完成一个CLI工具

2020全栈蜕变之路 - 知识脑图

https://www.processon.com/view/link/5e89ab04e4b0a1e6dcb47e67

课程目标

- 完成一个CLI工具
- 尤大神推荐的no webpack工具vite
- Node的替代者Deno

为什么前端还要学Node

- 跨域、Token鉴权、Restful XSS CSRF 风格都是什么?
- 如何只用JS开发一个全栈应用?
- 如何实现像Nuxt那样的约定路由功能?约定菜单是否可行?

学习Node的目标

- 优秀的前端 - 可以和后端有效沟通
- 敏捷的全栈 - 快速开发全栈应用
- 架构师 - 践行工程化思想 + 看 用编自己

Web Application的阶段

- 后端模板 JSP ASP PHP
- 前后端分离 Jquery
- 工程化 ng webpack glup
- 全栈时代
- 云 + 端时代 - 体验 微信云开发 serverless

CLI

创建工程

```
mkdir vue-auto-router-cli
cd vue-auto-router-cli
npm init -y
npm i commander download-git-repo ora
handlebars figlet clear chalk open watch -s
```

```
# bin/kkb.js
#指定脚本解释器为node
#!/usr/bin/env node
console.log('cli.....')

# package.json
"bin": {
  "kkb": "./bin/kkb.js"
},
```

```
# 将npm 模块链接到对应的运行项目中去
npm link

# 删除的情况
ls /usr/local/bin/
rm /usr/local/bin/kkb
```

定制命令行界面

> commander.js

kkb.js文件

```
#!/usr/bin/env node
const program = require('commander')
program.version(require('../package').versi
on)

program
    .command('init <name>')
    .description('init project')
    .action(name => {
      console.log('init ' + name)
})

program.parse(process.argv)
```

打印欢迎界面

/lib/init.js

```
const {promisify} = require('util')
const figlet = promisify(require('figlet'))
const clear = require('clear')
const chalk = require('chalk')
const log = content =>
console.log(chalk.green(content))
module.exports = async name => {
    // 打印欢迎画面
    clear()
    const data = await figlet('KKB
Welcome')
    log(data)
}
```

```
// bin/kkb.js
program
    .command('init <name>')
    .description('init project')
    .action(require('../lib/init'))
```

克隆脚手架

/lib/download.js

```
const {promisify} = require('util')
module.exports.clone = async
function(repo,desc) {
    const download =
promisify(require('download-git-repo'))
    const ora = require('ora')
    const process = ora(`下载.....${repo}`)
    process.start()
    await download(repo, desc)
    process.succeed()
}
```

/lib/init.js

```
const {clone} = require('./download')
module.exports.init = async name => {
    // console.log('init ' + name)
    log('🚀创建项目:' + name)
    // 从github克隆项目到指定文件夹
    await clone('github:su37josephxia/vue-
template', name)
}
```

安装依赖

```
// promisiy化spawn
// 对接输出流
const spawn = async (...args) => {
    const { spawn } =
require('child_process');
    return new Promise(resolve => {
        const proc = spawn(...args)
        proc.stdout.pipe(process.stdout)
        proc.stderr.pipe(process.stderr)
        proc.on('close', () => {
            resolve()
        })
    })
}

module.exports.init = async name => {

    // ....
    log('安装依赖')
    await spawn('cnpm', ['install'], { cwd:
`./${name}` })
    log(chalk.green(`
👌安装完成:
To get Start:
===========================
    cd ${name}
```

```
    npm run serve
============================
            `))

}
```

约定路由功能

- loader 文件扫描
- 代码模板渲染 hbs Mustache风格模板

/lib/refresh.js

```
const fs = require('fs')
const handlebars = require('handlebars')
const chalk = require('chalk')
module.exports = async () => {

    // 获取页面列表
    const list =
        fs.readdirSync('./src/views')
```

```
            .filter(v => v !== 'Home.vue')
            .map(v => ({
                name: v.replace('.vue',
'').toLowerCase(),
                file: v
            }))

    // 生成路由定义
    compile({
        list
    }, './src/router.js',
'./template/router.js.hbs')

    // 生成菜单
    compile({
        list
    }, './src/App.vue',
'./template/App.vue.hbs')

    /**
     * 编译模板文件
     * @param meta 数据定义
     * @param filePath 目标文件路径
     * @param templatePath 模板文件路径
     */
    function compile(meta, filePath,
templatePath) {
```

```
        if (fs.existsSync(templatePath)) {
            const content =
fs.readFileSync(templatePath).toString();
            const result =
handlebars.compile(content)(meta);
            fs.writeFileSync(filePath,
result);
        }
        console.log(chalk.green(`🚀
${filePath} 创建成功`))
    }
}
```

/bin/kkb

```
program
    .command('refresh')
    .description('refresh routers...')
    .action(require('../lib/refresh'))
```

代码伺服监测(了解)

- webpack loader是否更加合理

```
npm i watch -s
```

lib/serve

```
const spawn = (...args) => {
    const { spawn } =
require('child_process');
    const proc = spawn(...args)
    proc.stdout.pipe(process.stdout)
    proc.stderr.pipe(process.stderr)
    return proc
}

module.exports = async () => {
    const watch = require('watch')
    let process
    let isRefresh = false
    watch.watchTree('./src', async (f) => {
        console.log('change...', f)
        if (!isRefresh) {
            isRefresh = true
            process && process.kill()
            await require('./refresh')()
            setTimeout(() => { isRefresh =
false }, 5000)
            process = spawn('npm', ['run',
'serve'])
        }
    })
```

```
}
```

发布npm

```
#!/usr/bin/env bash
npm config get registry # 检查仓库镜像库
npm config set
registry=http://registry.npmjs.org
echo '请进行登录相关操作: '
npm login # 登陆
echo "-------publishing-------"
npm publish # 发布
npm config set
registry=https://registry.npm.taobao.org #
设置为淘宝镜像
echo "发布完成"
exit
```

Vite

Vue3.0相关资料

https://www.josephxia.com/vue3/

Vite，一个基于浏览器原生 ES imports 的开发服务器。利用浏览器去解析 imports，在服务器端按需编译返回，完全跳过了打包这个概念，服务器随起随用。同时不仅有 Vue 文件支持，还搞定了热更新，而且热更新的速度不会随着模块增多而变慢。针对生产环境则可以把同一份代码用 rollup 打包。虽然现在还比较粗糙，但这个方向我觉得是有潜力的，做得好可以彻底解决改一行代码等半天热更新的问题。

- 原生ES import
- 抛弃打包概念
- 支持Vue文件
- 支持热更新

https://github.com/vuejs/vite

```
npx create-vite-app my-vite
cd my-vites
npm install
npm run dev
```

Deno

相关文章

https://juejin.im/post/5e8da37de51d4546ec1cb432

相关代码：

https://github.com/su37josephxia/deno-fans

宣传资料

==========

全栈架构师的必备技能 - 完成一个CLI工具

总是在用vue-cli create-react-app umi等cli工具。今天教你自己手写一个cli工具。

- 从零构建一个Cli工具

Deno为什么要尝鲜

为什么要讲这个

Vue3.0为什么?

为什么要讲

- 尝鲜Deno(下一代NodeJS)

  作为Node之父Ryan的最新作品。Deno解决了困扰Node世界多年的难缠问题比如统一的API，难缠的npm包管理，以及安全机制的缺失，TS的原生支持。作为下一代JS/TS运行时标准的有力竞争者我们有必要第一时间了解。

  - Node之父打造的全新TS运行时环境
  - 统一异步API
  - 完美支持TypeScript
  - 无需Npm包管理
  - 安全沙箱机制

- 揭秘vite - 跟随Vue3.0一起发布的高效开发服务器

vite 是一个基于 Vue3 单文件组件的非打包开发服务器。随Vue3.0一起推出。特点是快捷高效，无需webpack支持非常有特点。为未来的前端工程化提供了另外一种可能。

- 特点
  - 简易Http服务器
  - 无需webpack
    - Vue文件直接渲染
    - 热更新

Vite原理浅析