# 一、 内置对象

## 1） 对象

对象是由属性和方法组成的,使用点语法访问

```
<script>
        // js中的对象
        // name age叫做对象的属性
        var obj={name:"Maria",age:20};
        console.log(obj);
        console.log("name的值为",obj.name);
        console.log("age的值为",obj.age);
        obj.age=18;
        console.log("修改过的age的值为",obj.age);
</script>
```

```
<body>
    <table>
        <tr>
            <td>姓名</td>
            <td>年龄</td>
        </tr>
        <tr id="content">
            <td id="name1"></td>
            <td id="age1"></td>
        </tr>
    </table>
```

```
    <script>
        // js中的对象
        // name age叫做对象的属性
        var obj={name:"Maria",age:20};
        // 找到id=content的元素
        // 拼接一个字符串，将字符串添加到content中
        var content=document.getElementById("content");
        var html="";
        html+="<td>"+obj.name+"</td>";
        html+="<td>"+obj.age+"</td>";
        content.innerHTML=html
    </script>
</body>
```

# 2） Array 数组

## 1. 创建

## 2. 特点

- 数组用于存储若干数据,自动为每位数据分配下标,从0开始
- 数组中的元素不限数据类型,长度可以动态调整
- 动态操作数组元素： 根据元素下标读取或修改数组元素， arr[index]

## 3. 属性和方法

1. 属性 : length 表示数组长度,可读可写
2. 方法 :

- push(data)
  在数组的末尾添加一个或多个元素,多个元素之间使用逗号隔开
  返回添加之后的数组长度

```
<script>
        var arr=[4,0,7,9,0,0,2,5,3,1]
    var arr2=new Array();
        // 将arr中的非0值存入arr2中
        for(var i=0;i<arr.length;i++){
            if(arr[i]!=0){
            arr2.push(arr[i])
            }
        }
        console.log(arr2)
</script>
```

- pop()
  移除末尾元素
  返回被移除的元素
- unshift(data)
  在数组的头部添加一个或多个元素
  返回添加之后的数组长度

- shift()
  移除数组的第一个元素
  返回被移除的元素
- splice(index,num)

  从数组中添加/删除项目

  返回被删除的项目

  splice(0,0,1):从索引位为0的位置删除0个元素，然后用元素1替换被删除的元素，相当于添加元素。
- toString()
  将数组转换成字符串类型
  返回字符串结果
- join(param)
  将数组转换成字符串,可以指定元素之间的连接符,如果参数省略,默认按照逗号连接
  返回字符串

```
arr.join("-")
"1-2-123"
```

- reverse()
  反转数组,倒序重排
  返回重排的数组,注意该方法**直接修改原数组**的结构
- sort()
  对数组中元素排序,默认按照Unicode编码升序排列
  返回重排后的数组,**直接修改原有数组**
  参数 : 可选,自定义排序算法
  例:

```
//自定义升序
function sortASC(a,b){
  return a-b;
}
```

作用：作为参数传递到sort()中,会自动传入两个元素进行比较,如果a-b>0,交换元素的值,**自定义升序排列**

```
//自定义降序
function sortDESC(a,b){
  return b-a;
}
//如果返回值>0,交换元素的值,b-a表示降序排列
```

## 4. 二维数组

数组中的每个元素又是数组

```
var arr1 = [1,2,3];
var arr2 = [[1,2],[3,4],[5,6,7]];
//操作数组元素
var r1 = arr2[0] //内层数组
var num = r1[0]; //值 1
//简写
var num2 = arr2[1][0];
```

# 3）String 对象

## 1. 创建

```
    var str = "100";
    var str2 = new String("hello");
```

## 2. 特点

字符串采用数组结构存储每位字符,自动为字符分配下标,从0开始

## 3. 属性

length： 获取字符串长度

```
<script>
        var str1="nihao";
        console.log(str1); //nihao
        str1[2]="e";       //不报错也不生效
        console.log(str1); //nihao
</script>
```

## 4. 方法

- 转换字母大小写
  toUpperCase() 转大写字母
  toLowerCase() 转小写字母
  返回转换后的字符串,不影响原始字符串
- 获取字符或字符编码
  charAt(index)　获取指定下标的字符
  charCodeAt(index) 获取指定下标的字符编码

参数为指定的下标,可以省略,默认为0

```
```html
"s".charCodeAt()
115
```
```

- 获取指定字符的下标
  - indexOf(str,fromIndex)
    作用 : 获取指定字符的下标,从前向后查询,找到即返回
    参数 :
    str 表示要查找的字符串,必填
    fromIndex 表示起始下标,默认为0
    返回 :
    返回指定字符的下标,查找失败返回-1

```
str="this is a test string"
str.indexOf("i")
2
str.indexOf("i",8)
18
```

- 截取字符串

  substring(startIndex,endIndex)

  作用 : 根据指定的下标范围截取字符串,startIndex ~ endIndex-1

  参数 :

  startIndex　表示起始下标

  endIndex　表示结束下标,可以省略,省略表示截止末尾

```
str
"this is a test string"
str.substring(10,14)
"test"
```

- substr(startIndex,len)

  作用:  根据下标截取指定的字符串

```
str
"this is a test string"
str.substr(10,14)
"test string"
```

- 分割字符串

  split(param)

  作用 : 将字符串按照指定的字符进行分割,以数组形式返回分割结果

  参数 : 指定分隔符,必须是字符串中存在的字符,如果字符串中不存在,分割失败,仍然返回数组

```
<script>
        var str="2019-12-31";
        var year=str.substr(0,4);//2019
        var month=str.substr(5,2);//12
        var day=str.substr(8,2);//31

        var arr=str.split("-");// ["2019", "12", "31"]
        var str1=arr.join("-");// 2019-12-31

</script>
```

- 模式匹配
- 正则表达式对象 RegExp

  RegExp : Regualr Expression

  1. 语法 :

     var reg1 = /匹配模式/ig;

     var reg2 = new RegExp('匹配模式','修饰符');

     正则表达式对象可以接收一个变量。

```
var reg1=/test/ig;
var reg2=new RegExp("test","ig");
```

2. 属性：

   lastIndex : 可读可写，表示下一次匹配的起始索引
   注意：

   1. 默认情况下，正则表达式对象不能重复调用方法，
      如果重复调用，结果会出错：
      由于 lastIndex 保存再一次匹配的起始下标，
      重复调用时，不能保证每次都从下标0开始
      验证，可以手动调整 lastIndex 为 0。
   2. 只有正则对象设置全局匹配 g ， 该属性才起作用。
3. 方法：

   test(str) :验证字符串中是否存在满足正则匹配模式的内容，存在则返回true，

   不存在返回false参数为要验证的字符串。

- 作用 : 借助正则表达式实现字符串中固定格式内容的查找和替换
  正则表达式 :
  var reg1 = /字符模式/修饰符;
  修饰符 :
  i : ignorecase 忽略大小写
  g : global 全局范围
  字符串方法 :
  - match(regExp/subStr)
    作用 : 查找字符串中满足正则格式或满足指定字符串的内容
    返回 : 数组,存放查找结果
  - replace(regExp/subStr,newStr)
    作用 : 根据正则表达式或字符串查找相关内容并进行替换
    返回 : 替换后的字符串,不影响原始字符串。

```
<script>
        var reg1=/test/ig;
        var reg2=new RegExp("test","ig");
        var str="this is test thing";
        // 调用正则表达式的test方法
        // 如果能匹配到内容返回true，否则返回else
        console.log(reg1.test(str));//true
        console.log(reg2.test(str));//true
</script>

<script>
        var reg1=/test/ig;
        var reg2=new RegExp("test","ig");
        var str="this is test thing";
        // 调用正则表达式的test方法
        // 如果能匹配到内容返回true，否则返回else
        console.log(reg1.lastIndex);//0
        console.log(reg1.test(str));//true
        console.log(reg1.lastIndex);//12
        console.log(reg1.test(str));//false
    </script>

<script>
        var reg1=/test/ig;
        var reg2=new RegExp("is","ig");
        var str="this is test thing";
        // 字符串的match()方法
        console.log(str.match(reg2));
</script>
```

```
<script>
        var reg1=/test/ig;
        var reg2=new RegExp("is","ig");
        var str="this is test thing";
        // 字符串的replace()方法
        console.log(str.replace(reg2,"**"));//th** ** test thing
</script>
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        #content{
            width:250px;

        }
        #p1 span{
            color:red;
        }
    </style>
</head>
<body>
    <div id="content">
        <input type="text" id="txt">
        <button id="btn">查找</button>
        <p id="p1">
            Lorem, ipsum dolor sit  <span>amet</span> consectetur
adipisicing elit. Laboriosam corporis magnam nostrum eum explicabo est
repellendus quisquam neque exercitationem nisi unde doloribus itaque
similique molestiae autem ab, consequuntur placeat esse.
        </p>
    </div>
    <script>
        txt=document.getElementById("txt");
        p1=document.getElementById("p1");
        btn=document.getElementById("btn");
        btn.onclick=function(){
            var val=txt.value;
            var pval=p1.innerText;
            var reg1=new RegExp(val,"ig")
            p1.innerHTML=pval.replace(reg1,"<span>"+val+"</span>");

        };
    </script>
</body>
```

# 4) Math 对象

## 1. 定义

Math对象主要提供一些列数学运算的方法

## 2. 属性

1. 圆周率： Math.PI
2. 自然对数 : Math.E

## 3. 方法

1. Math.random(); 生成0-1之间的随机数
2. Math.ceil(x); 对x向上取整,忽略小数位,整数位+1
3. Math.floor(x); 对x向下取整,舍弃小数位,保留整数位
4. Math.round(x); 对x四舍五入取整数

# 5）日期对象

## 1. 创建日期对象

```
1. var date2 = new Date("2011/11/11");
2. var date3 = new Date("2011/11/11 11:11:11");
```

## 2. 日期对象方法

1. 读取或设置当前时间的毫秒数：getTime()
2. 获取时间分量
   - getFullYear()
   - getMonth()
   - getDate()

```
<script>
        var dt=new Date();
        console.log(dt);
        // 使用日期对象方法获取 年月日 时分秒
        var y=dt.getFullYear();
        var m=dt.getMonth()+1; //月份值从0开始
        var d=dt.getDate();
        var h=dt.getHours();
        var M=dt.getMinutes();
        var s=dt.getSeconds();
        console.log(y,m,d,h,M,s);
        // 创建指定日期的对象
        var dt2=new Date("2020-1-3 0:0:0");
        console.log(dt2);
        // 计算时间差
        var t=dt2-dt;//结果是毫秒数
        console.log("从现在到1月3日凌晨还有"+t+"毫秒");
        // 求还有多少小时
        h=Math.floor(t/(3600*1000));
        console.log(h);
    </script>
```