在箭头函数中，this与封闭词法上下文的this保持一致。在全局代码中，它将被设置为全局对象

var globalObject = this;

var foo = (() => this);

console.log(foo() === globalObject); // true

*注意：如果将thisArg传递给call、bind、或者apply，它将被忽略。不过你仍然可以为调用添加参数，不过第一个参数（thisArg）应该设置为null。*

// 接着上面的代码

// 作为对象的一个方法调用

var obj = {foo: foo};

console.log(obj.foo() === globalObject); // true

// 尝试使用call来设定this

console.log(foo.call(obj) === globalObject); // true

// 尝试使用bind来设定this

foo = foo.bind(obj);

console.log(foo() === globalObject); // true

无论如何，foo的this被设置为它被创建时的上下文（在上面的例子中，就是全局对象）。这同样适用于在其他函数中创建的箭头函数：这些箭头函数的this被设置为外层执行上下文。

// 创建一个含有bar方法的obj对象，bar返回一个函数，这个函数返回它自己的this，

// 这个返回的函数是以箭头函数创建的，所以它的this被永久绑定到了它外层函数的this。

// bar的值可以在调用中设置，它反过来又设置返回函数的值。

var obj = {bar: function() {

var x = (() => this);

return x;

}

};

// 作为obj对象的一个方法来调用bar，把它的this绑定到obj。

// x所指向的匿名函数赋值给fn。

var fn = obj.bar();

// 直接调用fn而不设置this，通常(即不使用箭头函数的情况)默认为全局对象，若在严格模式则为undefined

console.log(fn() === obj); // true

// 但是注意，如果你只是引用obj的方法，而没有调用它(this是在函数调用过程中设置的)

var fn2 = obj.bar;

// 那么调用箭头函数后，this指向window，因为它从 bar 继承了this。

console.log(fn2()() == window); // true

在上面的例子中，一个赋值给了 obj.bar的函数（称为匿名函数 A），返回了另一个箭头函数（称为匿名函数 B）。因此，函数B的this被永久设置为obj.bar（函数A）被调用时的this。当返回的函数（函数B）被调用时，它this始终是最初设置的。在上面的代码示例中，函数B的this被设置为函数A的this，即obj，所以它仍然设置为obj，即使以通常将this设置为undefined或全局对象（或者如前面示例中全局执行上下文中的任何其他方法）进行调用。