集合容器概述

1. 什么是集合

- 个 ，准
- ， 不
- 主 3 ：set( ) list(列) map( )

2. 集合的特点

- 主 下两 ：
  - 于 储 ， ， 了也 储 中
  - 不 为 变 前 义

3. 集合和数组的区别

- ； 变
- 以 储 ，也 以 储 ； 只 储
- 储 元 个 ； 储 以 不

4. 使用集合框架的好处

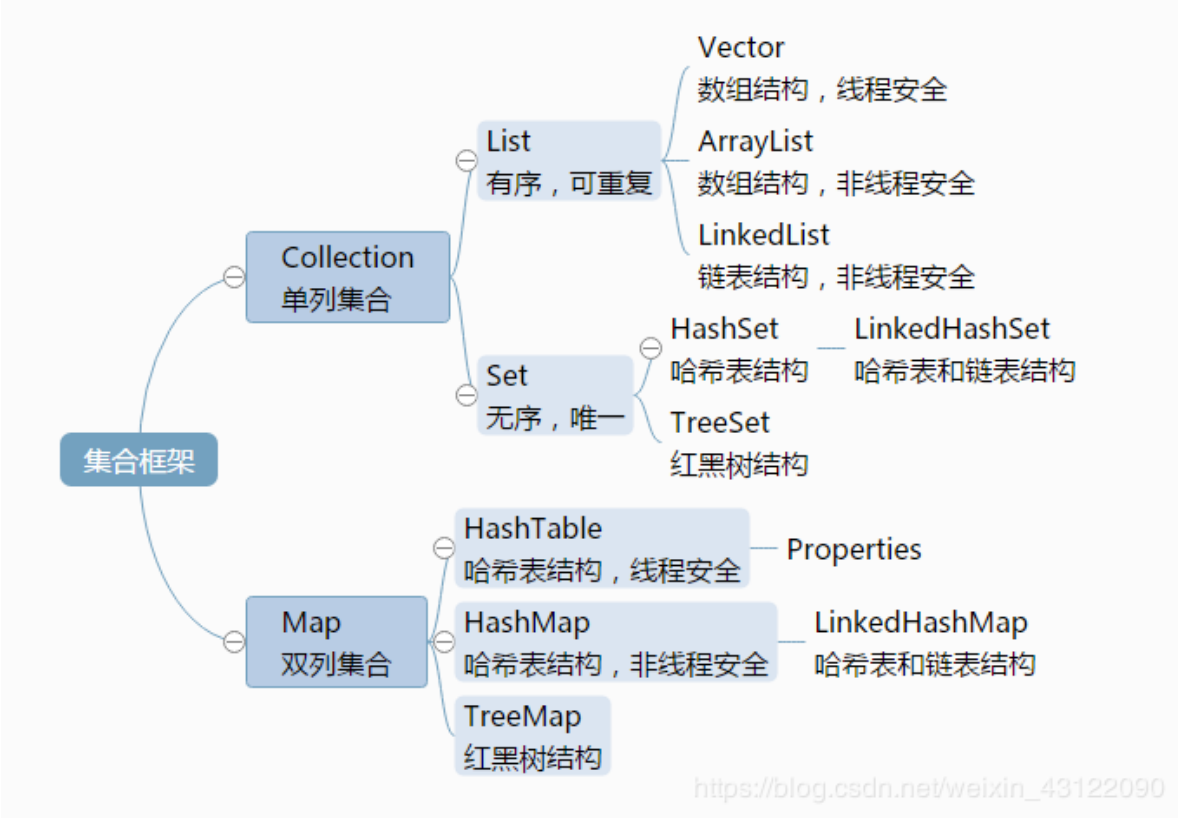
1. ；
2. 供了 ，使 ， 了 ；
3. 以 便 写 ， 代 作
4. 使 JDK ， 以 低代 习 API

5. 常用的集合类有哪些?

- Map 口 Collection 口 口:

1. Collection 口 口包 : Set 口 List 口
2. Map 口 主 : HashMap TreeMap Hashtable ConcurrentHashMap以及 Properties
3. Set 口 主 : HashSet TreeSet LinkedHashSet
4. List 口 主 : ArrayList LinkedList Stack以及Vector

6. List，Set，Map三者的区别?



- Java 分为 Collection Map 两 ，Collection 口 Set List Queue三 口 们 Set List，Map 口不 collection 口
- Collection 主 List Set两 口
  - List: 个 (元 入 取出) ，元 以 ， 以 入 个null元 ，元 ArrayList LinkedList Vector
  - Set: 个 (入 取出 不) ，不 以 储 元 ，只允 入 个null元 ， 保 元 Set 口 HashSet LinkedHashSet 以及 TreeSet
- Map 个 值 ， 储 值 之 Key ， ；value 不 ，允 Map 于Collection 口，从Map 中 元 ，只 出 ， 会 值
  - Map : HashMap TreeMap HashTable LinkedHashMap ConcurrentHashMap

7. 集合框架底层数据结构

- Collection
  1. List
     - Arraylist： Object
     - Vector： Object
     - LinkedList： 双
  2. Set
     - HashSet（ ， ）： 于 HashMap ， HashMap 保 元
     - LinkedHashSet： LinkedHashSet 与 HashSet， 且其内 LinkedHashMap 似于 们之前 LinkedHashMap 其内 于 Hashmap ，不 区别
     - TreeSet（ ， ）： （ 二叉 ）
- Map
  - HashMap： JDK1.8之前HashMap + ， HashMap 主体， 则 主 为了 决 冲 （“ ” 决冲 ）.JDK1.8以 决 冲 了 变化， 于 值（ 为8） ， 化为 ，以减
  - LinkedHashMap：LinkedHashMap HashMap， 以 仍 于 列 即 另 ，LinkedHashMap 上 上， 加 了 双 ，使 上 以保 值 入 作， 了 关
  - HashTable： + ， HashMap 主体， 则 主 为了 决 冲
  - TreeMap： （ 二叉 ）

8. 哪些集合类是线程安全的?

- Vector： Arraylist 了个 synchronized （ 全）， 为 低， 不 使
- hashTable： hashMap 了个synchronized (全)，不 使
- ConcurrentHashMap： Java5中 发 全HashMap Segment HashEntry Segment ConcurrentHashMap ，HashEntry则 于 储 -值 个ConcurrentHashMap 包 个Segment ， Segment HashMap 似， ； 个Segment 包 个HashEntry ， 个HashEntry 个 元 ； 个Segment 个HashEntry 元 ， HashEntry 修 ， 先 Segment （ 使 ）
- ...

9. Java集合的快速失败机制 “fail-fast”?

- java 制， 个 上 变 作 ， 会产 fail-fast 制
- 例 ：假 两个 （ 1 2）， 1 Iterator 历 A中 元 ， 个 候 2修 了 A （ 上 修 ， 不 单 修 元 内 ）， 么 个 候 会 出 ConcurrentModificationException ，从 产 fail-fast 制

- 原 ： 代 历 中 内 ， 且 历 中使 个 modCount 变 历 内 发 变化， 会 变modCount 值 代 使 hashNext()/next() 历下 个元 之前， 会 modCount变 为expectedmodCount值， 历； 则 出 ， 历
- 决办 ：
  1. 历 中， 及到 变modCount值 全 加上synchronized
  2. 使 CopyOnWriteArrayList ArrayList

10. 怎么确保一个集合不能被修改?

- 以使 Collections. unmodifiableCollection(Collection c) 创 个只 ， 变 任何 作 会 出 Java. lang. UnsupportedOperationException
- 例代 下:

```
  List<String> list = new ArrayList<>();
  list. add("x");
  Collection<String> clist = Collections. unmodifiableCollection(list);
  clist. add("y"); //
System. out. println(list. size());
```

Collection接口

List接口

11. 迭代器 Iterator 是什么?

- Iterator 口 供 历任何 Collection 口 们 以从 个 Collection 中使 代 取 代 例 代 取代了 Java 中 Enumeration， 代 允 代 中 元
- 为 Collection 了Iterator 代

```
public interface Collection<E> extends Iterable<E> {
    // Query Operations
```

12. Iterator 怎么使用? 有什么特点?

- Iterator 使 代 下:

```
  List<String> list = new ArrayList<>();
  Iterator<String> it = list. iterator();
  while(it. hasNext()){
    String obj = it. next();
    System. out. println(obj);
  }
```

- Iterator 只 单 历，但 加 全， 为 以 保， 前 历 元 候， 会 出 ConcurrentModificationException

13. 如何边遍历边移除 Collection 中的元素?

- 历 修 Collection 使 Iterator.remove() ， 下:

```
Iterator<Integer> it = list.iterator();
while(it.hasNext()){
   *// do something*
   it.remove();
}
```

错误代 下:

```
for(Integer i : list){
   list.remove(i)
}
```

- 以上 代 会 **ConcurrentModificationException 异常** 为 使 foreach(for(Integer i : list)) 句 ，会 动 个iterator 历 list，但 list Iterator.remove() 修 Java 不允 个 历 Collection 另 个 修

14. Iterator 和 ListIterator 有什么区别?

- Iterator 以 历 Set List ， ListIterator 只 历 List
- Iterator 只 单 历， ListIterator 以双 历（ 前/ 历）
- ListIterator Iterator 口， 加了 些 功 ， 加 个元 个元 取前 元 位

15. 遍历一个 List 有哪些不同的方式？每种方法的实现原理是什么？Java 中 List 遍历的最佳实践是什么?

- 历 以下几 :
  1. for 历， 于 个 ， 依 取 个位 元 ， 取到 个元 停
  2. 代 历，Iterator Iterator 个 ， 不 ， 历 口 Java Collections 中 了 Iterator
  3. foreach 历 foreach 内 也 了 Iterator ，使 不 Iterator 优 代 ，不 出 ； 只 做 单 历，不 历 中 作 ，例 删
- 佳 : Java Collections 中 供了 个 RandomAccess 口， List Random Access
  - 个 了 口， Random Access， 位 取元 为 O(1)， ArrayList
  - 口， 不 Random Access， LinkedList

    - 做 ， Random Access 列 for 历， 则 Iterator foreach 历

16. 说一下 ArrayList 的优缺点

- ArrayList 优 下：
    - ArrayList 以 ， ArrayList 了 RandomAccess 口， 候
    - ArrayList 加 个元 候 便
- ArrayList 下：
    - 删 元 候， 做 元 制 作 制 元 ， 么 会
    - 入元 候，也 做 元 制 作， 上
- ArrayList 加

17. 如何实现数组和 List 之间的转换?

- List：使 Arrays. asList(array)
- List ：使 List toArray()
- 代 例：

```
// list to array
List<String> list = new ArrayList<String>();
list.add("123");
list.add("456");
list.toArray();

// array to list
String[] array = new String[]{"123","456"};
Arrays.asList(array);
```

18. ArrayList 和 LinkedList 的区别是什么?

- ：ArrayList 动 ， LinkedList 双
- ：ArrayList LinkedList 候 ， 为 LinkedList 储 ， 以 动 从前 依
- 加 删 ： 加 删 作，LinkedList ArrayList ， 为 ArrayList 删 作 内 其他 下
- 内 占 ：LinkedList ArrayList 占内 ， 为 LinkedList 了 储 ， 储了两个 ， 个 前 个元 ， 个 个元
- 全：ArrayList LinkedList 不 ，也 不保 全；
- ， 取 中 元 ， 使 ArrayList， 入 删 作 ， 使 LinkedList
- LinkedList 双 也叫双 ， ， 个 中 两个 ，分别 前 以，从双 中 任 个 ， 以 便 前

19. ArrayList 和 Vector 的区别是什么?

- 两个 了 List 口（List 口 了 Collection 口），他们
  - 全：Vector 使 了 Synchronized ， 全 ， ArrayList 全
  - ：ArrayList 优于 Vector
  - ：ArrayList Vector 会 动 ，只不 Vector 会 加 1 倍， ArrayList 只会 加 50%
- Vector 以 两个 全 个Vector 但 个 Vector 代 作上
- Arraylist不 ， 以 不 保 全 使 Arraylist

20. 插入数据时，ArrayList、LinkedList、Vector谁速度较快？阐述 ArrayList、Vector、LinkedList 的存储性能和特性?

- ArrayList Vector 使 储 元 于 储 以便 加 入元 ， 们 允 元 ，但 入元 及 元 动 内 作， 以 入
- Vector 中 于加了 synchronized 修 ， **Vector 是线程安全容器，但性能上较 ArrayList差**
- LinkedList 使 双 储， 前 历，但 入 只 前 前 即 ， 以 **LinkedList 插入速度较快**

21. 多线程场景下如何使用 ArrayList?

- ArrayList 不 全 ， 到 ， 以 Collections synchronizedList 其 全 再使 例 像下 ：

```
List<String> synchronizedList = Collections.synchronizedList(list);
synchronizedList.add("aaa");
synchronizedList.add("bbb");

for (int i = 0; i < synchronizedList.size(); i++) {
    System.out.println(synchronizedList.get(i));
}
```

22. 为什么 ArrayList 的 elementData 加上 transient 修饰?

- ArrayList 中 义 下:

  private transient Object[] elementData;
- 再 下 ArrayList 义:

```
public class ArrayList<E> extends AbstractList<E>
     implements List<E>, RandomAccess, Cloneable, java.io.Serializable
```

- 以 到 ArrayList 了 Serializable 口， ArrayList 列化 transient 作 不 elementData 列化， 写了 writeObject :

```
  private void writeObject(java.io.ObjectOutputStream s) throws
java.io.IOException{
      *// Write out element count, and any hidden stuff*
          int expectedModCount = modCount;
      s.defaultWriteObject();
      *// Write out array length*
          s.writeInt(elementData.length);
      *// Write out all elements in the proper order.*
          for (int i=0; i<size; i++)
              s.writeObject(elementData[i]);
      if (modCount != expectedModCount) {
          throw new ConcurrentModificationException();
  }
```

- 列化 ，先 defaultWriteObject() 列化 ArrayList 中 transient 元 ， 历 elementData，只 列化 入 元 ， 加 了 列化 ，又减 了 列化之 件

23. List 和 Set 的区别

- List , Set Collection 口
- List : 个 (元 入 取出) ，元 以 ， 以 入 个null元 ，元 ArrayList LinkedList Vector
- Set : 个 (入 取出 不) ，不 以 储 元 ，只允 入 个null元 ， 保 元 Set 口 HashSet LinkedHashSet 以及 TreeSet
- 另 List for ，也 下 历，也 以 代 ，但 set只 代， 为他 ， 下 取 值
- Set List
  - Set: 元 低下，删 入 ， 入 删 不会 元 位 变
  - List: 似，List 以动 ， 元 ， 入删 元 低， 为会 其他元 位 变

Set接口

24. 说一下 HashSet 的实现原理?

- HashSet 于 HashMap ，HashSet 值 于HashMap key上，HashMap value 为present， HashSet 单， 关 HashSet 作， 上 HashMap 关 ，HashSet 不允 值

25. HashSet如何检查重复？HashSet是如何保证数据不可重复的?

- HashSet 中add ()元 ，判 元 依 ，不仅 hash值， equles
- HashSet 中 add () 会使 HashMap put()

- HashMap key , 以 出 HashSet 加 去 值 作为HashMap key, 且 HashMap中 K/V , 会 V V, V 以不会 (HashMap key 先 hashcode 再 equals)
- 以下 HashSet 分 :

```
  private static final Object PRESENT = new Object();
  private transient HashMap<E,Object> map;

  public HashSet() {
      map = new HashMap<>();
  }

  public boolean add(E e) {
      //    HashMap  put    ,PRESENT
return map.put(e, PRESENT)==null;
  }
```

hashCode（）与equals（）的相关规定:

1. 两个 ，则hashcode 也

- hashCode jdk 串 出 int 值

2. 两个 ， 两个equals true
3. 两个 hashcode值， 们也不
4. 上，equals ，则hashCode 也
5. hashCode() 为 上 产 值 写hashCode()，则 class 两个 何 不会 (即使 两个)

==与equals的区别

1. == 判 两个变 例 不 个内 equals 判 两个变 例 内 值 不
2. == 内 equals() 串 内

26. HashSet与HashMap的区别

HashMap	HashSet
了Map 口	Set 口
储 值	仅 储
put（） map中 加元	add（） Set中 加元
HashMap使 (Key) Hashcode	HashSet使 hashcode值， 于两个 hashcode ， 以equals() 判 ， 两个 不 ， 么 false
HashMap 于 HashSet ， 为 使 取	HashSet HashMap

Map接口

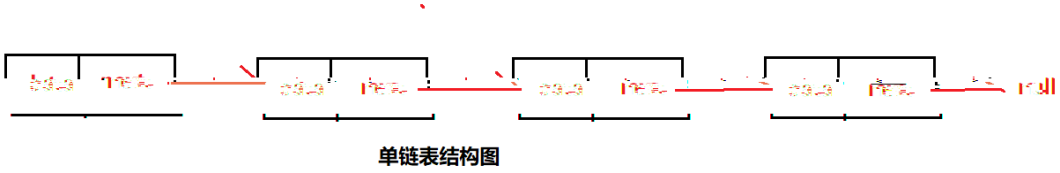
27. 什么是Hash算法

- 任 二 制 为 二 制值， 个 二 制值叫做值

28. 什么是链表

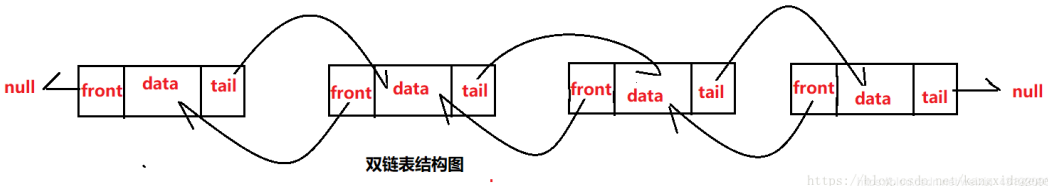
- 以 上不 ， 作， 删功
- 分为单 双

1. 单 ： 个 包 两 分， 分 变 data,另 分 下 next



单链表结构图

2. 双 ： 了包 单 分， 加 pre前 个



双链表结构图

- 优

    - 入删 （为next 其下个 ， 变 以便 加删元）
    - 内利 ，不会 内（以使内中 不 （于node ），且 候创 ）
    - ，
-
    - 不 ， 从 个 历， 低

29. 说一下HashMap的实现原理?

- HashMap ：HashMap 于 Map 口 供 作， 允 使 null值 null 不保 ， 别 不保 久不变
- HashMap ： Java 中， 两 ， 个 ，另 个 （ ），以 两个 ，HashMap也不例 HashMap 上 个“ 列” ，即 体
- HashMap 于Hash
    1. 们 HashMap中put元 ，利 key hashCode hash 出 前 元 中 下
    2. 储 ， 出 hash值 key， 两 况

(1) key ，则 原 值;

(2) key不 （出 冲 ），则 前 key-value 入 中

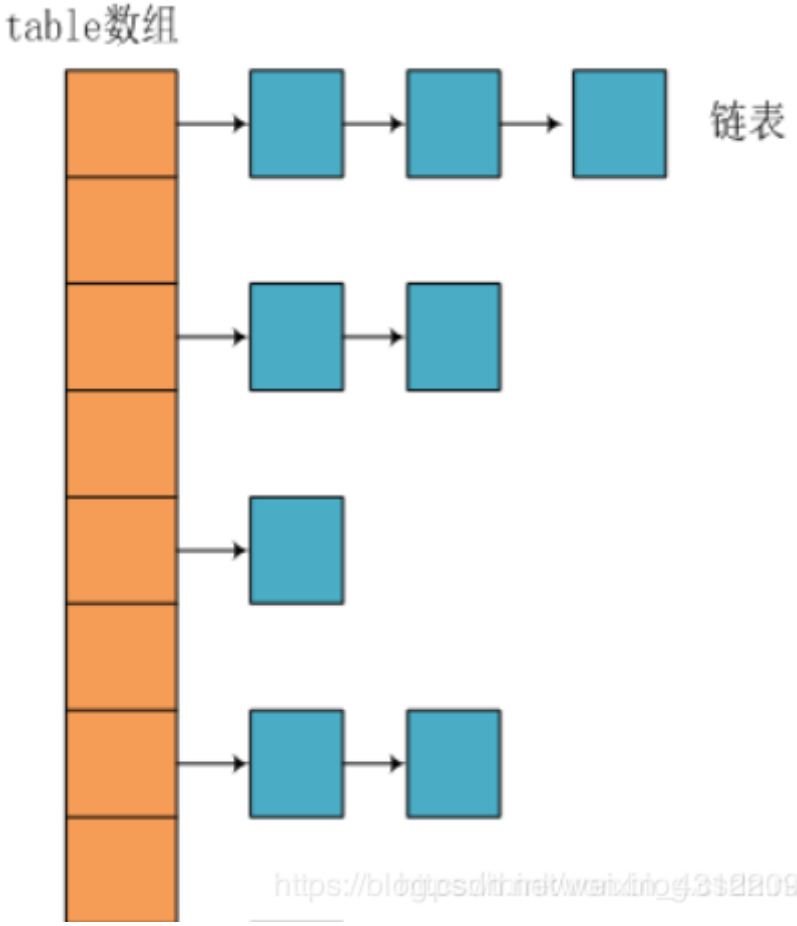
    3. 取 ， 到hash值 下 ， 判 key ，从 到 值
    4. 了以上 不 HashMap 何 决hash冲 ， 使 了 储 ， 冲 key 入 中， 发 冲 中做
- Jdk 1.8中 HashMap 做了优化， 中 八个之 ， 会为 ，从原 O(n)到O(logn)

30. HashMap在JDK1.7和JDK1.8中有哪些不同？HashMap的底层实现

- Java中，保 两 单 ：**数组的特点是：寻址容易，插入和删除困难；链表的特点是：寻址困难，但插入和删除容易；** 以 们 ，发 两 优势，使 叫做**拉链法** 以 决 冲

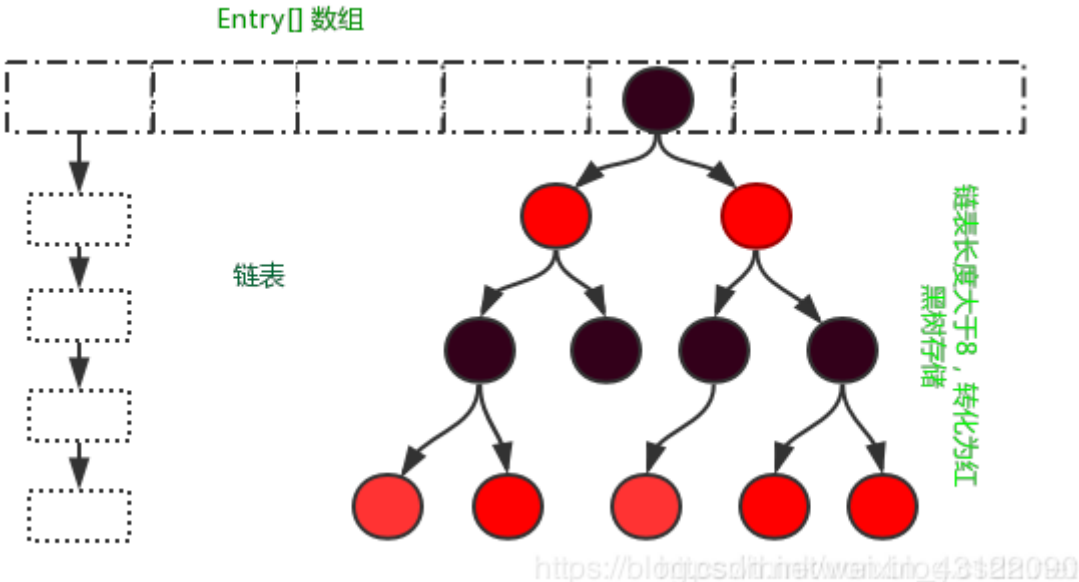
HashMap JDK1.8之前

- JDK1.8之前 **拉链法**： 也 创 个 ， 中 个 到 冲 ，则 冲 值加到 中即



HashMap JDK1.8之后

- 于之前 ，jdk1.8 决 冲 了 变化， 于 值（ 为8） ， 化为 ，以减



JDK1.7 VS JDK1.8 比较

- JDK1.8主 决 优化了 下 ：

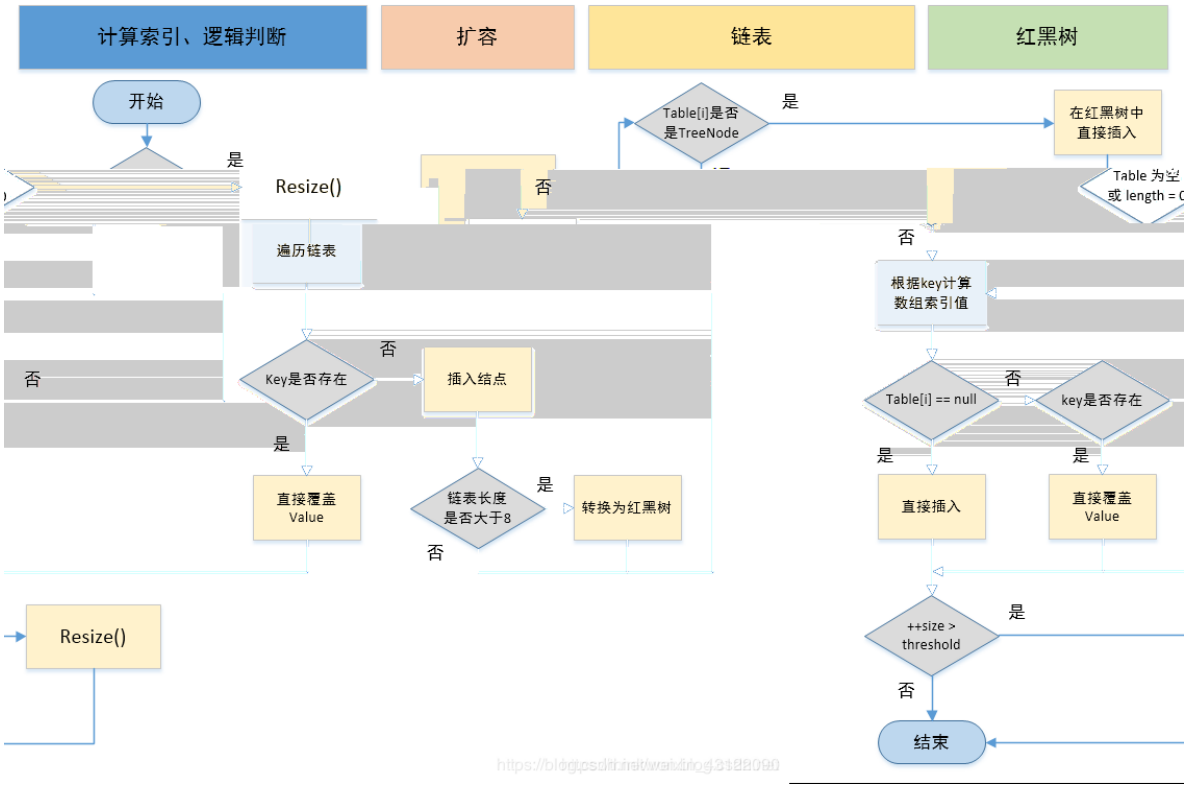
1. resize 优化
2. 入了 ， 免单 ， 参
3. 决了 ，但仍 全 ， 会 丢

不同	JDK 1.7	JDK 1.8
储	+	+ +
初 化	单 函 ： inflateTable()	到了 函

”e nÕi‹ i…’°}8@ !? — f i@

32. HashMap的put方法的具体流程?

- 们put 候， 先 `key` `hash`值， 了`hash` ，`hash` `key.hashCode()`与`key.hashCode()>>>16` 作， 16bit 0， 个 0 不变， 以hash函 作 ：**高16bit不变，低16bit和高16bit做了一个异或，目的是减少碰撞** 函 ， 为bucket 2 ， 下 `index = (table.length - 1) & hash`， 不做hash ， 于 列 只 几个低bit位，为了减 列 ， 了 作 之 ，使 16bit 低16bit 单 减 ， 且JDK8中了 O(logn) 升 下
- putVal




```
public V put(K key, V value) {
    return putVal(hash(key), key, value, false, true);
}

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

//   Map.put
final V putVal(int hash, K key, V value, boolean onlyIfAbsent,
               boolean evict) {
    Node<K,V>[] tab; Node<K,V> p; int n, i;
    //     ①  tab
 // table               0
  if ((tab = table) == null || (n = tab.length) == 0)
        n = (tab = resize()).length;
    //     ②     index     null
 // (n - 1) & hash                                        (
   )
    if ((p = tab[i = (n - 1) & hash]) == null)
        tab[i] = newNode(hash, key, value, null);
```

```
  //
 else {
       Node<K,V> e; K k;
       //    ③     key            value
       //                (          )  hash       key
 if (p.hash == hash &&
           ((k = p.key) == key || (key != null && key.equals(k))))
               //                e    e
 e = p;
       //    ④
// hash          key
 //                TreeNode              putTreeVal          node, e     null
       else if (p instanceof TreeNode)
           //
 e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);
       //    ⑤
//
 else {
           //
 for (int binCount = 0; ; ++binCount) {
               //

 //
 if ((e = p.next) == null) {
                   //
 p.next = newNode(hash, key, value, null);
                   //                                          8
                   if (binCount >= TREEIFY_THRESHOLD - 1) // -1 for 1st
                       //
 treeifyBin(tab, hash);
                   //
 break;
               }
               //              key             key
 if (e.hash == hash &&
                   ((k = e.key) == key || (key != null && key.equals(k))))
                   //
 break;
               //                        e = p.next
 p = e;
           }
       }
       //        key                           hash    key             value

 if (e != null) {
           //    e  value
           V oldValue = e.value;
           // onlyIfAbsent  false        null
           if (!onlyIfAbsent || oldValue == null)
               //
 e.value = value;
           //
 afterNodeAccess(e);
           //
 return oldValue;
       }
   }
   //
```

```
  ++modCount;
    //    ⑥
 //
  if (++size > threshold)
        resize();
    //
  afterNodeInsertion(evict);
    return null;
}
```

1. 判 值 table[i] 为 为null， 则 resize() ；
2. 值key hash值 到 入 i， table[i]==null， 加，⑥， table[i]不为 ， ③；
3. 判 table[i] 个元 key ， value， 则 ④，hashCode以及equals；
4. 判 table[i] 为treeNode，即table[i] ， ，则 中 入 值 ， 则 5；
5. 历table[i]，判 于8， 于8 为 ， 中 入 作， 则 入 作； 历 中 发 key value即 ；
6. 入 功 ，判 值 size 了 threshold， ，

33. HashMap的扩容操作是怎么实现的?

1. jdk1.8中，resize hashmap中 值 于 值 初 化 ， resize ；
2. 候， 2倍；
3. Node 位 么 原位 ， 么 动到原偏 两倍 位

- putVal()中， 们 到 个函 使 到了2 resize() ，resize() 初 化 会 其 ， 于其临 值值(为12)， 个 候 也会伴 上 元 分发， 也 JDK1.8 个优化 ， 1.7中， 之 去 其Hash值， Hash值 其 分发，但 1.8 中，则 个 位 中 判 (e.hash & oldCap) 为0， hash分 ， 元 位 么停 原 位 ， 么 动到原 位 + 加 个位 上

```
  final Node<K,V>[] resize() {
      Node<K,V>[] oldTab = table;//oldTab    hash
int oldCap = (oldTab == null) ? 0 : oldTab.length;
      int oldThr = threshold;
      int newCap, newThr = 0;
      if (oldCap > 0) {//   oldCap             hash
if (oldCap >= MAXIMUM_CAPACITY) {//
threshold = Integer.MAX_VALUE;
              return oldTab;//
}//       hash                                  oldCap        16
          else if ((newCap = oldCap << 1) < MAXIMUM_CAPACITY &&
                   oldCap >= DEFAULT_INITIAL_CAPACITY)
              newThr = oldThr << 1; // double threshold           threshold
      }
      //         0    threshold                  cap     threshold
     2  n
//
```

```
else if (oldThr > 0) // initial capacity was placed in threshold
          newCap = oldThr;
      //             map              threshold 16, 16*0.75
      else {               // zero initial threshold signifies using
defaults
          newCap = DEFAULT_INITIAL_CAPACITY;
          newThr = (int)(DEFAULT_LOAD_FACTOR * DEFAULT_INITIAL_CAPACITY);
      }
      //    threshold =    cap * 0.75
      if (newThr == 0) {
          float ft = (float)newCap * loadFactor;
          newThr = (newCap < MAXIMUM_CAPACITY && ft <
(float)MAXIMUM_CAPACITY ?
                    (int)ft : Integer.MAX_VALUE);
      }
      threshold = newThr;
      //                                table
      @SuppressWarnings({"rawtypes","unchecked"})
          Node<K,V>[] newTab = (Node<K,V>[])new Node[newCap];//   hash
table = newTab;//                   hash
//                          resize

if (oldTab != null) {
          //
for (int j = 0; j < oldCap; ++j) {
              Node<K,V> e;
              if ((e = oldTab[j]) != null) {
                  //                       e
   GC
oldTab[j] = null;
                  //    e.next==null
if (e.next == null)
                      //       hash
newTab[e.hash & (newCap - 1)] = e;
                  //    e  TreeNode   e.next!=null
else if (e instanceof TreeNode)
                      ((TreeNode<K,V>)e).split(this, newTab, j, oldCap);
                  // e            e.next!=null
else { // preserve order
                      // loHead,loTail                          1
                      Node<K,V> loHead = null, loTail = null;
                      // hiHead,hiTail                       1
                      Node<K,V> hiHead = null, hiTail = null;
                      Node<K,V> next;
                      //
do {
                          next = e.next;
                          if ((e.hash & oldCap) == 0) {
                              if (loTail == null)
                                  //      head             e  e
                 loHead
                                  //
loHead = e;
                              else
                                  // loTail.next      e
                                  loTail.next = e;
                              // loTail           e
```

```
                                //          loTail  loHead
loTail.next
//                next                    lowHead.next.next.....
                                //    loTail        lowHead

loTail = e;
                            }
                            else {
                                if (hiTail == null)
                                    //      head             e,        hiHead

hiHead = e;
                                else
                                    hiTail.next = e;
                                hiTail = e;
                            }
                        } while ((e = next) != null);
                        //        ,   tail    null

if (loTail != null) {
                            loTail.next = null;
                            newTab[j] = loHead;
                        }
                        if (hiTail != null) {
                            hiTail.next = null;
                            newTab[j + oldCap] = hiHead;
                        }
                    }
                }
            }
        }
        return newTab;
    }
```

34. HashMap是怎么解决哈希冲突的?

- : 决 个 之前, 们 先 **什么是哈希冲突**, 了 冲 之前 们 **什么是哈希** ;

什么是哈希?

- Hash, 为“ 列”, 也 为“ ” , Hash 使 任 二 制 为 二 制值, 个 二 制值叫做 值

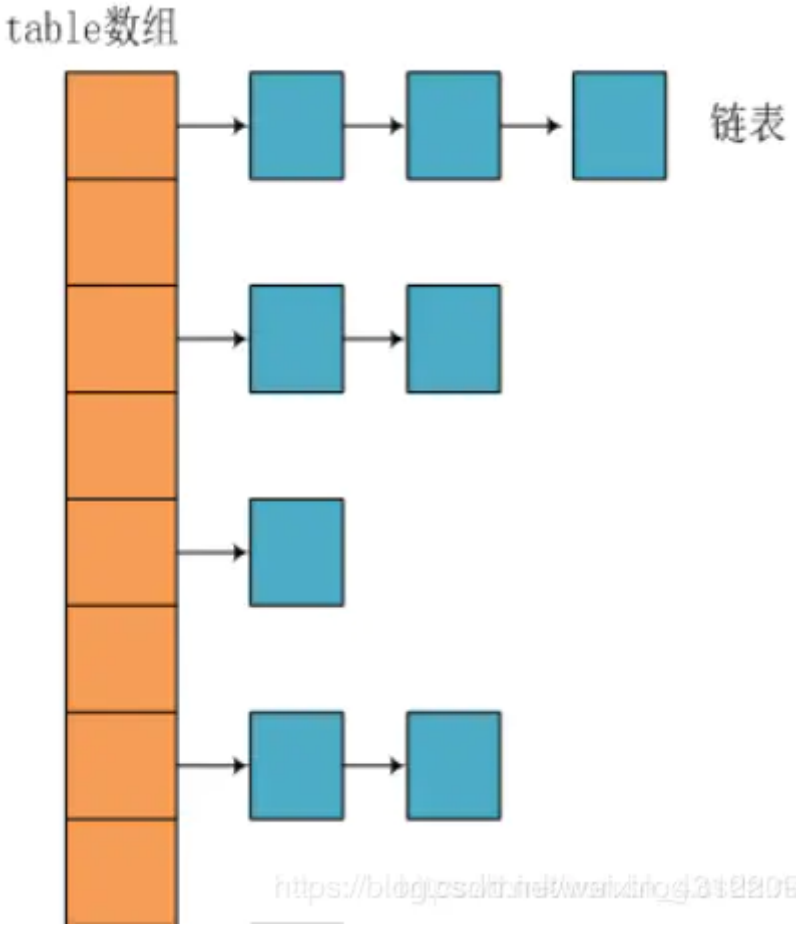
什么是哈希冲突?

- **当两个不同的输入值，根据同一散列函数计算出相同的散列值的现象，我们就把它叫做碰撞（哈希碰撞）**

HashMap的数据结构

- Java中, 保 两 单 :
  - : , 入 删 ;
  - : , 但 入 删 ;

- 以 们 ，发 两 优势， 以使 ：
以 决 冲 ：



- hash值 个 hash值 位;
- 个 ， 个 位 占 况下 下 个 以使 位
- **但相比于hashCode返回的int类型，我们HashMap初始的容量大小 `DEFAULT_INITIAL_CAPACITY = 1 << 4`（即2的四次方16）要远小于int类型的范围，所以我们如果只是单纯的用hashCode取余来获取对应的bucket这将会大大增加哈希碰撞的概率，并且最坏情况下还会将HashMap变成一个单链表**， 以 们 hashCode作 优化

hash()函数

- 上 到 ，主 为 使 hashCode取余， 么 于**参与运算的只有hashCode的低位**， 位 到任何作 ， 以 们 hashCode取值出 位也参与 ， 低hash ，使 分 ， 们 作 为**扰动**， **JDK 1.8**中 hash()函 下:

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

- **JDK 1.7**中， 为 ，**相比在1.7中的4次位运算，5次异或运算（9次扰动），在1.8中，只进行了1次位运算和1次异或运算（2次扰动）**；

总结

- 单 下HashMap 使 了 些 决 冲 :
  - hash值 个 hash值 位;
  - 个 , 个 位 占 况下 下 个 以使 位

35. 能否使用任何类作为 Map 的 key?

以使 任何 作为 Map key, 使 之前, 以下几 :

- 写了 equals() , 也 写 hashCode()
- 例 与 equals() hashCode() 关 则
- 个 使 equals(), 不 hashCode() 中使
- 义 Key 佳 使之为不 变 , hashCode() 值 以 , 不 变 也 以 保 hashCode() equals() 不会 变, 会 决与 变 关了

36. 为什么HashMap中String、Integer这样的包装类适合作为K?

- : String Integer 包 保 Hash值 不 准 , 减 Hash 几
  - final , 即不 变 , 保 key 不 , 不会 取hash值不 况
  - 内 写了`equals()` `hashCode()` , 了HashMap内 (不 以 去上 putValue), 不 出 Hash值 况;

37. 如果使用Object作为HashMap的Key，应该怎么办呢?

- : 写`hashCode()` `equals()`
  1. **重写`hashCode()`是因为需要计算存储数据的存储位置**, 不 从 列 中 个 关 分 , 但 会 Hash ;
  2. **重写`equals()`方法**, 反 传 以及 于任何 null 值x，x.equals(null) false 几个 , **目的是为了保证key在哈希表中的唯一性**;

38. HashMap为什么不直接使用hashCode()处理后的哈希值直接作为table的下标?

- : `hashCode()` int , 其 为-(2 ^ 31)~(2 ^ 31 - 1), 40亿个 , HashMap 16（初 化 值）~2 ^ 30，HashMap 况下 取不到 值 , 且 上也 以 供 么 储 , 从 `hashCode()` 出 值 不 内, 匹 储位 ;
- **那怎么解决呢?**
  1. HashMap 了 `hash()` , 两 动使 值 低位 , 低 也使 分 ;
  2. 保 为2 候，使 `hash()` 之 值与 (&) (- 1) 取 下 储, 取余 作 加 , 二 也 为只

为2 ，h&(length-1) 价于h%length，三 决了“ 值与不匹 ” ；

39. HashMap 的长度为什么是2的幂次方

- 为了 HashMap 取 ， ，也 分 匀， 个 / 个 到 个 / 中
- **这个算法应该如何设计呢?**
  - 们 先 会 到 %取余 作 但 ， 了：“取余(%) 作中 2 则 价于与其 减 与(&) 作（也 hash%length==hash&(length-1) 前 length 2 n ；） ” 且 二 制位 作 &， 于% ， 了 HashMap 为什么 2
- **那为什么是两次扰动呢?**
  - ： 加 值低位 ，使 分 匀，从 储下 位 & 匀 ， 减 Hash冲 ，两 了， 到了 位低位 参与 ；

40. HashMap 与 HashTable 有什么区别?

1. **线程安全**： HashMap 全 ，HashTable 全 ；HashTable 内 `synchronized` 修 （ 你 保 全 使 ConcurrentHashMap ）；
2. **效率**： 为 全 ，HashMap HashTable 另 ，HashTable ，不 代 中使 ；（ 你 保 全 使 ConcurrentHashMap ）；
3. **对Null key 和Null value的支持**： HashMap 中，null 以作为 ， 只 个， 以 个 个 值为 null 但 HashTable 中 put 值只 个 null，NullPointerException
4. **初始容量大小和每次扩充容量大小的不同** ：
5. 创 不 初 值，Hashtable 初 为11，之 充， 变为原 2n+1 HashMap 初 化 为16 之 充， 变为原 2倍
6. 创 了 初 值， 么 Hashtable 会 使 你 ， HashMap 会 其 充为2 也 HashMap 使 2 作为 ， 会介 到为什么 2
7. **底层数据结构**： JDK1.8 以 HashMap 决 冲 了 变化， 于 值（ 为8） ， 化为 ，以减 Hashtable 制
8. 使 ： Hashtable 以 到，Hashtable 保 不 使 ， 单 下使 HashMap 代， 使 则 ConcurrentHashMap 代

41. 什么是TreeMap 简介

- TreeMap 个**有序的key-value集合**，
- TreeMap 于**红黑树（Red-Black tree）实现** **其键的自然顺序进行排序**，**创建映射时提供的 Comparator 进行排序**，具体取决于使
- TreeMap **非同步**

42. 如何决定使用 HashMap 还是 TreeMap?

- 于 Map中 入 删 位元 作，HashMap ，假 你 个 key 历，TreeMap 于你 collection ，也 HashMap中 加元 会 ， map 为TreeMap key 历

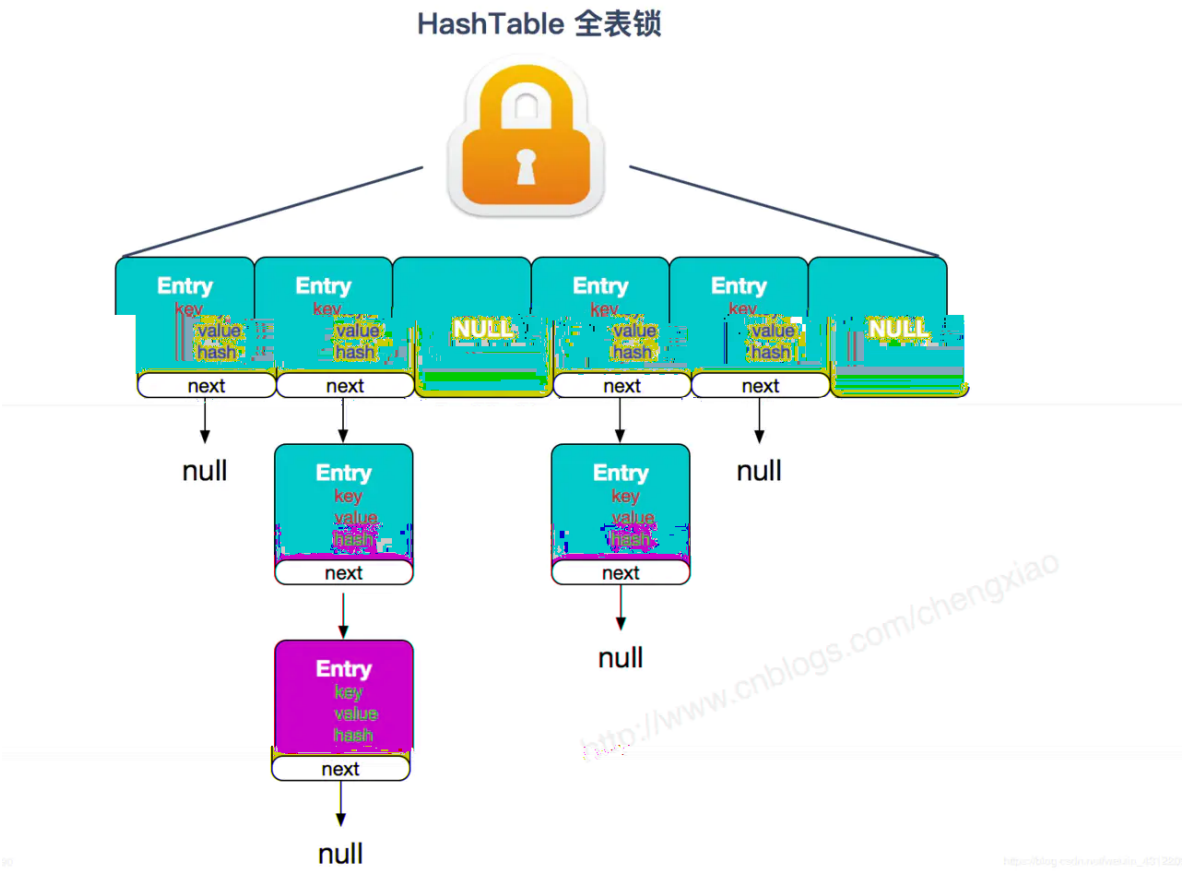
43. HashMap 和 ConcurrentHashMap 的区别

1. ConcurrentHashMap 个 了分割分 (Segment)， 个分 上 lock 保 ， 于HashTable synchronized 了 些， 发 ， HashMap 制，不 全 (JDK1.8之 ConcurrentHashMap 了 全 ,利 CAS)
2. HashMap 值 允 null，但 ConCurrentHashMap 不允

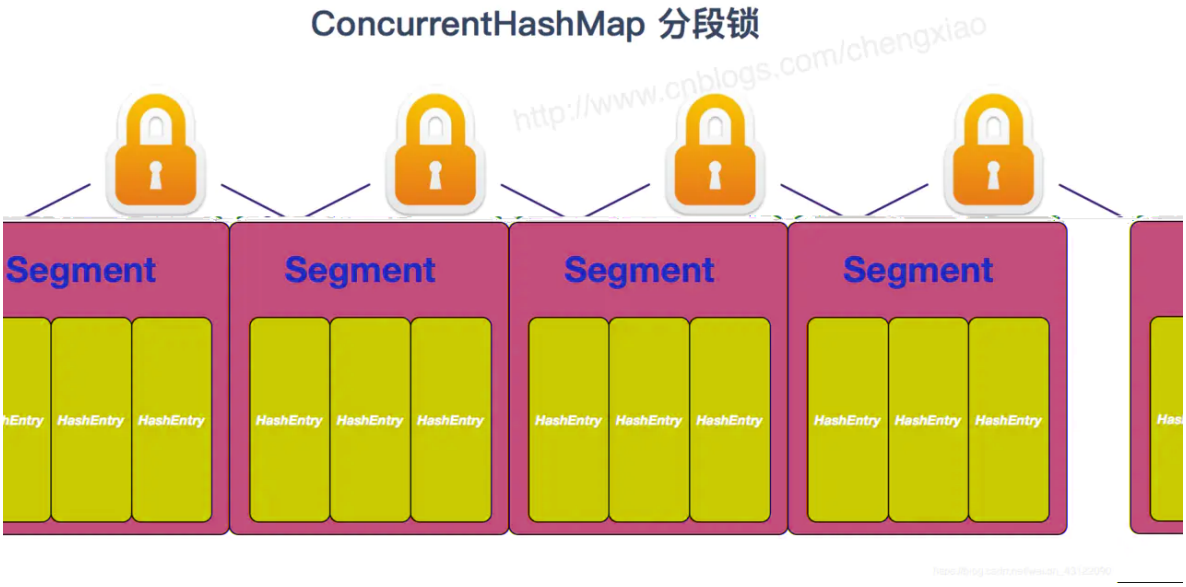
44. ConcurrentHashMap 和 Hashtable 的区别?

- ConcurrentHashMap Hashtable 区别主 体 全 上不
  - **底层数据结构**: JDK1.7 ConcurrentHashMap **分段的数组+链表** ，JDK1.8 HashMap1.8 ， + / 二叉 Hashtable JDK1.8 之前 HashMap 似 **数组+链表** ， HashMap 主 体， 则 主 为了 决 冲 ；
  - **实现线程安全的方式**:
  1. **在JDK1.7的时候，ConcurrentHashMap（分段锁）** 个 了分割分 (Segment)， 只 其中 分 ， 不 ， 不会 争， 发 (分 16个Segment， Hashtable 16 倍 ） **到了 JDK1.8 的时候已经摒弃了Segment的概念，而是直接用 Node 数组+链表+红黑树的数据结构来实现，并发控制使用 synchronized 和 CAS 来操作。 (JDK1.6以后 对synchronized锁做了很多优化)** 个 像 优化 且 全 HashMap，JDK1.8中 到 Segment ，但 化了 ，只 为了兼 ；
  2. ② **Hashtable(同一把锁)** :使 synchronized 保 全， 低下 个 ，其他 也 ， 会 入 ， 使 put 加元 ，另 个 不 使 put 加元 ，也不 使 get， 争会 低
- **两者的对比图**:

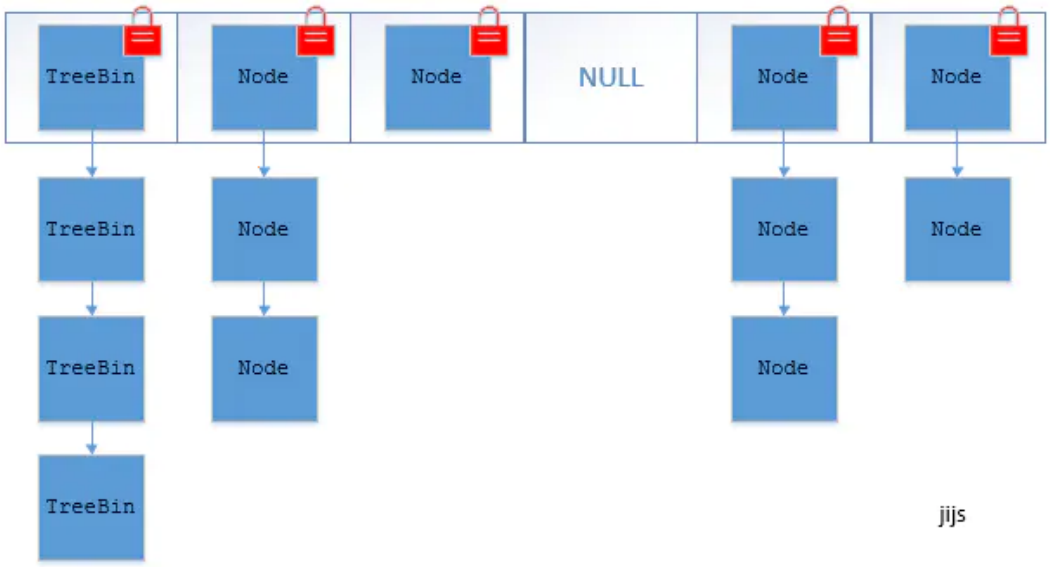
1、HashTable:



2、 JDK1.7的ConcurrentHashMap：



3、JDK1.8的ConcurrentHashMap（TreeBin: 红黑二叉树节点 Node: 链表节点）：

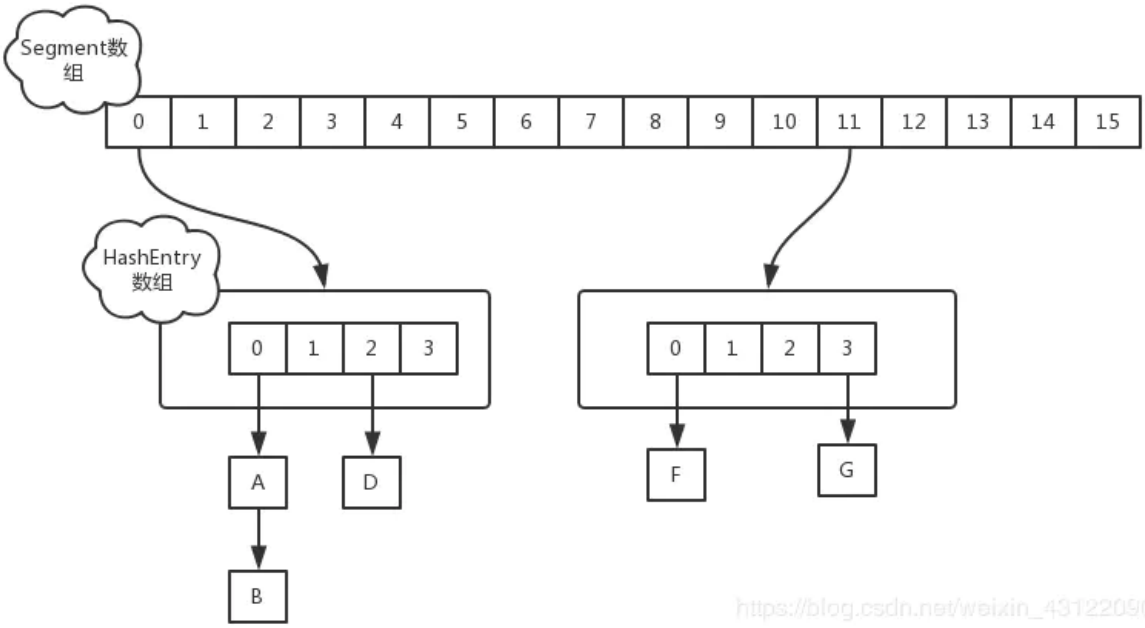


- ：ConcurrentHashMap 了HashMap HashTable 二 优势 HashMap ，HashTable 了 使 了synchronized 关 ， 以HashTable 住 个 ConcurrentHashMap

45. ConcurrentHashMap 底层具体实现知道吗？实现原理是什么？

JDK1.7

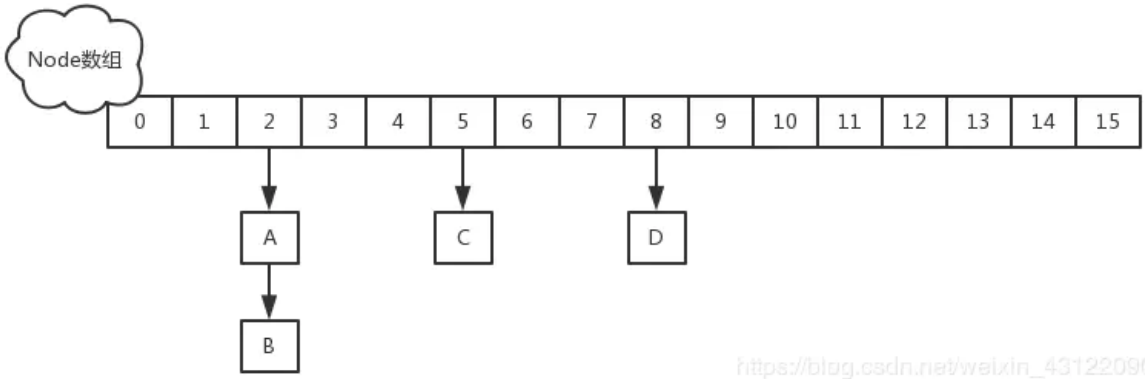
- 先 分为 储， ， 个 占 其中 个 ，其他 也 其他
- JDK1.7中，ConcurrentHashMap Segment + HashEntry ， 下：
- 个ConcurrentHashMap 包 个Segment Segment HashMap 似， ， 个Segment包 个HashEntry ， 个HashEntry 个 元 ， 个Segment 个HashEntry 元 ， HashEntry 修 ， 先 Segment



1. 包 两个 内 HashEntry Segment；前 值 ， 充 ；
2. Segment 入 ReentrantLock， 个Segment 个HashEntry 元 ， HashEntry 修 ， 先 Segment

JDK1.8

- **JDK1.8中，放弃了Segment臃肿的设计，取而代之的是采用Node + CAS + Synchronized来保证并发安全进行实现**，synchronized只 前 二叉 ， 只 hash不冲 ， 不会产 发， 又 升N倍
- 下:



- **附加源码，有需要的可以看看**
- 入元 （ 去 ）：
- 位 Node 初 化，则 CAS 入 ；

```
else if ((f = tabAt(tab, i = (n - 1) & hash)) == null) {
    if (casTabAt(tab, i, null, new Node<K,V>(hash, key, value, null)))
        break;                   // no lock when adding to empty bin
}
```

- 位 Node不为 ，且 前 不 于 动 ，则 加synchronized ， hash不 于0，则 历 入 ；

```
if (fh >= 0) {
    binCount = 1;
    for (Node<K,V> e = f;; ++binCount) {
        K ek;
        if (e.hash == hash &&
            ((ek = e.key) == key ||
             (ek != null && key.equals(ek)))) {
            oldVal = e.val;
            if (!onlyIfAbsent)
                e.val = value;
            break;
        }
        Node<K,V> pred = e;
        if ((e = e.next) == null) {
            pred.next = new Node<K,V>(hash, key, value, null);
            break;
        }
    }
}
```

1. TreeBin ，，则 putTreeVal 中 入 ； binCount不为0， put 作 产 了 ， 前 个 到8个，则 treeifyBin 化为 ， oldVal不为 ， 作， 元 个 产 ，则 值；
2. 入 个 ，则 addCount() 元 个 baseCount；

辅助工具类

46. Array 和 ArrayList 有何区别?

- Array 以 储 ，ArrayList 只 储
- Array ， ArrayList 动
- Array 内 ArrayList ， addAll removeAll iteration 只 ArrayList

47. 如何实现 Array 和 List 之间的转换?

- Array List： Arrays. asList(array)；
- List Array：List toArray()

48. comparable 和 comparator的区别?

- comparable 口 上 出 java.lang包， 个 compareTo(Object obj)

- comparator 口 上 出 java.util 包, 个compare(Object obj1, Object obj2)

- 们 个 使 义 , 们 写compareTo compare , 们 个 两 , 个song 中 分别 , 们 以 写compareTo 使 制 Comparator 以两个 Comparator , 二 代 们只 使 两个参 Collections.sort().

49. Collection 和 Collections 有什么区别?

- java.util.Collection 个 口 (个 口) 供了 作 口 Collection 口 Java 中 具体 Collection 口 义 为 具体 供了 化 作 , 其 口 List与Set
- Collections则 个 具 / 助 , 其中 供了 列 , 于 中元 以及 全 作

50. TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具类中的 sort()方法如何比较元素?

- TreeSet Comparable 口, 口 供了 元 compareTo() , 入元 会 元 TreeMap 值 Comparable 口从 元
- Collections 具 sort 两 ,
- 传入 中 Comparable 口以 元 ;

?

- comparable 口 上 出 java.lang包, 个 compareTo(Object obj)
- comparator 口 上 出 java.util 包, 个compare(Object obj1, Object obj2)

- 们 个 使 义 , 们 写compareTo compare , 们 个 两 , 个song 中 分别 , 们 以 写compareTo 使 制 Comparator 以两个 Comparator , 二 代 们只 使 两个参 Collections.sort().

51. Collection 和 Collections 有什么区别?

- java.util.Collection 个 口 (个 口) 供了 作 口 Collection 口 Java 中 具体 Collection 口 义 为 具体 供了 化 作 , 其 口 List与Set
- Collections则 个 具 / 助 , 其中 供了 列 , 于 中元 以及 全 作

52. TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具类中的 sort()方法如何比较元素?

- TreeSet Comparable 口， 口 供了 元 compareTo() ， 入元 会 元 TreeMap 值 Comparable 口从 元
- Collections 具 sort 两 ，
- 传入 中 Comparable 口以 元 ；
- 二 不 制 中 元 ，但 传入 二个参 ，参 Comparator 口 （ 写 compare 元 ）， 于 个临 义 则，其 口 入 元 ，也 (Java 中 函)