# 1 集合

## 1.1 HashMap

1. 存储结构

   数组、链表、红黑树（JDK1.8）

2. 特点

   快速存储、快速查找、可伸缩(负载因子0.75 扩容2倍)

   负载因子较小：浪费存储空间

   负载因子较大：哈希冲突概率增大，降低效率

3. hash算法

   hashCode ^ (hashCode>>16)

4. 数组下标计算

   hash%16

5. 哈希冲突

   不同对象计算出的数组下标相等。

   使用单向链表解决哈希冲突，链表长度大于8，转红黑树

## 1.2 HashTable和ConcurrentHashMap实现线程安全

HashTable: 一把synchronized

ConcurrentHashMap：分段锁 jdk1.8及以后分段颗粒度减小

1. **equals() 与 == 的区别是什么**
   - equals()：判断两个对象是否相等
     - 类没有覆盖 equals() 方法。则通过 equals() 比较该类的两个对象时，等价于通过“==”比较这两个对象。
     - 类覆盖了 equals() 方法。一般，我们都覆盖 equals() 方法来比较两个对象的内容是否相等；若它们的内容相等，则返回 true (即，认为这两个对象相等)。
   - ==： 判断两个对象的地址是不是相等。即，判断两个对象是不是同一个对象
2. **hashCode() 的作用是什么**

   hashCode() 的作用是**获取哈希码**，也称为散列码；它实际上是返回一个int整数。这个**哈希码的作用**是确定该对象在哈希表中的索引位置。 仅仅当创建并某个“类的散列表”(关于“散列表”见下面说明)时，该类的hashCode() 才有用 。
3. **hashCode() 和 equals() 之间有什么联系**
   - **第一种 不创建“类对应的散列表”**

     “hashCode() 和 equals() ”没有关系
   - **第二种 创建“类对应的散列表”**

在这种情况下，该类的“hashCode() 和 equals() ”是有关系的：
1)、如果两个对象相等，那么它们的hashCode()值一定相同。
这里的相等是指，通过equals()比较两个对象时返回true。
2)、如果两个对象hashCode()相等，它们并不一定相等。
因为在散列表中，hashCode()相等，即两个键值对的哈希值相等。然而哈希值相等，并不一定能得 出键值对相等。补充说一句：“两个不同的键值对，哈希值相等”，这就是哈希冲突。

hashCode()相等，通过equals()比较它们也返回true，认为对象相等。

# 1.3 Arrays.asList()使用指南

`Arrays.asList()` 方法返回的并不是 `java.util.ArrayList` ，而是 `java.util.Arrays` 的一个内部类,这个内部类并没有实现集合的修改方法或者说并没有重写这些方法。

- 如何正确的将数组转换为ArrayList
  - 自己动手实现
  - 最简便的方法(推荐)

    List list = new ArrayList<>(Arrays.asList("a", "b", "c"))
  - 使用 Java8 的Stream(推荐)

    Integer [] myArray = { 1, 2, 3 };
    List myList = Arrays.stream(myArray).collect(Collectors.toList());
    //基本类型也可以实现转换（依赖boxed的装箱操作）
    int [] myArray2 = { 1, 2, 3 };
    List myList = Arrays.stream(myArray2).boxed().collect(Collectors.toList());
  - 使用 Guava(推荐)
    - 于不可变集合，你可以使用 `ImmutableList` 类及其 `of()` 与 `copyOf()` 工厂方法：（参数不能为空）

      ```
      List<String> il = ImmutableList.of("string", "elements");  // from varargs
      List<String> il = ImmutableList.copyOf(aStringArray);      // from array
      ```

    - 对于可变集合，你可以使用 `Lists` 类及其 `newArrayList()` 工厂方法

      List l1 = Lists.newArrayList(anotherListOrCollection);  // from collection
      List l2 = Lists.newArrayList(aStringArray);             // from array
      List l3 = Lists.newArrayList("or", "string", "elements"); // from varargs

# 1.4 Collection.toArray()`方法使用的坑&如何反转数组

- 该方法是一个泛型方法：`T[] toArray(T[] a);` 如果 `toArray` 方法中没有传递任何参数的话返回的是 `Object` 类型数组。

```
String [] s= new String[]{
    "dog", "lazy", "a", "over", "jumps", "fox", "brown", "quick", "A"
};
List<String> list = Arrays.asList(s);
Collections.reverse(list);
s=list.toArray(new String[0]);//没有指定类型的话会报错
```

由于JVM优化，`new String[0]`作为`Collection.toArray()`方法的参数现在使用更好，`new String[0]`就是起一个模板的作用，指定了返回数组的类型，0是为了节省空间，因为它只是为了说明返回的类型。

# 1.5 Java集合框架常见问题

## 1.5.1 说说List,Set,Map三者的区别

**一、结构特点**
List和Set是存储单列数据的集合，Map是存储键值对这样的双列数据的集合；
List中存储的数据是有顺序的，并且值允许重复；Map中存储的数据是无序的，它的键是不允许重复的，但是值是允许重复的；Set中存储的数据是无顺序的，并且不允许重复，但元素在集合中的位置是由元素的hashcode决定，即位置是固定的（Set集合是根据hashcode来进行数据存储的，所以位置是固定的，但是这个位置不是用户可以控制的，所以对于用户来说set中的元素还是无序的）。
**二、实现类**
List接口有三个实现类：
1.1 LinkedList
基于链表实现，链表内存是散列的，增删快，查找慢；
1.2 ArrayList
基于数组实现，非线程安全，效率高，增删慢，查找快；
1.3 Vector
基于数组实现，线程安全，效率低，增删慢，查找慢；
Map接口有四个实现类：
2.1 HashMap
基于 hash 表的 Map 接口实现，非线程安全，高效，支持 null 值和 null
键；
2.2 HashTable
线程安全，低效，不支持 null 值和 null 键；
2.3 LinkedHashMap
是 HashMap 的一个子类，保存了记录的插入顺序；
2.4 SortMap 接口
TreeMap，能够把它保存的记录根据键排序，默认是键值的升序排序

Set接口有两个实现类：
3.1 HashSet
底层是由 Hash Map 实现，不允许集合中有重复的值，使用该方式时需要重写 equals()和 hash Code()方法；
3.2 LinkedHashSet
继承于 HashSet，同时又基于 LinkedHashMap 来进行实现，底层使用的是 LinkedHashMap
三、区别

1. List 集合中对象按照索引位置排序，可以有重复对象，允许按照对象在集合中的索引位置检索对象，例如通过list.get(i)方法来获取集合中的元素；
2. Map 中的每一个元素包含一个键和一个值，成对出现，键对象不可以重复，值对象可以重复；
3. Set 集合中的对象不按照特定的方式排序，并且没有重复对象，但它的实现类能对集合中的对象按照特定的方式排序，例如 Tree Set 类，可以按照默认顺序，也可以通过实现 Java.util.Comparator< Type >接口来自定义排序方式。

```
 Set set = new HashSet();
Iterator it = set.iterator();
while (it.hasNext()) {
 String str = it.next();
```

```
 System.out.println(str);
}
```

## 1.5.2 Arraylist 与 LinkedList 区别