Javascript 原型

那么今天的课程要求：

不一定要求你们能掌握，因为一天理解透原型，不现实；

所以，要求你们，都记好笔记，在以后，遇到原型的时候，有一个笔记可以去看一看；

复习一下我们的构造函数：

构造函数和普通函数的区别：

1. 构造函数通过new关键字来实例化调用；
2. 构造函数内部可以使用this关键字；（普通函数的this可以指向自己么？原则上不可以啊，都是指向window对象，构造函数的this才是指向自己）；

构造函数的定义：

构造函数用来实例化对象，创建对象，构造对象的；

娱乐性的说法：构造函数其实是一个菜谱，当你调用他，就是照着这个菜谱做一份菜；

构造函数有一个属性：constructor,可以通过这个属性，来查看你所实例化的对象是否来自构造函数，我们可以称之为‘指针’；

构造函数，方法真的很好用，在你们以后的操作中，不要因为麻烦，感觉自己似懂非懂，没掌握，怕出错，就不用构造函数，这样不行，要学会接受新东西，以后开发，要多尝试采用面向对象方式开发；

但是，构造函数，有一个缺点，那就是浪费内存；

当你用构造函数，每次实例化一个对象的时候，都会开辟一个内存空间，但是明明那些方法代码都是相同，所以，这样就很浪费内存；这样，既不环保，也不效率；

那么，我们可不可以让showName方法在内存中只生成一次呢？

在javascript中，每一个构造函数都有一个prototype属性，然后这个属性指向另一个对象，这个对象的所有方法，都会被构造函数继承

function Xs(name,age){

this.name = name;

this.age = age;

}

/\*type是属性，相当于上面的name、age\*/

Xs.prototype.type = '学生';//这一条，给构造函数继承了一个属性，通过prototype指向构造函数，给他一个type属性，属性值为‘学生’；

Xs.prototype.shcool = function(){alert('我要去上学')}

var wzq = new Xs('吴泽权',18);

var wmz = new Xs('王明总',19);

console.log(wmz.shcool==wzq.shcool)；

也就是说，我们通过prototype，把school方法和type属性绑定在了Xs这个构造函数上面，然后通过new实例化两个对象的时候， 这两个对象，就继承了Xs的type和school，因为是继承来得，所以不用开辟新内存空间吧；

function Xs(){//这是第一个构造函数

this.type = '学生';

}

function Dxs(name,age){//这是第二个构造函数

Xs.call(this,arguments)//那么，强行翻译，就是把this(Dxs)的原本的型号指向并绑定XS；

this.name = name;

this.age = age;

}

var wzq = new Dxs('吴泽权',18);

console.log(wzq.type)

让子对象继承父对象，就可以用call或者apply方法，就可以继承父对象的属性或者方法，这也是最简单的一种方法；

那么，我们并不常用这个方法，我们更喜欢用原型（prototype）模式

function Xs(){//这是第一个构造函数

this.type = '学生';

}

function Dxs(name,age){//这是第二个构造函数

this.name = name;

this.age = age;

}

//上面是第一种方法，我们平时更常用的，是prototype属性来继承

Dxs.prototype = new Xs();

//将我们的Dsx的prototype对象指向Xs的实例，它会删除prototype对象原本的值，并且赋予一个新值（就是替换）

Dxs.prototype.constructor = Dxs;

console.log(Dxs.prototype.constructor==Dxs)

任何一个prototype对象，都有一个constructor属性，这个属性指向他的构造函数，前面提到过，这是一个指针，把prototype指向Dxs；

如果没有Dxs.prototype = new Xs();这一行代码，Dsx.prototype.constructor指向谁？指向大学生

但是，我们加入Dxs.prototype = new Xs();这一行后，Dxs的prototype是不是指向Xs，那么这个constructor指针是不是指向Xs

也就是说，我们直接使用prototype继承，会导致继承链非常混乱，那么我们必须手动纠正回来

***必须必须必须注意***，在编程的过程中，务必遵循，如果替换了prototype对象，那么，下一步必须给这个prototype对象添加上constructor属性，并指回原本自己的构造函数；

var wzq = new Dxs('吴泽权',18);

console.log(wzq.type)

上面都是两种方法，那么我们还有第三种方法：直接继承prototype

其实就是第二种方法的改进

//第三种方法：直接继承prototype

/\*prototype写在函数外面\*/

/\*function Xs(){}

Xs.prototype.type = '学生';

function Dxs(name,age){

this.name = name;

this.age = age;

}

Dxs.prototype = Xs.prototype;

Dxs.prototype.constructor = Dxs;

var wzq = new Dxs('吴泽权',18);

alert(wzq.type)\*/

/\*优点：效率比较高，不用创建Xs的实例对象，比较省内存\*/

/\*缺点：Dxs.prototype和Xs的prototype现在指向了同一个对象了\*/

Dxs.prototype.type = '大学生';

console.log(Xs.prototype.type);\*/

所以，这个方法是有问题的，问题出在哪儿？

Dxs.prototype.constructor = Dxs;

这一行代码，把Xs.prototype对象的constructor属性也给改掉了

第四种方法：利用空对象最为中介

由于第三种方法“直接继承prototype”存在缺点，所有就有了第四种方法，利用一个空对象作为中介：

/\*var F = function(){};

F.prototype = Xs.prototype;\*/

function Xs(){}

Xs.prototype.type = '学生';

function Dxs(name,age){

this.name = name;

this.age = age;

}

/\*Dxs.prototype = new F();

Dxs.prototype.constructor = Dxs;

var wzq = new Dxs('吴泽权',18);

alert(wzq.type)\*/

/\*首先我们F空对象继承Xs父对象，然后Dxs继承我们的空对象，这样，继承就完成了，

并且由于空对象几乎不占内存，所以不怎么好资源，其次，由于我们Dxs继承的是空对象，所以对

Dxs.prototype对象进行修改，不会影响到父对象Xs,至于空对象，影不影响，没关系\*/

//那么，我们利用空对象继承的方法，明显，每一个对象继承Xs的时候，都要创建一个空对象，那么就可以封装，对不对？

function extend(Child,Parent){

var F = function(){};

F.prototype = Parent.prototype;

Child.prototype = new F();

Child.prototype.constructor = Child;

Child.uber = Parent.prototype;

//意思，为子对象设置一个uber属性，这个属性指向父对象的prototype属性，uber是“向上”、“上一层”的意思

//这个就相当于给子对象开了一条路，可以直接调用父对象方法，在这里只是为了实现继承的完整性，属于备用性质，因为子对象继承的是空对象，所以要通过这行代码才能调用得到父对象；

}

extend(Dxs,Xs);

var wzq = new Dxs('吴泽权',18);

alert(wzq.type)

最后一个继承，拷贝继承；