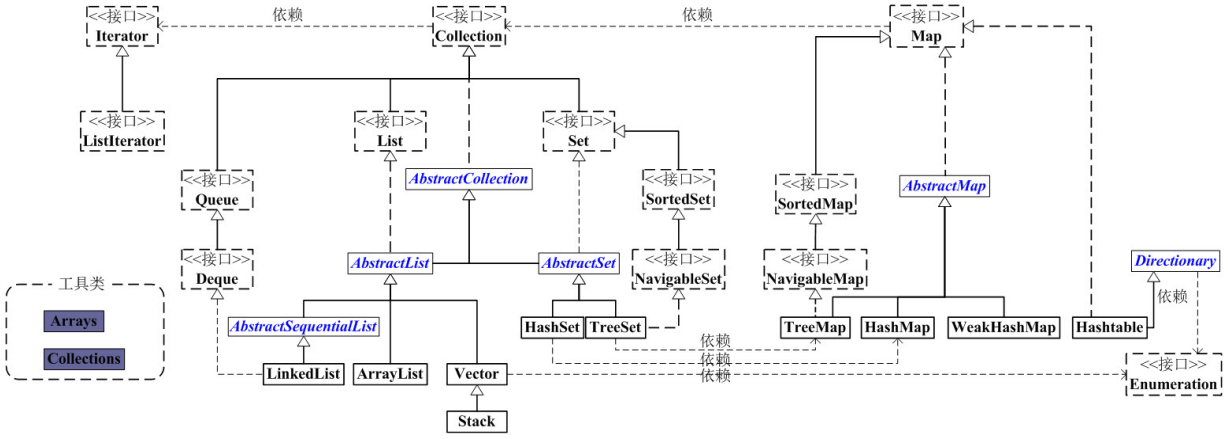
Java中的集合框架大类可分为Collection和Map，是Java集合框架的根接口。两者的区别：

1、Collection是单列集合；Map是双列集合

2、Collection中只有Set系列要求元素唯一；Map中键需要唯一，值可以重复

3、Collection的数据结构是针对元素的；Map的数据结构是针对键的。

Java集合主要可以划分为4个部分：List列表、Set集合、Map映射、工具类(Iterator迭代器、Enumeration枚举类、Arrays和Collections)。Java集合工具包位置是java.util.*



遍历

Iterator

遍历集合的工具，Collection继承了Iterator，iterator()方法返回一个Iterator<E>对象。

boolean	**hasNext**() 如果仍有元素可以迭代，则返回 true。
E	**next**() 返回迭代的下一个元素。

ListIterator

专门为遍历List而存在的，ListIterator()返回一个ListIterator对象。与Iterator相比，ListIterator增加了前向迭代的功能，且增加了如

下方法：

**boolean hasPrevious(): **如果以逆向遍历列表。如果迭代器有上一个元素，则返回 true。

Object previous():返回迭代器的前一个元素。

void add(Object o):将指定的元素插入列表（可选操作）。

工具集

Arrays和Collections。它们是操作数组、集合的两个工具类。

java.util.Collections类 提供了对Set、List、Map的辅助方法。

Collections 工具类常用方法:

1. 排序
2. 查找,替换操作
3. 同步控制(不推荐，若需要线程安全的集合类型时考虑使用 JUC 包下的并发集合)

排序

```
void reverse(List list)//反转
void shuffle(List list)//随机排序
void sort(List list)//按自然排序的升序排序
void sort(List list, Comparator c)
   //定制排序，由Comparator控制排序逻辑
void swap(List list, int i , int j)//交换两个索引位置的元素
void rotate(List list, int distance)
   //旋转。当distance为正数时，将list后distance个元素整体移到前面。当distance为负数时，将 list的前distance个元素整体移到后面。
```

查找,替换

```
int binarySearch(List list, Object key)
//对List进行二分查找，返回索引，注意List必须是有序的
int max(Collection coll)
//根据元素的自然顺序，返回最大的元素。类比int
min(Collection coll)
int max(Collection coll, Comparator c)
//根据定制排序，返回最大元素，排序规则由Comparatator类控制。类比int min(Collection coll, Comparator c)
void fill(List list, Object obj)
//用指定的元素代替指定list中的所有元素。
int frequency(Collection c, Object o)
//统计元素出现次数
int indexOfSubList(List list, List target)
//统计target在list中第一次出现的索引，找不到则返回-1，类比
int lastIndexOfSubList(List source, list target).
boolean replaceAll(List list, Object oldVal, Object
newVal), 用新元素替换旧元素
```

同步控制

Collections提供了多个synchronizedXxx()方法，。HashSet，TreeSet，ArrayList,LinkedList,HashMap,TreeMap 都是线程不安全的。Collections提供了多个静态方法可以把他们包装成线程同步的集合。

但效率非常低，需要线程安全的集合类型时去使用 JUC 包下的并发集合。

方法如下：

```
synchronizedCollection(Collection<T>  c) //返回指定 collection 支持的同步（线程安全的）collection。
synchronizedList(List<T> list)//返回指定列表支持的同步（线程安全的）List。
synchronizedMap(Map<K,V> m) //返回由指定映射支持的同步（线程安全的）Map。
synchronizedSet(Set<T> s) //返回指定 set 支持的同步（线程安全的）set。
```

Collections还可以设置不可变集合，提供了如下三类方法：

```
emptyXxx(): 返回一个空的、不可变的集合对象，此处的集合既可以是List，也可以是Set，还可以是Map。
singletonXxx(): 返回一个只包含指定对象（只有一个或一个元素）的不可变的集合对象，此处的集合可以是：List，Set，Map。
```

unmodifiableXxx(): 返回指定集合对象的不可变视图，此处的集合可以是：List，Set，Map。
上面三类方法的参数是原有的集合对象，返回值是该集合的"只读"版本。

Arrays类

1. 排序 : sort()

```
sort(int[] a)                    //按照数字顺序排列指定的数组。如果是字符串，也会对每个字符串的位置进行比较。
sort(int[] a,int fromIndex,int toIndex) //按升序排列数组的指定范围
parallelSort(int[] a)                                    //按照数字顺序排列指定的数组(并行的)。同sort方法一样也有按范围的排序
parallelSort                    //给字符数组排序，sort也可以
```

2. 查找 : binarySearch()

```
Arrays.binarySearch(e, 'c')// 排序后再进行二分查找，否则找不到
```

3. 比较: equals()

```
//元素数量相同，并且相同位置的元素相同。 另外，如果两个数组引用都是null，则它们被认为是相等的 。
```

4. 填充 : fill()

```
        Arrays.fill(g, 3);              //全部替换
        Arrays.fill(h, 0, 2, 9);        //指定范围替换
```

5. 转列表: asList()

```
/*
        * 返回由指定数组支持的固定大小的列表，是一个Arrays的内部类，并没有实现集合的修改方法，底层还是数组，体现的是适配器模式，只是装换了插口，后台还是数组。故不能使用add/remove/clear，否则抛出UnsupportedOperationException异常。
        * （将返回的列表更改为“写入数组”。）该方法作为基于数组和基于集合的API之间的桥梁，与Collection.toArray()相结合 。
        * 返回的列表是可序列化的，并实现RandomAccess 。
        * 此方法还提供了一种方便的方式来创建一个初始化为包含几个元素的固定大小的列表如下：
   List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");
                    */
```

6. 转字符串 : toString()

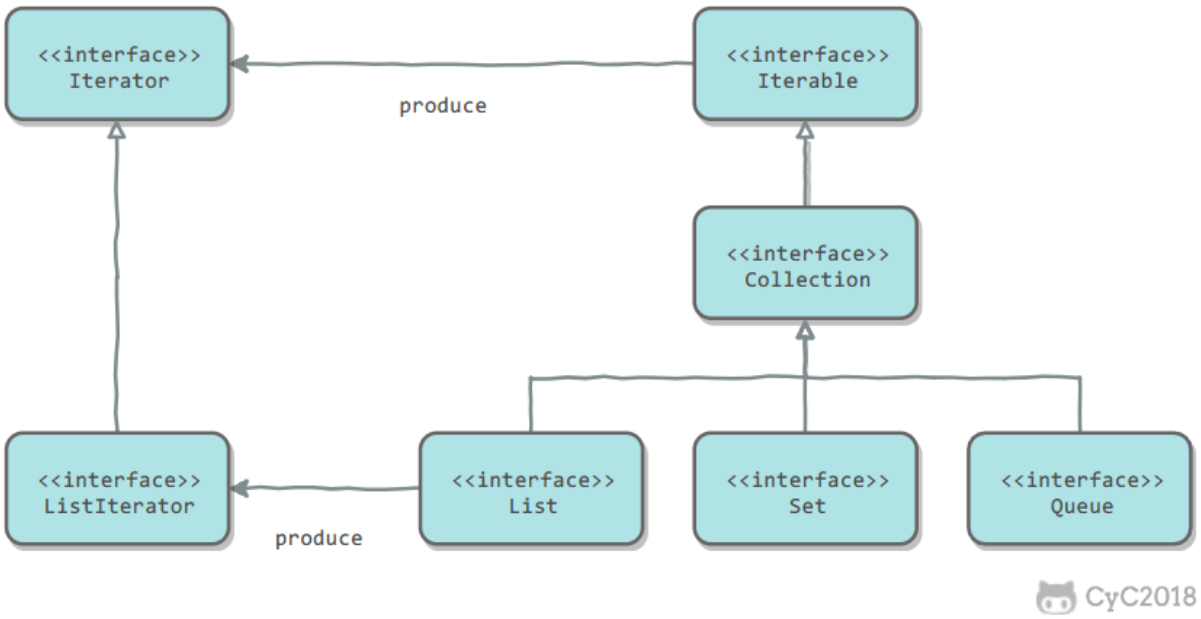
```
//返回指定数组的内容的字符串表示形式。
```

7. 复制: copyOf()

```
   // copyOf 方法实现数组复制,h为数组，6为复制的长度
        int[] h = { 1, 2, 3, 3, 3, 3, 6, 6, 6, };
        int i[] = Arrays.copyOf(h, 6);
     // copyOfRange将指定数组的指定范围复制到新数组中
        int j[] = Arrays.copyOfRange(h, 6, 11);
        System.out.println("Arrays.copyOfRange(h, 6, 11)：");
        // 输出结果66600(h数组只有9个元素这里是从索引6到索引11复制所以不足的就为0)
```

容器中的设计模式

迭代器模式



Collection 继承了 Iterable 接口，其中 iterator() 方法产生一个 Iterator 对象，通过这个对象就可以迭代遍历 Collection 中的元素。从 JDK 1.5 之后可以使用 foreach 方法来遍历实现了 Iterable 接口的聚合对象。

```
List<String> list = new ArrayList<>();
list.add("a");
list.add("b");
for (String item : list) {
    System.out.println(item);
}
```

适配器模式

java.util.Arrays#asList() 可以把数组类型转换为 List 类型。

```
@SafeVarargs
public static <T> List<T> asList(T... a)
```

应该注意的是 asList() 的参数为泛型的变长参数，不能使用基本类型数组作为参数，只能使用相应的包装类型数组。

```
Integer[] arr = {1, 2, 3};
List list = Arrays.asList(arr);
```

也可以使用以下方式调用 asList()：

```
List list = Arrays.asList(1, 2, 3);
```