

ECMAScript 6 Primer

ES 6 标准入门

(第2版)

阮一峰 著



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

ECMAScript 6 Primer

ES 6 标准入门

(第2版)

阮一峰 著



电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内容简介

ES6（又名 ES2105）是 JavaScript 语言的新标准，2015 年 6 月正式发布后，得到了迅速推广，是目前业界超级活跃的计算机语言。本书是国内仅有的一本 ES6 教程，在前版基础上增补了大量内容——对标准进行了彻底的解读，所有新增的语法知识（包括即将发布的 ES7）都给予了详细介绍，并且紧扣业界开发实践，给出了大量简洁易懂、可以即学即用的示例代码。

本书为中级难度，适合对 JavaScript 语言或 ES5 已经有所了解的读者，用来提高水平，了解这门语言的最新发展；也可当作参考手册，查寻 ES6/ES7 新增的语法点。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

ES6 标准入门 / 阮一峰著. —2 版. —北京：电子工业出版社，2015.12

ISBN 978-7-121-27657-6

I. ①E… II. ①阮… III. ①程序设计 IV. ①TP311.1

中国版本图书馆 CIP 数据核字 (2015) 第 285565 号

策划编辑：张春雨

责任编辑：白涛

印刷：北京中新伟业印刷有限公司

装订：三河市皇庄路通装订厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开本：787×980 1/16 印张：21 字数：538 千字

版次：2015 年 12 月第 1 版

印次：2015 年 12 月第 1 次印刷

定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

推荐序 1

为什么我们要关心标准

“ECMAScript 是 JavaScript 语言的国际标准，JavaScript 是 ECMAScript 的实现。”

本书第 1 章的这句话已经清楚地告诉我们，这是一本不实用的书。我们学习了这本书，并不意味着掌握了一项实用的技术，而只是掌握了一个未来可能会发布的技术标准。而标准，有可能在将来被实现，变成主流，也有可能就仅仅是一个标准，没有人真的去实践它。如果你再了解一下第 1 章里面介绍的 ECMAScript 4.0 草案的血泪史，或者回顾一下曾经红极一时的 XHTML，就更容易明白这一点了。

那我们为什么不直接忽略标准，拥抱实践就好呢？来，我们一起翻开小学课本，跟我念：柏林已经来了命令，阿尔萨斯和洛林的学校只许教德语了……（《最后一课》）

当统治者宣布一门语言成为“标准”的时候，不管是在现实生活还是技术领域里面，往往就意味着所有其他的选项自动消失了，我们只能去学习“统治者”的语言。幸运的是，在技术领域里面，跳出来争取对技术的影响力和主导权，不但不违反任何一国的宪法，往往还是被鼓励的。

因此，技术的未来发展，是我们可以去发出声音，去影响，乃至去引领的。而要做到这些，我们需要搞清楚，ECMA 和各大互联网巨头们，他们正在做什么，正在把技术往哪里引领；他们引领的方向，到底是对所有人有利的，还是只是对某些公司有利；我们中国的开发者和中国的公司，要怎么加入到这些标准的制订过程中，把标准带到更好的方向上。

最近几年，越来越多的中国公司加入到各种国际标准组织中，参与到各种标准（尤其是在东亚文字处理、排版、输入法相关的领域）制订过程中，发出了中国技术人员的声音。随着中国国力的增强，中国开发厂商和技术人员的影响力发展壮大，可以预见，不久的将来，中国工程师也许会深入参与到 ECMAScript 7 和 HTML6 这样的技术标准的制订过程里面，跟各国的专

推荐序 1

家一起探讨，我们中国的开发者不喜欢这样，更喜欢那样。在那些标准大会上，我们的发言权将来自于我们对标准的深入理解、我们对技术发展的独到眼光和我们建设起来的技术影响力。

作为一个 JS 开发者，实话说，对于 ECMAScript 6 里面的很多内容（比如 let 语句），我并不同意。但是很遗憾，这个标准的制定过程没我们什么事。但是如果我们从现在开始关注国际标准，翻译标准文档，让更多人了解标准，更多公司加入标准组织、参与标准制订，也许未来的中国技术圈不但会是很多人的一个圈子，还会是很有影响力的一个圈子。

“我们说的话，让世界都认真听话。”（S.H.E，《中国话》）

腾讯驻 W3C 顾问委员会代表 黄希彤（stone）

黄希彤（网名 emu），Web 性能优化（WPO）领域实践者，信息无障碍领域推动者。腾讯 Web 前端专家，腾讯驻 W3C 顾问委员会代表，腾讯 QQ 空间技术总监。

推荐序 2

因为一件往事，我现在轻易不敢给别人写序或者书评。那天我在想，如果我要给这本书写序，是不是应该先把这本书拿给贺老(hax)看看。后来呢，我到阮一峰老师的GitHub上看了一看，发现这本书有605个star，若干个已解决和未解决的issue，所以我就放心了。开源真是好啊！

这本书是关于ES6的，我对ES6并没有特别系统的研究，但是也在工作中使用了一部分ES6的特性，使用得最多的是Promise，其他的特性只是研究，很少使用，主要是因为本身支持ES6的环境和工具有限。浏览器就不说了，现在的前端工程师在一些产品中能够抛弃IE6已经是很幸福的事情了，但是即使是IE8，离真正的ES6也还很遥远。在其他领域，比如手机游戏领域，cocos2d-js v3.0使用的脚本引擎是SpiderMonkey v28，因此情况要好很多，但是周边的一些工具，比如closure compiler不能很好地压缩和优化ES6，当然你可以采用转换工具先将ES6转成ES5，然后再做压缩和优化，但是这多出来的一步造成更多出错的可能，而且和享受ES6的语法糖的快乐相比，开销有点大——如果无论如何需要再转一步，那么为什么我们不干脆考虑TypeScript或者其他选择呢？

为什么会选择使用ES6的Promise，那是因为Promise算是比较好解决异步嵌套问题的方案，另外Promise本身在低版本下也有比较好的polyfill实现(<https://github.com/jakearchibald/es6-promise>)，对于我和一些前端工程师来说，是十分乐意为将来去写一些能够向前兼容的符合标准的代码的。

目前这个阶段，前端学习ES6，并不意味着能够很快将ES6的好处带到工作中，因为我们毕竟还受到现在的浏览器环境的制约。但是，即使单纯从学习一门编程语言的核心API的角度来说，ES6也是值得学习的。它的很多新特性，真正涉及现代编程语言概念中很流行的部分，不管是解构赋值还是迭代器或者yield，都是超棒超赞的思想，不但易于理解，也能节省很多键盘操作，而另一些诸如const、作用域之类的设定，则让脚本引擎代替程序员人肉检查做更多的事情，让我们最终上线的代码变得更加安全和更加优美。

不管怎样，ES6代表着一种前端的未来，这种未来，无疑能让前端工程师们工作得更高效，也更有乐趣。更进一步说，ECMAScript还是开放的标准，对这门语言的新特性，有什么好的想法，

推荐序 2

都是有机会提交为标准的，也就是说，前端程序员的未来，是由我们前端程序员自己来创造的，还有什么比自由更加美好的呢？所以，为了未来，加油！

— 360 奇舞团团长 月影

吴亮（网名月影），先后在微软亚洲研究院做过访问学生，在金蝶软件有限公司担任过核心开发工程师、设计师和项目经理，在百度电子商务事业部担任过 Web 开发项目经理。现任奇虎 360 高级技术经理，360 前端团队奇舞团负责人。多年来致力于 JavaScript 技术和 Web 标准的推广，活跃于国内各技术社区，现为 w3ctech 顾问。

推荐序 3

同大多数读者一样，我最早看到阮一峰先生的文字是在其博客上。他的第一篇博文于 2003 年写就，迄今已有 1500 多篇文章，可谓高产。阮先生并非计算机相关专业，但这一点并没有妨碍他从事技术写作，其文字朴实，思路清晰，所有人都能看懂，更能感受到他写文章的用心程度，而这本书完美地体现了他的一贯风格。另外，这本书是开源作品，也很好地践行了他一贯的贡献原则。

自我写下第一行前端代码到现在已经十来年了，前端的基础设施也发生了巨大的变化。变化最大的还是浏览器环境，从原来烂熟 IE6 的各种 bug 和 hack，到现在 IE6 已经完全不在我的考虑范围内。其次是前端的工程化程度，2011 年，我做 FIS (<http://fis.baidu.com>) 时，完全没想到前端的工程化进展会如此之快。而变化最慢的，要数语言本身了，1999 年发布的 ECMAScript 3.0 其实相当于第 1 版；十年后的 2009 年，才发布第 2 版：ECMAScript 5.0；预计 ECMAScript 6 要到 2015 年发布。

我的一贯主张是，要学好 JavaScript，ECMAScript 标准比什么书都强。在标准中，已经用最严谨的语言和最完美的角度展现了语言的实质和特性。理解语言的本质后，你已经从沙堆里挑出了珍珠，能经受得起时光的磨砺。

我从 2009 年开始正式接触 ECMAScript 规范，当时我在写百度的 JavaScript 基础库 Tangram 1.0，ECMAScript 5 还处于草案状态。我自己打印了一本小册子，上下班时在地铁上慢慢看。才知道有很多问题在网络上被包装了太多次，解释得千奇百怪，但用规范的语言来描述，竟是如此简单。

ECMAScript 标准经历了很多变故——尤其是 ECMAScript 4 那次——也从语言的角度反映了各大厂商之间的立场差异。不过，ECMAScript 5 的正式发布和发展，为所有 Web 开发者奠定了稳定的基础，尽管浏览器之间存在大量差异，尤其是 DOM，但在 JavaScript 语言层面，都相对严格地遵循着 ECMAScript 5 的规范。

JavaScript 遵守“一个 JavaScript”的原则，所有版本都需要向后兼容。Web 语言的解释器版本不是由开发者而是由用户决定的，所以 JavaScript 无法像 Python、Ruby、Perl 那样，发布一

推荐序 3

个不向下兼容的大版本，这也就是 ECMAScript 4 失败的根源，由于它会导致大量已有网页的“bug”，浏览器厂商会强烈反对。当然，ECMAScript 6 的 strict mode 也在尝试逐步淘汰一些不良实践。

ECMAScript 6 相比 5，有了很大的进步。经过这次改进，JavaScript 语法更精简，变得更有表现力了；在严格模式下，开发者受到了适当而必要的约束；新增了几种数据类型（map、set）和函数能力（Generator、迭代器）；进一步强化了 JavaScript 的特点（promise、proxy）；并且让 JavaScript 能适用于更大型的程序开发（modules、class）。更重要的是，这个规范会被浏览器厂商、不同的平台广泛支持。

实际上，所有的语言改进都是从使用者的最佳实践中提炼出来的。JavaScript 的约束一直很少，这一灵活性让开发者能相当自由地积累形形色色的使用经验和实践，也就是说，我们所有 ECMAScript 的使用者，也是其标准的间接贡献者。

百度高级工程师，前端通用组技术负责人 雷志兴

雷志兴（网名 berg），资深工程师，2007 年加入百度工作至今，负责过多项前端基础技术、架构的设计和搭建；骑行爱好者，行程万余公里；微信公众号“行云出岫”（DevLife）的维护者。

前言

2012年年底，我开始动手做一个开源项目《JavaScript 标准参考教程》（<https://github.com/ruanyf/jstutorial>）。原来的设想是将自己的学习笔记整理成一本书，哪里料到，这个项目不断膨胀，最后变成了关于 ECMAScript 5 及其外围 API 的全面解读和参考手册，写了一年多还没写完。

那个项目的最后一章就是 ECMAScript 6 的语法简介。那一章也是越写越长，最后我不得不决定，把它独立出来，作为一个新项目，也就是您现在看到的这本书。

JavaScript 已经是互联网开发的第一大语言，而且正在变成一种全领域的语言。著名程序员 Jeff Atwood 甚至提出了一条“*Atwood 定律*”：“所有可以用 JavaScript 编写的程序，最终都会出现 JavaScript 的版本。”（Any application that can be written in JavaScript will eventually be written in JavaScript.）

ECMAScript 正是 JavaScript 的国际标准，这就决定了该标准的重要性。而 ECMAScript 6 是 ECMAScript 历史上最大的一次版本升级，在语言的各个方面都有极大的变化，即使是熟练的 JavaScript 程序员，也需要重新学习。由于 ES6 的设计目标是企业级开发和大型项目，所以可以预料，除了互联网开发者，将来还会有大量应用程序开发者（甚至操作系统开发者）成为 ES6 的学习者。

我写作这本书的目标，就是想为上面这些学习者，提供一本篇幅较短、简明易懂、符合中文表达习惯的 ES6 教程。它由浅入深、循序渐进，既有重要概念的讲解，又有 API 接口的罗列，便于日后当作参考手册查阅，还提供大量示例代码，让读者不仅一看就懂，还能举一反三，直接复制用于实际项目之中。

需要声明的是，为了突出重点，本书只涉及 ES6 与 ES5 的不同之处，不对 JavaScript 已有的语法做全面讲解，毕竟市场上这样的教程已有很多了。因此，本书不是 JavaScript 入门教材，不适合初学者。阅读本书之前，需要对 JavaScript 的基本语法有所了解。

前言

我本人也是一个 ES6 的学习者，不敢说自己有多高的水平，只是较早地接触了这个主题，持续地读了许多资料，追踪标准的进展，做了详细的笔记而已。虽然我尽了最大努力，并且原稿在 GitHub 上公开后，已经得到了大量的勘误，但是本书的不如人意之处恐怕还是有不少。

欢迎大家访问本书的项目主页（<https://github.com/ruanyf/es6tutorial>），提出意见，以及提交 pull request。这些都会包括在本书的下一个版本中。

阮一峰

2014 年 6 月 4 日，写于上海

目录

第 1 章 ECMAScript 6 简介	001	2.3 const 命令	015
1.1 ECMAScript 和 JavaScript 的关系	001	2.4 跨模块常量	016
1.2 ECMAScript 的历史	001	2.5 全局对象的属性	017
1.3 部署进度	002	第 3 章 变量的解构赋值	018
1.4 Babel 转码器	003	3.1 数组的解构赋值	018
命令行环境	004	基本用法	018
浏览器环境	005	默认值	020
Node.js 环境	005	3.2 对象的解构赋值	021
在线转换	006	3.3 字符串的解构赋值	024
1.5 Traceur 转码器	006	3.4 数值和布尔值的解构赋值	024
直接插入网页	006	3.5 函数参数的解构赋值	024
在线转换	007	3.6 圆括号问题	025
命令行转换	008	不能使用圆括号的情况	025
Node.js 环境的用法	008	可以使用圆括号的情况	026
1.6 ECMAScript 7	009	3.7 用途	026
第 2 章 let 和 const 命令	010	第 4 章 字符串的扩展	029
2.1 let 命令	010	4.1 字符的 Unicode 表示法	029
基本用法	010	4.2 codePointAt()	030
不存在变量提升	011	4.3 String.fromCodePoint()	031
暂时性死区	011	4.4 字符串的遍历器接口	031
不允许重复声明	012	4.5 at()	032
2.2 块级作用域	013	4.6 normalize()	032
为什么需要块级作用域	013	4.7 includes(), startsWith(), endsWith()	033
ES6 的块级作用域	013	4.8 repeat()	033

4.9	padStart(), padEnd()	034	第 7 章 数组的扩展	059	
4.10	模板字符串	034	7.1	Array.from()	059
4.11	实例: 模板编译	037	7.2	Array.of()	061
4.12	标签模板	038	7.3	数组实例的 copyWithin()	062
4.13	String.raw()	042	7.4	数组实例的 find() 和 findIndex()	063
第 5 章 正则的扩展		043	7.5	数组实例的 fill()	063
5.1	RegExp 构造函数	043	7.6	数组实例的 entries()、keys() 和 values()	064
5.2	字符串的正则方法	043	7.7	数组实例的 includes()	064
5.3	u 修饰符	043	7.8	数组的空位	065
5.4	y 修饰符	045	7.9	数组推导	067
5.5	sticky 属性	047	第 8 章 函数的扩展	069	
5.6	flags 属性	048	8.1	函数参数的默认值	069
5.7	RegExp.escape()	048		基本用法	069
第 6 章 数值的扩展		049		与解构赋值默认值结合使用	070
6.1	二进制和八进制数值表示法	049		参数默认值的位置	071
6.2	Number.isFinite(), Number.isNaN()	049		函数的 length 属性	072
6.3	Number.parseInt(), Number.parseFloat()	051		作用域	073
6.4	Number.isInteger()	051		应用	074
6.5	Number.EPSILON	051	8.2	rest 参数	074
6.6	安全整数和 Number.isSafeInteger()	052	8.3	扩展运算符	075
6.7	Math 对象的扩展	054		含义	075
	Math.trunc()	054		替代数组的 apply 方法	076
	Math.sign()	054		扩展运算符的应用	077
	Math.cbrt()	055	8.4	name 属性	079
	Math.clz32()	055	8.5	箭头函数	080
	Math.imul()	056		基本用法	080
	Math.fround()	056		使用注意点	081
	Math.hypot()	057		嵌套的箭头函数	083
	对数方法	057	8.6	函数绑定	084
	三角函数方法	058	8.7	尾调用优化	085
6.8	指数运算符	058		什么是尾调用	085
				尾调用优化	086

尾递归	087	第 11 章 Proxy 和 Reflect	116
递归函数的改写	088	11.1 Proxy 概述	116
8.8 函数参数的尾逗号	089	11.2 Proxy 实例的方法	119
第 9 章 对象的扩展	090	get()	119
9.1 属性的简洁表示法	090	set()	121
9.2 属性名表达式	092	apply()	122
9.3 方法的 name 属性	093	has()	123
9.4 Object.is()	094	construct()	123
9.5 Object.assign()	095	deleteProperty()	124
9.6 属性的可枚举性	097	defineProperty()	125
9.7 属性的遍历	098	enumerate()	125
9.8 __proto__ 属性, Object.setPrototypeOf(), Object.getPrototypeOf()	099	getOwnPropertyDescriptor()	126
9.9 对象的扩展运算符	101	getPrototypeOf()	126
第 10 章 Symbol	103	isExtensible()	127
10.1 概述	103	ownKeys()	127
10.2 作为属性名的 Symbol	105	preventExtensions()	128
10.3 实例: 消除魔术字符串	106	setPrototypeOf()	128
10.4 属性名的遍历	107	11.3 Proxy.revocable()	129
10.5 Symbol.for(), Symbol.keyFor()	109	11.4 Reflect 概述	129
10.6 内置的 Symbol 值	110	11.5 Reflect 对象的方法	130
Symbol.hasInstance	110	第 12 章 二进制数组	133
Symbol.isConcatSpreadable	110	12.1 ArrayBuffer 对象	134
Symbol.species	111	概述	134
Symbol.match	111	ArrayBuffer.prototype.byteLength	135
Symbol.replace	112	ArrayBuffer.prototype.slice()	135
Symbol.search	112	ArrayBuffer.isView()	136
Symbol.split	112	12.2 TypedArray 视图	136
Symbol.iterator	112	概述	136
Symbol.toPrimitive	113	构造函数	136
Symbol.toStringTag	113	数组方法	138
Symbol.unscopables	114	字节序	140
		BYTES_PER_ELEMENT 属性	141
		ArrayBuffer 与字符串的互相转换	142
		溢出	142

TypedArray.prototype.buffer	143	14.5	Iterator 接口与 Generator 函数	174
TypedArray.prototype.byteLength,		14.6	遍历器对象的 return()、throw()	175
TypedArray.prototype.byteOffset	143			
TypedArray.prototype.length	143	14.7	for...of 循环	175
TypedArray.prototype.set()	144		数组	175
TypedArray.prototype.subarray()	144		Set 和 Map 结构	177
TypedArray.prototype.slice()	144		计算生成的数据结构	177
TypedArray.of()	144		类似数组的对象	178
TypedArray.from()	145		对象	179
12.3 复合视图	145		与其他遍历语法的比较	180
12.4 DataView 视图	146	第 15 章	Generator 函数	181
12.5 二进制数组的应用	148	15.1	简介	181
AJAX	148		基本概念	181
Canvas	148		yield 语句	182
WebSocket	149		与 Iterator 接口的关系	184
Fetch API	149	15.2	next 方法的参数	185
File API	149	15.3	for...of 循环	187
第 13 章 Set 和 Map 数据结构	151	15.4	Generator.prototype.throw()	189
13.1 Set	151	15.5	Generator.prototype.return()	193
基本用法	151	15.6	yield* 语句	194
Set 实例的属性和方法	152	15.7	作为对象属性的 Generator 函数	199
遍历操作	153	15.8	Generator 函数的 this	200
13.2 WeakSet	155	15.9	Generator 函数推导	201
13.3 Map	157	15.10	含义	202
Map 结构的目的是基本用法	157		Generator 与状态机	202
实例的属性和操作方法	159		Generator 与协程	202
遍历方法	160	15.11	应用	203
与其他数据结构的互相转换	162	第 16 章	Promise 对象	208
13.4 WeakMap	164	16.1	Promise 的含义	208
第 14 章 Iterator 和 for...of 循环	166	16.2	基本用法	208
14.1 Iterator 的概念	166	16.3	Promise.prototype.then()	211
14.2 数据结构的默认 Iterator 接口	168	16.4	Promise.prototype.catch()	212
14.3 调用 Iterator 接口的场合	172	16.5	Promise.all()	215
14.4 字符串的 Iterator 接口	173			

16.6	Promise.race()	216	17.5	async 函数	234
16.7	Promise.resolve()	216		含义	234
16.8	Promise.reject()	217		async 函数的实现	235
16.9	两个有用的附加方法	218		async 函数的用法	236
	done()	218		注意点	236
	finally()	218		与 Promise、Generator 的比较	238
16.10	应用	219	第 18 章	Class	240
	加载图片	219	18.1	Class 基本语法	240
	Generator 函数与 Promise 的结合	219		概述	240
16.11	async 函数	220		constructor 方法	243
第 17 章	异步操作和 async 函数	221		实例对象	243
17.1	基本概念	221		name 属性	244
	异步	221		Class 表达式	245
	回调函数	221		不存在变量提升	245
	Promise	222		严格模式	246
17.2	Generator 函数	223	18.2	Class 的继承	246
	协程	223		基本用法	246
	Generator 函数的概念	223		类的 prototype 属性和 __proto__ 属性	247
	Generator 函数的数据交换和错误处理	224		extends 的继承目标	248
	异步任务的封装	224		Object.getPrototypeOf()	249
17.3	Thunk 函数	225		super 关键字	249
	参数的求值策略	225		实例的 __proto__ 属性	250
	Thunk 函数的含义	226	18.3	原生构造函数的继承	250
	JavaScript 语言的 Thunk 函数	226	18.4	Class 的取值函数 (getter) 和存 值函数 (setter)	253
	Thunkify 模块	227	18.5	Class 的 Generator 方法	254
	Generator 函数的流程管理	228	18.6	Class 的静态方法	254
	Thunk 函数的自动流程管理	229	18.7	Class 的静态属性	255
17.4	co 模块	230	18.8	new.target 属性	256
	基本用法	230	18.9	Mixin 模式的实现	258
	co 模块的原理	230	第 19 章	修饰器	259
	基于 Promise 对象的自动执行	231	19.1	类的修饰	259
	co 模块的源码	232	19.2	方法的修饰	261
	处理并发的异步操作	233			

目录

19.3	为什么修饰器不能用于函数	262	21.4	对象	286
19.4	core-decorators.js	263	21.5	数组	288
	@autobind	263	21.6	函数	288
	@readonly	263	21.7	Map 结构	289
	@override	263	21.8	Class	290
	@deprecated (别名 @deprecated)	264	21.9	模块	291
	@suppressWarnings	265	21.10	ESLint 的使用	292
19.5	使用修饰器实现自动发布事件	265	第 22 章 读懂 ECMAScript 规格		293
19.6	Mixin	266	22.1	概述	293
19.7	Trait	267	22.2	相等运算符	293
19.8	Babel 转码器的支持	269	22.3	数组的空位	295
第 20 章 Module		270	22.4	数组的 map 方法	296
20.1	严格模式	271	第 23 章 参考链接		299
20.2	export 命令	271	23.1	官方文件	299
20.3	import 命令	272	23.2	综合介绍	299
20.4	模块的整体加载	273	23.3	let 和 const	302
20.5	module 命令	274	23.4	解构赋值	302
20.6	export default 命令	274	23.5	字符串	303
20.7	模块的继承	276	23.6	正则	304
20.8	ES6 模块加载的实质	277	23.7	数值	304
20.9	循环加载	278	23.8	数组	304
	CommonJS 模块的加载原理	279	23.9	函数	305
	CommonJS 模块的循环加载	279	23.10	对象	306
	ES6 模块的循环加载	280	23.11	Proxy 和 Reflect	306
20.10	ES6 模块的转码	282	23.12	Symbol	308
	ES6 module transpiler	282	23.13	二进制数组	308
	SystemJS	282	23.14	Set 和 Map	309
第 21 章 编程风格		284	23.15	Iterator	310
21.1	块级作用域	284	23.16	Generator	311
	let 取代 var	284	23.17	Promise 对象	312
	全局常量和线程安全	285	23.18	Class	314
	严格模式	285	23.19	Decorator	314
21.2	字符串	285	23.20	Module	315
21.3	解构赋值	285	23.21	工具	316