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）。

无论有没有参数，直接调用 `Date` 总是返回当前时间。

```
Date();             // 'Wed Sep 06 2023 20:39:21 GMT+0800 (中国标准时间)'
Date(2023, 09, 06); // 'Wed Sep 06 2023 20:39:33 GMT+0800 (中国标准时间)'
```

1. 构造函数

关于 `Date` 构造函数的参数：

参数可以是负整数，代表 1970-01-01 00:00:00 之前的时间。

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

只要是能被 Date.parse() 方法解析的字符串，都可以当作参数。

```
new Date('2023-9-12')
new Date('2023/9/12')
new Date('09/12/2023')
new Date('2023-SEP-12')
new Date('SEP, 12, 2023')
new Date('SEP 12, 2023')
new Date('September, 12, 2023')
new Date('September 12, 2023')
new Date('12 Sep 2023')
new Date('12, September, 2023')
// Tue Sep 12 2023 00:00:00 GMT+0800 (中国标准时间)
```

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。

```
new Date(2023) // 2023 被解释为毫秒数，而不是年份。
// Thu Jan 01 1970 08:00:02 GMT+0800 (中国标准时间)
```

不管有几个参数，下面返回的都是 2023 年 1 月 1 日零点：

```
new Date(2023, 0)
new Date(2023, 0, 1)
new Date(2023, 0, 1, 0)
new Date(2023, 0, 1, 0, 0)
new Date(2023, 0, 1, 0, 0, 0)
new Date(2023, 0, 1, 0, 0, 0, 0)
// 都返回：Sun Jan 01 2023 00:00:00 GMT+0800 (中国标准时间)
```

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。

月份从 0 开始计算，天数从 1 开始计算。除了日期的默认值为 1，时、分、秒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。

```
new Date(2023, 15) // 月份数从 0 开始算起，11 代表 12 月，15 代表下一年 4 月
// Mon Apr 01 2024 00:00:00 GMT+0800 (中国标准时间)
new Date(2023, 0, 0) // 天数从 1 开始算起，1 代表当月第一天，0 代表上月最后一天
// Sat Dec 31 2022 00:00:00 GMT+0800 (中国标准时间)
```

参数还可以使用负数，表示扣去的时间。

```
new Date(2023, -1)  // 月份扣除 1，表示上一年 12 月，其他值使用默认值，天、时、分、秒、毫秒都是 0
// Thu Dec 01 2022 00:00:00 GMT+0800 (中国标准时间)
new Date(2023, 0, -1) // 天数扣除 2，表示上个月倒数第二天
// Fri Dec 30 2022 00:00:00 GMT+0800 (中国标准时间)
```

2. 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。**

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

```
d2 - d1 // 2000 年，3 月 1 日到 4 月 1 日，一共 31 天，换算成毫秒数为 2678400000
// 2678400000
d2 + d1 // 2000 年 3 月 1 日和 2000 年 4 月 1 日，组合成新的字符串
'Sat Apr 01 2000 00:00:00 GMT+0800 (中国标准时间)Wed Mar 01 2000 00:00:00 GMT+0800
(中国标准时间)'
```

3. 静态方法

3.1. 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。

```
Date.now() // 1694500635966
```

3.2. 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：

```
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`：

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

3.3. 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。

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

// 用法
Date.UTC(2023, 8, 12, 14, 39, 40, 567); // 1694529580567
```

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。

```
let d1 = Date.UTC(2023, 8, 12, 14, 39, 40, 567);
let d2 = new Date(2023, 8, 12, 14, 39, 40, 567).getTime();
d2 - d1; // 28800000  折合 8 小时，也就是说中国地区时间比世界标准时间快 8 小时
```

4. 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。

4.1. 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。

```
let d = new Date();

d.valueOf() // 1694437337735
d.getTime() // 1694437337735
```

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。

```
let start = new Date();
let end = new Date();
let elapsed = end - start;
```

4.2. to 类方法

（1）Date.prototype.toString()

`toString` 方法返回一个完整的日期字符串。

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

d.toString(); // 'Sun Jan 01 2023 00:00:00 GMT+0800 (中国标准时间)'
d;            // Sun Jan 01 2023 00:00:00 GMT+0800 (中国标准时间)
```

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。

```
let d = new Date(2023, 0, 1);
d.toUTCString(); // 'Sat, 31 Dec 2022 16:00:00 GMT'
```

（3）Date.prototype.toISOString()

`toISOString` 方法返回对应时间的 ISO8601 写法。

```
let d = new Date(2023, 0, 1);
d.toISOString(); // '2022-12-31T16:00:00.000Z'
```

`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。

```
let d = new Date(2023, 0, 1);
d.toJSON(); // '2022-12-31T16:00:00.000Z'
```

（5）Date.prototype.toDateString()

`toDateString` 方法返回日期字符串（不含小时、分和秒）。

```
let d = new Date(2023, 0, 1);
d.toDateString() // 'Sun Jan 01 2023'
```

（6）Date.prototype.toTimeString()

`toTimeString` 方法返回时间字符串（不含年月日）。

```
let d = new Date(2023, 0, 1);
d.toTimeString() // '00:00:00 GMT+0800 (中国标准时间)'
```

（7）本地时间

> - 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）。

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

d.toLocaleString();     // '2023/1/1 00:00:00'
d.toLocaleDateString(); // '2023/1/1'
d.toLocaleTimeString(); // '00:00:00'
```

4.3. 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。
- `getDay()`：返回星期几，0（星期天）到 6（星期六）。
- `getFullYear()`：返回四位的年份。
- `getMonth()`：返回月份（0表示1月，11表示12月）。
- `getDate()`：返回实例对象对应每个月的几号（从1开始）。
- `getHours()`：返回小时（0-23）。
- `getMinutes()`：返回分钟（0-59）。
- `getSeconds()`：返回秒（0-59）。
- `getMilliseconds()`：返回毫秒（0-999）。
- `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。

```
let d = new Date('September 12, 2023');;

d.getDate();           // 12
d.getMonth();          // 8
d.getFullYear();       // 2023
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。

计算本年度还剩下多少天：

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

4.4. 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。

> - setFullYear(year [, month, date])：设置四位年份。
> - setMonth(month [, date])：设置月份（0-11）。
> - 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。
> - setHours(hour [, min, sec, ms])：设置小时（0-23）。
> - setMinutes(min [, sec, ms])：设置分钟（0-59）。
> - setSeconds(sec [, ms])：设置秒（0-59）。
> - setMilliseconds(ms)：设置毫秒（0-999）。
> - 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。

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。

```
let d = new Date(2023, 8, 12);

d;             // Tue Sep 12 2023 00:00:00 GMT+0800 (中国标准时间)
d.setDate(15); // 1694707200000
d;             // Fri Sep 15 2023 00:00:00 GMT+0800 (中国标准时间)
```

`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。

```
let d6 = new Date(2023, 7, 12);
d6.setDate(32); // 1693497600000
d6;             // Fri Sep 01 2023 00:00:00 GMT+0800 (中国标准时间)

let d2 = new Date (2023, 7, 12);
d2.setDate(-1); // 1690646400000
d2;             // Sun Jul 30 2023 00:00:00 GMT+0800 (中国标准时间)
```

`d1.setDate(32)` 将日期设为 1 月份的 32 号，因为 1 月份只有 31 号，所以自动折算为 2 月 1 日。
`d2.setDate(-1)` 表示设为上个月的倒数第二天，即 6 月 30 日。

`set` 类方法和 `get` 类方法，可以结合使用，得到相对时间。

```
let d = new Date();

d.setDate(d.getDate() + 1000);      // 将日期向后推1000天
d.setHours(d.getHours() + 6);       // 将时间设为6小时后
d.setFullYear(d.getFullYear() - 1); // 将年份设为去年
```