JavaScript

1 闭包

- 闭包就是能够读取其他函数内部变量的函数
- 闭包是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见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链域
- 闭包的特性：
    - 函数内再嵌套函数
    - 内部函数可以引用外层的参数和变量
    - 参数和变量不会被垃圾回收机制回收

说说你对闭包的理解

- 使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念
- 闭包 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始终保持在内存中
- 闭包的另一个用处，是封装对象的私有属性和私有方法
- **好处**：能够实现封装和缓存等；
- **坏处**：就是消耗内存、不正当使用会造成内存溢出的问题

使用闭包的注意点

- 由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在IE中可能导致内存泄露
- 解决方法是，在退出函数之前，将不使用的局部变量全部删除

2 说说你对作用域链的理解

- 作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到`window`对象即被终止，作用域链向下访问变量是不被允许的
- 简单的说，作用域就是变量与函数的可访问范围，即作用域控制着变量与函数的可见性和生命周期

3 JavaScript原型，原型链？有什么特点?

- 每个对象都会在其内部初始化一个属性，就是`prototype`(原型)，当我们访问一个对象的属性时
- 如果这个对象内部不存在这个属性，那么他就会去`prototype`里找这个属性，这个`prototype`又会有自己的`prototype`，于是就这样一直找下去，也就是我们平时所说的原型链的概念
- 关系：`instance.constructor.prototype = instance.__proto__`
- 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体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变
- 当我们需要一个属性的时，`Javascript`引擎会先看当前对象中是否有这个属性， 如果没有的
- 就会查找他的`Prototype`对象是否有这个属性，如此递推下去，一直检索到`Object` 内建对象
- **原型：**
  - `JavaScript`的所有对象中都包含了一个`[__proto__]`内部属性，这个属性所对应的就是该对象的原型
  - JavaScript的函数对象，除了原型`[__proto__]`之外，还预置了`prototype`属性
  - 当函数对象作为构造函数创建实例时，该 prototype 属性值将被作为实例对象的原型`[__proto__]`。
- **原型链：**
  - 当一个对象调用的属性/方法自身不存在时，就会去自己`[__proto__]`关联的前辈`prototype`对象上去找
  - 如果没找到，就会去该`prototype`原型`[__proto__]`关联的前辈`prototype`去找。依次类推，直到找到属性/方法或`undefined`为止。从而形成了所谓的“原型链”
- **原型特点：**
  - `JavaScript`对象是通过引用来传递的，当修改原型时，与之相关的对象也会继承这一改变

4 请解释什么是事件代理

- 事件代理（`Event Delegation`），又称之为事件委托。是`JavaScript`中常用绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托

给父元素，让父元素担当事件监听的职务。事件代理的原理是DOM元素的事件冒泡。使用事件代理的好处是可以提高性能

- 可以大量节省内存占用，减少事件注册，比如在`table`上代理所有`td`的`click`事件就非常棒
- 可以实现当新增子对象时无需再次对其绑定

5 Javascript如何实现继承?

- 构造继承
- 原型继承
- 实例继承
- 拷贝继承
- 原型`prototype`机制或`apply`和`call`方法去实现较简单，建议使用构造函数与原型混合方式

```
function Parent(){
    this.name = 'wang';
}

function Child(){
        this.age = 28;
}

Child.prototype = new Parent();//继承了Parent，通过原型

var demo = new Child();
alert(demo.age);
alert(demo.name);//得到被继承的属性
```

6 谈谈This对象的理解

- `this`总是指向函数的直接调用者（而非间接调用者）
- 如果有`new`关键字，`this`指向`new`出来的那个对象
- 在事件中，`this`指向触发这个事件的对象，特殊的是，`IE`中的`attachEvent`中的`this`总是指向全局对象`Window`

7 事件模型

> `W3C`中定义事件的发生经历三个阶段：捕获阶段（`capturing`）、目标阶段（`targetin`）、冒泡阶段（`bubbling`）

- 冒泡型事件：当你使用事件冒泡时，子级元素先触发，父级元素后触发
- 捕获型事件：当你使用事件捕获时，父级元素先触发，子级元素后触发

- DOM事件流：同时支持两种事件模型：捕获型事件和冒泡型事件
- 阻止冒泡：在W3c中，使用stopPropagation()方法；在IE下设置cancelBubble = true
- 阻止捕获：阻止事件的默认行为，例如click - <a>后的跳转。在W3c中，使用preventDefault()方法，在IE下设置window.event.returnValue = false

8 new操作符具体干了什么呢?

- 创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型
- 属性和方法被加入到 this 引用的对象中
- 新创建的对象由 this 所引用，并且最后隐式的返回 this

9 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javascript来操作DOM而更新页面。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ajax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(){
    if(xhr.readyState == 4){
        if(xhr.status == 200){
            success(xhr.responseText);
        } else {
            /** false **/
            fail && fail(xhr.status);
        }
    }
}
```

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。

10 如何解决跨域问题?

> 首先了解下浏览器的同源策略 同源策略`/SOP（Same origin policy）`是一种约定，由Netscape公司1995年引入浏览器，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，浏览器很容易受到`XSS`、`CSFR`等攻击。所谓同源是指"**协议+域名+端口**"三者相同，即便两个不同的域名指向同一个ip地址，也非同源

那么怎样解决跨域问题的呢?

- **通过jsonp跨域**

```
var script = document.createElement('script');
script.type = 'text/javascript';

// 传参并指定回调执行函数为onBack
script.src = 'http://www.....:8080/login?user=admin&callback=onBack';
document.head.appendChild(script);

// 回调执行函数
function onBack(res) {
    alert(JSON.stringify(res));
}
```

- **document.domain + iframe跨域**

> 此方案仅限主域相同，子域不同的跨域应用场景

1.）父窗口：

```
<iframe id="iframe" src="http://child.domain.com/b.html">
</iframe>
<script>
    document.domain = 'domain.com';
    var user = 'admin';
</script>
```

2.）子窗口：

```
document.domain = 'domain.com';
// 获取父窗口中变量
alert('get js data from parent ---> ' + window.parent.user);
```

- **nginx代理跨域**
- **nodejs中间件代理跨域**
- **后端在头部信息里面设置安全域名**

11 模块化开发怎么做?

- 立即执行函数,不暴露私有成员

```
var module1 = (function(){
        var _count = 0;
        var m1 = function(){
            //...
        };
        var m2 = function(){
            //...
        };
        return {
            m1 : m1,
            m2 : m2
        };
})();
```

12 异步加载JS的方式有哪些?

- 设置`<script>`属性 async="async"（一旦脚本可用，则会异步执行）
- 动态创建`script DOM`：`document.createElement('script');`
- `XmlHttpRequest` 脚本注入
- 异步加载库 `LABjs`
- 模块加载器 `Sea.js`

13 那些操作会造成内存泄漏?

> JavaScript 内存泄露指对象在不需要使用它时仍然存在，导致占用的内存不能使用或回收

- 未使用 var 声明的全局变量
- 闭包函数(Closures)
- 循环引用(两个对象相互引用)
- 控制台日志(console.log)
- 移除存在绑定事件的DOM元素(IE)
- `setTimeout` 的第一个参数使用字符串而非函数的话，会引发内存泄漏
- 垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收

14 XML和JSON的区别?

- 数据体积方面
  - `JSON` 相对 `于XML` 来讲，数据的体积小，传递的速度更快些。
- 数据交互方面
  - `JSON` 与 `JavaScript` 的交互更加方便，更容易解析处理，更好的数据交互
- 数据描述方面
  - `JSON` 对数据的描述性比 `XML` 较差
- 传输速度方面
  - `JSON` 的速度要远远快于 `XML`

15 谈谈你对webpack的看法

- `WebPack` 是一个模块打包工具，你可以使用 `WebPack` 管理你的模块依赖，并编绎输出模块们所需的静态文件。它能够很好地管理、打包 `Web` 开发中所用到的 `HTML` 、 `Javascript` 、 `CSS` 以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源， `webpack` 有对应的模块加载器。 `webpack` 模块打包器会分析模块间的依赖关系，最后 生成了优化且合并后的静态资源

16 说说你对AMD和Commonjs的理解

- `CommonJS` 是服务器端模块的规范， `Node.js` 采用了这个规范。 `CommonJS` 规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。 `AMD` 规范则是非同步加载模块，允许指定回调函数
- `AMD` 推荐的风格通过返回一个对象做为模块对象， `CommonJS` 的风格通过对 `module.exports` 或 `exports` 的属性赋值来达到暴露模块对象的目的

17 常见web安全及防护原理

- `sql`注入原理
  - 就是通过把`SQL`命令插入到`web`表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令
- 总的来说有以下几点
  - 永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双`"-"`进行转换等
  - 永远不要使用动态拼装SQL，可以使用参数化的`SQL`或者直接使用存储过程进行数据查询存取
  - 永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接
  - 不要把机密信息明文存放，请加密或者`hash`掉密码和敏感的信息

XSS原理及防范

- `Xss(cross-site scripting)`攻击指的是攻击者往`Web`页面里插入恶意`html`标签或者`javascript`代码。比如：攻击者在论坛中放一个看似安全的链接，骗取用户点击后，窃取`cookie`中的用户私密信息；或者攻击者在论坛中加一个恶意表单，当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点

XSS防范方法

- 首先代码里对用户输入的地方和变量都需要仔细检查长度和对`”<”,”>”,”;”,”’”`等字符做过滤；其次任何内容写到页面之前都必须加以encode，避免不小心把`html tag`弄出来。这一个层面做好，至少可以堵住超过一半的XSS攻击

XSS与CSRF有什么区别吗?

- `XSS`是获取信息，不需要提前知道其他用户页面的代码和数据包。`CSRF`是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。要完成一次`CSRF`攻击，受害者必须依次完成两个步骤
- 登录受信任网站`A`，并在本地生成`Cookie`
- 在不登出`A`的情况下，访问危险网站`B`

CSRF的防御

- 服务端的`CSRF`方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数
- 通过验证码的方法

18 用过哪些设计模式?

- 工厂模式：
  - 工厂模式解决了重复实例化的问题，但还有一个问题,那就是识别问题，因为根本无法
  - 主要好处就是可以消除对象间的耦合，通过使用工程方法而不是`new`关键字
- 构造函数模式
  - 使用构造函数的方法，即解决了重复实例化的问题，又解决了对象识别的问题，该模式与工厂模式的不同之处在于
  - 直接将属性和方法赋值给 `this`对象;

19 为什么要有同源限制?

- 同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议
- 举例说明：比如一个黑客程序，他利用`Iframe`把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过`Javascript`读取到你的表单中`input`中的内容，这样用户名，密码就轻松到手了。

20 offsetWidth/offsetHeight,clientWidth/clientHeight与scrollWidth/scrollHeight的区别

- `offsetWidth/offsetHeight`返回值包含**content + padding + border**，效果与e.getBoundingClientRect()相同
- `clientWidth/clientHeight`返回值只包含**content + padding**，如果有滚动条，也**不包含滚动条**
- `scrollWidth/scrollHeight`返回值包含**content + padding + 溢出内容的尺寸**

21 javascript有哪些方法定义对象

- 对象字面量： `var obj = {};`
- 构造函数： `var obj = new Object();`
- Object.create(): `var obj = Object.create(Object.prototype);`

22 常见兼容性问题?

- `png24`位的图片在iE6浏览器上出现背景，解决方案是做成`PNG8`
- 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,，但是全局效率很低，一般是如下这样解决：

```
body,ul,li,ol,dl,dt,dd,form,input,h1,h2,h3,h4,h5,h6,p{
margin:0;
padding:0;
}
```

- `IE`下,`event`对象有`x`,`y`属性,但是没有`pageX`,`pageY`属性
- `Firefox`下,`event`对象有`pageX`,`pageY`属性,但是没有`x`,`y`属性.

23 说说你对promise的了解

- 依照 `Promise/A+` 的定义， `Promise` 有四种状态:
  - `pending:` 初始状态, 非 `fulfilled` 或 `rejected.`
  - `fulfilled:` 成功的操作.
  - `rejected:` 失败的操作.
  - `settled: Promise`已被`fulfilled`或`rejected`，且不是`pending`
- 另外， `fulfilled`与 `rejected`一起合称 `settled`
- `Promise` 对象用来进行延迟(`deferred`)和异步(`asynchronous`)计算

Promise 的构造函数

- 构造一个 `Promise`，最基本的用法如下:

```
var promise = new Promise(function(resolve, reject) {

        if (...) {  // succeed

            resolve(result);

        } else {   // fails

            reject(Error(errMessage));

        }
    });
```

- `Promise` 实例拥有 `then` 方法（具有 `then` 方法的对象，通常被称为 `thenable`）。它的使用方法如下:

```
promise.then(onFulfilled, onRejected)
```

- 接收两个函数作为参数，一个在 `fulfilled` 的时候被调用，一个在`rejected` 的时候被调用，接收参数就是 `future`， `onFulfilled` 对应`resolve`，`onRejected`对应 `reject`

24 你觉得jQuery源码有哪些写的好的地方

- `jquery`源码封装在一个匿名函数的自执行环境中，有助于防止变量的全局污染，然后通过传入`window`对象参数，可以使`window`对象作为局部变量使用，好处是当`jquery`中访问`window`对象的时候，就不用将作用域链退回到顶层作用域了，从而可以更快的访问window对象。同样，传入`undefined`参数，可以缩短查找`undefined`时的作用域链
- `jquery`将一些原型属性和方法封装在了`jquery.prototype`中，为了缩短名称，又赋值给了`jquery.fn`，这是很形象的写法
- 有一些数组或对象的方法经常能使用到，`jQuery`将其保存为局部变量以提高访问速度
- `jquery`实现的链式调用可以节约代码，所返回的都是同一个对象，可以提高代码效率

25 vue、react、angular

- `Vue.js` 一个用于创建 `web` 交互界面的库，是一个精简的 `MVVM`。它通过双向数据绑定把 `View` 层和 `Model` 层连接了起来。实际的 `DOM` 封装和输出格式都被抽象为了`Directives` 和 `Filters`
- `AngularJS` 是一个比较完善的前端`MVVM`框架，包含模板，数据双向绑定，路由，模块化，服务，依赖注入等所有功能，模板功能强大丰富，自带了丰富的`Angular`指令
- `react` `React` 仅仅是 `VIEW` 层是`facebook`公司。推出的一个用于构建`UI`的一个库，能够实现服务器端的渲染。用了`virtual dom`，所以性能很好。

26 Node的应用场景

- 特点：
  - 1、它是一个`Javascript`运行环境
  - 2、依赖于`Chrome V8`引擎进行代码解释
  - 3、事件驱动
  - 4、非阻塞`I/O`
  - 5、单进程，单线程
- 优点：
  - 高并发（最重要的优点）
- 缺点：
  - 1、只支持单核`CPU`，不能充分利用`CPU`
  - 2、可靠性低，一旦代码某个环节崩溃，整个系统都崩溃

27 谈谈你对AMD、CMD的理解

- `CommonJS`是服务器端模块的规范，`Node.js`采用了这个规范。`CommonJS`规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。`AMD`规范则是非同步加载模块，允许指定回调函数
- `AMD`推荐的风格通过返回一个对象做为模块对象，`CommonJS`的风格通过对`module.exports`或`exports`的属性赋值来达到暴露模块对象的目的

es6模块 CommonJS、AMD、CMD

- `CommonJS` 的规范中，每个 `JavaScript` 文件就是一个独立的模块上下文（`module context`），在这个上下文中默认创建的属性都是私有的。也就是说，在一个文件定义的变量（还包括函数和类），都是私有的，对其他文件是不可见的。
- `CommonJS`是同步加载模块,在浏览器中会出现堵塞情况，所以不适用
- `AMD` 异步，需要定义回调`define`方式
- `es6` 一个模块就是一个独立的文件，该文件内部的所有变量，外部无法获取。如果你希望外部能够读取模块内部的某个变量，就必须使用`export`关键字输出该变量 `es6`还可以导出类、方法，自动适用严格模式

28 那些操作会造成内存泄漏

- 内存泄漏指任何对象在您不再拥有或需要它之后仍然存在
- `setTimeout` 的第一个参数使用字符串而非函数的话，会引发内存泄漏
- 闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环)

29 web开发中会话跟踪的方法有哪些

- `cookie`
- `session`
- `url`重写
- 隐藏`input`
- `ip`地址

30 JS的基本数据类型和引用数据类型

- 基本数据类型：`undefined`、`null`、`boolean`、`number`、`string`、`symbol`
- 引用数据类型：`object`、`array`、`function`

31 介绍js有哪些内置对象

- `Object` 是 `JavaScript` 中所有对象的父对象
- 数据封装类对象：`Object`、`Array`、`Boolean`、`Number` 和 `String`
- 其他对象：`Function`、`Arguments`、`Math`、`Date`、`RegExp`、`Error`

32 说几条写JavaScript的基本规范

- 不要在同一行声明多个变量
- 请使用`===/!==`来比较`true/false`或者数值
- 使用对象字面量替代`new Array`这种形式
- 不要使用全局函数
- `Switch`语句必须带有`default`分支
- `If`语句必须使用大括号
- `for-in`循环中的变量 应该使用`var`关键字明确限定作用域，从而避免作用域污

33 JavaScript有几种类型的值

- 栈：原始数据类型（`Undefined`，`Null`，`Boolean`，`Number`、`String`）
- 堆：引用数据类型（对象、数组和函数）
- 两种类型的区别是：存储位置不同；
- 原始数据类型直接存储在栈(`stack`)中的简单数据段，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；
- 引用数据类型存储在堆(`heap`)中的对象,占据空间大、大小不固定,如果存储在栈中，将会影响程序运行的性能；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

34 javascript创建对象的几种方式

> `javascript`创建对象简单的说,无非就是使用内置对象或各种自定义对象，当然还可以用`JSON`；但写法有很多种，也能混合使用

- 对象字面量的方式

```
person=
{firstname:"Mark",lastname:"Yun",age:25,eyecolor:"black"};
```

- 用`function`来模拟无参的构造函数

```
function Person(){}
    var person=new Person();//定义一个function，如果使用new"实例化",该function可以看作是一个Class
        person.name="Mark";
        person.age="25";
        person.work=function(){
        alert(person.name+" hello...");
}
person.work();
```

- 用`function`来模拟参构造函数来实现（用`this`关键字定义构造的上下文属性)

```
function Pet(name,age,hobby){
        this.name=name;//this作用域：当前对象
        this.age=age;
        this.hobby=hobby;
        this.eat=function(){
            alert("我叫"+this.name+",我喜欢"+this.hobby+",是个程序员");
        }
}
var maidou =new Pet("麦兜",25,"coding");//实例化、创建对象
maidou.eat();//调用eat方法
```

- 用工厂方式来创建（内置对象）

```
var wcDog =new Object();
     wcDog.name="旺财";
     wcDog.age=3;
     wcDog.work=function(){
       alert("我是"+wcDog.name+",汪汪汪......");
     }
     wcDog.work();
```

- 用原型方式来创建

```
function Dog(){}
Dog.prototype.name="旺财";
Dog.prototype.eat=function(){
    alert(this.name+"是个吃货");
}
var wangcai =new Dog();
wangcai.eat();
```

- 用混合方式来创建

```
 function Car(name,price){
    this.name=name;
    this.price=price;
}
Car.prototype.sell=function(){
    alert("我是"+this.name+"，我现在卖"+this.price+"万元");
}
var camry =new Car("凯美瑞",27);
camry.sell();
```

35 eval是做什么的

- 它的功能是把对应的字符串解析成`JS`代码并运行
- 应该避免使用`eval`，不安全，非常耗性能（`2`次，一次解析成`js`语句，一次执行)
- 由`JSON`字符串转换为JSON对象的时候可以用`eval，var obj =eval('('+ str +')')`

36 null，undefined 的区别

- `undefined` 表示不存在这个值。
- `undefined` :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但是还没有定义。当尝试读取时会返回 `undefined`
- 例如变量被声明了，但没有赋值时，就等于`undefined`
- `null` 表示一个对象被定义了，值为“空值”
- `null` : 是一个对象(空对象, 没有任何属性和方法)
- 例如作为函数的参数，表示该函数的参数不是对象；
- 在验证`null`时，一定要使用 `===`，因为 `==`无法分别`null` 和 `undefined`

37 ["1", "2", "3"].map(parseInt) 答案是多少

- `[1, NaN, NaN]`因为 `parseInt` 需要两个参数 `(val, radix)`，其中`radix` 表示解析时用的基数。
- `map`传了 `3`个`(element, index, array)`，对应的 `radix` 不合法导致解析失败。

38 javascript 代码中的"use strict";是什么意思

- `use strict`是一种`ECMAscript 5`添加的（严格）运行模式,这种模式使得Javascript 在更严格的条件下运行,使`JS`编码更加规范化的模式,消除`Javascript`语法的一些不合理、不严谨之处，减少一些怪异行为

39 JSON 的了解

- `JSON(JavaScript Object Notation)`是一种轻量级的数据交换格式
- 它是基于`JavaScript`的一个子集。数据格式简单, 易于读写, 占用带宽小
- `JSON`字符串转换为JSON对象:

```
var obj =eval('('+ str +')');
var obj = str.parseJSON();
var obj = JSON.parse(str);
```

- `JSON`对象转换为JSON字符串:

```
var last=obj.toJSONString();
var last=JSON.stringify(obj);
```

40 js延迟加载的方式有哪些

- 设置`<script>`属性`defer="defer"`（脚本将在页面完成解析时执行）
- 动态创建`script DOM`：`document.createElement('script');`
- `XmlHttpRequest`脚本注入
- 延迟加载工具`LazyLoad`

41 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

42 渐进增强和优雅降级

- 渐进增强 ： 针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。
- 优雅降级 ： 一开始就构建完整的功能，然后再针对低版本浏览器进行兼容

43 defer和async

- `defer`并行加载`js`文件，会按照页面上`script`标签的顺序执行
- `async`并行加载`js`文件，下载完成立即执行，不会按照页面上`script`标签的顺序执行

44 说说严格模式的限制

- 变量必须声明后再使用
- 函数的参数不能有同名属性，否则报错
- 不能使用with语句
- 不能对只读属性赋值，否则报错
- 不能使用前缀0表示八进制数，否则报错
- 不能删除不可删除的属性，否则报错
- 不能删除变量`delete prop`，会报错，只能删除属性`delete global[prop]`
- `eval`不会在它的外层作用域引入变量
- `eval`和`arguments`不能被重新赋值
- `arguments`不会自动反映函数参数的变化
- 不能使用`arguments.callee`
- 不能使用`arguments.caller`
- 禁止`this`指向全局对象
- 不能使用`fn.caller`和`fn.arguments`获取函数调用的堆栈
- 增加了保留字（比如`protected`、`static`和`interface`）

45 attribute和property的区别是什么

- `attribute`是`dom`元素在文档中作为`html`标签拥有的属性；
- `property`就是`dom`元素在`js`中作为对象拥有的属性。
- 对于`html`的标准属性来说，`attribute`和`property`是同步的，是会自动更新的
- 但是对于自定义的属性来说，他们是不同步的

46 谈谈你对ES6的理解

- 新增模板字符串（为`JavaScript`提供了简单的字符串插值功能）
- 箭头函数
- `for-of`（用来遍历数据—例如数组中的值。）
- `arguments`对象可被不定参数和默认参数完美代替。
- `ES6`将p`romise`对象纳入规范，提供了原生的`Promise`对象。
- 增加了`let`和`const`命令，用来声明变量。
- 增加了块级作用域。
- `let`命令实际上就增加了块级作用域。

- 还有就是引入`module`模块的概念

47 ECMAScript6 怎么写class么

- 这个语法糖可以让有`OOP`基础的人更快上手`js`，至少是一个官方的实现了
- 但对熟悉`js`的人来说，这个东西没啥大影响；一个`Object.creat()`搞定继承，比`class`简洁清晰的多

48 什么是面向对象编程及面向过程编程，它们的异同和优缺点

- 面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了
- 面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为
- 面向对象是以功能来划分问题，而不是步骤

49 面向对象编程思想

- 基本思想是使用对象，类，继承，封装等基本概念来进行程序设计
- 优点
  - 易维护
    - 采用面向对象思想设计的结构，可读性高，由于继承的存在，即使改变需求，那么维护也只是在局部模块，所以维护起来是非常方便和较低成本的
  - 易扩展
  - 开发工作的重用性、继承性高，降低重复工作量。
  - 缩短了开发周期

50 对web标准、可用性、可访问性的理解

- 可用性（Usability）：产品是否容易上手，用户能否完成任务，效率如何，以及这过程中用户的主观感受可好，是从用户的角度来看产品的质量。可用性好意味着产品质量高，是企业的核心竞争力
- 可访问性（Accessibility）：Web内容对于残障用户的可阅读和可理解性
- 可维护性（Maintainability）：一般包含两个层次，一是当系统出现问题时，快速定位并解决问题的成本，成本低则可维护性好。二是代码是否容易被人理解，是否容易修改和增强功能。

51 如何通过JS判断一个数组

- instanceof方法
  - `instanceof` 运算符是用来测试一个对象是否在其原型链原型构造函数的属性

```
var arr = [];
arr instanceof Array; // true
```

- constructor方法
  - `constructor` 属性返回对创建此对象的数组函数的引用，就是返回对象相对应的构造函数

```
var arr = [];
arr.constructor == Array; //true
```

- 最简单的方法
  - 这种写法，是 `jQuery` 正在使用的

```
Object.prototype.toString.call(value) == '[object Array]'
// 利用这个方法，可以写一个返回数据类型的方法
var isType = function (obj) {
     return Object.prototype.toString.call(obj).slice(8,-1);
}
```

- `ES5` 新增方法 `isArray()`

```
var a = new Array(123);
var b = new Date();
console.log(Array.isArray(a)); //true
console.log(Array.isArray(b)); //false
```

52 谈一谈let与var的区别

- `let` 命令不存在变量提升，如果在 `let` 前使用，会导致报错
- 如果块区中存在 `let` 和 `const` 命令，就会形成封闭作用域
- 不允许重复声明，因此，不能在函数内部重新声明参数

53 map与forEach的区别

- `forEach`方法，是最基本的方法，就是遍历与循环，默认有3个传参：分别是遍历的数组内容`item`、数组索引`index`、和当前遍历数组`Array`
- `map`方法，基本用法与`forEach`一致，但是不同的，它会返回一个新的数组，所以在callback需要有`return`值，如果没有，会返回`undefined`

54 谈一谈你理解的函数式编程

- 简单说，"函数式编程"是一种"编程范式"（programming paradigm），也就是如何编写程序的方法论
- 它具有以下特性：闭包和高阶函数、惰性计算、递归、函数是"第一等公民"、只用"表达式"

55 谈一谈箭头函数与普通函数的区别?

- 函数体内的`this`对象，就是定义时所在的对象，而不是使用时所在的对象
- 不可以当作构造函数，也就是说，不可以使用`new`命令，否则会抛出一个错误
- 不可以使用`arguments`对象，该对象在函数体内不存在。如果要用，可以用`Rest`参数代替
- 不可以使用`yield`命令，因此箭头函数不能用作`Generator`函数

56 谈一谈函数中this的指向

- this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁，实际上this的最终指向的是那个调用它的对象
- 《javascript语言精髓》中大概概括了4种调用方式：
- 方法调用模式
- 函数调用模式
- 构造器调用模式

```
graph LR
A-->B
```

- apply/call调用模式

57 异步编程的实现方式

- 回调函数
  - 优点：简单、容易理解
  - 缺点：不利于维护，代码耦合高
- 事件监听(采用时间驱动模式，取决于某个事件是否发生)：

    - 优点：容易理解，可以绑定多个事件，每个事件可以指定多个回调函数
    - 缺点：事件驱动型，流程不够清晰
- 发布/订阅(观察者模式)
    - 类似于事件监听，但是可以通过‘消息中心’，了解现在有多少发布者，多少订阅者
- Promise对象
    - 优点：可以利用then方法，进行链式写法；可以书写错误时的回调函数；
    - 缺点：编写和理解，相对比较难
- Generator函数
    - 优点：函数体内外的数据交换、错误处理机制
    - 缺点：流程管理不方便
- async函数
    - 优点：内置执行器、更好的语义、更广的适用性、返回的是Promise、结构清晰。
    - 缺点：错误处理机制

58 对原生Javascript了解程度

- 数据类型、运算、对象、Function、继承、闭包、作用域、原型链、事件、`RegExp`、`JSON`、`Ajax`、`DOM`、`BOM`、内存泄漏、跨域、异步装载、模板引擎、前端`MVC`、路由、模块化、`Canvas`、`ECMAScript`

59 Js动画与CSS动画区别及相应实现

- ```
  CSS3
  ```

  的动画的优点
    - 在性能上会稍微好一些，浏览器会对`CSS3`的动画做一些优化
    - 代码相对简单
- 缺点
    - 在动画控制上不够灵活
    - 兼容性不好
- `JavaScript`的动画正好弥补了这两个缺点，控制能力很强，可以单帧的控制、变换，同时写得好完全可以兼容`IE6`，并且功能强大。对于一些复杂控制的动画，使用`javascript`会比较靠谱。而在实现一些小的交互动效的时候，就多考虑考虑`CSS`吧
  ```

60 JS 数组和对象的遍历方式，以及几种方式的比较

> 通常我们会用循环的方式来遍历数组。但是循环是 导致js 性能问题的原因之一。一般我们会采用下几种方式来进行数组的遍历

- `for in`循环
- `for`循环
- `forEach`
  - 这里的 `forEach`回调中两个参数分别为 `value`， `index`
  - `forEach` 无法遍历对象
  - IE不支持该方法； `Firefox` 和 `chrome` 支持
  - `forEach` 无法使用 `break`， `continue` 跳出循环，且使用 `return` 是跳过本次循环
- 这两种方法应该非常常见且使用很频繁。但实际上，这两种方法都存在性能问题
- 在方式一中，`for-in`需要分析出`array`的每个属性，这个操作性能开销很大。用在 `key` 已知的数组上是非常不划算的。所以尽量不要用`for-in`，除非你不清楚要处理哪些属性，例如 `JSON`对象这样的情况
- 在方式2中，循环每进行一次，就要检查一下数组长度。读取属性（数组长度）要比读局部变量慢，尤其是当 `array` 里存放的都是 `DOM` 元素，因为每次读取都会扫描一遍页面上的选择器相关元素，速度会大大降低

61 gulp是什么

- `gulp`是前端开发过程中一种基于流的代码构建工具，是自动化项目的构建利器；它不仅能对网站资源进行优化，而且在开发过程中很多重复的任务能够使用正确的工具自动完成
- Gulp的核心概念：流
- 流，简单来说就是建立在面向对象基础上的一种抽象的处理数据的工具。在流中，定义了一些处理数据的基本操作，如读取数据，写入数据等，程序员是对流进行所有操作的，而不用关心流的另一头数据的真正流向
- gulp正是通过流和代码优于配置的策略来尽量简化任务编写的工作
- Gulp的特点：
  - **易于使用**：通过代码优于配置的策略，gulp 让简单的任务简单，复杂的任务可管理
  - **构建快速** 利用 `Node.js` 流的威力，你可以快速构建项目并减少频繁的 `IO` 操作
  - **易于学习** 通过最少的 `API`，掌握 `gulp` 毫不费力，构建工作尽在掌握：如同一系列流管道

62 说一下Vue的双向绑定数据的原理

- `vue.js` 则是采用数据劫持结合发布者-订阅者模式的方式，通过 `Object.defineProperty()` 来劫持各个属性的 `setter`， `getter`，在数据变动时发布消息给订阅者，触发相应的监听回调

63 事件的各个阶段

- 1：捕获阶段 ---> 2：目标阶段 ---> 3：冒泡阶段
- `document` ---> `target` 目标 ----> `document`
- 由此，

  ```
  addEventListener
  ```

  的第三个参数设置为

  ```
  true
  ```

  和

  ```
  false
  ```

  的区别已经非常清晰了

  - `true` 表示该元素在事件的“捕获阶段”（由外往内传递时）响应事件
  - `false` 表示该元素在事件的“冒泡阶段”（由内向外传递时）响应事件

64 let var const

let

- 允许你声明一个作用域被限制在块级中的变量、语句或者表达式
- let绑定不受变量提升的约束，这意味着let声明不会被提升到当前
- 该变量处于从块开始到初始化处理的“暂存死区”

var

- 声明变量的作用域限制在其声明位置的上下文中，而非声明变量总是全局的
- 由于变量声明（以及其他声明）总是在任意代码执行之前处理的，所以在代码中的任意位置声明变量总是等效于在代码开头声明

const

- 声明创建一个值的只读引用 (即指针)

- 基本数据当值发生改变时，那么其对应的指针也将发生改变，故造成 `const` 申明基本数据类型时
- 再将其值改变时，将会造成报错， 例如 `const a = 3`；`a = 5` 时 将会报错
- 但是如果是复合类型时，如果只改变复合类型的其中某个 `Value` 项时， 将还是正常使用

65 快速的让一个数组乱序

```
var arr = [1,2,3,4,5,6,7,8,9,10];
arr.sort(function(){
    return Math.random() - 0.5;
})
console.log(arr);
```

66 如何渲染几万条数据并不卡住界面

> 这道题考察了如何在不卡住页面的情况下渲染数据，也就是说不能一次性将几万条都渲染出来，而应该一次渲染部分 `DOM`，那么就可以通过 `requestAnimationFrame` 来每 `16 ms` 刷新一次

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <ul>控件</ul>
  <script>
    setTimeout(() => {
      // 插入十万条数据
      const total = 100000
      // 一次插入 20 条，如果觉得性能不好就减少
      const once = 20
      // 渲染数据总共需要几次
      const loopCount = total / once
      let countOfRender = 0
      let ul = document.querySelector("ul");
      function add() {
        // 优化性能，插入不会造成回流
        const fragment = document.createDocumentFragment();
```

```
        for (let i = 0; i < once; i++) {
          const li = document.createElement("li");
          li.innerText = Math.floor(Math.random() * total);
          fragment.appendChild(li);
        }
        ul.appendChild(fragment);
        countOfRender += 1;
        loop();
      }
      function loop() {
        if (countOfRender < loopCount) {
          window.requestAnimationFrame(add);
        }
      }
      loop();
    }, 0);
  </script>
</body>
</html>
```

67 希望获取到页面中所有的checkbox怎么做?

> 不使用第三方框架

```
var domList = document.getElementsByTagName('input')
var checkBoxList = [];
var len = domList.length;    //缓存到局部变量
while (len--) {    //使用while的效率会比for循环更高
    if (domList[len].type == 'checkbox') {
        checkBoxList.push(domList[len]);
    }
}
```

68 怎样添加、移除、移动、复制、创建和查找节点

创建新节点

```
createDocumentFragment()    //创建一个DOM片段
createElement()   //创建一个具体的元素
createTextNode()   //创建一个文本节点
```

添加、移除、替换、插入

```
appendChild()      //添加
removeChild()      //移除
replaceChild()      //替换
insertBefore()      //插入
```

查找

```
getElementsByTagName()     //通过标签名称
getElementsByName()      //通过元素的Name属性的值
getElementById()         //通过元素Id，唯一性
```

69 正则表达式

> 正则表达式构造函数`var reg=new RegExp("xxx")`与正则表达字面量`var reg=//`有什么不同？匹配邮箱的正则表达式？

- 当使用`RegExp()`构造函数的时候，不仅需要转义引号（即`\`"表示"），并且还需要双反斜杠（即`\\`表示一个`\`）。使用正则表达字面量的效率更高

邮箱的正则匹配：

```
var regMail = /^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+((.[a-zA-Z0-9_-]{2,3}){1,2})$/;
```

70 Javascript中callee和caller的作用?

- `caller`是返回一个对函数的引用，该函数调用了当前函数；
- `callee`是返回正在被执行的`function`函数，也就是所指定的`function`对象的正文

> 那么问题来了？如果一对兔子每月生一对兔子；一对新生兔，从第二个月起就开始生兔子；假定每对兔子都是一雌一雄，试问一对兔子，第n个月能繁殖成多少对兔子？（使用`callee`完成）

```
var result=[];
  function fn(n){   //典型的斐波那契数列
     if(n==1){
          return 1;
     }else if(n==2){
             return 1;
     }else{
          if(result[n]){
                  return result[n];
          }else{
```

```
                //argument.callee()表示fn()
                result[n]=arguments.callee(n-1)+arguments.callee(n-2);
                return result[n];
            }
        }
    }
```