01.Vue3 入门和安装

学习要点：

1.Vue3.x 简介
2.教学方式
3.安装测试

本节课我们来开始了解 Vue3.x 渐进式前端框架，学会安装并测试。

一. Vue3.x 简介

1. Vue 一款基于 JavaScript 的构建用户界面的渐进式框架，版本为 3.x；
2. 所谓渐进式，就是按需分配，用多少，就引入多少，由开发者自行决定；
3. Vue 的核心库只关心视图层，利于上手，还有利于和其它库整合使用；
4. 那么，学习这款框架需要哪些基础呢，具体如下：
   (1) . HTML5/JavaScript (必须)；
   (2) . 任意一门前端库或框架，有后端语言基础更佳 (可选)；

PS：如果已有了一门或以上的前端框架或库的基础，可以迅速学起；
PS：如果这是你第一门前端框架，需要不时的停顿或查阅资料帮助理解；

二. 教学方式

1. Vue3.x 的教学方式，目前分为三种：
   (1) .从 Vue2.x 升级到 Vue3.x，只讲迁移方式和改变，不再讲解基本语法；
   PS：因为你已经有了 Vue2.x 基础，所以没必要重复讲解，课程的规划可以偏底层一些，重点讲解新特性和与旧版本的编码区别，以及性能提升的区别；

   (2) .假设你可能没有 Vue2.x 基础或有 Vue2.x 基础，但还是会讲解基础语法；
   PS：此时，可以直接使用脚手架工具 Vue Cli4.5+来演示课程；

   (3) .不做任何假设你的基础，从零开始讲解 Vue3.x 的语法基础，不采用脚手架工具；
   PS：手册上，对新手也是建议使用 CDN 引入的方式，避免脚手架一顿乱七八糟的东西；
   PS：因为脚手架需要有 Webpack 基础才能理解，只是为了演示语法显得很多余；
   PS：除非必须使用，在核心篇采用 CDN 或下载后引入的方式进行教学；

   PS：Vue3.x 核心篇采用第三种教学方式，而阶段二工具篇会讲解基于 Vue3.x 的脚手架；

三. 安装测试

1. 如何使用 Vue3.x，官方给出了几种方式，其实上面已经大概说了，具体：
   (1). CDN 方式引入，网络不好，下载后本地引入即可；

```
<script src="https://unpkg.com/vue@next"></script>
```

或

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

(3) . npm 结合 webpack 等前端构建工具，命令为：

```
npm install vue@next
```

(4) . Cli 脚手架工具，官方提供的脚手架工具，命令为：

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

(5) . Vite，官方提供的前端构建工具，命令为：

```
npm init vite-app <project-name>
```

PS：以上提前安装好演示即可，脚手架是阶段二的内容；
PS：至于 npm 的安装，以及国内镜像，在 Webpack 课程第一节讲解过；
PS：使用 CDN 方式，采用的是 file://模式，想要 http://模式，需要 Node.js 静态服务器；
PS：使用 file://一样可以测试学习，阶段二会讲解，如果想提前用，可参考手册，如下：
https://cli.vuejs.org/zh/guide/deployment.html

2. 关于 Vue Devtools 工具，目前支持 Vue3.x 是 6.0.x 的 Beta 版本；
https://github.com/vuejs/vue-devtools/releases/tag/v6.0.0-beta.2

PS：支持 Vue2.x 的 5.3.x 不支持 Vue3.x，而 6.0.x 的 Beta 还有一些问题，可等更新；

3. 开发工具推荐 WebStorm 或 VScode，都是比较优秀的开发工具，这里选了前者；
4. 学习版本会提示信息，这里采用的是火狐浏览器(大家用谷歌一样的)，屏蔽即可；

02.渲染和挂载

学习要点：

1.渲染挂载

2.搜索方式

本节课我们来开始了解 Vue3.x 创建使用的声明渲染，了解数据的动态改变。

一. 渲染挂载

1. 前端框架基本都是为了简化：模版渲染、事件绑定和用户交互问题；
2. Vue 的 MVVM 模型：即视图(View)-视图模型(ViewModel)-模型(Model)；
3. Vue 的操作核心：即使用模版语法声明式的将数据渲染进 DOM 中去；
4. 比如，创建一个 DOM 结构，内部需要一个动态变化数据使用 Vue 处理；

```
<div id="app">
    计数器：0                                    // 这里的 0 需要动态改变
</div>
```

5. 那么，首先引入 Vue.js，然后在下方编写需要的 JS 代码；

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

6. 其次，在引入 Vue 的下面编写一个对象，用于实现 0 的动态改变；

```
// 声明一个选项对象
const App = {
    // 初始数据
    data() {
        return {
            counter : 100
        }
    }
}
```

PS：选项对象内部包含初始数据，方法、事件、声明周期等等要素；

7. 这个选项对象 App，是需要当作参数传递到 Vue 的全局 API：createApp()；
8. createApp()方法会返回应用实例，再通过这个实例来挂载对应的 DOM 即可；

```
// 全局 API 对象.创建实例(App).挂载('#app')
Vue.createApp(App).mount('#app')
```

PS：如果想查阅这个全局 API，更多的文档了解，选择 API 参考：

https://vue3js.cn/docs/zh/api/global-api.html#createapp

9. 最后一步：需要通过 Vue 改变的 Counter 值在 DOM 中渲染出来；

```
<div id="app">
    计数器：{{counter}}
</div>
```

PS：{{插值}}，这里用双大括号表示，插值对应的是 data()里的 counter 属性；

二. 搜索方式

1. 一般来说，课程中只会拎出重点来阐述，建议学习完毕后再过一遍文档补全；
2. 对于盲区部分，比如 MVVM 是啥？前期其实不用理解，学完后会后能理解七七八八；
3. 强迫症，可以去搜索 MVVM 理解下即可；但学完后，回来再理解效果更好；
4. Vue2.x 中使用 data ：{}的方式，现在是 data() { return }，怎么补全理解？
5. 可通过搜索：vue data return 三个关键字看文档理解；

03.文本插值及指令

学习要点：

1.文本插值

本节课我们来开始学习 Vue3.x 的模板中文本插值的使用方法。

一. 文本插值

1. 数据绑定最常用的方式就是：“Mustache”语法，即双花括号进行文本插值；

```
<div id="app">
    计算器：{{counter}}
</div>
```

PS：这里也可以使用 JSX 语法，而不使用模板的文本插值，有兴趣可以参考手册；

2. 绑定在模板中的插值{{counter}}，模板是可变的，可以理解为变量；
3. 如果要设置成常量那种不可改变的量，可以使用指令：v-once；

```
<div id="app">
    计算器：<span v-once>{{counter}}</span>
</div>
```

PS：为了测试，可在控制台或 log 输出的方式来验证数据是否能够被改变；

```
// 获取到对象名
const vm = Vue.createApp(App).mount('#app')
vm.counter = 200            // 赋值操作
console.log(vm.counter)
```

PS：关于 vm 的详解，在手册的“应用&组件实例”部分，后面会逐步了解到；

4. 文本插值解析出来的是普通文本，而非 HTML 代码，测试代码如下：

```
rawContent : '<span style="color:red;">Vue3.x</span>'
是否字体有变：{{rawContent}}
```

PS：这里得到的结果是，并未解析 HTML 代码，如果要解析，需要使用指令：v-html；

```
是否字体有变：<span v-html="rawContent"></span>
```

PS：强烈建议不要在动态渲染 html 内容，除非非常信任，否则容易遭受 XSS 攻击；

5. 模板的文本插值是不能在 HTML attribute(属性)，先看下错误写法：

```
<span class="{{className}}">Vue3.x</span>
```

PS：attribute 一般不翻译，因为歧义较多，理解成属性或特性或其它都行；

6. 如果要让 HTML 的 attribute 支持模板插值，需要使用 v-bind 指令：

```
<span v-bind:class="className">Vue3.x</span>
```

PS：可通过控制台的查看器观察 HTML 的变化；

7. 在模板语法中，支持 JavaScript 表达式，比如：运算、函数等；

```
<!--注意：语句不支持，比如赋值语句，条件判断等-->
{{counter+ 1}}
{{true ? '真' : '假'}}
{{'bbbb@163.com'.split('@')}}
```

04.指令和缩写方法

学习要点：

1.指令

2.缩写方法

本节课我们来开始学习 Vue3.x 的指令和其缩写的方法。

一. 指令

1. 回顾下上节课，我们已经学习了：v-once、v-html、v-bind 等指令；
2. 指令：即在 HTML 中带有 v-前缀的特殊 attribute(属性)，值为单个 JS 表达式；
3. 比如：我们想让一段内容是否可见，可以通过 v-if 的值为 false 对其隐藏；

```
flag : false
<span v-if="flag">{{counter}}</span>
```

PS：可通过控制台的查看器来观察 HTML 的变化；

4. 有些指令可以接受一些“参数”，比如动态的设置 url 链接地址，采用 v-bind；

```
url : 'http://www.baidu.com'
<a v-bind:href="url">百度或搜狗</a>
```

PS：可以在控制台设置 vm.url = 'http://www.sogou.com'来改变 url 实现切换；

5. 上面的例子中 v-bind:href 中 href 看似固定，实则可以动态改变，具体如下：

```
<a v-bind:[attrname]="url">attr 动态改变</a>
attrname : 'href'
```

PS：这里的动态参数都用小写即可，否则会出错，在 attribute 还可以使用修饰符，后面再说；

二. 缩写方法

1. 在实际使用过程中，v-bind:和 v-on:(事件部分会讲)，会非常常用；
2. 而这两种质量后续会带各种参数和修饰符导致过于繁琐，故提供了简写方案：

```
// v-bind:
v-bind:href = :href
v-bind:[key] = :[key]

// v-on:
v-on:click = @click
v-on:[key] = @[key]
```

05.事件和计算属性

学习要点：

1.简单事件

2.计算属性

本节课我们来开始学习 Vue3.x 的事件的入门和计算属性的用法；

一. 简单事件

1. 上节课，我们最后谈到了一个新指令：v-on:click，点击触发事件；

```
<button v-on:click="counter++">+counter</button>
<button v-on:click="counter+=2">+counter</button>
```

PS：当我们点击了按钮即可实现累加 1，或累加 2 的操作；

2. 如果触发的事件业务逻辑较多，那么行内表达式是不够的，需要触发固定方法；

```
<button v-on:click="addCounter">counter+methods</button>
```

```
const App = {
    // 初始数据
    data() {},

    // 事件处理方法
    methods : {
        // 累加方法
        addCounter() {
            // 这里的 this 就是 vue 代理对象
            this.counter++
        }
    }
}
```

PS：所有的方法均可放在 methods 属性对象里，提供给 v-on 触发；

二. 计算属性

1. 对于插值可以进行简单运算，可一旦过于复杂，则模版维护将变得异常困难；

```
<!--两个插值拼装-->
{{firstName + lastName}}
```

```
firstName : 'Mr.',
lastName : 'Lee'
```

2. 上面的例子出现要使用两个插值，如果假设插值内部还要各种运算，极不方便；
3. 此时，我们可以用第二套方案，就是使用方法进行返回，把运算和显示整合起来；

```
<!--直接调用方法，但需要括号-->
{{fullName()}}

methods : {
    // 连接两个属性字符串
    fullName() {
        return this.firstName + this.lastName
    }
}
```

PS：第二套方案在复杂性高的情况下优于第一种，并且插值只调用一次即可；
PS：缺点也很明显，插值调用必须方法模式(有括号)，和属性方式容易混淆；
PS：其次，每次都必须执行方法，没有缓存会在复杂的情况下影响性能；

4. 那么，系统提供了第三种方法解决了这两个问题，就是计算属性；

```
<!--计算属性调用方法-->
{{fullName2}}

const App = {
    // 初始数据
    data() {},

    // 事件处理方法
    methods : {},

    // 计算属性
    computed : {
        // 连接三个属性字符串
        fullName2() {
            return this.firstName + this.LastName
        }
    }
}
```

PS：计算属性和属性一样的调用方式，而方法调用却需要括号，所以推荐用计算属性；
PS：计算属性具有缓存，当值没有改变时，不会重新执行方法，而去使用缓存；
PS：可以参考手册去了解一下计算属性缓存的相关细节，本人推荐需要在实战中体会更好；

06.Getter 和 Setter

学习要点：

1.Getter

2.Setter

本节课我们来开始学习 Vue3.x 的计算属性中的 Getter 和 Setter 方法；

一. Getter

1. 当我们输出 vm 对象的时候，发现属性字段有 Getter 和 Setter 方法；
2. Getter：即取值，Setter：即赋值，计算属性默认执行的是 Getter 取值方法；
3. 那么正常使用计算属性时，我们都是 return 返回一个最终结果即可；
4. 如果我们强制使用 Getter 方法获取，那么计算属性的格式改写如下：

```
// 计算属性
computed : {
    // 连接两个属性字符串
    fullName : {
        // Getter
        get() {
            return this.firstName + this.LastName
        }
    }
}
```

二. Setter

1. Setter 是赋值，我们可以在这里直接修改属性字段的内容，如下：

```
// Setter
set(value) {
    const str = value.split('.')
    this.firstName = str[0] + '.'
    this.lastName = str[1]
    console.log('set')
}
```

PS：Setter 会被先执行进行赋值，然后 Getter 执行取出，可通过控制台输出了解顺序；

PS：当没有赋值操作，则执行一次 get，当有赋值操作，会以 get->set->get 三次执行；

PS：赋值通过 vm.fullName = 'Miss.Wang'进行赋值；

```
// 可以直接对计算属性赋值
vm.fullName = 'Miss.Wang'
```

07.Class 和 Style 绑定

学习要点：

1.Class 绑定

2.Style 绑定

本节课我们来开始学习 Vue3.x 的 class 和 style 绑定的方法。

一. Class 绑定

1. 先复习一下前面课程中 v-bind:的用法：v-bind:class 或:class 简写即可；

```
<style>
    .red {
        color: red;
    }
</style>

<span v-bind:class="className">Class 绑定效果</span>
data() {
    return {
        className : 'red'
    }
},
```

PS：再重复一次，className 如果作为 v-bind:class 参数的值，是不需要加双括号的；

2. 如果想通过一个布尔值来实现，是否要加载这个样式，那么可以改写成如下：

```
// 在 data() 创建一个布尔属性
isRed : false

// 当 isRed 为 false：class=''，当为 true 时：class='red'，解构语法用{}包含
// {class 名称 ： 布尔属性}
<span v-bind:class="{red : isRed}">Css 绑定效果</span>
```

PS：这里属性 className 就没有被使用，并且支持多个 class 叠加，-号需单引号防解析错误；

```
v-bind:class="{red : isRed, 'big-font' : isBigFont}"
```

3. 判断 class 样式的布尔属性太多，且指令的参数值太复杂导致压力大，可简写：

```
<span v-bind:class="classObject">Css 绑定效果</span>
data() {
    return {
        counter : 100,
        classObject : {
            red : true,
```

```
            'big-font' : true
        }
    }
},
```

PS：使用计算属性来实现如上的功能：

```
// 计算属性
computed : {
    classObject() {
        return {
            red : true,
            'big-font' : true
        }
    }
}
```

4. 还有一种列表式数组方式：即你添加就有效果，不添加就没有效果，如下：

```
<span v-bind:class="[redClass, bigFontClass]">Css 绑定效果</span>

data() {
    return {
        redClass : 'red',
        bigFontClass : 'big-font',
    }
},
```

PS：列表式数组也支持类似布尔是否显示，通过数组的三元来实现；

```
<span v-bind:class="[isRed ? redClass : '', bigFontClass]">Css 绑定效果</span>
```

PS：三种方式第一次用建议选选其中一种，否则会比较混乱；学习后期，可根据场景查询使用；

二. Style 绑定

1. 有了 Class 绑定基础，Style 行内绑定就简单多了，方式也是类似：

```
// 解构绑定
<span v-bind:style="{color : styleColor}">Style 绑定效果</span>
// 绑定属性对象或计算属性绑定
<span v-bind:style="StyleObject">Style 绑定效果</span>
// 列表式数组绑定
<span v-bind:style="[redStyle]">Style 绑定效果</span>
```

08.条件判断渲染

学习要点：

1.条件显示指令

本节课我们来开始学习 Vue3.x 的条件判断渲染 v-if 的使用方式。

一. 条件显示指令

1. 先复习一下前面课程中 v-if 指令的简单用法：通过布尔属性隐藏或显示：

```
<div v-if="flag">我是否显示</div>

data() {
    return {
        flag : true
    }
},
```

2. 当然，也支持 v-else 的条件判断，具体如下：

```
<div v-if="flag">我是否显示</div>
<div v-else>404</div>
```

PS：如果布尔值为 false，则显示 404；

3. 如果有多个分支判断的话，支持加入 v-else-if 的判断比较，具体如下：

```
<div v-if="type === 'A'">A</div>
<div v-else-if="type === 'B'">B</div>
<div v-else-if="type === 'C'">C</div>
<div v-else>404</div>
```

PS：属性里设置 type : 'B'，最后显示 B；

4. 如果条件判断的区域是一大段代码，我们可以使用<template>来整合多个元素；

```
<template v-if="flag">
    <h2>标题</h2>
    <h4>小标题</h4>
    <div>内容.....</div>
</template>
```

5. 如果只是简单的切换显示或隐藏，可以使用 v-show，但不支持 else 和 template；

```
<div v-show="flag">show</div>
```

PS：v-if 是条件渲染，惰性的，适时的销毁和重建渲染的内容；

PS：v-show，就是 display 的隐藏显示切换；结论：时时频繁切换用 v-show，反之 v-if；

09.循环列表渲染

学习要点：

1.循环列表指令

本节课我们来开始学习 Vue3.x 的循环列表指令 v-for 的使用方式。

一. 循环列表指令

1. 使用 v-for 指令来渲染一个列表，其中 array 是数据源，item 是元素别名；

```
<ul>
    <li v-for="item in array">{{item.city}}</li>
</ul>

data() {
    return {
        array : [
            {
                city : '北京'
            },
            {
                city : '上海'
            },
            {
                city : '广州'
            },
            {
                city : '深圳'
            }
        ]
    }
},
```

PS：也可以在遍历的元素项中设置 index，来获取有序编号，从 0 开始的；

```
<li v-for="(item, index) in array">{{index}}.{{item.city}}</li>
```

PS：这里的 in 可以使用 JS 迭代器语法 of：(item, index) of array；

PS：index 参数在第二位置，item 在第一个参数位置；

2. 如果数据是一个对象，内部包含多个属性，那也可以使用 v-for 遍历；

```
<ul>
    <li v-for="item in object">{{item}}</li>
</ul>
```

```
data() {
    return {
       object :  {
           name : 'Mr.Lee',
           gender : '男',
           age : 100
       }
     }
}
```

PS：这里支持(value, name, key)的参数，支持单独输出；

```
<li v-for="(value, name, key) in object">{{key}}.{{name}}.{{value}}</li>
```

3. 支持<template>进行多代码块的渲染方式，比如：

```
<ul>
    <template v-for="(item, index) in array">
        <li>{{index}}</li>
        <li>{{item.city}}</li>
    </template>
</ul>
```

4. v-for 支持数值循环渲染，如下：

```
<span v-for="n in 10">{{n}}</span>
```

5. v-if 和 v-for 不推荐在同一元素上使用，因为 if 比 for 权限高，导致无法访问；
6. 那可以结合<template>来实现使用 if 来判断渲染，具体如下：

```
<ul>
    <template v-for="item in array">
        <li v-if="item.city !== '上海'">{{item.city}}</li>
    </template>
</ul>
```

10.key 和数组检测

学习要点：

1.数组检测

2.key 问题

本节课我们来开始学习 Vue3.x 的 v-for 中的数组检测和 key 的问题。

一. 数组检测

1. 为了更好的渲染视图，Vue 提供了如下的变更方法，执行后可渲染视图更新；

```
push()/pop()/shift()/unshift()/splice()/sort()/reverse();
```

PS：在控制台输入 app.array.push({city : '重庆'})，视图立马渲染更新；

2. 如果想替换掉数组的某个选项，可以如下操作：

```
// 替换掉第二条的内容
vm.array[1] = {city : '南京'}
```

PS：Vue 也提供了比如 filter()/concat()/slice()来替换并返回一个新数组；

```
vm.array.slice(1,4)
```

二. key 问题

1. 我们要创建一个按钮，来点击添加一个选项，具体如下：

```
<button v-on:click="add">添加</button>
```

```
methods : {
    // 在项目顶端添加一个记录
    add() {
        this.array.unshift({city : '重庆'})
    }
},
```

PS：此时在渲染端，做一个复选按钮；

```
<li v-for="item in array"><input type="checkbox">{{item.city}}</li>
```

PS：当我们单击按钮时，在列表顶端添加了一个选项，但复选的勾却错位了；

2. 此时，我们需要对每一个项目限定一个唯一的 id 指明来解决这个问题；

```
{
    id : 1,
    city : '北京'
},
```

```
add() {
    this.array.unshift({id : 5, city : '重庆'})
}

// v-bind:key 来绑定 id
<li v-for="item in array" v-bind:key="item.id">
          <input type="checkbox">{{item.city}}
</li>
```

11.事件处理能力

学习要点：

1.事件处理

2.事件修饰符

本节课我们来开始学习 Vue3.x 的事件处理方法以及事件的修饰符能力。

一. 事件处理

1. 先复习一下前面用过的事件能力，直接使用和使用方法，如下：

```
<button v-on:click="counter++">counter+</button>
<button v-on:click="addCounter">counter+</button>

methods : {
    addCounter() {
        this.counter++
    }
},
```

2. 我们可以通过点击执行方法，并且处理事件对象，具体如下：

```
methods : {
    // event 对象，名称自定义，e 也行
    addCounter(event) {
        console.log(event.target.innerText)
    }
},
```

PS：如果不传递参数，默认接受的事件对象名是 event；

PS：如果不是 event 名称，比如 e，则需要显式的传递事件对象参数；

3. 如果执行方法时，有普通参数传递，默认情况下 event 也会隐式传递的；

```
<button v-on:click="addCounter(5)">counter+5</button>
addCounter(num) {
    this.counter += num
    console.log(event)        // 隐式传参事件对象名必须是 event
}
```

PS：如果要强行接受，是无法接收到事件对象参数的，比如：

```
addCounter(e, num)  或  addCounter(num, e)
```

PS：只有显式进行事件对象参数传递，才能有效接收；

```
<button v-on:click="addCounter(5, $event)">counter+5</button>
addCounter(num, e)
```

PS：事件对象用途属于 JS 基础知识，比如：阻止冒泡、取消默认行为，获取触发元素内容等；

二．事件修饰符

1. 系统提供了常用的事件对象修饰符，让你直接定义视图中事件的执行方式；

```
<div v-on:click="counter++">
    <button v-on:click.stop="addCounter(5, $event)">counter+5</button>
</div>

// 只执行一次
<button v-on:click.once="counter++">counter+</button>
```

PS：这里使用了.stop 修饰符，表示阻止冒泡；如果不阻止，每次会+6；全部修饰符如下：

```
.stop / .prevent / .capture / .self / .once / .passive
https://vue3js.cn/docs/zh/guide/events.html#事件修饰符
```

3. 系统还提供了按键事件修饰符，比如回车键按下触发；

```
<button v-on:keyup.enter="counter++">counter+</button>
.enter / .tab /.delete / .esc / .space / .up / .down / .left / .right
```

12.表单输入绑定

学习要点：

1.输入绑定

本节课我们来开始学习 Vue3.x 的表单输入绑定的使用方法。

一. 输入绑定

1. 使用 v-model 指令可以实现 input、textarea 和 select 的双向绑定；
2. 它本质上是一个语法糖，负责监听用户输入数据并进行数据的更新操作；
3. 我们使用 input 来测试 v-model 的使用方法，并实现时时更新数据；

```
<input type="text" placeholder="请输入内容" v-model="message">
<div>{{message}}</div>

data() {
    return {
        message : 'Hello, Vue3.x~'
    }
},
```

PS：当我们在 input 文本框编辑的时候，div 中的内容会时时更新；

PS：v-model 对 input 和 textarea 处理时，会抛出 input 事件来时时更新；

4. 单个复选框，可以直接绑定布尔值 true 和 false，自动识别是否勾选；

```
<input type="checkbox" id="checkbox" v-model="checked">
<label for="checkbox">{{checked}}</label>

return {
      checked : true,
}
```

PS：v-model 在 checkbox 和 radio 处理时，会抛出 change 事件来时时更新；

5. 多个复选框，可以通过绑定实现选择获取信息；

```
<input type="checkbox" id="a" value="Mr.Lee" v-model="checkedNames">
<label for="a">Mr.Lee</label>

<input type="checkbox" id="b" value="Mr.Wang" v-model="checkedNames">
<label for="b">Mr.Wang</label>

<input type="checkbox" id="c" value="Mr.Zhang" v-model="checkedNames">
<label for="c">Mr.Zhang</label>
```

```
<div>{{checkedNames}}</div>

return {
      checkedNames : [],
}
```

6. 单选按钮，绑定后和复选框实现一样的效果；

```
<input type="radio" id="one" value="男" v-model="gender">
<label for="one">男</label>

<input type="radio" id="two" value="女" v-model="gender">
<label for="two">女</label>

return {
      gender : '男'
}
```

7. 下拉选择框，分单选和多选，基本同上；

```
<select v-model="selected">
    <option>北京</option>
    <option>上海</option>
    <option>广州</option>
    <option>深圳</option>
</select>

<div>{{selected}}</div>

return {
      select : ''
}
```

PS：v-model 在 select 处理时，会抛出 change 事件来时时更新；

13.值绑定和修饰符

学习要点：

1.值绑定

2.修饰符

本节课我们来开始学习 Vue3.x 的表单中 value 值的绑定和修饰符功能。

一. 值绑定

1. 使用 v-model 操作 input 等，是直接操作和绑定 value 值的；
2. 但是对于 checkbox 等表单，是操作它的 checked 等首选项的；
3. 所以，此时我们需要对这些表单元素的 value 值进行绑定操作；

```
<input type="checkbox" id="checkbox" v-model="checked">
<div>{{checked}}</div>

return {
    checked : true
}
```

PS：虽然可以进行双向绑定切换，但本质上这个 checkbox 并没有 value 值；

PS：我们可以通过控制台查看器观察，此时我们还需要进行值绑定操作；

```
<input type="checkbox" id="checkbox" v-model="checked"
                                        v-bind:value="checked">
```

PS：单选框，下拉框均可以使用这种方法给 value 赋值；

二. 修饰符

1. 有时我们不需要数据绑定的时候逐个触发，可以使用.lazy 修饰符；

```
<!--比如回车或失去焦点触发-->
<input type="text" v-model.lazy="message">
{{message}}
```

2. 使用.number 修饰符将输入的内容转换为 number 数值类型；

```
<!--默认输入的类型是字符串，设置为数值-->
<input type="text" v-model.number="number" type="number">
{{typeof number}}
```

3. 使用.trim 清理掉内容左右的空格；

```
<input type="text" v-model.trim="message">
```

14.组件的概念

学习要点：

1.组件概念

2.简单组件

本节课我们来开始学习 Vue3.x 的组件的基本概念以及简单的使用；

一. 组件概念

1. 组件是一种封装可复用的集合，通过组件化，将更好的完成繁杂的需求；
2. 从已知的概念中理解：类似函数或方法封装的重复代码，或 HTML 的代码块引入；
3. 组件化特点，具体如下：
   (1) .拥有唯一性的组件名称，方便调用；
   (2) .以组件名称为 HTML 元素标签形式存在，扩展了 HTML；
   (3) .组件可以复用，且组件与组件之间互不干涉；

二. 简单组件

1. 为了方便理解，我们清理掉之前的全部代码，重新编写如下：

```
// 创建一个 Vue 应用实例
const app = Vue.createApp({})

// 定义一个全局组件
app.component('button-counter', {
    // 数据
    data() {
        return {
            counter : 100
        }
    },

    // 模板
    template : `
        计算器：{{counter}}
        <button v-on:click="counter++">
            counter++
        </button>
        <br>
    `
})

// 挂载
app.mount('#app')
```

2. 而对于 HTML 调用的部分，可复用的能力体现出来，如下：

```
<div id="app">
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
</div>
```

15.组件的注册

学习要点：

1.规则说明

2.全局局部

本节课我们来开始学习 Vue3.x 的组件注册的基本规则以及全局和局部的注册。

一. 规则说明

1. 上一节课，我们简单的进行了组件的注册，了解了如何创建注册和复用方法；
2. 那么，我们探讨一下组件名称的一些规则，具体如下：
   (1) . 组件名称是唯一的；
   (2) . 命名规则可以采用 kebab-case(button-counter)，减号隔开；
   (3) . 命名规则也可以用 PascalCase(ButtonCounter)，单词首字母大写；
   (4) . 这里推荐使用 kebab-case，全小写并用减号隔开，遵循 W3C，防止冲突；

   PS：这里防止冲突是，防止与未来的 HTML 扩展元素标签的冲突；

二. 全局和局部

1. 上一节课，我们定义了一个全局组件，用连缀方法再写一遍，如下：

```
// 定义一个全局组件
Vue.createApp({}).component('button-counter', {}).mount('#app')
```

2. 有全局注册，就有局部注册的方法，局部注册的方法如下：

```
// 可以理解为根组件，最顶层的组件
const app = Vue.createApp({
    data() {
        return {}
    },

    // 定义局部组件(也就是这个根组件的子组件)
    components : {
        'button-counter' : {
            // 数据
            data() {
                return {
                    counter : 100
                }
            },

            // 模板
            template : `
                计算器：{{counter}}
```

```
            <button v-on:click="counter++">counter++</button>
            <br>
        `
    },

    'button-counter2' : {},
    'button-counter3' : {},
  }
})
```

3. 当然，我们可以把组件的参数二分离出来，让代码更清晰且更好管理；

```
// 定义一个局部组件的对象内容部分
const buttonCounter = {}

components : {
    'button-counter' : buttonCounter,
}
```

PS：在调用上，并没有差异，而我们推荐使用局部组件，因为在以后的项目构建上更具优势；

16.组件 Prop 通信

学习要点：

1.实际问题

2.数据父传子

本节课我们来开始学习 Vue3.x 的组件中父组件给子组件传递数据的方法。

一. 实际问题

1. 根组件(父组件)的 data()数据，子组件在目前的知识是无法获取的；
2. 我们如果想要子组件和父组件同时正确的获取数据，采用如下方法：

```
// 定义一个局部组件
const htmlA = {
    // 数据
    data() {
        return {
            message : '子组件-Vue3.x~'
        }
    },

    // 模板
    template : `
        <div>{{message}}</div>
    `
}

// 创建一个 Vue 应用实例，这个本身也是一个组件，根组件，最顶层的
const app = Vue.createApp({
    data() {
        return {
            message : '父组件-Vue3.x~'
        }
    },

    // 创建一个局部组件，也是一个子组件
    components : {
        'html-a' : htmlA
    }
})

<div id="app">
    <html-a></html-a>
</div>
```

PS：上面的例子中暴露的问题，就是子组件无法直接使用父组件的 message 属性；

PS：所以只能在 HTML 端进行设计，但这样做，父组件的内容无法和子组件参与逻辑；

```
<div id="app">
    <div>{{message}}</div>
    <html-a></html-a>
</div>
```

二．数据父传子

1. 根据上面的问题，我们最好是能够直接从子组件获取到父组件的数据内容；
2. 这时，Vue3.x 提供了一个 props 来获取父组件给子组件传递的值；

```
// 定义一个局部组件
const htmlA = {
    // 数据
    data() {
        return {
            message : '子组件-Vue3.x~'
        }
    },

    // props
    props : ['abc'],

    // 模板
    template : `
        <div>{{abc}}</div>
        <div>{{message}}</div>
    `
}

<div id="app">
    <html-a v-bind:abc="message"></html-a>
</div>
```

PS：props 里定义数组，表示可以定义多个；在 HTML 端使用 v-bind:abc 来获取父组件对应值；

PS：这里采用了 abc 表示 props 的元素名称可以自定义，如果定义成 parent-message，则：

```
props : ['parentMessage'],
// 模板
template : `
    <div>{{parentMessage}}</div>
    <div>{{message}}</div>
`

<html-a v-bind:parent-message="message"></html-a>
```

17.组件 Prop 类型

学习要点：

1.静态 Prop

2.Prop 类型

本节课我们来开始学习 Vue3.x 的组件中 Prop 类型的设置方法。

一. 静态 Prop

1. 我们之前使用的 v-bind:parent-message="message"，这是动态 Prop；

```
<html-a v-bind:parent-message="message"></html-a>
```

PS：动态 Prop 可以将父组件的数据属性传递到子组件中；

2. 那么静态 Prop 又是什么意思？其实就是直接通过 HTML 端给子组件赋值；

```
<html-a parent-message="直接传入静态字符串"></html-a>
```

PS：没有使用 v-bind 就是静态 Prop，然后将字符串直接赋值给 parentMessage；

二. Prop 类型

1. 上一节课，我们采用了 props : ['abc', 'def']这种形式构建数据父传子；
2. 而这种方式，是以字符串的形式来进行值传递的，其实还可设置数据类型；

```
props: {
    title: String,
    likes: Number,
    isPublished: Boolean,
    commentIds: Array,
    author: Object,
    callback: Function,
    contactsPromise: Promise    // 或任何其他构造函数
}
```

PS：手册摘入的所有数据类型；

3. 当我们传入数值时，我们可以通过 Prop 类型进行限制和验证；

```
<!--传入数值-->
<html-a v-bind:parent-message="100"></html-a>

//prop 通信
props : {
    parentMessage : Number
},
```

PS：当传入字符串的时候，不需要 v-bind 的，当传入数字、数组、布尔和对象时则需要；
PS：当传入不匹配的类型时，控制台会提示警告；null 和 undefined 会通过任何类型检查；

4. 在父类设置一个对象，然后属性设置一个值传递给子组件；

```
// 第一步，父组件
return {
    message : '父组件 Vue3.x~',
    post : {
        likes : 100
    }
}

// 第二步，HTML 端
<html-a v-bind:parent-message="post.likes"></html-a>

// 第三步，子组件
props : {
     parentMessage : Number
},
```

5. 当直接传递一个对象的时候，我们可以在插值通过对象的调用方法获取值；

```
<html-a v-bind:parent-message="{name : 'Mr.Lee'}"></html-a>

// 模板
template : `
    <div>{{parentMessage.name}}</div>
`
```

PS：当然，也可以直接传递父组件的对象，一个意思；而数组、布尔等类型都一样自行练习；

6. 也可以传入对象的所有 property(属性)，然后通过查看器查看节点变化；

```
<html-a v-bind="post"></html-a>
```

18.组件单向数据流

学习要点：

1.单向数据流

本节课我们来开始学习 Vue3.x 的组件中 Prop 的单向数据流。

一. 单向数据流

1. 父组件的 data 值更新后通过 props 选项交给子组件进行渲染，反之则不行；
2. 这就是单向数据流(单向下行绑定)，不能通过子组件来改变父组件的状态；
3. 这样做的是为了防止父组件发生改变后，数据流变得难以理解；
4. 父组件更新时，子组件所有 props 值也会更新，你不能改变子组件的 props 值；
5. 通过控制台输入 vm.message 赋值，子组件的自动渲染刷新；

```
vm.message = '改变父组件'
```

5. 我们可以通过设置一个父组件的计数器属性，并且通过子组件去更改它；

```
data() {
    return {
        message : '父组件 Vue3.x~',
        counter : 100
    }
},

//prop 通信
props : ['parentMessage', 'parentCounter'],

// 模板
template : `
    <div>{{parentCounter}}</div>
    <button v-on:click="parentCounter++">counter+</button>
`,
```

PS：此时控制台会告诉你，Props 是只读，无法修改；

6. 那么，我们如何改变子组件这个值呢？可以考虑只改变子组件的值，父组件不变；

```
// 数据
data() {
    return {
        message : '子组件 Vue3.x~',
        childCounter : this.parentCounter    // this 可以调用 props 内的属性
    }
},
```

```
// 模板
template : `
    <div>{{counter}}</div>
    <div>{{childCounter}}</div>
    <button v-on:click="childCounter++">counter+</button>
`,
```

PS：上面的写法就是通过使用子组件的 data 属性，让 props 父组件的值先赋值过来；
PS：然后通过操作这个子组件的 childCounter 实现累加，而并未修改父组件的 Counter；
PS：当然，通过计算属性 computed 或方法 methods 来修改 childCounter 也是可以的；

19.组件 Prop 验证

学习要点：

1.Prop 验证

本节课我们来开始学习 Vue3.x 的组件中 Prop 类型验证的方法。

一. Prop 验证

1. 接着 Prop 类型继续往下探讨，重点研究一下 Prop 类型验证的问题，如下；

```
<!--传入数值-->
<html-a v-bind:parent-message="100"></html-a>

//prop 通信
props : {
    parentMessage : Number
},
```

PS：上面例子是之前的代码，意味必须传递一个数值，否则会提示类型不匹配；

2. 当然，除了限定一个类型之外，还可以设置限定多个类型，比如：

```
props : {
    parentMessage : [Number, String]
},
```

PS：这样，传递过来的类型数值和字符串均可通过；

3. 如果没有值传入的时候，也可以给其设置一个默认值；

```
props : {
    parentMessage : {
        type : [Number, String],
        default : 100
    }
},
```

PS：默认值也可以通过函数进行返回；

```
props : {
    parentMessage : {
        type : [Number, String],
        default : function () {
            return 200
        }
    }
},
```

4. 上面的例子中使用的 type 类型检查，全部类型检查如下：
   String/Number/Boolean/Array/Object/Date/Function/Symbol

   PS：如果你创建了一个对象，要求传递的检查这个对象类型，也是支持的；

5. 使用自定义验证函数 validator 来处理传递过来的数据；

```
props : {
    parentMessage : {
        validator(value) {
            return ['a', 'b', 'c'].indexOf(value) !== -1
        }
    }
},
```

   PS：此时，传递过来的值必须是 a,b,c 中的一种，否则提示不匹配；

20.组件自定义事件

学习要点：

1.自定义事件

2.验证抛出的事件

本节课我们来开始学习 Vue3.x 的组件中自定义事件的方法。

一. 自定义事件

1. 系统内置的事件比如：click、change 等等，也可以自定义事件；
2. 而组件中的自定义事件，可以满足之前无法实现的需求：更改父组件的值；
3. 由于这块知识点比较绕，第一次学习可能会摸不招头脑，所以，我按照步骤走一遍；

(1). 先创建两个子组件 htmlA 和 htmlB，都用 props 加载父组件的内容；

```
const htmlA = {...}
const htmlB = {...}

// 父组件绑定两个子组件
components : {
    'html-a' : htmlA,
    'html-b' : htmlB
},
```

(2). 在其中一个子组件，比如 htmlA 中定义一个自定义事件，和 props 的定义方式一样；

```
// 定义子组件的自定义事件
emits : ['child-event'],
```

(3). 通过子组件<html-a>中使用 v-on 以自定义事件去执行父组件的一个方法；

```
<html-a v-on:child-event="parentFn" v-bind:parent-message="message"></html-a>
```

(4). 在子组件模板中添加一个按钮，点击后来执行这个自定义事件，并抛出一个值；

```
// button 按钮点击后触发一个方法，方法里执行自定义事件，并抛值
template : `
   <div>{{parentMessage}}</div>
   <div>{{message}}</div>
   <button v-on:click="childClick(message)">{{message}}</button>
`,

//方法
methods : {
    childClick(message) {
        this.$emit('child-event', message)
    }
}
```

(5). 最后一步了：执行父组件的一个方法，并获取子组件传递的值，然后更改父组件内容；

```
//这里是父组件的方法
methods : {
    parentFn(value) {
        this.message = value
        console.log('子组件触发了我~')
    }
}
```

PS：一套流程下来，我们终于可以修改父组件的内容了；内容较绕，需要多多理清；

二. 验证抛出的事件

1. 这个和 props 用法的一样的，具体如下：

```
emits : {
    // 不验证
    //'child-event' : null

    // 复杂验证
    'child-event' : (value) => {
        if (value !== 'abc') {
            console.log('验证失败~')
            return false
        } else {
            return true
        }

    }
},
```

PS：文档上，这块内容分了两个部分，一个是“基础组件”，另一个是“深入组件”都有讲解；

21.组件双向绑定

学习要点：

1.双向绑定

2.组件双向

本节课我们来开始学习 Vue3.x 的组件使用双向绑定的方法。

一. 双向绑定

1. 首先，回顾一下双向绑定的使用方法，使用 v-model 即可实现；

```
<input type="text" v-model="message">
```

PS：双向绑定即可实现内容的响应式更新；

2. 当然，它的原理就是通过 input 事件来进行内容的替换，如果改成完全形式如下：

```
<input type="text" v-bind:value="message"
                   v-on:input="message = $event.target.value">
```

PS：通过 input 事件来获取输入的值，再赋值给 message，而<input>再通过 v-bind 更新值；

二. 组件双向

1. 首先，在组件调用的地方改写成如下格式：

```
<html-a v-model:parent-message="message"></html-a>
```

PS：将 v-bind:改成 v-model 即双向绑定，而模板内部需要绑定对应的父组件属性；

2. 然后，根据第一个要点的写法，我们编写相应的 input 代码；

```
// 模板
template : `
    <div><input type="text" v-bind:value="parentMessage"
                        v-on:input="$emit('update:parentMessage',
                                            $event.target.value)">
    </div>
    <div>{{parentMessage}}</div>
    <div>{{message}}</div>
`
```

PS：这里通过触发 input 事件来执行自定义事件，并抛出数据；

3. 也可以通过计算属性中的 get()和 set()来改写组件中的双向绑定；

```
template : `
    <div><input type="text" v-model="value"></div>
`
```

```
// 计算属性
computed : {
    value : {
        get() {
            return this.parentMessage
        },
        set(value) {
            this.$emit('update:parentMessage', value)
        }
    }
}
```

22.组件插槽使用

学习要点：

1.插槽使用

本节课我们来开始学习 Vue3.x 的组件插槽的使用方法。

一. 插槽使用

1. 组件中有一些高级的小功能，比如：插槽<slot>，可以实现内容的动态分发；
2. 有时，我们需要在组件模板中定义大量重复的内容区域，可以用插槽来避免重复；
3. 首先，我们先看下如下代码中的场景：

```
<div id="app">
    <html-a></html-a>
    <html-a></html-a>
    <html-a></html-a>
    <html-a></html-a>
</div>

// 模板
template : `
    <div>{{message}}</div>
    <div>****</div>
`
```

PS：这里有四个<html-a>，我们希望输出内容时，增加不同的符号，现有知识只能同一符号；

4. 如果可以在<html-a>内部输入不同的符号内容，通过某种机制(插槽)来实现排版：

```
<div id="app">
    <html-a>****</html-a>
    <html-a>@@@@</html-a>
    <html-a>####</html-a>
    <html-a>!!!!</html-a>
</div>
```

PS：这里被组件元素包含的内容，并不会真正被渲染，会直接被忽略掉；

5. 通过插值来分发组件内部包含的内容，具体如下：

```
// 模板
template : `
    <div>{{message}}</div>
    <div><slot></slot></div>
`
```

6. 如果组件元素中没有内容，也可以通过插槽<slot>提供一个默认值；

```
<html-a></html-a>
<div><slot>0000</slot></div>
```

23.具名插槽和作用域

学习要点：

1.具名插槽

2.作用域问题

本节课我们来开始学习 Vue3.x 的组件插槽的名称设置和作用域问题。

一. 具名插槽

1. 具名插槽也就是给插槽起一个名字，然后调用对应的内容，而其它则被忽略；

```
<div id="app">
    <html-a>****</html-a>
    <html-a v-slot:header>header</html-a>
</div>

// 模板
template : `
    <div>{{message}}</div>
    <div><slot name="header"></slot></div>
`
```

PS：具名插槽的好处不言而喻了，可以有效控制各种复杂的布局和内容展示；

PS：v-slot 也支持缩写方案，用#号代替即可；

```
<html-a #header>header</html-a>
```

二. 作用域问题

1. 用于作用域的问题，插槽内的值是无法获取到子组件 data 属性的内容的；

```
<html-a>{{message}}</html-a>
```

PS：这里的{{message}}到底是子组件的 message 还是父组件的 message？

2. 为了解决这个问题，插槽提供了 v-slot:default 方案来处理，具体如下：

```
<html-a v-slot:default="slotProps">
    {{slotProps.message}}
</html-a>


template : `
    <div>{{message}}</div>
    <div><slot v-bind:message="message"></slot></div>
`
```

PS：这里命名会有误导性，这里的 slotProps 是可以自定义的，下面左边的 message 自定义；
PS：如果改为 v-bind:abc，那么调用就是：slotProps.abc 即可；

3. 下面是作用域插槽的一些简写，具体如下：

```
<html-a v-slot="slotProps">
    {{slotProps.message}}
</html-a>
```

4. 使用 ES6+的解构语法，更加方便，具体如下：

```
<html-a v-slot="{message}">
    {{message}}
</html-a>

//或

<html-a v-slot="{message : info}">
    {{info}}
</html-a>
```

24.实例和生命周期

学习要点：

1.应用实例

2.生命周期

本节课我们来开始学习 Vue3.x 的应用实例和生命周期的问题。

一. 应用实例

1. 首先，回顾一下：我们创建一个应用实例，具体方案如下：

```
// 创建一个应用实例，{}里是选项对象
const app = Vue.createApp({})

// 这里的 app 是实例
console.log(app)

// app.mount()这个返回的是一个代理对象，并不返回应用本身
// 它返回的是这个组件本身，也就是根组件
const vm = app.mount('#app')
```

PS：.mount()方法作用是挂载 DOM 元素，是 Vue 的应用 API，并允许链式调用；

PS：在“API 参考”中找到“应用 API”目录，可以参考更多的应用 API；

PS：目前学习了两个，另一个是注册全局组件的.component()；

2. 由于，.mount()返回的是根组件实例对象，那么它可以直接调用 data()内的数据；

```
const app = Vue.createApp({
    data() {
        return {
            count : 100
        }
    }
})

const vm = app.mount('#app')
console.log(vm.count)
```

二. 生命周期

1. 首先，可以参考一下文档中生命周期的图示，看下它执行的流程；
2. 其次，我们要了解一下常用的声明周期的钩子，具体如下：

(1) .created 钩子：当实例被创建后会执行(在模板渲染前执行，一般用于初始化属性值)；

(2) .mounted 钩子：当实例被挂载后会执行(在模板渲染完成后执行，此时可以操作 DOM)；

(3) .updated 钩子：当虚拟 DOM 被修改后会执行；

```
<div id="app">
    <div id="abc">{{count}}</div>
</div>

// 创建一个应用实例，{}里是选项对象
const app = Vue.createApp({
    data() {
        return {
            count : 0
        }
    },

    // 初始化
    created() {
        this.count = 100
        console.log('初始化~')

        // 无法获取 DOM
        //console.log(document.getElementById('abc').innerText)
    },

    // 挂载后执行
    mounted() {
        // 获取 DOM
        console.log(document.getElementById('abc').innerText)
    },

    // 数据被修改后执行
    updated() {
        console.log('数据被修改了~')
    }

})
```

25.动画和转场效果

学习要点：

1.转场效果

本节课我们来开始学习 Vue3.x 的动画和转场效果的简单演示。

一. 转场效果

1. 动画效果并不是 Vue 的核心功能，这里简单的了解下即可，原因如下：
   (1) . 内容不难，但繁杂(主要 CSS 调用多)；
   (2) . 学了一个，其它的参考手册，对应着搭建即可完成效果；

2. 我们就选择一个讲解即可，“进场和离场”效果用的比较多一些；
3. 创建一个按钮、要切换进场和离场的内容，以及设置默认显示的属性；

```
<div id="app">
    <p v-if="show">显示/隐藏</p>
    <button>切换{{show}}</button>
</div>

data() {
    return {
        show : true
    }
}
```

4. 用一个表达式来实现简单的布尔值的切换功能，具体如下：

```
<button v-on:click="show = !show">切换{{show}}</button>
```

5. 构建所需进场和离场的 CSS，推荐复制官方的，然后根据自己需求修改；

```
<style>
    .fade-enter-active, .fade-leave-active {
        transition: opacity 0.5s ease;
    }

    .fade-enter-from, .fade-leave-to {
        opacity: 0;
    }
</style>
```

6. 最后一步，使用 Vue 的效果组件<transition>来实现过渡效果；

```
<transition name="fade">
    <p v-if="show">显示/隐藏</p>
</transition>
```

PS：这是手册上的第一个小案例，我们一句句分析一下；

(1) .show : true，用于判断 dom 节点 p 状态；

(2) .show = !show，注意这里不是!=，而是!show，表示 show=true|false；

(3) .<transition>是 Vue 过渡动画组件，包含在里面才能实现动画效果；

(4) .name=fade，定义 CSS 样式的前缀名称，都已 fade 为前缀；

PS：在进入过渡或离开的过程中，有 6 种 class 可供切换，没有 name 默认为 v：

(1) .v-enter：进入过渡的开始状态；

(2) .v-enter-active：进入过渡的生效状态；

(3) .v-enter-to：进入过渡的结束状态；

(4) .v-leave：离开过渡的开始状态；

(5) .v-leave-active：离开过渡的生效状态；

(6) .v-leave-to：离开过渡的结束状态；

26.ref 和$refs 的使用

学习要点：

1.ref 和$refs

本节课我们来开始学习 Vue3.x 中的 ref 和$refs 的使用。

一. ref 和$refs

1. 有时，我们要操作一下 html 中的 DOM 内容，比如<input>或其中的 value；
2. ref 是在元素标签中进行设置注册，而$refs 则通过注册的信息进行内容获取；

```
<!--使用 ref 进行注册-->
<input type="text" ref="user" value="Mr.Lee">

// 初始化
created() {
    // 无法获取
    console.log(this.$refs.user)
},

// 挂载后
mounted() {
    // 可以获取，this.$refs 获取到 dom 节点
    console.log(this.$refs.user)
}

const vm = app.mount('#app')
console.log(vm.$refs.user.value)
```

3. 在子组件上使用 ref 时，在外部父组件调用时，根据调用链对应即可；

```
<!--在子组件注册 ref-->
<html-a ref="child"></html-a>

// 模板
template : `
    <input type="text" ref="user" value="Mr.Wang">
`

// 挂载后，父组件的
mounted() {
    // 调用子组件的 ref
    console.log(this.$refs.child.$refs.user.value)
},
```

```
// 外部调用
const vm = app.mount('#app')
console.log(vm.$refs.child.$refs.user.value)
```

27.watch 监听入门

学习要点：

1.watch 监听器

本节课我们来开始学习 Vue3.x 中的 watch 监听器的使用。

一. watch 监听器

1. watch 是一个监听选项对象，用于对数据的变化实现监听的过程；
2. 首先，我们先使用简单的 v-model 来实现简单的监听过程；

```
<div id="app">
    <input type="text" v-model="user"><br>
    {{user}}
</div>
```

3. watch 监听选项对象，和 data、computed、methods 处在同一层级上；
4. 在内部设置的方法名和被监听的 v-model 的值名是一样的，并且有两个参数；

```
// 监听
watch : {
    // 方法名和双向绑定名称一致
    // 参数 1 为新值，参数 2 为上一次值
    user(newValue, oldValue) {
        console.log('new:' + newValue + ', old:' + oldValue)
    }
}
```

5. 当然，我们也可以将逻辑封装到方法里，具体如下：

```
// 方法
methods : {
    getValue(newValue, oldValue) {
        console.log('new:' + newValue + ', old:' + oldValue)
    }
},

// 监听
watch : {
    // 将实际逻辑封装到方法里
    user : 'getValue'
}
```

6. 如果要监听的类型是对象，那么需要配合计算属性进行监听，具体如下：

```
<div id="app">
    <input type="text" v-model="user.name"><br>
    {{user.name}}
</div>

data() {
    return {
        user : {
            name : 'Mr.Lee'
        }
    }
},

// 计算属性
computed : {
    getUserName() {
        return this.user.name
    }
},

// 方法
methods : {
    getValue(newValue, oldValue) {
        console.log('new:' + newValue + ', old:' + oldValue)
    }
},

// 监听
watch : {
    // 调用计算属性监听
    getUserName : 'getValue'
}
```

PS：手册查阅 watch()选项对象，在“计算属性和侦听器”以及 API 文档中的 Data 选项对象；

28.watch 监听进阶

学习要点：

1.watch 进阶

本节课我们来开始学习 Vue3.x 中的 watch 监听器进阶的使用。

一. watch 进阶

1. 上一节第三种方式，通过计算属性来实现 watch()监听，也可以用字符串形式；

```
// 监听对象
watch : {
    // 字符串形式
    'user.name' : 'getUser'
}
```

2. 如果说，对象有多个数据那么上面两种方法就存在了局限性，需要更深度的监听；
3. watch 提供了监听的三种属性，具体如下：

(1) .handle：要监听的方法名；

(2) .deep：是否开启深度监听；

(3) .immediate：是否以当前的初始值执行方法(立即执行)；

```
// 监听对象
watch : {
    user : {
        handler : 'getUser',          // 执行方法
        deep : true,                  // 开启深度监听
        immediate : true              // 立即执行
    }
}
```

PS：此时在 getUser 方法传递的参数只有一个，且只是 user 对象实例；

```
// 方法
methods : {
    // 深度监听，传递 User 对象
    getUser(obj) {
        // 此时可以处理 user 对象下的所有内容
        console.log(obj)
    }
},
```

4. 使用数组的方式，可以同时实现多个监听；

```
// 方法
methods : {
    getUser(obj) {
        console.log('监听 1~')
    },

    getUser2(obj) {
        console.log('监听 2~')
    }
},


// 监听对象
watch : {
    user : [
        {
            handler : 'getUser',
            deep : true,
        },
        {
            handler : 'getUser2',
            deep : true,
        }
    ]
}
```

29.$watch 的使用

学习要点：

1.$watch

本节课我们来开始学习 Vue3.x 中的$watch 实例方法的使用。

一. $watch

1. $watch 实例方法和 watch 选项对象的作用功能是一样的，比如在初始化执行；

```
<input type="text" v-model="count"><br>
{{count}}

data() {
    return {
        count : 0
    }
},

// 初始化
created() {
    this.$watch('count', (newValue, oldValue) => {
        console.log(newValue + ',' + oldValue)
    })
}

const vm = app.mount('#app')
vm.$watch('count', (newValue, oldValue) => {
    console.log(newValue + ',' + oldValue)
})
```

PS：这里本身会返回一个取消侦听的函数；

```
const unwatch = vm.$watch...

// 可以根据业务适时取消监听
unwatch()
```

2. 如果对于复杂的对象类型，需要使用箭头函数来表示需要监听的内容；

```
this.$watch(() => this.user.name, (newValue, oldValue) => {
     console.log(newValue + ',' + oldValue)
})
```

PS：这里使用字符串形式无效，需要使用箭头函数的语法来声明；

3. $watch 也支持深度监听的，支持关键字 deep 和 immediate 选项：

```
// 深度监听
this.$watch('user', (obj) => {
    console.log(obj.age)
}, {
    deep : true,
    immediate : true
})
```

30.混入可复用

学习要点：

1.复用混入

本节课我们来开始学习 Vue3.x 中的用于组件复用的方法。

一. 复用混入

1. 混入(mixin)：用来分发 Vue 组件中可复用的部分，先简单的看下语法：

```
// 创建一个要复用的混入 Mixin
const baseMixin = {
    data() {
        return {
            count : 0
        }
    },

    created() {
        this.count = 100
    }
}

// 创建一个应用实例，
const app = Vue.createApp({
    // 载入复用的混入
    mixins : [baseMixin],
})
```

2. 选项合并：如果在组件体内设置自己的选项内容时，会进行恰当的合并；

(1) .重名时，会替换掉混入的部分，没有重名，则合并；

(2) .如果复用的部分有声明周期钩子，比如 created()，将会执行这个；

(3) .如果复用的部分都有声明周期钩子，那么两个钩子都会被执行；

(4) .如果复用的部分有 methods 等方法，重名将覆盖，不重名则合并；

```
// 创建一个应用实例，
const app = Vue.createApp({
    // 载入复用的混入
    mixins : [baseMixin],
    data() {
        return {
            message : 'Vue3.x~',
            count : 200
        }
```

```
    }
})
```

3. 声明周期钩子，复用和组件内均有时，都会被执行；

```
const baseMixin = {
    created() {
        this.count = 100
        console.log('我是复用的钩子~')
    }
}

const app = Vue.createApp({
    // 载入复用的混入
    mixins : [baseMixin],

    created() {
        console.log('我是组件内的钩子~')
    },
}
```

PS：methods 内的方法，重名则覆盖，不重名则合并；其它请参阅手册并自行测试；

PS：全局混入使用 app.mixin()，和全局组件注册类似，不再赘述；

PS：Vue3.x 基础部分在这节课完结，还有进阶(高级)、工具(脚手架、路由、状态)、新特性将在后面的阶段二(工具篇)、阶段三(新特性篇)补全知识点；