# day13

## 集合的概述和体系结构

1. 集合 : 就是一个容器, 容器中可以存储很多的数据,与数组容器很相似;

集合是可变长度的容器, 容器大小可伸缩 ; 而数组是定长容器

实际开发场景 : 客户到金融机构柜台查询个人本月的转账流水记录, 流水记录有多少条未知, 因此使用定长数组容器承装, 不太合适; 需要一个长度可变的容器装载流水记录, 于是集合使用合适

1. 集合中存储全部的都是引用数据类型数据, 如果集合中需要存储基本数据类型, 也可以直接存储, 因为JDK1.5特性中, 自动拆装箱可以实现基本类型与其对应包装类自动转换
2. 集合体系结构



## Collection接口

### Collection接口的概述和常用方法

1. Collection 接口 : 来自于java.util包, 是单列集合顶层父接口, 在Collection中的方法功能兼容List和Set都可以使用
2. Collection因为是接口, 不能实例化对象, 于是需要一个实现类, 找到ArrayList

Collection c = new ArrayList(); // 接口的多态

1. Collection中常用功能 :
2. add(Object obj) : 向集合中添加数据obj, 末尾追加
3. remove(Object obj) : 将参数obj从集合中删除
4. clear() : 清空集合, 删除集合中的所有元素, 返回值类型void
5. contains(Object obj) : 验证集合中是否包含元素obj, 包含返回true, 不包含返回false
6. size(): 获取到集合中的元素个数, 返回值类型int
7. isEmpty() : 验证集合是否为空, 如果集合为空, 那么返回true; 如果集合不为空返回false

代码

|  |
| --- |
| package com.ujiuye.collection;  import java.util.ArrayList;  import java.util.Collection;  public class Demo01\_Collection中常用方法 {  public static void main(String[] args) {  Collection c = new ArrayList();  // 1)add(Object obj) : 向集合中添加数据obj, 末尾追加  c.add("abc");  c.add("123");  c.add("hello");    System.out.println(c);//[abc, 123, hello]    // 2)remove(Object obj) : 将参数obj从集合中删除  c.remove("123");  System.out.println(c);//[abc, hello]    // 5)size(): 获取到集合中的元素个数, 返回值类型int  System.out.println(c.size());// 2    // 6)inEmpty():验证集合中的是否为空, 如果集合为空, 那么返回true; 如果集合不为空返回false  System.out.println(c.isEmpty());// false    // 3)clear() : 清空集合, 删除集合中的所有元素, 返回值类型void  c.clear();  System.out.println(c);//[]    // 4)contains(Object obj) : 验证集合中是否包含元素obj, 包含返回true, 不包含返回false  System.out.println(c.contains("abc"));//false    // 5)size(): 获取到集合中的元素个数, 返回值类型int  System.out.println(c.size());    // 6)inEmpty():验证集合中的是否为空, 如果集合为空, 那么返回true; 如果集合不为空返回false  System.out.println(c.isEmpty());// true  }  } |

### 集合的第一种遍历方式---集合转数组

遍历 : 将集合中的数据一个一个获取出来

1. toArray() : 将集合中的数据同步到一个Object[]数组中, 集合转数组; 通过遍历数组相当于遍历集合中的数据

代码

|  |
| --- |
| package com.ujiuye.collection;  import java.util.ArrayList;  import java.util.Collection;  public class Demo02\_集合第一种遍历方式 {  public static void main(String[] args) {  // 1. 创建出一个集合容器  Collection c = new ArrayList();  // 基本数据类型利用自动拆装箱机制, 让基本类型直接封装成对应引用类型存储在集合中  c.add(15);  c.add(-8);  c.add(99);    // 2. 使用toArray方法, 将集合转换成一个Object[]数组  Object[] objArr = c.toArray();    for(int index = 0; index < objArr.length; index++) {  //Type mismatch: cannot convert from Object to Integer  Integer obj = (Integer)objArr[index];  System.out.println(obj);  }  }  } |

### Collection中带All的方法功能

1. addAll(Collection c): 将参数集合c中的所有元素, 都添加到方法调用集合中
2. removeAll(Collection c) : 将参数集合c中的所有元素都从方法调用集合中删除
3. containsAll(Collection c): 验证方法调用集合中, 是否完全包含参数c中的所有元素, 如果完全包含返回true; 否则返回false
4. retainAll(Collection c) : 将方法调用集合与参数集合c中交集数据, 保留到方法调用集合中

removeAll 删除交集数据; retainAll 保留交集数据

代码

|  |
| --- |
| package com.ujiuye.collection;  import java.util.ArrayList;  import java.util.Collection;  public class Demo03\_Collection中All功能 {  public static void main(String[] args) {  Collection c = new ArrayList();  c.add("a");  c.add("b");    Collection c1 = new ArrayList();  c1.add("1");  c1.add("2");    // 1.addAll(Collection c): 将参数集合c中的所有元素, 都添加到方法调用集合中  c.addAll(c1);  System.out.println(c);//[a, b, 1, 2]    // 2.removeAll(Collection c) : 将参数集合c中的所有元素都从方法调用集合中删除  c.removeAll(c1);  System.out.println(c);//[a, b]    // 3.containsAll(Collection c): 验证方法调用集合中, 是否完全包含参数c中的所有元素,如果完全包含返回true; 否则返回false  Collection c2 = new ArrayList();  c2.add("b");  System.out.println(c.containsAll(c2));// true  System.out.println(c.containsAll(c1));// false    // 4. retainAll(Collection c) : 将方法调用集合与参数集合c中交集数据, 保留到方法调用集合中  // c--->[a,b]  // c2--->[b]  /\*c.retainAll(c2);  System.out.println(c);//[b]  \*/    // c--->[a,b]  // c1-->[1,2]  c.retainAll(c1);  System.out.println(c);//[]  }  } |

### 集合第二种遍历方式---迭代器

1. 迭代 : 更迭, 有从一个到下一个的过程
2. 迭代器 : 专门使用在集合之上, 用于将集合中的元素一个一个获取出来的对象
3. 获取集合迭代器对象的方式:

Collection接口中有一个方法功能 :

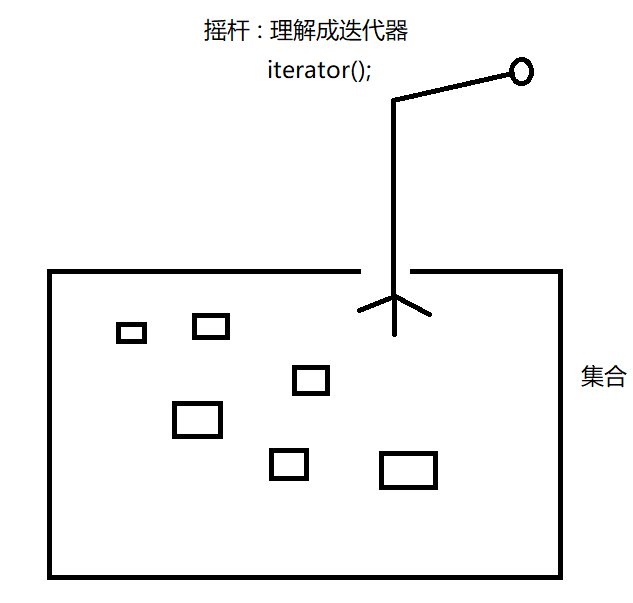
iterator() : 获取到当前集合的迭代器对象, 返回值类型Iterator

1. Iterator接口, 表示迭代器, 来自于java.util包, 方法返回值类型是接口, 实际上这个方法返回的是接口的一个实现类对象
2. 如何通过迭代器遍历(迭代)集合中元素, 需要使用Iterator中方法功能

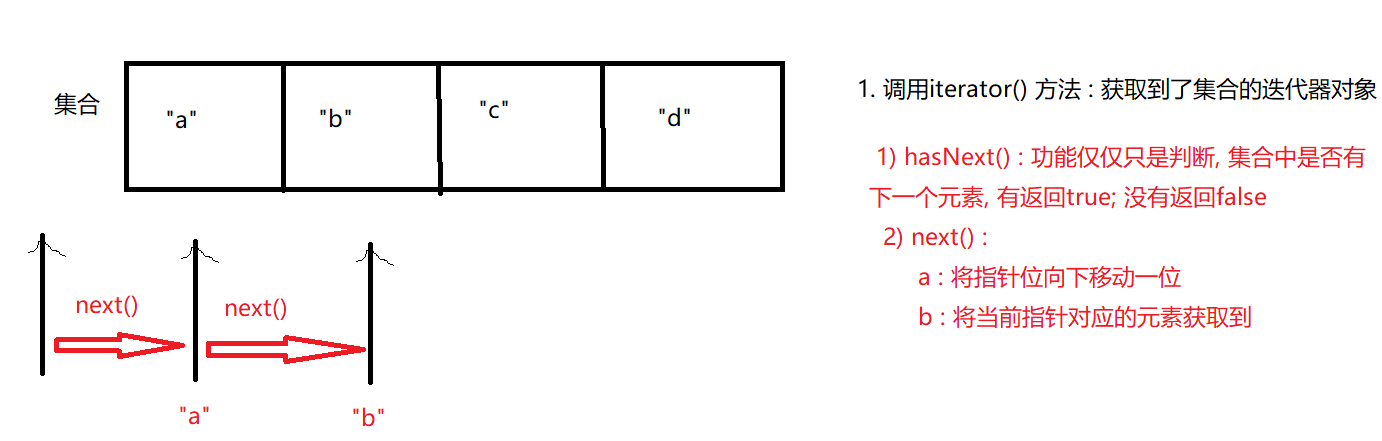
a : hasNext() : 判断集合中是否还具有下一个元素数据, 如果有返回true, 如果没有返回false

b : next() : 获取到集合中的下一个元素, 返回获取到的元素结果, Object类型

c : remove() : 删除目前正在迭代的元素



1. hasNext和next方法之间的配合原理以及迭代器使用的注意点



注意 : 如果集合中所有元素已经迭代完毕, 还继续使用next(), 代码会报错:

NoSuchElementException : 没有这个元素异常

代码

|  |
| --- |
| package com.ujiuye.collection;  import java.util.ArrayList;  import java.util.Collection;  import java.util.Iterator;  public class Demo04\_集合迭代器遍历 {  public static void main(String[] args) {  Collection c = new ArrayList();  c.add("ab");  c.add("123");  c.add("hello");    // 1. 获取到集合c迭代器对象  Iterator it = c.iterator();  // 2. hasNext : 判断集合中是否还具有下一个元素数据, 如果有返回true, 如果没有返回false  while(it.hasNext()) {  // 3. next() : 获取到集合中的下一个元素, 返回获取到元素结果, Object类型  Object obj = it.next();  String s = (String)obj;  System.out.println(s);  }  // NoSuchElementException  System.out.println(it.next());  }  } |

## List

### List集合概述

1. List接口 : 来自于java.util包, 是Collection父接口中的其中一个子接口
2. List集合特征:
3. 存取元素有序(存入集合中的元素顺序与从集合中取出元素的顺序一致)
4. 有索引
5. 可以在集合中存储重复元素
6. List是接口, 不能实例化对象, 需要一个实现类, 找到ArrayList

List list = new ArrayList(); // 接口多态性

### List集合中特有方法

增删改查4个方法:

1. add(int index, Object obj) : 将元素obj添加到集合指定index索引位置上
2. remove(int index) : 删除index索引位置上的元素, 将删除元素的原值返回
3. set(int index, Object obj) : 将集合中指定的index索引位置上的元素替换成obj, 将被替换的原值作为返回值类型
4. get(int index) : 获取到集合中指定index索引位置上的元素, 返回值类型Object

代码

|  |
| --- |
| package com.ujiuye.list;  import java.util.ArrayList;  import java.util.List;  public class Demo01\_List集合特有方法 {  public static void main(String[] args) {  List list = new ArrayList();// list集合中的索引范围0-3  list.add("a");  list.add("b");  list.add("c");  list.add("d");    // 1.add(int index, Object obj) : 将元素obj添加到集合指定index索引位置上  list.add(2,"first");  System.out.println(list);// [a, b, first, c, d]    // 2.remove(int index) : 删除index索引位置上的元素, 将删除元素的原值返回  Object obj = list.remove(4);  System.out.println(obj);// d  System.out.println(list);// [a, b, first, c]    // 3.set(int index, Object obj) : 将集合中指定的index索引位置上的元素替换成obj, 将被替换的元素原值作为返回值类型  Object obj1 = list.set(2, "modify");  System.out.println(obj1);// first  System.out.println(list);// [a, b, modify, c]    // 4. get(int index) : 获取到集合中指定index索引位置上的元素, 返回值类型Object  System.out.println(list.get(1));// b  System.out.println(list.get(3));// c  }  } |

### List集合第三种遍历---索引遍历

1. get(int index): 表示将指定index索引位置元素获取到;

如果可以将一个List集合的每一个索引都获取到, 再结合get(index)方法, 就能获取到集合中的每一个元素

1. List集合元素索引范围: 0----size()-1

代码

|  |
| --- |
| package com.ujiuye.list;  import java.util.ArrayList;  import java.util.List;  public class Demo02\_List集合特有索引遍历方式 {  public static void main(String[] args) {  List list = new ArrayList();  list.add("a");  list.add("b");  list.add("c");  list.add("d");    for(int index = 0; index < list.size(); index++) {  Object obj = list.get(index);  String s = (String)obj;  System.out.println(s);  }  }  } |

### 并发修改异常

1. ConcurrentModificationException

并发 修改 异常

1. 并发修改异常发生原因:

当使用迭代器迭代一个集合时, 一边迭代一边使用集合对象修改(增加,删除)集合中的元素, 就会发生并发修改异常; 因为一边遍历一边修改, 迭代器无法保证迭代的结果是正确的, 就会报出异常

1. 解决并发修改异常方案:
2. 使用List集合特有的迭代器ListIterator遍历:

ListIterator : 接口, 来自于java.util包; 表示列表迭代器，在原有Iterator迭代器功能基础上, 还允许程序员一边迭代遍历一边修改集合列表;

a : 获取ListIterator : 在List集合中, 有方法功能 listIterator(), 方法返回值类型是ListIterator 接口, 方法实际返回的就是接口的一个实现类对象

b: 需要使用迭代器ListIterator 中方法功能向集合中添加或者修改元素, 千万不要使用集合对象本身向集合中添加或删除修改元素

1. List集合特有的索引+get形式, 不会发生并发修改异常

需求: 定义出一个集合, 集合中存储字符串数据, 遍历集合的同时, 验证, 如果集合中存在元素hello, 向集合中添加元素world

代码

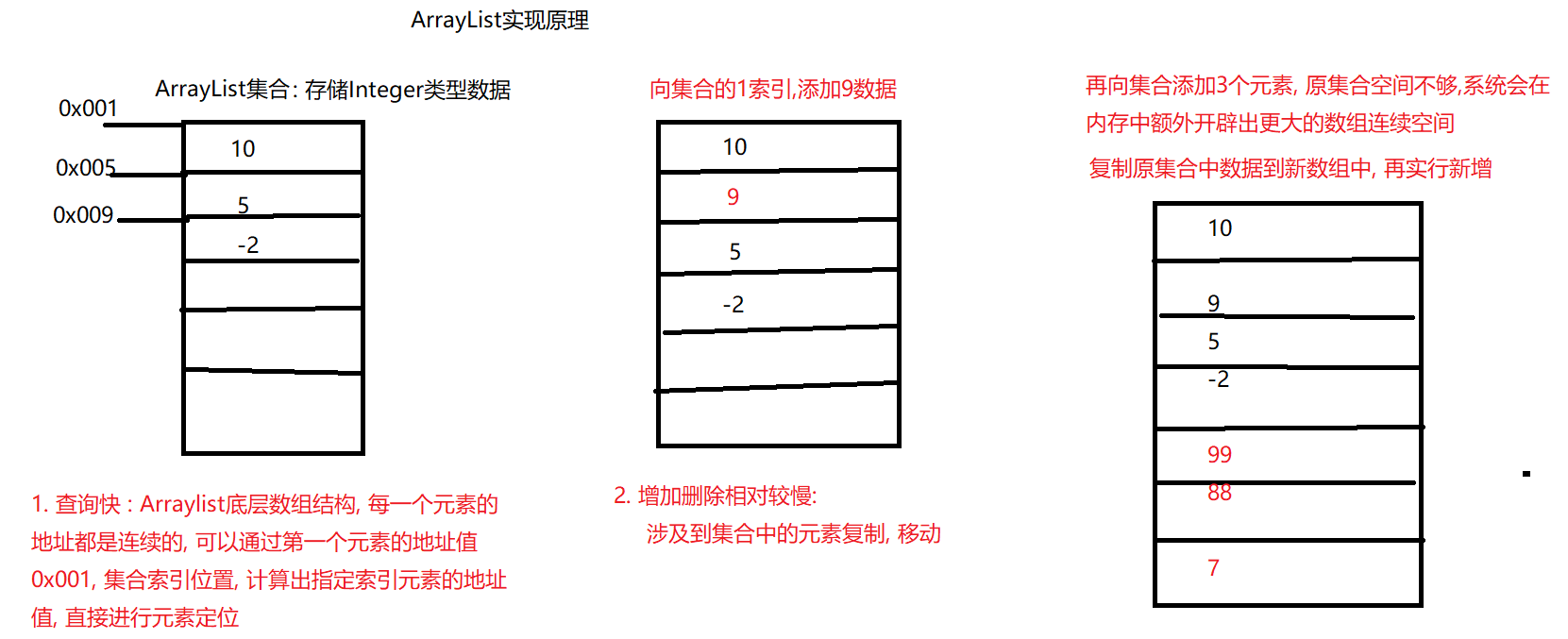
|  |
| --- |
| package com.ujiuye.list;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.List;  import java.util.ListIterator;  public class Demo03\_并发修改异常 {  public static void main(String[] args) {  List list = new ArrayList();  list.add("a");  list.add("hello");  list.add("123abc");  // addEle(list);  addEle3(list);  }    // 2. 解决并发修改异常第二种方案:  public static void addEle3(List list) {  // 索引+get方法遍历集合list  for(int index = 0; index < list.size();index++) {  String s = (String)list.get(index);  if("hello".equals(s)) {  list.add("world");  }  }  System.out.println(list);  }      // 1. 解决并发修改异常第一种方案:  public static void addEle2(List list) {  // 1) list调用特有迭代器 : listIterator()  ListIterator it = list.listIterator();  // 2) 迭代集合list中的元素  while(it.hasNext()) {  String s = (String)it.next();  if("hello".equals(s)) {  it.add("world");  }  }    System.out.println(list);  }    /\*需求: 定义出一个集合, 集合中存储字符串数据, 遍历集合的同时, 验证,  如果集合中存在元素hello, 向集合中添加元素world  发生并发修改异常方法  \*/  public static void addEle(List list) {  // 方法中可以使用list集合  Iterator it = list.iterator();  while(it.hasNext()) {  String s = (String)it.next();  if("hello".equals(s)) {  list.add("world");  }  }  System.out.println(list);  }  } |

### List接口的实现类

List接口 : 主要使用实现类ArrayList 和 LinkedList, 通过实现类底层不同的数据结构, 数据实现, 区分不同类型

#### ArrayList

1. 底层数组实现, 顺序存储, 特点 : 查询快, 增加删除相对较慢



#### LinkedList

1. 字节实现, 链式存储(不需要空间地址连续, 可以利用内存中零散空间), 查询慢, 增删快

节点存储一个元素时, 每一个节点都有两部分 : 数据域, 地址域

