JS中call、apply、bind使用指南，带部分原理。

为什么需要这些？主要是因为this，来看看this干的好事。

```
box.onclick = function(){
    function fn(){
        alert(this);
    }
    fn();
};
```

我们原本以为这里面的this指向的是box,然而却是Window。一般我们这样解决：

```
box.onclick = function(){
    var _this = this;
    function fn(){
        alert(_this);
    }
    fn();
};
```

将this保存下来。

还有一些情况，有时我们想让伪数组也能够调用数组的一些方法，这时call、apply、bind就派上用场了。

我们先来解决第一个问题修复this指向。

```
box.onclick = function(){
    function fn(){
        alert(this);
    }
    fn();
};
```

改成如下：

```
box.onclick = function(){
    function fn(){
        console.log(this);
    }
    fn.call(this);
};
```

很神奇吧，call的作用就是改变this的指向的，第一个传的是一个对象，就是你要借用的那个对象。

```
fn.call(this);
```

这里的意思是让this去调用fn这个函数，这里的this是box，这个没有意见吧？如果这个你不清楚，很可能你是javscript的新朋友。box调用fn，这句话非常重要，我们知道this它始终指向一个对象，刚好box就是一个对象。那么fn里面的this就是box。

```
box.onclick = function(){
    function fn(){
        console.log(this);
    }
    fn.call(this);
};
```

这句话在某些情况下是可以简写的，比如：

```
box.onclick = function(){
    var fn = function(){
        console.log(this); //box
    }.call(this);
};
```

或者这样：

```
box.onclick = function(){
    (function(){
        console.log(this);
    }.call(this)); //box
};
```

又或者这样：

```
var objName = {name:'JS2016'};
var obj = {
    name:'0 _ 0',
    sayHello:function(){
        console.log(this.name);
    }.bind(objName)
};
obj.sayHello();//JS2016
```

call和apply、bind但是用来改变this的指向的，但也有一些小小的差别。下面我们来看看它们的差别在哪。

```
function fn(a,b,c,d){
    console.log(a,b,c,d);
}

//call
fn.call(null,1,2,3);

//apply
fn.apply(null,[1,2,3]);

//bind
var f = fn.bind(null,1,2,3);
f(4);
```

结果如下：

```
1 2 3 undefined
1 2 3 undefined
1 2 3 4
```

前面说过第一个参数传的是一个你要借用的对象，但这么我们不需要，所有就传了一个null，当然你也可以传其他的，反正在这里没有用到，除了第一个参数后面的参数将作为实际参数传入到函数中。

call就是挨个传值，apply传一个数组，bind也是挨个传值，但和call和apply还有多少不同，使用call和apply会直接执行这个函数，而bind并不会而是将绑定好的this重新返回一个新函数，什么时候调用由你自己决定。

```
var objName = {name:'JS2016'};
var obj = {
    name:'0 _ 0',
```

```
        sayHello:function(){
            console.log(this.name);
        }.bind(objName)
    };
    obj.sayHello();//JS2016
```

这里也就是为什么我要用bind的原因，如果用call的话就会报错了。自己想想这个sayHello在obj都已经执行完了，就根本没有sayHello这个函数了。

这几个方法使用的好的话可以帮你解决不少问题比如：

正常情况下Math.max只能这样用

```
    Math.max(10,6)
```

但如果你想传一个数组的话你可以用apply

```
    var arr = [1,2,30,4,5];
    console.log(Math.max.apply(null,arr));
```

又或者你想让伪数组调用数组的方法

```
    function fn(){
        [].push.call(arguments,3);
        console.log(arguments); //[1, 2, 3]
    }
    fn(1,2);
```

再者：

```
    var arr = ['aaabc'];
    console.log(''.indexOf.call(arr,'b')); //3
```

牛逼不，简直偷梁换柱，当然还有很多可以利用的，自己尽情花辉去吧。

简单说一下这种偷梁换柱的原理吧，实际上浏览器内部根本就不在乎你是谁，它只关心你传给我的是不是我能够运行的，如下：

正常情况

```
    var str = 'aaabc';
    console.log(str.indexOf('b'));
```

而这种情况其实做的事情和上面一模一样，看我来拆解。

```
var arr = ['aaabc'];
''.indexOf.call(arr);
```

这句话就是说让arr调用字符串的indexOf方法，前面说过了浏览器内部不在乎你是谁，所以谁都可以来调用，但不是100%成功，具体看如下。

```
''.indexOf.call(arr,'b')
```

这里的arr就是['aaabc']，内部很可能拆成了'aaabc'，因此就成了下面的这段代码。

```
'aaabc'.indexOf('b');
```

这就是它们的秘密。

这里得说一下bind在某些浏览器下不兼容。我们来模拟一个玩玩。

```
Function.prototype.$bind = function(obj){
        //保存当前this
    var _this = this;
        //截取除了第一个以外的所有实际参数
    var a = [].slice.call(arguments,1);
        //返回一个新函数
    return function(){
        //让当前那个调用的函数的this指向obj，并且把实参传给它，这里用了concat是因为，我们可能在绑定以后还传递参数，所以才把他们合并起来。如f(4)这个是在绑定以后传的参数，a这个argument是绑定时的。
        _this.apply(obj,a.concat([].slice.call(arguments)));
    };
};
function fn(a,b,c,d){
    console.log(a,b,c,d);
}
var f = fn.$bind(null,1,2,3);
f(4);
```

这个方法和实际上的bind还是差别很大的，如

```
var arr = ['JSS'];
var index = ''.indexOf.$bind(arr,'S');
console.log(index())
----------
function fff(){
    [].push.$bind(arguments,1);
    console.log(arguments);
}
fff();
```

这些都没法使用，因为技术有限就没办法带大家封装一个完美的bind了，如果有需要网上搜索一下吧。

结束了啊。

分类: javascript

标签: jscall、bind、apply

追梦子
关注 - 0
粉丝 - 217
+加关注

« 上一篇：设计模式之原型，学习笔记
» 下一篇：Javascript函数中的高级运用

posted @ 2016-08-19 12:00 追梦子 阅读(750) 评论(0) 编辑 收藏