**介绍**

构造函数大家都很熟悉了，不过如果你是新手，还是有必要来了解一下什么叫构造函数的。构造函数用于创建特定类型的对象——不仅声明了使用的对象，构造函数还可以接受参数以便第一次创建对象的时候设置对象的成员值。你可以自定义自己的构造函数，然后在里面声明自定义类型对象的属性或方法。

**基本用法**

在JavaScript里，构造函数通常是认为用来实现实例的，JavaScript没有类的概念，但是有特殊的构造函数。通过new关键字来调用定义的否早函数，你可以告诉JavaScript你要创建一个新对象并且新对象的成员声明都是构造函数里定义的。在构造函数内部，this关键字引用的是新创建的对象。基本用法如下：

function Car(model, year, miles) {  
 this.model = model;  
 this.year = year;  
 this.miles = miles;  
 this.output= function () {  
 return this.model + "走了" + this.miles + "公里";  
 };  
}  
var tom= new Car("大叔", 2009, 20000);  
var dudu= new Car("Dudu", 2010, 5000);  
  
console.log(tom.output());  
console.log(dudu.output());

上面的例子是个非常简单的构造函数模式，但是有点小问题。首先是使用继承很麻烦了，其次output()在每次创建对象的时候都重新定义了，最好的方法是让所有Car类型的实例都共享这个output()方法，这样如果有大批量的实例的话，就会节约很多内存。

解决这个问题，我们可以使用如下方式：

function Car(model, year, miles) {  
 this.model = model;  
 this.year = year;  
 this.miles = miles;  
 this.output= formatCar;  
}  
  
function formatCar() {  
 return this.model + "走了" + this.miles + "公里";  
}

这个方式虽然可用，但是我们有如下更好的方式。

**构造函数与原型**

JavaScript里函数有个原型属性叫prototype，当调用构造函数创建对象的时候，所有该构造函数原型的属性在新创建对象上都可用。按照这样，多个Car对象实例可以共享同一个原型，我们再扩展一下上例的代码：

function Car(model, year, miles) {  
 this.model = model;  
 this.year = year;  
 this.miles = miles;

}  
  
/\*  
注意：这里我们使用了Object.prototype.方法名，而不是Object.prototype  
主要是用来避免重写定义原型prototype对象  
\*/  
Car.prototype.output= function () {  
 return this.model + "走了" + this.miles + "公里";  
};  
  
var tom = new Car("大叔", 2009, 20000);  
var dudu = new Car("Dudu", 2010, 5000);  
  
console.log(tom.output());  
console.log(dudu.output());

这里，output()单实例可以在所有Car对象实例里共享使用。

另外：我们推荐构造函数以大写字母开头，以便区分普通的函数。

**只能用new吗？**

上面的例子对函数car都是用new来创建对象的，只有这一种方式么？其实还有别的方式，我们列举两种：

function Car(model, year, miles) {  
 this.model = model;  
 this.year = year;  
 this.miles = miles;  
 // 自定义一个output输出内容  
 this.output = function () {  
 return this.model + "走了" + this.miles + "公里";  
 }  
}  
  
//方法1：作为函数调用  
Car("大叔", 2009, 20000); //添加到window对象上  
console.log(window.output());  
  
//方法2：在另外一个对象的作用域内调用  
var o = new Object();  
Car.call(o, "Dudu", 2010, 5000);  
console.log(o.output());

该代码的方法1有点特殊，如果不适用new直接调用函数的话，this指向的是全局对象window，我们来验证一下：

//作为函数调用  
var tom = Car("大叔", 2009, 20000);  
console.log(typeof tom); // "undefined"  
console.log(window.output()); // "大叔走了20000公里"

这时候对象tom是undefined，而window.output()会正确输出结果，而如果使用new关键字则没有这个问题，验证如下：

//使用new 关键字  
var tom = new Car("大叔", 2009, 20000);  
console.log(typeof tom); // "object"  
console.log(tom.output()); // "大叔走了20000公里"

**强制使用new**

上述的例子展示了不使用new的问题，那么我们有没有办法让构造函数强制使用new关键字呢，答案是肯定的，上代码：

function Car(model, year, miles) {  
 if (!(this instanceof Car)) {  
 return new Car(model, year, miles);  
 }  
 this.model = model;  
 this.year = year;  
 this.miles = miles;  
 this.output = function () {  
 return this.model + "走了" + this.miles + "公里";  
 }  
}  
  
var tom = new Car("大叔", 2009, 20000);  
var dudu = Car("Dudu", 2010, 5000);  
  
console.log(typeof tom); // "object"  
console.log(tom.output()); // "大叔走了20000公里"  
console.log(typeof dudu); // "object"  
console.log(dudu.output()); // "Dudu走了5000公里"

通过判断this的instanceof是不是Car来决定返回new Car还是继续执行代码，如果使用的是new关键字，则(this instanceof Car)为真，会继续执行下面的参数赋值，如果没有用new，(this instanceof Car)就为假，就会重新new一个实例返回。

**原始包装函数**

JavaScript里有3中原始包装函数：number, string, boolean，有时候两种都用：

// 使用原始包装函数  
var s = new String("my string");  
var n = new Number(101);  
var b = new Boolean(true);  
  
  
// 推荐这种  
var s = "my string";  
var n = 101;  
var b = true;

推荐，只有在想保留数值状态的时候使用这些包装函数，关于区别可以参考下面的代码：

// 原始string  
var greet = "Hello there";  
// 使用split()方法分割  
greet.split(' ')[0]; // "Hello"  
// 给原始类型添加新属性不会报错  
greet.smile = true;  
// 单没法获取这个值

console.log(typeof greet.smile); // "undefined"  
  
// 原始string  
var greet = new String("Hello there");  
// 使用split()方法分割  
greet.split(' ')[0]; // "Hello"  
// 给包装函数类型添加新属性不会报错  
greet.smile = true;  
// 可以正常访问新属性  
console.log(typeof greet.smile); // "boolean"