1. 什么是面向对象：

面向对象OOP是一种通用的思想，具有封装、继承、多态三大特性

1. 对象的引入：

<script>

var arr = [1,2,3,4,5]; //数组对象

//arr.length //数组的属性

//arr.push(6); //数组的函数

var a=12; //变量: 自由的

arr.a = 5; //属性: 属于一个对象

//alert(arr.a);

//函数 : 属于window

function show(){

alert('a');

};

show();

//方法 ： 属于一个对象

arr.fn=function(){

alert('b');

};

arr.fn();

</script>

1. 对象的组成：属性，方法 （对象的属性和方法是对象特有的）

如：

<script>

var arr=[12,65,78];

//this: 当前的方法， 属于谁

arr.show=function(){

alert(this.length);

//alert(this);

};

arr.show(); //对象的方法只能通过对象调用

/\*

oDiv.onclick=function(){

alert(this);

};

\*/

</script>

实际上，JS中的普通函数也是对象的方法，是window对象的方法

<script>

function show(){

alert(this);

};

//show(); //window

//window.show();

等效于：

window.show=function(){

alert(this);

};

show(); //window

1. 创建空白对象：以上都是系统自带的对象，我们可以new Object()定义自己的对象

eg:

<script>

//Object: 空白对象

var obj = new Object(); //创建对象

/\*

//属性

obj.aaa = 12; //添加对象属性

//方法

obj.show=function(){ //添加对象方法

alert(this.aaa);

};

obj.show();\*/

//如：给对象添加name，sex属性

obj.name = '如花';

obj.sex = '男';

//给对象添加showName,showSex方法

obj.showName=function(){

alert('我的名字:'+this.name);

};

obj.showSex=function(){

alert('我的名字:'+this.sex);

};

//通过对象调用对象中的方法

obj.showName();

obj.showSex();

/\*\*如果还想要创建一个同样的对象，则需要再new Object一次\*\*\*\*/

var obj2 = new Object();

obj2.name = '花花';

obj2.sex = '女';

obj2.showName=function(){

alert('我的名字:'+this.name);

};

obj2.showSex=function(){

alert('我的名字:'+this.sex);

};

obj2.showName();

obj2.showSex();

</script>

对于同一类对象，如果多次用到，重复创建非常的消耗资源，可以使用工厂方法构造对象，如对上面的案例进行优化：

<script>

function createPerson(name,sex){ //创建一个工厂方法，封装对象

//1.创建对象（原料）

var obj=new Object();

//2.赋值 （加工）

obj.name=name;

obj.sex=sex;

obj.showName=function(){

alert(this.name);

}

obj.showSex=function(){

alert(this.sex);

}

//3.返回(出厂)

return obj;

}

var p1=createPerson("zhangsan","男");

var p2=createPerson("lisi","女");

//上面所创建的对象p1和p2仍是不同的对象，其中各个对象都有自己的资源(函数), alert(p1.showName==p2.showName)返回的是false

//而且在创建对象的时候只是调用了一个函数，并未给使用者有new的直观感受

p1.showName();

p2.showName();

</script>

//上面的工厂方法仍存在两个问题：

1. 没有new

2. 每个对象都有自己的函数(资源)

1. 没有new的问题解决：使用this关键字

this关键字的含义：当前的方法属于谁，就指向谁

函数前面有new的时候，指向该函数赋值的对象

**<script>**

**function createPerson(name,sex){**

**//函数在外界被new时，隐含了**

**// this=new Object();**

**this.name=name;**

**this.sex=sex;**

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

**alert(this.name);**

**}**

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

**alert(this.sex);**

**}**

**//当该函数在外界被new的时候，隐含了**

**//return this;**

**}**

**var p1=new createPerson("zhangsan","男"); //称createPerson为构造方法**

**var p2=new createPerson("lisi","女");**

**p1.showName();**

**p2.showName();**

**</script>**

1. 每个对象都有自己的函数(资源)的问题解决：

引入Prototype原型：案例，给数组添加求和方法

<script>

var arr1=new Array(1,2,3,4,5);

//求数组的和

arr1.sum=function(){ //匿名函数：没有名字的函数

var result=0;

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

result+=this[i];

}

alert(result);

}

arr1.sum();//调用

这时如果还有一个数组，也想求所有元素的的和，这个时候需要在写一遍求和方法，非常的消耗资源，有没有一种方式，让求和方法成为数组的共性方法，只有是数组对象，就都具备了求和功能呢。这就需要用到原型prototype

//arr2=[3,4,5,6];

</script>

使用原型优化：

**<script>**

**var arr1=[2,3,4,6];**

**Array.prototype.sum=function(){ //原型函数是所有该类对象共有的方法**

**var result=0;**

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

**result+=this[i];**

**}**

**alert(result);**

**}**

**arr1.sum();**

**var arr2=[6,7,8];**

**arr2.sum();**

**</script>**

原型不仅可以定义函数，也可以定义属性

1. 优化上面createPerson的对象：

**<script>**

**function Person(name,sex){//Person的构造方法**

**this.name=name;**

**this.sex=sex;**

**}**

**//原型函数，Person对象的共有函数**

**Person.prototype.showName=function(){**

**alert(this.name);**

**}**

**Person.prototype.showSex=function(){**

**alert(this.sex);**

**}**

**var p1=new Person("zhangsang","男");//创建对象**

**var p2=new Person("lisi","女");**

**p1.showName();//对象调用函数**

**p2.showName();**

**alert(p1.showName()==p2.showName());//现在这两个函数是否相等呢？**

**</script>**

**一般形象地将上面的构造函数Person又称为类，类名一般首字母大写**

**类 对象**

**模子 蛋糕**

**例如:**

**var arr=new Array();**

**Array类 不具备实际功能, 只能用来构造对象**

**arr对象 具备真正功能, 被类给构造出来**

**在JS中类的本质上是函数，可以通过typeof查看系统的任何一个类的类型，可以发现它们都是function类型的。**

**alert(typeof Array); //function**

**alert(typeof Date);**

1. prototype属性和对象自己的属性之间的优先级：

**<script>**

**var arr=new Array(1,2,4);**

**arr.a=10;//数组对象属性a**

**Array.prototype.a=90;//数组原型属性a**

**alert(arr.a);//对象属性和原型属性名重名时，对象属性优先级高，弹出10**

**</script>**

**可以通过 delete (arr.a); 删除优先级高的属性**

**var arr=new Array(1,2,4);**

**arr.a=10;//数组对象属性a**

**Array.prototype.a=90;//数组原型属性a**

**delete(arr.a);//删除优先级高的属性**

**alert(arr.a);//弹出90**