20180321_如果想在 GC 中生存 1 次怎么办

1- 生存一次，释放掉对象的引用，但是在对象的finalize方法中重新建立引用，但是有一此方法只会被调用一次，所以能在GC中生存 一次

2- finalize() 是一个用于释放非 Java 资源的方法.JVM 有很大的可能不调用对象的 finalize() 方法，因此很难证明使用该方法释放资源是有效的。

3- 是在Java中很不幸，如果内存总是充足的，那么垃圾回收可能永远不会进行，也就是说 filalize()可能永远不被执行，显然指望它做收尾工作是靠不住的。system.gc() 也是靠不住的。

4- 那么finalize()究竟是做什么的呢？它最主要的用途是回收特殊渠道申请的内存。Java程序有垃圾回收器，所以一般情况下内存问题不用程序员操心。但有一种JNI(Java Native Interface)调用non-Java程序（C或C++），finalize()的工作就是回收这部分的内存。

生存一次，释放掉对象的引用，但是在对象的finalize方法中重新建立引用，但是有一此方法只会被调用一次，所以能在GC中生存 一次

《**JAVA**编程思想》：

Java提供**finalize**()方法，垃圾回收器准备释放内存的时候，会先调用finalize()。

(1).对象不一定会被回收。

(2).垃圾回收不是析构函数。

(3).垃圾回收只与内存有关。

(4).**垃圾回收和finalize()都是靠不住的，只要JVM还没有快到耗尽内存的地步，它是不会浪费时间进行垃圾回收的。**

有时当撤消一个对象时，需要完成一些操作。例如，如果一个对象正在处理的是非Java 资源，如文件句柄或window 字符字体，这时你***要确认在一个对象被撤消以前要保证这些资源被释放。为处理这样的状况，Java 提供了被称为收尾（finalization ）的机制。使用该机制你可以定义一些特殊的操作，这些操作在一个对象将要被垃圾回收程序释放时执行。***

要给一个类增加***收尾（finalizer ），你只要定义finalize ( ) 方法***即可。Java 回收该类的一个对象时，就会调用这个方法。在finalize ( )方法中，你要**指定在一个对象被撤消前必须执行的操作。垃圾回收周期性地运行，检查对象不在被运行状态引用或间接地通过其他对象引用。**就在对象被释放之前，Java 运行系统调用该对象的finalize( ) 方法。

finalize()方法的通用格式如下：

```
protected void finalize( ){
    // finalization code here
}
```

其中，关键字protected是防止在该类之外定义的代码访问finalize()标识符。该标识符和其他标识符将在第7章中解释。

理解finalize( ) 正好在垃圾回收以前被调用非常重要。例如当一个对象超出了它的作用域时，finalize( ) 并不被调用。这意味着你**不可能知道何时——甚至是否——finalize( ) 被调用。**因此，你的程序应该提供其他的方法来释放由对象使用的系统资源，而不能依靠finalize( ) 来完成程序的正常操作。

注意：如果你熟悉C ，那你知道***C 允许你为一个类定义一个撤消函数（destructor ）***，它在对象正好出作用域之前被调用。Java不支持这个想法也不提供撤消函数。finalize() 方法只和撤消函数的功能接近。当你对Java 有丰富经验时，你将看到因为Java使用垃圾回收子系统，几乎没有必要使用撤消函数。

理解finalize()-析构函数的替代者

by Tim Gooch

在许多方面，Java 类似于 C++。Java 的语法非常类似于 C++，Java 有类、方法和数据成员；Java 的类有构造函数； Java 有异常处理。

但是，如果你使用过 C++ 会发现 Java 也丢掉一些可能是你熟悉的特性。这些特性之一就是析构函数。取代使用析构函数，Java 支持finalize() 方法。

在本文中，我们将描述 finalize() 与 C++ 析构函数的区别。另外，我们将创建一个简单的 Applet 来演示 finalize() 是如何**工作**的。

最终的界限

与 Java 不同，C++ 支持局部对象（基于栈）和全局对象（基于堆）。因为这一双重支持，C++ 也提供了自动构造和析构，这导致了对构造函数和析构函数的调用，（对于堆对象）就是内存的分配和释放。

在 Java 中，所有对象都驻留在堆内存，因此局部对象就不存在。结果，Java 的设计者觉得不需要析构函数（象 C++ 中所实现的）。

取而代之，Java 定义了一个特殊的方法叫做finalize() ，它提供了 C++ 析构函数的一些功能。但是，finalize() 并不完全与 C++ 的析构函数一样，并可以假设它会导致一系列的问题。finalize() 方法作用的一个关键元素是 Java 的垃圾回收器。

垃圾回收器

在 C/C++、Pascal和其他几种多种用途的编程语言中，开发者有责任在内存管理上发挥积极的作用。例如，如果你为一个对象或数据结构分配了内存，那么当你不再使用它时必须释放掉该内存。

在 Java 中，当你创建一个对象时，Java 虚拟机（JVM）为该对象分配内存、调用构造函数并开始跟踪你使用的对象。当你停止使用一个对象（就是说，当没有对该对象有效的引用时），JVM 通过垃圾回收器将该对象标记为释放状态。

当垃圾回收器将要释放一个对象的内存时，它调用该对象的finalize() 方法（如果该对象定义了此方法）。垃圾回收器以独立的低优先级的方式运行，只有当其他线程挂起等待该内存释放的情况出现时，它才开始运行释放对象的内存。（事实上，你可以调用System.gc() 方法强制垃圾回收器来释放这些对象的内存。）

在以上的描述中，有一些重要的事情需要注意。首先，只有当垃圾回收器释放该对象的内存时，才会执行finalize()。如果在 Applet 或应用程序退出之前垃圾回收器没有释放内存，垃圾回收器将不会调用finalize()。

其次，除非垃圾回收器认为你的 Applet 或应用程序需要额外的内存，否则它不会试图释放不再使用的对象的内存。换句话说，这是完全可能的：一个 Applet 给少量的对象分配内存，没有造成严重的内存需求，于是垃圾回收器没有释放这些对象的内存就退出了。

显然，如果你为某个对象定义了finalize() 方法，JVM 可能不会调用它，因为垃圾回收器不曾释放过那些对象的内存。调用System.gc() 也不会起作用，因为它仅仅是给 JVM 一个建议而不是命令。

finalize() 有什么优点呢?

如果finalize() 不是析构函数，JVM 不一定会调用它，你可能会疑惑它是否在任何情况下都有好处。事实上，在 Java 1.0 中它并没有太多的优点。

根据 Java 文档，**finalize() 是一个用于释放非 Java 资源的方法。**但是，JVM 有很大的可能不调用对象的finalize() 方法，因此很难证明使用该方法释放资源是有效的。

Java 1.1 通过提供一个System.runFinalizersOnExit() 方法部分地解决了这个问题。（不要将这个方法与 Java 1.0 中的System.runFinalizations() 方法相混淆。）不象System.gc() 方法那样，System.runFinalizersOnExit() 方法并不立即试图启动垃圾回收器。而是当应用程序或 Applet 退出时，它调用每个对象的finalize() 方法。

正如你可能猜测的那样，通过调用System.runFinalizersOnExit() 方法强制垃圾回收器清除所有独立对象的内存，当清除代码执行时可能会引起明显的延迟。现在建立一个示例 Applet 来演示 Java 垃圾回收器和finalize() 方法是如何相互作用的。

回收垃圾

通过使用Java Applet Wizard 创建一个新的 Applet 开始。当提示这样做时，输入*final_things*作为 Applet 名，并选择不要生成源文件注释。

接下来，在Java Applet Wizard 进行第三步，不要选择多线程选项。在第五步之前，根据需要修改 Applet 的描述。

当你单击Finish 后，Applet Wizard 将生成一个新的工作空间，并为该项目创建缺省的 Java 文件。从列表 A 中选择适当的代码输入（我们已经突出显示了你需要输入的代码）。

当你完成代码的输入后，配置Internet 浏览器将System.out 的输出信息写到Javalog.txt 文件中。（在IE 选项对话框的高级页面中选择起用 Java Logging。）

编译并运行该 Applet。然后，等待 Applet 运行（你将在状态栏中看到 Applet 已启动的信息），退出浏览器，并打开Javalog.txt 文件。你将会发现类似于下列行的信息：

1000 things constructed

0 things finalized

正如你能够看到的那样，建立了1,000个对象仍然没有迫使垃圾回收器开始回收空间，即使在 Applet 退出时也没有对象被使用。

现在，删除在stop() 方法第一行中的注释符以起用System.gc() 方法。再次编译并运行该 Applet ，等待 Applet 完成运行，并退出浏览器。当你再次打开Javalog.txt 文件，你将看到下列行：

1000 things constructed

963 things finalized

这次，垃圾回收器认为大多数对象未被使用，并将它们回收。按顺序，当垃圾回收器开始释放这些对象的内存时，JVM 调用它们的finalize() 方法。

继承finalize()?

顺便，如果你在类中定义了finalize() ，它将不会自动调用基类中的方法。在我们讨论了finalize() 与 C++ 的析构函数的不同点后，对这个结论不会惊讶，因为***为某个类定制的清除代码另一个类不一定会需要***。

如果你决定要通过派生一个类的finalize() 方法来调用基类中的finalize() 方法，你可以象其他继承方法一样处理。

```
 protected void finalize(){
          super.finalize();
          // other finalization code...
}
```

除了允许你控制是否执行清除操作外，这个技术还使你可以控制当前类的finalize() 方法何时执行。

结论

然而有益的是，Java 的自动垃圾回收器不会失去平衡。作为便利的代价，你不得不放弃对系统资源释放的控制。不象 C++ 中的析构函数，Java Applet 不会自动执行你的类中的finalize() 方法。事实上，如果你正在使用 Java 1.0，即使你试图强制它调用finalize() 方法，也不能确保将调用它。

因此，你不应当依靠finalize() 来执行你的 Applet 和应用程序的资源清除工作。取而代之，你应当明确的清除那些资源或创建一个try...finally 块（或类似的机制）来实现。

finalize方法是与Java编程中的垃圾回收器有关系。即：当一个对象变成一个垃圾对象的时候，如果此对象的内存被回收，那么就可以调用系统中定义的finalize方法来完成 当然，Java的内存回收可以由JVM来自动完成。如果你手动使用，则可以使用上面的方法。 举例说明：

[java] view plain copy print?

```
1.  public class FinalizationDemo {
2.      public static void main(String[] args) {
3.          Cake c1 = new Cake(1);
4.          Cake c2 = new Cake(2);
5.          Cake c3 = new Cake(3);
6.
7.          c2 = c3 = null;
8.          System.gc(); //Invoke the Java garbage collector
9.      }
10. }
11.
12. class Cake extends Object {
13.     private int id;
```

```
14.     public Cake(int id) {
15.         this.id = id;
16.         System.out.println("Cake Object " + id + "is created");
17.     }
18.
19.     protected void finalize() throws java.lang.Throwable {
20.         super.finalize();
21.         System.out.println("Cake Object " + id + "is disposed");
22.     }
23. }
```

[java] view plain copy print?

```
1. C:\1>java FinalizationDemo
2. Cake Object 1is created
3. Cake Object 2is created
4. Cake Object 3is created
5. Cake Object 3is disposed
6. Cake Object 2is disposed
```

final

修饰符（关键字）如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承。因此一个类不能既被声明为 abstract的，又被声明为final的。将变量或方法声明为final，可以保证它们在使用中不被改变。被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为final的方法也同样只能使用，不能重载。

---

finally

异常处理时提供 finally 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）。一般异常处理块需要。

---

finalize

方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。**它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。**finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的。

Java中所有类都从Object类中继承finalize()方法。

当垃圾回收器(garbage colector)决定回收某对象时，就会运行该对象的finalize()方法。值得C++程序员注意的是，finalize()方法并不能等同与析构函数。Java中是没有析构函数的。C++的析构函数是在对象消亡时运行的。由于C++没有垃圾回收，对象空间手动回收，所以一旦对象用不到时，程序员就应当把它delete()掉。所以析构函数中经常做一些文件保存之类的收尾工作。但**是在Java中很不幸，如果内存总是充足的，那么垃圾回收可能永远不会进行，也就是说filalize()可能永远不被执行，显然指望它做收尾工作是靠不住的。**

那么finalize()究竟是做什么的呢？它最主要的用途是回收特殊渠道申请的内存。Java程序有垃圾回收器，所以一般情况下内存问题不用程序员操心。但有一种JNI(Java Native Interface)调用non-Java程序（C或C++），finalize()的工作就是回收这部分的内存。