**JavaScript是如何实现继承的(六种方式)**

作者：Evan\_Chen 字体：[[增加](javascript:turnbig()) [减小](javascript:turnsmall())] 类型：转载 时间：2016-03-31[我要评论](http://www.jb51.net/article/81766.htm#comments)

大多OO语言都支持两种继承方式： 接口继承和实现继承 ，而ECMAScript中无法实现接口继承，ECMAScript只支持实现继承，而且其实现继承主要是依靠原型链来实现，下文给大家技术js实现继承的六种方式，需要的朋友参考下

前言：大多OO语言都支持两种继承方式： 接口继承和实现继承 ，而ECMAScript中无法实现接口继承，ECMAScript只支持实现继承，而且其实现继承主要是依靠 原型链 来实现。

**1.原型链**

基本思想：利用原型让一个引用类型继承另外一个引用类型的属性和方法。

构造函数，原型，实例之间的关系：每个构造函数都有一个原型对象，原型对象包含一个指向构造函数的指针，而实例都包含一个指向原型对象的内部指针。

原型链实现继承例子：

function SuperType() {

this.property = true;

}

SuperType.prototype.getSuperValue = function() {

return this.property;

}

function subType() {

this.property = false;

}

//继承了SuperType

SubType.prototype = new SuperType();

SubType.prototype.getSubValue = function (){

return this.property;

}

var instance = new SubType();

console.log(instance.getSuperValue());//true

**2.借用构造函数**

基本思想：在子类型构造函数的内部调用超类构造函数，通过使用call()和apply()方法可以在新创建的对象上执行构造函数。

例子：

function SuperType() {

this.colors = ["red","blue","green"];

}

function SubType() {

SuperType.call(this);//继承了SuperType

}

var instance1 = new SubType();

instance1.colors.push("black");

console.log(instance1.colors);//"red","blue","green","black"

var instance2 = new SubType();

console.log(instance2.colors);//"red","blue","green"

**3.组合继承**

基本思想：将原型链和借用构造函数的技术组合在一块，从而发挥两者之长的一种继承模式。

例子：

function SuperType(name) {

this.name = name;

this.colors = ["red","blue","green"];

}

SuperType.prototype.sayName = function() {

console.log(this.name);

}

function SubType(name, age) {

SuperType.call(this,name);//继承属性

this.age = age;

}

//继承方法

SubType.prototype = new SuperType();

Subtype.prototype.constructor = Subtype;

Subtype.prototype.sayAge = function() {

console.log(this.age);

}

var instance1 = new SubType("EvanChen",18);

instance1.colors.push("black");

consol.log(instance1.colors);//"red","blue","green","black"

instance1.sayName();//"EvanChen"

instance1.sayAge();//18

var instance2 = new SubType("EvanChen666",20);

console.log(instance2.colors);//"red","blue","green"

instance2.sayName();//"EvanChen666"

instance2.sayAge();//20

**4.原型式继承**

基本想法：借助原型可以基于已有的对象创建新对象，同时还不必须因此创建自定义的类型。

原型式继承的思想可用以下函数来说明：

function object(o) {

function F(){}

F.prototype = o;

return new F();

}

例子：

var person = {

name:"EvanChen",

friends:["Shelby","Court","Van"];

};

var anotherPerson = object(person);

anotherPerson.name = "Greg";

anotherPerson.friends.push("Rob");

var yetAnotherPerson = object(person);

yetAnotherPerson.name = "Linda";

yetAnotherPerson.friends.push("Barbie");

console.log(person.friends);//"Shelby","Court","Van","Rob","Barbie"

ECMAScript5通过新增Object.create()方法规范化了原型式继承，这个方法接收两个参数：一个用作新对象原型的对象和一个作为新对象定义额外属性的对象。

var person = {

name:"EvanChen",

friends:["Shelby","Court","Van"];

};

var anotherPerson = Object.create(person);

anotherPerson.name = "Greg";

anotherPerson.friends.push("Rob");

var yetAnotherPerson = Object.create(person);

yetAnotherPerson.name = "Linda";

yetAnotherPerson.friends.push("Barbie");

console.log(person.friends);//"Shelby","Court","Van","Rob","Barbie"

**5.寄生式继承**

基本思想：创建一个仅用于封装继承过程的函数，该函数在内部以某种方式来增强对象，最后再像真正是它做了所有工作一样返回对象。

例子：

function createAnother(original) {

var clone = object(original);

clone.sayHi = function () {

alert("hi");

};

return clone;

}

var person = {

name:"EvanChen",

friends:["Shelby","Court","Van"];

};

var anotherPerson = createAnother(person);

anotherPerson.sayHi();///"hi"

**6.寄生组合式继承**

基本思想：通过借用函数来继承属性，通过原型链的混成形式来继承方法

其基本模型如下所示：

function inheritProperty(subType, superType) {

var prototype = object(superType.prototype);//创建对象

prototype.constructor = subType;//增强对象

subType.prototype = prototype;//指定对象

}

例子：

function SuperType(name){

this.name = name;

this.colors = ["red","blue","green"];

}

SuperType.prototype.sayName = function (){

alert(this.name);

};

function SubType(name,age){

SuperType.call(this,name);

this.age = age;

}

inheritProperty(SubType,SuperType);

SubType.prototype.sayAge = function() {

alert(this.age);

}

以上内容给大家介绍了javascript实现继承的六种方式，希望对大家有所帮助！