### 什么是 ArrayList 类？

java提供的一个容器 java.util.ArrayList集合类，让我们可以更便捷的存储和操作对象数据。

是大小可变的数组的实现，存储在内的数据称为元素。此类提供一些方法来内部存储的元素，

Array中也可不断添加元素，其大小也是自动增长的。

查看类 import java.util.ArrayList;

public class ArrayList<E> extends AbstractList<E>

implements List<E>, RandomAccess, Cloneable, java.io.Serializable{ …… }

---> <E> 表示一种指定的数据类型，叫做泛型

---> 在出现 E 的位置，我们可以使用引用数据类型将其替换即可，表示我们将存储那种引用类型的元素。

小贴士：什么是引用数据类型

1. 引用数据类型有三种：类class，接口instance，数组Array

2. 引用数据类型一般是通过new来创建的

3. 基本数据类型的封装类也是引用数据类型 int ---> Integer char ---> Character "涉及到jdk1.5中封装类的自动拆箱与自动装箱的过程"

4. 对于String类型

---> String s = "abc";

---> String s1 = new String("abc");

---> s == s1 的结果是false，因为s存放在方法区(常量池), s1 存放在堆空间中，且每次new都是创建一个对象

构造方法

1. 参数为空的构造方法 ---> public ArrayList(){ …… } // 默认初始容量为10；

使用：List arrayList = new ArrayList();

### 什么是Actor类？

Java Vector 类

Java 数据结构

Vector 类实现了一个动态数组。和 ArrayList 很相似，但是两者是不同的：

Vector 是同步访问的。

Vector 包含了许多传统的方法，这些方法不属于集合框架。

Vector 主要用在事先不知道数组的大小，或者只是需要一个可以改变大小的数组的情况。

Vector 类支持 4 种构造方法。

第一种构造方法创建一个默认的向量，默认大小为 10：

Vector()

第二种构造方法创建指定大小的向量。

Vector(int size)

第三种构造方法创建指定大小的向量，并且增量用 incr 指定。增量表示向量每次增加的元素数目。

Vector(int size,int incr)

第四种构造方法创建一个包含集合 c 元素的向量：

Vector(Collection c)

### 问：Vector的初始容量？ArrayList的初始容量？

Vector 类

初始容量(默认容量)

Vector vector = new Vector();

vector.capacity();

当前实际存储容量

Vector vector1 = new Vector();

vector.size();

ArrayList 类

构造器

方式一：

ArrayList<String> arrayList= new ArrayList<>();

方式二

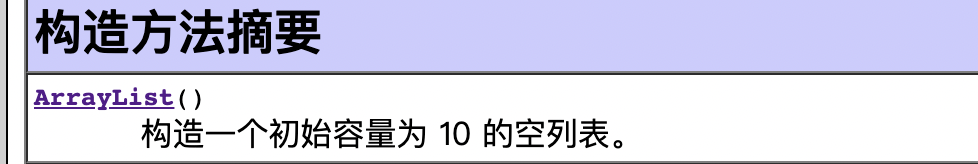
ArrayList arrayList1 = new ArrayList();

当前实际容量

arrayList.size(); // 0

arrayList1.size(); // 0

初始默认容量：



没有提供获得初始容量方法？

如何求？

package package3;/\*\*

\* package3

\* RunDi

\*/

/\*\*

\* @program: RunDi

\* @description:

\* @param:

\* @return:

\* @author zzhd

\*\*/

import java.lang.reflect.Field;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* 问：什么是Field ？

\* 在java反射中，Field类描述的是类的属性信息

\*

\*/

public class FieldTest {

public static void main(String[] args) {

List list = new ArrayList();

// List list1 = new ArrayList<>();

/\*

ArrayList类实现了List接口

\*/

/\*

什么是List类？

List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。

用户能够使用索引（元素在List中的位置，类似于数组下 >标）来访问List中的元素，这类似于Java的数组

什么是Collection？

Collection接口是 (java.util.Collection)是Java集合类的顶级接口之一，

整个集合框架就围绕一组标准接口而设计

\*/

/\*

getClass()方法

1. 返回对象（Object）运行时的类

2. 返回的对象（Class）是被表示类的（synchronized）方法锁定的对象（synchronized是同步方法）

getDeclaredField(name)方法

1. 返回一个Filed对象，该对象反映由class的对象表示的类或接口的参数字段声明的修饰类型

2. 参数name，是一个指定所需字段的简单名称

\*/

try {

Field field = list.getClass().getDeclaredField("elementData");

// elementDate ---> 使用了transient关键字修饰

/\*\*

\* transient关键字解析：

\* 变量修饰符

\* 如果用transient声明一个实例变量，当对象存储时，他的值不需要维持。

\* 也就是说，用transient关键字标记的成员变量不参与序列化的过程。

\*

\* 序列化

\* 是将对象的状态信息转换为可以存储和传输的形式的过程；

\* 在序列化期间，对象将其当前状写入到临时或持久性存储区，

\* 以后通过从存储区中读取或反序列化对象的状态，重新创建该对象。

\*

\* 序列化目的

\* 1. 以某种存储形式使自定义对象持久化

\* 2. 将对象从一个地方存储到另一个地方

\*

\*/

System.out.println(field); // transient java.lang.Object[] java.util.ArrayList.elementData

/\* transient ---> elementData的修饰类型

getClass() ---> java.lang.Object[]

getDeclaredField --> java.util.ArrayList.elementData --> 指elementData存在的位置

\*/

System.out.println(field.isAccessible()); // false

field.setAccessible(true);

/\*\*

\* 禁用Java安全访问检查开关，使得Field 对象名能 对该类中的成员属性进行访问

\* 此操作不管成员的修饰类型是private default abstract public 都能访问

\*/

int size = -1;

try {

System.out.println(field.getName() + " " + field.get(list));

/\*

getName() : elementData

get(list) : [Ljava.lang.Object;@511d50c0

\*/

size = ((Object[])field.get(list)).length;

System.out.println("size = " + size); // 0

}catch(Exception e){

e.printStackTrace();

}

list.add(1);

Field field1 = list.getClass().getDeclaredField("elementData");

field1.setAccessible(true);

int size1 = -1;

try {

size1 = ((Object[]) (field1.get(list))).length;

}catch (Exception e){

e.printStackTrace();

}

System.out.println("size1 = " + size1); // 10

/\*\*

\* 结论：

\* 第一次创建数组的时候如果没有指定大小，则默认{}就是一个空数组，容量为0不是10，只有当第一次添加的时候

\* 才扩容，jdk8 中改为这种懒加载模式，应该有其道理。

\* 但是在阿里开发手册中建议中，创建手册先预定其大小。

\*

\*/

}catch (NoSuchFieldException e){ // 没找到File异常

e.printStackTrace();

}catch (NullPointerException e){ // 空指针异常

e.printStackTrace();

}catch (SecurityException e){ // 安全异常

e.printStackTrace();

}

}

}