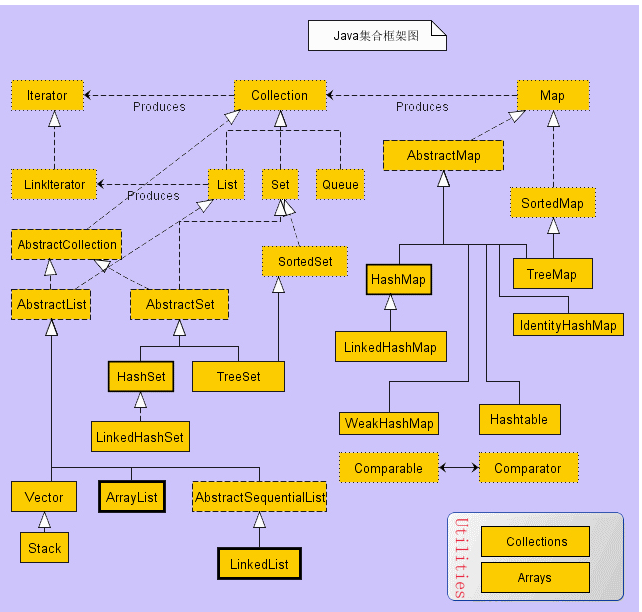
# java中的集合框架

java中的集合类主要有两大分支：（1）Collection （2）Map。

为了实现迭代遍历，定义了Iterable接口，collection接口继承了Iterable接口，以支持所有集合类实现迭代遍历。

类图如下：



# 用过哪些集合 ？每种集合有什么特点。

# 集合的体系

## 1.List

**List 有序(存储顺序和取出顺序一致)，可重复。**

### 1.1 ArrayList

**线程不安全**，底层使用**数组**实现，**查询快，增删慢。效率高**。 每次容量不足时进行扩容，JDK1.7后自增长度的一半。

### 1.2 LinkedList

线程不安全，底层使用双向链表实现，查询慢，增删快。效率高。

### 1.3 Vector

底层使用**数组**实现，**查询快，增删慢**。效率低 每次容量不足时，默认自增长度的一倍（如果不指定增量的话）

## 2. Set

**Set 无序(存储顺序和取出顺序不一致)，不可重复。**

存入Set里的元素都是唯一的，区分存入的元素是否重复是通过调用Equals方法来进行判断，所有存入Set中的元素类必须实现Equals方法。

### 2.1HashSet

HashSet是将存入Set中的元素以Hash链表的方式存储起来，所有元素存入Set中的位置通过调用hashCode方法获取hash值来决定，所以需要设计好的hashcode方法尽量将存入的值散列开有利于提升HashSet的访问效率。

### 2.2 LinkedHashSet

继承于HashSet和HashSet的唯一区别是维护了一个双向列表来维护元素的顺序，所有访问是按顺序访问。

### 2.3 TreeSet

TreeSet继承于AbstractSet，并且实现了NavigableSet接口。

TreeSet实现了一个顺序访问的不重复元素的Set，底层使用红黑树进行数据的存储，来加快访问的速度。

### 2.4 EnumSet

EnumSet只能用来存放Enum类型的数据，也不允许重复数据，性能是最好的。

## 3 Map

Map和Set的区别是Set只有值，而Map是一个键值对<key,value>，Set不存重复的元素，Map中的key不能相同。

### 3.1 HashMap

和HashSet相似，用hash链表来存储，讲键值对作为一个元素存储

### 3.2 LinkedHashMap

类似于HashMap，但是迭代遍历它时，取得“键值对”的顺序是其插入次序，或者是最近最少使用(LRU)的次序。只比HashMap慢一点。而在迭代访问时反而更快，因为它使用链表维护内部次序。

**3.3 TreeMap**

基于红黑树数据结构的实现。查看“键”或“键值对”时，它们会被排序(次序由Comparabel或Comparator决定)。TreeMap的特点在于，你得到的结果是经过排序的。TreeMap是唯一的带有subMap()方法的Map，它可以返回一个子树。

3.4 **Hashtable**

与 HashMap类似,不同的是:key和value的值均不允许为null;它支持线程的同步，即任一时刻只有一个线程能写Hashtable,因此也导致了Hashtale在写入时会比较慢。

## 介绍Collection

## 知道哪些Map

**HashMap**：我们最常用的Map，它根据key的HashCode 值来存储数据,根据key可以直接获取它的Value，同时它具有很快的访问速度。HashMap最多只允许一条记录的key值为Null(多条会覆盖);允许多条记录的Value为 Null。非同步的。

**TreeMap**: 能够把它保存的记录根据key排序,默认是按升序排序，也可以指定排序的比较器Comparator改变排序的方式。当用Iterator 遍历TreeMap时，得到的记录是排过序的。TreeMap不允许key的值为null。非同步的。

**Hashtable**: 与 HashMap类似,不同的是:key和value的值均不允许为null;它支持线程的同步，即任一时刻只有一个线程能写Hashtable,因此也导致了Hashtale在写入时会比较慢。

**LinkedHashMap**: 保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的.在遍历的时候会比HashMap慢。key和value均允许为空，非同步的。

## List和Set有什么区别

List：1.可以允许重复的对象。

　　  2.可以插入多个null元素。

        3.是一个有序容器，保持了每个元素的插入顺序，输出的顺序就是插入的顺序。

        4.常用的实现类有 ArrayList、LinkedList 和 Vector。ArrayList 最为流行，它提供了使用索引的随意访问，而 LinkedList 则对于经常需要从 List 中添加或删除元素的场合更为合适。

Set：1.不允许重复对象

    2. 只允许一个 null 元素

　　  3. 无序容器，你无法保证每个元素的存储顺序，TreeSet通过 Comparator  或者 Comparable 维护了一个排序顺序。

        4.Set 接口最流行的几个实现类是 HashSet、LinkedHashSet 以及 TreeSet。最流行的是基于 HashMap 实现的 HashSet；TreeSet 还实现了 SortedSet 接口，因此 TreeSet 是一个根据其 compare() 和 compareTo() 的定义进行排序的有序容器。

// List、Set和Map有什么区别?

1.Map不是collection的子接口或者实现类。Map是一个接口。

2.Map 的 每个 Entry 都持有两个对象，也就是一个键一个值，Map 可能会持有相同的值对象但键对象必须是唯一的。

3. TreeMap 也通过 Comparator  或者 Comparable 维护了一个排序顺序。

4. Map 里你可以拥有随意个 null 值但最多只能有一个 null 键。

5.Map 接口最流行的几个实现类是 HashMap、LinkedHashMap、Hashtable 和 TreeMap。（HashMap、TreeMap最常用）

## 什么场景下使用list，set，map呢？

1.如果你经常会使用索引来对容器中的元素进行访问，那么 List 是你的正确的选择。如果你已经知道索引了的话，那么 List 的实现类比如 ArrayList 可以提供更快速的访问,如果经常添加删除元素的，那么肯定要选择LinkedList。

2.如果你想容器中的元素能够按照它们插入的次序进行有序存储，那么还是 List，因为 List 是一个有序容器，它按照插入顺序进行存储。

3.如果你想保证插入元素的唯一性，那么可以选择一个 Set 的实现类，比如 HashSet、LinkedHashSet 或者 TreeSet。所有 Set 的实现类都遵循了统一约束比如唯一性，而且还提供了额外的特性比如 TreeSet 还是一个 SortedSet，所有存储于 TreeSet 中的元素可以使用 Java 里的 Comparator 或者 Comparable 进行排序。LinkedHashSet 也按照元素的插入顺序对它们进行存储。

4.如果你以键和值的形式进行数据存储那么 Map 是你正确的选择。你可以根据你的后续需要从 Hashtable、HashMap、TreeMap 中进行选择。

## 数组(Array)和列表(ArrayList)有区别？

1、存储内容比较：  
Array 数组可以包含基本类型和对象类型；  
ArrayList 却只能包含对象类型；  
Array 数组在存放的时候一定是同种类型的元素。ArrayList 就不一定了 。

2、空间大小比较：  
Array 数组的空间大小是固定的,所以需要事前确定合适的空间大小。  
ArrayList 的空间是动态增长的,而且每次添加新的元素的时候都会检查内部数组的空间是否足够。

3.方法上的比较：  
ArrayList 方法上比 Array 更多样化，比如添加全部 addAll()、删除全部 removeAll()、返回迭代器 iterator() 等。

## arraylist和linkedlist的区别

ArrayList 和 LinkedList 都实现了 List 接口，他们有以下的不同点：

1.ArrayList 是基于索引的数据接口，它的底层是数组。它可以以O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList 中的元素以双向链表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是O(n)。

2.相对于 ArrayList，LinkedList的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。

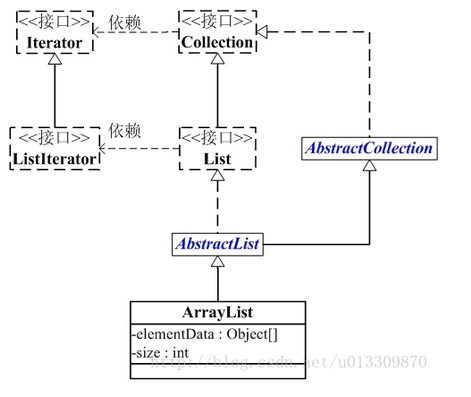
3.LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

## ArrayList底层实现原理

ArrayList简介

ArrayList就是动态数组，用MSDN中的说法，就是Array的复杂版本，它提供了动态的增加和减少元素，实现了Collection和List接口，可以灵活的设置数组的大小。要注意的是ArrayList并不是线程安全的，因此一般建议在单线程中使用ArrayList。

ArrayList的继承关系



ArrayList的方法使用和源码解析

底层数据结构：动态数组

数组初始容量为 10

数组最大容量：Integer.MAX\_VALUE - 8

 ArrayList是List接口的一个可变大小的数组的实现；

 ArrayList的内部是使用一个Object对象数组来存储元素的；

初始化ArrayList的时候，可以指定初始化容量的大小，如果不指定，就会使用默认大小，为10。

指定容量：List<String> myList = new ArrayList<String>(7)

默认容量：myList = new ArrayList();//第一次add的时候，这个数组就会被初始化一个大小为10的数组

list包含另外一个特定的collection的元素:

ArrayList<Integer> list = new ArrayList<>(set);

//用户可以往ArrayList中传入一个容器只要这个容器是Collection类型的。调用ArrayList(Collection<? extends E> c)接口的时候会将容器数组化处理并将这个数组值赋给Object数组。

size(), isEmpty(), get(), set() iterator(), ListIterator() 方法的时间复杂度都是 O(1)

add() 添加操作的时间复杂度平均为 O(n)

其他所有操作的时间复杂度几乎都是 O(n)

1. **indexOf(Object o)方法**



//对于indexof方法做几点说明：ArrayList中可以存放null元素，indexof是返回elementData数组中值相同的首个元素的下标，indexof中比较方法是equals而equals是比较元素的值，**因此必须对null单独查找**。如果未找到该元素则返回-1 。

2. **add(E e)方法**

功能：往ArrayList中添加元素。



//add方法比较复杂，涉及到扩充数组容量的问题。其中要弄清楚size和elementData.length的区别，size指的是数组中存放元素的个数，elementData.length表示数组的长度，当new一个ArrayList系统默认产生一个长度为10的elementData数组，elementData.length=10，但是由于elementData中还未放任何元素所有size=0。如果加入元素后数组大小不够会先进行扩容，每次扩容都将数组大小增大一半比如数组大小为10一次扩容后的大小为10+5=10;ArrayList的最大长度为 2^32 .

## ArrayList的初始化容量，扩容？

不同JdK版本ArrayList 扩容机制不同。

1. **JDK1.6及以前版本**

 默认的构造函数初试化时的长度为10，也可以指定ArrayList的初始长度，一旦发现容量不足，会自动扩充容量，新的大小是：

int newCapacity = (oldCapacity \* 3)/2 + 1；

也就是原有容量的1.5倍+1。然后通过底层的复制方法将原有数据复制过来。

**2.JDK1.7**

默认的构造函数 初试化时的长度为10, 一旦发现容量不足，会自动扩充容量，新的大小是：

int newCapacity = oldCapacity + (oldCapacity >> 1);

将新容量更新为旧容量的1.5倍。然后通过底层的复制方法将原有数据复制过来。

具体的过程如下：

首先得到数组的旧容量，然后进行oldCapacity + (oldCapacity >> 1)，将oldCapacity 右移一位，我们知道位运算的速度远远快于整除运算，整句运算式的结果就是将新容量更新为旧容量的1.5倍；后检查新容量是否大于最小需要容量，若还是小于最小需要容量，那么就把最小需要容量当作数组的新容量。再检查新容量是否超出了ArrayList所定义的最大容量，若超出了，则调用hugeCapacity()来比较minCapacity和MAX\_ARRAY\_SIZE，如果minCapacity大于最大容量，则新容量则为ArrayList定义的最大容量，否则，新容量大小则为minCapacity。

从代码上，我们可以看出区别：

第一：在容量进行扩展的时候，其实例如整除运算将容量扩展为原来的1.5倍加1，而jdk1.7是利用位运算，从效率上，jdk1.7就要快于jdk1.6。

第二：JDK1.6中，算出newCapacity时，其没有和ArrayList所定义的MAX\_ARRAY\_SIZE作比较，为什么没有进行比较呢，原因是jdk1.6没有定义这个MAX\_ARRAY\_SIZE最大容量，也就是说，其没有最大容量限制的，但是jdk1.7做了一个改进，进行了容量限制。

## linkedlist 怎么取倒数第二个结点？

//这是一道剑指offer的算法题

设置一快一慢指针，快指针先走N步，然后快慢指针同时走，当快指针走到末尾时，慢指针对应的节点就是倒数第N个节点。

## Collection 和 Collections的区别

1.Collection是集合类的上级接口，子接口主要有Set 和List。map集合虽然也属于集合体系，但是map并不继承collection，map和collection是平级关系。

2.Collections是针对**集合类的一个帮助类**，提供了操作集合的**工具方法**：一系列静态方法实现对各种集合的**搜索**、**排序**、**线程安全化等**操作。

## HashSet原理

HashSet的实现原理总结如下：

①是基于HashMap实现的，默认构造函数是构建一个初始容量为16，负载因子为0.75 的HashMap。封装了一个 HashMap 对象来存储所有的集合元素，所有放入 HashSet 中的集合元素实际上由 HashMap 的 key 来保存，而 HashMap 的 value 则存储了一个 PRESENT，它是一个静态的 Object 对象。

②当我们试图把某个类的对象当成 HashMap的 key，或试图将这个类的对象放入 HashSet 中保存时，重写该类的equals(Object obj)方法和 hashCode() 方法很重要，而且这两个方法的返回值必须保持一致：当该类的两个的 hashCode() 返回值相同时，它们通过 equals() 方法比较也应该返回 true。通常来说，所有参与计算 hashCode() 返回值的关键属性，都应该用于作为 equals() 比较的标准。

③HashSet的其他操作都是基于HashMap的。

## TreeSet实现原理？TreeSet实现了哪些接口？

1.TreeSet原理:  
  TreeSet存储对象的时候, 可以排序, 但是需要指定排序的算法；  
 Integer能排序(有默认顺序), String能排序(有默认顺序), 自定义的类存储的时候出现异常(没有顺序)；  
 如果想把自定义类的对象存入TreeSet进行排序, 那么必须实现Comparable接口：  
 \*   在类上implement Comparable；  
 \*   重写compareTo()方法；  
 \*   在方法内定义比较算法, 根据大小关系, 返回正数负数或零；  
 \*   在使用TreeSet存储对象的时候, add()方法内部就会自动调用compareTo()方法进行比较, 根据比较结果使用二叉树形式进行存储。  
  
TreeSet中的元素支持两种排序方式：自然排序和定制排序。

TreeSet的基本操作（add、remove、和contains）的时间复杂度都为log（n），并且 TreeSet也是非同步的，即是非线程安全的；   
TreeSet的Iterator返回的迭代器也是基于Fail-Fast机制。

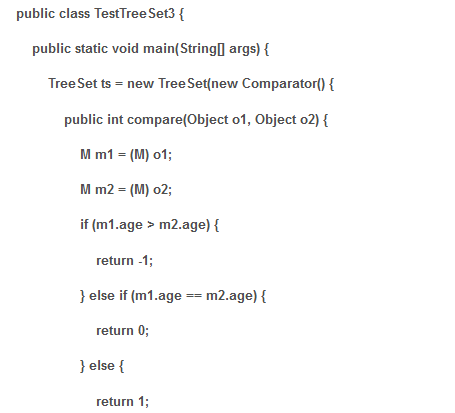
**TreeSet支持两种排序方法：自然排序和定制排序。TreeSet默认采用自然排序。**

1、自然排序    TreeSet会调用集合元素的compareTo(Object obj)方法来比较元素之间大小关系，然后将集合元素按升序排列，这种方式就是自然排序。（比较的前提：两个对象的类型相同）。

 java提供了一个Comparable接口，该接口里定义了一个compareTo(Object obj)方法，该方法返回一个整数值，实现该接口的类必须实现该方法，实现了该接口的类的对象就可以比较大小。当一个对象调用该方法与另一个对象进行比较，例如obj1.comparTo(obj2),如果该方法返回0，则表明这两个对象相等；如果返回一个正整数，则表明obj1大于obj2；如果该方法返回一个负整数，则表明obj1小于obj2.如果试图把一个对象添加进TreeSet时，则该对象的类必须实现Comparable接口。

2、定制排序

TreeSet的自然排序是根据集合元素的大小，TreeSet将他们以升序排列。如果需要实现定制排序，例如降序，则可以使用Comparator接口。该接口里包含一个int compare（T o1， T o2）方法，该方法用于比较o1和o2的大小。  
   如果需要实现定制排序，则需要在创建TreeSet集合对象时，并提供一个Comparator对象与该TreeSet集合关联，由该Comparator对象负责集合元素的排序逻辑。



   上面程序中创建了一个Comparator接口的匿名内部类对象，该对象负责ts集合的排序。所以当我们把M对象添加到ts集合中时，无须M类实现 Comparable接口，因为此时TreeSet无须通过M对象来比较大小，而是由与TreeSet关联的Comparator对象来负责集合元素的排序。使用定制排序时，TreeSet对集合元素排序时不管集合元素本身的大小，而是由Comparator对象负责集合元素的排序规则。

### 2.TreeSet实现了哪些接口？

TreeSet是基于TreeMap（红黑树）实现的，它继承于AbstractSet抽象类，实现了Serializable, Cloneable, Iterable<E>, Collection<E>, NavigableSet<E>, Set<E>, SortedSet<E>接口。

TreeSet继承于AbstractSet，即是一个Set集合，具有Set的属性和方法；   
TreeSet实现了NavigableSet接口，意味着支持一系列最优的搜索方法；   
TreeSet实现了Cloneable接口，意味着可以被克隆，得到副本。

## CopyOnWriteArrayList的实现原理

CopyOnWriteArrayList容器是Collections.synchronizedList(List list)的替代方案，CopyOnWriteArrayList在某些情况下具有更好的性能，考虑读远大于写的场景，如果把所有的读操作进行加锁，因为只有一个读线程能够获得锁，所以其他的读线程都必须等待，大大影响性能。CopyOnWriteArrayList称为“写时复制”容器，就是在多线程操作容器对象时，把容器复制一份，这样在线程内部的修改就与其他线程无关了，而且这样设计可以做到不阻塞其他的读线程。从JDK1.5开始Java并发包里提供了两个使用CopyOnWrite机制实现的并发容器，它们是CopyOnWriteArrayList和CopyOnWriteArraySet。

**CopyOnWriteArrayList源码**

先说说CopyOnWriteArrayList容器的实现原理：简单地说，就是在需要对容器进行操作的时候，将容器拷贝一份，对容器的修改等操作都在容器的拷贝中进行，当操作结束，再把容器容器的拷贝指向原来的容器。这样设计的好处是实现了读写分离，并且读读不会发生阻塞。下面的源码是CopyOnWriteArrayList的add方法实现：



上面的三个步骤实现了写时复制的思想，在读数据的时候不会锁住list，因为写操作是在原来容器的拷贝上进行的。而且，可以看到，如果对容器拷贝修改的过程中又有新的读线程进来，那么读到的还是旧的数据。

CopyOnWrite并发容器用于读多写少的并发场景。比如**白名单，黑名单，商品类目的访问和更新场景。**

**CopyOnWriteArrayList的缺点**

从CopyOnWriteArrayList的实现原理可以看到因为在需要容器对象的时候进行拷贝，所以存在两个问题：内存占用问题和数据一致性问题。

**内存占用问题**

因为需要将原来的对象进行拷贝，这需要一定的开销。特别是当容器对象过大的时候，因为拷贝而占用的内存将增加一倍（原来驻留在内存的对象仍然在使用，拷贝之后就有两份对象在内存中，所以增加了一倍内存）。而且，在高并发的场景下，因为每个线程都拷贝一份对象在内存中，这种情况体现得更明显。由于JVM的优化机制，将会触发频繁的Young GC和Full GC，从而使整个系统的性能下降。

**数据一致性问题**

CopyOnWriteArrayList不能保证**实时一致性**，因为读线程在将引用重新指向原来的对象之前再次读到的数据是旧的，所以CopyOnWriteArrayList **只能保证最终一致性**。因此在需要实时一致性的场景CopyOnWriteArrayList是不能使用的。

**CopyOnWriteArrayList小结**

1. CopyOnWriteArrayList适用于读多写少的场景
2. 在并发操作容器对象时不会抛出ConcurrentModificationException，并且返回的元素与迭代器创建时的元素是一致的
3. 容器对象的复制需要一定的开销，如果对象占用内存过大，可能造成频繁的YoungGC和Full GC
4. CopyOnWriteArrayList不能保证数据实时一致性，只能保证最终一致性