### 继承

js的六种继承：原型链，借用构造函数，组合继承，原型式继承，寄生式继承，寄生组合式继承。

分清通过原型链继承和构造函数继承的区别方式。

定义父类：

    function father(name,age){

      this.name = name

      this.age = age

      this.sayHello = function(){

        console.log('hello son');

      }

    }

    father.prototype.sex = 'man'

    father.prototype.sayHello2 = function(){

      console.log('hello my son');

    }

#### 1.原型链继承

通过原型链，把父类的实例对象放在子类的原型上

优点：能继承父类构造函数内部的方法，父类构造函数原型上的方法和属性

缺点：继承的父类构造函数原型上的属性是共享的，一旦修改，全部修改。

    // 原型链继承

    function Son(){

    }

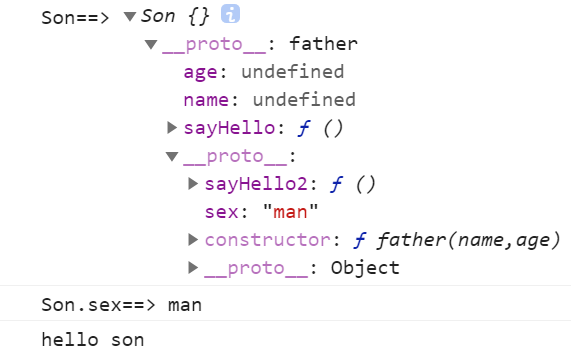
  // 把父类的实例放到子类的原型上

    Son.prototype = new father()

    console.log("Son==>", new Son());

    console.log("Son.sex==>", new Son().sex);

    new Son().sayHello();



#### 2.使用构造函数继承（经典继承）

在子类型构造函数的内部使用call（）或apply（）引入父类构造函数。

优点：（1）子类型构造函数可以向父类构造函数传递参数。

（2）解决了引用值共享问题，每个实例都拥有自己的属性。

缺点：（1）多执行一次call（）

（2）父类上的原型不能被继承

    // 构造函数继承

    function Son(name,age){

      father.call(this,name,age)

    }

    const son = new Son('小明',18)

    console.log("son==>", son)

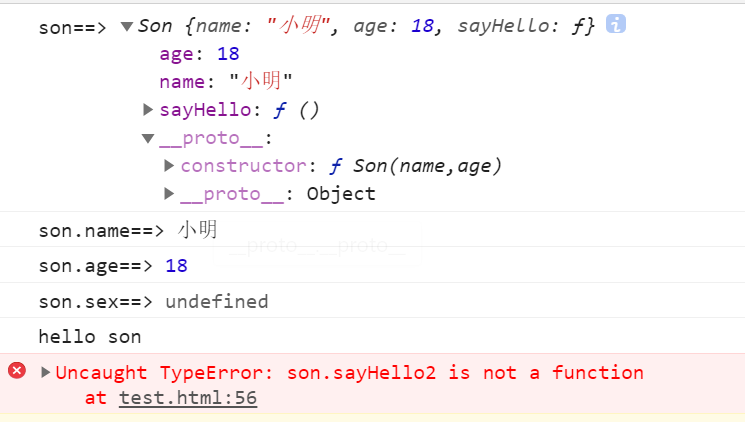
    console.log("son.name==>", son.name)

    console.log("son.age==>", son.age)

    console.log("son.sex==>", son.sex)

son.sayHello()

son.sayHello2()



#### 3.组合继承（伪经典继承）

将原型链和借用构造函数的技术组合到一起，发挥二者的优势，使用原型链实现对原型属性和方法的继承，使用构造函数实现对实例属性的继承。

优点： （1）父类上的原型可以被继承。

（2）解决了引用值共享的问题。

（3）子类可以向父类传递参数。

缺点：多执行一次call（）。

    // 组合式继承

    function Son(name,age){

      father.call(this,name,age)

    }

    Son.prototype = new father()

    const son = new Son('小明',18)

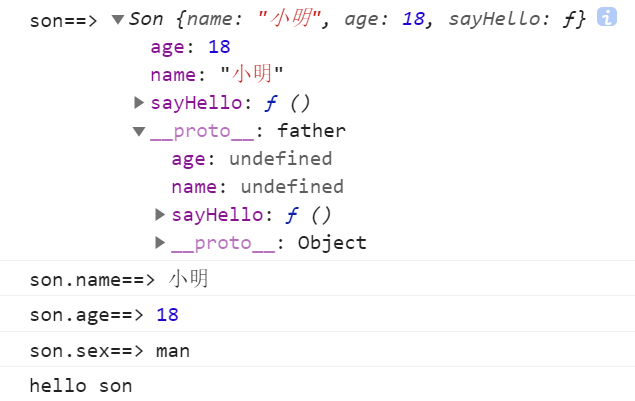
    console.log("son==>", son);

    console.log("son.name==>", son.name)

    console.log("son.age==>", son.age)

    console.log("son.sex==>", son.sex);

    son.sayHello();



#### 4.原型式继承

利用已有的对象，在里面创建临时性构造函数（空构造函数），然后将传入的对象直接赋值给临时性构造函数的原型。

用一个函数包装一个对象，然后返回这个函数的调用，这个函数就变成了可以随意增添对象的实例或对象。object.creat()就是这个原理。

缺点：（1）继承包含多个引用类型是，所有实例的引用地址是一样的。

（2）无法实现服用。

// 原型继承

    function object(o){

      function F(){}

      F.prototype = o

      return new F()

    }

    var phone = {

      name:'phone',

      color:['red','orange','yellow']

    }

    var iphone = object(phone)

    var huawei  = object(phone)

    iphone.name = '我是苹果'

    console.log(huawei.name);

    iphone.color.push('green')

    console.log(huawei.color);

    console.log('iphone==>',iphone);

    console.log('huawei==>',huawei);



ES5中新增Object.creat()实现了原型式继承，和上面结果一样

ar phone = {

      name:'phone',

      color:['red','orange','yellow']

    }

    var iphone = Object.create(phone)

    var huawei  = Object.create(phone)

    iphone.name = '我是苹果'

    console.log(huawei.name);

    iphone.color.push('green')

    console.log(huawei.color);

    console.log('iphone==>',iphone);

    console.log('huawei==>',huawei);



#### 5.寄生式继承

在原型式继承的基础上，创建一个用于封装继承的函数。没有创建自定义类型，只是给原型式继承套了个壳子。

优点：比原型式继承多了一个增强对象。

缺点：（1）继承包含多个引用类型是，所有实例的引用地址是一样的。

（2）没用到原型无法实现复用。

// 寄生式继承

    function createObject (o) {

      var clone = Object.create(o);         // 先原型继承

      clone.sayHi = function () {    //增强对象

          console.log('hi');

      }

      return clone;                  //返回对象

    }

   var phone = {

      name:'phone',

      color:['red','orange','yellow']

    }

    var iphone = createObject(phone)

    var huawei  = createObject(phone)

    iphone.name = '我是苹果'

    console.log(huawei.name);

    iphone.color.push('green')

    console.log(huawei.color);

    console.log('iphone==>',iphone);

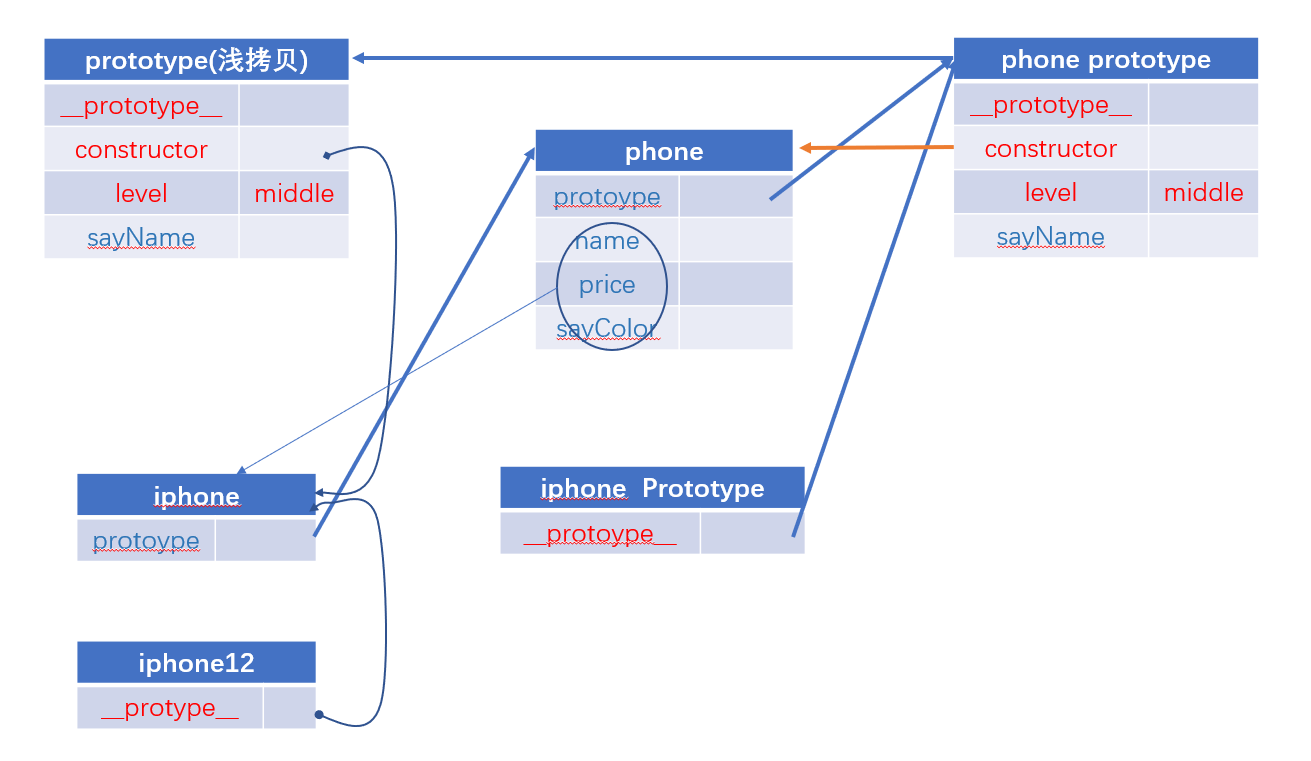
    console.log('huawei==>',huawei);



#### 6.寄生组合式继承

引用类型最理想的继承范式。

集寄生式继承和组合继承的优点于一身。



  function inheritPrototype(son,father){

    // 新建对象，将父类的原型浅拷贝到一个空对象中

    var prototype = Object.create(father.prototype)

    //增强对象

    prototype.constrctor = son

    // 指定对象，将新创建的对象赋值给子类的原型

    son.prototype = prototype

  }

  function phone(name,price){

    this.name = name

    this.price = price

    this.sayColor = function(){

      console.log('我是红色的');

    }

  }

  phone.prototype.level = 'middle'

  phone.prototype.sayName = function(){

    console.log(this.name);

  }

  // 构造函数式继承属性

  function iphone(name,price){

    phone.call(this,name)

    this.price = price

  }

  // 寄生式继承

  inheritPrototype(iphone,phone)

  const iphone12 = new iphone('苹果12',8999)

  iphone12.level = 'high'

  console.log('iphone12==>',iphone12);

