- 如果一个对象通过 `Object.defineProperty(..)` 重新配置这个对象的属性，比如修改它的 `writable` 属性，就会发出 `"reconfigure"` 改变事件。参见本系列《你不知道的 JavaScript（上卷）》第二部分可以获取更多信息；
- 如果一个对象通过 `Object.preventExtensions(..)` 变为不可扩展，就会发出 `"preventExtensions"` 改变事件。

因为 `Object.seal(..)` 和 `Object.freeze(..)` 也都意味着 `Object.preventExtensions(..)`，所以它们也会发出相应的改变事件。另外，对象的每个属性都会发出 `"reconfigure"` 改变事件。如果一个对象的 `[[Prototype]]` 改变，或者通过 `__proto__` setter 来设置，或者使用 `Object.setPrototy peOf(..)` 来设置，都会发出 `"setPrototype"` 改变事件。

注意，这些改变事件会在改变发生后立即发出。不要把这一点和代理混淆（参见第 7 章），代理是可以在动作发生之前拦截的。对象观察支持在变化（或一组变化）发生后响应。

### 8.2.1 自定义改变事件

除了前面 6 类内置改变事件，你也可以侦听和发出自定义改变事件。

考虑：

```
function observer(changes){
    for (var change of changes) {
        if (change.type == "recalc") {
            change.object.c =
                change.object.oldValue +
                change.object.a +
                change.object.b;
        }
    }
}

function changeObj(a,b) {
    var notifier = Object.getNotifier( obj );
    obj.a = a * 2;
    obj.b = b * 3;

    // 把改变事件排到一个集合中
    notifier.notify( {
        type: "recalc",
        name: "c",
        oldValue: obj.c
    } );
}

var obj = { a: 1, b: 2, c: 3 };

Object.observe(
    obj,
```

```
    observer,
    ["recalc"]
);

changeObj( 3, 11 );

obj.a;        // 12
obj.b;        // 30
obj.c;        // 3
```

改变集合（"recalc" 自定义事件）已经排入队列准备发送给观测者，但是还没有发送，因此 obj.c 的值仍然是 3。

默认情况下，改变会在当前事件循环的最后发送（参见本系列《你不知道的 JavaScript（中卷）》第二部分）。如果你想要立即发送，可以使用 Object.deliverChangeRecords(observer)。一旦改变事件发送后，你就可以看到 obj.c 如预期地更新为：

```
obj.c;        // 42
```

在前面的例子中，我们用完成改变事件记录来调用 notifier.notify(..)。还有一种改变记录入队的方式是使用 performChange(..)，这会把指定事件类型从其余事件记录属性中分离出来（通过函数回调）。考虑：

```
notifier.performChange( "recalc", function(){
    return {
        name: "c",
        // this就是在观察之中的对象
        oldValue: this.c
    };
} );
```

某些情况下，这种关注分离可能会更干净地映射到你的使用模式。

### 8.2.2 结束观测

就像普通的事件侦听器一样，你可能希望停止观测一个对象的改变事件。为此，可以通过 Object.unobserve(..) 来实现。

举例来说：

```
var obj = { a: 1, b: 2 };

Object.observe( obj, function observer(changes) {
    for (var change of changes) {
        if (change.type == "setPrototype") {
            Object.unobserve(
                change.object, observer
            );
```

```
            break;
        }
    }
} );
```

在这个小例子中，我们侦听改变事件，直到看到 "setPrototype" 事件发生，然后就停止观察任何新改变事件。

## 8.3 幂运算符

有提案提出为 JavaScript 新增一个运算符用于执行幂运算，就像 Math.pow(..) 一样。考虑：

```
var a = 2;

a ** 4;       // Math.pow( a, 4 ) == 16

a **= 3;      // a = Math.pow( a, 3 )
a;            // 8
```

** 实际上和 Python、Ruby、Perl 以及其他一些语言中的同名运算符一样。

## 8.4 对象属性与 ...

我们在 2.5 节已经看到，... 运算符展开和收集数组的用法很直观，那么对于对象呢？

本考虑在 ES6 中支持这个功能，但已经被推迟到了 ES6 之后（也就是“ES7”或者“ES2016”或者……）。下面是它在“ES6 之后”的时代中可能的工作方式：

```
var o1 = { a: 1, b: 2 },
    o2 = { c: 3 },
    o3 = { ...o1, ...o2, d: 4 };

console.log( o3.a, o3.b, o3.c, o3.d );
// 1 2 3 4
```

... 运算符可能也会用于把对象的解构属性收集到一个对象：

```
var o1 = { b: 2, c: 3, d: 4 };
var { b, ...o2 } = o1;

console.log( b, o2.c, o2.d );     // 2 3 4
```

在这里，...o2 把解构的 c 和 d 属性重新收集回到 o2 对象（o2 没有 o1 中的 a b 属性）。

再次声明，这只是 ES6 之后的在考虑之中的提案。但是如果实现的话会很酷。

## 8.5 Array#includes(..)

JavaScript 开发者需要执行的一个极其常见的任务就是在值数组中搜索一个值。一直以来实现这个任务的方法是：

```
var vals = [ "foo", "bar", 42, "baz" ];

if (vals.indexOf( 42 ) >= 0) {
    // 找到了!
}
```

使用 `>= 0` 检查的原因是，如果找到的话 `indexOf(..)` 返回一个 `0` 或者更大的数字值，如果没有找到就会返回 `-1`。换句话说，我们是在布尔值上下文中使用返回索引的函数。因为 `-1` 为真而不是假，所以需要更多的手动检查。

在本系列《你不知道的 JavaScript（中卷）》第一部分中，探讨了另外一种我更偏爱的模式：

```
var vals = [ "foo", "bar", 42, "baz" ];

if (~vals.indexOf( 42 )) {
    // 找到了!
}
```

这里的 `~` 运算符把 `indexOf(..)` 返回值规范为更适合强制转换为布尔型的值范围。也就是说，`-1` 产生 `0`( 假 )，所有其他值产生非 0 值 ( 真)，这正是判断是否找到这个值所需的。

我认为这是一个改进，然而其他人强烈反对。但是，没有人认为 `indexOf(..)` 的搜索逻辑是完美的。比如，它无法找到数组中 `NaN` 值。

于是出现了一个获得了大量支持的提案，提出增加一个真正返回布尔值的数组搜索方法，称为 `includes(..)`:

```
var vals = [ "foo", "bar", 42, "baz" ];

if (vals.includes( 42 )) {
    // 找到了!
}
```

`Array#includes(..)` 使用的匹配逻辑能够找到 `NaN` 值，但是无法区分 `-0` 和 `0`（参见本系列《你不知道的 JavaScript（中卷）》第一部分）。如果你不关心程序中的 `-0` 值，那么这可能就是你所需要的。如果你确实在意这个 `-0` 值的话，那么你就需要实现自己的搜索逻辑，很可能是使用 `Object.is(..)` 工具（参见第 6 章）。

## 8.6 SIMD

我们在本系列《你不知道的 JavaScript（中卷）》第二部分中详细介绍了单指令多数据（Single Instruction, Multiple Data，SIMD），但是值得在这里简单说明一下，因为这很可能会是接下来出现在未来 JavaScript 中的特性之一。

SIMD API 暴露了可以同时对多个数字值运算的各种底层（CPU）指令。比如，你可以指定两个**向量**，其中分别有 4 个或 8 个数字，把它们的对应元素一次全部相乘（数据并行！）。

考虑：

```
var v1 = SIMD.float32x4( 3.14159, 21.0, 32.3, 55.55 );
var v2 = SIMD.float32x4( 2.1, 3.2, 4.3, 5.4 );

SIMD.float32x4.mul( v1, v2 );
// [ 6.597339, 67.2, 138.89, 299.97 ]
```

除了 `mul(..)`（相乘）之外，SIMD 还会包含其他几个运算，比如 `sub()`、`div()`、`abs()`、`neg()`、`sqrt()` 以及很多其他运算。

对于下一代高性能 JavaScript 应用来说，并行数学运算是很关键的。

## 8.7 WebAssembly (WASM)

在本部分即将完成的时候，Brendan Eich 对 WebAssembly（WASM）的最新声明，可能会对 JavaScript 的未来发展路线产生重大影响。这里我们无法详细介绍 WASM，因为在编写本部分的时候它还处于刚刚开始的阶段。但如果不简要介绍一下这一主题，本部分就是不完整的。

最近（以及不久的将来）JavaScript 语言设计修改上的最大压力之一就是需要成为更适合从其他语言（比如 C/C++、ClojureScript 等）变换 / 交叉编译的目标语言。显然，代码作为 JavaScript 运行的性能问题一直是一个主要关注点。

本系列《你不知道的 JavaScript（中卷）》第二部分中介绍过，几年前一组 Mozilla 开发者为 JavaScript 引入了一个新思路，称为 ASM.js。ASM.js 是合法 JavaScript 的一个子集，这个子集最严格地限制了那些使得 JavaScript 引擎难于优化的行为。结果就是兼容 ASM.js 的代码运行在支持 ASM 的引擎上时效率有巨大的提升，几乎与原生优化的等价 C 程序相当。很多人把 ASM.js 看作是高性能要求 JavaScript 应用使用 JavaScript 的最可能支柱。

换句话说，浏览器中运行代码的所有路径都通过 JavaScript。

直到 WASM 发布之前，是这样的。WASM 为其他语言在浏览器运行时环境中运行提供了一条新路径，不需要先通过 JavaScript。本质上说，如果 WASM 发布，JavaScript 引擎将

会获得执行二进制格式代码的新能力，这种格式某种程度上类似于字节码（bytecode，就像 JVM 上运行的那样）。

WASM 提出了一种代码的高度压缩 AST（语法树）二进制表示格式，然后可以直接向 JavaScript 引擎发出指令，而它的基础结构，不需要通过 JavaScript 解析，甚至不需要符合 JavaScript 的规则。像 C 或 C++ 这样的语言可以被直接编译为 WASM 格式而不是 ASM.js，这样通过跳过 JavaScript 解析会获得额外的速度优势。

WASM 的近期目标是与 ASM.js 和真正 JavaScript 相当。但最终的预期是，WASM 将会增加新功能，而这些新功能是超出 JavaScript 所能做的。比如像线程这样的激进功能给 JavaScript 带来了很大压力——这个改变将会给整个 JavaScript 生态系统带来巨大震撼——将很可能会成为一个 WASM 扩展，缓解 JavaScript 本身的修改压力。

实际上，这个新的发展图景为很多语言打开了新的道路，使其能够进入 Web 运行时。对于 Web 平台来说，这是一个令人激动的新特性。

对于 JavaScript 来说这意味着什么？ JavaScript 将会变得无关紧要或者“死去”吗？绝对不会！看起来在以后的几年里，ASM.js 不会有太大的发展了，但在 Web 平台中 JavaScript 的主体还是非常安全的。

WASM 的支持者认为，WASM 的成功将意味着 JavaScript 的设计可以免于被不现实的需求撕裂的压力。重点是，对于应用中的高性能部分 WASM 是更好的目标，可以用其他多种语言编写。

有趣的是，JavaScript 是未来不太可能转化为 WASM 的语言之一。未来的修改可能会刻划出 JavaScript 的一个适合于转化为 WASM 的子集，但是这条发展路径的优先级似乎并不高。

尽管 JavaScript 很可能不会转化为 WASM，但是 JavaScript 代码和 WASM 代码将能够最大程度地交互，就像现在的模块交互一样自然。你可以设想调用像 `foo()` 这样的 JavaScript 函数，而实际上调用的是一个同名的能够在你的其余 JavaScript 的限制之外良好运行的 WASM 函数。

当下用 JavaScript 编写的代码将可能继续用它编写，至少在可见的未来是这样。transpile 到 JavaScript 的东西将可能最终考虑使用 WASM 替代。对于那些性能要求极高，不能容忍多层抽象的功能，最有可能的选择是寻找合适的非 JavaScript 语言编写，然后以 WASM 为目标。

这个转变可能会比较缓慢，需要几年才能完成。WASM 进入所有主流浏览器平台可能至少也需要数年。同时，WASM 项目（https://github.com/WebAssembly）已经有一个早期的 polyfill 对其基本宗旨提供了概念证明。

但随着时间的发展，也随着 WASM 学到更多非 JavaScript 技巧，很可能当前一些 JavaScript 的东西会被重构为以 WASM 为目标的语言。举例来说，框架、游戏引擎以及其他常用工具中性能敏感的部分都可能从这样的转变中获益。在自己的 Web 应用中使用这些工具的开发者很可能不会注意到使用和集成过程中的差别，只会自动受益于性能和功能的提高。

可以确定的是，随着时间的发展 WASM 会越来越真实，对 Javascript 的发展方向和设计的影响也会越来越大。这可能是“ES6 之后”中最值得开发者关注的重要主题之一。

## 8.8 小结

如果说本质上本系列的其他几本书都是提出这个挑战：“你（可能）不（像你以为的那么）懂 JavaScript”，那么本部分就是在说：“你不再懂 JavaScript 了”。本部分覆盖了这个语言大量的 ES6 新主题，这是这个语言的令人激动的新特性和范式，将会永久地改进 JavaScript 程序。

但 ES6 并不是 JavaScript 的终结。还早得很呢！在“ES6 之后”这段时间已经出现了大量处于各种开发阶段的新特性。在这一章里，我们简单了解了那些在不久的将来很可能进入 JavaScript 的新特性。

`async function` 是建立在生成器 + promise 模式（参见第 4 章）之上的强大的语法糖。`Object.observe(..)` 为观察对象改变事件增加了直接的原生支持，这对于实现数据绑定很重要。幂运算符 `**`、针对对象属性的 `...` 以及 `Array#includes(..)` 都是对现有机制简单但有用的改进。最后，SIMD 把高性能 JavaScript 的革命带入一个新时代。

听起来像陈词滥调，但 JavaScript 的未来是光明的！这个系列，特别是本书的这一部分，已经把挑战放在了读者的面前。你还在等什么？是时候开始学习和探索了！

微信连接

回复“JavaScript”查看相关书单

微博连接

关注@图灵教育 每日分享IT好书

QQ连接

图灵读者官方群I：218139230
图灵读者官方群II：164939616

**图灵社区**
**iTuring.cn**

在线出版，电子书，《码农》杂志，图灵访谈


“当你努力理解自己所写的代码时，你不但创造了更好的工作成果，也提升了自己的能力。这些代码不再只是你的工作，而是成了你的手艺。这就是我喜爱本书的理由。”
——**Jenn Lukas**
前端顾问

“我认为没有比这更全面、更深入的ES6参考书了。”
——**Angus Croll**
Twitter工程师

# 你不知道的JavaScript下卷

JavaScript语言有很多复杂的概念，但却用简单的方式体现出来（比如回调函数），因此，JavaScript开发者无需理解语言内部的原理，就能编写出功能全面的程序。然而，JavaScript的这些复杂精妙的概念才是语言的精髓，即使是经验丰富的JavaScript开发者，如果没有认真学习，也无法真正理解语言本身的特性。正是因为绝大多数人不求甚解，一遇到出乎意料的行为就认为是语言本身有缺陷，进而把相关的特性加入黑名单，久而久之就排除了这门语言的多样性，人为地使它变得不完整、不安全。

“你不知道的JavaScript”系列就是要让不求甚解的JavaScript开发者迎难而上，深入语言内部，弄清楚JavaScript每一个零部件的用途，轻松理解前端圈里出现的各种技术、框架和流行术语。本书介绍了该系统的两个主题：“起步上路”以及“ES6及更新版本”。

## 作者介绍

**Kyle Simpson**，推崇开放的互联网，对JavaScript、HTML5、实时/端对端通信和Web性能有深入研究。他是技术书作家、技术培训师、讲师和开源社区的活跃成员。

封面设计：Karen Montgomery Randy Comer 张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机／程序设计／JavaScript

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行

This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

O'REILLY®
oreilly.com
YouDontKnowJS.com

ISBN 978-7-115-47165-9

ISBN 978-7-115-47165-9
定价：79.00元

# 看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱:
ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks