今日学习内容：

- 数组的拷贝
- 数组的冒泡排序
- 数组的二分查找

今日学习目标：

- 了解数组的冒泡排序
- 了解数组的二分查找
- 会调用API中操作数组的常用方法，比如：arraycopy, sort，binarySearch，toString方法
- 熟悉数组元素的增删改查操作,并对面向对象封装的概念升华领悟

数组拓展

1.1 数组拷贝（掌握）

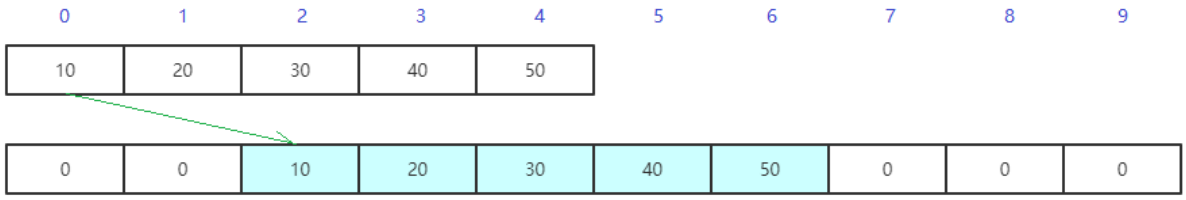
需求：定义一个方法arraycopy, 从指定源数组中从指定的位置开始复制指定数量的元素到目标数组的指定位置。

例如：把源数组(srcArr)从0开始，复制5个元素到目标数组(destArr)的索引为2的位置上。

拷贝前:

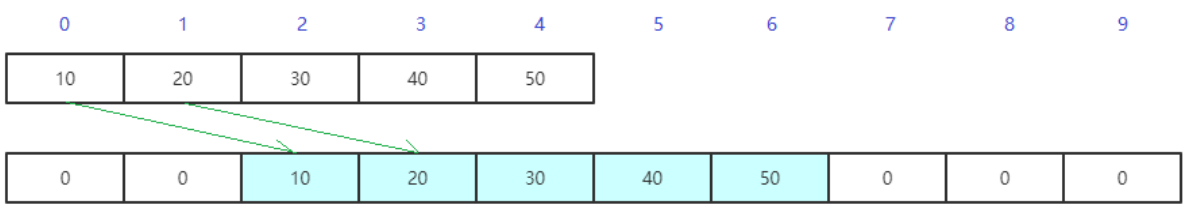
拷贝过程:

第1步：




```
destArr[2] = srcArr[0]
```

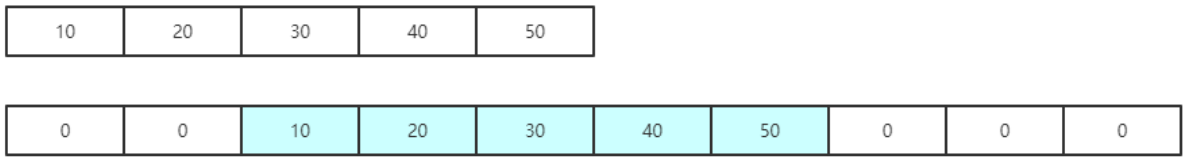
第2步：



```
destArr[3] = srcArr[1]
```

以此类推...

拷贝后:



代码如下:

```
public class ArrayUtils {
    private ArrayUtils() {}

    /**
     * 复制数组操作
     *
     * @param src     源数组
     * @param srcPos  源数组中的开始索引位置
     * @param dest    目标数组
     * @param destPos 目标数据中的开始索引位置
     * @param length  要复制的数组长度
     */
    public static void arraycopy(int[] src, int srcPos, int[] dest, int destPos,
int length) {
        for (int index = 0; index < length; index++) {
            dest[destPos + index] = src[srcPos + index];
        }
    }
}
```

测试结果

```
public class Test01ArrayCopy {
    public static void main(String[] args) {
        int[] srcArr = new int[]{10, 20, 30, 40, 50, 60, 70};
        int[] destArr = new int[10];

        ArrayUtils.arraycopy(srcArr, 2, destArr, 4, 3);

        // 测试复制数组后的结果
        for (int i = 0; i < destArr.length; i++) {
            System.out.print(destArr[i] + ",");
        }
    }
}
```

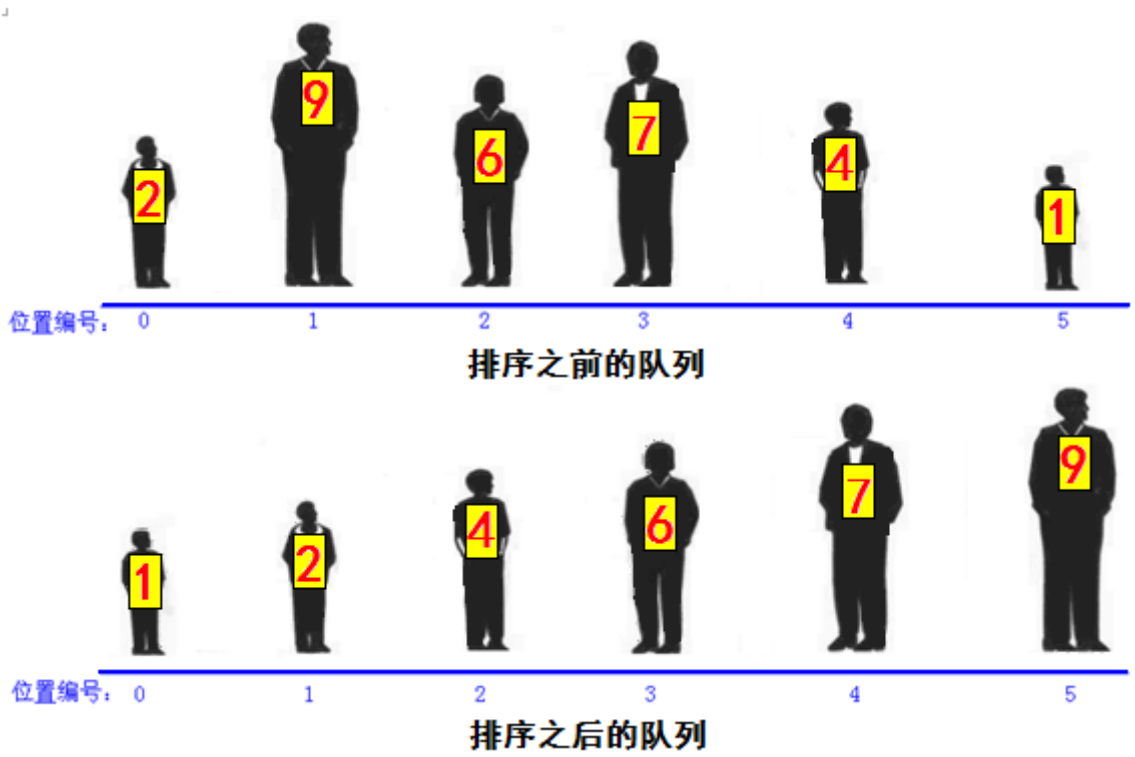
输出结果:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 30, 40, 50, 0, 0, 0]

1.2. 排序操作（掌握原理）

需求：完成对int[] arr = new int[]{2,9,6,7,4,1}数组元素的升序排序操作.



1.2.1.冒泡排序原理（掌握原理）

对未排序的各元素从头到尾依次比较相邻两个元素的大小关系，如果前一个元素大于后一个元素则交换位置，经过第一轮比较后可以得到最大值，同理第二轮比较后出现第二大值等。

针对int[] arr = new int[]{ 2, 9, 6, 7, 4, 1 }数组元素做排序操作：

该数组有6个元素，只需要5轮比较。

```
第1轮比较：需要比较5次，比较完出现第一个大值。
第2轮比较：需要比较4次，比较完出现第二大值。
第3轮比较：需要比较3次，比较完出现第三大值。
...
可以看出如有N个元素，则需要N-1轮比较，第M轮需要N-M次比较。
```

交换数组中两个元素的方法

```
public class ArrayUtils {
    private ArrayUtils() {}

    // 通过索引交换数组索引处值
    public static void swap(int[] arr, int index1, int index2) {
        int temp = arr[index1];
        arr[index1] = arr[index2];
        arr[index2] = temp;
    }
}
```

排序代码：

```
public class ArrayUtils {
    private ArrayUtils() {}

    public static void swap(int[] arr, int index1, int index2) {
        int temp = arr[index1];
        arr[index1] = arr[index2];
        arr[index2] = temp;
    }

    public static void bubbleSort(int[] arr) {
        //第一轮：
        for (int i = 1; i <= arr.length-1; i++) {
            if (arr[i - 1] > arr[i]) {
                swap(arr, i - 1, i);
            }
        }

        //第二轮：
        for (int i = 1; i <= arr.length - 2; i++) {
            if (arr[i - 1] > arr[i]) {
                swap(arr, i - 1, i);
            }
        }

        //第三轮：
        for (int i = 1; i <= arr.length - 3; i++) {
            if (arr[i - 1] > arr[i]) {
                swap(arr, i - 1, i);
            }
        }

        // 第四轮呢？
    }
}
```

寻找规律，优化上述代码：

```
public static void bubbleSort(int[] arr) {
    for (int times = 1; times <= arr.length - 1; times++) {
        //times表示第几轮比较，值为：1,2,3,4,5
        for (int i = 1; i <= arr.length - times; i++) {
            if (arr[i - 1] > arr[i]) {
                ArrayUtils.swap(arr, i - 1, i);
            }
        }
    }
}
```

测试

```
public class TestO2ArraySort {
    public static void main(String[] args) {

        int[] arr = new int[] { 2, 9, 6, 7, 4, 1 };
        ArrayUtils.bubbleSort(arr);
        // 测试复制数组后的结果
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + ",");
        }
    }
}
```

1.3 二分法查找（掌握原理）

查找数组元素的算法：

- 线性查找：从头找到尾，性能比较低。
- 二分法查找（折半查找）：前提**数组元素是有序的**，性能非常优异。

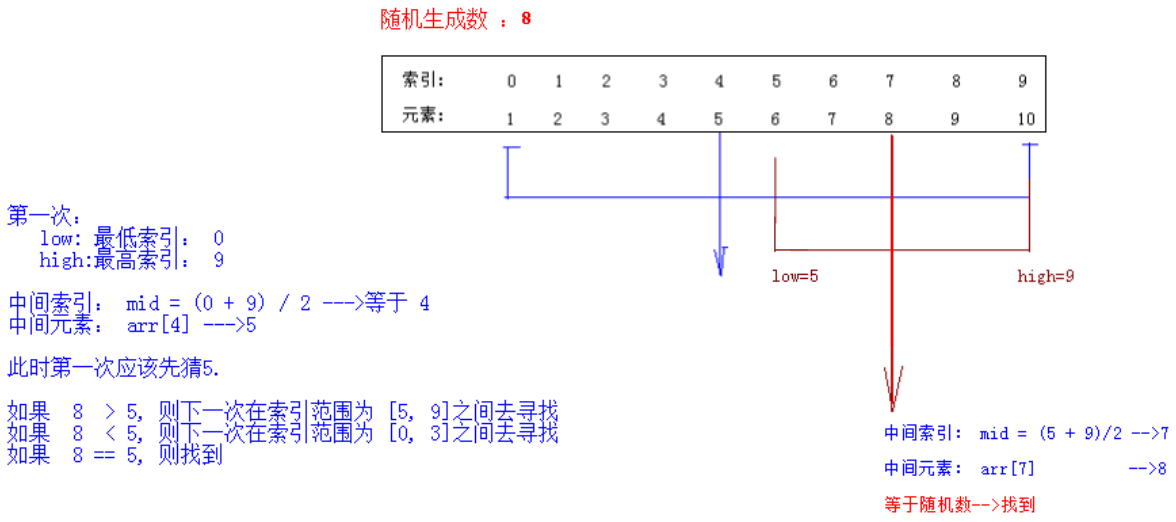
1.3.1 二分法搜索原理（掌握原理）

猜数字游戏：电脑随机生成一个[1 , 100]之间的商品价格，等玩家来猜，猜之后，电脑提示出三种结果：你猜的数偏大，偏小和猜中了。

此时为了以最少次数猜中，玩家决定

先猜（1+100）/ 2的商50，如果此时电脑提示猜的偏小了，那就能推断出该随机数在[51 , 100]之间，此时再猜（51+100）/ 2的商75，如果电脑提示猜大了，那么该随机数必在[51 , 74]之间，那么继续猜（51+74）/2的商，如此每次都可以排除剩下结果一半的可能性，直到猜到为止。

关键字：**以最少的次数猜中**，在规定的时间(10分钟)猜中最多。



代码如下：

```
public class ArrayUtils {
    private ArrayUtils() {}

    // 二分查找法
    public static int binarySearch(int[] arr, int key) {
        int low = 0;
```

```
        int high = arr.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            int midVal = arr[mid];

            if (midVal < key) {
                low = mid + 1;
            } else if (midVal > key) {
                high = mid - 1;
            } else {
                return mid;
            }

        }
        // 没有找到
        return -1;
    }
}
```

测试

```
public class Test03ArraySearch {
    public static void main(String[] args) {

        int[] arr = new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9};
        int idx = ArrayUtils.binarySearch(arr,8);
        System.out.println("idx = " + idx);
    }
}
```

1.4. 操作数组的API-Arrays（掌握）

类似打印数组元素的这样的工具性的方法，其实SUN公司的攻城狮们早就写好代码了，并封装在了很多工具类中，我们把这种预先定义好的方法，称为API。对于我们而言，最基本的要求就是能调用这些方法，当然我们对自己有更高的要求，应该知其然，并知其所以然。

学习API一定要掌握一个秘诀：**文档在手, 天下我有! 一定要经常性的查文档！！**

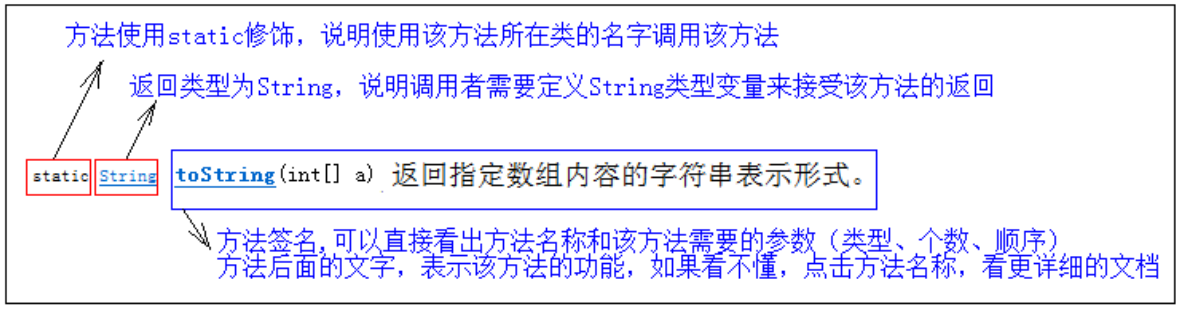
Arrays工具类中的方法，一般都是使用static修饰的。

打开JDK帮助文档，搜索Arrays类，进入该类的文档页面，去找toString方法，发现在Arrays类中有多个toString方法，他们之间属于重载关系，分别用于打印不同类型的数组。

static String	**toString**(boolean[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(byte[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(char[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(double[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(float[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(int[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(long[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(Object[] a) 返回指定数组内容的字符串表示形式。
static String	**toString**(short[] a) 返回指定数组内容的字符串表示形式。

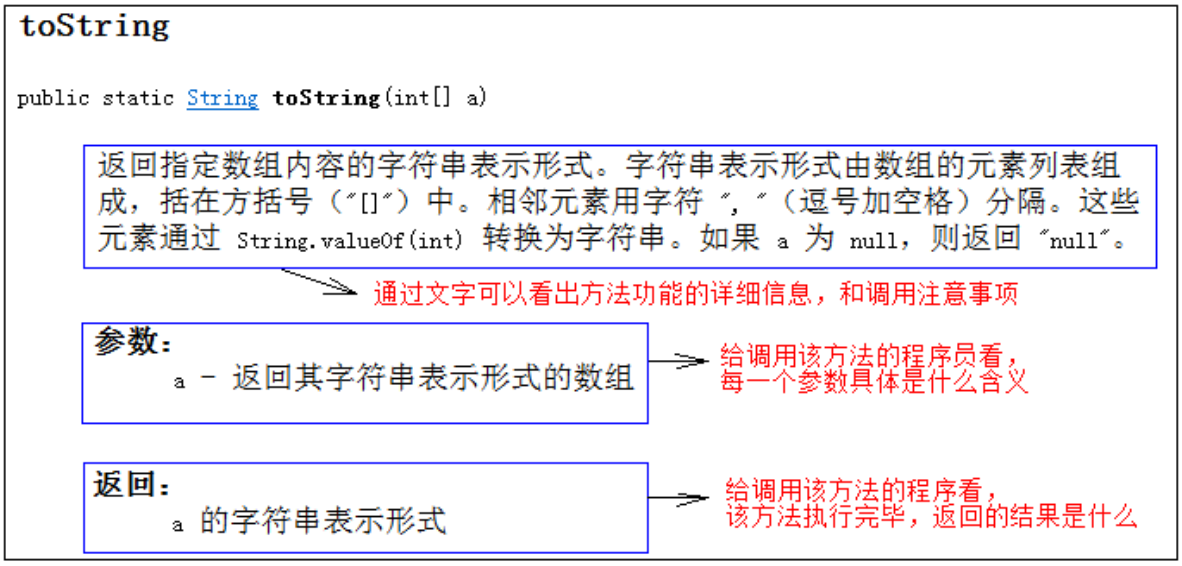
如: 查看Arrays类中将int类型数组转换成字符串的toString方法。

方法列表：



如果看方法列表看不懂怎么使用，使用鼠标左键点击该方法名称，进入该方法的详细：

方法详细：



如果看不懂，就要静下心来多看几次，必须掌握每一部分到底在表达什么意思。

1.4.1. 打印数组元素（会用）

API中还有一个专门操作数组的工具类Arrays，该类提供了对数组元素的拷贝、元素搜索、元素排序、打印等功能方法，且该类为不同数据类型的数组都重载了相同功能的方法。

需求：通过调用Arrays类中的toString方法完成打印数组元素的功能，掌握如何给类定义包、导入类以及看API文档。

```
public class Test01ToString {
    public static void main(String[] args) {
        int[] arr = new int[] { 10, 20, 30, 40, 50, 60, 70 };
        String str = Arrays.toString(arr);
        System.out.println(str);
    }
}
```

1.4.2. 拷贝数组元素（会用）

Arrays 中提供了数组复制的方法，copyOf(int[] original, int newLength) 复制指定的数组，截取或者用0填充。

System类中提供了数组元素拷贝的方法，并且支持任意类型的数组拷贝，而不仅仅是int类型数组。

```
public class Test02ArrayCopy {
    public static void main(String[] args) {
        // copyOf
        // 截取场景
        int[] arr = new int[] { 10, 20, 30, 40, 50, 60, 70 };
        // int[] newArr = Arrays.copyOf(arr, 3);

        // 填充场景
        int[] newArr = Arrays.copyOf(arr, 10);
        System.out.println(Arrays.toString(newArr));

        // System.arraycopy()
        int[] arr2 = {1,2,3,4};
        int[] newArr2 = new int[10];
        System.arraycopy(arr2,0,newArr2,0,arr2.length);
        System.out.println(Arrays.toString(newArr2));
    }
}
```

1.4.3. 数组元素排序（会用）

Arrays类中已经提供了数组排序的方法sort，并且是调优之后的，性能非常优异，在开发中只需要我们直接调用该方法即可即可。

```java
public class Test03Sort{
    public static void main(String[] args) {
        int[] arr = new int[] { 2, 9, 6, 7, 4, 1 };
        System.out.println(Arrays.toString(arr));//排序前
        Arrays.sort(arr);
        System.out.println(Arrays.toString(arr));//排序后
    }
}
```

1.4.4. 数组元素二分查找（会用）

Arrays类中已经提供了数组元素的二分查找。

```java
public class Test04Search{
    public static void main(String[] args) {
        int[] arr = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int index = Arrays.binarySearch(arr, 8);
        System.out.println(index);
    }
}
```

小结：排序和二分法查找的原理需要掌握，当然，在实际开发中，会调用Arrays类中方法完成相关功能即可。

1.5 数组元素的增删改查操作（掌握）

假设我现在是某个篮球队的教练，需要安排5个球员上场打球。此时需要模拟上场球员的存储，简单一点，我们就只存储上场球员的球衣号码。那么此时我需要以下几个操作：

1．**初始一个容量**为5的容器，用来存储场上的5个球衣号码。

2．安排5个球员上场，比如球员号码分别为11、22、33、44、55。

3．**查询**指定索引位置球员的球衣号码是多少，如查询索引位置为2的球衣号码是33。

4．**替换**场上索引位置为2的球员，使用333号替换33号。

5．**罚下**场上索引位置为2的球员（直接罚下，没有补位）。

6．**打印**出场上球员的球衣号码，打印风格如 [11，22，33，44，55]。

操作前后效果图：

操作前：

索引	0	1	2	3	4
元素					

操作后：

索引	0	1	2	3	4
元素	11	22	33	44	55

思考1：用于什么存储上面的号码？，需不需要一个存储号码的容器？你想到谁？

思考2：试想给你一个数组 `[11,22,33,44,55]`，会不会对数组容器中的元素进行**增**加、**删**除、修**改**、**查**询

思考3：试想我再给你一个数组 `[10,20,30,40]` 还需要你增删改查，刚才写的代码能复用吗？

思考4：我们能不能对封装思维的理解进一步升华？

1.5.1 初始化操作

使用Integer数组来存储场上球员号码，提供了两个构造器，一个用于自定义初始化容量，一个用于 使用默认的初始化容量10。

```
public class PlayerList {
    // 存储场上球员的号码
    private Integer[] players = null;
    // 记录场上球员的数量
    private int size = 0;

    public PlayerList(int capacity){
        if(capacity < 0){
            System.out.println("初始容量不能为负数");
            return;
        }
        this.players = new Integer[capacity];
    }

    public PlayerList() {
        this(0);
    }
}
```

测试代码

```
public class Test01PlayerList {
    public static void main(String[] args) {
        //初始容量为默认的10
        PlayerList list1 = new PlayerList();
        //自定义初始化容量为5
        PlayerList list2 = new PlayerList(5);
        //自定义初始化容量为20
        PlayerList list3 = new PlayerList(20);
        }
    }
}
```

1.5.2 追加操作

```
// 向球场上添加一个球员(号码)
public void add(Integer number){
    this.players[size] = number;
    // 球员个数增加1
    size++;
}
```

测试代码

```
public class Test01PlayerList {
    public static void main(String[] args) {

        //初始化容器,设置初始化容量为5
        PlayerList list = new PlayerList(5);
        System.out.println(list);

        //向容器中添加5个元素（球员号码）
        list.add(11);
        list.add(22);
        list.add(33);
        list.add(44);
        list.add(55);
        //打印容器中每一个元素
        System.out.println(list);
    }
}
```

因为数组的长度是固定的，此时的players数组只能存储5个元素，如果再多存储一个就报错：数组索引越界。此时就要考虑在保存操作时对数组做扩容操作。

扩容的原理是：

- 创建一个原数组长度两倍长的新数组
- 把旧数组中的所有元素拷贝到新数组中
- 把新数组的引用赋给旧数组变量

保存操作时扩容操作：

```
// 向球场上添加一个球员(号码)
public void add(Integer number){
    // 检测容器容量是否已满，如果已满，需要拓容。
    if(size == players.length){
        this.players = Arrays.copyOf(this.players, size * 2);
    }

    // ---------------------------
    this.players[size] = number;
    // 球员个数增加1
    size++;
}
```

1.5.3 打印输出操作

```
public String toString() {
    if(null == players){
        return "null";
    }

    if(0 == size){
        return "[]";
    }

    StringBuilder sb = new StringBuilder();
    sb.append("[");
    for(int index = 0;index < size;index++){
        sb.append(this.players[index]);

        if(index != size - 1){
            sb.append(",");
        }else{
            sb.append("]");
        }
    }

    return sb.toString();
}
```

1.5.4 查询操作

需求：查询指定索引位置球员的球衣号码是多少，如查询索引位置为2的球衣号码是33。

其实就是返回数组中，指定索引对应的元素值。

```
public Integer get(int index) {
    if (index < 0 || index >= size) {
        System.out.println("索引越界");
        return null;
    }
    return players[index];
}
```

测试代码：

```
public class Test01PlayerList {
    public static void main(String[] args) {
        Integer r = list.get(2);
        System.out.println(r);
    }
}
```

1.5.5 修改操作

需求：替换场上索引位置为2的球员，使用333号替换33号

```
//替换指定位置的球员号码
public void set(int index, Integer newPlayerNumber) {
    if (index < 0 || index >= size) {
        System.out.println("索引越界");
        return;
    }
    this.players[index] = newPlayerNumber;
}
```

1.5.6 删除操作

需求：罚下场上索引位置为2的球员（直接罚下，没有补位）。

删除操作的原理，把后续的元素整体往前挪动一个位置。

```
//删除指定位置的球员号码
public void remove(int index) {
    if (index < 0 || index >= size) {
        System.out.println("索引越界");
        return;
    }

    for (int i = index; i < size - 1; i++) {
        players[i] = players[i + 1];
    }

    players[size - 1] = null;
    size--;
}
```

测试代码

```
public class Test01PlayerList {
    public static void main(String[] args) {

        PlayerList list = new PlayerList(5);
        list.add(11);
        list.add(22);
        list.add(33);
        list.add(44);
        list.add(55);

        list.remove(2);
        System.out.println(list);
    }
}
```

1.5.7 让容器支持存储任意数据类型的元素

此时元素类型是Integer类型，也就是只能存储整型的数据，但是却不能存储其他类型的数据，此时我们可以考虑吧元素类型改成Object，那么Object数组可以存储任意类型的数据。

```
import java.util.Arrays;

public class MyArrayList {
    // 存储元素容器
    private Object[] elementData = null;
    // 记录元素个数
    private int size = 0;

    // 自定义初始容量
    public MyArrayList(int initialCapacity) {
        if (initialCapacity < 0) {
            System.out.println("初始容量不能为负数");
            return;
        }
        this.elementData = new Object[initialCapacity];
    }

    // 默认初始容量为10
    public MyArrayList() {
        this(10);
    }

    // 向容器中添加一个元素
    public void add(Object e) {
        // 如果容器容量已满,此时需要扩容，此时扩容机制为原来容量的2倍
        if (size == elementData.length) {
            this.elementData = Arrays.copyOf(this.elementData, size * 2);
        }
        // ----------------------------------------
        this.elementData[size] = e;
        size++;// 容器中元素数量加1
    }

    // 查询指定位置的元素
    public Object get(int index) {
        if (index < 0 || index >= size) {
            System.out.println("索引越界");
            return null;
        }
        return this.elementData[index];
    }

    // 替换指定索引位置的元素
    public void set(int index, Object e) {
        if (index < 0 || index >= size) {
            System.out.println("索引越界");
            return;
        }
        this.elementData[index] = e;
    }

    // 删除指定索引位置的元素
```

```
    public void remove(int index) {

        if (index < 0 || index >= size) {
            System.out.println("索引越界");
            return;
        }
        for (int i = index; i < size - 1; i++) {
            this.elementData[i] = this.elementData[i + 1];
        }
        this.elementData[size - 1] = null;
        size--;
    }

    public String toString() {
        if (elementData == null) {// 如果没有初始化容器
            return "null";
        }
        if (size == 0) {// 如果容器中元素数量为0
            return "[]";
        }
        StringBuilder sb = new StringBuilder(40);
        sb.append("[");
        for (int index = 0; index < size; index++) {
            sb.append(this.elementData[index]);
            // 如果不是最后一个元素
            if (index != size - 1) {
                sb.append(",");
            } else {
                sb.append("]");
            }
        }
        return sb.toString();
    }
}
```