``VUE：不建议直接操作DOM

> Vue.js是前端三大新框架：Angular.js、React.js、Vue.js之一，Vue.js目前的使用和关注程度在三大框架中稍微胜出，并且它的热度还在递增
>
> Vue的核心库只关注视图层，Vue的目标是通过尽可能简单的 API 实现响应的数据绑定，在这一点上Vue.js类似于后台的模板语言
>
> Vue也可以将界面拆分成一个个的组件，通过组件来构建界面，然后用自动化工具来生成单页面(SPA - single page application)系统

- Vue.js使用文档已经写的很完备和详细了，通过以下地址可以查看： https://cn.vuejs.org/v2/guide/
- vue.js如果当成一个库来使用，可以通过下面地址下载： https://cn.vuejs.org/v2/guide/installation.html

Vue下载

- **cnmp安装**：
  - **升级npm**：cnpm install npm -g
  - **npm config set cache "C:\nodejs\node_cache"**
  - **npm config set prefix "C:\nodejs\node_global"**
  - **安装cnpm**： 一个国内的npm安装工具
    - 安装淘宝镜像中的cnpm：**npm install -g cnpm --registry=https://registry.npm.taobao.org**
- 设置安装包路径：
  - **cnpm config set cache "C:\nodejs\node_cache"**
- 设置模块安装路径：
  - **cnpm config set prefix "C:\nodejs\node_global"**
  - 之后使用命令安装的模块存储在**C:\nodejs\node_global\node_modules**里
- 使用**cnmp**安装vue：
  - **cnpm install vue -g**
- 安装**vue**命令行工具：
  - **cnpm install vue-cli -g**

vue-devtools调试工具

- 下载vue-devtools：
  - ```
    git clone https://github.com/vuejs/vue-devtools
    ```
- 进入到vue-devtools目录下安装依赖包：
  - ```
    cd vue-devtools-dev
    cnpm install
    cnpm run build
    ```

```
cnpm install yargs
cnpm install acorn
cnpm insatll global-prefix
cnpm insatll @vue/component-compiler-utils
cnpm install vue-style-loader
```

- > 修改shells>chrome文件夹下的mainifest.json 中的persistent为true
- ```
  cnpm install sockjs
  cnpm install express
  cnpm install body-parser
  cnpm install merge-descriptors
  cnpm install finalhandler
  cnpm install on-finished
  cnpm install parseurl
  cnpm install statuses
  cnpm install unpipe
  cnpm install path-to-regexp
  cnpm install methods
  cnpm install utils-merge
  cnpm install setprototypeof
  cnpm install safe-buffer
  cnpm install send
  cnpm install proxy-addr
  cnpm install type-is
  cnpm install vary
  cnpm install raw-body
  cnpm install serve-static
  cnpm install serve-index
  cnpm install http-proxy-middleware
  cnpm install http-proxy
  cnpm install eventemitter3
  cnpm install requires-port
  cnpm install follow-redirects
  cnpm install webpack-dev-middleware
  cnpm install selfsigned
  cnpm install node-forge
  cnpm install opn
  cnpm install is-wsl
  ```

- > **注意**：在console界面下访问Vue的实例及其内部属性，要记得加$符号；自己定义的变量不加。

Vue-CDN

- 两个版本的vuejs：CDN
  - vue.js：开发版本，包含了有帮助的命令行警告
    - ```
      <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
      ```
  - vue.min.js：生产环境版本，优化了尺寸和速度

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

Vue语法

- > 每个vue应用都是通过实例化一个新的vue对象开始的
- 创建第一个模板语法：
  - ```

        {{ message }}
        <!-- 这个也叫做插值表达式 -->

    ```
  - ```
    var vm = new Vue({ // vm这个变量不允许使用连字符，可以使用下划线，比如vm-data是不允许的
        el: "#content",
        // 对应document中的一个标签，当vue对象创建后，这个标签内的区域就被接管
        data: {
            message: "这是vue里的变量"
        }
    })
    ```
- 当一个vue实例被创建时，vue的响应式系统中加入了对其data对象中能找到的所有属性
- 当这些属性值被改变时，视图也会发生**相应**，并将对应属性更新为新的值
- 也可以通过定义函数来改变实例中data对象中的数据，数据改变，视图中的数据也将改变
  - ```

        {{ message }}
        <button @click="ChangeMsg">改变</button>
        <!-- 绑定点击事件为定义好的vue函数 -->

    ```
  - ```
    <script type="text/javascript">
        window.onload = function(){
            var vm = new Vue({
                el: "#app",
                data: {
                    message: "我对应的是message的变量"
                },
                methods:{ // 定义一个函数 并绑定在按钮的点击事件上
                    ChangeMsg:function(){
                        this.message = "我被改变了";
                        // 修改当前实例中的message变量
                    }
                }
            })
        }
    </script>
    ```

返回值

- ```

      <h1>{{ classType }}学习</h1>
      {{ content }}
      {{ describe() }}

  ```

- ```
  <script type="text/javascript">
      window.onload = function(){
          var vm = new Vue({
              el: "#app", // getElementById('app')
              data: {
                  classType: "vue",
                  content: "这是vue的一个测试",
              },
              methods:{
                  describe:function(){
                      return "这是一个函数的返回值"
                  },
              }
          })
      }
  </script>
  ```

Vue模板指令

- > **模板语法指的是如何将数据放入html中**
  >
  > **Vue.js使用了基于HTML的模板语法，允许开发者声明式地将DOM绑定至底层 Vue 实例的数据。所有Vue.js的模板都是合法的 HTML ，所以能被遵循规范的浏览器和HTML 解析器解析**

- 插入值，模板变量：
  - 数据绑定最常见的形式就是使用`Mustache`语法(双大括号) 的文本插值

绑定内容

v-html

- **v-html**：将内容按照html格式进行插入
- > **注意**：在网站上动态渲染任意 HTML 是非常危险的，因为容易导致XSS攻击。只在可信内容上使用`v-html`，**永不**用在用户提交的内容上

- ```

  ```
  ```

```
var vm = new Vue({
    el: "#app",
    data: {
      content: "<b>段落标签</b>文本内容"
    },
})
```

v-text

- **v-text**：将内容按照文本格式进行插入，但会覆盖原有标签内的内容，不会有加载的闪烁问题
- ```

          {{ gender ? '男' : '女' }}
          <!-- ok? true:false -->

  ```
- ```
  var vm = new Vue({
      el: "#app",
      data: {
          gender: true, // 变量值为true时，显示模板变量中左边的值
          content: "<b>段落标签</b>文本内容"
      },
  })
  ```

v-cloak

- **v-cloak**：解决加载时的闪烁问题
- > 这个指令可以隐藏未编译的标签直到实例准备完毕
  - ```

        {{ message }}

    <script type="text/javascript" src="js/vue.js"></script>
    <script type="text/javascript">
        new Vue({
            el: "#app",
            data:{
                message: "测试",
            }
        })
    </script>
    ```
  - 在上面的代码中，如果网速够慢的清空下，页面首先加载显示出的内容是`{{ message }}`
- **解决办法**：通过`v-clock`指令，在使用到模板变量的标签上写入，并设置一个`v-clock`的类样式
    ```

```
<style type="text/css">
    [v-cloak]{
        display: none;
    }
</style>
```

- ```
  <div v-cloak id="app">
      <p v-cloak>{{ message }}</p>

  ```

绑定属性

v-bind

- 如果我们需要设置的模板变量是一个属性，比如a标签的href属性：
- ```
  <div id="app">
      <a v-bind:href="message">连接</a>
      <a :href="message +'abc'">连接</a>
      <!-- 属性内的模板变量写法已被移除，使用v-bind:attr 或 :attr -->
  </div>
  ```
- ```
  var vm = new Vue({
      el: "#app",
      data: {
          message: "https://www.baidu.com"
      }
  })
  ```

绑定事件

v-on

- v-on：给元素绑定对应事件
- ```
  <div id="app">
      <button v-on:click="show">按钮</button>
      <button @click="show">按钮</button>
  </div>
  ```
- ```
  new Vue({
      el: "#app",
      method: {
          show: function(){
              alert("弹一下")
          }
      }
  })
  ```

跑马灯效果
  ```

- ```

      <h3 v-html="message"></h3>
      <button @click="start">开始</button>
      <button @click="stop">停止</button>

  ```

- ```
  new Vue({
      el: "#app",
      data: {
          message: "这是一个跑马灯",
          sT: null, // 定时器实例
      },
      methods:{
          work(){
              this.message = this.message.substring(1) + this.message[0]
              // 循环定时器所作的事情
          },
          start(){
              if (this.sT==null) { // 判断此时是否已有定时器开启
                  console.log("开启定时器")
                  this.sT = setInterval(this.work,400)
              } else {
                  console.log("已经开启 不在开启")
              }

          },
          stop(){ // 关闭定时器 设置定时器变量为null
              console.log("关闭定时器")
              clearInterval(this.sT)
              this.sT = null
          }
      }
  ```

绑定事件修饰符

阻止冒泡：.stop

> 比如一个按钮在一个div中，并且按钮和div均有自己的事件，那么此时点击按钮，事件会像冒泡一样从按钮开始一直到div进行触发，.stop用来阻止默认的事件触发行为

- ```
  <div id="fDiv" @click="divClick">
      <button id="fBtn" @click="btnClick">按钮</button>

  ```

- ```
  <script type="text/javascript">
      window.onload = function(){
          var vm = new Vue({
              el: "#fDiv", // 控制区域
              data: {},
              methods: {
                  divClick(){
  ```

```
                    console.log("div被点击了")
                },
                btnClick(){
                    console.log("按钮被点击了")
                }
            },
        })
    }
</script>
```

- 通过.stop修饰阻止冒泡：
    - ```
      <div id="fDiv" @click="divClick">
          <button id="fBtn" @click.stop="btnClick">按钮</button>

      ```

阻止默认行为：.prevent

> 比如像a标签这样的，在点击时他有默认的跳转动作，可以通过.sprevent阻止该默认行为

- ```
  <div id="fDiv">
      <a href="https://www.baidu.com" @click.prevent="aLink">去百度</a>
  </div>
  ```

- ```
  var vm = new Vue(){
      el: "#fDiv",
      methods:{
          aLink(){
              console.log("连接被点击")
          }
      }
  }
  ```

捕获事件：.capture

- > 默认的事件触发处理机制是冒泡机制，通过.capture即可将冒泡顺序从里向外，颠倒顺序
  >
  > 也可理解为谁有该修饰符，先触发谁的事件

- ```
  <div id="fDiv" @click.capture="divClick">
      <button id="fBtn" @click="btnClick">按钮</button>
  </div>
  ```

- ```
  <script type="text/javascript">
      window.onload = function(){
      var vm = new Vue({
          el: "#fDiv", // 控制区域
          data: {},
          methods: {
              divClick(){
                  console.log("div被点击了")
              },
  ```
  ```

```
            btnClick(){
                console.log("按钮被点击了")
            }
        },
    })
    }
</script>
```

自身事件：.self

- > 与capture和冒泡不同，.self只有是自身触发的当前的事件才真正执行处理的回调函数
  >
  > 并且.self只会阻止当前元素的事件触发行为

- ```
  <div id="fDiv" @click.self="divClick">
      <button id="fBtn" @click.self="btnClick">按钮</button>

  ```

- ```
  // 与上同
  ```

单次事件：.once

- > 使用.once只触发一次事件函数

- ```

      <a href="https://www.baidu.com" @click.prevent.once="aLink">去百度</a>
      <!-- 连接无法跳转的阻止事件 只会出现一次 -->

  ```

- ```
  var vm = new Vue(){
      el: "#fDiv",
      methods:{
          aLink(){
              console.log("连接被点击")
          }
      }
  }
  ```

表单绑定 v-model

v-model

- > 使用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
  >
  > 根据表单上的值，自动更新模板变量中的值

- > 注意v-model会忽略表单的初始值，比如：`checked`、`value`、`selected`，如果需要的话，应该在javascript中首先声明初始值

text

```
<div id="container">
    <h3 v-html="message"></h3>
    <input type="text" v-model="message">
</div>
```

- ```
  <script>
      window.onload = function(){
          var vm = new Vue({
              el: "#container",
              data: {
                  message: "这是个表单内容",
              },
          })
      }
  </script>
  ```

textarea

- ```
  <div id="container">
      <h3 v-html="message"></h3>
      <textarea v-model="message"></textarea>
  </div>
  ```

- ```
  // 同上
  ```

checkbox

- **单个复选框**：数据为绑定为`true`和`false`的布尔值
  - ```
    <div id="container">
        <h3 v-html="checked"></h3>
        <input type="checkbox" v-model="checked">
    </div>
    ```
  - ```
    <script>
        window.onload = function(){
            var vm = new Vue({
                el: "#container",
                data: {
                    checked: true,
                },
            })
       }
    </script>
    ```
- **多个复选框**：选中的结果会绑定到同一个数组，将保存的`v-model`变量创建为数组
    ```

```
<div id="container">
    <h3 v-html="checked"></h3>
    <input name="fruit" type="checkbox" value="apple"  v-model="checked">苹果
    <input name="fruit" type="checkbox" value="banana" v-model="checked">香蕉
    <input name="fruit" type="checkbox" value="orange" v-model="checked">橘子
</div>
```

```
<script>
    window.onload = function(){
        var vm = new Vue({
            el: "#container",
            data: {
                checked: new Array,
            },
        })
    }
</script>
```

radio

```
<div id="container">
    <h3 v-html="picked"></h3>
    <input type="radio" name="gender" value="junior" v-model="picked">男
    <input type="radio" name="gender" value="girl" v-model="picked">女
</div>
```

```
<script>
    window.onload = function(){
        var vm = new Vue({
            el: "#container",
            data: {
                picked: "哈哈哈哈",
            },
        })
    }
</script>
```

select

```
<div id="container">
    <h3 v-html="selected"></h3>
    <select v-model="selected">
        <option disabled value="">你想去哪</option>
        <option value="山西">山西</option>
        <option value="北京">北京</option>
        <option value="上海">上海</option>
    </select>
</div>
```

```
<script>
    window.onload = function(){
        var vm = new Vue({
            el: "#container",
            data: {
                selected: "",
            },
        })
    }
</script>
```

selects

- > 设置select标签的multiple属性即可设置为多选下拉菜单，按着ctrl键可以多选

- ```

      <h3 v-html="selecteds"></h3>
      <select multiple v-model="selecteds">
          <option value="上衣">上衣</option>
          <option value="裤子">裤子</option>
          <option value="鞋">鞋</option>
      </select>

  ```

- ```
  <script>
      window.onload = function(){
          var vm = new Vue({
              el: "#container",
              data: {
                  selecteds: new Array, // 多重数据一般都要保存成数组
              },
          })
      }
  </script>
  ```

修饰符

.lazy

> 默认情况下，v-model在input和textarea表单中进行同步输入框的改动，添加了.lazy修饰符之后，对应的v-model绑定事件触发机制将变为change事件，只有在光标失去焦点时会触发

- ```

      <h3 v-html="message"></h3>
      <input type="text" v-model.lazy="message">

  ```
  ```

```
<script>
    window.onload = function(){
        var vm = new Vue({
            el: "#container",
            data: {
                message: "这是个表单内容",
            },
        })
    }
</script>
```

.number

> 如果用户希望将输入表单的内容处理为Number类型，可以使用.number给v-model进行修饰；如果表单字符串无法被处理为数字，则返回原始的值

- ```

      <h3 v-html="typeof message"></h3>
      <input type="text" v-model.number="message">

  ```

- ```
  // 与上同
  ```

.trim

> 使用.trim可以自动过滤输入框的首尾空格

- ```

      <input type="text" v-model.trim="message">
      <br>
      <input type="text" v-model="message">
      <!-- 通过查看另一个表单中同步的缩进 -->

  ```

- ```
  // 与上同
  ```

动态绑定

> 当某些情况下，无法确定表单中所代表的属性值，可以使用v-bind进行动态绑定，v-model获取到的表单输入此时则是我们定义的v-bind属性值

- ```

      <h3 v-html="message"></h3>
      <input type="radio" v-model="message" :value="choiceA"> A
      <input type="radio" v-model="message" :value="choiceB"> B

  ```
  ```

```
<script>
    window.onload = function () {
        var vm = new Vue({
            el: "#container",
            data: {
                message: "", // 表单绑定变量
                choiceA: "Yes!", // 属性绑定变量，未来不需要修改标签中的value值即可动态修改
                choiceB: "No!",

            },
        })
    }
</script>
```

计算属性

- 关键词：computed
  - 模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的

    在模板中放入太多的逻辑会让模板过重且难以维护

    也就是说，某些时候页面中的模板变量如果需要复杂的运算处理，应该使用**计算属性**，而不是直接在模板位置进行计算。

- ```
  <script type="text/javascript">
      window.onload = function () {
              var vm = new Vue({
                  el: "#container",
                  data: {
                      String1:"这是一个字符串",
                  },
                  methods: {
                      MreverseString(){
                          return this.String1.split("").reverse().join("")
                      } // 定义一个函数进行字符串逆置
                  },
                  computed: {
                      CreverseString(){
                          return this.String1.split("").reverse().join("")
                      } // 定义一个计算属性进行字符串逆置
                  }
              })
          }
  </script>
  ```
  ```

```
<div v-cloak id="container">
    <p>这是一个字符串:{{ String1 }} </p>
    <p>他的逆置:{{ String1.split("").reverse().join("") }} </p>
    <p>他的逆置:{{ CreverseString }} </p>
    <!-- 计算属性直接写入函数名 -->
    <p>他的逆置:{{ MreverseString() }} </p>
    <!-- 普通methods函数调用需加括号 -->
</div>
```

- > **注意**：虽然计算属性和函数都可以达成同样的目的，但是computed会缓存结果，计算属性如果发现依赖的属性message未发生改变，再次访问计算属性不会重复运算函数，而是直接利用已有结果；如果依赖数据发生改动，计算属性函数才会重新运算。
- 在函数及计算属性中添加日志输出即可看到这个效果：
  - ```
    methods: {
        MreverseString() {
            console.log("MreverseString被运算了")
            return this.String1.split("").reverse().join("")
        }
    },
    computed: {
        CreverseString() {
            console.log("CreverseString被运算了")
            return this.String1.split("").reverse().join("")
        }
    }
    ```
  - 在终端下进行计算属性以及函数的访问即可看到效果。

计算属性SetAttr

- > 默认的计算属性只有获取getattr的方式，我们可以手动为他添加一个setter
- ```
  computed:{
      CreverseString: {
          get: function(){
              return this.String1.split("").reverse().join("")
          }
          set: function(val){
              this.String1 = val.split("").reverse().join("")
              // 如果当前的逆置之后字符串为val，那么原本的字符串需要再颠倒一次
          }
      }
  }
  ```
-

侦听属性

- > 侦听属性的作用是侦听某些属性的变化，从而做相应的操作，进行对数据变化的相应，
  >
  > 侦听属性是一个对象（字典），key值是要监听的元素，值是当监听的元素发生改变时要执行的函数；

> 监听函数有两个参数，一个是当前值，另一个是变化后的值

- 比如监听一个变量的变化
  - ```
    <script type="text/javascript">
        window.onload = function(){
            var vm = new Vue({
                el: "#container",
                data:{
                    content: "", // 表单内容
                    count: 0, // 记录表单内用户敲了多少次
                },
                watch:{
                    content:function (oldVal,newVal){
                        // 只要在文本框输入内容影响到了age数据发生改变，就会触发
                        this.count += 1
                    },
                },
            })
        }
    </script>
    ```
  - ```
    <div id="container">
        <p><label>你敲了:{{ count }}次</label></p>
        <input type="text" placeholder="请输入你的年纪" v-model="content">
    </div>
    ```

属性绑定：Class

- > 使用`v-bind:class`指令来设置元素的class属性；
  >
  > 属性表达式的类型可以是字符串、对象或数组

数组属性

- > 可以通过为元素绑定一个数组，用来为元素设置单个或多个样式，类名在数组中用单引号
- ```
  <style type="text/css">
      .fontBold {
          font-weight: bold;
      }
      .fontRed {
          color: red;
      }
  </style>
  ```
- ```
  <div id="container">
      <p :class="['fontBold','fontRed']">这是一个段落</p>
  </div>
  ```

动态属性

- > 可以通过为元素绑定一个对象，对象的key是样式类，对象的value是true或false来动态切换class
- ```
  <script type="text/javascript">
      window.onload = function () {
          var vm = new Vue({
              el: "#container",
              data: {
                  flag: true,
              },
              methods: {
                  toggle() {
                      if (this.flag){ // 判断当前toggle变量的属性，对称变换
                          this.flag = false
                      }else{
                          this.flag = true
                      }
                  }
              }
          })

          }
  </script>
  ```
- ```
  <div id="container">
      <p :class="{fontBold: flag}" @click="toggle">这是一个段落</p>
      <p :class="{flag? fontBold:''}" @click="toggle">这是一个段落</p>
      <!-- 三元表达式 -->
  </div>
  ```

属性绑定：Style

- > 使用`v-bind:style`语法，为元素绑定样式
- ```
  <p :style="{color:'red','font-weight':'bold'}">
      一段文字
  </p>
  ```
- 也可以在vue的data中定义一个对象，用来描述样式，其中带有连字符的样式属性要加引号
  - ```
    <div id="container">
        <p :style="styleObj">一段文字</p>
    </div>
    ```
  - ```
    data: {
        styleObj: {
            color:'red',
            'font-weight':'bold',
        }
    },
    ```
- data中的对象也可以通过数组类型绑定到元素上
    ```

-
```
<div id="container">
    <p :style="[styleObj1,styleObj2]">一段文字</p>
    <!-- 对于js的样式绑定不需要加引号，因为就是一个变量 -->
</div>
```

-
```
styleObj1: {
    border: '1px solid gray',
    width: '100px',
},
styleObj2:{
    background: 'black',
    color: 'blue',
}
```

条件渲染

> 通过条件指令可以控制元素的显示及隐藏，或者说叫做创建和销毁

v-if

- > `v-if`指令用于条件性的渲染一块内容。这块内容只会在指令的表达式返回`truthy`值的时候渲染

-
```
<div v-cloak id="container">
    <h3 v-if="oh3">h3标题</h3>

    <p v-if="gender === 'girl'">你是女的</p>
    <p v-else-if="gender === 'boy'">你是男的</p>
    <p v-else>不男不女</p>
</div>
```

-
```
<script type="text/javascript">
    window.onload = function(){
        var vm = new Vue({
            el: "#container",
            data:{
                oh3:"a",
                gender: 'other'
            },

        })
    }
</script>
```

- truthy和ture的区别：
  - 隐含有true属性的变量不可以认为它是true,它不是boolean类型

v-show

- > 与v-if不同的是：`v-show` 的元素始终会被渲染并保留在 DOM 中
  >
  > `v-show` 只是简单地切换元素的 CSS 属性 `display`

- ```
  <div v-cloak id="container">
      <h3 v-if="oh3">h3标题</h3>
      <h4 v-show="oh4">h4标题</h4>

  ```

- ```
  <script type="text/javascript">
      window.onload = function(){
          var vm = new Vue({
              el: "#container",
              data:{
                  oh3:"1", // v-if 在该变量不为真时直接消失在document中
                  oh4:"1", // v-show 处理不为真的变量条件 绑定元素不会消失
              },

          })
      }
  </script>
  ```

列表渲染

v-for

> 把一个数组对应为一组元素

- > 用 `v-for` 指令根据一组数组的选项列表进行渲染
  >
  > `v-for` 指令需要使用 `item in items` 形式的特殊语法，`items` 是源数据数组并且 `item` 是数组元素迭代的别名

  - ```
    <ol id="container">
        <li v-for="user in users">
            {{ user.name }}
        </li>
    </ol>
    ```

  - ```
    <script type="text/javascript">
        window.onload = function(){
            var vm = new Vue({
                el: "#container",
                data:{
                    users: [
                        {name:"张三",age:18},
                        {name:"李四",age:20},
                        {name:"王五",age:19},
                    ]
                },

            })
        }
    </script>
    ```

- > v-for还可以支持将当前循环索引作为渲染时的第二个参数
  - ```

        {{ index }}:{{ user.age }}

    ```
  - ```
    data:{
        users: [
            {name:"张三",age:18},
            {name:"李四",age:20},
            {name:"王五",age:19},
        ]
    },
    ```
- 使用`v-for`迭代访问一个对象
  - ```

        {{ key }}
        <!-- 当v-for渲染时只有一个参数，此时参数为value值 -->

    ```
  - ```
    myself : {
        name:"恩泽",
        age:"17",
    }
    ```
- `v-for`支持最多三个参数，同时获取遍历对象的key和value值，以及index索引位置
  - > 要注意的是，此时的key和value和python中的顺序是颠倒的，key在后，value在前
  - ```

        {{ index }}: {{ key }} - {{ value }}

    ```
  - ```
    myself : {
        name:"恩泽",
        age:"17",
    }
    ```
- `v-for`进行一段取值
  - ```

            {{ n }}

        <!-- 1 2 3 4 5 6 7 8 -->

    ```
  -
    ```

选项卡练习

- ```
  <script type="text/javascript">
      window.onload = function(){
          var vm = new Vue({
              el: "#container",
              data: {
                  choicId: null,
              }
          })
      }
  </script>
  ```

- ```
  <style>
      li{
          list-style-type: none;
          border: 3px outset lightgreen;
          width: 100px;
          background:lightblue;
          margin:5px;
      }
      li:hover{
          border: 3px inset gray;
          cursor: pointer;
      }
      [v-cloak]{
          display: none;
      }
  </style>
  ```

- ```
  <div v-cloak id="container">
      <ol list>
          <li @click="choicId = 1">A</li>
          <li @click="choicId = 2">B</li>
          <li @click="choicId = 3">C</li>
          <li @click="choicId = 4">D</li>
      </ol>
      aaaaaaaaaa
      bbbbbbbbbb
      cccccccccc
      dddddddddd

  ```

- > **js中两个等号和三个等号的区别：**

  - "=="表示： 如果两边值的类型不同的时候，是要先先进行类型转换后，才能做比较
    equality 等同

  - "==="表示：不需要做类型转换，如果两边值的类型不同，就表示一定是不等的
    identity 恒等
  ```

注意

> Vue无法检测到对于数组的索引设置及长度修改

> Vue无法检测到以下对于对象属性的删除或添加

- ```
  Vue.set($vm.Object,"key","val") // 对于对象 这样的添加方式可以触发状态更新
  Vue.set($vm.Array, index, newVal) // 对于数组 添加元素 触发状态更新
  // vm.items.splice(newLength) // 设置数组长度
  ```

- javascript.splice(where, num, [additem1,additem2...])：删除或添加元素

> 此外，当v-for与v-if同时使用时，v-for有更高的优先级，这会造成重复遍历得到的元素都要在做一次v-if的判断，如果我们是为了有目的判断当前是否需要渲染这个元素，或是跳过这个循环，可以将v-if放在外层元素，比如template标签中，(template标签无实际意义，默认不展示，但是可以起到包裹作用)

过滤器

- > Vue.js 允许你自定义过滤器，可被用于一些常见的文本，对它们进行格式化
  >
  > 过滤器可以用在两个地方：**双花括号插值和 v-bind 表达式** (后者从 2.1.0+ 开始支持)
  >
  > 过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示

- 语法：
  - ```
    <p>
        {{ message | filter }}
    </p>

    <p v-bind:type="message | filter"> </p>
    ```

- 过滤器本质上是一个函数，比如我们定义一个将表单输入的内容中所有的字母变大写的过滤器
  - ```
    <body>
        <div v-cloak id="container">
            <input type="text" v-model="message">
            展示: {{ message | toUpper }}

    </body>
    ```

  - ```
    <script type="text/javascript">
        window.onload = function () {
            var vm = new Vue({
                el: "#container",
                data: {
                    message: "",
                },
                filters: {
                    toUpper: function (value) {
                        if (!value) return '' // 字符串内容为空 直接返回
                        console.log("正在变大小")
                        return String(value).toUpperCase()
                    }
    ```

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

- 过滤器函数也可以有多个参数：

```
{{ message | filter(arg1, arg2) }}
// message 第一个参数
// arg1 第二个参数
// arg2 第三个参数
```

```
<div v-cloak id="container">
    <input type="text" v-model="message">
    <p>展示: {{ message | toLong("| "," |") }}</p>
</div>
```

```
filters: {
    toLong(value,arg1,arg2){
        if (!value) return ''
        return arg1 + value + arg2
    }
}
```

Vue实例生命周期

> 每个Vue实例在被创建时都要经过一系列的初始化过程
>
> 例如：需要设置数据监听、编译模板、将实例挂载到DOM并在数据变化时更新DOM 等。
>
> 同时在这个过程中会自动运行一些叫做生命周期钩子的函数，我们可以使用这些函数，在实例的不同阶段加上我们需要的代码，实现特定的功能

```
- beforeCreate 数据还没有监听，没有绑定到vue对象实例，同时也没有挂载对象
- created 数据已经绑定到了对象实例，但是还没有挂载对象

- beforeMount 模板已经编译好了，根据数据和模板已经生成了对应的元素对象，将数据对象关联到了对象的 $el属性
-----------------------------------------------------------------------------------------
$el属性是一个HTMLElement对象，也就是这个阶段，vue实例通过原生的createElement等方法来创建这个html片段，准备注入到我们vue实例指明的el属性所对应的挂载点
-----------------------------------------------------------------------------------------
- mounted 将$el的内容挂载到了el，相当于我们在jQuery执行了$(el).html($el),生成页面上真正的dom
-----------------------------------------------------------------------------------------
上面我们就会发现页面的元素和我们$el的元素是一致的。在此之后，我们能够用方法来获取到el元素下的dom对象，并进行各种操作
-----------------------------------------------------------------------------------------

- beforeUpdate 数据发生变化时调用
- 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

// window.$vm.$destroy()
```

- beforeDestroy Vue实例销毁前
- destroyed Vue实例销毁后

> 一大段代码进行钩子函数的调用过程监控

```
<script type="text/javascript">
    window.onload = function () {
    function showData(process, vm) {
        console.log(process)
        console.log("vue数据:", vm.message) // 当前Vue中的数据
        console.log("Vue挂载el:") // Vue接管的元素
        console.log(vm.$el)
        console.log("真实Dom:")
        console.log(document.getElementById("container").innerHTML)
        console.log('-----------------')
    } // 这个函数用来输出相关信息的
    new Vue({
        el: "#container",
        data: {
            message: "aaaaa",
        },
        beforeCreate: function () {
            showData("创建Vue实例前", this)
        },
        created: function () {
            showData("创建Vue实例后", this)
        },
        beforeMount: function () {
            showData("挂载到Dom前", this)
        },
        mounted: function () {
            showData("挂载到Dom后", this)
        },
        beforeUpdate: function () {
            showData("数据发生变化时", this)
        },
        updated: function () {
            showData("数据发生变化后", this)
        },
        beforeDestroy: function () {
            showData("Vue实例销毁前", this)
        },
        destroyed: function () {
            showData("Vue实例销毁后", this)
        }
    })
}
</script>
```

```
<div id="container">
    <p v-html="message"></p>
</div>
```

组件

ES6语法

> ES6是JavaScript语言的新版本，它也可以叫做ES2015，之前学习的JavaScript属于ES5，ES6在它的基础上增加了一些语法
>
> ES6是未来JavaScript的趋势，而且vue组件开发中会使用很多的ES6的语法，所以掌握这些常用的ES6语法是必须的

变量声明

- let：定义封闭作用域的变量，并且变量只能声明一次
- const：定义封闭作用域的常量，并且变量只能声明一次
- > let和const是新增的声明变量的开头的关键字，在这之前，变量声明是用var关键字；
  >
  > 这两个关键字和var的区别是，它们声明的变量没有**预解析**，无法脱离定义空间使用
  >
  > let和const的区别是，let声明的是一般变量，const申明的常量，不可修改
- ```
  console.log(a) // undefined
  console.log(b) // b is not defined
  console.log(c) // c is not defined
  var a =  1
  var a = 2
  let b = 2
  // let b = 3 // Identifier 'b' has already been declared
  const c = 3
  // const c = 4 // Identifier 'c' has already been declared
  c = 4 //  Assignment to constant variable
  ```

箭头函数

> 可以把箭头函数理解成匿名函数的第二种写法，箭头函数的作用是可以在对象中绑定this；
>
> 解决了JavaScript中this指定混乱的问题

- 定义函数的一般方式
  - ```
    function func(){
        ...
    }
    ```
- 匿名赋值创建函数
  - ```
    var func = function(){
        ...
    }
    ```
- 箭头函数的写法
    ```

```
var func = (a, b) => {
    // 这样的函数在嵌套时，会自动绑定外部作用域下的this
}
var func = a => {
    // 一个参数时，可以省略参数
}
```

- ```
  window.onload = function () {
      var vm = new Vue({
          el: "#container",
          data: { message: "abcdef", },
          methods: {
              show() {
                  console.log("这是show函数:", this.message),
                      func = () => {
                      console.log("我是内部函数:", this.message)
                  },
                      func(), // 调用一下这个内部函数
              }
          }
      })
  }
  ```

- ```
  <div id="container">
      <button @click="show">按钮</button>
  </div>
  ```

Vue组件

- > 组件(Component)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**；
  >
  > 所有的 Vue 组件同时也都是 Vue 的实例，所以可接受相同的选项对象 (除了一些根级特有的选项) 并提供相同的生命周期钩子；

注册全局组件

- 注册一个全局组件语法格式如下：
  - ```
    Vue.component(tagName, options)
    ```
    - **tagName**：组件名
    - **options**：配置选项
- 比如这样一个全局组件：
    ```

```
Vue.component('button_show', {
    data: function () {
        return {
            count: 0
        }
    }, // 当前组件会需要的数据，定义为函数的返回值
    template: '<button @click="count++">按钮:{{ count }}</button>'
    // 组件的标签模板
})
```

- 接下来可以在任何Vue接管的元素中使用该组件，
  - ```

        <button_show></button_show>

    ```
  - ```
    window.onload = function () {
        var vm = new Vue({
            el: "#container",
        })
    }
    ```

data必须是函数

> 组件就是vue的实例，所有vue实例中属性和方法，组件中也可以用；
>
> 其中data属性必须是一个函数，因为组件会重复使用在多个地方，为了使用在多个地方的组件数据相对独立，data属性需要用一个函数的返回值来将数据处理为不同的每个个体

Prop传递数据

> Prop 是你可以在组件上注册的一些自定义特性。
>
> 当一个值传递给一个 prop 特性的时候，它就变成了那个组件实例的一个属性。
>
> 为了给组件传递数据，我们可以用一个 `props` 选项将一些特性值列举在其中

- ```
  <script type="text/javascript">
      Vue.component("myp",{
          props: ["content","like"], // 需要两个外界传入的值
          template: "<p :class='like'>{{ content }}</p>"
          // 组件绑定未来要接受的变量，要用到v;bind
      })
      window.onload = function(){
          var vm = new Vue({
              el: "#container",
              data: {
                  content: '这是p段落的文本',
                  like: 'beauty', // 要传递的变量
              }
          })
      }
  </script>
  ```
  ```

- ```
  .beauty{
      width: 100px;
      color: red;
      background: green;
  }
  ```

- ```
  <div id="container">
      <myp :like="like" :content="content"></myp>
      <!-- 传递到组件中 -->
  </div>
  ```

注册私有组件

组件开发自动化工具

- > **Node.js**
  >
  > Node.js是一个新的后端(后台)语言，它的语法和JavaScript类似，所以可以说它是属于前端的后端语言
  >
  > **运行环境：后端语言一般运行在服务器端，前端语言运行在客户端的浏览器上**
  >
  > **功能：后端语言可以操作文件，可以读写数据库，前端语言不能操作文件，不能读写数据库。**

- Node.js如果安装成功，可以查看Node.js的版本,在终端输入如下命令
  - ```
    node -v
    npm
    ```

- 全局安装vue脚手架，vue-cli，这玩意儿可以自动生成项目模板
- ```
  vue-cli
  cnpm install --global vue-cli
  ```

单页应用 SPWA

> 单页Web应用（**single page web application**，SPWA），就是将系统所有的操作交互限定在一个web页面中。
>
> 单页应用程序 (SPA)是加载单个HTML页面，系统的不同功能通过加载不同功能组件的形式来切换，不同功能组件全部封装到了js文件中，这些文件在应用开始访问时就一起加载完；
>
> 整个系统在切换不同功能时，页面的地址是不变的，系统切换可以做到局部刷新，也可以叫做无刷新，这么做的目的是为了给用户提供更加流畅的用户体验

- 通过`vue-cli`脚手架开启一个项目：
  - ```
    vue init webpack myproject
    ```

  - ```
    - Project name: 项目名称，如果不需要就直接回车。注：此处项目名不能使用大写。
    - Project description: 项目描述，直接回车
    - Author: 作者
    - vue build: 构建方式 默认即可
    - install vue-router? 是否安装vue的路由插件
    ```

```
- Use ESLint to lint your code? 是否使用ESLint检测你的代码?
    (ESLint 是一个语法规则和代码风格的检查工具，可以用来保证写出语法正确、风格统一的代码。)

- Pick an ESLint preset:选择分支风格
    1.standard(https://github.com/feross/standard) js的标准风格
    2.Airbnb(https://github.com/airbnb/javascript) JavaScript最合理的方法，这个github
地址说是JavaScript最合理的方法
    3.none (configure it yourself) 自己配置

- Setup unit tests? 是否安装单元测试
- Pick a test runner 选择一个单元测试运行器
    1.Jest (Jest是由Facebook发布的开源的、基于Jasmine的JavaScript单元测试框架)
    2.Karma and Mocha
    3.none

- Setup e2e tests with Nightwatch(Y/n)?是否安装E2E测试框架NightWatch
    (E2E，也就是End To End，就是所谓的“用户真实场景”。)

- Should we run 'npm install' for you after the project has been created?
项目创建后是否要为你运行“npm install”?
    yes,use npm(使用npm)
    yes,use yarn(使用yarn)
    no,I will handle that myself(自己操作)
```

- 启动开发服务器：
  - ```
    cd myproject // 进入目录
    cnpm install // 安装依赖
    cnpm run dev // 开启服务
    ```

- 项目目录介绍
  - ```
    - src 主开发目录，所有的单文件组件都会放在这个目录下
    - static 项目静态目录，所有的css、js都会放在这个文件夹下
    - dist 项目打包发布文件夹，最后要上线单文件夹项目都在这个文件夹中
    - node_modules node的包目录
    - config 配置目录
    - build 项目打包时依赖的目录
    ```

单文件组件

> 将一个组件相关的html结构，css样式，以及交互的JavaScript代码从html文件中剥离出来，合成一个文件，这种文件就是单文件组件，相当于一个组件具有了结构、表现和行为的完整功能，方便组件之间随意组合以及组件的重用，这种文件的扩展名为`.vue`，比如：`menu.vue`
>
> 组件文件一般定义在src目录下的components文件夹里

- template标签定义HTML部分

```
<teamplate>
    <div class="" @click="">
        <label>
            账号
            <input type="text">
        </label>
    </div>
</teamplate>
```

- js写成模块导出的形式
  - ```
    <script>
        // 使用export default命令，为模块指定默认输出
        export default{
            data: function(){
                return {
                    name:"张三",
                    age:16,
                }
            }
        }
    </script>
    ```

- 样式中的编写，如果含有scope关键字，表示这些样式是组件局部的，
  - ```
    <style scoped>
    .beauty{
        width:100px;
        line-height:50px;
        border-bottom:1px solid #ddd;
        margin:0px auto;
    }
    </style>
    ```

路由

- > 当拥有一个组件文件时，要在项目的src目录下的router目录下的index.js文件下进行组件的路由加载配置
  >
  > 在导入组件文件时，可以使用`@`符号，代表从src目录起
  >
  > 比如：*import* index *from* '@/components/index'

- ```
  import Vue from 'vue'
  import Router from 'vue-router'
  import HelloWorld from '@/components/HelloWorld'
  import first from '@/components/first' // 从组件目录下导入组件文件，不需要加后缀
  import index from '@/components/index'

  Vue.use(Router)

  export default new Router({
    mode: 'history',
    routes: [
  ```
  ```

```
    {
      path: '/',
      component: index,
    },
    {
      path: '/first', // 访问路径
      component: first
    }
  ]
})
```

- 当配置好路由之后，需要在最主要的App.Vue文件下进行连接引入：

  通过`<router-link to="连接地址">首页</router-link>`标签进行连接引入

  通过`<router-view></router-view>`标签进行路由加载，可以简写为：`<router-view/>`

- ```
  <template>

      <router-link to="/">首页</router-link>
      <router-link to="/first">第一个页面</router-link>
      <router-view></router-view>

  </template>

  <script>
  </script>

  <style>
  </style>
  ```

- 在App.Vue文件下的`template`标签处如果已经引入了其他跳转连接；

  那么在子组件的`template`部分不需要在进行引入
  ```