17.9.10 使用 call()和 apply()

call()和 apply()是 Function 对象的原型方法，它们能够将特定函数当做一个方法绑定到指定对象上并进行调用。具体用法如下：

```
function.call(thisobj, args...)
function.apply(thisobj, args)
```

其中参数 thisobj 表示指定的对象，参数 args 表示要传递给被调用函数的参数。call()方法只能接收多个参数列表，而 apply()只能接收一个数组或者伪类数组，数组元素将作为参数传递给被调用的函数。

【示例 1】当函数被绑定到指定对象上之后，将利用传递的参数执行函数，并返回函数的返回值。

```
function f(x,y){                    // 定义一个简单的函数
    return x+y;
}
function o(a,b){                    // 定义一个函数结构的伪对象
    return a*b;
}
alert(f.call(o,3,4));               // 返回 7
```

在上面示例中，f 是一个简单的函数，而 o 是一个构造函数对象。通过 call()方法把函数 f 绑定到对象 o 身上，变为它的一个方法，然后动态调用函数 f，同时把参数 3 和 4 传递给函数 f，则调用函数 f 后返回值为 7。

实际上，上面示例可以转换为下面代码：

```
function f(x,y){                    // 定义一个简单的函数
    return x+y;
}
function o(a,b){                    // 定义一个函数结构的伪对象
    return a*b;
}
o.m =f;                             // 为对象 o 定义一个方法 m，该方法将调用函数 f
alert(o.m(3,4));                    // 返回 7。调用对象 o 的方法 m
delete o.m;                         // 删除对象 o 的方法 m
```

【示例 2】apply()与 call()方法功能和用法都相同，唯一的区别是它们传递给参数的方式不同。其中 apply()是以数组形式传递参数，而 call()方法以多个值的形式传递参数。针对上面示例，使用 apply()方法来调用函数 f，则设计代码如下所示：

```
function f(x,y){
    return x+y;
}
function o(a,b){
    return a*b;
}
alert(f.apply(o,[3,4]));            // 返回 7
```

【示例 3】设计把一个数组或伪类数组的所有元素作为参数进行传递时，使用 apply()方法就非常便利。

```
function max(){                     // 删除对象 o 的
    var m = Number.NEGATIVE_INFINITY;
      // 声明一个负无穷大的数值
    for( var i = 0; i < arguments.length; i ++ ){
      // 遍历函数所有的实参
       if( arguments[i] > m )       // 如果实参值大于变量 m，
       m = arguments[i];            // 则把该实参值赋值给 m
    }
    return m;                       // 返回最大值
}
var a = [23, 45, 2, 46, 62, 45, 56, 63];
  // 声明并初始化数组
```

```
var m = max.apply( Object, a );   // 把函数 max 绑定为 Object 对象的方法，并动态调用
alert( m );                       // 返回 63
```

在上面示例中，设计定义一个函数 max()，用来计算所有参数中最大值参数。首先通过 apply()方法，动态调用 max()函数，然后把它绑定为 Object 对象的一个方法，并把包含多个值的数组传递给它，最后返回经过 max()计算后的最大数组元素。

如果不使用 call()方法，希望使用 max()函数找出数组中最大值元素，就需要把数组所有元素全部读取出来，再逐一传递给 call()方法，显然这种做法是比较笨拙的。

【示例 4】也可以把数组元素通过 apply()方法传递给 Math 的 max()方法来计算数组的最大值元素。

```
var a = [23, 45, 2, 46, 62, 45, 56, 63];          // 声明并初始化数组
var m = Math.max.apply( Object, a );              // 调用系统函数 max
alert( m );                       // 返回 63
```

【示例 5】使用 call()和 apply()方法可以把一个函数转换为指定对象的方法，并在这个对象上调用该方法。这种行为只是临时的，函数实际上并没有作为对象的方法而存在，当函数被动态调用之后，这个对象的临时方法也会自动被注销。

```
function f(){}                    // 定义空函数
f.call( Object );                 // 把函数 f 绑定为 Object 对象的方法
Object.f();                       // 再次调用该方法，则返回编译错误
```

【示例 6】call()和 apply()方法能够动态改变函数内 this 指代的对象，这在面向对象编程中是非常有用的。下面示例使用 call()方法不断改变函数内 this 指代对象，主要通过变换 calll()方法的第一个参数值来实现。

```
var x = "o";                      // 定义全局变量 x，初始化为字符 o
function a(){                     // 定义函数类结构 a
    this.x = "a";                 // 定义函数内局部变量 x，初始化为字符 a
}
function b(){                     // 定义函数类结构 b
    this.x = "b";                 // 定义函数内局部变量 x，初始化为字符 b
}
function c(){                     // 定义普通函数，提示变量 x 的值
    alert( x );
}
function f(){                     // 定义普通函数，提示当前指针所包含的变量 x 的值
    alert( this.x );
}
// 返回字符 o，即全局变量 x 的值。this 此时指向 window 对象
f();
// 返回字符 o，即全局变量 x 的值。this 此时指向 window 对象
f.call( window );
// 返回字符 a，即函数 a 内部的局部变量 x 的值。this 此时指向函数 a
f.call( new a() );
// 返回字符 b，即函数 b 内部的局部变量 x 的值。this 此时指向函数 b
f.call( new b() );
// 返回 undefined，即函数 c 内部的局部变量 x 的值，但是该函数并没有定义 x 变量，所以返回没有定义。this 此时指向函数 c
f.call( c );
```

【示例 7】在函数体内，call()和 apply()方法的第一个参数就是调用函数内 this 的值。为了更好理解，用户可以看下面示例。

```
function f(){                     // 定义函数类结构
    this.a ="a";                  // 定义成员 a 并赋值，a 为属性
    this.b = function(){          // 定义成员 b 并赋值，b 为方法
        alert("b");
    }
}
function e(){                     // 定义函数
    f.call(this);                 // 在函数体内动态调用函数 f，this 指代函数 e
    alert(a);                     // 显示变量 a 的值
```

```
}
e()                                    // 返回字符串 a
```

上面示例显示，如果在函数体内，使用 call()和 apply()方法动态调用外部函数，并把 call()和 apply()方法的第一个参数值设置为关键字 this，则当前函数 e 将继承函数 f 的所有属性。即使用 call()和 apply()方法能够复制调用函数的内部变量给当前函数体。

【示例 8】在下面示例中，使用 apply()方法循环更改当前 this 指针，从而实现循环更改函数的结构。

```
function r( x ){                       // 定义一个简单的函数
    return ( x );
}
// 定义一个稍复杂的函数，该函数将修改第一个参数值，并返回参数集合
function f( x ){
    x[0] = x[0] + ">";
    return x;
}
function o(){                          // 循环更改函数 r 中返回值
    var temp = r;
    r = function(){
        return temp.apply( this, f( arguments ) );
    }
}
function a(){                          // 定义函数 a
    o();                               // 调用函数 o，修改函数 r 的结构，即返回值
    alert( r( "=" ) );                 // 显示函数 r 的返回值
}
for( var i = 0 ; i < 10; i ++ ){       // 循环调用函数 a
    a();
}
```

执行上面示例，会看到提示信息框中的提示信息不断变化，。该示例的核心就在于函数 o 的设计。在这个函数中，首先使用一个临时变量存储函数 r。然后修改函数 r 的结构，在修改的 r 函数结构中，通过调用 apply()方法修改原来函数 r 的指针指向当前对象，同时执行原函数 r，并把执行函数 f 的值传递给它，从而实现修改函数 r 的 return 语句的后半部分信息，即为返回值增加一个前缀字符“=”。这样每次调用函数 o 时，都会为其增加一个前缀字符“=”，从而形成一种动态的变化效果。