

# js进阶一（prototype、prototype指向、原型继承、构造继承、组合继承、为w。

o o

## 文章目录

## prototype

原型？

实例对象中有\_\_proto\_\_这个属性,叫原型,也是一个对象,这个属性是给浏览器使用,不是标准的属性----->proto----->可以叫原型对象

构造函数中有prototype这个属性,叫原型,也是一个对象,这个属性是给程序员使用,是标准的属性----->prototype—>可以叫原型对象

实例对象的\_\_proto\_\_和构造函数中的prototype相等—>true

又因为实例对象是通过构造函数来创建的,构造函数中有原型对象prototype

实例对象的\_\_proto\_\_指向了构造函数的原型对象prototype

通过原型来添加方法,解决数据共享,节省内存空间

我们来看一个例子

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
//通过原型来添加方法,解决数据共享,节省内存空间
Person.prototype.eat = function () {
  console.log("吃凉菜");
};

var p1 = new Person("小明", 20);

p1.__proto__.eat();
p1.eat()
Person.prototype.eat()

console.log(p1.constructor == Person);
console.log(p1.__proto__.constructor == Person);
console.log(p1.__proto__ == Person.prototype);
console.log(p1.__proto__.constructor == Person.prototype.constructor);
```

输出如下：

```
吃凉菜
吃凉菜
true
true
true
true
```

原型指向可以改变

实例对象的原型\_\_proto\_\_指向的是该对象所在的构造函数的原型对象

实例对象中有\_\_proto\_\_原型

构造函数中有prototype原型

prototype是对象

所以,prototype这个对象中也有\_\_proto\_\_,那么指向了哪里

实例对象中的\_\_proto\_\_指向的是构造函数的prototype

所以,prototype这个对象中\_\_proto\_\_指向的应该是某个构造函数的原型prototype

```
function Person() {  
  
}  
Person.prototype.eat=function () {  
  console.log("吃东西");  
};  
  
var per=new Person();  
console.dir(per);  
console.dir(Person);  
  
console.log(Person.prototype.__proto__);  
  
//per实例对象的__proto__----->Person.prototype的__proto__----->Object.prototype的__proto__是null  
  
console.log(per.__proto__===Person.prototype);  
console.log(per.__proto__.__proto__===Person.prototype.__proto__);  
console.log(Person.prototype.__proto__===Object.prototype);  
console.log(Object.prototype.__proto__);
```

输出如下:

```
Person  
  f Person()  
{constructor: f, __defineGetter__: f, __defineSetter__: f, hasOwnProperty: f, __lookupGetter__: f, ...}  
true  
true  
true  
null
```

## prototype中方法互相访问

```
function Animal(name,age) {  
  this.name=name;  
  this.age=age;  
}  
//原型中添加方法  
Animal.prototype.eat=function () {  
  console.log("动物吃东西");  
  this.play();  
};  
Animal.prototype.play=function () {  
  console.log("玩球");  
};  
var dog=new Animal("小苏",20);  
dog.eat();
```

输出如下:

```
动物吃东西  
玩球
```

## prototype中找寻属性、方法

实例对象使用的属性或者方法,先在实例中查找,找到了则直接使用,找不到则,去实例对象的\_\_proto\_\_指向的原型对象prototype中找,找到了则使用,找不到则报错

```
function Person(age,sex) {
    this.age=age;//年龄
    this.sex=sex;
    this.eat=function () {
        console.log("构造函数中的吃");
    };
}
Person.prototype.sex="女";
Person.prototype.eat=function () {
    console.log("原型对象中的吃");
};

var per=new Person(20,"男");
console.log(per.sex);//男
per.eat();
```

输出如下:

```
男
构造函数中的吃
```

我们在看下面一个例子:

```
function Person(age,sex) {
    this.age=age;
    this.sex=sex;
}
Person.prototype.sex="女";
var per=new Person(10,"男");
console.log(per.sex);
//因为JS是一门动态类型的语言,对象没有什么,只要点了,那么这个对象就有了这个东西,没有这个属性,只要对象.属性名字,对象就有这个属性了,但是,该属性没有赋值,所以,结果是:undefined
console.log(per.fdsfdsfsdfs);
```

输出如下:

```
男
undefined
```

## 改变prototype指向

原型指向可以改变

实例对象的原型\_\_proto\_\_指向的是该对象所在的构造函数的原型对象

构造函数的原型对象(prototype)指向如果改变了,实例对象的原型(proto)指向也会发生改变

原型的指向是可以改变的

实例对象和原型对象之间的关系是通过\_\_proto\_\_原型来联系起来的,这个关系就是原型链

```
function Person(age) {
  this.age = age;
}
Person.prototype.eat = function () {
  console.log("人正在吃东西");
};
//学生构造函数
function Student(sex) {
  this.sex = sex;
}

//学生的原型中添加方法----先在原型中添加方法
Student.prototype.sayHi = function () {
  console.log("您好哦");
};
//改变了Student原型对象的指向
Student.prototype = new Person(10);

var stu = new Student("男");
stu.eat();
// stu.sayHi();
console.info(stu.sex)
console.info(stu.age)
console.dir(stu)
```

输出如下：

```
人正在吃东西
男
10
```

```
Student
sex: "男"
__proto__: Person
age: 10
__proto__:
eat: f ()
constructor: f Person(age)
__proto__: Object
```

## 内置对象添加**prototype**方法

```
String.prototype.sayHi = function () {
  console.log(this + "哈哈,萨芬");
};

//字符串就有了打招呼的方法
var str2 = "sadly";
str2.sayHi();
```

## **window**全局对象

```

<script>

//通过自调用函数产生一个随机数对象,在自调用函数外面,调用该随机数对象方法产生随机数
(function (window) {
    //产生随机数的构造函数
    function Random() {
    }
    //在原型对象中添加方法
    Random.prototype.getRandom = function (min, max) {
        return Math.floor(Math.random() * (max - min) + min);
    };
    //把Random对象暴露给顶级对象window--->外部可以直接使用这个对象
    window.Random = Random;
})(window);
//实例化随机数对象
var rm = new Random();
//调用方法产生随机数
console.log(rm.getRandom(0, 5));
//全局变量
</script>

```

## 通过原型实现继承

```

function Person(name, age, sex) {
    this.name = name;
    this.sex = sex;
    this.age = age;
}
Person.prototype.eat = function () {
    console.log("人可以吃东西");
};
Person.prototype.sleep = function () {
    console.log("人在睡觉");
};
Person.prototype.play = function () {
    console.log("生活就是不一样的玩法而已");
};

function Student(score) {
    this.score = score;
}
//改变学生的原型的指向即可=====>学生和人已经发生关系
Student.prototype = new Person("小明", 10, "男");
Student.prototype.study = function () {
    console.log("学习很累很累的哦.");
};

//相同的代码太多,造成了代码的冗余(重复的代码)

var stu = new Student(100);
console.log(stu.name);
console.log(stu.age);
console.log(stu.sex);
stu.eat();
stu.play();
stu.sleep();
console.log("下面的是学生对象中自己有的");
console.log(stu.score);
stu.study();

console.dir(stu)

```

输出结果如下：

```
小明
10
男
人可以吃东西
生活就是不一样的玩法而已
人在睡觉
下面的是学生对象中自己有的
100
学习很累很累的哦.
```

```
▼ Student ⓘ
  score: 100
  ▼ __proto__: Person
    age: 10
    name: "小明"
    sex: "男"
    ▶ study: f ()
    ▼ __proto__:
      ▶ eat: f ()
      ▶ play: f ()
      ▶ sleep: f ()
      ▶ constructor: f Person(name, age, sex)
```

## 继承例子

```
//动物的构造韩素
function Animal(name,weight) {
  this.name=name;
  this.weight=weight;
}
//动物的原型的方法
Animal.prototype.eat=function () {
  console.log("天天吃东西,就是吃");
};

//狗的构造函数
function Dog(color) {
  this.color=color;
}
Dog.prototype=new Animal("哮天犬","50kg");
Dog.prototype.bitePerson=function () {
  console.log("哼~汪汪~咬死你");
};

//哈士奇
function ErHa(sex) {
  this.sex=sex;
}
ErHa.prototype=new Dog("黑白色");
ErHa.prototype.playHost=function () {
  console.log("哈哈~要坏衣服,要坏桌子,拆家..嘎嘎...好玩,开心不,惊喜不,意外不");
};
var erHa=new ErHa("雄性");
console.log(erHa.name,erHa.weight,erHa.color);
erHa.eat();
erHa.bitePerson();
erHa.playHost();

console.info(erHa)
```

效果如下：

哮天犬 50kg 黑白色  
天天吃东西,就是吃  
哼~汪汪~咬死你  
哈哈~要坏衣服,要坏桌子,拆家..嘎嘎...好玩,开心不,惊喜不,意外不

```
▼ ErHa {sex: "雄性"} ⓘ  
  sex: "雄性"  
  __proto__: Animal  
    color: "黑白色"  
    ▶ playHost: f ()  
      ▼ __proto__: Animal  
        ▶ bitePerson: f ()  
          name: "哮天犬"  
          weight: "50kg"  
        ▼ __proto__:  
          ▶ eat: f ()  
          ▶ constructor: f Animal(name,weight)
```

## 借用构造函数

为了数据共享,改变原型指向,做到了继承—通过改变原型指向实现的继承

缺陷:因为改变原型指向的同时实现继承,直接初始化了属性, 继承过来的属性的值都是一样的了,所以,这就是问题  
只能重新调用对象的属性进行重新赋值,

借用构造函数:构造函数名字.call(当前对象,属性,属性,属性...);

解决了属性继承,并且值不重复的问题

缺陷:父级类别中的方法不能继承

```
function Person(name, age, sex, weight) {  
  this.name = name;  
  this.age = age;  
  this.sex = sex;  
  this.weight = weight;  
}  
Person.prototype.sayHi = function () {  
  console.log("您好");  
};  
function Student(name,age,sex,weight,score) {  
  //借用构造函数  
  Person.call(this,name,age,sex,weight);  
  this.score = score;  
}  
var stu1 = new Student("小明",10,"男","10kg","100");  
console.log(stu1.name, stu1.age, stu1.sex, stu1.weight, stu1.score);  
  
var stu2 = new Student("小红",20,"女","20kg","120");  
console.log(stu2.name, stu2.age, stu2.sex, stu2.weight, stu2.score);
```

输出如下:

小明 10 男 10kg 100  
小红 20 女 20kg 120

## 组合继承


```
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
Person.prototype.sayHi = function () {
  console.log("阿涅哈斯诶呦");
};

function Student(name, age, sex, score) {
  //借用构造函数:属性值重复的问题十神
  Person.call(this, name, age, sex);
  this.score = score;
}
//改变原型指向----继承
Student.prototype = new Person();//不传值
Student.prototype.eat = function () {
  console.log("吃东西");
};
var stu = new Student("小黑", 20, "男", "100分");
console.log(stu.name, stu.age, stu.sex, stu.score);
stu.sayHi();
stu.eat();

console.dir(stu)
```

输出结果如下：

小黑 20 男 100分  
阿涅哈斯诶呦  
吃东西

```
▼ Student 
  age: 20
  name: "小黑"
  score: "100分"
  sex: "男"
  ▼ __proto__: Person
    age: undefined
    ► eat: f ()
    name: undefined
```