

# Java【集合】面试题

---

以下面试题，基于网络整理，和自己编辑。具体参考的文章，会在文末给出所有的链接。

如果胖友有自己的疑问，欢迎在星球提问，我们一起整理吊吊的 Java【集合】面试题的大保健。

而题目的难度，尽量按照从容易到困难的顺序，逐步下去。

[Java 集合的脑图](#)

## 基础

---

### Java 集合框架有哪些？

---

Java 集合框架，可以看看 [《Java 集合框架》](#) 文章。

 说出一些集合框架的优点？

集合框架的部分优点如下：

- 1、使用核心集合类降低开发成本，而非实现我们自己的集合类。
- 2、随着使用经过严格测试的集合框架类，代码质量会得到提高。
- 3、通过使用 JDK 附带的集合类，可以降低代码维护成本。
- 4、复用性和可操作性。

 集合框架中的泛型有什么优点？

Java5 引入了泛型，所有的集合接口和实现都大量地使用它。泛型允许我们为集合提供一个可以容纳的对象类型。因此，如果你添加其它类型的任何元素，它会在编译时报错。这避免了在运行时出现 `ClassCastException`，因为你将会在编译时得到报错信息。

泛型也使得代码整洁，我们不需要使用显式转换和 `instanceOf` 操作符。它也给运行时带来好处，因为不会产生类型检查的字节码指令。

### Java 集合框架的基础接口有哪些？

---

- `Collection`，为集合层级的根接口。一个集合代表一组对象，这些对象即为它的元素。Java 平台不提供这个接口任何直接的实现。
  - `Set`，是一个不能包含重复元素的集合。这个接口对数学集合抽象进行建模，被用来代表集合，就如一副牌。
  - `List`，是一个有序集合，可以包含重复元素。你可以通过它的索引来访问任何元素。`List` 更像长度动态变换的数组。
- `Map`，是一个将 `key` 映射到 `value` 的对象。一个 `Map` 不能包含重复的 `key`，每个 `key` 最多只能映射一个 `value`。
- 一些其它的接口有 `Queue`、`Deque`、`SortedSet`、`SortedMap` 和 `ListIterator`。

 为何 `Collection` 不从 `Cloneable` 和 `Serializable` 接口继承？

Collection 接口指定一组对象，对象即为它的元素。

- 如何维护这些元素由 Collection 的具体实现决定。例如，一些如 List 的 Collection 实现允许重复的元素，而其它的如 Set 就不允许。
- 很多 Collection 实现有一个公有的 clone 方法。然而，把它放到集合的所有实现中也是没有意义的。这是因为 Collection 是一个抽象表现，重要的是实现。

当与具体实现打交道的时候，克隆或序列化的语义和含义才发挥作用。所以，具体实现应该决定如何对它进行克隆或序列化，或它是否可以被克隆或序列化。在所有的实现中授权克隆和序列化，最终导致更少的灵活性和更多的限制，**特定的实现应该决定它是否可以被克隆和序列化。**

### 🔗 为何 Map 接口不继承 Collection 接口？

尽管 Map 接口和它的实现也是集合框架的一部分，但 Map 不是集合，集合也不是 Map。因此，Map 继承 Collection 毫无意义，反之亦然。

如果 Map 继承 Collection 接口，那么元素去哪儿？Map 包含 key-value 对，它提供抽取 key 或 value 列表集合(Collection)的方法，但是它不适合“一组对象”规范。

### 🔗 Collection 和 Collections 的区别？

- Collection，是集合类的上级接口，继承与他的接口主要有 Set 和 List。
- Collections，是针对集合类的一个工具类，它提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

### 🔗 集合框架里实现的通用算法有哪些？

Java 集合框架提供常用的算法实现，比如排序和搜索。

Collections 类包含这些方法实现。大部分算法是操作 List 的，但一部分对所有类型的集合都是可用的。部分算法有排序、搜索、混编、最大最小值。

### 🔗 集合框架底层数据结构总结

#### 1) List

- ArrayList：Object 数组。
- Vector：Object 数组。
- LinkedList：双向链表(JDK6 之前为循环链表，JDK7 取消了循环)。

#### 2) Map

- HashMap：
  - JDK8 之前，HashMap 由数组+链表组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的（“拉链法”解决冲突）。
  - JDK8 以后，在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 8）时，将链表转化为红黑树，以减少搜索时间。
  - 不是用红黑树来管理 hashmap，而是在 hash 值相同的情况下（且重复数量大于 8），用红黑树来管理数据。红黑树相当于排序数据。可以自动的使用二分法进行定位。性能较高。一般情况下，hash 值做的比较好的话基本上用不到红黑树。
  - 当个数不多的时候，直接链表遍历更方便，实现起来也简单。而红黑树的实现要复杂的多。

因为红黑树需要进行左旋，右旋操作，而单链表不需要，以下都是单链表与红黑树结构对比。如果元素小于 8 个，查询成本高，新增成本低 如果元素大于 8 个，查询成本低，新增成本高

- LinkedHashMap：LinkedHashMap 继承自 HashMap，所以它的底层仍然是基于拉链式散列结构即由数组和链表或红黑树组成。另外，LinkedHashMap 在上面结构的基础上，增加了一条双向链表，使得上面的结构可以保持键值对的插入顺序。同时通过对链表进行相应的操作，实现了访问顺序相关逻辑。详细可以查看：[《LinkedHashMap 源码详细分析 \(JDK1.8\)》](#)。
- Hashtable：数组+链表组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的。
- TreeMap：红黑树（自平衡的排序二叉树）。

### 3) Set

- HashSet：无序，唯一，基于 HashMap 实现的，底层采用 HashMap 来保存元素。
- LinkedHashSet：LinkedHashSet 继承自 HashSet，并且其内部是通过 LinkedHashMap 来实现的。有点类似于我们之前说的 LinkedHashMap 其内部是基于 HashMap 实现一样，不过还是有一点点区别的。
- TreeSet：无序，唯一，红黑树(自平衡的排序二叉树)。

## 什么是迭代器(Iterator)?

Iterator 接口，提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代器实例的迭代方法。迭代器可以在迭代的过程中删除底层集合的元素，但是不可以直接调用集合的 `#remove(Object obj)` 方法删除，可以通过迭代器的 `#remove()` 方法删除。

### 🔗 Iterator 和 ListIterator 的区别是什么？

- Iterator 可用来遍历 Set 和 List 集合，但是 ListIterator 只能用来遍历 List。
- Iterator 对集合只能是前向遍历，ListIterator 既可以前向也可以后向。
- ListIterator 实现了 Iterator 接口，并包含其他的功能。比如：增加元素，替换元素，获取前一个和后一个元素的索引等等。

### 🔗 快速失败 (fail-fast) 和安全失败 (fail-safe) 的区别是什么？

差别在于 ConcurrentModification 异常：

- 快速失败：当你在迭代一个集合的时候，如果有另一个线程正在修改你正在访问的那个集合时，就会抛出一个 ConcurrentModification 异常。在 `java.util` 包下的都是快速失败。
- 安全失败：你在迭代的时候会去底层集合做一个拷贝，所以你在修改上层集合的时候是不会受影响的，不会抛出 ConcurrentModification 异常。在 `java.util.concurrent` 包下的全是安全失败的。

### 🔗 如何删除 List 中的某个元素？

有两种方式，分别如下：

- 方式一，使用 Iterator，顺序向下，如果找到元素，则使用 remove 方法进行移除。
- 方式二，倒序遍历 List，如果找到元素，则使用 remove 方法进行移除。

### 🔗 Enumeration 和 Iterator 接口有什么不同？

- Enumeration 跟 Iterator 相比较快两倍，而且占用更少的内存。
- 但是，Iterator 相对于 Enumeration 更安全，因为其他线程不能修改当前迭代器遍历的集合对象。同时，Iterators 允许调用者从底层集合中移除元素，这些 Enumerations 都没法完成。

对于很多胖友，可能并未使用过 Enumeration 类，所以可以看看 [《Java Enumeration 接口》](#) 文章。

### 🔗 为何 Iterator 接口没有具体的实现？

Iterator 接口，定义了遍历集合的方法，但它的实现则是集合实现类的责任。每个能够返回用于遍历的 Iterator 的集合类都有它自己的 Iterator 实现内部类。

这就允许集合类去选择迭代器是 fail-fast 还是 fail-safe 的。比如，ArrayList 迭代器是 fail-fast 的，而 CopyOnWriteArrayList 迭代器是 fail-safe 的。

## Comparable 和 Comparator 的区别？

- Comparable 接口，在 `java.lang` 包下，用于当前对象和其它对象的比较，所以它有一个 `#compareTo(Object obj)` 方法用来排序，该方法只有一个参数。
- Comparator 接口，在 `java.util` 包下，用于传入的两个对象的比较，所以它有一个 `#compare(Object obj1, Object obj2)` 方法用来排序，该方法有两个参数。

详细的，可以看看 [《Java 自定义比较器》](#) 文章，重点是如何自己实现 Comparable 和 Comparator 的方法。

### 🔗 compareTo 方法的返回值表示的意思？

- 大于 0，表示对象大于参数对象。
- 小于 0，表示对象小于参数对象
- 等于 0，表示两者相等。

### 🔗 如何对 Object 的 List 排序？

- 对 `Object[]` 数组进行排序时，我们可以用 `Arrays#sort(...)` 方法。
- 对 `List<Object>` 数组进行排序时，我们可以用 `Collections#sort(...)` 方法。

## 有哪些关于 Java 集合框架的最佳实践？

- 基于应用的需求来选择使用正确类型的集合，这对性能来说是非常重要的。例如，如果元素的大小是固定的，并且知道优先级，我们将会使用一个 Array，而不是 ArrayList。
- 一些集合类允许我们指定他们的初始容量。因此，如果我们知道存储数据的大概数值，就可以避免重散列或者大小的调整。
- 总是使用泛型来保证类型安全，可靠性和健壮性。同时，使用泛型还可以避免运行时的 ClassCastException 异常。
- 在 Map 中使用 JDK 提供的不可变类作为一个 key，这样可以避免 hashCode 的实现和我们自定义类的 equals 方法。
- 应该依照接口而不是实现来编程。
- 返回零长度的集合或者数组，而不是返回一个 `null`，这样可以防止底层集合是空的。

## 区别

### List 和 Set 区别？

List, Set 都是继承自 Collection 接口。

- 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。

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。

Set 和 List 对比：

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

## List 和 Map 区别？

- List 是对象集合，允许对象重复。
- Map 是键值对的集合，不允许 key 重复。

## Array 和 ArrayList 有何区别？什么时候更适合用 Array？

- Array 可以容纳基本类型和对象，而 ArrayList 只能容纳对象。
- Array 是指定大小的，而 ArrayList 大小是固定的，可自动扩容。
- Array 没有提供 ArrayList 那么多功能，比如 `addAll`、`removeAll` 和 `iterator` 等。

尽管 ArrayList 明显是更好的选择，但也有些时候 Array 比较好用，比如下面的三种情况。

- 1、如果列表的大小已经指定，大部分情况下是存储和遍历它们
- 2、对于遍历基本数据类型，尽管 Collections 使用自动装箱来减轻编码任务，在指定大小的基本类型的列表上工作也会变得很慢。
- 3、如果你要使用多维数组，使用 `[][]` 比 List 会方便。

## ArrayList 与 LinkedList 区别？

### ArrayList

- 优点：ArrayList 是实现了基于动态数组的数据结构，因为地址连续，一旦数据存储好了，查询操作效率会比较高（在内存里是连着放的）。
- 缺点：因为地址连续，ArrayList 要移动数据，所以插入和删除操作效率比较低。

### LinkedList

- 优点：LinkedList 基于链表的数据结构，地址是任意的，所以在开辟内存空间的时候不需要等一个连续的地址。对于新增和删除操作 `add` 和 `remove`，LinkedList 比较占优势。LinkedList 适用于要头尾操作或插入指定位置的场景。
- 缺点：因为 LinkedList 要移动指针，所以查询操作性能比较低。

### 适用场景分析：

- 当需要对数据进行对随机访问的情况下，选用 ArrayList。
- 当需要对数据进行多次增加删除修改时，采用 LinkedList。

如果容量固定，并且只会添加到尾部，不会引起扩容，优先采用 ArrayList。

- 当然，绝大多数业务的场景下，使用 ArrayList 就够了。主要是，注意好避免 ArrayList 的扩容，以及非顺序的插入。

### ArrayList 是如何扩容的？

直接看 [《ArrayList 动态扩容详解》](#) 文章，很详细。主要结论如下：

- 如果通过无参构造的话，初始数组容量为 0，当真正对数组进行添加时，才真正分配容量。每次按照 1.5 倍（位运算）的比率通过 `copyOf` 的方式扩容。
- 在 JDK6 中实现是，如果通过无参构造的话，初始数组容量为 10，每次通过 `copyOf` 的方式扩容后容量为原来的 1.5 倍。

重点是 1.5 倍扩容，这是和 `HashMap` 2 倍扩容不同的地方。

### 🔗 ArrayList 集合加入 1 万条数据，应该怎么提高效率？

`ArrayList` 的默认初始容量为 10，要插入大量数据的时候需要不断扩容，而扩容是非常影响性能的。因此，现在明确了 10 万条数据了，我们可以直接在初始化的时候就设置 `ArrayList` 的容量！

这样就可以提高效率了~

## ArrayList 与 Vector 区别？

`ArrayList` 和 `Vector` 都是用数组实现的，主要有这么三个区别：

- 1、`Vector` 是多线程安全的，线程安全就是说多线程访问同一代码，不会产生不确定的结果，而 `ArrayList` 不是。这个可以从源码中看出，`Vector` 类中的方法很多有 `synchronized` 进行修饰，这样就导致了 `Vector` 在效率上无法与 `ArrayList` 相比。

`Vector` 是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用。

- 2、两个都是采用的线性连续空间存储元素，但是当空间不足的时候，两个类的增加方式是不同。
- 3、`Vector` 可以设置增长因子，而 `ArrayList` 不可以。

适用场景分析：

- 1、`Vector` 是线程同步的，所以它也是线程安全的，而 `ArrayList` 是线程无需同步的，是不安全的。如果不考虑到线程的安全因素，一般用 `ArrayList` 效率比较高。

实际场景下，如果需要多线程访问安全的数组，使用 `CopyOnWriteArrayList`。

- 2、如果集合中的元素的数目大于目前集合数组的长度时，在集合中使用数据量比较大的数据，用 `Vector` 有一定的优势。

这种情况下，使用 `LinkedList` 更合适。

## HashMap 和 Hashtable 的区别？


`Hashtable` 是在 Java 1.0 的时候创建的，而集合的统一规范命名是在后来的 Java 2.0 开始约定的，而当时其他一部分集合类的发布构成了新的集合框架。

- `Hashtable` 继承 `Dictionary`，`HashMap` 继承的是 Java 2 出现的 `Map` 接口。
- 2、`HashMap` 去掉了 `Hashtable` 的 `contains` 方法，但是加上了 `containsValue` 和 `containsKey` 方法。
- 3、`HashMap` 允许空键值，而 `Hashtable` 不允许。
- 【重点】4、`Hashtable` 是同步的，而 `HashMap` 是非同步的，效率上比 `Hashtable` 要高。也因此，`HashMap` 更适合于单线程环境，而 `Hashtable` 适合于多线程环境。
- 5、`HashMap` 的迭代器（`Iterator`）是 fail-fast 迭代器，`Hashtable` 的 `enumerator` 迭代器不是 fail-fast 的。
- 6、`Hashtable` 中数组默认大小是 11，扩容方法是 `old * 2 + 1`，`HashMap` 默认大小是 16，扩容每次为 2 的指数大小。



一般现在不建议用 `HashTable` 。主要原因是两点：

- 一是，`HashTable` 是遗留类，内部实现很多没优化和冗余。
- 二是，即使在多线程环境下，现在也有同步的 `ConcurrentHashMap` 替代，没有必要因为是多线程而用 `HashTable` 。

 **Hashtable 的 #size() 方法中明明只有一条语句 "return count;"，为什么还要做同步？**

同一时间只能有一条线程执行固定类的同步方法，但是对于类的非同步方法，可以多条线程同时访问。所以，这样就有问题了，可能线程 A 在执行 `Hashtable` 的 `put` 方法添加数据，线程 B 则可以正常调用 `#size()` 方法读取 `Hashtable` 中当前元素的个数，那读取到的值可能不是最新的，可能线程 A 添加了完了数据，但是没有对 `count++`，线程 B 就已经读取 `count` 了，那么对于线程 B 来说读取到的 `count` 一定是不准确的。

而给 `#size()` 方法加了同步之后，意味着线程 B 调用 `#size()` 方法只有在线程 A 调用 `put` 方法完毕之后才可以调用，这样就保证了线程安全性。

## HashSet 和 HashMap 的区别？

- `Set` 是线性结构，值不能重复。`HashSet` 是 `Set` 的 `hash` 实现，`HashSet` 中值不能重复是用 `HashMap` 的 `key` 来实现的。
- `Map` 是键值对映射，可以空键空值。`HashMap` 是 `Map` 的 `hash` 实现，`key` 的唯一性是通过 `key` 值 `hashCode` 的唯一来确定，`value` 值则是链表结构。

因为不同的 `key` 值，可能有相同的 `hashCode`，所以 `value` 值需要是链表结构。

他们的共同点都是 `hash` 算法实现的唯一性，他们都不能持有基本类型，只能持有对象。

为了更好的性能，`Netty` 自己实现了 `key` 为基本类型的 `HashMap`，例如 [IntObjectHashMap](#)。

## HashSet 和 TreeMap 的区别？

- `HashSet` 是用一个 `hash` 表来实现的，因此，它的元素是无序的。添加，删除和 `HashSet` 包括的方法的持续时间复杂度是  $O(1)$ 。
- `TreeSet` 是用一个树形结构实现的，因此，它是有序的。添加，删除和 `TreeSet` 包含的方法的持续时间复杂度是  $O(\log n)$ 。

 **如何决定选用 HashMap 还是 TreeMap？**

- 对于在 `Map` 中插入、删除和定位元素这类操作，`HashMap` 是最好的选择。
- 然而，假如你需要对一个有序的 `key` 集合进行遍历，`TreeMap` 是更好的选择。

基于你的 `collection` 的大小，也许向 `HashMap` 中添加元素会更快，再将 `HashMap` 换为 `TreeMap` 进行有序 `key` 的遍历。

## HashMap 和 ConcurrentHashMap 的区别？

`ConcurrentHashMap` 是线程安全的 `HashMap` 的实现。主要区别如下：

- 1、`ConcurrentHashMap` 对整个桶数组进行了分割分段(`Segment`)，然后在每一个分段上都用 `lock` 锁进行保护，相对于 `Hashtable` 的 `syn` 关键字锁的粒度更精细了一些，并发性能更好。而 `HashMap` 没有锁机制，不是线程安全的。

JDK8 之后，`ConcurrentHashMap` 启用了一种全新的方式实现,利用 `CAS` 算法。

- 2、HashMap 的键值对允许有 `null`，但是 `ConcurrentHashMap` 都不允许。

## 队列和栈是什么，列出它们的区别？

栈和队列两者都被用来预存储数据。

```
java.util.Queue
```

是一个接口，它的实现类在Java并发包中。

- 队列允许先进先出（FIFO）检索元素，但并非总是这样。
- Deque 接口允许从两端检索元素。
- 栈与队列很相似，但它允许对元素进行后进先出（LIFO）进行检索。
  - Stack 是一个扩展自 Vector 的类，而 Queue 是一个接口。

## 原理

### HashMap 的工作原理是什么？

我们知道在 Java 中最常用的两种结构是数组和模拟指针（引用），几乎所有的数据结构都可以利用这两种来组合实现，HashMap 也是如此。实际上 HashMap 是一个“链表散列”。

HashMap 是基于 hashing 的原理。

#### [HashMap 图解](#)

- 我们使用 `#put(key, value)` 方法来存储对象到 HashMap 中，使用 `get(key)` 方法从 HashMap 中获取对象。
- 当我们给 `#put(key, value)` 方法传递键和值时，我们先对键调用 `#hashCode()` 方法，返回的 hashCode 用于找到 bucket 位置来储存 Entry 对象。

#### 🔗 当两个对象的 hashCode 相同会发生什么？

因为 hashCode 相同，所以它们的 bucket 位置相同，“碰撞”会发生。

因为 HashMap 使用链表存储对象，这个 Entry（包含有键值对的 Map.Entry 对象）会存储在链表中。

#### 🔗 hashCode 和 equals 方法有何重要性？

HashMap 使用 key 对象的 `#hashCode()` 和 `#equals(Object obj)` 方法去决定 key-value 对的索引。当我们试着从 HashMap 中获取值的时候，这些方法也会被用到。

- 如果这两个方法没有被正确地实现，在这种情况下，两个不同 Key 也许会产生相同的 `#hashCode()` 和 `#equals(Object obj)` 输出，HashMap 将会认为它们是相同的，然后覆盖它们，而非把它们存储到不同的地方。

同样的，所有不允许存储重复数据的集合类都使用 `#hashCode()` 和 `#equals(Object obj)` 去查找重复，所以正确实现它们非常重要。`#hashCode()` 和 `#equals(Object obj)` 方法的实现，应该遵循以下规则：

- 如果 `o1.equals(o2)`，那么 `o1.hashCode() == o2.hashCode()` 总是为 `true` 的。
- 如果 `o1.hashCode() == o2.hashCode()`，并不意味着 `o1.equals(o2)` 会为 `true`。

#### 🔗 HashMap 默认容量是多少？



默认容量都是 16，负载因子是 0.75。就是当 HashMap 填充了 75% 的 bucket 是就会扩容，最小的可能性是  $(16 * 0.75 = 12)$ ，一般为原内存的 2 倍。

### 🔗 有哪些顺序的 HashMap 实现类？

- LinkedHashMap，是基于元素进入集合的顺序或者被访问的先后顺序排序。
- TreeMap，是基于元素的固有顺序(由 Comparator 或者 Comparable 确定)。

### 🔗 我们能否使用任何类作为 Map 的 key？

我们可以使用任何类作为 Map 的 key，然而在使用它们之前，需要考虑以下几点：

- 1、如果类重写了 equals 方法，它也应该重写 hashCode 方法。
- 2、类的所有实例需要遵循与 equals 和 hashCode 相关的规则。
- 3、如果一个类没有使用 equals，你不应该在 hashCode 中使用它。
- 4、用户自定义 key 类的最佳实践是使之成为不可变的，这样，hashCode 值可以被缓存起来，拥有更好的性能。不可变的类也可以确保 hashCode 和 equals 在未来不会改变，这样就会解决与可变相关的问题了。

比如，我有一个类 MyKey，在 HashMap 中使用它。代码如下：

```
//传递给MyKey的name参数被用于equals()和hashCode()中
MyKey key = new MyKey('Pankaj'); //assume hashCode=1234
myHashMap.put(key, 'value');
// 以下的代码会改变key的hashCode()和equals()值
key.setName('Amit'); //assume new hashCode=7890
//下面会返回null，因为HashMap会尝试查找存储同样索引的key，而key已被改变了，匹配失败，返回null
myHashMap.get(new MyKey('Pankaj'));
```

- 那就是为何 String 和 Integer 被作为 HashMap 的 key 大量使用。

### 🔗 HashMap 的长度为什么是 2 的幂次方？

为了能让 HashMap 存取高效，尽量较少碰撞，也就是要尽量把数据分配均匀，每个链表/红黑树长度大致相同。这个实现就是把数据存到哪个链表/红黑树中的算法。

这个算法应该如何设计呢？我们首先可能会想到采用 % 取余的操作来实现。但是，重点来了：

- 取余(%)操作中如果除数是 2 的幂次则等价于与其除数减一的与(&)操作（也就是说  $hash \% length == hash \& (length - 1)$  的前提是 length 是 2 的 n 次方；）。
- 并且，采用二进制位操作 &，相对于 % 能够提高运算效率，

这就解释了 HashMap 的长度为什么是 2 的幂次方。

## HashSet 的工作原理是什么？

HashSet 是构建在 HashMap 之上的 Set hashing 实现类。让我们直接撸下源码，代码如下：

```
// HashSet.java

private transient HashMap<E, Object> map;

private static final Object PRESENT = new Object();
```

- `map` 属性，当我们创建一个 `HashMap` 对象时，其内部也会创建一个 `map` 对象。后续 `HashSet` 所有的操作，实际都是基于这个 `map` 之上的封装。
- `PRESENT` 静态属性，所有 `map` 中 KEY 对应的值，都是它，避免重复创建。
- OK，再来看一眼 `add` 方法，代码如下：

```
// HashSet.java

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

- 是不是一目了然。

### 🔗 HashSet 如何检查重复？

：正如我们上面看到 `HashSet` 的实现原理，我们自然可以推导出，`HashMap` 也是如何检查重复滴。

如下摘取自《Head First Java》第二版：

当你把对象加入 `HashSet` 时，`HashSet` 会先计算对象的 `hashCode` 值来判断对象加入的位置，同时也会与其他加入的对象的 `hashCode` 值作比较。

- 如果没有相符的 `hashCode`，`HashSet` 会假设对象没有重复出现。
- 但是如果发现有相同 `hashCode` 值的对象，这时会调用 `equals` 方法来检查 `hashCode` 相等的对象是否真的相同。
  - 如果两者相同，`HashSet` 就不会让加入操作成功。
  - 如果两者不同，`HashSet` 就会让加入操作成功。

## EnumSet 是什么？

`java.util.EnumSet`，是使用枚举类型的集合实现。

- 当集合创建时，枚举集合中的所有元素必须来自单个指定的枚举类型，可以是显示的或隐式的。`EnumSet` 是不同步的，不允许值为 `null` 的元素。
- 它也提供了一些有用的方法，比如 `#copyOf(Collection c)`、`#of(E first, E... rest)` 和 `#complementOf(EnumSet s)` 方法。

关于 `EnumSet` 的源码解析，见 [《EnumSet 源码分析》](#) 文章。

## TODO TreeMap 原理

Java 中的 `TreeMap` 是使用红黑树实现的。

TODO `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？

等到源码解析后，在进行补充。

## Java Priority Queue 是什么？

`PriorityQueue` 是一个基于优先级堆的无界队列，它的元素都以他们的自然顺序有序排列。

- 在它创建的时候，我们可以提供一个比较器 Comparator 来负责 PriorityQueue 中元素的排序。
- PriorityQueue 不允许 ``null 元素，不允许不提供自然排序的对象，也不允许没有任何关联 Comparator 的对象。
- 最后，PriorityQueue 不是线程安全的，在执行入队和出队操作它需要  $O(\log(n))$  的时间复杂度。

### 🔗 poll 方法和 remove 方法的区别？

poll 和 remove 方法，都是从队列中取出一个元素，差别在于：

- poll 方法，在获取元素失败的时候会返回空
- remove() 方法，失败的时候会抛出异常。

### 🔗 LinkedHashMap 和 PriorityQueue 的区别是什么？

- PriorityQueue 保证最高或者最低优先级的元素总是在队列头部，LinkedHashMap 维持的顺序是元素插入的顺序。
- 当遍历一个 PriorityQueue 时，没有任何顺序保证，但是 LinkedHashMap 保证遍历顺序是元素插入的顺序。

## 彩蛋

---

参考与推荐如下文章：

- [《2018 年最新 Java 面试题及答案整理》](#)
- [《互联网常见的 14 个 Java 面试题》](#)