

# 目标

目标

学习目标

全局作用域与局部作用域问题

remove删除节点

insertAdjacentHTML插入元素

参考网址：<https://developer.mozilla.org/zh-CN/docs/Web/API/Element/insertAdjacentHTML>

双击文字以后,可以编辑文字内容,失去焦点获取回车可以确定内容

面向过程版tab 栏切换

功能需求

删除功能

编辑功能

面向对象版tab 栏切换

功能需求

切换

添加

ES5新增方法

数组方法forEach遍历数组 常用

数组方法filter过滤数组 常用

数组方法some检测数组

数组方法map映射数组 了解

数组方法every检查数组 了解

## 查询商品案例

定义数组对象数据

需要先使用forEach遍历数据并渲染到页面中

根据价格筛选数据

根据商品名称筛选(作业)

## 今日总结

## 今日作业

使用面向对象的方式完成选项卡的删除功能

使用面向对象的方式完成选项卡的编辑功能

完成上课讲的案例,剩下根据商品名称筛选功能

完成当输入框输入文字进行实时匹配(模糊查询)

# 学习目标

- 完成面向过程版tab栏切换 把剩下的删除功能以及修改功能完成
- 完成面向对象版tab栏切换

# 全局作用域与局部作用域问题

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    // 在全局作用域下定义的变量就是全局变量
    // 在局部作用域下定义的变量就是局部变量
    // 全局作用域指的是函数外
    // 局部作用域指的是函数内
    // 全局变量在函数外,函数内都可以使用
    // 局部变量只能在函数内使用

    // 定义全局变量a
    var a = 10;
    function fn1(){
      console.log("在fn1函数里面输出a变量的值:" + a );
    }
    fn1();
    console.log("在fn1函数外面输出a变量的值:" + a );
    console.log("");

    // 解决方法,可以在函数外定义一个同名全局变量b,函数内赋值的时候,不加var关键字,这个时候函数内调用的就是全局变量了b
    var b = 100;
    function fn2(){
      // 在函数内使用var定义的变量就是局部变量了 只能在函数内才可以使用
      // var b = 20;

      // 把var去掉,这样使用的就是全局变量了
      b = 20;

      console.log("输出fn2函数里面的b变量" + b );
    }
    fn2();
    console.log("输出fn2函数外面的b变量" + b );

    /* function fn3(){
      console.log( b );
    }
    fn3(); */
  </script>
</head>
<body>

</body>
</html>

```

# remove删除节点

## 语法

- 1 要删除的节点.**remove()**

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <ul>
    <li>张三</li>
    <li>李四</li>
    <li>王五</li>
    <li>赵六</li>
    <li>孙七</li>
  </ul>

  <button>按钮1, 每点击一次, 就删除ul里面第一个子元素</button>
  <button>按钮2, 每点击一次, 就删除ul里面第一个子元素</button>
  <script>
    // 获取ul对象
    var ul = document.querySelector("ul");

    // 先使用以前删除节点的方式 父节点.removeChild(子节点)
    var btns = document.querySelectorAll("button");
    btns[0].onclick = function(){
      // console.log( ul.children[0] );
      // console.log( ul.firstChild );

      ul.removeChild( ul.children[0] );
    }

    // 使用 新方法 要删除的节点.remove();
    btns[1].onclick = function(){
      ul.children[0].remove();
    }
  </script>
</body>
</html>

```

## insertAdjacentHTML插入元素

MDN web docs  
mozilla

技术▼

参考与指南▼

反馈▼

 搜索 MDN

登录

element.insertAdjacentHTML

Web 开发技术 > Web API 接口参考 > Element > element.insertAdjacentHTML

中文 (简体) ▼

在此页面

语法

示例

注意

规范

浏览器兼容性

相关链接

相关主题

Document Object Model

Element

▼ 属性

accessKey

attributes

`insertAdjacentHTML()` 方法将指定的文本解析为 `Element` 元素，并将结果节点插入到 DOM 树中的指定位置。它不会重新解析它正在使用的元素，因此它不会破坏元素内的现有元素。这避免了额外的序列化步骤，使其比直接使用 `innerHTML` 操作更快。

## 语法

```
element.insertAdjacentHTML(position, text);
```

**position**

一个 `DOMString`，表示插入内容相对于元素的位置，并且必须是以下字符串之一：

- `'beforebegin'`：元素自身的前面。
- `'afterbegin'`：插入元素内部的第一个子节点之前。
- `'beforeend'`：插入元素内部的最后一个子节点之后。
- `'afterend'`：元素自身的后面。

参考网址：<https://developer.mozilla.org/zh-CN/docs/Web/API/Element/insertAdjacentHTML>

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div class="content"></div>
  <button>按钮1</button>
  <button>按钮2</button>
  <script>
    // 以前添加元素我们使用 父节点.appendChild(子节点)
    var content = document.querySelector(".content");
    var btns = document.querySelectorAll("button");
    btns[0].onclick = function(){
      var h1 = document.createElement("h1");
      h1.innerHTML = "我是标题1标签";
      content.appendChild( h1 );
    }
    btns[1].onclick = function(){
      // 父对象.insertAdjacentHTML("插入到哪里","字符串")
      content.insertAdjacentHTML("beforeend","<h2><span>h2标题里面的span标签</span></h2>");
    }
  </script>
</body>
</html>
```

双击文字以后,可以编辑文字内容,失去焦点获取回车  
可以确定内容

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    span{
      -webkit-touch-callout: none;
      -webkit-user-select: none;
      -khtml-user-select: none;
      -moz-user-select: none;
      -ms-user-select: none;
      user-select: none;
    }
  </style>
</head>
<body>
  <span>双击我可以编辑内容</span>
  <script>
    var span = document.querySelector("span");
    span.ondblclick = function(){
      // 获取span原来的内容
      var oldStr = span.innerHTML;
      // 设置span的标签内容为input标签 其实就是在span标签中添加了一个input子元素
      span.innerHTML = "<input type='text' />";
      // 得到span中第一个子元素,也就是input标签
      var input = span.children[0] ;
      // 设置input标签的内容
      input.value = oldStr;
      // 选中input标签中所有文字内容 语法: 文本框对象.select()
      input.select();

      // input失去焦点事件
      input.onblur = function(){
        // 把span标签内容设置为input的value值
        span.innerHTML = this.value;
      }

      // input键盘弹起事件
      input.onkeyup = function(e){
        if(e.keyCode == 13){
          // 手动调用失去焦点事件 对象名.事件名()
          this.blur();
        }
      }
    }
  </script>
</body>
</html>

```

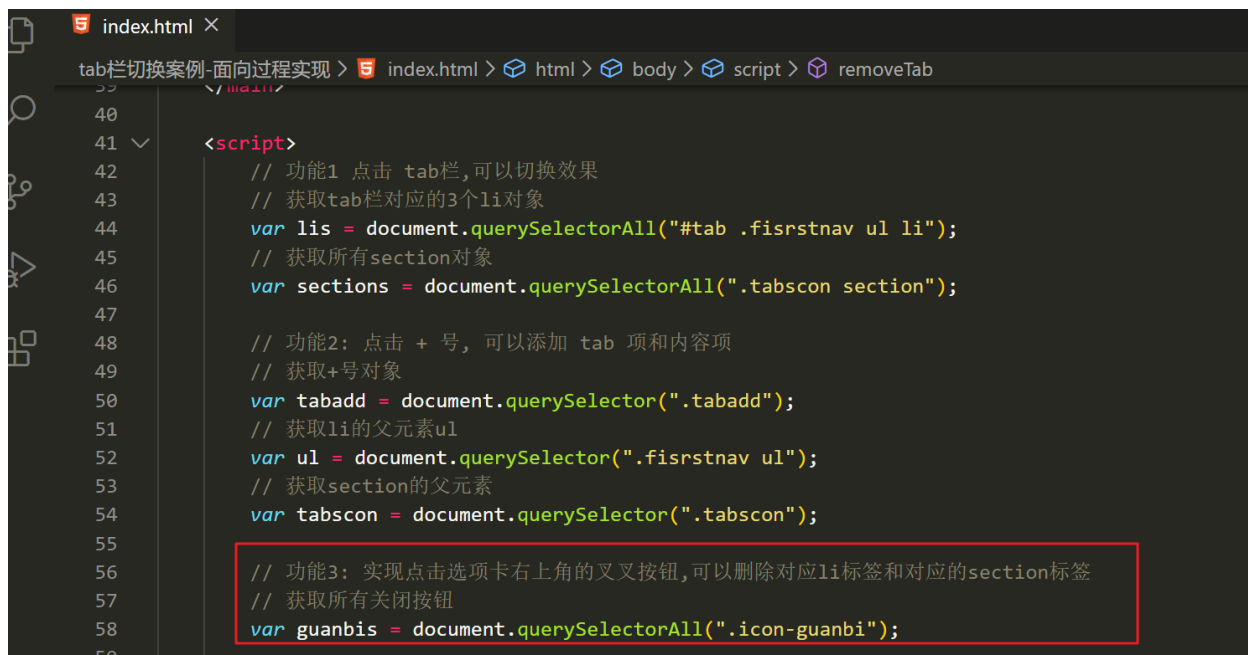
## 面向过程版tab 栏切换

## 功能需求

- 1 点击 **tab**栏,可以切换效果。
- 2 点击 **+** 号,可以添加 **tab** 项和内容项。
- 3 点击 **x** 号,可以删除当前的**tab**项和内容项。
- 4 双击**tab**项文字或者内容项文字,可以修改里面的文字内容

## 删除功能

1. 点击 **x** 可以删除当前的**li**选项卡和当前的**section**
2. **x**是没有索引号的,但是它的父亲**li** 有索引号,这个索引号正是我们想要的索引号
3. 所以核心思路是:点击 **x** 号可以删除这个索引号对应的 **li** 和 **section**
4. 但是,当我们动态删除新的**li**和索引号时,也需要重新获取 **x** 这个元素 需要调用**init**方法
5. **remove()**方法可以直接删除指定元素,就不需要使用我们之前学的**removeChild**方法了
6. 当我们删除的不是选中状态的**li**的时候,原来的选中状态**li**保保存不变
7. 当我们删除了选中状态的这个**li**的时候,让它的前一个**li**处于选定状态



```
index.html X
tab栏切换案例-面向过程实现 > index.html > html > body > script > removeTab
40
41 <script>
42 // 功能1 点击 tab栏,可以切换效果
43 // 获取tab栏对应的3个li对象
44 var lis = document.querySelectorAll("#tab .firsstnav ul li");
45 // 获取所有section对象
46 var sections = document.querySelectorAll(".tabscon section");
47
48 // 功能2: 点击 + 号,可以添加 tab 项和内容项
49 // 获取+号对象
50 var tabadd = document.querySelector(".tabadd");
51 // 获取li的父元素ul
52 var ul = document.querySelector(".firsstnav ul");
53 // 获取section的父元素
54 var tabscon = document.querySelector(".tabscon");
55
56 // 功能3: 实现点击选项卡右上角的叉叉按钮,可以删除对应li标签和对应的section标签
57 // 获取所有关闭按钮
58 var guanbi = document.querySelectorAll(".icon-guanbi");
59
```

```
// 因为我们一打开页面就需要给页面各种元素绑定相应的事件,比如li绑定点击切换功能,+号绑定点击添加tab等等
// init初始化给元素绑定事件
function init(){
    // 获取最近节点
    updateNode();

    // 循环给每个元素绑定事件
    for(var i=0;i<lis.length;i++){
        // 设置自定义属性
        lis[i].setAttribute("data-index" , i);

        // 给li绑定鼠标单击事件 可以给对象的事件驱动程序指定一个有名函数
        lis[i].onclick = toggleTab;

        // 给每个关闭按钮都绑定单击事件
        guanbis[i].onclick = removeTab;
    }

    // 因为+号只需要给一个元素绑定事件
    tabadd.onclick = addTab;
}
```



```

// 删除选项卡功能
function removeTab(e){
    // 因为关闭按钮绑定单击事件,他父元素li也有单击事件,所以会出现冒泡
    // 阻止冒泡
    e.stopPropagation();

    // 获取所点击选项卡的索引号,关闭按钮没有,但是关闭按钮的父元素有
    var index = this.parentNode.getAttribute("data-index");

    // 删除所点击的选项卡
    lis[index].remove();

    // 删除所点击的选项卡对应的内容层
    sections[index].remove();

    // 因为改变了选项卡的个数,所以需要调用一次初始化函数
    init();

    // 当我们删除了选中状态的这个li的时候,让它的前一个li处于选定状态
    if( this.parentNode.className == "liactive" ){
        // 让索引号-1 代表前一个的索引号
        index--;
        // 手动调用前一个li点击事件
        // &&如果前面的内容转成false, &&后面的代码就不执行
        lis[index] && lis[index].click();
    }
}

```

## 编辑功能

- 1 1. 双击选项卡li或者 section里面的文字,可以实现修改功能
- 2 2. 双击事件是: ondblclick
- 3 3. 如果双击文字,会默认选定文字,此时需要双击禁止选中文字
- 4 4. 禁止选中文字方法有两种
- 5 第一种:js解决方法
- 6 window.getSelection ? window.getSelection().removeAllRanges() :  
document.selection.empty();
- 7
- 8 第二种:css解决方法 给相应标签设置以下样式
- 9 -webkit-touch-callout: none;
- 10 -webkit-user-select: none;

```
11     -khtml-user-select: none;
12     -moz-user-select: none;
13     -ms-user-select: none;
14     user-select: none;
15
```

16 5. 核心思路：双击文字的时候，在 里面生成一个文本框，当失去焦点或者按下回车然后把文本框输入的值给原先

17 元素即可。

18 6. 文本框对象.`select()` 可以文本框里面的文字处于选定状态

19 7. 当我们文本框失去焦点的时候,可以通过span元素的innerHTML得到文本框里面的值

20 8. 按下回车也可以把文本框里面的值给span

21 9. 每个section也要实现双击可以编辑的功能

```
69
70     // 获取所有li中第一个span标签
71     var spans = document.querySelectorAll(".fistrstnav ul li span:first-child");
72
```

```
// 更新节点,获取当前最新所有li和所有section元素
function updateNode(){
    // 获取当前所有tab栏中li对象
    lis = document.querySelectorAll("#tab .fistrstnav ul li");
    // 获取当前所有section对象
    sections = document.querySelectorAll(".tabscon section");
    // 获取当前所有的关闭按钮对象
    guanbis = document.querySelectorAll(".icon-guanbi");
    // 获取当前所有li里面的第一个span对象
    spans = document.querySelectorAll(".fistrstnav ul li span:first-child");
}
```

```

// 编辑功能
function editTab(){
    // 获取span原来的内容
    var oldStr = this.innerHTML;
    // 设置span的标签内容为input标签 其实就是往span标签中添加了一个input子元素
    this.innerHTML = "<input type='text' />";
    // 得到span中第一个子元素,也就是input标签
    var input = this.children[0] ;
    // 设置input标签的内容
    input.value = oldStr;
    // 选中input标签中所有文字内容 语法: 文本框对象.select()
    input.select();
    // input失去焦点事件
    input.onblur = function(){
        // 把span标签内容设置为input的value值
        this.parentNode.innerHTML = this.value;
    }
    // input键盘弹起事件
    input.onkeyup = function(e){
        if(e.keyCode == 13){
            // 手动调用失去焦点事件 对象名.事件名()
            this.blur();
        }
    }
}

```

```

function init(){
    // 获取最近节点
    updateNode();

    // 循环给每个元素绑定事件
    for(var i=0;i<lis.length;i++){
        // 设置自定义属性
        lis[i].setAttribute("data-index" , i);

        // 给li绑定鼠标单击事件 可以给对象的事件驱动程序指定一个有名函数
        lis[i].onclick = toggleTab;

        // 给每个关闭按钮都绑定单击事件
        guanbis[i].onclick = removeTab;

        // 给每个span绑定双击事件
        spans[i].ondblclick = editTab;

        // 给每个section绑定双击事件
        sections[i].ondblclick = editTab;
    }

    // 因为+号只需要给一个元素绑定事件
    tabadd.onclick = addTab;
}

```

## 面向对象版tab 栏切换

## 功能需求

- 1 点击 **tab**栏,可以切换效果。
- 2 点击 **+** 号,可以添加 **tab** 项和内容项。
- 3 点击 **x** 号,可以删除当前的**tab**项和内容项。
- 4 双击**tab**项文字或者内容项文字,可以修改里面的文字内容

## 切换

- 1 为获取到的标题绑定点击事件,展示对应的内容区域,存储对应的索引
- 2 使用排他,实现只有一个元素的显示
- 3 因为清除所有标题元素跟所有内容元素的类属性这个功能后续还会用到,所以我们定义成类中的一个方法

```
<script>
// 面向对象就是把所有的功能分配到各个对象中,功能作为对象的方法即可,指挥对象工作
// 类 跟 实例对象
// 根据效果得到我们需要一个选项卡类
// 选项卡类具有什么属性具有什么功能
// 属性: 各种HTML元素就会作为这个类的属性
// 功能: 选项卡可以切换选项卡,增加选项卡,删除选项卡以及编辑选项卡对应的内容

// 定义一个变量that,用于缓存this
var that;

// 通过es6的方法定义一个类,类名需要大写
class Tab{
  // 构造器,初始化属性,给属性赋值
  constructor(id){
    // 缓存this,在实例化以后,会自动调用constructor()方法,这里的this就是指向实例对象
    that = this;

    // 类里面的属性需要使用this
    this.main = document.querySelector(id);
    this.lis = this.main.querySelectorAll(".firstnav ul li");
    this.sections = this.main.querySelectorAll(".tabscon section");

    // 类里面使用类的属性和方法都要记得加上this前缀
    this.init();
  }

  // 初始化操作,给页面相关元素绑定事件
  init(){
    for(var i=0;i<this.lis.length;i++){
      // 设置自定义属性 给每个li设置data-index属性
      this.lis[i].setAttribute("data-index",i);
      // 给每个li绑定鼠标单击事件
      this.lis[i].onclick = this.toggleTab;
    }
  }
}
```

```

// 清除所有li和section的类名
clearClass(){
    for(var j=0; j<this.lis.length; j++){
        this.lis[j].removeAttribute("class");
        this.sections[j].removeAttribute("class");
    }
}

// 切换选项卡
toggleTab(){
    // console.log( that );

    // 实例化对象才可以调用类定义的属性和方法
    // console.log( this );// this在这里代表点击的那个li
    // this.clearClass();

    // 排他操作,清除所有样式
    that.clearClass();

    // 单独设置点击的那个选项卡的样式
    this.className = "liactive";
    // 设置对应的内容层样式
    var index = this.getAttribute("data-index");
    that.sections[index].className = "conactive";
}

addTab(){
    console.log("添加选项卡");
}

removeTab(){
    console.log("删除选项卡");
}

editTab(){
    console.log("编辑选项卡");
}
}

new Tab("#tab");
</script>

```

## 添加

- 1 **1.** 点击 + 可以实现添加新的选项卡和内容
- 2 **2.** 第一步：创建新的选项卡li 和 新的 内容 section
- 3 **3.** 第二步：把创建的两个元素追加到对应的父元素中
- 4 **4.** 以前的做法：动态创建元素 createElement ，但是元素里面内容较多，需要innerHTML赋值,在 appendChild
- 5 追加到父元素里面。
- 6 **5.** 现在高级做法：利用 insertAdjacentHTML() 可以直接把字符串格式元素添加到父元素中
- 7 **6.** appendChild 不支持追加字符串的子元素，insertAdjacentHTML 支持追加字符串的元素
- 8 **7.** insertAdjacentHTML(追加的位置,"要追加的字符串元素")

- 9 8. 追加的位置有: beforeend 插入元素内部的最后一个子节点之后
- 10 9. 该方法地址: <https://developer.mozilla.org/zh-CN/docs/Web/API/Element/insertAdjacentHTML>

- 1 为添加按钮+ 绑定点击事件
- 2 实现标题与内容的添加,做好排他处理
- 3 但是新添加的li跟section没有切换功能,我们需要解决这个问题

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>面向对象 Tab</title>
  <link rel="stylesheet" href="./styles/tab.css">
  <link rel="stylesheet" href="./styles/style.css">
</head>

<body>

  <main>
    <h4>
      Js 面向对象 动态添加标签页
    </h4>
    <div class="tabsox" id="tab">
      <!-- tab 标签 -->
      <nav class="firsstnav">
        <ul>
          <li class="liactive"><span>测试1</span><span class="iconfont icon-guanbi"></span></li>
          <li><span>测试2</span><span class="iconfont icon-guanbi"></span></li>
          <li><span>测试3</span><span class="iconfont icon-guanbi"></span></li>
        </ul>
        <div class="tabadd">
          <span></span>
        </div>
      </nav>

      <!-- tab 内容 -->
      <div class="tabsocon">
        <section class="conactive">测试1</section>
        <section>测试2</section>
        <section>测试3</section>
      </div>
    </div>
  </main>

  <script>
    // 面向对象就是把所有的功能分配到各个对象中,功能作为对象的方法即可,指挥对象工作
    // 类 跟 实例对象
    // 根据效果得到我们需要一个选项卡类
    // 选项卡类具有什么属性具有什么功能
    // 属性: 各种HTML元素就会作为这个类的属性
    // 功能: 选项卡可以切换选项卡,增加选项卡,删除选项卡以及编辑选项卡对应的内容

    // 定义一个变量that,用于缓存this
    var that;

    // 通过es6的方法定义一个类,类名需要大写
    class Tab{
      // 构造器,初始化属性,给属性赋值
      constructor(id){
        // 缓存this,在实例化以后,会自动调用constructor()方法,这里的this就是指向实例对象
        that = this;
      }
    }
  </script>

```

```

// 类里面的属性需要使用this
this.main = document.querySelector(id);
this.lis = this.main.querySelectorAll(".firsstnav ul li");
this.sections = this.main.querySelectorAll(".tabscon section");
this.tabadd = this.main.querySelector(".tabadd");
this.ul = this.main.querySelector(".firsstnav ul");
this.tabscon = this.main.querySelector(".tabscon");

// 类里面使用类的属性和方法都要记得加上this前缀
this.init();
}

// 更新节点的方法
updateNode(){
    this.lis = this.main.querySelectorAll(".firsstnav ul li");
    this.sections = this.main.querySelectorAll(".tabscon section");
}

// 初始化操作,给页面相关元素绑定事件
init(){
    // 调用更新节点的方法
    this.updateNode();

    for(var i=0;i<this.lis.length;i++){
        // 设置自定义属性 给每个li设置data-index属性
        this.lis[i].setAttribute("data-index",i);
        // 给每个li绑定鼠标单击事件
        this.lis[i].onclick = this.toggleTab;
    }

    // 给+号绑定单击事件
    this.tabadd.onclick = this.addTab;
}

// 清除所有li和section的类名
removeClass(){
    for(var j=0; j<this.lis.length; j++){
        this.lis[j].removeAttribute("class");
        this.sections[j].removeAttribute("class");
    }
}

// 切换选项卡
toggleTab(){
    // console.log( that );

    // 实例化对象才可以调用类定义的属性和方法
    // console.log( this );// this在这里代表点击的那个li
    // this.removeClass();

    // 排他操作,清除所有样式
    that.removeClass();

    // 单独设置点击的那个选项卡的样式
    this.className = "liactive";
    // 设置对应的内容层样式
    var index = this.getAttribute("data-index");
    that.sections[index].className = "conactive";
}

// 添加选项卡
addTab(){
    that.removeClass();

    var li = '<li class="liactive"><span>新选项卡</span><span class="iconfont icon-guanbi"></span></li>';
    that.ul.insertAdjacentHTML("beforeend",li);

    var section = '<section class="conactive">测试 '+Math.random()+ '</section>';
    that.tabscon.insertAdjacentHTML("beforeend",section);

    that.init();
}

```

```
// 删除选项卡
removeTab(){
  console.log("删除选项卡");
}

// 编辑选项卡
editTab(){
  console.log("编辑选项卡");
}

new Tab("#tab");
</script>
</body>

</html>
```

## ES5新增方法

ES5 中给我们新增了一些方法，可以很方便的操作数组或者字符串，这些方法主要包括：

- 数组方法
- 字符串方法( trim去掉字符串左右空格符,已讲)
- 对象方法

### 数组方法

迭代(遍历)方法：**forEach()**、**filter()**、some()、map()、every();

### 数组方法forEach遍历数组 常用

语法:

```
1 arr.forEach(function(currentValue, index, arr) {})  
2 //相当于数组遍历的 for循环 没有返回值  
3  
4 currentValue: 数组当前项的值  
5 index: 可选,数组当前项的索引  
6 arr: 可选,数组对象本身
```

举例:



```

<script>
    var arr = [10,20,30,40,50];
    /*
        forEach语法

        数组对象.forEach( 回调函数 )
        数组对象.forEach( function(value,index,array){
            value代表当前数组元素的值,可选
            index代表当前数组元素的下标,可选
            array代表遍历正在遍历的那个数组,可选
        } )

        注意: forEach没有返回值  forEach中写return不会终止forEach的执行
        */

    var result = arr.forEach(function(value){
        console.log( value );
        // return false;
    })
</script>

```

## 数组方法filter过滤数组 常用

### 语法:

- 1 array.filter(function(currentValue, index, arr){})
- 2 currentValue: 数组当前项的值
- 3 index: 可选,数组当前项的索引
- 4 arr: 可选,数组对象本身

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,主要用于筛选数组注意:它直接返回一个新数组

### 举例:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    var arr = [5,7,2,98,90,95,80,85];
    // filter有过滤的意思,主要用于“筛选”满足条件的数组元素,返回值是一个满足条件元素组成的新数组 需要return一个布尔值 如果为true的,才会放进返回值数组中
    /* 语法
    array.filter(function(currentValue, index, arr){})
    currentValue: 数组当前项的值
    index: 可选,数组当前项的索引
    arr: 可选,数组对象本身 */

    var newArr = arr.filter( function(value) {
      // 返回数组元素值大于等于80的元素
      return value >= 80;
    });

    console.log( newArr );
  </script>
</body>
</html>

```

## 数组方法some检测数组

语法:

```

1 array.some(function(currentValue, index, arr){})
2
3 currentValue: 数组当前项的值
4 index: 可选,数组当前项的索引
5 arr: 可选,数组对象本身
6
7 回调函数中要写return

```

- some() 方法用于检测数组中的元素是否满足指定条件. 通俗点 查找数组中是否有满足条件的元素
- 注意它返回值是布尔值, 如果查找到这个元素, 就返回true, 如果查找不到就返回false.
- 如果找到第一个满足条件的元素,则终止循环. 不在继续查找.

举例:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    // 数组方法some检测数组
    /* 语法:
    array.some(function(currentValue, index, arr){})

    currentValue: 数组当前项的值
    index: 可选,数组当前项的索引
    arr: 可选,数组对象本身 */

    // 回调函数中要写return

    // some() 方法用于检测数组中的元素是否满足指定条件。通俗点 查找数组中是否有满足条件的元素
    // 注意它返回值是布尔值, 如果查找到这个元素, 就返回true , 如果查找不到就返回false.
    // 如果找到第一个满足条件的元素,则终止循环。不在继续查找。

    var arr = [10,20,30,20,50,70];
    var result1 = arr.some(function(value){
      console.log( value );
      return value > 100;
    });
    console.log( result1 );
    console.log( "" );

    var result2 = arr.some(function(value){
      console.log( value );
      return value < 30;
    });
    console.log( result2 );

    // some能做的,filter几乎都可以做,只是some找到满足条件的第一个元素,就会终止some代码的执行,返回值是一个布尔值
  </script>
</body>
</html>

```

## 数组方法map映射数组 了解

逐一处理原数组元素，返回到一个新数组

语法:

```

1 array.map(function(currentValue, index, arr){})
2
3 currentValue: 数组当前项的值
4 index: 可选,数组当前项的索引
5 arr: 可选,数组对象本身
6
7 返回值是回调函数的结果组成的每一个元素的新数组
8 回调函数中要写return

```

举例:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    // 数组方法map映射数组 逐一处理原数组元素，返回到一个新数组
    var arr = [10,50,5,6,3,8];
    // var newArr = [];
    // 想让每个数组元素都变成自己原来2倍
    /* for(var i=0;i<arr.length;i++){
      newArr.push( arr[i]*2 );
    } */

    /* arr.forEach(function(value){
      newArr.push( value*2 );
    }) */

    // console.log( newArr );

    var newArr = arr.map(function(value){
      return value*5;
    })
    console.log( newArr );
  </script>
</body>
</html>

```

## 数组方法every检查数组 了解

every 检测数组中的元素是否存在满足条件的值，全部满足条件返回true； 有一个不满足就返回false，不对空数组进行检测，不改变原数组

### 语法:

- 1 语法:
- 2 array.every(function(currentValue, index, arr){})
- 3
- 4 currentValue: 数组当前项的值
- 5 index: 可选,数组当前项的索引
- 6 arr: 可选,数组对象本身
- 7

- 8 返回值是一个布尔值
- 9
- 10 注意：全部元素在条件中成立，则为true

## 举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    // 数组方法every检查数组
    // every 检测数组中的元素是否存在满足条件的值，全部满足条件返回true； 有一个不满足就返回false，不对空数组进行检测，不改变原数组
    var arr = [85,58,70,52,92,40];
    var result1 = arr.every(function(value){
      return value > 60;
    });
    console.log( result1 );

    var result2 = arr.every(function(value){
      return value > 30;
    });
    console.log( result2 );
  </script>
</body>
</html>
```

## 查询商品案例

- 1 1. 把数据渲染到页面中（forEach）
- 2 2. 根据价格显示数据
- 3 3. 根据商品名称显示数据

## 定义数组对象数据

```
1 // 1.定义数组对象数据
2 var data = [{
3   id: 1,
4   pname: '小米',
5   price: 3999
6 }, {
7   id: 2,
8   pname: 'oppo',
9   price: 999
10 }, {
11   id: 3,
12   pname: '荣耀',
13   price: 1299
14 }, {
```

```
15     id: 4,  
16     pname: '华为',  
17     price: 1999  
18 }, ];
```

## 需要先使用forEach遍历数据并渲染到页面中

## 根据价格筛选数据

```
1  获取到搜索按钮并为其绑定点击事件,使用数组的filter方法将用户输入的价格信息筛选出来
```

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <style>
    table {
      width: 400px;
      border: 1px solid #000;
      border-collapse: collapse;
      margin: 0 auto;
    }

    td,
    th {
      border: 1px solid #000;
      text-align: center;
    }

    input {
      width: 50px;
    }

    .search {
      width: 600px;
      margin: 20px auto;
    }
  </style>
</head>

<body>
  <div class="search">
    按照价格查询: <input type="text" class="start"> - <input type="text" class="end"> <button class="search-price">搜索</button> 按照商品名称查询: <input type="text" class="product"> <button class="search-pro">查询</button>
  </div>
  <table>
    <thead>
      <tr>
        <th>id</th>
        <th>产品名称</th>
        <th>价格</th>
      </tr>
    </thead>
    <tbody>

    </tbody>
  </table>

  <script>
    // 1. 定义数组对象数据
    var data = [{
      id: 1,
      pname: '小米',
      price: 3999
    }, {
      id: 2,
      pname: 'oppo',
      price: 999
    }, {
      id: 3,
      pname: '联想',
      price: 1299
    }, {
      id: 4,
      pname: '华为',
      price: 1999
    },
  ];

    // 获取tbody元素
    var tbody = document.querySelector("tbody");

    // 封装一个显示数据的方法
    function setData(myData){
      // 清空tbody中的内容
      tbody.innerHTML = "";

      myData.forEach(function(value){
        // 创建tr元素
        var tr = document.createElement("tr");
        tr.innerHTML = "<td>" + value.id + "</td><td>" + value.pname + "</td><td>" + value.price + "</td>";
        // 追加tr到tbody元素中
        tbody.appendChild( tr );
      })
    }

    // 遍历显示表格数据
    setData( data );

    // 根据价格范围搜索满足条件的元素
    // 获取相关对象
    var start = document.querySelector(".start");
    var end = document.querySelector(".end");
    var searchPrice = document.querySelector(".search-price");
    searchPrice.onclick = function(){
      var startValue = start.value;
      var endValue = end.value;

      // 我们想从一个数组里面筛选出满足条件的元素,并且放进新数组中,那么可以使用filter
      // filter函数返回的是一个新数组
      var newData = data.filter(function(value){
        // 判断价格是否符合范围
        return startValue <= value.price && value.price <= endValue;
      });

      // 遍历筛选出来数据
      setData( newData );
    }
  </script>
</body>
</html>

```

## 根据商品名称筛选(作业)

- 1 获取用户输入的商品名称,为查询按钮绑定点击事件,将输入的商品名称与这个数据进行筛选
- 2 如果查询数组中唯一的元素,用some方法更合适,因为它找到这个元素,就不再进行循环,效率更高

# 今日总结

xmind今天不做了

# 今日作业

## 使用面向对象的方式完成选项卡的删除功能

1. 1. 点击 x 可以删除当前的li选项卡和当前的section
2. 2. x是没有索引号的, 但是它的父亲li 有索引号, 这个索引号正是我们想要的索引号
3. 3. 所以核心思路是: 点击 x 号可以删除这个索引号对应的 li 和 section
4. 4. 但是, 当我们动态删除新的li和索引号时, 也需要重新获取 x 这个元素 需要调用init方法
5. 5. remove()方法可以直接删除指定元素, 就不需要使用我们之前学的removeChild方法了
6. 6. 当我们删除的不是选中状态的li的时候, 原来的选中状态li保保存不变
7. 7. 当我们删除了选中状态的这个li的时候, 让它的前一个li处于选定状态

## 使用面向对象的方式完成选项卡的编辑功能

1. 1. 双击选项卡li或者 section里面的文字, 可以实现修改功能
2. 2. 双击事件是: ondblclick
3. 3. 如果双击文字, 会默认选定文字, 此时需要双击禁止选中文字
4. 4. 禁止选中文字方法有两种
5. 第一种: js解决方法
6. window.getSelection ? window.getSelection().removeAllRanges() : document.selection.empty();
- 7.
8. 第二种: css解决方法 给body标签设置以下样式
9. -webkit-touch-callout: none;
10. -webkit-user-select: none;
11. -khtml-user-select: none;
12. -moz-user-select: none;
13. -ms-user-select: none;
14. user-select: none;
- 15.



- 16 5. 核心思路: 双击文字的时候, 在 里面生成一个文本框, 当失去焦点或者按下回车然后把文本框输入的值给原先
- 17 元素即可。
- 18 6. 文本框对象.`select()` 可以文本框里面的文字处于选定状态
- 19 7. 当我们文本框失去焦点的时候, 可以通过span元素的innerHTML得到文本框里面的值
- 20 8. 按下回车也可以把文本框里面的值给span
- 21 9. 每个section也要实现双击可以编辑的功能

## 完成上课讲的案例,剩下根据商品名称筛选功能

- 1 获取用户输入的商品名称, 为查询按钮绑定点击事件, 将输入的商品名称与这个数据进行筛选
- 2 如果查询数组中唯一的元素, 用some方法更合适, 因为它找到这个元素, 就不再进行循环, 效率更高

按照价格查询:  -   按照商品名称查询:

id	产品名称	价格
1	小米	3999
2	oppo	999
3	荣耀	1299
4	华为	1999

## 完成当输入框输入文字进行实时匹配(模糊查询)

按照商品名称查询:

id	产品名称	价格
1	小米9	3999
2	oppo	999
3	荣耀V20	1699
4	华为	1999
5	华为nova	1999
6	华为P30	2999
7	华为Mate	5999
8	小米10	4999
9	荣耀V30	3599

