NRI IT Infra 基盤は進化する
技術創発 －NRI情報技術レポート－
2013年2月発行

Javaによるバッチシステムの開発

野村総合研究所
共通基盤推進部 上級テクニカルエンジニア

富岡 優（とみおか まさる）

情報技術本部にて、JavaEEによるフレームワークの開発や、社内外プロジェクトにおける技術支援を行っている。特に、Javaによる開発生産性向上のための仕組みの考案や普及を担当。

要旨

企業システムの中心の一つであるバッチ処理は、クライアントサーバシステムや Web システムを中心としたオンラインシステムと比較すると、技術革新による恩恵が薄く、近年でも COBOL によるバッチプログラムとシェルによる開発が未だに主流となっている。

多くの企業においては、COBOL とシェルによる旧来のバッチ処理と、Java を中心とした最新のオンラインシステムが混在し、開発業務だけでなくエンハンス時においても複数のシステムの混在による TCO(Total Cost of Ownership：総所有コスト)の高止まりが経営の観点から課題となってきている。

スマートフォンや HTML5 などのユーザエクスペリエンス技術のライフサイクルとは対極に、10 年以上同じバッチシステムを継続利用しているプロジェクトの実例は多い。バッチ処理技術を革新し、新たなバッチシステムを築き上げるために、Java によるバッチ処理基盤の最新技術動向を踏まえて企業システムへの適応方法を考察する。

キーワード：Java，バッチ，Spring Batch，JavaEE7，JSR-352

1．はじめに

近年、Java バッチ分野では Spring Batch 、JSR-352 といったデファクトスタンダード、標準技術の台頭がみられ、信頼性や将来性の面での大きな障壁は無くなりつつある。Java 言語のメリット・デメリットを正しく認識することで、バッチアプリケーションの開発で旧来からの COBOL やシェルを”仕方がなく使う”といった状況から脱却し、次世代のバッチシステムとして戦略的に Java を採用することができるようになった。

本稿では、Java によるバッチシステムの構築に取り組むため、課題や問題点を整理し、ユーザ要件に適した Java バッチの適応方法を考察する。

2．バッチシステムに Java を採用すべきか？

一般的なバッチシステムにおいて、COBOL とシェルといった単純なプログラムのみの連結でシステム要件を満たす場合には、多用途な商用の運用ツール(例えば、野村総合研究所 Senju、富士通 Systemwalker、IBM Tivoli、日立製作所 JP1 等)と組み合わせることで、可用性・信頼性・保守性を満たすバッチシステムを構築することが可能である。

これに対し、帳票作成、グラフ作成、暗号化、各種計算、データ分析、Web サービス通信、メール送信などの複雑な要件を実現する場合には、旧来の単純なプログラムの連結ではなく、Java における複雑なバッチアプリケーションプログラムの開発も選択肢として挙がってくる。

（1）Java を選択する理由

企業の IT 部門のニーズとして、バッチシステムを計画する場合には、以下の要件で Java 言語の選択が俎上に載る場合が多い。

① オンラインシステムとの統合

Java 言語のメリットとしてまず挙げられるのは、「マルチプラットフォームでの稼働」「ソフトウェアの高い開発生産性」「安全なコード」「言語がデファクトスタンダードであることによる豊富なライブラリやプログラマの練度の高さ」といった特性になる。

このメリットにより、オンラインシステムの構築においては、すでに Java による開発が主流となり、アプリケーション開発の第 1 候補として挙げられる。

Java 技術者やアプリケーション資産をバッチシステムでも有効利用できるのであれば、ソフトウェアの開発要員や保守要員を効率的にオンラインシステムと共有し、開発生産性向上や保守向けのリソースの最適化を図ることができる。

特に近年は、COBOL 技術者が開発現場で上級技術者として活躍する場合が多く、低コストの開発体制を確保することが困難となっており、

今後10年間の維持管理業務を含めるとTCOの肥大化の大きな懸念材料となる。

② システムの移行性の確保

基幹システムの再構築時に必ずといって良いほど問題となるのが、現有ソフトウェア資産の再利用である。実際の現場においても、当初の意図に反してプラットフォームに依存したアプリケーションの再利用が進まず、マイグレーションコストの増加により基幹システム再構築が行き詰まり、旧システムとの併存をやむをえず採用する場合も少なくない。

開発言語にJavaを利用する場合は、プラットフォームに依存しない言語の特性を生かすことにより、10年後のシステム更改においても移行性を確保することができる。

（２）Javaを選択しない理由

これらのメリットの裏返しとして、Java言語によるバッチアプリケーション開発を推奨できないケースも明らかに存在している。

① バッチ処理だけをJavaで構築する場合

オンライン処理はJavaでの開発であれば、開発リソース(要員・開発環境・ライブラリ)面でのメリットは高いが、軽量プログラミング言語(PHP、Perl、Python、Rubyなど)や.NET、C言語でオンラインシステム開発を行う場合に、バッチシステムのみをJavaで開発することは開発リソースの有効利用の面から適しているとは言えない。

② 処理性能、大量業務処理

Javaによるプログラムの特徴として、実行時に足かせとなるのが、処理速度や大量のメモリ管理処理がある。例えば、Javaで作られたプログラムは、JVM(Java仮想計算機)上で稼働するため、C言語のようにハードウェアを直接操作する命令は一切利用できず、ディスクI/O、メモリ操作すべてにおいて、JVM特有のオーバーヘッドが処理性能を劣化させる。また、バッチ処理で日本語文字コード(SJIS・EUC・JIS)を扱う場合、Java言語ではUnicodeが標準であるため、文字コード変換処理による性能劣化が常に性能問題の一因となる。

さらに、数十GB～TBの大量データを処理する場合には、JVMが管理するメモリ空間の利用サイズやメモリの直接操作に制約があり、Java特有のメモリサイジング、メモリ管理を考慮したプログラミング技術が必要となる。

③ 過去のバッチアプリケーション資産の存在

旧来のCOBOLやC、シェル他で開発されたバッチアプリケーション資産に対する移植性の少なさもJavaへの移行を阻害する大きな障壁となりえる。

単にプログラムのソースコードだけではなく、

設計開発方法、設計ツール、開発ツール、開発標準化、テストケース、テストデータ、運用リリースシステム、開発メンバーへの教育を含めたすべてが旧バッチシステムの資産であり、新しいバッチシステムへの資産移管がどの程度可能かを正しく評価する必要がある。

今まで述べてきたデメリットを許容できない場合は、Java によるバッチシステムの採用は慎重に行う必要がある。場合によっては、Java によるバッチシステムを採用せず、旧来の COBOL や C、シェル他を利用したバッチシステムを踏襲するか、または大量高速処理においては、Hadoop やNoSQL などの特定処理に特化した技術を採用することも検討すべきである。(表１参照)

３．Java バッチ フレームワーク

オンラインで Java を利用する多くのプロジェクトでは、開発生産コストやエンハンスフェーズのコスト面から Java バッチ処理を採用する動機はあるものの、現時点では大きな障壁がある。

なぜなら、単にバッチ処理方式として必要な機能要件を満たせばよい訳ではなく、非機能要

業務観点	Java バッチシステムに向く業務処理	Java バッチシステムに向かない業務処理
	帳票・画像・暗号化処理 統計分析などの複雑な計算 外部接続などの通信処理 など	集計やマッチングなど単純定型処理 固定長文字列操作処理 大量データ(ビッグデータ)処理 10 進数固定小数点計算 など
経営観点	Java バッチシステム基盤のメリット	Java バッチシステム基盤のデメリット
	システム移行に伴うポータビリティに優れる オンラインとのプログラム統合によるコスト削減	過去バッチ資産の継承、有効活用ができない
開発容易性 生産性観点	Java 言語の特性によるメリット	Java 言語の特性によるデメリット
	シンプルで安全な言語仕様 容易なマルチスレッド処理 オブジェクト指向によるプログラム再利用性 豊富なライブラリ、開発環境 など	コンパイル言語と比較して低速な処理性能 JVM 依存したメモリ操作 など

表1：バッチシステムにJava言語の採用を検討する際の考慮ポイント

件を実現するための仕組み、開発生産性を高めるための仕組み、品質を確保するための仕組み、処理速度を向上させるための部品などをシステム要件の特性に応じて準備する必要があるからである。

小規模のバッチシステムであれば単純なJava アプリケーションと運用ツールによって構成することが可能であるが、大規模な業務要件を満たす全体のバッチシステムとして運用するには、機能要件・非機能要件ともに満たす必要があり、これらを含めて新規開発を行うことは膨大なコストを必要とする。大規模なバッチアプリケーション開発では、必要となる機能を補う Java バッチフレームワークの積極的な活用を検討しなくてはならない。

Java バッチフレームワークは、バッチアプリケーションを「業務処理」「基盤機能」に分割し、バッチアプリケーション開発時には「業務処理」だけを開発するように「基盤機能」を提供するミドルウェアであり、一般的には「表 2 Java バッチシステムの主な機能要件」のような機能を提供する。

商用の Java バッチフレームワークの例としては NTT データの TERASOLUNA、NEC の WebOTX などがあり、オープンソースの例としては Spring Batch がある。

Spring Batch は現在世界中で最も採用されているフレームワークの一つであり、それをベースにしたJSR-352の規格が今後はJavaの標準として採用される可能性が高い。

特にバッチシステムのような長期間の耐用性が求められるシステム開発においては、特定のベンダー依存のソフトウェアではなく、デファクトスタンダードや標準規格を採用することで将来のエンハンス時にメリットを享受可能となる。

以下に代表的な Java バッチフレームワークのうち、Spring Batch および JSR-352 の規格に関して説明する。

（1）Spring Batch

Spring Batch は、2008 年 4 月に 1.0.0 版がリリースされて以降、世界中で最も普及しているオープンソースの Java バッチフレームワークの一つである。現在は最新バージョン 2.1.9 が Apache2 ライセンスで公開されている。

Spring Batch は Spring Framework の一部として開発され、AOP(Aspect Oriented Programming)フレームワーク、DI(Dependency Injection)コンテナを中心とした、POJO(Plain Old Java Object)ベースの開発方法を継承してバッチアプリケーションの開発を行う。

Spring Batch は Java オンラインフレームワークと同じように、ミッションクリティカルなエンタープライズアプリケーション開発においても、開発者の手間を省き、堅牢なアプリケーションを開発するために用意された Java バッチフレームワークである。

要件	Java バッチシステムで必要となる基盤機能	機能名
機能	XML などによりジョブフロー、ステップを定義する機能	ジョブフロー定義
	コマンドラインからのジョブ実行機能	コマンドライン起動
	オンラインからのバッチ呼び出し、バッチからのオンライン呼び出し	オンライン起動
	フラットファイルや CSV、TSV データ入出力部品	ファイル入出力機能
	実行結果による条件分岐設定機能	実行順序制御
	DB のデータ入出力部品 、O/R マッパー機能	DB 入出力
	入力データのバリデーション機能	バリデーション
可用性	定時実行・中断・再ランなど定時運用に伴うスケジュール機能	ジョブスケジュール機能
	実行中ジョブの中断・中断ポイントからの再実行機能	ジョブ中断・再実行機能
	ファイル排他制御・世代管理機能	ファイル管理
	サーバ間での分散実行機能	分散処理
	障害時の自動リトライ機能	自動リトライ機能
運用	ジョブ実行ステータスを DB などに貯めて管理する機能	ジョブステータス管理
	ログ出力部品、ジョブ番号、ステップ番号出力機能	ログ出力
	トランザクション制御部品、分散トランザクション制御機能	トランザクション制御
性能/拡張性	バッチロジック（ジョブステップ）の並列実行機能	並列実行
	性能向上のための JVM 常駐化機能	JVM 常駐化
	DB コネクションプーリング機能	DB コネクションプール
	byte 配列操作機能、NIO、NIO2 によるファイル I/O 高速化機能	ファイル IO 高速化
	中間ファイル・DB を使わずメモリ上の情報によりステップ間の連携を行う機能	ステップ間データ連携
	サーバ間での分散実行機能	サーバ間分散処理
セキュリティ	特定業務の監査ログを出力する。	監査ログ出力機能
生産性/品質	コントロールブレイク、マッチングなどの定型処理部品 ・テンプレート	定型処理部品
	GUI エディタによるジョブフロー設計	GUI ジョブフロー設計
	ソースコード雛型、定義ファイルなどの自動生成	ファイル自動生成
	Junit、モックフレームワークなどのテスト自動化支援機能	テスト自動化支援

表2 Javaバッチシステムの主な機能要件

Spring Batch の特徴として、データの入出力、データの加工や整形などに特化した役割分担が明確なアプリケーション開発手法、柔軟な拡張性や実行制御、それらを実現する非常に堅牢なジョブやステップの状態管理システムが挙げられる。

Spring Batch は、1 つのバッチ業務処理を構成する単位をジョブ(Job)とし、ジョブには 1 つもしくは複数のバッチ処理プログラム単位であるステップ(Step)の定義を行う。ステップは、データ入力処理を行う ItemReader、データの変換や編集を行う ItemProcessor、データの出力処理を行う ItemWriter の3つの処理から構成される。(図 1)

また、Spring Batch の DI／AOP 機能により、アプリケーション開発者は複雑な業務要件を満たすバッチアプリケーションの開発にあたっても、簡潔で保守性の高い実装を実現することが出来る。

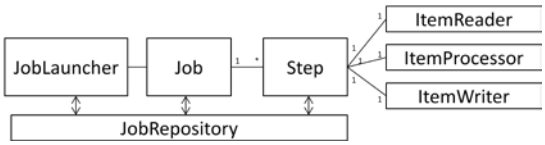

図1 Spring Batch アーキテクチャ

出所：
http://static.springsource.org/spring-batch/reference/html/domain.html

（２）Batch Applications for the JAVA Platform 1.0 （JSR-352 ）

Spring Batch のようなオープンソースの Java バッチフレームワークや、ベンダーによるさまざまな Java バッチフレームワークが登場し企業システムで利用されてきているが、JavaEE (Java Enterprise Edition ：企業用の機能セット) として、Java におけるバッチの標準化仕様を策定しようという動きが近年になってようやく活発化され、2013 年第2四半期に JavaEE7 の一部として仕様の正式リリースが計画されている。

このバッチ仕様の仕様策定書が JSR-352 (Java Specific Request #352) として、JCP(Java 標準化プロセス) で検討が進んでいる。2013 年1月現在、パブリックレビュー投票を通過しており、最終ドラフト案を策定の上、最終投票に進む予定である。

JSR-352 の一番の特徴としては、デファクトスタンダードである Spring Batch の基本構成をほぼそのまま踏襲していることにある。JSR-352 では ジョブやステップの構造、ItemReader、ItemProcessor、ItemWriter といったプログラムアーキテクチャは Spring Batch の基本構成とほぼ同じである。(図 2)

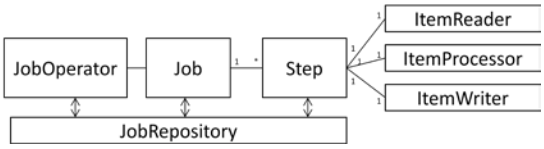

図2 JSR-352アーキテクチャ

出所：『JSR-352 Batch Applications for the Java Platform Public Draft 』P.10

JSR-352 の考え方は、Spring Batch と同じく、「ジョブ」「ステップ」の考え方に基づいて、バッチアプリケーションの処理フローを適切かつ明確に分割し、インターフェイスとサービス層の分離、および強化された拡張性を保持している。

Java のプログラムでありながら JCL のようなステップ間の制御を自動化するためのアーキテクチャが Spring Batch と同様に基本となっている。

JSR-352 は、Spring Batch の Java バッチシステムとしてのメリットはそのまま継承されており、さらに各ベンダーから JavaEE7 に完全準拠した商用アプリケーションサーバがリリースされれば、これらの Spring Batch に相当する基盤機能の多くは、JavaEE を実装したミドルウェア製品の商用サポートを受けられることになる。

サポートの不安からミッションクリティカルな基盤へのオープンソースソフトウェアの導入を躊躇している企業システムにおいても、JavaEE 標準規格として統一されることによって大きな恩恵を受けることができる。

[注意] JSR-352 は 2013 年 1 月時点では、仕様のドラフトのみ公開されており、実際の最終仕様やアプリケーションサーバへの実装方法に関しては本節の説明と異なる可能性がある。

4. Java バッチシステムへの移行

既存の COBOL やシェルを中心としたバッチシステムから、Java バッチシステムへ移行するには、さまざまな障壁がある。現時点では Java バッチシステムの要望の増加に反して、速やかに導入が進んでいるとは言い難く、「COBOL を仕方がなく継続利用」「実現できない一部だけ Java を採用」といった状況になる場合が多い。Java バッチシステムへの速やかな移行を実現するためには、Java バッチフレームワークの利用だけでなく、以下のような施策により障壁を取り除く必要がある。

（1）設計資産・スキルの再利用

バッチアプリケーションの多くは、集計、マスターマッチング、変換といった一連の単純処理の集積によって構成される。これらのシステムを単純に Java にプログラム移行する場合には、オブジェクト指向に従ったクラス設計、データモデルの設計、デザインパターンの適応といった旧来の開発方式と異なる Java 特有の設計スキル・開発スキルが必要となり、過去の設計資産などを有効に利用できない問題がある。

これに対して、Spring Batch と JSR-352 では、JCL における設計方法と同じく、ジョブとステップのインターフェイスを設け、最小単位の Java アプリケーションのフロー管理を行うアーキテクチャを採用している。

アプリケーション開発者は、最小単位の Java

アプリケーション(ステップ)の開発を行い、さらにジョブ内のステップのフローや外部リソースの物理情報を定義することでジョブの設計を行う(図３)。極力今までの資産(設計情報や要員スキル)を再利用しつつ、明解なバッチアプリケーション設計を新しい Java バッチフレームワークでも同様に行うことが出来る。

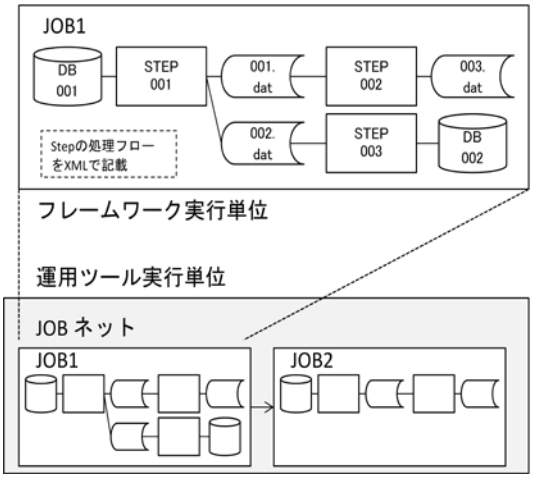

図3　Javaバッチでのジョブとステップの関係

（２）Java の性能問題への対処

多くの企業システムにおいては、取扱うデータ量の増加、ビジネスの複雑化に直面し、バッチの処理時間増加の課題に対しては慎重になる場合が多い。また、「2-（2）-② 処理性能、大量業務処理」で述べたような Java 言語固有の処理性能問題に対する漠然とした不安は非常に強い。

特に性能要件の厳しいシステムにおける Java の採用にあたっては、業務処理方式に応じた処理性能向上施策をあらかじめ準備する必要がある。

以下に Java バッチの高速化施策の事例を紹介する。

① JVM 常駐化

JVM 起動時のオーバーヘッドを無くし、外部リソースのキャッシュを利用するために、JVM の常駐化を行う。

② ステップ間パイプ機能

中間生成ファイルを作らず、ステップ間をパイプ(BlockingQueue)としてデータ連携を行う。

③ 外部コマンドの呼出し

大量データのソート処理など、ボトルネックとなる性能遅延箇所の一部を外部コマンドまたは JNI (Java Native Interface) として呼出す。

④ byte 配列処理機能

フラットファイル入出力処理において、Unicode への変換処理を極小化しbyte配列のまま集計・マッチング処理を行う。

5. まとめ

近年のバッチシステムでは、いわゆる集計・整形・マスターマッチング といった単純なデータ処理だけではなく、他システムとの通信連携、帳票・グラフの整形、セキュリティ要件に基づく暗号化処理、ビッグデータなどの高度な分析

処理の要件まで多様な機能が求められるようになってきている。旧来のCOBOLを中心としたバッチシステムのアーキテクチャの延長では、実現性や開発生産性が要求を満たせない場面に多々遭遇し、これらのアーキテクチャを革新していく潮流が続いている。また、経営層的観点からは、オンラインシステム・バッチシステムの共通化､標準化技術の採用による10年持つ基盤へのニーズは非常に強い。

Java 言語によるバッチシステムの構築はこれからの 10 年を見越した最適な解決策の一つと考えることができる。

Spring Batchの世界的な普及、JavaEE7におけるバッチ処理のサポートによって Java バッチ処理の仕組みはここ数年で爆発的な普及に向けた大きなターニングポイントにさしかかっているといえる。

ただ単に、Javaバッチフレームワークを導入しただけでは、旧来のバッチシステムとの設計思想ギャップや、Javaにおける処理速度のオーバーヘッドなどの課題を解決することが出来ず、Java によるメリットよりもデメリットが大きくなることも考えられる。

COBOL、シェルのプログラムだけでなく、標準化、設計方法、設計書、開発環境、テスト、教育、人的資源などの莫大な旧来のバッチアプリケーション資産を適切に継承する活動、処理性能をさらに向上させる機能を併せて導入することにより、Javaバッチシステムへの移管を柔軟に低コストで実現することができる。

●参考文献●

[1] Spring Batch-Reference Doumentation
http://static.springsource.org/spring-batch/

[2]JSR-352 Batch Applications for the Java Platform Public Draft
http://jcp.org/aboutJava/communityprocess/pr/jsr352/