脚手架按照我的理解也就是工厂，这个工厂里提供了脚手架制作人所提供的开发工具，配置了脚手架开发人给使用者的开发步骤，配置了整个工厂生产的流水线，也就是开发线路，（脚手架使用的命令）。

# 如何搭建脚手架？

## 搭建脚手架所用工具：node.js，npm

## 搭建脚手架所用工具类

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

在一个空文件夹下新建一个 .js文件，用于写生成脚手架的代码

npm init 生成 package.json文件

npm i 上面所要用的的工具类

{

"name": "zy-cli", “脚手架名”

"version": "1.0.0", “定义的脚手架版本”

"description": "xxxx", “给脚手架的描述”

"bin": {

"zy": "index.js" 在bin中定义脚手架的命令与命令执行的文件

},

}

## const program = require('commander');处理交互命令与解析命令参数

node.js命令行界面的完整解决方案,支持链式结构

安装 npm install commander –save

### 规定命令

program.version('0.0.1', '-v, --version’)规定命令-v,--vision命令为版本信息v1.1.0;

program.option(‘-p, --peppers’, ‘xxx’);规定命令-p,--peppers命令所得为xxx

program.parse(process.argv); 转义为process.argv命令数组-a 相当于单个字符传递，-abc = -a –b –c, --abc相当于整个字符串== -abc，--abc-a会变成驼峰-abcA，--no前缀开头的多词选项是其后选项的布尔值的反

### 自定义命令

<>表示必填[]表示非必填

lias() 定义指令名称的别称，效果与指令相同；

description() 定义指令的描述信息；

option() 定义指令后面的跟的选项信息，如：$ npm install . -g；

action() 处理指令以及选项的参数函数；

parse() 获取命令行工具的所有参数进行解析；

help() 打印帮助信息;

program.args 储存使用的指令，如：$ npm install，此时 program.args = ['install']

var program = require('commander');

program

.version('0.0.1')

.command('install [packagename]')

.alias('i')

.description('install command can setup package')

.option('-g, -global [env]', 'setup package in global env')

.action(function (packagename, options) {

console.log('指令 install 后面跟的参数值 package: ' + packagename);

console.log('选项 -g 后面跟的参数值 env: ' + env);

// 获得了参数，可以在这里做响应的业务处理

})

.parse(process.argv); // parse() 方法可以获得命令行的所有参数

if (!program.args.length) program.help();

执行命令时，将验证该命令的options，任何未知的option都将报错。 但是，如果基于action的命令如果没有定义action，则不验证options。

### ****定义命令行工具能够提供哪些选项****

program.option(flag, desc, fn, defaultValue)

flag 必填。定义命令行参数能够提供哪些参数；

desc 非必填。对参数进行描述；

fn 非必填。是一个函数；

defaultValue 非必填。默认值;

<> 符号表示参数必填，不填会报错;

[] 符号表示参数非必填。

### ****自定义帮助信息****

program.on('--help', function(){

console.log(' Examples:');

console.log('');

console.log(' $ custom-help --help');

console.log(' $ custom-help -h');

console.log('');

});

## const download = require('download-git-repo'); 从 Github、Gitlab 和 Bitbucket 下载仓库

<https://www.npmjs.com/package/download-git-repo>

GitHub： github:owner/name or owner/name

GitLab： gitlab:owner/name

Bitbucket： bitbucket:owner/name

download(‘[github/gitlab/Bitbucket]:[账户名]/[仓库名] #分支名’，‘下载到那个文件夹’，{clone: true}，（err）=> {});

## inquirer = require('inquirer'); 用户交互，交互信息处理

### inquirer参数

type：表示提问的类型，包括：input, confirm, list, rawlist, expand, checkbox, password, editor；

name: 存储当前问题回答的变量；

message：问题的描述；

default：默认值；

choices：列表选项，在某些type下可用，并且包含一个分隔符(separator)；

validate：对用户的回答进行校验；

filter：对用户的回答进行过滤处理，返回处理后的值；

transformer：对用户回答的显示效果进行处理(如：修改回答的字体或背景颜色)，但不会影响最终的答案的内容；

when：根据前面问题的回答，判断当前问题是否需要被回答；

pageSize：修改某些type类型下的渲染行数；

prefix：修改message默认前缀；

suffix：修改message默认后缀。

const promptList = [

{

type: 'input',

message: '设置一个用户名:',

name: 'name',

default: "test\_user" // 默认值

validate: function(val) {

if(val.match(/\d{11}/g)) { // 校验位数

return val;

}

return "请输入11位数字";

}

},

{

type: "confirm",

message: "是否使用监听？",

name: "watch",

prefix: "前缀" 在message前后打印

suffix: "后缀",

when: function(answers) {

return answers.name

}

},

{

type: 'list',

message: '请选择一种水果:',

name: 'fruit',

choices: [

"Apple",

"Pear",

"Banana"

],

filter: function (val) { // 使用filter将回答变为小写

return val.toLowerCase();

}

}，

{

type: 'rawlist',

message: '请选择一种水果:',

name: 'fruit',

choices: [

"Apple",

"Pear",

"Banana"

]

},

{

type: "expand",

message: "请选择一种水果：",

name: "fruit",

choices: [

{

key: "a",

name: "Apple",

value: "apple"

},

{

key: "O",

name: "Orange",

value: "orange"

},

{

key: "p",

name: "Pear",

value: "pear"

}

]

},

{

type: "checkbox",

message: "选择颜色:",

name: "color",

choices: [

{

name: "red"

},

new inquirer.Separator(), // 添加分隔符

{

name: "blur",

checked: true // 默认选中

},

{

name: "green"

},

new inquirer.Separator("--- 分隔符 ---"), // 自定义分隔符

{

name: "yellow"

}

]

}];

// 或者下面这样

const promptList = [{

type: "checkbox",

message: "选择颜色:",

name: "color",

choices: [

"red",

"blur",

"green",

"yellow"

],

pageSize: 2 // 设置行数

},

{

type: "password", // 密码为密文输入

message: "请输入密码：",

name: "pwd"

},

{

type: "editor",

message: "请输入备注：",

name: "editor"

}

];

inquirer.prompt(promptList).then(answers )=> {

console.log(answers); // 返回的结果

})

## handlebars对 HTML5/H5Template 仓库的模板中的 package.json 文件做一些修改

### package.json介绍

scripts字段指定了运行脚本命令的npm命令行缩写，比如start指定了运行npm run start时，所要执行的命令。

"scripts": {

"preinstall": "echo here it comes!",

"postinstall": "echo there it goes!",

"start": "node index.js",

"test": "tap test/\*.js"

}

dependencies字段指定了项目运行所依赖的模块，devDependencies指定项目开发所需要的模块。它们都指向一个对象。该对象的各个成员，分别由模块名和对应的版本要求组成，表示依赖的模块及其版本范围。指定版本：比如1.2.2，遵循“大版本.次要版本.小版本”的格式规定，安装时只安装指定版本。

波浪号（tilde）+指定版本：比如~1.2.2，表示安装1.2.x的最新版本（不低于1.2.2），但是不安装1.3.x，也就是说安装时不改变大版本号和次要版本号。

插入号（caret）+指定版本：比如ˆ1.2.2，表示安装1.x.x的最新版本（不低于1.2.2），但是不安装2.x.x，也就是说安装时不改变大版本号。需要注意的是，如果大版本号为0，则插入号的行为与波浪号相同，这是因为此时处于开发阶段，即使是次要版本号变动，也可能带来程序的不兼容。

latest：安装最新版本。

{

"devDependencies": {

"browserify": "~13.0.0",

"karma-browserify": "~5.0.1"

}

}

使用npm i xxxx –save自动更新dependencies字段值；

使用npm i xxxxx –save-dev自动更新devDependencies字段值；

devDependencies下列出的模块，是我们开发时用的依赖项，像一些进行单元测试之类的包，比如grunt-contrib-uglify，我们用它混淆js文件，它们不会被部署到生产环境。dependencies下的模块，则是我们生产环境中需要的依赖，即正常运行该包时所需要的依赖项。

peerDependencies字段，就是用来供插件指定其所需要的主工具的版本。

{

"name": "chai-as-promised",

"peerDependencies": {

"chai": "1.x"

}

}

bin项用来指定各个内部命令对应的可执行文件的位置。

"bin": {

"someTool": "./bin/someTool.js"

}

main字段指定了加载的入口文件，require('moduleName')就会加载这个文件。这个字段的默认值是模块根目录下面的index.js。

config字段用于添加命令行的环境变量。

browser指定该模板供浏览器使用的版本

engines字段指明了该模块运行的平台

man用来指定当前模块的man文档的位置。

preferGlobal的值是布尔值，表示当用户不将该模块安装为全局模块时（即不用–global参数），要不要显示警告，表示该模块的本意就是安装为全局模块。

style指定供浏览器使用时，样式文件所在的位置。样式文件打包工具parcelify，通过它知道样式文件的打包位置。

## const ora = require('ora'); 下载模板进度视觉美化

<https://www.npmjs.com/package/ora>

const spinner = ora(‘text’) 创建实例

spinner.start(‘text’) 开始转轮

spinner.stop(‘text’) 转轮停止

spinner.succeed(‘‘text’’) 转轮停止打钩

spinner.fail(‘text’)转轮失败打叉

在转轮为停止之前可以改变转轮状态

spinner.color = ‘yellow’

spinner.text = ‘other text’

## const chalk = require('chalk');为打印的信息加上颜色

https://www.npmjs.com/package/chalk

## const symbols = require('log-symbols');为打印的信息加上勾，×

console.log(logSymbols.success);