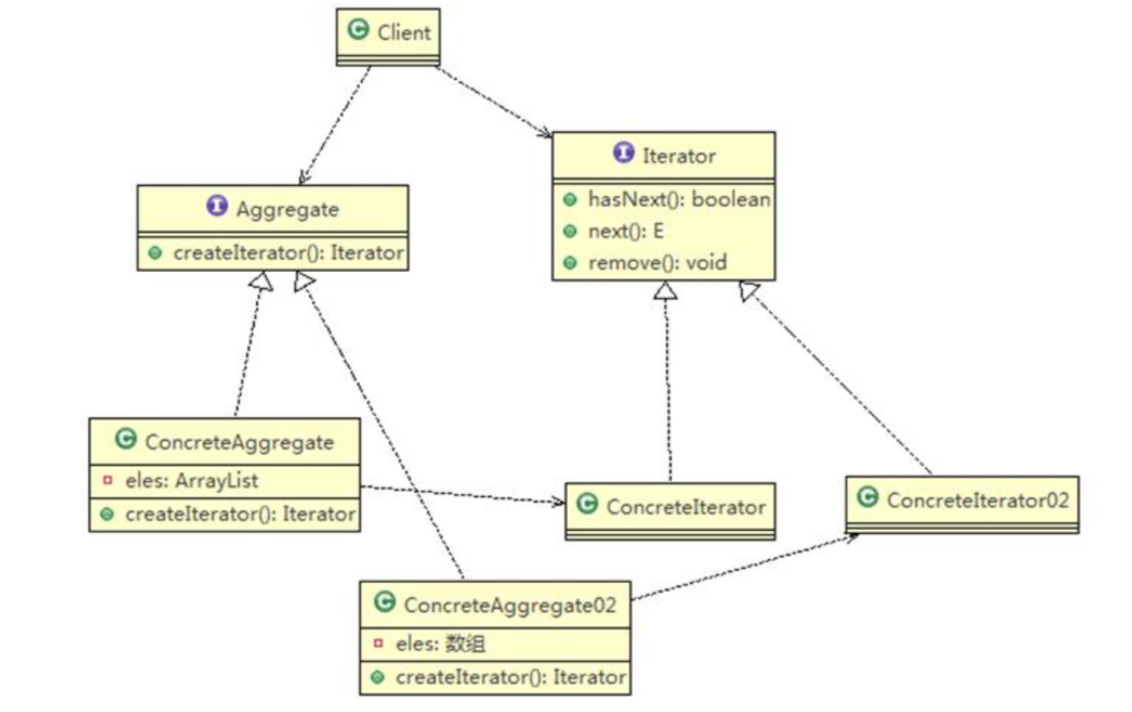
1. 迭代器模式定义：提供一种方法访问一个容器对象中各个元素，而又不暴露该对象的内部细节。
   1. 如果我们的集合元素是用**不同的方式**实现的，有数组，还有java的集合类，或者还有其他方式，当客户端要遍这些集合元素的时候就要使用多种遍历方式，而且还会暴露元素的内部结构，可以考虑使用迭代器模式解决
   2. 迭代器模式，提供一种遍历集合元素的**统一接口**，**用一致的方法遍历**集合元素，不需要知道集合对象的底层表示，即:不暴露其内部的结构
2. 类图



1. 迭代器模式的结构
   1. 抽象容器：一般是一个接口，提供一个iterator()方法，例如java中的Collection接口，List接口，Set接口等
   2. 具体容器：就是抽象容器的具体实现类，比如List接口的有序列表实现ArrayList，List接口的链表实现LinkedList，Set接口的哈希列表的实现HashSet等。
   3. 抽象迭代器：定义遍历元素所需要的方法，一般来说会有这么三个方法：取得下一个元素的方法**next**()，判断是否遍历结束的方法isDone()（或者叫**hasNext**()），移出当前对象的方法**remove**()
   4. 具体迭代器：实现迭代器接口中定义的方法，完成集合的迭代
2. 例

|  |
| --- |
| 1. 抽象容器   public interface Aggregate {   void addElement(Object ele)**;** Iterator<Object> iterator()**;** }   1. 采用List存储的具体容器   public class ListAggregate implements Aggregate {   private List<Object> elements**;** public ListAggregate() {  elements = new ArrayList<>(16)**;** }   @Override  public void addElement(Object ele) {  elements.add(ele)**;** }   @Override  public Iterator<Object> iterator() {  return new ListIterator(elements)**;** } }   1. 采用Array存储的具体容器   public class ArrayAggregate implements Aggregate {   private Object[] elements**;** private int numberOfElement**;** public ArrayAggregate() {  numberOfElement = 0**;** elements = new Object[16]**;** }   @Override  public void addElement(Object ele) {  elements[numberOfElement] = ele**;** numberOfElement += 1**;** }   @Override  public Iterator<Object> iterator() {  return new ArrayIterator(elements)**;** } }   1. List的具体迭代器   public class ListIterator implements Iterator {   private List<Object> elements**;** private int cursor = 0**;** public ListIterator(List<Object> elements) {  this.elements = elements**;** }   @Override  public boolean hasNext() {  if (cursor >= elements.size()) {  return false**;** } else {  return true**;** }  }   @Override  public Object next() {  Object element = elements.get(cursor)**;** cursor += 1**;** return element**;** } }   1. Array的具体迭代器   public class ArrayIterator implements Iterator {   private Object[] elements**;** private int cursor = 0**;** public ArrayIterator(Object[] elements) {  this.elements = elements**;** }   @Override  public boolean hasNext() {  if (cursor >= elements.length - 1 || elements[cursor] == null) {  return false**;** } else {  return true**;** }  }   @Override  public Object next() {  Object element = elements[cursor]**;** cursor += 1**;** return element**;** } }   1. Client   public class Client {   public static void main(String[] args) {  Aggregate listAggregate = new ListAggregate()**;** listAggregate.addElement("123")**;** listAggregate.addElement("456")**;** Iterator<Object> listIterator = listAggregate.iterator()**;** while (listIterator.hasNext()) {  System.*out*.println(listIterator.next())**;** }   Aggregate arrayAggregate = new ArrayAggregate()**;** arrayAggregate.addElement("789")**;** arrayAggregate.addElement("012")**;** Iterator<Object> arrayIterator = arrayAggregate.iterator()**;** while (arrayIterator.hasNext()) {  System.*out*.println(arrayIterator.next())**;** }   } } |

1. 常见场景
   1. JDK内置的迭代器(List/Set)
2. 迭代器模式的注意事项和细节
   1. 提供一个**统一的方法遍历对象**，客户**不用再考虑聚合的类型**，使用一种方法就可以遍历对象了。
   2. 隐藏了聚合的内部结构，客户端要遍历聚合的时候只能取到迭代器，而不会知道聚合的具体组成
   3. 提供了一种设计思想，就是一个类应该只有一个引起变化的原因(叫做单一责任原则)。在聚合类中，我们把迭代器分开，就是要把管理对象集合和遍历对象集合的责任分开，这样一来集合改变的话，只影响到聚合对象。而如果遍历方式改变的话，只影响到了迭代器。
   4. 当要展示一组相似对象，或者遍历一组相同对象时使用, 适合使用迭代器模式