1. 集合中的遍历

Collection<String> c=new ArrayList<>();

c.add(“hello”);

c.add(“world”);

Iterator<String> it=c.iterator();

while(it.hasNext()){

String s=it.next();

System.out.println(s);

}

1. List

2.1有序集合，可以重复 ，通过整数索引访问元素，并搜索列表中的元素

2.2 void add(int index,E element) E remove(int index) E get(int index)

E set(int index ,E element)

2.3并发修改异常

原因：迭代遍历的过程中，通过集合对象修改了集合中的元素，造成了迭代器获取元素中怕判断预期修改值和实际修改值不一致ConcurrentModificationException

解决：用for循环

for(int i=90;i<list.size();i++){

String s=list.get(i);

if(s.equals(“world”);{

list.add(0,”java”);

}

}

System.out.println(list);

* 1. ListIterator:允许程序员沿着任一方遍历

ListIterator<String> it=list.listIterator();

while(it.hasNext()){

String s=it.next();

if(s.equals(“world”)){

list.add(“java”);

}

}

2.5

|  |  |
| --- | --- |
| 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 void setName(String name){  this.name=name;  }  public void setAge(int age){  this.age=age;  }  public String getName(){  return name;  }  public int getAge(){  return age;  }  @Override  public Boolean equals(Object o){  if(this==o){  return true;  }  if(o==null||getClass()!=o.getClass()){  return false;  }  Student stu=(Student) o;  Return this.name.equals(stu.name)&&this.age==stu.age;  }  } | 测试类  public class ListDemo{  public static void main(String[] args){  List<Student>list=new ArrayList<Student>();  Student s1=new Student(“张三”,18);  Student s2=new Student(“王五”,20);  Student s3=new Student(“赵六”,19);  list.add(s1);  list.add(s2);  list.add(s3);  Iterator<Student> it=list.iterator();  While(it.hasNext()){  Student a=it.next();  System.out.println(s.getName()+”,”s.getAge());  }  for(int i=0;i<list.size();i++){  Student s=list.get(i);  System.out.println(s.getName()+”,”s.getAge());  }  for(Student s:list){  System.out.println(s.getName()+”,”s.getAge());  }  System.out.println(list.contains(new Student(new Student(“小强”,78)));  List.remove(new Student(“小强”,78));  list.forEach(System.out:println);  }  } |

* 1. ArrayList和LinkedList都线程不安全

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  public class demo{  public static void main(String args[]){  List<Integer>list=new ArrayList<>();  List<Integer>list2=new ArrayList<>();  list.add(1);  list.add(2);  list.add(0,10); //在0位置插入10  int num=list.get(0); //10  int num1=list.indexOf(0); //10  int num2=list.indexOf(6); //-1  boolean b=list.contains(new Integer(1)); //true  boolean b1=list.contains(1); //true    boolean f=list.isEmpty();  list1.addAll(list); //将集合list添加到list1中  }  } |

使用get()根据索引获取数据时，ArrayList的时间复杂度为O(1),LinkedList为O(n)

Vector和ArrayList的内部机制一样，但是线程安全有synchronized 性能不好

|  |  |
| --- | --- |
| ArrayList内部是动态数组实现，在增加空间时会复制全部数据到新的容量大一些的新开辟的数组中。慢 会产生大量垃圾，性能下降。适合顺序添加，随机访问 | 而LinkedList内部为双向链表，可以按需分配空间，扩展容量简单，快  因为LinkedList获取数据是需要移动指针，不适合查询。 |
| 1.9之前：默认的构造函数会默认开辟大小为10的数组，容量不足时会成2倍增长。  1.9之后：默认的构造函数会使用空数组，使用时开辟默认容量10 | public void addFirst(E e)  public void addLast(E e)  public E getFirst()  public E getLast()  public E removeFirst()  public E removeLast() |
| 删除元素时，需要做一次元素复制操作，如果要赋值的元素很多，就会比较耗费性能。  插入元素的时候，也需要做一次元素复制操作 |  |

**ArrayList和LinkedList的区别**

数据结构实现：ArrayList是动态数组的实现，而LinkedList是双向链表的实现

随机访问效率：ArrayList比LinkedList在随机访问的时候效率更高，因为LinkedList是线性的数据存储方式，所以需要移动指针从前往后依次查找。

增加和删除效率：在非首尾的增加和删除操作，LinkedList比ArrayList效率高，因为ArrayList增删操作要影响数组内的其他数据的下标。

内存空间占用：LinkedList比ArrayList更占内存，LinkedList的节点除了存储数据，还存储了两个引用，一个向前，一个向后

线程安全：都不保证线程安全

综合来说，需要频繁的读取集合中的元素时，更推荐ArrayList，而在插入和删除操作较多时，更推荐LinkedList

**多线程场景下如何使用ArrayList**

List<String> synchronizedList=Collections.synchronizecList(list);

synchronizedList.add(“aaa”);

synchronizedList.add(“bbb”);

for(int i=0;i<sychronizedList.size():i++){

System.out.println(sychronizedList.get(i));

}

**底层源码分析：**

ArrayList中的数组的定义：

private transient Object[] elementData;

ArrayList的定义：

public class ArrayList<E> extends AbstractList<E> implements List<E>,RandomAccess,

Cloneable,java.io.Serializable

ArrayList支持序列化，transient的作用是不希望elementData数组被序列化，重写了writeObject实现

private void writeObject(java.io.ObjectOutputStream s)throws java.io.IOException{

int exceptionModCount=modCount;

s.defaultWriteObject();

s.writeInt(elementData.length);

for(int i=0;i<size;++)

s.writeObject(elementData[i]);

if(modCount!=exceptedModCount){

throw new ConcurrentModificationException();

}

每次序列化时，先调用defaultWriteObject()方法序列化ArrayList中的非transient元素，然后遍历elementData，只序列化已存入的元素 ，这样既加快了序列化的速度，又减小了序列化后的文件大小

**数组和List之间的转换**

Arrays.asList(array) li.toArray()

Set

3.1 不能重复，没有索引，只能用迭代器或for-each

public class SetDemo{

public static void main(String[] args){

Set<String> set=new HashSet<String>();

set.add(“hello”);

set.add(“world”);

for(String s:set){

System.out.println(s);

}

}

}

3.2 哈希值简介：是JDK根据对象的地址或字符串或者数字算出来的int类型的数值，通过Object类中的public int hsahCode()返回哈希码。同一个对象多次调用此方法韩辉的哈希值是相同的。不同对象的哈希值是不同的。而重写此方法可以事项让不同对象的哈希值相同

3.3HashSet集合保证元素唯一性的源码分析：无序

根据对象的哈希值计算存储位置，如果当前位置没有元素则直接存入，如果当前位置有元存在则将要存入的元素与已存在的元素比较哈希值，如果哈希值不同则将当前元素存入，如果哈希值相等则用equals()方法比较两个元素的内容，如果内容不相同，则将当前元素存入，如果相同则不存入。

|  |  |
| --- | --- |
| 学生类  public class Student{  private String name;  private int age;  public Student(){  }  Public Stduent(String name int age){  this.nama=name;  this.age=age;  }  //setter getter方法略  @Override  Public Boolean equals(Object o){  If(this==o){  return true;  }  if(o==null||getClass()!=o.getClass()){  return false;  }  Student student=(Student)o;  return this.name.equals(stu.name)&&this.age==stu.age;  }  @Overrride  public int hashCode(){  int result=name!=null?name.hashCode():0;  result=31\*result+age;  return result;  }  } | 测试类：  public class HashSetDemo{  public static void main(String[] args){  HashSet<Student>hs=new HashSet<Student>();  Student s1=new Student(“张三”,18);  Student s2=new Student(“王五”,20);  hs.add(s1);  hs.add(s2);  for(Student s:hs){  System.out.println(s.getName()+”,”+s.getAge());  }  }  } |

**HashSet的实现原理：**

HashSet是基于HashMap实现的，HashSet的值存放于HashMap的key上，HashMap的value统一为PRESENT，因此，HashSet的实现比较简单，不允许重复值。

**HashSet如何检查重复，并保证不重复：**

向HashSet中add()元素时，判断元素是否存在的依据，**不仅要比较hash值，同时还要结合equals方法比较**。HashSet中的add()会调用HashMap中的put()方法。HashMap中的key是唯一的。

|  |
| --- |
| HashSet的源码；  private static final Object PRESENT=new Object();  private transient HashMap<E,Object> map;  public HashSet(){  map=new HashMap<>();  }  public boolean add(E e){  return map.put(e,PRESENT)==null；  } |

3.4TreeSet集合：

有序：具体的排序方式取决于构造方法

* TreeSet():根据其元素Nederland自然排序进行排序
* TreeSet（Comparator comparator）：根据指定的比较器进行排序

没有带索引的方法，所以不能使用普通的for循环遍历

Comparable

|  |  |
| --- | --- |
| 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 void setName(String name){  this.name=name;  }  public String getName(){  return name;  }  public void setAge(int age){  this.age=age;  }  public int getAge(){  return age;  }  @Override  public int compareTo(Student s){  int num=this,age-s.age; //升序  int num2=num==0?this.name.compare(s.name):num;  return num2;  }  } | public class TreeSetDemo{  public static void main(String[] args){  TreeSet<Student> ts=new TreeSet<Student>();  Student s1=new Student(“xishi”,29);  Student s2=new Student(“wangzhaojun”,28);  Student s3=new Student(“diaochan”,30);  Student s4=new Student(“yangguifei”,33);  Student s5=new Student(“linqinxia”,33);  Student s6=new Student(“lingqinxia”,33);  ts.add(s1);  ts.add(s2);  ts.add(s3);  ts.add(s4);  ts.add(s5);  ts.add(s6);  for(Student s:ts){  System.out.println(s.getName()+”,”+s.getAge());  }  }  } |

Comparator

|  |  |
| --- | --- |
| public class Student {  private String name;  private int age;  public Student(){  }  Public Student(String name,int age){  this.name=name;  this.age=age;  }  public void setName(String name){  this.name=name;  }  public String getName(){  return name;  }  public void setAge(int age){  this.age=age;  }  public int getAge(){  return age;  }  } | public class TreeSetDemo{  public static void main(String[] args){  TreeSet<Student> ts=new TreeSet<Student>(new Comparator<Student>(){  @Override  public int compare(Stduent s1,Studnt s2){  int num=s1.getAge()-s2.getAge();  int num2=num==0?s1.getName().compareTo(s2.getName()):num;  return num2;  }  });  Student s1=new Student(“xishi”,29);  Student s2=new Student(“wangzhaojun”,28);  Student s3=new Student(“diaochan”,30);  Student s4=new Student(“yangguifei”,33);  Student s5=new Student(“linqinxia”,33);  Student s6=new Student(“lingqinxia”,33);  ts.add(s1);  ts.add(s2);  ts.add(s3);  ts.add(s4);  ts.add(s5);  ts.add(s6);  for(Student s:ts){  System.out.println(s.getName()+”,”+s.getAge());  }  } |

**不重复的随机数案例**

public class SetDemo{

public static void main(String[] args){

Set<Integer> set =new TreeSet<Integer>();

Random r=new Random();

while(set.size()<10){

int number=r.nextInt(20)+1;

set.add(number);

}

for(Integer i:set){

System.out.println(i);

}

}

}

Map集合

保证线程安全：Map map = new ConcurrentHashMap();

**重要方法**

Set<Map.Entry<K,V>>entrySet() 获取所有键值对对象的集合

Collection values() 获取所有值的集合 Set keyset() 获取所有键的集合

**集合的遍历**

|  |  |
| --- | --- |
| public class MapDemo{  public static void main(String[] args){  Map<String,String> map=new HashMap<String,String>();  map.put(“张无忌”,”赵敏”);  map.put(“郭靖”,”黄蓉”);  map.put(“杨过”,”小龙女”);  Set<String> keyset=map.keySet();  for(String key:keyset){  String value=map.get(key);  System.out.println(key+”,”+value);  }  }  } | public class MapDemo{  public static void main(String[] args){  Map<String,String> map=new HashMap<String,String>();  map.put(“张无忌”,”赵敏”);  map.put(“郭靖”,”黄蓉”);  map.put(“杨过”,”小龙女”);  Set<Map.Entry<String,String>>entrySet=map.entrySet();  for(Map.Entry<String,String> me:entrySet){  String key=me.getKey();  String value=me.getValue();  System.out.println(key+”,”+value);  }  }  } |

**HashMap：线程不安全，是内部基于哈希表实现，该类继承AbstractMap，实现了map接口**

**key可以保存null和重复（会出现内容覆盖 ）value允许为null**

**HashTable：继承了Dictionary类，线程安全**

**不许有null**

**Properties类继承了Hsahtable类，**

**TreeMap：**

**key不许为null，value可以为 null**

**ConcurrentMap**

**key：不许为null value不许为null**

**集合框架底层数据结构**

List:

* ArrayList:Object数组
* Vector：Object数组，线程安全（相比ArrayList多了同步化机制 ）
* LinkedList：双向循环链表

Set：

* HashSet（无序，唯一）：基于HashMap实现的，底层采用HashMap来保存元素
* LinkedHashSet：继承了HashSet，内部通过LinkedHsahMap实现
* TreeSet（有序，唯一）：红黑树

Map：

* HashMap：数组+链表，数组是主体，链表长度大于阈值（8）时，，但是数组产度小于64时会先扩容，否则将链表转化为红黑树，以减少搜索时间。
* LinkedHashMap：继承HashMap。所以它的底层仍然是基于拉链式散列结构（即由数组和链表或红黑树组成），另外还增加了一条双向链表，使得上面的结构可以保持键值对的插入顺序。
* HashTable：数组+链表，数组是主体，链表解决哈希冲突。线程安全（相比hashmap多了线程安全）
* TreeMap：红黑树

stack和enumeration都是线程安全的

**集合的快速失败机制“fail-fast”**

Java集合的一种错误检测机制，当多个线程对集合进行结构上的改变时的操作时，有可能会产生。例如：假设存在两个线程（线程1，线程2），线程1通过Iterator在遍历集合A中的元素，在某个时候线程2修改了集合A的结构（不是简单的修改内容），那么就会抛出异常ConcurrentModificationException,从而产生fail-fast机制

原因：迭代器在遍历时直接访问集合中的内容，并且在遍历过程中使用一个modCount变量。集合在被遍历期间如果内容发生变化，就会改变modCount的值。每当迭代器使用hasNext（）/next（）遍历下一个元素之前，都会检测modCount变量是否为expectedmodCount值，是的话就返回遍历；否则抛出异常终止遍历。

解决：1.在遍历过程中，所有涉及到改变modCount值的地方全部加上sychronized

2.使用CopyOnWriteArrayList来替代ArrayList

边遍历便删除

Iterator<Integer> it=list.iterator();

while(it.hasNext()){

it.remove();

}

**怎么确保一个集合不能被修改**

可以使用Collections.unmodifiableCollection（Collection c）方法来创建一个只读集合，这样改变集合的任何操作都会抛出UbsupportedOperationException异常

List<String> list=new ArrayList<>();

list.add(“x”);

Collection<String> clist=Collextions.unmodifiableCollection(list);

list.add(“y”); //运行时报错

System.out.println(list.size());

**Iterator和ListIterator的区别**

Iterator可以遍历Set和List集合，而ListIterator只能便利List

Iterator只能单向遍历，而ListIterator可以双向遍历

ListIterator实现Iterator接口，然后添加了一些额外的功能，比如添加一个元素、替换一元素、获取前面或者后面元素的索引位置。