运算符的扩展

1. 指数运算符

指数运算符 **，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。

```
2 ** 3; // 8
2 ** 3 ** 2; // <=> 2 ** (3 ** 2), <=> 2 ** 9 // 512
```

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 ** 3;
```

2. 链判断运算符

以往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';
```

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
```

这样的层层判断非常麻烦，“链判断运算符”（optional chaining operator）?. 可以简化上面的写法：

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

使用链判断运算符 `?.` 运算符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`。

判断对象方法是否存在，如果存在就立即执行：

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

`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;
}
```

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。

3. `Null` 判断运算符

以往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;
```

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。

`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;
```

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;
```

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。

?? 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。

```
// 下面四个表达式都会报错
lhs && middle ?? rhs
lhs ?? middle && rhs
lhs || middle ?? rhs
lhs ?? middle || rhs
```

必须加入表明优先级的括号，才不会报错：

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

(lhs ?? middle) && rhs; // 或 lhs ?? (middle && rhs);

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

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

4. 逻辑赋值运算符

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。

```
user.id = user.id || 1; // 老的写法

user.id ||= 1; // 新的写法
```

`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。

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。

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

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';
}
```

5. #! 命令

JavaScript 脚本引入了 `#!` 命令，写在脚本文件或者模块文件的第一行。对于 JavaScript 引擎来说，会把 `#!` 理解成注释，忽略掉这一行。

```
// index.js
#! /usr/bin/env node // 写在脚本文件第一行
console.log(1);

#!/usr/bin/env node // 写在模块文件第一行
export {};
console.log(1);
```

有了这一行以后，Unix 命令行就可以直接执行脚本。

```
# 以前执行脚本的方式
$ node index.js

# hashbang 的方式
$ ./index.js
```