Vue3的表单和开发模式

王红元 coderwhy

v-model的基本使用

- **表单提交**是开发中非常常见的功能，也是和用户交互的重要手段：
  - 比如用户在登录、注册时需要提交账号密码；
  - 比如用户在检索、创建、更新信息时，需要提交一些数据；
- 这些都要求我们可以在**代码逻辑中获取到用户提交的数据**，我们通常会使用**v-model指令**来完成：
  - v-model指令可以在表单 input、textarea以及select元素上创建双向数据绑定；
  - 它会根据控件类型自动选取正确的方法来更新元素；
  - 尽管有些神奇，但 v-model 本质上不过是语法糖，它负责监听用户的输入事件来更新数据，并在某种极端场景下进行一些特殊处理；

```
<template id="my-app">
  <input type="text" v-model="message">
  <h2>{{message}}</h2>
</template>
```

v-model的原理

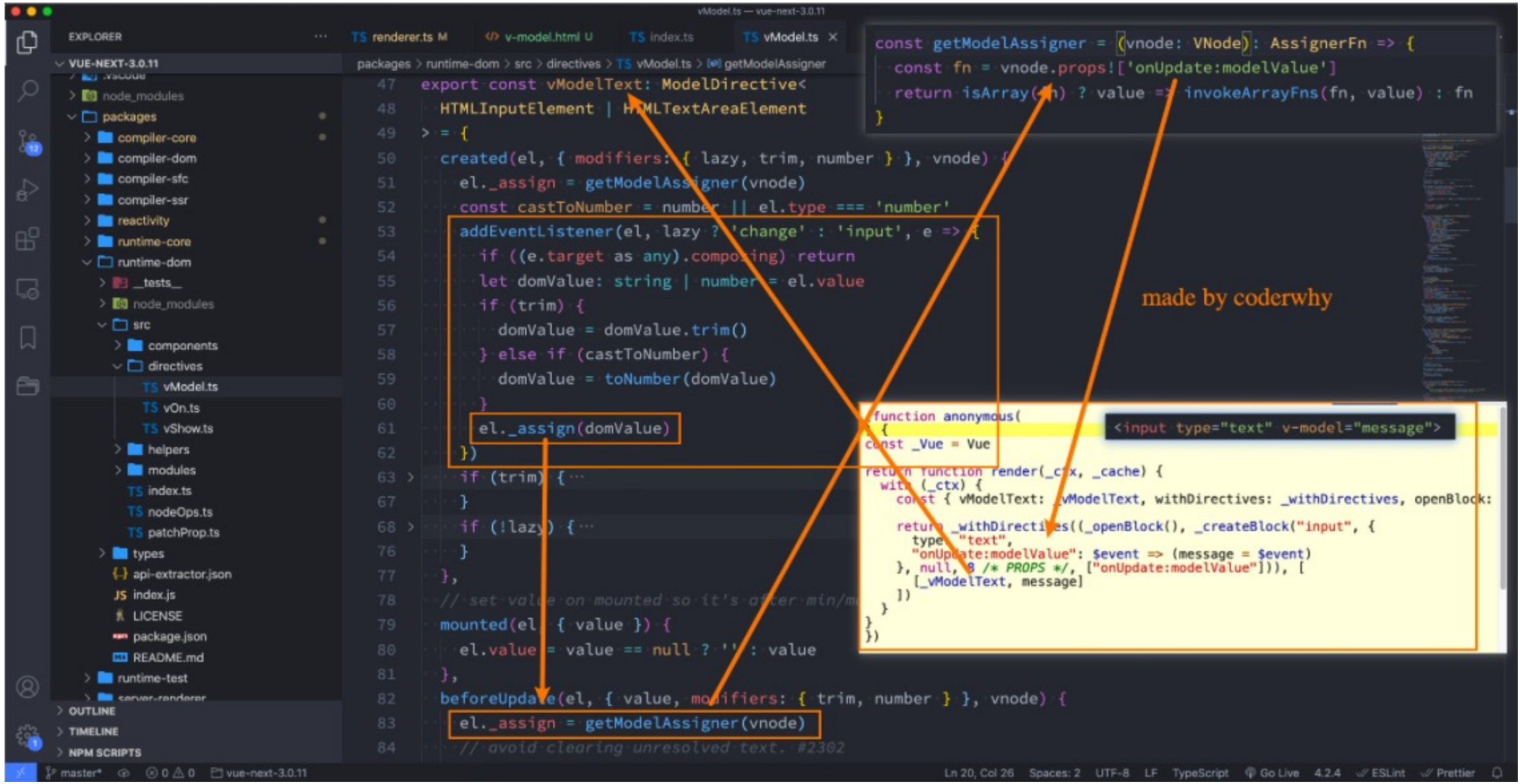
- 官方有说到，**v-model的原理**其实是背后有两个操作：
  - v-bind绑定value属性的值；
  - v-on绑定input事件监听到函数中，函数会获取最新的值赋值到绑定的属性中；

```html
<input v-model="searchText" />
```

等价于：

```html
<input :value="searchText" @input="searchText = $event.target.value" />
```

事实上v-model更加复杂



v-model绑定textarea

- 我们再来绑定一下**其他的表单类型**：textarea、checkbox、radio、select
- 我们来看一下绑定textarea：

```
<!-- 1.绑定text-area -->
<div>
  <textarea v-model="article" cols="30" rows="10"></textarea>
  <h2>article当前的值是: {{article}}</h2>
</div>
```

v-model绑定checkbox

- 我们来看一下v-model绑定**checkbox**：单个勾选框和多个勾选框
- **单个勾选框：**
  - v-model即为布尔值。
  - 此时input的value并不影响v-model的值。
- **多个复选框：**
  - 当是多个复选框时，因为可以选中多个，所以对应的data中属性是一个数组。
  - 当选中某一个时，就会将input的value添加到数组中。

```
<!-- 2.1.单选框 -->
<div>
  <label for="agreement">
    <input id="agreement" type="checkbox" v-model="isAgree">同意协议
  </label>
  <h2>isAgree当前的值是: {{isAgree}}</h2>
</div>
```

```
<!-- 2.2.多选框 -->
<div>
  <label for="basketball">
    <input id="basketball" type="checkbox" value="basketball" v-model="hobbies">篮球
  </label>
  <label for="football">
    <input id="football" type="checkbox" value="football" v-model="hobbies">足球
  </label>
  <label for="tennis">
    <input id="tennis" type="checkbox" value="tennis" v-model="hobbies">网球
  </label>
  <h2>hobbies当前的值是: {{hobbies}}</h2>
</div>
```

v-model绑定radio

- v-model绑定**radio**，用于选择其中一项；

```
<!-- 3.绑定radio -->
<div>
  <label for="male">
    <input type="radio" id="male" v-model="gender" value="male">男
  </label>
  <label for="female">
    <input type="radio" id="female" v-model="gender" value="female">女
  </label>
  <h2>gender当前的值是: {{gender}}</h2>
</div>
```

v-model绑定select

- **和checkbox一样，select也分单选和多选两种情况。**
- **单选：只能选中一个值**
  - v-model绑定的是一个值；
  - 当我们选中option中的一个时，会将它对应的value赋值到fruit中；
- **多选：可以选中多个值**
  - v-model绑定的是一个数组；
  - 当选中多个值时，就会将选中的option对应的value添加到数组fruit中；

```
<div>
  <select v-model="fruit">
    <option value="apple">苹果</option>
    <option value="orange">橘子</option>
    <option value="banana">香蕉</option>
  </select>
  <h2>fruit当前的值是: {{fruit}}</h2>
</div>
```

```
<div>
  <select v-model="fruit" multiple size="3">
    <option value="apple">苹果</option>
    <option value="orange">橘子</option>
    <option value="banana">香蕉</option>
  </select>
  <h2>fruit当前的值是: {{fruit}}</h2>
</div>
```

v-model的值绑定

- 目前我们在前面的案例中大部分的值都是在template中固定好的：
  - 比如gender的两个输入框值male、female；
  - 比如hobbies的三个输入框值basketball、football、tennis；
- 在真实开发中，我们的数据可能是来自服务器的，那么我们就可以先将值请求下来，绑定到data返回的对象中，再通过v-bind来进行值的绑定，这个过程就是**值绑定**。
  - 这里不再给出具体的做法，因为还是v-bind的使用过程。

v-model修饰符 - lazy

- lazy修饰符是什么作用呢？
  - 默认情况下，v-model在进行双向绑定时，绑定的是input事件，那么会在每次内容输入后就将最新的值和绑定的属性进行同步；
  - 如果我们在v-model后跟上lazy修饰符，那么会将绑定的事件切换为 change 事件，只有在提交时（比如回车）才会触发；

```
<template id="my-app">
  <input type="text" v-model.lazy="message">
  <h2>{{message}}</h2>
</template>
```

v-model修饰符 - number

- 我们先来看一下v-model绑定后的值是什么类型的：
  - message总是string类型，即使在我们设置type为number也是string类型；

```
<template id="my-app">
  <!-- 类型 -->
  <input type="text" v-model="message">
  <input type="number" v-model="message">
  <h2>{{message}}</h2>
</template>
```

- 如果我们希望转换为数字类型，那么可以使用 .number 修饰符：

```
<input type="text" v-model.number="score">
```

- 另外，在我们进行逻辑判断时，如果是一个string类型，在可以转化的情况下会进行隐式转换的：
  - 下面的score在进行判断的过程中会进行隐式转化的；

```
const score = "100";
if (score > 90) {
  console.log("优秀");
}
console.log(typeof score);
```

v-model修饰符 - trim

- 如果要自动过滤用户输入的首尾空白字符，可以给v-model添加 trim 修饰符：

```
<template id="my-app">
  <!-- 去除空格 -->
  <input type="text" v-model.trim="message">
</template>
```

v-mode组件上使用

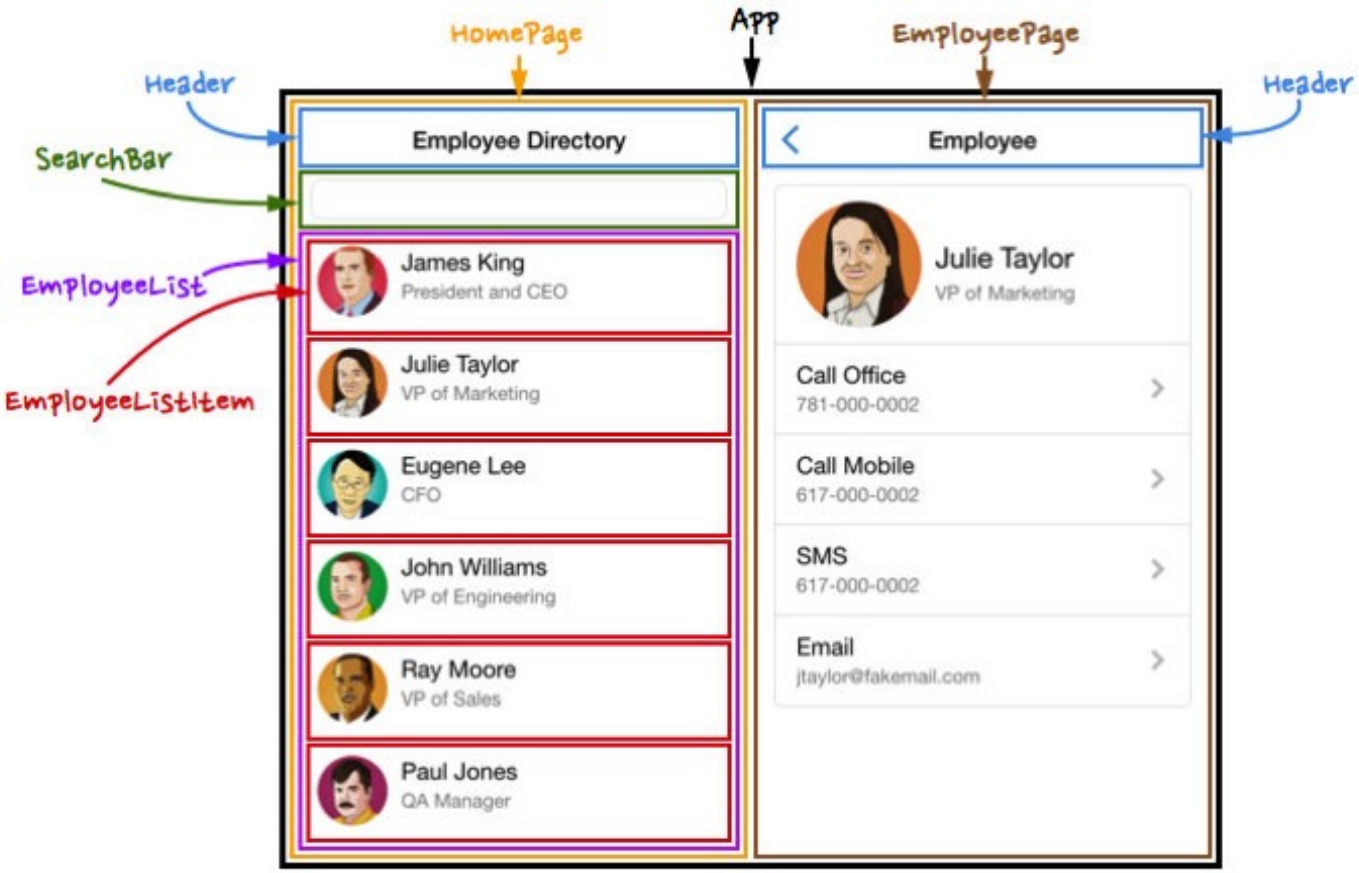
- v-model也可以使用在组件上，Vue2版本和Vue3版本有一些区别。
  - 具体的使用方法，后面讲组件化开发再具体学习。

人处理问题的方式

- **人面对复杂问题的处理方式：**
  - 任何一个人处理信息的逻辑能力都是有限的
  - 所以，当面对一个非常复杂的问题时，我们不太可能一次性搞定一大堆的内容。
  - 但是，我们人有一种天生的能力，就是将问题进行拆解。
  - 如果将一个复杂的问题，拆分成很多个可以处理的小问题，再将其放在整体当中，你会发现大的问题也会迎刃而解。

认识组件化开发

- 组件化也是类似的思想：
  - 如果我们将一个页面中所有的处理逻辑全部放在一起，处理起来就会变得非常复杂，而且不利于后续的管理以及扩展；
  - 但如果，我们讲一个页面拆分成一个个小的功能块，每个功能块完成属于自己这部分独立的功能，那么之后整个页面的管理和维护就变得非常容易了；
  - 如果我们将一个个功能块拆分后，就可以像搭建积木一下来搭建我们的项目；



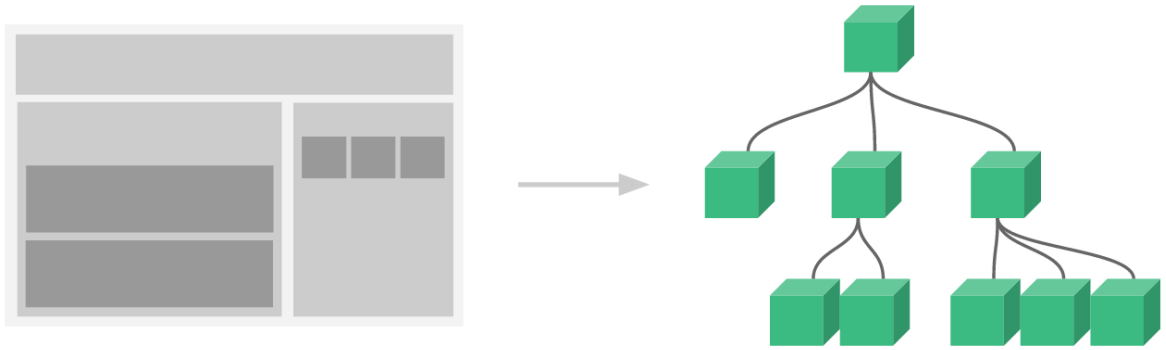
组件化开发

- **现在可以说整个的大前端开发都是组件化的天下**，无论从三大框架（Vue、React、Angular），还是跨平台方案的Flutter，甚至是移动端都在转向组件化开发，包括小程序的开发也是采用组件化开发的思想。
- 所以，学习组件化最重要的是它的思想，每个框架或者平台可能实现方法不同，但是思想都是一样的。

- 我们需要通过组件化的思想来思考整个应用程序：
  - 我们将一个完整的页面分成很多个组件；
  - 每个组件都用于实现页面的一个功能块；
  - 而每一个组件又可以进行细分；
  - 而组件本身又可以在多个地方进行复用；

Vue的组件化

- **组件化是Vue、React、Angular的核心思想，也是我们后续课程的重点（包括以后实战项目）：**
  - 前面我们的createApp函数传入了一个对象App，这个对象其实本质上就是一个组件，也是我们应用程序的根组件；
  - 组件化提供了一种抽象，让我们可以开发出一个个独立可复用的小组件来构造我们的应用；
  - 任何的应用都会被抽象成一颗组件树；



- 接下来，我们来学习一下在**Vue中如何注册一个组件**，以及之后**如何使用这个注册后的组件。**

注册组件的方式

- 如果我们现在有一部分**内容（模板、逻辑等）**，我们希望将这部分内容抽取到一个**独立的组件**中去维护，这个时候**如何注册一个组件**呢？
- 我们先从简单的开始谈起，比如下面的模板希望抽离到一个单独的组件：

```
<h2>{{title}}</h2>
<p>{{message}}</p>
```

- **注册组件分成两种：**
  - 全局组件：在任何其他的组件中都可以使用的组件；
  - 局部组件：只有在注册的组件中才能使用的组件；

注册全局组件

- 我们先来学习一下全局组件的注册：
  - 全局组件需要使用我们全局创建的app来注册组件；
  - 通过component方法传入组件名称、组件对象即可注册一个全局组件了；
  - 之后，我们可以在App组件的template中直接使用这个全局组件：

```
<template id="my-cpn">
  <h2>我是组件标题</h2>
  <p>我是组件内容，哈哈哈哈</p>
</template>

<script src="../js/vue.js"></script>
<script>
  const app = Vue.createApp(App);

  // 注册全局组件(使用app)
  app.component("my-cpn", {
    template: "#my-cpn"
  });

  app.mount('#app');
</script>
```

```
<template id="my-app">
  <my-cpn></my-cpn>
  <my-cpn></my-cpn>
  <my-cpn></my-cpn>
  <my-cpn></my-cpn>
</template>
```

全局组件的逻辑

- 当然，我们组件本身也可以有自己的代码逻辑：
  - 比如自己的data、computed、methods等等

```
// 注册全局组件(使用app)
app.component("my-cpn", {
  template: "#my-cpn",
  data() {
    return {
      title: "我是标题",
      message: "我是内容，哈哈哈哈"
    }
  },
  methods: {
    btnClick() {
      console.log("btnClick");
    }
  }
});
```

组件的名称

- 在通过app.component注册一个组件的时候，第一个参数是组件的名称，定义组件名的方式有两种：
- **方式一：使用kebab-case（短横线分割符）**
  - 当使用 kebab-case (短横线分隔命名) 定义一个组件时，你也必须在引用这个自定义元素时使用 kebab-case，例如 <my-component-name>；

```
app.component('my-component-name', {
  /* ... */
})
```

- **方式二：使用PascalCase（驼峰标识符）**
  - 当使用 PascalCase (首字母大写命名) 定义一个组件时，你在引用这个自定义元素时两种命名法都可以使用。也就是说 <my-component-name> 和 <MyComponentName> 都是可接受的；

```
app.component('MyComponentName', {
  /* ... */
})
```

注册局部组件

- 全局组件往往是在应用程序一开始就会**全局组件**完成，那么就意味着如果**某些组件我们并没有用到，也会一起被注册：**
  - 比如我们注册了三个全局组件：ComponentA、ComponentB、ComponentC；
  - 在开发中我们只使用了ComponentA、ComponentB，如果ComponentC没有用到但是我们依然在全局进行了注册，那么就意味着类似于webpack这种打包工具在打包我们的项目时，我们依然会对其进行打包；
  - 这样最终打包出的JavaScript包就会有关于ComponentC的内容，用户在下载对应的JavaScript时也会增加包的大小；
- **所以在开发中我们通常使用组件的时候采用的都是局部注册：**
  - 局部注册是在我们需要使用到的组件中，通过components属性选项来进行注册；
  - 比如之前的App组件中，我们有data、computed、methods等选项了，事实上还可以有一个components选项；
  - 该components选项对应的是一个对象，对象中的键值对是 组件的名称: 组件对象；

布局组件注册代码

```
const ComponentA = {
  template: "#component-a",
  data() {
    return {
      title: "我是ComponentA标题",
      message: "我是ComponentA内容，哈哈哈哈"
    }
  }
}

const ComponentB = {
  template: "#component-b",
  data() {
    return {
      title: "我是ComponentB标题",
      message: "我是ComponentB内容，呵呵呵呵"
    }
  }
}
```

```
const App = {
  template: '#my-app',
  components: {
    'component-a': ComponentA,
    'component-b': ComponentB,
  },
  data() {
    return {
      message: "Hello World"
    }
  }
}

Vue.createApp(App).mount('#app');
```

Vue的开发模式

- 目前我们使用vue的过程都是**在html文件中**，通过template编写自己的模板、脚本逻辑、样式等。
- **但是随着项目越来越复杂，我们会采用组件化的方式来进行开发：**
  - 这就意味着每个组件都会有自己的模板、脚本逻辑、样式等；
  - 当然我们依然可以把它们抽离到单独的js、css文件中，但是它们还是会分离开来；
  - 也包括我们的script是在一个全局的作用域下，很容易出现命名冲突的问题；
  - 并且我们的代码为了适配一些浏览器，必须使用ES5的语法；
  - 在我们编写代码完成之后，依然需要通过工具对代码进行构建、代码；
- 所以在真实开发中，我们可以通过一个后缀名为 .vue 的**single-file components (单文件组件)** 来解决，并且可以使用webpack或者vite或者rollup等构建工具来对其进行处理。

单文件的特点

- **在这个组件中我们可以获得非常多的特性：**
  - 代码的高亮；
  - ES6、CommonJS的模块化能力；
  - 组件作用域的CSS；
  - 可以使用预处理器来构建更加丰富的组件，比如TypeScript、Babel、Less、Sass等；

```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function() {
    return {
      greeting: "Hello"
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

如何支持SFC

- 如果我们想要使用这一的SFC的.vue文件，比较**常见的是两种方式**：
  - 方式一：使用Vue CLI来创建项目，项目会默认帮助我们配置好所有的配置选项，可以在其中直接使用.vue文件；
  - 方式二：自己使用webpack或rollup或vite这类打包工具，对其进行打包处理；

- 我们最终，无论是后期我们做项目，还是在公司进行开发，通常都会采用Vue CLI的方式来完成。

- 但是在学习阶段，为了让大家理解Vue CLI打包项目的过程，我会接下来穿插讲解一部分webpack的知识，帮助大家更好的理解Vue CLI的原理以及其打包的过程。

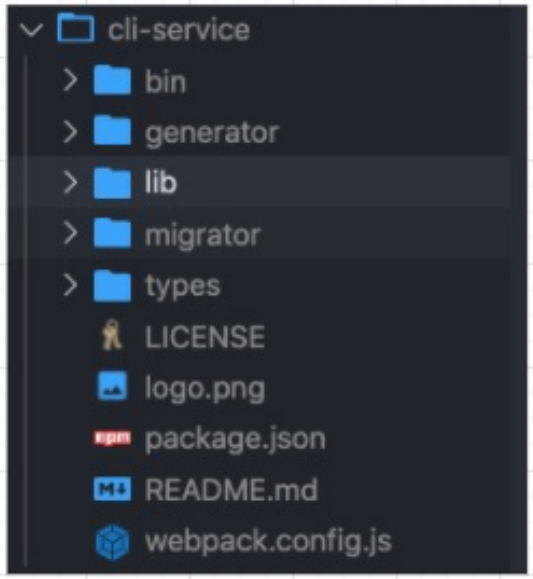
认识webpack

- **事实上随着前端的快速发展，目前前端的开发已经变的越来越复杂了：**
  - 比如开发过程中我们需要通过模块化的方式来开发；
  - 比如也会使用一些高级的特性来加快我们的开发效率或者安全性，比如通过ES6+、TypeScript开发脚本逻辑，通过sass、less等方式来编写css样式代码；
  - 比如开发过程中，我们还希望实时的监听文件的变化来并且反映到浏览器上，提高开发的效率；
  - 比如开发完成后我们还需要将代码进行压缩、合并以及其他相关的优化；
  - 等等....
- 但是对于很多的**前端开发者**来说，并不需要思考这些问题，日常的开发中根本就没有面临这些问题：
  - 这是因为目前前端开发我们通常都会直接使用三大框架来开发：Vue、React、Angular；
  - 但是事实上，这三大框架的创建过程我们都是借助于脚手架（CLI）的；
  - 事实上Vue-CLI、create-react-app、Angular-CLI都是基于webpack来帮助我们支持模块化、less、TypeScript、打包优化等的；

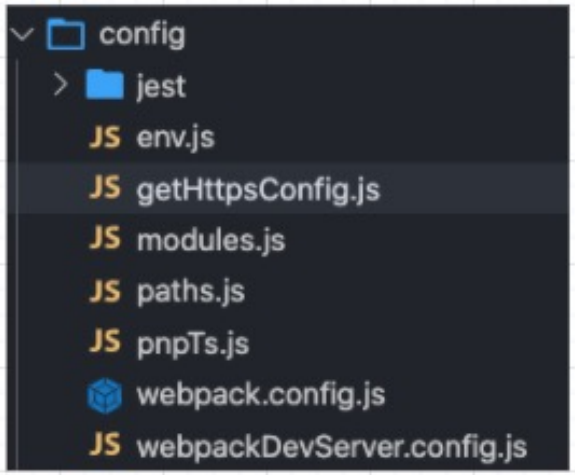
脚手架依赖webpack

- 事实上我们上面提到的所有脚手架都是依赖于webpack的：

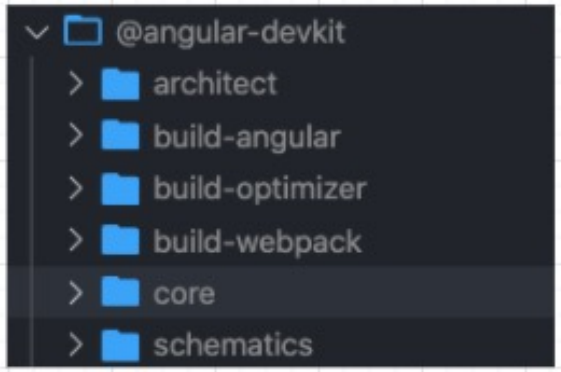
Vue webpack配置



React webpack配置



Angular webpack配置



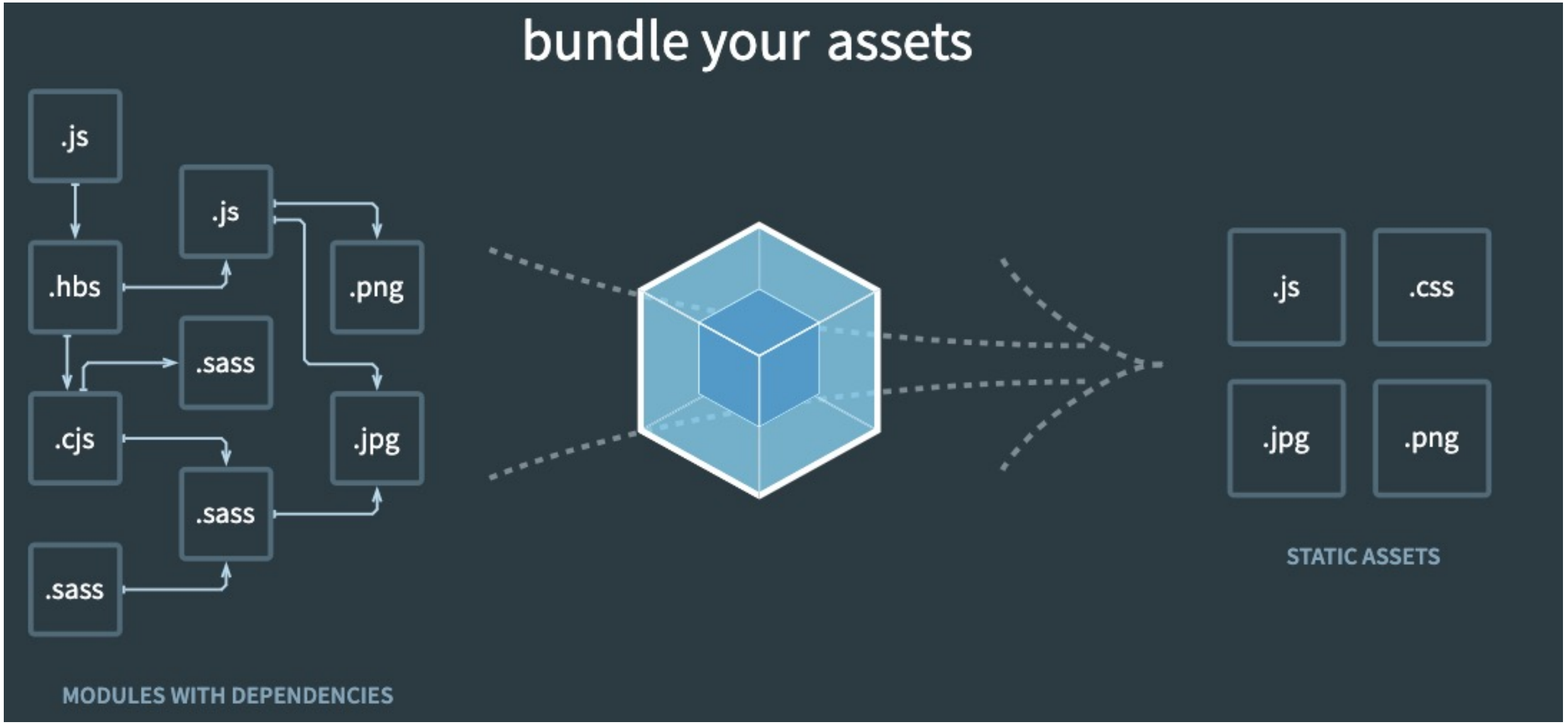
Webpack到底是什么呢？

- 我们先来看一下官方的解释：

```
webpack is a static module bundler for modern JavaScript applications.
```

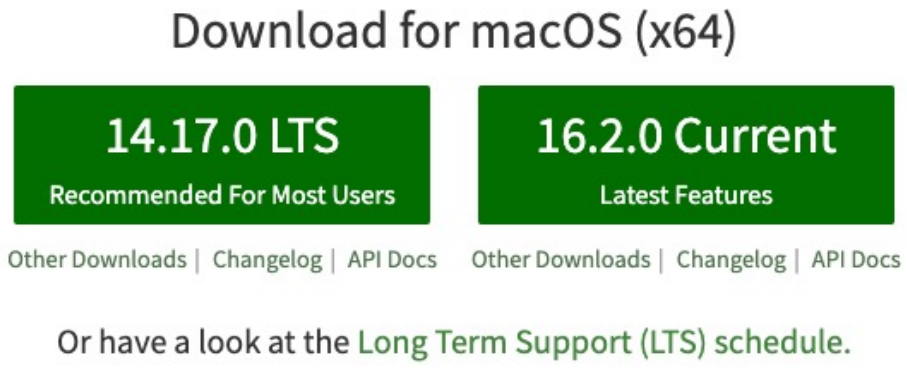
- **webpack是一个静态的模块化打包工具，为现代的JavaScript应用程序；**
- 我们来对上面的解释进行拆解：
  - 打包bundler：webpack可以将帮助我们进行打包，所以它是一个打包工具
  - 静态的static：这样表述的原因是我们最终可以将代码打包成最终的静态资源（部署到静态服务器）；
  - 模块化module：webpack默认支持各种模块化开发，ES Module、CommonJS、AMD等；
  - 现代的modern：我们前端说过，正是因为现代前端开发面临各种各样的问题，才催生了webpack的出现和发展；

Webpack官方的图片



Webpack的使用前提

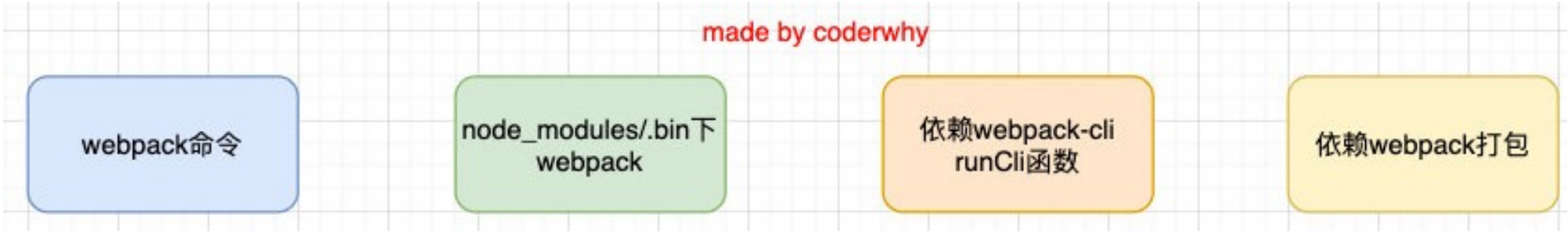
- webpack的官方文档是https://webpack.js.org/
  - webpack的中文官方文档是https://webpack.docschina.org/
  - DOCUMENTATION：文档详情，也是我们最关注的
- Webpack的运行是依赖Node环境的，所以我们电脑上必须有Node环境
  - 所以我们需要先安装Node.js，并且同时会安装npm；
  - 我当前电脑上的node版本是v14.15.5，npm版本是6.14.11（你也可以使用nvm或者n来管理Node版本）；
  - Node官方网站：https://nodejs.org/




```
[coderwhy@why ~ % node --version
v14.15.5
[coderwhy@why ~ % npm --version
6.14.11
```

Webpack的安装

- webpack的安装目前分为两个：webpack、webpack-cli
- **那么它们是什么关系呢？**
  - 执行webpack命令，会执行node_modules下的.bin目录下的webpack；
  - webpack在执行时是依赖webpack-cli的，如果没有安装就会报错；
  - 而webpack-cli中代码执行时，才是真正利用webpack进行编译和打包的过程；
  - 所以在安装webpack时，我们需要同时安装webpack-cli（第三方的脚手架事实上是没有使用webpack-cli的，而是类似于自己的vue-service-cli的东西）




```
npm install webpack webpack-cli -g # 全局安装
npm install webpack webpack-cli -D # 局部安装
```

Webpack的默认打包

- 我们可以通过webpack进行打包，之后运行打包之后的代码
  - 在目录下直接执行 webpack 命令

```
webpack
```

- **生成一个dist文件夹，里面存放一个main.js的文件，就是我们打包之后的文件：**
  - 这个文件中的代码被压缩和丑化了；
  - 我们暂时不关心他是如何做到的，后续我讲webpack实现模块化原理时会再次讲到；
  - 另外我们发现代码中依然存在ES6的语法，比如箭头函数、const等，这是因为默认情况下webpack并不清楚我们打包后的文件是否需要转成ES5之前的语法，后续我们需要通过babel来进行转换和设置；
- **我们发现是可以正常进行打包的，但是有一个问题，webpack是如何确定我们的入口的呢？**
  - 事实上，当我们运行webpack时，webpack会查找当前目录下的 src/index.js作为入口；
  - 所以，如果当前项目中没有存在src/index.js文件，那么会报错；