回到顶部

深克隆和浅克隆

浅克隆: 只是拷贝了基本类型的数据，而引用类型数据，复制后也是会发生引用，我们把这种拷贝叫做“（浅复制）浅拷贝”，换句话说，浅复制仅仅是指向被复制的内存地址，如果原地址中对象被改变了，那么浅复制出来的对象也会相应改变。

深克隆： 创建一个新对象，属性中引用的其他对象也会被克隆，不再指向原有对象地址。
JSON.parse、JSON.stringify()

- 回到顶部

es6的新特性都有哪些,箭头函数

let定义块级作用域变量 没有变量的提升，必须先声明后使用 let声明的变量，不能与前面的let，var，const声明的变量重名

const 定义只读变量 const声明变量的同时必须赋值，const声明的变量必须初始化，一旦初始化完毕就不允许修改 const声明变量也是一个块级作用域变量 const声明的变量没有“变量的提升”，必须先声明后使用 const声明的变量不能与前面的let， var ， const声明的变量重 const定义的对象\数组中的属性值可以修改,基础数据类型不可以

ES6可以给形参函数设置默认值

在数组之前加上三个点（...）展开运算符

数组的解构赋值、对象的解构赋值

箭头函数的特点

箭头函数相当于匿名函数，不能作为构造函数的，不能被new 箭头函数没有arguments实参集合,取而代之用...剩余运算符解决 箭头函数没有自己的this。他的this是继承当前上下文中的this 箭头函数没有函数原型 箭头函数不能当做Generator函数，不能使用yield关键字 不能使用call、apply、bind改变箭头函数中this指向 Set数据结构，数组去重

- 回到顶部

原型和原型链

所有的函数数据类型都天生自带一个prototype属性，该属性的属性值是一个对象 prototype的属性值中天生自带一个constructor属性，其constructor属性值指向当前原型所属的类 所有的对象数据类型，都天生自带一个*proto*属性，该属性的属性值指向当前实例所属类的原型

总结

把所有的对象共用的属性全部放在堆内存的一个对象（共用属性组成的对象），然后让每一个对象的**proto**存储这个「共用属性组成的对象」的地址。而这个共用属性就是原型，原型出现的目的就是为了减少不必要的内存消耗。而原型链就是对象通过**proto**向当前实例所属类的原型上查找属性或方法的机制，如果找到Object的原型上还是没有找到想要的属性或者是方法则查找结束，最终会返回undefined

- 回到顶部

`==`和`===`区别是什么

=赋值

==返回一个布尔值；相等返回true，不相等返回false； 允许不同数据类型之间的比较； 如果是不同类型的数据进行，会默认进行数据类型之间的转换； 如果是对象数据类型的比较，比较的是空间地址

=== 只要数据类型不一样，就返回false；

- 回到顶部

this指向

- 全局作用域下的this指向window
- 如果给元素的事件行为绑定函数，那么函数中的this指向当前被绑定的那个元素
- 函数中的this，要看函数执行前有没有 . ，有 . 的话，点前面是谁，this就指向谁，如果没有点，指向window
- 自执行函数中的this永远指向window
- 定时器中函数的this指向window
- 构造函数中的this指向当前的实例
- call、apply、bind可以改变函数的this指向
- 箭头函数中没有this，如果输出this，就会输出箭头函数定义时所在的作用域中的this

```
1. this关键字 上下文对象 ， this会随着调用方式不同发生变化
2. this指向不同调用方式产生的不同结果
3. new A => this指向实例对象
4. A() => this指向全局对象
5. o.a() => 对象方法调用this指向拥有该方法的对象
6. A.call() / A.apply() / A.bind()() 修改this指向
```


- 回到顶部

call bind apply 的区别

call() 和apply()的第一个参数相同，就是指定的对象。这个对象就是该函数的执行上下文。

call()和apply()的区别就在于，两者之间的参数。

call()在第一个参数之后的 后续所有参数就是传入该函数的值。

apply() 只有两个参数，第一个是对象，第二个是数组，这个数组就是该函数的参数。 bind() 方法和前两者不同在于： bind() 方法会返回执行上下文被改变的函数而不会立即执行，而前两者是 直接执行该函数。他的参数和call()相同。

- 回到顶部

js继承方式有哪些

原型链继承
new了一个空对象，这个空对象指向Animal并且Cat.prototype指向了这个空对象，这种就是基于原型链的继承。

- 特点：基于原型链，既是父类的实例，也是子类的实例。
- 缺点：1.无法实现多继承；2.所有新实例都会共享父类实例的属性。

```
function Cat(name) {
    this.name = name || 'tom'
}
Cat.prototype = new Animal()

var cat = new Cat()
cat.color.push('red')
cat.sleep() //tom正在睡觉!
cat.eat('fish') //tom正在吃：fish
console.log(cat.color) //["black", "red"]
console.log(cat instanceof Animal) //true
console.log(cat instanceof Cat) //true
var new_cat = new Cat()
console.log(new_cat.color) //["black", "red"]
```

构造继承

- 特点：可以实现多继承（call多个），解决了所有实例共享父类实例属性的问题。
- 缺点：1.只能继承父类实例的属性和方法；2.不能继承原型上的属性和方法。

```
function Dog(name) {
    Animal.call(this)
    this.name = name || 'mica'
}
var dog = new Dog()
dog.color.push('blue')
dog.sleep() // mica正在睡觉!
dog.eat('bone') //Uncaught TypeError: dog.eat is not a function
console.log(dog.color) //["black", "blue"]
console.log(dog instanceof Animal) //false
console.log(dog instanceof Dog) //true
var new_dog = new Dog()
console.log(new_dog.color) //["black"]
```

实例继承 核心：为父类实例添加新特性，作为子类实例返回

组合继承

- 特点：可以继承实例属性/方法，也可以继承原型属性/方法

- 缺点：调用了两次父类构造函数，生成了两份实例

```
function Mouse(name){
    Animal.call(this)
    this.name = name || 'jerry'
}
Mouse.prototype = new Animal()
Mouse.prototype.constructor = Mouse

var mouse = new Mouse()
mouse.color.push('yellow)
mouse.sleep() //jerry正在睡觉!
mouse.eat('carrot') //jerry正在吃：carrot
console.log(mouse instanceof Animal)//true
console.log(mouse instanceof Mouse)//true
var new_mouse = new Mouse()
console.log(new_mouse.color) //["black"]
```

拷贝继承
寄生组合继承 核心：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实 例方法/属性，避免的组合继承的缺点

- 回到顶部

闭包

闭包就是在函数里面声明函数

优点：在函数内部访问函数外部的变量，函数外部不可以访问函数内部的变量，保护变量不受外界污染。
缺点：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：

- 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
- 回到顶部

垃圾回收机制

含义：用来回收不可用的变量值所占用的内存空间
为什么？程序运行过程中会申请大量的内存空间，而对于一些无用的内存空间，如果不及时清理的话，会导致内存使用完（内存溢出），导致程序崩溃
方法：

- 标记清除法 当变量进入执行环境是，就标记这个变量为“进入环境”，当变量离开环境时，则将其标记为“离开环境”。垃圾收集器在运行的时候会给存储在内存中的所有变量都加上标记。然后，它会去掉环境中的变量以及被环境中的变量引用的标记。而在此之后再被加上标记的变量将被视为准备删除的变量，原因是环境中的变量已经无法访问到这些变量了。最后。垃圾收集器完成内存清除工作，销毁那些带标记的值，并回收他们所占用的内存空间。
- 引用计数法 跟踪记录每个值被引用的次数(会引起内存泄漏，不能解决循环引用的问题)

会造成内存泄漏
闭包
被遗忘的计时器或回调
循环引用 在引用计数策略下会导致内存泄漏，标记清除不会。 解决办法：手工解除循环引用。

- 回到顶部

var,let,const的区别

- var 声明的变量，其作用域为该语句所在的函数内，且存在变量提升现象
- let 声明的变量，其作用域为该语句所在的代码块内，不存在变量提升
- const声明的变量不允许修改
- 回到顶部

数据类型 typeof instanceof Object.prototype.toString.call()

number string boolean symbol null undefined以及object

typeof

typeof(null); object
typeof(undefined); undefined
能够快速区分基本数据类型 缺点：不能将Object、Array和Null区分，都返回object

instanceof

能够区分Array、Object和Function，适合用于判断自定义的类实例对象 缺点：Number，Boolean，String基本数据类型不能判断

Object.prototype.toString.call()

括号里面传入要判断的参数，判断Array，Function，Null各种类型都可以判断。

null 和 undefined的区别：
undefined类型只有一个值，就是undefined

- 变量声明没有赋值是undefined
- 调用函数时，应该提供的参数没有提供，该参数等于undefined。
- 对象没有赋值的属性，该属性的值为undefined。
- 函数没有返回值时，默认返回undefined。

Null类型也只有一个值，即null。null用来表示尚未存在的对象，常用来表示函数企图返回一个不存在的对象。用法

- 作为函数的参数，表示该函数的参数不是对象。
- 作为对象原型链的终点。
- 回到顶部

js作用域

JS中的作用域分为两种：全局作用域和函数作用域。函数作用域中定义的变量，只能在函数中调用，外界无法访问。没有块级作用域导致了if或for这样的逻辑语句中定义的变量可以被外界访问，因此ES6中新增了let和const命令来进行块级作用域的声明。

- 回到顶部

cookie,sessionStorage和localStorage

- cookie用来保存登录信息，大小限制为4KB左右
- localStorage是Html5新增的，用于本地数据存储，保存的数据没有过期时间，一般浏览器大小限制在5MB
- sessionStorage接口方法和localStorage类似，但保存的数据的只会在当前会话中保存下来，页面关闭后会被清空。

名称	生命期	大小限制	与服务期通信	
cookie	一般由服务器生成，可设置失效时间。如果在浏览器端生成Cookie，默认是关闭浏览器后失效	4KB	每次都会携带在HTTP头中，如果使用cookie保存过多数据会带来性能问题	
localStorage	除非被清除，否则永久保存	5KB	仅在浏览器中保存，不与服务器通信	
sessionStorage	仅在当前会话下有效，关闭页面或浏览器后被清除	5KB	仅在浏览器中保存，不与服务器通信	

- 回到顶部

常见的设计模式有哪些?

1、js工厂模式
2、js构造函数模式
3、js原型模式
4、构造函数+原型的js混合模式
5、构造函数+原型的动态原型模式
6、观察者模式
7、发布订阅模式

- 回到顶部

js中跨域方法

```
同源策略（协议+端口号+域名要相同）
```

1、jsonp跨域(只能解决get) 原理：动态创建一个script标签。利用script标签的src属性不受同源策略限制，因为所有的src属性和href属性都不受同源策略的限制，可以请求第三方服务器资源内容

```
1.去创建一个script标签
2.script的src属性设置接口地址
3.接口参数，必须要带一个自定义函数名，要不然后台无法返回数据
4.通过定义函数名去接受返回的数据

由于浏览器的同源策略限制，不允许跨域请求；但是页面中的 script、img、iframe标签是例外，不受同源策略限制。
Jsonp 就是利用script标签跨域特性进行请求。
JSONP 的原理就是，先在全局注册一个回调函数，
定义回调数据的处理；与服务端约定好一个同名回调函数名，
服务端接收到请求后，将返回一段 Javascript，在这段 Javascript
代码中调用了约定好的回调函数，并且将数据作为参数进行传递。
当网页接收到这段 Javascript 代码后，就会执行这个回调函数。
JSONP缺点：它只支持GET请求，而不支持POST请求等其他类型的HTTP请求。
```

2、document.domain 基础域名相同 子域名不同

3、window.name 利用在一个浏览器窗口内，载入所有的域名都是共享一个window.name

4、服务器设置对CORS的支持 原理：服务器设置Access-Control-Allow-Origin HTTP响应头之后，浏览器将会允许跨域请求

5、利用h5新特性window.postMessage()

6、proxy

- 回到顶部

前端有哪些页面优化方法

- 减少 HTTP请求数
- 从设计实现层面简化页面
- 合理设置 HTTP缓存
- 资源合并与压缩
- 合并 CSS图片，减少请求数的又一个好办法。
- 将外部脚本置底（将脚本内容在* 页面信息内容加载后再加载）
- 多图片网页使用图片懒加载。
- 在js中尽量减少闭包的使用
- 尽量合并css和js文件
- 尽量使用字体图标或者SVG图标，来代替传统的PNG等格式的图片
- 减少对DOM的操作
- 在JS中避免“嵌套循环”和 “死循环”
- 尽可能使用事件委托（事件代理）来处理事件绑定的操作
- 回到顶部

Ajax的四个步骤

1.创建ajax实例

2.执行open 确定要访问的链接 以及同步异步

3.监听请求状态

4.发送请求

- 回到顶部

ajax的状态码/http 状态码

1开头 表示客户端应该继续发送请求

2开头 成功

- 200 : 代表请求成功;

3开头 重定向

- 301 : 永久重定向;
- 302: 临时转移
- 304 : 读取缓存 [表示浏览器端有缓存，并且服务端未更新，不再向服务端请求资源]
- 307:临时重定向

以4开头的都是客户端的问题;

- 400 :数据/格式错误
- 401: 权限不够； （身份不合格，访问网站的时候，登录和不登录是不一样的）
- 404 : 路径错误，找不到文件

以5开头都是服务端的问题

- 500 : 服务器的问题
- 503: 超负荷;
- 回到顶部

JS中常见的异步任务

定时器、ajax、事件绑定、回调函数、async await、promise

- 回到顶部

map和forEach的区别

相同点

都是循环遍历数组中的每一项 forEach和map方法里每次执行匿名函数都支持3个参数，参数分别是item（当前每一项）、index（索引值）、arr（原数组），需要用哪个的时候就写哪个 匿名函数中的this都是指向window 只能遍历数组

不同点

map方法返回一个新的数组，数组中的元素为原始数组调用函数处理后的值。(原数组进行处理之后对应的一个新的数组。) map()方法不会改变原始数组 map()方法不会对空数组进行检测 forEach()方法用于调用数组的每个元素，将元素传给回调函数.(没有return，返回值是undefined)

注意：forEach对于空数组是不会调用回调函数的。

- 回到顶部

数组、字符串的方法

数组

```
1.push(); 方法：在数组的最后面添加内容，返回值是添加后数组的长度
2.pop() 方法：把数组的最后一个删除，返回值是删除的那一项
3.unshift() 方法：在数组的最前面添加内容，返回值是添加后数组的长度
4.shift() 方法：删除数组中的第一项，返回值是删除的那一项
5.concat() 方法：拼接数组
6.join() 方法：把数组中的每一项用特定的字符串连接起来
7.slice(a,b) 方法： 从索引a开始截取（包括a），一直截取到b（不包括b），如果里面只有一个参数，，那么就直接截取到末尾，不会改变原来的数组
8.splice(a,b,c); 方法：从索引a开始截取b个元素，并用c替换截取的元素，并改变原来的数组，如果只有一个参数，表示从这个索引开始截取到末尾
splice(a,b) 方法里面如果第一个参数为负数的话，使用方法跟slice() 方法一样，如果第二个参数为负数的话，表示截取的是一个空数组
9.sort(function (a,b){return a-b}); 数组排序方法
10.reverse(); 方法：数组翻转方法
11.forEach(function (ele,index){console.log(ele,index)}); 方法：遍历数组，传入一个回调函数，里面有三个参数，第一个是元素值，第二个是索引，第三个是数组
12.every(function (ele){return ele > 20}); 传入一个回调函数，返回值是boolean类型值
13.map(function (ele){return ele + 10}); 或 map(Math.sprt); 让数组中的每一个元素按照函数的方法去执行，返回一个新的数组
14.filter(functoon (ele){return ele < 50}); 起到过滤作用， 让数组中的每一个元素按照函数的方法去执行，把符合条件的元素放到一个新的数组中返回
15.eval(arr.join('+')); 如果数组中的每一项都是数字的话，用这个方法可以求数组中的数字之和
```

字符串

```
1.charAt();方法： 找到索引对应的字符串
2.charCodeAt();方法 ：返回索引对应的字符串的ASCII码
3.string.fromCharCode();方法 ：获取ASCII码对应的字符串
4.concat();方法：拼接字符串
5.slice(a,b);方法： 从索引a开始截取（包括a），一直截取到索引b（不包括b），如果里面只有一个参数，就直接截取到末尾
slice(a,b); 如果里面的参数是负数的话，使用方法跟数组中的slice()方法一样
6.substring(a,b)方法：从索引a开始截取（包括a），一直截取到索引b（不包括b），如果里面只有一个参数那么就直接截取到末尾
（1）substring(a,b); 如果第一个参数为负数的话，会自动转化为0，从索引0开始截取
（2）substring(a,b); 如果第二个参数为负数的话，会自动转化为0，并且会把第一个参数和第二个参数的位置互换
7.substr(a,c); 方法： 从索引a开始截取c个元素
substr(a,b); 如果第一个参数为负数的话，跟数组中 slice()使用方法一样
substr(a,b); 如果第二个参数为负数，或者两个参数都是负数的话，截取到的是空字符串
8.indexOf();方法：从前往后查找字符串对应的索引
9.lastIndexOf();方法：从后往前查找字符串对应的索引
10.replace();方法 ：替换字符串
11.trim();方法：把字符串两边的空格去掉
12.toLowerCase();方法：转小写
13.toUpperCase(); 方法：转大写
14.toLocaleCompare();方法：比较
15.search();方法：返回字符串对应的索引
16.match();方法：返回找到的字符串，没有就返回null
17.split();方法：用指定的字符串隔开，并以数组的方式返回
```


- 回到顶部

for...of和for...in的区别

for in

- 一般用于遍历对象的可枚举属性。以及对象从构造函数原型中继承的属性。对于每个不同的属性，语句都会被执行。
- 不建议使用for in 遍历数组，因为输出的顺序是不固定的。
- 如果迭代的对象的变量值是null或者undefined, for in不执行循环体，建议在使用for in循环之前，先检查该对象的值是不是null或者undefined

for of

for...of 语句在可迭代对象（包括 Array，Map，Set，String，TypedArray，arguments 对象等等）上创建一个迭代循环，调用自定义迭代钩子，并为每个不同属性的值执行语句

- 回到顶部