

Java SE 程序设计 北京圣思园科技有限公司

主讲人 张龙

All Rights Reserved



Java Collection

- 课程目标
 - 理解Java 集合框架
 - 熟练使用**java.util**包中的相关类与接口进行编程开发
 - 改善程序性能，今后进行程序编写时将大量使用集合类与相关接口



类集框架

- java.util包中包含了一些在Java 2中新增加的最令人兴奋的增强功能：类集。一个类集（collection）是一组对象。类集的增加使得许多java.util中的成员在结构和体系结构上发生根本的改变。它也扩展了包可以被应用的任务范围。类集是被所有Java程序员紧密关注的最新型的技术



类集框架

- 除了类集，`java.util`还包含了支持范围广泛的函数的各种各样的类和接口。这些类和接口被核心的Java包广泛使用，同时当然也可以被你编写的程序所使用。对它们的应用包括产生伪随机数，对日期和时间的操作，观测事件，对位集的操作以及标记字符串。由于`java.util`具有许多特性，因此它是Java中最被广泛使用的一个包



类集概述

- Java的类集（Collection）框架使你的程序处理对象组的方法标准化。在Java 2出现之前，Java提供了一些专门的类如Dictionary，Vector，Stack和Properties去存储和操作对象组。尽管这些类非常有用，它们却缺少一个集中，统一的主题。因此例如说使用Vector的方法就会与使用Properties的方法不同。以前的专门的方法也没有被设计成易于扩展和能适应新的环境的形式。而类集解决了这些（以及其他的一些）问题



类集概述

- 类集框架被设计用于适应几个目的。首先，这种框架是高性能的。对基本类集（动态数组，链接表，树和散列表）的实现是高效率的。一般很少需要人工去对这些“数据引擎”编写代码（如果有的话）。第二点，框架必须允许不同类型的类集以相同的方式和高度互操作方式工作。第三点，类集必须是容易扩展和/或修改的。为了实现这一目标，类集框架被设计成包含一组标准的接口。对这些接口，提供了几个标准的实现工具（例如LinkedList，HashSet和TreeSet），通常就是这样使用的。如果你愿意的话，也可以实现你自己的类集。为了方便起见，创建用于各种特殊目的的实现工具。一部分工具可以使你自己的类集实现更加容易。最后，增加了允许将标准数组融合到类集框架中的机制



类集概述

- 算法（Algorithms）是类集机制的另一个重要部分。算法操作类集，它在Collections类中被定义为静态方法。因此它们可以被所有的类集所利用。每一个类集类不必实现它自己的方案，算法提供了一个处理类集的标准方法



类集概述

- 由类集框架创建的另一项是Iterator接口。一个迭代程序（iterator）提供了一个多用途的，标准化的方法，用于每次访问类集的一个元素。因此迭代程序提供了一种枚举类集内容（enumerating the contents of a collection）的方法。因为每一个类集都实现Iterator，所以通过由Iterator定义的方法，任一类集类的元素都能被访问到。



类集概述

- 除了类集之外，框架定义了几个映射接口和类。映射（Maps）存储键/值对。尽管映射在对项的正确使用上不是“类集”，但它们完全用类集集成。在类集框架的语言中，可以获得映射的类集“视图”（collection-view）。这个“视图”包含了从存储在类集中的映射得到的元素。因此，如果选择了一个映射，就可以将其当做一个类集来处理



类集概述

- 对于由java.util定义的原始类，类集机制被更新以便它们也能够集成到新的系统里。所以理解下面的说法是很重要的：尽管类集的增加改变了许多原始工具类的结构，但它却不会导致被抛弃。类集仅仅是提供了处理事情的一个更好的方法

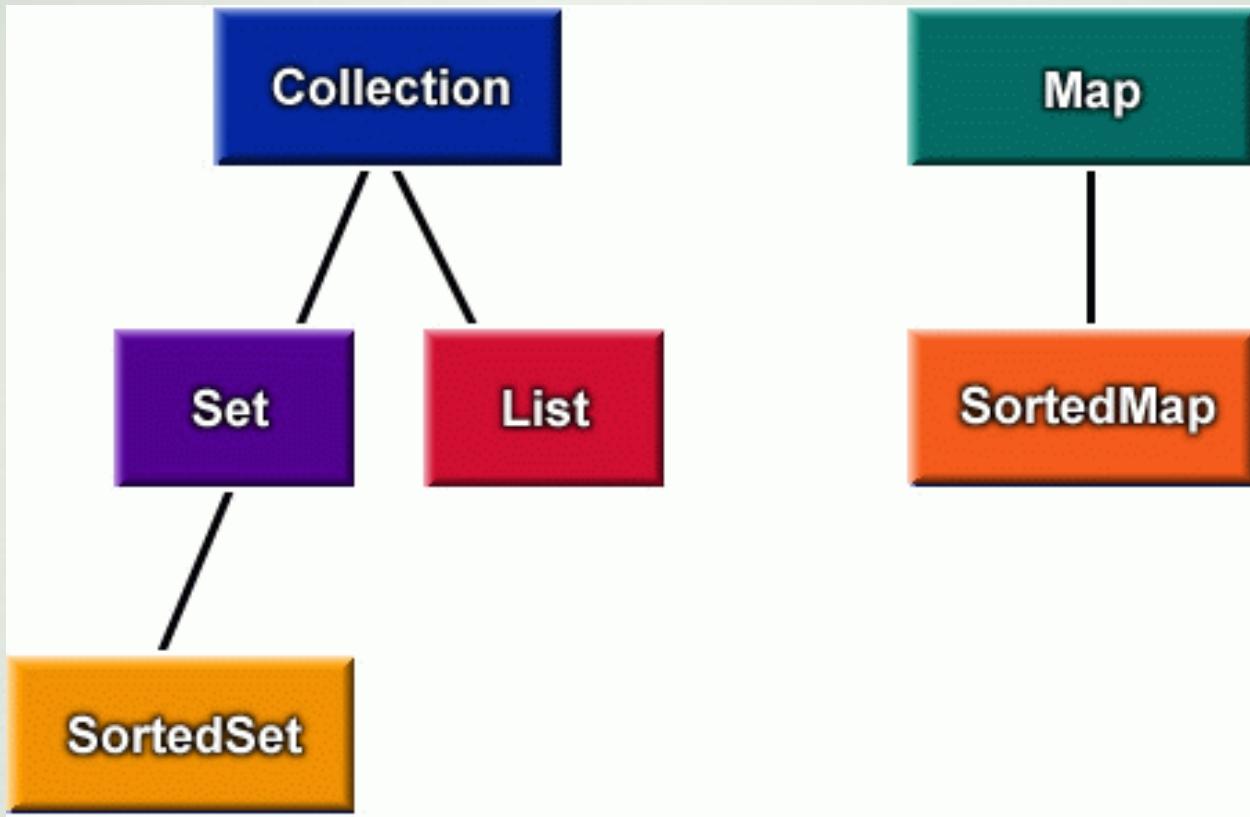


类集概述

- 最后的一点：如果你对C++比较熟悉的话，那么你可以发现Java的类集技术与在C++中定义的标准模板库（STL）相似。
- 在C++中叫做容器（container），而在Java中叫做类集



集合框架中的接口



所谓框架就是一个类库的集合。集合框架就是一个用来表示和操作集合的统一的架构，包含了实现集合的接口与类。



集合框架中的接口

- 除了类集接口之外，类集也使用Comparator, Iterator和ListIterator接口。
- 简单地说，Comparator接口定义了两个对象如何比较；Iterator和ListIterator接口枚举类集中的对象。



集合框架中的接口

- 为了在它们的使用中提供最大的灵活性，类集接口允许对一些方法进行选择。可选择的方法使得使用者可以更改类集的内容。支持这些方法的类集被称为可修改的（modifiable）。不允许修改其内容的类集被称为不可修改的（unmodifiable）。如果对一个不可修改的类集使用这些方法，将引发一个UnsupportedOperationException异常。所有内置的类集都是可修改的。



集合框架中的接口

- Collection接口是构造类集框架的基础。它声明所有类集都将拥有的核心方法。因为所有类集实现Collection，所以熟悉它的方法对于清楚地理解框架是必要的。其中几种方法可能会引发一个UnsupportedOperationException异常。正如上面解释的那样，这些发生在当类集不能被修改时。当一个对象与另一个对象不兼容，例如当企图增加一个不兼容的对象到一个类集中时。将产生一个ClassCastException异常



集合框架中的接口

- 调用add()方法可以将对象加入类集。注意add()带一个Object类型的参数。因为Object是所有类的超类，所以任何类型的对象可以被存储在一个类集中。然而原始类型不行。例如，一个类集不能直接存储类型int, char, double等的值。当然如果想存储这些对象，也可以使用原始类型包装器。可以通过调用addAll()方法将一个类集的全部内容增加到另一个类集中。



集合框架中的接口

- 可以通过调用remove()方法将一个对象删除。为了删除一组对象，可以调用removeAll()方法。调用retainAll()方法可以将除了一组指定的元素之外的所有元素删除。为了清空类集，可以调用clear()方法



集合框架中的接口

- 通过调用contains()方法，可以确定一个类集是否包含了一个指定的对象。
- 为了确定一个类集是否包含了另一个类集的全部元素，可以调用containsAll()方法
- 当一个类集是空的时候，可以通过调用isEmpty()方法来予以确认。
- 调用size()方法可以获得类集中当前元素的个数



集合框架中的接口

- `toArray()`方法返回一个数组，这个数组包含了存储在调用类集中的元素。通过在类集和数组之间提供一条路径，可以充分利用这两者的优点



集合框架中的接口

- 一个更加重要的方法是iterator()，该方法对类集返回一个迭代程序。当使用一个类集框架时，迭代程序对于成功的编程来说是至关重要的

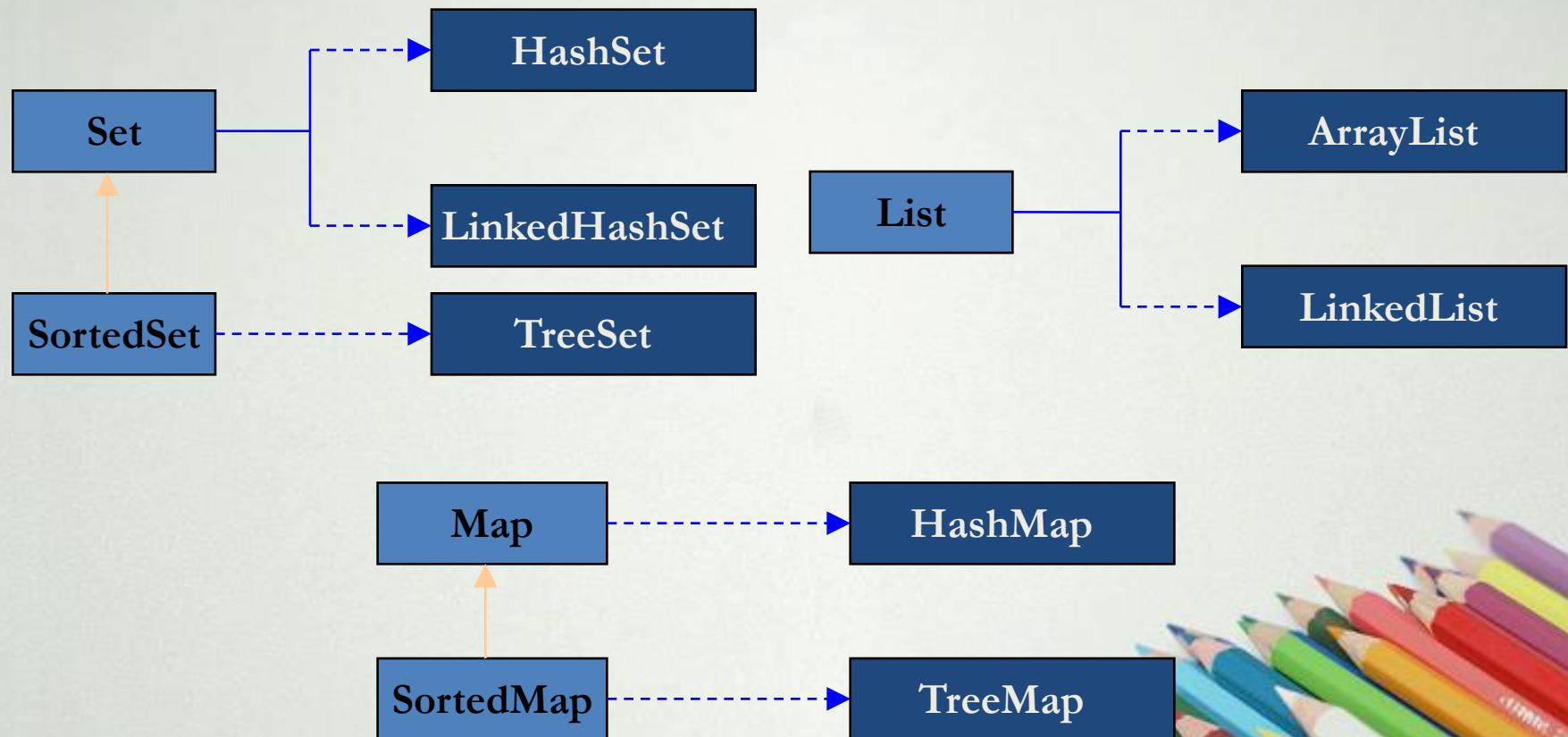


集合框架中的接口

- **Collection:** 集合层次中的根接口，JDK没有提供这个接口直接的实现类。
- **Set:** 不能包含重复的元素。SortedSet是一个按照升序排列元素的Set。
- **List:** 是一个有序的集合，可以包含重复的元素。提供了按索引访问的方式。
- **Map:** 包含了key-value对。Map不能包含重复的key。SortedMap是一个按照升序排列key的Map。



集合框架中的实现类



List接口

- List接口扩展了Collection并声明存储一系列元素的类集的特性。使用一个基于零的下标，元素可以通过它们在列表中的位置被插入和访问。一个列表可以包含重复元素



List接口

- 除了由Collection定义的方法之外，List还定义了一些它自己的方法。再次注意当类集不能被修改时，其中的几种方法引发UnsupportedOperation Exception异常。当一个对象与另一个不兼容，例如当企图将一个不兼容的对象加入一个类集中时，将产生ClassCastException异常



List接口

- 对于由Collection定义的add()和addAll()方法，List增加了方法add(int, Object)和addAll(int, Collection)。这些方法在指定的下标处插入元素。由Collection定义的add(Object)和addAll(Collection)的语义也被List改变了，以便它们在列表的尾部增加元素



List接口

- 为了获得在指定位置存储的对象，可以用对象的下标调用get()方法。为了给类表中的一个元素赋值，可以调用set()方法，指定被改变的对象的下标。调用indexOf()或lastIndexOf()可以得到一个对象的下标
- 通过调用subList()方法，可以获得列表的一个指定了开始下标和结束下标的子列表。subList()方法使得列表处理十分方便。



Set接口

- 集合接口定义了一个集合。它扩展了Collection并说明了不允许复制元素的类集的特性。因此，如果试图将复制元素加到集合中时，add()方法将返回false。它本身并没有定义任何附加的方法



SortedSet接口

- SortedSet接口扩展了Set并说明了按升序排列的集合的特性。当没有项包含在调用集合中时，其中的几种方法引发NoSuchElementException异常。当对象与调用集合中的元素不兼容时，引发ClassCastException异常。如果试图使用null对象，而集合不允许null时，引发NullPointerException异常



SortedSet接口

- SortedSet定义了几种方法，使得对集合的处理更加方便。调用first()方法，可以获得集合中的第一个对象。调用last()方法，可以获得集合中的最后一个元素。调用subSet()方法，可以获得排序集合的一个指定了第一个和最后一个对象的子集合。如果需要得到从集合的第一个元素开始的一个子集合，可以使用headSet()方法。如果需要获得集合尾部的一个子集合，可以使用tailSet()方法。



Collection类

- 现在，你已经熟悉了类集接口，下面开始讨论实现它们的标准类。一些类提供了完整的可以被使用的工具。另一些类是抽象的，提供主框架工具，作为创建具体类集的起始点



ArrayList

- **ArrayList**: 我们可以将其看作是能够自动增长容量的数组。
- 利用**ArrayList**的**toArray()**返回一个数组。
- **Arrays.asList()**返回一个列表。
- **迭代器(Iterator)** 给我们提供了一种通用的方式来访问集合中的元素。



ArrayList

- ArrayList类扩展AbstractList并执行List接口。
ArrayList支持可随需要而增长的动态数组。在Java中，标准数组是定长的。在数组创建之后，它们不能被加长或缩短，这也就意味着你必须事先知道数组可以容纳多少元素。但是，你直到运行时才能知道需要多大的数组。为了解决这个问题，类集框架定义了ArrayList。本质上，ArrayList是对象引用的一个变长数组。也就是说，ArrayList能够动态地增加或减小其大小。数组列表以一个原始大小被创建。当超过了它的大小，类集自动增大。当对象被删除后，数组就可以缩小。



ArrayList

- ArrayList有如下的构造函数
 - ArrayList()
 - ArrayList(Collection c)
 - ArrayList(int capacity)
 - 第一个构造函数建立一个空的数组列表。
 - 第二个构造函数建立一个数组列表，该数组列表由类集c中的元素初始化。
 - 第三个构造函数建立一个数组列表，该数组有指定的初始容量（capacity）。容量是用于存储元素的基本数组的大小。当元素被追加到数组列表上时，容量会自动增加



ArrayList

- 参见程序 `ArrayList1.java`
 - 使用由`toString()`方法提供的默认的转换显示类集的内容，`toString()`方法是从`AbstractCollection`继承下来的。尽管它对简短的例子程序来说是足够了，然而很少使用这种方法去显示实际中的类集的内容。通常编程者会提供自己的输出程序。但在下面的几个例子中，仍将采用由`toString()`方法创建的默认输出



ArrayList

- 尽管当对象被存储在ArrayList对象中时，其容量会自动增加。仍可以通过调用ensureCapacity()方法来人工地增加ArrayList的容量。如果事先知道将在当前能够容纳的类集中存储许许多多的项时，你可能会想这样做。在开始时，通过一次性地增加它的容量，就能避免后面的再分配。因为再分配是很花时间的，避免不必要的处理可以改善性能



ArrayList

- ensureCapacity()方法如下所示
 - void ensureCapacity(int cap)
 - 这里， cap是新的容量
- 相反地， 如果想要减小在ArrayList对象之下的数组的大小， 以便它有正好容纳当前项的大小， 可以调用trimToSize()方法。 该方法如下
 - void trimToSize()



从数组列表（ArrayList）获得数组（Array）

- 当使用ArrayList时，有时想要获得一个实际的数组，这个数组包含了列表的内容。可以通过调用方法toArray()来实现它。下面是几个为什么可能想将类集转换成为数组的原因
 - 对于特定的操作，可以获得更快的处理时间
 - 为了给方法传递数组，而方法不必重载去接收类集
 - 为了将新的基于类集的程序与不认识类集的老程序集成



从数组列表 (ArrayList) 获得数组 (Array)

- 参见程序 `ArrayList2.java`
- 参见程序 `ArrayList3.java`
 - `Arrays.asList()`
 - 返回一个受指定数组支持的固定大小的列表。（对返回列表的更改会“直写”到数组。）此方法同 `Collection.toArray` 一起，充当了基于数组的 API 与基于 `collection` 的 API 之间的桥梁



LinkedList类

- LinkedList类扩展AbstractSequentialList并执行List接口。它提供了一个链接列表数据结构。它具有如下的两个构造函数，说明如下
 - LinkedList()
 - LinkedList(Collection c)
 - 第一个构造函数建立一个空的链接列表。
 - 第二个构造函数建立一个链接列表，该链接列表由类集c中的元素初始化



LinkedList类

- 除了它继承的方法之外， LinkedList类本身还定义了一些有用的方法，这些方法主要用于操作和访问列表。使用addFirst()方法可以在列表头增加元素；使用addLast()方法可以在列表的尾部增加元素。它们的形式如下所示
 - `void addFirst(Object obj)`
 - `void addLast(Object obj)`
 - 这里， `obj`是被增加的项



LinkedList类

- 调用getFirst()方法可以获得第一个元素。调用getLast()方法可以得到最后一个元素。它们的形式如下所示：
 - Object getFirst()
 - Object getLast()
- 为了删除第一个元素，可以使用removeFirst()方法；为了删除最后一个元素，可以调用removeLast()方法。它们的形式如下所示
 - Object removeFirst()
 - Object removeLast()



LinkedList类

- 参见程序 **LinkedList1.java**

- 因为LinkedList实现List接口，调用add(Object)将项目追加到列表的尾部，如同addLast()方法所做的那样。使用add()方法的add(int, Object)形式，插入项目到指定的位置，如例子程序中调用add (1, “A2”) 的举例
 - 注意如何通过调用get()和set()方法而使得LinkedList中的第三个元素发生了改变。为了获得一个元素的当前值，通过get()方法传递存储该元素的下标值。为了对这个下标位置赋一个新值，通过set()方法传递下标和对应的新值



LinkedList类

- **LinkedList**是采用双向循环链表实现的。
- 利用**LinkedList**实现栈(stack)、队列(queue)、双向队列(double-ended queue)。



数据结构

- 一般将数据结构分为两大类：线性数据结构和非线性数据结构。**线性数据结构**有线性表、栈、队列、串、数组和文件；**非线性数据结构**有树和图。



线性表

- 线性表的逻辑结构是n个数据元素的有限序列：
$$(a_1, a_2, a_3, \dots, a_n)$$

n为线性表的长度($n \geq 0$)， $n=0$ 的表称为空表。
- 数据元素呈线性关系。必存在唯一的称为“第一个”的数据元素；必存在唯一的称为“最后一个”的数据元素；除第一个元素外，每个元素都有且只有一个前驱元素；除最后一个元素外，每个元素都有且只有一个后继元素。
- 所有数据元素在同一个线性表中必须是相同的数据类型。



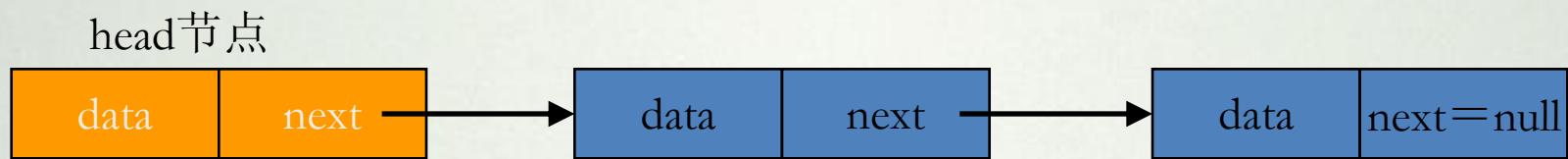
线性表

- 线性表按其存储结构可分为顺序表和链表。用顺序存储结构存储的线性表称为**顺序表**；用链式存储结构存储的线性表称为**链表**。
- 将线性表中的数据元素依次存放在某个存储区域中，所形成的表称为**顺序表**。**一维数组**就是用顺序方式存储的线性表。



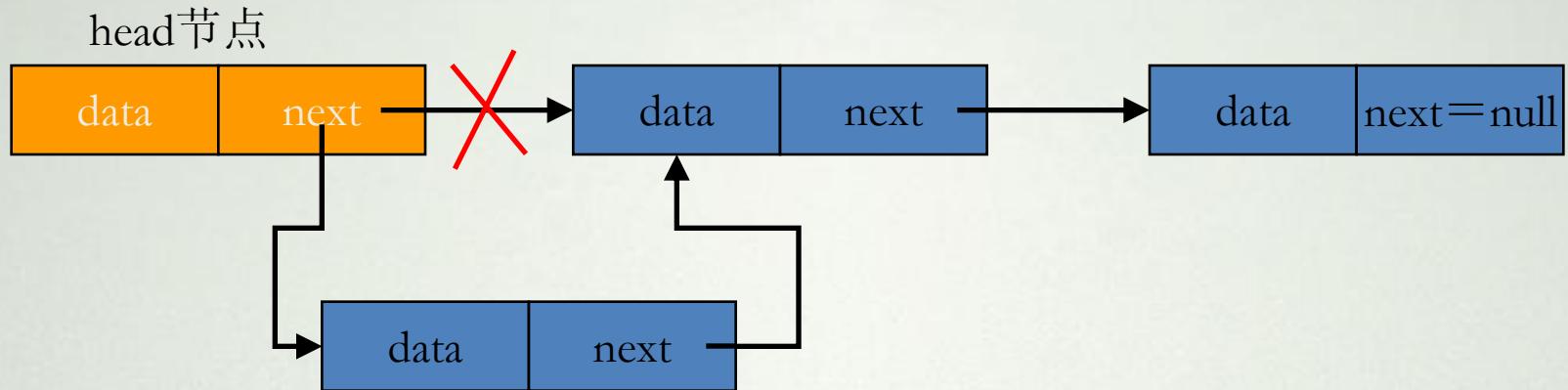
链表

- 单向链表

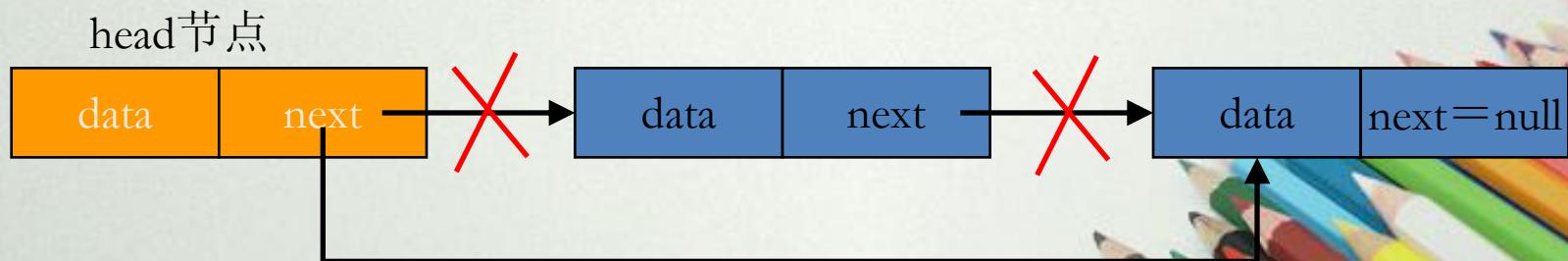


链表

插入

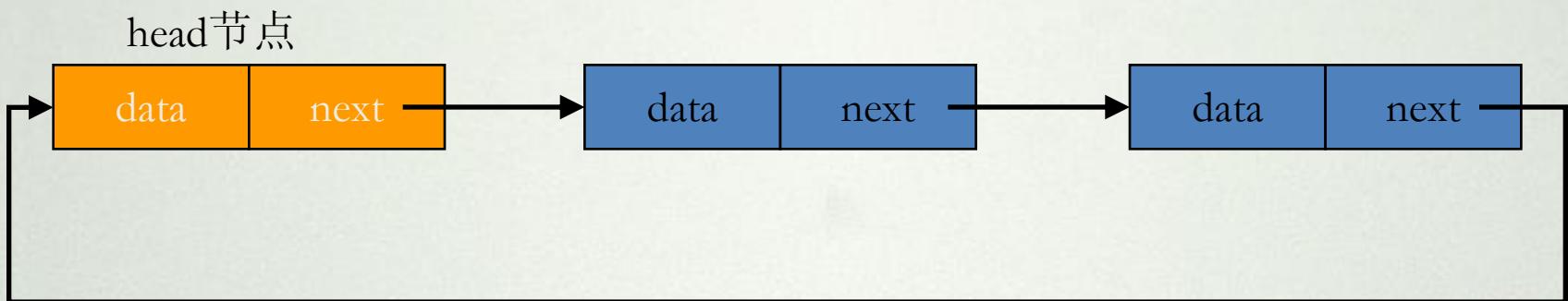


删除



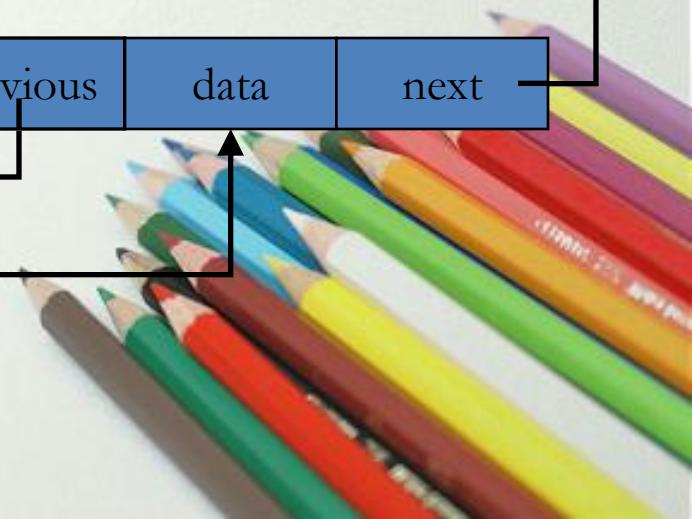
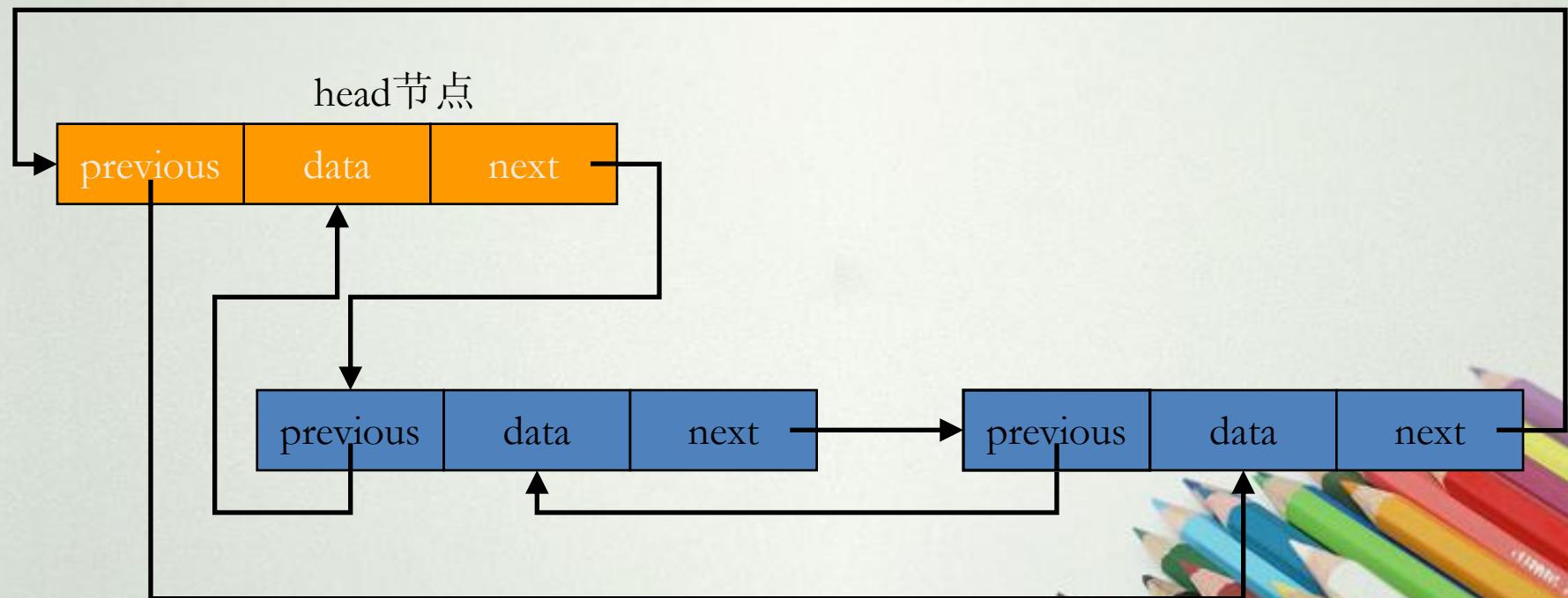
链表

- 循环链表



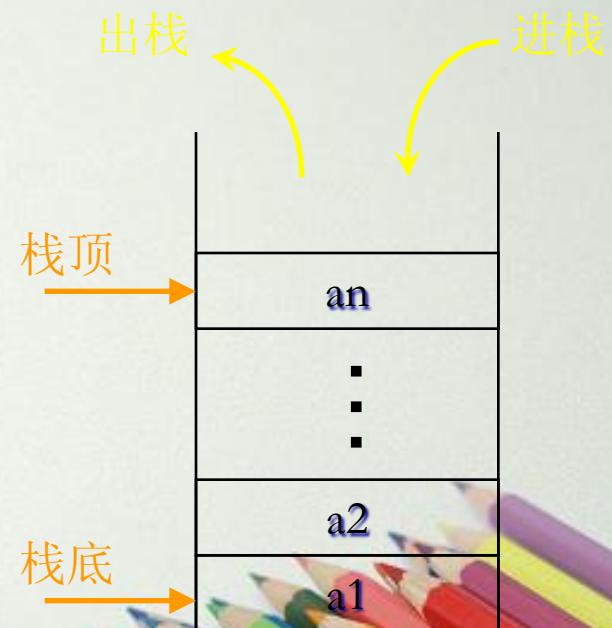
链表

- 双向循环链表



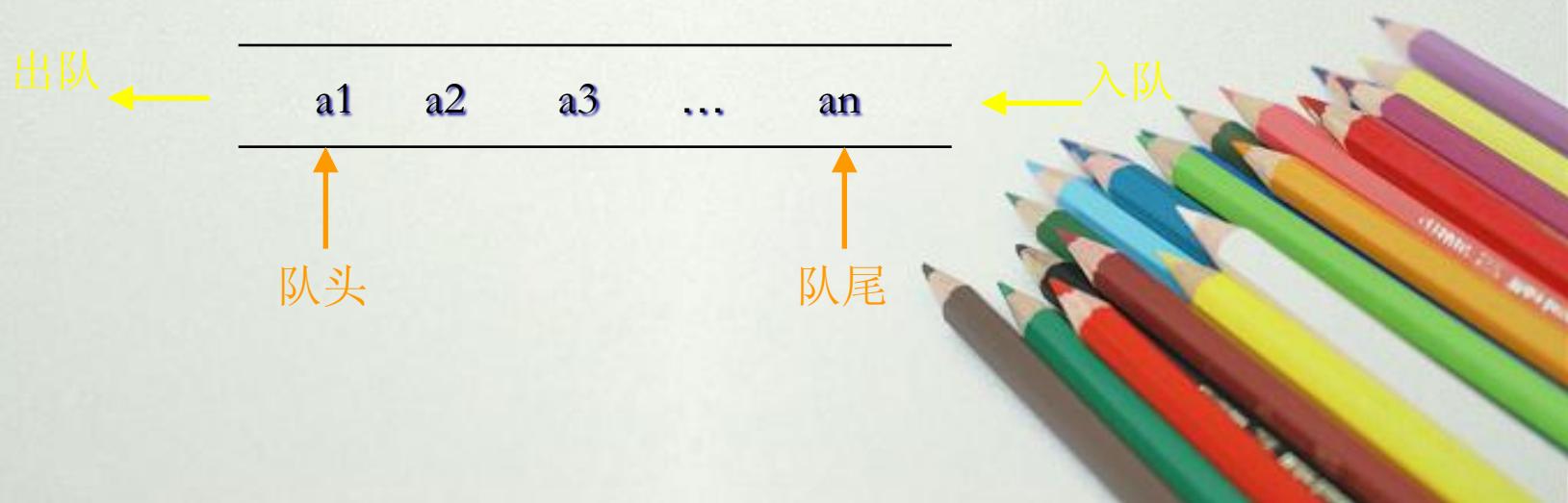
栈

- 栈(Stack)也是一种特殊的线性表，是一种后进先出(LIFO)的结构。
- 栈是限定仅在表尾进行插入和删除运算的线性表，表尾称为栈顶(top)，表头称为栈底(bottom)。
- 栈的物理存储可以用顺序存储结构，也可以用链式存储结构。



队列

- 队列(Queue)是限定所有的插入只能在表的一端进行，而所有的删除都在表的另一端进行的线性表。
- 表中允许插入的一端称为队尾(Rear)，允许删除的一端称为队头(Front)。
- 队列的操作是按先进先出(FIFO)的原则进行的。
- 队列的物理存储可以用顺序存储结构，也可以用链式存储结构。



LinkedList类

- 参见程序 MyQueue.java
- 参见程序 MyStack.java



ArrayList和LinkedList的比较

- ArrayList底层采用数组完成，而 LinkedList则是以一般的双向链表 (double-linked list)完成，其内每个对象除了数据本身外，还有两个引用，分别指向一个元素和后一个元素。
- 如果我们经常在List的开始处增加元素，或者在List中进行插入和删除操作，我们应该使用LinkedList，否则的话，使用ArrayList将更加快速。



HashSet类

- HashSet扩展AbstractSet并且实现Set接口。它创建一个类集，该类集使用散列表进行存储。散列表通过使用称之为散列法的机制来存储信息。在散列(hashing)中，一个关键字的信息内容被用来确定唯一的一个值，称为散列码(hash code)。而散列码被用来当做与关键字相连的数据的存储下标。关键字到其散列码的转换是自动执行的——你看不到散列码本身。你的程序代码也不能直接索引散列表。散列法的优点在于即使对于大的集合，它允许一些基本操作如add(), contains(), remove()和size()方法的运行时间保持不变



散列表

- 散列表又称为哈希表。散列表算法的基本思想是：以结点的关键字为自变量，通过一定的函数关系（散列函数）计算出对应的函数值，以这个值作为该结点存储在散列表中的地址。
- 当散列表中的元素存放太满，就必须进行再散列，将产生一个新的散列表，所有元素存放到新的散列表中，原先的散列表将被删除。在Java语言中，通过负载因子(**load factor**)来决定何时对散列表进行再散列。例如：如果负载因子是**0.75**，当散列表中已经有75%的位置已经放满，那么将进行再散列。
- 负载因子越高(越接近**1.0**)，内存的使用效率越高，元素的寻找时间越长。负载因子越低(越接近**0.0**)，元素的寻找时间越短，内存浪费越多。
- **HashSet**类的缺省负载因子是**0.75**。



HashSet类

- 下面的构造函数定义为：
 - HashSet()
 - HashSet(Collection c)
 - HashSet(int capacity)
 - HashSet(int capacity, float fillRatio)



HashSet类

- 第一种形式构造一个默认的散列集合。
- 第二种形式用c中的元素初始化散列集合。
- 第三种形式用capacity初始化散列集合的容量。
- 第四种形式用它的参数初始化散列集合的容量和填充比（也称为加载容量）。填充比必须介于0.0与1.0之间，它决定在散列集合向上调整大小之前，有多少能被充满。具体的说，就是当元素的个数大于散列集合容量乘以它的填充比时，散列集合被扩大。对于没有获得填充比的构造函数，默认使用0.75



HashSet类

HashSet没有定义更多的其他方法。

- 重要的是，注意散列集合并没有确保其元素的顺序，因为散列法的处理通常不让自已参与创建排序集合。如果需要排序存储，另一种类集——**TreeSet**将是一个更好的选择



HashSet类

- 参见程序 **HashSet1.java**
 - 如上面解释的那样，元素并没有按顺序进行存储



TreeSet类

- TreeSet为使用树来进行存储的Set接口提供了一个工具，**对象按升序存储**。访问和检索是很快的。在存储了大量的需要进行快速检索的排序信息的情况下，TreeSet是一个很好的选择。



TreeSet类

- 下面的构造函数定义为：
 - TreeSet()
 - TreeSet(Collection c)
 - TreeSet(Comparator comp)
 - TreeSet(SortedSet ss)



TreeSet类

- 第一种形式构造一个空的树集合，该树集合将根据其元素的自然顺序按升序排序。
- 第二种形式构造一个包含了c的元素的树集合。
- 第三种形式构造一个空的树集合，它按照由comp指定的比较函数进行排序（比较函数将在后面介绍）。
- 第四种形式构造一个包含了ss的元素的树集合



TreeSet类

- 参见程序 **TreeSet1.java**
 - 正如上面解释的那样，因为TreeSet按树存储其元素，它们被按照排序次序自动安排，如程序输出所示



通过迭代函数访问类集

- 通常希望循环通过类集中的元素。例如，可能会希望显示每一个元素。到目前为止，处理这个问题的最简单方法是使用iterator，iterator是一个或者实现Iterator或者实现ListIterator接口的对象。Iterator可以完成循环通过类集，从而获得或删除元素。ListIterator扩展Iterator，允许双向遍历列表，并可以修改单元



使用迭代函数

- 在通过迭代函数访问类集之前，必须得到一个迭代函数。每一个Collection类都提供一个**iterator()**函数，该函数返回一个对类集头的迭代函数。通过使用这个迭代函数对象，可以访问类集中的每一个元素，一次一个元素。通常，使用迭代函数循环通过类集的内容，步骤如下
 - 1. 通过调用类集的**iterator()**方法获得对类集头的迭代函数。
 - 2. 建立一个调用**hasNext()**方法的循环，只要**hasNext()**返回**true**，就进行循环迭代。
 - 3. 在循环内部，通过调用**next()**方法来得到每一个元素

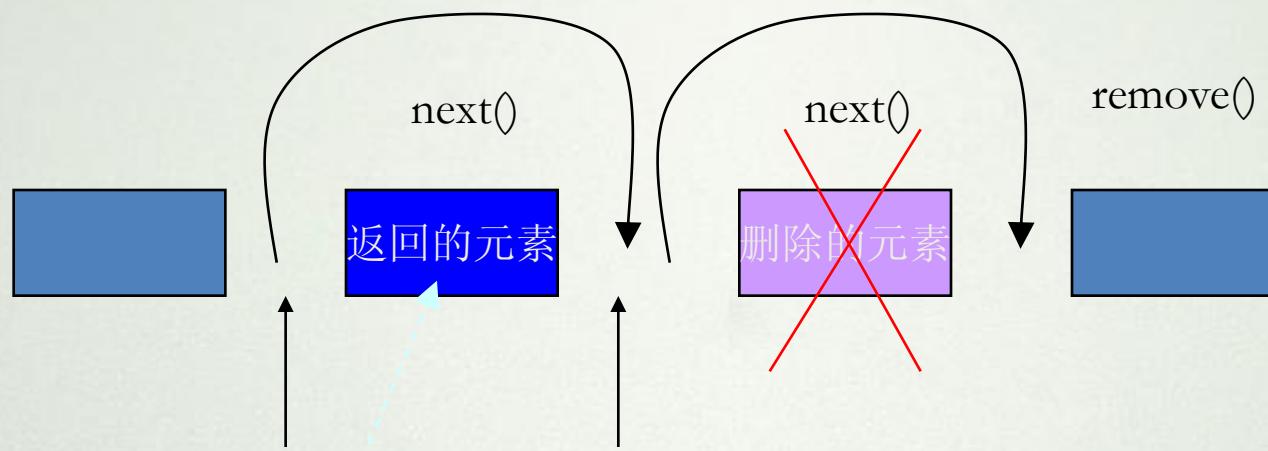


使用迭代函数

- 对于执行List的类集，也可以通过调用ListIterator来获得迭代函数。正如上面解释的那样，列表迭代函数提供了前向或后向访问类集的能力，并可让你修改元素。否则，ListIterator如同Iterator功能一样



迭代器的工作原理



使用迭代函数

- 参见程序 **Iterator1.java**
 - 这里是一个实现这些步骤的例子，说明了 Iterator 和 ListIterator。它使用 ArrayList 对象，但是总的原则适用于任何类型的类集。当然， ListIterator 只适用于那些实现 List 接口的类集



将用户定义的类存储于Collection中

- 为了简单，前面的例子在类集中存储内置的对象，如String或Integer。当然，类集并没有被限制为只能存储内置的对象。完全相反的是，类集的能力是它能存储任何类型的对象，包括你所创建的类的对象



将用户定义的类存储于Collection中

- 参见程序 `MailList.java`



处 理 映 射

- 除了类集，Java 2还在java.util中增加了映射。映射（map）是一个存储关键字和值的关联或者说是个关键字/值对的对象。给定一个关键字，可以得到它的值。关键字和值都是对象。关键字必须是唯一的。但值是可以重复的。有些映射可以接收null关键字和null值。而有的则不行



Map接口

- Map接口映射唯一关键字到值。关键字（key）是以后用于检索值的对象。给定一个关键字和一个值，可以存储这个值到一个Map对象中。当这个值被存储以后，就可以使用它的关键字来检索它。当调用的映射中没有项存在时，其中的几种方法会引发一个NoSuchElementException异常。而当对象与映射中的元素不兼容时，引发一个ClassCastException异常。如果试图使用映射不允许使用的null对象时，则引发一个NullPointerException异常。当试图改变一个不允许修改的映射时，则引发一个UnsupportedOperationException异常



Map接口

- 映射循环使用两个基本操作：get()和put()。使用put()方法可以将一个指定了关键字和值的值加入映射。为了得到值，可以通过将关键字作为参数来调用get()方法。调用返回该值。
- 映射不是类集，但可以获得映射的类集“视图”。为了实现这种功能，可以使用entrySet()方法，它返回一个包含了映射中元素的集合（Set）。为了得到关键字的类集“视图”，可以使用keySet()方法。为了得到值的类集“视图”，可以使用values()方法。类集“视图”是将映射集成到类集框架内的手段



SortedMap接口

- SortedMap接口扩展了Map，它确保了各项按关键字升序排序。当调用映射中没有的项时，其中的几种方法引发一个 NoSuchElementException 异常。当对象与映射中的元素不兼容时，则引发一个 ClassCastException 异常。当试图使用映射不允许使用的null对象时，则引发一个 NullPointerException 异常



SortedMap接口

- 排序映射允许对子映射（换句话说，就是映射的子集）进行高效的处理。使用headMap()，tailMap()或subMap()方法可以获得子映射。调用firstKey()方法可以获得集合的第一个关键字。而调用lastKey()方法可以获得集合的最后一个关键字



Map.Entry接口

- Map.Entry接口使得可以操作映射的输入。回想由Map接口说明的entrySet()方法，调用该方法返回一个包含映射输入的集合（Set）。这些集合元素的每一个都是一个Map.Entry对象



HashMap类

- HashMap类使用散列表实现Map接口。这允许一些基本操作如get()和put()的运行时间保持恒定，即便对大型集合，也是这样的
- 下面的构造函数定义为：
 - `HashMap()`
 - `HashMap(Map m)`
 - `HashMap(int capacity)`
 - `HashMap(int capacity, float fillRatio)`



HashMap类

- 第一种形式构造一个默认的散列映射。
- 第二种形式用m的元素初始化散列映射。
- 第三种形式将散列映射的容量初始化为capacity。
- 第四种形式用它的参数同时初始化散列映射的容量和填充比。容量和填充比的含义与前面介绍的HashSet中的容量和填充比相同。



HashMap类

- HashMap实现Map并扩展AbstractMap。它本身并没有增加任何新的方法
- 应该注意的是散列映射并不保证它的元素的顺序。因此，元素加入散列映射的顺序并不一定是它们被迭代函数读出的顺序



HashMap类

- 参见程序 `HashMapDemo.java`
 - 程序开始创建一个散列映射，然后将名字的映射增加到表中。接下来，映射的内容通过使用由调用函数`entrySet()`而获得的集合“视图”而显示出来。关键字和值通过调用由`Map.Entry`定义的`getKey()`和`getValue()`方法而显示。注意存款是如何被制成John Doe的账目的。`put()`方法自动用新值替换与指定关键字相关联的原先的值。因此，在John Doe的账目被更新后，散列映射将仍然仅仅保留一个“John Doe”账目。



TreeMap类

- TreeMap类通过使用树实现Map接口。 TreeMap提供了按排序顺序存储关键字/值对的有效手段，同时允许快速检索。应该注意的是，不像散列映射，树映射保证它的元素按照关键字升序排序



TreeMap类

- 下面的TreeMap构造函数定义为：
 - TreeMap()
 - TreeMap(Comparator comp)
 - TreeMap(Map m)
 - TreeMap(SortedMap sm)



TreeMap类

- 第一种形式构造一个空树的映射，该映射使用其关键字的自然顺序来排序。
- 第二种形式构造一个空的基于树的映射，该映射通过使用Comparator comp来排序（比较函数Comparators将在后面进行讨论）。
- 第三种形式用从m的输入初始化树映射，该映射使用关键字的自然顺序来排序。
- 第四种形式用从sm的输入来初始化一个树映射，该映射将按与sm相同的顺序来排序



TreeMap类

- TreeMap实现SortedMap并且扩展AbstractMap。而它本身并没有另外定义其他方法
- 参见程序 TreeMapDemo.java
 - 注意对关键字进行了排序。可以通过在创建映射时，指定一个比较函数来改变排序



比较函数

- TreeSet和TreeMap都按排序顺序存储元素。然而，精确定义采用何种“排序顺序”的是比较函数。通常在默认的情况下，这些类通过使用被Java称之为“自然顺序”的顺序存储它们的元素，而这种顺序通常也是你所需要的（A在B的前面，1在2的前面，等等）。如果需要用不同的方法对元素进行排序，可以在构造集合或映射时，指定一个Comparator对象。这样做为你提供了一种精确控制如何将元素储存到排序类集和映射中的能力



比较函数

- Comparator接口定义了两个方法：compare()和equals()。这里给出的compare()方法按顺序比较了两个元素：
- int compare(Object obj1, Object obj2)
- obj1和obj2是被比较的两个对象。当两个对象相等时，该方法返回0；当obj1大于obj2时，返回一个正值；否则，返回一个负值。如果用于比较的对象的类型不兼容的话，该方法引发一个ClassCastException异常。通过覆盖compare()，可以改变对象排序的方式。例如，通过创建一个颠倒比较输出的比较函数，可以实现按逆向排序



比较函数

- 这里给出的**equals()**方法， 测试一个对象是否与调用比较函数相等：
- **boolean equals(Object obj)**
- **obj**是被用来进行相等测试的对象。如果 **obj**和调用对象都是**Comparator**的对象并且使用相同的排序。该方法返回**true**。否则返回**false**。重写**equals()**方法是没有必要的，大多数简单的比较函数都不这样做



比较函数

- 参见程序 CompDemo.java
- 参见程序 CompDemo2.java



类集算法

- 类集框架定义了几种能用于类集和映射的算法。在Collections类中，这些算法被定义为静态方法。当试图比较不兼容的类型时，其中的一些算法引发一个ClassCastException异常；而当试图改变一个不可改变的类集时，则引发一个UnsupportedOperationException异常



类集算法

- 注意其中的几种方法，如synchronizedList()和synchronizedSet()被用来获得各种类集的同步（安全线程）拷贝。没有任何一个标准类集实现是同步的。必须使用同步算法来为其提供同步。
- 另一种观点：同步类集的迭代函数必须在synchronized块内使用。



类集算法

- 以unmodifiable开头的一组方法返回不能被改变的各种类集“视图”。这些方法当将一些进程对类集设为只读形式时很有用的
- Collections定义了三个静态变量：EMPTY_SET，EMPTY_LIST和EMPTY_MAP。它们都是不可改变的。EMPTY_MAP是在Java 2的1.3版中新增加的



类集算法

- 参见程序 `AlgorithmsDemo.java`
 - 注意`min()`和`max()`方法是在列表被混淆之后，对其进行操作的。两者在运行时，都不需要排序的列表



Arrays (数组)

- Java 2在java.util中新增加了一个叫做Arrays的类。这个类提供了各种在进行数组运算时很有用的方法。尽管这些方法在技术上不属于类集框架，但它们提供了跨越类集和数组的桥梁。在这一节中，分析由Arrays定义的每一种方法



Arrays (数组)

- `asList()`方法返回一个被指定数组支持的 `List`。换句话说，`列表和数组访问的是同一个单元`。它具有如下的形式
 - `static List asList(Object[] array)`
 - 这里`array`是包含了数据的数组



Arrays (数组)

- `binarySearch()`方法使用二分法搜索寻找指定的值。该方法必须应用于排序数组。它具有如下的形式
 - `static int binarySearch(byte[] array, byte value)`
 - `static int binarySearch(char[] array, char value)`
 - `static int binarySearch(double[] array, double value)`
 - `static int binarySearch(float[] array, float value)`
 - `static int binarySearch(int[] array, int value)`
 - `static int binarySearch(long[] array, long value)`
 - `static int binarySearch(short[] array, short value)`
 - `static int binarySearch(Object[] array, Object value)`
 - `static int binarySearch(Object[] array, Object value, Comparator c)`



Arrays (数组)

- 这里，array是被搜索的数组，而value是被查找的值。当array中包含的元素是不可比较的（例如Double和StringBuffer）或者当value与array中的类型不兼容时，后两种形式引发一个ClassCastException异常。在最后一种形式中，比较函数（Comparator）c用于确定array中的元素的顺序。在所有的形式中，如果array中含有value，则返回该元素的下标。否则，返回一个负值



Arrays (数组)

- 参见程序 ArraysDemo.java



从以前版本遗留下来的类和接口

- `java.util` 的最初版本中不包括类集框架。取而代之，它定义了几个类和接口提供专门的方法用于存储对象。随着在 Java 2 中引入类集，有几种最初的类被重新设计成支持类集接口。因此它们与框架完全兼容。尽管实际上没有类被摈弃，但其中某些仍被认为是过时的。当然，在那些重复从以前版本遗留下来的类的功能性的地方，通常都愿意用类集编写新的代码程序。一般地，对从以前版本遗留下来的类的支持是因为仍然存在大量使用它们的基本代码。包括现在仍在被 Java 2 的应用编程接口（API）使用的程序。



从以前版本遗留下来的类和接口

- 另一点，没有一个类集类是同步的。但是所有的从以前版本遗留下来的类都是同步的。这一区别在有些情况下是很重要的。当然，通过使用由Collections提供的算法也很容易实现类集同步



Enumeration接口

- Enumeration接口定义了可以对一个对象的类集中的元素进行枚举（一次获得一个）的方法。这个接口尽管没有被摈弃，但已经被Iterator所替代。Enumeration对新程序来说是过时的。然而它仍被几种从以前版本遗留下来的类（例如Vector和Properties）所定义的方法使用，被几种其他的API类所使用以及被目前广泛使用的应用程序所使用



Enumeration接口

- Enumeration指定下面的两个方法：
 - boolean hasMoreElements()
 - Object nextElement()
 - 执行后，当仍有更多的元素可提取时，
hasMoreElements()方法一定返回true。当所有元素都被
枚举了，则返回false。nextElement()方法将枚举中的下
一个对象做为一个类属Object的引用而返回。也就是每
次调用nextElement()方法获得枚举中的下一个对象。
调用例程必须将那个对象转换为包含在枚举内的对象
类型



Vector

- Vector实现动态数组。这与ArrayList相似，但两者不同的是：Vector是同步的，并且它包含了许多不属于类集框架的从以前版本遗留下来的方法。随着Java 2的公布，Vector被重新设计来扩展AbstractList和实现List接口，因此现在它与类集是完全兼容的



Vector

- 这里是Vector的构造函数：
 - Vector()
 - Vector(int size)
 - Vector(int size, int incr)
 - Vector(Collection c)



Vector

- 第一种形式创建一个原始大小为10的默认矢量。
- 第二种形式创建一个其原始容量由size指定的矢量。
- 第三种形式创建一个其原始容量由size指定，并且它的增量由incr指定的矢量。增量指定了矢量每次允许向上改变大小的元素的个数。
- 第四种形式创建一个包含了类集c中元素的矢量。
这个构造函数是在Java 2中新增加的



Vector

- 所有的矢量开始都有一个原始的容量。在这个原始容量达到以后，下一次再试图向矢量中存储对象时，矢量自动为那个对象分配空间同时为别的对象增加额外的空间。通过分配超过需要的内存，矢量减小了可能产生的分配的次数。这种次数的减少是很重要的，因为分配内存是很花时间的。在每次再分配中，分配的额外空间的总数由在创建矢量时指定的增量来确定。如果没有指定增量，在每个分配周期，矢量的大小增一倍



Vector

- Vector定义了下面的保护数据成员：
 - int capacityIncrement;
 - int elementCount;
 - Object elementData[];
 - 增量值被存储在capacityIncrement中。矢量中的当前元素的个数被存储在elementCount中。保存矢量的数组被存储在elementData中



Vector

- 因为Vector实现List，所以可以像使用ArrayList的一个实例那样使用矢量。也可以使用它的从以前版本遗留下来的方法来操作它。例如，在后面实例化Vector，可以通过调用addElement()方法而为其增加一个元素。调用elementAt()方法可以获得指定位置处的元素。调用firstElement()方法可以得到矢量的第一个元素。调用lastElement()方法可以检索到矢量的最后一个元素。使用indexOf()和lastIndexOf()方法可以获得元素的下标。调用removeElement()或removeElementAt()方法可以删除元素



Vector

- 参见程序 `VectorDemo.java`
- 随着Java 2的公布，`Vector`增加了对迭代函数的支持。现在可以使用迭代函数来替代枚举去遍历对象（正如前面的程序所做的那样）。例如，下面的基于迭代函数的程序代码可以被替换到上面的程序中



Vector

- // use an iterator to display contents

```
Iterator vItr = v.iterator();
```

```
System.out.println("\nElements in vector:");
```

```
while(vItr.hasNext())
```

```
    System.out.print(vItr.next() + " ");
```

```
System.out.println();
```



Vector

- 因为建议不要使编写枚举新的程序代码，所以通常可以使用迭代函数来对矢量的内容进行枚举。当然，业已存在的大量的老程序采用了枚举。不过幸运的是，枚举和迭代函数的工作方式几乎相同



Stack

- Stack是Vector的一个子类，它实现标准的后进先出堆栈。Stack仅仅定义了创建空堆栈的默认构造函数。Stack包括了由Vector定义的所有方法，同时增加了几种它自己定义的方法



Stack

- 参见程序 StackDemo.java



Dictionary

- 字典（Dictionary）是一个表示关键字/值存储库的抽象类，同时它的操作也很像映射（Map）。给定一个关键字和值，可以将值存储到字典（Dictionary）对象中。一旦这个值被存储了，就能够用它的关键字来检索它。因此，与映射一样，字典可以被当做关键字/值对列表来考虑。尽管在 Java 2 中并没有摈弃字典（Dictionary），由于它被映射（Map）所取代，从而被认为是过时的。然而由于目前Dictionary被广泛地使用，因此这里仍对它进行详细的讨论



Hashtable

- 散列表（Hashtable）是原始java.util中的一部分同时也是Dictionary的一个具体实现。然而，Java 2重新设计了散列表（Hashtable）以便它也能实现映射（Map）接口。因此现在Hashtable也被集成到类集框架中。它与HashMap相似，但它是同步的。
- 和HashMap一样，Hashtable将关键字/值对存储到散列表中。使用Hashtable时，指定一个对象作为关键字，同时指定与该关键字相关联的值。接着该关键字被散列，而把得到的散列值作为存储在表中的值的下标



Hashtable

- 散列表仅仅可以存储重载由Object定义的 hashCode()和equals()方法的对象。
hashCode()方法计算并返回对象的散列码。当然，equals()方法比较两个对象。幸运的是，许多Java内置的类已经实现了 hashCode()方法。例如，大多数常见的 Hashtable类型使用字符串（String）对象作为关键字。String实现hashCode()和equals()方法



Hashtable

- Hashtable的构造函数如下所示：
 - Hashtable()
 - Hashtable(int size)
 - Hashtable(int size, float fillRatio)
 - Hashtable(Map m)



Hashtable

- 第一种形式是默认的构造函数。
- 第二种形式创建一个散列表，该散列表具有由size指定的原始大小。
- 第三种形式创建一个散列表，该散列表具有由size指定的原始大小和由fillRatio指定的填充比。填充比必须介于0.0和1.0之间，它决定了在散列表向上调整大小之前散列表的充满度。具体地说，当元素的个数大于散列表的容量乘以它的填充比时，散列表被扩展。如果没有指定填充比，默
认使用0.75。最后，
- 第四种形式创建一个散列表，该散列表用m中的元素初始化。散列表的容量被设为m中元素的个数的两倍。默认的填充因子设为0.75。第四种构造函数是在Java 2中新增加的



Hashtable

- 参见程序 HTDemo.java



Hashtable

- 重要的一点是：和映射类一样，Hashtable不直接支持迭代函数。因此，上面的程序使用枚举来显示balance的内容。然而，我们可以获得允许使用迭代函数的散列表的集合视图。为了实现它，可以简单地使用由Map定义的一个类集“视图”方法，如entrySet()或keySet()方法。例如，可以获得关键字的一个集合“视图”，并遍历这些关键字。



Hashtable

- 参见程序 HTDemo2.java



Properties

- 属性（Properties）是Hashtable的一个子类。它用来保持值的列表，在其中关键字和值都是字符串（String）。Properties类被许多其他的Java类所使用。例如，当获得系统环境值时，System.getProperties()返回对象的类型



Properties

- Properties类的一个有用的功能是可以指定一个默认属性，如果没有值与特定的关键字相关联，则返回这个默认属性。例如，默认值可以与关键字一起在getProperty()方法中被指定——如 getProperty("name", "default value")。如果“name”值没有找到，则返回“default value”。当构造一个Properties对象时，可以传递Properties的另一个实例做为新实例的默认值。在这种情况下，如果对一个给定的Properties对象调用 getProperty("foo")，而“foo”并不存在时，Java在默认Properties对象中寻找“foo”。它允许默认属性的任意层嵌套



Properties

- 参见程序 PropDemo.java
- 参见程序 PropDemoDef.java



使用store()和load()

- Properties的一个最有用的方面是可以利用store()和load()方法方便地对包含在属性(Properties)对象中的信息进行存储或从盘中装入信息。在任何时候，都可以将一个属性(Properties)对象写入流或从中将其读出。这使得属性列表特别方便实现简单的数据库



使用store()和load()

- 参见程序 Phonebook.java



类集总 结

- 类集框架为程序员提供了一个功能强大的设计方案以解决编程过程中面临的大多数任务。下一次当你需要存储和检索信息时，可考虑使用类集
- 集合框架在我们今后的程序中将会被大量使用，希望同学们能达到熟练使用的程度

