Web前端进阶面试题 JS47道+http20道+数据结构8道，共计75道15394字

一、JavaScript提升篇

1、什么是跨域?

跨域需要针对浏览器的同源策略来理解，同源策略指的是请求必须是同一个端口，同一个协议，同一个域名，不同源的客户端脚本在没有明确授权的情况下，不能读写对方资源。

受浏览器同源策略的影响，不是同源的脚本不能操作其他源下面的对象。想要操作另一个源下的对象是就需要跨域。

2、什么是原型?

原型链：简单来讲就是原型组成的链，比如函数的原型是Function，Function的原型是Object，Object的原型仍然是Object，一直追溯到最终的原型对象。

函数通过prototype来追溯原型对象，对象通过*proto*来追溯原型对象。

通过一个构造函数创建出来的多个实例，如果都要添加一个方法，给每个实例去添加并不是一个明智的选择。这时就该用上原型了。

在实例的原型上添加一个方法，这个原型的所有实例便都有了这个方法。

原型链继承：

```
function A(){
    this.name="amy";
}
function B(){
    this.age="20"; //B继承了A,通过原型，形成链条
}
B.prototype=new A();
var a=new B();
console.log(a.name)//结果：amy
```

3、什么是闭包?

闭包就是一个函数引用另外一个函数的变量，因为变量被引用着所以不会被回收，它的最大用处有两个，一个是可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中，不会在外部函数调用后被自动清除。

由于局部变量无法共享和长久的保存，而全局变量可能造成变量污染，闭包的出现可以解决长久的保存变量又不会造成全局污染。

4、如何防抖?

何为防抖 多次触发事件后，事件处理函数只执行一次，并且是在触发操作结束时执行，一般用于scroll事件。

解决原理 对处理函数进行延时操作，若设定的延时到来之前再次触发事件，则清除上一次的延时操作定时器，重新定时。

```
let timer;
window.onscroll  = function () {
    if(timer){
        clearTimeout(timer)
    }
    timer = setTimeout(function () {
        //滚动条位置
        let scrollTop = document.body.scrollTop ||
document.documentElement.scrollTop;
        console.log('滚动条位置：' + scrollTop);
        timer = undefined;
    },200)
}
```

5、TCP的三次握手和四次挥手

TCP的三次握手和四次挥手实质就是TCP通信的连接和断开。

三次握手：为了对每次发送的数据量进行跟踪与协商，确保数据段的发送和接收同步，根据所接收到的数据量而确认数据发送、接收完毕后何时撤消联系，并建立虚连接。

四次挥手：即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。

1、三次握手

TCP协议位于传输层，作用是提供可靠的字节流服务，为了准确无误地将数据送达目的地，TCP协议采纳三次握手策略。

三次握手原理：

第1次握手：客户端发送一个带有SYN（synchronize）标志的数据包给服务端；

第2次握手：服务端接收成功后，回传一个带有SYN/ACK标志的数据包传递确认信息，表示我收到了；

第3次握手：客户端再回传一个带有ACK标志的数据包，表示我知道了，握手结束。

2、四次挥手

由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

四次挥手原理：

第1次挥手：客户端发送一个FIN，用来关闭客户端到服务端的数据传送，客户端进入FIN_WAIT_1状态；

第2次挥手：服务端收到FIN后，发送一个ACK给客户端，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），服务端进入CLOSE_WAIT状态；

第3次挥手：服务端发送一个FIN，用来关闭服务端到客户端的数据传送，服务端进入LAST_ACK状态；

第4次挥手：客户端收到FIN后，客户端t进入TIME_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，服务端进入CLOSED状态，完成四次挥手。

6、new 操作符原理

1. 创建一个类的实例：创建一个空对象 obj，然后把这个空对象的**proto**设置为构造

函数的 prototype。

2. 初始化实例：构造函数被传入参数并调用，关键字 this 被设定指向该实例 obj。
3. 返回实例 obj。
4.

7、事件委托做了什么

把一个元素响应事件（click、keydown……）的函数委托到另一个元素；

优点：减少内存消耗、动态绑定事件。

8、事件代理是什么

事件代理是利用事件的冒泡原理来实现的，何为事件冒泡呢？就是事件从最深的节点开

始，然后逐步向上传播事件，举个例子：页面上有这么一个节点树，div>ul>li>a;比如给

最里面的 a 加一个 click 点击事件，那么这个事件就会一层一层的往外执行，执行顺序

a>li>ul>div，有这样一个机制，那么我们给最外面的 div 加点击事件，那么里面的 ul，li，

a 做点击事件的时候，都会冒泡到最外层的 div 上，所以都会触发，这就是事件代理，

代理它们父级代为执行事件。

9、Eventloop

任务队列中，在每一次事件循环中，macrotask 只会提取一个执行，而 microtask 会一直提取，直到 microsoft 队列为空为止。

也就是说如果某个 microtask 任务被推入到执行中，那么当主线程任务执行完成后，会循环调用该队列任务中的下一个任务来执行，直到该任务队列到最后一个任务为止。而事件循环每次只会入栈一个 macrotask,主线程执行完成该任务后又会检查 microtasks 队列并完成里面的所有任务后再执行 macrotask 的任务。macrotasks: setTimeout, setInterval, setImmediate, I/O, UI rendering

microtasks: process.nextTick, Promise, MutationObserver

10、 如何实现跨域

JSONP：通过动态创建 script，再请求一个带参网址实现跨域通信。document.domain + iframe 跨域：两个页面都通过 js 强制设置 document.domain 为基础主域，就实现了同域。

location.hash + iframe 跨域：a 欲与 b 跨域相互通信，通过中间页 c 来实现。 三个页面，不同域之间利用 iframe 的 location.hash 传值，相同域之间直接 js 访问来通信。

window.name + iframe跨域：通过iframe的src属性由外域转向本地域，跨域数据即由iframe的 window.name 从外域传递到本地域。

postMessage 跨域：可以跨域操作的 window 属性之一。

CORS：服务端设置 Access-Control-Allow-Origin 即可，前端无须设置，若要带 cookie 请求，前后端都需要设置。

代理跨域：起一个代理服务器，实现数据的转发

11、写出原生 Ajax

Ajax 能够在不重新加载整个页面的情况下与服务器交换数据并更新部分网页内容，实现局部刷新，大大降低了资源的浪费，是一门用于快速创建动态网页的技术，ajax 的使用分为四部分：

1、创建 XMLHttpRequest 对象 var xhr = new XMLHttpRequest();

2、向服务器发送请求，使用 xmlHttpRequest 对象的 open 和 send 方法，

3、监听状态变化，执行相应回调函数

```
var xhr = new XMLHttpRequest();
xhr.open('get', 'aabb.php', true);
xhr.send(null);
xhr.onreadystatechange = function() {
if(xhr.readyState==4) {
```

```
if(xhr.status==200) {
console.log(xhr.responseText);
}
}
}
```

12、暂时性死区是什么，举1-2例说明

```
function f3(i){
   let i;
   console.log(i)
 };
f3(10)

let x = y,y = 10;
 function f2(){
   console.log(x,y)
 };
```

13、promise解决回调陷阱的链式写法

```
let p = new Promise(function(resolve, reject){
  if (true){
    resolve("成功");
   } else {
    reject("失败");
   }
})
p.then(function(v){
  return new Promise(function(resolve, reject){
    if (true){
      resolve("成功");
     } else {
      reject("失败");
     }
  })
}, function(v){
  console.log(v)
}).then(function(){
  console.log(4)
}, function(){
  console.log(5)
})
```

14、Promise对象实现Ajax封装

```
const getJSON = function(url,type, data) {
  const promise = new Promise(function(resolve, reject){
   const xmlHttp = new XMLHttpRequest();
   xmlHttp.open(type, url);
   if(type =='get'){
       xmlHttp.send();
   }else {
      xmlHttp.setRequestHeader("Content-Type", "application/json");
      xmlHttp.send(JSON.stringify(data));
   };
   xmlHttp.responseType = "json";
   xmlHttp.onreadystatechange =function(){
     if (xmlHttp.readyState !== 4) return;
     if (xmlHttp.status === 200) {
      resolve(xmlHttp.response);
     } else {
      reject(new Error(xmlHttp.statusText));
     }
   };
  });
  return promise;
};
```

15、简述promise

Promise 对象是 CommonJS 工作组提出的一种规范，目的是为异步编程提供统一接口。每

一个异步任务返回一个 Promise 对象，该对象有一个 then 方法，允许指定回调函数。□

f1().then(f2);

一个 promise 可能有三种状态：等待（pending）、已完成（resolved，又称 fulfilled）、

已拒绝（rejected）。

promise 必须实现 then 方法（可以说，then 就是 promise 的核心），而且 then 必须返回

一个 promise，同一个 promise 的 then 可以调用多次，并且回调的执行顺序跟它们被定义

时的顺序一致。

then 方法接受两个参数，第一个参数是成功时的回调，在 promise 由“等待”态转换到□

“完成”态时调用，另一个是失败时的回调，在 promise 由“等待”态转换到“拒绝”态时调用。同时，then 可以接受另一个 promise 传入，也接受一个“类 then”的对象或

方法，即 thenable 对象

16、说说你对 proxy 的理解

vue 的数据劫持有两个缺点:

1、无法监听通过索引修改数组的值的变化

2、无法监听 object 也就是对象的值的变化

所以 vue2.x 中才会有$set 属性的存在

proxy 是 es6 中推出的新 api，可以弥补以上两个缺点，所以 vue3.x 版本用 proxy 替换

object.defineproperty。

17、JS垃圾回收与V8垃圾回收

js中垃圾回收一般采用标识清除法和引用计数法

v8中垃圾回收采用分代回收：新生代，老生代和大对象

18、什么是宏任务和微任务，两者有什么区别?

ES6 规范中，宏任务（macrotask）又称为 task，微任务（microtask）又称为 jobs。

宏任务是由宿主发起的，而微任务是由 JS 本身发起。比如，宏任务有：setTimeOut、setInterval、文件操作等；微任务有：Promise.then、Promise.catch等。

整个JS在运行过程中主要执行以下事件循环（Even loop）：

```
主程序从上往下执行同步任务

异步任务会被放入异步任务队列中

当同步任务执行完成后，会去异步任务队列中执行异步事件
```

在异步任务中还有宏任务和微任务的区别：

```
宏任务：setTimeout、setInterval、Ajax、DOM事件等

微任务：promise、async/await、Object.observe等
```

宏任务和微任务的执行顺序：

```
同步任务->微任务->宏任务
```

两者的区别：

```
宏任务：DOM渲染后触发，如setTimeout

微任务：DOM渲染前触发，如promise
```

19、说说 Cookie 和 Token 的区别?

为什么要有 Cookie 呢?

我们都知道一般接口但是通过 HTTP 协议来进行数据交换的，而 HTTP 协议的特点是，无状态，工作前通过三次握手建立连接，工作完成后立刻通过四次挥手断开连接，每次连接都是独立存在的，没有任何状态将请求串联成一个整体，因此每次都需要重新验证是身份，即耗费了性能，也给黑客的攻击留下隐患。

Cookie 的出现，是来弥补 HTTP 无状态的问题的，Cookie 可以作为一个状态保存的状态机，用来保存用户的相关登录状态，当第一次验证通过后，服务器可以通过 set-cookie 令客户端将自己的 cookie 保存起来，当下一次再发送请求的时候，直接带上 cookie 即可，而服务器检测到客户端发送的 cookie 与其保存的 cookie 值保持一致时，则直接信任该连接，不再进行验证操作。

Token

Token, 令牌，代表执行某些操作的权利的对象，简单来说，就是类似 cookie 的一种验证信息，客户端通过登录验证后，服务器会返回给客户端一个加密的 token，然后当客户端再次向服务器发起连接时，带上token，服务器直接对token进行校验即可完成权限校验。

token相对cookie的优势：
1、支持跨域访问 ，将token置于请求头中，而cookie是不支持跨域访问的；

2、无状态化， 服务端无需存储token ，只需要验证token信息是否正确即可，而session需要在服务端存储，一般是通过cookie中的sessionID在服务端查找对应的session；

3、 无需绑定到一个特殊的身份验证 方案（传统的用户名密码登陆），只需要生成的token是符合我们预期设定的即可；

4、 更适用于移动端 （Android，iOS，小程序等等），像这种原生平台不支持cookie；

5、 避免CSRF跨站伪造攻击；

6、 非常适用于RESTful API ,这样可以轻易与各种后端（java，.net，python...）相结合，去耦合

20、写入一个函数，将abcd以实参传入返回[1,2,3,4,5,6]

var a = 1, b = [2,3],c=[4,5],d=6;

答：

```
function f(...items) {
   console.log(items);
  console.log(...items);
}
var a = 1, b = [2,3],c=[4,5],d=6;
f(a, ...b, ...c, d);
```

21、微任务&宏任务常见笔记题一

```
async function f() {
 console.log(1);
 return 2
}
f().then(result => {
 console.log(result);
})
console.log(3);
请写出解析的结果：
```

结果：

22、微任务&宏任务常见笔记题二

```
setTimeout(()=>{
 console.log(1)
},200)
setTimeout(()=>{
 console.log(2)
},100)
new Promise(function(resolve, reject){
 resolve(4)
 console.log(3)
}).then(function(d){
 console.log(d)
})
console.log(5);
```

请写出解析的结果：

结果：

35421

23、前端如何优化网站性能?

1、减少 HTTP 请求数量

在浏览器与服务器进行通信时，主要是通过 HTTP 进行通信。浏览器与服务器需要经过三次握手，每次握手需要花费大量时间。而且不同浏览器对资源文件并发请求数量有限（不同浏览器允许并发数），一旦 HTTP 请求数量达到一定数量，资源请求就存在等待状态，这是很致命的，因此减少 HTTP 的 请求数量可以很大程度上对网站性能进行优化。

CSS Sprites：国内俗称CSS精灵，这是将多张图片合并成一张图片达到减少HTTP请求的一种解决方案，可以通过CSS的background属性来访问图片内容。这种方案同时还可以减少图片总字节数。

合并 CSS 和 JS 文件： 现在前端有很多工程化打包工具，如：grunt、gulp、webpack等。为了减少 HTTP 请求数量，可以通过这些工具再发布前将多个CSS或者多个JS合并成一个文件。

采用 lazyLoad： 俗称懒加载，可以控制网页上的内容在一开始无需加载，不需要发请求，等到用户操作真正需要的时候立即加载出内容。这样就控制了网页资源一次性请求数量。

2、控制资源文件加载优先级

浏览器在加载HTML内容时，是将HTML内容从上至下依次解析，解析到link或者script标签就会加载href或者src对应链接内容，为了第一时间展示页面给用户，就需要将CSS提前加载，不要受 JS 加载影响。

一般情况下都是CSS在头部，JS在底部。

3、利用浏览器缓存

浏览器缓存是将网络资源存储在本地，等待下次请求该资源时，如果资源已经存在就不需要到服务器重新请求该资源，直接在本地读取该资源。

4、减少重排（Reflow）

> 基本原理：重排是DOM的变化影响到了元素的几何属性（宽和高），浏览器会重新计算元素的几何属性，会使渲染树中受到影响的部分失效，浏览器会验证 DOM 树上的所有其它结点的visibility属性，这也是Reflow低效的原因。如果Reflow的过于频繁，CPU使用率就会急剧上升。

减少Reflow，如果需要在DOM操作时添加样式，尽量使用 增加class属性，而不是通过style操作样式。

5、减少 DOM 操作

6、图标使用 IconFont 替换

24、网页从输入网址到渲染完成经历了哪些过程?

大致可以分为如下7步：

输入网址；

发送到DNS服务器，并获取域名对应的web服务器对应的ip地址；

与web服务器建立TCP连接；

浏览器向web服务器发送http请求；

web服务器响应请求，并返回指定url的数据（或错误信息，或重定向的新的url地址）；

浏览器下载web服务器返回的数据及解析html源文件；

生成DOM树，解析css和js，渲染页面，直至显示完成；

25、JS中常见的内存泄漏

1、意外的全局变量

> 函数中意外的定义了全局变量，每次执行该函数都会生成该变量，且不会随着函数执行结束而释放。

2、未清除的定时器

> 定时器没有清除，它内部引用的变量，不会被释放。

3、脱离DOM的元素引用

> 一个dom容器删除之后，变量未置为null，则其内部的dom元素则不会释放。

4、持续绑定的事件

> 函数中addEventListener绑定事件，函数多次执行，绑定便会产生多次，产生内存泄漏。

5、闭包引起内存泄漏

> 比如事件处理回调，导致DOM对象和脚本中对象双向引用。

6、console.log

console.log的对象是不能被垃圾回收

26、如何解决跨域问题

（1）通过jsonp跨域

（2）跨域资源共享（CORS）

（3）nginx代理跨域

（4）nodejs中间件代理跨域

27、谈谈垃圾回收机制方式及内存管理

JavaScript 在定义变量时就完成了内存分配。当不在使用变量了就会被回收，因为其开销比较大，垃圾收集器会定期（周期性）找出那些不在继续使用的变量，然后释放其内存。

（1）垃圾回收

标记清除法

当变量进入环境时，将这个变量标记为'进入环境'。当标记离开环境时，标记为‘离开环境’。离开环境的变量会被回收

引用技计数法

跟踪记录每个值被引用的次数，如果没有被引用，就会回收

（2）内存管理

内存分配=》内存使用=》内存回收

28、深拷贝、浅拷贝、以及如何实现?

深拷贝和浅拷贝都是针对复杂类型来说的，深拷贝式是层层拷贝,浅拷贝是只拷贝一层。

浅拷贝：使用object.assign、直接用=赋值只拷贝地址,它是将数据中的所有数据引用下来,指向同一个存放地址，拷贝后的数据修改后,会影响到原数据中的对象数据。

深拷贝：JSON.parse(JSON. stringify...),递归拷贝每一层对象是内容拷贝，将数据中的所有数据都拷贝下来,对拷贝后的数据进行修改，不会影响到原数据。

可以使用for...in、扩展运算符...、递归等递归函数实现深拷贝

递归：递归就是一个函数调用其本身，通过栈来实现。每执行一个函数，就新建一个函数栈。

29、箭头函数和普通函数的区别

（1）普通函数

可以通过bind、call、apply改变this指向

可以使用new

（2）箭头函数

本身没有this指向，

它的this在定义的时候继承自外层第一个普通函数的this

被继承的普通函数的this指向改变，箭头函数的this指向会跟着改变

箭头函数外层没有普通函数时，this指向window

不能通过bind、call、apply改变this指向

使用new调用箭头函数会报错，因为箭头函数没有constructor

30、同步和异步的区别

同步是一直阻塞模式，如果一个请求需要等待回调，那么会一直等待下去，直到返回结果

异步是非阻塞模式，无需等待回调，可执行下一步的代码

31、什么叫优雅降级和渐进增强?

渐进增强（Progressive Enhancement）：一开始就针对低版本浏览器进行构建页面，完成基本的功能，然后再针对高级浏览器进行效果、交互、追加功能达到更好的体验。

优雅降级（Graceful Degradation）：一开始就构建站点的完整功能，然后针对浏览器测试和修复。比如一开始使用 CSS3 的特性构建了一个应用，然后逐步针对各大浏览器进行 hack 使其可以在低版本浏览器上正常浏览。

其实渐进增强和优雅降级并非什么新概念，只是旧的概念换了一个新的说法。在传统软件开发中，经常会提到向上兼容和向下兼容的概念。渐进增强相当于向上兼容，而优雅降级相当于向下兼容。

区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要。降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带

32、请解释JSONP的工作原理，以及它为什么不是真正的AJAX

JSONP 是一种非正式传输协议，允许用户传递一个callback给服务端，然后服务端返回数据时会将这个 callback 参数作为函数名来包裹住 JSON 数据，这样客户端就可以随意定制自己的函数来自动处理返回数据了。

当GET请求从后台页面返回时，可以返回一段JavaScript代码，这段代码会自动执行，可以用来负责调用后台页面中的一个callback函数。

为什么它不是真正的Ajax：

它们的实质不同

ajax的核心是通过xmlHttpRequest获取非本页内容

jsonp的核心是动态添加script标签调用服务器提供的js脚本

jsonp只支持get请求，ajax支持get和post请求

33、Object.assgin()、扩展运算符（三点运算符）的区别

Object.assgin()是浅拷贝

三点运算符第一层是深拷贝，其他的都是浅拷贝

34、介绍一下js的数据类型有哪些，值是如何存储的?

（1）数据类型

```
基本数据类型：Number、Boolean、null、undefined、Symbol（ES6新增，表示独一无二的值）和Bigint（ES10新增）

引用数据类型：Object
```

（2）如何存储

```
原始数据类型：直接存储在栈中，占据空间小、大小固定，属于被频繁使用数据，所以放入栈中存储。

引用数据类型：同时存储在栈和堆中，占据空间大，大小不固定。引用数据类型在栈中存储了指针，该指针指向堆中该实体的起始地址。当解释器寻找引用值时，会首先检索其在栈中的地址，取得地址后从堆中获得实体。
```

35、map和Object的区别

（1）意外的键：Map默认不包含任意键，只包含插入的键值；Object有一个原型、原型链的键名可能和自己在对象上设置的键名发生冲突；

（2）键的类型：Map键的类型是任意的；Object键的类型是string和symbol；

（3）键的顺序：Map有序的，迭代的时候以其插入的顺序返回键值；Object无序的；

（4）size：Map的长度可以通过size属性获取；Object需要手动计算；

（5）迭代：Map是可迭代的；Object需要通过获取键来迭代；

（6）性能：Map在频繁增删键值对的场景下表现更好；Object在频繁添加和删除键值对的场景下未作出优化；

36、async 和 promise 的区别

（1）Async/Await 代码看起来简洁一些，使得异步代码看起来像同步代码

（2）async await与Promise一样，是非阻塞的。

（3）async await是基于Promise实现的，可以说是改良版的Promise，它不能用于普通的回调函数。

37、js常见的设计模式

（1）单例模式

（2）工厂模式

（3）构造函数模式

（4）发布订阅者模式

（5）迭代器模式

（6）代理模式

38、如何判断js数据类型?

（1）typeof：可以判断出string，number，boolean，undefined，symbol，function，bigint，但判断 typeof(null) 时值为 'object'; 判断数组和对象时值均为 'object'

（2）instanceof：可以判断一个实例是否属于某种类型，也可以判断一个实例是否是其父类型或者祖先类型的实例

（3）constructor：除了undefined和null之外，其他类型都可以通过constructor来判断。但如果声明了一个构造函数，并且改变了它的原型执行，这种情况下constructor也不能准确判断

（4）Object.prototype.toString：判断一个对象只属于某种内置类型，但不能准确判断一个实例是否属于某种类型

（5）Array.isArray：判断是否为数组

39、原生对象和宿主对象?

原生对象是ECMAScript规定的对象，所有内置对象都是原生对象，比如Array、 Date、 RegExp等;

宿主对象是宿主环境比如浏览器规定的对象，用于完善是ECMAScript的执行环境，比如Document、Location、Navigator等。

40、描述new一个对象的过程

（1）创建一个新对象，新对象的隐式原型**proto**指向new的构造函数的显示原型proptotype

（2）改变this指向，将构造函数的作用域赋给新的对象，并且执行构造函数的代码，为新的对象添加属性

（3）返回新的对象（return this）

41、什么是原型？什么是原型链?

原型：每个构造函数都有一个原型对象，实例化出来的对象都有一个原型，指向的是构造函数的原型对象，原型对象里面有一个指针constructor，指向的是它的构造函数。所有的原型对象都是Object构造函数的实例。它的原型指的就是Object原型对象，在往上找Object原型对象的原型就是null了。所有的构造函数都是function构造函数的实例，所有构造函数的原型指的就是function原型对象。

> 原型链：当在实例化的对象中访问一个属性时，首先会在该对象内部(自身属性)寻找，如找不到，则会向其__proto__指向的原型中寻找，如仍找不到，则继续向原型中__proto__指向的上级原型中寻找，直至找到或Object.prototype.__proto__为止（值为null），这种链状过程即为原型链。

原型的作用：

1.数据共享 节约内存内存空间

2.实现继承

42、js 获取原型的方法?

（1）Object.getPrototypeOf()

（2）**proto**

（3）constructor.prototype

43、set 和map 的区别?

（1）Map是键值对，Set是值的集合，键和值可以是任何的值；

（2）Map可以通过get方法获取值，而set不能因为它只有值，set只能用has来判断，返回一个布尔值；

（3）Set的值是唯一的可以做数组去重，Map由于没有格式限制，可以做数据存储

44、说说防抖和节流

防抖：n秒后在执行该事件，若在n秒内被重复触发，则重新计时

节流: n秒内只运行一次，若在n秒内重复触发，只有一次生效

45、require和import之间的区别?

（1）require对应导出的方法是module.exports，import对应的方法是export default/export

（2）require 是CommonJs的语法，import 是 ES6 的语法标准。

（3）require是运行运行时加载模块里的所有方法（动态加载），import 是编译的时候调用（静态加载），不管在哪里引用都会提升到代码顶部。

（4）require 是CommonJs的语法，引入的是的是整个模块里面的对象，import 可以按需引入模块里面的对象

（5）require 导出是值的拷贝，import 导出的是值的引用

46、栈和堆的区别?

（1）申请方式的不同。栈由系统自动分配，而堆是人为申请开辟;

（2）申请大小的不同。栈获得的空间较小，而堆获得的空间较大;

（3）申请效率的不同。栈由系统自动分配，速度较快，而堆一般速度比较慢;

（4）存储内容的不同。栈在函数调用时，函数调用语句的下一条可执行语句的地址第一个进栈，然后函数的各个参数进栈，其中静态变量是不入栈的。而堆一般是在头部用一个字节存放堆的大小，堆中的具体内容是人为安排;

（5）底层不同。栈是连续的空间，而堆是不连续的空间。

（6）生长方向不同。堆的生长方向向上，内存地址由低到高；栈的生长方向向下，内存地址由高到低。

（7）管理方式不同。栈由操作系统自动分配释放，无需我们手动控制；堆的申请和释放工作由程序员控制，容易产生内存泄漏;

47、forEach、for...in 、for...of三者的区别

（1）forEach遍历数组，但不能使用break、continue和return语句

（2）for...in是用来循环带有字符串key的对象的方法。实际是为循环"enumerable"(可枚举)对象而设计的。Js基本数据类型自带的原型属性不可枚举,通过Object.defineProperty0方法指定enumeralbe为false的属性不可枚举。

（3）for...in循环出的是key，for...of循环出的是value。

（4）for...of数组对象都可以遍历，它是ES6中新增加的语法。一个数据结构只有部署了Symbol.iterator 属性, 才具有 iterator接口可以使用 for...of循环。for...of遍历对象需要通过和Object.keys()

二、http/浏览器

1、说一下http和https

https 的 SSL 加密是在传输层实现的。

(1)http 和 https 的基本概念

http: 超文本传输协议，是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从 WWW 服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。

https: 是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版，即 HTTP 下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。

https 协议的主要作用是：建立一个信息安全通道，来确保数组的传输，确保网站的真实性。

(2)http 和 https 的区别?

http 传输的数据都是未加密的，也就是明文的，网景公司设置了 SSL 协议来对 http 协议传输的数据进行加密处理，简单来说 https 协议是由 http 和 ssl 协议构建的可进行加密传输和身份认证的网络协议，比 http 协议的安全性更高。

主要的区别如下：

Https 协议需要 ca 证书，费用较高。

http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

使用不同的链接方式，端口也不同，一般而言，http 协议的端口为 80，https 的端口为 443

http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

(3)https 协议的工作原理

客户端在使用 HTTPS 方式与 Web 服务器通信时有以下几个步骤，如图所示。

客户使用 https url 访问服务器，则要求 web 服务器建立 ssl 链接。

web 服务器接收到客户端的请求之后，会将网站的证书（证书中包含了公钥），返回或者说传输给客户端。

客户端和 web 服务器端开始协商 SSL 链接的安全等级，也就是加密等级。

客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站。

web 服务器通过自己的私钥解密出会话密钥。

web 服务器通过会话密钥加密与客户端之间的通信。

(4)https 协议的优点

使用 HTTPS 协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比

http 协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。

HTTPS 是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。

谷歌曾在 2014 年 8 月份调整搜索引擎算法，并称“比起同等 HTTP 网站，采用 HTTPS 加密的网站在搜索结果中的排名将会更高”。

(5)https 协议的缺点

https 握手阶段比较费时，会使页面加载时间延长 50%，增加 10%~20%的耗电。

https 缓存不如 http 高效，会增加数据开销。

SSL 证书也需要钱，功能越强大的证书费用越高。

SSL 证书需要绑定 IP，不能再同一个 ip 上绑定多个域名，ipv4 资源支持不了这种消耗。

2、TCP 和 UDP 的区别

（1）TCP 是面向连接的，udp 是无连接的即发送数据前不需要先建立链接。

（2）TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保证可靠交付。 并且因为 tcp 可靠，面向连接，不会丢失数据因此适合大数据量的交换。

（3）TCP 是面向字节流，UDP 面向报文，并且网络出现拥塞不会使得发送速率降低（因此会出现丢包，对实时的应用比如 IP 电话和视频会议等）。

（4）TCP 只能是 1 对 1 的，UDP 支持 1 对 1,1 对多。

（5）TCP 的首部较大为 20 字节，而 UDP 只有 8 字节。

（6）TCP 是面向连接的可靠性传输，而 UDP 是不可靠的。

3、fetch 发送 2 次请求的原因

fetch 发送 post 请求的时候，总是发送 2 次，第一次状态码是 204，第二次才成功?

原因很简单，因为你用 fetch 的 post 请求的时候，导致 fetch 第一次发送了一个 Options 请求，询问服务器是否支持修改的请求头，如果服务器支持，则在第二次中发送真正的请求。

4、Cookie、sessionStorage、localStorage 的区别

共同点：都是保存在浏览器端，并且是同源的

Cookie：cookie 数据始终在同源的 http 请求中携带（即使不需要），即 cookie 在浏览器和服务器间来回传递。而 sessionStorage 和 localStorage 不会自动把数据发给服务器，仅在本地保存。cookie 数据还有路径（path）的概念，可以限制 cookie 只属于某个路径下，存储的大小很小只有 4K 左右。（key：可以在浏览器和服务器端来回传递，存储容量小，只有大约 4K 左右）

sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持，localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie 只在设置的 cookie 过期时间之前一直有效，即使窗口或浏览器关闭。（key：本身就是一个回话过程，关闭浏览器后消失，session 为一个回话，当页面不同即使是同一页面打开两次，也被视为同一次回话）

localStorage：localStorage 在所有同源窗口中都是共享的；cookie 也是在所有同源窗口中都是共享的。（key：同源窗口都会共享，并且不会失效，不管窗口或者浏览器关闭与否都会始终生效）

补充说明一下 cookie 的作用：

保存用户登录状态。例如将用户 id 存储于一个 cookie 内，这样当用户下次访问该页面时就不需要重新登录了，现在很多论坛和社区都提供这样的功能。 cookie 还可以设置过期时间，当超过时间期限后，cookie 就会自动消失。因此，系统往往可以提示用户保持登录状态的时间：常见选项有一个月、三个 月、一年等。

5、iframe 是什么？有什么缺点?

定义：iframe 元素会创建包含另一个文档的内联框架

提示：可以将提示文字放在之间，来提示某些不支持 iframe 的浏览器

缺点：

会阻塞主页面的 onload 事件

搜索引擎无法解读这种页面，不利于 SEO

iframe 和主页面共享连接池，而浏览器对相同区域有限制所以会影响性能。

6、Cookie 如何防范 XSS 攻击

XSS（跨站脚本攻击）是指攻击者在返回的 HTML 中嵌入 javascript 脚本，为了减轻这些攻击，需要在 HTTP 头部配上，set-cookie：

httponly-这个属性可以防止 XSS,它会禁止 javascript 脚本来访问 cookie。

secure - 这个属性告诉浏览器仅在请求为 https 的时候发送 cookie。

结果应该是这样的：Set-Cookie=

7、介绍知道的 http 返回的状态码

100 Continue 继续。客户端应继续其请求

101 Switching Protocols 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到 HTTP 的新版本协议

200 OK 请求成功。一般用于 GET 与 POST 请求

201 Created 已创建。成功请求并创建了新的资源

202 Accepted 已接受。已经接受请求，但未处理完成

203 Non-Authoritative Information 非授权信息。请求成功。但返回的 meta 信息不在原始的服务器，而是一个副本

204 No Content 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，

可确保浏览器继续显示当前文档205 Reset Content 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文

档视图。可通过此返回码清除浏览器的表单域

206 Partial Content 部分内容。服务器成功处理了部分 GET 请求

300 Multiple Choices 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择

301 Moved Permanently 永久移动。请求的资源已被永久的移动到新 URI，返回信息会包括新的 URI，浏览器会自动定向到新 URI。今后任何新的请求都应使用新的 URI 代替

302 Found 临时移动。与 301 类似。但资源只是临时被移动。客户端应继续使用原有 URI

303 See Other 查看其它地址。与 301 类似。使用 GET 和 POST 请求查看

304 Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源

305 Use Proxy 使用代理。所请求的资源必须通过代理访问

306 Unused 已经被废弃的 HTTP 状态码

307 Temporary Redirect 临时重定向。与 302 类似。使用 GET 请求重定向

400 Bad Request 客户端请求的语法错误，服务器无法理解

401 Unauthorized 请求要求用户的身份认证

402 Payment Required 保留，将来使用

403 Forbidden 服务器理解请求客户端的请求，但是拒绝执行此请求

404 Not Found 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面

405 Method Not Allowed 客户端请求中的方法被禁止

406 Not Acceptable 服务器无法根据客户端请求的内容特性完成请求

407 Proxy Authentication Required 请求要求代理的身份认证，与 401 类似，但请求者应当使用代理进行授权

408 Request Time-out 服务器等待客户端发送的请求时间过长，超时

409 Conflict 服务器完成客户端的 PUT 请求是可能返回此代码，服务器处理请求时发生了冲突

410 Gone 客户端请求的资源已经不存在。410 不同于 404，如果资源以前有现在被永久删除了可使用 410 代码，网站设计人员可通过 301 代码指定资源的新位置

411 Length Required 服务器无法处理客户端发送的不带 Content-Length 的请求信息

412 Precondition Failed 客户端请求信息的先决条件错误

413 Request Entity Too Large 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个 Retry-After 的响应信息

414 Request-URI Too Large 请求的 URI 过长（URI 通常为网址），服务器无法处理

415 Unsupported Media Type 服务器无法处理请求附带的媒体格式

416 Requested range not satisfiable 客户端请求的范围无效

417 Expectation Failed 服务器无法满足 Expect 的请求头信息

500 Internal Server Error 服务器内部错误，无法完成请求

501 Not Implemented 服务器不支持请求的功能，无法完成请求

502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应503 Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求。

延时的长度可包含在服务器的 Retry-After 头信息中

504 Gateway Time-out 充当网关或代理的服务器，未及时从远端服务器获取请求

505 HTTP Version not supported 服务器不支持请求的 HTTP 协议的版本，无法完成处理；

8、强缓存、协商缓存什么时候用哪个

因为服务器上的资源不是一直固定不变的，大多数情况下它会更新，这个时候如果我们还访问本地缓存，那么对用户来说，那就相当于资源没有更新，用户看到的还是旧的资源；所以我们希望服务器上的资源更新了浏览器就请求新的资源，没有更新就使用本地的缓存，以最大程度的减少因网络请求而产生的资源浪费。

9、前端优化

降低请求量：合并资源，减少 HTTP 请求数，minify / gzip 压缩，webP，lazyLoad。

加快请求速度：预解析 DNS，减少域名数，并行加载，CDN 分发。

缓存：HTTP 协议缓存请求，离线缓存 manifest，离线数据缓存 localStorage。

渲染：JS/CSS 优化，加载顺序，服务端渲染，pipeline。

10、GET 和 POST 的区别

get 参数通过 url 传递，post 放在 request body 中。

get 请求在 url 中传递的参数是有长度限制的，而 post 没有。

get 比 post 更不安全，因为参数直接暴露在 url 中，所以不能用来传递敏感信息。□

get 请求只能进行 url 编码，而 post 支持多种编码方式get 请求会浏览器主动 cache，而 post 支持多种编码方式。

get 请求参数会被完整保留在浏览历史记录里，而 post 中的参数不会被保留。

GET 和 POST 本质上就是 TCP 链接，并无差别。但是由于 HTTP 的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包。

11、输入 URL 到页面加载显示完成发生了什么?

DNS 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

12、CDN的优化原理

许多网站运营者都信奉着一个原则，这个原则指导着运营者们优化网站的访问速度、页面效果等，它就是“8秒原则”。“8秒原则”是指，用户在打开网站时，记载时间不能超过8秒，一旦时间过长，网站将会失去这个用户，即便网站的页面制作精美、内容丰富。

如今，越来越多的站长为了提升网站的访问速度，使用cdn加速来为网站加持。

cdn加速的工作原理

cdn加速的工作原理，就是将源站的各类资源像复制粘贴一样，缓存到全国各地甚至全球各地的cdn节点上，当用户对源站发起访问时，用户就可以获取到离自己最近的数据信息，不必到源站进行访问。这样，避免访问源站时的线路拥堵，也减轻了源站的访问压力，同时，让用户得到更快的访问体验。

cdn加速的好处

1、提高安全性

网站与cdn加速服务建立连接后，用户在访问时只能访问cdn节点，源站就会隐藏起来，在一定的程度上起到保护源站被攻击的风险。由于cdn加速的各个节点较为分散，攻击者在发起攻击时无法全部下手，增加了他们的攻击难度，攻击一个节点仅仅是影响一个节点的缓存访问而已。

2、提升网站的访问速度

cdn加速最直接的好处的就是大大提升了网站的访问速度，cdn加速可以突破带宽的速度瓶颈限制，扩大了带宽的可接待容量，用户在访问网站时就不会拥挤。cdn加速的多个节点布置，能够让用户在不同地方都能访问到最近的节点资源上，让用户更快获取消息。

3、网站不容易挂机

当网站同时间涌入巨大流量时，使用了cdn之后，可以减少网站宕机的情况，同时你的网站可以接收更多的流量。用户访问网站的时间提高了，跳出率将会大大降低，这也有利于网站的各类转化。

13、浏览器的内核分别是什么?

（1）Mozilla Firefox的Gecko

（2）Chrome的Blink（WebKit的分支）

（3）Opera的内核原为Presto，现为Blink

（4）IE的内核Trident

（5）Safari的内核WebKit

14、浏览器是如何渲染页面的?

渲染的流程如下：

1.解析HTML文件，创建DOM树。

自上而下，遇到任何样式（link、style）与脚本（script）都会阻塞（外部样式不阻塞后续外部脚本的加载）。

2.解析CSS。优先级：浏览器默认设置<用户设置<外部样式<内联样式<HTML中的style样式；

3.将CSS与DOM合并，构建渲染树（Render Tree）

4.布局和绘制，重绘（repaint）和重排（reflow）

15、GET请求方式的长度限制到底是多少?

误区：我们经常以为GET请求参数的大小存在限制，而POST请求参数大小的无限制的

实际上HTTP协议从没有规定GET/POST的请求长度显示是多少。对GET请求参数的限制是来源于浏览器或者web服务器，浏览器或web服务器限制了url的长度。对POST请求起限制作用的是服务器处理程序的处理能力。

再次强调：

```
HTTP 协议 未规定 GET 和 POST 的长度限制

GET 的最大长度显示是因为浏览器和 web 服务器限制了 URL 的长度

不同的浏览器和 WEB 服务器，限制的最大长度不一样

要支持 IE，则最大长度为 2083 byte，若只支持 Chrome，则最大长度 8182 byte
```

补充各大浏览器对url的最大长度限制：

```
Firefox：对Firefox浏览器URL的长度限制为：65536个字符。

Safari：URL最大长度限制为80000个字符。

Opera：URL最大长度限制为190000个字符。

Google(chrome)：URL最大长度限制为8182个字符。

Apache(Server)：能接受的最大url长度为8192个字符

Microsoft Internet Information Server(IIS)：n能接受最大url的长度为16384个字符。
```

注意：（若长度超限，则服务端返回414标识）

16、什么是同源策略（Same origin policy）?

同源策略是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。

```
同源策略是浏览器的行为，是为了保护本地数据不被JavaScript代码获取回来的数据污染，因此拦截的是客户端发出的请求回来的数据接收，即请求发送了，服务器响应了，但是无法被浏览器接收。
```

同源策略的作用：

```
为了保护用户信息的安全，防止恶意的网络窃取
```

什么是同源：

```
端口、域名、协议相同
```

安全限制具体都阻止了哪些东西不可以被访问：

无法读取非同源策略下的cookie、localstorage

无法解除非同源的dom

无法向非同源的地址发送ajax请求

跨域访问：

跨域访问的解决方案是 CORS!

17、什么是跨域资源共享（CORS）？

CORS，全称Cross-Origin Resource Sharing，是一种允许当前域（domain）的资源（比如html/js/web service）被其他域（domain）的脚本请求访问的机制，通常由于同域安全策略（the same-origin security policy）浏览器会禁止这种跨域请求。

怎么用CORS：

CORS 可以配合 token 来防止 CSRF（跨站请求伪造） 攻击

18、什么是跨站请求伪造（CSRF）？

跨站请求伪造（英语：Cross-site request forgery），也被称为 one-click attack 或者 session riding，通常缩写为 CSRF 或者 XSRF， 是一种挟制用户在当前已登录的Web应用程序上执行非本意的操作的攻击方法。

简单的说是攻击者通过伪造用户的浏览器的请求，向一个用户自己曾经认证访问过的网站发送出去，使目标网站接收并误以为是用户的真实操作而去执行命令。常用于盗取账号、转账、发送虚假消息等

CSRF 攻击攻击原理及过程：

（1）用户 C 打开浏览器，访问受信任网站 A，输入用户名和密码请求登录网站 A；

（2）在用户信息通过验证后，网站 A 产生 Cookie 信息并返回给浏览器，此时用户登录网站 A 成功，可以正常发送请求到网站 A；

（3）用户未退出网站 A 之前，在同一浏览器中，打开一个 TAB 页访问网站 B；

（4）网站 B 接收到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点 A；

（5）浏览器在接收到这些攻击性代码后，根据网站 B 的请求，在用户不知情的情况下携带 Cookie 信息，向网站 A 发出请求。网站 A 并不知道该请求其实是由 B 发起的，所以会根据用户 C 的Cookie 信息以 C 的权限处理该请求，导致来自网站 B 的恶意代码被执行。

如何预防CSRF：

1、提交验证码

在表单中添加一个随机的数字或字母验证码。通过强制用户和应用进行交互。来有效地遏制CSRF攻击。

2、Referer Check

检查假设是非正常页面过来的请求，则极有可能是CSRF攻击。

3、token验证

在 HTTP 请求中以参数的形式添加一个随机产生的 token，并在服务器端建立一个拦截器来验证这个 token，假设请求中没有 token 或者 token 内容不对，则觉得可能是 CSRF 攻击而拒绝该请求。token 必须足够随机。敏感的操作应该使用POST，而不是GET。比如表单提交。

4、在HTTP头中自己定义属性并验证

这样的方法也是使用 token 并进行验证。这里并非把 token 以参数的形式置于 HTTP 请求之中，而是把它放到 HTTP 头中自己定义的属性里。通过 XMLHttpRequest 这个类，能够一次性给全部该类请求加上 csrftoken 这个 HTTP 头属性。并把 token 值放入当中。这样攻克了上种方法在请求中添加 token 的不便。同一时候，通过 XMLHttpRequest 请求的地址不会被记录到浏览器的地址栏，也不用操心 token 会透过 Referer 泄露到其它站点中去。

19、什么是跨站攻击（XXS）？

跨站攻击，即Cross Site Script Execution(通常简写为XSS)是指攻击者利用网站程序对用户输入过滤不足，输入可以显示在页面上对其他用户造成影响的HTML代码，从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式。

如何预防XXS：

1、HttpOnly防止获取cookie

在cookie中设置了HttpOnly属性，那么通过js脚本将无法读取到cookie信息，这样能有效的防止XSS攻击

2、输入检查（不要相信用户的所有输入）

3、输出检查（存的时候转义或者编码

20、HTTP的几种请求方法用途?

1、GET方法

发送一个请求来取得服务器上的某一资源

2、POST方法

向URL指定的资源提交数据或附加新的数据

3、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

4、HEAD方法

只请求页面的首部

5、DELETE方法

> 删除服务器上的某资源

6、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

7、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

8、CONNECT方法

> 把请求连接转换到透明的TCP/IP通道

三、数据结构和算法

1、什么是数组?

数组是由相同类型的元素（element）的集合所组成的数据结构，分配一块连续内存来存储。

特点：相同类型，连续内存，固定长度。

2、Js中的数组是真正的“数组“么?

不是真正的数组，因为js数组可以存放不同类型的值。

3、什么是队列?

一种遵循先进先出 (FIFO / First In First Out) 原则的一组有序的项；队列在尾部添加新元素，并从头部移除元素。最新添加的元素必须排在队列的末尾

4、 什么是链表？与数组的区别是?

存储有序的元素集合，但不同于数组，链表中的元素在内存中并不是连续放置的；每个元素由一个存储元素本身的节点和一个指向下一个元素的引用（指针/链接）组成。

数组：连续且固定长度空间，不能动态扩展，查找高效，添加修改元素低效。

链表：不需要连续内存空间，大小可动态变化，查找低效，添加修改高效。

5、什么是栈?

一种遵从先进后出 (LIFO) 原则的有序集合；新添加的或待删除的元素都保存在栈的末尾，称作栈顶，另一端为栈底。在栈里，新元素都靠近栈顶，旧元素都接近栈底。

栈的特点：后进先出（last-in,first-out）

6、什么是哈希及哈希冲突?

Hash也称散列、哈希，对应的英文都是Hash。基本原理就是把任意长度的输入，通过Hash算法变成固定长度的输出。

哈希冲突：不同的内容的hash值相同，即哈希冲突。

7、二叉树有几种遍历方式?

三种：先序遍历，中序遍历，后序遍历

8、简述冒泡排序?

基本思想: 冒泡排序，类似于水中冒泡，较大的数沉下去，较小的数慢慢冒起来，假设从小到大，即为较大的数慢慢往后排，较小的数慢慢往前排