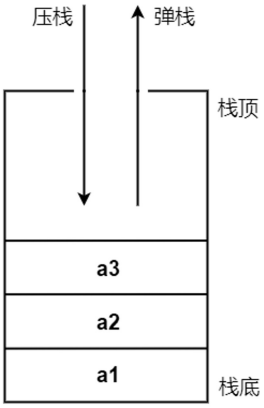
day03_List、Set、数据结构、Collections

一.数据结构

1.栈(Stack)

栈是一种运算受限的线性表，其限制是仅允许在线性表的一端进行插入和删除操作，不允许在其他任何位置进行添加、查找、删除等操作。

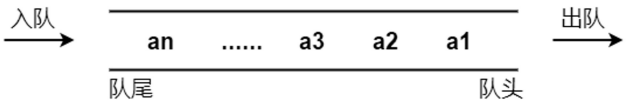


特点：先进后出，只能从栈的一端对元素进行操作

元素 a1、a2、a3 按照顺序从栈顶压栈进入，上面的元素必须先进栈，下面的元素才能出栈。

2.队列(Queue)

队列也是一种运算受限的线性表，其限制是仅允许在线性表的一端进行插入、仅允许在线性表的另一端进行删除。



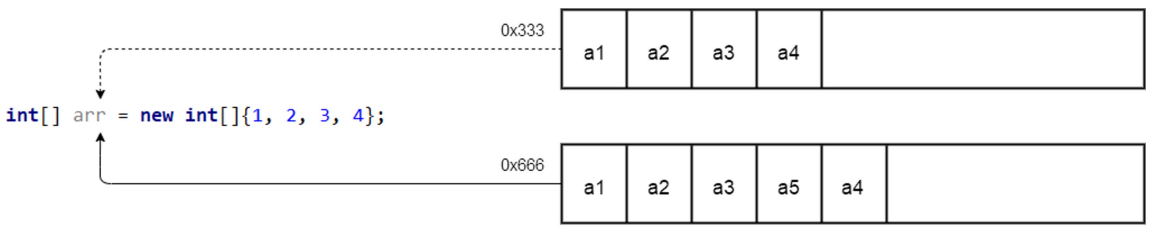
特点：先进先出，只能从队列的一端取出元素，另一端插入元素

元素 a1、a2、a3 按顺序入队，前面的元素必须先入队，后面的元素才能出队

3.数组(Array)

数组是有序的元素序列，是在内存中开辟一段连续的空间用来存放数组元素。数组具有以下特点：

- 查询快：由于数组的地址是连续的，根据数组的首地址、索引，可以快速访问到指定位置的元素。
- 增删慢：由于数组的长度是固定的，增加、删除一个元素，必须先创建一个新的数组，再把源数组的数据复制过来进行增加、删除的操作。

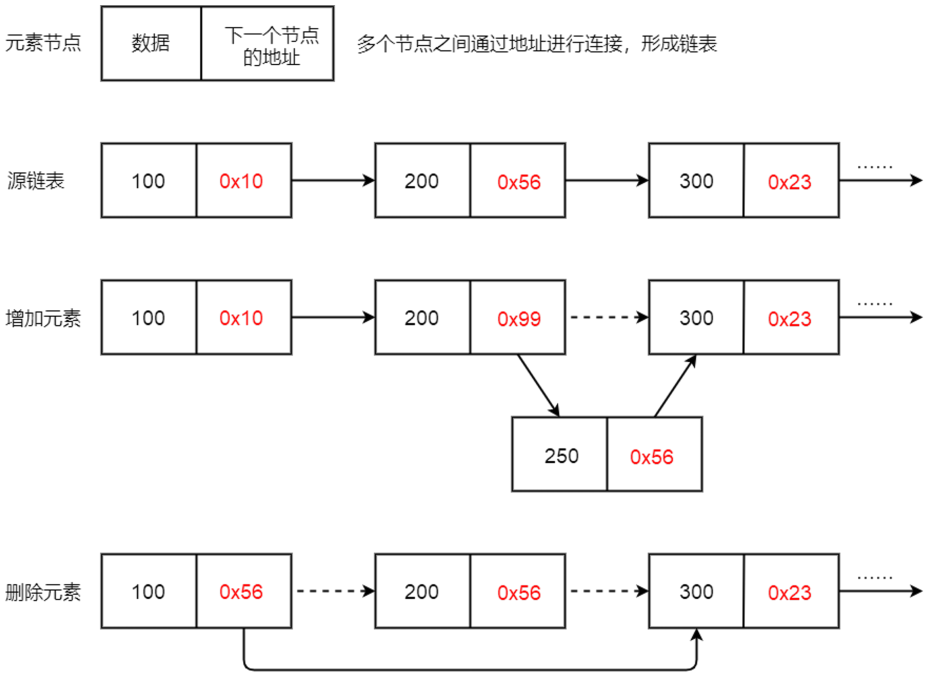


要在索引为3处增加一个元素 a5，必须在堆内存中创建一个新的数组，先把源数组的数据复制过去，再增加新的数据 a5到索引为3处。此时源数组会被销毁(垃圾回收)，将新数组的地址赋值给引用变量。在内存中频繁地创建、销毁数组，效率低下。

4.链表(Linked List)

链表由一系列节点 node (链表中每一个元素称为一个节点)组成，通过一系列的节点将不连续的内存联系起来，将那种碎片内存进行合理的利用。每个元素节点包括两个部分：存储数据元素的数据域、存储下一个节点地址的指针域。链表具有以下特点：

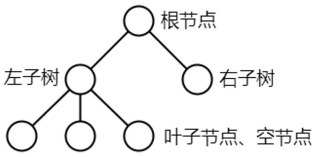
- 查询慢：由于链表中的地址不是连续的，每次查询元素，都要从头开始查询。
- 增删快：在任意位置增加、删除一个元素，对链表的整体结构没有影响。



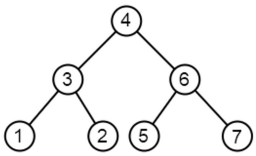
增加、删除元素只需要修改连接下个元素的地址即可

5.红黑树

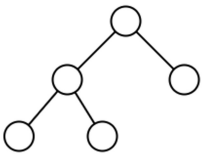
(1).计算机中的树形存储结构：使用每个节点来存放数据



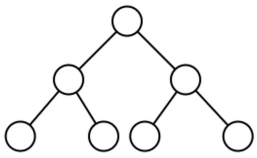
(3).排序树、查找树：在二叉树的基础上，每个节点的左子树数据都小于右子树



(2).二叉树：每个节点的分支不能超过两个



(4).平衡树：每个节点的左子树、右子树个数相等



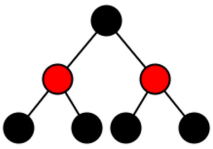
红黑树：一种趋近于平衡树的二叉查找树，查询每个叶子节点的最大次数不会超过最小次数的2倍

约束：

a.节点可以是红色或者黑色，根节点、叶子节点必须是黑色

b.每个红色节点的子节点都是黑色的

c.任一节点到其每一个叶子节点的所有路径上的黑色节点数相同



二.List 集合

java.util.List 接口继承自 java.util.Collection 接口，是单列集合的一个重要分支，一般将实现了 List 接口的对象称为 List系集合。List系集合具有以下特点：

- 是一个元素存取有序的集合
- 集合中可以有重复的元素
- 是一个带有索引的集合，通过索引就可以精确的操作集合中的元素

List 作为 Collection 集合的子接口，不但继承了 Collection 接口中的抽象方法，而且还具有一些根据元素索引来操作集合的特有抽象方法。List 接口三个常见的实现类 ArrayList、LinkedList、Vector 会实现 List 接口中的所有抽象方法。(包括从Collection继承过来的抽象方法，见代码 ListUniqueMethod)

1.ArrayList 集合

java.util.ArrayList 集合底层的存储结构是数组结构。元素增删慢、查找快。由于日常开发中多为查询数据、遍历数据，所以 ArrayList 是常用的集合。(ArrayList 类的实现不是同步的，意味着是多线程)

2.LinkedList 集合

java.util.LinkedList 集合底层的存储结构是链表结构。元素增删快、查找慢。LinkedList 类作为 List 接口的实现类，不仅可以使用 List 接口中的方法，自己还拥有一些操作链表首尾元素的特有方法。

3.Vector 集合(了解)

java.util.Vector 集合类是从JDK 1.0开始就有的集合类，但是从JDK 1.2开始就被改进的 ArrayList 集合所替代，所以 Vector 集合与 ArrayList 集合基本一样，区别在于 Vector 集合是线程安全的集合，意味着是单线程的。

三.Set 集合

java.util.Set 接口继承自 java.util.Collection 接口，也是单列集合的一个重要分支。一般将实现了 Set 接口的对象称为 Set系集合。Set系集合具有以下特点：

- 是一个元素存取无序的集合
- 集合中不可以有重复的元素
- 是一个没有索引的集合(不能使用for循环遍历，只能使用迭代器、增强for遍历)

Set 作为 Collection 集合的子接口，它与 Collection 接口中的方法基本一致，并没有对 Collection 接口进行功能上的扩充。Set 集合三个常见的实现类 TreeSet、HashSet、LinkedHashSet 会实现 Set 接口中的所有抽象方法。

1.HashSet 集合

java.util.HashSet 集合的底层其实是一个 java.util.HashMap 支持，即底层存储结构是哈希表。HashSet 类作为 Set 接口的实现类，可以使用 Set 接口中的所有抽象方法。

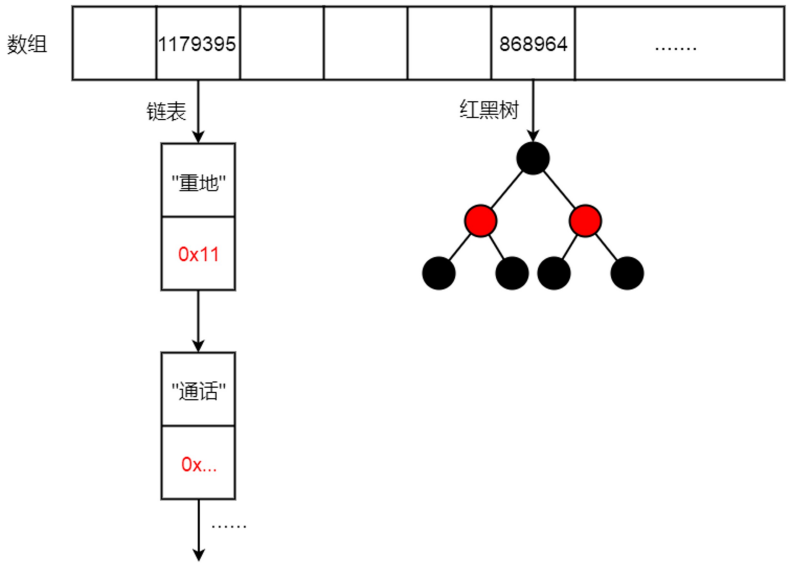
(1).HashSet 集合的用法

(2).HashSet 集合的存储结构——哈希值、哈希表

- 哈希值：是模拟出来的对象的逻辑地址值，不是对象实际存储的物理地址，是一个由系统随机给出十进制整数。使用 Object 类中的 hashCode方法，可以获取对象的哈希值。

```
public class Person /*extends Object*/ {
    @Override
    public int hashCode() {
        return 1;
    }
}

public static void main(String[] args) {

    // (1).获取对象的哈希值：调用hashCode()方法获取
    Person p1 = new Person( name: "张三",  age: 18);
    System.out.println(p1.hashCode()); // 3812
    Person p2 = new Person( name: "张三",  age: 18);
    System.out.println(p2.hashCode()); // 21297  属性完全相同的对象哈希值不一样
    /*
    打印对象名时调用了toString方法，toString方法的源码：
    return getClass().getName() + "@" + Integer.toHexString(hashcode());
    所以返回的地址值就是：对象的哈希值的十六进制
     */
    System.out.println(p1); // SetSet.Person@16b98e56

    // (2).Person类重写Object类的hashCode()方法
    System.out.println(p1.hashCode()); // 1
    System.out.println(p1);            // SetSet.Person@1

    // 有两个字符串的哈希值刚好一样
    System.out.println("重地".hashCode()); // 1179395
    System.out.println("通话".hashCode()); // 1179395
}
```

- 哈希表：哈希表是一种数据结构，底层采用 "数组+链表+红黑树" 实现，极大地提高了查询速度。



数组结构：用来存储元素的哈希值，把相同哈希值的元素分为一组

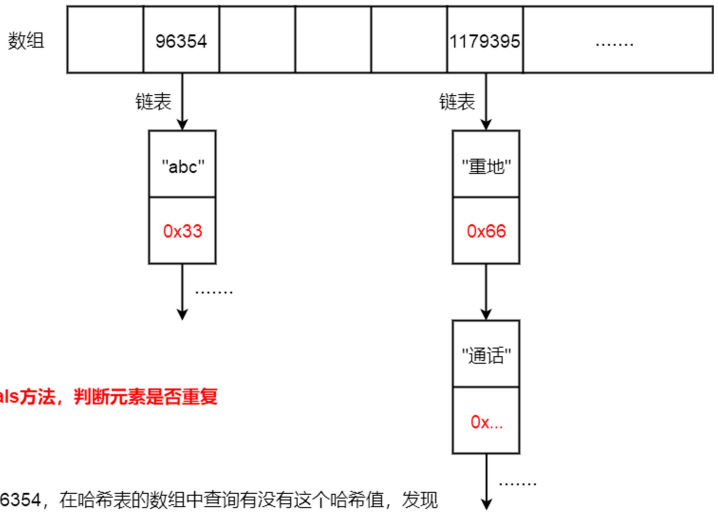
链表结构：用来存储元素数据，为了解决哈希冲突，把相同哈希值的元素连接在一起

红黑树结构：如果链表的长度超过了8位，为了提高查询速度，就会把链表转换为红黑树

利用哈希表存储数据时，先计算元素的哈希值，根据哈希值找到在数组中的对应位置，再利用链表、红黑树在对应的数组索引处进行数据的存储。所以哈希表就是利用数组对元素进行分组，再利用链表、红黑树来存储元素数据。

(3).HashSet 集合不存储重复元素的原理

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

    HashSet<String> hashSet = new HashSet<>();
    String s1 = new String( original: "abc");
    String s2 = new String( original: "abc");
    hashSet.add(s1);
    hashSet.add(s2);
    hashSet.add("重地");
    hashSet.add("通话");
    System.out.println(hashSet); // [重地, 通话, abc]

}
```



HashSet 集合在调用add方法时，会调用元素的hashCode、equals方法，判断元素是否重复

hashSet.add(s1);
add方法会调用s1的hashCode方法，计算字符串"abc"的哈希值是96354，在哈希表的数组中查询有没有这个哈希值，发现没有就会把哈希值96354存入数组，并把"abc"存入该数组索引下的链表

hashSet.add(s2);
add方法会调用s2的hashCode方法，计算字符串"abc"的哈希值是96354，在哈希表的数组中查询有没有这个哈希值，发现有(哈希冲突)，s2会调用equals方法和哈希值相同的元素进行比较，s2.equals(s1)返回true，两个元素相同，不会把s2存入链表中

hashSet.add("重地");
add方法会调用"重地"的hashCode方法，计算字符串"重地"的哈希值是1179395，在哈希表的数组中查询有没有这个哈希值，发现没有就会把哈希值1179395存入数组，并把"重地"存入该数组索引下的链表

hashSet.add("通话");
add方法会调用"通话"的hashCode方法，计算字符串"通话"的哈希值是1179395，在哈希表的数组中查询有没有这个哈希值，发现有(哈希冲突)，"通话"会调用equals方法和哈希值相同的元素进行比较，"通话".equals("重地")返回false，两个元素不同，就会把"通话"存入链表中

(4).HashSet 集合存储自定义类型元素

HashSet 集合存储自定义类型的元素时，为了保证集合中的对象不重复(属性完全相同的对象我们认为是重复的对象)，自定义的数据类型必须重写 hashCode、equals 方法。(String、Integer类已经重写了hashCode、equals方法)

- 重写 hashCode 方法的原因：由于属性完全相同的对象的哈希值是不一样的，所以要重写 hashCode 方法保证其哈希值一样。
- 重写 equals 方法的原因：见 "重写 Object 类的 equals 方法 "

2.LinkedHashSet 集合

java.util.LinkedHashSet 集合继承了 java.util.HashSet 集合类，所以与 HashSet 集合的原理与使用基本一样。但是 LinkedHashSet 集合的底层存储结构是 "哈希表+链表" ，多出来的链表用来记录元素的存储顺序，因此 LinkedHashSet 集合是存取有序的集合，这一点有别于 HashSet 集合。

3.可变参数

四.Collections 类