目录

今日作业

css问题-字体设置0可以去掉行内元素的里里面的间隙

```
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        /* 想去掉行内元素左右缝隙,可以给行内元素的"父元素"设置字体为0 */
        .box{
            font-size:0;
        }
        /* 因为父元素设置了字体大小0,有可能会让子元素的内容看不见,这个时候,可以单独给子元素设置字体大小 */
        .box span{
            font-size:16px;
        }
    </style>
</head>
<body>
    <div class="box">
        <span style="background: red">span1</span>
        <span style="background: blue">span2</span>
        <span style="background: green">span3</span>
    </div>
</body>
```

css问题-span里面放文字放图片

```
<!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>
        *{
            margin: 0;
            padding: 0;
        }
        .bigBox{
            width: 1000px;
            height: 500px;
            background: orange;
            margin: 100px auto;
            /* 爷爷元素相对定位 */
            position: relative;
        }
        .bigBox .box{
            width: 800px;
            height: 400px;
            background:pink;
            margin:100px auto;
            /* 父元素相对定位 */
            position: relative;
        }
        .bigBox .box span{
            background: skyblue;
            /* 子元素绝对定位 */
            position: absolute;
            left: 0;
            top: 0;
            /* 元素如果绝对定位脱标以后,默认宽度由内容决定,但是设置百分比的时候,是相对父辈中最近的那个非静态定位元素;如果父辈中都没有非静态定位元素,就会相对body设置宽度 */
            width:50%;
            /* 默认行内元素设置宽度高度无效,默认行内元素的宽度由内容决定 */
            /* width:50%; */
        }
    </style>
</head>
<body>
    <div class="bigBox">
        <div class="box">
            <span>我是span的文字文字文字<img src="作业讲评-弹幕/images/1.gif"/></span>
        </div>
    </div>
</body>
</html>
```

案例-弹幕效果

分析:

1. 可以在文本框中输入内容,按回车发送内容　　键盘事件对象 判断用户是否按了回车键
2. 可以在文本框中输入内容,点击发送按钮发送内容　　给按钮绑定点击事件
3. 核心思路,把发送的内容放在span标签中,把span添加到指定盒子中　　appendChild
4. 弹幕内容是背景图的右边开始向左边滚动　　利用我们之前封装的动画函数
5. 弹幕内容移动到背景图最左边以后,需要移除span元素　removeChild
6. 弹幕内容垂直方向是随机的位置的　　随机函数Math.random()
7. 点击表情按钮显示/隐藏表情面板　display:block或者none
8. 可以选择某个标签,就会出现在输入框中,但是输入框不是图片,而是[em_索引号]
9. 显示的时候,需要把对应[em_索引号]内容转成img图片标签

HTML+CSS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>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        img{
            vertical-align: bottom;
```

```
}
input{
    outline: none;
    border:none;
}
li{
    list-style:none;
}
button{
    border:none;
}
.box{
    width: 800px;
    height: 550px;
    margin: 20px auto;
}
.box .videobox {
    height: 500px;
    position: relative;
}
/* 子元素选择器,选择子元素 */
.box>.videobox>img{
    width: 100%;
    height: 100%;
}
.box .videobox .danmu{
    position: absolute;
    left: 800px;
    top: 0;
    color:red;
    background: skyblue;
    /* 强制文字一行显示,不换行 */
    white-space: nowrap;
}
.box .send{
    height: 40px;
    background: white;
    padding-top: 10px;
    /* 盒子阴影 */
    box-shadow: 0px 0px 7px 0px rgba(0,0,0,0.3);
    position: relative;
}
.box .send input{
    width: 700px;
    height: 28px;
    margin-left: 20px;
    float: left;
    background: #f4f4f4;
    border:1px solid #e7e7e7;
    text-indent: 13px;
}
.box .send .btn{
    width: 60px;
    height: 30px;
    color:white;
    background:#00a1d6;
    text-align: center;
    line-height: 30px;
    float: left;
    cursor: pointer;
}
.box .send .btn:hover{
    background: #00b5e5;
}
.box .send .biaoqing>img{
    width: 22px;
    height: 22px;
    position: absolute;
    right: 86px;
    top: 50%;
    margin-top: -11px;
    cursor: pointer;
}
.box .send .biaoqing ul{
    width: 394px;
```

```
            height: 124px;
            background: white;
            position: absolute;
            right: 0;
            bottom: 40px;
            border:1px solid #cccccc;
            /* 隐藏表情列表 */
            display: none;
        }
        .box .send .biaoqing ul li{
            float: left;
            margin-right: 2px;
        }
    </style>
</head>
<body>
    <div class="box">
        <!-- 上部分 -->
        <div class="videobox">
            <!-- 背景图 -->
            <img src="images/bg.jpg"/>
            <!-- 弹幕内容 -->
            <span class="danmu">我是弹<img src="images/28.gif"/>幕的<img src="images/1.gif"/>内容</span>
        </div>
        <!-- 下部分 -->
        <div class="send">
            <!-- 文本框 -->
            <input type="text" placeholder="发个弹幕见证当下">
            <!-- 表情部分 -->
            <div class="biaoqing">
                <!-- 小笑脸 -->
                <img src="images/biaoqing.png"/>

                <!-- 表情列表 -->
                <ul>
                    <li><img src="images/1.gif"></li>
                    <li><img src="images/2.gif"></li>
                    <li><img src="images/3.gif"></li>
                    <li><img src="images/4.gif"></li>
                    <li><img src="images/5.gif"></li>
                    <li><img src="images/6.gif"></li>
                    <li><img src="images/7.gif"></li>
                    <li><img src="images/8.gif"></li>
                    <li><img src="images/9.gif"></li>
                    <li><img src="images/10.gif"></li>
                    <li><img src="images/11.gif"></li>
                    <li><img src="images/12.gif"></li>
                    <li><img src="images/13.gif"></li>
                    <li><img src="images/14.gif"></li>
                    <li><img src="images/15.gif"></li>
                    <li><img src="images/16.gif"></li>
                    <li><img src="images/17.gif"></li>
                    <li><img src="images/18.gif"></li>
                    <li><img src="images/19.gif"></li>
                    <li><img src="images/20.gif"></li>
                    <li><img src="images/21.gif"></li>
                    <li><img src="images/22.gif"></li>
                    <li><img src="images/23.gif"></li>
                    <li><img src="images/24.gif"></li>
                    <li><img src="images/25.gif"></li>
                    <li><img src="images/26.gif"></li>
                    <li><img src="images/27.gif"></li>
                    <li><img src="images/28.gif"></li>
                    <li><img src="images/29.gif"></li>
                    <li><img src="images/30.gif"></li>
                    <li><img src="images/31.gif"></li>
                    <li><img src="images/32.gif"></li>
                    <li><img src="images/33.gif"></li>
                    <li><img src="images/34.gif"></li>
                    <li><img src="images/35.gif"></li>
                    <li><img src="images/36.gif"></li>
                    <li><img src="images/37.gif"></li>
                    <li><img src="images/38.gif"></li>
                    <li><img src="images/39.gif"></li>
```

```
                <li><img src="images/40.gif"></li>
                <li><img src="images/41.gif"></li>
                <li><img src="images/42.gif"></li>
                <li><img src="images/43.gif"></li>
                <li><img src="images/44.gif"></li>
                <li><img src="images/45.gif"></li>
                <li><img src="images/46.gif"></li>
                <li><img src="images/47.gif"></li>
                <li><img src="images/48.gif"></li>
                <li><img src="images/49.gif"></li>
                <li><img src="images/50.gif"></li>
                <li><img src="images/51.gif"></li>
                <li><img src="images/52.gif"></li>
                <li><img src="images/53.gif"></li>
                <li><img src="images/54.gif"></li>
                <li><img src="images/55.gif"></li>
                <li><img src="images/56.gif"></li>
                <li><img src="images/57.gif"></li>
                <li><img src="images/58.gif"></li>
                <li><img src="images/59.gif"></li>
                <li><img src="images/60.gif"></li>
                <li><img src="images/61.gif"></li>
                <li><img src="images/62.gif"></li>
                <li><img src="images/63.gif"></li>
                <li><img src="images/64.gif"></li>
                <li><img src="images/65.gif"></li>
                <li><img src="images/66.gif"></li>
                <li><img src="images/67.gif"></li>
                <li><img src="images/68.gif"></li>
                <li><img src="images/69.gif"></li>
                <li><img src="images/70.gif"></li>
                <li><img src="images/71.gif"></li>
                <li><img src="images/72.gif"></li>
                <li><img src="images/73.gif"></li>
                <li><img src="images/74.gif"></li>
                <li><img src="images/75.gif"></li>
            </ul>
        </div>
        <!-- 发送按钮 -->
        <button class="btn">发送</button>
    </div>
</div>
</body>
</html>
```

完整代码

```
// 封装动画函数
// obj是需要做动画的对象
// target是需要移动到哪里,也就是目标值
// callback回调函数,做完动画以后执行的函数  可选参数
function animate(obj,target,callback){
    // 先清除原有的定时器
    clearInterval( obj.timer );

    // 开启定时器
    obj.timer = setInterval(function(){
        // 利用公式得到缓慢动画的步长值
        var step = (target - obj.offsetLeft) / 10;
        // 如果步长值大于0,则向上取整;小于0,则向下取整
        step = step > 0 ? Math.ceil( step) : Math.floor( step );
        // 判断是否达到目标值
        if( obj.offsetLeft == target ){
            // 清除定时器
            clearInterval( obj.timer );
```

```
            // 清除定时器以后就代表动画执行完毕,调用回调函数
            if( callback ){
                callback();
            }
        }else{
            obj.style.left = obj.offsetLeft + step + "px"
        }
    },15)
}


// 匀速运动函数
function animate_linear(obj,target,callback){
    // 先清除原有的定时器
    clearInterval( obj.timer );

    // 开启定时器
    obj.timer = setInterval(function(){
        // 利用公式得到缓慢动画的步长值
        // var step = (target - obj.offsetLeft) / 10;
        // 如果步长值大于0,则向上取整;小于0,则向下取整
        // step = step > 0 ? Math.ceil( step) : Math.floor( step );


        // 判断目标值跟当前值的大小比较    决定步长值为正值还是负值
        var step = obj.offsetLeft > target ? -1 : 1;


        // 判断是否达到目标值
        if( obj.offsetLeft == target ){
            // 清除定时器
            clearInterval( obj.timer );
            // 清除定时器以后就代表动画执行完毕,调用回调函数
            if( callback ){
                callback();
            }
        }else{
            obj.style.left = obj.offsetLeft + step + "px"
        }
    },15)
}
```

```
<!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>
        *{
            margin: 0;
            padding: 0;
        }
        img{
            vertical-align: bottom;
        }
        input{
            outline: none;
            border:none;
        }
        li{
```

```
        li{
            list-style:none;
        }
        button{
            border:none;
        }
        .box{
            width: 800px;
            height: 550px;
            margin: 20px auto;
        }
        .box .videobox {
            height: 500px;
            position: relative;
            overflow:hidden;
        }
        /* 子元素选择器,选择子元素 */
        .box>.videobox>img{
            width: 100%;
            height: 100%;
        }
        .box .videobox .danmu{
            position: absolute;
            left: 800px;
            top: 0;
            color:red;
            /* background: skyblue; */
            /* 强制文字一行显示,不换行 */
            white-space: nowrap;
        }
        .box .send{
            height: 40px;
            background: white;
            padding-top: 10px;
            /* 盒子阴影 */
            box-shadow: 0px 0px 7px 0px rgba(0,0,0,0.3);
            position: relative;
        }
        .box .send input{
            width: 700px;
            height: 28px;
            margin-left: 20px;
            float: left;
            background: #f4f4f4;
            border:1px solid #e7e7e7;
            text-indent: 13px;
        }
        .box .send .btn{
            width: 60px;
            height: 30px;
            color:white;
            background:#00a1d6;
            text-align: center;
            line-height: 30px;
            float: left;
            cursor: pointer;
        }
        .box .send .btn:hover{
            background: #00b5e5;
        }
        .box .send .biaoqing>img{
            width: 22px;
            height: 22px;
            position: absolute;
            right: 86px;
            top: 50%;
            margin-top: -11px;
            cursor: pointer;
        }
        .box .send .biaoqing ul{
            width: 394px;
            height: 124px;
            background: white;
            position: absolute;
            right: 0;
            bottom: 40px;
            border:1px solid #cccccc;
            /* 隐藏表情列表 */
            display: none;
        }
        .box .send .biaoqing ul li{
            float: left;
            margin-right: 2px;
        }
    </style>
</head>
<body>
    <div class="box">
        <!-- 上部分 -->
        <div class="videobox">
            <!-- 背景图 -->
            <img src="images/bg.jpg"/>
            <!-- 弹幕内容 -->
            <!-- <span class="danmu">我是弹<img src="images/28.gif"/>幕的<img src="images/1.gif"/>内容</span> -->
        </div>
        <!-- 下部分 -->
        <div class="send">
            <!-- 文本框 -->
            <input type="text" placeholder="发个弹幕见证当下">
            <!-- 表情部分 -->
            <div class="biaoqing">
                <!-- 小笑脸 -->
```

```
            <img src="images/biaoqing.png" class="xiaoliang"/>

            <!-- 表情列表 -->
            <ul>

            </ul>
        </div>
        <!-- 发送按钮 -->
        <button class="btn">发送</button>
    </div>
</div>


<script src="js/animate.js"></script>
<script>
    // 功能1:点击笑脸图片显示表情列表,再点击笑脸图片隐藏表情列表
    // 获取笑脸对象
    var xiaoliang = document.querySelector(".xiaoliang");
    // 获取表情列表
    var biaoqingUl = document.querySelector(".biaoqing ul");
    // 获取文本框对象
    var input = document.querySelector("input");


    // 用一个变量保存现在表情列表是否隐藏  true代表表情列表隐藏了  false代表表情列表没有隐藏
    var flag = true;
    xiaoliang.onclick = function(){
        if( flag ){
            biaoqingUl.style.display = "block";
            flag = false;
        }else{
            biaoqingUl.style.display = "none";
            flag = true;
        }
    }


    // 功能2. 循环显示表情列表中所有表情
    // <li><img src="images/74.gif"></li>
    for(var i=1;i<=75;i++){
        // 创建一个li节点标签
        var li = document.createElement("li");
        li.innerHTML = '<img src="images/'+i+'.gif">';
        // 把li标签追加到表情列表ul中
        biaoqingUl.appendChild( li );
    }


    // 功能3: 点击表情列表中每个表情(其实img表情),就放相应的内容添加到input的value
    // 什么事件委托  给父元素绑定事件  借助了冒泡的原理
    biaoqingUl.onclick = function(e){
        e = e || window.event;
        // 这里的this代表绑定事件的对象
        // console.log( this );
        // e.target可以找到触发事件的对象
        // console.log( e.target );


        // 获取当前被点击表情对应图片的src属性
        var imgSrc = e.target.src;
        // 让字符串根据某个符号分割成数组
        var arr = imgSrc.split("/");
        // 从数组中得到最后一个元素,就是图片名文件包括后缀的
        var fileFullName = arr[arr.length - 1];
        // 找到最后一个.出现的位置
        var pos = fileFullName.lastIndexOf(".");
        // 再从下标0开始截取到最后一个.之前
        // 得到文件名
        var filename =  fileFullName.substring(0,pos) ;
        // 拼接字符串成 [em_1]
        var str = "[em_" + filename + "]";

        // 设置input文本框的内容
        input.value = input.value + str;


        // 隐藏表情列表
        biaoqingUl.style.display = "none";
    }


    // 功能4:点击发送按钮,可以发送input中的内容(其实就是发送弹幕)
    // 获取按钮对象
    var btn = document.querySelector(".btn");
    // 获取.videobox对象
    var videobox = document.querySelector(".videobox");


    btn.onclick = function(){
        // 调用创建弹幕的函数
        creteDanmu();
    }


    // 封装一个把字符串中的[em_1]转成图片的函数
    function replaceBiaoqing(str){
        // 正则表达式过几天学
        var reg = /\[em_([0-9]*)\]/gi;
        return str.replace(reg, "<img src='images/$1.gif'/>");
```

```
        }


        // 功能7:按下回车,发送弹幕,但是我们发现,按下回车发送弹幕跟点击发送按钮发送弹幕的效果是一样的,所以这个时候,我们可以把创建弹幕的代码封装到函数中


        // 创建弹幕的函数
        function creteDanmu(){
            // 功能6: 判断input中内容是否为空
            if( input.value == ""){
                alert("请填写弹幕内容");
                // 让后面的代码不执行
                return false;
            }
            // 创建弹幕,其实就是创建一个span标签设置内容,把span添加到.videobox层中
            var span = document.createElement("span");
            // 设置span类名
            span.className = "danmu";


            // 设置span弹幕内容
            span.innerHTML = replaceBiaoqing( input.value );
            // 追加span到.videobox中
            videobox.appendChild( span );


            // 注意:因为获取span高度的时候,需要页面上有这个元素才可以正确获取到
            // console.log( span.offsetHeight );
            var spanTop = Math.floor( Math.random()*(videobox.offsetHeight-span.offsetHeight+1) );
            span.style.top = spanTop +"px";


            // 功能5: 开始执行匀速动画,让整个弹幕左边之外
            // console.log( "弹幕内容的宽度"+span.offsetWidth);
            animate_linear(span, -span.offsetWidth, function(){
                // 当内容执行完毕动画以后,证明内容已经飞出边界,就不需要这个span标签了,那么可以删除它
                span.parentNode.removeChild( span );
            } );


            // 清除input文本的内容
            input.value = "";
        }


        // 给input绑定按下键盘事件
        input.onkeydown = function(e){
            var e = e || window.event;
            if( e.keyCode == 13 ){
                // 调用创建弹幕的函数
                creteDanmu();
            }
        }
    </script>
</body>
</html>
```

学习目标

- 能够说出什么是面向对象
- 能够说出类和对象的关系
- 能够使用class创建中自定义类
- 能够说出什么是继承

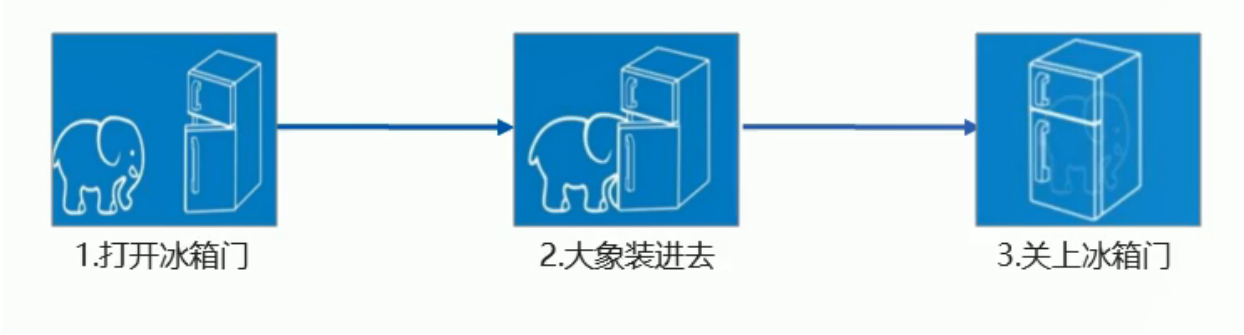
面向过程与面向对象

面向过程

面向过程变成 POP (Process-oriented programming)

面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候再一个一个的依次调用就可以了。

举个例子:将大象装进冰箱,面向过程做法



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>
        // 面向过程
        // 面向过程变成 POP (Process-oriented programming)
        // 面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候再一个一个的依次调用就可以了。
        // 打开冰箱门=> 大象装进去 => 关上冰箱门


        // 把每个步骤封装到函数中
        function Elephant(){
            console.log("大象装进冰箱中");
        }
        function openDoor(){
            console.log("打开冰箱门了~");
        }
        function closeDoor(){
            console.log("关闭冰箱门了~");
        }
        // 按步骤调用函数即可
        openDoor();
        Elephant();
        closeDoor();
    </script>
</body>
</html>
```

小结: 面向过程,就是按照我们分析好了的步骤,按照步骤解决问题.

面向对象

面向对象编程 OOP(Object Oriented Programming)

面向对象是把事务分解成为一个个对象，然后由对象之间分工与合作。

举例:将大象装进冰箱,面向对象做法

```
先找出对象,并写出这些对象的功能:
1. 大象对象
进去

2. 冰箱对象
打开冰箱门
关闭冰箱门

3. 使用大象和冰箱的功能
```

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>
        // 面向对象
        // 面向对象编程 OOP(Object Oriented Programming)
        // 面向对象是把事务分解成为一个个对象，然后由对象之间分工与合作。
        // 也就是一句话,所以的功能都将由对象来完成
        /*  举例:将大象装进冰箱,面向对象做法
        先找出对象,并写出这些对象的功能:
        1. 大象对象
        进去

        2. 冰箱对象
        打开冰箱门
        关闭冰箱门

        3. 使用大象和冰箱的功能 */


        // 创建一个大象的构造函数
        function Elephant( uname ){
            // 大象的名字  属性
            this.uname = uname;
            // 大象的方法
            this.goin = function(){
                console.log(this.uname + "进去冰箱了");
```

```
            }
        }


        // 创建一个冰箱的构造函数
        function Refrigerator( brand ){
            // 冰箱的品牌  属性
            this.brand = brand ;
            // 冰箱的功能  打开冰箱门
            this.openDoor = function(){
                console.log( this.brand +"冰箱打开了冰箱门" );
            }
            // 冰箱的功能  关闭冰箱门
            this.closeDoor = function(){
                console.log( this.brand +"冰箱关闭了冰箱门" );
            }
        }


        // 怎么从构造函数中得到具体的对象？通过new关键字实例化构造函数,得到对象
        // 语法  var obj = new 构造函数名(实参);
        // 为什么要创建对象,因为对象才可以使用构造函数里面的方法和属性
        var daxiang1 = new Elephant("粉色大象");
        var bingxiang1 = new Refrigerator("海尔");
        // 面向对象的做法就是功能让对象来完成
        bingxiang1.openDoor();
        daxiang1.goin();
        bingxiang1.closeDoor();


        console.log("");


        var daxiang2 = new Elephant("天空蓝大象");
        var bingxiang2 = new Refrigerator("美的");
        // 面向对象的做法就是功能让对象来完成
        bingxiang2.openDoor();
        daxiang2.goin();
        bingxiang2.closeDoor();
    </script>
</body>
</html>
```

小结:

1. 面向对象是以对象功能来划分问题,而不是步骤
2. 在面向对象程序开发思想中,每一个对象都是功能中心,具有明确分工
3. 面向对象编程具有灵活、代码可复用、容易维护和开发的优点,更适合多人合作的大型软件项目

面向对象的特性

- 封装性,我们经常把代码封装到函数中
- 继承性,儿子继承爸爸的姓
- 多态性,同一个对象,在不同的时刻,体现出不同的状态

继承：继承自拖拉机，实现了扫地的接口

封装：无需知道如何运作，开动即可

多态：平时扫地，天热当风扇

重用：没用额外动力，重复利用了发动机能量

多线程：多个扫把同时工作

低耦合：扫把可以换成拖把而无须改动

组件编程：每个配件都是可单独利用的工具

适配器模式：无需造发动机，继承自拖拉机，只取动力方法

代码托管：无需管理垃圾，直接扫到路边即可

面向过程与面向对象对比

	面向过程	面向对象
优点	性能比面向对象高，适合跟硬件联系很紧密的东西，例如单片机就采用的面向过程编程。	易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统 更加灵活、更加易于维护
缺点	不易维护、不易复用、不易扩展	性能比面向过程低

用面向过程的方法写出来的程序是一份蛋炒饭，而用面向对象写出来的程序是一份盖浇饭。

ES6 中的类和对象

面向对象

面向对象更贴近我们的实际生活, 可以使用面向对象描述现实世界事物. 但是事物分为具体的事物和抽象的事物

手机 抽象的(泛指的) 一个类别 类

小米手机,具体的(特指的) 对象

面向对象的思维特点:

1. 抽取（抽象）对象共用的属性和行为组织(封装)成一个类(模板)
2. 对类进行实例化, 获取类的对象
3. 面向对象编程我们考虑的是有哪些对象，按照面向对象的思维特点,不断的创建对象,使用对象,指挥对象做事情

对象

现实生活中：万物皆对象，对象是**一个具体的事物**，看得见摸得着的实物。例如，一本书、一辆汽车、一个人可以是“对象”，一个数据库、一张网页、一个与远程服务器的连接也可以是“对象”。

在 JavaScript 中，对象是一组无序的相关属性和方法的集合，所有的事物都是对象，例如字符串、数值、数组、函数等。

对象是由属性和方法组成的：是一个无序键值对的集合,指的是一个具体的事物

- 属性：事物的特征，在对象中用属性来表示（常用名词）
- 方法：事物的行为，在对象中用方法来表示（常用动词）

创建对象

以下代码是对对象的复习

- 字面量{}创建对象
- new Object方式创建对象
- 构造函数方式创建对象

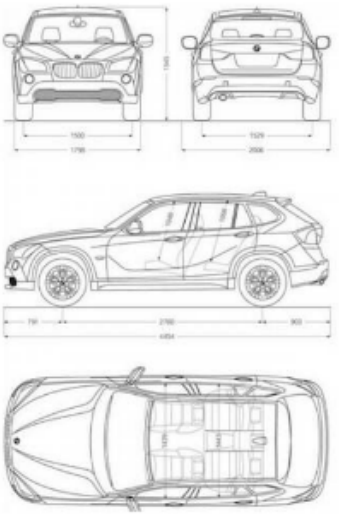
举例:

```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>
        // 使用三种方式创建同学们自己心仪未来对象  想想对象需要有什么属性  对象需要有什么方法  女朋友 girlFriend

        // 字面量{}创建对象
        var girlFriend1 = {};
        girlFriend1.uname = "萌";
        girlFriend1.weight = "168kg";
        girlFriend1.height = "168cm";
        girlFriend1.sing = function(){
            console.log( this.uname + "会唱歌" );
        }
        console.log("身高" + girlFriend1.height );
        girlFriend1.sing();
        console.log("");


        // new Object方式创建对象
        var girlFriend2 = new Object();
        girlFriend2.uname = "钟无艳";
        girlFriend2.play = function(){
            console.log( girlFriend2.uname +"小粉锤锤你噢~");
        }
        console.log( "名字为:" + girlFriend2.uname );
        girlFriend2.play();
        console.log("");

        // 构造函数方式创建对象
        function GirlFriend(uname,age){
            this.uname = uname;
            this.age = age;
            this.dance = function(){
                console.log( "名字"+this.uname+",年龄" +this.age+ "岁在跳舞");
            }
        }
        // 通过new关键字得到实例化
        var girlFriend3 = new GirlFriend("安琪拉",18);
        console.log( girlFriend3.uname );
        girlFriend3.dance();
        console.log("");


        var girlFriend4 = new GirlFriend("甄姬",20);
        console.log( girlFriend4.uname );
        girlFriend4.dance();
    </script>
</body>
</html>
```

类

- 在 ES6 中新增加了类的概念，可以使用 class 关键字声明一个类，之后以这个类来实例化对象。
- 类抽象了对象的公共部分，它**泛指**某一大类（class）
- 对象**特指**某一个，通过类实例化一个具体的对象

汽车设计图纸（构造函数）



靠，一辆真宝马！(对象实例)

面向对象的思维特点:

1. 抽取（抽象）对象共用的属性和行为组织(封装)成一个**类**(模板)
2. 对类进行实例化, 获取类的**对象**

创建类

语法:

```
//步骤1 使用class关键字 类名一般首字母大写
class className {
 // class body 类里面的主体, 类中的内容一般是属性或者方法
}
//步骤2使用定义的类创建实例 注意new关键字
var xx = new className();
```

创建实例

语法:

```
var xx = new className();
```

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>
        /* 创建类


            class 类名{
                类内容
            }
        */
        class Car{


        }


        /* 通过类得到具体的一个对象,通常我们叫实例化对象

            new 类名(参数1,参数2...)
        */
        var car1 = new Car();
        console.log( car1 );


        var car2 = new Car();
        console.log( car2 );
    </script>
</body>
</html>
```

类 constructor 构造函数

constructor() 方法是类的构造函数(默认方法)，**用于传递参数,返回实例对象**，通过 new 命令生成对象实例时，自动调用该方法。如果没有显示定义，类内部会自动给我们创建一个**constructor()**

一般我们会把类的共有属性放到 constructor 里面

语法：

```
class Person {
    constructor(name,age) { // constructor 构造方法或者构造函数
        this.name = name;
        this.age = age;
```

```
  }
}
```

创建实例:

```
var ldh = new Person('刘德华', 18);
console.log(ldh.name)
```

注意:

1. 通过class 关键字创建类, 类名我们还是习惯性定义首字母大写
2. 类里面有个constructor 函数,可以接受传递过来的参数,同时返回实例对象
3. constructor 函数 只要 new 生成实例时,就会自动调用这个函数, 如果我们不写这个函数,类也会自动生成这个函数
4. 多个函数方法之间不需要添加逗号分隔
5. 生成实例 new 不能省略
6. 语法规范, 创建类 类名后面不要加小括号,生成实例 类名后面加小括号, 构造函数不需要加function

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>
        // 注意:
        // 通过class 关键字创建类, 类名我们还是习惯性定义首字母大写
        // 类里面有个constructor 函数,可以接受传递过来的参数,同时返回实例对象
        // constructor 函数 只要 new 生成实例时,就会自动调用这个函数, 如果我们不写这个函数,类也会自动生成这个函数
        // 多个函数方法之间不需要添加逗号分隔
        // 生成实例 new 不能省略
        // 语法规范, 创建类 类名后面不要加小括号,生成实例 类名后面加小括号, 构造函数不需要加function


        // 创建一个类
        class Car{
            // 类 constructor 构造函数  它是一个内置方法来的,  "new实例化"以后就会"自动"调用这个constructor方法
            // constructor 构造函数 主要用于"初始化对象属性的值",对象公共的特性我们就会放在属性里面
            /* constructor(参数1,参数2...){
                this.属性名1 = 参数1;
                this.属性名2 = 参数2;
                ...
            } */

            // 类中添加属性和方法
            // 1. 类中添加属性放到 constructor 里面
            constructor(carName,carWeight){
                // 接收实例化传递过来的参数,给属性赋值
                this.carName   = carName;
                this.carWeight = carWeight;
            }


            // 2. 类中添加方法是通过在类中定义函数实现的
            // 3. 但是要注意,我们类里面所有的函数不需要写"function关键字"
            // 4. 并且多个函数方法之间"不需要添加逗号"分隔
            run(){
                console.log( this.carName + "正在行驶中...");
                console.log("");
            }


            backCar(){
                console.log( this.carName + "倒车请注意,倒车请注意...");
                console.log("");
            }
        }
        // 通过new关键字实例化得到对象
        var car1 = new Car("宝马","2000kg");
        console.log( car1 );
        car1.run();

        var car2 = new Car("奔驰","1000kg");
        console.log( car2 );
        car2.run();
    </script>
</body>
</html>
```

通过结果我们可以看出,运行结果和使用构造函数方式一样

类中添加属性和方法

语法:

```
1. 类中添加属性放到 constructor 里面
2. 类中添加方法是通过在类中定义函数实现的
3. 但是要注意,我们类里面所有的函数不需要写function关键字
4. 并且多个函数方法之间"不需要添加逗号"分隔
```

举例:

```
<!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>
        <script type="text/javascript">
            // 1. 创建类 class   创建一个明星类
             class Star{
                 // 类的共有属性放到 constructor 里面
                 constructor(uname,age){
                     this.uname = uname;
                     this.age   = age;
                 }
                 sing(song){
                     console.log(this.uname+"唱"+song);
                 }
             }

             // 2. 利用类创建对象 new
             var ldh = new Star("刘德华",18);
             var zxy = new Star("张学友",20);
             console.log(ldh);
             console.log(zxy);


             // 注意:
             // 我们类里面所有的函数不需要写function关键字
             // 多个函数方法之间不需要添加逗号分隔
             ldh.sing("冰雨");
```

```
            zxy.sing("吻别");
        </script>
    </head>
    <body>

    </body>
</html>
```

类的继承

现实中的继承：子承父业，比如我们都继承了父亲的姓。

程序中的继承：子类可以继承父类的一些属性和方法。

语法： 通过extends关键字

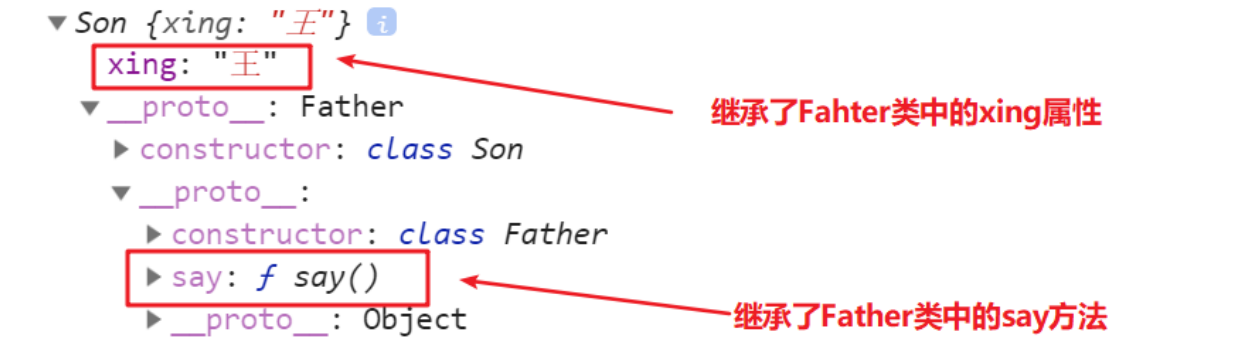
```
// 父类
class Father{
}

// 子类继承父类
class  Son  extends Father {
}
```

举例: 定义一个父类有姓属性,有一个say的方法可以说明自己的姓是什么,再定义一个子类继承父类,实例化子类

代码如下:

```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>
        // 类的继承
        // 现实中的继承：子承父业，比如我们都继承了父亲的姓。
        // 程序中的继承：子类可以继承父类的一些属性和方法。


        /*
            语法：  通过extends关键字
            // 父类
            class Father{


            }

            // 子类继承父类
            class  Son  extends Father {


            }
        */


        // 举例：定义一个父类有姓属性,有一个say的方法可以说明自己的姓是什么,再定义一个子类继承父类,实例化子类

        // 创建父类
        class Father{
            // 给类名添加属性 使用 constructor 构造函数
            constructor(xing){
                this.xing = xing;
            }


            say(){
                console.log("姓" + this.xing);
            }
        }


        // 创建子类
        class Son extends Father{


        }


        // 因为Son类继承了Father类,所以Son中有Father类定义的属性和方法
        var son1 = new Son("王");
        console.log( son1 );
        son1.say();
    </script>
</body>
</html>
```

效果如下:



姓王

super 关键字

super 关键字用于访问和调用对象父类上的函数。可以调用父类的构造函数，也可以调用父类的普通函数

举例:以下代码中,子类调用自己的构造方法,会报错,因为父类里面的函数必须使用父类中constructor的属性才行

```
<!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>
        // 创建父类
        class Father{
            constructor(x,y){
                this.x = x;
                this.y = y;
            }


            sum(){
                console.log( this.x + this.y );
            }
        }


        // 创建子类
        class Son extends Father{
            // 如果子类有继承父类的话,子类调用自己的构造方法,"会报错",因为父类里面的函数必须使用父类中constructor的属性才行
            constructor(x,y){
                this.x = x;
                this.y = y;
            }
        }


        var son1 = new Son(5,8);
        console.log( son1 );
        son1.sum();
    </script>
</body>
</html>
```

```
Uncaught ReferenceError: Must call super constructor in derived class before accessing 'this' or returning from derived constructor    09-super关键字.html:26
    at new Son (09-super关键字.html:26)
    at 09-super关键字.html:31
```

举例:通过super关键字调用父类中的构造方法,这样子类就可以向父类中传递参数了

```
Son {x: 5, y: 8}
13
```

```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>
        // 创建父类
        class Father{
            constructor(x,y){
                this.x = x;
                this.y = y;
            }

            sum(){
                console.log( this.x + this.y );
            }
        }

        // 创建子类
        class Son extends Father{
            // 如果子类有继承父类的话,子类调用自己的构造方法,"会报错",因为父类里面的函数必须使用父类中constructor的属性才行
            /* constructor(x,y){
                this.x = x;
                this.y = y;
            } */

            // 如果子类的构造函数跟父类完全一样的时候,可以不写构造函数
            constructor(x,y){
                // 调用父类中的构造方法,这样子类就可以像父类中传递参数了 super是超级的意思
                super(x,y);
            }
        }

        var son1 = new Son(5,8);
        console.log( son1 );
        son1.sum();
    </script>
</body>
</html>
```

今日总结

xmind你懂的

今日作业

请看作业文件夹