Vue第二节课

学习目标:

1、理解侦听器

2、掌握综合实例

一、侦听器watch

在 Vue 中， watch （侦听器）是一种用于监听数据变化并执行相应操作的特殊选项。通过定义一个 watch 对象，你可以监视指定的数据，并在数据变化时触发相应的回调函数。

Vue **侦听器提供了两种语法格式。一种是方法格式的写法，另一种是对象格式的写法。**

1、方法格式

```
watch:{// 侦听器
//以函数的方式侦听，函数有两个默认形参 分别代表 更新后和更新前的值 // 侦听对象:count
    count(newVal,oldVal){
        console.log('更新后的值',newVal);
        console.log('更新前的值',oldVal);
    }
}
```

语法举例:

```
<div id="div3">
<h1 >要求如下：
点击购买按钮，自动计算订单件数、总价（每件商品价格为10元）
</h1>
<button @click="shopping">点击购买商品</button>
<h3>当前订单件数:{{count}}</h3>
<h2>当前订单总价:{{jiaGe}}</h2>
</div>
```

watch 选项可以在 Vue 组件的选项对象中进行定义。以下是 watch 选项的基本语法：

```
let div3 = new Vue({
    el: "#div3",
    data : {
        count : 0
    },
    methods:{
        shopping(){
            this.count++
        }
    },
    computed: {
        jiaGe(){
            return this.count*20
        }
    },
```

```
    watch:{// 侦听器
        // 侦听对象: count ;
        //以函数的方式侦听, 函数有两个默认形参 分别代表 更新后和更新前的值
        count(newVal,oldVal){
            console.log('更新后的值',newVal);
            console.log('更新前的值',oldVal);
        }
    }
})
```

2、对象格式(推荐使用)

深度监听(deep)

深度监听（deep）是 Vue 中 watch 选项的一个配置项，用于开启对对象或数组内部属性的深度观测。默认情况下， Vue 的侦听器（watch）只会对数据的引用进行观察，当引用发生变化时才会触发侦听器。而深度监听（deep）可以让 Vue 深入观察对象或数组内部的属性，并在其内部属性发生变化时也触发侦听器。

一般用于监听对象， 可以深度监听到对象中的值

```
watch: {// 侦听器
    侦听对象:{
        // 这里的i是默认形参名，可以随意指定。
        handler(i){
            console.log(i.键);
            console.log('侦听到事情后 一般我们会做一些事情 就可以在侦听器内 来写逻辑代码');
        },
        // 开启侦听器 ，深度监听到对象中每一个属性的变化
        deep : true
    }
}
```

```
<div id="div3">
<h1>要求如下：点击购买按钮，自动计算订单件数、总价（每件商品价格为10元） </h1>
<button @click="shopping">点击购买商品</button>
<h3>当前订单件数:{{count.money}}</h3>
<h2>当前订单总价:{{jiaGe}}</h2>
</div>
```

```
let div3 = new Vue({
    el: "#div3",
    data: {
        count: {
            money:0
        }
    },
    methods: {
        shopping() {
            this.count.money++
        }
    },
    computed: {
        jiaGe() {
```

```
            return this.count.money * 20
        }
    },
    watch: {// 侦听器
        // 侦听对象: count
        count:{
            // 函数handler有1个默认形参
            handler(i){
                console.log(i.money);
            },
            // 开启侦听器
            // 控制侦听器深度监听到对象中每一个属性的变化
            deep : true
        }
    }
})
```

二、综合实例

vue全部知识点运用，实现一个可以添加删除的vue页面

1、添加操作

先创建好添加按钮标签，以及创建Vue实例，定义一个数组用来保存添加的数据和展示数组全部的数据。

1、methods:里面定义`addNetItem`添加元素方法，并且用`v-model`绑定`newitem`

2、button标签设置事件`@click="addNetItem"`

3、将`items`中的每个元素展示用`v-for`遍历输出

4、设置`computed`属性，来对应`p`标签的列表项目总数的结果。

5、用判断`v-if="isEmpty"`：当`items`里面没有元素，显示`列表为空`。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Vue Example</title>
```

```
    <!-- 开发环境版本 -->
    <script src="vue.js"></script>
</head>
<body>
<div id="app">
    <input type="text" v-model="newitem" placeholder="输入新项目">
    <button @click="addNetItem">添加</button>
    <p v-if="isEmpty">列表为空！</p>
    <p>列表项目总数: {{ count }}</p>
    <hr/>
    <ol>
        <!-- 列表项循环展示里面的元素数据 -->
        <li v-for="(item, index) in items" :key="index">
        <span>{{ item }}</span>
        </li>
    </ol>
</div>

<script type="text/javascript">
    // 创建Vue实例
    var app = new Vue({
        el: '#app',
        // 定义Vue实例的数据对象
        data: {
            // 初始列表项数组
            items: ['学习Vue', '学习Django基础', '学习Django前台'],
            // 输入框绑定的数据
            newitem: '',
        },
        // 计算属性，根据items数组的长度返回是否为空的布尔值
        computed: {
            isEmpty() {
                return this.items.length === 0;
            },
            // 计算属性，返回items数组的长度
            count() {
                return this.items.length;
            }
        },
        methods: {
            // 添加数据的方法
            addNetItem() {
                // 将输入的字符串去掉首尾两边的空白字符串然后赋值给trimmendItem
                let trimmedItem = this.newitem.trim();
                // 判断是否输入的是一个空字符串，如果不是空的就执行下面的代码
                if (trimmedItem) {
                    // 将输入的字符串添加到items这个数组中
                    this.items.push(trimmedItem);
                    // 清空输入框，将 newitem 的值设置为空字符串
                    // 这样做是为了防止用户重复点击添加按钮时添加重复的项
                    this.newitem = '';
                }
            }
        }
    })
</script>
</body>
</html>
```

2、删除操作

在method属性中，加入`deleteItem`函数来对`items`中的元素进行删除。

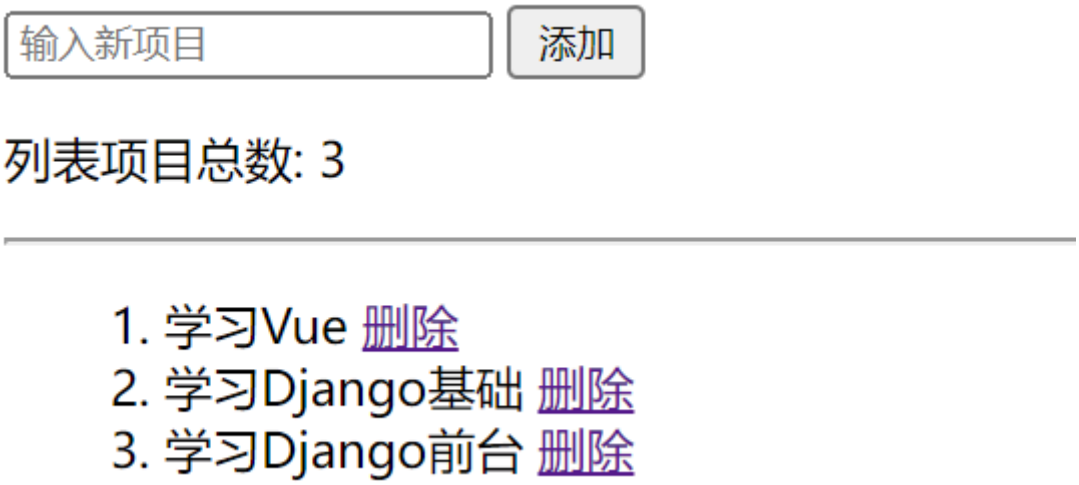
1、在`li`标签中添加对应删除操作`a`标签来绑定`deleteItem`函数

2、在Vue实例中`method`属性加入`deleteItem`方法（`splice`方法来对`items`中元素进行删除）

```
// 以上已经展示了程序整体，这里只写method属性中的代码 和  对应删除的标签代码

// 在li标签中加入一个a标签
<ol>
    <li v-for="(item, index) in items" :key="index">
        <span>{{ item }}</span>
        <!-- 删除按钮，点击时调用deleteItem方法 -->
        <a href="#" @click="deleteItem(index)">删除</a>
    </li>
</ol>

// vue实例中的代码
method:{
    // 删除指定索引的列表项
    deleteItem(index) {
        // 这个方法讲js第二节课讲过
        this.items.splice(index, 1);
    },
}
```



3、添加移动按钮,实现移动功能

如第一张图演示的结果，我们可以调节展示出来的**元素前后顺序**，`moveItem` 方法负责根据给定的当前索引 `currentIndex` 和移动方向 `direction` 来移动列表中的项。

1、body标签中写入向上向下移动的按钮，在显示内容item前后加入

`<a href="#" @click="moveItem(index, -1)" v-if="index > 0">↑</a>`

`<span>{{ item }}</span>`

`<a href="#" @click="moveItem(index, 1)" v-if="index < items.length - 1">↓</a>`

加入`v-if="index > 0"`:表示当前是第一个元素，则只有向下移动的操作。

`v-if="index < items.length - 1"`：表示当前是最后一个元素，则只有向上移动。

如果在这两种之间则有向上和向下移动的按钮。

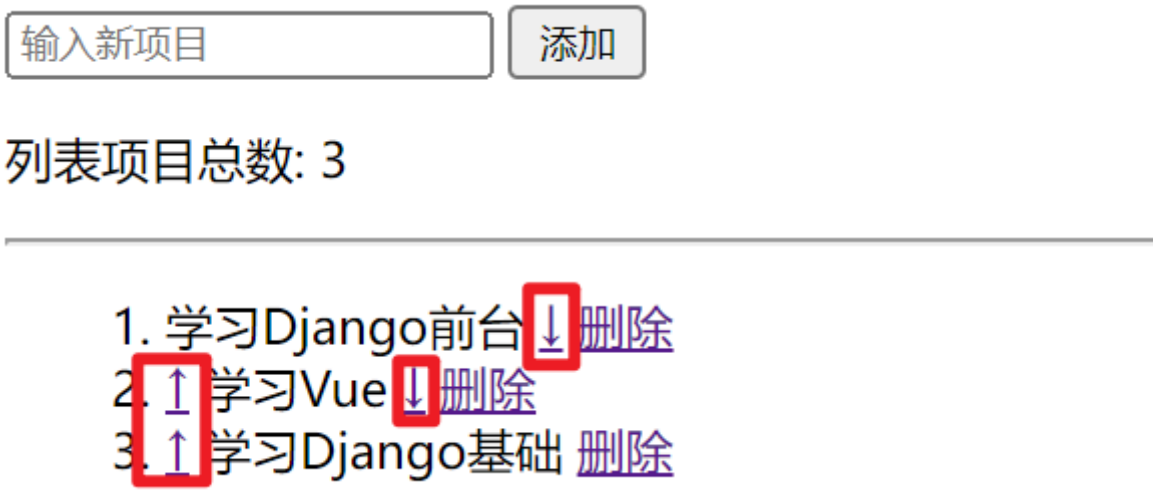
2、`method`属性中写入`moveItem`方法，来进行上下移动的功能。

```
// 以上已经展示了程序整体，这里只写method属性中的代码 和  对应删除的标签代码

// body中的代码在li标签中加入两个a标签
<ol>
    <li v-for="(item, index) in items" :key="index">
        <!-- 上移按钮，点击时调用moveItem方法并传递当前索引和-1（上移） -->
        <a href="#" @click="moveItem(index, -1)" v-if="index > 0">↑</a>
        <span>{{ item }}</span>
        <!-- 下移按钮，点击时调用moveItem方法并传递当前索引和1（下移） -->
        <a href="#" @click="moveItem(index, 1)" v-if="index < items.length -
1">↓</a>
        <a href="#" @click="deleteItem(index)">删除</a>
    </li>
</ol>

// vue实例中的代码
method:{
    moveItem(currentIndex, direction) {
    // 计算新索引，它是当前索引加上传入的方向值（-1 表示上移，1 表示下移）
    let newIndex = currentIndex + direction;
    // 检查新索引是否在数组的有效范围内（不小于0且小于数组长度）
    if (newIndex >= 0 && newIndex < this.items.length) {
```

```
            // 从原始位置移除项，currentIndex 为移除的起始索引，1 表示移除一个元素
            // 将移除的元素存储在变量 item 中
            let item = this.items.splice(currentIndex, 1)[0];
            // 将变量 item（即刚刚移除的元素）插入到新索引位置
            // newIndex 为插入的起始索引，0 表示在该位置插入不移除任何元素
            this.items.splice(newIndex, 0, item);
        }
        // 如果新索引不在有效范围内，则不执行任何操作，防止数组越界错误
    },
}
```



作业：自己实现好这个综合案例

截图发我效果