传递参数

```
function addTen(num) {
    num += 10;
    return num;
}
```



```
var count = 20;
var result = addTen(count);
alert(count);    //20, 没有变化
alert(result);   //30
```

FunctionArgumentsExample01.htm

```
function setName(obj) {
    obj.name = "Nicholas";
}

var person = new Object();
setName(person);
alert(person.name);    //"Nicholas"
```

FunctionArgumentsExample02.htm

以上代码中创建一个对象，并将其保存在了变量 person 中。然后，这个变量被传递到 setName() 函数中之后就被复制给了 obj。在这个函数内部，obj 和 person 引用的是同一个对象。换句话说，即使这个变量是按值传递的，obj 也会按引用来访问同一个对象。于是，当在函数内部为 obj 添加 name 属性后，函数外部的 person 也将有所反映；因为 person 指向的对象在堆内存中只有一个，而且是全局对象。有很多开发人员错误地认为：在局部作用域中修改的对象会在全局作用域中反映出来，就说明参数是按引用传递的。为了证明对象是按值传递的，我们再看一看下面这个经过修改的例子：

```
function setName(obj) {
    obj.name = "Nicholas";
    obj = new Object();
    obj.name = "Greg";
}

var person = new Object();
setName(person);
alert(person.name);    //"Nicholas"
```

这个例子与前一个例子的唯一区别，就是在 setName()函数中添加了两行代码：一行代码为 obj 重新定义了一个对象，另一行代码为该对象定义了一个带有不同值的 name 属性。在把 person 传递给 setName()后，其 name 属性被设置为"Nicholas"。然后，又将一个新对象赋给变量 obj，同时将其 name 属性设置为"Greg"。如果 person 是按引用传递的，那么 person 就会自动被修改为指向其 name 属性值为"Greg"的新对象。但是，当接下来再访问 person.name 时，显示的值仍然是"Nicholas"。这说明即使在函数内部修改了参数的值，但原始的引用仍然保持未变。实际上，当在函数内部重写 obj 时，这个变量引用的就是一个局部对象了。而这个局部对象会在函数执行完毕后立即被销毁。

基础类型不改变原始值，引用类型会改变，因为引用类型和元变量指向堆中同一个对象。

可以把 ECMAScript 函数的参数想象成局部变量。