Vol.2013-SE-179 No.22
2013/3/12

# 分散オブジェクトシステムにおけるGCの不連携による メモリ不足例外の原因検出手法

神田 翔太[1]　杉山 安洋[2]

**概要**：Java の分散オブジェクト技術である RMI では，ローカルホストとリモートホストの GC の不連携により不要なリモートオブジェクトが回収されず，メモリ不足例外が発生する場合がある．そこで，本研究では，メモリ不足例外が発生した際に，各 Java 仮想マシンのヒープダンプを取得・解析し，それらのヒープダンプからリモートオブジェクトが別の JVM 上のオブジェクトから参照されているかなどの情報を取得することにより，プログラムから使用されていないリモートオブジェクトを検出する手法を述べる．

## 1. はじめに

Java 言語で用いられる分散オブジェクト技術として RMI（Remote Method Invocation）[3] がある．RMI を用いると，ある Java 仮想マシン (JVM) 上のオブジェクトが，異なる JVM 上にあるオブジェクトのメソッドを呼び出すことが可能となる．RMI では，図 1 のように，まずスタブと呼ばれるオブジェクトをダウンロードし，クライアントオブジェクトはスタブに対してメソッド呼び出しを行う．スタブは，RMI ランタイムのオブジェクトを通じて，異なる JVM 上にあるオブジェクトへメソッド呼び出しを行う．

Java 言語には，GC（Garbage Collection）[2] という機能が備わっており，実行中のプログラムが使用しなくなったオブジェクトのメモリ領域を自動的に解放してくれる．これにより，一度確保されたメモリ領域が再利用できるようになる．しかし，オブジェクトを生成しようとした際に，GC が上手く動作せず空きメモリ領域が確保できないとメモリ不足例外が発生する．

分散システムでも同様に，プログラムが使用しなくなったリモートオブジェクトは自動的に解放されることが望ましい．しかし，分散システムは複数の JVM 上で実行されており，ある JVM 上のオブジェクトが他の JVM 上のオブジェクトを参照している状態にあるため，オブジェクトが他の JVM 上のオブジェクトから参照されていないかを調べるというような JVM 間での GC の連携が必要となる．RMI では，参照カウント方式 [6] の GC を利用しているが，

1 日本大学大学院工学研究科情報工学専攻
Graduate School of Engineering, Nihon University
2 日本大学工学部情報工学科
Department of Computer Science, College of Engineering, Nihon University

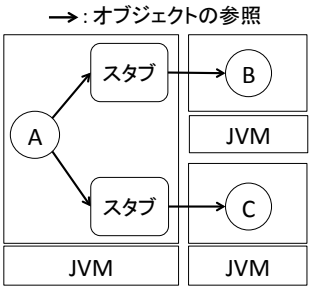


**図 1**　RMI の概要

プログラムから使用されていないリモートオブジェクトが回収されるためには，そのリモートオブジェクトのスタブが回収されていなければならない．しかし，ローカルホスト側の空きメモリ領域が多い場合には，スタブオブジェクトは GC により回収されないため，それに伴い使用されていないリモートオブジェクトも回収されないという問題が発生する．この問題を本稿では，GC の不連携と呼ぶ．

ここで，例外発生時のオブジェクトの情報を知る手段としてヒープダンプ [4] がある．ヒープダンプとは，ある特定の時刻に，JVM が動的に確保可能なメモリ領域であるヒープ内に含まれていた全オブジェクトの情報を出力したスナップショットのことである．ヒープを調べることにより，オブジェクトがどのオブジェクトを参照しているか，またどのオブジェクトから参照されているかを見つけることができる．ヒープダンプを解析するツールとしては，jhat (Java Heap Analysis Tool）[5] や Eclipse Memory Analyzer[7] などがあり，これらのツールを使用することにより，あるオブジェクトに至る参照，このオブジェクトが参照しているオブジェクトなどを閲覧することができ，ヒープダンプ取得時のプログラムの状況を分析することが

© 2013 Information Processing Society of Japan

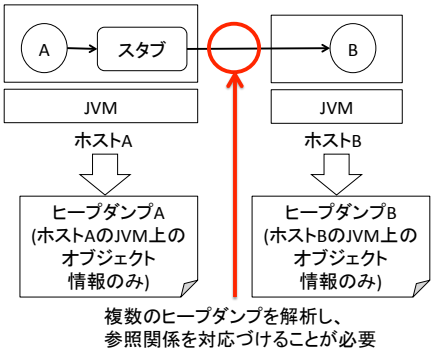


**図 2** 複数のヒープダンプの解析

できる．

しかし，分散システムにおいて，クライアントオブジェクトとリモートオブジェクトは異なる JVM 上にあるため，各 JVM から出力されたヒープダンプが必要となる．また，図 2 のように，それぞれのヒープダンプからは，そのヒープダンプが出力された JVM 上のオブジェクト情報しか得られない．そのため，JVM 間にまたがるオブジェクトの参照関係を分析するためには，複数のヒープダンプを解析して，参照関係を対応づけることが必要となる．既存のツールでは，複数のヒープダンプからオブジェクトの参照を関連づけることはできない．

そこで本稿では，分散システムにおいてメモリ不足例外が発生した場合に，各 JVM から出力された複数のヒープダンプを解析し，スタブとリモートオブジェクトの参照関係を対応付け，プログラムから使用されていない不要なリモートオブジェクトを検出する手法を述べる．

本稿の構成は以下の通りである．まず 2 章では GC の不連携について述べる．3 章ではヒープダンプを用いたデバッグ手法とその問題点を述べ，4 章では不要なリモートオブジェクトを検出する手法について述べる．そして，5 章ではツールについて述べる．そして 6 章では関連研究について述べ，7 章で今後の課題を述べる．

## 2. ガベージコレクションの不連携

RMI では図 3 のような参照カウント方式の GC を採用している．この方式を，図 3 の番号順に説明する．この方式では，まず，クライアントオブジェクトがリモートオブジェクトのスタブをダウンロードすると，そのリモートオブジェクトは他の JVM 上のオブジェクトから参照されるようになるので，参照カウントが増加される．そして，そのスタブが，クライアントオブジェクトから参照されなくなり GC により回収されると，参照カウントが減少される．参照カウントが 0 になり，リモートオブジェクトと同一 JVM 上のオブジェクトからも参照されていなければ，このリモートオブジェクトは GC により回収される．しかし，参照カウントが減少するのは，ローカルホスト側にある参照されなくなったスタブが GC により回収された後で

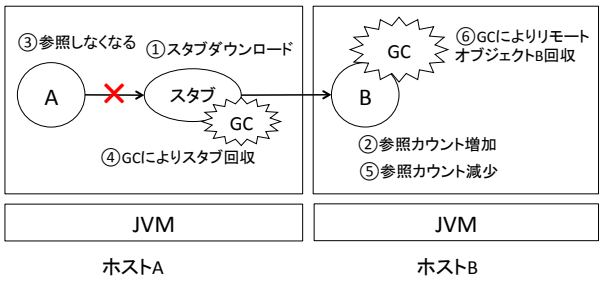


**図 3** RMI の参照カウント GC

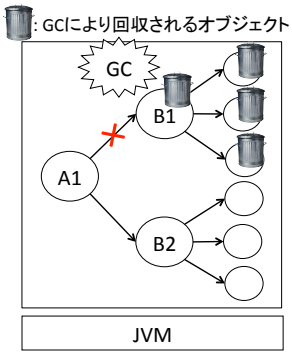


**図 4** スタンドアロンシステムの場合

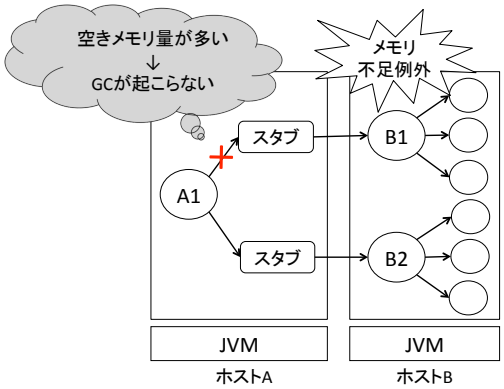


**図 5** RMI を用いた分散システムの場合

ある．このため，ローカルホスト側のメモリ領域に余裕がある場合には，GC が発生せず，参照されなくなったスタブがいつまでも回収されないということが起こる．それに伴い，使用されていないリモートオブジェクトも回収されない．プログラムから使用されなくなった不要なオブジェクトがいつまでもメモリに残り続けてしまうと，本来そのようなオブジェクトが削除されていれば起きないようなメモリ不足例外が発生してしまう場合がある．

例えば，図 4 のようなスタンドアロンシステムがあるとする．オブジェクト A1 がオブジェクト B1・B2 に対してメソッド呼び出しを行い，そのメソッドの中でインスタンス変数としてさらに様々なオブジェクトを生成している．この際，オブジェクト B1 や B2 のメソッドを実行するにはメモリを多く消費してしまうため B1 の処理が終わり次第，B1 への参照に明示的に null を代入するなどしてオブジェクト A1 から参照されなくなるとする．ここで，オブ

ジェクト B2 に対してメソッド呼び出しを行った際に，空きメモリ領域が少ないため，参照されてないオブジェクト B1 及びその参照先のオブジェクトは GC により回収される．それにより，空きメモリ領域が増加し，オブジェクト B2 はさらに新たなオブジェクトを生成できるようになる．

上記のようなスタンドアロンシステムのプログラムを RMI を用いて分散化したのが図 5 である．分散システムでも同様に，オブジェクト B1 は，処理が終わり次第，オブジェクト A1 からは参照されなくなるとする．しかし，オブジェクト B2 のメソッド実行中にホスト B 側でメモリの空き領域が少なくなっても，ホスト A 側のメモリの空き領域が多い場合，参照されなくなったスタブが削除されていないため，リモートオブジェクト B1 とその参照先のオブジェクトは削除されない．そのため，空きメモリ領域は増加せず，オブジェクト B2 が新たにオブジェクトを生成しようとした際にメモリ不足例外が発生してしまう．

## 3. ヒープダンプによるデバッグ手法の問題点

分散システム中のクライアントオブジェクトとリモートオブジェクトは異なる JVM 上にあるため，分散システム全体の状況を把握するためには，複数のヒープダンプが必要である．上記でも述べたが，各ヒープダンプには，出力された JVM のメモリ中に存在するオブジェクトに関する情報しか得られないため，オブジェクトと同一 JVM にあるオブジェクトへの参照しか辿れない．以降このような参照をローカル参照と呼ぶ．RMI を用いると，図 6 のような RMI ランタイムに関するオブジェクトが多数生成されるが，これらの，RMI ランタイムに関するオブジェクトのローカル参照を辿っても，あるオブジェクトが異なる JVM 上にあるどのオブジェクトを参照しているかはわからない．ここで，図 6 の$Proxy0 というのは，スタブのことである．また，ObjID と UID は，リモートオブジェクトを識別するために使用されるオブジェクトである．ObjID と UID については 4 章で詳しく述べる．

例えば，図 5 のような分散システムのヒープダンプを取得し，jhat を用いて解析すると，オブジェクト A1 とオブジェクト B1 の情報は図 7，8 のように表示される．ここでは，@の前がオブジェクトのクラス名であり，@の後がオブジェクトのハッシュコードの符号なし 16 進表現である．また，Instance data members の所にそのオブジェクトの参照先となるインスタンス変数のオブジェクトが表示され，Reference to this object の所にそのオブジェクトがどこから参照されているかという参照元オブジェクトの情報が表示される．

図 7 がオブジェクト A1 の情報である．この情報をみると null が入っている変数があることと$Proxy0 というオブジェクトを参照しているということしかわからない．jhat では，この$Proxy0 のローカル参照をさらに辿れるが，辿っていってもオブジェクト B2 を参照しているという情報は得られない．また，図 8 のオブジェクト B1 の参照元オブジェクト情報をみても，RMI ランタイムに関するオブジェクトから参照されているということしかわからない．この場合も，ローカル参照しか辿れないため，さらに参照元オブジェクトを辿っていっても，別 JVM にあるどのスタブから参照されているかという情報は得られない．これはオブジェクト B2 でも同様である．このように，それぞれのヒープダンプを既存のツールを用いて閲覧してもローカル参照しか辿れないため，他の JVM にあるオブジェクトまでの参照は辿ることができない．そのため，それぞれのヒープダンプから得られる情報を分析し，JVM 間にまたがるオブジェクトの参照関係の対応付けを行う必要がある．

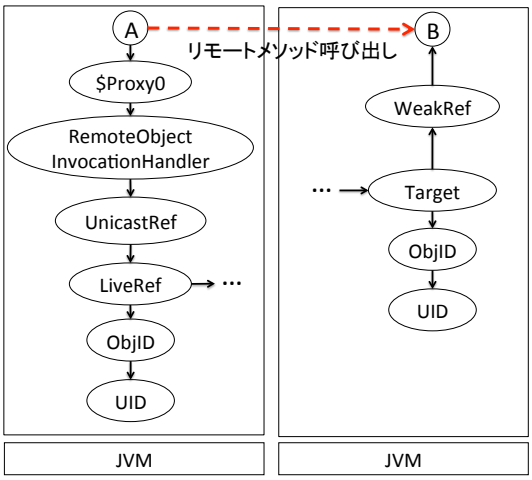


**図 6** ヒープメモリ中の RMI ランタイムに関するオブジェクト

また，メモリ不足例外発生時のローカルホスト側のヒープダンプを取得することが難しいという問題点もある．Java には，メモリ不足例外発生時にヒープダンプを出力するオプションが用意されている．これを使用することにより，メモリ不足例外が発生するリモートホスト側の JVM のヒープダンプを取得することはできる．しかし，リモートホスト側でメモリ不足になった際のローカルホスト側のヒープダンプを同時に取得することは難しい．プログラムのプロセス番号を指定して，任意のタイミングでヒープダンプを取得する jmap というコマンドもあるが，リモートホスト側でメモリ不足例外が発生し，ローカルホスト側に通知されるとプログラムは終了してしまうため，このコマンドを用いて手動でヒープダンプを取得することは難しい．

## 4. 提案手法

分散システムにおいて，メモリ不足例外が発生した際に，GC の不連携により削除されていない不要なリモートオブジェクトを検出したい．

そこで本稿では，以下のような手法により不要なリモートオブジェクトを検出する．

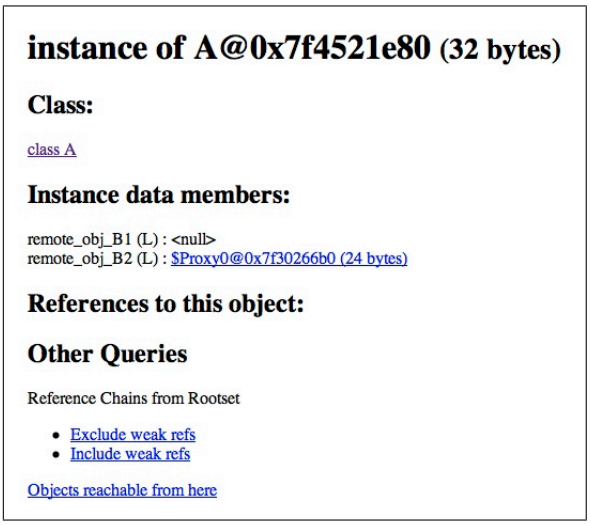


**図 7** jhat によるオブジェクト A1 の情報

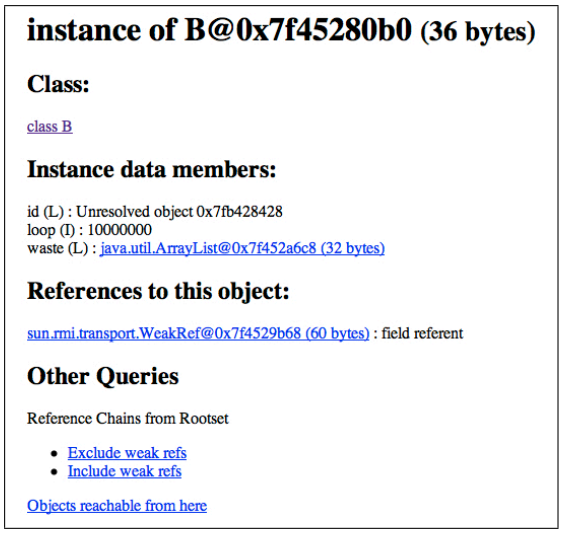


**図 8** jhat によるオブジェクト B1 の情報

(1) 各 JVM のヒープダンプの取得，解析
(2) リモートオブジェクトを識別するために使用される ObjID オブジェクトを用いて，スタブとリモートオブジェクトの対応付け
(3) 各リモートオブジェクトの参照元オブジェクトの調査
  (a) リモートホスト側で同一 JVM 上のオブジェクトから参照されているか
  (b) ローカルホスト側でスタブが他のオブジェクトから参照されているか
(4) 同一 JVM 上のオブジェクトからも参照されてなく，スタブがどこからも参照されていないリモートオブジェクトを検出

例えば，図 5 のような分散システムにおいて，本手法では，図 9 のようにして，不要なリモートオブジェクトを検出する．まず，各 JVM から取得したヒープダンプを解析すると，ヒープメモリ中にあるオブジェクトの参照関係がわかるようになる．次に，RMI ランタイムに関するクラスの 1 つである ObjID クラスのオブジェクトを用いて，スタブとリモートオブジェクトを対応づける．これに関しては，4.1 で詳しく述べるが，これにより，リモートオブジェクトとスタブオブジェクトを対応づけることができるようになる．そして，最後にリモートオブジェクトが同一 JVM 上のオブジェクトから参照されているかとスタブオブジェクトが参照されているかを判断し，図 9 のオブジェクト B1 のような使用されていない不要なリモートオブジェクトを検出する．

### 4.1 リモートオブジェクトとスタブの対応付け

RMI では，リモートオブジェクトを識別するために使用される ObjID というクラスがある．この ObjID クラスのオブジェクトは，図 6・図 9 のように，スタブの参照先オブジェクトを辿った先や，リモートオブジェクトの参照

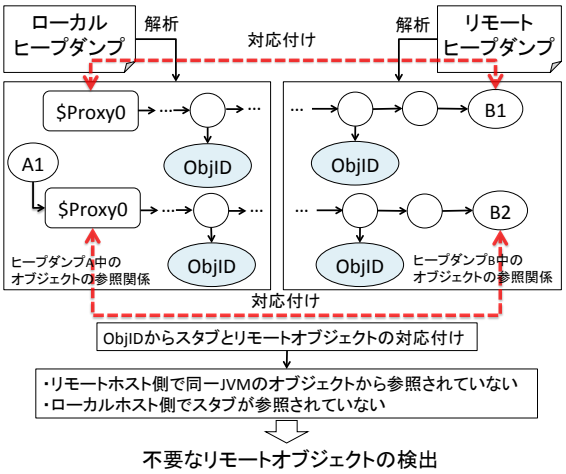


**図 9** 提案手法の概要図

元オブジェクトを辿った先にそれぞれ存在する．本手法では，この ObjID オブジェクトが持つ変数を基にスタブとリモートオブジェクトを対応づける．

この ObjID というクラスは，RMI ランタイムにエクスポートされたリモートオブジェクトを識別するために使用され，リモートオブジェクトを識別するための一意のリモートオブジェクト番号である objNum と UID クラスのオブジェクトを持っている．UID というクラスは，生成元のホスト上で常に一意である識別子を生成し，3 つのプリミティブ値の unique，time，count で構成されている．unique は，UID が生成された JVM を一意に識別する値である．time は，UID を生成した JVM の稼働時刻に，現在の時間をミリ秒で返す System.currentTimeMillis() から返される値である．count は，同じ JVM 上の同じ time 値に生成された UID を識別するための値である．これら 4 つの値 (objNum，unique，time，count) を調べることにより，ObjID がどのリモートオブジェクトに関連しているのかを識別する．よって，異なるヒープダンプ中に存在する

それぞれの ObjID と UID が持つ 4 つの値を比較し，もし一致した場合，そのリモートオブジェクトとスタブは参照関係があると対応づける．

### 4.2 不要なリモートオブジェクトの検出

本手法では，リモートオブジェクトが，同一 JVM 上のオブジェクト (RMI ランタイムに関するオブジェクトは除く) から参照されていなく，そのリモートオブジェクトと対応づけられたスタブがどこからも参照されていなければ，プログラムから使用されていないと判断し，不要なリモートオブジェクトとして検出する．

まず，リモートオブジェクトが同一 JVM 上のオブジェクトから参照されているかどうかを調べる．リモートオブジェクトの場合，図 6 のように，RMI ランタイムに関するオブジェクトからは必ず参照されているので，ここではそれら以外のオブジェクトから参照されているかを調べる．もし，RMI ランタイムに関するオブジェクト以外から参照されている場合，そのリモートオブジェクトは使用されていると考えられる．しかし，そのようなオブジェクトからの参照がない場合には，そのリモートオブジェクトは同一 JVM 上では使用されていないと判断できる．

次に，リモートオブジェクトと対応づけられたスタブが他のオブジェクトから参照されているかを調べる．もし，スタブの参照元オブジェクトがあれば，リモートオブジェクトは別 JVM 上のオブジェクトから使用されていると考えられる．しかし，スタブがどのオブジェクトからも参照されていない場合は，そのリモートオブジェクトは他の JVM 上のオブジェクトからは使用されていないと判断できる．

## 5. ツールの実行例と実装

不要なリモートオブジェクトを検出するためには，まずメモリ不足例外発生時の各 JVM ごとのヒープダンプが必要である．また，ユーザが jhat などの既存のツールを用いてリモートオブジェクトとスタブの対応付けを目視で確認するには，非常に労力を要する．そこで，メモリ不足例外発生時にヒープダンプを自動的に出力するツールと，リモートオブジェクトとスタブオブジェクトの対応づけを表示し，不要なリモートオブジェクトを検出するツールを開発した．本稿では，以降それぞれのツールをヒープダンプ取得ツールと原因検出ツールと呼ぶ．

これらのツールの概要は図 10 のようになっている．まずヒープダンプ取得ツールにより，メモリ不足例外発生時の各 JVM のヒープダンプを出力する．ヒープダンプ取得ツールを実行する際には，実行するユーザプログラムのアプリケーション名と実行するプログラムの種類を引数として入力してもらう．実行するプログラムの種類とは，メモリ不足例外が発生するリモートプログラムなのか，メモリ不足例外が通知されるローカルプログラムなのかである．これは，実行するプログラムを判断するために入力してもらう必要がある．出力されたヒープダンプを原因検出ツールにかけると，それらのヒープダンプが解析される．そして，ユーザにリモートオブジェクトのクラス名を入力してもらうことにより，原因検出ツールはそのリモートオブジェクトとスタブオブジェクトの対応付けを行い，不要なリモートオブジェクトを検出する．そして，最後に，リモートオブジェクトの参照関係と不要なリモートオブジェクトの検出結果を表示する．これによりユーザは，リモートオブジェクトが同一 JVM 上のオブジェクトから参照されていることや別の JVM 上のオブジェクトから参照されているかどうかが一目でわかるようになる．また，GC の不連携により，削除されていない不要なリモートオブジェクトを容易に発見することができるようになる．

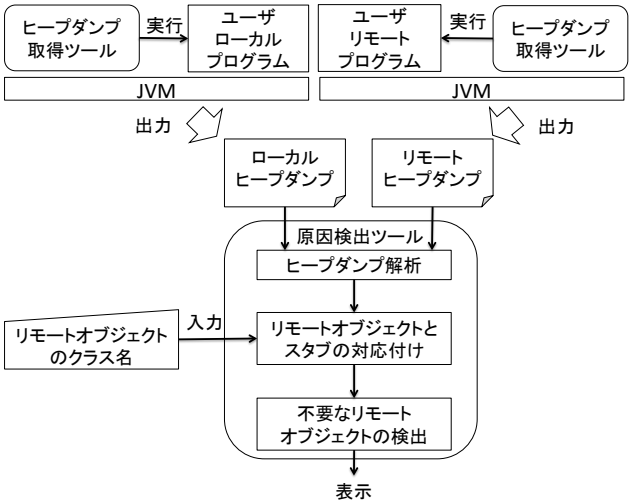


**図 10** ツール概要

### 5.1 実行例

まず，図 5 の分散システムがメモリ不足例外を発生させた時の場合について述べる．メモリ不足例外発生時のヒープダンプをヒープダンプ取得ツールによって取得し，原因検出ツールにかけ，リモートオブジェクトのクラス名を入力すると，図 11 のように各情報が画面に表示されるようになる．

画面の左には，入力したクラス名のリモートオブジェクトの参照元オブジェクト情報が階層的に表示される．図 5 では，リモートオブジェクトが 2 つあるため，それらの参照情報が図 11 では表示されている．リモートオブジェクトが同一 JVM のオブジェクトから参照されていれば，参照元オブジェクトが表示される．次に，リモートオブジェクトと対応づけられたスタブは，別 JVM からの参照を選択することにより表示される．図 5 のオブジェクト B2 のようにスタブが他のオブジェクトから参照されていれば，その参照元オブジェクトも表示される．しかし，図 5 のオ

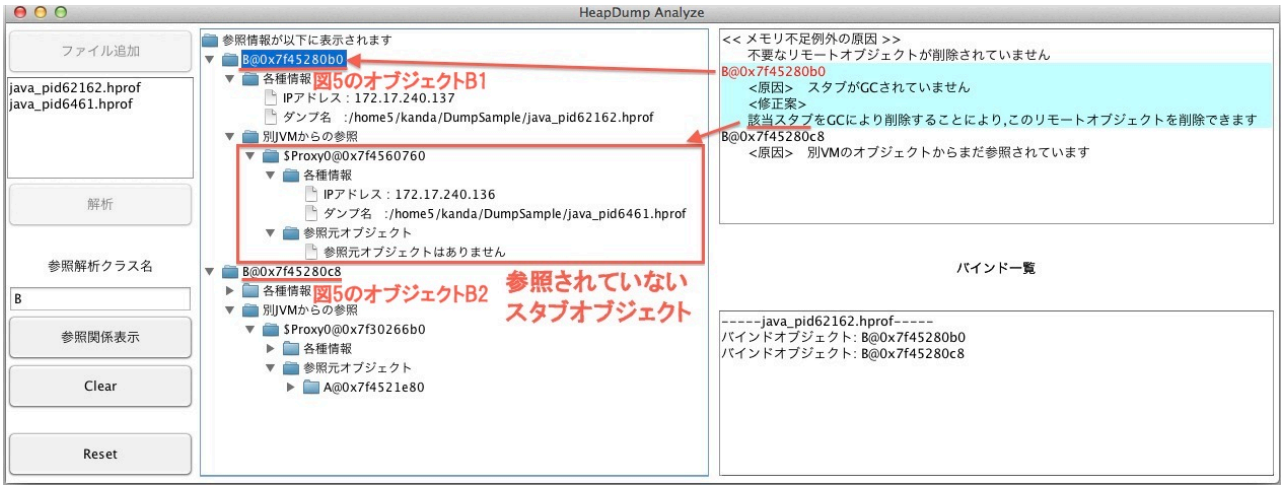


**図 11** ツール実行例

ブジェクト B1 のようにスタブがどこからも参照されていない場合は，参照元オブジェクトを選択した際に，参照元オブジェクトはありませんと表示される．なお，本来であれば，スタブの参照先オブジェクトやリモートオブジェクトの参照元オブジェクトに RMI ランタイムに関するオブジェクトが多数存在するが，それらを画面上に表示してしまうと，ユーザにとって見づらくなってしまうので，原因検出ツールでは必要最低限のオブジェクトしか表示されていない．また，各種情報を選択すると，そのオブジェクトが存在するホストの IP アドレスとどのヒープダンプから情報を取得したオブジェクトなのかを閲覧することができる．

画面の右上には，メモリ不足例外が発生した原因や，各リモートオブジェクトが GC によって削除されていない原因と削除可能ならばその旨を表示している．もし原因が GC の不連携によるものであれば，削除可能なリモートオブジェクトが赤文字で表示されるようになる．そして，削除されていない原因として，そのリモートオブジェクトのスタブが他のオブジェクトから参照されていないのに GC されていないということが表示される．また，修正案として，該当スタブを GC することによりリモートオブジェクトが削除できるようになるという旨のメッセージが表示される．

続いて，図 12 のような循環参照を起こしているプログラムの場合について述べる．図 12 の場合，オブジェクト A からオブジェクト B への参照がなくなると，オブジェクト B，C はプログラムからは使用されていないが，互いに参照し合っているため，RMI の参照カウント GC では，どちらも参照カウントが 1 のままとなり，削除されない．そのような場合には，図 13 のように表示される．原因検出ツールは，上記でも述べたように，オブジェクトの参照元オブジェクトを階層的に表示するが，循環参照が起きている場合，同一のオブジェクトが再び表示された段階で参照

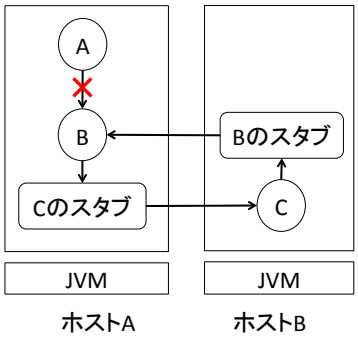


**図 12** 循環参照がある分散システム

元オブジェクトの表示を止める．また，循環参照を起こしているオブジェクトには，その旨も表示し，循環参照があることをユーザが一目で分かるようにしている．

このように表示することにより，スタブとリモートオブジェクトの参照関係の対応付けがとれ，既存のツールでは閲覧できなかった JVM 間にまたがるオブジェクトの参照関係がわかるようになる．また，GC の不連携により削除されていないリモートオブジェクトを一目で発見できるようになる．ユーザがこの情報をみて，該当スタブが参照されなくなった時に，System.gc() などを呼び出すようにプログラムを修正すると，ローカルホスト側で GC が発生し該当スタブが回収されるので，スタブと対応しているリモートオブジェクトも GC により回収されるようになる．

### 5.2 実装

ヒープダンプ取得ツールと原因検出ツールの実装概要について述べる．なお，原因検出ツールでは，jhat のソースコードを参考にして実装を行った．

#### 5.2.1 ヒープダンプ取得ツール

ヒープダンプ取得ツールは図 14 のように動作する．まず，引数に指定されたのがリモートプログラムの場合，そのアプリケーションを実行する際に，”-

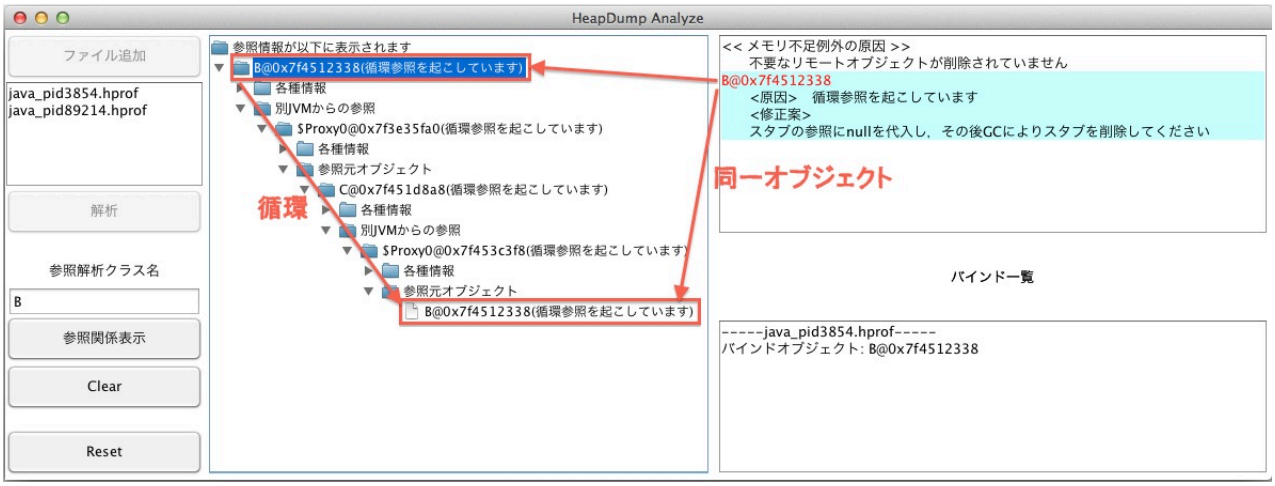


**図 13** 循環参照を起こしている場合の表示例

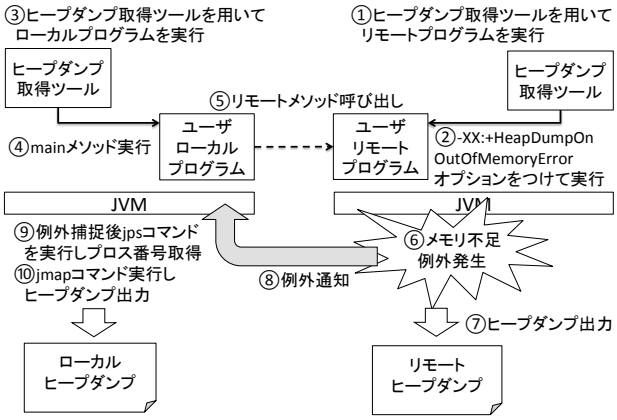


**図 14** ヒープダンプの取得

**図 15** ヒープダンプのバイナリデータ

XX:HeapDumpOnOutOfMemoryError" というオプションをつけて実行する．これにより，リモートプログラムがメモリ不足例外を発生した際に，その時点のヒープダンプが出力されるようになる．

引数に指定されたのがローカルプログラムの場合，ヒープダンプ取得ツールはローカルプログラムの main メソッドを実行する．ローカルプログラムが，リモートプログラムに対してメソッド呼び出しを行い，リモートプログラムがメモリ不足例外となると，ヒープダンプ取得ツールに例外が通知される．ヒープダンプ取得ツールはその例外を捕捉し，jps コマンドを用いてローカルプログラムを実行しているプロセス番号を取得後，jmap コマンドを用いて取得したプロセス番号のヒープダンプを出力する．

### 5.2.2 原因検出ツール

ヒープダンプのバイナリデータは図 15 のようになっている．はじめの 4 バイトは，JAVA という文字列情報が入っており，その後には，ダンプのバージョンやファイル生成時刻などの情報が入っている．続いて 1 バイト読み込むと，どんな情報が入っているかを識別するための値（RecordType）が入っており，その後その情報が何バイト入っているかを示す値が入っている．そして，そのバイト数分 RecordType が示す情報が入っている．後は，RecordType，RecordType の長さ，RecordType が示す情報の繰り返しである．この RecordType が示す情報の中に，プログラムで使用されているクラス名やオブジェクト情報，参照情報などが含まれている．

原因検出ツールでは，ヒープダンプのバイナリデータを解析し，図 16 のようにして，オブジェクトの参照先オブジェクトや参照元オブジェクトの情報を取得する．まず，ヒープダンプを読み込み解析すると，JavaHeapObject というクラスのオブジェクトが，メモリ上にあるオブジェクトごとに生成される．この JavaHeapObject からは，各オブジェクトのクラス名や参照先オブジェクト，参照元オブジェクトの情報を得られる．その後，生成された JavaHeapObject を Snapshot というクラスの Hashtable に格納する．リモートオブジェクトとスタブオブジェクトの対応付けを行う際に，この Hashtable から，必要なオブジェクト情報をもつ JavaHeapObject を取得する．例えば，まずリモートオブジェクトのオブジェクト情報を持つ JavaHeapObject を Hashtable から取得する．取得した JavaHeapObject より参照元オブジェクト情報を取得し，参照元のオブジェクト情報を持つ JavaHeapObject を Hashtable から取得する．このように，JavaHeapObject から参照元情報を取得し，

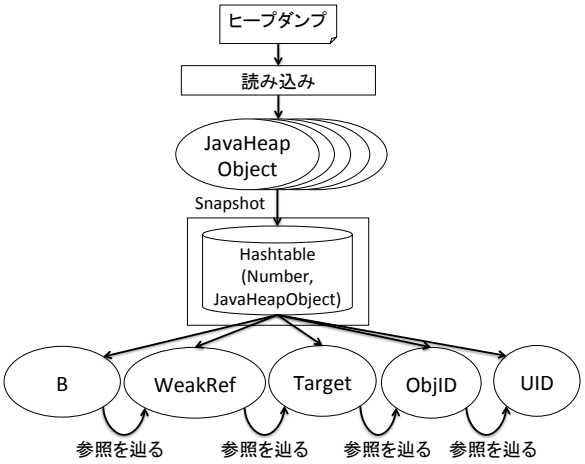


**図 16** オブジェクト情報の取得

それに該当するオブジェクト情報を持つ JavaHeapObject を Hashtable から取得することで，ObjID と UID のオブジェクト情報を取得する．そして得られた ObjID と UID のオブジェクト情報から，さらに対応付けに必要な 4 つの値を取得し，比較することにより，スタブオブジェクトとリモートオブジェクトを対応づける．

不要なリモートオブジェクトの検出の際も，このリモートオブジェクト情報とスタブオブジェクト情報を持つ JavaHeapObject から参照元オブジェクト情報を得ることにより，それらのオブジェクトが他のオブジェクトから参照されているかどうかがわかる．

## 6. 関連研究

分散システムにおけるデバッグを支援する研究として，文献 [1] のようなものがある．これは，クライアントプログラムとサーバプログラムの動作を対応づけて監視，表示するものである．プログラムの処理の遷移をスレッド単位で表示したり，エラーに関わる処理を表示することにより，複雑な分散システムの動作を理解することが可能となる．

上記のようなシステムでは，分散システムの動作理解の支援にはつながるが，上記で述べたような例外の発生原因となるオブジェクトは特定できない．本稿では，複数のヒープダンプを対応づけて解析することにより，JVM 間にまたがるオブジェクトの参照を対応づけられることに特徴がある．これにより，メモリ不足例外が発生した場合に，例外発生原因となるオブジェクトを特定することができるようになる．

## 7. まとめと今後の課題

本稿では，複数のヒープダンプからリモートオブジェクトとスタブオブジェクトの対応付けを行い，GC の不連携により削除されなかった不要なリモートオブジェクトを検出する手法と，そのような不要なオブジェクトを自動的に検出するツールについて述べた．原因検出ツールは，分散システム上のスタブとリモートオブジェクトを対応づけを行うことにより，プログラムから使用されていない不要なリモートオブジェクトを簡単に検出することができる．そのため，既存のツールなどを使って，関連づけに必要な値を調べる必要がなくなる．特に，リモートオブジェクトが多数ある場合，そのオブジェクトごとに ObjID が持つ値を目視で確認するには非常に労力を要するため，それらの対応付けを自動的に行うことにより，リモートオブジェクトの分散システム上の参照関係や不要なリモートオブジェクトを一目で把握できるため，特に有効であると考える．

今後の課題としては，各ツールの機能改善が挙げられる．まず，ヒープダンプ取得ツールは，分散システムが 3 つ以上の JVM 上で動作している場合，メモリ不足例外も発生せず，その例外も通知されない JVM が出てくるため，その JVM のヒープダンプが取得できないという問題がある．そのため，3 つ以上の JVM 上で動作している場合でも，メモリ不足例外発生時の各 JVM のヒープダンプを取得できるようにすることが今後の課題として挙げられる．また，ヒープダンプ取得ツールを用いてヒープダンプを出力すると，ヒープダンプはそれぞれのホスト上に生成される．そのため，原因検出ツールを使う際にユーザは，sftp コマンドなどによりヒープダンプを原因検出ツールを実行するホスト上に集めなければならない．そこで，原因検出ツールでヒープダンプを選択する際に，他のホスト上にあるヒープダンプも読み込めるようにすることが挙げられる．

最後に，本稿で GC の不連携による問題点を挙げたが，GC をうまく連携させることで，本稿で挙げたような不要なリモートオブジェクトをメモリ不足例外が発生する前に事前に削除できるようにし，例外発生によりプログラムが停止することなく最後まで実行できるような仕組みを作ることが今後の課題である．

### 参考文献

[1] 黒田澤希，上山善嗣，新井利明，分散プログラム開発支援ツールの開発，全国大会講演論文集 第 49 回，p.83-84，1994
[2] Oracle : Java HotSpot Garbage Collection http://www.oracle.com/technetwork/java/javase/tech/gc-142724.html
[3] Oracle : Java Remote Method Invocation http://docs.orale.com/javase/1.5.0/docs/guide/rmi/index.html
[4] Oracle : Java VisualVM http://docs.oracle.com/javase/jp/6/technotes/guides/visualvm/heapdump.html
[5] Oracle : jhat - Java Heap Analysis Tool http://docs.oracle.com/javase/6/docs/technotes/tools/share/jhat.html
[6] Oracle : リモートオブジェクトのガベージコレクション http://docs.oracle.com/javase/jp/6/platform/rmi/spec/rmi-arch4.html
[7] The Eclipse Foundation : Memory Analyzer http://www.eclipse.org/mat/

© 2013 Information Processing Society of Japan