ECMAScript 函数的参数与大多数其他语言中函数的参数有所不同。ECMAScript 函数不介意传递进

来多少个参数，也不在乎传进来参数是什么数据类型。也就是说，即便你定义的函数只接收两个参数，在调用这个函数时也未必一定要传递两个参数。可以传递一个、三个甚至不传递参数，而解析器永远不会有什么怨言。之所以会这样，原因是 ECMAScript 中的参数在内部是用一个数组来表示的。在函数体内可以通过 arguments 对象来访问这个参数数组，从而获取传递给函数的每一个参数。

name，而该参数的值也可以通过访问 arguments[0]来获取。因此，那个函数也可以像下面这样重写，即不显式地使用命名参数：

```
function sayHi() {
    alert("Hello " + arguments[0] + "," + arguments[1]);
}
```

这个事实说明了 ECMAScript 函数的一个重要特点：命名的参数只提供便利，但不是必需的。

执行以上代码会依次出现 3 个警告框，分别显示 2、0 和 1。由此可见，开发人员可以利用这一点让函数能够接收任意个参数并分别实现适当的功能。请看下面的例子：

```
function doAdd() {
    if(arguments.length == 1) {
        alert(arguments[0] + 10);
    } else if (arguments.length == 2) {
        alert(arguments[0] + arguments[1]);
    }
}

doAdd(10);          //20
doAdd(30, 20);      //50
```

```
function doAdd(num1, num2) {
    arguments[1] = 10;
    alert(arguments[0] + num2);
}
```

FunctionExample09.htm

每次执行这个 doAdd()函数都会重写第二个参数，将第二个参数的值修改为 10。因为 arguments 对象中的值会自动反映到对应的命名参数，所以修改 arguments[1]，也就修改了 num2，结果它们的值都会变成 10。不过，这并不是说读取这两个值会访问相同的内存空间；它们的内存空间是独立的，但它们的值会同步。另外还要记住，如果只传入了一个参数，那么为 arguments[1]设置的值不会反应到命名参数中。这是因为 arguments 对象的长度是由传入的参数个数决定的，不是由定义函数时的命名参数的个数决定的。

关于参数还要记住最后一点：没有传递值的命名参数将自动被赋予 undefined 值。这就跟定义了变量但又没有初始化一样。例如，如果只给 doAdd()函数传递了一个参数，则 num2 中就会保存 undefined 值。

ECMAScript 中的所有参数传递的都是值，不可能通过引用传递参数。