Java中的List集合

1. **List<E>接口（列表）**：存在于**java.util包**中，两个父类接口：Collection<E>和Interable<E>。
2. **接口 List<E>**有很多实现子类，其中使用最多的是**ArrayList<E>(**数组列表**)和LinkedList<E>(**链接列表**)，**还有一个就是**Vector向量**。
3. **List体系的集合**的特点：
4. **list**体系**集合中的每个元素都具有索引（角标），因此可以利用索引查找元素**；
5. list体系集合中可以存放**相同的多个元素，允许元素重复**；（set不允许重复）
6. list体系集合是**有序存放**的（**存入和取出的顺序一致**），不是**排序（set是无序的，排序）**；
7. List体系集合可以实现对元素的**增删改查如：add、remove、set、get等方法**。
8. **List集合**中的元素进行**查找（indexOf）、删除（remove）、是否包含（contains）操作**时，需要利用到元素的equals方法进行元素间的比较的；如果List集合中存储的对象是自己创建的对象，需要覆写equals方法，建立此类对象的比较依据，实现实体类自己有意义的比较。
9. Arrays类中有个静态方法**asList（T… e）**利用了**可变参数特性**。



例如：

List<String> list = **new** ArrayList<String>();

list.add("abc");list.add("cde");list.add("ghg");

**可以直接把元素写到asList方法中，即可实现添加。**

List<String> list2 = Arrays.*asList*("abc","cde","ghg");

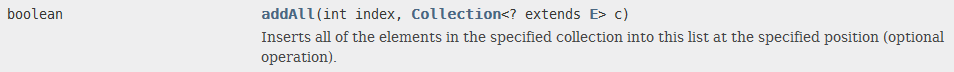
System.***out***.println(list2+"\n"+list);

1. **List<E>接口中的方法介绍：**

**既然List<E>接口是继承于Collection接口的，那么Collection接口中的方法List接口都具有，下面主要是列举一些List接口特有的方法。**

1. **add（int index，E element）：只有List集合具有索引，故有一个根据索引进行添加元素的add方法。**

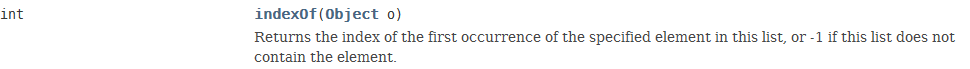


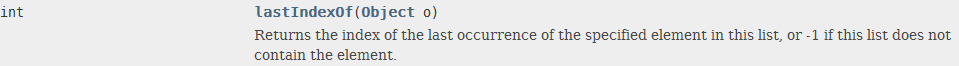


1. **get（int index）**：根据**指定索引**获取**元素**。



1. **获取索引方法**：
2. **indexOf（Object o ）**： 获取某个对象的索引。

b） lastIndexOf（Object o）：



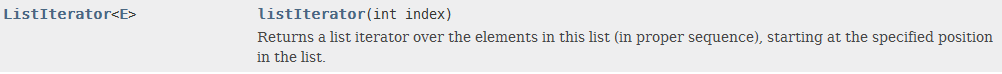
1. **获取迭代器**：
2. **iterator 方法**：获取的是**iterator迭代器**，但是接口Iterator中的方法只有三个，**hasNext、next、remove**；操作集合的方法法困乏，而**ListIterator接口**中有很多操作List集合的方法。



1. **listIterator方法**：获取一个listIterator迭代器，**默认从索引为0处开始迭代**。



1. **listIterator（int index）方法**：指定集合中从哪个索引出开始迭代。



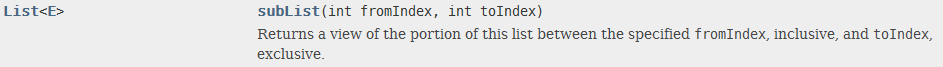
1. **remove（int index）**：删除指定索引处的元素，并将删除的元素返回回来。



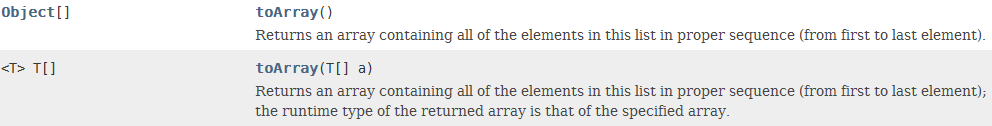
1. **修改集合元素方法**：**set（int index，E element）**：返回原来的元素。



1. **subList（int fromindex，int toindex）**： 截取获得**子列表**。



1. **toArray方法**：将list集合转换成数组。（对Collection中的方法进行**覆写**）



1. **List体系中的三大集合类型：**
2. **Vector向量：**
   1. **内部是数组数据结构；是同步的（线程安全的），效率较低；（现在vector几乎不再使用了，被ArrayList取代了）增删和查询的速度都非常慢。**
   2. **Enumeration 接口（枚举）与Iterator接口的功能是重复的。**

**vector中有个方法叫作elements（） 返回一个此向量的枚举对象。只要是打印输出容器的所有内容，for循环应优先考虑。**

1. **ArrayList数组列表：**

**内部也是数组数据结构， 是不同步的；替代了Vector。**

**查找元素的速度特别快（有角标），增删速度较慢。数组结构的地址空间是连续的，使查询速度快。**

例子：ArrayList al = new ArrayList();//空参数的构造方法，默认容量为10

p = (Person)it.next();

System.***out***.println(p.getName() + " : "+p.getAge());

//需要向下转型；而且next方法调用一次，就会往后走一次，如果需要输出一个对象的两个之上的属性或方法，一定要把此对象赋给一个引用

// it.next().getName() 不能使用的原因是it.next() 返回地是Object型的，

// 已经把对象向上转型为Object了，所以调用对象的特有方法时，必须向下转型。

1. **LinkedList链表列表：**

**内部是双向链表（链接列表）数据结构；是不同步的；增删元素的速度特别快，查询元素的速度较慢。链表结构的地址空间是不连续的，故导致查询速度慢。**

1. ArrayList、Vector的**数据结构**采用的都是**数组结构**，只不过Vector扩充时采用**倍数**扩增，而**ArrayList**进行的是**半数倍**扩充。
2. **链表结构**：
3. 单项链表：data和next；
4. 双向链表（循环链表）：previous、data和next。
5. **ArrayList 与LinkedList 的 区别**：
6. ArrayList 的底层是利用**数组结构**实现的，而LinkedList 的底层是通过**双向链表结构**实现的；
7. 当执行**删除或者插入**操作时，采用**LinkedList** 比较好；当执行**搜索查询操作**时，**ArrayList**比较好。
8. **队列 （Queue）**和**栈（Stack）**：无论是栈还是队列，里面都有一个**成员变量**，这个就是存放到栈或者队列中的元素，该成员变量可以使用**LinkedList类型**来实现。
9. **ArrayList的特有方法:**
10. **三个构造方法**
    1. **空参数：创建一个默认容量为10的集合；**



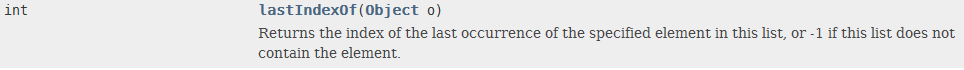
* 1. **指定集合容量大小：**



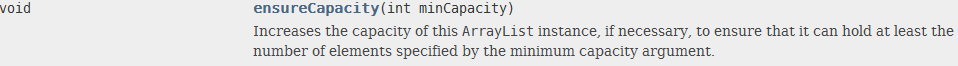
* 1. **集合作为参数，生成一个新的ArrayList集合，元素顺序为集合的迭代顺序。**



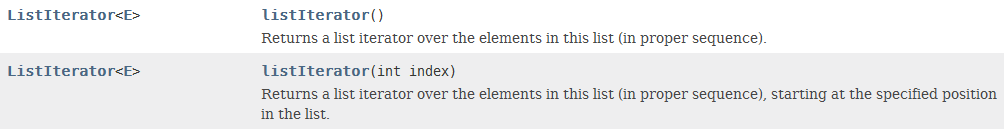
1. 特有方法：
2. lastIndexOf（）



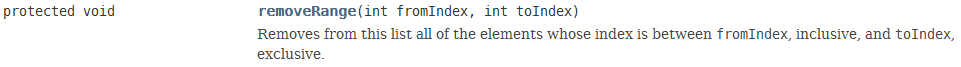
1. ensure



1. listIterator方法



1. removeRange方法



1. **sort方法**：



1. **trimToSize方法**：



1. **LinkedList集合**：
2. **两个构造方法**：
   * 1. **空参数：创建一个空列表。**



* + 1. **集合做参数**：

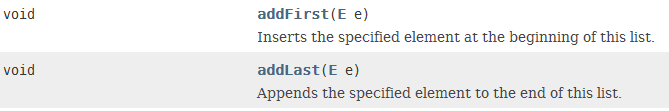


1. LinkedList集合的特有方法：**LinkedList集合也有索引**。

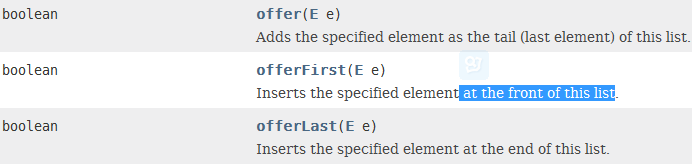
**add方法被offer取代、get方法被peek取代、remove方法被poll取代。取代的原因是当链表集合为空时，不再抛出异常，而是返回null。**

1. add方法：除了add和addAll方法外，特有方法为

**addFirst和addLast方法**：**在首和尾添加元素**。**add方法和offer方法差别不大，功能一样。**



**offer方法**：



1. 迭代器：**descendIterator和listIterator**方法：

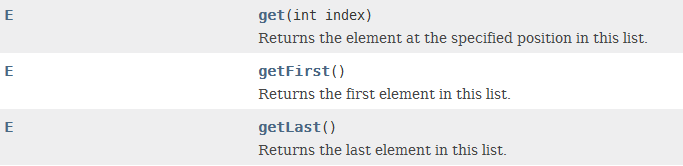




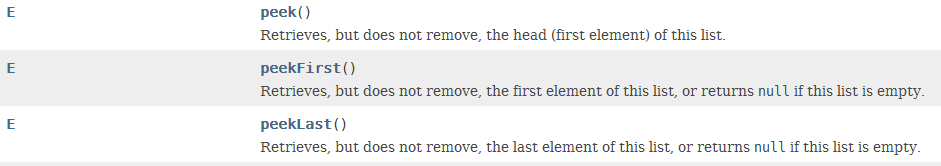
1. **获取元素方法**：

element方法、get、**getFirst、getLast方法**：**getFirst（）；getLast（）；获取元素但不删除元素，但是如果链表为空，抛出NoSuchElementException 异常，JDK1.6之后被peakFirst（） ；peakLast（）方法取代，差别在于如果链表为空时，不抛出异常，而是返回null。**





**peek方法取代了get、getFirst、getLast方法**：

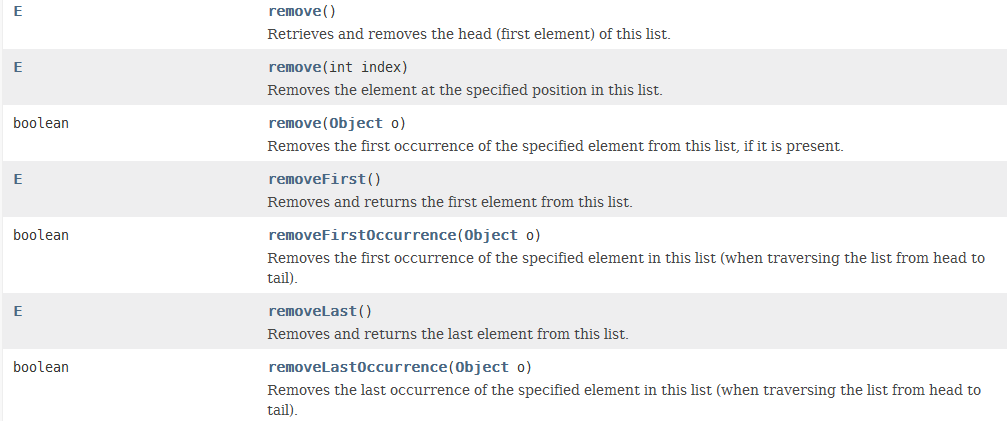


1. pop方法：和push方法：

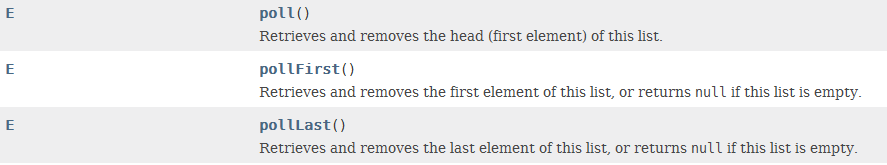




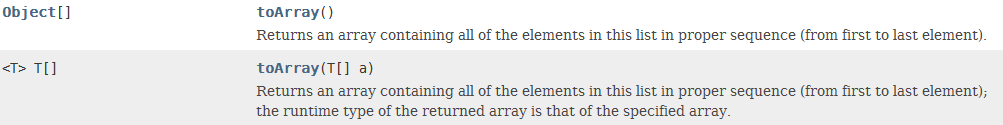
1. remove方法：**removeFirst（）；removeLast()；获取元素且删除此元素，但是当链表为空时，抛出NoSuchElementException 异常，JDK1.6之后被poolFirst（） ；poolLast（）方法取代，差别在于如果链表为空时，不抛出异常，而是返回null。**



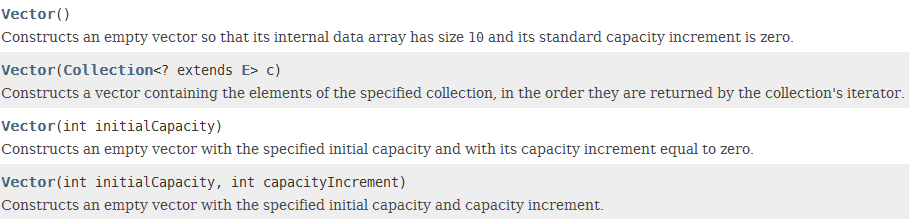
**poll方法：取代了remove、removeFirst、removeLast方法。**



1. **set修改方法：返回原来的元素。**
2. **toArray方法:**



1. **removeFirst（） 与getFirst（）的区别**：都会返回第一个对象；但是前者返回后，立即将第一个元素输出，但是后者**不会删除第一个元素**。removeLast和getLast方法同理。pollFirst和peekFirst、pollLast和peekLast方法同上。
2. **Vector向量**：（已经很少使用，逐渐被ArrayList取代）
3. 构造方法：



1. 特有方法：
2. **elements（**）: 返回一个枚举对象。



1. **sort方法**：



1. capacity和size







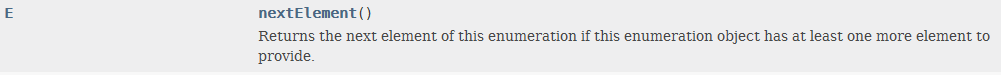
1. **trimToSize方法**：



1. Enumeration接口：和Iterator接口功能一致。只有两个方法：
2. hasMoreElements



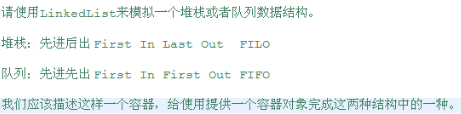
1. nextElement()方法:



1. 练习题
2. **关于LinkedList类的面试题：**

**请使用LinkedList ；来模拟一个堆栈或者队列的数据结构。**

**分析：我们应该描述这样一个容器，给使用者提供一个容器对象，完成这两种结构的一种。**



**① 堆栈：先进后出FILO，后进先出LIFO**

**package** sone.November26.pm;

**import** java.util.LinkedList;

**public** **class** DuiZhan {

**private** LinkedList link =**null**;

DuiZhan(){

link = **new** LinkedList();

}

**public** **void** myAdd(Object obj) {

link.addFirst(obj);

}

**public** Object myGet() {

**return** link.removeFirst();

//因为取出来后就不存在了，故需要使用removeFirst（）,不能使用getFirst（）

}

**public** **boolean** myIsEmpty() {

**return** link.isEmpty();

}}

**② 队列：先进先出LILO**

**package** sone.November26.pm;

**import** java.util.LinkedList;

**public** **class** DuiLie {

**private** LinkedList link = **null**;

**public** DuiLie() {

link = **new** LinkedList();

}

**public** **void** myAdd(Object obj) {

link.addFirst(obj);

}

**public** Object myGet() {

**return** link.removeLast();

}

**public** **boolean** myIsEmpty() {

**return** link.isEmpty();

}

}

**ackage** sone.November26.pm;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

//下午5:24:56

DuiLie dl = **new** DuiLie();

dl.myAdd("abc1");

dl.myAdd("abc2");

dl.myAdd("abc3");

dl.myAdd("abc4");

**while**(!dl.myIsEmpty())

{

System.***out***.println(dl.myGet());

}

DuiZhan dz = **new** DuiZhan();

dz.myAdd("abc1");

dz.myAdd("abc2");

dz.myAdd("abc3");

dz.myAdd("abc4");

**while**(!dz.myIsEmpty())

{

System.***out***.println(dz.myGet());

}

}

}