创建一个类

1、类的名称，首字母要大写。

2、通过this关键字为类添加属性或方法

例如：var Book=function（id,bookname,price）{

this.id=id;

this.bookname=bookname;

this.price=price;

}

3、通过原型prototype也可以添加属性和方法

Book.prototype.display=function(){}

Book.prototype={

display:function(){}

}

4、这样就是一种封装

实例化对象的时候要用new 关键字

5、问题：通过this添加的属性和方法同在prototype中添加的属性和方法有什么区别

通过this添加的属性和方法是当前对象上添加的。然而JavaScript是一种基于原型prototype的语言，所以每创建一个对象，它都有一个原型用于指向其继承的属性和方法。通过prototype方法并不是对象自身的，所以使用这些方法时，需要通过prototype一级一级的查找通过this定义的属性或者方法是该对象本身拥有的，所以每次通过类创建一个新的对象时，this指向的属性和方法都会得到相应的创建。而通过prototype几次的属性或者方法是每个对象通过prototype访问到，所以我们每次通过类创建一个新对象时这些属性和方法不会再次创建。

6、属性与方法封装（私有属性和方法、共有属性和方法）

由于JavaScript的函数的作用域，声明在函数内部的变量以及方法在外界是访问不到的，通过此特性即可创建类的私有变量和方法。而在函数内部通过this创建的属性和方法，在类创建对象时，每个对象自身都拥有一份并且可以在外部访问到。因此通过this创建的属性可以看作是对象的共有属性和对象的共有方法，而通过this创建的方法，不但可以访问这些对象的共有属性和私有属性。我们看作为特权方法。在对象创建时通过使用特权方法可以初始化对象的一些属性，因此这些在创建对象调用的特权方法也可以看作是构造器。

例如：

var Book=function（id,name,price）

{

var num01=10; //私有变量

function findid(){

}; //私有方法

this.getId=function(){

}

this.getName=function(){

}

this.id=id; 共有属性

this.copy=function(){

}

this.setName(name); //访问器

this.setPrice(price);

}

7、类的静态公有属性和静态公有方法

Book.chinese=true;

Book.resetTime=function(){

};

8、类的公有属性和公有方法

Book.propotype={

isjsbook:false;

display:functin(){

}

}

9、创建对象安全模式

var Book=function（id,name,price）{

if(this instanceof Book)

{

this.id=id;

this.name=name;

this.price=price;

}

else

{

return new Book(id,name,price);

}

}

继承：

1、类式继承

function Person() {

this.superValue = "super";

}

Person.prototype.getSuperValue=function(){

return this.superValue;

};

function Student(){

this.studentValue="student";

}

Student.prototype=new Person();

Student.prototype.getStudentValue= function () {

return this.studentValue;

};

var stu1=new Student();

alert(stu1.getSuperValue());

alert(stu1.getStudentValue());

类式继承的缺点：

·由于子类通过其原型prototype对父类实例化，继承了父类，所以说父类中的共有属性要是引用类型，就会在子类中被所有实例共用，因此一个子类的实例更改子类原型从父类构造函数中继承来的共有属性就会直接影响到其他子类。

·由于子类实现的继承是靠其原型prototype对父类的实例化实现的，因此在创建父类的时候，是无法向父类传递参数的，因此在实例化父类的时候也无法对父类的构造函数内的属性进行初始化。

function Person(){

this.books=['java','javascript','html5'];

}

function Student()

{

}

Student.prototype=new Person();

var stu1=new Student();

var stu2=new Student();

stu2.books;

stu1.books.push('另一本书');

stu2.books;

2、构造函数继承

function Person(name)  
{  
 this.books=['java','javascript','html5'];  
 this.name=name;  
}  
  
Person.prototype.showBooks=function(){  
 return this.books;  
};  
  
function Student(name)  
{  
 Person.call(this,name);  
}  
  
var *stu1*=new Student('汪星人');  
var *stu2*=new Student('猫星人');  
  
*stu1*.books.push('白雪公主');  
  
alert(*stu1*.books);  
alert(*stu1*.name);  
  
alert(*stu2*.books);  
alert(*stu2*.name);

这里call（this,name）语句，是构造函数式继承的精华，由于call这个方法可以更改函数的作用环境，因此在子类中，对person调用这个方法就是将子类中的变量在父类中执行一遍，由于父类中是给this绑定属性的，因此子类自然也就继承了父类的共有属性，由于这种类型的继承没有涉及到原型，所以父类的原型方法自然不会被子类继承，而如果想被子类继承就必须要放在构造函数中，这样创建出来的每个实例都会单独拥有一份而不能共用，这样以来也违背了代码复用的原则。为此要结合使用:称为 组合式继承

3、组合式继承

前两种继承模式的特点：

1）类式继承是通过子类的原型prototype对父类实例化来实现的。

2）构造函数式继承是通过在子类的构造函数作用环境中执行父类的构造函数来实现的。

组合式继承就是实现上述两点：

function Person(name)  
{  
 this.books=['java','javascript','html5'];  
 this.name=name;  
}  
  
Person.prototype.getName=function(){  
 return this.name;  
};  
  
function Student(name,time)  
{  
 Person.call(this,name);  
 this.time=time;  
}  
  
Student.prototype=new Person();  
Student.prototype.getTime=function(){  
 return this.time;  
};  
  
var *stu1*=new Student('汪星人',2016);  
 *stu1*.books.push('新的一本书');  
 alert(*stu1*.books);  
 *stu1*.getName();  
 *stu1*.getTime();  
  
var *stu2*=new Student('猫星人',2017);  
  
alert(*stu2*.books);  
*stu2*.getName();  
*stu2*.getTime();

此类型的继承，父类的构造方法调用了两次，所有还不是完美的继承方式。

4、原型式继承

道格拉斯·克罗福德说：借助原型prototype可以根据已有的对象创建一个新的对象，同时不必创建新的自定义对象类型。

function inheritObject(o)  
{  
 function F(){};  
 F.prototype=o;  
 return new F();  
}  
  
var *book*={  
 name:"jsbook",  
 alikeBook:['css','html']  
};  
  
var *newBook1*=inheritObject(*book*);  
*newBook1*.name='hello';  
*newBook1*.alikeBook.push('xml');  
  
var *newBook2*=inheritObject(*book*);  
*newBook2*.name='hello111';  
*newBook2*.alikeBook.push('xml111');  
  
alert(*newBook1*.name);  
alert(*newBook1*.alikeBook);  
  
alert(*newBook2*.name);  
alert(*newBook2*.alikeBook);  
  
alert(*book*.name);  
alert(*book*.alikeBook);

**设计模式**

1、简单的工厂模式

体用用品店

定义类各个类

篮球类

var Basketball=function(){

this.info='篮球在美国';

};

Basketball.prototype={

getMember: function()

{

console.log('每个队有5名队员 组成');

},

getBallSize:function(){

console.log('篮球比较大');

}

}

足球类

var Football=function(){

this.info='足球在欧洲';

};

Football.prototype={

getMember: function()

{

console.log('每个队有11名队员 组成');

},

getBallSize:function(){

console.log('足球有点小');

}

}

工厂

var sportFactory=function(name)

{

switch(name)

{

case 'NBA':

return new Basketball();

case 'wordCup':

return new Football();

}

}

var football=SportFactory('wordCup');

console.log(football);

console.log(football.info);

football.getMember();

此种模式是通过类实例化对象创建的

2、也是一种工厂模式

function createBook(name,price)

{

var o=new Object();

o.name=name;

o.price=price;

o.getName=fucntion()

{

return this.name;

}

return o;

}

此中方法是通过创建一个新的对象然后包装增强其属性和功能来实现的

3、工厂方法模式

实例：广告投放

<!DOCTYPE html>  
<html>  
<head lang=**"en"**>  
 <meta charset=**"UTF-8"**>  
 <title></title>  
 <script>  
  
 window.onload=function(){  
  
  
 var Factory=function(type,content)  
 {  
 if(this instanceof Factory)  
 {  
 var s= new this[type](content);  
 }else  
 {  
 return new Factory(type,content);  
 }  
 };  
  
 Factory.prototype={  
 Java:function(content)  
 {  
 this.content=content;  
 (  
 function(content){  
 var Odiv=document.createElement('div');  
 Odiv.innerHTML=content;  
 Odiv.style.border='1px solid red';  
 Odiv.style.width='400px';  
 Odiv.style.height='300px';  
 Odiv.style.float='left';  
 document.getElementById('container').appendChild(Odiv);  
 }  
  
 )(content);  
 },  
  
 Javascript:function(content)  
 {  
 this.content=content;  
 (  
 function(content){  
 var Odiv=document.createElement('div');  
 Odiv.innerHTML=content;  
 Odiv.style.border='1px solid blue';  
 Odiv.style.width='400px';  
 Odiv.style.height='300px';  
 Odiv.style.float='left';  
 document.getElementById('container').appendChild(Odiv);  
 }  
  
 )(content);  
 },  
  
 Php: function(content)  
 {  
 this.content=content;  
 (  
 function(content){  
 var Odiv=document.createElement('div');  
 Odiv.innerHTML=content;  
 Odiv.style.border='1px solid green';  
 Odiv.style.width='400px';  
 Odiv.style.height='300px';  
 Odiv.style.float='left';  
 document.getElementById('container').appendChild(Odiv);  
 }  
  
 )(content);  
 }  
 };  
  
 var data=[{type:'Java',content:'Java哪家好'},  
 {type:'Javascript',content:'Javascript哪家好'},  
 {type:'Php',content:'php哪家好'}];  
  
 for(var i=2;i>=0;i--)  
 {  
 Factory(data[i].type, data[i].content);  
 }  
  
 };  
 </script>  
</head>  
<body>  
<div id=**"container"**>  
  
</div>  
</body>  
</html>

主要用于创建不同种类的对象

思考：创建不同的按钮