

Java Heap - Stack memory

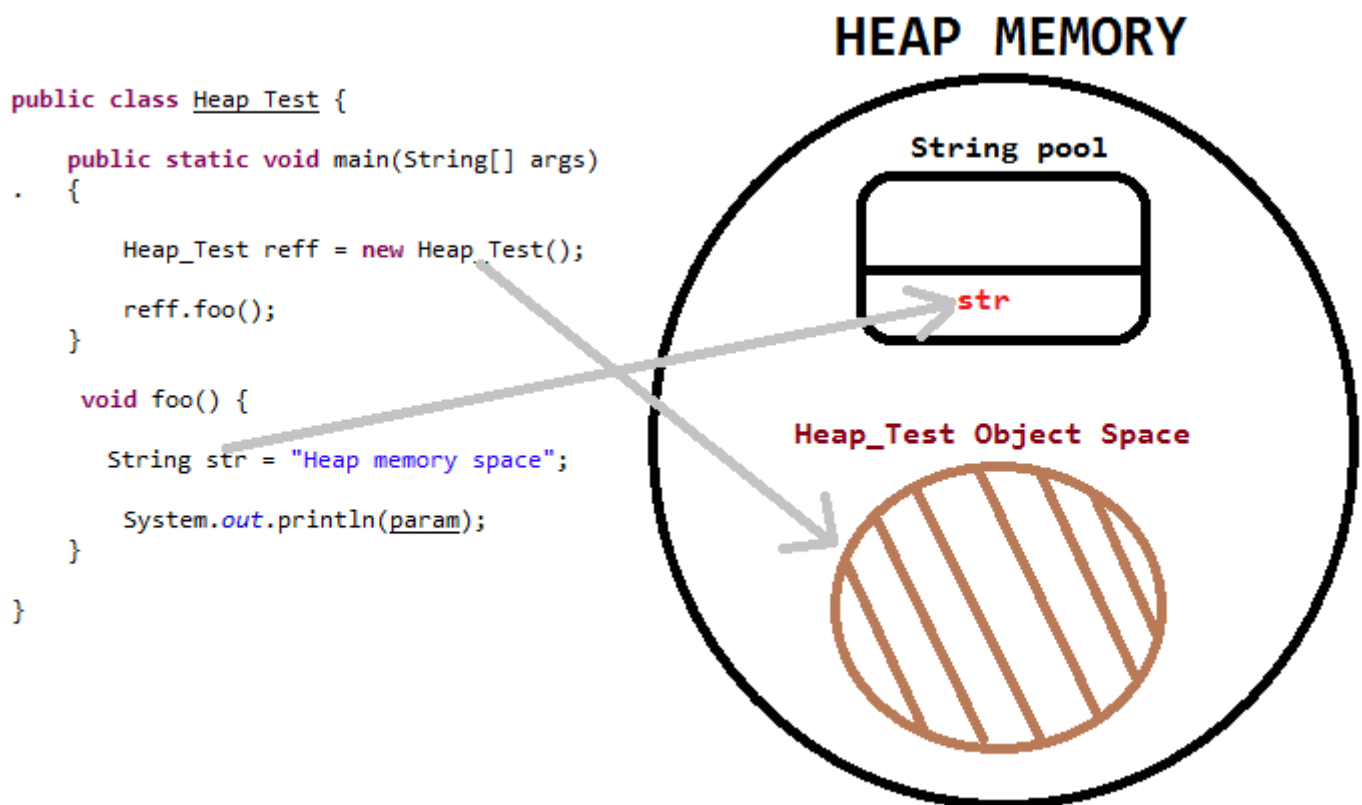
Phân biệt cách hoạt động và cấp phát của Heap và Stack memory trong Java

Xem lại:

- [Heap memory](#)
- [Stack memory](#)

1. Heap memory

Heap là một vùng nhớ trong bộ nhớ được sử dụng để lưu trữ các đối tượng khi từ khóa new được gọi ra, các biến static và các biến toàn cục (biến instance).

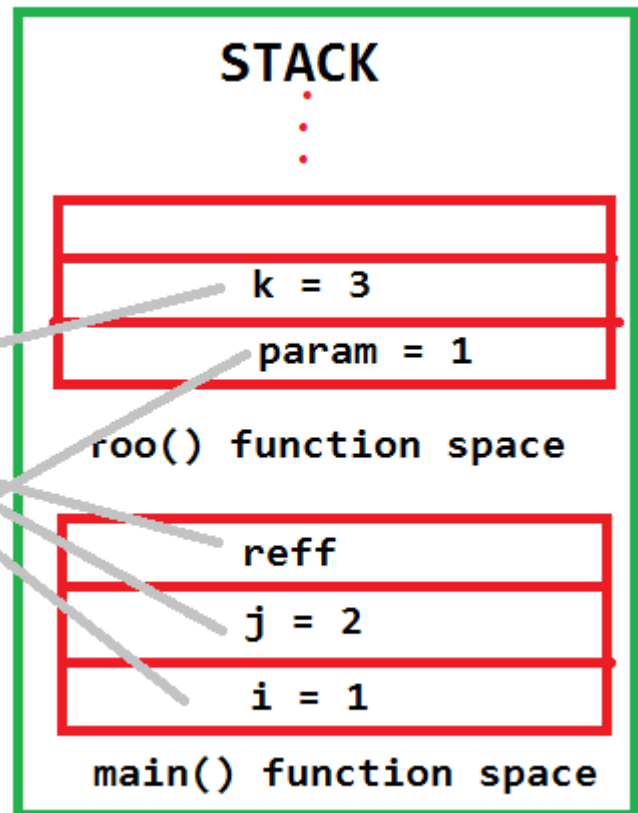


2. Stack memory

Stack là một vùng nhớ được sử dụng để lưu trữ các tham số và các biến local của phương thức mỗi khi một phương thức được gọi ra. Các tham số và các biến local của một phương thức tạo thành một bản ghi kích hoạt, còn được gọi là một stack frame. Các bản ghi kích hoạt được đẩy vào một stack khi phương thức được gọi và đẩy ra khỏi stack khi phương thức trả về. Sự tồn tại tạm thời của

các biến này quyết định thời gian sống của các biến.

```
public class Stack_Test {  
    public static void main(String[] args) {  
        int i=1;  
        int j=2;  
        Stack_Test reff = new Stack_Test();  
        reff.foo(i);  
    }  
    void foo(int param) {  
        int k = 3;  
        System.out.println(param);  
    }  
}
```



3. Heap and Stack memory

Chương trình sau là ví dụ cách quản lý và cấp phát bộ nhớ Heap và Stack trong Java

```
public class Heap_Stack {  
  
    //main() method thread creates space in stack memory  
    public static void main(String[] args) {  
  
        // primitive datatype created inside main() method space in stack memory  
        int i=1;  
  
        // Object created in heap memory and its reference obj in stack memory  
        Object obj = new Object();  
  
        // Heap_Stack Object created in heap memory and its reference objnew in stack  
        memory  
        Heap_Stack objnew = new Heap_Stack();  
    }  
}
```

```

        // New space for foo() method created in the top of the stack memory
        objnew.foo(obj);

    }

    private void foo(Object p) {

        // String for p.toString() is created in String Pool and reference str created in
        stack memory
        String str = p.toString();

        System.out.println(str);
    }
}

```

Step 1. Khi chạy chương trình, một thread sẽ khởi tạo và sẽ gọi hàm main ở dòng 1. Một khối bộ nhớ được tạo trong stack cho hàm main().

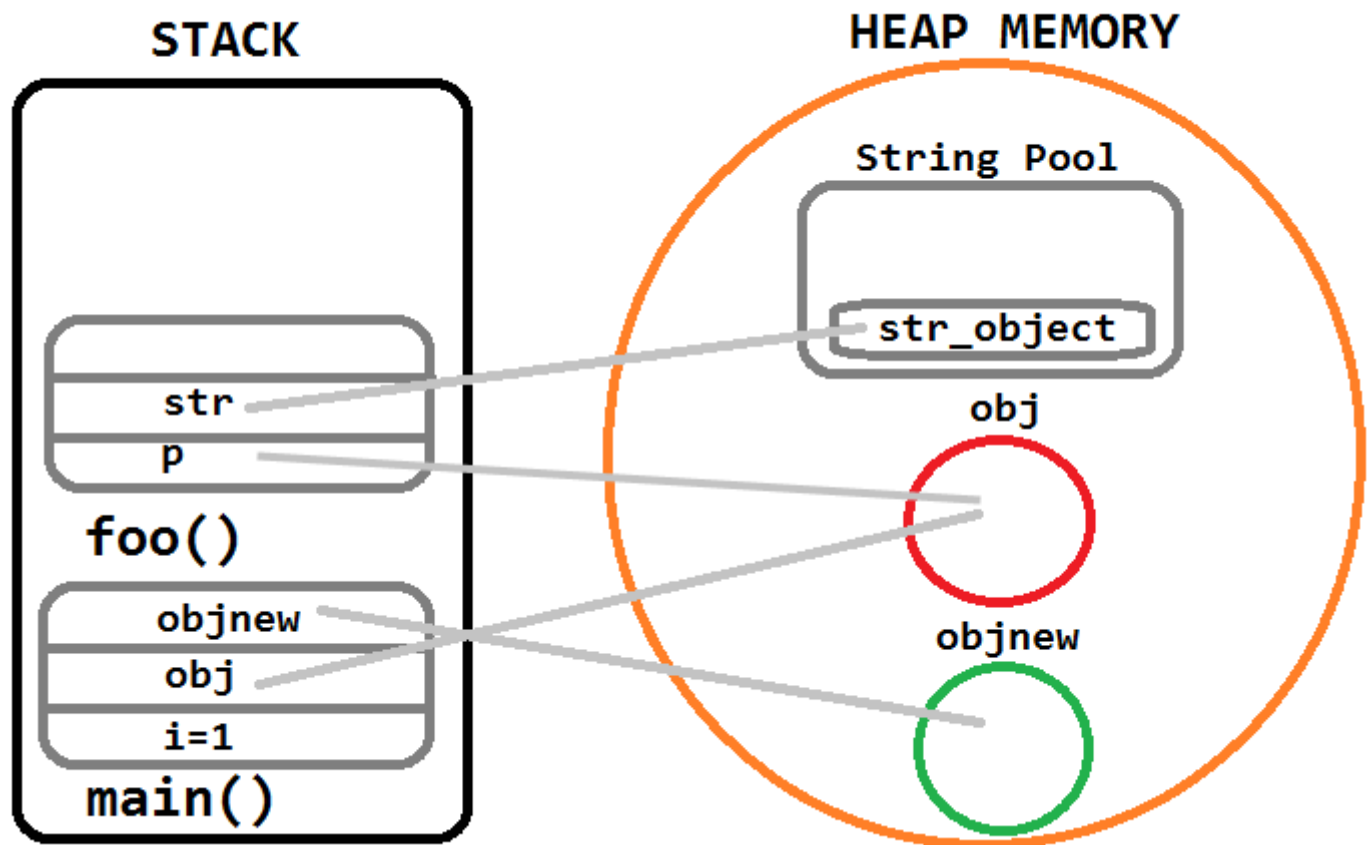
- `int i=1` : Một biến local được tạo, loại primitive được lưu trong cùng khối bộ nhớ của hàm main()
- `Object obj = new Object()` : một đối tượng được tạo loại Object sẽ được lưu trong bộ nhớ Heap và biến tham chiếu `obj` được lưu trong Stack của hàm main()
- `Heap_Stack objnew = new Heap_Stack()` : một đối tượng được tạo loại Object sẽ được lưu trong bộ nhớ Heap và biến tham chiếu `objnew` được lưu trong Stack của hàm main()

Step 2. Hàm `foo()` thì được gọi, vì vậy nó sẽ tạo một khối mới bộ nhớ trong stack cho hàm `foo()`

- `Object p` : Một biến local `p` được tạo (Giá trị được truyền vào), loại tham chiếu được lưu trong cùng khối bộ nhớ của hàm `foo()`
- `String str=p.toString()` : Một biến String được tạo được lưu trong String Pool, biến tham chiếu `str` được lưu trong vùng nhớ tiếp theo của stack `foo()`
- Hàm `foo()` sẽ kết thúc sau khi hàm `System.out.println()` được thực thi, vì vậy bộ nhớ trong stack cho hàm `foo()` sẽ được giải phóng.

Step 3. Theo quy luật LIFO, `foo()` vào sau chết trước, và sau đó hàm `main()` cũng kết thúc, bộ nhớ trong stack cho hàm `main()` cũng được giải phóng.

Step 4. Chương trình kết thúc.



4. Heap vs Stack memory

Sự khác nhau Java Heap và Stack memory

#	Heap Memory	Stack Memory
1	Java Heap Memory là bộ nhớ được sử dụng ở runtime để lưu các Objects. Bất cứ khi nào ở đâu trong chương trình của bạn khi bạn tạo Object thì nó sẽ được lưu trong Heap (thực thi toán tử new).	Stack Memory là bộ nhớ để lưu các biến local trong hàm và lời gọi hàm ở runtime trong một Thread java. Các biến local bao gồm: loại nguyên thủy (primitive), loại tham chiếu tới đối tượng trong heap (reference), khai báo trong hàm, hoặc đối số được truyền vào hàm.
2	Thời gian sống của bộ nhớ Heap dài hơn so với Stack. Thời gian sống của object phụ thuộc vào Garbage Collection của java. Garbage Collection sẽ chạy trên bộ nhớ Heap để xoá các Object không được sử dụng nữa, nghĩa là object không được referece trong chương trình.	Thường có thời gian sống ngắn.
3	Các objects trong Heap đều được truy cập bởi tất cả các các nơi trong ứng dụng, bởi các threads khác nhau .	Stack chỉ được sử dụng cho một Thread duy nhất . Thread ngoài không thể truy cập vào được.

4	Cơ chế quản lý của Heap thì phức tạp hơn. Heap được phân làm 2 loại Young-Generation, Old-Generation. Đọc thêm về Garbage Collection để hiểu rõ hơn.	Cơ chế hoạt động là LIFO (Last-In-First-Out), chạy sau chết trước.
5	Dung lượng Heap thường lớn hơn Stack.	Bộ nhớ stack thường nhỏ.
6	Sử dụng -Xms và -Xmx để định nghĩa dung lượng bắt đầu và dung lượng tối đa của bộ nhớ heap.	Dùng -Xss để định nghĩa dung lượng bộ nhớ stack.
7	Khi Heap bị đầy chương trình hiện lỗi <code>java.lang.OutOfMemoryError</code> : Java Heap Space	Khi stack bị đầy bộ nhớ, chương trình phát sinh lỗi: <code>java.lang.StackOverflowError</code>
8	Truy cập vùng nhớ Heap chậm hơn Stack.	Truy cập stack nhanh hơn Heap
9	Dung lượng sử dụng của Heap sẽ tăng giảm phụ thuộc vào Objects sử dụng.	Bất cứ khi nào gọi 1 hàm, một khối bộ nhớ mới sẽ được tạo trong Stack cho hàm đó để lưu các biến local. Khi hàm thực hiện xong, khối bộ nhớ cho hàm sẽ bị xoá, và giải phóng bộ nhớ trong stack.

Revision #3

Created 3 November 2019 15:57:42 by Laptrinh.vn

Updated 3 November 2019 16:18:20 by Laptrinh.vn