迭代器模式(Iterator)

# 迭代器

## 迭代器的目的

迭代器的目的是**遍历所有的元素，并不会对元素进行排序**。

最基本的方法**：hasNext和next方法，这两个方法配合使用，调用next之前必须调用hasNext判断是否还有元素**。

## 迭代器的分类

内部迭代器和外部迭代器

# 迭代器模式：

## 定义：

**迭代器模式**：提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露其内部的表示。

## 优点：

**迭代器模式**让我们顺序访问一个聚集对象的每一个元素，而又不用知道其内部是如何表示的。

# 集合的概念：

通常所说的集合：指的是Java中的Collection及其实现类。

而广义的集合是指一群对象，其存储方式可以是各式各样的数据结构，例如：列表、数组、散列表。无论用什么方式存储，一律可以视为是集合。

# 当采用不同的集合存储数据时，怎样提供同一的接口遍历所有的数据呢？

通过实现各自集合的迭代器，迭代器又都实现了接口Iterator，这样就可以利用同一个接口进行遍历所有的数据了。

# 示例

对于**Iterator**、**Iterable**接口都可以自己创建，当然也可以直接使用JDK提供的java.lang.Iterable和java.util.Iterator接口。

## 下面的例子就是，给数组提供一个迭代器：

Java中没有提供给数组提供迭代器的方法，这里通过实现Iterator接口，创建一个针对数组的迭代器：

Menu类：

**public** **class** Menu **implements** Iterable<String> {

**int** position = 0;

String[] menus = **new** String[4];

// 默认菜单上有四个菜

**public** Menu() {

setMenuItem("土豆丝");

setMenuItem("豆芽");

setMenuItem("麻婆豆腐");

setMenuItem("西红柿");

}

**public** **void** **setMenuItem**(String item) {

menus[position++] = item;

}

// 通过Iterator方法获取迭代器对象

**public** Iterator<String> iterator() {

**return** **new** MenuIterator<String>(menus);

}

}

MenuIterator**菜单迭代器**：

**public** **class** MenuIterator<String> **implements** Iterator<String> {

**int** position = -1;

String[] menus;

**public** MenuIterator(String[] menus) {

**this**.menus = menus;

}

@Override

**public** **boolean** hasNext() {

position++;

**if** (position < 0 || position >= menus.length) {

**return** **false**;

}

**return** **true**;

}

@Override

**public** String next() {

**return** menus[position];

}

}

测试：

**public** **static** **void** main(String[] args) {

Menu menu1 = **new** Menu();

Iterator<String> iterator = menu1.iterator();

**while** (iterator.hasNext()) {

System.***out***.println(iterator.next());

}

}



对于Collection下面的集合类的迭代器同样是这种方法获取的。

对于Map<K,V>，只可以获取keys或values的迭代器，因为通过Map的keySet或values方法就可以获取一个Collection的实现类或Set的实现类。





## 实现ArrayList的迭代器

**Menu类**：

**public** **class** Menu **implements** Iterable<String> {

ArrayList<String> arrayList = **new** ArrayList<String>();

**public** Menu() {

arrayList.add("小鸡炖蘑菇");

arrayList.add("黄瓜肉丝");

arrayList.add("油泼面");

arrayList.add("北京布丁");

}

**public** **void** addItem(String item) {

arrayList.add(item);

}

@Override

**public** Iterator<String> iterator() {

**return** arrayList.iterator();

}

}

MyIterator：迭代器

**public** **class** MyIterator<String> **implements** Iterator<String> {

ArrayList<String> arrayList;

**int** position = -1;

**public** MyIterator(ArrayList<String> arrayList) {

**this**.arrayList = arrayList;

}

@Override

**public** **boolean** hasNext() {

position++;

**if** (position < 0 || position >= arrayList.size()) {

**return** **false**;

}

**return** **true**;

}

@Override

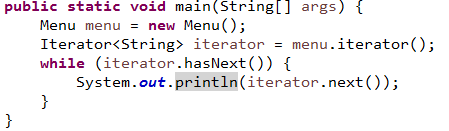
**public** String next() {

**return** arrayList.get(position);

}

}

测试;





## 借助Collection的Iterator方法：

**public** **class** Menu **implements** Iterable<String> {

**Hashtable<Integer, String> hashTable = new Hashtable<Integer, String>();**

**public** Menu() {

hashTable.put(0, "西瓜");

hashTable.put(1, "冬瓜");

hashTable.put(2, "南瓜");

hashTable.put(3, "北瓜");

}

**public** **void** addItem(**int** num, String item) {

hashTable.put(num, item);

}

@Override

**public** Iterator<String> iterator() {

Collection<String> collection = hashTable.values();

**return collection.iterator()**;

}

}

