6L-03

GCアサーションにおけるメモリリーク要因の提示機能

楊　光†　小宮　常康‡

†,‡電気通信大学　大学院情報理工学研究科

1 背景

メモリリークしたオブジェクトを発見するための機能にGCアサーション[1]がある．例えば以下のJavaプログラムでは，2行目で新しく作られたオブジェクトoをm2に渡した．

```
1 void m1() {
2   D o = new D();
3   m2(o);
4   assert_dead(o);
5   o = null;
6   ...
7 }
```

4行目のassert_dead(o)[1]は，プログラマの意図として，m2の実行を終えた時点でoはGCで回収可能であることを表明している（回収できなければメモリリークが生じている）．assert_deadにより，次回のGCの際，oで示すオブジェクトが死んでいなければ図1のような警告が発せられる．

```
Warning: an object that was asserted dead is
reachable.
Type: D;
Path to object: HashSet; -> D;
```

図1　assert_deadの警告メッセージの例

図1の警告メッセージから，HashSet型のオブジェクトからD型のオブジェクトまでのオブジェクトのリンクが作られたため，D型のオブジェクトがメモリリークしているのが分かる．しかし文献[1]のassert_deadでは，プログラムのどこ（行番号）でそのリンクが作られたかは分からない．

そこで本研究では，警告メッセージにおいて，GCルートからメモリリークしたオブジェクトまでのリンクがプログラム中のどこで作られたかも示すようにGCアサーション機能を改良する．

オブジェクトリンクを作ったプログラムの位置情報を収集するためのナイーブな手法では，全てのオブジェクトのデータ構造を改造し，オブジェクトのフィールドごとに，行番号を記録するためのフィールドを追加，そしてプログラムの実行時，全ての代入においてライトバリアで行番号を記録すれば良い．しかし，この手法では，メモリと実行のオーバーヘッドが大きい．そこで本研究では，該当行番号を一度に一つの型に絞って収集する手法を提案する．本手法はナイーブな手法に比べて，行番号を集めきるまでの時間はより要するが，メモリと実行のオーバーヘッドは小さい．

Enhanced GC assertions: Identifying memory leak locations
†Guang Yang
‡Tsuneyasu Komiya
†,‡Graduate School of Informatics and Engineering, The University of Electro-Communications

2 提案手法

大規模なサーバプログラムでは，同じロジックコードが繰返し実行されることがよくある．このようなコードにメモリリークを引き起こすロジックが含まれている場合，多くのリークオブジェクトが繰返し出現する．そして，一般的にこれらのリークオブジェクトまでのオブジェクトリンクは同じパターンになると考えられる．

例えば，1節のプログラムでは，メソッドm1が繰返し実行される場合，メソッドm1で作成された全てのD型のオブジェクトはいずれもHashSetに格納されてリークするケースが多いだろう．

これは，提案する手法が想定する適用可能条件となっている．

2.1 オリジナルのassert_dead

オリジナルのassert_dead(o)が呼び出された後，oで示すオブジェクトには「assert_dead」のマークが付けられる．次回のGCでコレクタがヒープ内のオブジェクトを辿る際，そのマークの付いたオブジェクトに遭遇すると図1に示された警告メッセージが出力される．

また，オリジナルのassert_deadのアルゴリズムでは，コレクタは深さ優先探索でヒープオブジェクトを走査し，走査したオブジェクトの情報は順番にキャッシュされるため，警告メッセージのオブジェクトリンクの情報はこれを使って実装される．

2.2 今回の拡張内容

拡張版assert_deadが呼び出されると，まず，オ

リジナルの assert_dead とほぼ同じように実行される：次回の GC の際，コレクタが「assert_dead」マークの付いたオブジェクトに遭遇すると，型に加えフィールドの情報も含んだリーク情報（図 2）を保存しておく．

```
A#field1; -> B#field2; -> C#field1; -> D;
```

図 2　拡張版 assert_dead が保存するリンク情報の例

次に，しばらくのプログラムの実行で，保存しておいたリンク情報にあるオブジェクトリンクの作成された行番号を一つ一つ見つけて記録していく．

図 3 のステップ①の赤い枠の部分のオブジェクトリンクの作成された行番号情報を収集するために，型 A のクラスの定義に field1 への代入に対する行番号情報記録用スロット f1_ln_slot を追加し [1]，型 A のオブジェクトの field1 に参照を書込むセッターメソッドにも行番号記録用ライトバリアを追加する．この時点からは，全ての新しく作成される型 A のオブジェクトが f1_ln_slot のフィールドを持ち，そしてこのような型 A のオブジェクトの field1 に，型 B のオブジェクトが代入される際に，代入操作のソースコードに位置する行番号が f1_ln_slot に記録される．

続いて，ステップ②の赤い枠の部分に進む．型 A のクラスの定義から，行番号情報記録用スロット f1_ln_slot と，型 A のオブジェクトの field1 セッターの行番号記録用ライトバリアを削除する．型 B のクラスの定義に行番号情報記録用スロット f2_ln_slot を追加し，そして型 B の field2 セッターに行番号記録用ライトバリアを追加する．この時点からは，型 B のオブジェクトが行番号情報記録用スロット f2_ln_slot を持ち，型 B の field2 のセッターに行番号記録用ライトバリアが追加される．

プログラムの実行に従って，最終的に，当該のオブジェクトリンクを形成する全ての代入に関する行番号情報が記録されたら，図 4 のような行番号を含んだ警告メッセージを示す．プログラムの 20 行目，30 行目と 55 行目の代入によって型 D のオブジェクトがリークしていたことが分かるようになった．

[1] スロットの追加やセッターの変更，ライトバリアの設置は，動的に行われることが望ましいが本稿ではその技術については触れない．本研究での実装では，実行を一旦止めてソースコードを変更，再ビルドし，再実行している（この一連の操作を自動化することも考えている）．

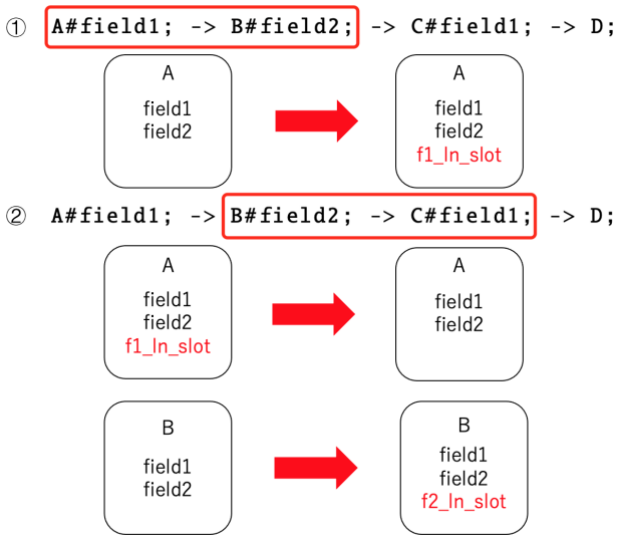

図 3　行番号記録用のスロットの追加の例

```
Warning: an object that was asserted dead is
reachable.
Type: D;
Path to object: A; @ln20 -> B; @ln30; ->
C; @ln55 -> D;
```

図 4　拡張版 assert_dead の警告メッセージの例

3　関連研究

Clifford らの研究[2]では，オブジェクトの扱いに関する情報をオブジェクトに持たせるが GC の際にその情報用スロットは消滅させる手法（若いオブジェクトに関する情報の効率的な管理手法）が提案されている．

4　まとめ

段階的行番号収集方式により，ナイーブな方式よりメモリと実行のオーバーヘッドを低減する手法を提案した．

参考文献

[1] Edward E. Aftandilian and Samuel Z. Guyer. GC assertions: Using the garbage collector to check heap properties. Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2009. pp. 235-244.

[2] Daniel Clifford et al.. Memento mori: dynamic allocation-site-based optimizations. Proceedings of the 2015 International Symposium on Memory Management. 2015. pp. 105-117.