Java Just-in-Time コンパイラ実用化と配布の経験

首藤 一幸　村岡 洋一

早稲田大学 理工学部
〒 169–8555 東京都 新宿区 大久保 3–4–1
{shudoh,muraoka}@muraoka.info.waseda.ac.jp

あらまし　我々は，1998 年より Java Just-in-Time コンパイラを開発し，世の中に配布してきた．研究の基盤として有用なものを開発するという第一の目標と同時に，当初より，実用的なソフトウェアの開発をもう一方の目標として掲げてきた．配布と並行して，開発と実用性向上に取り組んできたその過程で，利用者や他の開発者，研究者とのさまざまな意見交換，情報交換を経験してきた．本稿では，実際的なソフトウェア開発を目標とする研究者にとって興味深いであろう我々の経験を紹介する．

キーワード: 実用化; 公開; 品質; ライセンス

1　はじめに

我々は，1998 年より Java バイトコードの実行時コンパイラ（JIT コンパイラ）shuJIT [6][10] の開発に取り組んできた．開発の動機は単純に創作意欲であったが，この JIT コンパイラを続く研究の基盤と位置付けることで，研究上の意義を見出した．そのため，基盤，材料として利用しやすいものを作ることが目標となった．それと同時に，当時 Linux で使用できる JIT コンパイラが少なかったこともあり，もう一方の目標として，実用的なソフトウェアに仕立てて配布し，世の中で広く使ってもらえることを目指した．

多くの開発者が経験している通り，ソフトウェア開発の際，アイディアを実証できた，または単に動作したという段階から実用になるという段階まで質を向上させることには大きな労力が要る．また，配布しようという場合には，使用法などの文書を用意することはほぼ必須であるし，利用者からの問い合わせも発生する．必ずしも問い合わせに対応する必要はないものの，対応しないと，そのソフトウェア，ひいては開発者本人が信頼を失いかねない．

このように，ソフトウェアの実用化や配布には大きな労力が伴い，しかもその労力は，アイディアを実証するまでとは違った種類のものである．さらに，研究者の社会的役割は新規性の高い仕事にあるという考えに基づくと，この種の労力は研究者が割くべきものではないとも言えるかも知れない．しかし，実用化によって，その過程で初めて問題が明らかになる場合もある．また，配布することでアイディアの有効性を他者に直接確認してもらえる上，利用者から問題の報告を受けることができれば，それによって質の向上を図れる．さらに，ソースコードまで公開すれば，他の研究者の参考になるだけでなく，研究基盤として利用されることもあろう．

本稿では，我々が shuJIT を配布してその実用性を高めてきた経験と，その過程で得た，ソフトウェア公開にあたっての指針を紹介する．JIT コンパイラに限定した技術的な経験ではなく，利用者，他の研究者とのインタラクションなど，ソフトウェア開発に共通の経験について述べていく．

2　shuJIT の概要

shuJIT は，Java バイトコードの実行時コンパイラ，すなわち JIT コンパイラである．Intel 社の IA-32（x86）プロセッサをターゲットとし，FreeBSD および Linux 上で，Sun Microsystems 社の classic VM という Java 仮想マシンと共に動作する．

開発には，次の方針で臨んだ．

- **研究基盤**として利用しやすいものとする．
- **実用的**なソフトウェアとする．
- **少ない労力で短期間で開発**する．ひとりで，長くとも数ヵ月程度の期間で作る．

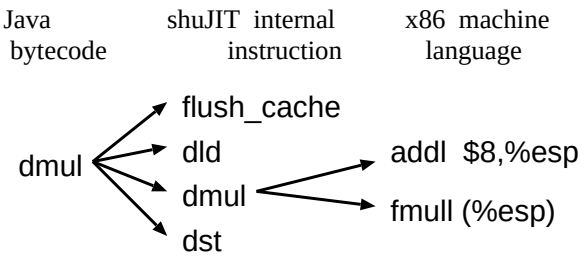

図 1: コード生成手法

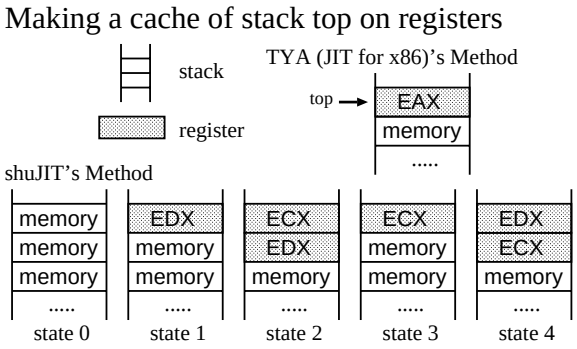

図 2: スタック状態

この方針に基づいて種々の工学的選択を行った．例えば，shuJIT 側であらかじめ用意したネイティブコードをつなぎ合わせていくというコード生成手法（図 1）は，研究基盤としての扱いやすさと開発期間の短縮を同時に狙った選択である．これによって，JIT コンパイラにとって重要なコンパイル時間の短さも達成された．この方式を採用したことで，コンパイラ内にアセンブラを持つ必要がなくなり，その分，開発の手間を削減できた．同時に，shuJIT 開発者にとっては，コンパイラが生成する生成するネイティブコードを直接手で記述できるという利点があり，これによって，shuJIT を基盤とした諸研究 [10][7] が可能となった．一方，このコード生成方式には，レジスタ割り付けを行えないがために複数レジスタを活用しづらいという問題がある．この問題に対し，スタック状態（図 2）という考えを導入し，各状態に応じたネイティブコードを用意しておくことで，問題を緩和した [10]．

また，コンパイル時間の短さを損なわずに行える最適化をいくつか実装してきた．すなわち，メソッドのインライン展開，末尾再帰の除去，ピープホール最適化，トラップを利用した例外の捕捉，ループ先頭の 16 バイト境界への整列などである．

表 1 に，SPEC JVM98 [8] を用いて，400 MHz の K6–2 プロセッサで，各種 JIT コンパイラと shuJIT の性能を比較した結果を示す．

3 公開後の反応

shuJIT の開発は 1998 年 6 月 15 日に開始した．その後，1 度の作り直しを経て，9 月 20 日に web ページでソースコードを公開し，同時にメイリングリストにてアナウンスを行った．それから 2001 年 3 月 9 日までに，ソースコードは 7558 回，コンパイル済みバイナリは 8476 回のダウンロードがあった．

公開の際は，shuJIT 自身の紹介や利用法を書いたいわゆる README ファイルを英語で用意した他，英語と日本語で web ページを用意した．英語で書いたのはなるべく多くの人に知ってもらうためであり，日本語で書いたのは，英語を読む労力を避けがちな日本人にも知ってもらうためであった．

反応は，口頭のものを除いては，すべて電子メイルで受け取った．手もとに残っているメイルの内訳は次の通りである．削除してしまったメイルも含めると、実際はこれより多少多い。

問題の報告	40 件
コメント	13 件
質問	11 件
動作報告	11 件
その他	5 件

問題の報告は，ほとんどが Segmentation Violation による Java 仮想マシンごとの異常終了についてである．質問は，JIT コンパイラを作成する際の API についてが大勢を占めた．その他は例えば，shuJIT を紹介した，といった連絡があった．

3.1 多量の問題報告

実用化には品質の向上が欠かせず．質を向上していくためには利用者からの問題報告は大変貴重である．多くの開発者は、自分自身でも，ソフトウェアが想定した動作をするかどうか試験するだろう．しかし現実には，あらゆる入力，環境に対して試験することは不可能である．配布されたソフトウェアは，利用者の手もとで，開発者が想定していなかった，または試験できなかった様々な条件で利用され，結果的に試験され

SPEC JVM98	_227_ mtrt	_202_ jess	_201_ compress	_209_ db	_222_mpeg-audio	_228_ jack	_213_ javac	Geometric Mean
IBM JDK 1.3.0	25.7	21.7	14.4	12.2	33.0	28.6	12.6	19.7
HotSpot Server VM	25.7	27.3	14.0	9.77	26.8	37.1	11.1	19.5
OpenJIT 1.1.15	8.16	8.70	9.88	3.71	11.1	9.06	4.34	7.33
shuJIT	5.18	7.99	9.16	3.83	10.7	7.12	3.51	6.29
JBuilder JIT	9.44	8.43	10.3	3.10	13.5	2.54	3.77	6.14
TYA 1.7v2	5.79	7.08	7.55	3.39	8.90	6.23	3.75	5.79
sunwjit.so	2.32	3.06	9.24	計測不能	8.52	5.91	3.44	4.74
Kaffe 1.0.6	3.29	2.47	18.0	4.44	7.96	2.05	2.82	4.36
インタプリタ	2.58	2.69	1.95	1.72	2.19	2.29	2.01	2.18

表 1: SPEC JVM98 の結果 (K6–2 / 400 MHz)

ることになる．利用者からの報告で，開発者が気付いていなかった問題が明らかになることは，残念ながら非常に多い．

shuJIT については，これまで数十件の問題報告を受けた．そのほとんどが，私が気付いていなかった問題であり，これらの報告が質の向上に大きく役立った．とはいえ，これらの報告に対してひとりですべて対応することは負担でもあった．アイディアの実証段階で発覚する問題，例えば，ベンチマークプログラムで発露する問題には，利用者からの報告を受けるまでもなく対処できる．ところが，利用者から報告される問題は，アイディア実証の段階で発覚しなかった，つまり，アイディア実証だけが目的であるなら対処する必要もない問題なのである．しかも，対処には，問題によっては数日を要した．

しかし，利用者からの報告に対応しないと，信頼を失いかねない．問題報告までしてくれる利用者は稀であり貴重なので，質の向上のためにはつなぎ留めておくことが望ましい．

3.2 再現が困難な問題報告

私の手もとでは問題の再現が困難な報告を数件受け取った．例えば，JavaServer Pages(JSP) や Servlet, Object Request Broker(ORB) などが連係する比較的複雑なシステムで問題が発生していて，こちらで同じ環境を構成するには大きな手間がかかったり，またはこちらにそれらの利用経験がない場合である．

正常に動作しているように見えて 1 日に 1，2 回異常終了する，という報告を受けたことがある．当の利用者すら問題を再現できなかったため，対処は断念した．その後の他の修整によって，その問題も解決している可能性はある．

こういった，再現が困難な問題に対処するために，異常終了時には，原因追求の助けとなる情報を可能な限り多く出力するようにしておくことが望ましい．開発者側で再現できなくとも，その出力を元に解決できることが理想である．とはいえ，場合によっては，異常終了時のメモリダンプ（`core` ファイル）とデバッガで充分であろう．shuJIT は，異常終了時には，各レジスタの値とプログラムカウンタの値，また，その周辺のメモリの内容，その時点で実行されていたメソッドの名前を出力する．古い版を使う利用者がいることも考えると，バージョンナンバも出力することが望ましい．

3.3 shuJIT を基盤とした研究

Matt Welsh は，JIT コンパイラを利用した研究のアイディアを知らせてきた．単に Java プログラムの高速実行に利用するのではなく，JIT コンパイラが生成するコードを変えることで，通信や入出力の性能を改善しようというものであった [9]．分散オブジェクトへの応用 [10] と同様のアイディアであり，Java 言語に多次元配列 [4] が導入された場合にも必要となる技法である．彼は，実装の基盤として用いる JIT コンパイラを選ぶ際，TYA [2] と shuJIT を比較し，実装の労力が少なく済みそうという理由から shuJIT を選んだ．

このアイディアは実際に shuJIT に実装され，後には，OpenJIT [5] と GCJ [1] に実装された．多くの JIT コンパイラはソースコードが手に入らない中，これらの処理系はソースコードが公開されている．それによって，JIT コンパイラを基盤とした諸研究が可能となったのである．

3.4 ライセンス条項の変更依頼

ライセンス条項を GNU general public license（GPL）から，より制約の緩い，例えばBSD や X Window System のようなものに変更して欲しいという要請を受けたことがある．NetBSD の配布 CD-ROM に収録するために GPL を避けたい，という理由であった．

shuJIT は GPL のもとに配布されている．このライセンス条項は，誰でも複製物を頒布できること，ソースコードを入手できることを目的としている．そのため，利用者側で変更を加えたものを配布する場合にはその変更内容も公開しなければならないなどの制約があり，商用の妨げとなる場合がある．

ソフトウェアを配布する際，著作者は，ライセンス条項を規定することで，そのソフトウェアがどのように利用されるのかをある程度規定することができる．法的な検討の末に作られた既存の条項もいくつもあるので [3]，それらから選択することで，自作の条項に対して意に反した解釈をされてしまう危険を避けることもできる．

ライセンス条項の決定は，そのソフトウェアがどう利用されて欲しいかについての意志表明である．また，あるライセンス条項を選ぶということは，その条項に票を投じるという，支持表明の意味もある．

3.5 todo リストの英訳依頼

todo リストは，to do，すなわち為すべきことを書いた一覧表の通称であり，開発者の備忘録である．未だ達成されていない課題が書かれる．shuJIT では日本語で書いている．日本語を読めない利用者から，これを英語で書いて欲しいという要望を受け取った．

todo リストを読みたいということは，そのソフトウェアがまだ出来ないことを知りたいか，開発をしたいかのどちらかであろう．もし後者であれば，その利用者は開発者になってくれるかもしれない．日本語圏と英語圏で開発者候補の数を比較すると英語圏の方が多いであろうから，もし英語圏の開発者を巻き込みたいのであれば，開発者が興味を持ちそうな文書は英語で書いておくことが望ましい．

このように，ソフトウェア公開の際は，各種文書をどの言語で用意するかを検討しておくとよいだろう．すべて英語で用意すべきとは主張しない．英語以外を母国語とする利用者，開発者を逃す可能性もあるので，著作者，開発者の戦略を反映して決めればよいと考える．

3.6 JIT コンパイラ開発の誘い

SGI 社，Java グループのマネージャから，TYA の開発者 Albrecht Kleine と私に対して，JIT コンパイラを開発しないかというメイルがあった．SGI 社の MIPS 用 JIT コンパイラを元に IA-32 用を開発しないかという誘いであった．

結局は参加しなかったものの，こういった協業の提案もソフトウェア公開の結果である．受ける提案の内容は，もともと興味があって作成，公開しているソフトウェアに関係しているはずなので，やはり興味深いものであることが多いだろう．自分が何をしていて，何に興味を持っているのかを社会に伝える手段としても，ソフトウェアの公開は有効であった．

4 経験のまとめ

4.1 労力

よく言われるように，動作した，というアイディアの実証段階から，公開を経て，実用になる段階まで質を向上させることには大きな労力を要した．報告のあった数十を含めた問題への対処，問題報告への返答だけでなく，コンパイル方法をはじめとする各種文書の用意，48 回のリリースのたびに行った復帰試験（regression test）にもかなりの時間と労力を費した．

これらは新規性が高くないため，研究者は雑用と考えがちな仕事である．また，JIT コンパイラの開発作業ではあるものの，アイディアの実証という範囲を大きく超える作業も多かった．例えば，Java 仮想マシンの仕様に従わせるための改修作業である．厳密に従った実装とするためには手間がかかり，かつ，実装法によっては性能が犠牲になってしまうような仕様がある．コンパイル手法，最適化手法を実証することだけが目的なら，厳密に従わずともまず問題にはなることはまずない．しかし，仕様にいかに忠実であるかは，品質の重要な評価基準である．手間をかけて，厳密，かつ，性能低下のほとんどない仕様を実装した．

ひとつの例は，クラスとインタフェースが初期化されるタイミングである．仕様は，そのクラスのインスタンスが作成された時，そのクラスの静的メソッドが呼ばれた時，静的変数への代入か使用があった時，と

定めている．これに厳密に従おうとすると，インスタンスの作成，静的メソッドの呼び出し，静的変数への代入と使用時に，クラスが初期化されているかどうかを調べて，もしまだならば，初期化の処理を呼び出さねばならない．静的変数アクセスのたびに調べるのはいかにも効率が悪く，性能は下がる．JIT コンパイラ開発者にとって実装が楽なのは，JIT コンパイル時にクラスを初期化してしまう方法だが，これでは仕様を守れない．性能を犠牲にせずに仕様に従うためには，コードの自己書き換え，トラップ（シグナル）の使用といった技法を用いねばならず，実装には手間がかかるのである．

しかしながら，JIT コンパイラにも，実用化の労力について有利な点はある．ユーザインタフェース（UI）が不要であるという点である．JIT コンパイラは Java 仮想マシンから呼び出されるため，利用者から直接操作されることがない．逆に，JIT コンパイラの有無や種類によって，ユーザプログラムの挙動が変わってはいけないわけである．利用者からソースコードを直接受け取るという通常のコンパイラには，プログラムの誤りをいかに判りやすく報告するか，または修整するかといった UI の問題が付きまとう．JIT コンパイラ開発にはこの問題はない．

4.2 新規性

本稿で述べてきた類の作業の多くは，技術的，理論的に何か新規性のある活動ではない．アイディアの実証という段階を過ぎている．私個人は，博士後期課程を通してのこの開発に満足しているものの，誰にでも，特に研究者たろうとする学生に奨められる活動ではない．

4.3 Test Suite

ソフトウェアの質を向上させていくためには，充実したテストコード群が欠かせない．特に言語処理系はその入力，つまりソースコードが採り得るパターンが膨大なので，試験のためのプログラムはいくらあっても足りない．

shuJIT では，自前で用意した数十のテストコード（未公開），Java 仮想マシン Kaffe に含まれるテストコード，Java コンパイラ `javac`，Linpack ベンチマーク，統合開発環境や文書作成ソフトウェア（一太郎 Ark）などの大規模なソフトウェア，適当な Java アプレットなどを用いて試験を行ってきた．特に，新しい版をリリースする際はひと通りの試験を行い，機能追加や修整によって新たな問題が発生していないかを確認した．これを復帰試験（regression test）という．

それでも発見できなかった問題は多い．Sun 社は Java Compatibility Kit（JCK）という大規模なテストコード群を持っていて，Java のライセンスを与えた企業，団体に対して，出荷前に JCK をパスすることを義務付けている．我々は Sun 社と何ら契約をしていないため，JCK は入手できていない．しかし，1999 年 9 月，SGI 社の開発者が shuJIT を JCK で試験した結果を知らせてくれた．‘VM tests’ では，2437 あるテストのうち 100 で失敗，‘Lang test’ では，2561 あるテストのうち 182 で失敗という惨澹たる結果であった．しかし我々は JCK を持っていないため，問題を修整することはできなかった．合計 300 近い問題のうち，いくつが今なお残っているかも不明である．充実したテストコード群の重要性を非常に強く感じた．

4.4 第一印象

ソフトウェアも人と同じで，第一印象が大切であると考える．自らの利用者としての経験から言うと，初めて試した際に悪い印象を受けたソフトウェアは，なかなか再び試そうとは思い立たない．

開発者としての経験から言うと，やはり初期の印象は大切である．ソフトウェアが，一旦，論文，記事，web ページなどで紹介されると，その文章は長く残る．性能が重要である JIT コンパイラでは例えば，ベンチマークの結果が web ページなどに掲載される．開発者にしてみると非常に古い版でのベンチマーク結果が web ページに残り続け，参照，紹介され続けることがあるのである．実際に，1999 年 2 月のベンチマーク結果が，メイリングリスト（java-linux）で 2001 年になってから紹介された．

多人数で質を向上させていこうという方針とは相反するが，ソフトウェアや付属文書，web ページの質はある程度以上に高めてから，公開，配布に踏み切った方が良いと考えている．

4.5 ソースコードの公開

shuJIT は，コンパイル済みバイナリだけでなく，ソースコードも公開している．バイナリを公開する

だけでも，実際に稼働していることの他者に対する証明にはなる．しかし，もしソースコードを公開しなければ，3.3 節で述べた，shuJIT を基盤とした他者による研究はなされ得なかった．他の研究者から，参考になった，という声を聞くこともなかっただろう．

反面，ソースコードを公開する際は，意図しない利用法をされないように，ライセンス条項をより慎重に策定，もしくは選択する必要があるだろう．

5 まとめ

一研究者が，実用化を目指してソフトウェアを公開，配布し，その質を向上させてきた 2 年半の経験を述べた．ソフトウェア公開の，以下のメリットを実感した．

- 実用化，配布することで，**品質の向上**に非常に有効な，利用者からの問題報告を得られる．
- ソフトウェアの公開には，自分達の活動，興味の対象を**広報**する効果がある．
- 開発に関係する文書も公開することで，他の開発者を巻き込める可能性がある．

また，以下の教訓を得た．

- 実用化，配付には，アイディアの実証までに必要な労力よりはるかに大きな労力を要する．
- 再現が困難な問題の解決まで視野に入れて，問題の追跡に有用な情報はなるべく多く出力するようにする．
- ソースコードは他の研究者，開発者の参考になる．
- どのような文書を用意するか，例えば，何語で記述するかによって，巻き込む利用者，開発者を選択できる．
- 品質の向上には，充実したテストコード群が欠かせない．
- ソフトウェアも第一印象が大切であり，ある程度以上の質に達してから公開した方が良い．

ソフトウェアの実用化，配布，質の向上には大きな労力が伴うが，多くの利用者，開発者，研究者とのインタラクションは刺激的であり，得るものも多い．その労力を無駄と考えず，研究と実践的なソフトウェア開発の両立を目指していきたい．

参考文献

[1] The gnu compiler for the java programming language. http://gcc.gnu.org/java/.

[2] Albrecht Kleine. TYA Archive. http://sax.sax.de/~ adlibit/.

[3] さまざまなライセンスとそれらについての解説. http://www.gnu.org/philosophy/license-list.ja.html.

[4] J. E. Moreira, S. P. Midkiff, M. Gupta, P. V. Artigas, M. Snir, , and R. D. Lawrence. Java programming for high-performance numerical computing. *IBM Systems Journals, Java Performance Issue*, Vol. 39, No. 1, February 2000.

[5] H. Ogawa, K. Shimura, S. Matsuoka, F. Maruyama, Y. Sohda, and Y. Kimura. OpenJIT: An open-ended, reflective JIT compile framework for java. In *Proc. of 14th European Conference on Object-Oriented Programming (ECOOP '2000)*, June 2000.

[6] Kazuyuki Shudo. shuJIT—JIT compiler for Sun JVM/x86, 1998. http://www.shudo.net/jit/.

[7] Kazuyuki Shudo and Yoichi Muraoka. Efficient implementation of strict floating-point semantics. In *Proc. of 2nd Workshop on Java for High-Performance Computing (in conj. with ICS'00)*, pp. 27–38, May 2000.

[8] Standard Performance Evaluation Corporation. SPEC JVM98, 1998. http://www.spec.org/osg/jvm98/.

[9] Matt Welsh and David Culler. Jaguar: Enabling efficient communication and I/O from Java. In *Concurrency: Practice and Experience, Special Issue on Java for High-Performance Applications*, December 1999.

[10] 首藤一幸, 根山亮, 村岡洋一. プログラマに単一マシンビューを提供する分散オブジェクトシステムの実現. 情報処理学会論文誌:プログラミング, Vol. 40, No. SIG 7, pp. 66–79, August 1999.