掌握ECMAScript规范，成为Web前端开发高手

- 剖析ECMAScript 2018规范，启发读者领悟ES编程思想
- 揭示ES各个知识点之间的联系，透彻理解ECMAScript语言
- 众多小示例代码，快速掌握ECMAScript编程方法

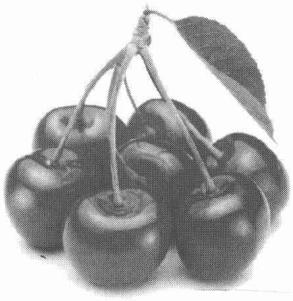
Web
前端技术

ECMAScript 2018 快速入门

黄灯桥 编著

示例源代码

清華大學出版社



ECMAScript 2018 快速入门

黄灯桥 编著

清华大学出版社
北京

内 容 简 介

ECMAScript 是一种由 ECMA 国际通过 ECMA-262 标准化的脚本程序设计语言，目前最新版本为 ECMAScript 2018。JavaScript 是这个标准的一个实现和扩展。这种语言广泛用于 Web 前端开发，可以说想成为一名前端开发高手，就必须掌握 ECMAScript。

本书分为 13 章，较为系统地介绍 ECMAScript 语言，内容包括变量与常量、表达式和运算符、字符串、数字和符号、数组和类型化数组、对象、函数、集合和映射、迭代器和生成器、Promise 对象与异步函数、代理、类和模块，最后引导读者自己动手写一个 JS 框架。

本书适合 Web 前端初学者、不了解 ECMAScript 的 Web 前端开发人员，也适合高等院校和培训学校相关专业的师生进行参考。

图书在版编目（CIP）数据

ECMAScript 2018 快速入门 / 黄灯桥编著. — 北京：清华大学出版社，2019
ISBN 978-7-302-51681-1

I. ①E… II. ①黄… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 264262 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市吉祥印务有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张**：9　　**字　　数**：230 千字
版　　次：2019 年 1 月第 1 版　　**印　　次**：2019 年 1 月第 1 次印刷
定　　价：29.00 元

产品编号：069716-01

前 言

1994 年，第一个比较成熟的浏览器 Navigator（0.9 版本）发布的时候，只是一个纯浏览器工具，不具备互动功能。为了解决互动问题，网景公司希望能通过一种脚本语言来实现。至于使用什么语言，网景公司一时也难以决定。这时，Sun 公司推出了 Java，给人的感觉就是拥抱了 Java，就等于拥抱了未来。作为 Java 的信徒，网景公司于是选择与 Sun 合作，推出 JavaScript 语言。

网景的本意是制作一个 Java 的简化版脚本语言，但很不幸，他们请来开发 JavaScript 语言的设计师 Brendan Eich[1]并不是 Java 的信徒，只用 10 天时间就将 JavaScript 写出来了。不过，JavaScript 并没有成为简化版的 Java，而是成了一个大杂烩，使用了 C 语言的语法、Java 语言的数据类型和内存管理，借鉴 Scheme，把函数式开发作为主要开发方式，借鉴 Self 语言，使用基于原型（prototype）的继承机制。这个大杂烩就这样成为我们现在使用起来比较酸爽的 JavaScript。以至于作为 JavaScript 的设计师，他自己一点都不喜欢这个作品：“与其说我爱 JavaScript，不如说我恨它。它是 C 语言和 Self 语言结合的产物。十八世纪英国文学家约翰逊博士说得好：‘它的优秀之处并非原创，它的原创之处并不优秀’”。这都是和公司决策层妥协后的结果。而这造成的最终结果就是不少人误以为 JavaScript 就是 Java，为了修正 JavaScript 的开发问题，不断衍生出不同版本的衍生语言，如 CoffeeScript、TypeScript 等。这件事给编者的第一观感就是情怀这东西，太有毒了，不得不时常用来提醒自己。想当初，编者也是有情怀的，也曾自学过 Java，但在使用过后，加上 Sun 和微软的 Java 之争，就再也不去考虑了。

网景推出了这样一种语言，微软等其他公司也不甘落后，都各自推出了自己的脚本语言。如果各大公司都这样各自独立地发展下去，那么最头疼的就是开发人员了，为了解决各浏览器的兼容性问题，那可是要费九牛二虎之力的。还好，网景做了一个好的表率，在 1996 年 11 月，将 JavaScript 提交给了国家标准化组织 ECMA，使 JavaScript 成了一种国际标准，各大公司虽然有私心，但也不得不遵守标准，不然，最大的后果就是被开发人员甚至用户抛弃。在 1997 年，ECMA 发布 262 号标准文件的第一版，也就是 ECMAScript 1.0。JavaScript 这个名称只是人们习惯的说法，它的正式名称是 ECMAScript，这样做一是因为 JavaScript 是网景公司的商标，不便于使用，二是为了保证这门语言的开放性和中立性。

---

[1] 有关 Brendan Eich 的详细信息请参阅：https://baike.baidu.com/item/Brendan%20Eich。

ECMAScript 在 1998 年发布了 2.0 版本，在 1999 年发布了 3.0 版本，然后就戛然而止。这其中的原因是在 2000 年酝酿 ECMAScript 4.0 的时候，版本太过于激进了，导致标准委员会的一些成员不愿意接受。2007 年 10 月，ECMAScript 4.0 版草案发布，但在发布正式版本前，又发生了严重分歧。其中，雅虎、微软和谷歌等为首的大公司反对大幅升级，主张小幅改动，这毕竟关系到他们的利益，而以 Brendan Eich 为首的 Mozilla 则坚持当前的草案。事情闹得不可开交，到了 2008 年 7 月，ECMA 不得不中止 ECMAScript 4.0 的开发，将其中一部分小的功能改善发布为 3.1 版本，而激进的改动则留待以后解决。在 2009 年 12 月，将 ECMAScript 3.1 改名为 ECMAScript 5.0 发布。2011 年 6 月，ECMAScript 5.1 版发布，并且成为 ISO 国际标准（ISO/IEC 16262:2011）。

2013 年 3 月，ECMAScript 6 草案冻结，不再添加新功能。新功能设想将被放到 ECMAScript 7。2013 年 12 月，ECMAScript 6 草案发布。然后是 12 个月的讨论期，听取各方反馈。2015 年 6 月，ECMAScript 6 正式通过，成为国际标准。从这一版本开始，ECMAScript 6 更名为 ECMAScript 2015，也就是以年份作为版本号，不再使用以往的版本号。之后就每年发布一个版本，直到如今的 ECMAScript 2018。

综观 JavaScript 的发展，与行业的发展是密不可分的。在 JavaScript 诞生后，程序员就开始考虑如何利用 JavaScript 来构建更丰富的客户端。例如，编者 2000 年在一家网络公司工作时，为了实现项目中的日期选择，就与同事合作编写了日期选择器，之后，又完成了一个颜色选择器。不过，这项工作没持续多长时间，编者就离开了，没继续从事这方面的工作。当编者重新走上 Web 开发的路途时，JavaScript 已经从以前的单一组件化（如 HTC、DXHtml 等）走向了框架化。

在这里不得不提一下微软、谷歌和雅虎的贡献。XMLHttpRequest 对象是微软发明的，被整合到了 IE 4 中，不过，在当时并没有引起什么轰动，直到谷歌在谷歌地图中用它实现了令人惊艳的交互效果，才引发了一场我们称之为 Web 2.0 的技术革命。伴随着 Web 2.0 的步伐，各种 JavaScript 框架以井喷的形式爆发。在这波大潮当中，雅虎的 YUI 可以说是第一个相当实用的图形界面（UI）库，可惜的是，最终用的人并不多。不过，雅虎的另一个贡献 YUI Compressor 却非常受欢迎，它本来是为 YUI 服务的，可将零散的 YUI 文件压缩为一个单一的脚本包，而最终发展成为各种框架或脚本应用程序的压缩打包程序，而这无疑是脚本发展的一个跨越。试想一下，对于一个大型项目或者框架来说，总不可能把脚本都写在一个文件里，但文件太多，对服务器负载和浏览器的响应来说都是难以接受的，通过压缩和打包的方式将文件压缩并合并到一个文件，对于服务器的负载和浏览器的响应就好很多了，无疑大大促进了框架的发展。

随着移动互联网的发展，JavaScript 的应用越来越广泛，项目逐渐扩大，为了满足发展的需要，JavaScript 不得不变，于是更新频率越来越快。

JavaScript 版本当然是越新越好，但要使用最新版本的 JavaScript，就必须考虑浏览器的兼容性问题。如果不能兼容大多数浏览器，那么部分工作就要重来，这是任何项目都难以承受的。编者在使用 Ext JS 开发项目的时候，排在第一位的需求是客户对浏览器有什么要求，如果一定要兼容 IE 8 以下的浏览器，编者建议不要使用 Ext JS，换其他对这些浏览

器兼容性更好的版本。当然，这在开发效率上是有所降低的，并且开发成本会上升。好在这方面要求比较高的客户不算太多，毕竟现在还死守 IE 浏览器的用户不是太多。编者秉持的观点是为了这一点点的用户而去牺牲开发效率、开发成本以及维护成本，不值得。而且用户也不是铁板一块，或许他们早厌烦了 IE，只是没有机会，或者没有动力，或者不知道怎么去更换浏览器而已。在使用最新版本的 ECMAScript 方面就比编者使用 Ext JS 弹性大了，通过代码转换器和填充代码等方式，可以将代码转换为兼容老旧浏览器的代码，非常方便，还好，Ext JS 这个框架也在往这个方向迁移。可以预见，未来 JavaScript 的开发模式基本都是使用 NodeJS 以及各种类库，使用最新的 ECMAScript 来开发项目的。能早点熟悉这种开发模式，对于要进入这个行业的开发人员来说是必不可少的。本书最后的例子就是为大家熟悉这种开发模式而专门添加的。

本书的开发环境如下。

- 操作系统：Windows 10。
- 开发工具：Visual Studio Code。
- NodeJS：10.7.0。
- 浏览器：Firefox 61.0.1

本书的源代码放在 GitHub 上，大家可以自行到相应的地址下载。具体的下载地址是：https://github.com/tianxiaode/ECMAScript2018。

希望本书能给每位读者带来帮助，如果对本书有任何意见和建议，或者有任何技术上的问题，请发邮件到 huangdengqiao@outlook.com，或者加入 QQ 群：391747779、193238033 和 131404874。如果想了解最新的 Ext JS 动态或编者的最新博文，可访问编者的博客：http://blog.csdn.net/tianxiaode。

在本书的出版过程中，得到了清华大学出版社图格新知事业部编辑的大力支持，在他们的努力下，促成了本书的出版，在此表示衷心的感谢。此外，还要感谢那些在互联网上默默耕耘的博客作者以及在各大论坛回复问题的大牛们，是他们的努力耕耘，才使编者找到解决问题的办法，是他们让编者有了进一步提高技能的机会。

编 者

2018 年 11 月

目　录

第 1 章　变量与常量 1

1.1　var 的问题 1
1.2　let 语句 2
1.3　使用 let 的好处 2
1.3.1　避免重复声明 2
1.3.2　避免变量未声明就使用 3
1.3.3　避免全局变量成为全局对象的属性 3
1.3.4　简化代码 3
1.3.5　模拟私有成员 4
1.4　const 语句 5
1.5　建议的方式 6
1.6　小　结 6

第 2 章　表达式和运算符 7

2.1　解构赋值 7
2.1.1　自定义变量名 8
2.1.2　剩余项 8
2.1.3　默认值 9
2.1.4　函数参数 9
2.1.5　忽略某些数据 10
2.1.6　克隆数组 11
2.1.7　克隆对象 11
2.1.8　数据交换 11
2.1.9　返回多个值 12
2.2　幂运算符 12
2.2.1　右结合 13
2.2.2　带歧义的幂运算 13
2.3　Math 对象 13

2.4 小 结 14

第 3 章 字符串 15

3.1 四字节字符的定义方式 15
3.2 新增的方法 15
3.2.1 codePointAt 方法 15
3.2.2 fromCodePoint 方法 16
3.2.3 normalize 方法 16
3.2.4 includes 方法 17
3.2.5 startsWith 方法 17
3.2.6 endsWith 方法 18
3.2.7 repeat 方法 18
3.2.8 padStart 和 padEnd 方法 18
3.3 正则表达式 19
3.3.1 u 标志 19
3.3.2 y 标志 19
3.3.3 DOTALL 模式（s 标志） 20
3.3.4 flags 属性 20
3.3.5 命名捕获组 20
3.3.6 在后向引用中使用命名捕获组 21
3.3.7 在替换中使用命名捕获组 21
3.3.8 Unicode 属性转义 21
3.3.9 后行断言 22
3.4 模板字面量 22
3.4.1 语法 23
3.4.2 多行文本 24
3.4.3 嵌入数据 24
3.4.4 嵌套模板 24
3.4.5 带标签的模板字面量 26
3.4.6 原始值 26
3.4.7 转义字符序列的功能修改 27
3.5 小 结 27

第 4 章 数字和符号 28

4.1 数 字 28
4.1.1 Number.isFinite() 28
4.1.2 Number.isNaN() 29
4.1.3 Number.parseInt()和 Number.parseFloat() 29

4.1.4 Number.isInteger()....29
4.1.5 安全整型....30
4.2 符号....30
4.2.1 值的唯一性....31
4.2.2 私有成员....31
4.2.3 获取符号属性....32
4.2.4 全局共享....33
4.3 众所周知的符号....33
4.3.1 Symbol.hasInstance....33
4.3.2 Symbol.isConcatSpreadable....34
4.3.3 Symbol.iterator....35
4.3.4 Symbol.match....35
4.3.5 Symbol.replace....35
4.3.6 Symbol.search....35
4.3.7 Symbol.split....35
4.3.8 Symbol.species....35
4.3.9 Symbol.toPrimitive....36
4.3.10 Symbol.toStringTag....36
4.3.11 Symbol.unscopables....37
4.4 小 结....37

第 5 章 数组和类型化数组....39

5.1 新方法....39
5.1.1 of 方法....39
5.1.2 from 方法....40
5.1.3 find 方法....40
5.1.4 findIndex 方法....41
5.1.5 fill 方法....41
5.1.6 copyWithin 方法....41
5.1.7 includes 方法....41
5.1.8 entries 方法....42
5.1.9 keys 方法....42
5.1.10 values 方法....42
5.2 类型化数组....42
5.2.1 数据类型....42
5.2.2 数组缓冲区....43
5.2.3 数据视图....43
5.2.4 类型化数组即视图....45

5.2.5 类型化数组与数组的主要区别 45
5.3 小 结 46

第 6 章 对象 47

6.1 属性简写 47
6.2 方法简写 47
6.3 未定的属性名和方法名 48
6.4 新方法 49
6.4.1 is 方法 49
6.4.2 assign 方法 49
6.4.3 values 方法 51
6.4.4 entries 方法 52
6.4.5 getOwnPropertyDescriptors 方法 52
6.5 原型 53
6.6 super 关键字 54
6.7 小 结 55

第 7 章 函数 56

7.1 参数的默认值 56
7.1.1 设置默认值 56
7.1.2 参数默认值表达式 57
7.1.3 使用参数值作为默认值 57
7.1.4 使用参数值作为默认值表达式的参数 58
7.2 剩余参数 58
7.3 扩展运算符 59
7.4 name 属性 59
7.4.1 声明函数 59
7.4.2 函数表达式 60
7.4.3 对象的方法 60
7.4.4 bind 方法创建的函数 60
7.4.5 new Function 创建的函数 60
7.4.6 实例 61
7.5 new.target 属性 61
7.6 在参数中使用尾后逗号 62
7.7 箭头函数 62
7.7.1 基本语法 62
7.7.2 箭头必须与参数在同一行 63
7.7.3 不绑定 this 63

7.7.4 没有 arguments 对象.....65
7.7.5 定义方法时使用箭头函数.....65
7.7.6 不能用作构造函数.....65
7.7.7 没有原型.....65
7.7.8 不能作为生成器.....65
7.7.9 返回对象字面量.....65
7.8 小 结.....65

第 8 章 集合和映射.....67

8.1 Set.....67
8.1.1 基本语法.....67
8.1.2 添加和删除元素.....67
8.1.3 清空 Set.....68
8.1.4 判断某个值是否存在.....68
8.1.5 遍历.....68
8.1.6 返回元素总数.....68
8.1.7 转换为数组.....68
8.2 WeakSet.....68
8.2.1 基本语法.....69
8.2.2 添加和删除元素.....69
8.2.3 判断某个值是否存在.....69
8.2.4 不可遍历.....69
8.3 Map.....69
8.3.1 基本语法.....70
8.3.2 添加和删除元素.....70
8.3.3 获取值.....70
8.3.4 清空 Map.....71
8.3.5 判断某个键是否存在.....71
8.3.6 遍历.....71
8.3.7 返回元素总数.....71
8.4 WeakMap.....71
8.4.1 基本语法.....71
8.4.2 添加和删除元素.....72
8.4.3 获取值.....72
8.4.4 判断某个键是否存在.....72
8.4.5 不可遍历.....72
8.5 小 结.....72

第 9 章　迭代器和生成器 73

9.1　迭代器 73
9.1.1　内置迭代器 73
9.1.2　next 方法 73
9.1.3　对象与迭代器 74
9.2　异步迭代器 74
9.3　生成器 74
9.3.1　基本语法 75
9.3.2　返回可迭代对象 75
9.3.3　在类或对象中定义生成器 75
9.3.4　高级生成器 76
9.3.5　抛出错误 78
9.3.6　return 语句 78
9.3.7　任务队列 79
9.3.8　异步生成器 81
9.4　for…of 循环 81
9.5　for await…of 循环 82
9.6　小　结 82

第 10 章　Promise 对象与异步函数 83

10.1　Promise 对象 83
10.1.1　基本语法 83
10.1.2　Promise 的状态 84
10.1.3　then 方法 84
10.1.4　catch 方法 85
10.1.5　all 方法 85
10.1.6　race 方法 86
10.1.7　resolve 方法 86
10.1.8　reject 方法 86
10.1.9　finally 方法 86
10.2　异步函数 86
10.3　小　结 87

第 11 章　代理 88

11.1　Reflect 对象 88
11.2　使用代理 89
11.3　可代理的操作 90

11.3.1 getPrototypeOf 操作 90
11.3.2 setPrototypeOf 操作 90
11.3.3 isExtensible 操作 91
11.3.4 preventExtensions 操作 91
11.3.5 getOwnPropertyDescriptor 操作 91
11.3.6 defineProperty 操作 92
11.3.7 has 操作 92
11.3.8 get 操作 93
11.3.9 set 操作 94
11.3.10 deleteProperty 操作 94
11.3.11 ownKeys 操作 95
11.3.12 apply 操作 96
11.3.13 construct 操作 96
11.4 可撤销的代理对象 97
11.5 小 结 97

第 12 章 类和模块 98

12.1 类 98
12.1.1 类的声明 98
12.1.2 类表达式 98
12.1.3 定义属性 99
12.1.4 定义方法 99
12.1.5 访问器属性 99
12.1.6 静态方法 100
12.1.7 继承 100
12.1.8 使用 super 调用父类同名方法 100
12.2 模 块 101
12.2.1 导出 101
12.2.2 导入 102
12.2.3 合并导出 103
12.2.4 无绑定的导入 103
12.3 模块的加载 104
12.4 小 结 104

第 13 章 自己动手写一个框架 105

13.1 框架的类型 105
13.2 开发工具 105
13.2.1 开发工具的选择 105

13.2.2 安装 Visual Studio Code 106
13.2.3 配置开发环境 110
13.3 编写框架 111
13.3.1 创建和配置项目 111
13.3.2 安装 Gulp 112
13.3.3 安装 Browserify 113
13.3.4 安装 Watchify 113
13.3.5 安装 Babel 113
13.3.6 安装 Uglify 114
13.3.7 配置 Gulp 114
13.3.8 添加 DOM 操作 118
13.3.9 添加样式操作 119
13.3.10 添加样式类操作 122
13.3.11 添加属性操作 123
13.3.12 添加事件操作 125
13.3.13 添加 Ajax 功能 127
13.4 小 结 129

第1章　变量与常量

在 ECMAScript 2015（以下简称 ES 6）之前，定义变量只有唯一一个 var 语句，而且存在不少问题。在 ES 6 中加入了 let 和 const 语句来解决这些长期遗留的问题。为了简化代码，还加入了解构赋值这种定义变量的方式。本章将讲述这些变化。

1.1　var 的问题

比较经典的 var 问题就是循环体内的变量值非预期值，具体代码如下：

```
for(var i=0; i<5; i++){
    setTimeout(function(){
        console.log(i);
    },1000)
}
```

以上代码中，预期的输出是 0~4，实际输出的却全是 5。运行结果非预期的原因是变量提升（hoisting）。所谓变量提升，是指变量可以在声明之前使用，如在 for 语句之前加入“console.log(i);”，会看到输出 undefined，这说明变量 i 在执行 var 语句之前就已经可用，只是还没有赋值。出现变量提示的原因是 JavaScript 在执行时，变量声明总会在任何代码执行之前，也就是说，无论你在任何位置使用 var 声明变量，都等同于在代码开头使用 var 声明变量，如在 for 循环前使用 var 声明变量 i 与在 for 语句的表达式内声明是没有区别的，这也是在 JavaScript 的最佳实践中建议将所有变量声明放在代码头部的主要原因，这样可以尽量减少错误。变量 i 提升的主要后果就是在 setTimeout 的回调函数内共享 i 值，由于回调函数是在循环执行完成后才执行的，这时候的 i 值已经是 5，因此回调函数输出 5 也就不奇怪了。

要解决变量提升问题，方案之一就是将变量值转换为局部变量的值，防止共享，例如以下代码：

```
for(var i=0;i<5;i++){
    (function(i){
        setTimeout(function(){
            console.log(i);
```

```
        },1000);
    }(i));
}
```

以上代码中，通过匿名函数将 i 值转换为匿名函数的局部变量 i 的值，从而避免 setTimeout 的回调函数受共享值的影响，也就能输出预期的值了。虽然以上方法可以很好地解决变量提升的问题，但对于开发人员来说是相当不友好的，大大增加了开发人员的工作量，也增加了犯错误的概率，于是，在 ES 6 加入了 let 语句和 const 语句。

1.2 let 语句

语句 let 与 var 的最大不同是 let 语句定义的是一个块级作用域的变量，也就是说，该变量只在块内有效，在块外是无效的，也不会出现变量提升的问题，例如以下代码：

```
console.log(i);
console.log(j);

{
    var i= 10;
    let j =10;
}
```

执行以上代码会显示 i 是 undefined，而 j 则提示引用错误“ReferenceError: j is not defined”。这说明使用 let 语句定义的变量 j 不会被提升到块外，只能在块内使用。

在上一节的 for 循环代码中，将 var 替换为 let，就不需要进行变量转换了，可直接输出所预期的值。

1.3 使用 let 的好处

1.3.1 避免重复声明

先来看以下代码：

```
var i=10;
var i= 20;
console.log(i);
```

执行以上代码会输出 20，这说明使用 var 定义的变量是可以重复声明的，且会使用新的值覆盖旧的值，而这也是不少开发人员一不小心就会犯错误的地方。将以上代码的 var 替换为 let，执行后就会提示语法错误“Identifier 'i' has already been declared”，表示 i 已经声明了，不能再声明，这就大大减少了犯错误的可能。

1.3.2 避免变量未声明就使用

受变量提升影响，使用 var 声明的变量是可以在声明之前使用的，例如以下代码：

```
function foo(){
    i = 1;
    console.log(i);
    var i = 2;
}
foo();
```

以上代码执行后会输出 1，说明 i 在声明之前就能使用了。这样的代码会令人很疑惑，这里到底是需要使用变量 i 还是写错了变量名？尽管编码规范可以很好地防止这种问题，但还是会有漏网之鱼的。如果在这里将 var 换成 let，代码就不能执行了，直接抛出引用错误“ReferenceError: i is not defined”，从而避免了变量未声明就使用的情况。

1.3.3 避免全局变量成为全局对象的属性

先来看以下代码：

```
var i=10;
console.log(window.i);
```

代码执行后会输出 10，这说明使用 var 声明的全局变量 i 会成为全局对象 window 的一个属性，可通过 window 对象来访问。虽然编码规范一直强调尽量不要使用全局变量，但也会有需要的时候。如果在需要的时候，碰上在不同地方声明了相同的变量，那就是灾难，你可能需要从不同文件中去寻找那些重复声明。

使用 let 来定义变量则很好地解决了这个问题，当出现重复定义的时候，它会提示语法错误，而且不会在 window 对象中添加属性，这样就避免了不必要的污染。将上面代码中的 var 替换为 let，执行后会显示 undefined，说明 let 语句并没有为 window 对象添加 i 属性，这正是我们需要的。

1.3.4 简化代码

在 1.1 节的 for 循环内，为了能让代码正确运行，需要在调用 setTimeout 方法时封装一个匿名函数，代码比较累赘，而使用 let 后，就不需要匿名函数了，大大简化了代码。