# 1.集合

## 1.1.Collection<E>

本身是一个存储数据的容器---不限定大小

Collection<E>---Java中集合的顶级接口。E表示泛型，用于指定集合中存储元素类型的。集合中只能存储对象。例：Collection<String> c;---表示集合中的元素的类型是String字符串。

### 1.1.1.重要方法

1>add();

添加元素。已经重写了toString方法，把每个元素用“，”拼接成一个字符串。

2>remove();

移除元素。如果移除的元素不存在，会直接忽略。

3>containe();

判断指定的元素是否存在。

4>clear();

清空集合。

5>equals(Object o);

比较。

6>isEmpty();

判断集合是否为空。

7>iterator();

返回在此 collection 的元素上进行迭代的迭代器。

8>size();

获取集合中元素个数。

9>toArray();

将集合转化为数组。数组的类型只能是Object类。

10>toArray(new String[0]);

参数表示要转化的数组类型。 如果传入的数组长度小于了集合中元素的个数，那么会将底层的数组返回。如果传入的数组的长度大于等于了集合中元素的个数，会将底层数组的元素复制到传入的数组中，返回传入的数组。

当传入的数组容量过大时，你就分不清数组中的null到底是元素还是真的空着，所以最好传入最小的数组大小。

11>代码示例

public class CollectionDemo {

public static void main(String[] args) {

// 限定集合中的元素类型为String类型

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

// 添加元素

c.add("dea");

c.add("rst");

c.add("poi");

// 遍历集合

for (String s : c) {

System.out.println(s);

}

// 移除指定的元素---移除之前会对集合中的元素进行判断

// c.remove("des");

// 判断集合中是否包含了指定的元素

// System.out.println(c.contains("rat"));

// 清空集合

// c.clear();

// 判断集合中是否包含元素

// System.out.println(c.isEmpty());

// 获取元素个数---不是集合大小

// System.out.println(c.size());

// 将集合转化为一个数组

// Object[] os = c.toArray();

// for (Object o : os) {

// System.out.println(o);

// }

// 传入的参数只是限定了返回值类型

// String[] os = c.toArray(new String[0]);

// for (String o : os) {

// System.out.println(o);

// }

System.out.println(c);

}

}

## 1.2.List<E> 列表

List接口保证元素的存入顺序，元素有序，可以重复，是Collection<E>的子接口。

在List中会对元素进行编号，编号从0开始，元素存在下标，所以可以通过下标来获取和操作对应的元素。

### 1.2.1.重要方法

1>get(int index);

获取指定下标对应的元素。

2>add(int index, E e);

指定的下标处，插入指定的元素。在下标index的位置插入d。index <= size。index > size就会报错。

3>equals(object o)

依次调用每个位置上的元素的equals方法来比较的。两个集合在进行比较的时候和元素的顺序有关。

底层过程：

1.比较两个列表的元素个数是否一致，如果不一致，返回false。

2.如果元素个数一致，则一次比较对应下标位置上的元素。底层比较元素的时候用的是equals方法。

凡是引用类型的对象，除非确定不存在为null的可能，否则一律先==后equals。

4>indexOf(object o);

获取指定元素第一次出现的下标。如果元素不存在就返回-1。

5>lastIndexOf(object o);

最后一次出现的下标。

6>remove(int index);

移除指定下标的元素。

7>set(int index,E e);

表示替换指定下标位置上的元素。

8>size();

获取列表的大小。

9>subList(int index,int index);

截取列表。包含起始下标，而不包含结束下标。

10>代码示例

public class ListDemo {

public static void main(String[] args) {

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

// 添加元素

list.add("ade");

list.add("edg");

list.add("tyk");

list.add("abc");

list.add("edg");

// 从指定下标开始到指定下标结束，截取出一个子列表

// List<String> slist = list.subList(1, 3);

// System.out.println(slist);

// 替换指定下标位置上的元素

// list.set(5, "def");

// 移除指定下标位置上的元素

// list.remove(8);

// 获取指定的元素在列表中第一次出现的下标

// System.out.println(list.indexOf("edf"));

// 获取指定下标位置上的元素

// System.out.println(list.get(2));

// 遍历列表

// for (int i = 0; i < list.size(); i++) {

// System.out.println(list.get(i));

// }

// List<String> list2 = new ArrayList<String>();

// list2.add(new String("ade"));

// list2.add(new String("edg"));

// list2.add("tyk");

// list2.add("abc");

// 在列表中，和元素的顺序有关

// System.out.println(list.equals(list2));

// 向指定的下标位置上插入指定的元素

// list.add(1,"def");

// list.add(4,"hil");

// list.add(6,"ert");

System.out.println(list);

}

}

### 1.2.2.ArrayList<E>

List的实现类，基于数组的，内存空间是连续的，初始容量是10，每次扩容上次容量的一半，是一个线程不安全的集合，增删元素比较慢，查询元素较快。

练习：用数组实现一个简易版的ArrayList---String

(add/remove/contains/set/get/isEmpty/size/indexOf/toString)

public class ListExer {

private String[] arr;

// 标记数组中的元素个数

// 标记可操作的位置

private int i = 0;

public ListExer() {

arr = new String[10];

}

public ListExer(int capacity) {

// 需要判断容量是否符合要求

if (capacity < 0) {

capacity = 1;

}

arr = new String[capacity];

}

// 扩容

private void grow() {

if (arr.length <= 1) {

arr = Arrays.copyOf(arr, arr.length + 1);

}

if (this.i >= this.arr.length) {

arr = Arrays.copyOf(arr, arr.length + (arr.length >> 1));

}

}

// 添加元素

public void add(String s) {

// 是否需要扩容

this.grow();

arr[this.i] = s;

this.i++;

}

public void add(int index, String s) {

// 判断下标越界

if (index > this.i) {

throw new IndexOutOfBoundsException("Index:" + index + ", Size:" + this.i);

}

// 判断扩容

this.grow();

// for(int i = this.i - 1; i >= index; i--){

// arr[i+1] = arr[i];

// }

System.arraycopy(arr, index, arr, index + 1, this.i - index);

arr[index] = s;

this.i++;

}

// 移除指定下标上的元素

public void remove(int index) {

// 判断越界

if (index >= this.i) {

throw new IndexOutOfBoundsException("Index:" + index + ", Size:" + this.i);

}

// for (int i = index; i < this.i - 1; i++) {

// arr[i] = arr[i + 1];

// }

System.arraycopy(arr, index + 1, arr, index, this.i - index - 1);

this.i--;

}

// 移除指定的元素

public void remove(String s) {

// 获取这个元素的下标

int index = this.indexOf(s);

if (index != -1) {

this.remove(index);

}

}

public void set(int index, String s) {

// 判断越界

if (index >= this.i) {

throw new IndexOutOfBoundsException("Index:" + index + ", Size:" + this.i);

}

arr[index] = s;

}

public String get(int index) {

// 判断越界

if (index >= this.i) {

throw new IndexOutOfBoundsException("Index:" + index + ", Size:" + this.i);

}

return arr[index];

}

// 获取指定元素出现的位置

public int indexOf(String s) {

for (int i = 0; i < arr.length; i++) {

if (arr[i] == s || arr[i] != null && arr[i].equals(s)) {

return i;

}

}

return -1;

}

// 判断是否包含指定元素

public boolean contains(String s) {

return this.indexOf(s) != -1;

}

// 判断集合为空

public boolean isEmpty() {

return this.i <= 0;

}

// 获取元素个数

public int size() {

return this.i;

}

public String toString() {

StringBuilder sb = new StringBuilder("[");

for (int i = 0; i < this.i; i++) {

sb.append(arr[i]).append(", ");

}

String s = sb.toString();

if (s.length() >= 2)

s = s.substring(0, s.length() - 2);

return s += "]";

}

public static void main(String[] args) {

ListExer le = new ListExer(1);

le.add("a");

le.add("b");

le.add("c");

le.add("d");

le.add("e");

le.add("f");

le.add(0, "g");

le.remove(0);

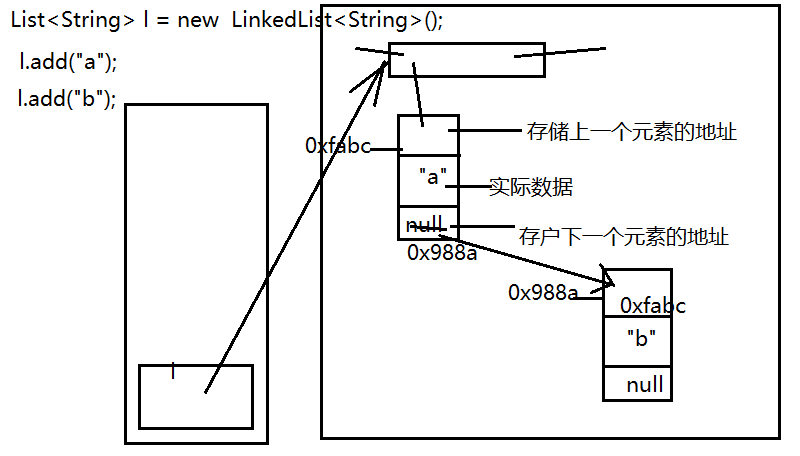
System.out.println(le);

}

}

### 1.2.3.LinkedList<T>

基于链表实现的，内存空间是不连续的，不用设置初始容量，增删元素相对较快，查询元素较慢，是一个线程不安全的集合。



### 1.2.4.Vector

向量---Java中最早的集合---基于数组实现的。默认初始容量是10，每次扩容一倍，是一个线程安全的集合。

#### 1.2.4.1.重要方法

1>capacity();

获取集合的容量。

2>elements();

返回向量组件的枚举。名义上是枚举，其实返回的是迭代器。

### 1.2.5.Stack<E> 继承了Vector

栈的原则是先进后出或者后进先出。

最先放入的元素叫栈底元素。

最后放入的元素叫栈顶元素。

将元素放入栈中的操作叫入栈（压栈）。

将元素从栈中取出叫出栈（弹栈）。

#### 1.2.5.1.重要方法

1)isEmpty();

判断栈是否为空。

2)push();

入栈、压栈。

3)peek();

获取栈顶元素，不删除。

如果栈为空，则抛出EmptyStackException(空栈异常)。

4)pop();

出栈,获取并移除栈顶元素。

如果栈为空，则抛出EmptyStackException(空栈异常)。

5)search();

获取指定元素在栈中的位置。获取的时候从栈顶到栈底的顺序查找，以1位基数。如果元素不存在返回-1。

#### 1.2.5.2.代码示例

Stack<String> s = new Stack<String>();

// 判断栈是否为空

// System.out.println(s.empty());

// 压栈

s.push("a");

s.push("b");

s.push("c");

s.push("d");

// 获取栈顶元素---不移除

// 如果栈为空，则抛出一个EmptyStackException

// System.out.println(s.peek());

// 出栈---获取并移除栈顶元素

// System.out.println(s.pop());

// 获取指定元素在栈中的位置

// 从栈顶到栈底依次查找，以1为基数

System.out.println(s.search("c"));

System.out.println(s);

练习：用数组实现Stack

import java.util.Arrays;

import java.util.EmptyStackException;

public class StackExer {

// 存储数据

private String[] data = new String[10];

// 定义一个变量来记录元素的个数

private int size;

public StackExer() { }

public boolean empty() {

return size <= 0;

}

public void push(String s) {

// 判断数组是否需要进行扩容

if (size >= data.length) {

data = Arrays.copyOf(data, data.length \* 2);

}

data[size++] = s;

}

public String peek() {

// 判断栈是否为空

if (size <= 0) {

throw new EmptyStackException();

}

return data[size - 1];

}

public String pop() {

// 获取栈顶元素

String s = peek();

size--;

return s;

}

public int search(String s) {

// int count = 1;

for (int i = size - 1; i >= 0; i--) {

if (s == data[i] || s != null && s.equals(data[i])) {

// return count;

return size - i;

}

// count++;

}

return -1;

}

}

### 1.2.6.总结

1>ArrayList和数组有什么区别？

ArrayList的容量可变，数组的容量是固定的。ArrayList只可以存储引用类型，数组不做限制。

容量：ArrayList可变，数组固定

元素类型：ArrayList可以存储引用类型，数组不限制。

2>List和数组有什么区别？

List的容量是可变的，数组的容量是固定的，List只能存储引用类型，数组不做限制。数组的内存空间是连续的，List的的内存空间是根据实现类的不同而不同，针对LinkedList而言，内存空间是不连续的，针对ArrayList、Vocter、Stack而言内存空间是连续的。

3>ArrayList和Vector有什么区别？

ArrayList和Vector都是基于数组实现的，并且其底层的数组的默认初始容量都是10.对于ArrayList而言，其扩容机制是基于右移运算的，每次右移1位，所以扩容完成之后，容量就变为原来的1.5倍。对于Vector而言，其扩容机制是基于三元运算求和的，如果不指定增量的话，扩容完成之后就变为原来的两倍。ArrayList是一个线程不安全的集合，但是可以通过Collections.synchronizedList();方法将其包装成一个线程安全的集合。Vector是一个线程安全的集合。

4>LinkedList和ArrayList那个增添数据更快？

如果增删的元素的位置相对靠前，此时LinkedList较快；如果增删的元素的位置靠后，此时ArrayList较快。

如果需要扩容，LinkedList偏快；如果不需要扩容，ArrayList偏快。

## 1.3.Queue接口，队列

队列遵循先进先出的原则。先放入的元素叫队头元素，最后放入的叫队尾元素。

### 1.3.1.重要方法

1>offer();

插入元素。如果失败返回false。

2>add();

插入元素。如果失败则抛出异常。

3>peek();

获取队头元素。如果获取失败，则返回null。

4>element();

获取队头元素。如果获取失败则抛出一个异常

5>poll();

移除队头。如果移除失败则返回一个null。

6>remove();

移除队头。如果失败则返回一个NoElementException。

## 1.4.Collections

是一个操作集合的工具类，提供一系列的操作集合的静态方法。Collections的构造方法都是私有的。

### 1.4.1.重要方法

1>replaceAll(List<T> list,T oldt,T newt);

用一个对象替换另一个对象。

2>reverse(List list);

反转集合。

3>sort(List<T> list);

将集合从小到大排序。

4>Sort(List<T> list,comparator<? super T> c);

指定规则进行排序。

### 1.4.2.代码示例

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

list.add("Grace");

list.add("John");

list.add("Jack");

list.add("Alice");

list.add("Mike");

// 将集合从小到大来排序

// Collections.sort(list);

// 反转集合

// Collections.reverse(list);

// 按照首字母的顺序升序排列

Collections.sort(list, new Comparator<String>() {

// 返回值类型是int

// 如果返回了一个正数，此时表示s1 > s2 -> 默认将s1排到s2后边

// 如果返回了一个负数，此时表示s2 > s1 -> 默认将s2排到s1后边

@Override

public int compare(String s1, String s2) {

return s1.charAt(0) - s2.charAt(0);

}

});

System.out.println(list);

## 1.5.Comparator<T> 比较器

比较器---是一个接口，重写compare方法，将比较规则写到compare方法中---根据返回值的正负来确定大小；如果返回值是正数，表示第一个参数排到第二个参数之后；反之表示第一个参数排到第二个参数之前。

如果没有指定排序规则，这个时候要求集合中的元素对应的类必须实现Comparable，比较规则是写在compareTo方法中。

### 1.5.1.重要方法

1>compareTo(T o1,T o2);

用来排序两个参数。

2>equals(object obj);

指示某个其他对象是否等于此Comparator。

## 1.6.迭代器

### 1.6.1.Iterable<T>

如果一个对象允许用增强for循环来遍历，那么这个对象对应的类必须实现Iterable 接口，增强for（foreach）循环本质上就是在进行迭代遍历。增强for循环不允许对原集合进行增删操作。

Iterable<T> interface 是JDK1.5的特性之一。

#### 1.6.1.1.重要方法

1>iterator();

返回一个在一组 T 类型的元素上进行迭代的迭代器。

### 1.6.2.Enumeration<E>

不要使用这个类的方法，来删除元素。只删除偶数位，永远删除不完。

#### 1.6.2.1.重要方法

1>hasMoreElements();

判断是否还有更多的元素。

2>nextElement();

表示挪动指针到下一个元素。

#### 1.6.2.2.代码示例

Vector<String> v = new Vector<String>();

// System.out.println(v.capacity());

// for (int i = 0; i < 21; i++) {

// v.add("a");

// }

// 获取集合的容量

// System.out.println(v.capacity());

v.add("abc");

v.add("asd");

v.add("wed");

v.add("def");

// 获取迭代器对象

Enumeration<String> e = v.elements();

// 判断是否有后续元素

while (e.hasMoreElements()) {

// 挪动指针获取该元素

String s = e.nextElement();

System.out.println(s);

}

System.out.println(v);

### 1.6.3.Iterator<E>

推荐使用的迭代器。它在操作集合的时候，将集合进行了复制，操作的是复制之后的集合，在复制后的集合上进行迭代标记操作。在迭代器操作集合的时候，不允许原集合自己进行增删操作。

#### 1.6.3.1.重要方法

1>hasNext;

判断是否有可迭代的元素。

2>next；

返回迭代的下一个元素。

3>remove();

表示将当前在迭代的元素进行移除。会改变原集合。

#### 1.6.3.2.代码示例

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

set.add("ab");

set.add("bc");

set.add("cd");

set.add("de");

set.add("ef");

// 获取迭代器对象

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

// 判断是否有下一个元素

while (it.hasNext()) {

// 挪动指针，获取元素

String s = it.next();

System.out.println(s);

// 表示移除在迭代的元素

// it.remove();

// 在迭代过程中不允许对原集合进行增删操作

// set.remove(s); //报错。

}

System.out.println(set);

# 2.泛型

## 2.1.介绍

ParameterizedType，学名参数化类型。JDK1.5特性之一。泛型使得代码可以适应多种类型。像容器，List< T >，大量使用了泛型，它的主要目的之一就是用来指定容器要持有什么类型的对象。

1>泛型发展历史

①List list = new ArrayList();：早期没有泛型集合可以存储任意数据，底层以Object类型来存储。

②List list = new ArrayList<E>();：仅仅起了一个建议性的作用。

③List<E> list = new ArrayList<E>();：强制元素类型必须是指定的类型。

④List<E> list = new ArrayList<>();：JDK1.7之后可以省去后面泛型，JVM在编译的时候会做自动类型推导。

## 2.2.泛型的擦除

用具体类型来替换泛型的过程叫泛型的擦除。发生在编译期。

## 2.3.定义泛型类

### 2.3.1.泛型命名

泛型的命名只要遵循标识符的命名规则即可。习惯上用一个大写字母来表示。

T ---type，类型。

E ---element，元素。

K ---key，键。

V ---value，值。

R ---result，结果。

因为泛型不确定具体类型，所以只允许声明，不允许初始化。

泛型可以作为参数来使用，同样也可以作为返回值使用。

泛型在定义的时候不限制个数，只要以逗号隔开就行。

①类的泛型

表示属于当前类的泛型。

class Demo<T,E,R>{}

②方法的泛型

表示只属于当前方法的泛型。

public <V>void m(V v){}

③返回值的泛型

public <R> R m(){}

### 2.3.2.泛型的继承

①兼容

Arrays.asList();允许以操作list的方式来操作一个数组。

数组允许存在向下兼容，也就是说数组中的元素类型可以存在继承关系。

Object[] os = new String[10];

在泛型中不存在向下兼容，是指在声明的时候前后必须一致。

ArrayList<Object> list = new ArrayList<String>();//这个是错误的。

②通配符

“?”在泛型的继承中是一个通配符。

List<?> list;：表示这个集合的元素类型暂时不确定，往往作为参数。

③泛型的上限

<? extends 类/接口>

表示传入的是类/接口及其子类/子接口。这就是规定了泛型的上限。在规定上限方法里，为来防止添加的元素与原类型不一致，只能添加null，不能添加其他元素。

④泛型的下限

<? super 类/接口>

表示传入的这个类/接口及其父类/父接口对象。这就是规定了泛型的下限。在规定下限方法里，能添加元素，元素类型必须是最小类型，null也可以添加。

Java中不允许同时规定上限和下限。可以使用两个方法进行这个上限下限规定的操作。

⑤泛型继承嵌套

Java中允许互相嵌套。

代码示例：

public static void main(String[] args) {

// 以操作List的方式来操作一个数组

List<Integer> ins = Arrays.asList(2, 1, 5, 6, 1, 8, 0);

List<Double> dos = Arrays.asList(2.1, 3.34, 6.53, 7.12, 5.2);

print(ins);

print(dos);

}

// 遍历元素为数值类型的List

// ? extends 类/接口 表示传入的是类/接口及其子类/子接口

// ? extends Number --- 表示传入的是Number及其子类对象

// 规定了泛型的上限

public static void print(List<? extends Number> list) {

// 能添加元素吗？--不能，除了null

list.add(null);

for (Number number : list) {

System.out.println(number);

}

}

// 传入String及其父类对象

// ? super 类/接口 表示传入这个类/接口及其父类/父接口对象

// ? super String 表示传入的String及其父类或者父接口对象

// 规定了泛型的下限

public static void m(List<? super String> list) {

// 能添加元素吗？---能

list.add("a");

list.add(null);