第8天 集合

今日学习内容

* List集合
* Set集合

今日学习目标

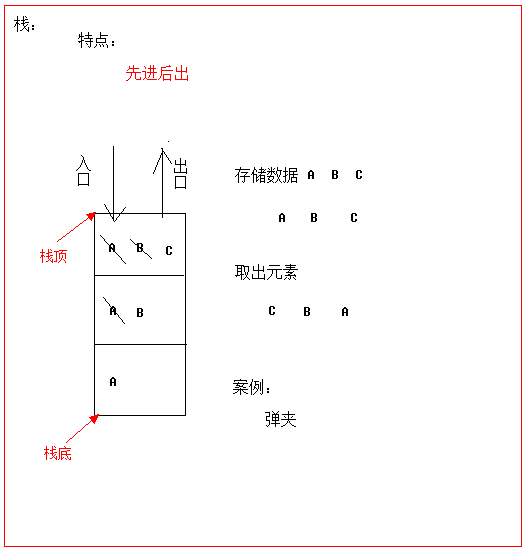
* 能够说出List集合特点:有序,索引和重复
* 使用List存储的数据结构:ArrayList是数组结构 LinkedList是链表结构
* 能够说出List常见的三个的特点
* 能够说出Set集合的特点:无索引,不重复
* 说出哈希表的特点
* 使用HashSet集合存储自定义元素
* 说出判断集合元素唯一的原理

# 数据结构

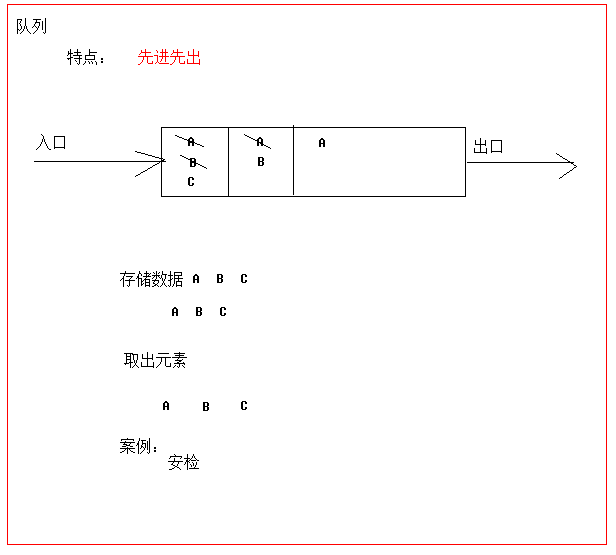
## List集合存储数据的结构

List接口下有很多个集合，它们存储元素所采用的结构方式是不同的，这样就导致了这些集合有它们各自的特点，供给我们在不同的环境下进行使用。数据存储的常用结构有：堆栈、队列、数组、链表。我们分别来了解一下：

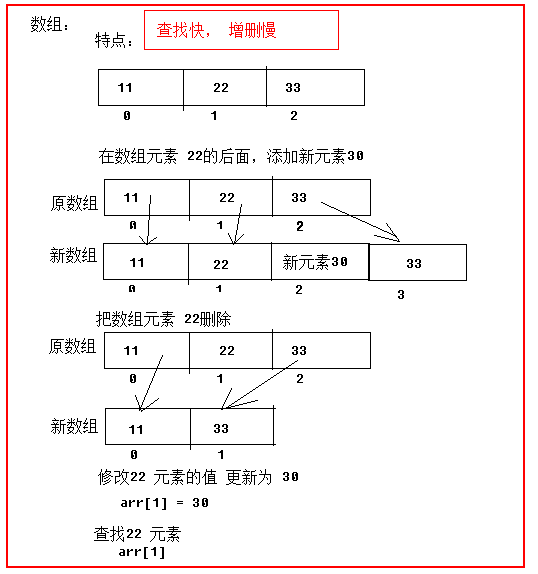
* 堆栈，采用该结构的集合，对元素的存取有如下的特点：
  + 先进后出（即，存进去的元素，要在后它后面的元素依次取出后，才能取出该元素）。例如，子弹压进弹夹，先压进去的子弹在下面，后压进去的子弹在上面，当开枪时，先弹出上面的子弹，然后才能弹出下面的子弹。
  + 栈的入口、出口的都是栈的顶端位置
  + 压栈：就是存元素。即，把元素存储到栈的顶端位置，栈中已有元素依次向栈底方向移动一个位置。
  + 弹栈：就是取元素。即，把栈的顶端位置元素取出，栈中已有元素依次向栈顶方向移动一个位置。



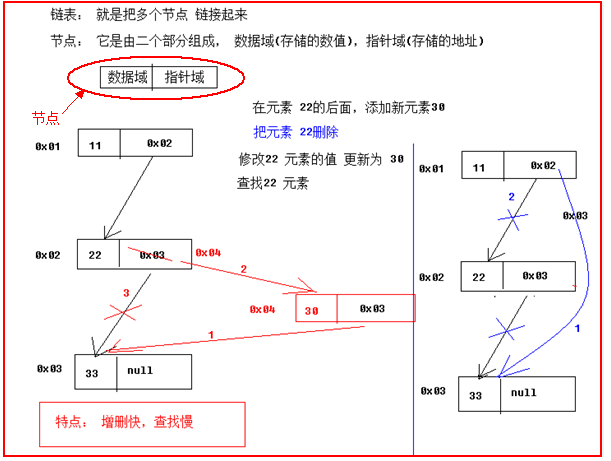
* 队列，采用该结构的集合，对元素的存取有如下的特点：
  + 先进先出（即，存进去的元素，要在后它前面的元素依次取出后，才能取出该元素）。例如，安检。排成一列，每个人依次检查，只有前面的人全部检查完毕后，才能排到当前的人进行检查。
  + 队列的入口、出口各占一侧。例如，下图中的左侧为入口，右侧为出口。



* 数组，采用该结构的集合，对元素的存取有如下的特点：
  + 查找元素快：通过索引，可以快速访问指定位置的元素
  + 增删元素慢：
    - **指定索引位置增加元素**：需要创建一个新数组，将指定新元素存储在指定索引位置，再把原数组元素根据索引，复制到新数组对应索引的位置。如下图
    - **指定索引位置删除元素：**需要创建一个新数组，把原数组元素根据索引，复制到新数组对应索引的位置，原数组中指定索引位置元素不复制到新数组中。如下图



* 链表，采用该结构的集合，对元素的存取有如下的特点：
  + 多个节点之间，通过地址进行连接。例如，多个人手拉手，每个人使用自己的右手拉住下个人的左手，依次类推，这样多个人就连在一起了。
  + 查找元素慢：想查找某个元素，需要通过连接的节点，依次向后查找指定元素
  + 增删元素快：
    - 增加元素：操作如左图，只需要修改连接下个元素的地址即可。
    - 删除元素：操作如右图，只需要修改连接下个元素的地址即可。



## LinkedList集合(有空可以先看一下List这个接口和方法,以及List实现类的各种数据结构特点)

LinkedList集合数据存储的结构是链表结构。方便元素添加、删除的集合。实际开发中对一个集合元素的添加与删除经常涉及到首尾操作，而LinkedList提供了大量首尾操作的方法。如下图



LinkedList是List的子类，List中的方法LinkedList都是可以使用，这里就不做详细介绍，我们只需要了解LinkedList的特有方法即可。在开发时，LinkedList集合也可以作为堆栈，队列的结构使用。

方法演示：

LinkedList<String> link = **new** LinkedList<String>();

//添加元素

link.addFirst("abc1");

link.addFirst("abc2");

link.addFirst("abc3");

//获取元素

System.*out*.println(link.getFirst());

System.*out*.println(link.getLast());

//删除元素

System.*out*.println(link.removeFirst());

System.*out*.println(link.removeLast());

**while**(!link.isEmpty()){ //判断集合是否为空

System.*out*.println(link.pop()); //弹出集合中的栈顶元素

}

# Set接口(特点以及方法?)

查阅Set集合的API介绍，通过元素的equals方法，来判断是否为重复元素，它是个不包含重复元素的集合。Set集合取出元素的方式可以采用：迭代器、增强for。

Set集合有多个子类，这里我们介绍其中的HashSet、LinkedHashSet这两个集合。

## HashSet集合介绍

查阅HashSet集合的API介绍：此类实现Set接口，由哈希表支持（实际上是一个 HashMap集合）。HashSet集合不能保证的迭代顺序与元素存储顺序相同。

HashSet集合，采用哈希表结构存储数据，保证元素唯一性的方式依赖于：hashCode()与equals()方法。

## HashSet集合存储数据的结构（哈希表）

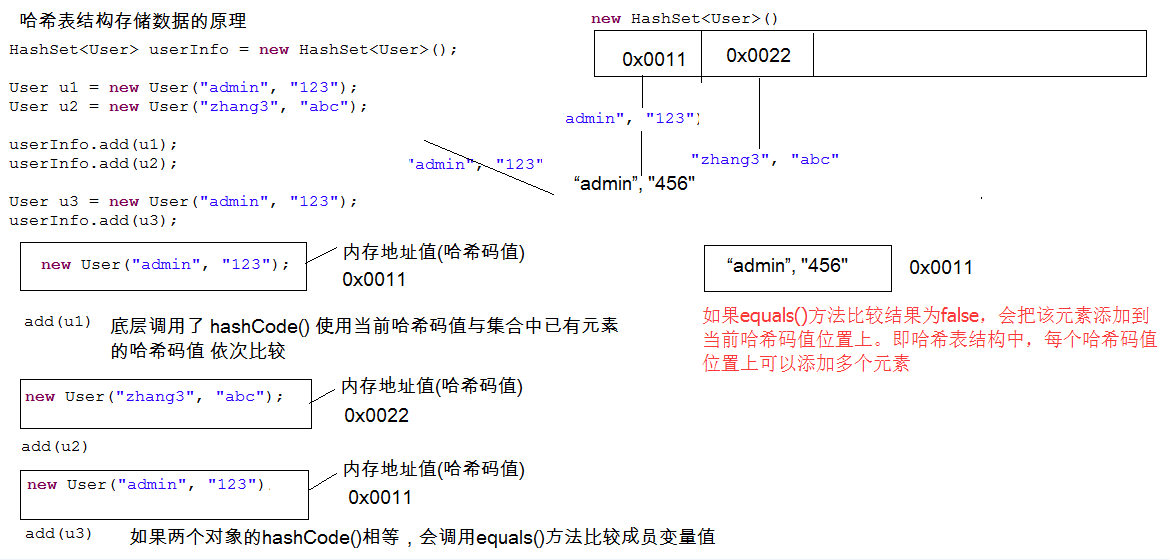
什么是哈希 呢？

哈希表底层使用的也是数组机制，数组中也存放对象，而这些对象往数组中存放时的位置比较特殊，当需要把这些对象给数组中存放时，那么会根据这些对象的特有数据结合相应的算法，计算出这个对象在数组中的位置，然后把这个对象存放在数组中。而这样的数组就称为哈希数组，即就是哈希表。

当向哈希表中存放元素时，需要根据元素的特有数据结合相应的算法，这个算法其实就是Object类中的hashCode方法。由于任何对象都是Object类的子类，所以任何对象有拥有这个方法。即就是在给哈希表中存放对象时，会调用对象的hashCode方法，算出对象在表中的存放位置，这里需要注意，如果两个对象hashCode方法算出结果一样，这样现象称为哈希冲突，这时会调用对象的equals方法，比较这两个对象是不是同一个对象，如果equals方法返回的是true，那么就不会把第二个对象存放在哈希表中，如果返回的是false，就会把这个值存放在哈希表中。

注意:String类重写了hashcode方法,按照自己的方式计算,”abc”和”acD”哈希值相同

总结：保证HashSet集合元素的唯一，其实就是根据对象的hashCode和equals方法来决定的。如果我们往集合中存放自定义的对象，那么保证其唯一，就必须复写hashCode和equals方法建立属于当前对象的比较方式。



## HashSet存储JavaAPI中的类型元素

给HashSet中存储JavaAPI中提供的类型元素时，不需要重写元素的hashCode和equals方法，因为这两个方法，在JavaAPI的每个类中已经重写完毕，如String类、Integer类等。

* 创建HashSet集合，存储String对象。

**publicclass** HashSetDemo {

**publicstaticvoid** main(String[] args) {

//创建HashSet对象

HashSet<String> hs = **new** HashSet<String>();

//给集合中添加自定义对象

hs.add("zhangsan");

hs.add("lisi");

hs.add("wangwu");

hs.add("zhangsan");

//取出集合中的每个元素

Iterator<String> it = hs.iterator();

**while**(it.hasNext()){

String s = it.next();

System.*out*.println(s);

}

}

}

输出结果如下，说明集合中不能存储重复元素：

wangwu

lisi

zhangsan

## HashSet存储自定义类型元素

给HashSet中存放自定义类型元素时，需要重写对象中的hashCode和equals方法，建立自己的比较方式，才能保证HashSet集合中的对象唯一

* 创建自定义对象Student

**publicclass** Student {

**private** String name;

**privateint**age;

**public** Student(String name, **int** age) {

**super**();

**this**.name = name;

**this**.age = age;

}

**public** String getName() {

**return**name;

}

**publicvoid** setName(String name) {

**this**.name = name;

}

**publicint** getAge() {

**return**age;

}

**publicvoid** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return**"Student [name=" + name + ", age=" + age + "]";

}

@Override

**publicint** hashCode() {

**finalint** prime = 31;

**int** result = 1;

result = prime \* result + age;

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

**publicboolean** equals(Object obj) {

**if** (**this** == obj)

**returntrue**;

**if**(!(obj **instanceof** Student)){

**System.out.println(**"类型错误"**)**;

**returnfalse**;

}

Student other = (Student) obj;

**returnthis**.age == other.age&&**this**.name.equals(other.name);

}

}

* 创建HashSet集合，存储Student对象。

**publicclass** HashSetDemo {

**publicstaticvoid** main(String[] args) {

//创建HashSet对象

HashSet hs = **new** HashSet();

//给集合中添加自定义对象

hs.add(**new** Student("zhangsan",21));

hs.add(**new** Student("lisi",22));

hs.add(**new** Student("wangwu",23));

hs.add(**new** Student("zhangsan",21));

//取出集合中的每个元素

Iterator it = hs.iterator();

**while**(it.hasNext()){

Student s = (Student)it.next();

System.*out*.println(s);

}

}

}

输出结果如下，说明集合中不能存储重复元素：

Student [name=lisi, age=22]

Student [name=zhangsan, age=21]

Student [name=wangwu, age=23]

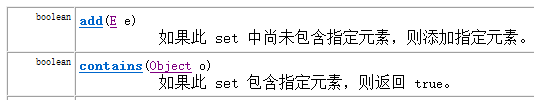
# 判断集合元素唯一的原理

## ArrayList的contains方法判断元素是否重复原理



ArrayList的contains方法调用时,会使用传入的元素的equals方法依次与集合中的旧元素所比较，从而根据返回的布尔值判断是否有重复元素。此时，当ArrayList存放自定义类型时，由于自定义类型在未重写equals方法前，判断是否重复的依据是地址值，所以如果想根据内容判断是否为重复元素，需要重写元素的equals方法。

## HashSet的add/contains等方法判断元素是否重复原理



Set集合不能存放重复元素，其添加方法在添加时会判断是否有重复元素，有重复不添加，没重复则添加。

HashSet集合由于是无序的，其判断唯一的依据是元素类型的hashCode与equals方法的返回结果。规则如下：

先判断新元素与集合内已经有的旧元素的HashCode值

* 如果不同，说明是不同元素，添加到集合。
* 如果相同，再判断equals比较结果。返回true则相同元素；返回false则不同元素，添加到集合。

所以，使用HashSet存储自定义类型，如果没有重写该类的hashCode与equals方法，则判断重复时，使用的是地址值，如果想通过内容比较元素是否相同，需要重写该元素类的hashcode与equals方法。

# Collection总结

## 知识点总结

* Collection:

|- List 可以存储重复元素,有序的(元素存取顺序),有索引

|- ArrayList

|- LinkedList

|- Set 不能存储重复元素,无序的(元素存取顺序,LinkedHashSet是有序的),无索引

|- HashSet

|- LinkedHashSet

* Collection方法：
  + boolean add(Object e) 把给定的对象添加到当前集合中
  + void clear() 清空集合中所有的元素
  + boolean remove(Object o) 把给定的对象在当前集合中删除
  + boolean contains(Object o) 判断当前集合中是否包含给定的对象
  + boolean isEmpty() 判断当前集合是否为空
  + Iterator iterator() 迭代器，用来遍历集合中的元素的
  + int size() 返回集合中元素的个数
  + Object[] toArray() 把集合中的元素，存储到数组中
* Iterator : 迭代器
  + Object next()返回迭代的下一个元素
  + boolean hasNext()如果仍有元素可以迭代，则返回 true。
* List与Set集合的区别？

List:

它是一个有序的集合(元素存与取的顺序相同)

它可以存储重复的元素

Set:

它是一个无序的集合(元素存与取的顺序可能不同)

它不能存储重复的元素

* List集合中的特有方法(下标)
  + void add(int index, Object element) 将指定的元素，添加到该集合中的指定位置上
  + Object get(int index)返回集合中指定位置的元素。
  + Object remove(int index) 移除列表中指定位置的元素, 返回的是被移除的元素
  + Object set(int index, Object element)用指定元素替换集合中指定位置的元素,返回值的更新前的元素
* ArrayList:

底层数据结构是数组，查询快，增删慢

线程不安全，效率高

* LinkedList:

底层数据结构是链表，查询慢，增删快

线程不安全，效率高

* 泛型： 用来约束数据的数据类型
  + 泛型的格式：

<数据类型>

泛型可以使用在 类，接口，方法上

* + 泛型的好处

A:提高了程序的安全性

B:将运行期遇到的问题转移到了编译期

C:省去了类型强转的麻烦

* 增强for:在循环的过程中不能修改集合的长度

简化数组和Collection集合的遍历

格式：

for(元素数据类型 变量 : 数组或者Collection集合) {

使用变量即可，该变量就是元素

}

好处：简化遍历(语法糖)

* HashSet:

元素唯一不能重复

底层结构是 哈希表结构

元素的存与取的顺序不能保证一致

* LinkedHashSet:

元素唯一不能重复

底层结构是 链表结构+哈希表结构

+ 元素的存与取的顺序一致

如何保证元素的唯一的？

重写hashCode() 与 equals()方法