集合框架

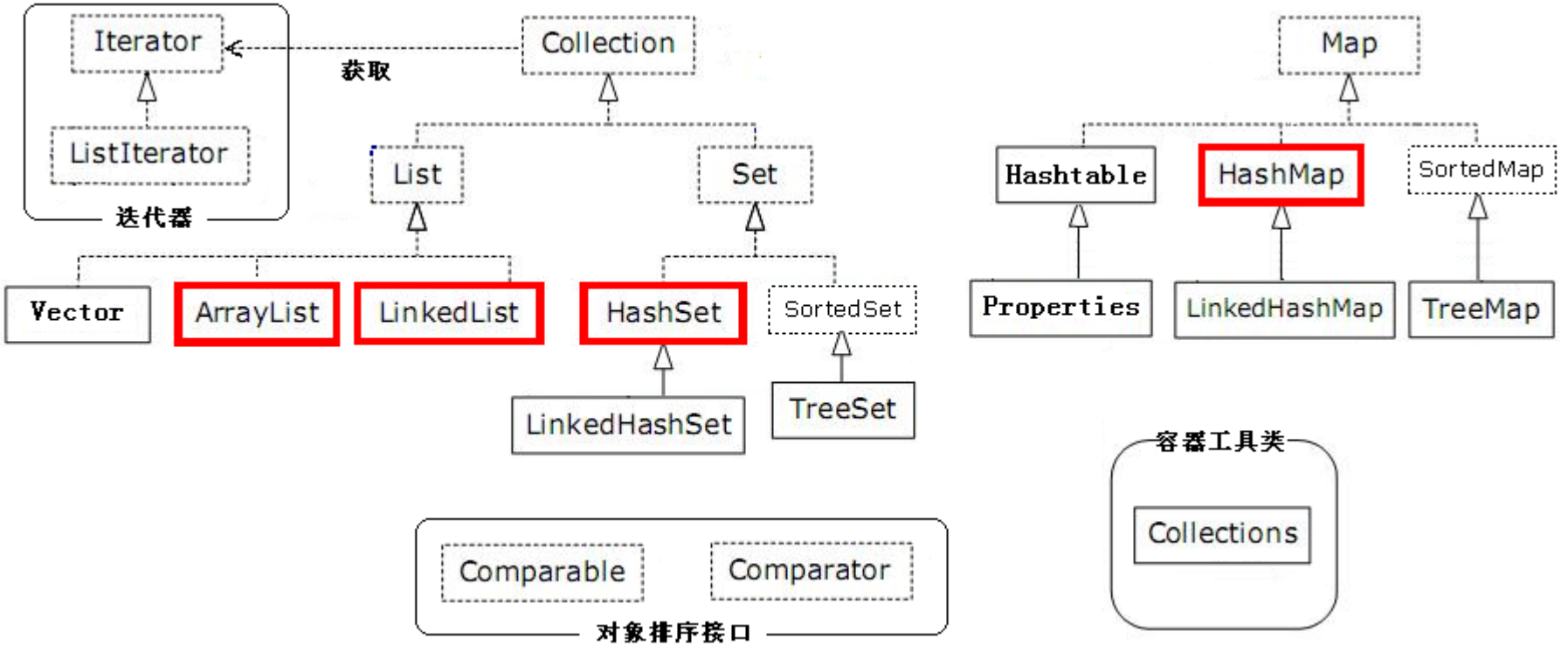
天津卓讯科技有限公司

第9章 集合框架

- 9.1 集合概述及API
- 9.2 Collection接口和Iterator接口
- 9.3 Set接口及常用实现类
- 9.4 List接口及常用实现类
- 9.5 Map接口及常用实现类
- 9.6 遗留集合类
- 9.7 排序集合
- 9.8 集合工具类Collections类
- 9.9 集合类的线程安全问题

9.1.1 集合概述

- 集合：Java SE API所提供的一系列类(java.util包内)的实例，可以用于动态存放多个对象。
- Java Collections Framework图如下：



9.1.2 集合API

- Collection接口——声明了一组管理它所存储元素的方法。常用子接口：
  - Set接口：存放的元素不包含重复的集合接口 （无序 不重复）
  - List接口：存放的元素有序且允许有重复的集合接口
- 说明：
  - “元素” - 对象，实例
  - “重复” - 两个对象通过equals相等
  - “有序” - 元素存入的顺序与取出的顺序相同
- Map接口——定义了存储“键(key)-值(value)映射对”的方法。

9.2.1 Collection接口

- Collection接口中定义的方法：
  - int size(); 返回此collection中的元素数。
  - boolean isEmpty(); 判断此collection中是否包含元素。
  - boolean contains(Object obj); 判断此collection是否包含指定的元素。
  - boolean contains(Collection<?> c); 判断此collection是否包含指定collection中的所有元素。
  - boolean add(E element); 向此collection中添加元素。
  - boolean addAll(Collection<? extends E> c);将指定collection中的所有元素添加到此collection中
  - boolean remove(Object element); 从此collection中移除指定的元素。
  - boolean removeAll(Collection<?> c); 移除此collection中那些也包含在指定collection中的所有元素。
  - void clear(); 移除些collection中所有的元素。
  - boolean retainAll(Collection<?> c); 仅保留此collection中那些也包含在指定collection的元素。
  - Iterator iterator(); 返回在此collection的元素上进行迭代的迭代器。
  - T[] toArray(T[] arr); 把此collection转成数组。

9.2.2 Iterator接口

- Iterator对象称作迭代器，用以方便的实现对集合内元素的遍历操作。Iterator接口中定义了如下方法：
  - boolean hasNext(); //判断游标右边是否有元素
  - Object next() ; //返回游标右边的元素并将游标移动到下一个位置
  - void remove(); //删除游标左面的元素
- 凡是能用 Iterator 迭代的集合都可以用JDK5.0中增强的for循环来更简便的遍历。

```
Collection<String> coll = ...;
for(String str : coll){
    System.out.println(str);
}
```

9.3 Set接口及常用实现类

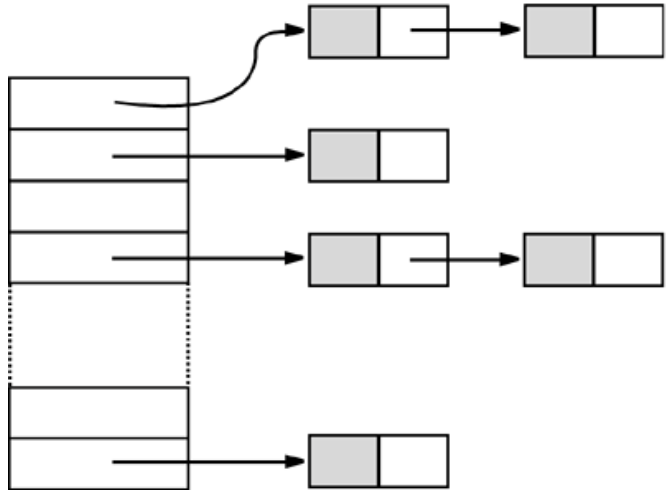
- Set接口介绍
- HashSet实现类
- LinkedHashSet实现类

9.3.1 Set接口

- Set接口没有提供Collection接口额外的方法，但实现Set接口的集合类中的元素是不可重复的。
  - 重复：两个对象equals()相等。
  - Set集合与数学中“集合”的概念相对应。
- JDK API中所提供的Set集合类常用的有：
  - HashSet
  - LinkedHashSet

9.3.2 HashSet类

- HashSet：不保存元素的存入顺序。
- HashSet根据元素的哈希码进行存放，取出时也可以根据哈希码快速找到。
- 使用示例

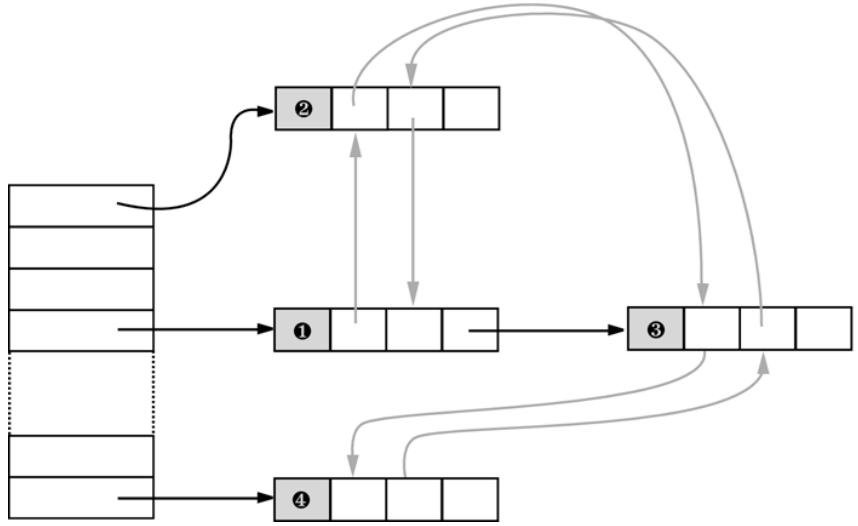


Set存放对象的原理

- 存对象时：
  - 根据每个对象的哈希码值(调用hashCode()获得)用固定的算法算出它的存储索引，把对象存放在一个叫散列表的相应位置(表元)中：
    - 如果对应的位置没有其它元素，就只需要直接存入。
    - 如果该位置有元素了，会将新对象跟该位置的所有对象进行比较(调用equals())，以查看是否已经存在了：还不存在就存放，已经存在就直接使用。
- 取对象时：
  - 根据对象的哈希码值计算出它的存储索引，在散列表的相应位置(表元)上的元素间进行少量的比较操作就可以找出它。
- Set系的集合存、取、删对象都有很高的效率。
- 对于要存放到Set集合中的对象，对应的类一定要重写equals()和hashCode(Object obj)方法以实现对象相等规则。

9.3.3 LinkedHashSet

•根据元素的哈希码进行存放，同时用链表记录元素的加入顺序。



9.4 List接口及常用实现类

- List接口
- ArrayList实现类
- LinkedList实现类

9.4.1 List接口

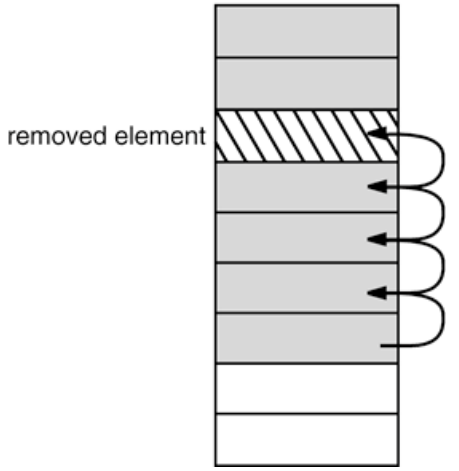
- 实现List接口的集合类中的元素是有序的，且允许重复。
- List集合中的元素都对应一个整数型的序号记载其在集合中的位置，可以根据序号存取集合中的元素。
- JDK API所提供的List集合类常用的有
  - ArrayList
  - LinkedList

9.4.1 List接口

- List接口比Collection接口中新增的几个实用方法：
  - public Object get(int index)
    - 返回列表中的元素数
  - public Object add(int index, Object element);
    - 在列表的指定位置插入指定元素.将当前处于该位置的元素（如果有的话）和所有后续元素向右移动
  - public Object set(int index, Object element) ;
    - 用指定元素替换列表中指定位置的元素
  - public Object remove(int index)
    - 移除列表中指定位置的元素
  - public ListIterator listIterator()
    - 返回此列表元素的列表迭代器

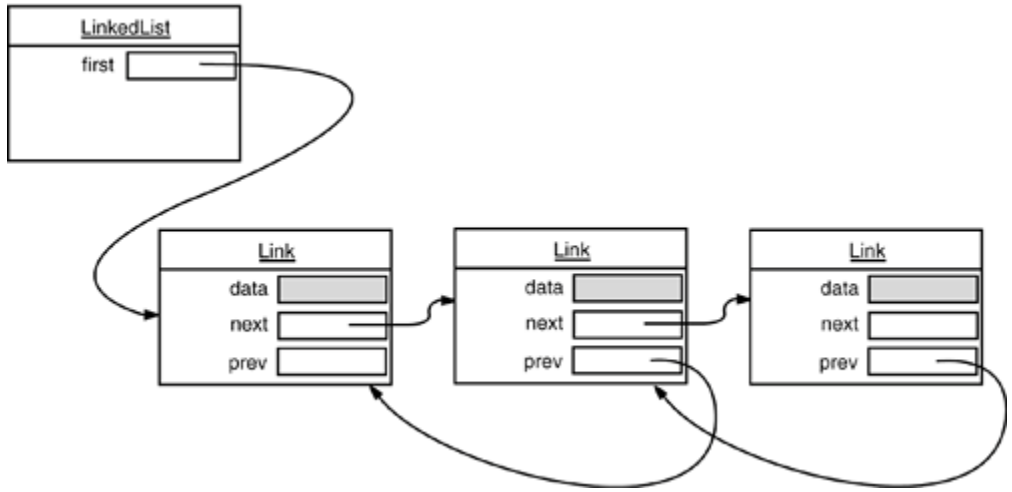
9.4.2 ArrayList

- ArrayList是使用数组结构实现的List集合。
- 优点：
  - 对于使用索引取出元素有较好的效率
    - 它使用索引来快速定位对象
- 缺点：
  - 元素做删除或插入速度较慢
    - 因为使用了数组，需要移动后面的元素以调整索引顺序。
- 示例



9.4.3 LinkedList

- LinkedList是使用双向链表实现的集合。
- LinkedList新增了一些插入、删除的方法。
- 优点：
  - 对频繁的插入或删除元素有较高的效率
  - 适合实现栈(Stack)和队列(Queue)
- 示例



9.5 Map接口及常用实现类

- Map接口——映射
- HashMap实现类
- LinkedHashMap实现类

9.5.1 Map接口

- 实现Map接口的集合类用来存储“键-值”映射对。
- JDK API中Map接口的实现类常用的有
  - HashMap
  - LinkedHashMap
- Map实现类中存储的“键-值”映射对是通过键来唯一标识，Map底层的“键”是用Set来存放的。
  - 所以，存入Map中的映射对的“键”对应的类必须重写hashcode()和equals()方法。常用String作为Map的“键”。

9.5.1 Map接口

- Map接口中定义的一些常用方法：
  - Object put(Object key, Object value); //将指定的“键-值”对存入Map中
  - Object get(Object key); //返回指定键所映射的值
  - Object remove(Object key); //根据指定的键把此“键-值”对从Map中移除。
  - boolean containsKey(Object key); //判断此Map是否包含指定键的“键-值”对。
  - boolean containsValue(Object value); //判断是否包含指定值的“键-值”对。
  - boolean isEmpty(); //判断此Map中是否有元素。
  - int size(); //获得些Map中“键-值”对的数量。
  - void clear(); //清空Map中的所有“键-值”对。
  - Set keySet(); //返回此Map中包含的键的Set集。
  - Collection values(); //返回此Map中包含的值的Collection集。
  - Set<Map.Entry<K,V>> entrySet(); //返回此Map中包含的“键-值”对的Set集

9.5.2 HashMap

- HashMap内部使用哈希表对 “键-值”映射对 进行散列存放。
  - 使用频率最高的一个集合。
- 示例

9.5.3 LinkedHashMap

- LinkedHashMap是HashMap的子类
  - 使用哈希表存放映射对
  - 用链表记录映射对的插入顺序。

9.6 遗留集合类

- 9.6.1 Vector
- 9.6.2 Stack
- 9.6.3 Hashtable
- 9.6.4 Properties

9.6.1 Vector

- 旧版的ArrayList，它大多数操作跟ArrayList相同，区别之处在于Vector是线程同步的。
- 它有一枚举方式可以类似Iterator进行遍历访问：

```
import java.util.Vector;
import java.util.Enumeration;
public class VectorTest{
        public static void main(String[] args){
                Vector<String> v = new Vector<String>();
                v.add("向量");
                v.add("123");
                v.add("abc");
                Enumeration<String> e = v.elements();
                while(e.hasMoreElements()){
                        System.out.println("---" + e.nextElement());
                }
        }
}
```

9.6.2 Stack类

- Stack类是Vector的子类，它是以后进先出（LIFO）方式存储元素的栈。
- Stack类中增加了5个方法对Vector类进行了扩展：

方法名	方法介绍
public E push(E item)	把元素压入堆栈顶部
public E pop()	移除堆栈顶部的对象，并作为此方法的值返回该对象
public E peek()	查看堆栈顶部的对象，但不从堆栈中移除它。
public boolean empty()	测试堆栈是否为空。
public int search(Object o)	返回对象在堆栈中的位置，以1为基数。

9.6.3 HashTable

- 旧版的HashMap，操作大多跟HashMap相同，只是它保证线程的同步。
- 它有一个子类Properties (属性集)比较常用

9.6.4 Properties

- Properties类表示了一个持久的属性集。Properties可保存在流中或从流中加载。属性集中每个键及其对应值都是一个字符串。
  - 不建议使用 put 和 putAll 这类存放元素方法，应该使用setProperty(String key, String value)方法，因为存放的"键-值"对都是字符串。类似取值也应该使用getProperty(String key)。
  - 不支持泛型操作。

Properties示例

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
public class PropertiesTest {
    public static void main(String[] args) {
        InputStream is = Thread.currentThread()
                                    .getContextClassLoader()
                                    .getResourceAsStream("config.properties");
        Properties prop = new Properties();
        try {
            prop.load(is);
        } catch (IOException e) {       e.printStackTrace();    }
        String name = prop.getProperty("name");
        String pwd = prop.getProperty("pwd");
        System.out.println(name + ", " + pwd);
    }
}
```

9.7 排序集合

- 9.7.1 Comparable接口
- 9.7.2 TreeSet类
- 9.7.3 Comparator接口
- 9.7.4 TreeMap类

9.7.1 Comparable接口

- 所有可“排序”的对象，对应的类都必须实现java.lang.Comparable接口，实现该接口中的唯一方法：
  - public int compareTo(Object obj);　该方法如果
    - 返回 0，表示 this == obj
    - 返回负数，表示 this < obj
    - 返回正数，表示 this > obj
- 可“排序”的类通过Comparable接口的compareTo方法来确定该类对象的排序方式。

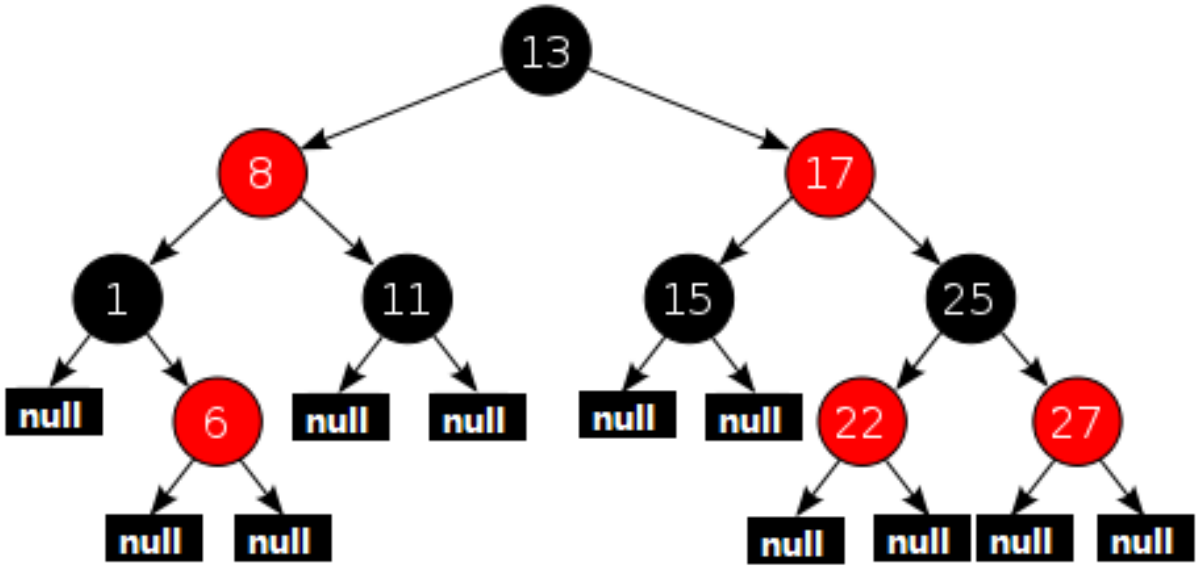
示例

```
public class Student  implements Comparable{
    private int id;         //编号
    private String name;   //姓名
    private double score;  //考试得分
    public Student(){}
    public Student(int id, String name, double score) {
        this.id = id;       this.name = name;       this.score = score;
    }
    //省略所有属性的getter和setter方法

    //实现compareTo方法，按成绩升序排序
    public int compareTo(Object o) {
        Student other = (Student)o;
        if(this.score > other.score){           return 1;
        }else if(this.score < other.score){           return -1;
        }else{           return 0;       }
    }
    public boolean equals(Object obj) {…}
    public int hashCode() { …}
}
```

9.7.2 TreeSet

- TreeSet使用红黑树结构对加入的元素进行排序存放，所以放入TreeSet中元素必须是可“排序”的。



9.7.3 Comparator接口

- 使用Comparable接口定义排序顺序局限性：实现此接口的类只能按compareTo()定义的这一种方式排序。
- 如果同一类型对象要有多种排序方式，应该为该类定义不同的比较器(实现Comparator接口的类)。
- Comparator接口中的比较方法：
  - public int compare(Object a, Object b);
    - 返回 0，表示 this == obj
    - 返回正数，表示 this > obj
    - 返回负数，表示 this < obj
- TreeSet有一个构造方法允许给定比较器，它就会根据给定的比较器对元素进行排序

多个比较器的示例

```
/** 学生考试得分比较器 */
class StudentScoreComparator implements Comparator<Student>{
    public int compare(Student o1, Student o2) {
        if(o1.getScore() > o2.getScore()){          return 1;
        }else if(o1.getScore() < o2.getScore()){          return -1;
        }else{          return 0;       }
    }
}
/** 学生姓名比较器 */
class StudentNameComparator implements Comparator<Student>{
    public int compare(Student o1, Student o2) {
        return o1.getName().compareTo(o2.getName());
    }
}
 //使用不同比较器的排序集合
Set<Student> set = new TreeSet<Student>(new StudentScoreComparator());
Set<Student> set2 = new TreeSet<Student>(new StudentNameComparator());
```

9.7.4 TreeMap

- TreeMap内部使用红黑树结构对“key”进行排序存放，所以放入TreeMap中的“key-value”对的“key”必须是可“排序”的。

小结：选择适当的集合

- HashSet、LinkedHashSet
- ArrayList、LinkedList
- HashMap、LinkedHashMap
- 选择标准：
  - 存放要求
    - 不重复 - Set
    - 有序 - List
    - “key-value”对 - Map
  - 读和改的效率
    - Hash* - 两者都最高
    - Array* - 读快改慢
    - Linked* - 读慢改快

9.8 Collections类

- java.util.Collections类是操作集合的工具类，提供了一些静态方法实现了基于集合的一些常用算法
  - void sort(List list)　根据元素的自然顺序 对指定List列表按升序进行排序。List列表中的所有元素都必须实现 Comparable 接口。
  - void shuffle(List list)　对List列表内的元素进行随机排列
  - void reverse(List list)　对List列表内的元素进行反转
  - void copy(List dest, List src)　将src列表内的元素复制到dest列表中
  - List synchronizedList(List list)　返回指定列表支持的同步(线程安全的)列表

9.9 集合类的线程安全

- 集合类大多数默认都没有考虑线程安全问题，程序必须自行实现同步以确保共享数据在多线程下存取不会出错：
  - 使用同步锁
    - synchronized(list){ list.add(…); }
  - 用java.uitl.Collections的synchronizedXxx()方法来返回一个同步化的容器对象
    - List list = Collections.synchronizedList(new ArrayList());
      - 这种方式在迭代时仍要用synchronized修饰

```
List list = Collections.synchronizedList(new ArrayList());
…
synchronized(list) {
        Iterator i = list.iterator();
        while (i.hasNext()) {
            foo(i.next());
        }
}
```

9.9 集合类的线程安全

- JDK5.0之后，新增了java.util.concurrent这个包，其中包括了一些确保线程安全的容器类，它们在效率与安全性上取得了较好的平衡。
  - ConcurrentHashMap
  - CopyOnWriteArrayList
  - CopyOnWriteArraySet

总结

- Collection接口
- Iterator接口
- Comparable接口、Comparator接口
- Set、List、Map接口
- Collections类
- ArrayList、LinkedList类
- HashSet、LinkedHashSet、TreeSet类
- HashMap、LinkedHashMap、TreeMap类
- Properties类