<KIV> KATEDRA INFORMATIKY A VÝPOČETNÍ TECHNIKY

Hung Ngoc Hoang

Diplomová práce

Inženýrská informatika
Softwarové inženýrství
2020/2021

Vedoucí práce:
Ing. Richard Lipka, Ph.D.

Detekce problémů se správou paměti v Java aplikacích

Abstrakt

Java je objektově orientovaný programovací jazyk populární díky automatické správě paměti a platformové nezávislosti. Kultura objektově orientovaného programování vybízí programátory, aby se soustředili především na objektový návrh aplikace, což je jednou z možných příčin neefektivního využití paměti. Práce popisuje některé případy, při kterých může docházet k neefektivnímu využívání paměti. Cílem práce je rozšíření nástroje, který je schopen detekovat případy plýtvání pamětí, o analýzu hluboce duplicitních objektů. Nástroj analyzoval několik reálných aplikací a odhalil velké množství případů duplicitních objektů.

Úvod

Cílem práce je analyzovat případy neefektivního využití paměti v Java aplikacích a následně rozšířit nástroj, který tyto případy umí detekovat. Důraz bude kladen na použitelnost nástroje na reálných aplikacích.

Východiska, analytická část

V současnosti v Javě vznikají vysoce škálovatelné programy, které jsou příliš abstraktní a trpí zmíněným neefektivním využitím paměti. Tyto programy potřebují poměrně velké množství strojového času a operační paměti k dosažení relativně jednoduchých funkcionalit. Je časté, že instance serveru potřebuje několik gigabajtů operační paměti, aby byla schopna obsloužit několik stovek uživatelů, přestože celá aplikace má být schopna obsloužit uživatelů milióny.

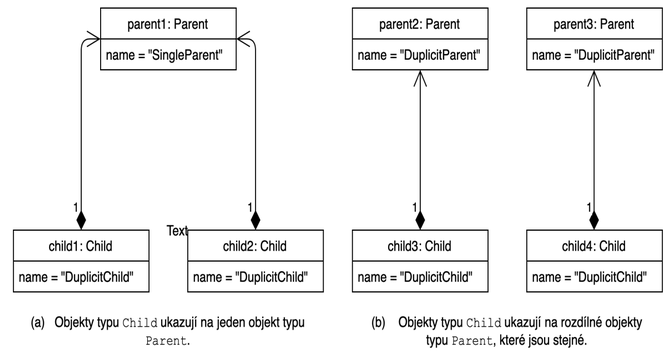
Příkladem neefektivního využití paměti jsou duplicitní řetězce v paměti. Java se snaží omezit duplicitu řetězců tzv. fondem řetězců, který do jisté míry tento problém řeší. Problém duplicitních řetězců můžeme zobecnit na duplicitu objektů. Na haldě se může vyskytovat velké množství objektů, které nejsou řetězce a mají duplicitní obsah. Pokud se jedná o neměnné objekty, tak zbytečně zabírají paměť navíc a zatěžují Garbage collection (GC), jelikož musí spravovat paměť, kterou pro ně alokoval. Detekce duplicitních instancí v paměti je hlavním cílem této diplomové práce.

Nástroj, jehož rozšíření je cílem této práce, používá k detekci duplicitních objektů mělké porovnání objektů, které porovnává referenční typy pomocí jejich hodnot, což jsou adresy objektů. Druhou možností je použít hluboké porovnání, které referenční typy porovnává na základě objektů, na které odkazují.

Hlavní aspekty realizace

Nástroj Memory Analyzer, který byl vytvořen Ing. Martinem Machem, byl během této práce rozšířen. Nástroj je napsaný v jazyce Java a analýzu provádí nad soubory, které obsahují snímek paměti (angl. heap dump) ve formátu hprof. Výstup aplikace je textový.

Hlavním rozšířením nástroje byla implementace detekce duplicitních objektů hlubokým porovnáním. Následující obrázek ukazuje dva případy duplicitních objektů. V prvním případě lze duplicitu detekovat jak mělkým, tak hlubokým porovnáním. V druhém případě lze duplicitu detekovat pouze hlubokým porovnáním objektů.



(a) Objekty typu `Child` ukazují na jeden objekt typu `Parent`.

(b) Objekty typu `Child` ukazují na rozdílné objekty typu `Parent`, které jsou stejné.

Příklady duplicitních objektů

Nástroj byl dále rozšířen o analýzu referencí a interaktivní shell, který lze použít pro snadnou kontrolu stavu jednotlivých instancí pomocí jejich číselných identifikátorů.

Dosažené výsledky

Původní i nová verze nástroje byly otestovány na testovacích aplikacích s uměle vytvořenými problémy. Původní verze nástroje odhalila minimum problémů, zato nová verze nástroje úspěšně odhalila problémy všechny. Dále byla nová verze otestována na reálných aplikacích, ve kterých odhalila velké množství duplicitních instancí. V případě analýzy vývojového prostředí IntelliJ IDEA tvořily duplicity téměř 16% všech instancí.

Tříd	Instancí	Duplicit	Doba trvání
22831	492248	79855	14 min 54 s

Analýza vývojového prostředí IntelliJ IDEA

Většina duplicit pocházela z tzv. datových tříd. Datová třída je taková třída, která obsahuje pouze data a metody k nim přistupující. Příkladem jsou instance, do kterých si vývojové prostředí ukládá data z XML souborů.

```
Class: com.intellij.XmlTextImpl
  myGapPhysicalStarts : Object =‘0‘
  myDisplayText : Object = ‘0‘
  myGapDisplayStarts : Object = ‘0‘

Class: com.intellij.XmlTagImpl
  myHC : int =‘0‘
  myImpl : Object = ‘0‘
  myValue : Object = ‘0‘
  myValue : Object = ‘0‘
```

Příklad nalezených duplicit v IntelliJ IDEA

Závěr

Výsledkem práce je nástroj Memory Analyzer rozšířený o detekci duplicit hlubokým porovnáním objektů. Nástroj byl otestován na testovacích aplikacích s uměle vytvořenými problémy. Rozšířený nástroj všechny problémy úspěšně odhalil. Dále byl otestován na reálných aplikacích, ve kterých odhalil velké množství duplicitních instancí. Nástroj lze nově ovládat v interaktivním režimu.

FAKULTA APLIKOVANÝCH VĚD ZÁPADOČESKÉ UNIVERZITY V PLZNI