## Js事件委托

把一个或一组元素的响应事件的函数委托到它的父层或更外层元素上。真正绑定事件的是外层元素，当事件响应到目标元素时，会通过事件冒泡机制触发外层元素的绑定事件。

优点：

减少内存消耗

假如一个ul下100个li都需要绑定事件，如果写100个addEventListen，内存消耗很大，可以将这个点击事件绑定到父层，在执行事件的时候，再去判断目标元素是哪个li。

事件冒泡：当一个元素接收到事件的时候，会把它接受到的事件传给自己的父级元素，一直到window。

取消事件冒泡

function stopBubble(e) {

//如果提供了事件对象，则这是一个非IE浏览器

if ( e && e.stopPropagation )

//因此它支持W3C的stopPropagation()方法

e.stopPropagation();

else

//否则，我们需要使用IE的方式来取消事件冒泡

window.event.cancelBubble = true;

}

## Js作用域和作用域链

Js作用域可以理解为变量、对象、函数的一个集合，作用域就是一段代码的执行环境，

作用域分为局部作用域和全局作用域，在函数内定义的变量为局部作用域，在函数外定义的变量为全局作用域。

当执行一个函数的时候，会先生成该函数的局部作用域，此时会形成一个作用域链，通过该函数的局部作用域可以向上找到全局作用域，局部作用域可以通过作用域链访问全局作用域变量，反之不行。

销毁，局部作用域在函数执行完后会销毁（不考虑闭包情况），全局作用域会在页面关闭后销毁

## Js原型和原型链

原型是为了解决构造函数所生成的实例无法共享属性的问题。

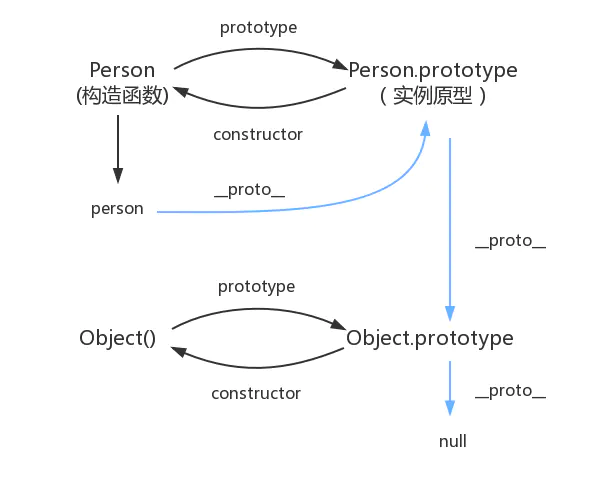
每个函数都会有一个prototype属性，这个对象就是我们说的原型对象。

每个对象都有\_\_proto\_\_属性，这个属性指向该对象的原型。

每个原型对象都有一个constructor属性指向关联的构造函数

Function Person(){}

Person == Person.prototype.constructor

在js内部，每一个对象只有指向它原型对象的内部链接，这个原型对象又有自己的原型，直到某个对象的原型为null为止，这样一层一层的链结构又称为原型链。  


## Js防抖和节流

防抖:对于短时间内连续触发的事件 如window.onscroll = function(){}，防抖的含义就是让某个时间内，指定的事件处理函数只执行一次

example

**function** **debounce**(fn,delay){

**let** timer = null *//借助闭包*

**return** **function**() {

**if**(timer){

clearTimeout(timer)

}

timer = setTimeout(fn,delay)

}

}

节流:使用防抖方案来处理问题的结果是，如果有用户在限制时间内，不停的触发滚动事件，那么回调函数就永远无法执行，节流的效果是如果短时间内大量触发同一事件，那么在函数执行一次之后，该函数在指定的时间内就不再工作，直到过了这段时间才重新生效。

example

**function** **throttle**(fn,delay){

**let** valid = true

**return** **function**() {

**if**(!valid){

*//休息时间 暂不接客*

**return** false

}

*// 工作时间，执行函数并且在间隔期内把状态位设为无效*

valid = false

setTimeout(() => {

fn()

valid = true;

}, delay)

}

}

## Es基础

页可见区域宽： document.body.clientWidth;

网页可见区域高： document.body.clientHeight;

网页可见区域宽： document.body.offsetWidth (包括边线的宽);

网页可见区域高： document.body.offsetHeight (包括边线的宽);

网页正文全文宽： document.body.scrollWidth;

网页正文全文高： document.body.scrollHeight;

网页被卷去的高： document.body.scrollTop;

网页被卷去的左： document.body.scrollLeft;

网页正文部分上： window.screenTop;

网页正文部分左： window.screenLeft;

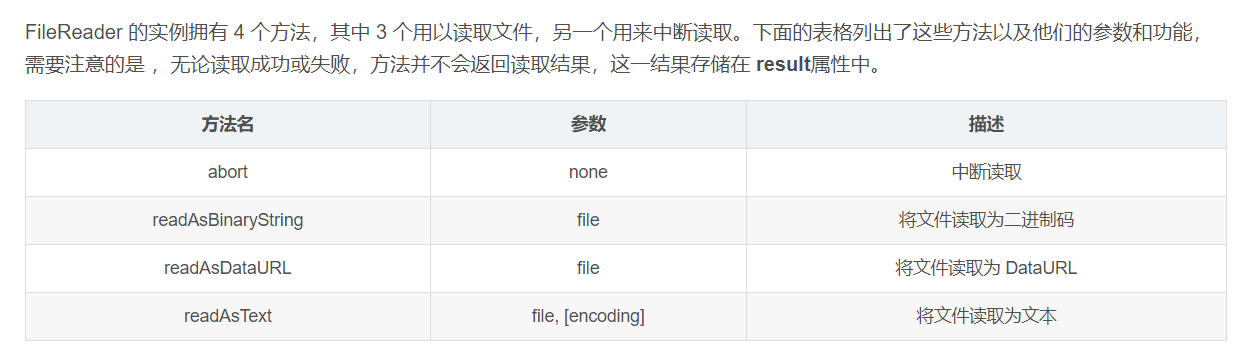
屏幕分辨率的高： window.screen.height;

屏幕分辨率的宽： window.screen.width;

屏幕可用工作区高度： window.screen.availHeight;

FileReader

FileReader是用于操作文件的api，该api不能像后台语言操作文件的api那样通过路径直接操作硬盘中的文件，而是要配合type=file属性的input标签绑定onChange事件的回调函数，通过回调函数中的file对象才能进行对该文件的操作。





URL.createObjectURL(blob) 和FileReader.readAsDataURL(file)

通过FileReader.readAsDataURL(file)可以获取一个文件的dataUrl，异步执行，如果要获取一张图片的base64，则使用这个

通过URL.createObjectURL(blob)可以获取当前文件的一个内存url，通过执行，可以节省性能并更快速，只不过需要在不使用的情况下手动释放内存。

两个方法都可以用于浏览器上预览本地图片或视频。createObjectURL微信小程序不支持，如果需要兼容微信小程序，则使用FileReader。

es6 Set

Set是es6新推出来的一种数据结构，是值得集合，类似数组， typeof返回的是object，特点是 有效，并且值不会重复，一般可以配合 Array.from 进行数组去重，拥有size属性，

.clear清除所有值

Proxy属于一种元编程，就是对编程语言进行编程，proxy在目标对象之前添加一层拦截，外界对于目标对象的访问，都要先经过这层拦截，通过这个机制，可以对外界的访问进行过滤和改写。Proxy内部一共有13中拦截操作，get、set、deleteProperty等。

Es6 Map

Map是es6新推出来的一种数据结构，有序，拥有size属性，频繁增删键值对更适合

使用规范语言 用Map.set 去添加键值对，Map对象之间可以合并，但是会保持键的唯一性，也可以和数组进行合并,Map的键可以为任意类型， fun、obj、string、int、undefined

扩展运算符可以将实现了iterator接口的数据结构转换为数组，Array.from可以将实现了iterator接口的数据结构和类数组（即有length属性的对象）转换为数组

Export 和export default的区别

1.export与export default均可用于导出常量、函数、文件、模块等；

2.你可以在其它文件或模块中通过import+(常量 | 函数 | 文件 | 模块)名的方式，将其导入，以便能够对其进行使用；

3.在一个文件或模块中，export、import可以有多个，export default仅有一个；

4.通过export方式导出，在导入时要加{ }，export default则不需要。

**Object.getPrototypeOf()** 方法返回指定对象的原型（内部[[Prototype]]属性的值）

插入100个节点如何优化

1. 在进行for循环时缓存length，避免每次循环都要进行计算length属性
2. document.createDocumentFragment()创建文档片段，循环时将li添加到这里，此时，该文档片段还没有插入到页面中，等待for循环结束，将该文档片段插入至页面中，该方法适合在一次性插入节点时使用，如频繁插入则不适合。

Object.defineProperty方法会直接在一个对象上定义一个新属性，或修改一个对象上已有的属性，并返回该对象（Object, name, descriptor(描述)）

value：属性的值

writable：值是否可以重写 Boolean

enumerable：属性是否可以被枚举 Boolean

configurable：目标属性是否可以被删除或是否可以再次修改特性 Boolean

set：设置属性值的方法

get：获取属性值时调的方法

**hasOwnProperty()** 方法会返回一个布尔值，指示对象自身属性中是否具有指定的属性（也就是，是否有指定的键，不包含原型值）

Object.assign方法用于对象的合并，将源对象(source)的所有可枚举属性，复制到目标对象(target) 并返回该对象 Object.assign(target,source1,source2)

1.当源对象中只有一级属性，没有二级属性，此方法为深拷贝，当方法中有方法时，从二级属性以后就是浅拷贝

深度克隆方法

1.Object.assign

function deepClone(source){

let target = undefined;

if(typeof source == "object"){

target = Array.isArray(target) ? [] : {};

for(let k in source){

if(source.hasOwnProperty(k)){

if(typeof source[k] == "object"){

target[k] = deepClone(source[k])

}else{

target[k] = source[k]

}

}

}

}else{

target = source

}

return target;

}

2.JSON.stringfy和JSON.parse  
用JSON.stringfy把对象转换成字符串，再用JSON.parse把字符串转换成新的对象

可以转成JSON格式的对象才能使用这种方法，如果对象中包含function和RegExp这些就不能用这种方法了

function deepClone(obj) {

let \_obj = JSON.stringify(obj);

let objClone = JSON.parse(\_obj);

return objClone;

}

Symbol是es6中新引入的一种数据类型，表示独一无二的值，最大的用法就是用来定义对象中的唯一属性名。

Es6的三个点有两个名字 rest参数和扩展运算符

当用在函数定义时的形参前面时，称为rest参数，用于接收不确定的参数

当与解构赋值组合使用时，称为rest参数，用于接收剩余的值，存在数组中

当用在字符串或数组前面时，称为扩展运算符，将数组或字符串进行拆解

展开运算符对于实现了iterator接口的对象可以转成真正的数组。

[Object.assign()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/assign) 方法只能拷贝源对象的可枚举的自身属性，同时拷贝时无法拷贝属性的特性们，而且访问器属性会被转换成数据属性，也无法拷贝源对象的原型，该方法配合 [Object.create()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/create) 方法可以实现上面说的这些。

Object.create(

Object.getPrototypeOf(obj),

Object.getOwnPropertyDescriptors(obj)

)

Object.assign无法完全拷贝源对象属性，但是可以拷贝目标对象所有属性及特性

Object.create用于创建一个新的对象，第一个参数表示新创建对象的原型，第二个参数将会放在新创建对象的实例上，并且上述这种浅拷贝方法还可以正确拷贝get和set方法。如果使用Object.assign(Object.create(Object.getPrototypeOf(obj)),obj) 将会无法正确拷贝源对象的get和set方法

## Asnyc await

async 函数是什么？一句话，它就是 Generator 函数的语法糖

async函数返回的是一个promise对象，如果在函数中return一个直接量，async会把这个直接量通过Promise.resolve()封装成promise对象

await表达式只能放在async函数中使用，await等待的是一个表达式，这个表达式的计算结果可以是promise对象或者其它值，通常用于等待一个async函数的返回值promise对象，

当await表达式等待promise对象时，它会阻塞当前async函数后面的代码，等待promise对象的resolve，然后得到resolve值，作为await表达式的运算结果

Async/await的优势在于处理then链，使得异步代码写起来与同步代码一样，可读性更高

单一的 Promise 链并不能发现 async/await 的优势，但是，如果需要处理由多个 Promise 组成的 then 链的时候，优势就能体现出来了（很有意思，Promise 通过 then 链来解决多层回调的问题，现在又用 async/await 来进一步优化它）

await 命令后面的 Promise 对象，运行结果可能是 rejected，所以最好把 await 命令放在 try...catch 代码块中。

## Class

Js语言中，生成实例对象的方法是通过构造函数，这种写法与传统的面向对象编程语言差异很大，es6为了更接近传统语言的写法，引入了class累的概念，作为对象的模板，通过class关键字，可以定义类，新的class写法只是让对象原型的写法更加清晰，更像面向对象的编程语法，class中有一个constructor构造方法，在生成实例的时候会调用这个构造函数

类可以通过extends关键字进行继承，子类在继承父类时，必须使用super函数在constructor方法中调用父类的constructor方法，super关键字出现在constructor方法中只能用在做函数调用，代表父类的构造方法，出现在子类方法中，当做对象使用，在子类普通函数中，super代表父类的原型，在静态方法中指向父类，

在子类的静态方法中通过super调用父类方法时，方法内部的this指向当前的子类，普通函数中，this指向当前子类实例

## Reduce

reduce 为数组中的每一个元素依次执行回调函数，不包括数组中被删除或从未被赋值的元素，接受四个参数：初始值（或者上一次回调函数的返回值），当前元素值，当前索引，调用 reduce 的数组。

Reduce可以设置初始值 arr.reduce(callback(prev, cur, index, arr), initValue)

用法

将多维数组转化为一维

let arr = [[0, 1], [2, 3], [4,[5,6,7]]]

const newArr = function(arr){

return arr.reduce((pre,cur)=>pre.concat(Array.isArray(cur)?newArr(cur):cur),[])

}

console.log(newArr(arr)); //[0, 1, 2, 3, 4, 5, 6, 7]

数组去重的方法

1. [...new Set(arr)]

2. Array.from(new Set(arr))

3. arr.reduce((prev, cur)=>{ if(prev.includes(cur)) {return prev} else {return prev.concat(cur)} },[])

4. indexOf 去重

var **array** = [];

**for** (var i = 0; i < arr.length; i++) {

**if** (**array** .indexOf(arr[i]) === -1) {

**array** .push(arr[i])

}

}