**this指向哪里？**

一般而言，在Javascript中，this指向函数**执行时**的当前对象。

**没有明确的当前对象时**

当没有明确的执行时的当前对象时，this指向全局对象window。

箭头函数：

箭头函数相当于匿名函数，并且简化了函数定义。箭头函数有两种格式，一种像上面的，只包含一个表达式，连{ ... }和return都省略掉了。还有一种可以包含多条语句，这时候就不能省略{ ... }和return：

x => {

**if** (x > 0) {

**return** x \* x;

}

**else** {

**return** - x \* x;

}

}

如果参数不是一个，就需要用括号()括起来：

*// 两个参数:*

(x, y) => x \* x + y \* y

*// 无参数:*

() => 3.14

*// 可变参数:*

(x, y, ...rest) => { **var** i, sum = x + y;

**for** (i=0; i<rest.length; i++) {

sum += rest[i];

}

**return** sum;

}

如果要返回一个对象，就要注意，如果是单表达式，这么写的话会报错：

*// SyntaxError:*

x => { foo: x }

因为和函数体的{ ... }有语法冲突，所以要改为：

*// ok:*

x => ({ foo: x })

箭头函数看上去是匿名函数的一种简写，但实际上，箭头函数和匿名函数有个明显的区别：箭头函数内部的this是词法作用域，由上下文确定。

回顾前面的例子，由于JavaScript函数对this绑定的错误处理，下面的例子无法得到预期结果：

**var** obj = {

birth: 1990,

getAge: **function** () {

**var** b = **this**.birth; *// 1990*

**var** fn = **function** () {

**return** **new** Date().getFullYear() - **this**.birth; *// this指向window或undefined*

};

**return** fn();

}

};

现在，箭头函数完全修复了this的指向，this总是指向词法作用域，也就是外层调用者obj：

**var** obj = {

birth: 1990,

getAge: **function** () {

**var** b = **this**.birth; *// 1990*

**var** fn = () => **new** Date().getFullYear() - **this**.birth; *// this指向obj对象*

**return** fn();

}

};

obj.getAge(); *// 25*

函数在执行过程中，如果没有遇到return语句（函数末尾如果没有return，就是隐含的return undefined;），控制权无法交回被调用的代码。

generator跟函数很像，定义如下：

**function**\* foo(x) {

**yield** x + 1;

**yield** x + 2;

**return** x + 3;

}

generator和函数不同的是，generator由function\*定义（注意多出的\*号），并且，除了return语句，还可以用yield返回多次。

直接调用一个generator和调用函数不一样，fib(5)仅仅是创建了一个generator对象，还没有去执行它。

调用generator对象有两个方法，一是不断地调用generator对象的next()方法：

var f = fib(5);

f.**next**(); // {value: 0, done: **false**}

f.**next**(); // {value: 1, done: **false**}

f.**next**(); // {value: 1, done: **false**}

f.**next**(); // {value: 2, done: **false**}

f.**next**(); // {value: 3, done: **false**}

f.**next**(); // {value: undefined, done: **true**}

next()方法会执行generator的代码，然后，每次遇到yield x;就返回一个对象{value: x, done: true/false}，然后“暂停”。返回的value就是yield的返回值，done表示这个generator是否已经执行结束了。如果done为true，则value就是return的返回值。

当执行到done为true时，这个generator对象就已经全部执行完毕，不要再继续调用next()了。

第二个方法是直接用for ... of循环迭代generator对象，这种方式不需要我们自己判断done：

窗体顶端

'use strict'

function\* fib(max) {

var

t,

a = 0,

b = 1,

n = 0;

while (n < max) {

yield a;

[a, b] = [b, a + b];

n ++;

}

return;

}

窗体底端