# プログラム言語 Ruby におけるメソッドキャッシング手法の検討

笹田 耕一
東京農工大学大学院 工学教育部

## 1 はじめに

筆者はオブジェクト指向スクリプト言語 Ruby[1] で記述されたプログラムを高速に実行するための処理系 *YARV: Yet Another Ruby VM* を開発している．YARV は Ruby で記述されたプログラムを適切な命令列にコンパイルし，それを高速に実行する仮想機械（VM: Virtual Machine）である．

Ruby はクラスベースのオブジェクト指向プログラミング言語であるため，メソッドディスパッチのためにはレシーバのクラスから，その親クラスへメソッドの定義を発見するまで検索する必要があるが，これを毎回行うのはオーバヘッドが大きいため，最適化が必要である．

そこで，YARV では命令中に直前に呼び出したメソッド定義をキャッシュするインラインメソッドキャッシュを実装した．本稿ではその手法の検討について述べ，実装し評価した結果を示す．

## 2 メソッドキャッシュ

オブジェクト指向プログラム言語の実装では，メソッド探索の最適化が必要であり，さまざまな方式が検討されている [2].

Ruby では変数やメソッドの返値の型が明示されないことや，メソッドの再定義が可能であるため，静的解析によるメソッドディスパッチの最適化が不可能である．そのため，これらに対応できる動的な手法が求められる．

メソッド検索を行うたびに，その結果をキャッシュするメソッドキャッシュを利用した最適化は実装も容易であり，さまざまな処理系で利用されている．

現在の Ruby 処理系では，VM 1 つに対し，1 つのメソッドキャッシュ用の表を有している．表にはクラスとメソッド定義の組が 1 エントリとして格納する．すべてのメソッドディスパッチはこの表のレシーバのクラスとセレクタに対応するエントリを参照し，キャッシュが有効であればキャッシュしたメソッド定義を返し，検索を終了する．以下，この手法をグローバルメソッドキャッシュという．この方式では 95% 以上のメソッドキャッシュヒット率をもつ [1, 3].

## 3 インラインメソッドキャッシュ

YARV では命令オペランドに一度検索したメソッド定義をキャッシュするインラインメソッドキャッシュを設計し実装した．

```
/* vmsc : VM state counter          */
/* entry: inline method cache entry */
if(entry->recv_class == recv_class &&
   entry->vmsc == vmsc){
  method = entry->method;
} else{
  entry->recv_class = recv_class;
  entry->vmsc = vmsc;
  entry->method = method =
    lookup(selector, recv_class);
}
```

図 1: インラインメソッドキャッシュアルゴリズム

メソッドディスパッチ命令 `send` のオペランドにメソッドキャッシュ用のオペランドを用意し，その領域にレシーバのクラスとメソッド定義情報を格納する．ディスパッチ時，レシーバのクラスをキャッシュしたクラスと比較し，等しければヒットとする．

もしキャッシュミスが起こった場合は，従来のグローバルメソッドキャッシュを用いて検索を行う．

1 つのメソッドディスパッチに注目すると，おおむね前回と同じメソッドが呼ばれるという実行特性から [2]，インラインメソッドキャッシュを用いることで高いキャッシュのヒット率を期待できる．

**VM 状態カウンタ** Ruby プログラムでは，任意の箇所でメソッド定義を変更することができるため，キャッシュの内容がクリア可能でなければならない．

たとえば，クラス `A` を継承したクラス `B` があり，メソッド `A#m`（クラス `A` に定義されているメソッド `m`）がある状態で，クラス `B` のオブジェクト `b` に対しメソッド `m` を実行（`b.m()`）した場合，`A#m` を実行し，この定義をキャッシュする．その後，`B#m` を新たに定義した場合，`b.m()` は `B#m` を実行するべきだが，`A#m` をキャッシュしているため，誤って `A#m` を実行してしまう．

これを防ぐためには，(1) 再定義時にキャッシュをクリアする (2) 再定義されていればキャッシュを利用しない，の二つの方式が考えられる．(1) では，コンパイルしたすべての命令列の中のキャッシュエントリを走査しクリアする必要があるため，再定義のオーバヘッドが大きい，キャッシュエントリの集合の管理が必要，などの問題がある．

そこで，YARV では (2) を適用することとした．これを実現するために ***VM* 状態カウンタ** (VMSC: Virtual Machine State Counter) を設けた．これを利用した

---

A study on Inline Method Caching in Ruby Programming Language
Koichi Sasada: Graduate School Technology, Tokyo University of Agriculture and Technology

表 1: マイクロベンチマークの結果（単位は秒）

| | (1) | (2) | (3) |
|---|---|---|---|
| (A) | 2.79 | 2.88 | 3.11 |
| (B) | 3.45 | 3.40 | 3.28 |

(1) レシーバのみチェック（再定義に非対応）
(2) VM 状態カウンタもチェック
(3) グローバルメソッドキャッシュのみ
(A) 多態性を利用しないプログラム
(B) 多態性を利用したプログラム

表 2: インラインメソッドキャッシュのヒット率

| | キャッシュヒット率（%） | ミスが発生したディスパッチ率（%） |
|---|---|---|
| cal | 98.7 | 0.9 |
| RSS | 97.2 | 9.8 |
| ReXMLbib | 76.6 | 8.3 |
| SOAP | 95.5 | 26.4 |
| Webrick | 97.8 | 6.2 |

＊ヒット率について初期ミスを除き計算

キャッシュアルゴリズムを図 1 に示す.

メソッドが再定義されるごとに VMSC はインクリメントされる．インラインメソッドキャッシュに書き込む際には，そのときの VMSC の値もメソッド定義とともにエントリに格納する.

キャッシュを引くときには，レシーバクラスのチェックとともに，現在の VMSC とエントリの VMSC の値も比較する．もし違うならば，そのキャッシュエントリは無効であると判断することが可能である.

VMSC を用いる手法 (2) は，(1) に比べ実装が非常に簡単であるという利点がある.

オーバヘッドを考えると，(1) にくらべ (2) では VMSC のチェックのための分岐がひとつ増えるが，再定義操作は一般的にまれであるため，最近のプロセッサの分岐予測機能によりかなりの遅延隠蔽が期待できる．また，(2) では異なるセレクタのメソッドについての再定義時にもインラインキャッシュが無効化してしまうが，再定義操作の希少性から問題ないと考えられる.

## 4 評価

本項では YARV に実装した機構について評価する．また，Ruby プログラム自体のインラインキャッシュの有効性についても検証する.

**性能評価** YARV に実装したインラインメソッドキャッシュの性能を調べるため，マイクロベンチマークを実行した（表 1）．評価は Intel Pentium III 500MHz, Windows2000 + cygwin (gcc 3.3.3), ruby 1.9.0 (2005-01-09), YARV rev-120 で行った.

表 1 中 (A) は，多態性を利用せず，必ずインラインメソッドキャッシュがヒットする例である．グローバルメソッドキャッシュのみの場合に比べ，8% の性能向上を得ることができた．ただし，VMSC のチェックを省略することで 3% ほど性能向上が可能であることがわかった.

表中 (B) は多態性を利用するプログラムで，キャッシュミスしか起こらないようなプログラムを作成した．この場合，インラインキャッシュのチェックが余計なオーバヘッドとなり，グローバルメソッドキャッシュのみに比べ 3% 性能低下した.

**Ruby プログラムにおけるインラインメソッドキャッシュの有効性** インラインメソッドキャッシュは，多態性を利用するプログラムではキャッシュヒット率が減少し，すでに見たようにインラインキャッシュが余計なオーバヘッドになる．そこで，いくつかの Ruby プログラムについてどの程度のキャッシュヒット率が見込めるかについて調査した（表 2）．利用したプログラムは Ruby サンプルプログラムとして用意されているものと，自作の Ruby プログラム（ReXMLbib）である.

これによると，ほとんどの場合でキャッシュヒット率が 95% を超えた．これによりインラインメソッドキャッシュが有効であることがわかる．ヒット率が低い ReXMLbib のミスが発生したメソッドディスパッチ率（すべてのメソッドディスパッチ箇所数と比較）を見ると，約 10% と偏りがある．つまり，特定のメソッドディスパッチでのみミスが頻発していることがわかる.

## 5 おわりに

本稿では Ruby におけるインラインキャッシュによる最適化について述べ，その設計と実装した結果を示した．VM 状態カウンタを用いることで，簡単な実装にもかかわらずメソッドディスパッチの性能が 8% 向上した．また，Ruby プログラムではインラインキャッシュのヒット率は十分高いことがわかった.

本稿で述べたインラインメソッドキャッシュ方式は，並行実行したときの排他制御についての対応が不十分であるため，これについての検討が必要である．また，インラインメソッドキャッシュがミスするのは，一部のメソッドディスパッチに偏っていることがわかったため，この性質を利用して最適化する手法を検討したい.

**謝辞** 本処理系の開発プロジェクトは，IPA（情報処理推進機構）の公募事業 2004 年度未踏ソフトウェア創造事業「未踏ユース」（プロジェクト・マネージャ：筧 捷彦早稲田大学教授）に採択され，支援を受けました.

## 参考文献

[1] まつもとゆきひろ, 石塚圭樹 オブジェクト指向スクリプト言語 Ruby, 株式会社アスキー (1999).
[2] 小野寺民也 オブジェクト指向言語におけるメッセージ送信の高速化技法, 情報処理, **38**, 4 (1997), 301–310.
[3] 青木峰郎 Ruby ソースコード完全解説, インプレス (2002).