# Generational Garbage Collection in Python and Java

Yikun Su

## Abstract

During the process of the program, Garbage Collection helps to save memory space and improve efficiency of our computers. In this article, we are going to give a brief introduction of an important algorithm in GC called Generational Garbage Collection, which can help improve the speed of memory allocation and save time.

**ACM Reference Format:**
Yikun Su. 2023. Generational Garbage Collection in Python and Java. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 Introduction

Garbage Collection, also called GC, plays an significant role in saving memory space and improving computers' efficiency. During the process of the program, it is common that the program produces many objects(like variables) which still occupy memory space after finishing their "work". If we do not clear(or release) these "garbage", it is easy to result in a serious problem that the memory space runs out because of too much garbage, which leads to system breakdown. In order to solve such problems and improve computers' efficiency, designers have designed a special algorithm called "Garbage Collection", also called GC, which can release garbage and recycle memory space automatically. It is run by "Garbage Collectors" and supported by GC Algorithms.

In Java and Python especially, we also use GC to collect garbage in an efficient way. No matter how different their exact algorithms are from each other, they do both apply one of the most important GC algorithms, the Generational Garbage Collection, to be their main idea when running GC. Generational Garbage Collection, in which we divide memory space into several parts and process them in different ways, is used to improve the efficiency of objects' memory allocation and garbage collection. But before we talk about it, let's look back at how objects are stored in memory space. For most objects, they have a short lifetime and only a few have a long lifetime. This should be carefully noticed since garbage collection would result in "stop the world"(STW) stage while running, which means that the program would stop processing. Put it simply, if we just use the algorithms without certain conditions, the "STW" would be really time-consuming and leads to low efficiency.

In an effort to solve such problem and improve the speed of memory allocation, Java and Python, as we have talked about above, adopt the thought of Generational Garbage Collection.The following will look into why and how they use such thought more deeply.


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 2 GGC Specifically for Python and Java

### 2.1 For Python

In Python [1], Generational Garbage Collection is used to solve the problem of circular reference which may happen in Reference Counting Algorithm. In Reference Counting Algorithm, each object corresponds to a counter. If an object is created or referred to, we have the counter increment; else if the object is released from the reference, we have the counter decrement. When the counter is equal to 0, the related object is viewed as garbage and released.

As Reference Counting Algorithm features easy, high efficiency and low-latency operations, it is adopted to do the major GC work regardless of the problem of circular reference. Meanwhile, other algorithms have some complicated problems. For example, Tracing GC can avoid producing circular reference, but it is used mainly in dealing with big data and would cause the whole program to stop, which is both memory-consuming and time-consuming. Therefore, Python choose a combination of Reference Counting Algorithm and Circular GC. The latter method mainly deals with the problem of circular reference, using the thought of Generational Garbage Collection.

While using Generational Garbage Collection in Python, we devide memory into 3 parts, which are respectively called the younger generation, the middle generation and the old generation. The frequency of GC in the three generations would decrease as the life of objects become longer.

The newly created objects will be stored in the young generation and their generation will add 1 after finding and clearing the inactive (not cited) objects. The older the object is, the longer their time to live is, namely bigger their threshold is. That is to say, we will perform GC process while the number of objects reaches the threshold. Then the objects survived will be transported to the middle generation, then the old generation. Therefore, the objects in the old generation are the oldest and maybe live in the whole process of the system.

### 2.2 For Java

As for Java [2], there still exists the problem of circular reference. This time, Java choose not to use Reference Counting Algorithm but simply do the Generational Garbage Collection. Many garbage collectors in Java adopt the thought of Generational Garbage Collection, including G1, CMS and Serial GC. The following mainly talks about how Generational Garbage Collection works.

In this process, the memory is divided into 2 parts, which are respectively called the young generation and the old generation. The young generation occupy $\frac{1}{3}$ of the heap memory and the old generation $\frac{2}{3}$ of it. The newly created objects would be stored in the young generation, and the ones living after GC process for many times will be stored in the old generation. The transportation from the young generation to the old generation is performed if the objects are too old or have occupied too many space of memory.

## 3 Other differences on GC

What we talked about above is really different from Python and Java. Now let's take a brief look at other differences on GC in the two programming language.

### 3.1 On the method

Python choose mainly Reference Counting Algorithm while Java choose Generational Garbage Collection directly.

### 3.2 On the solutions of circular reference

Python choose Generational Garbage Collection while Java simply get rid of Reference Counting Algorithm.

### 3.3 On the specific Generational Garbage Collection

Python divides memory space into 3 parts, adopting a simple mechanism but might have trouble with large data sets. Java divides memory space into 2 parts, doing well in programs which requires high-performance but needs more complicated algorithms like Tracing GC, whose problems are difficult to solve.

## 4 Conclusion

For the growing concern about memory safety and more efficient programs, GC with Generational Garbage Collection will play a more important role in programming languages, and more advanced garbage collectors and GC algorithms will be developed, including Parallel and concurrent garbage collection mechanism to be applied in Python and G1, CMS, and ZGC [3]frequently used in Java.

## References

[1] GUO Fen,LIU Ming. Principle of Python garbage collector and its application[J]. INFORMATION TECHNOLOGY,2009(7):93-97. DOI:10.3969/j.issn.1009-2552.2009.07.026.

[2] ZENG Tian hui,YU Shi cai,DONG Rong hui. Performance tuning of Java garbage collection mechanism[J]. COMPUTER ENGINEERING AND DESIGN,2006,27(17):3242-3244,3247. DOI:10.3969/j.issn.1000-7024.2006.17.042.

[3] HETING Li. Recoverable Generational Garbage Collector for Java in NVM[D].Shanghai Jiao Tong University School of Software, School of Electronic Information and Electrical Engineering,2020.DOI:10.27307/d.cnki.gsjtu.2019.002771.