day07 日期对象及对象讲解

日期对象概述

日期对象是用于表示日期时间的一个对象，他里面包含对应设置日期时间及获取日期时间的方法。

日期对象的声明

使用new关键词声明

```
new Date()
```

无参构造声明（构建对象的方法叫做构造函数）

- 构造函数首字母大写
- 使用new关键词调用构造函数产生新的对象

```
//无参构造声明是获取当前的本地时间
var date = new Date()
//他是以本地的时间格式进行显示的 UTC（构建标准时间）
console.log(date)
```

传入字符串作为参数（格式不正确会出现Invalid Date）

```
//传递对应的字符串 以字符串转为对应的时间
var date = new Date('2000/2/14 10:43:10')
console.log(date)
```

传递多个number类型的参数（如果值超出自动向上递增）

```
//传递多个数字 最多支持传递6个参数 年 月 日 时 分 秒
//传递的月份 从0开始到11
var date = new Date(2000,2,12,10,40,10)
console.log(date)//3月12
var date = new Date(2022,10,10)
console.log(date)//2022年11月10日 00:00:00
```

传递毫秒值来构建

```
//传递一个参数为number值 他将number值识别为毫秒 + 格林兰治时间（1970/1/1 00:00:00）
var date = new Date(2000)
console.log(date)
```

日期对象的比对

日期对象在对比的过程中会自动转为毫秒值进行相关比对计算

```
var date2 = new Date()
console.log(date2 - date1)//毫秒值
console.log(date2+date1)//连接操作
console.log(date2*date1)//毫秒值计算
```

日期对象的相关方法

get开头 获取

- 获取年 getYear
- 获取月 getMonth （月份0-11）
- 获取日 getDate
- 获取星期几 getDay （星期天是0）
- 获取时 getHours
- 获取分 getMinutes
- 获取秒 getSeconds
- 获取毫秒 getMilliseconds
- 获取时间戳 getTime
- 获取时区偏移量 getTimezoneOffset

```
var date = new Date()
//获取的相关方法
console.log(date.getFullYear()) //获取年
console.log(date.getMonth()) //获取月 （0-11）
console.log(date.getDate()) //当前日
console.log(date.getDay()) //获取星期几 （星期天为0）
console.log(date.getHours()) //获取小时 （0-23）
console.log(date.getMinutes()) //获取分钟 （0-59）
console.log(date.getSeconds()) //获取秒中 (0-59)
console.log(date.getMilliseconds()) //获取毫秒 （0-999）
console.log(date.getTime()) //离格林兰兹时间的毫秒值 时间戳（唯一标签）
console.log(date.getTimezoneOffset())//获取时区
```

获取UTC时间（抛除时区的影响）

- getUTCHours
- ....

set开头 设置

- setFullYear 设置年份
- setMonth 设置月份
- setDate 设置天
- setHours 设置小时
- setMinutes 设置分钟
- setSeconds 设置秒钟
- setMilliseconds 设置毫秒

```
var date = new Date();
date.setFullYear(2000,11,12) //设置年份 还可以传入月份及日期
date.setMonth(10) //设置月份 还可以传入日期
date.setHours(16) //设置小时 还可以传入分秒毫秒
date.setDate(10) //设置天
date.setMinutes(10) //设置分钟 还可以传入对应的秒 毫秒
date.setMilliseconds(10) //设置毫秒
date.setSeconds(10) //设置秒 还可以传入毫秒
console.log(date)
```

设置UTC时间（抛除时区的影响）

- setUTCHours
- ....

转字符串相关方法

```
var date = new Date()
console.log(date.toString())
console.log(date.toDateString()) //转日期
console.log(date.toTimeString()) //转时间
console.log(date.toLocaleString()) //根据本地格式转为字符串 （可以进行格式化）
console.log(date.toLocaleDateString()) //根据本地格式转为日期字符串
console.log(date.toLocaleTimeString()) //根据本地格式转为时间字符串
console.log(date.toUTCString()) //返回UTC时间字符串
console.log(date.toISOString()) //遵从ISOS协议的字符串
```

练习

设计一个函数返回俩个时间之间间隔的天数

设计一个函数返回一个时间七天后的时间 以YYYY-mm-dd hh:MM:ss显示

计算今年一共过了几个星期

打印输出今年最后一个星期天的时间

对象

概述

对象是一个存储的容器，他是一个引用的数据类型。他里面可以存储一些属性（值）及方法（函数其实就是对应的动作），他里面存储是以**key:value**的形式进行存储的（key是唯一标识不可以重复，value是值），**可以通过key来访问对应的value。**

对象的声明（Object）

赋值声明

```
var obj = {} //空对象
console.log(obj)
```

以new关键词来调用构造函数进行声明（new Object）

```
var obj = new Object()//空对象
console.log(obj)
```

Object属于顶层父类 所有的其他对象都是他的子类

对象在比较的时候比较的是地址

```
{} == {} //false
```

对象的增删改查

查 （for in遍历查询）

- 可以通过对象名.属性名来进行访问
- 可以通过对象名[属性名字符串]来进行访问

```
var obj = {
    name:'jack',//key为name的属性他的值为jack
    sayHello:function(){ //sayHello 对应的值为一个函数
        console.log('hello')
    },
    children:{
        username:'rose'
    }
}
//for in来遍历对象的中属性
for(var key in obj){
    console.log(key)
    console.log(obj[key])
}
//通过key来访问对应的值
//对象名.属性名
console.log(obj.name)
//通过[]来访问 对象名[属性名字符串]
console.log(obj['name'])
obj.sayHello()
//通过父对象访问里面的属性对象
console.log(obj.children.username)
console.log(obj['children']['username'])
```

增 （对象属性赋值）

```
//增加 对于没有的属性进行赋值就是添加
obj.age = 18
console.log(obj)
```

修改 (重新赋值)

```
//修改 对于有的属性重新赋值就是修改
obj.name = 'tom'
console.log(obj)
```

删除 delete关键词

```
//对于属性进行删除
// delete obj.age
delete obj['age']
console.log(obj)
```

this关键词

this表示这个，this是一个指针引用他没有特定的值，他是根据对应的调用着来执行其引用地址。

对象中的函数里的this指向当前对象

```
//对象中函数里的this 指向对应的对象
obj.say = function(){
    console.log(this == obj)//true
}
obj.say()
```

函数中的this 指向调用函数的对象

```
//对象中函数里的this 指向对应的对象
obj.say = function(){
    console.log(this == obj)//true
}
obj.say()
function test(){
    console.log(this)//指向window
}
//所有在全局声明的变量及函数都是全局的变量 全局变量指向global对象 在浏览器中的global对象是window
//所以对应的全局声明的函数是属于window 也就是说这个函数是由window来调用的 所以里面的this就指向window
window.test() // test() 省略了window window可以被省略的
var a = 'hello'
console.log(window.a)
console.log(window['a'])
```

- this在普通函数中指向其调用者，在全局范围内调用者为window 在对象中调用者为当前对象
- 在事件处理中 this指向当前事件的派发者

```
<button id="btn">点我</button>
<script>
    var btn = document.getElementById('btn')
    btn.onclick = function(){
        console.log(this) //按钮
    }
</script>
```

window对象的俩个函数

setInterval 定时器 （在一定事件内循环执行某个操作 执行多次）

写法

以匿名函数的形式

```
//参数1对应执行的代码  参数2为间隔的时间 参数3为传递给参数1的函数的参数（省略）
setInterval(function(arg){},2000,1)
```

以具名函数的形式

```
function fn(arg){
    console.log(arg)
}
setInterval(fn,2000,2)
```

以字符串的形式（不建议）

```
setInterval('console.log("hello")',1000)
```

clearInterval 清除定时器（必须要做 如果不做那么定时器会无限执行）

```
var i = 0
var timer = setInterval(function (arg) {//setInterval会返回一个标识这个标识是number
类型
    i++
    console.log(arg)
    console.log(i,timer)
    //当i到达100的时候停止
    if(i==100){
        clearInterval(timer)//清除定时器 一定要做的
    }
}, 100, 1)
```

setTimeout 延时器 (延迟执行某个操作 执行一次)

clearTimeout清除延时器

```
//和setInterval用法一致 只不过他只执行一次
var timer = setTimeout(function(){
    console.log(2)
},3000)
//清除延时器
console.log(timer)
clearTimeout(timer)
```

总结

- **setInterval及setTimeout是异步的 也就是说他不会阻塞正常代码的执行**
- **setInterval和setTimeout是通过定时触发器线程计时完放入任务对象等待js引擎空闲再执行 他一定慢于js引擎直接执行的代码**
- **同步比异步快**
- **定时器或延时器开启后必须关闭否则会占用内容**
- **不要在定时器中嵌套定时器（如果需要的话那么每一次都需要在外层的定时器都需要关闭内层定时器）**
- **外层定时器的间隔时间一定要打印里层的间隔时间**

计时器案例

```
<div id="showText">00:00:00</div> <button id="start">开始计时</button> <button
id="stop">停止计时</button>
<script>
    //获取所有的元素
```

```
    var show = document.getElementById('showText')
    var start = document.getElementById('start')
    var stopBtn = document.getElementById('stop')
    //定时器
    var timer
    //点击开始按钮开始计时
    start.onclick = function () {
        //清除上一次的定时器
        clearInterval(timer)
        var h = 0 //时
        var m = 0 //分
        var s = 0 //秒
        //时间发生变化
        timer = setInterval(function () {
            //在每次变化的时候改变对应的showText里面的显示内容
            s++
            if(s==60){
                m++
                s=0
            }
            if(m==60){
                h++
                m=0
            }
            //将对应的内容进行拼接显示到对应的showText中
            show.innerHTML = `${fn(h)}:${fn(m)}:${fn(s)}`
        }, 1000)
    }
    //补零的方法
    function fn(number){
        if(number<10){
            return '0'+number
        }
        return number
    }
    //点击停止按钮停止计时
    stopBtn.onclick = function () {
        //清除定时器
        clearInterval(timer)
    }
</script>
```