Java 数据结构 Java 泛型

Java 集合框架

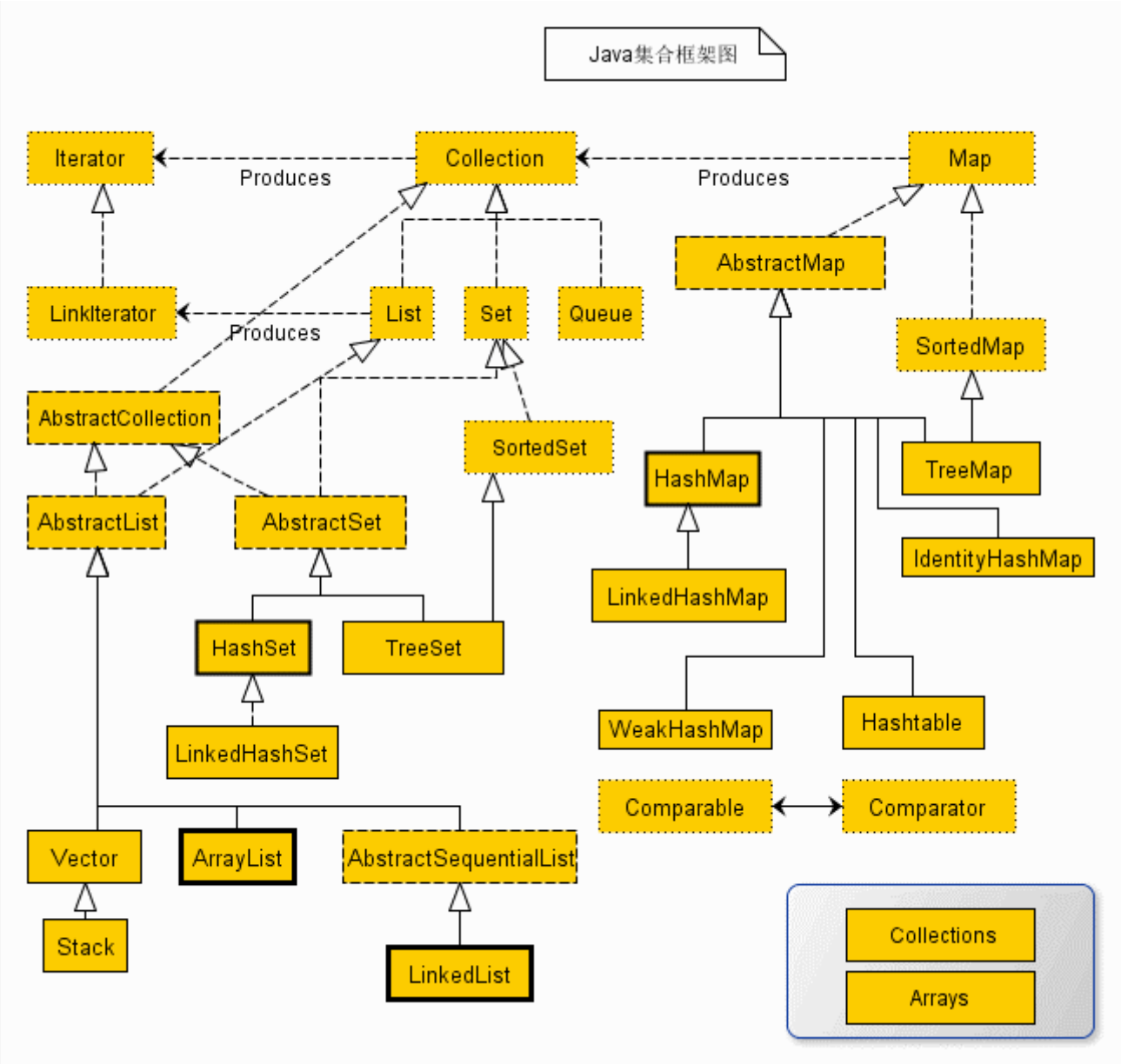
早在 Java 2 中之前，Java 就提供了特设类。比如：Dictionary, Vector, Stack, 和 Properties 这些类用来存储和操作对象组。

虽然这些类都非常有用，但是它们缺少一个核心的，统一的主题。由于这个原因，使用 Vector 类的方式和使用 Properties 类的方式有着很大不同。

集合框架被设计成要满足以下几个目标。

- 该框架必须是高性能的。基本集合（动态数组，链表，树，哈希表）的实现也必须是高效的。
- 该框架允许不同类型的集合，以类似的方式工作，具有高度的互操作性。
- 对一个集合的扩展和适应必须是简单的。

为此，整个集合框架就围绕一组标准接口而设计。你可以直接使用这些接口的标准实现，诸如： **LinkedList**, **HashSet**, 和 **TreeSet** 等,除此之外你也可以通过这些接口实现自己的集合。



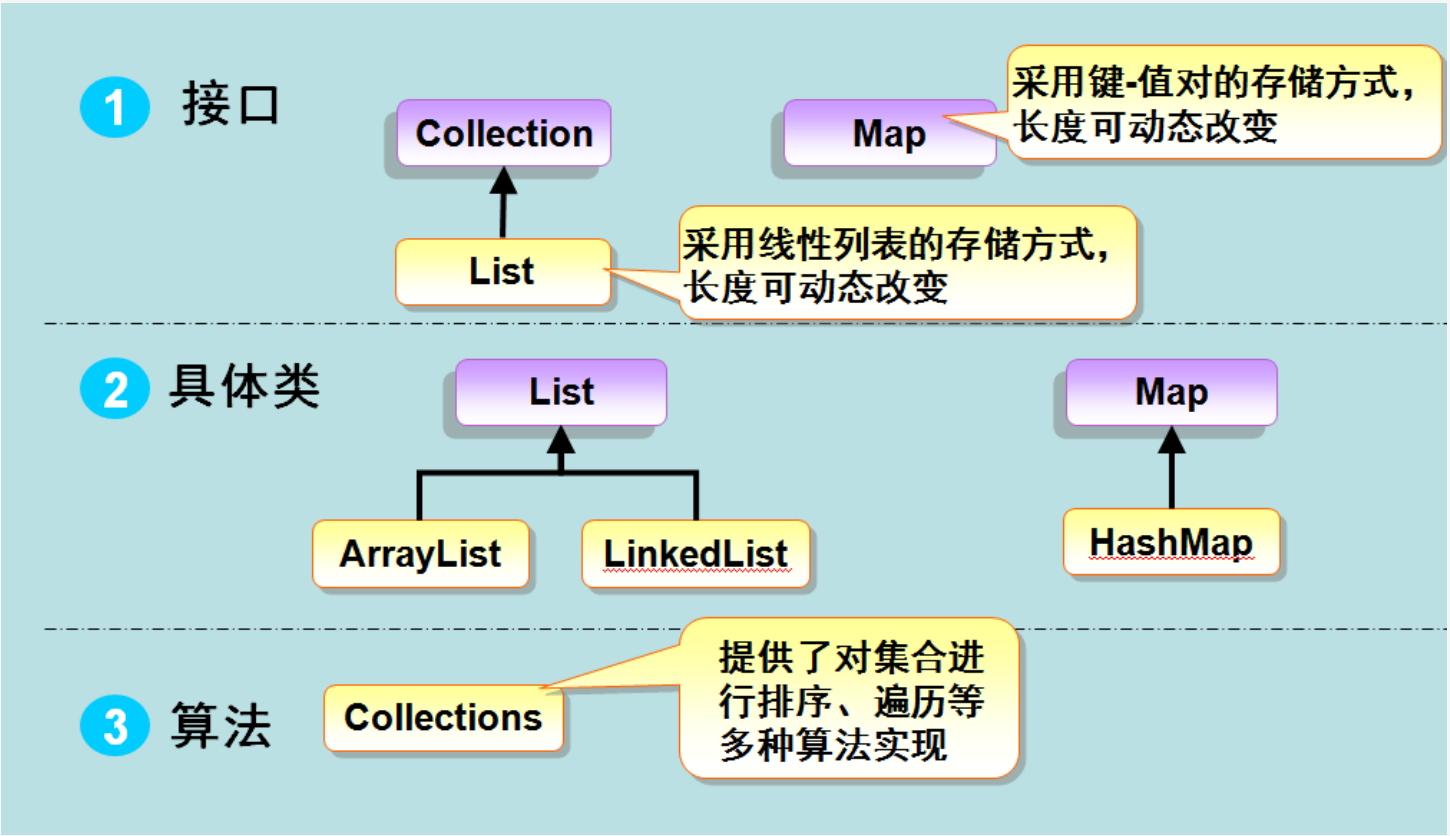
从上面的集合框架图可以看到，Java 集合框架主要包括两种类型的容器，一种是集合（Collection），存储一个元素集合，另一种是图（Map），存储键/值对映射。Collection 接口又有 3 种子类型，List、Set 和 Queue，再下面是一些抽象类，最后是具体实现类，常用的有 ArrayList、LinkedList、HashSet、LinkedHashSet、HashMap、LinkedHashMap 等等。

集合框架是一个用来代表和操纵集合的统一架构。所有的集合框架都包含如下内容：

- **接口：**是代表集合的抽象数据类型。例如 Collection、List、Set、Map 等。之所以定义多个接口，是为了以不同的方式操作集合对象
- **实现（类）：**是集合接口的具体实现。从本质上讲，它们是可重复使用的数据结构，例如：ArrayList、LinkedList、HashSet、HashMap。
- **算法：**是实现集合接口的对象里的方法执行的一些有用的计算，例如：搜索和排序。这些算法被称为多态，那是因为相同的方法可以在相似的接口上有着不同的实现。

除了集合，该框架也定义了几个 Map 接口和类。Map 里存储的是键/值对。尽管 Map 不是集合，但是它们完全整合在集合中。

集合框架体系如图所示



Java 集合框架提供了一套性能优良，使用方便的接口和类，java集合框架位于java.util包中， 所以当使用集合框架的时候需要进行导包。

集合接口

集合框架定义了一些接口。本节提供了每个接口的概述：

序号	接口描述
1	Collection 接口 Collection 是最基本的集合接口，一个 Collection 代表一组 Object，即 Collection 的元素, Java不提供直接继承自Collection的类，只提供继承于的子接口(如List和set)。 Collection 接口存储一组不唯一，无序的对象。
2	List 接口 List接口是一个有序的 Collection，使用此接口能够精确的控制每个元素插入的位置，能够通过索引(元素在List中位置，类似于数组的下标)来访问List中的元素，第一个元素的索引为 0，而且允许有相同的元素。 List 接口存储一组不唯一，有序（插入顺序）的对象。
3	Set Set 具有与 Collection 完全一样的接口，只是行为上不同，Set 不保存重复的元素。 Set 接口存储一组唯一，无序的对象。
4	SortedSet

	继承于Set保存有序的集合。
5	Map Map 接口存储一组键值对象，提供key（键）到value（值）的映射。
6	Map.Entry 描述在一个Map中的一个元素（键/值对）。是一个Map的内部类。
7	SortedMap 继承于 Map，使 Key 保持在升序排列。
8	Enumeration 这是一个传统的接口和定义的方法，通过它可以枚举（一次获得一个）对象集合中的元素。这个传统接口已被迭代器取代。

Set和List的区别

- 1. Set 接口实例存储的是无序的，不重复的数据。List 接口实例存储的是有序的，可以重复的元素。
- 2. Set检索效率低下，删除和插入效率高，插入和删除不会引起元素位置改变 **<实现类有HashSet,TreeSet>**。
- 3. List和数组类似，可以动态增长，根据实际存储的数据的长度自动增长List的长度。查找元素效率高，插入删除效率低，因为会引起其他元素位置改变 **<实现类有ArrayList,LinkedList,Vector>** 。

集合实现类（集合类）

Java提供了一套实现了Collection接口的标准集合类。其中一些是具体类，这些类可以直接拿来使用，而另外一些是抽象类，提供了接口的部分实现。

标准集合类汇总于下表：

序号	类描述
1	**AbstractCollection** 实现了大部分的集合接口。
2	**AbstractList** 继承于AbstractCollection 并且实现了大部分List接口。
3	**AbstractSequentialList** 继承于 AbstractList ，提供了对数据元素的链式访问而不是随机访问。
4	LinkedList 该类实现了List接口，允许有null（空）元素。主要用于创建链表数据结构，该类没有同步方法，如果多个线程同时访问一个List，则必须自己实现访问同步，解决方法就是在创建List时候构造一个同步的List。例如：

	`Listlist=Collections.synchronizedList(newLinkedList(...));` LinkedList 查找效率低。
5	ArrayList 该类也是实现了List的接口，实现了可变大小的数组，随机访问和遍历元素时，提供更好的性能。该类也是非同步的,在多线程的情况下不要使用。ArrayList 增长当前长度的50%，插入删除效率低。
6	**AbstractSet** 继承于AbstractCollection 并且实现了大部分Set接口。
7	HashSet 该类实现了Set接口，不允许出现重复元素，不保证集合中元素的顺序，允许包含值为null的元素，但最多只能一个。
8	LinkedHashSet 具有可预知迭代顺序的 `Set` 接口的哈希表和链接列表实现。
9	TreeSet 该类实现了Set接口，可以实现排序等功能。
10	**AbstractMap** 实现了大部分的Map接口。
11	HashMap HashMap 是一个散列表，它存储的内容是键值对(key-value)映射。 该类实现了Map接口，根据键的HashCode值存储数据，具有很快的访问速度，最多允许一条记录的键为null，不支持线程同步。
12	TreeMap 继承了AbstractMap，并且使用一颗树。
13	WeakHashMap 继承AbstractMap类，使用弱密钥的哈希表。
14	LinkedHashMap 继承于HashMap，使用元素的自然顺序对元素进行排序.
15	IdentityHashMap 继承AbstractMap类，比较文档时使用引用相等。

在前面的教程中已经讨论通过java.util包中定义的类，如下所示：

序号	类描述
1	Vector 该类和ArrayList非常相似，但是该类是同步的，可以用在多线程的情况，该类允许设置默认的增长长度，默认扩容方式为原来的2倍。
2	Stack 栈是Vector的一个子类，它实现了一个标准的后进先出的栈。
3	Dictionary Dictionary 类是一个抽象类，用来存储键/值对，作用和Map类相似。
4	Hashtable Hashtable 是 Dictionary(字典) 类的子类，位于 java.util 包中。
5	Properties Properties 继承于 Hashtable，表示一个持久的属性集，属性列表中每个键及其对应值都是一个字符串。
6	BitSet 一个Bitset类创建一种特殊类型的数组来保存位值。BitSet中数组大小会随需要增加。

集合算法

集合框架定义了几种算法，可用于集合和映射。这些算法被定义为集合类的静态方法。

在尝试比较不兼容的类型时，一些方法能够抛出 ClassCastException异常。当试图修改一个不可修改的集合时，抛出UnsupportedOperationException异常。

集合定义三个静态的变量：EMPTY_SET，EMPTY_LIST，EMPTY_MAP的。这些变量都不可改变。

序号	算法描述
1	Collection Algorithms 这里是一个列表中的所有算法实现。

如何使用迭代器

通常情况下，你会希望遍历一个集合中的元素。例如，显示集合中的每个元素。

一般遍历数组都是采用for循环或者增强for，这两个方法也可以用在集合框架，但是还有一种方法是采用迭代器遍历集合框架，它是一个对象，实现了Iterator 接口或ListIterator接口。

迭代器，使你能够通过循环来得到或删除集合的元素。ListIterator 继承了Iterator，以允许双向遍历列表和修改元素。

序号	迭代器方法描述
1	使用 Java Iterator 这里通过实例列出Iterator和listIterator接口提供的所有方法。

遍历 ArrayList

实例

```
import java.util.*;
public class Test{
public static void main(String[] args) {
List<String> list=new ArrayList<String>();
list.add("Hello");
list.add("World");
list.add("HAHAHAHA");
//第一种遍历方法使用foreach遍历List
for (String str : list) { //也可以改写for(int i=0;i<list.size();i++)这种形式
System.out.println(str);
}
//第二种遍历，把链表变为数组相关的内容进行遍历
String[] strArray=new String[list.size()];
list.toArray(strArray);
for(int i=0;i<strArray.length;i++) //这里也可以改写为 foreach(String str:strArray)这种形式
{
System.out.println(strArray[i]);
}
//第三种遍历 使用迭代器进行相关遍历
Iterator<String> ite=list.iterator();
while(ite.hasNext())//判断下一个元素之后有值
{
System.out.println(ite.next());
}
}
}
```

解析：

三种方法都是用来遍历ArrayList集合，第三种方法是采用迭代器的方法，该方法可以不用担心在遍历的过程中会超出集合的长度。

遍历 Map

实例

```
import java.util.*;
public class Test{
public static void main(String[] args) {
Map<String, String> map = new HashMap<String, String>();
map.put("1", "value1");
map.put("2", "value2");
map.put("3", "value3");
//第一种：普遍使用，二次取值
System.out.println("通过Map.keySet遍历key和value：");
for (String key : map.keySet()) {
System.out.println("key= "+ key + " and value= " + map.get(key));
}
//第二种
System.out.println("通过Map.entrySet使用iterator遍历key和value：");
```

```
Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();
while (it.hasNext()) {
Map.Entry<String, String> entry = it.next();
System.out.println("key= " + entry.getKey() + " and value= " + entry.getValue());
}
//第三种：推荐，尤其是容量大时
System.out.println("通过Map.entrySet遍历key和value");
for (Map.Entry<String, String> entry : map.entrySet()) {
System.out.println("key= " + entry.getKey() + " and value= " + entry.getValue());
}
//第四种
System.out.println("通过Map.values()遍历所有的value，但不能遍历key");
for (String v : map.values()) {
System.out.println("value= " + v);
}
}
}
```

如何使用比较器

TreeSet和TreeMap的按照排序顺序来存储元素. 然而，这是通过比较器来精确定义按照什么样的排序顺序。

这个接口可以让我们以不同的方式来排序一个集合。

序号	比较器方法描述
1	使用 Java Comparator 这里通过实例列出Comparator接口提供的所有方法

总结

Java集合框架为程序员提供了预先包装的数据结构和算法来操纵他们。

集合是一个对象，可容纳其他对象的引用。集合接口声明对每一种类型的集合可以执行的操作。

集合框架的类和接口均在java.util包中。

任何对象加入集合类后，自动转变为Object类型，所以在取出的时候，需要进行强制类型转换。

← Java 数据结构 Java 泛型 →