第六章面向对象的程序设计 笔记

创建对象

- 工厂模式 由于构造函数和对象字面量创建多个对象时会产生大量重复的代码 因此工厂模式诞生

```
function createPerson(name,age,job){
var o = new Object();
o.name = name;
o.age = age;
o.job = job;
o.sayName = function(){
    alert(this.name);
}
return o;
}

var person1 = createPerson('jack',18,'student');

person1.sayName();
//工厂模式实际上就是用一个函数封装了创建对象的过程；缺点是 不能知道对象的类型
```

- 构造函数模式
    - 与工厂模式相比 构造函数名多数为大写开头 没有显式地创建对象（因此在调用时就要new） 没有return
    - 其实有点Java里 class 内味
    - 缺点 每当创建一个实例时 就会创建一个新的函数对象（sayName） 比方说新建了两个Person类的实例 他们都有相同的 sayName 方法，可是由于是两个指向不同的实例，因此就会出现实现同样功能却不是同一个对象的问题

```
function Person(name,age,job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.sayName = function(){
        alert(this.name);
    }
}

var person1 = new Person('jack',15,'student')
person1.sayName();//jack

alert(person1 instanceof Object);//true
alert(person1 instanceof Person);//true
```

- 缺点的解决方法 可以将定义在Person 内部的方法 放在外部定义（全局函数） 那这样封装又没有意义了。。该怎么办咧？（原型模式可以解决这个问题！）
- 原型模式
    - 我们创建的每一个函数都有一个属性---prototype（原型） 该属性是一个指向一个对象的指针
    - 不必在构造函数中定义对象实例的信息 而是将这些信息直接添加到原型对象中
    - 但是新对象所有的属性和方法都是共享的

```
function Person(){
}

Person.prototype.name = 'jack';
Person.prototype.age = 18;
Person.prototype.job = 'student';
Person.prototype.sayName = function(){
    alert(this.name);
}

var person1 = new Person();
person1.sayName();//jack

var person2 = new Person();
person2.sayName();//jack

alert(person1.sayName == person2.sayName);//true
```

- 不能通过实例对象重写原型中的值

```
//接上一段代码
var person3 = new Person();
person3.name = 'rose';
alert(person3.name);//rose 来自实例对象
alert(person1.name);//jack 来自原型

delete person3.name;//将实例对象中的name属性删除 就不会屏蔽原型对象中的属性
alert(person3.name);//jack 来自原型
```

- 原型模式简写

```
//原型模式更简单的写法 就是和对象字面量结合
function Person(){
```

```
}

Person.prototype = {
    //construtoe:Person,
    name:'jack',
    age:18,
    job:'student',
    sayName:function(){
        alert(this.name);
    }
}
```

- 原型模式的动态性 即使是先创建实例再修改原型对象的属性，也会从实例中反映出来；

组合使用构造函数模式和原型模式

```
//组合使用构造函数和原型模式
function Person(name,age,job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.friends = ['Shelby','Court'];
}

Person.prototype = {
    constructor : Person,
    sayName:function(){
        alert(this.name);
    }
}

var person1 = new Person('jack',20,'student');
var person2 = new Person('rose',19,'student');

person1.friends.push('Van');
alert(person1.friends);//Shelby,Court,Van
alert(person2.friends);//Shelby,Court
```

- 动态原型模式 将所有信息都封装到构造函数里 且在有必要的情况下初始化原型对象
- 寄生构建函数模式
- 稳妥构建函数模式

继承

原型链实现继承

- 原型链 通过原型对象等于另一个类型的实例来实现继承

```
//原型链
function SuperType(){
    this.prototype = true;
}

SuperType.prototype.getSuperValue = function(){
    return this.prototype;
}

function SubType(){
    this.subprototype = false;
}
//新建一个SuperType的实例以达到Subtype继承SuperType的目的
SubType.prototype = new SuperType();

//重新将方法写入SubType的原型（SuperTyper实例）中
SubType.prototype.getSuperValue = function(){
    return this.subprototype;
}

var instance = new SubType();
alert(instance.getSuperValue());//false 书上写的是true? ? 可是明明已经重写了
getSuperValue方法欸
```

- 确定原型和实例的关系

```
//确定原型和实例的关系
alert(instance instanceof Object);//true
alert(instance instanceof SubType);//true
alert(instance instanceof SuperType);//true
```

- 通过原型链实现继承时不能使用对象字面量创建原型方法，因为这样做会重写原型链

借用构造函数 实现继承

- 原型链的问题 属性共享的问题 创建子类的实例是不能传参（没办法再不影响对象实例的情况下传参）

```
//原型链的问题
function SuperType(){
    this.colors = ['red','blue','green'];
}


function SubType(){
}
//继承
SubType.prototype = new SuperType();
```

```
var instance1 = new SubType();
instance1.colors.push('black');
alert(instance1.colors);//red,blue,green,black

var instance2 = new SubType();
alert(instance2.colors);//red,blue,green,black 两个实例的colors属性实际上都是来自
SubType的原型SuperType的实例
```

- 借用构造函数

```
//借用构造函数
function SuperType(){
    this.colors = ['red','blue','green'];
}

function SubType(){
    //继承
    //在此处调用父类的构造函数啦
    SuperType.call(this);
}

var instance1 = new SubType();
instance1.colors.push('black');
alert(instance1.colors);//red,blue,green,black

var instance2 = new SubType();
alert(instance2.colors);//red,blue,green
```

- 传递参数

```
//借用构造函数传参
function SuperType(name){
    this.name = name;
}

function Subtype(){
    //继承SuperType 同时传参
    SuperType.call(this,'jack');
    this.age = 19;
}

var instance = new Subtype();
alert(instance.name);//jack
alert(instance.age);//19
```

- 由于方法都在构造函数中定义 在外部还是不能传参 没有复用性可言 因此也不会单独使用

组合继承 （原型链和借用构造函数组合使用）最常用

```
//组合继承
//父类构造函数
function SuperType(name){
    this.name = name;
    this.colors = ['red','blue','yellow'];
}
//父类方法定义在他的原型对象中 避免多次创建function对象的现象发生
SuperType.prototype.sayName = function(){
    alert(this.name);
}

//子类继承属性
function SubType(name,age){
    //继承父类的属性name
    SuperType.call(this,name);
    this.age = age;
}

//子类继承方法
SubType.prototype = new SuperType();
//在子类的实例对象（即父类的实例）中定义新方法
SubType.prototype.sayAge = function(){
    alert(this.age);
}

var person1 = new SubType('rose',18);
person1.colors.push('black');
alert(person1.colors);//red,blue,yellow,black
person1.sayName();//rose
person1.sayAge();//18

var person2 = new SubType('jack',20);
person2.colors.push('green');
alert(person2.colors);//red,blue,yellow,black,green
person2.sayName();//jack
person2.sayAge();//20
```

- 总之就是 方法用原型链继承 属性用借用构造函数继承

原型式继承

寄生式继承

寄生组合式继承