# 迭代器模式简介

迭代器模式是一种使用频率非常高的设计模式。迭代器用于对一个聚合对象进行遍历。通过引入迭代器可以将数据的遍历功能从聚合对象中分离出来，聚合对象只负责存储数据，而遍历数据由迭代器完成，简化了聚合对象的设计，更符合单一职责原则的要求。

## 概述

1. 在软件开发中有些类可以存储多个成员对象（元素），这写类被称为聚合类（Aggregate Classes），对应的对象称为聚合对象。为了更加方便地操作这些聚合对象，同时可以很灵活地为聚合对象增加不同的遍历方法，可以访问一个聚合对象中的元素但又不希望暴露它的内部结构，此时可以引入迭代器实现对聚合对象中成员的遍历，还可以根据需要很方便地增加新的遍历方式。
2. 在软件系统中聚合对象拥有两个职责：一是存储数据；而是遍历数据。从依赖性来看，前者是聚合对象的基本职责，用于存储对象；而后者既是可变化的，又是可分离的。因此可以将遍历数据的行为从聚合对象中分离出来，封装在迭代器对象中，由迭代器来提供遍历聚合对象内部数据的行为，这将简化聚合对象的设计，更符合单一职责原则。
3. 迭代器模式（Iterator Pattern）的定义如下：提供一种方法顺序范文一个聚合对象中的各个元素，而又不用暴露该对象的内部表示。

4）迭代器模式又称游标（Cursor）模式，它是一种对象行为型模式。

## 结构

在迭代器模式结构中包含聚合和迭代器两个层次，聚合层次由聚合抽象类和具体聚合类组成，迭代器层次由抽象迭代器和具体迭代器组成。考虑到系统的灵活性和可扩展性，在迭代器模式中应用了工厂方法模式。其模式结构图如下：



1. Iterator（抽象迭代器）：它定义了访问和遍历元素的接口，声明了用于遍历数据元素的方法，例如用于获取第一个元素的first( )方法、用于访问下一个元素的next( )方法、用于判断是否还有下一个元素的hasNext( )方法、用于获取当前元素的currentItem( )方法等，在具体迭代器中将实现这些方法。
2. ConcreteIterator（具体迭代器）：它实现了抽象迭代器接口，完成对聚合对象的遍历，同时在具体迭代器中通过游标来记录在聚合对象中所处的当前位置，在具体实现时游标通常是一个表示位置的非负整数。
3. Aggregate（抽象聚合类）：它用于存储和管理元素对象，声明一个createIterator( )方法用于创建一个迭代器对象，充当抽象迭代器工厂角色。
4. ConcreteAggregate（具体聚合类）：它是抽象聚合类的子类，实现了在抽象聚合类中声明的createIterator( )方法，该方法返回一个与该具体聚合类对应的具体迭代器ConcreteIterator实例。

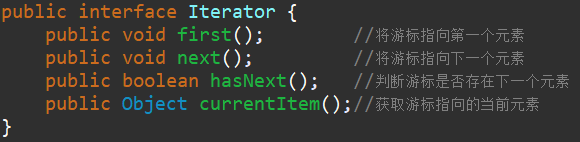
# 实现

## 实现原理

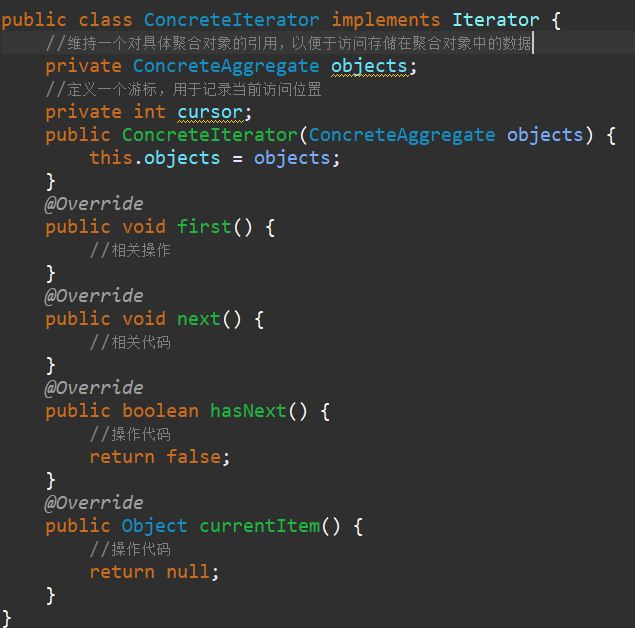
在迭代器模式中，提供了一个外部的迭代器来对聚合对象进行访问和遍历，迭代器定义了一个访问该聚合元素的接口，并且可以跟踪当前遍历的元素，了解哪些元素已经遍历过，哪些没有。

在迭代器模式中应用了工厂方法模式，抽象迭代器对应于抽象产品角色，具体迭代器对应于具体产品角色，抽象聚合类对应于抽象工厂角色，具体聚合类对应具体工厂角色。

1、在抽象迭代器中声明了用于遍历聚合对象中所存储元素的方法，其典型代码如下：

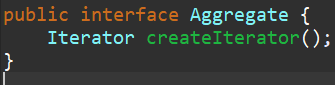


2、在具体迭代器中将实现抽象迭代器声明的遍历数据的方法，其典型代码如下：

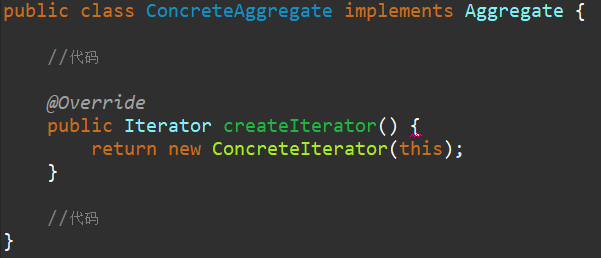


需要注意的是，抽象迭代器接口的设计非常重要，一方面需要充分满足各种遍历操作的要求，尽量为各种遍历方法都提供声明；另一方面又不能包含太多方法，接口中声明的方法太多将给子类的实现带来麻烦。因此可以考虑使用抽象类来设计抽象迭代器，在抽象类中为每一个方法提供一个空的默认实现。如果需要在具体迭代器中为聚合对象增加全新的遍历操作，则必须修改抽象迭代器和具体迭代器的源代码，这将违反开闭原则，因此在设计时要考虑全面，避免之后修改接口。

1. 聚合类用于存储数据并负责创建迭代器对象，最简单的抽象聚合类代码如下：



1. 具体聚合类作为抽象聚合类的子类，一方面负责存储数据，另一方面实现了在抽象聚合类中声明的工厂方法createIterator( )，用于返回一个与该具体聚合类对应的具体迭代器对象，典型代码如下：



## 实例

实例说明：某软件公司为某商场开发了一条销售管理系统，在对该系统进行分析和设计时，开发人员发现经常需要对系统中的商品数据、客户数据等进行遍历，为了复用这些遍历代码，开发人员设计一个抽象的数据集合类AbstractObjectList，而将存储商品和客户等数据的类设计为其子类，AbstractObjectList类的结构如图所示：



上图中List类型的对象objects用于存储数据，方法说明如下：

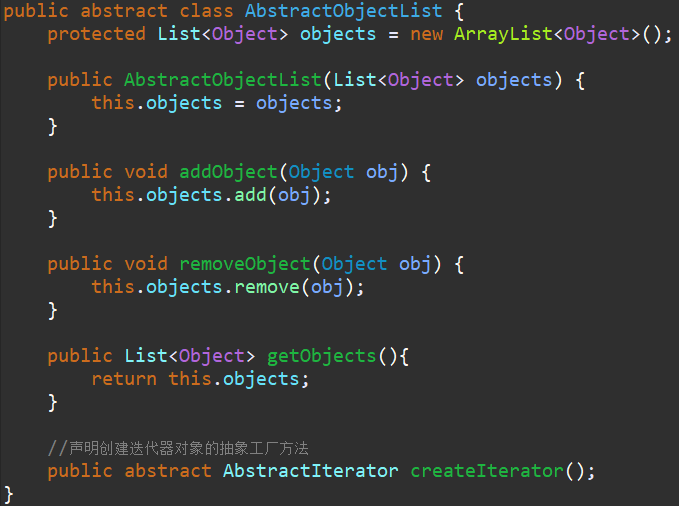
|  |  |
| --- | --- |
| **方法名** | **方法说明** |
| AbstractObjectList( ) | 构造方法，用于给objects对象赋值 |
| addObject( ) | 增加元素 |
| removeObject( ) | 删除元素 |
| getObjects( ) | 获取所有元素 |
| next( ) | 移至下一元素 |
| isLast( ) | 判断当前元素是否是最后一个元素 |
| previous( ) | 移至上一元素 |
| isFirst( ) | 判断当前元素是否是第一个元素 |
| getNextItem( ) | 获取下一个元素 |
| getPreviousItem( ) | 获取上一个元素 |

AbstractObjectList类的子类ProductList和CustomerList分别用于存储商品数据和客户数据。通过分析发现AbstractObjectList类的职责非常重，它既负责存储和管理数据，又负责遍历数据，违反了单一职责原则，实现代码将非常复杂。因此开发人员决定使用迭代器模式对AbstractObjectList类进行重构，将负责遍历数据的方法提取出来封装到专门的类中，实现数据存储和数据遍历分离，还可以给不同的具体数据集合类提供不同的遍历方式。现给出使用迭代器重构后的解决方案，结构图如下：

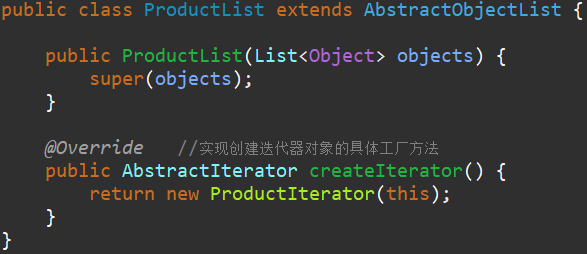


结构图中，AbstractObjectList充当抽象聚合类，ProductList充当具体聚合类，AbstractIterator充当抽象迭代器，ProductIterator充当具体迭代器。同时，为了详细说明自定义迭代器的实现过程，没有使用JDK中内置的迭代器，事实上JDK内置迭代器已经实现了一个List对象的正向遍历。

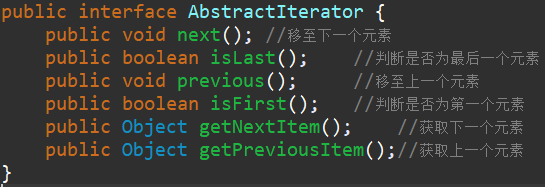
1. AbstractObjectList：抽象聚合类



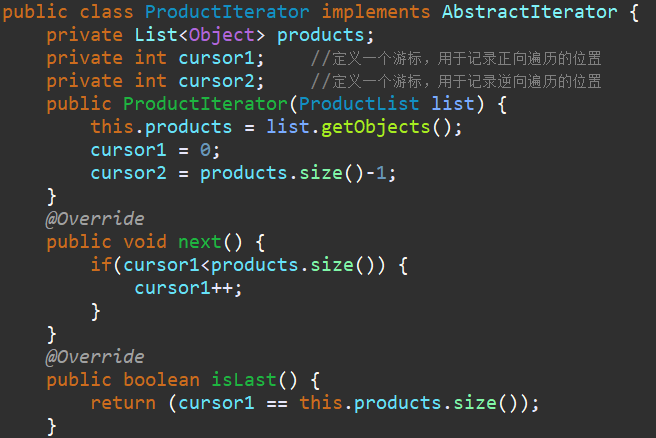
1. ProductList：商品数据类，充当具体聚合类。

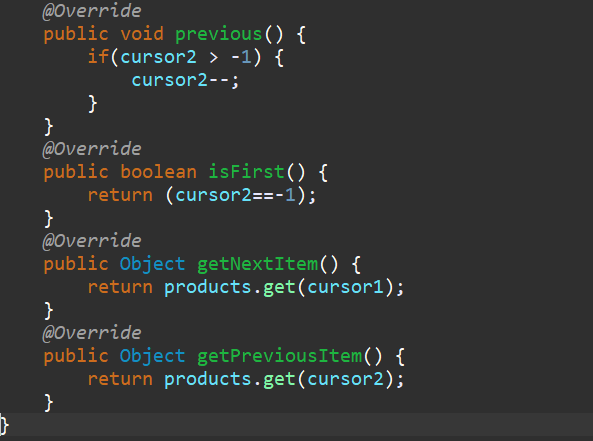


1. AbstractIterator：抽象迭代器

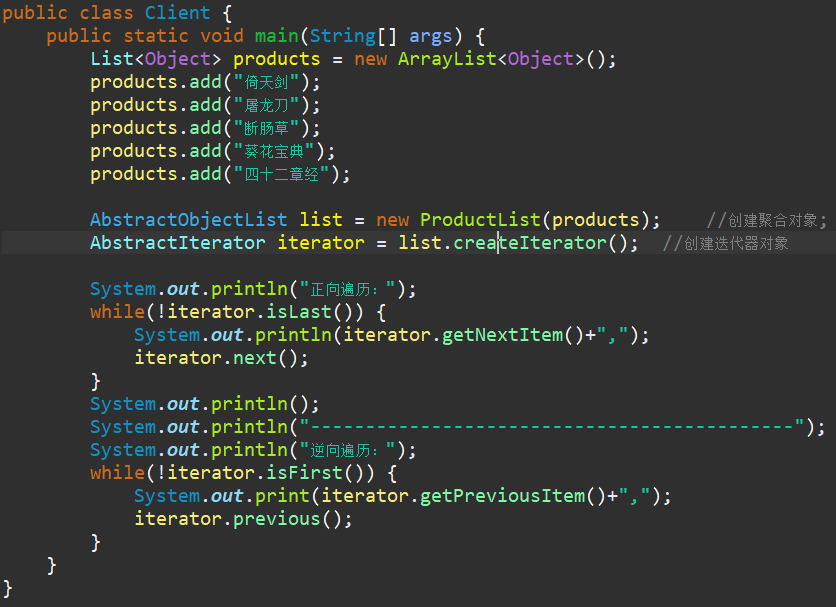


1. ProductIterator：商品迭代器，充当具体迭代器





1. Client：客户端测试类



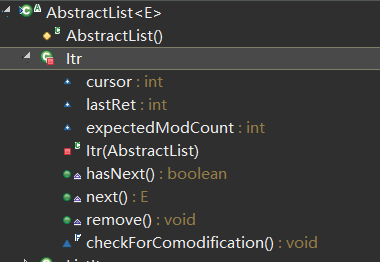
运行结果如下：



如果需要增加一个新的具体聚合类，例如客户数据集合类，并且需要为客户数据集合类提供不同于商品数据集合类的正向遍历和逆向遍历操作，只需增加一个新的聚合子类和一个新的具体迭代器类集合，原有类库代码无须修改，符合开闭原则；如果需要为ProductList类更换一个迭代器，只需要增加一个新的具体迭代器类作为抽象迭代器类的子类，重新实现遍历方法，原有迭代器代码无须修改，也符合开闭原则；但是如果要在迭代器中增加新的方法，则需要修改抽象迭代器源代码，这将违背开闭原则，所以，良好的抽象迭代器的设计很重要。

## 使用内部类实现迭代器

在上一节所示的迭代器模式结构图中，具体迭代器类和具体聚合类之间存在双重关系，其中一个关系为关联关系，在具体迭代器中需要维持一个对具体聚合对象的引用，该关联关系的目的是访问存储在聚合对象中的数据，以便迭代器能够对这些数据进行遍历操作。除了使用关联关系外，为了能够让迭代器可以访问到聚合对象中的数据，还可以将迭代器设计为聚合类的内部类，JDK中的迭代器类就是通过这种方法来实现的，如AbstractList类结构片段如下：



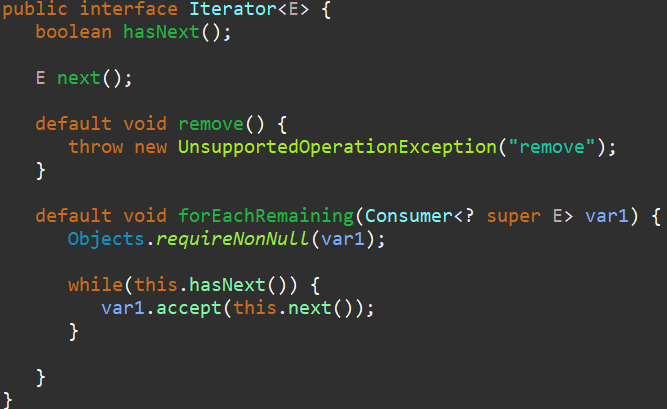
下面对上一节实例中的ProductList类进行修改，将ProductIterator类作为ProductList类的内部类，代码如下：



无论使用哪种实现机制，客户端代码都是一样的，也就是说客户端无须关心具体迭代器对象的创建细节，只需通过调用工厂方法createIterator( )即可得到一个可用的迭代器对象，这也是使用工厂方法模式的好处，通过工厂来封装对象的创建过程，简化了客户端的调用。

## JDK内置迭代器

为了让开发者能够更加方便地操作聚合对象，在Java、C#等编程语言中都提供了内置迭代器。在Java集合框架中，常用的List和Set等聚合类都继承或实现了java.util.Collection接口，在Collection接口中存在一个iterator( )方法，用于返回一个Iterator类型的迭代器对象，以便遍历聚合中的对象。在Iterator类型的迭代器中，存在了判断是否有元素、取出元素等操作。





# 优缺点及适用环境

迭代器模式是一种使用频率非常高的设计模式，通过引入迭代器可以将数据的遍历功能从聚合对象中分离出来，聚合对象只负责存储数据，而遍历数据由迭代器来完成。由于很多编程语言的类库都已经实现了迭代器模式，因此在实际开发中只需要直接使用Java、C#等语言已定义好的迭代器即可，迭代器已经成为操作聚合对象的基本工具之一。

## 优点

迭代器模式的优点主要如下：

1. 迭代器模式支持以不同的方式遍历一个聚合对象，在同一个聚合对象上可以定义多种遍历方式。在迭代器模式中只需要用一个不同的迭代器来替换原有迭代器即可改变遍历算法，也可以自己定义迭代器的子类以支持新的遍历方式。
2. 迭代器模式简化了聚合类。由于引入了迭代器，在原有的聚合对象中不需要再自行提供数据遍历等方法，这样可以简化聚合类的设计。
3. 在迭代器模式中由于引入了抽象层，增加新的聚合类和迭代器类都很方便，无须改原有代码，满足开闭原则。

## 缺点

迭代器模式的缺点主要如下：

1. 由于迭代器模式将存储数据和遍历数据的职责分离，在增加新的聚合类时需要对应增加新的迭代器类，类的个数成对增加，这在一定程度上增加了系统的复杂性。
2. 抽象迭代器的设计难度较大，需要充分考虑到系统将来的扩展。在自定义迭代器时创建一个考虑全面的抽象迭代器并不是一件很容易的事情。

## 适用环境

在以下情况下可以考虑适用迭代器模式：

1. 访问一个聚合对象的内容而无需暴露它的内部表示。将聚合对象的访问与内部数据的存储分离，使得访问聚合对象时无须了解其内部实现细节。
2. 需要为一个聚合对象提供多种遍历方法。
3. 为遍历不提供的聚合结构提供一个统一的接口，在该接口的实现类中为不同的聚合结构提供不同的遍历方式，而客户端可以一致性地操纵该接口。