高级Day 03---JavaScript的继承

育知同创 YU ZHI TONG CHUANG

一、继承的概念

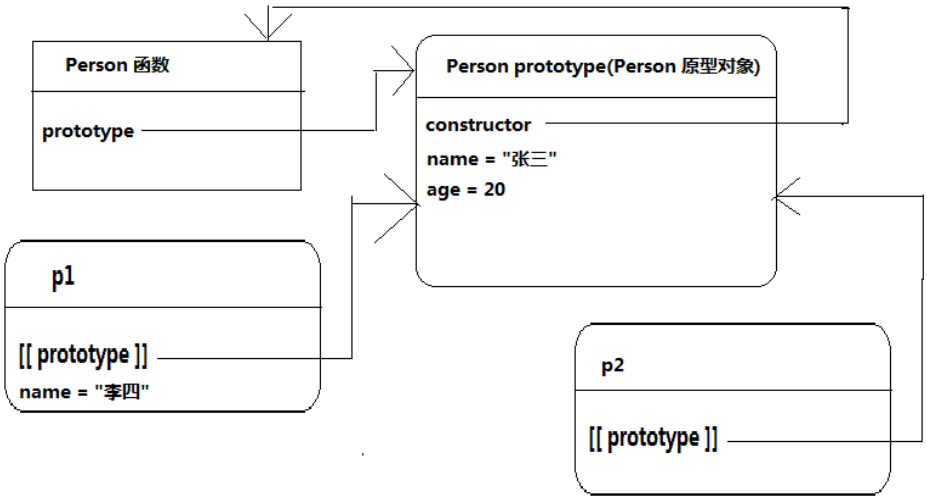
继承是所有的面向对象的语言最重要的特征之一。大部分的oop语言的都支持两种继承：接口继承和实现继承。比如基于类的编程语言Java，对这两种继承都支持。从接口继承抽象方法 (只有方法签名)，从类中继承实例方法。

但是对JavaScript来说，没有类和接口的概念(ES6之前)，所以只支持实现继承，而且继承在 原型链 的基础上实现的。等了解过原型链的概念之后，你会发现继承其实是发生在对象与对象之间。这是与其他编程语言很大的不同。

二、原型链的概念

> 在JavaScript中，将原型链实现继承的主要方法。其基本思想是利用原型让一个引用类型继承另一个引用类型的属性和方法

再回顾下，构造函数、原型(对象)和对象之间的关系。每个构造函数都有一个属性 prototype 指向一个原型对象，每个原型对象也有一个属性 constructor 指向函数，通过new 构造函数() 创建出来的对象内部有一个不可见的属性[[prototype]]指向构造函数的原型。当每次访问对象的属性和方法的时候，总是先从p1中找，找不到则再去p1指向的原型中找。



> 下面我们开始一步步的构造原型链，来实现继承

2.1 更换构造函数的原型

原型其实就是一个对象，只是默认情况下原型对象是浏览器会自动帮我们创建的，而且自动让构造函数的 prototype 属性指向这个自动创建的原型对象。

其实我们完全可以把原型对象更换成一个我们自定义类型的对象。

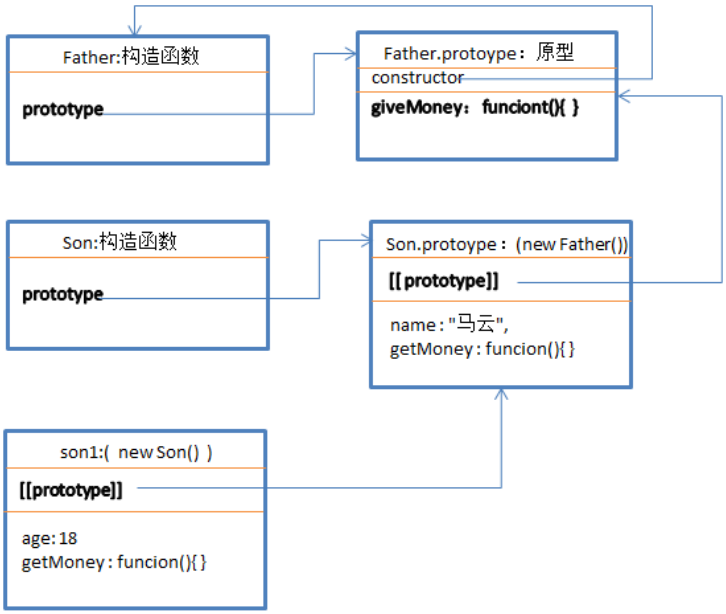
> 看下面的代码：

```
<script type="text/javascript">
    //定义一个构造函数。
    function Father () {
        // 添加name属性.  默认直接赋值了。当然也可以通过构造函数传递过来
        this.name = "马云";
    }
    //给Father的原型添加giveMoney方法
    Father.prototype.giveMoney = function () {
        alert("我是Father原型中定义的方法");
    }
    //再定义一个构造函数。
    function Son () {
        //添加age属性
        this.age = 18;
    }
    //关键地方：把Son构造方法的原型替换成Father的对象。
    Son.prototype = new Father();
    //给Son的原型添加getMoney方法
    Son.prototype.getMoney = function () {
        alert("我是Son的原型中定义的方法");
    }
    //创建Son类型的对象
    var son1 = new Son();

    //发现不仅可以访问Son中定义属性和Son原型中定义的方法，也可以访问Father中定义的属性和Father原型中的方法。
    //这样就通过继承完成了类型之间的继承。
    // Son继承了Father中的属性和方法，当然还有Father原型中的属性和方法。
    son1.giveMoney();
    son1.getMoney();
    alert("Father定义的属性：" + son1.name);
    alert("Son中定义的属性：" + son1.age);

</script>
```

> 上面的代码其实就完成了Son继承Father的过程。那么到底是怎么完成的继承呢？
>
> 看下面的示意图：



> 说明：

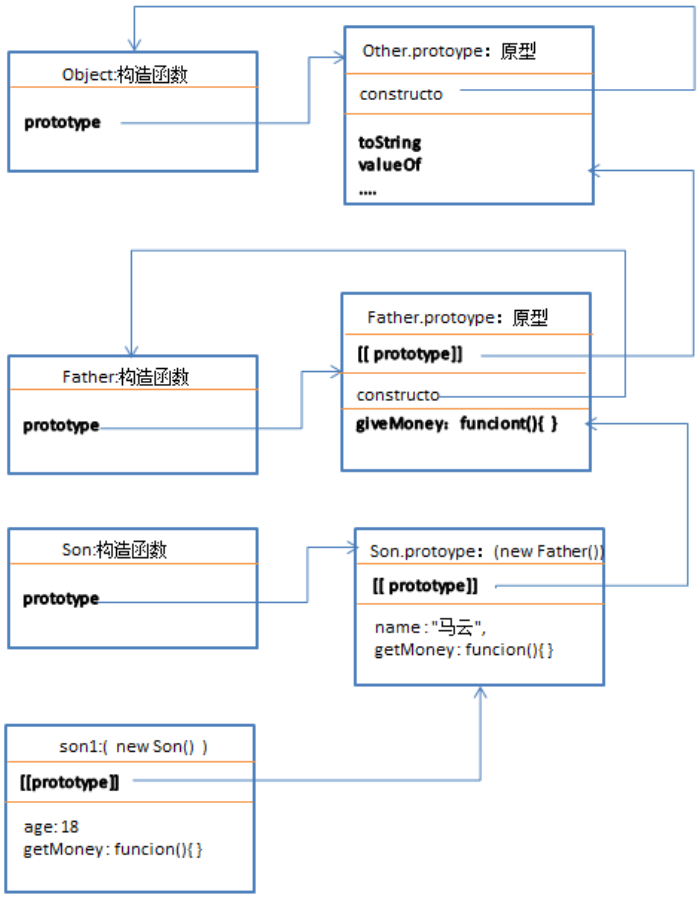
1. 定义Son构造函数后，我们没有再使用Son的默认原型，而是把他的默认原型更换成了Father类型对象。
2. 这时，如果这样访问 son1.name, 则先在son1中查找name属性，没有然后去他的原型(Father对象)中找到了，所以是"马云"。
3. 如果这样访问 son1.giveMoney(), 则现在son1中这个方法，找不到去他的原型中找，仍然找不到，则再去这个原型的原型中去找，然后在Father的原型对象找到了。
4. 从图中可以看出来，在访问属性和方法的时候，查找的顺序是这样的：对象->原型->原型的原型->...->原型链的顶端。 就像一个链条一样，这样由原型连成的"链条"，就是我们经常所说的原型链。
5. 从上面的分析可以看出，通过原型链的形式就完成了JavaScript的继承。

2.2 默认顶端原型

其实上面原型链还缺少一环。

在 JavaScript 中所有的类型如果没有指明继承某个类型，则默认是继承的 Object 类型。这种 默认继承也是通过原型链的方式完成的。

> 下面的图就是一个完整的原型链：



> 说明：

1. 原型链的顶端一定是Object的原型对象。这也是为什么我们随意创建一个对象，就有很多方法可以调用，其实这些方法都是来自Object的原型对象。
2. 通过对象访问属性方法的时候，一定是会通过原型链来查找的，直到原型链的顶端。
3. 一旦有了继承，就会出现多态的情况。假设需要一个Father类型的数据，那么你给一个Father对象，或Son对象都是没有任何问题的。而在实际执行的过程中，一个方法的具体执行结果，就看在原型链中的查找过程了。给一个实际的Father对象则从Fahter的原型链中查找，给一个实际的Son则从Son的原型链中查找。
4. 因为继承的存在，Son的对象，也可以看出Father类型的对象和Object类型的对象。 子类型对象可以看出一个特殊的父类型对象。

2.3 测试数据的类型

到目前为止，我们有3中方法来测试数据的类型。

1. typeof：一般用来测试简单数据类型和函数的类型。如果用来测试对象，则会一直返回object，没有太大意义。

```
<script type="text/javascript">
    alert(typeof 5); // number
    var v = "abc";
    alert(typeof v);  // string
    alert(typeof function () {

    });  //funcion
    function Person () {

    }
    alert(typeof new Person()); //object

</script>
```

2. instanceof: 用来测试一个对象是不是属于某个类型。结果为boolean值。

```
<script type="text/javascript">
    function Father () {
    }
    function Son () {
    }

    Son.prototype = new Father();
    var son = new Son();
    alert(son instanceof Son);  // true
    // Son通过原型继承了Father
    alert(son instanceof Father);  // true
    //Father又默认继承了Objcet
    alert(son instanceof Object); // true
</script>
```

3. isPrototypeOf(对象) : 这是个原型的方法，参数传入一个对象，判断参数对象是不是由这个原型派生出来的。 也就是判断这个原型是不是参数对象原型链中的一环。

```
<script type="text/javascript">
    function Father () {

    }
    function Son () {

    }

    Son.prototype = new Father();
    var son = new Son();
    alert(Son.prototype.isPrototypeOf(son));  // true
    alert(Father.prototype.isPrototypeOf(son)); // true
    alert(Object.prototype.isPrototypeOf(son)); // true
</script>
```

2.4 原型链在继承中的缺陷

原型链并非完美无缺，也是存在一些问题的。

2.4.1 父类型的属性共享问题

在原型链中，父类型的构造函数创建的对象，会成为子类型的原型。那么父类型中定义的实例属性，就会成为子类型的原型属性。对子类型来说，这和我们以前说的在原型中定会以方法，构造函数中定义属性是违背的。子类型原型中的属性被所有的子类型的实例所共有，如果有个一个实例去更改，则会很快反应的其他的实例上。

> 看下面的代码：

```
<script type="text/javascript">
    function Father () {
        this.girls = ["志玲", "凤姐"];
    }
    function Son () {

    }
    // 子类的原型对象中就有一个属性 girls ，是个数组
    Son.prototype = new Father();
    var son1 = new Son();
    var son2 = new Son();
    //给son1的girls属性的数组添加一个元素
    son1.girls.push("亦非");
    //这时，发现son2中的girls属性的数组内容也发生了改变
    alert(son2.girls);  // "志玲", "凤姐", "亦非"
</script>
```

2.4.2 向父类型的构造函数中传递参数问题

在原型链的继承过程中，只有一个地方用到了父类型的构造函数，Son.prototype = new Father();。只能在这个一个位置传递参数，但是这个时候传递的参数，将来对子类型的所有的实例都有效。

如果想在创建子类型对象的时候传递参数是没有办法做到的。

如果想创建子类对象的时候，传递参数，只能另辟他法。

三、借用构造函数调用继承

3.1 借用的方式

> 借用构造函数调用继承，又叫伪装调用继承或冒充调用继承。虽然有了继承两个子，但是这种方法从本质上并没实现继承，只是完成了构造方法的调用而已。

使用call或apply这两个方法完成函数借调。这两个方法的功能是一样的，只有少许的区别(暂且不管)。功能都是更改一个构造方法内部的this指向到指定的对象上。

> 看下面的代码：

```
<script type="text/javascript">
    function Father (name,age) {
        this.name = name;
        this.age = age;

    }
    //这样直接调用，那么father中的this只的是 window。 因为其实这样调用的： window.father("李四", 20)
    // name 和age 属性就添加到了window属性上
    Father("李四", 20);
    alert("name:" + window.name + "\nage:" + window.age);

    //使用call方法调用，则可以改变this的指向
    function Son (name, age, sex) {
        this.sex = sex;
        //调用Father方法(看成普通方法)，第一个参数传入一个对象this，则this(Son类型的对象)就成为了
Father中的this
        Father.call(this, name, age);
    }
    var son = new Son("张三", 30, "男");
    alert("name:" + son.name + "\nage:" + son.age + "\nsex:" + son.sex);
    alert(son instanceof Father); //false
</script>
```

> 函数借调的方式还有别的实现方式，但是原理都是一样的。但是有一点要记住，这里其实并没有真的继承，仅仅是调用了Father构造函数而已。也就是说，son对象和Father没有任何的关系。

3.2 借用的缺陷

Father的原型对象中的共享属性和方法，Son没有办法获取。因为这个根本就不是真正的继承。

四、组合继承

组合函数利用了原型继承和构造函数借调继承的优点，组合在一起。成为了使用最广泛的一种继承方式。

```
<script type="text/javascript">
    //定义父类型的构造函数
    function Father (name,age) {
        // 属性放在构造函数内部
        this.name = name;
        this.age = age;
        // 方法定义在原型中
        if((typeof Father.prototype.eat) != "function"){
            Father.prototype.eat = function () {
                alert(this.name + " 在吃东西");
            }
        }

    }
    // 定义子类类型的构造函数
    function Son(name, age, sex){
        //借调父类型的构造函数，相当于把父类型中的属性添加到了未来的子类型的对象中
        Father.call(this, name, age);
        this.sex = sex;
    }
    //修改子类型的原型。这样就可以继承父类型中的方法了。
    Son.prototype = new Father( );
    var son1 = new Son("志玲", 30, "女");
    alert(son1.name);
    alert(son1.sex);
    alert(son1.age);
    son1.eat();
</script>
```

> 说明：

1. 组合继承是我们实际使用中最常用的一种继承方式。
2. 可能有个地方有些人会有疑问：Son.prototype = new Father( );这不照样把父类型的属性给放在子类型的原型中了吗，还是会有共享问题呀。但是不要忘记了，我们在子类型的构造函数中借调了父类型的构造函数，也就是说，子类型的原型中有的属性，都会被子类对象中的属性给覆盖掉。就是这样的。