回忆之前学过的事件函数中的this

function fn(){

console.log(this); // ?

}

btn.onclick = fn;

btn.onclick = function (){

fn();

}

引导：我们已经学习过这种方式调用函数，哪么我们再来看看函数其它的调用方式以及它们的this指向。

**再来看看两个面试中常见的案例：**

var numbers = {

numberA: 5,

numberB: 10,

sum: function() {

console.log(this === numbers); // ?

function calculate() {

console.log(this === numbers); // ?

return this.numberA + this.numberB;

}

return calculate();

}

};

console.log(numbers.sum()); // ?

var numbers = {

numberA: 5,

numberB: 10,

sum: function() {

console.log(this === numbers); // ?

function calculate() {

console.log(this === numbers); // ?

return this.numberA + this.numberB;

}

return calculate.call(this);

}

};

console.log(numbers.sum()); // ?

**学习目标**

1. js中的this指向
2. 函数调用的四种方式
3. 独立分析出各种函数调用中this的指向。

**什么是this？**

1、在JS中，this是当前执行函数的上下文。

2、JS中有4种不同的函数调用方式，每种方法都定义了自己的上下文。

3、函数中的this是在调用的时候确定的。（而不是在函数定义的时候）

函数调用、方法调用、构造函数调用、隐式调用

注意：理解this的关键点就是要对函数调用以及它如何影响上下文有个清晰的理解。

我们下面分别讲解这四种，每种分成（什么是？它里面的this指的是谁？）

**一些术语：**

1、函数调用 指 执行构成一个函数的代码。

如：

function fn(){

alert(this);

}

fn();

2、函数调用的上下文指this在函数体中的值

3、函数的作用域指的是在函数体内可以使用的变量、对象以及函数的集合。

**一、函数调用**

定义：当一个函数名后面跟着圆括号，这就是函数调用。看看下面的函数调用：

如1：

function hello(name) {

return 'Hello ' + name + '!';

}

var message = hello('World');

console.log(message); // ?

如2：自执行函数调用

var message = (function(name) {

return 'Hello ' + name + '!';

})('World');

console.log(message); // ?

函数调用中的this

this 在函数调用中是一个全局对象（全局对象是由执行的环境决定的。在浏览器里它是window对象）。如1：

function sum(a, b) {

console.log(this === window); // ?

this.myNumber = 20;

return a + b;

}

console.log(sum(15, 16)); // ?

console.log(window.myNumber); // ?

当this在所有函数作用域以外(最上层的作用域：全局执行的上下文)调用时，它也指向全局对象：如2：

console.log(this === window); // ?

this.myString = 'Hello World!';

console.log(window.myString); // ?

**二、方法调用**

方法：一个方法是作为一个对象的属性存储的函数

如：

var myObject = {

helloFunction: function() {

return 'Hello World!';

}

};

var message = myObject.helloFunction();

说明：helloFunction是myObject的一个方法。为了使用这个方法, 使用属性访问：myObject.helloFunction

方法调用：

当一个表达式以属性访问的形式执行时，执行的是方法调用，如下myObject.helloFunction()

[1, 2].join(',')

/\s/.test('beautiful world')

函数调用和方法调用的区别：

他们最主要的区别在于方法调用要求函数以属性访问的形式调用。

案例：区分下面6个调用方式，是方法调用还是函数调用？

['Hello', 'World'].join(', ');

({ ten: function() { return 10; } }).ten();

var obj = {};

obj.myFunction = function() {

return new Date().toString();

};

obj.myFunction();

var otherFunction = obj.myFunction;

otherFunction();

parseFloat('16.60');

isNaN(0);

方法调用中的this

在方法调用中，this是指拥有这个方法的对象。

如：下面会打印四次，分别打印什么？

var calc = {

num: 0,

increment: function() {

console.log(this === calc); // ?

this.num += 1;

return this.num;

}

};

console.log(calc.increment()); // ?

console.log(calc.increment()); // ?

例2：下面会打印两次，分别打印什么？

var calc = {

num: 0,

increment: function() {

console.log(this === calc); // ?

this.num += 1;

return this.num;

}

};

var test = calc.increment;

console.log(test()); // ?

结论：一个对象中的方法可以赋值给另一个变量。当这个变量调用方法时，它会变成函数调用，this变成全局对象。

**三、构造函数调用**

**四、隐式调用**

当函数被.call()或者.apply()调用时，执行的是隐式调用。

格式：函数.call(this指向, 'value1', 'value2')

格式：函数.apply(this指向, ['value1', 'value2'])

定义：它们是函数下面的方法，作用是调用函数，并改变函数中的this指向。

两者的主要区别是：

.call()接受一组参数，例如myFunction.call(thisValue, 'value1', 'value2')

.apply()接受的一组参数必须是一个类似数组的对象，例如myFunction.apply(thisValue, ['value1', 'value2'])

如：

function increment(number) {

return ++number;

}

console.log(increment.call(undefined, 10)); // ?

console.log(increment.apply(undefined, [10])); // ?

隐式调用中的this

在隐式调用.call()或.apply()中，this是第一个参数，如果第一个参数为undefined或者null，则this指向window

如：三次分别打印什么

var json = {

names : "彭作洪"

}

function sum(str){

console.log(this.names + str); // ?

}

sum.call(json, '你好');

sum.apply(json, ['欢迎你']);

sum('下次再来');

哪么我们回头再来看看最开始的面试题

小结：

必须是在函数调用时，才能确定this。

事件调用，函数调用，方法调用，构造函数调用，隐式调用

最后，我们再来看看几个练习题

var a = 8,

o = {

a: 10,

b: {

a: 12,

fn: function() {

console.log(this.a);

}

}

}

o.b.fn();

var a = 20;

var json = {

a: 10,

b: function() {

alert(this.a);

}

};

function aaa(fn) {

fn();

}

aaa(json.b);

json.b();

var name = 'jay';

var person = {

name: 'kang',

pro: {

name: 'zs',

getName: function() {

return this.name;

}

}

}

console.log(person.pro.getName());

var pepole = person.pro.getName;

console.log(pepole())

function Foo() {

var i = 0;

return function() {

console.log(i++);

}

}

var f1 = Foo(),

f2 = Foo();

f1();

f1();

f2();

var name = "design";

var $ = {

name: "张三",

say: function() {

alert(name);

}

}

$.say();

var foo = {

bar: function() {

return this.baz;

},

baz: 1

};

var a = (function() {

return typeof arguments[0]();

})(foo.bar);

console.log(a);

console.log(sub());

var x = 10;

function sub() {

x--;

return x;

}

var a = 20,

b = function(c) {

s = a + c;

};

b(30);

console.log(s)

var name = "The Window";

var object = {

　　　name : "My Object",

　　　getNameFunc : function(){

　　　 return function(){

　　　　　　　return this.name;

　　　　　};

　　　}

};

alert(object.getNameFunc()());

function say() {

var num = 666;

var sayAlert = function() {

alert(num);

}

num++;

return sayAlert;

}

var sayAlert = say();

sayAlert()

sayAlert()

function box(obj) {

var obj;

obj.name = 'Lee';

obj = new Object();

obj.name = 'kkk';

}

var obj = new Object();

obj.name = 'test';

box(obj);

console.log(obj.name);

function a(xx) {

this.x = xx;

return this;

}

var x = a(5);

console.log(x);

var y = a(6);

console.log(x.x);

console.log(y.x);