读音：[vju:]，view

1. 了解VUE

- Vue 是什么
  - 主流的渐进式 JavaScript 框架
- 什么是渐近式
  - 可以和传统的网站开发架构融合在一起，例如可以简单的把它当作一个类似 JQuery 库来使用。
  - 也可以使用Vue全家桶框架来开发大型的单页面应用程序 。
- 使用它的原因
  - vue.js 体积小，编码简洁优雅，运行效率高，用户体验好.
  - 无Dom操作，它能提高网站应用程序的开发效率,页面性能较好
- 什么场景下使用它
  - 一般是需要开发单页面应用程序（ Single Page Application, 简称:SPA ）的时候去用
  - 单页面应用程序，如：网易云音乐 https://music.163.com/
  - 因为 Vue 是 渐进式 的，Vue 其实可以融入到不同的项目中，即插即用

1.1 前端框架历史

- jquery阶段（2006-2020）
  - 特点：节点操作简单易用，浏览器兼容，提供很多api，拿来即用
    - 查找节点
    - dom操作
    - ajax
    - 运动
    - 插件封装
- angular阶段（2009-2014） ng-
  - 特点：MVC模式，双向数据绑定，依赖注入

    - 2009 年诞生的，起源于个人开发，后来被 Google 收购了
- react阶段（2013）
    - 特点：virtual DOM(虚拟节点)、组件化，性能上碾轧angularJS
    - 2013年5月开源的，起源于 Facebook 的内部项目，对市场上所有 JS 框架都不满意，于是自己写了一套
- vue阶段（2014-2016）v-
    - 特点：综合angular与react的优点，MVVM模式，是一款高性能高效率的框架：双向数据绑定（MVVM）、组件化、虚拟节点
    - Vue 不支持 IE8 及以下版本，因为 Vue 使用了 IE8 无法模拟 ECMAScript 5 特性。推荐使用最新谷歌浏览器。
    - 使用情况：
        - BAT 级别的企业：React 最多 > Angular > Vue.js
        - 中小型公司：Vue.js 更多一些，有中文文档学习成本低

1.2 学习资源

- 英文官网：https://vuejs.org/
- 中文官网（中文文档很友好）：https://cn.vuejs.org/
- 官方教程：https://cn.vuejs.org/v2/guide/
- GitHub：https://github.com/yyx990803
- API文档：https://cn.vuejs.org/v2/api/

不建议买书，官方文档很详细，多查官方文档，因为很多书基本上都是直接抄官方文档的

1.3 发展历史

- 作者：尤雨溪（微博：尤小右），一位华裔前 Google 工程师，江苏无锡人。
    - 个人博客：http://www.evanyou.me/
    - 新浪微博：http://weibo.com/arttechdesign
    - 知乎：https://www.zhihu.com/people/evanyou/activities
- 2013年12月8号在 GitHub 上发布了 0.6 版
- 2015年10月份正式发布了 1.0 版本，开始真正的火起来

- 2016年10月份正式发布了 2.0 版
- 2019.4.8号发布了 Vue 2.5.10 版本 https://github.com/vuejs/vue/releases
- 1.x 版本老项目可能还在用，新项目绝对都是选择 2.x

2. 安装和引入

- 开发环境：development（开发环境，未压缩）
- 生产环境：production（上线，压缩）
- script标签
- cdn

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.17/dist/vue.js"></script>
```

- npm

能很好和webpack等打包工具配合使用，命令行窗口，安装2.6.10版本的 vue 模块

```
npm install vue@2.6.10
```

- vue-cli脚手架

快速的搭建基于webpack的开发环境

3. Vue 核心技术

3.1 v-model的运用

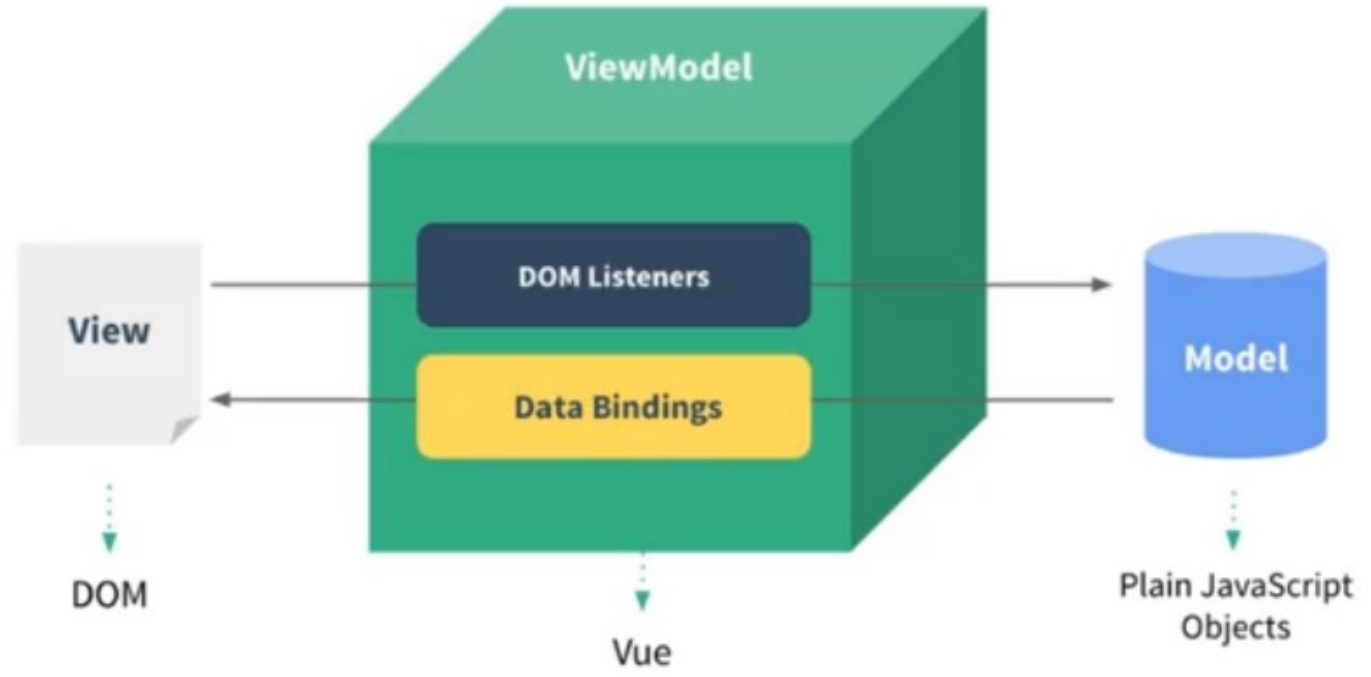
```
<body>
    <div id="app">
        <!-- {{}}用于标签体内显示数据 -->
        Hello, {{ content }} <br />
        Hello2, {{ content }} <br />
        <h3 v-text="content"></h3>
        <!-- v-model进行数据的双向绑定 -->
        <input type="text" v-model="content">
    </div>
```

```
    <div id="app2"></div>

    <script src="./node_modules/vue/dist/vue.js"></script>
    <script>
        var vm = new Vue({
            el: '#app', //指定被Vue管理的入口，值为选择器，不可以指定body或者是html
            data: { // 用于初始化数据，在Vue实例管理的Dom节点下，可通过模板语法来引用
                content: 'Vue.js'
            }
        })
    </script>
</body>
```

3.2 分析 MVVM 模型

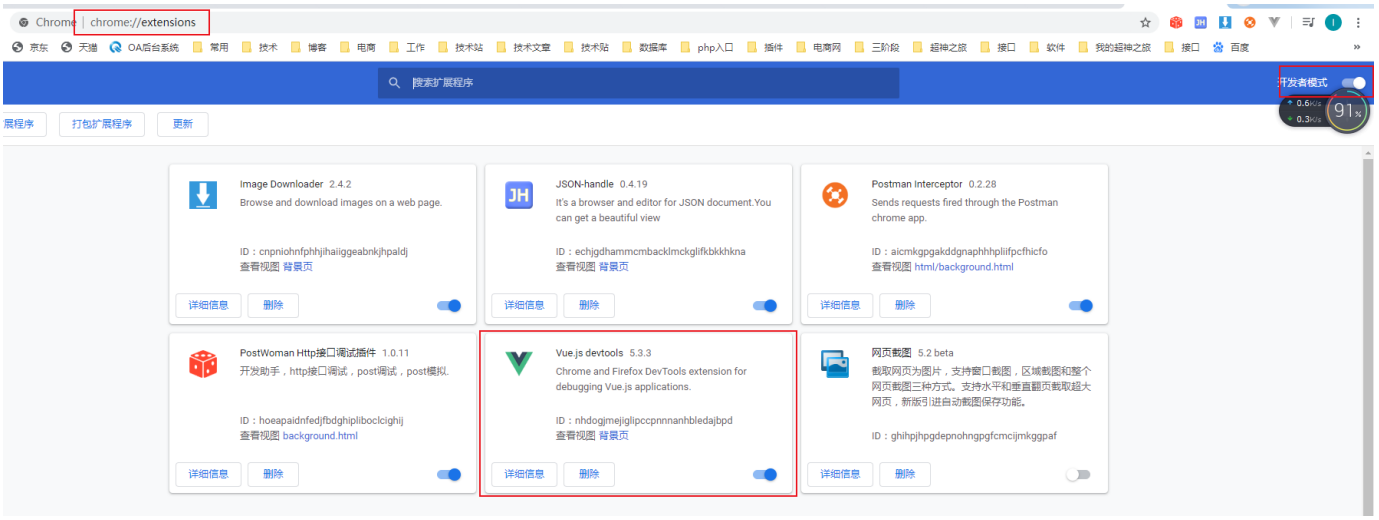
- 常见面试题：什么是 MVVM 模型？ MVVM 是 Model-View-ViewModel 的缩写，它是一种软件架构风格
  - Model：模型， 数据对象（data选项当中 的）
  - View：视图，模板页面（用于渲染数据）
  - ViewModel：视图模型，其实本质上就是 Vue 实例
- 核心思想
  - 通过数据驱动视图 把需要改变视图的数据初始化到 Vue中，然后再通过修改 Vue中的数据，从而实现对视图的更新
- 声明式编程
  - 按照 Vue 的特定语法进行声明开发，就可以实现对应功能，不需要我们直接操作Dom元素
- 命令式编程
  - Jquery它就是，需要手动去操作Dom才能实现对应功能



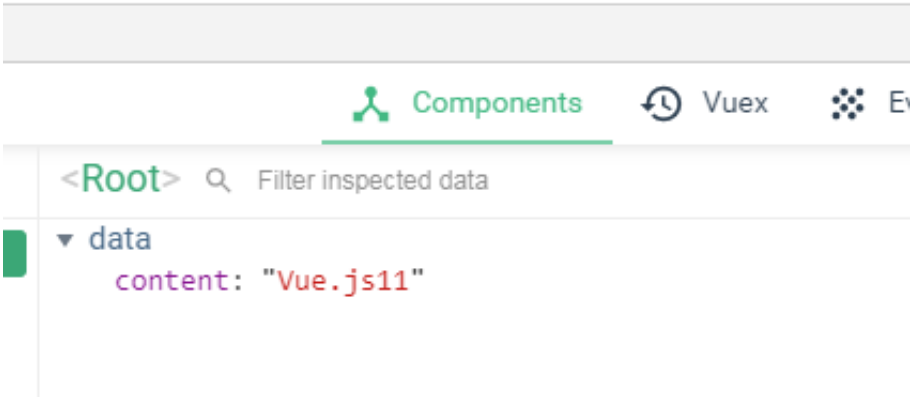
3.3 Vue Devtools 插件安装

Vue Devtools 插件让我们在一个更友好的界面中审查和调试 Vue 项目。

- 谷歌浏览器访问： chrome://extensions ，然后右上角打开 开发者模式 ， 打开的效果如下



- 将软件直接拖到上面页面空白处，会自动安装
- 当你访问Vue开发的页面时，按 F12 可 Vue 标签页



3.4 模板数据绑定渲染

3.4.1 双大括号语法 {{}}

- 格式： {{表达式}}
- 作用：
  - 使用在标签体中，用于获取数据
  - 可以使用 JavaScript 表达式

3.4.2 一次性插值 v-once

- 通过使用 v-once 指令，你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。

3.4.3 输出HTML指令 v-html

- 格式： v-html='xxxx'
- 作用： 如果是HTML格式数据，双大括号会将数据解释为普通文本，为了输出真正的 HTML，你需要使用 vhtml 指令

3.4.4 输出文本指令 v-text

- 格式： v-text='xxxx'
- 作用：渲染数据到页面，但是不会渲染标签，当做文本处理

3.4.5 解决{{}}闪现问题 v-clock

- 格式： v-text
- 作用：为了解决{{}}的闪现问题

```
<style>
        [v-cloak]{
            display: none
        }
    </style>
<body>
    <!--
        {{JS表达式}}
     -->
    <div id="app">
        <h3>1、{{}}双大括号输出文本内容</h3>
        <!-- 文本内容 -->
        <p>{{message}}</p>
        <!-- JS表达式 -->
        <p>{{score + 1}}</p>

        <h3>2、 一次性插值 v-once </h3>
        <span v-once>{{message}}</span>

        <h3>3、指令输出真正的 HTML 内容 v-html</h3>
        <p>双大括号：{{contentHtml}}</p>
        <!--
             v-html:
             1. 如果输出的内容是HTML数据，双大括号将数据以普通文本方式进行输出，
             为了输出真正ＨＴＭＬ的效果，就需要使用v-html 指定


         -->
        <p>v-html: <span v-html="contentHtml"></span></p>
        <h3>4、指令输出内容 v-text</h3>
        <span v-text="contentHtml"></span>

        <h3>5、指令解决问题v-clock</h3>
        <p v-clock>{{message}}</p>
    </div>
    <script src="./node_modules/vue/dist/vue.js"></script>
    <script>
        var vm = new Vue({
            el: '#app',
            data: {
```

```
                message: '钢铁侠',
                score: 100,
                contentHtml: `<span style="color:red">此内容为红色字体
                    alert('hello world')
                    </span>`
            }
        })
    </script>
</body>
```

3.5 元素属性绑定指令 v-bind

- 完整格式： v-bind:元素的属性名='xxxx'
- 缩写格式： :元素的属性名='xxxx'
- 作用：将数据动态绑定到指定的元素上

3.6 事件绑定指令 v-on

- 完整格式： v-on:事件名称="事件处理函数名"
- 缩写格式： @事件名称="事件处理函数名" 注意： @ 后面没有冒号
- 作用：用于监听 DOM 事件

```
<body>
   <div id="app">
        <h3>1、元素绑定指令 v-bind</h3>
        <img v-bind:src="imgUrl">
        <img :src="imgUrl">
         <a :href="mxgUrl">跳转</a>

         <h3>2、事件绑定指令 v-on</h3>
         <input type="text" value="1" v-model="num">
         <button @click="add">点击+1</button>
     </div>
     <script src="./node_modules/vue/dist/vue.js"></script>
     <script>
         var vm = new Vue({
            el: '#app',
```

```
            data: {
                imgUrl: 'https://cn.vuejs.org/images/logo.png',
                mxgUrl: 'https://www.baidu.com',
                num: 10
            },
            methods: { // 指定事件处理函数 v-on:事件名="函数名" 来进行调用
                add: function() { //定义了add函数
                    console.log('add被调用')
                    this.num ++
                }
            }
        })
    </script>
</body>
```

3.7 计算属性和监听器

3.7.1 计算属性 computed

- computed 选项定义计算属性
- 计算属性 类似于 methods 选项中定义的函数
  - 计算属性 会进行缓存，只在相关响应式依赖发生改变时它们才会重新求值。
  - 函数 每次都会执行函数体进行计算
  - computed 选项内的计算属性默认是 getter 函数,不过在需要时你也可以提供一个 setter

3.7.2 监听器 watch

- 当属性数据发生变化时,对应属性的回调函数会自动调用, 在函数内部进行计算
- 通过 watch 选项 或者 vm 实例的 $watch() 来监听指定的属性

```
<body>
    <div id="app">
        数学：<input type="text" v-model="mathScore">
        英语：<input type="text" v-model="englishScore"><br>
        <!-- v-model调用函数时，不要少了小括号 -->
```

```
        总得分（函数-单向绑定）：<input type="text" v-model="sumScore()"><br>
        <!-- 绑定计算属性后面不加上小括号 -->
        总得分（计算属性-单向绑定）：<input type="text" v-model="sumScore1"><br>
        总得分（计算属性-双向绑定）：<input type="text" v-model="sumScore2">

        <!-- 通过 watch 选项 监听数学分数， 当数学更新后回调函数中重新计算总分sumSco
        总得分（监听器）：<input type="text" v-model="sumScore3">

    </div>
    <script src="./node_modules/vue/dist/vue.js"></script>
    <script>
        /*
        1. 函数没有缓存，每次都会被调用
        2. 计算属性有缓存 ，只有当计算属性体内的属性值被更改之后才会被调用，不然不会被
        3. 函数只支持单向
        4. 计算属性默认情况下：只有getter函数，而没有setter函数，所以只支持单向
            如果你要进行双向，则需要自定义setter函数
        */
        var vm = new Vue({
            el: '#app',
            data: {
                mathScore: 80,
                englishScore: 90,
                sumScore3: 0 // 通过监听器，计算出来的总得分
            },

            methods: {
                sumScore: function () { //100
                    console.log('sumScore函数被调用')
                    // this 指向的就是 vm 实例 , 减0是为了字符串转为数字运算
                    return (this.mathScore - 0) + (this.englishScore - 0)
                }
            },

            computed: { //定义计算属性选项
                //这个是单向绑定，默认只有getter方法
                sumScore1: function () { //计算属性有缓存，如果计算属性体内的属性值
                    console.log('sumScore1计算属性被调用')
                    return (this.mathScore - 0) + (this.englishScore - 0)
                },
```

```
46
47              sumScore2: { //有了setter和getter之后就可以进行双向绑定
48                  //获取数据
49                  get: function () {
50                      console.log('sumScore2.get被调用')
51                      return (this.mathScore - 0) + (this.englishScore - 0)
52                  },
53                  //设置数据，  当 sumScore2 计算属性更新之后 ，则会调用set方法
54                  set: function (newValue) { // newVulue 是 sumScore2 更新之后
55                      console.log('sumScore2.set被调用')
56                      var avgScore = newValue / 2
57                      this.mathScore = avgScore
58                      this.englishScore = avgScore
59                  }
60              }
61          },
62
63          //监听器,
64          watch: {
65              //需求：通过 watch 选项 监听数学分数，  当数学更新后回调函数中重新计算
66              mathScore: function (newValue, oldValue) {
67                  console.log('watch监听器,监听到了数学分数已经更新')
68                  //  newValue 是更新后的值，oldValue更新之前的值
69                  this.sumScore3 = (newValue - 0) + (this.englishScore - 0)
70              }
71          },
72      })
73
74      //监听器方式2: 通过 vm 实例进行调用
75      //第1个参数是被监听 的属性名，  第2个是回调函数
76      vm.$watch('englishScore', function (newValue) {
77          //newValue就是更新之后的英语分数
78          this.sumScore3 = (newValue - 0) + (this.mathScore - 0)
79      })
80
81      vm.$watch('sumScore3', function (newValue) {
82          //newValue就是更新之后部分
83          var avgScore = newValue / 2
84          this.mathScore = avgScore
85          this.englishScore = avgScore
```

```
        })
    </script>
</body>
```

3.8 Class 与 Style 绑定 v-bind

- 通过 class 列表和 style 指定样式是数据绑定的一个常见需求。它们都是元素的属性，都用 v-bind 处理，其中表达式结果的类型可以是：字符串、对象或数组
- 语法格式
  - v-bind:class='表达式' 或 :class='表达式
  - class 的表达式可以为：
    - 字符串 :class="activeClass"
    - 对象 :class="{active: isActive, error: hasError}"
    - 数组 :class="['active', 'error']" 注意要加上单引号,不然是获取data中的值
    - v-bind:style='表达式' 或 :style='表达式`
  - style 的表达式一般为对象
    - :style="{color: activeColor, fontSize: fontSize + 'px'}"
    - 注意：对象中的value值 activeColor 和 fontSize 是data中的属性

```
<style>
        .active {
            color: green;
        }
        .delete {
            background: red;
        }
        .error {
            font-size: 35px;
        }
    </style>
</head>
<body>
    <div id="app">
        <h3>Class绑定，v-bind:class 或 :class </h3>
```

```
        <!-- <p class="active">字符串表达式</p> -->
        <p v-bind:class="activeClass">字符串表达式</p>
        <!-- key值是样式 名，value值是data中绑定的属性
        当isDelete为true的时候，delete就会进行渲染
        -->
        <p :class="{delete: isDelete, error: hasError}">对象表达式</p>

        <p :class="['active', 'error']">数组表达式</p>

        <h3>Style绑定，v-bind:style 或 :style</h3>
        <p :style="{color: activeColor, fontSize: fontSize + 'px'}">Style绑定</p

    </div>
    <script src="./node_modules/vue/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                activeClass: 'active',
                isDelete: false,
                hasError: true,
                activeColor: 'red',
                fontSize: 100
            }
        })
    </script>
</body>
```

3.9 条件渲染 v-if

3.9.1 条件指令

- v-if 是否渲染当前元素
- v-else
- v-else-if
- v-show 与 v-if 类似，只是元素始终会被渲染并保留在 DOM 中,只是简单切换元素的 CSS 属性 display 来显

- 示或隐藏

```
<!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>v-if的使用</title>
    <style>
        .con {
            width: 200px;
            height: 200px;
            background: red;
            margin-bottom: 10px;
        }

        .green {
            width: 200px;
            height: 200px;
            background: green;
            margin-bottom: 10px;
        }
    </style>
</head>

<body>
    <div id="app">
        <input type="button" value="下拉菜单1" @click="change()">
        <!-- v-if是创建和删除元素  v-if和v-else中间不要隔开否则报错 -->
        <div class="con" v-if="isok1"></div>
        <div class="green" v-else></div><br>

        <input type="button" value="下拉菜单2" @click="change2()">
        <!-- v-show显示和隐藏 -->
        <div class="con" v-show="isok2"></div>
    </div>
</body>
```

```
<script src="../node_modules/vue/dist/vue.js"></script>
<script>
    /*
        v-if:创建和删除元素
        v-show:显示和隐藏元素
        结论：如果需要频频的显示隐藏元素，建议v-show性能更好
    */


    //实例化app对象
    let app = new Vue({
        el: '#app',//el 放挂载对象,里面写的是选择器,不能挂载在html和body节点上
        data: {//放数据的地方
            isok1: false,
            isok2: false
        },
        methods: {//methods存放方法的地方
            change() {
                console.log('点击按钮1');
                console.log(this.isok1);//this指的是app实例
                this.isok1 = !this.isok1;
            },
            change2() {
                this.isok2 = !this.isok2;
            }
        }
    });

</script>

</html>
```

3.9.2 v-if 与 v-show 比较

- 什么时候元素被渲染
    - v-if 如果在初始条件为假，则什么也不做，每当条件为真时，都会重新渲染条件元素
    - v-show 不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行

切换

- 使用场景选择
  - v-if 有更高的切换开销，
  - v-show 有更高的初始渲染开销。
  - 因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行后条件很少改变，则使用 v-if 较好

3.10 列表渲染 v-for

3.10.1 v-for 迭代数组

- 语法： v-for="(alias, index) in array"
- 说明： alias : 数组元素迭代的别名； index : 数组索引值从0开始(可选)

3.10.2 v-for 迭代对象的属性

- 语法： v-for="(value, key, index) in Object"
- 说明： value : 每个对象的属性值; key : 属性名(可选); index : 索引值(可选) 。
- 可用 of 替代 in 作为分隔符

```
<!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>v-for</title>
</head>

<body>
    <div id="app">
        <h1>v-for渲染数据</h1>
        <!-- <ul v-for="(item, index) in list"> -->
        <!-- item指的是：遍历数组的每一项  index：数组的下标 -->
        <ul v-for="(item, index) of list">
```

```
            <li>{{ index + 1 }}.{{ item.name }} 工资：{{ item.salary }}</li>
            <!-- <li>刘强东 工资：3000</li>
            <li>小马哥 工资:4000</li> -->
        </ul>

        <!-- v-for遍历对象 item：键值  key：键名 index:第几对键值对 -->
        <ul v-for="(value, key, index) in goods">
            <p>{{index + 1}}.{{ key }}:{{ value }}</p>
            <!-- <p>2.tel:锤子手机</p>
            <p>3.price:1999</p> -->
        </ul>
    </div>
</body>
<script src="../node_modules/vue/dist/vue.js"></script>
<script>
    //实例化app对象
    let app = new Vue({
        el: '#app',//el 放挂载对象,里面写的是选择器,不能挂载在html和body节点上
        data: {//放数据的地方
            list: [
                {
                    name: '马云',
                    salary: 2000
                }, {
                    name: '刘强东',
                    salary: 3000
                }, {
                    name: '小马哥',
                    salary: 4000
                }
            ],
            goods: {
                name: '罗老师',
                tel: '锤子手机',
                price: '1999'
            }
        },
        methods: {

        }
```

```
    });

</script>

</html>
```

4.0 事件处理 v-on相关

4.0.1 事件处理方法

- 完整格式： v-on:事件名="函数名" 或 v-on:事件名="函数名(参数……)"
- 缩写格式： @事件名="函数名" 或 @事件名="函数名(参数……)" **注意： @ 后面没有冒号**
- event ：函数中的默认形参，代表原生 DOM 事件
- 当调用的函数，有多个参数传入时，需要使用原生DOM事件，则通过 $event 作为实参传入
- 作用：用于监听 DOM 事件

```
<body>
<div id="app">
<h2>1. 事件处理方法</h2>
<button @click="say">Say {{msg}}</button>
<button @click="warn('hello', $event)">Warn</button>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">

```

4.0.2 事件修饰符

- .stop 阻止单击事件继续传播 event.stopPropagation()
- .prevent 阻止事件默认行为 event.preventDefault()
- .once 点击事件将只会触发一次

4.0.3 按键修饰符

- 格式： v-on:keyup.按键名 或 @keyup.按键名
- 常用按键名：
  - .enter
  - .tab
  - .delete (捕获“删除”和“退格”键)
  - .esc
  - .space
  - .up
  - .down
  - .left
  - .right

```
<!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>事件修饰符</title>
    <style>
        div {
            padding: 50px;
        }

        .father {
            background: hotpink;
        }

        .son {
            background: khaki;
        }
```

```
    </style>
</head>

<body>
    <div id="app">
        <!-- 注意：方法名后面的圆括号可以省略不写,但是如果你需要传参就必须写 -->
        <div class="father" @click="outer">
            <!-- 阻止事件冒泡 -->
            <div class="son" @click.stop="inner"></div>
        </div>
        <!-- 默认行为：a跳转 submit提交 选择文字 回车换行 右键菜单 -->
        <a href="http://www.baidu.com" @click.prevent="show">百度</a>
        <!-- 事件只能触发一次 -->
        <p @click.once="change">变了: {{ num }}</p>
        <!-- <input type="text" @keyup.13="submit"> -->
        <input type="text" @keyup.enter="submit">
    </div>
</body>
<script src="../node_modules/vue/dist/vue.js"></script>
<script>

    /*
        事件的修饰符:https://cn.vuejs.org/v2/guide/events.html#%E4%BA%8B%E4%BB%B
            * @click.stop 阻止冒泡
            * @click.prevent 阻止默认行为
            * @click.once 事件只能触发一次
            * @keyup.键值  键盘修饰符 可以写键值，也可以写单词
    */
    //实例化app对象
    let vm = new Vue({
        el: '#app',//el 放挂载对象,里面写的是选择器,不能挂载在html和body节点上
        data: {//放数据的地方
            num: 0
        },
        methods: {
            outer() {
                alert('父节点')
            },
            inner() {
                alert('子节点');
```

```
            },
            show() {
                console.log('阻止了默认行为');
            },
            change() {
                this.num++;
            },
            submit() {
                console.log('回车提交了');
            }
        }
    });

</script>

</html>
```