# Simi Documentation

Source Link: https://github.com/saidake/simi/tree/master/docs

Author: Craig Brown
Version: 1.0.0
Date: Feb 2, 2025

## DomBom / Concept

### 浏览器

浏览器进程：浏览器是多进程，每一个 tab 标签有一个独立进程（多个空白 tab 标签会合并成一个进程），浏览器内核的渲染进程属于浏览器多进程中的一种。

GUI 渲染线程:　渲染页面，解析 HTMLCSS 构成 RENDERTREE，重绘或回流都会调起该线程（JS 引擎和 GUI 渲染不能同时进行）

JS 引擎线程:　单线程工作，负责解析运行 JavaScript 脚本（JS 引擎工作时，GUI 渲染线程会被挂起在 JS 任务队列，等待 JS 引擎空闲的时候继续执行---- JS 运行耗时过长就会导致页面阻塞）

事件触发线程:　事件被触发时，该线程 把对应的事件回调函数 添加到任务队列的队尾，等待 JS 引擎处理。

定时器触发线程:　开启定时器触发线程 来计时并触发计时，浏览器定时计数器并不是由 JS 引擎计数的，阻塞会导致计时不准确（计时完成后会被添加到任务队列中，等待 JS 引擎处理）

异步 http 请求线程:　http 请求的时候会开启一条请求线程（请求完成有结果了之后，将请求的回调函数添加到任务队列中，等待 JS 引擎处理）

### 进程

进程是资源分配的最小单位，线程是进程的执行流，是 CPU 调度的最小单位（如何调度进程和线程，完全由操作系统决定，程序自己不能决定什么时候执行，执行多长时间。）

进程由至少一个线程组成，同个进程之中的多个线程之间是共享该进程的资源的

进程间不会相互影响，一个线程挂掉将导致整个进程挂掉

进程使用的内存地址可以上锁，即一个线程使用某些共享内存时，其他线程必须等它结束，才能使用这一块内存

进程使用的内存地址可以限定使用量

### DOM 渲染

DOMTREE:　解析 页面上所有的 html 结构 构建 DOMTREE，并行请求 css/image/js　（内链和外链的 css，都会阻碍后续的 DOMTREE 渲染，但是能在解析 css 的同时解析 dom）

CSSOM:　等待 css 文件下载完成，根据 css 嵌套结构渲染 CSSOM　（根据继承规则，底层样式覆盖上层样式，但不包含不在屏幕上显示的元素<link><title><script><meta>等）

RENDERTREE:　CSSOM 构建结束后，和 DOM 一起生成 RENDERTREE　（渲染树中不包含<head>和 display：none 或尺寸为 0 的元素，但可以包含 visibility：hidden 的元素）

REPAINT：渲染树 中的元素外观　发生改变，不影响布局时，产生重绘

REFLOW：渲染树 中的元素的布局（尺寸，内容，位置，新增）发生改变时，产生重绘回流　（回流必将引起重绘，而重绘不一定会引起回流）

【 JS 获取布局属性值会引起回流（如：offsetLeft、scrollTop、getComputedStyle 等），浏览器需要通过回流计算最新值 】

性能优化:　复杂 dom 操作:　可以先隐藏(display:"none")，操作完成后再显示

多个 dom 创建:　使用 DocumentFragment 创建完后一次性的加入 document，或使用字符串拼接方式构建好对应 HTML 后再使用 innerHTML 来修改页面

布局属性值:　缓存 Layout 属性值，如：var left = elem.offsetLeft; 这样，多次使用 left 只产生一次回流

table 布局:　避免用 table 布局，因为 table 元素一旦触发回流就会导致 table 里所有的其它元素回流

css 表达式:　避免使用 css 表达式(expression)，因为每次调用都会重新计算值（包括加载页面）

css 属性简写:　尽量使用 css 属性简写，如：用 border 代替 border-width, border-style, border-color

修改 css 样式:　尽量批量修改元素样式：elem.className 和 elem.style.cssText 代替 elem.style.xxx

### 事件循环

JS 事件循环:　分为 浏览器循环 和 Node 事件循环，两者的实现技术不一样（浏览器 Event Loop 是 HTML 中定义的规范，Node Event Loop 是由 libuv 库实现）

JS 线程　JS 有一个 main thread 主线程和 call-stack 调用栈（微任务和宏任务都会放在任务队列等待执行）

JS 调用栈　　函数被调用时，会被添加到栈顶，执行完成后从栈顶部移出该函数，直到栈内被清空（处理函数嵌套的情况）（线程处理完异步等方法后，会将函数添加到任务队列中）

同步任务、异步任务　同步任务会在调用栈中按照顺序排队等待主线程执行

异步任务等待有结果后将回调函数添加到任务队列中，等待主线程空闲的时候取出执行【任务队列是先进先出的队列结构。JS 单线程中的同步任务和异步任务。】

浏览器事件循环　宏任务 script 执行完毕，栈被清空，主线程空闲时 ---> 读取任务队列的回调函数到调用栈中执行　script 宏任务 ---> 微任务队列 ---> 宏任务队列 ---> 微任务队列 .....

第一次事件循环中，JavaScript 引擎会把整个 script 代码当成一个宏任务执行，执行完成后再检测本次循环中是否寻在微任务

宏任务队列可以有多个，微任务队列只有一个，不同宏任务队列按照提前设置的队列优先级来调用

执行过 task 和 大量的 microtask 后，会进行 renderUI（不是每次循环都会执行 renderUI，但每次间隔 60hz 一帧 16ms）

task 宏任务：　script, setTimeout, setInterval, I/O, render UI 【node：setImmediate】

microTask 微任务：　Promises, Object.observe 【node：process.nextTick MutationObserver】

// aaa 10　wait 10　acc　优先执行同一时间线上的任务　宏 aaa 微 acc >>> 宏 ddd 微 dada 微 dbdb >>> 宏 kkk >>>宏 wait　(同路线的 宏任务重叠会跳过后方的宏任务)

// ddd 10　dada　dbdb　在 setTimeout 外部 resolve 时会导致直接执行微任务

// kkk 10

**事件总线**

一个页面发生的事件要传递到上一个界面，或者别的界面 就构成了一条事件线

事件总线是对发布-订阅模式的一种实现。它是一种集中式事件处理机制，允许不同的组件之间进行彼此通信而又不需要相互依赖，达到一种解耦的目的。

事件是由事件源和事件处理组成。

发布订阅模式主要有两个角色：

发布方（Publisher）：也称为被观察者，当状态改变时负责通知所有订阅者。

订阅方（Subscriber）：也称为观察者，订阅事件并对接收到的事件进行处理。

发布订阅模式有两种实现方式：

简单的实现方式：由 Publisher 维护一个订阅者列表，当状态改变时循环遍历列表通知订阅者。

委托的实现方式：由 Publisher 定义事件委托，Subscriber 实现委托。

**优化页面加载**

css 样式放在文件的 style 标签

js 文件放在文件末尾（页面静态优先显示，再加载交互效果）

压缩 js, css 代码（去掉注释空格换行）

拆分比较大的 js，css 文件

减少页面 http 请求数量

服务器开启 gzip 压缩

多个域名存储网站资源（过多的域名会使 DNS 解析负担加重，因此一般控制在 2-4 个 ）

懒加载

精灵图

## DomBom / Dom

**document 内置**

**成员**

document.documentElement　获取根标签 html　// var ifm = document.getElementByid('ifm'); var dom =

ifm.contentWindow.document;　获取 iframe 里的 document　　window.frames[0]
document.body　　获取 body
document.title　　获取标题
document.cookie　　新增或更新 一个或多个页面的 cookie 值

document.cookie="userId=828; userName=hulk";　　注意，第一个后方的 userName 等不会新增或更新，不会生效

document.cookie="str="+escape（"I love ajax"）；　　编码特殊字符

var expires = new Date();
expires.setTime(expires.getTime() - 10);
document.cookie = 'username='+escape('echo')+';expires=' + expires.toGMTString();　　通过过期时间删除 cookie

**方法**

getElementById("fa")　　id　　返回一个标签
getElementsByTagName("div")　　标签名　　返回标签数组 HTMLCollection　（html 元素集合）
getElementsByName("nam")　　name 属性的值　　返回标签数组 HTMLCollection　（h5 低版本不支持）
getElementsByClassName("cls")　　类样式的名字　　返回标签数组 NodeList （节点集合）

document.getElementById("xxxiframe").contentWindow;　iframe 通过获取到的 window 对象操作 HTML 元素，这和普通页面一样　// topWin.document.getElementById("exit").style.visibility = "visible";

querySelector("selector")　　选择器　　返回一个标签
querySelectorAll("selector")　　选择器　　返回标签数组 NodeList

createElement("p")　　返回一个 对象　搭配 appendChild(pObj)使用　可点添加属性
write("<a> </a>")　　页面加载时使用（所有内容加载完毕后会使用 会清空页面所有内容）
execCommand（“copy”）　　执行一些文本编辑命令，是否显示用户界面，额外参数（ dom.select() 选中文本，触发 select 事件）　【deprecated】

**dom 内置**

**标签属性**

style.display="block"　style="color:red;"　　设置行内样式（合并单词时去掉- 首字母变大写）（属性只有一个值时可用 Boolean 表示值，可以添加一些自定义属性存入数值）（外界无法直接获取行内样式　但可以直接传值设置行内样式）
className=“”　　设置 class（会覆盖原样式 class）
innerHTML　　标签内容，支持 HTML 标签　（清空原有内容）（innerText　textContent 仅能输出所有文本 不包含子标签）
innerText　　双标签内文本　（表单用 value）（低版本火狐不支持）
textContent　标签内文本　（ie8 不支持）
contentEditable　　是否可以编辑【可以使 div 可以输入】

value　　获取表单值
id　　获取 id
disable=true　　disable 属性可以直接设置或获取
files　　文件类型 input 的文件列表 dom.files[0]　（上传文件才有值）
attributes　　获取标签上所有属性 // dom.attributes.kkkk　
scrolling　　属性可设置或返回 iframe 框架是否拥有滚动条 // dom.scrolling="yes"

**属性方法**

setAttribute("name",100)　　设置属性　　（相当于在原标签里加　this.是 js 点添加对象成员，不能创建属性）
getAttribute("name")　　获取自定义属性　（自定义属性只有 getAttribute 能访问 ）
removeAttribute("classname")　移除属性或 class（名字）　　getElementsByClassName("")可以清空 class 值

getAttributeNode("id")　　获取属性节点【标签内部属性节点 id style 等】

father.appendChild(pObj)　　追加到子元素最后（动态创建追加后可注册事件）

father.removeChild(oldChild)　　移除子级元素　// var remove_obj = document.getElementById(id);　var parent_obj = remove_obj.parentNode;　parent_obj.removeChild(remove_obj);

father.insertBefore(newChild, refChild)　新子元素 插入到 参照子元素的前面

father.replaceChild(newChild, refChild)　新子元素 替换 参照子元素

father.cloneNode(true)　　true 带属性克隆　false 仅标签　返回克隆调用的对象

**子元素节点**

nodeType　节点的类型（1----标签节点,　2---属性节点,　3---文本节点）（可以在事件内部使用，this.nodeType）

nodeName　节点的名字（大写标签名　　小写属性名　　#text）

nodeValue　节点的值　（null　　属性值,　　文本内容）

parentNode　父级节点

parentElement　父级元素

childNodes　子节点数组

children　子元素数组

firstChild　　第一个子节点（IE8 获取子节点或兄弟节点会得到的是元素，不支持获取元素）

lastChild　　最后一个子节点

firstElementChild　第一个子元素

lastElementChild　最后一个子元素

previousSibling　　某个元素的前一个兄弟节点

nextSibling　　某个元素的后一个兄弟节点

previousElementSibling　某个元素的前一个兄弟元素

nextElementSibling　　某个元素的后一个兄弟元素

**三大系列**

offsetLeft　offsetTop　　定位偏移　（相对最近的 relative 的父盒子的左上角　当前盒子的坐标-----父盒子 padding，子盒子 margin 会影响坐标）

offsetWidth　offsetHeight　　占位宽高　（wh+padding+border）　offsetParent

clientLeft　clientTop　　左上边框宽度

clientWidth clientHeight　去边占位宽高（wh+padding）　// document.documentElement.clientHeight

document.body.clientHeight　可视区域

clientX　clientY　　可视区域坐标　　【IE 不支持】

scrollLeft　scrollTop　　向上/左卷曲的长度　（window 没有 scroll 系列，scroll 可修改）　// document.documentElement.**scrollTop**　document.body.**scrollTop**

scrollWidth　scrollHeight　　去边占位宽高（wh+padding），不足默认为父级宽高，减去卷曲距离与可视区域盘判断　// document.documentElement.scrollHeight　document.body.scrollHeight

scrollX: 0　scrollY: 1200　　滚动条 滚动的距离

window.pageYOffset　window.pageXOffset　　页面可视区域向上/左卷曲的长度（只读）

window.innerWidth　window.innerHeight　　页面可视区域宽高

**dom 事件**

**可覆盖事件**

onclick　　点击　（循环添加事件时创建很多相同的匿名事件会占用大量内存 ）

onfocus　　获取焦点

onblur　　失去焦点

onkeyup　键盘抬起　（复制粘贴无法触发）

onkeydown　键盘按下　（复制粘贴无法触发）

onmouseover　鼠标进入（子元素也触发 会覆盖父元素）

onmouseout　鼠标离开（子元素也触发 会覆盖父元素）
onmousedown　鼠标按下
onmouseup　鼠标抬起
onmousemove　目标元素内的鼠标移动（搭配 e.clientX　e.clientY）　父盒子范围包括子盒子（定位超出也算）　可以被其他盒子遮挡

onscroll　滚动条滚动事件（配合 scrollTop　scrollLeft 使用）
onresize　窗口改变大小　window.onresize
onsubmit　提交事件

select()　选中一个 <textarea> 元素或者一个带有 text 字段的 <input> 元素里的所有内容
remove()　移除当前元素

window.onload　页面加载完毕
window.onunload　页面关闭后触发事件　【谷歌不支持】
window.onbeforeunload　页面关闭之前触发事件　【谷歌不支持】

## H5 新增事件

onchange　表单内容改变
oninput　当前元素内容的改变（添加，删除）
oninvalid　正则验证不通过触发
window.ononline　网络连通触发
window.onoffline　网络断开触发

## 多事件绑定

addEventListener("无 on 事件",　(event)=>{},　false)　绑定事件，function 函数时，内部 this 是对象（false 默认事件冒泡--先寻找最下方的事件执行，true 事件捕获--直接先执行当前事件再执行下方事件）　【IE8 不支持】
removeEventListener("click", fun, false)　解绑事件，必须指定移除的命名事件函数
attachEvent("有 on 事件",　(event)=>{} )　绑定事件，function 函数时，内部 this 是 window　【谷歌，火狐，IE11 不支持】
detachEvent("onclick", fun)　解绑事件，必须指定移除的命名事件函数

onclick=(event)=>{}　绑定事件函数　// dom.onclick = func;　dom.onmouseover = func;　多事件
onclick=null　解绑事件函数

```
function fun(e) {
   switch(e.type){case "click": ;break; case "mouseover": ;break;}　// 多事件绑定同一函数
}
```

## 事件参数 event

event=event || window.event;　有 e 用 e，火狐不支持 window.event
clientX　clientY　基于浏览器窗口—鼠标坐标（absolute 定位注意父级向上移动　盒子 margin）
pageX　pageY　基于页面文档—鼠标坐标（clientX+scrollLeft)　【 IE 不支持】
screenX　screenY　基于手机屏幕—鼠标坐标（加上手机工具栏）
eventPhase　返回事件传播的当前阶段 1 （捕获阶段: 从下到上）　2 （目标阶段: 最开始选择的那个）　3 （冒泡阶段: 从上到下==点击上方，会触发下方的事件）
type　无 on 事件的类型 "click"
keycode　按键按下/抬起的值
stopPropagation();　阻止事件冒泡　window.event.cancelBubble=true;
**preventDefault**();　阻止浏览器默认行为（超链接等）　【 onclick=function(){ return false; } 只执行函数　不执行默认事件-跳转提交表单等　】
currentTarget　触发事件的当前 dom（target 不稳定）

target === this　　触发此事件的 dom（this 可因冒泡发生变化，e.target 不会变化）

dataTransfer.setData("MIMEtype", "Data")　　数据的存储与获取（只能在 ondrop 事件中获取）

## DomBom = BomAPI

### window 成员

#### window

parent　　指这个页面的父页面，如果一个窗口没有父窗口，则它的 parent 属性为自身的引用（window　指当前的页面）

devicePixelRatio　设备像素比

frameElement　　返回嵌入当前 window 对象的元素 (比如 <iframe> 或者 <object>),如果当前 window 对象已经是顶层窗口，则返回 null.

if (window.frameElement) window.frameElement.src = 'http://mozilla.org/';　　如果当前窗口被包含在一个框架里面，则将该框架的地址跳到'http://mozilla.org/'

atob("xxEncodedStr")　解码使用 base-64 编码的字符串　【返回字节字符串】

#### window.location

href　　整个 URL 地址　// http://localhost:8080/examsystem/index.html?r=table%2Frenderjs#fsf

location.href="http://www.jd.com"　　跳转到指定页面　　当前页面跳转（JS 编程式跳转）

hash　　#fsf　　URL 内，# 和后面的内容

host　　localhost:63342　　主机名和端口号（省略使用默认端口）

hostname　　localhost　　主机名

pathname　　/examsystem/index.html　　文件的路径

port　　80　　端口号

protocol　　https:　　协议

search　　?keyword=fafafa&from_source=banner_search　搜索的内容

origin　　localhost:63342　　发起请求的主机名和端口号（省略使用默认端口）

assign("http://www.jd.com")　方法　　当前页面跳转

reload();　　重新加载--刷新

replace("http://www.jd.com")　没有历史记录（不能后退）

msSaveOrOpenBlob("xxFileData", "xxFileName")　　用户在客户端上保存文件，方法如同从 Internet 下载文件，这是此类文件保存到"下载"文件夹的原因。

#### window.history

back()　　后退

forward()　　前进

push('/admin')　　记录当前网站，跳转后可以返回这个路径

replace('/admin')　替换当前网站，无法返回

go(-1)　　后退，正数为前进

#### window.navigator

userAgent　用户浏览器类型

platform　　浏览器所在系统平台类型

appVersion　设备类型（android）

language　　浏览器语言环境　　// "en-US"

#### window.URL

createObjectURL("xxfile")　根据传入的参数创建一个指向该参数对象的 URL. 这个 URL 的生命仅存在于它被创建的这个文档里，新的 URL 对象，表示指定的 File 对象或 Blob 对象 【string】

revokeObjectURL("xxurl")　释放一个通过 URL.createObjectURL()创建的对象 URL，当你要已经用过了这个对象 URL，然后要让浏览器知道这个 URL 已经不再需要指向对应的文件的时候，就需要调用这个方法.

## window 方法

### 编码

encodeURIComponent( 'http://a.com/test.php?name=stale' )　把字符串作为 URI 组件进行编码　//
http%3A%2F%2Fa.com%2Ftest.php%3Fname%3Dstale

btoa("")　base-64 编码字符串

atob("")　base-64 解码的字符串

eval("x=10;y=20;document.write(x*y)")　执行字符串内部的代码，返回计算结果

eacape("")　显示中文的编码 unescape("")显示编码的中文

### 样式

getComputedStyle(element,"伪类选择器")　全部 CSS 属性对象【IE8 不支持】　getComputedStyle(eleobj, null).left

eleobj.currentStyle.left　全部 CSS 属性对象【谷歌火狐不支持】

getSelection().removeAllRanges()　防止文字选中

document.selection.empty()　防止文字选中-兼容

### 窗口

alert()　弹框

confirm()　确定取消弹框 返回 true false （不使用，不能设置样式）

prompt("")　弹框提示并输入

open("")　新窗口打开链接

### 定时器

setInterval( ()=> {}, 1000)　定时器，异步执行, 返回值定时器的 id 值，　页面加载完毕后：　过一秒-执行　过一秒-执行..... （提前执行一次函数 解决延迟的 1s）　var id = setInterval(function() {}, 0);　while (id--) clearInterval(id);　清空前方所有定时器【其实可以选择到 iframe 内的元素】

clearInterval(timerid);　清理定时器（否则在内存占空间）

setTimeout( ()=>{}, 1000)　异步延时执行，返回值就是定时器的 id 值　过一秒-执行结束 【setTimeout 即使时间为 0 也会在 script 标签最后执行】

clearTimeout(timerid);　清理定时器（否则在内存占空间）

# DomBom / WebApi

### 表单

dom.setCustomValidity("string")　表单设置默认的提示信息（正则验证不通过时）

### class 方法

**classList**.add("red")　追加一个 class 样式

**classList**.remove("blue")　移除一个 class 样式

**classList**.toggle("black")　切换一个 class 样式（有就删除，无就添加）

**classList**.contains("black")是否包含 class 样式（返回 Boolean）

**classList**.item(2)　获取一个 class 样式

dataset["aName"]　获取自定义属性 （data-a-name="" 自定义属性规范）

### 全屏

requestFullScreen()　开启全屏显示　【浏览器前缀】

document.cancelFullScreen()　退出全屏显示（整个文档）　【浏览器前缀】

document.fullscreenElement　是否为全屏状态　【浏览器前缀 **moz**FullScreenElement **webkit**FullscreenElement **ms**FullscreenElement】

### 文件读取

new FileReader()　文件读取对象

result　读取文件结果

readAsText()　读取文本文件，返回文本字符串 （默认编码 UTF-8）

readAsBinaryString()　读取任意文件，返回二进制字符串 （传递后台存储）

readAsDataURL(dom.files[0])　读取图片或可嵌入文件，读取文件完毕时存储在 **r**.result （DataURL 字符串：data:开头，将资源转换为 base64 编码的字符串存储在 url 中）

abort()　　中断读取*

onabort　　读取文件中断片时触发

onerror　　读取错误时触发

onload　　文件读取成功完成时触发

onloadend　　读取完成时触发，无论成功还是失败

onloadstart　　开始读取时触发

onprogress　　读取文件过程中持续触发

**拖拽事件**

draggable="true"　　设置为拖拽元素（图片和超链接默认可以拖拽）

ondragstart　　应用于拖拽元素，当拖拽开始时调用　　（为 document 添加事件 通过 e.dataTransfer.getDate("")识别其他事件里触发的对象 - 避免全局变量存储对象识别）

ondragleave　　应用于拖拽元素，当鼠标离开拖拽元素原位置时调用

ondrag　　应用于拖拽元素，整个拖拽过程都会调用--持续

ondragend　　应用于拖拽元素，当拖拽结束时调用

ondragenter　应用于目标元素，当拖拽元素鼠标进入时调用

ondragover　应用于目标元素，当停留在目标元素上时调用

ondrop　　应用于目标元素，当在目标元素上松开鼠标时调用-dom.appendChild（默认不会触发，可在 ondragover 中阻止浏览器默认行为可以触发）

e.dataTransfer.getDate("MIMEtype")　获取其他事件里设置的数据（识别元素 id）

ondragleave　应用于目标元素，当鼠标离开目标元素时调用

**地理定位**

navigator.geolocation.getCurrentPosition(p=>{}, error=>{},　{})　【PC 浏览器默认无法获取地理信息：使用第三方接口（百度地图 高德地图 qq 地图　开放平台 JavaScript API）】

p=>{}　　获取地理信息成功的回调函数（自动传入地理信息参数 p）

p.corrds.latitude　　纬度

p.coords.longitude　经度

p.coords.accuracy　精度

p.coords.altitude　海拔高度

err=>{}　获取地理信息失败的回调函数 （自动传入错误信息 error）

error.PERMISSION_DENIED　　用户拒绝定位请求

error.POSITION_UNAVAILABLE　定位信息不可用

error.TIMEOUT　　请求超时

error.UNKNOWN_ERROR　　未知错误

{}　获取当前地理信息的方式

enableHighAccuracy: true/false　是否使用高精度

timeout: 300　　设置超时事件，单位 ms

maximumAge: 300　　浏览器重新获取地理位置信息的时间间隔，单位 ms

**web 存储**

var ws= window.**sessionStorage**　存储在内存中，存储容量 5mb 左右（只在当前页面生效，其他页面或不同浏览器则失效 - 刷新无影响）【默认为字符串，需要 JSON.parse】

ws.setItem(key, value)　键值对存储数据

ws.getItem(key)　　获取数据（获取不到为 null）

ws.removeItem(key)　删除数据（key 错误不会报错，也不会删除数据）

ws.clear()　　清空所有数据

var ws= window.**localStorage**　本地存储，存储在硬盘中，存储容量 20mb 左右---只存放字符串数据（同一浏览器中永久生效，不同浏览器失效）

ws.setItem(key, value)　键值对存储数据

ws.getItem(key)　获取数据（获取不到为 null）

ws.removeItem(key)　删除数据（key 错误不会报错，也不会删除数据）

ws.clear()　清空所有数据

```
var request = window.indexedDB.open('my-database', 1);  打开一个数据库【数据库名，类型[1 新建，省略 默认当前版本]】
    request.onsuccess = function(event) {
        db = event.target.result;
        var transaction = db.transaction(['newUsers'], 'readwrite');      newUsers 是对象仓库名
        var objStore = transaction.objectStore('newUsers');
        var req = objStore.get(1);   // 读取数据，put 修改或新增数据
        req.onsuccess = function(e) {
        if (req.result) {
           console.log('已经查询到数据为：');
            console.log(req.result);
        } else {
            console.log('未查询到数据');
           objStore.put({"a":1}, 3)  更新或新增数据  3  {"a":1}
        }
        }
    }
```

**离线网页缓存**

<html lang="en" manifest="xxx.appcache">　缓存配置文件路径 - 文本文件（扩展名建议 appcache）【必须在服务器配置 MIMEtype 识别　.appcache　text/cache-manifest】

xxx.appcache　CACHE MANIFEST

CACHE:

../images/1.jpg　#需要缓存的文件清单列表　* 代表所有文件

NETWORK:

../images/3.jpg　#配置每一次都需要从服务器获取的文件清单列表

FALLBACK:　#配置文件无法获取则使用指定文件进行替代　/ 代表所有文件

../images/3.jpg ../images/2.jpg

**视频播放器**

**load()**　重新加载音频/视频元素

**play()**　开始播放音频/视频

**pause()**　暂停当前播放的音频/视频

**currentTime**　视频播放的当前时间（秒为单位，小数值，可被赋值）【dom 支持错误时 直接打开源文件测试】

**duration**　视频总时间　（秒为单位，小数值）

**paused**　视频播放状态 true/false

**oncanplay**　可以播放视频/音频时触发 - 加载完毕

**ontimeupdate**　播放时间变化事件（currentTime 变化）

**onended**　播放完毕-重置

结构<video>

播放暂停　（视频加载完毕显示视频）

进度条　播放时间 已加载时间　总时间　（最上层盒子用来响应事件，宽度为动态获取的百分比，跳播 - offsetx 获取偏移值求出跳播时间设置 currentTime 的值）

时间显示

全屏

## AJAX

构造函数 XMLHttpRequest

var xhr =new XMLHttpRequest(); 浏览器对象

open('GET', 'add.php?id=1', [true]) 请求行（协议版本自动设置）【默认为 true 异步，false 为同步 - 等到全部响应报文接收完再执行下方代码】

setRequestHeader('key', 'value') 请求头 'Content-Type' 'application/x-www-form-urlencoded' 【请求体需要 urlencoded 格式时必须指定，即使设置了 cookie 请求头也无法携带 cookie】

withCredentials = true 携带 cookie

send('key1=value1&key2=value2'); 请求体，可为空 【格式对应 {"foo":"123"} Content-Type application/json】

**onreadystatechange**=function(){ } XHR 状态改变事件 0 - INSENT 代理 XHR 被创建，但未调用 open()方法

1 - OPENED open()方法已经被调用，建立连接

2 - HEADERS_RECEIVED send()方法已经被调用，获取到状态行和响应头

3 - LOADING 响应体下载中，可能为空，可能包含部分数据

4 - DONE 响应体下载完成，可以直接使用 responseText

**onerror** = function(e){ } 请求失败

**onload** = function(e){ } 4 状态-响应体下载完成 HTML5

getAllResponseHeaders() 响应头内容

getResponseHeader('server') 指定响应头内容

xhr.responseText 响应体内容（字符串）【不管响应状态码多少都会触发回调函数】

xhr.response 响应体内容（类型根据 responseType 变化）【兼容性不好】

xhr.responseType 响应体类型（""="text" "json"）

**readyState** 状态数 01234【判断响应体下载完成】（console 展开对象显示为最后的状态）

status 状态码

statusText 状态描述

JSON.parse('str') JSON 字符串 转换 对象

JSON.stringify(json) 对象 转换 JSON 字符串

解析 json header("Content-Type: application/json")

解析 XML header("Content-Type: application/xml") xhr.responseXML.documentElemtnet.getElementByTagName('name')[0] 获取服务端的 XML 数据的根标签

缺点：影响 SEO 优化，搜索引擎抓取网页主页面源代码（百度无法抓取 ajax 加载的数据，GoogleBing 能抓取 ajax 加载的数据）

异步上传文件

var data=new **FormData**() HTML5 新增成员，用于 ajax 客户端预服务端传递二进制数据

data.append('avatar', this.files[0]) DOM 属性 files

var xhr=new XMLHttpRequest() xhr.open('POST', '/xxx.php') xhr.send(data) ..... ajax 发送文件 $_FILES['avatar']访问

```
let oldFnOpen = XMLHttpRequest.prototype.open;
XMLHttpRequest.prototype.open=function(){
  return oldFnOpen.apply(this, arguments)
}
```

兼容方案（IE5/6 不支持）：var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new ActiveXObject('Microsoft.XMLHTTP')

## canvas

var x=document.querySelector("cavas") 获取 canvas 标签

var ctx=x.**getContext**('2d')　　获取绘制工具箱，无 3d 效果（只有 canvas 标签才能使用）

**canvas**　　直接获取 canvasDOM 对象

**width height**　　设置画布宽高

**beginPath**()　　开启新路径（需再次描边显示）

**closePath**()　　自动闭合路径 – 与 beginPath 无关联（起点和 lineto 终止点无法完全闭合）

**moveTo**(100, 100)　移动画笔，坐标 x，y　（移动后绘制，再描边显示 这个过程可多次）

**lineTo**(200, 100)　绘制直线路径，目的坐标 x，y （默认黑色宽度 1px 对齐像素刻度绘制-2px 显示 颜色填充不饱和-变淡）【需要 stroke 描边才能显示】

**stroke**()　　对路径描边显示（描边填充可同时进行）

**fill**()　　对路径填充（非零填充 – 从区域向外拉出一条线，相对区域中心，顺时针+1 逆时针-1）

**strokeStyle**='blue' 'rgb(0,0,0)'　描边颜色 （默认无颜色，描边必须设置）

**setLineDash**( [5,10,15] ) 实 5 空 10 实 15 设置虚线

**getLineDash**()　　获取虚线不重复的一段排列数组

**lineDashOffset**="20"　　设置虚线前后偏移量 - 长度不变

**lineWidth**=10　　描边线条宽度 px

**lineCap**='butt'　　线两端帽子类型（方的会边长）【"butt" 默认 "round" 圆的 "square" 方的】

**lineJoin**='miter'　　相交线的拐点　　【"miter"默认 "round" 圆的 "bevel" 斜面 】

**globalAlpha**=0.2;　　填充或描边的透明度

**fillStyle**='blue'　　填充颜色

**rect**(x,y,w,h)　　绘制矩形路径 -左上角为位置（没有独立路径 - 可统一填充，描边）

**strokeRect**(x,y,w,h)　描边矩形（有独立路径，不影响别的绘制- beginPath）-无需绘制路径可配合 fillStyle

**fillRect**(x,y,w,h)　　填充矩形（有独立路径，不影响别的绘制- beginPath）-无需绘制路径

**clearRect**(x,y,w,h)　　擦除指定矩形区域（可清除区域中所有内容）

**arc**(x,y, R, **0, Math.PI/2**, [false]) 绘制圆弧路径【圆心坐标 半径 始末弧度点 [方向] 】(先将画笔移到中心可闭合成扇形)

**font**='20px 微软雅黑' 设置字大小和字体

**strokeText**("ff",**x,y**, [20]) 描边字体 【绘制文本 文本坐标-左下角 文本最大宽度-超出长度会压缩】

**fillText**("ff",**x,y**, [20]) 填充字体 【绘制文本 文本坐标-左下角 文本最大宽度-超出长度会压缩】

**textAlign**=　　文本对齐方式-基于左下角坐标 【left center right start 默认 end】

**direction**=　　无法可视的效果【ltr rtl】 left to right right to left

**textBaseline**=　　设置基线垂直对齐方式【top middle bottom hanging-印度文基线 alphabetic-英文基线 ideographic-中文基线】

**measureText**().width 获取文本对象宽度 （下划线 - 获取下划线坐标描线）

**drawImage**(img,x,y, **[w,h]**)　　绘制图片 【图片对象/canvas 对象/video 对象 xy 图片绘制左上角坐标 wh 图片绘制缩放尺寸】

**drawImage**(img, **[x,y,w,h]** ,[x1,y1,w1,h1]) 绘制图片 【图片对象/canvas 对象/video 对象 xywh 图片中的截取坐标宽高 x1y1w1h1 画布中绘制坐标宽高】

save()　　保存画布的当前所有状态

restore()　　你可以调用任意多次 save 方法。每一次调用 restore 方法，上一个保存的状态就从栈中弹出，所有设定都恢复。

toDataURL()　　转换成 base64 压缩图片 URL .....

translate(100, 100)　移动坐标轴（x 是左右偏移量，y 是上下偏移量）

scale(0.5, 1)　　坐标轴的 x y 缩放比例（）

rotate(Math.PI/2)　旋转 （原点在 0, 0）

var img= new Image() img.onload=function(){ **ctx.drawImage()** }　　img.src="" （加载图片到内存但不显示）

var lg=ctx.createLinearGradient(x0,y0, x1,x1)　　创建渐变方案（两点表示渐变长度方向） ctx.fillStyle=lg

lg.addColorStop(1, ‘pink’)　　在 100%位置添加颜色

遍历渐变　　lineTo 不断绘制点并使 rgb 颜色增长

遍历点绘制函数　lineTo(x, y)　y=2x  sinx

折线图　网格 - 设置网格大小　　两循环横纵向网格线（去除变宽 2px 问题 x/y-0.5）

坐标轴- 设置坐标轴外边距 - 设置箭头边长度　计算原点和坐标轴终点（H-间距 W-间距）

坐标点- 设置点坐标 - 设置点方形尺寸　　坐标点周围绘制一个方形并填充 size=6

连线 - 需要知道上一个点的位置 （prevCanvasX = canvasX）

扇形图　根据数据比例转换成弧度，计算每次始末弧度-循环开启新路径，移动画笔绘制扇形路径（外存开始弧度）

标题 - 固定从弧中心延伸出去线的长度-利用角度计算终点坐标　根据圆心判断下划线方向 根据文本获取长度

矩形说明　说明，矩形大小，外边距，矩形间距

帧动画　绘制出一张截取出的图片　定时器{ ctx.clearRect(); index++ ctx.drawImage() if(index..){index=0} }

行走小人　x 轴为索引 y 方向确定（索引超出变为 0）

构造函数　定义图片路径，ctx 宽高，步数，方向，xy 偏移步数　同时初始化

初始化函数　回调函数{ 1. 绘制初始状态 2. document 注册事件改变 that 的各个成员数据---每走一步 x 轴图片索引加一 循环播放，与 y 轴方向无关 }

加载图片函数　传入图片加载完毕时的回调函数

绘制图片函数　直接使用 this 各项成员

## websocket

var ws = new WebSocket('ws://localhost:8080');

readyState　属性返回实例对象的当前状态，共有四种。

WebSocket.CONNECTING　0，表示正在连接。

WebSocket.OPEN　1，表示连接成功，可以通信了。

WebSocket.CLOSING　2，表示连接正在关闭。

WebSocket.CLOSED　3，表示连接已经关闭，或者打开连接失败。

binaryType = "arraybuffer";　"blob";　指定收到的是 ArrayBuffer 数据 或 二进制数据 blob 对象

onopen = event=> {　指定连接成功后的回调函数

event.code;

event.reason;

event.wasClean;

}

onclose = event=> {}　指定连接关闭后的回调函数

onmessage = event=> {　指定收到服务器数据后的回调函数 （服务器数据可能是文本，也可能是二进制数据 blob 对象或 Arraybuffer 对象）

event.data

event.data.byteLength

event.data.size

}

addEventListener('open', event => { } );　指定多个回调函数

send('str' 或 file 或 );　向服务器发送数据

send(file)

# DomBom / Cases

## 1. 假滚动条

计算滚动条高度 box/content=bar/scroll

计算移动距离 barMove/contentMove=barMaxMove/contentMaxMove

按下　获取相对盒子坐标 spaceY=e.client-offset　移动 document 滚动条 e.client-spaceY

抬起　document

## 2. scroll 滚动条到底部

scrollTop + clientHeight == offsetTop　Y 轴上的滚动距离 + 可视区域的高度 = 盒子到顶部的距离

## 3. 懒加载

页面：　一些不必要元素 可以在主页面加载完毕时 创建追加

列表：　拉到在底部时，重新请求 10 条数据，追加到数组中

图片：　<img src="blank.jpg" data-src="a.com/a.jpg">

img 的 src 设为同一张空白未加载的图片，真正的图片地址存储在自定义属性中（如 data-src）

当 js 监听到该图片元素进入可视窗口时，即将自定义属性中的地址存储到 src 属性中，达到懒加载的效果。

## 4. 定时器动画

渐变：添加定时器，缓慢减少 opacity 到 0 就 clearInterval

延时禁用：　到时间修改按钮 value disable

图片时钟：　获取当前时间 根据时间设置 Img 的 src

摇一摇：　用随机数设置 margin 配合定时器

## 5. 循环排他

先统一清除所有　设置当前属性 current

手风琴：排他 减速宽度变大变小效果（背景图片会因变宽显示出来）

## 6. 轮播图

移动端轮播图方案　wrapper 宽高设置为 100%，图片都为定位 left:100% left:200%　只需要移动 wrapper 就可以使所有图片移动

移动效果　style="transform: translateX(-200%); transition: all 0.3s ease 0s; "

限制宽高盒子 > 很宽的盒子，定位调整位置实现滚动 >内部浮动靠齐组件　获取需要的对象-----克隆添加 li

1. 一排小按钮：　自定义索引外存-----mouseover 事件｛循环排他｝　（可添加优化加载）
2. 左右按键（将要移动时 跳到替身的 left 位置再移动）　左按钮（pic--移动 替身为最后一张）　右按钮（pic++移动 替身为第一张）

以停留索引（替身不停留要特殊判断）同步一排小按钮

3. 自动播放　开启定时器调用右按键的事件函数（鼠标进入清理定时器 鼠标离开重新开启定时器）

移动式　通过 relative 直接移动大宽度 wrapper 实现滚动　（内部图片固定 float 排列，将第一张图片复制到最后一张后，防止最后一张跳到第一张有空白）【给 left 添加动画】

渐隐式　通过 float:left 和 relative，left:-100px 将大宽度 wrapper 下的所有内部图片挤压在最左边。

当前显示的盒子 z-index:1 opacity:1 实现渐隐 【给 opacity 添加动画】

全局操作：　给显示的图片盒子添加无作用类 class="slider_active" 给每个图片盒子添加属性 data-index="0"

## 7. 搜索输入匹配内容

onkeyup：　如果 dv 存在则删除（每次都重新判断创建 dv）

this.value 获取输入框内容，创建临时数组

把输入框.value 和总数组对比，匹配到开头的放到临时数组

输入框.value 或临时数组为空时，如果 dv 存在则删除，并 return 结束 onkeyup(删除变空时用)

父级 appendChild("dv")

以临时数组长度循环创建显示内容的 p 元素

dv.appendChild("pObj")---设置 p 内容为临时数组

p 添加 onmouseover， onmouseout 时间

## 8. 浏览器滚动条控制导航栏

大于某个高度 ｛ 添加一个定位样式 className="nav fixed" --------下方盒子添加一个 marginTop 稳定位置 ｝

小于原高度　{恢复样式　className="nav"　---------marginTop="0px";}

## 9. 筋斗云

鼠标进入 { animate(cloud,this.offsetLeft)移动云彩到当前位置 }

鼠标点击 {获取 this.offsetLeft 并外存 a 默认为 0} 记忆位置

鼠标离开 {animate(cloud,a) 移动到上次点击的位置 a}

## 10. 旋转木马

数组旋转 push-shift unshift-pop------assign 重新分配 添加定时器锁（一个定时器执行完才能执行下一个定时器）

## 11. 拖拽对话框

按下 {【获取相对盒子坐标 space=e.client-offset】

document 移动-- e.client-space} doucment 抬起 （定距离法）

## 12. 贪吃蛇

贪吃蛇 食物域 { 食物数组 食物类（ 宽高 颜色 坐标）

初始化函数（删除食物，创建食物并存入数组，设置各种样式）

删除函数（移除食物并从数组删除）}

小蛇域 { 小蛇数组 小蛇类（宽高 方向 初始身体样式数组）

初始化函数（删除小蛇 循环创建小蛇并设置各种样式 ）

移动函数（移动一步-从后到前先身体再头部- 获取食物判断吃到）

删除函数（移除小蛇并从数组删除） }

游戏域 { 游戏类（食物 小蛇 地图 this 外存）

初始化函数（食物初始 小蛇初始）

自动移动小蛇函数（走一步初始化 判断撞到）

绑定按键函数（改变小蛇的方向）}

小蛇变长时，最后两个方块初始坐标相同，移动时被前面坐标覆盖