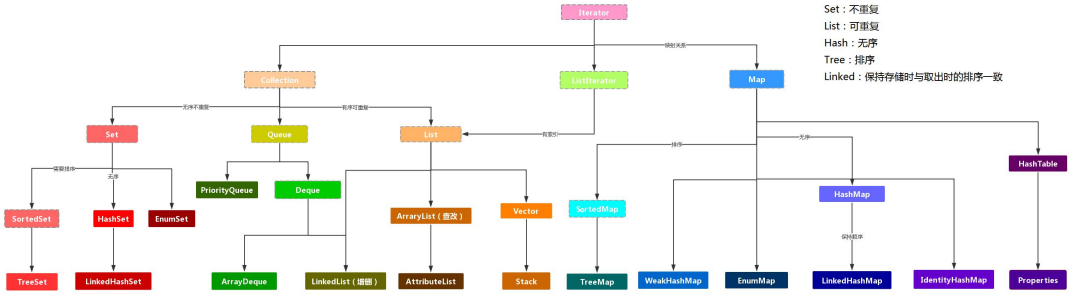
目录

1. 列举 java 的集合以及集合之间的继承关系



2. 集合类以及集合框架

集合类：

集合类存放于 java.util 包中。

集合类存放的都是对象的引用，而非对象本身，出于表达上的便利，我们称集合中的对象就是指集合中对象的引用（reference）。

集合类型主要有 3 种：set(集）、list(列表）和 map(映射)。

总的说来，Java API 中所用的集合类，都是实现了 Collection 接口，他的一个类继承结构如下：

Collection<--List<--Vector

Collection<--List<--ArrayList

Collection<--List<--LinkedList

Collection<--Set<--HashSet

Collection<--Set<--HashSet<--LinkedHashSet

Collection<--Set<--SortedSet<--TreeSet

集合框架：（就是 List，map，Collection）

集合框架是为表示和操作集合而规定的一种统一的标准的体系结构。任何集合框架都包含三大块内容：对外的接口、接口的实现和对集合运算的算法。

1）对外的接口：集合的抽象数据结构。接口允许我们独立地操纵集合而不用考虑集合的具体实现。

2）接口的实现：接口的具体实现类。从本质上来讲，它们是可重用的数据结构。

3）集合运算算法 ：在实现了集合接口的对象上执行有用的计算，比如排序和搜索，的方法。算法是多态的，同名的方法可以被任何合适的接口实现类调用，从本质上来讲，算法是可重用的功能 。

3. 容器类介绍以及之间的区别

容器类估计很多人没听这个词，Java 容器主要可以划分为 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）

Collection 是 List 和 Set 两个接口的基接口

List 在 Collection 之上增加了"有序"

Set 在 Collection 之上增加了"唯一"

ArrayList 是实现 List 的类...所以他是有序的。它里边存放的元素在排列上存在一定的先后顺序,是采用数组存放元素 。

List LinkedList 采用的则是链表。

Collection 和 Map 接口之间的主要区别在于：

Collection 中存储了一组对象，而 Map 存储关键字/值对(在 Map 对象中，每一个关键字最多有一个关联的值)。

Map:不能包括两个相同的键，一个键最多能绑定一个值。null 可以作为键，这样的键只有一个；可以有一个或多个键所对应的 值为 null。当 get()方法返回 null 值时，即可以表示 Map 中没有该键，也可以表示该键所对应的值为 null。因此，在 Map 中不能由 get()方法来判断 Map 中是否存在某个键，而应该用 containsKey()方法来判断。

继承 Map 的类有：HashMap，HashTable

HashMap：Map 的实现类，缺省情况下是非同步的，可以通过 Map Collections.synchronizedMap(Map m)来达到线程同步

HashTable：Dictionary 的子类，确省是线程同步的。不允许关键字或值为 null

4. List,Set,Map 的区别

List,Set 都是继承自 Collection 接口，Map 则不是

1. List 特点：元素有放入顺序，元素可重复 ，Set 特点：元素无放入顺序，元素不可重复，重复元素会覆盖掉，（注意：元素虽然无放入顺序，但是元素在 set 中的位置是有该元素的 HashCode 决定的，其位置其实是固定的，加入 Set 的 Object 必须定义 equals()方法，另外 list 支持 for 循环，也就是通过下标来遍历，也可以用迭代器，但是 set 只能用迭代，因为他无序，无法用下标来取得想要的值。）

2. Set 和 List 对比：

1) Set：检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。
2) List：和数组类似，List 可以动态增长，查找元素效率高，插入删除元素效率低，因为会引起其他元素位置改变。

3. Map 适合储存键值对的数据

4. 线程安全集合类与非线程安全集合类

非线程安全的：

LinkedList、ArrayList、HashSet、HashMap、StringBuilder

线程安全的：

Vector、HashTable、StringBuffer

5. List 和 Set 和 Map 的实现方式以及存储方式

List：

常用实现方式有：ArrayList 和 LinkedList

ArrayList 的存储方式：数组，查询快

LinkedList 的存储方式：链表，插入，删除快

Set：

常用实现方式有：HashSet 和 TreeSet

HashSet 的存储方式：哈希码算法，加入的对象需要实现 hashcode（）方法，快速查找元素

TreeSet 的存储方式：按序存放，想要有序就要实现 Comparable 接口

附加：

集合框架提供了 2 个实用类：collections（排序，复制、查找）和 Arrays 对数组进行（排序，复制、查找）

Map：

常用实现方式有：HashMap 和 TreeMap

HashMap 的存储方式：哈希码算法，快速查找键值

TreeMap 存储方式：对键按序存放

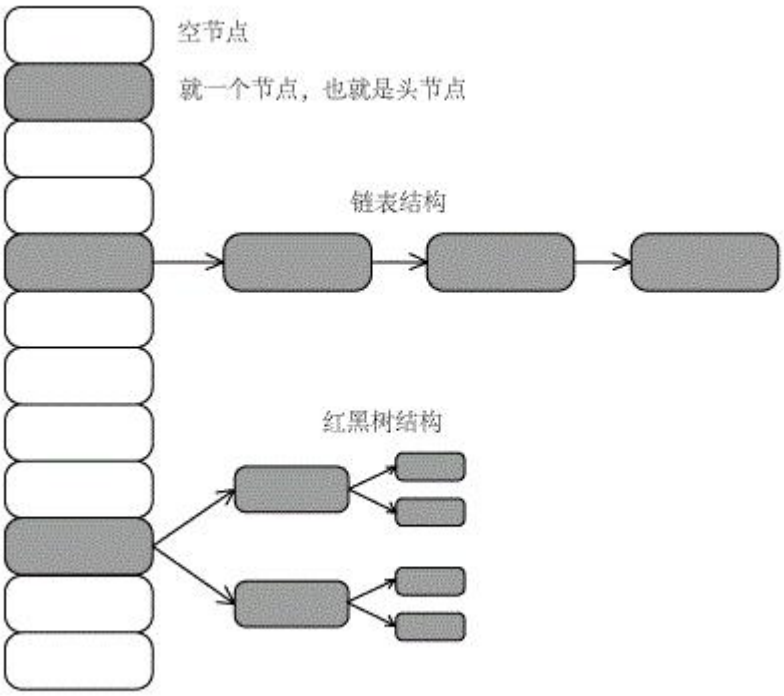
6. HashMap 的实现原理

HashMap 实际上是数组+红黑叔+链表的结合体

如果数组该位置上已经存放有其他元素了，那么在这个位置上的元素将以链表/红黑树的形式存放。

如果数组该位置上没有元素，就直接将该元素放到此数组中的该位置上。

如果数组该位置上有元素，判断是否是红黑树结果，如果是，则用红黑树插入法插入。如果不是红黑树，那就是链表，加入到链表最后（链表大小超过 8 要转换成红黑树结构）



7. HashMap 数据结构？

HashMap 的数据结构

数据结构中有数组+链表+红黑树来实现对数据的存储，但这两者各有利弊。

数组：

数组存储区间是连续的，占用内存严重，故空间复杂度大。但数组的二分查找时间复杂度小，为 O(1)；

数组特点：寻址容易，插入和删除困难；

链表：

链表存储区间离散，占用内存比较宽松，故空间复杂度很小，但时间复杂度很大，达O（N）。

链表特点：寻址困难，插入和删除容易。

红黑树：

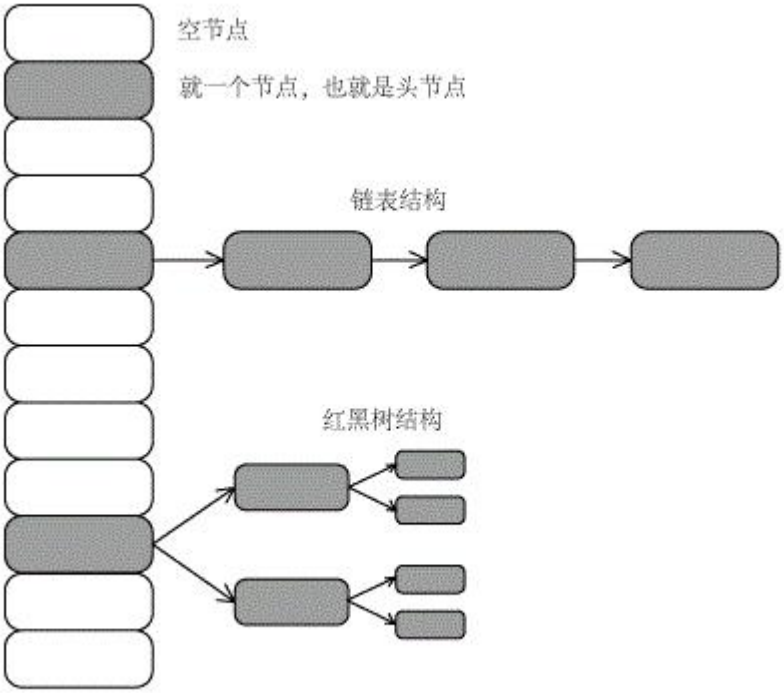
红黑树本质上是一种二叉查找树，但它在二叉查找树的基础上额外添加了一个标记（颜色），同时具有一定的规则。这些规则使红黑树保证了一种平衡，插入、删除、查找的最坏时间复杂度都为O(logn)。

红黑树特点：红黑树不追求"完全平衡"，它只要求部分达到平衡，但是提出了为节点增加颜色，红黑是用非严格的平衡来换取增删节点时候旋转次数的降低，任何不平衡都会在三次旋转之内解决

哈希表

那么我们综合两者的特性，做出一种寻址容易，插入删除也容易的数据结构，这就是哈希表。

哈希表（(Hash table）既满足了数据的查找方便，同时不占用太多的内存空间，使用也十分方便。



哈希表是由**数组+链表+红黑树**组成的，一个长度为 16 的数组中，每个元素存储的是一个链表的头结点。那么这些元素是按照什么样的规则存储到数组中呢。一般情况是通过(n - 1) & hash 获得，也就是元素的 key 的哈希值对数组长度**取模**得到。其中 hash 代码如下：

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

HashMap 其实也是一个线性的数组实现的，所以可以理解为其存储数据的容器就是一个线性数组。这可能让我们很不解，一个线性的数组怎么实现按键值对来存取数据呢？这里 HashMap 有做一些处

理。

首先 HashMap 里面实现一个静态内部类 Entry，其重要的属性有 key，value，next，从属性 key,value 我们就能很明显的看出来 Entry 就是 HashMap 键值对实现的一个基础 bean，我们上面说到 HashMap 的基础就是一个线性数组，这个数组就是 Entry[ ]，Map 里面的内容都保存在 Entry[ ]里面。

```
/**
* The table, resized as necessary. Length MUST Always be a power of two.
*/
    transient Entry[] table;
```

8. HashMap 源码理解

9. HashMap 如何 put 数据（从 HashMap 源码角度讲解）？

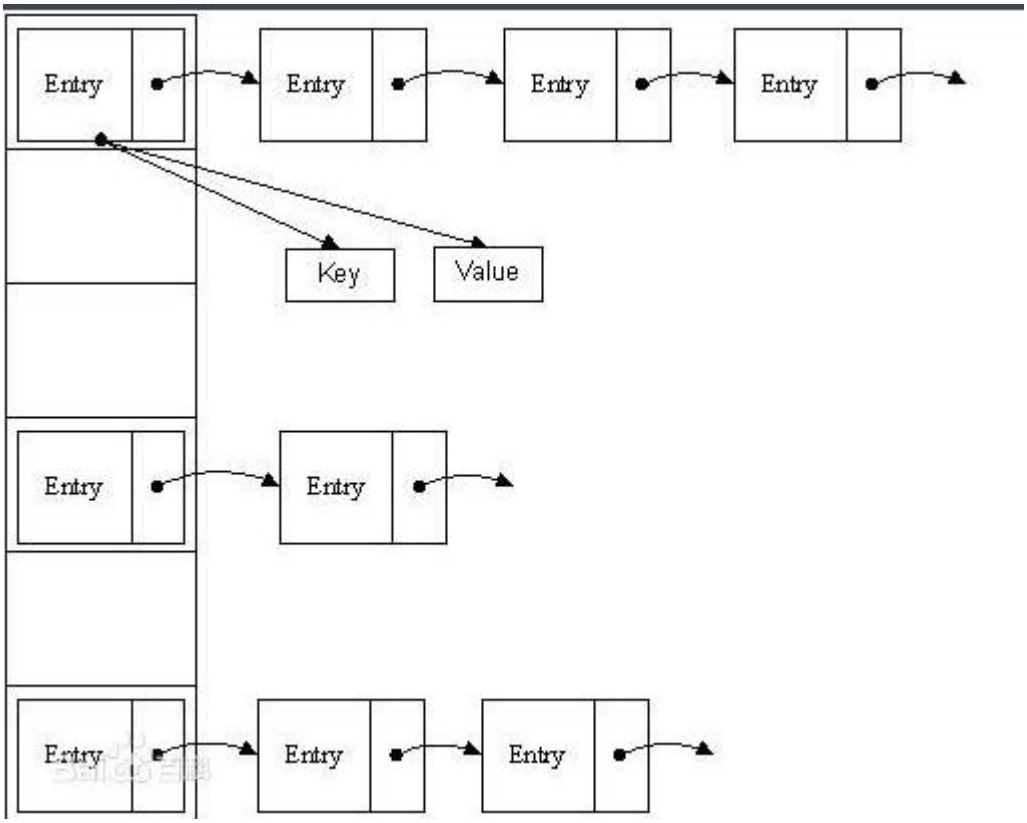
10. HashMap 怎么手写实现？

https://blog.csdn.net/qq_19431333/article/details/55505675

HashMap 有两个影响性能的重要参数：初始容量和加载因子。容量是 Hash 表中桶的个数，当 HashMap 初始化时，容量就是初始容量。加载因子是衡量 hash 表多满的一个指标，用来判断是否需要增加容量。当 HashMap 需要增加容量时，将会导致 rehash 操作。

默认情况下，0.75 的加载因子在时间和空间方面提供了很好的平衡。加载因子越大，增加了空间利用率但是也增加了查询的时间。

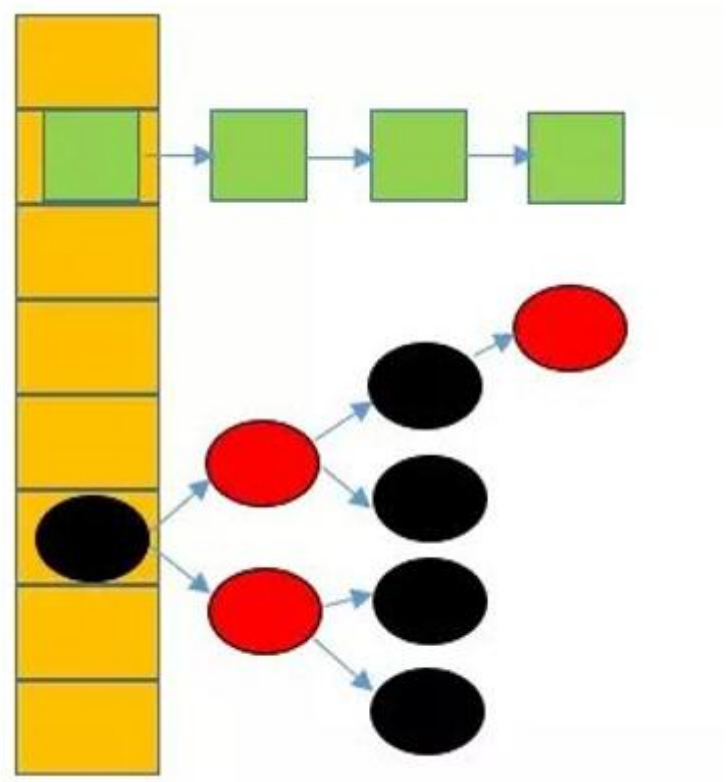
在 JDK1.7 之前，HashMap 采用的是数组+链表的结构，其结构图如下：



左边部分代表 Hash 表，数组的每一个元素都是一个单链表的头节点，链表是用来解决冲突的，如果不同的 key 映射到了数组的同一位置处，就将其放入单链表中。

JDK1.8 的结构

JDK1.8 之前的 HashMap 都采用上图的结构，都是基于一个数组和多个单链表，hash 值冲突的时候，就将对应节点以链表形式存储。如果在一个链表中查找一个节点时，将会花费 O(n)的查找时间，会有很大的性能损失。到了 JDK1.8，当同一个 Hash 值的节点数不小于 8 时，不再采用单链表形式存储，而是采用红黑树，如下图所示：



HashMap 中有几个重要的字段，如下：

```
//Hash 表结构
transient Node<K,V>[] table;

//元素个数
transient int size;

//确保 fail-fast 机制
transient int modCount;

//下一次增容前的阈值
int threshold;

//加载因子
final float loadFactor;

 //默认初始容量
static final int DEFAULT_INITIAL_CAPACITY = 1 << 4; // aka 16

//最大容量
static final int MAXIMUM_CAPACITY = 1 << 30;
```

```
//加载因子
static final float DEFAULT_LOAD_FACTOR = 0.75f;

//链表转红黑树的阈值
static final int TREEIFY_THRESHOLD = 8;
```

HashMap 一共有 4 个构造方法，主要的工作就是完成容量和加载因子的赋值。

Hash 表都是采用的懒加载方式，当第一次插入数据时才会创建。

源码看容器系列的 HashMap 分析

11. ConcurrentHashMap 的实现原理

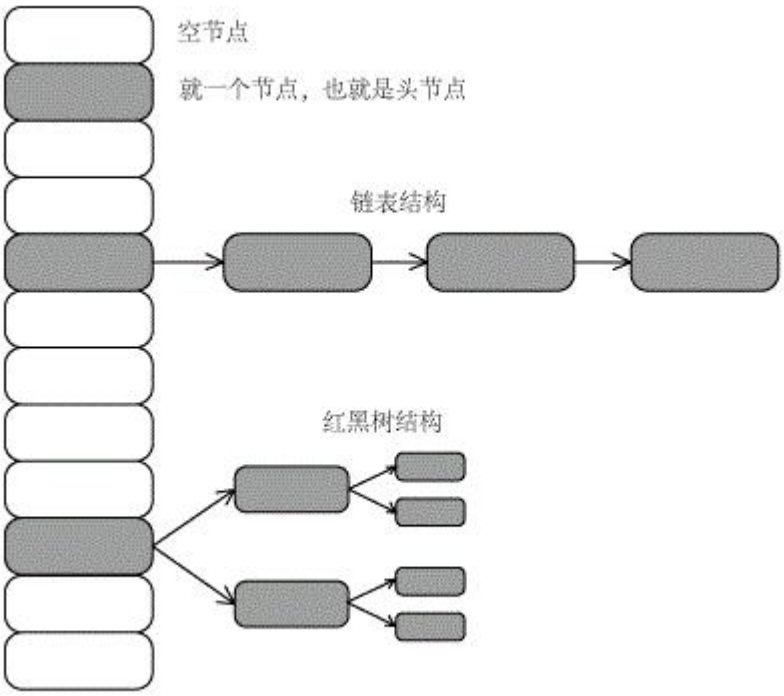
查看 concurrentHashMap 介绍

12. ArrayMap 和 HashMap 的对比

https://blog.csdn.net/vansbelove/article/details/52422087

HashMap

HashMap 内部是使用一个默认容量为 16 的数组来存储数据的，而数组中每一个元素却又是一个链表的头结点或者红黑树的根节点，所以，更准确的来说，HashMap 内部存储结构是使用**哈希表的数组+链表+红黑树**。如图：



这些 Entry 数据是按什么规则进行存储的呢？就是通过计算元素 key 的 hash 值，然后对 HashMap 中数组长度取余得到该元素存储的位置，计算公式为(key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);

ArrayMap

ArrayMap 是一个<key,value>映射的数据结构，它设计上更多的是考虑内存的优化，内部是使用两个数组进行数据存储，一个数组记录 key 的 hash 值，另外一个数组记录 Value 值，它和 SparseArray 一样，也会对 key 使用二分法进行从小到大排序，在添加、删除、查找数据的时候都是先使用二分查找法得到相应的 index，然后通过 index 来进行添加、查找、删除等操作，所以，应用场景和 SparseArray 的一样，如果在数据量比较大的情况下，那么它的性能将退化至少

50%。

ArrayMap 应用场景

1.数据量不大，最好在千级以内

2.数据结构类型为 Map 类型

HashMap 和 ArrayMap 各自的优势

1.查找效率

HashMap 因为其根据 hashcode 的值直接算出 index，所以其查找效率是随着数组长度增大而增加的。

ArrayMap 使用的是二分法查找，所以当数组长度每增加一倍时，就需要多进行一次判断，效率下降。

所以对于 Map 数量比较大的情况下，推荐使用

2.扩容数量

HashMap 初始值 16 个长度，每次扩容的时候，直接申请双倍的数组空间。

ArrayMap 每次扩容的时候，如果 size 长度大于 8 时申请 size*1.5 个长度，大于 4 小于 8 时申请 8 个，小于 4 时申请 4 个。

这样比较 ArrayMap 其实是申请了更少的内存空间，但是扩容的频率会更高。因此，如果当数据量比较大的时候，还是使用 HashMap 更合适，因为其扩容的次数要比 ArrayMap 少很多。

3.扩容效率

HashMap 每次扩容的时候时重新计算每个数组成员的位置，然后放到新的位置。

ArrayMap 则是直接使用 System.arraycopy。

所以效率上肯定是 ArrayMap 更占优势。

这里需要说明一下，网上有一种传闻说因为 ArrayMap 使用 System.arraycopy 更省内存空间，这一点我真的没有看出来。arraycopy 也是把老的数组的对象一个一个的赋给新的数组。当然效率上肯定 arraycopy 更高，因为是直接调用的 c 层的代码。

4.内存耗费

以 ArrayMap 采用了一种独特的方式，能够重复的利用因为数据扩容而遗留下来的数组空间，方便下一个 ArrayMap 的使用。而 HashMap 没有这种设计。

由于 ArrayMap 只缓存了长度是 4 和 8 的时候，所以如果频繁的使用到 Map，而且数据量都比较小的时候，ArrayMap 无疑是相当的节省内存的。

5.总结

综上所述，数据量比较小，并且需要频繁的使用 Map 存储数据的时候，推荐使用 ArrayMap。

而数据量比较大的时候，则推荐使用 HashMap。

13. HashTable 实现原理

https://www.jianshu.com/p/526970086e4e

Hashtable 与 HashMap 的区别

Hashtable 与 HashMap 都是用来存储 key-value 类型数据的，两者有如下区别：

1、Hashtable 不允许 null key 和 null value，HashMap 允许。

2、Hashtable 是线程安全的，HashMap 不是线程安全的。

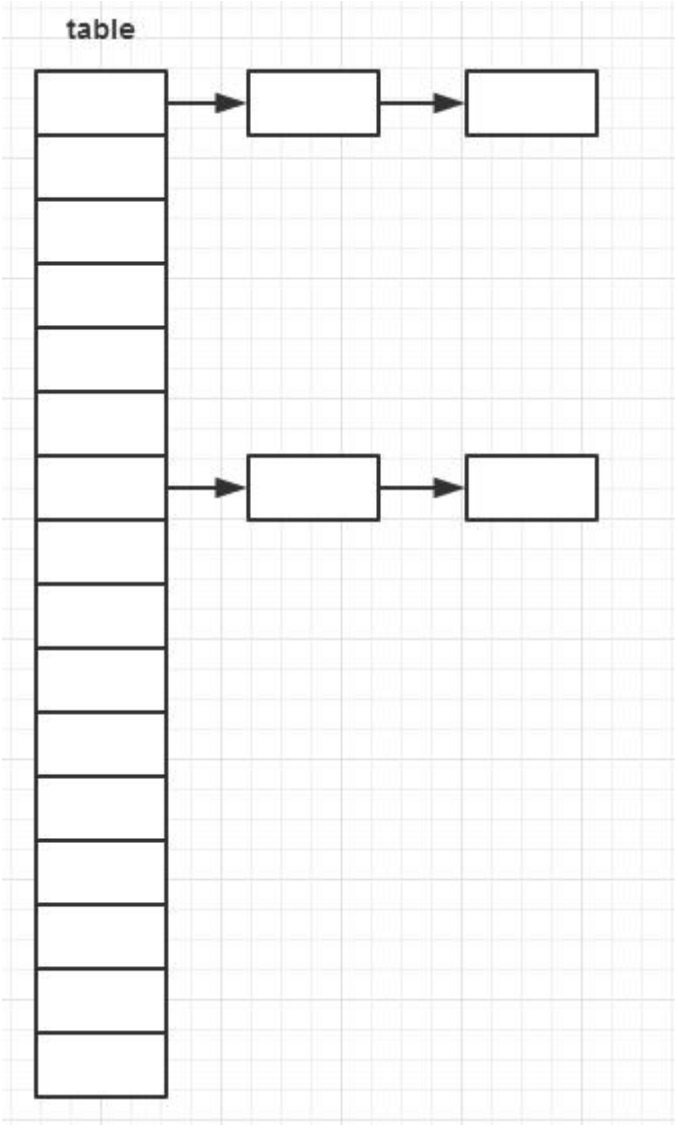
3、HashMap 的迭代器(Iterator)是 fail-fast 迭代器，而 Hashtable 的 enumerator 迭代器不是 fail-fast 的。所以当有其它线程改变了 HashMap 的结构（增加或者移除元素），将会抛出 ConcurrentModificationException，但迭代器本身的 remove()方法移除元素则不会抛出 ConcurrentModificationException 异常。但这并不是一个一定发生的行为，要看 JVM。这条同样也是 Enumeration 和 Iterator 的区别。

4、Hashtable 继承自 Dictionary,HashMap 继承自 AbstractMap。

5、两者都实现了 Map 接口。

Hashtable 的数据结构如下图所示，采用 Entry 数组+链表的方式实

现。



在 put 方法上增加了 synchronized

附加：Hashtable 如何保证线程安全

前面提到的 put 方法中是通过 synchronized 来保证线程安全的。查看源码，发现对外提供的 public 方法，几乎全部加上了 synchronized 关键字。如下：

```
public synchronized int size(){}
```

```
public synchronized boolean isEmpty() {}

public synchronized Enumeration<K> keys() {}

public synchronized Enumeration<V> elements() {}

public synchronized boolean contains(Object value) {}

public synchronized boolean containsKey(Object key) {}

public synchronized V get(Object key) {}

public synchronized V put(K key, V value) {}

public synchronized V remove(Object key) {}

public synchronized void putAll(Map<? extends K, ? extends V> t) {}

public synchronized void clear() {}

public synchronized Object clone() {}

public synchronized String toString() {}

public synchronized boolean equals(Object o) {}

public synchronized int hashCode() {}
```

所以从这个特性上看，Hashtable 是通过简单粗暴的方式来保证线程安全的额。所以 Hashtable 的性能在多线程环境下会非常低效。前面介绍的 ConcurrentHashMap 其实就是 Java 开发团队为了替换他而开发的，性能提高了不少。笔者也建议大家在多线程环境下抛弃 Hashtable，改用 ConcurrentHashMap，或者用 HashMap 配合外部锁，例如 ReentrantLock 来提高效率。

大概是 Java 团队已经开发出替代者 ConcurrentHashMap，所以

Hashtable 从源码的实现来看，和 JDK5 相比，没有什么提升，大概是 Java 团队放弃了这个类的维护了。

14. TreeMap 具体实现

https://yikun.github.io/2015/04/06/Java-TreeMap%E5%B7%A5%E4%BD%9C%E5%8E%9F%E7%90%86%E5%8F%8A%E5%AE%9E%E7%8E%B0/

看 java 容器介绍

15. SpareArray 原理(源码)

https://blog.csdn.net/zxt0601/article/details/78342675

暂时不看

16. HashMap 和 HashTable 的区别

HashMap 和 Hashtable 都实现了 Map 接口，但决定用哪一个之前先要弄清楚它们之间的分别。主要的区别有：**线程安全性，同步(synchronization)，以及速度。**

HashMap 几乎可以等价于 Hashtable，除了 HashMap 是非 synchronized 的，并可以接受 null(HashMap 可以接受为 null 的键

值(key)和值(value)，而 Hashtable 则不行)。

HashMap 是非 synchronized，而 Hashtable 是 synchronized，这意味着 Hashtable 是线程安全的，多个线程可以共享一个 Hashtable；而如果没有正确的同步的话，多个线程是不能共享 HashMap 的。Java 5 提供了 ConcurrentHashMap，它是 HashTable 的替代，比 HashTable 的扩展性更好。

另一个区别是 HashMap 的迭代器(Iterator)是 fail-fast 迭代器，而 Hashtable 的 enumerator 迭代器不是 fail-fast 的。所以当有其它线程改变了 HashMap 的结构（增加或者移除元素），将会抛出 ConcurrentModificationException，但迭代器本身的 remove()方法移除元素则不会抛出 ConcurrentModificationException 异常。但这并不是一个一定发生的行为，要看 JVM。这条同样也是 Enumeration 和 Iterator 的区别。

由于 Hashtable 是线程安全的也是 synchronized，所以在单线程环境下它比 HashMap 要慢。如果你不需要同步，只需要单一线程，那么使用 HashMap 性能要好过 Hashtable。

HashMap 不能保证随着时间的推移 Map 中的元素次序是不变的。

17. HashMap 与 HashSet 的区别

HashSet 实现了 Set 接口，它不允许集合中有重复的值，当我们提到 HashSet 时，第一件事情就是在将对象存储在 HashSet 之前，要

先确保对象重写 equals()和 hashCode()方法，这样才能比较对象的值是否相等，以确保 set 中没有储存相等的对象。如果我们没有重写这两个方法，将会使用这个方法的默认实现。

HashMap 实现了 Map 接口，Map 接口对键值对进行映射。Map 中不允许重复的键。Map 接口有两个基本的实现，HashMap 和 TreeMap。TreeMap 保存了对象的排列次序，而 HashMap 则不能。HashMap 允许键和值为 null。HashMap 是非 synchronized 的，但 collection 框架提供方法能保证 HashMap synchronized，这样多个线程同时访问 HashMap 时，能保证只有一个线程更改 Map。

18. HashSet 与 HashMap 怎么判断集合元素重复？

自己看总结

https://blog.csdn.net/ning109314/article/details/17354839

HashMap 中判断元素是否相同主要有两个方法，一个是判断 key 是否相同，一个是判断 value 是否相同

HashSet 不能添加重复的元素，当调用 add（Object）方法时候，首先会调用 Object 的 hashCode 方法判 hashCode 是否已经存在，如不存在则直接插入元素；

如果已存在则调用 Object 对象的 equals 方法判断是否返回

true，如果为 true 则说明元素已经存在，如为 false 则插入元素。

发现 HashSet 竟然是借助 HashMap 来实现的，利用 HashMap 中 Key 的唯一性，来保证 HashSet 中不出现重复值。

从这段代码中可以看出，HashMap 中的 Key 是根据对象的 hashCode() 和 euqals()来判断是否唯一的。

结论：为了保证 HashSet 中的对象不会出现重复值，在被存放元素的类中必须要重写 hashCode()和 equals()这两个方法。

19. 集合 Set 实现 Hash 怎么防止碰撞

重写 hashcode（）和 equles（）方法

可以看到在遍历 table 中的元素判断键和值，

1，如果 hash 码值不相同，说明是一个新元素，存储；

如果没有元素和传入对象（也就是 add 的元素）的 hash 值相等，那么就认为这个元素在 table 中不存在，将其添加进 table；

2.1，如果 hash 码值相同，且 equles 判断相等，说明元素已经存在，不存；

2.2，如果 hash 码值相同，且 equles 判断不相等，说明元素不存在，存；

如果有元素和传入对象的 hash 值相等,那么,继续进行 equles()判断，如果仍然相等，那么就认为传入元素已经存在，不再添加，结

束，否则仍然添加；

可见 hashcode()和 equles()在此显得很关键了，下面就来了解一下 hashcode 和 equles 这两个方法：

首先要明确：只通过 hash 码值来判断两个对象时否相同合适吗？答案肯定是不合适的，因为存在两个元素的 hash 码值相同但是并不是同一个元素这样的情况；

那么要问什么是 hash 码值？

在 java 中存在一种 hash 表结构，它通过一个算法，计算出的结果就是 hash 码值；这个算法叫 hash 算法；

hash 算法是怎么计算的呢？

是通过对象中的成员来计算出来的结果；

如果成员变量是基本数据类型的值，那么用这个值 直接参与计算； 如果成员变量是引用数据类型的值，那么获取到这个成员变量的哈希码值后，再参数计算

因此在 hashSet 的 add 方法添加元素时，仅仅依靠 hash 值判断是否存在是不完全的 还要依靠 equals 方法。

20. ArrayList 和 LinkedList 的区别，以及应用场景

1、ArrayList 是基于数组实现的，其构造函数为：

```
private transient Object[] elementData;
private int size;
```

ArryList 初始化时，elementData 数组大小默认为 10；

每次 add（）时，先调用 ensureCapacity（）保证数组不会溢出，如果此时已满,会扩展为数组 length 的 1.5 倍+1,然后用 array.copy 的方法，将原数组拷贝到新的数组中；

ArrayList 线程不安全，Vector 方法是同步的，线程安全；

2、LinkedList 是基于双链表实现的：

```
Object element;
Entry next,previous;
```

初始化时，有个 header Entry，值为 null；

使用 header 的优点是：在任何一个条目（包括第一个和最后一个）都有一个前置条目和一个后置条目，因此在 LinkedList 对象的开始或者末尾进行插入操作没有特殊的地方；

使用场景：

1） 如果应用程序对各个索引位置的元素进行大量的存取或删除操作，ArrayList 对象要远优于 LinkedList 对象；

2） 如果应用程序主要是对列表进行循环，并且循环时候进行插入或者删除操作，LinkedList 对象要远优于 ArrayList 对象；