**公司：万讯科技**

**岗位：web前端组**

**组员：江增宇**

**时间：2018-9-11**

**使用 Node.js 开发简单的脚手架工具**

# 前言

* 像我们熟悉的vue-cli，react-native-cli等脚手架，如vue-cli只需要输入简单的命令：vue init webpack project， 既可快速帮我们生成一个初始项目。在实际工作中，我们可以定制一个属于自己公司的脚手架，来提高自己的工作效率。

# 思路

* 要开发脚手架，首先要理清思路，脚手架是如何工作的？
* 我们可以借鉴 vue-cli 的基本思路。vue-cli 是将项目模板放在 git 上，运行的时候再根据用户交互下载不同的模板，经过模板引擎渲染出来，生成项目。
* 这样将模板和脚手架分离，就可以各自维护，即使模板有变动，只需要上传最新的模板即可，而不需要用户去更新脚手架就可以生成最新的项目。那么就可以按照这个思路来进行开发了。

# 依赖所需第三方库

首先来看看会用到哪些库？

* commander.js，可以自动的解析命令和参数，用于处理用户输入的命令。
* download-git-repo，下载并提取 git 仓库，用于下载项目模板。
* Inquirer.js，通用的命令行用户界面集合，用于和用户进行交互。
* handlebars.js，模板引擎，将用户提交的信息动态填充到文件中。
* ora，下载过程久的话，可以用于显示下载中的动画效果。
* chalk，可以给终端的字体加上颜色。
* log-symbols，可以在终端上显示出 √ 或 × 等的图标。

# 初始化项目

首先创建一个空项目，暂时命名为 okii-cli，然后新建一个 index.js 文件，再执行 npm init 生成一个 package.json 文件。最后安装上面需要用到的依赖。cmd命令执行

npm install commander download-git-repo inquirer handlebars ora chalk log-symbols -S



安装完成后在package.json下dependencies会安装好如下依赖



# 处理命令行-commander

## 1) package.json添加bin

node.js 内置了对命令行操作的支持，在 package.json 中的 bin 字段可以定义命令名和关联的执行文件。所以现在package.json中加上 bin 的内容：

{

"name": "okii-cli",

"version": "1.0.0",

"description": "基于node的脚手架工具",

"bin": {

" okii ": "index.js"

},

...

}

## 2）定义init命令

在 index.js 中来定义 init 命令：

#!/usr/bin/env node

const program = require('commander');

program.version('1.0.0', '-v, --version')

.command('init <name>')

.action((name) => {

console.log(name);

});

program.parse(process.argv);

调用 version('1.0.0', '-v, --version') 会将 -v 和 –version 添加到命令中，可以通过这些选项打印出版本号。  
调用 command('init <name>') 定义 init 命令，name 则是必传的参数，为项目名。  
action() 则是执行 init 命令会发生的行为，要生成项目的过程就是在这里面执行的，这里暂时只打印出 name。  
其实到这里，已经可以执行 init 命令了。我们来测试一下，在 okii-cli 的同级目录下执行：

node ./index.js init HelloWorld



可以看到命令行工具也打印出了 HelloWorld，那么很清楚， action((name) => {}) 这里的参数 name，就是我们执行init 命令时输入的项目名称。

此时命令已经完成，接下来就要下载模板生成项目结构了。

# 下载模板- download-git-repo

download-git-repo 支持从 Github、Gitlab 和 Bitbucket 下载仓库，各自的具体用法可以参考官方文档。

由于是展示项目，所以把模板仓库放在了本人 Github 上，那么在 action() 中进行操作下载模板：

//index.js文件

#!/usr/bin/env node

const program = require('commander');

const download = require('download-git-repo');

program.version('1.0.0', '-v, --version')

.command('init <name>')

.action((name) => {

download(' qzdev/HTML5', name, (err) => {

console.log(err ? 'Error' : 'Success')

})

});

program.parse(process.argv);

!/usr/bin/env node这种用法是为了防止操作系统用户没有将node装在默认的/usr/bin路径里

download() 第一个参数就是仓库地址，但是有一点点不一样。实际的仓库地址是 qzdev/HTML5 ，可以看到端口号后面的 ‘/‘ 在参数中要写成 ‘:’，#master 代表的就是分支名，不同的模板可以放在不同的分支中，更改分支便可以实现下载不同的模板文件了。第二个参数是路径，上面我们直接在当前路径下创建一个 name 的文件夹存放模板，也可以使用二级目录比如 test/${name}

# 七、命令行交互-inquirer

* 命令行交互功能可以在用户执行 init 命令后，向用户提出问题，接收用户的输入并作出相应的处理。这里使用 inquirer.js 来实现。

const inquirer = require('inquirer');

inquirer.prompt([

{

type: 'input',

name: 'author',

message: '请输入作者名称'

}

]).then((answers) => {

console.log(answers.author);

})

通过这里例子可以看出，问题就放在 prompt() 中，问题的类型为 input 就是输入类型，name 就是作为答案对象中的 key，message 就是问题了，用户输入的答案就在 answers 中，使用起来就是这么简单。更多的参数设置可以参考官方文档。

通过命令行交互，获得用户的输入，从而可以把答案渲染到模板中。

# 八、渲染模板- handlebars

这里用 handlebars 的语法对 HTML5/H5Template 仓库的模板中的 package.json 文件做一些修改，以下就是我远程Github代码仓库的 package.json 文件！

{

"name": "{{name}}",

"version": "1.0.0",

"description": "{{description}}",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "{{author}}",

"license": "ISC"

}

* 并在下载模板完成之后将用户输入的答案渲染到 package.json 中

program.version('1.0.0', '-v, --version')

.command('init <name>')

.action((name) => {

inquirer.prompt([

{

name: 'description',

message: '请输入项目描述'

},

{

name: 'author',

message: '请输入作者名称'

}

]).then((answers) => {

download('qzdev/HTML5',name,(err) => {

const meta = {

name,

description: answers.description,

author: answers.author

}

const fileName = `${name}/package.json`;

const content = fs.readFileSync(fileName).toString();

const result = handlebars.compile(content)(meta);

fs.writeFileSync(fileName, result);

})

})

});

这里使用了 node.js 的文件模块 fs，将 handlebars 渲染完后的模板重新写入到文件中。

# 九、视觉美化 - ora

* 在用户输入答案之后，开始下载模板，这时候使用 ora 来提示用户正在下载中。

const ora = require('ora');

// 开始下载

const spinner = ora('正在下载模板...');

spinner.start();

// 下载失败调用

spinner.fail();

// 下载成功调用

spinner.succeed();

然后通过 chalk 来为打印信息加上样式，比如成功信息为绿色，失败信息为红色，这样子会让用户更加容易分辨，同时也让终端的显示更加的好看。

const chalk = require('chalk');

console.log(chalk.green('项目创建成功'));

console.log(chalk.red('项目创建失败'));

除了给打印信息加上颜色之外，还可以使用 log-symbols 在信息前面加上 √ 或 × 等的图标

const chalk = require('chalk');

const symbols = require('log-symbols');

console.log(symbols.success, chalk.green('项目创建成功'));

console.log(symbols.error, chalk.red('项目创建失败'));

# 十、完整示例

//index.js

#!/usr/bin/env node

const fs = require('fs');

const program = require('commander');

const download = require('download-git-repo');

const handlebars = require('handlebars');

const inquirer = require('inquirer');

const ora = require('ora');

const chalk = require('chalk');

const symbols = require('log-symbols');

program.version('1.0.0', '-v, --version')

.command('init <name>')

.action((name) => {

if(!fs.existsSync(name)){

inquirer.prompt([

{

name: 'description',

message: '请输入项目描述'

},

{

name: 'author',

message: '请输入作者名称'

}

]).then((answers) => {

const spinner = ora('正在下载模板...');

spinner.start();

download(' qzdev/HTML5', name, (err) => {

if(err){

spinner.fail();

console.log(symbols.error, chalk.red(err));

}else{

spinner.succeed();

const fileName = `${name}/package.json`;

const meta = {

name,

description: answers.description,

author: answers.author

}

if(fs.existsSync(fileName)){

const content = fs.readFileSync(fileName).toString();

const result = handlebars.compile(content)(meta);

fs.writeFileSync(fileName, result);

}

console.log(symbols.success, chalk.green('项目初始化完成'));

}

})

})

}else{

// 错误提示项目已存在，避免覆盖原有项目

console.log(symbols.error, chalk.red('项目已存在'));

}

})

program.parse(process.argv);

留下思考：安装过程中，就会询问是否安装自己所需的软件了

参考文档链接：

http://blog.gdfengshuo.com/article/27/

https://www.cnblogs.com/wmhuang/p/7092658.html

https://github.com/vuejs/vue-cli