Android设计模式源码解析之迭代器(Iterator)模式

> 本文为 Android 设计模式源码解析 中 迭代器模式 分析
> Android系统版本： 5.0
> 分析者： haoxiqiang，分析状态：完成，校对者：，校对状态：未完成

1. 模式介绍

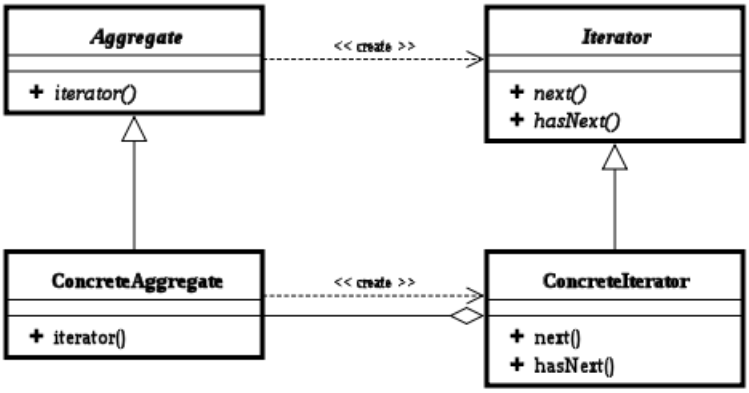
模式的定义

迭代器（Iterator）模式，又叫做游标（Cursor）模式。GOF给出的定义为：提供一种方法访问一个容器（container）对象中各个元素，而又不需暴露该对象的内部细节。

模式的使用场景

Java JDK 1.2 版开始支持迭代器。每一个迭代器提供next()以及hasNext()方法，同时也支持remove()(1.8的时候remove已经成为default throw new UnsupportedOperationException("remove"))。对Android来说,集合Collection实现了Iterable接口,就是说,无论是List的一大家子还是Map的一大家子,我们都可以使用Iterator来遍历里面的元素,可以使用Iterator的集合

2. UML类图



角色介绍

- 迭代器接口Iterator：该接口必须定义实现迭代功能的最小定义方法集比如提供hasNext()和next()方法。
- 迭代器实现类：迭代器接口Iterator的实现类。可以根据具体情况加以实现。
- 容器接口：定义基本功能以及提供类似Iterator iterator()的方法。
- 容器实现类：容器接口的实现类。必须实现Iterator iterator()方法。

3. 模式的简单实现

简单实现的介绍

我们有一个数组,对其遍历的过程我们希望使用者像ArrayList一样的使用,我们就可以用过iterator来实现.

实现源码

下面我们自己实现一个Iterator的集合

```
...
public Iterator<Mileage> iterator() {
    return new ArrayIterator();
}

private class ArrayIterator implements Iterator<Mileage> {
/**
 * Number of elements remaining in this iteration
```

```
 * Number of elements remaining in this iteration
 */
private int remaining = size;

/**
 * Index of element that remove() would remove, or -1 if no such elt
 */
private int removalIndex = -1;

@Override
public boolean hasNext() {
    return remaining != 0;
}

@Override
public Mileage next() {
    Mileage mileage = new Mileage();
    removalIndex = size-remaining;
    mileage.name = String.valueOf(versionCodes[removalIndex]);
    mileage.value = versionMileages[removalIndex];
    remaining-=1;
    return mileage;
}

@Override
public void remove() {
    versionCodes[removalIndex]=-1;
    versionMileages[removalIndex]="It was set null";
}
}
...
```

使用的过程如下,我们特意使用了remove方法,注意这个只是一个示例,和大多数的集合相比,该实现并不严谨

```
AndroidMileage androidMileage = new AndroidMileage();
Iterator<AndroidMileage.Mileage> iterator =androidMileage.iterator();
while (iterator.hasNext()){
    AndroidMileage.Mileage mileage = iterator.next();
    if(mileage.name.equalsIgnoreCase("16")){
        //remove掉的是当前的这个,暂时只是置空,并未真的移掉
        iterator.remove();
    }
    Log.e("mileage",mileage.toString());
}
```

下面直接写出几种集合的遍历方式,大家可以对比一下

- HashMap的遍历 `HashMap<String, String> colorMap=new HashMap<>(); colorMap.put("Color1","Red"); colorMap.put("Color2","Blue"); Iterator iterator = colorMap.keySet().iterator(); while( iterator. hasNext() ){ String key = (String) iterator.next(); String value = colorMap.get(key); }`
- JSONObject的遍历 `String paramString = "{menu:{\"1\":\"sql\", \"2\":\"android\", \"3\":\"mvc\"}}"; JSONObject menuJSONObj = new JSONObject(paramString); JSONObject menuObj = menuJSONObj.getJSONObject("menu"); Iterator iter = menuObj.keys(); while(iter.hasNext()){ String key = (String)iter.next(); String value = menuObj.getString(key); }`
  就目前而言，各种高级语言都有对迭代器的基本实现，没必要自己实现迭代器，使用内置的迭代器即可满足日常需求。

Android源码中的模式实现

一个集合想要实现Iterator很是很简单的,需要注意的是每次需要重新生成一个Iterator来进行遍历.且遍历过程是单方向的,HashMap是通过一个类似HashIterator来实现的,我们为了解释简单,这里只是研究ArrayList(此处以Android L源码为例,其他版本略有不同)

```
@Override public Iterator<E> iterator() {
    return new ArrayListIterator();
}

private class ArrayListIterator implements Iterator<E> {
    /** Number of elements remaining in this iteration */
    private int remaining = size;

    /** Index of element that remove() would remove, or -1 if no such elt */
    private int removalIndex = -1;

    /** The expected modCount value */
    private int expectedModCount = modCount;

    public boolean hasNext() {
        return remaining != 0;
    }

    @SuppressWarnings("unchecked") public E next() {
        ArrayList<E> ourList = ArrayList.this;
        int rem = remaining;
        if (ourList.modCount != expectedModCount) {
            throw new ConcurrentModificationException();
        }
        if (rem == 0) {
            throw new NoSuchElementException();
        }
        remaining = rem - 1;
        return (E) ourList.array[removalIndex = ourList.size - rem];
    }

    public void remove() {
        Object[] a = array;
        int removalIdx = removalIndex;
        if (modCount != expectedModCount) {
            throw new ConcurrentModificationException();
        }
        if (removalIdx < 0) {
            throw new IllegalStateException();
        }
        System.arraycopy(a, removalIdx + 1, a, removalIdx, remaining);
        a[--size] = null;  // Prevent memory leak
        removalIndex = -1;
        expectedModCount = ++modCount;
    }
}
```

- java中的写法一般都是通过iterator()来生成Iterator,保证iterator()每次生成新的实例
- remaining初始化使用整个list的size大小,removalIndex表示remove掉的位置,modCount在集合大小发生变化的时候后都会进行一次modCount++操作,避免数据不一致,前面我写的例子这方面没有写,请务必注意这点
- hasNext方法中,因为remaining是一个size->0的变化过程,这样只需要判断非0就可以得知当前遍历的是否还有未完成的元素
- next,第一次调用的时候返回array[0]的元素,这个过程中removalIndex会被设置成当前array的index
- remove的实现是直接操作的内存中的数据,是能够直接删掉元素的,不展开了

4. 杂谈

优点与缺点

优点

- 面向对象设计原则中的单一职责原则，对于不同的功能,我们要尽可能的把这个功能分解出单一的职责，不同的类去承担不同的职责。Iterator模式就是分离了集合

对象的遍历行为，抽象出一个迭代器类来负责，这样不暴露集合的内部结构，又可让外部代码透明的访问集合内部的数据。

缺点

- 会产生多余的对象，消耗内存；
- 遍历过程是一个单向且不可逆的遍历
- 如果你在遍历的过程中,集合发生改变,变多变少,内容变化都是算,就会抛出来ConcurrentModificationException异常.