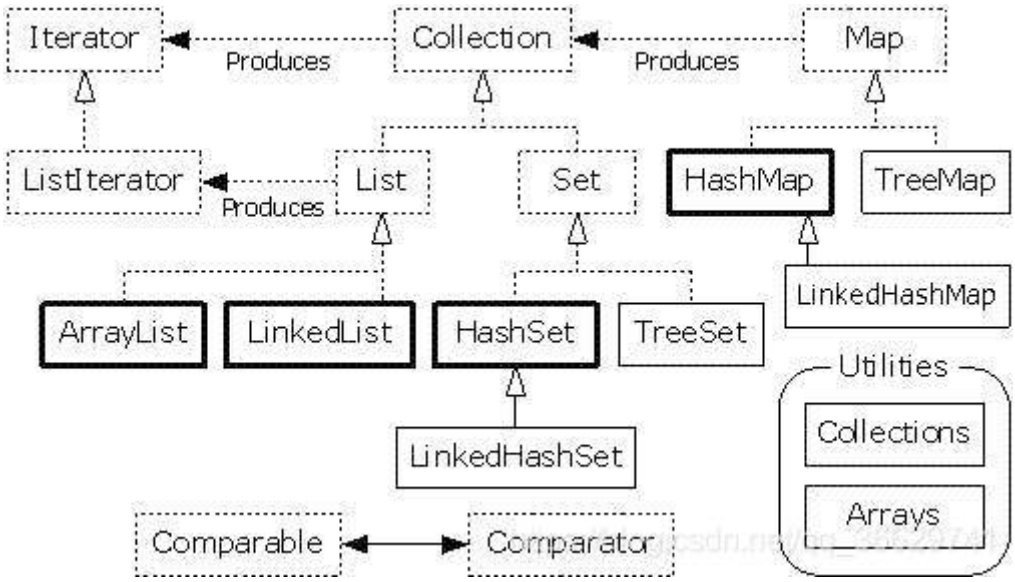
集合框架



集合类存在的意义

存储对象使用数组，**存储对象**使用集合(集合中也只能保存对象)

集合和数组的不同

数组虽然也可以存储对象，但长度是固定的；**集合长度是可变的。**
数组中可以存储基本数据类型，**集合只能存储对象。(可以是不同类型的)**

集合框架的由来

不断向上抽取而来的，因为不同的容器对于数据的存储方式不同，所以进行了单独的划分(即**数据结构**)

集合框架为所有集合类提供了增删改查，清空容器，判断容器是否为空等接口，供他们自己实现

迭代器

- 迭代器是**集合的取出元素的方式**，会直接访问(例如**判断/取出**)集合中的元素。所以将迭代器通过**内部类**的形式来进行描述。
- 通过容器的iterator()方法**获取该内部类的对象**，这个对象可以直接访问集合中的成员·
  (iterator()方法相当于抓娃娃机的控制杆，内部类的对象(对于Arraylist是Itr这个内部类)则相当于**夹娃娃机里面的夹子**，可以访问集合中的成员)

```
ArrayList al = new ArrayList();
Iterator it = al.iterator();//获取迭代器，用于取出集合中的元素。
```

collection下的常见子接口

1. List

元素是有序的，且可以重复(因为有索引)

特有方法

增

- **add**(index,element);
- **addAll**(index,Collection);

删

- **remove**(index);

改

- **set**(index,element);

查

- **get**(index):
- **subList**(from,to);
- **listIterator();**
  不能同时用集合和迭代器修改List中的元素，会出现ConcurrentModificationExceotion并发修改异常，迭代器Iterator()没法添加，于是定义了listIterator()子接口

  int indexOf(obj):获取指定元素的位置。
  ListIterator listIterator();

三种常用List

- **ArrayList:**底层的数据结构使用的是**数组结构。特点：查询/修改速度很快。但是增删稍慢。线程不同步。**
- **LinkedList:**底层使用的**链表数据结构。特点：增删速度很快，查询稍慢。线程不同步。**
- **Vector:**底层是数组数据结构。**线程同步**。被ArrayList替代了。因为效率低。（已经不用了）

List的特有方法

- offerFirst(); 队首插入
- offerLast();队尾插入
- peekFirst();
- peekLast();

获取元素，但不删除元素。如果集合中没有元素，会返回null。

- pollFirst();
- pollLast();
  获取元素，但是元素被删除。如果集合中没有元素，会返回null。

2. Set

元素是无序的，元素不可以重复

set基本可以被认为等于collection(除了set集合是无序的，元素不可以重复)

常用Set

1. HashSet

按照哈希算法来存储集合中的元素，因此查询/存取效率很高
特点

- 不保证插入元素的顺序(可能相同，也可能不同)
- 线程不安全
- 元素值可以为null

Hashset集合判断两个元素是否相等

通过equals方法返回相等(true)，HashCode值也相等，所以如果需要重写equals方法，则需要保证equals返回true时，HashCode方法返回值也相同(如果不同则会导致HashSet把这两个相同值的元素当成不同的元素处理)，所以向HashCode中添加可变对象时也要格外小心，不要与已有的值重复。

2. LinkedHashSet

使用链表维护插入对象的顺序，所以性能略低于HashSet，但是遍历时效率更高。

3. TreeSet

SortedSet接口的实现类，保证元素有序(有序并不是指插入顺序，而是元素的实际值大小)
因为TreeSet集合保证元素有序，所以提供了访问第一个，前一个，后一个，最后一个元素的方法。并提供了截取子TreeSet的方法

只有当需要维护一个有序的Set时，才需要用TreeSet，否则选用性能更好的HashSet

4. EnmuSet

存储枚举类对象

3. Map

Map用来保存具有映射关系的数据(键值对)，Map中所有的key组合在一起，就形成了一个Set。
从源码角度来说，Java是先实现了Map，然后把所有value值都设定为null，就实现了Set.

4.deque 双端队列

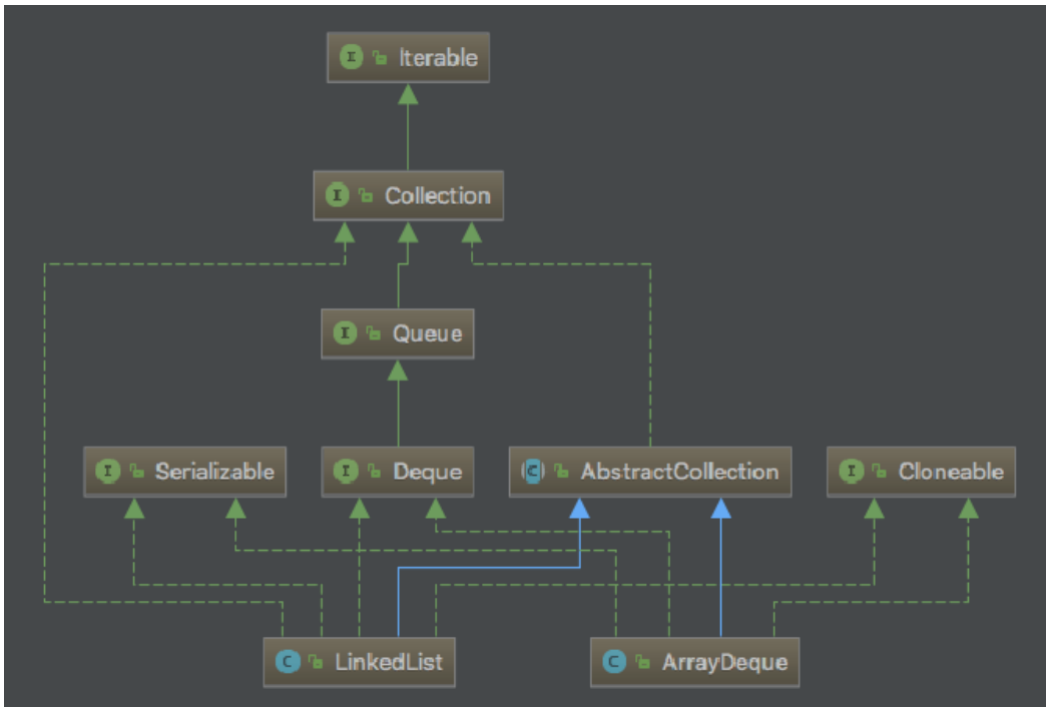
Java中实际上提供了`java.util.Stack`来实现栈结构,但官方目前已不推荐使用,而是使用`java.util.Deque`双端队列来实现队列与栈的各种需求.如下图所示`java.util.Deque`的实现子类有`java.util.LinkedList`和`java.util.ArrayDeque`.顾名思义前者是基于链表,后者基于数据实现的双端队列.

Deque 是 *Double ended queue (双端队列)* 的缩写,读音和 deck 一样，蛋壳。

Deque 继承自 Queue,直接实现了它的有 LinkedList, ArayDeque, ConcurrentLinkedDeque 等。

Deque 支持容量受限的双端队列，也支持大小不固定的。一般双端队列大小不确定。

Deque 接口定义了一些从头部和尾部访问元素的方法。比如分别在头部、尾部进行插入、删除、获取元素。



Deque与Queue相对应的接口：

Queue Method	Equivalent Deque Method	说明
`add(e)`	`addLast(e)`	向队尾插入元素，失败则抛出异常
`offer(e)`	`offerLast(e)`	向队尾插入元素，失败则返回 `false`
`remove()`	`removeFirst()`	获取并删除队首元素，失败则抛出异常
`poll()`	`pollFirst()`	获取并删除队首元素，失败则返回 `null`
`element()`	`getFirst()`	获取但不删除队首元素，失败则抛出异常
`peek()`	`peekFirst()`	获取但不删除队首元素，失败则返回 `null`

Deque与Stack对应的接口：

Stack Method	Equivalent Deque Method	说明
`push(e)`	`addFirst(e)`	向栈顶插入元素，失败则抛出异常
无	`offerFirst(e)`	向栈顶插入元素，失败则返回 `false`
`pop()`	`removeFirst()`	获取并删除栈顶元素，失败则抛出异常
无	`pollFirst()`	获取并删除栈顶元素，失败则返回 `null`
`peek()`	`peekFirst()`	获取但不删除栈顶元素，失败则抛出异常
无	`peekFirst()`	获取但不删除栈顶元素，失败则返回 `null`

操作集合的工具类：Collection

Collection工具类里提供了大量方法对集合元素进行排序，查询和修改等操作，还提供了将集合对象设置为不可见，实现同步控制等方法。