**教你从零开始搭建一款前端脚手架工具**

在实际的开发过程中，从零开始建立项目的结构是一件让人头疼的事情，所以各种各样的脚手架工具应运而生。笔者使用较多的yoeman，express-generator和vue-cli便是当中之一。它们功能丰富，但最核心的功能都是能够快速搭建一个完整的项目的结构，开发者只需要在生成的项目结构的基础上进行开发即可，非常简单高效。

**前言**

在实际的开发过程中，从零开始建立项目的结构是一件让人头疼的事情，所以各种各样的脚手架工具应运而生。笔者使用较多的yoeman，express-generator和vue-cli便是当中之一。它们功能丰富，但最核心的功能都是能够快速搭建一个完整的项目的结构，开发者只需要在生成的项目结构的基础上进行开发即可，非常简单高效。

作为一个不折腾会死星人，在熟悉了使用方法以后就开始琢磨起它们的原理来了。经过仔细研究文档和源码，终于算是摸清了其核心的原理，并且依据这个原理自己搭建了一款叫做SCION的脚手架。

现在让我们就以SCION为例，从零开始搭建一款属于我们自己的脚手架工具吧!

**核心原理**

yoeman搭建项目需要提供yoeman-generator。yoeman-generator本质上就是一个具备完整文件结构的项目样板，用户需要手动地把这些generator下载到本地，然后yoeman就会根据这些generator自动生成各种不同的项目。

vue-cli提供了相当丰富的选项和设定功能，但是其本质也是从远程仓库把不同的模版拉取到本地，而并非是什么“本地生成”的黑科技。

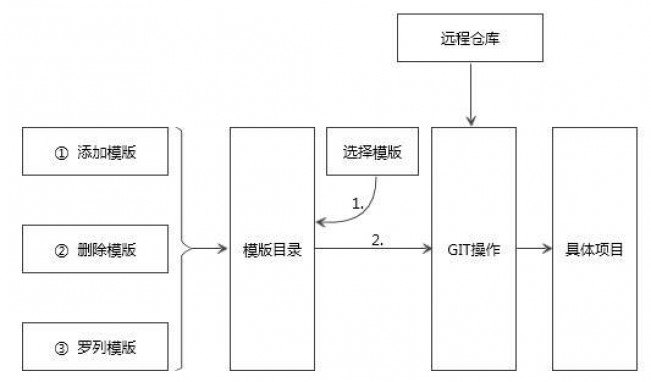
这样看来，思路也就有了——首先建立不同的样板项目，然后脚手架根据用户的指令引用样板项目生成实际项目。样板项目可以内置在脚手架当中，也可以部署在远程仓库。为了更广的适用范围，SCION采用的是第二种方式。

**技术选型**

1. node.js：整个脚手架工具的根本组成部分，推荐使用最新的版本。
2. es6：新版本的node.js对于es6的支持度已经非常高，使用es6能够极大地提升开发效率和开发感受。
3. commander：TJ大神开发的工具，能够更好地组织和处理命令行的输入。
4. co：TJ大神开发的异步流程控制工具，用更舒服的方式写异步代码。
5. co-prompt：还是TJ大神的作品……传统的命令行只能单行一次性地输入所有参数和选项，使用这个工具可以自动提供提示信息，并且分步接收用户的输入，体验类似npm init时的一步一步输入参数的过程。

**整体架构**

国际惯例，着手开发之前得先弄明白整体架构，看图：

[](http://s5.51cto.com/wyfs02/M00/85/A3/wKiom1eqxani1IB6AAAu4rWKmBI310.jpg-wh_651x-s_2632773634.jpg)

首先明白模版的概念。一个模版就是一个项目的样板，包含项目的完整结构和信息。

模版的信息都存放在一个叫做templates.json的文件当中。

用户可以通过命令行对templates.json进行添加、删除、罗列的操作。

通过选择不同的模版，SCION会自动从远程仓库把相应的模板拉取到本地，完成项目的搭建。

最终整个脚手架的文件结构如下：

1. =================
2. |\_\_ bin
3. |\_\_ scion
4. |\_\_ command
5. |\_\_ **add**.js
6. |\_\_ **delete**.js
7. |\_\_ init.js
8. |\_\_ list.js
9. |\_\_ node\_modules
10. |\_\_ package.json
11. |\_\_ templates.json

**入口文件**

首先建立项目，在package.json里面写入依赖并执行npm install：

1. "dependencies": {
2. "chalk": "^1.1.3",
3. "co": "^4.6.0",
4. "co-prompt": "^1.0.0",
5. "commander": "^2.9.0"
6. }

在根目录下建立\bin文件夹，在里面建立一个无后缀名的scion文件。这个bin\scion文件是整个脚手架的入口文件，所以我们首先对它进行编写。

首先是一些初始化的代码：

1. #!/usr/bin/env node --harmony
2. 'use strict'
3. // 定义脚手架的文件路径
4. process.env.NODE\_PATH = \_\_dirname + '/../node\_modules/'
6. const program = require('commander')
8. // 定义当前版本
9. program
10. .version(require('../package').version )
12. // 定义使用方法
13. program
14. .usage('<command>')

从前文的架构图中可以知道，脚手架支持用户输入4种不同的命令。现在我们来写处理这4种命令的方法：

1. program
2. .command('add')
3. .description('Add a new template')
4. .alias('a')
5. .**action**(() => {
6. require('../command/add')()
7. })
9. program
10. .command('list')
11. .description('List all the templates')
12. .alias('l')
13. .**action**(() => {
14. require('../command/list')()
15. })
17. program
18. .command('init')
19. .description('Generate a new project')
20. .alias('i')
21. .**action**(() => {
22. require('../command/init')()
23. })
25. program
26. .command('delete')
27. .description('Delete a template')
28. .alias('d')
29. .**action**(() => {
30. require('../command/delete')()
31. })

commander的具体使用方法在这里就不展开了，可以直接到官网去看详细的文档。

最后别忘了处理参数和提供帮助信息：

1. program.parse(process.argv)
3. if(!program.args.length){
4. program.help()
5. }

完整的代码请看这里。

使用node运行这个文件，看到输出如下，证明入口文件已经编写完成了。

1. Usage: scion <command>

4. Commands:
6. **add**|a      **Add** a new template
7. list|l     List all the templates
8. init|i     Generate a new project
9. **delete**|d   **Delete** a template
11. Options:
13. -h, --help     output usage information
14. -V, --version  output the version number

**处理用户输入**

在项目根目录下建立\command文件夹，专门用来存放命令处理文件。

在根目录下建立templates.json文件并写入如下内容，用来存放模版信息：

1. {"tpl":{}}

**添加模板**

进入\command并新建add.js文件：

1. 'use strict'
2. const co = require('co')
3. const prompt = require('co-prompt')
4. const config = require('../templates')
5. const chalk = require('chalk')
6. const fs = require('fs')
8. module.exports = () => {
9. co(**function** \*() {
11. // 分步接收用户输入的参数
12. let tplName = yield prompt('Template name: ')
13. let gitUrl = yield prompt('Git https link: ')
14. let branch = yield prompt('Branch: ')
16. // 避免重复添加
17. if (!config.tpl[tplName]) {
18. config.tpl[tplName] = {}
19. config.tpl[tplName]['url'] = gitUrl.replace(/[\u0000-\u0019]/g, '') // 过滤unicode字符
20. config.tpl[tplName]['branch'] = branch
21. } **else** {
22. console.log(chalk.red('Template has already existed!'))
23. process.exit()
24. }
26. // 把模板信息写入templates.json
27. fs.writeFile(\_\_dirname + '/../templates.json', JSON.stringify(config), 'utf-8', (err) => {
28. if (err) console.log(err)
29. console.log(chalk.green('New template added!\n'))
30. console.log(chalk.grey('The last template list is: \n'))
31. console.log(config)
32. console.log('\n')
33. process.exit()
34. })
35. })
36. }

**删除模板**

同样的，在\command文件夹下建立delete.js文件：

1. 'use strict'
2. const co = require('co')
3. const prompt = require('co-prompt')
4. const config = require('../templates')
5. const chalk = require('chalk')
6. const fs = require('fs')
8. module.exports = () => {
9. co(**function** \*() {
10. // 接收用户输入的参数
11. let tplName = yield prompt('Template name: ')
13. // 删除对应的模板
14. if (config.tpl[tplName]) {
15. config.tpl[tplName] = undefined
16. } **else** {
17. console.log(chalk.red('Template does not exist!'))
18. process.exit()
19. }
21. // 写入template.json
22. fs.writeFile(\_\_dirname + '/../templates.json', JSON.stringify(config),     'utf-8', (err) => {
23. if (err) console.log(err)
24. console.log(chalk.green('Template deleted!'))
25. console.log(chalk.grey('The last template list is: \n'))
26. console.log(config)
27. console.log('\n')
28. process.exit()
29. })
30. })
31. }

**罗列模板**

建立list.js文件：

1. 'use strict'
2. const config = require('../templates')
4. module.exports = () => {
5. console.log(config.tpl)
6. process.exit()
7. }

**构建项目**

现在来到我们最重要的部分——构建项目。同样的，在\command目录下新建一个叫做init.js的文件：

1. 'use strict'
2. const **exec** = require('child\_process').**exec**
3. const co = require('co')
4. const prompt = require('co-prompt')
5. const config = require('../templates')
6. const chalk = require('chalk')
8. module.exports = () => {
9. co(**function** \*() {
10. // 处理用户输入
11. let tplName = yield prompt('Template name: ')
12. let projectName = yield prompt('Project name: ')
13. let gitUrl
14. let branch
16. if (!config.tpl[tplName]) {
17. console.log(chalk.red('\n × Template does not exit!'))
18. process.exit()
19. }
20. gitUrl = config.tpl[tplName].url
21. branch = config.tpl[tplName].branch
23. // git命令，远程拉取项目并自定义项目名
24. let cmdStr = `git clone ${gitUrl} ${projectName} && cd ${projectName} && git checkout ${branch}`
26. console.log(chalk.white('\n Start generating...'))
28. **exec**(cmdStr, (error, stdout, stderr) => {
29. if (error) {
30. console.log(error)
31. process.exit()
32. }
33. console.log(chalk.green('\n √ Generation completed!'))
34. console.log(`\n cd ${projectName} && npm install \n`)
35. process.exit()
36. })
37. })
38. }

可以看到，这一部分代码也非常简单，关键的一句话是

1. let cmdStr = `git clone ${gitUrl} ${projectName} && cd ${projectName} && git checkout ${branch}`

它的作用正是从远程仓库克隆到自定义目录，并切换到对应的分支。熟悉git命令的同学应该明白，不熟悉的同学是时候补补课啦!

全局使用

为了可以全局使用，我们需要在package.json里面设置一下：

1. "bin": {
2. "scion": "bin/scion"
3. },

本地调试的时候，在根目录下执行

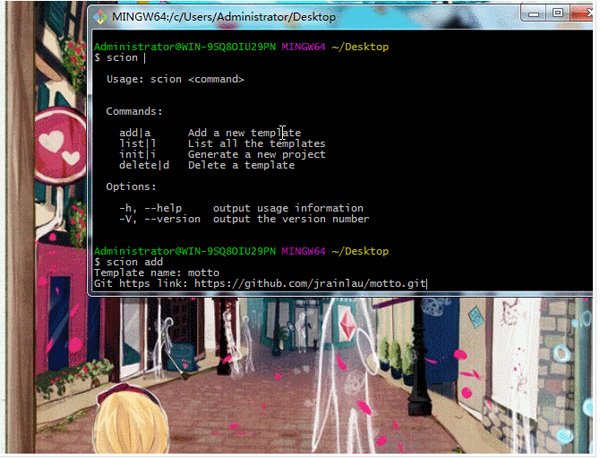
1. npm link

即可把scion命令绑定到全局，以后就可以直接以scion作为命令开头而无需敲入长长的node scion之类的命令了。

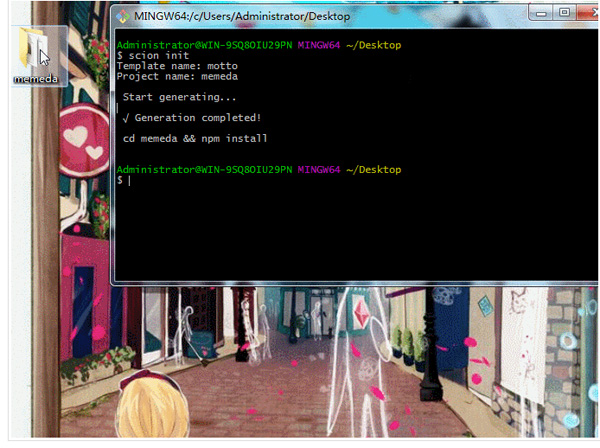
现在我们的脚手架工具已经搭建好了，一起来尝试一下吧!

**使用测试**

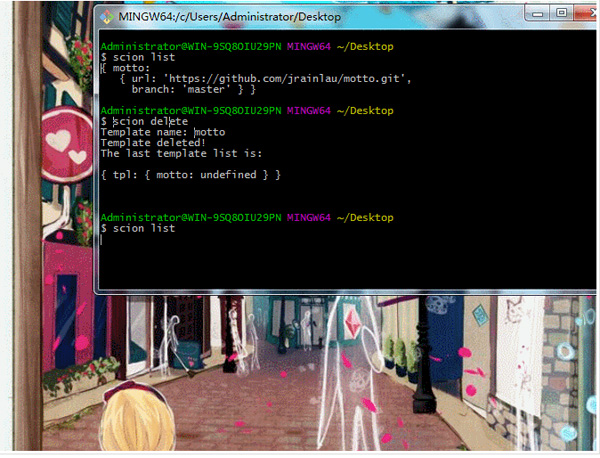
* add | a 添加模版命令

[](http://s2.51cto.com/wyfs02/M01/85/A4/wKiom1eq2S6jeJxhAAKfoUP_a2U520.jpg)

* init | i 生成项目命令

[](http://s2.51cto.com/wyfs02/M02/85/A4/wKiom1eq2T-Agag8AAI79Dn2RiU623.jpg)

* delete | d 删除模版命令 和 list | l 罗列模版命令

[](http://s1.51cto.com/wyfs02/M01/85/A4/wKioL1eq2VGA-dl5AAKKCXUJxZQ623.jpg)

大功告成啦!现在我们的整个脚手架工具已经搭建完成了，以后只需要知道模板的git https地址和branch就可以不断地往SCION上面添加，团队协作的话只需要分享SCION的templates.json文件就可以了。

**后记**

看起来并不复杂的东西，实际从零开始搭建也是颇费了一番心思。最大的难题是在开始的时候并不懂得如何像npm init那样可以一步一步地处理用户输入，只懂得一条命令行把所有的参数都带上，这样的用户体验真的很不好。研究了vue-cli和yoeman也没有找到相应的代码，只好不断地google，最后总算找到了一篇文章，可以用co和co-prompt这两个工具实现，再一次膜拜无所不能的TJ大神，也希望能够有小伙伴告诉我vue-cli它们是怎么实现的。

这个脚手架只具备最基本的功能，还远远没有达到市面上同类产品的高度，在日后再慢慢填补吧，不管怎么说，完成SCION的过程中真的学习到了很多东西。

感谢你的阅读。

【编辑推荐】

1. [JavaScript 节流函数 Throttle 详解](http://developer.51cto.com/art/201606/513518.htm)
2. [微软向其它JavaScript引擎开放Node.js](http://developer.51cto.com/art/201608/515449.htm)
3. [在JavaScript中，如何判断数组是数组？(alpha)](http://developer.51cto.com/art/201608/515537.htm)
4. [如何打造一个令人愉悦的前端开发环境（一）](http://developer.51cto.com/art/201608/515555.htm)
5. [GraphQL的JavaScript实现graphql-js应用举例](http://developer.51cto.com/art/201608/515612.htm)