Java内存回收

终于来到了令人激动过的Java垃圾回收机制。我为你许下一个美好承诺(垃圾回收机制)，你却辜负我的期待！！ 本笔记所有知识点来自于李刚《Java程序员的基本素养》

- JVM何时回收对象所占内存
- JVM会不会漏掉回收对象，造成内存泄漏
- JVM回收内存实现细节
- 垃圾回收机制实现细节

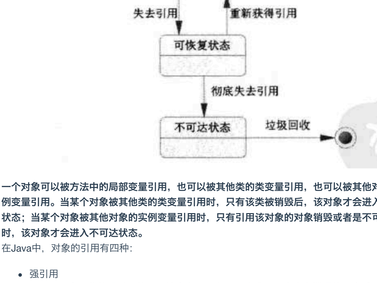
注：本人也是初次接触这方面知识，因此按照书籍上的编排顺序来记录笔记，以便有个更好的关联性。

1:Java引用的种类

1. 当程序员在键盘上敲下new关键字时，其实就是在想JVM申请内存空间，JVM会根据变量类型来分配相对应大小的内存；当堆中的对象失去引用时，该对象就会被JVM清除掉，并且回收他们所占用的空间。
2. Java的内存管理包括内存分配和内存回收两部分，且这两部分都是由JVM自动完成的。
3. JVM判定是否回收一个对象的标准在于：该对象是否还存在引用。如果存在，则不回收，否则就进行回收，释放内存。
4. ***JVM的有向图机制：***
   因为JVM判定是否回收一个对象的标准是是否存在引用，因此JVM如何去遍历在堆中的整个有引用的对象的呢。这里利用的就是JVM的有向图机制。
   在Java中，可以把引用变量和对象作为有向图中的顶点，其中引用关系作为有向边，所以有向边必然是从引用变量指向对象的。而且又因为Java中所有的对象都是在某个线程中产生的，因此线程起点可以当作整个有向图的起点。因此如果在这个以线程起点为有向图中对象是处于可达状态，那么该对象不会被垃圾回收机制回收，否则就会被垃圾回收机制回收。
   下面是一个例子，例子来自于书本：

```
class Node {
Node next;
String name;
public Node(String name) {
    this.name = name;
}
public class NodeTest {
public static void main(String[] args) {
    Node n1 = new Node("n1");
    Node n2 = new Node("n2");
    Node n3 = new Node("n3");
    n1.next = n2;
    n3 = n2;
    n2 = null;
}
}
}
```

   下面这幅图表示了上述代码中对象和引用之间的关系：



因此在这个以main为有向图起点的有向图中，第二个对象有两条路径，第一个对象有一条路径，但是对于第三个对象，没有引用变量指向这个对象，因此这个对象就会被垃圾回收机制回收。

对象在堆中运行的三种状态

- 可达状态:
  当一个对象被创建后，如果有一个以上的引用变量指向它，那么它就处于可达状态，程序可以用这个引用变量来使用这个对象的属性和方法。
- 可恢复状态:
  当堆中的一个对象失去引用之后，它首先会进入可恢复状态。系统会在调用垃圾回收机制回收他之前，会先使用finalize方法进行资源清理，如果在这个过程中，重新使得一个以上的引用变量指向该对象，那么这个对象会再次达到可达状态，否则该对象进入不可达状态。
- 不可达状态:
  像在可恢复状态中，如果系统调用finalize方法还不能使得某个对象重新有引用变量指向，那么该对象就会处于不可达状态，接下来就会被垃圾回收机制回收了。
  下图表示了这三种状态的转换：

一个对象可以被方法中的局部变量引用，也可以被其他类的类变量引用，也可以被其他对象的实例变量引用。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，只有引用该对象的对象销毁或者是不可达状态时，该对象才会进入不可达状态。
在Java中，对象的引用有四种：

- 强引用
  JVM是不会回收的强引用的，无论内存是否充足。造成Java中内存泄漏的一个重要原因就是强引用。
- 软引用
  当内存充足时，软引用和强引用一样，当内存不足时，JVM就会回收软引用对象。
- 弱引用
  无论内存是否充足，系统垃圾回收机制运行时都会回收该对象
- 虚引用
  虚引用主要用于对象垃圾回收的状态。

1. ***内存泄漏***
   Java中不断为创建的对象分配内存，而那些不再使用的内存空间应当及时的回收回来，以便给其他的对象使用。如果存在无用的内存没有被回收回来，那么就会出现内存泄漏。
   在C++中，内存的回收必须由程序员显示的来回收，如果程序员没有进行内存回收，就会造成内存泄漏。而在Java中，由于存在垃圾回收机制，对于那些在堆中处于不可达状态的对象，垃圾回收机制会进行自动回收，而对于那些处于可达状态，但是程序员又永远不会使用的对象，垃圾回收机制是不会进行回收的，因为他们是强引用。这种情况就会造成内存泄漏。

   - **C++内存泄漏和Java内存泄漏的区别**
     C++的内存泄漏主要是针对于Java中那些不可达的状态的对象，但是C++中不存在JVM中的垃圾回收机制，因此程序对于这种对象无能为力，但是在Java中JVM会检测堆中每个对象的运行状态，因此会回收它。
     Java中内存泄漏往往存在于那种强引用对象，这些对象仍然存在引用变量指向它但是程序却永远不会用到这些对象，JVM对于这种对象是不会进行回收的，在C++中程序员可以显式的去释放这些内存，因此Java中内存泄漏往往出现在这种情况。

2. ***垃圾回收机制***
   垃圾回收机制主要完成两件事件:

   - 跟踪并且监控堆中每个Java对象，当某个对象处于不可达状态的时候，垃圾回收机制就会回收它；
   - 清理内存分配，和内存回收时产生的内存碎片

- 我们之前提到，当堆中的一个对象失去引用时，JVM的垃圾回收机制就会回收它。但实际上是当一个堆中对象失去引用时，它不会立即被回收，只有当垃圾回收机制运行时候，它才会被垃圾回收机制回收。

1. ***内存管理小技巧***

   - 尽量使用直接量
     当使用一些基本数据类型的包装类时，尽量使用直接量，而非new对象。这些包装类包含String, Byte, Short, Integer, Double, Float, Boolean, Character, Long. 比如:

2. 当使用new方法时，不仅仅会在字符串池中存在"str"字符串，而且会在堆中创建一个char[]组，其中存储的是s t r.

```
String str = "str";//建议使用这种
String str = new String("str");//不建议使用这种
```

   - 使用StringBuilder和StringBuffer来连接字符串
     因为String类型的字符串是不可变的，因此在连接字符串的过程中，往往会生成很多临时性的字符串。而StringBuiler和StringBuffer代表的是字符串序列可变的字符串。
   - 尽早释放无用对象的引用
     通常情况下局部方法的引用变量生命期很短，因此在方法结束之后，变量生命期也就结束了。对于那些无用的对象我们可以显示的把它释放，等到JVM来回收它。
   - 尽量少用静态变量来指向堆中的对象
   - 避免在经常调用的方法和循环中创建新的对象

后记：本章内容较多，而且设计垃圾回收机制的实现算法。感兴趣可以自己阅读书本。