【济南中心】JAVA 编程阶梯：基础篇之第十五章

- **集合框架**

集合的由来

数组长度是固定,当添加的元素超过了数组的长度时需要对数组重新定义,太麻烦,java 内部给我们提供了集合类,能存储任意对象,长度是可以改变的,随着元素的增加而增加,随着元素的减少而减少。

集合类的特点

集合只能存储对象,//jdk1.5 版本后可以存基本数据类型 ,自动装箱为基本数据类型包装类。

集合的长度是可变的

集合可以存储不同类型的对象

数组和集合的区别

区别 1 :

数组既可以存储基本数据类型,又可以存储引用数据类型,基本数据类型存储的是值,引用数据类型存储的是地址值

集合只能存储引用数据类型(对象)集合中也可以存储基本数据类型,但是在存储的时候会自动装箱变成对象

区别 2:

数组长度是固定的,不能自动增长

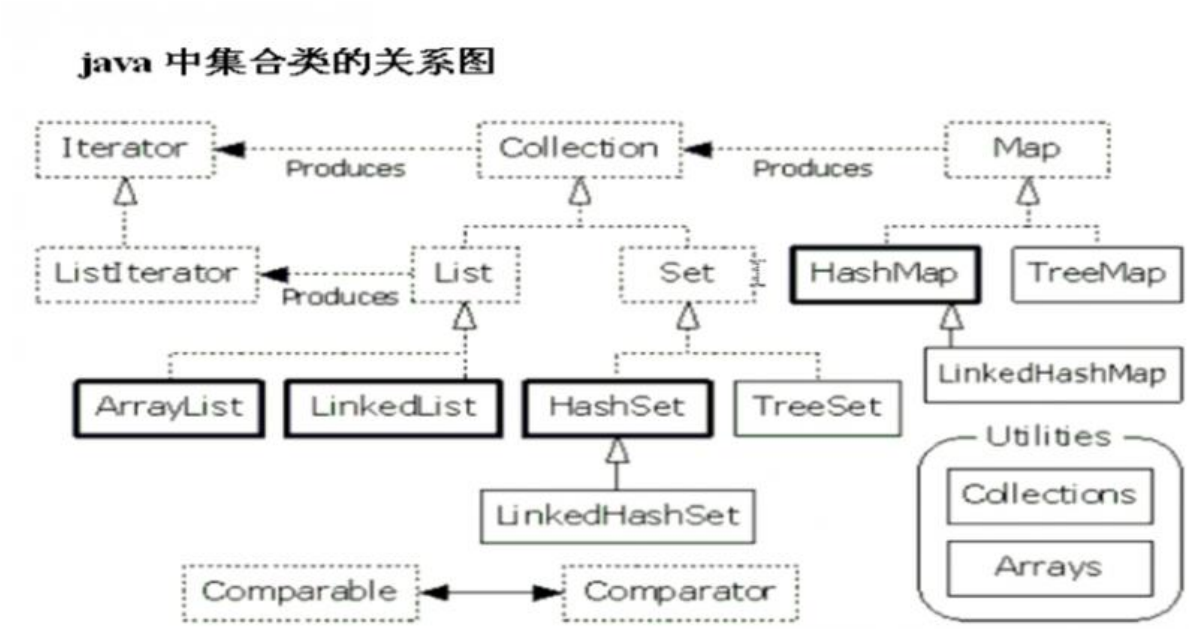
集合的长度的是可变的,可以根据元素的增加而增长

数组和集合什么时候用

1,如果元素个数是固定的推荐用数组

2,如果元素个数不是固定的推荐用集合

集合继承体系图



- **Collection 集合的基本功能**

a)添加：

i. boolean add(E e)　　添加元素 e

j. boolean addAll(Collection c)　将集合 c 中的所有元素添加到当前集合

b)删除：

i. void clear():清空容器

j. boolean remove(Objec object):　　移除一个元素

k. boolean removeAll(Collection c):　移除与 c 所有的相同元素

//retainall(Collection c) 保留与 c 相同的元素

c)判断：

i. boolean contains(Object object):判断是否包含此元素

j. boolean containsAll(Collection c):判断是否包含一堆元素

k. boolean equals(Object object):比较此collection与指定对象是否相等

m. boolean isEmpty():判断是否集合为空

d)获取：

h. Iterator iterator():取出

i. int hashCode():返回此 collection 的哈希值

j. int size():返回此 collection 中元素的个数

k. boolean retainAll(Collection c):取交集

m. Object toArray():返回此 collection 中所有元素的数组

n. T[] toArray(T[] a):返回包含此 collection 中所有元素的数值

集合的遍历之集合转数组遍历

```
package com.itcast;

import java.util.ArrayList;
import java.util.Collection;

/**
 * toArray()遍历集合
 *
 * @author Somnus
 *
 */
public class Demo {

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

```
                Collection<People> coll = new ArrayList<People>();

                coll.add(new People("小明", 23)); // Object obj = new Student("
张三",23);

                coll.add(new People("小红", 24));

                coll.add(new People("小王", 25));

                coll.add(new People("小李", 26));

                Object[] arr = coll.toArray(); // 将集合转换成数组

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

                        People s = (People) arr[i]; // 强转成 Student

                        System.out.println(s.getName() + "," +
s.getAge());
                }
        }
}

class People {
        private String name;
        private int age;

        public People() {
        }

        public People(String name, int age) {
                super();
                this.name = name;
                this.age = age;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
```

```
	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}
}
```

集合的遍历之迭代器遍历(集合的专用遍历方式)

```
package com.itcast;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

/**
 * Iterator 遍历集合
 *
 * @author Somnus
 *
 */
public class Demo {

	public static void main(String[] args) {

		Collection<People> coll = new ArrayList<People>();
		coll.add(new People("小明", 23));
		coll.add(new People("小红", 24));
		coll.add(new People("小王", 25));
		coll.add(new People("小李", 26));
		System.out.println("-----------第一种使用方式
---------------");
		Iterator<People> it = coll.iterator(); // 获取迭代器
		while (it.hasNext()) { // 判断集合中是否有元素
			People s = (People) it.next(); // 向下转型
			System.out.println(s.getName() + "," +
s.getAge()); // 获取对象中的姓名和年龄
		}
		System.out.println("-----------第二种使用方式
---------------");
		for (Iterator<People> ite = coll.iterator(); ite.hasNext();) {
			People s = (People) ite.next(); // 向下转型
```

```
                                        System.out.println(s.getName() + "," +
s.getAge()); // 获取对象中的姓名和年龄
                                }
                }
}

class People {
                private String name;
                private int age;

                public People() {
                }

                public People(String name, int age) {
                                super();
                                this.name = name;
                                this.age = age;
                }

                public String getName() {
                                return name;
                }

                public void setName(String name) {
                                this.name = name;
                }

                public int getAge() {
                                return age;
                }

                public void setAge(int age) {
                                this.age = age;
                }
}
```

迭代器

(1)迭代器就是取出集合元素的方式

(2)迭代器的作用

因为每个集合中元素的取出方式都不一样，于是就把元素的取出方式进行抽

取，并定义在集合内部，

这样取出方式就可以直接访问集合内部的元素；

而每个容器的数据结构不同，所以取出动作的细节也不一样，但是有共性内容：判断和取出。

那么就将共性内容进行抽取，从而形成了接口 Iterater

(3)获取迭代器的方法：

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

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

(3)迭代器方法：

boolean hasNext() 如果仍有元素可以迭代,则返回 true。

E next() 返回迭代的下一个元素。

void remove() 从迭代器指向的 collection 中移除迭代器返回的最后一个元素（可选操作）。

- **List 集合概述**

List 接口是 Collection 接口的一个子接口。

List 接口中的元素有如下特点(对角标的操作都是特有方法，因为有序：

A:元素有序(存储顺序和取出顺序一致)

B:元素可以重复

List 接口中的特有方法

A:add(int index,Object obj):在指定位置加入元素

B:remove(int index):移除指定位置的元素

C:set(int index,Object obj):修改指定位置的元素

D:get(int index):获取指定位置的元素

E:indexOf(Object obj):获取指定元素的位置

F:subList(int start,int end):从一个大的 List 中截取一个小的 List

G:listIterator():返回一个 List 接口特有的迭代器

List 集合通过 size()和 get()方法结合遍历

```
/**[/align] * 通过 size()和 get()方法结合遍历集合
  *
  * @author Somnus
  *
  */
public class Demo {

		public static void main(String[] args) {

			List<People> list = new ArrayList<>();
			list.add(new People("张三", 18));
			list.add(new People("李四", 18));
			list.add(new People("王五", 18));
			list.add(new People("赵六", 18));
			for (int i = 0; i < list.size(); i++) {
				People s = (People) list.get(i);
				System.out.println(s.getName() + "," +
s.getAge());
			}
		}
}

class People {
		private String name;
		private int age;

		public People() {
		}
```

```
	public People(String name, int age) {
		super();
		this.name = name;
		this.age = age;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}
}
```

集合框架并发修改异常产生的原因及解决方案

/**[/align] * 需求：判断集合里面有没有"孙先生"，如果有当找到"孙先生"的时候添加一个"吴女士"，请写代码实现。

```
 *
 * @author Somnus
 */
public class Demo {

	public static void main(String[] args) {

		List<People> list = new ArrayList<People>();
		list.add(new People("赵先生", 19));
		list.add(new People("钱先生", 19));
		list.add(new People("孙先生", 19));
		list.add(new People("李先生", 19));
		list.add(new People("周先生", 19));
		// Iterator<People> it = list.iterator();
		// while (it.hasNext()) {
		// People p = (People) it.next();
		// if (p.getName().equals("孙先生")) {
```

```
                        // list.add(new People("吴女士", 19)); //
                        // 这里会抛出 ConcurrentModificationException 并发修改异常
                        // }
                        // }

                        // 在迭代时，不可以通过集合对象的方法操作集合中的元素，因为会发生 ConcurrentModificationException(当方法检测到对象的并发修改，但不允许这种修改时，抛出此异常)
                        // 解决方案
                        // a:迭代器迭代元素，迭代器修改元素(ListIterator 的特有功能 add)
                        // b:集合遍历元素，集合修改元素
                        ListIterator<People> lit = list.listIterator(); // 如果想在遍历的过程中添加元素,可以用 ListIterator 中的 add 方法
                        while (lit.hasNext()) {
                                People p = (People) lit.next();
                                if (p.getName().equals("孙先生")) {
                                        lit.add(new People("吴女士", 20));
                                }
                        }
                }
}

class People {
        private String name;
        private int age;

        public People() {
        }

        public People(String name, int age) {
                super();
                this.name = name;
                this.age = age;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
```

```
        public int getAge() {
                return age;
        }

        public void setAge(int age) {
                this.age = age;
        }
}
```

ListIterator：列表迭代器

Iterator 方法有限，只能对元素进行判断、取出和删除的操作

ListIterator 可以对元素进行添加和修改动作等。

获取列表迭代器方法：

ListIterator<E> listIterator() 返回此列表元素的列表迭代器（按适当顺序）。

ListIterator<E> listIterator(int index)

返回此列表中的元素的列表迭代器（按适当顺序），从列表中指定位置开始。

列表迭代器方法：

void add(E e) 将指定的元素插入列表（可选操作）。

boolean hasPrevious() 如果以逆向遍历列表，列表迭代器有多个元素，则返回 true。

int nextIndex() 返回对 next 的后续调用所返回元素的索引。

E previous() 返回列表中的前一个元素。

int previousIndex() 返回对 previous 的后续调用所返回元素的索引。 void set(E e) 用指定元素替换 next 或 previous 返回的最后一个元素（可选操作）。

Vector 类概述

Vector(JDK1.0):底层数据结构是数组数据结构.特点是查询和增删速度都很慢。

默认长度是 10，当超过长度时,按 100%延长集合长度。

线程同步。

Vector 功能跟 ArrayList 功能一模一样，已被 ArrayList 替代)

Vector 类特有功能

public void addElement(E obj)

public E elementAt(int index)

public Enumeration elements()

Vector 的迭代

```
/**
  * Vector 迭代
  *
  * @author Somnus
  */
public class Demo {

		public static void main(String[] args) {

			Vector<String> v = new Vector<String>(); // 创建集合对象,List
的子类

			v.addElement("a");
			v.addElement("b");
			v.addElement("c");
			v.addElement("d");
			// Vector 迭代
			Enumeration<String> en = v.elements(); // 获取枚举
			while (en.hasMoreElements()) { // 判断集合中是否有元素
				System.out.println(en.nextElement());// 获取
集合中的元素

			}
		}
}
```

数据结构之数组和链表

数组的特点：

1.查询快修改也快

2.增删慢

链表的特点：

1.查询慢,修改也慢

2.增删快

List 的三个子类

ArrayList:

底层数据结构是数组，查询快，增删慢。

线程不安全，效率高。

Vector:

底层数据结构是数组，查询快，增删慢。

线程安全，效率低。

Vector 相对 ArrayList 查询慢(线程安全的)

Vector 相对 LinkedList 增删慢(数组结构)

LinkedList:

底层数据结构是链表，查询慢，增删快。

线程不安全，效率高。

Vector 和 ArrayList 的区别

Vector 是线程安全的,效率低

ArrayList 是线程不安全的,效率高

共同点:都是数组实现的

ArrayList 和 LinkedList 的区别

ArrayList 底层是数组结果,查询和修改快

LinkedList 底层是链表结构的,增和删比较快,查询和修改比较慢

共同点:都是线程不安全的

List 的三个子类分别都是什么时候用？

查询多用 ArrayList

增删多用 LinkedList

如果都多 ArrayList