day19 面向对象

面向对象概述（oop）

面向对象是一种编程思维（oop），将对应的你需要用到的**对象进行提取**，将**对应方法综合到对应的对象**中，在需要调用对应的方法的时候去找对应的对象。（**万物皆对象**（任何东西都可以抽取为对象），主要的过程就是**找对应的对象做对应的事情。**）

示例

相亲 （面向过程）过程中会产生很多的行为（行为比较分散）

- 找媒婆 (介绍人)
- 提需求 （你）
- 匹配人 （媒婆）
- 见面
- 相互了解
- ....

面向对象（主要是找对应的对象 去做对应的事情（忽略过程））

- 媒婆
- 我
- 相亲对象

面向对象的核心就是找对象

面向对象的三大特性

- 封装 （将对应的属性和方法抽取封装到对应的类（构造函数）中）
- 继承 （子类继承父类 子类拥有父类非私有属性及方法）
- 多态 （一个东西多种形态体（基于继承的） 重载（一个类多个函数重名（js中不允许的）） 重写（子类重写父类方法））

对象创建声明的方式

使用new关键词来声明 (实际调用的都是构造函数)

- **es6新增的class （其实调用也是构造函数）**

```
class Person{
    //类的构造器
    constructor(name){
        this.name = name
    }
}
//调用类中的构造器
let person = new Person('jack')
```

- **es3自带的构造函数**

```
function Person(name){
    this.name = name
}
let person = new Person('tom')
```

构造函数和class的区别

- 构造函数的兼容会比class要好
- 构造函数会进行预编译 class不会进行预编译
- 构造函数他可以当作普通函数调用 class不行

new构造函数的时候主要过程

- 自动创建对象
- 手动属性赋值
- 自动返回对象

使用工厂函数来声明创建（不需要new关键词）

```
//对象工厂函数 (忽略细节)
function factory(name){
    //创建一个对象
    let obj = new Object()
    //给对应的属性赋值
    obj.name = name
    //返回这个对象
    return obj
}
let obj = factory('jack')
console.log(obj)
```

使用工厂函数的主要过程

- 手动创建对象
- 手动属性赋值
- 手动返回对象

封装

抽取对应的属性和方法 （属性为名词 方法为动词）

示例

从前有座山 山里有个庙 庙里有老和尚 老和尚对小和尚说 山下的女人是老虎

对象的抽取

- 山

```
class Hill{
    constructor(temple){
        this.temple = temple //属性
    }
}
```

- 庙

```
class Temple{
    constructor(oldMonk){
        this.monk = oldMonk //属性
    }
}
```

- 老和尚、小和尚

```
class Monk{
    constructor(name){
        this.name = name //属性
        this.say = function(monk){
            console.log(`${this.name}对${monk.name}说山下的女人是老虎`)//方法
        }
    }
}
```

继承

子类继承父类的非私有属性及方法

示例

class的extends关键词来实现继承

```
//人类
class Person {
    constructor(name, age) {
        this.name = name
        this.age = age
        this.height = 180
        this.eatFood = function () {
            console.log(this.name + '在吃饭')
        }
    }
}
//使用学生类来继承人类
class Student extends Person {
    constructor(name, age, score) {
        super(name, age) //super指向对应的父类的构造函数
        this.score = score
    }
}
let student = new Student('张三', 18, 60)
console.log(student.name) //继承父类的属性
console.log(student.height) //继承父类的属性
student.eatFood() //继承父类的方法
console.log(student.score) //继承父类的属性
```

多态

一个内容的多种形态主要有俩个表现方式

- **重载 （在同一个类中有俩个同名的方法 通过参数个数及参数类型来分辨的）(JS没有重载)**

```
class Person{
    constructor(){
        // this.sayHello = function(a){
        //   console.log(a)
        // }
        // //覆盖上面的函数 因为对应的js是弱类型语言 不能强制指定类型 不能强制指定参数个数
        // this.sayHello = function(a,b){
        //   console.log(a,b)
        // }
        //模拟实现
        this.sayHello = function(a,b){
            if(arguments.length==1){
                console.log(a)
            }
            if(arguments.length==2){
                console.log(a,b)
            }
        }
    }
}
new Person().sayHello('hello')
new Person().sayHello('hello','world')
```

- **重写 （在子类中重写父类的方法）**

```
class Person{
    constructor(){
        this.sayHello = function(){
            console.log('父类的函数')
        }
    }
}
class Student extends Person{
    constructor(){
        super()
        this.sayHello = function(){
            console.log('重写的函数')
        }
    }
}
new Student().sayHello() //重写的函数
```

练习

- 编写一个动物类，该类包含name的属性，和say的方法。 通过say方法可以打印动物说话了。编写一个Dog类继承动物类，要求 该类中包含颜色的属性，该类重写say方法，要求打印父类的say方法里的内容，并且打印 动物颜色+动物名字+“叫了”。（备注狗会一直叫）

```
class Animal {
    constructor(name) {
        this.name = name
    }
    //先编译执行
    say () {
        console.log(this.name + '说话了')
```

```
    }
}
class Dog extends Animal {
    constructor(name, color) {
        super(name)
        this.color = color
        this.fn = this.say //接收的父类的say方法
        //重写say方法
        this.say = function () {
            let _this = this
            this.fn()
            setInterval(() => {
                console.log(_this.color + _this.name + '叫了')
            },1000)
        }
    }
}
let dog = new Dog('旺财', '土黄')
dog.say()
```

tab栏切换案例（面向对象）

- 提取属性 点击的按钮 切换的内容
- 提取方法 事件处理的方法 切换的方法

```
class Tab {
    constructor(naviBar, content) {
        this.naviBar = naviBar //传入的按钮
        this.content = content //传入的内容
        this.index = 0
        //调用handler
        this.handlerClick()
    }
    //切换对应的内容部分
    toggle() {
        //遍历对应的传入的按钮
        //排他
        Array.from(this.naviBar).forEach((btn) => {
            btn.className = ''
        })
        this.naviBar[this.index].className = 'select'
        //遍历对应的传入的内容
        Array.from(this.content).forEach((v) => {
            v.style.display = 'none'
        })
        this.content[this.index].style.display = 'block'
    }
    handlerClick() {
        var that = this
        //给对应的naviBar添加点击事件
        Array.from(this.naviBar).forEach((btn, i) => {
            btn.onclick = function () {
                that.index = i
                //切换
                that.toggle()
            }
        })
```

```
    }
}
//获取所有的a
var bar = document.querySelectorAll('.header>a')
//获取所有的内容
var content = document.querySelectorAll('.content>div')
new Tab(bar,content)
```

面向对象的拖拽

基础全局拖拽

```
//拖拽的元素 属性
//拖拽的处理就是对应方法
class Drag {
    constructor(element) {
        this.element = element //拖拽的元素
        //调用对应的鼠标事件
        this.handlerMouseEvent()
    }
    //处理鼠标事件
    handlerMouseEvent() {
        //先按下
        this.element.onmousedown = (e) => {
            e = e || window.event
            //记录按下的位置
            let downPoint = {
                x: e.offsetX,
                y: e.offsetY
            }
            //再移动
            document.onmousemove = (e) => {
                e = e || window.event
                //获取每次移动的位置
                var target = {
                    x:e.pageX - downPoint.x,
                    y:e.pageY - downPoint.y
                }
                //设置对应的element
                this.element.style.left = target.x + 'px'
                this.element.style.top =  target.y + 'px'
            }
            //再弹起
            document.onmouseup = () => {
                document.onmousemove = document.onmouseup = null
            }
        }
    }
}
new Drag(document.querySelector('div'))
```

区间拖拽

```
//拖拽的元素 属性
//拖拽的处理就是对应方法
export default class Drag {
```

```
  //拖拽的元素  区间元素
  constructor(element, intervalElement) {
    this.element = element
    this.intervalElement = intervalElement || document
    //记录按下的坐标
    this.downPoint = {
      x: 0,
      y: 0
    }
    //目标坐标
    this.targetPoint = {}
    this.down()
  }
  //移动的处理
  move() {
    this.intervalElement.onmousemove = (e) => {
      //记录移动的位置
      //减去对应的区间元素离页面的位置 区间内的位置
      let currentPoint = {
        x: e.pageX - this.getIntervalToPage().left,
        y: e.pageY - this.getIntervalToPage().top
      }
      //目标位置
      this.targetPoint = {
        x: currentPoint.x - this.downPoint.x,
        y: currentPoint.y - this.downPoint.y
      }
      //检索区间
      this.checkRange()
      //设置位置
      this.setPoint()
    }
  }
  setPoint() {
    //设置对应的位置
    this.element.style.left = this.targetPoint.x + 'px'
    this.element.style.top = this.targetPoint.y + 'px'
  }
  //区间判断
  checkRange() {
    //获取最大移动距离
    let max = {
      x: this.intervalElement.clientWidth - this.element.offsetWidth,
      y: this.intervalElement.clientHeight - this.element.offsetHeight,
    }
    //遍历
    for (var key in this.targetPoint) {
      //最小值判断
      if (this.targetPoint[key] < 0) {
        this.targetPoint[key] = 0
      }
      //最大值判断
      if (this.targetPoint[key] > max[key]) {
        this.targetPoint[key] = max[key]
      }
    }
  }
  //按下
```

```
  down() {
    //在对应的拖拽元素里面按下
    this.element.onmousedown = (e) => {
      e = e || window.event
      //记录对应的按下位置
      this.downPoint.x = e.offsetX
      this.downPoint.y = e.offsetY
      this.move()
      this.up()
    }
  }
  //弹起
  up() {
    document.onmouseup = () => {
      this.intervalElement.onmousemove = document.onmouseup = null
    }
  }
  //获取页面离区间的位置
  getIntervalToPage() {
    let element = this.intervalElement
    let distance = {
      left: 0,
      top: 0
    }
    while (element.offsetParent) {
      distance.left += element.offsetLeft
      distance.top += element.offsetTop
      element = element.offsetParent
    }
    return distance
  }
}
```

放大镜的实现

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

    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            .bigBox {
                width: 450px;
                height: 450px;
                position: relative;
                border: 1px solid #000;
            }

            .bigBox>img {
                width: 100%;
            }

            .moveBox {
                background-color: rgba(200, 200, 0, 0.5);
```

```
                position: absolute;
                left: 0;
                top: 0;
                display: none;
            }

            .imgBox {
                width: 540px;
                height: 540px;
                border: 1px solid #000;
                overflow: hidden;
                position: relative;
                display: none;
            }

            .imgBox>img {
                width: 800px;
                height: 800px;
                position: absolute;
            }
        </style>
    </head>

    <body>
        <div class="bigBox">
            <img
src="//img10.360buyimg.com//n0/jfs/t1/95534/23/29085/80747/627a190aE97686972/9d3
17c1e510b7eec.jpg" alt="">
            <div class="moveBox"></div>
        </div>
        <div class="imgBox">
            <img
src="https://img10.360buyimg.com//n0/jfs/t1/95534/23/29085/80747/627a190aE976869
72/9d317c1e510b7eec.jpg.avif"
                 alt="">
        </div>
        <script type="module">
            import Drag from './drag.js'
            //moveBox要在bigbox中移动
            //图片要在对应的imgbox里面移动
            //movebox的移动 鼠标永远在中心点 不需要按下 不需要弹起
            //使用放大镜继承拖拽
            class Loupe extends Drag{
                //传入四个元素分别为 大盒子 移动的盒子 图片盒子 移动的图片
                constructor(bigBox,moveBox,imgBox,bigImg){
                    super(moveBox,bigBox)//父类构造调用 按下操作（不需要）
                    // this.bigBox = bigBox
                    // this.moveBox = moveBox
                    this.imgBox = imgBox
                    this.bigImg = bigImg
                    //取消按下事件
                    this.element.onmousedown = null
                    //调用事件
                    this.enter()
                    this.leave()
                }
                //重写设置位置的函数
                setPoint() {
```

```
                    //设置对应的位置
                    this.element.style.left = this.targetPoint.x + 'px'
                    this.element.style.top = this.targetPoint.y + 'px'
                    //同时设置对应的大盒子里面的图片位置 这个位置绝对是负值
                    // bigBox(2)/moveBox(1) = bigImg(4)/imgBox (2)
                    //  bigBox(2) / bigImg(4) =   moveBox(1) / imgBox (2)
                    this.bigImg.style.left = -1 * this.targetPoint.x *
this.bigImg.clientWidth / this.intervalElement.clientWidth + 'px'
                    this.bigImg.style.top =  -1 *  this.targetPoint.y *
this.bigImg.clientHeight / this.intervalElement.clientHeight + 'px'
                }
                //移入
                enter(){
                    this.intervalElement.onmouseenter = ()=>{
                        //显示对应的element
                        this.element.style.display = 'block'
                        //显示ImgBox
                        this.imgBox.style.display = 'block'
                        //初始化移动盒子的宽高
                        this.init()
                        //设置对应的按下的中心点
                        this.downPoint = {
                            x: this.element.offsetWidth / 2,
                            y: this.element.offsetHeight / 2
                        }
                        this.move()
                    }
                }
                //移出
                leave(){
                    this.intervalElement.onmouseleave = ()=>{
                        //显示对应的element
                        this.element.style.display = 'none'
                        //显示ImgBox
                        this.imgBox.style.display = 'none'
                    }
                }
                init(){
                    //指定element的宽度和高度
                    // moveBox = bigBox(2)/ bigImg(4) * imgBox (2)
                    this.element.style.width = this.intervalElement.clientWidth
/ this.bigImg.clientWidth * this.imgBox.clientWidth + 'px'
                    this.element.style.height =
this.intervalElement.clientHeight / this.bigImg.clientHeight *
this.imgBox.clientHeight + 'px'
                }
            }
            var bigbox = document.querySelector('.bigBox')
            var moveBox = document.querySelector('.moveBox')
            var imgBox = document.querySelector('.imgBox')
            var bigImg = document.querySelector('.imgBox>img')
            new Loupe(bigbox,moveBox,imgBox,bigImg)
        </script>
    </body>

</html>
```