我们只需要执行一次回调，有以下场景：

* 每次 resize/scroll 触发统计事件
* 本输入的验证（连续输入文字后发送 AJAX 请求进行验证，验证一次就好）

1. 事件循环

答：JavaScript 引擎分宏任务和微任务。宿主发起的任务称为宏任务，把JavaScript 引擎发起的任务称为微任务。许多微任务的队列组成了宏观任务。

例如：Promise产生的是 JavaScript 引擎内部的微任务，而 SetTimeout 是浏览器API，它产生宏任务。

如何分析异步执行的顺序？

* 首先我们分析有多少个宏任务；
* 在每个宏任务中，分析有多少个微任务；
* 根据调用次序，确定宏任务中的微任务执行次序；
* 根据宏任务的触发规则和调用次序，确定宏任务的执行次序；
* 确定整个顺序。

## ES6

1. Promise 的实现原理

答：

Promise 对象的特点：

（1）对象的状态不受外界影响。Promise对象代表一个异步操作，有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）。

（2）一旦状态改变，就不会再变，任何时候都可以得到这个结果。

Promise 优点：

有了Promise对象，就可以将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数。此外，Promise对象提供统一的接口，使得控制异步操作更加容易。

Promise 缺点：

首先，无法取消Promise，一旦新建它就会立即执行，无法中途取消。其次，如果不设置回调函数，Promise内部抛出的错误，不会反应到外部。第三，当处于pending状态时，无法得知目前进展到哪一个阶段（刚刚开始还是即将完成）。

Then 方法

* + Promise 必须实现 then 方法
  + Then 必须接受两个函数作参数
  + Then 返回的必须是一个 Promise 实例

promise里面的then函数仅仅是注册了后续需要执行的代码，真正的执行是在resolve方法里面执行的，理清了这层，再来分析源码会省力的多。

1. Class 和 Module
2. Symbol

唯一的属性 key，必须使用字符串使用

1. Reflect 和 Proxy

ES6 为了操作对象而提供的新的 API，均包含13个方法

Proxy 为拦截器，用于修改某些操作的默认行为。

Reflect 的设计目的是：将Object对象的一些明显属于语言内部的方法（比如Object.defineProperty），放到Reflect对象上。让Object操作都变成函数行为。与Proxy方法一一对应。

Reflect 的第一个参数必须为 target 对象。

Proxy 的用法：

Var proxy = new Proxy(target, handler);

1. Async 和 Await （ES2016）

## jQuery

1. JQuery的源码看过吗？能不能简单概况一下它的实现原理？
2. jQuery 的属性拷贝(extend)的实现原理是什么，如何实现深拷贝？

答：实现浅拷贝的方式：

Object.assign(), Array.prototype.slice(),Array.prototype.concat()只拷贝一层

实现深拷贝的方式：

JSON.parse(JSON.stringify()) – 但是无法处理函数

手写递归算法

1. 谈一下Jquery中的bind(),live(),delegate(),on()的区别？
2. Sizzle 引擎的实现原理？
3. 事件绑定和事件循环？

## 浏览器

1. 常见的浏览器内核有哪些？
2. 答：Trident内核：IE,MaxThon,TT,The World,360,搜狗浏览器等。[又称MSHTML]
   1. Gecko内核：Netscape6及以上版本，FF,MozillaSuite/SeaMonkey等
   2. Presto内核：Opera7及以上。 [Opera内核原为：Presto，现为：Blink;]
   3. Webkit内核：Safari,Chrome等。 [ Chrome的：Blink（WebKit的分支）]
3. **请简述浏览器加载URL的过程。**

答：

浏览器核心知识点有浏览器模型、渲染原理、JS运行机制、JS引擎解析流程等。

涉及主要流程体系：

1. 从浏览器接收url到开启网络请求线程（这一部分可以展开浏览器的机制以及进程与线程之间的关系）
2. 开启网络线程到发出一个完整的http请求（这一部分涉及到dns查询，tcp/ip请求，五层因特网协议栈等知识）
3. 从服务器接收到请求到对应后台接收到请求（这一部分可能涉及到负载均衡，安全拦截以及后台内部的处理等等）
4. 后台和前台的http交互（这一部分包括http头部、响应码、报文结构、cookie等知识，可以提下静态资源的cookie优化，以及编码解码，如gzip压缩等）
5. 单独拎出来的缓存问题，http的缓存（这部分包括http缓存头部，etag，catch-control等）
6. 浏览器接收到http数据包后的解析流程（解析html-词法分析然后解析成dom树、解析css生成css规则树、合并成render树，然后layout、painting渲染、复合图层的合成、GPU绘制、外链资源的处理、loaded和domcontentloaded等）
7. CSS的可视化格式模型（元素的渲染规则，如包含块，控制框，BFC，IFC等概念）
8. JS引擎解析过程（JS的解释阶段，预处理阶段，执行阶段生成执行上下文，VO，作用域链、回收机制等等）
9. 其它（可以拓展不同的知识模块，如跨域，web安全，hybrid模式等等内容）

下面分别详细介绍每个阶段的知识点，

**从浏览器接收url到开启网络请求线程**

浏览器的进程、线程、js运行机制

进程可能包括**主控进程**，**插件进程，GPU，tab页（浏览器内核）**等等。

浏览器内核是**多线程**的，包含**GUI渲染线程、JS引擎线程、事件触发线程、定时器线程、网络请求线程。**

**解析URL：**

URL 包含protocal、host、port、path、query、fragment(#后面的hash值)

每一次网络请求都是单独的线程。

**开启网络线程到发出一个完整的http请求：**

DNS 域名解析查询到 IP（域名多时效率问题，可以考虑dns-prefetch）

TCP/IP 三次握手和四次挥手（tcp连接的浏览器最大限制为2-10个）

五层网络协议

**从服务器接收到请求到对应后台接收到请求：**

负载均衡（Nginx）

后台处理（Tomcat容器，安全拦截、跨域验证）

前后台交互（HTTP报文、头部字段、状态码）

Cookie优化（多域名拆分为子域名加载静态资源）

Gzip压缩

长连接与短连接

HTTP1.0和HTTP2.0（多路复用、首部压缩、二进制分帧、服务器推送）

HTTPS 加密原理

HTTP缓存（强缓存/弱缓存）

**解析页面流程**

解析HTML，构建DOM树（convention转换 bytes、Tokenizing分词、Lexing词法分析、DOM构建）

解析CSS，生成CSS规则树

合并DOM树和CSS规则，生成render树

布局render树（Layout/reflow），负责各元素尺寸、位置的计算

绘制render树（paint），绘制页面像素信息

浏览器会将各层的信息发送给GPU，GPU会将各层合成（composite），显示在屏幕上

Reflow(Layout)和Repaint

简单层和复合层

资源外链下载

CSS下载

JS资源加载与执行

阻塞，defer 和 async

Img 图片加载

**CSS 可视化格式模型**

包含块

控制框

BFC和IFC

定位体系和浮动

**JS解释引擎**

JS预处理（变量提升、分号补全）

JS执行

* + 执行上下文，执行堆栈概念（如全局上下文，当前活动上下文）
  + VO（变量对象）和AO（活动对象）
  + 作用域链
  + this机制等

**GC垃圾回收机制**

标记清除

引用计数

原文链接：<https://mp.weixin.qq.com/s/qMsf4DcMhn2cf0fXC-PLVA>

1. 如何解决浏览器跨域问题？
2. 如何避免XSS和CSRF攻击？

## 前端框架

1. 请简述 Vue 生命周期

答：根实例创建之前会调用很多方法（钩子函数），

beforeCreate // 该阶段还什么都没有，一般不会用到

created // 发送 ajax ，做初始化操作，该阶段 data 和 methods 已经被注入

beforeMount // 执行该函数之前需先检查有无 el 元素和 template, 二者均存在才进入该函数，没有什么实际意义

mounted // 真是DOM 渲染完了，可以操作 dom 了

beforeUpdate // 数据发生变化即可出发，一般使用 watch 替代该方法

updated // 数据变化后执行

beforeDestroy // 销毁之前可以清除定时器、解绑事件

destroyed // 用不到，没什么意义

1. Vue 父子组件的通信方式有哪几种？

答：发布/订阅模式（on/emit）

原理：父亲绑定一些事件，儿子触发这些事件，并将参数传递过去，这里是单向数据流。

另外：多个兄弟组件互相通信，可自定义 eventBus 对象（发布/订阅）

1. Vue 中事件函数修饰符有哪些？

答：stop/prevent/self/native/once/left/right

1. Vue 中常用的指令

答：v-bind,v-if,v-show,v-html,v-for,v-model,v-cload,v-on,v-once

1. 请简述 Vue的响应式原理。

答：vue 实例创建时，会遍历 data 选项的属性，用 Object.defineProperty 将其转为 getter/setter 属性并在内部追踪相关依赖，在属性被访问和修改时通知变化。

每个组件实例都有相应的 watcher 程序实例，它会在组件渲染的过程中把属性记录为依赖，当依赖项的 setter 被调用时，会通知 watcher 重新计算，从而使关联组件得以更新。

1. Vue-Router的工作原理。

答：两种方式，hash/history/abstract

1. 如何使用Vuex进行跨组件状态管理
2. 服务端渲染（如Vue SSR），首屏直出当然是最理想的方案。

## 前端工程化，构建、打包

1. Webpack 常用的配置项，配置文件包含哪些内容？

答：

Entry //源文件地址

Output // 目标文件地址

Resolve // alias 设置路径缩写，extensions 可省略文件类型后缀

Module // rules 使用各种 loader，

Plugin // webpack 插件

常见loader：

1. 样式

Css-loader // 解析css 中的代码

Style-loader //将css模块作为样式导出到DOM中

Less-loader // 加载和转义less文件

Sass-loader //加载和转义sass/scss文件

1. 脚本

Script-loader //在全局上下文中执行一次javascript文件，不需要解析

Babel-loader //加载ES6 代码后使用Babel转义为ES5后浏览器才能解析

1. Files文件

url-loader //多数用于加载图片资源,超过文件大小显示则返回data URL

raw-loader //加载文件原始内容(utf-8格式)

1. 加载框架

Vue-loader // 加载和转义 Vue 组件

常见Plugin：

VuelLoaderPlugin

CleanWebpackPlugin

HtmlWebpackPlugin // 依据一个简单的index.html模板，生成一个自动引用你打包后的JS文件的新index.html。

HotModuleReplacementPlugin //允许你在修改组件代码后，自动刷新实时预览修改后的效果。

extract-text-webpack-plugin

UglifyJsPlugin // 压缩 js 文件

ExtractTextPlugin // 分离 css 和 js 文件

1. Webpack热更新实现原理?

答:

1. css-loader 的原理

答：loader的原理就是 export 导出函数，解析css 文件中的代码

1. webpack 如何全局引用 jQuery

答：先把jQuery对象声明成为全局变量`jQuery`，再通过管道进一步又声明成为全局变量`$`。

1. loader 和 plugin 有什么区别？

答：

loader：用于对模块源码的转换，loader 描述了 webpack 如何处理非 JavaScript 模块，并且在 bundle 中引入这些依赖。Loader 可以将不同的语言（如 TypeScript）转换为 JavaScript，或者将内联图像转换为 dataURL。任何开发技术栈都可以使用 webpack。

Plugin： 目的在于解决 loader 无法实现的其他事，从打包优化和压缩，到重新定义环境变量，功能强大到可以处理各种各样的任务。Webpack提供了很多开箱即用的插件：CommonChunkPlugin主要用于提取第三方库和公共模块，避免首屏加载的bundle文件，或者按需加载的bundle文件体积过大，导致加载时间过长，是一把优化的利器。而在多页面应用中，更是能够为每个页面间的应用程序共享代码创建bundle。

Mode：可以在 config 文件里面配置，也可以在 CLI 参数中配置：webpack --mode=production（一般会选择在CLI，也就是npm scripts里面进行配置）。

Resolve ：resolver是个库，帮助webpack找到bundle需要引入的模块代码，打包时，webpack使用enhanced-resolve来解析路径。

Manifest：管理所有模块之间的交互。runtime将能够查询模块标识符，检索出背后对应的模块。

webpack原理：从配置文件定义的模块列表开始，处理应用程序，从入口文件开始递归构建一个依赖图，然后将所有模块打包为少量的bundle，通常只有一个，可由浏览器加载。

## 简历项目相关

1.实现左侧题型列表的点击、拖拽事件以及调整题目顺序的拖拽事件

mousemove/mouseup/mousedown

e = e || window.event;

x = e.clientX, y = e.clientY;

\_showMoveDom(e)

2.兼容 mui 关于 WebView 页面管理的部分接口，使业务模块开发人员可以不用关心接口实现，使用统一的接口完成 APP 和普通浏览器中的兼容；实现移动端引导页、节日欢迎页及首页的预加载和缓存机制；

3.本人还设计了第三方系统接入久其格格的单点登录系统和权限校验系统。

4.Webpack 常用的打包命令及常用配置项。

5.多页面应用包含哪些页面？单页面路由如何使用？

答：包含3个独立页面，首页、上报总览、数据查询。首页的路由包括：首页运行、首页设计、首页预览、首页切换、首页列表，及首页头部和内容为首页运行的二级路由.

使用的是h5history 模式

## 算法

对象数组排序

打乱数组

有序数组中任意两个元素之和为某一固定值的所有组合。

## 计算机基础

TCP 连接的三次握手和四次挥手，原理及可能的出错情况？

答：

先介绍一下相关术语，位码，即 TCP 标志位，共 6 种。

URG 紧急指针是否有效。为 1，表示某一位需要被优先处理。

ACK 确认号是否有效，一般置为 1。0 表示无效。

PSH 提示接收端应用程序立即从TCP缓冲区把数据读走。

RST 对方要求重新建立连接，复位。

SYN 请求建立连接，并在其序列号的字段进行序列号的初始值设定。建立连接，设置为1。

FIN 希望断开连接。

序列号 seq ： 占4个字节，用来标记数据段的顺序。

确认号ack ：占4个字节，期待收到对方下一个报文段的第一个数据字节的序号；序列号表示报文段携带数据的第一个字节的编号；而确认号指的是期望接收到下一个字节的编号；因此当前报文段最后一个字节的编号+1即为确认号。

**内容：**

第一次握手：建立连接时客户端发送 SYN=1 到服务器，并进入 SYN\_SENT 状态，等待服务器确认，同时发送随机序列号 seq\_num=X；

第二次握手：服务器收到 syn 包，必须确认客户端的 SYN，即 ACK= 1，ack\_num=X+1；同时而也发送自己的 syn 包，即 SYN=1，seq\_num = Y；此时服务器进入 SYN\_RECV 状态。

第三次握手：客户端收到服务器的 ack+syn 包，向服务器发送确认包 ack，seq\_num=X+1,ACK=1,ack=Y+1。此包发送完毕，客户端和服务器进入 ESTABLISHED 状态。

**必要性**：TCP 通过三次握手方可建立**可靠**的**全双工**连接。

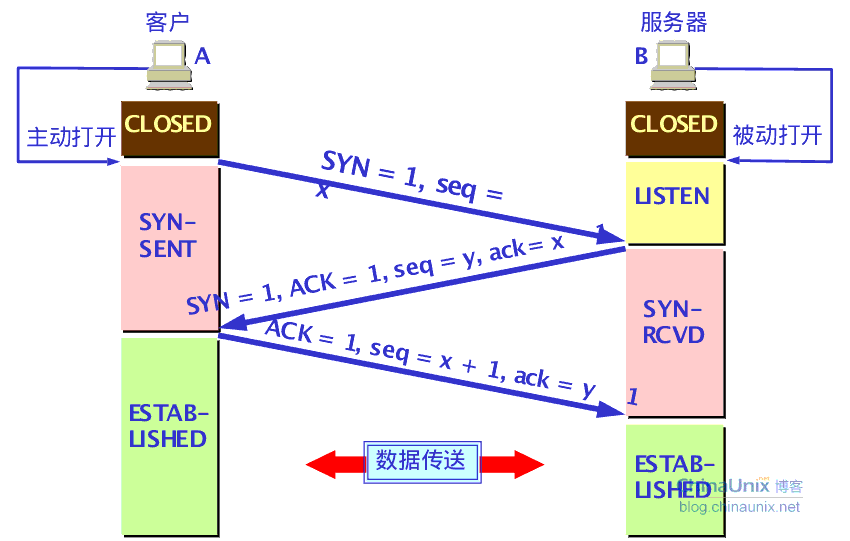
1）第一次握手和第二次握手（ACK部分）建立了从客户端到服务器传送数据的可靠连接；

2）第二次握手（SYN部分）和第三次握手建立了从服务器到客户端传送数据的可靠连接；

3）由于我们期望建立全双工连接，所以两个方向的通信都是需要的，于是合并了服务器发送的ACK和SYN。

**4）第三次握手的必要性：防止已失效的请求报文段突然又传送到了服务端而造成连接的误判**。

示意图：



三次握手示意图

四次挥手**内容**：

1）客户端进程发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入FIN-WAIT-1（终止等待1）状态。 TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。

2）服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。

3）客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）。

4）服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最后确认）状态，等待客户端的确认。

5）客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过**2∗MSL**（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。

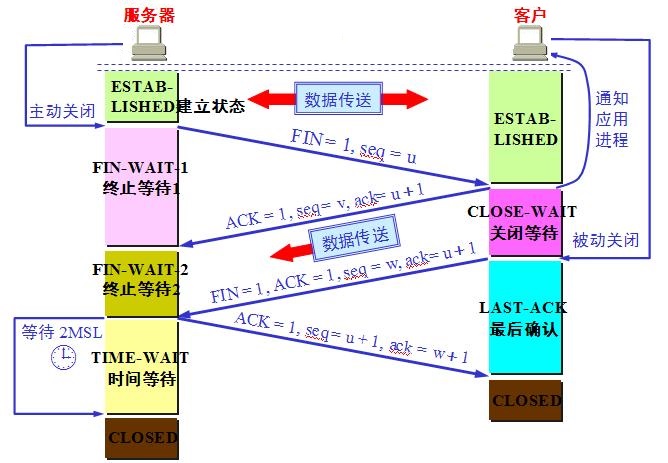
6）服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

**必要性**：为保证单向通信的可行性，所以多一次握手。

1）主动断开方发送FIN时，被动断开方要回复ACK，意思是“我收到你的FIN了”；

2）主动断开方发送FIN并不意味着立即关闭TCP连接，而是告诉对方自己没有更多的数据要发送了，只有当对方发完自己的数据再发送FIN后，才意味着关闭TCP连接；

3）被动断开方收到FIN并回复ACK后，此时TCP处于“半关闭”状态，为保证被动断开方可以继续发送数据，所以第二个FIN并不会伴随ACK发送，所以比连接时多一个报文段。



TCP的四次挥手示意图

状态迁移过程：

客户端：　　CLOSED->SYN\_SENT->ESTABLISHED->FIN\_WAIT\_1->FIN\_WAIT\_2->TIME\_WAIT->CLOSED

服务端

CLOSED->LISTEN->SYN\_RECEIVED->ESTABLISHED->CLOSE\_WAIT->LAST\_ACK->CLOSE

【问题2】为什么TIME\_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？

答：虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假象网络是不可靠的，有可以最后一个ACK丢失。所以TIME\_WAIT状态就是用来重发可能丢失的ACK报文。

MSL是Maximum Segment Lifetime英文的缩写，中文可以译为“报文最大生存时间”，他是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。因为tcp报文（segment）是ip数据报（datagram）的数据部分，具体称谓请参见《数据在网络各层中的称呼》一文，而ip头中有一个TTL域，TTL是time to live的缩写，中文可以译为“生存时间”，这个生存时间是由源主机设置初始值但不是存的具体时间，而是存储了一个ip数据报可以经过的最大路由数，每经过一个处理他的路由器此值就减1，当此值为0则数据报将被丢弃，同时发送ICMP报文通知源主机。RFC 793中规定MSL为2分钟，实际应用中常用的是30秒，1分钟和2分钟等。

2MSL即两倍的MSL，TCP的TIME\_WAIT状态也称为2MSL等待状态，当TCP的一端发起主动关闭，在发出最后一个ACK包后，即第3次握手完成后发送了第四次握手的ACK包后就进入了TIME\_WAIT状态，必须在此状态上停留两倍的MSL时间，等待2MSL时间主要目的是怕最后一个ACK包对方没收到，那么对方在超时后将重发第三次握手的FIN包，主动关闭端接到重发的FIN包后可以再发一个ACK应答包。在TIME\_WAIT状态时两端的端口不能使用，要等到2MSL时间结束才可继续使用。当连接处于2MSL等待阶段时任何迟到的报文段都将被丢弃。不过在实际应用中可以通过设置SO\_REUSEADDR选项达到不必等待2MSL时间结束再使用此端口。TTL与MSL是有关系的但不是简单的相等的关系，MSL要大于等于TTL。

【问题4】如果已经建立了连接，但是客户端突然出现故障了怎么办？

TCP还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75分钟发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。