第一章：安装 Node

本章内容：

- 安装并运行 Node
- 安装 Node 包管理器（NPM:Node Package Manager）
- 用 NPM 安装，卸载和升级包

2009年的JSCOnf大会上，一个叫Ryan Dahl的年轻程序员向人们展示了一个他正在做的一个项目，一个基于Google V8引擎的JavaScript运行平台，它提供了一套事件循环和低IO的应用程序编程接口（API）。和其他的服务端平台不同， JavaScript天生就是事件驱动IO，而这个项目又大大降低了编写事件驱动应用程序的复杂度，因此它很快就以不可思议的速度的成长流行起来，并应用到实际项目中。(*Jack：这段翻译的不太靠谱，原文：This project was not like other server-side JavaScript platforms where all the I/O primitives were event-driven and there was no way around it.*)

这个项目被命名为 Node.js，开发人员习惯叫它 Node，Node 提供了一套纯事件驱动非堵塞的工具包，用来构建高并发应用程序。

注意：Node 可以让你简单的构建快速可伸缩的网络服务。

自从被 Ryan Dahl 介绍以后，Node 受到了业界广泛的关注。他们已经开始用 Node 来部署快速并且可伸缩的网络服务。Node 实在太吸引人了。

一方面因为 JavaScript，JavaScript 是这个星球上应用最广泛的编程语言，大部分 Web 程序员都在浏览器端使用过 JavaScript，服务器端是它一个很自然的扩展。

另一方面因为 Node 娇小可爱，Node 的核心函数集很小，并且现有的 API 都非常精炼，为开发人员最小化了复杂度。当你想构建一些更加复杂的应用时，你只用挑选，安装一些你喜欢的第三方模块就可以了。

还有一个让 Node 如此吸引人的原因，它很容易上手，你可以在几分钟内完成下载安装，并运行起来.

通常按照官方网站(http://nodejs.org)上的步骤安装 Node 就可以了，它支持 Windows，Linux，Macintosh 以及 Solaris。

在 Windows 上安装 Node

Node 从 0.6.0 版本开始支持 Windows，要在 Windows 上安装 Node，只用从 Http://nodejs.org/#download 下载 node-v*.msi，然后双击运行即可，然后你可能会遇到类似图 1-1 的安全对话框。

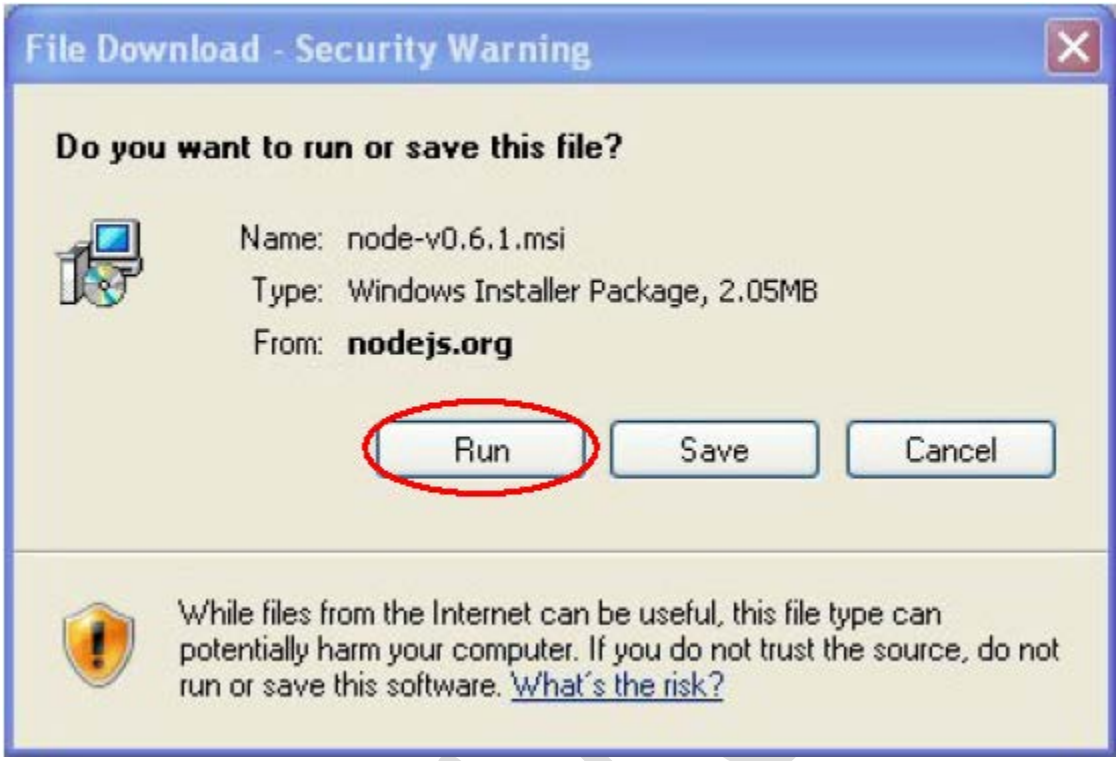

图 1-1

单击“运行”按钮，等下载完毕后会出现另外一个安全对话框（图 1-2），提醒你是否确定操作。

图 1-2

如果你同意，Node 安装向导就会出现（图 1-3），点击下一步 Node 就会开始安装，等一小会儿就安装完啦！见图 1-4

图 1-3

图 1-4

译者：Jack Yao，本系列其它文章请查看 http://yaohuiji.com/2013/01/08/pro-node-article-list/

在 Mac OS X 下安装

如果你使用 Mac OS X，你可以使用安装向导来安装 Node，首先，先到 http://nodejs.org/#download 下载 node-v*.pkg，下载完以后双击运行，你会看到安装向导的第一个对话框，见图 1-5

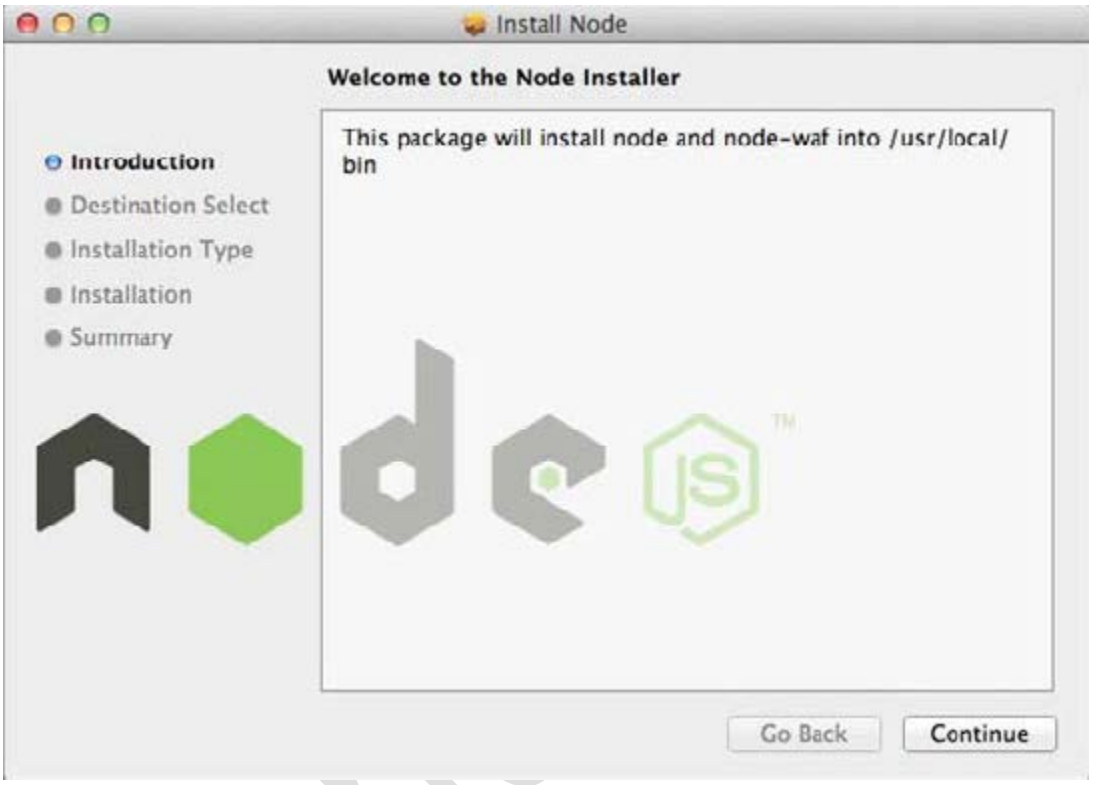

图 1-5

点“继续”安装，然后向导会要求你输入系统用户的密码，确定后安装就会开始，又是一小会儿，Node 又安装好啦！见图 1-6

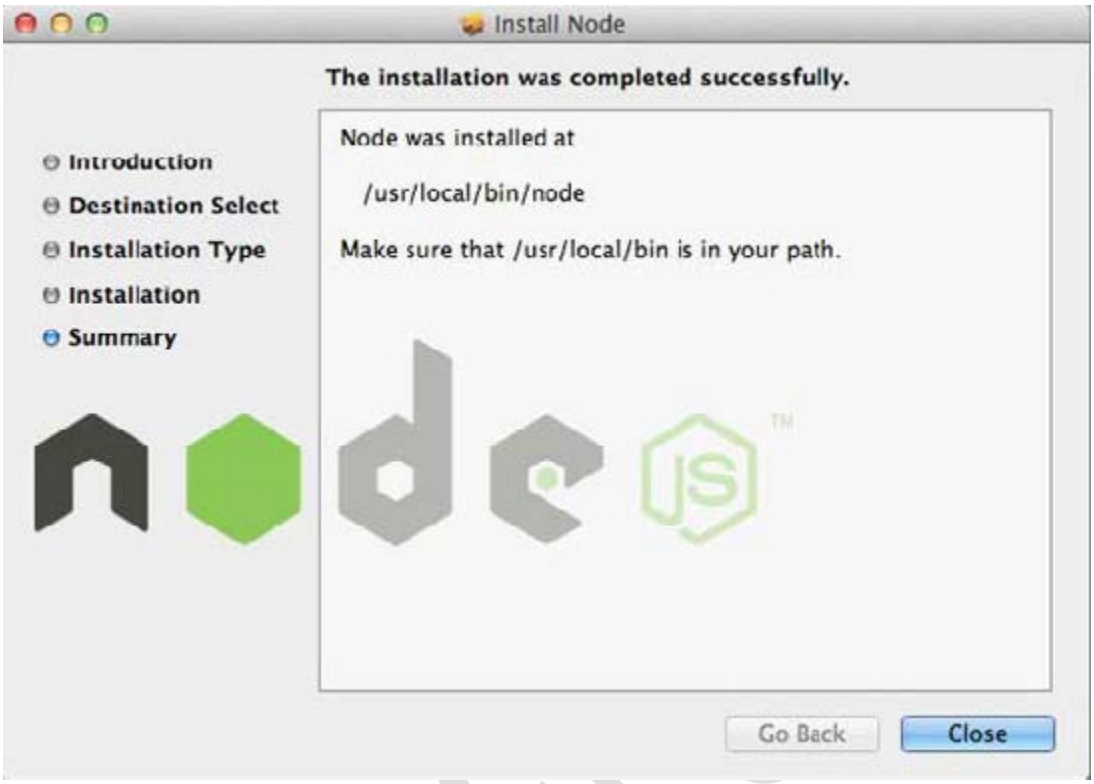

图 1-6

用源码安装

如果你使用 UNIX 系统，可以通过编译源代码来安装。首先，你需要选择你要安装的 Node 版本，然后下载相应的源码并构建，安装运行 Node。

注意：Node 依赖几个第三方代码库，不过幸运的是它们大部分已经包含在 Node 发布包里了，如果你从源码开始构建，你需要下面两个东西：

- **python（2.4 以上版本）——随 Node 发布的构建工具需要 python 环境来运行**
- **libssl-dev——如果你打算使用 SSL/TLS 加密，你需要安装这个，libssl 是 openssl 工具用到的类库，在 Linux 和 UNIX 系统下，你通常可以用系统的包管理器来安装。libssl 在 Mac OS X 下是预安装的，因此如果你用 Mac OS X 系统通常就不用再安装 libssl 了。**

选择 Node 版本

官方网站 nodejs.org 上通常有两个不同的 Node 版本可以下载：稳定版和最新版。

对于 Node，版本号最小位代表这个版本的稳定性，稳定版本使用偶数数字（如 0.2，0.4，0.6），非稳定版本使用奇数（0.1，0.3，0.5，0.7）。

非稳定版不仅功能上不稳定，而且 API 也有可能在后续版本中改变，稳定版中已发布的 API 是不会修改的。对于每个稳定分支，新的补丁不仅包含 bug 修复，也包括非稳定版中 API 的修改。

除非你想测试下最新非稳定版中的新特性，否则你应该选择最新的稳定版本。非稳定版

本对 Node 核心团队来说就像用来测试新特性的战场。

虽然，越来越多的项目和公司成功的在他们的产品中使用了 Node（官网首页有展示），但是你可能得学着忍受 API 从非稳定版到稳定版时发生的变化，当然，这就是学习一门新技术的代价。

下载 Node 源代码

现在你知道该下载哪个版本了，然后到官方网站 http://nodejs.org 找到对应的 tar 包，然后复制下载链接，如果你使用的 UNIX 系统，你的系统可能已经安装了 wget，这意味着你用一句 shell 命令就可以下载了：

$ wget http://nodejs.org/dist/v0.6.1/node-v0.6.12.tar.gz

如果你没安装 wget，你可能需要使用 curl：

$curl –O http://nodejs.org/dist/v0.6.1/node-v0.6.12.tar.gz

如果你这两个工具都没有安装，你得想别的办法把 tar 包下载到你的本地目录里——比如通过浏览器或者通过本地网络。

（本书的例子使用写作时的最新的稳定版：0.6.12）

构建 Node

现在我们有源码了，可以用它来构建 Node 的可执行文件。首先，你需要解压缩前面下载的 tar 包：

$ tar xfz node-v0.6.12.tar.gz

然后进入源码目录：

$ cd node-v0.6.12

配置:

$ ./configure

顺利的话你会看到成功的提示:

'configure' finished successfully (9.278s)

然后就可以开始编译了：

$ make

编译完毕，会有如下提示：

'build' finished successfully (0.734s)

安装 Node

当构建成，用下面的命令来安装 Node：

$ make install

这个操作会把 Node 可执行文件复制到/user/local/bin/node

如果遇到了权限问题,在命令前面加上 sudo,以 root 用户执行即可:

$ sudo make install

运行 Node

现在已经可以运行 Node 了，你可以先简单的体验一下 Node 的命令行交互界面（CLI:command-line interface），只需要调用 Node 可执行文件就行：

$ node

这个操作会启动 Node 的命令行交互界面，并等待你的输入，输入下面的命令来让 Node 做点事看看：

> console.log('Hello World!');

Hello World!

> undefined

也可以运行一个 JavaScript 脚本文件，比如，你创建了一个叫 hello_world.js 的文件，并包含以下内容：

console.log('Hello World!');

然后用这个脚本的文件名作为第一个参数参数来调用 Node 可执行文件：

$ node hello_world.js

Hello World!

最后，用 Ctrl+D 或者 Ctrl+C 来退出 Node 命令行交互界面。

准备和使用 Node 包管理器

到目前为止，你只能使用 Node 本身的语言特性和核心函数，这就是为什么大多数程序平台都有一个用来下载、安装，管理第三方模块的系统，在 Node 里，我们使用 Node 包管理器（NPM: Node Package Manager）

NPM 包含三部分：一个用来存放第三方包的代码库，一个管理本地已经安装包的机制，一个用来定义包依赖关系的标准。NPM 提供了一个公共的注册服务，它包含了大家发布的所有包，并提供了一个命令行工具，用来下载，安装和管理这些包。你可以按照 Node 的包格式标准来制定你的包或者应用需要依赖的其他第三方包。

虽然不用了解 NPM 就可以开始使用 Node，但是如果要用第三方包你就必须要学习它了，因为 Node 本身只提供了一些低层的 API，使用第三方模块可以大幅减少开发复杂度，不用什么都得亲自编码。NPM 允许你在一个沙箱里下载和使用模块，你可以尽情地试验你感兴趣的东西，而不用担心污染全局的包环境。

NPM 和 Node 以前需要独立安装，从 0.6.0 版本以后，NPM 已经包含在了 Node 安装包里。

使用 NPM 来安装，升级和卸载包

NPM 非常强大，你可以用很多方式来使用，它的代码库集中管理了所有的公共模块，你可以通过 http://search.npmjs.org 来访问它。Node 开源模块的作者可以把自己的模块发布到 NPM 上，其他人就可以用包安装描述里的模块名来下载，安装这个模块。

这部分内容，包含一些安装，删除包的常用操作，知道这些足以让你开始管理你自己应用程序所依赖的第三方包了，虽然如此，你首先还是需要明白这些命令在“全局”和“本地”模式下的区别，以及他们是如何影响依赖关系和模块查找的。

NPM 模块的全局和本地模式

NPM 的操作主要有两种模式：全局和本地。这两种模式会影响包存放的目录结构，以及 Node 加载包时的顺序。

本地模式是 NPM 的默认操作模式，在这个模式下，NPM 只工作在工作目录下，不会造成系统范围的修改，这个模式让你在某个 Node 程序下尽情地安装，测试模块，而不会影响你电脑上的其他 Node 程序。

全局模式适合那些将被很多程序使用，而且总是被全局加载的公共模块，比如命令行工具这些公不会被应用程序直接使用的模块。

如果你不知道一个模块该用哪个模式安装，那就应该使用本地模式。如果一个模块的作者需要某个模块被全局的安装，通常他会在安装说明里指出。

全局模式

如果你安装 Node 时使用的默认目录，在全局模式下面，NPM 会把包安装到/usr/local/lib/node_modules。如果你执行下面的命令，NPM 会搜索并下载名为 sax 的最新版并安装到/usr/local/lib/node_modules/sax 目录下。

$ npm install –g sax

注意：如果你当前 shell 用户没有足够的权限，你需要使用 root 用户登录或者使用 sudo 来执行命令：

$ sudo npm install –g sax

随后在你的 Node 脚本里需要 sax 模块的时候，使用下面的语句来加载：

var sax = require('sax');

如果你没有在应用程序目录下用本地模式安装过 sax，Node 将会在前面的安装目录里查找名为 sax 的模块，否则会优先加载本地版本。

默认模式是本地模式，因此你需要在 NPM 命令后加上-g 标记来启用全局模式。

本地模式

本地模式是 Node 包依赖机制的默认推荐模式，这个模式下，NPM 安装的所有东西都在当前工作目录（根目录也不例外），而不会影响任何全局的设置。这种机制可以让你一个个的设置应用程序的依赖模块以及它们的版本，而不用担心会污染全局的模块空间。这意味着你可以有依赖同一个模块不同版本的两个应用，它们却不会产生冲突。

在这个模式下，NPM 使用当前工作目录下的 node_modules 目录来存放模块，比如你当前工作目录是 /home/user/apps/my_app，NPM 将会用 /home/user/apps/my_app/node_modules 来存放所有本地模块。这意味着，如果你在代码里使用模块名来引用模块，Node 首先会到这个本地的 node_modules 目录下查找，如果没找到才会去搜索全局的 node_modules 目录，本地模块优先级总是高于全局模块。

安装模块

使用下面命令来安装一个模块的最新版本:

$ npm install <package name>

例如，下载和安装名一个名为 sax 的模块的最新版本，你首先需要把你应用程序的根目录设置为当前目录，然后输入：

$ npm install sax

这个操作，会在当前目录下建立 node_modules 子目录（如果不存在的话），然后在下面安装 sax 模块。

你也可以通过下面的命令，来选择安装某个特定的版本：

$ npm install <package name>@<version spec>

使用指定的版本号替换命令里的<version spec>占位符即可，比如，要下载 sax 模块的 0.2.5 版本，你只用运行：

$ npm install sax@0.2.5

<version spec>占位符也可以用一个版本范围来替换，比如，要安装 sax 模块 0.2 分支的最新版，可以运行：

$ npm install sax@0.2.x

或者，安装版本号小于 0.3 的最新版：

$ npm install sax@"<0.3"

甚至可以指定一个版本范围：

$ npm install sax@">=0.1.0<0.3.1"

卸载模块

使用下面命令可以卸载一个本地模块：

$ npm uninstall <package name>

如果要卸载的是一个全局模块，加上-g 标记即可：

$ npm uninstall -g <package name>

更新模块

使用下面命令来更新本地模块：

$ npm update <package name>

这个命令会尝试获取最新版的模块包并更新本地版本，如果本地没有安装，则会安装它，如果需要更新的是全局环境，需要加上-g 标记：

$ npm update -g <package name>

使用可执行文件

模块可以包含一个或多个可执行文件，如果你使用默认目录设置来安装一个全局模块，NPM 会把可执行文件安装到/usr/local/bin 目录下，这个目录通常也被设置为系统 PATH 环境

变量的一部分。如果你局部安装这个模块，NPM 会把所有可执行文件放到./node_modules/.bin 目录下。

处理依赖关系

NPM 不仅安装你需要的模块包，而且会安装这些模块所依赖的其它模块，比如，如果你需要安装模块 A，而 A 又依赖模块 B 和 C，那么在你安装 A 的时候 B 和 C 同时会被安装到./node_modules/A/node_modules 目录下。

例如，你用下面的命令本地安装了一个叫 nano 的模块：

$npm install nano

NPM 的输出会类似这样：

```
nano@0.9.3 ./node_modules/nano
├── underscore@1.1.7
└── request@2.1.1
```

这告诉你 nano 模块依赖 underscore 和 request 两个模块，并且还指出了安装的版本。如果你现在去查看./node_modules/nano/node_modules 目录，你会发现这两个模块已经被安装了：

$ ls node_modules/nano/node_modules
request underscore

使用 package.json 文件定义依赖关系

当开始编写一个应用程序时，可以在应用程序根目录创建一个 package.json 文件来定义应用程序的元数据，比如应用的名字，作者，代码库地址，联系方式等等。程序依赖的外部模块也在这个文件里指定。

如果不打算把程序发布到 NPM 上，就可以不用建这个文件，不过即使你的程序是私有的，这个文件其实也有用，它可以告诉 NPM 这个应用程序的依赖关系。（译者注：比如你把项目源码从开发环境复制到生产环境，可以通过调用 npm install 来一次性安装所有依赖包，npm 会通过 package.json 内指定的依赖关系来自动完成依赖模块的下载安装，不用自己一个个去操作，稍候有详细介绍）

package.json 是一个 JSON 格式的文件，包含了一系列属性，但是如果仅仅是为了说明程序的依赖关系，则只用一个 dependencies 属性就行。比如，一个叫 MyApp 的应用程序依赖 sax,nano 和 request 模块，只需要建立这样一个 package.json：

```
{
    "name" : "MyApp",
    "version" : "1.0.0",
    "dependencies" : {
        "sax" : "0.3.x",
        "nano" : "*",
        "request" : ">0.2.0"
    }
}
```

你指定了 MyApp 应用，依赖 0.3 版本的 sax，任意版本的 nano，以及版本高于 0.2.0 的

request 模块。

注意：你可能发现，如果你指定了 name 和 version 字段，NPM 会不工作，这只会发生在旧版本的 NPM，因为最初 NPM 是针对公共模块使用的，而不是私有程序。

然后，在应用程序的根目录，执行：

$ npm install

这样，NPM 就会分析依赖关系以及你本地的 node_modules 目录，并自动的下载和安装缺失的模块。

你也可以通过下面的命令把所有本地模块更新到符合你定义的依赖项设置的最新版本：

$npm update

事实上，你仅用 update 方法就行了，因为它会让 NPM 自动获取那些缺失的依赖模块。

小结

本章学习了如何安装 Node 和 Node 包管理器（NPM）,现在你可以使用 NPM 来安装，卸载，删除任何第三方模块，还学习了如何配合 package.json 文件来使用 NPM 管理应用程序依赖项。

现在你安装了 Node 和 NPM，可以去动手试试啦，不过，首先你需要知道一些关于 Node 和事件驱动的相关知识，下章将会介绍这些内容。

译者注：

本文对应原文第一部分第一章：Introduciton and Setup：Installling Node

本系列文章列表和翻译进度，请移步：Node.js 高级编程：用 Javascript 构建可伸缩应用（〇）