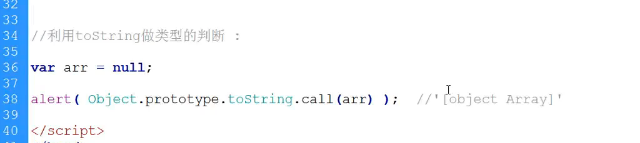
*//面向对象第一种方式  
// var obj = {};***var *obj1*** = **new** Object();  
***obj1***.**name** = **'zs'**;  
***obj1***.showName = **function** () {  
 ***console***.log(**this**.**name**);  
};  
  
*//工厂方式***function** *createPerson*(name) {  
 **var** obj2 = **new** Object();  
 obj2.**name** = name;  
 obj2.showName = **function** () {  
 ***console***.log(**this**.**name**);  
 }  
 **return** obj2;  
}  
  
  
*// new去调用一个函数，函数中的this就是创建出来的对象，函数的返回值就是this（隐式返回）  
//构造函数***function** *CreateCat*(name) {  
 **this**.**name** = name;  
 **this**.showName = **function** () {  
 ***console***.log(**this**.**name**);  
 }  
}  
**var *c1*** = **new** *CreateCat*(**'lisi'**);  
**var *c2*** = **new** *CreateCat*(**'zs'**);  
***c1***.showName();  
***c2***.showName();  
***console***.log(***c1***.showName == ***c2***.showName);  
  
*//原型：去改写对象下面的公用的方法 、 属性，让其在内存中只存在一份，以提高性能。（属性大多情况不一样，方法都一致）***var *arr1*** = [1,2,3];  
**var *arr2*** = [4,5,6,7];  
*// arr1.sum = function () {  
// var result = 0;  
// for(var i = 0; i < this.length; i++){  
// result += this[i];  
// }  
// }  
// //arr1 具备了方法sum，而arr2没有，我们想让arr1 arr2都有，那就需要在原型上添加****arr***.**prototype**.sum = **function** () {  
 ***console***.log(**'共同的方法'**);  
};  
*//注意：普通方法的优先级要高于原型的方法  
  
//包装对象：基本类型String、Number、Boolean都有自己的包装对象  
// var str= new String();***var *str*** = **'hello'**;  
String.**prototype**.lastVlaue = **function** () {  
 **return this**.charAt(**this**.**length** - 1);  
}  
***str***.lastVlaue();*//str具备了 lasstVlaue方法*

***arr*** = [1,2,3];  
***console***.log(**arr**.**constructor**);  
  
*//constructor 属性是程序自带的，JS会自动在prototype下添加一个属性constructor，而这个属性指向了构造函数***function** *Aaa*() {  
 ***console***.log(**'Aaaa'**);  
}  
*// Aaa.prototype.constructor = Aaa; //自动生成***var *a1*** = **new** *Aaa*();  
***console***.log(***a1***.*constructor*);  
  
*Aaa*.**prototype**.**name** = **'lisi'**;  
  
*//这种做法等于将原型对象的地址内容进行了替换，而不是像 Aaa.prototype.name 只是添加。  
// Aaa.prototype = {  
// name: 'lisi'  
// }  
//正确做法：额外修正原型对象的constructor属性  
Aaa*.**prototype** = {  
 constructor: *Aaa*,  
 **name**: **'lisi'**}





继承：

*//属性继承：使用call调用父类构造函数***function** *Father*(name) {  
 **this**.**name** = name;  
}  
*Father*.**prototype**.showName = **function** () {  
 ***console***.log(**this**.**name**);  
};  
  
**function** *Son*(name,age) {  
 *Father*.call(**this**,name); *//继承属性* **this**.**age** = age; *//添加子类自己的属性*}  
  
  
*// Son.prototype = Father.prototype; //继承方法，缺点：对象进行了引用，如果修改子类，会影响父类  
// Son.prototype.showAge = function () {}***function** *extend*(obj1,obj2) { *//拷贝方式继承方法* **for**(**var** attr **in** obj2){  
 obj1[attr] = obj2[attr];  
 }  
}  
*extend*(*Son*.**prototype**,*Father*.**prototype**);  
*Son*.**prototype**.showAge = **function** () {  
 ***console***.log(**this**.**age**);  
}  
  
**var *s*** = **new** *Son*(**'lisi'**,30);  
***s***.showName();  
***s***.showAge();

第二种继承方式：一句话实现继承：

(function(){}).prototype = Father.prototype;

Son.prototype = new Fr();

Son.prototype.constructor = Son; //修正指向问题

第三种继承--原型继承  
**var *Father*** = {  
 **name**: **'lisi'**}  
**function** *cloneObje*(obj) {  
 **var** *F* = **function** () {}  
 *F*.**prototype** = obj;  
 **return new** *F*();  
}  
  
**var *son*** = *cloneObje*(***Father***);  
***son***.**name** = **'zs'**;

