Vue基础 – 模板语法

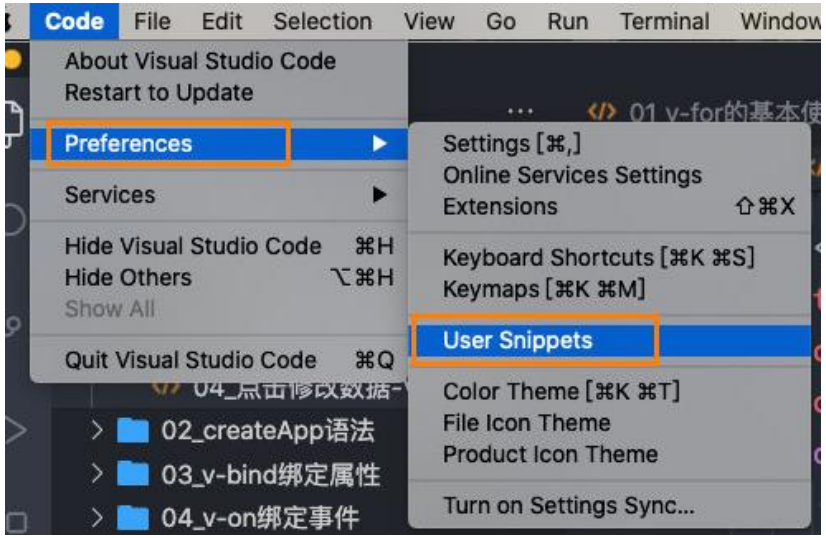
王红元 coderwhy

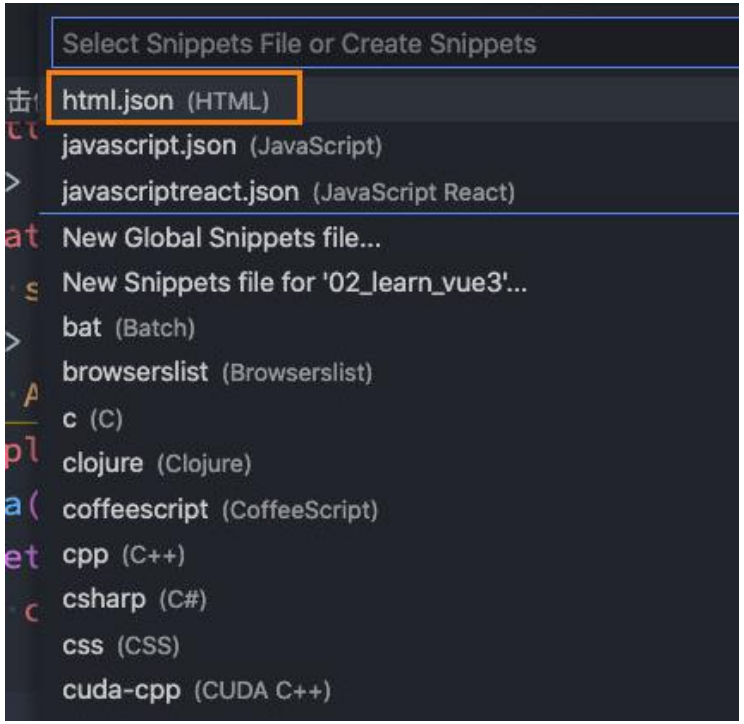
目录
content

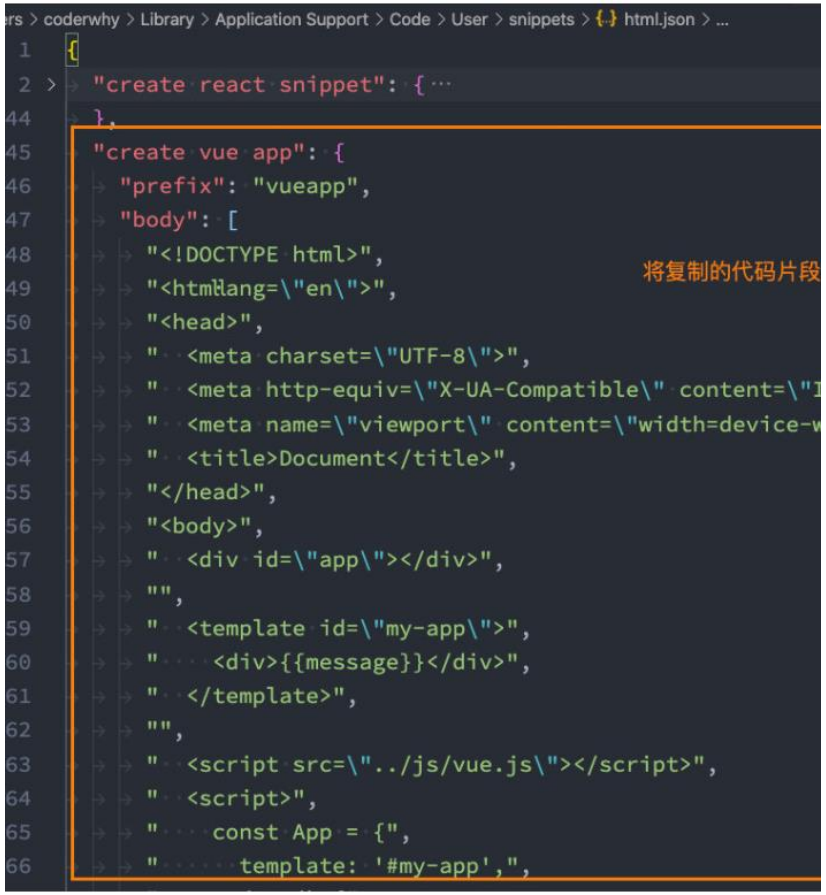
VSCode代码片段

- **我们在前面练习Vue的过程中，有些代码片段是需要经常写的，我们再VSCode中我们可以生成一个代码片段，方便我们快速生成。**
- **VSCode中的代码片段有固定的格式，所以我们一般会借助于一个在线工具来完成。**
- **具体的步骤如下：**
  - 第一步，复制自己需要生成代码片段的代码；
  - 第二步，https://snippet-generator.app/在该网站中生成代码片段；
  - 第三步，在VSCode中配置代码片段；

代码片段过程


Code
File
Edit
Selection
View
Go
Run
Terminal
Window
About Visual Studio Code
Restart to Update
Preferences
Services
Hide Visual Studio Code ⌘H
Hide Others ⌥⌘H
Show All
Quit Visual Studio Code ⌘Q
Settings [⌘,]
Online Services Settings
Extensions ⇧⌘X
Keyboard Shortcuts [⌘K ⌘S]
Keymaps [⌘K ⌘M]
User Snippets
Color Theme [⌘K ⌘T]
File Icon Theme
Product Icon Theme
Turn on Settings Sync...
02_createApp语法
03_v-bind绑定属性
04_v-on绑定事件


Select Snippets File or Create Snippets
html.json (HTML)
javascript.json (JavaScript)
javascriptreact.json (JavaScript React)
New Global Snippets file...
New Snippets file for '02_learn_vue3'...
bat (Batch)
browserslist (Browserslist)
c (C)
clojure (Clojure)
coffeescript (CoffeeScript)
cpp (C++)
csharp (C#)
css (CSS)
cuda-cpp (CUDA C++)


rs > coderwhy > Library > Application Support > Code > User > snippets > {} html.json > ...
"create react snippet": {
},
"create vue app": {
"prefix": "vueapp",
"body": [
"<!DOCTYPE html>",
"<html lang=\"en\">",
"<head>",
"<meta charset=\"UTF-8\">",
"<meta http-equiv=\"X-UA-Compatible\" content=\"I
"<meta name=\"viewport\" content=\"width=device-w
"<title>Document</title>",
"</head>",
"<body>",
"<div id=\"app\"> ",
"",
"<template id=\"my-app\">",
" {{message}} ",
"</template>",
"",
"<script src=\"../js/vue.js\"></script>",
"<script>",
"const App = {",
"template: '#my-app',",
将复制的代码片段

模板语法

- **React的开发模式：**
  - React使用的jsx，所以对应的代码都是编写的类似于js的一种语法；
  - 之后通过Babel将jsx编译成 React.createElement 函数调用；
- **Vue也支持jsx的开发模式（后续有时间也会讲到）：**
  - 但是大多数情况下，使用基于HTML的模板语法；
  - 在模板中，允许开发者以声明式的方式将DOM和底层组件实例的数据绑定在一起；
  - 在底层的实现中，Vue将模板编译成虚拟DOM渲染函数，这个我会在后续给大家讲到；
- **所以，对于学习Vue来说，学习模板语法是非常重要的。**

Mustache双大括号语法（掌握）

- **如果我们希望把数据显示到模板（template）中，使用最多的语法是 “Mustache”语法 (双大括号) 的文本插值。**
  - 并且我们前端提到过，data返回的对象是有添加到Vue的响应式系统中；
  - 当data中的数据发生改变时，对应的内容也会发生更新。
  - 当然，Mustache中不仅仅可以是data中的属性，也可以是一个JavaScript的表达式。
- **另外这种用法是错误的：**

```
<template id="my-app">
  <div>
    <!-- mustache基本使用 -->
    <h2>{{message}}</h2>
    <!-- JavaScript表达式 -->
    <h2>{{ counter * 2}}</h2>
    <h2>{{message.split(" ").reverse().join(" ")}}</h2>
    <!-- 调用一个methods中的函数 -->
    <h2>{{reverse(message)}}</h2>
  </div>
</template>
```

```
<!-- 错误的写法 -->
<!-- 这是一个赋值语句，不是表达式 -->
<h2>{{var name = "Hello"}}</h2>
<!-- 控制流的if语句也是不支持的，可以使用三元运算符 -->
<h2>{{ if (true) { return message } }}</h2>
```

```
<!-- 三元运算符 -->
<h2>{{ true ? message: counter }}</h2>
```

v-once指令（了解）

- **v-once用于指定元素或者组件只渲染一次：**
  - 当数据发生变化时，元素或者组件以及其所有的子元素将视为静态内容并且跳过；
  - 该指令可以用于性能优化；

```
<h2 v-once>当前计数: {{counter}}</h2>
<button @click="increment">+1</button>
```

- **如果是子节点，也是只会渲染一次：**

```
<div v-once>
  <h2>当前计数: {{counter}}</h2>
  <button @click="increment">+1</button>
</div>
```

v-text指令（了解）

- 用于更新元素的 textContent：

```
<span v-text="msg"></span>
<!-- 等价于 -->
<span>{{msg}}</span>
```

v-html

- **默认情况下，如果我们展示的内容本身是 html 的，那么vue并不会对其进行特殊的解析。**
  - 如果我们希望这个内容被Vue可以解析出来，那么可以使用 v-html 来展示；

```
<template id="my-app">
  <div v-html='info'></div>
</template>

<script src="../js/vue.js"></script>
<script>
  const App = {
    template: '#my-app',
    data() {
      return {
        info: `<span style='color: red; font-size: 30px'>哈哈哈</span>`
      }
    }
  }
```

v-pre

- v-pre用于**跳过元素和它的子元素的编译过程**，显示原始的Mustache标签：
  - 跳过不需要编译的节点，加快编译的速度；

```
<template id="my-app">
  <div v-pre>{{message}}</div>
</template>
```

{{message}}

v-cloak

- **这个指令保持在元素上直到关联组件实例结束编译。**
  - 和 CSS 规则如 [v-cloak] { display: none } 一起用时，这个指令可以隐藏未编译的 Mustache 标签直到组件实例准备完毕。

```
    <style>
      [v-cloak] {
        display: none;
      }
    </style>
</head>
<body>
    <div id="app"></div>

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

- <c

v-bind的绑定属性

- **前端讲的一系列指令，主要是将值插入到模板内容中。**
- **但是，除了内容需要动态来决定外，某些属性我们也希望动态来绑定。**
  - 比如动态绑定a元素的href属性；
  - 比如动态绑定img元素的src属性；

- **绑定属性我们使用v-bind：**
  - **缩写**：：
  - **预期**：any (with argument) | Object (without argument)
  - **参数**：attrOrProp (optional)
  - **修饰符**：
    - ✓ .camel - 将 kebab-case attribute 名转换为 camelCase。
  - **用法**：动态地绑定一个或多个 attribute，或一个组件 prop 到表达式。

绑定基本属性

- v-bind用于**绑定一个或多个属性值**，或者**向另一个组件传递props值**（这个学到组件时再介绍）；
- **在开发中，有哪些属性需要动态进行绑定呢?**
  - 还是有很多的，比如图片的链接src、网站的链接href、动态绑定一些类、样式等等

```
<template id="my-app">
  <!-- 完整的写法 -->
  <img v-bind:src="src" alt="">
  <!-- 语法糖写法 -->
  <img :src="src" alt="">
  <!-- 注意和上面的区别 -->
  <img src="src" alt="">

  <!-- 绑定a元素 -->
  <a :href="href"></a>
</template>
```

- v-bind有一个对应的**语法糖**，也就是简写方式。
- 在开发中，我们通常会使用语法糖的形式，因为这样更加简洁。

绑定class介绍

- **在开发中，有时候我们的元素class也是动态的，比如：**
  - 当数据为某个状态时，字体显示红色。
  - 当数据另一个状态时，字体显示黑色。
- **绑定class有两种方式：**
  - 对象语法
  - 数组语法

绑定class – 对象语法

- **对象语法：**我们可以传给 :class (v-bind:class 的简写) 一个对象，以动态地切换 class。

```
<template id="my-app">
  <!-- 1.普通的绑定方式 -->
  <div :class="className">{{message}}</div>
  <!-- 2.对象绑定 -->
  <!-- 动态切换class是否加入: {类(变量): boolean(true/false)} -->
  <div class="why" :class="{nba: true, 'james': true}"></div>
  <!-- 3.案例练习 -->
  <div :class="{'active': isActive}">哈哈哈</div>
  <button @click="toggle">切换</button>
  <!-- 4.绑定对象 -->
  <div :class="classObj">哈哈哈</div>
  <!-- 5.从methods中获取 -->
  <div :class="getClassObj()">呵呵呵</div>
</template>
```

绑定class – 数组语法

- **数组语法：**我们可以把一个数组传给 :class，以应用一个 class 列表；

```
<template id="my-app">
    <!-- 1.直接传入一个数组 -->
    <div :class="['why', nba]">哈哈哈</div>
    <!-- 2.数组中也可以使用三元运算符或者绑定变量 -->
    <div :class="['why', nba, isActive? 'active': '']">呵呵呵</div>
    <!-- 3.数组中也可以使用对象语法 -->
    <div :class="['why', nba, {'actvie': isActive}]">嘻嘻嘻</div>
</template>
```

绑定style介绍

- **我们可以利用v-bind:style来绑定一些CSS内联样式：**
  - 这次因为某些样式我们需要根据数据动态来决定；
  - 比如某段文字的颜色，大小等等；
- **CSS property 名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用引号括起来) 来命名；**
- **绑定class有两种方式：**
  - 对象语法
  - 数组语法

绑定style演练

■ **对象语法:**

```
<template id="my-app">
  <!-- 1.基本使用: 传入一个对象, 并且对象内容都是确定的 -->
  <div :style="{color: 'red', fontSize: '30px', 'background-color': 'blue'}">{{message}}</div>
  <!-- 2.变量数据: 传入一个对象, 值会来自于data -->
  <div :style="{color: 'red', fontSize: size+'px', 'background-color': 'blue'}">{{message}}</div>
  <!-- 3.对象数据: 直接在data中定义好对象在这里使用 -->
  <div :style="styleObj">{{message}}</div>
</template>
```

■ **数组语法:**

□ :style 的数组语法可以将多个样式对象应用到同一个元素上;

```
<template id="my-app">
  <div :style="[styleObj1, styleObj2]">{{message}}</div>
</template>
```

动态绑定属性

- **在某些情况下，我们属性的名称可能也不是固定的：**
  - 前端我们无论绑定src、href、class、style，属性名称都是固定的；
  - 如果属性名称不是固定的，我们可以使用 :[属性名]= “值” 的格式来定义；
  - 这种绑定的方式，我们称之为动态绑定属性；

```
<template id="my-app">
  <!-- 属性的名称是动态的 -->
  <div :[name]="value">{{message}}</div>
</template>
```

绑定一个对象

- **如果我们希望将一个对象的所有属性，绑定到元素上的所有属性，应该怎么做呢?**
  - 非常简单，我们可以直接使用 v-bind 绑定一个 对象;
- **案例：info对象会被拆解成div的各个属性**

```
<template id="my-app">
  <div v-bind="info">{{message}}</div>
</template>
```

v-on绑定事件

- **前面我们绑定了元素的内容和属性，在前端开发中另外一个非常重要的特性就是交互。**
- **在前端开发中，我们需要经常和用户进行各种各样的交互：**
  - 这个时候，我们就必须监听用户发生的事件，比如点击、拖拽、键盘事件等等
  - 在Vue中如何监听事件呢？使用v-on指令。
- **接下来我们来看一下v-on的用法：**

v-on的用法

- **v-on的使用：**
  - **缩写**：@
  - **预期**：Function | Inline Statement | Object
  - **参数**：event
  - **修饰符**：
    - ✓ .stop - 调用 event.stopPropagation()。
    - ✓ .prevent - 调用 event.preventDefault()。
    - ✓ .capture - 添加事件侦听器时使用 capture 模式。
    - ✓ .self - 只当事件是从侦听器绑定的元素本身触发时才触发回调。
    - ✓ .{keyAlias} - 仅当事件是从特定键触发时才触发回调。
    - ✓ .once - 只触发一次回调。
    - ✓ .left - 只当点击鼠标左键时触发。
    - ✓ .right - 只当点击鼠标右键时触发。
    - ✓ .middle - 只当点击鼠标中键时触发。
    - ✓ .passive - { passive: true } 模式添加侦听器
  - **用法**：绑定事件监听

v-on的基本使用

- 我们可以使用v-on来监听一下点击的事件：

```
<!-- 1.基本使用 -->
<!-- 绑定一个表达式 -->
<button v-on:click="counter++"></button>
<!-- 绑定到一个methods方法中 -->
<button v-on:click="btnClick">按钮1</button>
```

- v-on:click可以写成@click，是它的语法糖写法：

```
<!-- v-on的语法糖 -->
<button @click="btnClick">按钮2</button>
```

- 当然，我们也可以绑定其他的事件：

```
<!-- 绑定鼠标移动事件 -->
<div @mousemove="mouseMove">div的区域</div>
```

- 如果我们希望一个元素绑定多个事件，这个时候可以传入一个对象：

```
<!-- 2.绑定对象 -->
<button v-on="{click: btnClick, mousemove: mouseMove}">特殊按钮3</button>
```

v-on参数传递

- **当通过methods中定义方法，以供@click调用时，需要注意参数问题：**
- 情况一：如果该方法不需要额外参数，那么方法后的()可以不添加。
  - 但是注意：如果方法本身中有一个参数，那么会默认将原生事件event参数传递进去
- 情况二：如果需要同时传入某个参数，同时需要event时，可以通过$event传入事件。

```
<!-- 3.内联语句 -->
<!-- 默认会把event对象传入 -->
<button @click="btn4Click">按钮4</button>
<!-- 内联语句传入其他属性 -->
<button @click="btn5Click($event, 'why')">按钮5</button>
```

```
btn4Click(event) {
  console.log(event);
},
btn5Click(event, message) {
  console.log(event, message);
},
```

v-on的修饰符

- **v-on支持修饰符，修饰符相当于对事件进行了一些特殊的处理：**
  - .stop - 调用 event.stopPropagation()。
  - .prevent - 调用 event.preventDefault()。
  - .capture - 添加事件侦听器时使用 capture 模式。
  - .self - 只当事件是从侦听器绑定的元素本身触发时才触发回调。
  - .{keyAlias} - 仅当事件是从特定键触发时才触发回调。
  - .once - 只触发一次回调。
  - .left - 只当点击鼠标左键时触发。
  - .right - 只当点击鼠标右键时触发。
  - .middle - 只当点击鼠标中键时触发。
  - .passive - { passive: true } 模式添加侦听器

```
<!-- 4.修饰符 -->
<div @click="divClick">
  <button @click.stop="btnClick">按钮6</button>
</div>
<input type="text" @keyup.enter="onEnter">
```

条件渲染

- 在某些情况下，我们需要根据当前的条件决定某些元素或组件是否渲染，这个时候我们就需要进行条件判断了。
- Vue提供了下面的指令来进行条件判断：
  - v-if
  - v-else
  - v-else-if
  - v-show
- 下面我们来对它们进行学习。

v-if、v-else、v-else-if

- **v-if、v-else、v-else-if用于根据条件来渲染某一块的内容：**
  - 这些内容只有在条件为true时，才会被渲染出来；
  - 这三个指令与JavaScript的条件语句if、else、else if类似；

```
<template id="my-app">
  <input type="text" v-model.number="score">
  <h2 v-if="score > 90">优秀</h2>
  <h2 v-else-if="score > 80">良好</h2>
  <h2 v-else-if="score > 60">普通</h2>
  <h2 v-else>不及格</h2>
</template>
```

- **v-if的渲染原理：**
  - v-if是惰性的；
  - 当条件为false时，其判断的内容完全不会被渲染或者会被销毁掉；
  - 当条件为true时，才会真正渲染条件块中的内容；

template元素

- **因为v-if是一个指令，所以必须将其添加到一个元素上：**
  - 但是如果我们希望切换的是多个元素呢?
  - 此时我们渲染div，但是我们并不希望div这种元素被渲染；
  - 这个时候，我们可以选择使用template；
- **template元素可以当做不可见的包裹元素，并且在v-if上使用，但是最终template不会被渲染出来：**
  - 有点类似于小程序中的block

```
<template id="my-app">
  <template v-if="showHa">
    <h2>哈哈哈哈</h2>
    <h2>哈哈哈哈</h2>
    <h2>哈哈哈哈</h2>
  </template>
  <template v-else>
    <h2>呵呵呵呵</h2>
    <h2>呵呵呵呵</h2>
    <h2>呵呵呵呵</h2>
  </template>
  <button @click="toggle">切换</button>
</template>
```

v-show

- v-show和v-if的用法看起来是一致的，也是根据一个条件决定是否显示元素或者组件：

```
<template id="my-app">
  <h2 v-show="isShow">哈哈哈哈</h2>
</template>
```

v-show和v-if的区别

- **首先，在用法上的区别：**
  - v-show是不支持template；
  - v-show不可以和v-else一起使用；
- **其次，本质的区别：**
  - v-show元素无论是否需要显示到浏览器上，它的DOM实际都是有存在的，只是通过CSS的display属性来进行切换；
  - v-if当条件为false时，其对应的原生压根不会被渲染到DOM中；
- **开发中如何进行选择呢?**
  - 如果我们的原生需要在显示和隐藏之间频繁的切换，那么使用v-show；
  - 如果不会频繁的发生切换，那么使用v-if；