this练习题

2019/05/01 21:13

```
var f = (
    function f(){
        return 1
    },
    function g(){
        return 2
    }
)();
typeof f;
```

此题注意f是一个逗号表达式，说明直接执行的是function g（）结果f=2类型是number

```
var x = 1;
if (function (){}){
    x += typeof f;
}
console.log(x);
```

lundefined

function是true之后变成undefined

我认为很难得题

```
var name = 222;
var a = {
    name : "111",
    say : function (){
        console.log(this.name);
    }
}
var fun = a.say;  //相当将a里的function拿出来
fun();  //在外面执行，于是this指向的是window结果是222
a.say()  //结果是111，在a中调用say方法执行

var b = {
    name : "333",
    say : function (fun) {
        //this --> b
        fun();
        //function (){
        //console.log(this.name);
    //}
    }
}
b.say(a.say);  //b.say()this变为b
//a.say没有执行所以this不是a
```

```
//里面的且也没写this.fun()执行所以也不是b
//所以还是走预编译环节this是window
b.say = a.say;
b.say();//333
var b = {
    name : "333"
    say : function (fun) {
        test();
    }
}
b.say(a.say);  //111  test执行没调用别的所以还是全局
function test(){
    console.log(this);
}
```

arguments.callee

指向函数自身引用

在那个函数里面就指代那个函数

返回的是函数名可以用在立即执行函数递归上

```
var num(function (n){
    if (n==1){
        return 1;
    }
    return n *arguments.callee(n-1);
}(100))
```

caller

函数自身的属性

函数被调用时所在的环境、

```
var foo = 123;
function print() {
    this.foo = 234;
    console.log(foo); //234
    //因为本来没有foo所以要打印GO里面的foo原先是123
    // this是window改变了foo的值
}
print();


var foo = 123;
function print() {
    //var this = Object.create(print.ptototype)
    this.foo = 234;
    console.log(foo);
    }
```

```
    new print()//123 new之后就相当于this对象中添加了foo属性
    //没有改变window的foo


var bar = {a : "022"};
function print() {
    bar.a = 'a';
    Object.prototype.b = 'b';   //所有的对象的最终原型
    return function inner() {
        console.log(bar.a);//a
        console.log(bar.b);//b
    }
}
print()(); //第一个括号返回函数，第二个括号函数执行
```