1.集合框架介绍



2.java.util.Collection**接口（注意这也是个接口）**

**单列集合:顶层集合Collection接口**

所有单列集合的最顶层的接口，里边定义了所有单列集合共性的方法。

任意的单列集合都可以使用Collection接口中的方法。

3.Collection常用功能（共性方法）

public boolean add(E e);把给定的对象添加到当前集合中。

public void clear();清空集合中所有的元素。

public boolean remove(E e);把给定的对象在当前集合中删除。

public boolean contaions(E e);判断当前集合是否包含给定的对象。

public boolean isEmpty();判断当前集合是否为空。

public int size();返回集合中元素的个数。

public Object[] toArray();把集合中的元素，存储到数组中。

详细看下图：





4.Iterator接口（也叫迭代器）

1、迭代的概念：（注意Iterator是个接口，无法直接使用）

迭代：即Collection集合元素的**通用获取方式**。在取元素之前先要判断集合中有没有元素，如果有，就把这个元素取出来，继续再判断，如果还有就再取出来。一直把集合中所有的元素全部取出来。这种取出方式专业术语称为迭代。

Interator**接口**的常用方法：

public boolean hsaNext();判断集合中还有没有下一个元素。

public E next();取出集合中的下一个元素。

备注：Iterator是一个接口，这个接口实现类的获取有些特殊：Collection接口中有一个方法，叫iterator(),这个方法返回的就是迭代器的实现类对象。

Iterator<E> iterator()返回在此collection的元素上进行迭代的迭代器。

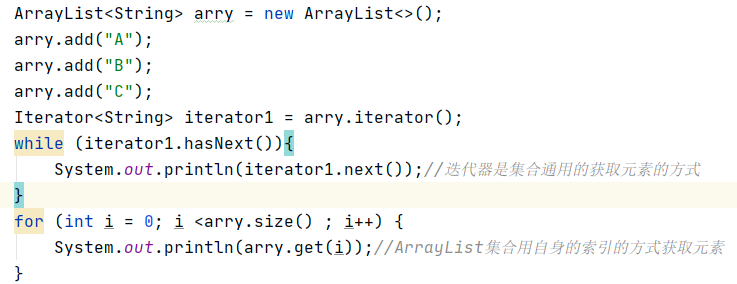
2、迭代器的的使用步骤：

1.使用集合中的方法iterator()获取迭代器的实现类对象，使用Iterator接口接收。

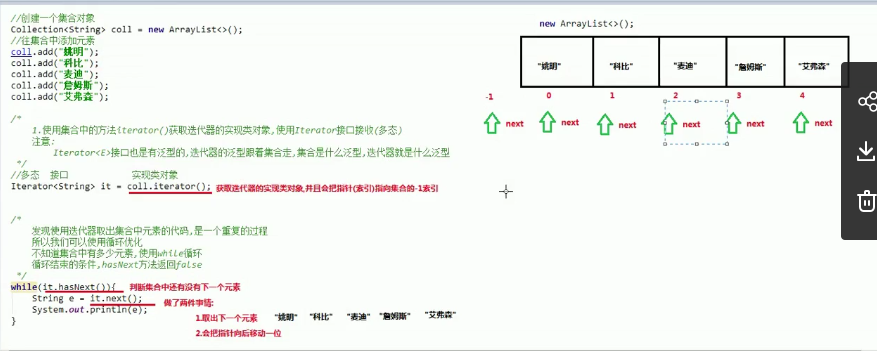
2.使用Iterator接口中的方法hasNext（）判断还有没有下一个元素。

3.使用Iterator接口中国的方法next（）取出集合中的下一个元素。





3、迭代器的实现原理



4、增强for循环

增强for循环（也称for each循环）是**JDK 1.5**后出来的一个高级for循环，专门用来遍历数组和集合的。它的内部原理其实是Iterator迭代器，所以在遍历的过程中，不能对集合中的元素进行增删操作。

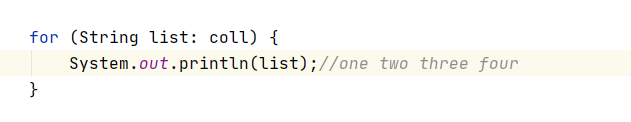
格式：

for(元素的数据类型 变量 ： Collection集合or数组){

//写操作的代码

}

它用于遍历Collection和数组。通常只进行遍历元素，不要在遍历的过程中对元素进行增删操作。



Collection<E> extends Iterable<E>:所有的单列集合都可以使用增强for循环

public interface Iterable<T>实现这个接口允许对象成为“foreach”语句的目标。

**遍历集合和数组就用foreach（增强for）好了，比较的方便。**

5.泛型

1、泛型是一种未知的数据类型，当我们不知道使用什么数据类型的时候，就可以使用泛型，泛型也可以看作是一个变量，用来接收数据类型。

E e:Element元素

T t:Type类型

如：ArrayList集合在定义的时候，不知道集合中都会存储什么类型的数据，所以类型使用泛型。

public class ArrayList<E>{} E:未知的数据类型

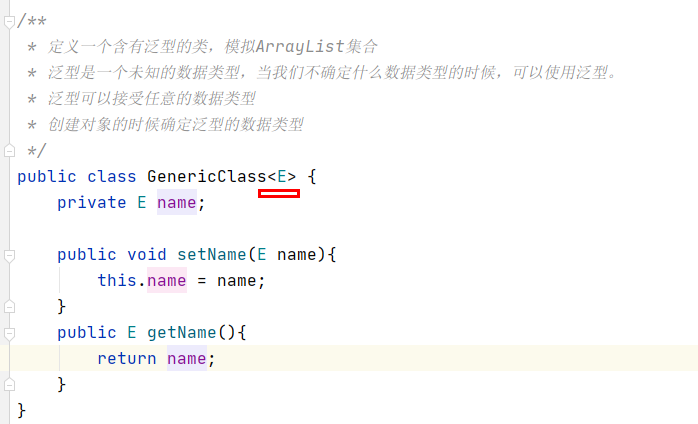
创建集合对象的时候，就会确定泛型的数据类型。

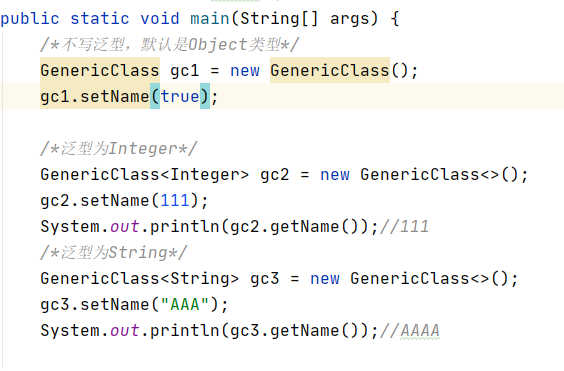
2、使用泛型的好处

****

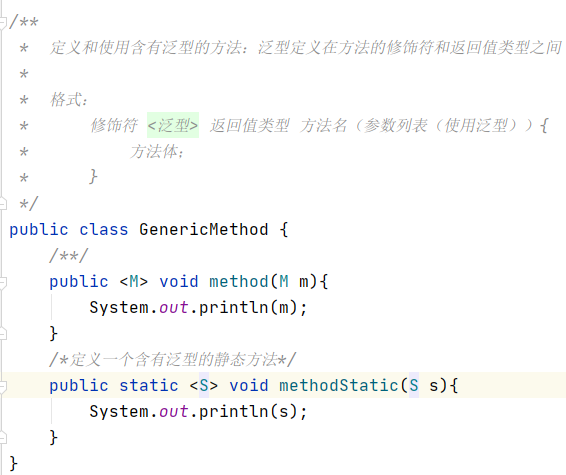
****

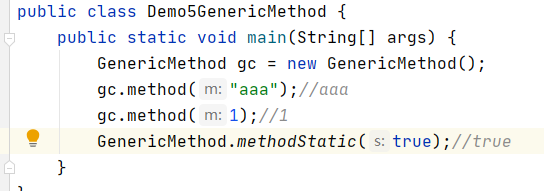
3、定义和使用含有泛型的类



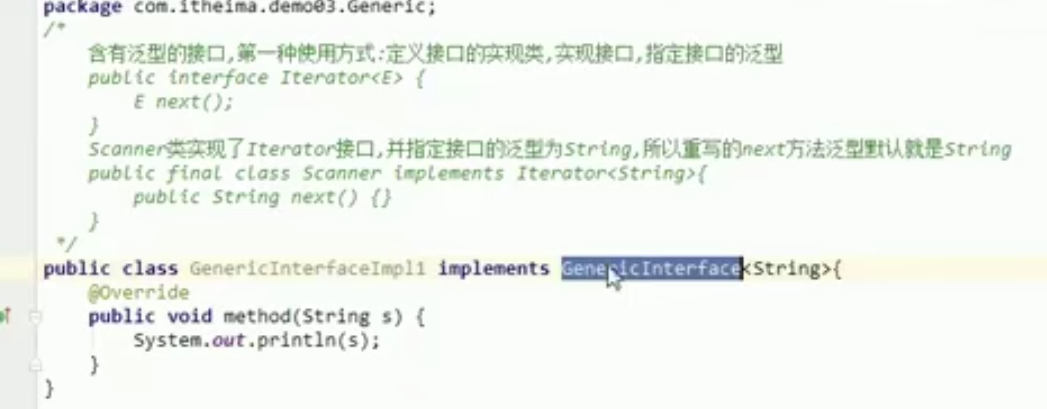


4、定义和使用含有泛型的方法

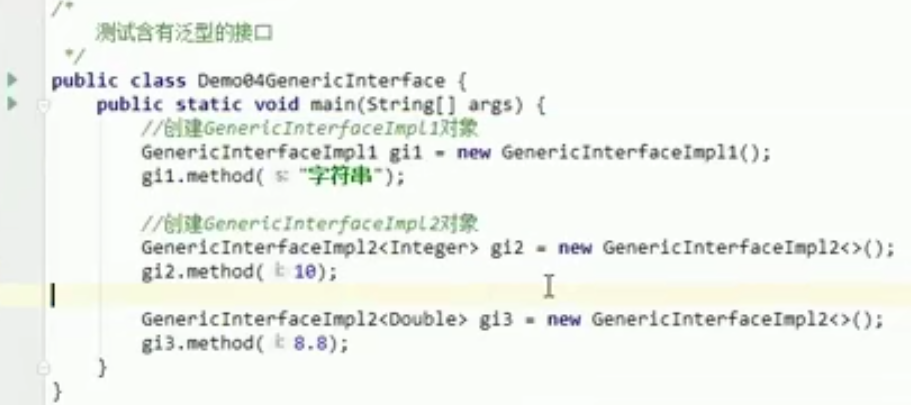




5、定义和使用含有泛型的接口







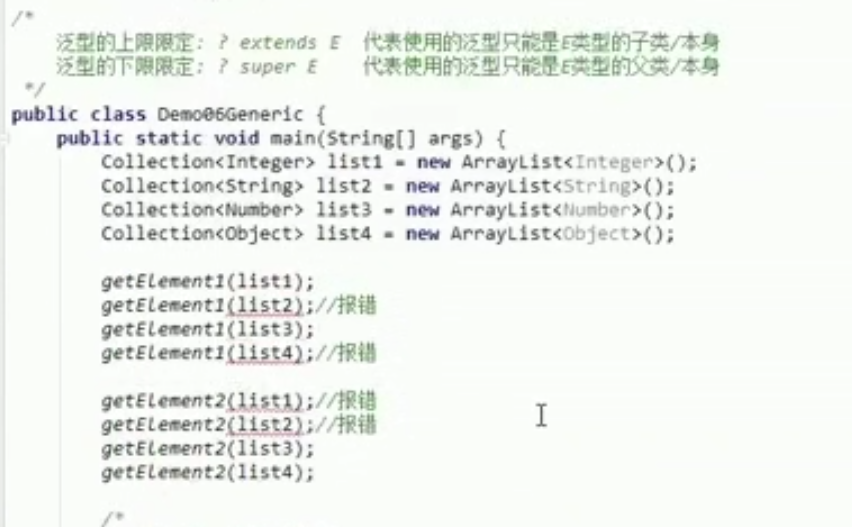
6、泛型通配符

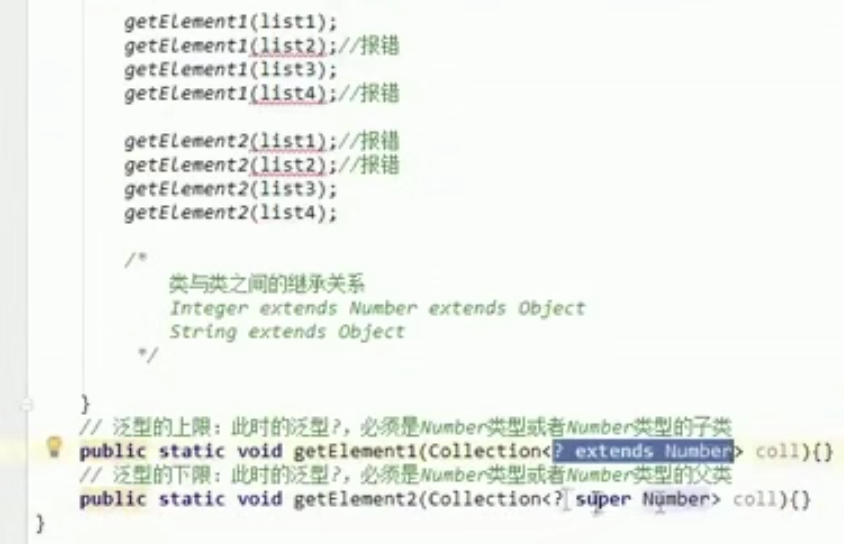
当使用泛型类或者接口时，传递的数据中，泛型类型不确定，可以通过通配符<?>表示，但是一旦使用通配符后，只能使用Object类中的共性方法，集合中元素自身方法无法使用。

通配符基本使用：

泛型的通配符：**不知道使用什么数据类型来接收的时候，此时就可以使用？。？表示未知通配符。**

此时只能接收数据，不能往该集合中存储数据。

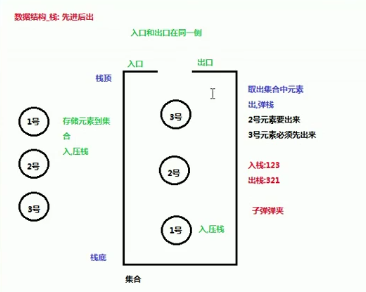




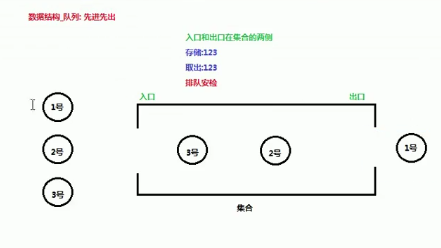
7.常见的数据结构（与集合有关的）

数据存储的常用结构有：栈，队列，数组，链表和红黑树。

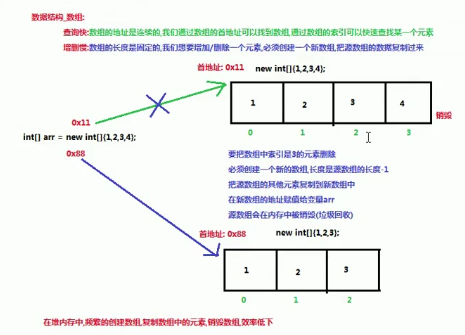
栈（stack）：先进后出。



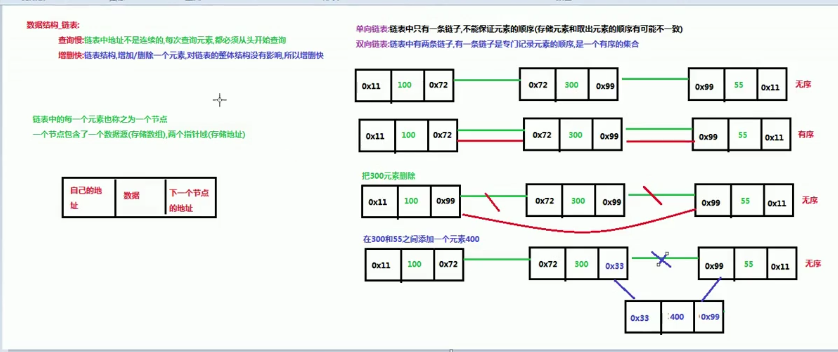
队列：先进先出。



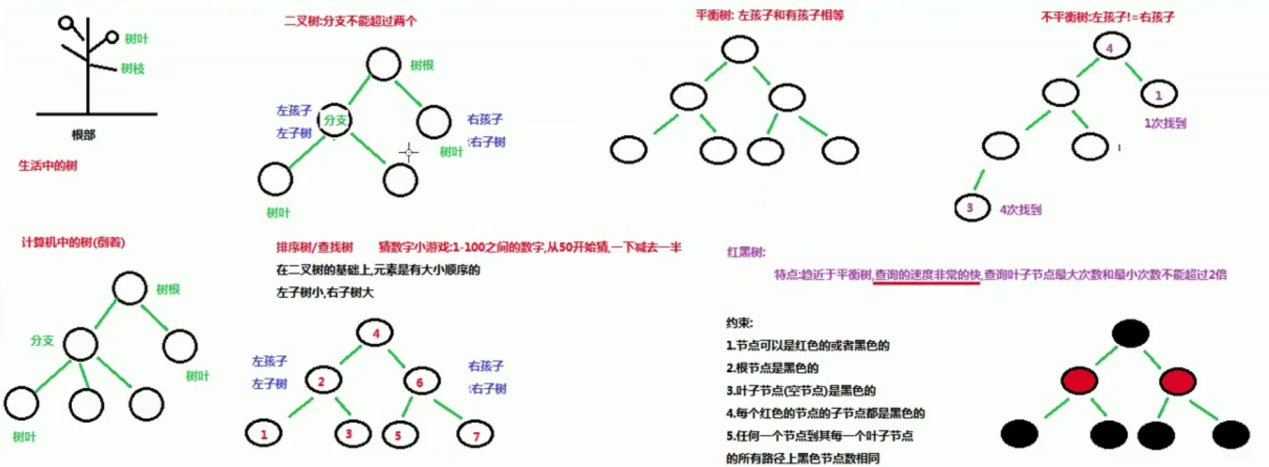
数组：Array是有序的元素序列。查询快，增删慢



链表：linkedlist：由一系列结点组成。查询慢，增删快



红黑树



1.List集合

Java.util.List接口 extends Collection接口

List接口的**特点**：

1.有序的集合，存储元素和取出元素的顺序一致。

2.有索引，包含一些带索引的方法。

3.允许存储重复的元素。

List接口中**带有索引的方法**（**特有**）：

-public void add(int index,E element):将指定的元素，添加到该集合中的指定索引位置。

-public E get(int index):返回集合中指定位置的元素。

-public E remove(int index):移除列表中指定位置的元素，返回的是被移除的元素。

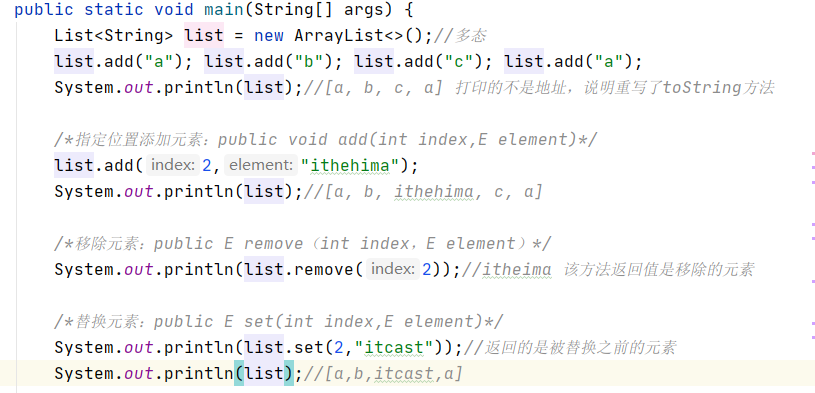
-public E set(int index,E element):用指定元素替换集合中指定位置的元素，返回值是更新前的元素。

注意：操作索引的时候，一定要防止索引越界异常。

IndexOutOfBoundsException：索引越界异常，集合会报。

ArrayIndexOutOfBoundsException:数组索引越界异常。

StringIndexOutOfBoundsException：字符串索引越界异常。





2.List集合的实现类：

1、ArrayList集合：（之前学过）List接口的大小可变数组的实现。**注意：此实现不是同步的。**[说明是多线程，效率高，速度快]

ArrayList集合中的add()底层调用的是System.arraycopy()方法，也就是复制数组，所以说数组的查找快，增删慢。

/\*\*

java.util.ArrayList集合数据存储的结构是数组结构，元素增删慢，查找快，由于日常开发中使用最多的功能为查询数据，遍历数据，所以ArrayList是最常用的集合。

许多程序员开发时非常随意地使用ArrayList完成任意需求，并不严谨，该做法不提倡。

\*\*/

2、LinkList集合:List接口的链表实现。元素增删快，查找慢。注意：**注意：此实现不是同步的。**[说明是多线程，效率高，速度快]

LinkedList集合的特点：

1、底层是一个链表结构，查询慢，增删快。

2、里边包含了大量操作首尾元素的方法。

注意：使用LinkedList集合特有的方法，不能使用多态。

-public void addFirst(E e):将指定元素插入此列表的开头。

-public void addLast(E e):将指定元素插入此列表的结尾。

-public void push(E e):将元素推入此列表所表示的堆栈。效果等同于addFirst(E e)

-public E getFirst():返回此列表的第一个元素。

-public E getLast():返回此列表的最后一个元素。

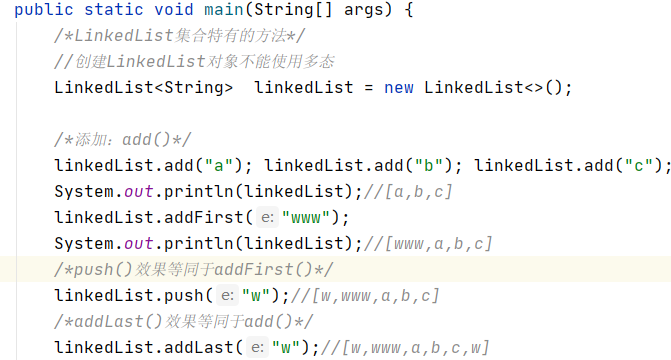
-public E removeFirst():移除并返回此列表的第一个元素。

-public E removeLast():移除并返回此列表的最后一个元素。

-public E pop():从此列表所表示的堆栈所表示的堆栈出弹出一个元素.

(该方法等效于removeFirst())

-public boolean isEmpty():如果列表不包含元素，返回true。





3.Vector集合（JDK1.2才实现List接口）

该集合是JDK1.1版本就有的，而其他集合是JDK1.2版本才有的。底层也是数组。但是**注意：该集合是同步的，【同步即是单线程，速度慢，效率低】**

添加元素：addElement(E e);

遍历元素（JDK1.2才有迭代器）：

2.Set接口

public interface Set<E> extends Collection<E>一个不包含重复元素的collection。

特点：

1.不允许存储重复的元素

2.没有索引，没有用索引的方法，也不能使用普通的for循环遍历。

1、HashSet集合

java.uti.HashSet集合 implements Set接口

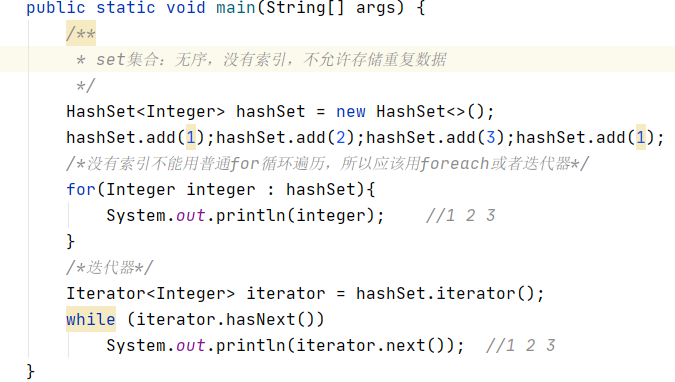
HashSet特点：

1、不允许存储重复的元素。

2、没有索引，没有带索引的方法，也不能使用普通的for循环遍历

3、是一个无序的集合，存储元素和取出元素的顺序有可能不一致。

4、底层是一个哈希表结构（查询的速度非常快）



/\*\*

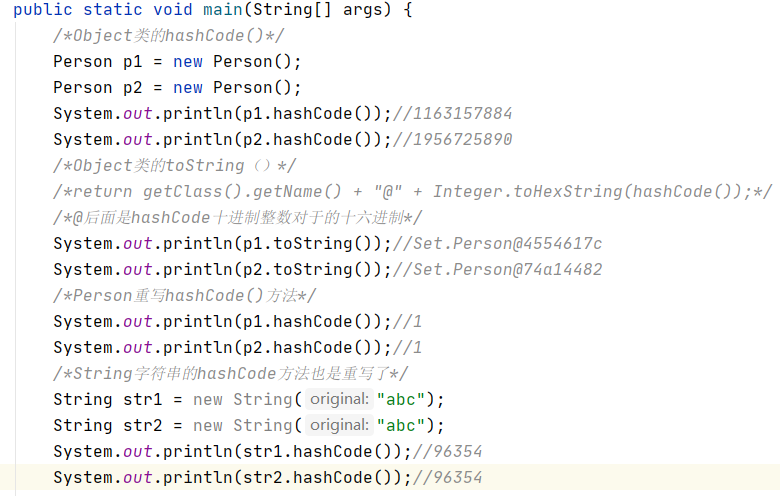
1.哈希值：是一个**十进制**的**整数**，系统随机给出（就是对象的地址值，是一个逻辑地址，是模拟出来得到地址，不是数据实际存储的物理地址）。

Object类中有一个方法可以获得对象的哈希值。

int hashCode(); 返回该对象的哈希码值

public native int hashCode（）；

native:代表该方法调用的是本地操作系统的方法。



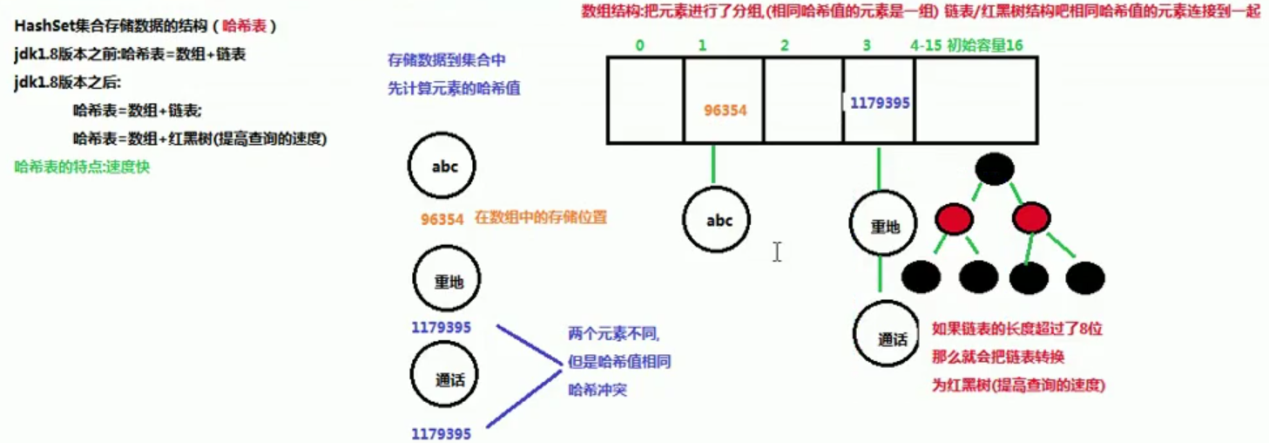
2.HashSet集合存储数据的结构（哈希表）

JDK1.8版本之前：哈希表 = 数组 + 链表

JDK1.8版本之后：哈希表 = 数组 + 链表；

哈希表 = 数组 + 红黑树（提高查询的速度）

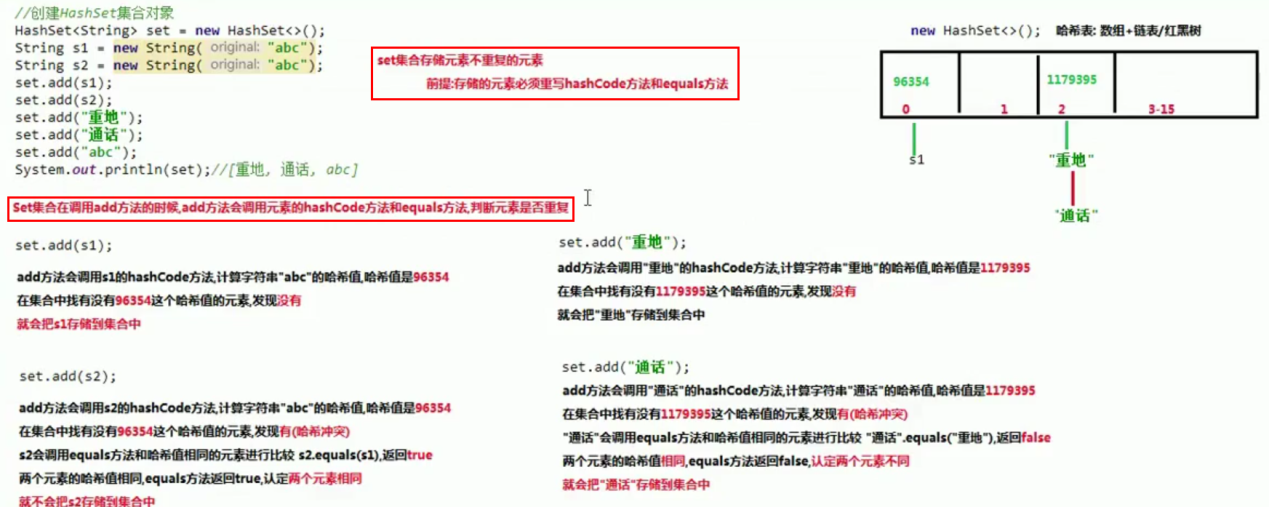
哈希表的特点速度快



3.Set集合不存储重复元素的原理

**Set集合底层是一个哈希表结构，存储的是哈希值。**

【前提：存储的元素的类型（String，Integer，…Student，Person…）,必须重写hashCode方法和equals方法，以保证集合元素的唯一。】

****

4.HashSet存储自定义类型元素

给HashSet中存放**自定义类型元素**时，需要**重写对象中的hashCode方法和equals方法，建立自己的比较方式，以保证HashSet集合中的对象唯一。**

【重写可以使用idea自带的重写方法】

****

5.LinkedHashSet集合

Java.util.LinkedHashSet集合 extends HashSet集合

LinkedHashSet集合特点：

底层是一个哈希表（数组+链表/红黑树）+链表：多了一条链表（记录元素的存储顺序），保证元素的有序，所以这是一个有序的集合。



/\*\*

可变参数：

是JDK1.5之后出现的新特性

使用前提：

当方法的参数列表数据类型已经确定，但是参数的个数不确定，就可以使用可变参数。

使用格式：定义方法时使用

修饰符 返回值类型 方法名（数据类型…变量名）{}

即相当于：修饰符 返回值类型 方法名（数据类型[] 变量名）{}

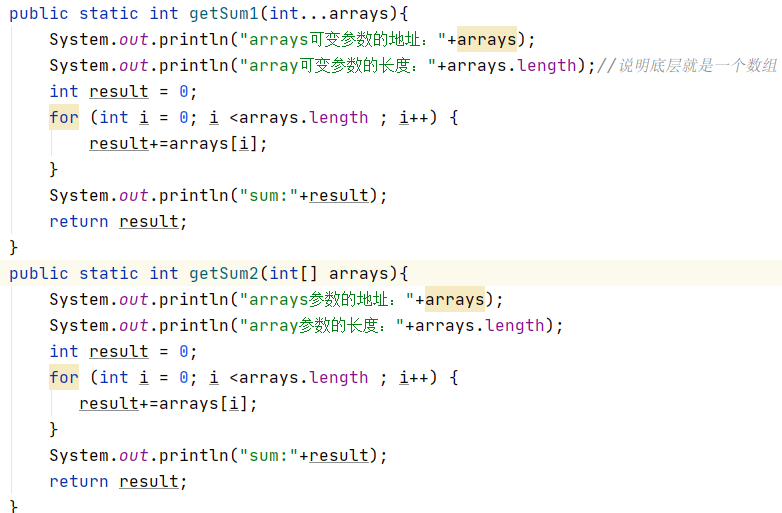
可变参数的原理：

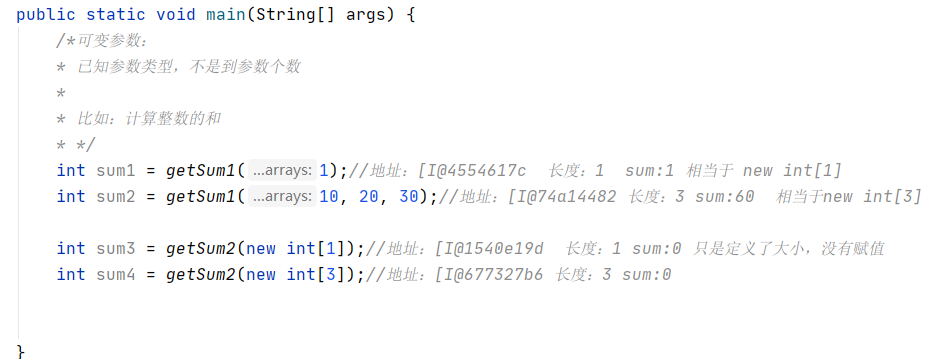
可变参数底层就是一个数组，根据传递参数个数不同，会创建不用长度的数组，来存储这些参数，传递的参数个数可以是0个（不传递），1,2,3…多个

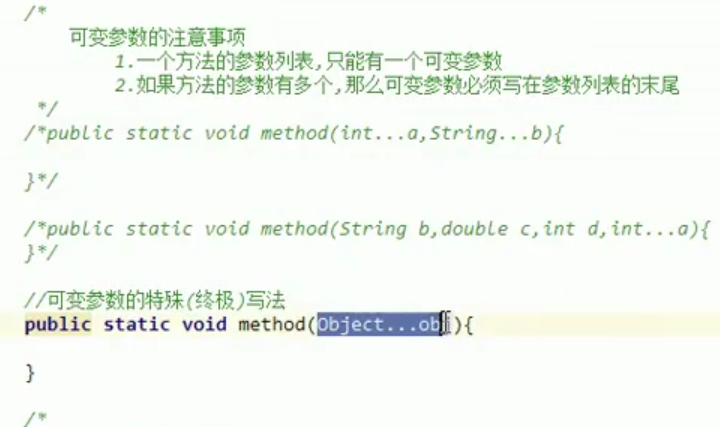
注意事项：

1.一个方法的参数列表，只能有一个可变参数

2.如果方法的参数有多个，那么可变参数必须写在参数列表的末尾







Collections

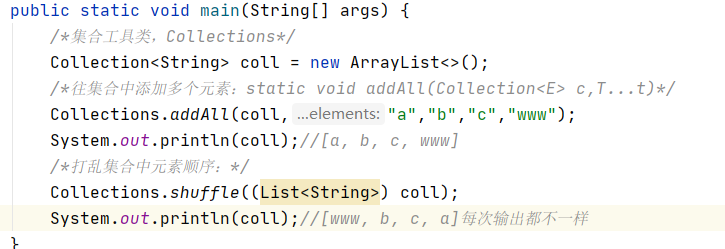
java.uti.collections是集合工具类，用来对集合进行操作。

常用方法：

-public **static** <T> boolean addAll(*Collection<T> c*,**T… elements**):往集合中添加多个元素。

-public **static** void shuffle(**List<?> list**)：打乱顺序，打乱集合中元素的顺序。

List集合才能使用（图中使用的是Collection集合（多态），只要向下转型就可以了）



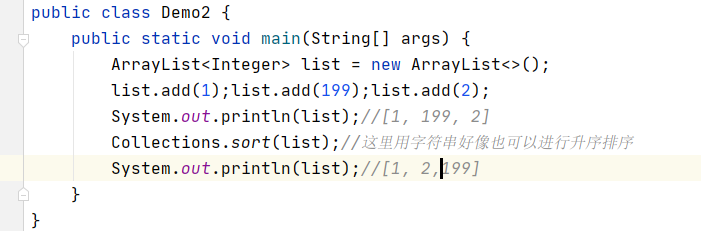
-public static <T> void sort(List<T> list):将集合中元素按照默认规则排序。（升序）

注意事项：

该方法的使用前提是：被排序的集合里边存储的元素，必须实现Comparable接口，重写接口中的compareTo(Object obj)方法，定义排序的规则。

Comparable接口的排序规则：

自己（this）-参数：升序 [否则降序]







-public static <T> void sort(List<T> list,Comparator<? super T>):将集合中的元素按照指定规则排序。

Comparator和Comparable的区别

Comparable:自己(this)和别人（参数）比较，自己需要实现Comparable接口，重写比较规则comparTo方法

Comparator:相当于找一个第三方的裁判，比较两个

Comparator的排序规则：o1-o2升序 ； o2-o1降序。



1.Map<K,V>集合(接口)

public interface Map<K,V>

java.util.Map<K,V>集合（接口）

Map集合的特点：

1.Map集合是一个**双列集合**，一个元素包含两个值（key,value）

2.Map集合中的元素，key和value的**数据类型可以相同，也可以不同**

3.Map集合中的元素，**key**是**不允许重复**的，value是可以重复的

4.Map集合中的元素，key和value是**一一对应**的。



2.Map常用子类

java.util.**HashMap<K,V>**集合 implements Map<K,V>接口

**HashMap**集合的特点：

1.HashMap集合底层是哈希表：查询速度非常快。

JDK1.8之前：数组+单向链表

JDK1.8之后：数组+单向链表/红黑树（链表长度超过8）：提高查询速度

2.HashMap集合是一个无序的集合，存储元素和取出元素的顺序有可能不一致。

java.util.**LinkedHashMap<K,V>**集合 extends HashMap<K,V>集合

**LinkedHashMap**集合的特点：

1.LinkedHashMap集合底层是哈希表+链表（保证迭代的顺序）

2.LinkedHashMap集合是一个有序的集合，存储和取出元素的顺序是一致的。

3.Map接口中的常用方法

**-public V put(K key,V value):**把指定的键与指定的值添加到Map集合中。

返回值V:

存储键值对的时候，key不重复，返回值V是null

存储键值对的时候，key重复,会使用新的value替换map中重复的value,并返回被替代的value值。

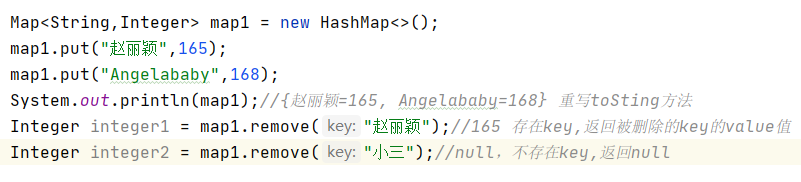


**-public V remove(Object key):**把指定的键所对应的键值对元素在map集合中删除，返回被删除的元素。

返回值V：

Key存在，V返回被删除的value值

Key不存在，V返回null



**-public V get(Object key):**根据指定的键，在Map集合中获取对应的值。

返回值：

Key存在，V返回对应的值

Key不存在，返回null

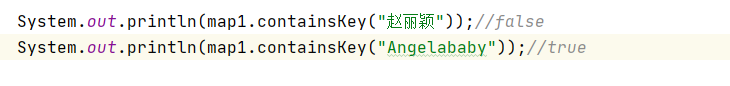


**-public boolean containsKey(Object key):**判断map集合中是否包含指定的key

返回值：

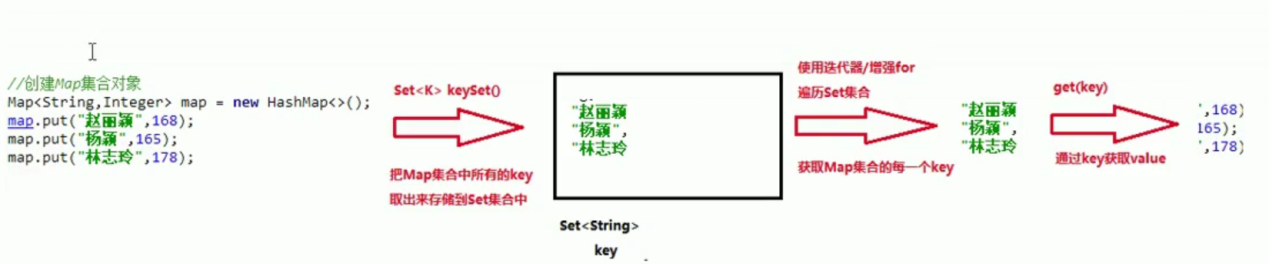
包含，返回true

不包含，返回false



4.Map集合遍历的两种方式：

流程视图：



Map集合的第一种遍历方式：通过键找值的方式。

Map集合中的方法：

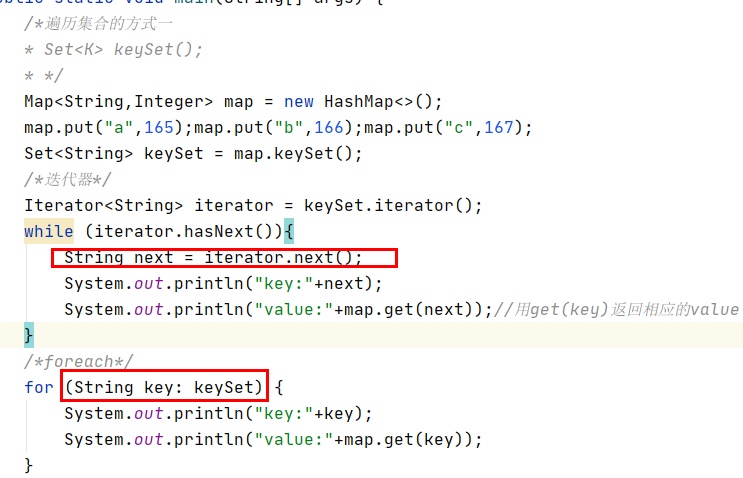
Set<K> keySet();返回此映射中包含的键的Set视图。

实现步骤：

1.使用Map集合中的keySet()，把Map集合所有的key取出来，存储到一个Set集合。

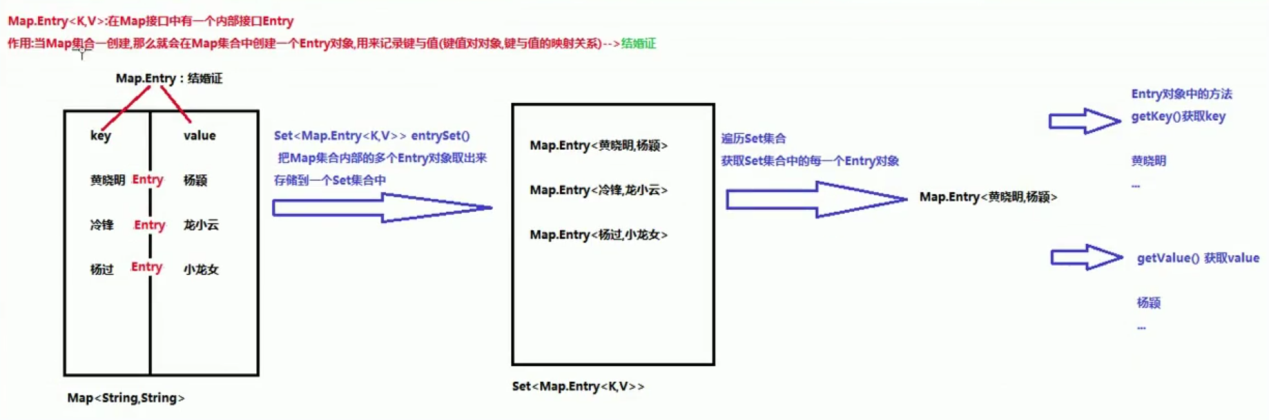
2.遍历Set集合，获取Map集合中的每一个key。

3.通过Map集合中的get(key),通过key来找到value。



Map集合的第二种遍历方式：使用Entry对象遍历方式。

1.什么是Entry键值对对象？



Map集合中的方法：

Set<Map.Entry<K,V>> entrySet()：返回此映射中包含的映射关系的Set视图。

实现步骤：

1.使用Map集合中的entrySet(),把Map集合中多个Entry对象取出来，存储到一个Set集合中。

2.遍历Set集合，获取每一个Entry对象

3.使用Entry对象中的方法getKey(),getValue()获取对象的key和value值



5.HashMap存储自定义类型键值：

/\*\*

HashMap存储自定义类型键值

Map集合**保证key**是**唯一**的：

作为**key**的元素，必须重写hashCode方法和equals方法，**以保证key唯一**

\*\*/





6.LinkedHashMap集合

java.util.LinkedHashMap<K,V> extends HashMap<K,V>

Map接口的哈希表和链表的实现，具有可预知的迭代顺序。

底层原理：

哈希表+链表（记录元素的顺序）



7.Hashtable集合

**（小备注：之前学习的所有集合都可以存储null为键或值）**

java.util.Hashtable<K,V>集合 implements Map<K,V>接口

Hashtable:底层也是一个哈希表，是一个**线程安全**的集合，是**单线程**集合，速度**慢**。

HashMap：底层是一个哈希表，是一个**线程不安全**的集合，是**多线程**集合，速度**快**。

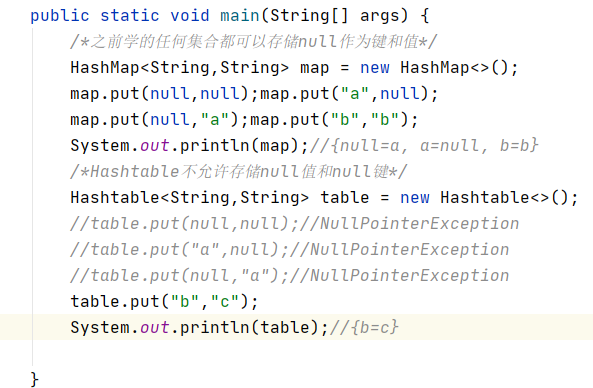
HashMap集合（之前学过的所有集合）：都可以存储null值，null键

Hashtable集合，**不能**存储null值，**也不能**存储null键

Hashtable和Vector集合一样，在jdk1.2版本之后就被更先进的集合（HashMap,ArrayList）替代。

Hashtable的子类Properties依然活跃在历史舞台。

Properties集合是**唯一一个**和IO流相结合的**集合**。



8.JDK9的新特性：

List接口，Set接口，Map接口：里面增加了一个静态方法of,可以一次性的给集合添加多个元素。

static<E> List<E> of(E… elements)

**使用前提：**

当集合中存储的元素的**个数**已经**确定**了，**不在改变**时才能使用。

注意：

1.of方法只适用于List接口，Set接口，Map接口，不适合接口的实现类。

2.of方法的返回值是一个不能改变的集合，集合不能再使用add,put方法添加元素，会抛出异常。

3.Set接口和Map接口在调用of方法的时候，不能有重复的元素，否则会抛出异常。

