**[Node.js](http://lib.csdn.net/base/nodejs" \o "Node.js知识库" \t "_blank)**自出现之日起，就注定了它的不平凡。基于**[JavaScript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "_blank)**，可以让**[前端开发](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "_blank)**者不用学习任何一门后端语言即可开发后台server；基于异步事件回调和非阻塞IO，可以让开发者不用过分的担心很多同步的问题；高效而庞大的类库，可以让开发者将更多的经历放在项目开发，类库随查随用。

说了这么多，离本文的主题到底有什么关系？那我们从Web Server说起。

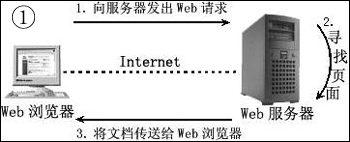
Web Server介绍

基本定义

Web Server，Web服务器，一般指网站服务器，是指驻留于因特网上某种类型计算机的程序，可以向浏览器等Web客户端提供文档，也可以放置网站文件，让全世界浏览；可以放置数据文件，让全世界下载。   
通俗一点可以理解为这样：   
\* Web服务器存放一些资源（文件、数据等），可以通过因特网的形式进行访问，资源的展现形式是Web Server内部确定好的。   
\* Web服务器维持可达性，即通过IP（域名）和端口可以访问到Web服务器，并获取Web Server提供的资源。

常见的Web Server有nginx、lighttpd，apache，IIS、jetty、Tomcat等。

流程介绍

Web Server同客户端之间通过某些协议进行数据交互，常见的有HTTP、FTP等。   
协议是客户端和服务器进行数据连接、交互的一种规范，只有按照这个规范才能解析相应的数据。   
一般的，客户端与服务器交互示意图如下：   


* 客户端发送一个Web请求，如在浏览器输入[http://www.sogou.com](http://www.sogou.com/" \t "_blank)
* 通过Internet传输，请求传递到Web服务器，Web服务器按照一定的协议（如http）对请求进行解析。
* Web服务器处理请求，寻找资源，如果成功找到资源，则将状态码及文档内容回传至客户端，如果未找到，则返回状态码及其他信息，告知客户端资源未找到，如常见的404页面。
* 客户端接收数据，进行相应的处理，如浏览器展现页面。

自定义Web Server

Nginx、Apache等服务器相对较为成熟，自有一套搭建流程。有时为了快速实现功能，将静态资源、动态脚本均通过一个自定义的Server运行起来，提供对外服务。自定义Server只需满足以下条件：

* 暴露端口，客户端通过IP和端口可以访问；
* 数据按照某种协议进行传输，可以使用通用协议（如http），也可以自定传输协议。
* 能够处理正常的请求并返回正确的数据；
* 对错误的请求有一定的容错性，如资源未找到，内部处理异常等。

下面我们回归本章节的主题，搭建一个[**node.js**](http://lib.csdn.net/base/nodejs)的Web Server。

Node.js准备工作

安装

不过多介绍，通过官网直接找到下载包，下载[**操作系统**](http://lib.csdn.net/base/operatingsystem)对应的安装包进行安装。

package.json

安装nodejs之后，通过npm命令初始化一个项目，该项目下回自动生成一个package.json的文件，该文件作为项目的核心文件，定义了项目的名称，版本号等信息，其中最重要的当数依赖包的定义。下面，截取了部门package.json文件内容：   

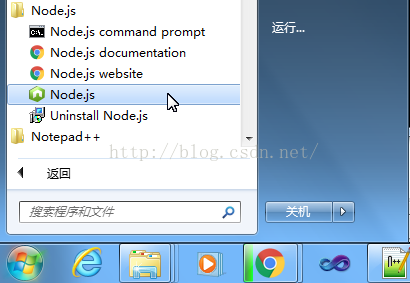

* name: 项目名称
* version: 版本号
* author：开发者
* description： 描述信息
* scripts: 定义了一些启动、测试的脚本
* dependencies: 项目的依赖包，格式按章包名 + 版本号，如上图
* Node.js是开源项目，它的创始人是Ryan Dahl。后来这个开源项目非常火爆，Joyent这个公司注意到了Node.js，决定赞助这个项目。Ryan Dahl于2010年加入该公司，全职负责Node.js项目的开发。
* 关于Node.js，后来因为Joyent的管理问题，几个重要的开发者出走，另立山头，开始了一个叫做io.js的开源项目，其社区一度非常活跃。不过，2015年9月15日，io.js和Node.js再度合体，Node.js v4.0.0发布了。

# 下载Node.js

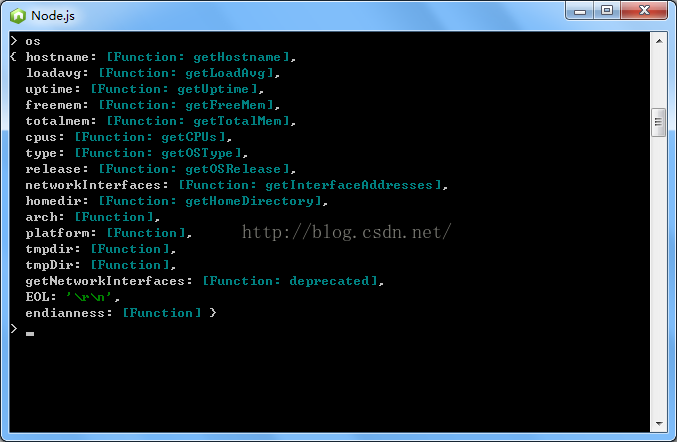
* 官网地址：https://nodejs.org/en/
* 您可以直接点击下图的链接进入：

[](https://nodejs.org/en/)

安装完成后，开始菜单里会添加一个Node.js的目录，如下图所示：



点击Node.js，打开一个命令行窗口，输入“os”回车试试



# 第一个Web服务器

接触的多了，发现好多语言进行Web开发的时候是可以基于语言本身自己写Web服务器的!

[**Go**](http://lib.csdn.net/base/go)语言如此，**[Python](http://lib.csdn.net/base/python" \o "Python知识库" \t "_blank)**语言如此，Node.js也是如此；

但通常我们在项目开发时可能不会这样做，因为通常已经有比较完善的服务器框架可以使用了。

将以下代码保存为“helloworld.js”文件：

**[javascript]** [view plain](http://blog.csdn.net/testcs_dn/article/details/50532775) [copy](http://blog.csdn.net/testcs_dn/article/details/50532775)

1. **var** http = require('http');
3. **var** server = http.createServer(**function**(request, response) {
4. response.end("hello world!");
5. });
7. server.listen(8080);

一个Web服务器就这么几行代码吗？

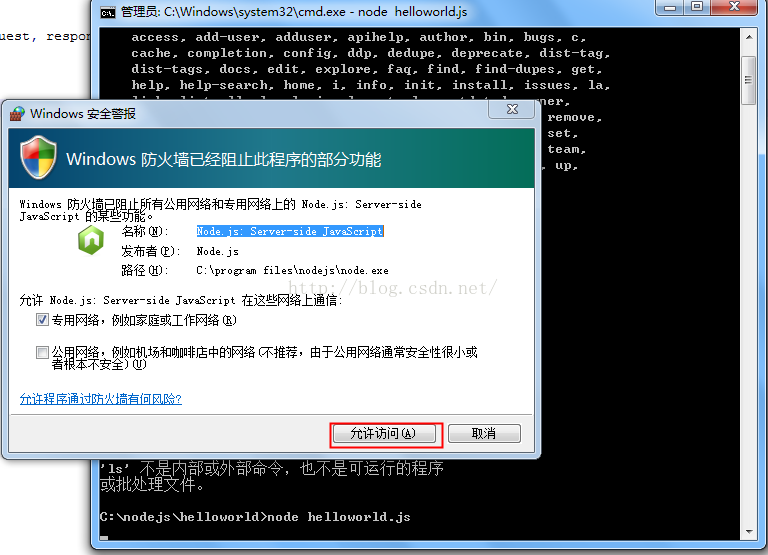
对，没错!！不过这是一个比较弱智的Web服务器，它只会返回“hello world!”。

require对于很多前端小伙伴应该是再熟悉不过了吧？

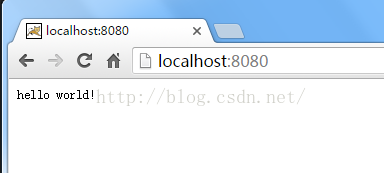
**只要会js就行了，你就可以成为Web开发前端+后端的高手**！

打开命令行，切换到“helloworld.js”文件的保存目录，执行：node helloworld.js

如果你开启了防火墙，应该会看到下图：



点击允许访问，打开浏览器，输入“<http://localhost:8080/>”访问看看吧！



在浏览器以外运行 JavaScript 对于 JavaScript 爱好者来说非常神奇，同时也肯定是web 应用程序开发界最受欢迎的进步之一。全球各地的开发者张开双臂拥抱NodeJS。

对于新手来说，Node.js 是 JavaScript 运行时，允许你在浏览器之外服务器上运行 JavaScript 代码。它是基于谷歌 Chrome 的 V8 虚拟机引擎建立的，V8 虚拟机引擎用于在 Chrome 浏览器中支持 JavaScript。

NodeJS 逐渐普及的步伐是因为它可以让你只使用 JavaScript 就可以建立大规模、实时性、可扩展的移动和 Web 应用程序。

随着节点生态系统的增长，框架也开始着手加快工作流程。有许多的 Node.js 框架，允许你构建实时的端到端网络应用，而无需任何其他第三方 Web 服务器、应用服务器、工具和技术。

通用性的 Node 框架就像快递，Koa 和 Hapi 更加灵活，让你做你想要的一切，最大限度的满足开发需求。但是，在最初你需要投入更多的努力，更加依靠开发者作出正确的决策。

而另一方面，像 Mean.io，Meteor, Derby 和 Mojito，有自己的标准结构和技术体系，灵活性较差。然而这些框架不会给开发者太多做错误决策的空间。

让我们来看看目前可用的最好、最强大的 NodeJS 框架，并帮助你建立实时的、各种规模和复杂性的可扩展 Web 应用程序 -

### 1. [Node.js Express](http://www.oschina.net/p/expressjs)

对于一个已经在使用 node.js 的开发人员来说，Express 或者"node.js express"并不是一个新鲜事。Express框架提供了对node.js原生API的比较好的封装，从而使开发者更加容易地使用node.js。

Express 框架提供了用来开发强壮的 web/移动应用，以及 API 的所有功能。并且开发人员还能够方便地为它开发插件和扩展，从而增加 Express 的能力。下面列出了一些 Express 提供的基本的功能：

- 可以和任何的第三方数据库进行通讯

- 可以使用任何的用户认证方式

- 可以使用任何符合 Express 接口定义的模板引擎

- 可以按照需要定义工程目录

通过使用 Node Express，你可以使用更少的代码来实现功能。至少通过使用 Node Express，你可以实现中间件来响应 http 请求，可以定义路由表来定义对不同请求的响应函数，还可以使用模板引擎来输出 html 页面。

对于一个 NodeJS 开发新手来说，Express 还提供了如下的好处：1）Express的学习曲线并不陡峭，可以很快上手；2）Express 有非常庞大的社区，和组织良好的文档，新手可以很容易得到所需要的一切。

Express 根据 MIT 协议进行开源，目前 StrongLoop 对它提供支持。

你可以通过 Express 的官方网站 [expressjs.com](http://expressjs.com/)获得更多的信息。

### 2. [Sails.js](http://www.oschina.net/p/sailsjs)(node.js mvc)

Sails 作为一个非常稳固的 nodejs 框架，提供了建立任何规模的 web 应用所需要的所有功能功能。

Sails.js 在底层使用了 express框架来提供对 http 请求的处理，同时使用 Socket.IO 框架来处理WebSocket 请求。同时作为一个前端应用开发框架，它允许开发人员选择他/她熟悉的技术来开发应用。

同时 Sails.js 也通过 waterline 框架实现了 ORM 功能。通过这个功能，你的应用程序可以在不进行大的修改的前提下，就可以从一个后端数据库，切换到另外一个后端数据库（也可以是一个NoSQL数据库）。

Sails 特别适合用来开发对数据的实时更新有较高要求的应用，比如多人棋类游戏，单页web应用等等。如果你对 Ruby, Django 或者 Zend 有一定的了解，那么你将非常容易理解Sail中的概念。

简单来说，Sails.js 既给开发者提供了一个优秀的 MVC 框架，也提供了一定的灵活性，让开发者可以自主选择前端开发方式和后端的数据库。 Sails.js 是由 Mike McNeil 创建的，现在由 Treeline and balderdash 提供支持。Sails.js 在 MIT 协议下开源。

你可以在网站 [sailsjs.org](http://sailsjs.org/) 上找到更多关于这个框架的内容。

### 3. [KOA](http://www.oschina.net/p/koa)

KOA 是 node.js mvc 框架的后起之秀，在2013第四个季度才发布了第一个版本。开发 KOA 的人员基本来自 Express 开发团队，TJ Holowaychuk 是 KOA 开发团队的领导者。虽然 KOA 大部分开发人员来自 Express，但是他们使用了完全不同的技术来开发 KOA，并且 KOA 正成为 Express 一个强有力的竞争对手。

KOA 框架的核心是 ES6 的 generator。KOA 使用 generator 来实现中间件的流程控制，使用try/catch 来增强异常处理，同时在 KOA 框架中你再也看不到复杂的 callback 回调了。

KOA 框架本身非常小，只打包了一些必要的功能，但是它本身通过良好的模块化组织，让开发人员可以按照自己的想法来实现一个扩展性非常好的应用。

许多 JavaScript/Node.js 的忠实开发者都开始选择使用 KOA 来开发新的项目，因为 KOA 提供了更多的灵活性开发应用程序。

你可以通过[koajs.com](http://koajs.com/)获取更多的信息。

### 4. [Meteor](http://www.oschina.net/p/meteor)

Meteor 框架是 Node.js 上最出色的全栈框架。项目在 GitHub 上有 28K+ 的赞，拥有大量的自定义包，庞大的社区支持，非常好的教程和文档。在这个领域 Meteor 毫无疑问是王者，你可以用它构建纯 Javascript 的实时 Web 和 手机应用。

Meteor 最优秀的部分是，无论是服务器端的数据库访问，商业逻辑实现，还是客户端的展示，所有的流程都是无缝连接，开箱即用。整个框架使用统一的 API，Meteor API 同时适用于客户端和服务器端。

它使用的 DDP 协议可以让你在后端连接简单的数据库服务、企业数据仓库、甚至 IOT 传感器。Meteor 带有自己默认的栈，但又有足够的灵活性，可以让你选择自己的技术方案。如果你不需要尝试其他的框架或者没有其他的条件限制，你可以直接使用默认配置，进行快速地应用开发。

Meteor 拥有专业化的开发团队，顶级风投的大量资金支持，这都让 Meteor 能够时刻保持业界领先。

你可以通过 [meteor.com](https://www.meteor.com/) 网站进一步了解 Meteor。

### 5. [Derby.js](http://www.oschina.net/p/derbyjs)

Derby.JS 跟它的直接竞争对手 Meteor、Mean.io、以及 Mojito 一样，也是一个全栈框架。它运行在 Nodejs + mongo + Redis 的上层。Derby 主要部分是一个叫做 Racer 的数据同步引擎，它能够让数据在数据库、服务器和浏览器之间的同步变得轻而易举。

Racer 的确能够让基于 Derby 框架的应用运行地更快，无论是在浏览器端还是服务器端，对于单页面应用来说，它都是一个完美的选择方案。Derby 经常被用来和业界老大 Meteor 进行比较，Meteor 项目已经开发了一段很长时间，因而能够提供更多的开箱即用的功能，使得在更短时间内开发复杂的 web 应用变得更加容易。

而 Derby 更适合于需要更快运行速度的应用，并且它的模块化方式能够让应用更灵活，更容易扩展。Derby 最近的发展有些缓慢，但它并没有出局，仍有改写 Nodejs 全栈框架游戏规则的潜力。

你可以通过 [derbyjs.com](http://derbyjs.com/) 网站进一步了解 Derby。

### 6. [Flatiron.js](http://www.oschina.net/p/flatiron)(Node.js MVC 框架)

Flatiron 框架背后的核心思想是让你能使用它所提供的组件以及一些第三方库构建你自己的全栈框架。很酷不是吗？我个人十分的喜欢这种方式。然而，这带来的是更高的复杂度，并有可能会被使用错误组件的开发者搞得一团糟。

你能称之为一个由多个相互独立的组件松散地组建起来的全栈 MVC 框架。Flatiron 框架支持Director，一个从头到脚都使用JavaScript 搭建起来的，并不需要任何依赖项的 URL 路由组件。

通过一个叫 Plates 的模板引擎，Flatiron 能够支持模版语言，然而数据管理是通过 json 实现的，并能与任何一种数据库一起使用。Flatiron 现在由 Nodejitsu 以及其他的社区成员在进行维护，并做的相当不错，是一个不那么流行却值得一看的框架。

你能在 [flatironjs.org](http://flatironjs.org/) 上获得更多信息。

### 7. [Hapi](http://www.oschina.net/p/hapijs" \t "_blank)

Hapi 是为数不多的不依赖于 Express 的 node.js 框架，现在甚至已经完全独立于 Express 了。在最近一段时间中，很多开发者选择了 Hapi 而非 Express，这使得它或多或少变为了 Express 的竞争对手。

Hapi 在众多 nodejs 的框架中并非一个老牌选手，然而它却成功的在这当中创造了自己的一个生态圈。Hapi 致力于完全的分离 node HTTP 服务器、路由以及业务逻辑，并更多的聚焦于如何尽可能的通过配置而非代码来控制东西。

Hapi 最初是由 Eran Hammer 以及在 Walmart labs 的团队为了工作需要开发的。其后便以极快的速度受到了欢迎，现在已在 MIT 许可下成为一个开源的框架，能够免费的被下载和使用。

迪士尼、雅虎、Pebble、beats 音乐以及 Walmart 这样的公司都在使用 HAPI 作为他们旗下一个或多个项目的网络应用框架，它的影响力便可见一斑了。

你能在 [hapijs.com](http://hapijs.com/)上找到更多关于 Hapi 的信息。

### 8. [Mean.IO](http://www.oschina.net/p/mean)

Mean 是 Mongo DB，Express，Angular 和 Node.js 捆绑在一起的组合。基本上说只要有它，你就拥有了数据库层，服务器端和网页前端的整套工具，足以开发所有类型的现代网络应用。

Mean 是一个完整独立的包，它涵盖了应用开发的所有方面。尤其适合于那些需要快速开始开发的人。它内置多种技术而且在联合使用时变现非常好。可以用于创建任意大小和复杂度的应用。

使用 Mean，开发者可以避免经历混合和匹配不同的技术栈。通过 mean 栈，你可以减少安装和配置MongoDB，Express，Angular 和 Node.js需要的时间。Mean.io 的另一个巨大好处就是所有的栈都使用 JavaScript，服务器端 Express 对MongoDB 的访问(json)和通多 Angular 从 Node 到客户端。

在这里 - [mean.io](http://mean.io/)了解更多Mean.io的相关信息。

还有一个名为 mean.js 的 mean 分支也相当流行。

### 9. [Mojito](http://www.oschina.net/p/yahoo-mojito" \t "_blank)

Mojito 由 Yahoo 开发并迅速取得成功。然而很快又带着关于框架的空前的成功坐到了冷板凳，就像Meteor 和 Mean stack 那样。

Mojito 同样是一个 MVC 应用框架，非常适合于创建使用 HTML5，JavaScript 和 CSS3 的高性能的网络和手机应用。Mojito 的根本目标是提供一个框架，该框架用于构建标准的基于跨平台的应用。使之可以同时运行在客户端和服务器端，并实现高性能。

你可以在 Yahoo 开发者网页—— [mojito](https://developer.yahoo.com/cocktails/mojito/" \t "_blank) 获得更多信息。

### 10. [Socket Stream](http://www.oschina.net/p/socketstream)

SocketStream 是一个有趣的框架，专注于客户端和服务端数据的快速同步，它致力于前后端数据的实时更新。

它最大的特点是不严格要求你使用指定的客户端技术，也不限定数据库的 ORM。我趋向于将它和有同样功能的兄弟项目 Sail.js 做比较，它更适合做单页 web 应用，多用户游戏，聊天客户端，网络应用，交易平台以及所有的需要将数据从服务端实时推送到客户端的应用。

服务端和客户端使用 JSON 来传输数据，比较理想的是使用 websockets 在服务端事件发生时自动将数据推送到客户端，Socket stream 是由 Owen Barnes 创建，现在由 Paul Jensen 和团队维护，他们的工作让这个框架得到了应有的荣耀。

SocketStream framework 在最近几月获得了很好的发展，未来一片光明。

更多 SocketStream 信息请浏览 github 上的 - [socketstream](https://github.com/socketstream/socketstream" \t "_blank)。

其它类似优秀框架还有：[total.js](http://www.oschina.net/p/total-js), [Geddy.JS](http://www.oschina.net/p/geddy), [Locomotive](http://www.oschina.net/p/locomotive), compound 和 [Restify](http://www.oschina.net/p/restify" \t "_blank)。

Web 和应用开发的风景变化是非常快速的，开发人员转向快速框架和清洁项目交付（clean project delivery）。使用 node 框架的最大优势是提供了高层级结构的盒子，你可以关注扩展您的应用程序而不是努力在建设上支出。

框架提供了多样的特性，工作在不同的底层，试图解决构建实时的常见问题，并解决了可伸缩的和复杂的 Web 应用程序在速度上的问题。在这篇文章里讨论的框架是当今市场上最好的 node.js 框架。

# 使用 Node.js 的优势和劣势都有哪些？

Node.js其实有很多大公司都在用的，比如eBay, Microsoft, 你可以去Node.js官网看看：[node.js](https://link.zhihu.com/?target=http%3A//nodejs.org/" \t "_blank)  
要想知道更详细的列表，可以看这里：[https://github.com/joyent/node/wiki/Projects,-Applications,-and-Companies-Using-Node](https://link.zhihu.com/?target=https%3A//github.com/joyent/node/wiki/Projects%2C-Applications%2C-and-Companies-Using-Node" \t "_blank)

再稍微花点时间，搜集一些网上资料回答第一个问题：优势和劣势。  
要讲清楚这个问题，先讲讲整个Web应用程序架构（包括流量、处理器速度和内存速度）中的瓶颈。瓶颈在于服务器能够处理的并发连接的最大数量。Node.js解决这个问题的方法是：更改连接到服务器的方式。每个连接发射一个在Node.js引擎的进程中运行的事件，而不是为每个连接生成一个新的OS线程（并为其分配一些配套内存）。Node.js不会死锁，因为它根本不允许使用锁，它不会直接阻塞 I/O 调用。Node.js还宣称，运行它的服务器能支持数万个并发连接。

Node本身运行V8 JavaScript。V8 JavaScript引擎是Google用于其Chrome浏览器的底层JavaScript引擎。Google使用V8创建了一个用C++编写的超快解释器，该解释器拥有另一个独特特征：您可以下载该引擎并将其嵌入任何应用程序。V8 JavaScript引擎并不仅限于在一个浏览器中运行。因此，Node.js实际上会使用Google编写的V8 JavaScript引擎，并将其重建为可在服务器上使用。

**Node.js优点：**  
1、采用事件驱动、异步编程，为网络服务而设计。其实Javascript的匿名函数和闭包特性非常适合事件驱动、异步编程。而且JavaScript也简单易学，很多前端设计人员可以很快上手做后端设计。  
2、Node.js非阻塞模式的IO处理给Node.js带来在相对低系统资源耗用下的高性能与出众的负载能力，非常适合用作依赖其它IO资源的中间层服务。3、Node.js轻量高效，可以认为是数据密集型分布式部署环境下的实时应用系统的完美解决方案。Node非常适合如下情况：在响应客户端之前，您预计可能有很高的流量，但所需的服务器端逻辑和处理不一定很多。

**Node.js缺点**：  
1、可靠性低  
2、单进程，单线程，只支持单核CPU，不能充分的利用多核CPU服务器。一旦这个进程崩掉，那么整个web服务就崩掉了。

不过以上缺点可以可以通过代码的健壮性来弥补。目前Node.js的网络服务器有以下几种支持多进程的方式：

#1 开启多个进程，每个进程绑定不同的端口，用反向代理服务器如 Nginx 做负载均衡，好处是我们可以借助强大的 Nginx 做一些过滤检查之类的操作，同时能够实现比较好的均衡策略，但坏处也是显而易见——我们引入了一个间接层。  
#2 多进程绑定在同一个端口侦听。在Node.js中，提供了进程间发送“文件句柄” 的功能，这个功能实在是太有用了（貌似是yahoo 的工程师提交的一个patch） ，不明真相的群众可以看这里： [Unix socket magic](https://link.zhihu.com/?target=http%3A//www.lst.de/%7Eokir/blackhats/node121.html)  
#3 一个进程负责监听、接收连接，然后把接收到的连接平均发送到子进程中去处理。

在Node.js v0.5.10+ 中，内置了cluster 库，官方宣称直接支持多进程运行方式。Node.js 官方为了让API 接口傻瓜化，用了一些比较tricky的方法，代码也比较绕。这种多进程的方式，不可避免的要牵涉到进程通信、进程管理之类的东西。  
此外，有两个Node.js的module：multi-node 和 cluster ，采用的策略和以上介绍的类似，但使用这些module往往有一些缺点：  
#1 更新不及时  
#2 复杂庞大，往往绑定了很多其他的功能，用户往往被绑架  
#3 遇到问题难以解决

Node表现出众的典型示例包括：

1、RESTful API  
提供RESTful API的Web服务接收几个参数，解析它们，组合一个响应，并返回一个响应（通常是较少的文本）给用户。这是适合Node的理想情况，因为您可以构建它来处理数万条连接。它仍然不需要大量逻辑；它本质上只是从某个数据库中查找一些值并将它们组成一个响应。由于响应是少量文本，入站请求也是少量的文本，因此流量不高，一台机器甚至也可以处理最繁忙的公司的API需求。

2、Twitter队列  
想像一下像Twitter这样的公司，它必须接收tweets并将其写入数据库。实际上，每秒几乎有数千条tweet达到，数据库不可能及时处理高峰时段所需的写入数量。Node成为这个问题的解决方案的重要一环。如您所见，Node能处理数万条入站tweet。它能快速而又轻松地将它们写入一个内存排队机制（例如memcached），另一个单独进程可以从那里将它们写入数据库。Node在这里的角色是迅速收集tweet，并将这个信息传递给另一个负责写入的进程。想象一下另一种设计（常规PHP服务器会自己尝试处理对数据库本身的写入）：每个tweet都会在写入数据库时导致一个短暂的延迟，因为数据库调用正在阻塞通道。由于数据库延迟，一台这样设计的机器每秒可能只能处理2000条入站tweet。每秒处理100万条tweet则需要500个服务器。相反，Node能处理每个连接而不会阻塞通道，从而能够捕获尽可能多的tweets。一个能处理50000条tweet的Node机器仅需20台服务器即可。

3、电子游戏统计数据  
如果您在线玩过《使命召唤》这款游戏，当您查看游戏统计数据时，就会立即意识到一个问题：要生成那种级别的统计数据，必须跟踪海量信息。这样，如果有数百万玩家同时在线玩游戏，而且他们处于游戏中的不同位置，那么很快就会生成海量信息。Node是这种场景的一种很好的解决方案，因为它能采集游戏生成的数据，对数据进行最少的合并，然后对数据进行排队，以便将它们写入数据库。使用整个服务器来跟踪玩家在游戏中发射了多少子弹看起来很愚蠢，如果您使用Apache这样的服务器，可能会有一些有用的限制；但相反，如果您专门使用一个服务器来跟踪一个游戏的所有统计数据，就像使用运行Node的服务器所做的那样，那看起来似乎是一种明智之举。

总的来说，Node.js的应用场景

1) 适合

JSON APIs——构建一个Rest/JSON API服务，Node.js可以充分发挥其非阻塞IO模型以及JavaScript对JSON的功能支持(如JSON.stringfy函数)  
单页面、多Ajax请求应用——如Gmail，前端有大量的异步请求，需要服务后端有极高的响应速度  
基于Node.js开发Unix命令行工具——Node.js可以大量生产子进程，并以流的方式输出，这使得它非常适合做Unix命令行工具  
流式数据——传统的Web应用，通常会将HTTP请求和响应看成是原子事件。而Node.js会充分利用流式数据这个特点，构建非常酷的应用。如实时文件上传系统transloadit  
准实时应用系统——如聊天系统、微博系统，但Javascript是有垃圾回收机制的，这就意味着，系统的响应时间是不平滑的(GC垃圾回收会导致系统这一时刻停止工作)。如果想要构建硬实时应用系统，Erlang是个不错的选择

2) 不适合

CPU使用率较重、IO使用率较轻的应用——如视频编码、人工智能等，Node.js的优势无法发挥  
简单Web应用——此类应用的特点是，流量低、物理架构简单，Node.js无法提供像Ruby的Rails或者Python的Django这样强大的框架  
NoSQL + Node.js——如果仅仅是为了追求时髦，且自己对这两门技术还未深入理解的情况下，不要冒险将业务系统搭建在这两个漂亮的名词上，建议使用MySQL之类的传统数据库

如果系统可以匹配Node.js的适用场景，那么是时候采取具体的措施来说服老板了。

说服自己老板采用Node.js的方式

构建一个简单的原型——花一周时间构建系统某一部分的原型是非常值得的，同时也很容易和老板在某一点达成一致，等到系统真的在某一部分应用了Node.js，就是打开局面的时候  
寻找开发者——首先JavaScript语言的普及度很高，一般公司都不乏Web前端工程师，而此类工程师的学习门槛也非常低。这就意味着Node.js很容易招人，或者公司就隐藏了一些高手  
强大的社区支持——Node.js社区非常活跃，吸引很多优秀的工程师，这就意味着公司可以很容易从社区得到免费或者付费的支持  
系统性能考虑——JavaScript引擎Google V8，加之原生异步IO模型，使得Node.js在性能的表现非常出色，处理数以千计的并发请求非常轻松  
专业公司的支持——使用开源技术的最大问题是，原作者不承诺对其产品进行技术支持或者质量保证。现在Node.js已经得到Joyent公司的赞助，这就保证了未来Node.js的发展是可持续性的