ポータブルなデータベースガーベジコレクション

郷原　浩之[†]　葛上　昌司[††]

Hibernate 上でポータブルなデータベースガーベジコレクション (DBGC) を実現した. 本ソフトウェアは, ミドルウェアとして利用する Hibernate が DB の差異を吸収することでポータブル性を実現している. Hibernate が保有する完全なテーブル間関連のメタデータを利用して GC を実現するために必須の到達可能経路情報を獲得している. GC のアルゴリズムは湯淺の snapshot-at-the-beginning 法を用いた. マーキングは SQL を用いた集合演算によって行っている. GC のマークを世代番号として表現した整数値のマーキングフラグを導入したことで, スイープを通常の mark-and-sweep 法の順序とは異なり, 任意のタイミングで行うことができる. オブジェクトの新規追加とポインタの参照先の付け替えに関する情報は Hibernate が持つオブジェクトの状態監視機構を改変して獲得している. 世代番号を利用することによって DB アプリケーションと並列に GC を行うことができる.

A portable database garbage collection

Hiroyuki Gohara[†] and Shouji Kuzukami[††]

We implemented a portable database garbage collection system(DBGC) running on Hibernate. Portability of this system is realized by using Hibernate as a middleware, which absorbs the differences of DB management systems. We get the reachability of objects which is necessary for realizing GC from metadata of table relations Hibernate holds. We adopt Yuasa's snapshot-at-the-beginning algorithm as the GC algorithm. Marking is executed as set operations by SQL. We adopted integer value as the marking flag, so the sweep phase can be executed at any time independent of usual mark-sweep sequence. Information of newly added objects and pointers which are changed can be obtained by peeking Hibernate's object observation system. DBGC can be executed in parallel with DB services utilizing integer typed marking flag.

1. データベースガーベジコレクションとは

リレーショナルデータベース (以下, DB) にはメモリ上で存在するようなガーベジコレクション (以下, GC) の機構が存在しない. 従ってデータの削除は手動で行うことを余儀なくされているが, 不正確なデータの削除は DB 内部の制約違反を誘発し DB 全体の破壊を招くために, 通常は削除を示すフラグを設置するのみでデータは残しておく. このように全てのデータを残しておく方法は, 必要以上のセキュリティリスクを取り, 必要以上に高性能な DB を導入することを余儀なくさせるため, データを正確に削除するシステムが求められてきた.

本ソフトウェアは Hibernate がサポートする多くのリレーショナル DB 上で動作するデータベースガーベジコレクション (以下, DBGC) を

† 東京大学大学院工学系研究科システム創成学専攻
Dept. of Systems Innovation, School of Engineering, The University of Tokyo.
†† (株) 情報基盤開発
Intelligent Infrastructure Development. Co., Ltd.

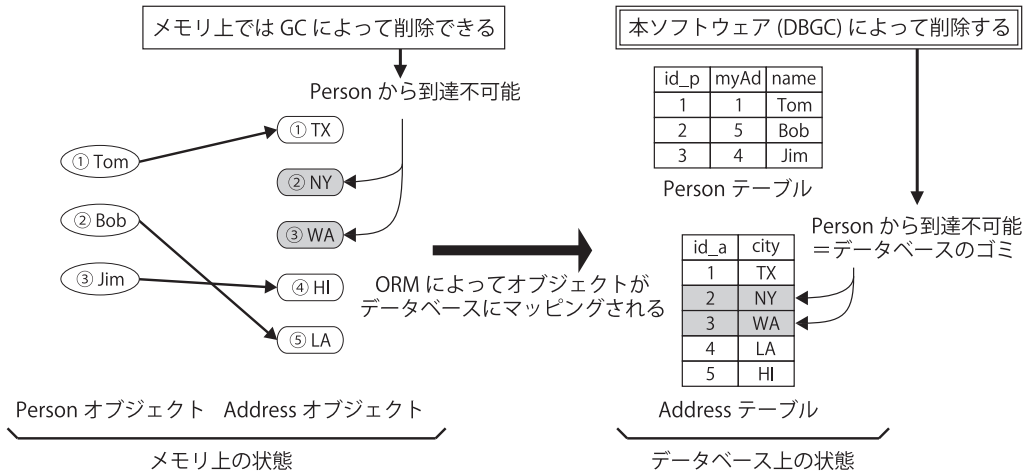

図 1 DBGC が削除すべきデータベースのゴミの説明

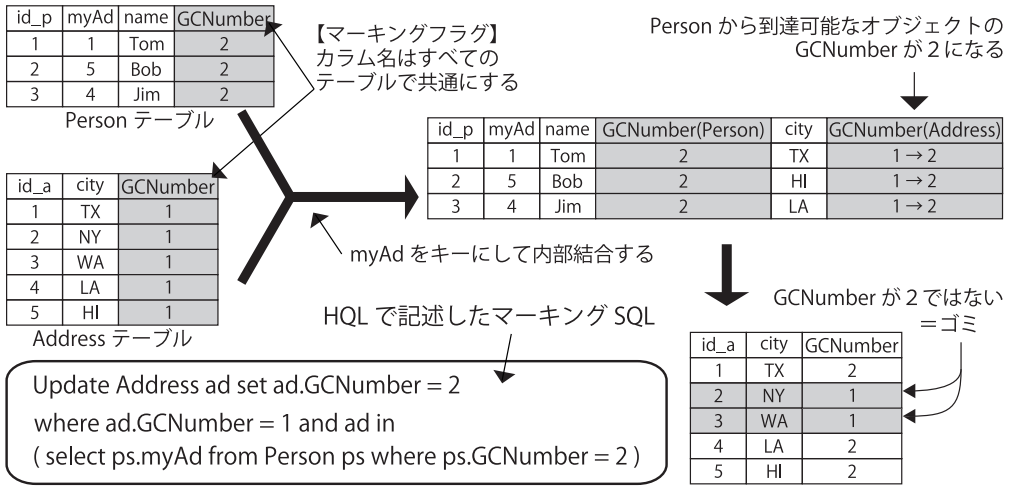

図 2 SQL によって集合演算マーキングをする仕組み

実現した.

図 1 の左側はメモリ上のオブジェクトの状態を表している. メモリ上のオブジェクトはオブジェクトリレーショナルマッピングツール (以下, ORM) によって図 1 の右側のようなテーブル構造で DB に保存される.

メモリ上では, Person オブジェクトをルートとしたとき, 灰色で示された Address オブジェクトはどの Person オブジェクトからも参照されていないので, GC によって削除される. 一方で, DB 上では Person テーブルをルートとしたとき, 灰色で示された Address テーブルの行はどの Person

テーブルの行からも参照されていないので削除したいのだが, GC が存在しないため, 灰色で示されたゴミは手動で削除するしかない.

このようにルートから到達不可能なオブジェクトをゴミと定義し, メモリ上の GC で削除されるようなオブジェクトを DB 上からも削除することが DBGC の目的である.

オブジェクト DB 上での GC の理論 [1][2][3][4][5][6] やソフト [8] は存在する. 本ソフトウェアはオブジェクト DB よりも広く普及しているリレーショナル DB で GC を実現した点に特徴がある.

2. SQL による集合演算マーキング

GC のアルゴリズムには mark-and-sweep 法を用いている. DBGC の特徴はマーキングを SQL による集合演算によって行っている点である. 図 1 で定義した DB のゴミを SQL による集合演算によって同定している様子を図 2 で示した.

SQL 文にはオブジェクトを一意に特定するような識別子 (id) は記述されておらず, テーブル名とマーキングフラグと内部結合のための関連名のみで構成されている点が集合演算としてマーキングを行う DBGC のマーキング SQL の特徴である.

図 2 の例では Person テーブルをマーキング元, Address テーブルをマーキング先としている. ここで Person テーブルにある'myAd' という関連名を用いれば, 図 2 の右のような結果を導く両テーブルの内部結合クエリを生成できる. この結果に対してマーキングの有無に関する条件を update 文の条件句に加えることで適切なデータのマーキングフラグを操作可能なマーキング SQL を発行できる.

図 2 で示した GCNumber というプロパティがマーキングフラグを表す. GC を何回生き残ったかという世代番号を表現することができるように, マーキングフラグは整数値とする.

3. ポータビリティの実現

本ソフトウェアではミドルウェアとして Java で標準的な ORM である Hibernate を採用した. Hibernate は DB へのアクセスを抽象化するため, Hibernate 上での動作を前提とすればポータビリティが実現する.

Hibernate は HQL という独自の SQL 方言をサポートする. HQL は Hibernate によって実行時に各 DB の SQL 方言に変換されるため HQL で記述されたマーキング SQL のポータビリティも保証される.

4. マーキングの実現

集合演算マーキングを行うためにはマーキング元テーブル名, マーキング先テーブル名とその関連名が必要である (図 2).

また全ての関連について収束するまでマーキングを行うこと必要であるが, SQL の発行はディスクへのアクセスを伴い, 実行時間が大きくなるので, SQL の発行回数を最小限に抑えたい. そのため, 本ソフトウェアは発行すべき SQL の順序を適切に並び替えることでマーキングが収束するまでの SQL の発行回数を最小化する.

4.1 到達可能経路情報の取得

全ての ORM はテーブルに関連付けられたある一つの関連オブジェクトから別のオブジェクトについて読み込み保存を行うため, 完全なテーブル間関連がメタデータとして設定される. このメタデータを利用することで到達可能経路情報が得られる.

本ソフトウェアでは到達可能経路を隣接行列で表し, 参照グラフマトリックスと名づけた. 参照グラフマトリックスはマーキング SQL に必要なマーキング元テーブル名, マーキング先テーブル名とその関連名を保持する.

Hibernate はクラスのメンバの型, プロパティ名, 関連が記憶された ClassMetaData を各クラ

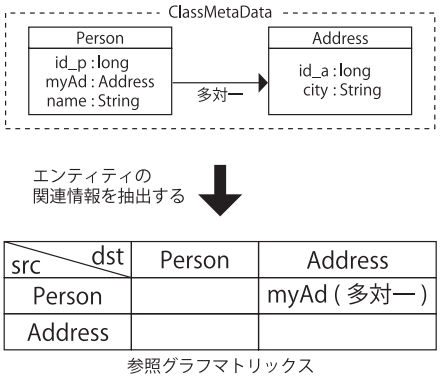

図 3　参照グラフマトリックスを獲得する方法

スについてそれぞれ保持する．その ClassMetaData をすべて読み込み解析することで, 完全なテーブル間関連のメタデータとすることができ, 参照グラフマトリックスを作ることができる (図 3)．本ソフトウェアでは Hibernate の API を通じて ClassMetaData を獲得する．参照グラフマトリックスが得られれば全てのマスに対応するマーキング SQL をマーキングが収束するまで発行することで DBGC が実現する．

4.2　マーキングの収束条件

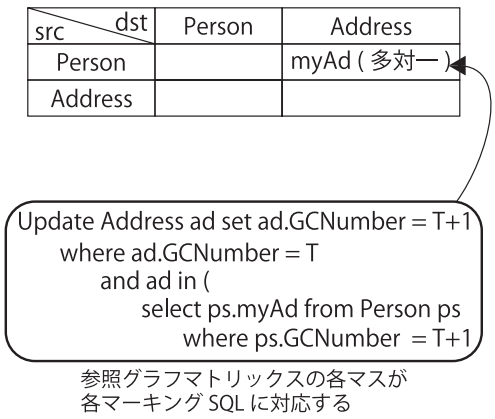

図 4　参照グラフマトリックスとマーキング SQL の対応関係を表した

参照グラフマトリックスは DB の全データの型の参照関係を表すため DB 中の全データグラフは参照グラフマトリックス中のいずれかのマスに割り当てられている (図 4)．

このため任意に設定した GC ルートから追跡可能なデータを全て辿るというマーキング条件は, 参照グラフマトリックスの全マスに対応するマーキング SQL を全く更新がなくなるまで繰り返すことである．HQL で update 文を実行すると, update された列の数を戻り値として獲得できる．マーキング SQL による更新の有無はこの機能を使うことで判定できる．

4.3　SQL 発行回数最小化のためのグラフ解析

DB のアクセス回数を最小化することがマーキング実行時間を短縮するために最も重要である．そのため, ここでは発行すべき SQL の順序を適切に並び替えることでマーキングが収束するまでの SQL の発行回数を最小化する．

(1)　エンティティ群に循環参照がない場合

エンティティ群に循環参照がない場合, エンティティ群の参照グラフに対してトポロジカルソートによって SQL の最適な発行順序を決定することができる．

図 5 は同一の SQL 発行回数であっても発行順序によってマーキング結果に差が出る例を示した．(α) の順序でマーキングを行うと, 3 回のマーキングで A から到達可能なエンティティのみがマークされた状態になる．しかしながら (β) の順序でマーキングを行うと, 初回の C → D のマーキングでは C の到達可能性が未定であるので, どの D もマーキングされない．3 回のマーキング終了時に, A から到達可能な D が全くマーキングされていない状態になっている．そのため (β) では完全なマーキングのためにはもう 1 度 C → D のマーキング SQL 発行が必要である．つまり合計で 4 回の SQL 発行となり, SQL の順序によって収束するまでの SQL 発行回数が異なることがわかる．

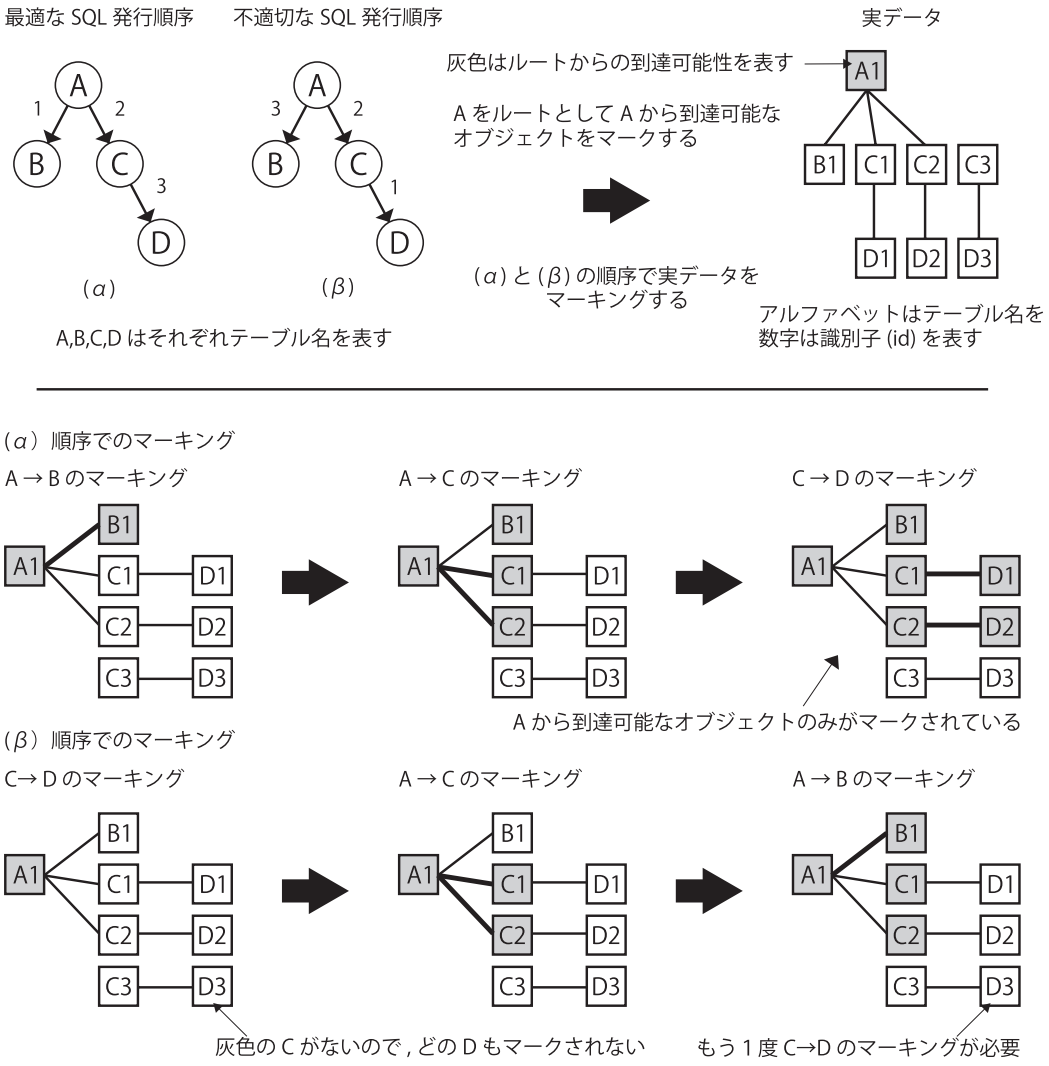

図 5　エンティティ群に循環参照がない場合の最適な SQL 発行順序

(2)　エンティティ群に循環参照がある場合

エンティティ群に循環参照がある場合, エンティティ群の参照グラフから全ての循環参照を同定する. この段階では得られた循環参照が別の循環参照の部分集合とならないようなものを一かたまりにする.

図 6 の例では, 循環参照しているエンティティ群は (C, D, E, F),(C ,D, F),(D, E, F) の 3 つがあるが, いずれも (C, D, E, F) の部分集合となるため,(C, D, E, F) を一かたまりにする.

このような循環参照を一かたまりとして扱うことで, 参照グラフから循環参照を擬似的になくすことができる. 循環参照がなくなれば, 先述のようにトポロジカルソートをすることで, SQL の最適な発行順序が決定できる. ただしこの時, マーキングが循環参照しているエンティティ群に到達したら, 循環参照が収束するまでマーキングを行う (図 6).

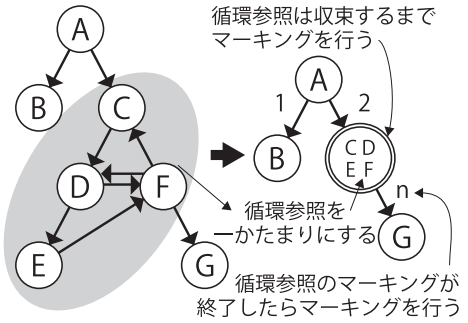

図 6 エンティティ群に循環参照がある場合の最適な SQL 発行順序

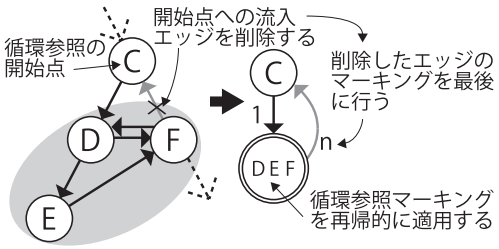

図 7 循環参照のマーキング手法

循環参照のエンティティ群のうち, 循環参照の外から参照があるエンティティを循環参照のマーキング開始点とする. 一般にこのようなマーキング開始点は複数の候補が存在する可能性があるが, 本アルゴリズムでは任意の一つを選べばよい. 次にマーキング開始点への流入エッジを削除する. 図 7 では (C, D, E, F) のうち A からの参照が存在する C をマーキング開始点とする.

マーキング開始点への流入エッジを削除することは, マーキング開始点に該当するエンティティを参照循環の構成要素から外すことに相当するため, この操作後は図 6 のアルゴリズムによって最適な SQL の発行順序が決定できる. この順序による SQL とその最後に削除したエッジに該当するマーキング SQL を加えたものを一つの SQL 発行単位とし, 収束するまでこの単位による SQL 発行を繰り返す. そして, 循環参照の部分集合として存在する循環参照にもこのアルゴリズムを再帰的に適用すれば, 任意のグラフ構造をもつエンティティ群に対して適切な SQL 発行の順序でマーキングが完了する (図 7).

5. 湯浅式 GC による並列 GC の実現

DBGC を他のサービスと並列に実行するための GC のアルゴリズムとして湯浅の snapshot-at-the-beginning GC[7] を採用した.

5.1 新規追加とポインタ付け替え情報

湯浅式 GC の実現のためにはオブジェクトの新規追加とポインタの付け替え情報を把握する必要がある. Hibernate はデータベースへのアクセスを減らす目的で, オブジェクトの状態を監視し, 変更内容の差分のみを DB と通信する. このオブジェクト監視機構はダーティチェックと呼ばれ, ダーティチェック機構を一部改編をして湯浅式 GC に必要な情報を手に入れることにした.

ダーティチェックによって得られたオブジェクトの新規追加情報とポインタの付け替え情報を他のサービスと共有するためにオブジェクトメタデータテーブルを用意する.

5.2 トランザクションと GCNumber

本ソフトウェアでは Hibernate のトランザクションを拡張し, トランザクションも GCNumber を保持する. トランザクションは開始時点の GCNumber が割り当てられる. トランザクションの最中ににオブジェクトが新規追加された場合はそのトランザクションが保有する GCNumber をそのオブジェクトに割り当てる.

またトランザクションの最中にオブジェクトのポインタ付け替えがあった場合には, 付け替え元オブジェクトと付け替え先オブジェクトの情報とそのトランザクションが保持する GCNumber をオブジェクトメタデータテーブルに書き込む.

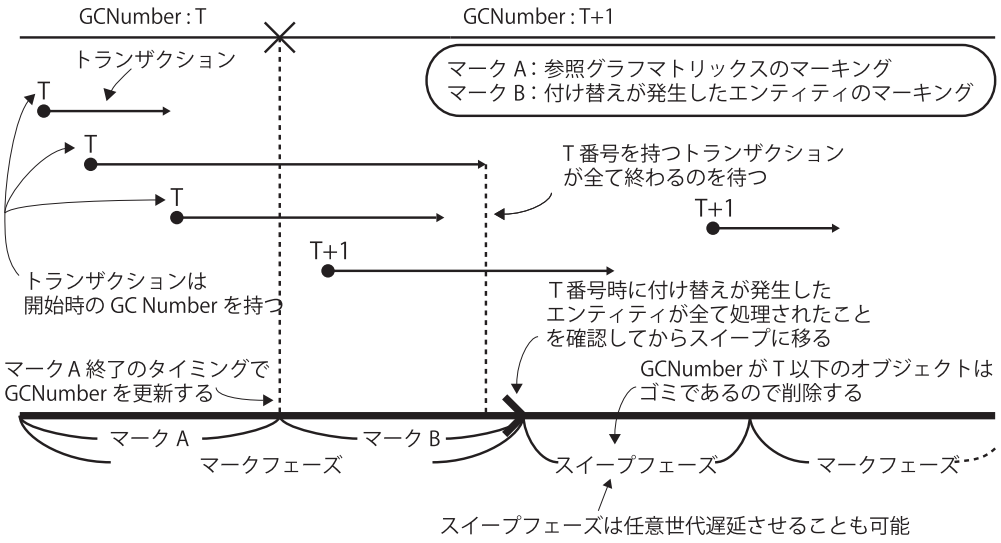

図 8　DB 上で湯浅式 GC が動作する仕組みとトランザクション管理

5.3　マークフェーズ

マークフェーズではまずグラフ解析によって得られた SQL 発行順序に沿ってルートエンティティからマーキングを進める (図 8 のマーク A). マーク A が終了したら GCNumber を更新する. マーク A の最中にオブジェクトが DB に新規追加されたり, DB のオブジェクトの参照ポインタが付け替えられた場合は, オブジェクトメタデータテーブルの情報に基づいてマーキングを行う. 付け替えられたオブジェクトは, そのオブジェクトをルートにしてエンティティグラフ全体をマークする.

ある GCNumber でマーキングが収束したことを保証するためには, その GCNumber 時に行われたオブジェクトの新規追加やポインタの付け替えのマーキングが終了したとを保証しなくてはならない. しかしながら, 現時点ではまだ GCNumber が更新される前のトランザクションが生き残っている可能性がある.

そこで, GCNumber が更新される前のトラザクションが全て終了するまで待機する. 待機している間に GCNumber が更新される前のトラザクションの中でオブジェクトの新規追加やポインタの付け替えがあった場合には, その情報をオブジェクトメタデータテーブルに書き込み, 同様に処理する.

GCNumber が更新される前のトラザクションが全て終了しかつ GCNumber が更新される前の付け替え情報がオブジェクトメタデータテーブルから完全になくなったらスイープフェーズに移る.

5.4　スイープフェーズ

ルートから到達可能なオブジェクトは最新の GCNumber を必ず保持する. それゆえに最新以外の GCNumber を保持するオブジェクトを削除することで, スイープが実現する. 本ソフトウェアでは安全にオブジェクトを削除する目的で Hibernate の削除メソッドを利用している.

ルートからの到達可能性の同定のみを目的とする場合などは, 必ずしもマークフェーズの後に必ずスイープを行う必要はない. GC のマーキングフラグを整数値にしているため, 最新の GCNumber を保持しないオブジェクトはマーキング対象とならないことが保証できる. それゆえ

にスイープフェーズのみを任意世代遅延させて実行することが可能である.

最新以外の GCNumber はそのオブジェクトがルートから到達不可能となったタイミングを表すため, GCNumber 更新のタイミングを記憶しておくことで, 一定時間前から DB に保存されているオブジェクトのみを削除するといったことも可能になる.

例えば, マークを連続して行い, その後スイープを行っても, マーク & スイープを交互に 2 回行った状態と同等な結果が得られる.

6. 現　　状

本ソフトウェアは湯淺式 GC を拡張することで, スイープを任意のタイミングで実行できるようになった.

この機能は特定の期間データを預かった後, そのデータを削除することが契約に盛り込まれているウェブサービスの領域において貢献できる.

6.1 入　　手

本ソフトウェアは Apache ライセンスを適用して, http://sourceforge.net/projects/database-gc/ にて, オープンソースで公開しており, 使用することができる.

6.2 技術的制約

本ソフトウェアではガーベジコレクションが対象とするオブジェクトには本ソフトウェアが用意する GC フラグをもつクラス (GCCollectableEntity) を必ず継承することにしているが, この制約のために Hibernate が用意する one-to-one 関連マッピングや unionSubclass を用いた継承マッピングを使用することができない. これらのマッピングは本ソフトウェアではサポート対象外としている. one-to-one 関連マッピングは必要ならば別の方法で表わすことができ, unionsublass についてはそもそも使用頻度が低いので, この 2 つをサポート対象外とすることは問題とならない.

6.3 技術的障壁

Hibernate は HQL を各 SQL 方言に変換するが, その HQL コンパイラにバグがあり, 本システムは HSQLDB のみでの完全な動作が確認されている. メモリ上でのオブジェクト操作を前提とする Hibernate の性質上, HQL での DB 上でオブジェクトの操作を行うことになる update 文の使用は推奨されておらず, update 文の実装が他よりも貧弱である. 現在は PostgreSQL や MySQL 上でも完全な動作をするよう HQL コンパイラでの update 文のバグ修正に取り組んでいる.

またコレクションも Set のみをサポートしている. 今後は List などのサポートや Map のサポートも行う. いずれも実現のための技術的障壁は Hibernate の HQL コンパイラにあるので, Hibernate の品質向上が本ソフトウェアのソフトウェアの品質向上になるものと考えている.

スイープフェーズでは最新の GCNumber 以外のオブジェクトの削除を Hibernate が持つ delete メソッドを使用している. このメソッドではオブジェクトを一つづつ削除するため, 実行時間が長い. スイープのために必要な HQL の delete 文の実行は, DB 自体の制約と整合を取るために容易には使用できず, 多くの場合完全性制約違反となるため, delete 文自体の実行ができない. 循環参照が全くない場合は, トポロジカルソートの逆の順序で SQL を発行することで, 完全性制約に違反せず delete 文による SQL スイープが実現する. しかしながら循環参照がある場合の SQL スイープについてはまだ実現方法が考えつかない. 現状では, 循環参照のみを Hibernate の delete メソッドで削除し, 循環参照外を SQL によって削除することを検討している.

ゴミであると同定されたオブジェクトの全てのポインタに null を代入すれば, 先述のような順序を意識せず, delete 文による SQL スイープが実現する. しかしながら現段階では Hibernate が update 文による null の代入を許容していない.

スイープの観点からも HQL コンパイラの修正こそが優先して取り組むべき課題であると考えている.

将来的には分散 DB での DBGC を考えている. DBGC に不可欠な join 演算子の使用が制限されることが, 現在の技術的障壁である.

謝辞

本ソフトウェアは「未踏 IT 人材発掘・育成事業 (未踏ユース)」の 2008 年度下期採択案件として IPA の支援を受け, 竹内郁雄プロジェクトマネージャーの指導のもと開発された.

参 考 文 献

1) Amsaleg,L. ,Franklin, M. and Gruber,O.: Efficient Incremental Garbage Collection for Client-Server Object Database Systems, Proceedings of the 21th International Conference on Very Large Data Bases(1995) pp.42 - 53
2) Amsaleg,L. ,Franklin, M. J. and Gruber,O.: Garbage collection for a client-server persistent object store, ACM Transactions on Computer Systems, Vol. 17, No. 3, August 1999, pp. 153-201
3) Brown F.: Incremental Garbage Collection in Massive Object Stores, ACSC '01: Proceedings of the 24th Australasian conference on Computer science,Gold Coast, Queensland, Australia(2001) pp.38 - 46, IEEE Computer Society
4) Buttler, M. H.: Storage Reclamation in Object Oriented Database Systems, In Proceedings of the ACM SIGMOD Annual Conference on Management of Data (SIGMOD '87, San Francisco, CA, May 27-29), U. Dayal, Ed. ACM Press, New York, NY, pp.410 - 425.
5) Cook, J. E ,Wolf, A. L. and Zorn, B. G.:A Highly Effective Partition Selection Policy for Object Database Garbage Collection, IEEE Trans. Knowledge and Data End., vol.10, no.1, Jan./Feb. 1998, pp.153 - 172
6) Roy P. ,Seshadri S. ,Silberschatz A. ,Sudarshan S. and Ashwin S.: Garbage collection in object-oriented databases using transactional cyclic reference counting, The VLDB Journal(1998) 7 pp.179 - 193
7) Yuasa, T.: Real-Time Garbage Collection on General Purpose Machines, Journal of Software and Systems(1990) pp.181 - 198
8) ZODB Distributed GC http://pypi.python.org/pypi/zc.zodbdgc