es5:

```
let Animal = function(type){
    this.type = type;
    this.eat1= function(){
        console.log("i am eat1")
    }
}
Animal.prototype.eat = function(){
    console.log("i am eat food")
}
let dog = function(){
    Animal.call(this,'dog')
    //  当前this指向内存中的一个位置，因为调用父类时也是引用的内存中的这个位置，所以当执行this.xxx = yyy时，当前这个this相当于也执行了相同的操作，所以可以实现继承父类的属性。
}
```

dog.prototype = Animal.prototype;（两者原型指向同一位置，实现了继承，但是改动子类原型的方法会影响父类原型的方法）

dog.prototype = new Animal();（子类原型指向父类的实例，改变子类的原型里的方法只会影响父类的一个实例，不会影响到父类的原型）

```
dog.prototype.eat = function(){
    console.log("i am eat shit")
}
var an = new Animal();
var reddog = new dog();
console.log(Animal)
console.log(reddog)
reddog.eat()
an.eat()
```

es6:

```
class reddog extends Animal{
```

```
        constructor(type){
            super(type)// 如果没有自己的属性可以不写，如果有要写入super方法，执行父类的构造函数
            this.color = 'red'
        }
        eat (){
            Animal.walk()
            console.log("i am eat shit")
        }
    }
```