

78、ArrayBlockingQueue (公平、非公平)

用数组实现的有界阻塞队列。此队列按照先进先出 (FIFO) 的原则对元素进行排序。默认情况下不保证访问者公平的访问队列，所谓公平访问队列是指阻塞的所有生产者线程或消费者线程，当队列可用时，可以按照阻塞的先后顺序访问队列，即先阻塞的生产者线程，可以先往队列里插入元素，先阻塞的消费者线程，可以先从队列里获取元素。通常情况下为了保证公平性会降低吞吐量。我们可以使用以下代码创建一个公平的阻塞队列

```
ArrayBlockingQueue fairQueue = new ArrayBlockingQueue(1000, true);
```

79、LinkedBlockingQueue (两个独立锁提高并发)

基于链表的阻塞队列，同 ArrayListBlockingQueue 类似，此队列按照先进先出 (FIFO) 的原则对元素进行排序。而 LinkedBlockingQueue 之所以能够高效的处理并发数据，还因为其对于生产者端和消费者端分别采用了独立的锁来控制数据同步，这也意味着在高并发的情况下生产者和消费者可以并行地操作队列中的数据，以此来提高整个队列的并发性能。LinkedBlockingQueue 会默认一个类似无限大小的容量 (Integer.MAX_VALUE)

80、PriorityBlockingQueue (compareTo 排序实现优先)

是一个支持优先级的无界队列。默认情况下元素采取自然顺序升序排列。可以自定义实现 compareTo() 方法来指定元素进行排序规则，或者初始化 PriorityBlockingQueue 时，指定构造参数 Comparator 来对元素进行排序。需要注意的是不能保证同优先级元素的顺序。

81、DelayQueue (缓存失效、定时任务)

是一个支持延时获取元素的无界阻塞队列。队列使用 PriorityQueue 来实现。队列中的元素必须实现 Delayed 接口，在创建元素时可以指定多久才能从队列中获取当前元素。只有在延迟期满时才能从队列中提取元素。我们可以将 DelayQueue 运用在以下应用场景：

1. 缓存系统的设计：可以用 DelayQueue 保存缓存元素的有效期，使用一个线程循环查询 DelayQueue，一旦能从 DelayQueue 中获取元素时，表示缓存有效期到了。
2. 定时任务调度：使用 DelayQueue 保存当天将会执行的任务和执行时间，一旦从 DelayQueue 中获取到任务就开始执行，从比如 TimerQueue 就是使用 DelayQueue 实现的

82、SynchronousQueue (不存储数据、可用于传递数据)

是一个不存储元素的阻塞队列。每一个 put 操作必须等待一个 take 操作，否则不能继续添加元素。SynchronousQueue 可以看成是一个传球手，负责把生产者线程处理的数据直接传递给消费者线程。队列本身并不存储任何元素，非常适合于传递性场景，比如在一个线程中使用的数据，传递给另外一个线程使用， SynchronousQueue 的吞吐量高于 LinkedBlockingQueue 和 ArrayBlockingQueue。

83、LinkedTransferQueue

是一个由链表结构组成的无界阻塞 TransferQueue 队列。相对于其他阻塞队列， LinkedTransferQueue 多了 tryTransfer 和 transfer 方法。

1. transfer 方法：如果当前有消费者正在等待接收元素（消费者使用 take() 方法或带时间限制的 poll() 方法时）， transfer 方法可以把生产者传入的元素立刻 transfer（传输）给消费者。如果没有消费者在等待接收元素， transfer 方法会将元素存放在队列的 tail 节点，并等到该元素被消费者消费了才返回。
2. tryTransfer 方法。则是用来试探下生产者传入的元素是否能直接传给消费者。如果没有消费者等待接收元素，则返回 false。和 transfer 方法的区别是 tryTransfer 方法无论消费者是否接收，方法立即返回。而 transfer 方法是必须等到消费者消费了才返回。
对于带有时间限制的 tryTransfer(E e, long timeout, TimeUnit unit) 方法，则是试图把生产者传入的元素直接传给消费者，但是如果消费者没有消费该元素则等待指定的时间再返回，如果超时还没有消费元素，则返回 false，如果在超时时间内消费了元素，则返回 true。

84、LinkedBlockingDeque

是一个由链表结构组成的双向阻塞队列。所谓双向队列指的你可以从队列的两端插入和移出元素。双端队列因为多了一个操作队列的入口，在多线程同时入队时，也就减少了一半的竞争。相比其他的阻塞队列， LinkedBlockingDeque 多了 addFirst , addLast , offerFirst , offerLast , peekFirst , peekLast 等方法，以 First 单词结尾的方法，表示插入，获取 (peek) 或移除双端队列的第一个元素。以 Last 单词结尾的方法，表示插入，获取或移除双端队列的最后一个元素。另外插入方法 add 等同于 addLast ，移除方法 remove 等效于 removeFirst 。但是 take 方法却等同于 takeFirst ，不知道是不是 Jdk 的 bug ，使用时还是用带有 First 和 Last 后缀的方法更清楚。在初始化 LinkedBlockingDeque 时可以设置容量防止其过渡膨胀。另外双向阻塞队列可以运用在“工作窃取”模式中。

85、在 java 中守护线程和本地线程区别

java 中的线程分为两种：守护线程（Daemon）和用户线程（User）。

任何线程都可以设置为守护线程和用户线程，通过方法 Thread.setDaemon(boolean)；true 则把该线程设置为守护线程，反之则为用户线程。Thread.setDaemon() 必须在 Thread.start() 之前调用，否则运行时会抛出异常。

两者区别：

唯一的区别是判断虚拟机（JVM）何时离开，Daemon 是为其他线程提供服务，如果全部的 User Thread 已经撤离，Daemon 没有可服务的线程，JVM 撤离。也可以理解为守护线程是 JVM 自动创建的线程（但不一定），用户线程是程序创建的线程；比如 JVM 的垃圾回收线程是一个守护线程，当所有线程已经撤离，不再产生垃圾，守护线程自然就没事可干了，当垃圾回收线程是 Java 虚拟机上仅剩的线程时，Java 虚拟机会自动离开。

扩展：

Thread Dump 打印出来的线程信息，含有 daemon 字样的线程即为守护进程，可能会有：服务守护进程、编译守护进程、Windows 下的监听 Ctrl+break 的守护进程、Finalizer 守护进程、引用处理守护进程、GC 守护进程。

86、线程与进程的区别？

进程是操作系统分配资源的最小单元，线程是操作系统调度的最小单元。
一个程序至少有一个进程，一个进程至少有一个线程。

87、什么是多线程中的上下文切换？

多线程会共同使用一组计算机上的 CPU，而线程数大于给程序分配的 CPU 数量时，为了让各个线程都有执行的机会，就需要轮转使用 CPU。不同的线程切换使用 CPU 发生的切换数据等就是上下文切换。

88、死锁与活锁的区别，死锁与饥饿的区别？

死锁：是指两个或两个以上的进程（或线程）在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。

产生死锁的必要条件：

- 1、互斥条件：所谓互斥就是进程在某一时间内独占资源。
- 2、请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。
- 3、不剥夺条件：进程已获得资源，在未使用完之前，不能强行剥夺。
- 4、循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

活锁：任务或者执行者没有被阻塞，由于某些条件没有满足，导致一直重复尝试，失败，尝试，失败。

活锁和死锁的区别在于，处于活锁的实体是在不断的改变状态，所谓的“活”，而处于死锁的实体表现为等待；活锁有可能自行解开，死锁则不能。

饥饿：一个或者多个线程因为种种原因无法获得所需要的资源，导致一直无法执行的状态。

Java 中导致饥饿的原因：

- 1、高优先级线程吞噬所有的低优先级线程的 CPU 时间。
- 2、线程被永久堵塞在一个等待进入同步块的状态，因为其他线程总是能在它之前持续地对该同步块进行访问。
- 3、线程在等待一个本身也处于永久等待完成的对象（比如调用这个对象的 wait 方法），因为其他线程总是被持续地获得唤醒。

89、Java 中用到的线程调度算法是什么？

采用时间片轮转的方式。可以设置线程的优先级，会映射到下层的系统上面的优先级上，如非特别需要，尽量不要用，防止线程饥饿。

90、什么是线程组，为什么在 Java 中不推荐使用？

ThreadGroup 类，可以把线程归属到某一个线程组中，线程组中可以有线程对象，也可以有线程组，组中还可以有线程，这样的组织结构有点类似于树的形式。

为什么不推荐使用？因为使用有很多的安全隐患吧，没有具体追究，如果需要使用，推荐使用线程池。

91、为什么使用 Executor 框架？

每次执行任务创建线程 new Thread() 比较消耗性能，创建一个线程是比较耗时、耗资源的。

调用 new Thread() 创建的线程缺乏管理，被称为野线程，而且可以无限制的创建，线程之间的相互竞争会导致过多占用系统资源而导致系统瘫痪，还有线程之间的频繁交替也会消耗很多系统资源。

直接使用 new Thread() 启动的线程不利于扩展，比如定时执行、定期执行、定时定期执行、线程中断等都不便实现。

92、在 Java 中 Executor 和 Executors 的区别？

Executors 工具类的不同方法按照我们的需求创建了不同的线程池，来满足业务的需求。
Executor 接口对象能执行我们的线程任务。

ExecutorService 接口继承了 Executor 接口并进行了扩展，提供了更多的方法我们能获得任务执行的状态并且可以获取任务的返回值。
使用 ThreadPoolExecutor 可以创建自定义线程池。

Future 表示异步计算的结果，他提供了检查计算是否完成的方法，以等待计算的完成，并可以使用 get() 方法获取计算的结果。

93、如何在 Windows 和 Linux 上查找哪个线程使用的 CPU 时间最长？

参考：

<http://daiguahub.com/2016/07/31/> 使用 jstack 找出消耗 CPU 最多的线程代码/

94、什么是原子操作？在 Java Concurrency API 中有哪些原子类(atomic classes)？

原子操作 (atomic operation) 意为“不可被中断的一个或一系列操作”。处理器使用基于对缓存加锁或总线加锁的方式来实现多处理器之间的原子操作。在 Java 中可以通过锁和循环 CAS 的方式来实现原子操作。CAS 操作—— Compare & Set，或是 Compare & Swap，现在几乎所有的 CPU 指令都支持 CAS 的原子操作。

原子操作是指一个不受其他操作影响的操作任务单元。原子操作是在多线程环境下避免数据不一致必须的手段。

int++ 并不是一个原子操作，所以当一个线程读取它的值并加 1 时，另外一个线程有可能会读到之前的值，这就会引发错误。

为了解决这个问题，必须保证增加操作是原子的，在 JDK1.5 之前我们可以使用同步技术来做到这一点。到 JDK1.5，java.util.concurrent.atomic 包提供了 int 和 long 类型的原子包装类，它们可以自动的保证对于他们的操作是原子的并且不需要使用同步。

java.util.concurrent 这个包里面提供了一组原子类。其基本的特性就是在多线程环境下，当有多个线程同时执行这些类的实例包含的方法时，具有排他性，即当某个线程进入方法，执行其中的指令时，不会被其他线程打断，而别的线程就像自旋锁一样，一直等到该方法执行完成，才由 JVM 从等待队列中选择一个另一个线程进入，这只是一个逻辑上的理解。

原子类：AtomicBoolean, AtomicInteger, AtomicLong, AtomicReference

原子数组：AtomicIntegerArray, AtomicLongArray, AtomicReferenceArray

原子属性更新器：AtomicLongFieldUpdater, AtomicIntegerFieldUpdater, AtomicReferenceFieldUpdater

解决 ABA 问题的原子类：AtomicMarkableReference（通过引入一个 boolean 来反映中间有没有变过），AtomicStampedReference（通过引入一个 int 来累加来反映中间有没有变过）

95、Java Concurrency API 中的 Lock 接口(Lock interface)是什么？对比同步它有什么优势？

Lock 接口比同步方法和同步块提供了更具扩展性的锁操作。他们允许更灵活的结构，可以具有完全不同的性质，并且可以支持多个相关类的条件对象。

它的优势有：

可以使锁更公平

可以使线程在等待锁的时候响应中断

可以让线程尝试获取锁，并在无法获取锁的时候立即返回或者等待一段时间

可以在不同的范围，以不同的顺序获取和释放锁

整体上来说 Lock 是 synchronized 的扩展版，Lock 提供了无条件的、可轮询的(tryLock 方法)、定时的(tryLock 带参方法)、可中断的(lockInterruptibly)、可多条件队列的(newCondition 方法)锁操作。另外 Lock 的实现类基本都支持非公平锁(默认)和公平锁，synchronized 只支持非公平锁，当然，在大部分情况下，非公平锁是高效的选择。

96、什么是 Executors 框架？

Executor 框架是一个根据一组执行策略调用，调度，执行和控制的异步任务的框架。无限制的创建线程会引起应用程序内存溢出。所以创建一个线程池是个更好的解决方案，因为可以限制线程的数量并且可以回收再利用这些线程。利用 Executors 框架可以非常方便的创建一个线程池。

97、什么是阻塞队列？阻塞队列的实现原理是什么？如何使用阻塞队列来实现生产者-消费者模型？

阻塞队列（BlockingQueue）是一个支持两个附加操作的队列。这两个附加的操作是：在队列为空时，获取元素的线程会等待队列变为非空。当队列满时，存储元素的线程会等待队列可用。阻塞队列常用于生产者和消费者的场景，生产者是往队列里添加元素的线程，消费者是从队列里拿元素的线程。阻塞队列就是生产者存放元素的容器，而消费者也只从容器里拿元素。

JDK7 提供了 7 个阻塞队列。分别是：

ArrayBlockingQueue：一个由数组结构组成的有界阻塞队列。
LinkedBlockingQueue：一个由链表结构组成的有界阻塞队列。
PriorityBlockingQueue：一个支持优先级排序的无界阻塞队列。
DelayQueue：一个使用优先级队列实现的无界阻塞队列。
SynchronousQueue：一个不存储元素的阻塞队列。
LinkedTransferQueue：一个由链表结构组成的无界阻塞队列。
LinkedBlockingDeque：一个由链表结构组成的双向阻塞队列。

Java 5 之前实现同步存取时，可以使用普通的一个集合，然后在使用线程的协作和线程同步可以实现生产者，消费者模式，主要的技术就是用好，wait, notify, notifyAll, synchronized 这些关键字。而在 java 5 之后，可以使用阻塞队列来实现，此方式大大简少了代码量，使得多线程编程更加容易，安全方面也有保障。

BlockingQueue 接口是 Queue 的子接口，它的主要用途并不是作为容器，而是作为线程同步的工具，因此他具有一个很明显的特性，当生产者线程试图向 BlockingQueue 放入元素时，如果队列已满，则线程被阻塞，当消费者线程试图从中取出一个元素时，如果队列为空，则该线程会被阻塞，正是因为它所具有这个特性，所以在程序中多个线程交替向 BlockingQueue 中放入元素，取出元素，它可以很好的控制线程之间的通信。

阻塞队列使用最经典的场景就是 socket 客户端数据的读取和解析，读取数据的线程不断将数据放入队列，然后解析线程不断从队列取数据解析。

98、什么是 Callable 和 Future?

Callable 接口类似于 Runnable，从名字就可以看出来了，但是 Runnable 不会返回结果，并且无法抛出返回结果的异常，而 Callable 功能更强大一些，被线程执行后，可以返回值，这个返回值可以被 Future 拿到，也就是说，Future 可以拿到异步执行任务的返回值。可以认为是带有回调的 Runnable。

Future 接口表示异步任务，是还没有完成的任务给出的未来结果。所以说 Callable 用于产生结果，Future 用于获取结果。

99、什么是 FutureTask? 使用 ExecutorService 启动任务。

在 Java 并发程序中 FutureTask 表示一个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等方法。只有当运算完成的时候结果才能取回，如果运算尚未完成 get 方法将会阻塞。一个 FutureTask 对象可以对调用了 Callable 和 Runnable 的对象进行包装，由于 FutureTask 也是调用了 Runnable 接口所以它可以提交给 Executor 来执行。

100、什么是并发容器的实现？

何为同步容器：可以简单地理解为通过 synchronized 来实现同步的容器，如果有多个线程调用同步容器的方法，它们将会串行执行。比如 Vector, Hashtable, 以及 Collections.synchronizedSet, synchronizedList 等方法返回的容器。可以通过查看 Vector, Hashtable 等这些同步容器的实现代码，可以看到这些容器实现线程安全的方式就是将它们的状态封装起来，并在需要同步的方法上加上关键字 synchronized。

并发容器使用了与同步容器完全不同的加锁策略来提供更高的并发性和伸缩性，例如在 ConcurrentHashMap 中采用了一种粒度更细的加锁机制，可以称为分段锁，在这种锁机制下，允许任意数量的读线程并发地访问 map，并且执行读操作的线程和写操作的线程也可以并发的访问 map，同时允许一定数量的写操作线程并发地修改 map，所以它可以在并发环境下实现更高的吞吐量。

101、多线程同步和互斥有几种实现方法，都是什么？

线程同步是指线程之间所具有的一种制约关系，一个线程的执行依赖另一个线程的消息，当它没有得到另一个线程的消息时应等待，直到消息到达时才被唤醒。线程互斥是指对于共享的进程系统资源，在单个线程访问时的排它性。当有若干个线程都要使用某一共享资源时，任何时刻最多只允许一个线程去使用，其它要使用该资源的线程必须等待，直到占用资源者释放该资源。线程互斥可以看成是一种特殊的线程同步。

线程间的同步方法大体可分为两类：用户模式和内核模式。顾名思义，内核模式就是指利用系统内核对象的单一性来进行同步，使用时需要切换内核态与用户态，而用户模式就是不需要切换到内核态，只在用户态完成操作。用户模式下的方法有：原子操作（例如一个单一的全局变量），临界区。内核模式下的方法有：事件，信号量，互斥量。

102、什么是竞争条件？你怎样发现和解决竞争？

当多个进程都企图对共享数据进行某种处理，而最后的结果又取决于进程运行的顺序时，则我们认为这发生了竞争条件（race condition）。

103、为什么我们调用 start()方法时会执行 run()方法，为什么我们不能直接调用 run()方法？

当你调用 start()方法时你将创建新的线程，并且执行在 run()方法里的代码。但是如果你直接调用 run()方法，它不会创建新的线程也不会执行调用线程的代码，只会把 run 方法当作普通方法去执行。

104、Java 中你怎样唤醒一个阻塞的线程？

在 Java 发展史上曾经使用 suspend()、resume()方法对于线程进行阻塞唤醒，但随之出现很多问题，比较典型的还是死锁问题。

解决方案可以使用以对象为目标的阻塞，即利用 Object 类的 wait()和 notify()方法实现线程阻塞。

首先，wait、notify 方法是针对对象的，调用任意对象的 wait()方法都将导致线程阻塞，阻塞的同时也将释放该对象的锁，相应地，调用任意对象的 notify()方法则将随机解除该对象阻塞的线程，但它需要重新获取改对象的锁，直到获取成功才能往下执行；其次，wait、notify 方法必须在 synchronized 块或方法中被调用，并且要保证同步块或方法的锁对象与调用 wait、notify 方法的对象是同一个，如此一来在调用 wait 之前当前线程就已经成功获取某对象的锁，执行 wait 阻塞后当前线程就将之前获取的对象锁释放。

105、在 Java 中 CyclicBarrier 和 CountdownLatch 有什么区别？

CyclicBarrier 可以重复使用，而 CountdownLatch 不能重复使用。Java 的 concurrent 包里面的 CountDownLatch 其实可以把它看作一个计数器，只不过这个计数器的操作是原子操作，同时只能有一个线程去操作这个计数器，也就是同时只能有一个线程去减这个计数器里面的值。你可以向 CountDownLatch 对象设置一个初始的数字作为计数值，任何调用这个对象上的 await()方法都会阻塞，直到这个计数器的计数值被其他的线程减为 0 为止。

所以在当前计数到达零之前，await 方法会一直受阻塞。之后，会释放所有等待的线程，await 的所有后续调用都将立即返回。这种现象只出现一次——计数无法被重置。如果需要重置计数，请考虑使用 CyclicBarrier。

CountDownLatch 的一个非常典型的应用场景是：有一个任务想要往下执行，但必须要等到其他的任务执行完毕后才可以继续往下执行。假如我们这个想要继续往下执行的任务调用一个 CountDownLatch 对象的 await()方法，其他的任务执行完自己的任务后调用同一个 CountDownLatch 对象上的 countDown()方法，这个调用 await()方法的任务将一直阻塞等待，直到这个 CountDownLatch 对象的计数值减到 0 为止。

CyclicBarrier 一个同步辅助类，它允许一组线程互相等待，直到到达某个公共屏障点 (common barrier point)。在涉及一组固定大小的线程的程序中，这些线程必须不时地互相等待，此时 CyclicBarrier 很有用。因为该 barrier 在释放等待线程后可以重用，所以称它为循环的 barrier。

106、什么是不可变对象，它对写并发应用有什么帮助

不可变对象(Immutable Objects)即对象一旦被创建它的状态（对象的数据，也即对象属性值）就不能改变，反之即为可变对象(Mutable Objects)。不可变对象的类即为不可变类(Immutable Class)。Java 平台类库中包含许多不可变类，如 String、基本类型的包装类、 BigInteger 和 BigDecimal 等。不可变对象天生是线程安全的。它们的常量（域）是在构造函数中创建的。既然它们的状态无法修改，这些常量永远不会变。

不可变对象永远是线程安全的。
只有满足如下状态，一个对象才是不可变的：
它的状态不能在创建后再被修改；
所有域都是 final 类型；并且，
它被正确创建（创建期间没有发生 this 引用的逸出）。

107、Java 中用到的线程调度算法是什么？

计算机通常只有一个 CPU，在任意时刻只能执行一条机器指令，每个线程只有获得 CPU 的使用权才能执行指令。所谓多线程的并发运行，其实是指从宏观上看，各个线程轮流获得 CPU 的使用权，分别执行各自的任务。在运行池中，会有多个处于就绪状态的线程在等待 CPU。JAVA 虚拟机的一项任务就是负责线程的调度，线程调度是指按照特定机制为多个线程分配 CPU 的使用权。

有两种调度模型：分时调度模型和抢占式调度模型。分时调度模型是指让所有的线程轮流获得 cpu 的使用权，并且平均分配每个线程占用的 CPU 的时间片，这个也比较好理解。java 虚拟机采用抢占式调度模型，是指优先让可运行池中优先级高的线程占用 CPU，如果可运行池中的线程优先级相同，那么就随机选择一个线程，使其占用 CPU。处于运行状态的线程会一直运行，直至它不得不放弃 CPU。

108、什么是线程组，为什么在 Java 中不推荐使用？

线程组和线程池是两个不同的概念，他们的作用完全不同，前者是为了方便线程的管理，后者是为了管理线程的生命周期，复用线程，减少创建销毁线程的开销。

JVM面试题

1、java中会存在内存泄漏吗， 请简单描述。

会。自己实现堆载的数据结构时有可能会出现内存泄露，可参看effective java.

2、64位JVM中，int的长度是多數？

Java中，int类型变量的长度是一个固定值，与平台无关，都是32位。意思就是说，在32位和64位的Java虚拟机中，int类型的长度是相同的。

3、Serial与Parallel GC之间的不同之处？

Serial与Parallel在GC执行的时候都会引起stop-the-world。它们之间主要不同serial收集器是默认的复制收集器，执行GC的时候只有一个线程，而parallel收集器使用多个GC线程来执行。

4、32位和64位的JVM，int类型变量的长度是多數？

32位和64位的JVM中，int类型变量的长度是相同的，都是32位或者4个字节。

5、Java中WeakReference与SoftReference的区别？

虽然WeakReference与SoftReference都有利于提高GC和内存的效率，但是WeakReference，一旦失去最后一个强引用，就会被GC回收，而软引用虽然不能阻止被回收，但是可以延迟到JVM内存不足的时候。

6、JVM选项-XX:+UseCompressedOops有什么作用？为什么要使用

当你将你的应用从32位的JVM迁移到64位的JVM时，由于对象的指针从32位增加到了64位，因此堆内存会突然增加，差不多要翻倍。这也会对CPU缓存（容量比内存小很多）的数据产生不利的影响。因为，迁移到64位的JVM主要动机在于可以指定最大堆大小，通过压缩OOP可以节省一定的内存。通过-XX:+UseCompressedOops选项，JVM会使用32位的OOP，而不是64位的OOP。

7、怎样通过Java程序来判断JVM是32位还是64位？

你可以检查某些系统属性如sun.arch.data.model或os.arch来获取该信息。

8、32位JVM和64位JVM的最大堆内存分别是多數？

理论上说上32位的JVM堆内存可以到达 2^{32} ，即4GB，但实际上会比这个小很多。不同操作系统之间不同，如Windows系统大约1.5GB，Solaris大约3GB。64位JVM允许指定最大的堆内存，理论上可以达到 2^{64} ，这是一个非常大的数字，实际上你可以指定堆内存大小到100GB。甚至有的JVM，如Azul，堆内存到1000G都是可能的。

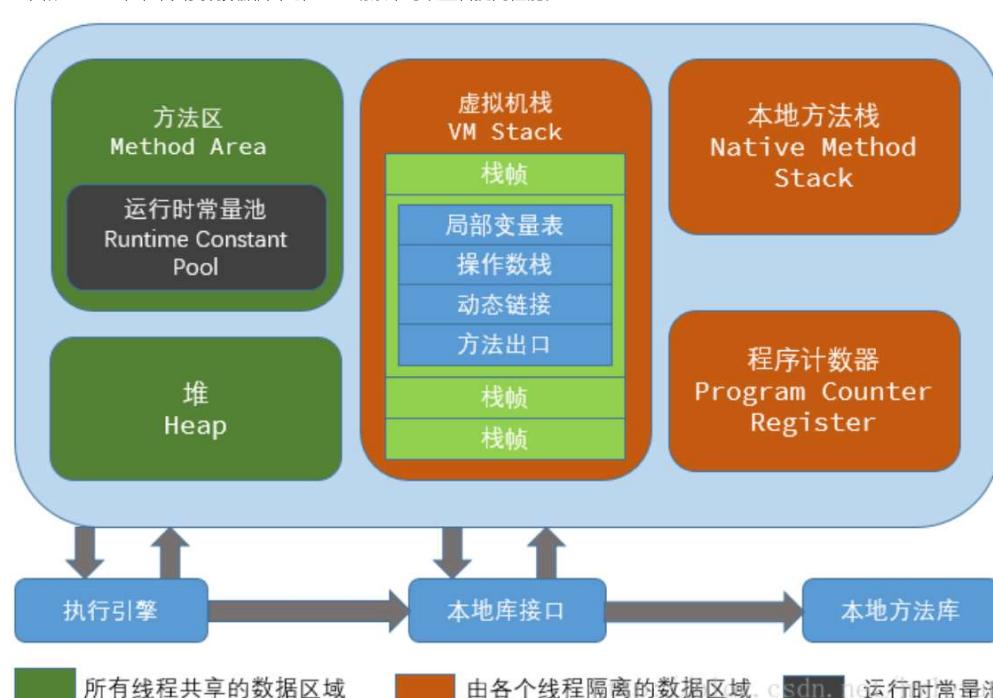
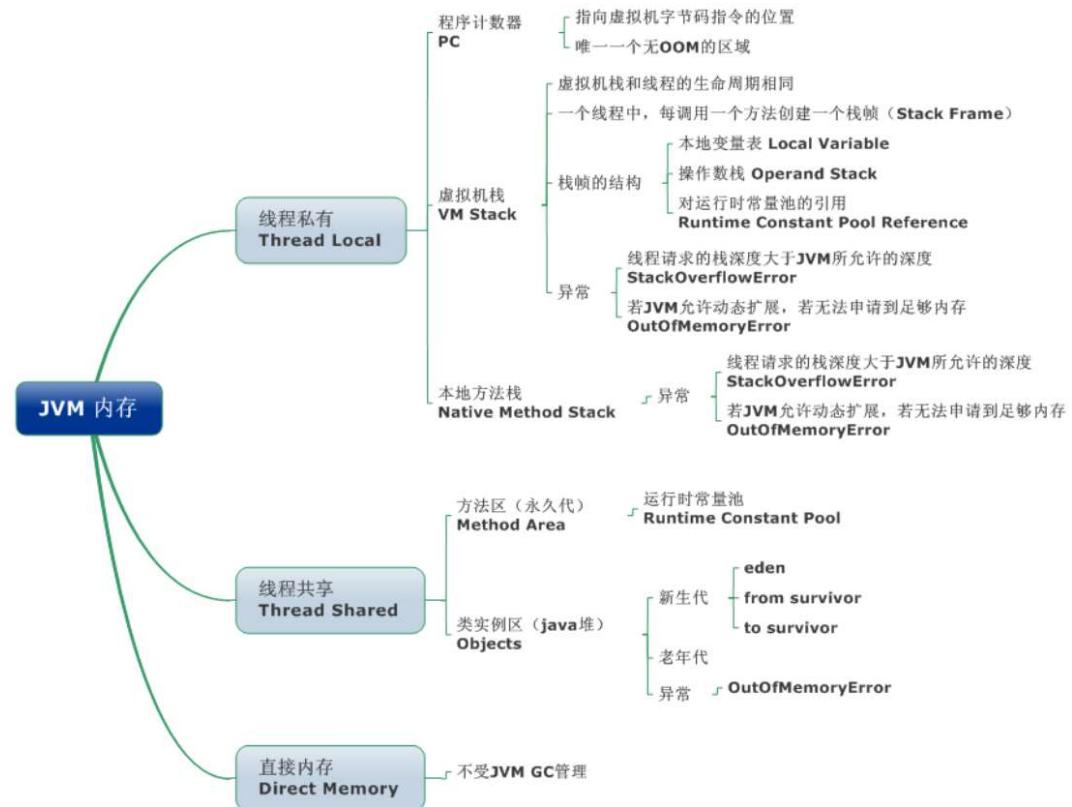
9、JRE、JDK、JVM及JIT之间有什么不同？

JRE代表Java运行时（Java run-time），是运行Java应用所必须的。JDK代表Java开发工具（Java development kit），是Java程序的开发工具，如Java编译器，它也包含JRE。JVM代表Java虚拟机（Java virtual machine），它的责任是运行Java应用。JIT代表即时编译（Just In Time compilation），当代码执行的次数超过一定的阈值时，会将Java字节码转换为本地代码，如，主要的热点代码会被编译为本地代码，这样有利大幅度提高Java应用的性能。

10、解释 Java 堆空间及 GC ?

当通过 Java 命令启动 Java 进程的时候，会为它分配内存。内存的一部分用于创建堆空间，当程序中创建对象的时候，就从对空间中分配内存。GC 是 JVM 内部的一个进程，回收无效对象的内存用于将来的分配。

11、JVM 内存区域



12、程序计数器(线程私有)

一块较小的内存空间，是当前线程所执行的字节码的行号指示器，每条线程都要有一个独立的程序计数器，这类内存也称为“线程私有”的内存。

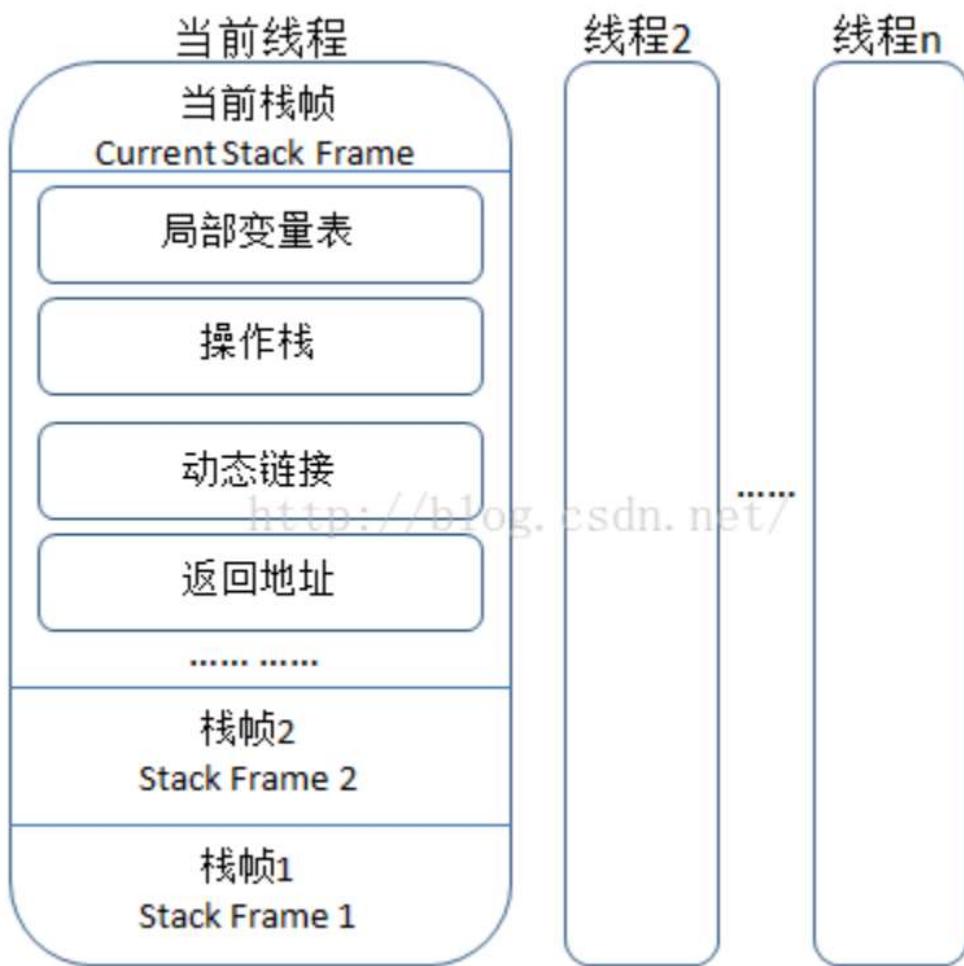
正在执行 java 方法的话，计数器记录的是虚拟机字节码指令的地址（当前指令的地址）。如果还是 Native 方法，则为空。

这个内存区域是唯一一个在虚拟机中没有规定任何 OutOfMemoryError 情况的区域。

13、虚拟机栈(线程私有)

是描述 Java 方法执行的内存模型，每个方法在执行的同时都会创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每一个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。

栈帧（Frame）是用来存储数据和部分过程结果的数据结构，同时也被用来处理动态链接（Dynamic Linking）、方法返回值和异常分派（Dispatch Exception）。栈帧随着方法调用而创建，随着方法结束而销毁——无论方法是正常完成还是异常完成（抛出了在方法内未被捕获的异常）都算作方法结束。



14、本地方法区(线程私有)

本地方法区和 Java Stack 作用类似，区别是虚拟机栈为执行 Java 方法服务，而本地方法栈则为 Native 方法服务，如果一个 VM 实现使用 C-linkage 模型来支持 Native 调用，那么该栈将会是一个 C 栈，但 HotSpot VM 直接就把本地方法栈和虚拟机栈合二为一。

15、你能保证 GC 执行吗？

不能，虽然你可以调用 System.gc() 或者 Runtime.gc()，但是没有办法保证 GC 的执行。

16、怎么获取 Java 程序使用的内存？堆使用的百分比？

可以通过 java.lang.Runtime 类中与内存相关方法来获取剩余的内存，总内存及最大堆内存。通过这些方法你也可以获取到堆使用的百分比及堆内存的剩余空间。Runtime.freeMemory() 方法返回剩余空间的字节数，Runtime.totalMemory() 方法总内存的字节数，Runtime.maxMemory() 返回最大内存的字节数。

17、Java 中堆和栈有什么区别？

JVM 中堆和栈属于不同的内存区域，使用目的也不同。栈常用于保存方法帧和局部变量，而对象总是在堆上分配。栈通常都比堆小，也不会在多个线程之间共享，而堆被整个 JVM 的所有线程共享。

18、描述一下 JVM 加载 class 文件的原理机制

JVM 中类的装载是由类加载器 (ClassLoader) 和它的子类来实现的，Java 中的类加载器是一个重要的 Java 运行时系统组件，它负责在运行时查找和装入类文件中的类。

由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。

加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后 JVM 对类进行初始化，包括：1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2)如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：根加载器 (BootStrap) 、扩展加载器 (Extension) 、系统加载器 (System) 和用户自定义类加载器 (java.lang.ClassLoader 的子类) 。

从 Java 2 (JDK 1.2) 开始，类加载过程采取了父亲委托机制 (PDM) 。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

1. Bootstrap : 一般用本地代码实现，负责加载 JVM 基础核心类库 (rt.jar) ；
2. Extension : 从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap ；
3. System : 又叫应用类加载器，其父类是 Extension 。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

19、GC 是什么？为什么要有 GC ？

GC 是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或 Runtime.getRuntime().gc() ，但 JVM 可以屏蔽掉显示的垃圾回收调用。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是一个单独的低优先级的线程运行，不可预知的情况下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，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在 Java 诞生初期，垃圾回收是 Java 最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今 Java 的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得 iOS 的系统比 Android 系统有更好的用户体验，其中一个深层次的原因就在于 Android 系统中垃圾回收的不可预知性。

20、堆 (Heap-线程共享) -运行时数据区

是线程共享的一块内存区域，创建的对象和数组都保存在 Java 堆内存中，也是垃圾收集器进行垃圾收集的最重要的内存区域。由于现代 VM 采用分代收集算法，因此 Java 堆从 GC 的角度还可以细分为：新生代 (Eden 区、 From Survivor 区和 To Survivor 区) 和老年代。

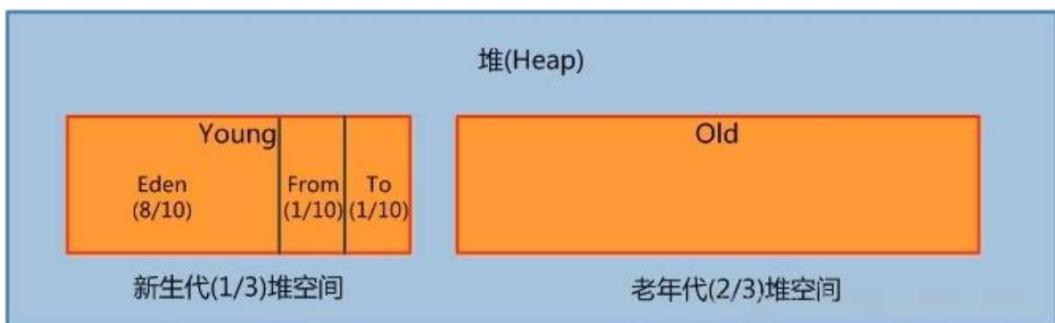
21、方法区/永久代 (线程共享)

即我们常说的永久代 (Permanent Generation) ，用于存储被 JVM 加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。HotSpot VM 把 GC 分代收集扩展至方法区，即使用 Java 堆的永久代来实现方法区，这样 HotSpot 的垃圾收集器就可以像管理 Java 堆一样管理这部分内存，而不必为方法区开发专门的内存管理器。永久代的内存回收的主要目标是针对常量池的回收和类型的卸载，因此收益一般很小。

运行时常量池 (Runtime Constant Pool) 是方法区的一部分。 Class 文件中除了有类的版本、字段、方法、接口等描述等信息外，还有一项信息是常量池 (Constant Pool Table) ，用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后存放到方法区的运行时常量池中。 Java 虚拟机对 Class 文件的每一部分 (自然也包括常量池) 的格式都有严格的规定，每一个字节用于存储哪种数据都必须符合规范上的要求，这样才会被虚拟机认可、装载和执行。

22、JVM 运行时内存

Java 堆从 GC 的角度还可以细分为：新生代 (Eden 区、 From Survivor 区和 To Survivor 区) 和老年代。



23、新生代

是用来存放新生的对象。一般占据堆的 1/3 空间。由于频繁创建对象，所以新生代会频繁触发 MinorGC 进行垃圾回收。新生代又分为 Eden 区、SurvivorFrom、SurvivorTo 三个区。

Eden 区

Java 新对象的出生地（如果新创建的对象占用内存很大，则直接分配到老年区）。当 Eden 区内存不够的时候就会触发 MinorGC，对新生代区进行一次垃圾回收。

SurvivorFrom

上一次 GC 的幸存者，作为这一次 GC 的被扫描者。

SurvivorTo

保留了一次 MinorGC 过程中的幸存者。

MinorGC 的过程 (复制->清空->互换)

MinorGC 采用复制算法。

1 : eden、survivorFrom 复制到 survivorTo，年龄+1

首先，把 Eden 和 SurvivorFrom 区域中存活的对象复制到 SurvivorTo 区域（如果有对象的年龄以及达到了老年的标准，则赋值到老年区），同时把这些对象的年龄+1（如果 SurvivorTo 不够位置了就放到老年区）；

2 : 清空 eden、survivorFrom

然后，清空 Eden 和 SurvivorFrom 中的对象；

3 : SurvivorTo 和 SurvivorFrom 互换

最后，SurvivorTo 和 SurvivorFrom 互换，原 SurvivorTo 成为下一次 GC 时的 SurvivorFrom 区。

24、老年代

主要存放应用程序中生命周期长的内存对象。

老年代的对象比较稳定，所以 MajorGC 不会频繁执行。在进行 MajorGC 前一般都先进行了一次 MinorGC，使得有新生代的对象晋身入老年代，导致空间不够用时才触发。当无法找到足够大的连续空间分配给新创建的较大对象时也会提前触发一次 MajorGC 进行垃圾回收腾出空间。

MajorGC 采用标记清除算法：首先扫描一次所有老年代，标记出存活的对象，然后回收没有标记的对象。 MajorGC 的耗时比较长，因为要扫描再回收。 MajorGC 会产生内存碎片，为了减少内存损耗，我们一般需要进行合并或者标记出来方便下次直接分配。当老年代也满了装不下的时候，就会抛出 OOM (Out of Memory) 异常。

25、永久代

指内存的永久保存区域，主要存放 Class 和 Meta (元数据) 的信息。Class 在被加载的时候被放入永久区域，它和存放实例的区域不同。GC 不会在主程序运行期对永久区域进行清理。所以这也导致了永久代的区域会随着加载的 Class 的增多而胀满，最终抛出 OOM 异常。

26、JAVA8 与元数据

在 Java8 中，永久代已经被移除，被一个称为“元数据区”（元空间）的区域所取代。元空间的本质和永久代类似，元空间与永久代之间最大的区别在于：元空间并不在虚拟机中，而是使用本地内存。因此，默认情况下，元空间的大小仅受本地内存限制。类的元数据放入 `nativememory`，字符串池和类的静态变量放入 `java 堆中`，这样可以加载多少类的元数据就不再由 `MaxPermSize` 控制，而由系统的实际可用空间来控制。

27、引用计数法

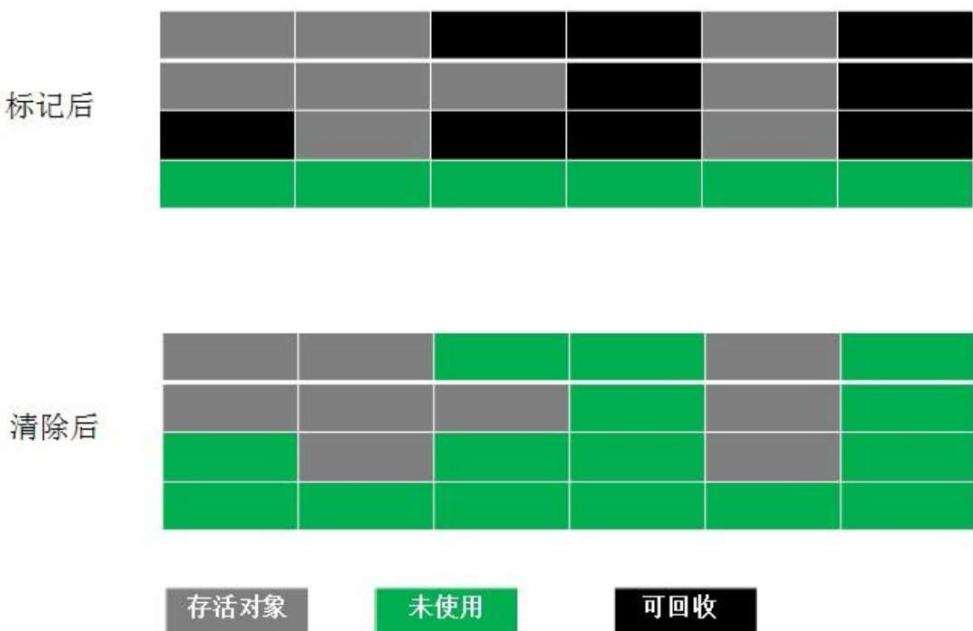
在 Java 中，引用和对象是有关联的。如果要操作对象则必须用引用进行。因此，很显然一个简单的办法是通过引用计数来判断一个对象是否可以回收。简单说，即一个对象如果没有任何与之关联的引用，即他们的引用计数都不为 0，则说明对象不太可能再被用到，那么这个对象就是可回收对象。

28、可达性分析

为了解决引用计数法的循环引用问题，Java 使用了可达性分析的方法。通过一系列的“GC roots”对象作为起点搜索。如果在“GC roots”和一个对象之间没有可达路径，则称该对象是不可达的。要注意的是，不可达对象不等价于可回收对象，不可达对象变为可回收对象至少要经过两次标记过程。两次标记后仍然是可回收对象，则将面临回收。

29、标记清除算法 (Mark-Sweep)

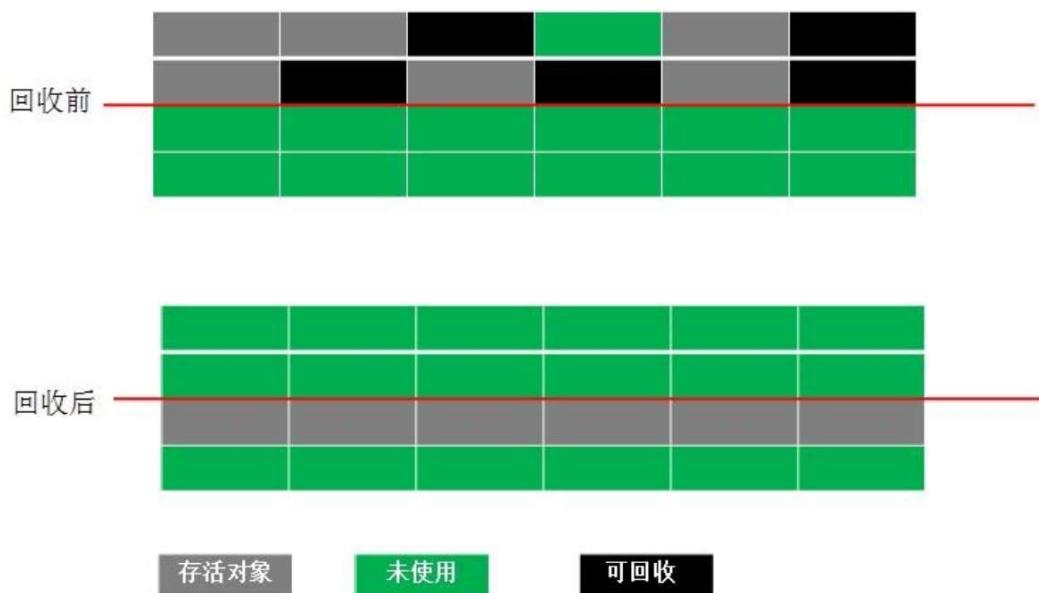
最基础的垃圾回收算法，分为两个阶段，标注和清除。标记阶段标记出所有需要回收的对象，清除阶段回收被标记的对象所占用的空间。如图



从图中我们就可以发现，该算法最大的问题是内存碎片化严重，后续可能发生大对象不能找到可利用空间的问题。

30、复制算法 (copying)

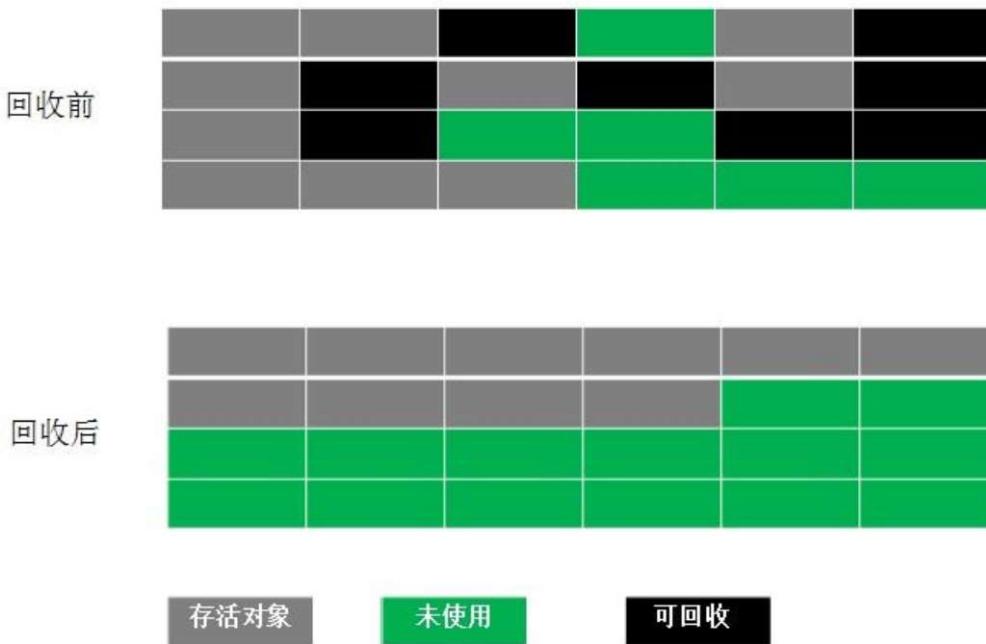
为了解决 Mark-Sweep 算法内存碎片化的缺陷而被提出的算法。按内存容量将内存划分为等大小的两块。每次只使用其中一块，当这一块内存满后将尚存活的对象复制到另一块上去，把已使用的内存清掉，如图：



这种算法虽然实现简单，内存效率高，不易产生碎片，但是最大的问题是可用内存被压缩到了原本的一半。且存活对象增多的话，Copying 算法的效率会大大降低。

31、标记整理算法(Mark-Compact)

结合了以上两个算法，为了避免缺陷而提出。标记阶段和 Mark-Sweep 算法相同，标记后不是清理对象，而是将存活对象移向内存的一端。然后清除端边界外的对象。如图：

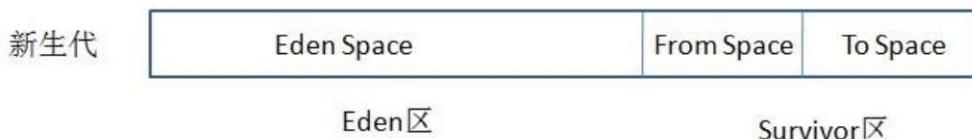


32、分代收集算法

分代收集法是目前大部分 JVM 所采用的方法，其核心思想是根据对象存活的不同生命周期将内存划分为不同的域，一般情况下将 GC 堆划分为老生代(Tenured/Old Generation)和新生代(YoungGeneration)。老生代的特点是每次垃圾回收时只有少量对象需要被回收，新生代的特点是每次垃圾回收时都有大量垃圾需要被回收，因此可以根据不同区域选择不同的算法。

33、新生代与复制算法

目前大部分 JVM 的 GC 对于新生代都采取 Copying 算法，因为新生代中每次垃圾回收都要回收大部分对象，即要复制的操作比较多，但通常并不是按照 1 : 1 来划分新生代。一般将新生代划分为一块较大的 Eden 空间和两个较小的 Survivor 空间(From Space, To Space)，每次使用Eden 空间和其中的一块 Survivor 空间，当进行回收时，将该两块空间中还存活的对象复制到另一块 Survivor 空间中。



34、老年代与标记复制算法

而老年代因为每次只回收少量对象，因而采用 Mark-Compact 算法。

1. JAVA 虚拟机提到过的处于方法区的永生代(Permanet Generation)，它用来存储 class 类，常量，方法描述等。对永生代的回收主要包括废弃常量和无用的类。
2. 对象的内存分配主要在新生代的 Eden Space 和 Survivor Space 的 From Space(Survivor 目前存放对象的那一块)，少数情况会直接分配到老生代。
3. 当新生代的 Eden Space 和 From Space 空间不足时就会发生一次 GC，进行 GC 后，EdenSpace 和 From Space 区的存活对象会被挪到 To Space，然后将 Eden Space 和 FromSpace 进行清理。
4. 如果 To Space 无法足够存储某个对象，则将这个对象存储到老生代。
5. 在进行 GC 后，使用的便是 Eden Space 和 To Space 了，如此反复循环。
6. 当对象在 Survivor 区躲过一次 GC 后，其年龄就会+1。默认情况下年龄到达 15 的对象会被移到老生代中。

35、JAVA 强引用

在 Java 中最常见的就是强引用，把一个对象赋给一个引用变量，这个引用变量就是一个强引用。当一个对象被强引用变量引用时，它处于可达状态，它是不可能被垃圾回收机制回收的，即使该对象以后永远都不会被用到 JVM 也不会回收。因此强引用是造成 Java 内存泄漏的主要原因之一。

36、JAVA软引用

软引用需要用 SoftReference 类来实现，对于只有软引用的对象来说，当系统内存足够时它不会被回收，当系统内存空间不足时它会被回收。软引用通常用在对内存敏感的程序中。

37、JAVA弱引用

弱引用需要用 WeakReference 类来实现，它比软引用的生存期更短，对于只有弱引用的对象来说，只要垃圾回收机制一运行，不管 JVM 的内存空间是否足够，总会回收该对象占用的内存。

38、JAVA虚引用

虚引用需要 PhantomReference 类来实现，它不能单独使用，必须和引用队列联合使用。虚引用的主要作用是跟踪对象被垃圾回收的状态。

39、分代收集算法

当前主流 VM 垃圾收集都采用“分代收集”(Generational Collection)算法，这种算法会根据对象存活周期的不同将内存划分为几块，如 JVM 中的新生代、老年代、永久代，这样就可以根据各年代特点分别采用最适当的 GC 算法。

40、在新生代-复制算法

每次垃圾收集都能发现大批对象已死，只有少量存活，因此选用复制算法，只需要付出少量存活对象的复制成本就可以完成收集。

41、在老年代-标记整理算法

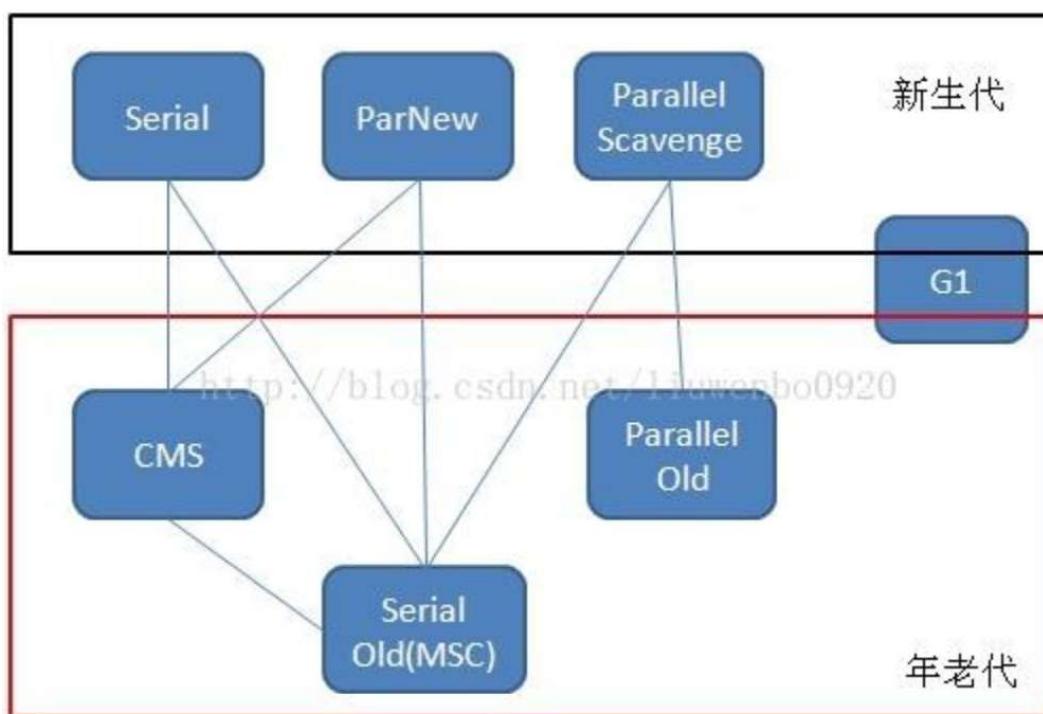
因为对象存活率高、没有额外空间对它进行分配担保，就必须采用“标记—清理”或“标记—整理”算法来进行回收，不必进行内存复制，且直接腾出空闲内存。

42、分区收集算法

分区算法则将整个堆空间划分为连续的不同小区间，每个小区间独立使用，独立回收，这样做的好处是可以控制一次回收多少个小区间，根据目标停顿时间，每次合理地回收若干个小区间(而不是整个堆)，从而减少一次 GC 所产生的停顿。

43、GC 垃圾收集器

Java 堆内存被划分为新生代和年老代两部分，新生代主要使用复制和标记-清除垃圾回收算法；年老代主要使用标记-整理垃圾回收算法，因此 Java 虚拟机针对新生代和年老代分别提供了多种不同的垃圾收集器，JDK1.6 中 Sun HotSpot 虚拟机的垃圾收集器如下：



44、Serial 垃圾收集器 (单线程、复制算法)

Serial (英文连续) 是最基本垃圾收集器，使用复制算法，曾经是 JDK1.3.1 之前新生代唯一的垃圾收集器。 Serial 是一个单线程的收集器，它不但只会使用一个 CPU 或一条线程去完成垃圾收集工作，并且在进行垃圾收集的同时，必须暂停其他所有的工作线程，直到垃圾收集结束。

Serial 垃圾收集器虽然在收集垃圾过程中需要暂停所有其他的工作线程，但是它简单高效，对于限定单个 CPU 环境来说，没有线程交互的开销，可以获得最高的单线程垃圾收集效率，因此 Serial 垃圾收集器依然是 Java 虚拟机运行在 Client 模式下默认的新生代垃圾收集器。

45、ParNew 垃圾收集器 (Serial+多线程)

ParNew 垃圾收集器其实是 Serial 收集器的多线程版本，也使用复制算法，除了使用多线程进行垃圾收集之外，其余的行为和 Serial 收集器完全一样，ParNew 垃圾收集器在垃圾收集过程中同样也要暂停所有其他的工作线程。

ParNew 收集器默认开启和 CPU 数目相同的线程数，可以通过 -XX:ParallelGCThreads 参数来限制垃圾收集器的线程数。【Parallel：平行的】

ParNew 虽然是除了多线程外和Serial 收集器几乎完全一样，但是ParNew垃圾收集器是很多 java 虚拟机运行在 Server 模式下新生代的默认垃圾收集器。

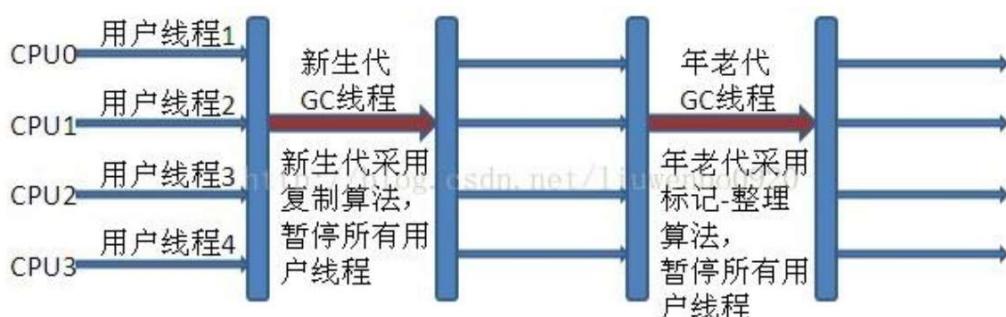
46、Parallel Scavenge 收集器（多线程复制算法、高效）

Parallel Scavenge 收集器也是一个新生代垃圾收集器，同样使用复制算法，也是一个多线程的垃圾收集器，它重点关注的是程序达到一个可控制的吞吐量 (Throughput)，CPU 用于运行用户代码的时间/CPU 总消耗时间，即吞吐量=运行用户代码时间/(运行用户代码时间+垃圾收集时间))，高吞吐量可以最高效率地利用 CPU 时间，尽快地完成程序的运算任务，主要适用于在后台运算而不需要太多交互的任务。自适应调节策略也是 ParallelScavenge 收集器与 ParNew 收集器的一个重要区别。

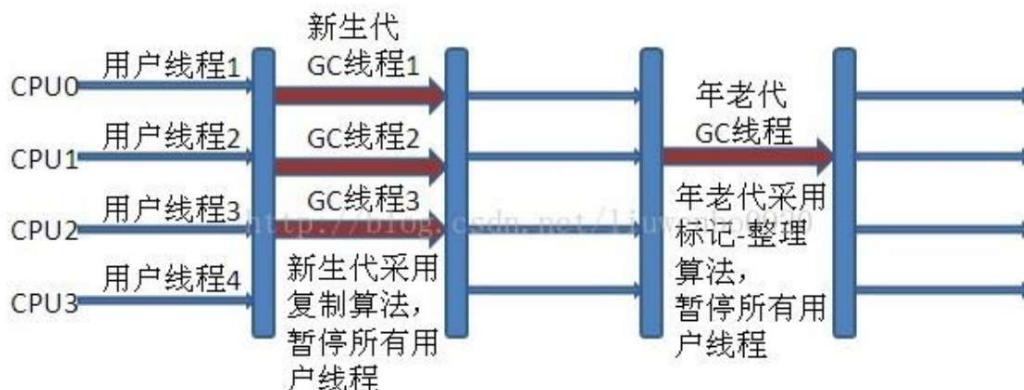
57、Serial Old 收集器（单线程标记整理算法）

Serial Old 是 Serial 垃圾收集器年老代版本，它同样是个单线程的收集器，使用标记-整理算法，这个收集器也主要是运行在 Client 默认的 java 虚拟机默认的年老代垃圾收集器。在 Server 模式下，主要有两个用途：

1. 在 JDK1.5 之前版本中与新生代的 Parallel Scavenge 收集器搭配使用。
2. 作为年老代中使用 CMS 收集器的后备垃圾收集方案。新生代 Serial 与年老代 Serial Old 搭配垃圾收集过程图：



新生代 Parallel Scavenge 收集器与 ParNew 收集器工作原理类似，都是多线程的收集器，都使用的是复制算法，在垃圾收集过程中都需要暂停所有的工作线程。新生代 ParallelScavenge/ParNew 与年老代 Serial Old 搭配垃圾收集过程图：

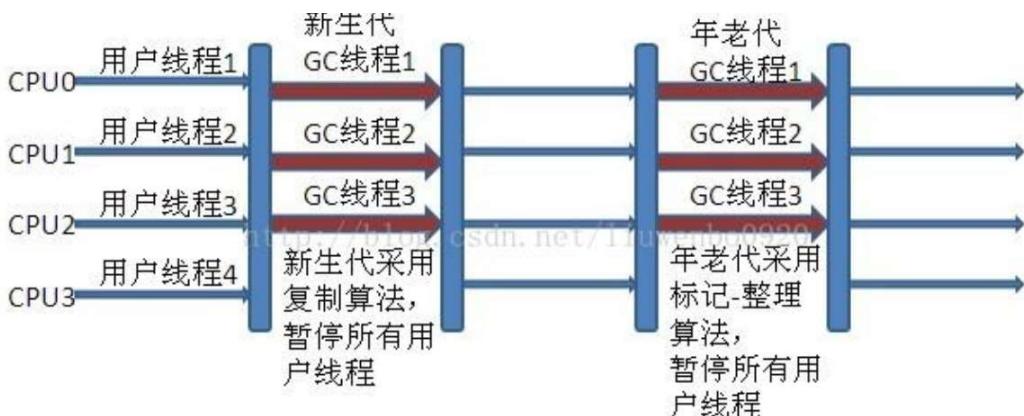


58、Parallel Old 收集器（多线程标记整理算法）

Parallel Old 收集器是 Parallel Scavenge 的年老代版本，使用多线程的标记-整理算法，在 JDK1.6 才开始提供。

在 JDK1.6 之前，新生代使用 ParallelScavenge 收集器只能搭配年老代的 Serial Old 收集器，只能保证新生代的吞吐量优先，无法保证整体的吞吐量，Parallel Old 正是为了在年老代同样提供吞吐量优先的垃圾收集器，如果系统对吞吐量要求比较高，可以优先考虑新生代 Parallel Scavenge 和年老代 Parallel Old 收集器的搭配策略。

新生代 Parallel Scavenge 和年老代 Parallel Old 收集器搭配运行过程图



59、CMS 收集器 (多线程标记清除算法)

Concurrent mark sweep(CMS)收集器是一种年老代垃圾收集器，其最主要目标是获取最短垃圾回收停顿时间，和其他年老代使用标记-整理算法不同，它使用多线程的标记-清除算法。最短的垃圾收集停顿时间可以为交互比较高的程序提高用户体验。CMS 工作机制相比其他的垃圾收集器来说更复杂。整个过程分为以下 4 个阶段：

初始标记

只是标记一下 GC Roots 能直接关联的对象，速度很快，仍然需要暂停所有的工作线程。

并发标记

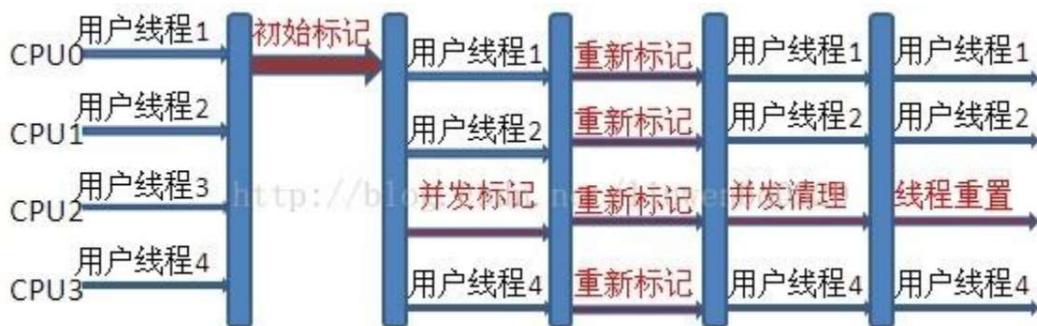
进行 GC Roots 跟踪的过程，和用户线程一起工作，不需要暂停工作线程。

重新标记

为了修正正在并发标记期间，因用户程序继续运行而导致标记产生变动的那一部分对象的标记记录，仍然需要暂停所有的工作线程。

并发清除

清除 GC Roots 不可达对象，和用户线程一起工作，不需要暂停工作线程。由于耗时最长的并发标记和并发清除过程中，垃圾收集线程可以和用户现在一起并发工作，所以总体上来看 CMS 收集器的内存回收和用户线程是一起并发地执行。CMS 收集器工作过程



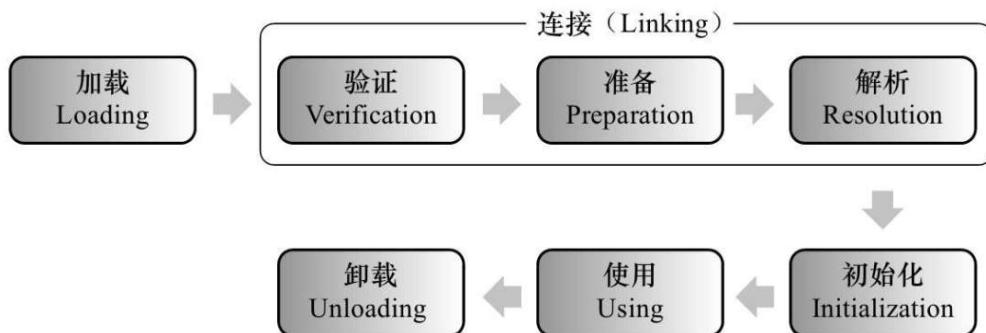
60、G1 收集器

Garbage first 垃圾收集器是目前垃圾收集器理论发展的最前沿成果，相比与 CMS 收集器， G1 收集器两个最突出的改进是：

1. 基于标记-整理算法，不产生内存碎片。
2. 可以非常精确控制停顿时间，在不牺牲吞吐量前提下，实现低停顿垃圾回收。G1 收集器避免全区域垃圾收集，它把堆内存划分为大小固定的几个独立区域，并且跟踪这些区域的垃圾收集进度，同时在后台维护一个优先级列表，每次根据所允许的收集时间，优先回收垃圾最多的区域。区域划分和优先级区域回收机制，确保 G1 收集器可以在有限时间获得最高的垃圾收集效率

61、JVM 类加载机制

JVM 类加载机制分为五个部分：加载，验证，准备，解析，初始化，下面我们就分别来看一下这五个过程。



加载

加载是类加载过程中的一一个阶段，这个阶段会在内存中生成一个代表这个类的 `java.lang.Class` 对象，作为方法区这个类的各种数据的入口。注意这里不一定非得要从一个 Class 文件获取，这里既可以从 ZIP 包中读取（比如从 jar 包和 war 包中读取），也可以在运行时计算生成（动态代理），也可以由其它文件生成（比如将 JSP 文件转换成对应的 Class 类）。

验证

这一阶段的主要目的是为了确保 Class 文件的字节流中包含的信息是否符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。

准备

准备阶段是正式为类变量分配内存并设置类变量的初始值阶段，即在方法区中分配这些变量所使用的内存空间。注意这里所说的初始值概念，比如一个类变量定义为：

实际上变量 v 在准备阶段后的初始值为 0 而不是 8080，将 v 赋值为 8080 的 put static 指令是程序被编译后，存放于类构造器方法之中。

但是注意如果声明为：

```
public static final int v = 8080;
```

在编译阶段会为 v 生成 ConstantValue 属性，在准备阶段虚拟机会根据 ConstantValue 属性将 v 赋值为 8080。

解析

解析阶段是指虚拟机将常量池中的符号引用替换为直接引用的过程。符号引用就是 class 文件中的：

```
public static int v = 8080;
```

实际上变量 v 在准备阶段过后的初始值为 0 而不是 8080，将 v 赋值为 8080 的 put static 指令是程序被编译后，存放于类构造器方法之中。但是注意如果声明为：

在编译阶段会为 v 生成 ConstantValue 属性，在准备阶段虚拟机会根据 ConstantValue 属性将 v 赋值为 8080。

解析

解析阶段是指虚拟机将常量池中的符号引用替换为直接引用的过程。符号引用就是 class 文件中的：

```
public static final int v = 8080;
```

在编译阶段会为 v 生成 ConstantValue 属性，在准备阶段虚拟机会根据 ConstantValue 属性将 v 赋值为 8080。

解析

解析阶段是指虚拟机将常量池中的符号引用替换为直接引用的过程。符号引用就是 class 文件中的：

1. CONSTANT_Class_info
2. CONSTANT_Field_info
3. CONSTANT_Method_info

等类型的常量。

符号引用

符号引用与虚拟机实现的布局无关，引用的目标并不一定要已经加载到内存中。各种虚拟机实现的内存布局可以各不相同，但是它们能接受的符号引用必须是一致的，因为符号引用的字面量形式明确定义在 Java 虚拟机规范的 Class 文件格式中。

直接引用

直接引用可以是指向目标的指针，相对偏移量或是一个能间接定位到目标的句柄。如果有了直接引用，那引用的目标必定已经在内存中存在。

初始化

初始化阶段是类加载最后一个阶段，前面的类加载阶段之后，除了在加载阶段可以自定义类加载器以外，其它操作都由 JVM 主导。到了初始阶段，才开始真正执行类中定义的 Java 程序代码。

类构造器

初始化阶段是执行类构造器方法的过程。方法是由编译器自动收集类中的类变量的赋值操作和静态语句块中的语句合并而成的。虚拟机会保证子方法执行之前，父类的方法已经执行完毕，如果一个类中没有对静态变量赋值也没有静态语句块，那么编译器可以不为这个类生成()方法。注意以下几种情况不会执行类初始化：

1. 通过子类引用父类的静态字段，只会触发父类的初始化，而不会触发子类的初始化。
2. 定义对象数组，不会触发该类的初始化。
3. 常量在编译期间会存入调用类的常量池中，本质上并没有直接引用定义常量的类，不会触发定义常量所在的类。
4. 通过类名获取 Class 对象，不会触发类的初始化。
5. 通过 Class.forName 加载指定类时，如果指定参数 initialize 为 false 时，也不会触发类初始化，其实这个参数是告诉虚拟机，是否要对类进行初始化。
6. 通过 ClassLoader 默认的 loadClass 方法，也不会触发初始化动作。

62、类加载器

虚拟机设计团队把加载动作放到 JVM 外部实现，以便让应用程序决定如何获取所需的类，JVM 提供了 3 种类加载器：

启动类加载器(Bootstrap ClassLoader)

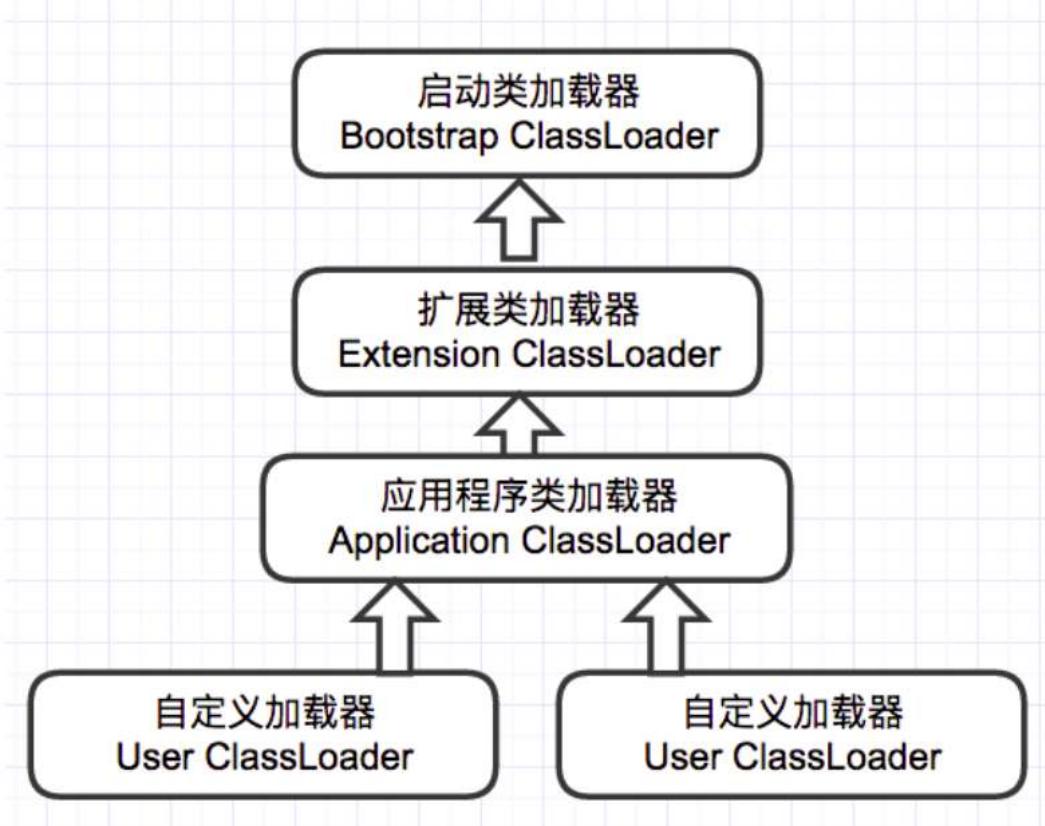
负责加载 JAVA_HOME\lib 目录中的，或通过-Xbootclasspath 参数指定路径中的，且被虚拟机认可（按文件名识别，如 rt.jar）的类。

扩展类加载器(Extension ClassLoader)

负责加载 JAVA_HOME\lib\ext 目录中的，或通过 java.ext.dirs 系统变量指定路径中的类库。

应用程序类加载器(Application ClassLoader)：

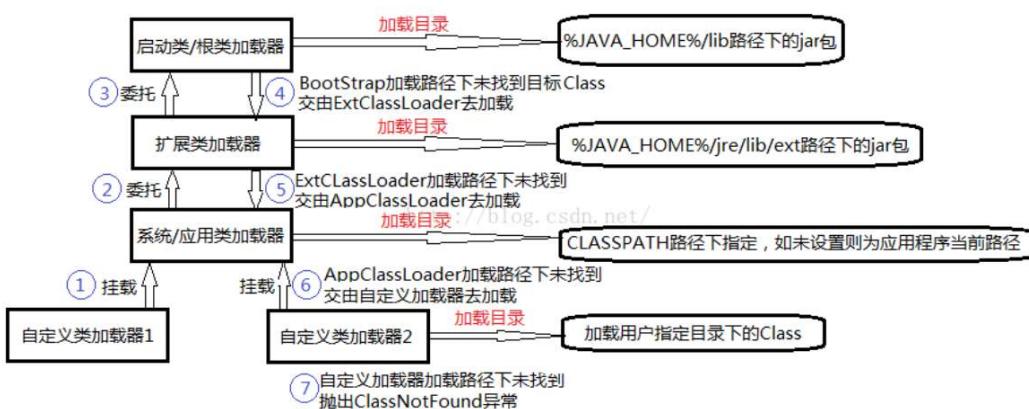
负责加载用户路径 (classpath) 上的类库。JVM 通过双亲委派模型进行类的加载，当然我们也可以通过继承 java.lang.ClassLoader 实现自定义的类加载器。



63、双亲委派

当一个类收到了类加载请求，他首先不会尝试自己去加载这个类，而是把这个请求委派给父类去完成，每一个层次类加载器都是如此，因此所有的加载请求都应该传送到启动类加载其中，只有当父类加载器反馈自己无法完成这个请求的时候（在它的加载路径下没有找到所需加载的Class），子类加载器才会尝试自己去加载。

采用双亲委派的一个好处是比如加载位于 rt.jar 包中的类 java.lang.Object，不管是哪个加载器加载这个类，最终都是委托给顶层的启动类加载器进行加载，这样就保证了使用不同的类加载器最终得到的都是同样一个 Object 对象



64、OSGI (动态模型系统)

OSGi(Open Service Gateway Initiative)，是面向 Java 的动态模型系统，是 Java 动态化模块化系统的一系列规范。

65、动态改变构造

OSGi 服务平台提供在多种网络设备上无需重启的动态改变构造的功能。为了最小化耦合度和促使这些耦合度可管理，OSGi 技术提供一种面向服务的架构，它能使这些组件动态地发现对方。

66、模块化编程与热插拔

OSGi 旨在为实现 Java 程序的模块化编程提供基础条件，基于 OSGi 的程序很可能可以实现模块级的热插拔功能，当程序升级更新时，可以只停用、重新安装然后启动程序的其中一部分，这对企业级程序开发来说是非常具有诱惑力的特性。

OSGi 描绘了一个很美好的模块化开发目标，而且定义了实现这个目标的所需要服务与架构，同时也有成熟的框架进行实现支持。但并非所有的应用都适合采用 OSGi 作为基础架构，它在提供强大功能同时，也引入了额外的复杂度，因为它不遵守了类加载的双亲委托模型。

67、JVM内存模型

线程独占:栈,本地方法栈,程序计数器
线程共享:堆,方法区

68、栈

又称方法栈,线程私有的,线程执行方法是都会创建一个栈帧,用来存储局部变量表,操作栈,动态链接,方法出口等信息.调用方法时执行入栈,方法返回式执行出栈.

69、本地方法栈

与栈类似,也是用来保存执行方法的信息.执行Java方法是使用栈,执行Native方法时使用本地方法栈.

70、程序计数器

保存着当前线程执行的字节码位置,每个线程工作时都有独立的计数器,只为执行Java方法服务,执行Native方法时,程序计数器为空.

71、堆

JVM内存管理最大的一块,对被线程共享,目的是存放对象的实例,几乎所有的对象实例都会放在这里,当堆没有可用空间时,会抛出OOM异常.根据对象的存活周期不同,JVM把对象进行分代管理,由垃圾回收器进行垃圾的回收管理

72、方法区

又称非堆区,用于存储已被虚拟机加载的类信息,常量,静态变量,即时编译器优化后的代码等数据.1.7的永久代和1.8的元空间都是方法区的一种实现。

73、分代回收

分代回收基于两个事实:大部分对象很快就不使用了,还有一部分不会立即无用,但也不会持续很长时间

堆分代			
年轻代	Deden	Survivor1	Survivor2
老年代	Tenured	Tenured	Tenured
永久代	PremGen/MetaSpace	PremGen/MetaSpace	PremGen/MetaSpace

年轻代->标记-复制

老年代->标记-清除

74、堆和栈的区别

栈是运行时单位,代表着逻辑,内含基本数据类型和堆中对象引用,所在区域连续,没有碎片;堆是存储单位,代表着数据,可被多个栈共享(包括成员中基本数据类型、引用和引用对象),所在区域不连续,会有碎片。

1、功能不同

栈内存用来存储局部变量和方法调用,而堆内存用来存储Java中的对象。无论是成员变量,局部变量,还是类变量,它们指向的对象都存储在堆内存中。

2、共享性不同

栈内存是线程私有的。

堆内存是所有线程共有的。

3、异常错误不同

如果栈内存或者堆内存不足都会抛出异常。

栈空间不足:java.lang.StackOverflowError。

堆空间不足:java.lang.OutOfMemoryError。

4、空间大小

栈的空间大小远远小于堆的

75、什么时候会触发FullGC

除直接调用System.gc外，触发Full GC执行的情况有如下四种。

1. 旧生代空间不足

旧生代空间只有在新生代对象转入及创建为大对象、大数组时才会出现不足的现象，当执行Full GC后空间仍然不足，则抛出如下错误：

java.lang.OutOfMemoryError: Java heap space

为避免以上两种状况引起的Full GC，调优时应尽量做到让对象在Minor GC阶段被回收、让对象在新生代多存活一段时间及不要创建过大的对象及数组。

2. Permanet Generation空间满

PermanetGeneration中存放的为一些class的信息等，当系统中要加载的类、反射的类和调用的方法较多时，Permanet Generation可能会被占满，在未配置为采用CMS GC的情况下会执行Full GC。如果经过Full GC仍然回收不了，那么JVM会抛出如下错误信息：

java.lang.OutOfMemoryError: PermGen space

为避免Perm Gen占满造成Full GC现象，可采用的方法为增大Perm Gen空间或转为使用CMS GC。

3. CMS GC时出现promotion failed和concurrent mode failure

对于采用CMS进行旧生代GC的程序而言，尤其要注意GC日志中是否有promotion failed和concurrent mode failure两种状况，当这两种状况出现时可能会触发Full GC。

promotionfailed是在进行Minor GC时，survivor space放不下、对象只能放入旧生代，而此时旧生代也放不下造成的；concurrent mode failure是在执行CMS GC的过程中同时有对象要放入旧生代，而此时旧生代空间不足造成的。

应对措施为：增大survivorspace、旧生代空间或调低触发并发GC的比率，但在JDK 5.0+、6.0+的版本中有可能会由于JDK的bug29导致CMS在remark完毕后很久才触发sweeping动作。对于这种状况，可通过设置-XX:CMSMaxAbortablePrecleanTime=5（单位为ms）来避免。

4. 统计得到的Minor GC晋升到旧生代的平均大小大于旧生代的剩余空间

这是一个较为复杂的触发情况，Hotspot为了避免由于新生代对象晋升到旧生代导致旧生代空间不足的现象，在进行Minor GC时，做了一个判断，如果之前统计所得到的Minor GC晋升到旧生代的平均大小大于旧生代的剩余空间，那么就直接触发Full GC。

例如程序第一次触发MinorGC后，有6MB的对象晋升到旧生代，那么当下一次Minor GC发生时，首先检查旧生代的剩余空间是否大于6MB，如果小于6MB，则执行Full GC。

当新生代采用PSGC时，方式稍有不同，PS GC是在Minor GC后也会检查，例如上面的例子中第一次Minor GC后，PS GC会检查此时旧生代的剩余空间是否大于6MB，如小于，则触发对旧生代的回收。除了以上4种状况外，对于使用RMI来进行RPC或管理的Sun JDK应用而言，默认情况下会一小时执行一次Full GC。可通过在启动时通过-XX:CMSMaxAbortablePrecleanTime=3600000来设置Full GC执行的间隔时间或通过-XX:+DisableExplicitGC来禁止RMI调用System.gc

76、什么是Java虚拟机？为什么Java被称作是“平台无关的编程语言”？

Java虚拟机是一个可以执行Java字节码的虚拟机进程。Java源文件被编译成能被Java虚拟机执行的字节码文件。Java被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。Java虚拟机让这个变为可能，因为它知道底层硬件平台的指令长度和其他特性。

77、对象分配规则

1. 对象优先分配在Eden区，如果Eden区没有足够的空间时，虚拟机执行一次Minor GC。
2. 大对象直接进入老年区（大对象是指需要大量连续内存空间的对象）。这样做的目的是避免在Eden区和两个Survivor区之间发生大量的内存拷贝（新生代采用复制算法收集内存）。
3. 长期存活的对象进入老年区。虚拟机为每个对象定义了一个年龄计数器，如果对象经过了1次Minor GC那么对象会进入Survivor区，之后每经过一次Minor GC那么对象的年龄加1，知道达到阀值对象进入老年区。
4. 动态判断对象的年龄。如果Survivor区中相同年龄的所有对象大小的总和大于Survivor空间的一半，年龄大于或等于该年龄的对象可以直接进入老年区。
5. 空间分配担保。每次进行Minor GC时，JVM会计算Survivor区移至老年区的对象的平均大小，如果这个值大于老年区的剩余值大小则进行一次Full GC，如果小于检查HandlePromotionFailure设置，如果true则只进行Monitor GC，如果false则进行Full GC

78、描述一下JVM加载class文件的原理机制？

JVM中类的装载是由类加载器（ClassLoader）和它的子类来实现的，java中的类加载器是一个重要的Java运行时系统组件，它负责在运行时查找和装入类文件中的类。

由于Java的跨平台性，经过编译的Java源程序并不是一个可执行程序，而是一个或多个类文件。当Java程序需要使用某个类时，JVM会确保这个类已经被加载、连接（验证、准备和解析）和初始化。

类的加载是指把类的.class文件中的数据读入到内存中，通常是创建一个字节数组读入.class文件，然后产生与所加载类对应的Class对象。加载完成后，Class对象还不完整，所以此时的类还不可用。

当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后JVM对类进行初始化，

包括：

1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；

2)如果类中存在初始化语句，就依次执行这些初始化语句。类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader的子类）。

从Java 2 (JDK 1.2) 开始，类加载过程采取了父亲委托机制（PDM）。PDM更好的保证了Java平台的安全性，在该机制中，JVM自带的Bootstrap是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM不会向Java程序提供对Bootstrap的引用。下面是关于几个类加载器的说明

Bootstrap：一般用本地代码实现，负责加载JVM基础核心类库（rt.jar）；
Extension：从java.ext.dirs系统属性所指定的目录中加载类库，它的父加载器是Bootstrap；
System：又叫应用类加载器，其父类是Extension。它是应用最广泛的类加载器。它从环境变量classpath或者系统属性java.class.path所指定的目录中记载类，是用户自定义加载器的默认父加载器。

79、Java对象创建过程

- 1.JVM遇到一条新建对象的指令时首先去检查这个指令的参数是否能在常量池中定义到一个类的符号引用。然后加载这个类（类加载过程在后边讲）
- 2.为对象分配内存。一种办法“指针碰撞”、一种办法“空闲列表”，最终常用的办法“本地线程缓冲分配(TLAB)”
- 3.将除对象头外的对象内存空间初始化为0
- 4.对对象头进行必要设置

80、简述Java的对象结构

Java对象由三个部分组成：对象头、实例数据、对齐填充。
对象头由两部分组成，第一部分存储对象自身的运行时数据：哈希码、GC分代年龄、锁标识状态、线程持有的锁、偏向线程ID（一般占32/64 bit）。第二部分是指针类型，指向对象的类元数据类型（即对象代表哪个类）。如果是数组对象，则对象头中还有一部分用来记录数组长度。
实例数据用来存储对象真正的有效信息（包括父类继承下来的和自己定义的）
对齐填充：JVM要求对象起始地址必须是8字节的整数倍（8字节对齐）

81、如何判断对象可以被回收

判断对象是否存活一般有两种方式：
引用计数：每个对象有一个引用计数属性，新增一个引用时计数加1，引用释放时计数减1，计数为0时可以回收。此方法简单，无法解决对象相互循环引用的问题。
可达性分析（Reachability Analysis）：从GC Roots开始向下搜索，搜索所走过的路径称为引用链。当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的，不可达对象。

82、JVM的永久代中会发生垃圾回收么

垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。如果你仔细查看垃圾收集器的输出信息，就会发现永久代也是被回收的。这就是为什么正确的永久代大小对避免Full GC是非常重要的原因。请参考下Java8：从永久代到元数据区（注：Java8中已经移除了永久代，新加了一个叫做元数据区的native内存区）

83、垃圾收集算法

GC最基础的算法有三种：标记-清除算法、复制算法、标记-压缩算法，我们常用的垃圾回收器一般都采用分代收集算法。
标记-清除算法，“标记-清除”（Mark-Sweep）算法，如它的名字一样，算法分为“标记”和“清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收掉所有被标记的对象。
复制算法，“复制”（Copying）的收集算法，它将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用完了，就将还存活的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。
标记-压缩算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。
分代收集算法，“分代收集”（Generational Collection）算法，把Java堆分为新生代和老年代，这样就可以根据各个年代的特点采用最适当的收集算法

84、调优命令有哪些？

Sun JDK监控和故障处理命令有jps jstat jmap jhat jstack jinfo

1. jps，JVM Process Status Tool,显示指定系统内所有的HotSpot虚拟机进程。
2. jstat，JVM statistics Monitoring是用于监视虚拟机运行时状态信息的命令，它可以显示出虚拟机进程中的类装载、内存、垃圾收集、JIT编译等运行数据。
3. jmap，JVM Memory Map命令用于生成heap dump文件
4. jhat，JVM Heap Analysis Tool命令是与jmap搭配使用，用来分析jmap生成的dump，jhat内置了一个微型的HTTP/HTML服务器，生成dump的分析结果后，可以在浏览器中查看
5. jstack，用于生成java虚拟机当前时刻的线程快照。
6. jinfo，JVM Configuration info 这个命令作用是实时查看和调整虚拟机运行参数

85、调优工具

常用调优工具分为两类，jdk自带监控工具：jconsole和jvisualvm，第三方有：MAT(Memory Analyzer Tool)、GChisto。

1. jconsole，Java Monitoring and Management Console是从java5开始，在JDK中自带的java监控和管理控制台，用于对JVM中内存，线程和类等的监控

2. jvisualvm , jdk自带全能工具，可以分析内存快照、线程快照；监控内存变化、GC变化等。
3. MAT , Memory Analyzer Tool , 一个基于Eclipse的内存分析工具，是一个快速、功能丰富的Javaheap分析工具，它可以帮助我们查找内存泄漏和减少内存消耗
4. GChisto , 一款专业分析gc日志的工具

86、Minor GC与Full GC分别在什么时候发生？

新生代内存不够用时候发生MGC也叫YGC，JVM内存不够的时候发生FGC

87、你知道哪些JVM性能调优

设定堆内存大小
-Xmx : 堆内存最大限制。
设定新生代大小。新生代不宜太小，否则会有大量对象涌入老年代
-XX:NewSize : 新生代大小
-XX:NewRatio 新生代和老生代占比
-XX:SurvivorRatio : 伊甸园空间和幸存者空间的占比
设定垃圾回收器 年轻代用 -XX:+UseParNewGC 老年代用-XX:+UseConcMarkSweepGC

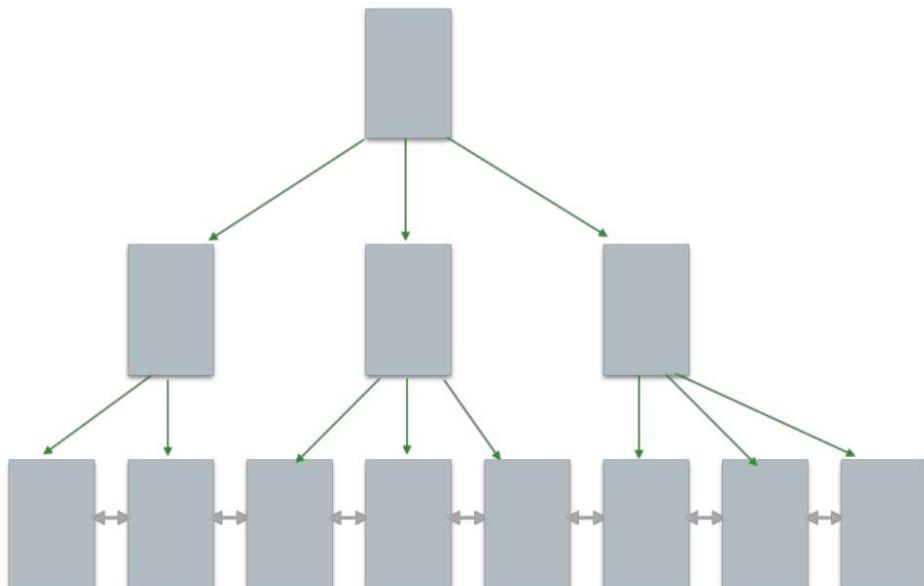
Mysql面试题

1、数据库存储引擎

数据库存储引擎是数据库底层软件组织，数据库管理系统（DBMS）使用数据引擎进行创建、查询、更新和删除数据。不同的存储引擎提供不同的存储机制、索引技巧、锁定水平等功能，使用不同的存储引擎，还可以获得特定的功能。现在许多不同的数据库管理系统都支持多种不同的数据引擎。存储引擎主要有：1. MyISAM, 2. InnoDB, 3. MEMORY, 4. ARCHIVE, 5. FEDERATED。

2、InnoDB (B+树)

InnoDB 底层存储结构为B+树，B树的每个节点对应innodb的一个page，page大小是固定的，一般设为16k。其中非叶子节点只有键值，叶子节点包含完整数据



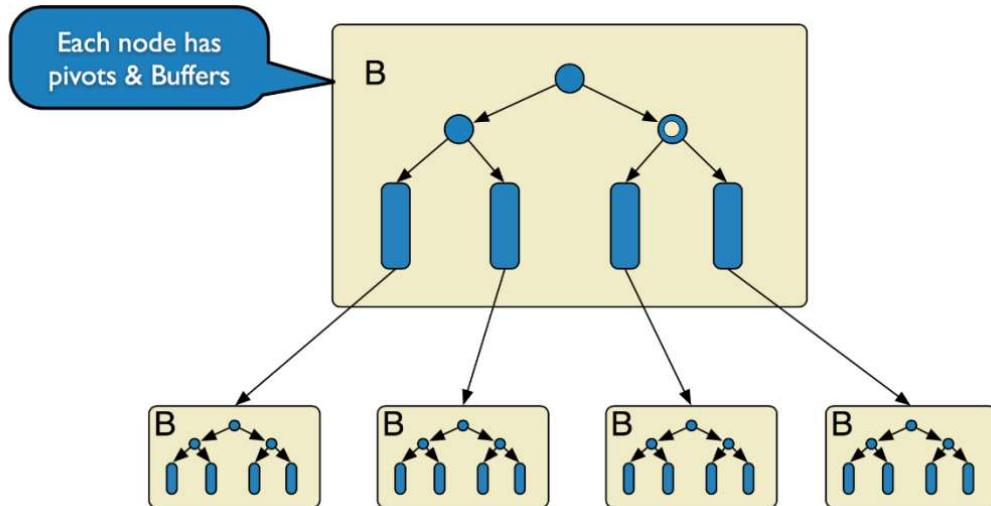
适用场景：

- 1) 经常更新的表，适合处理多重并发的更新请求。
- 2) 支持事务。
- 3) 可以从灾难中恢复（通过 bin-log 日志等）。
- 4) 外键约束。只有他支持外键。
- 5) 支持自动增加列属性 auto_increment。

2、TokuDB (Fractal Tree-节点带数据)

TokuDB 底层存储结构为 Fractal Tree,Fractal Tree 的结构与 B+树有些类似，在 Fractal Tree 中，每一个 child 指针除了需要指向一个 child 节点外，还会带有一个 Message Buffer，这个 Message Buffer 是一个 FIFO 的队列，用来缓存更新操作。

例如，一次插入操作只需要落在某节点的 Message Buffer 就可以马上返回了，并不需要搜索到叶子节点。这些缓存的更新会在查询时或后台异步合并应用到对应的节点中。



TokuDB 在线添加索引，不影响读写操作，非常快的写入性能，Fractal-tree 在事务实现上有优势。它主要适用于访问频率不高的数据或历史数据归档

3、MyISAM

MyISAM 是 MySQL 默认的引擎，但是它没有提供对数据库事务的支持，也不支持行级锁和外键，因此当 INSERT(插入) 或 UPDATE(更新) 数据时即写操作需要锁定整个表，效率便会低一些。

ISAM 执行读取操作的速度很快，而且不占用大量的内存和存储资源。在设计之初就预想数据组织成有固定长度的记录，按顺序存储的。--- ISAM 是一种静态索引结构。

缺点是它不支持事务处理。

4、Memory

Memory (也叫 HEAP) 堆内存：使用存在内存中的内容来创建表。每个 MEMORY 表只实际对应一个磁盘文件。MEMORY 类型的表访问非常得快，因为它的数据是放在内存中的，并且默认使用 HASH 索引。但是一旦服务关闭，表中的数据就会丢失掉。Memory 同时支持散列索引和 B 树索引，B 树索引可以使用部分查询和通配查询，也可以使用 <> 和 >= 等操作符方便数据挖掘，散列索引相等的比较快但是对于范围的比较慢很多

5、数据库引擎有哪些

如何查看mysql提供的所有存储引擎

```
mysql> show engines;
```

MySQL 5.7 Command Line Client

```
Type 'help,' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show engines;
```

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL

mysql常用引擎包括：MYISAM、Innodb、Memory、MERGE

1. MYISAM : 全表锁，拥有较高的执行速度，不支持事务，不支持外键，并发性能差，占用空间相对较小，对事务完整性没有要求，以 select、insert为主的应用基本上可以使用这引擎
2. InnoDB: 行级锁，提供了具有提交、回滚和崩溃恢复能力的事务安全，支持自动增长列，支持外键约束，并发能力强，占用空间是 MYISAM 的 2.5 倍，处理效率相对会差一些
3. Memory: 全表锁，存储在内容中，速度快，但会占用和数据量成正比的内存空间且数据在 MySQL 重启时会丢失，默认使用 HASH 索引，检索效率非常高，但不适用于精确查找，主要用于那些内容变化不频繁的代码表
4. MERGE : 是一组 MYISAM 表的组合

6、InnoDB与MyISAM的区别

1. InnoDB 支持事务，MyISAM 不支持，对于 InnoDB 每一条 SQL 语句都默认封装成事务，自动提交，这样会影响速度，所以最好把多条 SQL 语句放在 begin 和 commit 之间，组成一个事务；
2. InnoDB 支持外键，而 MyISAM 不支持。对一个包含外键的 InnoDB 表转为 MyISAM 会失败；
3. InnoDB 是聚集索引，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此，主键不应该过大，因为主键太大，其他索引也都会很大。而 MyISAM 是非聚集索引，数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。
4. InnoDB 不保存表的具体行数，执行 select count(*) from table 时需要全表扫描。而 MyISAM 用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；
5. InnoDB 不支持全文索引，而 MyISAM 支持全文索引，查询效率上 MyISAM 要高

7、索引

索引 (Index) 是帮助 MySQL 高效获取数据的数据结构。常见的查询算法，顺序查找，二分查找，二叉排序树查找，哈希散列法，分块查找，平衡多路搜索树 B 树 (B-tree)，索引是对数据库表中一个或多个列的值进行排序的结构，建立索引有助于快速获取信息。

你也可以这样理解：索引就是加快检索表中数据的方法。数据库的索引类似于书籍的索引。在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息。在数据库中，索引也允许数据库程序迅速地找到表中的数据，而不必扫描整个数据库。

mysql 有 4 种不同的索引：

- 主键索引 (PRIMARY)
- 唯一索引 (UNIQUE)
- 普通索引 (INDEX)
- 全文索引 (FULLTEXT)

索引并非是越多越好，创建索引也需要耗费资源，一是增加了数据库的存储空间，二是在插入和删除时要花费较多的时间维护索引。

索引加快数据库的检索速度

索引降低了插入、删除、修改等维护任务的速度

唯一索引可以确保每一行数据的唯一性

通过使用索引，可以在查询的过程中使用优化器，提高系统的性能

索引需要占物理和数据空间

8、常见索引原则有

1. 选择唯一性索引，唯一性索引的值是唯一的，可以更快速的通过该索引来确定某条记录。
2. 为经常需要排序、分组和联合操作的字段建立索引。
3. 为常用作为查询条件的字段建立索引。
4. 限制索引的数目：
 - 越多的索引，会使更新表变得很浪费时间。尽量使用数据量少的索引
 - 5. 如果索引的值很长，那么查询的速度会受到影响。尽量使用前缀索引
 - 6. 如果索引字段的值很长，最好使用值的前缀索引。
 - 7. 删除不再使用或者很少使用的索引
 - 8. 最左前缀匹配原则，非常重要的原则。
 - 9. 尽量选择区分度高的列作为索引区分度的公式是表示字段不重复的比例
 - 10. 索引列不能参与计算，保持列“干净”：带函数的查询不参与索引。
 - 11. 尽量的扩展索引，不要新建索引

9、数据库的三范式是什么

第一范式：列不可再分

第二范式：行可以唯一区分，主键约束

第三范式：表的非主属性不能依赖与其他表的非主属性 外键约束

且三大范式是一级一级依赖的，第二范式建立在第一范式上，第三范式建立在第二范式上。

10、第一范式(1st NF - 列都是不可再分)

第一范式的目标是确保每列的原子性：如果每列都是不可再分的最小数据单元（也称为最小的原子单元），则满足第一范式 (1NF)

The diagram illustrates the normalization process from First Normal Form (1NF) to Second Normal Form (2NF). On the left, a 1NF table has multiple rows with the same value in the 'BuyerID' column. On the right, after normalization, each distinct 'BuyerID' value is assigned a unique row, ensuring that no non-key column depends on another non-key column.

BuyerID	Address
1	中国北京市
2	美国纽约市
3	英国利物浦
4	日本东京市
...	...

BuyerID	Country	City
1	中国	北京
1	中国	北京
4	日本	东京
2	美国	纽约
...

11、第二范式(2nd NF - 每个表只描述一件事情)

首先满足第一范式，并且表中非主键列不存在对主键的部分依赖。第二范式要求每个表只描述一件事情。

The diagram illustrates the normalization process from First Normal Form (1NF) to Second Normal Form (2NF). On the left, a 1NF table contains multiple rows for the same order number (001). On the right, the table is split into two: 'Orders' (containing OrderID and OrderDate) and 'Products' (containing ProductID and Price), with a one-to-many relationship established through the OrderID key.

Orders	
字段	例子
订单编号	001
产品编号	A001
订购日期	2000-2-3
价格	\$29.00
...	...

Orders	
字段	例子
订单编号	001
订购日期	2000-2-3

Products	
字段	例子
产品编号	A001
价格	\$29.00

12、第三范式(3rd NF - 不存在对非主键列的传递依赖)

第三范式定义是，满足第二范式，并且表中的列不存在对非主键列的传递依赖。除了主键订单编号外，顾客姓名依赖于非主键顾客编号。

The diagram illustrates the normalization process from Second Normal Form (2NF) to Third Normal Form (3NF). On the left, a 2NF table contains multiple rows for the same customer ID (AB001). On the right, the table is further normalized by removing the dependency of customer name on customer ID, resulting in separate 'Orders' and 'Customer' tables.

Orders	
字段	例子
订单编号	001
订购日期	2000-2-3
顾客编号	AB001
顾客姓名	Tony
...	...

Orders	
字段	例子
订单编号	001
订购日期	2000-2-3
顾客编号	AB001
...	...

13、数据库是事务

事务(TRANSACTION)是作为单个逻辑工作单元执行的一系列操作，这些操作作为一个整体一起向系统提交，要么都执行、要么都不执行。
事务是一个不可分割的工作逻辑单元。事务必须具备以下四个属性，简称 ACID 属性：

原子性 (Atomicity)

1. 事务是一个完整的操作。事务的各步操作是不可分的（原子的）；要么都执行，要么都不执行。
一致性 (Consistency)
2. 当事务完成时，数据必须处于一致状态。

隔离性 (Isolation)

3. 对数据进行修改的所有并发事务是彼此隔离的，这表明事务必须是独立的，它不应以任何方式依赖于或影响其他事务。

永久性 (Durability)

4. 事务完成后，它对数据库的修改被永久保持，事务日志能够保持事务的永久性

14、SQL优化

1. 查询语句中不要使用select *
2. 尽量减少子查询，使用关联查询 (left join,right join,inner join) 替代
3. 减少使用IN或者NOT IN ,使用exists , not exists或者关联查询语句替代
4. or 的查询尽量用 union或者union all 替代(在确认没有重复数据或者不用剔除重复数据时，union all会更好)
5. 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

6、应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如： select id from t where num is null 可以在num上设置默认值0，确保表中num列没有null值，然后这样查询： select id from t where num=0

15. 简单说一说drop、 delete与truncate的区别

SQL中的drop、 delete、 truncate都表示删除，但是三者有一些差别
delete和truncate只删除表的数据不删除表的结构
速度,一般来说: drop> truncate >delete
delete语句是dml,这个操作会放到rollback segment中,事务提交之后才生效;
如果有相应的trigger,执行的时候将被触发. truncate,drop是ddl,操作立即生效,原数据不放到rollbacksegment中,不能回滚. 操作不触发trigger

16. 什么是视图

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询

17. 什么是内联接、 左外联接、 右外联接？

内联接 (Inner Join) : 匹配2张表中相关联的记录。
左外联接 (Left Outer Join) : 除了匹配2张表中相关联的记录外，还会匹配左表中剩余的记录，右表中未匹配到的字段用NULL表示。
右外联接 (Right Outer Join) : 除了匹配2张表中相关联的记录外，还会匹配右表中剩余的记录，左表中未匹配到的字段用NULL表示。在判定左表和右表时，要根据表名出现在Outer Join的左右位置关系

18. 并发事务带来哪些问题？

在典型的应用程序中，多个事务并发运行，经常会操作相同的数据来完成各自的任务（多个用户对同一数据进行操作）。并发虽然是必须的，但可能会导致以下的问题。

脏读 (Dirty read) : 当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。

丢失修改 (Lost to modify) : 指在一个事务读取一个数据时，另外一个事务也访问了该数据，那么在第一个事务中修改了这个数据后，第二个事务也修改了这个数据。这样第一个事务内的修改结果就被丢失，因此称为丢失修。例如：事务1读取某表中的数据A=20，事务2也读取A=20，事务1修改A=A-1，事务2也修改A=A-1，最终结果A=19，事务1的修改被丢失。

不可重复读 (Unrepeatable read) : 指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。

幻读 (Phantom read) : 幻读与不可重复读类似。它发生在一个事务 (T1) 读取了几行数据，接着另一个并发事务 (T2) 插入了一些数据时。在随后的查询中，第一个事务 (T1) 就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。

不可重复读和幻读区别：

不可重复读的重点是修改比如多次读取一条记录发现其中某些列的值被修改，幻读的重点在于新增或者删除比如多次读取一条记录发现记录增多或减少了

19. 事务隔离级别有哪些?MySQL的默认隔离级别是?

SQL 标准定义了四个隔离级别：

READ-UNCOMMITTED(读取未提交) : 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。

READ-COMMITTED(读取已提交) : 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生。

REPEATABLE-READ(可重复读) : 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生

SERIALIZABLE(可串行化) : 最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读

隔离级别	脏读	不可重复读	幻影读
READ-UNCOMMITTED	√	√	√
READ-COMMITTED	✗	√	√
REPEATABLE-READ	✗	✗	√
SERIALIZABLE	✗	✗	✗

MySQL InnoDB 存储引擎的默认支持的隔离级别是 REPEATABLE-READ (可重读) 。我们可以通过
SELECT @@tx_isolation; 命令来查看

```

mysql> SELECT @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+

```

这里需要注意的是：与 SQL 标准不同的地方在于 InnoDB 存储引擎在 REPEATABLE-READ (可重读) 事务隔离级别下使用的是 Next-Key Lock 锁算法，因此可以避免幻读的产生，这与其他数据库系统(如SQL Server)是不同的。所以说InnoDB 存储引擎的默认支持的隔离级别是 REPEATABLE-READ (可重读) 已经可以完全保证事务的隔离性要求，即达到了 SQL 标准的 SERIALIZABLE(可串行化) 隔离级别。因为隔离级别越低，事务请求的锁越少，所以大部分数据库系统的隔离级别都是 READCOMMITTED(读取提交内容)，但是你要知道的是InnoDB 存储引擎默认使用 REPEATABLEREAD (可重读) 并不会有性能损失

InnoDB 存储引擎在 分布式事务 的情况下一般会用到 SERIALIZABLE(可串行化) 隔离级别。

20、大表如何优化？

当MySQL单表记录数过大时，数据库的CRUD性能会明显下降，一些常见的优化措施如下：

限定数据的范围

务必禁止不带任何限制数据范围条件的查询语句。比如：我们当用户在查询订单历史的时候，我们可以控制在一个月的范围内；

读/写分离

经典的数据库拆分方案，主库负责写，从库负责读；

垂直分区

根据数据库里面数据表的相关性进行拆分。例如，用户表中既有用户的登录信息又有用户的基本信息，可以将用户表拆分成两个单独的表，甚至放到单独的库做分库。

简单来说垂直拆分是指数据表列的拆分，把一张列比较多的表拆分为多张表。如下图所示，这样来说大家应该就更容易理解了



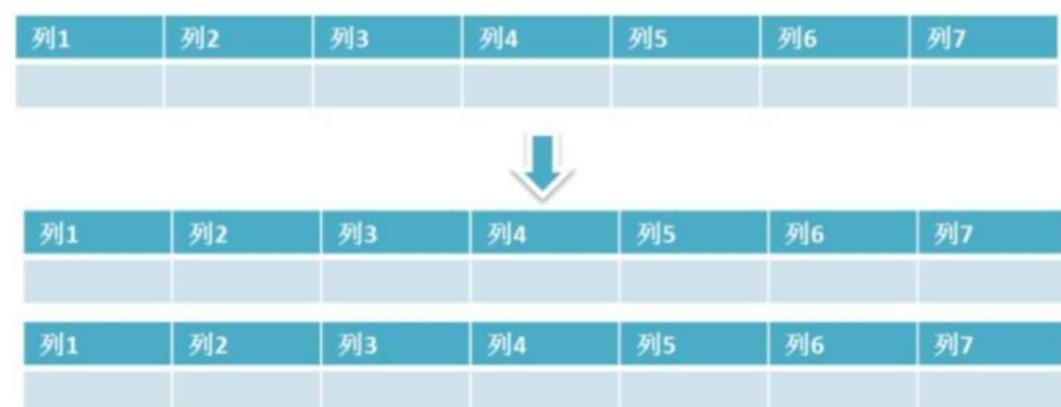
垂直拆分的优点：可以使得列数据变小，在查询时减少读取的Block数，减少I/O次数。此外，垂直分区可以简化表的结构，易于维护。

垂直拆分的缺点：主键会出现冗余，需要管理冗余列，并会引起Join操作，可以通过在应用层进行Join来解决。此外，垂直分区会让事务变得更加复杂；

21、水平分区

保持数据表结构不变，通过某种策略存储数据分片。这样每一片数据分散到不同的表或者库中，达到了分布式的目的。水平拆分可以支撑非常大的数据量。

水平拆分是指数据表行的拆分，表的行数超过200万行时，就会变慢，这时可以把一张的表的数据拆成多张表来存放。举个例子：我们可以将用户信息表拆分成多个用户信息表，这样就可以避免单一表数据量过大对性能造成影响。



水平拆分可以支持非常大的数据量。需要注意的一点是：分表仅仅是解决了单一表数据过大的问题，但由于表的数据还是在同一台机器上，其实对于提升MySQL并发能力没有什么意义，所以水平拆分最好分库。

水平拆分能够支持非常大的数据量存储，应用端改造也少，但分片事务难以解决，跨节点Join性能较差，逻辑复杂。《Java工程师修炼之道》的作者推荐尽量不要对数据进行分片，因为拆分会带来逻辑、部署、运维的各种复杂度，一般的数据表在优化得当的情况下支撑千万以下的数据量是没有太大问题的。如果实在要分片，尽量选择客户端分片架构，这样可以减少一次和中间件的网络I/O。

1. 下面补充一下数据库分片的两种常见方案：

- 客户端代理：分片逻辑在应用端，封装在jar包中，通过修改或者封装JDBC层来实现。当当网的Sharding-JDBC、阿里的TDDL是两种比较常用的实现。
- 2. 中间件代理：在应用和数据中间加了一个代理层。分片逻辑统一维护在中间件服务中。我们现在谈的MyCat、360的Atlas、网易的DDB等等都是这种架构的实现。

详细内容可以参考：MySQL大表优化方案：<https://segmentfault.com/a/1190000006158186>

22、分库分表之后，id 主键如何处理

因为要是分成多个表之后，每个表都是从1开始累加，这样是不对的，我们需要一个全局唯一的id来支持。

生成全局id有下面这几种方式：

UUID：不适合作为主键，因为太长了，并且无序不可读，查询效率低。比较适合用于生成唯一的名字的标示比如文件的名字。

数据库自增id：两台数据库分别设置不同步长，生成不重复ID的策略来实现高可用。这种方式生成的id有序，但是需要独立部署数据库实例，成本高，还会有性能瓶颈。

利用redis生成id：性能比较好，灵活方便，不依赖于数据库。但是，引入了新的组件造成系统更加复杂，可用性降低，编码更加复杂，增加了系统成本。

Twitter的snowflake算法：Github地址：<https://github.com/twitter-archive/snowflake>。

美团的Leaf分布式ID生成系统：Leaf是美团开源的分布式ID生成器，能保证全局唯一性、趋势递增、单调递增、信息安全，里面也提到了几种分布式方案的对比，但也需要依赖关系数据库、Zookeeper等中间件。感觉还不错。美团技术团队的一篇文章：<https://tech.meituan.com/2017/04/21/mt-leaf.html>

23、存储过程(特定功能的SQL语句集)

一组为了完成特定功能的SQL语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象。

24、存储过程优化思路

1. 尽量利用一些sql语句来替代一些小循环，例如聚合函数，求平均函数等。
2. 中间结果存放于临时表，加索引。
3. 少使用游标。sql是个集合语言，对于集合运算具有较高性能。而cursors是过程运算。比如对一个100万行的数据进行查询。游标需要读表100万次，而不使用游标则只需要少量几次读取。
4. 事务越短越好。sqlserver支持并发操作。如果事务过多过长，或者隔离级别过高，都会造成并发操作的阻塞，死锁。导致查询极慢，cpu占用率极地。
5. 使用try-catch处理错误异常。
6. 查找语句尽量不要放在循环内

25、触发器(一段能自动执行的程序)

触发器是一段能自动执行的程序，是一种特殊的存储过程，触发器和普通的存储过程的区别是：触发器是当对某一个表进行操作时触发。诸如：update、insert、delete这些操作的时候，系统会自动调用执行该表上对应的触发器。SQL Server 2005中触发器可以分为两类：DML触发器和DDL触发器，其中DDL触发器它们会影响多种数据定义语言语句而激发，这些语句有create、alter、drop语句。

26、数据库并发策略

并发控制一般采用三种方法，分别是乐观锁和悲观锁以及时间戳。

27、MySQL中有哪几种锁？

- 1、表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。
- 2、行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。
- 3、页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般

28、MySQL中有哪些不同的表格？

共有5种类型的表格：

- 1、MyISAM
- 2、Heap
- 3、Merge
- 4、INNODB
- 5、ISAM

29、简述在 MySQL 数据库中 MyISAM 和 InnoDB 的区别

MyISAM :

不支持事务，但是每次查询都是原子的；
支持表级锁，即每次操作是对整个表加锁；
存储表的总行数；
一个 MYISAM 表有三个文件：索引文件、表结构文件、数据文件；
采用非聚集索引，索引文件的数据域存储指向数据文件的指针。辅索引与主索引基本一致，但是辅索引不用保证唯一性。

InnoDB :

支持 ACID 的事务，支持事务的四种隔离级别；
支持行级锁及外键约束：因此可以支持写并发；

不存储总行数：

一个 InnoDB 引擎存储在一个文件空间（共享表空间，表大小不受操作系统控制，一个表可能分布在多个文件里），也有可能为多个（设置为独立表空，表大小受操作系统文件大小限制，一般为 2G），受操作系统文件大小的限制；
主键索引采用聚集索引（索引的数据域存储数据文件本身），辅索引的数据域存储主键的值；因此从辅索引查找数据，需要先通过辅索引找到主键值，再访问辅索引；最好使用自增主键，防止插入数据时，为维持 B+树结构，文件的大调整。

30、MySQL 中 InnoDB 支持的四种事务隔离级别名称，以及逐级之间的区别？

SQL 标准定义的四个隔离级别为：

- 1、read uncommitted : 读到未提交数据
- 2、read committed : 脏读，不可重复读
- 3、repeatable read : 可重读
- 4、serializable : 串行事物

31、CHAR 和 VARCHAR 的区别？

- 1、CHAR 和 VARCHAR 类型在存储和检索方面有所不同
- 2、CHAR 列长度固定为创建表时声明的长度，长度值范围是 1 到 255 当 CHAR 值被存储时，它们被用空格填充到特定长度，检索 CHAR 值时需删除尾随空格。

32、主键和候选键有什么区别？

表格的每一行都由主键唯一标识。一个表只有一个主键。
主键也是候选键。按照惯例，候选键可以被指定为主键，并且可以用于任何外键引用。

33、myisamchk 是用来做什么的？

它用来压缩 MyISAM 表，这减少了磁盘或内存使用。

34、MyISAM Static 和 MyISAM Dynamic 有什么区别？

在 MyISAM Static 上的所有字段有固定宽度。动态 MyISAM 表将具有像 TEXT, BLOB 等字段，以适应不同长度的数据类型。
MyISAM Static 在受损情况下更容易恢复。

35、如果一个表有一列定义为 TIMESTAMP，将发生什么？

每当行被更改时，时间戳字段将获取当前时间戳。

列设置为 AUTO INCREMENT 时，如果在表中达到最大值，会发生什么情况？

它会停止递增，任何进一步的插入都将产生错误，因为密钥已被使用。

怎样才能找出最后一次插入时分配了哪个自动增量？

LAST_INSERT_ID 将返回由 Auto_increment 分配的最后一个值，并且不需要指定表名称

36、你怎么看到为表格定义的所有索引？

索引是通过以下方式为表格定义的：

```
SHOW INDEX FROM <tablename>;
```

37、LIKE 声明中的 % 和 _ 是什么意思？

%对应于 0 个或更多字符，_只是 LIKE 语句中的一个字符

如何在 Unix 和 MySQL 时间戳之间进行转换？

UNIX_TIMESTAMP 是从 MySQL 时间戳转换为 Unix 时间戳的命令

FROM_UNIXTIME 是从 Unix 时间戳转换为 MySQL 时间戳的命令

38、列对比运算符是什么？

在 SELECT 语句的列比较中使用=, <>, <=, <, >=, >, <<, >>, <=>, AND, OR 或 LIKE 运算符。

39、BLOB 和 TEXT 有什么区别？

BLOB 是一个二进制对象，可以容纳可变数量的数据。TEXT 是一个不区分大小写的 BLOB。

BLOB 和 TEXT 类型之间的唯一区别在于对 BLOB 值进行排序和比较时区分大小写，对 TEXT 值不区分大小写。

40、MySQL_fetch_array 和 MySQL_fetch_object 的区别是什么？

以下是 MySQL_fetch_array 和 MySQL_fetch_object 的区别：

MySQL_fetch_array() - 将结果行作为关联数组或来自数据库的常规数组返回。

MySQL_fetch_object - 从数据库返回结果行作为对象。

41、MyISAM 表格将在哪里存储，并且还提供其存储格式？

每个 MyISAM 表格以三种格式存储在磁盘上：

“.frm”文件存储表定义

数据文件具有“.MYD”（ MYData ）扩展名

索引文件具有“.MYI”（ MYIndex ）扩展名

42、MySQL 如何优化 DISTINCT？

DISTINCT 在所有列上转换为 GROUP BY，并与 ORDER BY 子句结合使用。

SELECT DISTINCT t1.a FROM t1,t2 where t1.a=t2.a;

43、如何显示前 50 行？

在 MySQL 中，使用以下代码查询显示前 50 行：

```
SELECT*FROM    LIMIT 0,50;
```

44、可以使用多少列创建索引？

任何标准表最多可以创建 16 个索引列。

45、NOW() 和 CURRENT_DATE() 有什么区别？

NOW() 命令用于显示当前年份，月份，日期，小时，分钟和秒。

CURRENT_DATE() 仅显示当前年份，月份和日期。

46、什么是非标准字符串类型？

1、TINYTEXT

2、TEXT

3、MEDIUMTEXT

4、LONGTEXT

47、什么是通用 SQL 函数？

1、CONCAT(A, B) - 连接两个字符串值以创建单个字符串输出。通常用于将两个或多个字段合并为一个字段。

- 2、FORMAT(X, D)- 格式化数字 X 到 D 有效数字。
- 3、CURRDATE(), CURRTIME()- 返回当前日期或时间。
- 4、NOW() - 将当前日期和时间作为一个值返回。
- 5、MONTH(), DAY(), YEAR(), WEEK(), WEEKDAY() - 从日期值中提取给定数据。
- 6、HOUR(), MINUTE(), SECOND() - 从时间值中提取给定数据。
- 7、DATEDIFF (A , B) - 确定两个日期之间的差异，通常用于计算年龄
- 8、SUBTIME (A , B) - 确定两次之间的差异。
- 9、FROM_DAYS (INT) - 将整数天数转换为日期值

48、MySQL 支持事务吗？

在缺省模式下，MySQL 是 autocommit 模式的，所有的数据库更新操作都会即时提交，所以在缺省情况下，MySQL 是不支持事务的。但是如果你的 MySQL 表类型是使用 InnoDB Tables 或 BDB tables 的话，你的 MySQL 就可以使用事务处理。使用 SET AUTOCOMMIT=0 就可以使 MySQL 允许在非 autocommit 模式，在非autocommit 模式下，你必须使用 COMMIT 来提交你的更改，或者用 ROLLBACK 来回滚你的更改。

49、MySQL 里记录货币用什么字段类型好

NUMERIC 和 DECIMAL 类型被 MySQL 实现为同样的类型，这在 SQL92 标准允许。他们被用于保存值，该值的准确精度是极其重要的值，例如与金钱有关的数据。当声明一个类是这些类型之一时，精度和规模的能被(并且通常是)指定。

例如：

在这个例子中，9(precision)代表将被用于存储值的总的小数位数，而 2(scale)代表将被用于存储小数点后的位数。因此，在这种情况下，能被存储在 salary 列中的值的范围是从-9999999.99 到 9999999.99。

```
salary DECIMAL(9,2)
```

在这个例子中，9(precision)代表将被用于存储值的总的小数位数，而 2(scale)代表将被用于存储小数点后的位数。因此，在这种情况下，能被存储在 salary 列中的值的范围是从-9999999.99 到 9999999.99。

50、MySQL 有关权限的表都有哪几个？

MySQL 服务器通过权限表来控制用户对数据库的访问，权限表存放在 MySQL 数据库里，由 MySQL_install_db 脚本初始化。这些权限表分别 user , db , table_priv , columns_priv 和 host.

51、列的字符串类型可以是什么？

字符串类型是：

- 1、SET
- 2、BLOB
- 3、ENUM
- 4、CHAR
- 5、TEXT

52、MySQL 数据库作发布系统的存储，一天五万条以上的增量，预计运维三年，怎么优化？

- 1、设计良好的数据库结构，允许部分数据冗余，尽量避免 join 查询，提高效率。
- 2、选择合适的表字段数据类型和存储引擎，适当的添加索引。
- 3、MySQL 库主从读写分离。
- 4、找规律分表，减少单表中的数据量提高查询速度。
- 5、添加缓存机制，比如 memcached , apc 等。
- 6、不经常改动的页面，生成静态页面。
- 7、书写高效率的 SQL。比如 SELECT * FROM TABLE 改为 SELECT field_1,field_2,field_3 FROM TABLE.

53、锁的优化策略

- 1、读写分离
- 2、分段加锁
- 3、减少锁持有的时间
- 4、多个线程尽量以相同的顺序去获取资源不能将锁的粒度过于细化，不然可能会出现线程的加锁和释放次数过多，反而效率不如一次加一把大锁。

54、索引的底层实现原理和优化

B+树，经过优化的 B+树

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此 InnoDB 建议为大部分表使用默认自增的主键作为主索引。

55、什么情况下设置了索引但无法使用

- 1、以“%”开头的 LIKE 语句，模糊匹配
- 2、OR 语句前后没有同时使用索引
- 3、数据类型出现隐式转化（如 varchar 不加单引号的话可能会自动转换为 int 型）

56、实践中如何优化 MySQL

最好是按照以下顺序优化：

- 1、SQL 语句及索引的优化
- 2、数据库表结构的优化
- 3、系统配置的优化
- 4、硬件的优化

详细可以查看 阿里 P8 架构师谈：MySQL 慢查询优化、索引优化、以及表等优化总结

57、优化数据库的方法

- 1、选取最适用的字段属性，尽可能减少定义字段宽度，尽量把字段设置 NOTNULL，例如‘省份’、‘性别’最好适用 ENUM
- 2、使用连接(JOIN)来代替子查询
- 3、适用联合(UNION)来代替手动创建的临时表
- 4、事务处理
- 5、锁定表、优化事务处理
- 6、适用外键，优化锁定表
- 7、建立索引
- 8、优化查询语句

58、简单描述 MySQL 中，索引，主键，唯一索引，联合索引的区别，对数据库的性能有什么影响（从读写两方面）

索引是一种特殊的文件(InnoDB 数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

普通索引(由关键字 KEY 或 INDEX 定义的索引)的唯一任务是加快对数据的访问速度。

普通索引允许被索引的数据列包含重复的值。如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用关键字 UNIQUE 把它定义为一个唯一索引。也就是说，唯一索引可以保证数据记录的唯一性。

主键，是一种特殊的唯一索引，在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字 PRIMARY KEY 来创建。

索引可以覆盖多个数据列，如像 INDEX(columnA, columnB)索引，这就是联合索引。

索引可以极大的提高数据的查询速度，但是会降低插入、删除、更新表的速度，因为在执行这些写操作时，还要操作索引文件

59、数据库中的事务是什么？

事务 (transaction) 是作为一个单元的一组有序的数据库操作。如果组中的所有操作都成功，则认为事务成功，即使只有一个操作失败，事务也不成功。如果所有操作完成，事务则提交，其修改将作用于所有其他数据库进程。如果一个操作失败，则事务将回滚，该事务所有操作的影响都将取消。

事务特性：

- 1、原子性：即不可分割性，事务要么全部被执行，要么就全部不被执行。
- 2、一致性或可串性。事务的执行使得数据库从一种正确状态转换成另一种正确状态
- 3、隔离性。在事务正确提交之前，不允许把该事务对数据的任何改变提供给任何其他事物
- 4、持久性。事务正确提交后，其结果将永久保存在数据库中，即使在事务提交后有了其他故障，事务的处理结果也会得到保存。

或者这样理解：

事务就是被绑定在一起作为一个逻辑工作单元的 SQL 语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上有个节点。为了确保要么执行，要么不执行，就可以使用事务。要将有组语句作为事务考虑，就需要通过 ACID 测试，即原子性，一致性，隔离性和持久性。

60、SQL 注入漏洞产生的原因？如何防止？

SQL 注入产生的原因：程序开发过程中不注意规范书写 sql 语句和对特殊字符进行过滤，导致客户端可以通过全局变量 POST 和 GET 提交一些 sql 语句正常执行。

防止 SQL 注入的方式：

开启配置文件中的 magic_quotes_gpc 和 magic_quotes_runtime 设置 执行 sql 语句时使用 addslashes 进行 sql 语句转换

Sql 语句书写尽量不要省略双引号和单引号。

过滤掉 sql 语句中的一些关键词：update、insert、delete、select、*。

提高数据库表和字段的命名技巧，对一些重要的字段根据程序的特点命名，取不易被猜到的。

61、为表中得字段选择合适得数据类型

字段类型优先级: 整形>date,time>enum,char>varchar>blob,text优先考虑数字类型，其次是日期或者二进制类型，最后是字符串类型，同级别得数据类型，应该优先选择占用空间小的数据类型

62、存储时期

Datetime:以 YYYY-MM-DD HH:MM:SS 格式存储时期时间，精确到秒，占用 8 个字节得存储空间，datetime 类型与时区无关

Timestamp:以时间戳格式存储，占用 4 个字节，范围小 1970-1-1 到 2038-1-19，显示依赖于所指定得时区，默认在第一个列行的数据修改时可以自动得修改

timestamp 列得值

Date: (生日) 占用得字节数比使用字符串.datetime.int 储存要少，使用 date 只需要 3 个字节，存储日期月份，还可以利用日期时间函数进行日期间得计算

Time:存储时间部分得数据

注意:不要使用字符串类型来存储日期时间数据（通常比字符串占用得储存空间小，在进行查找过滤可以利用日期得函数）

使用 int 存储日期时间不如使用 timestamp 类型

63、对于关系型数据库而言，索引是相当重要的概念，请回答有关索引的几个问题

1、索引的目的是什么？

快速访问数据表中的特定信息，提高检索速度

创建唯一性索引，保证数据库表中每一行数据的唯一性。

加速表和表之间的连接

使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间

2、索引对数据库系统的负面影响是什么？

负面影响：

创建索引和维护索引需要耗费时间，这个时间随着数据量的增加而增加；索引需要占用物理空间，不光是表需要占用数据空间，每个索引也需要占用物理空间；当对表进行增、删、改、的时候索引也要动态维护，这样就降低了数据的维护速度。

3、为数据表建立索引的原则有哪些？

在最频繁使用的、用以缩小查询范围的字段上建立索引。

在频繁使用的、需要排序的字段上建立索引

4、什么情况下不宜建立索引？

对于查询中很少涉及的列或者重复值比较多的列，不宜建立索引。

对于一些特殊的数据类型，不宜建立索引，比如文本字段 (text) 等

64、解释 MySQL 外连接、内连接与自连接的区别

先说什么是交叉连接：交叉连接又叫笛卡尔积，它是指不使用任何条件，直接将一个表的所有记录和另一个表中的所有记录一一匹配。

内连接则是只有条件的交叉连接，根据某个条件筛选出符合条件的记录，不符合条件的记录不会出现在结果集中，即内连接只连接匹配的行。

外连接 其结果集中不仅包含符合连接条件的行，而且还会包括左表、右表或两个表中的所有数据行，这三种情况依次称之为左外连接，右外连接，和全外连接。

左外连接，也称左连接，左表为主表，左表中的所有记录都会出现在结果集中，对于那些在右表中并没有匹配的记录，仍然要显示，右边对应的那些字段值以NULL 来填充。右外连接，也称右连接，右表为主表，右表中的所有记录都会出现在结果集中。左连接和右连接可以互换，MySQL 目前还不支持全外连接。

65、Myql 中的事务回滚机制概述

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位，事务回滚是指将该事务已经完成的对数据库的更新操作撤销。

要同时修改数据库中两个不同表时，如果它们不是一个事务的话，当第一个表修改完，可能第二个表修改过程中出现了异常而没能修改，此时就只有第二个表依旧是未修改之前的状态，而第一个表已经被修改完毕。而当你把它们设定为一个事务的时候，当第一个表修改完，第二个表修改出现异常而没能修改，第一个表和第二个表都要回到未修改的状态，这就是所谓的事务回滚

66、SQL 语言包括哪几部分？每部分都有哪些操作关键

SQL 语言包括数据定义(DDL)、数据操纵(DML)、数据控制(DCL)和数据查询 (DQL) 四个部分。

数据定义 : Create Table,Alter Table,Drop Table, Create/Drop Index 等

数据操纵 : Select,insert,update,delete,

数据控制 : grant,revoke

数据查询 : select

67、完整性约束包括哪些？

数据完整性(Data Integrity)是指数据的精确(Accuracy)和可靠性(Reliability)。

分为以下四类：

- 1、实体完整性：规定表的每一行在表中是惟一的实体。
- 2、域完整性：是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。
- 3、参照完整性：是指两个表的主关键字和外关键字的数据应一致，保证了表之间的数据的一致性，防止了数据丢失或无意义的数据在数据库中扩散。
- 4、用户定义的完整性：不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。用户定义的完整性即是针对某个特定关系数据库的约束条件，它反映某一具体应用必须满足的语义要求。与表有关的约束：包括列约束(NOT NULL (非空约束))和表约束(PRIMARY KEY、foreign key、check、UNIQUE)。

68、什么是锁？

答：数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。

加锁是实现数据库并发控制的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。

基本锁类型：锁包括行级锁和表级锁

69、什么叫视图？游标是什么？

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，视图通常是一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

70、什么是存储过程？用什么来调用？

答：存储过程是一个预编译的 SQL 语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次 SQL，使用存储过程比单纯 SQL 语句执行要快。可以用一个命令对象来调用存储过程。

71、如何通俗地理解三个范式？

第一范式：1NF 是对属性的原子性约束，要求属性具有原子性，不可再分解；

第二范式：2NF 是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；

第三范式：3NF 是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。

范式化设计优缺点:

优点:

可以尽量得减少数据冗余，使得更新快，体积小

缺点:对于查询需要多个表进行关联，减少写得效率增加读得效率，更难进行索引

优化

反范式化:

优点:可以减少表得关联，可以更好得进行索引优化

缺点:数据冗余以及数据异常，数据得修改需要更多的成本

72、什么是基本表？什么是视图？

答：基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表

73、试述视图的优点？

- (1) 视图能够简化用户的操作
- (2) 视图使用户能以多种角度看待同一数据；
- (3) 视图为数据库提供了一定程度的逻辑独立性；
- (4) 视图能够对机密数据提供安全保护

74、NULL 是什么意思

NULL 这个值表示 UNKNOWN(未知)；它不表示“”(空字符串)。对 NULL 这个值的任何比较都会生产一个 NULL 值。您不能把任何值与一个 NULL 值进行比较，并在逻辑上希望获得一个答案。使用 IS NULL 来进行 NULL 判断

75、主键、外键和索引的区别？

主键、外键和索引的区别

定义：

主键-唯一标识一条记录，不能有重复的，不允许为空

外键-表的外键是另一表的主键，外键可以有重复的，可以是空值

索引-该字段没有重复值，但可以有一个空值

作用：

主键-用来保证数据完整性

外键-用来和其他表建立联系用的

索引-是提高查询排序的速度

个数：

主键-主键只能有一个

外键-一个表可以有多个外键

索引-一个表可以有多个唯一索引

76、你可以用什么来确保表格里的字段只接受特定范围里的值？

Check 限制，它在数据库表格里被定义，用来限制输入该列的值。触发器也可以被用来限制数据库表格里的字段能够接受的值，但是这种方法要求触发器在表格里被定义，这可能会影响性能。

77、说说对 SQL 语句优化有哪些方法？（选择几条）

1、Where 子句中：where 表之间的连接必须写在其他 Where 条件之前，那些可以过滤掉最大数量记录的条件必须写在 Where 子句的末尾，HAVING 最后。

2、用 EXISTS 替代 IN、用 NOT EXISTS 替代 NOT IN。

3、避免在索引列上使用计算

4、避免在索引列上使用 IS NULL 和 IS NOT NULL

5、对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

6、应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描

7、应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描

78、什么是乐观锁

乐观锁认为一个用户读数据的时候，别人不会去写自己所读的数据；悲观锁就刚好相反，觉得自己读数据库的时候，别人可能刚好在写自己刚读的数据，其实就是持一种比较保守的态度；时间戳就是不加锁，通过时间戳来控制并发出现的问题。

79、什么是悲观锁

悲观锁就是在读取数据的时候，为了不让别人修改自己读取的数据，就会先对自己读取的数据加锁，只有自己把数据读完了，才允许别人修改那部分数据，或者反过来说，就是自己修改某条数据的时候，不允许别人读取该数据，只有等自己的整个事务提交了，才释放自己加上的锁，才允许其他用户访问那部分数据。

80、什么是时间戳

时间戳就是在数据库表中单独加一列时间戳，比如“TimeStamp”，每次读出来的时候，把该字段也读出来，当写回去的时候，把该字段加 1，提交之前，跟数据库的该字段比较一次，如果比数据库的值大的话，就允许保存，否则不允许保存，这种处理方法虽然不使用数据库系统提供的锁机制，但是这种方法可以大大提高数据库处理的并发量，以上悲观锁所说的加“锁”，其实分为几种锁，分别是：排它锁（写锁）和共享锁（读锁）。

81、什么是行级锁

行级锁是一种排他锁，防止其他事务修改此行；在使用以下语句时，Oracle 会自动应用行级锁：

1. INSERT、UPDATE、DELETE、SELECT ... FOR UPDATE [OF columns] [WAIT n | NOWAIT];
2. SELECT ... FOR UPDATE 语句允许用户一次锁定多条记录进行更新
3. 使用 COMMIT 或 ROLLBACK 语句释放锁。

82、什么是表级锁

表示对当前操作的整张表加锁，它实现简单，资源消耗较少，被大部分 MySQL 引擎支持。最常使用的 MYISAM 与 INNODB 都支持表级锁定。表级锁定分为表共享读锁（共享锁）与表独占写锁（排他锁）。

83、什么是页级锁

页级锁是 MySQL 中锁定粒度介于行级锁和表级锁之间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折衷的页级，一次锁定相邻的一组记录。BDB 支持页级锁

Redis面试题

1、什么是 Redis?

Redis 是完全开源免费的，遵守 BSD 协议，是一个高性能的 key-value 数据库。

Redis 与其他 key - value 缓存产品有以下三个特点：

Redis 支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

Redis 不仅仅支持简单的 key-value 类型的数据，同时还提供 list , set , zset , hash 等数据结构的存储。

Redis 支持数据的备份，即 master-slave 模式的数据备份。

Redis 优势

性能极高 – Redis 能读的速度是 110000 次/s,写的速度是 81000 次/s 。丰富的数据类型 – Redis 支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

原子 – Redis 的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过 MULTI 和 EXEC 指令包起来。

丰富的特性 – Redis 还支持 publish/subscribe, 通知, key 过期等等特性。

2、Redis 与其他 key-value 存储有什么不同？

Redis 有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis 的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。Redis 运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样 Redis 可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

3、Redis 的数据类型？

Redis 支持五种数据类型：string (字符串) , hash (哈希) , list (列表) , set (集合) 及 zset sorted set : 有序集合)。

我们实际项目中比较常用的是 string , hash 如果你是 Redis 中高级用户，还需要加上下面几种数据结构 HyperLogLog、Geo、Pub/Sub。

如果说还玩过 Redis Module , 像 BloomFilter , RedisSearch , Redis-ML , 面试官得眼睛就开始发亮了。

4、使用 Redis 有哪些好处？

- 1、速度快，因为数据存在内存中，类似于 HashMap , HashMap 的优势就是查找和操作的时间复杂度都是 O1)
- 2、支持丰富数据类型，支持 string , list , set , Zset , hash 等
- 3、支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行
- 4、丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除

5、Redis 相比 Memcached 有哪些优势？

- 1、Memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类
- 2、Redis 的速度比 Memcached 快很
- 3、Redis 可以持久化其数据

6、Memcache 与 Redis 的区别都有哪些？

- 1、存储方式 Memcache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。Redis 有部份存在硬盘上，这样能保证数据的持久性。
- 2、数据支持类型 Memcache 对数据类型支持相对简单。Redis 有复杂的数据类型。
- 3、使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。Redis 直接自己构建了 VM 机制，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

7、Redis 是单进程单线程的？

Redis 是单进程单线程的，redis 利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销。

8、一个字符串类型的值能存储最大容量是多少？

512M

9、Redis持久化机制

Redis是一个支持持久化的内存数据库，通过持久化机制把内存中的数据同步到硬盘文件来保证数据持久化。当Redis重启后通过把硬盘文件重新加载到内存，就能达到恢复数据的目的。

实现：单独创建fork()一个子进程，将当前父进程的数据库数据复制到子进程的内存中，然后由子进程写入到临时文件中，持久化的过程结束了，再用这个临时文件替换上次的快照文件，然后子进程退出，内存释放。

RDB是Redis默认的持久化方式。按照一定的时间周期策略把内存的数据以快照的形式保存到硬盘的二进制文件。即Snapshot快照存储，对应产生的数据文件为dump.rdb，通过配置文件中的save参数来定义快照的周期。（快照可以是其所表示的数据的一个副本，也可以是数据的一个复制品。）

AOF：Redis会将每一个收到的写命令都通过Write函数追加到文件最后，类似于MySQL的binlog。当Redis重启时会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。当两种方式同时开启时，数据恢复Redis会优先选择AOF恢复。

10、缓存雪崩、缓存穿透、缓存预热、缓存更新、缓存降级等问题

一、缓存雪崩

我们可以简单的理解为：由于原有缓存失效，新缓存未到期（例如：我们设置缓存时采用了相同的过期时间，在同一时刻出现大面积的缓存过期），所有原本应该访问缓存的请求都去查询数据库了，而对数据库CPU和内存造成巨大压力，严重的会造成数据库宕机。从而形成一系列连锁反应，造成整个系统崩溃。

解决办法：

大多数系统设计者考虑用加锁（最多的解决方案）或者队列的方式保证不会有大量的线程对数据库一次性进行读写，从而避免失效时大量的并发请求落到底层存储系统上。还有一个简单方案就是将缓存失效时间分散开。

二、缓存穿透

缓存穿透是指用户查询数据，在数据库没有，自然在缓存中也不会有。这样就导致用户查询的时候，在缓存中找不到，每次都要去数据库再查询一遍，然后返回空（相当于进行了两次无用的查询）。这样请求就绕过缓存直接查数据库，这也是经常提的缓存命中率问题。

解决办法：

最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。另外也有一个更为简单粗暴的方法，如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。通过这个直接设置的默认值放到缓存，这样第二次到缓冲中获取就有值了，而不会继续访问数据库，这种办法最简单粗暴。

5TB的硬盘上放满了数据，请写一个算法将这些数据进行排重。如果这些数据是一些32bit大小的数据该如何解决？如果是64bit的呢？

对于空间的利用到达了一种极致，那就是Bitmap和布隆过滤器(Bloom Filter)。

Bitmap：典型的就是哈希表

缺点是，Bitmap对于每个元素只能记录1bit信息，如果还想完成额外的功能，恐怕只能靠牺牲更多的空间、时间来完成了

布隆过滤器（推荐）

就是引入了 $k(k>1)$ 个相互独立的哈希函数，保证在给定的空间、误判率下，完成元素判断的过程。

它的优点是空间效率和查询时间都远远超过一般的算法，缺点是有一定的误识别率和删除困难。

Bloom-Filter算法的核心思想就是利用多个不同的Hash函数来解决“冲突”。

Hash存在一个冲突（碰撞）的问题，用同一个Hash得到的两个URL的值有可能相同。为了减少冲突，我们可以多引入几个Hash，如果通过其中的一个Hash值我们得出某元素不在集合中，那么该元素肯定不在集合中。只有在所有的Hash函数告诉我们该元素在集合中时，才能确定该元素存在于集合中。这便是Bloom-Filter的基本思想。

Bloom-Filter一般用于在大数据量的集合中判定某元素是否存在。

三、缓存预热

缓存预热这个应该是一个比较常见的概念，相信很多小伙伴都应该可以很容易的理解，缓存预热就是系统上线后，将相关的缓存数据直接加载到缓存系统。这样就可以避免在用户请求的时候，先查询数据库，然后再将数据缓存的问题！用户直接查询事先被预热的缓存数据！

解决思路：

- 1、直接写个缓存刷新页面，上线时手工操作下；
- 2、数据量不大，可以在项目启动的时候自动进行加载；
- 3、定时刷新缓存

四、缓存更新

除了缓存服务器自带的缓存失效策略之外（Redis默认的有6中策略可供选择），我们还可以根据具体的业务需求进行自定义的缓存淘汰，常见的策略有两种：

- (1) 定时去清理过期的缓存；
- (2) 当有用户请求过来时，再判断这个请求所用到的缓存是否过期，过期的话就去底层系统得到新数据并更新缓存。

两者各有优劣，第一种的缺点是维护大量缓存的key是比较麻烦的，第二种的缺点就是每次用户请求过来都要判断缓存失效，逻辑相对比较复杂！具体用哪种方案，大家可以根据自己的应用场景来权衡

五、缓存降级

当访问量剧增、服务出现问题（如响应时间慢或不响应）或非核心服务影响到核心流程的性能时，仍然需要保证服务还是可用的，即使是有损服务。系统可以根据一些关键数据进行自动降级，也可以配置开关实现人工降级。

降级的最终目的是保证核心服务可用，即使是有损的。而且有些服务是无法降级的（如加入购物车、结算）。

以参考日志级别设置预案：

- (1) 一般：比如有些服务偶尔因为网络抖动或者服务正在上线而超时，可以自动降级；
- (2) 警告：有些服务在一段时间内成功率有波动（如在95~100%之间），可以自动降级或人工降级，并发送告警；
- (3) 错误：比如可用率低于90%，或者数据库连接池被打爆了，或者访问量突然猛增到系统能承受的最大阈值，此时可以根据情况自动降级或者人工降级；
- (4) 严重错误：比如因为特殊原因数据错误了，此时需要紧急人工降级。服务降级的目的，是为了防止Redis服务故障，导致数据库跟着一起发生雪崩问题。因此，对于不重要的缓存数据，可以采取服务降级策略，例如一个比较常见的做法就是，Redis出现问题，不去数据库查询，而是直接返回默认值给用户

11、热点数据和冷数据是什么

热点数据，缓存才有价值

对于冷数据而言，大部分数据可能还没有再次访问到就已经被挤出内存，不仅占用内存，而且价值不大。频繁修改的数据，看情况考虑使用缓存

对于上面两个例子，寿星列表、导航信息都存在一个特点，就是信息修改频率不高，读取通常非常高的场景。

对于热点数据，比如我们的某IM产品，生日祝福模块，当天的寿星列表，缓存以后可能读取数十万次。再举个例子，某导航产品，我们将导航信息，缓存以后可能读取数百万次。

数据更新前至少读取两次，缓存才有意义。这个是最基本的策略，如果缓存还没有起作用就失效了，那就没有太大价值了。

那存不存在，修改频率很高，但是又不得不考虑缓存的场景呢？有！比如，这个读取接口对数据库的压力很大，但是又是热点数据，这个时候就需要考虑通过缓存手段，减少数据库的压力，比如我们的某助手产品的，点赞数，收藏数，分享数等是非常典型的热点数据，但是又不断变化，此时就需要将数据同步保存到Redis缓存，减少数据库压力

12、单线程的redis为什么这么快

(一)纯内存操作

(二)单线程操作，避免了频繁的上下文切换

(三)采用了非阻塞I/O多路复用机制

13、redis的数据类型，以及每种数据类型的使用场景

一共五种

(一)String

这个其实没啥好说的，最常规的set/get操作，value可以是String也可以是数字。一般做一些复杂的计数功能的缓存。

(二)hash

这里value存放的是结构化的对象，比较方便的就是操作其中的某个字段。博主在做单点登录的时候，就是用这种数据结构存储用户信息，以cookieId作为key，设置30分钟为缓存过期时间，能很好的模拟出类似session的效果。

(三)list

使用List的数据结构，可以做简单的消息队列的功能。另外还有一个就是，可以利用lrange命令，做基于redis的分页功能，性能极佳，用户体验好。本人还用一个场景，很合适—取行情信息。也就是个生产者和消费者的场景。LIST可以很好的完成排队，先进先出的原则。

(四)set

因为set堆放的是一堆不重复值的集合。所以可以做全局去重的功能。为什么不用JVM自带的Set进行去重？因为我们的系统一般都是集群部署，使用JVM自带的Set，比较麻烦，难道为了一个做一个全局去重，再起一个公共服务，太麻烦了。

另外，就是利用交集、并集、差集等操作，可以计算共同喜好，全部的喜好，自己独有的喜好等功能。

(五)sorted set

sorted set多了一个权重参数score，集合中的元素能够按score进行排列。可以做排行榜应用，取TOP N操作

14、redis的过期策略以及内存淘汰机制

redis采用的是定期删除+惰性删除策略。

为什么不用定时删除策略？

定时删除,用一个定时器来负责监视key过期则自动删除。虽然内存及时释放，但是十分消耗CPU资源。在大并发请求下，CPU要将时间应用在处理请求，而不是删除key,因此没有采用这一策略。

定期删除+惰性删除是如何工作的呢？

定期删除，redis默认每个100ms检查，是否有过期的key,有过期key则删除。需要说明的是，redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms,全部key进行检查，redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。于是，惰性删除派上用场。也就是说在你获取某个key的时候，redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。

采用定期删除+惰性删除就没其他问题了么？

不是的，如果定期删除没删除key。然后你也没即时去请求key，也就是说惰性删除也没生效。这样，redis的内存会越来越高。那么就应该采用内存淘汰机制。在redis.conf中有一行配置

```
maxmemory-policy volatile-lru
```

该配置就是配内存淘汰策略的(什么，你没配过？好好反省一下自己)

volatile-lru：从已设置过期时间的数据集 (server.db[i].expires) 中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集 (server.db[i].expires) 中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集 (server.db[i].expires) 中任意选择数据淘汰

allkeys-lru：从数据集 (server.db[i].dict) 中挑选最近最少使用的数据淘汰

allkeys-random：从数据集 (server.db[i].dict) 中任意选择数据淘汰

no-eviction (驱逐)：禁止驱逐数据，新写入操作会报错

ps：如果没有设置 expire 的key, 不满足先决条件(prerequisites); 那么 volatile-lru, volatile-random 和 volatile-ttl 策略的行为, 和 noeviction(不删除) 基本上一致

15、Redis 常见性能问题和解决方案？

- (1) Master 最好不要做任何持久化工作，如 RDB 内存快照和 AOF 日志文件
- (2) 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一次
- (3) 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网内
- (4) 尽量避免在压力很大的主库上增加从库
- (5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

16、为什么Redis的操作是原子性的，怎么保证原子性的？

对于Redis而言，命令的原子性指的是：一个操作的不可以再分，操作要么执行，要么不执行。

Redis的操作之所以是原子性的，是因为Redis是单线程的。

Redis本身提供的所有API都是原子操作，Redis中的事务其实是要保证批量操作的原子性。

多个命令在并发中也是原子性的吗？

不一定，将get和set改成单命令操作，incr。使用Redis的事务，或者使用Redis+Lua==的方式实现。

17、Redis事务

Redis事务功能是通过MULTI、EXEC、DISCARD和WATCH 四个原语实现的

Redis会将一个事务中的所有命令序列化，然后按顺序执行。

1.redis 不支持回滚“Redis 在事务失败时不进行回滚，而是继续执行余下的命令”，所以 Redis 的内部可以保持简单且快速。

2.如果在一个事务中的命令出现错误，那么所有的命令都不会执行；

3.如果在一个事务中出现运行错误，那么正确的命令会被执行。

1) MULTI命令用于开启一个事务，它总是返回OK。MULTI执行之后，客户端可以继续向服务器发送任意多条命令，这些命令不会立即被执行，而是被放到一个队列中，当EXEC命令被调用时，所有队列中的命令才会被执行。

2) EXEC：执行所有事务块内的命令。返回事务块内所有命令的返回值，按命令执行的先后顺序排列。当操作被打断时，返回空值 nil。

3) 通过调用DISCARD，客户端可以清空事务队列，并放弃执行事务，并且客户端会从事务状态中退出。

4) WATCH 命令可以为 Redis 事务提供 check-and-set (CAS) 行为。可以监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不会执行，监控一直持续到EXEC命令

18、Redis 的持久化机制是什么？各自的优缺点？

Redis 提供两种持久化机制 RDB 和 AOF 机制：

1、RDB(Redis DataBase)持久化方式：是指用数据集快照的方式半持久化模式)记录 redis 数据库的所有键值对,在某个时间点将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。

优点：

- 1、只有一个文件 dump.rdb，方便持久化。
- 2、容灾性好，一个文件可以保存到安全的磁盘。
- 3、性能最大化，fork 子进程来完成写操作，让主进程继续处理命令，所以是 IO最大话。使用单独子进程来进行持久化，主进程不会进行任何 IO 操作，保证了 redis的高性能!
- 4.相对于数据集大时，比 AOF 的启动效率更高。

缺点：

1、数据安全性低。RDB 是间隔一段时间进行持久化，如果持久化之间 redis 发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候)

2、AOF(Append-only file)持久化方式：是指所有的命令行记录以 redis 命令请求协议的格式完全持久化存储)保存为 aof 文件。

优点：

- 1、数据安全，aof 持久化可以配置 appendfsync 属性，有 always，每进行一次命令操作就记录到 aof 文件中一次。
- 2、通过 append 模式写文件，即使中途服务器宕机，可以通过 redis-check-aof 工具解决数据一致性问题。
- 3、AOF 机制的 rewrite 模式。AOF 文件没被 rewrite 之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的 flushall））

缺点：

- 1、AOF 文件比 RDB 文件大，且恢复速度慢。
- 2、数据集大的时候，比 rdb 启动效率低。

19、Redis 常见性能问题和解决方案：

- 1、Master 最好不要写内存快照，如果 Master 写内存快照，save 命令调度 rdbSave 函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务。
- 2、如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步—3、为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网
- 4、尽量避免在压力很大的主库上增加从
- 5、主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...这样的结构方便解决单点故障问题，实现 Slave 对 Master 的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变。

20、redis 过期键的删除策略？

- 1、定时删除：在设置键的过期时间的同时，创建一个定时器 timer。让定时器在键的过期时间来临时，立即执行对键的删除操作。
- 2、惰性删除：放任键过期不管，但是每次从键空间中获取键时，都检查取得的键是否过期，如果过期的话，就删除该键；如果没有过期，就返回该键。
- 3、定期删除：每隔一段时间程序就对数据库进行一次检查，删除里面的过期键。至于要删除多少过期键，以及要检查多少个数据库，则由算法决定。

21、Redis 的回收策略（淘汰策略）？

- volatile-lru**：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰
- volatile-ttl**：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰
- volatile-random**：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰
- allkeys-lru**：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰
- allkeys-random**：从数据集（server.db[i].dict）中任意选择数据淘汰
- no-eviction（驱逐）**：禁止驱逐数据

注意这里的 6 种机制，volatile 和 allkeys 规定了是对已设置过期时间的数据集淘汰数据还是从全部数据集淘汰数据，后面的 lru、ttl 以及 random 是三种不同的淘汰策略，再加上一种 no-eviction 永不回收的策略

使用策略规则：

- 1、如果数据呈现幂律分布，也就是一部分数据访问频率高，一部分数据访问频率低，则使用 allkeys-lru
- 2、如果数据呈现平等分布，也就是所有的数据访问频率都相同，则使用 allkey

22、为什么 Redis 需要把所有数据放到内存中？

Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 Redis 具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘 I/O 速度为严重影响 Redis 的性能。在内存越来越便宜的今，Redis 将会越来越受欢迎。如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

23、Redis 的同步机制了解么？

Redis 可以使用主从同步，从从同步。第一次同步时，主节点做一次 bgsave，并同时将后续修改操作记录到内存 buffer，待完成后将 rdb 文件全量同步到复制节点，复制节点接受完成后将 rdb 镜像加载到内存。加载完成后，再通知主节点将期间修改的操作记录同步到复制节点进行重放就完成了同步过程。

24、Pipeline 有什么好处，为什么要用 pipeline？

可以将多次 IO 往返的时间缩减为一次，前提是 pipeline 执行的指令之间没有因果相关性。使用 redis-benchmark 进行压测的时候可以发现影响 Redis 的 QPS 峰值的一个重要因素是 pipeline 批次指令的数目。

25、是否使用过 Redis 集群，集群的原理是什么？

- 1)、Redis Sentinel 着眼于高可用，在 master 宕机时会自动将 slave 提升为 master，继续提供服务。
- 2)、Redis Cluster 着眼于扩展性，在单个 Redis 内存不足时，使用 Cluster 进行分片存储。

26、Redis 集群方案什么情况下会导致整个集群不可用？

答：有 A , B , C 三个节点的集群，在没有复制模型的情况下，如果节点 B 失败了，那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。

27、Redis 支持的 Java 客户端都有哪些？官方推荐用哪个？

Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

28、Jedis 与 Redisson 对比有什么优缺点？

Jedis 是 Redis 的 Java 实现的客户端，其 API 提供了比较全面的 Redis 命令的支持；Redisson 实现了分布式和可扩展的 Java 数据结构，和 Jedis 相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等 Redis 特性。Redisson 的宗旨是促进使用者对 Redis 的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

29、Redis 如何设置密码及验证密码？

设置密码：config set requirepass 123456

授权密码：auth 123456

30、说说 Redis 哈希槽的概念？

Redis 集群没有使用一致性 hash，而是引入了哈希槽的概念，Redis 集群有 16384 个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置哪个槽，集群的每个节点负责一部分 hash 槽。

31、Redis 集群的主从复制模型是怎样的？

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型，每个节点都会有 N-1 个复制品。

32、Redis 集群会有写操作丢失吗？为什么？

Redis 并不能保证数据的强一致性，这意味着在实际中集群在特定的条件下可能会丢失写操作。

33、Redis 集群之间是如何复制的？

异步复制

34、Redis 集群最大节点个数是多少？

16384 个

35、Redis 集群如何选择数据库？

Redis 集群目前无法做数据库选择，默认在 0 数据库。

36、怎么测试 Redis 的连通性？

使用 ping 命令

37、怎么理解 Redis 事务？

1) 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

2) 事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

38、Redis 事务相关的命令有哪几个？

MULTI、EXEC、DISCARD、WATCH

39、Redis key 的过期时间和永久有效分别怎么设置？

EXPIRE 和 PERSIST 命令。

40、Redis 如何做内存优化？

尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key，而是应该把这个用户的所有信息存储到一张散列表里面。

41、Redis 回收进程如何工作的？

一个客户端运行了新的命令，添加了新的数据。Redis 检查内存使用情况，如果大于 maxmemory 的限制，则根据设定好的策略进行回收。一个新的命令被执行，等等。所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地收回回到边界以下。如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

42、都有哪些办法可以降低 Redis 的内存使用情况呢？

如果你使用的是 32 位的 Redis 实例，可以好好利用 Hash, list, sorted set, set 等集合类型数据，因为通常情况下很多小的 Key-Value 可以用更紧凑的方式存放到一起。

43、Redis 的内存用完了会发生什么？

如果达到设置的上限，Redis 的写命令会返回错误信息（但是读命令还可以正常返回。）或者你可以将 Redis 当缓存来使用配置淘汰机制，当 Redis 达到内存上限时会冲刷掉旧的内容。

44、一个 Redis 实例最多能存放多少的 keys？List、Set、Sorted Set 他们最多能存放多少元素

理论上 Redis 可以处理多达 232 的 keys，并且在实际中进行了测试，每个实例至少存放了 2 亿 5 千万的 keys。我们正在测试一些较大的值。任何 list、set、和 sorted set 都可以放 232 个元素。换句话说，Redis 的存储极限是系统中的可用内存值。

45、MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据？

Redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。相关知识：Redis 提供 6 种数据淘汰策略：

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-eviction（驱逐）：禁止驱逐数据

46、Redis 最适合的场景？

1、会话缓存（Session Cache）

最常用的一种使用 Redis 的情景是会话缓存（session cache）。用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？幸运的是，随着 Redis 这些年的改进，很容易

易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台 Magento 也提供 Redis 的插件。

2、全页缓存（FPC）

除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。再次以agento 为例，Magento 提供一个插件来使用 Redis 作为全页缓存后端。此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

3、队列

Redis 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。如果你快速的在 Google 中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如，Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。

4、排行榜/计数器

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变得非常简单，Redis 只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的 10 个用户—我们称之为“user_scores”，我们只需要像下面一样执行即可：当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行：ZRANGE user_scores 0 10 WITHSCORES Agora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 Redis 来存储数据的，你可以在这里看到。

5、发布/订阅

最后（但肯定不是最不重要的）是 Redis 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 Redis 的发布/订阅功能来建立聊天系统！

47、假如 Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀开头的，如果将它们全部找出来？

使用 keys 指令可以扫出指定模式的 key 列表。对方接着追问：如果这个 redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题？

这个时候你要回答 redis 关键的一个特性：redis 的单线程的。keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

48、如果有大量的 key 需要设置同一时间过期，一般需要注意什么？

如果大量的 key 过期时间设置的过于集中，到过期的那个时间点，redis 可能会出现短暂的卡顿现象。一般需要在时间上加一个随机值，使得过期时间分散一些。

49、使用过 Redis 做异步队列么，你是怎么用的？

一般使用 list 结构作为队列，rpush 生产消息，lpop 消费消息。当 lpop 没有消息的时候，要适当 sleep 一会再重试。

如果对方追问可不可以不用 sleep 呢？

list 还有个指令叫 blpop，在没有消息的时候，它会阻塞住直到消息到来。如果对方追问能不能生产一次消费多次呢？使用 pub/sub 主题订阅者模式，可以实现1:N 的消息队列。

如果对方追问 pub/sub 有什么缺点？

在消费者下线的情况下，产生的消息会丢失，得使用专业的消息队列如 RabbitMQ 等。

如果对方追问 redis 如何实现延时队列？

我估计现在你很想把面试官一棒打死如果你手上有一根棒球棍的话，怎么问的这么详细。但是你很克制，然后神态自若的回答道：使用 sortedset，拿时间戳作为 score，消息内容作为 key 调用 zadd 来生产消息，消费者用 zrangebyscore 指令获取 N 秒之前的数据轮询进行处理。到这里，面试官暗地里已经对你竖起了大拇指。但是他不知道的是此刻你却竖起了中指，在椅子背后。

50、使用过 Redis 分布式锁么，它是什么回事

先拿 setnx 来争抢锁，抢到之后，再用 expire 给锁加一个过期时间防止锁忘记了释放。

这时候对方会告诉你说你回答得不错，然后接着问如果在 setnx 之后执行 expire 之前进程意外 crash 或者要重启维护了，那会怎么样？

这时候你要给予惊讶的反馈：唉，是喔，这个锁就永远得不到释放了。紧接着你需要抓一抓自己得脑袋，故作思考片刻，好像接下来的结果是你主动思考出来的，然后回答：我记得 set 指令有非常复杂的参数，这个应该是可以同时把 setnx 和 expire 合成一条指令来用的！对方这时会显露笑容，心里开始默念：摁，这小子还不错。

Memcached面试题

1、Memcached 是什么，有什么作用？

Memcached 是一个开源的，高性能的内存缓存软件，从名称上看 Mem 就是内存的意思，而 Cache 就是缓存的意思。Memcached 的作用：通过在事先规划好的内存空间中临时缓存数据库中的各类数据，以达到减少业务对数据库的直接高并发访问，从而达到提升数据库的访问性能，加速网站集群动态应用服务的能力。

2、memcached 服务在企业集群架构中有哪些应用场景？

一、作为数据库的前端缓存应用

a、完整缓存（易），静态缓存

例如：商品分类（京东），以及商品信息，可事先放在内存里，然后再对外提供数据访问，这种先放到内存，我们称之为预热，（先把数据存缓存中），用户访问时可以只读取 memcached 缓存，不读取数据库了。

b、执点缓存（难）

需要前端 web 程序配合，只缓存热点的数据，即缓存经常被访问的数据。先预热数据库里的基础数据，然后在动态更新，选读取缓存，如果缓存里没有对应的数据，程序再去读取数据库，然后程序把读取的新数据放入缓存存储。

特殊说明：

|| 如果碰到电商秒杀等高并发的业务，一定要事先预热，或者其它思想实现，例如：秒杀只是获取资格，而不是瞬间秒杀到手商品。

那么什么是获取资格？

- 就是在数据库中，把 0 标成 1.就有资格啦。再慢慢的去领取商品订单。因为秒杀过程太长会占用服务器资源。
- 如果数据更新，同时触发缓存更新，防止给用户过期数据。
- 对于持久化缓存存储系统，例如：redis，可以替代一部分数据库的存储，一些简单的数据业务，投票，统计，好友关注，商品分类等。
nosql= not onlysql

二、作业集群的 session 会话共享存储。

□ Memcached 服务在不同企业业务应用场景中的工作流程

- 当 web 程序需要访问后端数据库获取数据时会优先访问 Memcached 内存缓存，如果缓存中有数据就直接获取返回前端服务及用户，如果没有数据（没有命中），在由程序请求后端的数据库服务器，获取到对应的数据后，除了返回给前端服务及用户数据外，还会把数据放到 Memcached 内存中进行缓存，等待下次请求被访问，Memcache 内存始终是数据库的挡箭牌，从而大大的减轻数据库的访问压力，提高整个网站架构的响应速度，提升了用户体验。
- 当程序更新，修改或删除数据库中已有的数据时，会同时发送请求通知 Memcached 已经缓存的同一个 ID 内容的旧数据失效，从而保证 Memcache 中数据和数据库中的数据一致。
- 如果在高并发场合，除了通知 Memcached 过程的缓存失效外，还会通过相关机制，使得在用户访问新数据前，通过程序预先把更新过的数据推送到 memcache 中缓存起来，这样可以减少数据库的访问压力，提升 Memcached 中缓存命中率。
- || 数据库插件可以再写入更新数据库后，自动抛给 MC 缓存起来，自身不 Cache.

2、Memcached 服务分布式集群如何实现？

特殊说明：Memcached 集群和 web 服务集群是不一样的，所有 Memcached 的数据总和才是数据库的数据。每台 Memcached 都是部分数据。（一台 memcached 的数据，就是一部分 mysql 数据库的数据）

a、程序端实现

程序加载所有 mc 的 ip 列表，通过对 key 做 hash（一致性哈希算法）

例如：web1 (key)====>对应 A,B,C,D,E,F,G.....若干台服务器。（通过哈希算法实现）

b、负载均衡器

通过对 key 做 hash（一致性哈希算法）一致哈希算法的目的是不但保证每个对象只请求一个对应的服务器，而且当节点宕机，缓存服务器的更新重新分配比例降到最低。

3、Memcached 服务特点及工作原理是什么？

- a、完全基于内存缓存的
- b、节点之间相互独立
- c、C/S 模式架构，C 语言编写，总共 2000 行代码。
- d、异步 I/O 模型，使用 libevent 作为事件通知机制。
- e、被缓存的数据以 key/value 键值对形式存在的。
- f、全部数据存放于内存中，无持久性存储的设计，重启服务器，内存里的数据会丢失。
- g、当内存中缓存的数据容量达到启动时设定的内存值时，就自动使用 LRU 算法删除过期的缓存数据。
- h、可以对存储的数据设置过期时间，这样过期后的数据自动被清除，服务本身不会监控过期，而是在访问的时候查看 key 的时间戳，判断是否过期。
- j、memcache 会对设定的内存进行分块，再把块分组，然后再提供服务

4、简述 Memcached 内存管理机制原理？

早期的 Memcached 内存管理方式是通过 malloc 的分配的内存，使用完后通过 free 来回收内存，这种方式容易产生内存碎片，并降低操作系统对内存的管理效率。加重操作系统内存管理器的负担，最坏的情况下，会导致操作系统比 memcached 进程本身还慢，为了解决这个问题，Slab Allocation 内存分配机制就诞生了。

现在 Memcached 利用 Slab Allocation 机制来分配和管理内存。

Slab Allocation 机制原理是按照预先规定的大小，将分配给 memcached 的内存分割成特定长度的内存块（chunk），再把尺寸相同的内存块，分成组（chunks slab class），这些内存块不会释放，可以重复利用。

而且，slab allocator 还有重复使用已分配的内存的目的。也就是说，分配到的内存不会释放，而是重复利用。

Slab Allocation 的主要术语 Page 分配给 Slab 的内存空间，默认是 1MB。分配给 Slab 之后根据 slab 的大小切分成 chunk。

Chunk

用于缓存记录的内存空间。

SlabClass

特定大小的 chunk 的组

5、memcached 是怎么工作的？

Memcached 的神奇来自两阶段哈希 (two-stage hash)。Memcached 就像一个巨大的、存储了很多<key,value>对的哈希表。通过 key，可以存储或查询任意的数据。

客户端可以把数据存储在多台 memcached 上。当查询数据时，客户端首先参考节点列表计算出 key 的哈希值（阶段一哈希），进而选中一个节点；客户端将请求发送给选中的节点，然后 memcached 节点通过一个内部的哈希算法（阶段二哈希），查找真正的数据（item）

6、memcached 最大的优势是什么？

Memcached 最大的好处就是它带来了极佳的水平可扩展性，特别是在一个巨大的系统中。由于客户端自己做了一次哈希，那么我们很容易增加大量 memcached 到集群中。memcached 之间没有相互通信，因此不会增加 memcached 的负载；没有多播协议，不会网络通信量爆炸（implode）。memcached 的集群很好用。内存不够了？增加几台 memcached 吧；CPU 不够用了？再增加几台吧；有多余的内存？在增加几台吧，不要浪费了。

基于 memcached 的基本原则，可以相当轻松地构建出不同类型的缓存架构。除了这篇 FAQ，在其他地方很容易找到详细资料的。

7、memcached 和 MySQL 的 query

cache 相比，有什么优缺点？

把 memcached 引入应用中，还是需要不少工作量的。MySQL 有个使用方便的 query cache，可以自动地缓存 SQL 查询的结果，被缓存的 SQL 查询可以被反复地快速执行。Memcached 与之相比，怎么样呢？MySQL 的 query cache 是集中式的，连接到该 query cache 的 MySQL 服务器都会受益。

□ 当您修改表时，MySQL 的 query cache 会立刻被刷新（flush）。存储一个 memcached item 只需要很少的时间，但是当写操作很频繁时，MySQL 的 query cache 会经常让所有缓存数据都失效。

□ 在多核 CPU 上，MySQL 的 query cache 会遇到扩展问题（scalability issues）。在多核 CPU 上，query cache 会增加一个全局锁（global lock），由于需要刷新更多的缓存数据，速度会变得更慢。

□ 在 MySQL 的 query cache 中，我们是不能存储任意的数据的（只能是 SQL 查询结果）。而利用 memcached，我们可以搭建出各种高效的缓存。比如，可以执行多个独立的查询，构建出一个用户对象（user object），然后将用户对象缓存到 memcached 中。而 query cache 是 SQL 语句级别的，不可能做到这一点。在小的网站中，query cache 会有所帮助，但随着网站规模的增加，query cache 的弊将大于利。

□ query cache 能够利用的内存容量受到 MySQL 服务器空闲内存空间的限制。给数据库服务器增加更多的内存来缓存数据，固然是很好的。但是，有了 memcached，只要您有空闲的内存，都可以用来增加 memcached 集群的规模，然后您就可以缓存更多的数据。

8、memcached 和服务器的 local cache（比如 PHP 的 APC、mmap 文件等）相比，有什么优缺点？

首先，local cache 有许多与上面(query cache)相同的问题。local cache 能够利用的内存容量受到（单台）服务器空闲内存空间的限制。不过，local cache 有一点比 memcached 和 query cache 都要好，那就是它不但可以存储任意的数据，而且没有网络存取的延迟。

□ local cache 的数据查询更快。考虑把 highly common 的数据放在 localcache 中吧。如果每个页面都需要加载一些数量较少的数据，考虑把它们放在 local cached 吧。

□ local cache 缺少集体失效（group invalidation）的特性。在 memcached 集群中，删除或更新一个 key 会让所有的观察者觉察到。但是在 local cache 中，我们只能通知所有的服务器刷新 cache（很慢，不具扩展性），或者仅仅依赖缓存超时失效机制。

□ local cache 面临着严重的内存限制，这一点上面已经提到。

9、memcached 的 cache 机制是怎样的？

Memcached 主要的 cache 机制是 LRU（最近最少用）算法 + 超时失效。当您存数据到 memcached 中，可以指定该数据在缓存中可以呆多久。Which is forever, or some time in the future。如果 memcached 的内存不够用了，过期的 slabs 会优先被替换，接着就轮到最老的未被使用的 slabs。

10、memcached 如何实现冗余机制？

不实现！我们对这个问题感到很惊讶。Memcached 应该是应用的缓存层。它的设计本身就不带有任何冗余机制。如果一个 memcached 节点失去了所有数据，您应该可以从数据源（比如数据库）再次获取到数据。您应该特别注意，您的应用应该可以容忍节点的失效。不要写一些糟糕的查询代码，寄希望于 memcached 来保证一切！如果您担心节点失效会大大加重数据库的负担，那么您可以采取一些办法。比如您可以增加更多的节点（来减少丢失一个节点的影响），热备节点（在其他节点 down 了的时候接管 IP），等等

11、memcached 如何处理容错的？

不处理！在 memcached 节点失效的情况下，集群没有必要做任何容错处理。如果发生了节点失效，应对的措施完全取决于用户。节点失效时，下面列出几种方案供您选择：

□ 忽略它！在失效节点被恢复或替换之前，还有很多其他节点可以应对节点失效带来的影响。

□ 把失效的节点从节点列表中移除。做这个操作千万要小心！在默认情况下（余数式哈希算法），客户端添加或移除节点，会导致所有的缓存数据不可用！因为哈希参照的节点列表变化了，大部分 key 会因为哈希值的改变而被映射到（与原来）不同的节点上。

□ 启动热备节点，接管失效节点所占用的 IP。这样可以防止哈希紊乱（hashing chaos）。

□ 如果希望添加和移除节点，而不影响原先的哈希结果，可以使用一致性哈希算法（consistent hashing）。您可以百度一下一致性哈希算法。支持一致性哈希的客户端已经很成熟，而且被广泛使用。去尝试一下吧！

□ 两次哈希（reshing）。当客户端存取数据时，如果发现一个节点 down 了，就再做一次哈希（哈希算法与前一次不同），重新选择另一个节点（需要注意的是，客户端并没有把 down 的节点从节点列表中移除，下次还是有可能先哈希到它）。如果某个节点时好时坏，两次哈希的方法就有风险了，好的节点和坏的节点上都可能存在脏数据（stale data）。

12、如何将 memcached 中 item 批量导入导出？

您不应该这样做！Memcached 是一个非阻塞的服务器。任何可能导致 memcached 暂停或瞬时拒绝服务的操作都应该值得深思熟虑。向 memcached 中批量导入数据往往不是您真正想要的！想象看，如果缓存数据在导出导入之间发生了变化，您就需要处理脏数据了；

13、如果缓存数据在导出导入之间过期了，您又怎么处理这些数据呢？

因此，批量导出导入数据并不像您想象中的那么有用。不过在一个场景倒是有用。如果您有大量的从不变的数据，并且希望缓存很快热（warm）起来，批量导入缓存数据是很有帮助的。虽然这个场景并不典型，但却经常发生，因此我们会考虑在将来实现批量导出导入的功能。如果一个 memcached 节点 down 了让您很痛苦，那么您还会陷入其他很多麻烦。您的系统太脆弱了。您需要做一些优化工作。比如处理“惊群”问题（比如 memcached 节点都失效了，反复的查询让您的数据库不堪重负...这个问题在 FAQ 的其他提到过），或者优化不好的查询。记住，Memcached 并不是您逃避优化查询的借口。

14、memcached 是如何做身份验证的？

没有身份认证机制！memcached 是运行在应用下层的软件（身份验证应该是应用上层的职责）。memcached 的客户端和服务器端之所以是轻量级的，部分原因就是完全没有实现身份验证机制。这样，memcached 可以很快地创建新连接，服务器端也无需任何配置。如果您希望限制访问，您可以使用防火墙，或者让 memcached 监听 unix domain socket

15、memcached 的多线程是什么？如何使用它们？

线程就是定律（threads rule）！在 Steven Grimm 和 Facebook 的努力下，memcached 1.2 及更高版本拥有了多线程模式。多线程模式允许 memcached 能够充分利用多个 CPU，并在 CPU 之间共享所有的缓存数据。memcached 使用一种简单的锁机制来保证数据更新操作的互斥。相比在同一个物理机器上运行多个 memcached 实例，这种方式能够更有效地处理 multi gets。

如果您的系统负载并不重，也许您不需要启用多线程工作模式。如果您在运行一个拥有大规模硬件的、庞大的网站，您将会看到多线程的好处。

简单地总结一下：命令解析（memcached 在这里花了大部分时间）可以运行在多线程模式下。memcached 内部对数据的操作是基于很多全局锁的（因此这部分工作不是多线程的）。未来对多线程模式的改进，将移除大量的全局锁，提高 memcached 在负载极高的场景下的性能。

16、memcached 能接受的 key 的最大长度是多少？

key 的最大长度是 250 个字符。需要注意的是，250 是 memcached 服务器端内部的限制，如果您使用的客户端支持“key 的前缀”或类似特性，那么 key（前缀+原始 key）的最大长度是可以超过 250 个字符的。我们推荐使用使用较短的 key，因为可以节省内存和带宽。

17、memcached 对 item 的过期时间有什么限制？

过期时间最大可以达到 30 天。memcached 把传入的过期时间（时间段）解释成时间点后，一旦到了这个时间点，memcached 就把 item 置为失效状态。这是一个简单但 obscure 的机制。

18、memcached 最大能存储多大的单个 item？

1MB。如果你的数据大于 1MB，可以考虑在客户端压缩或拆分到多个 key 中。

为什么单个 item 的大小被限制在 1M byte 之内？

啊...这是一个大家经常问的问题！

简单的回答：因为内存分配器的算法就是这样的。

详细的回答：Memcached 的内存存储引擎（引擎将来可插拔...），使用 slabs 来管理内存。内存被分成大小不等的 slabs chunks（先分成大小相等的 slabs，然后每个 slab 被分成大小相等 chunks，不同 slab 的 chunk 大小是不相等的）。chunk 的大小依次从一个最小数开始，按某个因子增长，直到达到最大的可能值。

19、memcached 能够更有效地使用内存吗？

Memcache 客户端仅根据哈希算法来决定将某个 key 存储在哪个节点上，而不考虑节点的内存大小。因此，您可以在不同的节点上使用大小不等的缓存。但是一般都是这样做的：拥有较多内存的节点上可以运行多个 memcached 实例，每个实例使用的内存跟其他节点上的实例相同。

20、什么是二进制协议，我该关注吗？

关于二进制最好的信息当然是二进制协议规范：

二进制协议尝试为端提供一个更有效的、可靠的协议，减少客户端/服务器端因处理协议而产生的 CPU 时间。

根据 Facebook 的测试，解析 ASCII 协议是 memcached 中消耗 CPU 时间最多的环节。所以，我们为什么不改进 ASCII 协议呢？

21、memcached 的内存分配器是如何工作的？为什么不适用malloc/free！？为什么要使用slabs？

实际上，这是一个编译时选项。默认会使用内部的 slab 分配器。您确实应该使用内建的 slab 分配器。最早的时候，memcached 只使用 malloc/free 来管理内存。然而，这种方式不能与 OS 的内存管理以前很好地工作。反复地 malloc/free 造成了内存碎片，OS 最终花费大量的时间去查找连续的内存块来满足 malloc 的请求，而不是运行 memcached 进程。如果您不同意，当然可以使用 malloc！

只是不要在邮件列表中抱怨啊

slab 分配器就是为了解决这个问题而生的。内存被分配并划分成 chunks，一直被重复使用。因为内存被划分成大小不等的 slabs，如果 item 的大小与被选择存放它的 slab 不是很合适的话，就会浪费一些内存。Steven Grimm 正在这方面已经做出了有效的改进。

22、memcached 是原子的吗？

所有的被发送到 memcached 的单个命令是完全原子的。如果您针对同一份数据同时发送了一个 set 命令和一个 get 命令，它们不会影响对方。它们将被串行化、先后执行。即使在多线程模式，所有的命令都是原子的，除非程序有 bug：)

命令序列不是原子的。如果您通过 get 命令获取了一个 item，修改了它，然后想把它 set 回 memcached，我们不保证这个 item 没有被其他进程（process，未必是操作系统中的进程）操作过。在并发的情况下，您也可能覆盖了一个被其他进程 set 的 item。

memcached 1.2.5 以及更高版本，提供了 gets 和 cas 命令，它们可以解决上面的问题。如果您使用 gets 命令查询某个 key 的 item，memcached 会给出您返回该 item 当前值的唯一标识。如果您覆盖了这个 item 并想把它写回到 memcached 中，您可以通过 cas 命令把那个唯一标识一起发送给 memcached。如果该 item 存放在 memcached 中的唯一标识与您提供的一致，您的写操作将会成功。如果另一个进程在这期间也修改了这个 item，那么该 item 存放在 memcached 中的唯一标识将会改变，您的写操作就会失败。

23、如何实现集群中的 session 共享存储？

Session 是运行在一台服务器上的，所有的访问都会到达我们的唯一服务器上，这样我们可以根据客户端传来的 sessionID，来获取 session，或在对应 Session 不存在的情况下（session 生命周期到了/用户第一次登录），创建一个新的 Session；但是，如果我们在集群环境下，假设我们有两台服务器 A, B，用户的请求会由 Nginx 服务器进行转发（别的方案也是同理），用户登录时，Nginx 将请求转发至服务器 A 上，A 创建了新的 session，并将 SessionID 返回给客户端，用户在浏览其他页面时，客户端验证登录状态，Nginx 将请求转发至服务器 B，由于 B 上并没有对应客户端发来 sessionID 的 session，所以会重新创建一个新的 session，并且再将这个新的 sessionID 返回给客户端，这样，我们可以想象一下，用户每一次操作都有 1/2 的概率进行再次的登录，这样不仅对用户体验特别差，还会让服务器上的 session 激增，加大服务器的运行压力。

为了解决集群环境下的 session 共享问题，共有 4 种解决方案：

1. 粘性 session

粘性 session 是指 Nginx 每次都将同一用户的所有请求转发至同一台服务器上，即将用户与服务器绑定。

2. 服务器 session 复制

即每次 session 发生变化时，创建或者修改，就广播给所有集群中的服务器，使所有的服务器上的 session 相同。

3. session 共享

缓存 session，使用 redis，memcached。

4. session 持久化

将 session 存储至数据库中，像操作数据一样才做 session

24、memcached 与 redis 的区别

1、Redis 不仅仅支持简单的 k/v 类型的数据，同时还提供 list, set, zset, hash 等数据结构的存储。而 memcache 只支持简单数据类型，需要客户端自己处理复杂对象

2、Redis 支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用（PS：持久化在 rdb、aof）。

3、由于 Memcache 没有持久化机制，因此宕机所有缓存数据失效。Redis 配置为持久化，宕机重启后，将自动加载宕机时刻的数据到缓存系统中。具有更好的灾备机制。

4、Memcache 可以使用 Magent 在客户端进行一致性 hash 做分布式。Redis 支持在服务器端做分布式（PS：Twemproxy/Codis/Redis-cluster 多种分布式实现方式）

5、Memcached 的简单限制就是键（key）和 Value 的限制。最大键长为 250 个字符。可以接受的储存数据不能超过 1MB（可修改配置文件变大），因为这是典型 slab 的最大值，不适合虚拟机使用。而 Redis 的 Key 长度支持到 512k。

6、Redis 使用的是单线程模型，保证了数据按顺序提交。Memcache 需要使用 cas 保证数据一致性。CAS（Check and Set）是一个确保并发一致性的机制，属于“乐观锁”范畴；原理很简单：拿版本号，操作，对比版本号，如果一致就操作，不一致就放弃任何操作cpu 利用。由于 Redis 只使用单核，而 Memcached 可以使用多核，所以平均每一个核上 Redis 在存储小数据时比 Memcached 性能更高。而在 100k 以上的数据中，Memcached 性能要高于 Redis。

7、memcache 内存管理：使用 Slab Allocation。原理相当简单，预先分配一系列大小固定的组，然后根据数据大小选择最合适的块存储。避免了内存碎片。（缺点：不能变长，浪费了一定空间）memcached 默认情况下下一个 slab 的最大值为前一个的 1.25 倍。

8、redis 内存管理：Redis 通过定义一个数组来记录所有的内存分配情况，Redis采用的是包装的 malloc/free，相较于 Memcached 的内存管理方法来说，要简单很多。由于 malloc 首先以链表的方式搜索已管理的内存中可用的空间分配，导致内存碎片比较多

MongoDB面试题

1、mongodb是什么？

MongoDB 是由 C++语言编写的，是一个基于分布式文件存储的开源数据库系统。在高负载的情况下，添加更多的节点，可以保证服务器性能。MongoDB 旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

```
{  
    name: "sue",  
    age: 26,  
    status: "A",  
    groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

2、mongodb有哪些特点？

- 1.MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。
- 2.你可以在 MongoDB 记录中设置任何属性的索引(如： FirstName="Sameer",Address="8 Gandhi Road")来实现更快的排序。
- 3.你可以通过本地或者网络创建数据镜像，这使得 MongoDB 有更强的扩展性。
- 4.如果负载的增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上这就是所谓的分片。
- 5.Mongo 支持丰富的查询表达式。查询指令使用 JSON 形式的标记，可轻易查询文档中内嵌的对象及数组。
- 6.MongoDb 使用 update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段。
- 7.Mongodb 中的 Map/reduce 主要是用来对数据进行批量处理和聚合操作。
- 8.Map 和 Reduce。 Map 函数调用 emit(key,value)遍历集合中所有的记录，将 key 与 value 传给 Reduce 函数进行处理。
- 9.Map 函数和 Reduce 函数是使用 Javascript 编写的，并可以通过 db.runCommand 或 mapreduce 命令来执行 MapReduce 操作。
10. GridFS 是 MongoDB 中的一个内置功能，可以用于存放大量小文件。

11. MongoDB 允许在服务端执行脚本，可以用 Javascript 编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。

3、你说的NoSQL数据库是什么意思?NoSQL与RDBMS直接有什么区别?为什么要使用和不使用NoSQL数据库?说一说NoSQL数据库的几个优点?

NoSQL是非关系型数据库，NoSQL = Not Only SQL。

关系型数据库采用的结构化的数据，NoSQL采用的是键值对的方式存储数据。

在处理非结构化/半结构化的大数据时；在水平方向上进行扩展时；随时应对动态增加的数据项时可以优先考虑使用NoSQL数据库。

在考虑数据库的成熟度；支持；分析和商业智能；管理及专业性等问题时，应优先考虑关系型数据库。

4、NoSQL数据库有哪些类型？

NoSQL数据库的类型

例如：MongoDB, Cassandra, CouchDB, Hypertable, Redis, Riak, Neo4j, HBASE, Couchbase, MemcacheDB, RevenDB and Voldemort are the examples of NoSQL databases. [详细阅读](#)。

5、MySQL与MongoDB之间最基本的差别是什么？

MySQL和MongoDB两者都是免费开源的数据库。MySQL和MongoDB有许多基本差别包括数据的表示(data representation), 查询, 关系, 事务, schema的设计和定义, 标准化(normalization), 速度和性能。

通过比较MySQL和MongoDB, 实际上我们是在比较关系型和非关系型数据库, 即数据存储结构不同。 [详细阅读](#)

6、你怎么比较MongoDB、CouchDB及CouchBase?

MongoDB和CouchDB都是面向文档的数据库。MongoDB和CouchDB都是开源NoSQL数据库的最典型代表。除了都以文档形式存储外它们没有其他的共同点。MongoDB和CouchDB在数据模型实现、接口、对象存储以及复制方法等方面有很多不同。

细节可以参见下面的链接：

[MongoDB vs CouchDB](#)

[CouchDB vs CouchBase](#)

7、MongoDB成为最好NoSQL数据库的原因是什么？

以下特点使得MongoDB成为最好的NoSQL数据库：

- 面向文件的
- 高性能
- 高可用性
- 易扩展性
- 丰富的查询语言

6.32位系统上有什么细微差别？

journaling会激活额外的内存映射文件。这将进一步抑制32位版本上的数据库大小。因此，现在journaling在32位系统上默认是禁用的。

8、journal回放在条目(entry)不完整时(比如恰巧有一个中途故障了)会遇到问题吗？

每个journal (group)的写操作都是一致的，除非它是完整的否则在恢复过程中它不会回放。

9、分析器在MongoDB中的作用是什么？

MongoDB中包括了一个可以显示数据库中每个操作性能特点的数据库分析器。通过这个分析器你可以找到比预期慢的查询(或写操作);利用这一信息，比如，可以确定是否需要添加索引。

10、名字空间(namespace)是什么？

MongoDB存储BSON对象在从集(collection)中。数据库名字和从集名字以句点连结起来叫做名字空间(namespace)。

11、如果用户移除对象的属性，该属性是否从存储层中删除？

是的，用户移除属性然后对象会重新保存(re-save())。

12、能否使用日志特征进行安全备份？

是的。

13、允许空值null吗？

对于对象成员而言，是的。然而用户不能够添加空值(null)到数据库从集(collection)因为空值不是对象。然而用户能够添加空对象{}。

14、更新操作立刻fsync到磁盘？

不会，磁盘写操作默认是延迟执行的。写操作可能在两三秒(默认在60秒内)后到达磁盘。例如，如果一秒内数据库收到一千个对一个对象递增的操作，仅刷新磁盘一次。(注意，尽管fsync选项在命令行和经过getLastError_old是有效的)(译者：也许是坑人的面试题??)。

15、如何执行事务/加锁?

MongoDB没有使用传统的锁或者复杂的带回滚的事务，因为它设计的宗旨是轻量，快速以及可预计的高性能。可以把它类比成MySQL MyISAM的自动提交模式。通过精简对事务的支持，性能得到了提升，特别是在一个可能会穿过多个服务器的系统里。

16、为什么我的数据文件如此庞大?

MongoDB会积极的预分配预留空间来防止文件系统碎片。

17、启用备份故障恢复需要多久?

从备份数据库声明主数据库宕机到选出一个备份数据库作为新的主数据库将花费10到30秒时间。这期间在主数据库上的操作将会失败~包括写入和强一致性读取(strong consistent read)操作。然而，你还能在第二数据库上执行最终一致性查询(eventually consistent query)(在slaveOk模式下)，即使在这段时间里。

18、什么是master或primary?

它是当前备份集群(replica set)中负责处理所有写入操作的主要节点/成员。在一个备份集群中，当失效备援(failover)事件发生时，一个另外的成员会变成primary。

19、什么是secondary或slave?

Secondary从当前的primary上复制相应的操作。它是通过跟踪复制oplog(local.oplog.rs)做到的。

20、我必须调用getLastError来确保写操作生效了么?

不用。不管你有没有调用getLastError(又叫"Safe Mode")服务器做的操作都一样。调用getLastError只是为了确认写操作成功提交了。当然，你经常想得到确认，但是写操作的安全性和是否生效不是由这个决定的。

21、我应该启动一个集群分片(sharded)还是一个非集群分片的 MongoDB 环境?

为开发便捷起见，我们建议以非集群分片(unsharded)方式开始一个 MongoDB 环境，除非一台服务器不足以存放你的初始数据集。从非集群分片升级到集群分片(sharding)是无缝的，所以在你的数据集还不是很大的时候没必要考虑集群分片(sharding)。

22、分片(sharding)和复制(replication)是怎样工作的?

每一个分片(shard)是一个分区数据的逻辑集合。分片可能由单一服务器或者集群组成，我们推荐为每一个分片(shard)使用集群。

23、数据在什么时候才会扩展到多个分片(shard)里?

MongoDB 分片是基于区域(range)的。所以一个集合(collection)中的所有的对象都被存放到一个块(chunk)中。只有当存在多余一个块的时候，才会有多个分片获取数据的选项。现在，每个默认块的大小是 64Mb，所以你需要至少 64 Mb 空间才可以实施一个迁移。

24、当我试图更新一个正在被迁移的块(chunk)上的文档时会发生什么?

更新操作会立即发生在旧的分片(shard)上，然后更改才会在所有权转移(ownership transfers)前复制到新的分片上。

25、如果在一个分片(shard)停止或者很慢的时候，我发起一个查询会怎样?

如果一个分片(shard)停止了，除非查询设置了"Partial"选项，否则查询会返回一个错误。如果一个分片(shard)响应很慢，MongoDB则会等待它的响应。

26、我可以把moveChunk目录里的旧文件删除吗?

没问题，这些文件是在分片(shard)进行均衡操作(balancing)的时候产生的临时文件。一旦这些操作已经完成，相关的临时文件也应该被删除掉。但目前清理工作是需要手动的，所以请小心地考虑再释放这些文件的空间。

27、我怎么查看 Mongo 正在使用的链接?

```
db._adminCommand("connPoolStats");
```

28、如果块移动操作(moveChunk)失败了，我需要手动清除部分转移的文档吗?

不需要，移动操作是一致(consistent)并且是确定性的(deterministic);一次失败后，移动操作会不断重试;当完成后，数据只会出现在新的分片里(shard)。

29、如果我在使用复制技术(replication)，可以一部分使用日志(journaling)而其他部分则不使用吗？

可以。

30、当更新一个正在被迁移的块 (Chunk) 上的文档时会发生什么？

更新操作会立即发生在旧的块 (Chunk) 上，然后更改才会在所有权转移前复制到新的分片上。

31、MongoDB在A:{B,C}上建立索引，查询A:{B,C}和A:{C,B}都会使用索引吗？

不会，只会在A:{B,C}上使用索引。

32、如果一个分片 (Shard) 停止或很慢的时候，发起一个查询会怎样？

如果一个分片停止了，除非查询设置了“Partial”选项，否则查询会返回一个错误。如果一个分片响应很慢，MongoDB会等待它的响应。

33、MongoDB支持存储过程吗？如果支持的话，怎么用？

MongoDB支持存储过程，它是javascript写的，保存在db.system.js表中。

34、如何理解MongoDB中的GridFS机制，MongoDB为何使用GridFS来存储文件？

GridFS是一种将大型文件存储在MongoDB中的文件规范。使用GridFS可以将大文件分隔成多个小文档存放，这样我们能够有效的保存大文档，而且解决了BSON对象有限制的问题。

35、什么是NoSQL数据库？NoSQL和RDBMS有什么区别？在哪些情况下使用和不使用NoSQL数据库？

NoSQL是非关系型数据库，NoSQL = Not Only SQL。

关系型数据库采用的结构化的数据，NoSQL采用的是键值对的方式存储数据。

在处理非结构化/半结构化的大数据时；在水平方向上进行扩展时；随时应对动态增加的数据项时可以优先考虑使用NoSQL数据库。

在考虑数据库的成熟度；支持；分析和商业智能；管理及专业性等问题时，应优先考虑关系型数据库。

36、MongoDB支持存储过程吗？如果支持的话，怎么用？

MongoDB支持存储过程，它是javascript写的，保存在db.system.js表中。

37、如何理解MongoDB中的GridFS机制，MongoDB为何使用GridFS来存储文件？

GridFS是一种将大型文件存储在MongoDB中的文件规范。使用GridFS可以将大文件分隔成多个小文档存放，这样我们能够有效的保存大文档，而且解决了BSON对象有限制的问题。

38、为什么MongoDB的数据文件很大？

MongoDB采用的预分配空间的方式来防止文件碎片。

39、当更新一个正在被迁移的块 (Chunk) 上的文档时会发生什么？

更新操作会立即发生在旧的块 (Chunk) 上，然后更改才会在所有权转移前复制到新的分片上。

40、MongoDB在A:{B,C}上建立索引，查询A:{B,C}和A:{C,B}都会使用索引吗？

不会，只会在A:{B,C}上使用索引。

41、如果一个分片 (Shard) 停止或很慢的时候，发起一个查询会怎样？