集合框架体系--（上）

学习内容

- Collection集合
- Iterator迭代器
- 增强for循环
- 泛型
- list集合
- 初识数据结构

1.1 集合

1.1.1 集合介绍

前面的学习，我们知道数据多了，使用数组存放。而且数组中存放的都是基本类型的数据，并且数组是定长的。当在程序中创建的对象比较多的时候，需要对这些对象进行统一的管理和操作，那么首先我们就需要把这些对象存储起来。使用数组是可以存放对象的，我们可以定义对象数组来存放，但是数组这个容器存放对象，要对其中的对象进行更复杂操作时，数据就显的很麻烦。那怎么办呢?

Java中给我们提供了另外一类容器，专门用来存放对象，这个容器就是我们要学习的集合。

集合和数组既然都是容器，它们有啥区别呢?

- 数组的长度是固定的。集合的长度是可变的。
- 数组中存储的是同一类型的元素，可以存储基本数据类型值。集合存储的都是对象。而且对象的类型可以不一致。

集合貌似看起来比较强大，它啥时用呢?

当对象多的时候，先进行存储。

1.1.2 集合框架的由来

集合本身是一个工具，它存放在java.util包中。

JDK最早的1.0版本中。提供的集合容器很少。升级到1.2版，为了更多的需求，出现了集合框架。有了更多的容器。可以完成不同的需求。

这些容器怎么区分？**区分的方式**：每一个容器的数据结构(数据存储到的一种方式)不一样。

不同的容器进行不断的向上抽取，最后形成了一个集合框架，这个框架就是Collection接口。在Collection接口定义着集合框架中最最共性的内容。

在学习时：我们需要看最顶层怎么用， 创建底层对象即可。因为底层继承了父类中的所有功能。

1.1.3 Collection接口的描述

既然Collection接口是集合中的顶层接口，那么它中定义的所有功能子类都可以使用。查阅API中描述的Collection接口。Collection 层次结构 中的根接口。Collection 表示一组对象，这些对象也称为 collection 的元素。一些 collection 允许有重复的元素，而另一些则不允许。一些 collection 是有序的，而另一些则是无序的。

其实我们在使用ArrayList类时，该类已经把所有抽象方法进行了重写。那么，实现Collection接口的所有子类都会进行方法重写。

- Collecton接口常用的子接口有：List接口、Set接口
- List接口常用的子类有：ArrayList类、LinkedList类
- Set接口常用的子类有：HashSet类、LinkedHashSet类

1.1.4数组和集合的区别【理解】

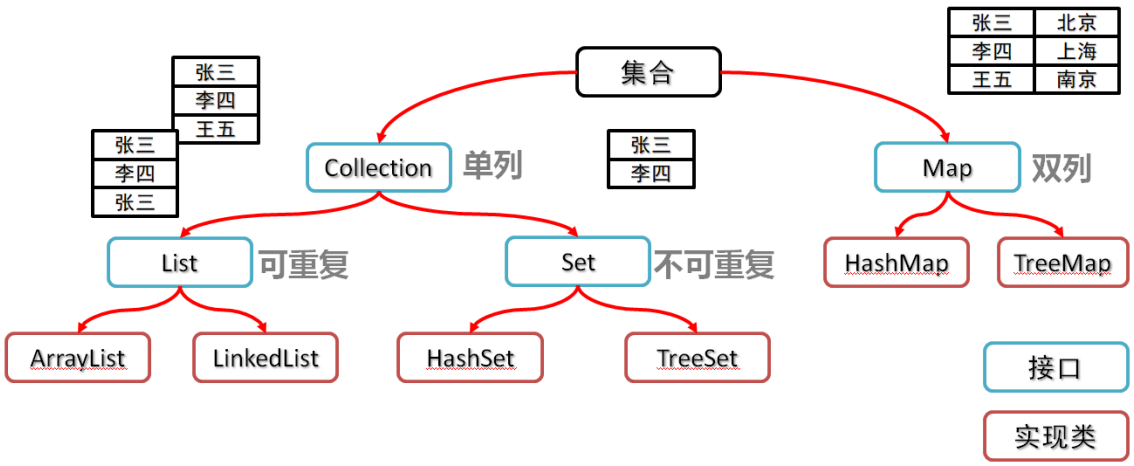
- 相同点

  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

1.1.5集合类体系结构【理解】



1.2Collection 集合应用

- 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

1.2.1Collection集合的遍历【应用】

- 迭代器介绍
  - Collection集合元素的通用获取方式：在取元素之前先要判断集合中有没有元素，如果有，就把这个元素取出来，继续在判断，如果还有就再取出出来。一直把集合中的所有元素全部取出。这种取出方式专业术语称为迭代。
  - 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 IteratorDemo01 {
    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);
        }
    }
}
```

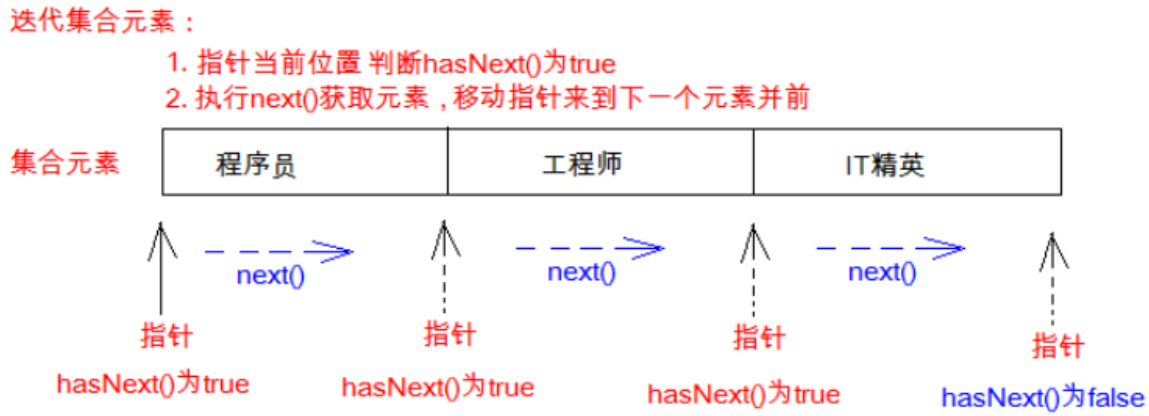
- 迭代器中删除的方法

  void remove(): 删除迭代器对象当前指向的元素

```
public class IteratorDemo02 {
    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);
    }
}
```

1.2.2迭代集合元素代码图解



1.3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 ForDemo01 {
    public static void main(String[] args) {
        ArrayList<String> list =  new ArrayList<>();
        list.add("I");
        list.add("T");
        list.add("枫");
```

```
            list.add("斗");
            list.add("者");

        //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);
            }
    }
}
```

1.3.1增强for循环和老式的for循环有什么区别?

注意：新for循环必须有被遍历的目标。目标只能是Collection或者是数组。

建议：遍历数组时，如果仅为遍历，可以使用增强for如果要对数组的元素进行 操作，使用老式for循环可以通过角标操作。

1.3.2 如何使用idea查看反编译文件?

file—>Project Structure->project->Project compiler output

找到编译后的.class文件

将这个文件直接拖到idea中即可查看，或者 查看反编译文件：打开项目的文件夹，右键show in Explorer,然后直接将反编译文件复制粘贴到该文件夹，然后再idea中双击查看即可。

增强for循环 反编译源码

```
private static void method2()
{
        ArrayList list = new ArrayList();
        list.add("a");
        list.add("b");
        list.add("c");
        String s;
        for (Iterator iterator = list.iterator(); iterator.hasNext(); System.out.println(s))
                s = (String)iterator.next();

}
```

将迭代器放在for循环中的好处：

不会重复定义变量，不产生额外垃圾

2泛型(遇到问题引入)

2.0泛型的引入

需求：打印集合中所有字符串的长度。

```java
//需求：打印集合中所有字符串的长度；
public static void main(String[] args) {
    //1、创建一个List集合
    List list = new ArrayList();
    //2、添加数据
    list.add("哈哈哈");
    list.add("呵呵");
    list.add("嘿嘿嘿嘿");
    list.add(1234);//添加的int型数据会自动装箱为Integer对象
    //3、调用函数，输出所有字符串的长度
    showLength(list);
}
public static void showLength(List list){
    //1、遍历参数集合，得到集合中所有数据
    for (Object object : list) {
        //2、将参数强制类型转换为String类型
        String str = (String)object;//遍历时最后一个数据时一个Integer类型的对象
        System.out.println(str.length());
    }
}
```

```
l "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String
st.generic.Demo1.showLength(Demo1.java:23)
st.generic.Demo1.main(Demo1.java:17)
```

问题：

在操作集合容器时，因为集合中可以保存不同类型的数据，所以很容易因为数据类型而发生问题；

解决思路：

1、在使用集合中的数据之前都先判断，然后再进行强制类型转换；但是这种做法不符合面向对象的一个原理：谁的事情该谁去做；

2、要求使用集合容器时一个容器对象只保存一种类型的数据；

结论：

- 虽然集合可以保存不同类型的数据，但是真正使用时，为了安全，还是应该在一个集合对象中只保存一种类型的数据；
- 问题是，要保证集合中只保存相同类型的数据，靠人工是不现实的，因为人工不可靠；
- 应该让程序自己限制保存的数据类型；编译的时候就能发现错误；**要实现这种要求，可以使用泛型技术实现**；

2.1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: <E,T> <K,V>

2.2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 GenericDemo01 {
    public static void main(String[] args) {
        Generic<String> g1 = new Generic<String>();
        g1.setT("唐三");
        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());
    }
}
```

2.3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) {  }
```

需求：定义一个工具类，可以保存和获取一个任意类型的对象；

原来的实现方法：

```
public class Demo3 {
    //需求：定义一个工具类，可以保存和获取一个任意类型的对象；
    public static void main(String[] args) {
        Tool t = new Tool();
        t.add("adfasdf");
        //因为再工具类中使用Object类型的变量保存数据，所以对象添加进去后都会失去原有的类
型信息，
```

```
        //被自动向上转型为Object类型，使用时需要强制向下转型
        Object object = t.get();
        String str = (String)object;
    }
}
//创建一个工具类
class Tool{
    //因为调用添加方法结束后，添加的对象不能消失了，还能够通过获取方法获取到，所以必须定义一个成员变量保存
    private Object data;
    //添加功能,因为可以添加任意类型的对象，所以参数的类型是Object
    public void add(Object obj){
        this.data = obj;
    }
    //获取功能
    public Object get(){
        //因为添加的对象保存在成员变量data中，所以这里返回data
        return data;
    }
}
```

使用泛型实现：

```
public class Demo4 {
    //需求：定义一个工具类，可以保存和获取一个任意类型的对象；
    public static void main(String[] args) {
        //类上的泛型都是再创建类的对象时赋值的
        MyTool<String> mt = new MyTool<>();
        mt.add("哈哈哈");
        String string = mt.get();
        System.out.println(string.length());
    }
}
//创建一个工具类
class MyTool<X/*这里表示定义了一个类型的变量*/>{
    //因为调用添加方法结束后，添加的对象不能消失了，还能够通过获取方法获取到，所以必须定义一个成员变量保存
    private X data;
    //添加功能,因为可以添加任意类型的对象，所以参数的类型是Object
    public void add(X obj){
        this.data = obj;
    }
    //获取功能
    public X get(){
        //因为添加的对象保存在成员变量data中，所以这里返回data
        return data;
    }
}
```

上面是给这个变量赋值；一旦给变量X赋值为String，则类中所有使用到类型变量X的地方，都会按照String使用；
如果不赋值，默认就是Object

问题：

类上的泛型是一个变量，那么这个变量是什么时候赋值的呢?

创建类的对象时赋值的;

因为类上的泛型是创建对象时才赋值，所以相当于类中一个非静态成员变量。

2.4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 GenericImpl01<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 GenericImpl02 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) {
        GenericImpl01<String> g1 = new GenericImpl01<String>();
        g1.show("柳岩");
        GenericImpl01<Integer> g2 = new GenericImpl01<Integer>();
        g2.show(30);

        GenericImpl02 g3 = new GenericImpl02();
        g3.show(10);
    }
}
```

5.5类型通配符

当使用泛型类或者接口时，传递的数据中，泛型类型不确定，可以通过通配符<?>表示。但是一旦使用泛型的通配符后，只能使用Object类中的共性方法，集合中元素自身方法无法使用。

定义：(查看ArrayList的构造方法)无法在类中使用

- 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 static void demo01() {
        // 左右的泛型必须统一
        // 右边的泛型能否为左边泛型的子类？不行,泛型不存在什么继承（如：装狗的笼子，装军犬的笼子）
        // 右边的泛型在jdk1.7之后可以不写，默认和左边一致
        // 集合里面可以放置的元素为你声明的泛型及其子类
        ArrayList<Object> list = new ArrayList<>();
        list.add(2);
        list.add("123");
    }
```

```
public class Test {
    public static void main(String[] args) {
        // 通配符
        ArrayList<String> list = new ArrayList<String>();
        test(list);

        ArrayList<Person<String>> list3 = new ArrayList<Father<String>>();
        test(list3); //这样也可以，但意义不大

        // extends  类型最高为extends后面的类型
        ArrayList<? extends Object> list4 = new ArrayList<String>();
        ArrayList<? extends String> list44 = new ArrayList<String>();
        list4.add(null); 这里是不报错是因为所有的引用类型默认都是null
        list4.add("aaa");
                         这里报错是因为添加的是extends后面的子类类型，而子类类型又不确定
                         所有不能添加。。
        // super   类型最低为super后面的类型
        ArrayList<? super String> list5 = new ArrayList<String>();
        ArrayList<? super String> list55 = new ArrayList<Object>();
        list5.add("123"); 这里可以添加因为?最低是super后面的类型，我添加String可以的
    }

    private static void test(ArrayList<?> list) { 这里能用ArrayList<Object> list 代替吗？
        System.out.println(list);                  不能，如果可以犯了右边泛型是左边泛型子类
        }                                          的错
```

3.List集合（有索引，有序，重复）

3.1List集合的概述和特点【理解】

- 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

3.2List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
boolean addAll(int index,Collection<? extends E> c)	将集合中所有元素都插入到列表中指定位置

3.3代码实现

一、添加方法

分析和步骤:

1）定义一个ListDemo类，在这个类中定义一个method_1()函数；

2）在method_1()函数中使用new关键字创建ArrayList类的对象list，并赋值给接口List类型；

3）使用集合对象list调用属于Collection接口中的add(E e)函数向集合List中添加字符串”aaaa”；

4）使用集合对象list调用属于接口List中特有的函数add(int index,Object element)根据指定的下标向集合中添加数据”abc”；

5）使用输出语句输出list对象中的值；

```
public static void method_1() {
        //创建集合对象
        List<String> list=new ArrayList<String>();
        List<String> list1=new ArrayList<String>();
        //向集合中添加数据
        list.add("aaaa");
        list.add("bbbb");
        list.add("cccc");
        list1.add("hhhh");
        list1.add("哈哈");
        //使用List接口中特有的函数向接口中添加数据
        //list.add(1, "dddd");//1表示要添加数据的下标位置(原来b的位置，b之后的元素索引加一)
//      list.add(4,"xyz");//指定的下标前面一定要有元素(否则报下标越界异常)
        //向集合list的下标为2的位置添加集合list1中所有的数据
        list.addAll(2, list1);
        //输出数据
        System.out.println(list);
    }
```

注意:

1）指定的下标前面一定要有数据；

2）添加不是覆盖，指定的位置添加了元素后，之前存在元素就会向后移动；

二、获取方法

分析和步骤：

1）在上述ListDemo类中定义一个method_2()函数；

2）在method_2()函数中使用new关键字创建ArrayList类的对象list，并赋值给接口List类型；

3）使用集合对象list调用属于Collection接口中的add(E e)函数向集合List中添加几个字符串数据；

4） 使用集合对象list调用属于接口List中特有的函数get(int index)根据指定的下标获取下标对应的数据；

5）使用输出语句输出获得的数据obj；

```
public static void method_2() {
        //创建集合对象
        List<String> list=new ArrayList<String>();
        //向集合中添加数据
        list.add("nba");
        list.add("nba");
        list.add("cba");
        list.add("wbna");
        list.add("wcba");
        //根据下标获得对应的元素
        //Object obj = list.get(2);
        //Object obj = list.get(7);//下标不能是空白区域
//      System.out.println(obj);
//      List subList = list.subList(1,0);//结束角标要大于等于起始角标
//      System.out.println(subList);
        //返回指定元素第一次出现的角标
        int index = list.indexOf("nbaa");//没有返回-1
        System.out.println(index);
}
```

注意： 下标的范围：index >= 0 && index <= size();

问题升级：由于List接口拥有下标，因此可以像遍历数组的方式遍历List集合：

说明：集合的大小是集合对象名list.size(),下标从0开始所以下标应该小于list.size()，没有等于。

```
//普通for循环遍历List集合
public static void method_3() {
    // 创建一个集合容器
    List list = new ArrayList();

    list.add("aaaa");
    list.add("anc");
    list.add("xyz");
    list.add("cba");
    list.add("aaaa");

    for( int i=0 ; i<list.size() ; i++ ){
        System.out.println(list.get(i));
    }
}
```

三、删除方法

分析和步骤:

1）在上述ListDemo类中定义一个method_3()函数;

2）在method_3()函数中使用new关键字创建ArrayList类的对象list，并赋值给接口List类型;

3）使用集合对象list调用属于Collection接口中的add(E e)函数向集合List中添加几个字符串数据，同时添加几个整数数据，100和32

例如：list.add(100);//在这里被自动装箱了，变成Integer类型了

List.add(32);

4）使用集合对象list调用属于接口Collection接口中的remove()函数，删除指定的字符串，然后输出集合;

5）用集合对象list调用属于接口List中特有的函数remove(int index)，如list.remove(100),这里会报错，这里不会认为100是集合中的数据，会被认为这是下标100，报越界异常，如果我们想让他变成集合中对象类型数据100，可以如下做法：list.remove(Integer.valueOf(100)),把100变成包装类对象;

```
public static void method_3() {
        //创建集合对象
        List list=new ArrayList();
        //向集合中添加数据
        list.add("nba");
        list.add("nba");
        list.add("cba");
        //添加整数数据，这里自动装箱成为了Integer类型了
        list.add(100);
        list.add(32);
        //使用Collection接口中的remove函数删除元素
        /*boolean boo = list.remove("nba");
```

```
        System.out.println(boo);
        //输出删除后的集合数据
        System.out.println(list);*/
        //删除集合中的100的数据
        /*
         * List接口中含有特有的remove函数，根据指定的下标来删除对应的集合中的数据
         * List接口也继承到了Collection接口中的remove函数，而Collection接口中的remove
         * 函数接收的是一个对象。
         * 因此我们在使用List中的remove方法时，如果参数指定的是一个整数类型的数据，这时，这个值
         * 是不会被装箱成为Integer类型，而这个int值仅仅只是个下标
         */
        //这里使用list调用的是List集合中的特有的remove函数，不是Collection集合中的，
        //而这里的100表示下标，不是对象数据，所以这里会报异常IndexOutOfBoundsException
        //Object obj = list.remove(100);
        //System.out.println(obj);
        //如果我们想使用list对象调用remove函数删除list集合中的100，可以将100转换为对象
        //Integer.valueOf(100)表示将int类型的100包装成Integer类的对象，
        //这里调用的是Collection中的remove函数
        boolean boo = list.remove(Integer.valueOf(100));
        System.out.println(boo);
        System.out.println(list);
    }
```

注意:

List接口中有个特有的remove函数，是根据下标来删除集合中的元素，List接口也继承到Collection接口中的remove函数，而Collection接口中的remove函数接收的是一个对象。因此我们在使用List的remove函数的时候，如果指定的是一个int类型值，这时这个int值不会被自动装箱成Integer类型，而这个int值仅仅只是下标。

四、修改方法

分析和步骤:

1）在上述ListDemo类中定义一个method_5()函数;

2）在method_5()函数中使用new关键字创建ArrayList类的对象list，并赋值给接口List类型;

3）使用集合对象list调用属于Collection接口中的add(E e)函数向集合List中添加几个字符串数据;

4） 使用集合对象list调用属于接口List中特有的函数set(int index ,E element)根据指定的下标修改集合中对应的数据;

5）使用输出语句输出list集合;

```java
// 演示List集合中的set方法
public static void method_5() {
    List list = new ArrayList();

    list.add("aaaa");
    list.add("anc");
    list.add(100);
    list.add(32);

    System.out.println(list);

    //使用list中的特有方法 set修改集合中的元素
    list.set(2, "nba"); //set是修改指定位置上的对象，原来位置上的对象就没有了
    list.add(2,"cba"); // add是在指定的位置上插入元素，原来位置上的元素会自动往后移动
    System.out.println(list);
}
```

注意：

1）set()函数修改指定的位置上的对象，原来位置上的对象就没有了；

2）而之前讲的add()函数是在指定的位置上插入元素，原来位置上的元素会自动往后移动的；

总结：List接口中的特有的增删改查：

增：add(int index , Object element)

删：remove(int index)

改：set(int index , Object newElement)

查：get(int index)

3.4.数据结构(简单介绍)

前言：数据结构，入门学习的朋友作为了解即可，不必深究！春招、秋招、应届生想要进大厂或者进一二线互联网公司的朋友，需要深入学习，掌握扎实的原理和知识点，这里入门课程我作为简单介绍，在后面加强课，同学们可以对号入座，有需要的就去学！

数据结构：数据的存储方式，不同的集合容器它们存储数据的方式都不一样。而我们学习众多的集合容器，重点是知道每个集合存储数据的方式即可。不同集合的存储方式不同，导致集合中的数据存取，以及元素能否重复等相关操作也都不相同。

3.4.1数据结构之栈和队列【记忆】

- 栈结构

  先进后出或者后进的先出。（**手枪的弹夹**）

- 队列结构

  先进先出或者后进的后出。（**排队买票**）

3.4.2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

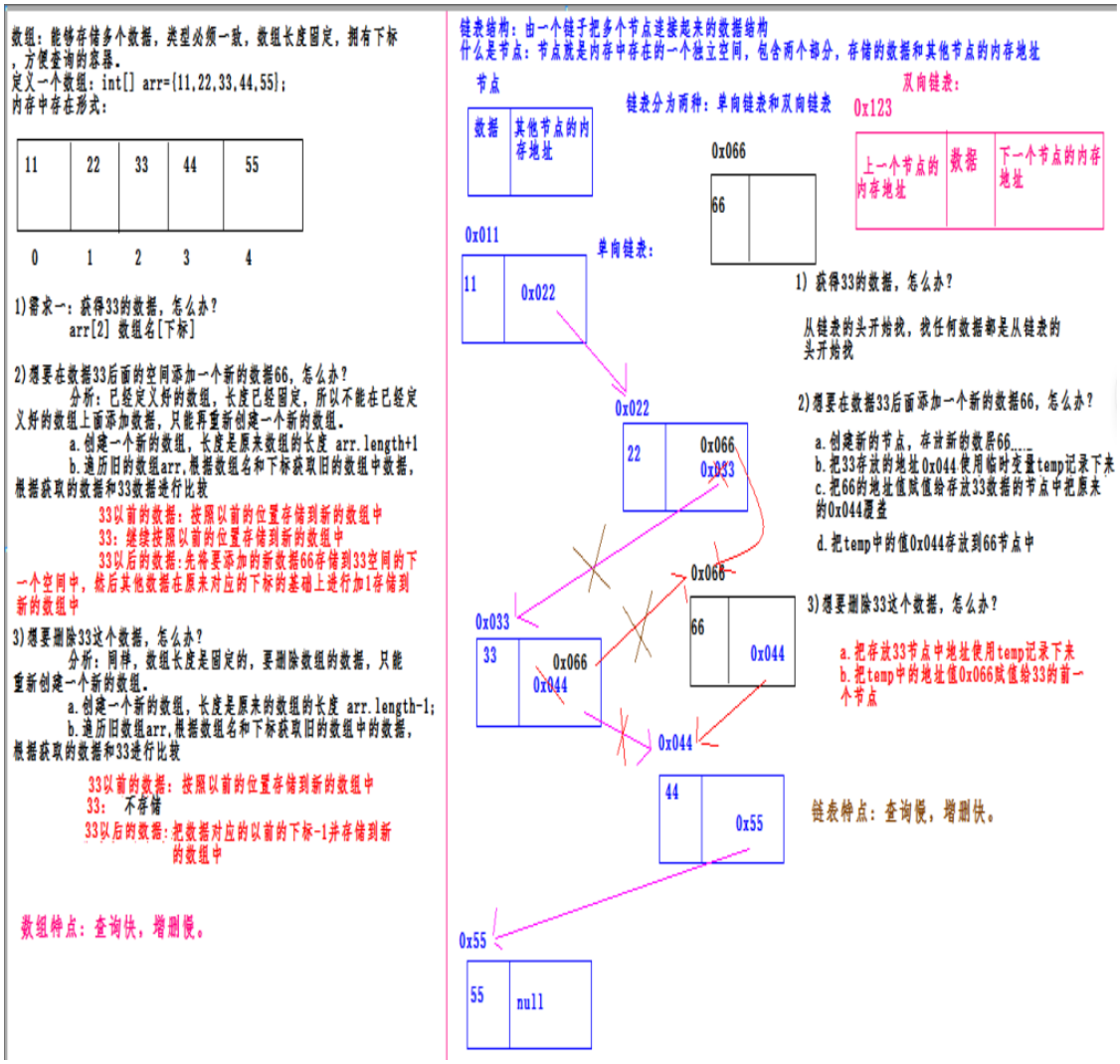
  查询慢、增删快

1、链表的简单介绍：

链表结构：由一个链子把多个节点连接起来的数据结构。
节点：在链表结构中每个可以保存数据的空间称为节点，而一个链表结构是由多个节点组成的。也就是说链表结构使用节点来存储数据。每个节点(存储数据的空间)可以分成若干部分，其中有一部分存储数据，另外一部分存储的是其他节点的地址。

说明：
1）节点：实际存储自己的数据+其他节点的地址，而作为链表结构中的最后一个节点的存储地址的空间是null。
2）链表结构查询或遍历时都是从头到尾的遍历。
3）链表结构是有头有尾的。因此LinkedList集合中定义了自己的特有的方法都是围绕链表的头和尾设计的。
4）链表分为两种：单向链表和双向链表。

2、数组结构和链表结构对数据的查询、添加、删除的区别对比图如下所示：



3.4.3、简单介绍数据结构常见的知识点

1、二叉树

2、二叉查找树

3、平衡二叉树

4、红黑树