JavaScript编写类

**地址：http://www.cnblogs.com/xdp-gacl/p/3700840.html**

在工作中经常用到JavaScript，今天总结一下JavaScript编写类的几种写法以及这几种写法的优缺点，关于JavaScript编写类的方式，在网上看到很多，而且每个人的写法都不太一样，经常看到的就是以下几种方式。

**1、构造函数方式**

用构造函数模拟"类"，在其内部用this关键字指代实例对象。

**基本语法：**

**function 类名(){  
     this.属性名;//公共属性  
     var 属性名;//私有属性  
    /\*凡是定义类的公共属性和公共方法都要使用this\*/  
    //定义类的公共函数  
    this.函数名=function(){  
            .....   
    }  
    //定义类的私有函数  
    function 函数名(){  
　　　　......  
    }  
}**

**范例：**

1 /\*定义一个Person类\*/

2 function Person(\_name,\_age,\_salary){

3 //Person类的公开属性，类的公开属性的定义方式是：”this.属性名“

4 this.name=\_name;

5 //Person类的私有属性，类的私有属性的定义方式是：”var 属性名“

6 var age=\_age;//私有属性

7 var salary=\_salary;//私有属性

8

9 /\*定义私有属性Age的对外公开访问方法\*/

10 this.setAge = function(intAge) {

11 age = intAge;

12 }

13 /\*定义私有属性Age的对外公开访问方法\*/

14 this.getAge = function() {

15 return age;

16 }

17

18 //定义Person类的公开方法(特权方法)，类的公开方法的定义方式是：”this.functionName=function(){.....}“

19 this.Show=function(){

20 document.writeln("在公开方法里面访问类的私有属性是允许的，age="+age+"\t"+"salary="+salary);//在公开方法里面访问类的私有属性是允许的

21 }

22 //公共方法

23 this.publicMethod = function(){

24 document.writeln("在公开方法里面访问类的私有方法是允许的");

25 privateFn();//在公开方法里面调用类的私有方法

26 privateFn2();//在公开方法里面调用类的私有方法

27 }

28 /\*

29 定义Person类的私有方法(内部方法)，

30 类的私有方法的定义方式是：”function functionName(){.....}“，

31 或者 var functionName=function(){....}

32 \*/

33 function privateFn(){

34 document.writeln("我是Person类的私有函数privateFn");

35 }

36

37 var privateFn2=function(){

38 document.writeln("我是Person类的私有函数privateFn2");

39 }

40 }

测试Person类

1 var p1 = new Person("孤傲苍狼",24,2300);

2 var p2 = new Person("白虎神皇",24,2300);

3 document.write("<pre>");

4 document.writeln("p1 instanceof Person的结果是："+(p1 instanceof Person));//p1是Person类的实例，结果是true

5 document.writeln("p2 instanceof Person的结果是："+(p2 instanceof Person));//p2是Person类的实例，结果是true

6 //当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等

7 document.writeln("当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等");

8 document.writeln("比较p1和p2这两个对象的show方法的内存地址是否一样：p1.show== p2.show的结果是："+(p1.show == p2.show));//false

9 document.writeln("p1.show == p2.show的结果是："+(p1.show == p2.show)+"，这证明p1对象和p2对象不是共享一个show方法，在内存中show方法的代码有2份，存放在两块内存区域");

10 document.writeln("name是Person类定义的public属性，可以使用类的对象去直接访问类的public属性");

11 document.writeln("p1.name="+p1.name);//访问公有属性，这是可以正常访问的

12 document.writeln("age和salary是Person类定义的private属性，不能使用类的对象去直接访问类私有属性，这是访问不了的，结果都是undefined");

13 document.writeln("p1.age="+p1.age+"，"+"p1.salary="+p1.salary)//不能使用类的对象去直接访问类私有属性，这是访问不了的，结果都是undefined

14 p1.show();//调用类的公共函数，这次允许的

15 p1.publicMethod();//调用类的公共函数，这次允许的

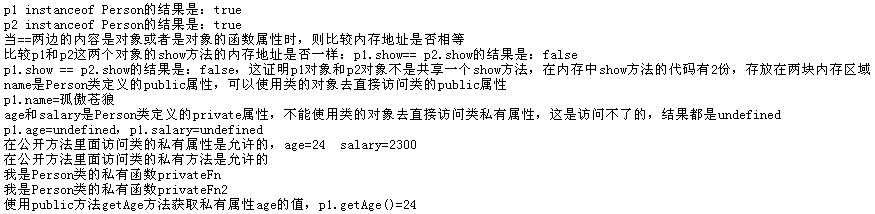
16 p1.setAge(24);//使用public方法setAge方法为私有属性age赋值

17 document.writeln("使用public方法getAge方法获取私有属性age的值，p1.getAge()="+p1.getAge());//使用getAge方法获取私有属性age的值

18 //document.writeln("p1.privateFn()："+p1.privateFn()+"&nbsp;p1.privateFn2()："+p1.privateFn2());//不能使用类的对象去调用类的私有方法，这里会报错”对象不支持此属性或者方法

19 document.write("</pre>");

测试结果：



这种方式的优点是：可以根据参数来构造不同的对象实例 ，每个对象的属性一般是不相同的，缺点是构造每个实例对象时，方法不能共享，Person类里面定义的那些方法，p1对象有一份，p2也有一份，那么在内存中就得开辟两块内存空间来分别存储p1的方法和p2的方法，这样就造成了内存的浪费。对于一个类的不同实例对象，这些对象的属性一般是不相同的，但是方法是相同的，所以节约内存的做法就是把方法放到内存的一块区域中存放，然后每个实例对象都从这块内存中取出方法。

**2、原型方式**

 需要说明的是，使用原型方式编写JavaScript类是无法给类添加私有属性和私有方法的，使用原型方式添加的属性和方法都是public的。

写法一：

1 /\*定义一个Person类\*/

2 function Person(\_name,\_age,\_weight,\_height){

3 this.init(\_name,\_age,\_weight,\_height);

4 }

5

6 /\*使用原型的方式定义Person类的public属性：name,age,weight,height，使用原型的方式添加的属性都是public的\*/

7 Person.prototype.name;

8 Person.prototype.age;

9 Person.prototype.weight;

10 Person.prototype.height;

11 /\*使用原型的方式给Person类添加public方法，使用原型的方式添加的方法都是public的\*/

12 /\*使用原型的方式给Person类添加init方法\*/

13 Person.prototype.init = function(\_name,\_age,\_weight,\_height) {

14 if(\_name != undefined && \_age!=undefined && \_weight!=undefined && \_height!=undefined){

15 this.name = \_name;

16 this.age = \_age;

17 this.weight=\_weight;

18 this.height=\_height;

19 document.writeln("this.name="+this.name+",this.age="+this.age+",this.weight="+this.weight+",this.height="+this.height);

20 }

21

22 }

23 /\*使用原型的方式给Person类添加show方法\*/

24 Person.prototype.show = function(){

25 document.writeln("show method");

26 }

测试Person类

1 document.write("<pre>");

2 var p1 = new Person("孤傲苍狼",24,115,160);

3 var p2 = new Person("白虎神皇",25,120,170);

4 var p3 = new Person();

5 p3.init("玄天邪帝",26,130,180);//调用public方法init初始化p3对象

6 document.writeln("p1 instanceof Person的结果是："+(p1 instanceof Person));//p1是Person类的实例，结果是true

7 document.writeln("p2 instanceof Person的结果是："+(p2 instanceof Person));//p2是Person类的实例，结果是true

8 document.writeln("p3 instanceof Person的结果是："+(p3 instanceof Person));//p3是Person类的实例，结果是true

9 //当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等

10 document.writeln("当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等");

11 document.writeln("比较p1和p2这两个对象的show方法的内存地址是否一样：p1.show == p2.show的结果是："+(p1.show == p2.show));//true

12 document.writeln("p1.show == p2.show的结果是："+(p1.show == p2.show)+"，这证明p1对象和p2对象共享一个show方法，在内存中show方法的代码只有一份，存放在内存的一块区域");//true

13 document.writeln("p1.name="+p1.name+"，p1.age="+p1.age+"，p1.weight="+p1.weight+"，p1.height="+p1.height);//访问公有属性，这是可以正常访问的

14 document.writeln("p2.name="+p2.name+"，p2.age="+p2.age+"，p2.weight="+p2.weight+"，p2.height="+p2.height);//访问公有属性，这是可以正常访问的

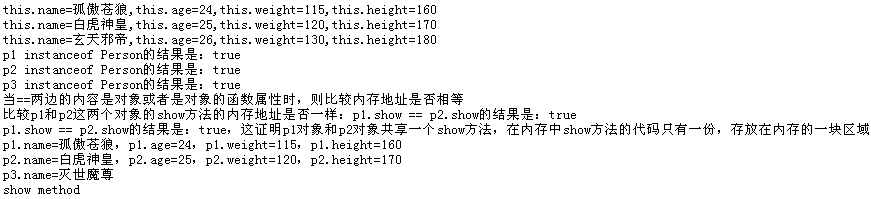
15 p3.name="灭世魔尊";//为公共属性重新赋值

16 document.writeln("p3.name="+p3.name);//访问公有属性，这是可以正常访问的

17 p1.show();//调用类的公共函数，这次允许的

18 document.write("</pre>");

测试结果：



写法二：

使用原型方式给类定义public属性和public方法更加优雅的写法，我个人推荐使用这种方式，这种方式看起来比较舒服

1 /\*定义类Person2\*/

2 function Person2(){

3

4 }

5

6 /\*使用原型方式给类定义public属性和public方法更加优雅的写法\*/

7 Person2.prototype = {

8 name:"",//public属性

9 age:0,//public属性

10 weight:0,//public属性

11 height:0,//public属性

12 /\*public方法\*/

13 init:function(\_name,\_age,\_weight,\_height) {

14 this.name = \_name;

15 this.age = \_age;

16 this.weight=\_weight;

17 this.height=\_height;

18 document.writeln("this.name="+this.name+",this.age="+this.age+",this.weight="+this.weight+",this.height="+this.height);

19 },

20 /\*public方法\*/

21 show:function(){

22 document.writeln("show method");

23 }

24 };

测试代码：

1 document.write("<pre>");

2 var p2\_1 = new Person2();

3 var p2\_2 = new Person2();

4 p2\_1.init("孤傲苍狼",24,115,160);

5 p2\_2.init("白虎神皇",25,120,170);

6 document.writeln("p2\_1.name="+p2\_1.name+"，p2\_1.age="+p2\_1.age+"，p2\_1.weight="+p2\_1.weight+"，p2\_1.height="+p2\_1.height);//访问公有属性，这是可以正常访问的

7 document.writeln("p2\_2.name="+p2\_2.name+"，p2\_2.age="+p2\_2.age+"，p2\_2.weight="+p2\_2.weight+"，p2\_2.height="+p2\_2.height);//访问公有属性，这是可以正常访问的

8 document.writeln("p2\_1 instanceof Person2的结果是："+(p2\_1 instanceof Person2));//p2\_1是Person2类的实例，结果是true

9 document.writeln("p2\_2 instanceof Person2的结果是："+(p2\_2 instanceof Person2));//p2\_2是Person2类的实例，结果是true

10 //当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等

11 document.writeln("当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等");

12 document.writeln("比较p2\_1和p2\_2这两个对象的init方法的内存地址是否一样：p2\_1.init == p2\_2.init的结果是："+(p2\_1.init == p2\_2.init));//true

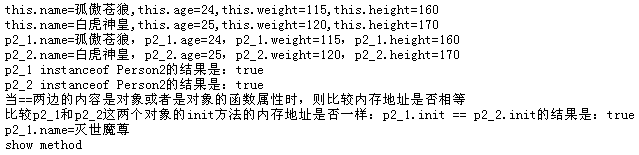
13 p2\_1.name="灭世魔尊";//为公共属性重新赋值

14 document.writeln("p2\_1.name="+p2\_1.name);//访问公有属性，这是可以正常访问的

15 p2\_1.show();//调用类的公共函数，这次允许的

16 document.write("</pre>");

测试结果：



 　　原型方式的优点：所有对象实例都共享类中定义的方法，这样就没有造成内存浪费。缺点，第一，不能定义类的私有属性和私有方法，第二，给在创建对象，给对象的属性初始化时，需要额外写一个初始化对象的方法。

**3、构造函数+原型**

构造函数方式和原型方式都有各自的优缺点，因此可以把这两种方式合并起来，用构造函数方式来定义类的属性(public属性，private属性)，用原型方式来定义类的方法(public方法)。互补不足，这就有了第三种写法。

1 /\*定义一个Person类\*/

2 function Person(\_name,\_age,\_salary){

3 //在Person类内部定义类的public属性和private属性以及private方法

4 //Person类的公开属性，类的公开属性的定义方式是：”this.属性名“

5 this.name=\_name;

6 //Person类的私有属性，类的私有属性的定义方式是：”var 属性名“

7 var age=\_age;//私有属性，只能在类内部使用

8 var salary=\_salary;//私有属性，只能在类内部使用

9 /\*

10 定义Person类的私有方法(内部方法)，只能在类内部使用

11 类的私有方法的定义方式是：”function functionName(){.....}“，

12 或者 var functionName=function(){....}

13 \*/

14 function privateFn(){

15 document.write("<pre>");

16 document.writeln("我是Person类的私有属性age，只能在Person类内部使用，初始化后age="+age);

17 document.writeln("我是Person类的私有函数privateFn，只能在Person类内部使用");

18 document.write("</pre>");

19 }

20

21 var privateFn2=function(){

22 document.write("<pre>");

23 document.writeln("我是Person类的私有属性salary，只能在Person类内部使用，初始化后salary="+salary);

24 document.writeln("我是Person类的私有函数privateFn2，只能在Person类内部使用");

25 document.write("</pre>");

26 }

27

28 privateFn();//在Person类内部调用私有方法

29 privateFn2();//在Person类内部调用私有方法

30 }

31

32 //使用prototype原型方式定义的方法(public方法)是无法访问类的私有属性和私有方法的

33 //使用prototype原型方式定义Person类的方public方法

34 Person.prototype={

35 setName:function(\_name){

36 this.name = \_name;

37 //privateFn();//不能调用Person类定义的私有方法privateFn()，会报错：缺少对象

38 },

39 getName:function(){

40 return this.name;

41 },

42 show:function(){

43 document.writeln("公开方法show");

44 },

45 //公共方法

46 publicMethod:function(){

47 document.writeln("公开方法publicMethod");

48 }

49 };

测试代码：

1 var p1 = new Person("孤傲苍狼",24,2300);

2 var p2 = new Person("白虎神皇",25,3000);

3 document.write("<pre>");

4 document.writeln("p1 instanceof Person的结果是："+(p1 instanceof Person));//p1是Person类的实例，结果是true

5 document.writeln("p2 instanceof Person的结果是："+(p2 instanceof Person));//p2是Person类的实例，结果是true

6 //当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等

7 document.writeln("当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等");

8 document.writeln("比较p1和p2这两个对象的show方法的内存地址是否一样：p1.show== p2.show的结果是："+(p1.show == p2.show));//true

9 document.writeln("p1.show == p2.show的结果是："+(p1.show == p2.show)+"，这证明p1对象和p2对象共享一个show方法，在内存中show方法的代码有1份，存放在1块内存区域");

10 document.writeln("name是Person类定义的public属性，可以使用类的对象去直接访问类的public属性");

11 document.writeln("p1.name="+p1.name);//访问公有属性，这是可以正常访问的

12 document.writeln("age和salary是Person类定义的private属性，不能使用类的对象去直接访问类私有属性，这是访问不了的，结果都是undefined");

13 document.writeln("p1.age="+p1.age+"，"+"p1.salary="+p1.salary)//不能使用类的对象去直接访问类私有属性，这是访问不了的，结果都是undefined

14 p1.show();//调用类的公共函数，这次允许的

15 p1.publicMethod();//调用类的公共函数，这次允许的

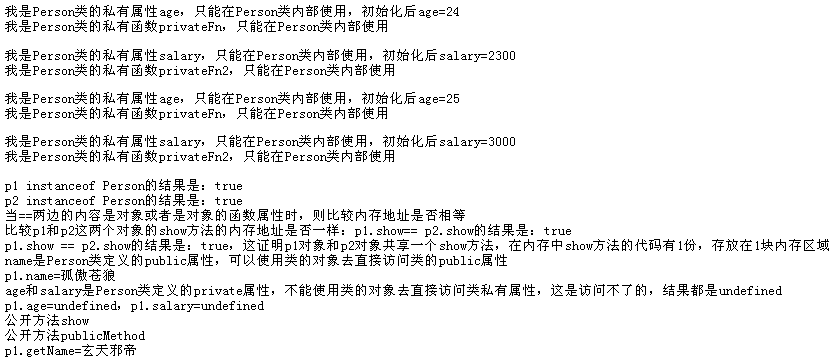
16 p1.setName("玄天邪帝");//调用类的公共函数设置为name属性重新赋值

17 document.writeln("p1.getName="+p1.getName());

18 //document.writeln("p1.privateFn()："+p1.privateFn()+"&nbsp;p1.privateFn2()："+p1.privateFn2());//不能使用类的对象去调用类的私有方法，这里会报错”对象不支持此属性或者方法

19 document.write("</pre>");

运行结果：



第三种方式通过前两种方式的结合，算是达到了一个比较理想的写法了，可以通过传参构造对象实例，对象实例都共享同一份方法不造成内存浪费。第三种方式在开发中用得最多，我本人也是采用这种方式来编写JavaScript类。

**极简主义法编写JavaScript类**

前两天在网上无意中发现了一篇使用极简主义法定义JavaScript类的文章，[原文链接](http://www.ruanyifeng.com/blog/2012/07/three_ways_to_define_a_javascript_class.html)，这个所谓的"极简主义法"我还是第一次听说，是荷兰程序员Gabor de Mooij提出来的，这种方法不使用this和prototype，代码部署起来非常简单，这大概也是它被叫做"极简主义法"的原因。下面就介绍如何使用极简主义法完成JavaScript的封装和继承

**1. 封装**

　　首先，它也是用一个对象模拟"类"。在这个类里面，定义一个构造函数createNew()，用来生成实例。

1 var Cat = {

2 　　　　createNew: function(){

3 　　　　　　// some code here

4 　　　　}

5 };

然后，在createNew()里面，定义一个实例对象，把这个实例对象作为返回值。

1 var Cat = {

2 　　　　createNew: function(){

3 　　　　　　var cat = {};

4 　　　　　　cat.name = "大毛";

5 　　　　　　cat.makeSound = function(){ alert("喵喵喵"); };

6 　　　　　　return cat;

7 　　　　}

8 };

使用的时候，调用createNew()方法，就可以得到实例对象。

1 var cat1 = Cat.createNew();

2 cat1.makeSound(); // 喵喵喵

这种方法的好处是，容易理解，结构清晰优雅，符合传统的"面向对象编程"的构造，因此可以方便地部署下面的特性。

**2. 继承**

　　让一个类继承另一个类，实现起来很方便。只要在前者的createNew()方法中，调用后者的createNew()方法即可。

　　先定义一个Animal类：

1 var Animal = {

2 　　　　createNew: function(){

3 　　　　　　var animal = {};

4 　　　　　　animal.sleep = function(){ alert("睡懒觉"); };

5 　　　　　　return animal;

6 　　　　}

7 };

然后，在Cat的createNew()方法中，调用Animal的createNew()方法

1 var Cat = {

2 　　　　createNew: function(){

3 　　　　　　var cat = **Animal.createNew()**;

4 　　　　　　cat.name = "大毛";

5 　　　　　　cat.makeSound = function(){ alert("喵喵喵"); };

6 　　　　　　return cat;

7 　　　　}

8 };

这样得到的Cat实例，就会继承Animal类。

1 var cat1 = Cat.createNew();

2 cat1.sleep(); // 睡懒觉

**3. 私有属性和私有方法**

**在createNew()方法中，只要不是定义在cat对象上的方法和属性，都是私有的。**

1 var Cat = {

2 　　　　createNew: function(){

3 　　　　　　var cat = {};

4 　　　　　　var sound = "喵喵喵";//私有属性

5 　　　　　　cat.makeSound = function(){

6 alert(sound);

7 };

8 　　　　　　return cat;

9 　　　　}

10 };

上例的内部变量sound，外部无法读取，只有通过cat的公有方法makeSound()来读取。

1 var cat1 = Cat.createNew();

2 alert(cat1.sound); // undefined

**4. 数据共享**

有时候，我们需要所有实例对象，能够读写同一项内部数据。这个时候，只要把这个内部数据，封装在类对象的里面、createNew()方法的外面即可。

1 var Cat = {

2 　　　　sound : "喵喵喵",

3 　　　　createNew: function(){

4 　　　　　　var cat = {};

5 　　　　　　cat.makeSound = function(){ alert(Cat.sound); };

6 　　　　　　cat.changeSound = function(x){ Cat.sound = x; };

7 　　　　　　return cat;

8 　　　　}

9 };

然后，生成两个实例对象：

1 var cat1 = Cat.createNew();

2 var cat2 = Cat.createNew();

3 cat1.makeSound(); // 喵喵喵

这时，如果有一个实例对象，修改了共享的数据，另一个实例对象也会受到影响。

1 cat2.changeSound("啦啦啦");

2 cat1.makeSound(); // 啦啦啦

极简主义，看起来很美好，但是也有缺点，首先是不能使用instanceof 判断对象所属的类，"cat1 instanceof Cat"无法通过，另外，极简主义虽然摆脱了使用原型链的缺点（属性不能私有、创建、继承对象不够直观），但也暴露了没用原型链的弊端：每一次生成一个实例，都必须为重复的内容，多占用一些内存。