# 第一章 欢迎进入Node.js 世界

Node.js是什么？你可能已经有所耳闻，甚至已经用上了，你也有可能对它很好奇。现在Node还很年轻，却非常流行。它在github受关注项目排行榜上位列第二，在Google小组和IRC频道中都有很多追随者，并且社区同仁们在NPM包管理网站上发布的模块多达15000个。所有这些都足以表明这个平台的强大吸引力。

官网上给Node下的定义是：“一个搭建在Chrome JavaScript运行时上的平台，用于构建高速、可伸缩的网络程序。Node.js采用事件驱动、非阻塞I/O模型，使它既轻量又高效，并成为构建运行在分布式设备的数据密集型实时程序的完美选择。”

我们在本章中会看到下面这些概念：

A. 为什么javascript对服务器开发很重要；

B. 浏览器如何用javascript处理I/O；

C. Node在服务端如何处理I/O；

D. DIRT程序是什么意思，为什么适用于Node开发；

E. 几个基础的Node程序示例。

我们先把视线转到JavaScript上…….

## 1.1 构建于JavaScript之上

无论好坏，javaScript都是世界上最流行的编程语言。只要你做过Web程序，就肯定遇到过JavaScript。JavaScript几乎遍布于Web上的每个角落，所以它已经实现了Java在20世纪90年代“一次编写，处处运行”的梦想。

在2005年Ajax革命前后，javaScript从一门“写着玩儿”的语言变成了一种被人们用来编写真正的、重要的程序的语言。这些程序中比较引人注目的先行者是Google地图和Gmail，但现在类似的web应用有一大堆，从Twitter到Facebook,再到GitHub。

自从2008年年末Google Chrome发布以来，得益于浏览器厂商的白热化竞争，JavaScript的性能以不可思议的速度得到大幅提升。现代化javaScript虚拟机的性能正改变着可以构建在Web上的应用类型。

在服务端编程，Node使用的是为Google Chrome提供的V8虚拟机。V8让Node在性能上得到了巨大的提升，因为它去掉了中间环节，执行的不是字节码，用的也不是解释器，而是直接编译成了本地机器码。Node在服务器使用JavaScript还有其他好处。

A. 开发人员用一种语言就能编写整个Web应用，这样可以减少开发客户端和服务器端时所需的语言切换。这样代码可以在客户端和服务端共享，比如在表单校验或游戏逻辑中使用同样一段代码。

B. JSON是目前非常流行的数据交换格式，并且还是JavaScript原生的。

C. 有些NoSql数据库中用的就是JavaScript语言（MongoDB），所以跟它们简直是天作之合。

D.JavaScript是一门编译目标语言，现在有很多可以编译成JavaScript的语言。

E. Node用的虚拟机（V8）会紧跟ECMAScript标准。换句话说，在Node中如果想用新的JavaScript语言特性，不用等到所有浏览器都支持。

JavaScript竟然成了一种引人瞩目的编写服务端应用的语言，之前谁能料到呢？基于前面提到的覆盖范围、性能和其他特性，node已经赚足了眼球。但JavaScript只是整幅拼图中的一块：node使用javaScript的方式更为有趣。为了理解Node环境，我们先看看你最熟悉的JavaScript环境：浏览器。

## 1.2 异步和事件触发：浏览器

Node为服务器端JavaScript提供了一个事件驱动的、异步的平台。它把JavaScript带到服务端中的方式跟浏览器把JavaScript带到客户端的方式几乎一模一样。了解浏览器的工作原理对我们了解Node的工作原理会有很大帮助。它们都是事件驱动和非阻塞的I/O处理。

我们来看一小段jQuery用XMLHttpRequest做Ajax请求的代码：

$.post(‘/resource.json’,function(data){

Console.log(data);

}};

这个程序会发送一个到resource.json的HTTP请求。当响应返回时会调用带着参数data的匿名函数，data就是从那个请求中得到的数据。

注意，代码没有写成下面这样：

Var data=$.post(‘/resource.json’);

Console.log(data);

在这个例子中，假定对resource.json的响应在准备好后会储存在变量data中，并且在此之前函数console.log不会执行。I/O操作会“阻塞”脚本继续执行，直到数据准备好。因为浏览器是单线程的，如果这个请求用了400ms才返回，那么页面上的其他时间都要等到那之后才能执行。可以想象一下，如果一副动画被停住了，或者用户试着跟页面交互时动不了，那种用户体验有多糟糕。

谢天谢地，实际情况不是这样的。当浏览器中有I/O操作时，该操作会在事件轮询的外面执行，然后当这个I/O操作完成时，它会发出一个“事件”，会有一个函数处理它。

这个I/O是异步的，并且不会“阻塞”脚本执行，事件轮询仍然可以响应页面上其他的交互或请求。这样，浏览器可以对客户做出响应，并且处理页面上的很多交互动作。

## 1.3 异步和事件触发：服务器

可能大多数人都了解传统的服务器编程的I/O模型，就像1.2 节那个“阻塞”的Jquery例子一样。下面是一个PHP的例子：

$result=mysql\_query(“select \* from myTable”);

Print\_r($result);

这段代码做了I/O操作，并且在所有数据回来之前，这个进程会被阻塞。对于很多程序而言，这个模型没什么问题，并且很容易理解。但有一点可能会被忽略：这个进程也有状态，或者说内存空间，并且在I/O完成之前基本什么也不会做。根据I/O操作的延迟情况，那可能会有10ms到几分钟的时间。延迟也可能是由下列意外情况引发的：

A. 硬盘正在执行维护操作，读/写都暂停了；

B. 因为负载增加，数据库查询变得更慢了；

C. 由于某种原因，今天从sitexyz.com拉取资源非常迟缓。

如果程序在I/O上阻塞了，当有更多请求过来时，服务器会怎么处理呢？在这种情景中通常会用多线程的方式。一种常见的实现是给每个连接分配一个线程，并为那些连接设置一个线程池。你可以把线程想象成一个计算工作区，处理器在这个工作区中完成指定的任务。线程通常都是处于进程之内的，并且会维护它自己的工作内存。每个线程会处理一到多个服务器连接。尽管这听起来是个很自然的委派服务器劳动力的方式，但程序内的线程管理会非常复杂。此外，当需要大量的线程处理很多并发的服务器连接时，线程会消耗额外的操作系统资源。线程需要CPU和额外的RAM来做上下文切换。

在Node中，I/O几乎总是在主事件轮询之外进行，使得服务器可以一直处于高效并且随时能够做出响应的状态，就像NGINX一样。这样进程就更加不会受I/O限制，因为I/O延迟不会拖垮服务器，或者像在阻塞方式下那样占用很多资源。因此一些在服务器上曾经是重量级的操作，在Node服务器上扔然可以是轻量级的。

这个混杂了事件驱动和异步的模型，几乎随处可用的JavaScript语言，帮我们打开了一个精彩纷呈的数据密集型实时程序的世界。