前面两篇文章介绍了JavaScript执行上下文中两个重要属性：VO/AO和scope chain。本文就来看看执行上下文中的this。

首先看看下面两个对this的概括：

- **this**是执行上下文（**Execution Context**）的一个重要属性，是一个与执行上下文相关的特殊对象。因此，它可以叫作上下文对象（也就是用来指明执行上下文是在哪个上下文中被触发的对象）。
- **this**不是变量对象（**Variable Object**）的一个属性，所以跟变量不同，this从不会参与到标识符解析过程。也就是说，在代码中当访问this的时候，它的值是直接从执行上下文中获取的，并不需要任何作用域链查找。this的值只在进入上下文的时候进行一次确定。

关于this最困惑的应该是，同一个函数，当在不同的上下文进行调用的时候，this的值就可能会不同。也就是说，this的值是通过函数调用表达式（也就是函数被调用的方式）的caller所提供的。

下面就看看在不同场景中，this的值。

全局上下文

在全局上下文（Global Context）中，this总是global object，在浏览器中就是window对象。

```
console.log(this === window);

this.name = "Will";
this.age = 28;
this.getInfo = function(){
    console.log(this.name + " is " + this.age + " years old");
};
window.getInfo();
// true
// Will is 28 years old
```

函数上下文

在一个函数中，this的情况就比较多了，this的值直接受函数调用方式的影响。

Invoke function as Function

当通过正常的方式调用一个函数的时候，this的值就会被设置为global object（浏览器中的window对象）。

但是，当使用"strict mode"执行下面代码的时候，this就会被设置为"undefined"。

```
function gFunc(){
    return this;
}

console.log(gFunc());
console.log(this === window.gFunc());
// window
```

```
// true
```

Invoke function as Method

当函数作为一个对象方法来执行的时候，this的值就是该方法所属的对象。

在下面的例子中，创建了一个obj对象，并设置name属性的值为"obj"。所以但调用该对象的func方法的时候，方法中的this就表示obj这个对象。

```
var obj = {
    name: "obj",
    func: function () {
        console.log(this + ":" + this.name);
    }
};

obj.func();
// [object Object]:obj
```

为了验证"方法中的this代表方法所属的对象"这句话，再看下面一个例子。

在对象obj中，创建了一个内嵌对象nestedObj，当调用内嵌对象的方法的时候，方法中的this就代表nestedObj。

```
var obj = {
    name: "obj",
    nestedObj: {
        name:"nestedObj",
        func: function () {
            console.log(this + ":" + this.name);
        }
    }
}

obj.nestedObj.func();
// [object Object]:nestedObj
```

对于上面例子中的方法，通常称为绑定方法，也就是说这些方法都是个特定的对象关联的。

但是，当我们进行下面操作的时候，temp将是一个全局作用里面的函数，并没有绑定到obj对象上。所以，temp中的this表示的是window对象。

```
var name = "Will";
var obj = {
    name: "obj",
    func: function () {
        console.log(this + ":" + this.name);
    }
};

temp = obj.func;
temp();
//  [object Window]:Will
```

Invoke function as Constructor

在JavaScript中，函数可以作为构造器来直接创建对象，在这种情况下，this就代表了新建的对象。

```
function Staff(name, age){
    this.name = name;
    this.age = age;
    this.getInfo = function(){
        console.log(this.name + " is " + this.age + " years old");
    };
}

var will = new Staff("Will", 28);
will.getInfo();
// Will is 28 years old
```

Invoke context-less function

对于有些没有上下文的函数，也就是说这些函数没有绑定到特定的对象上，那么这些上下文无关的函数将会被默认的绑定到global object上。

在这个例子中，函数f和匿名函数表达式在被调用的过程中并没有被关联到任何对象，所以他们的this都代表global object。

```
var context = "global";

var obj = {
    context: "object",
    method: function () {
        console.log(this + ":" +this.context);

        function f() {
            var context = "function";
            console.log(this + ":" +this.context);
        };
        f();

        (function(){
            var context = "function";
            console.log(this + ":" +this.context);
        })();
    }
};

obj.method();
// [object Object]:object
// [object Window]:global
// [object Window]:global
```

call/apply/bind改变this

this本身是不可变的，但是 JavaScript中提供了call/apply/bind三个函数来在函数调用时设置this的值。

这三个函数的原型如下：

- fun.apply(obj1 [, argsArray])
  - Sets obj1 as the value of this inside fun() and calls fun() passing elements of argsArray as its arguments.
- fun.call(obj1 [, arg1 [, arg2 [,arg3 [, ...]]]])
  - Sets obj1 as the value of this inside fun() and calls fun() passing arg1, arg2, arg3, ... as its arguments.
- fun.bind(obj1 [, arg1 [, arg2 [,arg3 [, ...]]]])
  - Returns the reference to the function fun with this inside fun() bound to obj1 and parameters of fun bound to the parameters specified arg1, arg2, arg3, ...

下面看一个简单的例子：

```
function add(numA, numB){
    console.log( this.original + numA + numB);
}

add(1, 2);

var obj = {original: 10};
add.apply(obj, [1, 2]);
add.call(obj, 1, 2);

var f1 = add.bind(obj);
f1(2, 3);

var f2 = add.bind(obj, 2);
f2(3);
// NaN
// 13
// 13
// 15
// 15
```

当直接调用add函数的时候，this将代表window，当执行"this.original"的时候，由于window对象并没有"original"属性，所以会得到"undefined"。

通过call/apply/bind，达到的效果就是把add函数绑定到了obj对象上，当调用add的时候，this就代表了obj这个对象。

DOM event handler

当一个函数被当作event handler的时候，this会被设置为触发事件的页面元素（element）。

```
var body = document.getElementsByTagName("body")[0];
body.addEventListener("click", function(){
    console.log(this);
});
// <body>…</body>
```

In-line event handler

当代码通过in-line handler执行的时候，this同样指向拥有该handler的页面元素。

看下面的代码：

```
document.write('<button onclick="console.log(this)">Show this</button>');
// <button onclick="console.log(this)">Show this</button>
document.write('<button onclick="(function(){console.log(this);})()">Show this</button>');
// window
```

在第一行代码中，正如上面in-line handler所描述的，this将指向"button"这个element。但是，对于第二行代码中的匿名函数，是一个上下文无关（context-less）的函数，所以this会被默认的设置为window。

前面我们已经介绍过了bind函数，所以，通过下面的修改就能改变上面例子中第二行代码的行为：

```
document.write('<button onclick="((function(){console.log(this);}).bind(this))()">Show
this</button>');
// <button onclick="((function(){console.log(this);}).bind(this))()">Show this</button>
```

保存this

在JavaScript代码中，同一个上下文中可能会出现多个this，为了使用外层的this，就需要对this进行暂存了。

看下面的例子，根据前面的介绍，在body元素的click handler中，this肯定是指向body这个元素，所以为了使用"greeting"这个方法，就是要对指向bar对象的this进行暂存，这里用了一个self变量。

有了self，我们就可以在click handler中使用bar对象的"greeting"方法了。

当阅读一些JavaScript库代码的时候，如果遇到类似self，me，that的时候，他们可能就是对this的暂存。

```
var bar = {
    name: "bar",
    body: document.getElementsByTagName("body")[0],

    greeting: function(){
        console.log("Hi there, I'm " + this + ":" + this.name);
    },

    anotherMethod: function () {
        var self = this;
        this.body.addEventListener("click", function(){
            self.greeting();
        });
    }
};

bar.anotherMethod();
// Hi there, I'm [object Object]:bar
```

同样，对于上面的例子，也可以使用bind来设置this达到相同的效果。

```
var bar = {
    name: "bar",
    body: document.getElementsByTagName("body")[0],

    greeting: function(){
        console.log("Hi there, I'm " + this + ":" + this.name);
    },

    anotherMethod: function () {
        this.body.addEventListener("click", (function(){
            this.greeting();
        }).bind(this));
    }
};

bar.anotherMethod();
// Hi there, I'm [object Object]:bar
```

总结

本文介绍了执行上下文中的this属性，this的值直接影响着代码的运行结果。

在函数调用中，this是由激活上下文代码的调用者（caller）来提供的，即调用函数的父上下文(parent context)，也就是说this取决于调用函数的方式，指向调用时所在函数所绑定的对象。

通过上面的介绍，相信对this有了一定的认识。