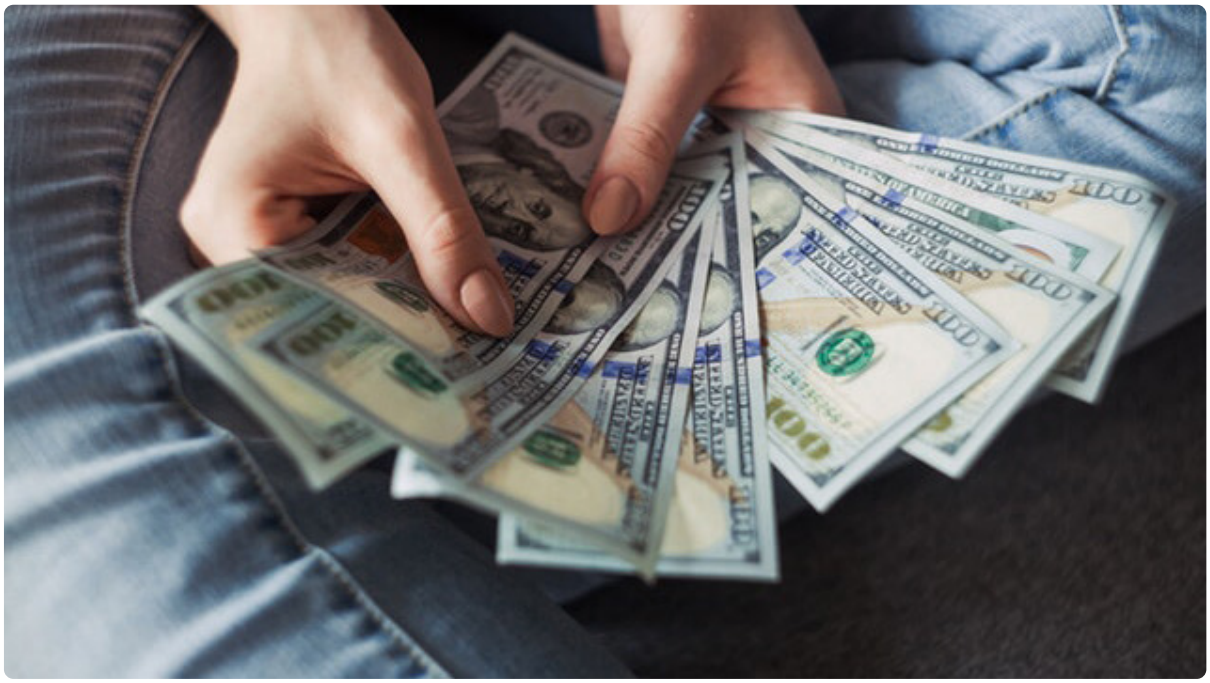
22 迭代器模式：银行的点钞机

更新时间：2019-08-22 11:14:34



> 一个人追求的目标越高，他的才力就发展得越快，对社会就越有益。
>
> ——高尔基

迭代器模式（Iterator Pattern）用于顺序地访问聚合对象内部的元素，又无需知道对象内部结构。使用了迭代器之后，使用者不需要关心对象的内部构造，就可以按序访问其中的每个元素。

> **注意：** 本文可能用到一些 ES6 的语法 Symbol、Iterator 等，如果还没接触过可以点击链接稍加学习 ~

1. 什么是迭代器

银行里的点钞机就是一个迭代器，放入点钞机的钞票里有不同版次的人民币，每张钞票的冠字号也不一样，但当一沓钞票被放入点钞机中，使用者并不关心这些差别，只关心钞票的数量，以及是否有假币。

这里我们使用 JavaScript 的方式来点一下钞：

```
var bills = ['MCK013840031', 'MCK013840032', 'MCK013840033', 'MCK013840034', 'MCK013840035']

bills.forEach(function(bill) {
  console.log('当前钞票的冠字号为 ' + bill)
})
```

是不是很简单，这是因为 JavaScript 已经内置了迭代器的实现，在某些个很老的语言中，使用者可能会为了实现迭代器而烦恼，但是在 JavaScript 中则完全不用担心。

2. 迭代器的简单实现

前面的 forEach 方法是在 IE9 之后才原生提供的，那么在 IE9 之前的时代里，如何实现一个迭代器呢，我们可以使用 for 循环自己实现一个 forEach：

```
var forEach = function(arr, cb) {
  for (var i = 0; i < arr.length; i++) {
    cb.call(arr[i], arr[i], i, arr)
  }
}

forEach(['hello', 'world', '!'], function(currValue, idx, arr) {
  console.log('当前值 ' + currValue + '，索引为 ' + idx)
})

// 输出： 当前值 hello，索引为 0
// 输出： 当前值 world，索引为 1
// 输出： 当前值 ！ ，索引为 2
```

2.1 jQuery 源码中迭代器实现

jQuery 也提供了一个 $.each 的遍历方法：

```
// jquery 源码 /src/core.js#L246-L265
each: function (obj, callback) {
  var i = 0

  // obj 为数组时
  if (isArrayLike(obj)) {
    for (; i < obj.length; i++) {
      if (callback.call(obj[i], i, obj[i]) === false) {
        break
      }
    }
  }

  // obj 为对象时
  else {
    for (i in obj) {
      if (callback.call(obj[i], i, obj[i]) === false) {
        break
      }
    }
  }

  return obj
}

// 使用
$.each(['hello', 'world', '!'], function(idx, currValue){
  console.log('当前值 ' + currValue + '，索引为 ' + idx)
})
```

这里的源码分为两个部分，前一个部分是形参 obj 为数组情况下的处理，使用 for 循环，以数组下标依次使用 call/apply 传入回调中执行，第二部分是形参 obj 为对象情况下的处理，是使用 for-in 循环来获取对象上的属性。另外可以看到如果 callback.call 返回的结果是 false 的话，这个循环会被 break。

源码位于： jquery/src/core.js#L246-L265

由于处理对象时使用的是 for-in，所以原型上的变量也会被遍历出来：

```
var foo = { paramProto: '原型上的变量' }

var bar = Object.create(foo, {
    paramPrivate: {
        configurable: true,
        enumerable: true,
        value: '自有属性',
        writable: true
    }
})

$.each(bar, function(key, currValue) {
    console.log('当前值为 「' + currValue + '」，键为 ' + key)
})

// 输出： 当前值为 「自有属性」 ，键为 paramPrivate
// 输出： 当前值为 「原型上的属性」，键为 paramProto
```

因此可以使用 `hasOwnProperty` 来判断键是否是在原型链上还是对象的自有属性。

我们还可以利用如果 `callback.call` 返回的结果是 `false` 则 `break` 的特点，来进行一些操作：

```
$.each([1, 2, 3, 4, 5], function(idx, currValue) {
    if (currValue > 3)
      return false
    console.log('当前值为 ' + currValue)
})

// 输出： 当前值为 1
// 输出： 当前值为 2
// 输出： 当前值为 3
```

2.2 underscore 源码中的迭代器实现

underscore 作为兼容到 IE6 的古董级工具库，自然也是有迭代器的实现：

```
// underscore 源码
_.each = function(obj, iteratee) {
    var i, length

    // obj 为数组时
    if (isArrayLike(obj)) {
        for (i = 0, length = obj.length; i < length; i++) {
            iteratee(obj[i], i, obj)
        }
    }

    // obj 为对象时
    else {
        var keys = _.keys(obj)
        for (i = 0, length = keys.length; i < length; i++) {
            iteratee(obj[keys[i]], keys[i], obj)
        }
    }
    return obj
}

// 使用
_.each(['hello', 'world', '!'], function(currValue, idx, arr) {
    console.log('当前值 ' + currValue + '，索引为 ' + idx)
})
```

underscore 迭代器部分的实现跟 jQuery 的差不多，只是回调 `iteratee` 的执行是直接调用，而不是像 jQuery 是使用 `call`，也不像 jQuery 那样提供了迭代终止 break 的支持，所以总的来说还是 jQuery 的实现更优。

另外，这里 `iteratee` 变量的命名也可以看出来迭代器的含义。

源码位于： underscore.js#L181-L195

3. JavaScript 原生支持

随着 JavaScript 的 ECMAScript 标准每年的发展，给越来越多好用的 API 提供了支持，比如 Array 上的 `filter`、`forEach`、`reduce`、`flat` 等，还有 Map、Set、String 等数据结构，也提供了原生的迭代器支持，给我们的开发提供了很多便利，也让 underscore 这些工具库渐渐淡出历史舞台。

另外，JavaScript 中还有很多类数组结构，比如：

1. `arguments`：函数接受的所有参数构成的类数组对象；
2. `NodeList`：是 `querySelector` 接口族返回的数据结构；
3. `HTMLCollection`：是 `getElementsBy` 接口族返回的数据结构；

对于这些类数组结构，我们可以通过一些方式来转换成普通数组结构，以 `arguments` 为例：

```
// 方法一
var args = Array.prototype.slice.call(arguments)

// 方法二
var args = [].slice.call(arguments)

// 方法三 ES6提供
const args = Array.from(arguments)

// 方法四 ES6提供
const args = [...arguments];
```

转换成数组之后，就可以快乐使用 JavaScript 在 Array 上提供的各种方法了。

4. ES6 中的迭代器

ES6 规定，默认的迭代器部署在对应数据结构的 `Symbol.iterator` 属性上，如果一个数据结构具有 `Symbol.iterator` 属性，就被视为可遍历的，就可以用 `for...of` 循环遍历它的成员。也就是说，`for...of` 循环内部调用的是数据结构的 `Symbol.iterator` 方法。

`for-of` 循环可以使用的范围包括 Array、Set、Map 结构、上文提到的类数组结构、Generator 对象，以及字符串。

> **注意：** ES6 的 Iterator 相关内容与本节主题无关，所以不做更详细的介绍，如果读者希望更深入，推介先阅读阮一峰的 <Iterator 和 for...of 循环> 相关内容。

通过 `for-of` 可以使用 `Symbol.iterator` 这个属性提供的迭代器可以遍历对应数据结构，如果对没有提供 `Symbol.iterator` 的目标使用 `for-of` 则会抛错：

```
var foo = { a: 1 }

for (var key of foo) {
   console.log(key)
}

// 输出： Uncaught TypeError: foo is not iterable
```

我们可以给一个对象设置一个迭代器，让一个对象也可以使用 `for-of` 循环：

```
var bar = {
    a: 1,
    [Symbol.iterator]: function() {
        var valArr = [
            { value: 'hello', done: false },
            { value: 'world', done: false },
            { value: '!', done: false },
            { value: undefined, done: true }
        ]
        return {
            next: function() {
                return valArr.shift()
            }
        }
    }
}

for (var key of bar) {
    console.log(key)
}

// 输出： hello
// 输出： world
// 输出： !
```

可以看到 `for-of` 循环连 `bar` 对象自己的属性都不遍历了，遍历获取的值只和 `Symbol.iterator` 方法实现有关。

5. 迭代器模式总结

迭代器模式早已融入我们的日常开发中，在使用 `filter`、`reduce`、`map` 等方法的时候，不要忘记这些便捷的方法就是迭代器模式的应用。当我们使用迭代器方法处理一个对象时，我们可以关注与处理的逻辑，而不必关心对象的内部结构，侧面将对象内部结构和使用者之间解耦，也使得代码中的循环结构变得紧凑而优美。

}

21 模板方法模式：咖啡厅制作咖啡

23 命令模式：江湖通缉令