Java的内存回收

1、对象的状态转换

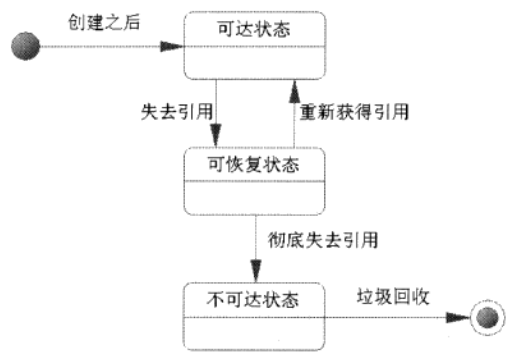
只要有引用变量引用一个对象，这个对象就不会被垃圾回收机制回收。JVM的垃圾回收机制采用有向图方式来管理内存中的对象，因此可以方便地解决循环引用的问题，比如3个对象相互引用。采用有向图管理内存中的对象具有较高的精度，但缺点是效率低。

当一个对象在堆内存中运行时，根据它在对应有向图中的状态，可以把它所处的状态分为3种：

可达状态：对象被创建后有一个以上的引用变量引用它，在有向图中可从起始顶点导航到该对象，则处于可达状态。

可恢复状态：程序中某个对象不再有任何引用变量引用它，它将进入可恢复状态，此时从有向图的起始点不能导航到该对象。在回收该对象之前，系统会调用可恢复状态的对象的finalize方法进行资源清理，如果系统在调用此方法重新让一个以上引用变量引用该对象，则这个对象会再次变为可达状态；否则，该对象将进入不可达状态。

不可达状态：当对象的所有关联被切断，且系统调用所有对象的finalize方法依然没有使该对象变成可达状态，那这个对象将永久性失去引用，最有变成不可达状态。只有当一个对象处于不可达状态时，系统才会真正回收该对象所占有的资源。



这里需要注意，一个对象可以被一个方法局部变量所引用，也可以被其他类的类变量引用，或者被其他对象的实例变量所引用。当某个对象被其他类的类变量引用时，只有该类被销毁后，该对象才会进入可恢复状态；当某个对象被其他对象的实例变量引用时，只有当引用该对象的对象被销毁或变成不可达状态后，该对象才会进入不可达状态。

2、四种对象的引用方式（强引用、软引用、弱引用、虚引用）

* 强引用：广泛使用的引用类型，被强引用所引用的Java对象不会被垃圾回收机制回收，即使系统内存非常紧张；即使有些Java对象以后永远都不会被用到，JVM也不会回收被强引用所引用的Java对象。另外，定义字符串时，系统会缓存这个字符串直接量（会使用强引用来引用它），系统不会回收被缓存的字符串常量。
* 软引用：需要通过SoftRefrence类来实现，当一个对象只具有软引用时，它有可能被垃圾回收机制回收。对于只有软引用的对象而言，当系统内存空间足够时，它不会被系统回收，程序也可使用该对象；当系统内存空间不足时，系统将会回收它。
* 弱引用：与软引用相似，区别在于弱引用所引用对象的生存期更短。弱引用通过WeakRefrence来实现，弱引用和软引用很像，但弱引用的引用级别更低。对于只有弱引用的对象而言，当系统垃圾回收机制运行时，不管系统内存是否足够，总会回收该对象所占用的内存。当然，比不是立即回收，而是等到系统垃圾回收机制运行时才会被回收。所以，弱引用具有很大的不确定性，因为垃圾回收机制的运行不受程序员控制，因此程序获取弱引用所引用的Java对象时必须小心空指针异常——通过弱引用所获取的Java对象可能是null。
* 虚引用：软引用和弱引用可以单独使用，但虚引用不能单独使用，必须和引用队列联合使用。虚引用的主要作用就是跟踪对象被垃圾回收的状态，程序可以通过检查与虚引用关联的引用队列中是否已经包含指定的虚引用，从而了解虚引用所引用的对象是否即将被回收。因为虚引用的对象被释放前，将把它对应的虚引用添加到它的关联的引用队列中，这使得可以在对象被回收之前采取行动。

使用这些引用类可以避免在程序执行期间将对象留在内存中，如果以软引用、弱引用或虚引用的方式引用对象，垃圾回收器就能够随意释放对象。如果希望尽可能减小程序在其生命周期中所占用的内存大小，这些引用类就很有好处。

3、Java的内存泄漏

定义：存在无用的内存没有被回收回来。比如程序中有一些Java对象，他们处于可达状态 ，但程序以后永远都不会再访问它们，那它们所占用的内存空间也不会被回收，它们所占用的空间也会产生内存泄漏。

例如：ArrayList中要删除最后一个元素，此时index会减一，如果不执行将删除位置的引用变量置为null，那么被删除的对象将一直处于可达状态，而对ArrayList而言永远不会访问到它，垃圾回收机制又不会回收它，这就产生了内存泄漏。

4、垃圾回收机制

垃圾回收机制主要完成两件事：

跟踪并监控每个Java对象，当某个对象处于不可达状态时，回收该对象所占用的内存；

清理内存分配、回收过程中产生的内存碎片。

5、内存管理的小技巧

（1）尽量使用直接量，少用new。

（2）使用StringBuilder和StringBuffer进行字符串拼接。

（3）尽早释放无用对象的引用。

（4）尽量少使用静态变量。因为类的静态成员变量，其生命周期与类同步，在类不被卸载的情况下，类对应的Class对象会常驻内存，直到程序运行结束，其静态成员变量也同样如此。如下程序：

|  |
| --- |
| class Person{  static Object *obj* = new Object(); } |

根据前面介绍的分代回收机制，JVM会将程序中Person类的信息存入Permanent代，也就是说，Person类、obj引用变量都将存放在Permanent代里，这将导致obj对象一直有效，从而使得obj所引用的Object得不到回收。

（5）避免在经常调用的方法、循环中创建Java对象

（6）缓存经常使用的对象

经常被使用的对象可以考虑用缓存池保存起来，当下次需要时就可以直接拿出来使用。典型的缓存就是数据库连接池，里边缓存了大量的数据库连接，每次程序需要访问数据库时都可以直接取出数据库连接。

实现缓存时通常有两种方式：

使用HashMap进行缓存；

直接使用某些开源的缓存项目。

（7）尽量不要使用finalize方法

前边介绍，一个对象失去引用之后，垃圾回收器准备回收该对象前会先调用对象的finalize方法来执行资源清理。所以很多程序员会考虑用此方法进行资源清理。但事实上这样做是非常拙劣的选择，在垃圾回收器本身已经严重制约应用程序性能的情况下，如果再选择使用finalize方法进行资源清理，无疑是一种火上浇油的行为，这样会导致回收器的负担更大，导致程序运行效率更差。

（8）考虑使用SoftReference

当程序要创建长度很大的数组时，可以考虑使用SoftRefrence来包装数组元素，而不是直接让数组元素来引用对象。SoftRefrence是个很好的选择：当内存足够时，它的功能等同于普通引用；当内存不足时，它会牺牲自己，释放软引用所引用的对象。所以，程序取出SoftRefrence所引用的Java对象之后，应该显示判断该对象是否为null，以考虑是否要重建该对象。