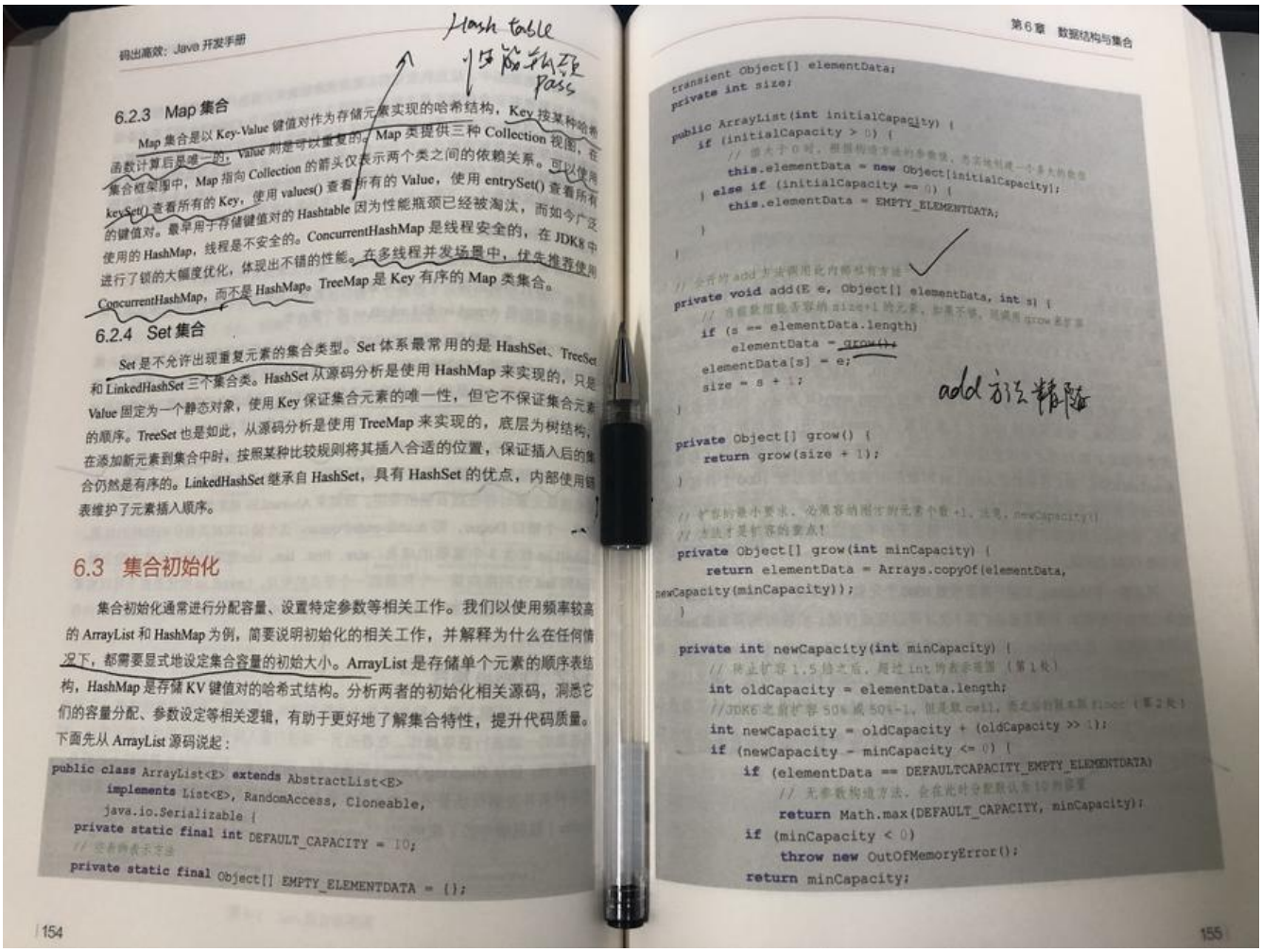
《码出高效》系列笔记（四）：数据结构与集合的数组和泛型

作者：matthewhan

原文链接：https://ld246.com/article/1584600562580

来源网站：链滴



良好的编码风格和完善统一的规约是最高效的方式。

前言

本篇汲取了本书中较为精华的知识要点和实践经验加上读者整理，作为本系列里的第四篇章第二节：据结构与集合的数组和泛型篇。

本系列目录:

数组与集合

数组是一种顺序表，可以使用索引下标进行快速定位并获取指定位置的元素。

为什么下标从 0 开始?

因为这样需要计算偏移量需要从当前下标减 1 的操作，加减法运算对 CPU 是一种双数运算，在数组标使用频率很高的场景下，该运算方式十分耗时。在 Java 的体系中，数组一旦分配内存后无法扩容。

```
String[] args1 = {"a", "b"};
String[] args2 = new String[2];
args2[0] = "a";
args2[1] = "b";
```

以上代码一般是数组的两种初始化方式，第一种是静态初始化，第二种是动态初始化。数组的容量大随着数组对象的创建就固定了。

数组的遍历优先推荐 JDK5 引进的 foreach 方式，即 for(e : obj); JDK8 以上可以使用 stream 操作

```
Arrays.stream(args1).forEach(str -> System.out.println(str));
Arrays.stream(args1).forEach(System.out::println);
```

数组转集合

将数组转集合后，不能使用集合的某些方法，以 Arrays.asList() 为例，不能使用其修改集合 **add**、**remove**、**clear** 方法，但是可以使用 **set** 方法。

```
String[] args1 = {"a", "b"};
List<String> asList = Arrays.asList(args1);
asList.set(1, "c");
System.out.println(asList);
asList.add("c");
asList.remove(0);
asList.clear();
System.out.println(asList);
```

后面会输出 UnsupportedOperationException 异常。

Arrays.asList() 体现的是适配器模式，其实是 Arrays 的一个名为 ArrayList 的内部类（阉割版），继自 AbstractList 类，实现了 **set** 和 **get** 方法。但是其他部分方法未实现所以会抛出该父类 AbstractList 的异常。

```
String[] args1 = {"a", "b"};
List<String> e1 = Arrays.asList(args1);
List<String> e2 = new ArrayList<>(2);
e2.add("a");
e2.add("b");
// 第一处
System.out.println(e1.getClass().getName());
// 第二处
System.out.println(e2.getClass().getName());
```

实际控制台打印情况：

1. java.util.Arrays$ArrayList
2. java.util.ArrayList

数组转集合在需要添加元素的情况下，利用 java.util.ArrayList 创建一个新集合。

```
String[] args = {"a", "b"};
List<String> list = new ArrayList<>(Arrays.asList(args));
```

集合转数组

集合转数组更加的可控。

```
List<String> e1 = new ArrayList<>(2);
e1.add("c");
e1.add("d");
String[] args = new String[1];
String[] args2 = new String[2];
e1.toArray(args);
// 第一处
System.out.println(Arrays.asList(args));
e1.toArray(args2);
// 第二处
System.out.println(Arrays.asList(args2));
```

实际控制台打印情况：

1. [null]
2. [c, d]

不同的区别在于即将复制进去的数组容量是否足够，如果容量不等，则弃用该数组，另起炉灶。

集合与泛型

泛型与集合的联合使用，可以把泛型的功能发挥到极致。

```
List list1 = new ArrayList(3);
list1.add(new Integer(666));
list1.add(new Object());
list1.add("666");

List<Object> list2 = new ArrayList<>(3);
list2.add(new Integer(666));
list2.add(new Object());
list2.add("666");

List<Integer> list3 = new ArrayList<>(3);
list3.add(new Integer(666));
// 以下都是编译出错
list3.add(new Object());
list3.add("666");

List<?> list4 = new ArrayList<>(3);
```

```
list4.remove(0);
list4.clear();
// 以下都是编译出错
list4.add(new Integer(666));
list4.add(new Object());
list4.add("666");
```

List<?> 是一个泛型，在没有赋值之前，表示它可以接收任何类型的集合赋值，赋值之后就不能随便里面添加元素了，但可以 remove 和 clear。

而 List<T> 最大的问题就是只能放置一种类型，如果要实现多种受泛型约束的类型，可以使用 <? extends T> 与 <? super T> 两种语法，但是两者的区别非常微妙。

- <? extends T> 是 Get First，适用于消费集合元素为主的场景；
- <? super T> 是 Put First，适用于生产集合元素为主的场景。

<? extends T> 可以赋值给任何**T 以及 T 子类**的集合，上界为 T。取出的类型带有泛型限制，向上转型为 T。null 可以表示任何类型，所以除了 null 外，任何元素都不得添进 <? extends T> 集合内。

<? super T> 可以赋值给任何**T 以及 T 的父类**集合，下界为 T。在生活，投票选举类似 <? super T> 的操作。选举代表时，你只能往里投票，取数据时，根本不知道是谁票，相遇泛型丢失。

extends 的场景是 put 功能受限，而 super 的场景是 get 功能受限。

extends 与 super 的差异

假设有一个斗鱼 TV 平台，拥有一个 DOTA2 板块，其下有一个恶心人的 D 能儿主播：谢彬 DD。

那我们从代码里可以这样写：

```
@Test
public void main() {

    List<DouYu> douYu = new ArrayList<>();
    List<DotA2> dotA2 = new ArrayList<>();
    List<DD> dd = new ArrayList<>();
    douYu.add(new DouYu());
    dotA2.add(new DotA2());
    dd.add(new DD());

    // 第一处，编译出错
    List<? extends DotA2> extendsDotA2FromDouYu = douYu;
    List<? super DotA2> superDotA2FromDouYu = douYu;

    List<? extends DotA2> extendsDotA2FromDotA2 = dotA2;
    List<? super DotA2> superDotA2FromDotA2 = dotA2;

    List<? extends DotA2> extendsDotA2FromDD = dd;
    // 第二处，编译出错
    List<? super DotA2> superDotA2FromDD = dd;
```

```
}
```

三个类的继承关系说明 DD < DotA2 < DouYu < Object。

第一处编译出错，因为只能赋值给 T 以及 T 的子类，上界是 DotA2 类。DouYu 类明显不符合 extens DotA2 类的情况。不能把 **douYu 对象**赋值给 <? extends DotA2>，因为 List<DouYu> 不只只有 DotA2 板块，还有吃♂鸡、颜♂值区、舞♂蹈区这些板块。

第二处编译出错，因为只能赋值给 T 以及 T 的父类，DD 类属于 DotA2 的子类，下界只能 DotA2 的对象。

```
// 以下<? extends DotA2>类型的对象无法进行add操作，编译出错
extendsDotA2FromDotA2.add(new DD());
extendsDotA2FromDotA2.add(new DotA2());
extendsDotA2FromDotA2.add(new DouYu());

superDotA2FromDotA2.add(new DD());
superDotA2FromDotA2.add(new DotA2());
// 该处编译出错，无法添加
superDotA2FromDotA2.add(new DouYu());
```

除了 **null** 以外，任何元素都不能添加进 <? extends T> 集合内。<? super T> 可以放，但是只能放去**自身以及子类**。

```
Object obj1 = extendsDotA2FromDotA2.get(0);
DotA2 obj2 = extendsDotA2FromDotA2.get(0);

Object obj3 = extendsDotA2FromDD.get(0);
// 该处编译出错，无法添加
DD obj4 = extendsDotA2FromDD.get(0);
```

首先 <? super T> 可以进行 **Get** 操作返回元素，但是类型会丢失。<? extends T> 可以返回带类型元素，仅限自身及父类，子类会被擦除。

小总结

对于一个笼子，只取不放，属于 Get First，应采用 <? extends T>；只放不取，属于 Put First，应用 <? super T>。