this

this 的指向

- 默认绑定　比较常见的有独立函数的调用。

```
function foo(){
    console.log(this.a);//this指向window对象
}
var a = 2;
foo();//2
```

如何判断应用使用了默认绑定呢，`foo`是直接使用不带任何修饰的函数进行调用的。注意如果使用了严格模式，`this`会绑定到`undefined`。

```
'use strict';
function foo(){
    console.log(this);//undefined
    console.log(this.a);
}
var a = 2;
foo(); // TypeError: Cannot read property 'a' of undefined
```

在`ES5`中，`this`是指当前函数中正在执行的上下文环境，`this`**永远指向最后调用它的那个对象，若是没有调用它则指向window**。匿名函数，定时器中的函数，由于没有默认的宿主对象，所以默认`this`指向`window`。 看以下例子。

```
var name = "windowsName";
function a(){
    var name = "Martin";
    console.log(this.name);//windowsName
    console.log("inner this->"+this);//inner this->[object Window]
}
a();
console.log("outer this->"+this);//outer this->[object Window]
```

- 因为`this`**永远指向最后调用它的那个对象**，我们看到最后调用`a`的地方是`a()`;前面没有调用的对象那么就是全局对象`window`，这就相当于是`window.a()`;所以`this`指向全局对象`window`。因此打印出来的`this.name`是`windowsName`。

```
var name = "windowsName";
function a(){
    var name = "Martin";
    fn:function(){
        console.log(this.name);//Martin
    }
}
a.fn();
```

- `this` **永远指向最后调用它的那个对象**，函数 `fn` 是对象 `a` 调用的，`this` 指向对象 `a` (在 `javascript` 中，函数既是函数也是对象，在这里 `a` 既是函数也是对象)，所以打印的值就是 `a` 中的 `name` 的值。

```
var name = "windowsName";
function a(){
    var name = "Martin";
    fn:function(){
        console.log(this.name);//Martin
    }
}
window.a.fn();
```

- `this` **永远指向最后调用它的那个对象**，最后调用 `fn` 函数的对象依然是 `a`。

```
var name = "windowsName";
var a = {
    //name:"Martin",
    fn:function () {
        console.log(this.name);  // undefined
    }
}
window.a.fn();
```

- 这里调用 `fn` 的是 `a` 对象，也就是说 `fn` 的内部的 `this` 是对象 `a`,而对象 `a` 中并没有对 `name` 进行定义，所以打印出来的值是 `undefined`。
- 这个例子还是说明了：`this` **永远指向最后调用它的那个对象**，因为最后调用 `fn` 的对象是 `a`，所以就算 `a` 中没有 `name` 这个属性，也不会继续向上一个对象寻找 `this.name`，而是直接输出 `undefined`。

```
var name = "windowsName";
var a = {
    name: "Martin",
    fn:function () {
        console.log(this.name);// windowsName
    }
}
var f = a.fn;
f();
```

- 这里为什么不是`Martin`?这是因为虽然将`a`对象的`fn`方法赋值给变量`f`了，但是没有调用。`this`**永远指向最后调用它的那个对象**，由于刚刚的`f`并没有被调用。所以下一句代码调用`f()`的时候相当于`window.f()`，也相当于`window.fn()`，所以`fn()`最后仍然是被`window`调用的。所以`this`指向的也就是`window`。
- 从以上几个例子看出`this`**永远指向最后调用它的那个对象。**

```
var name = "windowsName";
function fn() {
    var name = 'Martin';
    innerFunction();
    function innerFunction() {
        console.log(this.name);// windowsName
    }
}
fn();
```

- `this`**永远指向最后调用它的那个对象，若是没有调用它则指向window。**这里`fn.innerFunction()`这样的调用是没有用的，两个方法之间没有对象级的关系。所以这里的`this`还是指向`window`。看下面函数的调用。

怎样改变this的指向

- 使用ES6的箭头函数
- 在函数内部使用`var that = this`
- 使用`call`，`apply`，`bind`
- `new`实例化一个对象

```
var name = "windowsName";
var a = {
    name:"Martin",
    func1:function(){
        console.log(this.name)
    },
    func2:function(){
        setTimeout(function(){
            this.func1()
        },100);
    }
};
a.func2()// this.func1 is not a function
```

在不使用箭头函数的情况下，是会报错的，因为最后调用`setTimeout()`方法时，`setTimeout()`是`window`对象的方法，`window`对象调用了`setTimeout()`方法，根据`this`**永远指向最后调用它的那个对象**的原理，`this`指向`window`对象，但是在`window`对象中并没有`func1()`函数，所以报错（匿名函数，定时器中的函数，由于没有默认的宿主对象，所以默认`this`指向`window`）。

使用箭头函数绑定

根据MDN - Arrow_functions：

> 箭头函数拥有词法作用域的this值（即不会新产生自己作用域下的this, arguments, super 和 new.target 等对象）

箭头函数本身不具有`this`，它会直接绑定到它的词法作用域内的`this`，也就是定义它时的作用域内的`this`值（`JavaScript`中的作用域就是词法作用域，整个代码结构中只有函数可以限定作用域）。所以箭头函数的`this`始终指向函数定义生效时所在的对象，而非执行时，箭头函数默认使用父级作用域的`this`。

```
var obj = {
    a:function(){
        console.log(this);//ES5中this永远指向最后调用它的那个对象，这里打印出来的是obj
    }
}
obj.a();
```

```
var obj = {
    //注意这里并没有this
    a:()=>{
        console.log(this);//ES6箭头函数，打印出来的是window对象
    }
}
obj.a();
```

这里会比较神奇的是为何第二个例子打印出来的是`window`对象，因为`a()`方法的`this`在箭头函数中，所以指向父作用域中，父作用域就是最外层作用域（全局作用域），父作用域中的`this`指向`window`，所以输出`window`。还有人会疑问`a()`方法中的父作用域不是对象`obj = {}`吗？因为这里是个对象，对象中有属性，方法，但并没有`this`。

```
var name = "windowsName";
var a = {
    name:"Martin",
    func1:function(){
        console.log(this.name)
    },
    func2:function(){
        console.log(this);//通过a.func2()调用的话，这里的this指向对象a
        setTimeout(()=> {
            this.func1();
        },100);
    }
};
a.func2();// Martin
```

箭头函数中`this`，首先到它的父作用域找，如果父作用域还是箭头函数，那么接着向上找，直到找到我们要的`this`指向。 这里例子`setTimeout()`方法使用了箭头函数，其父作用域为`func2`方法体内，所以箭头函数内的`this`和你在`func2`方法体内其他使用的`this`的指向是一样的。当通过`a.func2()`，`func2`作用域中的`this`指向对象`a`，所以 箭头函数中所使用的`this`也指向对象`a`。（只有函数才能制造作用域结构，至少有一个作用域，即全局作用域 ）。

在函数内部使用 `var that = this`

```
var name = "windowsName";
var a = {
    name:"Martin",
    func1: function () {
        console.log(this.name)
    },
    func2:function () {
        var that = this;//这里通过对象a调用func2方法的话，a.func2(),则this的指向为对象a
        setTimeout( function() {
            that.func1()
        },100);
    }
};
a.func2();// Martin
```

- 在`func2`中，首先设置`var that = this`，这里的`this`是指调用`func2`的对象`a`，为了防止在`func2`中的`setTimeout`被`window`调用而导致的在`setTimeout`中的`this`为`window`。我们将`this`(这里的this指向对象 a) 赋值给一个变量`that`，这样，在`func2`中`setTimeout`里面我们使用`that`也就是指向对象`a`了。

使用 call，apply，bind

- `call`、`apply`、`bind`三者都是用来改变函数的`this`对象的指向的；
- `call`、`apply`、`bind`三者第一个参数都是`this`要指向的对象，也就是想指定的上下文；
- `call`、`apply`、`bind`三者都可以利用后续参数传参；

- `bind` 是返回对应函数，便于稍后调用；`apply` 、`call` 则是立即调用 。

```
var a = {
    name:"Martin",
    func1:function(){
        console.log(this.name)
    },
    func2:function(){
        setTimeout(function(){
            this.func1()
        }.call(a),100);
    }
};
a.func2();//Martin
```

```
var a = {
    name:"Martin",
    func1:function(){
        console.log(this.name)
    },
    func2:function(){
        setTimeout(function(){
            this.func1()
        }.apply(a),100);
    }
};
a.func2();//Martin
```

```
var a = {
    name:"Martin",
    func1:function(){
        console.log(this.name)
    },
    func2:function(){
        setTimeout(function(){
            this.func1()
        }.bind(a)(),100);
    }
};
a.func2();//Martin
```

关于call，apply，bind的使用在其他章节讲。

JavaScript中关于函数的调用

函数调用的方法一共有 4 种

- 函数调用-作为一个函数调用 `alert('Hello World!')`
- 方法调用-函数作为方法调用 `console.log('Hello World!')`
- 构造函数调用-使用构造函数调用函数 `new RegExp('\\d')`

- 间接调用-作为函数方法调用函数（call、apply）`alert.call(undefined, 'Hello World')`

函数调用

```
var name = "windowsName";
function a() {
    var name = "Martin";
    console.log(this.name);// windowsName
    console.log("inner-this:" + this);// inner-this:Window
}
a();
console.log("outer-this:" + this);// outer-this:Window
```

这样一个最简单的函数，不属于任何一个对象，就是一个函数，这样的情况在 `JavaScript` 的在浏览器中的非严格模式下，这个 `a` 方法默认是属于全局对象 `window` 的，在严格模式下，则是 `undefined`。 但这是一个全局的函数，很容易产生命名冲突，所以不建议这样使用。

方法调用

```
var name = "windowsName";
var a = {
    name: "Martin",
    fn : function () {
        console.log(this.name); // Martin
    }
}
a.fn();
```

这里更多的情况是将函数作为对象的方法使用 。

构造函数调用

```
function myFunction(arg1, arg2) {// 构造函数:
    this.firstName = arg1;
    this.lastName  = arg2;
}
var a = new myFunction("xie","tingfeng");//创建新对象a
a.lastName; //tingfeng
```

函数调用前使用了 `new` 关键字, 则是调用了构造函数。关于 `new` 的过程如下。

```
var a = new myFunction("xie","tingfeng");
new myFunction{
    var obj = {};
    obj.__proto__ = myFunction.prototype;
    var result = myFunction.call(obj,"xie","tingfeng");
    return typeof result === 'obj'? result : obj;
}
```

- 创建一个空对象 obj
- 将新创建的空对象的隐式原型指向其构造函数的显示原型
- 使用 call 改变 this 的指向（在 new 的过程中，我们是使用 call 改变了 this 的指向）
- 如果无返回值或者返回一个非对象值，则将 obj 返回作为新对象；如果返回值是一个新对象的话那么直接直接返回该对象。

间接调用

在`JavaScript`中, 函数是对象。`JavaScript`函数有它的属性和方法。`call()`和`apply()`是预定义的函数方法。 两个方法可用于调用函数，两个方法的第一个参数必须是对象本身。在`JavaScript`严格模式(`strict mode`)下, 在调用函数时第一个参数会成为`this`的值， 即使该参数不是一个对象。在`JavaScript`非严格模式(`non-strict mode`)下, 如果第一个参数的值是`null`或`undefined`, 它将使用全局对象替代。

```
var name = "windowsName";
function fn() {
    var name = 'Martin';
    innerFunction();
    function innerFunction() {
        console.log(this.name);// windowsName
    }
}
fn();
```

这里的`innerFunction()`的调用就是作为一个函数调用的（第一种方式），没有挂载在任何对象上，所以对于没有挂载在任何对象上的函数，在非严格模式下`this`就是指向`window`的。

```
var name = "windowsName";
var a = {
    name:"Martin",
    func1:function(){
        console.log(this.name)
    },
    func2:function(){
        setTimeout(function(){
            this.func1();
        },100 );
    }
};
a.func2();// this.func1 is not a function
```

这个例子也可以理解为**匿名函数的`this`永远指向`window`对象。匿名函数是无法其他被对象调用的**，所以匿名函数的`this`永远指向`window`对象。一般匿名函数的用法是让其自执行或者被其他函数调用，自执行的话就在匿名函数加`()`，被其他函数调用的话则如以上例子`setTimeout`。