目录

作业讲评

```html
<!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>
        // 一个父类中有money(钱) cars(车) house(房) company(公司) 属性,有管理(manage)的方法,子类要继承父类中的属性和方法


        class Father{
            constructor(money,cars,house,company){
                this.money   = money;
                this.cars    = cars;
                this.house   = house;
                this.company = company;
            }
            manage(){
                console.log( "可以管理" + this.company + "公司");
            }
        }


        class Son extends Father{
            constructor(money,cars,house,company){
                super(money,cars,house,company)
            }
        }
        var son1 = new Son("500亿","劳斯莱斯","海景房","高大尚");
        console.log( son1 );
        son1.manage();
    </script>
</body>
</html>
```

学习目标

- 能够掌握继承中属性或者方法查找原则
- 能够掌握类中需要注意的三个点
- 完成案例tab切换高级版

继承中属性或者方法查找原则:就近原则

1. 继承中,如果实例化子类调用一个方法,先看子类有没有这个方法,如果有就先执行子类的

2. 继承中,如果子类里面没有,就去查找父类有没有这个方法,如果有,就执行父类的这个方法(就近原则)

3.**如果子类想要继承父类的方法,同时在自己内部扩展自己的方法;**子类可以有自己独特的属性,利**用super 调用父类的构造函数,但是super 必须在子类this之前调用**

举例1: 父类和子类中存在同名方法

```
<!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>
        class Father{
            // 如果不需要对属性进行初始化赋值操作,可以不写constructor构造器


            // 父类有一个say方法
            say(){
                console.log("我是小头爸爸");
            }
        }


        class Son extends Father{
            // 子类中也有一个say方法
            say(){
                console.log("我是大头儿子");
            }
        }


        var son = new Son();
        console.log( son )
        // 继承中,如果实例化子类调用一个方法,先看子类有没有这个方法,如果有就先执行子类的
        son.say();
    </script>
</body>
</html>
```

举例2: super关键字调用父类的普通函数　　super.方法名()

```
<!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>
        class Father{
            say(){
                return "我是小头爸爸";
            }
        }


        class Son extends Father{
            say(){
                // 如果想在子类方法中调用父类的方法,就需要使用super关键字
                // super.say();
                // console.log( super.say() );


                console.log(super.say() + "的大头儿子");
            }
        }


        var son = new Son();
        console.log( son )
        son.say();
    </script>
</body>
</html>
```

效果如下:

举例3:如果子类想要继承父类的方法,同时在自己内部扩展自己的方法;子类也可以有自己独特的属性;不过利用super 调用父类的构造函数,但是super 必须在子类this之前调用

```html
<!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>
        class Father{
            constructor(x,y){
                this.x = x;
                this.y = y;
            }

            getSum(){
                console.log("两个数相加得到" + (this.x + this.y) );
            }
        }

        class Son extends Father{
            // 如果需要初始化属性值,就必须写在constructor构造器里面

            // 子类添加独有的属性
            constructor(x,y,z){
                // 如果子类中有构造器,必须通过super调用一次父类的构造器,否则会报错
                super(x,y);

                // 如果子类想要继承父类的方法,同时在自己内部扩展自己的方法;子类也可以有自己独特的属性;不过利用super 调用父类的构造函数,但是"super 必须在子类this之前调用"
                this.z = z;
            }

            // 子类添加独有的方法
            subtract(){
                console.log("三个数相减得到" + (this.x - this.y - this.z) );
            }
        }

        var son = new Son(10,20,80);
        console.log( son );
        son.getSum();
        son.subtract();
    </script>
</body>
</html>
```

运行效果如下:

注意: 子类在构造函数中使用super, 必须放到 this 前面 (必须先调用父类的构造方法,在使用子类构造方法)

三个注意点

1. 在 ES6 中类没有变量提升，所以必须先定义类，才能通过类实例化对象
2. 类里面的共有属性和方法一定要加this使用

3.类里面的this指向问题 重点难点

- constructor中的this指向的是new出来的实例对象
- 自定义的方法,一般也指向的new出来的实例对象

- 绑定事件之后this指向的就是触发事件的事件源

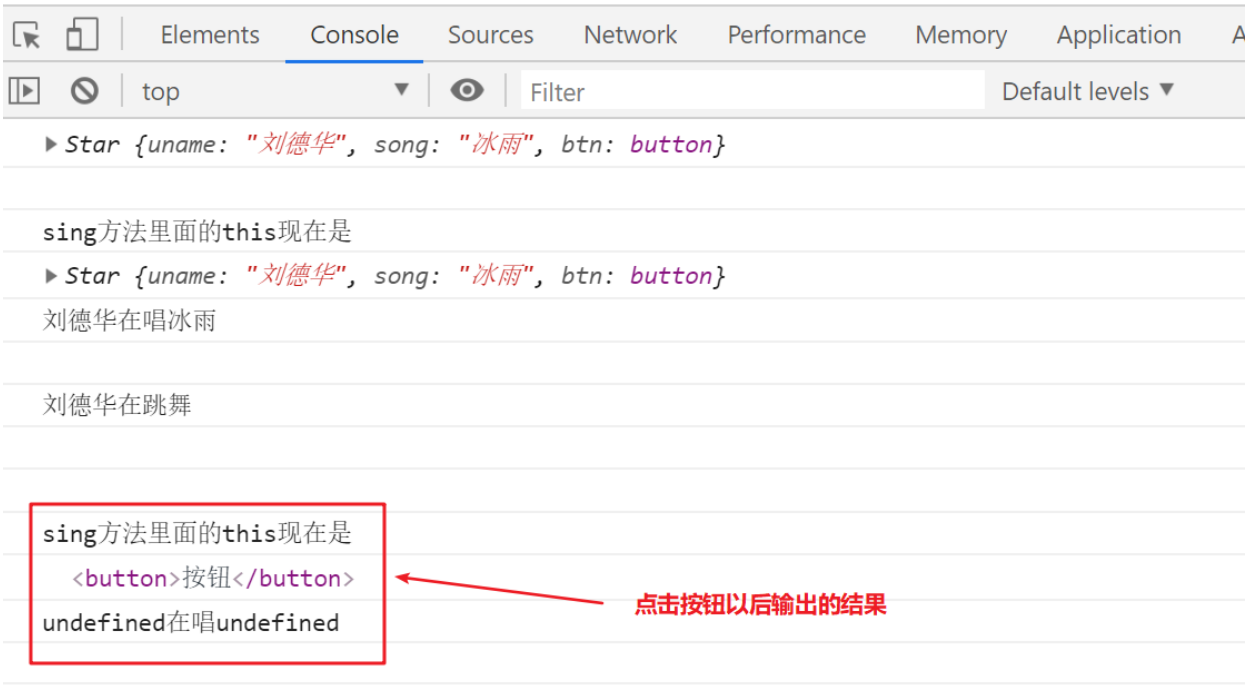
举例1:在 ES6 中类没有变量提升，所以必须先定义类，才能通过类实例化对象

```
<!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>
        // 构造函数也是函数中的一种,因为js有预解析  所以在构造函数之前实例化对象也可以正常使用


        /* // 实例化对象
        var car = new Cars("特斯拉");
        // 调用对象里面的方法
        car.run();


        function Cars(brand){
            this.brand = brand;
            this.run = function(){
                console.log( this.brand + "正在行驶");
            }
        } */



        // 在 ES6 中类没有变量提升，所以必须先定义类，才能通过类实例化对象

        // 在定义类之前实例化,会报错
        // var car = new Cars("比亚迪");
        // car.run();


        class Cars{
            constructor(brand){
                this.brand = brand;
            }


            run(){
                console.log( this.brand + "正在行驶");
            }
        }


        // 必须先定义类,再实例化对象
        var car = new Cars("比亚迪");
        car.run();
    </script>
</body>
</html>
```

举例2: 点击按钮调用 对象里面的方法　　对象里面的方法输出对象里面的某个属性

```
<!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>
    <button>按钮</button>


    <script>
        class Star{
            // 类里面的共有属性和方法"一定"要加this使用
            constructor(uname,song){
                this.uname = uname;
                this.song = song;


                // 想使用面向对象的方式,实现点击按钮调用类中的sing()方法

                // 获取按钮对象作为类中一个属性
                this.btn = document.querySelector("button");
                // 事件处理函数以前咱们都是写一个匿名,现在可以使用类中  方法名,但是不要加括号
                // this.sing这个方法虽然赋值给了按钮的onclick事件,但是会马上调用吗?不会,点击按钮以后才会调用,但是点击按钮以后,this又指向了按钮,所以点击按钮以后,sing方法里面的this就变成了button对象
                this.btn.onclick = this.sing;
            }
            sing(){
                console.log("");
                console.log("sing方法里面的this现在是");
                console.log( this );


                // 调用属性需要使用this
                console.log( this.uname +"在唱" + this.song);
                console.log("");


                // console.log("啦啦啦啦,我是快乐是小当家");
            }


            dance(){
                // 调用属性需要使用this
                console.log( this.uname +"在跳舞");
            }


            show(){
                // 调用其他方法也要使用this
                this.sing();
                this.dance();
            }
        }
        var ldh = new Star("刘德华","冰雨");
        console.log( ldh );
        ldh.show();
        console.log("");


    </script>
</body>
</html>
```

按钮



举例3: constructor构造器中输出this， 自定义方法中输出 this 用一个全局变量把this保存起来

```
constructor中的this指向的是new出来的实例对象
类中定义的方法,一般也指向的new出来的实例对象
绑定事件之后this指向的就是触发事件的事件源
```

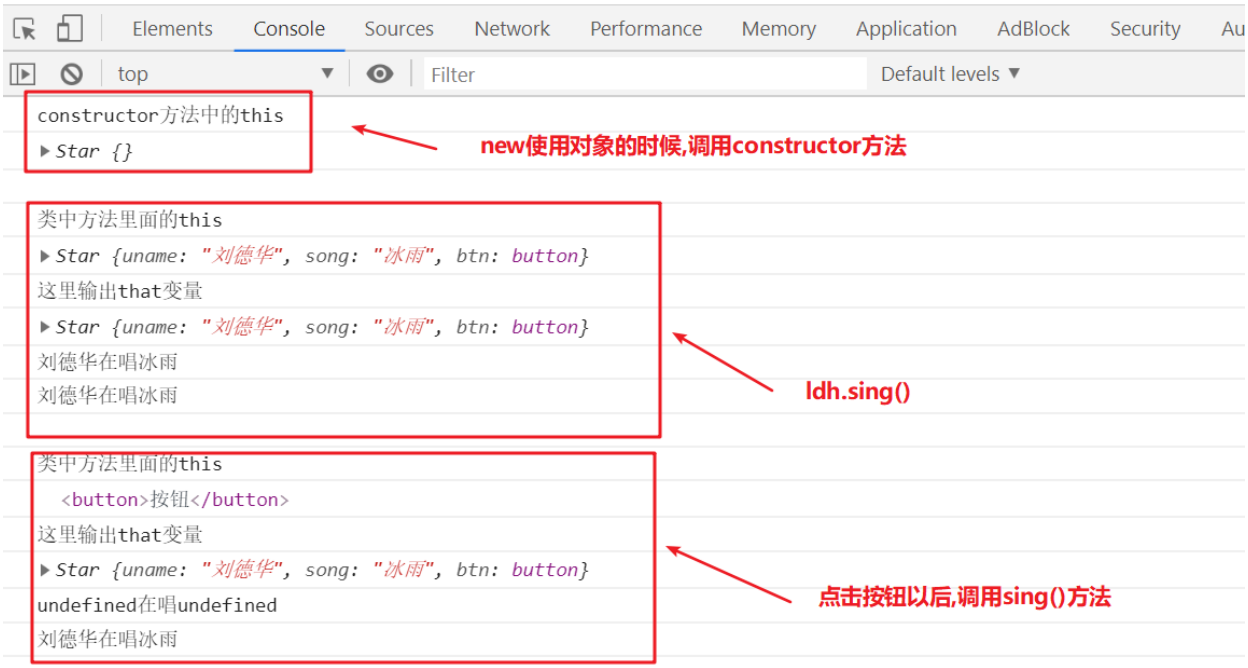
代码:

```
<!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>
    <button>按钮</button>


    <script>
        /*
            this指向问题
                constructor中的this指向的是new出来的实例对象
                类中自定义的方法,一般也指向的new出来的实例对象
```

```
            绑定事件之后this指向的就是触发事件的事件源
        */


        // 解决方法:我们可以把this缓存起来,因为在constructor()里面一直都是指向实例化对象
        // 定义一个变量that  大家喜欢使用这个变量来保存this
        var that = null;

        class Star{
            constructor(uname,song){
                console.log("constructor方法中的this");
                console.log( this );
                console.log("");


                // 缓存this
                that = this;


                this.uname = uname;
                this.song = song;
                this.btn = document.querySelector("button");
                // 当我们点击按钮,按钮去调用sing()方法,所以sing()的this指向按钮
                this.btn.onclick = this.sing;
            }
            sing(){
                console.log("类中方法里面的this");
                console.log( this );
                console.log("这里输出that变量");
                console.log( that );
                console.log( this.uname +"在唱" + this.song);


                // 改成如下,就可以正常访问属性了
                console.log( that.uname +"在唱" + that.song);
                console.log("");
            }
        }
        var ldh = new Star("刘德华","冰雨");
        ldh.sing();
    </script>
</body>
</html>
```

效果图:

按钮



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

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>
        // 功能1 点击 tab栏,可以切换效果


        // 获取tab栏对应的3个li对象
        var lis = document.querySelectorAll("#tab .fisrstnav ul li");
        // 获取所有section对象
        var sections = document.querySelectorAll(".tabscon section");

        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
            lis[i].onclick = function(){
                // 排他操作,清空所有li和所有section标签的类名
                for(var j=0;j<lis.length;j++){
                    lis[j].removeAttribute("class");
                    sections[j].removeAttribute("class");
                }


                // 单独设置当前被点击的li的类名
                this.className = "liactive";

                // 单独设置当前被点击的li对应的section显示
                var index = this.getAttribute("data-index");
                sections[index].className = "conactive";
            }
        }
    </script>
```

添加

1. 点击 + 可以实现添加新的选项卡和内容
2. 第一步: 创建新的选项卡li 和 新的 内容 section
3. 第二步: 把创建的两个元素追加到对应的父元素中
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. 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. 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. insertAdjacentHTML(追加的位置,"要追加的字符串元素")
9. 追加的位置有: beforeend 插入元素内部的最后一个子节点之后

```
9. 该方法地址: https://developer.mozilla.org/zh-CN/docs/Web/API/Element/insertAdjacentHTML
```

```
为添加按钮+ 绑定点击事件
实现标题与内容的添加,做好排他处理
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
```

初步代码:

```
// 功能2: 点击 + 号, 可以添加 tab 项和内容项
// 获取+号对象
var tabadd = document.querySelector(".tabadd");
// 获取li的父元素ul
var ul = document.querySelector(".fisrstnav ul");
// 获取section的父元素
var tabscon = document.querySelector(".tabscon");


tabadd.onclick = function(){
    // 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 追加到父元素里面


    /* // 添加新的选项卡
    var li = document.createElement("li");
    // 设置li的类名
    li.className = "liactive";
    li.innerHTML = '<span>新选项卡</span><span class="iconfont icon-guanbi"></span>';
    ul.appendChild( li );


    // 添加新的内容层
    var section = document.createElement("section");
    section.className = "conactive";
    section.innerHTML = "测试 "+Math.random();
    tabscon.appendChild( section ); */



    // 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
    // appendChild 不支持追加字符串的子元素, insertAdjacentHTML 支持追加字符串的元素
    // insertAdjacentHTML(追加的位置,"要追加的字符串元素")
    // 追加的位置有: beforeend 插入元素内部的最后一个子节点之后
    // 该方法地址: https://developer.mozilla.org/zh-CN/docs/Web/API/Element/insertAdjacentHTML


    // 创建一个HTML元素的字符串
    var li = '<li class="liactive"><span>新选项卡</span><span class="iconfont icon-guanbi"></span></li>';
    // 使用insertAdjacentHTML把HTML字符串当做节点添加到指定父元素中
    ul.insertAdjacentHTML("beforeend",li);


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

进阶代码: 把每个功能封装到各个函数中

```
<script>
    // 功能1 点击 tab栏,可以切换效果
    // 获取tab栏对应的3个li对象
    var lis = document.querySelectorAll("#tab .fisrstnav ul li");
    // 获取所有section对象
    var sections = document.querySelectorAll(".tabscon section");


    // 功能2: 点击 + 号, 可以添加 tab 项和内容项
    // 获取+号对象
```

```
var tabadd = document.querySelector(".tabadd");
// 获取li的父元素ul
var ul = document.querySelector(".fisrstnav ul");
// 获取section的父元素
var tabscon = document.querySelector(".tabscon");

// 因为再这样写下来,代码越来越多,越来越乱,那么这个时候,可以把一些常用的功能封装成函数
// 比如 清除所有li和section类名的功能
function clearClass(){
    for(var j=0;j<lis.length;j++){
        lis[j].removeAttribute("class");
        sections[j].removeAttribute("class");
    }
}


// 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 .fisrstnav ul li");
    // 获取当前所有section对象
    sections = document.querySelectorAll(".tabscon section");
}


// 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;
    }


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


// 切换选项卡的功能
function toggleTab(){
    clearClass();


    // 单独设置当前被点击的li的类名
    this.className = "liactive";

    // 单独设置当前被点击的li对应的section显示
    var index = this.getAttribute("data-index");
    sections[index].className = "conactive";
}

// 添加选项卡的功能
function addTab(){
    // 先清除所有的li和section标签的类名
    clearClass();


    // 创建一个HTML元素的字符串
    var li = '<li class="liactive"><span>新选项卡</span><span class="iconfont icon-guanbi"></span></li>';
    // 使用insertAdjacentHTML把HTML字符串当做节点添加到指定父元素中
    ul.insertAdjacentHTML("beforeend",li);


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


    // 调用初始化函数,实现各种元素的绑定事件
    init();
}
```

```
    }

    // 调用初始函数,给整个页面各种元素绑定事件
    init();
</script>
```

删除功能(还没讲)

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

编辑功能(还没讲)

1. 双击选项卡li或者 section里面的文字,可以实现修改功能
2. 双击事件是：ondblclick
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. 禁止选中文字方法有两种

第一种:js解决方法

```
window.getSelection ? window.getSelection().removeAllRanges() : document.selection.empty();
```

第二种:css解决方法 给body标签设置以下样式

```
        -webkit-touch-callout: none;
         -webkit-user-select: none;
         -khtml-user-select: none;
         -moz-user-select: none;
         -ms-user-select: none;
         user-select: none;
```

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

元素即可.

6. 文本框对象.select() 可以文本框里面的文字处于选定状态
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
8. 按下回车也可以把文本框里面的值给span
9. 每个section也要实现双击可以编辑的功能

作业

1. xmind总结一下新知识点
2. 把tab栏切换封装成函数
3. 继续完成剩下的删除功能以及修改功能
4. 选做-如果完成了面向过程的同学,可以尝试使用面向对象来做tab切换
5. 背面试题