link: null
title: 珠峰架构师成长计划
description: .vscode\launch.json
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=249 sentences=937, words=6453

1.create-vite

1.1 create-vite简介

- vite官网 (https://cn.vitejs.dev/guide/#scaffolding-your-first-vite-project)
- create-vite包 (https://www.npmjs.com/package/create-vite)
- create-vite源码 (https://github.com/vitejs/vite/tree/main/packages/create-vite)

1.2 使用

```
npm init vite
Need to install the following packages:
  create-vite
Ok to proceed? (y) y
√ Project name: ... vite-project
√ Select a framework: >> react
√ Select a variant: >> react

Scaffolding project in C:\aprepare\t1\vite-project...

Done. Now run:

  cd vite-project
  npm install
  npm run dev
```

1.3 create-vite源码调试

- minimist (https://www.npmjs.com/package/minimist)解析参数选项,类似的还有yargs (https://www.npmjs.com/package/yargs)和commander (https://www.npmjs.com/package/commander)
- kolorist (https://www.npmjs.com/package/kolorist)在控制台打印颜色,类似的还有chalk (https://www.npmjs.com/package/chalk)
- prompts (https://www.npmjs.com/package/prompts)交互式命令行，类似还有inquirer (https://www.npmjs.com/package/inquirer)

```
git clone https:
cd vite
yarn install
packages\create-vite\index.js
```

.vscode\launch.json

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "pwa-node",
            "request": "launch",
            "name": "Launch Program",
            "skipFiles": [
                "/**"
            ],
            "program": "${workspaceFolder}\\packages\\create-vite\\index.js",
            "args": ["create","vite-project"]
        }
    ]
}
```

1.4 create-vite功能

- [√] 支持参数解析
- [√] 支持自定义项目名
- [√] 支持空目录检查
- [√] 支持静态项目模板
- [×] 不支持lerna lerna (https://github.com/lerna/lerna)
- [×] 不支持文件异步写入create-react-app (https://github.com/facebook/create-react-app)
- [×] 不支持多进程执行命令create-react-app (https://github.com/facebook/create-react-app)
- [×] 不支持执行动态 node命令create-react-app (https://github.com/facebook/create-react-app)
- [×] 不支持自动安装依赖create-react-app (https://github.com/facebook/create-react-app)
- [×] 不支持自动启动服务create-react-app (https://github.com/facebook/create-react-app)
- [×] 不支持参数配置yam (https://github.com/yampkg/yam)
- [×] 不支持 gitub和 gitee仓库动态读取
- [×] 不支持模板标签选择
- [×] 不支持动态模板渲染
- [×] 不支持插件化配置技术栈vue-cli (https://github.com/vuejs/vue-cli)

2.初始化项目

2.1 lerna初始化

```
mkdir vite100
cd vite100
lerna init
```

2.2 使用yarn workspace

- 开发多个互相依赖的package时，workspace会自动对package的引用设置软链接(symlink),比yarn link更加方便，且链接仅局限在当前workspace中，不会对整个系统造成影响
- 所有package的依赖会安装在根目录的 node_modules下，节省磁盘空间，且给了yam更大的依赖优化空间
- Yam workspace只会在根目录安装一个node_modules，这有利于提升依赖的安装效率和不同package间的版本复用。而Lerna默认会进到每一个package中运行yam/npm install，并在每个package中创建一个node_modules
- yam官方推荐的方案，是集成yam workspace和lerna,使用yam workspace来管理依赖，使用lerna来管理npm包的版本发布

2.2.1 lerna.json

```
{
  "packages": [
    "packages/*"
  ],
  "version": "0.0.0",
+ "npmClient": "yarn",
+ "useWorkspaces": true
}
```

2.2.2 package.json

```
{
  "name": "root",
  "private": true,
  "devDependencies": {
    "lerna": "^4.0.0"
  },
+  "workspaces": [
+    "packages/*"
+ ]
}
```

2.3 创建子包

```
lerna create @vite100/config -y
lerna create @vite100/create -y
lerna create vite100 -y
lerna create @vite100/settings -y
lerna create @vite100/utils -y
lerna create @vite100/cli-plugin-router -y
```

2.4 安装依赖

- fs-extra (https://www.npmjs.com/package/fs-extra) 加强版的读写模块
- clone-git-repo (https://www.npmjs.com/package/clone-git-repo) 克隆git仓库
- axios (https://www.npmjs.com/package/axios) 请求接口
- cross-spawn (https://www.npmjs.com/package/cross-spawn) 开启子进程
- userhome (https://www.npmjs.com/package/userhome) 获取用户主目录
- chalk (https://www.npmjs.com/package/chalk) 控制台打印彩色文字
- ejs (https://www.npmjs.com/package/ejs) 模板渲染
- execa (https://www.npmjs.com/package/execa) 通过子进程执行命令
- glob (https://www.npmjs.com/package/glob) 按模式匹配文件
- inquirer (https://www.npmjs.com/package/inquirer) 交互式命令行选择
- isbinaryfile (https://www.npmjs.com/package/isbinaryfile) 判断是否是二进制文件
- vue-codemod (https://www.npmjs.com/package/vue-codemod) 通过AST修改源代码
- jscodeshift (https://www.npmjs.com/package/jscodeshift) 通过语法树修改源代码
- vite100 (vite100) 核心命令
- @vite100/settings (@vite100/settings) 常量配置
- @vite100/utils (@vite100/utils) 帮助方法
- @vite100/config (@vite100/config) 配置参数
- @vite100/create (@vite100/create) 创建项目

2.4.1 config\package.json

packages\config\package.json

```
{
  "dependencies": {
    "@vite100/settings": "^0.0.0",
    "@vite100/utils": "^0.0.0",
    "fs-extra": "^10.0.0",
    "userhome": "^1.0.0"
   }
}
```

2.4.2 create\package.json

packages\create\package.json

```
{
  "dependencies": {
    "@vite100/settings": "^0.0.0",
    "@vite100/utils": "^0.0.0",
    "chalk": "^4.1.2",
    "clone-git-repo": "^0.0.2",
    "ejs": "^3.1.6",
    "execa": "^5.1.1",
    "fs-extra": "^10.0.0",
    "glob": "^7.1.7",
    "inquirer": "^8.1.2",
    "isbinaryfile": "^4.0.8",
    "vue-codemod": "^0.0.5"
  }
}
```

2.4.3 utils\package.json

packages\utils\package.json

```
{
  "dependencies": {
    "@vite100/settings": "^0.0.0",
    "axios": "^0.21.2",
    "cross-spawn": "^7.0.3",
    "userhome": "^1.0.0",
    "npmlog": "^5.0.1",
    "ora": "^6.0.0",
    "userhome": "^1.0.0"
  }
}
```

2.4.4 vite100\package.json

packages\vite100\package.json

```
{
  "dependencies": {
    "@vite100/config": "^0.0.0",
    "@vite100/create": "^0.0.0"
  }
}
```

2.4.5 publishConfig

```
{
  "publishConfig": {
    "access": "public",
    "registry": "http://registry.npmjs.org"
  }
}
```

2.5 配置命令

2.5.1 package.json

packages\vite100\package.json

```
{
  "name": "vite100",
  "version": "0.0.0",
  "dependencies": {
    "@vite100/config":"^0.0.0",
    "@vite100/create":"^0.0.0"
  },
+ "bin":{
+   "vite100": "index.js"
+ }
}
```

2.5.2 index.js

packages\vite100\index.js

```
async function main() {
   let argv = process.argv.slice(2);
   console.log(argv);
}

main().catch((err) => {
    console.error(err);
});
```

- 一定要先添加 #!/usr/bin/env node再link,否则会用文本编辑器打开
- 这种情况可以 vite100\packages\vite100目录中执行 yarn unlink，再重新link

```
yarn link
yarn global bin
C:\Users\zhangrenyang\AppData\Local\Yarn\bin
C:\Users\zhangrenyang\AppData\Local\Yarn\Data\link\vite100
npm bin -g
C:\Users\zhangrenyang\AppData\Roaming\npm
vite100 create vite-project
```

3.实现配置命令

3.1 安装依赖

```
yarn workspace @vite100/config  add userhome fs-extra
yarn workspace @vite100/utils  add cross-spawn userhome fs-extra
```

3.2 settings\index.js

packages\settings\index.js

```
exports.COMMAND_SOURCE = `
const args = JSON.parse(process.argv[1]);
const factory = require('.');
factory(args);
`

exports.RC_NAME = ".vite100rc";
```

3.3 config.js

packages\utils\config.js

```
const userhome = require("userhome");
const fs = require("fs-extra");
const { RC_NAME } = require("@vite100/settings");
const configPath = userhome(RC_NAME);
let config = {};
if (fs.existsSync(configPath)) {
  config = fs.readJSONSync(configPath);
}
config.configPath=configPath;
module.exports = config;
```

3.4 executeNodeScript.js

packages\utils\executeNodeScript.js

```
const spawn = require("cross-spawn");
async function executeNodeScript({ cwd }, source, args) {
    return new Promise((resolve) => {
      const childProcess = spawn(
        process.execPath,
        ["-e", source, "--", JSON.stringify(args)],
        { cwd, stdio: "inherit" }
      );
      childProcess.on("close", resolve);
    });
}
module.exports = executeNodeScript;
```

3.5 log.js

packages\utils\log.js

```
const log = require('npmlog');
log.heading = 'vite100';
module.exports = log;
```

3.6 utils\index.js

packages\utils\index.js

```
exports.log = require('./log');
exports.executeNodeScript = require('./executeNodeScript');
exports.config = require('./config');
```

3.7 config\command.js

packages\config\command.js

```
const {executeNodeScript} = require('@vite100/utils');
const {COMMAND_SOURCE} = require('@vite100/settings');
const command = {
  command: "config [key] [value]",
  describe: "设置或查看配置项,比如GIT_TYPE设置仓库类型，ORG_NAME设置组织名",
  builder: (yargs) => {},
  handler:async function(argv){

    require('.')(argv);
  },
};
module.exports = command;
```

3.8 config\index.js

packages\config\index.js

```
const fs = require("fs-extra");
const { log ,config} = require("@vite100/utils");
async function factory(argv) {
  const { key, value } = argv;
  if (key && value) {
    config[key] = value;
    await fs.writeJSON(config.configPath, config, { spaces: 2 });
    log.info("vite100","(%s=%s)配置成功保存至%s", key, value, config.configPath);
  }else if(key){
      console.log('%s=%s',key, config[key]);
  }else{
    console.log(config);
  }
}
module.exports = factory;
```

3.9 vite100\index.js

packages\vite100\index.js

```
#!/usr/bin/env node
const yargs = require("yargs/yargs");
const configCmd = require("@vite100/config/command");
async function main() {
    const cli = yargs();
    cli
    .usage(`Usage: vite100  [options]`)
    .demandCommand(1, "至少需要一个命令")
    .strict()
    .recommendCommands()
+   .command(configCmd)
    .parse(process.argv.slice(2));
}

main().catch((err) => {
    console.error(err);
});
```

4.创建项目目录

4.1 create\command.js

packages\create\command.js

```
const {COMMAND_SOURCE} = require('@vite100/settings');
const {executeNodeScript} = require('@vite100/utils');
const command = {
  command: "create ",
  describe: "创建项目",
  builder: (yargs) => {
    yargs.positional("name", {
      type: "string",
      describe: "项目名称",
    });
  },
  handler:async function(argv){
    let args = {projectName:argv.name,workingDirectory:process.cwd()};

    require('.')(args);
  }
};
module.exports = command;
```

4.2 create\index.js

packages\create\index.js

```
const path = require("path");
const fs = require("fs-extra");
const execa = require("execa");
const { red } = require("chalk");
const { config, log } = require("@vite100/utils");

async function create(argv) {
    const { workingDirectory, projectName } = argv;
    const { GIT_TYPE, ORG_NAME } = config;
    if (!GIT_TYPE) {
        throw new Error(red("X") + " 尚未配置仓库类型!");
    }
    if (!ORG_NAME) {
        throw new Error(red("X") + " 尚未配置组织名称!");
    }
    const projectDir = path.join(workingDirectory, projectName);
    log.info("vite100", "创建的项目目录为%s", projectDir);
}
module.exports = (...args) => {
    return create(...args).catch(err => {
        console.error(err);
    });
};
```

4.3 vite100\index.js

packages\vite100\index.js

```
#!/usr/bin/env node
const yargs = require("yargs/yargs");
const configCmd = require("@vite100/config/command");
+const createCmd = require("@vite100/create/command");
async function main() {
    const cli = yargs();
    cli
    .usage(`Usage: vite100  [options]`)
    .demandCommand(1, "至少需要一个命令")
    .strict()
    .recommendCommands()
+   .command(createCmd)
    .command(configCmd)
    .parse(process.argv.slice(2));
 }

 main().catch((err) => {
     console.error(err);
 });
```

5. 获取选择项

5.1 router.js

packages\create\lib\promptModules\router.js

```
module.exports = cli => {
    cli.injectFeature({
        name: 'Router',
        value: 'router',
        description: '请选择路由模式',
        link: 'https://www.npmjs.com/package/react-router-dom'
    })
    cli.injectPrompt({
        name: 'historyMode',
        when: answers => answers.features.includes('router'),
        message: '请选择history的模式',
        type: 'list',
        choices: [
            {
                name: 'hash',
                value: 'hash'
            },
            {
                name: 'browser',
                value: 'browser'
            }
        ],
        default: 'browser'
    })
    cli.injectPrompt({
        name: 'appTitle',
        when: answers => answers.features.includes('router'),
        message: '请输入根组件的标题',
        type: 'text',
        default: 'AppTitle'
    })
    cli.onPromptComplete((answers, projectOptions) => {
        if (answers.features.includes('router')) {
            projectOptions.historyMode =answers.historyMode;
            projectOptions.appTitle=answers.appTitle;
        }
    })
}
```

5.2 getPromptModules.js

packages\create\lib\getPromptModules.js

```
function getPromptModules() {
    return ['router'].map(file => require(`./promptModules/${file}`));
}
module.exports = getPromptModules;
```

5.3 create\index.js

packages\create\index.js

```
const path = require("path");
const { red } = require("chalk");
const { config, log } = require("@vite100/utils");
+const getPromptModules = require('./lib/getPromptModules');
async function create(argv) {
    const { workingDirectory, projectName } = argv;
    const { GIT_TYPE, ORG_NAME } = config;
    if (!GIT_TYPE) {
        throw new Error(red("X") + " 尚未配置仓库类型!");
    }
    if (!ORG_NAME) {
        throw new Error(red("X") + " 尚未配置组织名称!");
    }
    const projectDir = path.join(workingDirectory, projectName);
    log.info("vite100", "创建的项目目录为%s", projectDir);
+   let promptModules = getPromptModules();
+   console.info("选择项promptModules", promptModules);
}
module.exports = (...args) => {
    return create(...args).catch(err => {
        console.error(err);
    });
};
```

6. 获取回答

6.1 PromptModuleAPI.js

packages\create\lib\PromptModuleAPI.js

```
class PromptModuleAPI {
    constructor(creator) {
        this.creator = creator;
    }

    injectFeature(feature) {
        this.creator.featurePrompt.choices.push(feature)
    }

    injectPrompt(prompt) {
        this.creator.injectedPrompts.push(prompt)
    }

    onPromptComplete(cb) {
        this.creator.promptCompleteCbs.push(cb)
    }
}
module.exports = PromptModuleAPI;
```

6.2 Creator.js

packages\create\lib\Creator.js

```
const { prompt } = require("inquirer");
const PromptModuleAPI = require("./PromptModuleAPI");
const defaultFeaturePrompt = {
    name: "features",
    type: "checkbox",
    message: "请选择项目特性:",
    choices: [],
}
class Creator {
    constructor(projectName, projectDir, promptModules) {
        this.projectName = projectName;
        this.projectDir = projectDir;
        this.featurePrompt = defaultFeaturePrompt;
        this.injectedPrompts = [];
        this.promptCompleteCbs = [];
        const promptModuleAPI = new PromptModuleAPI(this);
        promptModules.forEach((module) => module(promptModuleAPI));
    }
    async create() {

        const projectOptions = (this.projectOptions = await this.promptAndResolve());
        console.log('projectOptions',projectOptions);

    }
    async promptAndResolve() {
        let prompts = [this.featurePrompt, ...this.injectedPrompts];
        let answers = await prompt(prompts);
        let projectOptions = {plugins: {},};
        this.promptCompleteCbs.forEach((cb) => cb(answers, projectOptions));
        return projectOptions;
      }
}
module.exports = Creator;
```

6.3 create\index.js

packages\create\index.js

```
const path = require("path");
const { red } = require("chalk");
const { config, log } = require("@vite100/utils");
const getPromptModules = require('./lib/getPromptModules');
+const Creator = require('./lib/Creator');
async function create(argv) {
    const { workingDirectory, projectName } = argv;
    const { GIT_TYPE, ORG_NAME } = config;
    if (!GIT_TYPE) {
        throw new Error(red("X") + " 尚未配置仓库类型!");
    }
    if (!ORG_NAME) {
        throw new Error(red("X") + " 尚未配置组织名称!");
    }
    const projectDir = path.join(workingDirectory, projectName);
    log.info("vite100", "创建的项目目录为%s", projectDir);
    let promptModules = getPromptModules();
    console.info("选择项promptModules", promptModules);
+   let creator = new Creator(projectName, projectDir, promptModules);
+   await creator.create();
}
module.exports = (...args) => {
    return create(...args).catch(err => {
        console.error(err);
    });
};
```

7. 准备项目目录

7.1 Creator.js

packages\create\lib\Creator.js

```
const { prompt } = require("inquirer");
+const fs = require("fs-extra");
+const { red } = require("chalk");
const PromptModuleAPI = require("./PromptModuleAPI");
+const { log } = require("@vite100/utils");
const defaultFeaturePrompt = {
    name: "features",
    type: "checkbox",
    message: "请选择项目特性:",
    choices: [],
}
class Creator {
    constructor(projectName, projectDir, promptModules) {
        this.projectName = projectName;//项目名称
        this.projectDir = projectDir;//项目路径
        this.featurePrompt = defaultFeaturePrompt;//默认选项框
        this.injectedPrompts = [];//插入插入的选择框
        this.promptCompleteCbs = [];//选择结束之后的回调
        const promptModuleAPI = new PromptModuleAPI(this);
        promptModules.forEach((module) => module(promptModuleAPI));
    }
    async create() {
        //获取选择项
        const projectOptions = (this.projectOptions = await this.promptAndResolve());
        console.log('projectOptions', projectOptions);
        //{historyMode: 'browser',appTitle: 'AppTitle'}
+       //准备项目目录
+       await this.prepareProjectDir();
    }
+   async prepareProjectDir() {
+       let { projectDir } = this;
+       try {
+           await fs.access(projectDir);
+           const files = await fs.readdir(projectDir);
+           if (files.length > 0) {
+               const { overwrite } = await prompt({
+                   type: "confirm",
+                   name: "overwrite",
+                   message: `目标目录非空，是否要移除存在的文件并继续?`,
+               });
+               if (overwrite) {
+                   await fs.emptyDir(projectDir);
+               } else {
+                   throw new Error(red("X") + " 操作被取消");
+               }
+           }
+       } catch (error) {
+           await fs.mkdirp(projectDir);
+       }
+       log.info("vite100", "%s目录已经准备就绪", projectDir);
+   }
    async promptAndResolve() {
        let prompts = [this.featurePrompt, ...this.injectedPrompts];
        let answers = await prompt(prompts);
        let projectOptions = { plugins: {}, };
        this.promptCompleteCbs.forEach((cb) => cb(answers, projectOptions));
        return projectOptions;
    }
}
module.exports = Creator;
```

8. 下载模板

8.1 request.js

packages\utils\request.js

```
const axios = require("axios");
const { GIT_TYPE } = require("./config");
const GITEE = "https://gitee.com/api/v5";
const GITHUB = "https://api.github.com";

const BASE_URL = GIT_TYPE === "gitee" ? GITEE : GITHUB;
const request = axios.create({
    baseURL: BASE_URL,
    timeout: 5000,
});

request.interceptors.response.use(
    (response) => {
        return response.data;
    },
    (error) => {
        return Promise.reject(error);
    }
);

module.exports = request;
```

8.2 withLoading.js

packages\utils\withLoading.js

```
async function withLoading(message, fn, ...args) {
  const ora = await import("ora");
  const spinner = ora.default(message);
  spinner.start();
  const result = await fn(...args);
  spinner.succeed();
  return result;
}
module.exports = withLoading;
```

8.3 utils\index.js

packages\utils\index.js

```
exports.log = require('./log');
exports.executeNodeScript = require('./executeNodeScript');
exports.config = require('./config');
+exports.withLoading = require('./withLoading');
+exports.request = require('./request');
```

8.4 settings\index.js

packages\settings\index.js

```
//执行命令脚本
exports.COMMAND_SOURCE = `
const args = JSON.parse(process.argv[1]);
const factory = require('.');
factory(args);
`
//配置文件名称
exports.RC_NAME = ".vite100rc";

+//模板存放名称
+exports.TEMPLATES = ".vite100_templates";
```

8.5 Creator.js

packages\create\lib\Creator.js

```
const { prompt } = require("inquirer");
const fs = require("fs-extra");
const { red } = require("chalk");
+const userhome = require("userhome");
+const {promisify} = require('util');
+const clone = promisify(require('clone-git-repo'));
const PromptModuleAPI = require("./PromptModuleAPI");
+const { log,config,withLoading ,request} = require("@vite100/utils");
+const { TEMPLATES } = require("@vite100/settings");
const defaultFeaturePrompt = {
    name: "features",
    type: "checkbox",
    message: "请选择项目特性:",
    choices: [],
}
class Creator {
    constructor(projectName, projectDir, promptModules) {
        this.projectName = projectName;//项目名称
        this.projectDir = projectDir;//项目路径
        this.featurePrompt = defaultFeaturePrompt;//默认选项框
        this.injectedPrompts = [];//插入插入的选择框
        this.promptCompleteCbs = [];//选择结束之后的回调
        const promptModuleAPI = new PromptModuleAPI(this);
        promptModules.forEach((module) => module(promptModuleAPI));
    }
    async create() {
        //获取选择项
        const projectOptions = (this.projectOptions = await this.promptAndResolve());
        console.log('projectOptions', projectOptions);
        //{historyMode: 'browser',appTitle: 'AppTitle'}
        //准备项目目录
        await this.prepareProjectDir();
+       //下载模板，给templateDir赋值
+       await this.downloadTemplate();
    }
+    async downloadTemplate() {
+        const { GIT_TYPE, ORG_NAME } = config;
+        let repos = await withLoading("读取模板列表", async () =>
+            request.get(`/orgs/${ORG_NAME}/repos`)
+        );
+        let { repo } = await prompt({
+            name: "repo",
+            type: "list",
+            message: "请选择模板",
+            choices: repos.map((repo) => repo.name)
+        });
+        let tags = await withLoading("读取标签列表", async () =>
+            request.get(`/repos/${ORG_NAME}/${repo}/tags`)
+        );
+        let { tag } = await prompt({
+            name: "tag",
+            type: "list",
+            message: "请选择版本",
+            choices: tags,
+        });
+        let repository = GIT_TYPE + `:${ORG_NAME}/${repo}`;
+        if (tag) repository += `#${tag}`;
+        const downloadDirectory = userhome(TEMPLATES);
+        let templateDir = (this.templateDir = `${downloadDirectory}/${repo}/${tag}`);
+        log.info("vite3", "准备下载模板到%s", templateDir);
+        try {
+            await fs.access(templateDir);
+        } catch (error) {
+            log.info("vite100", "从仓库下载%s", repository);
+            await clone(repository, templateDir, { clone: true });
+        }
+    }
    async prepareProjectDir() {
        let { projectDir } = this;
        try {
            await fs.access(projectDir);
            const files = await fs.readdir(projectDir);
            if (files.length > 0) {
                const { overwrite } = await prompt({
                    type: "confirm",
                    name: "overwrite",
                    message: `目标目录非空，是否要移除存在的文件并继续?`,
                });
                if (overwrite) {
                    await fs.emptyDir(projectDir);
                } else {
                    throw new Error(red("X") + " 操作被取消");
                }
            }
        } catch (error) {
            await fs.mkdirp(projectDir);
        }
        log.info("vite100", "%s目录已经准备就绪", projectDir);
    }
    async promptAndResolve() {
        let prompts = [this.featurePrompt, ...this.injectedPrompts];
        let answers = await prompt(prompts);
        let projectOptions = { plugins: {}, };
        this.promptCompleteCbs.forEach((cb) => cb(answers, projectOptions));
        return projectOptions;
    }
}
module.exports = Creator;
```

9.启动项目

9.1 Creator.js

packages\create\lib\Creator.js

```
const { prompt } = require("inquirer");
const fs = require("fs-extra");
const { red } = require("chalk");
const userhome = require("userhome");
const {promisify} = require('util');
+const execa = require("execa");
const clone = promisify(require('clone-git-repo'));
const PromptModuleAPI = require("./PromptModuleAPI");
const { log,config,withLoading ,request} = require("@vite100/utils");
const { TEMPLATES } = require("@vite100/settings");
const defaultFeaturePrompt = {
    name: "features",
    type: "checkbox",
    message: "请选择项目特性:",
    choices: [],
}
class Creator {
    constructor(projectName, projectDir, promptModules) {
        this.projectName = projectName;//项目名称
        this.projectDir = projectDir;//项目路径
        this.featurePrompt = defaultFeaturePrompt;//默认选项框
        this.injectedPrompts = [];//插入插入的选择框
        this.promptCompleteCbs = [];//选择结束之后的回调
        const promptModuleAPI = new PromptModuleAPI(this);
        promptModules.forEach((module) => module(promptModuleAPI));
    }
    async create() {
        //获取选择项
        const projectOptions = (this.projectOptions = await this.promptAndResolve());
        console.log('projectOptions', projectOptions);
        //{historyMode: 'browser',appTitle: 'AppTitle'}
        //准备项目目录
        await this.prepareProjectDir();
        //下载模板，给templateDir赋值
        await this.downloadTemplate();
         //把项目拷贝到模板中
+       await fs.copy(this.templateDir, this.projectDir);
+        //初始化git仓库
+       await execa("git", ["init"], { cwd: this.projectDir, stdio: "inherit" });
+       log.info("vite100", "在%s安装依赖", this.projectDir);
+       await execa("npm", ["install"], { cwd: this.projectDir, stdio: "inherit" });
    }
    async downloadTemplate() {
        const { GIT_TYPE, ORG_NAME } = config;
        let repos = await withLoading("读取模板列表", async () =>
            request.get(`/orgs/${ORG_NAME}/repos`)
        );
        let { repo } = await prompt({
            name: "repo",
            type: "list",
            message: "请选择模板",
            choices: repos.map((repo) => repo.name)
        });
        let tags = await withLoading("读取标签列表", async () =>
            request.get(`/repos/${ORG_NAME}/${repo}/tags`)
        );
        let { tag } = await prompt({
            name: "tag",
            type: "list",
            message: "请选择版本",
            choices: tags,
        });
        let repository = GIT_TYPE + `:${ORG_NAME}/${repo}`;
        if (tag) repository += `#${tag}`;
        const downloadDirectory = userhome(TEMPLATES);
        let templateDir = (this.templateDir = `${downloadDirectory}/${repo}/${tag}`);
        log.info("vite3", "准备下载模板到%s", templateDir);
        try {
            await fs.access(templateDir);
        } catch (error) {
            log.info("vite100", "从仓库下载%s", repository);
            await clone(repository, templateDir, { clone: true });
        }
    }
    async prepareProjectDir() {
        let { projectDir } = this;
        try {
            await fs.access(projectDir);
            const files = await fs.readdir(projectDir);
            if (files.length > 0) {
                const { overwrite } = await prompt({
                    type: "confirm",
                    name: "overwrite",
                    message: `目标目录非空，是否要移除存在的文件并继续?`,
                });
                if (overwrite) {
                    await fs.emptyDir(projectDir);
                } else {
                    throw new Error(red("X") + " 操作被取消");
                }
            }
        } catch (error) {
            await fs.mkdirp(projectDir);
        }
        log.info("vite100", "%s目录已经准备就绪", projectDir);
    }
    async promptAndResolve() {
        let prompts = [this.featurePrompt, ...this.injectedPrompts];
        let answers = await prompt(prompts);
        let projectOptions = { plugins: {}, };
        this.promptCompleteCbs.forEach((cb) => cb(answers, projectOptions));
        return projectOptions;
```

```
    }
}
module.exports = Creator;
```

9.2 create\index.js

packages\create\index.js

```
const path = require("path");
const { red } = require("chalk");
+const execa = require("execa");
const { config, log } = require("@vite100/utils");
const getPromptModules = require('./lib/getPromptModules');
const Creator = require('./lib/Creator');
async function create(argv) {
    const { workingDirectory, projectName } = argv;
    const { GIT_TYPE, ORG_NAME } = config;
    if (!GIT_TYPE) {
        throw new Error(red("X") + " 尚未配置仓库类型!");
    }
    if (!ORG_NAME) {
        throw new Error(red("X") + " 尚未配置组织名称!");
    }
    const projectDir = path.join(workingDirectory, projectName);
    log.info("vite100", "创建的项目目录为%s", projectDir);
    let promptModules = getPromptModules();
    console.info("选择项promptModules", promptModules);
    let creator = new Creator(projectName, projectDir, promptModules);
    await creator.create();
+   log.info("vite100", "启动服务");
+   await execa("npm", ["run", "dev"], { cwd: projectDir, stdio: "inherit" });
}
module.exports = (...args) => {
    return create(...args).catch(err => {
        console.error(err);
    });
};
```

10.添加路由插件

10.1 router.js

packages\create\lib\promptModules\router.js

```
module.exports = cli => {
    cli.injectFeature({
        name: 'Router',
        value: 'router',
        description: '请选择路由模式',
        link: 'https://www.npmjs.com/package/react-router-dom'
    })
    cli.injectPrompt({
        name: 'historyMode',
        when: answers => answers.features.includes('router'),
        message: '请选择history的模式',
        type: 'list',
        choices: [
            {
                name: 'hash',
                value: 'hash'
            },
            {
                name: 'browser',
                value: 'browser'
            }
        ],
        default: 'browser'
    })
    cli.injectPrompt({
        name: 'appTitle',
        when: answers => answers.features.includes('router'),
        message: '请输入根组件的标题',
        type: 'text',
        default: 'AppTitle'
    })
    cli.onPromptComplete((answers, projectOptions) => {
        if (answers.features.includes('router')) {
+            projectOptions.plugins['cli-plugin-router'] = {
+                historyMode: answers.historyMode
+            }
            projectOptions.historyMode =answers.historyMode;
            projectOptions.appTitle=answers.appTitle;
        }
    })
}
```

10.2 Creator.js

packages\create\lib\Creator.js

```
const { prompt } = require("inquirer");
+const path = require("path");
const fs = require("fs-extra");
const { red } = require("chalk");
const userhome = require("userhome");
const {promisify} = require('util');
const execa = require("execa");
const clone = promisify(require('clone-git-repo'));
const PromptModuleAPI = require("./PromptModuleAPI");
const { log,config,withLoading ,request} = require("@vite100/utils");
const { TEMPLATES } = require("@vite100/settings");
const defaultFeaturePrompt = {
    name: "features",
    type: "checkbox",
```

```
    message: "请选择项目特性:",
    choices: [],
}
class Creator {
    constructor(projectName, projectDir, promptModules) {
        this.projectName = projectName;//项目名称
        this.projectDir = projectDir;//项目路径
        this.featurePrompt = defaultFeaturePrompt;//默认选项框
        this.injectedPrompts = [];//插入插入的选择框
        this.promptCompleteCbs = [];//选择结束之后的回调
+       this.plugins = [];//插件
        const promptModuleAPI = new PromptModuleAPI(this);
        promptModules.forEach((module) => module(promptModuleAPI));
    }
    async create() {
        //获取选择项
        const projectOptions = (this.projectOptions = await this.promptAndResolve());
        console.log('projectOptions', projectOptions);
        //{historyMode: 'browser',appTitle: 'AppTitle'}
        //准备项目目录
        await this.prepareProjectDir();
        //下载模板，给templateDir赋值
        await this.downloadTemplate();
         //把项目拷贝到模板中
        await fs.copy(this.templateDir, this.projectDir);
+       const pkgPath = path.join(this.projectDir, 'package.json');
+       let pkg = (this.pkg = await fs.readJSON(pkgPath));
+       const deps = Reflect.ownKeys(projectOptions.plugins);
+       deps.forEach(dep => pkg.devDependencies[dep] = `latest`);
+       await fs.writeJSON(pkgPath,pkg,{spaces:2});
         //初始化git仓库
        await execa("git", ["init"], { cwd: this.projectDir, stdio: "inherit" });
        log.info("vite100", "在%s安装依赖", this.projectDir);
        await execa("npm", ["install"], { cwd: this.projectDir, stdio: "inherit" });

    }
    async downloadTemplate() {
        const { GIT_TYPE, ORG_NAME } = config;
        let repos = await withLoading("读取模板列表", async () =>
            request.get(`/orgs/${ORG_NAME}/repos`)
        );
        let { repo } = await prompt({
            name: "repo",
            type: "list",
            message: "请选择模板",
            choices: repos.map((repo) => repo.name)
        });
        let tags = await withLoading("读取标签列表", async () =>
            request.get(`/repos/${ORG_NAME}/${repo}/tags`)
        );
        let { tag } = await prompt({
            name: "tag",
            type: "list",
            message: "请选择版本",
            choices: tags,
        });
        let repository = GIT_TYPE + `:${ORG_NAME}/${repo}`;
        if (tag) repository += `#${tag}`;
        const downloadDirectory = userhome(TEMPLATES);
        let templateDir = (this.templateDir = `${downloadDirectory}/${repo}/${tag}`);
        log.info("vite3", "准备下载模板到%s", templateDir);
        try {
            await fs.access(templateDir);
        } catch (error) {
            log.info("vite100", "从仓库下载%s", repository);
            await clone(repository, templateDir, { clone: true });
        }
    }
    async prepareProjectDir() {
        let { projectDir } = this;
        try {
            await fs.access(projectDir);
            const files = await fs.readdir(projectDir);
            if (files.length > 0) {
                const { overwrite } = await prompt({
                    type: "confirm",
                    name: "overwrite",
                    message: `目标目录非空，是否要移除存在的文件并继续?`,
                });
                if (overwrite) {
                    await fs.emptyDir(projectDir);
                } else {
                    throw new Error(red("X") + " 操作被取消");
                }
            }
        } catch (error) {
            await fs.mkdirp(projectDir);
        }
        log.info("vite100", "%s目录已经准备就绪", projectDir);
    }
    async promptAndResolve() {
        let prompts = [this.featurePrompt, ...this.injectedPrompts];
        let answers = await prompt(prompts);
        let projectOptions = { plugins: {}, };
        this.promptCompleteCbs.forEach((cb) => cb(answers, projectOptions));
        return projectOptions;
    }
}
module.exports = Creator;
```

11. 解析并应用插件

11.1 mergeDeps.js

packages\utils\mergeDeps.js

```
function mergeDeps(sourceDeps, depsToInject){
    let result = Object.assign({}, sourceDeps);
    for (const depName in depsToInject) {
        result[depName] = depsToInject[depName];
    }
    return result;
}
module.exports = mergeDeps;
```

11.2 loadModule.js

packages\utils\loadModule.js

```
const path = require('path');
const Module = require('module');
function loadModule(request,context){
  return Module.createRequire(path.resolve(context,'package.json'))(request);
}
module.exports = loadModule;
```

11.3 extractCallDir.js

packages\utils\extractCallDir.js

```
const path = require('path');
function extractCallDir(){
    const obj = {}
    Error.captureStackTrace(obj)
    const callSite = obj.stack.split('\n')[3]
    const namedStackRegExp = /\s\((.*):\d+:\d+\)$/
    let matchResult = callSite.match(namedStackRegExp)
    const fileName = matchResult[1]
    return path.dirname(fileName)
}
module.exports = extractCallDir;
```

11.4 writeFileTree.js

packages\utils\writeFileTree.js

```
const path = require('path');
const fs = require('fs-extra');
async function writeFileTree(projectDir, files) {
    Object.keys(files).forEach(file => {
        let content = files[file];
        if (file.endsWith('.ejs')) file = file.slice(0, -4);
        const filePath = path.join(projectDir, file);
        fs.ensureDirSync(path.dirname(filePath));
        fs.writeFileSync(filePath,content);
    });
}

module.exports = writeFileTree;
```

11.5 utils\index.js

packages\utils\index.js

```
+const path = require('path');
exports.log = require('./log');
exports.executeNodeScript = require('./executeNodeScript');
exports.config = require('./config');
exports.withLoading = require('./withLoading');
exports.request = require('./request');
+exports.loadModule = require('./loadModule');
+exports.mergeDeps = require('./mergeDeps');
+exports.extractCallDir = require('./extractCallDir');
+exports.writeFileTree = require('./writeFileTree');
+exports.isObject = val => typeof val === 'object';
+exports.isString = val => typeof val === 'string';
```

11.6 GeneratorAPI.js

packages\create\lib\GeneratorAPI.js

```
const fs = require('fs');
const ejs = require('ejs');
const path = require('path');
const { promisify } = require('util');
const glob = promisify(require('glob'));
const { isBinaryFile } = require('isbinaryfile');
const { runTransformation } = require('vue-codemod')
const { isObject, isString, extractCallDir, mergeDeps } = require("@vite100/utils");
class GeneratorAPI {
    constructor(id, creator, projectOptions) {
        this.id = id;
        this.creator = creator;
        this.projectOptions = projectOptions;
    }

    async _injectFileMiddleware(middleware) {
        this.creator.fileMiddlewares.push(middleware);
    }

    render(source) {
        const baseDir = extractCallDir();
        if (isString(source)) {
            source = path.resolve(baseDir, source)
            this._injectFileMiddleware(async (files, projectOptions) => {
                const templateFiles = await glob('**/*', { cwd: source, nodir: true });
                for (let i = 0; i < templateFiles.length; i++) {
                    let templateFile = templateFiles[i];
                    files[templateFile] = await renderFile(path.resolve(source, templateFile), projectOptions);
                }
            });
        }
    }

    extendPackage(toMerge) {
        const pkg = this.creator.pkg;
        for (const key in toMerge) {
            const value = toMerge[key];
            let existing = pkg[key];
            if (isObject(value) && (key === 'dependencies' || key === 'devDependencies')) {
                pkg[key] = mergeDeps(existing || {}, value);
            } else {
                pkg[key] = value;
            }
        }
    }

    transformScript(file, codemod, projectOptions = {}) {
        this._injectFileMiddleware((files) => {
            files[file] = runTransformation(
                {
                    path: file,
                    source: files[file]
                },
                codemod,
                projectOptions
            )
        })
    }

    injectImport(file, newImport) {
        const imports = (this.creator.imports[file] = this.creator.imports[file] || []);
        imports.push(newImport)
    }

    get entryFile() {
        return 'src/index.js';
    }
}

async function renderFile(templatePath, projectOptions) {
    if (await isBinaryFile(templatePath)) {
        return fs.readFileSync(templatePath);
    }
    let template = fs.readFileSync(templatePath, 'utf8');
    return ejs.render(template, projectOptions);
}

module.exports = GeneratorAPI;
```

11.7 injectImports.js

packages\create\lib\codemods\injectImports.js

```
function injectImports(fileInfo, api, { imports }) {
    const jscodeshift = api.jscodeshift
    const root = jscodeshift(fileInfo.source)
    const declarations = root.find(jscodeshift.ImportDeclaration)
    const toImportAST = imp => jscodeshift(`${imp}\n`).nodes()[0].program.body[0]
    const importASTNodes = imports.map(toImportAST);
    if (declarations.length) {
        declarations.at(-1).insertAfter(importASTNodes)
    } else {
        root.get().node.program.body.unshift(...importASTNodes)
    }
    return root.toSource()
}

module.exports = injectImports;
```

11.8 Creator.js

packages\create\lib\Creator.js

```
const { prompt } = require("inquirer");
```

```
const path = require("path");
const fs = require("fs-extra");
const { red } = require("chalk");
const userhome = require("userhome");
const {promisify} = require('util');
const execa = require("execa");
+const glob = promisify(require('glob'));
const clone = promisify(require('clone-git-repo'));
+const { runTransformation } = require('vue-codemod')
const PromptModuleAPI = require("./PromptModuleAPI");
+const GeneratorAPI = require('./GeneratorAPI');
+const { isBinaryFile } = require('isbinaryfile');
+const { log,config,withLoading ,request,loadModule,writeFileTree} = require("@vite100/utils");
const { TEMPLATES } = require("@vite100/settings");
const defaultFeaturePrompt = {
    name: "features",
    type: "checkbox",
    message: "请选择项目特性:",
    choices: [],
}
class Creator {
    constructor(projectName, projectDir, promptModules) {
        this.projectName = projectName;//项目名称
        this.projectDir = projectDir;//项目路径
        this.featurePrompt = defaultFeaturePrompt;//默认选项框
        this.injectedPrompts = [];//插入插入的选择框
        this.promptCompleteCbs = [];//选择结束之后的回调
        this.plugins = [];//插件
+       this.fileMiddlewares = [];//文件中间件
+       this.files = {};//最终输出的文件列表
+       this.pkg = {};//我描述内容
+       this.imports={};//额外的导入语句
        const promptModuleAPI = new PromptModuleAPI(this);
        promptModules.forEach((module) => module(promptModuleAPI));
    }
    async create() {
        //获取选择项
        const projectOptions = (this.projectOptions = await this.promptAndResolve());
        console.log('projectOptions', projectOptions);
        //{historyMode: 'browser',appTitle: 'AppTitle'}
        //准备项目目录
        await this.prepareProjectDir();
        //下载模板，给templateDir赋值
        await this.downloadTemplate();
         //把项目拷贝到模板中
        await fs.copy(this.templateDir, this.projectDir);
        const pkgPath = path.join(this.projectDir, 'package.json');
        let pkg = (this.pkg = await fs.readJSON(pkgPath));
        const deps = Reflect.ownKeys(projectOptions.plugins);
        deps.forEach(dep => pkg.devDependencies[dep] = `latest`);
        await fs.writeJSON(pkgPath,pkg,{spaces:2});
         //初始化git仓库
        await execa("git", ["init"], { cwd: this.projectDir, stdio: "inherit" });
        log.info("vite100", "在%s安装依赖", this.projectDir);
        await execa("npm", ["install"], { cwd: this.projectDir, stdio: "inherit" });
+       //解析插件,拿到插件的generator方法
+       const resolvedPlugins = await this.resolvePlugins(projectOptions.plugins);
+       //应用插件
+       await this.applyPlugins(resolvedPlugins);
+       await this.initFiles();
+       //准备文件内容
+       await this.renderFiles();
+       //删除插件的依赖
+       deps.forEach(dep =>  delete pkg.devDependencies[dep]);
+       this.files['package.json'] = JSON.stringify(pkg,null,2);
+       //把文件写入硬盘
+       await writeFileTree(this.projectDir, this.files);
+       //重新安装额外的依赖
+       await execa("npm", ["install"], { cwd: this.projectDir, stdio: "inherit" });
    }
+   async initFiles(){
+       const projectFiles = await glob('**/*', { cwd: this.projectDir, nodir: true });
+       for (let i = 0; i < projectFiles.length; i++) {
+           let projectFile = projectFiles[i];
+           let projectFilePath = path.join(this.projectDir,projectFile);
+           let content;
+           if (await isBinaryFile(projectFilePath)) {
+               content =  await fs.readFile(projectFilePath);
+           }else{
+               content =  await fs.readFile(projectFilePath,'utf8');
+           }
+           this.files[projectFile] = content;
+       }
+   }
+    async renderFiles() {
+        const {files,projectOptions} = this;
+        for (const middleware of this.fileMiddlewares) {
+          await middleware(files,projectOptions);
+        }
+        Object.keys(files).forEach(file => {
+          let imports = this.imports[file]
+          if (imports && imports.length > 0) {
+            files[file] = runTransformation(
+              { path: file, source: files[file] },
+              require('./codemods/injectImports'),
+              { imports }
+            )
+          }
+        })
+      }
+    async resolvePlugins(rawPlugins) {
+        const plugins = [];
+        for (const id of Reflect.ownKeys(rawPlugins)) {
+          const apply = loadModule(`${id}/generator`, this.projectDir);
```

```
+          let options = rawPlugins[id];
+          plugins.push({ id, apply, options });
+        }
+        return plugins;
+    }
+    async applyPlugins(plugins) {
+        for (const plugin of plugins) {
+          const { id, apply, options } = plugin;
+          const generatorAPI = new GeneratorAPI(id, this, options);
+          await apply(generatorAPI, options);
+        }
+      }
    async downloadTemplate() {
        const { GIT_TYPE, ORG_NAME } = config;
        let repos = await withLoading("读取模板列表", async () =>
            request.get(`/orgs/${ORG_NAME}/repos`)
        );
        let { repo } = await prompt({
            name: "repo",
            type: "list",
            message: "请选择模板",
            choices: repos.map((repo) => repo.name)
        });
        let tags = await withLoading("读取标签列表", async () =>
            request.get(`/repos/${ORG_NAME}/${repo}/tags`)
        );
        let { tag } = await prompt({
            name: "tag",
            type: "list",
            message: "请选择版本",
            choices: tags,
        });
        let repository = GIT_TYPE + `:${ORG_NAME}/${repo}`;
        if (tag) repository += `#${tag}`;
        const downloadDirectory = userhome(TEMPLATES);
        let templateDir = (this.templateDir = `${downloadDirectory}/${repo}/${tag}`);
        log.info("vite3", "准备下载模板到%s", templateDir);
        try {
            await fs.access(templateDir);
        } catch (error) {
            log.info("vite100", "从仓库下载%s", repository);
            await clone(repository, templateDir, { clone: true });
        }
    }
    async prepareProjectDir() {
        let { projectDir } = this;
        try {
            await fs.access(projectDir);
            const files = await fs.readdir(projectDir);
            if (files.length > 0) {
                const { overwrite } = await prompt({
                    type: "confirm",
                    name: "overwrite",
                    message: `目标目录非空，是否要移除存在的文件并继续?`,
                });
                if (overwrite) {
                    await fs.emptyDir(projectDir);
                } else {
                    throw new Error(red("X") + " 操作被取消");
                }
            }
        } catch (error) {
            await fs.mkdirp(projectDir);
        }
        log.info("vite100", "%s目录已经准备就绪", projectDir);
    }
    async promptAndResolve() {
        let prompts = [this.featurePrompt, ...this.injectedPrompts];
        let answers = await prompt(prompts);
        let projectOptions = { plugins: {}, };
        this.promptCompleteCbs.forEach((cb) => cb(answers, projectOptions));
        return projectOptions;
    }
}
module.exports = Creator;
```

12. 实现插件

12.1 App.js.ejs

packages\cli-plugin-router\template\src\App.js.ejs

```
import React from 'react';
function App(){
    return <div><%=appTitle%>div>
}
export default App;
```

12.2 routesConfig.js

packages\cli-plugin-router\template\src\routesConfig.js

```
import App from './App';
export default [
    {
        path: '/',
        component: App
    }
]
```

12.3 generator.js

packages\cli-plugin-router\generator.js

```
module.exports = async (api, options) => {
    api.render('./template');
    api.injectImport(api.entryFile,
        `import {${options.historyMode === 'hash' ? 'HashRouter' : 'BrowserRouter'} as Router,Route} from 'react-router-dom'`);
    api.injectImport(api.entryFile,
        `import routesConfig from './routesConfig'`);
    api.injectImport(api.entryFile,
        `import { renderRoutes } from "react-router-config"`);
    api.transformScript(api.entryFile, require('./injectRouter'))
    api.extendPackage({
        dependencies: {
            'react-router-dom': 'latest',
            'react-router-config': 'latest'
        }
    });
}
```

12.4 injectRouter.js

packages\cli-plugin-router\injectRouter.js

```
module.exports = (file, api) => {
    const jscodeshift = api.jscodeshift
    const root = jscodeshift(file.source)
    const appImportDeclaration = root.find(jscodeshift.ImportDeclaration, (node) => {
        if(node.specifiers[0].local.name === 'App'){
            return true
        }
    })
    if(appImportDeclaration)
        appImportDeclaration.remove();

    const appJSXElement = root.find(jscodeshift.JSXElement, (node) => {
        if (node.openingElement.name.name === 'App') {
            return true
        }
    })
    if(appJSXElement)
     appJSXElement.replaceWith(({ node }) => {
         return jscodeshift.jsxElement(
            jscodeshift.jsxOpeningElement(jscodeshift.jsxIdentifier('Router')), jscodeshift.jsxClosingElement(jscodeshift.jsxIdentifier('Router')), [
                jscodeshift.jsxExpressionContainer(
                    jscodeshift.callExpression(jscodeshift.identifier('renderRoutes'),[jscodeshift.identifier('routesConfig')])
                )
            ], false
         );
    })
    return root.toSource()
}
```

13.发布

13.1 创建组织

- create (https://www.npmjs.com/org/create)

13.2 发布

package.json

```
{
  "publishConfig": {
    "access": "public",
    "registry": "http://registry.npmjs.org"
  }
}
```

```
npm whoami
npm login
zhangrenyang2000
lerna publish
```

14.参考

14.1 lerna

命令 功能 lerna bootstrap 安装依赖 lerna clean 删除各个包下的node_modules lerna init 创建新的lerna库 lerna list 查看本地包列表 lerna changed 显示自上次release tag以来有修改的包， 选项通 list lerna diff 显示自上次release tag以来有修改的包的差异， 执行 git diff lerna exec 在每个包目录下执行任意命令 lerna run 执行每个包package.json中的脚本命令 lerna add 添加一个包的版本为各个包的依赖 lerna import 引入package lerna link 链接互相引用的库 lerna create 新建package lerna publish 发布

14.3 yarn

命令 说明 yarn -v 查看yarn版本 yarn config list 查看yarn的所有配置 yarn config set registry https://registry.npm.taobao.org/ (https://registry.npm.taobao.org/)

修改yarn的源镜像为淘宝源 yarn config set global-folder "D:\RTE\Yarn\global" 修改全局安装目录, 先创建好目录(global), 我放在了Yarn安装目录下(D:\RTE\Yarn\global) yarn config set prefix "D:\RTE\Yarn\global" 修改全局安装目录的bin目录位置 yarn config set cache-folder "D:\RTE\Yarn\cache" 修改全局缓存目录, 先创建好目录(cache), 和global放在同一层目录下 yarn config list 查看所有配置 yarn global bin 查看当前yarn的bin的位置 yarn global dir 查看当前yarn的全局安装位置

14.4 workspace

- yarn官网 (https://yarn.bootcss.com/docs/)
- yarn add
- yarn add

作用 命令 查看工作空间信息 yarn workspaces info 给所有的空间添加依赖 yarn workspaces run add lodash 给根空间添加依赖 yarn add -W -D typescript jest 给某个项目添加依赖 yarn workspace create-react-app3 add commander 删除所有的 node_modules lerna clean 等于 yarn workspaces run clean 安装和link所有的名 yarn install 等于 lerna bootstrap --npm-client yarn --use-workspaces 重新获取所有的 node_modules yarn install --force 查看缓存目录 yarn cache dir 清除本地缓存 yarn cache clean 在所有package中运行指定的命令 yarn workspaces run

14.3 yargs

- yargs (https://www.npmjs.com/package/yargs)帮助你构建交互命令行工具，可以解析参数生成优雅的用户界面

```
const yargs = require("yargs/yargs");
const cli = yargs();
cli
    .usage(`Usage: vite100  [options]`)
    .demandCommand(1, "至少需要一个命令")
    .strict()
    .recommendCommands()
    .command({
        command: "create ",
        describe: "创建项目",
        builder: (yargs) => {
            yargs.positional("name", {
                type: "string",
                describe: "项目名称",
            });
        },
        handler: async function (argv) {
           console.log(argv);

        }
    })
    .parse(process.argv.slice(2));
```

14.5 node -e

- node -e (https://nodejs.org/api/cli.html#cli_e_eval_script)可以直接执行一段js脚本并输入
- -e, --eval "script"
- 设置 stdion: 'inherit',当执行代码时,子进程将会继承主进程的stdin、stdout和stderr

```
node -e "console.log(process.argv)" -- a b
node -e "console.log(JSON.parse(process.argv[1]))" -- "{\"name\":\"zhufeng\"}"
node -e "console.log(process.cwd())"
```

```
const spawn = require("cross-spawn");
async function executeNodeScript({ cwd }, source, args) {
    return new Promise((resolve) => {
      const childProcess = spawn(
        process.execPath,
        ["-e", source, "--", JSON.stringify(args)],
        { cwd, stdio: "inherit" }
      );
      childProcess.on("close", resolve);
    });
}
module.exports = executeNodeScript;
```

14.6 clone

- clone-git-repo (https://www.npmjs.com/package/clone-git-repo)用来克隆和下载仓库

```
const clone = require('clone-git-repo');
let repository = 'gitee:zhufengtemplate/template-react#v1.0';
clone(repository,'./output', {clone:true},function (err) {
    console.log(err);
})
```

14.7 jscodeshift

- jscodeshift (https://www.npmjs.com/package/jscodeshift)是一个执行代码更改的工具包

```
let jscodeshift = require('jscodeshift');
const ast =  jscodeshift(`import ReactDOM from "react-dom"`);
console.log(ast.nodes());
console.log(ast.nodes()[0]);
console.log(ast.nodes()[0].program);
console.log(ast.nodes()[0].program.body[0]);
```

14.8 vue-codemod

- vue-codemod (https://www.npmjs.com/package/vue-codemod)包含了代码变量脚本的工具集

```
const { runTransformation } = require('vue-codemod')
let file = 'index.js';
let source = `
import React from 'react';
`;
let imports = ['import ReactDOM from "react-dom"'];
let transformed = runTransformation(
    { path: file, source },
    injectImports,
    { imports }
)
console.log(transformed);

function injectImports(fileInfo, api, { imports }) {
    const jscodeshift = api.jscodeshift
    const root = jscodeshift(fileInfo.source)
    const declarations = root.find(jscodeshift.ImportDeclaration)
    const toImportAST = imp => jscodeshift(`${imp}\n`).nodes()[0].program.body[0]
    const importASTNodes = imports.map(toImportAST);
    if (declarations.length) {
        declarations.at(-1).insertAfter(importASTNodes)
    } else {
        root.get().node.program.body.unshift(...importASTNodes)
    }
    return root.toSource()
}
```