バッチ処理およびオンライン非同期処理のための JavaフレームワークJBeXの開発と適用

松原 武司　原 亮吉　高石 稔

The Development and Adaptation of the Java Framework "JBeX" for Batch and Asynchronous Processing

Takeshi Matsubara, Ryokichi Hara and Minoru Takaishi

システム開発でJava™の採用が進むにつれ, Webアプリケーションに留まらず,「バッチ処理」や「オンライン非同期処理」の分野にもJavaを利用する事例が増えつつある. このとき開発基盤としてのJavaフレームワークの適用やコンポーネントの再利用による工数削減と品質向上が重要な課題になる. 筆者らは, バッチ処理だけでなくオンライン非同期処理にも適用可能なJavaフレームワークJBeX（Batch Execution and Control Environment for Java）を開発した. また, それを実プロジェクトに適用し工数削減と標準化による品質向上の効果を確認した.

As the use of Java™ as the programming language in system application development progresses, the examples of the use of Java in the field of "batch processing" and "online asynchronous processing" are increasing as well. It has been forecasted that the reduction of man-hours and the promotion of the improvement of quality by recycling applications and the component of the framework that is the basis for development become important problems. We developed the Java framework "Batch Execution and Control Environment for JAVA" (JBeX) which can be applied to not only batch processing but also online asynchronous processing. In addition, We applied it to several projects and confirmed the effect of the improvement in quality by man-hour reduction and standardization.

Key Words & Phrases：バッチ処理, 非同期処理, Java, フレームワーク, アセット, サービスコンポーネント
batch processing, asynchronous processing, Java, framework, asset, service componentt

1. はじめに

システム開発においてJava™の採用が進むにつれ, 現在の主要な適用分野であるJ2EEベースのWebアプリケーションに留まらず,「オンラインWebアプリケーションと連携する非同期処理」にはじまり, 旧来の「定時で起動するバッチ処理」の分野にもJavaの利用を選択する事例が増えている［1］［2］. その背景には, ①開発言語をJavaで統一しJ2EEコンポーネント（EJB, Webサービス等）の共有や既存Javaライブラリーの再利用を行いたいという要求, ②COBOLやC言語のスキルを持つ開発者の減少, ③Java言語スキル保持者数の増加, ④JavaVMの高速化などがある. このJavaによるバッチ処理やオンライン非同期処理の分野については, 実現のための要素技術は従来から存在していたが, その開発を支援し汎用的にプロジェクトで利用可能なソフトウェア資産としてのフレームワークは少ない. そのためシステム開発プロジェクトにおいて, 独自に要件の分析とソリューション開発を行っているケースが散見されていた.

一方で昨今のシステム開発では開発期間の短縮と品質向上の両立が課題であり, その解決のため, 既存資産である「アセット」の再利用による生産性と品質の両面での向上が期待されている.

筆者らは, バッチ処理やオンライン非同期処理の分野のシステム開発において, 生産性と品質の向上を実現するために, 汎用的なJava開発フレームワークJBeX（Batch Execution and Control Environment for Java）をサービスコンポーネントとして開発して実用化した.

本論文では, JBeXの設計上の考慮点と有効性の実証について報告する.

提出日：2007年9月3日　　再提出日：2008年1月9日

以下, 2章でJavaによるバッチ処理とオンライン非同期処理のためのフレームワークの要件と課題をまとめ, 3章でその要件に応えるJavaフレームワークJBeXの開発上の工夫と特徴を述べる. 4章ではJBeXの適用事例を紹介し, その有効性を評価する.

2. バッチ処理およびオンライン非同期処理のフレームワークに関する要件と課題

2.1 バッチ処理とオンライン非同期処理の両方に利用可能なJavaフレームワークの必要性

本論文ではバッチ処理とオンライン非同期処理を, 処理データ量や完了までに要する時間や起動トリガーなどの特性の違いから両者を区別しているが, 一方で「非同期的に何らかの業務処理を実行し, その状態と結果を後で参照可能な状態とする」という処理の本質においては同一である. このため, 同一のプログラムを状況に応じてそれぞれの処理形態で実行したいという要求の発生は自然な流れであり, 現実にJ2EEの普及と共にバッチ処理の中でその処理の一部としてオンライン処理と同様にEJBやWebサービスの呼び出しを行う, あるいはその逆にオンライン処理から処理負荷のオフロードを目的に非同期処理としてバッチ的な処理を実行する[3]など, これまでのデータベースやファイルへの入出力が中心の処理とは異なる要求が増えている.

この要求に対応しつつ複数のプロジェクトでの開発を効率化し, 工数削減と品質向上を実現するためには, システムにおいて両処理形態の開発に利用可能な共通基盤としてのJavaフレームワークが必要である.

2.2 フレームワークへの要件と課題詳細

このJavaフレームワークに対する基本的な要件を, 機能要件(FR)と非機能要件(NFR)の側面から整理し, 以下の6項目に集約した.

[FR-1]:バッチ処理とオンライン非同期処理の両処理の実装を支援するAPIやユーティリティが提供され, 要求処理用に複数の手段が提供されること

バッチ処理とオンライン非同期処理の特性の違いに起因するそれぞれの要件を吸収しつつ, 実行中のチェックポイント取得や処理途中でのキャンセル要求に対し処理プログラム(ジョブ)が対応できる機構が必要である. また, 多様な手段で実行できるよう, 処理要求側(クライアント)を含めたAPIの提供が求められると共に, 実行時パフォーマンスの確保に有効なユーティリティも提供されることが必要になる.

[FR-2]:ジョブ管理ミドルウェア(スケジューラー)との連携利用が円滑に実現可能であること

システム運用基盤において, バッチ処理は新規または既存のジョブ管理ミドルウェアによって制御され, マシン間やマシン内のジョブネット構成が管理されることが多い. Javaでバッチ処理を実装する場合, 必然的にそれらと連携しジョブ管理ミドルウェアからの呼び出しに対応して結果情報を返却することが必要になる.

[FR-3]:処理プログラムが専用のコンテナー(実行環境)を必要とせず単体テストやスタンドアロンJavaプログラムとして実行可能であること

開発効率の向上とテストの容易性を確保するためには, 特殊な実行環境やミドルウェアが不要であることが求められる. また, 従来は“フレームワークを必要としない程度の規模や処理である”という理由で適切な標準化がなされないまま開発されることも多かったスタンドアロンJavaプログラムもフレームワークの適用範囲とし, その上で標準化の対象とするためにも重要である.

[NFR-1]:フレームワークがJ2SEとJ2EE[1]両方の環境で動作可能であること

開発の観点からは, 処理プログラムがどちらのJavaプラットフォームで実行するかに依らず, 共通のプログラミングモデルで開発可能であることが望ましい. 従って, 基盤としてのJavaフレームワークはJ2SEとJ2EE 双方の環境で動作しつつ両者の違いを吸収し, ジョブには共通の動作環境を提供することが求められる.

[NFR-2]:プロジェクトで求められる多様な要求に対応可能な拡張性が提供されること

アセットとして複数のプロジェクトで再利用されるためには, プロジェクト固有の機能要件を取り込み実装可能とする拡張性が必須である. 加えてデバッグや問題判別時の効率性, 最終的なカスタマイズ性の“担保”として, ソースコードレベルでのカスタマイズやソースコード開示が可能であることが求められる場合もある. このように, オンラインWebアプリケーション用のフレームワークと同様にバッチ処理とオンライン非同期処理用のフレームワークに対しても, ミドルウェア製品と異なり高いレベルのカスタマイズ性が要求される傾向が強い. ソースコードが原則的に非開示であることを含めソフトウェア製品ではカスタマイズ性に自ずと限界が生じるため,

1 J2EEは, Java2標準機能であるJ2SEにエンタープライズサーバー用機能が追加されている

その充実したサポート体制は評価しつつもこれまで独自の基盤開発を選択しなければならなかったようなプロジェクトにおいても, その要求に極力対応できなければならない.

[NFR-3]：バッチ処理とオンライン非同期処理の実行状態や結果を永続的に保持しつつ, 処理特性毎に同時実行数制御が実現可能なこと. 加えて, 常駐型の複数プロセス構成が利用可能なこと

Javaによるバッチ処理またはオンライン非同期処理の基盤は, JavaVMの起動というシステム資源に対して高コストな処理の負荷が頻発することを抑制しつつ, プロセス障害に対する可用性とスケーラビリティ確保のため複数プロセス構成の実現が求められる. これは, CやCOBOLなどで開発されたネイティブプログラムと比較した場合, Javaプログラムの仮想マシン上で動作するという性質やI/O処理性能において相対的に劣ることによってパフォーマンス上の問題が懸念される場合への対応手段として, バッチ処理にスレッドとプロセス両レベルでの並列処理構成を採用するために重要な点である.

3. JBeX開発上の考慮点と特徴

バッチ処理とオンライン非同期処理の両方の処理に対して汎用的に利用可能なJavaフレームワークを実現するに際し, 特に以下の点を考慮・工夫している.

3.1 J2SEとJ2EEの両方の環境で動作可能

この実現に重要なことは, 両環境による起動と停止操作時の違いとJ2EEにおける“ コンテナー ”の隠蔽を図らねばならないという点にある. J2EEではエンタープライズアプリケーション（EAR）という形式でアプリケーションのパッケージ形式が定められておりこれに準拠しなければならないが, J2SEではこのようなコンテナー自体が存在しない.

この問題を解決するために, JBeXはJ2SE環境ではJBeXサーバープロセスの開始（初期化）はスクリプトコマンドによって常駐Javaプロセスを起動し, 停止を含めた管理リクエストは固有の待ち受けポートへTCP/IP通信を行うという形態でフレームワークが動作する. 一方J2EE環境ではJ2EEコンテナー上にデプロイし実行可能なEARとして動作し, 開始（初期化）と終了は起動用サーブレットのinit ()/destroy () メソッドを利用し, Java環境による起動と終了の制御方法の違いを吸収する.

環境の違いに起因する考慮点としては他にもJ2EEの固有機能（管理されたJNDIツリー, J2EEトランザクション, セキュリティ管理機能などのJ2EEコンテキスト）を利用可能なスレッドの生成は通常のjava.lang.Threadクラスを用いて行うことができず, またそのようなJ2EEアプリケーションサーバーから見て“ 管理されない ”スレッドの生成は推奨されないという問題がある. この制約はアプリケーションに対してJ2SEとJ2EE環境の差異を隠蔽し, 両方の環境で動作する基盤環境を提供することが求められるフレームワークを設計する上ではとりわけ問題となる.

JBeXはこの問題を解決する手段として, Java仕様提案JSR-237 [7] を受けてIBMとBEA Systemsが共同で策定した仕様であるCommonJ WorkManager API [8] を利用している. 同APIはJ2EE環境におけるJ2EEコンテキストを保持したスレッド生成を標準化するためのAPI仕様であり, IBM WebSphere® Application Server V6.0以降でサポートされている. なお, このJSR-237は, 現在策定中の次期J2EE仕様であるJ2EE6（JSR-316 [9]）に含まれることが予定されており, 将来的にも標準化されたAPIとして利用が可能である.

JBeXの実行環境がいずれの環境で動作するにせよその違いは隠蔽され, 業務処理自体がJ2EEに直接依存しない限り, ジョブクラスはJ2SEとJ2EEどちらの環境においても同じフレームワークのもとで動作する.

また, J2SEとJ2EE環境の違いやバッチ処理かオンライン非同期処理かどうかに関わりなく, フレームワークは処理結果の保存と参照に対して責任を持ち, 後の処理状態や結果取得要求に応えねばならない. 一般に非同期処理の実現はメッセージ指向ミドルウェア（MOM）を用いた処理と関連付けて議論されることが多いが, MOMの“ メッセージ ”は本質的には永続的なデータではなく, アプリケーションがそのメッセージを取得すれば保存領域（キューマネージャー）から削除されてしまう. バッチやオンライン非同期処理のように結果データを永続的に保持し, その後の参照に対応することが求められる処理基盤の実装はMOMだけでは実現することができない.

このためJBeXは1つのジョブ実行要求の受け付けから実行完了まで, その状態と結果データを永続的に保存するための一元的なデータストアとしてRDBを利用しており, その上で処理特性ごとに同時実行数や優先度をそれぞれ分けて制御しつつ, 先入れ先出し方式（FIFO）によって実行制御を実現するためのキューはRDBのテーブルによって実装している. またそれによって同時に複数の常駐型Javaプロセスを並列で実行し, ジョブ実行における並列処理と可用性を実現することができるようにしている.

ジョブの実行は一定間隔でRDB上のテーブルをポーリングし，同時実行数を制御しつつジョブクラスをインスタンス化して業務処理を開始するアーキテクチャーを使用しているが，将来的にはポーリング処理の採用によるRDBへのオーバーヘッドを回避し新たなジョブ実行形態をサポートするため，JMS（Java Messaging Service）基盤を活用した動作形態の追加も検討中である［10］．

3.2 ジョブ実装Javaクラスの多様な実行方式の提供

JBeXにおいてジョブとはAbstractJob抽象クラスを継承したJavaクラス（ジョブクラス）のことを指すが，フレームワークを常駐プロセス上で動作させずにスタンドアロンJavaアプリケーションとして実行する場合や，障害などで動作しない環境でコマンドラインでのジョブ実行を可能とするため，ジョブクラスをあたかもmainメソッドを持つ通常のスタンドアロンJavaアプリケーションであるかのように実行可能なフレームワーク設計を行った．また，JBeXはJava単体テスト用フレームワークとして普及しているJUnitを利用してジョブクラスを直接単体テストすることが可能なJUnit拡張フレームワークも提供する．

このようにジョブクラスに対しては目的と用途にあわせた複数の実行手段を提供することで，開発時のテストや運用時の利用場面を拡大することができる（**表1**）．

表1．ジョブクラスの実行形態

実行方法	利用場面
JUnitによる単体テスト実行	各開発者の環境でジョブクラスを開発し単体テスト実行する場合
スタンドアロンJavaアプリケーション実行	ジョブクラスをテスト実行する場合や，JBeXのサーバープロセスで実行させる必要がないか，またはスタンドアロンJavaアプリケーションとして実行する場合
JBeXサーバープロセス（常駐プロセス環境）上での実行	常駐プロセス上で実行する場合や，J2EEアプリケーションとして実行する場合（通常は本番環境）

3.3 バッチ処理およびオンライン非同期処理に適したAPIとユーティリティの提供

ジョブクラスの動作環境としての機能を提供しただけではJavaフレームワークとは呼べず，業務処理を実装する開発者の観点からはバッチ処理とオンライン非同期処理の実装に必要十分なAPIやユーティリティ群が提供されるかどうかが焦点となる．

このAPIの設計は，Javaフレームワークの学習と開発のしやすさに直結するため，簡単でありながらも要件を充足する機能をどのように提供するかが重要な部分である．この点について，JBeXはテンプレートメソッドパターンに則ったジョブ実行制御用APIを規定するAbstractJob抽象クラスを提供し，開発者が同クラスを継承しJavaプログラムとしてのジョブクラスを開発する（**表2**）．

表2．AbstractJobクラスが提供するテンプレートメソッド

メソッド名	実装すべき内容
doInit()	ジョブ初期化処理
doDestroy()	ジョブ終了処理
doProcessJob()	ジョブの主処理
doCancel()	実行中にキャンセル要求があった場合の処理
doSuspend()	実行中に一時中断要求があった場合の処理
doResume()	一時中断中に再開要求があった場合の処理

同クラスには，長時間処理となる傾向が強いバッチ処理の実装において必要な“チェックポイント情報の保存”や“処理途中でのキャンセル，中断や再開”といった割り込み要求に対して応じるための実行制御用APIが定義されており，これらの割り込み要求に応じるように実装されたジョブクラスとフレームワークとの協調動作により，処理途中でのジョブの中断にも対応が可能である．

また，JBeXはこれらのテンプレートメソッドとあわせて，プログラムの実行時に実行を要求したクライアントプログラム側によって指定されたパラメーター値を取得するAPI，ジョブ結果データの保存やログ出力用のAPI，J2SE環境でもJ2EE環境と同様にデータベースとのコネクションをデータソース（javax.sql.DataSource）経由で取得できるようにして両者の環境の違いを隠蔽する擬似JNDI（Java Naming and Directory Interface）環境構成用のユーティリティなどを合わせて提供し，バッチ処理とオンライン非同期処理プログラムの両方を同一のプログラミングモデルで実装できるようにしている．

処理の開始・実行状態と結果情報の参照を要求するクライアント側としては，通常のJavaクラスを始め，EJBステートレスセッションビーン，Webサービス（SOAP over HTTP）に対応し，複数のチャネルからの呼び出しを可能とした．また，単にJavaプログラムの開発場面だけでなくプロジェクトでの局面に応じた機能を提供することが重要であるとの認識に基づき，オープンソースソフトウェアの負荷テストツールであるApache JMeter［11］用拡張モジュールを標準提供し，オンライン非同期処理に対する負荷テスト作業を容易にしている．

3.4 プロジェクト要件への対応のための拡張性を提供

プロジェクトでの個別要件に対応を図るための拡張性の確保は，汎用的に利用されること自体が目的のひとつであるJavaフレームワークにとって重要な課題である．このため，JBeXの主要なコンポーネントは**図1**で示す構成とし，各コンポーネントの実装はXML形式の設定ファイル内の指定によりそれぞれ変更可能な設計を

採用している. これによってフレームワーク内の振る舞いの多くをプロジェクト要件に合わせて拡張可能としている.

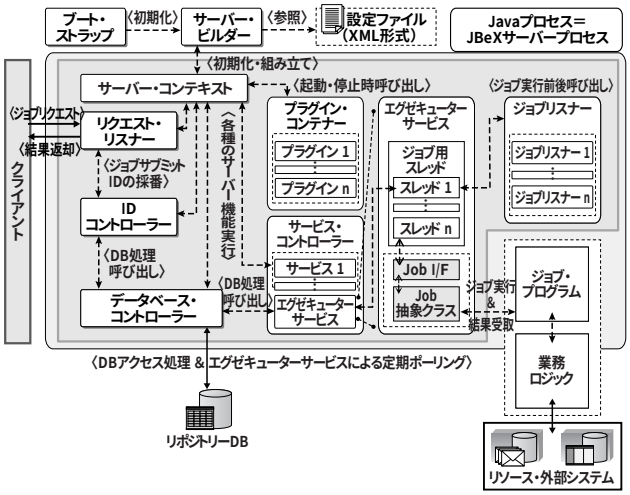

図1. JBeX内部コンポーネント構成

さらにはJ2EEベースのWeb開発フレームワークで培われたアプリケーション設計パターンをジョブクラスの開発場面においても適用できるよう, JBeXはそのサーバープロセスの起動や終了時に任意の機能を追加するための「プラグイン」, 指定したジョブの実行前後で任意の事前事後処理を追加する「ジョブリスナー」などの拡張ポイントを提供し, オンライン処理ともアプリケーション設計の基本的な考え方を共有できるようにした.

また, JBeXはバッチおよびオンライン非同期処理用Javaフレームワークとして開発されたものであるが, 内部設計上は“ J2SEとJ2EEの両環境で動作可能な, 汎用的なJava常駐型アプリケーション開発基盤フレームワーク ”として抽象度をさらに一段高めたレベルで設計し, その基盤上にJavaバッチおよびオンライン非同期処理用フレームワークを実装する構成を採用している. これにより, プロジェクトにおいてバッチ処理やオンライン非同期処理の実装とは無関係にしばしば必要となるJava常駐プログラムの開発場面においても, JBeXというJava基盤フレームワーク上での実装を可能とし, その適用範囲の拡大に繋げる工夫を施している.

3.5 ジョブ管理ミドルウェアとの容易な連携

マシン間やマシン内のジョブネットを制御するジョブ管理ミドルウェアは, 一般的には自らが管理する1つの“ ジョブ ”を1つのスクリプトやネイティブプログラムとして登録し扱う. このため, OS上は1プロセスであるJava仮想マシン内のJavaクラスとして動作する“ ジョブ ”との連携を行うためには, ジョブ管理ミドルウェアからみてジョブとして振舞う連携用スクリプトと, 実際の処理を行うJavaクラスを呼び出し実行しその終了までを待機する機能が必要になる. このジョブ管理ミドルウェアと, バッチ処理プログラムとしてのJavaクラスとの橋渡しをするための仕組みはプロジェクトで個別の作りこみを強いられることが多い機能である. この問題に対して, JBeXはジョブ実行要求から終了までを待機する機能をフレームワークが標準提供した上で, その機能を呼び出すためのスクリプトを含めて標準提供し, プロジェクトにおいて個別開発する必要性を排除した.

4. JBeXの適用事例と評価

4.1 適用事例

本論文の執筆時点において, JBeX適用プロジェクトは数件存在する.

JBeXの利用形態としては, 次の3種類がそれぞれ同程度の割合である.

- バッチ処理の基盤として利用
- オンライン非同期処理の基盤として利用
- オンライン非同期処理を主体としつつスケジュール型のJavaバッチ処理の基盤としても共通的に利用

この内訳はJ2EEオンライン処理システムへの適用を中心に, バッチ処理へもJavaの採用を行うケースへと展開していくという1章で述べた傾向と一致している.

適用プロジェクトでは, 本来予定していた独自のJavaバッチ基盤を構築する作業が, JBeXの適用と, プロジェクト固有要件に基づく部分に対する拡張の実施によって置き換えられ, 計画段階で試算した独自基盤開発のための必要工数から, 適用効果が高い場合は70～80%ほどの削減が達成される結果が見られた. この適用効果を左右する主な要因について, 以下にまとめる.

①［プラス要因－バッチとオンライン非同期両方の処理基盤として利用］

2つの処理形態両方の基盤として使用した場合は, それだけ基盤機能の開発量の削減とアプリケーション設計の共通化が可能となり, 単一の基盤としてのみ使用するよりも高い工数削減効果が得られる.

②［プラス要因－オンラインWebアプリケーションと業務ロジックを共通化］

全ての業務ロジックが対象とはなりえないことが多いが, 業務ロジックをオンラインWebアプリケーションと共通化可能なアプリケーション設計を採用し, バッチとオンライン処理で可能な限り業務ロジックの実装クラスを共通化することで工数削減効果が高まる.

③［マイナス要因－業務ロジック自体が複雑］

フレームワークに直接左右されない業務ロジックが複雑であるほど，相対的にアプリケーション設計と開発工数が上昇し，結果としてフレームワーク適用による工数削減効果は減じる傾向がある．

④［マイナス要因－ソースコードレベルのカスタマイズを実施］

JBeXが提供する拡張性の範囲を大きく超えた要件実現のための基盤機能を実装しなければならない場合，そのカスタマイズ範囲に反比例して工数削減効果は減じる傾向がある．スクラッチ開発を行う場合と比較すれば十分適用による工数削減効果は得られるとはいえ，カスタマイズ部分のテスト工数やバージョンアップ時のメンテナンスのための工数が継続的に発生するため，極力避けることが望ましい．

いずれの場合でも，既に実績のあるフレームワークを利用することで早期にアプリケーション開発上の標準化や運用設計に注力することが可能となり，結果としてプロジェクトの品質向上に寄与したという報告がある．また，フレームワーク適用による標準化の結果としてプログラム開発作業に対しグローバルデリバリーの人的資源の活用を容易にする副次的な効果も見られた．

4.2　代表的システム構成例

適用プロジェクトでのコンポーネント構成をもとにそれらを汎化し，仕様レベルのオペレーショナルモデルとしたものを示す（**図2**）．

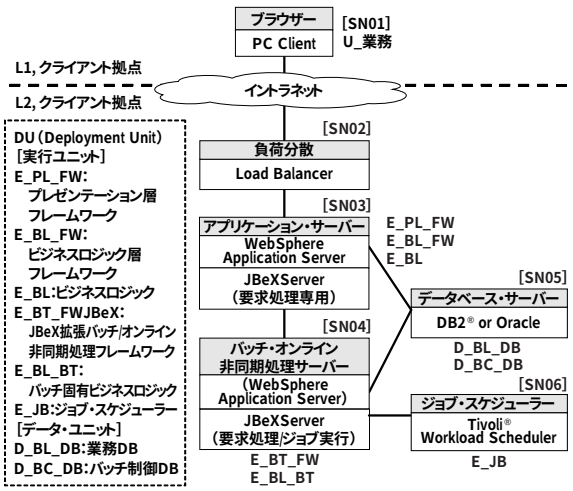

図2.　JBeX適用オペレーショナルモデル例

このように，インフラ構成上は複数ノードによる高可用構成を取りつつJ2EEアプリケーションサーバーの負荷の一部をバッチ処理専用サーバー（またはDBサーバー）へオフロードし，アプリケーション設計においてはJBeXを拡張し業務プログラムに対しては隠蔽した上で，業務ロジックをオンラインWebアプリケーションと共用可能とする構成が主流である．

5．おわりに

Javaによるバッチ処理とオンライン非同期処理の開発基盤となるJavaフレームワークJBeXを開発した．JBeXは，従来型のバッチ処理だけではなくオンライン非同期処理にも適用可能で，Javaプラットフォームの選択によらずアプリケーション開発に共通基盤を提供できるという特徴がある．JBeXを実プロジェクトに適用して，適用効果が高い場合には70～80%の工数削減と標準化による品質向上の効果を確認した．

同分野における開発フレームワークの整備についてはその重要性が認識されつつある［12］．これを受けてJavaバッチ処理分野においてはJBeXを含めSpring Batch［13］などのオープンソースソフトウェア他でその萌芽が見られる状況ではあるが，定型的な開発，コード自動生成などの生産性向上が見込める部分への発展・強化と，フレームワークの適用がアプリケーションのパフォーマンス向上に対しどのように寄与するかが今後の課題であろう．今後の展望として，生産性向上へのフォーカスに加え，JBeX適用プロジェクトの情報を集約しジョブ処理のスループットを含めたパフォーマンス指標の公開，再利用性の高いジョブクラスの標準提供，JBeXを基盤として大量レコード処理のパフォーマンスを効率化するサブフレームワークなどを検討しており，この分野においてシステム開発に対して即効性のある方法で生産性向上や開発の迅速化への貢献を継続して行っていきたい．

謝辞

JBeX開発メンバーとして貢献を頂いた日本アイ・ビー・エム株式会社 オープンシステム開発．吉田 健氏および大楠 貴浩氏，貴重なご意見をいただいたクライアントIT推進 流通．第3システムサービス部．谷口 裕亮氏に深謝します．

参考文献

[1] 日経BP社:“ Close Up 開発力を磨く ”日経コンピュータ(2005年1月10日号), pp.88-92 (2005).
[2] 日経BP社:“ 解体!レガシー・バッチ ” 日経コンピュータ(2006年8月21日号), pp.32-47 (2006).
[3] 小宮 聖則:“ Webシステムにおける暗黙的非同期処理アーキテクチャー, ” PROVISION, No.54, pp.88-96 (2007).
[4] 山本 宏:“ WebSphere XDのご紹介 ” http://www.ibm.com/jp/software/websphere/developer/was/xd/v6/1_3.html, (2007.8.10).
[5] IBM WebSphere Developer Domain“ バッチ処理の変革を支えるソリューション ”: http://www-06.ibm.com/jp/software/websphere/developer/bi/batch/index.html, (2007.3.14).
[6] H.M.Hess:“ Aligning technology and business: Applying patterns for legacy transformation, ” *IBM Systems Journal*,Vol.44, No.1,pp.25-45 (2005).
[7] JSR-237:Work Manager for Application Servers http://jcp.org/en/jsr/detail?id=237, (2003.12.15).
[8] CommonJ WorkManager API http://edocs.bea.com/wls/docs92/commonj/commonj.html#wp1058124, (2007).
[9] JSR-316: Java Platform, Enterprise Edition 6 (Java EE 6) Specification http://jcp.org/en/jsr/detail?id=316, (2007).
[10] IBM developerworks:Java theory and practice: Should you use JMS in your next enterprise application? http://www.ibm.com/developerworks/java/library/j-jtp0205.html, (2002.2.01).
[11] Apache JMeter: http://jakarta.apache.org/jmeter/ (2007).
[12] 木村 綾太郎:“ Javaバッチ実用化に向けたフレームワークの開発, ”NRI技術創発, Vol9, pp.42-69 (2007).
[13] Spring Batch: http://www.springframework.org/spring-batch/ (2007.8.12).
[14] Lara D 'Abreo:“ High-volume Transaction Processing in J2EE, ” http://www.devx.com/Java?article /20791, (2004.4.15).

日本アイ・ビー・エム株式会社
GTS アセット&SOAコンピテンシー
主任ITスペシャリスト
松原 武司 Takeshi Matsubara

[プロフィール]
1999年日本IBM入社.UNIX系システムの導入・運用担当を経てWeb系システム開発プロジェクトに携わる.以後,複数の業種・業界のお客様に対するアプリケーション開発場面における各種標準化の策定や技術支援,システム基盤用フレームワークの設計と開発に従事.
mtakeshi@jp.ibm.com

日本アイ・ビー・エム株式会社
GTS アセット&SOAコンピテンシー
ITスペシャリスト
原 亮吉 Ryokichi Hara

[プロフィール]
2004年日本IBM入社.SOA分野のシステム開発を支援するIBMサービス・コンポーネントの設計/開発を担当.現在は,JBeXの技術サポートとプロジェクトへの展開に従事している.
harakich@jp.ibm.com

日本アイ・ビー・エム株式会社
GTS アセット&SOAコンピテンシー
ITスペシャリスト
髙石 稔 Minoru Takaishi

[プロフィール]
2005年日本IBM入社.入社後,プロジェクトにおいてJava開発支援ツールの開発や移行支援を担当.同時にIBMサービス・コンポーネントの設計/開発を担当し,現在はJBeXの機能拡張や適用プロジェクトへの技術支援に従事している.
tminoru@jp.ibm.com