معركة الـ Stack و Heap

انا هخرج برة الشرح التقليدي ونبص على الموضوع من منظور الـ **CLR** ومتعمق شوية.

تعال نتخيل الذاكرة (RAM) ومقسمة لمنطقتين رئيسيتين بيشتغلوا مع بعض بس بقوانين مختلفة تماماً.

أولاً: الـ Stack

الـ Stack هو ذاكرة سريعة جداً، وصغيرة نسبياً

1. الهيكلة (LIFO - Last In, First Out)

تخيلها زي "رصة أطباق". الطبق اللي بتحطه آخر واحد، هو أول واحد لازم تشيله. مينفعش تسحب طبق من النص وإلا الرصة تقع.

- الـ Stack بيشتغل بمؤشر واحد بس مفيش Allocation معقد ولا بحث في الذاكرة.
- ترتيب البيانات في الـ Stack **ثابت ومعلوم وقت الـ Compile**
- مفيش GC على الـ Stack انتهاء الـ Scope هو نهاية العمر لأي قيمة عليه.
- الـ Stack مش مكان "تخزين طويل الأجل"، هو مساحة تنفيذ مؤقتة مرتبطة بالكود.
- غالبا كل ما البيانات تبقى أقرب لـ Stack، كل ما الأداء يبقى أعلي

2. كيف يعمل (The Stack Frame)

في الـ .NET، لكل **Thread** الـ Stack الخاص بيه (عادة حجمه 1MB).

- لما بتنادي على Function (مثلاً `Main` تنادي `Calculate`)، الـ CPU بيعمل حاجة اسمها **Stack Frame**.
- الـ Frame ده صندوق بيتحط فيه:
  - الـ Parameters بتاعة الدالة.
  - الـ Return Address (عشان يعرف يرجع لمين لما يخلص السطر اللي بعد الفانكشن).
  - الـ Local Variables (المتغيرات اللي عرفتها جوه الدالة).
- مجرد ما الدالة تخلص (`return` أو `{` closing brace)، الـ Frame ده كله "بيترمي" أو يتم مسحه في لحظة وحدة (Pop). عشان كده هو **سريع جداً** في التنظيف (Self-cleaning).

ثانياً: الـ Heap

الـ Heap هو مساحة عشوائية كبيرة، مفتوحة، وأي حد يقدر يوصل لأي حاجة فيها لو معاه "العنوان" (Address/Reference).

لما بنقول "Heap" في كلامنا اليومي غالبًا بنقصد **GC Heap** لكن تقنيًا، في أكتر من Heap جوه الـ Process وهنعرف دا كمان شوية

1. الهيكلة (FIFO - First In, First Out)

تخيل الـ Heap زي مخزن أمازون ضخم. الكراتين بتتحط في أي مكان فاضي، وعشان توصل لكرتونة معينة لازم يكون معاك رقم الرف (Address) ورقم الرف بيكون متخزن في الـ Stack.

2. تقسيمات الـ Heap في الـ .NET (هنا العمق!)

الـ Heap مش مجرد "كومة" واحدة، الـ CLR بيقسمه لحتت عشان الـ Garbage Collector (GC) يعرف يشتغل بذكاء انتظروا ملخص الـ GC إن شاء الله:

- **الـ SOH (Small Object Heap):** ده المكان الطبيعي لأي Object حجمه صغير (أقل من 85,000 bytes).
  - **الـ Generations (الأجيال):** الـ GC بيقسم الـ SOH لـ 3 أجيال:
    - **الـ Gen 0:** أي حاجة لسه معمولة new بتدخل هنا. ده حجمه صغير وبيتم تنظيفه بسرعة جداً.
    - **الـ Gen 1:** لو الـ Object نجا من التنظيف في Gen 0، بيترقى لـ Gen 1 (منطقة عازلة).
    - **الـ Gen 2:** لو فضل عايش كمان، بيروح Gen 2. ده للكبار فقط (Long-lived objects) والـ GC بيمسح هنا بتقل وبطء لأنه مكلف.
- **الـ LOH (Large Object Heap):**
  - أي Object حجمه أكبر من **85 KB** (غالباً الـ Arrays الكبيرة والـ Strings الطويلة) بيروح هنا علطول.
  - **مشكلته:** الـ LOH مش بيتعمل له Compaction (رص من جديد) بسهولة زي الـ SOH، فممكن يحصل Fragmentation (فراغات في الذاكرة لانهم مش ورا بعض في فراغات بينهم).
- **الـ POH (Pinned Object Heap):** (ظهرت حديثاً في .NET 5+)
  - مخصصة للـ Objects اللي محتاجة تفضل مكانها ومتتحركش (Pinned) عشان الـ Native Code يقدر يقرأها.

مثال عملي (Code Walkthrough)

تعال نكتب كود ونشوف بيحصل إيه في الميموري خطوة بخطوة:

```
public void MyMethod()
{
    // 1. Value Type (Local)
    int age = 25;

    // 2. Reference Type
    Employee emp = new Employee();
    emp.Salary = 5000;
}
```

ماذا يحدث في الخلفية؟

1. تبدأ: الـ `MyMethod` ينشئ CLR **Stack Frame**.

```
int age = 25:
```

- بيحجز 4 بايت في الـ **Stack Frame** ويكتب فيهم `25`.

```
Employee emp = new Employee():
```

- كلمة `new` بتروح للـ **Heap** (تحديداً Gen 0) تحجز مكان لحجم الـ Employee وتنشئه.
- كلمة `emp` بتروح للـ **Stack** وتحجز مكان صغير (Pointer) تشيل فيه "عنوان" الـ Employee اللي في الـ Heap (مثلاً `0x5A12`).

1. نهاية الدالة `}`:
    - الـ **Stack Frame** كله بيطير (Pop).
    - المتغير `age` اختفى.
    - المؤشر `emp` اختفى.
    - **لكن!!** الـ Object بتاع `Employee` لسه موجود في الـ **Heap** بس محدش بيشاور عليه (Orphaned).
    - هنا ييجي دور الـ **Garbage Collector** بعدين، يلاقي الـ Object ده محدش عاوزه، فيقوم مسحه من الـ Heap.

ملخص المقارنة

وجه المقارنة	Stack	Heap
السرعة	صاروخ (مجرد تحريك مؤشر)	أبطأ (بحث عن مكان وحجز)
الإدارة	أوتوماتيك (OS/CPU)	GC بواسطة الـ managed
الحجم	محدود (StackOverflowException)	ضخم (OutOfMemoryException)
دورة الحياة	مرتبط بنطاق الدالة (Scope)	GC يعيش لحد ما يتنظف من الـ
التخزين	Value Types (غالباً) + Pointers	Reference Types + Boxed Value Types

تعال نتعمق شوية ونعرف حاجة جديدة Loader Heap

عشان تتخيلها صح تعال نبص على التقسيم ده:

1. الذاكرة الكبيرة (Process Memory)

لما بتشغل برنامج .NET، الـ Operating System بيدي للبرنامج مساحة كبيرة في الذاكرة. الـ CLR بيقسم المساحة دي لقطعتين كبار:

- **القطعة الأولى: الـ Managed Heap (أو الـ GC Heap)**
  - ده اللي فيه (SOH, LOH, POH).
  - **القانون:** أي حاجة بتتعمل بـ `new` بتروح هنا.
  - **المدير:** الـ Garbage Collector هو اللي بيقرر مين يعيش ومين يموت ومين يتنقل من جيل لجيل.
- **القطعة الثانية: الـ Loader Heap (أو الـ Unmanaged/System Heap)**
  - ده اللي فيه (HFH, LFH, Stub Heap).
  - **القانون:** ده مخزن "بيانات النظام" والـ Metadata.
  - **المدير:** الـ CLR مباشرة. الـ GC ملوش أي سلطة هنا، ولا يقدر يمسح منه حرف.

هل معني كدا الـ Loader Heap جزء من الـ Heap؟

الـ **Loader Heap** بنعتبره **Runtime Data Structures** و هو **Heap من ناحية الذاكرة**

لكن مش Heap من ناحية الـ Garbage Collector ؟ يعني اي !!

لو انت لسا فاكر احنا قولنا ان المدير بتاعه هو الـ CLR الـ GC ميقدرش يمسح منه حرف

وبالتالي لا يُعتبر جزءًا من الـ Heap في معركة Stack vs Heap.

إيه هو الـ HFH (High Frequency Heap)؟

- **موقعه:** هو جزء من حاجة اسمها **Loader Heap**.
- **وظيفته:** الـ CLR بيستخدمه عشان يخزن فيه الـ **Internal Data Structures** اللي هو محتاجها **بشكل متكرر**
- **وسريع جداً** (عشان كده اسمه High Frequency).
- **بيشيل إيه؟:** بيشيل حاجات زي الـ `MethodTables` والـ `Interface Maps` ليهم بوست إن شاء الله. الحاجات دي لازم تكون موجودة ومتاحة بسرعة البرق عشان الكود بتاعك يشتغل، ومينفعش الـ GC ييجي يمسحها بالغلط.
- **هل هو GC Managed؟ هنا بقا الأمر مختلف عن الـ Heap العادي هنا ميقدرش يتنظف من الـ GC**
  ده Unmanaged Memory بتديرها الـ CLR مباشرة، ومش بيتم تنظيفها غير لما الـ AppDomain كله يتقفل.

---

2. فين الـ "Static Heap"؟ (أين تعيش الـ Statics؟)

مفيش حاجة رسمية في الـ Documentation اسمها "Static Heap" ككيان مستقل، لكن ده مصطلح دارج بيوصف مكان تخزين المتغيرات الـ `static`.

عشان تفهمها صح، لازم تفرق بين "المتغير" و "القيمة":

لو كتبنا:

```
static class GlobalData {
    public static int Counter = 10;         // Value Type
    public static User Admin = new User();   // Reference Type
}
```

بيتحطوا فين؟

1. الـ MethodTable (في الـ High Frequency Heap):

   الـ CLR لما بيحمل الكلاس GlobalData، بيعمله MethodTable جوه الـ HFH.

2. **المكان الفعلي للبيانات:**

- الـ `Counter` (الـ 4 بايت بتوعه): بيتخزنوا مباشرة جوه الـ `MethodTable` أو في مكان قريب جداً منه في الـ **Loader Heap**. دول بيفضلوا عايشين طول عمر البرنامج.
- الـ `Admin` (الـ Reference/Pointer): المؤشر نفسه بيتخزن في الـ **Loader Heap**.
- **المفاجأة:** الـ Object نفسه (`new User()`) بيروح فين؟ **بيروح الـ GC Heap العادي (SOH)!**

> الخلاصة: الـ "Static Heap" هو مجرد جزء من الـ Loader Heap بيخزن "جذور" (Roots) المتغيرات الـ Static، لكن الـ Objects الكبيرة اللي الـ Static بيشاور عليها بتعيش في الـ Heaps العادية.

---

3. التحديث الجديد: The Frozen Heap (أحدث حاجة في .NET 8)

ظهر نوع جديد من الـ Heap اسمه **Frozen Object Heap (FOH)**.

- **المشكلة:** زمان، الحاجات الـ Static (زي `string` ثابتة أو Configuration) كانت بتعيش في الـ SOH وتترقى لـ Gen 2 وتفضل قاعدة هناك والـ GC كل شوية يعدي عليها يتأكد إنها لسه موجودة (وده بيضيع وقت الـ CPU).
- **الحل (Frozen Heap):** ده Heap خاص جداً، أي حاجة بتتحط فيه بتبقى **Immutable** (غير قابلة للتغيير) والـ GC **بيتجاهلها تماماً** (مش بيعملها Scan).
- **الاستخدام:** الـ .NET Runtime بيستخدمه دلوقتي عشان يحط فيه الـ String Literals والـ Static Readonly objects عشان يسرع الأداء جداً.

وأخيرا تجميعة الصورة الكبيرة داخل الـ Heap

1. **الـ Managed Heaps (تحت سيطرة الـ GC):**
   - **الـ SOH:** (Gen0, Gen1, Gen2) ← للشغل اليومي.
   - **الـ LOH:** للأحجام الكبيرة.
   - **الـ POH:** للحاجات المثبتة (Pinned).
   - **الـ FOH (Frozen):** للحاجات الثابتة للأبد (New in .NET 8).
2. **الـ Loader Heaps (تحت سيطرة الـ CLR - Internal):**
   - **الـ High Frequency Heap:** لـ MethodTables والبيانات الوصفية (Metadata) اللي بتتكرر كتير.
   - **الـ Low Frequency Heap:** لحاجات نادرة الاستخدام وشبه ثابتة.

لاتنسوني من دعائكم ❦

a7medsabrii