

第 1 章 基本认识

1-1 薪资基本认识

北上广深城市：对于技术不同程度的掌握，可以拿到的薪资各有千秋：

- ①Html5+Css3+Jquery：编写 PC 端与移动端静态页面 6K-8K。
- ②假若可以掌握 Angular、Vue、React 任意一个框架，薪资可达 10K+。
- ③再掌握 Nodejs，薪资可达 13K+。
- ④继续掌握 Ionic、ReactNative，进行混合 APP 开发，薪资可达 15K-20K。

1-2 单页面应用开发框架认识

单页面应用开发框架主要分三大类：Vue、React、Angular。它们的特点是：

- ①单页面框架
- ②基于模块化组件化的开发模式

1-3 Vue 教程地址：<https://cn.vuejs.org/v2/guide/>

1-4 Git 2.21.0 版本安装及配置

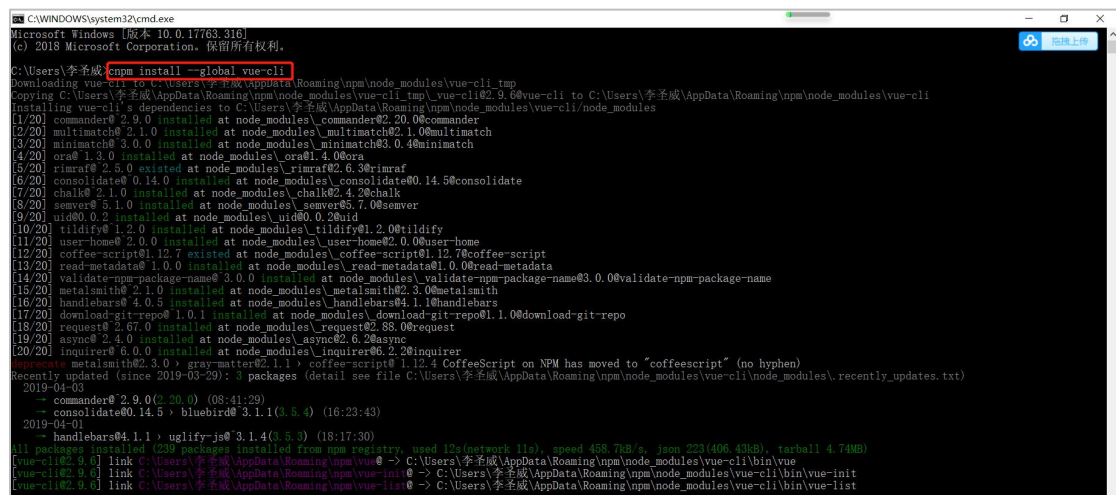
<http://www.mamicode.com/info-detail-2620338.html>

第2章 搭建Vue的开发环境

2-1 安装Nodejs运行时环境

2-2 搭建vue的开发环境

- ① 教程地址: <https://cn.vuejs.org/v2/guide/installation.html>
- ② 淘宝 npm 镜像地址: <http://npm.taobao.org/>
- ③ cmd 执行命令: `npm install -g cnpm --registry=https://registry.npm.taobao.org` 安装淘宝镜像。
- ④ cmd 执行命令: `npm /cnpm install --global vue-cli` 安装 vue 的脚手架工具。



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.17763.316]
(c) 2018 Microsoft Corporation. 保留所有权利。

C:\Users\李圣威>cnpm install --global vue-cli
Downloading vue-cli to C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\tmp
Copying C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\tmp\vue-cli@2.9.6\vue-cli to C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli
Installing vue-cli's dependencies to C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\node_modules
(1/20) commander@2.9.0 installed at node_modules\commander@2.20.0@commander
(2/20) multimatch@2.1.0 installed at node_modules\multimatch@2.1.0@multimatch
(3/20) minimatch@3.0.0 installed at node_modules\minimatch@3.0.4@minimatch
(4/20) ora@1.3.0 installed at node_modules\ora@1.4.0@ora
(5/20) rimraf@2.5.0 existed at node_modules\rimraf@2.6.3@rimraf
(6/20) consolidate@0.14.0 installed at node_modules\consolidate@0.14.5@consolidate
(7/20) chalk@2.1.0 installed at node_modules\chalk@2.4.2@chalk
(8/20) semver@5.1.0 installed at node_modules\semver@5.7.0@semver
(9/20) uid@0.0.2 installed at node_modules\uid@0.0.2@uid
(10/20) tildify@1.2.0 installed at node_modules\tildify@1.2.0@tildify
(11/20) user-home@2.0.0 installed at node_modules\user-home@2.0.0@user-home
(12/20) coffee-script@1.12.7 existed at node_modules\coffee-script@1.12.7@coffee-script
(13/20) read-metadata@1.0.0 installed at node_modules\read-metadata@1.0.0@read-metadata
(14/20) validate-npm-package-name@3.0.0 installed at node_modules\validate-npm-package-name@3.0.0@validate-npm-package-name
(15/20) metalsmith@2.1.0 installed at node_modules\metalsmith@2.3.0@metalsmith
(16/20) handlebars@4.0.5 installed at node_modules\handlebars@4.1.1@handlebars
(17/20) download-git-repo@1.0.1 installed at node_modules\download-git-repo@1.1.0@download-git-repo
(18/20) request@2.67.0 installed at node_modules\request@2.88.0@request
(19/20) async@2.4.0 installed at node_modules\async@2.6.2@async
(20/20) inquirer@6.0.0 installed at node_modules\inquirer@6.2.2@inquirer
- metalsmith@2.3.0 > gray-matter@2.1.1 > coffee-script@1.12.7 CoffeeScript on NPM has moved to "coffeescript" (no hyphen)
Recently updated (since 2019-03-29): 3 packages (detail see file C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\node_modules\recently_updates.txt)
2019-04-03
  - commander@2.9.0(C: 20.0) (08:41:29)
  - consolidate@0.14.5 > bluebird@3.1.1(3.5.4) (16:23:43)
2019-04-01
  - handlebars@4.1.1 > uglify-js@3.1.4(3.5.3) (18:17:30)
All packages installed (239 packages installed from npm registry, used 18.7MB in 1.5s, 222/406 files, 1mball 4.74MB)
[ue-cli@2.9.6] link C:\Users\李圣威\AppData\Roaming\npm\vue@ -> C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\bin\vue
[ue-cli@2.9.6] link C:\Users\李圣威\AppData\Roaming\npm\vue-init@ -> C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\bin\vue-init
[ue-cli@2.9.6] link C:\Users\李圣威\AppData\Roaming\npm\vue-list@ -> C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\bin\vue-list
```

2-3 创建项目

- ① 第一种方法: 新建一个文件夹, 用以保存项目, `cd` 到文件夹路径下, cmd 执行命令: `vue init webpack vue-demo01` 等待项目创建成功。? Use ESLint to lint your code? **No** 可以避免语法检查。

```
C:\WINDOWS\system32\cmd.exe
C:\Users\李圣威> link C:\Users\李圣威\AppData\Roaming\npm\vue-init -> C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\bin\vue-init
C:\Users\李圣威> link C:\Users\李圣威\AppData\Roaming\npm\vue-list -> C:\Users\李圣威\AppData\Roaming\npm\node_modules\vue-cli\bin\vue-list
C:\Users\李圣威> H:
H:\> cd H:\Vue
H:\Vue> vue init webpack vue-demo01
git?
Project name: vue-demo01
Project description: A Vue.js project
Author:
Vue build standalone? [y]
Install vue-router? [n]
Use ESLint to lint your code? [n]
Set up unit tests? [n]
Pick a test runner [jest]
Setup e2e tests with Nightwatch? [n]
Should we run npm install for you after the project has been created? (recommended) [n]
vue-cli - Generated "vue-demo01".

Installing project dependencies...
npm WARN deprecated browserslist@2.11.3: Browserslist 2 could fail on reading Browserslist >3.0 config used in other tools.
npm WARN deprecated bfj-node4@5.3.1: Switch to the 'bfj' package for fixes and new features!
npm WARN deprecated browserslist@1.7.7: Browserslist 2 could fail on reading Browserslist >3.0 config used in other tools.
npm WARN deprecated socks@1.1.10: If using 2.x branch, please upgrade to at least 2.1.6 to avoid a serious bug with socket data flow and an import issue introduced in 2.1.0
chromedriver@2.46.0 install H:\Vue\vue-demo01\node_modules\chromedriver
node install.js
Current existing ChromeDriver binary is unavailable, proceeding with download and extraction.
Downloading from file: https://chromedriver.storage.googleapis.com/2.46/chromedriver-win32.zip
Saving to file: C:\Users\李圣威\AppData\Local\Temp\2\46\chromedriver\chromedriver-win32.zip
Chromedriver installation failed with http(s) request: Error: connect ETIMEDOUT 172.161.178.443
npm WARN SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN code ELI5CYCLE
npm WARN chromedriver@2.46.0 install: 'node install.js'
npm WARN Exit status 1
npm WARN Failed at the chromedriver@2.46.0 install script.
npm WARN This is probably not a problem with npm. There is likely additional logging output above.
npm WARN A complete log of this run can be found in:
C:\Users\李圣威\AppData\Roaming\npm-cache\_logs\2019-04-05T07_52_32_5902-debug.log
npm WARN
```

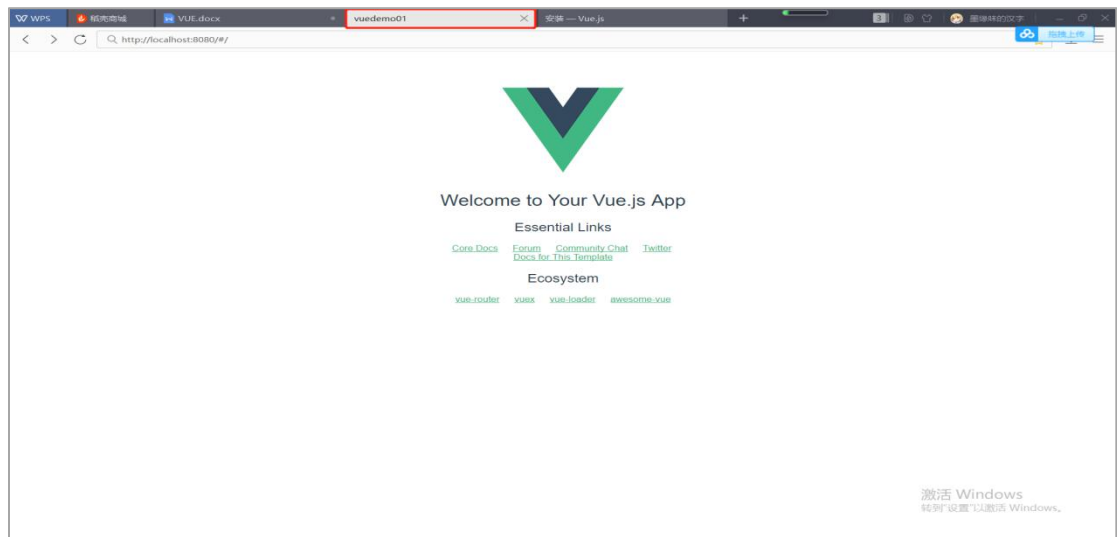
推荐使用第二种方法创建项目：cmd 执行命令：vue init **webpack-simple** vue-demo01，可以直接避免语法检查。

② cmd 执行命令：cd vue-demo01 进入到项目里面。假若项目创建有报错“WARN”，需要执行 npm/cnpm install 命令安装项目依赖包和模块，否则可以省略。

③ 运行创建的项目，cmd 执行命令：cnpm run dev，在网站地址栏输入：<http://localhost:8080>，即可成功轻松地查看项目了。

```
npm
npm WARN chromedriver@2.46.0 install: 'node install.js'
npm WARN Exit status 1
npm WARN Failed at the chromedriver@2.46.0 install script.
npm WARN This is probably not a problem with npm. There is likely additional logging output above.
npm WARN A complete log of this run can be found in:
C:\Users\李圣威\AppData\Roaming\npm-cache\_logs\2019-04-05T07_52_32_5902-debug.log
# Project initialization finished:
=====
To get started:
  cd vue-demo01
  npm run dev
Documentation can be found at https://vuejs-templates.github.io/webpack

H:\Vue>
H:\Vue> cd: vue-demo01
文件名、目录名或卷标语法不正确。
H:\Vue> cd vue-demo01
H:\Vue\vue-demo01> cnpm run dev
> vue-demo01@1.0.0 dev H:\Vue\vue-demo01
> webpack-dev-server --inline --progress --config build/webpack.dev.conf.js
13% building modules 27/31 modules 4 active ...&index=0H:\Vue\vue-demo01\src\App.vue[ parser: "babylon" ] is deprecated; we now treat it as [ parser: "babel" ].
95% emitting
Compiled successfully in 4477ms
Your application is running here: http://localhost:8080
```



第 3 章 模板语法

```
<script>
  export default {
    data () { /*业务逻辑里面定义的数据*/
      return {
        msg: '你好vue',
        title: '我是一个title',
        url: 'https://www.itying.com/themes/itying/images/logo.gif',
        h: '<h2>我是h2</h2>',
        list: ['1111', '2222', '3333'],
        flag: false,
        boxWdith: 500
      }
    }
  }
</script>
```

3-1 Vue 常用指令

指令 (Directives) 是带有 v- 前缀的特殊特性。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

3.1.1 Mustache 语法：绑定数据 - 文本插值

数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值：

```
<span>Message: {{ msg }}</span>
```

HTML

Mustache 标签将会被替代为对应数据对象上 msg 属性的值。无论何时，绑定的数据对象上 msg 属性发生了改变，插值处的内容都会更新。

3.1.2 v-text 指令：绑定数据 - 巧借指令

```
<div v-text="msg"></div>
```

3.1.3 v-once 指令：一次性插值

当数据改变时，插值处的内容不会更新。但请留心这会影响到该节点上的其它数据绑定。

```
<span v-once>这个将不会改变: {{ msg }}</span>
```

HTML

3.1.4 v-bind 指令：绑定属性

v-bind 指令可以用于响应式地更新 HTML 特性：如 href、src、title、class、id、disabled、style 等。

```
<div class="box" v-bind:style="{ 'width': boxWidth + 'px' }">宽度div</div>
```

```
<div :class="{ 'red': flag, 'blue': !flag }">blue</div>
```

```
<a v-bind:href="url">...</a>
```

HTML

3.1.5 v-html 指令：绑定并解析 html

Mustache 语法：双大括号 {{ }} 会将数据解释为普通文本，而非 HTML 代码。为了输出真正的 HTML，你需要使用 v-html 指令。

```
<p>Using mustaches: {{ rawHtml }}</p>
```

```
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

HTML

3.1.6 v-on 指令：监听 DOM 事件

v-on 指令，它用于监听 DOM 事件。在这里参数是监听的事件名。

其中，**v-on:click** 可以缩写成 **@click**。

```
<a v-on:click="doSomething">...</a>
```

HTML

3.1.7 v-for 指令：列表渲染

最基本的数据列表循环渲染。

```
<ul>
  <li v-for="(item, key) in list">{{key}}---{{item}}</li>
</ul>
```

其中，key：为索引：0,1,2,，此处指定索引为 0 和 1 的 item 的字体颜色为红色和蓝色。

```
<ul>
  <li v-for="(item, key) in list" :class="{ 'red': key==0, 'blue': key==1 }">{{key}}---{{item}}</li>
</ul>
```

- 0---1111
- 1---2222
- 2---3333

3.1.8 v-if 指令：条件判断

.....

3-2

第 4 章 MVVM 双向数据绑定

4-1 对 MVVM 的初步认识

MVVM：意即双向数据绑定，vue 其实是一个 MVVM 框架。

M：即 Model，是 v-model 绑定的数据。V：即 View，是数据渲染后的视图。

简单理解：model 改变会影响视图 view，view 视图改变反过来影响 model，且双向数据绑定必须在表单里面使用。

4-2 数据双向绑定（MVVM）示例

执行 setMsg（）方法后，被绑定的数据更新，视图显示的数据也将改变。

在表单中输入“你好，中国”后，视图改变，被绑定的数据也将被更新。

```
<h2>{{msg}}</h2>
<input type="text" v-model='msg' />
<button v-on:click="getMsg()">获取表单里面的数据get</button>
<button v-on:click="setMsg()">设置表单的数据set</button>
```

你好vue

你好vue

获取表单里面的数据get

设置表单的数据set

```
36 <script>
37   export default {
38     data () { /*业务逻辑里面定义的数据*/
39       return {
40         msg: '你好vue'
41       }
42     },
43     methods: { /*放方法的地方*/
44       getMsg(){
45         // alert('vue方法执行了');
46         //方法里面获取data里面的数据
47         alert(this.msg);
48       },
49       setMsg(){
50         this.msg="我是改变后的数据";
51       },
52       getInputValue(){
53         //获取ref定义的dom节点
54         console.log(this.$refs.userinfo);
55         this.$refs.box.style.background='red';
56         alert(this.$refs.userinfo.value);
57       }
58     }
59   }
60 </script>
```


第 5 章 ref 获取 DOM 节点及其 value

5-1 获取 ref 定义的 DOM 节点

其中：ref 定义的 DOM 节点 userinfo 即是：<input type="text"/>

```
<input type="text" ref="userinfo" /><br> <br>
<div ref="box">我是一个box</div><br><br>
<button v-on:click="getInputValue()">获取第二个表单里面的数据</button>
```

在业务逻辑中，通过 this.\$ref.userinfo 获取 ref 定义的 DOM 节点：<input type="text"/>。

通过 this.\$refs.userinfo.value 获取 DOM 节点下表单域输入的 value 值。

```
28 <script>
29   export default {
30     data () { /*业务逻辑里面定义的数据*/
31       return {
32         msg: '你好vue'
33       }
34     },
35     methods:{ /*放方法的地方*/
36       getInputValue(){
37         //获取ref定义的dom节点
38         console.log(this.$refs.userinfo);
39         this.$refs.box.style.background='red';
40         alert(this.$refs.userinfo.value);
41       }
42     }
43   }
44 </script>
```



第 6 章 方法

6-1 绑定点击事件的两种写法

6.1.1 第一种（ES6 语法）：方法名（）{ }

```
<button v-on:click="run1()">执行方法的第一种写法</button>
```

```
methods:{
  run1:function(){
    alert('执行方法的第一种写法:run1:function( ) { }');
  }
}
```

6.1.2 第二种：方法名：function（）{ }

```
<button @click="run2()">执行方法的第二种写法</button>
```

```
methods:{
  run2(){
    alert('执行方法的第二种写法：ES6语法：run2( ) { }')
  }
}
```

6-2 获取数据

点击按钮，弹出窗口提示：“你好，vue”。

```
67 <script>
68   export default {
69     data () {
70       return {
71         msg: '你好vue',
72       }
73     }
74   }
75 </script>
```

```
<button @click="getMsg()">获取data里面的msg</button>
```

```
getMsg(){
  alert(this.msg);
}
```

6-3 请求数据

点击按钮，数据列表的每一条数据将根据 key 索引，一一被渲染。其中 push（）方法的作用是：根据 for 循环的条件，往 list[] 数组添加新的数据。

```
data () {
  return {
    list:[]
  }
}
```

```
<button @click="requestData()">请求数据</button>
<ul>
  <li v-for="(item,key) in list">
    {{key}}---{{item}}
  </li>
</ul>
```

```
requestData(){
  for(var i=0;i<10;i++){
    this.list.push('我是第'+i+'条数据');
  }
}
```



6-4 改变数据

点击按钮，绑定的数据 msg 的初始值“你好，vue”将发生变化。

```
{msg}
<button @click="setMsg()">改变data里面的msg</button>

setMsg(){
  this.msg="我是改变后的数据"
}

data () {
  return {
    msg: '你好vue',
  }
}
```

6-5 执行方法传参

点击按钮，将弹出窗口提示：“111”。此处利用了方法传参的思想。

```
<button @click="deleteData('111')">执行方法传值111</button>

deleteData(value){
  alert(value);
}
```

6-6 事件对象 MouseEvent{...}

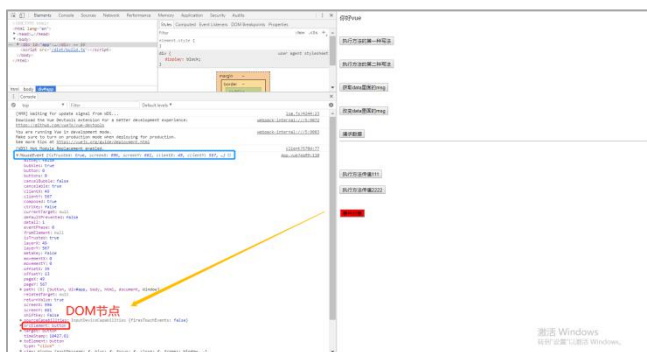
6.6.1 e.srcElement: 获取 DOM 节点

6.6.2 e.srcElement.dataset.aid: 获取自定义属性的值

console.log(e)=MouseEvent{...}。其中 **e.srcElement** 可以获取事件对象的 **DOM 节点**。

```
<button data-aid='123' @click="eventFn($event)">事件对象</button>
```

```
eventFn(e){
  console.log(e);
  // e.srcElement 获取dom节点
  e.srcElement.style.background='red';
  console.log(e.srcElement.dataset.aid); /*获取自定义属性的值*/
}
```



6.6.3 e.keyCode: 获取 keyCode 值对应按键

第 7 章 实现 todoList 代办事项列表

7-1 代办事项列表案例：www.todolist.cn

7-2 实现思路

7.2.1 纯数据添加



①在文本框输入数据 todoData（其中，初始绑定的 todoData 值为空值：""）。

```
<input type="text" v-model="todoData">
```

②点击“增加”按钮，获取文本框数据。

③利用 Javascript 中操作数组方法 `push()`，将获取的文本框数据堆栈至一个初始为空的数组 `list[]`。

```
<button @click="doAdd()"></button>
<script>
  export default {
    data() {
      return {
        todoData:"",
        list:[]
      }
    },
    methods:{
      doAdd(){
        this.list.push(this.todoData);
        this.todoData="";
      },
      removeData(key){
        this.list.splice(key,1);
      }
    }
  }
</script>
```

④ 利用 `v-for` 指令遍历渲染数组 `list[]`:

```
<ul><li v-for="item in list"></li></ul>
```

⑤ 假若想要删除新增的数据，可以在利用 `v-for` 渲染数组 `list[]` 的时候，使用 `key` 建立索引，添加删除方法 `removeData(key)`。此时，利用 Javascript 中数组操作方法 `splice(index,howmany)` 来删除索引对应的数据。

其中，参数 `index` 和 `howmany` 为必需项，`index` 指定了被删除数据的起始索引，即从哪里开始删除数据，`howmany` 指定了从起始索引处开始需要删除的数据的数量。

```
<ul>
  <li v-for="(item,key) in list">
    {{item}} ----
    <button @click="removeData(key)">删除</button>
  </li>
</ul>
```

7.2.2 将数据分成已完成和正进行两种状态

嘿嘿嘿

进行中

- ☐ 上课 ---- 删除
- ☐ 做作业 ---- 删除
- ☐ 跑步 ---- 删除
- ☐ 听音乐 ---- 删除
- ☐ 睡觉 ---- 删除

已完成

- ☒ 录制nodejs ---- 删除
- ☒ 录制ionic ---- 删除
- ☒ 起床 ---- 删除
- ☒ 刷牙 ---- 删除
- ☒ 洗脸 ---- 删除
- ☒ 吃饭 ---- 删除

① 我们在文本框输入的数据，点击添加按钮后，都是被渲染至同一个默认的有序列表中，为了将数据划分为：**【正进行】**和**【已完成】**两种状态。这时，我们给每一条数据附加一个状态标识。在此，我们为每一条数据添加一个 DOM 节点☐</input>，利用 v-model 指令绑定复选框 checkbox 的状态 checked (true||false)，并用 v-if 指令在 DOM 节点上根据数据所对应复选框 checkbox 的状态 checked (true||false) 来判断。假若未被选中，即 checked=false，则将数据渲染转移至**【正进行】**列表；假若被选中，即 checked=true，则渲染转移至**【已完成】**列表。具体实现思路如下：

```
<h2>进行中</h2>
<ul>
  <li v-for="(item,key) in list" v-if=" ! item.checked ">
    <input type="checkbox" v-model='item.checked'> {{item.title}} ----
    <button @click="removeData(key)">删除</button>
  </li>
</ul>
```

```
<h2>已完成</h2>
<ul class="finish">
  <li v-for="(item,key) in list" v-if=" item.checked ">
    <input type="checkbox" v-model='item.checked'> {{item.title}} ----
    <button @click="removeData(key)">删除</button>
  </li>
</ul>
```

在执行 doAdd () 方法时候，被添加数据默认的 checked 的值为 false。

```

<script>
  export default {
    data () {
      return {
        todoData:'',
        list: [
          {
            title: '录制 nodejs',
            checked: true
          },
          {
            title: '录制 ionic',
            checked: false
          }
        ]
      }
    },
    methods:{
      doAdd(e){
        console.log(e.keyCode);
        if(e.keyCode==13){
          this.list.push({
            title: this.todoData,
            checked: false
          })
          this.todoData=' ';
        }
      },
      removeData(key){
        this.list.splice(key,1);
      },
      getList(){
        console.log(this.list)
      }
    }
  }
</script>

```

② 优化：原本点击按钮触发 doAdd（）方法添加数据，现优化成按下 Enter 回车键即可触发 doAdd（）方法添加数据。此时，我们需要在 DOM 节点<input type="text" v-model='todo' />上进行键盘事件监听。

`console.log(e)=mouseEvent{...}`。其中，`e.keyCode` 获取按下的键盘按键的 Unicode 值：。假若是 Enter 回车键，对应的编码值为 13，即 `keyCode 13 = Enter` 键。

```
<input type="text" v-model='todo' @keydown="doAdd($event)" />
```

7.2.3 localStorage（未封装）实现刷新缓存持久化

此前，用户添加的代办项数据只能实现即时存储，只要执行页面刷新操作，客户端所有数据都将丢失，一切需要重来。为了解这个漏洞问题，我们引入 HTML5 中的 web 存储方式 [localStorage 对象](#)。具体思路如下：

① 每次增加删除数据或数据状态改变的时候，利用 `localStorage.setItem(key, value)` 来实现数据缓存。注意，`key` 为建立的存储空间索引，方便 `localStorage.getItem(key)` 根据索引读取数据，`value` 通常以字符串存储，因此必须利用 `JSON.stringify()` 方法将 JSON 数组 `list[]` 转换为 JSON 字符串。

`<input type="checkbox" v-model="item.checked" @change="saveList()" />` 状态改变要保存下。

```
methods:{
  doAdd(e){
    // console.log(e);
    if(e.keyCode==13){
      this.list.push({
        title:this.todoData,
        checked:false
      })
    }
    localStorage.setItem('list',JSON.stringify(this.list));
  },
  removeData(key){
    this.list.splice(key,1)
    localStorage.setItem('list', JSON.stringify(this.list));
  },
  saveList(){
    localStorage.setItem('list', JSON.stringify(this.list));
  }
}
```

```

<h2>进行中</h2>
<ul>
  <li v-for="(item,key) in list" v-if="!item.checked">
    <input type="checkbox" v-model="item.checked" @change="saveList()" />
    {{item.title}} --
    <button @click="removeData(key)">删除</button>
  </li>
</ul>

<h2>已完成</h2>
<ul>
  <li v-for="(item,key) in list" v-if="item.checked">
    <input type="checkbox" v-model="item.checked" @change="saveList()" />
    {{item.title}} --
    <button @click="removeData(key)">删除</button>
  </li>
</ul>

```

② 此时，数据已成功缓存。不过刷新页面时发现 **view** 视图里面数据的渲染好像失败了，一是因为缓存数据的格式为 JSON 字符串，二是因为可以成功渲染的数据格式为 JSON 数组 `list[]`。

要想把缓存数据再次渲染到 **view** 视图当中，我们分三步实现效果：

- 利用 `localStorage.getItem(key)` 方法读取保存的 JSON 字符串数据。
- 利用 [JSON.parse\(\)](#) 方法将 JSON 字符串数据转化为 JSON 数组。
- 判断 JSON 数组是否为空，假若数据存在，则赋值给 `this.list[]`。

生命周期函数 **mounted()**：即 **vue** 页面刷新便可触发的方法。

由此，最终成功实现了数据的本地持久化缓存！

```

mounted( ){
  var list=JSON.parse(localStorage.getItem('list'));
  console.log(list);
  if(list){ /*注意判断*/
    this.list=list;
  }
}

```

7.2.4 localStorage（已封装）实现刷新缓存持久化

为了简化代码逻辑，我们可以将 localStorage 本地存储、读取、删除的方法封装起来暴露出去，然后在.vue 组件中直接引入并调用即可。新建 storage.js 文件：

```
//封装操作 localStorage 本地存储的方法：模块化的文件
// nodejs 基础
var storage={
  setItem(key,value){ //存储数据
    localStorage.setItem(key,JSON.stringify(value));
  },
  getItem(key){ //读取数据
    return JSON.parse(localStorage.getItem(key));
  },
  removeItem(key){ //删除数据
    localStorage.removeItem(key);
  }
}
export default storage;//将模块暴露出去
```

```

<script>
  import storage from './model/storage.js';
  export default {
    data () {
      return {
        todoData:"",
        list: []
      }
    },
    methods:{
      doAdd(e){
        if(e.keyCode==13){
          this.list.push({
            title:this.todoData,
            checked:false
          })
        }
        storage.setItem('list',this.list);
      },
      removeData(key){
        this.list.splice(key,1)
        storage.setItem('list',this.list);
      },
      saveList(){
        storage.setItem('list',this.list);
      }
    },
    mounted(){
      var list=storage.getItem('list');
      if(list){ /*注意判断*/
        this.list=list;
      }
    }
  }
</script>

```

7.2.5 已完成和未完成数量的计算

为了方便用户直观的了解自身任务的数量，我们在添加、删除数据、改变数据状态以及页面

刷新时候，需要把 JSON 数组 list[] 的对象根据 checked 状态的不同拆分成两个数组：

① waitSize:[]

② completeSize:[]

然后在两个列表中分别这两个数组的长度 waitSize.length 和 completeSize.length，即可实现。

```
data () {  
  return {  
    waitSize:[],  
    completeSize:[],  
    todo: '' ,  
    list: []  
  }  
}
```

```
<div class="playing">  
<div>正进行</div>  
<div>{{waitSize.length}}</div>  
</div>
```

```
<div class="playing">  
<div>已完成</div>  
<div>{{completeSize.length}}</div>  
</div>
```

```
doAdd(e){  
  if(this.todo!=''){  
    if(e.keyCode==13){  
      this.list.push({  
        title:this.todo,  
        checked:false  
      })  
      this.todo='';  
      this.waitSize.push({  
        title:this.todo,  
        checked:false  
      });  
    }  
  }  
  storage.setItem('list',this.list);  
  storage.setItem('waitSize',this.waitSize);  
  storage.setItem('completeSize',this.completeSize);  
}
```

数据添加

```
saveList(){  
  storage.setItem('list',this.list);  
  /* 方法一 */  
  // this.waitSize.length=this.waitSize.length-1;  
  // this.completeSize.length=this.completeSize.length+1;  
  // storage.setItem('waitSize',this.waitSize);  
  // storage.setItem('completeSize',this.completeSize);  
  /* 方法二 */  
  this.waitSize=[];  
  this.completeSize=[];  
  var list=storage.getItem('list');  
  if(list){ /*注意判断*/  
    this.list=list;  
  }  
  console.log(this.list);  
  for(var i=0;i<this.list.length;i++){  
    if(this.list[i].checked==false){  
      this.waitSize.push(this.list[i]);  
      storage.setItem('waitSize',this.waitSize);  
    }else{  
      this.completeSize.push(this.list[i]);  
      storage.setItem('completeSize',this.completeSize);  
    }  
  }  
}
```

数据保存

```

removeData(key){
  this.list.splice(key,1); //删除数据
  storage.setItem('list',this.list); //重新保存最新数据
  //置空
  this.waitSize=[];
  this.completeSize=[];
  var list=storage.getItem('list');//获取最新数据
  if(list){ /*注意判断*/
    this.list=list;
  }
  // console.log(this.list);
  for(var i=0;i<this.list.length;i++){//遍历最新数据，根据状态checked往waitSize: []和completeSize: []添加数据对象
    if(this.list[i].checked==false){
      this.waitSize.push(this.list[i]);
      storage.setItem('waitSize',this.waitSize);
    }else{
      this.completeSize.push(this.list[i]);
      storage.setItem('completeSize',this.completeSize);
    }
  }
}
}

```

数据删除

```

mounted(){ /*生命周期函数 vue页面刷新就会触发的方法*/
  var list=storage.getItem('list');
  if(list){ /*注意判断*/
    this.list=list;
  }
  for(var i=0;i<this.list.length;i++){
    if(this.list[i].checked==false){
      this.waitSize.push(this.list[i]);
      storage.setItem('waitSize',this.waitSize);
    }else{
      this.completeSize.push(this.list[i]);
      storage.setItem('completeSize',this.completeSize);
    }
  }
}
}

```

页面刷新

第 8 章 vue 中键盘事件: @keydown="Fn(\$event)"

8-1 Vue 中为常用按键设置的别名:

https://blog.csdn.net/qq_38591756/article/details/88732443

别名	按键
.delete	delete（删除）/BackSpace（退格）
.tab	Tab
.enter	Enter（回车）
.esc	Esc（退出）
.space	Space（空格键）
.left	Left（左箭头）
.up	Up（上箭头）
.right	Right（右箭头）
.down	Down（下箭头）
.ctrl	Ctrl
.alt	Alt
.shift	Shift
.meta	windows 中为 window 键，mac 中为 command 键

8-2 keyCode 值及对应按键

keyCode	按键
48-57	0-9
65-90	a-z/A-Z
112-135	F1-F24
8	BackSpace (退格)
9	Tab
13	Enter (回车)
20	Caps_Lock (大写锁定)
32	Space (空格键)
37	Left (左箭头)
38	Up (上箭头)
39	Right (右箭头)
40	Down (下箭头)

8-3 判断方法

思路① **间接判断**：利用原生 e.keyCode。

```
<input type="text" v-model='todo' @keydown="doAdd($event)" />
```

```
doAdd(e){
  console.log(e.keyCode)
  if(e.keyCode==13){
    //1、获取文本框输入的值    2、把文本框的值push到list里面
    this.list.push({
      title: this.todo,
      checked: false
    })
    this.todo='';
  }
}
```

思路② **直接判断**：利用 vue 自身提供的修饰符 @keydown.enter="doAdd () "。

以上 enter，也可以写为数字 @keydown.13="fn"

其他类似 @keydown.up @keydown.down @keydown.right @keydown.left

```
<input type="text" v-model='todo' @keydown.13="doAdd($event)" />
```

```
doAdd(e){
  this.list.push({
    title: this.todo,
    checked: false
  })
  this.todo='';
}
```


第 9 章 CSS 样式

9-1 div 内文字上下左右居中

div 中的文字水平居中：text-align:center；即可。

div 中的文字垂直居中：line-height:值。值等于 div 的高度。

```
56 .playing div:nth-child(2){  
57     background-color: black;  
58     color: aliceblue;  
59     float: right;  
60     width: 36px;  
61     height: 36px;  
62     border-radius: 18px 18px 18px 18px;  
63     text-align: center;  
64     line-height: 36px;  
65 }
```

正进行

0

9-2 无序列表消除小黑点

list-style:none;