快速搭建NodeJs服务器

Create Time : 2017-03-14

前言

Node.js 是一个事件驱动 I/O 服务端 JavaScript 环境，也可以理解为服务器端运行的 JavaScript。JS 作为一门编程语言，是运行在称为 JS 运行时的虚拟机中的，而在 I/O 功能上，JS 更多依赖于宿主环境。一般我们遇到的宿主环境主要是浏览器，Node.js 则是在服务器端运行的高速 JavaScript 解释器。

近期遇到一个小型网站需要建立一个简单的 Web 服务器，原本想用 SpringMVC 解决，无奈 Spring 的哲学博大精深，自己才疏学浅，不能快速出货，因此决定用 Node.js 试试，这里就简单介绍下如何快速启动一个简单易用的 Web Server。这里我使用的开发环境是基于 Ubuntu 14.04 的 Elementary OS。

1. 准备工作

1.1 安装 NodeJS 环境和 npm 模块管理器

```
sudo apt-get install nodejs
sudo apt-get install npm
```

npm 是一个优秀的 Node 模块管理器，在开发中帮助我们解决很多第三方代码库的依赖管理事务。

1.2 设置 npm 代理

假如没有提前做任何额外措施，那么必然当你使用 npm 从网络下载自己需要的 NodeJs 模块时，速度会非常慢，甚至会因超时而失败，具体原因不多讲，推荐使用淘宝的 npm 镜像，问题会得到解决。

- 打开 ~/.npmrc 文件
- 输入 registry = https://registry.npm.taobao.org
- 保存退出

注意如果你曾设置过全局穿墙但是 npm 依然速度很慢，可以试试在 ~/.npmrc 文件中继续添加

- proxy=false

2. 用 Express Generator 搭建开发环境

Express 是目前最流行的基于 Node.js 的 Web 开发框架，可以快速地搭建一个完整功能的网站。

Express Generator 是 Express 的应用程序生成器工具，使用它可以快速建立完整的项目文件目录。

2.1 安装 Express Generator

```
$ npm install express-generator -g
```

2.2 建立 Express 应用程序

```
express --view=pug NodeApp
```

这里 view 参数是用来预设开发中使用的模板引擎的。更多参数如下

```
express -h

Usage: express [options] [dir]

  Options:

    -h, --help          output usage information
        --version       output the version number
    -e, --ejs           add ejs engine support
        --pug           add pug engine support
        --hbs           add handlebars engine support
    -H, --hogan         add hogan.js engine support
    -v, --view <engine> add view <engine> support (ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
    -c, --css <engine>  add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
        --git           add .gitignore
    -f, --force         force on non-empty directory
```

2.3 解决模块依赖

进入刚刚建立的 NodeApp 文件夹，可以看到里面有一个 package.json 文件，它定义了这个项目所需要的各种模块，以及项目的配置信息（比如名称、版本、许可证等元数据），npm可以依据它来管理项目模块。

打开 package.json 可以看到里面的内容如下

```
{
  "name": "NodeApp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.16.0",
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.0",
    "express": "~4.14.1",
    "morgan": "~1.7.0",
    "pug": "~2.0.0-beta10",
    "serve-favicon": "~2.3.2"
  }
}
```

现在在终端执行 npm 命令解决模块依赖，npm 会按照 package.json 文件的内容下载相应模块，当然，如果没有设置穿墙措施这一步是无法执行的

```
npm intsall
```

3. 初次启动服务器

基本上该有的配置和基本的逻辑代码 Express Generator 都帮我们做好了，那么我们其实已经可以用一行命令启动服务器了。

执行以下命令

```
npm start
```

访问 localhost:3000 即可看到 Express 欢迎页面了。

4. 设置访问地址

可以看到在之前建立的文件目录下还有一个 app.js 文件，其实它就充当了一个项目中的main函数的角色，里面使用了很多 Express 中间件和 Express 语法，这里不一一叙述。

在实际生产环境中，我们需要自己设定外部访问端口，比如通过 Http 的 80 端口访问我们的服务器，那么就可以在app.js文件的 "module.exports = app;" 语句前加上如下代码

```
var server = app.listen(80, "0.0.0.0", function () {
    console.log("服务器IP地址:" + ip.address());
    var host = server.address().address;
    var port = server.address().port;

    console.log("应用已启动，访问地址为 http://%s:%s", host, port)
});
```

这里简单解释下，80 意味着我们的服务器程序将监听本机的80端口，0.0.0.0 意味着本地和外部访问请求都将由我们的服务器程序进行处理。

同时启动了服务器后我们在终端也可以看到当前主机的IP地址以及服务器程序接受的访问地址。

5. 设置 index 页面内容

到这里为止，我们访问 localhost 地址会看到 Express 默认的欢迎页面，那么如何返回一个我们自己的页面呢。比如我们现在有一个 index.html 页面，我们需要在用户访问 localhost 时返回这个 html 页面，暂时不考虑静态文件的问题。

可以进入 app.js 文件，它现在应该长这样，

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', index);
app.use('/users', users);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

var server = app.listen(80, "0.0.0.0", function () {
    console.log("服务器IP地址:" + ip.address());
    var host = server.address().address;
    var port = server.address().port;

    console.log("应用已启动，访问地址为 http://%s:%s", host, port)
});

module.exports = app;
```

其中有一句是这样的

```
app.use('/', index);
```

它的含义是当服务器程序捕获访问路径为 "/" 的请求时，由 index 中间件进行相应处理。

在这里对于 Express 中间件网上有很好的描述

> 简单说，中间件（middleware）就是处理 HTTP 请求的函数。它最大的特点就是，一个中间件处理完，再传递给下一个中间件。App 实例在运行过程中，会调用一系列的中间件。 每个中间件可以从 App 实例，接收三个参数，依次为 request 对象（代表 HTTP 请求）、response 对象（代表 HTTP 回应），next 回调函数（代表下一个中间件）。每个中间件都可以对 HTTP 请求（request 对象）进行加工，并且决定是否调用next 方法，将 request 对象再传给下一个中间件。

那么 index 中间件从何而来呢？

```
var index = require('./routes/index');
```

所以我们可以去 routes 文件夹下查看 index 文件，它应该长这样

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

从代码注释可以看出，这里就是负责返回前面提到的 Express 欢迎页面的逻辑代码，当然它使用到了模板引擎的知识，我们不具体介绍，直接简单粗暴地实现我们的需求吧。

首先我们把 index.html 页面保存到文件目录下的 public/html 文件夹下（没有就自己创建），然后我们在 routes/index 文件中返回这个 html 页面，在这里我们将使用到文件读写方法。

```
var path = require('path');
res.sendFile(path.resolve('public/html/index.html'));
```

path 变量是 Express 中的变量，path.resolve 方法可以将传入的相对地址转换为绝对地址，这里面涉及到关于 NodeJs 文件路径的知识，不做具体介绍。

res 变量代表着服务器对于此次请求的返回对象，那么在这里相当于我们向客户端返回的是存放在 public/html/index.html 文件。

重启程序，访问 localhost，就可以看到 index.html 了。

6. 设置静态文件路径

对于 JS 和 CSS 这样的静态文件，在 Express 中都统一放置在 public 文件夹下，Express 遇到对静态文件的请求将会从 pulic 下读取并返回相应文件。

而设置这一路径的语句其实也在 app.js 中

```
app.use(express.static(path.join(__dirname, 'public')));
```

因此如果项目需要，也可以自行修改这一路经。

7. 后台运行服务器

在之前我们终端运行 npm start 的时候，应该可以看到所有访问打印的日志语句都在终端显示了，这时如果我们关闭终端，程序也将相应停止，那么如何在后台运行我们的服务器程序，并将打印的日志语句都写入到专门的日志文件呢，这里需要用到 Linux 的 nohup 命令和重定向符。

```
npm start 1> log 2> error
```

> 是linux下的重定向符。 > 将会重新写入目标文件，即不保存目标文件的原始内容，如果使用 >> 则会在目标文件后面附加内容。

在 Linux 中，一个程序可以在几个编号的文件流中的任一个上产生输出。然而我们必须把这些文件流的前三个看作标准输入，输出和错误，shell 内部参考它们为文件描述符 0，1 和 2。因此这个语句的意思就是将标准输出重定向到当前目录下的 log 文件中，将错误输出重定向到当前目录下的 error 文件中。

但是这样只解决了输出信息的转移输出问题，程序依然是在终端下运行的。可以使用 Linux 下的 nohup 命令实现这一目的，使用如下

```
nohup npm start 1> log 2> error &
```

不要忘记最末位的 & 符号。

> 在后台运行作业时要当心：需要用户交互的命令不要放在后台执行，因为这样你的机器就会在那里傻等。不过，作业在后台运行一样会将结果输出到屏幕上，干扰你的工作，因此需要重定向符的帮助。

后续

现在一个简单的 Web 服务器就搭建完成了，这只是一个 Web 服务器最开始的一段路，后面的开发则需要对 NodeJs 和 Express 的深入学习和使用。

参考

- NodeJs 安装 (http://www.runoob.com/nodejs/nodejs-install-setup.html)
- npm 穿墙 (http://www.cnblogs.com/hustskyking/p/npm-cross-wall.html)
- Express 框架 (http://javascript.ruanyifeng.com/nodejs/express.html#toc4)
- Express 应用程序生成器 (http://expressjs.com/zh-cn/starter/generator.html)
- package.json文件 (http://javascript.ruanyifeng.com/nodejs/packagejson.html)
- 浅析 NodeJs 的几种文件路径 (https://github.com/imsobear/blog/issues/48)
- Linux 重定向 (https://billie66.github.io/TLCL/book/zh/chap07.html)
- Linux 技巧：让进程在后台可靠运行的几种方法 (https://www.ibm.com/developerworks/cn/linux/l-cn-nohup/)