Monitorování alokací paměti za běhu Java aplikací

Jiří Velek

Vedoucí práce: Ing. Richard Lipka, Ph.D

Proč práce vznikala?

- Nízkoúrovňové zaměření
- Návaznost na předchozí práce (Ing. Hoang Ngoc Hung, Ing. Martin Mach)
- Absence software pro monitorování alokací paměti za běhu

Jak práce vznikala?

- Seznámení se s JVM a správou paměti
- Seznámení se s byte kódovou instrumentací v Javě
- Java Instrumentation API – Třída agenta, Instrumentation, Transformer
- Knihovna Javassist
- Získání pozice alokace - StackTraceElement
- Porovnávání objektů – Grafový algoritmus

Jak nástroj funguje?

- Pro instrukce newarray, anewarray, multianewarray:
  - Vytvoří na JVM zásobníku duplikát reference (DUP) a pouze zavolá metodu pro uložení reference (INVOKESTATIC)
- Pro instrukce new:
  - Je vytvořen uživatelský zásobník. Při nalezení instrukce new je na tento zásobník vložen název třídy, ke které se instrukce vztahuje. Na JVM zásobník je uložen duplikát reference (DUP)
  - Při nalezení instrukce invokespecial se porovná vrchol zásobníku s názvem třídy, ke které se instrukce vztahuje
  - Pokud se názvy třídy rovnají, pak je obsah vrcholu zásobníku smazán a za instrukcí invokespecial zavolána metoda pro uložení reference (INVOKESTATIC)
- Spuštění porovnávacího algoritmu na konci programu je zajištěno pomocí instrumentace metody main

Testování a výsledky

- Nástroj byl otestován na několika jednoduchých programech
- Otestováno i s knihovnou JavaFX
- Testy ověřily funkcionalitu instrumentace i porovnávacího algoritmu
- Porovnávací algoritmus funguje pro:
  - Jednoduché třídy s primitivními datovými typy
  - Pole
  - Třídy s komplexní strukturou (cykly, větší zanoření)
  - Částečně pro kolekce – obsahují metadata

Závěr

- Nástroj dokáže:
  - Nalézt všechny duplikáty instancí za běhu programu
  - Vypočítat velikosti alokovaných objektů
  - Zjistit, kde k alokacím došlo
- Nástroji chybí:
  - Grafické rozhraní
  - Přesné porovnávání kolekcí
  - Spouštění porovnávání „na vyžádání“
- V práci bylo dokázáno, že je možné využít instrumentaci Java byte kódu pro monitorování alokací paměti.
- V současném stavu je nástroj připraven k využití pro programy, které nevyžadují přesné porovnávání kolekcí

Děkuji Vám za pozornost

1. V práci není nikde šířeji zmíněno zpomalení aplikace díky prováděnému monitorování. Jak velké je a jaký to může být problém?

Analýza:

1. Při načtení třídy do paměti je spuštěn kód transformera. Budeme-li předpokládat, že třída nebude nikdy odstraněna z paměti, pak tento proces proběhne pouze jednou (většinou při spuštění programu).
2. Kód transformera probíhá při nejhorším v kvadratickém čase (je potřeba projít celou třídu, získat její metody, upravit je).
3. Všechny kroky nového byte kódu probíhají v konstantním čase – pouze dvě instrukce: duplikace reference, volání metody
4. Kód volání metody také probíhá v konstantním čase: přidání prvku do tabulky, přidání prvku do seznamu. (Jediná výjimka je u vícerozměrných polí, kdy je potřeba projít celé pole pro výpočet jeho velikosti).
5. Porovnávání je uskutečněno pouze jednou na konci běhu programu a jeho rychlost je lineárně závislá na počtu referencí.

Kroky 3 a 4 ovlivní běh programu (ovlivní user experience), kroky 1, 2 a 5 pouze „prodlouží dobu, po kterou je program zapnutý“. Většina kroků, které ovlivní UX jsou tedy konstantního času a neměly by výrazně zpomalit běh aplikace.

Test vytvoření 20 000 ArrayListů, každého s 10 000 prvků (datového typu Integer)

Průměrný běh testu bez nástroje [ns]	**Průměrný běh testu s nástrojem [ns]**
1162598860	2963919320

Zpomalení pouze cca 2.5x
Porovnávání ArrayListů trvalo 57 minut! (AMD Ryzen 5600G, 48GB Ram)