Lecture4-2 集合

1. 集合的介绍 Collection

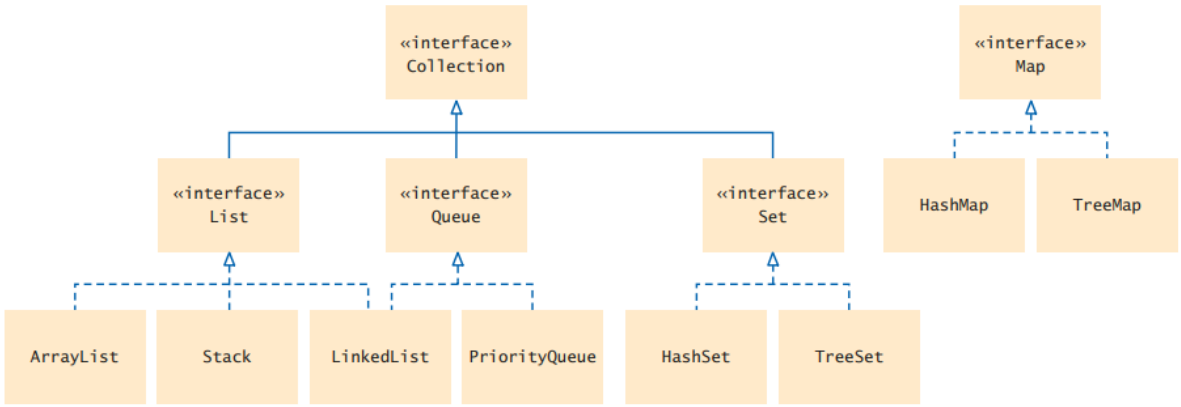
集合 Collection 是代表一组对象的对象（例如经典的ArrayList类）

集合框架是表示和操作集合的统一架构，允许独立于实现细节对集合进行操作

集合的好处

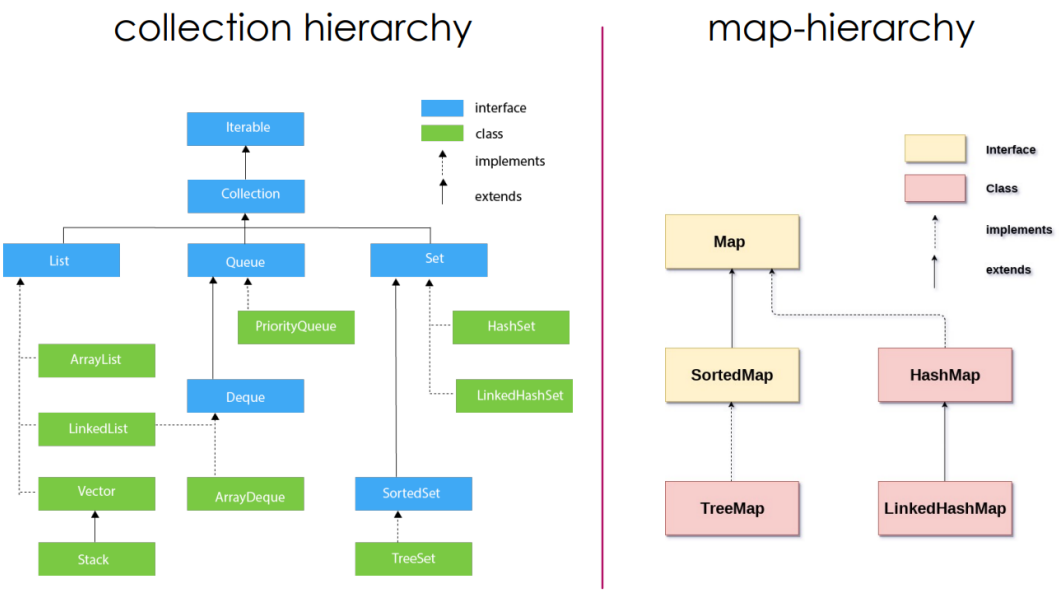
- 减少编程的工作：提供了数据结构和算法，你不需要自己实现
- 增加性能：提供高性能的数据结构和算法的实现，因为每个接口的各种实现是可互换的，所以可以通过切换实现来调优程序
- 提供不相关 API 之间的互操作性：建立一种公共语言来来回传递集合
- 减少学习 API 的工作：要求学习多个特定的集合
- 减少设计和实现 API 的工作：不需要自己取生成特定的集合 API
- 促进软件重用：为集合和算法提供标准接口

Java 集合的框架



集合的接口分成了两个组

- `java.util.Collection`
- `java.util.Map`



2. List

- 一个有序集合，它维护插入顺序
- 在显示列表内容时，它将按照元素插入到列表中的顺序显示元素
- List 允许重复元素

3. Set

- 一个无序集合，它不维护任何顺序
- 除了 `LinkedHashSet` ，它维护有顺序的 Set 实现（它以插入顺序维护元素）
- Set 不允许重复元素

4. 应该使用什么样的集合

- 接口：声明了哪些方法是有效的
- 实现类：不同操作有不同的效率

集合	实现类
`List`	`ArrayList`：数组实现，访问一个元素需要常量时间 $O(1)$，添加一个元素在最坏的情况下需要 $O(n)$ 时间 `LinkedList`：链表实现，访问和添加元素需要 $O(n)$ 的使劲按，`LinkedList` 比 `ArrayList` 需要更多的内存
`Set`	`HashSet`：提供常数时间性能，例如 $O(1)$ 添加，查找和删除 `TreeSet`：维持元素排序顺序，对于常见的操作，如添加、删除和包含，TreeSet提供了 $O(log(n))$ 时间保证 `LinkedHashSet`：维持元素的插入顺序，提供固定时间性能，如 $O(1)$ 的添加，查找和删除

`Map`	`HashMap`：为基本操作，如 get, put, containsKey, remove 等提供 $O(1)$ 时间性能，当时间性能很重要且不需要考虑 key 的顺序的话，使用 HashMap
	`LinkedHashMap`：基本操作，如 get, put, containsKey, remove 等提供 $O(1)$ 时间性能，当需要考虑元素插入顺序的时候使用 `LinkedHashMap`
	`TreeMap`：保证 $O(log(n))$ 时间成本的基本操作，如 get, put, containsKey, remove 等，当需要使用 key 的自然顺序或通过 Comparator 排序时，使用 `TreeMap`

5. hashCode() 和 equals()

- `hashCode()` 返回一个由哈希算法生成的整数值，算法来源于 `Object.hashCode()`
- 在Java应用程序的执行过程中，无论在同一对象上多次调用它，`hashCode()` 都必须一致地返回相同的值
- 调用 `equals(Object)` 方法时，如果两个对象相等，那么调用两个对象的 `hashCode()` 方法会产生相同的值
- 调用 `equals(Object)` 方法时，如果两个对象不相等，那么调用两个对象的 `hashCode()` 不一定会产生相同的值

==

- 基本类型：比较两个对象的值是否相同
- 引用类型：比较两个对象的地址是否相同

equals()

- 默认情况下与 `==` 相同
- 重写情况下
    - 先比较两个对象的 hashCode() 是否相同
        - 如果不相同返回 false
        - 如果相同进一步比较 equals() 内的定义

hashcode()

一种生成对象 hashcode 的方法，默认使用 `Object.hashCode()`，也可以自己重写