**[java笔记：关于复杂数据存储的问题--基础篇：数组以及浅拷贝与深拷贝的问题（上）](http://www.cnblogs.com/sharpxiajun/archive/2011/12/23/2299902.html)**

　　记得我在写javascript笔记时候说过：**程序就是由数据和运算组成**。所以对数据存储以及读取方式的研究是熟练掌握语言精髓的重要途径。我在上篇文章里说道我想重新回顾一些知识，这些知识就是数据存储的问题，而且是复杂数据存储的问题。我个人认为一名优秀的程序员应该有四个主要指标：**一是项目经验，二是程序优化的能力，三是良好的设计理念，四是快速准确定位程序bug的能力**。项目经验不说，这个需要积累，而其他的能力都是可以通过学习而不断强化的。而语言中数据存储能力掌握的优劣是你优化程序的水平的高低的重要指标，**你想让自己的程序越来越快，按什么数据模型快速存储数据，并且能很快的检索被存储的数据才是程序优化的本质**，因此学习数据的存储方式特别是复杂数据的存储方式就十分的重要了。

　　（学习和提高是永无止尽的，我不是技术狂热分子，但我是一个想把事业做精做透的人，更想自己有一天能做到真正意义的创新，所以我要不断的自我激励，时刻准备抓住机会和灵感的那天，因为我相信创新的灵感和机会源自于你不断准备灵感和机会到来的这个过程之中。）

　　呵呵，不说大话了，开始干实事了。不想创新，我只想写程序时候思路开阔，不会只要是数组就写ArrayList，碰到键值对就是HashMap，也许我们会有更好的选择，但你一定要知道更好的选择是什么。

　　java的第一个真理：**在java里除了基本类型就是对象**。java比C好用的一个重要指标：**java里提供了大量对对象操作的集合类，这些集合类大部分都在java.util包里**。

　　但是第一个用来存储复杂数据的数据结构不是List、Set或者是Map，而是数组(Array)。我就先从数组说起。

**如何定义数组**

　　如果有位面试官让你说出这个问题的答案，你能很全面的说出来吗？或许很多童鞋可以，但是我好几次都回答的不全面，大家先看下面的代码：

[复制代码](javascript:void(0);)

package cn.com.sxia;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class ArrayDefine {  
  
 public static void main(String[] args) {  
 //**注释一**：java里提供了两组定义数组的方式，二者是等价的，而int a2[]是C和C++语言的风格。在java里定义一个数组只是表明某个变量获得了数组的引用，但此时数组是没有被分配任何空间的，要让数组获得相应的内存空间就得对数组进行初始化。

int[] a1;  
 int a2[];  
 //**注释二**：假如我们知道我们那些数据要放到数组中，这些数据是确定无疑的，那么注释二的做法是一个简便的 数组初始化方式：**直接使用花括号初始化数组**。

int[] a3 = { 1, 2, 3, 4, 5 };  
 int a4[] = { 6, 7, 8, 9, 0 };  
 //**注释三**：这个实例给了我们另外一种选择：当我们只知道数组的个数而不知道数组内容时候数组该如何定义了？这个时候数组的length属性里存储了数组的长度，length或许是我们在使用数组这个数据结构中最常用的一个属性，适当了用局部变量记下它的值，而不是每次通过数组重新计算length的值是一种提高程序效率的有效方式。

int[] a5 = new int[3];  
 int a6[] = new int[3];  
 //**注释四**：这里我将数据类型换成了对象，用法和基本类型一样。  
 Integer[] b1;  
 Integer b2[];  
 Integer[] b3 = new Integer[3];  
 Integer b4[] = new Integer[3];  
 //**注释五**：用花括号初始化**对象**，大家看到对于Integer类型我们可以直接使用int类型初始化，java会自动把int转化这个Integer对象。

Integer[] b5 = { 1, 2, 3 };  
 Integer[] b6 = { new Integer(4), new Integer(5), new Integer(6) };  
 System.out.println("b5.length:" + b5.length);  
 System.out.println("b6.length:" + b6.length);  
 //**注释六**：这里给出了对象数组定义的另一种做法，效果和注释五下面的代码类似，不过注释六下面的代码会灵活点，例如我们把Integer换成Object，那么这个数组就变成了通用数组了，这个小技巧太小儿科了，这里就不深入了。

Integer[] b7 = new Integer[] { new Integer(4), new Integer(5),  
 new Integer(6) };  
 System.out.println("b7.length:" + b7.length);  
 }  
}

[复制代码](javascript:void(0);)

　　结果如下：

b5.length:3  
b6.length:3  
b7.length:3

　　　　　虽然数组存储基本类型和对象从代码表象上看用法差不多，但是它们在本质上还是有区别的：**对象的数组保存的是引用而基本类型的数组是直接保存基本类型的值**。

　　在java里我们会常常忽视数组，这是一个极其不好的习惯，数组不管在那个编程语言里它有时都会是优秀的复杂数据存储结构，我们把数组和java里的ArrayList类作比较，数组有如下的优势：

1. **效率很高**：数组是java中一种效率最高的存储和随机访问对象引用的方式，数组是使用一个简单的线性结构，这就让线程的访问速度非常的快。另外线程的长度是固定的，这就免去了动态长度所带来的性能开销，这也是数组比较快的重要原因。最后数组存储的数据都是统一类型的，因此使用数组时候就少了对数据类型的校验和转化工作，这样数组的效率相比集合类的那种可以存储任何类型的的特点比较起来，效率又会提升很多。
2. **数组有存储基本类型的能力**：java中的集合类是针对对象的存储设计的，而数组是什么数据类型都可以作为它的存储的内容。

　　不过java的util包还是提供对数组操作的工具类Arrays，该类的方式都是静态使用起来很方便，但是我个人觉得这个类使用价值不大，很多功能我们自己去写可能会更好些。

　　Arrays虽然不讨我喜欢，但是System类里的arraycopy倒是很讨我喜欢，在看大伙看代码之前我们先看看arrayCopy方法在jdk文档里的解释吧：

[复制代码](javascript:void(0);)

arraycopy  
public static void arraycopy(Object src,  
 int srcPos,  
 Object dest,  
 int destPos,  
 int length)从指定源数组中复制一个数组，复制从指定的位置开始，到目标数组的指定位置结束。从 src 引用的源数组到 dest 引用的目标数组，数组组件的一个子序列被复制下来。被复制的组件的编号等于 length 参数。源数组中位置在 srcPos 到 srcPos+length-1 之间的组件被分别复制到目标数组中的 destPos 到 destPos+length-1 位置。   
如果参数 src 和 dest 引用相同的数组对象，则复制的执行过程就好像首先将 srcPos 到 srcPos+length-1 位置的组件复制到一个带有 length 组件的临时数组，然后再将此临时数组的内容复制到目标数组的 destPos 到 destPos+length-1 位置一样。   
  
If 如果 dest 为 null，则抛出 NullPointerException 异常。   
  
如果 src 为 null, 则抛出 NullPointerException 异常，并且不会修改目标数组。   
  
否则，只要下列任何情况为真，则抛出 ArrayStoreException 异常并且不会修改目标数组：   
  
src 参数指的是非数组对象。   
dest 参数指的是非数组对象。   
src 参数和 dest 参数指的是那些其组件类型为不同基本类型的数组。   
src 参数指的是具有基本组件类型的数组且 dest 参数指的是具有引用组件类型的数组。   
src 参数指的是具有引用组件类型的数组且 dest 参数指的是具有基本组件类型的数组。   
否则，只要下列任何情况为真，则抛出 IndexOutOfBoundsException 异常，并且不会修改目标数组：   
  
srcPos 参数为负。   
destPos 参数为负。   
length 参数为负。   
srcPos+length 大于 src.length，即源数组的长度。   
destPos+length 大于 dest.length，即目标数组的长度。   
否则，如果源数组中 srcPos 到 srcPos+length-1 位置上的实际组件通过分配转换并不能转换成目标数组的组件类型，则抛出 ArrayStoreException 异常。在这种情况下，将 k 设置为比长度小的最小非负整数，这样就无法将 src[srcPos+k] 转换为目标数组的组件类型；当抛出异常时，从 srcPos 到 srcPos+k-1 位置上的源数组组件已经被复制到目标数组中的 destPos 到 destPos+k-1 位置，而目标数组中的其他位置不会被修改。（因为已经详细说明过的那些限制，只能将此段落有效地应用于两个数组都有引用类型的组件类型的情况。）   
  
  
参数：  
src - 源数组。  
srcPos - 源数组中的起始位置。  
dest - 目标数组。  
destPos - 目标数据中的起始位置。  
length - 要复制的数组元素的数量。   
抛出：   
IndexOutOfBoundsException - 如果复制会导致对数组范围以外的数据的访问。   
ArrayStoreException - 如果因为类型不匹配而使得无法将 src 数组中的元素存储到 dest 数组中。   
NullPointerException - 如果 src 或 dest 为 null。

[复制代码](javascript:void(0);)

　　我们的代码如下：

[复制代码](javascript:void(0);)

package cn.com.sxia;  
  
import java.util.Arrays;  
  
public class ArraysCopy {  
  
   
 public static void main(String[] args) {  
 /\*基本数据类型\*/  
 System.out.println("========================基本数据类型==========================");  
 int arr1[] = new int[5];  
 int arr2[] = new int[9];  
   
 //Arrays工具类的部分方法使用  
 Arrays.fill(arr1, 7);  
 System.out.println(Arrays.toString(arr1) + "arr1的长度是:" + arr1.length);  
 Arrays.fill(arr2, 9);  
 System.out.println(Arrays.toString(arr2) + "arr2的长度是:" + arr2.length);  
   
 //数组拷贝  
 System.arraycopy(arr1, 0, arr2, 0, arr1.length);  
 System.out.println("新的数组:" + Arrays.toString(arr2) + "arr2的长度是:" + arr2.length);  
   
 /\*对象的操作\*/  
 System.out.println("========================对象的操作==========================");  
 Integer[] arrObj1 = new Integer[5];  
 Integer[] arrObj2 = new Integer[9];  
 Arrays.fill(arrObj1, new Integer(7));  
 System.out.println(Arrays.toString(arrObj1) + "arrObj1的长度是:" + arrObj1.length);  
 Arrays.fill(arrObj2, new Integer(9));  
 System.out.println(Arrays.toString(arrObj2) + "arrObj2的长度是:" + arrObj2.length);  
   
 //数组拷贝  
 System.arraycopy(arrObj1, 0, arrObj2, 0, arrObj1.length);  
 System.out.println("新的数组:" + Arrays.toString(arrObj2) + "arrObj2的长度是:" + arrObj2.length);  
 }  
  
}

[复制代码](javascript:void(0);)

　　运行结果如下：

[复制代码](javascript:void(0);)

========================基本数据类型==========================  
[7, 7, 7, 7, 7]arr1的长度是:5  
[9, 9, 9, 9, 9, 9, 9, 9, 9]arr2的长度是:9  
新的数组:[7, 7, 7, 7, 7, 9, 9, 9, 9]arr2的长度是:9  
========================对象的操作==========================  
[7, 7, 7, 7, 7]arrObj1的长度是:5  
[9, 9, 9, 9, 9, 9, 9, 9, 9]arrObj2的长度是:9  
新的数组:[7, 7, 7, 7, 7, 9, 9, 9, 9]arrObj2的长度是:9

[复制代码](javascript:void(0);)

　　代码里我顺便演示了Arrays类的部分功能，使用数组经常因为数组大小一开始就固定好的缺点，我们不得不去重新拷贝数组，System.arraycopy方法提供了一个十分简便的方案，不过这个方法也是有问题的，从代码里我们可以认为System的arrayCopy方法什么样的数组都可以拷贝，但是对象数组的拷贝就有点不同，对象拷贝复制的是对象的引用而非对象的本身，这种拷贝叫做浅拷贝(shallow copy)。

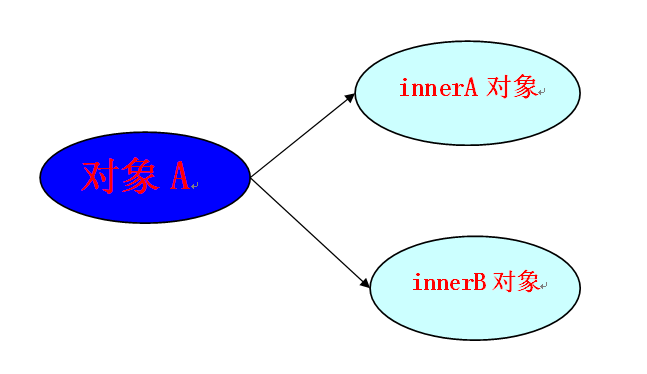
　　方法的问题就是这个浅拷贝所造成，下面我就谈谈浅拷贝以及它对应的深拷贝。

　　浅拷贝和深拷贝的定义如下：

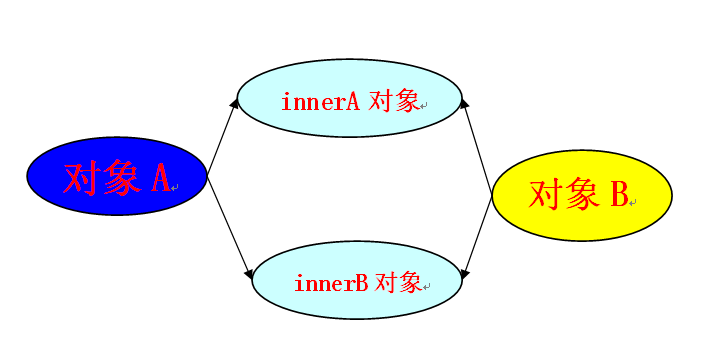
**浅拷贝**：比如A对象被复制到B对象，但是B对象只是复制了A对象本身，如果A对象里还有存在指向其他对象数组或者是引用，B对象内部不会复制这些内部的信息而是指向原来A对象引用的信息。

**深拷贝**：还是列举A对象被复制到B对象的例子，B对象拷贝到的是A对象的所有信息，包括A对象内部的对象引用。

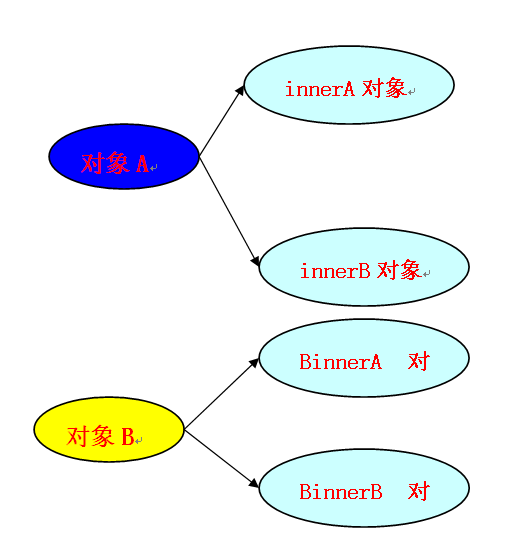
　　可能看到上面的解释很多人还是不太清晰浅拷贝和深拷贝的区别，我下面用图形来展示它们的区别，首先是A对象，A里包含了两个内部对象innerA和innerB，如下图：



　　如果把A对象拷贝到B对象，但是B对象内部并没有拷贝innerA对象和innerB对象，B对象内部还是指向原来的innerA对象和innerB对象，如下图：



　　如果把A对象深拷贝到B对象，那么结果如下图：



　　A的内部对象也会拷贝到B对象内部。

　　上面的图形应该可以清晰的表达出浅拷贝和深拷贝的区别了吧。为什么会有浅拷贝和深拷贝了，到底是什么原因产生了浅拷贝和深拷贝了？昨天一个朋友用C语言的方式给我做了解答，哎，可惜C语言我只是在大学里学过，现在忘记的差不多了，但是我哪位朋友指出了**浅拷贝和深拷贝是因为指针所产生的，产生的条件就是值的传递和返回，本质是数据在内存中存放的方式所造成**。

　　下面我就要探求深浅拷贝的本质了，首先我又要说一个java语言里的真理：**Java中所有参数传递都是按引用传递**。

 　　引用，引用还是引用，我前面讲了太多引用了啊。

　　今天写累了，我下一篇博文就从这个万恶的引用讲起。

　　重新研究编程语言是件很开心的事情，特别是在你已经做过一些项目以后你再看使用的语言的语法知识，你会有种豁然开朗的感觉：你知道那些知识很有用，那些知识比较难，更重要的是你知道你重新复习这些技术你今后能把他们用到什么样的地方，或许有人说这些都是基础，我们都做不少的项目了，何必再浪费时间，但你如果是对你所做的技术有更高的要求，你想进步而不是应付工作，学精一个东西一定会让你的成长不会很快碰到瓶颈了，不断学习和工作的人总会比懈怠的人收获的更多，这就是我的想法。