- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键字等。
- 加粗等宽字体（**`constant width bold`**）
  表示应该由用户输入的命令或其他文本。
- 等宽斜体（*`constant width italic`*）
  表示应该由用户输入的值或根据上下文确定的值替换的文本。

该图标表示提示或建议。

该图标表示一般注记。

该图标表示警告或警示。

使用代码示例

补充材料（代码示例、练习等）可以从 https://github.com/getify/You-Dont-Know-JS/tree/master/types%20&%20grammar 和 https://github.com/getify/You-Dont-Know-JS/tree/master/async & performance 下载。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN，比如："*You Don't Know JavaScript: Types & Grammar* by Kyle Simpson (O'Reilly). Copyright 2015 Getify Solutions, Inc., 978-1-491-90419-0"。

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

Safari® Books Online

Safari Books Online（http://www.safaribooksonline.com）是应运而生的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O'Reilly 的每一本书都有专属网页，你可以在那儿找到本书的相关信息，包括勘误表、示例代码以及其他信息。本书第一部分“类型和语法”的网站地址是 http://shop.oreilly.com/product/0636920033745.do。本书第二部分“异步和性能”的网址是 http://shop.oreilly.com/product/0636920033752.do。

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O’Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址：http://facebook.com/oreilly

请关注我们的 Twitter 动态：http://twitter.com/oreillymedia

我们的 YouTube 视频地址：http://www.youtube.com/oreillymedia

致谢

我要感谢很多人，是他们的帮助让本书以及整个系列得以出版。

首先，我要感谢我的妻子 Christen Simpson 以及我的两个孩子 Ethan 和 Emily，容忍我整天坐在电脑前工作。即使不写作的时候，我的眼睛也总是盯着屏幕做一些与 JavaScript 相关的工作。我牺牲了很多陪伴家人的时间，这个系列的丛书才得以为读者深入全面地介绍 JavaScript。对于家庭，我亏欠太多。

我要感谢 O’Reilly 的编辑 Simon St.Laurent 和 Brian MacDonald，以及所有其他的编辑和市场工作人员。和他们一起工作非常愉快；本系列丛书的写作、编辑和制作都以开源方式进行，在此实验过程中，他们给予了非常多的帮助。

我要感谢所有为本系列丛书提供建议和校正的人，包括 Shelley Powers、Tim Ferro、Evan Borden、Forrest L. Norvell、Jennifer Davis、Jesse Harlin 等。十分感谢 David Walsh 和 Jake Archibald 为本书作序。

我要感谢 JavaScript 社区中的许多人，包括 TC39 委员会的成员们，将他们的知识与我们分享，并且耐心详尽地回答我无休止的提问。他们是 John-David Dalton、Juriy“kangax”Zaytsev、Mathias Bynens、Rick Waldron、Axel Rauschmayer、Nicholas Zakas、Angus Croll、Jordan Harband、Reginald Braithwaite、Dave Herman、Brendan Eich、Allen Wirfs-Brock、Bradley Meck、Domenic Denicola、David Walsh、Tim Disney、Kris Kowal、Peter van der Zee、Andrea Giammarchi、Kit Cambridge，等等。还有很多人，我无法一一感谢。

“你不知道的 JavaScript”系列丛书是由 Kickstarter 发起的，我要感谢近 500 名慷慨的支持者，没有他们的支持就没有这套系列丛书：

Jan Szpila、nokiko、Murali Krishnamoorthy、Ryan Joy、Craig Patchett、pdqtrader、Dale Fukami、ray hatfield、R0drigo Perez [Mx]、Dan Petitt、Jack Franklin、Andrew Berry、Brian Grinstead、Rob Sutherland、Sergi Meseguer、Phillip Gourley、Mark Watson、Jeff Carouth、Alfredo Sumaran、Martin Sachse、Marcio Barrios、Dan、AimelyneM、Matt Sullivan、

Delnatte Pierre-Antoine、Jake Smith、Eugen Tudorancea、Iris、David Trinh、simonstl、Ray Daly、Uros Gruber、Justin Myers、Shai Zonis、Mom & Dad、Devin Clark、Dennis Palmer、Brian Panahi Johnson、Josh Marshall、Marshall、Dennis Kerr、Matt Steele、Erik Slagter、Sacah、Justin Rainbow、Christian Nilsson、Delapouite、D.Pereira、Nicolas Hoizey、George V. Reilly、Dan Reeves、Bruno Laturner、Chad Jennings、Shane King、Jeremiah Lee Cohick、od3n、Stan Yamane、Marko Vucinic、Jim B、Stephen Collins、Ægir Þorsteinsson、Eric Pederson、Owain、Nathan Smith、Jeanetteurphy、Alexandre ELISÉ、Chris Peterson、Rik Watson、Luke Matthews、Justin Lowery、Morten Nielsen、Vernon Kesner、Chetan Shenoy、Paul Tregoing、Marc Grabanski、Dion Almaer、Andrew Sullivan、Keith Elsass、Tom Burke、Brian Ashenfelter、David Stuart、Karl Swedberg、Graeme、Brandon Hays、John Christopher、Gior、manoj reddy、Chad Smith、Jared Harbour、Minoru TODA、Chris Wigley、Daniel Mee、Mike、Handyface、Alex Jahraus、Carl Furrow、Rob Foulkrod、Max Shishkin、Leigh Penny Jr.、Robert Ferguson、Mike van Hoenselaar、Hasse Schougaard、rajan venkataguru、Jeff Adams、Trae Robbins、Rolf Langenhuijzen、Jorge Antunes、Alex Koloskov、Hugh Greenish、Tim Jones、Jose Ochoa、Michael Brennan-White、Naga Harish Muvva、Barkóczi Dávid、Kitt Hodsden、Paul McGraw、Sascha Goldhofer、Andrew Metcalf、Markus Krogh、Michael Mathews、Matt Jared、Juanfran、Georgie Kirschner、Kenny Lee、Ted Zhang、Amit Pahwa、Inbal Sinai、Dan Raine、Schabse Laks、Michael Tervoort、Alexandre Abreu、Alan Joseph Williams、NicolasD、Cindy Wong、Reg Braithwaite、LocalPCGuy、Jon Friskics、Chris Merriman、John Pena、Jacob Katz、Sue Lockwood、Magnus Johansson、Jeremy Crapsey、Grzegorz Pawłowski、nico nuzzaci、Christine Wilks、Hans Bergren、charles montgomery、Ariel בר-לבב Fogel、Ivan Kolev、Daniel Campos、Hugh Wood、Christian Bradford、Frédéric Harper、Ionuț Dan Popa、Jeff Trimble、Rupert Wood、Trey Carrico、Pancho Lopez、Joël kuijten、Tom A Marra、Jeff Jewiss、Jacob Rios、Paolo Di Stefano、Soledad Penades、Chris Gerber、Andrey Dolganov、Wil Moore III、Thomas Martineau、Kareem、Ben Thouret、Udi Nir、Morgan Laupies、jory carson-burson、Nathan L Smith、Eric Damon Walters、Derry Lozano-Hoyland、Geoffrey Wiseman、mkeehner、KatieK、Scott MacFarlane、Brian LaShomb、Adrien Mas、christopher ross、Ian Littman、Dan Atkinson、Elliot Jobe、Nick Dozier、Peter Wooley、John Hoover、dan、Martin A. Jackson、Héctor Fernando Hurtado、andy ennamorato、Paul Seltmann、Melissa Gore、Dave Pollard、Jack Smith、Philip Da Silva、Guy Israeli、@megalithic、Damian Crawford、Felix Gliesche、April Carter Grant、Heidi、jim tierney、Andrea Giammarchi、Nico Vignola、Don Jones、Chris Hartjes、Alex Howes、john gibbon、David J. Groom、BBox、Yu Dilys Sun、Nate Steiner、Brandon Satrom、Brian Wyant、Wesley Hales、Ian Pouncey、Timothy Kevin Oxley、George Terezakis、sanjay raj、Jordan Harband、Marko McLion、Wolfgang Kaufmann、Pascal Peuckert、Dave Nugent、Markus Liebelt、Welling Guzman、Nick Cooley、Daniel Mesquita、Robert Syvarth、Chris Coyier、Rémy Bach、Adam Dougal、Alistair

Duggin、David Loidolt、Ed Richer、Brian Chenault、GoldFire Studios、Carles Andrés、Carlos Cabo、Yuya Saito、roberto ricardo、Barnett Klane、Mike Moore、Kevin Marx、Justin Love、Joe Taylor、Paul Dijou、Michael Kohler、Rob Cassie、Mike Tierney、Cody Leroy Lindley、tofuji、Shimon Schwartz、Raymond、Luc De Brouwer、David Hayes、Rhys Brett-Bowen、Dmitry、Aziz Khoury、Dean、Scott Tolinski - Level Up、Clement Boirie、Djordje Lukic、Anton Kotenko、Rafael Corral、Philip Hurwitz、Jonathan Pidgeon、Jason Campbell、Joseph C.、SwiftOne、Jan Hohner、Derick Bailey、getify、Daniel Cousineau、Chris Charlton、Eric Turner、David Turner、Joël Galeran、Dharma Vagabond、adam、Dirk van Bergen、dave ♥♫★ furf、Vedran Zakanj、Ryan McAllen、Natalie Patrice Tucker、Eric J. Bivona、Adam Spooner、Aaron Cavano、Kelly Packer、Eric J、Martin Drenovac、Emilis、Michael Pelikan、Scott F. Walter、Josh Freeman、Brandon Hudgeons、vijay chennupati、Bill Glennon、Robin R.、Troy Forster、otaku_coder、Brad、Scott、Frederick Ostrander、Adam Brill、Seb Flippence、Michael Anderson、Jacob、Adam Randlett、Standard、Joshua Clanton、Sebastian Kouba、Chris Deck、SwordFire、Hannes Papenberg、Richard Woeber、hnzz、Rob Crowther、Jedidiah Broadbent、Sergey Chernyshev、Jay-Ar Jamon、Ben Combee、luciano bonachela、Mark Tomlinson、Kit Cambridge、Michael Melgares、Jacob Adams、Adrian Bruinhout、Bev Wieber、Scott Puleo、Thomas Herzog、April Leone、Daniel Mizieliński、Kees van Ginkel、Jon Abrams、Erwin Heiser、Avi Laviad、David newell、Jean-Francois Turcot、Niko Roberts、Erik Dana、Charles Neill、Aaron Holmes、Grzegorz Ziółkowski、Nathan Youngman、Timothy、Jacob Mather、Michael Allan、Mohit Seth、Ryan Ewing、Benjamin Van Treese、Marcelo Santos、Denis Wolf、Phil Keys、Chris Yung、Timo Tijhof、Martin Lekvall、Agendine、Greg Whitworth、Helen Humphrey、Dougal Campbell、Johannes Harth、Bruno Girin、Brian Hough、Darren Newton、Craig McPheat、Olivier Tille、Dennis Roethig、Mathias Bynens、Brendan Stromberger、sundeep、John Meyer、Ron Male、John F Croston III、gigante、Carl Bergenhem、B.J. May、Rebekah Tyler、Ted Foxberry、Jordan Reese、Terry Suitor、afeliz、Tom Kiefer、Darragh Duffy、Kevin Vanderbeken、Andy Pearson、Simon Mac Donald、Abid Din、Chris Joel、Tomas Theunissen、David Dick、Paul Grock、Brandon Wood、John Weis、dgrebb、Nick Jenkins、Chuck Lane、Johnny Megahan、marzsman、Tatu Tamminen、Geoffrey Knauth、Alexander Tarmolov、Jeremy Tymes、Chad Auld、Sean Parmelee、Rob Staenke、Dan Bender、Yannick derwa、Joshua Jones、Geert Plaisier、Tom LeZotte、Christen Simpson、Stefan Bruvik、Justin Falcone、Carlos Santana、Michael Weiss、Pablo Villoslada、Peter deHaan、Dimitris Iliopoulos、seyDoggy、Adam Jordens、Noah Kantrowitz、Amol M、Matthew Winnard、Dirk Ginader、Phinam Bui、David Rapson、Andrew Baxter、Florian Bougel、Michael George、Alban Escalier、Daniel Sellers、Sasha Rudan、John Green、Robert Kowalski、David I. Teixeira (@ditma、Charles Carpenter、Justin Yost、Sam S、Denis Ciccale、Kevin Sheurs、Yannick Croissant、Pau Fracés、Stephen McGowan、Shawn Searcy、Chris Ruppel、Kevin Lamping、Jessica Campbell、Christopher

Schmitt、Sablons、Jonathan Reisdorf、Bunni Gek、Teddy Huff、Michael Mullany、Michael Fürstenberg、Carl Henderson、Rick Yoesting、Scott Nichols、Hernán Ciudad、Andrew Maier、Mike Stapp、Jesse Shawl、Sérgio Lopes、jsulak、Shawn Price、Joel Clermont、Chris Ridmann、Sean Timm、Jason Finch、Aiden Montgomery、Elijah Manor、Derek Gathright、Jesse Harlin、Dillon Curry、Courtney Myers、Diego Cadenas、Arne de Bree、João Paulo Dubas、James Taylor、Philipp Kraeutli、Mihai Păun、Sam Gharegozlou、joshjs、Matt Murchison、Eric Windham、Timo Behrmann、Andrew Hall、joshua price、Théophile Villard。

这套系列丛书的写作、编辑和制作都是以开源的方式进行的。我们要感谢 GitHub 让这一切成为可能！

再次向我没能提及的支持者们表示感谢。这套系列丛书属于我们每一个人，希望它能够帮助更多的人更好地了解 JavaScript，让当前和未来的社区贡献者受益。

第一部分

类型和语法

姜 南 译

序

有人说，JavaScript 是唯一一门可以先用后学的编程语言。

每次听到这话我都会心一笑，因为我自己就是这样，我猜很多开发人员可能也是如此。JavaScript，也许还包括 CSS 和 HTML，在互联网早期的大学计算机课程中并不是主流教学语言。初学者大多通过搜索引擎和“查看源代码”的方式来自学。

我仍然记得自己在高中时代开发的第一个网站。那是一个网上商店。因为是《007》的粉丝，所以我决定创建一家“黄金眼”商店。它应有尽有，背景音乐是“黄金眼”的主题曲，有一个用 JavaScript 开发的瞄准器在屏幕上跟随鼠标移动，并且每次点击鼠标就会发出一声枪响。想必 Q（《007》中的一个角色）也会为这个杰作感到骄傲吧。

之所以讲到这个故事，是因为我当时使用的开发方式直到现在仍然有许多开发人员在使用，那就是“复制＋粘贴”。在项目中我“复制＋粘贴”了大量 JavaScript 代码，但根本没有真正理解它们。那些十分流行的 JavaScript 工具库，如 jQuery，也在潜移默化地影响着我们，使我们不用再去深入了解 JavaScript 的本质。

我并不反对使用 JavaScript 工具库，实际上我还是 MooTools JavaScript 团队的一员。这些工具库之所以功能强大，正是因为它们的开发者理解这门语言的本质和优点，并将它们运用到了极致。学会使用这些工具库大有裨益，同时掌握这门语言的基础知识仍然是十分重要的。现在有了 Kyle Simpson 的“你不知道的 JavaScript”系列丛书，我们更有理由好好学习了。

《类型和语法》是该系列的第三本书，它介绍了 JavaScript 的核心基础知识，这些知识我们永远不可能从“复制＋粘贴”和 JavaScript 工具库中学到。本书对强制类型转换及其隐患、原生构造函数，以及 JavaScript 的所有基础知识，都做了详细的介绍，并配以示例代码。同本系列的其他作品一样，Kyle 的行文切中要点，没有多余的套话和修辞，正是我喜欢的技术书的风格。

希望大家喜欢这本书，并能够常读常新。

David Walsh（http://davidwalsh.name）
Mozilla 资深开发人员

第 1 章

类型

大多数开发者认为，像 JavaScript 这样的动态语言是没有类型（type）的。让我们来看看 ES5.1 规范（http://www.ecma-international.org/ecma-262/5.1/）对此是如何界定的：

> 本规范中的运算法则所操纵的值均有相应的类型。本节中定义了所有可能出现的类型。ECMAScript 类型又进一步细分为语言类型和规范类型。
>
> ECMAScript 语言中所有的值都有一个对应的语言类型。ECMAScript 语言类型包括 Undefined、Null、Boolean、String、Number 和 Object。

喜欢强类型（又称静态类型）语言的人也许会认为“类型”一词用在这里不妥。“类型”在强类型语言中的涵义要广很多。

也有人认为，JavaScript 中的“类型”应该称为“标签”（tag）或者“子类型”（subtype）。

本书中，我们这样来定义“类型”（与规范类似）：对语言引擎和开发人员来说，类型是值的内部特征，它定义了值的行为，以使其区别于其他值。

换句话说，如果语言引擎和开发人员对 `42`（数字）和 `"42"`（字符串）采取不同的处理方式，那就说明它们是不同的类型，一个是 `number`，一个是 `string`。通常我们对数字 `42` 进行数学运算，而对字符串 `"42"` 进行字符串操作，比如输出到页面。它们是不同的类型。

上述定义并非完美，不过对于本书已经足够，也和 JavaScript 语言对自身的描述一致。

1.1 类型

撇开学术界对类型定义的分歧，为什么说 JavaScript 是否有类型也很重要呢？

要正确合理地进行类型转换（参见第 4 章），我们必须掌握 JavaScript 中的各个类型及其内在行为。几乎所有的 JavaScript 程序都会涉及某种形式的强制类型转换，处理这些情况时我们需要有充分的把握和自信。

如果要将 `42` 作为 `string` 来处理，比如获得其中第二个字符 `"2"`，就需要将它从 `number`（强制类型）转换为 `string`。

这看似简单，但是强制类型转换形式多样。有些方式简明易懂，也很安全，然而稍不留神，就会出现意想不到的结果。

强制类型转换是 JavaScript 开发人员最头疼的问题之一，它常被诟病为语言设计上的一个缺陷，太危险，应该束之高阁。

全面掌握 JavaScript 的类型之后，我们旨在改变对强制类型转换的成见，看到它的好处并且意识到它的缺点被过分夸大了。现在先让我们来深入了解一下值和类型。

1.2 内置类型

JavaScript 有七种内置类型：

- 空值（`null`）
- 未定义（`undefined`）
- 布尔值（`boolean`）
- 数字（`number`）
- 字符串（`string`）
- 对象（`object`）
- 符号（`symbol`，ES6 中新增）

除对象之外，其他统称为“基本类型”。

我们可以用 `typeof` 运算符来查看值的类型，它返回的是类型的字符串值。有意思的是，这七种类型和它们的字符串值并不一一对应：

```
typeof undefined     === "undefined"; // true
typeof true          === "boolean";   // true
typeof 42            === "number";    // true
typeof "42"          === "string";    // true
typeof { life: 42 }  === "object";    // true

// ES6中新加入的类型
typeof Symbol()      === "symbol";    // true
```

以上六种类型均有同名的字符串值与之对应。符号是 ES6 中新加入的类型，我们将在第 3 章中介绍。

你可能注意到 null 类型不在此列。它比较特殊，typeof 对它的处理有问题：

```
typeof null === "object"; // true
```

正确的返回结果应该是 "null"，但这个 bug 由来已久，在 JavaScript 中已经存在了将近二十年，也许永远也不会修复，因为这牵涉到太多的 Web 系统，“修复”它会产生更多的 bug，令许多系统无法正常工作。

我们需要使用复合条件来检测 null 值的类型：

```
var a = null;

(!a && typeof a === "object"); // true
```

null 是基本类型中唯一的一个“假值”（falsy 或者 false-like，参见第 4 章）类型，typeof 对它的返回值为 "object"。

还有一种情况：

```
typeof function a(){ /* .. */ } === "function"; // true
```

这样看来，function（函数）也是 JavaScript 的一个内置类型。然而查阅规范就会知道，它实际上是 object 的一个“子类型”。具体来说，函数是“可调用对象”，它有一个内部属性 [[Call]]，该属性使其可以被调用。

函数不仅是对象，还可以拥有属性。例如：

```
function a(b,c) {
    /* .. */
}
```

函数对象的 length 属性是其声明的参数的个数：

```
a.length; // 2
```

因为该函数声明了两个命名参数，b 和 c，所以其 length 值为 2。

再来看看数组。JavaScript 支持数组，那么它是否也是一个特殊类型？

```
typeof [1,2,3] === "object"; // true
```

不，数组也是对象。确切地说，它也是 object 的一个“子类型”（参见第 3 章），数组的元素按数字顺序来进行索引（而非普通像对象那样通过字符串键值），其 length 属性是元素的个数。

1.3 值和类型

JavaScript 中的变量是没有类型的，只有值才有。变量可以随时持有任何类型的值。

换个角度来理解就是，JavaScript 不做“类型强制”；也就是说，语言引擎不要求变量总是持有与其初始值同类型的值。一个变量可以现在被赋值为字符串类型值，随后又被赋值为数字类型值。

42 的类型为 number，并且无法更改。而 "42" 的类型为 string。数字 42 可以通过强制类型转换（coercion）为字符串 "42"（参见第 4 章）。

在对变量执行 typeof 操作时，得到的结果并不是该变量的类型，而是该变量持有的值的类型，因为 JavaScript 中的变量没有类型。

```
var a = 42;
typeof a; // "number"

a = true;
typeof a; // "boolean"
```

typeof 运算符总是会返回一个字符串：

```
typeof typeof 42; // "string"
```

typeof 42 首先返回字符串 "number"，然后 typeof "number" 返回 "string"。

1.3.1 undefined 和 undeclared

变量在未持有值的时候为 undefined。此时 typeof 返回 "undefined"：

```
var a;

typeof a; // "undefined"

var b = 42;
var c;

// later
```

```
b = c;

typeof b; // "undefined"
typeof c; // "undefined"
```

大多数开发者倾向于将 undefined 等同于 undeclared（未声明），但在 JavaScript 中它们完全是两回事。

已在作用域中声明但还没有赋值的变量，是 undefined 的。相反，还没有在作用域中声明过的变量，是 undeclared 的。

例如：

```
var a;

a; // undefined
b; // ReferenceError: b is not defined
```

浏览器对这类情况的处理很让人抓狂。上例中，“b is not defined”容易让人误以为是“b is undefined”。这里再强调一遍，“undefined”和“is not defined”是两码事。此时如果浏览器报错成“b is not found”或者“b is not declared”会更准确。

更让人抓狂的是 `typeof` 处理 undeclared 变量的方式。例如：

```
var a;

typeof a; // "undefined"

typeof b; // "undefined"
```

对于 undeclared（或者 not defined）变量，`typeof` 照样返回 `"undefined"`。请注意虽然 `b` 是一个 undeclared 变量，但 `typeof b` 并没有报错。这是因为 `typeof` 有一个特殊的安全防范机制。

此时 `typeof` 如果能返回 undeclared（而非 undefined）的话，情况会好很多。

1.3.2 typeof Undeclared

该安全防范机制对在浏览器中运行的 JavaScript 代码来说还是很有帮助的，因为多个脚本文件会在共享的全局命名空间中加载变量。

很多开发人员认为全局命名空间中不应该有变量存在，所有东西都应该被封装到模块和私有 / 独立的命名空间中。理论上这样没错，却不切实际。然而这仍不失为一个值得为之努力奋斗的目标。好在 ES6 中加入了对模块的支持，这使我们又向目标迈近了一步。

举个简单的例子，在程序中使用全局变量 DEBUG 作为“调试模式”的开关。在输出调试信息到控制台之前，我们会检查 DEBUG 变量是否已被声明。顶层的全局变量声明 var DEBUG = true 只在 debug.js 文件中才有，而该文件只在开发和测试时才被加载到浏览器，在生产环境中不予加载。

问题是如何在程序中检查全局变量 DEBUG 才不会出现 ReferenceError 错误。这时 typeof 的安全防范机制就成了我们的好帮手：

```
// 这样会抛出错误
if (DEBUG) {
    console.log( "Debugging is starting" );
}

// 这样是安全的
if (typeof DEBUG !== "undefined") {
    console.log( "Debugging is starting" );
}
```

这不仅对用户定义的变量（比如 DEBUG）有用，对内建的 API 也有帮助：

```
if (typeof atob === "undefined") {
    atob = function() { /*..*/ };
}
```

如果要为某个缺失的功能写 polyfill（即衬垫代码或者补充代码，用来补充当前运行环境中缺失的功能），一般不会用 var atob 来声明变量 atob。如果在 if 语句中使用 var atob，声明会被提升（hoisted，参见《你不知道的 JavaScript（上卷）》[1] 中的“作用域和闭包”部分）到作用域（即当前脚本或函数的作用域）的最顶层，即使 if 条件不成立也是如此（因为 atob 全局变量已经存在）。在有些浏览器中，对于一些特殊的内建全局变量（通常称为“宿主对象”，host object），这样的重复声明会报错。去掉 var 则可以防止声明被提升。

还有一种不用通过 typeof 的安全防范机制的方法，就是检查所有全局变量是否是全局对象的属性，浏览器中的全局对象是 window。所以前面的例子也可以这样来实现：

```
if (window.DEBUG) {
    // ..
}

if (!window.atob) {
    // ..
}
```

注 1：此书已由人民邮电出版社出版。——编者注

与 undeclared 变量不同，访问不存在的对象属性（甚至是在全局对象 window 上）不会产生 ReferenceError 错误。

一些开发人员不喜欢通过 window 来访问全局对象，尤其当代码需要运行在多种 JavaScript 环境中时（不仅仅是浏览器，还有服务器端，如 node.js 等），因为此时全局对象并非总是 window。

从技术角度来说，typeof 的安全防范机制对于非全局变量也很管用，虽然这种情况并不多见，也有一些开发人员不大愿意这样做。如果想让别人在他们的程序或模块中复制粘贴你的代码，就需要检查你用到的变量是否已经在宿主程序中定义过：

```
function doSomethingCool() {
    var helper =
        (typeof FeatureXYZ !== "undefined") ?
        FeatureXYZ :
        function() { /*.. default feature ..*/ };

    var val = helper();
    // ..
}
```

其他模块和程序引入 doSomethingCool() 时，doSomethingCool() 会检查 FeatureXYZ 变量是否已经在宿主程序中定义过；如果是，就用现成的，否则就自己定义：

```
// 一个立即执行函数表达式(IIFE,参见《你不知道的JavaScript(上卷)》“作用域和闭包”
// 部分的3.3.2节)
(function(){
    function FeatureXYZ() { /*.. my XYZ feature ..*/ }

    // 包含doSomethingCool(..)
    function doSomethingCool() {
        var helper =
            (typeof FeatureXYZ !== "undefined") ?
            FeatureXYZ :
            function() { /*.. default feature ..*/ };

        var val = helper();
        // ..
    }

    doSomethingCool();
})();
```

这里，FeatureXYZ 并不是一个全局变量，但我们还是可以使用 typeof 的安全防范机制来做检查，因为这里没有全局对象可用（像前面提到的 window.___）。

还有一些人喜欢使用“依赖注入”（dependency injection）设计模式，就是将依赖通过参数显式地传递到函数中，如：