JavaScript进阶

---JS中的this

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- JS this简介及特点
- JS this四种应用场景
- JS this缺陷及解决方法

JavaScript语言中的this和其他语言中区别

- Java等面向对象语言中的this

在 Java 等面向对象的语言中，this 关键字的含义是明确且具体的，即指代当前对象，一般在编译期就已经确定下来，称为编译期绑定

```
public class Point{
    private int x = 0;
    private int y = 0;

    public Point(x,y){
        this.x = x;
        this.y = y;
    }
}
```

- JavaScript语言中的this

在 JavaScript 中，this 是动态绑定，或称为运行期绑定的。由于其运行期绑定的特性，JavaScript 中的 this 含义要丰富得多，它可以是全局对象、当前对象或者任意对象，这完全取决于函数的调用方式

- this不进行作用域传递（函数嵌套时的this缺陷）

内容提纲

- JS this简介及特点
- JS this四种应用场景
- JS this缺陷和解决方法

JavaScript语言中的this主要有以下4种应用场景

- 一般函数中的this
- 对象方法中的this
- 构造函数中的this
- 间接调用中的this

注意：不管哪种场景下调用，this指向的都是调用此函数的主体

一、一般函数中的this（非严格松散模式下）

- 一般函数中的this（非严格松散模式下）指代全局对象

```
> function thisTest(){
      console.log(this === window);
  }
  thisTest();
```

```
true
```

- 可以通过this在函数内添加、删除、修改全局对象属性

```
> var a=10,b="Hi";
  function thisTest(){
      this.a=20;
      delete this.b;
      this.c = "新添加的全局变量";
  }
  thisTest();
  console.log(a,c);
```

```
20 "新添加的全局变量"
```


参见实例demo01 一般函数中的this（非严格模式下）

一、一般函数中的this（严格模式）

- 一般函数中的this（严格模式）为undefined

```
> function thisTest(){
      'use strict'
      console.log(this);
  }
  thisTest();
undefined
```

- 可以用此特性来判断当前是否为严格模式

```
> //'use strict'
  function isStrictMode(){
      return this===undefined?true:false;
  }
  isStrictMode();
< false
```


参见实例demo02 一般函数中的this（严格模式下）

二、对象方法中的this（无函数嵌套的情况下）

- 函数作为对象的一个属性时，称之为对象的方法
- 对象方法中的this指代调用此方法的对象（无嵌套的情况下）

```
> var point = {
  x : 0,
  y : 0,
  moveTo : function(x, y) {
      this.x = x;
      this.y = y;
      }
  };
  point.moveTo(1, 1)//this 绑定到当前对象，即 point 对象
  console.log(point);
```

```
▶ Object {x: 1, y: 1}
```


参见实例demo03 对象方法中的this

三、构造函数中的this

- 构造函数中的this指代通过new新创建的对象
  - JS（ES5） 并没有类（class）的概念，而是使用基于原型（prototype）的继承方式
  - JS中的构造函数充当了类的角色，如果不使用 new 调用，则和普通函数一样。
  - 如果作为构造函数正确调用时，构造函数中的this 绑定到新创建的对象上

```
> function Point(x, y){
    this.x = x;
    this.y = y;
  }
  var p = new Point(2,3);
  p;
<· ▶ Point {x: 2, y: 3}
```

思考：若直接调用Point函数，会是怎样一种情况，直接调用的话this指的是谁?

参见实例demo04 构造函数中的this

四、间接调用中的this（call、apply）

- 通过call/applay间接调用的函数中的this（指代第一个参数）
  - JS中函数也是对象（函数对象），也有属性和方法（length、call、apply等）
  - JS中函数可以通过call和apply进行间接调用，动态的指定由谁来调用此函数

```
objA = {name:"AA",x:1};
objB = {name:"BB",x:5};
function test(){
    console.log(this.name,this.x);
}
objA.fun = test;
objA.fun(); //AA 1
objA.fun.call(objB); //BB 5
```

思考：call和apply方法是定义在哪里的?

参见实例demo05 间接调用中的this

内容提纲

- JS this简介及特点
- JS this四种应用场景
- JS this缺陷和解决方法

对象方法中的this（有函数嵌套的情况下）

- this不进行作用域传递、函数嵌套中的this存在缺陷

```
var point = {
x : 0,
y : 0,
moveTo : function(x, y) {
    // 内部函数
    function moveToX(x) {
        this.x = x;//this 绑定到了哪里?
    };
    // 内部函数
    function moveToY(y) {
        this.y = y;//this 绑定到了哪里?
    };
    moveToX(x);
    moveToY(y);
    }
};
```

```
point.moveTo(2,2);
console.log(point);
console.log(window.x,window.y);
```

```
▶ Object {x: 0, y: 0, moveTo: function}
2 2
```

moveTo方法中嵌套的moveToX函数作为一般函数看待，此函数中的this此时指向的是全局对象window而不是point

参见实例demo06前半部分 函数嵌套时this缺陷

如何解决对象方法中嵌套函数的this指向问题

- 方法一：使用变量（that或self）软绑定，使this指向正确

```
var point = {
x : 0,
y : 0,
moveTo : function(x, y) {
    var that = this;
    // 内部函数
    function moveToX(x) {
        that.x = x;//此时that指向point
    };
    // 内部函数
    function moveToY(y) {
        that.y = y;//此时that指向point
    };
    moveToX(x);
    moveToY(y);
    }
};
```

```
point.moveTo(2,2);
console.log(point);
console.log(x,y);//报ReferenceError
```

```
▶ Object {x: 2, y: 2}
```

moveTo方法中嵌套的函数中的that此时指向的是point对象。

这是软绑定形式，除此之外，还可以通过函数对象的bind方法进行硬绑定

参见实例demo06后半部分 this缺陷解决办法

如何解决对象方法中嵌套函数的this指向问题

- 方法二：使用call/apply间接调用，使this指向正确

```
var point = {
    x:0,y:0,
    moveTo:function (x,y) {
        function moveToX() {
            //this绑定到了哪里？
            this.x = x;
        }
        function moveToY() {
            //this绑定到了哪里？
            this.y = y;
        }
        moveToX.call(this);
        moveToY();
    }
};
```

```
point.moveTo(2,2);
console.log(point);//2,0
console.log(y);//打印x会报错
```

```
▶ {x: 2, y: 0, moveTo: f}
```

```
2
```


参见实例demo06后半部分 this缺陷解决办法

如何解决对象方法中嵌套函数的this指向问题

- 方法三：使用Function.prototype.bind，使this指向正确

```
var point = {
    x:0,y:0,
    moveTo:function (x,y) {
        function moveToX() {
            //this绑定到了哪里？
            this.x = x;
        }
        function moveToY() {
            //this绑定到了哪里？
            this.y = y;
        }
        moveToX.bind(point)();
        moveToY.bind(point)();
    }
};
```

```
point.moveTo(2,2);
console.log(point);
```

```
▶ {x: 2, y: 2, moveTo: f}
```


参见实例demo06后半部分 this缺陷解决办法

构造函数中的this（有函数嵌套的情况下）

- 构造函数中的this同样存在函数嵌套缺陷，解决办法同上

```
function Point(x, y){
    this.x = x;
    this.y = y;
    this.moveXY = function(x,y){
        function moveX(x){
            this.x+=x;
        }
        function moveY(y){
            this.y+=y;
        }
        moveX(x);
        moveY(y);
    };
}
var p = new Point(2,3);
p.moveXY(1,1);
console.log(p);//输出为 Piont{x:2,y:3}
```

```
function Point(x, y){
    this.x = x;
    this.y = y;
    this.moveXY = function(x,y){
        var that = this;//此处的this指new出来的新对象
        function moveX(x){
            that.x+=x;
        }
        function moveY(y){
            that.y+=y;
        }
        moveX(x);
        moveY(y);
    };
}
var p = new Point(2,3);
p.moveXY(1,1);
console.log(p);//输出为 Piont{x:3,y:4}
```


参见实例demo07 构造函数中的函数嵌套中的this

总结

- JS this简介及特点
- JS this四种应用场景
- JS this缺陷和解决方法

补充：JS this本质概述

- 函数对象的内部方法 [[call]]
  - 函数对象有一个叫[[Call]]内部方法，函数的执行其实是通过[[Call]]方法来执行的
  - [[Call]]方法接收两个参数thisArg和argumentList
  - thisArg和this的指向有直接关系，argumentList为函数的实参列表
- thisArg与4种情况的对应关系
  - 普通方法调用thisArg为undefined。
  - 通过call或apply调用，thisArg既为第一个参数。
  - 通过对象调用，thisArg指向该对象。
  - 在构造方法中，thisArg为新构造的对象
- 总原则：this指的是调用函数的那个主体

Thank You!