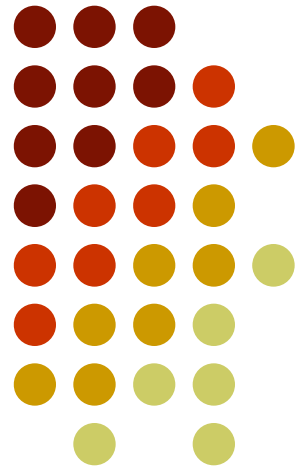


第9章 Java集合框架

1. 集合的概念
2. Collection接口与Iterator接口
3. LinkedList的使用
4. HashMap的使用
5. Vector的使用
6. Properties类的使用
7. Stack的使用

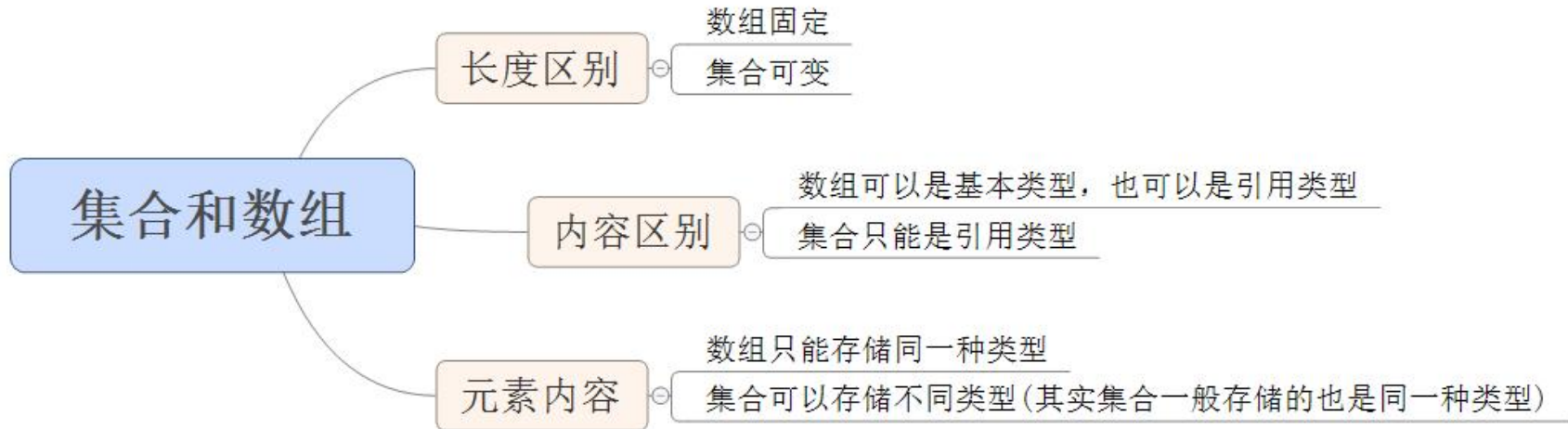




9.1集合的概念

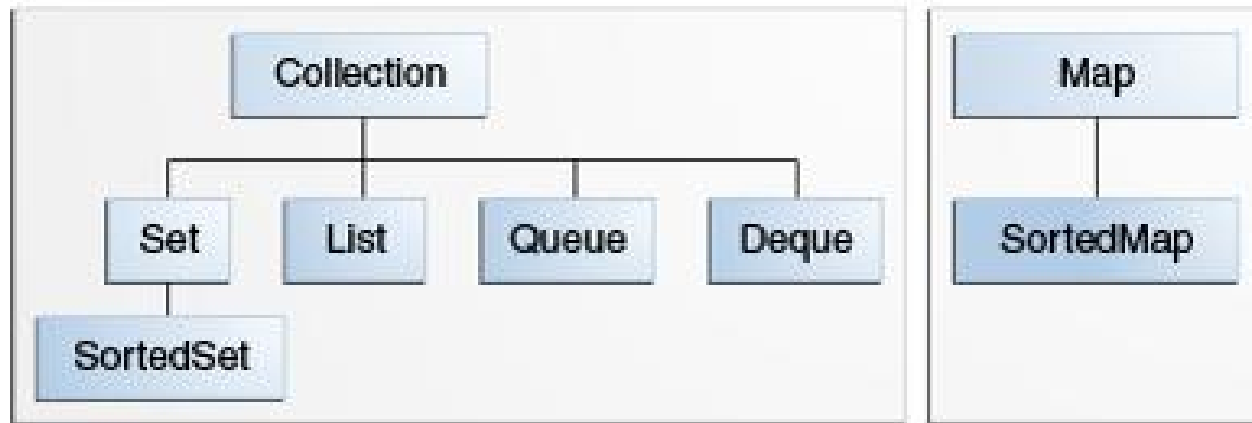
- 定义：集合是指具有某种特定性质的具体的或抽象的对象汇总而成的集体。其中，构成集合的这些对象则称为该集合的元素。
- 在编写面向对象的程序时，经常要用到一组类型相同的对象。可以使用数组来集中存放这些类型相同的对象,但数组一经定义便不能改变大小。因此，**Java**提供了一个集合框架(**Collections Framework**),该框架定义了一组接口和类，使得处理对象组更容易。
- 集合是数学中一个基本概念，也是集合论的主要研究对象。

集合和数组的区别

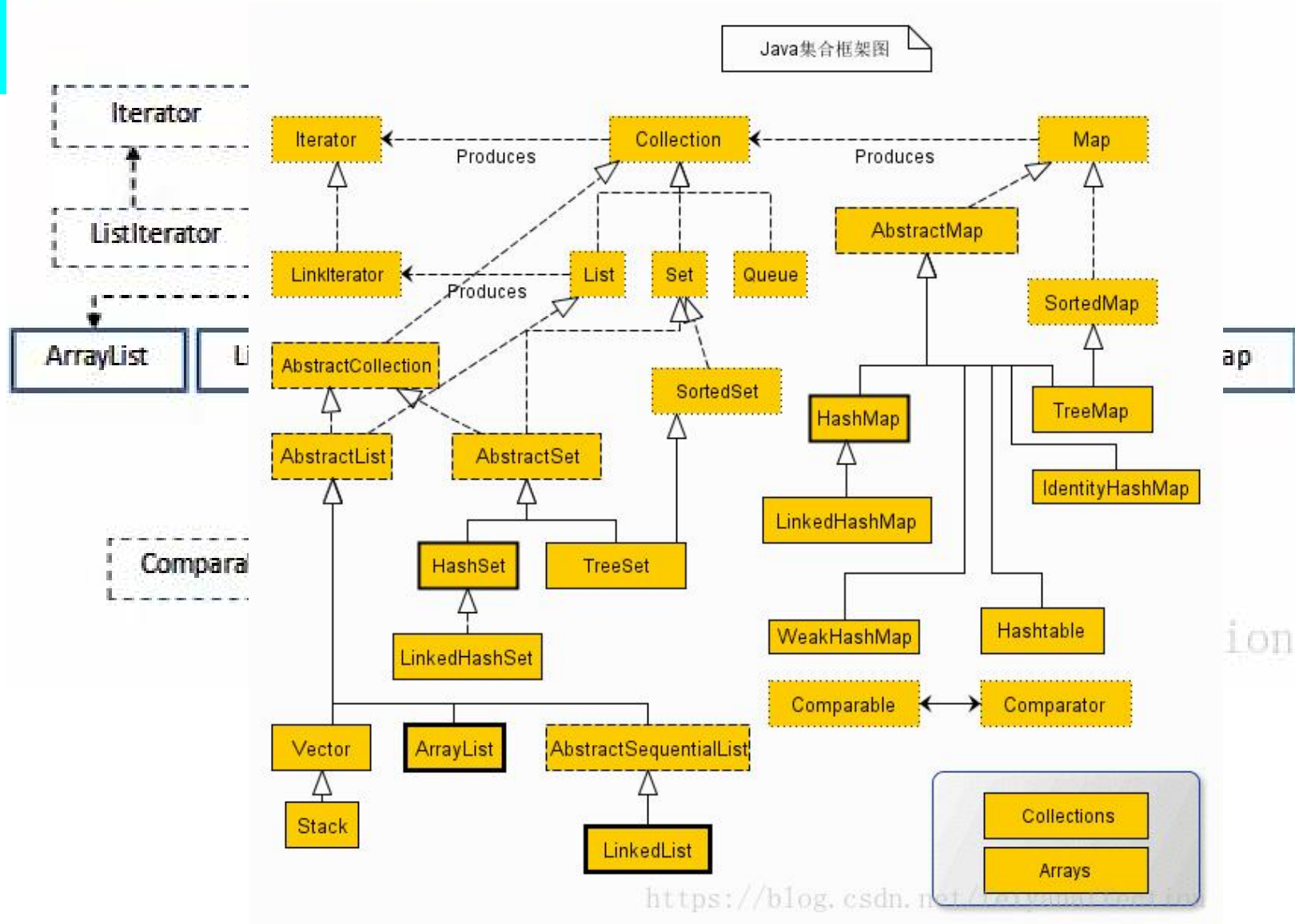


<http://shir/plogbtsgnccdn/feiyqqa34e49805>

- ① 菜鸟教程 <https://www.runoob.com/java/java-collections.html>
- ② feiyan的博客-CSDN博客_java集合
<https://blog.csdn.net/feiyanaffection/article/details/81394745>



2、集合框架结构图



<https://blog.csdn.net/feiyanaffecton/article/details/81394745>



Iterator接口

- Iterators - Similar to the familiar Enumeration interface, but more powerful, and with improved method names.
- ✓ **Iterator** - In addition to the functionality of the Enumeration interface, enables the user to remove elements from the backing collection with well-defined, useful semantics.
- ✓ **ListIterator** - Iterator for use with lists. In addition to the functionality of the Iterator interface, supports bidirectional iteration, element replacement, element insertion, and index retrieval.



java.util.Collection接口

➤ **Collection Interfaces**

- ✓ java.util.Set
- ✓ java.util.SortedSet
- ✓ java.util.NavigableSet
- ✓ java.util.Queue
- ✓ java.util.concurrent.BlockingQueue
- ✓ java.util.concurrent.TransferQueue
- ✓ java.util.Deque
- ✓ java.util.concurrent.BlockingDeque

- <http://docs.oracle.com/javase/tutorial/collections/index.html>
- <file:///F:/Java/docs/api/java/util/package-tree.html> <file:///D:/Java/docs/technotes/guides/collections/overview.html>



java.util.Map接口

- java.util.SortedMap
- java.util.NavigableMap
- java.util.concurrent.ConcurrentMap
- java.util.concurrent.ConcurrentNavigableMap



Class Hierarchy

- java.util.**AbstractCollection**<E> (implements java.util.Collection<E>)
 - java.util.**AbstractList**<E> (implements java.util.List<E>)
 - java.util.**AbstractSequentialList**<E>
 - java.util.**LinkedList**<E> (implements java.lang.Cloneable)
 - java.util.**ArrayList**<E> (implements java.lang.Cloneable)
 - java.util.**Vector**<E> (implements java.lang.Cloneable, java.util.Stack<E>)
 - java.util.**AbstractQueue**<E> (implements java.util.Queue<E>)
 - java.util.**PriorityQueue**<E> (implements java.io.Serializable)
 - java.util.**AbstractSet**<E> (implements java.util.Set<E>)
 - java.util.**EnumSet**<E> (implements java.lang.Cloneable)
 - java.util.**HashSet**<E> (implements java.lang.Cloneable)
 - java.util.**LinkedHashSet**<E> (implements java.lang.Cloneable)
 - java.util.**TreeSet**<E> (implements java.lang.Cloneable)
 - java.util.**ArrayDeque**<E> (implements java.lang.Cloneable, java.util.AbstractMap.Entry<K,V>)

Interface Hierarchy

- java.util.**Comparator**<T>
- java.util.**Enumeration**<E>
- java.util.**EventListener**
- java.util.**Formattable**
- java.lang.**Iterable**<T>
 - java.util.**Collection**<E>
 - java.util.**List**<E>
 - java.util.**Queue**<E>
 - java.util.**Deque**<E>
 - java.util.**Set**<E>
 - java.util.**SortedSet**<E>
 - java.util.**NavigableSet**<E>
- java.util.**Iterator**<E>
 - java.util.**ListIterator**<E>
 - java.util.**PrimitiveIterator**<T,T_CONS>
 - java.util.**PrimitiveIterator.OfDouble**
 - java.util.**PrimitiveIterator.OfInt**
 - java.util.**PrimitiveIterator.OfLong**
- java.util.**Map**<K,V>
 - java.util.**SortedMap**<K,V>
 - java.util.**NavigableMap**<K,V>
- java.util.**Map.Entry**<K,V>

<file:///F:/Java/docs/api/java/util/package-tree.html>



Collection Implementations

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	<u>HashSet</u>		<u>TreeSet</u>		<u>LinkedHashSet</u>
List		<u>ArrayList</u>		<u>LinkedList</u>	
Deque		<u>ArrayDeque</u>		<u>LinkedList</u>	
Map	<u>HashMap</u>		<u>TreeMap</u>		<u>LinkedHashMap</u>

double-ended queue ， 双端队列

<file:///Java/docs/technotes/guides/collections/index.html>

<file:///F:/Java/tutorial/collections/index.html>



集合概述

- 集合用于在类中对数据进行组织
- 集合是一种容器对象，用于按照一定的规则在其中保存一组对象
- 在J2SE中，引入了集合框架，它由集合库包构成（集合库包位于`java.util`包中）。该框架定义了许多用于实现集合的接口和抽象类，并且描述了某些机制，如迭代协议等
- 集合框架中共定义了9个集合接口：
 - ✓ 最基本的接口是 **Collection**
 - ✓ 5个接口扩展了 **Collection** 接口，它们是： **Set**， **List**， **SortedSet**， **Queue**， **BlockingQueue**
 - ✓ 其它的3个接口是： **Map**， **SortedMap**， **ConcurrentMap** 。这3个接口不扩展 **Collection** 接口



常用集合的分类

Collection 接口的接口 对象的集合（单列集合）

└—List 接口：元素按进入先后有序保存，可重复

|——└ LinkedList 接口实现类,链表,插入删除,没有同步,线程不安全

|——└ ArrayList 接口实现类,数组,随机访问,没有同步,线程不安全

|——└ Vector 接口实现类 数组， 同步， 线程安全

|——└ Stack 是Vector类的实现类

└—Set 接口： 仅接收一次，不可重复，并做内部排序

|——└ HashSet 使用hash表（数组）存储元素

|——└ LinkedHashSet 链表维护元素的插入次序

└——TreeSet 底层实现为二叉树，元素排好序



常用集合的分类

Map 接口 键值对的集合（双列集合）

- |——Hashtable 接口实现类， 同步， 线程安全
- |——HashMap 接口实现类， 没有同步， 线程不安全-
- |——| LinkedHashMap 双向链表和哈希表实现
- |——| WeakHashMap
- |——TreeMap 红黑树对所有的key进行排序
- |——IdentifyHashMap



Collection与Iterator接口

- Collection接口有下面两个基本方法：
 - ✓ `boolean add(Object obj)`
 - ✓ `Iterator iterator()`
 - ✓ `add`方法用于将对象添加给集合。如果添加对象后，该集合确实发生了变化，那么该方法返回**true**；如果该集合没有变化，则返回**false**。例如，如果你试图将一个对象添加给一个集合，而该集合中已经有该对象了，那么**add**请求将被拒绝，因为该集合拒绝纳入重复的对象
 - ✓ `iterator`方法用于返回一个实现了**Iterator**接口的类的对象。**Iterator**类型的对象称为迭代子对象，它专门用于访问集合中的各个元素

Collection与Iterator接口

- Iterator接口共配有下面3个方法：
 - ✓ **Object next()**//返回要访问的下一个对象
 - ✓ **boolean hasNext()**//如果存在另一个需要访问的元素，则返回true
 - ✓ **void remove**//用于删除上次调用next时返回的对象
 - ✓ 通过反复调用next方法，你可以逐个访问集合中的各个元素。但是，如果到达了集合的结尾，next方法便抛出了一个NoSuchElementException异常。因此，你必须在调用next方法之前调用hasNext方法。如果迭代子对象仍然拥有可供访问的元素，hasNext方法返回true
 - ✓ 想要查看集合中的所有元素，可使用如下方法：
 - **Iterator iter=c.iterator()**
 - **while(iter.hasNext()){**
 Object obj=iter.next();

 - **}**



Collection与Iterator接口

- 使用remove方法时必须小心:
 - ✓ 如何删除某个集合中的第一个元素:
 - ✓ `Iterator iter=c.iterator();`
 - ✓ `it.next();`//没有这一行将会抛出异常
 - ✓ `it.remove();`
 - ✓ 如何删除两个相邻的元素:
 - ✓ `it.remove();`
 - ✓ `it.next();`//没有这一行将会出错
 - ✓ `it.remove();`

具体的集合

- Java库提供了下面10个具体的集合类：
 - ✓ LinkedList
 - ✓ ArrayList
 - ✓ HashSet
 - ✓ TreeSet
 - ✓ HashMap
 - ✓ TreeMap
 - ✓ Vector
 - ✓ Stack
 - ✓ HashTable
 - ✓ Properties

集合的方法

Collection

`boolean add(E e)`

在集合末尾添加元素

`boolean remove(Object o)`

若本类集中有值与o的值相等的元素，则删除该元素，并返回true

`void clear()`

清除本类集中所有元素，调用完该方法后本类集将为空。

`boolean contains(Object o)`

判断集合中是否包含某元素

`boolean isEmpty()`

判断集合是否为空

`int size()`

返回集合中的元素个数

`boolean addAll(Collection c)`

将一个类集c中的所有元素添加到另一个类集

`Object[] toArray()`

返回一个包含了本类集中所有元素的数组，数组类型为：Object[]

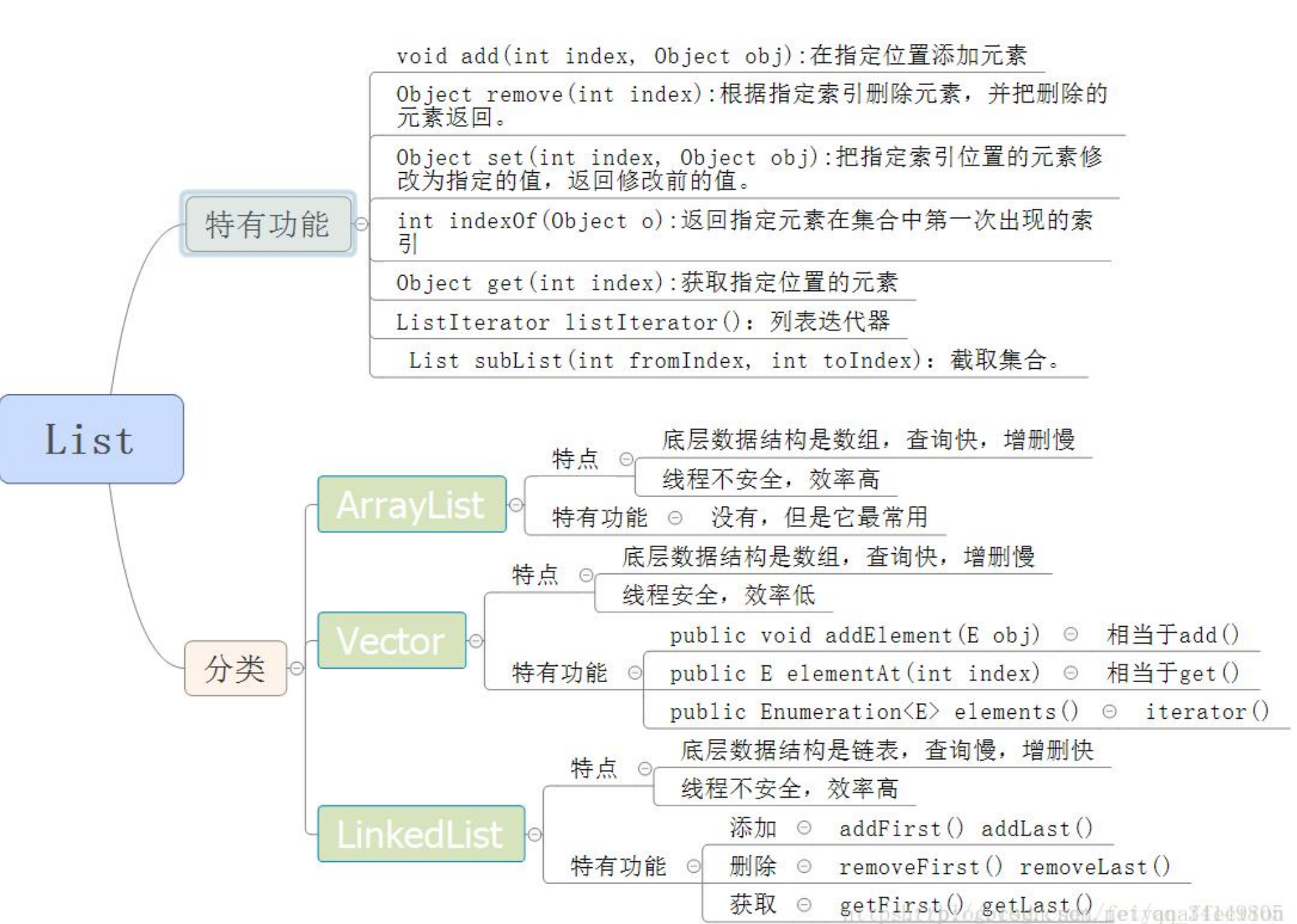
`Iterator iterator()`

迭代器，集合的专用遍历方式



List

- （1）**LinkedList** 底层数据结构是链表，查询慢，增删快，线程不安全，效率高，可以存储重复元素
- （2）**ArrayList**: 底层数据结构是数组，查询快，增删慢，线程不安全，效率高，可以存储重复元素
- （3）**Vector**: 底层数据结构是数组，查询快，增删慢，线程安全，效率低，可以存储重复元素





LinkedList 2-1

- 链接列表LinkedList实现了List接口
- 链接列表是一个有序集合，将每个对象存放在独立的链接中，每个链接中还存放着序列中下一个链接的索引。在java中，所有的链接列表实际上是双重链接的，即每个链接中还存放着对它的前面的链接的索引
- 链接列表适合于处理数据序列中数据数目不定，且频繁进行插入和删除操作的问题——插入或删除一个元素时，只需要更新其它元素的索引即可，不必移动元素的位置，效率很高

D:\JAVA\docs\api\java\util\LinkedList.html

LinkedList 2-2

- ListIterator（列表迭代子）接口中定义了许多方法用于访问链接列表LinkedList，LinkedList 类中有一个listIterator方法，可以返回一个ListIterator对象：
 - ✓ `ListIterator iter=list.listIterator();`
 - ✓ `Object oldValue=iter.next();`
 - ✓ `Iter.set(newValue);`
- 链接列表不支持快速随机访问。如果你想查看列表中的第n个元素，你必须从头开始查看，然后跳过前面的n-1个元素
- 如果你需要随机访问某个集合的话，请使用数组或者ArrayList，而不要使用LinkedList

LinkedList使用示例

```
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.util.Iterator;
public class LinkedListTest {
    public static void main(String[] args) {
        List<String> a = new LinkedList();
        a.add("艾米");
        a.add("卡尔");
        a.add("埃里卡");

        List b = new LinkedList();
        b.add("鲍勃");
        b.add("新格");
        b.add("弗朗斯");
        b.add("格里亚");

        // 把列表b和a合并起来
        ListIterator aIter = a.listIterator();
        Iterator bIter = b.iterator();

        while (bIter.hasNext()) {
            if (aIter.hasNext())
                aIter.next();
            aIter.add(bIter.next()); // 把b列表中的元素添
            // 加到a列表中
        }
        System.out.println(a);
    }
}
```

- 程序代码详见
LinkedListTest.java，程
序输出结果如下：



```
C:\PROGRA~1\XINOS~1\JCREAT~1\GE2001.exe
[艾米, 鲍勃, 卡尔, 新格, 埃里卡, 弗朗斯, 格里亚]
[鲍勃, 弗朗斯]
[艾米, 卡尔, 新格, 埃里卡, 格里亚]
Press any key to continue...
```



ArrayList

- 数组列表ArrayList实现了List接口
- ArrayList封装了一个动态再分配的Object[]数组，可以使用get和set方法来随机地访问其中的元素（对LinkedList不要使用这两个方法）
- ArrayList与Vector的作用大致相同，重要区别在于：**Vector类的所有方法都是同步方法，而ArrayList类的方法都是非同步方法。建议你在不需要同步时使用ArrayList类，以提高效率**

Vector

- 向量类Vector是一个“旧的”集合类。它与ArrayList类一样，封装了一个动态的Object[]数组。二者最大的区别在于：**Vector类的所有方法都是同步方法，而ArrayList类的方法都是非同步方法**
- 第二个区别在于：Vector类的方法要比ArrayList类的方法多，在某些情况下使用起来比ArrayList类方便
- 创建了一个Vector对象后，可以往其中随意地插入不同的类的对象，既不需顾及类型也不需预先选定向量的容量，并可方便地进行查找。对于预先不知或不愿预先定义数组大小，并需频繁进行查找、插入和删除工作的情况，可以考虑使用Vector
- Vector 常用于保存从数据库中读取的记录



Vector使用示例:VectorTest.java

```
Vector v1=new Vector();
Integer integer1=new Integer(1);
v1.addElement("one");//加入的为字符串对象
v1.addElement(integer1);
v1.addElement(integer1);//加入的为Integer的对象
v1.addElement("two");
v1.addElement(new Integer(2));
v1.addElement(integer1);
v1.addElement(integer1);
System.out.println("The vector v1 is:\n\t"+v1);//将v1转换成字符串并打印
v1.insertElementAt("three",2);//往指定位置插入新的对象，指定位置后的对象依次往后顺延
v1.insertElementAt(new Float(3.9),3);

v1.setElementAt("four",2);//将指定位置的对象设置为新的对象

v1.removeElement(integer1);//从向量对象v1中删除对象integer1,由于存在多个integer1所以从头开始找
, 删除找到的第一个integer
Enumeration enum1=v1.elements();//使用枚举类(Enumeration)的方法来获取向量对象的每个元素
System.out.print("The vector v1(used method removeElement())is:");
while(enum1.hasMoreElements())    System.out.print(enum1.nextElement()+" ");

//重新设置v1的大小，多余的元素被行弃
v1.setSize(4);
System.out.println("The new vector(resized the vector)is:"+v1);
} }
```



Vector使用示例

- 程序代码详见VectorTest.java，程序输出结果如下：

The vector v1 is:

[one, 1, 1, two, 2, 1, 1]

The vector v1(used method insertElementAt())is:

[one, 1, three, 3.9, 1, two, 2, 1, 1]

The vector v1(used method setElementAt())is:

[one, 1, four, 3.9, 1, two, 2, 1, 1]

The vector v1(used method removeElement())is:one four 3.9 1 two 2 1 1

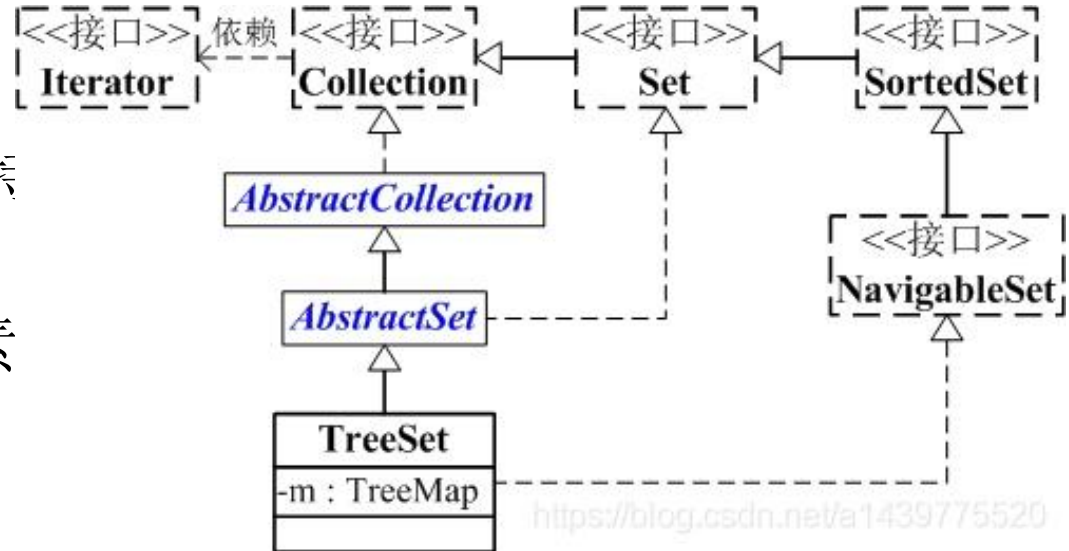
The position of object 1(top-to-bottom):3

The position of object 1(tottom-to-top):7

The new vector(resized the vector)is:[one, four, 3.9, 1]

Set

- Set接口是Collection接口的子接口
- Set类型的集合具有以下特点：
 - ✓ 不允许包含相同的元素
 - ✓ 至多有一个null元素



java.util.AbstractSet<E> (implements java.util.Set<E>)

java.util.EnumSet<E> (implements java.lang.Cloneable, java.io.Serializable)

java.util.HashSet<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)

java.util.LinkedHashSet<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)

java.util.TreeSet<E> (implements java.lang.Cloneable, java.util.NavigableSet<E>, java.io.Serializable)



TreeSet

- `public class TreeSet<E> extends AbstractSet<E>`
implements `NavigableSet<E>`, `Cloneable`, `Serializable`
- **TreeSet** 是一个有序的集合，它的作用是提供有序的**Set**集合。它继承于**AbstractSet**抽象类，实现了 `NavigableSet<E>`, `Cloneable`, `java.io.Serializable` 接口。
- **TreeSet** 的本质是一个"有序的，并且没有重复元素"的集合，它是通过**TreeMap**实现的。**TreeSet** 中含有一个 "NavigableMap 类型的成员变量" `m`，而 `m` 实际上是 "TreeMap 的实例"。



TreeSetIteratorTest.java

```
public static void main(String[] args) {  
    TreeSet set = new TreeSet();  
    set.add("one");  
    set.add("one");  
    set.add("two");  
    set.add("three");  
    set.add("four");  
    set.add("five");  
    set.add("five");  
    for (Iterator iter = set.iterator(); iter.hasNext();) {  
        System.out.printf("for each : %s\n", iter.next());  
    }  
}
```

---- For-each ----
for each : five
for each : four
for each : one
for each : three
for each : two

映象 (Map)

- Map（映象）用于存放一组“关键字—值”的数据对。即Map用于保存具有映射关系的数据，Map里保存着两组数据：**key**和**value**，它们都可以使任何引用类型的数据，但**key**不能重复。所以通过指定的**key**就可以取出对应的**value**。
- Java库提供了用于映象的两个通用实现方法，即散列映象（HashMap）和树状映象（TreeMap）。HashMap的运算速度比较快，如果你不需要按照排列顺序来访问关键字，那么最好选择HashMap。

HashMap

- 下面是建立散列映象以便对员工进行排序的代码：
- `HashMap staff=new HashMap();`
 - `Employee harry=new Employee("Hacker");`
 - `staff.put("987-98-9996", harry);`
 - 每当将一个对象添加到HashMap中时，必须提供一个关键字。在上面的代码中，关键字是个字符串，对应的值是个Employee对象
 - 若要检索一个对象，必须使用该关键字：
 - `String s="987-98-9996";`
 - `e=(Employee)staff.get(s);`//得到harry
 - 如果HashMap中没有存放某个关键字的信息，那么get方法将返回null



Map主要方法

`void clear()`

`boolean containsKey(Object key)`

查询Map中是否包含指定key, 如果包含则返回true

`boolean containsValue(Object value)`

查询Map中是否包含指定value, 如果包含则返回true

`Set entrySet()`

返回Map中所包含的键值对所组成的Set集合, 每个集合元素都是Map.Entry对象(Entry是Map的内部类)。

`Object get(Object key)`

返回指定key所对应的value, 如Map中不包含key则返回null

`boolean isEmpty()`

查询Map是否为空, 如果空则返回true

`Set keySet()`

返回该Map中所有key所组成的set集合

`Object put(Object key, Object value)`

添加一个键值对, 如果已有一个相同的key值则新的键值对覆盖旧的键值对

`void putAll(Map m)`

将指定Map中的键值对复制到Map中。

`Object remove(Object key)`

删除指定key所对应的键值对, 返回可以所关联的value, 如果key不存在, 返回null

`int size()`

返回该Map里的键值对的个数。

`Collection values()`

返回该Map里所有value组成的Collection

内部类Entry

`Object getKey()` ○ 返回该Entry里包含的key值

`Object getValue()` ○ 返回该Entry里包含的value值

`Object setValue(V value)` ○ 设置该Entry里包含的value值, 并返回新设置的value值

<http://www.bilibili.com/video/av10681849305>



HashMap

- 关键字必须是独一无二的。你不能为同一个关键字存放两个值。如果你使用同一个关键字两次调用put方法，那么它所产生的第二个值便取代第一个值。——实际上，put方法返回的是关键字参数存放的上一个值，该特性很有用：如果put方法返回了一个非null值，那么你就知道原来的值被取代了
- remove方法用于从HashMap中删除元素，size方法用于返回HashMap中的元素数量

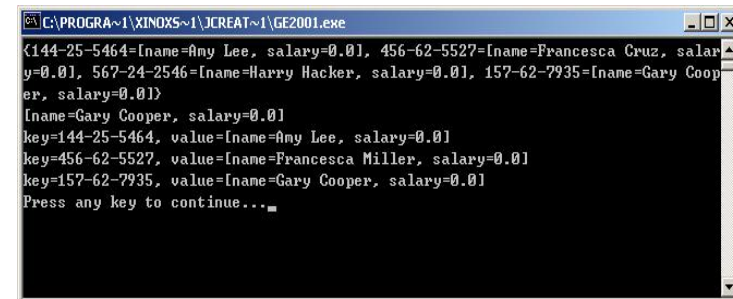
HashMap使用示例

```
public static void main(String[] args) {
    Map staff = new HashMap();
    staff.put("144-25-5464", new Employee("Amy Lee"));
    staff.put("567-24-2546", new Employee("Harry Hacker"));
    staff.put("157-62-7935", new Employee("Gary Cooper"));
    staff.put("456-62-5527", new Employee("Francesca Cruz"));
    // print all entries打印所有的条目
    System.out.println(staff);
    // remove an entry删除一个条目
    staff.remove("567-24-2546");
    // replace an entry替换一个条目
    staff.put("456-62-5527", new Employee("Francesca Miller"));
```

Set entries = staff.entrySet();// 返回映象中的条目集，即关键字/值对

```
Iterator iter = entries.iterator();
while (iter.hasNext()) {
    Map.Entry entry = (Map.Entry) iter.next();// 条目集的元素
    // 是内部类Map.Entry的对象
    Object key = entry.getKey();// 返回该条目的关键字
    Object value = entry.getValue();// 返回该条目的值
    System.out.println("key=" + key + ", value=" + value);
}
```

➤ 程序代码详见
MapTest.java，程序输出
结果如下：



TreeMap:



<http://t.cn/R6bbsgncdm/fei/yqqa34b4980u>

Collections类

- Collections类具有许多静态的实用方法对集合进行操作，可根据需要选用
- 例如，Collections类的sort方法可用于为实现List接口的集合进行排序：
 - ✓ List staff=new LinkedList();
 - ✓
 - ✓ Collections.sort(staff, new Comparator(){
 - public int compare(Object a, Object b){//自定义比较器
 - double salaryDifference=(Employee)a. getSalary() - (Employee)b. getSalary() ;
 - if (salaryDifference < 0) return -1;
 - if (salaryDifference > 0) return 1;
 - return 0
 - }
 - ✓ }



Enumeration 接口

- Enumeration中封装了有关枚举数据集合的方法
- 在Enumeration中只有2个方法：
 - ✓ hasMoreElements()//判断集合中是否还有其它元素
 - ✓ nextElement()//获取集合中下一个元素
 - ✓ 利用这两个方法可以依次获得集合中的元素
- Vector中提供的方法：
public final synchronized Enumeration elements()

此方法将向量对象对应到一个枚举类型。`java.util`包中的其它类中也大都有这类方法，以便于用户获取对应的枚举类型。

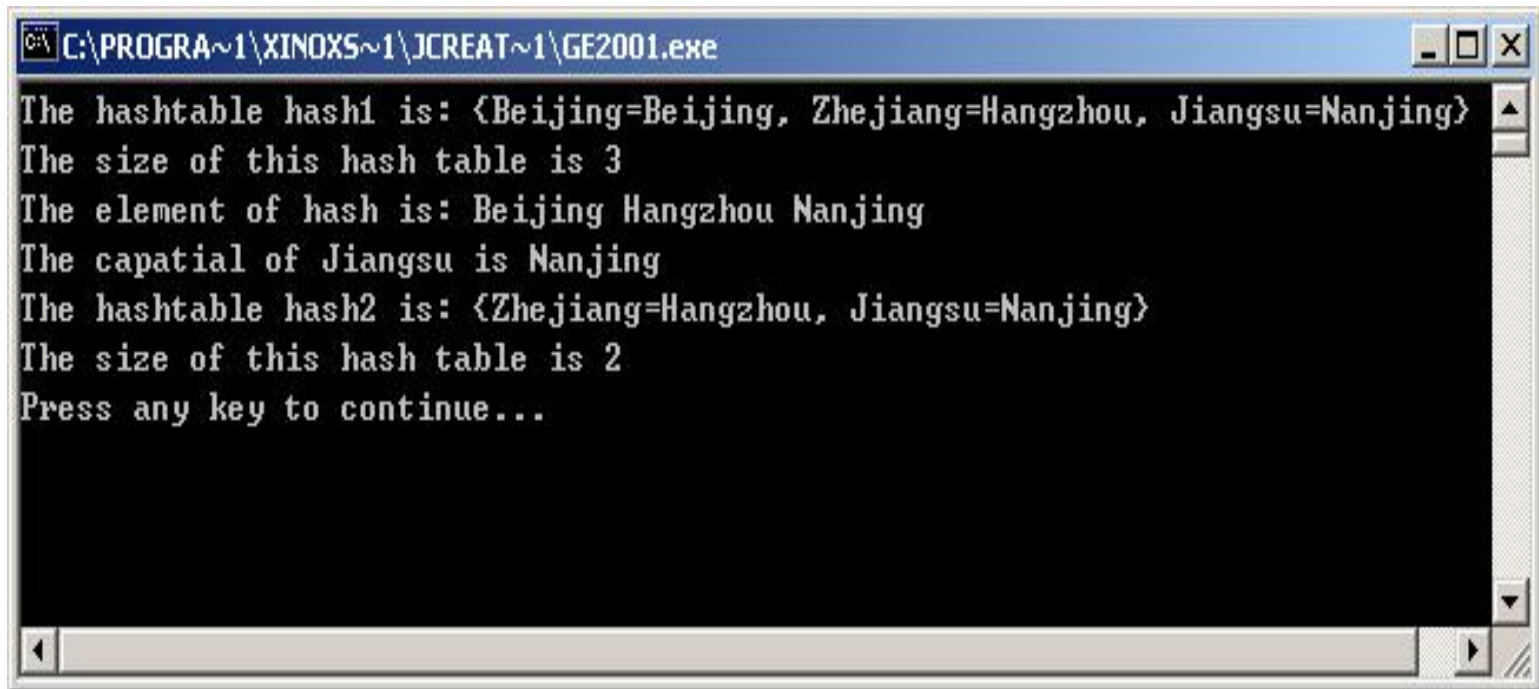


Hashtable

- 哈希表类**Hashtable**是一个“旧的”集合类。它的作用与**HashMap**是相同的，它们拥有相同的接口。
- 二者之间最大的区别是：**Hashtable**类的各个方法是同步的，而**HashMap**类的方法是非同步的。因此**HashMap**比**Hashtable**速度快，效率高

Hashtable使用示例

- 程序代码详见HashTableTest.java，程序输出结果如下：



```
C:\PROGRA~1\XINOS~1\JCREAT~1\GE2001.exe
The hashtable hash1 is: {Beijing=Beijing, Zhejiang=Hangzhou, Jiangsu=Nanjing}
The size of this hash table is 3
The element of hash is: Beijing Hangzhou Nanjing
The capatial of Jiangsu is Nanjing
The hashtable hash2 is: {Zhejiang=Hangzhou, Jiangsu=Nanjing}
The size of this hash table is 2
Press any key to continue...
```




Properties 4-1

- 属性集类Properties是Hashtable的子类，是个类型非常特殊的映象结构。它有以下3个特殊性：
 - ✓ 关键字和值都是字符串
 - ✓ 属性表可以保存到一个文件，也可以从一个文件那里加载
 - ✓ 有一个默认辅助属性表
- 属性集常常用于为程序设定配置选项



Properties 4-2

- 在DOS窗口中，我们常使用如下命令来设置环境变量：
 - ✓ `set temp=c:\windows\temp`
 - ✓ `set CLASSPATH=c:\jdk\lib`
 - ✓ 可以将这些设置变成属性集：
 - ✓ `Properties settings=new Properties();`
 - ✓ `settings.put("temp", "c:\windows\temp");`
 - ✓ `settings.put("CLASSPATH", "c:\jdk\lib");`
 - ✓ 把该属性列表保存到一个文件中：
 - ✓ `settings.store(new FileOutputStream("c:\myauto.ini"), "环境变量配置")`//第二个参数是要在文件中包含的注释
 - ✓ 从文件中读入属性列表：
 - ✓ `FileInputStream sf=new FileInputStream("c:\myauto.ini");`
 - ✓ `settings.load(sf);`



Properties 4-3

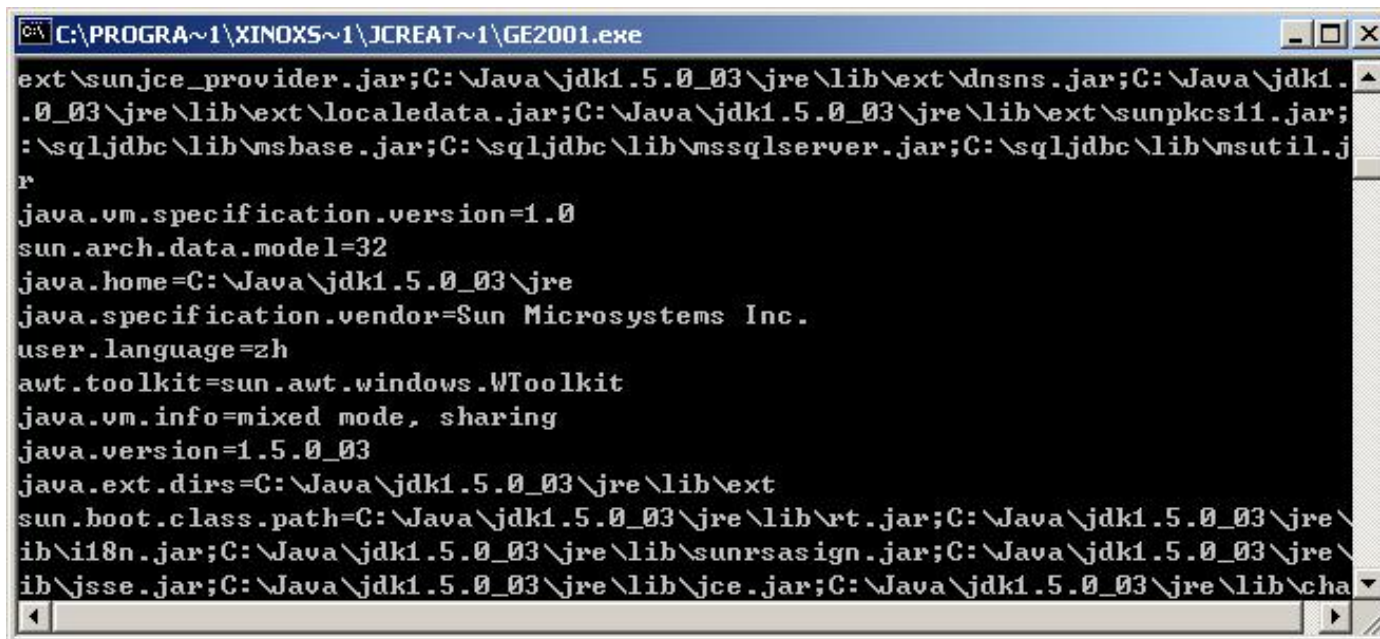
- 每当你想让用户定制一种应用程序时，属性集是个非常有用的工具
- 假设我们允许用户在配置文件 `CustomWorld.ini` 中设定下面的条件：窗口的大小、字体、字体的磅值、背景颜色、消息字符串，如果用户没有指定其中的某些值，我们将为他们提供一个默认值。 **Properties** 类提供2种方法可以用来提供默认值

Properties 4-4

- 第一种方法：每当你查看一个字符串的值时就设定一个默认值：
 - ✓ `String font = settings. getProperty("FONT","Courier");`
 - ✓ 如果在属性表中有一个("FONT"属性，那么font将被设置为该字符串；否则， font将被设置为Courier
- 第二种方法：将所有的默认值包装在一个辅助属性集中，然后把该属性集作为参数传递给你要查看的属性集的构造函数：
 - ✓ `Properties defaultSettings=new Properties();`
 - ✓ `defaultSettings. put("FONT","Courier");`
 - ✓ `defaultSettings. put("SIZE","10");`
 - ✓ `defaultSettings. put("MESSAGE","hello,world");`
 - ✓ `.....`
 - ✓ `Properties settings=new Properties(defaultSettings);`
 - ✓ `FileInputStream sf=new FileInputStream("c:\myauto.ini");`
 - ✓ `settings. load(sf);`

显示系统属性信息

- 在你的电脑上，有关java的系统信息将被存放在一个由System类的getProperties方法返回的Properties对象中。程序代码详见SystemInfo.java，程序输出结果如下：



```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
ext\sunjce_provider.jar;C:\Java\jdk1.5.0_03\jre\lib\ext\dnssns.jar;C:\Java\jdk1.
.0_03\jre\lib\ext\localedata.jar;C:\Java\jdk1.5.0_03\jre\lib\ext\sunpkcs11.jar;
:\sqljdbc\lib\msbase.jar;C:\sqljdbc\lib\mssqlserver.jar;C:\sqljdbc\lib\msutil.j
r
java.vm.specification.version=1.0
sun.arch.data.model=32
java.home=C:\Java\jdk1.5.0_03\jre
java.specification.vendor=Sun Microsystems Inc.
user.language=zh
awt.toolkit=sun.awt.windows.WToolkit
java.vm.info=mixed mode, sharing
java.version=1.5.0_03
java.ext.dirs=C:\Java\jdk1.5.0_03\jre\lib\ext
sun.boot.class.path=C:\Java\jdk1.5.0_03\jre\lib\rt.jar;C:\Java\jdk1.5.0_03\jre\
ib\i18n.jar;C:\Java\jdk1.5.0_03\jre\lib\sunrsasign.jar;C:\Java\jdk1.5.0_03\jre\
ib\jsse.jar;C:\Java\jdk1.5.0_03\jre\lib\jce.jar;C:\Java\jdk1.5.0_03\jre\lib\cha
```

用户定制应用程序示例

- 程序代码详见CustomWorld.java，程序输出结果如下：



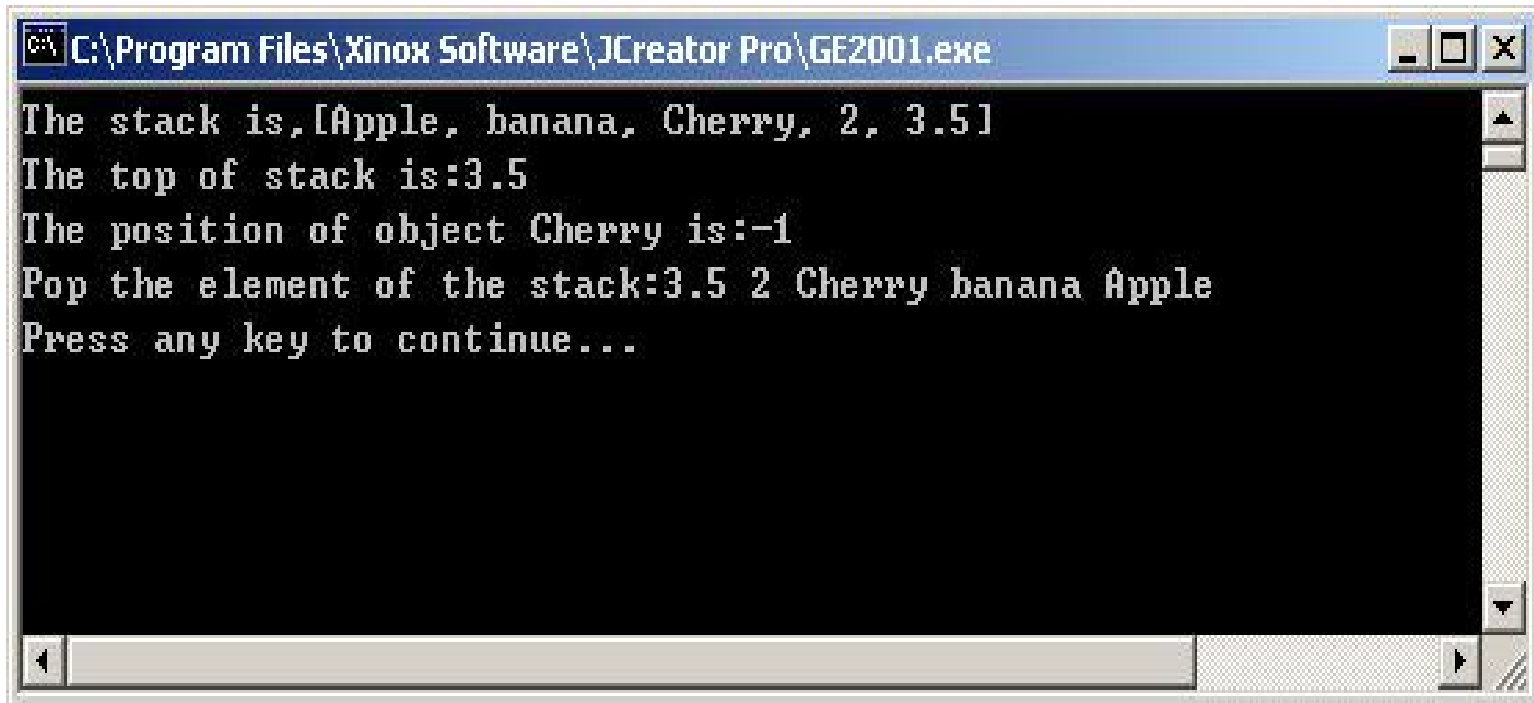


Stack

- **Stack**类是**Vector**类的子类。它向用户提供了堆栈这种高级的数据结构。栈的基本特性就是先进后出，即先放入栈中的元素将后被推出。**Stack**类中提供了相应方法完成栈的有关操作。

Stack使用示例

- 程序代码详见StackTest.java，程序输出结果如下：



```
The stack is,[Apple, banana, Cherry, 2, 3.5]
The top of stack is:3.5
The position of object Cherry is:-1
Pop the element of the stack:3.5 2 Cherry banana Apple
Press any key to continue...
```




总结

List是有序的集合，**Set**是无序的集合。**Map**是无序的键值对

◦List、Set都是继承自Collection接口，Map则不是

List: (1)ArrayList (2) LinkedList (3) Vector

Set: (1)HashSet (2) LinkedHashSet (3) TreeSet

- ◆ **List**特点：元素有放入顺序，元素可重复
- ◆ **Set**特点：元素无放入顺序，元素不可重复，重复元素会覆盖掉（元素在set中的位置是有该元素的HashCode决定的）
- ◆ **TreeSet**的本质是一个"有序的，并且没有重复元素"的集合，它是通过**TreeMap**实现的。

***Vector**是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用。

	HashSet	HashMap	HashTable	TreeSet	ArrayList	Vector	EnumSet
是否有序	无序	无序	无序	有序	有序	有序	无序
存储结构	集合	键值对	键值对	集合	集合	集合	集合
线程安全	是	否	是	是	是	否	是