1、ArrayList和linkedList的区别

Array(数组）是基于索引**(index)**的数据结构，它使用索引在数组中搜索和读取数据是很快的。

Array获取数据的时间复杂度是O(1),但是要删除数据却是开销很大，因为这需要重排数组中的所有数据,(因为删除数据以后, 需要把后面所有的数据前移)

缺点: 数组初始化必须指定初始化的长度, 否则报错

例如:

```
int[] a = new int[4];//推荐使用int[] 这种方式初始化

int c[] = {23,43,56,78};//长度：4，索引范围：[0,3]
```

List—是一个有序的集合，可以包含重复的元素，提供了按索引访问的方式，它继承Collection。

List有两个重要的实现类：ArrayList和LinkedList

ArrayList: 可以看作是能够自动增长容量的数组

ArrayList的toArray方法返回一个数组

ArrayList的asList方法返回一个列表

ArrayList底层的实现是Array, 数组扩容实现

LinkList是一个双链表,在添加和删除元素时具有比ArrayList更好的性能.但在get与set方面弱于ArrayList.当然,这些对比都是指数据量很大或者操作很频繁。

2、 HashMap和HashTable的区别

1、两者父类不同

HashMap是继承自AbstractMap类，而Hashtable是继承自Dictionary类。不过它们都实现了同时实现了map、Cloneable（可复制）、Serializable（可序列化）这三个接口。

2、对外提供的接口不同

Hashtable比HashMap多提供了elments() 和contains() 两个方法。

elments() 方法继承自Hashtable的父类Dictionnary。elements() 方法用于返回此Hashtable中的value的枚举。

contains()方法判断该Hashtable是否包含传入的value。它的作用与containsValue()一致。事实上，contansValue() 就只是调用了一下contains() 方法。

3、对null的支持不同

Hashtable：key和value都不能为null。

HashMap：key可以为null，但是这样的key只能有一个，因为必须保证key的唯一性；可以有多个key值对应的value为null。

4、安全性不同

HashMap是线程不安全的，在多线程并发的环境下，可能会产生死锁等问题，因此需要开发人员自己处理多线程的安全问题。

Hashtable是线程安全的，它的每个方法上都有synchronized 关键字，因此可直接用于多线程中。

虽然HashMap是线程不安全的，但是它的效率远远高于Hashtable，这样设计是合理的，因为大部分的使用场景都是单线程。当需要多线程操作的时候可以使用线程安全的ConcurrentHashMap。

ConcurrentHashMap虽然也是线程安全的，但是它的效率比Hashtable要高好多倍。因为ConcurrentHashMap使用了分段锁，并不对整个数据进行锁定。

5、初始容量大小和每次扩充容量大小不同

6、计算hash值的方法不同

3、Collection包结构，与Collections的区别

Collection是集合类的上级接口，子接口有 Set、List、LinkedList、ArrayList、Vector、Stack、Set；

Collections是集合类的一个帮助类， 它包含有各种有关集合操作的静态多态方法，用于实现对各种集合的搜索、排序、线程安全化等操作。此类不能实例化，就像一个工具类，服务于Java的Collection框架。

4、泛型常用特点（待补充）

泛型是Java SE 1.5之后的特性，《Java 核心技术》中对泛型的定义是：

> “泛型”意味着编写的代码可以被不同类型的对象所重用。

“泛型”，顾名思义，“泛指的类型”。我们提供了泛指的概念，但具体执行的时候却可以有具体的规则来约束，比如我们用的非常多的ArrayList就是个泛型类，ArrayList作为集合可以存放各种元素，如Integer, String，自定义的各种类型等，但在我们使用的时候通过具体的规则来约束，如我们可以约束集合中只存放Integer类型的元素，如

```
List<Integer> iniData = new ArrayList<>()
```

使用泛型的好处？

以集合来举例，使用泛型的好处是我们不必因为添加元素类型的不同而定义不同类型的集合，如整型集合类，浮点型集合类，字符串集合类，我们可以定义一个集合来存放整型、浮点型，字符串型数据，而这并不是最重要的，因为我们只要把底层存储设置了Object即可，添加的数据全部都可向上转型为Object。 更重要的是我们可以通过规则按照自己的想法控制存储的数据类型。

5、说说List,Set,Map三者的区别

List(对付顺序的好帮手)： List接口存储一组不唯一（可以有多个元素引用相同的对象），有序的对象

Set(注重独一无二的性质):不允许重复的集合。不会有多个元素引用相同的对象。

Map(用Key来搜索的专): 使用键值对存储。Map会维护与Key有关联的值。两个Key可以引用相同的对象，但Key不能重复，典型的Key是String类型，但也可以是任何对象。

6、Array与ArrayList有什么不一样？

Array与ArrayList都是用来存储数据的集合。ArrayList底层是使用数组实现的，但是arrayList对数组进行了封装和功能扩展，拥有许多原生数组没有的一些功能。我们可以理解成ArrayList是Array的一个升级版。

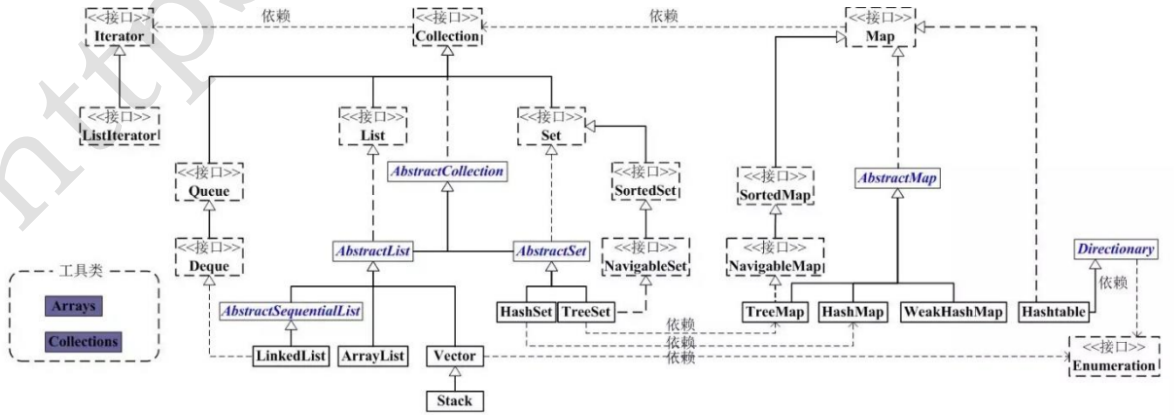
7、Map有什么特点

以键值对存储数据 元素存储循序是无序的 不允许出现重复键

8、集合类存放于 Java.util 包中， 主要有几 种接口

主要包含set(集）、 list(列表包含 Queue）和 map(映射)。

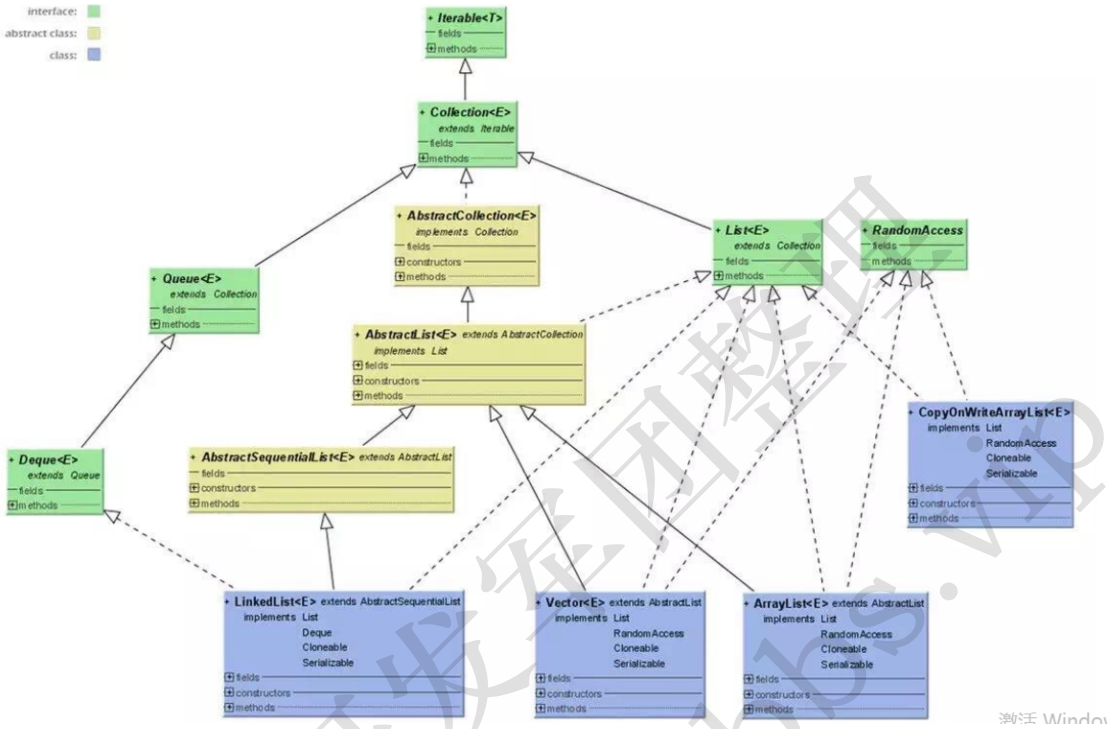
1. Collection： Collection 是集合 List、 Set、 Queue 的最基本的接口。
2. Iterator：迭代器，可以通过迭代器遍历集合中的数据
3. Map：是映射表的基础接口



9、什么是list接口

Java 的 List 是非常常用的数据类型。 List 是有序的 Collection。 Java List 一共三个实现类： 分别是 ArrayList、 Vector 和 LinkedList 。

list接口结构图



10、说说ArrayList（数组）

ArrayList 是最常用的 List 实现类，内部是通过数组实现的，它允许对元素进行快速随机访问。数 组的缺点是每个元素之间不能有间隔， 当数组大小不满足时需要增加存储能力，就要将已经有数 组的数据复制到新的存储空间中。 当从 ArrayList 的中间位置插入或者删除元素时，需要对数组进 行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除。

11、Vector（ 数组实现、 线程同步）

Vector 与 ArrayList 一样，也是通过数组实现的，不同的是它支持线程的同步，即某一时刻只有一 个线程能够写 Vector，避免多线程同时写而引起的不一致性，但实现同步需要很高的花费，因此， 访问它比访问 ArrayList 慢 。

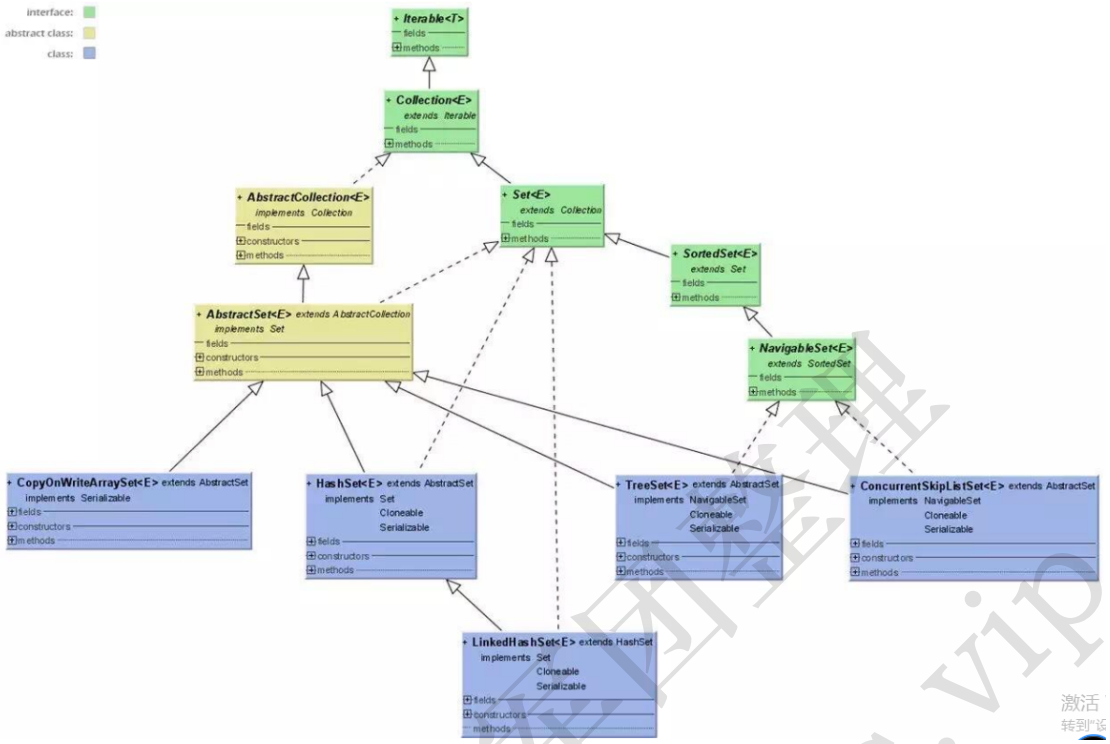
12、说说LinkList（链表）

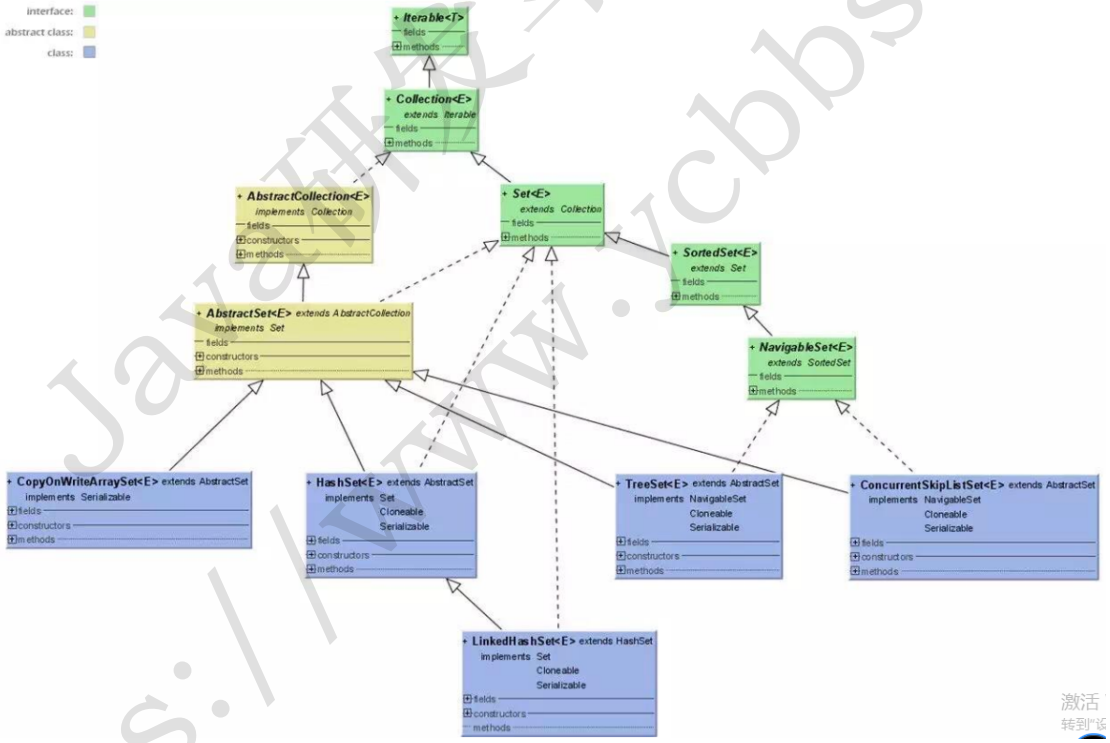
LinkedList 是用链表结构存储数据的，很适合数据的动态插入和删除，随机访问和遍历速度比较 慢。另外，他还提供了 List 接口中没有定义的方法，专门用于操作表头和表尾元素，可以当作堆 栈、队列和双向队列使用

13、什么Set集合

Set 注重独一无二的性质,该体系集合用于存储无序(存入和取出的顺序不一定相同)元素， 值不能重 复。对象的相等性本质是对象 hashCode 值（java 是依据对象的内存地址计算出的此序号） 判断 的， 如果想要让两个不同的对象视为相等的，就必须覆盖 Object 的 hashCode 方法和 equals 方 法。

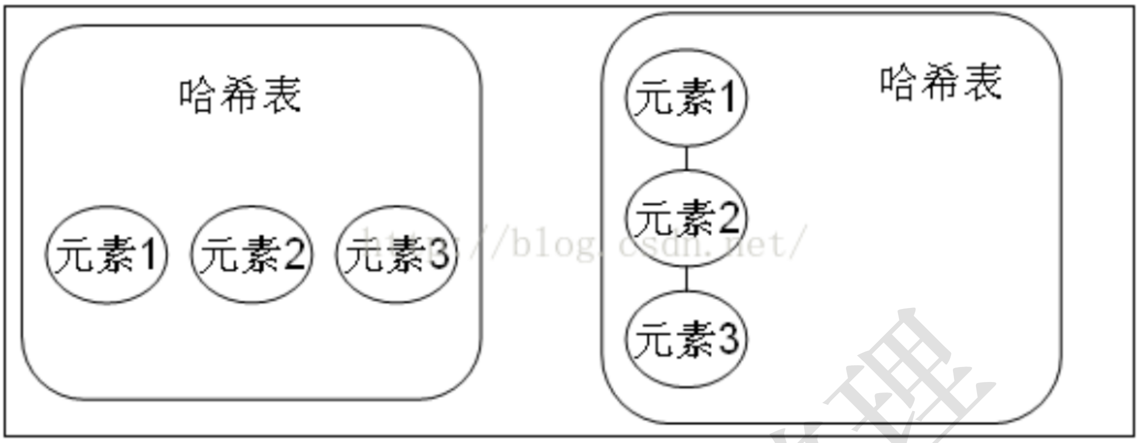
set结构结构图





14、HashSet（ Hash 表）

哈希表边存放的是哈希值。 HashSet 存储元素的顺序并不是按照存入时的顺序（和 List 显然不 同） 而是按照哈希值来存的所以取数据也是按照哈希值取得。元素的哈希值是通过元素的 hashcode 方法来获取的, HashSet 首先判断两个元素的哈希值，如果哈希值一样，接着会比较 equals 方法 如果 equls 结果为 true ， HashSet 就视为同一个元素。如果 equals 为 false 就不是 同一个元素。 哈希值相同 equals 为 false 的元素是怎么存储呢,就是在同样的哈希值下顺延（可以认为哈希值相 同的元素放在一个哈希桶中）。也就是哈希一样的存一列。 如图 1 表示 hashCode 值不相同的情 况； 图 2 表示 hashCode 值相同，但 equals 不相同的情况。



HashSet 通过 hashCode 值来确定元素在内存中的位置。 一个 hashCode 位置上可以存放多个元 素。

15、什么是TreeSet（二叉树）

1. TreeSet()是使用二叉树的原理对新 add()的对象按照指定的顺序排序（升序、降序），每增 加一个对象都会进行排序，将对象插入的二叉树指定的位置。
2. Integer 和 String 对象都可以进行默认的 TreeSet 排序，而自定义类的对象是不可以的， 自 己定义的类必须实现 Comparable 接口，并且覆写相应的 compareTo()函数，才可以正常使 用。
3. 在覆写 compare()函数时，要返回相应的值才能使 TreeSet 按照一定的规则来排序
4. 比较此对象与指定对象的顺序。如果该对象小于、等于或大于指定对象，则分别返回负整 数、零或正整数

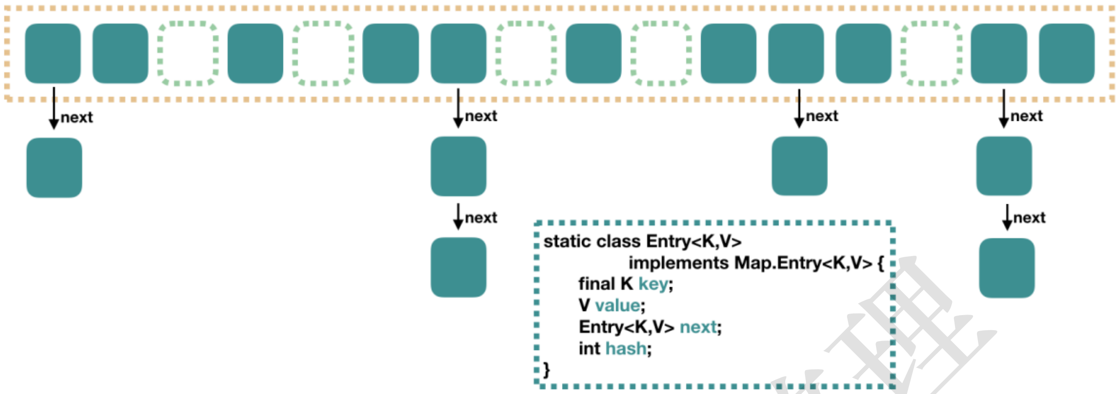
16、说说LinkHashSet（ HashSet+LinkedHashMap）

对于 LinkedHashSet 而言，它继承与 HashSet、又基于 LinkedHashMap 来实现的。 LinkedHashSet 底层使用 LinkedHashMap 来保存所有元素，它继承与 HashSet，其所有的方法 操作上又与 HashSet 相同，因此 LinkedHashSet 的实现上非常简单，只提供了四个构造方法，并 通过传递一个标识参数，调用父类的构造器，底层构造一个 LinkedHashMap 来实现，在相关操 作上与父类 HashSet 的操作相同，直接调用父类 HashSet 的方法即可。

17、HashMap（数组+链表+红黑树）

HashMap 根据键的 hashCode 值存储数据，大多数情况下可以直接定位到它的值，因而具有很快 的访问速度，但遍历顺序却是不确定的。 HashMap 最多只允许一条记录的键为 null，允许多条记 录的值为 null。 HashMap 非线程安全，即任一时刻可以有多个线程同时写 HashMap，可能会导 致数据的不一致。如果需要满足线程安全，可以用 Collections 的 synchronizedMap 方法使 HashMap 具有线程安全的能力，或者使用 ConcurrentHashMap。 我们用下面这张图来介绍 HashMap 的结构。

Java7 HashMap 结构

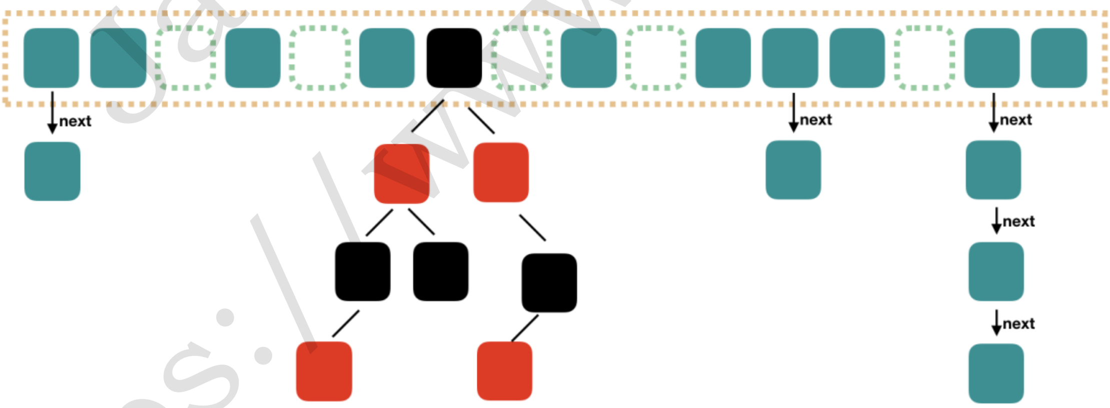


大方向上， HashMap 里面是一个数组，然后数组中每个元素是一个单向链表。上图中，每个绿色 的实体是嵌套类 Entry 的实例， Entry 包含四个属性： key, value, hash 值和用于单向链表的 next。

1. capacity：当前数组容量，始终保持 2^n，可以扩容，扩容后数组大小为当前的 2 倍。
2. oadFactor：负载因子，默认为 0.75。
3. threshold：扩容的阈值，等于 capacity * loadFactor

Java8 对 HashMap 进行了一些修改， 最大的不同就是利用了红黑树，所以其由 数组+链表+红黑 树 组成。 根据 Java7 HashMap 的介绍，我们知道，查找的时候，根据 hash 值我们能够快速定位到数组的具体下标，但是之后的话， 需要顺着链表一个个比较下去才能找到我们需要的，时间复杂度取决 于链表的长度，为 O(n)。为了降低这部分的开销，在 Java8 中， 当链表中的元素超过了 8 个以后， 会将链表转换为红黑树，在这些位置进行查找的时候可以降低时间复杂度为 O(logN)。

Java8 HashMap 结构



18、说说ConcurrentHashMap

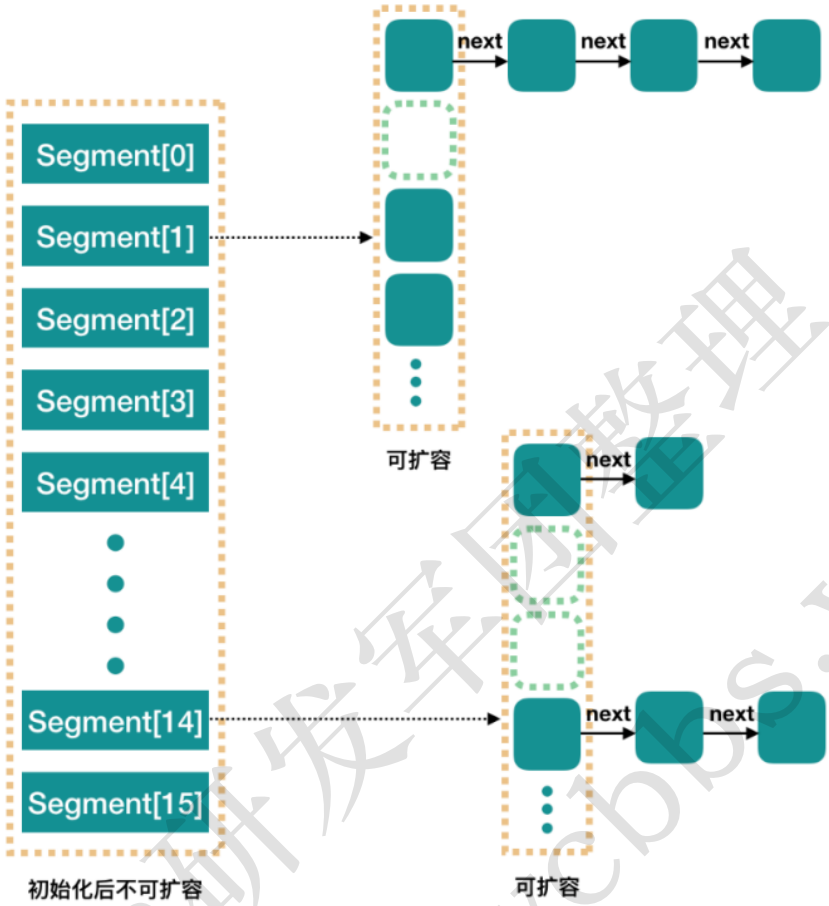
Segment 段

ConcurrentHashMap 和 HashMap 思路是差不多的，但是因为它支持并发操作，所以要复杂一 些。整个 ConcurrentHashMap 由一个个 Segment 组成， Segment 代表"部分"或"一段"的 意思，所以很多地方都会将其描述为分段锁。注意，行文中，我很多地方用了"槽"来代表一个 segment。

线程安全（Segment 继承 ReentrantLock 加锁）

简单理解就是， ConcurrentHashMap 是一个 Segment 数组， Segment 通过继承 ReentrantLock 来进行加锁，所以每次需要加锁的操作锁住的是一个 segment，这样只要保证每 个 Segment 是线程安全的，也就实现了全局的线程安全

Java7 ConcurrentHashMap 结构



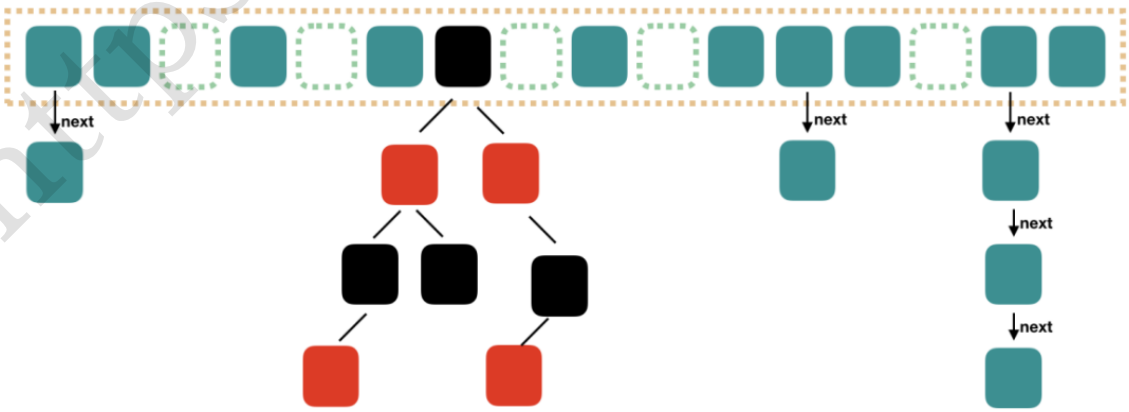
并行度（默认 16）

concurrencyLevel：并行级别、并发数、 Segment 数，怎么翻译不重要，理解它。默认是 16， 也就是说 ConcurrentHashMap 有 16 个 Segments，所以理论上， 这个时候，最多可以同时支 持 16 个线程并发写，只要它们的操作分别分布在不同的 Segment 上。这个值可以在初始化的时 候设置为其他值，但是一旦初始化以后，它是不可以扩容的。再具体到每个 Segment 内部，其实 每个 Segment 很像之前介绍的 HashMap，不过它要保证线程安全，所以处理起来要麻烦些。

Java8 实现（引入了红黑树）

Java8 对 ConcurrentHashMap 进行了比较大的改动,Java8 也引入了红黑树。

Java8 ConcurrentHashMap 结构



19、HashTable（线程安全）

Hashtable 是遗留类，很多映射的常用功能与 HashMap 类似，不同的是它承自 Dictionary 类， 并且是线程安全的，任一时间只有一个线程能写 Hashtable，并发性不如 ConcurrentHashMap， 因为 ConcurrentHashMap 引入了分段锁。 Hashtable 不建议在新代码中使用，不需要线程安全 的场合可以用 HashMap 替换，需要线程安全的场合可以用 ConcurrentHashMap 替换

20、TreeMap（可排序）

TreeMap 实现 SortedMap 接口，能够把它保存的记录根据键排序，默认是按键值的升序排序， 也可以指定排序的比较器，当用 Iterator 遍历 TreeMap 时，得到的记录是排过序的。 如果使用排序的映射，建议使用 TreeMap。 在使用 TreeMap 时， key 必须实现 Comparable 接口或者在构造 TreeMap 传入自定义的 Comparator，否则会在运行时抛出 java.lang.ClassCastException 类型的异常。 参考： https://www.ibm.com/developerworks/cn/java/j-lo-tree/index.html

21、LinkHashMap（记录插入顺序）

LinkedHashMap 是 HashMap 的一个子类，保存了记录的插入顺序，在用 Iterator 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的，也可以在构造时带参数，按照访问次序排序。 参考 1： http://www.importnew.com/28263.html 参考 2： http://www.importnew.com/20386.html#comment-648123

22、泛型类

泛型类的声明和非泛型类的声明类似，除了在类名后面添加了类型参数声明部分。和泛型方法一样，泛型类的类型参数声明部分也包含一个或多个类型参数，参数间用逗号隔开。一个泛型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。因为他们接受一个或多个参数，这些类被称为参数化的类或参数化的类型。

```
public class Box<T> {
    private T t;
    public void add(T t) {
        this.t = t;
    }
    public T get() {
        return t;
    }
}
```

23、类型通配符?

类型通配符一般是使用?代替具体的类型参数。 例如 List<?> 在逻辑上是List,List 等所有List<具体类型实参>的父类。

24、类型擦除

Java 中的泛型基本上都是在编译器这个层次来实现的。在生成的 Java 字节代码中是不包含泛型中的类型信息的。使用泛型的时候加上的类型参数，会被编译器在编译的时候去掉。这个过程就称为类型擦除。

如在代码中定义的 List和 List等类型，在编译之后都会变成 List。 JVM 看到的只是 List，而由泛型附加的类型信息对 JVM 来说是不可见的。

类型擦除的基本过程也比较简单，首先是找到用来替换类型参数的具体类。这个具体类一般是 Object。如果指定了类型参数的上界的话，则使用这个上界。把代码中的类型参数都替换成具体的类。