**函数式网页**

**Node.js:使用JavaScript创建高性能的网络程序**

**Stefan Tikov·innoQ**

**Steve Vinoski·Verivue**

Node.js，也称作Node，是一个服务器端的JavaScript运行环境（见http://nodejs.org）。它以谷歌的运行环境实现为基础，这个实现被称为V8引擎。V8和Node大多数情况下在C和C++中应用，重点关注性能和低内存消耗。但是，V8在浏览器中主要支持JavaScript（大多数情况下尤其是谷歌浏览器），而Node是为了支持服务器进程的长时间运行。

不同于其他大多数的现代运行环境，一个Node进程并不依靠多线程来支持业务逻辑的并发执行，它以一个异步的I/O事件模型为基础。Node的服务器进程可以看作是一个嵌入了JavaScript引擎去支持用户化的单线程后台程序。这与其他程序语言的事件系统不同，它以函数库的方式实现，所以Node在语言级上支持事件模型。

对这种方法来说，JavaScript是一个出色的选择，因为它支持事件的回调。举个例子，当一个浏览器完全地加载一个文件、一个用户按下一个按键或者一个Ajax的需求得到满足时，一个事件会触发一个回调。JavaScript的函数性特征使其能够极其简单地去创建能够注册为事件处理器的匿名函数对象。

**多线程互斥事件**

处理多个I/O来源，比如操作多个客户端连接的网络服务器的应用程序开发者，早就已经使用多线程的编程技术。开发者通过使用这种技术，能够使把他们的应用程序分作同时协作执行的几个行为，所以它变得非常受欢迎。如此一来，不仅使程序的逻辑变得更容易去理解、实现和维护，而且能使程序运行得快、更高效。

对于某些应用程序，比如执行大量I/O的网页服务器，多个线程能使应用程序更好地运用可用的处理器。当每个芯片一一对应地同时执行互不相同的线程时，在一个现代的多芯片系统中运行多个并发的线程是简单可行的。在单芯片系统中，单个处理器首先执行一个线程，然后转换到另一个线程并执行它，以此类推。例如，当目前的线程执行一个I/O操作，比如写入到一个TCP插口时，处理器会把它的运行环境切换到另一个线程。这种切换之所以发生是因为完成那个操作需要许多个处理器周期。处理器与其浪费时间去等待插口操作完成，不如在运转过程中设置I/O操作并执行另一个线程，从而使自己忙于有用的工作中。由于在等待I/O的过程中处理器不再是被封锁，所以当I/O操作完成时，处理器再次考虑回到原来的线程并准备去执行它。

尽管许多开发人员已经成功地把多线程应用于产品的应用程序中，但是他们中的大多数仍然同意多线程编程并不是那么容易掌握运用的。它存在着许多的问题，而这些问题很难去独立出来解决，比如死机停顿和未能保护在线程之间分享的资源。由于主要由OS决定哪个线程执行和执行多久，所以当运用多线程时，开发者也失去了一定程度上的控制。

事件驱动编程提供了一个更高效的可供替代的选择，在应用程序活动中为开发人员在线程转换上提供更多的控制权。在这个模型中，应用程序依赖于事件通知工具，例如Unix系统调用的select（）和poll（），Linux epoll服务和在BSD Unix变量中可调用的kqueue及kevent如OS X。应用程序在某些事件中注册了权利，例如正准备在特定的插口上读取的数据。当事件发生时，通知系统会及时通知程序从而使得它能及时处理事件。

对于事件驱动编程来说，异步I/O非常重要，因为它防止应用程序在等待一个I/O操作的过程中被锁定。例如，如果应用程序写入到一个插口并且填充了插口的潜在缓冲区，通常地，这个插口会阻塞应用程序的写入直至缓冲区的空间变得可使用，因此阻止了应用程序进行任何其他有用的工作。但是，如果插口不是处于封锁状态，它就会返回一个目前不能进行进一步写入的指示给应用程序，从而通知了应用程序等下再来试一遍。假设应用程序已经在那个插口中注册了事件通知系统的权利，那么它可以去做其他的事情，因为当插口的写入缓存区有可用的空间时，应用程序将会接收到一个事件通知。

和多线程编程相似，异步I/O的事件驱动编程也是有问题的。其中一个问题是，不是所有进程间通信的通道都能被连接到前面我们提到的事件通知工具。例如，在大多数的OS中，对于两个通过共享内存通信的应用程序来说，共享内存的分段没有提供句柄或者使程序能在事件中注册的文件描述符。当遇到这种情况时，开发人员必须求助于其他替代手段，比如在写入到共享内存的同时写入到一个通道或者一些其他事件能力机制中。

另一个重大的问题是，用某种编程语言书写应用程序去处理事件和异步I/O存在绝对复杂性。这是因为不同的事件在不同的环境中需要不同的做法。典型地，程序使用回调函数去处理事件。在缺少匿名函数以及闭包特性的语言中，比如C，开发者必须特别为每个事件和每个环境编写独立的函数。当使用这些函数去解决一个事件变得难以置信的复杂时，需确保函数全都能和它们需要的数据以及环境信息联系起来。所以，很多这样的程序完成后仅仅是变得令人费解，造成条式代码和全局变量不可维护地混乱缠绕着。

**不是传统意义上的JavaScript**

无论你是怎样看待JavaScript这个编程语言的，它已经毫无疑问地成为了任何基于HTML现代程序的主要元素。服务器端的JavaScript是逻辑上的下一步，使一个单一的编程语言能够应用在一个基于网页的分散式应用程序的方方面面。这个想法并不新颖，实际上，Rhino JavaScript执行环境已经被使用很久了。尽管如此，服务器端JavaScript还不是一个主流的方法，而仅仅在最近才受到了巨大的欢迎。

我们相信造成这种效应的原因有很多。一系列以HTML 5为标志的技术的共同到来降低了对可替代客户端平台的吸引力，明确了学习和开发JavaScript去建立丰富用户界面的强制执行必要性。NoSQL-type数据库，比如CouchDB和Riak，使用了JavaScript去定义数据视图以及过滤规则。其他动态语言，比如Ruby和Python，也已成为服务器端开发可接受的选择。最后，Mozilla和Google都已推出高性能的运行极快且容易掌握的JavaScript运行环境实现方式。

**Node编程模型**

Node的I/O方法是严格的：异步互动并不是例外，它们也服从规则。每一个I/O操作通过高阶函数的方法处理，高阶函数把一个函数作为自己参数，从而详细说明当有需要的时候怎么去执行。只有在极少数情况下，Node的开发者添加一个能同时工作的常用函数—比如，为了移除或者重命名文件。但是，一般地，当网络需求或者文件I/O的操作被调用时，控制会立刻返回到调用者手中。当有趣的事情发生时—例如，如果可以从一个网络的插口读取并获得数据，一个输出流就准备写入，或者当错误发生时—合适的回调函数就会被调用。

图1是实现一个为来自磁盘静态文件服务的HTTP网页服务器的简单例子。即使对于非网页开发者来说，比如那些测试过任何像C的语言的开发者，JavaScript的句法也是相当明显而熟悉的。其中一个特定主题是function（...）句法，这创立了一个未命名函数。JavaScript是一个函数式语言，正因如此，它支持高阶函数。任何一个开发者编写或者阅读Node程序都会发现这些细节。

程序的主要流程由明确被调用的函数决定。这些函数宁可注册合适的处理器回调，也从不在任何与I/O相关的东西上封锁。如果你已经在其他编程语言的事件函数库中见过相似的概念，你可能想知道明确封锁后在哪里调用事件循环隐藏。事件循环概念对于Node的活动性能来说是非常核心的，以致它隐藏在实现中；这个程序的主要目的是简单地去设立相应的处理期。http.createServer函数，是一个低水平效率的HTTP协议实现的外包装，把传递函数作为唯一的参数。当以往的数据为新需求而准备被读取时，这个函数就被调用。在另一个环境中，同时读取和传回一个文件这种幼稚的做法可能摧毁事件的作用。Node没有提供同时读取一个文件的机会—唯一的选择是，每当数据能够被读取时，通过readFile注册另一个被调用的函数。

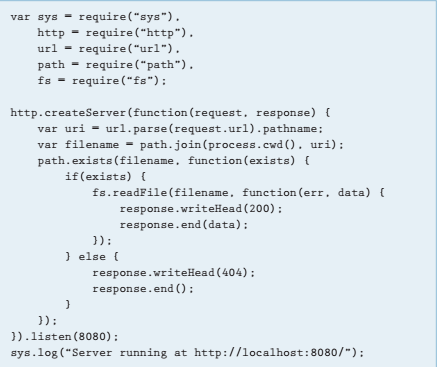


图1.一个简单的HTTP文件服务器。事件触发执行输入或输出操作的匿名函数。进入请求引发

服务器去解析目标URI，寻找与URI路径匹配的本地文件，如果找到，读取文件目录并且把它们

独立写入合适的HTTP头文件中，作为对客户端的响应。

**并发程序设计**

通常使用一些方法像“node<scriptname>”从命令行中调用的Node服务器进程，虽然是单线程运行，但是能够同时服务很多的客户端。这听起来是矛盾的，但是仔细想想，你会发现在代码周围有一个隐式主循环，并且实际上在循环中发生的东西只是许多的注册回调。并没有真实的I/O，更不用说业务逻辑进程处理，发生在循环体中。I/O相关事件触发实际的进程处理，比如正在实施的链接，或者，正在发送或从插口中接收的字节，或外部的系统。

图2是一个稍微更复杂的过分简单化HTTP服务器变量，但是它可以做到更多。此外，它从一个HTTP请求中解析了URI并且在服务器中映射URI的路径组件到一个文件名上。但是这次，文件被读取成更小的数据块而不是立刻全部被读取。在一定情境中，为回调方案提供的函数被调用。这类情境包括，当文件系统层准备传送大量的字节到应用程序时，当文件已经被全部读取时，或者当某种错误出现时。如果数据是可获得的，它就被写入到HTTP输出流。Node复杂的HTTP函数库支持使用HTTP 1.1 chunked把数据转换成编程语言。此外，从文件中读取和写入到HTTP数据流都是异步发生的。

图2中的例子展示了开发者可以在适度的资源需求下很简单地建立一个高性能的异步事件驱动网络服务器。主要原因是JavaScript具有函数功能特征，支持事件回调。事实上，这个模式对任何客户端的JavaScript开发人员来说是非常有名的。另外，导致的异步I/O缺乏使得开发者必须一开始就接受异步模型。这是Node和在其他编程环境中使用异步I/O的主要不同之一，在其他环境中，这仅仅是许多选择中的一个并且通常被认为是过分超前的。

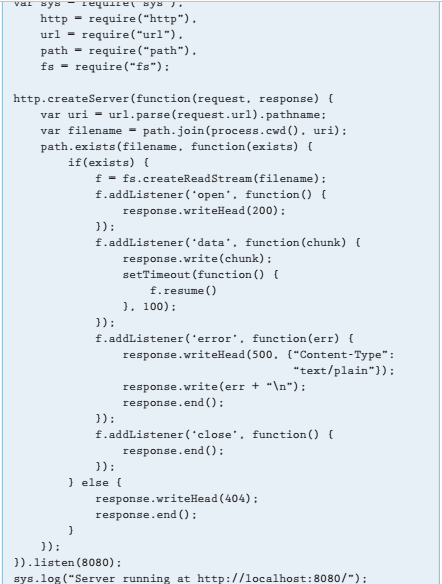


图2.一个简单的流动HTTP文件服务器。文件的组块从磁盘中被读取，然后传输到客户端

并使用HTTP的“chunked”转换成代码。

**运行多个程序**

在超过一个可使用的物理CPU或者芯片的硬件环境中，并行执行不是一个错觉而是一个事实。尽管OS可以通过它的异步I/O互动高效地安排一个Node进程，而同时允许其他进程在系统中运行，可是Node仍然在一个简单的进程中运行，因此，不能并行执行它的核心业务逻辑。这个问题在Node世界中最常见的解决办法是运行多个进程实例。

为了支持这个功能，multi-node函数库（可见http://github.com/kriszyp/multi-node）利用了OS在进程间分享插口的性能（并且在少于200行的Node JavaScript中实现）。举个例子，你可以通过调用multi-node的lis-ten（）函数，而同时运行比如图1和图2中显示的那些HTTP服务器。这就开始运行了多个进程，这些进程都听命于相同的端口，从而高效地把OS作为一个有效的负载平衡器来使用。

**服务器端的JavaScript生态系统**

Node是支持服务器端JavaScript开发的其中一个比较有名的框架和环境。程序员群体已经创建了一整个函数库系统给Node，或者说与Node兼容。其中，一些工具如node-mysql或者node-couchdb通过在相关的NoSQL数据存储时支持异步互动，分别地扮演着重要的角色。很多框架提供了一个功能全面的网页堆栈，比如Connect和Express，在一定程度上比得上Ruby世界中的Rack和Rail，如果还未受到欢迎的话。Node包管理器，npm，使函数库及其附带能够顺利安装。最后，客户端JavaScript的许多可使用的函数库被编写去适应CommonJS模块系统，这些库也可与Node一起配合工作。一个令人印象深刻的可使用模块列表在http://github.com/ry/node/wiki/modules。

鉴于以上所述，在大多数网页开发工程中，JavaScript相关知识是先进UI互动的一个先决条件，使用一种编程语言去处理每件事这个选择非常吸引人。Node.js的构筑学使得运用一个有高度表现力的函数式语言去进行服务器编程变得简单，而不需要牺牲性能和走出编程的主流。