2020初级中级高级web前端面试题超全面细节解析

一、CSS问题

一、flex弹性布局，

> 可以简单的使一个元素居中（包括水平和垂直居中） 栅格式系统布局，bootstrap grid

二、圣杯和双飞翼布局 三栏是布局（两边两栏宽度固定，中间栏宽度自适应）

> 方案一：position（绝对定位法） center的div需要放在最后面 绝对定位法原理将左右两边使用absolute定位，因为绝对定位使其脱离文档流，后面的center会自然流动到他们的上卖弄，然后margin属性，留出左右两边的宽度。就可以自适应了。

> 方案二：float 自身浮动法 center的div需要放到后面 自身浮动法的原理就是对左右使用float:left和float：right，float使左右两个元素脱离文档流，中间的正常文档流中，使用margin指定左右外边距对其进行一个定位。

> 圣杯布局：原理就是margin负值法。使用圣杯布局首先需要在center元素外部包含一个div,包含的div需要设置float属性使其形成一个BFC，并且这个宽度和margin的负值进行匹配

三、左边定宽，右边自适应

> 方案一：左边设置浮动，右边宽度设置100% .left{float:left} .right:{width:100%}

> 方案二：左设置浮动，右用cacl去补宽度计算 .left{float:left} .right:{width:cacl(100vw-200px}

> 方案三：父容器设置display：flex right部分是设置flex：1

> 方案四：右边div套个包裹、并前置、左及包裹 双浮动

四、水平居中

> 行内元素居中（父元素text-align:center）

> 块状元素居中（块状元素没发用text-align）

```
- 1. 宽度一定：margin:auto
- 2.宽度不定：块级变行内，然后在父上text-aligin
    float
```

五、BFC juejin.im/post/5909db...

> 理解：BFC是css布局的一个概念，是一块独立的渲染区域，一个环境，里面的元素不会影响到外部的元素

如何生成BFC：（脱离文档流）

【1】根元素，即HTML元素（最大的一个BFC）

【2】float的值不为none

【3】position的值为absolute或fixed

【4】overflow的值不为visible（默认值。内容不会被修剪，会呈现在元素框之外）

【5】display的值为inline-block、table-cell、table-caption

> BFC布局规则：

1.内部的Box会在垂直方向，一个接一个地放置。

2.属于同一个BFC的两个相邻的Box的margin会发生重叠

3.BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此, 文字环绕效果，设置float

4.BFC的区域不会与float box重叠。

5.计算BFC的高度，浮动元素也参与计算

> BFC作用： 1.自适应两栏布局

2.可以阻止元素被浮动元素覆盖

3.可以包含浮动元素---清除内部浮动 > > >

> 原理: 触发父div的BFC属性，使下面的子div都处在父div的同一个BFC区域之内

4.分属于不同的BFC时，可以阻止margin重叠

> display:flex; 在父元素设置，子元素受弹性盒影响，默认排成一行，如果超出一行，按比例压缩 flex:1; 子元素设置，设置子元素如何分配父元素的空间，flex:1,子元素宽度占满整个父元素align-items:center 定义子元素在父容器中的对齐方式，center 垂直居中justify-content:center 设置子元素在父元素中居中，前提是子元素没有把父元素占满，让子元素水平居中。

2.css3的新特性

> transtion transition-property 规定设置过渡效果的 CSS 属性的名称。

> transition-duration 规定完成过渡效果需要多少秒或毫秒。

> transition-timing-function 规定速度效果的速度曲线。

> transition-delay 定义过渡效果何时开始。

> animation属性可以像Flash制作动画一样，通过控制关键帧来控制动画的每一步，实现更为复杂的动画效果。

> ainimation实现动画效果主要由两部分组成：

> 通过类似Flash动画中的帧来声明一个动画；

> 在animation属性中调用关键帧声明的动画。

> translate 3D建模效果

3.img中alt和title的区别

> 图片中的 alt属性是在图片不能正常显示时出现的文本提示。

> 图片中的 title属性是在鼠标在移动到元素上的文本提示。

4.用纯CSS创建一个三角形

```
 <style>
    div {
        width: 0;
        height: 0;
        border-top: 40px solid transparent;
        border-left: 40px solid transparent;
        border-right: 40px solid transparent;
        border-bottom: 40px solid #ff0000;
    }
    </style>
</head>
<body>
```

```
  <div></div>
</body>
复制代码
```

5.如何理解CSS的盒子模型?

> 标准盒子模型：宽度=内容的宽度（content）+ border + padding

> 低版本IE盒子模型：宽度=内容宽度（content+border+padding)

6.如何让一个div水平居中

> 已知宽度，block元素 ，添加添加margin:0 auto属性。

> 已知宽度，绝对定位的居中 ，上下左右都为0，margin:auto

7.如何让一个div水平垂直居中

```
div {
position: relative / fixed; /* 相对定位或绝对定位均可 */
width:500px;
height:300px;
top: 50%;
left: 50%;
margin-top:-150px;
margin-left:-250px;
 外边距为自身宽高的一半 */
background-color: pink; /* 方便看效果 */
 }

.container {
display: flex;
align-items: center; /* 垂直居中 */
justify-content: center; /* 水平居中 */

}
.container div {
width: 100px; /* 可省 */
height: 100px; /* 可省 */
background-color: pink; /* 方便看效果 */
}
复制代码
```

8.如何清除浮动?

> clear清除浮动（添加空div法）在浮动元素下方添加空div,并给该元素写css样式 {clear:both;height:0;overflow:hidden;}

> 给浮动元素父级设置高度

> 父级同时浮动（需要给父级同级元素添加浮动）

> 父级设置成inline-block，其margin: 0 auto居中方式失效

> 给父级添加overflow:hidden 清除浮动方法

> 万能清除法 after伪类 清浮动（现在主流方法，推荐使用）

```
float_div:after{
content:".";
clear:both;
display:block;
height:0;
overflow:hidden;
visibility:hidden;
}
.float_div{
zoom:1
}
复制代码
```

9.css3实现三栏布局，左右固定，中间自适应

> 圣杯布局/双飞翼布局

```
<style>
       * {
           margin: 0;
           padding: 0;
       }
       .middle,
       .left,
       .right {
           position: relative;
           float: left;
           min-height: 130px;
       }
       .container {
           padding: 0 220px 0 200px;
           overflow: hidden;
       }
       .left {
           margin-left: -100%;
           left: -200px;
           width: 200px;
```

```
            background: red;
        }
        .right {
            margin-left: -220px;
            right: -220px;
            width: 220px;
            background: green;
        }
        .middle {
            width: 100%;
            background: blue;
            word-break: break-all;
        }
    </style>
</head>
<body>
    <div class='container'>
        <div class='middle'></div>
        <div class='left'></div>
        <div class='right'></div>
    </div>
</body>
复制代码
```

10.display:none 和 visibility: hidden的区别

> display:none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。

> visibility:hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。

11.CSS中 link 和@import 的区别是?

> link属于HTML标签，而@import是CSS提供的页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载

> import只在IE5以上才能识别，而link是HTML标签，无兼容问题

> link方式的样式的权重 高于@import的权重.

12.position的absolute与fixed共同点与不同点

> 共同点： 改变行内元素的呈现方式，display被置为block 让元素脱离普通流，不占据空间 默认会覆盖到非定位元素上

> 不同点： absolute的"根元素"是可以设置的 fixed的"根元素"固定为浏览器窗口。当你滚动网页，fixed元素与浏览器窗口之间的距离是不变的。

13..transition和animation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才会随时间改变属性值，并且transition为2帧，从from ....to，而animation可以一帧一帧的。

> transition 规定动画的名字 规定完成过渡效果需要多少秒或毫秒 规定速度效果 定义过渡效果何时开始 animation 指定要绑定到选择器的关键帧的名称

14.CSS优先级

```
不同级别：总结排序：!important > 行内样式>ID选择器 > 类选择器 > 标签 > 通配符 > 继承 > 浏览器默认属性
        1.属性后面加!import 会覆盖页面内任何位置定义的元素样式
        2.作为style属性写在元素内的样式
        3.id选择器
        4.类选择器
        5.标签选择器
        6.通配符选择器（*）
        7.浏览器自定义或继承
**同一级别：后写的会覆盖先写的**
复制代码
```

> css选择器的解析原则：选择器定位DOM元素是从右往左的方向，这样可以尽早的过滤掉一些不必要的样式规则和元素

15.雪碧图：

```
        多个图片集成在一个图片中的图
        使用雪碧图可以减少网络请求的次数，加快允许的速度
        通过background-position，去定位图片在屏幕的哪个位置
```

JS问题

第一章 初识JavaScript

JavaScript（JS）是什么?

- JavaScript一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言。
- JavaScript通常用来操作HTML页面，响应用户操作，验证传输数据等
- java和JavaScript有什么关系？java和JavaScript没有关系
- jQuery和JavaScript有什么关系？jQuery是由JS编写的一个js库。

JS代码写在哪里?

- 内嵌 js，
- 外链 js文件里面，利用src属性引入
- 标签属性里面（不推荐）
- script标签中的属性type="text/javascript"或language=”javascript”，HTML5新规则下可以什么都不用加；

- script标签可以放置于任何位置，不同的位置要注意加载顺序，通常放在head或body结束之前；

写JS代码需要注意什么?

- 严格区分大小写；
- 语句字符都是半角字符；（字符串里面可以使任意字符）
- 某些完整语句后面要写分号 (;)
- 代码要缩进，缩进要对齐。

JS里的系统弹窗代码

- alert('内容')
- confirm('确定?)
- prompt('请输入您的姓名:')

变量

很多时候，当我们重复使用某个元素或者某个数据时，内容可能太长或者数据要进行改变，这时就需要定义变量来代替他们。

语法：`var` + `变量名` 或者 `let` + `变量名` (var 和 let 声明的区别见后续章节)

Js中的注释

- 单行注释 //
- 多行注释 /* */

获取元素

- 特殊的标签

```
document.body
document.head
document.title
复制代码
```

- 一般标签

```
document.getElementById()       // 匹配ID名称…
ele.getElementsByTagName()      // 匹配标签名是…的集合动态方法
document.getElementsByName()    // 匹配name是…的集合  动态方法
ele.getElementsByClassName()    // 匹配class名称…的集合  动态方法
ele.querySelector()             // 匹配css选择器的第一个
ele.querySelectorAll()          // 匹配css选择器匹配的所有集合
复制代码
```

获取和修改元素HTML

- 元素HTML内容

```
// ele代表获取的元素
ele.innerHTML             // 获取元素HTML
```

```
ele.innerHTML = '字符串'  // 修改元素HTML
复制代码
```

- 元素文本内容

```
// 标准
ele.textContent             // 获取元素文本
ele.textContent = '字符串'  // 修改元素文本

// 非标准(ie低版本)
ele.innerText               // 获取元素文本
ele.innerText = '字符串'    // 修改元素文本
复制代码
```

第二章 函数、自定义属性、事件

变量与属性的区别

```
//此为变量
var a = 123;

//此时object 为节点对象 (node)
var object = document.getElementById('box');

//为object自定义了abc属性，且此属性的值888888
object.abc = 888888;

// 对象.属性名 可以获取属性值
object.abc;  //888888
复制代码
```

- 属性分为 : 节点属性 和 js属性
- 节点属性 : 元素自带属性
- js属性 : js中自定义的属性

当js属性为合法节点属性的时候，可以修改节点属性的值

```
//此时object 为节点对象 (node)
var object = document.getElementById('box')

// 修改节点对象object的id属性, id属性为节点对象的合法属性
var object.id = 'box';
复制代码
```

函数

函数申明

- 有名函数

```
// 此时abc就代表此函数，函数不会自动运行，需要调用运行。
function abc(){
    console.log(我是有名字的函数)
}
复制代码
```

带名字的函数，需函数名加小括号运行，如abc();

- 匿名函数

```
function (){}
复制代码
```

匿名函数，不进行赋值会报错

- 函数表达式

```
// 匿名函数
var a = function(){}
a() //此时用变量名加()来调用

//匿名函数
var b = function abc(){}
b() //此时用变量名加()来调用; 如果abc()调用会报错
复制代码
```

函数/对象方法

- 对象可以自定义属性
- 对象的属性，如果赋值的是一个函数function(){}, 称之为对象的方法

```
// 此时object 为节点对象 (nodelist)
var object = document.getElementById('box');
// 为object自定义了方法
object.abc = function(){};
复制代码
```

事件属性

事件 : 是当事人，在特定的时间在特定的地点发生了某事

- js中的事件: 元素.事件属性 = 事件函数

```
// 获取当前符合条件的节点
var object = document.getElementById('box');
// 给当前节点添加一个点击事件，配合这对应的处理函数
object.onclick = function(){}
复制代码
```

function () {}是一个固定的写法，在这个例子中，它被称之为事件函数。

事件函数也可以使用有名函数

```
var object = document.getElementById('box');
object.onclick =  fn;
function fn(){}
复制代码
```

- javascript中的事件 **[鼠标事件, 键盘事件, 表单事件, 系统事件, 等等]**

```
onclick ———— 点击（单击）事件
onmouseover ——— 鼠标滑入事件（会冒泡）
onmouseout————鼠标离开事件（会冒泡）
onmouseenter———鼠标滑入事件（不会冒泡）
onmouseleave———鼠标离开事件（不会冒泡）
ondblclick ————— 双击（单击）事件
复制代码
```

更多方法参考www.w3school.com.cn/tags/html_r...

- 函数中的this

事件函数里面的this就是指触发这个事件的元素

```
<script>
    box.onclick = function(){
       alert( this); //box,本函数是由box触发的
    }
    function fn(){
        //this指向window,因为fn()属于window
        // fn()其实是window.fn();是window让fn执行的
        alert( this );
    }
    fn();
</script>
复制代码
```

更多关于this的介绍请看后续章节

第三章 操作元素属性 CSS样式以及 []的使用

cssStyle 操作

- style 对象
- 复合样式改写 background-color ------> backgroundColor
- cssText
- tyle.float的兼容 cssFloat /styleFloat

attribute 属性节点

- 获取： getAttribute(名称)

```
优势：  用.和[]的形式无法操作元素的自定义属性 getAttribute可以操作元素的自定义属性
设置：  el.setAttribute(名称, 值)
包含：  el.hasAttribute(名称)
删除：  el.removeAttribute(名称)
复制代码
```

[]的运用

当我们需要使用字符串表示属性的时候，或者此时属性会变化的时候

```
obj.style.width = '100px';
//可改为
obj.style['width'] = '100px';
//亦可
var str = 'width';
obj.style[str] = '100px'
复制代码
```

第四章 javascript数据类型 判断 条件语句

javascript数据类型

七大数据类型 Number String Boolean Null Object Undefined **es6新增Symbol**

- number 数字

```
let num = 123
复制代码
```

- String 字符串

```
let str = '只会番茄炒蛋'
复制代码
```

- Boolean 布尔值

```
// true / false
let falg = true
复制代码
```

- Null 空

```
// 函数表达式
let abc = function(){}

// 函数声明/定义
```

```
funticon abc () {}
复制代码
```

- Object 对象

```
// (节点对象、自定义对象、array(数组)、json、function、系统对象)
复制代码
```

- Undefined 未定义
- Symbol

```
// Symbol是由ES6规范引入的一项新特性，它的功能类似于一种标识唯一性的ID。
复制代码
```

判断语句

判断语句返回布尔值

```
    ==  // 判断值
    >   // 大于
    <   // 小于
    <=  // 小于等于
    >=  // 大于等于
    !=  // 不等于
    === // 全等于 除了判断值,还会判断数据类型
    !== // 不全等
复制代码
```

if 条件语句

```
if ( 条件 ) {
    code // 这是条件 满足时执行的代码
}

// 如果 () 括号里面是true则运行{} 中代码
if ( 条件 ) {
    code 1 // 这是条件满足时执行的代码
} else {
    code 2 // 这是条件不满足时执行的代码
}
// 如果 () 括号里面是false 则运行 else 大括号中的代码, 总有一个会运行
if ( 条件一 ) {
    code 1 // 这是条件一满足时执行的代码
} else if (条件二) {
    code 2 // 这是条件二满足时执行的代码
} else {
    code 3  // 这是以上两种条件都不满足时执行的代码
}
复制代码
```

if () 中的条件会强制转化为布尔值，为false数据: false 0 '' null undefined NaN

三目运算

三目运算符的语法为conditions ? statementA : statementB ;

```
let num = 5
num = 5 ? 6 : 5
// 上面这段话的意思就是 num 等于5的话 值就改为6 不等于5的话值改为5
复制代码
```

switch case 条件语句

```
switch (data) {
    case 1:
        code1 // 执行代码
        break // break 来阻止代码自动地向下一个 case 运行。
    case 2:
        code2 //// 执行代码
        break // break 来阻止代码自动地向下一个 case 运行。
    default:
        与 case 1 和 case 2 不同时执行的代码
}
复制代码
```

break break 语句。它用于跳出 switch() 语句，来阻止代码自动地向下一个 case 运行

default 关键词来规定匹配不存在时做的事情

第五章 for循环 while后循环 do while前循环

for循环

- for () {}循环

```
for (初始值; 判断条件; 步幅) {
    code // 执行代码
}

for (let i = 0; i < 5; i++) { // i初始值为0；i是否小于5；每次循环后i加1
    console.log(i) // 0 1 2 3 4
}
复制代码
```

- continue 跳过本次循环

```
for (let i = 0; i < 5; i++) { // i初始值为0；i是否小于5；每次循环后i加1
    if  (i === 3) {
        continue
    }
```

```
    // 因为在i为3的时候跳过了此次循环，所有3没有被打印出来
    console.log(i) // 0 1 2 4
}
复制代码
```

- break的运用

```
for (let i = 0; i < 5; i++) { // i初始值为0；i是否小于5；每次循环后i加1
    if  (i === 3) {
          console.log(i) // 3
          break // 跳出中止循环，提升性能
    }
}
复制代码
```

变量自增/自减

- i++ 和 i-- **先赋值后自增或者自减**

```
let a = 10;
let b = a++ // 先把a的值赋值给b，然后在自增1
console.log(b) // 20
console.log(a) // 21
复制代码
```

- ++i 和 --i **先自增或者自减后赋值**

```
let a = 10;
let b = ++a // a的值先自增1，在赋值给b
console.log(b) // 21
console.log(a) // 21
复制代码
```

while 后循环 do while前循环

- while后循环

```
while (条件) {
    code // 这里写要执行的代码，条件满足不断执行
}
// 条件满足才会执行code代码
复制代码
```

- while前循环

```
do {
    code 1 // code 1会先执行一遍，然后在根据条件决定是否再执行code 1;
} while (条件) {
    code // 这里写要执行的代码，条件满足不断执行
}
复制代码
```

第六章 运算符 类型转换

算术运算

```
加————[+]
减————[-]
乘————[*]
除————[ / ]
取模/取余————[%]
复制代码
```

- 隐式类型转换

```
+ 在有字符串的时候,会进行字符串拼接
- * / % 会尽力把不是数字的转化为数字
复制代码
```

- NaN ———— not a number(不是一个数字)

```
不是数字的数字类型（number类型）
NaN和自己都不相等
isNaN( obj ) 判断是否为NaN,是返回true,否返回false;
复制代码
```

- 显示类型转化

转数字

```
Number() 可以用于任何数据类型转换成数值
parseInt()、parseFloat():专门用于把字符串转换成数值都是忽略前导的空格

1) Number()
能把字符串转化为数字。
如果字符串是空的（不包含任何字符），则将其转换为0
如果带非数字的字符串,返回NaN。
undefined,返回NaN。
true和false将分别转换为1和0。
null值，返回0。
var a = Number( '-100.02' );
console.log( a );       // -100.02
var a = Number( '100px' );
console.log( a );       // NaN
var a = Number( '' );
console.log( a );       // 0
var a = Number( undefined );
console.log( a );       // NaN
var a = Number( true );
console.log( a );       // 1
var a = Number( null );
console.log( a );       // 0

2) parseInt() （取整）取 非数字整前的数字 ，或小数点前的数字
```

```
3) parseFloat() 能取得小数，第二个小数点前的数字
复制代码
```

转字符串

```
String( obj );
obj.toString();
复制代码
```

赋值运算

```
=  +=  -=  *=  /=  %=  ++  --
复制代码
```

比较运算

```
<————小于
> ————大于
= = ———— 等于
<= ————小于等于
>= ———— 大于等于
!= ———— 不等于
= = =———— 全等，除了值的判断，还会进行unicode 编码的对比
!==————不全等
返回boolean值
复制代码
```

逻辑运算

```
|| ——— 逻辑或
&& ——— 逻辑与

赋值操作
let c = a || b // 如果a为true,则用a赋值, 如何a为false,则用b赋值

let c = a && b // 如果a为true,则通过，用 b 赋值,如果a为false,用 a 赋值

布尔值操作
if (a || b) {
    //如果a为true,则为true
    //如果a为false,则看b
}

if (a && b) {
    //如果a为true,则通过，看b,b为true则为true
    //如果a为false,则false
}

取反
if (!obj) {
    // 首先会把obj转化为布尔值，如果 obj是true,则!obj为false
```

```
}
复制代码
```

运算符优先级

JavaScript中的运算符优先级是一套规则。该规则在计算表达式时控制运算符执行的顺序。具有较高优先级的运算符先于较低优先级的运算符执行。

下图按从最高到最低的优先级列出JavaScript运算符。具有相同优先级的运算符按从左至右的顺序求值

第七章 函数[自执行] [传参] [return] getComputedStyle()

函数自执行

函数自执行方式,即创建立即调用一次

- 函数后面加用小括号,然后在用小括号包起来

```
(function(){}()) // 函数后面加用小括号,然后在用小括号包起来
复制代码
```

- 函数用小括号包起来,然后后面加小括号

```
(function(){})() // 函数用小括号包起来,然后后面加小括号
复制代码
```

- 函数后面加小括号,然后在函数前面加 + - ~ ！ 其中的一个符号

```
+function(){}()
-function(){}()
!function(){}()
~function(){}()
复制代码
```

函数传参

- 对应传参

形参:即形式参数，是用来接收函数调用时传递过来的数据，命名与变量一致

实参:即真实参数，是给形参传递的具体数据，任何数据类型都可以称为实参

```
function fn(a, b) { // a,b为形参，且a 为 20，b为10，一一对应
    console.log(a) // 20
    console.log(b) // 10
    console.log(a + b) // 30
}
fn(20, 10) // 20,10为实参
复制代码
```

- 不定参 arguments

不定参：实参个数不确定 arguments: 是所有实参的集合，arguments是一个类数组，arguments.length 可以返回实参个数

```
function fn() {
    console.log(arguments) // 返回一个包含实参的类数组
}
fn(20, 10, 5) // 20, 10, 5为实参
复制代码
```

关于什么是类数组,请看以后的章节

函数中的return

- 函数中默认return的返回值为undefined

```
function fn(a, b) {
    a + b
}
let a = fn(10, 20)
console.log(a) // undefined, 函数如果没有指定返回值,默认返回undefined
复制代码
```

- 自定义返回值

有时候我们需要函数中返回我们需要的值，这个时候return很有用

```
function fn(a, b) {
    return a + b
}
let a = fn(10, 20)
console.log(a) // 30
复制代码
```

return 返回的数据类型可以是任意类型

```
function fn(a, b) {
    a +b
    return function () {
        alert('ok')
    }
}
let a = fn(10, 20)
a() // 此时a就是返回的函数, a()打开了一个系统弹窗
复制代码
```

return 然后的代码不再执行，整个函数结束

```
function fn(a, b) {
    a +b
    return function () {
        alert('ok')
    }
    console.log('我不会被打印出来,因为上面有return')
}
fn()
复制代码
```

getComputedStyle()

getComputedStyle(obj,null)[cssStyle]获取计算后的样式对象,只读

```
<style>
    #elem-container{
    position: absolute;
```

```
    left:     100px;
    top:      200px;
    height:   100px;
    }
</style>
<div id="elem-container">dummy</div>
<script>
    let elem = document.getElementById("elem-container");
    let theCSSprop = window.getComputedStyle(elem,null)['left']
    console.log(theCSSprop) // 100px
</script>
复制代码
```

不要获取复合样式:如background

不要获取未设置的样式: 谷歌是具体宽度, ie是auto

兼容：ie8及以下 obj.currentStyle[cssStyle]

```
if (window.getComputedStyle) {
    return getComputedStyle(obj)[attr]
} else {
    return obj.currentStyle[attr]
}
复制代码
```

第八章 作用域 js预解析 闭包

作用域 脚本的有效范围，作用范围。分两大类：全局(script)和局部（function ）

全局(script)域

```
直接定义在script标签下的变量及函数，他们都作用在一个域，全局作用域，so..
<script>
    var a = 123;
    alert( window.a ); // 123
    function abc(){}
    alert( window.abc ); // function abc(){}
 </script>
复制代码
```

直接定义在script标签下的变量 称之为全局变量,script标签下的函数，称之为全局函数

全局变量及函数 都是window的一个属性,都能通过window.变量名访问

局部（function ）域

任何一个function(){},都会开启一个局部作用域

定义在function(){} 内部的变量称之为 局部变量

作用域链： 局部作用域内部可以访问父级作用域变量及全局作用域变量，也可以访问父级的函数，及全局函数 (往上爬)

```
let a = 10
function fn() {
    console.loa(a) // 10
}
复制代码
```

局部变量会覆盖父级（全局）变量，函数亦如此

```
let a = 10
function fn() {
    let a = 20
    console.loa(a) // 20
}
复制代码
```

javascript解析

javascript解析 即读取代码过程

- javascript解析 是 致上而下
- 预解析:正式解析前的工作，预解析过程会出现 变量提升，函数提升

```
function () {
    console.log(a) // undefined
    var a = 10
}
复制代码
```

- 变量提升

在作用域内声明的变量会被提升到作用域的顶部，且对其赋值undefined,这个过程称之为变量提升

```
上面的列子解析过程为
function() {
    var a = undefined
    console.log(a) // undefined
    var a = 10
}
复制代码
```

- 函数提升

在作用域内的函数定义函数会被提升到作用域的顶部，其值为其函数本身,这个过程称之为函数提升

```
function () {
    console.log(fn) // function fn () {}
    function fn () {}
```

```
}
复制代码
```

- var和函数重名函数优先，留下函数，函数和函数重名 后面定义的覆盖前面的-后来居高

```
console.log(a) // function a() { console.log(a2) }
var a = 10
function a() {
    console.log(a1)
}
function a() {
    console.log(a2)
}
a() // 报错
console.log(a) // 10
复制代码
```

- 不会提升的函数：在作用域内的函数表达式函数不会被提升到作用域的顶部，so ~

```
function () {
    console.log(fn) // undefined
    var fn = function () {}
}
复制代码
```

闭包

- js垃圾回收机制

```
js 中的 变量 函数 不再使用后，会被自动js垃圾回收机制回收
复制代码
```

- 形成闭包条件

```
条件一：  函数内部嵌套函数
条件二：  内部函数引用外部函数的 变量 参数
使用 return 返回了 此内部函数,上面的 变量 和参数 不会被回收

例如:
function fn(x) {
    var a = 5;
    function bibao() {
        var b = a + x
        console.log(x) // 20
        console.log(a) // 5
        console.log(b) // 25
    }
    return bibao
}
var c = fn(20)
console.log(c()) // 20 5 25
复制代码
```

第九章 字符串方法和数组

String 字符串

String即文本（字符串），字符串方法都不改原字符串；

创建字符串的三种办法: new String(), String(), 直接量，三种方式创建出来可以创建

```
var str = new String('hello')

var str = String('hello')

var str = 'hello' // 直接量
复制代码
```

string.length 属性可以返回字符串长度

string[index] 通过下标获取字符串

String方法

- str.concat(str,str...) 字符串拼接

```
用于把一个或多个字符串连接 到一块，返回拼接好的字符串
复制代码
```

- str.indexOf(value,index)查找字符串，返回查找字符串首次出现的位置;

```
方法对大小写敏感!
value 匹配字符
index 开始检索的位置, 合法值是 0 到 string.length - 1,默认0
匹配失败返回-1
复制代码
```

- str.charAt(index) 返回指定索引的字符串

```
var str = 'hello'
console.log(str.charAt(3)) // l
复制代码
```

- str.charCodeAt(index)返回指定索引的ASCII编码
- str.substring(start,end) 截取字符串，从start 开始，截止到end前，不包含end

```
如果没有end则从num开始整个查找
如果 start 比 stop 大，那么该方法在提取子串之前会先交换这两个参数。str.substring(1,4)
复制代码
```

- str.slice(start,end) 截取字符串，从start 开始，截止到end前，不包含end
- str.toLocaleUpperCase()/ str.toLocaleLowerCase()

```
str.toLocaleUpperCase() 把字符串转换为大写。
str.toLocaleLowerCase() 把字符串转换为小写。
复制代码
```

- str.replace(value/RegExp,new) 用一些字符替换另一些字符,new可以是字符串，也可以是函数
- str.split(value/RegExp,length-1) 方法用于把一个字符串分割成字符串数组, 返回分割后的数组
- str.search(value/RegExp)返回 检索字符串首次出现的位置;未找到返回-1
- str.match(value/RegExp)``查找指定的值，返回匹配的值。未找到返回null.正则可以找一个或多个表达式

更多字符串方法请见developer.mozilla.org/zh-CN/

Array 数组

创建数组的三种办法: new Array(), Array(), [] ， 三种方式创建出来都是一样的

```
var arr = new Array()

var arr = Array()

var arr = [] // 直接量
复制代码
```

- arr.length可以访问数组的长度
- 创建即指定数组长度Array(length)及 new Array(length),length是 数字的时候，创建的并不是数组的项，而是数组的长度，项的内容为undefined
- [] 通过数组索引，访问值

```
var arr = [1, 2, 3, 4, 5]
arr[0] // 1
复制代码
```

- 修改数组指定索引下的值

```
var arr = [1, 2, 3, 4, 5]
arr[0] = 8888
console.log(arr) // [8888, 2, 3, 4, 5]
复制代码
```

- 在数组后面添加项

```
var arr = [1, 2, 3, 4, 5]
arr.length = 8888
console.log(arr) // [1, 2, 3, 4, 5, 8888]
复制代码
```

- arr.indexOf(item) 查找项
- 数组去重

利用for循环给数组去除重复项

```
var arr = [1,2,3,4,5,6,5,4,3,2,1];
var arr2 = []
for (let i = 0; i < arr.length; i++) {
    if (arr2.indexOf(arr[i] == -1)) {
        arr2.push(arr[i])
    }
}
console.log(arr2) // [1, 2, 3, 4, 5, 6]
复制代码
```

Array() 数组方法

- arr.unshift(item1,item1,....) 向数组的头部添加一个或更多元素，并返回（新的长度）。
- arr.push(item1,item1,....) 向数组的尾部添加一个或更多元素，并返回（新的长度）。
- arr.shift( ) 删除数组的第一个元素（返回删除对象）;。
- arr.pop( ) 删除数组的最后一个元素（返回删除对象）。
- arr.splice(index,howmany,item1,.....,itemX) （删除/添加） 元素，然后（只返回删除对象）。

```
index 必需。整数，规定添加/删除项目的索引，可以使用负数,如果是添加，原有元素会往高位移动。
howmany 必需。要删除的项目数量。如果设置为 0，则不会删除项目。
item1, ..., itemX可选。向数组添加的新项目。
复制代码
```

- arr.sort() 排序

```
默认arr.sort() 以首字符编码大小排序
数组length小于10以冒泡排序
冒泡排序下依次比较,
return > 0 调换位置，= 0不调换位置，< 0 不调换位置
数组length大于10以二分排序
复制代码
```

- arr.reverse() 反转数组

以上方法不创建新的数组，而是直接修改原有的数组,同时索引会变化

以下方法会创建出一个新的数组, 而不是直接修改原数组

- arr.concat() 数组拼接

```
该数组是通过把所有 arrX 参数添加到 arr 中生成的。
如果要进行 concat() 操作的参数是数组，那么添加的是数组中的元素，而不是数组 —不修改原数组
复制代码
```

- arr.slice() 截取

```
arr.slice(start,end)方法从已有的数组中返回选定的元素
复制代码
```

- arr.join() 拼接成字符串
- Array.isArray( ) 判断是不是数组

ECMAscript5 的遍历数组方法

```
以下下方法都能实现遍历，语法也都一样，只是返回值不一样————————————不修改原数组
array.xxxx( function(currentValue,index,arr ), thisValue )
参数 描述
currentValue ————必须。当前元素的值
index ——————————可选。当期元素的索引值
arr————————————可选。当期元素属于的数组对象
thisValue ——————可选。对象作为该执行回调时使用，传递给函数，用作 "this" 的值。
如果省略了 thisValue ，”this” 的值为 “undefined”

function(currentValue, index,arr) 必须。函数，数组中的每个元素都会执行这个函数
复制代码
```

- forEach()

```
arr.forEach() 从头至尾遍历数组 ————————————————————————无返回值
复制代码
```

- map() 返回值数组

```
arr.map() 返回一个数组，包含函数所有返回值————————————————————返回数组
var arr = [1, 2, 3, 4]
var newArr = arr.map(function(x){
    return x * x
})
console.log(newArr) // [1, 4, 9, 16]
复制代码
```

- filter() true数组

```
arr.filter() 返回值是一个 return 值为true或能转化为true的值————————————返回数组
var arr = [1, 2, 3, 4]
var newArr = arr.filter(item => {
    return item > 3
})
console.log(newArr) // [4]
复制代码
```

- every()

```
arr.every() 针对所有元素，即都为true 则返回true————————————————————返回值
var  arr = [1,2,3,4];
var newArr = arr.every(item => {return item < 5});
console.log(newArr) // true, 因为数组的每一项都小于5

var newArr = arr.every(item => {return item < 3});
```

```
console.log(newArr) // false, 因为数组中的某一项不小于3
复制代码
```

- some()

```
arr.some() 是否存在 即有一个是true则为true————————————————————————返回值
var  arr = [1,2,3,4];
var newArr = arr.some(item => {return item % 2 === 0});
console.log(newArr) // true, 因为有偶数存在
复制代码
```

第十章 对象(JSON) for/in function[all apply bind]

JSON

- 创建对象（JSON） 对象是Javascript的基本数据结构，对象是引用类型 创建对象的三种方式 对象直接量，new Oject(), Object.create({})[ES5],create创建需要一个对象参数

```
// 对象都是一个key(键):value( 值 )一一对应

var obj = {} // 直接量
var obj = new Object()
var obj = Object.create()
复制代码
```

- 访问JSON的值

obj.attribute 和 obj[attribute]

```
var obj = {
    age: 20,
    name: '番茄炒蛋',
    sex: '男'
}
console.log(obj.age) // 20
console.log(obj[age]) // 20
复制代码
```

- 修改JSON的属性值

```
var obj = {
    name: '番茄炒蛋'
}
obj.name = '只会番茄炒蛋'
复制代码
```

- 添加JSON属性

```
var obj = {
    name: '番茄炒蛋'
```

```
}
obj.age = 20
复制代码
```

- 删除JSON属性

```
var obj = {
    name: '番茄炒蛋',
    age: 20
}
delete obj.name 或者 delete obj[name]
复制代码
```

- JSON数字属性

```
var obj = {
    name: '番茄炒蛋',
    age: 20
}
obj[1] = 'hello'
obj[2] = 'word'
复制代码
```

- in 判断对象是否存在某个属性

```
var obj = {
    name: '番茄炒蛋',
    age: 20
}
console.log('age' in obj) // true
复制代码
```

for in遍历json

- for in 遍历JSON

```
var obj = {
    name: '番茄炒蛋',
    age: 20
}
for (let attr in obj) { //attr 为属性，attr不是必须的，可以为任意变量名
    console.log(attr) // 属性名 name age
    console.log(obj[attr]) // 对应的属性值 '番茄炒蛋' 20
}
复制代码
```

- for in 也可以遍历数组

```
var arr = [1, 2, 3, 4]

for (let attr in arr) { //attr 为属性，attr不是必须的，可以为任意变量名
    console.log(attr) // 下标
```

```
    console.log(obj[attr]) // 对应下标的值 1 2 3 4
}
```
复制代码

for循环不能遍历JSON

JSON对象仿jQuery 链式操作 css html

```
function $ (option) {
    var t = typeOf option
    if (t == 'function') {
        window.onload = option
    } else if (t.toLowerCase() == 'string') {
        var ele = option.subString(1, option.length)
        el = document.getElementById(ele)
    }
    var obj = {
        css: function (attr, val) {
            el.style[attr] = val
            return obj;
        },
        html: function (val) {
            el.innerHTML = val
            return obj
        }
    }
    return obj
}
$('#box').css('backgroundColor','red').html('hello');
```
复制代码

JSON.parse() 对象化 / JSON.stringify() 对象字符化

- JSON.parse() JSON.parse(obj)方法解析一个JSON字符串，构造由字符串描述的JavaScript值或对象。可以提供可选的reviver函数以在返回之前对所得到的对象执行变换。

```
var obj = '{
    "name": "只会番茄炒蛋",
    "age": 10,
    "sex": "男"
}'

JSON.parse(obj)
// 解析后的值为:
obj = {
    name: "只会番茄炒蛋",
    age: 10,
    sex: "男"
}
```
复制代码

- JSON.stringify() JSON.stringify(obj)与JSON.parse()进行的是反操作

```
JSON.stringify({});                   // '{}'
JSON.stringify(true);                 // 'true'
JSON.stringify("foo");                // '"foo"'
JSON.stringify([1, "false", false]);  // '[1,"false",false]'
JSON.stringify({ x: 5 });             // '{"x":5}'
JSON.stringify({x: 5, y: 6});         // "{"x":5,"y":6}"
复制代码
```

Function call() applay() bind()方法

- call()和apply都用于函数调用

```
function fn () {
    console.log(this)
}
fn() // window
fn.call('hello') // String {"hello"}
fn.call(123) // Number {123}
复制代码
```

区别

call(thisvalue, val1，val2，....)

```
// thisvalue 是函数内部this的值
// 后面是参数列表
复制代码
```

apply(thisvalue, [val1，val2，....])

```
// thisvalue 是函数内部this的值
// 后面是参数数组，所有参数放数组里面
复制代码
```

- bind()都用于创建中

```
1) 适用于匿名函数
var fn = function (a, b) {
    console.log(this, a, b)
}.bind('hello', 1, 2)
fn() // String {"hello"} 1 2

2)有名函数,有些特殊
function fn() {
    console.log(this)
}
fn.bind('hello')() // String {"hello"}

3)自执行函数
(function fn() {
```

```
    console.log(this)
}.bind('hello')())  // String {"hello"}

(function fn() {
    console.log(this)
}.bind('hello'))() // String {"hello"}

(function fn() {
    console.log(this)
}).bind('hello')() // String {"hello"}
复制代码
```

第十一章 定时器 Math函数

定时器

- setInterval()

setInterval(function(){}, 1000) 多用于动画

第一个参数是一个函数

第二个参数是事件, 表示1秒(1000毫秒)后调用一次, 然后每个1秒调用执行一次第一个函数里面的内容

```
1) 一般使用
var a = 0;
setInterval(function () {
    a++;
    console.log(a) // 每隔一秒打印a 并且a在自增
}, 1000)

var a = 0;
function fn() {
    a++;
    console.log(a)
}
setInterval(fn, 1000)  // 和上面的写法数据一样

2)第一个参数fn 与 fn()的区别, fn()会不等延迟直接调用, 后面不在调用
var a = 0;
function fn() {
    a++;
    console.log(a)
}
setInterval(fn(), 1000)  // 1 打印1,然后就不在调用

3) 带return值的fn
var a = 0;
function fn() {
    a++;
    console.log(a)
    return function(){console.log('ok')}
}
```

```
setInterval(fn(), 1000) // 1 打印1,然后就不在调用
复制代码
```

- clearInterval() 清除定时器

```
clearInterval(timerManger) 里面的参数是定时管理器
var timer = setInterval(function(){}, 1000) // 设置变量timer为定时管理器
clearInterval(timer) // 清除timer定时管理器
复制代码
```

- setTimeout() 一次定时器

setTimeout(function(){},1000)

第一个参数是一个函数

第二参数是时间，表示1秒（1000毫秒）后调用一次，然后不再调用

```
var a = 0;
setTimeout(function () {
    a++;
    console.log(a) // 1 只有一次打印
})
复制代码
```

- clearTimeout() 清除定时器

```
clearTimeout(timerManger) 里面的参数是定时管理器
var timer = clearTimeout(function(){}, 1000) // 设置变量timer为定时管理器
clearTimeout(timer) // 清除timer定时管理器
复制代码
```

Math 数字函数

Math对象用于执行数学任务 Math对象 无需new，直接调用Math方法就行

- Math.random() 求随机值 左闭右开区间

```
// 随机 0~1之间的数
var rand = Math.random()
console.log(rand) // 0~1之间的数

// 随机 5~10之间的数
var rand =  Math.random() *(10-5) + 5;
console.log(rand) // 5~10之间的数

// 封装随机x至y之间的数
function random(x, y) {
    var rand = x + Math.random() * (y - x)
    return rand
```

```
}
复制代码
```

- Math.round()————四舍五入

```
var num = 12.6
Math.round(num) // 13

var num = 12.3
Math.round(num) // 12
复制代码
```

- Math.ceil() ————向上取整 (上舍入)
- Math.floor()————向下取整 (下舍入)
- Math.abs()————求绝对值
- Math.pow(x,y)————x的y次幂（x的y次方）
- Math.sqrt(x) ————返回数的平方根
- Math.max(x,y,z...)——-求x和y的最大值
- Math.min(x,y,z...)——-求x和y的最小值

Math方法二

“度”的定义是，“两条射线从圆心向圆周射出，形成一个夹角和夹角正对的一段弧。当这段弧长正好等于圆周长的360分之一时，两条射线的夹角的大小为1度。（如图1） 弧度的定义是：两条射线从圆心向圆周射出，形成一个夹角和夹角正对的一段弧。当这段弧长正好等于圆的半径时，两条射线的夹角大小为1弧度。

角所对的弧长是半径的几倍，那么角的大小就是几弧度。

它们的关系可用下式表示和计算：

(弧度)= 弧长 / 半径

圆的周长是半径的 2π倍，所以一个周角（360度）是 2π弧度。

- 度跟弧度之间的换算

```
据上所述，一个平角是 π 弧度。
即 180度=π弧度
由此可知：
弧度=π/180度 （ ≈0.017453弧度 ）
复制代码
```

- Math.sin（弧度） 正弦 对边比斜边 一个以弧度表示的角
- Math.cos（弧度）余弦 邻边比斜边 是 -1.0 到 1.0 之间的数
- Math.PI

```
Math.PI 即π 是圆的周长和它的直径之比。这个值近似为 3.141592653589793
一弧度 = π/180；将角度乘以（2PI/360） 0.017453293 即可转换为弧度
复制代码
```

第十二章 日期对象Date

日期

- new Date() 本地时间

```
var d = new Date()
console.log(d) // Mon Sep 16 2019 15:48:31 GMT+0800 (中国标准时间)
复制代码
```

- toUTCString() 当前 世界时

toUTCString() 根据世界时，把 Date 对象转换为字符串。

```
var d = new Date();
var utc =  d.toUTCString()
console.log(ytc) // "Mon, 16 Sep 2019 07:48:31 GMT"
复制代码
```

- 获取具体时间

```
getFullYear()       // 年
getMonth()          // 月( 0 ~ 11 )
getDate()           // 天( 1 ~ 31 )
getDay()            // 星期( 0 ~ 6 )
getHours()          // 时
getMinutes()        // 分
getSeconds()        // 秒
getMilliseconds()   // 毫秒
getTime()           // 返回 1970 年 1 月 1 日至今的毫秒数
复制代码
```

日期格式化

var date = new Date()

- date.toLocaleString() ————————按照本地时间输出
- date.toLocaleDateString() ————————本地时间 年 月 日
- date.toLocaleTimeString() ————————本地时间 时 分 秒
- date.toTimeString()————————————本地 时 分 秒 时区
- date.UTC() ————————————————世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数

更多方法参考www.w3school.com.cn/tags/html_r...

cookie，sessionStorage，localStorage

1、cookie，sessionStorage，localStorage是存放在客户端，session对象数据是存放在服务器上 实际上浏览器和服务器之间仅需传递session id即可，服务器根据session-id找到对应的用户session对象 session存储数据更安全一些，一般存放用户信息，浏览器只适合存储一般的数据 2、cookie数据始终在同源的http请求中携带，在浏览器和服务器来回传递，里面存放着session-id

sessionStorage，localStorage仅在本地保存 3、大小限制区别，cookie数据不超过4kb，localStorage在谷歌浏览中2.6MB 4、数据有效期不同，cookie在设置的（服务器设置）有效期内有效，不管窗口和浏览器关闭 sessionStorage仅在当前浏览器窗口关闭前有效，关闭即销毁（临时存储） localStorage始终有效

SessionStorage和localStorage区别： 1.sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在用一个会话的页面中才能被访问（也就是说在第一次通信过程中） 并且在会话结束后数据也随之销毁，不是一个持久的本地存储，会话级别的储存 2.localStorage用于持久化的本地存储，除非主动删除数据，否则不会过期

token、cookie、session三者的理解？？？！！！

```
1、token就是令牌，比如你授权(登录)一个程序时,他就是个依据,判断你是否已经授权该软件（最好的身份认证，安全性好，且是唯一的）
    用户身份的验证方式

2、cookie是写在客户端一个txt文件，里面包括登录信息之类的，这样你下次在登录某个网站，就会自动调用cookie自动登录用户名
    服务器生成，发送到浏览器、浏览器保存，下次请求再次发送给服务器（存放着登录信息）

3、session是一类用来客户端和服务器之间保存状态的解决方案，会话完成被销毁（代表的就是服务器和客户端的一次会话过程）
    cookie中存放着sessionID，请求会发送这个id。sesion因为request对象而产生。
复制代码
```

基于Token的身份验证：（最简单的token: uid用户唯一的身份识别 + time当前事件戳 + sign签名）

```
  1、用户通过用户名和密码发送请求
  2、服务器端验证
  3、服务器端返回一个带签名的token，给客户端
  4、客户端储存token，并且每次用于发送请求
  5、服务器验证token并且返回数据
  每一次请求都需要token
复制代码
```

cookie与session区别

```
  1、cookie数据存放在客户的浏览器上，session数据放在服务器上。
  2、cookie不是很安全，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。
  3、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。
  4、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。
复制代码
```

session与token区别

```
  1、session认证只是把简单的User的信息存储Session里面，sessionID不可预测，一种认证手段。只存在服务端，不能共享到其他的网站和第三方App
  2、token是oAuth Token，提供的是认证和授权，认证针对用户，授权是针对App，目的就是让某APP有权访问某用户的的信息。Token是唯一的，
     token不能转移到其他的App，也不能转到其他用户上。（适用于App）
  3、session的状态是存在服务器端的，客户端只存在session id， Token状态是存储在客户端的
复制代码
```

Cookie的弊端有哪些？？？（优势：保存客户端数据，分担了服务器存储的负担）

1、数量和长度的限制。每个特定的域名下最多生成20个cookie（chorme和safari没有限制）
2、安全性问题。

设计模式

> 一、观察者模式：juejin.im/post/5a14e9... juejin.im/post/5af05d... 在软件开发设计中是一个对象(subject)，维护一系列依赖他的对象（observer），当任何状态发生改变自动通知他们。强依赖关系 简单理解：数据发生改变时，对应的处理函数就会自动执行。一个Subjet,用来维护Observers,为某些event来通知（notify）观察者

> 二、发布-订阅者 有一个信息中介，过滤 耦合性低 它定义了一种一对多的关系，可以使多个观察者对象对一个主题对象进行监听，当这个主题对象发生改变时，依赖的所有对象都会被通知到。

- -两者的区别： 1.观察者模式中，观察者知道Subject ,两者是相关联的，而发发布订阅者只有通过信息代理进行通信 2.在发布订阅模式中，组件式松散耦合的。正好和观察者模式相反。 3.观察者大部分是同步的，比如事件的触发。Subject就会调用观察者的方法。而发布订阅者大多数是异步的（） 4.观察者模式需要在单个应用程序地址空间中实现，而发布订阅者更像交叉应用模式。

数据结构和算法

一、两个栈实现一个队列，两个队列实现一个栈 www.cnblogs.com/MrListening...

二、红黑树（解决二叉树依次插入多个节点时的线型排列） juejin.im/post/5a27c6...

三、最小栈的实现（查找最小元素，用两个栈配合栈内元素的下标）juejin.im/post/5a2ff8...

四、十大排序

```
1.冒泡排序：重复走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把它们交换过来。
  实现过程：1.比较相邻的元素。如果第一个比第二个大，就交换他们两个
            2.对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样在最后的元素应该会是最大的数
            3.针对所有的元素重复以上的步骤，除了最后一个
            4.重复步骤1-3，直到排序完成。
2.选择排序：首先在未排序序列中找到最小值，放在排序序列的起始位置，然后，在从剩下未排序元素中继续寻找最小值，然后放在与排序序列的末尾
  实现过程：

3.插入排序：构建有序序列，对于未排序数据，在已排序序列中冲后向前扫描，找到相应位置并插入
  实现过程：1.从第一个元素开始，该元素可以认为已经被排序
            2.取出下一个元素，在已排序的元素序列中冲后向前扫描
            3.如果该元素（以排序）大于新元素，将元素向后移一位
            4.在取出一个元素，比较之前的，直到找到自己合适的位置

4.桶排序：将数据分布到有限数量的桶里，每个桶在分别排序

1.快速排序：快速排序使用分治法把一个串（list）分为两个子串（sub-lists）.具体算法实现
  实现过程：1.从数组中挑出一个元素，成为一个基准
            2.重新排列数组，所有元素比基准小的摆在基准前面，所有元素比基准大的摆在基准后面（相同的可以摆在一边）
                这个分区退出之后，该基准就处于数列的中间位置。成为分区操作。
            3.递归的把小于基准值的子数列和大于基准值元素的子数列排序
算法实现：  function quickSort (arr) {
                if  (arr.length <= 1)  {return arr}
                var destIndex = Math.floor(arr.length/2)
                var left = [], right = [];
```

```
                var dest = arr.splice(destIndex,1)[0];
                for (var i =0;i<arr.length;i++){
                        if (arr[i]<dest) {
                        left.push(arr[i])
                        } else {
                        right.push(arr[i]) }
                return quickSort(left).concat([dest],quickSort(right)
```

2.堆排序：利用对这种数据结构所涉及的一种排序算法，堆积是一个近乎完全二叉树的结构，并同时满足堆积的性质：即子节点的键值或索引总是小于（或大于）
实现过程：1.
复制代码

五、数组去重 juejin.im/post/5aed61...

```
1.双重循环
2.indexOf
3.数组排序去重 最快你Olong
复制代码
```

六、字符串

```
判断回文字符串：（递归的思想）
        1.字符串分隔，倒转，聚合[...obj].reverse().join('')
        2.字符串头部和尾部，逐次向中间检测
                实现：function isPalindrome(line) {
                        line += '';
                        for (var i=0,j=line.length-1;i<j;i++,j--) {
                                if (line.chartAt(i) !== line.chartAt(j) {
                                return false
                        }

        3.递归
复制代码
```

七、二分查找（有序数组的查找）

```
二分查找可以解决已排序数组的查找问题，即只要数组中包含T(要查找的值)，那么通过不断的缩小包含T的数据范围，就可以最终要找到的数
（1）一开始,数据范围覆盖整个数组。
（2）将数组的中间项与T进行比较，如果T比数组的中间项小，则到数组的前半部分继续查找，反之，则到数组的后半部分继续查找。
（3）就这样，每次查找都可以排除一半元素，相当于范围缩小一半。这样反复比较，反复缩小范围，最终会在数组中找到T
代码实现：function binarySearch (data, dest, start, end){
                var end = end || data.length-1;
                var start = start || 0;
                var m = Math.floor((start+end)/2);
                if (dest<data[m]){
                        return binarySearch(data, dest, 0, m-1)
                } else {
                        return binarySearch(data, dest, m+1, end)
                }}
                return false
```

结语

感谢您的观看，如有不足之处，欢迎批评指正。