

劉逸的留意世界

紀錄IT、趨勢的點點滴滴

[相簿](#) | [部落格](#) | [留言](#) | [名片](#)

⚡ 來收藏有興趣的內容吧！

Heap: 執行時期儲存兩大要角

Aug 01 Mon 2011 11:42

分享: [f](#) [p](#) [t](#) [Like 118](#)

現代電腦系統大多依照Von Neumann Architecture設計而成，其中一特色stored programming乃指『程式執行一定要將欲執行的指令跟資料放入記憶體方可執行』，由此可知執行過程中記憶體所佔的地位厥偉之處。但許多工程師卻搞不清楚記憶體中的stack跟heap space到底有何居別，下面簡單針對兩者加以論述，希望對讀者有所幫助～

三分天下。程式執行過程中其實主要分成三大區塊：**global**、**stack**、**heap**三塊。其中global區塊最易理解，主要存放全域變數或宣告為**static**的靜態變數在此就不多做贅述；另外兩個區塊分別為stack跟heap這兩者往往混淆不清，尤其在java中有時候會出現stack overflow或heap overflow到底兩者差異在哪，若工程師連這都不清楚那以後怎麼去調整JVM中的heap memory space跟stack memory space的大小呢？

貼心的系統全自動化管理區塊：Stack Memory Space！在記憶體中不外乎就是要存放變數、函式相關資訊等資料，使運作過程可以順利取得所需的變數或函式所在地。**要讓系統可以全自動化管理，代表需可被預期此變數或函數資訊的生命週期，一旦完全可預測代表可以安心的交由系統管理，這些資訊也將在執行過程中被存放在stack空間。**Stack中常見的存放資訊如下：區域變數(local variable)、函式參數(function/method parameter)、函數的返回位址(function/method return address)等資訊。為何上述資訊會放於stack之中，簡單來說：

```
void method1() {
```

個人資訊

[+ 關注](#)

暱稱：劉逸

分類：[數位生活](#)好友：共0位 ([看全部](#))

我的好友

沒有資料可以顯示


```
int x = 100;
```

```
}
```

上述的`int x = 100`，系統會在`stack`中找一個區塊給`x`，另外裡面的內容為`100`。然而，`x`會被放入`stack`主要是因為在編譯時期系統已經可以預知`x`從何時開始配置跟何時結束回收(當然就是看所屬`block`結束就跟著回收)，由於配置跟回收的規則明確，當然就往`stack`擺囉。

在舉一例子：

```
void method2() {
```

```
    method1();
```

```
}
```

上述當呼叫`method1()`時，系統會先把`method2`的返回位址存到`stack`當中，為何是存放在`stack`呢，因為函式的呼叫有後進先出的概念，當`method1()`被呼叫而開始執行，待結束時必定會查找該返回何處，故最後一定會讀取函式的返回位址，既然如此明確而有條理，當然也是往`stack`放！

可預測性外加後進先出的生存模式，令`stack`無疑是最佳的存放策略。由於程式語言中變數跟函式的生命週期皆為後進先出的概念，也就是越晚產生的會越先被回收或銷毀。正因如此只要是可預測性的相關資訊都是往`stack`存放。此外，由於**`stack`中的資料之存活週期規律故由系統自行產生與回收其空間即可，就不勞工程師們費心啦！**

天啊！程式中竟然有不可預測其存活時間的資料存在。在程式中，有部分的需求總是在執行中依據實際情況才會動態增減，這些資訊是難以被預測哪時候開始有？量有多少？何時該回收？...**這些不可預測的因素造成上述的`stack`區塊不適合運用於此**。當資訊為動態配置產生，系統會存放在另外一塊空間，稱之為**『Heap』(注意這裡的Heap跟資料結構中的Heap不相關，可別會錯意！)**。Heap的區塊專收執行期間動態產生的資料，由於為動態產生故結束點無法由系統來掌握，故需使用者自行回收空間。在C++或Java中利用`new`語法產生的就是動態配置的物件，需存放於`heap`中。

奇怪跑越久記憶體用越多的怪現象。**許多時候執行的程式都沒有改變，但卻常出現隨時間執行越久程式所耗用的空間將越多，最後造成`out of memory`**。工程師也不知為何如此，就是定期在**`out of memory`之前`restart`程式即可**。這中現象層出不窮，一般大多是因為工程師沒有正確將記憶體回收所導致。Heap中的資料如果沒有正常的回收，將會逐步成長到將記憶體消耗殆盡，下次發生上述問題的實後，切記自己檢查一下

熱門文章

[\(103915\)stack vs heap: 執行時期儲存兩大要角](#)

[\(58663\)身處IT產業...30歲後才發現的事part3～四位coding天才](#)

[\(28903\)論物件導向part 8: Why Overloading、Overriding](#)

[\(27022\)Framework之我見](#)

[\(24387\)論物件導向part 5: Polymorphism](#)

文章分類



[臥龍邏輯思考力 \(19\)](#)

[臥龍論IT \(87\)](#)

[臥龍雜記 \(93\)](#)

[臥龍點將錄 \(2\)](#)

[全文分享區 \(1\)](#)

最新文章

[平行時空](#)

[當個不留戀的甲方...夠瀟灑的乙方](#)

[第一次幫客戶寫的文案](#)

[你呢?有留18分鐘給自己嗎?](#)

[勇於進場然後:面對,要球,對決](#)

[值得細細品味...白日夢冒險王\(The Secret Life of Walter Mitty\)](#)

[Rain man. One for bad, two for good.](#)

[他不笨,他是我爸爸\(I am sam\)](#)

heap空間的資料有無正常回收。論述到此有些讀者可能會覺得納悶：為何在寫Java都不需要注意回收空間的問題？～答案是因為**Java**中會採用**Garbage Collection**(垃圾回收)的機制自動檢查**Heap**中哪些資料已經沒有被使用，當確認資料已經沒有使用會自動將空間回收，如此工程師就專注撰寫程式即可，不用擔心記憶體回收不當等問題。

The conclusion is...。當產生stack overflow一般是因為過多的函式呼叫(例如：遞迴太深)、或區域變數使用太多，此時請試著將stack size調大一點，另外檢查看看函式的呼叫跟變數的使用量。反之，當發生heap overflow請檢查是否都有正確將heap space的資料回收，另外採行的動態配置是否合理，不要過渡濫用而new出無謂的空間，若真的是程式過於複雜造成，請將heap size調大一些。

程式的撰寫，其實跟製造業的加工廠很相似，套句郭董常說的：『魔鬼藏在細節中』～～

全站熱搜

[里長台灣鯛](#) [小豆廊](#) [益銘號自製手打麵](#) [紫微斗數](#) [芒果冰](#) [粽子](#)

創作者介紹



劉逸

劉逸的留意世界

+ 關注

劉逸 發表在 [痞客邦](#) 留言(16) 人氣(103920)

[E-mail轉寄](#)

[曾幾何時...改變或被改變?](#)

[借用一下A的N次方理論](#)

最新留言

07/30 chesterdesigner:
六年過去了，請問作者對於

03/05 nn:
關於第三點...我覺得如果能

12/09 路人甲:
您的比喻真棒

08/24 訪客:
好有道理 哈哈

08/15 訪客:
一昧(ㄇㄟˋ) 釋義: 1.糊塗

文章精選

文章精選 ▼

所有文章列表

文章搜尋

搜尋

新聞交換(RSS)

[PIXNET](#) [RSS](#)

[PIXNET](#) [ATOM](#)

[REPLY](#) [RSS](#)

誰來我家



參觀人氣

本日人氣: 37

累積人氣: 503789

全站分類: [數位生活](#)

個人分類: [臥龍論IT](#)

此分類上一篇: [論物件導向part 5: Polymorphism](#)

此分類下一篇: [論物件導向part 6: abstract class](#)

上一篇: [論物件導向part 5: Polymorphism](#)

下一篇: [一直在想](#)

留言列表 (16)

發表留言

#1  低調粉絲 於 2011/08/03 01:45



將看似複雜的事情，以簡單的方式做說明
給臥龍一百個讚～

#2  訪客 於 2012/03/22 22:55



獲益良多，感謝大大

QR Code



POWERED BY




[\(登入\)](#)

#3  kenny 於 2012/08/05 00:08



謝謝老師
終於搞清楚了
獲益良多!!

#4  小肥 於 2013/01/09 17:16



謝謝老師，你說的資料結構HEAP是堆積樹嗎？你說不相干是說系統HEAP並非使用堆積原理，而是需要解構、回收


這裡的HEAP跟資料結構的HEAP一點關係都沒有，只是剛好也叫做HEAP

劉逸 於 2013/01/10 00:29 回覆

#5  particle71982  於 2013/03/08 15:10




有學到給讚!!

#6  訪客 於 2013/03/31 17:33



這些資料簡單易懂
感謝作者講解

#7  訪客 於 2013/05/28 10:38



您好, 請問在<資料結構精華導讀>一書中有"Recursive vs Non-recursive之比較"一表:
Recursive執行需要額外動態的空間(system stack); Non-recursive不需要額外的stack memory.

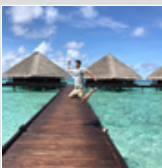
但是, 由此篇說明, 動態產生的記憶體, 應該屬於heap才對? 還請幫忙解除迷惑, 謝謝!

stack跟heap都是動態的記憶體管理。只是一個是系統可辨識的放在stack中。一個是設計師自己在程式過程跟記憶體要的, 放在heap中。

以上...^^

劉逸 於 2013/05/28 15:09 回覆

#8  黃昭維 於 2014/10/14 17:26




有看有推 講解的非常清楚!!!!

#9  小汀  於 2015/03/12 18:19



很棒的分享喔
值得推薦給朋友們知道

#10  真棒 於 2015/06/02 15:57



講解的真棒

#11  phoebelin0606  於 2015/09/13 11:42



謝謝您淺而易懂得分享

#12  Una Super 於 2016/01/29 16:45



老師你好,

謝謝老師的分享,受益良多~
另外有個地方不太明白,GC的部分,
有GC就不用自己管理heap,交給GC就可以了
這樣理解對嗎?

Una

#13  訪客 於 2016/09/09 14:57



謝謝老師

#14  wangdu.zeus 於 2017/07/14 14:18



大大講解的真好，受益良多！

#15  阿達 於 2017/08/10 14:21

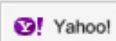
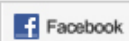


謝謝，說明的好清楚。

#16  阿星 於 2018/06/07 10:01



受益良多~~~ 感謝大大!! 新手真的好需要基本觀念!!!



您尚未登入，將以**訪客**身份留言。亦可[登入](#)留言

您的暱稱 ...

留個言吧 ...

☐ 悄悄話

[+ 其他選項](#)

送出留言