Atitit gc 垃圾回收原理与概论and 自动资源管理的艺术 v2

[1. 为什么需要gc 1](#_Toc22022)

[1.1. 如果长期不被释放，可能导致OOM。 1](#_Toc30381)

[1.2. ，目的在于防止由程序猿引入的人为的内存泄露 1](#_Toc17204)

[2. 思考GC需要完成的3件事情： 1](#_Toc12951)

[2.1. 哪些内存需要回收？什么时候回收？如何回收？ 1](#_Toc1080)

[2.2. 自动资源管理的对象db conn,http conn,file 2](#_Toc5101)

[3. 如何关联----wrap file，network socket 2](#_Toc27448)

[4. 生活中常见的垃圾gc法 3](#_Toc21896)

[4.1. 移动法 标记法 3](#_Toc31666)

[5. 常见gc算法 3](#_Toc30930)

[5.1. 引用计数算法：（老牌垃圾回收算法 3](#_Toc5573)

[5.2. 根搜索算法的概念： 3](#_Toc6243)

[5.3. 1、标记清除算法的概念： 3](#_Toc212)

[5.4. 复制算法：（新生代的GC） 3](#_Toc16533)

[5.5. 七、分代收集算法：（新生代的GC+老年代的GC） 3](#_Toc30794)

[6. ref 4](#_Toc1459)

[6.1. 垃圾回收的算法与实现 Ruby之父松本行弘推荐GC算法 Garbage Collection 一本书掌握自动内存回收的机制 4](#_Toc13196)

[6.2. 垃圾回收算法手册：自动内存管理的艺术 5](#_Toc3251)

# 为什么需要gc

## ****如果长期不被释放，可能导致OOM****。

1. GC：Garbage Collection 垃圾收集。这里所谓的垃圾指的是**在系统运行过程当中所产生的一些无用的对象，这些对象占据着一定的内存空间，如果长期不被释放，可能导致OOM**。

## ，目的在于防止由程序猿引入的人为的内存泄露

在C/C++里是由程序猿自己去申请、管理和释放内存空间，因此没有GC的概念。而在Java中，**后台专门有一个专门用于垃圾回收的线程**来进行监控、扫描，自动将一些无用的内存进行释放，这就是垃圾收集的一个基本思想，目的在于防止由程序猿引入的人为的内存泄露。

# 思考GC需要完成的3件事情：

## 哪些内存需要回收？什么时候回收？如何回收？

## 自动资源管理的对象db conn,http conn,file

内存区域中的**程序计数器、虚拟机栈、本地方法栈**这3个区域随着线程而生，线程而灭；**栈中的栈帧**随着方法的进入和退出而有条不紊地执行着出栈和入栈的操作，每个栈帧中分配多少内存基本是**在类结构确定下来时就已知**的。**在这几个区域不需要过多考虑回收的问题，因为方法结束或者线程结束时，内存自然就跟着回收了**。

而**Java堆和方法区**则不同，一个接口中的多个实现类需要的内存可能不同，一个方法中的多个分支需要的内存也可能不一样，我们只有在程序处于运行期间时才能知道会创建哪些对象，**这部分内存的分配和回收都是动态的**，GC关注的也是这部分内存，后面的文章中如果涉及到“内存”分配与回收也仅指着一部分内存。

# 如何关联----wrap file，network socket

我希望改进这一点，也就是说，那所有 gc 相关的数据集中在一起，整个收集过程，除了最终释放那些不再使用的内存外，不会碰用户数据块的内存。

gc 最重要的一点，就是要对堆栈上的数据进行关联。在收集发生时，堆栈上所有临时分配出来的内存块都不应该被释放掉。C 语言本身不提供堆栈遍历的特性，所以要想个自然的方案让用户可以方便的做到这点。

在用户的调用栈上，每个调用级上，临时分配的内存都被自然挂接在当前级别的堆栈挂接点上，一旦调用返回，当前级别的所有临时内存块都应该和根断开。当然，如果内存块作为返回值出现的话，需要保留。在 C 里，我们需要给每个函数的入口和出口都做一个监护，保证 gc 的正确工作。（如果是 C++ ，要稍微方便一点，在函数进入点设置一个 guard 对象即可）因为这个监护过程会非常频繁，对其的优化是重点工作。

Atitit gc资源释放器死锁解除器 适合于游戏引擎，数据库释放 爬虫网络连接释放等

# 生活中常见的垃圾gc法

## 移动法 标记法 分代法（物业市政垃圾桶

# 常见gc算法

## ****引用计数算法：（老牌垃圾回收算法****

引用计数算法的实现简单，判定效率也高，大部分情况下是一个不错的算法。很多地方应用到它。例如：

微软公司的COM技术：Computer Object Model

使用ActionScript3的FlashPlayer

Python

但是，主流的java虚拟机并没有选用引用计数算法来管理内存，其中最主要的原因是：**它很难解决对象之间相互循环引用的问题**。

## ****根搜索算法的概念：****

由于引用计数算法的缺陷，所以JVM一般会采用一种新的算法，叫做**根搜索算法**。它的处理方式就是，**设立若干种根对象，当任何一个根对象到某一个对象均不可达时，则认为这个对象是可以被回收的**

## ****1、标记清除算法的概念：****

标记-清除算法是现代垃圾回收算法的思想基础。标记-清除算法将垃圾回收分为两个阶段：标记阶段和清除阶段。一种可行的实现是，在标记阶段，**首先通过根节点，标记所有从根节点开始的可达对象**。

## ****复制算法：（新生代的GC）****

现在的商业虚拟机都采用这种收集算法来回收新生代，新生代中的对象98%都是“朝生夕死”的，所以并不需要按照1:1的比例来划分内存空间，而是**将内存分为一块比较大的Eden空间和两块较小的Survivor空间**，每次使用Eden和其中一块Survivor

## ****七、分代收集算法：（新生代的GC+老年代的GC）****

当前商业虚拟机的GC都是采用的“分代收集算法”，这并不是什么新的思想，只是**根据对象的存活周期的不同将内存划分为几块儿**。一般是把Java堆分为新生代和老年代：**短命对象归为新生代，长命对象归为老年代**。

* **少量对象存活，适合复制算法**：在**新生代**中，每次GC时都发现有大批对象死去，只有少量存活，那就选用**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GC。
* **大量对象存活，适合用标记-清理/标记-整理**：在**老年代**中，因为对象存活率高、没有额外空间对他进行分配担保，就必须使用“**标记-清理**”/“标记-整理”算法进行GC。

注：**老年代的对象中，有一小部分是因为在新生代回收时，老年代做担保，进来的对象；绝大部分对象是因为很多次GC都没有被回收掉而进入老年代**。

# ref

## 垃圾回收的算法与实现 Ruby之父松本行弘推荐GC算法 Garbage Collection 一本书掌握自动内存回收的机制

第1章　学习GC之前

第2章　GC标记-清除算法

第3章　引用计数法

第4章　GC复制算法

第5章　GC标记-压缩算法

第6章　保守式GC

第7章　分代垃圾回收

第8章　增量式垃圾回收

第9章　RC Immix算法

第10章　Python的垃圾回收

10.4　第0层 通用的基础分配器　197

10.5　第1层 Python低级内存分配器　198

10.6　第2层 Python对象分配器　208

10.7　第3层 对象特有的分配器　231

第11章　DalvikVM的垃圾回收

第12章　Rubinius的垃圾回收

第13章　V8的垃圾回收

## 垃圾回收算法手册：自动内存管理的艺术

第1章　引言 1

第2章　标记–清扫回收 15

第3章　标记–整理回收 28

第4章　复制式回收 38

第5章　引用计数 49

第6章　垃圾回收器的比较 64

第7章　内存分配 72

第8章　堆内存的划分 87

第9章　分代垃圾回收 95

第10章　其他分区策略 117

第11章　运行时接口 138

第12章　特定语言相关内容 188

第13章　并发算法预备知识 202

第14章　并行垃圾回收 242

第15章　并发垃圾回收 271

第16章　并发标记–清扫算法 285

第17章　并发复制、并发整理算法 298

第18章　并发引用计数算法 322

第19章　实时垃圾回收 333

《垃圾回收算法手册：自动内存管理的艺术》([英] 理查德·琼斯)【简介\_书评\_在线阅读】 - 当当图书.html