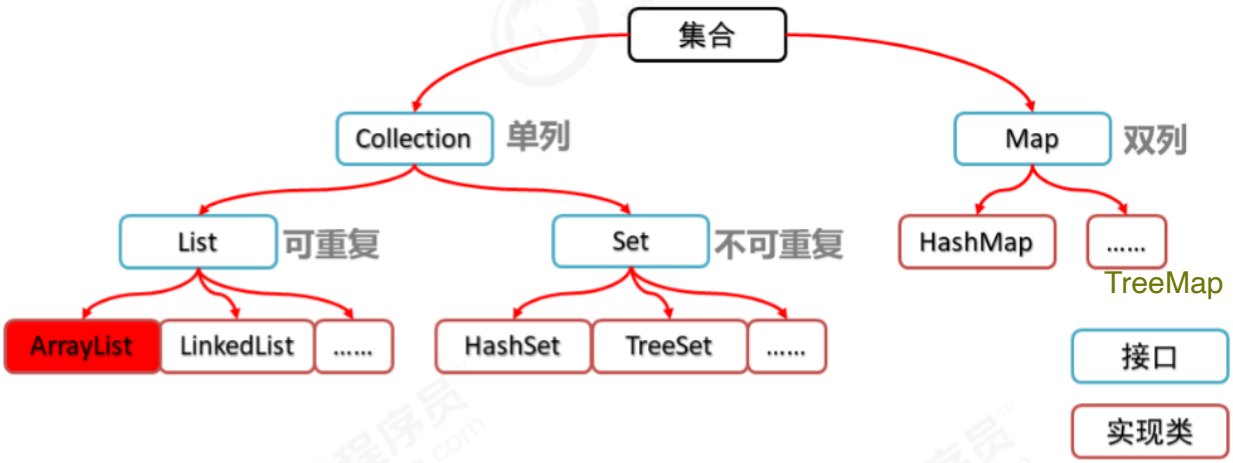
1.Collection集合

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
  - 是单例集合的顶层接口，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

```java
public class CollectionDemo01 {
    public static void main(String[] args) {
        //创建Collection集合的对象
        Collection<String> c = new ArrayList<String>();

        //添加元素：boolean add(E e)
        c.add("hello");
        c.add("world");
        c.add("java");

        //输出集合对象
        System.out.println(c);
    }
}
```

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

```
public class IteratorDemo {
    public static void main(String[] args) {
        //创建集合对象
        Collection<String> c = new ArrayList<>();

        //添加元素
        c.add("hello");
        c.add("world");
        c.add("java");
        c.add("javaee");

        //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
        Iterator<String> it = c.iterator();

        //用while循环改进元素的判断和获取
        while (it.hasNext()) {
            String s = it.next();
            System.out.println(s);
        }
    }
}
```

1.5集合使用步骤图解【理解】

- 使用步骤

```java
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

```java
public class Student {
    private String name;
    private int age;

    public Student() {
    }

    public Student(String name, int age) {
        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;
    }
}
```

- 测试类

```
public class CollectionDemo {
    public static void main(String[] args) {
        //创建Collection集合对象
        Collection<Student> c = new ArrayList<Student>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);

        //把学生添加到集合
        c.add(s1);
        c.add(s2);
        c.add(s3);

        //遍历集合(迭代器方式)
        Iterator<Student> it = c.iterator();
        while (it.hasNext()) {
            Student s = it.next();
            System.out.println(s.getName() + "," + s.getAge());
        }
    }
}
```

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

```
public class Student {
    private String name;
    private int age;

    public Student() {
    }

    public Student(String name, int age) {
        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;
    }
}
```

  - 测试类

```
public class ListDemo {
    public static void main(String[] args) {
        //创建List集合对象
        List<Student> list = new ArrayList<Student>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);

        //把学生添加到集合
        list.add(s1);
        list.add(s2);
```

```
        list.add(s3);

        //迭代器方式
        Iterator<Student> it = list.iterator();
        while (it.hasNext()) {
            Student s = it.next();
            System.out.println(s.getName() + "," + s.getAge());
        }

        System.out.println("--------");

        //for循环方式
        for(int i=0; i<list.size(); i++) {
            Student s = list.get(i);
            System.out.println(s.getName() + "," + s.getAge());
        }

    }
}
```

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

```
public class ListDemo {
    public static void main(String[] args) {
        //创建集合对象
        List<String> list = new ArrayList<String>();

        //添加元素
        list.add("hello");
        list.add("world");
        list.add("java");

        //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
//        Iterator<String> it = list.iterator();
//        while (it.hasNext()) {
//            String s = it.next();
//            if(s.equals("world")) {
//                list.add("javaee");
//            }
//        }

        for(int i=0; i<list.size(); i++) {
            String s = list.get(i);
```

```
            if(s.equals("world")) {
                list.add("javaee");
            }
        }

        //输出集合对象
        System.out.println(list);
    }
}
```

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

```
public class ListIteratorDemo {
    public static void main(String[] args) {
        //创建集合对象
        List<String> list = new ArrayList<String>();

        //添加元素
        list.add("hello");
        list.add("world");
        list.add("java");

        //获取列表迭代器
        ListIterator<String> lit = list.listIterator();
        while (lit.hasNext()) {
            String s = lit.next();
            if(s.equals("world")) {
                lit.add("javaee");
            }
        }

        System.out.println(list);

    }
}
```

2.6增强for循环【应用】

1.实现Iterable接口的类允许其对象成为增强型for语句的目标
2.其内部原理是一个Iterator迭代器

- 定义格式

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

- 示例代码

```java
public class ForDemo {
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5};
        for(int i : arr) {
            System.out.println(i);
        }

        System.out.println("--------");

        String[] strArray = {"hello","world","java"};
        for(String s : strArray) {
            System.out.println(s);
        }

        System.out.println("--------");

        List<String> list = new ArrayList<String>();
        list.add("hello");
        list.add("world");
        list.add("java");

        for(String s : list) {
            System.out.println(s);
        }

        System.out.println("--------");

        //内部原理是一个Iterator迭代器
        /*
        for(String s : list) {
            if(s.equals("world")) {
                list.add("javaee"); //ConcurrentModificationException
            }
        }
        */
    }
}
```

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

```java
public class Student {
    private String name;
    private int age;

    public Student() {
    }
```

```
    public Student(String name, int age) {
        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;
    }
}
```

- 测试类

```
public class ListDemo {
    public static void main(String[] args) {
        //创建List集合对象
        List<Student> list = new ArrayList<Student>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);

        //把学生添加到集合
        list.add(s1);
        list.add(s2);
        list.add(s3);

        //迭代器：集合特有的遍历方式
        Iterator<Student> it = list.iterator();
        while (it.hasNext()) {
            Student s = it.next();
            System.out.println(s.getName()+","+s.getAge());
        }
        System.out.println("--------");

        //普通for：带有索引的遍历方式
        for(int i=0; i<list.size(); i++) {
            Student s = list.get(i);
            System.out.println(s.getName()+","+s.getAge());
        }
```

```
        System.out.println("--------");

        //增强for：最方便的遍历方式
        for(Student s : list) {
            System.out.println(s.getName()+","+s.getAge());
        }
    }
}
```

3.数据结构

1.计算机存储，组织数据的方式。相互之间存在一种或者多种特定关系的 数据元素的集合

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

- 队列结构

  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

```
public class Student {
    private String name;
    private int age;

```

```
    public Student() {
    }

    public Student(String name, int age) {
        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;
    }
}
```

- 测试类

```
public class ArrayListDemo {
    public static void main(String[] args) {
        //创建ArrayList集合对象
        ArrayList<Student> array = new ArrayList<Student>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);

        //把学生添加到集合
        array.add(s1);
        array.add(s2);
        array.add(s3);

        //迭代器：集合特有的遍历方式
        Iterator<Student> it = array.iterator();
        while (it.hasNext()) {
            Student s = it.next();
            System.out.println(s.getName() + "," + s.getAge());
        }
        System.out.println("--------");

        //普通for：带有索引的遍历方式
        for(int i=0; i<array.size(); i++) {
            Student s = array.get(i);
```

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

        //增强for：最方便的遍历方式
        for(Student s : array) {
            System.out.println(s.getName() + "," + s.getAge());
        }
    }
}
```

4.3LinkedList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