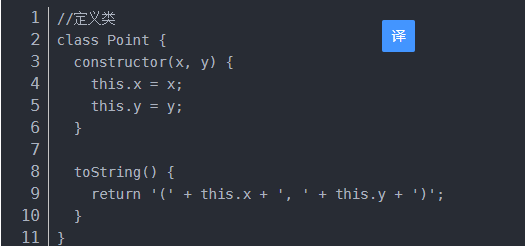
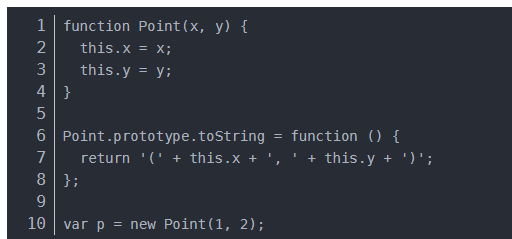
**constructor 方法**

ES6 的 class 属于一种“语法糖”，所以只是写法更加优雅，更加像面对对象的编程，其思想和 ES5 是一致的。  
类和模块的内部，默认就是严格模式，所以不需要使用use strict指定运行模式。



等同于



其中 constructor 方法是类的构造函数，是一个默认方法，通过 new 命令创建对象实例时，class必须要用new命令创建，不然会报错(TypeError: Class constructor Foo cannot be invoked without ‘new’)，自动调用该方法。一个类必须有 constructor 方法，如果没有显式定义，一个默认的 consructor 方法会被默认添加。所以即使你没有添加构造函数，也是会有一个默认的构造函数的。一般 constructor 方法返回实例对象 this ，但是也可以指定 constructor 方法返回一个全新的对象，让返回的实例对象不是该类的实例。

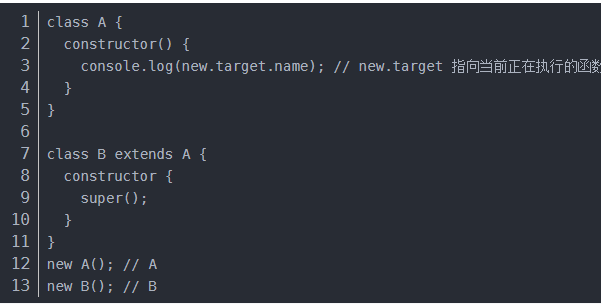
class继承中，子类必须在constructor方法中调用super方法，否则新建实例时会报错。这是因为子类自己的this对象，必须先通过父类的构造函数完成塑造，得到与父类同样的实例属性和方法，然后再对其进行加工，加上子类自己的实例属性和方法。如果不调用super方法，子类就得不到this对象。

super 这个关键字，既可以当做函数使用，也可以当做对象使用。这两种情况下，它的用法完全不用。

## 1. 当做函数使用



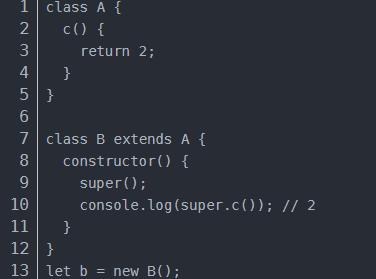
注：在 constructor 中必须调用 super 方法，因为子类没有自己的 this 对象，而是继承父类的 this 对象，然后对其进行加工,而 super 就代表了父类的构造函数。super 虽然代表了父类 A 的构造函数，但是返回的是子类 B 的实例，即 super 内部的 this 指的是 B，因此 super() 在这里相当于 ```A.prototype.constructor.call(this, props)``。



可以看到，在 super() 执行时，它指向的是 子类 B 的构造函数，而不是父类 A 的构造函数。也就是说，super() 内部的 this 指向的是 B。

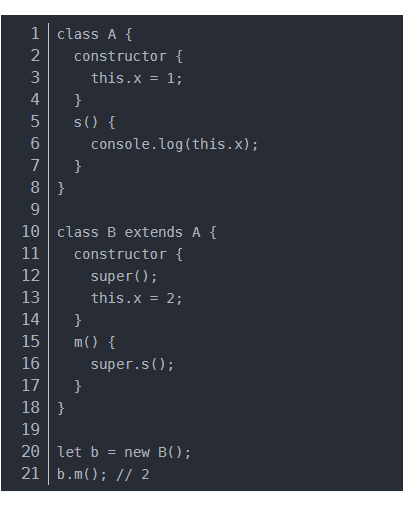
## 当做对象使用

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



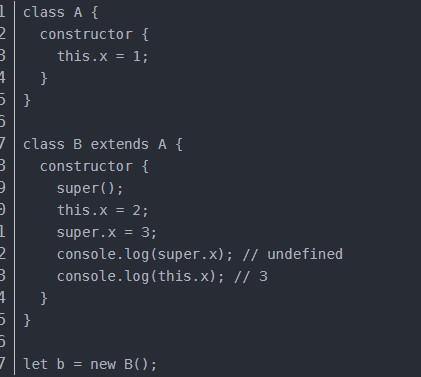
上面代码中，子类 B 当中的 super.c()，就是将 super 当作一个对象使用。这时，super 在普通方法之中，指向 A.prototype，所以 super.c() 就相当于 A.prototype.c()。

通过 super 调用父类的方法时，super 会绑定子类的 this。



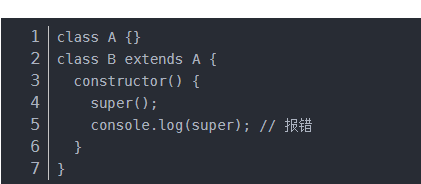
上面代码中，super.s() 虽然调用的是 A.prototytpe.s()，但是 A.prototytpe.s()会绑定子类 B 的 this，导致输出的是 2，而不是 1。也就是说，实际上执行的是 super.s.call(this)。

由于绑定子类的 this，所以如果通过 super 对某个属性赋值，这时 super 就是 this，赋值的属性会变成子类实例的属性。



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，调用的是 A.prototype.x，但并没有 x 方法，所以返回 undefined。

## 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。



上面代码中，console.log(super); 的当中的 super，无法看出是作为函数使用，还是作为对象使用，所以 JavaScript 引擎解析代码的时候就会报错。这是，如果能清晰的表明 super 的数据类型，就不会报错。

最后，由于对象总是继承其他对象的，所以可以在任意一个对象中，使用 super 关键字。