# 迭代器模式

## 一、基本概念 **1、定义：**

##### 迭代器模式（Iterator），提供了一种方法顺序访问一个聚合对象中各个元素，而又不暴露该对象内部的表示。

##### 优势是调用者可以不知道当前的对象的内部元素，则均可以访问，并且可以实现顺序倒叙各种方法。

##### 但是当前该模式有许多接口已经可以完成其目的 例如for range循环等，无需在自己进行实现。

### 2、组成环节：

##### 1、Aggregate(聚焦抽象类):聚焦抽象类，具体抽象类的父类，声明实现方法。

##### 2、ConcreteAggregate(聚焦类)：聚焦抽象类的子类，具体调用的类，可以生成一个对应的迭代类型。

##### 3、Iterator(迭代抽象类)：用于定义得到开始对象，得到下一个对象，判断访问是否结束，当前对象的内容的统一方法。

##### 4、ConcreteIterator(迭代类)：迭代抽象类的子类，继承以上四种方法，供调用者具体调用使用。

## 二、go语言中实现

##### 1、首先定义一个Iterator类型，并且定义其中的五种方法。

type Iterator interface {  
 Set(a []string)  
 First() string  
 Next() string  
 IsDone() bool  
 CurrentItem() string  
}

##### 2、再定义一个具体的迭代类，并且对这五种方法进行实现。

type ConcreteIterator struct {  
 count int  
 list []string  
}  
  
func (ci \*ConcreteIterator) Set(a []string) {  
 ci.list = a  
}  
  
func (ci \*ConcreteIterator) First() string {  
 ci.count = 0  
 return ci.list[0]  
}  
  
func (ci \*ConcreteIterator) Next() string {  
 ci.count++  
 if ci.IsDone() {  
 return ""  
 }  
 return ci.list[ci.count]  
}  
  
func (ci \*ConcreteIterator) IsDone() bool {  
 return ci.count >= len(ci.list)  
}  
  
func (ci \*ConcreteIterator) CurrentItem() string {  
 return ci.list[ci.count]  
}

##### 3、定义抽象的聚合类，并且声明其中的方法，返回一个迭代类。

type Aggregate interface {  
 CreateInterator() Iterator  
}

##### 实现具体的聚合类，实现其中的方法

type ConcreteAggregate struct{}  
  
func (ca ConcreteAggregate) CreateInterator() Iterator {  
 return &ConcreteIterator{}  
}

##### 5、开始进行main方法的调用。

func main() {  
 ca := ConcreteAggregate{}  
 ci := ca.CreateInterator()  
 ci.Set([]string{"1", "2", "3"})  
 ci.First()  
 for !ci.IsDone() {  
 fmt.Printf("%s is in iterator\n", ci.CurrentItem())  
 ci.Next()  
 }  
}

##### 5、结果的输出。

1 is in iterator

2 is in iterator

3 is in iterator

**（本方法采取的是顺序的遍历访问）**