link: null
title: 珠峰架构师成长计划
description: 首先使用 npm 将 Lerna 安装到全局环境中
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=133 sentences=239, words=1702

1.lerna

- Lerna 官网 (www.lernajs.cn) 是一个管理工具，用于管理包含多个软件包（package）的 JavaScript 项目
- Lerna Git 仓库 (github.com/lerna/lerna) 是一种工具，针对 使用 git 和 npm 管理多软件包代码仓库的工作流程进行优化

首先使用 npm 将 Lerna 安装到全局环境中

```
npm install -g lerna
```

接下来，我们将创建一个新的 git 代码仓库

```
mkdir lerna4 && cd lerna4
```

现在，我们将上述仓库转变为一个 Lerna 仓库：

```
lerna init
lerna notice cli v4.0.0
lerna info Initializing Git repository
lerna info Creating package.json
lerna info Creating lerna.json
lerna info Creating packages directory
lerna success Initialized Lerna files
```

```
lerna4 / packages / 放置多个软件包(package);
package.json;
lerna.json;
```

.gitignore

```
node_modules.idea.vscode;
```

2.lerna 源码

```
npm install -g yrm
npm install -g nrm
```

```
git clone https:
```

.vscode\launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "skipFiles": ["/**"],
      "program": "${workspaceFolder}\\core\\lerna\\cli.js",
      "args": ["ls"]
    }
  ]
}
```

```
lerna 入口核心包
@lerna/cli
@lerna/create 创建包命令
@lerna/init 初始化lerna项目
```

3.创建 npm 私服

```
cnpm install verdaccio -g
```

```
verdaccio
http:
npm adduser --registry http:
npm publish --registry http:
```

4.创建包

```
lerna create lerna4 --registry http:
lerna success create New package lerna4 created at ./packages\lerna4
lerna create @lerna4/cli --registry http:
lerna success create New package @lerna4/cli created at ./packages\cli
lerna create @lerna4/create --registry http:
lerna success create New package @lerna4/create created at ./packages\create
lerna create @lerna4/init --registry http:
lerna success create New package @lerna4/init created at ./packages\init
```

4.单元测试

```
npm install --save-dev jest
```

```

lerna run test

lerna run test --scope lerna4

lerna exec -- jest

lerna exec --scope lerna4 -- jest
```

```
{
  "name": "root",
  "private": true,
  "devDependencies": {
    "lerna": "^4.0.0"
  },
+ "scripts": {
+    "test":"jest"
+   }
}
```

```
module.exports = {
  testMatch: ["**/__tests__/**/*.test.js"],
};
```

packages\lerna4\package.json

```
{
+  "scripts": {
+    "test": "jest"
+   }
}
```

packages\lerna4\lib\lerna4.js

```
module.exports = lerna4;
function lerna4() {
  return "lerna4";
}
```

packages\create__tests__\create.test.js

```
;

const create = require("..");
describe("@lerna4/create", () => {
  it("create", () => {
    expect(create()).toEqual("create");
  });
});
```

6.eslint

- eslint (https://www.npmjs.com/package/eslint)是一个插件化并且可配置的 JavaScript 语法规则和代码风格的检查工具
- 代码质量问题：使用方式有可能有问题
- 代码风格问题：风格不符合一定规则
- vscode-eslint (https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint)

```
cnpm i eslint  --save-dev
```

```
module.exports = {
  parserOptions: { ecmaVersion: 2017, sourceType: "module" },
  extends: ["eslint:recommended"],
  rules: {
    "no-unused-vars": ["off"],
  },
  env: { node: true, jest: false },
};
```

```
__tests__;
```

```
  "scripts": {
    "test": "jest",
+   "lint":"eslint --ext .js packages/**/*.js --no-error-on-unmatched-pattern --fix"
  }
```

7.Prettier

```
cnpm i   prettier eslint-plugin-prettier  --save-dev
```

```
module.exports = {
  extends: ['eslint:recommended'],
  //让所有可能会与 prettier 规则存在冲突的 eslint rule失效，并使用 prettier 的规则进行代码检查
  //相当于用 prettier 的规则，覆盖掉 eslint:recommended 的部分规则
+ plugins: ['prettier'],
  rules: {
    'no-unused-vars': ['off'],
    //不符合prettier规则的代码要进行错误提示
+   'prettier/prettier': ['error', { endOfLine: 'auto' }],
  },
  env: { node: true, jest: false },
};
```

```
module.exports = {
  singleQuote: true,
};
```

8. editorconfig

- editorconfig (https://editorconfig.org/)帮助开发人员在不同的编辑器和 IDE 之间定义和维护一致的编码样式
- 不同的开发人员，不同的编辑器，有不同的编码风格，而 EditorConfig 就是用来协同团队开发人员之间的代码的风格及样式规范化的一个工具，而.editorconfig 正是它的默认配置文件
- EditorConfig (https://marketplace.visualstudio.com/items?itemName=EditorConfig.EditorConfig)
- vscode 这类编辑器，需要自行安装 editorconfig 插件
- Unix 系统里，每行结尾只有换行,即 `\n` LF(Line Feed)
- Windows 系统里面，每行结尾是 `&#x6362;&#x884C;` `&#x56DE;&#x8F66;`，即 `\r\n` CR/LF
- Mac 系统里，每行结尾是 `&#x56DE;&#x8F66;`，即 `\r` CR(Carriage Return)

```
root = true

[*]
indent_style = space
indent_size = 2
end_of_line = lf
charset = utf-8
trim_trailing_whitespace = true

[*.md]
trim_trailing_whitespace = false
```

- 开发过程中，如果写出代码质量有问题的代码，eslint 能够及时提醒开发者，便于及时修复
- 如果写出代码格式有问题的代码，prettier 能够自动按照我们制定的规范、格式化代码
- 不同开发者如果使用不同的编辑器(webstorm/vscode)或系统(windows/mac),能够执行统一的代码风格标准

9. git hook

- 可以在 `git commit` 之前检查代码，保证所有提交到版本库中的代码都是符合规范的
- 可以在 `git push` 之前执行单元测试,保证所有的提交的代码经过的单元测试
- husky (https://www.npmjs.com/package/husky)可以让我们向项目中方便添加 `git hooks`
- lint-staged (https://www.npmjs.com/package/lint-staged) 用于实现每次提交只检查本次提交所修改的文件

```
cnpm i husky --save-dev
npm set-script prepare "husky install"
```

```
npx husky add .husky/pre-commit "npx lint-staged"
```

```
cnpm install commitizen cz-customizable @commitlint/cli @commitlint/config-conventional --save-dev
```

```
npx husky add .husky/commit-msg "npx --no-install commitlint --edit $1"
```

```
npx husky add .husky/commit-msg "npx --no-install commitlint --edit $1"
```

```
module.exports = {
  types: [
    { value: "feat", name: "feat:一个新特性" },
    { value: "fix", name: "fix:修复BUG" },
  ],
  scopes: [{ name: "sale" }, { name: "user" }, { name: "admin" }],
};
```

```
module.exports = {
  extends: ["@commitlint/config-conventional"],
};
```

```
  "scripts": {
    "test": "jest",
    "lint": "eslint --ext .js packages/**/*.js --no-error-on-unmatched-pattern --fix",
    "prepare": "husky install",
+   "commit": "cz"
  },
```

10.发布上线

```
npx husky add .husky/pre-push "npm run test"
```

```
lerna version
lerna publish
```

11.安装命令

packages\lerna4\cli.js

```

require(".")(process.argv.slice(2));
```

packages\lerna4\index.js

```
module.exports = main;
function main(argv) {
  console.log(argv);
}
```

packages\lerna4\package.json

```
{
+  "main": "index.js",
+  "bin":{
+     "lerna4":"cli.js"
+  }
}
```

```
cd packages\lerna4
npm link
lerna4
```

12. yargs

```
const yargs = require("yargs/yargs");
const argv = process.argv.slice(2);
const cli = yargs(argv);

const opts = {
  loglevel: {
    defaultDescription: "info",
    describe: "报告日志的级别",
    type: "string",
    alias: "L",
  },
};

const globalKeys = Object.keys(opts).concat(["help", "version"]);
cli
  .options(opts)
  .group(globalKeys, "Global Options:")
  .usage("Usage: $0  [options]")
  .demandCommand(1, "至少需要一个命令，传递--help查看所有的命令和选项")
  .recommendCommands()
  .strict()
  .fail((msg, err) => {

    console.error("lerna", msg, err);
  })
  .alias("h", "help")
  .alias("v", "version")
  .wrap(cli.terminalWidth())
  .epilogue(

    `当1个命令失败了，所有的日志将会写入当前工作目录中的lerna-debug.log`
  )
  .command({
    command: "create ",
    describe: "创建一个新的lerna管理的包",
    builder: (yargs) => {
      yargs
        .positional("name", {
          describe: "包名(包含scope)",
          type: "string",
        })
        .options({
          registry: {
            group: "Command Options:",
            describe: "配置包的发布仓库",
            type: "string",
          },
        });
    },
    handler: (argv) => {
      console.log("执行init命令", argv);
    },
  })
  .parse(argv);
```

13.跑通 init 命令

```
lerna link
lerna bootstrap
```

packages\cli\package.json

```
"dependencies": {
    "@lerna4/cli":"^0.0.4",
    "@lerna4/init":"^0.0.4"
},
+  "main": "index.js",
```

packages\cli\index.js

```
const yargs = require("yargs/yargs");
function lernaCLI() {
  const cli = yargs();

  const opts = {
    loglevel: {
      defaultDescription: "info",
      describe: "报告日志的级别",
      type: "string",
      alias: "L",
    },
  };

  const globalKeys = Object.keys(opts).concat(["help", "version"]);
  return cli
    .options(opts)
    .group(globalKeys, "Global Options:")
    .usage("Usage: $0  [options]")
    .demandCommand(1, "至少需要一个命令，传递--help查看所有的命令和选项")
    .recommendCommands()
    .strict()
    .fail((msg, err) => {

      console.error("lerna", msg, err);
    })
    .alias("h", "help")
    .alias("v", "version")
    .wrap(cli.terminalWidth())
    .epilogue(

      `当1个命令失败了，所有的日志将会写入当前工作目录中的lerna-debug.log`
    );
}
module.exports = lernaCLI;
```

packages\init\command.js

```
exports.command = "init";
exports.describe = "创建一个新的Lerna仓库";
exports.builder = (yargs) => {
  console.log("执行init builder");
};
exports.handler = (argv) => {
  console.log("执行init命令", argv);
};
```

packages\lerna4\package.json

```
+  "main": "index.js"
```

packages\lerna4\index.js

```
const cli = require('@lerna4/cli');
const initCmd = require('@lerna4/init/command');
function main(argv) {
  return cli().command(initCmd).parse(argv);
}

module.exports = main;
```

14.实现 init 命令

```
lerna add fs-extra  packages/init
lerna add  execa  packages/init
```

packages\init\command.js

```
exports.command = "init";
exports.describe = "创建一个新的Lerna仓库";
exports.builder = () => {
  console.log("执行init builder");
};
exports.handler = (argv) => {
  console.log("执行init命令", argv);
  return require(".")(argv);
};
```

packages\init\index.js

```
const path = require("path");
const fs = require("fs-extra");
const execa = require("execa");

class InitCommand {
  constructor(argv) {
    this.argv = argv;
    this.rootPath = path.resolve();
  }
  async execute() {
    await execa("git", ["init"], { stdio: "pipe" });
    await this.ensurePackageJSON();
    await this.ensureLernaConfig();
    await this.ensurePackagesDir();
    console.log("Initialized Lerna files");
  }
  async ensurePackageJSON() {
    console.log("创建 package.json");
    await fs.writeJson(
      path.join(this.rootPath, "package.json"),
      {
        name: "root",
        private: true,
        devDependencies: {
          lerna: "^4.0.0",
        },
      },
      { spaces: 2 }
    );
  }
  async ensureLernaConfig() {
    console.log("创建 lerna.json");
    await fs.writeJson(
      path.join(this.rootPath, "lerna.json"),
      {
        packages: ["packages/*"],
        version: "0.0.0",
      },
      { spaces: 2 }
    );
  }
  async ensurePackagesDir() {
    console.log("创建 packages 目录");
    await fs.mkdirp(path.join(this.rootPath, "packages"));
  }
}
function factory(argv) {
  new InitCommand(argv).execute();
}
module.exports = factory;
```

15.实现 create 命令

```
lerna add pify  packages/crate
lerna add  init-package-json  packages/crate
lerna add  dedent  packages/crate
```

packages\lerna4\index.js

```
const cli = require('@lerna4/cli');
const initCmd = require('@lerna4/init/command');
const createCmd = require('@lerna4/create/command');
function main(argv) {
  return cli()
   .command(initCmd)
+  .command(createCmd)
   .parse(argv);
}

module.exports = main;
```

packages\lerna4\package.json

```
{
  "dependencies": {
    "@lerna4/cli":"^0.0.4",
    "@lerna4/init":"^0.0.4",
+   "@lerna4/create":"^0.0.4"
  },
}
```

packages\create\command.js

```
exports.command = "create ";
exports.describe = "创建一个新的lerna管理的包";
exports.builder = (yargs) => {
  console.log("执行init builder");
  yargs
    .positional("name", {
      describe: "包名(包含scope)",
      type: "string",
    })
    .options({
      registry: {
        group: "Command Options:",
        describe: "配置包的发布仓库",
        type: "string",
      },
    });
};
exports.handler = (argv) => {
  console.log("执行create命令", argv);
  return require(".")(argv);
};
```

packages\create\index.js

```
const path = require("path");
const fs = require("fs-extra");
const dedent = require("dedent");
const initPackageJson = require("pify")(require("init-package-json"));
class CreateCommand {
  constructor(options) {
    this.options = options;
    this.rootPath = path.resolve();
    console.log("options", options);
  }
  async execute() {
    const { name, registry } = this.options;
    this.targetDir = path.join(this.rootPath, "packages/cli");
    this.libDir = path.join(this.targetDir, "lib");
    this.testDir = path.join(this.targetDir, "__tests__");
    this.libFileName = `${name}.js`;
    this.testFileName = `${name}.test.js`;
    await fs.mkdirp(this.libDir);
    await fs.mkdirp(this.testDir);
    await this.writeLibFile();
    await this.writeTestFile();
    await this.writeReadme();
    var initFile = path.resolve(process.env.HOME, ".npm-init");
    await initPackageJson(this.targetDir, initFile);
  }
  async writeLibFile() {
    const libContent = dedent`
        module.exports = ${this.camelName};
        function ${this.camelName}() {
            // TODO
        }
    `;
    await catFile(this.libDir, this.libFileName, libContent);
  }
  async writeTestFile() {
    const testContent = dedent`
    const ${this.camelName} = require('..');
    describe('${this.pkgName}', () => {
        it('needs tests');
    });
  `;
    await catFile(this.testDir, this.testFileName, testContent);
  }
  async writeReadme() {
    const readmeContent = dedent`## Usage`;
    await catFile(this.targetDir, "README.md", readmeContent);
  }
}
function catFile(baseDir, fileName, content) {
  return fs.writeFile(path.join(baseDir, fileName), `${content}\n`);
}
function factory(argv) {
  new CreateCommand(argv).execute();
}

module.exports = factory;
```

16.参考

命令 说明 lerna init 初始化项目

命令 说明 lerna create 创建 package lerna add 安装依赖 lerna link 链接依赖

命令 说明 lerna exec 执行 shell 脚本 lerna run 执行 npm 命令 lerna clean 清空依赖 lerna bootstrap 重新安装依赖

命令 说明 lerna version 修改版本号 lerna changed 查看上个版本以来的所有变更 lerna diff 查看 diff lerna publish 发布项目

- 规范化的 `git commit`可以提高 `git log`可读性，生成格式良好的 `changelog`
- Conventional Commits (https://www.conventionalcommits.org/zh-hans/v1.0.0/) 是一种用于给提交信息增加人机可读含义的规范
- 它提供了一组简单规则来创建清晰的提交历史
- 通过在提交信息中描述功能、修复和破坏性变更，使这种惯例与semver (https://semver.org/) 相互对应

```
[可选 范围]:

[可选 正文]

[可选 脚注]
```

- feat: 类型 为 feat 的提交表示在代码库中新增了一个功能（这和语义化版本中的 MINOR 相对应）
- fix: 类型 为 fix 的提交表示在代码库中修复了一个 bug（这和语义化版本中的 PATCH 相对应）

- docs: 只是更改文档
- style: 不影响代码含义的变化（空白、格式化、缺少分号等）
- refactor: 代码重构，既不修复错误也不添加功能
- perf: 改进性能的代码更改
- test: 添加确实测试或更正现有的测试
- build: 影响构建系统或外部依赖关系的更改（示例范围：gulp、broccoli、NPM）
- ci: 更改持续集成文件和脚本（示例范围：Travis、Circle、BrowserStack、SauceLabs）
- chore: 其他不修改 src 或 test 文件。
- revert: commit 回退
- 可以为提交类型添加一个围在圆括号内的作用域，以为其提供额外的上下文信息