



第15课：日期对象详解

主讲老师：万章



日期

创建日期对象

日期对象的运算

日期对象的方法

字符串方法



创建日期对象

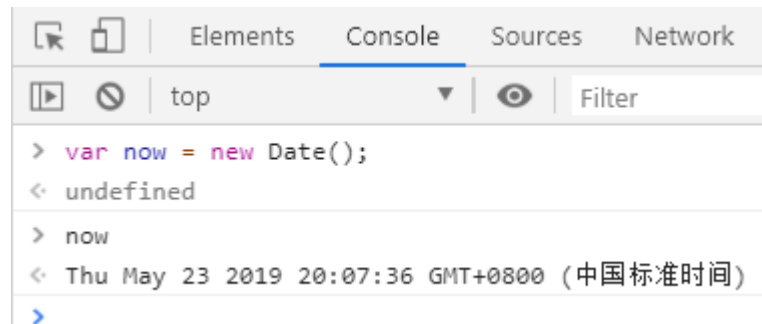
创建当前时间的日期对象

ECMAScript 中的Date 类型是在早期Java 中的java.util.Date 类基础上构建的。为此，Date类型使用自UTC（Coordinated Universal Time，国际协调时间）1970 年1 月1 日午夜（零时）开始经过的毫秒数来保存日期。在使用这种数据存储格式的条件下

Date 类型保存的日期能够精确到1970 年1月1 日之前或之后的285 616 年。

要创建一个日期对象，使用new 操作符和Date 构造函数即可，如下所示。

```
var now = new Date();
```



在调用Date 构造函数而不传递参数的情况下，新创建的对象自动获得当前日期和时间

创建特定时间的日期对象

`let nowT1 = new Date(123456789)` //这个参数是一个毫秒值 从1970年1月1日00:00:00开始加上这个一个毫秒值

`let nowT2 = new Date("January 6,2014")` //参数为日期字符串

`let nowT3 = new Date(2019, 5, 1, 19, 30, 50, 20)` //参数为多个整数包括:年 月 日 时 分 秒 毫秒
注意:这里的月份是从0开始的

`let nowT4 = new Date("2019-5-1")`

`let nowT5 = new Date("2019/5/1")`

//注意: 字符串参数是时间节点 数字参数会默认为毫秒值



日期对象的计算

计算代码运行时间

ECMAScript 5 添加了 `Date.now()` 方法，返回表示调用这个方法时的日期和时间的毫秒数。这个方法简化了使用 `Data` 对象分析代码的工作。

```
//取得开始时间
var start = Date.now();
//调用函数
doSomething();
//取得停止时间
var stop = Date.now();
result = stop - start; //计算doSomething函数的运行时间
```

```
> var t1=Date.now();
< undefined
> var t2=Date.now();
< undefined
> t1
< 1558625846222
> t2
< 1558625847706
> t2-t1
< 1484
>
```

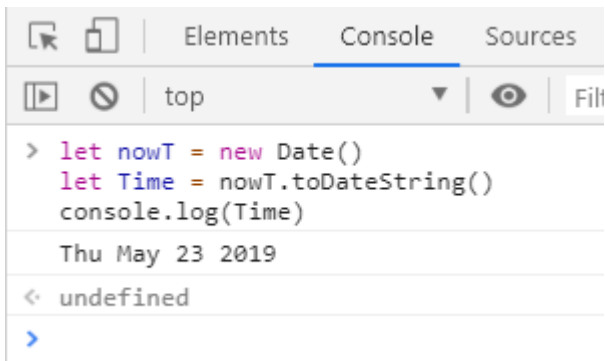


日期对象的方法

日期对象的格式化方法

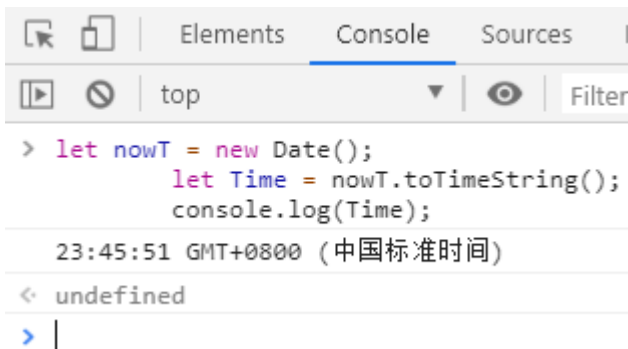
```
let nowT1 = Date.now(); // 返回当前事件距离1970年1月1日00:00:00 之间的时间戳距离
let nowT2 = Date.parse(2019, 5, 1); // 接收一个日期字符串 返回从1970 - 1 - 1 00:00:00 到该日期的毫秒数
let nowT3 = Date.UTC(2019, 5, 1); // 接收以逗号隔开的日期参数 返回从1970 - 1 - 1 00:00:00 到该日期的毫秒数 接收的月份是0 - 11
```

```
let nowT = new Date();
let Time = nowT.toString();
console.log(Time);
```



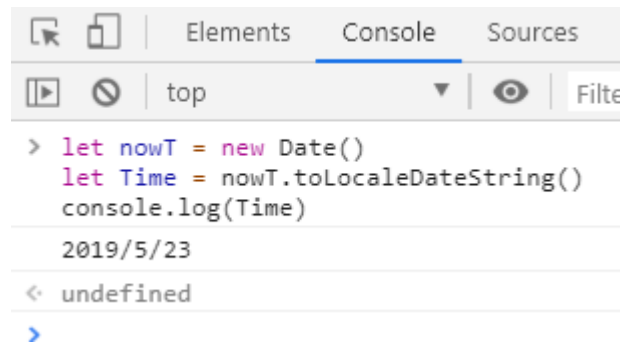
```
> let nowT = new Date()
let Time = nowT.toString()
console.log(Time)
Thu May 23 2019
< undefined
>
```

```
let nowT = new Date();
let Time = nowT.toTimeString();
console.log(Time);
```



```
> let nowT = new Date();
let Time = nowT.toTimeString();
console.log(Time);
23:45:51 GMT+0800 (中国标准时间)
< undefined
>
```

```
let nowT = new Date();
let Time = nowT.toLocaleDateString();
console.log(Time);
```



```
> let nowT = new Date()
let Time = nowT.toLocaleDateString()
console.log(Time)
2019/5/23
< undefined
>
```

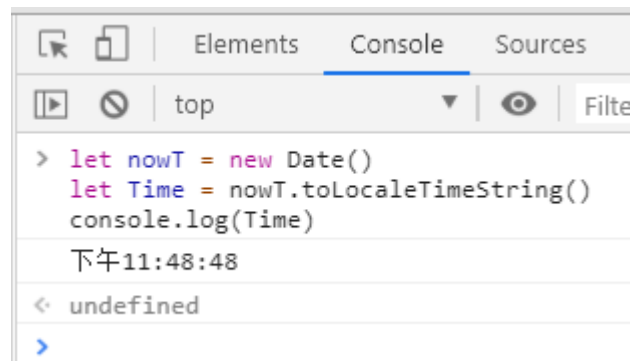
toString() 返回的是星期 月 日 年

toTimeString() 返回的是时 分 秒 时区

toLocaleDateString() 返回的是年/月/日

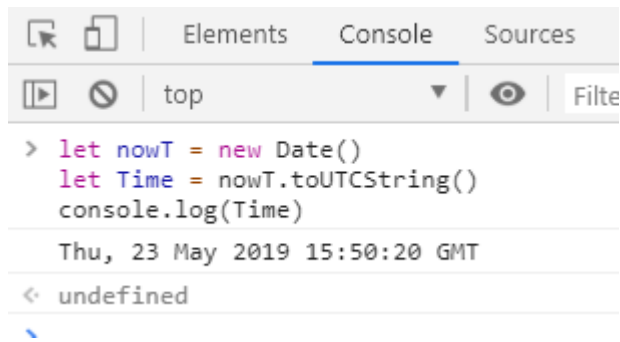
日期对象的格式化方法

```
let nowT = new Date()
let Time = nowT.toLocaleTimeString()
console.log(Time)
```



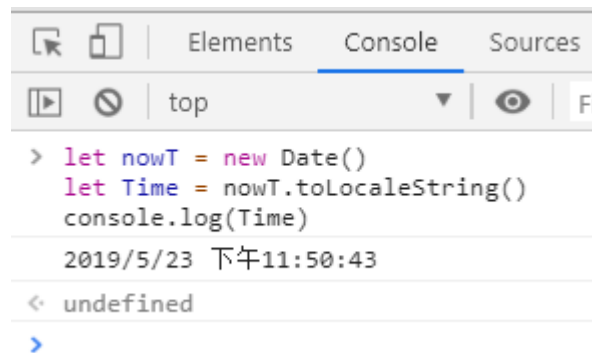
`toLocaleTimeString()` 返回本地时 分 秒

```
let nowT = new Date()
let Time = nowT.toUTCString()
console.log(Time)
```



`toUTCString()` 返回对应的UTC时间
也就是国际标准时间 比北京晚8个
小时

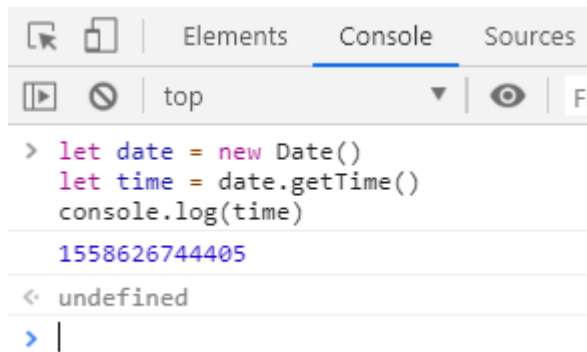
```
let nowT = new Date()
let Time = nowT.toLocaleString()
console.log(Time)
```



`toLocaleString()` 返回本地时间

日期对象的日期方法

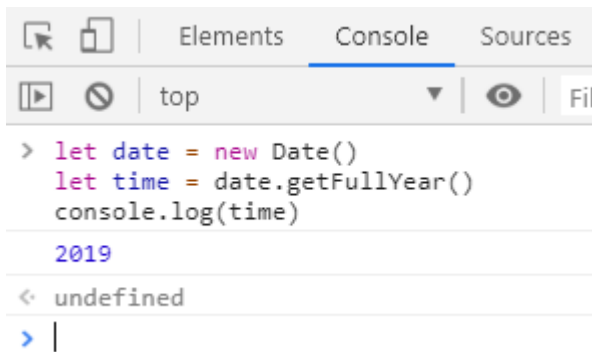
```
let date = new Date()
let time = date.getTime()
console.log(time)
```



```
> let date = new Date()
let time = date.getTime()
console.log(time)
1558626744405
< undefined
> |
```

getTime() 返回一个毫秒值
时间为此刻到时间零点的时间

```
let date = new Date()
let time = date.getFullYear()
console.log(time)
```



```
> let date = new Date()
let time = date.getFullYear()
console.log(time)
2019
< undefined
> |
```

getFullYear() 返回年

1. getMonth() 返回月 注意:得到的月份是从0开始 要返回当前月需要加1
2. getDate() 返回日期
3. getHours() 返回小时
4. getMinutes() 返回分钟
5. getSeconds() 返回秒
6. getDay() 返回星期
7. getTimezoneOffset() 返回是当前事件与UTC的时区差异 以分钟数表示(考虑夏令营时)



字符串方法

字符串和字符串的API

```
let str = 'asd ';  
str.length = 1; //无法手动修改, 只读  
console.log(str.length); //4
```

length属性

返回字符串中字符得长度
只能读不能改,

```
let str = "asdfgh";  
str.indexOf("d"); //2  
str.indexOf("A"); //-1
```

在字符串中查询某字符是否存在, 存在返回下标, 不存在返回-1; 返回第一次匹配的字符的下标

```
let str = "asdf";  
str[0]; // "a" 低版本ie不兼容  
str.charAt(0); // "a"
```

charAt(下标值)

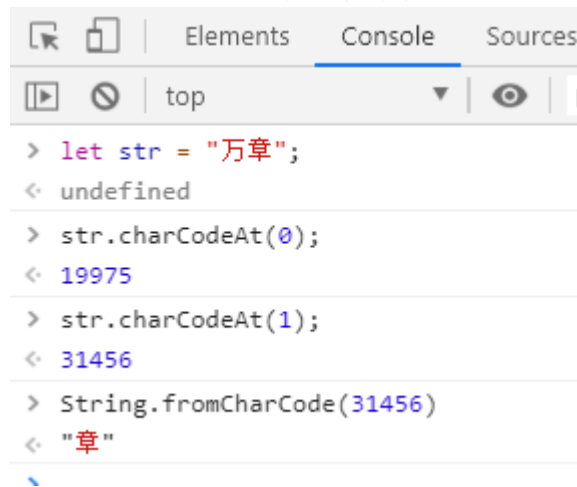
获取字符串中得下标得值, 只可
获取不可修改

```
let str = "大家好, 我是渣渣辉";  
str.lastIndexOf("渣"); //7  
str.lastIndexOf("万"); //-1
```

返回最后一个匹配的字符的下标

```
let str1 = "asd",  
    str2 = "fgh";  
let str3 = str1.concat(str2, "j"); // "asdfghj"  
let str4 = str1+str2+"j"; // "asdfghj"
```

字符串拼接 concat或
者+返回新得字符串



- 返回单个字符的unicode编码charCodeAt
- 通过编码返回单个字符String.fromCharCode

字符串和字符串的API

```
let str = "box-1";  
let str2 = str.slice(0,3);  
// "box" 从下标0开始到下标3结束, 一共3个字符, 区间[0,3)  
str2 = str.slice(2); // "x-1" 从第2位开始, 到结束  
str3 = str.slice(-1); // "1" 从最后一位开始到结束, 可以是负数
```

字符串裁切slice,
slice(开始裁切的位置, 结束裁切的位置)

如果只有一个参数, 那么默认把该参数设置为开始位置, 一直裁切到字符串末尾

如果参数是负数, 那么就是倒过来数, 从数组的结尾开始数数.

例: 设参数为-a, 那么开始位置的索引就是string.length-a

字符串和字符串的API

```
Elements Console Sources Network
top
> let str = "1,2,3";
  let str1="万章1万章2万章3万章4";
< undefined
> var a=str1.split("章");
< undefined
> a
< ▶ (5) ["万", "1万", "2万", "3万", "4"]
> var b=str.split("");
< undefined
> b
< ▶ ["1", "2", "3"]
> var b=str.split(",");
< undefined
> b
< ▶ (5) ["1", ",", "2", ",", "3"]
> |
```

字符串切割split

该方法可以将一个字符串变为一个数组

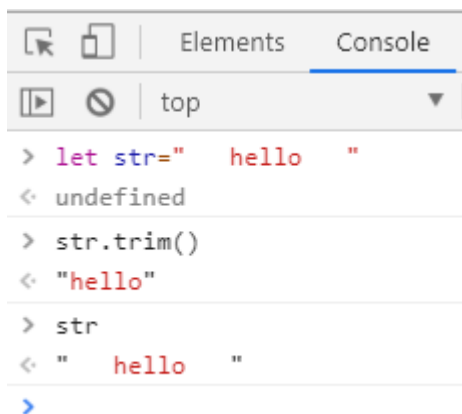
split(切割方法)

如果传入空值, 那么就是把整个字符串都变成数组的一个数组项目

如果传入空字符串,那么每一个单个字符都变成数组的一个数组项目

如果传入其他的字符,那么以这个字符为分割线,分隔字符串, 并把分隔后的每一小块的字符变成数组的一个数组项目

字符串和字符串的API



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following sequence of commands and results:

```
> let str="  hello  "
< undefined

> str.trim()
< "hello"

> str
< "  hello  "
```

The first command initializes a variable `str` with the value `" hello "` (including leading and trailing spaces). The second command calls `str.trim()`, which returns the string `"hello"` with the spaces removed. The third command displays the original `str` variable, which remains `" hello "`.

`trim()` 删除字符串前面和后面得空格