**输出结果:**

let name='cn'

function fn(){

this.name = 'hh'

alert(this.name); }

fn();

弹出结果是'hh';分析:

全局作用域或者普通函数中 this 指向全局对象 window.

一个函数中有this，但是函数没有被上一级的对象所调用，那么这个this指向的就是window.

**输出结果:**

var user = "哈哈"

var obj = {

user:"啦啦",

fn:function(){

console.log(this.user);

}

}

obj.fn();

打印结果是'啦啦';分析:

如果一个函数中有this，这个函数又被上一级的对象所调用，那么this指向的就是上一级的对象。

谁调用函数,函数的this就指向谁.

**输出结果:**

var o = {

a:10,

b:{

fn:function(){

console.log(this.a);

}

}

}

o.b.fn();

打印结果是'undefined';分析:

尽管对象b中没有属性a，这个this指向的也是对象b，因为this只会指向它的上一级对象，不管这个对象中有没有this要的东西。不管这个链式调用有多长.

**输出结果:**

var o = {

a:10,

b:{

a:12,

fn:function(){

console.log(this.a);

}

}

}

o.b.fn();

var bar = o.b.fn;

bar();

两次打印结果分别是'12'和'undefined';分析:

o.b.fn();打印12;this指向对象b;

var bar = o.b.fn;

bar(); 打印undefined

this永远指向的是最后调用它的对象，也就是看它执行的时候是谁调用的。虽然函数fn是被对象b所引用，但是在将fn赋值给变量bar的时候并没有执行，所以最终指向的是window。

**输出结果:**

function Test(){

this.user = "foo";

}

var a = new Test ();

console.log(a.user);

打印结果为"foo";new关键字就是创建一个对象实例;隐式的改变this指向; 这个this指向实例化对象a;

**输出结果:**

let obj1={

a:222

};

let obj2={

a:111,

fn:function(){

console.log(this.a);

}

}

obj2.fn.call(obj1);

打印结果为"222"; call 或者 apply主要用途就是

改变 this 的指向（把 this 从 obj2 指向到 obj1 ）.

**输出结果:**

let obj={

a:222,

fn:function(){

setTimeout(function(){console.log(this.a)})

}

};

obj.fn();

打印结果为"undefined"; setTimeout中函数内的this是指向了window对象，这是由于setTimeout()调用的代码运行在与所在函数完全分离的执行环境上。这会导致这些代码中包含的 this 关键字会指向 window.

**输出结果:**

let obj={

a:222,

fn:function(){

setTimeout(()=>{console.log(this.a)})

}

};

obj.fn();

打印结果为"222";箭头函数没有自己的this;看其外是否有函数;如果有,外层函数的this就是箭头函数的this;如果没有,this就是window.