# Vue 3.x学习笔记

## 1、安装

前安装过vue的2.0版本，你需要把2.0相关的删除

npm uni -g vue-cli

### 1.1 通过Vue-cli脚手架安装

yarn global add @vue/cli  
npm install -g @vue/cli  
cnpm install -g @vue/cli

安装后通过vue -V检查版本，应该是4.5.\*。使用Vue-cli创建项目

vue create prj-vue1  
yarn serve # 运行项目

### 1.2 通过Vite脚手架创建项目

npm init vite-app prj-vue2 # 使用npm 安装  
cd prj-vue2  
npm install  
npm run dev  
# 或用yarn  
yarn create vite-app prj-vue2  
cd prj-vue2  
yarn  
yarn dev

1.3 使用scss

使用scss的话，需要安装sass-loader和node-sass

## 2、目录结构

src：源目录

assets：静态文件目录

components：自定义组件目录

App.vue：根组件，被main.js引用

## 3、绑定数据

### 3.1 绑定普通数据

### 3.2 v-bind 绑定属性，动态参数和属性

### 3.3 v-html 绑定html

### 3.4 v-for 循环绑定，遍历对象的各个属性

v-for="value, index in array"  
v-for="value, attr, index in object"

## 4、自定义方法

在methonds中定义自定义方法函数

methods:{  
 setMsg(){  
   
 },...  
}

点击事件触发方法

<button v-on:click="setMsg()"></button> //简写为@click  
<button @click="setMsg()"></button>

## 5、绑定数据到style、class

:class="{}" // 可以绑定到变量、数组甚至对象,常用的是结合三目运算符  
:style="{}"

## 6、事件

事件参数、事件对象

1次事件可以执行多个方法

事件修饰符

.stop  
.prevent 阻止默认事件行为  
.capture  
.self  
.once  
.passiv

按键修饰符

.tab  
.enter  
.delete //同时捕捉“删除”和“退格”键  
.esc  
.space  
.up  
.down  
.left  
.right

## 7、双向数据绑定

获取表单数据

<input type="text" id="username" />  
<input type="text" ref="age" />  
可以使用原生js的语法：（原生dom操作非常耗费性能，尽量不用）  
var usernameObj = document.querySelector("#username")  
alert(usernameObj.value)  
或者使用vue ref  
alert(this.$refs.age.value)

双向数据绑定

data中定义数据变量，标签中使用 v-model="var"

## 8、vue模板中使用Javascript表达式

表达式、条件判断、计算属性和watch监听

v-if v-else v-else-if，如果需要一组标签需要控制的话，可以使用Template

v-show 通过css控制显示，内容会产生，但不显示

## 9、计算属性

computed:{  
 // 计算属性使用的变量如果修改了，则属性值会跟着改变  
}

## 10、监听数据变化

watch:{  
 // 监听变量的变化，执行某些代码  
}

11、Vue3中集成Sass/scsss

11.1 安装sass-loader node-sass

npm install -D sass-loader node-sass

11.2 style中配置sass/scss

<style lang="scss" scoped> <!--scoped 表示样式作用域为当前组件-->  
 scss语法  
</style>

## 11、Vue中的组件

组件通常放到components中，命名为 xx.vue，组件通常由template、script和style组成。

引用组件的话，需要引入和挂载。

import Home from './components/Home'  
  
components: {  
 "v-home": Home //v-home为给组件标签起的别名，如不冲突，也可直接使用组件名称  
}

组件可以嵌套引用。

## 12、组件间传值

### 12.1 父组件给子组件传值

通过给子组件设置属性，子组件内部设置props来实现。

<my-header :title="值或者变量"></my-header> <!--父组件中引用子组件-->  
  
<!--子组件中定义-->  
<h2>{{title}}</h2>  
<script>  
export default{  
 data() {  
 return {  
   
 }  
 },  
 props: ["title"]  
}  
</script>

甚至可以把父组件本身（this）作为变量传递给子组件。

父组件向子组件传递数据是一个**单向数据流**，子组件不能修改父组件的值。

### 12.2 父组件主动获取子组件数据，执行子组件方法

1. 调用子组件时定义一个ref

* <v-header ref="header"></v-header>

1. 父组件获取子组件数据

* this.$refs.header.属性

1. 父组件执行子组件方法

* this.$refs.header.方法

### 12.3 子组件获取父组件数据，执行父组件方法

this.$parent.属性  
this.$parent.方法()

这种方式的数据是**双向**的，可以相互修改数据。

## 13、自定义组件事件实现子组件给父组件传值

<!--父组件中自定义事件-->  
@事件名="方法"  
<!--子组件中调用自定义事件-->  
<script>  
export default{  
 emits:["事件名称"],  
 data() {  
 return {   
 }  
 },  
 methods:{  
 run(){  
 this.$emit("事件名", "传递的值，可选")  
 }  
 }  
 props: ["title"]  
}  
</script>

## 14、非父子组件传值

没有引用关系的组件是非父子组件。

通过调用第三方模块mitt来实现。

yarn install mitt

## 15、自定义组件的双向数据绑定

### 15.1 使用v-model实现双向数据绑定

### 15.2 自定义组件中使用slots

## 16、Vue3.x中，非Prop的Attribute继承

一个非prop的attribute是指传向一个组件，但该组件并没有相应的props或emits定义的attribute。常见的包括class、style和id属性。

可以在组件中设置禁用属性的继承，然后自定义继承。

<template>  
<div>  
 <input v-bind="$attrs" /> <!--在这里设置继承属性的标签-->  
</div>  
</template>  
  
<script>  
export default{  
 inheritAttrs: false, //禁用默认集成  
 data() {  
 ...  
 },  
}  
</script>

当有多个根节点时，也需要指定继承属性的节点标签。

## 17、Vue的生命周期函数

beforeCreate 、created、beforeMounted、mounted（最常用）、beforeUnmount（销毁，常用）、unmounted

## 18、使用第三方模块

Axios、FetchJsonp

## 19、Mixin实现组件功能的复用

定义mixin，然后在组件中通过mixins引用mixin来实现复用

<script>  
import baseMixin from '...'  
export default{  
 mixins: [baseMixin],  
 inheritAttrs: false, //禁用默认集成  
 data() {  
 ...  
 },  
}  
</script>

### 19.1 mixin选项的合并

组件和mixin中都定义同一名称的属性时，当前组件的属性会覆盖mixin中的相同属性，方法同样。

### 19.2 全局配置使用Mixin

在入口文件中通过 app.mixin(mixin模块)来全局配置，然后在组件中就可以直接使用mixin中定义的属性和方法了。

## 20、Vue3中的Teleport

Vue3.x中的组件模板属于该组件，但有时我们想把模板中的内容移动到当前组件之外的DOM中，这时可以使用Teleport。

<teleport to="body"> <!--将在body中输出内容-->  
 ...  
</teleport>

to的目标可以时标签，也可以是id（#id）

### 20.1 使用Teleport实现模态对话框组件

## 21、Composition API介绍

解决大型项目中的代码组织与逻辑复用问题。

目前，我们使用的是“options” API 构建组件。 为了将逻辑添加到Vue组件中，我们填充（options）属性，如data、methods、computed等。 这种方法最大的缺点是，它本身不是一个工作的JavaScript代码。 您需要确切地知道模板中可以访问哪些属性以及this关键字的行为。在底层，Vue编译器需要将此属性转换为工作代码。正因为如此，我们无法从自动建议或类型检查中获益。

Composition API希望将通过当前组件属性、可用的机制公开为JavaScript函数来解决这个问题。 Vue核心团队将组件Composition API描述为“一套附加的、基于函数的api，允许灵活地组合组件逻辑”。 使用Composition API编写的代码更易读，并且场景不复杂，这使得阅读和学习变得更容易。

Composition API 提供了以下函数：

* setup
* ref
* reactive
* watchEffect
* watch
* computed
* toRefs
* 生命周期的hooks

### 21.1 setup方法

vue3.0将组件的逻辑都写在了函数内部，setup()会取代vue2.x的data()函数，返回一个对象，暴露给模板。而且只在初始化的时候调用一次，因为值可以被跟踪，所以我们通过vue3来改变编程习惯。

在setup方法中，通过ref、reactive定义相应式数据，使用数据需要return。

<script>  
import {ref, reactive, toRefs} from "vue"  
export default {  
 setup(){  
 var name = ref("名称")  
 var userinfo = reactive({  
 username:"xxx",  
 age:20  
 })  
 var getData = () => {  
 alert(userinfo.username)  
 alert(name.value) //获取ref的数据有些不同  
 userinfo.username = "新名字"  
 name.value = "新值"   
 }  
 return {name, userinfo, getData}  
 }  
}  
</script>

reactive 接口，它的作用是处理你的对象让它经过 Proxy 的加工变为一个响应式的对象。

### 21.2 toRefs 解构响应式对象的数据

把一个响应式对象转换成普通对象，该普通对象的每一个property都是一个ref。

<script>  
import {ref, reactive, toRefs} from "vue"  
return {  
 ...toRefs(userinfo)  
}  
</script>

### 21.3 computed 计算属性

<script>  
import {ref, reactive, toRefs} from "vue"  
setup() {  
 let fullName = computed(() => {  
 return userinfo.firstname + userinfo.lastname  
 })  
 return {  
 ...toRefs(userinfo),  
 fullName  
 }  
}  
</script>

### 21.4 readonly “深层”只读代理

传入一个对象（响应式或者普通）或ref，返回一个原始对象的只读代理。该对象内部的任何嵌套的属性也是只读的。

<script>  
import {ref, reactive, toRefs, readonly} from "vue"  
setup() {  
 let obj = {  
 username = "非响应式数据"  
 age: 20  
 }  
 var userinfo = reactive({  
 username:"相应式数据",  
 age:20  
 })  
 userinfo = readonly(userinfo) // userinfo变成了只读非响应式数据  
 let fullName = computed(() => {  
 return userinfo.firstname + userinfo.lastname  
 })  
 return {  
 ...toRefs(userinfo),  
 fullName  
 }  
}  
</script>

obj由于没有使用reactive定义，所以是个非响应式数据，也叫原始对象。

### 21.5 watchEffect 监听响应式对象数据

### 21.6 watch监听数据，功能更强大一些

* 监听到reactive对象返回的对象
* 可以指定监听对象
* 可以得到数据变化前后的值

<script>  
import {ref, reactive, toRefs, watch} from "vue"  
setup() {  
 var userinfo = reactive({  
 username:"相应式数据",  
 age:20  
 })  
 return {  
 ...toRefs(userinfo),  
 fullName  
 }  
}  
</script>

### 21.7 生命周期函数

生命周期中除了beforeCreate、Created函数。

### 21.8 props 接收父组件的传值

在 Vue2.0 中我们可以使用 props 属性值完成父子通信，在 3.x 中则需要定义 props 属性去指定接受值的类型，然后利用 setup 的第一个参数获取 props 使用。

<!--父组件，调用content组件-->  
<template>  
 <div id="app">  
 <img alt="Vue logo" src="./assets/logo.png">  
 <div>{{msg}}的年龄为{{age}}</div>  
 <div>{{double}}</div>  
 <Content :msg='msg' />  
 <button @click="add">+</button>  
 </div>  
</template>  
<script>  
export default {  
 name: 'App',  
 components:{  
 Content  
 },  
 setup() {  
 const state = reactive({  
 msg:'王大合', age:18,  
 double : computed(() =>{  
 return state.age \* 2  
 })  
 })  
 function add() {  
 state.age += 1  
 }  
 return {...toRefs(state),add}  
 }  
}  
</script>

<!--子组件定义-->  
<template><div>{{data}}</div></template>  
<script>  
import {ref} from 'vue'  
export default {  
 name:'content',  
 props:{  
 msg:String  
 },  
 setup(props) {  
 const data = ref(props.msg)  
 return {data}  
 }  
}  
</script>

setup 函数的第二个参数是一个上下文对象 context，这个上下文对象中包含了一些有用的属性，这些属性在 Vue2.0 中需要通过 this 才能访问到，在 vue3.0 中，访问他们变成以下形式：

setup(props, ctx) {  
 console.log(ctx) // 在 setup() 函数中无法访问到 this  
 console.log(this) // undefined  
}

具体能访问到以下有用的属性：

* slot
* attrs
* emit

## 22、 Provider Inject组件跨多层传值

<!--在父组件中定义-->  
<script>   
provide：{  
 title: "要传递的值"  
}//或者用方法返回对象也可以  
provide(){  
 return {  
 title: "要传递的值"  
 }  
} // 两种方法都可以  
</script>  
  
<!--在子组件中引用-->  
<script>  
export default{  
 inject: ['title'] // 这里title要加引号  
}  
</script>

传统方式的父组件变量修改后不会传递到子组件。

Composition Api方式下，变量的修改可以进行同步传递，而且是双向传递。

## 23、引入typescript

### 23.1 安装

输入vue add typescript命令安装Typescript。或者使用yarn add typescript -D。

1. 是否使用class-style 组件语法？选择n（n产生的代码和antd 给出的示例代码一致）
2. 是否使用Babel，选Y
3. 是否切换所有.js文件为.ts文件？选Y
4. 是否允许.js文件被编译？选Y
5. 是否检查所有的依赖？选n

### 23.2 初始化tsconfig.json

npx tsc --init

### 23.3 使用typescript

在script标签中声明使用typescript语言。

<script lang="ts">  
import { defineComponent } from "vue"  
  
import Home from './components/Home.vue' //这个地方需要给出扩展名  
export default defineComponent({  
 ...  
})

如果import App from App.vue报错：Cannot find module './App.vue'，这时因为ts还没识别vue文件，需要进行下面的配置。

在项目根目录添加shim.d.ts文件

declare module "\*.vue" {  
 import { Component } from "vue";  
 const component: Component;  
 export default component;  
}

### 23.4 原始数据类型

JavaScript 的类型分为两种：原始数据类型（[Primitive data types](https://developer.mozilla.org/en-US/docs/Glossary/Primitive)）和对象类型（Object types）。

原始数据类型包括：布尔值、数值、字符串、null、undefined 以及 [ES6 中的新类型 Symbol](http://es6.ruanyifeng.com/#docs/symbol)。

下面主要介绍**前五种**原始数据类型在 TypeScript 中的应用。

* **布尔值** 在 TypeScript 中，使用 boolean 定义布尔值类型
* let isDone: boolean = false
* 注意，使用构造函数 Boolean 创造的对象**不是**布尔值：
* let createdByNewBoolean: boolean = new Boolean(1)  
  // new Boolean() 返回的是一个 Boolean 对象  
  let createdByBoolean: boolean = Boolean(1)  
  // 直接调用 Boolean 也可以返回一个 boolean 类型
* 在 TypeScript 中，boolean 是 JavaScript 中的基本类型，而 Boolean 是 JavaScript 中的构造函数。
* **数值** 使用 number 定义数值类型
* let decLiteral: number = 6;  
  let hexLiteral: number = 0xf00d;  
  // ES6 中的二进制表示法  
  let binaryLiteral: number = 0b1010;  
  // ES6 中的八进制表示法  
  let octalLiteral: number = 0o744;  
  let notANumber: number = NaN;  
  let infinityNumber: number = Infinity;
* 编译结果：
* var decLiteral = 6;  
  var hexLiteral = 0xf00d;  
  // ES6 中的二进制表示法  
  var binaryLiteral = 10;  
  // ES6 中的八进制表示法  
  var octalLiteral = 484;  
  var notANumber = NaN;  
  var infinityNumber = Infinity;
* 其中 0b1010 和 0o744 是 [ES6 中的二进制和八进制表示法](http://es6.ruanyifeng.com/#docs/number#二进制和八进制表示法)，它们会被编译为十进制数字。
* **字符串** 使用 string 定义字符串类型
* let myName: string = 'Tom';  
  let myAge: number = 25;  
  // 模板字符串  
  let sentence: string = `Hello, my name is ${myName}.  
  I'll be ${myAge + 1} years old next month.`;
* 其中 用来定义 ES6 中的模板字符串，${expr}` 用来在模板字符串中嵌入表达式。
* **空值** JavaScript 没有空值（Void）的概念，在 TypeScript 中，可以用 void 表示没有任何返回值的函数
* function alertName(): void {  
   alert('My name is Tom');  
  }
* 声明一个 void 类型的变量没有什么用，因为你只能将它赋值为 undefined 和 null
* let unusable: void = undefined
* 与 void 的区别是，undefined 和 null 是所有类型的子类型。也就是说 undefined 类型的变量，可以赋值给 number 类型的变量，而 void 类型的变量不能赋值给 number 类型的变量
* let u: undefined = undefined;  
  let n: null = null  
  // 这样不会报错  
  let num: number = undefined;  
  let num: number = u;

## 24、 搭建第一个 Vite 项目

### 24.1 初始化项目

npm init @vitejs/app my-vue-app --template vue  
yarn create @vitejs/app my-vue-app --template vue  
# 然后  
cd my-vue-app  
yarn  
yarn dev

vite支持的预定义模板有：

* vanilla
* vue
* vue-ts
* react
* react-ts
* preact
* preact-ts
* lit-element
* lit-element-ts

### 24.2 添加sass支持

yarn add sass

### 24. 3 安装 router、vuex组件

yarn add vue-router@4.0  
yarn add vuex@4.0

## 99、待学习解决问题汇总

### 99.1 组件的受控模式和非受控模式

什么叫组件的受控模式和非受控模式？

如果你也想写一个组件库，记得每个输入类的组件都应该有受控和非受控模式。[1]

### 99.2 组件的API设计