# 手撸一个自己的前端脚手架

很多小伙伴一直很纠结什么是脚手架？其实核心功能就是创建项目初始文件,那问题又来了,市面上的脚手架不够用？为什么还要自己写？ （珠峰架构）

只要提到脚手架你就会想到，*vue-cli* 、 *create-react-app* 、 *dva-cli* ... 他们的特点不用多说那就是**专一**！但是在公司中你会发现有以下一系列的问题!

- 业务**类型多**
- 多次造轮子，**项目升级**等问题
- 公司代码**规范**,无法统一

在自己开发cli前，那肯定先要看些优秀的cli是如何实现的！虽然不是第一个吃螃蟹的，那也要想想怎么吃更好^_^# （珠峰架构）

## 1.必备模块

我们先从大家众所周知的vue-cli入手,先来看看他用了哪些**npm包**来实现的

- **commander** ： 参数解析 --help其实就借助了他~
- **inquirer** ： 交互式命令行工具，有他就可以实现命令行的选择功能
- **download-git-repo** ： 在git中下载模板
- **chalk** ： 粉笔帮我们在控制台中画出各种各样的颜色
- **metalsmith** ： 读取所有文件,实现模板渲染
- **consolidate** ： 统一模板引擎 （珠峰架构）

先幻想下要实现的功能:

根据模板初始化项目 `zhu-cli create project-name`
初始化配置文件 `zhu-cli config set repo repo-name`

## 2.工程创建

废话不多说我们开始创建项目，编写自己的脚手架~~~ （珠峰架构）

```
npm init -y # 初始化package.json
npm install eslint husky --save-dev # eslint是负责代码校验工作,husky提供了git钩子功能
npx eslint --init # 初始化eslint配置文件
```

### 2.1 创建文件夹

```
├── bin
|   └── www  // 全局命令执行的根文件
├── package.json
├── src
|   ├── main.js // 入口文件
|   └── utils   // 存放工具方法
|── .huskyrc    // git hook
|── .eslintrc.json // 代码规范校验
```

### 2.2 eslint配置

配置package.json 校验src文件夹下的代码

```
"scripts": {
    "lint":"eslint src"
}
```

## 2.3 配置husky

当使用git提交前校验代码是否符合规范

```
{
  "hooks": {
    "pre-commit": "npm run lint"
  }
}
```

## 2.4 链接全局包

设置在命令下执行zhu-cli时调用bin目录下的www文件 （珠峰架构）

```
"bin": {
    "zhu-cli": "./bin/www"
}
```

www文件中使用main作为入口文件，并且以node环境执行此文件

```
#! /usr/bin/env node
require('../src/main.js');
```

链接包到全局下使用 （珠峰架构）

```
npm link
```

我们已经可以成功的在命令行中使用`zhu-cli`命令，并且可以执行main.js文件!

# 3.解析命令行参数

**commander:The complete solution for node.js command-line interfaces**

先吹一波commander,commander可以自动生成help，解析选项参数！ （珠峰架构）

像这样 `vue-cli --help!`
像这样 `vue-cli create <project-namne>`

## 3.1 使用commander

```
npm install commander
```

main.js就是我们的入口文件

```
const program = require('commander');

program.version('0.0.1')
  .parse(process.argv); // process.argv就是用户在命令行中传入的参数
```

执行`zhu-cli --help`是不是已经有一提示了！ （珠峰架构）

这个版本号应该使用的是当前cli项目的版本号，我们需要动态获取，并且为了方便我们将常量全部放到util下的`constants`文件夹中 （珠峰架构）

```
const { name, version } = require('../../package.json');

module.exports = {
  name,
  version,
};
```

这样我们就可以**动态获取版本号**了

```
const program = require('commander');

const { version } = require('./utils/constants');

program.version(version)
  .parse(process.argv);
```

## 3.2 配置指令命令

根据我们想要实现的功能配置执行动作，遍历产生对应的命令

```
const actionsMap = {
  create: { // 创建模板
    description: 'create project',
    alias: 'cr',
    examples: [
      'zhu-cli create <template-name>',
    ],
  },
  config: { // 配置配置文件
    description: 'config info',
    alias: 'c',
    examples: [
      'zhu-cli config get <k>',
      'zhu-cli config set <k> <v>',
    ],
  },
  '*': {
    description: 'command not found',
  },
};
// 循环创建命令
Object.keys(actionsMap).forEach((action) => {
  program
    .command(action) // 命令的名称
    .alias(actionsMap[action].alias) // 命令的别名
```

```
    .description(actionsMap[action].description) // 命令的描述
    .action(() => { // 动作
      console.log(action);
    });
});

program.version(version)
  .parse(process.argv);
```

## 3.3 编写help命令

监听help命令打印帮助信息　（珠峰架构）

```
program.on('--help', () => {
  console.log('Examples');
  Object.keys(actionsMap).forEach((action) => {
    (actionsMap[action].examples || []).forEach((example) => {
      console.log(`  ${example}`);
    });
  });
});
```

到现在我们已经把命令行配置的很棒啦，接下来就开始实现对应的功能!

# 4.create命令

create命令的主要作用就是去git仓库中拉取模板并下载对应的版本到本地，如果有模板则根据用户填写的信息渲染好模板，生成到当前运行命令的目录下~

```
action(() => { // 动作
  if (action === '*') { // 如果动作没匹配到说明输入有误
    console.log(acitonMap[action].description);
  } else { // 引用对应的动作文件 将参数传入
    require(path.resolve(__dirname, action))(...process.argv.slice(3));
  }
}
```

根据不同的动作，动态引入对应模块的文件　（珠峰架构）

创建create.js

```
// 创建项目
module.exports = async (projectName) => {
  console.log(projectName);
};
```

执行`zhu-cli create project`,可以打印出 `project`

## 4.1 拉取项目

我们需要获取仓库中的所有模板信息，我的模板全部放在了git上，这里就以git为例，我通过axios去获取相关的信息~ ~ ~　（珠峰架构）

```
npm i axios
```

这里借助下github的 api

```
const axios = require('axios');
// 1).获取仓库列表
const fetchRepoList = async () => {
  // 获取当前组织中的所有仓库信息,这个仓库中存放的都是项目模板
  const { data } = await axios.get('https://api.github.com/orgs/zhu-cli/repos');
  return data;
};

module.exports = async (projectName) => {
  let repos = await fetchRepoList();
  repos = repos.map((item) => item.name);
  console.log(repos)
};
```

发现在安装的时候体验很不好没有任何提示,而且最终的结果我希望是可以供用户选择的!

## 4.2 inquirer & ora

我们来解决上面提到的问题　（珠峰架构）

```
npm i inquirer ora
```

```
module.exports = async (projectName) => {
  const spinner = ora('fetching repo list');
  spinner.start(); // 开始loading
  let repos = await fetchRepoList();
  spinner.succeed(); // 结束loading

  // 选择模板
  repos = repos.map((item) => item.name);
  const { repo } = await Inquirer.prompt({
    name: 'repo',
    type: 'list',
    message: 'please choice repo template to create project',
    choices: repos, // 选择模式
  });
  console.log(repo);
};
```

我们看到的命令行中选择的功能基本都是基于inquirer实现的，可以实现不同的询问方式

## 4.3 获取版本信息

和获取模板一样，我们可以故技重施　（珠峰架构）

```
const fetchTagList = async (repo) => {
  const { data } = await axios.get(`https://api.github.com/repos/zhu-
cli/${repo}/tags`);
  return data;
};
// 获取版本信息
spinner = ora('fetching repo tags');
spinner.start();
```

```
let tags = await fetchTagList(repo);
spinner.succeed(); // 结束loading

// 选择版本
tags = tags.map((item) => item.name);
const { tag } = await Inquirer.prompt({
  name: 'tag',
  type: 'list',
  message: 'please choice repo template to create project',
  choices: tags,
});
```

我们发现每次都需要去开启loading、关闭loading，重复的代码当然不能放过啦！我们来简单的封装下

```
const wrapFetchAddLoding = (fn, message) => async (...args) => {
  const spinner = ora(message);
  spinner.start(); // 开始loading
  const r = await fn(...args);
  spinner.succeed(); // 结束loading
  return r;
};
// 这回用起来舒心多了～～～
let repos = await wrapFetchAddLoding(fetchRepoList, 'fetching repo list')();
let tags = await wrapFetchAddLoding(fetchTagList, 'fetching tag list')(repo);
```

## 4.4 下载项目

我们已经成功获取到了项目模板名称和对应的版本，那我们就可以直接下载啦！ （珠峰架构）

```
npm i download-git-repo
```

很遗憾的是这个方法不是promise方法，没关系我们自己包装一下

```
const { promisify } = require('util');
const downLoadGit = require('download-git-repo');
downLoadGit = promisify(downLoadGit);
```

node中已经帮你提供了一个现成的方法，将异步的api可以快速转化成promise的形式～

下载前先找个临时目录来存放下载的文件,来～继续配置常量 （珠峰架构）

```
const downloadDirectory = `${process.env[process.platform === 'darwin' ? 'HOME'
: 'USERPROFILE']}/.template`;
```

这里我们将文件下载到当前用户下的`.template`文件中,由于系统的不同目录获取方式不一样,`process.platform`在windows下获取的是`win32`我这里是mac 所有获取的值是`darwin`,在根据对应的环境变量获取到用户目录

```
const download = async (repo, tag) => {
  let api = `zhu-cli/${repo}`; // 下载项目
  if (tag) {
    api += `#${tag}`;
  }
  const dest = `${downloadDirectory}/${repo}`; // 将模板下载到对应的目录中
  await downLoadGit(api, dest);
  return dest; // 返回下载目录
};


// 下载项目
const target = await wrapFetchAddLoding(download, 'download template')(repo,
tag);
```

如果对于简单的项目可以直接把下载好的项目拷贝到当前执行命令的目录下即可。

安装`ncp`可以实现文件的拷贝功能 （珠峰架构）

```
npm i ncp
```

像这样:

```
let ncp = require('ncp');
ncp = promisify(ncp);
// 将下载的文件拷贝到当前执行命令的目录下
await ncp(target, path.join(path.resolve(), projectName));
```

当然这里可以做的更严谨一些，判断一下当前目录下是否有重名文件等...，还有很多细节也需要考虑像多次创建项目是否要利用已经下载好的模板，大家可以自由的发挥～

## 4.5 模板编译

刚才说的是简单文件，那当然直接拷贝就好了，但是有的时候用户可以定制下载模板中的内容，拿`package.json`文件为例，用户可以根据提示给项目命名、设置描述等 （珠峰架构）

这里我在项目模板中增加了ask.js

```
module.exports = [
    {
      type: 'confirm',
      name: 'private',
      message: 'ths resgistery is private?',
    },
    ...
]
```

根据对应的询问生成最终的`package.json`

下载的模板中使用了`ejs`模板 （珠峰架构）

```
{
  "name": "vue-template",
  "version": "0.1.2",
  "private": "<%=private%>",
```

```
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build"
  },
  "dependencies": {
    "vue": "^2.6.10"
  },
  "autor":"<%=author%>",
  "description": "<%=description%>",
  "devDependencies": {
    "@vue/cli-service": "^3.11.0",
    "vue-template-compiler": "^2.6.10"
  },
  "license": "<%=license%>"
}
```

> 写到这里，大家应该想到了！核心原理就是将下载的模板文件，依次遍历根据用户填写的信息渲染模板，将渲染好的结果拷贝到执行命令的目录下

安装需要用到的模块 （珠峰架构）

```
npm i metalsmith ejs consolidate
```

```
const MetalSmith = require('metalsmith'); // 遍历文件夹
let { render } = require('consolidate').ejs;
render = promisify(render); // 包装渲染方法

// 没有ask文件说明不需要编译
if (!fs.existsSync(path.join(target, 'ask.js'))) {
  await ncp(target, path.join(path.resolve(), projectName));
} else {
  await new Promise((resovle, reject) => {
    MetalSmith(__dirname)
      .source(target) // 遍历下载的目录
      .destination(path.join(path.resolve(), projectName)) // 输出渲染后的结果
      .use(async (files, metal, done) => {
        // 弹框询问用户
        const result = await Inquirer.prompt(require(path.join(target,
'ask.js')));
        const data = metal.metadata();
        Object.assign(data, result); // 将询问的结果放到metadata中保证在下一个中间件中
可以获取到
        delete files['ask.js'];
        done();
      })
      .use((files, metal, done) => {
        Reflect.ownKeys(files).forEach(async (file) => {
          let content = files[file].contents.toString(); // 获取文件中的内容
          if (file.includes('.js') || file.includes('.json')) { // 如果是js或者
json才有可能是模板
            if (content.includes('<%')) { // 文件中用<% 我才需要编译
              content = await render(content, metal.metadata()); // 用数据渲染模板
              files[file].contents = Buffer.from(content); // 渲染好的结果替换即可
            }
          }
        });
```

```
        done();
      })
      .build((err) => { // 执行中间件
        if (!err) {
          resovle();
        } else {
          reject();
        }
      });
  });
}
```

> 这里的逻辑就是上面描述的那样，实现了模板替换！到此安装项目的功能就完成了，我们发现这里面所有用到的地址的路径都写死了，我们希望这是一个更通用的脚手架，可以让用户自己配置拉取的地址～

# 5.config命令

新建config.js 主要的作用其实就是配置文件的读写操作，当然如果配置文件不存在需要提供默认的值，先来编写常量 （珠峰架构）

`constants.js`的配置

```
const configFile = `${process.env[process.platform === 'darwin' ? 'HOME' :
'USERPROFILE']}/.zhurc`; // 配置文件的存储位置
const defaultConfig = {
  repo: 'zhu-cli', // 默认拉取的仓库名
};
```

编写`config.js`

```
const fs = require('fs');
const { defaultConfig, configFile } = require('./util/constants');
module.exports = (action, k, v) => {
  if (action === 'get') {
    console.log('获取');
  } else if (action === 'set') {
    console.log('设置');
  }
  // ...
};
```

一般`rc`类型的配置文件都是`ini`格式也就是:

```
repo=zhu-cli
register=github
```

下载`ini`模块解析配置文件 （珠峰架构）

```
npm i ini
```

这里的代码很简单，无非就是文件操作了

```
const fs = require('fs');
const { encode, decode } = require('ini');
const { defaultConfig, configFile } = require('./util/constants');

const fs = require('fs');
const { encode, decode } = require('ini');
const { defaultConfig, configFile } = require('./util/constants');

module.exports = (action, k, v) => {
  const flag = fs.existsSync(configFile);
  const obj = {};
  if (flag) { // 配置文件存在
    const content = fs.readFileSync(configFile, 'utf8');
    const c = decode(content); // 将文件解析成对象
    Object.assign(obj, c);
  }
  if (action === 'get') {
    console.log(obj[k] || defaultConfig[k]);
  } else if (action === 'set') {
    obj[k] = v;
    fs.writeFileSync(configFile, encode(obj)); // 将内容转化ini格式写入到字符串中
    console.log(`${k}=${v}`);
  } else if (action === 'getVal') {
    return obj[k];
  }
};
```

`getVal`这个方法是为了在执行create命令时可以获取到配置变量 （珠峰架构）

```
const config = require('./config');
const repoUrl = config('getVal', 'repo');
```

这样我们可以将create方法中所有的`zhu-cli`全部用获取到的值替换掉啦！

到此基本核心的方法已经ok！剩下的大家可以自行扩展啦！

# 6.项目发布

终于走到最后一步啦,我们将项目推送`npm`上,流程不再赘述啦！

```
nrm use npm
npm publish # 已经发布成功～～
```

可以通过`npm install zhu-cli -g`进行安装啦！ （珠峰架构）

咨询最新课程可以添加课程顾问

老师微信