继承

1.原型继承

\*  就是子对象自动拥有父对象的属性和方法， 继承可以提高代码的复用性。

\*  JS里的继承主要依靠是的原型链。让原型对象（每一个构造函数都有一个原型对象）的值，等于另一个类型的实例，即实现了继承；另外一个类型的原型再指向第三个类型的实例，以此类推，也就形成了一个原型链

    function Animal(newName,newAge){

        this.name = newName;

        this.age = newAge;

    }

    Animal.prototype.eat = function(str){

        console.log(this.name + "吃" + str);

    }

    function Person(newId){

        this.id = newId;

    }

    Person.prototype = new Animal("老王",18);

    let p1 = new Person("007");

    console.log(p1.name,p1.age,p1.id);

    p1.name = "小明";

    p1.eat("米饭");

    let p2 = new Person("008");

    console.log(p2.name,p2.age,p2.id);

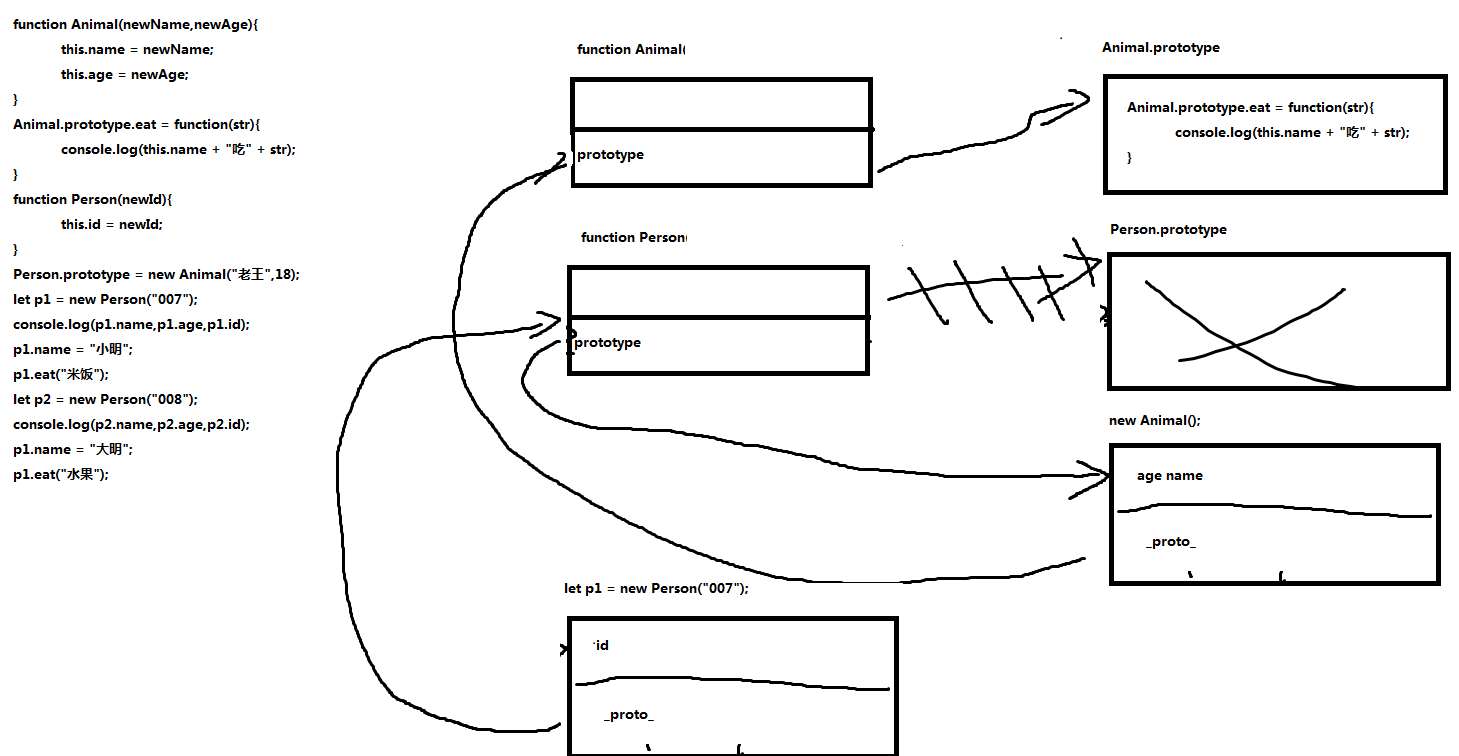
    p1.name = "大明";

    p1.eat("水果");

\* 使用instanceof操作符检测对象类型

    console.log(p1 instanceof Person);

    console.log(p2 instanceof Person);



2.原型链继承：

让子对象的原型指向父对象的实例，父对象的原型指向爷爷对象的实例，依次类推，就形成了原型链

    function Animal(newAge){

        this.age = newAge;

    }

    Animal.prototype.eat = function(){

        console.log("Animal.eat");

    }

    function Person(newId){

        this.id = newId;

    }

    Person.prototype = new Animal(18);

    Person.prototype.study = function(){

        console.log("Person.study");

    }

    function Pupli(newTall){

        this.tall = newTall;

    }

    Pupli.prototype = new Person("007");

    Pupli.prototype.play = function(){

        console.log("Pupli.play");

    }

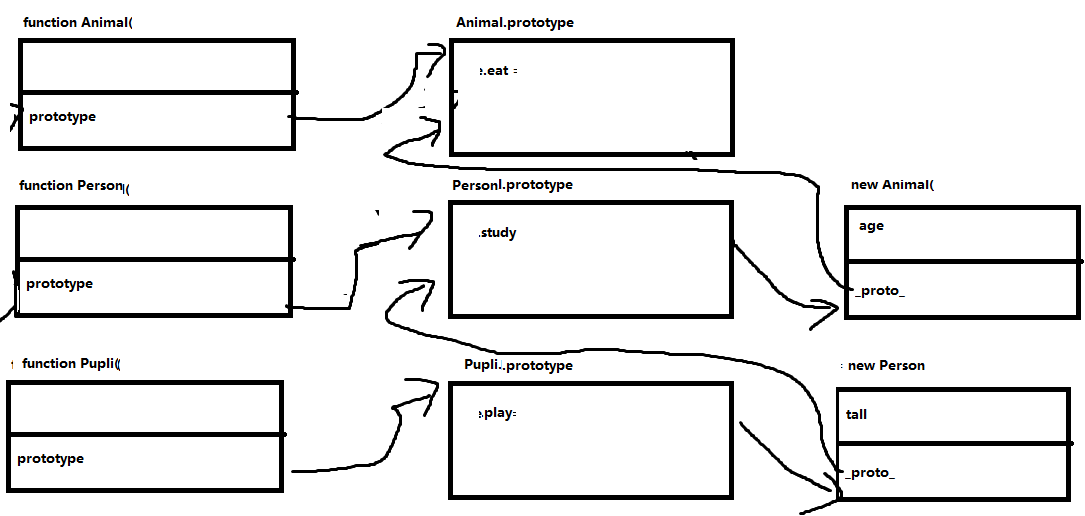
    let pu = new Pupli(140);

    console.log(pu.age,pu.id,pu.tall);

    pu.eat();

    pu.study();

    pu.play();

**让子对象的原型指向父对象的实例，父对象的原型指向爷爷对象的实例，依次类推，就形成了原型链**

3.原型继承中的注意事项：

  1. 先定义原型继承关系，再添加子类的自定义方法或属性(原型的属性，即共享的属性和方法要在原型继承关系确立后，再定义)。

  2. 利用原型链继承，给子类添加原型方法时，不可以重写prototype

    function Animal(newAge){

        this.age = newAge;

    }

    function Person(newId){

        this.id = newId;

    }

    Person.prototype.eat = function(){

        console.log("Person eat");

    }

    Person.prototype = new Animal(15);

    let p = new Person("007");

    console.log(p.id);

    p.eat();

缺点：

  1. 被继承的类型(父类)里包括引用类型的属性的时候，它会被所有实例共享其值

  2. 创建子类型的实例时，没法传参给被继承类型。

4.深拷贝浅拷贝

    class Student{

        constructor(newName,newId) {

            this.name = newName;

            this.id = newId;

        }

        clone(){

            return new Student(this.name,this.id);

        }

        showValue(){

            console.log(this.name + this.id);

        }

    }

    let s1 = new Student("小明","9527");

    let s2 = s1.clone();

    s2.name = "老王";

    s1.showValue();

   function Birthday(y,m,d){

        this.y = y;

        this.m = m;

        this.d = d;

    }

    function Animal(newBir){ //Birthday与Animal关联

        this.bir = newBir;

    }

    function Person(){

    }

    Person.prototype = new Animal(new Birthday(1,2,3));

    let p1 = new Person();

    console.log(p1.bir.y,p1.bir.m,p1.bir.d);

    let p2 = p1;

    console.log(p2.bir.y,p2.bir.m,p2.bir.d);

    p1.bir.y = 666;

    p1.bir.m = 666;

    p1.bir.d = 666;

    console.log(p1.bir.y,p1.bir.m,p1.bir.d);

    console.log(p2.bir.y,p2.bir.m,p2.bir.d);

    console.log(p1.bir.y,p1.bir.m,p1.bir.d);

在父类传入的参数有引用元素时，则需要考虑深拷贝和浅拷贝的问题。

浅拷贝：传参时值只传递了引用空间的地址

深拷贝：开辟空间，且赋值

function Birthday(y,m,d){

        this.y = y;

        this.m = m;

        this.d = d;

    }

    function Animal(newBir){

        this.bir = newBir;

    }

    function Person(){

    }

    Person.prototype = new Animal(new Birthday(1,2,3));

    Person.prototype.clone = function(){

        let p = new Person();

        //引用类型

        p.bir = new Birthday(this.bir.y,this.bir.m,this.bir.d);

        return p;

    }

    let p1 = new Person();

    console.log(p1.bir.y,p1.bir.m,p1.bir.d);

    let p2 = p1.clone();

    console.log(p2.bir.y,p2.bir.m,p2.bir.d);

    p1.bir.y = 666;

    p1.bir.m = 666;

    p1.bir.d = 666;

    console.log(p1.bir.y,p1.bir.m,p1.bir.d);

    console.log(p2.bir.y,p2.bir.m,p2.bir.d);

5.call和apply的继承

这两个方法，已经讲过，是调用函数的，当然也能调用构造函数

    function Person(newId,newName){

        this.id = newId;

        this.name = newName;

    }

    function Student(newId,newName,newScore){

        //借用构造方法

        Person.call(this,newId,newName);

        this.score = newScore;

    }

    let student = new Student("007","老王",99);

    console.log(student.id,student.name,student.score);

但是这样的“继承”的方式也有弊端：

        1.无法继承父类原型上的属性和方法

    2.单独使用这种借用的模式，所有要继承的属性和方法都要在父类型的构造函数里定义，

    特别是实例共享的属性和方法也写在构造函数里，那么这样会浪费内存。所以，很少很少单独使用

        //如:

    function Person(newId,newName){

        this.id = newId;

        this.name = newName;

    }

    Person.prototype.eat = function(){

        console.log("Person eat");

    }

    function Student(newId,newName,newScore){

        //借用构造方法

        Person.call(this,newId,newName);

        this.score = newScore;

    }

    let student = new Student("007","老王",99);

    console.log(student.id,student.name,student.score);

    student.eat();//报错

    必须改成以下形式：

    function Person(newId,newName){

        this.id = newId;

        this.name = newName;

        this.eat = function(){

            console.log("Person eat");

        }

    }

    function Student(newId,newName,newScore){

        //借用构造方法

        Person.call(this,newId,newName);

        this.score = newScore;

    }

    let student = new Student("007","老王",99);

    console.log(student.id,student.name,student.score);

    student.eat();

组合继承:

    结合前两种方式：原型链式继承和Call()/Apply()方式继承，我们就能解决前面提出的那些问题。

    利用原型链继承共有的属性和方法，利用Call/Apply来初始化自己的但是和父类型同名的属性或方法。

    function Person(newId,newName){

        this.id = newId;

        this.name = newName;

    }

    Person.prototype.eat = function(){

        console.log("Person eat");

    }

    function Student(newId,newName,newScore){

        Person.call(this,newId,newName);

        this.score = newScore;

    }

    Student.prototype = new Person();

    let student = new Student("007","老王",99);

    console.log(student.id,student.name,student.score);

    student.eat();

6.ES6的继承

    class Person{

        constructor(newId,newName) {

            this.id = newId;

            this.name = newName;

        }

        showValue(){

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

        }

    }

    class Student extends Person{

        constructor(newId,newName,newScore){

            super(newId,newName);

            this.score = newScore;

        }

        eat(){

            console.log("student eat");

        }

        showValue(){

            super.showValue();

            console.log(s.score);

        }

    }

    let s = new Student(1,"老王",99);

    s.showValue();

    s.eat();