07-第七章 ECMAScript6 Class 类 继承

ECMAscript 6

> S6 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

二、Class 基本用法

```
class Me{}
```

```
class Me{
    constructor(x,y){
            this.x = x;
            this.y = y;
        }

}
```

二、Class 表达式

> 与函数一样，类也可以使用表达式的形式定义

```
const MyClass = class Me{}
```

如果在内部不需要使用`当前类`，即`Me`，那么Me还可以省掉

```
const MyClass = class {};
        console.log( MyClass); //class {}
```

采用 Class 表达式，可以写出`立即执行的 Class`

```
const obj = new class {};
        console.log( obj );//class {}
```

上面代码使用表达式定义了一个类。需要注意的是，这个类的名字是MyClass而不是Me，Me只在 Class 的`内部代码可用`，指代`当前类`。

三、constructor 方法

一个类`必须有constructor方法`，通过new命令生成对象实例时，自动调用该方法。，如果没有显式定义，一个空的constructor方法会被默认添加。

```
class Point{}

//等同于
class Point{
    constructor(){}
}
```

constructor方法默认返回实例对象（即this），完全可以指定返回另外一个对象。

```
class Point{
        constructor(){
            return Object.create(null);
        }
       }
```

四、Class方法

```
onst MyClass = class Me{
            getClassName(){
                return Me.name
            }
        }
        //var c = new Me();//报错
        var c = new MyClass();
        console.log(c.getClassName() );//Me
        console.log( c.constructor.name )//Me
```

> 注意，定义“类”的方法的时候，前面不需要加上function这个关键字，直接把函数定义放进去了就可以了。另外，方法之间不需要逗号分隔，加了会报错

e6所有的方法都是定义在Class的`prototype`上面的

```
class Point {
  constructor() {
    // ...
  }
  toString() {
    // ...
  }
  toValue() {
    // ...
  }
}
// 等同于
Point.prototype = {
  constructor() {},
  toString() {},
  toValue() {},
};
```

```
class Point{};
        console.log(typeof Point);//function
        console.log( Point.prototype.constructor == Point );//true
        var obj = new Point;
        console.log( obj.constructor == Point.prototype.constructo
r );//true
```

五、Static 静态方法

> 类相当于实例的原型，所有在类中定义的方法，都会被实例继承。如果在一个方法前，加上static关键字，就表示该方法不会被实例继承，而是直接通过类来调用，这就称为“ 静态方法 ”

```
        class A{
            static hello(){
                console.log('static');
            }
            static bar(){

            }
        }
        A.hello();//static
```

六、Class改写传统写法

传统写法

```
function Point(x,y){
            this.x = x;
            this.y = y;
        }
        Point.prototype.show = function(){
            return `x:${this.x}\ny:${this.y}`;
        }

        var p = new Point(1,6);
       console.log(  p.show() );
       //x:1 y:6
```

es6 的Class改写

```
class Point{
            constructor(x,y){
                this.x = x;
                this.y = y;
            }
            show(){
                return `x:${this.x}\ny:${this.y}`;
            }
       }
       var p = new Point(1,6);
       console.log(p.show());// x:1 y:6
```

上面代码定义了一个“类”，可以看到里面有一个constructor方法，这就是构造方法，而this关键字则代表实例对象。也就是说，ES5 的构造函数Point，对应 ES6 的Point类的构造方法

七、Class 的取值函数(getter)和存值函数(setter)

> 与 ES5 一样，在“类”的内部可以使用get和set关键字，对某个属性设置存值函数和取值函数，拦截该属性的存取行为。

```
class A{
            constructor(x,y){
                this.x = x;
                this.y = y;
            }
            get prop(){
                return this.x;

            }
            set prop(val){
                this.x = val;
            }

        }

        var a = new A(10,20);
        console.log(a.prop);//10
        a.prop = 100;
        console.log(a.prop);//100
```

存值函数和取值函数是设置在prototype属性的 Descriptor 对象上的。

```
class A{
            constructor(x,y){
                this.x = x;
                this.y = y;
            }
            get prop(){
                return this.x;

            }
            set prop(val){
                this.x = val;
            }

        }

         var obj = new A(10,20);
        var desc = Object.getOwnPropertyDescriptor(A.prototype,'prop');
        console.log( 'set' in desc );//true
        console.log( desc.set.name );//set prop
```

上面代码中，存值函数和取值函数是定义在prop属性的描述对象上面，这与 ES5 完全一致。

八、Class 不存在变量提升

> 类不存在变量提升（hoist），这一点与 ES5 完全不同

```
        new Point();//ReferenceError: Point is not defined
        class Point{};
```

九、new.target

> ES6 为new命令引入了一个 `new.target` 属性，该属性一般用在构造函数之中，如果该函数是通过 `new` 调用的会返回构造函数，new.target会返回 `undefined` ，因此这个属性可以用来确定构造函数是怎么调用的。

```
        function Point(){
            console.log(new.target == this.constructor);
        }
        //let p = new Point();//true
        Point();//false;
```

在Class类中也是如此

```
class Point{
            constructor(){
                console.log( new.target === this.constructor );
            }
        }

        new Point();//true
```

二、Class继承

> Class 可以通过extends关键字实现继承，这比 ES5 的通过修改原型链实现继承，要清晰和方便很多。

一、 `extends`

```
        class A{}
         class B extends A{}
```

上面代码定义了一个B类，该类通过extends关键字，继承了A 类的`所有属性和方法`。但是由于没有部署任何代码，所以这两个类完全一样，等于复制了一个A类

二、 constructor

> 任何一个`子类都有constructor方法`，如果子类没有定义constructor方法，这个方法会被默认添加，代码如下。也就是说，不管有没有显式定义，

(1)代码等同于

```
class A{}
    class B extends A{
        constructor(){
            super();//调用父类constructor()
        }
    }
    var b = new B();
```

```
    class A{
        constructor(){
            console.log( new.target.name)
        }
    }
    class B extends A{
    }
    new A();//A
    new B();//B
```

三、 super 关键字

> super这个关键字，既可以当作函数使用，也可以当作对象使用

一、super在`constructor`函数中作为`函数调用`时，代表父类的(`constructor`)构造函数，在其他地方就会报错。

注意

子类必须在constructor方法中调用`super`方法，否则新建实例时会报错。这是因为子类没有自己的this对象，而是继承父类的this对象，然后对其进行加工。如果不调用super方法，子类就得不到`this`对象

```
        class A{}
        class B extends A{
            constructor(){}
        }
        var b = new B();//ReferenceError: this is not defined
```

B继承了父类A，但是它的构造函数没有调用super方法，导致新建实例时报错。

另一个需要注意的地方是，在子类的构造函数中，只有调用`super`之后，才可以使用`this`关键字，否则会报错。这是因为子类实例的构建，是基于对父类实例加工，只有super方法才能返回父类实例。

```
        class A{}
        class B extends A{
            constructor(x){
                //this.x = x;//ReferenceError: this is not define
d
                super();
                this.x = x;//ReferenceError: this is not defined
            }
        }
        var b = new B(10);
```

二、super作为对象时，在普通方法中，指向父类的(`prototype`)原型对象

```
    class A{
            show(){
                console.log('ok');
            }
        }
        A.prototype.x =2;
        class B extends A{
            constructor(){
                super();
                //下面都是指向A.prototype
                super.show();//ok
                console.log(super.x);//2
            }

        }

       new B();
```

三、、super作为对象时,在静态方法中，指向`父类`。

```
        class A{
             show(){
                console.log('ok');
             }
        }
        class B extends A{
           static hello(){
            super.prototype.show();
           }

        }
        B.hello();//hello
```

注意

使用super的时候，必须显式指定是作为函数、还是作为对象使用，否则会报错。

> ES5 的继承，实质是先创造子类的实例对象this，然后再将父类的方法添加到this上面（Parent.apply(this)）。ES6 的继承机制完全不同，实质是先创造父类的实例对象this（所以必须先调用super方法），然后再用子类的构造函数修改this。

```
        class A{
            constructor(x,y){
                this.x = x;
                this.y = y;
            }
            show(){
                return `(${this.x},${this.y})`
            }
         }
         var a = new A(10,20);
         console.log( a.show() );//(10,20)
         ///--------------------------------------
         class B extends A{
            constructor(x,y,color){
                super(x,y);//调用父类constructor（x,y）
                this.color =  color;
            }
            toString(){
                return this.color+' '+super.show();

            }
         }
        var b = new B(30,40,'red');
        console.log( b.show() );//(30,40)
        console.log( b.toString() );//red (30,40)
        console.log(b);//B {x: 30, y: 40, color: "red"}
```

四、instanceof

```
class A{}
        class B extends A{
            constructor(x){
                super();
                this.x = x;
            }
        }
        var obj = new B(10);
        console.log( obj instanceof B );//true
        console.log( obj instanceof A );//true
```

实例对象obj同时是A和B两个类的实例，这与 ES5 的行为完全一致

---

(5) `static` 静态方法

> 父类的静态方法，也会被子类继承。

```
      class A{
             static show(){
                 alert('static')
             }
         }
         class B extends A{
         }

         B.show();//static
```

四、`Object.getPrototypeOf()`

> `Object.getPrototypeOf` 方法可以用来从子类上 `获取父类` 。

```
class A{
              show(){
                 console.log('ok');
              }
         }
         class B extends A{
            static hello(){
             super.prototype.show();
            }

         }
         console.log( Object.getPrototypeOf(B) === A );//true
```

因此，可以使用这个方法判断，`一个类是否继承了另一个类`

五、Class的`Prototype`和`__proto__`

> 每一个对象都有`__proto__`属性，指向对应的构造函数的prototype属性。
> Class 作为构造函数的语法糖，同时有prototype属性和`__proto__`属性，因此同时存在两条继承链。

(1) 子类的`__proto__`属性，表示构造函数的继承，总是指向父类
(2)子类prototype属性的`__proto__`属性，表示方法的继承，总是指向父类的prototype属性

```
class A{}
class B extends A{}
console.log( B.__proto__ === A );//true
console.log(B.prototype.__proto__ === A.prototype );//true

```

类的继承是按照下面的模式实现的

```
class A{}
        class B{}
        Object.setPrototypeOf(B,A);
        Object.setPrototypeOf(B.prototype,A.prototype);
```

六、`extends`的继承目标

extends关键字后面可以跟多种类型的值。

```
class B extends A{}
```

> (1)
> 上面代码的A，只要是一个有prototype属性的函数，就能被B继承。由于函数都有prototype属性（除了Function.prototype函数），因此A可以是`任意函数`。

```
function fn(x,y){
            this.x = x;
            this.y = y;

        }
        class A extends fn{
            constructor(x,y,age){
                super(x,y);
                this.age = age;
            }
        }

        var a = new A(10,20,30);
        console.log(a)
```

(2)子类继承原生构造函数
原生构造函数是指语言内置的构造函数
`Boolean()`
`Number()`
`String()`
`Array()`
`Date()`
`Function()`
`RegExp()`
`Error()`
`Object()`

```
class B extends Object{}
       var obj = {age:20};
       // var r = Object.defineProperty(obj,'index',{
       //  value:888
       // })
       var r = B.defineProperty(obj,'index',{
        value:888
       })
       console.log(obj);//{age: 20, index: 888}
       console.log( B.__proto__  === Object);//true
       console.log( B.prototype.__proto__  === Object.prototyp
e);//true
```

总结

```
1.      Class 类
2.        一、class 申明
3.          二、class 表达式
4.               --不使用当前类的情况
5.               --立即执行的Class
6.          三、constructor
7.                  --constructor方法默认返回实例对象（即this）
8.          四、Class方法
9.                   --方法前面不要加function,没有逗号
10.                  --
11.         五、static 静态方法
12.         六、传统方法改写成Class
13.         七、Class取值函数(getter)和存值函数(setter)
14.         八、Class不存在变量提升
15.         九、new.target 在构造函数内部中代指构造函数
16.
17.     Class 类继承
18.         一、extends关键字
19.         二、constructor
20.         三、super 关键字
21.              (1)--函数调用--指父类的constructor
22.                  --此是必须的，生成this,否则报错
23.                  --super()调用前使用this报错
24.              (2)--在普通方法中--对象--指父类的prototype
25.              (3)--在静态方法中--对象--指父类
26.
27.              使用super的时候，必须显式指定是作为函数、还是作为对象使用，否则会报错。
28.
29.         四、instanceof
30.                  --子类实例 instanceof 指向子类也指向父类
31.         五、static静态方法
32.                  --父类的静态方法也会被继承
33.         六、Obejct.getprototypeOf()
34.                  --可以获取父类
35.
36.         七、Class的prototype及__proto__
37.         八、extends继承目标
38.                  --任意函数
39.                  --内置构造函数
```