ECMAScript 6 入门

作者：阮一峰

目录

其他

- 源码
- 修订历史
- 反馈意见

运算符的扩展

本章介绍 ES6 后续标准添加的一些运算符。

---

1. 指数运算符

ES2016 新增了一个指数运算符（`**`）。

```
2 ** 2 // 4
2 ** 3 // 8
```

这个运算符的一个特点是右结合，而不是常见的左结合。多个指数运算符连用时，是从最右边开始计算的。

```
// 相当于 2 ** (3 ** 2)
2 ** 3 ** 2
// 512
```

上面代码中，首先计算的是第二个指数运算符，而不是第一个。

指数运算符可以与等号结合，形成一个新的赋值运算符（`**=`）。

```
let a = 1.5;
a **= 2;
// 等同于 a = a * a;

let b = 4;
b **= 3;
// 等同于 b = b * b * b;
```

---

2. 链判断运算符

编程实务中，如果读取对象内部的某个属性，往往需要判断一下，属性的上层对象是否存在。比如，读取 `message.body.user.firstName` 这个属性，安全的写法是写成下面这样。

```
// 错误的写法
const  firstName = message.body.user.firstName || 'default';

// 正确的写法
const firstName = (message
  && message.body
  && message.body.user
  && message.body.user.firstName) || 'default';
```

上面例子中， `firstName` 属性在对象的第四层，所以需要判断四次，每一层是否有值。

三元运算符 `?:` 也常用于判断对象是否存在。

```
const fooInput = myForm.querySelector('input[name=foo]')
const fooValue = fooInput ? fooInput.value : undefined
```

上面例子中，必须先判断 `fooInput` 是否存在，才能读取

这样的层层判断非常麻烦，因此 ES2020 引入了“链判断运算符”（optional chaining operator） `?.`，简化上面的写法。

```
const firstName = message?.body?.user?.firstName || 'default';
const fooValue = myForm.querySelector('input[name=foo]')?.value
```

上面代码使用了 `?.` 运算符，直接在链式调用的时候判断，左侧的对象是否为 `null` 或 `undefined`。如果是的，就不再往下运算，而是返回 `undefined`。

下面是判断对象方法是否存在，如果存在就立即执行的例子。

```
iterator.return?.()
```

上面代码中， `iterator.return` 如果有定义，就会调用该方法，否则 `iterator.return` 直接返回 `undefined`，不再执行 `?.` 后面的部分。

对于那些可能没有实现的方法，这个运算符尤其有用。

```
if (myForm.checkValidity?.() === false) {
  // 表单校验失败
  return;
}
```

上面代码中，老式浏览器的表单对象可能没有 `checkValidity()` 这个方法，这时 `?.` 运算符就会返回 `undefined`，判断语句就变成了 `undefined === false`，所以就会跳过下面的代码。

链判断运算符 `?.` 有三种写法。

- `obj?.prop` // 对象属性是否存在
- `obj?.[expr]` // 同上
- `func?.(...args)` // 函数或对象方法是否存在

下面是 `obj?.[expr]` 用法的一个例子。

```
let hex = "#C0FFEE".match(/#([A-Z]+)/i)?.[1];
```

上面例子中，字符串的 `match()` 方法，如果没有发现匹配会返回 `null`，如果发现匹配会返回一个数组， `?.` 运算符起到了判断作用。

下面是 `?.` 运算符常见形式，以及不使用该运算符时的等价形式。

```
a?.b
// 等同于
a == null ? undefined : a.b

a?.[x]
// 等同于
a == null ? undefined : a[x]

a?.b()
// 等同于
a == null ? undefined : a.b()

a?.()
// 等同于
a == null ? undefined : a()
```

上面代码中，特别注意后两种形式，如果 `a?.b()` 和 `a?.()` 。如果 `a?.b()` 里面的 `a.b` 有值，但不是函数，不可调用，那么 `a?.b()` 是会报错的。 `a?.()` 也是如此，如果 `a` 不是 `null` 或 `undefined` ，但也不是函数，那么 `a?.()` 会报错。

使用这个运算符，有几个注意点。

（1）短路机制

本质上， `?.` 运算符相当于一种短路机制，只要不满足条件，就不再往下执行。

```
a?.[++x]
// 等同于
a == null ? undefined : a[++x]
```

上面代码中，如果 `a` 是 `undefined` 或 `null` ，那么 `x` 不会进行递增运算。也就是说，链判断运算符一旦为真，右侧的表达式就不再求值。

（2）括号的影响

如果属性链有圆括号，链判断运算符对圆括号外部没有影响，只对圆括号内部有影响。

```
(a?.b).c
// 等价于
(a == null ? undefined : a.b).c
```

上面代码中， `?.` 对圆括号外部没有影响，不管 `a` 对象是否存在，圆括号后面的 `.c` 总是会执行。

一般来说，使用 `?.` 运算符的场合，不应该使用圆括号。

（3）报错场合

以下写法是禁止的，会报错。

```
// 构造函数
new a?.()
new a?.b()

// 链判断运算符的右侧有模板字符串
a?.`{b}`
a?.b`{c}`

// 链判断运算符的左侧是 super
super?.()
super?.foo

// 链运算符用于赋值运算符左侧
a?.b = c
```

（4）右侧不得为十进制数值

为了保证兼容以前的代码，允许 `foo?.3:0` 被解析成 `foo ? .3 : 0` ，因此规定如果 `?.` 后面紧跟一个十进制数字，那么 `?.` 不再被看成是一个完整的运算符，而会按照三元运算符进行处理，也就是说，那个小数点会归属于后面的十进制数字，形成一个小数。

---

3. Null 判断运算符

读取对象属性的时候，如果某个属性的值是 `null` 或 `undefined` ，有时候需要为它们指定默认值。常见做法是通过 `||` 运算符指定默认值。

```
const headerText = response.settings.headerText || 'Hello, world!';
const animationDuration = response.settings.animationDuration || 300;
const showSplashScreen = response.settings.showSplashScreen || true;
```

上面的三行代码都通过 `||` 运算符指定默认值，但是这样写是错的。开发者的原意是，只要属性的值为 `null` 或 `undefined`，默认值就会生效，但是属性的值如果为空字符串或 `false` 或 `0`，默认值也会生效。

为了避免这种情况，ES2020 引入了一个新的 Null 判断运算符 `??`。它的行为类似 `||`，但是只有运算符左侧的值为 `null` 或 `undefined` 时，才会返回右侧的值。

```
const headerText = response.settings.headerText ?? 'Hello, world!';
const animationDuration = response.settings.animationDuration ?? 300;
const showSplashScreen = response.settings.showSplashScreen ?? true;
```

上面代码中，默认值只有在左侧属性值为 `null` 或 `undefined` 时，才会生效。

这个运算符的一个目的，就是跟链判断运算符 `?.` 配合使用，为 `null` 或 `undefined` 的值设置默认值。

```
const animationDuration = response.settings?.animationDuration ?? 300;
```

上面代码中，如果 `response.settings` 是 `null` 或 `undefined`，或者 `response.settings.animationDuration` 是 `null` 或 `undefined`，就会返回默认值300。也就是说，这一行代码包括了两级属性的判断。

这个运算符很适合判断函数参数是否赋值。

```
function Component(props) {
  const enable = props.enabled ?? true;
  // …
}
```

上面代码判断 `props` 参数的 `enabled` 属性是否赋值，基本等同于下面的写法。

```
function Component(props) {
  const {
    enabled: enable = true,
  } = props;
  // …
}
```

`??` 本质上是逻辑运算，它与其他两个逻辑运算符 `&&` 和 `||` 有一个优先级问题，它们之间的优先级到底孰高孰低。优先级的不同，往往会导致逻辑运算的结果不同。

现在的规则是，如果多个逻辑运算符一起使用，必须用括号表明优先级，否则会报错。

```
// 报错
lhs && middle ?? rhs
lhs ?? middle && rhs
lhs || middle ?? rhs
lhs ?? middle || rhs
```

上面四个表达式都会报错，必须加入表明优先级的括号。

```
(lhs && middle) ?? rhs;
lhs && (middle ?? rhs);

(lhs ?? middle) && rhs;
```

```
lhs ?? (middle && rhs);

(lhs || middle) ?? rhs;
lhs || (middle ?? rhs);

(lhs ?? middle) || rhs;
lhs ?? (middle || rhs);
```

---

4. 逻辑赋值运算符

ES2021 引入了三个新的逻辑赋值运算符（logical assignment operators），将逻辑运算符与赋值运算符进行结合。

```
// 或赋值运算符
x ||= y
// 等同于
x || (x = y)

// 与赋值运算符
x &&= y
// 等同于
x && (x = y)

// Null 赋值运算符
x ??= y
// 等同于
x ?? (x = y)
```

这三个运算符 `||=`、`&&=`、`??=` 相当于先进行逻辑运算，然后根据运算结果，再视情况进行赋值运算。

它们的一个用途是，为变量或属性设置默认值。

```
// 老的写法
user.id = user.id || 1;

// 新的写法
user.id ||= 1;
```

上面示例中，`user.id` 属性如果不存在，则设为 `1`，新的写法比老的写法更紧凑一些。

下面是另一个例子。

```
function example(opts) {
  opts.foo = opts.foo ?? 'bar';
  opts.baz ?? (opts.baz = 'qux');
}
```

上面示例中，参数对象 `opts` 如果不存在属性 `foo` 和属性 `baz`，则为这两个属性设置默认值。有了"Null 赋值运算符"以后，就可以统一写成下面这样。

```
function example(opts) {
  opts.foo ??= 'bar';
  opts.baz ??= 'qux';
}
```

留言

Phil_Libra • a year ago

有点不太理解

```
if (myForm.checkValidity?.() === false) {
// 表单校验失败
return;
}
```

这个例子。如果浏览器支持checkValidity()，那表单通过校验的情况下应该返回true，那就和不支持checkValidity()一样，跳过下面的语句了。

1 | 1 • Reply • Share ›

snow ink • 8 months ago

我尝试 a?.b=123 以做到有这个b就赋值，没有就什么也不做。但是这样似乎不是正确语法，那么正确语法是怎样的

| • Reply • Share ›

风早千雪 snow ink • 6 months ago

a.b &&= 123

| • Reply • Share ›

小树 snow ink • 6 months ago

a?.b && (a.b = 123)

| • Reply • Share ›

凌影 snow ink • 8 months ago

if(a?.b) a.b=123

| • Reply • Share ›