

【干货】用大白话聊聊JavaSE — ArrayList 深入剖析和Java基础知识详解（二）

在上一节中，我们简单阐述了Java的一些基础知识，比如多态，接口的实现等。

然后，演示了ArrayList的几个基本方法。

ArrayList是一个集合框架，它的底层其实就是一个数组，这一点，官方文档已经说得很清楚了。

作为一个容器，ArrayList有添加元素，删除元素，以及获取元素的方法。

本节我们先不看ArrayList底层的源码，而是按照平常的思路来模拟一下ArrayList的具体实现。看看如果我们自己来写的话，会怎么实现ArrayList的功能？

1. 新建一个MyList类

好的，我们来模拟一下ArrayList类，怎么模拟呢，是不是这样就行了？

```
import java.util.ArrayList;

public class MyList extends ArrayList{

}
```

写完了。

额，开个玩笑，别打我。。。

好的，让我们开始吧。

这个MyList类，主要用来模拟一下ArrayList的基本方法，我们新建一个MyList 类

```
package jianshu;

public class MyList {

}
```

现在的MyList是不是啥也没有啊，就好像一个新生的婴儿一样，纯洁得像一张白纸。

我们现在需要给MyList添加一个新的身份。

添加身份，不就是实现接口或者继承某个类么？

原先的ArrayList因为继承了List接口，所以必须实现List接口所有的抽象方法，我们为了简单起见，就不去实现List接口了。

我们来定义一个简单的List接口，名字就叫做 SimpleList 吧。
里面定义几个常用的抽象方法。

```
package jianshu;

/**
 * 简单的List接口
 * @author 剽悍一小兔
 *
 */
public interface SimpleList {

    /**
     * 添加元素
     * @param obj
     * @return boolean
     */
    boolean add(Object obj);

    /**
     * 根据元素下标删除元素
     * @param index
     */
    void remove(int index);

    /**
     * 根据元素下标获取对应的元素
     * @return Object
     */
    Object get(int index);

    /**
     * 将当前的SimpleList转换成Object数组
     */
}
```

```

        * @return Object[]
        */
        Object[] toArray();

        /**
         * 获取当前列表中元素的个数
         * @return int
         */
        int size();
    }

```

现在，让MyList实现这个接口。这步操作，就相当于给MyList添加一个新的身份。

因为MyList可以变身成为SimpleList，那么就必须拥有SimpleList的所有能力。

所以，我们是不是必须要实现SimpleList中所有的抽象方法呢？

```

package jianshu;

public class MyList implements SimpleList{

    public boolean add(Object obj) {
        return false;
    }

    public void remove(int index) {

    }

    public Object get(int index) {
        return null;
    }

    public Object[] toArray() {
        return null;
    }

    public int size() {
        return 0;
    }

}

```

接着，定义一个测试类，专门用来测试MyList

```

package jianshu;

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

    }
}

```

2. 构造函数设计

2.1 容器选型

我们完全按照ArrayList的规范来，打开api，发现其实ArrayList不止一个构造方法。

ArrayList有三个构造方法，分别为

ArrayList() --- 空构造方法。

ArrayList(Collection<? extends E> c) --- 传入参数为一个Collection对象。

ArrayList(int initialCapacity) --- 传入参数为一个int类型的数字，initialCapacity表示容量，在ArrayList被new出来的时候就规定一下初始容量是多少。

我们知道，Java在定义数组的时候，必须有一个长度。

比如：

```
Object[] objs = new Object[3];
```

这样我就定义了一个长度为3的数组。

这个是显示定义的。

当然还可以这样：

```
Object[] objs = new Object[]{1,2,3};
```

虽然没有明确指出数组的长度是多少，但是我们都知它的长度就是3，这属于隐式定义。

我们的MyList本身没有存储数据的能力，为了让它具备这方面的能力，是不是要给他定义一个属性啊。

一个Java类，无非就是属性和方法，大部分情况下，方法无非就是用来给属性赋值的。

属性是干嘛用的，不就是用来存储数据的吗？

你说对不对呢？

Java有八种基本数据类型：

分别为整型 int，短整型 short，长整型 long，字节型 byte，布尔型 boolean，字符型 char，单精度浮点数 float，双精度浮点数 double。

比如说int，Java编译器规定int占四个字节，也就是4个byte，一个字节占8位，一个位就是一个bit。

bit是计算机中最小的单位，它只有0和1两种状态。

我们常说一个文件有多少兆，这个兆就是MB，1MB有1024KB，1KB有1024个字节。

当你定义了一个int类型的变量，在运行的时候就会在Java虚拟机中申请一个4个字节的空间。

1MB有1024KB，1KB有1024个字节。所以说1MB可以存放 $(1024 * 1024 / 4) = 262144$ 个int变量。

Java虚拟机的默认内存是64MB，所以最多应该能存放16777216个int类型的变量。

当你定义一个int类型的变量，那么运行的时候，虚拟机的剩余内存就会被减掉4个字节。

所以，属性是干嘛用的，我们在写Java类的时候，为什么要定义属性。

我觉得没有别的含义了，定义属性就是为了存储数据的嘛。

我们写一个

```
private int a;
```

Java虚拟机（JVM）跑起来，一旦我们new了这个对象。

这个a变量就会被放到JVM的内存中，然后JVM就会专门开辟一个空间，来装载这个数据。

然后，我们才可以在计算机中操作这些个数据。

你总不可能说，我有一个数字100，就要计算机对这个数字进行加减乘除的运算吧。

计算机怎么知道这个事情呢？

你是不是必须要告诉计算机有一个数字100，它才会知道？

为了装载这些数据，所以才有了八种基本数据类型，每一个数据类型就好比一个篮子，有的篮子大一点，比如long类型，可以放好长好长的数字。有的篮子小一点，比如byte类型，只能放一点点大的数字。

言归正传，我们的 MyList 也需要像ArrayList那样，可以add，也可以get。

那么，我们是不是必须要有一个属性，用来储存这些数据呢？

很显然，Java给我们提供的8中基本数据类型都无法满足这个需求。

接下来，我们想到，是不是可以定义一个数组，作为我们的容器呢？

数组，严格来说也是一个类，直接继承自Object。我们不是可以通过new来生成一个数组对象吗？而且数组拥有一个length属性，这些证据都足以说明数组也是一个类。

为了验证这一点，我们来做个实验。

```
Object arr = new Object[12];
```

这样写，是不会报错的，说明数组也拥有一个 Object 身份。这又是多态，我们怀疑数组是否继承自Object，用多态的写法去验证一下就好了。

基本上确定下来了，我们就采用 Object 数组作为存储数据的容器吧。

```
private Object[] elementData;
```

2.2 数组容量初始化

容器选型完毕后，开始着手设计构造函数。因为我们的底层采用数组来作为存储数据的媒介，而数组这个东西，我们知道是要有一个初始容量的。

那么，我们是不是可以用构造函数的方式来给数组进行初始化呢？

```
public MyList(int initialCapacity){
    this.elementData = new Object[initialCapacity];
}
```

这样就行了。

为了看效果，我们需要有一个方法来获取数组中的值，所以现在来改写一个toString方法。

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();

    sb.append("[");
    for (int i = 0; i < elementData.length; i++) {
        sb.append(elementData[i].toString()).append(",");
    }

    //去掉最后一个逗号
    String str = sb.toString().substring(0, sb.toString().length() - 1);

    str += "]";

    return str;
}
```

3. add方法实现

我们通过add方法来给数组添加数据

我们给数组添加元素的方式是这样的：

```
Object[] arr = new Object[12];
arr[0] = "Hello";
arr[1] = "World";
System.out.println(arr[0]);
System.out.println(arr[1]);
```

可见，添加元素的时候，我们必须要知道它的下标。

为了方便，我们增加一个私有属性size，来存储当前数组中 **实际存在** 的元素个数。

```
private int size;
```

size默认是0

当我们调用add方法的时候，只需要动态地给 size + 1 就行了。

add方法初步实现：

```
/**
 * 添加新的元素
 */
public boolean add(Object obj) {
    elementData[size ++] = obj;
    return true;
}
```

return true 代表添加成功。

测试：

```
SimpleList list = new MyList(3);

list.add("Hello");
list.add("World");
list.add("Java");

System.out.println(list);
```

```
[Hello,World,Java]
```

可见，的确是成功添加进去了。

MyList的容量为3，我就添加了3个元素。如果我添加两个呢？

```
SimpleList list = new MyList(3);

list.add("Hello");
list.add("World");

System.out.println(list);
```


一运行，报错了：

```
Exception in thread "main" java.lang.NullPointerException
    at jianshu.MyList.toString(MyList.java:52)
    at java.lang.String.valueOf(String.java:2854)
    at java.io.PrintStream.println(PrintStream.java:821)
    at jianshu.TestMyList.main(TestMyList.java:12)
```

从错误信息中可以看出，我们的toString方法报错了。

错误位置是第52行。

```
46 @Override
47 public String toString() {
48     StringBuilder sb = new StringBuilder();
49
50     sb.append("[");
51     for (int i = 0; i < elementData.length; i++) {
52         sb.append(elementData[i].toString()).append(",");
53     }
54
55     //去掉最后一个逗号
56     String str = sb.toString().substring(0, sb.toString().length() - 1);
57
58     str += "];";
59
60     return str;
61 }
```



```
for (int i = 0; i < elementData.length; i++) {
    sb.append(elementData[i].toString()).append(",");
}
```

这个for循环出了问题，原因很简单，因为数组的长度有3个，而我们只添加了两个元素。

也就是说，只有elementData[0]，和elementData[1]有数据，

而elementData[2] = null

null.toString() 当然就报错了。

来改一下咯，最先想到的应该是修改for循环中的elementData.length，显然我们不应该用elementData的长度来作为总个数，要知道，elementData.length是数组的长度，而非数组中实际元素的个数。

所以，我们是不是应该要采用size属性呢？

```
for (int i = 0; i < size; i++) {
    sb.append(elementData[i].toString()).append(",");
}
```

再运行一次，

[Hello,World]

OK了。

4. remove方法实现

remove方法传入一个int类型的数字，这个就是需要删除的元素下标。

我们根据这个下标找到这个元素。当然，还得确保你传入的参数不能小于0，也不能超过数组中实际元素的个数。

remove方法初稿：

```
/**
 * 根据下标删除元素
 */
public void remove(int index) {
    if(index < 0 || index >= size){
        throw new IndexOutOfBoundsException("您输入的下标为: "+index+",而数组中最大的下标为: "+(size - 1));
    }
    elementData[index] = null;
}
```

测试一下，如果我们删除下标为3的元素，看看会怎样？

```
public class TestMyList {
    public static void main(String[] args) {
        SimpleList list = new MyList(3);

        list.add("Hello");
        list.add("World");
        list.add("Java");

        list.remove(3);

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

报错了。

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: 您输入的下标为: 3,而数组中最大的下标为: 2
    at jianshu.MyList.remove(MyList.java:35)
    at jianshu.TestMyList.main(TestMyList.java:13)
```

这个异常信息是我们自己定义的。

提示信息已经很清楚了，他说您输入的下标为：3,而数组中最大的下标为：2。

| 0 | 1 | 2 |
|-------|-------|------|
| Hello | World | Java |

数组中元素个数是3，下标最大为2。

那我们传一个0吧。

```
list.remove(0);
```

又报错了：

```
Exception in thread "main" java.lang.NullPointerException
    at jianshu.MyList.toString(MyList.java:58)
    at java.lang.String.valueOf(String.java:2854)
    at java.io.PrintStream.println(PrintStream.java:821)
    at jianshu.TestMyList.main(TestMyList.java:15)
```

又是空指针，第58行。

它和上面add方法测试的时候报一样的错误，错误代码也一样，也是那个for循环报错。

```
for (int i = 0; i < size; i++) {  
    sb.append(elementData[i].toString()).append(",");  
}
```

因为我们用remove方法删除元素的时候，直接把那个元素赋一个null来达到删除的目的，显然这是不合理的。

那我们换一种思路，比如删除Hello，能不能让后面的元素全部往前移动一个位置呢？

比如我删除Hello，然后数组就变成了这样

| 0 | 1 | 2 |
|-------|------|------|
| World | Java | null |

然后让size减一，表示数组中实际存在的元素个数 - 1。

因为我们的toString方法循环数组的时候，是根据size来的，所以哪怕最后一个位置是null，也不会被遍历到。

好了，现在问题就演变为，我如何才能把要删除的那个元素后面的所有元素，都左移一个单位呢？

方案已经确定了，剩下的就是如何实现的问题。

要是有一个数组拷贝的方法就好了。比如数组有三个元素，下标分别为0,1,2。

我们删除下标为0的元素，只要把下标为1,2的元素拷贝到下标为0,1的地方，就可以了。

还真有这个方法，就是

System.arraycopy(src, srcPos, dest, destPos, length);

参数的含义（看了api，我反复琢磨以后，感觉这样翻译比较好）

src：需要拷贝的数组。

srcPos：从哪里开始拷贝？

dest：目标数组

destPos：从哪里开始粘贴？

length：拷贝的元素个数

改进后的remove方法：

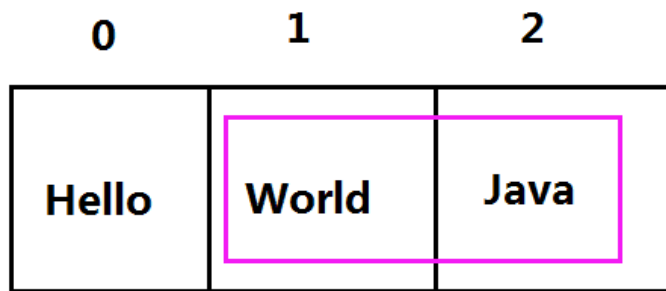
```
/**  
 * 根据下标删除元素  
 */  
public void remove(int index) {  
    if(index < 0 || index >= size){  
        throw new IndexOutOfBoundsException("您输入的下标为: "+index+", 而数组中最大的下标为: "+(size - 1));  
    }  
    System.arraycopy(elementData, index + 1, elementData, index, elementData.length - index - 1 );  
    elementData[elementData.length - 1] = null;  
    size = size - 1;  
}
```

稍微解释一下这句话吧：

System.arraycopy(elementData, index + 1, elementData, index, elementData.length - index - 1);

比如我要删除下标为0的元素，那么需要拷贝的数组和目标数组都是elementData吧，这个没问题。

index 等于 0，表示我要删除下标为0的元素。那么接下来，我是不是要把这两个元素都往左边移动一个单位呀？



← 左移一个单位，把Hello给冲掉。

那么，从哪里开始拷贝？

是不是从下标为1的地方开始拷贝？

也就是从World开始拷贝吧。（所以第二个参数是 $\text{index} + 1$ ）

粘贴到哪？

是不是从Hello的地方开始粘贴，也就是第0个元素。（所以第四个参数是 index ）

拷贝多少个元素呢？

我们需要把Hello后面的两个元素都拷贝一下，所以是2个元素。

可是我们不能直接写一个2吧，`remove`方法可不知道你本来有多少个元素，所以这个地方需要动态计算一下。

即通过数组的最大下标减去需要删除的元素下标。

其实最后一个参数应该写成

`(elementData.length - 1) - index`

这样可能比较好理解。

移动位置后，最后一个元素肯定还是之前的元素，所以我们需要把它赋空。然后`size - 1`，这样`toString`的时候没问题了。不然又会报空指针。

测试：

```
public class TestMyList {
    public static void main(String[] args) {
        SimpleList list = new MyList(3);
        list.add("Hello");
        list.add("World");
        list.add("Java");
        list.remove(0);
        System.out.println(list);
    }
}
```

[World, Java]

如果我要把所有的元素都删掉咋办？

那就连续删3次咯。

在这里我们需要把`toString`方法改一下：

如果 `size` 为 0，就直接返回一个 `[]`

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

测试

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



```
SimpleList list = new MyList(3);  
list.add("Hello");  
list.add("World");  
list.add("Java");  
list.remove(0);  
list.remove(0);  
list.remove(0);  
System.out.println(list);  
}  
}
```

结果:

[]

OK了。

未完待续。