Java集合

优逸客 让天下没有难学的技术

官网：http://www.sxuek.com/

# 第一章 Java 集合概述

一方面， 面向对象语言对事物的体现都是以对象的形式，为了方便对多个对象的操作，就要对对象进行存储。另一方面，使用Array存储对象方面具有一些弊端，而Java 集合就像一种容器，可以动态地把多个对象的引用放入容器中。

集合：也叫做容器，用来存放一组元素；

集合和数组的区别：

数组：1.数组的长度必须提前指定，并且指定后不能修改；

1. 数组只能保存相同数据类型的元素；
2. 适合保存基本类型元素；

集合：1.集合在使用时，长度不用指定，而且可以实现动态扩容；

2.集合如果没有指定泛型，可以存储所有数据类型（内部Object类型来处理）；

如果指定了泛型，则只能存储泛型要求的类型元素：

3.适合保存引用类型元素（集合内部将元素全部当作对象来处理）；

Java 集合类可以用于存储数量不等的多个对象，还可用于保存具有映射关系的关联数组。

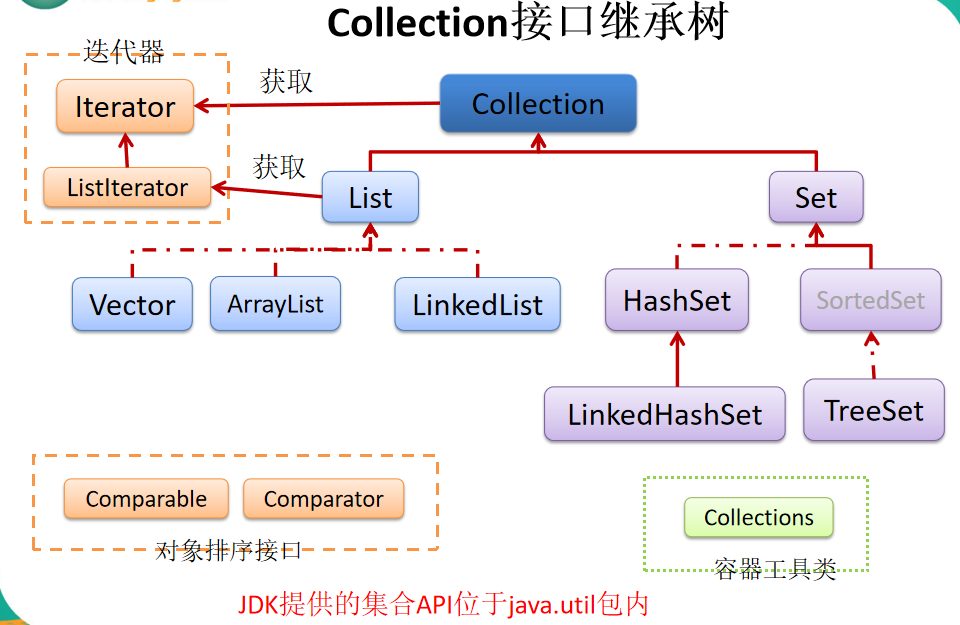
Java 集合可分为 Collection 和 Map 两种体系：

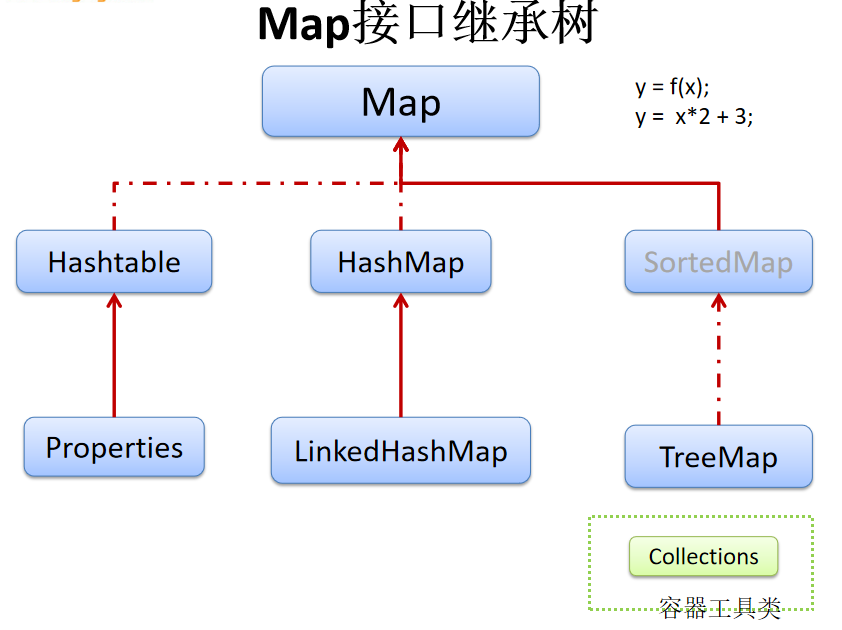
1）Collection接口：

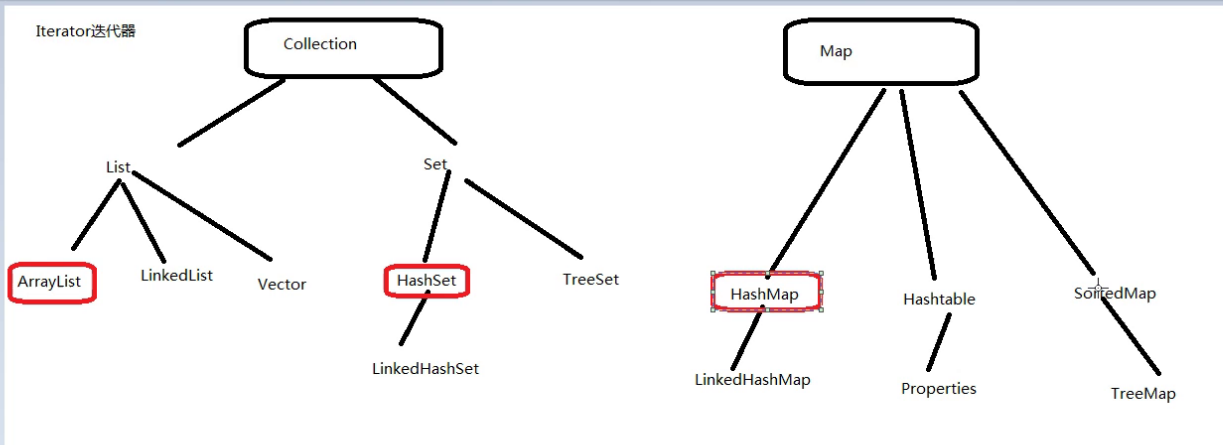
Set接口：元素无序、不可重复的集合；

List接口：元素有序，可重复的集合；

1. Map接口：具有映射关系“key-value对”的集合；





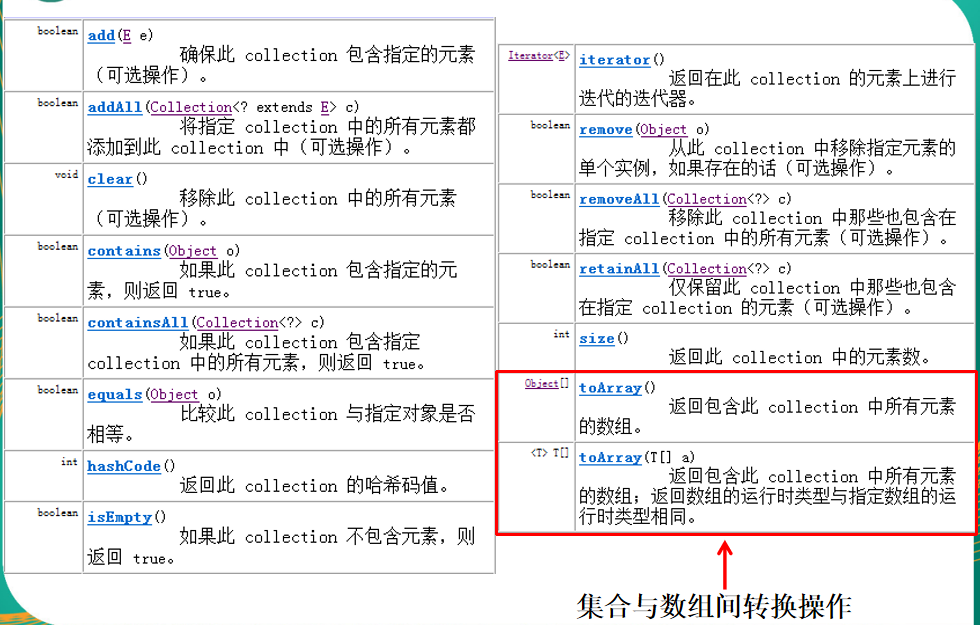


# 第二章 Collection 接口

查看API说明：

Collection 接口是 List、Set 和 Queue接口的父接口，该接口里定义的方法既可用于操作 Set 集合，也可用于操作 List 和 Queue 集合。JDK不提供此接口的任何直接实现，而是提供更具体的子接口(如：Set和List)实现。

在 Java5 之前，Java 集合会丢失容器中所有对象的数据类型，把所有对象都当成 Object 类型处理；从 JDK 5.0 增加了泛型以后，Java 集合可以记住容器中对象的数据类型。

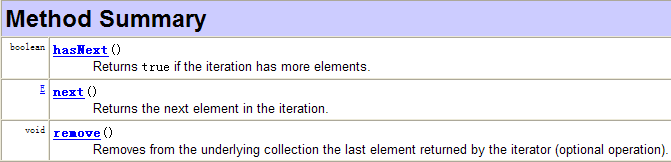


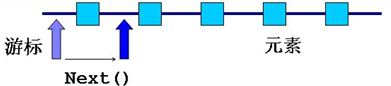
# 第三章 Iterator迭代器接口

查看API说明：

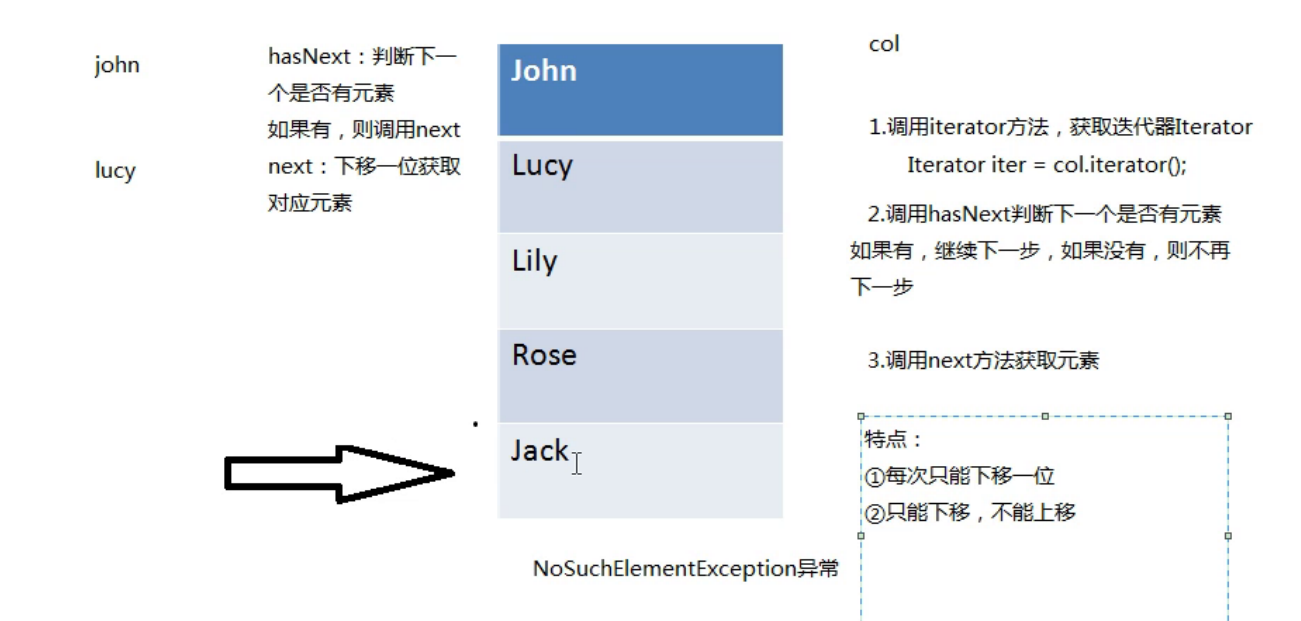
Iterator对象称为迭代器(设计模式的一种)，主要用于遍历 Collection 集合中的元素。所有实现了Collection接口的集合类都有一个iterator()方法，用以返回一个实现了Iterator接口的对象。Iterator 仅用于遍历集合，Iterator 本身并不提供承装对象的能力。如果需要创建 Iterator 对象，则必须有一个被迭代的集合。

## 3.1 Iterator接口的方法





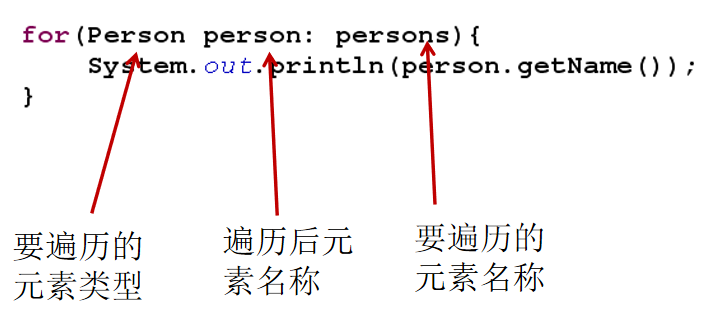




在调用it.next()方法之前必须要调用it.hasNext()进行检测。若不调用，且下一条记录无效，直接调用it.next()会抛出NoSuchElementException异常。

使用 foreach 循环遍历集合元素

Java 5.0 提供了 foreach 循环迭代访问 Collection



### 3.1.1 Collection接口的遍历时容易出现的问题：

### 3.1.2 Iterator接口遍历源码分析

# 第四章 List 接口

Java中数组用来存储数据的局限性；

List集合类中元素有序、且可重复，集合中的每个元素都有其对应的顺序索引。

List容器中的元素都对应一个整数型的序号记载其在容器中的位置，可以根据序号存取容器中的元素。

有序、无序是指在进行插入操作时，插入位置的顺序性，先插的位置在前，后插的位置在后，则为有序，反之无序；

JDK API中List接口的实现类常用的有：ArrayList、LinkedList和Vector。

List 集合里添加了一些根据索引来操作集合元素的方法：

## 4.1 List实现类之一：ArrayList

查看API说明：

ArrayList 是 List 接口的典型实现类、主要实现类；

本质上，ArrayList是对象引用的一个”变长”数组，ArrayList 是线程不安全的，而 Vector 是线程安全的，即使为保证 List 集合线程安全，也不推荐使用Vector。

Arrays.asList(…) 方法返回的 List 集合既不是 ArrayList 实例，也不是 Vector 实例。 Arrays.asList(…) 返回值是一个固定长度的 List 集合；

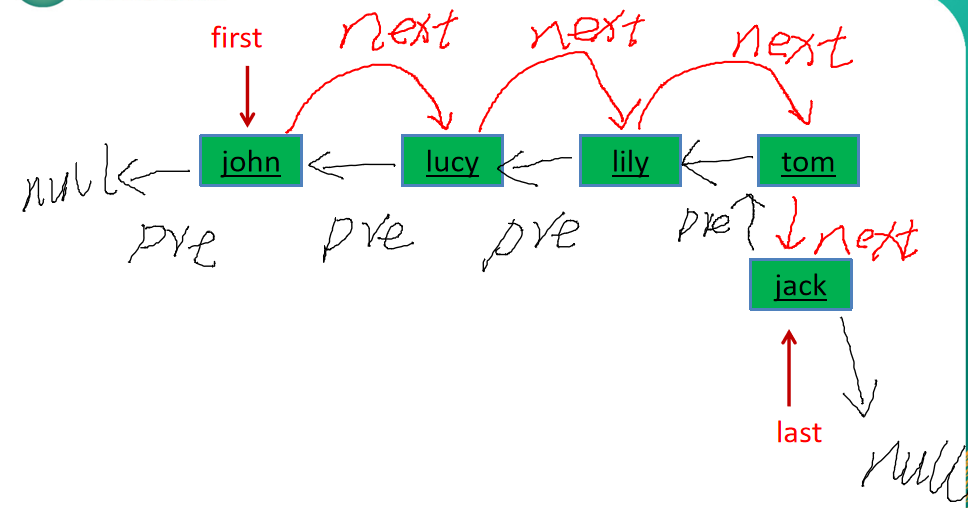
### 4.1.1 List接口的特点和特有方法：

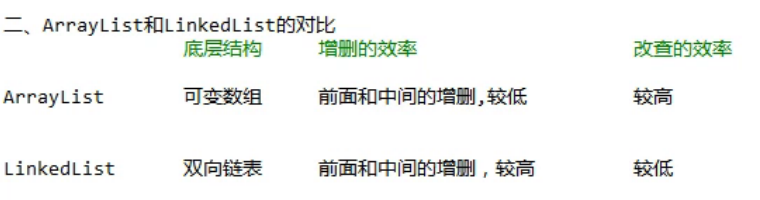
### 4.1.2 List接口的遍历方式

## 4.2 List实现类之二：LinkedList

查看API说明：

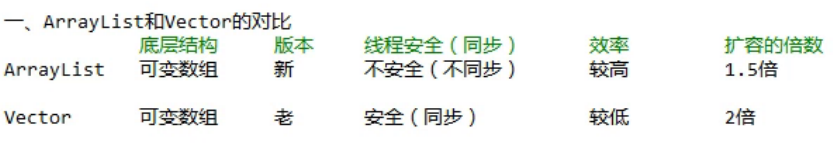
双向链表，LinkedList中维护了两个重要的属性first和last,分别指向首节点和尾节点。每一个节点（Node类型）里面又维护了三个属性item、next、prev，分别指向当前元素、下一个、上一个元素。





## 4.3 List 实现类之三：Vector

查看API说明：



## 4.4 List接口实现类总结：



# 第五章 Set 接口

查看API说明：

Set接口是Collection的子接口，set接口没有提供额外的方法；

Set 集合不允许包含相同的元素，如果试把两个相同的元素加入同一个 Set 集合中，则产生覆盖。

Set 判断两个对象是否相同不是使用 == 运算符，而是根据 equals 方法；

## 5.1 Set实现类之一：HashSet

查看API说明：

底层结构：维护了一个HashMap对象，基于哈希表结构存储元素；

哈希表结构：

底层构成：数组 + 链表 + 红黑树；



HashSet 按 Hash 算法来存储集合中的元素，因此具有很好的存取和查找性能。

HashSet 具有以下特点：

1）不能保证元素的排列顺序；

2）HashSet 不是线程安全的；

3）集合元素可以是 null；

当向 HashSet 集合中存入一个元素时，HashSet 会调用该对象的 hashCode() 方法来得到该对象的 hashCode 值，然后根据 hashCode 值决定该对象在 HashSet 中的存储位置。

HashSet 集合判断两个元素相等的标准：两个对象通过 hashCode() 方法比较相等，并且两个对象的 equals() 方法返回值也相等。

### 5.1.1 Set接口的特点和方法

### 5.1.3 hashCode() 方法

如果两个元素的 equals() 方法返回 true，但它们的 hashCode() 返回值不相等，hashSet 将会把它们存储在不同的位置，但依然可以添加成功。

对于存放在Set容器中的对象，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。

重写 hashCode() 方法的基本原则：

在程序运行时，同一个对象多次调用 hashCode() 方法应该返回相同的值；

当两个对象的 equals() 方法比较返回 true 时，这两个对象的 hashCode() 方法的返回值也应相等；

对象中用作 equals() 方法比较的 Field，都应该用来计算 hashCode 值；

### 5.1.4 equals()的重写：

以自定义的Customer类为例，何时需要重写equals()？

当一个类有自己特有的“逻辑相等”概念,当改写equals()的时候，总是要改写hashCode()，根据一个类的equals方法（改写后），两个截然不同的实例有可能在逻辑上是相等的，但是，根据Object.hashCode()方法，它们仅仅是两个对象。

因此，违反了“相等的对象必须具有相等的散列码”。

结论：复写equals方法的时候一般都需要同时复写hashCode方法。

为什么用eclipse复写hashCode方法，有31这个数字？

选择系数的时候要选择尽量大的系数。因为如果计算出来的hash地址越大，所谓的“冲突”就越少，查找起来效率也会提高。（减少冲突）并且31只占用5bits,相乘造成数据溢出的概率较小。

31可以由i\*31== (i<<5)-1来表示,现在很多虚拟机里面都有做相关优化.（提高算法效率）。31是一个素数，素数作用就是如果我用一个数字来乘以这个素数，那么最终的出来的结果只能被素数本身和被乘数还有1来整除！。(减少冲突)

案例：

## 5.2 Set实现类之二：LinkedHashSet

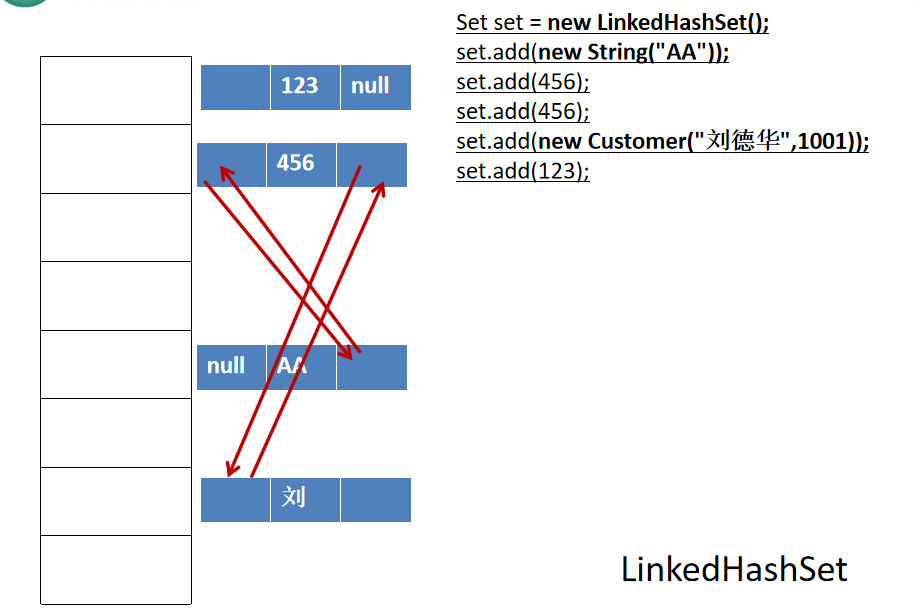
查看API说明：

LinkedHashSet 是 HashSet 的子类；

LinkedHashSet 根据元素的 hashCode 值来决定元素的存储位置，但它同时使用链表维护元素的次序，这使得元素看起来是以插入顺序保存的。

LinkedHashSet插入性能略低于 HashSet，但在迭代访问 Set 里的全部元素时有很好的性能。

LinkedHashSet 不允许集合元素重复。



## 5.3 Set实现类之三：TreeSet

查看API说明：

特点：1）不允许重复，里面不能存储null对象（null对象无法排序）；

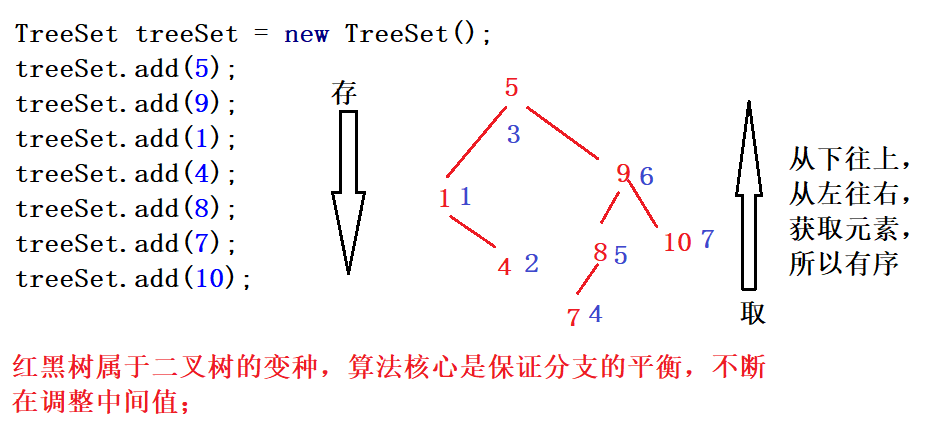
2）实现了对元素的排序：

1>自然排序（默认，必须实现Comparable接口）；

2>自定义排序；

TreeSet如何实现去重：

通过笔记哦方法的返回值是否为0来判断是否重复；



TreeSet 是 SortedSet 接口的实现类，TreeSet 可以确保集合元素处于排序状态。

### 5.3.1 排序-自然排序

自然排序：TreeSet 会调用集合元素的 compareTo(Object obj) 方法来比较元素之间的大小关系，然后将集合元素按升序排列，如果试图把一个对象添加到 TreeSet 时，则该对象的类必须实现 Comparable 接口。

实现 Comparable 的类必须实现 compareTo(Object obj)方法，两个对象即通过 compareTo(Object obj) 方法的返回值来比较大小。

Comparable 的典型实现：

BigDecimal、BigInteger 以及所有的数值型对应的包装类：按它们对应的数值大小进行比较；

Character：按字符的 unicode值来进行比较

Boolean：true 对应的包装类实例大于 false 对应的包装类实例

String：按字符串中字符的 unicode 值进行比较

Date、Calendar：后边的时间、日期比前面的时间、日期大

向 TreeSet 中添加元素时，只有第一个元素无须比较compareTo()方法，后面添加的所有元素都会调用compareTo()方法进行比较。

因为只有相同类的两个实例才会比较大小，所以向 TreeSet 中添加的应该是同一个类的对象，对于 TreeSet 集合而言，它判断两个对象是否相等的唯一标准是：两个对象通过 compareTo(Object obj) 方法比较返回值，当需要把一个对象放入 TreeSet 中，重写该对象对应的 equals() 方法时，应保证该方法与 compareTo(Object obj) 方法有一致的结果：如果两个对象通过 equals() 方法比较返回 true，则通过compareTo(Object obj) 方法比较应返回 0。

### 5.3.2 排序-定制排序

TreeSet的自然排序是根据集合元素的大小，进行元素升序排列。如果需要定制排序，比如降序排列，可通过Comparator接口的帮助。需要重写compare(T o1,T o2)方法。

利用int compare(T o1,T o2)方法，比较o1和o2的大小：如果方法返回正整数，则表示o1大于o2；如果返回0，表示相等；返回负整数，表示o1小于o2。

要实现定制排序，需要将实现Comparator接口的实例作为形参传递给TreeSet的构造器。

此时，仍然只能向TreeSet中添加类型相同的对象。否则发生ClassCastException异常。

使用定制排序判断两个元素相等的标准是：通过Comparator比较两个元素返回了0。

案例：

# 第六章 Map接口

查看API帮助文档；

Map与Collection并列存在。用于保存一组具有映射关系的数据:Key-Value

（映射：根据一个人的姓名，可以找到对应的人）；

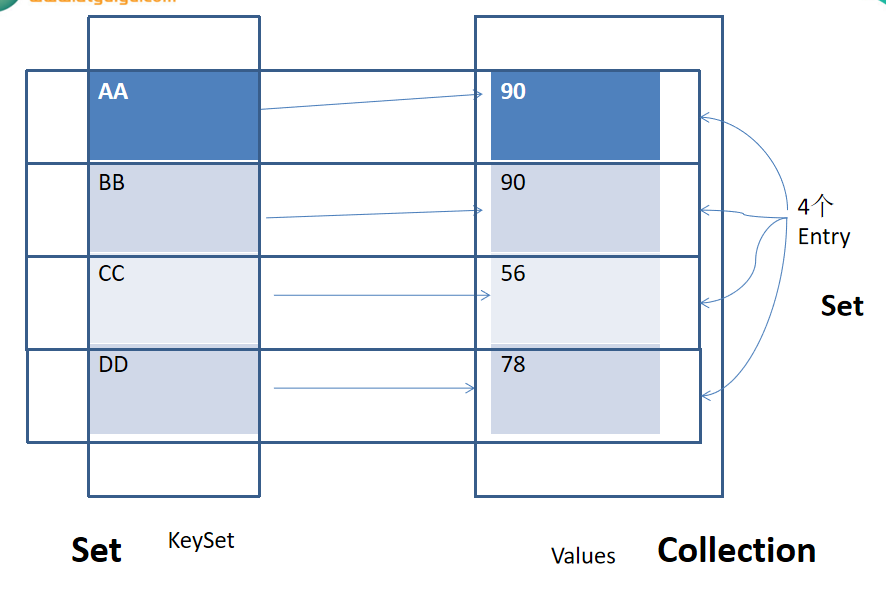
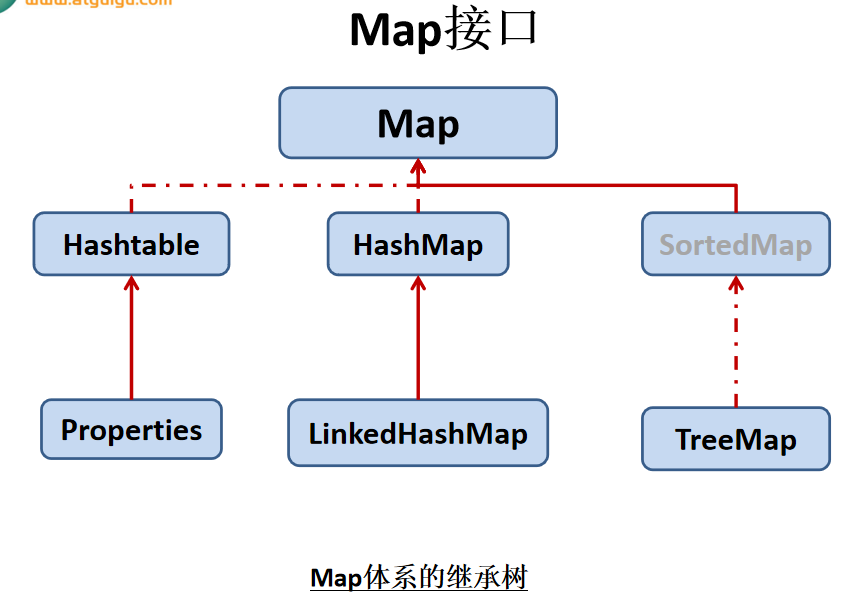
Map 中的 key 和 value 都可以是任何引用类型的数据；

Map 中的 key 用Set来存放，不允许重复，即同一个 Map 对象所对应的类，须重写hashCode()和equals()方法

常用String类作为Map的“键”

key 和 value 之间存在单向一对一关系，即通过指定的 key 总能找到唯一的、确定的 value；

Map接口的常用实现类：HashMap、TreeMap和Properties。



## 6.1 Map 常用方法

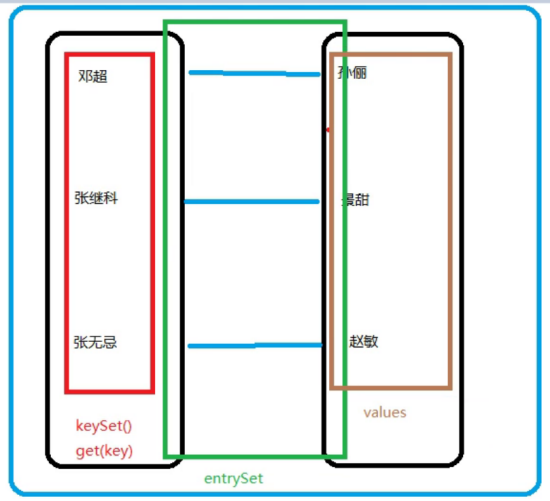
1）添加、删除操作：

2）元视图操作的方法：

3）元素查询的操作：

### 6.1.1 Map接口的特点和常见方法：

### 6.1.2 Map接口遍历方式



## 6.2 Map实现类之一：HashMap

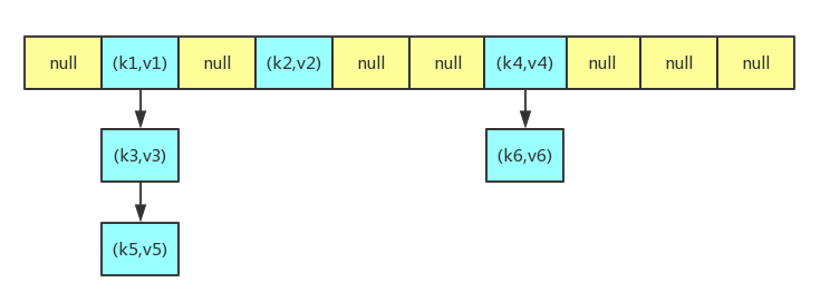
查看API帮助文档；

底层是哈希表；

允许使用null键和null值，与HashSet一样，不保证映射的顺序。

HashMap 判断两个 key 相等的标准是：两个 key 通过 equals() 方法返回 true，hashCode 值也相等。

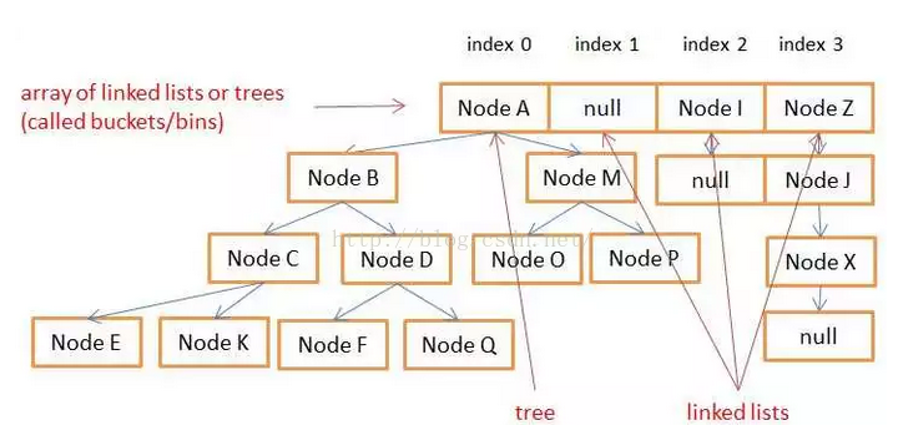
HashMap 判断两个 value相等的标准是：两个 value 通过 equals() 方法返回 true。



HashMap的存储结构：

JDK 7及以前版本：HashMap是数组+链表结构(即为链地址法)

JDK 8版本发布以后：HashMap是数组+链表+红黑树实现。

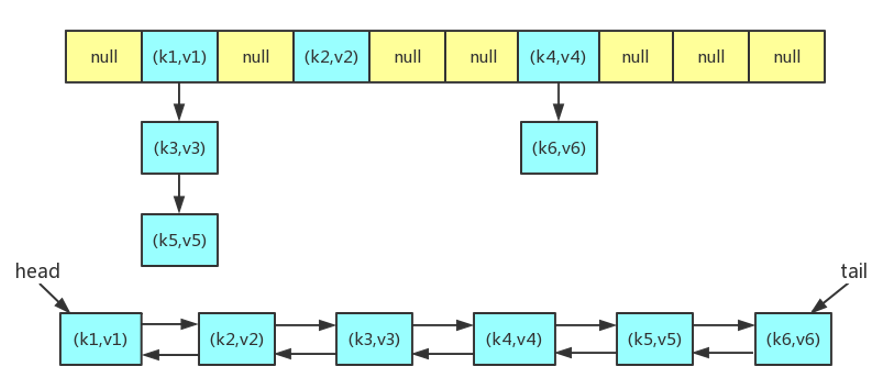


### 6.2.1 HashMap源码分析

## 6.3 Map实现类之二：LinkedHashMap

LinkedHashMap 是 HashMap 的子类

与LinkedHashSet类似，LinkedHashMap 可以维护 Map 的迭代顺序：迭代顺序与 Key-Value 对的插入顺序一致



## 6.4 Map实现类之三：TreeMap

底层结构：红黑树，可以实现对元素的key进行排序；

自然排序：要求key的元素类型实现Comparable，并实现comparaeTo()方法；

定制排序：要求创建TreeMap对象时，传入Comarator比较强对象，并实现compare()方法；

TreeMap存储 Key-Value 对时，需要根据 key-value 对进行排序。TreeMap 可以保证所有的 Key-Value 对处于有序状态。

TreeMap 的 Key 的排序：

自然排序：TreeMap 的所有的 Key 必须实现 Comparable 接口，而且所有的 Key 应该是同一个类的对象，否则将会抛出 ClasssCastException

定制排序：创建 TreeMap 时，传入一个 Comparator 对象，该对象负责对 TreeMap 中的所有 key 进行排序。此时不需要 Map 的 Key 实现 Comparable 接口

TreeMap判断两个key相等的标准：两个key通过compareTo()方法或者compare()方法返回0。

若使用自定义类作为TreeMap的key，所属类需要重写equals()和hashCode()方法，且equals()方法返回true时，compareTo()方法应返回0。

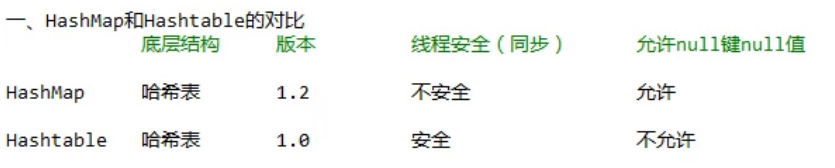
## 6.5 Map实现类之四：Hashtable

Hashtable是个古老的 Map 实现类，线程安全。

与HashMap不同，Hashtable 不允许使用 null 作为 key 和 value

与HashMap一样，Hashtable 也不能保证其中 Key-Value 对的顺序

Hashtable判断两个key相等、两个value相等的标准，与hashMap一致。



## 6.6 Map实现类之五：Properties

# 第七章 Collections工具类

查看API帮助文档；

Collections 是一个操作 Set、List 和 Map 等集合的工具类；

Collections 中提供了一系列静态的方法对集合元素进行排序、查询和修改等操作，还提供了对集合对象设置不可变、对集合对象实现同步控制等方法；

## 7.1 Collections工具类的常见方法

# 第八章 泛型

## 8.1 泛型概述

泛型：JDK1.5的特性，参数化的类型；可以将某一个类型当做参数传递给类、接口或者方法中；

泛型的好处:

1. 编译时检查待添加的元素类型，提高了类型的安全性；
2. 减少了类型转换的次数，提高了效率；

没有使用泛型：String --->Object --->String;

使用泛型：String--->String--->String;

1. 减少编译警告；

案例：

## 8.2 泛型API:

## 8.3 自定义泛型

## 8.4 泛型中的通配符使用