

# 7 迭代器模式

迭代器模式指提供一种方法顺序访问一个聚合对象的各个元素，而不需要暴露对象的内部表示。最终将迭代的过程从业务逻辑中分离出来，达到不关心对象内部构造也可按顺序访问内部元素的诉求

其实现在流行的大部分语言Java，JS，Ruby等都已经有了内置的迭代器实现

## jQuery中的迭代器

例如jQuery中的\$.each函数，如下：

```
1 /**
2  * jQuery中的迭代器
3  */
4 $.each([1,2,3], function(i,n){
5     console.log('当前下标'+i, '当前值为'+n)
6 })
```

## 自己实现一个迭代器

按照上面的用法我们可以借助js的for语句实现：

```
1 /**
2  * 实现自己的迭代器
3  */
4 var each = function(arr, callback){
5     for(var i = 0, l = arr.length; i < l; i++){
6         callback.call(arr[i], i, arr[i]); // 下标和元素传递过去
7     }
8 }
9
10 each([1,2,3], function(i,n){
11     console.log('当前下标'+i, '当前值为'+n)
12 })
```

## 内部迭代器和外部迭代

### 1. 内部迭代器

上面编写的each函数就属于此，函数内部定义好了迭代规则，外部使用简单，不需要关注内部实现。但是这也是缺点，例如如果你希望同时迭代2个数组就没办法了。就必须再新写一个doubleEach函数了。

在一些没有闭包的语言中，内部迭代器本身的实现也比较复杂，因为循环处理所需要的数据都要以参数的方式传递进去，灵活性比较弱

## 2. 外部迭代器

外部迭代器必须由调用者显式地请求迭代下一个元素。

会增加调用的复杂度，但是优点是增强了灵活性，可以手工控制迭代过程，适用面更广

```
1 // 一种外部迭代器的实现
2 var Iterator = function(obj){
3     var current = 0;
4
5     var next = function(){
6         current += 1;
7     }
8
9     var getCurItem = function(){
10         return obj[current];
11     }
12
13     var isDone = function(){
14         return current >= obj.length;
15     }
16
17     return {
18         next: next,
19         isDone: isDone,
20         getCurItem: getCurItem
21     }
22 }
```