第 1 章 Node入门

1

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

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。

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[①]）；

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。

① 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。

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?"

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 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。

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[①]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

① 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/。

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；

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

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/。

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？