

# day02

## Vue常用特性

### 表单基本操作

- 获取单选框中的值
  - 通过v-model

```
<!--
  1、 两个单选框需要同时通过v-model 双向绑定 一个值
  2、 每一个单选框必须要有value属性 且value 值不能一样
  3、 当某一个单选框选中的时候 v-model 会将当前的 value值 改变 data 中的 数据

  gender 的值就是选中的值，我们只需要实时监控他的值就可以了
-->
<input type="radio" id="male" value="1" v-model='gender'>
<label for="male">男</label>

<input type="radio" id="female" value="2" v-model='gender'>
<label for="female">女</label>

<script>
  new Vue({
    data: {
      // 默认会让当前的 value 值为 2 的单选框选中
      gender: 2,
    },
  })
</script>
```

- 获取复选框中的值
  - 通过v-model
  - 和获取单选框中的值一样
  - 复选框 checkbox 这种的组合时 data 中的 hobby 我们要定义成数组 否则无法实现多选

```
<!--
  1、 复选框需要同时通过v-model 双向绑定 一个值
  2、 每一个复选框必须要有value属性 且value 值不能一样
  3、 当某一个单选框选中的时候 v-model 会将当前的 value值 改变 data 中的 数据

  hobby 的值就是选中的值，我们只需要实时监控他的值就可以了
-->

<div>
  <span>爱好: </span>
```

```

<input type="checkbox" id="ball" value="1" v-model='hobby'>
<label for="ball">篮球</label>
<input type="checkbox" id="sing" value="2" v-model='hobby'>
<label for="sing">唱歌</label>
<input type="checkbox" id="code" value="3" v-model='hobby'>
<label for="code">写代码</label>
</div>
<script>
  new Vue({
    data: {
      // 默认会让当前的 value 值为 2 和 3 的复选框选中
      hobby: ['2', '3'],
    },
  })
</script>

```

- 获取下拉框和文本框中的值
  - 通过v-model

```

<div>
  <span>职业: </span>
  <!--
    1、 需要给select 通过v-model 双向绑定 一个值
    2、 每一个option 必须要有value属性 且value 值不能一样
    3、 当某一个option选中的时候 v-model 会将当前的 value值 改变 data 中的 数据
       occupation 的值就是选中的值，我们只需要实时监控他的值就可以了
  -->
  <!-- multiple 多选 -->
  <select v-model='occupation' multiple>
    <option value="0">请选择职业...</option>
    <option value="1">教师</option>
    <option value="2">软件工程师</option>
    <option value="3">律师</option>
  </select>
  <!-- textarea 是 一个双标签 不需要绑定value 属性的 -->
  <textarea v-model='desc'></textarea>
</div>
<script>
  new Vue({
    data: {
      // 默认会让当前的 value 值为 2 和 3 的下拉框选中
      occupation: ['2', '3'],
      desc: 'nihao'
    },
  })
</script>

```

## 表单修饰符

- .number 转换为数值
  - 注意点：

- 当开始输入非数字的字符串时，因为Vue无法将字符串转换成数值
- 所以属性值将实时更新成相同的字符串。即使后面输入数字，也将被视作字符串。
- .trim 自动过滤用户输入的首尾空白字符
  - 只能去掉首尾的 不能去除中间的空格
- .lazy 将input事件切换成change事件
  - .lazy 修饰符延迟了同步更新属性值的时机。即将原本绑定在 input 事件的同步逻辑转变为绑定在 change 事件上
- 在失去焦点 或者 按下回车键时才更新

```
<!-- 自动将用户的输入值转为数值类型 -->
<input v-model.number="age" type="number">

<!--自动过滤用户输入的首尾空白字符 -->
<input v-model.trim="msg">

<!-- 在“change”时而非“input”时更新 -->
<input v-model.lazy="msg" >
```

## 自定义指令

- 内置指令不能满足我们特殊的需求
- Vue允许我们自定义指令

### Vue.directive 注册全局指令

```
<!--
  使用自定义的指令，只需在对用的元素中，加上'v-'的前缀形成类似于内部指令'v-if', 'v-text'的形式。
-->
<input type="text" v-focus>
<script>
// 注意点:
// 1、 在自定义指令中 如果以驼峰命名的方式定义 如 vue.directive('focusA',function(){})
// 2、 在HTML中使用的时候 只能通过 v-focus-a 来使用

// 注册一个全局自定义指令 v-focus
Vue.directive('focus', {
  // 当绑定元素插入到 DOM 中。 其中 el为dom元素
  inserted: function (el) {
    // 聚焦元素
    el.focus();
  }
});
new Vue({
  el: '#app'
});
</script>
```

### Vue.directive 注册全局指令 带参数

```
<input type="text" v-color='msg'>
```

```

<script type="text/javascript">
  /*
    自定义指令-带参数
    bind - 只调用一次，在指令第一次绑定到元素上时候调用

  */
  Vue.directive('color', {
    // bind声明周期，只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置
    // el 为当前自定义指令的DOM元素
    // binding 为自定义的函数形参 通过自定义属性传递过来的值 存在 binding.value 里面
    bind: function(el, binding){
      // 根据指令的参数设置背景色
      // console.log(binding.value.color)
      el.style.backgroundColor = binding.value.color;
    }
  });
  var vm = new Vue({
    el: '#app',
    data: {
      msg: {
        color: 'blue'
      }
    }
  });
</script>

```

## 自定义指令局部指令

- 局部指令，需要定义在 directives 的选项 用法和全局用法一样
- 局部指令只能在当前组件里面使用
- 当全局指令和局部指令同名时以局部指令为准

```

<input type="text" v-color='msg'>
<input type="text" v-focus>
<script type="text/javascript">
  /*
    自定义指令-局部指令
  */
  var vm = new Vue({
    el: '#app',
    data: {
      msg: {
        color: 'red'
      }
    },
    //局部指令，需要定义在 directives 的选项
    directives: {
      color: {
        bind: function(el, binding){
          el.style.backgroundColor = binding.value.color;
        }
      },
      focus: {

```

```

        inserted: function(el) {
            el.focus();
        }
    }
}
});
</script>

```

## 计算属性 computed

- 模板中放入太多的逻辑会让模板过重且难以维护 使用计算属性可以让模板更加的简洁
- **计算属性是基于它们的响应式依赖进行缓存的**
- computed比较适合对多个变量或者对象进行处理后返回一个结果值，也就是数多个变量中的某一个值发生了变化则我们监控的这个值也就会发生变化

```

<div id="app">
  <!--
    当多次调用 reverseString 的时候
    只要里面的 num 值不改变 他会把第一次计算的结果直接返回
    直到data 中的num值改变 计算属性才会重新发生计算
  -->
  <div>{{reverseString}}</div>
  <div>{{reverseString}}</div>
  <!-- 调用methods中的方法的时候 他每次会重新调用 -->
  <div>{{reverseMessage()}}</div>
  <div>{{reverseMessage()}}</div>
</div>
<script type="text/javascript">
  /*
    计算属性与方法的区别:计算属性是基于依赖进行缓存的, 而方法不缓存
  */
  var vm = new Vue({
    el: '#app',
    data: {
      msg: 'Nihao',
      num: 100
    },
    methods: {
      reverseMessage: function(){
        console.log('methods')
        return this.msg.split('').reverse().join('');
      }
    },
    //computed 属性 定义 和 data 已经 methods 同级
    computed: {
      // reverseString 这个是我们自己定义的名字
      reverseString: function(){
        console.log('computed')
        var total = 0;
        // 当data 中的 num 的值改变的时候 reverseString 会自动发生计算
        for(var i=0;i<=this.num;i++){
          total += i;
        }
      }
    }
  });

```

```

    }
    // 这里一定要有return 否则 调用 reversestring 的 时候无法拿到结果
    return total;
  }
}
});
</script>

```

## 侦听器 watch

- 使用watch来响应数据的变化
- 一般用于异步或者开销较大的操作
- watch 中的属性 一定是data 中 已经存在的数据
- 当需要监听一个对象的改变时，普通的watch方法无法监听到对象内部属性的改变，只有data中的数据才能够监听到变化，此时就需要deep属性对对象进行深度监听

```

<div id="app">
  <div>
    <span>名: </span>
    <span>
      <input type="text" v-model='firstName'>
    </span>
  </div>
  <div>
    <span>姓: </span>
    <span>
      <input type="text" v-model='lastName'>
    </span>
  </div>
  <div>{{fullName}}</div>
</div>

<script type="text/javascript">
  /*
    侦听器
  */
  var vm = new Vue({
    el: '#app',
    data: {
      firstName: 'Jim',
      lastName: 'Green',
      // fullName: 'Jim Green'
    },
    //watch 属性 定义 和 data 已经 methods 平级
    watch: {
      // 注意: 这里firstName 对应着data 中的 firstName
      // 当 firstName 值 改变的时候 会自动触发 watch
      firstName: function(val) {
        this.fullName = val + ' ' + this.lastName;
      },
      // 注意: 这里 lastName 对应着data 中的 lastName
      lastName: function(val) {

```

```

        this.fullName = this.firstName + ' ' + val;
    }
}
});
</script>

```

## 过滤器

- Vue.js允许自定义过滤器，可被用于一些常见的文本格式化。
- 过滤器可以用在两个地方：双花括号插值和v-bind表达式。
- 过滤器应该被添加在JavaScript表达式的尾部，由“管道”符号指示
- 支持级联操作
- 过滤器不改变真正的 data，而只是改变渲染的结果，并返回过滤后的版本
- 全局注册时是filter，没有s的。而局部过滤器是filters，是有s的

```

<div id="app">
  <input type="text" v-model='msg'>
  <!-- upper 被定义为接收单个参数的过滤器函数，表达式 msg 的值将作为参数传入到函数中 -->
  <div>{{msg | upper}}</div>
  <!--
    支持级联操作
    upper 被定义为接收单个参数的过滤器函数，表达式msg 的值将作为参数传入到函数中。
    然后继续调用同样被定义为接收单个参数的过滤器 lower，将upper 的结果传递到lower中
  -->
  <div>{{msg | upper | lower}}</div>
  <div :abc='msg | upper'>测试数据</div>
</div>

<script type="text/javascript">
  // lower 为全局过滤器
  vue.filter('lower', function(val) {
    return val.charAt(0).toLowerCase() + val.slice(1);
  });
  var vm = new Vue({
    el: '#app',
    data: {
      msg: ''
    },
    //filters 属性 定义 和 data 已经 methods 同级
    // 定义filters 中的过滤器为局部过滤器
    filters: {
      // upper 自定义的过滤器名字
      // upper 被定义为接收单个参数的过滤器函数，表达式 msg 的值将作为参数传入到函数中
      upper: function(val) {
        // 过滤器中一定要有返回值 这样外界使用过滤器的时候才能拿到结果
        return val.charAt(0).toUpperCase() + val.slice(1);
      }
    }
  });
</script>

```

## 过滤器中传递参数

```
<div id="box">
  <!--
    filterA 被定义为接收三个参数的过滤器函数。
    其中 message 的值作为第一个参数，
    普通字符串 'arg1' 作为第二个参数，表达式 arg2 的值作为第三个参数。
  -->
  {{ message | filterA('arg1', 'arg2') }}
</div>
<script>
  // 在过滤器中 第一个参数 对应的是 管道符前面的数据  n 此时对应 message
  // 第2个参数  a 对应 实参  arg1 字符串
  // 第3个参数  b 对应 实参  arg2 字符串
  Vue.filter('filterA',function(n,a,b){
    if(n<10){
      return n+a;
    }else{
      return n+b;
    }
  });

  new Vue({
    el:"#box",
    data:{
      message: "哈哈"
    }
  })
</script>
```

## 生命周期

- 事物从出生到死亡的过程
- Vue实例从创建 到销毁的过程，这些过程中会伴随着一些函数的自调用。我们称这些函数为钩子函数

### 常用的 钩子函数



beforeCreate	在实例初始化之后，数据观测和事件配置之前被调用 此时data 和 methods 以及页面的DOM结构都没有初始化 什么都做不了
created	在实例创建完成后被立即调用此时data 和 methods已经可以使用 但是页面还没有渲染出来
beforeMount	在挂载开始之前被调用 此时页面上还看不到真实数据 只是一个模板页面而已
mounted	el被新创建的vm.\$el替换，并挂载到实例上去之后调用该钩子。 数据已经真实渲染到页面上 在这个钩子函数里面我们可以使用一些第三方的插件
beforeUpdate	数据更新时调用，发生在虚拟DOM打补丁之前。 页面上数据还是旧的
updated	由于数据更改导致的虚拟DOM重新渲染和打补丁，在这之后会调用该钩子。 页面上数据已经替换成最新的
beforeDestroy	实例销毁之前调用
destroyed	实例销毁后调用

## 数组变异方法

- 在 Vue 中，直接修改对象属性的值无法触发响应式。当你直接修改了对象属性的值，你会发现，只有数据改了，但是页面内容并没有改变
- 变异数组方法即保持数组方法原有功能不变的前提下对其进行功能拓展

push()	往数组最后面添加一个元素，成功返回当前数组的长度
pop()	删除数组的最后一个元素，成功返回删除元素的值
shift()	删除数组的第一个元素，成功返回删除元素的值
unshift()	往数组最前面添加一个元素，成功返回当前数组的长度
splice()	有三个参数，第一个是想要删除的元素的下标（必选），第二个是想要删除的个数（必选），第三个是删除 后想要在原位置替换的值
sort()	sort() 使数组按照字符编码默认从小到大排序,成功返回排序后的数组
reverse()	reverse() 将数组倒序，成功返回倒序后的数组

## 替换数组

- 不会改变原始数组，但总是返回一个新数组

filter	filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。
concat	concat() 方法用于连接两个或多个数组。该方法不会改变现有的数组
slice	slice() 方法可从已有的数组中返回选定的元素。该方法并不会修改数组，而是返回一个子数组

## 动态数组响应式数据

- Vue.set(a,b,c) 让 触发视图重新更新一遍，数据动态起来
- a是要更改的数据、 b是数据的第几项、 c是更改后的数据

## 图书列表案例

- 静态列表效果
- 基于数据实现模板效果
- 处理每行的操作按钮

### 1、提供的静态数据

- 数据存放在vue 中 data 属性中

```
var vm = new Vue({
  el: '#app',
  data: {
    books: [{
      id: 1,
      name: '三国演义',
      date: ''
    }, {
      id: 2,
      name: '水浒传',
      date: ''
    }, {
      id: 3,
      name: '红楼梦',
      date: ''
    }, {
      id: 4,
      name: '西游记',
      date: ''
    }
  ]
}); var vm = new Vue({
  el: '#app',
  data: {
    books: [{
      id: 1,
      name: '三国演义',
      date: ''
    }, {
      id: 2,
      name: '水浒传',
      date: ''
    }, {
      id: 3,
      name: '红楼梦',
      date: ''
    }, {
      id: 4,
      name: '西游记',
      date: ''
    }
  ]
});
```

```
    }}  
  }  
});
```

## 2、把提供好的数据渲染到页面上

- 利用 v-for 循环 遍历 books 将每一项数据渲染到对应的数据中

```
<tbody>  
  <tr :key='item.id' v-for='item in books'>  
    <!-- 对应的id 渲染到页面上 -->  
    <td>{{item.id}}</td>  
    <!-- 对应的name 渲染到页面上 -->  
    <td>{{item.name}}</td>  
    <td>{{item.date}}</td>  
    <td>  
      <!-- 阻止 a 标签的默认跳转 -->  
      <a href="" @click.prevent>修改</a>  
      <span>|</span>  
      <a href="" @click.prevent>删除</a>  
    </td>  
  </tr>  
</tbody>
```

## 3、添加图书

- 通过双向绑定获取到输入框中的输入内容
- 给按钮添加点击事件
- 把输入框中的数据存储到 data 中的 books 里面

```
<div>  
  <h1>图书管理</h1>  
  <div class="book">  
    <div>  
      <label for="id">  
        编号:  
      </label>  
      <!-- 3.1、通过双向绑定获取到输入框中的输入的 id -->  
      <input type="text" id="id" v-model='id'>  
      <label for="name">  
        名称:  
      </label>  
      <!-- 3.2、通过双向绑定获取到输入框中的输入的 name -->  
      <input type="text" id="name" v-model='name'>  
      <!-- 3.3、给按钮添加点击事件 -->  
      <button @click='handle'>提交</button>  
    </div>  
  </div>  
</div>  
<script type="text/javascript">  
  /*  
    图书管理-添加图书
```

```

*/
var vm = new Vue({
  el: '#app',
  data: {
    id: '',
    name: '',
    books: [{
      id: 1,
      name: '三国演义',
      date: ''
    }, {
      id: 2,
      name: '水浒传',
      date: ''
    }, {
      id: 3,
      name: '红楼梦',
      date: ''
    }, {
      id: 4,
      name: '西游记',
      date: ''
    }
  ],
  methods: {
    handle: function(){
      // 3.4 定义一个新的对象book 存储 获取到输入框中 书 的id和名字
      var book = {};
      book.id = this.id;
      book.name = this.name;
      book.date = '';
      // 3.5 把book 通过数组的变异方法 push 放到 books 里面
      this.books.push(book);
      //3.6 清空输入框
      this.id = '';
      this.name = '';
    }
  }
});
</script>

```

## 4 修改图书-上

- 点击修改按钮的时候 获取到要修改的书籍名单
  - 4.1 给修改按钮添加点击事件， 需要把当前的图书的id 传递过去 这样才知道需要修改的是哪一本书籍
- 把需要修改的书籍名单填充到表单里面
  - 4.2 根据传递过来的id 查出books 中 对应书籍的详细信息
  - 4.3 把获取到的信息填充到表单

```

<div id="app">
  <div class="grid">
    <div>

```

```

<h1>图书管理</h1>
<div class="book">
  <div>
    <label for="id">
      编号:
    </label>
    <input type="text" id="id" v-model='id' :disabled="flag">
    <label for="name">
      名称:
    </label>
    <input type="text" id="name" v-model='name'>
    <button @click='handle'>提交</button>
  </div>
</div>
</div>
<table>
  <thead>
    <tr>
      <th>编号</th>
      <th>名称</th>
      <th>时间</th>
      <th>操作</th>
    </tr>
  </thead>
  <tbody>
    <tr :key='item.id' v-for='item in books'>
      <td>{{item.id}}</td>
      <td>{{item.name}}</td>
      <td>{{item.date}}</td>
      <td>
        <!--
          4.1 给修改按钮添加点击事件, 需要把当前的图书的id 传递过去
          这样才知道需要修改的是哪一本书籍
        -->
        <a href="" @click.prevent='toEdit(item.id)'>修改</a>
        <span>|</span>
        <a href="" @click.prevent>删除</a>
      </td>
    </tr>
  </tbody>
</table>
</div>
</div>
<script type="text/javascript">
  /*
    图书管理-添加图书
  */
  var vm = new Vue({
    el: '#app',
    data: {
      flag: false,
      id: '',
      name: '',

```

```

books: [{
  id: 1,
  name: '三国演义',
  date: ''
},{
  id: 2,
  name: '水浒传',
  date: ''
},{
  id: 3,
  name: '红楼梦',
  date: ''
},{
  id: 4,
  name: '西游记',
  date: ''
}]
},
methods: {
  handle: function(){
    // 3.4 定义一个新的对象book 存储 获取到输入框中 书 的id和名字
    var book = {};
    book.id = this.id;
    book.name = this.name;
    book.date = '';
    // 3.5 把book 通过数组的变异方法 push 放到 books 里面
    this.books.push(book);
    //3.6 清空输入框
    this.id = '';
    this.name = '';
  },
  toEdit: function(id){
    console.log(id)
    //4.2 根据传递过来的id 查出books 中 对应书籍的详细信息
    var book = this.books.filter(function(item){
      return item.id == id;
    });
    console.log(book)
    //4.3 把获取到的信息填充到表单
    // this.id 和 this.name 通过双向绑定 绑定到了表单中 一旦数据改变视图自动更新
    this.id = book[0].id;
    this.name = book[0].name;
  }
}
});
</script>

```

## 5 修改图书-下

- 5.1 定义一个标识符， 主要是控制 编辑状态下当前编辑书籍的id 不能被修改 即 处于编辑状态下 当前控制书籍编号的输入框禁用
- 5.2 通过属性绑定给书籍编号的 绑定 disabled 的属性 flag 为 true 即为禁用
- 5.3 flag 默认值为false 处于编辑状态 要把 flag 改为true 即当前表单为禁用

- 5.4 复用添加方法 用户点击提交的时候依然执行 handle 中的逻辑如果 flag为true 即 表单处于不可输入状态 此时执行的用户编辑数据数据

```

<div id="app">
  <div class="grid">
    <div>
      <h1>图书管理</h1>
      <div class="book">
        <div>
          <label for="id">
            编号:
          </label>
          <!-- 5.2 通过属性绑定 绑定 disabled 的属性 flag 为 true 即为禁用 -->
          <input type="text" id="id" v-model='id' :disabled="flag">
          <label for="name">
            名称:
          </label>
          <input type="text" id="name" v-model='name'>
          <button @click='handle'>提交</button>
        </div>
      </div>
    </div>
    <table>
      <thead>
        <tr>
          <th>编号</th>
          <th>名称</th>
          <th>时间</th>
          <th>操作</th>
        </tr>
      </thead>
      <tbody>
        <tr :key='item.id' v-for='item in books'>
          <td>{{item.id}}</td>
          <td>{{item.name}}</td>
          <td>{{item.date}}</td>
          <td>
            <a href="" @click.prevent='toEdit(item.id)'>修改</a>
            <span>|</span>
            <a href="" @click.prevent>删除</a>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
<script type="text/javascript">
  /*图书管理-添加图书*/
  var vm = new Vue({
    el: '#app',
    data: {
      // 5.1 定义一个标识符， 主要是控制 编辑状态下当前编辑书籍的id 不能被修改
      // 即 处于编辑状态下 当前控制书籍编号的输入框禁用
    }
  })

```

```

        flag: false,
        id: '',
        name: '',
    },
    methods: {
        handle: function() {
            /*
                5.4 复用添加方法 用户点击提交的时候依然执行 handle 中的逻辑
                如果 flag为true 即 表单处于不可输入状态 此时执行的用户编辑数据数据
            */
            if (this.flag) {
                // 编辑图书
                // 5.5 根据当前的ID去更新数组中对应的数据
                this.books.some((item) => {
                    if (item.id == this.id) {
                        // 箭头函数中 this 指向父级作用域的this
                        item.name = this.name;
                        // 完成更新操作之后, 需要终止循环
                        return true;
                    }
                });
                // 5.6 编辑完数据后表单要以可以输入的状态
                this.flag = false;
            } // 5.7 如果 flag为false 表单处于输入状态 此时执行的用户添加数据
            else {
                var book = {};
                book.id = this.id;
                book.name = this.name;
                book.date = '';
                this.books.push(book);
                // 清空表单
                this.id = '';
                this.name = '';
            }
            // 清空表单
            this.id = '';
            this.name = '';
        },
        toEdit: function(id) {
            /*
                5.3 flag 默认值为false 处于编辑状态 要把 flag 改为true 即当前表单为禁
            */
            this.flag = true;
            console.log(id)
            var book = this.books.filter(function(item) {
                return item.id == id;
            });
            console.log(book)
            this.id = book[0].id;
            this.name = book[0].name;
        }
    }
}

```

用



```

    }
  });
</script>

```

## 6 删除图书

- 6.1 给删除按钮添加事件 把当前需要删除的书籍id 传递过来
- 6.2 根据id从数组中查找元素的索引
- 6.3 根据索引删除数组元素

```

<tbody>
  <tr :key='item.id' v-for='item in books'>
    <td>{{item.id}}</td>
    <td>{{item.name}}</td>
    <td>{{item.date}}</td>
    <td>
      <a href="" @click.prevent='toEdit(item.id)'>修改</a>
      <span>|</span>
      <!-- 6.1 给删除按钮添加事件 把当前需要删除的书籍id 传递过来 -->
      <a href="" @click.prevent='deleteBook(item.id)'>删除</a>
    </td>
  </tr>
</tbody>
<script type="text/javascript">
  /*
    图书管理-添加图书
  */
  var vm = new Vue({
    methods: {
      deleteBook: function(id){
        // 删除图书
        /// 6.2 根据id从数组中查找元素的索引
        // var index = this.books.findIndex(function(item){
        //   return item.id == id;
        // });
        /// 6.3 根据索引删除数组元素
        // this.books.splice(index, 1);
        // -----
        /// 方法二：通过filter方法进行删除

        # 6.4 根据filter 方法 过滤出来id 不是要删除书籍的id
        # 因为 filter 是替换数组不会修改原始数据 所以需要 把 不是要删除书籍的id 赋值给 books
        this.books = this.books.filter(function(item){
          return item.id != id;
        });
      }
    }
  });
</script>

```

# 常用特性应用场景

## 1 过滤器

- Vue.filter 定义一个全局过滤器

```
<tr :key='item.id' v-for='item in books'>
  <td>{{item.id}}</td>
  <td>{{item.name}}</td>
  <!-- 1.3 调用过滤器 -->
  <td>{{item.date | format('yyyy-MM-dd hh:mm:ss')}}</td>
  <td>
    <a href="" @click.prevent='toEdit(item.id)'+>修改</a>
    <span>|</span>
    <a href="" @click.prevent='deleteBook(item.id)'+>删除</a>
  </td>
</tr>

<script>
#1.1 vue.filter 定义一个全局过滤器
Vue.filter('format', function(value, arg) {
  function dateFormat(date, format) {
    if (typeof date === "string") {
      var mts = date.match(/(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})/);
      if (mts && mts.length >= 3) {
        date = parseInt(mts[2]);
      }
    }
    date = new Date(date);
    if (!date || date.toUTCString() === "Invalid Date") {
      return "";
    }
    var map = {
      "M": date.getMonth() + 1, //月份
      "d": date.getDate(), //日
      "h": date.getHours(), //小时
      "m": date.getMinutes(), //分
      "s": date.getSeconds(), //秒
      "q": Math.floor((date.getMonth() + 3) / 3), //季度
      "S": date.getMilliseconds() //毫秒
    };
    format = format.replace(/([ymdhmsqS]+)/g, function(all, t) {
      var v = map[t];
      if (v !== undefined) {
        if (all.length > 1) {
          v = '0' + v;
          v = v.substr(v.length - 2);
        }
        return v;
      } else if (t === 'y') {
        return (date.getFullYear() + '').substr(4 - all.length);
      }
      return all;
    });
  }
  return value;
});
```

```

    });
    return format;
  }
  return dateFormat(value, arg);
})
#1.2 提供的数据 包含一个时间戳 为毫秒数
[
  {
    id: 1,
    name: '三国演义',
    date: 2525609975000
  }, {
    id: 2,
    name: '水浒传',
    date: 2525609975000
  }, {
    id: 3,
    name: '红楼梦',
    date: 2525609975000
  }, {
    id: 4,
    name: '西游记',
    date: 2525609975000
  }
];
</script>

```

## 2 自定义指令

- 让表单自动获取焦点
- 通过Vue.directive 自定义指定

```

<!-- 2.2 通过v-自定义属性名 调用自定义指令 -->
<input type="text" id="id" v-model='id' :disabled="flag" v-focus>

<script>
  # 2.1 通过Vue.directive 自定义指定
  Vue.directive('focus', {
    inserted: function (el) {
      el.focus();
    }
  });
</script>

```

## 3 计算属性

- 通过计算属性计算图书的总数
  - 图书的总数就是计算数组的长度

```

<div class="total">
  <span>图书总数: </span>
  <!-- 3.2 在页面上 展示出来 -->
  <span>{{total}}</span>

```

```
</div>

<script type="text/javascript">
  /*
    计算属性与方法的区别:计算属性是基于依赖进行缓存的, 而方法不缓存
  */
  var vm = new Vue({
    data: {
      flag: false,
      submitFlag: false,
      id: '',
      name: '',
      books: []
    },
    computed: {
      total: function(){
        // 3.1 计算图书的总数
        return this.books.length;
      }
    },
  });
</script>
```

## 生命周期

