认识Vue

1.1. Vue 简介

1.1.1. 官网

1. 英文官网: https://vuejs.org/
2. 中文官网: https://cn.vuejs.org/

1.1.2. 介绍与描述

1. 动态构建用户界面的渐进式 JavaScript 框架
2. 作者: 尤雨溪

1.1.3. Vue 的特点

1. 遵循 MVVM 模式
2. 编码简洁, 体积小, 运行效率高, 适合移动/PC 端开发
3. 它本身只关注 UI, 也可以引入其它第三方库开发项目

1.1.4. 与其它 JS 框架的关联

1. 借鉴 Angular 的模板和数据绑定技术
2. 借鉴 React 的组件化和虚拟 DOM 技术

1.1.5. Vue 周边库

1. vue-cli: vue 脚手架
2. vue-resource
3. axios
4. vue-router: 路由
5. vuex: 状态管理
6. element-ui: 基于 vue

Vue3简介

- 2020年9月18日，Vue.js发布3.0版本，代号：One Piece（海贼王）
- 耗时2年多、2600+次提交、30+个RFC、600+次PR、99位贡献者
- github上的tags地址：https://github.com/vuejs/vue-next/releases/tag/v3.0.0
- 截至发稿，vue版本已更新到 3.2.6

1.性能的提升

- 打包大小减少41%
- 初次渲染快55%, 更新渲染快133%
- 内存减少54%

  ……

2.源码的升级

- 使用Proxy代替defineProperty实现响应式
- 重写虚拟DOM的实现和Tree-Shaking

  ……

3.拥抱TypeScript

- Vue3可以更好的支持TypeScript

4.新的特性

1. Composition API（组合API）
   - setup配置
   - ref与reactive
   - watch与watchEffect
   - provide与inject
   - ……
2. 新的内置组件
   - Fragment
   - Teleport
   - Suspense
3. 其他改变
   - 新的生命周期钩子
   - data 选项应始终被声明为一个函数
   - 移除keyCode支持作为 v-on 的修饰符
   - ……

Vue快速上手

现阶段公司项目中主要使用Vue2，但是Vue3是趋势，现在的Vue3生态已经在完善中，很多插件库都有Vue3版本。因此直接学习Vue3即可。

详细的Vue3资料可以在 官网找到

Vue.js (vuejs.org)

安装Vue

将 Vue.js 添加到项目中主要有四种方式：

1. 在页面上以 CDN 包的形式导入。
2. 下载 JavaScript 文件并自行托管。
3. 使用 npm 安装它。
4. 使用官方的 CLI 来构建一个项目，它为现代前端工作流程提供了功能齐备的构建设置 (例如，热重载、保存时的提示等等)。

一般我们练习选择第一或第二中方式，后面做项目，我们再切换到第四种方式。

1.在页面上以 CDN 包的形式导入

```
<script src="https://unpkg.com/vue@next"></script>    #指定最新版本
```

或者

```
<script src="https://unpkg.com/vue@3.2.6"></script>    #指定版本
```

2.下载 JavaScript 文件并自行托管。

进入 https://unpkg.com/vue@3.2.6/dist/vue.global.prod.js

复制内容,保存到vue.js

```
<script src="./vue.js"></script>
```

注意文件的相对路径!

第一个Vue应用

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

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>test1</title>
</head>

<body>
  <div id="root">
    <h1>
      {{name}}
      {{age}}
    </h1>
  </div>

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

    //创建vue对象
    const app = Vue.createApp({
      data(){  // data函数返回模板用到的变量,
        name='hello',
        age=18
        return { // 必须以对象形式返回
          name,
          age
        }
      }
    })
    app.mount('#root')
  </script>
</body>

</html>
```

以上代码保存到HTML文件中，以服务器环境打开（这里必须以服务器环境打开，否则代码是无法工作的）

vscode可以安装live server插件，打开很方便。

模板的理解

上面的代码HTML部分写死的内容决定了整体页面的框架与样式，但是局部地区：如h1标签内容需要动态更改。

这里用{{}}包裹的部分可以放js表达式，用于动态显示其内容。

模板中的语法有两类

1. 插值语法（双大括号表达式）
2. 指令（以 v-开头）

插值语法

1. 功能: 用于解析标签体内容
2. 语法: {{xxx}} ，xxxx 会作为 js 表达式运行

指令语法

1. 功能: 解析标签属性、解析标签体内容、绑定事件
2. 举例：v-bind:href = 'xxxx' ，xxxx 会作为 js 表达式被解析
3. 说明：Vue 中有有很多的指令，此处只是用 v-bind 举个例子

例：

```
<body>
  <div id="root">
    <a v-bind:href="link">点我跳转到xxx</a>
  </div>

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

    //创建vue对象
    const app = Vue.createApp({
      data(){
        name='hello'
        age=18
        link = 'http://baidu.com'
        return {
          name,
          age,
          link
        }
      }
    })
    app.mount('#root')
  </script>
</body>
```

href里的link会被解析成变量值，从而可以点击访问到链接。

篇幅关系，以上代码案例只显示了body中的内容，包括html和js部分，后续也都相同

Vue常用指令

接下来的Vue指令按照用途进行划分，写法都是v-指令这种形式，个别的存在简写，后面再介绍。

数据绑定

单向数据绑定

1. 语法：v-bind:href ="xxx" 或简写为 :href
2. 特点：数据只能从 js 流向html

双向数据绑定

1. 语法： v-model="xxx"
2. 特点：数据不仅能从 js流向html，还能从html流向 js

案例：

```
<body>
  <div id="app">
    <input type="text" v-bind:value='name'>
    <input type="text" v-model='age'>
    <h3>name:{{name}}</h3>
    <h3>age:{{age}}</h3>
  </div>
  <script src="../vue.js"></script>
  <script>
    const app = Vue.createApp({
      data(){
        let name ='xiaoming'
        let age = 12
        return{
          name,
          age,
        }
      }
    })

   const vm = app.mount('#app')
  </script>
</body>
```


修饰符

`.lazy`

在默认情况下，`v-model` 在每次 `input` 事件触发后将输入框的值与数据进行同步 (除了上述输入法组织文字时)。你可以添加 `lazy` 修饰符，从而转为在 `change` 事件*之后*进行同步：

```
<!-- 在“change”时而非“input”时更新 -->
<input v-model.lazy="msg" />
```

比如注册时，你输入好用户名后，系统会向后台发起请求查看你输入的名称是否已经被人注册了，是就提示你，所以这个发请求不能你每输入一个字符就查询一次，应该输完了再请求，这样不会给后端造成太大压力。

`.number`

如果想自动将用户的输入值转为数值类型，可以给 `v-model` 添加 `number` 修饰符：

```
<input v-model.number="age" type="number" />
```

这通常很有用，因为即使在 `type="number"` 时，HTML 输入元素的值也总会返回字符串。如果这个值无法被 `parseFloat()` 解析，则会返回原始的值。

`.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：

```
<input v-model.trim="msg" />
```

事件监听

我们可以使用 `v-on` 指令 (通常缩写为 `@` 符号) 来监听 DOM 事件，并在触发事件时执行一些 JavaScript。用法为 `v-on:事件="方法名"` 或使用快捷方式 `@事件="方法名"`

例如：

```
<body>
  <div id="app">
    <input type="text" v-bind:value='name'>
    <input type="text" v-model.lazy='age'>
    <h3>name:{{name}}</h3>
    <h3>age:{{age}}</h3>
    <button @click='age++'>点我加1岁</button>
  </div>
  <script src="../vue.js"></script>
  <script>
    const app = Vue.createApp({
      data(){ //数据定义在这里
        let name ='xiaoming'
        let age = 12
        return{
          name,
          age,
        }
      },
      methods:{//方法定义在这里
        add(){
          this.age++
        }
      }
    })

   const vm = app.mount('#app')
  </script>
</body>
```

事件参数

默认情况下事件形参：event，即传递的函数没有括号，但是这里可以接收形参,比如：

```
<body>
  <div id="app">
    <input type="text" v-bind:value='name'>
    <input type="text" v-model.lazy='age'>
    <h3>name:{{name}}</h3>
    <h3>age:{{age}}</h3>
    <button @click='add'>点我加1岁</button>
  </div>
  <script src="../vue.js"></script>
  <script>
    const app = Vue.createApp({
      data(){
        let name ='xiaoming'
        let age = 12
        return{
          name,
          age,
        }
      },
      methods:{
        add(event){
          console.log(event.target.tagName)
          this.age++
        }
      }
    })

   const vm = app.mount('#app')
  </script>
</body>
```

@click='add' 这里调用时，隐式传递了event，add在接收了event之后就可以调用事件对象方法。

无需事件形参

如果函数内部不需要操作事件对象，或者只接收普通参数，那么可以在调用时传入形参，

```
<body>
  <div id="app">
    <input type="text" v-bind:value='name'>
    <input type="text" v-model.lazy='age'>
    <h3>name:{{name}}</h3>
    <h3>age:{{age}}</h3>
    <button @click='add(10)'>点我加10岁</button>
  </div>
  <script src="../vue.js"></script>
  <script>
    const app = Vue.createApp({
      data(){
        let name ='xiaoming'
        let age = 12
        return{
          name,
```

```
          age,
        }
      },
      methods:{
        add(num){
          this.age+=num
        }
      }
    })

   const vm = app.mount('#app')
  </script>
```

同时需要事件形参和普通参数

通过$event表示事件对象，传给函数

```
<body>
  <div id="app">
    <input type="text" v-bind:value='name'>
    <input type="text" v-model.lazy='age'>
    <h3>name:{{name}}</h3>
    <h3>age:{{age}}</h3>
    <button @click='add(10,$event)'>点我加10岁</button>
  </div>
  <script src="../vue.js"></script>
  <script>
    const app = Vue.createApp({
      data(){
        let name ='xiaoming'
        let age = 12
        return{
          name,
          age,
        }
      },
      methods:{
        add(num,event){
          console.log(event.target.tagName)
          this.age+=num
        }
      }
    })

   const vm = app.mount('#app')
  </script>
```

其他事件

除了监听click事件，还可以监听其他事件，如键盘事件@keyup.enter 表示监听回车键，enter属于键盘别名

Vue 为最常用的键提供了别名：

- `.enter`
- `.tab`
- `.delete` (捕获“删除”和“退格”键)

- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`

条件渲染

v-if

我们可以用v-if来控制哪些元素显示，哪些不显示。

```
<body>
  <div id="app">
    <input type="text" v-bind:value='name'>
    <input type="text" v-model.lazy='age'>
    <h3>name:{{name}}</h3>
    <h3>age:{{age}}</h3>
    <button @click='add(10,$event)'>点我加10岁</button>
    <h3 v-if='eat'>吃饭了</h3>
    <button @click="eat=!eat">开饭/消化</button>
  </div>
  <script src="../vue.js"></script>
  <script>
    const app = Vue.createApp({
      data(){
        let name ='xiaoming'
        let age = 12
        let eat = false
        return{
          name,
          age,
          eat,
        }
      },
      methods:{
        add(num,event){
          console.log(event.target.tagName)
          this.age+=num
        }
      }
    })
   const vm = app.mount('#app')
  </script>
</body>
```

同时可以配合v-else实现条件分支

```
    <h3 v-if='eat'>吃饭了</h3>
    <h3 v-else>没吃</h3>
    <button @click="eat=!eat">开饭/消化</button>
```

反复点击按钮观察效果。

`v-else-if`，顾名思义，充当 `v-if` 的“else-if 块”，并且可以连续使用：

与 `v-else` 的用法类似，`v-else-if` 也必须紧跟在带 `v-if` 或者 `v-else-if` 的元素之后。

v-show

用法与v-if相同，区别是v-show不会销毁元素，只是把元素的display改成none从而不显示，相比较v-if性能开销小，再元素频繁切换时建议使用v-show

列表渲染

我们可以用 `v-for` 指令基于一个数组来渲染一个列表。`v-for` 指令需要使用 `item in items` 形式的特殊语法，其中 items 是源数据数组，而 `item` 则是被迭代的数组元素的**别名**。

```
<body>
  <div id="app">
    <input type="text" v-bind:value='name'>
    <input type="text" v-model.lazy='age'>
    <h3>name:{{name}}</h3>
    <h3>age:{{age}}</h3>
    <button @click='add(10,$event)'>点我加10岁</button>
    <h3 v-show='eat'>吃饭了</h3>
    <button @click="eat=!eat">开饭/消化</button>
    <h3>周末安排</h3>
    <ul>
      <li v-for="task in tasks">{{task}}</li>
    </ul>
  </div>
  <script src="../vue.js"></script>
  <script>
    const app = Vue.createApp({
      data(){
        let name ='xiaoming'
        let age = 12
        let eat = false
        const tasks = ['吃饭','睡觉','打豆豆']
        return{
          name,
          age,
          eat,
          tasks
        }
      },
      methods:{
        add(num,event){
          console.log(event.target.tagName)
          this.age+=num
        }
      }
    })
   const vm = app.mount('#app')
  </script>
</body>
```