

## 函数属性和方法

前面曾经提到过，ECMAScript 中的函数是对象，因此函数也有属性和方法。每个函数都包含两个属性：`length` 和 `prototype`。其中，`length` 属性表示函数希望接收的命名参数的个数，如下面的例子所示。

```
function sayName(name){
    alert(name);
}

function sum(num1, num2){
    return num1 + num2;
}

function sayHi(){
    alert("hi");
}

alert(sayName.length);    //1
alert(sum.length);        //2
alert(sayHi.length);      //0
```

在 ECMAScript 核心所定义的全部属性中，最耐人寻味的就要数 `prototype` 属性了。对于 ECMAScript 中的引用类型而言，`prototype` 是保存它们所有实例方法的真正所在。换句话说，诸如 `toString()` 和 `valueOf()` 等方法实际上都保存在 `prototype` 名下，只不过是各自对象的实例访问罢了。在创建自定义引用类型以及实现继承时，`prototype` 属性的作用是极为重要的（第 6 章将详细介绍）。在 ECMAScript 5 中，`prototype` 属性是不可枚举的，因此使用 `for-in` 无法发现。

每个函数都包含两个非继承而来的方法：`apply()` 和 `call()`。这两个方法的用途都是在特定的作用域中调用函数，实际上等于设置函数体内 `this` 对象的值。首先，`apply()` 方法接收两个参数：一个是在其中运行函数的作用域，另一个是参数数组。其中，第二个参数可以是 `Array` 的实例，也可以是 `arguments` 对象。例如：

这两个方法的用途都是在特定的作用域中调用函数，实际上等于设置函数体内 `this` 对象的值。

```
function sum(num1, num2){
    return num1 + num2;
}

function callSum1(num1, num2){
    return sum.apply(this, arguments);    // 传入 arguments 对象
}

function callSum2(num1, num2){
    return sum.apply(this, [num1, num2]);    // 传入数组
}

alert(callSum1(10,10));    //20
alert(callSum2(10,10));    //20
```

*FunctionTypeApplyMethodExample01.*



在严格模式下，未指定环境对象而调用函数，则 `this` 值不会转型为 `window`。除非明确把函数添加到某个对象或者调用 `apply()` 或 `call()`，否则 `this` 值将是 `undefined`。

`call()` 方法与 `apply()` 方法的作用相同，它们的区别仅在于接收参数的方式不同。对于 `call()` 方法而言，第一个参数是 `this` 值没有变化，变化的是其余参数都直接传递给函数。换句话说，在使用 `call()` 方法时，传递给函数的参数必须逐个列举出来，如下面的例子所示。

```
function sum(num1, num2){
    return num1 + num2;
}

function callSum(num1, num2){
    return sum.call(this, num1, num2);
}

alert(callSum(10,10));    //20
```

事实上，传递参数并非 `apply()` 和 `call()` 真正的用武之地；它们真正强大的地方是能够扩充函数赖以运行的作用域。下面来看一个例子。

事实上，传递参数并非 `apply()` 和 `call()` 真正的用武之地；它们真正强大的地方是能够扩充函数赖以运行的作用域。下面来看一个例子。

```
window.color = "red";
var o = { color: "blue" };

function sayColor(){
    alert(this.color);
}

sayColor();                //red

sayColor.call(this);        //red
sayColor.call(window);      //red
sayColor.call(o);           //blue
```

使用 `call()`（或 `apply()`）来扩充作用域的最大好处，就是对象不需要与方法有任何耦合关系。在前面例子的第一个版本中，我们是先将 `sayColor()` 函数放到了对

象 o 中，然后再通过 o 来调用它的；而在这里重写的例子中，就不需要先前那个多余的步骤了。

ECMAScript 5 还定义了一个方法：bind()。这个方法会创建一个函数的实例，其 this 值会被绑定到传给 bind() 函数的值。例如：

```
window.color = "red";
var o = { color: "blue" };

function sayColor(){
    alert(this.color);
}
var objectSayColor = sayColor.bind(o);
objectSayColor();    //blue
```

每个函数继承的 toLocaleString() 和 toString() 方法始终都返回函数的代码。

返回代码的格

式则因浏览器而异——有的返回的代码与源代码中的函数代码一样，而有的则返回函数代码的内部表示，即由解析器删除了注释并对某些代码作了改动后的代码。由于存在这些差异，我们无法根据这两个方法返回的结果来实现任何重要功能；不过，这些信息在调试代码时倒是很有用。另外一个继承的valueOf() 方法同样也只返回函数代码。