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": "okii-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

program.command(‘zy <dir>’)

.option(‘-r, --recursive’, ‘xxx’)

.action((dir, program) => {

//调了zy后后面的命令

//program.recursive

}).parse(process.argv);

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。

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

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

## const handlebars = require('handlebars');对 HTML5/H5Template 仓库的模板中的 package.json 文件做一些修改

## HTML5/H5Template 仓库的模板

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

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

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