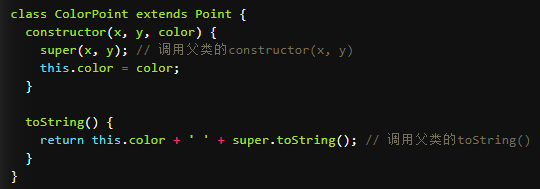
参考摘抄于：<https://es6.ruanyifeng.com/#docs/class-extends>

**1、简介**

class可以通过extends关键字实现继承，这比ES5通过修改原型链实现继承，要清晰方便的多。

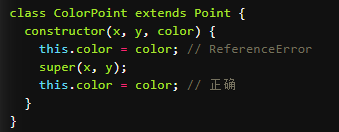


constructor方法和toString方法中，都出现了super关键字，他在这里表示父类的构造函数，用来新建父类的this对象。

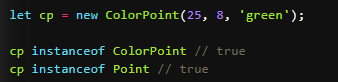
子类必须在构造函数constructor方法中使用super方法，否则新建实例会报错。这是因为子类的this对象，必须通过父类的构造函数完成塑造，得到和父类同样的实例属性和方法，然后再对其进行加工，添加自己的实例属性和方法。如果不调用super方法，子类就得不到this对象。

ES5的继承 ，实际上是先创造子类实例对象的this，然后再将父类的的方法添加到this上面(Parent.apply(this))。ES6的继承机制完全不同，实质是先将父类实例对象的属性和方法，加到this上面（所以必须先调用super方法），然后再用子类的构造函数修改this。

还有一个要注意的地方，在子类的构造函数中，必须调用了super方法后，才能使用this关键字，否则会报错。是因为子类实例的构建基于父类实例，只有super方法才能调用父类实例。



通过子类创建出来的实例对象，同时是子类和父类两个类的实例。



最后，父类的静态方法也会被子类继承。

**2、Object.getPrototypeOf()**

Object.getPrototypeOf方法用于通过子类获取到父类。可以用该方法判断一个类是否继承了另一个类。

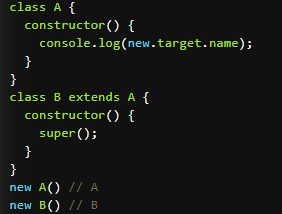


**3、super关键字**

super这个关键字既可以当对象使用也可以当函数使用。

<1>作为函数调用时，代表父类的构造函数，ES6规定，子类的构造函数必须执行一次super方法。

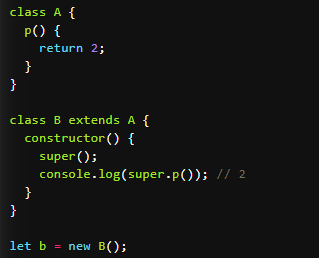
注意，super虽然代表了父类的构造函数，但是返回的是子类的实例，即super内部的this指向的是子类的实例。因此，super()在这里相当于A.prototype.constructor.call(this)。



上面代码中，new.target指向当前执行的函数，new B()返回了B，则说明，子类B中super()执行时，它指向的是B的构造函数。也就说明super()中this指向了B。

super作为函数时，只能出现在子类的构造函数当中，否则会报错。

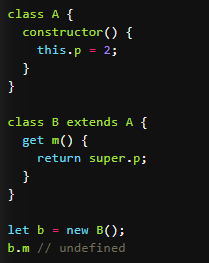
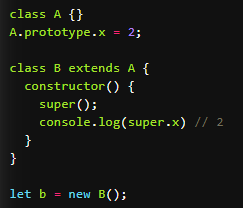
<2>作为对象时，在普通方法中，指向父类的原型对象。在静态方法中，指向父类。



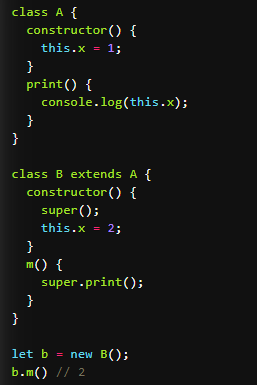
上面代码中，子类B中的super.p()是将super当做一个对象来使用，此时，super指向A.prototype。

需要注意的是，由于super指向的是父类的原型对象，所以定义在父类实例上的属性和方法是无法通过super对象取到的。

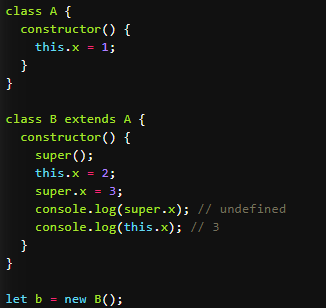
定义在父类的原型上属性，就可以通过super属性拿到。

ES6规定，在子类的普通函数中通过super调用父类的方法时，方法内部的this指向子类的实例。

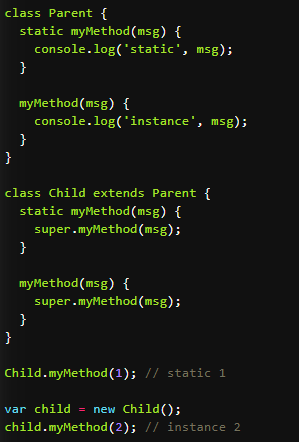
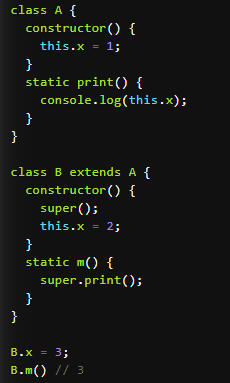


由于this指向子类实例，所以如果通过super对某个属性赋值，这个属性就会添加到子类的实例上。



上面代码中，super.x赋值为3，这时等同于对this.x赋值为3。而当读取super.x的时候，读的是A.prototype.x，所以返回undefined。

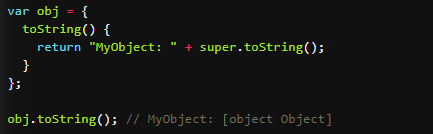
如果super作为对象，用在静态方法中，这时，super指向父类而不是父类的原型。如下图左。

另外，在子类的静态方法中通过super对象调用父类的方法时，this指向当前子类，而不是子类的实例。如上图右。

注意，使用super时，需明确指出是作为函数使用还是对象使用。单独使用会报错。如：（console.log(super) //报错）

最后，由于对象总是继承自其他对象，所以可以在任意对象中使用super关键字。

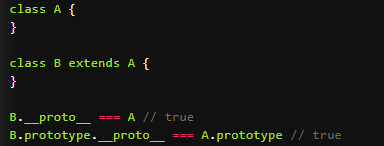


**4、类的prototype属性和\_\_proto\_\_属性**

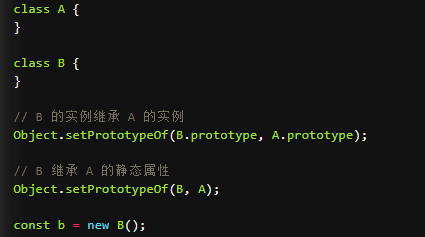
class作为构造函数的语法糖，同时拥有prototype和\_\_proto\_\_两个属性，因此同时存在两条继承链。

<1>子类的\_\_proto\_\_属性，表示构造函数的继承，总是指向父类。

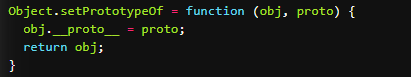
<2>子类的prototype属性的\_\_proto\_\_属性，表示方法的继承，总是指向父类的prototype属性。



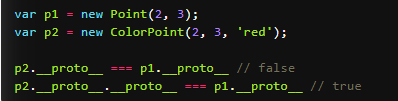
出现上述结果是因为，类的继承是按下面的模式实现的。

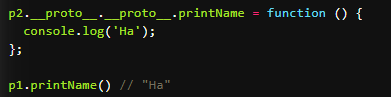


而Object.setPrototypeOf方法的实现就是将第二个参数的值赋值给第一个参数的\_\_proto\_\_属性。



子类实例的\_\_proto\_\_属性的\_\_proto\_\_属性指向父类实例的\_\_proto\_\_属性，也就是子类原型的原型就是父类的原型。

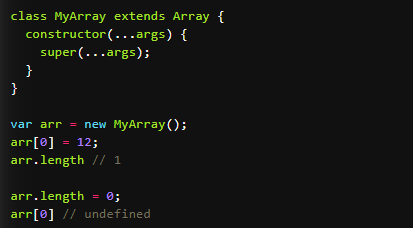




**5、原生构造函数的继承**

由于ES5是先新建子类实例对象的this，再将父类的属性添加到子类上，但是父类的内部属性是无法获取的，所以无法继承原生对象的构造函数。

ES6允许继承原生构造函数来定义子类，因为ES6是先建父类的实例对象this，再用子类的构造函数修饰this，使得父类构造函数的所有行为都可以继承。



注意：ES6继承Object的子类，无法通过super方法像父类传参，这是因为ES6改变了Object构造函数的行为，一旦发现Object方法不是通过new Object()调用时，ES6规定Object构造函数会忽略参数。