我所理解的继承

ES6之前的继承

es6之前，javascript本质上不能算是一门面向对象的编程语言，因为它对于封装、继承、多态这些面向对象语言的特点并没有在语言层面上提供原生的支持。但是，它引入了原型`prototype`的概念，可以以另一种方式模仿类，并通过原型链的方式实现了父类子类之间共享属性的继承以及身份确认机制。所以在ES6之前，继承在**原型链**的基础上实现的，继承其实是发生在对象与对象之间。

1. 原型链继承

简而言之，就是直接将父类的一个实例赋给子类的原型

```
function Person(name) {
    this.name = name;
    this.className = 'person';
}
Person.prototype.getClassName = function () {
    console.log(this.className);
}

function Man() {

}
Man.prototype = new Person();
let man = new Man();
man.getClassName(); //person
```

> 这种方法是直接new了一个父类的实例，然后赋给子类的原型。这样也就相当于直接将父类原型中的方法属性以及挂在this上的各种方法属性全赋给了子类的原型，简单粗暴！再来看看man,它是Man的一个实例，因为man本身没有getClassName方法，那么就会去原型链上去找，找到的是person的getClassName。这种继承方式下，所有的子类实例会共享一个父类对象的实例，这种方案最大问题就是**子类无法通过父类创建私有属性**。比如每一个Person都有一个名字，我们在初始化每个Man的时候要指定一个不同名字，然后子类将这个名字传递给父类，对于每个man来说，保存在相应person中的name应该是不同的，但是这种方式根本做不到。所以，这种继承方式，实战中基本不用。

2. 借用构造函数

```
function Person(name) {
    this.name = name;
    this.className = 'person';
}
Person.prototype.getName = function () {
    console.log(this.name);
}
```

```
function Man(name) {
    Person.apply(this, arguments);
}
let man1 = new Man('Davin');
let man2 = new Man('Jack');

console.log(man1.name); // Davin
console.log(man2.name); // Jack
console.log(man1.getName()); // 报错
```

> 这里在子类的在构造函数里用子类实例的this去调用父类的构造函数，从而达到继承父类属性的效果。这样一来，每new一个子类的实例，构造函数执行完后，都会有自己的一份资源(name)。但是这种办法只能继承父类构造函数中声明的实例属性，并没有继承父类原型的属性和方法，所以就找不到getName方法，所以1处会报错。为了同时继承父类原型，从而诞生了组合继承的方式:

3. 组合继承

```
function Person(name) {
    this.name = name || 'default name';
    this.className = 'person';
}
Person.prototype.getName = function () {
    console.log(this.name);
}
function Man(name) {
    Person.call(this, name);
}
Man.prototype = new Person();
let man1 = new Man('Davin');

console.log(man1.name); // Davin
console.log(man1.getName()); // Davin
```

> 这个例子很简单，这样不仅会继承构造函数中的属性，也会复制父类原型链中的属性。但是，有个问题，`Man.prototype = new Person();` 这句执行后，Man的原型为：`{name: "default name", className: "person"}`，也就是说Man的原型中已经有了一个name属性，而之后创建man1时传给构造的函数的name则是通过this重新定义了一个name属性，相当于只是覆盖掉了原型的name属性（原型中的name依然还在），这样很不优雅。

4. 分离组合继承

这是目前es5中主流的继承方式，可以将继承分为两步：构造函数属性继承和建立子类和父类原型的链接。所谓的分离就是分两步走；组合是指同时继承子类构造函数和原型中的属性。

```
function Person(name) {
    this.name = name;
    this.className = 'person';
```

```
}
Person.prototype.getName = function () {
    console.log(this.name);
}
function Man(name) {
    Person.apply(this, arguments)
}
//注意此处
Man.prototype = Object.create(Person.prototype);
let man1 = new Man('Davin');

console.log(man1.name); // Davin
console.log(man1.getName()); // Davin
```

> 这里用到了`Object.creat(obj)`方法，该方法会对传入的obj对象进行浅拷贝。和上面组合继承的主要区别就是：将父类的**原型**复制给了子类原型。这种做法很清晰：
>
> 1. 构造函数中继承父类属性 / 方法，并初始化父类。
> 2. 子类原型和父类原型建立联系。

ES6的继承方式

ES6提供了更接近传统语言的写法，引入了Class（类）这个概念，作为对象的模板。通过`class`关键字，可以定义类。基本上，ES6的`class`可以看作只是一个语法糖，它的绝大部分功能，ES5都可以做到，新的`class`写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已。

```
class User {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    fullName() {
        return `${this.firstName} ${this.lastName}`
    }
}

let user = new User('David', 'Chen');
user.fullName()  // David Chen
```

以上的类定义语法非常直观，它跟以下的ES5语法是一个意思：

```
function User(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

User.prototype.fullName = function () {
    return '' + this.firstName + this.lastName
}
```

ES6 并没有改变 JavaScript 基于原型的本质，只是在此之上提供了一些语法糖。`class` 就是其中之一。其他的还有 `extends`， `super` 和 `static` 。它们大多数都可以转换成等价的 ES5 语法。

我们来看看一个`class`继承的例子：

```
class Parent {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    fullName() {
        return `${this.firstName} ${this.lastName}`;
    }
};
// extends:继承
class Child extends Parent {
    constructor(firstName, lastName, age) {
        super(firstName, lastName);
        this.age = age;
    }

    fullName() {
        return `${super.fullName()} (${this.age})`;
    }
};

let xiaoming = new Child('小', '明', 16);

console.log(xiaoming.fullName()); // 小明16
```