去岳不群公司面试，被刷新了三观！

大家好，我是小二呀。

今天心情很差，因为前天去了一家名叫岳不群的公司面试，没想到，公司高层的价值观有很大的问题，具体原因我现在没法说，等到时候这件事尘埃落定后，我一定会拉出来狠狠鞭尸。

透露一点，岳不群公司说他们为了一统江湖，可能会在 2025 年从华山搬迁至黑木崖。

真的是郁闷了一整天，情绪非常低落，于是就上 B站重温了一遍央视的《笑傲江湖》，不得不说，这部电视剧真的太经典了。

岳不群明明是个道貌岸然的坏人，却被称为江湖君子剑，背后搞的全部都是小动作，抢辟邪剑谱，嫁祸令狐冲，暗算定逸师太，暗算左冷禅，暗算林平之，最后差点暗算令狐冲。

亏我特么还去面试了，就为了学点紫霞神功。

对比岳不群，我反而倍加欣赏林平之和东方不败，一个为了寻仇，极度的扭曲，但却敢作敢为，比同样练了辟邪剑谱的岳不群强一万倍；而东方不败，同样练了葵花宝典，暗算了任我行，但对杨莲亭的爱，确实真真切切，看得我都感动了（）。

但有一说一，岳不群公司也不都是坏人，比如说宁中则面试官，就特别的温柔，问的问题也都特别在点子上，不信你们看看。

岳不群公司面经

hashmap 底层实现机制

JDK 8 中 HashMap 的数据结构是`数组`+`链表`+`红黑树`。

![三分恶面渣逆袭：JDK 8 HashMap 数据结构示意图](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/84dfb5e4b7ea48f983c78be44848651d~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1116&h=593&s=75881&e=png&b=fffcfc)

HashMap 的核心是一个动态数组（`Node[] table`），用于存储键值对。这个数组的每个元素称为一个“桶”（Bucket），每个桶的索引是通过对键的哈希值进行哈希函数处理得到的。

当多个键经哈希处理后得到相同的索引时，会发生哈希冲突。HashMap 通过链表来解决哈希冲突——即将具有相同索引的键值对通过链表连接起来。

不过，链表过长时，查询效率会比较低，于是当链表的长度超过 8 时（且数组的长度大于 64），链表就会转换为红黑树。红黑树的查询效率是 O(logn)，比链表的 O(n) 要快。数组的查询效率是 O(1)。

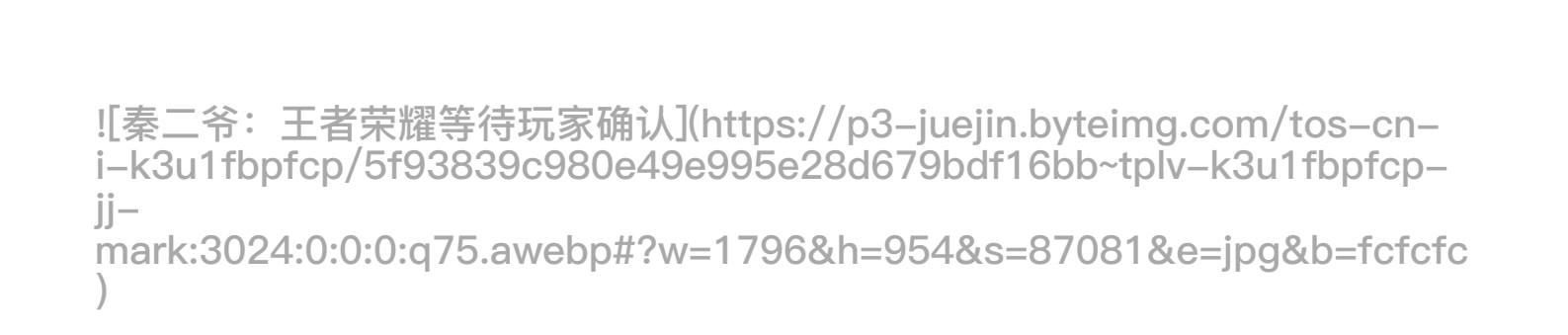
总的来说，HashMap 是一种通过哈希表实现的键值对集合，它通过将键哈希化成数组索引，并在冲突时使用链表或红黑树来存储元素，从而实现快速的查找、插入和删除操作。

并发编程CountDownLatch 和消息队列

CountDownLatch 是 JUC 包中的一个同步工具类，用于协调多个线程之间的同步。它允许一个或多个线程等待，直到其他线程中执行的一组操作完成。它通过一个计数器来实现，该计数器由线程递减，直到到达零。

* 初始化：创建 CountDownLatch 对象时，指定计数器的初始值。
* 等待（await）：一个或多个线程调用 await 方法，进入等待状态，直到计数器的值变为零。
* 倒计数（countDown）：其他线程在完成各自任务后调用 countDown 方法，将计数器的值减一。当计数器的值减到零时，所有在 await 上等待的线程会被唤醒，继续执行。

当等待多个线程完成各自的启动任务后再启动主线程的任务，就可以使用 CountDownLatch，以王者荣耀为例。



创建五个线程，分别代表大乔、兰陵王、安其拉、哪吒和铠等五个玩家。每个玩家都调用了`countDown()`方法，表示已经就位。主线程调用`await()`方法，等待所有玩家就位。

```
public static void main(String[] args) throws InterruptedException {
    CountDownLatch countDownLatch = new CountDownLatch(5);

    Thread daqiao = new Thread(() -> {
        System.out.println("大乔已就位！");
        countDownLatch.countDown();
    });
    Thread lanlingwang = new Thread(() -> {
        System.out.println("兰陵王已就位！");
        countDownLatch.countDown();
    });
    Thread anqila = new Thread(() -> {
        System.out.println("安其拉已就位！");
        countDownLatch.countDown();
    });
    Thread nezha = new Thread(() -> {
        System.out.println("哪吒已就位！");
        countDownLatch.countDown();
    });
    Thread kai = new Thread(() -> {
        System.out.println("铠已就位！");
        countDownLatch.countDown();
    });

    daqiao.start();
    lanlingwang.start();
    anqila.start();
    nezha.start();
    kai.start();

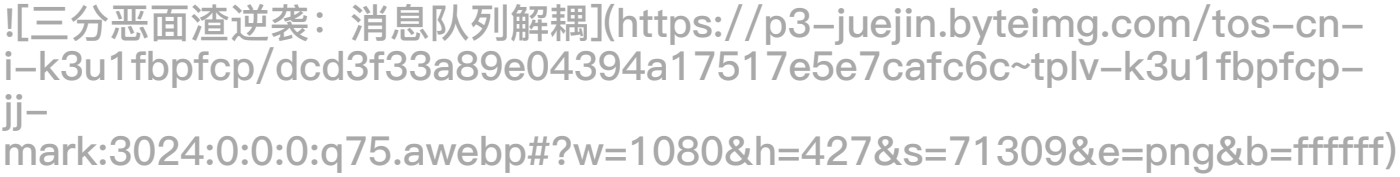
    countDownLatch.await();
    System.out.println("全员就位，开始游戏！");
}
```

```

消息队列（Message Queue, MQ）是一种非常重要的中间件技术，广泛应用于分布式系统中，以提高系统的可用性、解耦能力和异步通信效率。

①、**解耦**

生产者将消息放入队列，消费者从队列中取出消息，这样一来，生产者和消费者之间就不需要直接通信，生产者只管生产消息，消费者只管消费消息，这样就实现了解耦。

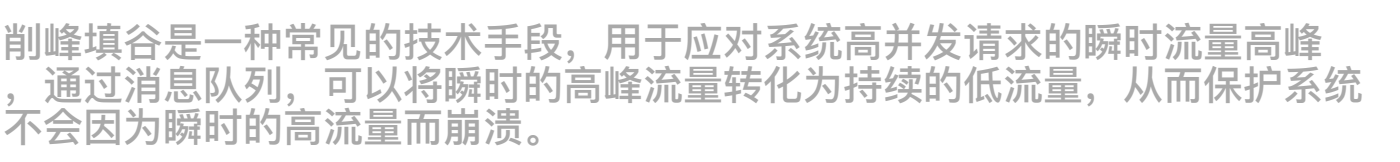



②、**异步**：

系统可以将那些耗时的任务放在消息队列中异步处理，从而快速响应用户的请求。


③、**削峰**：

削峰填谷是一种常见的技术手段，用于应对系统高并发请求的瞬时流量高峰，通过消息队列，可以将瞬时的高峰流量转化为持续的低流量，从而保护系统不会因为瞬时的高流量而崩溃。


```

java虚拟机的垃圾回收机制

垃圾回收（Garbage Collection，GC），顾名思义就是释放垃圾占用的空间，防止内存爆掉。有效的使用可以使用的内存，对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收。

JVM 在做垃圾回收之前，需要先搞清楚什么是垃圾，什么不是垃圾，那么就需要一种垃圾判断算法，通常有引用计数算法、可达性分析算法。

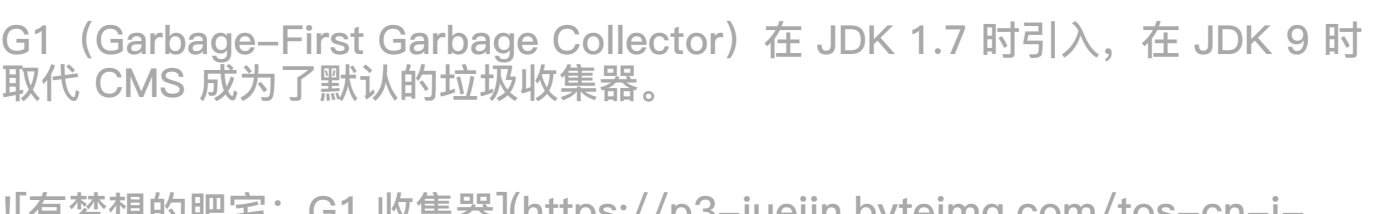
在确定了哪些垃圾可以被回收后，垃圾收集器要做的事情就是进行垃圾回收，如何高效地进行垃圾回收呢?

可以采用标记清除算法、复制算法、标记整理算法、分代收集算法等。

JVM 提供了多种垃圾回收器，不同的垃圾回收器适用于不同的场景和需求，包括 CMS GC、G1 GC、ZGC 等。

CMS 是第一个 GC 停顿时间（STW 的时间）的垃圾收集器，JDK 1.5 时引入，JDK9 被标记弃用，JDK14 被移除。

G1（Garbage-First Garbage Collector）在 JDK 1.7 时引入，在 JDK 9 时取代 CMS 成为了默认的垃圾收集器。



ZGC 是 JDK11 推出的一款低延迟垃圾收集器，适用于大内存低延迟服务的内存管理和回收，SPEC jbb 2015 基准测试，在 128G 的大堆下，最大停顿时间才 1.68 ms，停顿时间远胜于 G1 和 CMS。

讲一下mysql索引相关的知识，怎么创建索引，怎么进行索引优化，索引的底层结构

索引就好像书的目录，通过目录去查找对应的章节内容会比一页一页的翻书快很多。

可通过 `create index` 创建索引，比如：

```
create index idx_name on students(name);
```

尽管索引能提高查询性能，但不当的使用也会带来一系列问题。在加索引时需要注意以下几点：

①、选择合适的列作为索引

* 经常作为查询条件（WHERE 子句）、排序条件（ORDER BY 子句）、分组条件（GROUP BY 子句）的列是建立索引的好候选。
* 区分度低的字段，例如性别，不要建索引
* 频繁更新的字段，不要作为主键或者索引
* 不建议用无序的值(例如身份证、UUID)作为索引，当主键具有不确定性，会造成叶子节点频繁分裂，出现磁盘存储的碎片化

②、避免过多的索引

* 每个索引都需要占用额外的磁盘空间。
* 更新表（INSERT、UPDATE、DELETE 操作）时，所有的索引都需要被更新。
* 维护索引文件需要成本；还会导致页分裂，IO 次数增多。

③、利用前缀索引和索引列的顺序

* 对于字符串类型的列，可以考虑使用前缀索引来减少索引大小。
* 在创建复合索引时，应该根据查询条件将最常用作过滤条件的列放在前面。

MySQL 的默认存储引擎是 InnoDB，它采用的是 B+树索引，B+树是一种自平衡的多路查找树，和红黑树、二叉平衡树不同，B+树的每个节点可以有 m 个子节点，而红黑树和二叉平衡树都只有 2 个。

和 B 树不同，B+树的非叶子节点只存储键值，不存储数据，而叶子节点存储了所有的数据，并且构成了一个有序链表。

这样做的好处是，非叶子节点上由于没有存储数据，就可以存储更多的键值对，再加上叶子节点构成了一个有序链表，范围查询时就可以直接通过叶子节点间的指针顺序访问整个查询范围内的所有记录，而无需对树进行多次遍历。查询的效率会更高。

![用户1260737：B+树](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/844651edf6e348b58b066f64b1d928f4~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1080&h=737&s=355600&e=png&b=fefbfa)

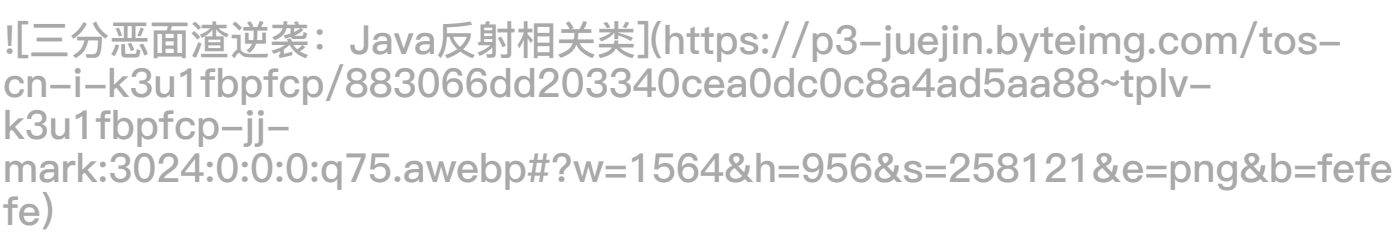
Java底层反射机制，哪些地方用到了反射

创建一个对象是通过 new 关键字来实现的，比如：

```
Person person = new Person();
```

Person 类的信息在编译时就确定了，那假如在编译期无法确定类的信息，但又想在运行时获取类的信息、创建类的实例、调用类的方法，这时候就要用到反射。

反射功能主要通过 `java.lang.Class` 类及 `java.lang.reflect` 包中的类如 Method, Field, Constructor 等来实现。



比如说我们可以装来动态加载类并创建对象：

```
String className = "java.util.Date";
Class<?> cls = Class.forName(className);
Object obj = cls.newInstance();
System.out.println(obj.getClass().getName());
```

反射有哪些应用场景?

①、Spring 框架就大量使用了反射来动态加载和管理 Bean。

```
Class<?> clazz = Class.forName("com.example.MyClass");
Object instance = clazz.newInstance();
```

②、Java 的动态代理（Dynamic Proxy）机制就使用了反射来创建代理类。代理类可以在运行时动态处理方法调用，这在实现 AOP 和拦截器时非常有用。

```
InvocationHandler handler = new MyInvocationHandler();
MyInterface proxyInstance = (MyInterface) Proxy.newProxyInstance(
    MyInterface.class.getClassLoader(),
    new Class<?>[] { MyInterface.class },
    handler
);
```

③、JUnit 和 TestNG 等测试框架使用反射机制来发现和执行测试方法。反射允许框架扫描类，查找带有特定注解（如 `@Test`）的方法，并在运行时调用它们。

```
Method testMethod = testClass.getMethod(”testSomething”);
testMethod.invoke(testInstance);
```

内容来源
---

* 星球嘉宾三分恶的面渣逆袭：[javabetter.cn/sidebar/san...](http://cxyroad.com/”https://javabetter.cn/sidebar/sanfene/nixi.html”)
* 二哥的 Java 进阶之路（GitHub 已有 12000+star）：[javabetter.cn](http://cxyroad.com/”https://javabetter.cn”)
* 后记：尽管我面试表现的很不错，但岳不群公司的 HR 还是没有给我发 offer，理由是我问他们 2025 年真的要从华山搬迁到黑木崖吗?
* 他们 HR 反问我是从哪里听来的消息，有损公司的名誉，说公司并未搬迁或准备搬迁。
* 我心里暗喜：“你就装吧，看你还能装多久！就算是你岳不群公司给我发了 offer，我也是不去的。”

原文链接: https://juejin.cn/post/7381413498003472395