几种继承方式:

**（1）类的继承——原型链继承**

function Cat(){ }

Cat.prototype = new Animal();

Cat.prototype.name = 'cat';

介绍：在这里我们可以看到new了一个空对象,这个空对象指向Animal并且Cat.prototype指向了这个空对象，这种就是基于原型链的继承。

实例可继承的属性有：

父类构造函数属性

父类原型的属性。（新实例不会继承父类实例的属性！）

实例的构造函数的属性

缺点：1、新实例无法向父类构造函数传参。

　　　2、继承单一。

　　　3、所有新实例都会共享父类实例的属性。（原型上的属性是共享的，一个实例修改了原型属性，另一个实例的原型属性也会被修改！）

**（2）构造继承：**使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

**特点：**1、只继承了父类构造函数的属性，没有继承父类原型的属性。

　　　2、解决了原型链继承缺点1、2、3。

　　　3、可以继承多个构造函数属性（call多个）。

　　　4、在子实例中可向父实例传参。

缺点：1、只能继承父类构造函数的属性。

　　　2、无法实现构造函数的复用。（每次使用都要重新调用）

　　　3、每个新实例都有父类构造函数的副本，臃肿。

1. **组合继承(es6之前最常用)：相当于构造继承和原型链继承的组合体。**

//父类

function Person(name,age) {

this.name = name;

this.age = age;

}

Person.prototype.show = function() {

console.log(this.name,this.age);

}

//子类继承

function Worker(name,age,job) {

Person.call(this,name,age);

this.job = job;

}

Worker.prototype = new Person();

Worker.prototype.constructor = Worker;

Worker.prototype.showJob = function() {

console.log(this.job);

}

特点：1、可以继承父类原型上的属性，可以传参，可复用。

　　　2、每个新实例引入的构造函数属性是私有的。

缺点：调用了两次父类构造函数（耗内存），子类的构造函数会代替原型上的那个父类构造函数。

**（4）实例继承和拷贝继承**

实例继承：为父类实例添加新特性，作为子类实例返回

拷贝继承：拷贝父类元素上的属性和方法

上述两个实用性不强，不一一举例。

**（5）寄生组合继承：**通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性

function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

(function(){

// 创建一个没有实例方法的类

var Super = function(){};

Super.prototype = Animal.prototype;

//将实例作为子类的原型

Cat.prototype = new Super();

})();

重点：就是给原型式继承外面套了个壳子。

优点：没有创建自定义类型，因为只是套了个壳子返回对象（这个），这个函数顺理成章就成了创建的新对象。

缺点：没用到原型，无法复用。

**es6继承的写法:**

//父类

class Person{

constructor(name,age) {

this.name = name;

this.age = age;

}

show() {

console.log(this.name,this.age);

}

}

//子类继承

class Worker extends Person {

constructor(name,age,job) {

super(name,age);

this.job = job;

}

showJob() {

console.log(this.job);

}

}

**创建对象的方式:**

①通过调用系统的构造函数创建对象

var obj1=new Object();

②自定义构造函数创建对象

var obj2=new 自定义构造函数();

③字面量的方式创建对象

var obj3={};

④工厂模式创建对象

把创建对象的代码封装在一个函数中,可以实现一次性创建多个对象

function createObject(){

var obj=new Object(); //创建对象

obj.name=”小芳”; //没有重新定义时,为默认值

obj.age=10;

obj.sayHi=function(){

console.log(“hai小健文,我叫”+this.name+”我今年”+this.age);

};

return obj; //一定要返回这个对象

}