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：

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

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：

```
/**
 * 实现自己的迭代器
 */
var each = function(arr, callback){
    for(var i = 0, l = arr.length; i < l; i++){
        callback.call(arr[i], i, arr[i]); // 下标和元素传递过去
    }
}

each([1,2,3], function(i,n){
    console.log('当前下标'+i, '当前值为'+n)
})
```

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

```
// 一种外部迭代器的实现
var Iterator = function(obj){
    var current = 0;

    var next = function(){
        current += 1;
    }

    var getCurItem = function(){
        return obj[current];
    }

    var isDone = function(){
        return current >= obj.length;
    }

    return {
        next: next,
        isDone:isDone,
        getCurItem: getCurITem
    }
}
```

倒序迭代器

由于GoF中对迭代器的定义非常松散，也没有规定我们以顺序，倒叙还是中序来迭代，我们完全可以实现一个倒序访问的迭代器：

```
/**
 * 倒序迭代器
 */
var reverseEach = function(arr, callback){
```

```
    for(let i = arr.length - 1;i >= 0;i--){
        callback(arr[i]);
    }
}
```

迭代器模式的应用举例

之前有一块根据不同的浏览器获取对应的上传组件对象的方法：

```
/**
 * 重构前的函数
 */

const getUploadObj = function(){
    try{
        // IE
        return new ActiveXObject("TXFTNActiveX.FTNUpload");
    } catch(e) {
        if(supportFlash()){
            var str = '<object type="application"></object>';
            return $(str).appendTo($('body'))
        } else {
            var str = '<input name="file"/>';
            return $(str).appendTo($('body'))
        }
    }
}
```

这段代码问题比较多：

- 充斥try-catch和if-else逻辑，违反开闭原则
- 难以阅读
- 新增获取H5上传等逻辑必须改原来的代码

策略拆分和迭代重构之后：

```
/**
 * 重构后的函数
 */
const getActivexUploadObj = function(){
    try{
        // IE
```

```
        return new ActiveXObject("TXFTNActiveX.FTNUpload");
    } catch(e) {
        return false
    }
}

const getFlashUploadObj = function(){
    if(supportFlash()){
        var str = '<object type="application"></object>';
        return $(str).appendTo($('body'))
    }
    return false;
}

const getFormUploadObj = function(){
    var str = '<input name="file"/>';
    return $(str).appendTo($('body'));
}

const iteratorUploadObj = function(){
    for(let i = 0; i < arguments.length; i++){
        const uploadObj = arguments[i]();
        if(uploadObj !== false){
            return uploadObj;
        }
    }
}

const uploadObj = iteratorUploadObj(getActivexUploadObj, getFlashUploadObj, getF
```

- 各种获取上传对象的方法互不干扰
- 几种逻辑解耦开来
- 增加了新的逻辑也只需要新增getxx方法，不需要动迭代逻辑