DAY17 面向对象基础和原型

2023年5月15日 16:33

面向对象的概述

一.面向对象的概述

1.首先，我们要明确，面向对象不是语法，是一个思想，是一种编程模式.

二.对比面向对象和面向过程。

1.面向过程：关注着过程的编程模式，按照顺一步一步完成，造成代码的冗余(重复)，开发中产生维护以及性能方面的影响

在面向过程的时候，我们要关注每一个元素，每一个元素之间的关系，顺序，。。。

2.面向对象：关注着对象的编程模式，类。利用对象解决问题，不适合简单的逻辑，适合复杂的，学习成本高，前端开发在工具化的时代，没有那么复杂，完全使用的时候也不多。

在面向对象的时候，我们要关注的就是找到一个对象来帮我做这个事情，我等待结果

面向对象就是对面向过程的封装

我们以前的编程思想是，每一个功能，都按照需求一步一步的逐步完成

我们以后的编程思想是，每一个功能，都先创造一个面馆，这个面馆能帮我们作出一个面（完成这个功能的对象），然后用 *面馆创造出一个面，我们只要等到结果就好了

解读：类->生成对象(属性和方法)

面馆(类)->不同种类面(对象)->对象具有相同的属性和方法(面粉...)

```
// const obj = new Object()
// const obj1 = {}
// const arr1 = [1, 2, 3];
// const arr2 = ['apple', 'zhangsan'];
```

三.梳理思想

面向对象适合复杂的逻辑，大型的项目，比较好管理维护扩展包括后续的迭代。

面向过程适合简单的逻辑，简洁的项目，

前端如果每一个逻辑都采用面向对象有点没必要，如果完全采用面向过程又不方便，形成的代码冗余(重复)

最终选择最适合前端开发的是函数式编程，介于面向过程和面向对象的中间。

但是还是要好好的掌握面向对象编程的思想，因为可以理解作者对js语言的一些配置，同时也能够更好的学习未来的工具。

四.学习目标

1.理解作者的那些类对象属性和方法是如何设计。

2.很多的经典工具都是采用面向对象的方式实现，理解工具的原理有帮助。

3.面向对象是js里面最深层次的应用，提升我们的基础以及工作能力。

4.面向对象可以帮助我们更好的学习后端语言，大部分后端语言都是基于面向对象的。

学习面向对象的好处和目标

1.几乎所有的企业级语言都是以面向对象为核心的开发模式。

2.开发中使用的工具，包括作者提供的对象都是采用面向对象进行开发的，更好的理解这门语言。

3.对作者提供的方法属性进行迭代，修改，替换...

4.提升JavaScript的核心基础。

找对象(创建对象)

1.利用构造函数或者字面量的方式创建对象

```
// const obj = new Object();
// obj.name = 'zhangsan';
// obj.age = 18;
// obj.sex = '男';
// obj.show = function () {
//   return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
// }
// const obj = {
//   name: 'zhangsan',
//   age: 18,
//   sex: '男',
//   show() {
//     return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
//   }
// }

// const obj = {};
// obj.name = 'zhangsan';
// obj.age = 18;
// obj.sex = '男';
// obj.show = function () {
//   return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
// }
// console.log(obj.name);
// console.log(obj.age);
// console.log(obj.sex);
// console.log(obj.show());
```

2.创建多个对象

```
// const obj1 = {
//   name: 'zhangsan',
//   age: 18,
//   sex: '男',
//   show() {
//     return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
//   }
// }
// const obj2 = {
//   name: 'lisi',
//   age: 19,
//   sex: '女',
//   show() {
//     return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
//   }
// }
```

问题：创建创建多个对象，会产生代码冗余

解决：利用函数封装解决

3.工厂函数

3.1. 先创建一个工厂函数

```
function createObj(name, age, sex) {
  3.2.手动创建一个对象
  const obj = new Object();
  3.3.手动的向对象中添加成员(属性和方法可以称之为对象的成员)
  obj.name = name;
  obj.age = age;
  obj.sex = sex;
  obj.show = function () {
    return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
  }
  3.4.手动返回一个对象
  return obj;
}

// let obj1 = createObj('zhangsan', 18, '男');
// let obj2 = createObj('lisi', 19, '女');
// let obj3 = createObj('wangwu', 20, '女');
// console.log(obj3.name);
// console.log(obj3.age);
// console.log(obj3.sex);
// console.log(obj3.show());
```

总结：工厂函数需要经历三个步骤

- 手动创建对象
- 手动添加成员
- 手动返回对象

问题：对象的识别问题，利用工厂函数生成的对象一定是利用Object构造函数创建出来的。因为创建过程是手动的(可见的)，使用这个方式创建的任何对象都是基于object。

解决：构造函数(类)

4.构造函数

首字母大写，new调用

```
function CreateObj(name, age, sex) {//构造函数(类)
  // 手动的向对象中添加成员(属性和方法可以称之为对象的成员)
  this.name = name;
  this.age = age;
  this.sex = sex;
  this.show = function () {
    return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
  }
}
let c1 = new CreateObj('zhangsan', 18, '男');//c1实例对象，来自于CreateObj类
console.log(c1.name, c1.age, c1.sex);
```

构造函数会比工厂函数简单一点

- 自动创建对象
- 手动添加成员
- 自动返回对象

注意：构造函数里面的this指向new出来的实例对象

总结：

找对象最终找到的是通过构造函数生成的实例对象。

```
// function Array() {//覆盖的系统提供的构造函数
//   this.length = arguments.length;
//   this.push = function () {
//   }
// }
// let arr1 = new Array(1, 2, 3, 4);
// console.log(arr1.length);
// arr1.push(5)
// console.log(arr1);
```

列出找对象的过程

1.利用自定义对象的创建方式来找对象

弊端：多对象产生代码冗余

2.封装函数实现 - 工厂函数

解决代码冗余，无法解决对象的识别问题，因为对象的创建和返回都是手动的，是可见的。

3.通过构造函数

构造函数可以解决对象的识别问题，因为创建对象和返回对象都是自动的，是不可见的，形成独立的对象。

总结：面向对象的核心就是对象，对象来自于构造函数(类)

自定义构造函数

面向对象的核心是找对象

找对象最适合的方式是通过自定义构造函数来实现

认识自定义的构造函数

1.使用构造函数的流程(面向对象开发的流程)

1.1.先书写一个构造函数

function Tab() { }

1.2.在构造函数内向对象添加一些成员(属性和方法)

```
function Tab() {
   this.titles = document.querySelectorAll('.tab_title a');//属性
   this.items = document.querySelectorAll('.item');//属性
   this.init = function () {//方法
     // this:实例对象
     this.titles.forEach((title, index) => {
       title.onclick = () => {
         this.titles.forEach((ele, i) => {
           this.titles[i].classList.remove('active');
           this.items[i].classList.remove('show');
         })
         title.classList.add('active');
         this.items[index].classList.add('show');
       }
     });
   }
}
```

1.3.使用这个构造函数创造一个对象（和 new 连用）

```
let t1 = new Tab();//使用   t1:实例对象      Tab:构造函数
t1.init();
```

new关键字的解读

1.工厂函数

```
function createObj(name, age, sex) {
  const obj = new Object();
  obj.name = name;
  obj.age = age;
  obj.sex = sex;
  obj.show = function () {
    return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
  }
  return obj;
}
```

2.构造函数

```
function CreateObj(name, age, sex) {
  // this = new Object(); //this引用该对象
  // 属性和方法被加入到 this 引用的对象中
  this.name = name;
  this.age = age;
  this.sex = sex;
  this.show = function () {
    return `我的名字叫${this.name},我今年${this.age}岁,我是${this.sex}的`;
  }
  // return this;  并且最后隐式的返回 this(系统自动完成)
}
```

3.new干了什么是天生具有的-JavaScript天生具有的

以 new 操作符调用函数的时候，函数内部发生以下变化：

3.1.自动创建一个空对象，并且this变量引用该对象。

3.2.属性和方法被加入到 this 引用的对象中。

3.3.并且最后隐式的返回 this

区分普函数和构造函数

```
// function fn() {
//   console.log(this);
// }
```

fn();//this->window

new fn();//this->实例对象 fn{}

```
// function fn() {
//   return '我是函数的返回值'
```

```
// }
```

console.log(fn());//我是函数的返回值

console.log(new fn());//fn {} fn是构造函数，构造函数会隐式返回this(实例对象)，注意构造函数里面的return是无效的。

构造函数的基本使用

一.自定义构造函数(类)的特点。

1.和普通函数一样，只不过调用的时候要和 new 调用，不然就是一个普通函数调用。

2.首字母大写只是根据作者的约定来的(推荐的写法，方便区分)，并不是构造函数不可缺的。

3.如果不需要传递参数，那么可以不写()，如果传递参数就必须写

4.每次new的时候，构造函数内部的 this 都是指向当前这次的实例对象

```
// function createPerson(name, age) {//构造函数
//   this.name = name;
//   this.age = age
// }
// new createPerson;  无法传递参数
// let p1 = new createPerson('zhangsan', 18);
// console.log(p1.name);
// console.log(p1.age);
```

二.面向对象的弊端(构造函数里面的成员都是私有的)

```
function createPerson(name, age) {//构造函数
  this.name = name;
  this.showname = function () {
    return '我的名字叫: ' + this.name;
  }
}

let p1 = new createPerson('zhangsan', 18); //实例化
let p2 = new createPerson('lisi', 19); //实例化
console.log(p1.showname == p2.showname);//false   p1下面的showname和p2下面的showname不是一个方法。

let a1 = new Array('zhangsan', 'lisi', 'wangwu');
console.log(a1.length);//3
console.log(a1.push('zhaoliu'));//4
let a2 = new Array('apple', 'banana');
console.log(a2.length);//2
console.log(a2.push('orange'));//3
console.log(a1.push == a2.push);//true    a1下面的push和a2下面的push是一个方法。
```

总结：

对象下面包含成员(属性和方法)

对象的属性描述的是对象的外观或者特点，应该是私有的。

对象的方法代表的是对象具有的某个功能，应用是公有的。

证明构造函数里面的成员都是私有的，方法不适合放在构造函数里面，而且系统提供的构造函数也没有放方法。

原型

一.原型(prototype)

原型的出现，就是为了解决 构造函数的缺点

1.每一个函数天生自带一个成员，叫做 prototype，是一个对象空间

2.即然每一个函数都有，构造函数也是函数，构造函数也有这个对象空间

3.这个prototype对象空间可以由函数名来访问

4.将公有的属性和方法放到这个对象空间里面。

5.原型里面的this依然指向实例对象

```
function CreatePerson(name, age) {//构造函数
  this.name = name;//私有的属性
  this.showname = function () {//私有的方法
    return '我的名字叫: ' + this.name;
  }
}
```

第一种实现：

```
CreatePerson.prototype = {//公有的对象空间
  age: 18,//公有的属性
  showage: function () {
    return '我的年龄是: ' + this.age;
  }
}
```

第二种实现：

```
CreatePerson.prototype.age = 18
CreatePerson.prototype.showage = function () {//公有的对象空间
  return '我的年龄是: ' + this.age;
}


let p1 = new CreatePerson('zhangsan', 18); //实例化
```

```
let p2 = new CreatePerson('lisi', 19); //实例化

console.log(p1.showname == p2.showname);//false
console.log(p1.showage == p2.showage);//true  可以确定系统使用的方法就是绑定在原型prototype上面
```

总结：

面向对象开发的方式：混合开发(构造函数+原型)

将私有的属性写在构造函数中，将公有的方法写在原型上

构造函数：负责私有的属性

原型：负责公有的方法

系统构造函数也是这样设置的

```
function Array() { }
Array.prototype.push = function () {
  return 'hehe'
};
......
let a1 = new Array(1, 2, 3, 4);
console.log(a1.length);//4
console.log(a1.push(5));//5
console.log(a1);// [1, 2, 3, 4, 5]
```