目　录

致谢

Introduction

Develop a npm package

- Develop node-validator
- Write test cases
- Manage npm package

Node.js command line tool

- Request API
- Beautify the output
- Complete the tool

Web development with Express

- Use Express
- Develop the Express app
- Server deploy

Appendix

- Node.js basics
- npm basics
- Reference
- Update

致谢

当前文档 《Node In Action（Node.js实战 英文版）》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-03-17。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/node-in-action-en

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

Introduction

- Node.js in action

Node.js in action

This is a tiny book about Node.js (view update on this page). It aims to help the ones who are new to Node.js to learn to create their own Node.js projects and have fun coding with it.

Before reading this book, you should have already know much about the JavaScript programming language, and I will try hard to explain things in brief sentences and brings your three tutorial of developing Node.js projects. You would learn a lot during the reading period.

> ***NOTE**: If you meet problems about Node.js and npm, refer to the sections in appendix: Node.js basics and npm basics*

And if you have any question or doubt about my poor English, feel free to create issues on the book's Github or send E-mail to me.

One more thing, if you enjoy reading this book, star the repository on Github or donate me a cup of coffee via Alipay.

Develop a npm package

- Develop npm package

Develop npm package

In the first chapter, I will introduce you the develop process of node-validator and give your some instructions about how to develop a npm module, write test case about it and finally publish it to the npm. Anyway, welcome to the family of npm.

Before develop the first npm module, let's have a look at what it is like.

Firstly, you can have `node-validator` installed on your own computer with a simple command `npm install is-valid` in the Node.js REPL or create an empty JavaScript file in any place:

```
1. var validator = require('is-valid');
2.
3. validator.isEmail('foo@bar.net'); // true
```

Yes, that's it. It's really simple but also meaningful.

Develop node-validator

- A string validator

A string validator

We are going to develop a npm module which is used to validate string.

First of all, build the folder structure of a package:

```
node-validator
  |- lib/
  |- test/
  |- package.json
  |- index.js
  |- README.md
```

As the project is quite simple, it’s fine to put all the codes in `index.js` .

However, we can place all the implement codes in the `lib` folder for extending the project later and regard the `index.js` file as an entrance of the moudle.

```
module.exports = require('./lib');
```

Creating a `index.js` file in the `lib` in order to write the core of our moudle:

```
module.exports = function () {
    console.log('Hello npm!');
}
```

The program above is the npm version of the well-known “Hello World”. There’s no problem to use `node index.js` to have a test of it.

In the module system of CommonJS, `module.exports` can output a function or an object. So we can easily write codes like this:

```
module.exports = {
    isEmail: function () {},
    isAllEnglish: function () {}
```

```
};
```

After that, we can continue programming the module using regular expressions:

```
module.exports = {
    isEmail: function (str) {
        return /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$/.test(str);
    },
    isAllEnglish: function (str) {
        return /^[a-zA-Z]+$/.test(str);
    }
}
```

Up to now, the first alpha version of `node-validator` is ready for use.

Using `npm link` command could place the current module to the system's npm module folders. In the other words, if we modify the codes in our project, we could feel that when using the module.

Assuming we have changed the name to `validator-test` in the `package.json` file, then the code below can work as expected:

```
var validator = require('validator-test');

validator.isEmail('foo@bar.net'); // true
```

There's no doubt that we can add more functions to the object showing above. And we can also split different functions into different files and combine them into one module using the module system of Node.js.

Write test cases

- Write test cases

Write test cases

There’re lots of excellent test frameworks written in Node.js. For example, mocha, jasmine and so on.

We would like to use mocha to write our test cases.

> *Anyone who wants to know more about Mocha could visit its homepage for more information: mocha.org*

- Install mocha

```
npm install -g mocha
```

- Use `assert`

```
var assert = require('assert');
```

“assert” includes many Node.js modules about assertion, e.g. should.js, expect. Most of this modules are practices on Behavior Driven Development, a.k.a, BDD.

- Example of assert

```
describe('Array', function() {
    describe('#indexOf()', function() {
        it('should return -1 when the value is not present', function() {
            [1,2,3].indexOf(5).should.equal(-1);
            [1,2,3].indexOf(0).should.equal(-1);
        });
    });
});
```

It seems that we have just described a thing using English. Yes, what we are going to do is to describe what the functions in “node-validators” would works like.

Following the examples above, here comes the prototype of our use cases:

```
var assert = require('assert');
var validator = require('validator-test');

describe('Validator', function () {
    describe('#isEmail', function () {
        it('should return true when the string is an email address', function () {
            if (validator.isEmail('foo@bar.net') !== true) {
                throw new Erorr('Validator not right');
            }
        });
    });
});
```

Type `mocha` in your terminal, this would run the `test.js` file in `test` folder automatically:

```
mocha
```

The we will get the result:

```
Validator
#isEmail
✓ should return true when the string is an email address


1 passing (6ms)
```

What’s next? Just write test cases for every functions in “node-validator” in order to cover all.

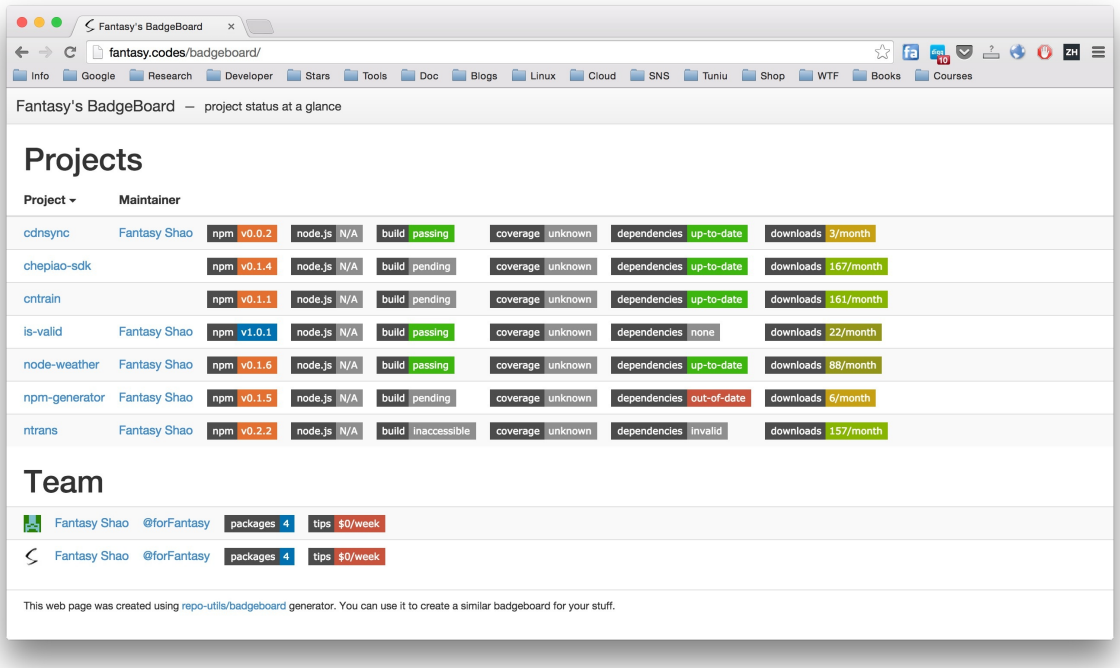
Manage npm package

- Manage npm package
  - Bug tracking

Manage npm package

You can view all the npm modules that you have published to npm on your homepage.

But if you are eager to view them in a more direct way, it’s pretty cool to use a tool named badgeboard.



There’re already many excellent services for managing and get in touch of your npm modules.

As the picture shows above, here is a list of common services:

- Continuous integration: travis
- Test cases coverage: coveralls
- Module dependencies: david-dm

These tools can help you manage your modules easily and know the situation of the modules.

For example, using “david-dm” could know whether the dependencies of a module is newest. And if its dependencies are already behind, you’d better upgrade the module.

Bug tracking

Even with tools such as travis and coveralls, there would also be many situations that test cases can’t cover or something out of expectation.

For npm, it’s just a platform for hosting modules and provide tools for the modules, and npm does not have features like BUG tracking and feedback. For this reason, we could use Github issues to collect the feedback of different users.

Node.js command line tool

- Develop command line tool
  - Command line parameters
  - node-translator

Develop command line tool

In the second chapter, we're going to learn to develop command line tool with Node.js

In Linux (or Unix) systems, command line tools are especially helpful for daily work and development. And Node.js could provide us a rapid developing experience to build a command line tool.

Following is a simple Node.js script:

```
1. #!/usr/bin/env node
2.
3. console.log('Hello CLI');
```

Command line parameters

We will need to get parameters for the command line tool. TJ has open sourced a practical module named Commander.js which is used to get different kinds of command line parameters. It is really convenient for our development.

For example, we could use commander.js like this:

```
1. #!/usr/bin/env node
2.
3. var program = require('commander');
4.
5. program.version('0.0.1').parse(process.argv);
```

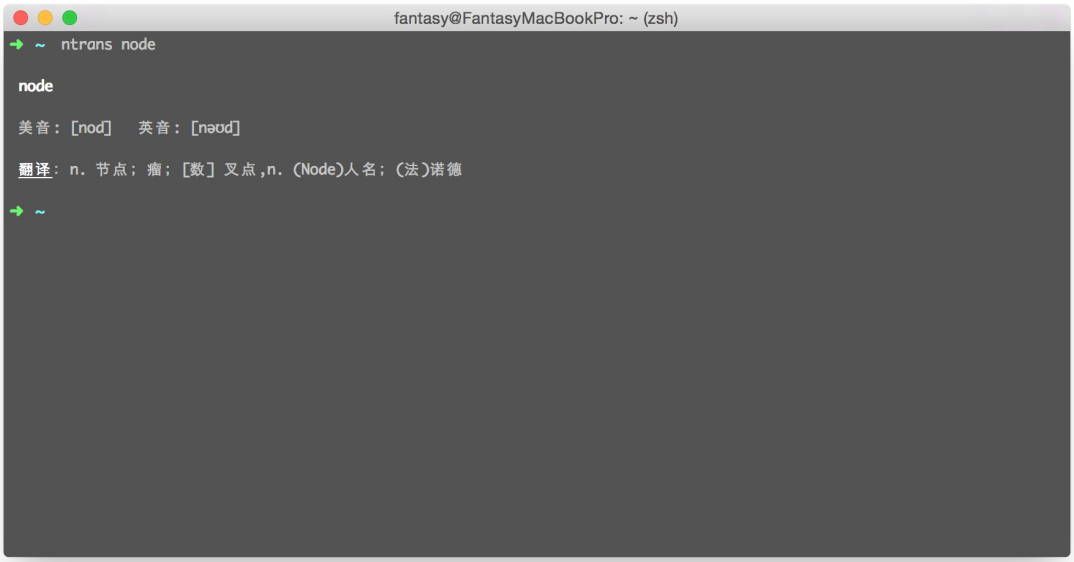
It's easy and efficient to use this module, and now we could get the tool's version just like many other existed command line tool:

```
1. node test --version
```

Obviously, this module is required by the tools which needs passing a lot parameters.

node-translator

node-translator is a command line tool which translate words using Youdao Dic’s API in terminals. And in this chapter we are going to complete it:



All the sections in this chapter would explain things using its codes.

Request API

- Request API
  - Use request
  - Youdao dic

Request API

The core feature of our command line tool is to request the API of Youdao dic. After handling with the response of the API, we could display the results to the terminal.

Use request

request is one of the most popular module on npm, it has been downloaded for 20~30 million time per day.

Just like Ajax methods of jQuery used in Front-end development and AFNetworking in iOS development, **request** is widely used in Node.js applications for requesting APIs.

And it is extremely easy and simple to understand its usage, following is the simplest way to use it:

```
1. var request = require('request');
2.
3. request('http://www.google.com', function (error, response, body) {
4.     if (!error && response.statusCode == 200) {
5.         console.log(body);
6.     }
7. });
```

Youdao dic

Youdao dic provides a Open platform for developers to register and apply for API services.

The most common way to request API is quite simple, we could just add parameters in the URL and request the API with GET method:

```
http://fanyi.youdao.com/openapi.do?keyfrom=node-translator&key=2058911035&type=data&doctype=json&version=1.1&q=test
```

It’s convenient for us to find what the parameters mean in the documentations.

Next, we can have a try to call the API using request, we are going to get the Chinese meaning of the word ‘test’:

```
var request = require('request');

request('http://fanyi.youdao.com/openapi.do?keyfrom=node-translator&key=2058911035&type=data&doctype=json&version=1.1&q=test', function (error, response, body) {
    if (!error && response.statusCode == 200) {
        console.log(body);
    }
});
```

Assuming the codes above has been saved as a JavaScirpt file named `request-test.js`, we could run it using `node request-test.js` and get the following result:

```
{"translation":["测试"],"basic":{"us-phonetic":"tɛst","phonetic":"test","uk-phonetic":"test","explains":["n. 试验；检验","vt. 试验；测试","vi. 试验；测试","n. (Test)人名；(英)特斯特"]},"query":"test","errorCode":0,"web":[{"value":["测试","试验","检验"],"key":"test"},{"value":["测试工程师","测试员","软件测试工程师"],"key":"Test engineer"},{"value":["硬度试验","硬度测试","硬度实验"],"key":"hardness test"}]}
```

That’s it! The JSON contents in the result body is what we need for our tool.

Beautify the output

- Beautify the output
  - Add color
  - Format the output style

Beautify the output

Actually, we have finished the core feature of the tool with the data we needed.

However, it is better to output the result in a more user-friendly way for a command line tool.

Add color

colors is a module for adding colors for command line tools, and here is a simple example:

```
var colors = require('colors');

console.log('Color'.green);
```

By adding the name of a color after a string, we could make the output string colorful. For the list of colors it supports, you'd better refer the document on Github or npm.

Format the output style

In the last chapter, we got the result in JSON as below:

```
{"translation":["测试"],"basic":{"us-phonetic":"tɛst","phonetic":"test","uk-phonetic":"test","explains":["n. 试验；检验","vt. 试验；测试","vi. 试验；测试","n. (Test)人名；(英)特斯特"]},"query":"test","errorCode":0,"web":[{"value":["测试","试验","检验"],"key":"test"},{"value":["测试工程师","测试员","软件测试工程师"],"key":"Test engineer"},{"value":["硬度试验","硬度测试","硬度实验"],"key":"hardness test"}]}
```

Formatted:

```
{
```

```
    "translation":["测试"],
    "basic":{
        "us-phonetic":"tɛst",
        "phonetic":"test",
        "uk-phonetic":"test",
        "explains":[
            "n. 试验；检验",
            "vt. 试验；测试",
            "vi. 试验；测试",
            "n.(Test)人名；(英)特斯特"
        ]
    },
    "query":"test",
    "errorCode":0,
    "web":[{
        "value":["测试","试验","检验"],
        "key":"test"
    },{
        "value":["测试工程师","测试员","软件测试工程师"],
        "key":"Test engineer"
    },{
        "value":["硬度试验","硬度测试","硬度实验"],
        "key":"hardness test"
    }]
}
```

A better way to output the result should be with proper new lines, spaces. About this, you can refer the output.js file in the ‘node-translator’.

Complete the tool

- Complete the tool
  - Get parameters

Complete the tool

In the previous sections, we have already know how to use `request` to request the API of Youdao Dictionary and use `colors` to make the tool colorful.

And now we are going to make it complete.

Get parameters

In the first chapter, I've introduced `commander.js` which is created by TJ, however, `node-translator` does not need lots of parameters, so we could get the parameters directly from the command line.

It is easy for us to get the array of parameters by using `process.argv` according to the official Node.js documentation.

```
node test.js a b
```

The return of `process.argv` is `['node', 'test.js', 'a', 'b']`.

In the previous section, we could get the data needed by `GET` method. Actually, there are more situations to pass parameters by command line rather than hard coded in our programs.

So we are able to get parameters in the way we mentioned above:

```
var param = process.argv[2];
var word = param ? param : '';
```

Then we could append parameters to the URL and request for appropriate return values:

```
var request = require('request');

request('http://fanyi.youdao.com/openapi.do?keyfrom=node-
```

```
    translator&key=2058911035&type=data&doctype=json&version=1.1&q=' + word, function (error,
    response, body) {
4.      if (!error && response.statusCode == 200) {
5.          console.log(body);
6.      }
7.  });
```

Web development with Express

- Web development with Express

Web development with Express

In the third chapter, we are going to learn the most popular Node.js Web development framework — Express, to develop a small website, including sections about how to deploy.

The project is open source on Github: github.com/SFantasy/Riki and deployed on Amazon Web Service: riki.fantasy.codes.

Use Express

- Use Express
  - Hello World in Express
  - Express generator

Use Express

Express is the most popular Web development framework created by TJ, and the latest version of it is 4.x.

Hello World in Express

Here is a simple snippet code using Express, return a string `Hello World` via method `res.send()` :

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
    res.send('Hello World');
});

app.listen(3000, function () {
    console.log('Express server started.');
});
```

Run `app.js` :

```
node app.js
```

Express generator

Express also provide a tool to generate project in seconds like other popular web frameworks: Express Generator

- Install

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

- Generate project

```
1. express test
```

Enter the folder and install all the dependencies:

```
1. cd test && npm install
```

Now you are able to run this project with `npm start` . Awesome!

Develop the Express app

- Develop Express application
  - Routes
  - MongoDB
  - Reference

Develop Express application

MVC is common architecture which many applications prefer. MVC stands for Model-View-Controller:

- Model - Implement the main features of the application, handling databases and so on.
- Controller - Dealing with requests, communicating with Models and Views.
- View - The UI layer of the application

So, we would like to choose this application development mode, and here is the basic structure of our folders:

```
- models/
- controllers/
- views/
- app.js
- package.json
```

And in this project, I also create a Service layer which take the responsibility to handling the data — so the Model layer just needs to define the Data objects.

Routes

In the project, we could define routing rules in the Controllers, but it is better to declare the rules in separated files.

For some small projects, a single route file is enough. Because there would not be many routing rules in the project comparing to other codes.

However, it is a better practice to use multiple routing files as it will make your project's structure clearer. Here is a example routing rules:

```
app.get('/', site.index);
app.post('/register', user.register);
```

`site` and `user` are the names of two Controllers, defining different `GET` or `POST` rules.

MongoDB

MongoDB is convenient for Node.js to manipulating for MongoDB is a document database which use JSON objects to store data. So it is quite simple to handle data with MongoDB using Node.js.

In the project, we would like to use `Mongoose` as our ORM for MongoDB. Below is an example of connecting MongoDB:

```
var mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1/test', function (err) {
  if (err) {
    console.log('connect to %s error: ', err.message);
    process.exit(1);
  }
});
```

And define a Scheme called User:

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var UserSchema = new Schema({
  name: { type: String },
  password: { type: String }
});

mongoose.model('User', UserSchema);
```

Reference

- MongoDB: https://www.mongodb.org/
- Mongoose: http://mongoosejs.com/

Server deploy

- Deploy app on server
- Deploy
- Nginx

Deploy app on server

In this section, you will get to know how to deploy your Node.js application on Amazon Web Service.

There’re two ways to install Node.js on Ubuntu and other Linux distributions:

1. Compile and install from the source of Node.js
2. Using package management tool (like apt-get and yum) of the Operating System: instruction on Github), and I chose this approach:

```
1. curl -sL https://deb.nodesource.com/setup | sudo bash -
2. sudo apt-get install nodejs
```

Then you could use VIM to edit a simple JavaScript file to have a test on your server:

```
1. console.log('Hello Node.js and AWS.');
```

Deploy

Firstly, I should clone the repository on Github:

```
1. git clone https://github.com/SFantasy/Riki.git
```

Then install the dependencies:

```
1. npm install
```

For we run the App using supervisor during the developing period, it’s better to choose PM2 to run it on EC2.

Install PM2: `sudo npm install -g pm2`

Edit the “scripts” field on package.json

```
1. pm2 start ./bin/www
```

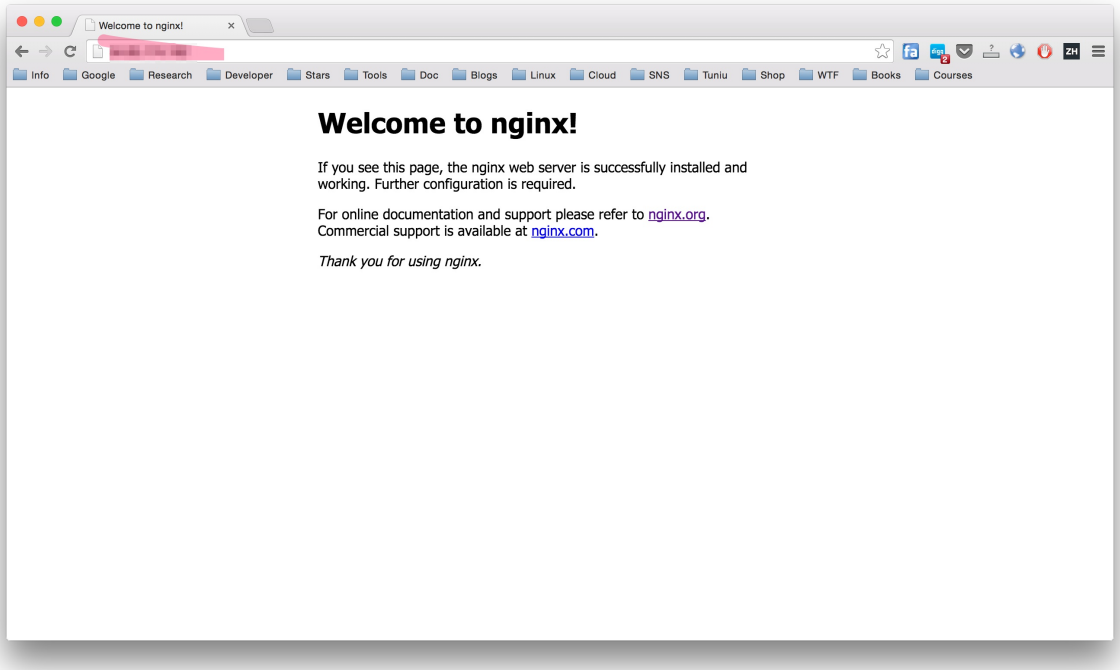
Now we are able to start our application using `npm start` command.

Nginx

It’s very simple to install Nginx on Ubuntu:

```
1. sudo apt-get install nginx
```

Nginx has been started when the installation completed. If we visit the public IP of EC2, the below is what we can see:



And now, we have to modify the config file of Nginx to make the proxy work.

> *The default config file is at* `/etc/nginx/nginx.conf`

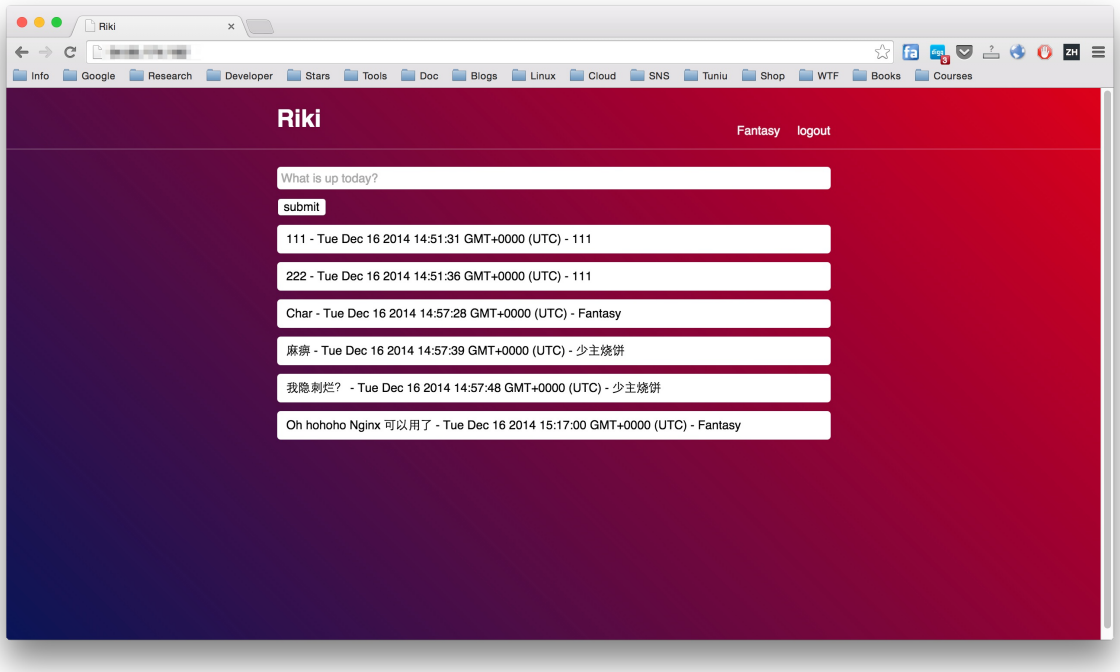
```
1. upstream riki {
2.     server 127.0.0.1:3000;
```

```
}

server {
    listen 80;
    server_name localhost;

    location / {
        proxy_pass http://riki;
    }
}
```

And now we can visit Riki via IP without `:3000` :



Appendix

- Appendix

Appendix

This chapter introduce some basic knowledge of Node.js and npm.

Node.js basics

- Node.js basics
  - Node.js version “Hello World”
  - Version management of Node.js
  - Module system
  - The simplest Node.js server:

Node.js basics

If you have some basic knowledge about Node.js, you can skip this chapter or just regard it as a reference.

Node.js version “Hello World”

```
1. console.log('Hello World');
```

- Run the script on your terminal

```
1. node test.js
```

Version management of Node.js

With the effort of worlds’s contributors, Node.js is getting better and better. So it is a usual for us to change and update the version of Node.js on our machine.

I recommend NVM to do this stuff. Its features are complete and convinent. For more infomation, you could visit its Github page.

Module system

Node.js implement the CommonJS sepc):

- Use `global` to define a global variable:

```
1. global.test = true;
```

- Use `require` to import a module:

```
var fs = require('fs');
```

- Use `module.exports` to exports a module:

```
module.exports = function () {
    console.log('Hello Node.js');
};
```

- Use `exports` to export multiple methods or Object:

```
exports.foo = function () {};
exports.bar = function () {};
```

The simplest Node.js server:

```
var http = require('http');

http.createServer(function (req, res) {
    res.send('Hello');
    res.end();
}).listen(3000);
```

Visit it on localhost:300 and you could see the word “Hello”.

npm basics

- npm
  - Introduction
  - Basic use
    - Install dependencies
    - Module management
  - npm mirrors

npm

Introduction

npm is usually short for Node Package Manager, and there exists many other alias.

In my opinion, the relationship between Node.js and npm is just similar with Maven on Java, PIP on Python and GEM on Ruby.

With npm, you could manage the dependencies of your Node.js projects with ease.

Basic use

The basic usage of npm includes dependencies management, module management and scripts.

> *Surely it is better to refer the documentation of npm or via* `npm -l` *or* `man npm` *to read the documentation.*

Install dependencies

- Install the dependencies of a projects

```
1. npm install
```

- Install a module

```
1. npm install express
```

- Install a module as project’s dependency

```
npm install express --save
```

- Install a module as project’s dev-dependency

```
npm install express --save-dev
```

Module management

- Initial a module

```
npm init
```

This command only create a package.json for your module, and will not create other files.

- Publish a module

```
npm publish
```

No matter whether it is the first time you publish it, use this command. And it is a must to `npm adduse` before you publish.

npm mirrors

Due to the existence of “GFW”, developers in China can use Taonpm as the mirror of npm and install modules without panic.

Reference

- Reference

Reference

- Node.js API
- npm Documentation
- Wikipedia MVC

Update

- Update

Update