**iframe内敛框架：**iframe框架加载完后才触发的onload事件，可用js动态设置iframe的src值来避免阻塞，但是并不经常使用

1．获取子窗口：

iframe.contentWindow 该iframe为iframe标签的引用

简易写法：window.frames[“iframe的name值”]

IE专用：document.iframess[name].contentWindow

Document.iframes[i].contentWindow

2．获取父窗口：

window.parent

上述的方法前提是子窗口和父窗口必须是同源的

**hash值：**解决不同源的问题

在src后面加上#123，怎称其为hash值，因为在子窗口中可以通过location.hash()来访问这个值，在通过slice(1)来获取hash值#后面的值

**标签相关：**

<label> for属性for(input的id值)，将<label>与input绑定在一起，两者关系：<label>属于input但input不属于<label>

图片预加载和懒加载：

预加载：先将img标签插入DOM树种，在异步加载img图片

懒加载：当前元素在屏幕中显示时才加载图片，可动态添加图片来实现

Math.random()产生随机数，Math.floor()向下取整，Math.ceil()向上取整

文档碎片：可以存储一段dom结构，并将其一并插入DOM树中，这样randertree只要渲染一遍,创建的文档碎片可以当作元素节点但是不占用DOM结构

创建: document.createElementFragment();

String.indexOf(“str”, index):表示string中是否有str字符串，index规定从哪位开始搜索，如果有怎返回找到的位置的索引值，否则返回-1

**数组：**

ES6新增的方法：includes，find，findindex, keys, values, entries, fill, Array.of, copyWithin, from

Array.prototype.forEach(function (ele, index) {}): 循环遍历数组中的每个元素，不改变原数组

Array.prototype.map( callback(value, index, array ) ): 参数分别为值，索引，以及数组本身，map遍历每个元素并将回调函数的返回值构成一个**新数组返回**(新数组和元素组等长，即使返回undefined，数组上也会显示)，由于字符串和数组有相似性，所以string可以借用数组方法

Array.prototype.filter(function (ele, index) {}): 循环遍历数组每个元素，并将function()返回true的元素保存，返回false的删除，并**返回新数组**，不改变原数组

Array.prototype.some(callback(item, index){}): 回调函数返回值只要有true结果就为true

Array.prototype.every(callback(item, index){}): 回调函数返回值全部为true结果才为true

Array.prototype.reduce(function (previous, ele, index) {}, pre) //previous代表上一次函数执行后的返回值,后面的pre参数可有可无，如果有表示从第一位的previous的值，没有则表示从第二位开始，previous表示第一位数组元素

reduceRight()是让数组从右边开始执行

Array.prototype.splice(start, count, …arr)裁剪数组的从start到start+count，并将…arr加入start位置，**会改变原数组**

Array.prototype.slice(start, end)，获取从start到end位置的元素，返回新数组，**不改变原数组**

Array.prototype.copyWithin(target, start, end)，将数组start到end位置的元素复制到target位置，改变原数组

Array.prototype.fill(value, start, end)，将数组start到end位置都填充为value，此方法为数组提供一个很好的初始化方式

push从数组尾部插入, pop从数组尾部删除, shift从数组头部删除, unshift从数组头部插入

reserve将数组倒叙，sort(function () )按照函数参数排列对比方式排列，sort方法按照冒泡排序方式进行遍历排序

返回从头开始第一个符合条件的元素 Array.prototype.find(callback())

返回从头开始第一个符合条件的元素的索引号 findIndex()

返回从头开始第一个特定元素的索引号 Array.indexOf(arr, item, startindex)

返回从尾开始第一个特定元素的索引号 lastIndexOf

Array.prototype.Includes(value)返回是否含有该元素

Array.prototype.fill(value, startindex, endindex): 将start到end用value填充

Array.prototype.concat(arr): 数组arr与原数组相结合，返回新数组，可以同时填写多个数组

Array.prototype.keys(): 返回数组键值的迭代器，values() 返回值的迭代器，entries()返回键值和值的迭代器(每个都是数组[key, value])

Array.of(…arr): 用于创建包含参数的数组，与new Array()相比好处是可以直接创建一个数字的数组(new Array(m)是创建一个length为m的空数组, 每一位都是empty)

Array.from()：讲类数组转化为数组(常见类数组：arguments，nodeList)

**数组返回值问题：**

不会改变原数组的方法(即返回新数组)：slice, concat, map, filter, forEach

会改变原数组的方法：pop, push, shift, unshift, splice, sort

**对象：**

1. **属性：用于对象属性的设置**

Object.prototype.writable: 是否可改写，若属性是一个引用值，即使设置了writable属性，仍然可以改变其引用的对象的属性值，设置该属性的变量无法改变属性值，但是不会报错

Object.prototype.enumerable: 是否可枚举

Object.prototype.configurable: 是否科配置，设置该属性后，value值只能通过直接复制改变，不能通过defineProperty改变（会报错Type error）

Object.prototype.constructor: 用于创建一个对象的原型

1. **常用方法：**

Object.prototype.hasOwnProperty(value)：表示对象本身是否包含某个属性，不包含原型上

Object.prototype.isPrototypeOf(obj): 表示对象是否在对象的原型链上

Object.prototype.propertyIsEnumerable(value): 指定属性是否可枚举，for in 循环枚举对象属性时，会自动调用该方法

Object.prototype.toString(): 返回对象的基本数据类型

Object.prototype.watch(value, callback): 给对象属性增加监听，unwatch(): 移除对象属性的某个监听，不建议使用，因为目前只有火狐浏览器支持该功能

1. **Object构造函数上的方法：**

Object.assign(target, …sources): 将任意多的元对象自身的可枚举属性拷贝(浅拷贝)到目标属性，然后返回目标对象

Object.create(obj): 建立一个新对象，并将新对象的\_\_proto\_\_指向obj

Object.defineProperty(obj, prop, {config}): 给对象添加一个新属性，或者修改原有的属性值，

Object.defineProperties(obj, { prop1:{configure}, prop2:{configure} })

Object.entries(obj): 按顺序将对象的属性及值以数组(第0位是属性，第一位是值)形式输出，每个属性都是一个一维数组，最终返回一个二维数组

Object.seal(obj): 密封属性，不可更改配置，不能添加属性

Object.freeze(obj): 冻结对象，即对象不能添加属性，不能修改属性，不能删除属性，不能修改属性的配置，并返回这个冻结的对象

Object.getOwnPropertyDescriptor(obj, prop): 返回对象的指定属性的配置信息

Object.getOwnPropertyNames(obj): 返回一个由对象属性名构成的数组

Object.getOwnPrototypeOf(obj): 返回对象的原型

Object.is()与恒等于相似，都用与判断两值是否相等，但是is可以判断NaN与NaN相等，并且可以判断+0和-0不想等

Object.isExtensible(obj): 判断一个对象是否可以添加属性

Object.isFreeze(obj): 判断一个对象是否被冻结

Object.isSealed(obj): 判断一个对象是否是密封的

Object.preventExtensions(obj): 让一个对象变得不可拓展

Object.setPrototypeOf(obj, prototype): 将指定对象的原型设置为对象或者null

Object.keys(), Object.values() , Object.entries()与数组方法不同的是，方法会返回一个数组，也是迭代器对象，可以通过for of循环遍历

**迭代器：**Iterato

**String:** 字符串方法：

String.prototype.startWith(str): 数组是否以str开头

String.prototype.endWith(str): 数组是否以str结尾

String.prototype.includes(str1, str2, …): 是否包含这些字符串

**模板字符串：**

语义化更好

防止注入XSS：普通字符串中的script标签会执行

更标准的字符串，更好的处理字符串拼接问题(支持直接换行，${}里面可以进行js代码的书写，会自动执行)

深度克隆：

function deepClone (target, option) {

var src, copy;

if(option){

for(var prop in option){

src = target[prop];

copy = option[prop];

if(copy && typeof copy == "object"){

if(Object.prototype.toString.call(copy) == "[object Array]"){

src = src ? src : [];

}else{

src = src ? src : {};

}

target[prop] = deepClone(src, copy);

}else{

target[prop] = copy;

}

}

}

return target;

}

ajax:

1. 浏览器
2. Ajax对象
3. Ajax，open(“GET/POST”, url, true)
4. Ajax.send()
5. onreadystatechange == 4
6. status == 200 404(信息错误) …

**创建ajax对象：**

Get请求：信息添加在php文件名后面，’.php?userName = name’

Post请求：信息添加在send函数中，宁外，post方法要设置一个请求头，

Xhr.setRequestHeader(‘Content-type’, ‘application/x-www-form-urlcoded’)

function ajaxFunc (method, url, data, callback) {

var xhr = null; //构建ajax对象

if(window.XMLHttpRequest){

xhr = new XMLHttpRequest();

}else{

xhr = ActiveXObject(‘Microsoft.XMLHttp’);

}

if(method == “GET”){

xhr.open(method, url, true);

xhr.send();

}else if(method == “post”){

xhr.open(method , url , true);

xhr.setRequsetHeader(“Content-type”, “application/x-www-form-urlcoded”);

xhr.send();

}

onreadystatechange = function () {

if(xhr.readyState == 4 && xhr.status == 200){

callback(xhr.responseText);

}

}

}

readyState的四个状态：

0 - (未初始化)还没有调用send()方法

1 - (载入)已调用send()方法，正在发送请求

2 - (载入完成)send()方法执行完成，

3 - (交互)正在解析响应内容

4 - (完成)响应内容解析完成，可以在客户端调用了

同原协议：只有当网址的协议，域名，端口都相同的情况可以获取数据库的数据

顶级域名分类：

.com：商业机构

.net：从事互联网服务的机构

.org：非盈利性组织

.com.cn：国内商业机构

.net.cn：国内从事互联网服务的机构

.org.cn：国内非营利组织机构

DNS(域名服务器)，用来将网址解析成ip地址，解析的过程是从后往前解析

以前www用于表示是网页服务，但是现在服务器已经可以识别页面的类型，所以已经不需要再加www这个域名了，国外已经有公司不使用www，而中国很多公司仍采用www一方面是为了尊重用户使用习惯，另一方面www用于绑定c记录，由于DNS只能绑定a记录（a记录保存网址的ip地址），c记录是网址的别名，还有就是www时，cookie的作用域更大

TCP协议，基于建立连接来传输数据，

UDP协议，不必建立连接，只要知道ip地址就直接发送数据包，不管是否接受到等因素，相对于UDP，TCP更加的安全，但是由于要建立连接，速度会比较慢，而UDP安全性不高，容易丢失数据，但是实时性很高，常用于一些实时性的，不怕数据丢失的工作

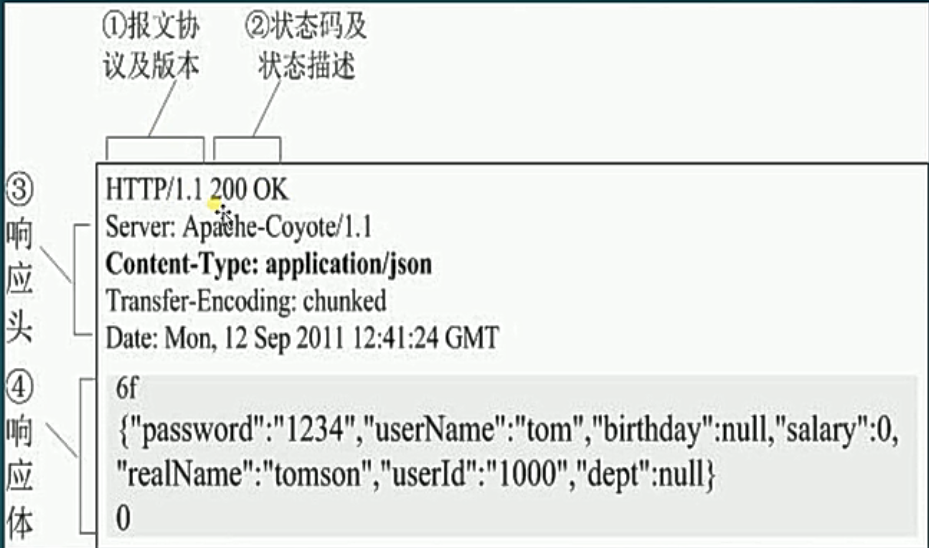
客户端与服务器建立连接：

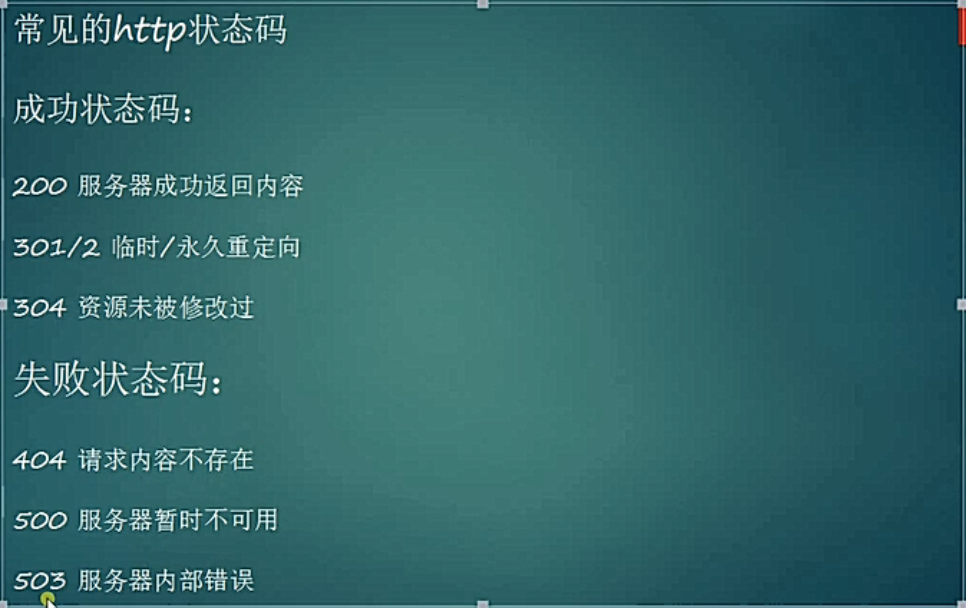
1. 先Client端发送来连接，请求报文
2. Server端接受连接后回复ACK报文，并为这次连接分配资源
3. Client端接受到ACK报文后也向server端发送ACK报文，并分配资源，这样TCP连接就建立了

Client端向server端断开连接过程：

1. Client端发送FIN报文（发起中断连接），server端收到FIN报文
2. Server端发送ACK报文，Server端进入确认状态，确认数据是否发送结束，Client端进入FIN\_WAIT状态，等待Server端发送FIN报文
3. 当Server端确定数据已经发送完成，则向Client端发送FIN报文，表示Server端数据发送结束
4. Client端接受到FIN报文后，向Server端发送ACK报文确定断开连接，Client端进入TIME\_WAIT等待状态，当经过2MSL后没有接收到回复，确认断开连接







服务器代理中转：服务器向另一个服务器请求数据的过程。通过服务器代理中转一个页面可以向其他服务器请求数据，方法可行是因为服务器和服务器之

间没有同源策略的限制。

domain，基础域名相同的两个脚本中写入相同的document.domain = ‘基础域名’，这样两个脚本之间就可以相互取数据了

**jsonp:原理，实际上也是GET请求**

1. script标签用src引入js文件时不受同源策略的影响
2. 把数据放到服务器上，并且数据为json形式
3. 因为我们无法监控通过<script>的src属性是否把数据获取完成，所以需要一个处理
4. 实现定义好处理跨域获取数据的函数
5. 在src属性末尾加上？cb=dojson，在另一个文件中需要传递数据必须在dojson()中

动画效果：

setTimeout，函数在固定时间会执行一次，但是只有当屏幕刷新时才会屏幕才会改变，也就是说如果执行时间和屏幕刷新时间不一样，那么就有可能出现卡顿或者掉帧的情况

**requestAnimationFrame()，**页面刷新前执行一次，用cancelAnimationFrame()来取消事件

兼容：window.requestAnimFrame = (function () {

return window.requestAnimationFrame ||

window.webkitRequestAnimationFrame ||

window.mozRequestAnimationFrame ||

function () {

window.setTimeout(callback, 1000/60);

}

})()

取消也需要进行兼容：

window.requestAnimFrame = (function () {

return window.cancelAnimationFrame ||

window.webkitCancelAnimationFrame ||

window.mozCancelAnimationFrame ||

function () {

window.clearTimeout(callback, 1000/60);

}

})()

本地存储：localStorage永久存储，sessionStorage临时存储

localStorage存储的信息在相同域下的文件都可以访问到，但是sessionStorage不可以

localStorage与cookie不同之处：



存储方式：

LocalStorage.prop = value;

若要存储对象，可以用JSON.stringify(obj)将对象转化成字符串形式进行存储

LocalStorage.setItem(prop, value)

提取方式：

可以直接使用localStorage.prop提取属性值

localStorage.getItem(prop)

删除方式：

localStorage.removeItem(prop)

localStorage.clear()清除全部

存储时不建议出现分号，逗号空格等奇怪的符号

encodeURIComponent()将符号转化成escape格式的字符串

decodeURIComponent()将escape格式字符串转化成符号

cookie:需要与服务器发送消息

查看cookie是否可用：navigate.cookieEnabled()

存值：document.cookie =”prop = value”

取值：document.cookie

history:用于存放浏览的历史记录

history.length可以显示已经浏览的页面个数

history.back() 返回上一次访问的页面

history.forward() 去返回的页面

history.go(number) number是正向后翻几页，负向前翻几页

history.pushState(state, title, url)在当前页面添加一条历史记录，但是不会刷新页面，所以会看到虽然网址改变了，但是内容还没有改变

state: 一个指定网址相关的状态对象，(history改变时)popstate事件触发时，该对象会传入回掉函数中，

title: 新页面的标题，所有浏览器都忽略了这个值，直接填null

url: 新的网址，必须与前一个页面处在同一个域，浏览器的地址栏将显示这个网址

history.replaceState(state, title, url)替换当前的历史记录

popstate事件：当历史记录发生改变的时候会触发该事件，但是调用history.pushState()和history.replaceState()不会出发popstate事件

hashchange事件，当页面的hash值发生变化时触发，常用于构建单页面应用

worker：当页面需要实现一个计算量巨大的操作，会卡顿页面就可使用worker来异步的计算，并返回结果

**实现方式：**

var work = new Worker(‘ .js’); 向.js文件发送数据

js文件中用onmessage事件来监听，数据储存在e.data中，计算完成后通过postMessage(data)来向主线程发送结果

主线程通过work.onmessage事件来接受数据，数据存储在e.data中

close()和terminate()都是用来结束一个worker，但是close()是在worker.js中调用的，而terminate()则是在worker对象上面调用的

geolocation:获取地理位置信息的对象

1. **getCurrentPosition**(successFun, errorFun, options);获取当前路由的位置信息，当成功获取信息后会向successFun中以参数形式传入Position对象，可以通过参数查看相关信息

当访问失败时，errorFun会接收到信息Positionerror对象，其中的值: code: 1，用户拒绝 2，访问不到，比如不能翻墙 3，连接超时

options对象，配置参数

enableHighAccuracy是否需要高精度位置，默认false

timeout单位ms，请求超时时间默认为infinity

maximumAge: 单位是ms，watchPosition方法则不停的取用用户的位置，并不停的更新用户信息。位置信息过期时间，设置为0表示无条件获取新的地理位置，默认为0

2. var watch = geolocation.watchPosition()用于注册监听器，在设备的地理位置发生改变的时候自动调用，参数与getCurrentPosition相同，用clearWatch(watch)来清除监听

3. devicemotion监听加速变化，属性在e中访问，需要手机陀螺仪支持，其包含的属性（均为只读）：(1)accelerationIncludingGravity(包括中心引力，z轴方向加了9.8，在x,y方向上的值两者相同)重力加速度 (2)acceleration重力加速度 (3)rotationRate(alpha, beta, gamma)旋转速度 (4)interval获取的时间间隔

4. devicerientation监听方向上的变化，包含的属性在事件对象e中访问：(1)alpha表示设备在Z轴上的旋转角度0~360 (2)beta表示设备在X轴上的旋转角度，范围是-180~180，用来描述前后旋转的角度 (3)gamma表示Y轴上的旋转角度，-90~90，用来描述设备左右旋转

window.DeviceOrientationEvent判断浏览器是否有devicerientation事件、

audio: 音频播放 video：视频播放

1. controls: 添加视频播放的控制器
2. autoplay：页面加载完成后自动播放
3. preload(none/metadata/auto)：预加载，规定是否在页面加载后载入数据

none: 不需要加载数据，你提前加载数据

metadata: 元数据，如市场，比特率，帧大小这样的元数据

auto: 先加载元素据和一部分播放数据，然后便播放边加载

1. loop: 是否循环播放
2. poster: 海报，只有video有这个属性

可以动态创建标签，document.createElement(‘audio/video’) new Audio(src)但是没有new Video()

方法：play()播放, pause()暂停, load()重新播放

状态：played正在播放，paused暂停，seeking在调试新的播放点，ended播放停止后值变为true

事件：play开始播放触发, pause暂停触发, loaded当加载完元素据时触发, loadeddata浏览器加载完当前帧数据准备播放时触发, ended当前播放结束后触发

volume：音量属性， muted设置是否静音

playbackRate: 播放速度[0.5, 4]其他值浏览器不会播放出声音

currentTime: 设置或获取当前播放位置

duration：获取播放的总时常

played: 返回当前播放了几段

offsered: 属性返回当前已经缓冲的时间段

seekable: 返回可以跳转的时间段

played, offsered, seekable都是TimeRanges对象， 都有一个length属性，及start() 和end()方法，length表示当前的一个时间段，start()与end()分别返回当前时间段的起始时间和结束时间

**拖拽：设置draggable =true，使元素可拖拽**

**拖拽相关事件：**

|  |  |  |
| --- | --- | --- |
| **被拖拽元素上的事件** | **dragstart**  **dragend** | **元素开始被拖拽时触发**  **元素拖拽结束时触发** |
| **目标元素上的事件** | **dragenter**  **dragover**  **drop** | **别拖拽元素进入目标元素时触发**  **被拖拽元素在目标元素上移动时触发**  **被拖拽元素在目标元素上放开鼠标时触发** |

要想触发drop事件必须在dragover中取消默认事件

drag过程中为了确定移动的目标，可以在dragStart中设置e.dataTransfer.setData(prop, value),并可以通过e.dataTransfer.getData(prop)获取数据

**fileReader:**

浏览器默认将拖进的文件下载并打开，可以在drop中取消默认事件，且要在dragover中取消默认事件来支持drop事件，e.dataTransfer.files中有相关的文件信息

fileReader用于读取文件：

相关方法：

abort()：终止读取

readAsBinaryString ()：将文件读取为二进制编码

readAsDataURL()：将文件读取为DataURL编码

readAsText()：见文件读取为文本

readAsArrayBuffer()：将文件读取为arraybuffer

相关事件：读取结束后结果存储在this.result

onloadstart: 文件读取开始时

onprogress: 文件读取中

onloaded: 读取完成时

onload: 成功读取完成时

onabort: 中断读取时触发

onerror: 读取出错时触发

WebSocket对象提供了一组API，用于创建和管理WebSocket连接， 以及通过连接来发送和接受数据。Websocket是一种新协议，和HTTP协议基本没有关系，只是因为兼容了现有浏览器的握手规范而已，借用了HTTP协议来完成握手

http1.0: 大多实现位每个请求和响应都要生成新的连接成为短链接

http1.1: 一个链接用于多次数据传输，会连接一段时间，成为长链接

http中服务器不能主动向客户端发起连接，而WebSocket服务器和客户端均可以主动发送消息

webSocket通过调用new WebSocket(url)接口时，浏览器就会与地址为url的WebSocket建立握手连接，过程为下：

1. browser与websocket服务器通过tcp建立连接，如果这个连接失败，将不会继续进行，应用程序会接受到错误消息通知
2. 在tcp成功建立连接后，browser通过HTTP协议传送websocket支持的版本号，协议的字版本号，原始地址，主机地址等信息给服务器端
3. Websocket接收到信息后，如果数据包数据和格式正确客户端和服务器端的协议版本号匹配，就接受握手链接，并给出相应的回复回复同样采用HTTP协议
4. Browser接收到回复信息后，如果数据包数据和格式没有问题就表示连接成功，触发onopen消息，此时web开发者就可以通过send接口向服务器发送数据，否则，握手连接失败，Web应用程序会收到onerror消息，并能够直到失败的原因

WebSocket事件：’ws://echo.websocket.org’测试的网址

open: 建立socket链接时触发

message: 客户端从服务器端接受数据时触发

error: 连接发生错误触发

close:连接关闭时触发

WebSocket方法：

.send() 发送数据

.close() 断开连接

优点：1. 可以双向建立连接 2. 不需要频繁的建立连接

前端模块化设计：

使开发模块化方式：

1. 函数
2. 对象
3. 匿名函数返回对象
4. 依赖(jQuery写法)将函数体以参数形式传进立即执行函数，并返回window
5. command.js：规范是每个文件都是一个模块，其内部定义的变量是属于这个模块，不会对外暴露，也就不会污染全局变量，该规范最初是在服务器端的node.js中，前端的webpack也是对command.js原生支持的

通过require方法来同步加载所要依赖的其他模块，然后通过exports或者mudule.exports来导出需要暴露的接口，浏览器不兼容command.js，CommandJs采用同步加载不同模块文件，适用于服务器端，因为模块文件都放在硬盘上加载速度非常快

// index.js

var module = require("module.js");

module.aa("杰哥很帅");

// module.js

module.exports = {

aa: function (str) {

console.log(str);

}

}

1. AMD采用异步加载方式，允许指定回掉函数，等加载完指定模块即可调用回掉函数，AMD得意产出require.js

AMD的核心是，通过define来封装一个模块，用require来加载一个模块，若封装的模块要调用其他模块的话，可以在define第一个参数加上一个数组对象，用来存储引用的模块名，html中的script标签需要使用data-main属性指定主js文件，也就是那个文件调用的其他文件，也可以在该文件中做相应的配置处理

// index.js

define(function () {

function aa () {

console.log('杰哥真帅')

}

})

// module.js

require(['index'], function (index) {

index.aa();

})

1. CMD是按需加载，什么时候需要用到某个文件就加载某个文件，核心思想是通过define来定义一个模块，通过exports来到出模块，每个模块都需要三个环境变量，require，exports，module，script中使用seajs.use()来引用主模块，主模块引用其他文件

seajs.use("main.js")

main.js

define(function (require, exports, module) {

var index= require("index");

console.log(index.name);//ddd函数

})

math.js

define(function (require, exports, module) {

function ddd () {

}

exports.name = ddd;

})

1. ES6提供了很多种导出方式：(可以通过export导出模块，对象，变量)
2. export{…obj}, 同样通过import{prop1, prop2}参数结构的方式来导入
3. export{oldName as newName} 给oldName起个别名newName,导入方式相同
4. export var/let fun = function () {}
5. export default “default”/obj 通过import tem from “…” 即tem<=“default”/obj
6. export \* from “…js” 将某文件全部导出

使用git打包文件的过程：

1. 下载webpack
2. 用init创建package.json文件
3. 在当前的开发环境中下载webpack ，会多出node\_modules和package-lock.json两个文件
4. 下载webpack-cli到全局中
5. 设置默认入口src文件夹
6. 运行webpack，会生成出口文件夹dist文件夹
7. 接下来就是四大模块进行环境配置，并且每次文件发生变化都要用webpack重新打包

四个核心模块：用于配置不同模块的相关属性

entry：入口模块，指webpack应该从那个文件开始调用其他模块

output：输出模块，输出一般为\_bundles文件(自己设置)

loader：转换器，在module中设置，将所有类型(例如.less)的文件转化成webpack可以识别的文件类型

plugins: 插件，从打包到压缩，一直到重新定义环境中的变量，插件接口功能巨大，可以处理各种各样的任务

loader模块常用的一些转换器：

css-loader处理css文件, style-loader将css加入到style标签中, postcss-loader对css进行兼容性处理, file-loader打包文件所需转换器, url-loader打包图片, less-loader处理less文件(处理less文件还需要下载less这个插件), babel-loader将ES6的代码转化成ES5的代码

webpack指令

webpack –watch: 开启自动打包

-g表示将安装包安装到全局，默认是安装到当前目录下

-save-dev表示下载到当前开发环境中

npm init创建一个package.json文件

webpack四大模块环境配置：在自己创建的webpack.config.js文件中配置注意点

\_dirname表示webpack.config.js文件的父级的地址

output中设置filename：[name]表示入口文件名

module中需要加入不同的loader文件来加载不同的文件，其中rules中test通过正则匹配文件的后缀名来确认文件类型，use中引入不同的loader文件，loader文件会从后向前来将文件转化

Uglifujs-webpack-plugin的配置，在package.json中的scirpt中添加一个start，当在命令行中输入npm run start可以执行此命令，可以是多个命令，webpack-dev-server是开启本地服务器，--devtool是否及如何生成源映射，eval-source-map把每个module封装到eval里包裹起来执行，并且会在末尾追加注释，把注释里面的sourceMap都转化为DataURL，--progress显示编译百分比过程，--colors提示显示高亮颜色

源映射：为了在源码中相应位置显示打包后的代码中的错误，让源文件和打包后的文件产生一种映射关系，叫做源映射

<img>标签格式引入的图片和利用DataURL引入的base64格式的图片：

DataURL的优点：

* 当访问外部资源很麻烦或受限时
* 当图片是在服务器端用程序动态生成，每个访问用户显示的都不同时。
* 当图片的体积太小，占用一个HTTP会话不是很值得时。

缺点：

* Base64编码的数据体积通常是原数据的体积4/3，也就是Data URL形式的图片会比二进制格式的图片体积大1/3。
* Data URL形式的图片不会被浏览器缓存，这意味着每次访问这样页面时都被下载一次。这是一个使用效率方面的问题——尤其当这个图片被整个网站大量使用的时候。

gulp: 主要有4个功能：gulp.src, gulp.task, gulp.dest, gulp.watch都是函数

gulp.task:创建任务

gulp.src: 获取文件，将文件设置为文件流

gulp.dest: 将文件流变成文件并设置在指定地址

.pipe()起传输作用，gulp操作都需要经过pipe()

gulp.watch(“监听的文件地址”， “触发的任务名”)

gulp.task(“default”, [‘任务1’，’任务2’]按顺序执行任务

.pipe(connect.reload()) //自动刷新页面

gulp中的常用插件:

gulp-htmlclean: 压缩html文件

gulp-imagemin: 压缩图片

gulp-uglify: 压缩js文件

gulp-concate: 合并文件

gulp-strip-debug:

gulp-less: 处理less文件

gulp-postcss: css文件添加前缀，即兼容性处理

gulp-cssnano: 压缩css文件（cssnano也可以）

connect.reload() //自动刷新

connect.server({

port: 8080; //设置端口号

livereload：true //浏览器自动刷新

})

ES6新方法：

let:

函数：箭头函数不具有this属性，如果需要使用的话，箭头函数的this会自动绑定到离自己最近(父级)的包含this的元素上

var name = "window";

var obj = {

name: "shuaige",

print: () => {

console.log(this.name) //window

}

}

扩展字符串：``

ES6的数组新方法：

keys()遍历数组的键值

values()遍历数组的值

entries()遍历数组的键值以及值，用[index, value]表示

以上方法都会返回一个不同的迭代器，通过for of循环来遍历

迭代器方法：next()可以手动将迭代器对象指向下一个元素

find(function(){})按照条件寻找第一个满足条件的元素，并返回相应的元素值，findIndex()和find()相似，不同的是findIndex()返回下标

includes(ele)返回bool值，表示是否有这个值，和indexOf相似，indexOf返回下标，并且不能判定NaN，但是includes可以

对象拓展：

obj = {

[keys] : value

}这里的keys可进行字符串拼接

Object.getOwnPropertyDescriptor(obj, keys)可以获取对象的属性及其属性配置

Object.is()与恒等于相似，都用与判断两值是否相等，但是is可以判断NaN与NaN相等，并且可以判断+0和-0不想等

Object.assign(obj1, obj2…)将若干可枚举属性放到一个对象中,并返回该对象，会把不是引用的参数值隐式转换为对象，除了(null, undefined,会报错)

对象的拓展运算符：…obj ,与数组扩展运算符相似，只是括号不一样

Object.keys(), Object.values() , Object.entries()与数组方法不同的是，方法会返回一个数组，也是迭代器对象，可以通过for of循环遍历

CLASS：构造类，使用时需要使用new操作符，class也是函数

class Person {

constructor(name, age=4){ //可设置默认值

this.name = name;

this.age = age;

}

eat(){ //方法设置在构造函数的原型上

}

}

var p1 = new Person(name, age);

Person.prototype.sleep = function () { //class可以通过prototype添加属性

}

Object.assign(Person.prototype, {sleep(){}}) //通过assign添加方法

console.log(typeof Person) //function,使用时必须用new操作

console.log(p1.eat === Person.prototype.eat) //true

构造函数使用new构造对象时，自动调用constructor来构造实例对象，构造函数的方法在构造函数的原型上

相对与原生js的立即执行函数，class也有立即执行class立即生成一个实例对象

var a = new class{

constructor(name, age = 0){

this.name = name;

this.age = age;

}

…

}(name, age)

constructor作为class的入口函数，在构造实例对象时会自动触发，所以可以通过constructor来硬绑定函数，防止函数被单独抽取出来(此时this为undefined)，**class中会默认为严格模式，而严格模式下this要么指向某一个对象，要么指向undefined，不会指向window**

可以通过箭头函数解决这个问题：

class Person{

constructor(name, age) {

this.name = name;

this.age = age;

this.print = (name = "ddd") => {

this.print(name);

}

}

print(name) {

console.log(name)

}

}

箭头函数没有自身没有作用域， 它的this会指向父级作用于在这里就是class的实例对象

Class继承：class A extends B{}A会继承B的属性及方法，super会默认执行，并默认的调用父级的constructor，也可以用过super访问父类的属性

此时的原型链为：**A.\_\_proto\_\_ == B**

Promise:

Promise()有两个状态，1：pending等待状态， 2：resolved成功或者rejected失败状态

const promise = new Promise(function (resolved, rejected) {

//resolved和rejected为两个函数，成功时执行不同方法

//当执行resolved时，状态会由pending转到resolved

//当执行rejected，状态会由pending到rejected

})

promise.then(fn1, fn2) //会判断创建promise时执行的是resolved(执行fn1)还是rejected(执行fn2)

var p4 = Promise.all([p1, p2, p3]) //如果p1, p2, p3返回状态都是resolved的时候,才返回resolved状态，否则返回错误状态的信息

var p5 = Promise.race([p1, p2, p3]) //谁先返回状态Promise就返回谁的状态

Promise和then手动封装：

function MyPromise (fn) {

if(fn == "undefined"){

throw Error(`${fn} is not a function`)

}

var self = this;

this.resolvedArr = [];

this.rejectedArr = [];

this.status = "pending";

this.data = null;

function resolved (data) {

if(self.status === "pending"){

self.status = "resolved";

self.data = data;

setTimeout(function () {

self.resolvedArr.forEach((ele, index) => {ele()});

})

}

}

function rejected (err) {

if(self.status === "pending"){

self.status = "rejected";

self.data = data;

setTimeout(function () {

self.rejectedArr.forEach((ele, index) => {ele()});

})

}

}

fn(resolved, rejected);

}

MyPromise.prototype.then = function (onResolved, onRejected) {

var self = this;

if(this.status == "resolved"){

return new MyPromise((resolved, rejected) => {

var res = onResolved(self.data);

if(res instanceof MyPromise){

res.then(resolved, rejected);

}else{

resolved(res);

}

})

}

if(this.status == "rejected"){

return new MyPromise((resolved, rejected) => {

var res = onRejected(self.data);

if(res instanceof MyPromise){

res.then(resolved, rejected);

}else{

resolved(res);

}

})

}

if(this.status == "pending"){

return new MyPromise((resolved, rejected) => {

self.resolvedArr.push((

(onResolved) => {

return function () {

var res = onResolved(self.data);

if(res instanceof MyPromise){

res.then(resolved, rejected);

}else{

resolved(res);

}

}

}

)(onResolved));

self.rejectedArr.push((

(onRejected) => {

return function () {

var res = onRejected(self.data);

if(res instanceof MyPromise){

res.then(rejected, rejected);

}else{

rejected(res);

}

}

}

)(onRejected));

})

}

}

GIT远程仓库建立及GIT使用：

Git工作区及文件状态：

工作目录：working directory

暂存区：staging area

Git仓库区：repository

git remote –v: 查看远程仓库

git remote rename oldname newname: 给远端重命名

git remote add <name> <remote>: 添加源仓库地址为远程仓库

git clone <repository>: 克隆仓库

git status: 查看文件状态，

git add <file>: 将文件放置在暂存区, git add .表示当前所有改动的文件

git commit：把文件提交到仓库中

git checkout <file>: 撤销工作区改变，相对于上一次暂存区

git reset HEAD <file>: 暂存区文件撤销,HEAD后面加上^或者~1表示回退1个版本

git reset -- hard HEAD~n <file>:回退全部，包括HEAD，index，working tree

git reset -- mixed HEAD <file>: 回退部分，包括HEAG，index

git reset -- soft HEAD <file>: 只回退HEAD

浏览器的线程：

**js引擎线程**：解释执行js代码，用户输入，网络请求

**GUI线程**：绘制用户界面，与js主线程是互斥的

**http网络请求线程**：处理用户的get, post请求，等返回结果后将回调函数推入任务队列

**定时触发器线程**：setTimeout, setInterval等待时间结束后讲执行函数推入任务队列中

**浏览器事件处理线程**：将click, mouse等交互事件发生后将这些事件放入任务队列中

任务队列 == macrotask != microtask

microtask会在任务队列执行前执行，

1. 具体执行时间：任务队列(macrotask = task queue)回调后执行，前提条件是当前没有其他执行中的代码。
2. 每个task末尾执行

toString() : Array和Object原型上都有toString方法，数组的toString方法可以将数组转化为字符串。且将中括号去除，Object的toString方法返回对象的基本类型，除了undefined，null

String(): 可以将数组转化为字符串和toString一样，但是String是构造函数，另外可以把null, undefined等转化为字符串型，String(new Object) === (new Object).toString.call(new Object)

JSON.stringify(): 可以将数组,对象等转化为字符串且不去除任何东西，当stringify处理的数据含有undefined，null等在不同情况下的处理是不同的，(1)单独出现是处理为undefined (2)在数组中处理为null (3)在对象中会直接忽略

**重排重绘：**

重排：当DOM元素影响了元素的几何属性，浏览器需要重新计算元素的几何属性，同样其他元素的几何属性也会和位置也会因此受到影响。浏览器会使渲染树中受影响的部分失效，并重新构造渲染树

重绘：浏览器重新绘制受影响的部分到屏幕上

**重排发生的情况:**

添加或删除可见的DOM元素

元素位置改变

元素的尺寸改变

内容改变

页面渲染器初始化

浏览器窗口尺寸改变

**重绘发生的情况：**

当元素外观发生改变但是不改变布局的时候

**触发重排的属性：**

1.**盒子模型**相关属性会触发重布局：

width, height, padding, margin, display, border-width, border, min-height

2.**定位属性及浮动**也会触发重布局：

top, left, right, bottom, position, float, clear

3.改变**节点内部文字结构**也会触发重布局：

text-align, overflow, font-weight, font-family, line-height, vertical-align, white-space, font-size

**触发重绘的属性：**

color, border-style, border-radius, visibility, text-decoration, background, outline, box-shadow

**设计模式：**

设计模式的六大原则：

单一指责原则：一个方法只用来做一件事情

开闭原则：在不修改源代码的前提下进行拓展

迪米特法则：降低耦合度，是两个模块相互独立，不发生相互关系

接口隔离原则：使用多个专门的接口，而不是用单一的总接口

依赖倒置原则：

里氏替换原则：所有引用基类的地方必须能透明地使用其子类的对象