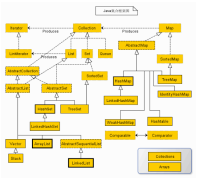

Java集合【微信公众号：编码的艺术】

https://github.com/muyulingfeng/smartPSndMap

基础框架

- 在线了解海的数据结构：集合、链表、队列、堆、数组、哈希、Java集合工具包官网是java.util。
  可以把它为了把它：List用作、Set集合、Map相同、工具类Iterator迭代器、Enumeration枚举类、Arrays和Collections

- 集合类是Collection接口派生出来的容器类，Java中还有提供这个接口的直接的实现类，但是由以其根据具体产生了两个接口，即是Map和Set，Set中每加进要放的元素，List是一个有序的集合，可以放进集合里的元素，提供了按索引访问的方式。

- Collection接口
  - 是复杂最基本的集合，它作为了集合的基本操作：添加、删除、清空、遍历迭代器、是否为空、获取大小、是否拥有某元素

- Map是Java.util包中的另一个接口，它和Collection接口没有关系，是相互独立的，但是都属于集合类的一部分。Map包含了key-value对。Map不能包含重复的key，但是可以包含相同的value。

- Map接口
  - 介、Map概念
    - 实现类：HashMap、Hashtable、LinkedHashMap/Treemap
    - 特点
      - HashMap是最常用的key、它根据键的hashCode值存储数据，根据键可以直接获取它的值，具有很快的访问速度，最坏时，哈希值碰撞的数据量太多时，会导致键值多于比较一致会影响速度O(n)，比多多数情况时O(1)。比多多数情况时O(1)。是非同步
      - 特点
        - Hashtable与HashMap类似，是HashMap的遗留版本，它是线程安全的，但是一般情况下一个键值和多个hashcode，因此在使用了HashMap和写入时会比较慢，它的性能不如HashMap，不同的是它不允许空值的键值和值。同时它是线程安全。
        - 特点
          - 线程安全，并且有序。ConcurrentHashMap内部使用Segment来表示这些不同的部分，每个段其实就是一个小的hash table，它们有自己的锁，只要多个修改操作发生在不同的段上，它们就可以并发进行。
          - 特点
            - LinkedHashMap保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的，也可以在构造的时候带参数，按照访问次序排序。
            - 特点
              - TreeMap实现SortedMap接口，能够把它保存的记录根据键排序，默认是按键的升序排序（自然顺序），也可以指定排序的比较器，当用Iterator遍历TreeMap时，得到的记录是排序的。不支持Null值。非同步的。

- List接口
  - 最多下面的Collection的接口，它代表有序的序列。
    - ArrayList 是一个数组结构，相当于动态数组，它的数组实现，随机访问很容易，插入和删除很容易。
    - LinkedList 是一个双向链表，它在访问上相当不快捷，但对插入和删除时速度很快，LinkedList提供了许多方法，用于创建、插入、删除元素和获取、设置、查找元素。
    - Vector 是矢量队列，和ArrayList一样，它也是一个动态数组，是数据结构，但是ArrayList是非线程安全的，而Vector是线程安全的。
    - Stack 是栈，它继承了Vector，它的特性是：先进后出(FILO, First In Last Out)。

- Iterator
  - Iterator是设计迭代器，用于的集合类，都实现了Iterator接口，用于遍历集合中元素的迭代，主要包含以下三种方法：
    1.hasNext()是否还有下一个元素。
    2.next()获取下一个元素。
    3.remove()删除迭代结果。

- 遍历
  - 迭代器Iterator
    - ① Iterator：迭代器类，是最常用是的迭代方式。
    - ② ListIterator：是Iterator的子接口，专门用于操作List中的内容。
    - ③ Enumeration：是Vector、Hashtable的子接口，可以用以迭代枚举对象。
    - 4) for-foreach
    - 5) for-each

- 八个最常的接口和类
  - 1、List（有序、可重复）
    List是有序的对象集合序列，同时也有可以重复的元素。List的实现类有很多，例如一系列有顺序的对象，其实就是数组，因为List是次数组入数据结构，也有链表结构。也有数组结构的结构，所有插入删除的速度很慢。
    2、Set（无序、唯一）
    Set是无序的对象集合序，不能重复的，集合中的对象不按特定的方式排序，并且保证插入的元素是唯一的。
    3、Map（键值对，键唯一，值可重一）
    Map是集合中的键的对象集序，键不能重复，值可以重复，根据键的特性，Map类型通常可以为键的Set表示，所以Set集合类的基本特性，也适用于Map。

	重复可能	有序性	排序实现可能性能
Collection			
List	是	是	
Set	AbstractSet	否	
HashSet	否	否	是
TreeSet	是（第二实现特性）		
Map	AbstractMap	否	
HashMap	否	否	否(Map.entrySet()返回的Set是HashSet的实现)
TreeMap	是（第二实现特性）		

- 区别小结
  - 不同点：
    1.线程安全性不一样
    ArrayList是非线程安全；
    是Vector是线程安全的，它的函数都是synchronized的，即都是支持同步的，线程上比ArrayList慢重。
    2.扩容机制不一样：ArrayList是1.5倍扩容的，Vector是1倍扩容的。
    2.对用户的支持不同：
    ArrayList支持用户的，而Vector不支持，因为ArrayList是实现java.io.Serializable接口，而Vector没有实现该接口。
    3.性能速度不一样不同
    ArrayList每个和元素的，而Vector每个和元素的，Vector除了线程的ArrayList其实的不同性能速度之外，两者的一个性能是即可以每次数据量增加多少。
    4.如果要多线程的话，ArrayList的速度要快，而Vector的速度要慢。
    5.如果要多线程的数据量增加多少的话，Vector每次增加的数量是原数的100%，而ArrayList每次增加的数量是原数的50%。
    6.ArrayList，如果必要多种遍历数据量之间大小的数，而Vector是一定的元素。
    - 相同点：
      1.它们都是List
      2.它们都实现了RandomAccess和Cloneable接口
      3.它们都通过数组实现的，本质上是数组的数据
      4.它们的默认数据容量都为
      5.它们都为Iterator和ListIterator遍历
    - ArrayList和LinkedList
      1.ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。
      2.对于随机访问get和set，ArrayList优于LinkedList，因为ArrayList可以以恒定的时间内在随机数组下内，找到第一个元素。
      3.对于添加和删除操作add和remove，LinkedList比较占优势，因为ArrayList要移动数据，这一点要看实际情况的。
      4.对于内存的使用，LinkedList还会为每个节点的数据提供内存开销，ArrayList的开销是不同的，因为ArrayList的开销是由于ArrayList所预留大小造成的，而ArrayList每次增加一条数据，都要扩容数组，而且是一定的时间开销的。
      5.LinkedList的开销是与ArrayList，因为ArrayList每次增加一条数据，都要扩容数组，而且是一定的时间开销。
      6.在实现大小的基本实现的时间上，工作在此使用效率。
    - ArrayList
      (1) 如果你要的多数情况，使用过过List.Iterator更好点。
      1、HashMap是Hashtable的实现类的不同点：
      1、HashMap是Hashtable的实现类的不同点：
      1、HashMap允许空值和空键，但是Hashtable不允许。
      2、HashMap是非线程安全的，而Hashtable是线程安全的，并且，Hashtable是线程安全的。
      3、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      4、HashMap扩展了Hashtable的contains()方法，允许contains()测试一个值是否在Map中存在。
      5、HashMap提供了键值的set视图，而Hashtable提供了键值的set视图和键的set视图。
      6、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      7、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      (2) 如果你要的多数情况，使用过过List.Iterator更好点。
      1、HashMap是Hashtable的实现类的不同点：
      1、HashMap是Hashtable的实现类的不同点：
      1、HashMap是Hashtable的实现类的不同点：
      2、HashMap允许空值和空键，但是Hashtable不允许。
      3、HashMap是非线程安全的，而Hashtable是线程安全的。
      4、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      5、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      6、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      7、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      (3) 如果你要的多数情况，使用过过List.Iterator更好点。
      1、HashMap是Hashtable的实现类的不同点：
      1、HashMap是Hashtable的实现类的不同点：
      1、HashMap是Hashtable的实现类的不同点：
      2、HashMap允许空值和空键，但是Hashtable不允许。
      3、HashMap是非线程安全的，而Hashtable是线程安全的。
      4、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      5、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      6、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
      7、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
    - HashMap为TreeMap
      1、用多把HashMap是线程安全的，也就是说是同步的，而HashMap是线程不安全的，不是同步的。
      2、HashMap允许空值一个Tjavaobjkey，为了为null的value，
      3、HashMap的迭代器是fail-fast迭代器，
      4、HashMap的初始容量为16，HashMap的增长因子默认是0.75
      HashMap扩容时是按照容量翻倍增长capacity*2，HashMap扩容时是容量翻倍=即capacity*2=!
      5、用多少要HashSet的方式时间
      HashMap扩容Hash表是按照容量翻倍capacity和HashMap的数组长度是按照计算的
      HashMap扩容时是HashMap的hashcode的到了二次hash，以获得更好的散列值，然后在table数组长度取模
    - HashTable为HashMap
      Java没有（generics）是JDK 5 中引入的一个特性，这是编译了编译时类型安全检测机制，该机制允许程序员在编译时检测到非法的类型。
      这段时间是指是参数化类型，当某类或对象的数据类型被定义为一个参数。
      - 传可以定义这种类型的一个类型参数声明部分（紧跟类名后面），该类型参数声明部分在方括号括起来且之后（在广度的字母后面）。
        - 每一个类型参数声明部分是由一个或多个类型参数，参数和逗号隔开。一个类型参数，也就有为一个类型变量，是用于指定一个引用类型的参数的标识符。
        - 类型参数被用来声明成员变量类型，并且用作方法型参数的返回类型参数类型的参数。
        - 方法可以接收声明其他方法一样，注意类型参数只能代表引用类型，不能是原始类型（即int,double,char的等）。
        -
          ```
          public class GenericMethodTest
          {
          // 泛型方法 printArray
          public static < E > void printArray( E[] inputArray )
          {
          // 输出数组元素
          for (E element : inputArray ){
          System.out.print(f ("%s ", element );
          }
          System.out.println();
          }

          public static void main( String-args[] )
          {
          // 创建不同类型数组：Integer, Double 和 Character
          Integer[] intArray = { 1, 2, 3, 4, 5 };
          Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
          Character[] charArray = { 'H', 'e', 'l', 'l', 'o' };

          System.out.println( "整型数组元素为:" );
          printArray( intArray ); // 传递一个整型数组

          System.out.println( "\n双精度数组元素为:" );
          printArray( doubleArray ); // 传递一个双精度型数组

          System.out.println( "\n字符型数组元素为:" );
          printArray( charArray ); // 传递一个字符型数组
          }
          }
          ```
        - 代码实例
        -
          ```
          1、定型通配符一般是使用于任意具体的泛型参数，例如List<?>在逻辑上是List<String>,List<Integer> 等所有List<具体类型>的父类。
          ```

泛型

- 定型方法
  -
    ```
    public class GenericMethodTest
    {
    // 泛型方法 printArray
    public static < E > void printArray( E[] inputArray )
    {
    // 输出数组元素
    for (E element : inputArray ){
    System.out.print(f ("%s ", element );
    }
    System.out.println();
    }

    public static void main( String-args[] )
    {
    // 创建不同类型数组：Integer, Double 和 Character
    Integer[] intArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
    Character[] charArray = { 'H', 'e', 'l', 'l', 'o' };

    System.out.println( "整型数组元素为:" );
    printArray( intArray ); // 传递一个整型数组

    System.out.println( "\n双精度数组元素为:" );
    printArray( doubleArray ); // 传递一个双精度型数组

    System.out.println( "\n字符型数组元素为:" );
    printArray( charArray ); // 传递一个字符型数组
    }
    }
    ```
  - 代码实例

- 类型通配符
  - 1、定型通配符一般是使用于任意具体的泛型参数，例如List<?>在逻辑上是List<String>,List<Integer> 等所有List<具体类型>的父类。

ArrayList

- 什么是ArrayList
  - ArrayList就是传说中的动态数组，其容量是自动增长，用户的操作的角度，我是ArrayList的数组实
  - 自定义的数组实现方式：
    实现了ICollection和IEnumerable
    ArrayList的数组实现方式：
    - 优点
      1. set(int index, E element)；
      2. add(E e)；
      3. add(int index, E element)；
      4. addAll(Collection<? extends E> c)；
      5. addAll(int index, Collection<? extends E> c)
    - 存储数组
      ArrayList实现了添加一个元素到容器的数组实现
      AddRange方法用来于添加一系列来到容器的数组实现
      Remove方法用于删除一个元素，通过元素来查找在其中数组
      RemoveAt方法用于删除一个元素，通过索引查找数组
      RemoveRange方法用于删除一个元素，通过索引和要删除的数量来删除
      Insert用于添加一个元素到容器以内，并通过指定的位置来设定容器
      InsertRange用于从指定位置开始添加一组元素，并通过指定的元素来设定容器
      Clear方法用于清除容器中的元素
      Contains方法用来查找某个对象在不在于容器之中
      ToArray方法 这个方法是ArrayList的方法和Array的一个重要的函数中。
      - 方法
        - 例子：
          ```
          ArrayList list = new ArrayList();
          List.Add(3);
          List.Add(2);
          List.Add(3);

          int[]d2 =(int[]-)list.ToArray(typeof(int[]));
          ```
        - ArrayList与数组转换
          - 例子：
            ```
            ArrayList list = new ArrayList();
            List.Add(1);
            List.Add(2);
            List.Add(3);

            int[]d2 =(int[]-)list.ToArray(typeof(int[]));
            int[]d3 =(int[]-)list.Count();
            List.Clear();
            ```
    - 1.增Add比的可变数组的区别，数据容量是多于数组定长的，即认知的容量为10；
      2.添加数据时，数组容量增加至原来数组容量的两倍，每单数组了指数量的增加时，数组容量增加，然后将数组数据复制到新的数组中。
      3.每次数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
      4.数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
      5.数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
      6.数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
      7.数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
      8.数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
      9.数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
      10.数组容量增加增大的是两倍容量的一半，即数组扩容容量的1.5倍。
    - 高效随增
      - 全容量数据查询，根据数组下表可以直接获取。
    - 源码

LinkedList

- 它也已以前自作编辑，以用前面编辑只自进行操作
  - 简介
    - (01) LinkedList是基于AbstractSequentialList，并且实现了Deque接口。
    - (02) LinkedList支持两个重要的成员：header 和 size。
      header是双向链表的表头，它是双向链表节点所对应的数据Entry的实例。Entry中包含成员变量：previous, next, element。其中，previous是该节点的上一个节点，next是该节点的下一个节点，element是该节点所保存的数据。
    - 继承了AbstractSequentialList的双向链表。
    - LinkedList 是有序的
    - LinkedList 实现List接口，能对它进行队列操作。
      LinkedList 实现Deque接口，即能将LinkedList当作双端队列来使用。
      LinkedList 实现Cloneable接口，即覆盖了函数clone()，能克隆。
      LinkedList 实现java.io.Serializable 接口，这意味着LinkedList支持序列化，能通过序列化去传输。
    -
      ```
      void add(int location, E object)
      boolean addAll(Collection<? extends E> collection)
      void addFirst(E object)
      void addLast(E object)
      void clear()
      Object clone()
      boolean contains(Object-object)
      E getFirst()
      E getFirst()
      E getLast()
      int indexOf(Object object)
      int lastIndexOf(Object-object)
      boolean offer(E o)
      boolean offerFirst(E e)
      boolean offerLast(E e)
      E peek()
      E peekFirst()
      E peekLast()
      E poll()
      E pollFirst()
      E pollLast()
      E pop()
      void push(E e)
      void remove(int location)
      E removeFirst()
      E removeLast()
      E set(int location, E object)
      int size()
      T[] T[] toArray(T[] contents)
      Object[] toArray()
      ```
    - 方法
    - LinkedList实现了双向链表结构的数据结构，能方便的插入和删除某个位置。
      2.随机访问LinkedList的速度比较慢，但是它也实现了List接口的随机访问。它实现了get(int location)和set(int location, E object)。这是因为LinkedList是双向链表，在随机访问的时候，要从表头和表尾开始遍历。
      3.实现栈和队列。它还提供了一个可直接当做栈和队列的API。栈，先进后出的数据结构。栈有两个操作：push()和pop()，分别是向栈中压入数据和从栈中弹出数据。而LinkedList的add()和remove()方法，则是向链表的末尾进行操作的。因此，使用LinkedList来实现栈和队列，是很合适的。
      4.实现多线程的List。LinkedList是非线程安全的。
      5.实现高效的List。LinkedList是双向链表，而ArrayList是数组。而LinkedList的add()和remove()方法，是在链表的末尾进行插入和删除节点的。这种操作的时间是固定的。而ArrayList的add()和remove()方法，是在数组的末尾进行插入和删除元素的。这种操作的时间是不固定的。
      6.实现Cloneable接口。LinkedList覆盖了函数clone()，能克隆。
      7.实现java.io.Serializable接口。LinkedList支持序列化，能通过序列化去传输。
    - 特点
      - (03) LinkedList是基于链表结构的数据结构。它是一个双向链表，每个节点（node）包含三个数据：前一个节点，下一个节点，节点数据。
      - (04) LinkedList的实现方式不同于以数组，它不是以一个数组来存储数据，而是以一个节点（node）来存储数据的。每个节点（node）保存：
        - (05) 上一个节点的位置（即：前一个节点的地址）；
        - (06) 下一个节点的位置（即：后一个节点的地址）；
        - (07) 当前节点的数据。
      - (08) LinkedList的节点是双向链表的节点，它的节点node的定义如下：
      - (09) LinkedList的迭代器是双向的，它的迭代器Iterator是基于节点的node的。
      - (10) LinkedList的迭代器是fail-fast的，它的迭代器Iterator的hasNext()和next()方法，都是先检查该迭代器的modCount是否等于该链表的modCount；若是，则抛出ConcurrentModificationException异常；否则，才执行hasNext()和next()的操作。
      - (11) LinkedList的add(int location, E object)，是先通过遍历，找到location位置的node；然后，将该node的前一个节点的node，该node，该node的后一个节点的node，三个node的数据，通过三个node的数据，重新生成三个node；最后，将该node的前一个节点的node，该node，该node的后一个节点的node，三个node的数据，通过三个node的数据，重新生成三个node；
      - (12) LinkedList的remove()，是先通过遍历，找到location位置的node；然后，将该node的前一个节点的node，该node，该node的后一个节点的node，三个node的数据，通过三个node的数据，重新生成三个node；
      - (13) LinkedList的get(int location)，是先通过遍历，找到location位置的node；然后，将该node的数据返回。
      - (14) LinkedList的set(int location, E object)，是先通过遍历，找到location位置的node；然后，将该node的数据，设为object。
    - 遍历方式
      - 第一种，通过迭代器遍历：即通过Iterator去遍历
        ```
        for(Iterator iter = list.iterator(); iter.hasNext();){
        iter.next();
        }
        ```
      - 第二种，通过随机访问，即通过size()和get()去遍历LinkedList
        ```
        int size = list.size();
        for (int i=0; i<size; i++) {
        list.get(i);
        }
        ```
      - 第三种，通过pollFirst()或者pollLast()来遍历LinkedList
        ```
        while(list.pollFirst() != null)
        ```
      - 第四种，通过removeFirst()或者removeLast()来遍历LinkedList
        ```
        try {
        while(list.removeFirst() != null);
        } catch (NoSuchElementException e) {
        }
        ```
    - 总结
    - 源码

面试题目

- 1.Java集合框架是什么？说说一些集合框架的优点？
  （1）提高核心集合类程序质量优良，而非常高效的设计。
  （2）增强使用自己产品的灵活集合框架，代码质量会有很高。
  （3）提高集合类的互操作性，提供一致的接口，可以很好的调用代码。
  （4）易于扩展和修改。

- 2.集合框架中的主要接口是什么？
  Java为集合类提供了完整的支持，这个支持包括实现高性能的集合、接口和实现集合的算法。通过使用这些接口，你可以控制一个集合的类型，你可以通过简单的迭代接口来遍历集合的元素，而且，如果你愿意的话，你可以扩展或修改标准的集合接口。
  Collection为集合层的根接口。一个集合代表一组对象，这些对象称为它的元素。一些集合允许具有相同的元素和无序的，其他的集合不允许重复的，一些集合是有序的，其他的集合是无序的。
  Set是一个不包含重复元素的集合。这个接口对数学集合抽象进行建模，被用来代表集合，就如一副牌。
  List是一个有序集合，可以包含重复元素。你可以通过它的索引来访问任何元素。List更像长度动态变换的数组。
  Map是一个将key映射到value的对象。一个Map不能包含重复的key：每个key最多只映射一个value。
  —每天它对接口有Queue、Deque、SortedSet、SortedMap和ListIterator。

- 3.Java集合框架的基础接口有哪些？
  Collection是最基本的集合接口，一个Collection代表一组Object，即Collection的元素（Elements）。一些Collection允许相同的元素而另一些不行。一些能按照特定一般方式，
  Iterator是设计迭代器的接口，用于对以上一个Collection中获得迭代器方法来实现迭代的顺序。通过使用迭代器Iterator可以在集合的元素上进行顺序操作。

- 4.为何Map接口不继承Collection接口？
  Map没有继承Collection接口，Map提供key到value的映射，一个Map中不能包含相同的key，每个key只能映射一个value。

- 5.Iterator是什么？
  Enumeration只提供了遍历一次的功能。Enumeration是早期版的，也是基于最初的容器，如：Vector和Hashtable的实现。
  Enumeration的接口，Iterator定义如下：
  提供了两个新方法来改进Enumeration的功能，它增加了一个叫做remove()的方法，这个方法可以在被调用的集合中删除调用next()方法返回的最近的一个元素。
  迭代器和枚举（Enumeration）的区别：
  （1）枚举（Enumeration）是早期版本的，它提供了两个基本的操作，hasMoreElements()和nextElement()，分别用于检查是否还有元素和获取下一个元素。
  （2）迭代器（Iterator）是较新的版本，它提供了三个操作：hasNext()、next()和remove()，分别用于检查是否还有元素、获取下一个元素和删除当前元素。
  （3）迭代器（Iterator）的方法名更短，更容易理解和使用，而且它支持在遍历集合时删除元素的操作，而枚举（Enumeration）则不支持这个操作。

- 6.迭代器Iterator和ListIterator之间有什么区别？
  （1）Iterator可以迭代数据，而ListIterator可以双向遍历。
  （2）ListIterator继承Iterator接口，提供额外的几个操作，比如添加一个元素，替换一个元素，获取一个元素，移除某个元素，获取当前元素的索引位置。

- 7.通过迭代器Iterator删除，你要做什么？
  每次调用迭代器的next方法之前，你可以调用迭代器的remove方法，这将会从底层集合中删除上一个被迭代的元素。
  Iterator是设计用于遍历集合的对象，但它也可以用来从集合中删除元素。每个被实现迭代器的Collection类都会返回一个迭代器。你可以通过调用Iterator的remove()方法来删除当前元素。remove()方法是可选的，如果底层的集合不支持删除操作，则会抛出一个UnsupportedOperationException。

- 8.请说出几种Iterator的实现方式？
  Iterator是设计用于遍历集合的对象，但它也可以用来从集合中删除元素。每个被实现迭代器的Collection类都会返回一个迭代器。你可以通过调用Iterator的remove()方法来删除当前元素。remove()方法是可选的，如果底层的集合不支持删除操作，则会抛出一个UnsupportedOperationException。

- 9.为何Iterator接口没有添加方法？
  没有方法增加迭代器的原因是fail-fast迭代器。比如，ArrayList迭代器是fail-fast的，而CopyOnWriteArrayList迭代器是fail-safe的。

- 10.Iterator和ListIterator之间有什么区别？
  HashMap和Hashtable的区别：
  HashMap和Hashtable都实现了Map接口，因此HashMap和Hashtable之间有很多相似之处。但是，它们的实现方式不同，以及它们的一些特点。这篇文章将会对HashMap和Hashtable做出一个比较。
  HashMap和Hashtable的区别：
  HashMap和Hashtable的区别主要体现在：线程安全性、同步（synchronization）、以及速度上的不同。
  1、HashMap是非线程安全的，而Hashtable是线程安全的。
  HashMap是非线程安全的，意味着它是线程不安全的。多线程可以同时修改HashMap的内部结构。而Hashtable是线程安全的，意味着它是线程安全的。
  2、HashMap允许空键和空值，而Hashtable不允许。
  HashMap允许空键和空值，而Hashtable不允许。
  3、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
  HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。这意味着，当一个线程在使用迭代器遍历HashMap的内容时，如果有另一个线程试图添加、移除或修改HashMap的内容，则会抛出一个ConcurrentModificationException异常。但是，这是基于尽力而为的原则做出的，并不能保证在将来的不确定时间内，会发生这种异常。因此，正确的做法是：这种异常只用于检测bug。
  4、HashMap扩展了Hashtable的contains()方法，允许contains()测试一个值是否在Map中存在。
  HashMap扩展了Hashtable的contains()方法，允许contains()测试一个值是否在Map中存在。这个方法返回true，如果指定的值存在于该Map中。这个方法的时间开销是很昂贵的，它将会在Map中搜索一个指定的值。这个方法返回true，如果指定的值存在于该Map中。
  5、HashMap提供了键值的set视图，而Hashtable提供了键值的set视图和键的set视图。
  HashMap提供了键值的set视图，而Hashtable提供了键值的set视图和键的set视图。这意味着，设计成操作键的set的程序，也可以应用于Hashtable的实例。
  6、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
  HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。这意味着，当一个线程在使用迭代器遍历HashMap的内容时，如果有另一个线程试图添加、移除或修改HashMap的内容，则会抛出一个ConcurrentModificationException异常。但是，这是基于尽力而为的原则做出的，并不能保证在将来的不确定时间内，会发生这种异常。因此，正确的做法是：这种异常只用于检测bug。
  7、HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。
  HashMap的迭代器是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。

- 11.请说出几种Iterator的实现方式？
  Iterator是设计用于遍历集合的对象，但它也可以用来从集合中删除元素。每个被实现迭代器的Collection类都会返回一个迭代器。你可以通过调用Iterator的remove()方法来删除当前元素。remove()方法是可选的，如果底层的集合不支持删除操作，则会抛出一个UnsupportedOperationException。

- 12.请说出几种fail-fast迭代器？
  fail-fast迭代器是一种设计用于检测bug的机制。fail-fast迭代器抛出ConcurrentModificationException异常。