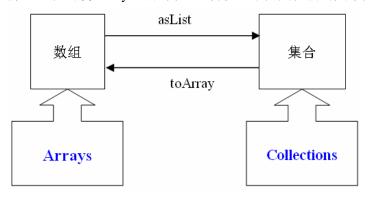
## Collections Framework 中的算法(之一)——综述

## 江苏 无锡 缪小东

在博客其它文章中提到 Java Collections Framework——java 中面向对象的数据结构的框架! 几乎任何学习 java 的朋友都要对此包(java.util)非常熟悉。

该框架主要包含接口、实现和算法三部分。在本博客关于 FailFast 机制的文章中纵向研究了 List 接口和 List 的实现。关于算法一点都没有提及。本系列主要讲述该框架中的算法!

Java Collections Framework 中的算法基本都包含在 Arrays 和 Collections 类中。这两个类有一个共同的特点: 1.构造方法都是私有的一一即不可以实例化、不可以继承; 2.其中的方法都是 static 的一一即静态的。因此他们完全是个工具类。他们两联系了整个 Collections Framework——不明白吗!! 任何 Collections Framework 中的实现都有一个 to Array 方法,该方法将任何数据结构转变为数组,转变为数组 后可以通过 Arrays 完成一系列的操作。Collections 类本身可以对任何数据结构进行一系列的操作。在 Arrays 中提供了 as List 方法,从而使 Arrays 可以向 List 转换。下面是他们的关系图!



关于数组和集合,他们之间是有较大的区别的:数组表示一组同一类型的"元素"!这些"元素"可以是基本数据类型,也可以是引用类型,数组最大的特点是:一旦创建数组,其容量是不可以变化的。而集合的特点是:一般只可以包含对象,并且集合中元素的数目是可以变化的!

下面是 Arrays 类的类图:

```
Arrays
                                                         (from util)
sort(arg0 : long[]) : void
Sort(arg0 : long[], arg1 : int, arg2 : int) : void
sort(arg0 : int[]) : void
♦sort(arg0 : int[], arg1 : int, arg2 : int) : void
Sort(arg0 : short[]) : void
sort(arg0 : short[], arg1 : int, arg2 : int) : void
sort(arg0 : char[]) : void
Sort(arg0: char[], arg1: int, arg2: int): void
♦sort(arg0 : byte[]) : void
♦sort(arg0: byte[], arg1: int, arg2: int): void
sort(arg0 : double[]) : void
sort(arg0 : double[], arg1 : int, arg2 : int) : void
♦sort(arg0 : float[]) : void
♦sort(arg0 : float[], arg1 : int, arg2 : int) : void
sort(arg0 : Logical View::java::lang::Object[]) : void
♦sort(arg0 : Logical View::java::lang::Object[], arg1 : int, arg2 : int) : void
sort(arg0 : Logical View::java::lang::Object[], arg1 : Comparator) : void
Sort(arg0 : Logical View::java::lang::Object[], arg1 : int, arg2 : int, arg3 : Comparator) : void
♥binarySearch(arg0 : long[], arg1 : long) : int
♦binarySearch(arg0 : int[], arg1 : int) : int
binarySearch(arg0 : short[], arg1 : short) : int
binarySearch(arg0 : char[], arg1 : char) : int
♦binarySearch(arg0 : byte[], arg1 : byte) : int
♦binarySearch(arg0 : double[], arg1 : double) : int
❤binarySearch(arg0:float[],arg1:float):int
♦binarySearch(arg0 : Logical View::java::lang::Object[], arg1 : Object) : int
♦binarySearch(arq0 : Logical View::java::lang::Object[], arq1 : Object, arq2 : Comparator) : int
>equals(arg0 : long[], arg1 : long[]) : boolean
♦equals(arg0 : int[], arg1 : int[]) : boolean
Ŷequals(argO : short[], arg1 : short[]) : boolean
Ŷequals(arg0 : char[], arg1 : char[]) : boolean
oequals(argO : byte[], arg1 : byte[]) : boolean
🎙 equals (arg0 : boolean[], arg1 : boolean[]) : boolean
♦equals(arg0 : double[], arg1 : double[]) : boolean
♦equals(arg0 : float[], arg1 : float[]) : boolean
♦equals(arg0 : Logical View::java::lang::Object[], arg1 : Logical View::java::lang::Object[]) : boolear
 fill(argU : long) |, arg1 : long) : void
❤fill(argO: long[], arg1: int, arg2: int, arg3: long): void
♦fill(arg0:int[], arg1:int):void
❤fill(arg0:int[], arg1:int, arg2:int, arg3:int):void
♦fill(arg0 : short[], arg1 : short) : void
fill(arg0 : short[], arg1 : int, arg2 : int, arg3 : short) : void
fill(arg0 : char[], arg1 : char) : void
❤fill(argO:char[], arg1:int, arg2:int, arg3:char):void
❤fill(argO: byte[], arg1: byte): void
❤fill(arg0: byte[], arg1: int, arg2: int, arg3: byte): void
fill(arg0 : boolean[], arg1 : boolean) : void
♥fill(argO: boolean[], arg1: int, arg2: int, arg3: boolean): void
fill(arg0 : double[], arg1 : double) : void
fill(arg0 : double[], arg1 : int, arg2 : int, arg3 : double) : void
❤fill(argO:float[], arg1:float):void
❤fill(argO:float[], arg1:int, arg2:int, arg3:float):void
♥fill(arg0: Logical View::java::lang::Object[], arg1: Object): void
ofill(arg0: Logical View∷java∷lang::Object[], arg1: int, arg2: int, arg3: Object): void
 ▼asList(argU : Logical View::java::lang::Object[]) : List
```

从上图可以看出 Arrays 类主要包括一些排序、查找、相等和赋值的操作,当然啦还包含一个 asList 的方法。

下图为 Collections 类的类图:

## Collections (from util) ♦sort(arq0 : List) : void sort(arg0 : List, arg1 : Comparator) : void ♦binarySearch(arg0 : List, arg1 : Object) : int ❤binarySearch(arg0 : List, arg1 : Object, arg2 : Comparator) : int ❤reverse(argO : List) : void ♦shuffle(arg0 : List) : void ♦shuffle(arg0 : List, arg1 : Random) : void ♦swap(arg0 : List, arg1 : int, arg2 : int) : void Mill(arg0 : List, arg1 : Object) : void ♦copy(arα0: List, arα1: List): void min(arqD: Collection): Object min(arg0 : Collection, arg1 : Comparator) : Object max(arq0 : Collection) : Object max(arg0 : Collection, arg1 : Comparator) : Object rotate(argU : List, arg1 : int) : void ŶreplaceAll(arg0 : List, arg1 : Object, arg2 : Object) : boolean ondexOfSubList(arg0 : List, arg1 : List) : int NastIndexOfSubList(arg0 : List, arg1 : List) : int onmodifiableCollection(arg0 : Collection) : Collection onmodifiableSet(arg0 : Set) : Set ♥unmodifiableSortedSet(arq0 : SortedSet) : SortedSet ŶunmodifiableList(arq0 : List) : List ounmodifiableMap(arg0 : Map) : Map ounmodifiableSortedMap(arg0 : SortedMap) : SortedMap synchronizedCollection(argU : Collection) : Collection ♣synchronizedCollection(arg0 : Collection, arg1 : Object) : Collection SynchronizedSet(arg0 : Set) : Set ♦synchronizedSet(arg0 : Set, arg1 : Object) : Set synchronizedSortedSet(arg0 : SortedSet) : SortedSet synchronizedList(arg0 : List) : List ♣synchronizedList(arg0 : List, arg1 : Object) : List 🗣synchronizedMap(argO : Map) : Map synchronizedSortedMap(arg0 : SortedMap) : SortedMap ♥singleton(arg0 : Object) : Set ♦singletonList(arg0 : Object) : List ♦singletonMap(arg0 : Object, arg1 : Object) : Map ❤nCopies(arg0 : int, arg1 : Object) : List ♦reverseOrder() : Comparator enumeration(arg0 : Collection) : Enumeration Nist(arq0: Enumeration): ArrayList

Collections 类中主要包含:排序、查找、翻转、填充、拷贝、最大值、最小值、查序等等方法。还包含一些引用相关的方法如创建某些不可变的集合、同步的集合、单例的集合,以上三类方法是典型的Decorator模式的应用!

下面的文章会仔细介绍他们的源代码!

## 更多精彩请关注:

http://blog.163.com/miaoxiaodong78/