# Css

###### 盒模型

盒模型的的组成：内容 边框(border) 内边距(padding) 外边距(margin)。

###### Flex

弹性布局

###### css的单位

常见的有： % 百分比 px 像素 1em 等于当前的字体尺寸 2em等于当前字体尺寸的两倍。

###### Css的常见选择器

标签选择器 类选择器 ID选择器 ID选择器一般是给js使用

后代选择器 子选择器 并集选择器 序选择器

伪类选择器 的4种状态 a:link,默认状态 a:visited,被访问过

a:hover,鼠标悬停时 a:active,鼠标点击时 一般情况下遵循爱恨原则love hate＝ lvha。

###### BFC清除浮动

1：用伪类选择器 .clearfix:after{ content: ""; clear: both; display: block; height: 0px; visibility: hidden; } .clearfix{ \*zoom:1; }

2：给父元素添加一个:display:inline-block 3：父元素也添加一个浮动

2和3的弊端 无法使用margin

4：给父元素增加高

###### 常见页面布局

1. 固定布局 2.流式布局（百分比布局） 3.弹性布局（伸缩布局）

###### 响应式布局

1. 就是一个网站能够兼容多个终端
2. 过程

* 在html的中添加媒体查询和相关配置
* 构想媒体查询的节点尺寸
* 添加媒体查询
* 提炼css中的公用代码，放在公共的index.css中，直接引入，然后针对不同断点尺寸新建不同的css，

1. css3新特性

创建动画的步骤：给需要执行动画的元素指定动画的名称 创建指定名称的动画 给需要执行动画的元素指定动画的执行时间

animation：动画的名称 动画的执行时间 动画的速度 延迟时间 动画的执行次数 是否执行往返动画; 简写：animation：all 3s。

添加过渡的步骤：布局页面 触发修改属性 给修改属性添加过渡效果

transition：需要过渡的属性 执行过渡的时间 执行过渡的速度 延迟时间,需要过渡的属性 执行过渡的时间 执行过渡的速度 延迟时间. 简写：transition:all 3s

2D旋转：transform 3D转化：在父元素上添加 transform-style: preserve-3d;

###### display哪些取值

none 此元素不会被显示，并且不占据页面空间

inline 行内元素 元素会在一行内显示，超出屏幕宽度自动换行，不能设置宽度和高度，元素的宽度和高度只能是靠元素内的内容撑开。

block 块级元素 会独占一行，如果不设置宽度，其宽度会自动填满父元素的宽度，可以设置宽高，即使设置了宽度，小于父元素的宽度，块级元素也会独占一行。

inline-block 行内块元素 与行内元素一样可以再一行内显示，而且可以设置宽高，可以设置margin和padding。

###### 相邻的两个inline-block节点为什么会出现间隔，该如何解决

1：改变书写方式 2：font-size:0

3： 使用margin设置负值 4：使用word-spacing

###### meta viewport 移动端适配

概念：通常viewport是指视窗、视口，浏览器上用来显示网页的那部分区域

如何设置理想视口代码：

<meta name="viewport"content="width=device-width,user-scalable=no,initial-scale=1.0, maximum-scale=1.0,minimum-scale=1.0">

###### rem布局的优缺点

rem是相当于html根元素来判断的大小，屏幕变化的时候不是很好控制，用百分比了话，可以根据屏幕的大小，占比多少，相对比较好控制

###### 1像素边框问题

1. 可以用box-shadow 阴影来实现
2. viewport + rem 方案
3. 设置 border-image 方案

# HTML

###### 语义化

就是用合理、正确的标签来展示内容

易于用户阅读，样式丢失的时候能让页面呈现清晰的结构。

方便其他设备解析，如盲人阅读器根据语义渲染网页

有利于开发和维护，语义化更具可读性，代码更好维护，与CSS3关系更和谐。

###### 新标签的特性

用于绘画的 canvas 元素

用于媒介回放的 video 和 audio 元素

对本地离线存储的更好的支持

新的特殊内容元素，比如 article、footer、header、nav、section

新的表单控件，比如 calendar、date、time、email、url、search

.详细语义（兼容性好的标签）www

header:页眉（网页（部分区域）的头部 顶部 导航区域等等）;

footer:页脚（网页（部分区域）的底部|版权区域等等）;

section 标签定义网页中的区域（部分）；

article 内容是引用其他地方的；

aside 跟 article 是一起使用；是辅助 article 区域的内容;

nav 导航链接部分;

###### svg canvas

SVG(Scalable Vector Graphics)可缩放矢量图形,一种用来描述二维矢量图形的XML语言

代码格式如：

<svg width='800' height='600' viewBox='0 0 80 60'>

<rect >

</svg>

Canvas是一个画布,用来在网页上绘制图像

<canvas></canvas>

可以通过网上查找

#### input和textarea的区别

Input中的text是单行文本框，textarea是多行文本框。

###### 5.用一个div模拟textarea的实现

给div添加contenteditable=true即可

1. 移动设备忽略将页面中的数字识别为电话号码的方法

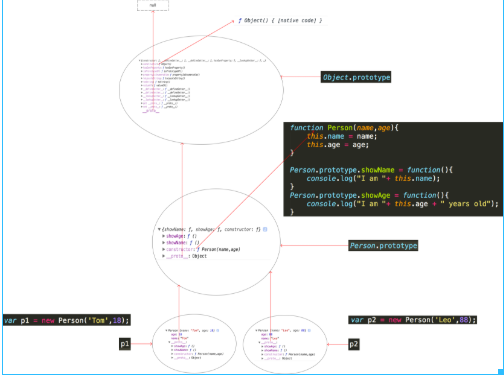
代码如下：<meta name = "format-detection" content = "telephone=no">

# JS

###### 原型/原型链/构造函数/实例/继承

原型：原型就是一个对象，所有的实例共享对象上的方法和属性

原型链：实例对象的\_\_proto\_\_对象-->构造函数的原型对象，构造函数的原型对象的\_\_proto\_\_-->Object的原型对象，Object的原型对象的\_\_proto\_\_ -->null

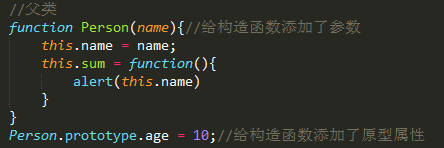


构造函数： 构造函数的首字母必须大写，用来区分于普通函数

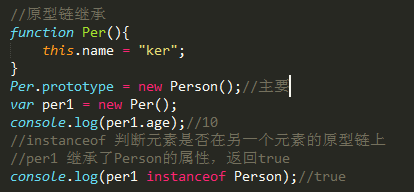
　　　　 内部使用的this对象，来指向即将要生成的实例对象

　　　　 使用New来生成实例对象

继承：　想要继承，就必须要提供个父类（继承谁，提供继承的属性）



**一、原型链继承**



　　　　重点：让新实例的原型等于父类的实例。

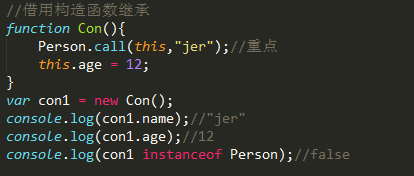
　　　　特点：1、实例可继承的属性有：实例的构造函数的属性，父类构造函数属性，父类原型的属性。（新实例不会继承父类实例的属性！）

　　　　缺点：1、新实例无法向父类构造函数传参。

　　　　　　　2、继承单一。

　　　　　　　3、所有新实例都会共享父类实例的属性。（原型上的属性是共享的，一个实例修改了原型属性，另一个实例的原型属性也会被修改！）

**二、借用构造函数继承**

****

重点：用.call()和.apply()将父类构造函数引入子类函数（在子类函数中做了父类函数的自执行（复制））

　　　　特点：1、只继承了父类构造函数的属性，没有继承父类原型的属性。

　　　　　　　2、解决了原型链继承缺点1、2、3。

　　　　　　　3、可以继承多个构造函数属性（call多个）。

　　　　　　　4、在子实例中可向父实例传参。

　　　　缺点：1、只能继承父类构造函数的属性。

　　　　　　　2、无法实现构造函数的复用。（每次用每次都要重新调用）

　　　　　　　3、每个新实例都有父类构造函数的副本，臃肿。

**三、组合继承（组合原型链继承和借用构造函数继承）（常用）**



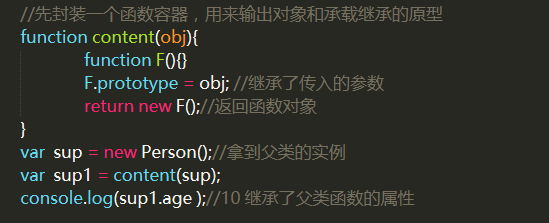
　　　　重点：**结合了两种模式的优点，传参和复用**

　　　　特点：1、可以继承父类原型上的属性，可以传参，可复用。

　　　　　　　2、每个新实例引入的构造函数属性是私有的。

　　　　缺点：调用了两次父类构造函数（耗内存），子类的构造函数会代替原型上的那个父类构造函数。

**四、原型式继承**

****

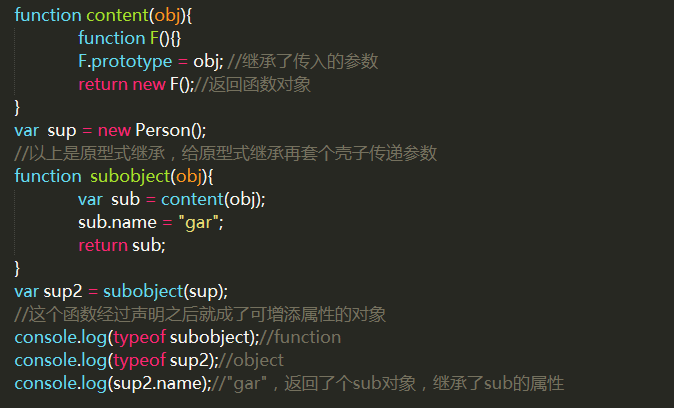
　　重点：用一个函数包装一个对象，然后返回这个函数的调用，这个函数就变成了个可以随意增添属性的实例或对象。object.create()就是这个原理。

　　　　特点：类似于复制一个对象，用函数来包装。

　　　　缺点：1、所有实例都会继承原型上的属性。

　　　　　　　2、无法实现复用。（新实例属性都是后面添加的）

**五、寄生式继承**

****

　　　　重点：就是给原型式继承外面套了个壳子。

　　　　优点：没有创建自定义类型，因为只是套了个壳子返回对象（这个），这个函数顺理成章就成了创建的新对象。

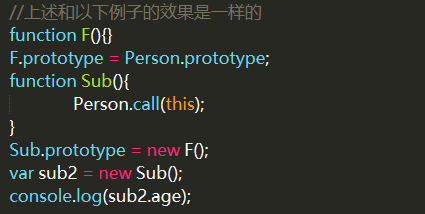
　　　　缺点：没用到原型，无法复用。

**六、寄生组合式继承（常用）**

寄生：在函数内返回对象然后调用

　　　　组合：1、函数的原型等于另一个实例。2、在函数中用apply或者call引入另一个构造函数，可传参





　　　　重点：修复了组合继承的问题

###### Arguments

****arguments****此对象大多用来针对同个方法多处调用并且传递参数个数不一样时进行使用。根据****arguments****的索引来判断执行的方法。

使用方法：

虽然argu对象不是一个数组，但是访问单个参数的方式和数组的方式相同、

###### 数据类型判断

基本数据类型： typeof 可以判断

Undefined 未定义 ：未定义类型只有一个值undefined,这个值的意思是变量没有定义 未定义有两种情况,一个是真的没有定义,一个是定义了但是没有赋值

Null 空对象：空对象类型只有一个值null,代表一个空对象指针

Number 数字类型 ：用来表示数字

String 字符串类型 ：用单引号或者双引号括起来的字符序列

Boolean 布尔类型 ：布尔型也被称为逻辑值类型或者真假值类型

所有类型的值都可以表示布尔值：真:true,非零数字,非空字符,非空对象 假:false,数字零,空字符,null,undefine

Undefined和Null是一种数据类型,而undefined和null是值,是一个常量

引用数据类型 ：object对象

#### **作用域链、闭包、作用域**

作用域：全局变量 和 局部变量

全局变量：定义在所有函数外面,所有的函数都可以使用的变量

局部变量：定义在函数的内部,只能在函数内部或者函数的子函数中使用

闭包：有权访问另一个函数作用域中变量的函数,即子函数有权访问父函数中的变量

作用域链：当代码在一个环境中执行时，会创建变量对象的一个作用域链（作用域形成的链条）

作用域链的前端，始终都是当前执行的代码所在环境的变量对象

作用域链中的下一个对象来自于外部环境，而在下一个变量对象则来自下一个外部环境，一直到全局执行环境

全局执行环境的变量对象始终都是作用域链上的最后一个对象

内部环境可以通过作用域链访问所有外部环境，但外部环境不能访问内部环境的任何变量和函数。

var n = 10;

function outer(){

function inner(){

function center(){

console.log(n);

}

center();

}

inner();

var n = 15;

}

outer(); //=> undefined

###### 原始ajax写法

get方式 ：

function btnClick() {

//创建核心对象

xmlhttp = null;

if (window.XMLHttpRequest) {// code for Firefox, Opera, IE7, etc.

xmlhttp = new XMLHttpRequest();

} else if (window.ActiveXObject) {// code for IE6, IE5

xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

}

//编写回调函数

xmlhttp.onreadystatechange = function() {

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

alert(xmlhttp.responseText)

}

}

//open设置请求方式和请求路径

xmlhttp.open("get", "/Ajax/ajax2?username=张三");//一个servlet，后面还可以写是否同步

//send 发送

xmlhttp.send();

}

Post方式：

function btnClick() {

//创建核心对象

xmlhttp = null;

if (window.XMLHttpRequest) {// code for Firefox, Opera, IE7, etc.

xmlhttp = new XMLHttpRequest();

} else if (window.ActiveXObject) {// code for IE6, IE5

xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

}

//编写回调函数

xmlhttp.onreadystatechange = function() {

/\* alert(xmlhttp.readyState); \*/

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

alert(xmlhttp.responseText)

}

/\* alert(123); \*/

}

//open设置请求方式和请求路径

xmlhttp.open("post", "/Ajax/ajax2");//一个servlet，后面还可以写是否同步

//设置请求头

xmlhttp.setRequestHeader("content-type", "application/x-www-form-urlencoded")

//send 发送

xmlhttp.send("username=张三");

}

###### JQuery发送ajax的方法

$.ajax({

url:'./data.json', //请求的地址

type:'GET',//指定请求的方法,默认是GET

data:{name:'Tom',age:18},//发送到服务器的数据，字符串键值对或者对象

dataType:'json',//返回数据的类型,可以是xml json script text html,默认是text

success:function(返回数据,状态信息,jqXHR对象){},//请求成功后的回调函数

error:function(jqXHR对象,状态信息){},//请求失败后的回调函数

complete:function(jqXHR对象,状态信息){},//请求完成后的回调函数

timeout:3000, //设置请求超时的时间,单位为毫秒

statusCode:{304:function(){},503:function(){}.....}//设置不同状态码的回调函数

headers:{test:'test'}//设置请求头

)}

###### 图片懒加载、预加载

图片懒加载：

**意义：** 懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

**实现方式：**

       1.第一种是纯粹的延迟加载，使用setTimeOut或setInterval进行加载延迟.

    2.第二种是条件加载，符合某些条件，或触发了某些事件才开始异步下载。

    3.第三种是可视区加载，即仅加载用户可以看到的区域，这个主要由监控滚动条来实现，一般会在距用户看到某图片前一定距离遍开始加载，这样能保证用户拉下时正好能看到图片。

图片预加载：

预加载就是提前加载图片，当用户需要查看时可直接从本地缓存中渲染

可以提高用户的体验

###### 常用方法：常用的是new Image();设置其src来实现预载，再使用onload方法回调预载完成事件。

###### this关键字

this关键字代表自身，在程序中主要的用途

1.用this在自身的构造方法内部调用其他的构造方法

2.用this代表自身类的对象

—直接使用this

—使用this引用成员变量

—使用this调用成员方法

this的值是当前对象的引用，只能用它来调用属于当前对象的方法或者使用this引用成员变量和局部变量重名

###### 为什么会有同源策略

设置同源限制主要是为了安全，如果没有同源限制存在浏览器中的cookie等其他数据可以任意读取，不同域下DOM任意操作，ajax任意请求的话如果浏览了恶意网站那么就会泄漏这些隐私数据

###### 怎么判断两个对象是否相等

利用JSON.stringify（obj1）===JSON.stringify（obj2）就可以进行判断，但是该方法存在一定的问题，当对象属性的顺序不同时，将失效

###### 事件

事件委托、代理：

好处：减少绑定DOM事件

动态添加的DOM元素也可以有事件处理

<button id="btn">添加</button>

<ul id="ul1">

<li>li1<a href="javascript:;" class="del">删除 </a></li>

</ul>

**var oUl = document.getElementById('ul1');**

**var oBtn = document.getElementById('btn');**

**oBtn.addEventListener('click',function(){**

**var oLi = document.createElement('li');**

**oLi.innerHTML = 'new data';**

**var oA = document.createElement('a');**

**oA.innerHTML = '删除';**

**oA.href = "javascript:;";**

**oA.className = 'del';**

**oLi.appendChild(oA);**

**oUl.appendChild(oLi);**

**},false);**

**oUl.addEventListener('click',function(ev){**

**if(ev.target.className == 'del'){**

**oUl.removeChild(ev.target.parentNode);**

**}**

**},false)**

阻止默认行为：

**event.preventDefault() 还有return false**

阻止默认冒泡：

****event.stopPropagation()****

###### 11.window.onload事件和domcontentloaded的区别

**Window.onload 事件触发时，页面上所有的DOM，样式表，脚本，图片，flash都已经加载完成了。**

**DOMContentLoaded 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash**

**12 .for...in迭代和for...of有什么区别**

推荐在循环对象属性的时候，使用for...in， 在遍历数组的时候的时候使用for...of。

for...in循环出的是key，for...of循环出的是value

1. **call apply区别，原生实现bind。**

apply()的第一个参数是null，在非严格模式下，第一个参数为null或者undefined时会自动替换为指向全局对象，apply()的第二个参数为数组或类数组。

call()作用和apply()一样，同样可实现继承，唯一的区别就在于call()接收的是参数列表，而apply()则接收参数数组。

bind()的作用与call()和apply()一样，都是可以改变函数运行时上下文，区别是**call()和apply()在调用函数之后会立即执行，而bind()方法调用并改变函数运行时上下文后，返回一个新的函数，供我们需要时再调用。**

1. **数组问题**

**1：数组去重**

###### function uniq(array){

###### var temp = []; //一个新的临时数组

###### for(var i = 0; i < array.length; i++){

###### if(temp.indexOf(array[i]) == -1){

###### temp.push(array[i]);

###### }

###### }

###### return temp;

###### }

###### var aa = [1,2,2,4,9,6,7,5,2,3,5,6,5];

###### console.log(uniq(aa));

###### 2.数组排序

var arr1 = [3,8,10,20,2,11];

console.log('1::',arr1);

arr1.sort(function(n1,n2){

return n2 - n1;

});

console.log('1::',arr1);

###### 数组常用的方法

1. join() （数组转字符串）

数组转字符串，方法只接收一个参数：即默认为逗号分隔符（）。

1. push()和pop()（数组尾操作）

push()：方法可向数组的末尾添加一个或多个元素，并返回新的长度。

pop()：方法用于删除并返回数组的最后一个元素。

1. shift() 和 unshift()（数组首操作）

shift()：方法用于把数组的第一个元素从其中删除，并返回第一个元素的值。

unshift()：方法可向数组的开头添加一个或更多元素，并返回新的长度。

5. .sort()（排序）

方法用于对数组的元素进行排序。

6 .concat() （连接两个或多个数组）

concat() 方法用于连接两个或多个数组。该方法不会改变现有的数组，而仅 仅会返回被连接数组的一个副本。在没有给 concat()方法传递参数的情况下，它只是复制当前数组并返回副本。

7. splice() （数组更新）

splice() 方法向/从数组中添加/删除项目，然后返回被删除的项目。（该方法会改变原始数组）

##### 遍历

##### arr.forEach(function(value,index){

##### console.log(value);

##### })

##### 遍历不会生成新的数组,没有返回值

##### 映射

##### var newArr = arr.map(function(value,index){

##### return value\*10;

##### })

##### 遍历数组,把函数的每一次返回值映射成新的数组并返回

##### 过滤

##### var newArr = arr.filter(function(value,index){

##### return value > 10;

##### })

##### 遍历数组,把函数的每一次返回值为真的原数组中的值组成新的数组并返回

##### 全真判断

##### var bRes = arr.every(function(value,index){

##### return value > 10;

##### })

##### 遍历数组,所有值为真的情况下返回真,否则返回假

##### 有真判断

##### var bRes = arr.some(function(value,index){

##### return value > 10;

##### })

##### 遍历数组,有一个值为真的情况下返回真,否则返回假

1. 设计模式之一 单例模式

1：优缺点：优点：节省内存，加快对象访问速度，在内存中只有一个对象，节省内存空间；因此对象需要被公用的场合适合使用，如多个模块使用同一个数据源连接对象等等  
2 缺点：不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。

1. BOM属性对象方法

BOM(浏览器对象模型),提供用于访问浏览器的对象,这些对象和网页内容无关.

**window对象：**

窗口位置

window.screenLeft/window.screenTop 窗口相对于屏幕左边/顶部的距 离(IE Chrome Safari Opera)

window.screenX/window.screenY 窗口相对于屏幕左边/顶部的距离(FireFox)

窗口大小

浏览器视口宽高(不包含浏览器的边框,如果有滚动条则包括)

window.innerWidth

window.innerHeight

浏览器窗口整个自身的宽高(包含浏览器边框)

window.outerWidth

window.outerHeight

边框包括工具栏,侧边栏,窗口镶边和调正窗口大小的边框。

**window.open**(地址,打开方式);作用:打开一个新窗口并返回新打开的窗口对象

"\_blank",空白页面打开,默认值

"\_self",当前窗口打开

**navigator对象**

window.navigator.userAgent 用于获取当前的浏览器的信息

**location对象**

document.location和window.location是同一个对象,代表当前页面的相关信息对象,可以读取和赋值

**history对象**

history对象代表用户上网的记录,从窗口被打开的那一刻算起

###### 服务端渲染

更利于SEO 更利于首屏渲染

**SSR的局限：** 服务端压力较大 开发条件受限 学习成本相对较高

服务端渲染是先向后端服务器请求数据，然后生成完整首屏html返回给浏览器；而客户端渲染是等js代码下载、加载、解析完成后再请求数据渲染，等待的过程页面是什么都没有的，就是用户看到的白屏。就是服务端渲染不需要等待js代码下载完成并请求数据，就可以返回一个已有完整数据的首屏页面

###### 垃圾回收机制

1. 标记清除：垃圾回收器将定期从全局对象开始查找所有所有可以获得的对象和收集所有不能获得的对象。对不可获得的对象将会清除。
2. 引用计数： 跟踪记录每个值被引用的次数，当声明一个变量并将一个引用类型的值赋给该变量时，这个值的引用次数就是1，如果这个值再被赋值给另一个变量，则引用次数加1。相反，如果一个变量脱离了该值的引用，则该值引用次数减1，当次数为0时，就会等待垃圾收集器的回收。

在一个进程内部，要同时干多件事，就需要同时运行多个“子任务”，我们把进程内的这些“子任务”称为线程（Thread）

###### 为啥什么javascript是20.进程和线程 任务队列

单线程：

JavaScript的主要用途是与用户互动，以及操作DOM,这个用途决定了它只能是单线程，否则会带来很复杂的同步问题。

为什么要有异步任务：

**为了避免IO阻塞,主线程完全可以不管IO设备，挂起处于等待中的任务，先运行排在后面的任务。等到IO设备返回了结果，再回过头，把挂起的任务继续执行下去。这样就出现了异步**

###### 任务队列：

1.所有同步任务都在主线程上执行，形成一个执行栈

2.主线程之外，还存在一个"任务队列",只要异步任务有了运行结果,就在"任务队列"之中放置一个事件。

3.一旦"执行栈"中的所有同步任务执行完毕，系统就会读取"任务队列"，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，对应的回调函数进入执行栈，开始执行。

4.主线程不断重复上面的第三步。这个过程是循环不断的,所以整个的这种运行机制又称为Event Loop（事件循环）

1. **async和await**

**语法**

**async function fn(){ await ....}**

**注意点**

**async function表示这个函数里面有异步处理**

**async函数返回一个 Promise 对象，可以使用then方法添加回调函数。**

**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。**

**await只能放到async函数体中**

**await表示后面的处理需要等待,通常await后面是一个 Promise 对象，返回该对象的结果。如果不是 Promise 对象，就直接返回对应的值。**

**await命令后面的 Promise 对象如果变为reject状态，程序中止,reject的参数会被catch方法的回调函数接收到。**

**防止出错的方法，是await语句放在try...catch代码块之中。**

# ES6

#### 声明 let、const

###### let特点：

###### 1.let定义时不会进行变量声明提升

###### 2.变量不允许被重复定义

###### 3.变量不可以被删除

###### 4.在for循环当中用let定义i 循环时可以保存 i 的值

###### 5.在块级元素中用let定义的变量只能在当前块级作用域中使用（注：凡是用{}包裹的都是块级作用域，但是除了对象以外）

###### const特点：

             1.const声明的变量值不允许被修改

　　      2.不会进行变量声明提升

             3.常量不允许被删除

###### 2 .结构赋值

Let { a,b ,c} = [1,2,3]

Console.log(a,b,c)//1,2,3

数组的元素是按次序排列的，变量的取值也是按照位置来决定的

Let {foo:baz} = {foo:"lion",bar:"mouse"};

console.log(baz); *//lion*

对象的属性是没有次序的，只要变量名与属性同名即可

###### 声明类与继承：class、extend

class类:

用class定义的类也是函数

类不存在变量提升,

一个类必须有constructor方法,如果没有显式定义,系统默认添加一个空的constructor方法

constructor方法默认返回实例对象,是类的默认方法,通过new命令生成对象实例时,自动调用该方法.

代码格式：

class Person{

constructor(name,age){

this.name = name;

this.age = age;

}

.......

｝

继承：class 子类 extends 父类

通过extends关键字实现继承

父类的静态方法,可以被子类继承

子类必须在constructor方法中调用super方法,该super方法实际上是调用父类的constructor

子类如果没有定义会默认创建一个constructor方法并且添加一个空的super方法

在子类的构造函数中,只有调用super之后,才可以使用this关键字

代码格式：

**class Student extends Person{**

**constructor(name,age,major){**

**super(name,age);**

**this.major = major;**

**}**

**.....**

**｝**

1. **Promise对象 异步处理**

**Promise的概念**

Promise 是一种异步编程解决方案

将异步操作以同步操作的流程表达出来,避免了层层嵌套的回调函数

Promise对象提供统一的接口,使得控制异步操作更加容易

Promise有三种状态

pending（进行中）

fulfilled（已成功）

rejected（已失败）.

一旦状态改变,就不会再变

**Promise对象的方法**

then(resolved状态的回调,rejected状态的回调),then方法返回的是一个新的Promise实例

catch(回调)

如果异步操作抛出错误,状态就会变为rejected,当then中没有指定rejected状态的回调时就会调用catch方法指定的回调函数

如果运行中抛出错误,也会被catch方法捕获

finally(回调)

不管Promise对象最后状态如何,都会执行的操作

finally方法的回调函数不接受任何参数

Promise构造函数接受一个函数作为参数,该函数的两个参数分别是resolve和reject

Pending =>fulfilled resolve成功时将参数传递出去

Pending =>rejected reject 失败时将参数传递出去

Promise新建后会立即执行

1. **generator（异步编程、yield、next()）**

**作用： 异步操作同步化**

定义

Generator 函数形式上就像是一个普通函数

function关键字与函数名之间有一个星号

函数体内部使用yield表达式定义不同的内部状态

调用

Generator 函数的调用方法与普通函数一样，也是在函数名后面加上一对圆括号

调用 Generator 函数后，该函数并不执行,返回一个遍历器对象

调用遍历器对象的next方法

next方法处理流程

遇到yield表达式，就暂停执行后面的操作，并将紧跟在yield后面的那个表达式的值，作为返回的对象的value属性值

下一次调用next方法时，再继续往下执行，直到遇到下一个yield表达式

如果没有再遇到新的yield表达式，就一直运行到函数结束，直到return语句为止，并将return语句后面的表达式的值，作为返回的对象的value属性值

如果该函数没有return语句，则返回的对象的value属性值为undefined

next方法作用

next方法的作用是分阶段执行Generator函数。

每次调用next方法，会返回一个对象，表示当前阶段的信息（value属性和done属性）

value属性是yield语句后面表达式的值，表示当前阶段的值；

done属性是一个布尔值，表示 Generator 函数是否执行完毕，即是否还有下一个阶段

next 方法的参数

next方法可以带一个参数，该参数就会被当作上一个yield表达式的返回值。

由于next方法的参数表示上一个yield表达式的返回值，所以在第一次使用next方法时，传递参数是无效的。

for...of循环可以自动遍历 Generator 函数运行时生成的Iterator对象，且此时不再需要调用next方法。

1. **箭头函数**

箭头函数就是使用"箭头"(=>)来定义函数

箭头函数的基本用法

let fn = (x,y)=>{

console.log(x,y);

return x + y;

}

如果函数体只有返回值,函数体的花括号和return语句可以省略,简写

let fn = (x,y)=> x + y

如果只有一个参数,圆括号可以省略,简写:

let fn = x =>{

console.log(x);

return x + 10;

}

###### this.的指向问题

箭头函数中的this是在定义时确定的(定义时属于谁),而不是在运行时确定(谁调用)的

箭头函数导致this总是指向函数定义生效时所在的对象

**箭头函数和普通函数的区别**

箭头函数没有原型对象,所以不能用箭头函数来定义构造函数

箭头函数中没有arguments

如果定义有名称的箭头函数,需要把箭头函数赋值给变量,所以箭头函数不存在声明的提升

###### map和set有没有用过，如何实现一个数组去重，map数据结构有什么优点？

Set

类似于数组,但值唯一的数据结构

Set对象的size属性返回Set实例的值的总数 set对象.size

Set对象的生成

let s = new Set()

let s = new Set(数组)

Set对象的操作方法

add(value):添加某个值,返回 Set 结构本身

delete(value):删除某个值,返回一个布尔值,表示删除是否成功

has(value):返回一个布尔值,表示该值是否为Set的成员

clear():清除所有值,没有返回值

Set对象的遍历方法

keys():返回键名的遍历器

values():返回键值的遍历器

entries():返回键值对的遍历器

forEach(function(value,key){}):使用回调函数遍历

利用Set值唯一去除数组重复值

var arr1 = [11,22,33,44,11,22];

let s1 = new Set(arr1);

var arr2 = [...s1];

注意点

Set只有值没有键,所以key和value一样

Set的遍历顺序就是插入顺序

Map

类似于对象,但键的范围不限于字符串,各种类型的值(包括对象)都可以当作键的数据结构

Map对象的size属性返回Map实例的键值对的总数

Map对象的生成

let map = new Map()

Map对象的操作方法

set(key,value):设置键名key对应的键值为value，然后返回整个 Map 结构,如果key已经有值，则键值会被更新，否则就新生成该键。

get(key):读取key对应的键值，如果找不到key，返回undefined。

delete(key):delete方法删除某个键，返回true。如果删除失败，返回false。

has(key):has方法返回一个布尔值，表示某个键是否在当前 Map 对象之中。

clear():清除所有键值,没有返回值

Map对象的遍历方法

keys():返回键名的遍历器

values():返回键值的遍历器

entries():返回键值对的遍历器

forEach(function(value,key){}):使用回调函数遍历

注意点

Map的遍历顺序就是插入顺序

###### map 这以其特殊的 key ，value存储方式，提供了我们灵活便捷的[数据处理](https://www.baidu.com/s?wd=%E6%95%B0%E6%8D%AE%E5%A4%84%E7%90%86&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)方式，可以轻松的实现负责的数据结构

Map方法去重

**function arrayNonRepeatfy(arr) {**

**let hashMap = new Map();**

**let result = new Array(); // 数组用于返回结果**

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

**if(hashMap.has(arr[i])) { // 判断 hashMap 中是否已有该 key 值**

**hashMap.set(arr[i], true); // 后面的true 代表该 key 值在原始数组中重复了，false反之**

**} else { // 如果 hashMap 中没有该 key 值，添加**

**hashMap.set(arr[i], false);**

**result.push(arr[i]);**

**}**

**}**

**return result;**

**}**

**let arr = [1, 1, 1, 2, 3, 3, 4, 5, 5, "a", "b", "a"];console.log(arrayNonRepeatfy(arr)); // [ 1, 2, 3, 4, 5, 'a', 'b' ]**

set去除重复项

Var arr = [11,22,33,11,44,22]

Var s1 = new set(arr)

Console.log(s1)

###### ES6怎么编译成ES5,css-loader原理,过程

1. 初始化项目 npm init
2. 全局安装Babel-cli npm install -g babel-cli
3. 本地安装babel-preset-es2015 和 babel-cli

安装完成后，我们可以看一下我们的package.json文件，已经多了devDependencies选项

1. 新建.babelrc 在根目录下新建.babelrc文件，并打开录入的代码

{

    "presets":[

        "es2015"

    ],

    "plugins":[]

}

1. 建立后在终端输入转换命令 这次ES6成功转化为ES5的语法。

babel src/index.js -o dist/index.js

**Css-loader的过程**

**安装相关的loader ：npm install --save-dev style-loader css-loader**

修改配置文件： **module: {**

**rules: [**

**{**

**test: /\.css$/,**

**use: [**

**'style-loader',**

**'css-loader'**

**]**

**}**

**]**

**}**

**保持从新启动.**

**加载图片loder也需要安装别的 并重新配置**