**文章目录**

* [面向对象](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "_1" \t "_self)
  + [概念](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "_2" \t "_self)
  + [对象和关联数组](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "_6" \t "_self)
  + [继承和prototype](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "prototype_15" \t "_self)
  + [构造器实现伪继承](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "_148" \t "_self)
  + [使用apply或call实现伪继承](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "applycall_250" \t "_self)
* [创建对象](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "_308" \t "_self)
  + [使用new关键字调用构造器创建对象](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "new_309" \t "_self)
  + [使用Object类直接创建对象](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "Object_324" \t "_self)
  + [使用JSON语法创建对象](https://blog.csdn.net/qq_36957681/article/details/90108143" \l "JSON_340" \t "_self)

**面向对象**

**概念**

**JavaScript中的每个函数都可用于创建对象，返回的对象既是该类的实例，也是Object类的实例。**

**JavaScrtpt的函数定义不支持继承语法。所以习惯称JavaScript是基于对象的脚本语言。**

**开发者自己定义的类没有父子关系，但这些类都是Object类的子类。**

**对象和关联数组**

**JavaScript中的对象本质上是一个关联数组，类似Java里的Map数据结构，由一组key-value对组成。其中value既可以是值（包括基本类型的值和复合类型的值），也可以是函数。**

**当value是基本类型的值或复合类型的值时，此时value是该对象的属性值。**

**当value是函数时，此时value是该对象的方法。**

**访问对象的属性**

**使用 obj.propName 的形式**

**使用 obj[propName] 的形式，有的时候必须使用这种形式，当需遍历对象属性的时候。**

**继承和prototype**

**JavaScript的函数没有提供显式的继承语法**

**<script type="text/javascript">**

**function Person(name, age){**

**this.name = name;**

**this.age = age;**

**this.info = function(){**

**document.write("姓名：" + this.name);**

**document.write("年龄：" + this.age);**

**}**

**}**

**var p1 = new Person("hhh", 22);**

**p1.info();**

**document.writeln("<hr>");**

**var p2 = new Person("aaa", 26);**

**p2.info();**

**</script>**

**使用上述方法为类定义方法不好，原因：**

**性能低下： 每次创建一个Person对象时，系统都会创建一个新的info函数。这会造成系统内存泄漏，从而引起性能下降。**

**使得info函数中的局部变量产生闭包： 使得局部变量可以在函数外被访问到。**

**闭包 指用到其他函数的局部变量的函数。**

**局部变量 在函数外部是访问不到的，可以用闭包方法return该变量，这样就可以在函数外部访问到局部变量。**

**<script type="text/javascript">**

**function Person(){**

**var locVal = "局部变量";**

**this.info = function(){**

**return locVal;**

**}**

**}**

**var p = new Person();**

**var val = p.info();**

**//输出val返回值，该返回值就是局部变量locVal**

**alert(val);**

**</script>**

为避免上面的两种情况，不建议直接在函数定义中直接为该函数定义方法，而是建议使用**prototype属性**。

JavaScript的所有类都有一个prototype属性，如果为JavaScript类的prototype属性增加属性、方法，则可视为对原有类的扩展。

示例：

<script type="text/javascript">

function Person(name, age){

this.name = name;

this.age = age;

}

var p1 = new Person("hhh", 22);

//此处p1不可调用info方法，变量p1还没有info方法

//将info方法增加到Person的prototype属性上

Person.prototype.info = function(){

document.write("姓名：" + this.name);

document.write("年龄：" + this.age);

}

//执行p1的info方法

p1.info();

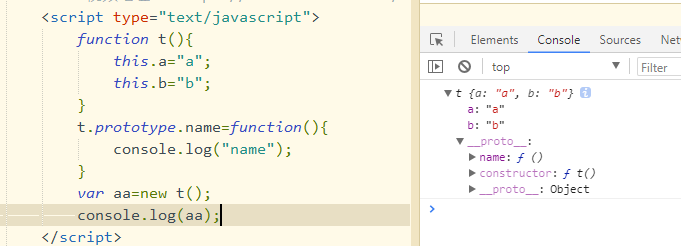
document.writeln("<hr>");

//创建Person的实例p2

var p2 = new Person("aaa", 26);

p2.info();

</script>



上述是用prototype为Person类增加一个info方法，这样让所有Person实例共享一个info方法，而且该info方法不在Person函数之内，因此不会产生闭包。

JavaScript是一门动态语言，不仅可以为对象动态地增加属性和方法，也可以动态地为类增加属性和方法。

为对象增加：创建实例p后，p.name = "name"; 可以为实例p添加自己的属性name。

为类增加：Person.prototype.name = "name"; 可以为Person类添加属性name。

通过prototype属性为一个类动态增加属性和方法，被认为是一种“伪继承”，实质是修改原有的类，并不是产生一个新的子类。

通过使用prototype属性，可以对JavaScript的内建类进行扩展

下例是为内建类Array增加indexof方法，该方法用于判断数组中是否包含了某元素。

<script type="text/javascript">

Array.prototype.indexof = function(obj){

var result = -1;

for(var i=0; i<this.length; i++){

if(this[i] == obj){

result = i;

break;

}

}

return result;

}

var arr = [4, 5, 7, -2];

alert(arr.indexof(-2));

</script>

建议在类定义结束后立即增加该类所需的方法。

尽量避免使用内嵌函数为类定义方法，而应该使用增加prototype属性的方式来增加方法。

继承一：原型链继承方法

在默认情况下，JavaScript类的prototype属性值是一个Object对象，将JavaScript类的prototype设为父类实例，可实现JavaScript语言的继承。

<script type="text/javascript">

function Person(name, age){

this.name = name;

this.age = age;

}

Person.prototype.sayHello = function(){

console.log(this.name + "向你打招呼！");

}

var per = new Person("张三", 22);

per.sayHello();

function Student(grade){

this.grade = grade;

}

//将Student的prototype设为Person对象

Student.prototype = new Person("未命名", 0);

Student.prototype.intro = function(){

console.log("%s是个学生，读%d年级", this.name, this.grade);

}

var stu = new Student(5);

stu.name = "李四"; //原型链继承，为属性赋值时需这样一句一句进行赋值，所以将采用下一种构造器继承方法来对属性进行赋值。

console.log(stu instanceof Student);

console.log(stu instanceof Person);

stu.sayHello();

stu.intro();

console.log(stu.name + stu.age);

</script>

## 构造器实现伪继承

继承二：构造器继承方法  
原型链继承方法可以实现父类的属性、方法继承，但是为属性赋值麻烦，因而使用构造器继承方法实现属性简便赋值。

<script type="text/javascript">

function Person(name, age){

this.name = name;

this.age = age;

this.sayHello = function(){

console.log(this.name + "向你打招呼！");

}

}

var per = new Person("张三", 22);

per.sayHello();

function Student(name, age, grade){

//定义一个实例属性引用Person类

this.inherit\_temp = Person;

//以this为调用者，调用Person类的构造器——这样Person构造器中this就全部换成了当前的Student

this.inherit\_temp(name, age);

this.grade = grade;

}

Student.prototype.intro = function(){

console.log("%s是个学生，读%s年级", this.name, this.grade);

}

var stu = new Student("李四", 34, 5);

console.log(stu instanceof Student);//输出true

console.log(stu instanceof Person);//伪继承，输出false

stu.sayHello();

stu.intro();

</script>

上面代码示例将sayHello方法写在Person类中，Student类可以继承该方法。  
但下面代码示例用prototype属性动态增加sayHello方法，此时Student类将不能继承该方法。

<script type="text/javascript">

function Person(name, age){

this.name = name;

this.age = age;

}

Person.prototype.sayHello = function(){

console.log(this.name + "向你打招呼！");

}

var per = new Person("张三", 22);

per.sayHello();

function Student(name, age, grade){

this.inherit\_temp = Person;

this.inherit\_temp(name, age);

this.grade = grade;

}

Student.prototype.intro = function(){

console.log("%s是个学生，读%s年级", this.name, this.grade);

}

var stu = new Student("李四", 34, 5);

console.log(stu instanceof Student);//输出true

console.log(stu instanceof Person);//伪继承，输出false

stu.sayHello();//将产生错误

stu.intro();

</script>

要是想实现上面代码中sayHello方法的继承，需添加Student.prototype = new Person(); 。

<script type="text/javascript">

function Person(name, age){

this.name = name;

this.age = age;

}

Person.prototype.sayHello = function(){

console.log(this.name + "向你打招呼！");

}

var per = new Person("张三", 22);

per.sayHello();

function Student(name, age, grade){

this.inherit\_temp = Person;

this.inherit\_temp(name, age);

this.grade = grade;

}

Student.prototype = new Person();//需放在下面语句的前面

Student.prototype.intro = function(){

console.log("%s是个学生，读%s年级", this.name, this.grade);

}

var stu = new Student("李四", 34, 5);

console.log(stu instanceof Student);//输出true

console.log(stu instanceof Person);//伪继承，输出false

stu.sayHello();//正确输出

stu.intro();//正确输出

</script>

使用apply或call实现伪继承

继承三：对象冒充继承方法

上一种方法实现伪继承的关键在于子类构造器需要以this作为调用者来调用父类构造器，这样父类构造器中的this就会变成代表子类，子类就可以得到原父类定义的实例属性和方法，因此这种伪继承方法可以用apply或call来实现。

Person.call(this, name, age);

//或

Person.apply(this, [name, age]);

用上面一句话代替下面两句话。

this.inherit\_temp = Person;

this.inherit\_temp(name, age);

最终标准继承写法

* 用**对象冒充方法**继承父类属性，方便为属性赋值。
* 用**原型链方法**来继承父类通过prototype属性增加的方法。

**标准写法：**

<script type="text/javascript">

function Animal(sex, age){

this.sex = sex;

this.age = age;

}

Animal.prototype.info = function(){

console.log("性别:" + this.sex + "年龄：" + this.age);

}

function Dog(sex, age, type){

Animal.call(this, sex, age);

this.type = type;

}

Dog.prototype = new Animal();

Dog.prototype.jiao = function(){

console.log("汪汪汪...");

}

var hashiqi = new Dog("men", 5, "hashiqi");

hashiqi.info();

hashiqi.jiao();

function Cat(sex, age, type){

Animal.call(this, sex, age);

this.type = type;

}

Cat.prototype = new Animal();

Cat.prototype.jiao = function(){

console.log("喵喵喵...");

}

var boshi = new Cat("women", 3, "boshi");

boshi.info();

boshi.jiao();

</script>

创建对象

使用new关键字调用构造器创建对象

<script type="text/javascript">

//定义一个命名函数，同时也定义了一个Person类

function Person(name, age){

this.name = name;

this.age = age;

}

//创建Person实例

var p1 = new Person();

var p2 = new Person('张三', 20);

</script>

使用Object类直接创建对象

<script type="text/javascript">

//创建空对象

var myObj = new Object();

//增加属性

myObj.name = '张三；

myObj.age = 20;

//增加方法

myObj.info = function abc(){

document.writeln("姓名：" + this.name + "年龄：" + this.age);

}

</script>

使用JSON语法创建对象

JSON语法： 以{开始，以}结束，对象的每个属性名和属性值之间以英文冒号（:）隔开，多个属性之间以英文逗号（,）隔开，最后一个属性定义后不再有英文逗号。

属性值 不仅可以是普通字符串，也可以是任何基本类型，还可以是函数、数组，甚至可以是另一个JSON语法创建的对象。

<script type="text/javascript">

person = {

name : '张三',

age : 30,

//使用JSON对象为其指定一个属性

son : {

name : '李四',

age : 10

},

//使用JSON语法为person直接分配一个方法

info : function(){

document.writeln("姓名：" + this.name + "<br>年龄：" + this.age);

}

};

person.info();//显示张三30

alert(person.son.name)//显示李四

</script>

<script type="text/javascript">

var person = {

name : '张三',

age : 30,

schools : ['小学', '中学', '大学'],

//使用JSON对象为其指定一个属性

parents : [

{

name : 'father',

age : 60,

address : '广州'

},

{

name : 'mother',

age : 58,

address : '上海'

}

],

info : function(){

document.writeln("姓名：" + this.name + "<br>年龄：" + this.age);

}

};

alert(person.parents[0].age);//输出60

alert(person.schools[2]);//输出大学

</script>

本文链接：<https://blog.csdn.net/qq_36957681/article/details/90108143>