

【架构师（第四篇）】脚手架开发之Lerna源码分析

一尾流莺 2022-02-22 09:24 👁 3133

关注

脚手架开发之 Lerna 源码分析

为什么要做源码分析

- 自我成长，提升编码能力和技术深度的需要
- 为我所用，应用到实际开发，实际产生效益
- 学习借鉴，站在巨人肩膀上，登高望远

为什么要分析 Lerna 源码

- 2w + star 的明星项目
- Lerna 是脚手架，对我们开发脚手架有借鉴价值
- Lerna 项目中蕴含大量的最佳实践，值得深入研究和学习

学习目标

- Lerna 源码结构和执行流程分析
- import-local 源码深度精读

学习收获

- 如何将源码分析的收获写进简历

APP内打开

- `node.js` 加载 `node_modules` 模块的流程
- 各种文件操作算法和最佳实践

知识点： 本地库作为依赖的方法 `file:` 路径

`lerna` 上线时会自动替换成线上的地址

[js](#) 复制代码

```
1  "dependencies": {  
2    "@lerna/global-options": "file:../global-options",  
3  }
```

yargs 使用

安装

[js](#) 复制代码

```
1 npm i yargs -S
```

最简单的 `yargs` 脚手架

[js](#) 复制代码

```
1 // \bin\index.js  
2  
3 // 引入 yargs 构造函数  
4 const yargs = require('yargs/yargs')  
5  
6 const { hideBin } = require('yargs/helpers')  
7  
8 // 解析参数  
9 const arg = hideBin(process.argv)
```

[APP内打开](#)

13 `.strict()` // 关闭 当快将 脚本执行时 会出现 `UNKNOWN argument: xxx` 的提示

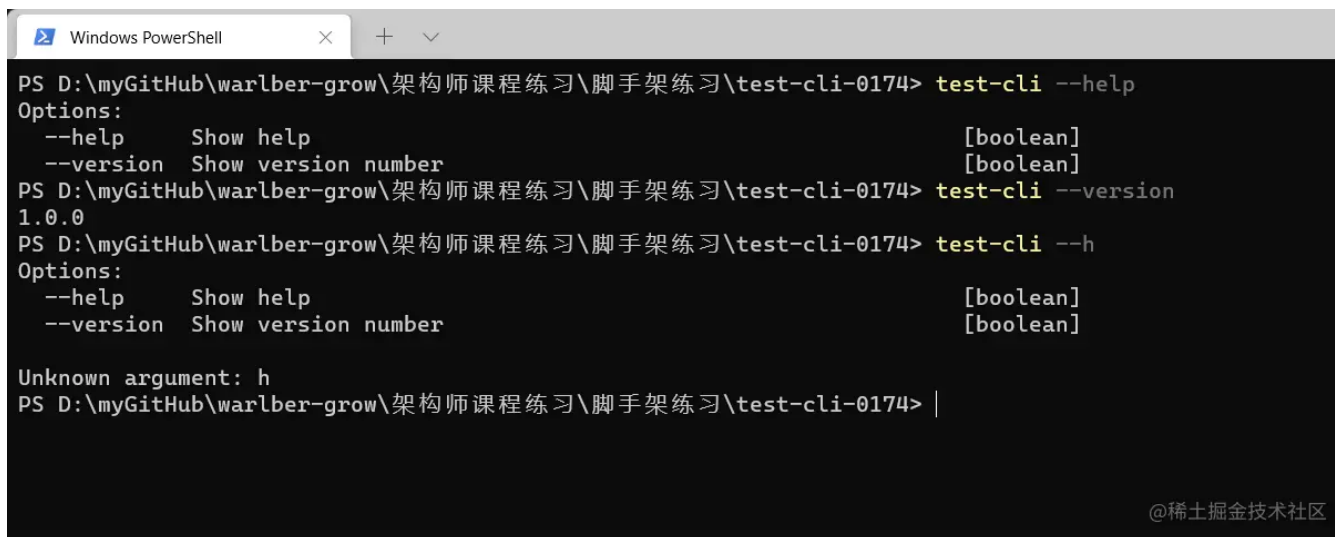
14 `.argv` // 可以解析参数

现在就可以在命令行运行了。

[js](#) 复制代码

```
1 test-cli --help
2 test-cli --version
3 test-cli --h
```

输出如下



```
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --help
Options:
  --help      Show help                                [boolean]
  --version   Show version number                       [boolean]
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --version
1.0.0
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --h
Options:
  --help      Show help                                [boolean]
  --version   Show version number                       [boolean]

Unknown argument: h
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> |
```

@稀土掘金技术社区

usage

打印在命令行最前面

[js](#) 复制代码

```
1 yargs(arg)
2 .usage("Usage:test-cli [command] <options>") // 打印在命令行最前面
3 .strict()
4 .argv
```

APP内打开



```
Usage:test-cli [command] <options>
```

```
Options:
```

```
--help      Show help
```

```
[boolean]
```

```
--version   Show version number
```

```
[boolean]
```

```
Unknown argument: h
```

```
PS D:\xinhudong\interact> |
```

@稀土掘金技术社区

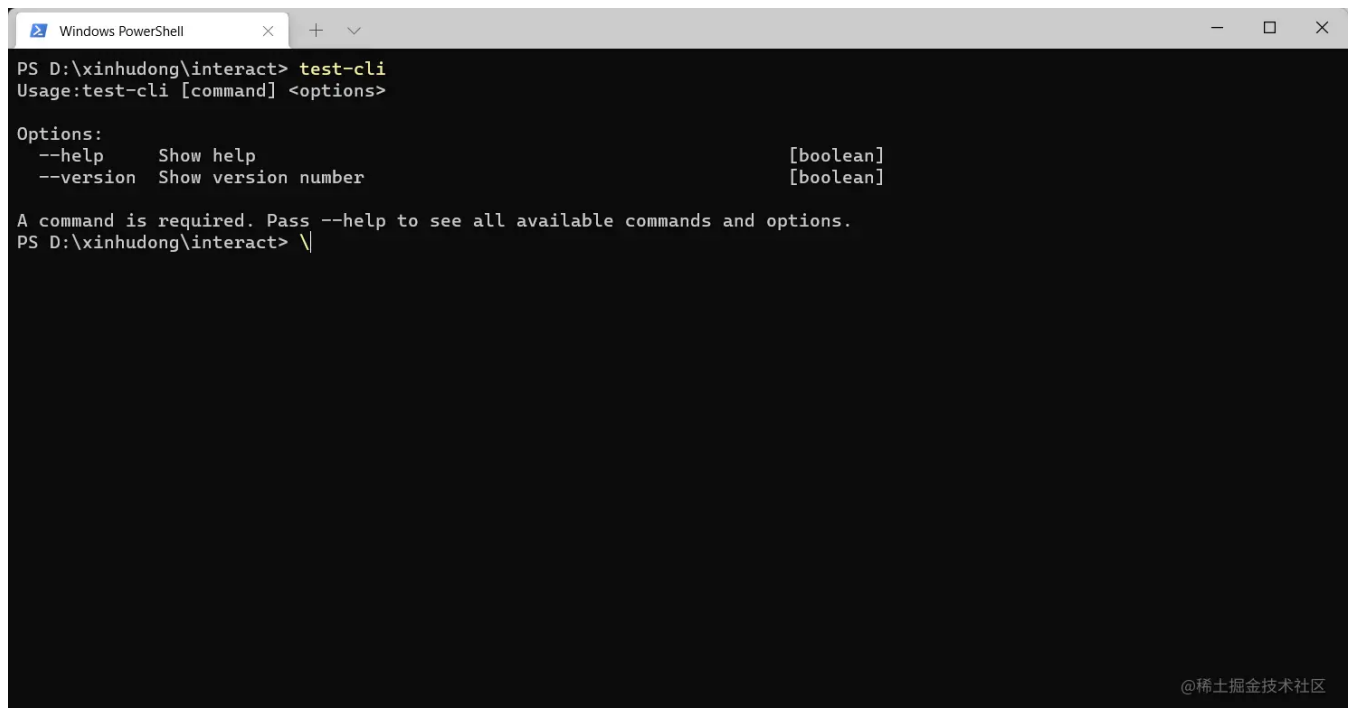
| demandCommand

设置最少需要输入的 `command` 的数量

js 复制代码

```
1 yargs(arg)
2 .demandCommand(1, "A command is required. Pass --help to see all available commands and option
3 .argv
```

当你不输入 `command` 的时候，就会报错



```
Windows PowerShell
PS D:\xinhudong\interact> test-cli
Usage:test-cli [command] <options>

Options:
  --help      Show help                      [boolean]
  --version   Show version number            [boolean]

A command is required. Pass --help to see all available commands and options.
PS D:\xinhudong\interact> \
```

@稀土掘金技术社区

| alias

APP内打开

掘金

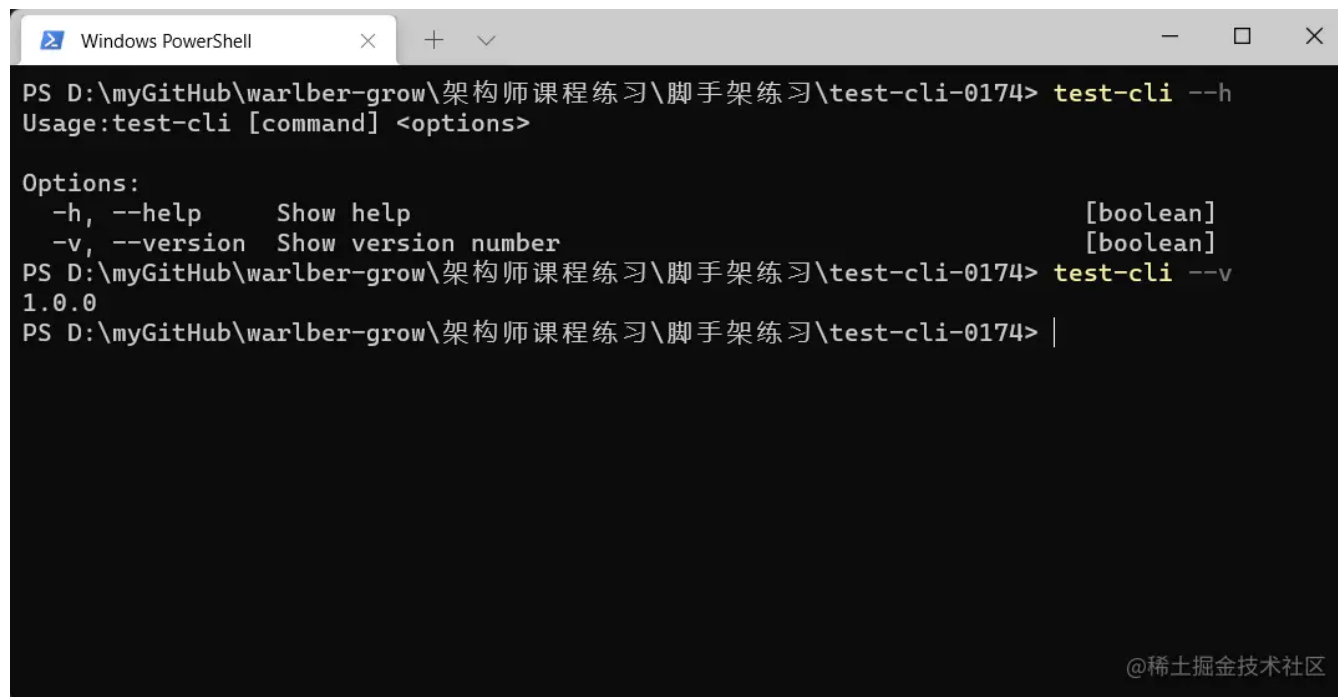
👍 15

💬 9

★ 收藏

```
2 .alias("h", "help")
3 .alias("v", "version")
4 .argv
```

这样输入 `h` 和输入 `help` 的结果是一样的, `v` 和 `version` 的结果是一样的



```
Windows PowerShell
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --h
Usage:test-cli [command] <options>

Options:
  -h, --help      Show help                                [boolean]
  -v, --version    Show version number                      [boolean]
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --v
1.0.0
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> |
```

@稀土掘金技术社区

| wrap

`cli` 的宽度

```
▼
js 复制代码

1 yargs(arg)
2 .wrap(100)
3 .argv
```

可以看到 `cli` 在命令行中的宽度发生了变化

APP内打开



```
Options:
-h, --help      Show help                                [boolean]
-v, --version   Show version number                      [boolean]
PS D:\xinhudong\interact> test-cli --h
Usage: test-cli [command] <options>

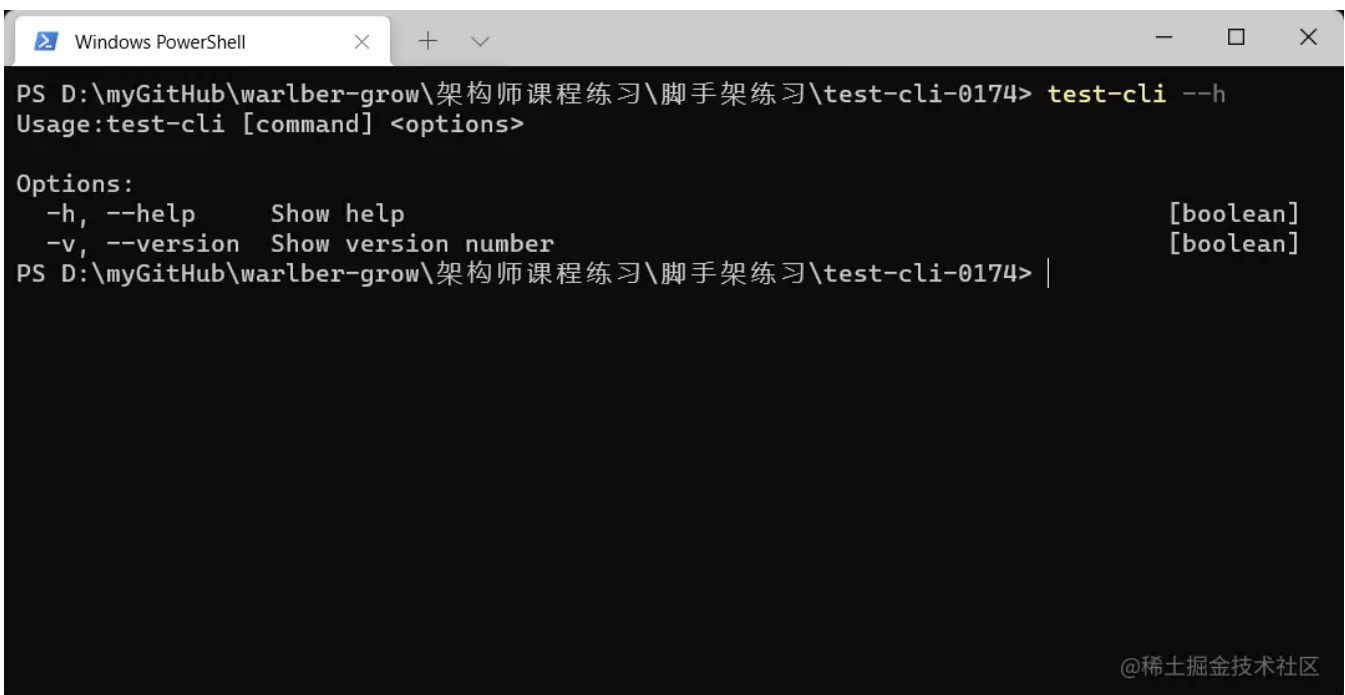
Options:
-h, --help      Show help                                [boolean]
-v, --version   Show version number                      [boolean]
PS D:\xinhudong\interact> |
```

@稀土掘金技术社区

`yargs.terminalWidth()` 这个方法会返回命令行界面的宽度，这样cli就会全屏展示了

js 复制代码

```
1 const cli = yargs(arg)
2 cli
3   .wrap(cli.terminalWidth())
4   .argv
```



```
Windows PowerShell
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --h
Usage: test-cli [command] <options>

Options:
-h, --help      Show help                                [boolean]
-v, --version   Show version number                      [boolean]
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> |
```

@稀土掘金技术社区

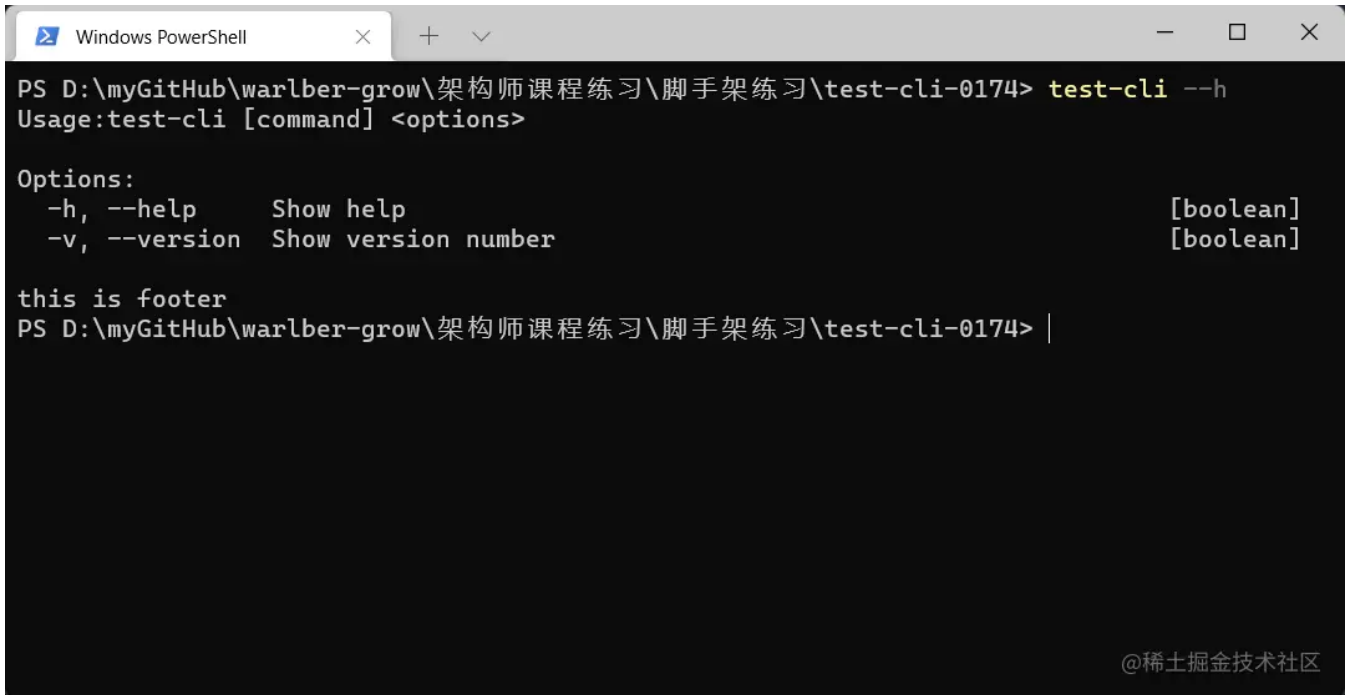
epilogue

APP内打开

 15 9 收藏

```
1 cli
2   .epilogue("this is footer")
3   .argv
```

可以看到 `cli` 的最后输出了 `this is footer`



```
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --h
Usage: test-cli [command] <options>

Options:
  -h, --help      Show help                                [boolean]
  -v, --version    Show version number                     [boolean]

this is footer
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> |
```

@稀土掘金技术社区

可以使用 `dedent` 这个库去去除缩进，使代码格式保持一致

▼ js 复制代码

```
1 cli
2   .epilogue(dedent(`
3     When a command fails, all logs are written to lerna-debug.log in the current working direc
4
5     For more information, find our manual at https://github.com/lerna/lerna
6   `))
7   .argv
```

options

增加一个全局的选项，对所有的 `command` 都有效

APP内打开

js 复制代码

```
5     describe: 'bootstrap debug moe',
6     alias: "d"
7   }
8 })
9 .argv
```

```
Windows PowerShell
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli --h
Usage: test-cli [command] <options>

Options:
  -d, --debug    bootstrap debug moe中文如何 [boolean]
  -h, --help     Show help [boolean]
  -v, --version  Show version number [boolean]

When a command fails, all logs are written to lerna-debug.log in the current working
directory.

For more information, find our manual at https://github.com/lerna/lerna

end...
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> |
```

@稀土掘金技术社区

| option

`options` 可以定义多个选项，而 `option` 只可以定义一个，作用是一样的。

可以添加 `hidden: true`，来隐藏 `option`，供内部人员开发时使用。

js 复制代码

```
1 .option("registry", {
2   type: 'string',
3   describe: "define global registry",
4   alias: "r",
5   // hidden: true
6 })
```

APP内打开


```
ry
Usage:test-cli [command] <options>

Options:
  -d, --debug      bootstrap debug moe中文如何      [boolean]
  -r, --registry   define global registry           [string]
  -h, --help       Show help                       [boolean]
  -v, --version    Show version number              [boolean]

When a command fails, all logs are written to lerna-debug.log in the current working
directory.

For more information, find our manual at https://github.com/lerna/lerna

end...

A command is required. Pass --help to see all available commands and options.
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> | @稀土掘金技术社区
```

| group

给 **option** 分组, **options** 是默认的组

▼

js 复制代码

```
1 cli
2   .group(['debug'], 'Deb Options:')
3   .group(['registry'], 'Publish Options:')
4   .argv
```

APP内打开

```

Deb Options:
-d, --debug bootstrap debug moe中文如何 [boolean]

Publish Options:
-r, --registry define global registry [string]

Options:
-h, --help Show help [boolean]
-v, --version Show version number [boolean]

When a command fails, all logs are written to lerna-debug.log in the current working
directory.

For more information, find our manual at https://github.com/lerna/lerna

end...
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> |

```

@稀土掘金技术社区

command

定义一个 `command`，接收四个参数

- 第一个: `command` 的格式, `name [port]`, `name` 是命令的名称, `port` 表示一个自定义的 `option`
- 第二个: 对 `command` 的描述
- 第三个: `builder` 函数, 在执行命令之前做的一些事情
- 第四个: `handler` 函数, 执行 `command` 的行为

注意: 定义脚手架的时候, 任何地方的别名都不可以出现重复, 不然会覆盖。

js 复制代码

```

1 cli
2 .command(
3   "init [name]",
4   "do init a project",
5   (yargs) => {
6     yargs.option("name", {
7       ...

```

APP内打开

 15

 9

 收藏

```
11  },
12  (argv) => {
13    console.log('🚀~ argv:', argv);
14  }
15  )
16  .argv
```

所有内容和别名都会出现在 `argv` 这个参数中。



```
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli init -d
-r npm -n sam-test
🚀~ argv: {
  _: [ 'init' ],
  d: true,
  debug: true,
  r: 'npm',
  registry: 'npm',
  n: 'sam-test',
  name: 'sam-test',
  '$0': 'C:\\Users\\17418\\AppData\\Roaming\\npm\\node_modules\\test-cli-0174\\bin\\in
dex.js'
}
```

@稀土掘金技术社区

另外，`command` 也支持对象的写法



js 复制代码

```
1 cli
2 .command({
3   command: "list",
4   aliases: ["ls", "la", "ll"],
5   describe: "List local packages",
6   builder: (yargs) => { },
7   handler: (yargs) => { }
8 })
9 .argv
```

APP内打开


```
Windows PowerShell
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> test-cli xxx
Did you mean ls?
Unknown argument: xxx
PS D:\myGitHub\warlber-grow\架构师课程练习\脚手架练习\test-cli-0174> |
```

@稀土掘金技术社区

| parse

会把定义的内容注入到当前的项目中

[js](#) 复制代码

```
1 // 定义一个内容
2 const context = {
3   testVersion: pkg.version,
4 };
5 // 不用在这里解析参数了
6 const cli = yargs()
7 cli
8   .parse(argv, context)
```

我们再次打印出 `args`，发现之前定义的 `testVersion` 已经出现在 `args` 中了

[APP内打开](#) 15 9 收藏



```
  -: [ 'ls' ],
  testVersion: '1.0.0',
  '$0': 'C:\\Users\\17418\\AppData\\Roaming\\npm\\node_modules\\test-cli-0174\\bin\\index.js'
}
PS D:\\myGitHub\\warlber-grow\\架构师课程练习\\脚手架练习\\test-cli-0174> |
```

@稀土掘金技术社区

Lerna 源码结构

js 复制代码

```
1 D:\\lerna-main
2 |— CHANGELOG.md
3 |— CODE_OF_CONDUCT.md
4 |— commands
5 |— CONTRIBUTING.md
6 |— core
7   |— child-process
8   |— cli
9   |— command
10  |— conventional-commits
11  |— filter-options
12  |— global-options
13  |— lerna
14  |— otplease
15  |— package
16  |— package-graph
17  |— project
18  |— prompt
19  |— validation-error
20 |— doc
```

APP内打开

👍 15

💬 9

★ 收藏

```
25 |   jest.integration.js
26 |   └─ lerna.json
27 |   └─ LICENSE
28 |   └─ node_modules
29 |   └─ package-lock.json
30 |   └─ package.json
31 |   └─ README.md
32 |   └─ scripts
33 |   └─ setup-integration-timeout.js
34 |   └─ setup-unit-test-timeout.js
35 |   └─ utils
36 |   └─ __fixtures__
```

入口文件

可以在根目录的 `package.json` 文件中发现脚手架的入口

▼

js 复制代码

```
1  "bin": {
2    "lerna": "core/lerna/cli.js"
3  },
```

Lerna 初始化分析

根据入口文件，发现 `Lerna` 初始化的时候执行了 `main` 方法。

▼

js 复制代码

```
1  // core\lerna\cli.js
2
3  // 引入 import-local 这个库
4  const importLocal = require("import-local");
5
6  // import-local 的逻辑后面单独分析
7  if (importLocal(__filename)) {
8    require("npmlog").info("cli", "using local version of lerna");
9  } else {
10 // 引入当前目录下的index.js模块 这个模块返回了一个 main 方法 并把 process.argv.slice(2) 作为参数执行
11   require(".")(process.argv.slice(2)); APP内打开 n(process.argv.slice(2))
12 }
```

```
1 // core\lerna\index.js
2 module.exports = main;
3 function main(argv) {
4 }
```

index.js 完整代码

```
1 // core\lerna\index.js
2 "use strict";
3
4 //
5 const cli = require("@lerna/cli");
6
7 // 引入若干指令
8 const addCmd = require("@lerna/add/command");
9 const bootstrapCmd = require("@lerna/bootstrap/command");
10 const changedCmd = require("@lerna/changed/command");
11 const cleanCmd = require("@lerna/clean/command");
12 const createCmd = require("@lerna/create/command");
13 const diffCmd = require("@lerna/diff/command");
14 const execCmd = require("@lerna/exec/command");
15 const importCmd = require("@lerna/import/command");
16 const infoCmd = require("@lerna/info/command");
17 const initCmd = require("@lerna/init/command");
18 const linkCmd = require("@lerna/link/command");
19 const listCmd = require("@lerna/list/command");
20 const publishCmd = require("@lerna/publish/command");
21 const runCmd = require("@lerna/run/command");
22 const versionCmd = require("@lerna/version/command");
23
24 // 引入 package.json 模块
25 const pkg = require("./package.json");
26
27 // 输出 main 方法
28 module.exports = main;
29
30 // main 方法
31 function main(argv) {
32   // 定义一个对象，里面保存一个 lernaVersion package.json 中的 version 属性的值
33   const context = {
34     lernaVersion: pkg.version,
```

APP内打开


```

15     .addCommand(cli) // 脚本在运行时的默认规则
16   })
17   .alias("h", "help") // 别名
18   .alias("v", "version") // 别名
19   .wrap(cli.terminalWidth()) // 配置cli的宽度和命令行一样
20   .epilogue(dedent`
21 `); // 配置 cli 结尾的内容
22 }

```

接下来看一下 `globalOptions` 这个东西都干了什么


[js 复制代码](#)

```

1 // core\global-options\index.js
2 function globalOptions(yargs) {
3   // 定义了一堆的 option
4   const opts = {
5     loglevel: {
6       defaultDescription: "info",
7       describe: "What level of logs to report.",
8       type: "string",
9     },
10    concurrency: {
11      defaultDescription: os.cpus().length,
12      describe: "How many processes to use when lerna parallelizes tasks.",
13      type: "number",
14      requiresArg: true,
15    },
16    "reject-cycles": {
17      describe: "Fail if a cycle is detected among dependencies.",
18      type: "boolean",
19    },
20    "no-progress": {
21      describe: "Disable progress bars. (Always off in CI)",
22      type: "boolean",
23    },
24    progress: {
25      // proxy for --no-progress
26      hidden: true,
27      type: "boolean",
28    },
29    "no-sort": {
30      describe: "Do not sort packages (dependencies before dependents).",
31      type: "boolean",

```

[APP内打开](#)

```

36     type: "boolean",
37   },
38   "max-buffer": {
39     describe: "Set max-buffer (in bytes) for subcommand execution",
40     type: "number",
41     requiresArg: true,
42   },
43 };
44
45 // 拿到这些 option 的名称
46 const globalKeys = Object.keys(opts).concat(["help", "version"]);
47
48
49 return yargs
50   .options(opts) // 给 yargs 添加 全局options
51   .group(globalKeys, "Global Options:") // 对 options 进行分组
52   .option("ci", { // 添加了一个隐藏的 option
53     hidden: true,
54     type: "boolean",
55   });
56 }

```

Command 执行过程

前面提到 `main` 方法当中添加了很多 `command`，再来看看 `Command` 执行过程是什么样的。

以 `listCmd` 为例


[js 复制代码](#)

```

1 // commands\list\command.js
2
3 const { filterOptions } = require("@lerna/filter-options");
4 const listable = require("@lerna/listable");
5
6 exports.command = "list"; // 配置命令的名称
7
8 exports.aliases = ["ls", "la", "ll"]; // 配置命令的别名
9
10 exports.describe = "List local packages"; // 配置命令的描述
11
12 exports.builder = (yargs) => { // 配置命令的 builder 做的事情
13   listable.options(yargs);

```

[APP内打开](#)

```
17
18 exports.handler = function handler(argv) { // 配置命令在执行过程做的事情
19   return require(".")(argv); // 调用当前目录下 index.js 导出的 factory 方法
20 };
21
```

继续看看 `handler` 所执行的 `factory` 方法。

[js 复制代码](#)

```
1 // commands\list\index.js
2
3 module.exports = factory;
4
5 function factory(argv) {
6   return new ListCommand(argv); // 实例化一个 ListCommand
7 }
8
9 // ListCommand 的结构
10 class ListCommand extends Command {
11   get requiresGit() {
12   }
13
14   initialize() {
15   }
16
17   execute() {
18   }
19 }
20
21 module.exports.ListCommand = ListCommand;
```

可以看到 `ListCommand` 是通过继承来了，继续看看父类的内容

[js 复制代码](#)

```
1 // core\command\index.js
2
3 class Command {
4   constructor(_argv) {
5     // 深拷贝 argv
6     const argv = cloneDeep(_argv);
7     // 添加 name 属性 FooCommand => foo
```

[APP内打开](#)

```
12  if (!this.composed) {
13    // composed commands have already logged the lerna version
14    log.notice("cli", `v${argv.lernaVersion}`);
15  }
16  // 最终的执行过程
17  let runner = new Promise((resolve, reject) => {
18    // 定义一个微任务 chain.then 会被加入到微任务队列
19    let chain = Promise.resolve();
20    // 会行程队列, 一个接一个执行
21    chain = chain.then(() => {
22      this.project = new Project(argv.cwd);
23    });
24    chain = chain.then(() => this.configureEnvironment());
25    chain = chain.then(() => this.configureOptions());
26    chain = chain.then(() => this.configureProperties());
27    chain = chain.then(() => this.configureLogging());
28    chain = chain.then(() => this.runValidations());
29    chain = chain.then(() => this.runPreparations());
30    // 核心内容
31    chain = chain.then(() => this.runCommand());
32
33    chain.then(
34      (result) => {
35      },
36      (err) => {
37      }
38    );
39  });
40  // 向 argv 中定义 cwd 和 $0 两个参数
41  for (const key of ["cwd", "$0"]) {
42    Object.defineProperty(argv, key, { enumerable: false });
43  }
44  // 对 argv 属性做一些处理
45  Object.defineProperty(this, "argv", {
46    value: Object.freeze(argv),
47  });
48  // 对 runner 属性做一些处理
49  Object.defineProperty(this, "runner", {
50    value: runner,
51  });
52  }
53  // 核心内容
54  runCommand() {
55    return Promise.resolve()
56      .then(() => this.initialize()) // initialize 方法
57      .then((proceed) => {
```

APP内打开

```
61 // early exits set their own exitcode (if non-zero)
62 });
63 }
64 // initialize 和 execute 强制用户实现，否则会报错
65 initialize() {
66   throw new ValidationError(this.name, "initialize() needs to be implemented.");
67 }
68
69 execute() {
70   throw new ValidationError(this.name, "execute() needs to be implemented.");
71 }
72 }
```

`initialize` 和 `execute` 强制用户实现，否则会报错。

那么现在返回来再看看这两个方法的实现，不用关心 `lerna` 的源码，主要是看一下执行过程。

[js 复制代码](#)

```
1 // commands\list\index.js
2 module.exports = factory;
3
4 function factory(argv) {
5   return new ListCommand(argv);
6 }
7
8 class ListCommand extends Command {
9   get requiresGit() {
10     return false;
11   }
12
13   initialize() {
14     // 也是通过 chain 微任务队列的方式
15     let chain = Promise.resolve();
16
17     chain = chain.then(() => getFilteredPackages(this.packageGraph, this.execOpts, this.options))
18     chain = chain.then((filteredPackages) => {
19       this.result = listable.format(filteredPackages, this.options);
20     });
21
22     return chain;
23   }
24
25   execute() {
```

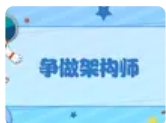
[APP内打开](#)



```
31 // 打印log 执行完毕
32 this.logger.success(
33     "found",
34     "%d %s",
35     this.result.count,
36     this.result.count === 1 ? "package" : "packages"
37 );
38 }
39 }
40
41 module.exports.ListCommand = ListCommand;
42
```

标签: 前端 架构

文章被收录于专栏:



我要当架构师

为了提升自己的能力，我在某课网买了一个架构师的课程。本专栏...

已订阅

相关小册



VIP Vue3 企业级项目实战

程序员十三

2936购买

¥69



VIP 基于 Vite 的组件库工程化实战

全栈然叔

3006购买

¥29.9

评论

APP内打开

输入评论 (Enter换行, Ctrl + Enter发送)

15

9

收藏



feng_cc LV.3 JY.5

8月前

大佬，你是用哪个IDE的，vscode很多require的包跳转不了，怎样配置？

👍 点赞 3



一尾流莺 👤

8月前

我也用的vscode,我这是能跳转的,也没做啥配置😂

👍 点赞 3 回复



feng_cc

8月前

难道是安装依赖有问题？

“我也用的vscode,我这是能跳转的,也没做啥配置😂”

👍 点赞 3 回复

查看更多回复 ▾



ZoeLee LV.3 JY.4

1年前

你这个不就是慕课上的课吗

👍 点赞 1



一尾流莺 👤

1年前

是啊,我专栏介绍明确指出是在某课买的课程呀

👍 点赞 3 回复



csdn来挖墙脚 LV.4 JY.5

1年前

我要当一名架构师, 🤪🤪🤪

👍 点赞 2



一尾流莺 👤

1年前

可以你很有钱途

APP内打开

👍 15

💬 9

★ 收藏

“可以你很有钱途”

👍 点赞 💬 回复

相关推荐

一尾流莺 1年前

【架构师（第五篇）】脚手架之import-local执行流程及简历设计

👁 1354 👍 12 💬 评论

一尾流莺 1年前

【架构师（第二篇）】脚手架架构设计和框架搭建

👁 1035 👍 15 💬 8

一尾流莺 7月前

【架构师（第四十九篇）】服务端开发之认识 Docker-compose

👁 1151 👍 8 💬 评论

汪汪队汪汪汪 28天前

搭建属于自己的动态博客

👁 27 👍 点赞 💬 评论

成长ing12138 2年前

配置腾讯云轻量服务器教程

👁 989 👍 1 💬 评论

一尾流莺 1年前

【架构师（第三篇）】脚手架开发之掌握Lerna操作流程

👁 710 👍 17 💬 7

一尾流莺 1年前

【架构师（第三十八篇）】服务端开发之本地安装最新版 MySQL 数据库

👁 1831 👍 7 💬 2

一尾流莺 1年前

【架构师（第十四篇）】脚手架之 egg.js 和 mongodb 的使用

APP内打开

👍 15

💬 9

☆ 收藏

👁 2.1w 👍 11 💬 2

一尾流莺 1年前

【架构师（第四十五篇）】 服务端开发之认识 Github actions

👁 1133 👍 6 💬 评论

一尾流莺 1年前

【架构师（第二十一篇）】 编辑器开发之需求分析和架构设计

👁 904 👍 5 💬 4

Hogwarts霍格沃兹测试... 1年前

技术分享 | web自动化测试-执行 JavaScript 脚本

👁 398 👍 2 💬 评论

一尾流莺 1年前

【架构师（第二十三篇）】 编辑器开发之画布区域组件的渲染

👁 900 👍 9 💬 评论

APP内打开