你的this又指到哪里去啦?

前言

在 JavaScript 中，关键字“this”通常用来引用当前执行上下文中的对象。它的值取决于函数被调用的方式以及所在的上下文。

为什么要有this?

为了让**对象中的函数**有能力**访问对象自己的属性**

示例一

```
let obj = {
    myname: 'lt',
    age: 18,
    bar: function() { // 在对象内部方法中使用对象内部的属性
        console.log(myname);
    }
}
obj.bar()
```

打印结果会**报错**

示例二

```
let obj = {
    myname: 'lt',
    age: 18,
```

```
    bar: function() { // 在对象内部方法中使用对象内部的属性
        console.log(this.myname);
    }
}
obj.bar()


```

打印结果为

this可以在些场景使用
==========

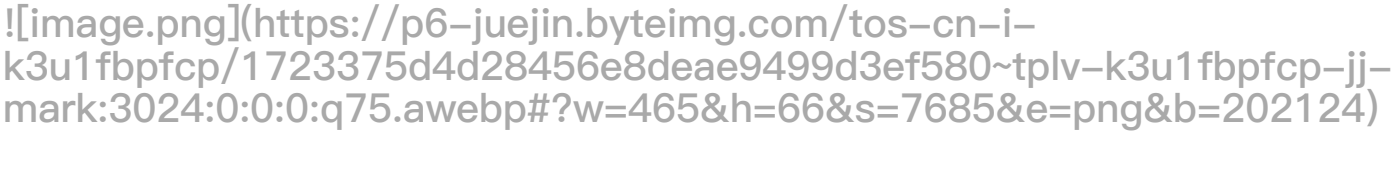
能形成作用域的地方就能使用this(**块级作用域内不可使用**)，即this可以在**全局、函数体(箭头函数不承认this)** 内使用。**特例：eval()内也能使用this。**

this在全局

```
console.log(this);


```

放到浏览器的运行结果为：

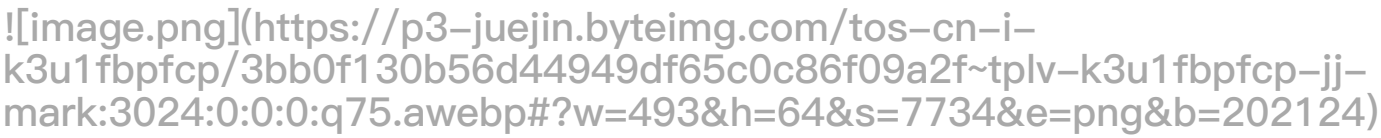


window是全局对象，所以this写在全局，this代指的就是window。

this在函数体内

```
function foo() {
    console.log(this);
}
foo()
```

放到浏览器的运行结果为：



会发现this的指向还是window,所以我们就要了解this的绑定规则才能更好地判断this的指向。

this 的绑定规则
==========

默认绑定规则：当一个函数独立调用，不带任何修饰符时。

* 函数在哪个词法作用域下生效，函数中的this就指向哪里。

```
function foo() {
    console.log(this);
}
foo()
```

可见foo函数是被独立调用并且它的词法作用域是全局即被声明在了全局，所以this就指向全局，即window。

```
function foo() {
    console.log(this);
}
function bar() {
    foo()
}
bar()
```

打印结果也是window,因为foo函数是被独立调用并且还是声明在了全局，所以this就指向全局，即window。

其实只要是默认绑定规则 this 就指向window

```
function foo() {
    console.log(this.a);
}

var a = 1

foo()
```

浏览器输出结果是：1。

因为**在全局，通过 var 声明的变量相当于在window对象上加入了一个属性。**

隐式绑定规则：当函数的引用有上下文对象时。(即函数被某个对象所拥有)

* 函数的 this 指向引用它的对象

```
var obj = {
    a: 1,
```

```
    foo: foo // 没有(), 是引用foo函数
}

function foo() {
    console.log(this.a);
}

obj.foo()
```

浏览器输出结果是：1。因为foo函数被obj对象引用，并且foo函数不是独立调用，所以this指向obj对象。

隐式丢失：当一个函数被多个对象链式调用时，函数的this指向就近的那个对象。(就近原则)

```
var obj = {
    a: 1,
    foo: foo // 没有(), 是引用foo函数
}

var obj2 = {
    a: 2,
    obj: obj
}
function foo() {
    console.log(this.a);
}

obj2.obj.foo()
```

这里输出结果还是：1。this指向obj对象，就近原则。

显示绑定：通过这些方法 call apply bind 把函数的this指向到一个指定的对象上去

* 这三种方法都能接收参数，apply是以数组的方式接收参数；bind会返回一个函数体，接收参数方式可任意(见代码)。

```
var obj = {
    a: 1,
}

function foo() {
    console.log(this.a, x + y);
}

foo.call(obj, 2, 3)
foo.apply(obj, [2, 3]) // 接收参数的方式是通过数组来接受
var bar = foo.bind(obj, 2, 3, 4) // 4是多余的，打印结果还是5
bar()
var bar = foo.bind(obj)
bar(2, 3) // 这样传参数结果也是5
var bar = foo.bind(obj, 2) // 可以靠bind接收参数也可以靠返回的函数体
bar接收，打印结果还是5
bar(3)
var bar = foo.bind(obj, 3, 4) // 就近原则，打印结果是7
bar(2)

```

new 绑定：this 指向创建的实例对象

```
function Person() {
// new 的步骤
//  var obj = { // 创建一个obj对象
//      name = 'ltt'
//  }
 // Person.call(obj) //让this指向obj对象
 // Object.__proto__ = Person.prototype // 实例对象继承构造函数上的方法
    this.name = 'ltt'
 // return obj // 返回到实例对象p p2
}

let p = new Person() // new在构造函数内部创建出的对象返回给p实例对象
let p2 = new Person()

```

this指向new在构造函数内部创建出的obj对象，p和p2是两个独立的对象

箭头函数不承认this
==========

箭头函数没有this这个机制，写在箭头函数中的this是它外层非箭头函数的this。

箭头函数书写规则

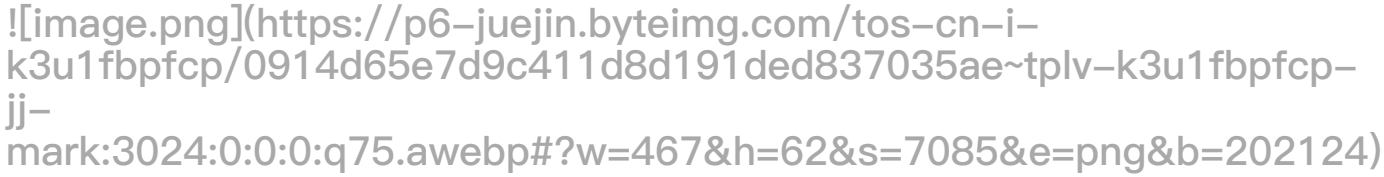
```
var bar = () => {

}
```

示例

```
function foo() {
    var bar = () => {
        console.log(this);
    }
    bar()
}
foo()
```

输出结果



this指向window,说明this是属于foo函数的。

k3u1fbpfcp/8ceac6e0ca8946478caf6f79cbf3a0c3~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=474&h=202&s=20769&e=jpg&b=f5f1f0)

原文链接: https://juejin.cn/post/7366826090836164647