# 七天学会NodeJS

**参考资料：**[**http://nqdeng.github.io/7-days-nodejs/**](http://nqdeng.github.io/7-days-nodejs/)

## NodeJS基础

### 什么是NodeJS

JS是脚本语言，脚本语言都需要一个解析器才能运行。对于卸载HTML页面的JS，浏览器充当了解析器的角色，而对于需要独立运行的JS，NodeJS就是一个解析器。

每一种解析器都是一个运行环境，不但允许JS定义各种数据结构，进行各种运算，还允许JS使用运行环境提供的内置对象和方法做一些事情。例如运行在浏览器中的JS的用途是操作DOM，浏览器就提供了document之类的内置对象。而运行在NodeJS中的JS的用途是操作磁盘文件或搭建HTTP服务器，NodeJS就相应提供了fs、http等内置对象。

### NodeJS有啥用处

除了可以直接运行JS文件之外，NodeJS还有什么其他用处。

NodeJS的作者说，他创造NodeJS的目的是为了实现高性能的web服务器，他首先看重的是事件机制和异步IO的优越性，而不知JS。但是他需要选择一种编程语言实现他的想法，这种编程语言不能自带IO功能，并且需要能良好支持事件机制。JS没有自带IO功能，天生就是用于处理浏览器中的DOM事件，并且拥有一大群程序员，因此首选。

对于前端而言，虽然不是人人都要拿NodeJS写一个服务器，但简单可至使用命令交互模式调试JS代码片段，复杂可至编写工具提升工具效率。

### 如何安装

#### 安装程序

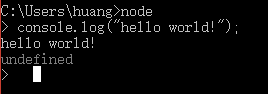
NodeJS提供了安装程序，可以在[官网](https://nodejs.org/en/download/)下载并安装。

参考：[windows 下安装nodejs](https://jingyan.baidu.com/article/b0b63dbfca599a4a483070a5.html)

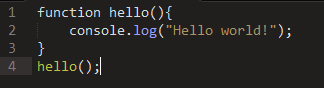
#### 编译安装

### 如何运行

打开终端，键入node进入命令交互模式，可以输入一条代码语句后立即执行并显示结果，例如：



如果要运行一大段代码的话，可以先写一个JS文件再运行，例如有一下hello.js



写好后在终端键入node hello.js运行，结果如下：



#### 权限问题

在Linux系统下，使用NodeJS监听80或者443断开提供http服务需要root权限，有两种方式可以做到。

一种方式是使用sudo命令运行NodeJS

一种方式是使用chmod +s命令让NodeJS总是以root权限运行。

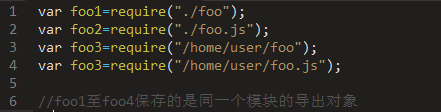
### 模块

在编写稍大一点的程序时一般都会将代码模块化。在NodeJS中，一般将代码合理拆分到不同的JS文件中，每一个文件就是一个模块，而文件路径就是模块名。

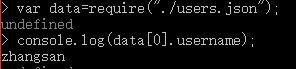
在编写每个模块的时候都有require，exports，module三个预先定义好的变量可供使用。

#### require

require函数用于在当前模块中加载和使用别的模块，传入一个模块名，返回一个模块导出对象。模块名可使用相对路径，或者是绝对路径。另外，模块名中的.js扩展名可以省略。例如：



另外，可以使用一下方式加载和使用一个json文件。



#### exports

Exports对象是当前模块的导出对象，用于导出模块公有方法和属性。别的模块通过require函数使用当前模块时得到的就是当前模块的exports对象。

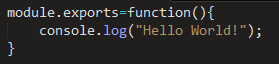
以下例子中导出了一个公有方法。



#### module

通过module对象可以访问到当前模块的一些相关信息，但最多的用途就是替换当前模块的导出对象。

例如模块导出对象默认是一个普通对象，如果想改成一个函数的话，可以使用一下方式。



以上代码中，模块默认导出对象替换为一个函数。

#### 模块初始化

一个模块中的JS代码仅在模块第一次被使用的时候执行一次，并在执行过程中初始化模块的导出对象。之后，缓存起来的导出对象被重复使用。

#### 主模块

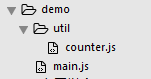
通过命令行参数传递给NodeJS以启动程序的模块被称为主模块。主模块负责调度组成整个程序的其他模块完成工作。

例如，通过以下命令启动程序时，main.js就是主模块。

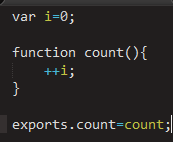


#### 完整示例

例如有如下目录。

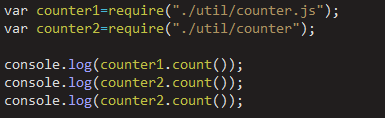


其中counter.js内容如下

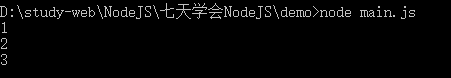


该模块定义了一个私有变量i，并在exports对象导出了一个公有方法count。

主模块main.js内容如下



运行



可以看到，counter.js并没有因为被require了两次而初始化两次。

### 小结

* NodeJS是一个JS脚本解析器，在任何操作系统下安装NodeJS本质上做的事情都是把NodeJS执行程序复制到一个目录，然后保证这个目录在系统path环境变量下，以便终端下可以使用node命令。
* 终端下直接输入node命令可进入命令交互模式，很适合用来测试一些JS代码片段，比如正则表达式。
* NodeJS使用CMD模块系统，主模块作为程序入口点，所有模块在执行过程中只初始化一次。
* 除非JS模块不能满足需求，否则不要轻易使用二进制模块，否则你的用户会叫苦连天。

## 代码的组织和部署

### 模块的路径解析规则

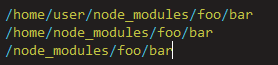
require函数支持斜杠/或盘符C:开头的绝对路径，也支持./开头的相对路径。但这两种路径在模块之间建立强耦合关系，一旦某个模块文件的存放位置需要变更，使用该模块的其它模块的代码也需要调整。因此require函数支持第三种形式的路径，写法类似于foo/bar，并依次按照以下规则解析路径，直到找到模块位置。

1. 内置模块

如果传递给require函数的是NodeJS内置模块名称，不做路径解析，直接返回内部模块的导出对象，例如require(‘fs’)

1. node\_modules目录

NodeJS定义了一个特殊的目录用于存放模块。例如某个模块的绝对路径是/home/user/hello.js，在该模块中使用require(‘foo/bar’)方式加载模块时，则NodeJS依次尝试使用以下路径。



1. NODE\_PATH环境变量

与PATH环境变了类似，NodeJS允许通过NODE\_PATH环境变量来指定额外的模块搜索路径。NODE\_PATH环境变量中包含一到多个目录路径，路径之间在Linux下使用:分隔，在Windows下使用;分隔。例如定义了以下NODE\_PATH环境变量：



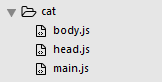
当使用require(‘foo/bar’)的方式加载模块时，则NodeJS依次尝试以下路径。



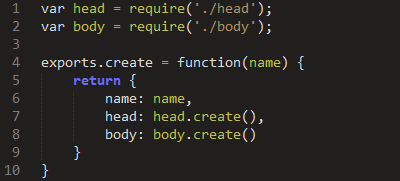
### 包（package）

JS模块的基本单位是单个JS文件，但复杂的模块往往是由多个子模块组成。为了便于管理和使用，我们可以把由多个子模块组成的大模块称为包，并把所有子模块都放在同一个目录下。

在组成一个包的子模块中，需要一个入口模块，入口模块的导出对象被称为包的导出对象。例如有以下目录结构。



其中cat目录定义了一个包，其中更包含了3个子模块。main.js作为入口模块，其内容如下：



在其他模块使用包的时候，需要加载包的入口模块。使用require(‘/home/user/lib/cat/main’)能达到目的，但是入口模块名称出现在路径中看上去不是个好主意。因此我们需要做点额外的工作，让包使用起来更像是单个模块。

#### index.js

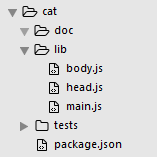
当模块的文件名是index.js，加载模块时可以使用模块所在目录的路径代替模块文件路径。例如以下两条语句等价



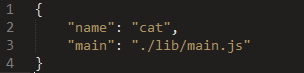
这样处理后，就只需要把包目录路径传递给require函数。

#### pakage.json

如果想自定义入口模块的文件名和存放位置，就需要在包目录下包含一个package.json文件，并在其中指定入口模块的路径。



其中package.json内容如下：



可以使用require(‘/home/user/lib/cat’)的方式加载模块。NodeJS会根据包目录下的package.json找到入口模块所在位置。

### 命令行程序

使用NodeJS编写的东西，要么是一个包，要么是一个命令行程序，而前者最终会用于开发后者。因此我们在部署代码时需要一些技巧，让用户觉得自己是在使用一个命令行程序。