第06天 集合

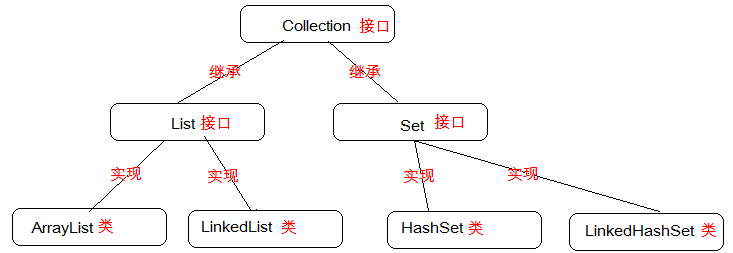
今日内容介绍

* 集合&迭代器
* 增强for & 泛型
* 常见数据结构
* List子体系

# 集合&迭代器

## 集合体系结构

### 集合体系图



在最顶层的父接口Collection中一定定义了所有子类集合的共同属性和方法,因此我们首先需要学习Collection中共性方法,然后再去针对每个子类集合学习它的特有方法

### 案例代码一:

**package** com.itheima\_01;

**import** java.util.ArrayList;

/\*

\* ArrayList

\* 集合的体系结构：

\* 由于不同的数据结构（数据的组织，存储方式），所以Java为我们提供了不同的集合，

\* 但是不同的集合他们的功能都是相似，不断的向上提取，将共性抽取出来，这就是集合体系结构形成的原因

\*

\* 体系结构：

\* 怎么学习?最顶层开始学习，因为最顶层包含了所有的共性

\* 怎么使用？使用最底层，因为最底层就是具体的实现

\*

\* Collection

\* List

\* ArrayList

\*/

**public** **class** CollectionDemo {

**public** **static** **void** main(String[] args) {

//创建集合对象

ArrayList al = **new** ArrayList();

//添加元素

al.add("hello");

al.add("world");

al.add("java");

//遍历集合

**for**(**int** x = 0;x < al.size();x++) {

System.*out*.println(al.get(x));

}

}

}

## Collection中的常用功能

boolean add(Object e): 向集合中添加元素

void clear():清空集合中所有元素

boolean contains(Object o):判断集合中是否包含某个元素

boolean isEmpty():判断集合中的元素是否为空

boolean remove(Object o):根据元素的内容来删除某个元素

int size():获取集合的长度

Object[] toArray():能够将集合转换成数组并把集合中的元素存储到数组中

### 案例代码二:

**package** com.itheima\_01;

**import** java.util.ArrayList;

**import** java.util.Collection;

/\*

\* Collection

\* boolean add(E e)

\* void clear()

\* boolean contains(Object o)

\* boolean isEmpty()

\* boolean remove(Object o)

\* int size()

\* Object[] toArray()

\* Iterator<E> iterator()

\*

\*/

**public** **class** CollectionDemo2 {

**public** **static** **void** main(String[] args) {

//创建集合对象

//Collection c = new Collection();//Collection是接口，不能实例化

Collection c = **new** ArrayList();//多态，父类引用指向子类对象

//boolean add(E e)

System.*out*.println(c.add("hello"));//永远可以添加成功，因为ArrayList他允许重复

System.*out*.println(c.add("world"));

//void clear()：清空集合

//c.clear();

//boolean contains(Object o) :判断集合中是否包含指定元素

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

//boolean isEmpty() :是否为空

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

//boolean remove(Object o) :删除元素

//System.out.println(c.remove("java"));

//int size() :返回集合中的元素个数

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

//Object[] toArray() :将集合转换成一个Object类型的数组

Object[] objs = c.toArray();

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

System.*out*.println(objs[i]);

}

System.*out*.println(c);

}

}

## 迭代器

java中提供了很多个集合，它们在存储元素时，采用的存储方式不同。我们要取出这些集合中的元素，可通过一种通用的获取方式来完成。

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，继续在判断，如果还有就再取出出来。一直把集合中的所有元素全部取出。这种取出方式专业术语称为迭代。

集合中把这种取元素的方式描述在Iterator接口中。Iterator接口的常用方法如下

hasNext（）方法：判断集合中是否有元素可以迭代

next（）方法：用来返回迭代的下一个元素，并把指针向后移动一位。

### 案例代码三:

**package** com.itheima\_02;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Iterator;

/\*

\* 集合的遍历方式：

\* 1.toArray(),可以把集合转换成数组，然后遍历数组即可

\* 2.iterator(),可以返回一个迭代器对象，我们可以通过迭代器对象来迭代集合

\*

\* Iterator：可以用于遍历集合

\* E next() :返回下一个元素

\* boolean hasNext() ：判断是否有元素可以获取

\*

\* 注意：Exception in thread "main" java.util.NoSuchElementException

\* 使用next方法获取下一个元素，如果没有元素可以获取，则出现NoSuchElementException

\*/

**public** **class** IteratorDemo {

**public** **static** **void** main(String[] args) {

//method();

//创建集合对象

Collection c = **new** ArrayList();

//添加元素

c.add("hello");

c.add("world");

c.add("java");

//获取迭代器对象

Iterator it = c.iterator();

//Object next() :返回下一个元素

//boolean hasNext() ：判断是否有元素可以获取

/\*if(it.hasNext())

System.out.println(it.next());

if(it.hasNext())

System.out.println(it.next());

if(it.hasNext())

System.out.println(it.next());

if(it.hasNext())

System.out.println(it.next());\*/

**while**(it.hasNext()) {

System.*out*.println(it.next());

}

}

**private** **static** **void** method() {

//创建集合对象

Collection c = **new** ArrayList();

//添加元素

c.add("hello");

c.add("world");

c.add("java");

//获取数组

Object[] objs = c.toArray();

//遍历数组

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

System.*out*.println(objs[i]);

}

}

}

## 并发修改异常:

并发修改异常产生原因:

当使用迭代器遍历集合的时候,使用了集合中的 增加/删除 方法,导致并发修改异常产

### 案例代码四:

**package** com.itheima\_02;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.ListIterator;

/\*

\* 需求：判断集合中是否包含元素java，如果有则添加元素android

\* Exception in thread "main" java.util.ConcurrentModificationException:并发修改异常

\* 迭代器是依赖于集合的，相当于集合的一个副本，当迭代器在操作的时候，如果发现和集合不一样，则抛出异常

\*

\* 解决方案:

\* 你就别使用迭代器

\* 在使用迭代器进行遍历的时候使用迭代器来进行修改

\*/

**public** **class** IteratorDemo3 {

**public** **static** **void** main(String[] args) {

//method();

//创建集合对象

//Collection c = new ArrayList();

List c = **new** ArrayList();

//添加元素

c.add("hello");

c.add("world");

c.add("java");

//我们可以通过遍历来获取集合中的每一个元素，然后进行比较即可

/\*Iterator it = c.iterator();

while(it.hasNext()) {

String s = (String)it.next();

if(s.equals("java")) {

c.add("android");

}

}\*/

ListIterator lit = c.listIterator();

**while**(lit.hasNext()) {

String s = (String)lit.next();

**if**(s.equals("java")) {

lit.add("android");

}

}

System.out.println(c);

}

**private** **static** **void** method() {

//创建集合对象

Collection c = **new** ArrayList();

//添加元素

c.add("hello");

c.add("world");

c.add("java");

//判断集合中是否包含元素java

**if**(c.contains("java")) {

c.add("android");

}

System.out.println(c);

}

}

并发修改异常解决方案:

A:不使用迭代器遍历集合,就可以在遍历的时候使用集合的方法进行增加或删除

B:依然使用迭代器遍历,那么就需要使用Iterator的子接口ListIterator来实现向集合中添加

# 增强for&泛型

## 泛型

### 泛型的引入

在前面学习集合时，我们都知道集合中是可以存放任意对象的，只要把对象存储集合后，那么这时他们都会被提升成Object类型。当我们在取出每一个对象，并且进行相应的操作，这时必须采用类型转换

#### 案例代码五:

**package** com.itheima\_03;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Iterator;

/\*

\* 使用集合存储自定义对象并遍历

\* 由于集合可以存储任意类型的对象，当我们存储了不同类型的对象，就有可能在转换的时候出现类型转换异常，

\* 所以java为了解决这个问题，给我们提供了一种机制，叫做泛型

\* \*

\*/

**public** **class** GenericDemo {

**public** **static** **void** main(String[] args) {

//创建集合对象

Collection c = **new** ArrayList();

//创建元素对象

Student s = **new** Student("zhangsan",18);

Student s2 = **new** Student("lisi",19);

//添加元素对象

c.add(s);

c.add(s2);

//遍历集合对象

Iterator it = c.iterator();

**while**(it.hasNext()) {

String str = (String)it.next();

System.*out*.println(str);

}

}

}

**class** Student {

String name;

**int** age;

**public** Student(String name,**int** age) {

**this**.name = name;

**this**.age = age;

}

}

以上代码会发生强制转换异常,原因就是String str = (String)it.next() ,存入集合的是Student,而强转为String,String与Student之间没有任何子父关系不能强转,未使用泛型前有可能发声强制转换异常的问题

### 泛型的使用

当类上定义<>的时候就可以使用泛型,例如ArrayList类的定义:

class ArrayList<E>,那么我们在创建ArrayList对象的时候就可以指定<>中E的类型

ArrayList<String> al=new ArrayList<String>(),那么String就把E替换掉了

#### 案例代码六:

**package** com.itheima\_03;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Iterator;

/\*

\* 使用集合存储自定义对象并遍历

\* 由于集合可以存储任意类型的对象，当我们存储了不同类型的对象，就有可能在转换的时候出现类型转换异常，

\* 所以java为了解决这个问题，给我们提供了一种机制，叫做泛型

\*

\* 泛型：是一种广泛的类型，把明确数据类型的工作提前到了编译时期，借鉴了数组的特点

\* 泛型好处：

\* 避免了类型转换的问题

\* 可以减少黄色警告线

\* 可以简化我们代码的书写

\*

\* 什么时候可以使用泛型？

\* 问API，当我们看到<E>，就可以使用泛型了

\*

\*/

**public** **class** GenericDemo {

**public** **static** **void** main(String[] args) {

//创建集合对象

Collection<Student> c = **new** ArrayList<Student>();

//创建元素对象

Student s = **new** Student("zhangsan",18);

Student s2 = **new** Student("lisi",19);

//添加元素对象

c.add(s);

c.add(s2);

//遍历集合对象

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

**while**(it.hasNext()) {

//String str = (String)it.next();

//System.out.println(str);

Student stu = it.next();

System.*out*.println(stu.name);

}

}

}

**class** Student {

String name;

**int** age;

**public** Student(String name,**int** age) {

**this**.name = name;

**this**.age = age;

}

}

## 增强for

增强for循环是JDK1.5以后出来的一个高级for循环，专门用来遍历数组和集合的。它的内部原理其实是个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){

}

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集合元素进行增删操作。

### 案例代码七:

**package** com.itheima\_04;

**import** java.util.ArrayList;

**import** java.util.Collection;

/\*

\* foreach:增强for循环，一般用于遍历集合或者数组

\* 格式：

\* for(元素的类型 变量 : 集合或者数组对象) {

\* 可以直接使用变量;

\* }

注意：在增强for循环中不能修改集合，否则会出现并发修改异常。

public interface Iterable<T>

实现这个接口允许对象成为 "foreach" 语句的目标。

\*/

**public** **class** ForEachDemo {

**public** **static** **void** main(String[] args) {

//创建集合对象

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

//添加元素

c.add("hello");

c.add("world");

c.add("java");

//增强for循环遍历集合

/\*for(Object obj : c) {

System.out.println(obj);

}\*/

/\*for(String s : c) {

System.out.println(s.toUpperCase());

}\*/

**for** (String string : c) {

c.add("android");

System.*out*.println(string);

}

}

}

# 常见数据结构

## 数组

数组，采用该结构的集合，对元素的存取有如下的特点：

查找元素快：通过索引，可以快速访问指定位置的元素

增删元素慢 ,每次添加元素需要移动大量元素或这创建新的数组



## 链表

链表，采用该结构的集合，对元素的存取有如下的特点：

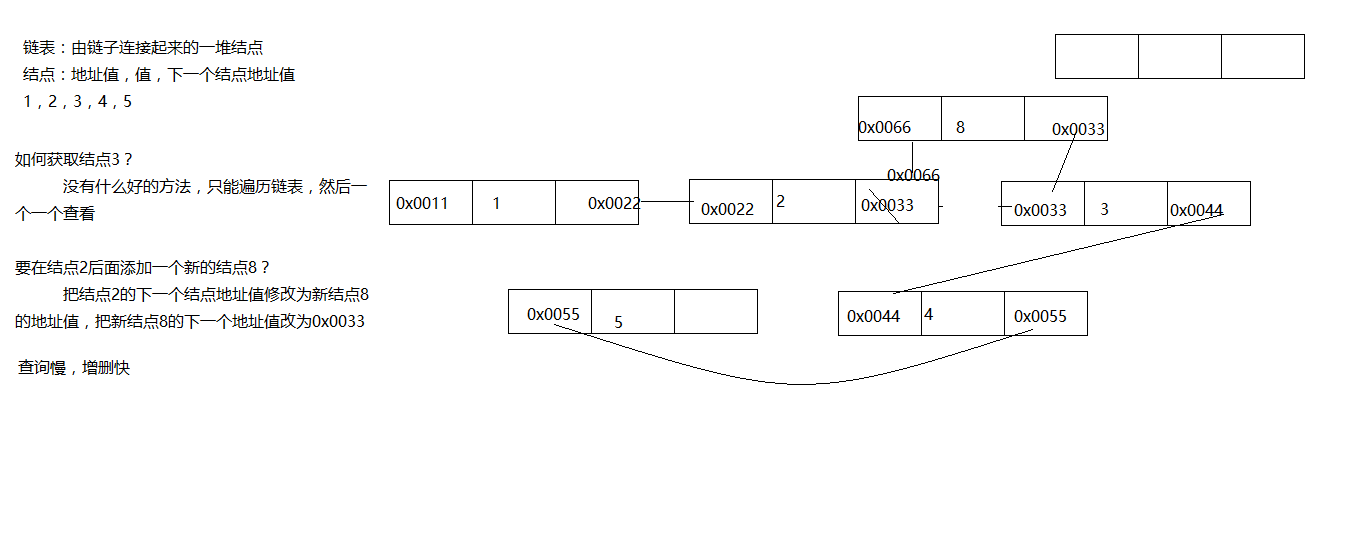
A:多个节点之间，通过地址进行连接。例如，多个人手拉手，每个人使用自己的右手拉住下个人的左手，依次类推，这样多个人就连在一起了。

B:查找元素慢：想查找某个元素，需要通过连接的节点，依次向后查找指定元素

C:增删元素快：

增加元素：只需要修改连接下个元素的地址即可。

删除元素：只需要修改连接下个元素的地址即可



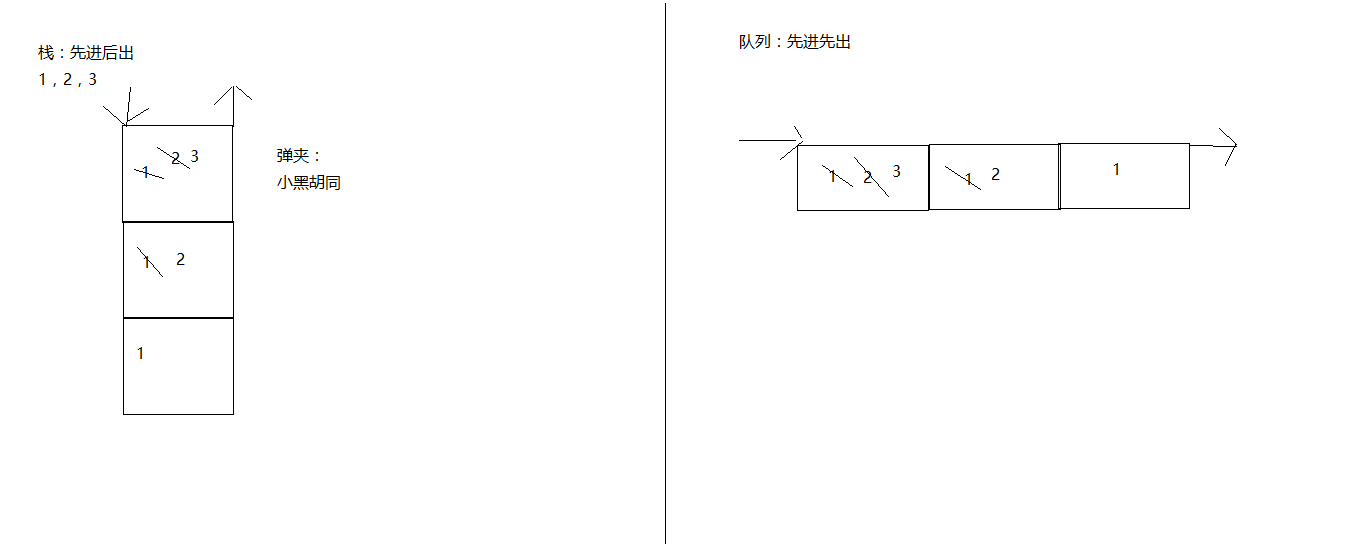
## 栈&队列

A:堆栈，采用该结构的集合，对元素的存取有如下的特点：

先进后出（即，存进去的元素，要在后它后面的元素依次取出后，才能取出该元素）。例如，子弹压进弹夹，先压进去的子弹在下面，后压进去的子弹在上面，当开枪时，先弹出上面的子弹，然后才能弹出下面的子弹。

B:队列，采用该结构的集合，对元素的存取有如下的特点：

先进先出（即，存进去的元素，要在后它前面的元素依次取出后，才能取出该元素）。例如，安检。排成一列，每个人依次检查，只有前面的人全部检查完毕后，才能排到当前的人进行检查。



# List子体系

## List子体系特点

A:有序的（存储和读取的顺序是一致的）

B:有整数索引

C:允许重复的

## List的特有功能

void add(int index, E element) :将元素添加到index索引位置上

E get(int index) :根据index索引获取元素

E remove(int index) :根据index索引删除元素

E set(int index, E element):将index索引位置的的元素设置为element

### 案例代码八:

**package** com.itheima\_05;

**import** java.util.ArrayList;

**import** java.util.List;

/\*

\* List:

\* 有序的（存储和读取的顺序是一致的）

\* 有整数索引

\* 允许重复的

\*

\* List的特有功能：

\* void add(int index, E element)

\* E get(int index)

\* E remove(int index)

\* E set(int index, E element)

\*

\* 增删改查

\*

\*/

**public** **class** ListDemo {

**public** **static** **void** main(String[] args) {

//创建的列表对象

List list = **new** ArrayList();

//void add(int index, E element) : 在指定索引位置添加指定元素

list.add(0, "hello");

list.add(0, "world");

list.add(1, "java");

//E get(int index) :根据索引返回元素

/\*System.out.println(list.get(0));

System.out.println(list.get(1));

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

//System.out.println(list.get(3));

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

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

}\*/

//E remove(int index) : 删除指定元素并返回

//System.out.println(list.remove(5));

//E set(int index, E element) : 将指定索引位置的元素替换为指定元素，并将原先的元素返回

System.*out*.println(list.set(0, "android"));

System.*out*.println(list);

}

}

## LinkedList特有功能

LinkedList底层使用的是链表结构,因此增删快,查询相对ArrayList较慢

void addFirst(E e) :向链表的头部添加元素

void addLast(E e):向链表的尾部添加元素

E getFirst():获取链头的元素,不删除元素

E getLast():获取链尾的元素,不删除元素

E removeFirst():返回链头的元素并删除链头的元素

E removeLast():返回链尾的元素并删除链尾的元素

### 案例代码九:

**package** com.itheima\_06;

**import** java.util.LinkedList;

/\*

\* List的常用子类：

\* ArrayList

\* 底层是数组结构，查询快，增删慢

\* LinkedList

\* 底层结构是链表，查询慢，增删快

\*

\* 如何选择使用不同的集合？

\* 如果查询多，增删少，则使用ArrayList

\* 如果查询少，增删多，则使用LinkedList

\* 如果你不知道使用什么，则使用ArrayList

\*

\* LinkedList的特有功能：

\* void addFirst(E e)

\* void addLast(E e)

E getFirst()

E getLast()

E removeFirst()

E removeLast()

\*

\*/

**public** **class** LinkedListDemo {

**public** **static** **void** main(String[] args) {

LinkedList list = **new** LinkedList();

list.add("hello");

list.add("world");

//void addFirst(E e) :将元素添加到索引为0的位置

//void addLast(E e) ：将元素添加到索引为size()-1的位置

list.addFirst("java");

list.addLast("android");

//E getFirst() :获取索引为0的元素

//E getLast() ：获取索引为size()-1的元素

//System.out.println(list.getFirst());

//System.out.println(list.getLast());

//E removeFirst() :删除索引为0的元素并返回

//E removeLast() ：删除索引为size()-1的元素并返回

System.*out*.println(list.removeFirst());

System.*out*.println(list.removeLast());

System.*out*.println(list);

}

}

## 案例代码九

package com.itheima\_07;

import java.util.ArrayList;

import java.util.List;

/\*

\* 需求：定义一个方法，返回指定列表中指定元素的索引位置

\*

\* 判断元素是否存在

\*

\*/

public class ListTest {

public static void main(String[] args) {

List list = new ArrayList();

list.add("hello");

list.add("world");

list.add("java");

//int index = index(list,"php");

//System.out.println(index);

//boolean flag = contains(list, "php");

//System.out.println(flag);

boolean flag = list.contains("php");

System.out.println(flag);

}

public static int index(List list,Object other) {

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

//获取列表中的元素

Object obj = list.get(x);

//使用列表中的元素和指定的元素进行比较

if(obj.equals(other)) {

return x;

}

}

//查找不到指定的元素

return -1;

}

public static boolean contains(List list,Object other) {

//获取指定元素在指定列表中的索引位置

int index = index(list,other);

//如果索引位置大于等于0，则认为元素存在，否则不存在

if(index >= 0) {

return true;

}

else {

return false;

}

}

}