1.Collection集合

1.1数组和集合的区别【理解】

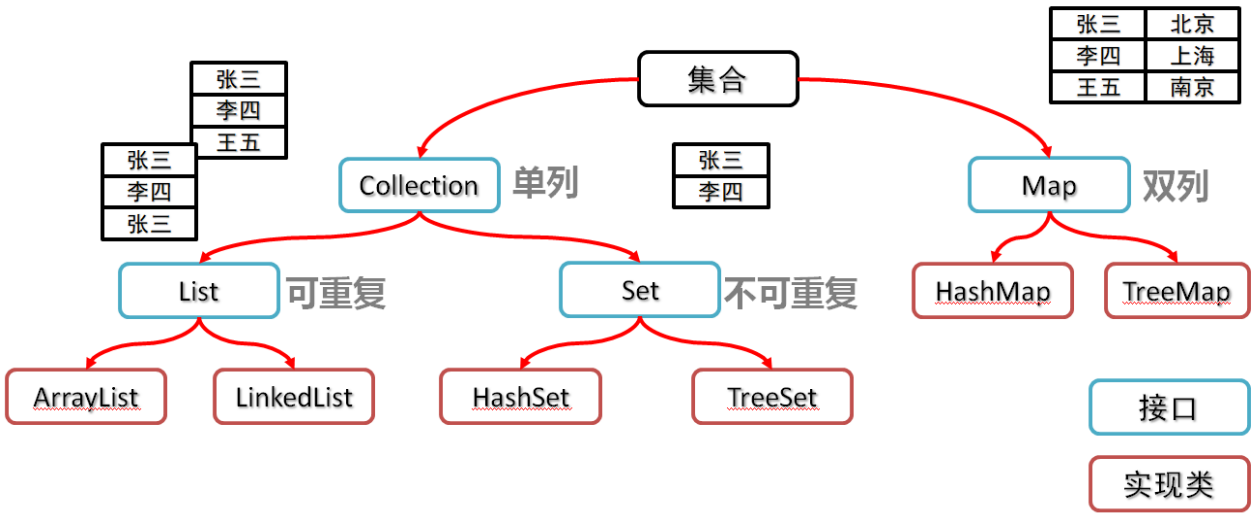
- 相同点

  都是容器,可以存储多个数据

- 不同点
  - 数组的长度是不可变的,集合的长度是可变的
  - 数组可以存基本数据类型和引用数据类型

    集合只能存引用数据类型,如果要存基本数据类型,需要存对应的包装类

1.2集合类体系结构【理解】



1.3Collection 集合概述和使用【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
- 创建Collection集合的对象
  - 多态的方式
  - 具体的实现类ArrayList
- Collection集合常用方法

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
boolean removeIf(Object o)	根据条件进行移除
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,通过集合对象的iterator()方法得到
- Iterator中的常用方法

  boolean hasNext(): 判断当前位置是否有元素可以被取出 E next(): 获取当前位置的元素,将迭代器对象移向下一个索引位置
- Collection集合的遍历

```
public class IteratorDemo1 {
    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);
        }
    }
}
```

- 迭代器中删除的方法

void remove(): 删除迭代器对象当前指向的元素

```
public class IteratorDemo2 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("a");
        list.add("b");
        list.add("b");
        list.add("c");
        list.add("d");

        Iterator<String> it = list.iterator();
        while(it.hasNext()){
            String s = it.next();
            if("b".equals(s)){
                //指向谁,那么此时就删除谁.
                it.remove();
            }
        }
        System.out.println(list);
    }
}
```

1.5增强for循环【应用】

- 介绍
  - 它是JDK5之后出现的,其内部原理是一个Iterator迭代器
  - 实现Iterable接口的类才可以使用迭代器和增强for
  - 简化数组和Collection集合的遍历
- 格式

  for(集合/数组中元素的数据类型 变量名 : 集合/数组名) {

  // 已经将当前遍历到的元素封装到变量中了,直接使用变量即可

  }
- 代码

```
public class MyCollectonDemo1 {
    public static void main(String[] args) {
        ArrayList<String> list =  new ArrayList<>();
        list.add("a");
        list.add("b");
        list.add("c");
        list.add("d");
        list.add("e");
        list.add("f");

        //1,数据类型一定是集合或者数组中元素的类型
        //2,str仅仅是一个变量名而已,在循环的过程中,依次表示集合或者数组中的每一个元素
        //3,list就是要遍历的集合或者数组

        for(String str : list){
```

```
            System.out.println(str);
        }
    }
}
```

2.List集合

2.1List集合的概述和特点【记忆】

- List集合的概述
  - 有序集合,这里的有序指的是存取顺序
  - 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集合的特点
  - 存取有序
  - 可以重复
  - 有索引

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

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
- 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

4.2LinkedList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

5.泛型

5.1泛型概述【理解】

- 泛型的介绍

  泛型是JDK5中引入的特性，它提供了编译时类型安全检测机制

- 泛型的好处

  1. 把运行时期的问题提前到了编译期间
  2. 避免了强制类型转换

- 泛型的定义格式
  - <类型>: 指定一种类型的格式.尖括号里面可以任意书写,一般只写一个字母.例如:
  - <类型1,类型2...>: 指定多种类型的格式,多种类型之间用逗号隔开.例如:

5.2泛型类【应用】

- 定义格式

```
修饰符 class 类名<类型> {  }
```

- 示例代码

- 泛型类

```
public class Generic<T> {
    private T t;

    public T getT() {
        return t;
    }

    public void setT(T t) {
        this.t = t;
    }
}
```

- 测试类

```
public class GenericDemo1 {
    public static void main(String[] args) {
        Generic<String> g1 = new Generic<String>();
        g1.setT("杨幂");
        System.out.println(g1.getT());

        Generic<Integer> g2 = new Generic<Integer>();
        g2.setT(30);
        System.out.println(g2.getT());

        Generic<Boolean> g3 = new Generic<Boolean>();
        g3.setT(true);
        System.out.println(g3.getT());
    }
}
```

5.3泛型方法【应用】

- 定义格式

```
修饰符 <类型> 返回值类型 方法名(类型 变量名) {  }
```

- 示例代码
  - 带有泛型方法的类

```
public class Generic {
    public <T> void show(T t) {
        System.out.println(t);
    }
}
```

  - 测试类

```
public class GenericDemo2 {
    public static void main(String[] args) {
        Generic g = new Generic();
        g.show("柳岩");
        g.show(30);
        g.show(true);
        g.show(12.34);
    }
}
```

5.4泛型接口【应用】

- 定义格式

```
修饰符 interface 接口名<类型> {  }
```

- 示例代码
  - 泛型接口

```
public interface Generic<T> {
    void show(T t);
}
```

  - 泛型接口实现类1

    定义实现类时,定义和接口相同泛型,创建实现类对象时明确泛型的具体类型

```
public class GenericImpl1<T> implements Generic<T> {
    @Override
    public void show(T t) {
        System.out.println(t);
    }
}
```

  - 泛型接口实现类2

    定义实现类时,直接明确泛型的具体类型

```
public class GenericImpl2 implements Generic<Integer>{
     @Override
     public void show(Integer t) {
          System.out.println(t);
     }
}
```

  - 测试类

```
public class GenericDemo3 {
    public static void main(String[] args) {
        GenericImpl1<String> g1 = new GenericImpl<String>();
        g1.show("林青霞");
        GenericImpl1<Integer> g2 = new GenericImpl<Integer>();
        g2.show(30);

        GenericImpl2 g3 = new GenericImpl2();
        g3.show(10);
    }
}
```

5.5类型通配符

- 类型通配符: <?>
  - ArrayList<?>: 表示元素类型未知的ArrayList,它的元素可以匹配任何的类型
  - 但是并不能把元素添加到ArrayList中了,获取出来的也是父类类型
- 类型通配符上限: <? extends 类型>
  - ArrayListList <? extends Number>: 它表示的类型是Number或者其子类型
- 类型通配符下限: <? super 类型>
  - ArrayListList <? super Number>: 它表示的类型是Number或者其父类型
- 泛型通配符的使用

```
public class GenericDemo4 {
    public static void main(String[] args) {
        ArrayList<Integer> list1 = new ArrayList<>();
        ArrayList<String> list2 = new ArrayList<>();
        ArrayList<Number> list3 = new ArrayList<>();
        ArrayList<Object> list4 = new ArrayList<>();

        method(list1);
        method(list2);
        method(list3);
        method(list4);

        getElement1(list1);
        getElement1(list2);//报错
        getElement1(list3);
        getElement1(list4);//报错

        getElement2(list1);//报错
        getElement2(list2);//报错
        getElement2(list3);
        getElement2(list4);
    }

    // 泛型通配符: 此时的泛型?,可以是任意类型

    public static void method(ArrayList<?> list){}
```

```
    // 泛型的上限：此时的泛型?,必须是Number类型或者Number类型的子类
    public static void getElement1(ArrayList<? extends Number> list){}
    // 泛型的下限：此时的泛型?,必须是Number类型或者Number类型的父类
    public static void getElement2(ArrayList<? super Number> list){}

}
```