

# Java Heap memory

## 1. Java Heap là gì

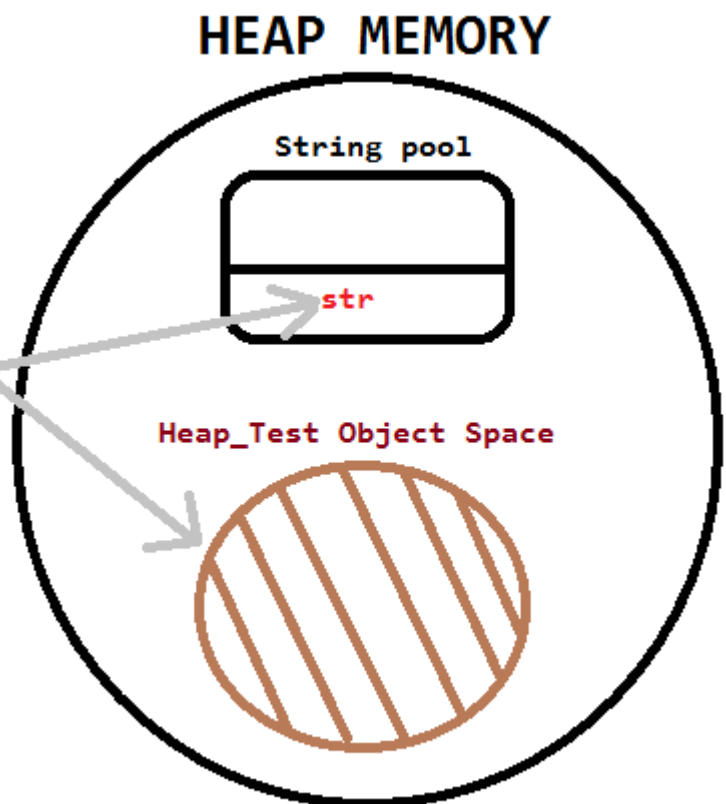
Khi JVM start up, nó được cấp phát bộ nhớ Memory từ Hệ điều hành. JVM sử dụng memory cho tất cả nhu cầu của nó và 1 phần bộ nhớ này được gọi là Heap memory. Heap trong java được sinh ra ở cuối cùng của vùng địa chỉ và di chuyển lên trên, bất cứ khi nào chúng ta sử dụng toán tử `new` hoặc khai báo đối tượng sẽ được cấp phát bộ nhớ từ Heap, còn khi object chết hoặc GC, memory sẽ quay trở lại Heap.

Bộ nhớ Heap trong Java được dùng để cấp phát bộ nhớ cho các đối tượng, các lớp JRE lúc thực thi. Bất cứ khi nào, chúng ta tạo đối tượng, nó sẽ được tạo trong bộ nhớ Heap.

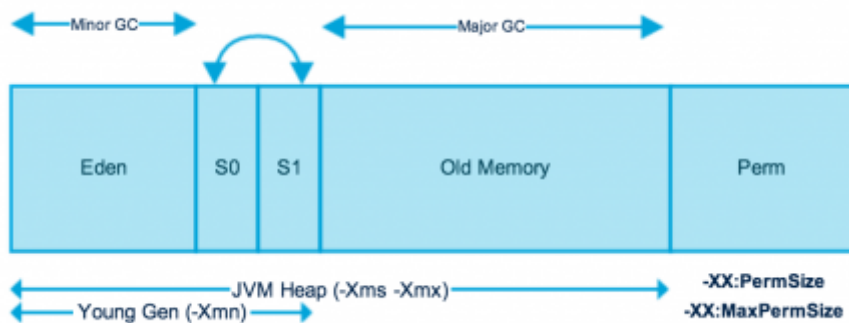
Với những đối tượng không còn được tham chiếu nữa thì trình thu thập rác (Garbage Collection) sẽ giải phóng bộ nhớ mà các đối tượng đó sử dụng.

Đối tượng được tạo trong bộ nhớ Heap có phạm vi truy cập toàn cục, tức là chúng ta có thể truy cập đối tượng đó ở bất kỳ đâu trong ứng dụng

```
public class Heap_Test {  
    public static void main(String[] args)  
    {  
        Heap_Test reff = new Heap_Test();  
        reff.foo();  
    }  
    void foo() {  
        String str = "Heap memory space";  
        System.out.println(param);  
    }  
}
```



## 2. Java Heap memory



Heap đôi khi được chia làm 2 vùng (hay thế hệ) gọi là **Nursey** (hay `young space`) và vùng `old space`.

*Nursery* là một phần của *Heap* để dành cho việc cấp phát cho các đối tượng mới. Khi *nursery* bắt đầu đầy, rác sẽ được thu gom bằng một *process* thu gom rác đặc biệt gọi là `young collection`, nơi mà các đối tượng sống đủ lâu trong *nursery* được `promoted` và di chuyển lên vùng `old space`, do đó giải phóng *nursery* để cấp phát các đối tượng khác.

Khi `old space` trở nên đầy, rác sẽ được thu gom bởi process khác được gọi là `old collection`. Lý do đằng sau 1 `nursery` là hầu hết các đối tượng đều là tạm thời và ngắn hạn. Một process `young collection` được thiết kế để nhanh chóng tìm các đối tượng mới được cấp phát mà vẫn tồn tại (*alive*) và di chuyển chúng khỏi *nursery*. Thông thường, `young collection` process sẽ giải phóng một số lượng bộ nhớ nhất định nhanh hơn `old collection` process hay `garbage collection` process của một *heap* mà không có vùng *nursery* (còn gọi là `single-generational heap`)

Từ các version release mới của JVM, có một phần của *nursery* được giữ lại gọi là `keep area`. Vùng này sẽ chứa các đối tượng được cấp phát gần đây nhất và chưa được thu gom cho đến khi `young collection` process tiếp theo được chạy. Điều này sẽ ngăn cản việc đối tượng được di chuyển lên `old space` vì các đối tượng chỉ vừa được cấp phát ngay trước khi `young collection` chạy.

`Permanent generation` của heap được sử dụng để lưu trữ String pool và siêu dữ liệu (metadata) khác nhau theo yêu cầu của JVM liên quan đến Class, phương thức và các java primitives khác.

## 3. Phân loại lưu trữ Object trong Heap

Trong quá trình cấp phát đối tượng, `JVM` sẽ phân biệt giữa các đối tượng nhỏ (*small object*) và lớn (*large object*). Giới hạn khi một đối tượng được xem là lớn phụ thuộc vào các yếu tố như JVM version, độ lớn *heap size*, chiến lược GC và `platform` được sử dụng, nhưng thường sự khác nhau giữa 2 loại đối tượng là khoảng **128kB**.

Các đối tượng nhỏ sẽ được cấp phát trong vùng gọi là `thread local areas` (**TLAs**), **TLAs** là các khối rỗng (*free chunks*) từ *heap* và được giao cho một `Java thread` sử dụng độc quyền. *Thread* này có thể cấp phát các đối tượng trong **TLA** của nó mà không cần phải đồng bộ với các *thread* khác. Khi **TLA** bắt đầu đầy, *thread* chỉ đơn giản yêu cầu một **TLA** mới.

Các đối tượng lớn sẽ không vừa bên trong một **TLA** được cấp phát trực tiếp trên *heap*. Khi một *Nursery* được dùng, các đối tượng lớn sẽ được phân bố trực tiếp trên `old space`. Sự phân bố các đối tượng lớn yêu cầu sự đồng bộ giữa các `Java thread`, mặc dù JVM dùng một hệ thống `caches` của các khối rỗng khác kích thước để giảm nhu cầu đồng bộ và tăng tốc độ cấp phát.

## 4. Tăng dung lượng Heap như thế nào?

Bằng cách thêm tham số khi chạy ứng dụng, chúng ta có thể tăng/giảm dung lượng bộ nhớ Heap:

```
java -Xmx120m -Xms30m -Xmn10m -XX:PermSize=20m -XX:MaxPermSize=20m -XX:+UseSerialGC -jar
Java2Demo.jar
```

Trong đó:

- Xms: Dung lượng bộ nhớ được cấp phát khi JVM được start up
- Xmx: Dung lượng bộ nhớ tối đa mà JVM được cấp phát
- Xmn: Kích thước của không gian heap Young generation.
- XX:PermSize, XX:MaxPermSize: Dung lượng bộ nhớ cấp phát và bộ nhớ tối đa của Permanent generation

Kích thước mặc định của không gian Heap trong Java là 128MB trên hầu hết JVM của 32 bit nhưng nó rất khác nhau từ JVM đến JVM.

Ví dụ: Mặc định tối đa và kích thước heap bắt đầu cho Hệ điều hành Solaris 32 bit (Phiên bản nền tảng SPARC) là -Xms = 3670K và -Xmx = 64M và giá trị mặc định của các tham số kích thước heap trên hệ thống 64 bit đã tăng lên khoảng 30%.

Ngoài ra, nếu bạn đang sử dụng trình thu gom rác thông lượng trong Java 1.5 kích thước heap tối đa mặc định của JVM sẽ là bộ nhớ vật lý / 4 và kích thước vùng heap ban đầu mặc định sẽ là bộ nhớ vật lý / 16. Một cách khác để tìm kích thước heap mặc định của JVM là khởi động một ứng dụng với các tham số heap mặc định và theo dõi bằng cách sử dụng JConsole có sẵn trên JDK 1.5 trở đi, trên tab VMSummary, bạn sẽ có thể thấy kích thước heap tối đa.

Nhân tiện, có thể tăng kích thước không gian heap java dựa trên nhu cầu ứng dụng của bạn và tôi luôn khuyến nghị điều này để tránh sử dụng các giá trị heap JVM mặc định, nếu ứng dụng của bạn lớn và nhiều đối tượng được tạo, bạn có thể thay đổi kích thước của vùng heap bằng cách sử dụng các tùy chọn JVM -Xms và -Xmx. Xms biểu thị kích thước bắt đầu của Heap trong khi -Xmx biểu thị kích thước tối đa của Heap trong Java.

Có một tham số khác gọi là -Xmn biểu thị Kích thước của thế hệ Heap Java mới. Chỉ có điều là bạn không thể thay đổi kích thước của Heap trong Java một cách linh hoạt, bạn chỉ có thể cung cấp tham số Kích thước Heap Java trong khi bắt đầu JVM. Tôi đã chia sẻ một số tùy chọn JVM hữu ích hơn liên quan đến không gian Heap Java và bộ sưu tập Rác trên bài đăng của tôi 10 tùy chọn JVM mà lập trình viên Java phải biết, bạn có thể thấy hữu ích.

## 10 điểm về không gian heap Java

1. Bộ nhớ Heap Java là một phần của bộ nhớ được hệ điều hành cấp cho JVM.
2. Bất cứ khi nào chúng ta tạo các đối tượng, chúng được tạo bên trong Heap của Java.
3. Không gian Heap Java được chia thành ba vùng hoặc thể hệ cho mục đích thu gom rác được gọi là Thế hệ mới (New Generation), Thế hệ cũ (Old Generation) hoặc Thế hệ cho thuê (Rental Generation) hoặc Không gian Perm (Perm Space). Tạo vĩnh viễn là rác được thu thập trong toàn bộ máy chủ trong điểm nóng JVM.
4. Bạn có thể tăng hoặc thay đổi kích thước của không gian Heap Java bằng cách sử dụng các tùy chọn dòng lệnh JVM -Xms, -Xmx và -Xmn. Đừng quên thêm từ "M" hoặc "G" sau khi chỉ định kích thước để biểu thị Mega hoặc Gig. Ví dụ: bạn có thể đặt kích thước java heap thành 258MB bằng cách thực hiện theo lệnh `java -Xmx256m HelloWorld`.
5. Bạn có thể sử dụng JConsole hoặc `Runtime.maxMemory()`, `Runtime.totalMemory()`, `Runtime.freeMemory()` để truy vấn các kích thước lập trình Heap trong Java.
6. Bạn có thể sử dụng lệnh "jmap" để lấy kết xuất Heap trong Java và "jhat" để phân tích kết xuất heap đó.
7. Không gian Heap Java khác với Stack được sử dụng để lưu trữ phân cấp cuộc gọi và các biến cục bộ.
8. Trình thu gom rác Java chịu trách nhiệm khôi phục bộ nhớ từ các đối tượng chết và trở về không gian Heap Java.
9. Nếu bạn gặp lỗi `java.lang.OutOfMemoryError`, đôi khi đó chỉ là vấn đề tăng kích thước heap, nhưng nếu nó bị lặp lại, hãy tìm nguyên nhân gây rò rỉ bộ nhớ trong Java.
10. Sử dụng các công cụ Phân tích kết xuất Profiler và Heap để hiểu không gian Heap Java và dung lượng bộ nhớ được phân bổ cho từng đối tượng.

---

Revision #3

Created 2 November 2019 17:19:25 by Laptrinh.vn

Updated 3 November 2019 15:29:27 by Laptrinh.vn