

Home

▼ Cuộc sống

[REVIEW] Nhìn lại 15 năm lập trình
Từng bước để trở thành một lập trình viên giỏi

▼ câu hỏi java

techlead

▼ Cấu trúc dữ liệu & giải thuật

1. Tìm hiểu Cấu trúc dữ liệu #1: “Chết vì thiếu hiểu biết”

▼ Design Pattern

<http://www.java2s.com/>

▼ Grails

100. Tìm hiểu về Grails
Create a REST API with Grails 2.3.x (Part 1)
Customize Properties from RESTful Responses and Improve the REST Service API (Part 3)
Exclude Properties from RESTful Responses and Improve the RESTService API (Part 2)
Log Raw HTTP Requests/Responses for better Debugging of the REST API (Part 4)

▼ Java

Các loại biến trong Java
<http://javadevelopervietnam.blogspot.com>
MyBatis - Một Java persistence framework hữu ích nên biết ngoài Hibernate
Nói về biến volatile trong Java
SERIES JAVA NHỮNG ĐIỀU CÓ THỂ BẠN ĐÃ BIẾT
String và một số phương thức của lớp String trong Java
Sự khác nhau giữa bộ nhớ Heap và Stack trong Java

▼ Java swing

How to use JDatePicker to display calendar component
JList basic tutorial and examples

▼ Java-core

<http://www.dineshonjava.com/search/label/...>
<http://www.wideskills.com/java-tutorial>
Interface in Java
Java Nested Classes – java inner class, static nested class, local inner class and anonymous inner class
Java String format Example
Thread trong Java

▼ JQuery

Tùy biến giao diện form với jQuery plugin: Uniform

▼ Maven

1. Maven là gì?
10. Tìm hiểu maven kipalog
2. Cài đặt maven
3. File POM
4. Maven Plugins
5. Tạo Maven Project

[Java](#) >

Sự khác nhau giữa bộ nhớ Heap và Stack trong Java

posted Aug 15, 2016, 8:07 PM by Movies Clip

Quản lý bộ nhớ trong lập trình Java là kiến thức cơ bản nhưng rất quan trọng, chúng ta sẽ thường xuyên gặp những rắc rối về quản lý bộ nhớ trong ứng dụng java nếu không quản lý tốt nó. Trong Java, chúng ta nên hiểu định nghĩa và khái niệm hoạt động của bộ nhớ Heap và Stack

Bộ nhớ 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

Bộ nhớ Stack

Bộ nhớ Stack được dùng để thực thi một luồng(thread) nào đó. Bộ nhớ này chứa thể hiện cụ thể các phương thức(các thể hiện này có thời gian sống ngắn) và các tham chiếu tới các đối tượng trong bộ nhớ heap qua các thể hiện phương thức trên. Bộ nhớ Stack hoạt động theo hình thức LIFO, khi một phương thức được gọi, một ngăn xếp(block) trong bộ nhớ Stack được tạo để chứa những dữ liệu nguyên thủy và tham chiếu đến các đối tượng khác được sử dụng trong phương thức. Ngay khi kết thúc phương thức ngăn xếp sẽ được giải phóng để phương thức khác sử dụng. Dung lượng của bộ nhớ Stack rất nhỏ so với bộ nhớ Heap

Chúng ta hãy tham khảo đoạn code dưới để các sử dụng bộ nhớ Heap và Stack

Mô:

```
package com.memory.test;

public class Memory {

    public static void main(String[] args) { // Line 1
        int i=1; // Line 2
        Object obj = new Object(); // Line 3
        Memory mem = new Memory(); // Line 4
        mem.foo(obj); // Line 5
    } // Line 9

    private void foo(Object param) { // Line 6
        String str = param.toString(); //// Line 7
        System.out.println(str);
    } // Line 8

}
```

Hình vẽ dưới mô tả bộ nhớ Stack và Heap mà đoạn code trên sử dụng để chứa dữ liệu nguyên thủy, đối tượng và biến tham chiếu

6. Build Maven Project
7. Maven dependence
9. Cấu hình maven với eclipse
How do I add a project as a dependency of another project?
Maven – Cấu hình và tích hợp Eclipse IDE
Using Maven within the Eclipse IDE - Tutorial

PHP-NodeJS

▼ Servlet

Cơ bản về servlet, servlet container
How do servlets work? Instantiation, shared variables and multithreading
web.xml Servlet Configuration

▼ Spring

1. Hiểu về Dependency Injection
10. Spring - Injecting Collection
11. Spring - Beans Auto-Wiring
12. Bean autowiring sử dụng @Autowired annotation
13. Spring - Java Based Configuration
14. AOP with Spring Framework
15. AOP with Spring Framework (P2)
2. IOC và DI
3. Architecture
4. Spring framework Hello World
5. Spring - IoC Containers Khởi tạo các đối tượng trong khung chứa của Spring
6. Spring beans là gì?
7. Spring - Bean Scopes
8. Spring - Dependency Injection
9. Spring - Injecting Inner Beans
Các file cấu hình (config) trong Spring MVC
how to send email in spring framework
Import/Read excel file 2003 or 2007 with Spring MVC
Phân quyền đơn giản trong Spring MVC với Spring Interceptor và Java Annotation
Spring Framework Tutorial with Example Projects
SpringAOP - example
Tìm hiểu về Spring
Using Spring Handler Interceptors to Decouple Business Logic

▼ spring security 4

Document

▼ SQL

1. Cơ chế đánh index của database
Các khái niệm RDBMS trong SQL
Các store trong project
Lệnh trong SQL

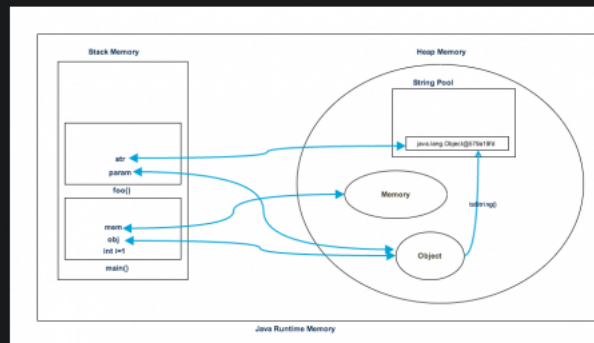
▼ Web design

Untitled Post

▼ webservices

Untitled Post

Sitemap



Để hiểu hơn chúng ta duyệt đoạn code trên chạy theo từng dòng

code trên, Java Runtime tải tất cả các lớp thực thi vào bộ nhớ Heap.

Khi thực thi đoạn

- Ở dòng 1, khi tìm thấy phương thức main(), Java Runtime tạo bộ nhớ stack cho luồng thực thi phương thức main()
- Dòng 2, dữ liệu nguyên thủy được tạo và sẽ được lưu trong bộ nhớ stack của phương thức main()
- Dòng 3, tạo đối tượng Object, nó sẽ được tạo trong bộ nhớ Heap và bộ nhớ stack chứa tham chiếu tới đối tượng đó
- Dòng 4, quá trình tương tự diễn ra như dòng 3 khi tạo đối tượng Memory.
- Dòng 5, khi gọi phương thức foo(), Java Runtime tạo mới ngăn xếp trong bộ nhớ stack để dùng cho phương thức foo(). Khi giá trị được truyền vào phương thức, một tham chiếu mới
- đến đối tượng Object trong ngăn xếp dành cho phương thức foo() ở dòng 6
- Dòng 7, một chuỗi được tạo trong String Pool trong bộ nhớ Heap và tạo một tham chiếu trong ngăn xếp của foo() tới đối tượng đó
- Dòng 8, kết thúc phương thức foo(), lúc này ngăn xếp dành cho foo() được giải phóng.
- Dòng 9, kết thúc phương thức main() và ngăn xếp dành cho main() cũng được giải phóng và kết thúc chương trình. Vì thế, Java Runtime giải phóng tất cả bộ nhớ mà chương trình dùng

Sự khác nhau giữa bộ nhớ Heap và Stack

Những giải thích trên giúp chúng ta dễ dàng phân biệt giữa bộ nhớ Stack và Heap, chúng ta sẽ tóm tắt lại theo các điểm sau

1. Bộ nhớ Stack chỉ được dùng khi một luồng(thread) thực thi, ngược lại bộ nhớ Heap được dùng trong tất cả các trường hợp sử dụng bộ nhớ
2. Khi một đối tượng được tạo, nó luôn được tạo trong bộ nhớ Heap và bộ nhớ stack sẽ chứa tham chiếu tới đối tượng đó. Bộ nhớ Stack chỉ chứa biến dữ liệu nguyên thủy và biến tham chiếu tới các đối tượng trong bộ nhớ heap.
3. Với bộ nhớ Heap, chúng ta có thể truy cập các đối tượng một cách toàn cục(globally), ngược lại với bộ nhớ stack chúng ta không thể truy cập các đối tượng thuộc luồng(thread) khác
4. Bộ nhớ Stack hoạt động theo hình thức LIFO, ngược lại quản lý bộ nhớ trong Heap phức tạp hơn vì nó được dùng ở phạm vi toàn cục.
5. Bộ nhớ Stack tồn tại trong thời gian ngắn, nhưng bộ nhớ Heap tồn tại từ lúc ứng dụng bắt đầu thực thi đến lúc kết thúc
6. Để định nghĩa kích thước tối thiểu và tối đa cho bộ nhớ Heap dùng tùy chọn -Xms và -Xmx, còn bộ nhớ stack dùng -Xss
7. Khi bộ nhớ stack đầy JRE ném ra ngoại lệ java.lang.StackOverflowError, còn bộ nhớ heap đầy nó ném ra ngoại lệ java.lang.OutOfMemoryError
8. Kích thước bộ nhớ stack nhỏ hơn nhiều với bộ nhớ Heap và tốc độ truy cập bộ nhớ nhanh hơn so với bộ nhớ Heap

Nhận xét

Bạn không có quyền thêm nhận xét

Bạn không có quyền thêm hình ảnh.

[Đăng nhập](#) | [Hoạt động gần đây của trang web](#) | [Báo cáo lạm dụng](#) | [Trang tin](#) | Được cung cấp bởi [Google Sites](#)

