**实验二 计算属性、监听、响应式实验**

**一、实验目的：**

（1）练习掌握选项式API、组合式API和setup语法糖的使用

（2）练习掌握ref、reactive响应式的使用

（3）\uee77

（4）练习监听器的使用

**二、实验内容：**

（1）练习掌握选项式API的使用，要求在程序中包含ref响应式、reactive响应式数据定义及使用，计算属性的定义及使用，监听器的使用。响应式数据的定义要包含简单数据类型、数组和对象的定义，监听要包括对单数据类型、数组和对象的监听。

代码：

<template>

<h1>01-选项试API</h1>

<div>

  <p style="color: brown;">响应式数据简单数据类型</p>

  <button @click="**increaseCount**">Counut:{{ count }}</button>

</div>

<hr>

<div>

  <p style="color: brown;">响应式数据对象 计算属性</p>

  <p>name:{{ person.namePerson }}</p>

  <p>age:{{ person.age }}</p>

  <p>是否成年：{{ **isAdult** }}</p>

  <button @click="**changeName**">改变名字</button>

  <button @click="**increaseAge**">增长年龄</button>

  <button @click="**reduceAge**">减少年龄</button>

</div>

<hr>

<div>

  <p style="color: brown;">响应式数据数组 侦听器</p>

  <p v-for="item in books">书名：{{ item.bookName }}价格：{{ item.price }}</p>

  <p>库存{{ **bookInventory** }}本书</p>

  <button @click="**reduceBooks**">减少书籍</button>

  <p>库存的书（侦听器数组）</p>

  <p v-for="item in sellingBooks">书名：{{ item.bookName }}价格：{{ item.price }}</p>

</div>

<hr>

<div>

  <p style="color: brown;">响应式数据基本数据 对象</p>

  输入的值：<input v-model="dataMsg" /><br>

  旧值：{{oldMsg}} <br> <br>

  <button @click="**changePerson**">改变名字</button>  <br>

  有人改了名字,旧名字为{{ oldperson.namePerson }},新名字为{{ person.namePerson }}

</div>

</template>

<script lang="ts">

type Book = {

  bookName:string,

  price:number

}

type Person = {

  namePerson:string,

  age:number

}

export default {

  // data() 返回的属性将会成为响应式的状态

  // 并且暴露在 `this` 上

**data**() {

    // data中的都是响应式数据

    return{

      count:0,

      person:{

        namePerson:'jack',

        age:18

      },

      books:[

        {bookName:'vue3',price:20},

        {bookName:'myBatis',price:25},

        {bookName:'SpringBoot',price:30},

        {bookName:'MySql',price:35},

        {bookName:'FastAPI',price:40}

      ],

      sellingBooks:[] as Book[],  // 类型注解

      dataMsg:'' as string,

      oldMsg:'' as string,

      newPerson:{

        namePerson:'雪碧',

        age:18

      },

      oldperson: {

        namePerson:'雪碧',

        age:18

      },

    }

  },

  // methods 是一些用来更改状态与触发更新的函数

  // 它们可以在模板中作为事件处理器绑定

  methods: {

**increaseCount**(){

      this.count++

    },

**changeName**(){

      this.person.namePerson = 'mark'

    },

**increaseAge**(){

      this.person.age++

    },

**reduceAge**(){

      this.person.age--

    },

**reduceBooks**(){

      this.books.**pop**()

    },

**changePerson**(){

      this.person = this.newPerson

    }

  },

  // 计算属性

  computed:{

**isAdult**(){

      if (this.person.age >= 18) {

        return '成年'

      }else{

        return '未成年'

      }

    },

**bookInventory**(){

      return this.books.length

    }

  },

  // 侦听器 显示还有什么书，刚才卖出去了什么书

  watch: {

    books: {

**handler**(newValue: Book[], oldValue: Book[]) {

        if(newValue == oldValue){

          this.sellingBooks = newValue

        }

        console.**log**('newValue:', newValue);

        console.**log**('oldValue:', oldValue);

      },

      deep: true,

      immediate: true // 立即触发侦听器

    },

    dataMsg:{

**handler**(newValue,oldValue){

        this.oldMsg = oldValue

        console.**log**(newValue)

        }

    },

    person:{

**handler**(newValue:Person,oldValue:Person){

        this.oldperson = oldValue

        console.**log**(newValue)

        },

        deep:true

    }

  }

}

</script>

运行截图：

（2）练习掌握组合式API的使用，要求在程序中包含ref响应式、reactive响应式数据定义及使用，计算属性的定义及使用，监听器的使用。响应式数据的定义要包含简单数据类型、数组和对象的定义，监听要包括对单数据类型、数组和对象的监听。

代码：

<template>

    <h1>02-组合式API</h1>

    <div>

        <p style="color: brown;">响应式数据简单数据类型</p>

        <button @click="**increaseCount**">Counut:{{ count }}</button>

    </div>

    <hr>

    <div>

        <p style="color: brown;">响应式数据对象 计算属性</p>

        <p>name:{{ person.namePerson }}</p>

        <p>age:{{ person.age }}</p>

        <p>是否成年：{{ isAdult }}</p>

        <button @click="**changeName**">改变名字</button>

        <button @click="**increaseAge**">增长年龄</button>

        <button @click="**reduceAge**">减少年龄</button>

    </div>

    <hr>

    <div>

        <p style="color: brown;">响应式数据数组 侦听器</p>

        <p v-for="item in books">书名：{{ item.bookName }}价格：{{ item.price }}</p>

        <p>库存{{ bookInventory }}本书</p>

        <button @click="**reduceBooks**">减少书籍</button>

        <p>库存的书（侦听器数组）</p>

        <p v-for="item in sellingBooks">书名：{{ item.bookName }}价格：{{ item.price }}</p>

    </div>

    <hr>

    <div>

        <p style="color: brown;">响应式数据基本数据 对象</p>

        输入的值：<input v-model="dataMsg" /><br>

        旧值：{{ oldMsg }} <br> <br>

        <button @click="**changePerson**">改变名字</button> <br>

        有人改了名字,旧名字为{{ oldperson.namePerson }},新名字为{{ person.namePerson }}

    </div>

</template>

<script lang="ts">

// 导入相关函数

import { ref, reactive, computed, watch } from 'vue'

type Book = {

    bookName: string,

    price: number

}

type Person = {

    namePerson: string,

    age: number

}

export default {

**setup**() {

        // 定义变量

        let count = **ref**(0)

        let person = **reactive**({

            namePerson: 'jack',

            age: 18

        })

        let books = **reactive**([

            { bookName: 'vue3', price: 20 },

            { bookName: 'myBatis', price: 25 },

            { bookName: 'SpringBoot', price: 30 },

            { bookName: 'MySql', price: 35 },

            { bookName: 'FastAPI', price: 40 }

        ])

        let sellingBooks: Book[] = **reactive**([])

        let dataMsg = **ref**('')

        let oldMsg = **ref**('')

        let newPerson: Person = **reactive**({

            namePerson: '雪碧',

            age: 18

        })

        let oldperson: Person = **reactive**({

            namePerson: '雪碧',

            age: 18

        })

        // 定义方法

        function **increaseCount**() {

            count.value++

        }

        function **changeName**() {

            person.namePerson = 'mark'

        }

        function **increaseAge**() {

            person.age++

        }

        function **reduceAge**() {

            person.age--

        }

        function **reduceBooks**() {

            books.**pop**()

        }

        function **changePerson**() {

            person.namePerson = newPerson.namePerson

        }

        // 计算属性

        const isAdult = **computed**(() => {

            if (person.age >= 18) {

                return '成年'

            } else {

                return '未成年'

            }

        })

        const bookInventory = **computed**(() => {

            return books.length

        })

        // 侦听器

**watch**(books, (newValue: Book[], oldValue: Book[]) => {

            if (newValue == oldValue) {

                sellingBooks.length = 0; // 清空sellingBooks

                newValue.**forEach**((book) => {

                    sellingBooks.**push**({ ...book }); // 展开操作符,将books的元素添加到sellingBooks

                });

            }

            console.**log**("sellingBooks", sellingBooks)

            console.**log**('newValue:', newValue);

            console.**log**('oldValue:', oldValue);

        }

        );

**watch**(dataMsg, (newValue, oldValue) => {

            oldMsg.value = oldValue

            console.**log**(newValue)

        });

**watch**(

            () => ({ ...person }), // 使用对象的深度克隆来观察 person 对象的变化

            (newValue, oldValue) => {

                // oldValue 包含 person 的旧状态，而不是同一引用的对象

                oldperson = oldValue;

                console.**log**(newValue);

            }

        );

        return {

            // 手动暴露出来

            count,

            person,

            books,

            sellingBooks,

            dataMsg,

            oldMsg,

            newPerson,

            oldperson,

            // 暴露方法

**increaseCount**,

**changeName**,

**increaseAge**,

**reduceAge**,

**reduceBooks**,

**changePerson**,

            // 计算属性

            isAdult,

            bookInventory

        }

    }

}

</script>

运行截图：



（3）练习掌握组合式API setup语法糖的使用，要求在程序中包含ref响应式、reactive响应式数据定义及使用，计算属性的定义及使用，监听器的使用。响应式数据的定义要包含简单数据类型、数组和对象的定义，监听要包括对单数据类型、数组和对象的监听。

代码：

<template>

    <h1>03-SetUp 语法糖</h1>

    <div>

        <p style="color: brown;">响应式数据简单数据类型</p>

        <button @click="**increaseCount**">Counut:{{ count }}</button>

    </div>

    <hr>

    <div>

        <p style="color: brown;">响应式数据对象 计算属性</p>

        <p>name:{{ person.namePerson }}</p>

        <p>age:{{ person.age }}</p>

        <p>是否成年：{{ isAdult }}</p>

        <button @click="**changeName**">改变名字</button>

        <button @click="**increaseAge**">增长年龄</button>

        <button @click="**reduceAge**">减少年龄</button>

    </div>

    <hr>

    <div>

        <p style="color: brown;">响应式数据数组 侦听器</p>

        <p v-for="item in books">书名：{{ item.bookName }}价格：{{ item.price }}</p>

        <p>库存{{ bookInventory }}本书</p>

        <button @click="**reduceBooks**">减少书籍</button>

        <p>库存的书（侦听器数组）</p>

        <p v-for="item in sellingBooks">书名：{{ item.bookName }}价格：{{ item.price }}</p>

    </div>

    <hr>

    <div>

        <p style="color: brown;">响应式数据基本数据 对象</p>

        输入的值：<input v-model="dataMsg" /><br>

        旧值：{{ oldMsg }} <br> <br>

        <button @click="**changePerson**">改变名字</button> <br>

        有人改了名字,旧名字为{{ oldperson.namePerson }},新名字为{{ person.namePerson }}

    </div>

</template>

<script setup lang="ts">

// 导入相关函数

import { ref, reactive, computed, watch } from 'vue'

type Book = {

    bookName: string,

    price: number

}

type Person = {

    namePerson: string,

    age: number

}

// 定义变量

let count = **ref**(0)

let person = **reactive**({

    namePerson: 'jack',

    age: 18

})

let books = **reactive**([

    { bookName: 'vue3', price: 20 },

    { bookName: 'myBatis', price: 25 },

    { bookName: 'SpringBoot', price: 30 },

    { bookName: 'MySql', price: 35 },

    { bookName: 'FastAPI', price: 40 }

])

let sellingBooks: Book[] = **reactive**([])

let dataMsg = **ref**('')

let oldMsg = **ref**('')

let newPerson: Person = **reactive**({

    namePerson: '雪碧',

    age: 18

})

let oldperson: Person = **reactive**({

    namePerson: '雪碧',

    age: 18

})

// 定义方法

function **increaseCount**() {

    count.value++

}

function **changeName**() {

    person.namePerson = 'mark'

}

function **increaseAge**() {

    person.age++

}

function **reduceAge**() {

    person.age--

}

function **reduceBooks**() {

    books.**pop**()

}

function **changePerson**() {

    person.namePerson = newPerson.namePerson

}

// 计算属性

const isAdult = **computed**(() => {

    if (person.age >= 18) {

        return '成年'

    } else {

        return '未成年'

    }

})

const bookInventory = **computed**(() => {

    return books.length

})

// 侦听器

**watch**(books, (newValue: Book[], oldValue: Book[]) => {

    if (newValue == oldValue) {

        sellingBooks.length = 0; // 清空sellingBooks

        newValue.**forEach**((book) => {

            sellingBooks.**push**({ ...book }); // 展开操作符,将books的元素添加到sellingBooks

        });

    }

    console.**log**("sellingBooks", sellingBooks)

    console.**log**('newValue:', newValue);

    console.**log**('oldValue:', oldValue);

}

);

**watch**(dataMsg, (newValue, oldValue) => {

    oldMsg.value = oldValue

    console.**log**(newValue)

});

**watch**(

    () => ({ ...person }), // 使用对象的深度克隆来观察 person 对象的变化

    (newValue, oldValue) => {

        // oldValue 包含 person 的旧状态，而不是同一引用的对象

        oldperson = oldValue;

        console.**log**(newValue);

    }

);

</script>

运行截图：



**三、心得体会：**

**1. …是什么？**

是 JavaScript 中的展开操作符（Spread Operator）。在这种上下文中，... 用于将一个对象或数组展开为其包含的元素。

... 对象中的所有属性和值分别展开到新的对象中。这意呈现了一种复制或克隆对象的方式，以便创建一个具有相同属性和值的新对象。这确保了新对象与原对象具有不同的引用。

2. 为什么创建不同的引用就可以获取到旧值？

使用深度克隆创建了一个新的对象，并将其用作 watch 的侦听目标。深度克隆创建了一个全新的对象，具有与原对象相同的属性和值，但是它们具有不同的引用。因此，在 watch 中，当原始 person 对象发生变化时，oldValue 包含了克隆对象的旧状态，因为 oldValue 和 person 具有不同的引用。

Vue的响应式系统通过比较新值和旧值来检测数据的变化。当数据变化时，Vue会触发视图的更新。因此，确保 oldValue 和 newValue 具有不同的引用是非常重要的，否则Vue可能无法正确检测到变化。

通过深度克隆 person 对象，确保了 watch 中的 oldValue 包含了正确的旧状态，从而使 Vue 能够正确地检测和触发视图的更新。这是处理深度嵌套对象的常见做法，以确保数据响应性正常工作。

3. 什么是对象的深度克隆？

对象的深度克隆是指创建一个新对象，该对象包含与原始对象相同的属性和值，但是这些属性和值都是全新的，具有不同的引用。深度克隆不仅仅复制了原始对象的属性，还会递归地复制对象属性的属性，以确保所有嵌套的对象都具有不同的引用。

深度克隆通常在需要创建原始对象的副本，而不与原始对象共享引用时使用。这对于确保对一个对象的更改不会影响到另一个对象非常有用，尤其是在处理复杂的数据结构时