## HTML

### <!DOCTYPE>

用来声明html版本

html5：<!DOCTYPE html>

HTML4 三种文档类型：Strict、Transitional 以及 Frameset。

### HTML5标准提供了哪些新的api

**1.HTML5的新特性**：

新的内容标签：header nav content footer article aside

更好的单元格体系:

音频、视频API:video audio

画布(Canvas) API//基于JavaScript

内联SVG//可伸缩矢量图形 (Scalable Vector Graphics)，基于xml

地理(Geolocation) API

网页存储(Web storage) API:localStorage,sessionStorage

拖拽释放(Drag and drop) API

新的表单控件calander date time email url search

新的input类型 color date datetime datetime-local email

### html5有哪些新特性，如何处理html5新标签的浏览器兼容问题，如何区分html和html5？

1.h5不基于SGML，主要是关于图像、位置、存储、多任务等功能的增加；canvas、video、audio、本地离线存储：localStorage长期存储数据，浏览器关闭后数据不丢失；sessionStorage的数据在浏览器关闭后自动删除；语义化更好的内容元素：article/footer/header/nav/section;表单控件：calendar date、time、email、search；新的技术：webworker、websocket、geolocation

因此不需要对dtd进行引用，但是需要doctype来规范浏览器的行为，而html基于sgml，所以需要对dtd进行引用，才能告知浏览器文档使用的文档类型

2.ie8/7/6支持通过document.createElement方法产生的标签，可以利用这一特性让浏览器支持h5新标签，浏览器支持新标签后还需要添加标签默认的样式，当然也可以使用成熟的框架例

3.如何区分h5和html：doctype声明、新增的结构元素/功能元素

### xhtml和html有什么区别

html是一种基本的web网页设计语言，xhtml是一个基于xml的置标语言；最主要的不同：xhtml元素必须被正确嵌套、必须关闭、标签必须小写、文档必须拥有根元素

### 简述一下src与href的区别

src用于替换当前元素；href用于在当前文档和引用资源之间确立联系。

src是source的缩写，指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置，需要下载且下载时会暂停其他活动

href是Hypertext Reference的缩写，指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，不会暂停其他活动

### 行内元素有哪些，块级元素有哪些？

行内元素：a b span img input select strong I em

块级元素：div ul li ol table p dl dt dd h1-h6 from

常见空元素：br hr img input link meta

### 常见的块级(block)元素和行内(inline)元素，以及它们有何不同？

**行内：a,b,span,img,strong,i,input,br**

**块级：div,h1~h6,li,table,ol,ul,hr**

不同：行内元素的默认宽度只与内容有关，块级元素默认宽度直接占满一行。

行内元素水平排列，块级元素垂直排列；块级元素可包含行内元素，反之不行。

### 请阐述table的缺点

a. 太深的嵌套，比如table>tr>td>h3，会导致搜索引擎读取困难，而且，最直接的损失就是大大增加了冗余代码量。

b. 灵活性差，比如要将tr设置border等属性，是不行的，得通过td

c. 代码臃肿，当在table中套用table的时候，阅读代码会显得异常混乱

d. 混乱的colspan与rowspan，用来布局时，频繁使用他们会造成整个文档顺序混乱。

e. table需要多次计算才能确定好其在渲染树中节点的属性

f. 不够语义

### 对web标准以及w3c的理解和认识

标签闭合，标签小写，不乱嵌套，提高机器人搜索几率，使用外链css和js脚本，结构行为表现的分离，文件下载与页面速度更快，内容能被更多的用户访问，内容能被更广泛的设备所访问，更少的代码和组件，容易维护改版方便，不需要变动页面内容，提高网站易用性;语义化的html即直观的认识标签，对搜索引擎的抓取有好处

### rem em rpx px fr vw vh ch

px像素，rpx微信独有，自适应屏幕，规定屏幕宽750rpx；em绑定当前字体大小；rem绑定html字体大小 fr 网格布局剩余宽度的分配 vw 自适应视口,1vw为视口宽度1%，vh代表视口高度1% ch 数字0的宽度

## CSS

### link与@import引入css的区别

link是html自带标签，@import是css特有语法，link不会暂停解析，@import会暂停解析直到下载完css

### 选择器选择prop="test"的元素

**[prop="test"]{}**

### CSS3有哪些新特性？

新增选择器 p:nth-child（n）{color: rgba（255, 0, 0, 0.75）}

弹性盒模型 display: flex;

多列布局 column-count: 5;

媒体查询 @media （max-width: 480px） {.box: {column-count: 1;}}

个性化字体 @font-face{font-family:BorderWeb;src:url（BORDERW0.eot）；}

颜色透明度 color: rgba（255, 0, 0, 0.75）；

圆角 border-radius: 5px;

渐变 background:linear-gradient（red, green, blue）；

阴影 box-shadow:3px 3px 3px rgba（0, 64, 128, 0.3）；右，下，阴影尺寸

倒影 box-reflect: below 2px;

文字装饰 text-stroke-color: red;

文字溢出 text-overflow:ellipsis;省略号

背景效果 background-size: 100px 100px;

边框效果 border-image:url（bt\_blue.png） 0 10;

转换 transform

旋转 transform: rotate（20deg）；

倾斜 transform: skew（150deg, -10deg）；

位移 transform:translate（20px, 20px）；

缩放 transform: scale（.5）；

平滑过渡 transition: all .3s ease-in .1s;

动画 @keyframes anim-1 {50% {border-radius: 50%;}} animation: anim-1 1s;

### css引入方式有哪些？CSS选择符有哪些?哪些属性可以继承?优先级算法如何计算?内联和important哪个优先级高?css的基本语句构成

1.css引入方式：内链 导入 外链

2.css选择符：id>class>标签>\*

3.内联和important中important优先级高

4.选择器｛属性1：值1；属性2：值2；....｝

### 盒模型

CSS 盒子模型(Box Model)

所有HTML元素可以看作盒子，在CSS中，"box model"这一术语是用来设计和布局时使用。

box

CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：边距，边框，填充，和实际内容。**边框(border)、边界/边距(margin)、补白/填充(padding)、内容区(content)**

盒模型允许我们在其它元素和周围元素边框之间的空间放置元素。

### 画一个三角形

div {

width:0px;

height:0px;

border-top:10px solid red;

border-right:10px solid transparent;

border-bottom:10px solid transparent;

border-left:10px solid transparent;

}//transparent在css颜色属性中代表透明黑色，即一个类似rgba(0,0,0,0)这样的值。

### 画一个半圆形

div{

width: 100px;

height: 50px;

border:1px solid black;

background-color: blue;

border-radius: 100px 100px 0 0;//从左上开始顺时针，右上，右下，左下

}

### 画一条0.5px的线

先画一条1px的线再

1. 缩放transfrom：scale（0.5）；
2. 渐变 background-image: linear-gradient(0deg, #f00 50%, transparent 50%);

### css animation和transition的区别

Animation和transition大部分属性是相同的，他们都是随时间改变元素的属性值，他们的主要区别是transition需要触发一个事件才能改变属性，而animation不需要触发任何事件的情况下能随时间改变属性值，并且transition为2帧，从from .... to，而animation可以一帧一帧的。

### visibility=hidden, opacity=0，display:none的区别

opacity=0只是变透明了，还会响应

visibility=hidden只是隐藏该元素，不能响应，但不改变布局

display:none则相当于从原来的渲染树中删除该元素。改变了布局，导致重排

// z-index涂层遮盖只是遮盖了该元素(无法响应)

### opacity和rgba的a的区别

opacity有继承性，子元素默认继承，rgba的a则只用于当前元素

### BFC

**（block formatting context，用于清除浮动，防止margin重叠等）**

直译成：块级格式化上下文，是一个独立的渲染区域，并且有一定的布局规则。

BFC区域不会与float box重叠

BFC是页面上的一个独立容器，子元素不会影响到外面

计算BFC的高度时，浮动元素也会参与计算

哪些元素会生成BFC：

根元素

float不为none的元素

position为fixed和absolute的元素

display为inline-block、table-cell、table-caption，flex，inline-flex的元素

overflow不为visible的元素

## 综合

### 元素在容器中居中

1.父元素固定宽高，利用定位及设置子元素margin值为自身的一半。

2.父元素固定宽高，子元素设置position: absolute，margin：auto平均分配margin

3.css3属性transform。子元素设置position: absolute; left: 50%; top: 50%;transform: translate(-50%,-50%);即可。

4.将父元素设置成display: table, 子元素设置为单元格 display: table-cell。

5.弹性布局display: flex。设置align-items: center; justify-content: center

6. 父元素弹性布局display: flex, 子元素设置margin：auto

### 垂直水平居中

1.绝对定位+margin:auto

把要垂直居中的元素相对于父元素绝对定位，

{

**margin: auto;//使元素自动居中**

width: 150px;//任意

height: 50px;//任意

//规定了元素外边距的边界与父元素边界(就是原来默认位置)的距离值。

left: 0;

right: 0;//去掉则变为垂直居中

top: 0;

bottom: 0;//去掉则变为水平居中

position: absolute;//继承父元素的边界

}

2.弹性盒模型+margin: auto

{

//父类display：flex;

//子类直接margin：auto

}

3.transform

{

// transform 属性允许你旋转，缩放，倾斜或平移给定元素。本质上这种居中技巧是建立在平移量的精准计算之上的。即需要知道父类长度和宽度和子类的长度和宽度。

transform: translate(, )

}

4.硬算

//需要知道父类长度和宽度和子类长度和宽度

5. 弹性盒模型+align-item:center+justify-content:center

### cookie和session的区别和用法

存储位置：cookie数据存放在客户的浏览器上，session数据放在服务器上。

安全性：cookie不是很安全,是明文，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗。考虑到安全应当使用session

限制：单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用COOKIE。

### cookie，localstorage和sessionstorage的区别

单个cookie保存的数据不能超过4K ,webstorage包含了localstorage和sessionstorage，大小无限制，随浏览器而变，大多数最大储存5m信息，保存在本地。

localstorage生命周期无限，除非清除缓存。

sessionstorage当关闭当前页面时被清除，不同页面之间不能共用

cookie能设置生命周期，生命周期一到则无效，默认关闭页面时清除。

### token

Token，其实就是服务端生成的一串加密字符串，储存在本地用于认证客户端身份，由服务端设定失效时间。

优势：无状态、防csrf(跨域请求攻击)、多站点使用，支持移动平台、性能快

### 如何优化页面的加载速度

总结：1.网络原理2.资源引入3.代码

1、网络原理服务器响应时间:有独立的服务器,提高Web服务器的质量,移除不必要的插件

2、网络原理浏览器缓存 :减少HTTP请求，从而提高网站加载速度

3、资源引入gzip压缩:它的工作原理是在发送HTML和CSS文件到浏览器之前压缩文件大小

4、异步脚本:网页负载就不必依赖于这些异步脚本

5、网络原理内容分发网络（CDN）:CDN是位于不同地理位置的服务器组成的网络。每个服务器都拥有所有网站的文件副本。有网站访问者请求文件和网页时就可以直接从就近网站服务器发送过来（也可以是从负载最小的服务器）

6、优化JavaScript、HTML和CSS:删除所有不必要的空 格和注释从而减小文件大小

7、资源引入置于顶部的样式表和底部的脚本

8、资源引入避免阻塞型的JavaScript和CSS

9、资源引入JavaScript的延迟解析

10、启用Keep Alive:用户请求网页时浏览器首先需访问HTML文件,然后它才读取这些文件，并请求与其他资料相关联。如果“Keep Alive”选项被禁止,下载网站的进程就会增加从而拖累网站速度。启用KeepAlive的另一个好处是，可以减少CPU使用

11、图像和文件格式:建议使用JPEG格式，而不是GIF和PNG图像，除非图像包含Alpha因子或者是透明的

12、优化代码：不使用内联CSS

13、资源引入文件分离:可以增加并行下载的数量

14、网络原理尽量减少HTTP请求:减少网站上的对象数量；最小化网站上的重定向数量；使用CSS Sprites技术；结合JavaScripts和CSS

15、资源引入去掉不必要的插件

16、网络原理减少DNS查询(DNS lookups)

### 一个页面从输入url到加载完成的过程都发生了什么，请详细说明

1、首先，在浏览器地址栏中输入url

2、浏览器先查看浏览器缓存-系统缓存-路由器缓存，如果缓存中有，会直接在屏幕中显示页面内容。若没有，则跳到第三步操作

3、在发送http请求前，需要域名解析(DNS解析)，解析获取相应的IP地址

4、浏览器向服务器发起tcp连接，与浏览器建立tcp三次握手

5、握手成功后，浏览器向服务器发送http请求，请求数据包

6、服务器处理收到的请求，将数据返回至浏览器

7、浏览器收到HTTP响应

8、浏览器解析渲染页面:解析html源码;生成Dom树、解析css样式、js交互

### 浏览器渲染过程

浏览器渲染的过程主要包括以下五步：

浏览器将获取的HTML文档解析成DOM树。

处理CSS标记，构成层叠样式表模型CSSOM(CSS Object Model)。

将DOM和CSSOM合并为渲染树(rendering tree)，代表一系列将被渲染的对象。

渲染树的每个元素包含的内容都是计算过的，它被称之为布局layout。浏览器使用一种流式处理的方法，只需要一次绘制操作就可以布局所有的元素。

将渲染树的各个节点绘制到屏幕上，这一步被称为绘制painting。

### 重排与重绘

重绘（repaint或redraw）：当盒子的位置、大小以及其他属性，例如颜色、字体大小等都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面上。重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。

触发重绘的条件：

1.改变字体

2.增加或者移除部分样式表如颜色

3.内容变化（input框输入文字）

4.激活部分css伪类（例如 :hover）

5.计算offsetWidth、offsetHeigth属性（浏览器的可见高度）

注意：table及其内部元素可能需要多次计算才能确定好其在渲染树中节点的属性值，比同等元素要多花两倍时间，这就是我们尽量避免使用table布局页面的原因之一。

重排（重构/回流/reflow）：当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建, 这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。

重绘和重排的关系：在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树，完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程称为重绘。所以，**重排必定会引发重绘，但重绘不一定会引发重排。**

减少重绘重排的方法有：

不在布局信息改变时做DOM查询，

使用csstext,className一次性改变属性

使用fragment

### 强缓存和协商缓存

HTTP Cache 是我们开发中接触最多的缓存，它分为强缓存和协商缓存。

**强缓存：直接从本地副本比对读取，不去请求服务器，返回的状态码是 200。**

**协商缓存：会去服务器比对，若没改变才直接读取本地缓存，返回的状态码是 304。**

强缓存：expires，cache-control(未设置no-cache和no-store)，

协商缓存：pragma(只有no-cache)，cache-control(设置no-cache)，last-modified，etag。

no-store：不缓存，直接请求最新资源。

**优先级：pragma>cache-control>expires>last-modified&&etag**

expires：是一个缓存字段，以格林尼治时间表示过期时间点，与客户端时间相比是否过期，受客户端时间影响。

cache-control：HTTP1.1新增，值有max-age,s-maxage（是一个时间长度字段，表示还有多少秒过期，且 s-maxage 的优先级高于 max-age）；还有public和private（前者表示客户端和服务器端都能缓存，后者只能客户端缓存，默认值是 private，当设置了 s-maxage 的时候表示允许代理服务器缓存，相当于 public）；no-cache和no-store，no-cache表示向服务器验证当前资源是否更新，no-store则直接请求服务器的当前资源，不询问是否更新。

last-modified: ，以格林尼治时间记录资源最后修改的时间,启用后会在请求头中返回if-modified-since字段（记录上一次修改的时间），若两者不一致则更新last-modified并返回修改后的资源。

etag: 是基于资源的内容编码生成的一串唯一的标识字符串，启用后请求头会带有 if-none-match 字段，对比两者即可。

浏览器刷新：F5刷新，刷新按钮，网页右键刷新；CTRL+F5刷新（硬性重新加载），清空其他缓存并强行设置 no-cache，直接重新请求资源

### 跨域

源：如果两个页面（接口）的协议，端口或者域名都相同，那么两个页面就有相同的源。

浏览器为了安全会限制跨域http请求

**解决方案**

1. CORS

（Cross-Origin Resource Sharing）,跨域资源共享

当使用XMLHttpRequest发送请求时，如果浏览器发现请求违反了同源策略就会自动加上一个额外的http请求头 origin；**后端在接受到请求后确定响应后会在 Response Headers 中加入一个属性 Access-Control-Allow-Origin；**浏览器判断响应中的 Access-Control-Allow-Origin 值是否和当前的地址相同，匹配成功后才继续响应处理，否则报错

缺点：忽略 cookie，浏览器版本有一定要求

1. 代理

**服务端请求不会跨域的特性**；

前端向服务器发送请求，经过代理，请求需要的服务器资源，让接口和当前站点同域。

缺点：需要额外的代理服务器

1. **JSONP等**

**标签能跨域加载资源的特性**，但是js读不到其中的内容。<script src="..."></script>，<img>，<link>，<iframe>等。代表为JSONP：通过动态创建<script src=”anotherOrigin”>，再请求一个带参网址实现跨域通信。缺点：易受xss攻击，只能用get请求

例：<script>

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.domain2.com:8080/

login?user=admin&callback=onBack';

document.head.appendChild(script);

// 回调执行函数

function onBack(res) {

alert(JSON.stringify(res));

} </script>

1. websocket

客户端和服务器之间存在持久的连接，而且双方都可以随时开始发送数据，绕过http协议。发送给后端，利用后端代理

1. **location.href location.href不受浏览器跨域限制**
2. postMessage

window.postMessage(message,targetOrigin) 方法是html5新引进的特性，可以使用它来向其它的window对象发送消息，无论这个window对象是属于同源或不同源

### js内存泄漏

1. 闭包
2. 全局变量
3. 定时器如setInterval
4. 递归的终止条件
5. while(true)等代码终止条件

### router实现原理

1. Hash路由 利用url上hash的改变，以#开头
2. history路由 基于html5规范，利用history.pushState || history.replaceState 来进行路由控制。

## JavaScript&DOM

### javascript放置位置

在 HTML 中，JavaScript 代码必须位于 <script> 与 </script> 标签之间。

head

body

把脚本置于 <body> 元素的底部，可改善显示速度，因为脚本编译会拖慢显示。

外置

外部 JavaScript 的优势

在外部文件中放置脚本有如下优势：

分离了 HTML 和代码

使 HTML 和 JavaScript 更易于阅读和维护

已缓存的 JavaScript 文件可加速页面加载

### JavaScript输出

使用 window.alert() 写入警告框

使用 document.write() 写入 HTML 输出

使用 innerHTML 写入 HTML 元素

使用 console.log() 写入浏览器控制台

注意：在 HTML 文档完全加载后使用 document.write() 将删除所有已有的 HTML

### ES6中let、const和var的区别

一，**var定义的变量，作用域是整个封闭函数，是全域的；let和const声明形成块作用域;作用域是在块级或者字块中；**

**二，变量提升**：不论通过var声明的变量处于当前作用于的第几行，都会提升到作用域的最顶部。而let声明的变量不会在顶部初始化，凡是在let声明之前使用该变量都会报错（引用错误ReferenceError）；**即var可以先使用再定义**

三，只要块级作用域内存在let，它所声明的变量就会绑定在这个区域；

**四，let和const不允许在相同作用域内重复声明**（报错同时使用var和let，两个let）。

const用来专门声明一个常量，它跟let一样作用于块级作用域，没有变量提升，重复声明会报错，

**五． const声明的常量不可改变，声明时必须初始化（赋值）当声明的是复合类数据时可以修改复合属性，即当前对象的指针不可变；**

**如const list = [];**

**list[0] = 10;**

### 会改变原数组的方法

push()

pop()

shift()

unshift()

splice()

sort()

reverse()

forEach()

### 不会改变原数组的方法，即返回新数组

filter()

concat()

slice()

map()

### 深拷贝与浅拷贝

浅拷贝（shallowCopy）只是增加了一个指针指向已存在的内存地址，深拷贝（deepCopy）是增加了一个指针并且申请了一个新的内存，使这个增加的指针指向这个新的内存， 使用深拷贝的情况下，释放内存的时候不会因为出现浅拷贝时释放同一个内存的错误。

### 防抖和节流

**防抖是控制触发次数，节流是控制执行频率**

防抖即在一定时间内只能执行一次的函数，**如果在该时间内又触发了该函数会重新计算时间**

**分为立即执行版和非立即执行版。**

非立即执行版的意思是触发事件后函数不会立即执行，而是在 n 秒后执行，如果在 n 秒内又触发了事件，则会重新计算函数执行时间。

例：content.onmousemove = debounce(count,1000);//触发后1s内执行一次

立即执行版的意思是触发事件后函数会立即执行，然后 n 秒内不触发事件才能继续执行函数的效果。

节流指能连续触发事件但是在 n 秒中只执行一次函数。节流会稀释函数的执行频率。

**对于节流，一般有两种方式可以实现，分别是时间戳版和定时器版。**

content.onmousemove = throttle(count,1000);//

时间戳版的函数触发是在时间段内开始的时候，而定时器版的函数触发是在时间段内结束的时候。

### 块作用域和函数作用域的区别

块作用域是ES6新添加的。块作用域由 { } 包括，if语句和for语句里面的{ }也属于块作用域。

### 如何让事件先冒泡后捕获

如果要实现先冒泡后捕获的效果，对于同一个事件，同时监听捕获和冒泡，分别对应相应的处理函数，监听到捕获事件，先暂缓执行，直到冒泡事件被捕获后再执行捕获之间。

### mouseover和mouseenter的区别

mouseover：**当鼠标移入元素或其子元素都会触发事件，所以有一个冒泡时重复触发的过程。**对应的移除事件是mouseout

mouseenter：**当鼠标移除元素本身（不包含元素的子元素）会触发事件**，也就是不会冒泡，对应的移除事件是mouseleave

### js的各种位置，比如clientHeight,scrollHeight,offsetHeight ,以及scrollTop, offsetTop,clientTop的区别？

clientHeight：表示的是可视区域的高度，不包含border和滚动条

offsetHeight：表示可视区域的高度，包含了border和滚动条

scrollHeight：表示了所有区域的高度，包含了因为滚动被隐藏的部分。

clientTop：表示边框border的厚度，在未指定的情况下一般为0

scrollTop：滚动后被隐藏的高度，获取对象相对于由offsetParent属性指定的父坐标(css定位的元素或body元素)距离顶端的高度。

### js拖拽功能的实现

首先是三个事件，分别是mousedown，mousemove，mouseup

当鼠标点击按下的时候，需要一个tag标识此时已经按下，可以执行mousemove里面的具体方法。

### JS的语言特性

运行在客户端浏览器上；

不用预编译，直接解析执行代码；

是弱类型语言，较为灵活；

与操作系统无关，跨平台的语言；

脚本语言、解释性语言

### this的指向

在非严格模式下使用 call 和 apply 时，如果用作 this 的值不是对象，则会被尝试转换为对象。null 和 undefined 被转换为全局对象。原始值如 7 或 'foo' 会使用相应构造函数转换为对象。因此 7 会被转换为 new Number(7) 生成的对象，字符串 'foo' 会转换为 new String('foo') 生成的对象。

当函数作为对象里的方法被调用时，this 被设置为调用该函数的对象。

如果该方法存在于一个对象的原型链上，那么 this 指向的是调用这个方法的对象

当一个函数用作构造函数时（使用new关键字），它的this被绑定到正在构造的新对象。

当函数被用作事件处理函数时，它的 this 指向触发事件的元素。

当代码被内联 on-event 处理函数 调用时，它的this指向监听器所在的DOM元素

### 获取所有节点

1. 获取根节点html
2. 获取html的所有子节点

### img加载失败时显示默认图片

<img src="图片的url地址" alt="图片XX" onerror="this.src='默认图片的url地址'"/>

除此之外，还可通过onerror设置各种img样式，方法等

# 问题

### Webpack

webpack是一个前端模块化方案，更侧重模块打包，我们可以把开发中的所有资源（图片、js文件、css文件等）都看成模块，通过loader（加载器）和plugins（插件）对资源进行处理，打包成符合生产环境部署的前端资源。

# HTML

## 元素

<html> HTML <html> 元素 表示一个HTML文档的根（顶级元素），所以它也被称为根元素。所有其他元素必须是此元素的后代。

<head> HTML head 元素 规定文档相关的配置信息（元数据），包括文档的标题，引用的文档样式和脚本等。

<body> 元素定义了 HTML 文档的主体。这个元素拥有一个开始标签 <body>，以及一个结束标签 </body>。

<link> HTML外部资源链接元素 (<link>) 规定了当前文档与外部资源的关系。该元素最常用于链接样式表，此外也可以被用来创建站点图标(比如PC端的“favicon”图标和移动设备上用以显示在主屏幕的图标) 。

<meta> HTML <meta> 元素表示那些不能由其它 HTML 元相关（meta-related）元素（(base、link, script、style 或 title）之一表示的任何Metadata信息。

<style> HTML的<style>元素包含文档的样式信息或者文档的部分内容。默认情况下，该标签的样式信息通常是CSS的格式。

<title> HTML <title> 元素 定义文档的标题，显示在Browser的标题栏或标签页上。它只应该包含文本，若是包含有标签，则它包含的任何标签都将被忽略。

<nav> HTML <nav>元素表示页面的一部分，其目的是在当前文档或其他文档中提供导航链接。导航部分的常见示例是菜单，目录和索引。

### 文本内容

<li> HTML <li> 元素 （或称 HTML 列表条目元素） 用于表示列表里的条目。它必须包含在一个父元素里：一个有序列表(ol)，一个无序列表(ul)，或者一个菜单 (menu)。在菜单或者无序列表里，列表条目通常用点排列显示；在有序列表里，列表条目通常在左边显示按升序排列的计数，例如数字或者字母。

<div> HTML <div> 元素 (或 HTML 文档分区元素) 是一个通用型的流内容容器，在不使用CSS的情况下，其对内容或布局没有任何影响。

<ol> HTML <ol> 元素表示有序列表，通常渲染为一个带编号的列表。

### 内联文本语义

**使用 HTML 内联文本语义（Inline text semantics）定义一个单词、一行内容，或任意文字的语义、结构或样式。**

<a> HTML <a> 元素（或称锚元素）可以创建通向其他网页、文件、同一页面内的位置、电子邮件地址或任何其他 URL 的超链接。

<b> HTML提醒注意（Bring Attention To）元素（<b>）用于吸引读者的注意到该元素的内容上（如果没有另加特别强调）。这个元素过去被认为是粗体（Boldface）元素，并且大多数浏览器仍然将文字显示为粗体。尽管如此，你不应将 <b> 元素用于显示粗体文字；替代方案是使用 CSS font-weight 属性来创建粗体文字。

<br> HTML <br> 元素在文本中生成一个换行（回车）符号。此元素在写诗和地址时很有用，这些地方的换行都非常重要。

<code> HTML <code> 元素呈现一段计算机代码. 默认情况下, 它以浏览器的默认等宽字体显示.

<i> HTML元素 <i> 用于表现因某些原因需要区分普通文本的一系列文本。例如技术术语、外文短语或是小说中人物的思想活动等，它的内容通常以斜体显示。

<span> HTML <span> 元素是短语内容的通用行内容器，并没有任何特殊语义。可以使用它来编组元素以达到某种样式意图（通过使用类或者Id属性），或者这些元素有着共同的属性，比如lang。应该在没有其他合适的语义元素时才使用它。<span> 与 div 元素很相似，但 div 是一个 块元素 而 <span> 则是 行内元素 .

<strong> Strong 元素 (<strong>)表示文本十分重要，一般用粗体显示。

### 图片和多媒体

<img> HTML <img> 元素将一份图像嵌入文档

<video> HTML <video> 元素 用于在HTML或者XHTML文档中嵌入媒体播放器，用于支持文档内的视频播放。

<audio> HTML <audio> 元素用于在文档中嵌入音频内容。 <audio> 元素可以包含一个或多个音频资源， 这些音频资源可以使用 src 属性或者source 元素来进行描述：浏览器将会选择最合适的一个来使用。也可以使用 MediaStream 将这个元素用于流式媒体。

### 内嵌内容

### 表格内容

### 表单元素

name属性主要是表单元素里才有的属性。与id都可以标识元素，但name可以重复，且

<button> HTML <button> 元素表示一个可点击的按钮，可以用在表单或文档其它需要使用简单标准按钮的地方。

<form> HTML <form> 元素表示文档中的一个区域，此区域包含交互控件，用于向 Web 服务器提交信息。

<input> HTML <input> 元素用于为基于Web的表单创建交互式控件，以便接受来自用户的数据; 可以使用各种类型的输入数据和控件小部件，具体取决于设备和user agent。

<label> HTML <label> 元素（标签）表示用户界面中某个元素的说明。

<output> HTML <output> 标签表示计算或用户操作的结果。

HTML 提示：使用小写标签

HTML 标签对大小写不敏感：<P> 等同于 <p>。许多网站都使用大写的 HTML 标签。

## HTML5

### 新元素

<article> 定义文档内的文章。

<footer> 定义文档或节的页脚。

<header> 定义文档或节的页眉。

<audio> 定义音频内容

<video> 标签定义视频，比如电影片段或其他视频流

<canvas> 定义图形，比如图表和其他图像,标签只是图形容器，您必须使用脚本来绘制图形

拖放特性：拖放是一种常见的特性，即抓取对象以后拖到另一个位置。在 HTML5 中，拖放是标准的一部分，任何元素都能够拖放。

新的 Input 类型 date，number，email。

# CSS

## 伪元素

CSS 伪元素用于设置元素指定部分的样式。

例如，它看用于：

设置元素的首字母、首行的样式

在元素的内容之前或之后插入内容

采用双引号：：

## 伪类

伪类用于定义元素的特殊状态。

例如，它可以用于：

设置鼠标悬停在元素上时的样式

为已访问和未访问链接设置不同的样式

设置元素获得焦点时的样式

采用单引号：

## CSS组合器

组合器是解释选择器之间关系的某种机制。

CSS 选择器可以包含多个简单选择器。在简单选择器之间，我们可以包含一个组合器。

CSS 中有四种不同的组合器：**后代选择器 (空格)、子选择器 (>)、相邻兄弟选择器 (+)、通用兄弟选择器 (~)**

### 后代选择器

后代选择器匹配属于指定元素后代的所有元素。

下面的例子选择 <div> 元素内的所有 <p> 元素：

div p {

background-color: yellow;

}

### 子选择器

子选择器匹配属于指定元素子元素的所有元素。

下面的例子选择属于 <div> 元素子元素的所有 <p> 元素：

div > p {

background-color: yellow;

}

### 相邻兄弟选择器

相邻兄弟选择器匹配所有作为指定元素的相邻同级的元素。

兄弟（同级）元素必须具有相同的父元素，“相邻”的意思是“紧随其后”。

下面的例子选择紧随 <div> 元素之后的所有 <p> 元素：

div + p {

background-color: yellow;

}

注：只有一个兄弟被选择

### 通用兄弟选择器

通用兄弟选择器匹配属于指定元素的同级元素的所有元素。

下面的例子选择属于 <div> 元素的同级元素的所有 <p> 元素：

div~p{}

## CSS外边距

CSS margin 属性用于在任何定义的边框之外，为元素周围创建空间。

通过 CSS，您可以完全控制外边距。有一些属性可用于设置元素每侧（**上、右、下和左**）的外边距。

**auto 值**您可以将 margin 属性设置为 auto，以使元素在其容器中水平居中。

然后，该元素将占据指定的宽度，并且剩余空间将在左右边界之间平均分配。

### 外边距合并

外边距合并指的是，当两个**垂直外边距**相遇时，它们将形成一个外边距。

合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者。

假设有一个空元素，它有外边距，但是没有边框或填充。在这种情况下，上外边距与下外边距就碰到了一起，它们会发生合并。**这就是一系列的段落元素占用空间非常小的原因**，因为它们的所有外边距都合并到一起，形成了一个小的外边距。

## CSS内边距

### 内边距和元素宽度

CSS width 属性指定元素内容区域的宽度。内容区域是元素（盒模型）的内边距、边框和外边距内的部分。

因此，如果元素拥有指定的宽度，则添加到该元素的内边距会添加到元素的总宽度中。这通常是不希望的结果。即总宽度=元素指定宽度+2\*内边距

若要元素指定宽度保持不变，无论填充量如何，那么您可以使用 **box-sizing：border-box** 属性。这将导致元素保持其宽度。如果增加内边距，则可用的内容空间会减少。

# JAVASCRIPTE

## js隐式转换数据

· 1.1 隐式转换介绍

· 1.2 隐式转换规则

· 1.3 坑一：字符串连接符与算术运算符隐式转换规则混淆

· 1.4 坑二：关系运算符：会把其他数据类型转换成number之后再比较关系

· 1.5 坑三：复杂数据类型在隐式转换时会先转成String，然后再转成Number运算

· 1.6-坑四：逻辑非隐式转换与关系运算符隐式转换搞混淆

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 值 | 转换为字符串 | 转换为数字 | 转换为布尔值 | 转换为对象 |
| undefined | “undefined” | NaN | false | throw TypeError |
| null | “null” | 0 | false | throw TypeError |
| true | “true” | 1 |  | new Boolean(“true”) |
| false | “false” | 0 |  | new Boolean(“false”) |
| “” |  | 0 | false | new String("") |
| “1.2” |  | 1.2 | true | new String(“1.2”) |
| “1.2a” |  | NaN | true | new String(“1.2a”) |
| “aaa” |  | NaN | true | new String(“aaa”) |
| 0 | “0” |  | false | new Number(0) |
| -0 | “0” |  | false | new Number(-0) |
| 1 | “1” |  | true | new Number(1) |
| NaN | “NaN” |  | false | new Number(NaN) |
| Infinity | “Infinity” |  | true | new Number(Infinity) |
| [] | “” | 0 | true |  |
| [9] | “9” | 9 | true |  |
| [“a”, “b”] | “a,b” | NaN | true |  |

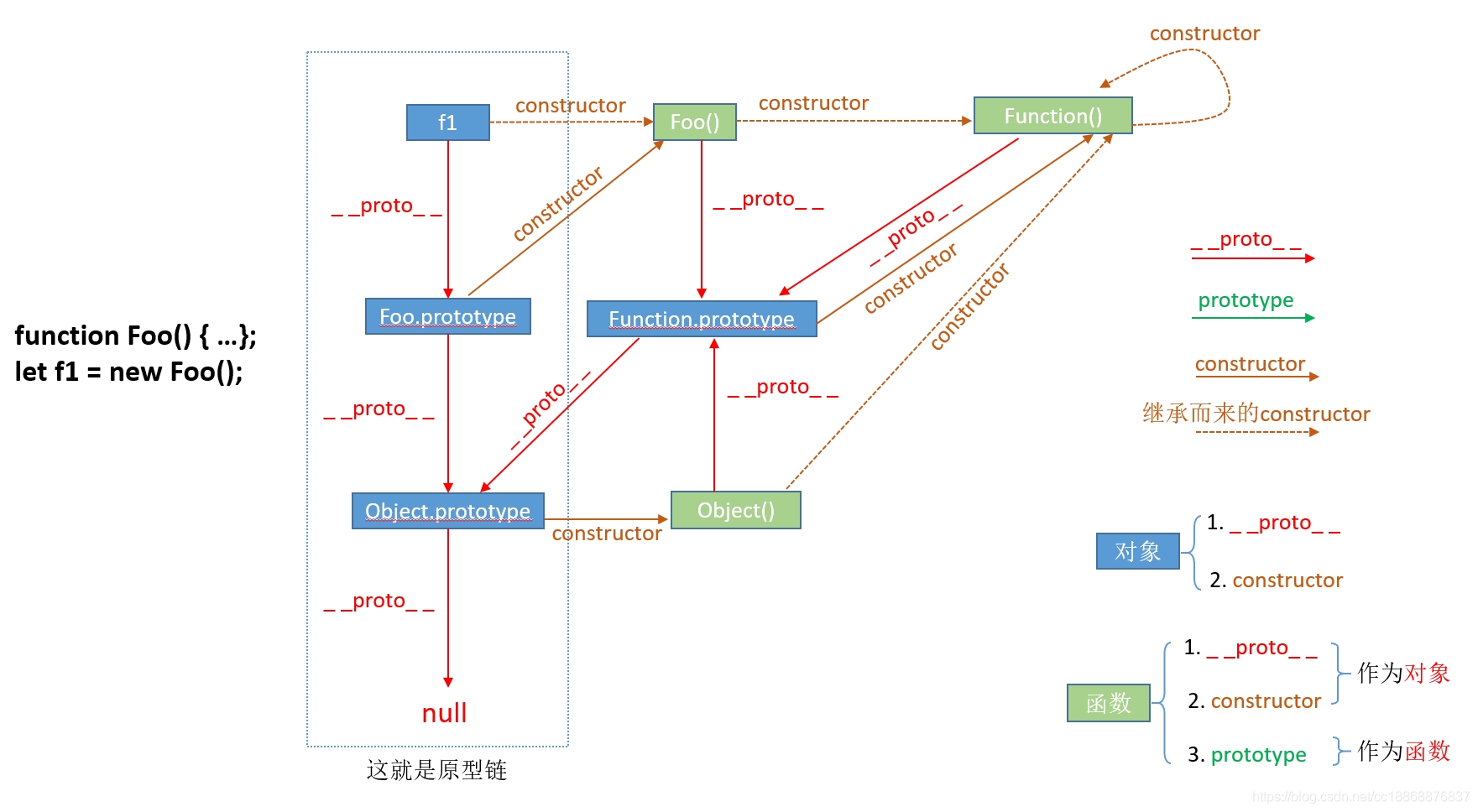
## prototype、constructor、\_\_proto\_\_

①\_\_proto\_\_和constructor属性是对象所独有的；② prototype属性是函数所独有的，因为函数也是一种对象，所以函数也拥有\_\_proto\_\_和constructor属性。

\_\_proto\_\_属性的作用就是当访问一个对象的属性时，如果该对象内部不存在这个属性，那么就会去它的\_\_proto\_\_属性所指向的那个对象（父对象）里找，一直找，直到\_\_proto\_\_属性的终点null，再往上找就相当于在null上取值，会报错。通过\_\_proto\_\_属性将对象连接起来的这条链路即我们所谓的原型链。

prototype属性的作用就是让该函数所实例化的对象们都可以找到公用的属性和方法，即f1.\_\_proto\_\_ === Foo.prototype。

constructor属性的含义就是指向该对象的构造函数，所有函数（此时看成对象了）最终的构造函数都指向Function。



## JS继承的实现方式

每个构造函数都有一个prototype属性，指向函数的原型对象；原型对象中又有一个constructor属性，重新指向构造函数。

### 原型链继承

**核心：** 通过prototype将父类的实例设为子类的原型

**优点：**非常纯粹的继承关系，实例是子类的实例，也是父类的实例

父类新增原型方法/原型属性，子类都能访问到

简单，易于实现

**缺点：**要想为子类新增属性和方法，必须要在创建子类之后执行，不能放到子类的构造器中

且无法实现多继承。

来自原型对象的所有属性被所有实例共享（来自原型对象的引用属性是所有实例共享的）

创建子类实例时，无法向父类构造函数传参

### 构造继承

**核心：通过call()函数等构造函数获取父类给子类**

使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

优点：能多继承，可传参

缺点：无法函数复用，每次都会新建方法。

### 组合继承

**核心：**结合构造继承与原型链继承。**借用构造函数来继承属性，原型链来继承方法。由于是先原型链，所以构造时又重新覆盖了原型的属性。**

### 拷贝继承（原型式继承）

function object(o){

function F(){};

F.prototype = o;

return new F();

}

以一个对象为基础，通过object()函数进行浅复制，再将得到的对象实例加以修改。

特点：支持多继承

缺点：改动一个实例的引用类型则所有实例的该引用类型改变。

### 寄生继承

**核心：**封装原型式继承，并在原型式继承返回实例后在实例上修改。最后返回实例

例：

function createAnother(original){

var clone = object(original);//object()函数创建对象

clone.sayHi = function(){ //增强这个对象

alert("hi");

};

return clone; //返回这个对象

}

### 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

## 闭包

一个函数和对其周围状态（lexical environment，词法环境）的引用捆绑在一起（或者说函数被引用包围），这样的组合就是闭包（closure）。也就是说，**闭包让你可以在一个内层函数中访问到其外层函数的作用域。**在 JavaScript 中，每当创建一个函数，闭包就会在函数创建的同时被创建出来。闭包包含在函数创建时作用域中的所有变量。

**函数在定义时的词法作用域以外的地方被调用就会形成闭包**。闭包使得函数可以继续访问定义时的词法作用域

## 原型链

每个实例对象（ object ）都有一个私有属性（称之为 \_\_proto\_\_ ）指向它的构造函数的原型对象（prototype ）。该原型对象也有一个自己的原型对象( \_\_proto\_\_ ) ，**层层向上直到一个对象的原型对象为 null**。根据定义，null 没有原型，并作为这个原型链中的最后一个环节。

## 作用域

**每个函数都有自己独立的执行环境，在代码完成的时候就确定了自己的作用域。**

执行函数前，会对函数中的关键字var和function进行预解析。就是对变量和函数的声明提前，此时不会给变量赋值，当函数与变量同名时，函数会覆盖掉变量；待变量执行了之后，那个名字才会代表变量。

**作用域链**：当所需要的变量在所在的作用域中查找不到的时候，它会一层一层向上查找，直到找到全局作用域还没有找到的时候，就会放弃查找。这种一层一层的关系，就是作用域链。

# JavaScript单线程和异步

JavaScript是单线程的。

任务分为宏任务和微任务，

微任务即当前正在处理的任务

宏任务按照任务队列依次将任务传进微任务处理

遇到new Promise()的需要立即执行

async相当于一个promise的语法糖，但不会立即执行，而是等到使用后才执行。await会执行完当前语局并将其后的任务存入微任务队列，then里面的函数则是存放进微任务队列，等待执行

requestAnimationFrame()指的是下一次浏览器重绘之前执行的函数，在settimeout之前执行

settimeout是在任务队列清空之后才开始执行，settimeout之间主要看设定时间参数来先后输出，其次才是context的先后顺序。

## yield与生成器

生成器为一种可以用来控制迭代器（iterator）的函数，它可以随时暂停，并可以在任意时候恢复。

创建生成器：在函数关键词后加\*，如function \* generator () {}、let generator = function \* () {}。

yield就是生成器中独特的return，只会返回一次，再次调用时继续执行函数并返回下一个yield。其返回的是一个对象，有两个属性，value和done，value代表应该返回的值，done代表所有yield是否执行完毕，若是则done=true；

例：function \* generator() {

yield 5;

}

const gen = generator();

gen.next(); // {value: 5, done: false}

gen.next(); // {value: undefined, done: true}，之后再执行下一步还是返回同样的对象。

当然也可以用return返回对象，但return之后的将不再执行。

yield 委托迭代：yield加\*可将它的工作委托给另一个生成器。通过这种方式，你就能将多个生成器连接在一起。

yield\* 是一个表达式，不是语句，所以它会有自己的值。

yield\* g1();//执行并获取g1()中的yield结果，

yield也可以在 next() 方法调用后返回传递的值

例：gen.next('A'); // {value: "A", done: false} 执行了 yield（yield）；代码

我们不仅可以用 next() 来迭代生成器，还可以用 for of 循环来一次得到生成器所有的值**（而不是对象）。**

例：function \* generator(arr) {

for (const el in arr)

yield el;

}

const gen = generator([0, 1, 2]);

for (const g of gen) {

console.log(g); // 0 -> 1 -> 2

}

## Async

ES7提供的async函数相当于generator函数的语法糖(即高效简化版)

创建语法：async function(){};//相当于把function后的\*换成function前async，将yield替换成await。

最终会返回成一个promise。

1. async函数内部return语句返回的值，会成为then方法回调函数的参数。
2. 只有async函数内部的异步操作执行完，才会执行then方法指定的回调函数。
3. 正常情况下，await命令后面是一个Promise对象。如果不是，会被转成一个立即resolve的Promise对象。
4. 只要一个await语句后面的Promise变为reject，那么整个async函数都会中断执行。

### await

await的语句执行完才执行后面的语句

# JavaScript 操作http

## fetch

fetch()方法提供了一种简单，合理的方式来跨网络异步获取资源。取代原生的XMLHttpRquest

fetch接受两个参数，url和init，其中url参数是必须的，而init参数是可选的, init是一个对象，在里面可以对这个请求进行配置，例如设置请求方法(默认GET方法)设置请求头等；如果不传入init参数，将会采用默认的配置.

返回一个包含Promise对象，在这个对象的resolve方法中可以访问到请求的结果，是一个Response对象。在之后的then中对response进行进一步操作

# javascript 模块（modules）

# 事件传播

一般情况下，事件会从根元素往下传播直到目标元素，这个阶段称为捕捉。到达目标元素之后触发事件，称为目标阶段，之后事件再从目标元素传导到根元素，称为冒泡阶段。事件的默认触发机制为冒泡触发。

### event.stopPropagation()

阻止事件在DOM中继续传播，即取消进一步的事件捕获或冒泡，防止再触发定义在别的节点上的监听函数，但是不包括在当前节点上新定义的事件监听函数。

event.target //事件触发的起源对象

event.currentTarget //事件触发的当前对象

event.srcElement //event.target的IE版本

### MutationObserver(callback())

在指定的DOM发生变化时被调用。

例：// 创建一个观察器实例并传入回调函数

const observer = new MutationObserver(callback());

// 以上述配置开始观察目标节点

observer.observe(targetNode, config);

## 事件冒泡

事件冒泡是HTML DOM API中事件传播的一种

方式，当一个事件发生在另一个元素中的一个元素中，并且两个元素都注册了该事件的句柄时。通过冒泡，事件首先由最内部的元素捕获和处理，然后传播到外部元素。执行从该事件开始，并转到其父元素。然后执行传递给父元素，以此类推，直到body元素。

## 事件捕捉

当鼠标点击或者触发dom事件时，浏览器会从根节点开始由外到内进行事件传播，即点击了子元素，如果父元素通过事件捕获方式注册了对应的事件的话，会先触发父元素绑定的事件。

## 事件监听

addEventListener()方法，用于向指定元素添加事件句柄，它可以更简单的控制事件，语法为

element.addEventListener(event, function, useCapture);

第一个参数是事件的类型(如 "click" 或 "mousedown").

第二个参数是事件触发后调用的函数。

第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。**false冒泡，true捕获，默认false。在DOM标准事件模型中，是先捕获后冒泡。**

## 事件委托

事件委托指的是，不在事件的发生地（直接dom）上设置监听函数，而是在其父元素上设置监听函数，通过事件冒泡，父元素可以监听到子元素上事件的触发，通过判断事件发生元素DOM的类型，来做出不同的响应。

好处：比较合适动态元素的绑定，新添加的子元素也会有监听函数，也可以有事件触发机制。

## 事件触发

当一个事件传播到目标元素上并且被监听到时，产生一个消息进入消息队列；

当事件被连续触发时，按照同步先执行完事件1再zhix1事件2

## 事件循环(Event Loop)

JS会首先判断代码是同步还是异步，同步进入主线程，异步进入任务队列；

同步任务进入主线程后一直执行，直到主线程空闲后，才会去任务队列中查看是否有可执行的异步任务，如果有就推入主线程中执行；

**即：主线程是微任务，任务队列是宏任务**

事件循环是一个先进先出（FIFO）队列，这说明回调是按照它们被加入队列的顺序执行的。