集合框架体系--下

学习内容

- List集合详解
- Set集合详解
- Map集合详解

一.List集合的实现类

- ArrayList集合（类）：实现List接口。-----重点掌握

  1.ArrayList集合中的特有方法是实现List 底层使用可变数组结构。

  2.查询遍历的效率比较高、增删的效率比较低

  3.属于线程不安全的集合类。执行效率比较高

- LinkedList集合（类）：实现List接口。------熟悉

  1. 底层使用链表结构。（链表：有头有尾）

  2. LinkedList集合中的特有方法都是围绕链表的头尾设计

  3. 查询遍历的效率比较慢、增删的效率比较高

  4.属于线程不安全的集合类。执行效率比较高

- Vector集合（类）：实现List接口。----------了解即可

  1.底层使用可变数组结构。查询遍历效率、增删效率都比较低。

  2.线程安全的集合类,由于效率比较低，实际开发中基本不使用了。

1.1 ArrayList介绍(重点掌握)

根据我们之前所学的知道，List下面有三个比较重要的实现类，分别是ArrayList、LinkedList和Vector类，我们这里先学习最重要的ArrayList类。

ArrayList类（集合）：它是List接口的一个直接实现类，因此ArrayList集合拥有List接口的所有特性。

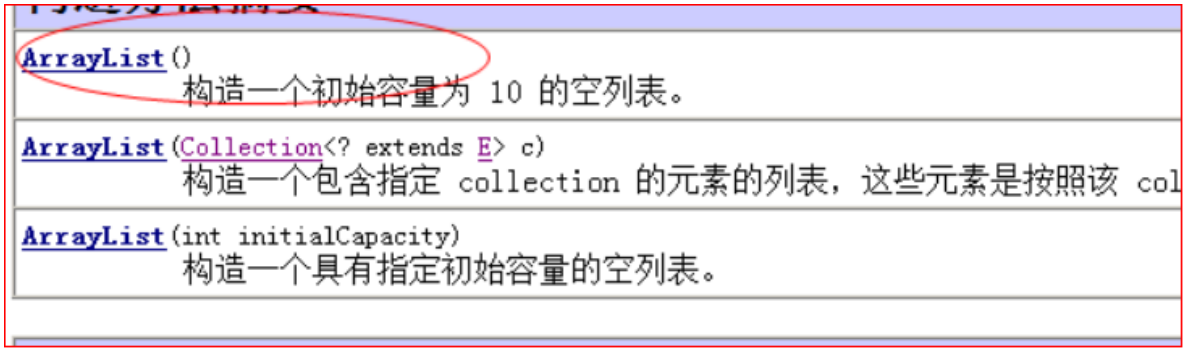
1.1.1ArrayList集合的特点：

1、它的底层使用的可变数组，如果初始容量不够会立刻重新创建数组，将原来的数据复制存放到新的数组前面空间中，多余的数据继续向数组其他空间进行存储。

2、它有下标，可以按照数组的方式操作ArrayList集合；

3、可以保证数据的存取顺序（第一个元素添加到集合中，那么取出也是第一个取出），同时还可以保存重复元素，也可以存储null元素；

4、ArrayList集合它增加和删除效率比较低，而查询（遍历）速度较快；



1.1.2 ArrayList和StringBuilder扩容原理与扩容源码介绍。

```
public static void main(String[] args) {
    /**
     * Arraylist 扩容原理
     *  初始化默认容量为10,当容量不足时，扩容为原来的1.5倍
     * StringBuilder
     *  初始化默认容量为16，当容量不足时，扩容为 16*2 + 2 = 34
     */
    ArrayList<String> list = new ArrayList<String>(20);
```

ArrayList查看扩容源码：

ArrayList 进去-》ctrl + 0 ->add() -> 进去 ->

ensureCapacityInternal->ensureExplicitCapacity

->grow

StringBuilder查看扩容源码：

StringBuilder进去->ctrl + 0 ->append()->进去连续几个

->ensureCapacityInternal->expandCapacity

1.2 LinkedList介绍(掌握)

1.2.1 LinkedList定义：

LinkedList集合：它也是List接口的实现类，它的底层使用的链接列表（链表）数据结构。

链表结构的特点：有头有尾。

链表结构：由一个链子把多个节点连接起来的数据结构。

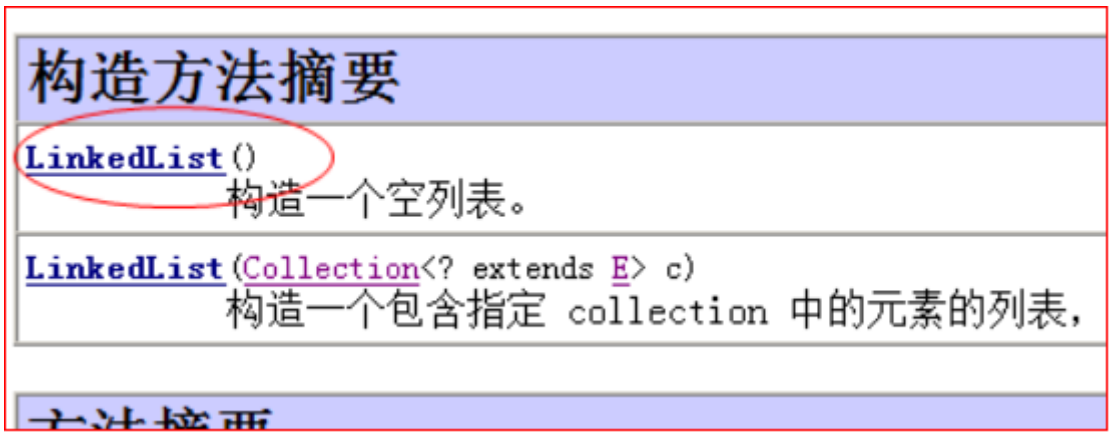
节点：在链表结构中每个可以保存数据的空间称为节点，而一个链表结构是由多个节点组成的。

也就是说链表结构使用节点来存储数据，每个节点(存储数据的空间)可以分成若干部分，其中有一部分存储数据，另外一部分存储的是其他节点的地址。

1.2.2 LinkedList特点：

- 它的底层使用的链表结构。
- 有头有尾，其中的方法都是围绕头和尾设计的。
- LinkedList集合可以根据头尾进行各种操作，但它的增删效率高，查询效率低。
- LinkedList集合增删效率高是因为底层是链表结构，如果增加或者删除只需要在增加或者删除节点的位置上记住新的节点的地址即可，而其他节点不需要移动，所以速度会快。
- 而查询遍历由于链表结构的特点，查询只能从头一直遍历到链表的结尾，所以速度会慢。
- LinkedList集合底层也是线程不安全。效率高。
- 也可以存储null元素。

1.2.3 LinkedList集合的特有方法



说明：

这里只有两个构造函数，不像ArrayList集合有三个构造函数，而这里不需要给容器初始化容量大小，因为LinkedList集合的特点就是链表结构，在底层如果新增加一个元素那么前一个节点记住新增加的节点地址，而新增加的节点记住后一个节点地址就可以，什么时候增加什么记住地址即可，不需要初始化容量大小。

由于LinkedList是链表结构，而链表有头有尾，所以在LinkedList集合中专门为链表结构提供了特有的函数。

练习：LinkedList集合中的特有函数

分析和步骤：

1. 使用new关键字创建LinkedList类的对象list，并赋值为LinkedList类型；
2. 使用对象list调用LinkedList集合中的addFirst()函数向集合中添加字符串；
3. 利用for循环和迭代器迭代遍历LinkedList集合；
4. 使用对象list调用LinkedList类中的getLast()函数获取最后一个元素并输出；
5. 使用对象list调用LinkedList类中的removeFirst()函数删除第一个元素并输出；

```
/**
 * @author IT枫斗者
 * @ClassName LinkedListDemo01.java
 * @From www.javatiaozao.com
```

```
 * @Description TODO
 */
public class LinkedListDemo01 {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();

        //添加元素
        list.addFirst("AAA");
        list.addFirst("bbb");
        list.addFirst("ccc");
        //遍历集合
        for (Iterator it =  list.iterator();it.hasNext();){
            System.out.println(it.next());
        }
        System.out.println("==============");
        //获取最后一个
        System.out.println(list.getLast());
        //删除第一个元素
        System.out.println(list.removeFirst());
    }
}
```

1.3疑问解答

1、遇到对线程有需求的情况，应该使用哪个集合类?

答：还使用LinkedList、ArrayList（在后面学习过程中，可以解决LinkedList\ ArrayList线程不安全的问题）

2、为什么ArrayList集合增删效率低，而查询速度快?

答：因为我们向集合中添加元素的时候，有时会将元素添加到集合中的最前面，或者有可能删除最前面的数据，这样就导致其他数据向后移动或者删除时向前移动，所以效率会低。

对于查询ArrayList集合，由于ArrayList集合是数组结构，而数组结构是排列有序的，并且下标是有序增加的，当查询ArrayList集合的时候可以按照排列顺序去查询，或者直接可以通过某个下标去查询，这样就会导致查询速度相对来说会快很多。

3、为什么LinkedList集合增删效率快，而查询速度慢?

答：LinkedList集合增删效率高是因为底层是链表结构，如果增加或者删除只需要在增加或者删除节点的位置上记住新 的节点的地址即可，而其他节点不需要移动，所以速度会快。而查询遍历由于链表结构的特点，查询只能从头一直遍历到链表的结尾，所以速度会慢。

注意：学习了这么多集合类，在开发中如果不知道使用哪个集合，到底什么时候使用ArrayList集合和LinkedList集合?

1）如果对集合进行查询操作建议使用ArrayList集合；

2）如果对集合进行增删操作建议使用LinkedList集合；

总结：如果实在把握不好使用的时机，建议大家以后在开发中都使用ArrayList集合即可，因为在实际开发中我们对于集合的操作几乎都是查询操作，很少执行增删操作的。

二.Set集合(无索引、无序、唯一)

2.1Set集合概述和特点【应用】

- 不可以存储重复元素

- 没有索引,不能使用普通for循环遍历

java.util
接口 Set<E>

类型参数：
E - 此 set 所维护元素的类型

所有超级接口：
Collection<E>, Iterable<E>

所有已知子接口：
NavigableSet<E>, SortedSet<E>

所有已知实现类：
AbstractSet, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet, JobStateReasons, LinkedHashSet, TreeSet

---

public interface Set<E>
extends Collection<E>

一个不包含重复元素的 collection。更确切地讲，set 不包含满足 e1.equals(e2) 的元素对 e1 和 e2，并且最多包含一个 null 元素。正如其名称所暗示的，此接口模仿了数学上的 *set* 抽象。

说明：

- Set接口中没有自己的特有函数，所有的函数全部来自于Collection接口。
- Set集合没有索引，只能通过Iterator迭代器遍历获取集合中的元素，Set集合不能使用ListIterator迭代器，因为ListIterator只是针对List集合特有的迭代器。
- 不包含重复元素,不保证集合中的元素顺序。

2.2Set集合的使用【应用】

存储字符串并遍历

```
public class MySet01 {
    public static void main(String[] args) {
        //创建集合对象
        Set<String> set = new TreeSet<>();
        //添加元素
        set.add("ccc");
        set.add("aaa");
        set.add("aaa");
        set.add("bbb");

//        for (int i = 0; i < set.size(); i++) {
//            //Set集合是没有索引的，所以不能使用通过索引获取元素的方法
//        }

        //遍历集合
        Iterator<String> it = set.iterator();
        while (it.hasNext()){
            String s = it.next();
            System.out.println(s);
        }
        System.out.println("-----------------------------------");
        for (String s : set) {
            System.out.println(s);
        }
    }
```

```
}
```

3.HashSet集合(理解)

3.1HashSet集合概述和特点【应用】

- 底层数据结构是哈希表
- 存取无序
- 不可以存储重复元素
- 没有索引,不能使用普通for循环遍历

java.util
类 HashSet<E>

java.lang.Object
  └ java.util.AbstractCollection<E>
      └ java.util.AbstractSet<E>
          └ java.util.HashSet<E>

类型参数：
E - 此 set 所维护的元素的类型

所有已实现的接口：
Serializable, Cloneable, Iterable<E>, Collection<E>, Set<E>

直接已知子类：
JobStateReasons, LinkedHashSet

---

public class HashSet<E>
extends AbstractSet<E>
implements Set<E>, Cloneable, Serializable

此类实现 Set 接口，由哈希表（实际上是一个 HashMap 实例）支持。它不保证 set 的迭代顺序；特别是它不保证该顺序恒久不变。此类允许使用 null 元素。

3.2HashSet集合的基本应用【应用】

存储字符串并遍历

```
public class HashSetDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        HashSet<String> set = new HashSet<String>();

        //添加元素
        set.add("hello");
        set.add("world");
        set.add("java");
        //不包含重复元素的集合
        set.add("world");

        //遍历
        for(String s : set) {
            System.out.println(s);
        }
    }
}
```

通过以上程序输出结果得出结论：

1）HashSet集合不能存储重复的元素；

2）HashSet集合存储元素的顺序不固定；

接下我们要分析为什么HashSet集合存储的数据顺序不固定和为什么不支持存储重复的元素？

答案肯定和HashSet集合的底层哈希表数据结构有关系，所以接下来我们要学习什么是哈希表。

3.3哈希值【理解】

- 哈希值简介

  是JDK根据对象的地址或者字符串或者数字算出来的int类型的数值

- 如何获取哈希值

  Object类中的public int hashCode()：返回对象的哈希码值

- 哈希值的特点
  - 同一个对象多次调用hashCode()方法返回的哈希值是相同的
  - 默认情况下，不同对象的哈希值是不同的。而重写hashCode()方法，可以实现让不同对象的哈希值相同

3.4哈希表结构【理解】

3.4.1 哈希表原理

哈希表：

它是一个数据结构，底层依赖的是数组，只是不按照数组的下标操作数组中的元素。需要根据数组中存储的元素的哈希值进行元素操作。

哈希表的存储过程：

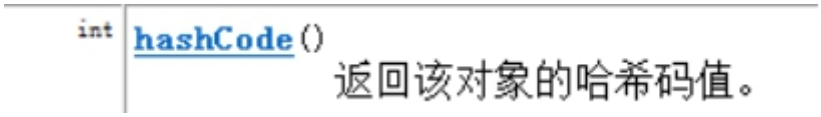
1）哈希表底层是一个数组，我们必须知道集合中的一个对象元素到底要存储到哈希表中的数组的哪个位置，也就是需要一个下标。

2）哈希表会根据集合中的每个对象元素的内容计算得出一个整数值。由于集合中的对象元素类型是任意的，而现在这里使用的算法必须是任意类型的元素都可以使用的算法。能够让任意类型的对象元素都可以使用的算法肯定在任意类型的对象所属类的父类中，即上帝类Object中，这个算法就是Object类中的hashCode()函数。

结论：要给HashSet集合中保存对象，需要调用对象的hashCode函数。

解释说明：

通过查阅API得知，使用Object的任意子类对象都可以调用Object类中的hashCode()函数并生成任意对象的哈希码值。



代码演示如下：

```
String str="bbbb";
int hashCode = str.hashCode();
System.out.println(hashCode);
System.out.println("AAAA".hashCode());
```

输出结果：
```
3016832
2000960
```

由以上结果可以看出任意对象调用Object类中的hashCode()函数都会生成一个整数。

3）hashCode算法，得到一个整数，但是这个整数太大了，这个值不能直接作为数组下标的。所以底层还会对这个值结合数组的长度继续计算运行，得到一个在0~数组长度之间的整数，这样就可以作为数组的下标了。

问题1：使用hashCode函数生成的一个过大整数是用什么算法将将生成哈希码值变成0~数组长度之间的数字呢?

其中一种最简单的算法是可以实现的，举例：假设底层哈希表中数组长度是5，那么下标的范围是0~4，

所以我们这里可以使用生成的哈希码值（过大的整数）对数组长度取余数，我们发现任何数在这里对5取余都是在 0 1 2 3 4 之间，所以这样就可以获取到0~数组长度之间的下标了。

问题2：如果数据过多，HashSet底层的数组存储不下，怎么办?

hashSet集合底层数组初始容量是16，如果hashSet中元素个数超过 16*0.75=12 的时候，就把数组的大小扩展 为 16*2=32 ，即扩大一倍。(加载因子默认是0.75F)

4）如果上述做法已经计算出底层数组的下标位置，那么就要判断计算出的下标位置是否已经有元素了：

A.如果下标对应的位置没有元素：直接存储数据；

B.如果下标对应的位置有元素：这时就必须调用对象的equals()函数比较两个对象是否相同：

如果结果是true：相同，直接将要添加的数据丢弃；

如果结果是false：不相同，那直接将数据存储到数组当前空间位置；

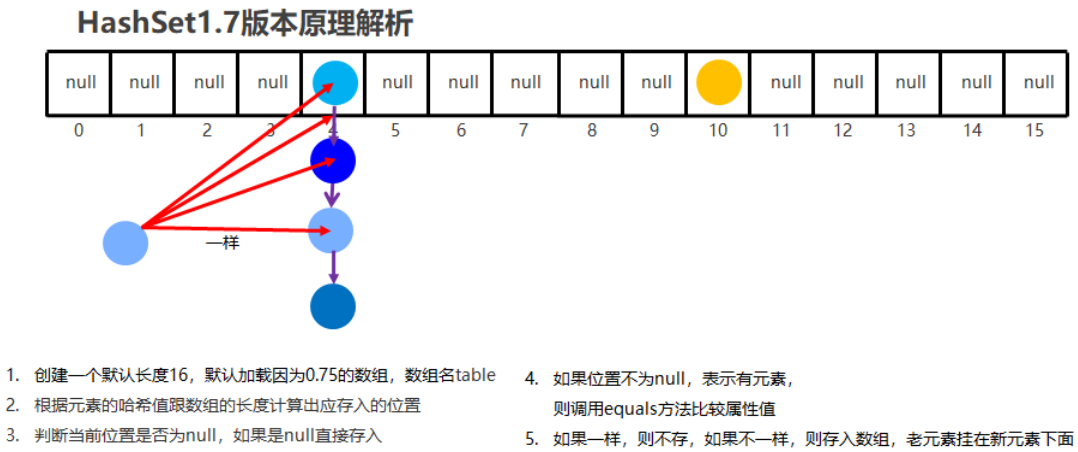
但是当前位置已经存在元素了，怎么将后来的数据存储到数组中呢?

这里需要使用类似链表的结构了，在当前位置上在画出来一个空间，然后将当前的对象数据保存到新划出来的空间中，在原来的空间中设置一个引用变量记录着新划分空间的地址，如果后面还有数据要存储当前空间，做法和上述相同。

3.4.2 哈希表图解

- JDK1.8以前

  数组 + 链表

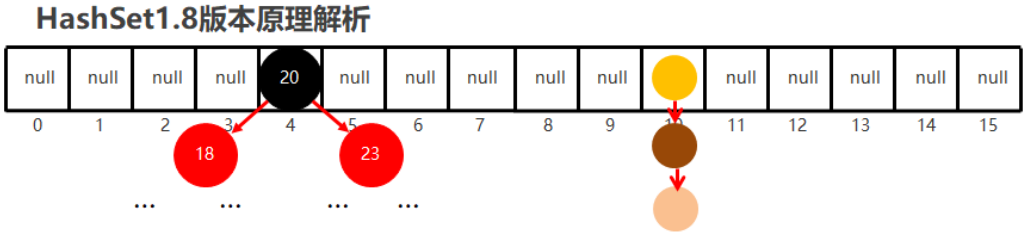


- JDK1.8以后
  - 节点个数少于等于8个

数组 + 链表

- 节点个数多于8个

  数组 + 红黑树



1. 创建一个默认长度16，默认加载因为0.75的数组，数组名table
2. 根据元素的哈希值跟数组的长度计算出应存入的位置
3. 判断当前位置是否为null，如果是null直接存入
4. 如果位置不为null，表示有元素，则调用equals方法比较属性值
5. 如果一样，则不存，如果不一样，则存入数组，老元素挂在新元素下面

3.5 疑问解答：

3.5.1 哈希表如何保证元素唯一?

答：哈希表保证元素唯一依赖两个方法：hashCode和equals。

哈希表底层其实还是一个数组，元素在存储的时候，会先通过hashCode算法结合数组长度得到一个索引。然后判断该索引位置是否有元素：如果没有，直接存储；如果有，再调用元素的equals方法比较是否相同：相同，(判定元素相同)直接舍弃；如果不同，也存储。

3.5.2 HashSet集合存储学生对象并遍历【应用】

- 案例需求
  - 创建一个存储学生对象的集合，存储多个学生对象，使用程序实现在控制台遍历该集合
  - 要求：学生对象的成员变量值相同，我们就认为是同一个对象
- 代码实现

  学生类

```
public class Student {
   private String name;
    private int age;

    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Student student = (Student) o;

        if (age != student.age) return false;
        return name != null ? name.equals(student.name) : student.name ==
null;
    }

    //如果不重写hashCode()方法，会导致hashSet集合中有重复元素出现
//    @Override
//    public int hashCode() {
//        int result = name != null ? name.hashCode() : 0;
//        result = 31 * result + age;
//        return result;
//    }
}
```

测试类

```
public class HashSetDemo02 {
    public static void main(String[] args) {
        //创建HashSet集合对象
        HashSet<Student> hs = new HashSet<Student>();

        //创建学生对象
        Student s1 = new Student("刘亦菲", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);

        Student s4 = new Student("王祖贤", 33);

        //把学生添加到集合
        hs.add(s1);
        hs.add(s2);
        hs.add(s3);
        hs.add(s4);

        //遍历集合(增强for)
        for (Student s : hs) {
            System.out.println(s.getName() + "," + s.getAge());
        }
    }
}
```

- 总结

  HashSet集合存储自定义类型元素,要想实现元素的唯一,要求必须重写hashCode方法和equals方法

3.5.3 为啥重写Hashcode和equals方法?

答：给哈希表中保存自定义学生对象时，由于每次创建的都是新的Student对象，而每个Student对象都有自己的唯一的内存地址，就是每个对象的内存地址都不相同。并且在给哈希表中存放这些对象的时候每个对象都需要调用自己从Object类中继承到的hashCode函数，而Object中的hashCode函数会根据每个对象自己内存地址计算每个对象哈希值，每个对象的内存地址不同，计算出来的哈希值也一定不同，最后就会导致生成的底层的数组的下标也不相同。如果下标不一样，根本就不会去调用equals比较是否是同一个元素，直接就会导致每个对象都可以正常的保存到哈希表中。

自己定义的对象，有自己所属的类，而这个类继承到了Object的hashCode函数，所以导致这个函数是根据对象内存地址计算哈希值，而我们更希望根据对象自己的特有数据(name和age的属性值)来计算哈希值。

3.5.4 问题：那么为什么我们之前向集合中保存String类的字符串对象时没有出现这种现象呢?

答： 因为在String类中已经复写了Object类中的hashCode()和equals()函数，所以存储字符串对象的时候，底层调用的hashCode()和equals()函数根本就不是Object类中的而是String类中已经复写好的函数。

也就是对于String类的对象调用自己的hashCode函数，根本就不是根据字符串对象的地址计算出来的哈希码值，而是根据字符串内容计算出来的。

对于equals函数，也是调用自己的函数，底层根本比较不是 this==obj,而是比较的是字符串对象的内容。

4.LinkedHashSet介绍(了解)


java.util
类 LinkedHashSet<E>

java.lang.Object
  └ java.util.AbstractCollection<E>
    └ java.util.AbstractSet<E>
      └ java.util.HashSet<E>
        └ java.util.LinkedHashSet<E>

类型参数：
E - 由此 set 维护的元素的类型

所有已实现的接口：
Serializable, Cloneable, Iterable<E>, Collection<E>, Set<E>

public class LinkedHashSet<E>
extends HashSet<E>
implements Set<E>, Cloneable, Serializable

具有可预知迭代顺序的 Set 接口的哈希表和链接列表实现。此实现与 HashSet 的不同之外在于，后者维护着一个运行于所有条目的双重链接列表。此链接列表定义了迭代顺序，即按照将元素插入到 set 中的顺序（插入顺序）进行迭代。注意，插入顺序不 受在 set 中重新插入的

4.1 LinkedHashSet定义和特点

LinkedHashSet集合：它的底层使用的链表+哈希表结构。它和HashSet集合的区别是LinkedHashSet是一个可以保证存取顺序的集合，并且LinkedHashSet集合中的元素也不能重复。

特点：

- 存取有序（底层有一个链接表）
- 保证元素的唯一（哈希表）
- 线程不安全，效率高

构造方法摘要

LinkedHashSet()	构造一个带默认初始容量 (16) 和加载因子 (0.75) 的新空链接哈希 set。
LinkedHashSet(Collection<? extends E> c)	构造一个与指定 collection 中的元素相同的新链接哈希 set。
LinkedHashSet(int initialCapacity)	构造一个带指定初始容量和默认加载因子 (0.75) 的新空链接哈希 set。
LinkedHashSet(int initialCapacity, float loadFactor)	构造一个带有指定初始容量和加载因子的新空链接哈希 set。

方法摘要

从类 java.util.HashSet 继承的方法

add, clear, clone, contains, isEmpty, iterator, remove, size

说明：LinkedHashSet集合没有自己的特有函数，所有的功能全部继承父类。

5.TreeSet集合

5.1TreeSet集合概述和特点【应用】

- 不可以存储重复元素
- 没有索引
- 可以将元素按照规则进行排序

  TreeSet()：根据其元素的自然排序进行排序

  TreeSet(Comparator comparator)：根据指定的比较器进行排序

5.2TreeSet集合基本使用【应用】

需求：存储Integer类型的整数并遍历

```
public class TreeSetDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Integer> ts = new TreeSet<Integer>();

        //添加元素
        ts.add(10);
        ts.add(40);
        ts.add(30);
        ts.add(50);
        ts.add(20);

        ts.add(30);

        //遍历集合
        for(Integer i : ts) {
```

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

5.3自然排序Comparable的使用【应用】

- 案例需求
  - 存储学生对象并遍历，创建TreeSet集合使用无参构造方法
  - 要求：按照年龄从小到大排序，年龄相同时，按照姓名的字母顺序排序
- 实现步骤
  1. 使用空参构造创建TreeSet集合
     - 用TreeSet集合存储自定义对象，无参构造方法使用的是自然排序对元素进行排序的
  2. 自定义的Student类实现Comparable接口
     - 自然排序，就是让元素所属的类实现Comparable接口，重写compareTo(T o)方法
  3. 重写接口中的compareTo方法
     - 重写方法时，一定要注意排序规则必须按照要求的主要条件和次要条件来写
- 代码实现

  学生类

```
public class Student implements Comparable<Student>{
    private String name;
    private int age;

    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
```

```
    }

    @Override
    public int compareTo(Student o) {
        //按照对象的年龄进行排序
        //主要判断条件：按照年龄从小到大排序
        int result = this.age - o.age;
        //次要判断条件：年龄相同时，按照姓名的字母顺序排序
        result = result == 0 ? this.name.compareTo(o.getName()) : result;
        return result;
    }
}
```

测试类

```
public class MyTreeSet02 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Student> ts = new TreeSet<>();
        //创建学生对象
        Student s1 = new Student("zhangsan",28);
        Student s2 = new Student("lisi",27);
        Student s3 = new Student("wangwu",29);
        Student s4 = new Student("zhaoliu",28);
        Student s5 = new Student("qianqi",30);
        //把学生添加到集合
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);
        ts.add(s4);
        ts.add(s5);
        //遍历集合
        for (Student student : ts) {
            System.out.println(student);
        }
    }
}
```

5.4比较器排序Comparator的使用【应用】

- 案例需求
  - 存储老师对象并遍历，创建TreeSet集合使用带参构造方法
  - 要求：按照年龄从小到大排序，年龄相同时，按照姓名的字母顺序排序
- 实现步骤
  - 用TreeSet集合存储自定义对象，带参构造方法使用的是比较器排序对元素进行排序的
  - 比较器排序，就是让集合构造方法接收Comparator的实现类对象，重写compare(T o1,T o2)方法
  - 重写方法时，一定要注意排序规则必须按照要求的主要条件和次要条件来写
- 代码实现

  老师类

```
public class Teacher {
    private String name;
    private int age;
```

```
    public Teacher() {
    }

    public Teacher(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Teacher{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

测试类

```
public class MyTreeSet04 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Teacher> ts = new TreeSet<>(new Comparator<Teacher>() {
            @Override
            public int compare(Teacher o1, Teacher o2) {
                //o1表示现在要存入的那个元素
                //o2表示已经存入到集合中的元素

                //主要条件
                int result = o1.getAge() - o2.getAge();
                //次要条件
                result = result == 0 ? o1.getName().compareTo(o2.getName())
: result;
                return result;
            }
        });
        //创建老师对象
        Teacher t1 = new Teacher("枫哥",23);
        Teacher t2 = new Teacher("罗老师",22);
        Teacher t3 = new Teacher("青木老师",24);
```

```
        Teacher t4 = new Teacher("韩清宗",24);
        //把老师添加到集合
        ts.add(t1);
        ts.add(t2);
        ts.add(t3);
        ts.add(t4);
        //遍历集合
        for (Teacher teacher : ts) {
            System.out.println(teacher);
        }
    }
}
```

5.5两种比较方式总结【理解】

- 两种比较方式小结
  - 自然排序: 自定义类实现Comparable接口,重写compareTo方法,根据返回值进行排序
  - 比较器排序: 创建TreeSet对象的时候传递Comparator的实现类对象,重写compare方法,根据返回值进行排序
  - 在使用的时候,默认使用自然排序,当自然排序不满足现在的需求时,必须使用比较器排序
- 两种方式中关于返回值的规则
  - 如果返回值为负数，表示当前存入的元素是较小值，存左边
  - 如果返回值为0，表示当前存入的元素跟集合中元素重复了，不存
  - 如果返回值为正数，表示当前存入的元素是较大值，存右边
-

5.6 成绩排序案例【应用】

- 案例需求
  - 用TreeSet集合存储多个学生信息(姓名,语文成绩,数学成绩,英语成绩),并遍历该集合
  - 要求: 按照总分从高到低出现
- 代码实现

  学生类

```
public class Student implements Comparable<Student> {
    private String name;
    private int chinese;
    private int math;
    private int english;

    public Student() {
    }

    public Student(String name, int chinese, int math, int english) {
        this.name = name;
        this.chinese = chinese;
        this.math = math;
        this.english = english;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public int getChinese() {
        return chinese;
    }

    public void setChinese(int chinese) {
        this.chinese = chinese;
    }

    public int getMath() {
        return math;
    }

    public void setMath(int math) {
        this.math = math;
    }

    public int getEnglish() {
        return english;
    }

    public void setEnglish(int english) {
        this.english = english;
    }

    public int getSum() {
        return this.chinese + this.math + this.english;
    }

    @Override
    public int compareTo(Student o) {
        // 主要条件: 按照总分进行排序
        int result = o.getSum() - this.getSum();
        // 次要条件: 如果总分一样,就按照语文成绩排序
        result = result == 0 ? o.getChinese() - this.getChinese() : result;
        // 如果语文成绩也一样,就按照数学成绩排序
        result = result == 0 ? o.getMath() - this.getMath() : result;
        // 如果总分一样,各科成绩也都一样,就按照姓名排序
        result = result == 0 ? o.getName().compareTo(this.getName()) :
result;
        return result;
    }
}
```

测试类

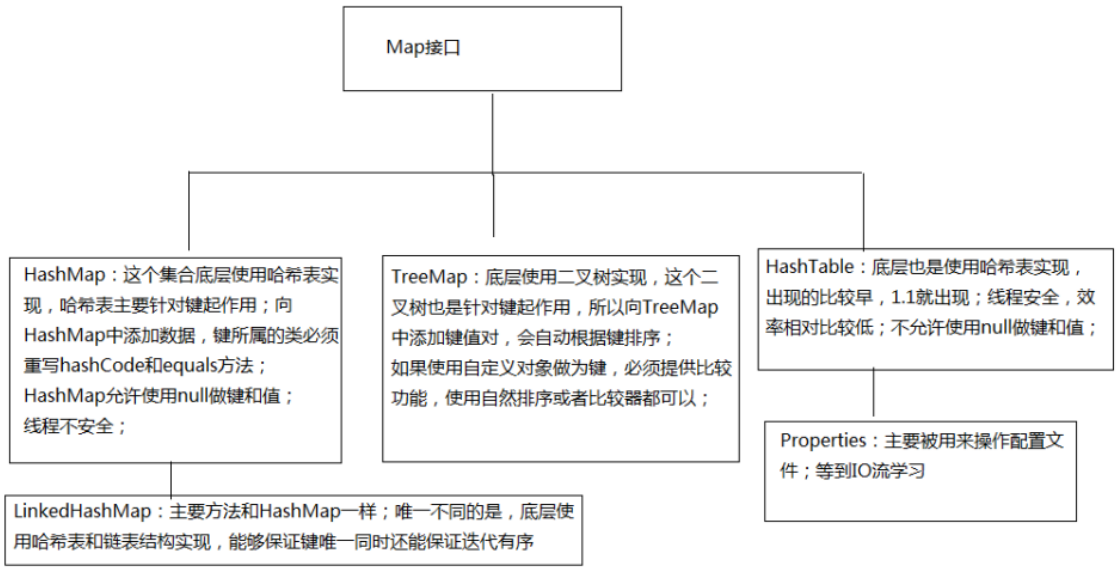
```
public class TreeSetDemo {
    public static void main(String[] args) {
        //创建TreeSet集合对象，通过比较器排序进行排序
        TreeSet<Student> ts = new TreeSet<Student>();
        //创建学生对象
        Student s1 = new Student("枫哥", 98, 100, 95);
        Student s2 = new Student("罗老师", 95, 95, 95);
        Student s3 = new Student("青木老师", 100, 93, 98);
```

```
        //把学生对象添加到集合
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);

        //遍历集合
        for (Student s : ts) {
            System.out.println(s.getName() + "," + s.getChinese() + "," +
s.getMath() + "," + s.getEnglish() + "," + s.getSum());
        }
    }
}
```

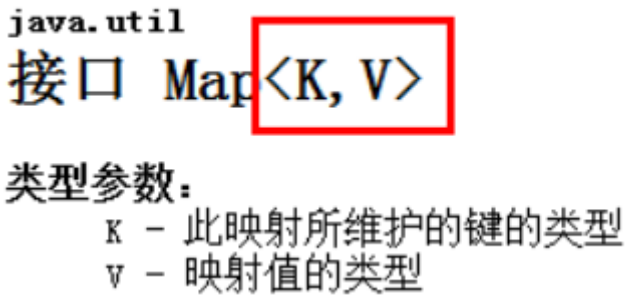
三.Map集合

1.1Map集合概述和特点【理解】



- Map集合概述

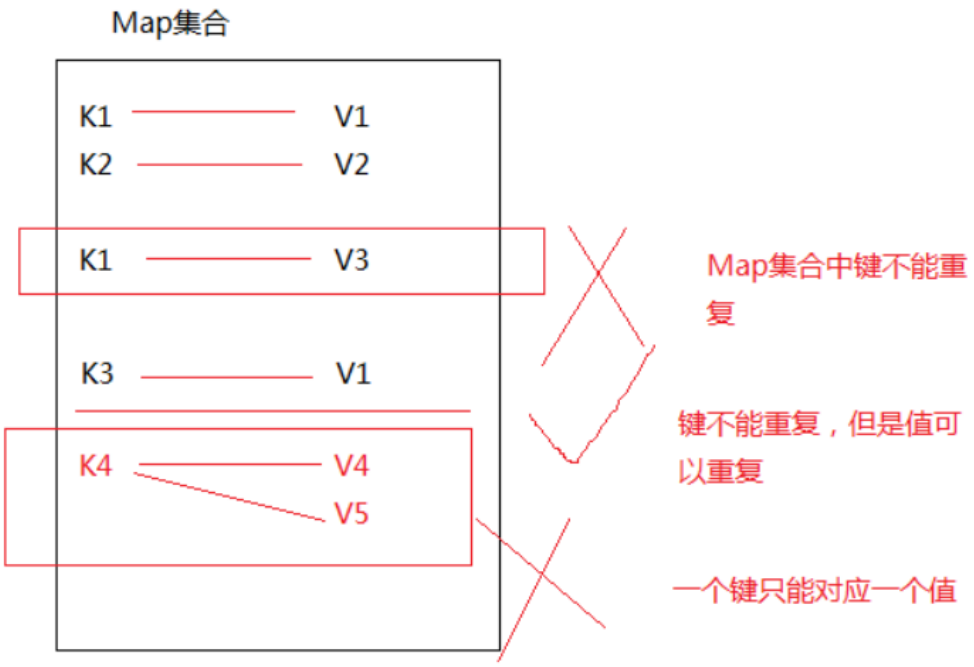
```
interface Map<K,V>   K：键的类型；V：值的类型
```




```
public interface Map<K,V>
```

将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。

- Map集合的特点
  - 双列集合,一个键对应一个值
  - 键不可以重复,值可以重复

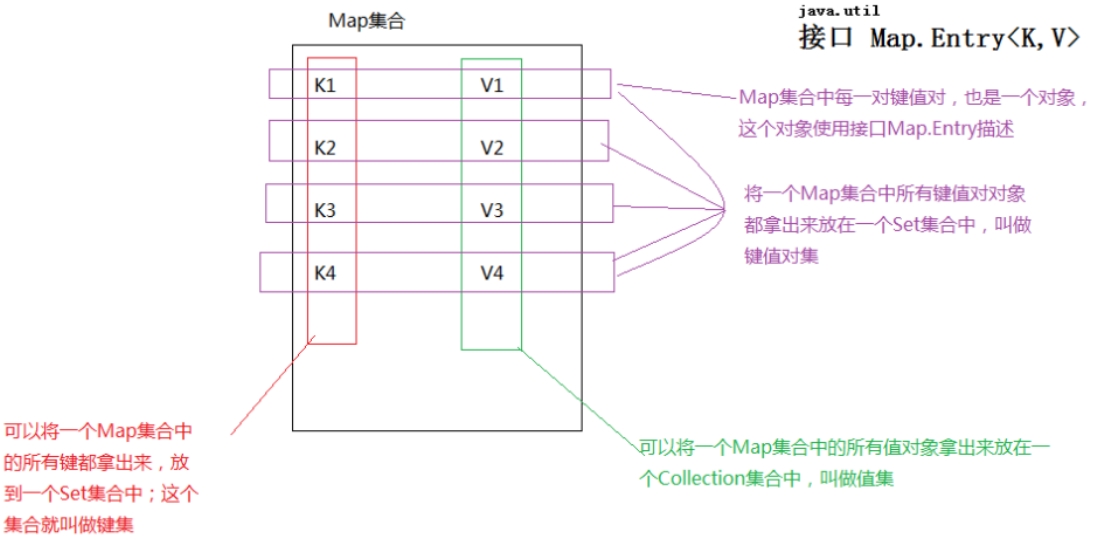


- Map集合的基本使用

```
public class MapDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String,String> map = new HashMap<String,String>();

        //V put(K key, V value) 将指定的值与该映射中的指定键相关联
        map.put("001","枫哥");
        map.put("002","罗老师");
        map.put("003","青木老师");
        map.put("003","韩清宗");

        //输出集合对象
        System.out.println(map);
    }
}
```



1.2Map集合的基本功能【应用】

- 方法介绍

方法名	说明
V put(K key,V value)	添加元素
V remove(Object key)	根据键删除键值对元素
void clear()	移除所有的键值对元素
boolean containsKey(Object key)	判断集合是否包含指定的键
boolean containsValue(Object value)	判断集合是否包含指定的值
boolean isEmpty()	判断集合是否为空
int size()	集合的长度，也就是集合中键值对的个数

- 示例代码

```
public class MapDemo02 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String,String> map = new HashMap<String,String>();

        //V put(K key,V value)：添加元素
        map.put("令狐冲","任盈盈");
        map.put("郭靖","黄蓉");
        map.put("杨过","小龙女");

        //V remove(Object key)：根据键删除键值对元素
//        System.out.println(map.remove("郭靖"));
//        System.out.println(map.remove("郭襄"));

        //void clear()：移除所有的键值对元素
//        map.clear();

        //boolean containsKey(Object key)：判断集合是否包含指定的键
//        System.out.println(map.containsKey("郭靖"));
//        System.out.println(map.containsKey("郭襄"));

        //boolean isEmpty()：判断集合是否为空
//        System.out.println(map.isEmpty());

        //int size()：集合的长度，也就是集合中键值对的个数
        System.out.println(map.size());

        //输出集合对象
        System.out.println(map);
    }
}
```

1.3Map集合的获取功能【应用】

- 方法介绍

方法名	说明
V get(Object key)	根据键获取值
Set keySet()	获取所有键的集合
Collection values()	获取所有值的集合
Set<Map.Entry<K,V>> entrySet()	获取所有键值对对象的集合

- 示例代码

```
public class MapDemo03 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("乔峰", "阿朱");
        map.put("段誉", "王语嫣");
        map.put("虚竹", "梦姑");

        //V get(Object key):根据键获取值
//        System.out.println(map.get("乔峰"));
//        System.out.println(map.get("虚竹"));

        //Set<K> keySet():获取所有键的集合
//        Set<String> keySet = map.keySet();
//        for(String key : keySet) {
//            System.out.println(key);
//        }

        //Collection<V> values():获取所有值的集合
        Collection<String> values = map.values();
        for(String value : values) {
            System.out.println(value);
        }
    }
}
```

1.4Map集合的遍历之按键遍历【应用】

- 遍历思路
  - 我们刚才存储的元素都是成对出现的，所以我们把Map看成是一个夫妻对的集合
    - 把所有的丈夫给集中起来
    - 遍历丈夫的集合，获取到每一个丈夫
    - 根据丈夫去找对应的妻子
- 步骤分析
  - 获取所有键的集合。用keySet()方法实现
  - 遍历键的集合，获取到每一个键。用增强for实现
  - 根据键去找值。用get(Object key)方法实现
- 代码实现

```
public class MapDemo01 {
    public static void main(String[] args) {
```

```
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("乔峰", "阿朱");
        map.put("段誉", "王语嫣");
        map.put("虚竹", "梦姑");

        //获取所有键的集合。用keySet()方法实现
        Set<String> keySet = map.keySet();
        //遍历键的集合，获取到每一个键。用增强for实现
        for (String key : keySet) {
            //根据键去找值。用get(Object key)方法实现
            String value = map.get(key);
            System.out.println(key + "," + value);
        }
    }
}
```

1.5Map集合的遍历之Entry键值对【应用】

- 遍历思路
  - 我们刚才存储的元素都是成对出现的，所以我们把Map看成是一个夫妻对的集合
    - 获取所有结婚证的集合
    - 遍历结婚证的集合，得到每一个结婚证
    - 根据结婚证获取丈夫和妻子
- 步骤分析
  - 获取所有键值对对象的集合
    - Set<Map.Entry<K,V>> entrySet()：获取所有键值对对象的集合
  - 遍历键值对对象的集合，得到每一个键值对对象
    - 用增强for实现，得到每一个Map.Entry
  - 根据键值对对象获取键和值
    - 用getKey()得到键
    - 用getValue()得到值
- 图解分析
- 代码实现

```
public class MapDemo02 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("乔峰", "阿朱");
        map.put("段誉", "王语嫣");
        map.put("虚竹", "梦姑");

        //获取所有键值对对象的集合
        Set<Map.Entry<String, String>> entrySet = map.entrySet();
        //遍历键值对对象的集合，得到每一个键值对对象
        for (Map.Entry<String, String> me : entrySet) {
            //根据键值对对象获取键和值
            String key = me.getKey();
```

```
            String value = me.getValue();
            System.out.println(key + "," + value);
        }
    }
}
```

2.HashMap集合

2.1HashMap集合概述和特点【理解】

- HashMap底层是哈希表结构的
- 依赖hashCode方法和equals方法保证键的唯一
- 如果键要存储的是自定义对象，需要重写hashCode和equals方法

2.2HashMap集合应用案例【应用】

- 案例需求
  - 创建一个HashMap集合，键是学生对象(Student)，值是居住地 (String)。存储多个元素，并遍历。
  - 要求保证键的唯一性：如果学生对象的成员变量值相同，我们就认为是同一个对象
- 代码实现

  学生类

```
public class Student {
    private String name;
    private int age;

    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
```

```
        Student student = (Student) o;

        if (age != student.age) return false;
        return name != null ? name.equals(student.name) : student.name ==
null;
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }
}
```

测试类

```
public class HashMapDemo {
    public static void main(String[] args) {
        //创建HashMap集合对象
        HashMap<Student, String> hm = new HashMap<Student, String>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);
        Student s4 = new Student("王祖贤", 33);

        //把学生添加到集合
        hm.put(s1, "上海");
        hm.put(s2, "武汉");
        hm.put(s3, "安徽");
        hm.put(s4, "北京");

        //遍历集合
        Set<Student> keySet = hm.keySet();
        for (Student key : keySet) {
            String value = hm.get(key);
            System.out.println(key.getName() + "," + key.getAge() + "," +
value);
        }
    }
}
```

2.3 HashMap和Hashtable的区别?

```
/**
 * @author IT枫斗者
 * @ClassName Test.java
 * @From www.javatiaozao.com
 * Hashtable和HashMap的区别?
 *  Hashtable:线程安全，效率低。不允许null键和null值
 *  HashMap:线程不安全，效率高。允许null键和null值
 *  Hashtable的方法是Synchronize的，而HashMap不是
 */
```

```
public class Test {
    public static void main(String[] args)
    {
        //创建一个HashMap集合
        HashMap<String, String> hm = new HashMap<String, String>();
        hm.put("001", "hello");
        hm.put(null, "world");
        hm.put("java", null);
        System.out.println(hm);//打印结果 {null=world, 001=hello, java=null}

        //创建一个Hashtable集合
        Hashtable<String, String> ht = new Hashtable<String, String>();
        ht.put(null, "world"); //NullPointerException
        ht.put("java", null); // NullPointerException
        System.out.println(ht);
    }

}
```

3.TreeMap集合

3.1TreeMap集合概述和特点【理解】

- TreeMap底层是红黑树结构
- 依赖自然排序或者比较器排序,对键进行排序
- 如果键存储的是自定义对象,需要实现Comparable接口或者在创建TreeMap对象时候给出比较器排序规则

3.2TreeMap集合应用案例一【应用】

- 案例需求
  - 创建一个TreeMap集合,键是学生对象(Student),值是籍贯(String),学生属性姓名和年龄,按照年龄进行排序并遍历
  - 要求按照学生的年龄进行排序,如果年龄相同则按照姓名进行排序
- 代码实现

  学生类

```
public class Student implements Comparable<Student>{
    private String name;
    private int age;

    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Student o) {
        //按照年龄进行排序
        int result = o.getAge() - this.getAge();
        //次要条件，按照姓名排序。
        result = result == 0 ? o.getName().compareTo(this.getName()) :
result;
        return result;
    }
}
```

测试类

```
public class Test1 {
    public static void main(String[] args) {
        // 创建TreeMap集合对象
        TreeMap<Student,String> tm = new TreeMap<>();

        // 创建学生对象
        Student s1 = new Student("张三",23);
        Student s2 = new Student("李四",22);
        Student s3 = new Student("王五",22);

        // 将学生对象添加到TreeMap集合中
        tm.put(s1,"上海");
        tm.put(s2,"北京");
        tm.put(s3,"杭州");

        // 遍历TreeMap集合,打印每个学生的信息
        tm.forEach(
                (Student key, String value)->{
                    System.out.println(key + "---" + value);
                }
        );
    }
}
```

3.3TreeMap集合应用案例二【应用】

- 案例需求
  - 给定一个字符串,要求统计字符串中每个字符出现的次数。
  - 举例: 给定字符串是“aababcabcdabcde”,在控制台输出: “a(5)b(4)c(3)d(2)e(1)”
- 代码实现

```
public class Test2 {
    public static void main(String[] args) {
        // 给定字符串
        String s = "aababcabcdabcde";
        // 创建TreeMap集合对象,键是Character,值是Integer
        TreeMap<Character,Integer> tm = new TreeMap<>();

        //遍历字符串，得到每一个字符
        for (int i = 0; i < s.length(); i++) {
            //c依次表示字符串中的每一个字符
            char c = s.charAt(i);
            // 判断当前遍历到的字符是否在集合中出现过
            if(!tm.containsKey(c)){
                //表示当前字符是第一次出现。
                tm.put(c,1);
            }else{
                //存在，表示当前字符已经出现过了
                //先获取这个字符已经出现的次数
                Integer count = tm.get(c);
                //自增，表示这个字符又出现了依次
                count++;
                //将自增后的结果再次添加到集合中。
                tm.put(c,count);
            }
        }
        //  a（5）b（4）c（3）d（2）e（1）
        //System.out.println(tm);
        tm.forEach(
                (Character key,Integer value)->{
                    System.out.print(key + "（" + value + "）");
                }
        );
    }
}
```

如何选择集合类?

Collection(单列集合)

List(有序,可重复)

ArrayList

底层数据结构是数组,查询快,增删慢

线程不安全,效率高

Vector

底层数据结构是数组,查询快,增删慢

线程安全,效率低

LinkedList

底层数据结构是链表,查询慢,增删快

线程不安全,效率高

Set(无序,唯一)

HashSet

底层数据结构是哈希表。

哈希表依赖两个方法：hashCode()和equals()

执行顺序：

首先判断hashCode()值是否相同

是：继续执行equals(),看其返回值

是true:说明元素重复，不添加

是false:就直接添加到集合

否：就直接添加到集合

最终：

自动生成hashCode()和equals()即可

LinkedHashSet

底层数据结构由链表和哈希表组成。

由链表保证元素有序。

由哈希表保证元素唯一。

TreeSet

底层数据结构是红黑树。(是一种自平衡的二叉树)

如何保证元素唯一性呢?

根据比较的返回值是否是0来决定

如何保证元素的排序呢?

两种方式

自然排序(元素具备比较性)

让元素所属的类实现Comparable接口

比较器排序(集合具备比较性)

让集合接收一个Comparator的实现类对象

Map(双列集合)

A:Map集合的数据结构仅仅针对键有效，与值无关。

B:存储的是键值对形式的元素，键唯一，值可重复。

HashMap

底层数据结构是哈希表。线程不安全，效率高

哈希表依赖两个方法：hashCode()和equals()

执行顺序：

首先判断hashCode()值是否相同

是：继续执行equals(),看其返回值

是true:说明元素重复，不添加

是false:就直接添加到集合

否：就直接添加到集合

最终：

自动生成hashCode()和equals()即可

LinkedHashMap

底层数据结构由链表和哈希表组成。

由链表保证元素有序。

由哈希表保证元素唯一。

Hashtable

底层数据结构是哈希表。线程安全，效率低

哈希表依赖两个方法：hashCode()和equals()

执行顺序：

首先判断hashCode()值是否相同

是：继续执行equals(),看其返回值

是true:说明元素重复，不添加

是false:就直接添加到集合

否：就直接添加到集合

最终：

自动生成hashCode()和equals()即可

TreeMap

底层数据结构是红黑树。(是一种自平衡的二叉树)

如何保证元素唯一性呢?

根据比较的返回值是否是0来决定

如何保证元素的排序呢?

两种方式

自然排序(元素具备比较性)

让元素所属的类实现Comparable接口

比较器排序(集合具备比较性)

让集合接收一个Comparator的实现类对象

到底使用那种集合

看需求。

是否是键值对象形式:

是：Map

键是否需要排序:

是：TreeMap

否：HashMap

不知道，就使用HashMap。

否：Collection

元素是否唯一:

是：Set

元素是否需要排序:

是：TreeSet

否：HashSet

不知道，就使用HashSet

否：List

要安全吗:

是：Vector(其实我们也不用它,后面我们讲解了多线程以后，我在给你回顾用谁)

否：ArrayList或者LinkedList

增删多：LinkedList

查询多：ArrayList

不知道，就使用ArrayList

不知道，就使用ArrayList

实际开发中，使用最多的是：

ArrayList；HashSet；HashMap；