# 一、数组

1、数组复习

* 数组是存储同一种数据类型的集合容器

2、数组的特点：

* 只能存储同一种数据类型的数据
* 一旦初始化，长度固定
* 数组中的元素与元素之间的内存地址是连续的

3、特殊的数组：Object类型的数组，可以存储任意类型的数据

# 二、集合

1、什么是集合？

* 集合是存储对象数据的集合容器

2、集合比数组的优势

* 集合可以存储任意类型的对象数据，数组只能存储同一种类型的数据
* 集合的长度是会发生变化的，数组的长度是固定的

# 三、Collection

----| Collection**单利集合**的根接口

--------| List接口：实现了List接口的集合类，具备的特点：**有序、可重复**

--------| Set接口：实现类Set接口的集合类，具备的特点：**无序，不重复**

1、Collection的常用方法

* 增加

boolean add(E e):添加成功返回true，失败返回false

boolean addAll(Collection c):把一个集合中的所有元素添加到另一个集合中去

* 删除

void clear():删除此集合中的所有元素

boolean remove(Object o):删除集合中指定元素的实例o

boolean removeAll(Collection c):删除指定集合中与集合c交集的所有元素

boolean retainAll(Collection c):保留指定集合与c集合交集的所有元素，删除其他一切元素

* 查看

int size():返回此集合中的元素数

* 判断

boolean isEmpty():判断集合是否为空。不包含任何元素返回true，否则返回false

boolean contains(Object o):如果该集合包含指定的元素o，返回true

containsAll(Collection<?> c):如果该集合包含指定集合c中的所有元素，返回true

* 迭代

Object[] toArray():返回一个包含此集合中所有元素的数组

Iterator<E> iterator():返回该集合中的元素的迭代器

## 1、toArray()：返回一个包含此集合中所有元素的数组

|  |
| --- |
| **package** collectionss;  **import** java.util.ArrayList; **import** java.util.Arrays; **import** java.util.Collection; **public class** Dome2 {  **public static void** main(String[] args) {  Collection c = **new** ArrayList();  c.add(**"小白"**);  c.add(**"小黑"**);  c.add(**"小红"**);  c.add(**"小兰"**);  c.add(**"小紫"**);  Object[] objects = c.toArray();  System.***out***.print(**"由集合c的toArray()方法转换成的数组中所包含的元素有："** + Arrays.*toString*(objects));  }  *//运行结果：由集合c的toArray()方法转换成的数组中所包含的元素有：[小白, 小黑, 小红, 小兰, 小紫]* } |

## 2、Iterator迭代器



1. 获取Iterator对象

|  |
| --- |
| Collection c = **new** ArrayList();  Iterator iterator = c.iterator();  // 疑问？Iterator()方法返回的是一个接口对象，为什么能够调用方法呢？   * iterator()方法实际上返回的是Iterator接口的实现类对象 |

1. Iterator的方法

* boolean hasNext()： 如果迭代具有更多元素，则返回 true 。
  + 问是否有元素可以遍历，有则返回true，没有返回false
  + 判断当前指针指向的位置是否有元素，有返回true，没有返回false
* E next()：返回迭代中的下一个元素。
  + 若当前指针指向的位置存在元素，则获取该元素，并将指针指向下一个元素位置l
  + 获取该指针位置的元素
* void remove()：从底层集合中删除此迭代器返回的最后一个元素（可选操作）。
  + // 删除 指针指向的元素（清空集合的元素）

|  |
| --- |
| **package** collectionss;  **import** java.util.ArrayList; **import** java.util.Collection; **import** java.util.Iterator; **public class** Dome2 {  **public static void** main(String[] args) {  Collection c = **new** ArrayList();  c.add(**"小白"**);  c.add(**"小哄"**);  c.add(**"小子"**);  c.add(**"小栏"**);  c.add(**"小率"**);Iterator iterator = c.iterator();  System.***out***.print(**"集合中的元素包括："**);**while** (iterator.hasNext()) {System.***out***.print(iterator.next() + **" "**);iterator.remove();  }  System.***out***.println();  System.***out***.println(**"集合中是否还存在元素："** + c);   }  *// 运行结果：集合中的元素包括：小白 小哄 小子 小栏 小率* } |

# 四.List

* List 有序集合（有序序列）。List集合中的元素是有序的，可重复的

## 1、List的常用方法

1. 添加

* add(E element):将元素添加到集合的末尾位置
* add(int index, E element):将元素放入list集合的指定位置
* addAll(Collection<? extends E> c):将指定集合添加到集合的末尾位置
* addAll(int index, Collection<? extends E> c):将指定集合添加到集合的指定位置

1. 获取：

* get(int index):获取指定位置的元素
* indexOf(Object o)：获取集合中指定元素第一次出现的下标，如果没有该元素，则返回-1
* lastIndexOf(Object o)：获取指定元素在集合中最后一次出现的下标，如果没有该元素，则返回-1
* subList(int fromIndex, int toIndex)：获取集合中从索引值from(包含)到toIndex(不包含)的元素集

1. 修改：

* set(int index, E element)：更改集合指定位置的元素的内容

## 2、List集合的迭代器：listIterator()

* listIterator()返回List接口（有序集合）特有的迭代器。
* 以下是ListIterator独有的方法：
  + hasPrevious():判断是否存在上一个元素
  + previous():指针先向上移动一个单位，再获取指针当前指向的元素
  + add(E e):向当前指针指向的位置插入元素
  + set(E e)：替代迭代器最后一次返回的元素

## 3、迭代器需要注意的事项

（1）迭代器在迭代过程中，不允许使用集合对象改变集合中的元素个数，如果需要添加或删除，只能使用迭代器的方法进行操作。（就是说 如果想使用集合对象改变集合的个数，那么集合对象操作集合元素个数的语句下边，不允许出现迭代器的操作。不管是不是在循环中，都不能将迭代器操作语句放在集合操作语句下边）

（2）如果使用过了集合对象改变集合中元素个数那么就会出现ConcurrentModificationException异常。

（3）迭代元素的过程指：迭代器创建到使用结束的时间

# 五、ArrayList的原理

## 1、笔试题目：

* 使用ArrayList无参的构造函数创建一个 对象时， 默认的容量是多少? 如果长度不够使用时又自增增长多少？
* ArrayList 底层是维护了一个Object数组实现的，当使用无参构造函数创建对象时，使用ArrayList维护的默认初始容量10，当需要增长时，每次自动增长0.5倍。

|  |
| --- |
| *源码解析：*  */\*\*Default initial capacity.默认初始容量\*/  private static final int DEFAULT\_CAPACITY = 10;*  **private void** grow(**int** minCapacity) {  *// overflow-conscious code 容量不够时溢出的代码* **int** oldCapacity = **elementData**.**length**;  **int** newCapacity = oldCapacity + (oldCapacity >> 1);// 新长度等于旧长度的1.5倍（自动增长0.5倍）  **if** (newCapacity - minCapacity < 0)  newCapacity = minCapacity;  **if** (newCapacity - ***MAX\_ARRAY\_SIZE*** > 0)  newCapacity = *hugeCapacity*(minCapacity);  *// minCapacity is usually close to size, so this is a win:* **elementData** = Arrays.*copyOf*(**elementData**, newCapacity); } |

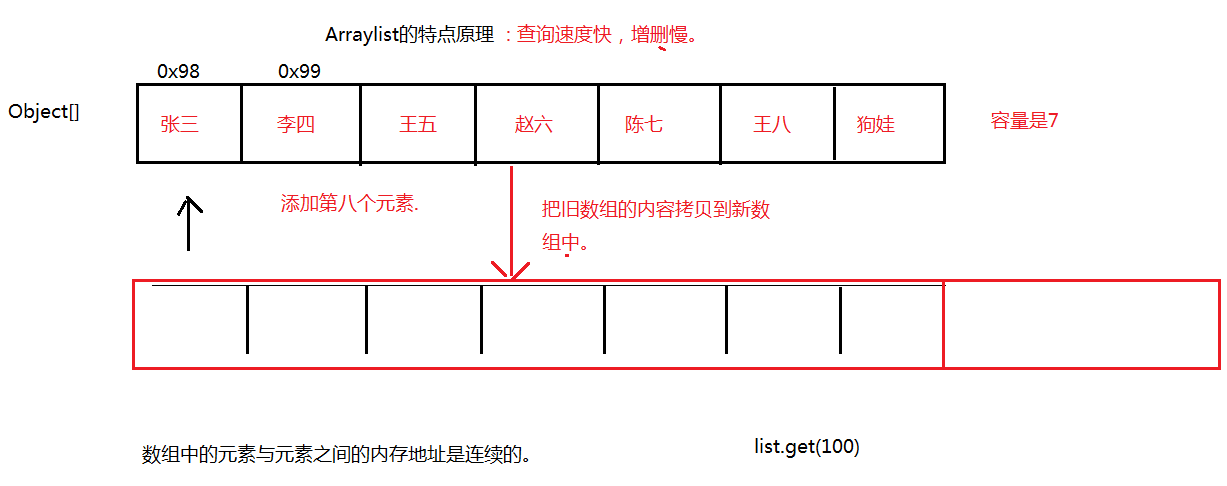
## 2、ArrayList的底层实现

* ArrayList的底层是维护了一个Object数组实现的。
* 它的特点是查询速度快，增删慢

## 3、ArrayList的方法

* **void** ensureCapacity(**int** minCapacity)：指定ArrayList对象的容量，一般不使用，因为ArrayList提供了初始化容量的构造方法，我们习惯使用构造方法来初始化数组容量。
* **void** trimToSize()：修改这个 ArrayList实例的容量是列表的当前大小

## 4、ArrayList的特点原理（查询快，增删慢）



1. 因为ArrayList底层维护的是一个Object数组，数组的元素与元素之间的内存地址是连续的，当要查询一个元素的时候，只要知道它的索引值，那么就能快速的查找到这个元素。
2. 当要增加一个元素时，如果它的容量是7，此时数组中正好有7个元素，当要插入第八个元素时，ArrayList会复制这个旧数组中的所有元素放入新数组中，并将容量扩充0.5倍，然后将第八个元素插入进去。
3. 当要删除ArrayList实例中的元素时，ArrayList会将被删除元素后边的所有元素往前移动一位（这是非常高的工作量）

## 5、什么时候适合使用ArrayList？

* 如果目前的数据查询比较多，增删比较少时，适合使用ArrayList。比如 图书馆

# 六、LinkedList的原理

* LinkedList底层是使用链表数据结构实现的，具有增删快，查询速度慢的特点

1. LinkedList特有的方法

* addFirst(E e) :将元素添加到集合的首位
* addLast(E e) :将元素添加到集合的末尾
* getFirst():获取集合的首位元素
* getLast():获取集合的末位元素
* removeFirst():删除集合的首位元素
* removeLast():删除集合的末位元素

# 七、数据结构（LinkedList的数据结构）

## 1、栈（jdk1.6引进）

* 主要是用于实现堆栈数据结构的存储方式
* 特点是 **先进后出**（杯子，先进的在最底层，后进的在它上边，想要取出，需要先取出后进的，再取出之前存进去的）
* 主要方法：
  + push(E e):将元素插入到集合的首位中去
  + pop():移除并返回集合的首位元素

## 2、队列（双端队列，jdk1.5引进）

* 主要是为了让你们可以使用LinkedList模拟队列数据结构的存储方式。
* 队列的特点是 **先进先出**（排队，先就位的先出列，排队进行）
* 主要方法：
  + Offer(E e)：将元素添加到集合的末尾处

## 3、返回逆序的迭代器对象

* descendingIterator():获取逆序迭代器对象

# 八.Vector

1、Vector类实现了可扩展的对象数组，像数组一样。但是Vector的大小可以根据需要实现增长或减小。

1. 面试题：Vector类 与ArrayList 类的区别
   1. 相同点：
      * 底层都是维护了一个 Object 数组实现的

b) 不同点：

* + - Vector 是线程安全的，效率低；ArrayList 是线程不安全的，操作效率高。
    - Vector 是 jdk1.0 开始使用的，ArrayList 是 jdk1.2 开始使用的。

3、Vector 的常用方法：

添加：

addElement(E e)：将元素添加到集合的末尾处

迭代器：

elements()：获取迭代器

hasMoreElements()：判断是否存在元素

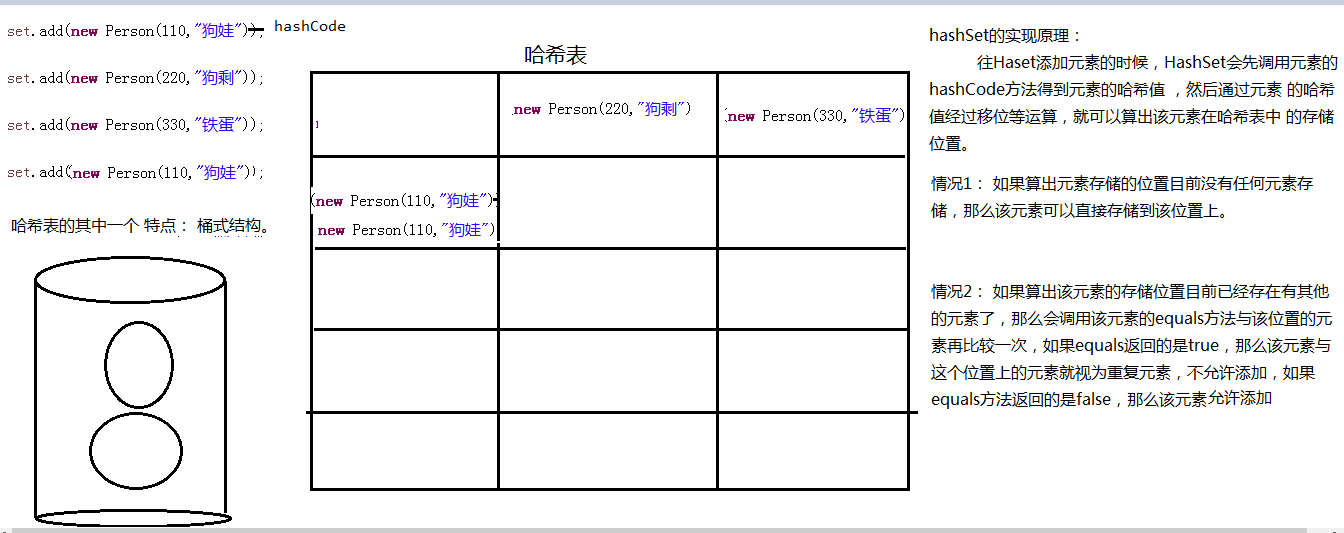
nextElement():获取元素

# 九．Set接口

1、实现了Set接口的集合类的特点：**无序、不可重复。**

2、无序：添加元素的顺序，与取出元素的顺序无关（进与出顺序不一致）

# 十、HashSet类



## 1、HashSet实现原理：

（1）底层实现

* HashSet 的底层是通过 **哈希表** 来实现的

（2）HashSet向集合中添加元素的原理

* 向HashSet集合中添加元素时，首先调用该元素的hashCode()方法，得到该元素的哈希值，然后通过位移等运算，计算出该元素应该存储的位置：
  + 情况一：如果该位置不存在任何元素，那么可以直接将该元素添加到该位置。
  + 情况二：如果该位置已存在元素，那么就调用该元素的equals()方法与该位置的元素再次进行比较：
    - 如果返回true，将该元素与该位置的元素视为同一元素，不允许添加
    - 如果返回false，该元素与该位置的元素视为不同元素，允许添加
* 哈希表的特点之一是：桶式结构
  + 哈希表的每个位置就像放了一个桶一样，可以在该位置存放多个不同的元素。

## 2、hashCode()

注意：

* hashCode()默认表示的是内存地址，
* 但是String已经重写了hashCode()方法，比较的是字符串的内容是否一致。

如果两个字符串的内容一致，那么这两个字符串的hashCode值一定也一致。

# 十一、TreeSet类

1、TreeSet类的存储，使用元素的自然顺序对元素进行排序存储，或者根据创建Set提供的Comparator接口的实现方法compareTo(T o)进行排序，具体取决于使用的构造方法

## 2、TreeSet存储元素的注意事项

* 如果元素本身具备了自然顺序的特性，那么就使用元素本身的自然顺序特性进行排序存储。
* 如果元素本身不具备自然顺序的特性，那么元素所属的类需要实现Comparable接口，在接口的compareTo(T o)方法中实现排序的业务；否则会出现异常：*set.Persons cannot be cast to java.lang.Comparable*
* 如果compareTo(T o)方法的返回结果是0，那么就被视为重复元素，不允许添加（TreeSet与HashCode和equals方法是没有任何关系的，它依赖的是compareTo方法）
* 往TreeSet添加元素时，如果元素自身没有自然顺序属性，元素所属类也没有实现comparable接口，那么在构造TreeSet对象时，就需要传入一个比较器
* 让TreeSet添加元素是，如果元素本身没有自然顺序属性，元素所属类实现了comparable接口，构建TreeSet接口时也传入了比较器，那么是以比较器的比较规则优先使用的。

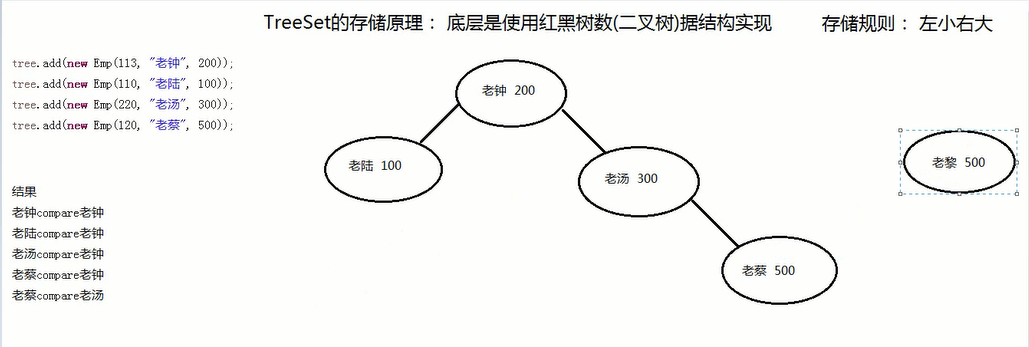
2.1自定义比较器：

* 自定义一个类，实现Comparator接口即可，将比较规则写在compare方法内

|  |
| --- |
| **// 自定义比较器**  **class** MyComparator **implements** Comparator {  @Override  **public int** compare(Object o1, Object o2) {  Persons p1 = (Persons) o1;  Persons p2 = (Persons) o2;  **return** p1.getSalary() - p2.getSalary();  } }  // 使用自定义比较器：  MyComparator myComparator = **new** MyComparator(); TreeSet treeSet = **new** TreeSet(myComparator); |

|  |
| --- |
| **package** set;  **import** java.util.TreeSet; **public class** Dome3 {  /\* TreeSet添加注意事项：  如果元素本身存在自然顺序特性，那么就按照元素自然顺序的特性排序存储。  如果元素本身不存在自然顺序特性，那么就需要实现java.lang.Comparable 接口，并实现接口的 int compareTo(T o)方法，并在该方法中实现排序业务。  否则出现异常：set.Persons cannot be cast to java.lang.Comparable \*/ **public static void** main(String[] args) {  TreeSet treeSet = **new** TreeSet();// 创建TreeSet对象  System.***out***.println(treeSet.add(**new** Persons(10340, **"小哄"**, **"人力"**)));  System.***out***.println(treeSet.add(**new** Persons(10000, **"小白"**, **"java"**)));  System.***out***.println(treeSet.add(**new** Persons(12340, **"小黑"**, **"会计"**)));  System.***out***.println(treeSet.add(**new** Persons(1230, **"小哈"**, **"设计"**)));   System.***out***.println(treeSet);  } }  // 实现Comparable接口 **class** Persons **implements** Comparable {  **private int salary**;  **private** String **name**;  **private** String **work**;   // 重写排序规则（重写compareTo方法）**@Override  public int compareTo(Object o) {  Persons p = (Persons) o;  *// 按照工资进行排序* return this.salary - p.salary;  }**  **public** Persons() {  }   **public int** getSalary() {  **return salary**;  }   **public void** setSalary(**int** salary) {  **this**.**salary** = salary;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public** String getWork() {  **return work**;  }   **public void** setWork(String work) {  **this**.**work** = work;  }   **public** Persons(**int** salary, String name, String work) {  **this**.**salary** = salary;  **this**.**name** = name;  **this**.**work** = work;  }   @Override  **public** String toString() {  **return "{ 工资："** + **this**.**salary** + **" 姓名："** + **this**.**name** + **" 岗位："** + **this**.**work** + **"}"**;  } } |

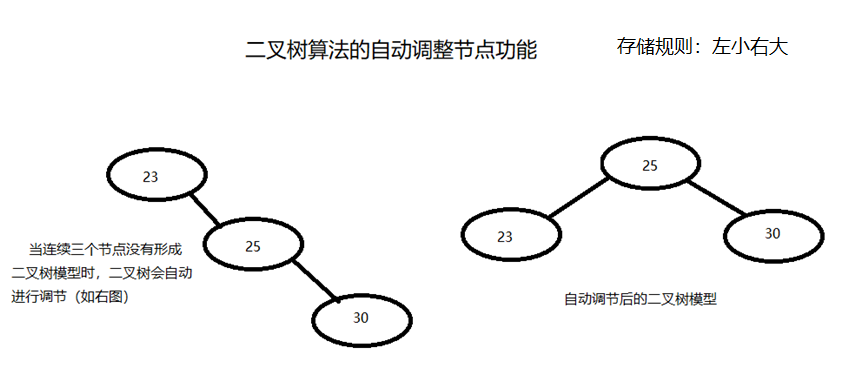
## 3、TreeSet的实现原理



## 4、二叉树算法

（1）自动调整节点

* 当连续三个节点没有形成二叉树算法时，二叉树会进行自动调整，将中间位置的元素放到树根位置，将第一个位置的元素放到左边，将第三个元素放到右边。（如图）



## 5、TreeSet比较String字符串

* TreeSet是可以比较字符串的，应为字符串实现了comparable接口。

5.1 字符串的比较规则

* 情况一：对应位置有不同的字符出现，比较的就是对应位置不同的字符。
* 情况二：对应位置上的字符都一样，那么比较的就是字符串的长度。

# 十二、泛型（jdk1.5使用的新特性）

## 1. 泛型的好处

将运行时异常提前至编译时。

避免了无畏的强制类型转换。

## 2. 泛型的格式

* **<String>**: 表示该类型只能存储字符串类型的数据

## 3、泛型格式注意事项：

* 泛型没有多态的概念，只能两边都写或只写一边。
* 推荐两边都写，只写一边是为了新老系统的兼容性问题。
  + 推荐： **TreeSet<String> ts = new TreeSet<String>();**
  + 兼容（老版本）：
    - * TreeSet<String> ts = new TreeSet();
      * TreeSet ts = new TreeSet<String>();

## 4、自定义泛型

* 自定义泛型就是一个数据类型的 占位符 或者 一个数据类型的 变量
* 例如：
  + - <T> <E>

## 5、方法泛型

* 格式

**修饰符 <自定义的泛型> 返回值类型 (使用自定义泛型 …){**

**业务代码……**

**}**

* <自定义泛型>:
  + - 代表数据类型的占位符或变量
    - <T> Type 类型
    - <E> Element 元素
* 泛型使用的注意事项
* 在泛型中不能使用基本数据类型，如果需要使用基本数据类型，那么就使用基本数据类型对应的包装类型。
* 基本数据类型的包装类

byte -------- Byte

short -------- Short

int -------- Integer

long -------- Long

double -------- Double

float -------- Float

boolean -------- Boolean

char -------- Character

* + 1. 在方法上自定义泛型，它的具体数据类型，是在调用该方法时，传入实参时确定具体的数据类型的（实参是什么类型，那么这个泛型就是什么类型）
    2. 自定义泛型只要符合标识符的命名规则即可。但是自定义泛型我们一般都习惯使用一个大写字母表示

<T> Type 数据类型

<E> Element 元素

## 6、泛型类

* 格式：

class 类名 <声明的自定义泛型>{

}

* 泛型类要注意的事项

1. 在类上自定义的泛型的具体数据类型，是在创建类的对象时确定的。
2. 如果一个类已经声明了自定义泛型，在创建类的对象时却没有指定泛型的具体数据类型，那么默认是 object类型的
3. 类上的自定义泛型，不能作用于静态方法，如果想要让静态方法使用泛型，那么需要在方法上自己声明使用。

## 7、泛型接口

* 格式：

interface 接口名 <自定义泛型>{ }

* 接口泛型注意事项

1. 接口上自定义的泛型类型，是在实现一个接口的时候指定的
2. 在接口上自定义的泛型，如果在实现接口的时候没有指定具体的数据类型，那么默认为Object类型

* 需求：目前我实现一个接口的时候，还不确定泛型的具体数据类型，需要等到创建实现类对象的时候才能确定，怎么延长接口自定义泛型的时间？
  + - 如果需要延长接口自定义泛型的具体数据类型，那么格式如下：

public class Dome4<T> implements Dao<T>{ }

## 8、泛型的上下线

需求1：定义一个函数可以接收任意类型的集合对象，要求接受的集合对象只能存储Integer或Integer的父类数据类型

需求2：定义一个函数可以接受任意类型的集合对象，要求接收的集合对象只能存储Number或Number的子类数据类型。

* 泛型的通配符 ？

？ super Integer：只能存储Integer或Integer的父类数据类型，泛型的下限

？ extends Number：只能存储Number或Number的子类数据类型，泛型的上限

# 十三、单列集合复习

1. 单列集合 的体系：

(1)Collection 单例集合的根接口

* + - List 如果是实现了List接口的集合类， 具备的特点：有序，重复。
    - ArraryList 底层 是使用了Object数组实现的，特点： 查询速度快，增删慢。
    - LinkedList 底层是使用了链表数据结构实现 的， 特点： 查询速度慢，增删快。
    - Vector Vector的实现与ArrayList是一致，但是是线程安全 的，操作效率低。 jdk1.0的时候出现的

(2)Set

* 如果是实现了Set接口的集合类，具备的特点：无序，不可重复。

1)HashSet 底层是使用了一个哈希表支持的， 特点：存取速度快。

* + HashSet添加元素的原理：
    - * + 往HashSet添加元素的时候，首先HashSet会调用元素的hashCOde方法得到元素的哈希码值，然后会经过一系列运算就可以算出该元素在哈希表中的存储位置:

情况1：如果算出该元素的位置目前没有任何元素存储，那么该元素可以直接存储

情况2： 如果算出该元素的位置目前已经存有其他的元素，那么还会调用元素的equals方法与该位置的元素再比较一次。如果equals方法返回的是false，那么该元素允许存储，如果euqlas方法返回的是true，那么该元素被视为重复元素，不允许存储。

1. TreeSet 底层是使用了红黑树（二叉树）数据结构实现的， 特点：会对元素进行排序存储。

* TreeSet要注意的事项：
  + 1.往TreeSet添加元素的时候，如果元素本身具备自然顺序的特性，那么会根据元素自然顺序的特性进行排序存储。
  + 2.往TreeSet添加元素的时候，如果元素本身不具备自然顺序的特性，那么元素所属的类必须要实现Comparable接口，把元素的比较规则定义在CompareTo方法上。
  + 3.往TreeSet添加元素的时候，如果元素本身不具备自然顺序的特性，而且元素所属的类没有实现COmparable接口，那么必须要在创建TreeSet对象的时候传入比较器。
  + 4.如果比较的方法(CompareTo 或者Compare )返回的是0的时候，那么该元素就被视为重复元素，不允许添加。
* 比较器的定义格式： 自定义一个类实现COmparator接口即可。

class 类名 implements Comparator{ }

2、泛型：泛型是jdk1.5出现的新特性。

（1）泛型的好处：

1. 将运行时出现 的问题提前至了编译时。

2. 避免了无谓强制类型转换。

（2）自定义泛型： 自定义泛型就是一个数据类型的占位符或者理解为一个数据类型的变量。

（3）泛型方法:

修饰符 <声明自定义的泛型>返回值类型 函数名(自定义的泛型 变量名..) { }

* 泛型方法要注意的事项：
  + 1.泛型方法中 的自定义泛型的具体数据类型是在调用该函数的时候传入实参时确定的。
  + 2. 自定义泛型所用 的标识符只要符合标识符 的命名规则即可。但是我们一般都习惯使用一个大写字母表示。

（4）泛型类：

* 泛型类的定义格式

class 类名<声明自定义的泛型>{ }

* 泛型类要注意的事项：
  + 1.泛型类上的自定义泛型是在使用该类创建对象的时候指定具体的数据类型的。
  + 2.如果一个类已经自定义了泛型，使用该类创建对象的时候如果没有指定泛型的具体数据类型，那么默认为Object类型。
  + 3.静态的函数不能使用类上自定义的泛型，如果静态函数需要使用，必须要在函数上自定义泛型。

（4）泛型接口：

* 泛型接口的定义格式：

interface 接口名<声明自定义的泛型>{ }

* 泛型接口要注意事项：
  + 1.泛型接口上的自定义泛型是在实现该接口的时候指定具体数据类型的。
  + 2.如果实现接口的时候没有指定接口上 的自定义泛型的具体数据类型，那么默认为Object数据类型。
  + 3. 如果需要在创建接口实现类对象的时候才指定接口上自定义泛型，那么需要以下格式： class<T> 类名 implements 接口<T>

（5）泛型上下限：

? super Integer 允许是Integer数据类型或者是Integer父类类型 泛型的下限

? extedns Number 允许是Number数据类型或者是Number子类的数据类型 泛型的上限。

# 十四、Map集合

## 1、Map接口

* 双列集合

-------- | Map集合：实现了Map接口的集合类，具备以下特点：

* + 存储的数据都是以键值对(key-value)的形式存在的，键不可重复，值可重复。
  + 注意：null可以作为map集合的 键(key)

---------------- | HashMap类：底层是基于 哈希表 实现的

---------------- | TreeMap类：

底层是基于 红黑树(二叉树)数据结构实现的，特点是 会对元素的键(key)进行自然顺序排序存储。

---------------- | HashTable类：底层是基于 哈希表 实现的，实现方式与 HashMap一致，但是HashTable是线程安全的，操作效率低。（HashTabella是jdk1.0开始使用的，HashMap是jdk1.2开始使用的）

## 2、Map接口的方法

|  |
| --- |
| 添加：   * **put(K key,V value)**：向map集合中添加一组键值对，如果之前该key不存咋，则返回null；如果之前该key存在，则返回key所映射的旧值，然后新值覆盖旧值 * **putAll(Map<? extends K,? extends V> m)**：将指定map2集合添加到当前map集合中   删除：   * **remove(Object key)**：根据key删除map集合中的一组数据，并返回该key所对应的value值 * **clear()**：清空map集合中的所有数据   获取：   * **get(Object key)**：根据指定的key获取所对应的value值 * **size()**：获取map集合键值对的个数   判断：   * **containsKey(Object key)**:判断map集合中是否存在指定的键 * **containsValue(Object value)**：判断map集合中是否存在指定的value * **isEmpty()**：判断map集合是否存在键值对，不存在返回true，存在返回false |

## 3、Map接口的三种迭代方式

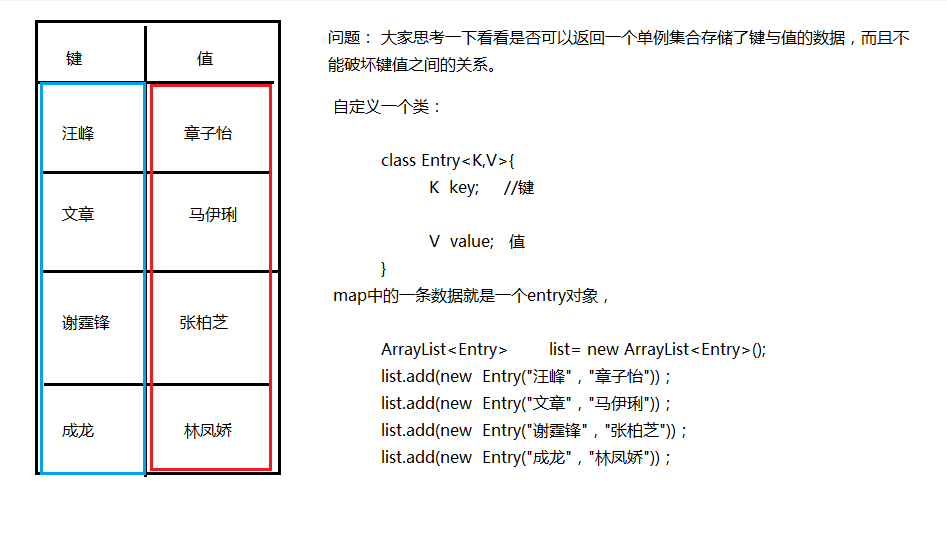
1. KeySet():

* 使用KeySet()方法进行遍历，缺点是KeySet()返回的只是所有键，没有value值。

2. Values():

* 使用Values()方法进行遍历，缺点是Values()返回的只是所有value值，没有键。

3. entrySet():返回一个set集合，集合中的元素是以 键值对 存储的。



# 十五、HashMap类

## 1、实现原理

* 底层是基于 哈希表 实现的。
* 首先调用键的 hashCode()方法得到哈希码值，然后经过运算就可以算出该元素在哈希表中的位置：
  + 情况一：如果算出该位置目前没有任何元素，那么可以直接添加到哈希表中
  + 情况二：如果该位置已经存在其他元素，那么会调用该元素的equals方法，与这个位置的元素进行比较，如果返回值是false，那么允许被存储;如果返回值是true，那么视为该元素是重复元素，不允许存储。

# 十六、TreeMap类

## 1、实现原理

* 底层是基于 红黑树(二叉树)数据结构 实现的（左小右大原则）
* 1. 往TreeMap集合中添加元素时，如果元素的键具备自然顺序特性，那么就会按照键的自然顺序排序存储。
* 2. 往TreeMap集合中添加元素时，如果元素的键不具备自然顺序属性，那么键的所属类需要实现Comparable接口，并将 比较规则 写在compareTo()方法中。
* 3. 往TreeMap集合中添加元素时，如果元素的键不具备自然顺序属性，键所属的类也没有实现Comparable接口，那么就必须在创建TreeMap对象的时候传入比较器。

# 十七、HashTable类

## 1、实现原理

* HashTable的底层也是基于 红黑树（二叉树）原理实现的，实现方式与HashMap一样，但是HashTable是线程安全的，操作效率低。
* HashTable是jdk 1.0开始使用的;HashMap是jdk 1.2开始使用的。

# 十八、Collections工具类

1. Collections是集合框架中的工具类，特点：该工具类中的方法都是静态的

## 1. Collections集合工具类

|  |
| --- |
| 1. 对list进行二分查找：   前提该集合一定要有序。  (1)int binarySearch(list,key);  //必须根据元素自然顺序对列表进行升级排序  //要求list 集合中的元素都是Comparable 的子类。  (2)int binarySearch(list,key,Comparator);  2，对list集合进行排序。  (1)sort(list);  //对list进行升序排序,其实使用的是list容器中的对象的compareTo方法  (2)sort(list,comaprator);  //按照指定比较器进行排序  3，对集合取最大值或者最小值。  (1)max(Collection)  (2)max(Collection,comparator)  (3)min(Collection)  (4)min(Collection,comparator) 4，对list集合进行反转。  (1)reverse(list);  8，可以将不同步的集合变成同步的集合。  (1)Set synchronizedSet(Set<T> s)  (2)Map synchronizedMap(Map<K,V> m)  (3)List synchronizedList(List<T> list) |

## 2、Arrays数组工具类

|  |
| --- |
| 1，二分查找,数组需要有序  binarySearch(int[])  binarySearch(double[])  2，数组排序  sort(int[])  sort(char[])……   1. 将数组变成字符串。   toString(int[])   1. 复制数组。  copyOf(); 2. 复制部分数组。   copyOfRange():   1. 比较两个数组是否相同。   equals(int[],int[]);   1. 将数组变成集合。   List asList(T[]);  这样可以通过集合的操作来操作数组中元素，但是不可以使用增删方法，add，remove。因为数组长度是固定的，会出现UnsupportOperationExcetion。  可以使用的方法：contains，indexOf。。。  如果数组中存入的基本数据类型，那么asList会将数组实体作为集合中的元素。  如果数组中的存入的引用数据类型，那么asList会将数组中的元素作为集合中的元素。 |

## 3.例题

**Collections集合操作类**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Arrays;  **import** java.util.List;  **class** Demo1  {  **public** **static** **void** main(String[] args)  {  ArrayList<Integer> list = **new** ArrayList<Integer>();  list.add(4);  list.add(3);  list.add(1);  list.add(2);  list.add(3);  // 排序  Collections.*sort*(list);  // 折半查找的前提是排序好的元素  System.*out*.println( Collections.*binarySearch*( list , 8 ) ); // 找不到返回-插入点-1  // 反序集合输出  Collections.*reverse*( list );  System.*out*.println( list );  // 求最值  System.*out*.println( Collections.*max*( list ) ); // 4  // fill() 使用指定的元素替换指定集合中的所有元素  // Collections.fill( list, 5 );  System.*out*.println( list );  // 将数组转换为集合  Integer is[] = **new** Integer[]{6,7,8};  List<Integer> list2 = Arrays.*asList*(is);  list.addAll( list2 );  System.*out*.println( list );  // 将List转换为数组  Object [] ins = list.toArray();  System.*out*.println( Arrays.*toString*( ins ) );  }  } |

**集合的练习**

问题： 定义一个Person数组，将Person数组中的重复对象剔除？

思路:

1. 描述一个Person类

2. 将数组转换为Arrays.asList() List

3. Set addAll( list )

4. hashCode()且equals()

|  |
| --- |
| **package** map;  **import** java.util.Arrays; **import** java.util.HashSet; **import** java.util.List; **import** java.util.Set;  **public class** Dome4 {   **public static void** main(String[] args) {  // Person数组  Person[] people = {**new** Person(01, **"百晓生"**)  , **new** Person(02, **"张岱"**)  , **new** Person(03, **"狗剩"**)  , **new** Person(03, **"狗剩"**)  , **new** Person(02, **"狗蛋"**)};  // 将数组转换成list集合  List<Person> list = Arrays.asList(people);  System.**out**.println(list.size());  // 利用Set集合不能出现重复元素的特性，到达去重的  Set<Person> set = **new** HashSet<Person>();  set.addAll(list);//将list集合添加到set集合中  System.**out**.println(set);  } }  **class** Person {  **private** String **name**; // 名字  **private int id**; // 身份证号   **public** Person() {  }   **public** Person(**int** id, String name) {  **this**.**id** = id;  **this**.**name** = name;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public int** getId() {  **return id**;  }   **public void** setId(**int** id) {  **this**.**id** = id;  }   @Override  **public** String toString() {  **return "{身份证号："** + **this**.**id** + **" 姓名"** + **this**.**name** + **"}"**;  }   // id和姓名都相同就认为是同一个人  @Override  **public boolean** equals(Object obj) {  Person p = **null**;  // instanceof ：左边是对象，右边是类。如果左边对象的实例是右边对象或右边对象的子类，返回true，否则返回false。  **if** (obj **instanceof** Person) {  p = (Person) obj;  }  **return** (**this**.**name**.equals(p.**name**)) && (**this**.**id** == p.**id**);  }   // 重写hashCode方法  @Override  **public int** hashCode() {  **return this**.**id**;  } } |