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

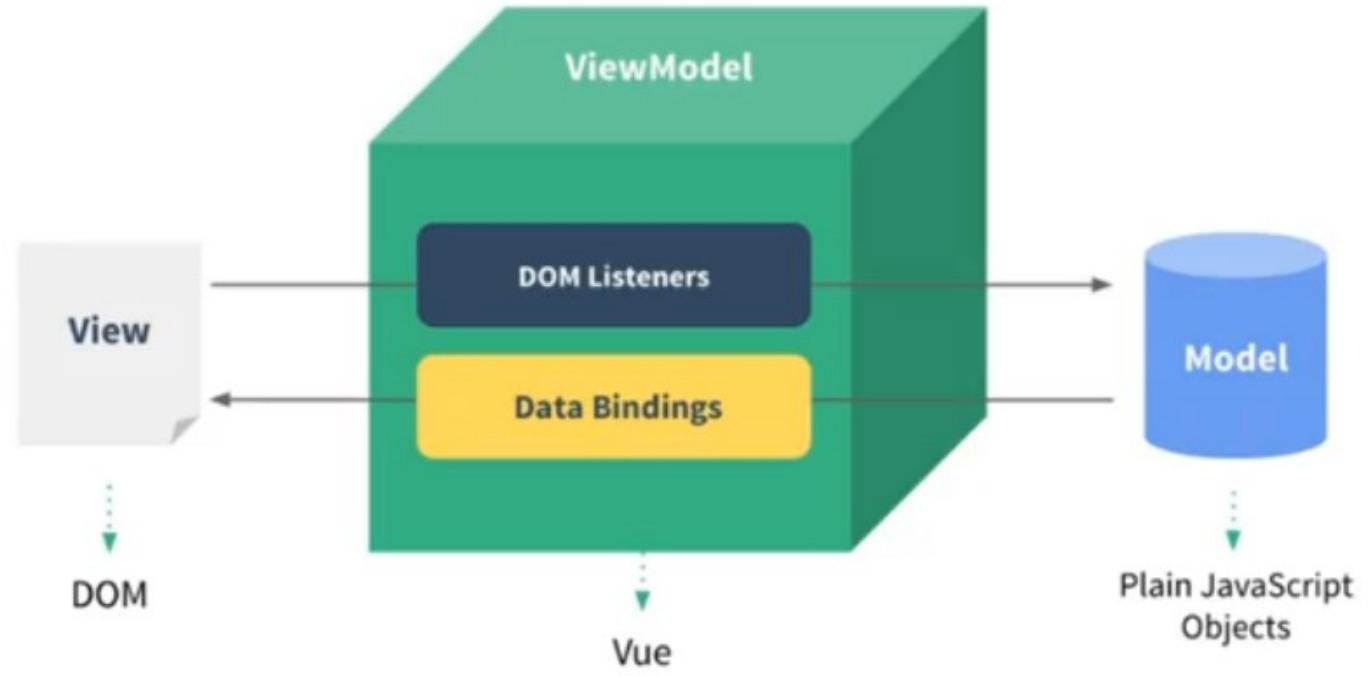
3.1 v-model的运用

```
<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

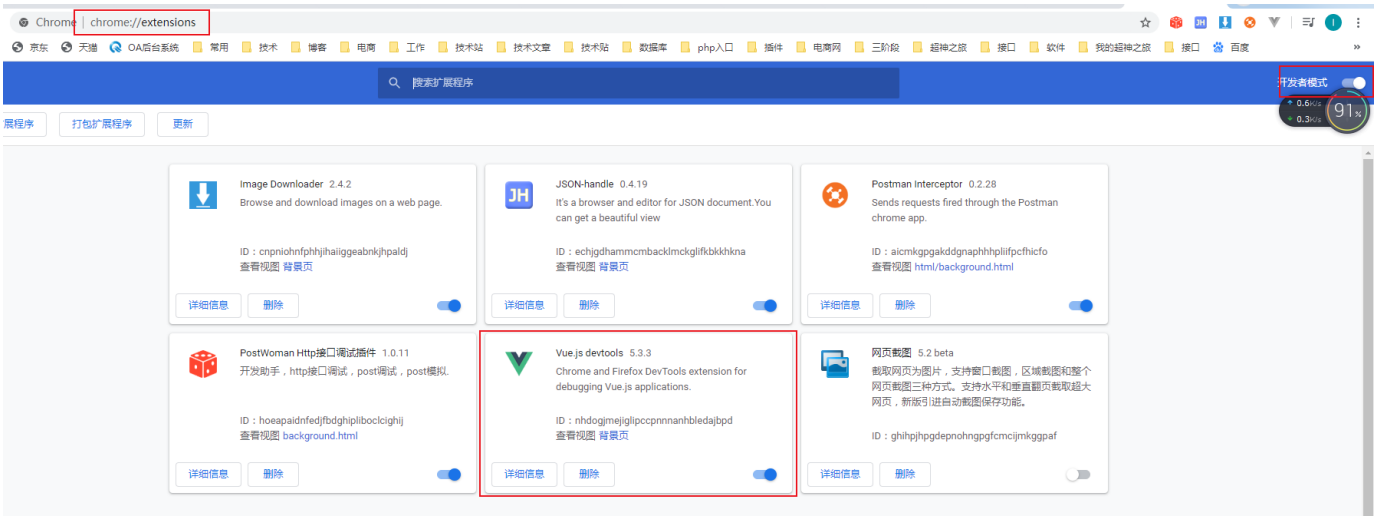
- 常见面试题：什么是 MVVM 模型？ MVVM 是 Model-View-ViewModel 的缩写，它是一种软件架构风格
  - Model：模型， 数据对象（data选项当中 的）
  - View：视图，模板页面（用于渲染数据）
  - ViewModel：视图模型，其实本质上就是 Vue 实例
- 核心思想
  - 通过数据驱动视图 把需要改变视图的数据初始化到 Vue中，然后再通过修改 Vue中的数据，从而实现对视图的更新
- 声明式编程
  - 按照 Vue 的特定语法进行声明开发，就可以实现对应功能，不需要我们直接操作Dom元素
- 命令式编程
  - Jquery它就是，需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



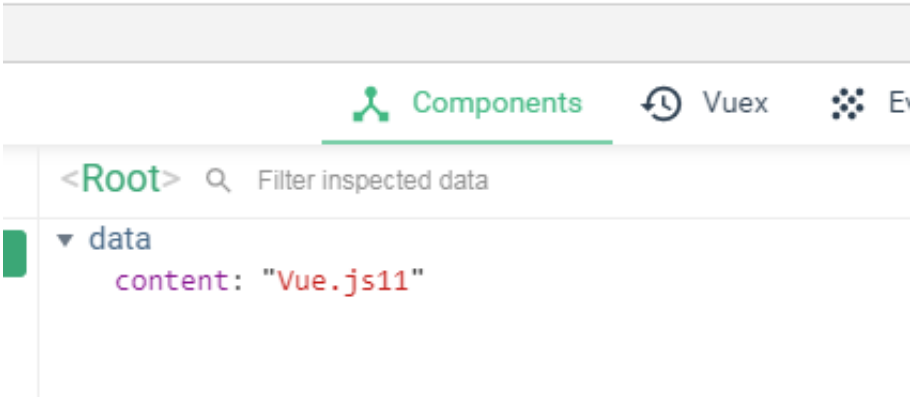
3.3 Vue Devtools 插件安装

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{{}}闪烁问题 v-clock

- 格式： v-clock
- 作用：为了解决{{}}的闪烁问题

```
<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>
```

3.11 事件处理 v-on相关

3.11.1 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">

```

3.11.2 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

3.11.3 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 可以写按键码，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>
```

3.12 表单数据双向绑定v-model

- 单向绑定：数据变，视图变；视图变（浏览器控制台上更新html），数据不变；
- 双向绑定：数据变，视图变；视图变（在输入框更新），数据变

3.12.1 基础用法

- v-model 指令用于表单数据双向绑定，针对以下类型：
    - text 文本
    - textarea 多行文本
    - radio 单选按钮
    - checkbox 复选框
    - select 下拉框

```
<body>
  <div id="demo">
    <form action="#">
     姓名(文本): <input type="text" >
      <br><br>
     性别(单选按钮):
```

```
        <input name="sex" type="radio" value="1"/>男
        <input name="sex" type="radio" value="0"/>女
      <br><br>
     技能(多选框):
        <input type="checkbox" name="skills" value="java">Java开发
        <input type="checkbox" name="skills" value="vue">Vue.js开发
        <input type="checkbox" name="skills" value="python">Python开发
      <br><br>
     城市(下拉框):
      <select name="citys">
        <option value="bj">北京</option>
      </select>
      <br><br>
     说明(多行文本): <textarea cols="30" rows="5"></textarea>
      <br><br>
      <button type="submit" >提交</button>
    </form>
  </div>
</body>
```

进行双向数据绑定后：

```
<body>
        <div id="demo">
                <!-- @submit.prevent 阻止事件的默认行为，当前阻止的是action行为 --
            <form action="#" @submit.prevent="submitForm">
                姓名(文本): <input type="text" v-model="name">
                <br><br>

                性别(单选按钮):
                    <input name="sex" type="radio" value="1" v-model="sex"/>男
                    <input name="sex" type="radio" value="0" v-model="sex"/>女
                <br><br>

                技能(多选框):
                    <input type="checkbox" name="skills" value="java" v-model="
                    <input type="checkbox" name="skills" value="vue" v-model="s
                    <input type="checkbox" name="skills" value="python" v-model
                <br><br>
```

```
                城市(下拉框):
                <select name="citys" v-model="city">
                    <option v-for="c in citys" :value="c.code">
                        {{c.name}}
                    </option>
                </select>
                <br><br>

                说明(多行文本): <textarea cols="30" rows="5" v-model="desc"></te
                <br><br>
                <button type="submit" >提交</button>
            </form>
        </div>

        <script src="./node_modules/vue/dist/vue.js"></script>
        <script>
            new Vue({
                el: '#demo',
                data: {
                    name: '',
                    sex: '1',   //默认选中的是 男
                    skill: ['vue', 'python'], //默认选中 Vue.js开发 、Python开发
                    citys: [
                        {code: 'bj', name: '北京'},
                        {code: 'sh', name: '上海'},
                        {code: 'gz', name: '广州'}
                    ],
                    city: 'sh', // 默认选中的城市: 上海
                    desc: ''
                },
                methods: {
                    submitForm: function () { //处理提交表单
                        //发送ajax异步处理
                        alert(this.name + ', ' + this.sex + ', ' + this.skills
                    }
                }
            })
        </script>
    </body>
```

3.12.2 v-model的修饰符

```
<!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>修饰符</title>
</head>

<body>
    <div id="app">

        <input type="text" v-model.trim="msg">
        <input type="text" v-model.number="num">
        <!-- v-model默认的效果:oninput事件效果,需求：失去焦点才触发 -->
        <input type="text" v-model.lazy="tex">
        <!-- 点击的时候传参 -->
        <input type="button" value="提交" @click="submit('666')">
        <!-- <input type="button" value="提交" @click="submit2"> -->
        <!-- <input type="button" value="提交" @click="submit3('666',$event)">
    </div>
</body>
<script src="../node_modules/vue/dist/vue.js"></script>
<script>

    /*
        v-model修饰符：
            * v-model.trim 去除前后空格
            * v-model.number 把字符串转成数字类型
            * v-model.lazy 失去焦点才触发
    */
    //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
```

```
            msg: '',
            num: 0,
            tex: ''
        },
        methods: {
            submit(data) {//形参
                // console.log(data);
                // let str = this.msg.trim();
                // console.log(str);

                console.log(this.msg);
            },
            submit2(ev) {
                console.log(ev.target.value);//value
                console.log(ev.target.tagName);//INPUT
            },
            submit3(data, ev) {//形参
                console.log(data);
                console.log(ev);
            }
        }
    });

</script>

</html>
```

3.13 指令v-pre

- v-pre 可以用来显示原始插入值标签 {{}} 。并跳过这个元素和它的子元素的编译过程。加快编译

```
<!-- v-pre作用：保留双花括号，当成文本显示不去编译 -->
<h1 v-pre>{{ 梦里花落知多少 }}</h1>
```

3.14 自定义指令的作用

除了内置指令外，Vue 也允许注册自定义指令。有的情况下，你仍然需要对普通 DOM 元素进行底层操作，这时候使用自定义指令更为方便

自定义指令文档： https://cn.vuejs.org/v2/guide/custom-directive.html

3.14.1 注册与使用自定义指令方式

- 注册全局指令:

```
// 指令名不要带 v-
Vue.directive('指令名', {
 // el 代表使用了此指令的那个 DOM 元素
 // binding 可获取使用了此指令的绑定值 等
 inserted: function (el, binding) {
   // 逻辑代码
}
})
```

- 注册局部指令：

```
directives : {
  '指令名' : { // 指令名不要带 v-
    inserted (el, binding) {
      // 逻辑代码
   }
 }
}
```

注意：注册时，指令名不要带 v-

- 使用指令：
  - 引用指令时，指令名前面加上 v-
  - 直接在元素上在使用即可 ： v-指令名='表达式'
- 需求：
  - 实现输出文本内容全部自动转为大写，字体为红色（功能类型于 v-text，但显示内容为大写)

- 当页面加载时，该元素将获得焦点 (注意： autofocus 在移动版 Safari 上不工作)

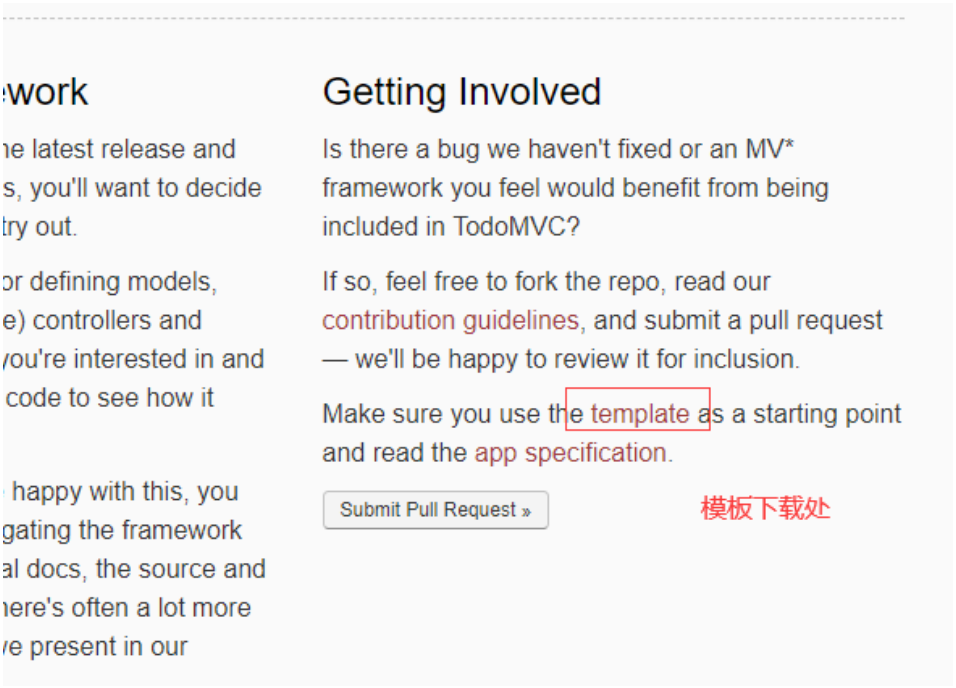
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <div id="app">
        <p v-upper-text="message">xxxxx</p>
        自动获取焦点：<input type="text" v-focus>
    </div>
    <div id="app2">
        <p v-upper-text="msg">xxxxx</p>

    </div>
    <script src="./node_modules/vue/dist/vue.js"></script>
    <script>
        // 注册全局自定义指令，可以在多个Vue管理的入口下使用该指令
        // 第一个参数为指令名，但是不要有v-开头
        Vue.directive('upper-text',{
            //一般对样式 的操作在bind中，bind函数只调用一次
            bind: function (el) {
                el.style.color = 'red'
            },
            //一般对js操作在inserted中，inserted也是只调用一次
            // el是当前指令作用的那个Dom元素，
            // binding用于获取使用了当前指令的绑定值(value)、表达式(expression)、指
            inserted: function (el, binding) {
                // 将所有字母文本内容转换为大写
                el.innerHTML = binding.value.toUpperCase()
            }
        })

        new Vue({
            el: '#app',
```

```
            data: {
                message: 'html5, 拼搏到无能为力'
            },
            //注册局部自定义指令：只能在当前Vue实例管理的入口 下引用这个指令
            directives: {
                'focus': { // 指令名，
                    bind: function () {

                    },
                    // 刷新页面自动获取焦点
                    inserted: function (el, binding) {
                        //被 v-focus 作用的那个元素在刷新页面后会自动 获取焦点
                        el.focus()
                    }
                }
            }
        })

        new Vue({
            el: '#app2',
            data: {
                msg: 'hello'
            }
        })
    </script>
</body>
</html>
```

4.经典实战项目-TodoMVC

4.1 项目介绍与演示

- TodoMVC 是一个非常经典的案例，功能非常丰富，并且针对多种不同技术分别都开发了此项目，比如React、AngularJS、JQuery等等。
- TodoMVC 案例官网：http://todomvc.com/
- 在官网首页右下角， 有 案例的模板下载 和 开发规范（需求文档），如下图：



4.2 需求说明

4.2.1 数据列表渲染

当任务列表（items ）没有数据时， #main 和 #footer 标识的标签应该被隐藏

任务涉及字段 : id 、任务名称（ name ）、是否完成（ completed true 为已完成）

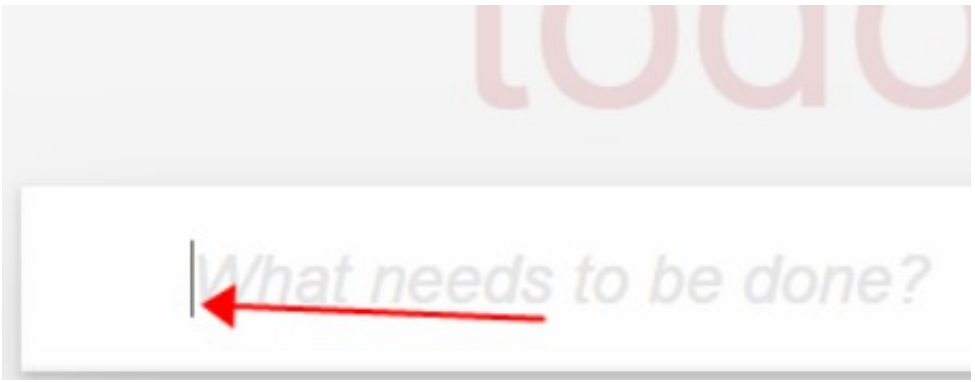
4.2.2 添加任务

1. 在最上面的文本框中添加新的任务。

2. 不允许添加非空数据。

3. 按 Enter 键添加任务列表中，并清空文本框。

4. 当加载页面后文本框自动获得焦点，在 input 上使用 autofocus 属性可获得



4.2.3 显示所有未完成任务数

1. 左下角要显示未完成的任务数量。确保数字是由 标签包装的。

2. 还要将 item 单词多元化(1 没有 s , 其他数字均有 s)： 0 items , 1 item , 2 items

示例： 2 items left

4.2.4 切换所有任务状态

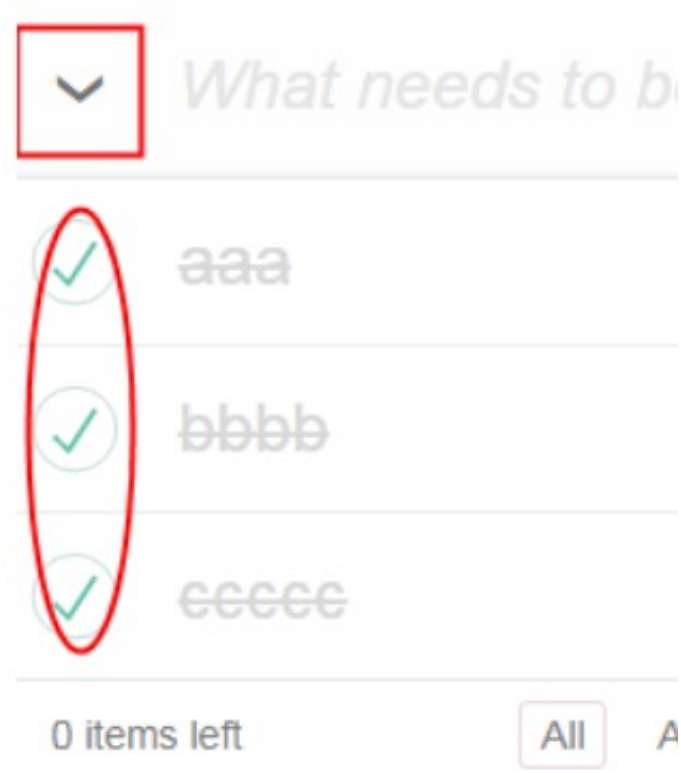
1. 点击复选框 V 后，将所有任务状态标记为复选框相同的状态。

- 分析：
  - 复选框状态发生变化后，就迭代出每一个任务项, 再将复选框状态值赋给每个任务项即可 。
  - 为复选框绑定一个 计算属性 ，通过这个计算属性的 set 方法监听复选框更新后就更新任务项状态。

2. 当 选中/取消 某个任务后，复选框 V 也应同步更新状态。

- 分析：
  - 当所有未完成任务数(remaining)为 0 时，表示所有任务都完成了，就可以选中复选框。
  - 在复选框绑定的 计算属性 的 get 方法中判断所有 remaining 是否为 0 ，从而绑定了 remaining ,
  - 当 remaining 发生变化后, 会自动更新复选框状态（为 0 复选框会自动选中，反之不选中）

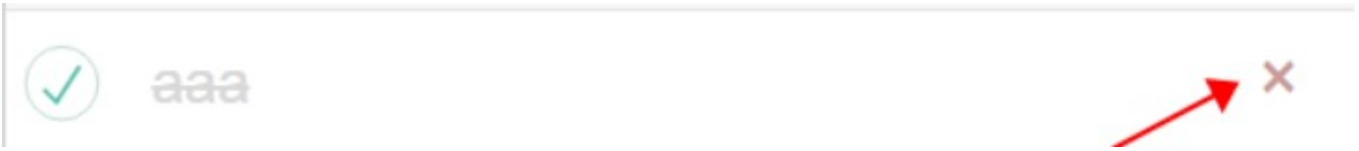
1.点击复选框 V 后，将所有任务状态标记为复选框相同的状态（全选功能）



2.当 选中/取消 了单个任务时，复选框 V 也应同步更新（反控制全选）

4.2.5 移除任务项

悬停在某个任务项上显示 X 移除按钮，可点击移除当前任务项



4.2.6 清除所有已完成任务

1. 单击右下角 Clear completed 按钮时，移除所有已完成任务。

2. 单击 Clear completed 按钮后，确保复选框清除了选中状态。

3. 当列表中没有已完成的任务时，应该隐藏 Clear completed 按钮

4.2.7 编辑任务项

1. 双击 <label> （某个任务项）进入编辑状态（在 <li> 上通过 .editing 进行切换状态）。

2. 进入编辑状态后输入框显示原内容，并获取编辑焦点。

3. 输入状态按 Esc 取消编辑， editing 样式应该被移除。

4. 按 Enter 键 或 失去焦点时 保存改变数据，移除 editing 样式

4.2.8 路由状态切换（过滤不同状态数据）

根据点击的不同状态（ All / Active / Completed ），进行过滤出对应的任务，并进行样式的切换

4.2.9 数据持久化

将所有任务项数据持久化到 localStorage 中,它主要是用于本地存储数据

4.3 实战

- 下载模板：git clone 仓库地址
- 安装依赖：npm i
- 引入vuejs
- 开始开发:编写app.js

index.html

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

```
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Template • TodoMVC</title>
        <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
        <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
        <!-- CSS overrides - remove if you don't need it -->
        <link rel="stylesheet" href="css/app.css">
    </head>
    <body>
        <section class="todoapp" id="todoapp">
            <header class="header">
                <h1>todos</h1>
                <!-- 添加任务， -->
                <input
                @keyup.enter="addItem"
                class="new-todo" placeholder="What needs to be done?" v-app-foc
            </header>
            <!-- This section should be hidden by default and shown when there
            <!-- items.length 当值为0时，表示false，则不显示 -->
            <!-- template元素，页面渲染之后这个template元素就不会有
            ，需要使用 v-if 与 template搭配，如果使用v-show是不行的 -->
            <template v-if="items.length">
                <section class="main" >
                    <input v-model="toggleAll" id="toggle-all" class="toggle-al
                    <label for="toggle-all">Mark all as complete</label>
                    <ul class="todo-list">
                        <!-- These are here just to show the structure of the l
                        <!-- List items should get the class `editing` when edi
                            when marked as completed -->
                        <li v-for="(item, index) in filterItems" :class="{compl
                            <div class="view">
                                <input class="toggle" type="checkbox" v-model="
                                <label @dblclick="toEdit(item)">{{item.content}
                                <button class="destroy" :value="item.id" @click
                            </div>
                            <input  v-todo-focus="item === currentItem" @keyup.
                             @keyup.esc="cancelEdit" class="edit" :value="item.
                        </li>
                    </ul>
                </section>
```

```
                <!-- This footer should hidden by default and shown when there
                <!-- items.length 当值为0时，表示false，则不显示 -->
                <footer class="footer"  >
                    <!-- This should be `0 items left` by default -->
                    <span class="todo-count"><strong>{{remaining}}</strong> ite
                    <!-- Remove this if you don't implement routing -->
                    <ul class="filters">
                        <li>
                            <a :class="{selected: filterStatus === 'all'}" href
                        </li>
                        <li>
                            <a :class="{selected: filterStatus === 'active'}" h
                        </li>
                        <li>
                            <a :class="{selected: filterStatus === 'completed'}
                        </li>
                    </ul>
                    <!-- Hidden if no completed items are left ↓ -->

                    <!-- 当总任务数 大于 未完成任务数量 ，则显示下面按钮 -->
                    <button v-show="items.length > remaining"
                    @click="removeCompleted" class="clear-completed">Clear comp
                </footer>
            </template>
        </section>
        <footer class="info">
            <p>Double-click to edit a todo</p>
            <!-- Remove the below line ↓ -->
            <p>Template by <a href="http://sindresorhus.com">Sindre Sorhus</a><
            <!-- Change this out with your name and url ↓ -->
            <p>Created by <a href="http://todomvc.com">you</a></p>
            <p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
        </footer>
        <!-- Scripts here. Don't remove ↓ -->
        <script src="node_modules/vue/dist/vue.js"></script>
        <script src="node_modules/todomvc-common/base.js"></script>
        <script src="js/app.js"></script>
    </body>
</html>

```

app.js

```
(function (Vue) { //表示依赖了全局的 Vue，其实不加也可以，只是更加明确点

   var STORAGE_KEY = 'items-vuejs';

   // 本地存储数据对象
   const itemStorage = {
      fetch: function () { // 获取本地数据
         return JSON.parse(localStorage.getItem(STORAGE_KEY) || '[]');
      },
      save: function (items) { // 保存数据到本地
         localStorage.setItem(STORAGE_KEY, JSON.stringify(items));
      }
   }

   //初始化任务
   const items = []
    //注册全局指令
    //指令名不要加上v-，在引用这个指令时才需要加上 v-
    Vue.directive('app-focus', {
        inserted (el, binding) {
            //聚集元素
            el.focus()
        }
    })

   var app = new Vue({
      el: '#todoapp',

      data: {
         //items,   // 对象属性简写，等价于items: items
         items: itemStorage.fetch(), //获取本地数据进行初始化
         currentItem: null, //上面不要少了逗号，接收当前点击的任务项
         filterStatus: 'all' // 上面不要少了逗号，接收变化的状态值
      },

      // 监听器
```

```
    watch: {
        // 如果 items 发生改变，这个函数就会运行
        items: {
            deep: true, // 发现对象内部值的变化，要在选项参数中指定 deep: true。
            handler: function(newItems, oldItems) {
                //本地进行存储
                itemStorage.save(newItems)
            }
        }
    },

    //自定义局部指令
     directives : {
         'todo-focus': { //注意指令名称
             update (el, binding) {
                 //只有双击的那个元素才会获取焦点
                 if(binding.value) {
                     el.focus()
                 }
             }
         }
     },

    // 定义计算属性选项
    computed: {
        // 过滤出不同状态数据
        filterItems () {
            //this.filterStatus 作为条件，变化后过滤不同数据
            switch (this.filterStatus) {
                case "active": // 过滤出未完成的数据
                    return this.items.filter( item => !item.completed)
                    break
                case "completed": // 过滤出已完成的数据
                    return this.items.filter( item => item.completed)
                    break
                default: // 其他，返回所有数据
                    return this.items
            }
        },
        // 复选框计算属性
```

```
        toggleAll : {
            get () { //等价于 get : functon () {...}
                console.log(this.remaining)
                //2. 当 this.remaining 发生变化后，会触发该方法运行
                // 当所有未完成任务数为 0 ，表示全部完成，则返回 true 让复选框选中
                //反之就 false 不选中
                return this.remaining === 0
            },
            set (newStatus) {
                // console.log(newStatus)
                //1. 当点击 checkbox 复选框后状态变化后，就会触发该方法运行，
                // 迭代出数组每个元素,把当前状态值赋给每个元素的 completed
                this.items.forEach((item) => {
                    item.completed = newStatus
                })
            }
        },
         // 过滤出所有未完成的任务项
         remaining () {
             /*
               return this.items.filter(function (item) {
                   return !item.completed
               }).length
               */
             //ES6 箭头函数
             return this.items.filter(item => !item.completed).length
         }
    }, // **注意** 后面不要少了逗号 ，

    methods: {
        //编辑完成
        finishEdit (item, index, event) {
            const content = event.target.value.trim();
            // 1. 如果为空，则进行删除任务项
            if (!event.target.value.trim()){
                //重用 removeItem 函数进行删除
                this.removeItem(index)
                return
            }
            // 2. 添加数据到任务项中
```

```
            item.content = content
            // 3. 移除 .editing 样式
            this.currentItem = null
        },

        //取消编辑
        cancelEdit () {
            // 移除样式
            this.currentItem = null
        },

        // 进入编辑状态,当前点击的任务项item赋值currentItem，用于页面判断显示 .edi
        toEdit (item) {
            this.currentItem = item
        },
        //移除所有已完成任务项
        removeCompleted () {
            // 过滤出所有未完成的任务，重新赋值数组即可
            this.items = this.items.filter(item => !item.completed)
        },

        // 移除任务项
        removeItem (index) {
            // 移除索引为index的一条记录
            this.items.splice(index, 1)
        },

        //增加任务项
        addItem (event) { //对象属性函数简写，等价于addItem: function () {
            console.log('addItem', event.target.value)
            //1. 获取文本框输入的数据
            const content = event.target.value.trim()
            //2. 判断数据如果为空，则什么都不做
            if (!content.length) {
                return
            }
            //3.如果不为空，则添加到数组中
            //  生成id值
            const id = this.items.length + 1
            //  添加到数组中
```

```
            this.items.push({
                id, //等价于 id:id
                content,
                completed: false
            })
            //4. 清空文本框内容
            event.target.value = ''
        }
    }
})

//当路由 hash 值改变后会自动调用此函数
window.onhashchange = function () {
    console.log('hash改变了' + window.location.hash)
    // 1.获取点击的路由 hash , 当截取的 hash 不为空返回截取的，为空时返回 'all'
    var hash = window.location.hash.substr(2) || 'all'

    // 2. 状态一旦改变，将 hash 赋值给 filterStatus
    //    当计算属性 filterItems  感知到 filterStatus 变化后，就会重新过滤
    //    当 filterItems 重新过滤出目标数据后，则自动同步更新到视图中
    app.filterStatus = hash
}
// 第一次访问页面时,调用一次让状态生效
window.onhashchange()

})(Vue);
```

5.过滤器

5.1 什么是过滤器

- 过滤器对将要显示的文本，先进行特定格式化处理，然后再进行显示
- 注意：过滤器并没有改变原本的数据, 只是产生新的对应的数据

5.2 使用方式

5.2.1 定义过滤器

全局过滤器：

```
Vue.filter(过滤器名称, function (value1[,value2,...] ) {
 // 数据处理逻辑
})
```

局部过滤器：在Vue实例中使用 filter 选项 ， 当前实例范围内可用

```
new Vue({
  filters: {
   过滤器名称: function (value1[,value2,...] ) {
    // 数据处理逻辑
  }
 }
})
```

5.2.2 过滤器的使用

过滤器可以用在两个地方：双花括号 {{}} 和 v-bind 表达式

```
<!-- 在双花括号中 -->
<div>{{数据属性名称 | 过滤器名称}}</div>
<div>{{数据属性名称 | 过滤器名称(参数值)}}</div>
<!-- 在 `v-bind` 中 -->
<div v-bind:id="数据属性名称 | 过滤器名称"></div>
<div v-bind:id="数据属性名称 | 过滤器名称(参数值)"></div>
```

5.3 案例演示

需求：

1. 实现过滤敏感字符，如当文本中有 tmd、sb 都将进行过滤掉

2. 过滤器传入多个参数 ，实现求和运算

```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>过滤器</title>
</head>
<body>
    <div id="app">
        <h3>测试过滤器单个参数</h3>
        <!-- 原始属性名|过滤器 -->
        <p>{{content | contentFilter}}</p>
        <input type="text" :value="content | contentFilter">

        <h3>测试过滤器多个参数</h3>
        <p>{{javaScore | add(vueScore, pythonScore)}}</p>
        <input type="text" :value="javaScore | add(vueScore, pythonScore)">
    </div>
    <script src="node_modules/vue/dist/vue.js"></script>
    <script>

        /*全局过滤器*/
        /* Vue.filter('contentFilter', function (value) {
            if(!value) {
                return ''
            }
            return value.toString().toUpperCase().replace('TMD', '***').replace
        }) */

        new Vue({
            el: '#app',
            data: {
                content: '小伙子，tmd就是个SB',
                javaScore: 90,
                vueScore: 99,
                pythonScore: 89
            },

            filters: { //定义局部 过滤器
```

```
                contentFilter (value) { // contentFilter 过滤名, value
                    if(!value) {
                        return ''
                    }
                    return value.toString().toUpperCase().replace('TMD', '***')
                },
                add (num1, num2, num3) {// add 过滤名， num1 其实就是引用时 | 左边
                    return num1 + num2 + num3
                }
            }
        })
    </script>
</body>
</html>
```