http://www.runoob.com/nodejs

简单的说 Node.js 就是运行在服务端的 JavaScript

查看node版本：$ node -v

通过node命令运行node.js程序

Node.js的组成

**引入 required 模块：**我们可以使用 **require** 指令来载入 Node.js 模块

**创建服务器：**服务器可以监听客户端的请求，类似于 Apache 、Nginx 等 HTTP 服务器

**接收请求与响应请求** 服务器很容易创建，客户端可以使用浏览器或终端发送 HTTP 请求，服务器接收请求后返回响应数据

创建node.js应用

第一步：引入required 模块

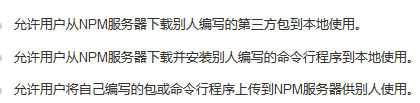
我们使用 **require** 指令来载入 http 模块，并将实例化的 HTTP 赋值给变量 http

第二步：创建服务器





NPM是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题，常见的使用场景有以下几种



由于新版的nodejs已经集成了npm，所以之前npm也一并安装好了。同样可以通过输入 **"npm -v"** 来测试是否成功安装。命令如下，出现版本提示表示安装成功:



Node.js 异步编程的直接体现就是回调。

异步编程依托于回调来实现，但不能说使用了回调后程序就异步化了。

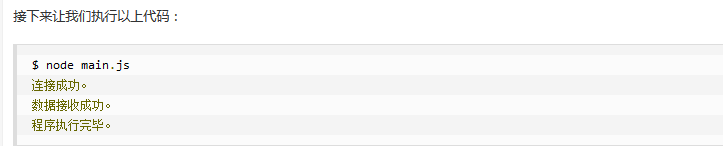
回调函数在完成任务后就会被调用，Node 使用了大量的回调函数，Node 所有 API 都支持回调函数

阻塞代码与非阻塞代码

以上两个实例我们了解了阻塞与非阻塞调用的不同。第一个实例在文件读取完后才执行完程序。 第二个实例我们不需要等待文件读取完，这样就可以在读取文件时同时执行接下来的代码，大大提高了程序的性能。

因此，阻塞是按顺序执行的，而非阻塞是不需要按顺序的，所以如果需要处理回调函数的参数，我们就需要写在回调函数内

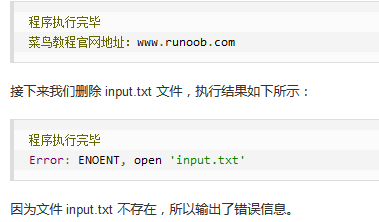
Node程序是如何工作的

在 Node 应用程序中，执行异步操作的函数将回调函数作为最后一个参数，而 回调函数接收错误对象作为第一个参数。

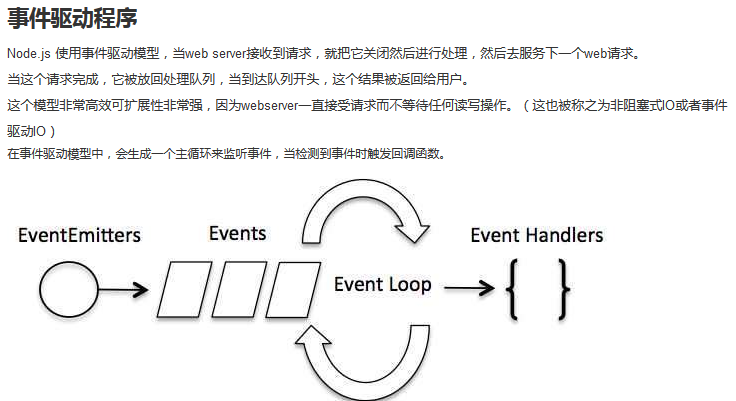


以上程序中 fs.readFile() 是异步函数用于读取文件。 如果在读取文件过程中发生错误，错误 err 对象就会输出错误信息。如果没发生错误，readFile 跳过 err 对象的输出，文件内容就通过回调函数输出。

执行以上代码，执行结果如下：



Node.js 事件循环

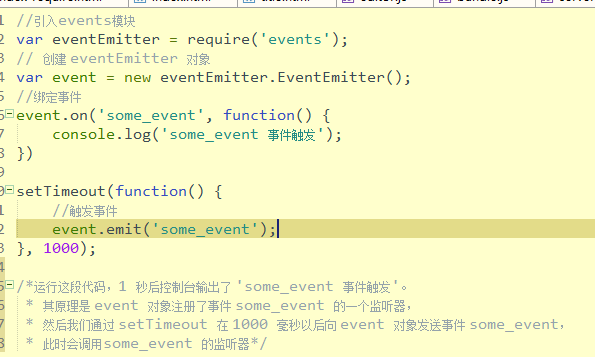
事件就是需要 **eventEmitter.on** 去绑定一个事件 通过 **eventEmitter.emit** 去触发这个事件其次说的是 事件的 **接收** 和 **发生** 是分开的Node.js 有多个内置的事件，我们可以通过引入 events 模块，并通过实例化 EventEmitter 类来绑定和监听事件

Node.js EventEmitter

Node.js 所有的异步 I/O 操作在完成时都会发送一个事件到事件队列。Node.js里面的许多对象都会分发事件：一个net.Server对象会在每次有新连接时分发一个事件， 一个fs.readStream对象会在文件被打开的时候发出一个事件。 所有这些产生事件的对象都是 events.EventEmitter 的实例

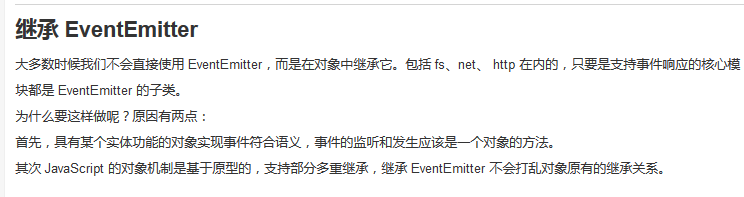
## EventEmitter 类

events 模块只提供了一个对象： events.EventEmitter。EventEmitter 的核心就是事件触发与事件监听器功能的封装。你可以通过require("events");来访问该模块EventEmitter 对象如果在实例化时发生错误，会触发 error 事件。当添加新的监听器时，newListener 事件会触发，当监听器被移除时，removeListener 事件被触发



EventEmitter 的每个事件由一个事件名和若干个参数组成，事件名是一个字符串，通常表达一定的语义。对于每个事件，EventEmitter 支持 若干个事件监听器。当事件触发时，注册到这个事件的事件监听器被依次调用，事件参数作为回调函数参数传递。让我们以下面的例子解释这个过程：

Node.js模块系统

为了让Node.js的文件可以相互调用，Node.js提供了一个简单的模块系统。 模块是Node.js 应用程序的基本组成部分，文件和模块是一一对应的。换言之，一个 Node.js 文件就是一个模块，这个文件可能是JavaScript 代码、JSON 或者编译过的C/C++ 扩展。

在 Node.js 中，创建一个模块非常简单，如下我们创建一个 **main.js** 文件，代码如下:

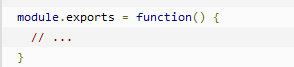


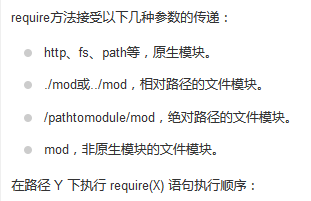
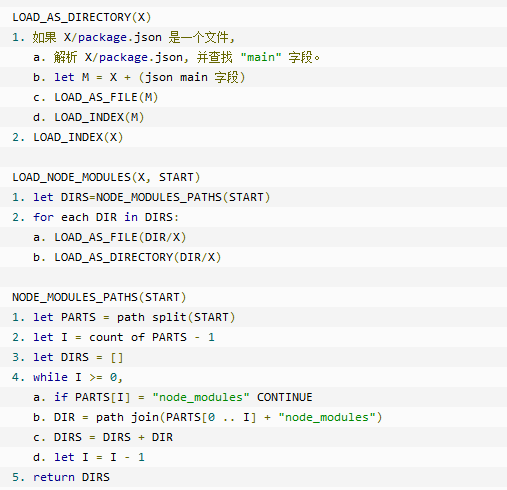
以上实例中，代码 require('./hello') 引入了当前目录下的 hello.js 文件（./ 为当前目录，node.js 默认后缀为 js）。

Node.js 提供了 exports 和 require 两个对象，其中 exports 是模块公开的接口，require 用于从外部获取一个模块的接口，即所获取模块的 exports 对象。

接下来我们就来创建 hello.js 文件，代码如下：



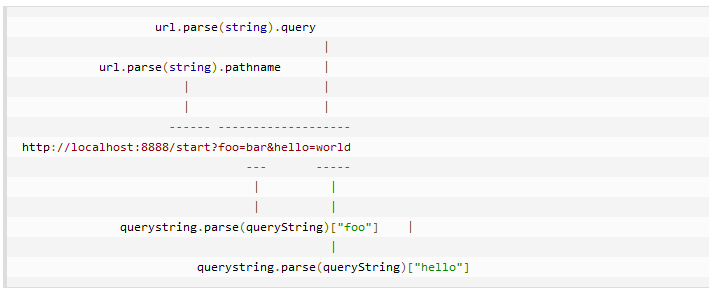
在以上示例中，hello.js 通过 exports 对象把 world 作为模块的访问接口，在 main.js 中通过 require('./hello') 加载这个模块，然后就可以直接访 问 hello.js 中 exports 对象的成员函数了。有时候我们只是想把一个对象封装到模块中，格式如下： 

Node.JS url 和 querystring 模块。

Node.js路由

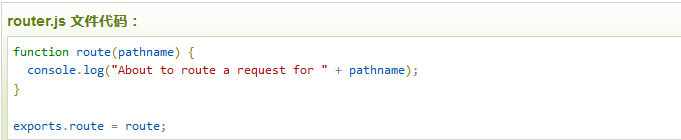
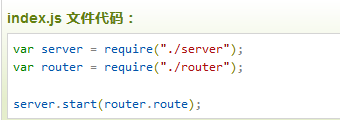
我们的请求数据存在request 对象中，该对象作为 onRequest() 回调函数的第一个参数传递。但是为了解析这些数据，我们需要额外的 Node.JS 模块，它们分别是 url 和 querystring 模块。



当然我们也可以用 querystring 模块来解析 POST 请求体中的参数，稍后会有演示。

现在我们来给 onRequest() 函数加上一些逻辑，用来找出浏览器请求的 URL 路径：



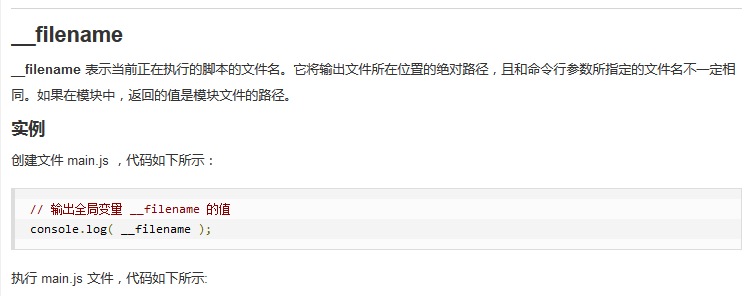


Node.js的全局对象JavaScript 中有一个特殊的对象，称为全局对象（Global Object），它及其所有属性都可以在程序的任何地方访问，即全局变量。在浏览器 JavaScript 中，通常 window 是全局对象， 而 Node.js 中的全局对象是 global，所有全局变量（除了 global 本身以外）都是 global 对象的属性。在 Node.js 我们可以直接访问到 global 的属性，而不需要在应用中包含它

当你定义一个全局变量时，这个变量同时也会成为全局对象的属性，反之亦然。需要注 意的是，在 Node.js 中你不可能在最外层定义变量，因为所有用户代码都是属于当前模块的， 而模块本身不是最外层上下文。

**注意：** 永远使用 var 定义变量以避免引入全局变量，因为全局变量会污染 命名空间，提高代码的耦合风险

以下是全局变量的属性值的访问







# Node.js GET/POST请求

