[https://mp.weixin.qq.com/s?\_\_biz=MzA5MTkxMDQ4MQ==&mid=2648933197&idx=1&sn=1ef33a6403680ee49b3acf22d4a4aa34&chksm=88621b73bf159265c8775bc7d80e44f68bc162b7301f5ac8dce9669d17643934404440b6560f&token=2027319240&lang=zh\_CN&scene=21#wechat\_redirect](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933197&idx=1&sn=1ef33a6403680ee49b3acf22d4a4aa34&chksm=88621b73bf159265c8775bc7d80e44f68bc162b7301f5ac8dce9669d17643934404440b6560f&token=2027319240&lang=zh_CN&scene=21" \l "wechat_redirect)

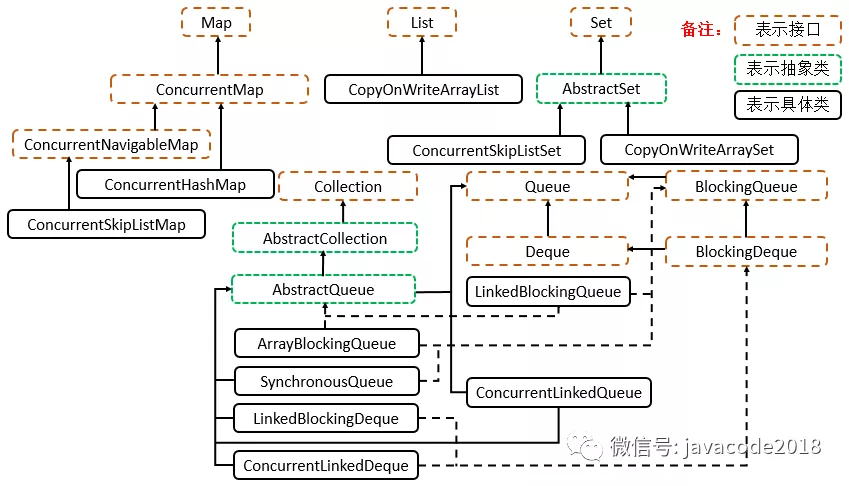
# JUC常见集合

## 本文内容

了解JUC常见集合，学会使用

* ConcurrentHashMap
* ConcurrentSkipListMap
* ConcurrentSkipListSet
* CopyOnWriteArraySet
* 介绍Queue接口
* ConcurrentLinkedQueue
* CopyOnWriteArrayList
* 介绍Deque接口
* ConcurrentLinkedDeque

## **JUC集合框架图**



图可以看到，JUC的集合框架也是从Map、List、Set、Queue、Collection等超级接口中继承而来的。所以，大概可以知道JUC下的集合包含了一一些基本操作，并且变得线程安全。

## **Map**

### ConcurrentHashMap

功能和HashMap基本一致，内部使用红黑树实现的。

**特性：**

* 迭代结果和存入顺序不一致
* key和value都不能为空
* 线程安全的

### ConcurrentSkipListMap

内部使用跳表实现的，放入的元素会进行排序，排序算法支持2种方式来指定：

* 通过构造方法传入一个Comparator
* 放入的元素实现Comparable接口

上面2种方式必选一个，如果2种都有，走规则1。

**特性：**

* 迭代结果和存入顺序不一致
* 放入的元素会排序
* key和value都不能为空
* 线程安全的

## **List**

### CopyOnWriteArrayList

实现List的接口的，一般我们使用ArrayList、LinkedList、Vector，其中只有Vector是线程安全的，可以使用Collections静态类的synchronizedList方法对ArrayList、LinkedList包装为线程安全的List，不过这些方式在保证线程安全的情况下性能都不高。

CopyOnWriteArrayList是线程安全的List，内部使用数组存储数据，集合中多线程并行操作一般存在4种情况：读读、读写、写写、写读，这个只有在写写操作过程中会导致其他线程阻塞，其他3种情况均不会阻塞，所以读取的效率非常高。

可以看一下这个类的名称：CopyOnWrite，意思是在写入操作的时候，进行一次自我复制，换句话说，当这个List需要修改时，并不修改原有内容（这对于保证当前在读线程的数据一致性非常重要），而是在原有存放数据的数组上产生一个副本，在副本上修改数据，修改完毕之后，用副本替换原来的数组，这样也保证了写操作不会影响读。

**特性：**

* 迭代结果和存入顺序一致
* 元素不重复
* 元素可以为空
* 线程安全的
* 读读、读写、写读3种情况不会阻塞；写写会阻塞
* 无界的

## **Set**

### ConcurrentSkipListSet

有序的Set，内部基于ConcurrentSkipListMap实现的，放入的元素会进行排序，排序算法支持2种方式来指定：

* 通过构造方法传入一个Comparator
* 放入的元素实现Comparable接口

上面2种方式需要实现一个，如果2种都有，走规则1

**特性：**

* 迭代结果和存入顺序不一致
* 放入的元素会排序
* 元素不重复
* 元素不能为空
* 线程安全的
* 无界的

### CopyOnWriteArraySet

内部使用CopyOnWriteArrayList实现的，将所有的操作都会转发给CopyOnWriteArrayList。

**特性：**

* 迭代结果和存入顺序不一致
* 元素不重复
* 元素可以为空
* 线程安全的
* 读读、读写、写读 不会阻塞；写写会阻塞
* 无界的

## **Queue**

Queue接口中的方法，我们再回顾一下：

| **操作类型** | **抛出异常** | **返回特殊值** |
| --- | --- | --- |
| 插入 | add(e) | offer(e) |
| 移除 | remove() | poll() |
| 检查 | element() | peek() |

3种操作，每种操作有2个方法，不同点是队列为空或者满载时，调用方法是抛出异常还是返回特殊值，大家按照表格中的多看几遍，加深记忆。

### ConcurrentLinkedQueue

高效并发队列，内部使用链表实现的。

**特性：**

* 线程安全的
* 迭代结果和存入顺序一致
* 元素可以重复
* 元素不能为空
* 线程安全的
* 无界队列

## **Deque**

先介绍一下Deque接口，双向队列(Deque)是Queue的一个子接口，双向队列是指该队列两端的元素既能入队(offer)也能出队(poll)，如果将Deque限制为只能从一端入队和出队，则可实现栈的数据结构。对于栈而言，有入栈(push)和出栈(pop)，遵循先进后出原则。

一个线性 collection，支持在两端插入和移除元素。名称 *deque* 是“double ended queue（双端队列）”的缩写，通常读为“deck”。大多数 Deque 实现对于它们能够包含的元素数没有固定限制，但此接口既支持有容量限制的双端队列，也支持没有固定大小限制的双端队列。

此接口定义在双端队列两端访问元素的方法。提供插入、移除和检查元素的方法。每种方法都存在两种形式：一种形式在操作失败时抛出异常，另一种形式返回一个特殊值（null 或 false，具体取决于操作）。插入操作的后一种形式是专为使用有容量限制的 Deque 实现设计的；在大多数实现中，插入操作不能失败。

下表总结了上述 12 种方法：



此接口扩展了 Queue接口。在将双端队列用作队列时，将得到 FIFO（先进先出）行为。将元素添加到双端队列的末尾，从双端队列的开头移除元素。从 Queue 接口继承的方法完全等效于 Deque 方法，如下表所示：

| **Queue 方法** | **等效 Deque 方法** |
| --- | --- |
| add(e) | addLast(e) |
| offer(e) | offerLast(e) |
| remove() | removeFirst() |
| poll() | pollFirst() |
| element() | getFirst() |
| peek() | peekFirst() |

## **ConcurrentLinkedDeque**

实现了Deque接口，内部使用链表实现的高效的并发双端队列。

**特性：**

* 线程安全的
* 迭代结果和存入顺序一致
* 元素可以重复
* 元素不能为空
* 线程安全的
* 无界队列

## **BlockingQueue**

关于阻塞队列，上一篇有详细介绍，可以看看：[掌握JUC中的阻塞队列](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933190&idx=1&sn=916f539cb1e695948169a358549227d3&chksm=88621b78bf15926e0a94e50a43651dab0ceb14a1fb6b1d8b9b75e38c6d8ac908e31dd2131ded&token=1963100670&lang=zh_CN&scene=21" \l "wechat_redirect" \t "https://mp.weixin.qq.com/_blank)