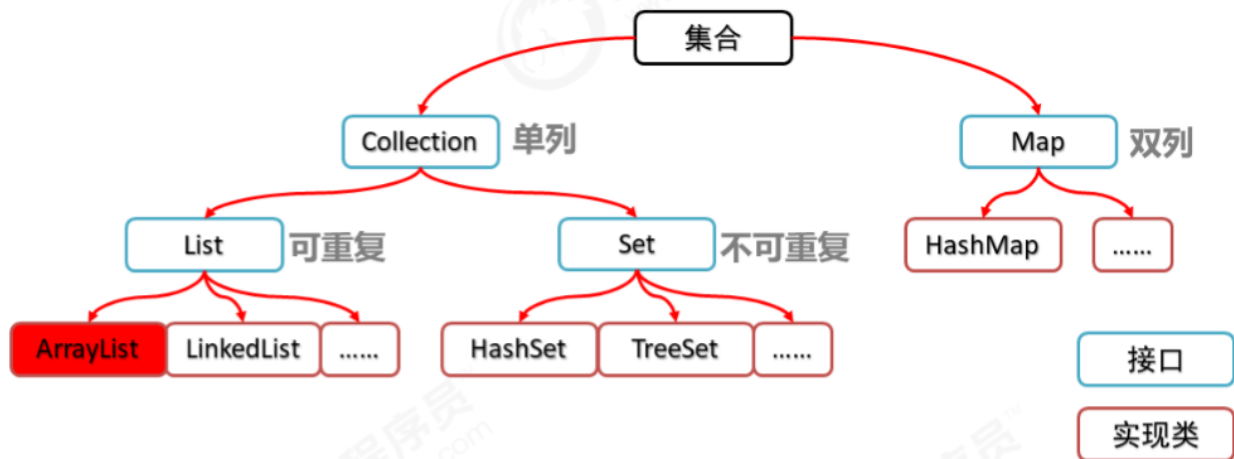


1.Collection集合

1.1集合体系结构【记忆】

- 集合类的特点
提供一种存储空间可变的存储模型，存储的数据容量可以随时发生改变
- 集合类的体系图



1.2Collection集合概述和基本使用【应用】

- Collection集合概述
 - 是单列集合的顶层接口，它表示一组对象，这些对象也称为Collection的元素
 - JDK 不提供此接口的任何直接实现，它提供更具体的子接口（如Set和List）实现
- Collection集合基本使用

```
1 public class CollectionDemo01 {
2     public static void main(String[] args) {
3         //创建Collection集合的对象
4         Collection<String> c = new ArrayList<String>();
5
6         //添加元素: boolean add(E e)
7         c.add("hello");
8         c.add("world");
9         c.add("java");
10
11        //输出集合对象
12        System.out.println(c);
13    }
14 }
```

1.3Collection集合的常用方法【应用】

方法名	说明
boolean add(E e)	添加元素
boolean remove(Object o)	从集合中移除指定的元素
void clear()	清空集合中的元素
boolean contains(Object o)	判断集合中是否存在指定的元素
boolean isEmpty()	判断集合是否为空
int size()	集合的长度，也就是集合中元素的个数

1.4Collection集合的遍历【应用】

- 迭代器的介绍
 - 迭代器，集合的专用遍历方式
 - `Iterator iterator()`：返回此集合中元素的迭代器，通过集合的`iterator()`方法得到
 - 迭代器是通过集合的`iterator()`方法得到的，所以我们说它是依赖于集合而存在的
- Collection集合的遍历

```

1  public class IteratorDemo {
2      public static void main(String[] args) {
3          //创建集合对象
4          Collection<String> c = new ArrayList<>();
5
6          //添加元素
7          c.add("hello");
8          c.add("world");
9          c.add("java");
10         c.add("javaee");
11
12         //Iterator<E> iterator()：返回此集合中元素的迭代器，通过集合的iterator()方法得到
13         Iterator<String> it = c.iterator();
14
15         //用while循环改进元素的判断和获取
16         while (it.hasNext()) {
17             String s = it.next();
18             System.out.println(s);
19         }
20     }
21 }

```

1.5集合使用步骤图解【理解】

- 使用步骤

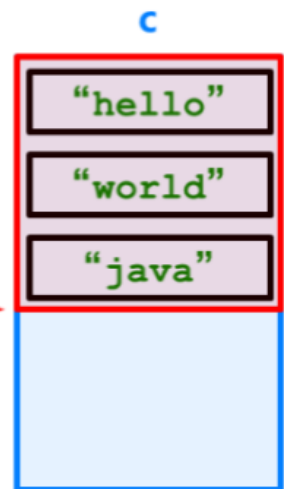
```

public static void main(String[] args) {
    Collection<String> c = new ArrayList<String>();
    c.add("hello");
    c.add("world");
    c.add("java");

    Iterator<String> it = c.iterator();
    while(it.hasNext()){
        String s = it.next();
        System.out.println(s);
    }
}

```

输出: hello
输出: world
输出: java



1.6集合的案例-Collection集合存储学生对象并遍历【应用】

- 案例需求

创建一个存储学生对象的集合，存储3个学生对象，使用程序实现在控制台遍历该集合

- 代码实现

- 学生类

```

1 public class Student {
2     private String name;
3     private int age;
4
5     public Student() {
6     }
7
8     public Student(String name, int age) {
9         this.name = name;
10        this.age = age;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28 }

```

- 测试类

```
1 public class CollectionDemo {
2     public static void main(String[] args) {
3         //创建Collection集合对象
4         Collection<Student> c = new ArrayList<Student>();
5
6         //创建学生对象
7         Student s1 = new Student("林青霞", 30);
8         Student s2 = new Student("张曼玉", 35);
9         Student s3 = new Student("王祖贤", 33);
10
11        //把学生添加到集合
12        c.add(s1);
13        c.add(s2);
14        c.add(s3);
15
16        //遍历集合(迭代器方式)
17        Iterator<Student> it = c.iterator();
18        while (it.hasNext()) {
19            Student s = it.next();
20            System.out.println(s.getName() + "," + s.getAge());
21        }
22    }
23 }
```

2.List集合

2.1List集合概述和特点【记忆】

- List集合概述
 - 有序集合(也称为序列)，用户可以精确控制列表中每个元素的插入位置。用户可以通过整数索引访问元素，并搜索列表中的元素
 - 与Set集合不同，列表通常允许重复的元素
- List集合特点
 - 有索引
 - 可以存储重复元素
 - 元素存取有序

2.2List集合的特有方法【应用】

方法名	描述
void add(int index,E element)	在此集合中的指定位置插入指定的元素
E remove(int index)	删除指定索引处的元素，返回被删除的元素
E set(int index,E element)	修改指定索引处的元素，返回被修改的元素
E get(int index)	返回指定索引处的元素

2.3集合的案例-List集合存储学生对象并遍历【应用】

- 案例需求

创建一个存储学生对象的集合，存储3个学生对象，使用程序实现在控制台遍历该集合

- 代码实现

- 学生类

```
1 public class Student {
2     private String name;
3     private int age;
4
5     public Student() {
6     }
7
8     public Student(String name, int age) {
9         this.name = name;
10        this.age = age;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28 }
```

- 测试类

```
1 public class ListDemo {
2     public static void main(String[] args) {
3         //创建List集合对象
4         List<Student> list = new ArrayList<Student>();
5
6         //创建学生对象
7         Student s1 = new Student("林青霞", 30);
8         Student s2 = new Student("张曼玉", 35);
9         Student s3 = new Student("王祖贤", 33);
10
11        //把学生添加到集合
12        list.add(s1);
13        list.add(s2);
```

```

14         list.add(s3);
15
16         //迭代器方式
17         Iterator<Student> it = list.iterator();
18         while (it.hasNext()) {
19             Student s = it.next();
20             System.out.println(s.getName() + "," + s.getAge());
21         }
22
23         System.out.println("-----");
24
25         //for循环方式
26         for(int i=0; i<list.size(); i++) {
27             Student s = list.get(i);
28             System.out.println(s.getName() + "," + s.getAge());
29         }
30     }
31 }
32

```

2.4并发修改异常【应用】

- 出现的原因

迭代器遍历的过程中，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判断预期修改值和实际修改值不一致，则会出现：ConcurrentModificationException

- 解决的方案

用for循环遍历，然后用集合对象做对应的操作即可

- 示例代码

```

1  public class ListDemo {
2      public static void main(String[] args) {
3          //创建集合对象
4          List<String> list = new ArrayList<String>();
5
6          //添加元素
7          list.add("hello");
8          list.add("world");
9          list.add("java");
10
11         //遍历集合，得到每一个元素，看有没有"world"这个元素，如果有，我就添加一个"javaee"元素，请写代码实现
12         //         Iterator<String> it = list.iterator();
13         //         while (it.hasNext()) {
14         //             String s = it.next();
15         //             if(s.equals("world")) {
16         //                 list.add("javaee");
17         //             }
18         //         }
19
20         for(int i=0; i<list.size(); i++) {
21             String s = list.get(i);

```

```

22         if(s.equals("world")) {
23             list.add("javaee");
24         }
25     }
26
27     //输出集合对象
28     System.out.println(list);
29 }
30 }

```

2.5列表迭代器【应用】

- ListIterator介绍
 - 通过List集合的listIterator()方法得到，所以说它是List集合特有的迭代器
 - 用于允许程序员沿任一方向遍历的列表迭代器，在迭代期间修改列表，并获取列表中迭代器的当前位置
- 示例代码

```

1 public class ListIteratorDemo {
2     public static void main(String[] args) {
3         //创建集合对象
4         List<String> list = new ArrayList<String>();
5
6         //添加元素
7         list.add("hello");
8         list.add("world");
9         list.add("java");
10
11        //获取列表迭代器
12        ListIterator<String> lit = list.listIterator();
13        while (lit.hasNext()) {
14            String s = lit.next();
15            if(s.equals("world")) {
16                lit.add("javaee");
17            }
18        }
19
20        System.out.println(list);
21
22    }
23 }

```

2.6增强for循环【应用】

- 定义格式

```

1 for(元素数据类型 变量名 : 数组/集合对象名) {
2     循环体;
3 }

```

- 示例代码

```

1 public class ForDemo {
2     public static void main(String[] args) {
3         int[] arr = {1,2,3,4,5};
4         for(int i : arr) {
5             System.out.println(i);
6         }
7
8         System.out.println("-----");
9
10        String[] strArray = {"hello","world","java"};
11        for(String s : strArray) {
12            System.out.println(s);
13        }
14
15        System.out.println("-----");
16
17        List<String> list = new ArrayList<String>();
18        list.add("hello");
19        list.add("world");
20        list.add("java");
21
22        for(String s : list) {
23            System.out.println(s);
24        }
25
26        System.out.println("-----");
27
28        //内部原理是一个Iterator迭代器
29        /*
30        for(String s : list) {
31            if(s.equals("world")) {
32                list.add("javaee"); //ConcurrentModificationException
33            }
34        }
35        */
36    }
37 }

```

2.7集合的案例-List集合存储学生对象三种方式遍历【应用】

- 案例需求

创建一个存储学生对象的集合，存储3个学生对象，使用程序实现在控制台遍历该集合

- 代码实现

- 学生类

```

1 public class Student {
2     private String name;
3     private int age;
4
5     public Student() {
6     }

```



```

7
8     public Student(String name, int age) {
9         this.name = name;
10        this.age = age;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28 }

```

○ 测试类

```

1  public class ListDemo {
2      public static void main(String[] args) {
3          //创建List集合对象
4          List<Student> list = new ArrayList<Student>();
5
6          //创建学生对象
7          Student s1 = new Student("林青霞", 30);
8          Student s2 = new Student("张曼玉", 35);
9          Student s3 = new Student("王祖贤", 33);
10
11         //把学生添加到集合
12         list.add(s1);
13         list.add(s2);
14         list.add(s3);
15
16         //迭代器：集合特有的遍历方式
17         Iterator<Student> it = list.iterator();
18         while (it.hasNext()) {
19             Student s = it.next();
20             System.out.println(s.getName()+" "+s.getAge());
21         }
22         System.out.println("-----");
23
24         //普通for：带有索引的遍历方式
25         for(int i=0; i<list.size(); i++) {
26             Student s = list.get(i);
27             System.out.println(s.getName()+" "+s.getAge());
28         }
29     }
30 }

```

```
29         System.out.println("-----");
30
31         //增强for：最方便的遍历方式
32         for(Student s : list) {
33             System.out.println(s.getName()+" "+s.getAge());
34         }
35     }
36 }
```

3.数据结构

3.1数据结构之栈和队列【记忆】

- 栈结构
先进后出
- 队列结构
先进先出

3.2数据结构之数组和链表【记忆】

- 数组结构
查询快、增删慢
- 链表结构
查询慢、增删快

4.List集合的实现类

4.1List集合子类的特点【记忆】

- ArrayList集合
底层是数组结构实现，查询快、增删慢
- LinkedList集合
底层是链表结构实现，查询慢、增删快

4.2集合的案例-ArrayList集合存储学生对象三种方式遍历【应用】

- 案例需求
创建一个存储学生对象的集合，存储3个学生对象，使用程序实现在控制台遍历该集合
- 代码实现
 - 学生类

```
1 public class Student {
2     private String name;
3     private int age;
4 }
```

```

5     public Student() {
6     }
7
8     public Student(String name, int age) {
9         this.name = name;
10        this.age = age;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28 }

```

◦ 测试类

```

1  public class ArrayListDemo {
2      public static void main(String[] args) {
3          //创建ArrayList集合对象
4          ArrayList<Student> array = new ArrayList<Student>();
5
6          //创建学生对象
7          Student s1 = new Student("林青霞", 30);
8          Student s2 = new Student("张曼玉", 35);
9          Student s3 = new Student("王祖贤", 33);
10
11         //把学生添加到集合
12         array.add(s1);
13         array.add(s2);
14         array.add(s3);
15
16         //迭代器：集合特有的遍历方式
17         Iterator<Student> it = array.iterator();
18         while (it.hasNext()) {
19             Student s = it.next();
20             System.out.println(s.getName() + "," + s.getAge());
21         }
22         System.out.println("-----");
23
24         //普通for：带有索引的遍历方式
25         for(int i=0; i<array.size(); i++) {
26             Student s = array.get(i);

```

```

27         System.out.println(s.getName() + "," + s.getAge());
28     }
29     System.out.println("-----");
30
31     //增强for：最方便的遍历方式
32     for(Student s : array) {
33         System.out.println(s.getName() + "," + s.getAge());
34     }
35 }
36 }

```

4.3 LinkedList集合的特有功能【应用】

- 特有方法

方法名	说明
public void addFirst(E e)	在该列表开头插入指定的元素
public void addLast(E e)	将指定的元素追加到此列表的末尾
public E getFirst()	返回此列表中的第一个元素
public E getLast()	返回此列表中的最后一个元素
public E removeFirst()	从此列表中删除并返回第一个元素
public E removeLast()	从此列表中删除并返回最后一个元素