如何开发 CLI

不局限于特定的业务场景，通用的 CLI 开发教程

项目地址：https://github.com/leer0911/cli

一、CLI 简介

CLI（Command Line Interface）命令行界面是在图形用户界面得到普及之前使用最为广泛的用户界面，它通常不支持鼠标，用户通过键盘输入指令，计算机接收到指令后，予以执行。也有人称之为字符用户界面（character user interface，CUI）。

社区流行的 CLI

- vue cli
- Create React App

二、Oclif 简介

在日常工作中，为了提高开发效率或统一开发方式，我们通常会开发团队内专属的 CLI 工具。这里介绍一种基于 Oclif 的方式。

Oclif 是由 Heroku（一个支持多种编程语言的云应用平台，在 2010 年被 Salesforce.com 收购）开发的 Node.js Open CLI 开发框架，它可以用来开发 **single-command CLI** 或 **multi-command CLI**，同时还提供了可扩展的插件机制和钩子机制。

2.1 CLI 类型

使用 Oclif 你可以创建两种不同类型的 CLI，即 **Single CLIs** 和 **Multi CLIs**。**Single CLIs** 类似于 Linux 或 MacOS 平台中常见的 `ls` 或 `cat` 命令。而 **Multi CLIs** 类似于前面提到的 Vue CLI，它们包含子命令，这些子命令本身也是 Single CLI。

2.2 快速开始

创建一个 **single-command CLI**：

Bash

```
npx oclif single mynewcli
```

创建一个 **multi-command CLI**：

Bash

```
npx oclif multi mynewcli
```

三、开发说明

> 注意：本内容基于 oclif 1.0.0

3.1 oclif 常用命令

- `oclif help` 查看帮助文档
- `oclif command NAME` 在 CLI 中新增命令
- `oclif hook NAME` 在 CLI 中新增钩子
- `oclif single [PATH]` 对应目录下生成 single-command CLI
- `oclif multi [PATH]` 对应目录下生成 multi-command CLI
- `oclif plugin [PATH]` 在 CLI 插件

3.2 oclif 常用 API

我们以 oclif 命令行生成 **multi-command CLI** （TypeScript 版本）为例，介绍相关 API。通过了解，你可以大概知道自己能做哪些事情，详情参阅 oclif API 文档。

一个基本命令脚本 API 说明如下（仅用于说明，非可运行脚本）：

TypeScript

```
import Command from "@oclif/command";

export class MyCommand extends Command {
  // 用于显示 CLI 中帮助文本的命令描述内容
  static description = "description of this example command";

  // 传递给命令的参数，如 mycli arg1 arg2 ( 跟位置有关系 )
```

```
  static args = [{ name: "firstArg" }, { name: "secondArg" }];

  // args 的可选参数说明
  static args = [
    {
      name: "file", // name of arg to show in help and reference with args[name]
      required: false, // make the arg required with `required: true`
      description: "output file", // help description
      hidden: true, // hide this arg from help
      parse: (input) => "output", // instead of the user input, return a different value
      default: "world", // default value if no arg input
      options: ["a", "b"], // only allow input to be from a discrete set
    },
  ];

  // 用于描述传递给命令的标识，分为可配置标识（必须传参，如：--file=./myFile）、布尔标识（true 或 false，如：--force）
  static flags = {
    // can pass either --force or -f
    force: flags.boolean({ char: "f" }),
    file: flags.string(),
  };

  // 详细配置
  static flags = {
    name: flags.string({
      char: "n", // shorter flag version
      description: "name to print", // help description for flag
      hidden: false, // hide from help
      multiple: false, // allow setting this flag multiple times
      env: "MY_NAME", // default to value of environment variable
      options: ["a", "b"], // only allow the value to be from a discrete set
      parse: (input) => "output", // instead of the user input, return a different value
      default: "world", // default value if flag not passed (can be a function that returns a string or undefined)
      required: false, // make flag required (this is not common and you should probably use an argument instead)
      dependsOn: ["extra-flag"], // this flag requires another flag
      exclusive: ["extra-flag"], // this flag cannot be specified alongside this other flag
    }),

    // flag with no value (-f, --force)
    force: flags.boolean({
      char: "f",
```

```
      char: "f",
      default: true, // default value if flag not passed (can be a function that returns a boolean)
      // boolean flags may be reversed with `--no-` (in this case: `--no-force`).
      // The flag will be set to false if reversed. This functionality
      // is disabled by default, to enable it:
      // allowNo: true
    }),
  };

  // 用于设置帮助文本中隐藏该命令
  static hidden = false;

  // 用于解释器在接收无效参数时是否失败，默认为 true （ 如果你需要很多参数，请设置为 false ）
  static strict = false;

  // 用于自定义帮助文本中命令的用法内容
  static usage = "mycommand --myflag";

  // 用于示例的描述内容
  static examples = ["$ mycommand --force", "$ mycommand --help"];

  // 别名
  static aliases = ["config:index", "config:list"];

  // run 方法是必须的，用于接收 arguments 和 flags
  async run() {
    // 以下为继承自 Command 父类的常用方法

    // 命令解析
    this.parse(MyCommand);

    // 获取 args 对象
    const { args } = this.parse(MyCLI);
    console.log(
      `running my command with args: ${args.firstArg}, ${args.secondArg}`
    );
    // 获取 argv 数组
    const { argv } = this.parse(MyCLI);
    console.log(`running my command with args: ${argv[0]}, ${argv[1]}`);

    // log 方法
    this.log("log");
    this.warn("warn");
    this.error("error");

    // 退出
```

```
    // 退出
    this.exit();
  }
}
```

3.3 钩子

生命周期事件：

- init 当 CLI 初始化完成，命令执行之前调用
- prerun 当命令执行前调用
- postrun 当命令成功被执行后调用
- command_not_found 当未找到命令并显示报错时调用

自定义事件：

通过调用 `this.config.runHook()` 来触发事件

3.4 本地开发

通过在项目根目录下载执行 `npm link` 来将包安装在全局。

`package.json` 中的 bin 的字段即为 CLI 名称

JSON

```
{
  "bin": {
    "cli": "./bin/run"
  }
}
```

如这里可以通过 `cli -h` 执行命令

3.5 交互式命令

TypeScript

```typescript
import { Command } from "@oclif/command";
import cli from "cli-ux";

export class MyCommand extends Command {
  async run() {
    // just prompt for input
    const name = await cli.prompt("What is your name?");

    // mask input after enter is pressed
    const secondFactor = await cli.prompt("What is your two-factor token?", {
      type: "mask",
    });

    // hide input while typing
    const password = await cli.prompt("What is your password?", {
      type: "hide",
    });

    this.log(`You entered: ${name}, ${secondFactor}, ${password}`);
  }
}
```

更为复杂的交互内容，推荐使用 inquirer

TypeScript

```typescript
import { Command, flags } from "@oclif/command";
import * as inquirer from "inquirer";

export class MyCommand extends Command {
  static flags = {
    stage: flags.string({ options: ["development", "staging", "production"] }),
  };

  async run() {
    const { flags } = this.parse(MyCommand);
    let stage = flags.stage;
    if (!stage) {
      let responses: any = await inquirer.prompt([
        {
          name: "stage",
          message: "select a stage",
          type: "list",
          choices: [
            { name: "development" },
            { name: "staging" },
            { name: "production" },
          ],
        },
      ]);
      stage = responses.stage;
    }
    this.log(`the stage is: ${stage}`);
  }
}
```

3.6 命令进度条

需要用到进度显示的 CLI 可以通过如下代码实现：

TypeScript

```typescript
import { Command } from "@oclif/command";
import cli from "cli-ux";

export class MyCommand extends Command {
  async run() {
    // start the spinner
    cli.action.start("starting a process");
    // do some action...
    // stop the spinner
    cli.action.stop(); // shows 'starting a process... done'

    // show on stdout instead of stderr
    cli.action.start("starting a process", "initializing", { stdout: true });
    // do some action...
    // stop the spinner with a custom message
    cli.action.stop("custom message"); // shows 'starting a process... custom message'
  }
}
```

更为复杂的进度条需求推荐使用 listr

3.7 通知

通过 node-notifier 实现跨平台通知：

TypeScript

```typescript
import { Command } from "@oclif/command";
import * as notifier from "node-notifier";

export class MyCommand extends Command {
  async run() {
    notifier.notify({
      title: "My notification",
      message: "Hello!",
    });
  }
}
```

实战

以 Vue-CLI 为例，一个脚手架大致的功能如下：

Bash

```
Usage: vue <command> [options]

Options:
  -V, --version                              output the version number
  -h, --help                                 output usage information

Commands:
  create [options] <app-name>                create a new project powered by vue-cli-service
  add [options] <plugin> [pluginOptions]     install a plugin and invoke its generator in an already created project
  invoke [options] <plugin> [pluginOptions]  invoke the generator of a plugin in an already created project
  inspect [options] [paths...]               inspect the webpack config in a project with vue-cli-service
  serve [options] [entry]                    serve a .js or .vue file in development mode with zero config
leedeMacBook:cli lee$ vue -h
Usage: vue <command> [options]

Options:
  -V, --version                              output the version number
  -h, --help                                 output usage information

Commands:
  create [options] <app-name>                create a new project powered by vue-cli-service
  add [options] <plugin> [pluginOptions]     install a plugin and invoke its generator in an already created project
  invoke [options] <plugin> [pluginOptions]  invoke the generator of a plugin in an already created project
  inspect [options] [paths...]               inspect the webpack config in a project with vue-cli-service
  serve [options] [entry]                    serve a .js or .vue file in development mode with zero config
  build [options] [entry]                    build a .js or .vue file in production mode with zero config
  ui [options]                               start and open the vue-cli ui
  init [options] <template> <app-name>       generate a project from a remote template (legacy API, requires @vue/cli-init)
  config [options] [value]                   inspect and modify the config
  outdated [options]                         (experimental) check for outdated
```

```
vue cli service / plugins
  upgrade [options] [plugin-name]            (experimental) upgrade vue cli service / plugins
  migrate [options] [plugin-name]            (experimental) run migrator for an already-installed cli plugin
  info                                       print debugging information about your environment

  Run vue <command> --help for detailed usage of given command.
```

TODO：结合公司业务场景，输出一个适合公司业务的脚手架。