vue3_composition_api

之前的时候给同学们说过，vue3有2种类型的语法

1. `optioins api` 选项式API

   在这一种语法里在，它将数据定义在data里面，将方法定义在 `methods` 里面，将计算属性定义在 `computed` 里面，监控属性要写在 `watch` 当中

   这一种写法的优点非常明显 ，对于初学者来说，很清晰简单洁，可以快速上手，但是缺点也很明显

```
<script type="module">
    import { createApp } from "./js/vue.esm-browser.js";
    const app = createApp({
        // 数据写在这里
        data() {
            return {
                msg: "hello world"
            }
        },
        //计算属性写在这里
        computed: {
            str() {
                return "哈哈哈····" + this.msg;
            }
        },
        //方法写在这里
        methods: {
            changeMsg() {
                this.msg = "大家好才是真的好";
            }
        },
        //监听写在这里
        watch: {
            msg(newValue, oldValue) {
                console.log(newValue, oldValue);
            }
        },
        // 生命周期写这里
        created() {

        }
    });
    app.mount("#app");
</script>
```

   在上面的代码里面，我们可以看到，变量与方法都在不同的选项里面，对于代码里比较少的页面这没有什么问题，但是对于代码量比较多的面面，这样很容易让开发者形成一种代码的**撕裂感**

2. `composition api` 组合式api

在这一种情况下面，vue已经不再强制性使用之前的那些选项了，如`data/methods/computed/watch/生命周期`，可以将代码任意组合，它多了一个新的函数叫`setup`

```
<body>
    <div id="app">
        <h2>msg的值为：{{msg}}</h2>
        <button type="button" @click="sayHello">按钮1</button>
    </div>
</body>
<script type="module">
    import { createApp } from "./js/vue.esm-browser.js";
    const app = createApp({
        setup(){
            //所有的变量，方法，监听，生命周期等都写在这里，不用再单独的写在每一个选项里面了
            const msg = "hello";

            const sayHello = ()=>{
                alert("你好啊");
            }


            //setup函数必须有一个返回值 ，并且这个返回值必须是一个对象
            //这个返回值的内容将会做为接管页面的区域所使用的内容存在
            return {
                msg,
                sayHello
            }
        }
    });
    app.mount("#app");
</script>
```

关于数据的定义

在之前的选项式API里面，我们使用了`data`这个选项来定义数据，但是在新的里，我们可以直接定义数据

```
<body>
    <div id="app">
        <h2>{{nickName}}</h2>
        <button type="button" @click="nickName='李四'">按钮</button>
    </div>
</body>
<script type="module">
    import {createApp} from "./js/vue.esm-browser.js";
    const app = createApp({
        setup(){
            let nickName = "张三"

            return {
                nickName
            }
```

```
        }
    });
    app.mount("#app");
</script>
```

在上面的代码里面，当我们去点击按钮的时候，我们发现页面上面展示的数据并没有发生变化，这是因为vue**在定义数据的时候，所有数据默认都不会与页面形成响应**

通过 `ref` 定义响应式数据

如果想实现响就在式数据的定义，可以通过 `vue` 内部的一些特殊的函数来完成，其中 `ref` 就可以完成这个操作

> `ref` 实现的是栈与堆同时响应，也称之为全盘响应数据

```
<body>
    <div id="app">
        <h2>nickName的值为：{{nickName}}</h2>
        <button type="button" @click="nickName='标哥哥'">改变nickName</button>
        <button type="button" @click="changeNickName">通过函数改变nickName</button>
    </div>
</body>
<script type="module">
    import { createApp, ref } from "./js/vue.esm-browser.js";
    const app = createApp({
        setup() {
            const nickName = ref("张珊");

            const changeNickName = () => {
                //在这里，怎么修改nickName的值
                nickName.value = "桃子";
            }

            return {
                nickName,
                changeNickName
            }
        }
    });
    app.mount("#app");
</script>
```

```
<body>
    <div id="app">
        <h2>nickName的值为:{{nickName}}</h2>
        <button type="button" @click="nickName='标哥哥'">改变nickName</button>
        <button type="button" @click="changeNickName">通过函数改变nickName</button>
    </div>
</body>
<script type="module">
    import { createApp, ref } from "./js/vue.esm-browser.js";
    const app = createApp({
        setup() {
            const nickName = ref("张珊");

            const changeNickName = () => {
                //在这里，怎么修改nickName的值
                nickName.value = "桃子";
            }

            return {
                nickName,
                changeNickName
            }
        }
    });
    app.mount("#app");
</script>
```

第二步：在页面使用ref的数据的时候，不用通过value

第一步：ref可以实现响应式数据的定义

第三步：在setup里面使用ref定义的变量，要通过value来访问

上面是通过 `ref` 定义了基本数据类型，现在我们通过 `ref` 来定义一个对象

```
<body>
    <div id="app">
        <h2 @click="userInfo.userName = '桃子'">昵称：{{userInfo.userName}}</h2>
        <h2 @click="userInfo.age = 20">年龄：{{userInfo.age}}</h2>
        <button type="button" @click="changeUserInfo">改变userInfo整体</button>
    </div>
</body>
<script type="module">
    import { createApp, ref } from "./js/vue.esm-browser.js";
    const app = createApp({
        setup() {
            const userInfo = ref({
                userName: "张珊",
                age: 18
            });


            const changeUserInfo = () => {
                // userInfo.value.userName = "哈哈哈";
                userInfo.value = {
                    userName:"小岳岳",
                    age:21
                }

            }

            return {
                userInfo,
                changeUserInfo
            }
        }
    });
    app.mount("#app");
</script>
```

```
<body>
    <div id="app">                                          第二步：页面上面直接使用这个变量或修改这个属性值 可以实现响应
        <h2 @click="userInfo.userName = '桃子'">昵称：{{userInfo.userName}}</h2>
        <h2 @click="userInfo.age = 20">年龄：{{userInfo.age}}</h2>
        <button type="button" @click="changeUserInfo">改变userInfo整体</button>
    </div>
</body>
<script type="module">
    import { createApp, ref } from "./js/vue.esm-browser.js";
    const app = createApp({
        setup() {
            const userInfo = ref({      第一步：先通过ref定义了对象
                userName: "张珊",
                age: 18
            });


            const changeUserInfo = () => {
                // userInfo.value.userName = "哈哈哈";
                userInfo.value = {
                    userName:"小岳岳",
                    age:21                          第三步：在setup里面使用这个变量要通过value来访问
                }                                   同时，如果直接修改userInfo整体，也是可以的

            }

            return {
                userInfo,
                changeUserInfo
            }
        }
    });
    app.mount("#app");
```

总结：

1. ref可以实现全局响应，无论是修改里面的属性还是修改这个变量整体，都可以实现响应
2. ref定义的数据如果在 `setup` 里面使用，要通过 `value` 来调用，如果在页面使用则不用 `value`

通过 `reactive` 实现堆响应数据

> `reactive` 只能定义对象的，不能定义基本数据类型
>
> `reactive` 只能响应堆，不能响应栈

```
<body>
    <div id="app">
        <h2 @click="userInfo.nickName='桃子'">昵称 ：  {{userInfo.nickName}}</h2>
        <h2 @click="userInfo.age = 20">年龄：{{userInfo.age}}</h2>
        <button type="button" @click="changeUserInfo">点击改变userInfo</button>
    </div>
</body>
<script type="module">
    import { createApp, ref, reactive } from "./js/vue.esm-browser.js";
    const app = createApp({
        setup() {

            let userInfo = reactive({
                nickName: "张三",
                age: 18
            })

            const changeUserInfo = ()=>{
                // console.log(userInfo.nickName);
                // console.log(userInfo.age);
                // userInfo.nickName = "桃子2号";

                //上面的操作没有问题，但是下面的操作有问题，因为下面没有实现数据响应，reactive只支持堆里面的响应数据
```

```
                    userInfo = {
                        nickName:"桃子3号",
                        age:20
                    }

                }


                return {
                    userInfo,
                    changeUserInfo
                }
            }
        });
        app.mount("#app");
    </script>
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ref数据</title>
</head>

<body>
    <div id="app">            第二步：页面上面使用对象及修改对象的属性都是可以的
        <h2 @click="userInfo.nickName='桃子'">昵称 : {{userInfo.nickName}}</h2>
        <h2 @click="userInfo.age = 20">年龄：{{userInfo.age}}</h2>
        <button type="button" @click="changeUserInfo">点击改变userInfo</button>
    </div>
</body>
<script type="module">
    import { createApp, ref, reactive } from "./js/vue.esm-browser.js";
    const app = createApp({
        setup() {

            let userInfo = reactive({    第一步：通过reactive定义了对象
                nickName: "张三",
                age: 18
            })

            const changeUserInfo = ()=>{    第三步：在setup内部修改userInfo的属性也是正常的
                // console.log(userInfo.nickName);  同时不需要value(它这一点与ref不一样)
                // console.log(userInfo.age);
                // userInfo.nickName = "桃子2号";

                //上面的操作没有问题，但是下面的操作有问题，因为下面没有实现数据响应，reactive只支持堆里面的响应数据
                userInfo = {
                    nickName:"桃子3号",
                    age:20      第四步：reactive实现的是堆响应，所以改堆里面是可以响应的，但是直接改栈是不可以
                }               实现响应数据的

            }
```

> 总结：为了更好的去理解和使用响应式数据，我们在setup里面定义的响应式数据最好都使用 `const` 来修饰

关于方法

在vue之前的 `options api` 里面，如果要定义方法，我们需要在 `methods` 里面去定义，但是在 `composition api` 里面，则直接定义成函数就可以了【最好定义成箭头函数】

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>关于方法</title>
```

```
</head>
<body>
    <div id="app">
        <!-- 第三步：使用这个函数  -->
        <button type="button" @click="sayHello">按钮</button>
    </div>
</body>
<script type="module">
    import {createApp} from "./js/vue.esm-browser.js"

    const app = createApp({
        setup(){
            //第一步：直接定义成箭头函数
            const sayHello = ()=>{
                alert("你好啊");
            }

            return {
                //第二步：将这个函数返回
                sayHello
            }
        }
    });


    app.mount("#app");
</script>
</html>
```

> 函数的用法与之前没有变化，只是定义方式发生了变化，该有的参数都有，该有的修饰符也都有

计算属性

在之前的 `options api` 里面，我们的计算属性定是在 `computed` 里面，现在写法如下

```

<body>
    <div id="app">
        <ul>
            <li v-for="item in foodList">
                {{item.foodName}}---{{item.price}}---
                <button type="button" @click="item.count--">-</button>
                {{item.count}}
                <button type="button" @click="item.count++">+</button>
            </li>
        </ul>
        <hr>
        <h2>总金额：{{totalMoney}}</h2>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {

```

```
            const foodList = reactive([
                { foodName: "白菜炒豆腐", price: 20, count: 1 },
                { foodName: "青椒炒皮蛋", price: 18, count: 2 },
                { foodName: "西红柿炒月饼", price: 8, count: 7 },
            ]);

            //这里就是计算属性
            const totalMoney = computed(()=>{
                let sum = 0;
                foodList.forEach(item=>{
                    sum+=item.price*item.count;
                });
                return sum;
            });

            return {
                foodList,
                totalMoney
            }
        }
    });


    app.mount("#app");
</script>
```

> 计算属性主要是通过 `computed` 的函数来实现的

侦听器

在以前的 `options api` 里面，我们要使用 `watch` 去监听一个数据的变化，现在的写法如下

第一种情况：基本数据类型的监听

```
<body>
    <div id="app">
        <h2 @click="userName='桃子小姐姐'">{{userName}}</h2>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed, ref,watch } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {
            const userName = ref("标哥哥");
            // 监听userName的变化
            watch(userName,(newValue,oldValue)=>{
                console.log(newValue,oldValue);
            });

            return {
                userName
            }
        }
    });
```

```
    app.mount("#app");
</script>
```

第二种情况况：上面是基本数据类型的监听，如果是复制数据类型的监听，我们要使用深度监听

```
<body>
    <div id="app">
        <h2 @click="userInfo.nickName='测试修改昵称'">{{userInfo.nickName}}</h2>
        <h2>{{userInfo.age}}</h2>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed, ref, watch } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {
            const userInfo = reactive({
                nickName:"张三",
                age:18
            });

            //vue3的composition api会自动的根据你监听的数据类型来决定是否要户用深度监听
            watch(userInfo,(newValue,oldValue)=>{
                console.log("我变化了");
            })

            return {
                userInfo
            }
        }
    });
    app.mount("#app");
</script>
```

第三种情况：在vue3的composition api里面，可以监听某一个对象下面的某一个属性的变化

```
<body>
    <div id="app">
        <h2 @click="userInfo.nickName='测试修改昵称'">{{userInfo.nickName}}</h2>
        <h2 @click="userInfo.age = 22">{{userInfo.age}}</h2>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed, ref, watch } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {
            const userInfo = reactive({
                nickName: "张三",
                age: 18
            });

            //如果只想监听某一个对象里面的某一个属性的变化

            watch(() => userInfo.age, (newValue, oldValue) => {
                console.log("我在发生变化");
```

```
            });


            return {
                userInfo
            }
        }
    });
    app.mount("#app");
</script>
```

> 在上面的代码里面，我们可以看到，当 `nickName` 发生变化的时候， `watch` 并没有执行，但是当 `age` 发生了变化以后， `watch` 就监听到了

第四种情况：同时监听多个值

函数是可以任意多次的任何时候的调用执行，所以可以写成下面的方式

```
<body>
    <div id="app">
        <h2 @click="userInfo.nickName='测试修改昵称'">{{userInfo.nickName}}</h2>
        <h2 @click="userInfo.age = 22">{{userInfo.age}}</h2>
        <h2 @click="hobby='睡觉'">{{hobby}}</h2>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed, ref, watch } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {
            const userInfo = reactive({
                nickName: "张三",
                age: 18
            });
            const hobby = ref("看书");

            //我想监听userInfo.age和hobby的变化，怎么办呢

            watch(()=>userInfo.age,(newValue,oldValue)=>{
                console.log("age在变化");
            });
            watch(hobby,(newValue,oldValue)=>{
                console.log("hobby在变化");
            });

            return {
                userInfo,
                hobby
            }
        }
    });
    app.mount("#app");
</script>
```

如果我们发现某些监听的业务逻辑是相同的，还可以写成下面的方式

```
<body>
    <div id="app">
        <h2 @click="userInfo.nickName='测试修改昵称'">{{userInfo.nickName}}</h2>
        <h2 @click="userInfo.age = 22">{{userInfo.age}}</h2>
        <h2 @click="hobby='睡觉'">{{hobby}}</h2>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed, ref, watch } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {
            const userInfo = reactive({
                nickName: "张三",
                age: 18
            });
            const hobby = ref("看书");

            //我想监听userInfo.age和hobby的变化，怎么办呢
            //这里是以数组的形式完成的
            watch([() => userInfo.age, hobby], (newValue, oldValue) => {
                console.log("你发生变化了");
            })

            return {
                userInfo,
                hobby
            }
        }
    });
    app.mount("#app");
</script>
```

dom选取操作

在之前的 `options api` 里面，我们如果要操作一个页面的DOM元素，要使用 `ref` ，在新的API下面也是一样的，但是操作方法变了

```
<body>
    <div id="app">
        <h2 ref="xxx">我爱北京天安门</h2>
        <button type="button" @click="abc">按钮</button>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed, ref, watch } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {
            const xxx = ref(null);
            const abc = () => {
                // this.$refs.xxx
                // console.log(this);   undefined]
                console.log(xxx.value);
```

```
            }
            return {
                abc,
                xxx
            }
        }
    });
    app.mount("#app");
</script>
```

代码分析：

1. composition api 不存在 this 关键字了，它不依赖于对象了，它是函数式编程
2. 如果想获取dom，应该通过 ref() 定义一个变量以后，返回到页面，然后页面再通过 ref 绑定dom元素

```
<body>
    <div id="app">   第三步：在dom上面绑定ref的属性
        <h2 ref="xxx">我爱北京天安门</h2>
        <button type="button" @click="abc">按钮</button>
    </div>
</body>
<script type="module">
    import { createApp, reactive, computed, ref, watch } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {

            const xxx = ref(null);   第一步：先定义一个ref变量

            const abc = () => {
                // this.$refs.xxx
                // console.log(this);   undefined]
                console.log(xxx.value);  第四步：在这里就可以获取到页面上面的dom元素了
            }
            return {
                abc,
                xxx       第二步：返回这个ref变量
            }
        }
    });
    app.mount("#app");
</script>

</html>
```

生命周期

之前在 options api 里面，我们可以得到了是"四个过程，八个状态",但是在 composition API里面，现在的 setup 就相当于创建阶段，其它的则没有发生改变

同时，每一个生命周期的钩子函数都变成了回调函数，在前面添加了 on

```
<script type="module">
    import { createApp, onBeforeMount,onMounted,onBeforeUpdate,onUpdated } from "./js/vue.esm-browser.js"
    const app = createApp({
        setup() {
            // 四个过程，八个状态

            console.log("我是创建阶段");


            onBeforeMount(()=>{
                console.log("我是挂载之前");
```

```
            });
            onMounted(()=>{
                console.log("页面已经被挂载了")
            })

            return {

            }
        }
    });
    app.mount("#app");
</script>

```

名称	opation api	composition api
创建之前	beforeCreate	setup
创建之后	created	setup
挂载之前	beforeMount	onBeforeMount
挂载之后	mounted	onMounted
更新之前	beforeUpdate	onBeforeUpdate
更新之后	updated	onUpdated
卸载之前【销毁】	beforeUnmount	onBeforeUnmount
卸载之后【销毁后】	unmounted	onUnmounted

vite当中去使用 composition api

官方网站 https://cn.vitejs.dev/

vue真正的部分还是在脚手架上面，之前我们使用的打包工具是 webpack ，使用的脚手架 @vue/cli

现在我们可以换一个方式了直接使用 vite

这个东西就可以看成是另一种的打包工具

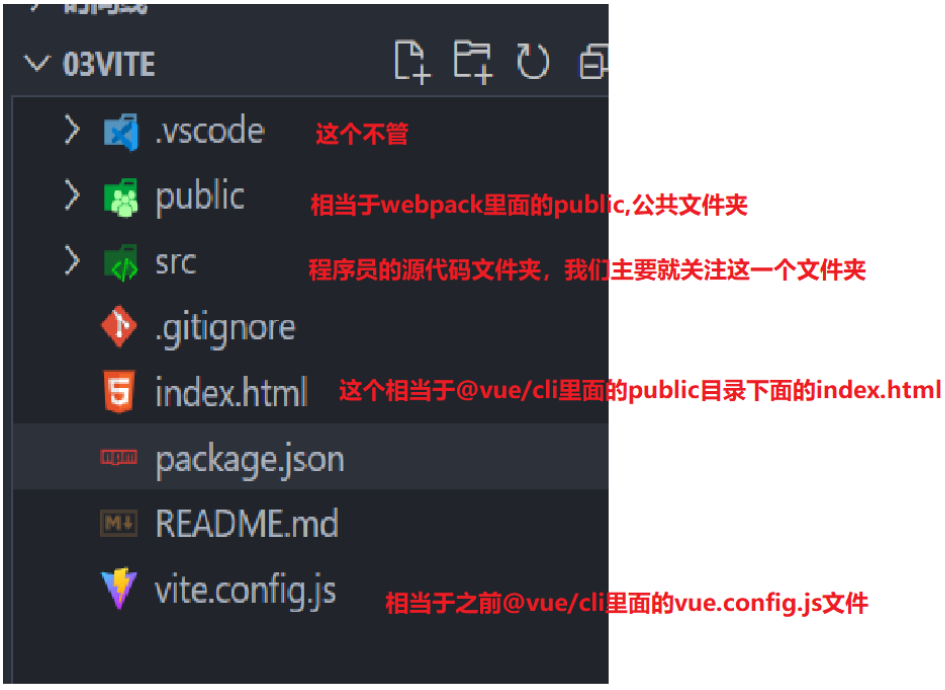
通过vite来创建vue的脚手架项目

```
$ npm create vite@latest
```

或直接创建的时候使用模板

```
# npm 6.x
npm create vite@latest my-vue-app --template vue

# npm 7+, extra double-dash is needed:
npm create vite@latest my-vue-app -- --template vue
```



main.js

```
import {createApp} from "vue";
// Vite里面默认是要导入后缀名的，而@vue/cli里面是不用后缀名的
import App from "./App.vue"
const app = createApp(App);

app.mount("#app")
```

App.vue

```
<template>
    <h2>App的组件</h2>
    <FoodItem></FoodItem>
</template>
<script>
import { ref, reactive, watch, onMounted } from "vue";
import FoodItem from "./components/FoodItem.vue";
export default {
    components: {
        FoodItem
    },
    setup() {

        const msg = ref("hello");

        const userInfo = reactive({
            userName: "张三",
            sex: "男"
        })

        return {
            msg,
            userInfo
        }
    }
}
</script>
<style scoped>

</style>
<!--
    通过上面的代码，我们可以得到一个点
    setup相当于结合了data,computed,methods,watch,生命周期
    除了这些与之前不一样了，其它的都一样
 -->
```

如果真是上面的情况，那就太好了，关键问题

通过上面的操作，我们发现，如果使用vue3的componsition api进行开发的时候，每次需要的东西都要进行 `return` 返回，这样太麻烦了，所幸，vue3在后期更新的时候把这种方式改良了一下，直接在 `script` 的标签上面加入 `setup` 的属性就可以实现语法糖的简化操作

```
<template>
  <h2>{{ msg }}</h2>
  <h2>{{ userInfo.nickName }}</h2>
  <h2>{{ userInfo.sex }}</h2>
</template>
```

```
<script setup>

// 所见即所得，所见即所用
import { ref, reactive } from "vue";

const msg = ref("疯了");
const userInfo = reactive({
  nickName: "标哥不疯了",
  sex: "男"
})
</script>
<style scoped>

</style>
```

defineProps接收组件参数

```
 <FoodItem sex="男" age="20"></FoodItem>
```

在内部可以如下方式接收

```
//数组语法
defineProps(["sex", "age"])
```

```
//对象语法
defineProps({
    sex:{
        type:String,
        default:"女"
    },
    age:{
        type:Number,
        required:true
    }
});
```