TURING 图灵程序设计丛书

[PACKT] PUBLISHING

[美] David Herron 著

鄢学鹃 吴天豪 廖健 译

Node Web Development

Node Web开发

人民邮电出版社
POSTS & TELECOM PRESS

David Herron

软件开发人员和软件质量工程师，在硅谷从业20多年，目前在雅虎担任质量工程小组的架构师，管控公司基于Node开发的Web应用平台的质量。

他曾为Sun公司主管工程师，并作为Java SE质量工程小组的架构师负责开发自动化测试工具（包括现在广泛用于GUI自动测试软件的AWT Robot类），期间参与了OpenJDK和JDK-Distros项目的启动，举办了世界性的Mustang Regressions大赛，让Java开发者社区寻找Java 1.6的bug。

任职Sun公司之前，他曾为VXtreme公司开发视频流处理工具（Windows Media Player的前身），在Wollongong集团从事电子邮件客户端和服务器软件的开发，加入了互联网工程任务组，负责改进与电子邮件相关的协议。

鄢学鹃

阿里云云手机开发者运营负责人，曾在网易做过UI设计师，在雅虎中国领导过前端团队，在口碑网领导过UED团队，还担任D2前端技术论坛顾问。对Web标准、前端开发模式、性能优化和自动化有较深入的研究。目前专注于从Mobile到PC领域的设计、技术和业务间的结合，常用ID：秦歌、三七。其译著有《JavaScript语言精粹》和《高性能网站建设进阶指南：Web开发者性能优化最佳实践》，个人博客是dancewithnet.com，Twitter账号是@kavenyan。

吴天豪

阿里云计算前端开发工程师，w3ctech杭州站负责人，w3ctech.com内容贡献者，负责过口碑网产品线的开发、基于移动浏览器的Web App开发，致力于构建快速、高效、可访问性高的Web应用。

廖健

阿里云资深前端开发工程师，有多年Flash平台开发经验，喜欢奔放的脚本语言，曾在D2论坛和HTML5研究小组线下沙龙做过技术分享，目前主要研究和实践Web技术在移动平台上的应用。

图灵程序设计丛书

[美] David Herron 著　　鄢学鹃 吴天豪 廖健 译

Node Web Development

Node Web开发

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

Node Web开发 / （美）赫伦（Herron,D.）著 ; 鄢学鹃, 吴天豪, 廖健译. -- 北京 : 人民邮电出版社, 2012.4
（图灵程序设计丛书）
书名原文: Node Web Development
ISBN 978-7-115-27832-6

Ⅰ. ①N… Ⅱ. ①赫… ②鄢… ③吴… ④廖… Ⅲ. ①网页制作工具－程序设计 Ⅳ. ①TP393.092

中国版本图书馆CIP数据核字(2012)第056420号

内 容 提 要

Node 是一个服务器端的 JavaScript 解释器，是构建快速响应、高度可扩展网络应用的轻量高效的平台。Node 使用事件驱动和非阻塞的 I/O 模型，非常适合数据密集、对实时响应要求高的分布式应用。微软、eBay、LinkedIn、雅虎等世界知名公司及网站均有使用 Node 的成功案例。

本书是基于 Node 开发 Web 应用的实用指南，全书共分 6 章，通过示例详尽介绍了 Node 的背景、原理及应用方法。全书内容涉及 Node 简介、Node 安装、Node 模块、实现不同版本的简单应用、实现简单的 Web 服务器和 EventEmitter，以及数据存储和检索。另外，本书涵盖了 Node 服务器端开发的主要挑战及应对方案。

本书适合 Web 前、后端开发人员学习参考。

图灵程序设计丛书
Node Web开发

◆ 著 [美] David Herron
译 鄢学鹃 吴天豪 廖 健
责任编辑 毛倩倩

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷

◆ 开本：800×1000 1/16
印张：7.25
字数：176千字 2012年 4 月第 1 版
印数：1－5 000册 2012年 4 月北京第 1 次印刷
著作权合同登记号 图字：01-2012-1950 号
ISBN 978-7-115-27832-6

定价：35.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

欢迎光临Node（也叫Node.js）开发的世界。Node是一种新兴的软件开发平台，它将JavaScript从Web浏览器移植到常规的服务器端。Node运行在Chrome的高速V8引擎上，并附带了一个快速、健壮的异步网络I/O组件库。Node主要用于构建高性能、高可扩展的服务器和客户端应用，以实现真正“实时的Web应用”。

在经过数年尝试用Ruby和其他语言实现Web服务器组件之后，Ryan Dahl在2009年开发了Node平台。这个探索使他从使用传统的、基于线程的并发模型转向使用事件驱动的异步系统，因为后者更简单（多线程系统以难于开发著称），系统开销更低（与对每个连接维护一个线程相比），因而能提高相应的速度。Node旨在提供一个“创建可扩展网络服务器的简单方式”。这个设计受到了Event Machine（Ruby）和Twisted框架（Python）的影响，并和它们有些类似。

本书致力于讲述如何用Node构建Web应用。我们会在书中介绍快速学习Node时一些必需的重要概念。本书会教你编写真正的应用，剖析其工作原理，并讨论如何在程序中应用这些理念。我们需要安装Node和npm，学习安装和开发npm包及Node模块。此外，我们还会开发一些应用，度量长时间运行的计算在Node的事件循环中的响应能力，介绍将高负载的工作分派到多个服务器的方法，并介绍Express框架。

本书内容

第1章“Node入门”，介绍了Node平台。这一章讲述了Node的用途、技术构架上的选择、Node的历史和服务器端JavaScript的历史，然后介绍为什么JavaScript仍将受困于浏览器。

第2章“安装并配置Node”，介绍如何配置Node开发环境，包括多种从源码编译和安装的场景，还会简单介绍在开发环境中如何部署Node。

第3章“Node模块”，解释了作为开发Node应用基本单位的模块。我们会全面介绍并开发Node模块。然后进一步介绍Node包管理器npm，给出一些使用npm管理已安装包的例子，还将涉及开发npm包并将其发布出来供他人使用。

第4章“几种典型的简单应用”，在读者已经有一些Node基础知识后，开始探索Node应用的开发。我们会分别使用Node、Connect中间件框架和Express应用框架开发一个简单的应用。虽然应用比较简单，但是我们可以通过其开发探索Node的事件循环，处理长时间的运算，了解异步和同步算法以及如何将繁重的计算交给后台服务器。

第5章“简单的Web服务器、EventEmitter和HTTP客户端”，介绍了Node里的HTTP客户端和服务器对象。我们会在开发HTTP服务器和客户端应用的同时全面深入介绍HTTP会话。

第6章“存取数据”，探讨大部分应用都需要的长期可靠的数据存储机制。我们会用SQL和MongoDB数据库引擎实现一个应用。在此期间，我们将用Express框架实现用户验证，更好地展示出错页面。

阅读要求

目前，我们一般会采用源码的方式安装Node，这种方式可以很好地用在类Unix和符合POSIX标准的系统上。当然，在接触Node之前，谦逊的心态是必需的，但最为重要的事情还是让大脑供血充足。

从源码安装的方式需要一个类Unix或类POSIX系统（比如Linux、Mac、FreeBSD、OpenSolaris等）、新的C/C++编译器、OpenSSL库和Python 2.4或更新版本。

Node程序可以用任何文本编辑器来写，不过一个能处理JavaScript、HTML、CSS等的文本编辑器会更有帮助。

尽管本书介绍的是Web应用开发，但你并不需要拥有一个Web服务器。Node有自己的Web服务器套件。

读者对象

本书写给所有想在一个新的软件平台上开拓新编程模式的软件工程师。

服务器端程序员或许能看到一些新奇的概念，对Web应用开发产生新的理解。JavaScript是一门强大的语言，Node的异步特性发挥了JavaScript的优势。

浏览器端JavaScript“攻城师”或许会觉得在Node中使用JavaScript和编写与DOM操作无关的JavaScript代码很有趣。（Node平台上没有浏览器，所以也没有DOM，除非你安装JSDom。）

虽然本书各章内容由浅入深，循序渐进，但到底如何阅读本书悉听尊便。

本书需要读者知道如何编写软件，并且对JavaScript等编程语言有所了解。

排版约定

在本书中，读者会发现不同的文本样式。下面是这些样式的示例和说明。

正文中的代码使用特殊字体：“`http`对象封装HTTP协议，它的`http.createServer`方法会创建一个完整的Web服务器，而`.listen`方法用于监听特定的端口。”

代码块是这样的：

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
```

```
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
```

代码块中会加粗突出显示代码，这表示需要读者格外注意：

```
var util = require('util');
var A = "a different value A";
var B = "a different value B";
var m1 = require('./module1');
util.log('A='+A+' B='+B+' values='+util.inspect(m1.values()));
```

命令行的输入输出是这样的：

```
$ sudo /usr/sbin/update-rc.d node defaults
```

新术语及**重要词汇**都会加粗显示。你将在屏幕上看到的文字，比如菜单或对话框中的文字，会这样在正文中提到：“一个真正安全的系统至少会有用户名和密码输入框。不过，我们这里就直接让用户单击Login按钮了。”

读者反馈

我们始终欢迎来自读者的反馈意见。我们想知道读者对本书的看法，读者喜欢哪些内容或不喜欢哪些内容。读者真正深有感触的反馈，对于我们开发图书产品至关重要。

一般的反馈可以发邮件到feedback@packtpub.com，但请在邮件标题中注明相关书名。

如果有关于新书的建议，你可以登录www.packtpub.com，填写SUGGEST A TITLE表单或者向suggest@packtpub.com发送邮件。

如果你在某个领域积累了丰富的经验，想写一本书，或者愿意与人合著或审校某本书，请阅读www.packtpub.com/authors上的作者指南。

读者服务

现在你已是Packt引以为荣的读者了，因此我们特别要交待几件事，以保障你作为读者的最大权益。

下载示例代码

在www.packtpub.com通过自己的账号购买图书的读者，可以下载所有已购买图书的代码[①]。如果这本书是你在其他地方购买的，访问www.packtpub.com/support并注册，我们将通过电子邮件将相关文件发送给你。

① 本书的代码文件也可在图灵社区（ituring. com. cn）本书网页免费注册下载。——编者注

勘误

虽然我们会全力确保本书内容的准确性，但错误仍在所难免。如果你发现了本书中的错误（包括文字和代码错误），而且愿意向我们提交这些错误，我们会十分感激。这样一来，不仅可以减少其他读者的疑虑，也有助于本书后续版本的改进。要提交错误，请访问www. packtpub.com/support，选择相关图书，单击errata submission form链接，然后输入勘误信息。经过验证后，你提交的勘误信息就会添加到已有的勘误列表中。要查看已有的勘误信息，请访问www.packtpub.com/support并选择相关图书。

反盗版声明

网上各种形式的盗版是一直存在的问题。Packt非常重视版权和许可证的保护。如果你在网上遇到以任何形式非法复制的我方作品，请尽快告知我们相关的地址或网站名称，以便我们采取补救措施。

请把邮件发送到copyright@packtpub.com，并在邮件里注明涉嫌侵权资料的链接。

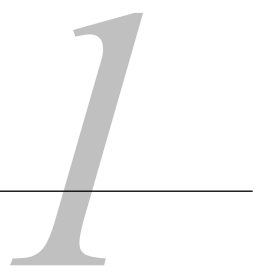
感谢你帮助我们保护作者和我们为你带来有价值内容的能力。

疑难解答

如果对本书的某些方面有疑问，请将电子邮件发送到questions@packtpub.com，我们会尽力解决。

目　录

第 1 章 Node入门



无论开发Web应用、应用服务器、任何类型的网络服务器或客户端还是通用编程，Node都是一个令人兴奋的新平台。它把异步I/O和服务器端JavaScript巧妙地组合在一起，聪明地运用了强大的JavaScript匿名函数和单线程执行的事件驱动架构，是专为网络应用的极高扩展性而设计的。

Node模型与大规模使用线程的普通应用服务器平台截然不同。由于使用事件驱动架构，内存占用量低，吞吐量高，且编程模型更简单。Node平台正处于高速成长阶段，很多人使用Node就是为了替代传统方法（使用Apache、PHP、Python等）。

Node的核心是一个独立的JavaScript虚拟机，通过扩展之后可适用于通用编程，并明确地专注于应用服务器开发。Node平台和开发Web应用的常用编程语言（PHP、Python、Ruby、Java等）不是一类东西，与那些向客户端提供HTTP协议的容器（Apache、Tomcat、Glassfish等）也没有直接可比性。而且，很多人认为它非常有可能取代传统的Web应用开发相关技术。

Node实现以非阻塞的I/O事件循环机制和文件与网络I/O库为中心，一切都以（来自Chrome浏览器的）V8 JavaScript引擎为基础。这个I/O库足以实现采用任何TCP或UDP协议的、任意类型的服务器，无论是DNS服务器，还是采用HTTP、IRC、FTP服务器。虽然它支持针对任何网络协议开发服务器或客户端，但最常用于构建普通网站，这种情况下可以取代像Apache/PHP或Rails这样的框架。

本书将向你介绍Node。我们假定你已知道如何编写软件、熟悉JavaScript，且对如何用其他语言开发Web应用有所了解。我们将以开发实际应用为例，因为阅读实际的代码是最好的学习方式。

1.1 Node 能做什么

Node是脱离浏览器编写JavaScript应用的平台。这里的JavaScript并非我们在浏览器中熟悉的JavaScript，Node没有内置DOM，也没有任何浏览器的功能。但它使用JavaScript语言和异步I/O框架，是一个强大的应用开发平台。

Node无法用于实现桌面GUI应用。目前Node既没有内置相当于Swing（或SWT）的功能组

件[①]，也没有Node扩展GUI工具包，而且不能内嵌到Web浏览器中。如果有可用的GUI工具包，Node就能用于构建桌面应用。一些项目已经开始为Node创建GTK绑定[②]，它能提供一个跨平台的GUI工具包。Node使用的V8引擎带有一个扩展API，允许编入C/C++代码以扩展JavaScript或集成原生代码库。

除了能原生执行JavaScript外，Node捆绑的模块还能提供如下功能：

- 命令行工具（shell脚本风格）；
- 交互式TTY风格编程（REPL[③]）；
- 出色的进程控制函数能监控子进程；
- 用Buffer对象处理二进制数据；
- 使用全面事件驱动回调函数的TCP或UDP套接字；
- DNS查找；
- 基于TCP库的HTTP和HTTPS客户端/服务器；
- 文件系统的存取；
- 内置了基于断言的单元测试能力。

Node的网络层是底层，但易于使用。例如，HTTP模块可以让你仅用几行代码编写出一个HTTP服务器（客户端），但这层让程序员非常贴近协议的要求，且让你精确控制将返回的请求响应的HTTP头。PHP程序员通常不需要关心HTTP头，但Node程序员需要。

换句话说，用Node编写HTTP服务器非常容易，但通常Web应用开发者不需要在这种细节层面上费神。例如，由于Apache已经存在，PHP程序员就不需要实现其HTTP服务器部分了。Node社区已经开发出许多类似于Connect的Web应用框架，让开发者能够快速配置可提供所有基础内容（会话、cookie、静态文件和日志等）的HTTP服务器，从而把时间和精力都放在业务逻辑上。

服务器端 JavaScript

有点不耐烦了？你正一边摇头一边抱怨：“在服务器上用一门浏览器语言做什么？”实际上，JavaScript在浏览器之外有着悠久且大量鲜为人知的历史。JavaScript就像其他语言一样是一门编程语言，而你的问题可能应该这样：“为什么JavaScript会一直被困在浏览器中?”

---

① Swing是一个为Java设计的工具包，是Java基础类的一部分，具体见http://zh.wikipedia.org/wiki/Swing_(Java)。SWT（Standard Widget Toolkit）最初由IBM开发，是一套用于Java的GUI系统，用来和Swing竞争，而开源集成开发环境Eclipse就是用Java和SWT开发的，具体见http://en.wikipedia.org/wiki/Standard_Widget_Toolkit。（本书脚注若无特殊说明均为译者注。）

② GTK+使用C语言开发，是类Unix系统下开发GUI应用程序的主流开发工具之一。它是自由软件，并是GNU计划的一部分，具体见http://zh.wikipedia.org/wiki/GTK。

③ REPL就是Read-Eval-Print Loop的缩写，意思是“阅读–评估–打印–循环”，它既可以作为独立的程序运行，也很容易作为程序的一部分使用。它为运行JavaScript脚本和查看结果提供了一种交互方式，详细内容见http://nodejs.org/docs/v0.6.6/api/repl.html 和http://nodejs.org/docs/v0.6.6/api/tty.html。

Web诞生之初，编写Web应用的工具还不够成熟。有些人尝试用Perl或TCL编写CGI脚本，PHP和Java语言刚刚被开发出来，而JavaScript甚至也被用于服务器端。Netscape的LiveWire服务端作为早期的Web应用服务器，就是用JavaScript编写的。微软ASP的某些版本使用的是JScript——它们自己的JavaScript实现。最近的一个服务器端JavaScript项目是在Java领域中使用的RingoJS应用框架。RingoJS构建于Rhino[①]之上，Rhino是一个用Java编写的JavaScript实现。

Node带来了一个前所未见的组合，那就是高速事件驱动I/O和V8高速JavaScript引擎；V8这个超快的JavaScript引擎是Google Chrome浏览器的核心。

1.2 为什么要使用 Node

JavaScript由于无处不在的浏览器而非常流行。它实现了许多现代高级语言的概念，比其他任何语言都不逊色。多亏了它的普及，软件行业才有大量经验丰富的JavaScript人才储备。

JavaScript是一门动态编程语言，拥有松散类型且可动态扩展的对象（能按需非正式地声明）。函数是一级对象，通常作为匿名闭包使用。这使得JavaScript比其他常用于编写Web应用的语言更加强大。理论上，这些特性使开发者的工作更加高效。平心而论，动态和非动态、静态类型和松散类型语言之间的优劣尚无定论，而且可能永远不会有定论。

JavaScript的一个短板是全局对象。所有的顶级变量都被扔给一个全局对象，这在混用多个模块时会导致难以预料的混乱。由于Web应用通常有大量的对象，且很可能是多个组织编写的，所以你自然会认为Node编程中的全局对象冲突会是个“雷区”。但其实不然，Node使用CommonJS[②]模块系统，这意味着模块的局部变量即使看起来像全局变量，实际上也是局部变量。这种模块间的清晰分离避免了全局对象的问题。

在Web应用服务器端和客户端使用同样的编程语言是人们由来已久的梦想。这个梦想可以追溯到早期的Java时代，那时Applet是用Java编写的服务器应用的前端，而对JavaScript的最初设想是将其作为Applet的一种轻量级脚本语言。但世事难料，到头来JavaScript取代Java成为在浏览器中使用的唯一语言。有了Node，在客户端和服务器端使用相同编程语言的梦想终于有望实现了，这门语言就是JavaScript。

语言在前后端通用有如下几个优势：

- 网线两端可能是相同的程序员；
- 代码能更容易地在服务器端和客户端间迁移；

---

① Rhino是用Java编写的JavaScript引擎，其设计目标是借助于强大的Java平台API简化JavaScript程序的编写。Rhino是Mozilla开发的自由软件，其最新的1.7r3版本实现了部分ECMAScript 5，可以从http://www.mozilla.org/rhino/下载它的源代码。

② CommonJS是一种规范，Node实现了这个规范。JavaScript是一门强大的、面向对象的语言，它有很多快速高效的解释器。JavaScript标准定义的API是为了构建基于浏览器的应用程序，不存在用于更广泛应用程序的标准库。CommonJS定义了构建很多普通应用程序（主要指非浏览器应用）的API，从而填补了这个空白。CommonJS的终极目标是提供一个类Python、Ruby和Java的标准库。这样的话，开发者可以使用CommonJS API编写应用程序，然后这些应用可以运行在不同的JavaScript解释器和不同的主机环境中。更多内容见http://www.commonjs.org/。

- 服务器端和客户端使用相同的数据格式（JSON）；
- 服务器端和客户端使用相同的开发工具；
- 服务器端和客户端使用相同的测试或质量评估工具；
- 当编写Web应用时，视图模板能在两端共享；
- 服务器端和客户端团队可使用相似的编程风格。

Node作为一个引人注目的平台，加上开发社区的共同努力，很容易把上述这些（甚至更多）优势变成现实。

1.2.1　架构问题：线程，还是异步事件驱动

Node的异步事件驱动架构据说是其拥有极高性能的原因。好吧，应该说还有V8 JavaScript引擎的巨大功劳。通常应用服务器端使用阻塞I/O和线程来实现并发。阻塞I/O会导致线程等待，从而造成线程资源浪费，因为当应用服务器处理请求时，需要等待I/O执行结束才能继续处理。

Node有一个无需I/O等待或执行环境切换的单独执行环境。Node的I/O调用会转换为请求处理函数，当某些数据可用时事件轮询会调度事件让这些函数工作。事件轮询和事件处理程序模型差不多，就像浏览器中的JavaScript一样。程序的执行最好能快速返回到事件循环中，以便马上调度下一个可执行的任务。

为了说明这一点，Ryan Dahl（在他的“Cinco de Node”演讲[①]中）问我们当执行如下代码时会发生什么：

```
result = query('SELECT * from db');
```

当然，在数据库层把这个查询发送到数据库、数据库查询结果并返回数据期间，程序会暂停。根据具体的查询情况，暂停时间可能相当长。这非常糟糕，因为线程空闲时可以处理另一个请求，如果所有线程都繁忙（记住计算机资源有限），请求将被丢弃。这当然是浪费资源。执行环境切换也不是没有代价的，使用的线程越多，CPU存储和恢复状态消耗的时间就越多。此外，每个线程的执行栈都占用内存。而是通过使用异步事件驱动的I/O，Node会省掉这里的大部分开销而其自己带来的开销却很小。

用线程实现并发通常面临类似这样的声音，即“开销大易出错”、“Java易出错的同步原语”或“设计并发软件复杂易出错”（实际上引号中的内容来自真实的搜索引擎结果）。复杂性来自于对共享变量的访问、避免死锁的各种策略和线程间的竞争。“Java的同步原语”就是这样一种策略，显然许多程序员发现其难以实现，因此倾向创建类似`java.util.concurrent`这样的框架来解决线程并发的问题，但有些人可能会认为掩盖复杂性并不会使事情更简单。

Node从不同的角度解决并发问题。通过事件轮询实现异步触发回调函数是一种更简单的并发模型，好理解且好实现。

Ryan Dahl指出理解读取对象的时间可以帮助理解异步I/O的重要性。从内存中读取的对象（纳

---

① 这是雅虎主办的技术会议，在这个会议上Ryan Dahl介绍了Node。具体情况和视频见http://www.yuiblog.com/blog/2010/05/20/video-dahl/。

秒级）比从硬盘或网络中读取对象（毫秒级或秒级）要快很多。读取外部对象的时间非常长，当用户等待页面加载完成，坐在浏览器前无聊地移动鼠标超过两秒时，这就显得没完没了了。

在Node中，前面的查询应该这样写：

```
query('SELECT * from db', function (result) {
  // 操作结果
});
```

这些代码实现了前面提到的查询。不同的是，查询结果不是调用函数的结果，而是提供给了一个稍后将调用的回调函数。此时，控制会立即返回事件循环，而服务器能够继续处理其他请求。这些请求中将会有一个是对查询的响应，那么该请求将调用回调函数。这种快速返回事件循环的模型确保了更高的服务器利用率。对于服务器的拥有者来说这太棒了，但更大的好处是它能让用户更快看到页面内容。

通常网页会引用几十个来源的数据，每个请求都会涉及上面讨论的查询和响应。通过异步查询，它们能并行发生，所以页面构造函数能同时发出几十个查询——无需等待且每个都有自己的回调函数——然后返回事件循环，而每个查询返回后会调用相应的回调函数。因为并行，所以收集数据比这些查询一个一个地同步完成要快得多。现在用户更高兴了，因为网页打开得更快了。

1.2.2 性能和利用率

Node之所以令人兴奋，还因为它的吞吐能力（每秒能响应的请求数）。与相似的应用（比如Apache）进行基准测试比较，可以看到Node的性能提升很大。

下面这个简单的HTTP服务器就是一个基准测试程序，它只是返回“Hello World”信息：

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
```

这是用Node能够构建的一个简单的Web服务器。`http`对象封装了HTTP协议，其`http.createServer`方法能创建一个完整的Web服务器，同时通过`.listen`方法监听指定端口。Web服务器上的每个请求（来自是任何URL的GET或PUT）都会调用这里提供的函数，就这么简简单单的几行代码。在这种情况下，无论是什么URL，它都会返回一个简单的`text/plain`响应“Hello World”。

由于简单，所以这个应用用以用来度量Node请求的最大值吞吐量。实际上许多已公布的基准测试程序都是从这个最简单的HTTP服务器开始的。

Node作者Ryan Dahl演示过一个简单的基准测试程序（http://nodejs. org/cinco_de_node.pdf），该程序返回一个1 MB的缓冲区（buffer），Node每秒能处理822个请求，而nginx每秒处理708个。他还指出nginx的峰值是4 MB内存，而Node是64 MB。

Dustin McQuay（http://www.synchrosinteractive.com/blog/9-nodejs/22- nodejs-has-a-bright-future）演示了据他所称类似的Node和PHP/Apache程序：

- PHP/Apache吞吐量为3187请求/秒；
- Node.js吞吐量为5569请求/秒。

RingoJS的作者Hannes Wallnöfer写了一篇博文提醒大家不要根据基准测试结果做重要决策（http://hns.github. com/2010/09/21/benchmark.html），随后使用基准测试程序比较了RingoJS和Node。RingoJS是一个应用服务器，构建在基于Java的Rhino JavaScript引擎之上。取决于应用场景，RingoJS和Node的性能没有多大区别。测试结果表明对于具有快速缓冲区或字符串分配机制的应用，Node的性能不如RingoJS。在后来的一篇博文中，他使用解析JSON字符串来模拟一个常见的任务，并发现RingoJS表现得更好。

Mikito Takada发表了关于基准测试和性能改进的博文，文中比较了基于Node构建的“48 hour hackathon”（48小时黑客马拉松）应用（http://blog.mixu.net/2011/01/17/performance-benchmarking-the-node-js-backend-of-our-48h-product-wehearvoices-net/）和一个他声称使用Django编写的类似应用。未经优化的Node版本相比Django版本有点儿慢（响应时间长），但一些优化（MySQL连接池、缓存等）极大地改进了其性能，使其轻而易举的击败了Django。最终的性能图表显示其性能（请求数/秒）几乎可与之前讨论的简单“Hello World”基准测试程序相媲美。

注意，Node应用高性能的关键是要快速返回事件循环。我们会在第4章对此进行详细介绍，如果回调处理程序执行时间“太长”，用Node也很难成就极快的服务器。在Ryan Dahl关于Node项目的一篇早期博文（http://four.livejournal.com/963421.html）中，他论述了事件处理程序执行时间要小于5 ms的必要性。这篇博文中的大部分想法都没有实现，但Alex Payne写了一篇关于它的有趣博文（http://al3x.net/2010/07/27/node.html）分析了“小规模应用”（scaling in the small）和“大规模应用”（scaling in the large）的区别。

对于小型Web应用，在用Node编码取代通常的“P”语言（Perl、PHP、Python等）编码时具有性能和实现上的优势。JavaScript是一门强大的语言，同时使用现代高速虚拟机设计的Node环比像PHP这样的解释语言更具性能和并发上的优势。

在讨论“大规模应用”时他说，企业级应用的编写总是困难且复杂的。典型的企业级应用都会使用负载均衡、缓存服务器、大量冗余机器，这些机器很可能分散在各个不同的地方，为世界各地的用户提供快速的Web浏览体验。或许对于整个系统来说应用开发平台并不那么重要。

在看到Node被真正长期实际地采用之前，很难知道它真正的价值有多大。

1.2.3　服务器利用率、成本和绿色 Web 托管服务

追求最佳效率（每秒内处理更多的请求）不仅仅是要通过优化得到极客一般的满足感，还要追求真正的商业和环境效益。像Node服务器那样，每秒处理更多的请求就意味着不必再购买大量的服务器。本质就是用更少的资源做更多的事情。

大致来说，购买更多的服务器，就要消耗更多电力和造成更大的环境污染，反之，服务器越

少，投入越少，对环境污染也越少。对于降低运行Web服务器成本和对环境的影响，现已存在一个专业的研究领域。这里的目标相当明显：更少的服务器、更低的成本和更小的环境污染。

英特尔的文章“通过服务器电能测量提高数据中心效率”（Increasing Data Center Efficiency with Server Power Measurements，http://download.intel.com/it/pdf/Server_Power_Measurement_final.pdf）给出了用于理解效率和数据中心成本的客观框架。有许多因素（诸如建筑、冷却系统和计算系统设计）都会对成本和环境产生影响。高效的建筑设计、冷却系统和计算机系统（数据中心效率、密度和存储密度）能降低成本和环境影响。但你可能因为部署一个低效的软件框架需要购买更多的服务器，以致毁掉这些收益，然而你也可以通过使用高效的软件框架放大数据中心的效率。

1.3 Node、Node.js 还是 Node .JS

这一平台的名字是Node.js，但本书使用Node这个拼写形式，因为我们遵循了nodejs.org网站的提示，它说Node.js（小写的.js）是商标，但整站都使用Node这个拼写形式，本书也与官方网站保持一致。

1.4 小结

本章介绍了很多知识，具体包括：

- JavaScript在浏览器之外亦有一番天地；
- 异步和阻塞I/O的不同之处；
- 关于Node的基本情况；
- Node的性能。

介绍完Node之后，我们准备深入讲述如何使用它。第2章将讨论如何设置Node环境，准备好了吗？

第 4 章 几种典型的简单应用

学习完Node模块之后，接下来就是学以致用的时候了。在这一章里，我们会尽量保持应用足够简单，从而可以把研究重点放在3个不同的Node应用框架上。之后几章我们会做一些复杂的应用，俗话说得好：学会跑之前，得先学会走。

开始吧。

4.1 Math Wizard

在这一章，我们所要做的就是一个简单的Math Wizard应用。Math Wizard的用户体验很不错，适合用于对儿童讲授算术。由于我们没有擅长用户体验的行家，所以做出来的Math Wizard应用只能用于对Node使用者讲授Web应用开发。有一点需要事先提醒下，不要期望你的孩子能够通过这个应用成为数学天才。

Math Wizard应用包括了主页、侧边导航栏和其他几个允许用户进行算术操作的页面。

是否使用 Web 框架

Web框架可以让你从复杂的HTTP协议实现中解脱出来。远离细节是一个程序员保持高效的工作方式。当你使用一个库或者框架，而这个库或者框架提供了很多封装好的函数来帮助处理细节时，这个感觉会特别明显。

在这一章，我们会先从编写一个不依赖于框架的应用（Math Wizard）开始，然后使用Connect和Express分别进行渐进增强式的开发。

4.2 不依赖框架的实现

我们先循序渐进学习Node应用的构建，慢慢感受Web框架带来的便捷。这意味着我们要从Node的核心包HTTP Server对象开始学起。

和其他Web应用一样，Math Wizard包含很多页面，每个页面对应一个URL。每个页面都有一些常见的元素（常用的页面结构和导航栏），不过每个页面的内容是不同的。在Math Wizard中，URL规则如下：

- / wizard应用的主页；
- /square 用于计算一个数的平方；
- /mult 用于计算两个数相乘；
- /factorial 用于计算一个数的阶乘；
- /fibonacci 用于计算斐波那契数。

我们先创建一个文件夹来存放源代码：

```
$ mkdir chap04
```

4.2.1 路由请求

Math Wizard的每一个页面都是由一个独立的模块实现，然后服务器把请求分别路由到这些模块。

我们所指的“请求的路由选择”就是一种把应用切分成多个模块的方式。与其在一个庞大的回调函数中实现应用的每一个细节，不如使用模块化的处理方式。请求的路由选择首先需要有代码对进来的HTTP请求进行检查，然后调用对应的模块处理请求。

创建一个app-node.js，其所包含内容如下：

```
var http_port = 8124;

var http    = require('http');
var htutil  = require('./htutil');

var server = http.createServer(function (req, res) {
  htutil.loadParams(req, res, undefined);
  if (req.requrl.pathname === '/') {
    require('./home-node').get(req, res);
  } else if (req.requrl.pathname === '/square') {
    require('./square-node').get(req, res);
  } else if (req.requrl.pathname === '/factorial') {
    require('./factorial-node').get(req, res);
  } else if (req.requrl.pathname === '/fibonacci') {
    require('./fibo-node').get(req, res);
    // require('./fibo2-node').get(req, res);
  } else if (req.requrl.pathname === '/mult') {
    require('./mult-node').get(req, res);
  } else {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end("bad URL "+ req.url);
  }
});

server.listen(http_port);
console.log('listening to http://localhost:8124');
```

这个请求路由器是非常简单的。每一个HTTP请求的回调函数都使用了req变量来持有请求的数据，res用来持有响应数据。这一请求路由器会检查请求的URL，然后将请求转发到对应的处理函数。

这一应用由一些模块组成，这些模块分别对应一个页面且都暴露了一个叫做.get的函数。这个函数使用了function(req, res)语句来实现Math Wizard中的一个页面。

如果请求的URL没有匹配任何模块，我们会返回一个404状态码，这个状态码说明对应的页面没有找到。

4.2.2 处理 URL 查询参数

htutil.loadParams函数可以帮助我们完成URL的解析并保存解析后的对象，这样Math Wizard应用的其他部分也可以引用这个对象。每一个页面都会包括一个输入为a和b的表单。当用户输入数字并点击提交按钮Submit时，URL会包含一个查询字符串，如下所示：

```
http://localhost:8124/mult?a=3&b=7
```

这些查询参数只会在输入数字并点击提交按钮的时候生成。这意味着每一个Math Wizard页面必须同时兼容参数存在和不存在的情况。htutil.loadParams函数可以很方便地查找这些参数，从而减少各个模块中重复的代码。

创建一个htutil.js，使其包含如下的代码：

```
var url = require('url');
exports.loadParams = function(req, res, next) {
  req.requrl = url.parse(req.url, true);
  req.a = (req.requrl.query.a && !isNaN(req.requrl.query.a))
    ? new Number(req.requrl.query.a)
    : NaN;
  req.b = (req.requrl.query.b && !isNaN(req.requrl.query.b))
    ? new Number(req.requrl.query.b)
    : NaN
    if (next) next();
}
```

loadParams函数会由HTTP请求处理函数调用，并接收HTTP请求处理函数传递的req和res参数。它会查找a和b对应的查询参数，并将其添加到req对象里。例子中我们使用:?运算符来确保req.a和req.b拥有值NaN或者一个数字，这取决于查询参数是否存在，以精简其他代码。现在你可以先忽略名为next的函数，我们会在后面介绍Connect框架时讨论这个方法。

在htutil.js里还有另外两个函数，它们负责页面布局的处理。Math Wizard给每个页面提供了一个通用的布局，布局代码的集中处理可以减少重复的代码。在稍后使用Express框架的时候，我们可以使用模板文件来控制页面布局，但是现在在这一版Math Wizard中我们只使用Node的核心部分，自然也就不使用模板了。

在htutil.js里，我们继续添加下面两个函数。它们是两个帮助构造页面的效用函数。

```
exports.navbar = function() {
    return ["<div class='navbar'>",
        "<p><a href='/'>home</a></p>",
        "<p><a href='/mult'>Multiplication</a></p>",
        "<p><a href='/square'>Square's</a></p>",
        "<p><a href='/factorial'>Factorial's</a></p>",
```

```
        "<p><a href='/fibonacci'>Fibonacci's</a></p>",
        "</div>"].join('\n');
}
```

这个函数提供了用于链接其他页面的HTML片段。它可以作为一个导航栏，使用户通过它访问应用的各个页面。

```
exports.page = function(title, navbar, content) {
    return ["<html><head><title>{title)</title></head>",
      "<body><h1>{title}</h1>",
      "<table><tr>",
      "<td>{navbar}</td><td>{content}</td>",
      "</tr></table></body></html>"
      ].join('\n')
      .replace("{title}", title, "g")
      .replace("{navbar}", navbar, "g")
      .replace("{content}", content, "g");
}
```

这个函数对应的是整个页面的HTML结构。它调用了一些参数来填充标题、导航栏和页面的内容。

我们在这里使用了正则表达式和replace函数，从而可以很方便地将数据替换为字符串。replace函数是一个字符串函数，它使用了一个正则表达式来匹配字符串，然后用提供的字符串替换匹配成功的文本。

下一节我们会介绍如何使用这些函数。

4.2.3　乘法运算

现在让我们看下如何在Math Wizard里创建一些能够进行数学运算的页面。首先要实现的是乘法（比如a*b）页面。

创建一个mult-node.js，使其包含下面的代码：

```
var htutil = require('./htutil');
exports.get = function(req, res) {
  res.writeHead(200, {
    'Content-Type': 'text/html'
  });
  var result = req.a * req.b;
  res.end(
    htutil.page("Multiplication", htutil.navbar(), [
      (!isNaN(req.a) && !isNaN(req.b) ?
        ("<p class='result'>{a} * {b} = {result}</p>"
          .replace("{a}", req.a)
          .replace("{b}", req.b)
          .replace("{result}", req.a * req.b))
          : ""),
      "<p>Enter numbers to multiply</p>",
      "<form name='mult' action='/mult' method='get'>",
      "A: <input type='text' name='a' /><br/>",
      "B: <input type='text' name='b' />",
```

```
        "<input type='submit' value='Submit' />",
        "</form>"
      ].join('\n'))
    );
}
```

和其他Math Wizard模块一样，乘法模块有两个目的。首先是显示算术结果，然后是显示一个让用户输入数字（一个或两个）的表单，用于乘法运算。

首先要注意的是我们使用了htutil.page函数。它提供了总体的页面布局控制功能，在这个函数里，我们只提供页面的主要内容区域。这个内容区域是一个字符串数组，最后会被用.join()函数联接起来。

如果用户提供了一个参数，那么最关键的部分就是按照下面的代码显示结果：

```
(!isNaN(req.a) && !isNaN(req.b) ?
  ("<p class='result'>{a} * {b} = {result}</p>"
    .replace("{a}", req.a)
    .replace("{b}", req.b)
    .replace("{result}", req.a * req.b))
: ""),
```

这个例子使用了?:运算符来检查参数是否已经提供，如果提供了，那么对req.a和req.b进行乘法运算并显示结果。

4.2.4 其他数学函数的执行

其他的Math Wizard模块和mult-node.js类似，使用一样的模式创建，让我们快速了解一下。

一个数的平方是这个数乘以它本身（比如a*a）。创建square-node.js，使其包含下面的代码。注意，我们使用了Math.floor方法来确保对req.a的值取整。

```
var htutil = require('./htutil');
exports.get = function(req, res) {
  res.writeHead(200, {
    'Content-Type': 'text/html'
  });
  res.end(
    htutil.page("Square", htutil.navbar(), [
      (!isNaN(req.a) ?
        ("<p class='result'>{a} squared = {sq}</p>"
          .replace("{a}", req.a)
          .replace("{sq}", req.a*req.a))
      : ""),
      "<p>Enter a number to see its square</p>",
      "<form name='square' action='/square' method='get'>",
      "A: <input type='text' name='a' />",
      "</form>"
    ].join('\n'))
  );
}
```

n的阶乘，用数学表达式表示是n!。它是n及所有小于n的正整数相乘的结果。阶乘运算在数

学的很多领域都会用到。接下来创建一个factorial-node.js，使其包含下面的代码：

```
var htutil = require('./htutil');
var math  = require('./math');

exports.get = function(req, res) {
  res.writeHead(200, {
    'Content-Type': 'text/html'
  });
  res.end(
    htutil.page("Factorial", htutil.navbar(), [
      (!isNaN(req.a) ?
        ("<p class='result'>{a} factorial = {fact}</p>"
        .replace("{a}", req.a)
        .replace("{fact}",
          math.factorial(Math.floor(req.a))))
      : ""),
      "<p>Enter a number to see it's factorial</p>",
      "<form name='factorial' action='/factorial' method='get'>",
      "A: <input type='text' name='a' />",
      "</form>"
    ].join('\n'))
  );
}
```

斐波那契数就是如下顺序的一组整数：0、1、1、2、3、5、8、13、21、34、55等。斐波那契数列中的每个数都是它前两个数之和。连续数的比例接近于黄金比例。让我们创建一个名为fibo-node.js的文件，用于创建计算斐波那契数的页面：

```
var htutil = require('./htutil');
var math  = require('./math');
exports.get = function(req, res) {
  res.writeHead(200, {
    'Content-Type': 'text/html'
  });
  res.end(
    htutil.page("Fibonacci", htutil.navbar(), [
      (!isNaN(req.a) ?
        ("<p class='result'>fibonacci {a} = {fibo}</p>"
          .replace("{a}", Math.floor(req.a))
        .replace("{fibo}", math.fibonacci(Math.floor(req.a))))
       : ""),
      "<p>Enter a number to see its fibonacci</p>",
      "<form name='fibonacci' action='/fibonacci' method='get'>",
      "A: <input type='text' name='a' />",
      "</form>"
    ].join('\n'))
  );
}
```

一些眼尖的读者可能已经注意到一个叫做math的模块。这个模块包含了一些数学函数的实现。我们需要创建一个math.js文件，使其包含下面的代码：

```
var factorial = exports.factorial = function(n) {
```

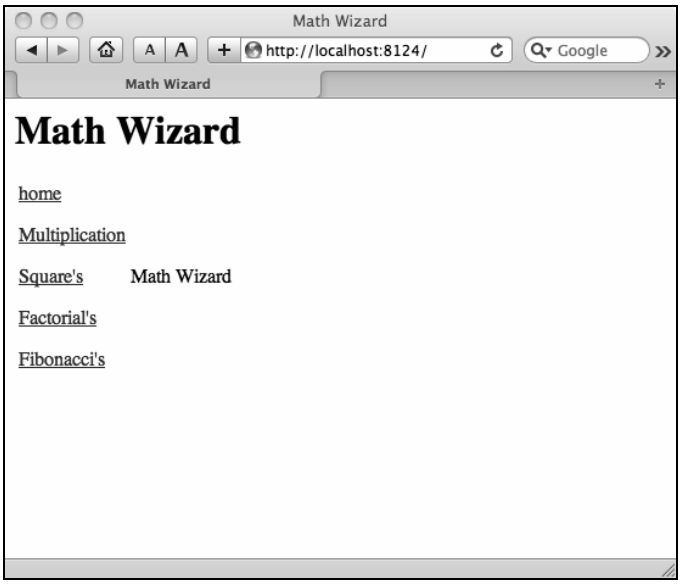
```
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}

var fibonacci = exports.fibonacci = function(n) {
  if (n === 1)
    return 1;
  else if (n === 2)
    return 1;
  else
    return fibonacci(n-1) + fibonacci(n-2);
}
```

这些方法简单地实现了一些标准的数学函数。就像我们能马上看到的，斐波那契函数是一个非常简单的计算密集型函数。

我们希望Math Wizard应用能有一个主页。创建一个homenode.js，使其包含下面的代码：

```
var htutil = require('./htutil');
exports.get = function(req, res) {
  res.writeHead(200, {
    'Content-Type': 'text/html'
  });
  res.end(
    htutil.page("Math Wizard",
      htutil.navbar(),
      "<p>Math Wizard</p>")
  );
}
```



输入下面的命令：

```
$ node app-node.js
```

因为app-node.js监听了8124端口，所以访问http://localhost:8124/的时候我们可以看到下面的内容。

4.2.5　扩展 Math Wizard

我们的孩子需要高质量的教育，而这个示例程序只是可教授算术的基本应用，但也许并不适合。数学运算是多种多样的，因此任何情况下，我们都可以很轻易地扩展Math Wizard，为其添加其他页面。遵循已有的模式给Math Wizard添加一个新的页面需要用到下面几个步骤。

- 在htutil.navbar上添加一个a标签。
  因为htutil.navbar函数包含了针对导航栏的HTML代码，任意一个新的Math Wizard页面都需要被像下面这样列出页面的URL和页面的名字：

```
"<p><a href='/newUrl'>Math Function Name</a></p>\n"+
```

- 在app-node.js里增加一条if语句。
  因为app-node.js文件包含请求的路由功能，所以需要一个新的if语句实现我们刚刚定义的URL的转发。这个URL需要匹配htutil.navbar函数里的URL。

```
if (req.requrl.pathname === '/newUrl') {
        require('./moduleName').get(req, res);
    }
```

- 增加一个页面处理函数模块，对外暴露一个get方法。
  我们之前已经看到一些处理函数模块（mult-node.js等），按照那些例子再创建一个这种模块还是很容易的。

4.2.6　长时间运行的运算（斐波那契数）

应用Math Wizard展示了Node应用中一个饱受批评的缺点。如果回调函数不能对请求进行快速的处理并返回Node事件循环，应用就会陷入泥潭。

为了让大家更直观地了解这个缺点，我们可以在斐波那契数计算页面上输入一个很大的数，比如50。这样就需要大量的时间来执行运算（大概需要几个小时到几天），Node进程的CPU占用率会变高，然后在其他浏览器窗口内，你可能无法使用Math Wizard了。所有这一切都是因为斐波那契序列的数的计算是非常庞大的任务。为什么浏览器也会无响应？这是因为密集型计算阻止了Node事件循环的执行，从而阻止了Node对浏览器请求进行的响应。

由于Node只有一个执行线程，请求处理程序必须在执行完之后快速返回事件循环。通常，异步编程能保证事件循环正常执行。甚至在加载地球另一端的服务器数据时，异步编程的作用也能体现出来，因为I/O是非阻塞的，控制能很快返回给事件循环。因为我们选择的简单斐波那契函数是一个长时间运行的阻塞型计算，所以它并不适用于这一模型。这样的事件处理程序会让系统

不用忙于处理请求，并让Node从即将处理的事务中解脱出来，从而实现一个极快的Web服务器。

在这个例子里，长时间运行的计算导致的问题（长响应时间问题）很明显。用于响应斐波那契数的时间几乎让你有时间去西藏进行一次欢快的旅行。长响应时间可能在你的应用中不是很明显，那么你如何知道自己的请求会占用太多时间呢？一个测量手段就是利用浏览器工具（比如YSlow）显示的响应延迟。用户使用浏览器的时候，能在一两秒内显示下一页是很重要的，否则你将有可能失去你的用户。

在Node里有两种解决这类问题的常见方案。

- **算法重构** 就像我们选择的斐波那契函数一样，非最佳的算法可以通过重写实现更高的效率。或者，即使没有优化空间，我们也可将其切分并通过事件循环分派到不同回调函数里。稍后我们会具体介绍一下这种方法。
- **创建一个后台服务** 你能想象一个专门用于计算斐波那契数的后台服务器吗？好吧，或许不能，但是这确实是非常普遍的将前台服务转移到后台执行的方法，我们会在这章的最后介绍如何实现一个后台运行的数学计算服务器。这个请求处理函数应可以异步地调用数据服务或者数据库、收集响应数据，然后发送给浏览器。

我们可以对斐波那契算法进行优化，但我们不这样做，而是将其从一个同步的函数转换成一个含有对应回调的异步函数。在这里使用异步的斐波那契算法并不是一个很好的选择，但它的确展示了算法重构方法。我们选择了对计算过程进行拆分，并通过事件循环分派给对应的回调函数。

首先，我们要对斐波那契函数进行实例化，用实例化后的对象取代原来实现的函数。如果已经写了一个比较原始又低效的函数，之后你可能不得不用一个更好的替换它。接下来在`math.js`中添加如下代码：

```
var fibonacciAsync = exports.fibonacciAsync = function(n, done) {
  if (n === 1 || n === 2)
    done(1);
  else {
    process.nextTick(function() {
      fibonacciAsync(n-1, function(val1) {
        process.nextTick(function() {
          fibonacciAsync(n-2, function(val2) {
            done(val1+val2);
          });
        });
      });
    });
  }
}
```

这就是我们的新的异步斐波那契算法。我们已经把它从一个简单的函数转换成一个异步驱动的计算函数，这样就可以通过回调函数传递结果，如下所示：

```
fibonacciAsync(n, function(value) {
   // 操作值
});
```

我们使用了`process.nextTick`方法将一个递归函数转换成各个步骤都由事件循环分派。这

个函数通过事件循环调用回调函数，确保函数能快速进入事件循环，这样服务器才能继续处理别的HTTP请求。这不是唯一一个通过事件循环派发算法步骤的方式。异步模块也可以做这个，它有一系列方法帮助实现异步的JavaScript。

在fibonacciAsync函数里，process.nextTick方法替换了原始算法里的这一表达式：

```
return fibonacci(n-1)+fibonacci(n-2);
```

这句代码的任务是计算两个斐波那契数，然后对其执行加法，最后将结果返回给调用函数。新的算法有3个异步函数来实现任务的各个步骤，并使用了process.nextTick方法确保这些都通过事件循环分派执行。

在我们介绍后面的内容之前，先花点时间斟酌下这个方案。它没有减少必需的计算量，只是将计算过程交给事件循环调度。它会使当前的Node进程占用所有CPU负载，这绝不是重构计算密集型算法（如斐波那契算法）的最佳途径。但是这展示了通过事件循环分派工作的技术，这个技术对一些算法来说是非常实用的，但是对其他算法就并不是那么实用。

这个技术是否实用还是取决于你和你使用的算法，你需要选择最佳的方法来处理长时间运行的计算。例如，这章的后面部分，我们会展示如何实现一个可以用HTTP访问的后台服务器，这项技术可以将计算任务推送到任何模块。

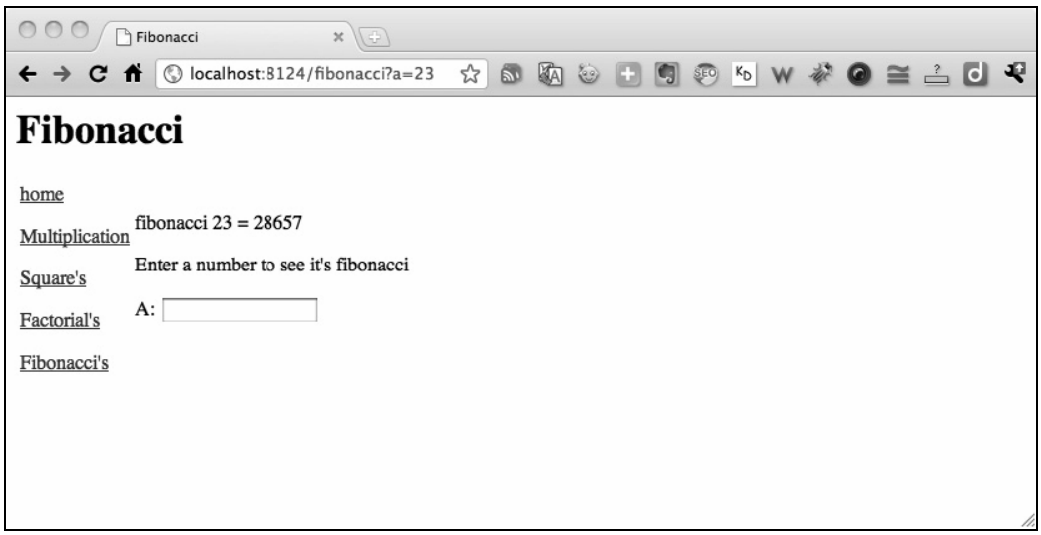
创建一个新文件fibo2-node.js，然后将app-node.js修改为require('./fibo2-node')，这样就可以使用新的斐波那契模块了。我们已经将这行代码添加到app-node.js文件里，不过它被注释掉了。接下来我们可以切换代码的注释来改变斐波那契算法的实现：

```
var htutil = require('./htutil');
var math  = require('./math');
function sendResult(req, res, a, fiboval) {
  res.writeHead(200, {
    'Content-Type': 'text/html'
  });
  res.end(
    htutil.page("Fibonacci", htutil.navbar(), [
      (!isNaN(fiboval) ?
        ("<p class='result'>fibonacci {a} = {fibo}</p>"
          .replace("{a}", a)
          .replace("{fibo}", fiboval))
      : ""),
      "<p>Enter a number to see its fibonacci</p>",
      "<form name='fibonacci' action='/fibonacci' method='get'>",
      "A: <input type='text' name='a' />",
      "</form>"
    ].join('\n'))
  );
}

exports.get = function(req, res) {
  if (!isNaN(req.a)) {
    math.fibonacciAsync(Math.floor(req.a), function(val) {
      sendResult(req, res, Math.floor(req.a), val);
    });
```

```
    } else {
      sendResult(req, res, NaN, NaN);
    }
  }
```

我们已经通过将运算转移到sendResult函数重构了斐波那契模块，我们以两种方式调用send Result，并依据是否需要显示斐波那契数判断以哪种方式调用。



虽然请求较大的斐波那契数依然需要很长的时间来计算，但是服务器不会阻塞，可以处理其他请求。当打开多个浏览器标签页的时候，你就会明显感觉到这点。在一个标签页中请求一个大的斐波那契数会需要很长的时间来计算。然后在另一个标签页中发起一个请求，就像你在截图里看到的，浏览器不是处于无响应状态，而是在顺利地处理请求。

4.2.7 还缺什么功能

在后面讨论Connect时，我们会发现Math Wizard里极小的dispatch函数做的事情和实际Web服务器做的相比要少一些。仅仅实现HTTP协议并不能造就一个完善的Web服务器或者Web应用，因为它缺失了很多近20年Web应用开发中积累下的最佳实践。

- Math Wizard不能关注请求的方式（GET、PUT、POST等）。要维持HTTP的语义就需要对GET、PUT或者POST请求进行不同的处理。
- 没有给错误的URL请求提供404页面。
- 没有为URL和表单屏蔽注入式的脚本攻击。
- 不支持cookie处理，也没有使用cookie维持会话。
- 不记录请求。
- 不支持身份验证。
- 不处理图像、CSS、JavaScript或者HTML这样的静态文件。

❑ 没有对页面尺寸、执行时间等施加限制。

就像我们将在介绍Connect和Express时看到的那样，Node的Web框架弥补了大部分丢失的特性。

4.2.8　使用 Connect 框架实现 Math Wizard

Connect（http://senchalabs.github.com/connect/）其实不是一个真正意义上的Web框架，而是Node的一个中间件[①]框架。它含有11个已绑定的中间件并具有丰富的第三方中间件以供选择。“中间件”这个术语或许会因为太过笼统让你感到困惑，因此接下来我们对这个词进行详细介绍。

TJ Holowaychuck将“中间件”描述为易于挂载和调用的模块，可以“无序”使用并为应用的快速开发提供常用的功能，比如请求的路由选择、身份验证、请求记录、cookie处理等（详见http://tjholowaychuk.com/post/664516126/connect-middleware-for-nodejs）。

中间件有以下两种形式。

❑ **过滤器**　处在中间层负责处理传入和传出的请求，但是本身不响应请求。过滤器的一个例子就是中间件*logger*，它专门提供可定制的信息记录。

❑ **供应者**　作为堆栈的终端，意味着一个进入的请求最后会在供应者处处理，并且供应者负责发送响应。供应者的一个例子就是为静态文件服务的中间件*static*。

在前面一节，我们看到了一个以`http.createServer`对象和一个函数构建的应用，这个函数会在每一个HTTP请求到达服务器的时候被调用。在使用Connect的时候，你就只需要用到`connect.createServer`对象，然后将中间件模块绑定到服务器上。作为中间件模块之一，`router`模块用于实现应用的URL。

了解这些内容之后，我们可以开始看一些代码了。

4.2.9　安装和设置 Connect

首先，确保Connect已经安装好了：

```
$ npm install connect
```

然后创建`app-connect.js`文件，使其内容如下：

```
var connect = require('connect');
var htutil  = require('./htutil');

connect.createServer()
  .use(connect.favicon())
  .use(connect.logger())
  .use('/filez', connect.static(__dirname + '/filez'))
  .use(connect.router(function(app){
    app.get('/',
      require('./home-node').get);
```

① 中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。

```
    app.get('/square', htutil.loadParams,
      require('./square-node').get);
    app.get('/factorial', htutil.loadParams,
      require('./factorial-node').get);
    app.get('/fibonacci', htutil.loadParams,
      require('./fibo2-node').get);
    app.get('/mult', htutil.loadParams,
      require('./mult-node').get);
  })).listen(8124);
console.log('listening to http://localhost:8124');
```

然后按照下面的方式启动服务器：

```
$ node app-connect.js
```

因为服务器监听了8124端口，所以直接在浏览器里访问http://localhost:8124/即可使用它。

恭喜！我们现在已经成功运行了第一个基于Connect开发的Node应用。

你会发现它看起来（及其行为）和之前的Math Wizard很类似，这是因为app-connect.js模块重用了app-node.js的模块。app.get函数所做的只是把请求转发到一个已存在的模块。

如果行为类似的话，那么Connect带来的优势在哪里呢？

不同点在于Connect提供了一个请求处理和调度框架来减轻应用开发的压力。它会帮你处理之前缺失的作为一个“完备Web服务器”应该有的特性，让你把精力更多的放在自己的应用上。但是这足以让Connect成为一个应用框架吗？

Connect没有以一个应用框架的形式呈现出来，而是作为一个应用框架的构建基石存在。Express就是一个建立在Connect之上的框架。Connect本身是非常实用的，了解Connect有助于理解Express，所以接下来我们会简单介绍下Connect，然后再介绍Express。

4.2.10 使用 Connect

我们已经学习了一部分有关Connect的知识，接下来看一些更详细的内容。Connect是Express框架的基础，它几乎突破了所有之前讨论过的基于HTTP服务器对象构建应用的限制。不要着急，我们先看一下app-connect.js。

在Connect里有很多配置服务器对象的方式。我们在app-connect.js里是这么做的：

```
var connect = require('connect');
connect.createServer()
  .use(connect.favicon())
  .use(connect.logger())
  .use('/filez', connect.static(__dirname + '/filez'))
  .use(connect.router(function(app){
    // 配置路由器
  })).listen( .. port number ..);
```

.use方法可用于绑定中间件到Connect服务器。它会配置一系列在接到请求时调用的中间件模块。当然，你的应用决定了使用哪些中间件模块。

.use方法允许我们链式调用.use从而获得更好的编程体验（server.use().use().use().use()）。

在这个例子里我们要配置的中间件有*favicon*、*logger*、*static*和*router*。

在创建一个Apache式访问记录时，logger中间件非常有用。默认情况下，logger会将数据输出到控制台上，但是也可以通过配置以其他任何格式输出或者记录到任意文件里。

static中间件实现一个“静态Web服务器”来传递存放在特定目录下的文件。这意味着如果你有一个目录，其中包含了要发送给浏览器的.html，.css或者.js文件，`connect.static`中间件可以帮你完成任务。

`favicon`是一些浏览器在地址栏、标签栏显示的小图片，这些小图片也是一个你的品牌存在的标志，favicon中间件会帮助处理这些图片。

router中间件用于将每一个URL请求正确传递到对应的处理函数。配置方法如下：

```
.use(connect.router(function(app){
  // 配置路由器
})
```

但是真正占主导地位的是路由器的配置代码，在配置代码处你可以声明应用要识别的URL，还有每一个URL对应的处理函数。router的配置形式如下：

```
app.requestName('path', function(req, res, next) {..});
```

*请求名*指的是指HTTP动作，比如`get`、`put`、`post`等。这意味着你可以在页面上设置一个HTML表单，将`method`设置为`POST`，然后用`app.get`函数将页面传递给浏览器，然后使用`app.post`函数来接收表单传递回的数据。我们会在第6章看到对应的例子，不过它看起来更像下面定义的函数：

```
app.get('/form', createPageWithForm);
app.post('/form', receiveValuesPostedWithForm);
```

上面例子里的回调函数比一般的请求处理函数多了一个函数类型的参数。它的定义形式是`function(req, res, next)`，`req`和`res`分别对应了HTTP请求数据和HTTP响应数据。`next`这个参数对应的是Connect提供的函数，用于确保所有的中间件函数已经执行。

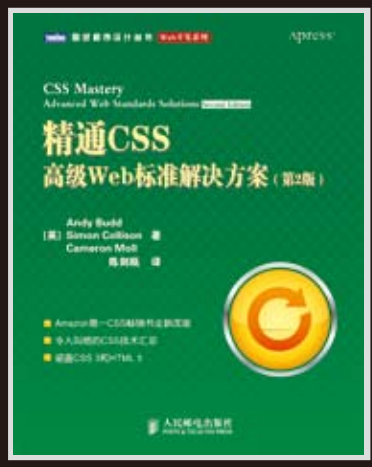
就像我们在`app-connect.js`里做的那样，路由可以分配到多个函数中。只要使用了`next`函数，Connect就可以依次对函数进行调用。和`app-node.js`里一样，我们在`app-connect.js`里使用的也是`htutil.loadParams`函数。你应该还记得，它使用了Connect提供的`next`函数。

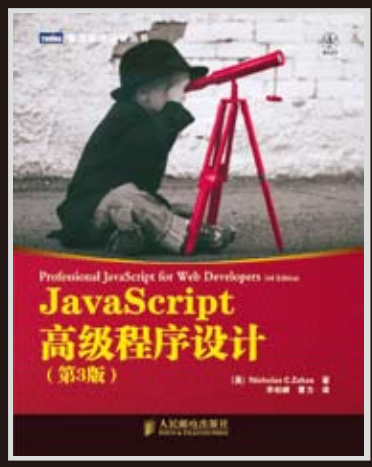
这是一个典型的路由器配置函数：

```
app.get('/square', htutil.loadParams,
 require('./square-node').get);
```

这个函数的参数是一个URL字符串和两个函数类型的参数，第一个是`htutil.loadParams`函数。路由器配置函数可以包含不限数量的函数，你可以为自己的应用构造一个处理函数队列。

总而言之，中间件和多个路由器函数组成了一种处理HTTP请求的状态机。我们已经看到了两个函数列。第一个是列在服务器配置中的中间件函数，还有就是我们刚刚看到的路由器函数列表。


CSS Mastery
精通CSS
高级Web标准解决方案（第2版）
Andy Budd
[英] Simon Collison 著
Cameron Moll
人民邮电出版社


Professional JavaScript for Web Developers
JavaScript
高级程序设计
（第3版）
人民邮电出版社


DOM Scripting
JavaScript DOM
编程艺术（第2版）
人民邮电出版社

Node Web Development

Node Web开发

作为服务器端的JavaScript解释器，Node是一个轻量高效的开发平台，用于构建响应快速、高度可扩展的Web应用。它使用事件驱动和非阻塞的I/O模型，非常适合开发数据密集、对实时响应要求高的分布式应用，在微软、eBay、LinkedIn、雅虎等世界知名公司均有成功的应用。

本书是Node开发基础教程，通过大量示例介绍如何使用HTTP服务器和客户端对象、Connect和Express应用框架、异步执行算法，以及如何结合使用SQL和MongoDB数据库。另外，本书同时针对开发和部署环境给出了实用的Node安装建议，介绍了HTTP服务器和客户端应用的开发，阐述了很多Node使用方式，包括在应用中使用数据库存储引擎，以及在有无Connect/Express Web应用框架的情况下开发网站的方法。本书还介绍了Node的CommonJS模块系统，帮助开发人员实现一些重要的面向对象设计方案。

本书适合具有一定JavaScript和Web应用开发基础知识、打算使用服务器端JavaScript开发高性能Web应用的开发人员阅读。

- 用Node平台打造高性能Web应用
- 雅虎架构师精准解读最炙手可热的Web开发技术
- 从基础到实践，示例丰富

[PACKT] PUBLISHING

图灵社区：www.ituring.com.cn
新浪微博：@图灵教育 @图灵社区
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机/网络技术/Node

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-27832-6
定价：35.00元