1、借助构造函数实现继承（原理是在子类中改变父级this的指向，缺点是只能部分继承，不能继承父类原型对象上的方法）；
2、借助原型链继承（缺点是原型链上的对象是共用的，改变某个对象的属性，原型链上的其他对象属性也会改变）；
3、组合方式（缺点是父级的构造函数执行了两次并把父类的constructor也继承了）；
4、组合继承优化1（缺点是把父类的constructor也继承了）；
5、组合继承优化2（原理是通过Object.create方法创建一个中间对象，参数是该对象的原型对象，然后把子类的构造函数赋值为该子类）。

```
/**
 * 借助构造函数实现继承
 */
function Parent1 () {
    this.name = 'parent1';
}
Parent1.prototype.say = function () {

};
function Child1 () {
    Parent1.call(this);
    this.type = 'child1';
}
console.log(new Child1(), new Child1().say());

/**
 * 借助原型链实现继承
 */
function Parent2 () {
    this.name = 'parent2';
    this.play = [1, 2, 3];
}
function Child2 () {
    this.type = 'child2';
}
Child2.prototype = new Parent2();
```

```
var s1 = new Child2();
var s2 = new Child2();
console.log(s1.play, s2.play);
s1.play.push(4);
/**
 * 组合方式
 */
function Parent3 () {
    this.name = 'parent3';
    this.play = [1, 2, 3];
}
function Child3 () {
    Parent3.call(this);
    this.type = 'child3';
}
Child3.prototype = new Parent3();
var s3 = new Child3();
var s4 = new Child3();
s3.play.push(4);
console.log(s3.play, s4.play);

/**
 * 组合继承的优化1
 * @type {String}
 */
function Parent4 () {
    this.name = 'parent4';
    this.play = [1, 2, 3];
}
function Child4 () {
    Parent4.call(this);
    this.type = 'child4';
}
Child4.prototype = Parent4.prototype;
```

```
var s5 = new Child4();
var s6 = new Child4();
console.log(s5, s6);

console.log(s5 instanceof Child4, s5 instanceof Parent4);
console.log(s5.constructor);

/**
 * 组合继承的优化2
 */
function Parent5 () {
    this.name = 'parent5';
    this.play = [1, 2, 3];
}
function Child5 () {
    Parent5.call(this);
    this.type = 'child5';
}
Child5.prototype = Object.create(Parent5.prototype);
```