函数内部属性

在函数内部，有两个特殊的对象：arguments 和 this。其中，arguments 在第 3 章曾经介绍过，它是一个类数组对象，包含着传入函数中的所有参数。虽然 arguments 的主要用途是保存函数参数，但这个对象还有一个名叫 callee 的属性，该属性是一个指针，指向拥有这个 arguments 对象的函数。请看下面这个非常经典的阶乘函数。

```
function factorial(num){
    if (num <=1) {
        return 1;
    } else {
        return num * factorial(num-1)
    }
}
```

定义阶乘函数一般都要用到递归算法；如上面的代码所示，在函数有名字，而且名字以后也不会变的情况下，这样定义没有问题。但问题是这个函数的执行与函数名 factorial 紧紧耦合在了一起。为了消除这种紧密耦合的现象，可以像下面这样使用 arguments.callee。

```
function factorial(num){
    if (num <=1) {
        return 1;
    } else {
        return num * arguments.callee(num-1)
    }
}
```

arguments.callee指向函数

函数内部的另一个特殊对象是 this ，其行为与 Java 和 C#中的 this 大致类似。换句话说， this
引用的是函数据以执行的环境对象——或者也可以说是 this 值（当在网页的全局作用域中调用函数时，this 对象引用的就是 window ）。来看下面的例子。

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

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

sayColor();      //"red"

o.sayColor = sayColor;
o.sayColor();    //"blue"
```

请读者一定要牢记，函数的名字仅仅是一个包含指针的变量而已。因此，即使是在不同的环境中执行，全局的 sayColor()函数与 o.sayColor()指向的仍然是同一个函数。

ECMAScript 5 也规范化了另一个函数对象的属性：caller。除了 Opera 的早期版本不支持，其他浏览器都支持这个 ECMAScript 3 并没有定义的属性。这个属性中保存着调用当前函数的函数的引用，如果是在全局作用域中调用当前函数，它的值为 null。例如：

```
function outer(){
    inner();
}

function inner(){
    alert(inner.caller);
}

outer();
```

以上代码会导致警告框中显示 outer()函数的源代码。因为 outer()调用了 inter()，所以 inner.caller 就指向 outer()。为了实现更松散的耦合，也可以通过 arguments.callee.caller 来访问相同的信息。

```
function outer(){
    inner();
}

function inner(){
    alert(arguments.callee.caller);
}

outer();
```