Date 对象

`Date`对象是 JavaScript 原生的时间库。它以国际标准时间（UTC）1970年1月1日00:00:00作为时间的零点，可以表示的时间范围是前后各1亿天（单位为毫秒）。

普通函数的用法

`Date`对象可以作为普通函数直接调用，返回一个代表当前时间的字符串。

```javascript
Date()
// "Tue Dec 01 2015 09:34:43 GMT+0800 (CST)"
```

注意，即使带有参数，`Date`作为普通函数使用时，返回的还是当前时间。

```javascript
Date(2000, 1, 1)
// "Tue Dec 01 2015 09:34:43 GMT+0800 (CST)"
```

上面代码说明，无论有没有参数，直接调用`Date`总是返回当前时间。

构造函数的用法

`Date`还可以当作构造函数使用。对它使用`new`命令，会返回一个`Date`对象的实例。如果不加参数，实例代表的就是当前时间。

```javascript
var today = new Date();
```

`Date`实例有一个独特的地方。其他对象求值的时候，都是默认调用`.valueOf()`方法，但是`Date`实例求值的时候，默认调用的是`toString()`方法。这导致对`Date`实例求值，返回的是一个字符串，代表该实例对应的时间。

```javascript
var today = new Date();

today
// "Tue Dec 01 2015 09:34:43 GMT+0800 (CST)"

// 等同于
today.toString()
// "Tue Dec 01 2015 09:34:43 GMT+0800 (CST)"
```

上面代码中，`today`是`Date`的实例，直接求值等同于调用`toString`方法。

作为构造函数时，`Date`对象可以接受多种格式的参数，返回一个该参数对应的时间实例。

```javascript
// 参数为时间零点开始计算的毫秒数
new Date(1378218728000)
// Tue Sep 03 2013 22:32:08 GMT+0800 (CST)

// 参数为日期字符串
new Date('January 6, 2013');
// Sun Jan 06 2013 00:00:00 GMT+0800 (CST)

// 参数为多个整数，
// 代表年、月、日、小时、分钟、秒、毫秒
new Date(2013, 0, 1, 0, 0, 0, 0)
// Tue Jan 01 2013 00:00:00 GMT+0800 (CST)
```

关于`Date`构造函数的参数，有几点说明。

第一点，参数可以是负整数，代表1970年元旦之前的时间。

```javascript
new Date(-1378218728000)
// Fri Apr 30 1926 17:27:52 GMT+0800 (CST)
```

第二点，只要是能被`Date.parse()`方法解析的字符串，都可以当作参数。

```javascript
new Date('2013-2-15')
new Date('2013/2/15')
new Date('02/15/2013')
new Date('2013-FEB-15')
new Date('FEB, 15, 2013')
new Date('FEB 15, 2013')
new Date('February, 15, 2013')
new Date('February 15, 2013')
new Date('15 Feb 2013')
new Date('15, February, 2013')
// Fri Feb 15 2013 00:00:00 GMT+0800 (CST)
```

上面多种日期字符串的写法，返回的都是同一个时间。

第三，参数为年、月、日等多个整数时，年和月是不能省略的，其他参数都可以省略的。也就是说，这时至少需要两个参数，因为如果只使用“年”这一个参数，`Date`会将其解释为毫秒数。

```javascript
new Date(2013)
```

```
// Thu Jan 01 1970 08:00:02 GMT+0800 (CST)
```

上面代码中，2013被解释为毫秒数，而不是年份。

```javascript
new Date(2013, 0)
// Tue Jan 01 2013 00:00:00 GMT+0800 (CST)
new Date(2013, 0, 1)
// Tue Jan 01 2013 00:00:00 GMT+0800 (CST)
new Date(2013, 0, 1, 0)
// Tue Jan 01 2013 00:00:00 GMT+0800 (CST)
new Date(2013, 0, 1, 0, 0, 0, 0)
// Tue Jan 01 2013 00:00:00 GMT+0800 (CST)
```

上面代码中，不管有几个参数，返回的都是2013年1月1日零点。

最后，各个参数的取值范围如下。

- 年：使用四位数年份，比如`2000`。如果写成两位数或个位数，则加上`1900`，即`10`代表1910年。如果是负数，表示公元前。
- 月：`0`表示一月，依次类推，`11`表示12月。
- 日：`1`到`31`。
- 小时：`0`到`23`。
- 分钟：`0`到`59`。
- 秒：`0`到`59`
- 毫秒：`0`到`999`。

注意，月份从`0`开始计算，但是，天数从`1`开始计算。另外，除了日期的默认值为`1`，小时、分钟、秒钟和毫秒的默认值都是`0`。

这些参数如果超出了正常范围，会被自动折算。比如，如果月设为`15`，就折算为下一年的4月。

```javascript
new Date(2013, 15)
// Tue Apr 01 2014 00:00:00 GMT+0800 (CST)
new Date(2013, 0, 0)
// Mon Dec 31 2012 00:00:00 GMT+0800 (CST)
```

上面代码的第二个例子，日期设为`0`，就代表上个月的最后一天。

参数还可以使用负数，表示扣去的时间。

```javascript
new Date(2013, -1)
// Sat Dec 01 2012 00:00:00 GMT+0800 (CST)
```

```
new Date(2013, 0, -1)
// Sun Dec 30 2012 00:00:00 GMT+0800 (CST)
```

上面代码中，分别对月和日使用了负数，表示从基准日扣去相应的时间。

日期的运算

类型自动转换时，`Date`实例如果转为数值，则等于对应的毫秒数；如果转为字符串，则等于对应的日期字符串。所以，两个日期实例对象进行减法运算时，返回的是它们间隔的毫秒数；进行加法运算时，返回的是两个字符串连接而成的新字符串。

```javascript
var d1 = new Date(2000, 2, 1);
var d2 = new Date(2000, 3, 1);

d2 - d1
// 2678400000
d2 + d1
// "Sat Apr 01 2000 00:00:00 GMT+0800 (CST)Wed Mar 01 2000 00:00:00 GMT+0800 (CST)"
```

静态方法

Date.now()

`Date.now`方法返回当前时间距离时间零点（1970年1月1日 00:00:00 UTC）的毫秒数，相当于 Unix 时间戳乘以1000。

```javascript
Date.now() // 1364026285194
```

Date.parse()

`Date.parse`方法用来解析日期字符串，返回该时间距离时间零点（1970年1月1日 00:00:00）的毫秒数。

日期字符串应该符合 RFC 2822 和 ISO 8061 这两个标准，即`YYYY-MM-DDTHH:mm:ss.sssZ`格式，其中最后的`Z`表示时区。但是，其他格式也可以被解析，请看下面的例子。

```javascript
Date.parse('Aug 9, 1995')
Date.parse('January 26, 2011 13:51:50')
Date.parse('Mon, 25 Dec 1995 13:30:00 GMT')
Date.parse('Mon, 25 Dec 1995 13:30:00 +0430')
Date.parse('2011-10-10')
Date.parse('2011-10-10T14:48:00')
```

上面的日期字符串都可以解析。

如果解析失败，返回`NaN`。

```javascript
Date.parse('xxx') // NaN
```

Date.UTC()

`Date.UTC`方法接受年、月、日等变量作为参数，返回该时间距离时间零点（1970年1月1日00:00:00 UTC）的毫秒数。

```javascript
// 格式
Date.UTC(year, month[, date[, hrs[, min[, sec[, ms]]]]])

// 用法
Date.UTC(2011, 0, 1, 2, 3, 4, 567)
// 1293847384567
```

该方法的参数用法与`Date`构造函数完全一致，比如月从`0`开始计算，日期从`1`开始计算。区别在于`Date.UTC`方法的参数，会被解释为 UTC 时间（世界标准时间），`Date`构造函数的参数会被解释为当前时区的时间。

实例方法

`Date`的实例对象，有几十个自己的方法，除了`valueOf`和`toString`，可以分为以下三类。

- `to`类：从`Date`对象返回一个字符串，表示指定的时间。
- `get`类：获取`Date`对象的日期和时间。
- `set`类：设置`Date`对象的日期和时间。

Date.prototype.valueOf()

`valueOf`方法返回实例对象距离时间零点（1970年1月1日00:00:00 UTC）对应的毫秒数，该方法等同于`getTime`方法。

```javascript
var d = new Date();

d.valueOf() // 1362790014817
d.getTime() // 1362790014817
```

预期为数值的场合，`Date`实例会自动调用该方法，所以可以用下面的方法计算时间的间隔。

```javascript
var start = new Date();
// ...
var end = new Date();
var elapsed = end - start;
```

to 类方法

（1）Date.prototype.toString()

`toString`方法返回一个完整的日期字符串。

```javascript
var d = new Date(2013, 0, 1);

d.toString()
// "Tue Jan 01 2013 00:00:00 GMT+0800 (CST)"
d
// "Tue Jan 01 2013 00:00:00 GMT+0800 (CST)"
```

因为`toString`是默认的调用方法，所以如果直接读取`Date`实例，就相当于调用这个方法。

（2）Date.prototype.toUTCString()

`toUTCString`方法返回对应的 UTC 时间，也就是比北京时间晚8个小时。

```javascript
var d = new Date(2013, 0, 1);

d.toUTCString()
// "Mon, 31 Dec 2012 16:00:00 GMT"
```

（3）Date.prototype.toISOString()

`toISOString`方法返回对应时间的 ISO8601 写法。

```javascript
var d = new Date(2013, 0, 1);

d.toISOString()
// "2012-12-31T16:00:00.000Z"
```

注意，`toISOString`方法返回的总是 UTC 时区的时间。

（4）Date.prototype.toJSON()

`toJSON`方法返回一个符合 JSON 格式的 ISO 日期字符串，与`toISOString`方法的返回结果完全相同。

```javascript
var d = new Date(2013, 0, 1);

d.toJSON()
// "2012-12-31T16:00:00.000Z"
```

（5）Date.prototype.toDateString()

`toDateString`方法返回日期字符串（不含小时、分和秒）。

```javascript
var d = new Date(2013, 0, 1);
d.toDateString() // "Tue Jan 01 2013"
```

（6）Date.prototype.toTimeString()

`toTimeString`方法返回时间字符串（不含年月日）。

```javascript
var d = new Date(2013, 0, 1);
d.toTimeString() // "00:00:00 GMT+0800 (CST)"
```

（7）本地时间

以下三种方法，可以将 Date 实例转为表示本地时间的字符串。

- `Date.prototype.toLocaleString()`：完整的本地时间。
- `Date.prototype.toLocaleDateString()`：本地日期（不含小时、分和秒）。
- `Date.prototype.toLocaleTimeString()`：本地时间（不含年月日）。

下面是用法实例。

```javascript
var d = new Date(2013, 0, 1);

d.toLocaleString()
// 中文版浏览器为"2013年1月1日 上午12:00:00"
// 英文版浏览器为"1/1/2013 12:00:00 AM"

d.toLocaleDateString()
```

```
// 中文版浏览器为"2013年1月1日"
// 英文版浏览器为"1/1/2013"

d.toLocaleTimeString()
// 中文版浏览器为"上午12:00:00"
// 英文版浏览器为"12:00:00 AM"
```

这三个方法都有两个可选的参数。

```javascript
dateObj.toLocaleString([locales[, options]])
dateObj.toLocaleDateString([locales[, options]])
dateObj.toLocaleTimeString([locales[, options]])
```

这两个参数中，`locales`是一个指定所用语言的字符串，`options`是一个配置对象。下面是`locales`的例子。

```javascript
var d = new Date(2013, 0, 1);

d.toLocaleString('en-US') // "1/1/2013, 12:00:00 AM"
d.toLocaleString('zh-CN') // "2013/1/1 上午12:00:00"

d.toLocaleDateString('en-US') // "1/1/2013"
d.toLocaleDateString('zh-CN') // "2013/1/1"

d.toLocaleTimeString('en-US') // "12:00:00 AM"
d.toLocaleTimeString('zh-CN') // "上午12:00:00"
```

下面是`options`的例子。

```javascript
var d = new Date(2013, 0, 1);

// 时间格式
// 下面的设置是，星期和月份为完整文字，年份和日期为数字
d.toLocaleDateString('en-US', {
  weekday: 'long',
  year: 'numeric',
  month: 'long',
  day: 'numeric'
})
// "Tuesday, January 1, 2013"

// 指定时区
d.toLocaleTimeString('en-US', {
```

```
  timeZone: 'UTC',
  timeZoneName: 'short'
})
// "4:00:00 PM UTC"

d.toLocaleTimeString('en-US', {
  timeZone: 'Asia/Shanghai',
  timeZoneName: 'long'
})
// "12:00:00 AM China Standard Time"

// 小时周期为12还是24
d.toLocaleTimeString('en-US', {
  hour12: false
})
// "00:00:00"

d.toLocaleTimeString('en-US', {
  hour12: true
})
// "12:00:00 AM"
```

get 类方法

`Date`对象提供了一系列`get*`方法，用来获取实例对象某个方面的值。

- `getTime()`：返回实例距离1970年1月1日00:00:00的毫秒数，等同于`valueOf`方法。
- `getDate()`：返回实例对象对应每个月的几号（从1开始）。
- `getDay()`：返回星期几，星期日为0，星期一为1，以此类推。
- `getFullYear()`：返回四位的年份。
- `getMonth()`：返回月份（0表示1月，11表示12月）。
- `getHours()`：返回小时（0-23）。
- `getMilliseconds()`：返回毫秒（0-999）。
- `getMinutes()`：返回分钟（0-59）。
- `getSeconds()`：返回秒（0-59）。
- `getTimezoneOffset()`：返回当前时间与 UTC 的时区差异，以分钟表示，返回结果考虑到了夏令时因素。

所有这些`get*`方法返回的都是整数，不同方法返回值的范围不一样。

- 分钟和秒：0 到 59
- 小时：0 到 23
- 星期：0（星期天）到 6（星期六）
- 日期：1 到 31
- 月份：0（一月）到 11（十二月）

```javascript
var d = new Date('January 6, 2013');

d.getDate() // 6
d.getMonth() // 0
d.getFullYear() // 2013
d.getTimezoneOffset() // -480
```

上面代码中，最后一行返回`-480`，即 UTC 时间减去当前时间，单位是分钟。`-480`表示 UTC 比当前时间少480分钟，即当前时区比 UTC 早8个小时。

下面是一个例子，计算本年度还剩下多少天。

```javascript
function leftDays() {
  var today = new Date();
  var endYear = new Date(today.getFullYear(), 11, 31, 23, 59, 59, 999);
  var msPerDay = 24 * 60 * 60 * 1000;
  return Math.round((endYear.getTime() - today.getTime()) / msPerDay);
}
```

上面这些`get*`方法返回的都是当前时区的时间，`Date`对象还提供了这些方法对应的 UTC 版本，用来返回 UTC 时间。

- `getUTCDate()`
- `getUTCFullYear()`
- `getUTCMonth()`
- `getUTCDay()`
- `getUTCHours()`
- `getUTCMinutes()`
- `getUTCSeconds()`
- `getUTCMilliseconds()`

```javascript
var d = new Date('January 6, 2013');

d.getDate() // 6
d.getUTCDate() // 5
```

上面代码中，实例对象`d`表示当前时区（东八时区）的1月6日0点0分0秒，这个时间对于当前时区来说是1月6日，所以`getDate`方法返回6，对于 UTC 时区来说是1月5日，所以`getUTCDate`方法返回5。

set 类方法

`Date`对象提供了一系列`set*`方法，用来设置实例对象的各个方面。

- `setDate(date)`：设置实例对象对应的每个月的几号（1-31），返回改变后毫秒时间戳。
- `setFullYear(year [, month, date])`：设置四位年份。
- `setHours(hour [, min, sec, ms])`：设置小时（0-23）。
- `setMilliseconds()`：设置毫秒（0-999）。
- `setMinutes(min [, sec, ms])`：设置分钟（0-59）。
- `setMonth(month [, date])`：设置月份（0-11）。
- `setSeconds(sec [, ms])`：设置秒（0-59）。
- `setTime(milliseconds)`：设置毫秒时间戳。

这些方法基本是跟`get*`方法一一对应的，但是没有`setDay`方法，因为星期几是计算出来的，而不是设置的。另外，需要注意的是，凡是涉及到设置月份，都是从0开始算的，即`0`是1月，`11`是12月。

```javascript
var d = new Date ('January 6, 2013');

d // Sun Jan 06 2013 00:00:00 GMT+0800 (CST)
d.setDate(9) // 1357660800000
d // Wed Jan 09 2013 00:00:00 GMT+0800 (CST)
```

`set*`方法的参数都会自动折算。以`setDate`为例，如果参数超过当月的最大天数，则向下一个月顺延，如果参数是负数，表示从上个月的最后一天开始减去的天数。

```javascript
var d1 = new Date('January 6, 2013');

d1.setDate(32) // 1359648000000
d1 // Fri Feb 01 2013 00:00:00 GMT+0800 (CST)

var d2 = new Date ('January 6, 2013');

d.setDate(-1) // 1356796800000
d // Sun Dec 30 2012 00:00:00 GMT+0800 (CST)
```

`set`类方法和`get`类方法，可以结合使用，得到相对时间。

```javascript
var d = new Date();

// 将日期向后推1000天
d.setDate(d.getDate() + 1000);
// 将时间设为6小时后
d.setHours(d.getHours() + 6);
// 将年份设为去年
```

d.setFullYear(d.getFullYear() - 1);
```

`set*`系列方法除了`setTime()`，都有对应的 UTC 版本，即设置 UTC 时区的时间。

- `setUTCDate()`
- `setUTCFullYear()`
- `setUTCHours()`
- `setUTCMilliseconds()`
- `setUTCMinutes()`
- `setUTCMonth()`
- `setUTCSeconds()`

```javascript
var d = new Date('January 6, 2013');
d.getUTCHours() // 16
d.setUTCHours(22) // 1357423200000
d // Sun Jan 06 2013 06:00:00 GMT+0800 (CST)
```

上面代码中，本地时区（东八时区）的1月6日0点0分，是 UTC 时区的前一天下午16点。设为 UTC 时区的22点以后，就变为本地时区的上午6点。

参考链接

- Rakhitha Nimesh，[Getting Started with the Date Object](http://jspro.com/raw-javascript/beginners-guide-to-javascript-date-and-time/)
- Ilya Kantor, [Date/Time functions](http://javascript.info/tutorial/datetime-functions)
```