《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

第08讲：为什么说函数是 JavaScript 的一等公民？

导入

拉勾教育
互联网人实战大学

数据类型与函数是很多高级语言中最重要的两个概念

前者用来存储数据，后者用来存储代码

JavaScript 中的函数相对于数据类型而言更加复杂

它可以有属性，也可以被赋值给一个变量，还可以作为参数被传递……

正是这些强大特性让它成了 JavaScript 的“一等公民”

this 关键字

拉勾教育
互联网人实战大学

什么是 this ？

this 是 JavaScript 的一个关键字，一般指向**调用它的对象**

- 首先 this 指向的应该是一个对象
  更具体地说是函数执行的“上下文对象”
- 其次这个对象指向的是“调用它”的对象
  如果调用它的不是对象或对象不存在
  则会指向全局对象（严格模式下为 undefined）

this 关键字

```
// 代码 1
var o = {
 fn() {
  console.log(this)
 }
}
o.fn() // o
// 代码 2
class A {
 fn() {
  console.log(this)
 }
}
var a = new A()
a.fn()// a
// 代码 3
function fn() {
 console.log(this)
}
fun() // 浏览器：Window；Node.js：global
```

this 关键字


```
function fn() {console.log(this)}
function fn2() {fn()}
fn2() // ?
```

this 关键字


```
function fn() {console.log(this)}
function fn2() {fn()}
var obj = {fn2}
obj.fn2() // ?
```

this 关键字


```
var dx = {
  arr: [1]
}
dx.arr.forEach(function() {console.log(this)}) // ?
```

需要传入 this 指向的函数还有：

every()、find()、findIndex()、map()、some()

this 关键字


```
class B {
 fn() {
  console.log(this)
 }
}
var b = new B()
var fun = b.fn
fun() // ?
```

this 关键字


```
class B {
  'use strict';
  fn() {
    console.log(this)
  }
}
```

this 关键字


```
var arrow = {fn: ()
=> {
  console.log(this)
}}
arrow.fn() // ?
```

this 关键字


```
var arrow = {
  fn() {
    const a = () => console.log(this)
    a()
  }
}
arrow.fn()  // arrow
```

this 关键字


```
[0].forEach(function() {console.log(this)}, 0) // ?
```

this 关键字


```
function getName() {console.log(this.name)}
var b = getName.bind({name: 'bind'})
b()
getName.call({name: 'call'})
getName.apply({name: 'apply'})
```

补充 1：箭头函数

拉勾教育
互联网人实战大学

箭头函数和普通函数的区别

- 不绑定 arguments 对象，也就是说在箭头函数内访问 arguments 对象会报错
- 不能用作构造器，也就是说不能通过关键字 new 来创建实例
- 默认不会创建 prototype 原型属性
- 不能用作 Generator() 函数，不能使用 yeild 关键字

函数的转换


```
add(1) // 1
add(1)(2) // 3
add(1, 2)(3, 4, 5)(6) // 21
```

函数的转换

隐式转换函数 toString() 和 valueOf()

toString() 函数会在打印函数的时候调用，比如 console.log

valueOf 会在获取函数原始值时调用，比如加法操作

函数的转换


```
function add(...args) {
  let arr = args
  function fn(...newArgs) {
    arr = [...args, ...newArgs]
    return fn;
  }
  fn.toString = fn.valueOf = function() {
    return arr.reduce((acc, cur) => acc + parseInt(cur))
  }
  return fn
}
```

原型

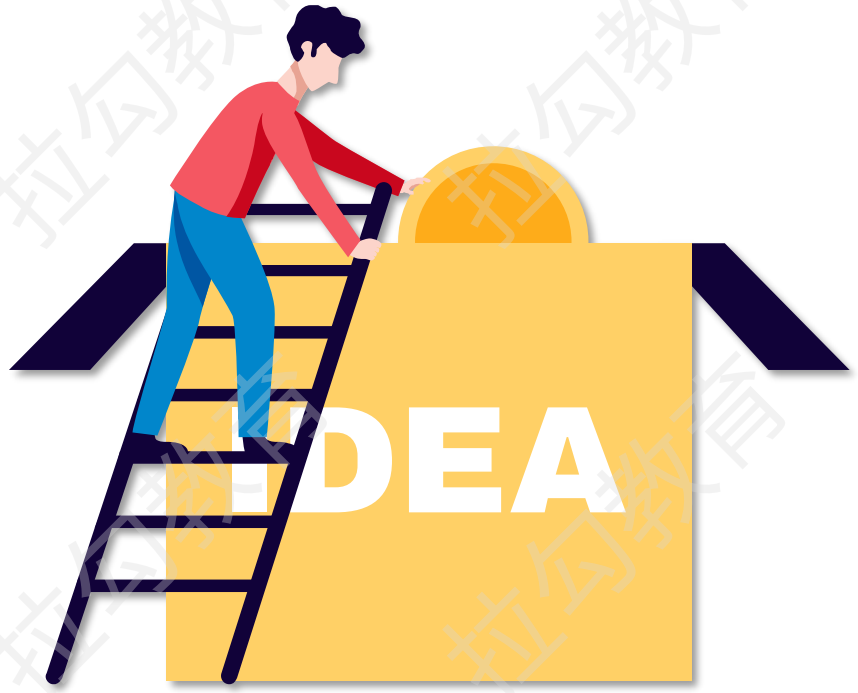
原型是 JavaScript 的重要特性之一

可以让对象从其他对象继承功能特性

所以 JavaScript 也被称为 **“基于原型的语言”**

严格地说原型应该是对象的特性

但函数其实也是一种特殊的对象



原型

拉勾教育
互联网人实战大学

```
function fn(){}
fn instanceof Object // true
```

什么是原型和原型链?

拉勾教育
互联网人实战大学

原型就是**对象的属性**

包括被称为隐式原型的 **__proto__ 属性**和被称为显式原型的 **prototype 属性**

什么是原型和原型链?


```
var a ={}
a.__proto__ === Object.prototype // true
var b= new Object()
b.__proto__ === a.__proto__ // true
```

什么是原型和原型链？


```
function fn() {}
fn.prototype.constructor === fn // true
```

什么是原型和原型链?


```
var parent = {code:'p',name:'parent'}
var child = {__proto__: parent, name: 'child'}
console.log(parent.prototype) // undefined
console.log(child.name) // "child"
console.log(child.code) // "p"
child.hasOwnProperty('name') // true
child.hasOwnProperty('code') // false
```

new 操作符实现了什么?


```
function F(init) {}
var f = new F(args)
```

new 操作符实现了什么?


```
function F(init) {}
var f = new F(args)
```

- 创建一个临时的空对象
  为了表述方便，我们命名为 fn
  让对象 fn 的隐式原型指向函数 F 的显式原型
- 执行函数 F()
  将 this 指向对象 fn，并传入参数 args
  得到执行结果 result
- 判断上一步的执行结果 result
  如果 result 为非空对象，则返回 result
  否则返回 fn

new 操作符实现了什么?


```
var fn = Object.create(F.prototype)
var obj = F.apply(fn, args)
var f = obj && typeof obj === 'object' ? obj : fn;
```

怎么通过原型链实现多层继承？


```
function A() {
}
A.prototype.a = function() {
  return 'a';
}
function B() {
}
B.prototype = new A()
B.prototype.b = function() {
  return 'b';
}
var c = new B();
c.b() // 'b'
c.a() // 'a'
```

补充 2：typeof 和 instanceof

类型	结果
Undefined	"undefined"
Boolean	"boolean"
Number	"number"
BigInt	"bigint"
String	"string"
Symbol	"symbol"
函数对象	"function"
其他对象及 null	"object"

补充 2：typeof 和 instanceof


```
left.__proto__.__proto__... =?= right.prototype
```

作用域

作用域是指赋值、取值操作的执行范围

通过作用域机制可以有效地防止变量、函数的重复定义

以及控制它们的可访问性

作用域

虽然在浏览器端和 Node.js 端作用域的处理有所不同

比如对于全局作用域，浏览器会自动将未主动声明的变量提升到全局作用域

而 Node.js 则需要显式的挂载到 global 对象上

比如在 ES6 之前，浏览器不提供模块级别的作用域

而 Node.js 的 CommonJS 模块机制就提供了模块级别的作用域

但在类型上可以分为**全局作用域（window/global）**、**块级作用域（let、const、try/catch）**、**模块作用域（ES6 Module、CommonJS）**及本课时重点讨论的**函数作用域**

命名提升

对于使用 var 关键字声明的变量以及创建命名函数的时候

JavaScript 在解释执行的时候都会将其声明内容提升到作用域顶部

这种机制称为 “**命名提升**”

命名提升


```
console.log(a) // undefined
var a = 1
console.log(b) // 报错
let b = 2
```

命名提升


```
fn() // 2
function fn() {
  return 2
}
```

命名提升


```
// 方式 1
var f = function() {...}
// 方式 2
function f() {...}
```

闭包

在函数内部访问外部函数作用域时就会产生闭包

闭包很有用，因为它允许将函数与其所操作的某些数据（环境）**关联**起来

这种关联不只是跨作用域引用，也可以实现数据与函数的隔离

闭包


```
var SingleStudent = (function () {
  function Student() {}
  var _student;
  return function () {
    if (_student) return _student;
    _student = new Student()
    return _student;
  }
}())
var s = new SingleStudent()
var s2 = new SingleStudent()
s === s2 // true
```

补充 3：经典笔试题


```
for( var i = 0; i < 5; i++ ) {
setTimeout(() => {
console.log( i );
}, 1000 * i)
}
```

补充 3：经典笔试题


```
var i;
for(i = 0; i < 5; i++) {
setTimeout(() => {
console.log(i);
}, 1000 * i)
}
```

补充 3：经典笔试题


```
for(let i = 0; i < 5; i++ ) {
setTimeout(() => {
console.log(i);
}, 1000 * i)
}
/**
等价于
```

补充 3：经典笔试题


```
for(var i = 0; i < 5; i++ ) {
  let _i = i + 1
setTimeout(() => {
console.log(_i)
}, 1000 * i)
}
*/
```

总结

介绍了函数相关的重要内容

包括 this **关键字的指向**、**原型与原型链的使用**

函数的隐式转换、**函数和作用域的关系**

总结

修改函数的 this 指向，到底有多少种方式呢？

Next：第09讲《为什么代码没有按照编写顺序执行？》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U