实战

2017-12-美团（工作1年）

1. String和StringBuilder区别？

2. HashMap存储方式，如何根据Key查找数据？

3. Hashtable和ConcurrentHashMap区别？

4. 多线程

5. 线程池

6. synchronized和ReentrantLock功能上的区别？

7. JVM内存分块？

8. JVM如何GC？

9. B+树和平衡二叉树区别？

10. 事务的访问控制？（作者注：此题可具体到Spring如何对事务进行管理）

11. MySQL的存储引擎？

12. Java中什么叫不可变对象？（作者注：此题可延伸至不可变引用和不可变对象指的是什么）

13. 索引如何提高查询效率？

2018-3-京东（工作1.5年）

1. 读过哪些JDK源码？

2. ConcurrentHashMap的优势在哪里？

3. ConcurrentHashMap为什么比Hashtable的性能更高？

4. ConcurrentHashMap的“读”会不会加锁？

5. ConcurrentHashMap的key是否可以为null？

6. 简单说一下线程池的原理？

7. 线程池中有哪些饱和策略？

8. 如果在线程池中遇到任务来的太多太快，线程池来不及处理任务队列不断增多，如何解决？

9. 在工作中用到过哪些设计模式？

10. Spring中在bean加载完成后如何立即执行一个方法，除了在配置文件中加入“init-method”属性？

11. Spring AOP在项目中做了哪些东西？

12. Spring是如何对事务进行管理的？

13. MySQL有做过哪些性能调优？

14. 为什么在读多写少的情况下，可以切换使用MySQL的MyISAM引擎，它和InnoDB有什么区别？

15. 有没有用过分布式锁？

16. 有没有了解分布式锁的实现？

17. 在工作中有没有JVM线上排查经验，OOM的场景有哪些？

2018-3-菜鸟（工作1.5年）

1. Spring AOP原理、在内部是如何设计的、在内部是如何实现的？

2. 如何实现一个IoC？

3. BeanFactory和FactoryBean有什么区别？

4. IoC有什么好处？直接new一个对象会带来什么问题？

5. 什么样的Bean适合单例模式？

6. Java的内存模型？

7. Java是如何GC的？

8. JVM在管理内存这块在JDK6、7、8有什么区别 ？（作者注：此处应该是问的GC回收器有什么不同：串行收集器、CMS、G1）

9. Full GC会带来什么问题？Minor GC和Full GC有什么区别？

10. 频繁的Full GC会带来什么问题？

11. 有没有线上故障排查经验？怎么解决的？引起了什么现象

12. 如何解决虚拟机栈的内存溢出？（作者注：引起虚拟机栈OOM的原因，并不是虚拟机栈StackOverFlow的原因）

13. 直接new一个线程会带来什么问题？

14. Java提供了哪几种线程池？

15. 使用线程池需要注意什么问题？

16. 是否了解线程泄露，如何解决？

17. CountDownLatch、CyclicBarrier有什么区别？

18. HashMap怎么实现的？ （作者注：此处可分为JDK7和JDK8回答）

19. 介绍下红黑树算法。

20. HashMap是线程安全的吗？

21. 什么是线程安全？

22. 线程安全从JVM指令集讲，是因为什么机制导致这个问题的？

23. HashMap和ConcurrentHashMap有什么区别？

24. ConcurrentHashMap内部使用什么机制保证了它线程安全？

25. Java中有几种锁机制？ （作者注：这个问题考查偏向锁、轻量级锁、重量级锁、自旋锁）

26. synchronized与其他的锁比较有什么优缺点？

27. 公平锁与非公平锁？

27. 公平锁与非公平锁？

28. 什么叫做可重入锁？

29. 有没有数据库优化经验？数据库的性能问题。

30. 一张表有1000千万数据，从哪些方面来设计提高它的搜索效率？

31. 索引是不是越多越好？

32. 设计索引有哪些原则？

33. Memcached和Redis有什么区别？

34. 一致性哈希这个概念有没有了解？

大纲

Java基础

1. String、StringBuilder 和StringBuffer区别

String是字符串常量，对象不可修改，对它的修改实际上是在常量池中重新创建一个字符串常量。

StringBuffer可在字符串对象上对自身进行修改，不必重新创建对象，是线程安全的类。

StringBuilder同StringBuffer，是线程不安全的类。

2. HashMap、LinkedHashMap、TreeMap、Hashtable、ConcurrentHashMap这几个的区别以及特点

都是Map的实现类，存储key-value形式的键值对。

HashMap是插入无序的，key值不重复，key允许为null值，线程不安全。

LinkedHashMap是插入有序的，并且可以设置为按访问有序排列，key值不重复，key允许为null值，线程不安全。

TreeMap是按字典序升序排序的，key值不重复，key不允许为null，线程不安全。

Hashtable是线程安全的HashMap类，其余特点与HashMap相同。

HashMap底层是一个散列表+链表的实现，从JDK8加入了红黑树的结构，当散列表的数据冲突形成链表，并且链表上的数据达到阈值8个时，就会将链表结构转换为红黑树结构。它的put过程是，计算key的hash值，与散列表的大小-1做与运算，计算出key值所在散列表的下标i，如果这个位置没有Entry对象，则直接放到这个位置上。如果在该位置上产生了冲突，则遍历整个链表，判断链表上是否有equals相等的key，有则直接替换。没有则将插入的key-value放置的链表的头部。

它的扩容过程是，当散列表中的数据到达阈值时，HashMap会进行扩容操作，扩容后的大小是之前散列表大小的两倍。创建新的Entry数组后则会将以前的Entry数组转移到新的散列表中，转移的方法是遍历散列表，重新计算key的数组下标，如果产生冲突后，会通过头插法的方式插入到新的散列表中。

LinkedHashMap其内部有两个大的数据结构，一个是和HashMap一样的数据结构数组+链表它本身就继承自HashMap，保证了Map的随机存取特性；另一个结构维护了一个双向链表，插入一个key-value除了放置到散列表上，还会添加到双向链表的尾部，保证了插入有序。LinkedHashMap因为它不仅能保证插入有序，还能访问有序，最近访问的会放置到链表尾部，这可以实现一个简单的LRU缓存，通过实现removeEldestEntry方法。

TreeMap内部维护一个红黑树，能保证按字典序升序排列。不能插入null值。

Hashtable是线程安全的HashMap类，但由于它是对普通方法上加锁，会将整个散列表锁住，以至于效率低下不怎么使用。get和put方法都会加锁，get方法加锁是因为resize扩容过后，Entry可能并不在以前的位置，所以需要对所有方法都加锁保证线程安全。

ConcurrentHashMap同样是线程安全的HashMap类，不同的是它不再锁住整张表，而是使用分段锁的概念，将一个散列表分成几个段，当插入的元素在不同段的时候不必加锁也能实现线程安全，只有在同一个段上的时候才会加锁保证线程安全。

3. HashSet、LinkedHashSet、TreeSet这几个的区别以及特点

HashSet的值不能重复且乱序排列，可以存储null值。它的内部维护了一个HashMap，添加的值作为HashMap的key进行插入，所以保证了它的这些特性。

LinkedHashSet保证了插入有序，它继承了HashSet类，通过调用HashSet的一个构造方法，创建一个LinkedHashMap对象以此通过LinkedHashMap来保证插入有序。

TreeSet保证了字典有序排列，同TreeMap一样，继承了NavigatorMap类，维护的是一个红黑树保证有序。

4. ArrayList、LinkedList、Vector集合的区别以及特点

ArrayList底层使用数组实现，按插入排序，数据可重复，线程不安全。在JDK7中调用无参构造方法时不会为它分配大小，JDK6构造时即会分配大小。JDK7默认大小是10，之后的扩容会按1.5倍大小，右移一位（>>），通过调用Arrays.copyOf进行数据的转移。

LinkedList底层使用链表实现，在JDK7是一个双向链表。同样按插入顺序，数据可重复，线程不安全，没有扩容问题。

Vector可看做是一个线程安全的ArrayList，它通过在方法上加入synchronized关键字来保证线程安全，效率不过。可使用Collections.synchronizedList创建线程安全的List。

在concurrent并发包中有ConcurrentHashMap线程安全的HashMap，却没有线程安全的List。这是由于ConcurrentHashMap通过分段锁的技术，既保证了线程安全，同时保证了编发性能。而List很难保证其并发性能，只有CopyOnWriteArrayList保证了只读的并发性能，而对于其修改操作同样需要锁住其整个List。

5.Java中的IO，它们之间的区别

Java中的IO有阻塞式IO（BIO），非阻塞式IO（NIO），以及异步IO（AIO）。

同步阻塞式的IO客户端与服务器端的关系是一应一答的关系，其根本特性是一件事一件事的来，例如当一个线程在执行过程中依赖需要等待的资源时，这个时候线程会处于阻塞状态。大量的客户端请求到来时，采用同步阻塞式的IO会造成性能瓶颈。可以采用伪异步的方式，在服务器端接收到客户端请求时创建线程处理，此时会造成创建线程过多的问题。

NIO解决BIO大并发性能低的问题，使用多路复用机制。对于BIO中的流，NIO对应的是Channel通道，在一个完整NIO的模型中，一共有三个部分组成：Channel、Buffer、Selector。一个客户端的请求数据对应一个Channel，无论是读还是写，都需要Channel写入Buffer，或者Buffer写入Channel。Selector就是NIO实现多路复用的基础。NIO中Selector可以使用单个线程对多个Channel进行处理，这也就是相对于BIO中伪异步方式带来的好处，一个线程就能在服务器端管理多个客户端请求，并且能达到很好的并发性能。

在NIO中使用Selector会不断的轮询各个Channel来确定是否有数据可读，而在AIO中则是数据准备好过主动通知数据使用者。它利用了操作系统底层的API支持，Unix系统下是epoll IO，在Windows下则是IOCP模型。

*Reactor和Proactor模式，同步采用Reactor，异步采用proactor

6.final, finally, finalize 的区别？

final：不可变的对象引用，常量，一旦定义不可修改。

finally：异常捕获中，无论如何都会执行，常用于处理文件流的关闭。

finalize：GC前会调用此方法，不建议使用，它的调用依赖于何时进行GC，而何时GC是无法确定的。

7. HashMap的初始化长度？负载因子是多少是什么？能容纳的最大个数？扩容机制？散列表长度的设计有什么特殊之处吗？为什么这么设计？如何确定索引位置？

初始化长度length=16，负载因子loadFactor=0.75，表示Map能容纳的数量，threshold

能容纳的最大个数是长度*负载因子=16 * 0.75 = 12。

在初始状态下，如果Map中的元素（Node）个数超过12个，此时便会进行扩容（resize），扩容后的容量是之前的两倍。

由于HashMap的底层实现是数组，而数组无法自动扩容，此时只能采取重新定义一个新的数组，代替原来的数组。JDK8由于引入了红黑树，其扩容机制较为复杂，JDk7的扩容机制如下：

创建双倍大小的数组；

遍历散列表中的元素；

使用头插法插入新的数组。

JDK8做了优化，不再重新计算hash，JDK1.7中rehash的时候，旧链表迁移新链表的时候，如果在新表的数组索引位置相同，则链表元素会倒置，但是从上图可以看出，JDK1.8不会倒置

此处关于HashMap的扩容机制还需深入理解

散列表的长度必须是2的n次方，本来如果设计为素数导致的冲突要小于2的n次方这种设计，例如Hashtable的初始长度，但它扩容后不能保证还是素数。2的n次方设计主要是为了在取模和扩容时做优化，同时为了减少冲突，HashMap定位散列表索引位置时，也加入了高位参与运算的过程。

取key的hashCode值、高位运算、取模运算。

8. 简单介绍一下java中的泛型，泛型擦除以及相关的概念

泛型是Java SE 1.5的新特性，泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口、泛型方法。 Java语言引入泛型的好处是安全简单。

在Java SE 1.5之前，没有泛型的情况的下，通过对类型Object的引用来实现参数的“任意化”，“任意化”带来的缺点是要做显式的强制类型转换，而这种转换是要求开发者对实际参数类型可以预知的情况下进行的。对于强制类型转换错误的情况，编译器可能不提示错误，在运行的时候才出现异常，这是一个安全隐患。

泛型的好处是在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，提高代码的重用率。

1. 泛型的类型参数只能是类类型（包括自定义类），不能是简单类型。
2. 同一种泛型可以对应多个版本（因为参数类型是不确定的），不同版本的泛型类实例是不兼容的。
3. 泛型的类型参数可以有多个。
4. 泛型的参数类型可以使用extends语句，例如。习惯上称为“有界类型”。
5. 泛型的参数类型还可以是通配符类型。例如Class<?> classType = Class.forName("java.lang.String");

9. 类加载为什么要使用双亲委派模式，有没有什么场景是打破了这个模式？

在Java中判断两个类是否是同一个类，不仅仅是类的全限定名称相同，还需要加载它们的类加载器相同。使用双亲委派模式，加载Object类时会始终使用启动类加载器进行加载，而不会使用自定义类加载器，如果不使用双亲委派模式的话程序会混乱不堪。

JNDI服务打破了双亲委派模式。按照双亲委派模式，启动类加载器会加载JNDI，此时启动类加载器找到无法对各厂商具体实现，引入了ThreadContextClassLoader，父加载器会请求子加载器对其进行加载。

10. 类的实例化顺序？

大致的顺序是，先静态方法、再构造方法，先父类后子类。

父类静态成员和静态初始化块，按代码顺序；

子类静态成员和静态初始化块，按代码顺序；

父类实例成员和实例初始化块，按代码顺序；

父类构造方法；

子类实例成员和实例初始化块，按代码顺序；

子类构造方法。

Java多线程并发

1. synchronized的实现原理以及锁优化？

Java中每个对象有一个监视器（monitor），synchronized关键字就是使用指令“monitorenter”、“monitorexit”获取对象监视器锁和释放监视器锁。

关于synchronized的锁优化，需理解偏向锁、轻量级锁、重量级锁

2. synchronized修饰代码块、普通方法、静态方法有什么区别。

修饰代码块获取的是指定的对象监视器。

修饰普通代码块并未显示使用monitorenter指令，而是通过标志位ACC_SYNCHRONIZED标志位来判断是否由synchronized修饰，获取的是该实例对象的监视器。

修饰静态方法同普通方法，不同的是获取的是这个类的监视器。

3. volatile 关键字及其实现原理？

1. 内存可见性；
2. 不可重排序。

由于Java内存模型的原因，Java中提供了主内存和线程的工作内存，针对普通变量的赋值和取值是和线程工作内存进行交互，而volatile修饰后则是会从主内存中刷新，保证工作内存和主内存数据一致。

read、load、use；assign、store、save。

重排序指的是JVM在对其进行优化时，会在不影响逻辑结果的情况下会对执行顺序有所改变。加入volatile关键字修饰后，在其指令中会形成内存屏障，在该关键字之前的操作不允许重排序。

4. Java 的信号灯

Semaphore。使用信号灯Semaphore可控制线程访问资源的个数，通过acquire获取一个许可，没有就等待，通过release释放一个许可。内部通过AQS（AbstractQueuedSynchronizer）实现，控制数赋值给AQS中的state值，成功获得许可则状态-1，如果为0，线程此时进入AQS的等待队列中，创建Semaphore时可指定它的公平性，默认非公平。

5. 怎么实现所有线程在等待某个事件的发生才会去执行？

使用CountDownLatch等待某一个指定的事件发生后，才让多个等待的线程继续执行。

6. CountDownLatch和CyclicBarrier的用法，以及相互之间的差别？

CountDownLatch是闭锁，等待事件发生的同步工具类。采用减计数方式，为0时释放所有线程。计数器设为1，当某个事件未到来时，线程await阻塞，某个事件发生后调用countDown方法计数器-1=0，此时线程被唤醒继续执行。计数器不可重置。

CyclicBarrier是栅栏，采用加计数方式。调用await计数器+1，当计数达到某个设置值时，释放所有等待的线程，用于等待线程的全部执行。计数器可置0。

7. synchronized 和 lock 有什么区别？

synchronized是一个关键字，jvm实现；lock是一个类。

synchronized不需要显示释放锁，退出同步代码块或者同步方法、抛出异常时会释放锁；lock需要显示释放。

synchronized是阻塞式的霍曲锁；lock可以采取非阻塞方式。

synchronized可重入、不可中断、非公平；lock可重入、可中断、可公平。

8. CAS？CAS 有什么缺陷，如何解决？

compare and swap。比较和替换，使用一个期望值和当前值进行比较，如果当前变量的值和我们期望的值相等，就使用一个新值替换当前变量的值。 缺陷，只对值有效，而不会判断当前值是否经历了几次修改，加入版本控制编号（此处不理解可参考Mysql的乐观锁实现）

9. 介绍ConcurrenHashMap？

线程安全的HashMap，与Hashtable在整个散列表上加锁不同的是，ConcurrentHashMap采用的是分段锁。将整个散列表分为几个部分，在不同部分加锁，称之为分段锁，key散列到不同的段可以并行存储互不影响，只有散列到同一个段上的时候才会加锁互斥。段的个数会根据设置的concurrentLevel来确定，concurrentLevel默认=16，段的个数会大于或等于concurrentLevel最小的2次幂。

put的主要逻辑也是：1.定位segment并确保segment已经初始化；2.调用segment的put方法。

10. 线程池的种类，区别和使用场景？

JDK为我们提供了4种线程池：

newFixedThreadPool：固定线程池，根据传入的参数，线程池的线程数量是固定的；

corePoolSize = maximumPoolSize = nThread。采用无界阻塞队列。LinkedBlockingQueue。

任务提交到线程池，使用池中的线程，如果池中满了的话则进入阻塞队列。

适用于执行长期的任务。

newSingleThreadPool：只包含一个线程的线程池；

corePoolSize = maximumPoolSize = 1。

适用于一个任务一个任务执行的场景。

newCachedTheradPool：可创建无限量的线程池，如果提交任务的速度大于任务处理的速度，则会无限量的创建线程；

其corePoolSize核心线程池为0，任务直接提到到同步队列，执行任务时会从maximumPoolSize线程池中寻找可用的线程，如果没有则创建线程。如果提交任务的速度大于任务处理的速度，则会无限量的创建线程；如果有线程的空闲时间超过指定大小，则线程会被销毁。

适用于执行短期异步的小程序，或则负载较轻的小程序。执行时间长的任务会不断创建线程。

newScheduledThreadPool：可以定时执行任务的线程池。

corePoolSize为传递进来的参数，maximumPoolSize为Integer.MAX_VALUE。dWorkQueue() 一个按超时时间升序排序的队列。

适用于周期性执行任务的场景

10. 分析线程池的实现原理和线程的调度过程？

1. 当线程池小于corePoolSize时，新提交任务将创建一个新线程执行任务，即使此时线程池中存在空闲线程。

2. 当线程池达到corePoolSize时，新提交任务将被放入workQueue中，等待线程池中任务调度执行

3. 当workQueue已满，且maximumPoolSize>corePoolSize时，新提交任务会创建新线程执行任务

4. 当提交任务数超过maximumPoolSize时，新提交任务由RejectedExecutionHandler处理

5. 当线程池中超过corePoolSize线程，空闲时间达到keepAliveTime时，关闭空闲线程

6. 当设置allowCoreThreadTimeOut(true)时，线程池中corePoolSize线程空闲时间达到keepAliveTime也将关闭

11. 线程池如何调优，最大数目如何确认？

根据业务场景，限制最大线程数以及最小线程数，包括工作的队列的选择。最大数目最好不超过cpu核心数。

12. ThreadLocal原理，用的时候需要注意什么？

ThreadLocal本地变量，各线程直接并不需要共享此变量。

在每个线程Thread内部有一个ThreadLocal.ThreadLocalMap类型的成员变量threadLocals并未实现Map接口但也是初始化一个16个大小的Entry数组，这个threadLocals就是用来存储实际的变量副本的，键值为当前ThreadLocal变量，value为变量副本（即T类型的变量）。

ThreadLocal可能导致内存泄漏。key被保存到了WeakReference对象中。使用线程池时，线程可能并不会被销毁，如果创建ThreadLocal的线程一直持续运行，那么这个Entry对象中的value就有可能一直得不到回收，发生内存泄露。

13. Condition接口及其实现原理

Condition接口提供了类似Object的监视器方法，与Lock配合可以实现等待/通知模式。Condition定义了等待/通知两种类型的方法，当前线程调用这些方法时，需要提前获取到Condition对象关联的锁。Condition对象是由Lock对象（调用Lock对象的newCondition()方法）创建出来的，换句话说，Condition是依赖Lock对象的。一般都会将Condition对象作为成员变量。当调用await()方法后，当前线程会释放锁并在此等待，而其他线程调用Condition对象的signal()方法，通知当前线程后，当前线程才从await()方法返回，并且在返回前已经获取了锁。

ConditionObject是同步器AbstractQueuedSynchronizer的内部类，因为Condition的操作需要获取相关联的锁，所以作为同步器的内部类也较为合理。每个Condition对象都包含着一个队列，该队列是Condition对象实现等待/通知功能的关键。

14. Fork/Join框架的理解

用于并行执行任务的框架， 是一个把大任务分割成若干个小任务，最终汇总每个小任务结果后得到大任务结果的框架。

第一步分割任务。首先我们需要有一个fork类来把大任务分割成子任务，有可能子任务还是很大，所以还需要不停的分割，直到分割出的子任务足够小。

第二步执行任务并合并结果。分割的子任务分别放在双端队列里，然后几个启动线程分别从双端队列里获取任务执行。子任务执行完的结果都统一放在一个队列里，启动一个线程从队列里拿数据，然后合并这些数据。

Fork/Join使用两个类来完成以上两件事情：

RecursiveAction：用于没有返回结果的任务。

RecursiveTask ：用于有返回结果的任务。

ForkJoinPool ：ForkJoinTask需要通过ForkJoinPool来执行，任务分割出的子任务会添加到当前工作线程所维护的双端队列中，进入队列的头部。当一个工作线程的队列里暂时没有任务时，它会随机从其他工作线程的队列的尾部获取一个任务

时没有任务时，它会随机从其他工作线程的队列的尾部获取一个任务。

15. 阻塞队列以及各个阻塞队列的特性

ArrayBlockingQueue ：有界阻塞队列，一旦指定了队列的长度，则队列的大小不能被改变

LinkedBlockingQueue ：可以通过构造方法设置capacity来使得阻塞队列是有界的，也可以不设置，则为无界队列

JVM

1. 详细JVM内存模型？

JVM内存模型一共分为：程序计数器、本地方法栈、虚拟机栈、方法区、堆。

2. 什么情况会出现内存溢出？内存泄露？

内存溢出：

堆内存溢出，会抛出heap space oom，此时堆内存不足以分配新的对象实例；

方法区内存溢出，会抛出permgen space oom，此时class对象太多或者jsp太多，或者字符串常量过多，方法区不足以分配新的内存空间；

虚拟机栈内存溢出，单线程下抛出stackoverflow，多线程抛出oom:unable create native thread，程序中线程过多，可创建线程数小，减小虚拟机栈的大小。

内存泄露：

未能及时GC对象，这部分对象既不能回收又不能使用。长生命周期的对象持有短生命周期对象的引用。尽量使用局部变量，或者在对象使用完过后赋值null。

3. Java线程栈

每个线程有一个线程栈，包括主线程。创建一个新的线程，即会产生一个新的线程栈。具有多个线程栈时，会并行执行。

4. JVM 年轻代到年老代的晋升过程的判断条件是什么？

对象分配在年轻代上，每经历一次minor gc其年龄就会+1，达到一定的阈值时会晋升到老年代。可以通过MaxTenuringThreshold设置阈值。

5. FullGC频繁，怎么排查？

JVM参数加上XX:HeapDumpBeforeFullGC，也就是在FullGC前记录堆内存的情况。dump文件显示程序中有大量的大对象，原因是程序启动后将数据库中的数据全部加载进了内存中，导致出现堆的内存较小无法继续分配持续FullGC，并出现了停止响应等问题。

6. JVM垃圾回收机制，何时触发MinorGC等操作？

分代垃圾回收机制：不同的对象生命周期不同，针对不同的对象采用不同的GC方式。

一共分为：年轻代、老年代、永久代。

年轻代：新创建的对象会分配在年轻代。

老年代：经过N次年轻代的minor gc仍然存活的对象会进入老年代；或者大对象的创建会直接进入老年代。

永久代：字符串常量、静态文件。

新生代的gc称为minor gc，只会在新生代上产生gc；老年代的gc称为full gc会发生在整个堆上。

触发条件：堆内存不足；程序较为空闲时会触发gc线程执行。

7. JVM 中一次完整的 GC 流程（从 ygc 到 fgc）是怎样的

对象优先在年轻代中分配，若没有足够的空间则进行minor gc，大对象直接进入老年代，长期存活的对象也直接进入老年代。对象如果在新生代中的一次minor gc后存活，则其年龄+1，如果到了阈值15则进入老年代，阈值可通过MaxTenuringThreshold设置。

各种回收器，各自优缺点，重点CMS、G1

串行收集器：顾名思义会使用串行的方式进行gc，会暂定所有线程，单线程。

并行收集器：同上也会暂停所有线程，不过会使用多线程的方式进行gc。

CMS：并发标记清除。发生在老年代。一共经历4个过程：初始标记、并发标记、重新标记、并发清除，在初始标记和重新标记阶段会暂停线程，不过时间较短，采用的是标记-清除算法。

G1：jdk7默认的gc收集器。发生在新生代以及老年代。

7. 各种回收算法。

标记-清除：标记出需要被gc的对象并直接清除。会造成空间不连续，碎片化。

标记-整理：标记并清除需要被gc的对象后会将存活的对象整理为连续的。

复制：将一个内存区域分成两块，其中一块进行gc时，将存活的对象复制放入另一块空间。