JS中的call、apply、bind方法详解

`github.com/tsrot` `call` `apply` `bind`

地址：http://blog.xieliqun.com/2016/08/10/call-apply-bind/

> call()、apply()、bind()都是函数对象的一个方法，它们的作用都是改变函数的调用对象。它的使用极大的简化了代码的调用。

一、方法定义

call方法

语法：`call([thisObj[,arg1[, arg2[, [,.argN]]]]])`
定义：调用一个对象的一个方法，以另一个对象替换当前对象。
说明：call 方法可以用来代替另一个对象调用一个方法。call 方法可将一个函数的对象上下文从初始的上下文改变为由 thisObj 指定的新对象。如果没有提供 thisObj 参数，那么 Global 对象被用作 thisObj。
arg1 … argN为被调用方法的传参。

apply方法

语法：`apply([thisObj[,argArray]])`
定义：应用某一对象的一个方法，用另一个对象替换当前对象。
说明：apply的第一个参数thisObj和call方法的一样，第二个参数argArray为一个传参数组。thisObj如果未传，那么 Global 对象被用作 thisObj。

bind方法

在ECMAScript5中扩展了叫bind的方法（IE6,7,8不支持）
语法：`bind([thisObj[,arg1[, arg2[, [,.argN]]]]])`
定义：应用某一对象的一个方法，用另一个对象替换当前对象。
说明：bind的thisObj参数也和call方法一样，thisObj如果未传，那么 Global 对象被用作 thisObj。
arg1 … argN可传可不传。如果不传，可以在调用的时候再传。如果传了，调用的时候则可以不传，调用的时候如果你还是传了，则不生效。例如：

```
var person = {
    name:"tsrot",
    age:24,
    sayHello:function(age){
        console.log(this.name);
        console.log(age);
    }
};
var son = {
    name:"xieliqun"
};
var boundFunc = person.sayHello.bind(son);
boundFunc(25); // xieliqun  25
```

```
var boundFunc = person.sayHello.bind(son,25);
boundFunc(); // xieliqun  25
```

```
var boundFunc = person.sayHello.bind(son,25);
boundFunc(30); // xieliqun  25
```

二、call、apply、bind的区别

1、call的arg传参需一个一个传，apply则直接传一个数组。

```
function hello(name,age){
    console.log(name);
    console.log(age);
}
hello.call(this,"tsrot",24);
hello.apply(this,["tsrot",24]);
```

2、call和apply直接执行函数，而bind需要再一次调用。

```
var obj = {
    x: 81,
};

var foo = {
    getX: function() {
        return this.x;
    }
}

console.log(foo.getX.bind(obj)());  //81
console.log(foo.getX.call(obj));    //81
console.log(foo.getX.apply(obj));   //81
```

三、运用场景

实现继承

```
function Animal(name) {
  this.name = name;
  this.showName = function () {
    console.log(this.name);
  }
}

function Cat(name) {
  Animal.call(this, name); //Cat继承了Animal的showName方法
}

var cat = new Cat('Black Cat');
cat.showName(); //Black Cat
```

数组追加

```
var array1 = [1 , 2 , 3, 5];
var array2 = ["xie" , "li" , "qun" , "tsrot"];
Array.prototype.push.apply(array1, array2);
console.log(array1);//[1, 2, 3, 5, "xie", "li", "qun", "tsrot"]
```

获取数组中的最大值和最小值

```
var num = [1,3,5,7,2,-10,11];
var maxNum = Math.max.apply(Math, num);
var minNum = Math.min.apply(Math, num);
console.log(maxNum); //11
console.log(minNum); //-10
```

将伪数组转化为数组

```
var fakeArr = {0:'a',1:'b',length:2};
var arr1 = Array.prototype.slice.call(fakeArr);
console.log(arr1[0]); //a
var arr2 = [].slice.call(fakeArr);
console.log(arr2[0]); //a

arr1.push("c");
console.log(arr1); //["a", "b", "c"]
```

保存this变量

```
// 正常情况下使用变量保存 this 值
var foo = {
    bar : 1,
    eventBind: function(){
        var _this = this ;
        $('.someClass').on('click',function(event) {
            /* Act on the event */
            console.log(_this.bar);     //1
        });
    }
}

// 使用 bind 进行函数绑定
var foo = {
    bar : 1,
    eventBind: function(){
        $('.someClass').on('click',function(event) {
            /* Act on the event */
            console.log(this.bar);      //1
        }.bind(this));
    }
}
```