如果你在Node REPL中看到三个点（...），这就意味着你需要输入更多的代码去完成当前的表达式、语句或者函数。如果你实在不明白它为什么会提示省略号，可以输入.break（包括前面的那个点）来消除当前的省略号：

```
> function () {
... }
... what?
... .break
>
```

1.2.2 编辑并运行JavaScript文件

另一种运行Node.js的方式就是：可以选择自己喜欢的文本编辑器，然后把JavaScript代码写到文件中，最后在命令行中使用node命令编译并执行代码。

为了演示这一点，把下面的代码保存为hello.js文件：

```
/**
 * Let's be sociable.
 */
console.log("Hello World!");
```

现在，你可以使用命令行执行该文件：

```
node hello.js
```

可以看到如下的输出结果：

```
Hello World!
```

因为没有使用Node shell，所以不会返回任何代码执行返回值的相关信息。

1.3 第一个Web服务器

你现在一定打算写一些更有趣的代码，并准备写一个小小的Web服务器。幸运的是，Node让这一切变得易如反掌。输入并保存代码到web.js文件中：

```
var http = require("http");

function process_request(req, res) {
    var body = 'Thanks for calling!\n';
    var content_length = body.length;
    res.writeHead(200, {
        'Content-Length': content_length,
        'Content-Type': 'text/plain'
    });
    res.end(body);
}

var s = http.createServer(process_request);
s.listen(8080);
```

要运行这个文件，需要输入：

```
node web.js
```

现在我们已经在电脑的8080端口上运行了一个Web服务器，可以使用命令行程序curl来测试它，这个命令在大部分Mac和Linux电脑上都是预置的（Windows用户请阅读下面的“Windows下如何下载Web资源”。也可以在浏览器中输入http://localhost:8080，但是，只有打开调试控制台，才会看到返回的响应代码）。

```
curl -i http://localhost:8080
```

现在，可以看到如下内容：

```
HTTP/1.1 200 OK
Content-Length: 20
Content-Type: text/plain
Date: Tue, 15 Feb 2013 03:05:08 GMT
Connection: keep-alive

Thanks for calling!
```

Windows下如何下载Web资源

默认情况下，Windows并没有提供可以从统一资源定位符（Uniform Resource Locator，URL）获取内容的命令行工具。但是，使用命令行工具下载非常有趣，因此我强烈推荐使用Windows版的cURL（从现在起，我会称之为curl）或者wget。

Curl：

可以访问http://curl.haxx.se/download.html，找到"Win32-Generic"区域，下载Windows二进制安装文件。

只需下载高亮的二进制文件中的一个，最好选择支持SSL和SSH功能的版本（如果跳转到另一个页面，点击下载"Download WITH SUPPORT SSL"），解压文件，将curl.exe放到PATH路径下或者用户目录下。要启动它，只需在命令提示符或者PowerShell中输入：

```
C:\Users\Mark\curl --help
```

Wget：

如果curl不能在你的Windows电脑上使用，那么wget是一个不错的替代品。可以从http://users.ugent.be/~bpuype/wget/上下载。

Wget的工作原理和curl差不多，但是命令行参数有所不同。要了解更多信息，可以查看帮助：

```
C:\Users\Mark\wget  --help
```

Node.js本身提供了许多强大的功能，前面程序的第一行使用了内置模块之一——http模块，它允许程序创建Web服务器。可以使用require函数引用该模块，从而拥有http的引用变量。

createServer函数只会接受一个函数参数，它会在用户连接到服务器时被调用。前面写的process_request函数会被作为参数传递进去，该函数会被赋予一个请求对象（即ServerRequest[1]实例）和一个响应对象（即ServerReponse实例）。服务器创建之后，会指定在某个特定的端口上监听传入的请求——这里，启动该程序时，使用了8080端口。

前面将-i参数传给curl命令，会将请求内容和响应的头信息一起打印出来。这样便于我们更加理解Node的工作原理。

从返回的信息中可以看到，ServerResponse#writeHead函数返回的200（OK）响应就在HTTP响应头里面，同时还包含了内容长度（Content-Length）和类型（Content-Type）。默认情况下，Node.js会将服务器的HTTP连接设置为keep-alive，表明可以在同一个连接中发送多个请求，当然，本书的大部分示例中都用不到该功能。

要停止运行的服务器，只需要简单地按下Ctrl+C即可。它会自动清理系统资源并停止服务。

[1]最新版本的API　doc中已替换成IncomingMessage1类。——译者注

1.4 调试Node.js程序

现在，重写前面的Web服务器，但是这次可能会由于粗心而引入一个小错误——body.length的拼写错误，将代码保存到debugging.js文件中，如下所示：

```
var http = require("http");
function process_request(req, res) {
    var body = 'Thanks for calling!\n';
    var content_length = body.lenggth;
    res.writeHead(200, {
        'Content-Length': content_length,
        'Content-Type': 'text/plain'
    });
    res.end(body);
}

var s = http.createServer(process_request);
s.listen(8080);
```

再次运行该程序：

```
node debugging.js
```

但是，当尝试连接到http://localhost:8080的时候，可能会得到：

```
client:~ marcw$ curl -i localhost:8080
HTTP/1.1 200 OK
Content-Length: undefined
Content-Type: text/plain
Date: Tue, 30 Oct 2012 04:42:44 GMT
Connection: keep-alive
```

上面并没有得到"Thanks for calling!"这样的信息，而且响应头Content-Length也不是预期的结果。

像这样的小程序中，这些错误会很容易被找到，但在一些大型程

序中，将很难定位具体的错误信息。为了解决这个问题，Node.js内置了一个调试器。如果想使用，只需要在启动的时候将debug参数添加到程序名称的前面即可：

```
node debug debugging.js
```

可以得到如下所示的信息：

```
client:Chapter01 marcw$ node debug debugging.js
< debugger listening on port 5858
connecting... ok
break in debugging.js:1
  1 var http = require("http");
  2
  3 function process_request(req, res) {
debug>
```

Node调试器中有一些关键的命令可供使用：

- cont——继续执行。
- next——跳到下一个语句。
- step——进入当前执行函数中的语句（如果是函数的话；否则，跳过）。
- out——跳出当前执行函数。
- backtrace——显示当前调用执行帧或调用栈。
- repl——启动Node REPL，允许查看变量值和执行代码。
- watch(expr)——向观察列表中添加表达式，这样在调试器中进入函数或者移动时会显示出来。
- list(n)——列出调试器中当前停止行的前面和后面的n行代码。

现在，假设程序中的Content-Length可能出错，则需要在行var content_length=body.lenggth;上添加断点，即第5行：

```
debug> setBreakpoint(5)
  1 var http = require("http");
  2
  3 function process_request(req, res) {
  4     var body = 'Thanks for calling!\n';
* 5     var content_length = body.lenggth;
  6     res.writeHead(200, {
debug>
```

现在第5行已经多了一个星号（*），表明存在断点。当启动调试器时，程序会在第一行停下。这时，可以使用cont命令恢复执行：

```
debug> cont
debug>
```

打开另外一个终端窗口或者命令行提示符，输入：

```
curl –i http://localhost:8080
```

我们注意到发生了两件事：

1）curl命令并没有立即返回信息。

2）在node debug进程中，可以看到：

```
break in debugging.js:5
  3 function process_request(req, res) {
  4     var body = 'Thanks for calling!\n';
* 5     var content_length = body.lenggth;
  6     res.writeHead(200, {
  7         'Content-Length': content_length,
```

我们可以跳过这一行：

```
debug> next
break in debugging.js:7
* 5     var content_length = body.lenggth;
  6     res.writeHead(200, {
  7         'Content-Length': content_length,
  8         'Content-Type': 'text/plain'
  9     });
```

现在，开启Node REPL，这样就可以检查一些变量值了：

```
debug> repl
Press Ctrl + C to leave debug repl
>
```

让我们分别看一下变量body和content_length的值：

```
> body
'Thanks for calling!\n'
> content_length
>
```

变量body的值与预期的一样。而变量content_length的值应该是20，但是却没有显示出来。由此可以得知给content_length赋值时出错，问题找到了！

最后，可以通过Ctrl+D关闭系统结束调试，或者输入cont继续运行服务器。在REPL中输入cont不会生效，会显示如下错误信息：'ReferenceError:cont is not defined'。因此，首先需要按下Ctrl+C退出REPL，然后使用cont命令。

尽管对调试器的介绍有些简短，但是绝对值得一试，因为它真的非常强大和实用。另外，还有一些Node.js社区成员编写的基于浏览器的调试器，其中最有前景的当属node-inspector。可以尽情尝试，看看它们是否对你的开发有帮助。

必要时，还可以使用最为简单的console.log(variable_name);，将其添加到代码中，然后可以在终端窗口中打印结果。它是获取信息最快最便捷的方法，可以有效定位和追溯错误或问题。

1.5 保持最新及获取帮助

正如前文所提到的那样，使用Node.js的挑战之一就是它不断地更新变化。尽管越来越多的API被标记为稳定或者锁定，但每一个新版本都会有些新变化，而几乎每周都会发布新版本。

通过以下方式，可以保持最新并且不会错过任何重要的变更和消息：

- 加入Node.js的邮件列表http://groups.google.com/group/nodejs。很多Node核心开发者都在该邮件列表中，并会在每一个新版本发布或变更的时候发布消息。

- 如果你拥有推特（Twitter）账号，可以关注@nodejs；当有新版本或者其他重要消息发布的时候，你会收到相应的推特消息。

- 经常访问nodejs.org，以保证了解最新的信息。

要想获取帮助，nodejs谷歌论坛（Google Groups）和node官网一样举足轻重。同样，StackOverflow.com也是一个非常活跃的社区，可以帮助解决Node相关的问题，你可以在那里找到许多非常棒的答案。

但是，我还发现，很多问题只需要通过简单的谷歌搜索就可以找到最好的答案。许多开发者经常会遇到类似的问题，并撰写一些博客和消息。因此，我一般都能通过简单的搜索找到理想的答案。

1.6 小结

至此，你已经成功地将Node.js安装到电脑上，并验证过能够正常运行，甚至已经运行并调试过一些问题了。那么，现在是时候进一步了解JavaScript这门语言了。

第2章　进一步了解JavaScript

如果你正在阅读本书，那么很可能你以前使用过JavaScript。也许你已经使用HTML、CSS及JavaScript编写过一个Web应用程序，通过直接操作浏览器文档对象模型（Document Object Model，DOM）或者使用框架（例如，使用jQuery、Prototype等来屏蔽一些混乱的细节）使客户端更加动态、更具交互性。可能你已经发现使用JavaScript工作是一件令人沮丧的事情，需要花费大量的时间处理不同浏览器的兼容性。而且很可能，你未曾从最基础的语言特性方面研究过JavaScript语言，那么，阅读本书将会让你受益匪浅。

好消息是现代Web浏览器正在逐步对JavaScript语言做一些必要的清理。此外，所有现代浏览器实现所遵守的规范——ECMAScript也一直在不断发展。Chrome V8 JavaScript引擎本身也在不断地改进和清理一些JavaScript语言糟粕，并添加许多被忽略的重要特性。

因此，即使你过去使用过JavaScript，本章仍然值得一读，你能够了解到一些被忽略的细节、一些新特性或者由V8和Node.js带来的改变。虽然本章的大部分内容适用于标准的JavaScript，但其中还是会经常展示一些由Google V8带来的新特性。对于非标准特性，我会使用（V8 JS）来标注。

2.1 数据类型

本节将会回顾一下JavaScript，看一下这门语言所提供的数据类型。对于本节中的大部分内容，我都会使用Node.js的Read-Eval-Print-Loop（REPL）来演示代码是如何工作的。代码中，使用粗体字来标识需要读者输入到解释器中的代码。

```
client:LearningNode marcw$ node
>
```

2.1.1 类型基础

Node.js的核心类型有：number（数字）、boolean（布尔值）、string（字符串）以及object（对象）。另外两种类型——函数（function）和数组（array）实际上是object的特殊形式。因为它们在语言以及运行时层面有一些额外的特性，因此将object、function（函数）以及array（数组）归类为复杂数据类型。null和undefined也是object的特殊形式，在JavaScript语言中有特殊作用。

undefined值代表还没有赋值或者不存在：

```
> var x;
undefined
> x = {};
{}
> x.not_valid;
undefined
>
```

null的另外一个准确的意思是“没有值”：

```
> var y;
undefined
> y
undefined
> y = null;
null
>
```

在JavaScript中，可以通过typeof操作符查看任何数据的类型：

```
> typeof 10
'number'
> typeof "hello";
'string'
> typeof function () { var x = 20; }
'function'
>
```

2.1.2 常量

虽然Node.js理论上支持const关键字，const关键字扩展在许多现代的JavaScript实现中也都被实现了，然而const关键字仍没有被广泛使用。对于常量，标准实践仍然是使用大写字母和变量声明：

```
> var SECONDS_PER_DAY = 86400;
undefined
> console.log(SECONDS_PER_DAY);
86400
undefined
>
```

2.1.3 number类型

JavaScript中所有数字都采用IEEE 754标准定义的64位双精度浮点数表示。所有的正负整数都可以使用2^{53}位准确表示，JavaScript中的数字类型和其他语言的整数数据类型非常相似：

```
> 1024 * 1024
1048576
> 1048576
1048576
> 32437893250 + 3824598359235235
3824630797128485
> -38423538295 + 35892583295
-2530955000
>
```

然而，使用数字类型最棘手的部分是，许多数字的真实值实际上是实际数值的一个近似值。例如：

```
> 0.1 + 0.2
0.30000000000000004
>
```

当对浮点数执行算术运算时，仅仅操作任意的实际数字并不一定能够得到准确值：

```
> 1 - 0.3 + 0.1 == 0.8
false
>
```

对于这些情况，并不需要检查某个值是否在某个近似范围内，它的大小是由与之比较的数值的大小定义的（搜索stackoverflow.com网站，可以找到一些与比较浮点数相关的策略和思路的文章及问题）。

在某些情况下，JavaScript需要严格意义上的64位的整数值，而不能出现近似值这样的错误，我们可以使用字符串（string）类型，手动操纵这些数字，或者使用一些能够进行大整数运算的模块。

在某个数被0整除这点上，JavaScript和其他语言有很大的不同，它只会简单地返回一个正无穷大（Infinity）或者负无穷大（-Infinity），而不会抛出一个运行时异常：

```
> 5 / 0
Infinity
> -5 / 0
-Infinity
>
```

正无穷大和负无穷大在JavaScript里都是合法的值，并且可以使用它们进行比较。

```
> var x = 10, y = 0;
undefined
> x / y == Infinity
true
>
```

我们可以使用parseInt和parseFloat函数将字符串转换为数字：

```
> parseInt("32523523626263");
32523523626263
> parseFloat("82959.248945895");

82959.248945895
> parseInt("234.43634");
234
> parseFloat("10");
10
>
```

如果这些函数无法解析传入的参数，将会返回一个特殊值NaN（not-a-number）：

```
> parseInt("cat");
NaN
> parseFloat("Wankel-Rotary engine");
NaN
>
```

为了测试NaN，我们需要使用isNaN函数：

```
> isNaN(parseInt("cat"));
true
>
```

最后，为了测试一个给定的值是否是一个合法的有限数（不是Infinity、-Infinity或者NaN），我们需要使用isFinite函数：

```
> isFinite(10/5);
true
> isFinite(10/0);
false
> isFinite(parseFloat("banana"));
false
>
```

2.1.4 boolean类型

JavaScript中的布尔（boolean）数据类型不仅简单而且容易使用。布尔值可以是true或者false，虽然技术上可以使用Boolean函数将其他值转换为布尔值，但实际上却很少需要使用这个函数，因为JavaScript语言会在需要时自动将任何值转换为布尔值，转换规则如下：

1）false、0、空字符串（""）、NaN、null以及undefined都等价于false。

2）其他值都等价于true。

2.1.5 string类型

JavaScript中的字符串（string）是一组Unicode字符（内部以16位UCS-2格式实现）组成的序列，可以表示世界上绝大部分字符，包括大部分亚洲语言中使用的字符。JavaScript语言没有单独的字符（char）或者字符数据类型，可以使用只有一个字符的字符串来表示字符。对于大部分使用Node.js编写的网络应用，需要使用UTF-8格式对外通信，Node会自动处理转换细节。而如果是操纵二

进制数据，那么处理字符串和字符集相关的经验会非常重要。

字符串可以使用单引号或者双引号封装。单引号和双引号在功能上是等价的，可以选择任意一个使用。如果想要在使用单引号的字符串中包含单引号，可以使用\'；同理，如果在使用双引号的字符串中包含双引号，可以使用\"：

```
> 'Marc\'s hat is new.'
'Marc\'s hat is new.'
> "\"Hey, nice hat!\", she said."
'"Hey, nice hat!", she said.'
>
```

要想获得一个JavaScript字符串的长度，只需使用length属性：

```
> var x = "cat";
undefined
> x.length;
3
> "cat".length;
3
> x = null;
null
```

在JavaScript中尝试获取值为null或undefined的字符串的长度时，将会抛出错误：

```
> x.length;
TypeError: Cannot read property 'length' of null
    at repl:1:2
    at REPLServer.self.eval (repl.js:109:21)
    at rli.on.self.bufferedCmd (repl.js:258:20)
    at REPLServer.self.eval (repl.js:116:5)
    at Interface.<anonymous> (repl.js:248:12)
    at Interface.EventEmitter.emit (events.js:96:17)
    at Interface._onLine (readline.js:200:10)
    at Interface._line (readline.js:518:8)
    at Interface._ttyWrite (readline.js:736:14)
    at ReadStream.onkeypress (readline.js:97:10)
```

要想将两个字符串组合在一起，可以使用+操作符：

```
> "cats" + " go " + "meow";
'cats go meow'
>
```

如果将其他类型的数据混入到字符串中，JavaScript将尽可能将其他数据转换成字符串：

```
> var distance = 25;
undefined
> "I ran " + distance + " kilometres today";
'I ran 25 kilometres today'
>
```

注意，当混入的表达式过多时，则可能出现许多有趣的结果：

```
> 5 + 3 + " is my favourite number";
'8 is my favourite number'
>
```

如果想将“53”作为我最喜欢的数字，可以在表达式前加上一个空字符串，用来提前强制转换数据类型。

```
> "" + 5 + 3 + " is my favourite number";
'53 is my favourite number'
>
```

许多人担心在处理字符串时使用连接运算符+会导致严重的性能问题。好消息是几乎所有现代浏览器的JavaScript实现——包括Node.js使用的ChromeV8引擎，已经对该问题进行了深度优化，因此现在的运行性能非常好。

字符串函数

JavaScript中为字符串提供了许多有趣的函数。使用indexOf函数可以在一个字符串中搜索另外一个字符串。

```
> "Wishy washy winter".indexOf("wash");
6
>
```

从一个字符串中截取一个子串，可以使用substr或splice函数（前者会接受一个开始索引和一个需要截取的字符串长度；而后者则会接受一个开始索引和一个结束索引）：

```
> "No, they're saying Booo-urns.".substr(19, 3);
'Boo'
> "No, they're saying Booo-urns.".slice(19, 22);
'Boo'
>
```

如果字符串中有某个分隔符，可以使用split函数将字符串分割成子字符串并返回一个数组：

```
> "a|b|c|d|e|f|g|h".split("|");
[ 'a',
  'b',
  'c',
  'd',
  'e',
  'f',
  'g',
  'h' ]
>
```

最后，可以使用trim函数（V8 JS）清除字符串前后的空白字符：

```
> '       cat    \n\n\n     '. trim();
'cat'
>
```

正则表达式

JavaScript支持功能强大的正则表达式，正则表达式的具体细节不在本书讨论范围之内，但我会简单明了地展示如何以及在什么时候使用正则表达式。有几个字符串函数可以接收正则表达式作为参数并执行。正则表达式不仅可以使用字面量格式（literal format）（通过将正则表达式放入两个斜杠字符[/]之间表示），也可以通过调用RegExp对象的构造器来表示：

```
/[aA]{2,}/
new RegExp("[Aa]{2,}")
```

以上两个都是正则表达式，用来表示一组两个或两个以上a字符的序列（大写或者小写）。

为了将字符串对象中两个或两个以上的a字符序列替换成b字母，我们可以使用replace函数，以下两种写法都是可行的：

```
> "aaoo".replace(new RegExp("[Aa]{2,}"), "b");
'boo'
> "aaoo".replace(/[Aa]{2,}/, "b");
'boo'
>
```

与indexOf函数类似，search函数接收一个正则表达式参数，并返回第一个匹配此正则表达式的子字符串的位置索引，如果匹配不存在则返回-1：

```
> "aaoo".search(/[Aa]{2,}/);
0
> "aoo".search(/[Aa]{2,}/);
-1
>
```

2.1.6 object类型

对象（object）是JavaScript语言的核心之一，我们总会使用到它。对象是一种相当动态和灵活的数据类型，可以轻松地为其新增或删除属性。我们可以使用以下两种方式创建对象，而后者就是所谓的对象字面量语法（object literal syntax），它是目前最推荐的写法。

```
> var o1 = new Object();
undefined
> var o2 = {};
undefined
>
```

我们还可以使用对象字面量语法指定对象的内容，在初始化时，可以指定对象成员的名字以及对应的值：

```
var user = {
    first_name: "marc",
    last_name: "wandschneider",
    age: Infinity,
    citizenship: "man of the world"
};
```

关于JSON

本书中最常用的东西之一就是JSON（事实上，包括网络和Web应用也是），即JavaScript对象表示法（JavaScript Object Notation）。这种数据交换格式充分发挥了基于文本的数据格式的灵活性，却没有像XML这样的其他格式所带来的麻烦（公平地讲，相较于后者，JSON缺少一些格式验证功能，但它仍然是最好使用的格式）。

JSON和对象字面量表示法非常相似，但是二者之间有两个关键的区别：对象字面量表示法使用单引号或双引号封装属性名，甚至可以不使用任何引号，而在JSON中却是强制使用引号的。另外，JSON中所有字符串都需要包含在双引号中：

```
// valid object literal notation, INVALID JSON:
var obj = {
    // JSON strings are supposed to use ", not '
    "first_name": 'Marc',
    // Must wrap property names for JSON
    last_name: "Wandschneider"
}

// valid JSON and object literal notation:
var obj = {
    "first_name": "Marc",
    "last_name": "Wandschneider"
}
```

实际上大部分JSON库兼容单引号字符串，但是为了提高兼容性，无论是编写还是生成JSON数据，最好还是使用双引号。

通常我们可以使用V8的JSON.parse和JSON.stringify函数来生成JSON数据。前者接收一个JSON字符串作为参数，并将其转换成一个对象（如果失败，则抛出一个错误），而后者接收一个对象作为参数，并返回一个JSON字符串表示。

当在代码中编写对象时，我们经常使用对象字面量表示法，但同时本书中也会编写大量的JSON。因此，了解二者之间的区别非常重要。这里我需要特别指出，无论何时，JSON都是必要的。

我们可以使用以下任意一种方法来给自己的对象添加新属性：

```
> user.hair_colour = "brown";
'brown'
> user["hair_colour"] = "brown";
'brown'
> var attribute = 'hair_colour';
undefined
> user[attribute] = "brown";
'brown'
> user
{ first_name: 'marc',
  last_name: 'wandschneider',
  age: Infinity,
  citizenship: 'man of the world',
  hair_colour: 'brown' }
>
```

如果尝试访问一个不存在的属性，并不会报错，而是会得到undefined这样的结果。

```
> user.car_make
undefined
>
```

当我们需要删除对象的某个属性时，可以使用delete关键字：

```
> delete user.hair_colour;
true
> user
{ first_name: 'marc',
  last_name: 'wandschneider',
  age: Infinity,
  citizenship: 'man of the world' }
>
```

JavaScript对象非常灵活，这使得它与其他语言的关联数组、哈希表、字典极其相似，但还是有一点不同：在JavaScript中获取一个关联数组对象的大小有些棘手。对象没有size或者length等属性或者方法。而为了得到对象的大小，可以使用如下写法（V8 JS）：

```
> Object.keys(user).length;
4
```

请注意，这里使用了非标准的JavaScript扩展Object.keys，但V8和大多数浏览器（除了IE）都已经支持了这个方法。

2.1.7 array类型

JavaScript中的数组（array）类型实际上是JavaScript对象的一个特殊形式，它拥有一系列额外特性，这使得数组非常实用和强大。我们可以使用传统的表示法或者数组字面量语法（array literal syntax）来创建数组：

```
> var arr1 = new Array();
undefined
> arr1
[]
> var arr2 = [];
undefined
> arr2
[]
>
```

和对象一样，我倾向于使用字面量语法来创建数组，而很少使用传统表示法创建数组。

如果我们对数组使用typeof运算符，会得到一个令人惊讶的结果：

```
> typeof arr2
'object'
>
```

因为数组实际上就是对象，所以typeof运算符会返回"object"，而这不是我们想要的结果。幸运的是，V8有一个语言扩展，可以确定是否为一个数组：Array.isArray函数（V8 JS）。

```
> Array.isArray(arr2);
true
> Array.isArray({});
false
>
```

JavaScript中数组类型的一个关键特性是length属性，使用方法如下：

```
> arr2.length
0
> var arr3 = [ 'cat', 'rat', 'bat' ];
undefined
> arr3.length;
3
>
```

默认情况下，JavaScript数组是通过数字来进行索引的：

```
// this:
for (var i = 0; i < arr3.length; i++) {
    console.log(arr3[i]);
}
// will print out this:
cat
rat
bat
```

我们可以通过以下两种方式在数组的末尾添加新项：

```
> arr3.push("mat");
4
> arr3
[ 'cat', 'rat', 'bat', 'mat' ]
> arr3[arr3.length] = "fat";
'fat'
> arr3
[ 'cat', 'rat', 'bat', 'mat', 'fat' ]
>
```

可以通过指定特定的元素索引插入新元素。如果该元素的索引超