# \*\*\*\*\*\*\*css2总结\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 1.元素与元素发生重合

## 1.1浮动: div结构需要换位置

浮动注意问题:

1. 元素永远都在父元素里浮动

2. 浮动元素永远不能超过前面的兄弟元素

3. 文字环绕浮动元素(本质)

4. 块元素在一行排列

## 1.2定位

position:relative;top:-200px;left:0;

position:absolute;top:0;left:0;-------------------推荐使用

position:fixed;top:0;left:0;

## 1.3负外边距: (文字区域盖不上)

margin-top:-200px;

# 2.粘连布局

块元素默认宽为父元素的100%,默认高为0.

<!--模拟屏幕区域: 因为移动端浏览器比较多,所以html,body操作不一样,为了避免操作html,body-->

/\*width,height 参照父元素拿的百分比\*/

#wrap{

width:100%;

height:100%

min-height: ;/\*最小高度\*/

max-height: ;/\*最大高度\*/

}

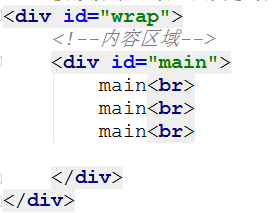
html,body{

Height:100%;/\*屏幕的百分比\*/

}

#wrap{  
 width: 100%;  
 min-height: 100%;  
 background: deeppink;  
 }

#main{  
 font-size: 30px;  
 background: yellowgreen;  
 padding-bottom: 50px;  
 }  
 #footer{  
 width: 100%;  
 height: 50px;  
 background: orange;  
 margin-top: -50px;  
 }



#总结  
 1.footer必须是一个独立的结构,与wrap没有任何嵌套关系  
 2.footer要使用margin为负来确定自己的位置  
 3.wrap区域必须要被自己的子元素撑开 [最小100%, 最大值由内容撑开)

# 3.浏览器滚动条的位置

考虑 document html body;

1.html body中只有一个有overflow属性时,这个属性会传给document

2.html body都存在overflow属性时.body的overflow属性才能生效,

3.当html body都存在overflow属性时.body的overflow属性一直都会作用于body身 上,html的overflow属性一直都会作用于document身上

/\*禁止系统滚动条\*/

html,body{

height:100%;  
 overflow:hidden;

}

# 4.清除浮动

## 4.1直接给父元素设置一个死的高度

## 4.2给父元素加属性

1.给父元素加上浮动,

2.给父元素加上定位 position:absolute;

3.给父元素加上定位 position:fixed;

注: margin设置的左右auto不生效

## 4.3 overflow:hidden;

-------------推荐使用

注: 2和3的本质都是开启了BFC,开启BFC后,父元素可以包含浮动的的子元素

## 4.4在box的后边添加一个空div

设置样式clear:both;

<div style=”clear:both” ></div>

## 4.5在box的后边添加<br clear=”all”>

## 4.6 clearfix

.clearfix:after{

content: ’’;

display: block;

clear: both;

}

# 5.定位

固定定位,参照浏览器窗口定位,与父元素没有任何关系

position:fixed;

相对定位,参照自身在文档流中的位置

position:relative;

绝对定位

position:absolute;

偏移量: top ,left,bottom,right

如果有定位父级元素,参照定位父级元素定位

如果没有定位父级元素,参照初始包含块定位

包含块:

“根元素”的包含块(也称为初始包含块),由用户代理建立,也叫浏览器厂商.在HTML中,根元素就是html元素.在大多数浏览器中,初始包含块是一个视窗大小的矩形,并不是视窗.

## 5.1绝对定位模拟固定定位

document上的滚动条移动导致初始包含块移动,为了保证当前的初始包含块不动,要把document上的滚动条干掉

html｛

height:100%;

overflow:hidden;

｝

给body设置滚动条

body{

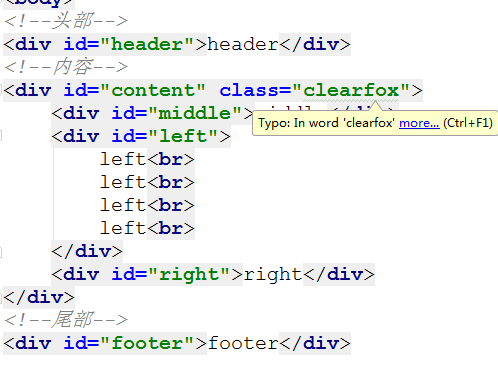
height:100%;

overflow:auto;

}

在body里设置另外一个块元素,高度2000px,让body显示滚动条

## 5.2三列布局



### 5.2.1圣杯布局

需求:

1.两边固定,中间自适应

2.先加载middle内容

3.等高布局(未实现)

步骤:

布局: 有头 有尾 有内容 先加载middle内容

浮动让三者在一行,出现高度塌陷,清浮动

left上去: margin-left: -100%; right同理

middle内容未显示,container:padding: 0 200px;

此时用相对定位把left,right拉回来

最小宽度问题: min-width:600px

问题: 内容过多,文字溢出未解决(等高布局解决)

\* {  
 margin: 0;  
 padding: 0;  
 }  
 body{  
 min-width: 600px;  
 }  
 #header,#footer{  
 width: 100%;  
 height: 50px;  
 line-height: 50px;  
 background: grey;  
 text-align: center;  
 }  
 #content{  
 padding: 0 200px;  
 }  
 #left,#right{  
 float: left;  
 width: 200px;  
 /\*height: 200px;\*/  
 background: deeppink;  
 }  
 #middle{  
 float: left;  
 width: 100%;  
 /\*height: 200px;\*/  
 background: yellowgreen;  
 }  
 #left{  
 margin-left: -100%;  
 position: relative;  
 top:0;  
 left: -200px;  
 }  
 #right{  
 margin-left: -200px;  
 position: relative;  
 top:0;  
 right: -200px;  
 }   
 .clearfox:after{  
 content: '';  
 display: block;  
 clear: both;  
 }

### 5.2.2双飞翼布局

步骤:

布局: 有头 有尾 有内容 先加载middle内容

浮动让三者在一行,出现高度塌陷,清浮动

给middle宽度100%

left上去: margin-left: -100%; right同理

处理middle: 改变dom结构

#middle-inner{

margin-left: 200px;

margin-right: 200px;

}

最小宽度问题: min-wi dth:600px

等高处理

#container{overflow: hidden;}

优点:

内容部分(中间列)优先加载

结构上仅仅添加了一个父级容器

任何一列都可以作为最高列

缺点:

多写了一成html结构

### 5.2.3等高布局(伪)



### 5.3 left、top、right、bottom的默认值为auto

### 5.4 html和body的默认高度为0

# 6.文本

## 6.1 省略号

设置当文本溢出包含元素时发生的样式

步骤:

1.包含元素为块元素或行内块

display: block;

2.文本永不换行

white-space: nowrap;

3.溢出文本截取掉

overflow: hidden;

4.实现省略号

text-overflow: ellipsis;

## 6.2行高

1.默认字体大小为16px

2.浏览器支持的最小字体为12px

3.chrome的默认行高为21px

4.firefox的默认行高为22px

5.IE的默认行高为18px

6.line-height: 1; 倍数: 字体大小

## 6.3垂直对齐

### 6.3.1典型的行内块元素

### img button input video canvas

行内块元素像素留白是给文本使用的

vertical-align:baseline;默认: 基线对齐

vertical-align:top; 上对齐

vertical-align:bottom; 对齐

vertical-align:middle; 居中对齐

改变基线默认位置: 1.font-size 字体大小改变 ,2.line-height

### 6.3.2行内块元素水平垂直居中

父元素｛

text-align:center;

line-height: ;

｝

行内块｛

display:inline-block;

vertical-align:middle

｝

行内块元素的缝隙问题

底部缝隙、左右换行缝隙

父元素字体大小设置为0

行内块元素设置浮动

### 6.3.3块元素水平垂直居中

已知元素的宽高

width: 200px;  
 height: 200px;  
 background: greenyellow;

1.{

position: absolute;  
 left:50%;  
 top: 50%;  
 margin-left:;  
 margin-top: ;

}

2.{

position: absolute;  
 left: 0;  
 top: 0;  
 right: 0;  
 bottom: 0;  
 margin:auto;

}

给元素设置宽高

width height

定位设置宽高

box{position: absolute;  
 left: 0;  
 top: 0;  
 right: 0;  
 bottom: 0;}

子元素的大小与父元素等大

让元素飘起来(浮动、定位)

此时宽高由子元素自动撑开

# 7.引起父元素高度塌陷的原因

## 7.1浮动

清除浮动6种方法

## 7.2定位

只能给父元素设置高来解决

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 1. for in

遍历对象上的属性名,枚举对象上的属性名

var obj = {

name: 'Tom',

age: 18,

height: 180

}

for(var i in obj){

// i得到的是属性名,类型是字符串string

console.log( obj[i] );//打印出的是属性值(Tom 18 180)

}

# 2. 读取对象上的属性值

2.1 属性值 = 对象名.属性名

console.log(obj.name);

2.2 属性值 = 对象名['属性名']

console.log['name'];//加引号

console.log[name];//没加引号的name是变量

console.log(obj.width);//undifined

# 3. 赋值语句(给对象设置新属性)

obj.age = 20;

obj[ 'age' ] = 22;

如果对象上没有这个属性,那么用

obj[name] = value

来设置新属性,name是新属性的属性名,value是属性值.

# 4. 数组合并

var arr1 = [1,1,1];  
 var arr2 = [2,2,2];  
 //数组合并  
 console.log(arr1.concat(arr2)); //[1,1,1,2,2,2]

# 5. 图片

var imgNode = document.getElementById('imgNode');  
 console.log('外部='+imgNode.width);  
  
 //保证所有资源加载完成 (img,js,css,font,包)  
 window.onload = function () {  
 console.log('内部='+imgNode.width);  
 }  
 //保证图片资源加载完成  
 imgNode.onload = function () {  
 console.log('img='+imgNode.width);  
 }  
 //图片资源加载失败  
 imgNode.onerror = function () {  
 console.log('222222222222');  
 }

# 6. 元素水平垂直居中

已知元素的宽高

width: 200px;  
 height: 200px;  
 background: greenyellow;

1.{

position: absolute;  
 left:50%;  
 top: 50%;  
 margin-left: -100px;  
 margin-top: -100px;

}

2.{

position: absolute;  
 left: 0;  
 top: 0;  
 right: 0;  
 bottom: 0;  
 margin:auto;

}

元素的高度未知

3.{

position: absolute;  
 left: 0;  
 top: 0;  
 transform: translate(-50%,-50%);

}

4.新版本

{

display: flex;

justify-content: center;  
 align-items: center;

}

5.老版本

{

display: -webkit-box;  
 -webkit-box-pack: center;  
 -webkit-box-align: center;

}

# 7. 拖拽的原理

元素的最终位置 = 元素的初始位置 + 鼠标的距离差

鼠标按下事件 onmousedown

鼠标抬起事件 onmouseup

鼠标移动事件 onmousemove

## 7.1获取鼠标的位置:

e是event对象

鼠标的位置 e.clientX e.clientY

鼠标相对于元素的位置 e.offsetX e.offsetY

鼠标相对于页面的位置 e.pageX e.pageY

鼠标相对于屏幕的位置 e.screenX e.screenY

## 7.2获取手指位置

changedTouches:

目标元素目标事件上的手指列表,最终我们选择手指列表要的是changedTouches

targetTouches: 目标元素上的手指列表

touches: 屏幕上的手指列表

box.addEventListener('touchstart',function(event){

//直接在事件当中把手指先给拿到,以后用的时候,我们可以直接获取位置；

var touch = event.changedTouches[0];

console.log(event);

console.log(touch.clientX);

console.log(touch.clientY);

})

# 8. 取消浏览器默认行为

1.dom0

return false;

注: IE8支持return false,但是IE8中禁止的浏览器默认行为里面不包含拖拽,

解决方案: 全局捕获.

2.dom2

event.preventDefault()

# 9. 全局捕获

全局捕获在ie底下才有实质性的作用  
 在谷歌中完全不存在全局捕获,设置全局捕获会报错  
 在火狐中存在全局捕获,但其没有作用

谷歌和火狐不需要设置,自己有这个功能

设置全局捕获:

element.setCapture()

释放全局捕获:

document.releaseCapture()

解决方法:

//设置全局捕获: element.setCapture()  
 if(box.setCapture){  
 box.setCapture();  
 }

box.setCapture && box.setCapture();

//释放全局捕获: document.releaseCapture()  
 if(document.releaseCapture){  
 document.releaseCapture();  
 }

document.releaseCapture && document.releaseCapture();

# 10.求视口的宽度、高度

## 10.1 pc端

1.浏览器最干净的视口区域

document.documentElement.clientWidth

document.documentElement.clientHeight

2.包含浏览器滚动条的区域

window.innerWidth

window.innerHeight

3.包含浏览器的外部边框(镶边)

window.outerWidth

window.outerHeight

4.屏幕宽度

screen.width

screen.height

## 10.2移动端

10.2.1布局视口

为了转移pc的网页,而引入的一个容器.它的大小是要远大于设备独立像素,各个设备差异不大.（980）

为了把这个布局视口当中的元素全部展现给客户,并且不能出现横向滚动条.浏览器厂商会自作主张把布局视口

进行压缩（压缩的是css面积）

最终：我们的手机宽度此时用三种像素描述就变为了：

物理像素 750

设备独立像素 375

css像素 980

10.2.2视觉视口

宽度永远是屏幕宽度所包含的css像素值

10.2.3理想视口

meta name = "viewport" content = "width=device-width"

最终达到的目标 布局适口 = 视觉视口 = 理想视口 = 设备独立像素 = 375

# 11.getBoundingClientRect()

返回值是一个对象,包含了一组用于描述边框的只读属性——left、top、right和bottom,

除了 width 和 height 外的属性都是**相对于视口的'左上角'**位置而言的.

left、top、right和bottom的位置都是参照视口的左上角获取的.

# 12.滚动条

####scale1滚动条高度的系数

滚动条高度 / 视口高度 = 视口高度 / 内容高度 = scale1

scale1 = 视口高度/内容高度

内容偏移量 = 滚动条偏移量 / scale1

滚动条偏移量 = 内容偏移量 \* scale1

# 13.鼠标移入移出事件

13.1第一种: 会产生事件冒泡

鼠标移入onmouseover

鼠标移出onmouseout

13.2第二种

鼠标移入onmouseenter

鼠标移出onmouseleave

多个元素的时候最好使用第二种

# 14.鼠标滚轮事件

ie/chrome : onmousewheel(dom0)  
 event.wheelDelta  
 上: 120  
 下: -120  
   
 firefox : DOMMouseScroll 必须用(dom2的标准模式)  
 event.detail  
 上: -3  
 下: 3  
   
 return false阻止的是 dom0 所触发的默认行为  
 dom2 需要通过event下面的event.preventDefault();

# 15.取消冒泡

**event**.stopPropagation();

# 16.绑定事件的方式

## 16.1绑定事件

1.dom0

box.onclick = function () {};

2.dom2

box.addEventListener( 'cilck' , function () {});

## 16.2解绑操作

1.dom0

box.onclick = null;

2.dom2

box.removeEventListener( 'click' , function () {})

## 16.3取消浏览器默认行为

1.dom0

return false;

2.dom2

event.preventDefault()

# 17.元素高亮效果

-webkit-tap-highlight-color: rgba(0,0,0,0);

-webkit-tap-highlight-color: transparent;

设置成透明

# 18.inline-block元素

img input button audio video canvas

# 19.检测浏览器窗口大小发生改变

window.onresize = function() {}

# 20.百分比参照自身

border-radius  
 background-size  
 transform-origin  
 transform: translate(-50%,-50%);  
 线性渐变: 轴线长度  
 径向渐变: 对角线一半

# 21.实参列表arguments

定义的函数没有写形参的时候,可以通过实参列表来获取参数.

实参列表 arguments ,是函数调用是传过来的参数,是一个列表.

# 22.函数三要素

功能 参数 返回值

# 23. js内置对象

## 23.1 Math对象

Math.PI 圆周率

Math.abs() 求绝对值

Math.ceil() 向上舍入

Math.floor() 向下舍入

Math.round() 四舍五入

Math.max() 求最大值 参数: (1,2,3,4,5,6,7)

Math.min() 求最小值 参数: (1,2,3,4,5,6,7)

Math.random() 求随机数 返回值是0到1之间的一个小数

Math.sqrt() 开根号

## 23.2 Date对象

var date = new Date();

console.log(date); //Wed Mar 20 2019 17:02:10 GMT+0800 (中国标准时间)

date.getFullYear(); //年份2019

date.getMonth(); //月份2

date.getDate(); //日期20

date.getDay(); //星期3 (0-6)

date.getHours(); //小时19

date.getMinutes(); //分钟17

date.getSeconds(); //秒52

date.getTime(); //获取毫秒

date.toLocaleDateString();//日期2019/3/20

date.toLocaleTimeString();//时间 下午7:17:52

date.valueOf(); //毫秒 1553080672481 根据世界时返回1970年1月1日到指 定日期的毫秒数

## 23.3基本包装类型

/\*

\* 值类型:

\* 数值型 字符串 布尔型

\*

\* 基本包装类型:

\* 基本包装类型是构造函数,利用构造函数实例化的是对象,我们在写程序时,会

\* 利用值类型的数据进行值的比较或者数学运算,对象都是存储在各自的内存地址中

\* 无法进行数学运算或者值的真假比较.当值类型的数据的变量需要调用基本包装类型

\* 的方法时,会自行创建一个基本包装对象,调用对象中的方法,当调用结束后,对象

\* 会自动清理掉,通常情况下,我们不会人为的去创建基本包装类型

\*

\*/

var num = 5; //number

var str = '哈哈'; //string

var bool = true; //boolean

var num01 = new Number(1);

var num02 = new Number(1);

var str01 = new String('哈哈');

console.log(num01);

console.log(str01);

var bool01 = new Boolean(true);

var bool02 = new Boolean(true);

if(num == 5){

alert('执行');

}

if(num01 == num02){

alert('执行');

}

console.log(num.toString());

console.log(num);

## 23.4字符串常用方法

/\*

\* 字符串的不可变性: 当变量保存一个字符串数据后,就不能更改了,当再赋值的时候

\* 是重新开辟了一个内存地址来保存

\*/

var str01 = '下午上课一点也不困';

console.log(str01.length);

for (var i = 0; i < str01.length; i++) {

console.log(str01[i]); //下 午 上 课 一 点 也 不 困

}

str01[0] = '上';

//str01 = '哈哈';

//console.log(str01);

var str02 = '下午a上课一点也不困';

console.log(str02.charAt(1));

//.charAt(索引),返回值是指定索引位置的字符串,超出索引,结果是空字符串

console.log(str02.charCodeAt(2));

//.charCodeAt(索引)返回在指定的位置的字符的Unicode编码

console.log(str02.concat('但老刘困','困了就睡'));

//.concat(字符串1,字符串2,...);返回的是拼接之后的新的字符串

console.log(str02.indexOf('课',5));

//.indexOf(要找的字符串,从某个位置开始的索引);

返回的是这个字符串的索引值,没找到则返回-1

console.log(str02.lastIndexOf('课',3));

//.lastIndexOf(要找的字符串,从某个位置开始的索引);

从后向前找,但是索引仍然是从左向右的方式,找不到则返回-1

console.log(str02.replace('下','上'));

//.replace("原来的字符串","新的字符串");用来替换字符串的

console.log(str02.slice(5,9));

//.slice(开始的索引,结束的索引);

从索引的位置开始提取,到索引的前一个结束,并返回这个提取后的字符串

var str03 = '周杰伦|蔡依林|林俊杰|吴亦凡|老刘';

console.log(str03.split('|')); //.split("要干掉的字符串");切割字符串

console.log(str02.substr(0,5)); //.substr(开始的索引,个数);返回的是截取后的新的字符串

console.log(str02.substring(5,9)); //返回截取后的字符串,不包含结束的索引的字符串

var str04='aBcDeFG';

console.log(str04.toLocaleLowerCase()); //转小写

console.log(str04.toLocaleUpperCase()); //转大写

var str = prompt();

console.log(str.trim().length);

if(str.trim() == '金牛座'){

alert('牛逼');

}

console.log(str.trim()); //去掉两端空格

## 23.5数组常用方法

var arr = [1, 2, 3, 4, 5];

### 23.5.1 pop();

arr.pop()

从数组尾部弹出一个元素并返回弹出的元素

length就会减1.

尾部:是指索引最大的地方.

### 23.5.2 push()

arr.push('哈哈')

从数组的尾部压入一个元素,并返回数组的新长度

### 23.5.3 shift()

arr.shift()

从数组的头部弹出一个元素,并返回此元素

### 23.5.4 unshift()

arr.unshift('呵呵')

从数组的头部压入一个元素.

### 23.5.5 reverse()

var arr02 = [1, 2, 3, 4, 5];

arr02.reverse()

对原数组进行逆序

### 23.5.6 concat()

var arr03 = [1, 2, 3, 4, 5];

arr03.concat( [6, ['haha', 'hehe'], 8] )

连接原数组的元素和传递的参数**形成一个新数组**并返回,不影响原来的数组.

如果传入的参数是数组,会把数组中的元素跟原数组的元素进行合并成一新数组.

[1,2,3].concat(9,1,4); => [1, 2, 3, 9, 1, 4]

[1,2,3].concat([9,true]); => [1, 2, 3, 9, true]

[1,2,3].concat([9,true, ['22',4, 9], 33]) => [1, 2, 3, 9, true, Array(3), 33]

### 23.5.7 join()

var arr04 = ['周杰伦', '蔡依林', '林宥嘉', '老刘'];

arr04.join('|')

可以把数组的元素(项)连接成字符串,接收一个参数为连接符号,默认是逗号,返回的结果为字符串.

[1, 2, 3].join(); => 1,2,3

[4,true, 3].join(“-”) ; => 4-true-3

### 23.5.8 slice()

复制数组的一部分

获取数组的一个片段或者子数组

接收1个到2个参数.参数：获取数组起始索引和结束索引(前一个)

如果只指定一个参数代表：从索引位置到数组结尾.

此方法对原数组没有影响.

m = [1,2,3,4,5];

m.slice(2); =>[3, 4, 5]

m.slice(3, 4); => [4]

### 23.5.9 splice() \*

arr04.splice(1, 1, ['哈哈', '呵呵']);

在原数组上进行插入或者删除数组元素,会影响原来数组.

返回的结果是删除的元素组成的数组.

参数：可以接受1个参数,2个参数或者2个以上的参数.

第一个参数是删除数据的索引位置到结束,包含自己,负值从后面开始

第二个参数是要删除数组元素的个数

第三个参数是要插入到原数组中的元素,插入的位置从第一个参数所在的索引开始.

1). 删除数据

[1, 2, 3, 4, 5].splice(2); // 删除=>[3, 4, 5] 原数组：[1,2]

[1, 2, 3, 4, 5].splice(3,2); // 删除=>[4, 5] 原数组：[1,2,3]

[1, 2, 3, 4, 5].splice(-2); // 删除=>[4, 5] 原数组：[1,2,3]

a = [1,2,3,4,5];

a.splice(3,2,33,'222',[99,98]); // 删除=>[4,5]

a => [1, 2, 3, 33, "222", Array(2)]

2). 插入数据

a=[1,2,3]; a.splice(1,0,5,6); //=>[] a=[1,5,6,2,3]

3). 替换数据

a=[1,2,3];a.splice(1,1,4); //=>[2] a=[1,4,3]

### 23.5.10 sort()

转成字符串排序

var arr05 = ['b', 'a', 'k', 'c', 'e'];

var arr06 = [2, 32, 24, 52, 6, 77, 31];

如果调用该方法时没有使用参数,将按字母顺序对数组中的元素进行排序,说得更精确点,是按照字符编码的顺序进行排序.要实现这一点,首先应把数组的元素都转换成字符串(如有必要),以便进行比较.

如果想按照其他标准进行排序,就需要提供比较函数,该函数要比较两个值,然后返回一个用于说明这两个值的相对顺序的数字.比较函数应该具有两个参数 a 和 b,其返回值如下：

若a小于b,在排序后的数组中a应该出现在b之前,则返回一个小于0的值.

若a等于b,则返回0.

若a大于b,则返回一个大于 0 的值.

function fn(a, b) {

if(a < b) {

return -1;

}

if(a == b) {

return 0;

}

if(a > b) {

return 1;

}

}

var num = fn(2,32);

if(num == 1){

}

console.log(arr06.sort(fn));

console.log(arr06.sort(function (a,b){

if(a<b){

return -1;

}

if(a==b){

return 0;

}

if(a>b){

return 1;

}

}));

console.log(arr06.sort(function(a, b) {

return a - b;

}));

# 24.对象相关

obj = {

name: '';

eat:function(){ }

}

function eat(){ //这个函数可以看做是 window.eat = function(){}

}

函数和方法的区别:

window下面的eat= function(){} 称为函数;

普通对象下面的eat:function(){}称为这个对象下的eat方法;

# 25.++和--

++在后,先赋值后++;

++在前,先++后赋值.

# 26.回调函数

网一个已经写好的函数中添加外部逻辑,必须使用回调函数.

# 27.元素隐藏的方式

display: none;   
 visibility: hidden;  
 opacity:0;  
 width: 0;  
 height: 0;  
 transform: scale(0);  
 transform: translate(-10000px)

# 28.循环遍历加监听解决方法

***<!--需求: 点击某个按钮, 提示"点击的是第n个按钮"-->*<script type="text/javascript">  
*//以下代码有问题* var *btns* = document.getElementsByTagName('button')  
 for (var *i*=0; *i*<*btns*.length; *i*++ ){  
 var *btn* = *btns*[*i*]  
 *btn*.onclick = function () {  
 alert(*i*)  
 }  
 }  
*//缓存计数器* var *btns* = document.getElementsByTagName('button')  
 for (var *i*=0; *i*<*btns*.length; *i*++ ){  
 var *btn* = *btns*[*i*]  
 *btn*.index = *i  
 btn*.onclick = function () {  
 alert(this.index)  
 }  
 }  
*//闭包解决* var *btns* = document.getElementsByTagName('button')  
 for (var *i*=0; *i*<*btns*.length; *i*++ ){  
 (function (i) {  
 var btn = *btns*[i]  
 btn.onclick = function () {  
 alert(i)  
 }  
 })(*i*)  
 }  
*//构建块级作用域* var *btns* = document.getElementsByTagName('button')  
 for (let *i*=0; *i*<*btns*.length; *i*++ ){  
 var *btn* = *btns*[*i*]  
 *btn*.onclick = function () {  
 alert(*i*)  
 }  
 }**

# 29.对象创建方式

## 29.1 Object构造函数模式

***\* 套路: 先创建空Object对象, 再动态添加属性/方法  
\* 适用场景: 起始时不确定对象内部数据  
\* 问题: 语句太多（创建多个对象的时候）***

**var *p*** = **new** Object()  
***p***.**name** = **'kobe'  
*p***.**age** = 18  
***p***.setName = **function** (name) {  
 **this**.**name** = name  
}

## 29.2对象字面量

***\* 套路: 使用{}创建对象, 同时指定属性/方法  
\* 适用场景: 起始时对象内部数据是确定的  
\* 问题: 如果创建多个对象, 有重复代码(方法)***

**var *p*** = {  
 **name**:**'kobe'**,  
 **age**:18,  
 setName:**function** (name) {  
 **this**.**name** = name  
 }  
}

## 29.3工厂模式

***\* 套路: 通过工厂函数动态创建对象并返回  
\* 适用场景: 需要创建多个对象  
\* 问题: 对象没有一个具体的类型, 都是Object类型***

**function** *creatPerson*(name,age) {  
 **var** p = {  
 **name**:name,  
 **age**:age,  
 setName:**function** (name) {  
 **this**.name = name  
 }  
 }  
 **return** p  
}

**var *p1*** = *creatPerson*(**'wade'**,18)  
**var *p2*** = *creatPerson*(**'peiqi'**,18)

**console**.log(typeof ***p1***)  
**console**.log(typeof ***p2***)

## 29.4自定义构造函数模式

***\* 套路: 自定义构造函数, 通过new创建对象  
\* 适用场景: 需要创建多个类型确定的对象  
\* 问题: 每个对象都有相同的数据, 浪费内存***

**function *Person*(name,age) {  
 this.name = name  
 this.age = age  
 this.setName = function (name) {  
 this.name = name  
 }  
}  
function *Dog*(name,age) {  
 this.name = name  
 this.age = age  
 this.setName = function (name) {  
 this.name = name  
 }  
}  
var *p1* = new *Person*('kobe',18)  
var *p2* = new *Person*('wade',19)  
var *d1* = new *Dog*('wc',19)  
var *d2* = new *Dog*('xh',19)**

## 29.5构造函数+原型的组合模式

***\* 套路: 自定义构造函数, 属性在函数中初始化, 方法添加到原型上  
\* 适用场景: 需要创建多个类型确定的对象***

**function** *Person*(name,age) {  
 **this**.**name** = name  
 **this**.**age** = age  
}  
*Person*.**prototype**.setName = **function** (name) {  
 **this**.**name** = name  
}

**var *p1*** = **new** *Person*(**'kobe'**,18)  
**var *p2*** = **new** *Person*(**'wade'**,19)  
***p1***.setName(**'haha'**)  
**console**.log(***p1***)

# 30.数组常用方法

## 30.1数组的简单方法

***push、unshift、pop、splice... 会影响原数组（会修改原数组数据）***

## 30.2数组的高级方法

### reduce

***统计、求和、累加。。。 计算购物车的总价***

### forEach

***遍历  
 for循环 （用来遍历数组） 性能最好，一般不用  
 for in循环 （用来遍历对象（多）、数组（极少））  
 for of循环 （用来遍历有iterator接口的容器，对象默认不行）  
 forEach（用来遍历数组）  
 while循环 （一般用来取代递归函数：函数内部调用自己函数）***

***数组: forEach((item,index)=>{}) item在前写的时候就可以省略 index  
对象: for in***

### map

***用来对原数组进行加工处理，生成新数组（不会修改原数组），新数组长度和原数组一样，但内容往往发生变化  
 对原数据进行加工显示，显示加工后的数据***

### filter

***用来对原数组进行加工处理，生成新数组（不会修改原数组），新数组长度比原数组更少，但内容不会变化  
 对原数据进行过滤显示，只显示符合条件的数据***

***args.reduce((previous, current) => {}, initValue)  
previous 上次函数返回值  
current 当前遍历数据的元素  
initValue 就是第一次 previous 的值***

### flat方法

***用于扁平化数组*const *arr* = [1, [2], [[3], [4]], [[[5]]]];  
*//自定义flat方法*function *flat*(arr) {  
 */\*  
 什么时候函数需要形参：如果函数内部需要使用函数外部的数据，传参  
 \*/  
 // 初始化是一个空数组* let result = [];  
 *// [[3], [4]] --> [3] [4]* arr.forEach((item) => {  
 if (Array.isArray(item)) {  
 *// 元素是数组  
 // concat连接数据的方法，它不会修改原数组* result = result.concat(*flat*(item)); *// 每次调用flat方法 返回一个result* } else {  
 *// 元素不是数组* result.push(item);  
 }  
 })  
 *// 返回一个扁平化后的数组* return result;   
}  
console.log(*flat*(*arr*));  
Array.prototype.flat = function () {  
 */\*  
 什么时候函数需要形参：如果函数内部需要使用函数外部的数据，传参  
 \*/  
 // 初始化是一个空数组* let result = [];  
 *// this指向调用方法的这个对象 -- arr* this.forEach((item) => {  
 if (Array.isArray(item)) {  
 result = result.concat(item.flat()); *// 每次调用flat方法 返回一个result* } else {  
 *// 元素不是数组* result.push(item);  
 }  
 })  
 *// 返回一个扁平化后的数组* return result;  
};  
console.log(*arr*.flat());**

# 31.await关键字

***await关键字只能在async函数中使用  
作用：  
 1. 等待: 只会等待promise对象，其他内容都不等  
 当promise内部状态变成成功的状态时，就不等了  
 当promise内部状态变成失败的状态时，中断async函数的执行（后面戴代码不执行了），并报错  
 2. 提取promise对象成功状态的返回值，赋值给左边***

# 32.const和let使用

***1. 绝大部分使用let定义变量，只有确定不变的量用const，90%的变量都是let定义的  
2. 绝大部分使用const定义变量，只有确定可变的量用let，90%的变量都是const定义的  
  
使用哪种方式：看我们定义的量是常量居多还是变量居多  
 我们在实际项目中常量居多  
不确定一个量到底是变量还是常量：默认用const定义，只有将来变化了，在改为let***

# 33.this指向

***1. 没有修改this指向的函数，直接调用 window window.fn()  
 2. obj.fn() obj  
 3. call / apply / bind this指向传入第一个参数  
 4. new fn() 创建出来的实例对象  
 5. () => {} 离它最近的一层包裹着它的函数的this***

# 34.call、apply、bind

***相同点：三者都能改变函数调用时this指向。  
 不同点:  
 call 第一个是要改变this指向的对象，后面参数是要调用函数传入的参数（以逗号隔开）  
 apply 第一个是要改变this指向的对象，第二个参数是一个数组（数组里面有函数调用的参数）  
 bind 传参和call一样。 不会执行函数，返回一个this改变后的函数***

# 35.git管理仓库

***## 1、本地仓库管理*\*** 建立忽略文件  
 **\*** .gitignore  
 **\*** 一定要在add之前  
**\*** git init  
**\*** git add ./\*/-A  
**\*** git commit -m 'xxx'  
  
***## 2、远程仓库管理*\*** 新建远程仓库  
  
***## 3、本地和远程的使用*\*** 本地仓库和远程仓库关联  
 **\*** git remote add origin xxxx  
**\*** 本地仓库提交代码到远程仓库去  
 **\*** git push origin master  
**\*** 本地仓库更新远程仓库最新的代码  
 **\*** git pull origin master  
**\*** 本地仓库需要拉取远程仓库的dev分支内容  
 **\*** 注意：默认clone下来是master分支  
 **\*** git fetch origin dev1:dev2 拉取远程仓库dev1分支的内容到本地的dev2分支上

# **36.高阶函数**

***1). 一类特别的函数  
 a. 接受函数类型的参数  
 b. 返回值是函数  
2). 常见  
 a. 定时器: setTimeout()/setInterval()  
 b. Promise: Promise(() => {}) then(value => {}, reason => {})  
 c. 数组遍历相关的方法:***

***forEach()/filter()/map()/reduce()/find()/findIndex()***

***d. 函数对象的bind()  
 e. Form.create()() / getFieldDecorator()()***

***3). 高阶函数更新动态, 更加具有扩展性***

# 37.高阶组件

***1). 本质就是一个函数  
 2). 接收一个组件(被包装组件), 返回一个新的组件(包装组件), 包装组件会向被包装组件传入特定属性  
 3). 作用: 扩展组件的功能  
 4). 高阶组件也是高阶函数: 接收一个组件函数, 返回是一个新的组件函数***