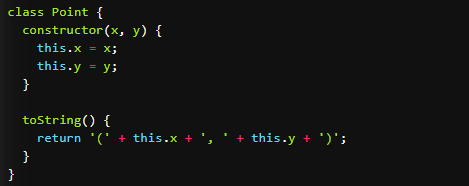
参考摘抄于：<https://es6.ruanyifeng.com/#docs/class>

**1、类的由来**

在JS中，传统生成实例对象的方法就是通过构造函数。ES6为了更接近传统语法，引入了类（class），作为对象的模型。通过class关键字，可以定义类。

**2、类的定义**



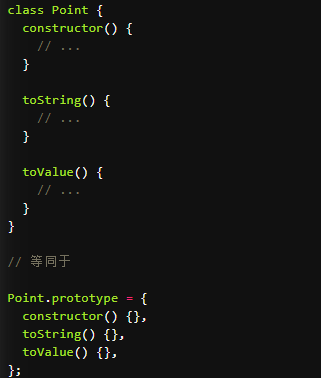
上面代码定义了一个“类”，里面的constructor就是构造方法，而this关键字就代表实例对象。注意上例定义toString方法时，不需要写function关键字，直接将函数定义进去就可以了。另外，方法与方法之间不需要加逗号，否则会报错。

**3、类的特点**

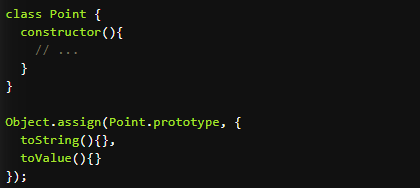


上面代码说明类的数据类型就是函数，类本身就指向构造函数。使用的时候也是直接对类使用new命令，跟构造函数的用法一致。

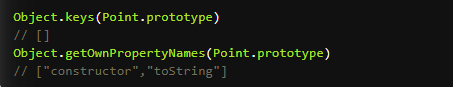
构造函数的prototype属性在class上继续存在，且类里面的所有方法都定义在类的prototype上。因此，在类的实例上调用方法，其实就是调用原型上的方法。



由于类的方法都定义在prototype上，所以类的新方法可以添加在prototype对象上面，故可利用Object.assign()可以很方便的一次性往类上添加多个方法。



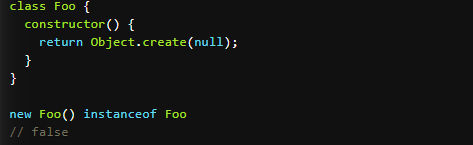
另外，类内部定义的方法都是不可枚举的。



上例中toString方法是类内部定义的，是不可枚举的。这点和ES5行为是不同的。如果使用构造函数定义，则toString方法是可以被枚举的。

constructor方法是类的默认方法，如果没有显式定义，一个空的constructor方法会被默认添加。使用new命令生成对象实例时，会自动调用该方法。

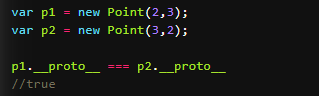
constructor方法默认返回实例对象（即this），也可以返回其他对象。返回其他对象会导致实例对象不是类的实例。如下图所示：



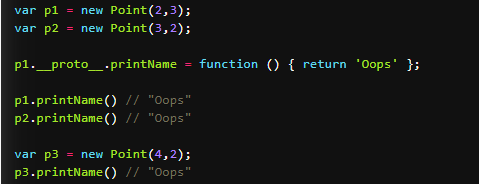
类必须使用new调用，直接调用会报错。而普通构造函数可以直接调用，这是二者的主要区别。

**4、类的实例**

与ES5一样，实例的属性除非显示的定义在其本身（即this对象上），否则都是定义在原型上（即定义在class上）。与ES5一样，类的所有实例共享一个原型对象。

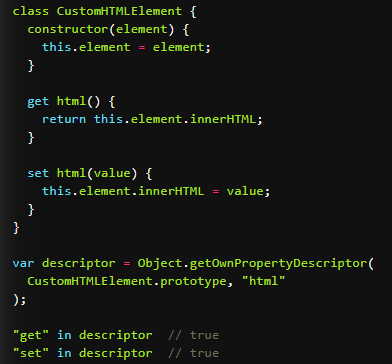


上例中，p1和p2都是Point的实例，他们的原型都是Point.prototype。故他们的\_\_proto\_\_相等。这也意味着，可以通过\_\_proto\_\_向“类”添加方法。\_\_proto\_\_不是语言本身的特性，是各大浏览器厂商具体实现时添加的私有属性。虽然目前许多现代浏览器厂商JS引擎中都提供了这个锅私有属性，但依旧不建议在生产中直接使用该属性，避免对环境产生依赖。生产环境中，应该使用Object.getPrototypeOf()方法来获取实例对象的原型，然后再来为原型添加方法和属性。



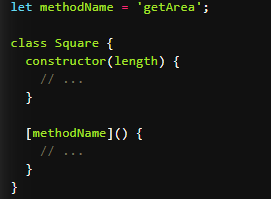
通过一个类生成两个实例对象，给任意一个实例对象的原型添加的方法或属性，另一个实例对象也可以通过原型对象获取到。

**5、取值函数（getter）和存值函数（setter）**



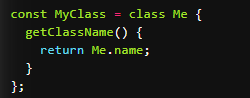
与ES5一样，在“类”的内部可以使用get和set关键字，对某个属性设置取值函数和存值函数，拦截该属性的存取行为。存值函数和取值函数是设置在属性的Descriptor对象上的。

**6、属性表达式**

类的属性名可以采用属性表达式。使用方括号语法。

**7、类表达式**

与函数一样，类也可以使用表达式的形式定义。



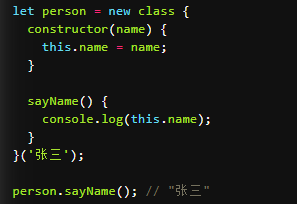


上面代码使用表达式定义了一个类。需要注意的是，这个类的名字是Me，但是Me只能在类的内部调用，指代当前类。在类的外部，只能使用MyClass。

如果内部的没有用到自己，则Me可以省略。表达式改为：



采用class表达式，可以写出立即执行的class。



上面代码中，person是一个立即执行类的实例。

**8、注意点**

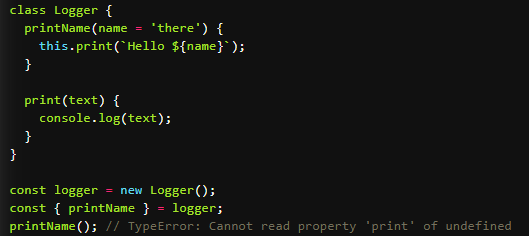
(1)严格模式：在类和模块的内部，默认就是严格模式。

(2)类不存在变量提升：必须先定义，后调用

(3)name属性：和ES5的构造函数一样，类的name就为声明时，紧跟在class关键字后面的字符。

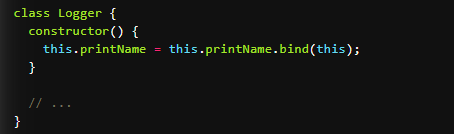
(4)Generator方法：类的内部可以声明Generator函数

(5)this的指向：类的方法内部如果含有this，该this默认指向类的实例。但是，使用时需非常小心，一旦单独使用该方法，很可能报错。

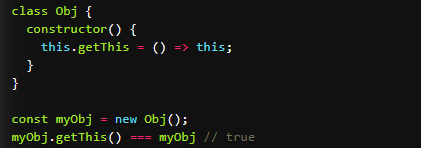


上例中，printName方法中的this，默认指向类Logger的实例logger。但是如果将这个方法单独提取出来使用，this会指向当前代码所运行的环境（由于class内部是严格模式，所以this其实指向的是undefined），所以会报错。

注意：一个比较简单的解决方法是在构造函数中绑定this。



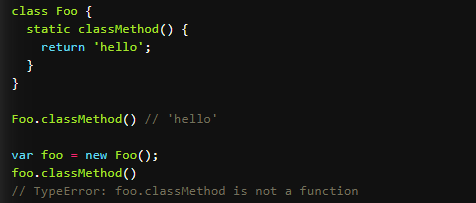
还有就是使用箭头函数。



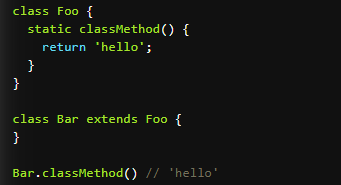
由于箭头函数的this总是指向定义时所在的对象。上例中箭头函数为了构造函数的内部，他的定义生效的时候是构造函数代码执行的时候，即生成实例对象的时候。这时，箭头函数的运行环境肯定是实例对象，故箭头函数的this指向实例对象。

**9、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对象继承。如果在一个方法上添加static关键字，就表示该方法不会被实例对象所继承，而是通过类来直接调用，称为类的静态方法。



在类内部，方法名前有static关键字的表明该方法是一个静态方法，可以直接在类上调用。而不能在类的实例上调用（调用会报错）。如果静态方法里存在this，该this指代的是类而不是类的实例。并且静态方法可以和非静态方法重名。



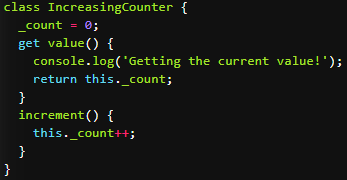
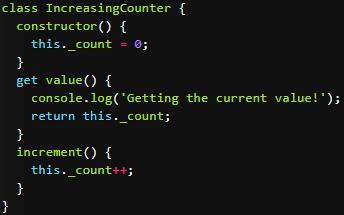
父类的静态方法可以被子类继承。



父类定义的静态方法子类也可以通过super对象在子类内部调用。

**10、实例属性的新写法**

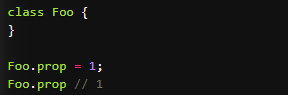
实例属性除了可以定义在类的constructor方法里面的this上，也可以定义在类的最顶层。



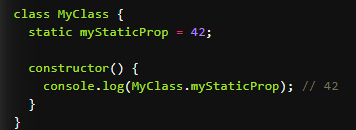
这种新写法的好处就是可以将类的所有实例属性定义在类的顶部，一眼就可以看出来这个类有哪些实例属性。

**11、静态属性**

静态属性是指定义在Class本身的属性，即Class.propName，而不是定义在实例对象(this)上的属性。



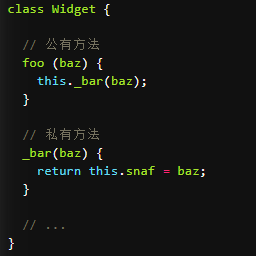
目前，只有这种写法是可行的，因为ES6规定，Class内部只有静态方法，没有静态属性。现在有一个提案提供了类的静态属性，写法是在实例属性前面加上static关键字。



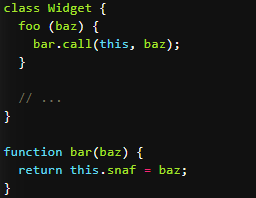
**12、私有方法和私有属性**

私有方法和私有属性，是只能在类的内部访问的方法和属性，外部不能访问。但ES6不提供，只能通过变通方法模拟实现。

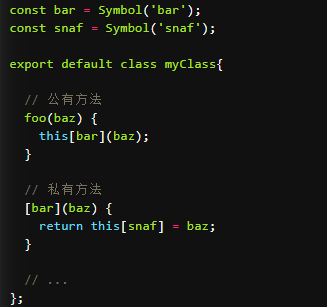
一种方法是通过命名区分，私有方法方法名前加”\_”，标识该方法属于类的私有方法。但是这种方法是不保险的，在类的外部还是可以调用到该方法。



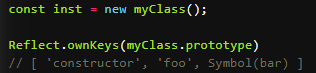
另一种方法是索性将方法移出类，因为类内部的所有方法都是对外部可见的。



还有一种方法是利用Symbol的唯一性，将私有方法命名为一个Symbol值。

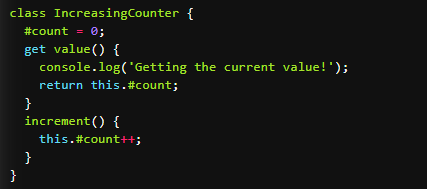


上面代码中bar和snaf都是Symbol值，一般情况下无法获取到他们；然而Reflect.ownKeys()依然可以拿到他们。

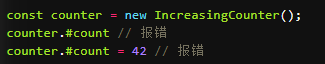


**13、私有属性的提案**

目前，有一个提案，为class添加了私有属性。方法是在属性名之前，使用#表示。

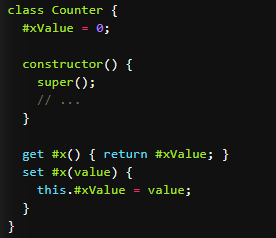


上面代码中，#count就是私有属性，只能在类内部使用，在类外部通过实例访问时，会报错。



私有方法也可以通过在方法名前添加”#”来作为标识。

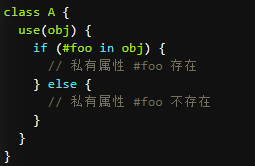
另外，私有属性也可以设置getter和setter方法。



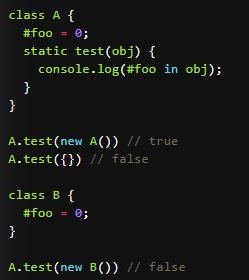
私有属性和私有方法前也可以添加static关键字，表示该属性或方法是一个静态的私有属性或方法。

**14、in运算符**

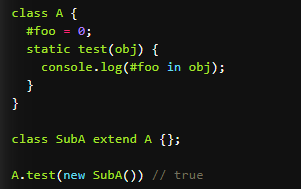
V8引擎改进了in运算符，是它可以用来判断私有属性。



注意，判断私有属性的时候，in只能用在定义该私有属性的内部。



子类从父类继承的私有属性，也可以用in运算符判断。

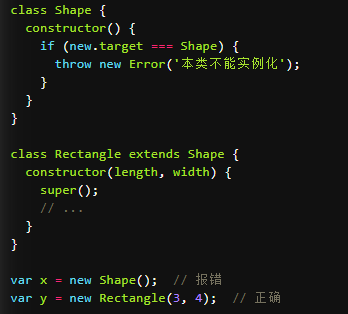


注意，in运算符对于Object.create或Object.setPrototypeOf形成的继承是无效的，因为这种继承不会传递私有属性。

**15、new.target属性**

new是从构造函数生成实例对象的命令。ES6为new命令引入一个new.target属性，该属性一般用于构造函数之中，返回new命令作用于的构造函数。如果构造函数不是通过new命令或Reflect.contruct()调用的，new.target会返回undefined。

需要注意的是，子类继承父类，new.target属性返回子类。利用这个特点可以写出不能直接独立使用、必须通过继承后才能使用的类。



注意，在函数外部使用new.target会报错。