Vue快速入门

1、简介

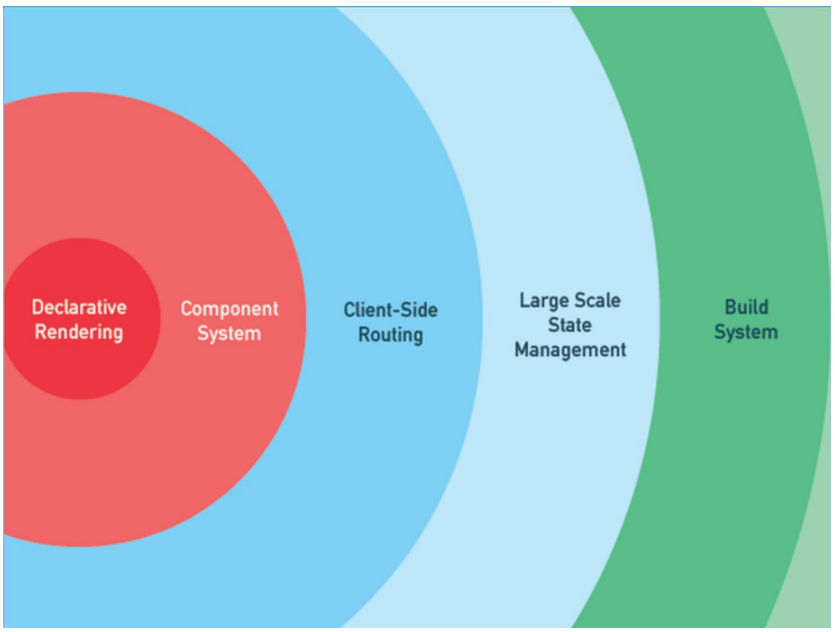
Vue (读音 /vju/，类似于 **view**)
是中国的大神**尤雨溪**开发的，为数不多的国人开发的世界顶级开源软件。
是一套用于构建用户界面的**渐进式框架** 。Vue 被设计为可以自底向上逐层应用。
MVVM响应式编程模型，避免直接操作DOM，降低DOM操作的复杂性。

1.2、MVVM编程思想

MVVM：页面输入改变数据，数据改变影响页面数据展示与渲染

M（model）：普通的javascript数据对象
V（view）：前端展示页面
VM（ViewModel）：用于双向绑定数据与页面，对于我们的课程来说，就是vue的实例

1.3、渐进式框架



“渐进式框架” 非常简单，就是用你想用或者能用的功能特性，你不想用的部分功能可以先不用。**VUE不强求你一次性接受并使用它的全部功能特性。**

- 场景一：公司刚开始一个项目，技术人员对Vue的掌握也不足够。那么我们就不能使用VUE了么？当然不是，如果你只是使用VUE做些**基础操作**，如：页面渲染、表单处理提交功能，那还是非常简单的，成熟技术人员上手也就一两天。完全可以用它去代替jquery。并不需要你去引入其他复杂特性功能。

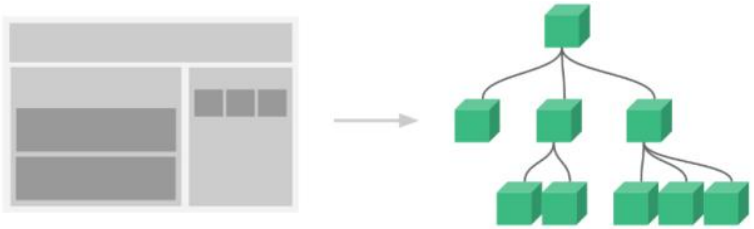
- 场景二：我们项目用了VUE，使用的效果也挺好。那么我们想逐渐实现代码组件化，实现代码的复用，或者是基于组件原型的跨项目的代码复用。那么我们就可以引入VUE的components组件特性了。

- 场景三：我们的项目规模逐渐的变大了，我们可能会逐渐用到前端路由、状态集中管理、并最终实现一个高度工程化的前端项目。这些功能特性我们可以逐步引入，当然不用也可以。

所以VUE的适用面很广，你可以用它代替老项目中的JQuery。也可以在新项目启动初期，有限的使用VUE的功能特性，从而降低上手的成本。

1.4、VUE核心功能

基础功能：页面渲染、表单处理提交、帮我们管理DOM(虚拟DOM)节点
组件化开发：增强代码的复用能力，复杂系统代码维护更简单



前端路由：更流畅的的用户体验、灵活的在页面切换已渲染组件的显示，不需与后端做多余的交互
状态集中管理：MVVM响应式模型基础上实现多组件之间的状态数据同步与管理
前端工程化：结合webpack等前端打包工具，管理多种静态资源，代码，测试，发布等，整合前端大型项目。

2.Vue—Hello World

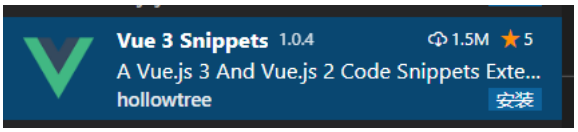
- **Vue的其他插件安装**

在使用 Vue 时，我们推荐在你的浏览器上安装 Vue Devtools。它允许你在一个更友好的界面中审查和调试 Vue 应用。

- **官網**

https://cn.vuejs.org/v2/guide/installation.html

- **可以在vscode中提示vue代码**



1. 初始化npm项目

```
npm init -y
```

2. 安装vue

```
npm install vue
```

3. html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>

<body>

    <div id="app">
        <h1> {{name}} ,非常帅</h1>
    </div>

    <script src="./node_modules/vue/dist/vue.js"></script>

    <script>
        //1、vue声明式渲染
        let vm = new Vue({
            el: "#app",//绑定元素
            data: {  //封装数据
                name: "张三"
            }
        });
    </script>
</body>

</html>
```

3.指令

基本概念

- 双向绑定
  - 模型变化 视图也会随之变化。
  - 视图变化 模型也会随之变化
- 方法
  - methods

v-text/v-html: 指定标签体

v-if v-else v-show

v-for : 遍历

v-on : 绑定事件监听

v-bind : 属性绑定

v-text/v-html: 指定标签体

- **v-text** : 当作纯文本
  - v-text是元素的 InnerText 属性，它的作用和之前我们使用的 {{}} 一样，用于数据绑定：

```

<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <div v-text="message"></div>
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script>
 var vm = new Vue({
 el: '#app',
 data: {
 message: "Hello!"
 },
})
</script>
</html>
```

- **v-html** : 将value作为html标签来解析
  - v-html是元素的 innerHTML，它用于绑定一段 html 标签：

```


<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Document</title>
</head>
<body>
<div id="app">
<div v-html="message"></div>
</div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script>
var vm = new Vue({
el: '#app',
data: {
message: "<div>您好，我是徐庶！</div>",
}
})
</script>
</html>
```

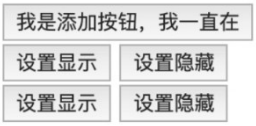
v-if v-else v-show

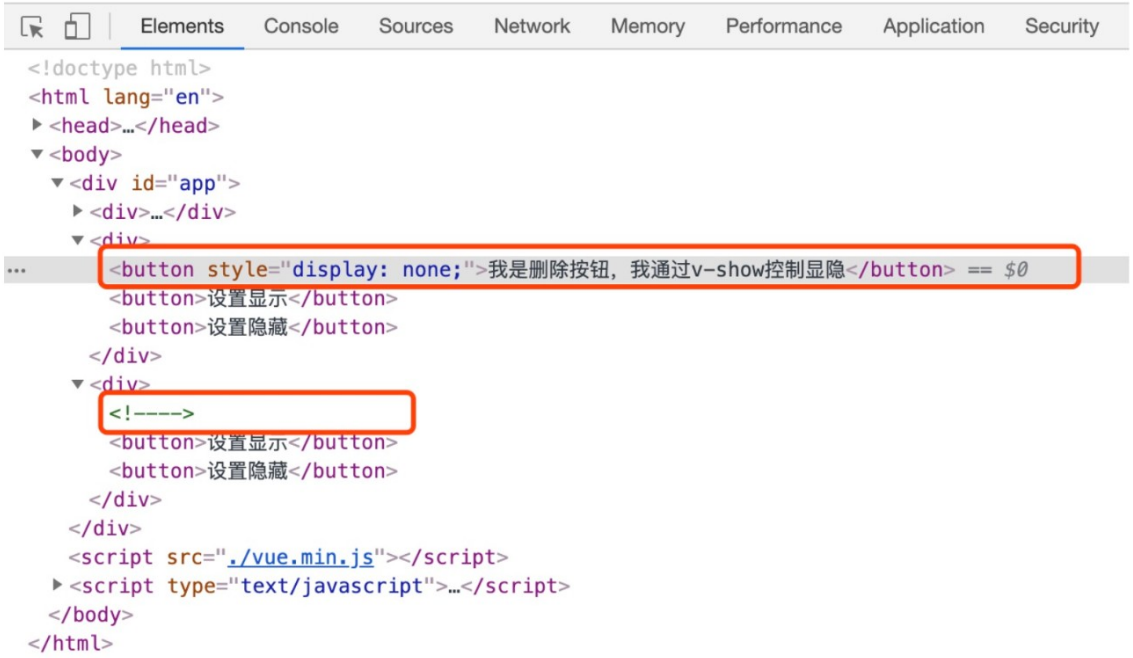
- vue提供了v-if和v-show两个指令来控制页面元素的显示和隐藏。我们先通过一段代码来看一下使用两个指令各有什么效果：
  - **v-if** : 如果vlaue为true, 当前标签会输出在页面中
  - **v-show**: 就会在标签中添加display样式, 如果vlaue为true, display=block, 否则是none
  - v-else : 与v-if一起使用, 如果value为false, 将当前标签输出到页面中

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Document</title>
</head>
<body>
<div id="app">
<div>
<button>我是添加按钮，我一直在</button>
</div>
<div>
<button id="show" v-show="deleteButton">我是删除按钮，我通过v-show控制显隐</button>
<button v-on:click="deleteButton = true">设置显示</button>
<button v-on:click="deleteButton = false">设置隐藏</button>
</div>
<div>
<button id="if" v-if="editButton">我是修改按钮，我通过v-if控制显隐</button>
<button v-on:click="editButton = true">设置显示</button>
<button v-on:click="editButton = false">设置隐藏</button>
</div>
```

```
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
var vm = new Vue({
 el: '#app',
 data() {
 return {
 deleteButton: true,
 editButton: true
 }
 }
})
</script>
</html>

```





- 上面我们已经了解了`v-if`的使用方法。事实上，`v-if`的条件渲染和JavaScript条件判断语句中的`if、else、else if`非常类似。
- **v-else、v-else-if**：与v-if一起使用，如果value为false，将当前标签输出到页面中

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

```
 <title>Document</title>
</head>
<body>
 <div id="app">
 <div v-if="number === 1">
 A
 </div>
 <div v-else-if="number === 2">
 B
 </div>
 <div v-else>
 C
 </div>
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
var vm = new Vue({
 el: '#app',
 data() {
 return {
 number: 1
 }
 }
})
</script>
</html>
```

v-for : 遍历

- `v-for` 用于列表的循环渲染。基本语法：`v-for="item in data"`，data 可以是数组或者对象，接下来我们介绍对两种数据类型的循环。
  - 遍历数组 : v-for="person in persons" $index

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <ul>
 <li v-for="item in music">{{item.name}}</li>
 </ul>
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
var vm = new Vue({
 el: '#app',
 data() {
```

```
 return {
 // 要循环的数组
 music: [
 { name: '青花瓷' },
 { name: '阳光总在风雨后' },
 { name: '十年' }
 ]
 }
 }
})
</script>
</html>

```

- 遍历对象 : v-for="value in person" $key

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <ul>
 <li v-for="item in obj">{{item}}</li>
 </ul>
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
var vm = new Vue({
 el: '#app',
 data() {
 return {
 obj: {
 name: '句号',
 age: 18,
 sex: '男'
 }
 }
 }
})
</script>
</html>
```

v-on：绑定事件监听

- 有时候，我们需要给元素绑定事件，`vue` 中提供了指令 `v-on` 来进行事件的绑定。用法：`v-on:事件名="方法"`，例如：v-on：click=“alert”。
- v-on:事件名, 可以缩写为: @事件名

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <button v-on:click="hello">hello</button>
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
var vm = new Vue({
 el: '#app',
 data: {},
 methods: {
 hello() {
 alert('hello')
 }
 }
})
</script>
</html>
```

v-bind：属性绑定

- v-bind用于给元素的属性赋值。v-bind后面是： 属性名=[变量名]。例如:v-bind:title="message":
- 很多属性值是不支持表达式的, 就可以使用v-bind

代码解释：

在 HTML 代码第 2 行，我们使用了 v-bind 指令给 div 标签的 title 属性赋值。

在 HTML 代码第 6 行，我们使用了 v-bind 指令给 a 标签的 href 属性赋值。

在 HTML 代码第 9 行，我们使用了 v-bind 指令给 img 标签的 src 属性赋值。

在 HTML 代码第 12 行，我们使用了 v-bind 指令给 bitton 标签的 disabled 属性赋值。

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <div v-bind:title="title">
 鼠标悬停查看消息!
 </div>
 <div>
 <a v-bind:href="href">图灵学院</a>
 </div>
 <div>
 <img v-bind:src="src"/>
 </div>
```

```
  <div>
  <button v-bind:disabled="disabled">禁用按钮</button>
  </div>
  </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
  var vm = new Vue({
  el: '#app',
  data: {
  title: "您好，我是徐庶老师，欢迎来到图灵学院！希望你可以在图灵学院学到更多的东西！",
  href: 'https://www.tulingxueyuan.cn/',
  src: 'http://img2.mukewang.com/szimg/5df1deec09ba554712000676-360-202.png',
  disabled: true
  }
  })
</script>
</html>
```

vue 还提供了指令 v-bind 的简写方式，可以直接通过:属性名的方式。

```
v-bind:src="src"  =  :src="src"
```

v-model 双向数据绑定

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <div id="app">
  <div>
  <label>年龄：</label>
  <input v-model="age"/>
  </div>
  <div>当前输入的年龄是：{{age}}</div>
  <button @click="add">加一岁</button>
  <button @click="alertYear">弹出年龄</button>
  </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
var vm = new Vue({
  el: '#app',
  data: {
  age: 10
  },
  methods: {
  add(){
  this.age = this.age + 1;
  },
```

```
31  alertYear() {
32  alert(this.age)
33  }
34  }
35 })
36 </script>
37 </html>
38
```

4.计算属性和侦听器

- **计算属性 VS 方法**

如果不使用计算属性，在 methods 里定义了一个方法，也可以实现相同的效果，甚至该方法还可以接受参数，使用起来更灵活。

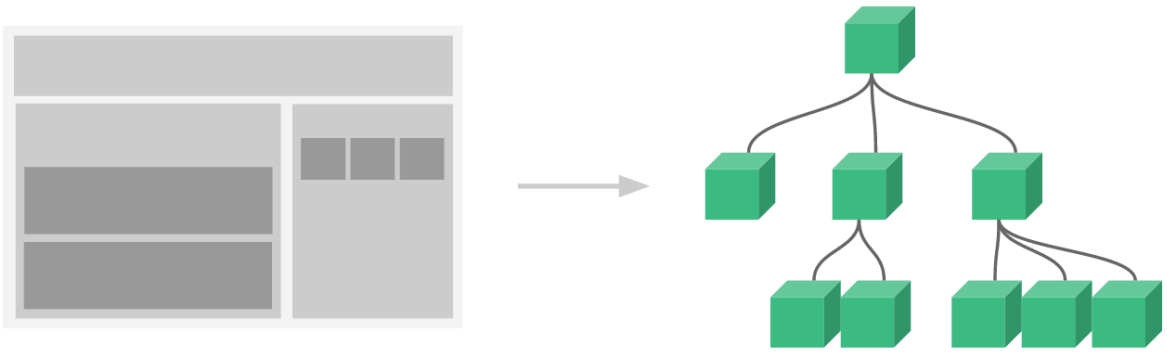
既然 methods 同样可以解决模板中复杂逻辑计算的问题，那么为什么还需要使用计算属性呢?

原因就是：计算属性是基于它的**依赖缓存**的。前面我们介绍过，计算属性的改变取决于其所依赖数据的变化，所以只要依赖数据不发生改变，计算属性就不会更新。当我们重复获取计算属性时它也不会重复计算，只会获取缓存的值。而我们每次调用 methods 都会重新计算一遍，这样将会消耗一部分性能。当然，如何你不希望对数据进行缓存，那么可以用方法来代替。

- **侦听器**
  - 通过侦听器来监听数据的变化，进行相应的逻辑处理。
  - 如何监听对象类型数据的某个属性进行侦听。

5.组件化基础

组件是 Vue.js 最强大的功能之一，组件可以扩展 HTML 元素，封装可重用的代码。组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树：



在编写组件时，我们需要不断思考如何提高组件的可复用性。

组件是可复用的 Vue 实例， 我们可以把页面中在多个场景中重复使用的部分，抽出为一个组件进行复用。组件大大提高了代码的复用率。

- **全局组件注册**。
  - 我们可以通过调用 Vue.component 的方式来定义全局组件，它接收两个参数：1. 组件名，2. 组件属性对象。

属性对象：组件的属性对象即为 Vue 的实例对象属性。

全局组件可以在任何其他组件内使用，所以当我们设计的组件，需要在不同地方使用的时候，我们应当注册全局组件。

```
1 // 注册
```

```
// 驼峰命名
Vue.component('MyComponentName', {/* */})
// 短横线命名
Vue.component('my-component-name', {/* */})
......
// 使用
<my-component-name></my-component-name>
// 也可以使用自闭和的方式
 <my-component-name />

```

具体示例如下：

```

<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <my-component></my-component>
 <my-component />
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
 Vue.component('myComponent', {
 template: '<div>Hello ! </div>'
 })
 var vm = new Vue({
 el: '#app',
 data() {
 return {}
 }
 })
</script>
</html>
```

代码解释：

JS 代码第 3-5 行，我们注册了一个全局组件 myComponent，并在 html 内使用两种方式引用了该组件。

- **局部组件注册**

我们也可以在 `Vue` 实例选项中注册局部组件，这样组件只能在这个实例中使用。局部组件的注册利用 `Vue` 实例的 `components` 对象属性，以组件名作为 `key` 值，以属性对象作为 `value`。由于局部组件只能在当前的 `Vue` 实例中使用，所以当我们设计的组件不需要在其他组件内复用时，可以设计为局部组件。

```
//注册
components: {
 'MyComponentName': {
 template: '<div>Hello ! </div>'
 }
}
......
```

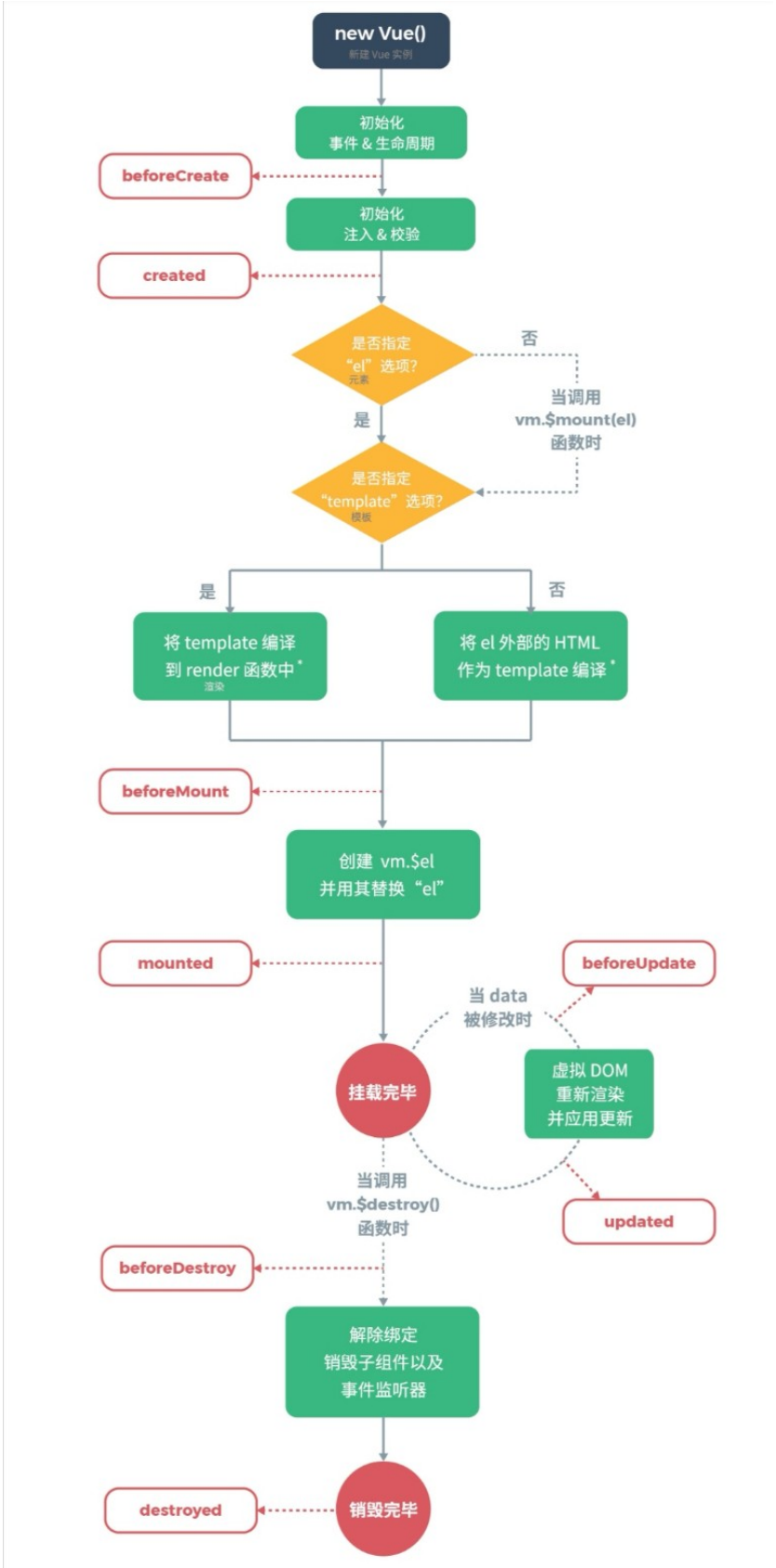
```
// 使用
<my-component-name></my-component-name>
 // 也可以使用自闭和的方式
 <my-component-name />
```

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <my-component></my-component>
 <my-component />
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
 var vm = new Vue({
 el: '#app',
 components: {
 'myComponent': {
 template: '<div>Hello ! </div>'
 }
 }
 })
</script>
</html>
```

6.生命周期和钩子函数

- **生命周期**

我们来看一下官网给的 Vue 生命周期的图：



从上面这幅图中，我们可以看到 `vue` 生命周期可以分为八个阶段，分别是：

- **beforeCreate（创建前）**
- **created（创建后）**
- **beforeMount (载入前)**
- **mounted（载入后）**

- **beforeUpdate（更新前）**
- **updated（更新后）**
- **beforeDestroy（销毁前）**
- **destroyed（销毁后）**

3.1 创建前（beforeCreate）

在实例初始化之后，此时的数据观察和事件机制都未形成，不能获得 DOM 节点。

3.2 创建后（created）

实例已经创建完成之后被调用。在这一步，实例已完成以下的配置：数据观测（data observer），属性和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始。

3.3 载入前（beforeMount）

在挂载开始之前被调用：这个过程是在模版已经在内存中编译完成， render 函数首次被调用，此时完成了虚拟 DOM 的构建，但并未被渲染。

3.4 载入后（mounted）

这个过程在模版挂载之后被调用，页面完成渲染，所以在这之后，我们可以操作和访问 DOM 元素。

3.5 更新前（beforeUpdate）

当数据更新时调用，在这一阶段 DOM 会和更改过的内容同步。

3.6 更新后（updated）

由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。
当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态，因为这可能会导致更新无限循环。

3.7 销毁前（beforeDestroy）

实例销毁之前调用。在这一步，实例仍然完全可用。

3.8 销毁后（destroyed）

Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。
实例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id = "app">
 {{ name }}
 <button @click="updateName">更新</button>
 <button @click="destroy">销毁</button>
 </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type = "text/javascript">
 var vm = new Vue({
 el: '#app',
 data: {
```

```
name:'hello !'
},
methods : {
updateName() {
console.log('准备修改名字啦！')
this.name = 'hello 慕课！'
},
destroy(){
vm.$destroy()
}
},
beforeCreate() {
// 此时页面数据未初始化
console.log('beforeCreate:' + this.name) // beforeCreate：  undefined
},
created() {
// 页面数据已经初始化
console.log('created:' + this.name) // beforeCreate：  hello !
},
beforeMount() {
console.log('beforeMount:' + this.name) // beforeCreate：  hello !
},
mounted() {
console.log('mounted:' + this.name) // beforeCreate：  hello !
},
// 点击更新按钮后会先触发 beforeUpdate
beforeUpdate() {
console.log('beforeUpdate:' + this.name) // beforeCreate：  hello 慕课!
},
updated() {
console.log('updated:' + this.name) // updated hello 慕课 !
},
// 点击销毁按钮后会先触发 beforeDestroy
beforeDestroy(){
console.log('beforeDestroy: before destroy') // beforeDestroy: before destroy
},
destroyed(){
console.log('destroyed: success') // destroyed: success
// 在这之后点击页面 更新 按钮将无任何效果
}
});
</script>
</html>
```

- **VueRouter**
  - Vue Router 是 Vue.js 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。 — 官方定义

    VueRouter 是 SPA（单页应用）的路径管理器，它允许我们通过不同的 URL 访问不同的内容。

```
npm install vue-router
```

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="app">
 <div>
 <router-link to="/index">首页</router-link>
 <router-link to="/article">文章</router-link>
 </div>
 <router-view></router-view>
 </div>
</body>

<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
<script type="text/javascript">

const Index = Vue.component('index', {
 template: '<div>Hello，欢迎使用慕课网学习 Vue 教程! </div>',
})

const Article = Vue.component('myArticle', {
 template: `<ul><li>1. Vue 计算属性的学习</li><li>2. React 基础学习</li></ul>`,
})

const routes = [
 { path: '/index', component: Index },
 { path: '/article', component: Article }
]

const router = new VueRouter({
 routes: routes
})

var vm = new Vue({
 el: '#app',
 router: router,
 data() {
 return {}
 }
})
</script>
</html>
```

- **axios**
  - 官方：http://www.axios-js.com/zh-cn/docs/

Axios 是一个基于 promise 的 HTTP 库，可以用在html和 vue、nodejs 中。

特性

- 从浏览器中创建 XMLHttpRequests
- 从 node.js 创建 http 请求
- 支持 Promise API
- 拦截请求和响应
- 转换请求数据和响应数据
- 取消请求
- 自动转换 JSON 数据
- 客户端支持防御 XSRF

安装

使用 npm:

```
$ npm install axios
```

使用 cdn:

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

执行 `GET` 请求

```
// 上面的请求也可以这样做
axios.get('/user', {
 params: {
 ID: 12345
 }
 })
 .then(function (response) {
 console.log(response);
 })
  .catch(function (error) {
  console.log(error);
  });
```

执行 `POST` 请求

```
axios.post('/user', {
 firstName: 'Fred',
 lastName: 'Flintstone'
 })
 .then(function (response) {
 console.log(response);
 })
 .catch(function (error) {
 console.log(error);
  });
```

执行多个并发请求

```
function getUserAccount() {
 return axios.get('/user/12345');
}

function getUserPermissions() {
 return axios.get('/user/12345/permissions');
}

axios.all([getUserAccount(), getUserPermissions()])
```

```
  .then(axios.spread(function (acct, perms) {
  // 两个请求现在都执行完成
  }));
```

7.使用Vue脚手架进行模块化开发

1.安装vue-cli

```
npm install -g @vue/cli
```

2. 安装初始化工具

拉取 2.x 模板 (旧版本)

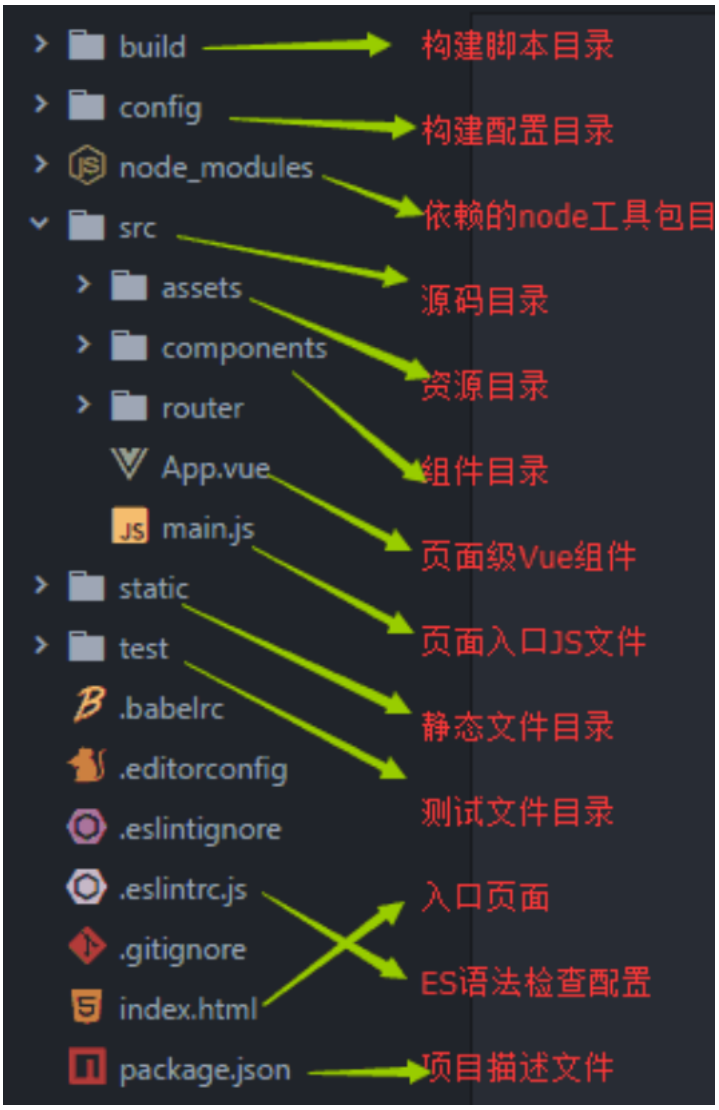
Vue CLI >= 3 和旧版使用了相同的 `vue` 命令，所以 Vue CLI 2 (`vue-cli`) 被覆盖了。如果你仍然需要使用旧版本的 `vue init` 功能，你可以全局安装一个桥接工具：

```
npm install -g @vue/cli-init
```

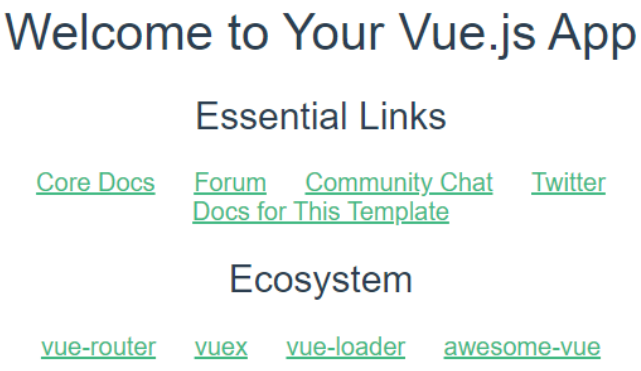
3. 创建项目

```
vue init webpack 项目名
// 确认项目名
// 输入项目描述
// 作者
// 让选择创建方式 默认 回车
// 选择是否安装vue-router y
// 是否安装eslint n
// unit test n
// e2e test n
// 选择NPM
进行安装....
```

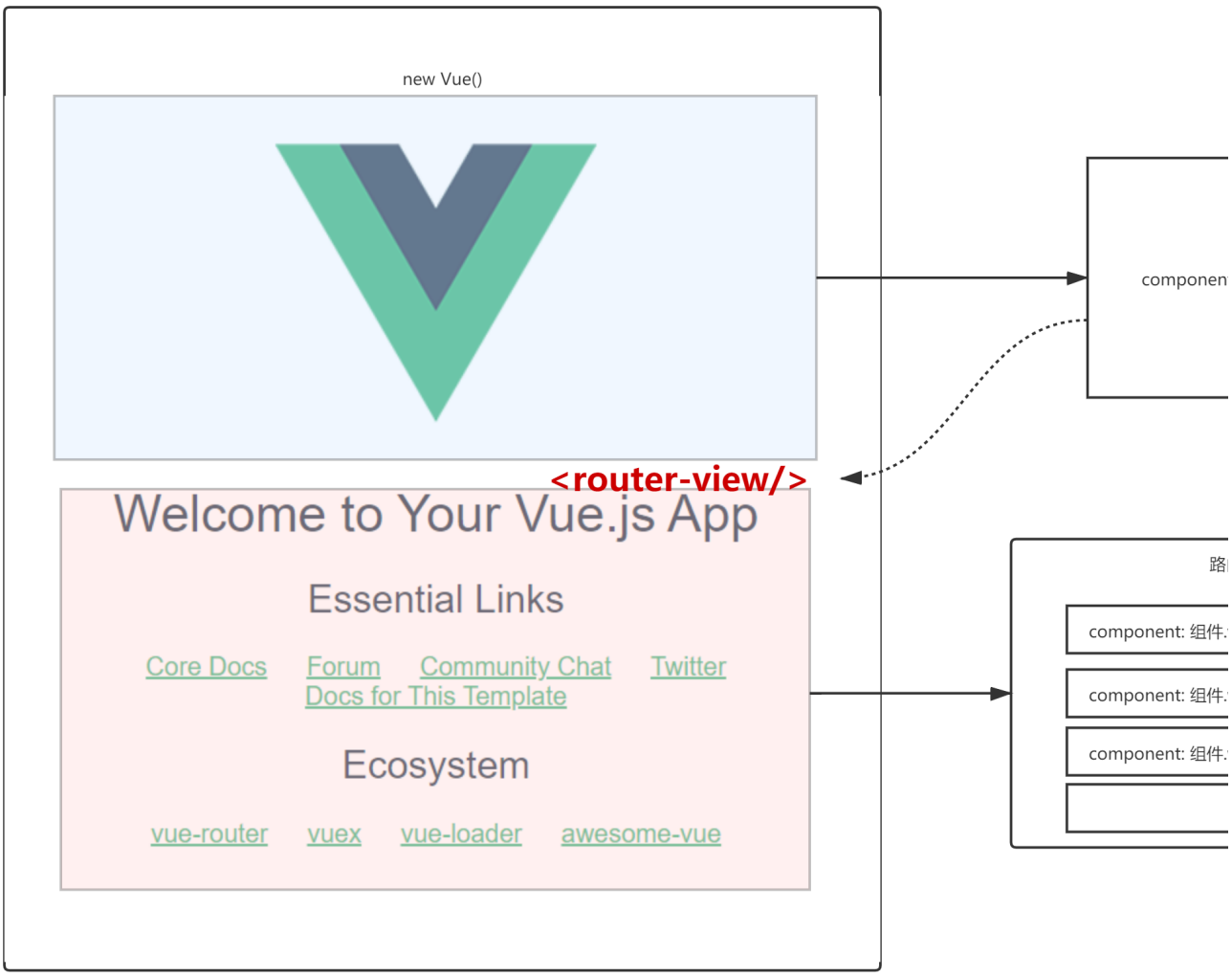
在vscode中打开



HelloWorld.vue



代码图意：


new Vue()
componen
<router-view/>
Welcome to Your Vue.js App
Essential Links
Core Docs
Forum
Community Chat
Twitter
Docs for This Template
Ecosystem
vue-router
vuex
vue-loader
awesome-vue
路
component: 组件.
component: 组件.
component: 组件.