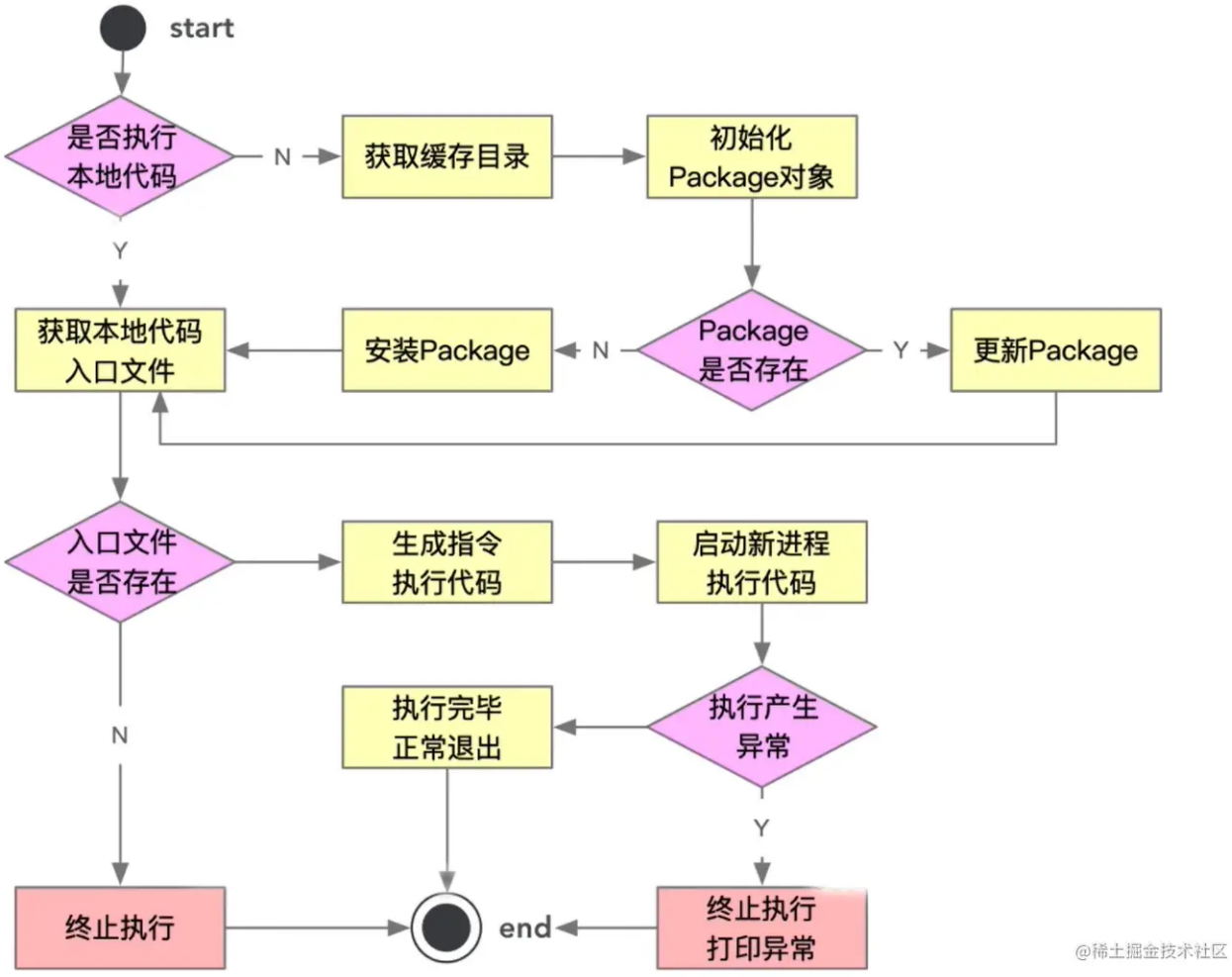
【架构师（第十一篇）】脚手架之命令注册和执行过程开发

一尾流莺 2022-04-08 08:47 1616

关注

脚手架命令动态加载功能架构设计图



是否执行本地代码

设置全局的 `targetPath`

js 复制代码

```
    process.env.CLI_TARGET_PATH = params.targetPath;
});
```

新建一个 `exec` 包，动态加载命令

```
lerna create @hzw-cli-dev/exec
```

新建一个 `package` 包，放在 `models` 目录下面

```
lerna create @hzw-cli-dev/package
```

新建一个 `command` 包，放在 `models` 目录下面

```
lerna create @hzw-cli-dev/command
```

npminstall 用法

```
const npmInstall = require('npminstall');
const path = require('path');
const userHome = require('user-home');

// 直接调用安装方法
npmInstall({
  root: path.resolve(userHome, '.hzw-cli-dev'), // 模块路径
  storeDir: path.resolve(userHome, '', 'node_modules'), // 模块安装位置
  register: 'https://registry.npmjs.org', // 设置 npm 源
  pkgs: [ // 要安装的包信息
    {
      name: 'warbler-js',
      version: '',
    },
  ],
});
```

js 复制代码

```
'use strict';

const { isObject } = require('@hzw-cli-dev/utils');
const { getRegister, getLatestVersion } = require('@hzw-cli-dev/get-npm-info');
const formatPath = require('@hzw-cli-dev/format-path');
const npmInstall = require('npminstall');
const fse = require('fs-extra');
const pathExists = require('path-exists').sync;
const pkgDir = require('pkg-dir').sync;
const path = require('path');
// Package 类 管理模块
class Package {
  /**
   * @description: 构造函数
   * @param {*} options 用户传入的配置信息
   * @return {*}
   */
  constructor(options) {
    if (!options) {
      throw new Error('Package 类的参数不能为空!');
    }
    if (!isObject(options)) {
      throw new Error('Package 类的参数必须是对象类型!');
    }
    // 获取 targetPath ,如果没有 则说明不是一个本地的package
    this.targetPath = options.targetPath;
    // 模块安装位置 缓存路径
    this.storeDir = options.storeDir;
    // package 的 name
    this.packageName = options.packageName;
    // package 的 Version
    this.packageVersion = options.packageVersion;
    // 缓存路径的前缀
    this.cacheFilePathPrefix = this.packageName.replace('/', '_');
  }

  /**
   * @description: 准备工作
   * @param {*}
   * @return {*}
   */
  async prepare() {}

```

```
49    get cacheFilePath() {}
50
51    /**
52     * @description: 获取最新版本模块缓存路径
53     * @param {*}
54     * @return {*}
55     */
56    getSpecificFilePath(packageVersion) {}
57
58    /**
59     * @description: 判断当前 package 是否存在
60     * @param {*}
61     * @return {*}
62     */
63    async exists() {}
64
65    /**
66     * @description: 安装 package
67     * @param {*}
68     * @return {*}
69     */
70    async install() {}
71
72    /**
73     * @description: 更新 package
74     * @param {*}
75     * @return {*}
76     */
77    async update() {}
78
79    /**
80     * @description:获取入口文件的路径
81     * 1.获取package.json所在的目录 pkg-dir
82     * 2.读取package.json
83     * 3.找到main或者lib属性 形成路径
84     * 4.路径的兼容(macOs/windows)
85     * @param {*}
86     * @return {*}
87     */
88    getRootFilePath() {}
89  module.exports = Package;
```

```js
/**
 * @description: 准备工作
 * @param {*}
 * @return {*}
 */
async prepare() {
  // 当缓存目录不存在的时候
  if (this.storeDir && !pathExists(this.storeDir)) {
    // 创建缓存目录
    fse.mkdirpSync(this.storeDir);
  }
  // 获取最新版本
  const latestVersion = await getLatestVersion(this.packageName);
  // 如果设定的版本号是最新的话，就赋值
  if (this.packageVersion === 'latest') {
    this.packageVersion = latestVersion;
  }
}
```

获取当前模块缓存路径

```js
/**
 * @description: 获取当前模块缓存路径
 * @param {*}
 * @return {*}
 */
get cacheFilePath() {
  return path.resolve(
    this.storeDir,
    `_${this.cacheFilePathPrefix}@${this.packageVersion}@${this.packageName}`,
  );
}
```


获取最新版本模块缓存路径

```
/**
 * @description: 获取最新版本模块缓存路径
 * @param {*}
 * @return {*}
 */
getSpecificFilePath(packageVersion) {
  return path.resolve(
    this.storeDir,
    `_${this.cacheFilePathPrefix}@${packageVersion}@${this.packageName}`,
  );
}
```

判断当前 package 是否存在

js 复制代码

```
/**
 * @description: 判断当前 package 是否存在
 * @param {*}
 * @return {*}
 */
async exists() {
  // 如果 this.storeDir 存在 ,就是需要下载安装,否则就是本地安装
  if (this.storeDir) {
    // 获取具体版本号
    await this.prepare();
    return pathExists(this.cacheFilePath);
  } else {
    // 查看本地路径是否存在
    return pathExists(this.targetPath);
  }
}
```

安装 package

js 复制代码

```
/**
 * @description: 安装 package
```

```
    await this.prepare();
    return npmInstall({
      root: this.targetPath, // 模块路径
      storeDir: this.storeDir, // 模块安装位置
      register: getRegister('npm'), // 设置 npm 源
      pkgs: [
        // 要安装的包信息
        {
          name: this.packageName,
          version: this.packageVersion,
        },
      ],
    });
  }
```

更新 package

js 复制代码

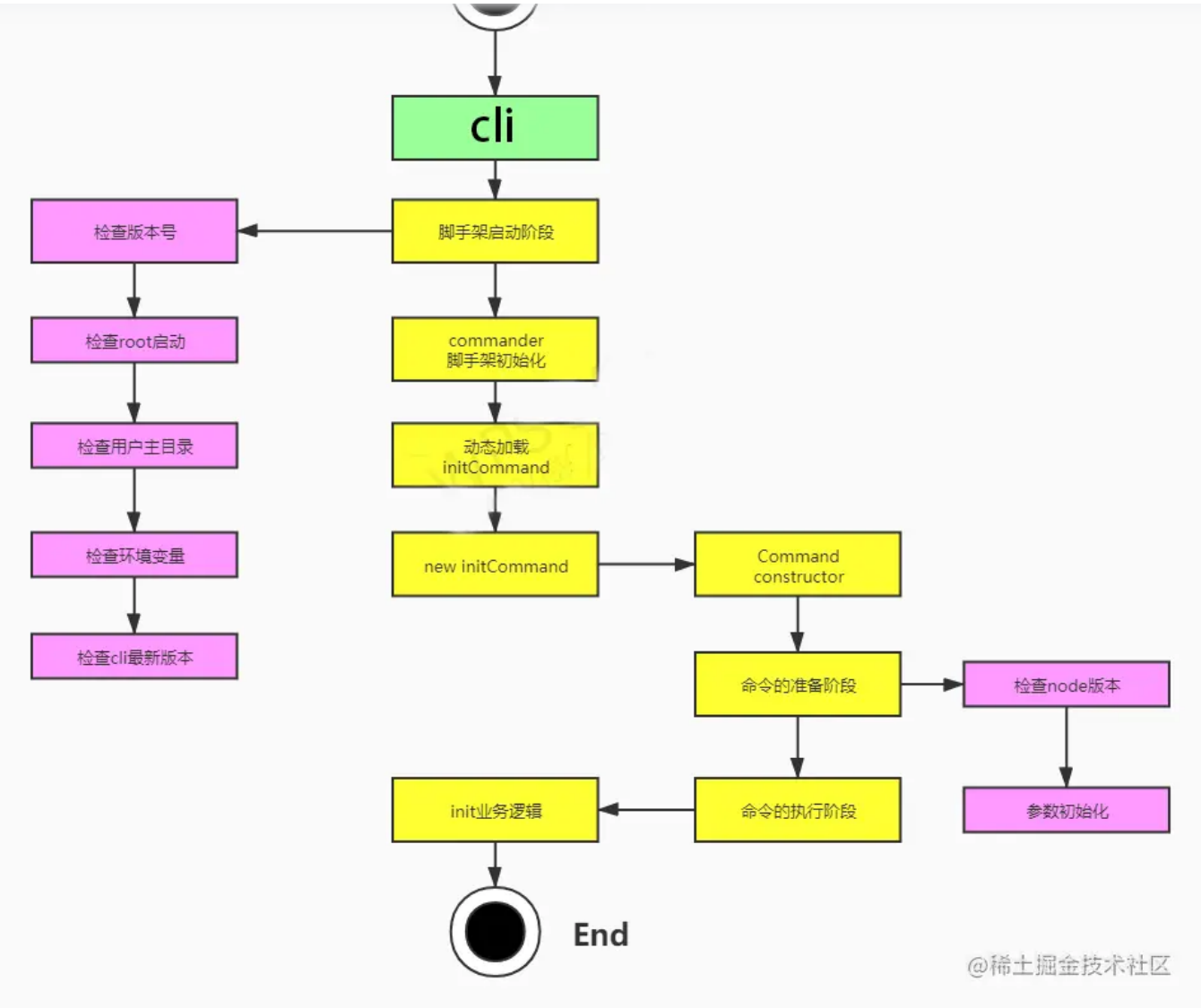
```
/**
 * @description: 更新 package
 * @param {*}
 * @return {*}
 */
async update() {
  await this.prepare();
  // 获取最新版本号
  const latestVersion = await getLatestVersion(this.packageName);
  // 查询本地是否已经是最新版本
  const localPath = this.getSpecificFilePath(latestVersion);
  const isLocalLatestVersion = pathExists(localPath);
  console.log('🚀🚀 ~ Package ~ latestVersion', latestVersion);
  console.log('🚀🚀 ~ Package ~ localPath', localPath);
  console.log('🚀🚀 ~ Package ~ isLocalLatestVersion', isLocalLatestVersion);
  // 如果不是最新版本 安装最新版本
  if (!isLocalLatestVersion) {
    await npmInstall({
      root: this.targetPath, // 模块路径
      storeDir: this.storeDir, // 模块安装位置
      register: getRegister('npm'), // 设置 npm 源
      pkgs: [
        // 要安装的包信息
```

```
      });
      this.packageVersion = latestVersion;
    }
  }
```

获取入口文件的路径

js 复制代码

```
/**
 * @description:获取入口文件的路径
 * 1.获取package.json所在的目录 pkg-dir
 * 2.读取package.json
 * 3.找到main或者lib属性 形成路径
 * 4.路径的兼容(macOs/windows)
 * @param {*}
 * @return {*}
 */
getRootFilePath() {
  function _getRootFIle(_path) {
    const dir = pkgDir(_path);
    if (dir) {
      const pkgFile = require(path.resolve(dir, 'package.json'));
      if (pkgFile && (pkgFile.main || pkgFile.lib)) {
        const rootPath =
          formatPath(path.resolve(dir, pkgFile.main)) ||
          formatPath(path.resolve(dir, pkgFile.lib));
        return rootPath;
      }
    }
    return null;
  }
  // 如果 this.storeDir 存在 ,就是需要下载安装,否则就是本地安装
  if (this.storeDir) {
    return _getRootFIle(this.cacheFilePath);
  }
  return _getRootFIle(this.targetPath);
}
```



Command 类的实现

js 复制代码

```
'use strict';

// 引入版本比对第三方库 semver
const semver = require('semver');
const LOWEST_NODE_VERSION = '12.0.0';
// 引入颜色库 colors
const colors = require('colors/safe');
const log = require('@hzw-cli-dev/log');
const { isArray } = require('@hzw-cli-dev/utils');
class Command {
  /**
   * @description: 命令的准备和执行阶段
```

```
  constructor(argv) {
    if (!argv) {
      throw new Error('参数不可以为空!');
    }
    if (!isArray(argv)) {
      throw new Error('参数必须是数组类型!');
    }
    if (argv.length < 1) {
      throw new Error('参数列表不可以为空!');
    }
    this._argv = argv;
    let runner = new Promise((resolve, reject) => {
      let chain = Promise.resolve();
      chain = chain.then(() => this.checkNodeVersion());
      chain = chain.then(() => this.initArgs());
      chain = chain.then(() => this.init());
      chain = chain.then(() => this.exec());
      chain.catch((error) => log.error(error.message));
    });
  }

  /**
   * @description: 检查当前的 node 版本,防止 node 版本过低调用最新 api 出错
   * @param {*}
   * @return {*}
   */
  checkNodeVersion() {
    // 获取当前 node 版本号
    const currentVersion = process.version;
    log.test('环境检查 当前的node版本是:', process.version);
    // 获取最低 node 版本号
    const lowestVersion = LOWEST_NODE_VERSION;
    // 对比最低 node 版本号
    if (!semver.gte(currentVersion, lowestVersion)) {
      throw new Error(colors.red('错误:node版本过低'));
    }
  }

  /**
   * @description:初始化参数
   * @param {*}
   * @return {*}
   */
  initArgs() {
    this._cmd = this._argv[this._argv.
    this._argv = this._argv.slice(0, this._argv.length - 1);
```

```
66    }
67
68    exec() {
69      throw new Error('command 必须拥有一个 exec 方法');
70    }
71  }
72
73  module.exports = Command;
74
```

> 注：第四周 第五章全部以及第七章全部 没看懂，后面需要再复习一下，但是不影响主课程的进行

exec 的实现

js 复制代码

```
1   'use strict';
2
3   const Package = require('@hzw-cli-dev/package');
4   const log = require('@hzw-cli-dev/log');
5   const path = require('path');
6
7   const cp = require('child_process');
8
9   // package 的映射表
10  const SETTINGS = {
11    // init: '@hzw-cli-dev/init',
12
13    init: '@imooc-cli/init',
14  };
15
16  // 缓存目录
17  const CACHE_DIR = 'dependencies';
18
19  /**
20   * @description: 动态加载命令
21   * 1.获取 targetPath
22   * 2.获取 modulePath
```

```
async function exec(...argv) {
  // 获取 targetPath
  let targetPath = process.env.CLI_TARGET_PATH;
  // 获取 homePath
  const homePath = process.env.CLI_HOME_PATH;
  log.verbose('targetPath', targetPath);
  log.verbose('homePath', homePath);
  // 获取 commander 实例
  const cmdObj = argv[argv.length - 1];
  // 获取命令名称 exp:init
  const cmdName = cmdObj.name();
  // 获取 package 的 name  exp:根据 init 命令 映射到 @hzw-cli-dev/init 包
  const packageName = SETTINGS[cmdName];
  // 获取 package 的 version
  const packageVersion = 'latest';
  // 模块安装路径
  let storeDir = '';
  // package 类
  let pkg = '';

  // 如果  targetPath 不存在
  if (!targetPath) {
    targetPath = path.resolve(homePath, CACHE_DIR); // 生成缓存路径
    storeDir = path.resolve(targetPath, 'node_modules');
    // 创建 Package 对象
    pkg = new Package({
      storeDir,
      targetPath,
      packageName,
      packageVersion,
    });
    // 如果当前 package 存在
    if (await pkg.exists()) {
      // 更新 package
      pkg.update();
    } else {
      // 安装 package
      await pkg.install();
    }
  } else {
    // 如果 targetPath 存在
    pkg = new Package({
      targetPath,
      packageName,
      packageVersion
```

```
  // 获取模块入口
  const rootFile = pkg.getRootFilePath();
  if (!rootFile) {
    throw new Error('模块不存在入口文件!');
  }
  // 执行模块
  try {
    // 在当前进程中调用
    // rootFile && require(rootFile)(argv);
    // 在 node 子进程中调用
    const cmd = argv[argv.length - 1];
    const newCmd = Object.create(null);
    // 给参数瘦身
    Object.keys(cmd).forEach((key) => {
      if (cmd.hasOwnProperty(key) && !key.startsWith('_') && key !== 'parent') {
        newCmd[key] = cmd[key];
      }
    });
    argv[argv.length - 1] = newCmd;
    // 组合code
    const code = `require('${rootFile}')(${JSON.stringify(argv)})`;
    const childProcess = spawn('node', ['-e', code], {
      cwd: process.cwd(),
      stdio: 'inherit',
    });
    childProcess.on('error', (error) => {
      log.error(error.message);
      process.exit(1);
    });
    childProcess.on('exit', (e) => {
      log.verbose('命令执行成功');
      process.exit(e);
    });
  } catch (error) {
    console.log(error.message);
  }
}

/**
 * @description: 封装一个 spawn 方法,兼容 mac 和 windows
 * windows : cp.spawn('cmd',['/c','node','-e',code],{})
 * mac : cp.spawn('node', ['-e', code],{})
 * @param {*} command 'cmd'
 * @param {*} args ['/c','node','-e',c
 * @param {*} options {}
 * @return {*} cp.spawn(cmd, cmdArgs, options)
```

```
126 var platform = process.platform;
127 switch (platform) {
128   case 'aix':
129     console.log('IBM AIX platform');
130     break;
131   case 'darwin':
132     console.log('Darwin platform(MacOS, IOS etc)');
133     break;
134   case 'freebsd':
135     console.log('FreeBSD Platform');
136     break;
137   case 'linux':
138     console.log('Linux Platform');
139     break;
140   case 'openbsd':
141     console.log('OpenBSD platform');
142     break;
143   case 'sunos':
144     console.log('SunOS platform');
145     break;
146   case 'win32':
147     console.log('windows platform');
148     break;
149   default:
150     console.log('unknown platform');
151 }
152  */
153
154 function spawn(command, args, options = {}) {
155   const win32 = process.platform === 'win32';
156   const cmd = win32 ? 'cmd' : command;
157   const cmdArgs = win32 ? ['/c'].concat(command, args) : args;
158   return cp.spawn(cmd, cmdArgs, options);
159 }
160
161 module.exports = exec;
162
```

标签: 前端 架构 前端框架

文章被收录于专栏:

相关小册

趣学 Node.js

死月 LV.3

2611购买

¥49.9

前端开发者的现代 C++ 课

刘晓伦liulun

853购买

¥39.9

评论

输入评论（Enter换行，Ctrl + Enter发送）

全部评论 8

最新 最热

feng_cc LV.3 JY.5 7月前

课程是先讲了package类，但我觉得，先完成execfunction再写package类比较合理，一开始就扣细节，很难把代码写下去的

点赞 1

一尾流莺 7月前

还行吧我觉得，脚手架这块讲的挺清楚的，到后面开发页面那块我有点看不下去了，好多都跳过了。

点赞 回复

wgf4242 JY.4 1年前

processon

点赞 回复

buchiyu LV.2 JY.5 1年前

我因为代码无意间删掉了 我又好好看了一下这个package类的代码 感觉有一些问题，你看看是不是，老师在exist的代码里写了 if (this.packageVersion === 'latest') { this.packageVersion = await getNpmLatestVersion(this.packageName) }。这样的话，举个例子，我latestVersion=1.2.1，现在本地storeDir里面是1.2.0，那我按照老师的代码走的话，当调用exist的时候就已经更新的最新版本，那么这个时候，pathExists(this.cacheFilePath)肯定是不存在的，那这样就会走到install，而非更新，我想了想，...

点赞 回复

郭二蛋 LV.3 JY.4 1年前

不错，当初买课的时候没做笔记，现在看看您的笔记😁辛苦啦

点赞 2

一尾流莺 1年前

我看老师做的那个记笔记的网站上有个郭二蛋，可是你本人啊，哈哈哈，有不少的地方我还参考了里面同学的笔记，有的流程图我直接截的图，哈哈哈哈😡😡😡

点赞 回复

郭二蛋 1年前

是本蛋没错啦，哈哈

> “我看老师做的那个记笔记的网站上有个郭二蛋，可是你本人啊，哈哈哈，有不少的... ”

点赞 回复

相关推荐

一尾流莺 1年前

【🐯初/中级前端面经】中小型公司面试 APP内打开 么?

11.5w 2201 206

12 8 收藏

一尾流莺 1年前

【今天你更博学了么】一个神奇的前端动画 API requestAnimationFrame

1.8w 370 42

一尾流莺 1年前

【面试题解】CSS盒子模型与margin负值

1.4w 109 8

一尾流莺 1年前

【图文并茂】六十多个 vscode 插件，助你打造最强编辑器

8650 227 36

一尾流莺 1年前

【面试题解】你了解JavaScript常用的的十个高阶函数么?

1.1w 101 14

一尾流莺 2年前

【流莺书签】Vue3+TS的收藏网址小项目

7653 124 12

一尾流莺 1年前

【面试题解】JavaScript数据类型相关的六个面试题

8618 38 4

一尾流莺 1年前

【面试题解】vue-router有几种钩子函数？具体是什么及执行流程是怎样的?

6594 122 10

一尾流莺 1年前

【面试题解】你了解call，apply，bind吗？那你可以手写一个吗?

7305 75 14

一尾流莺 1年前

【网页游戏】Vue3 + Typescript 自己动手实现一个贪吃蛇

6226 95 14

一尾流莺 1年前

【初学者笔记】前端工程化必须要掌握的 webpack

3440 126 25

一尾流莺 1年前

【面试题解】谈一谈JavaScript数据类型判断

5713 34 1

一尾流莺 2年前

【实战技巧】VUE3.0实现简易的可拖放列表排序

4614 78 3

一尾流莺 2年前

【年中总结】还是要继续努力呀 | 2021 年中总结

2963 104 47

一尾流莺 2年前

【初学者笔记】一文学会使用Vuex

3632 109 2

一尾流莺 1年前

【面试题解】初识 JavaScript 闭包

4033 47 41

一尾流莺 1年前

【架构师（第七篇）】脚手架之准备阶段编写

3549 38 21

一尾流莺 1年前

【源码学习】Vue 初始化过程 (附思维导图)

3006 79 6