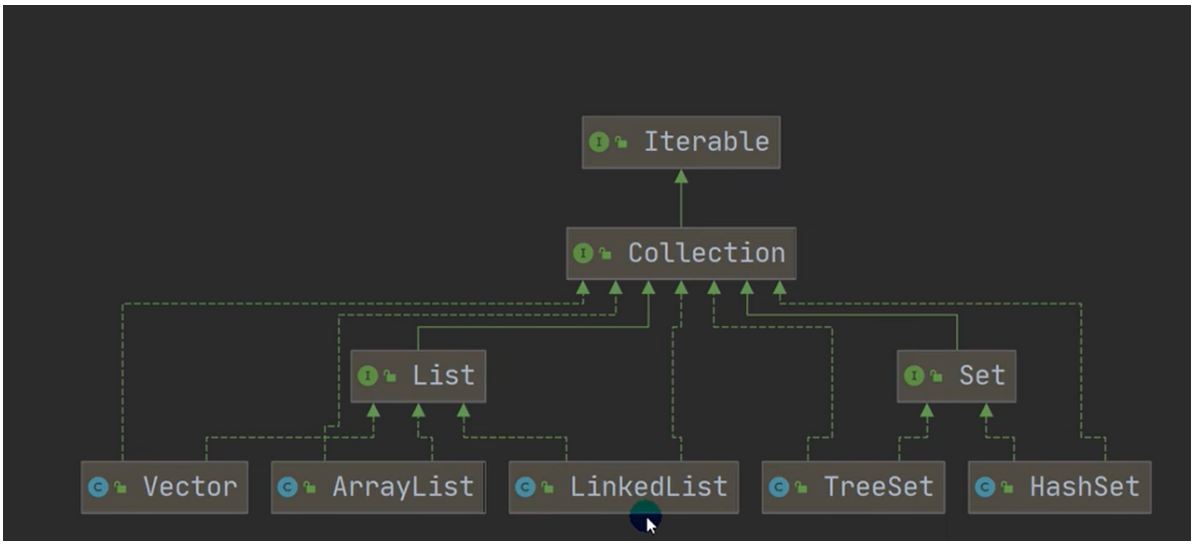
一、集合框架体系

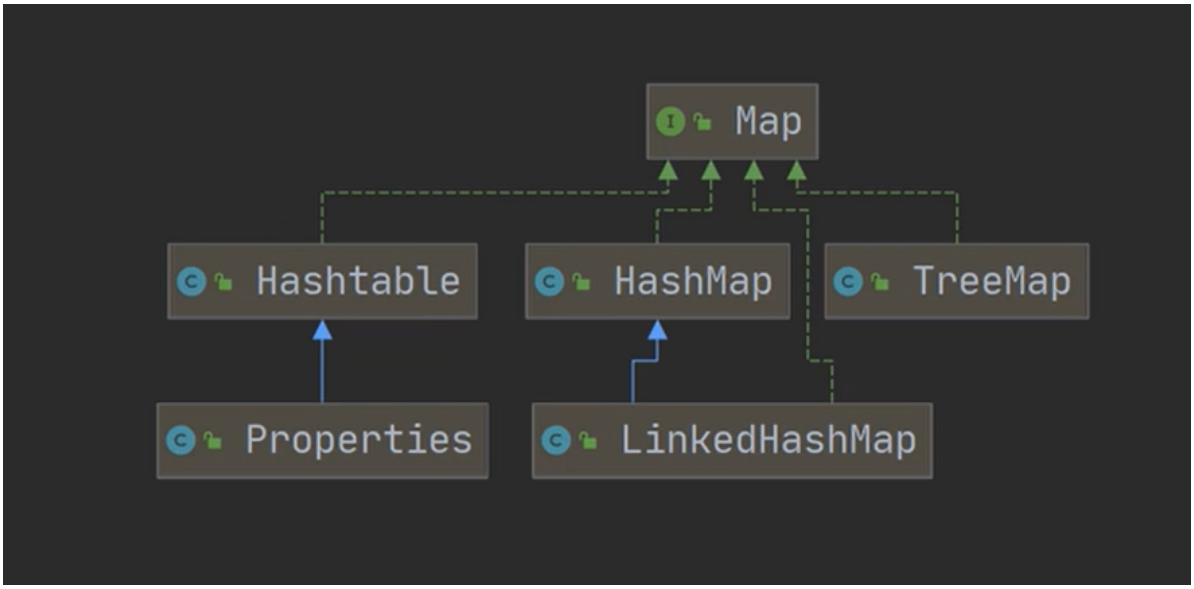
1、简介

> 集合可以动态保存任意多个对象，提供了一系列方便操作对象的方法（add、remove、set、get）

2、框架体系图-单列集合



3、框架体系图-双列集合



二、Collection

1、Collection接口实现类的特点

1.1、Collection实现子类可以存放多个元素，每个元素可以是Object
1.2、有些Collection的实现类，可以存放重复的元素，有些不可以
1.3、有些Collection的实现类，有些是有序的（List），有些不是有序（Set）
1.4、Collection接口没有直接的实现子类，是通过它的子接口Set和List来实现的

2、Collection接口常用方法

add：添加单个元素
remove：删除指定元素
contains：查找元素是否存在
size：获取元素个数
isEmpty：判断是否为空
clear：清空
addAll：添加多个元素
containsAll：查找多个元素是否都存在
removeAll：删除多个元素

3、Collection接口遍历元素-Iterator(迭代器)-简介

Iterator对象称为迭代器，用于遍历Collection集合中的元素。所有实现了Collection接口的集合类都有一个iterator()方法，用于返回一个实现了Iterator接口的对象，即可以返回一个迭代器。

4、Collection接口遍历元素-Iterator(迭代器)-原理

```
// 1、得到一个集合的迭代器
Iterator iterator = coll.iterator()
// 2、使用while循环配合hasNext方法遍历
while(iterator.hasNext()) {
    Object obj = iterator.next();
    System.out.println(obj);
}
// 3、退出while循环后，迭代器指向最后的元素，如果需要再次遍历，需要重置迭代器
iterator = coll.iterator();
```

5、Collection接口遍历元素-for循环增强-简介

增强for循环，可以代替iterator迭代器。特点：增强for就是简化版的iterator，本质一样，只能用于遍历集合或数组。

6、Collection接口遍历元素-for循环增强-语法

```
for(元素类型 元素名 : 集合名或数组名) {
    访问元素
}
```

7、List接口-简介

```
    List接口是Collection接口的子接口。List集合类中元素有序索引、且可重复。
```

8、List接口-常用方法

```
    8.1、在指定索引位置插入元素
        list.add(int index, Object ele);
    8.2、在指定索引位置插入多个元素
        list.addAll(int index, Collection ele);
    8.3、获取指定元素
        list.get(int index);
    8.4、返回obj在集合中首次出现的位置
        list.indexOf(Object obj);
    8.5、返回obj在当前集合中最后出现的位置
        list.lastIndexOf(Object obj);
    8.6、移除指定index位置的元素，并返回此元素
        list.remove(int index);
    8.7、设置指定index位置的元素为ele，相当于替换
        list.set(int index, Object ele);
    8.8、返回从fromIndex到toIndex位置的子集合
        list.subList(int formIndex, int toIndex);
```

9、List的三种遍历方式[ArrayList, LinkedList, Vector]

```
    9.1、iterator（快捷键：itit）
        Iterator iter = col.iterator();
        while(iter.hasNext()) {
            Object o = iter.next();
        }
    9.2、增强for循环（快捷键：I）
        for(Object o : col) {
        }
    9.3、普通for循环（快捷键：fori）
        for(int i = 0; i < list.size(); i++) {
            Object object = list.get(i);
            System.out.println(object);
        }
```

10、List-ArrayList-细节

```
    10.1、ArrayList 可以加入null，并且可以多个
    10.2、ArrayList 是由数组来实现数据存储的
    10.3、ArrayList 基本等同于Vector，除了ArrayList是线程不安全（执行效率高），在多线程情况下，不建议使用ArrayList
```

11、List-ArrayList-底层操作机制

```
    11.1、ArrayList中维护了一个Object类型的数组elementData
        // transient 表示瞬间、短暂的，表示该属性不会被序列化
        transient Object[] elementData;
    11.2、当创建ArrayList对象时，如果使用的是无参构造器，则初始elementData容量为0，第1次添加元素时，扩容elementData为10，如果需要再次扩容，则扩容elementData为1.5倍
    11.3、如果使用的是指定大小的构造器，则初始elementData容量为指定大小，如果需要扩容，则直接扩容elementData为1.5倍
```

```
# 无参构造器-扩容流程
    1) elementData数组 = {}，系统默认创建了一个空数组
    2) list.add，先确定是否要扩容，然后执行赋值
    3) private void ensureCapacityInternal(int minCapacity)，确定minCapacity为10
    4) modCount++，记录当前集合修改的次数，防止多线程问题
    5) 如果elementData的大小不够，就调用grow()扩容
    6) 使用扩容机制来确定要扩容到多大
    7) 第一次newCapacity = 10
    8) 第二次及以后按照1.5倍扩容
    9) 扩容使用 Array.copyOf()，用于保留原来的数据
```

```
# 有参构造器-扩容流程
    1) this.elementData = new Object[capacity]，创建一个指定大小elementData数组
    2) 第一次扩容，就按照elementData的1.5倍扩容
    3) 整个执行流程和无参构造器一样
```

12、List-Vector-简介

```
    Vector底层也是一个对象数组，protected Object[] elementData;
    Vector是线程同步的，即线程安全，Vector类的操作方法都带有synchronized
```

13、List-Vector VS ArrayList

	底层结构	版本	线程安全（同步）效率	扩容倍数
ArrayList	可变数组	jdk 1.2	不安全，效率高	如果有参构造1.5倍扩容； 如果无参：默认10，从第二次开始按1.5倍扩容
Vector	可变数组 Object[]	jdk 1.0	安全，效率不高	如果有参构造2倍扩容； 如果无参：默认10，从第二次开始按2倍扩容

14、List-ArrayList VS LinkedList

	底层结构	增删效率	改查效率
ArrayList	可变数组	较低 数组扩容	较高

	底层结构	增删效率	改查效率
LinkedList	双向链表	较高 链表追加	较低

如何选择？
1）如果改查操作多，选择ArrayList
2）如果增删操作多，选择LinkedList
3）一般来说，在程序中，80%-90%都是查询，因此大部分情况下会选择ArrayList

15、Set接口-简介

Set接口是无序（添加和取出的顺序不一致）且不允许包含重复元素（最多只有一个null）的

16、Set接口-常用方法

add：添加单个元素
remove：删除指定元素
contains：查找元素是否存在
size：获取元素个数
isEmpty：判断是否为空
clear：清空
addAll：添加多个元素
containsAll：查找多个元素是否都存在
removeAll：删除多个元素

17、Set接口-遍历方式

17.1、迭代器
17.2、增强for

18、Set接口-HashSet-全面说明

18.1、HashSet实现了Set接口
18.2、HashSet本质是HashMap
18.3、可以存放null，但只能有一个
18.4、HashSet不保证元素是有序的，取决于hash后，再确定索引的结果
18.5、不能有重复元素/对象

19、Set接口-HashSet-添加元素原理

19.1、向HaseSet中添加元素时，会先获得该数据的hash值，然后转成索引
19.2、找到存储数据表，查看该索引是否已经存在元素
19.3、如果没有，直接加入
19.4、如果有，通过equals（String有重写equals 注意区分）比较，只要与该链表中任何一个元素相同，放弃添加；如果不同，放在该链表后面
19.5、如果一条链表的元素达到8个，并且整个表的大小大于64，就会进行树化（红黑树）

20、Set接口-HashSet-扩容和树化原理

20.1、HashSet底层是HashMap，第一次添加时，table数组扩容到16，临界值（threshold）是16*加载因子（loadFactor）是0.75 = 12

20.2、如果table数组使用到了临界值12，就会扩容到16*2=32，新的临界值就是32*0.75=24，以此类推

20.3、在JAVA8中，如果一条链表的元素个数到达 TREEIFY_THRESHOLD(默认为8)，并且table的大小 >= MIN_TREEIFY_CAPACITY(默认64)，就会进行树化（红黑树），否则仍然采用数组扩容机制

21、Set接口-LinkedHashSet-全面说明

21.1、LinkedHashSet是HashSet的子类

21.2、LinkedHashSet底层是一个LinkedHashMap，底层维护了一个数组 + 双向链表

21.3、LinkedHashSet根据元素的hashCode值来决定元素的存储位置，同时使用链表维护元素的次序，是的元素看起来是以插入顺序保存的

21.4、LinkedHashSet不允许添加重复元素

三、Map

1、详解

1.1、Map和Collection并列存在。用于保存具有映射关系的数据：Key-Value

1.2、Map中的key和value可以是任何引用类型的数据，会封装到HashMap$Node对象中

1.3、Map中的key不允许重复

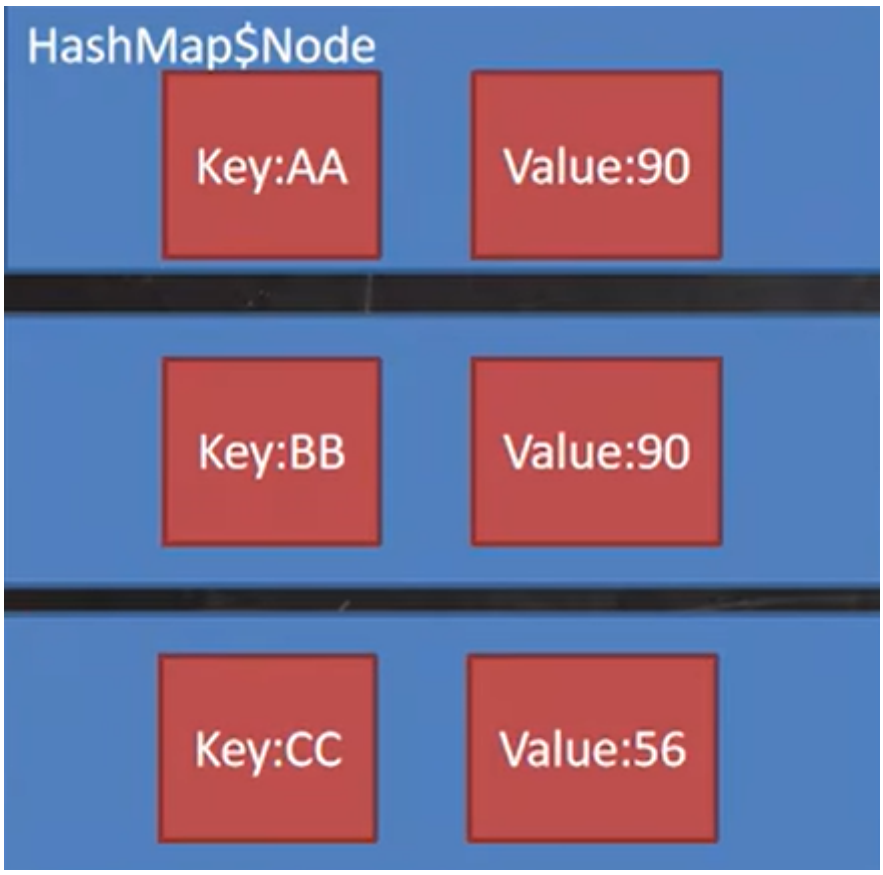
1.4、Map中的value可以重复

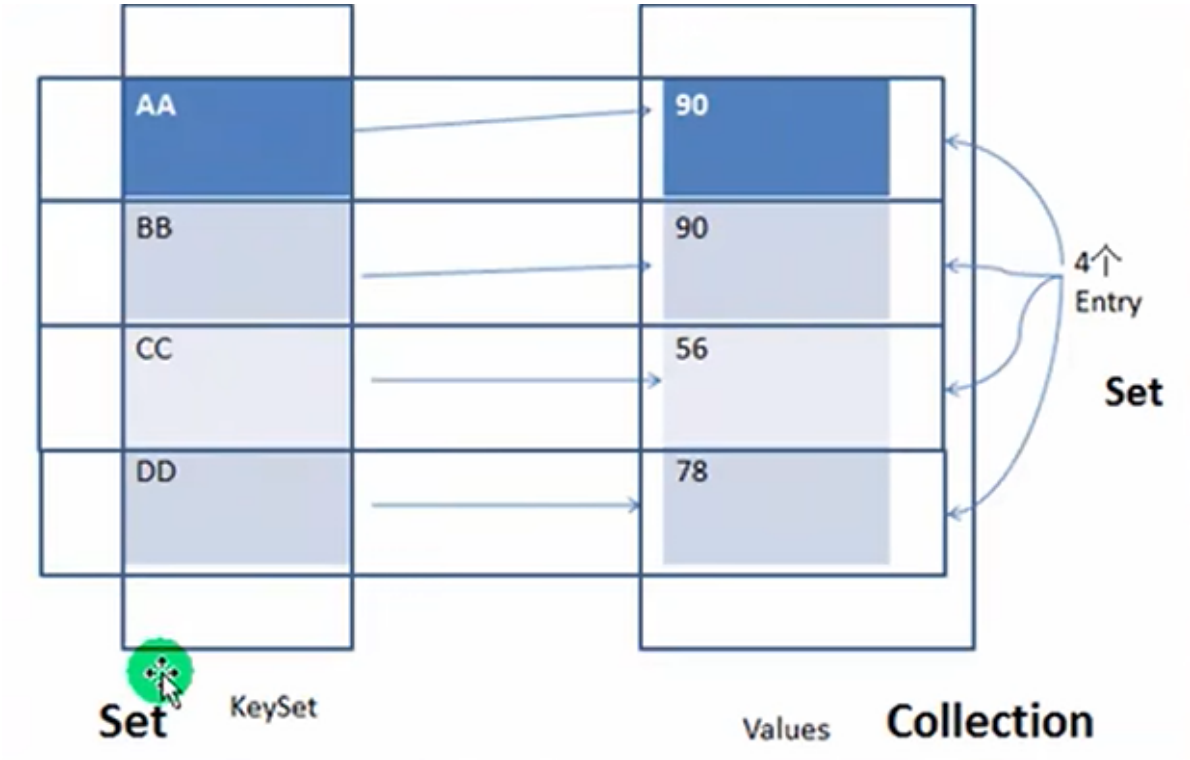
1.5、MAp的key可以为null，value也可以为null

1.6、常用String类作为Map的key

1.7、key和value之间存在单向一对一关系，即通过指定的key总能找到对应的value

1.8、一对 K-V 是放在一个HashMap$Node中的，又因为Node实现了Entry接口，也可以说一对K-V就是一个Entry





2、常用方法

```
1) put          添加
2) remove       根据键删除映射关系
3) get          根据键获取值
4) size         获取元素个数
5) isEmpty      判断个数是否为0
6) clear        清楚
7) containsKey  查找键是否存在
```

3、遍历方法

```
1) containsKey   查找键是否存在
2) keySet        获取所有的键
3) entrySet      获取所有关系
4) values        获取所有值
```

4、HashMap-总结

```
1) Map接口的常用实现类：HashMap、Hashtable、Properties
2) HashMap是Map接口使用频率最高的实现类
3) HashMap是以 key-value 对的方式来存储数据（HashMap$Node类型）
4) key不能重复，但是值可以重复，允许使用null键和null值
5) 如果添加相同的key，则会覆盖原来的 key-value，等同于修改
6) 与HashSet一样，不保证映射的顺序，因为底层是以hash表的方式来存储的
7) HashMap没有实现同步，因此是线程不安全的，方法没有做同步互斥的操作，没有synchronized
```

5、HashMap-原理

```
1) HashMap底层维护了Node类型的数组table，默认为null
2) 当创建对象时，将加载因子(loadfactor)初始化为0.75
3) 当添加key-value时，通过key的哈希值得到再table的索引。
   - 然后判断该索引处是否有元素，如果没有元素直接添加；
   - 如果该索引处有元素，继续判断该元素的key和准备加入的key是否相等，如果相等，则直接替换value，如果不相等需要判断是树结构还是链表结构，做出相应处理。
   - 如果添加时发现容量不够，则需要扩容。
4) 第一次添加，则需要扩容table容量为16，临界值(threshold)为12(16*0.75)
5) 以后再扩容，则需要扩容table容量为原来的两倍，临界值为原来的2倍，即24，以此类推
6) 在Java8中，如果一条链表的元素个数超过 TREEIFY_THRESHOLD(默认为8)，并且table的大小>= MIN_TREEIFY_CAPACITY(默认64)，就会进行树化（红黑树）
```

6、HashTable-总结

```
1) 存放的元素是键值对：即 k-v
2) HashTable的键和值都不能为null
3) HashTable使用方法基本上和HashMap一致
4) HashTable是线程安全的，HashMap是线程不安全的
```

7、Hashtable VS HashMap

	版本	线程安全（同步）	效率	允许null键null值
HashMap	1.2	不安全	高	可以
Hashtable	1.0	安全	较低	不可以

8、Properties-总结

1）Properties类继承自Hashtable类并且实现了Map接口，也是使用一种键值对的形式来保存数据
2）它的使用特点和Hashtable类似
3）Properties 还可以用于从 xx.properties文件中，加载数据到Properties类对象，并进行读取和修改
4）说明：工作后，xxx.properties文件通常作为配置文件

9、如何选择集合实现类

- 选择什么集合实现类，主要取决于业务操作特点，然后根据集合实现类特性进行选择
1）先判断存储类型（一组对象[单列] 或 一组键值对[双列]）
2）一组对象：Collection接口
允许重复：List
增删多：LinkedList[底层维护了一个双向链表]
改查多：ArrayList[底层维护Object类型的可变数组]
不允许重复：Set
无序：HashSet[底层是HashMap，维护了一个哈希表 即（数组+链表+红黑树）]
排序：TreeSet
插入和取出顺序一致：LinkedHashSet，维护数组+双向链表
3）一组键值对：Map
键无序：HashMap[底层是：哈希表，数组+链表+红黑树]
键排序：TreeMap
键插入和取出顺序一致：LinkedHashMap
读取文件：Properties

四、Collections

1、简介

Collections是一个操作Set、List和Map等集合的工具类，提供了一些列静态的方法对集合元素进行排序、查询和修改等操作。

2、排序方法

1）reverse(List)：反转List中元素的顺序
2）shuffle(List)：对List集合元素进行随机排序
3）sort(List)：根据元素的自然顺序对指定List集合元素按升序排序
4）sort(List, Comparator)：根据指定的Comparator产生的顺序对List集合元素进行排序
5）swap(List, int, int)：将指定list集合中的两个索引对应的元素进行交换