

バッチ

日本アイ・ビー・エム株式会社
WebSphere クライアント・テクニカル・プロフェッショナルズ
伊藤 隆

当セッションでは、WASv8.5で大きく強化されたバッチの機能について、お話いたします。

Agenda

1. WebSphereバッチ概要
 2. プログラミング・モデルと開発環境
 3. ジョブの実行と管理
 4. まとめ
- 参考文献

概要に続いて、内容は大きく2つのパートでお話します。

前半は、開発関連のトピック、後半が、実行環境に関連するトピックになります。

1. WebSphere バッチ概要

WebSphereバッチがもたらす価値
WASv8.5 バッチ概要
WebSphereバッチの基本概念

それでは、まず、WAS V8.5で統合&強化されたバッチ機能の概要についてお話します。

WebSphere バッチがもたらす価値

Java EE基盤による柔軟・高信頼・低コストのバッチ処理環境

- 実績あるJava EEアプリケーション・サーバーの利用
 - XML処理やサービス連携等の最新機能の活用
 - OLTPで培った高い信頼性
 - 開発要員・スキルの共通化
- Java EE OLTPとバッチでのロジックの共通化
 - 重複開発の排除 & 開発効率の向上
 - 段階的なオンライン化 & バッチのリニューアル
- オンライン ⇄ バッチ間でのサーバー資源共有
 - サーバー資源の集約・利用率向上によるコスト削減
 - 管理・運用の統合 & 効率向上
 - より多数のサーバー資源を活用した処理の高速化
 - OLTPとの連携で従来型のバッチ・ウィンドウを解消した24x365の実現

WASv8.5のバッチ機能は、Java EE基盤による高度で柔軟なバッチ環境を提供します。では、このようなJava EEによるバッチ基盤は、お客様に、どのような価値をもたらすでしょうか？ここでは、大きく3つの観点から、期待される効果を挙げたいと思います。

1つ目は、現在のJava EEサーバー製品が、最新機能を備え、かつ、高い信頼性を持つソフトウェアであり、広く使用されていることで非常に大きなスキル・ベースを持つ点です。いまや、Java EE製品は、企業の基幹系業務でも利用されるほど、信頼性が高く、かつ、企業のあらゆるシーンで利用されており、これをオンラインだけでなく、バッチでも利用しようというのは、当然の流れとも言えます。

2つ目は、OLTPとのコードの共通化がもたらす利点です。重複開発を排除して効率を向上するといった側面に加えて、バッチをオンライン化していく上で、コードを共有しながら段階的な移行が可能になります。

3つ目は、サーバー資源が共有できるという点です。バッチ用か、オンライン用かを問わず、リソースを共有して柔軟に利用することができることで、コストや効率が向上できます。また、オンラインの空き資源を活用することで、バッチ処理を高速化し、バッチ・ウィンドウを短縮できるといった効果も期待できます。更に、OLTPとより密に連携することで、相当量のデータを溜め込んでバッチ専用システムで処理するのではなく、随時少量のデータを短時間で処理して、いわゆるバッチウィンドウの解消を図ることも可能になります。

このように、様々な利点があり、実際に多くのお客様で、実際にJava EE基盤をバッチ処理を活用しようと検討、或いは、実際に活用しています。

WAS V8.5 バッチ概要

WebSphere XD Compute Gridを統合したエンタープライズ・バッチ基盤 基本的な機能

- Javaバッチ・プログラミング・モデル
 - 障害時の早期リカバリーを可能にするトランザクショナル・バッチ
 - 大規模バッチの処理時間を短縮する分割並列処理 **New**
- WAS上で稼働する信頼性の高いバッチ・コンテナ
 - コンテナによるチェック・ポイント&リスタート機能 **New**
 - C/C++プログラムや外部コマンド等ネイティブ・プログラムの実行機能
 - 既存資産を活用するCOBOLサポート(zOS版) **New**
- ジョブ・スケジューラによる一元的&高度なバッチ実行制御機能
 - XMLベースのxJCLによる柔軟なバッチ実行制御 **New**
 - 優先度やジョブ・クラスによるジョブ制御 **New**
 - ジョブ・ネットへの組込みを可能にする外部スケジューラとの統合 **New**

前ページで挙げたような価値に着目して、約7年前に、IBMは、WASのアドオン製品としバッチ製品をリリースしました。

その後、コアのバッチ機能を、V7のFeature Packに、次いで、WASv8に搭載してきましたが、遂に、今回リリースされたWASv8.5において、全バッチ機能が統合されました。このように、7年間もの間、進歩を重ねてきたバッチ機能は、先に上げた価値&効果を更に高める諸機能を提供します。

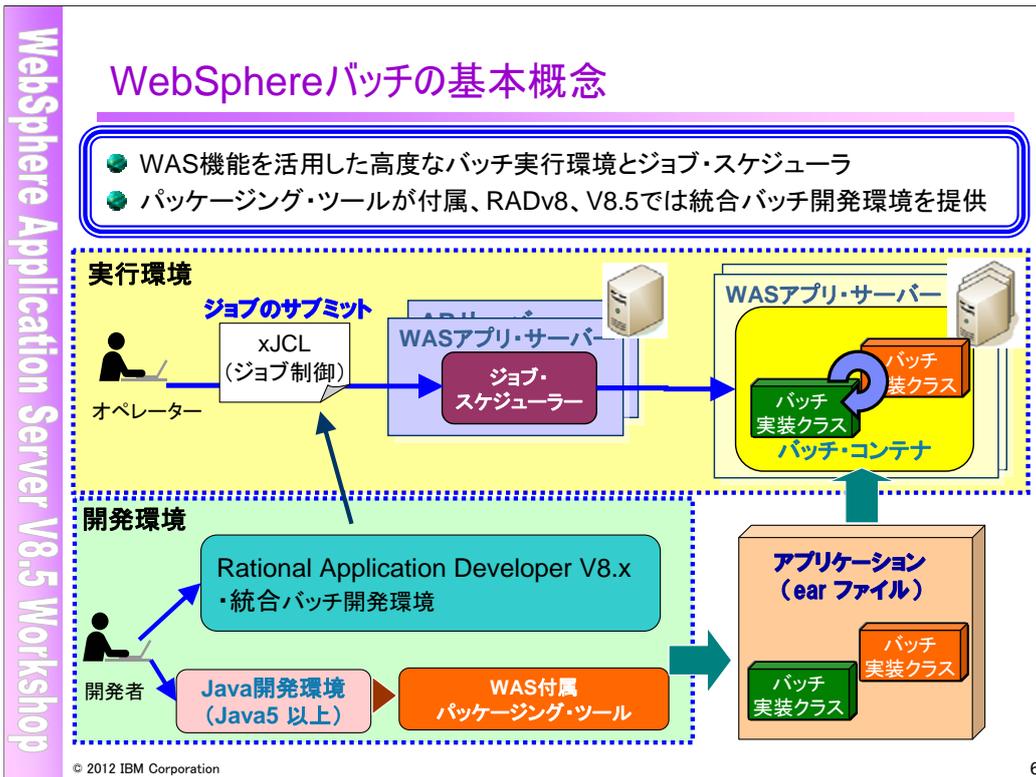
ここでは、その主な機能を簡単に紹介します。詳細は、2章、3章でご説明していきます。

まず、プログラミング・モデルに基づくJavaバッチの実現です。プログラミング・モデルを通じたランタイムとの連携により、チェックポイント・リスタートや、並列処理といった機能を実現します。

2番目は、ランタイム環境であるバッチ・コンテナの機能です。信頼性に加えて、Java以外の言語で開発されたコマンドやユーティリティといったネイティブ実行機能や、zOS版に限りませんが、COBOLの実行環境も提供します。

3番目は、ジョブ・スケジューラの機能です。ジョブが一元的に管理できるほか、優先度制御やジョブクラスといった高度な機能が利用できます。

また、企業で既に利用されている実績のあるTivoli Workload Schedulerのようなエンタープライズ・スケジューラから、ジョブを実行し連携するユーティリティ機能を提供します。これによって、企業内のジョブ・ネットに、WASによるバッチ基盤を組み込むことができます。



詳細の説明に入る前に、WASv8.5のバッチ機能を実現するためのコンポーネントや開発ツール等を俯瞰してみましょう。

まず、左下にあるのは開発環境です。

RADは、v8.0.4以降で、WebSphereバッチの開発機能を提供しています。これを利用すれば、ウィジェット形式で、バッチ開発を行い、パッケージング&テストまで行うことができます。

また、WASv8.5には、パッケージング・ツールが付属しますので、使い慣れたJava開発環境で開発を行い、それをパッケージング・ツールでパッケージングすることも可能です。

開発したアプリケーションは、拡張されたearファイルとしてパッケージングします。

また、開発者は、ジョブの実行者に対して、xJCLを作成するための雛形を提供する必要があります。

RADでは、このxJCLを作成するxJCLエディタが付属します。

実行に際しては、まず、アプリケーションのパッケージであるearファイルをWASにデプロイする必要があります。バッチ・アプリケーションを含むearファイルを、WASにデプロイすると、アプリケーション・サーバーはバッチ・コンテナを用意し、サーバー再起動後、利用が可能になります。

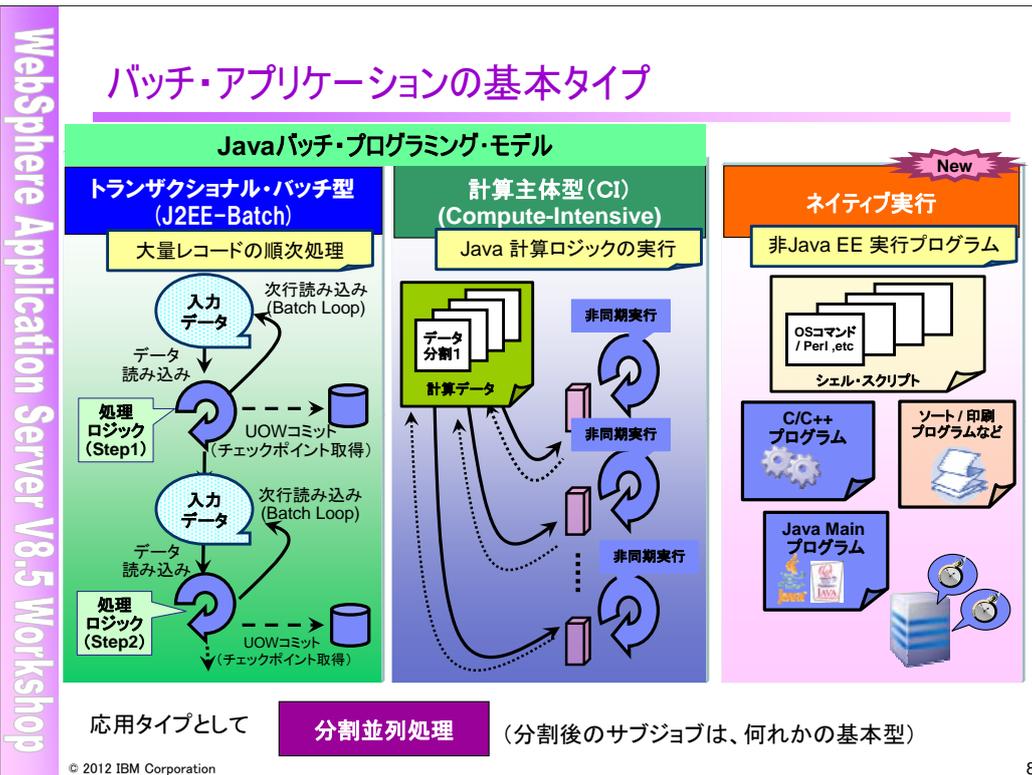
バッチ・コンテナ上で、アプリケーションが稼働できる状態になったら、オペレーターは、バッチの実行内容をxJCLに記述して、ジョブ・スケジューラにサブミットします。ジョブ・スケジューラは、アプリケーション・サーバーの中から適切はサーバーを選択して、ジョブをディスパッチします。

ディスパッチされたサーバーは、xJCLに基づいてバッチを実行します。

これが、開発を含めた、WASv8.5のバッチに関連する主要概念になります。

2. プログラミング・モデルと開発環境

バッチ・アプリケーションの基本タイプ
トランザクショナル・バッチ型
ネイティブ実行
ジョブの分割&並列処理



WASv8.5がサポートするバッチ・アプリケーションには、大きく分けて3つの基本形があります。

また、それらとは別に、分割並列処理がありますが、分割されて実行されるサブジョブは、これらJavaバッチ・プログラミングの基本形に属します。

まず、トランザクショナル・バッチ型は、データを1レコードずつ入力して処理を行うタイプで、処理中に、一定のタイミングで、チェックポイントをDBに記録します。開発者は、基本的に1レコードの処理を記述し、繰り返しのループ制御などは記述しません。レコード処理の繰り返しはコンテナが行います。また、ここにあるように、ジョブには複数のステップが記述できるので、前のステップでの処理結果の出力データを、次の処理のインプットにして処理することが可能です。WASv8.5で、Javaバッチといった場合、狭義には、このトランザクショナル・バッチ型を指す場合があります。

次の計算主体型は、逆に、コンテナからは、run()というメソッドを呼び出すだけで、あとの制御は全てユーザー・プログラムに委ねられます。データの存在もコンテナからは見えません。したがって、データ主体ではなく、計算主体向けということで、この名称になっています。

3つ目のタイプは、ネイティブ実行型、つまり、Javaとも限らず、C/C++で作成された実行プログラムであったり、OSや他のSWに付属のコマンド・ツールや、或いは、単独で実行するJava Mainタイプのプログラムなどを実行するものです。

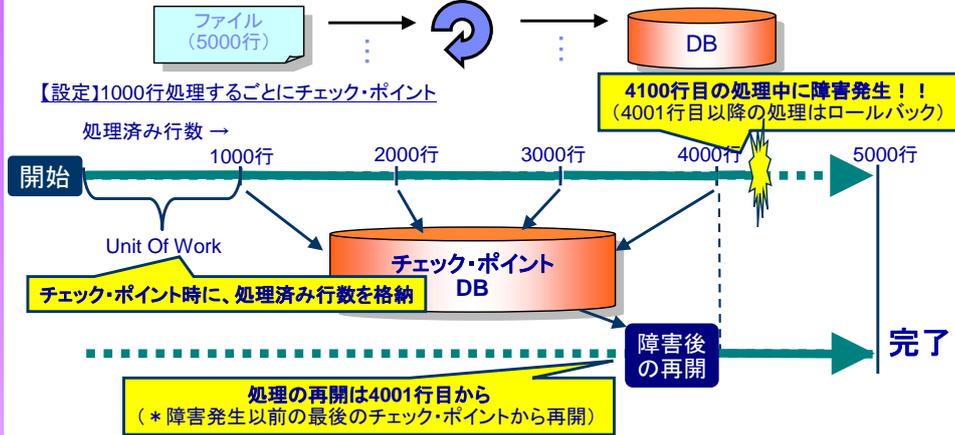
これによって、既存の資産と連携することで、開発のワークロードを圧縮しつつ、より高度なバッチ処理を実現できます。

以降、しばらくは、トランザクショナル・バッチについて詳細を見ていきます。

トランザクショナル・バッチ型:チェック・ポイント&リスタート

- 一定のタイミングでトランザクションをコミット(チェックポイント)
- チェックポイントからリスタートすることで、障害から復旧時間を短縮

ファイルから1レコードずつ読み込みDBを更新する例



- チェック・ポイント保存のタイミングは、処理レコード数または時間で指定
- チェック・ポイント保存のタイミング決定のロジックをユーザーが実装することも可能

© 2012 IBM Corporation

9

トランザクショナル・バッチについて、そのチェック・ポイント&リスタート機能について補足します。

コンテナは、バッチ処理において一定のタイミングで、トランザクションをコミットして、DBにチェックポイントを書き込みます。したがって、コミット時に他のDBへの更新が発生する処理では、グローバル・トランザクションになります。ただし、チェックポイントと同じDBへの更新であれば、ローカル・トランザクションを選択することもできます。

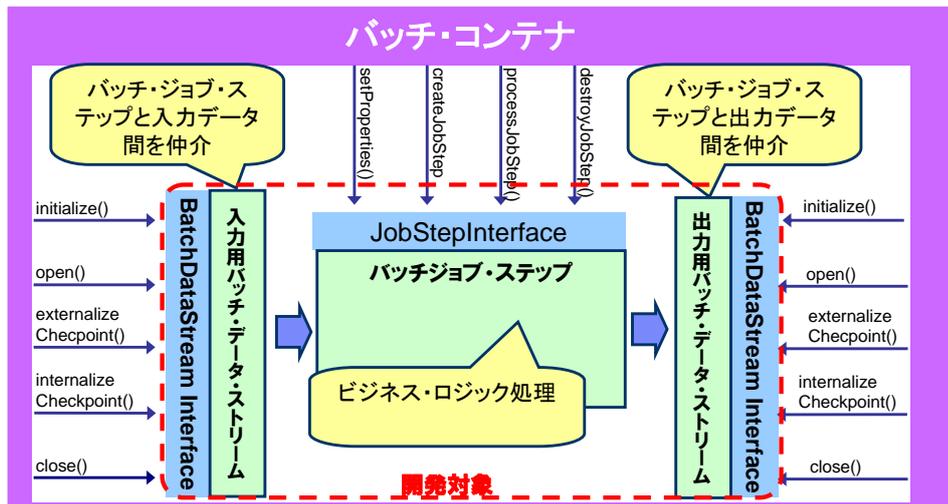
障害が発生すると、チェックポイントから、処理を再開します。これによって、それまでの処理を無駄にせず、処理再開から完了までの時間を大幅に短縮できます。

例えば、この図のように、1000行ごとにチェックポイントをとる設定とした場合、4100行目で障害が起これば、処理は4001行目から再開され、4000行までの処理は省くことができます。

チェックポイントのタイミングが重要になってきますが、これは、xJCL中に、レコード数、或いは、時間のどちらかで指定することができます。

トランザクショナル・バッチ型：プログラミング・モデル

- バッチデータ・ストリーム ……データの入出力制御やチェックポイント処理
- バッチジョブ・ステップ ……1レコードに対するビジネスロジック処理



© 2012 IBM Corporation

10

次に、トランザクショナル・バッチを、どう開発・実装するかを、もう少し詳しく見てみましょう。

トランザクショナル・バッチ型のプログラミング・モデルは、大きく2つ、処理内容を記述したバッチ・ジョブ・ステップと、データの入出力を担うバッチ・データ・ストリームの2つのJavaコンポーネントからなります。

この図は、1レコード入力に対して処理を行い、1レコード出力するという典型的なケースでの、開発対象コンポーネントを表しています。この場合、1つのバッチ・ジョブ・ステップに対して、入力用、出力用のバッチ・データ・ストリーム、合計3つのコンポーネントを開発することになります。バッチ・データ・ストリームや、バッチ・ジョブ・ステップは、ここにあるような、バッチ・コンテナとのインターフェースとなる各メソッドを実装する必要があります。

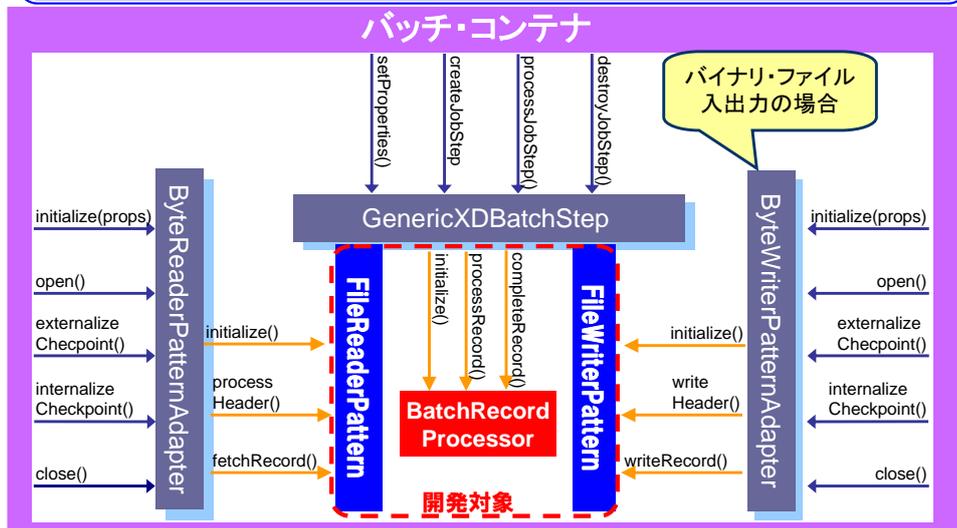
たとえば、JobStepInterfaceのprocessJobStep()は、まさに、レコード処理のためのメソッドで、終了のフラグが返るまで、繰り返し呼ばれます。

また、一定のタイミングで、バッチデータストリームに対して、externalizeCheckPoint()が呼ばれるので、チェックポイント情報、例えば、処理行数等、をコンテナに渡す必要があります。

インターフェースさえ実装すれば、ジョブ・ステップと、バッチ・データ・ストリーム間にやり取りなどの開発は、かなり自由度が大きいモデルとなっています。ただし、バッチコンテナとのインターフェースの実装が、かなり大変です。

トランザクショナル・バッチ型 : BDSフレームワーク

- データ入出力(ファイル入出力、DBアクセス等)の典型的な制御パターンをフレームワークで吸収、開発者は業務ロジック実装に集中



© 2012 IBM Corporation

11

BDSフレームワークは、幾つかの定型パターンについて、実装を提供し、開発の手間を大幅に削減します。

たとえば、入出力がそれぞれひとつのバイナリファイルであり、それぞれから、1レコード入力して1レコードを出力する処理であれば、次のようなパターンが適用できます。

それぞれ、インターフェース実装は、バイナリファイルの入力については、ByteReaderPatternAdapterを、出力には、ByteWritePatternAdapter、そして、バッチ・ステップについては、GenericXDBatchStepが提供されます。そして、開発は、それぞれから呼ばれるFileReaderPattern、FileWritePattern、BatchRecordProcessorを実装することになります。単純に見ても、14個のメソッドが、9個のメソッドで済むというだけでなく、例えば、このファイルのリード&ライトのケースでは、ファイルのオープンや読み込み、クローズといった定型的な処理は、パターン実装に含まれ、開発者が実装する必要がありません。

このように、BDSフレームワークによって、開発のワークロードを大幅に軽減することが可能です。

【参考】BDSフレームワーク: 提供されるパターンとクラス

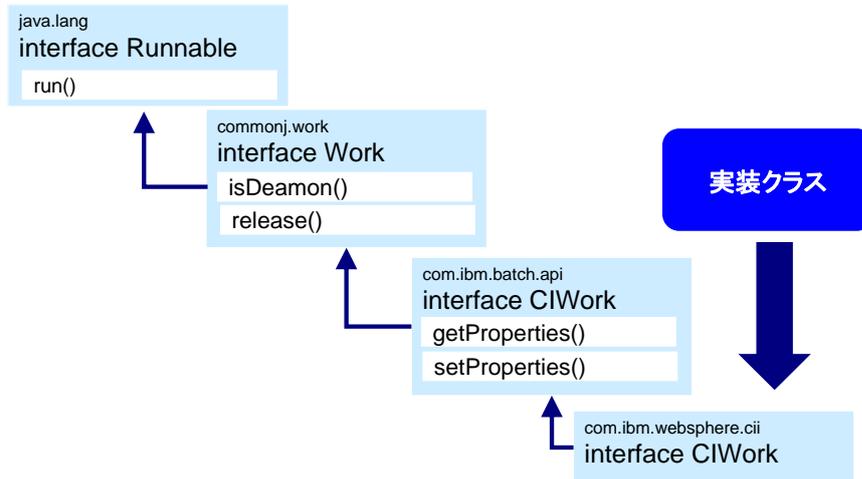
パターン名	説明	サポート・クラス
JDBCReaderPattern	JDBC 接続を使用してデータベースからデータを取得する場合に使用されます。	LocalJDBCReader JDBCReader CursorHoldableJDBCReader
JDBCWriterPattern	JDBC 接続を使用してデータベースにデータを書き込む場合に使用されます。	LocalJDBCWriter JDBCWriter
ByteReaderPattern	ファイルからバイト・データを読み取る場合に使用されます。	FileByteReader
ByteWriterPattern	ファイルにバイト・データを書き込む場合に使用されます。	FileByteWriter
FileReaderPattern	テキスト・ファイルの読み取りに使用されます。	TextFileReader
FileWriterPattern	テキスト・ファイルへの書き込みに使用されます。	TextFileWriter
RecordOrientedDataSet-ReaderPattern	z/OS データ・セットの読み取りに使用されます。	ZFileStreamOrientedTextReader ZFileStreamOrientedByteReader ZFileRecordOrientedDataReader
RecordOrientedDataset-WriterPattern	z/OS データ・セットへの書き込みに使用されます。	ZFileStreamOrientedTextWriter ZFileStreamOrientedByteWriter ZFileRecordOrientedDataReader
JPAReaderPattern	OpenJPA を使用してデータベースからデータを取得する場合に使用されます。	JPAReader
JPAWriterPattern	Java Persistence API (JPA) 接続を使用してデータベースにデータを書き込む場合に使用されます。	JPAWriter

* InfoCenterより転載

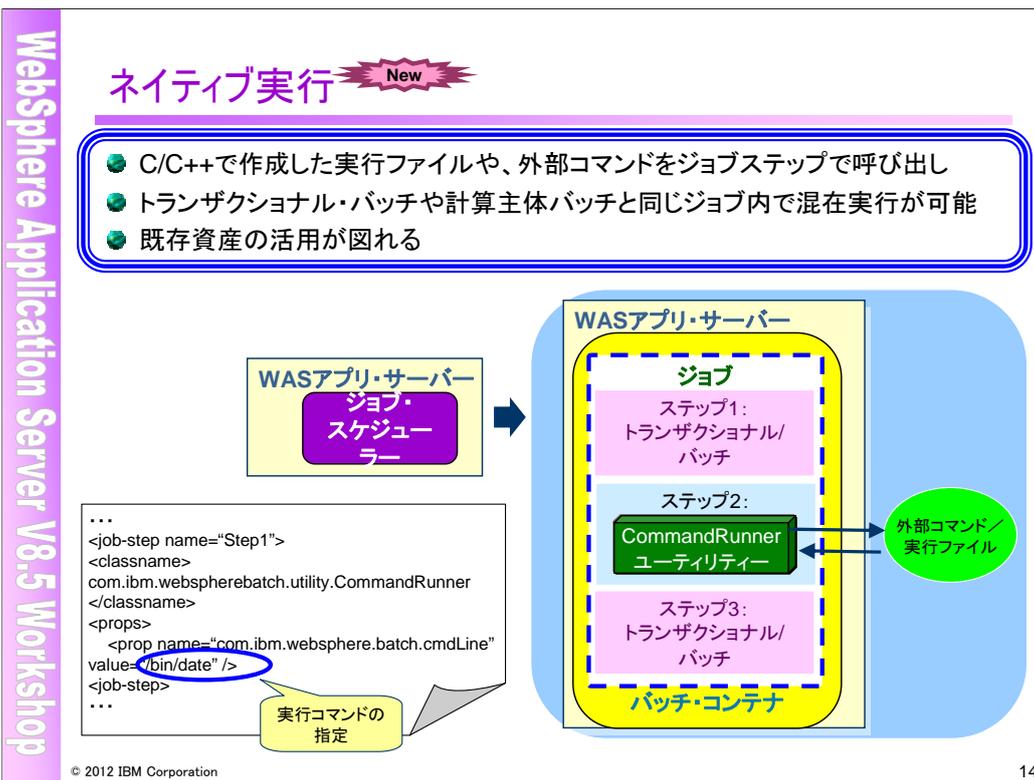
これらは、InfoCenterの抜粋で、BDSフレームワークが提供している入出力のパターンになります。ファイル、DBのほか、z/OSのデータセット用のパターンなども用意されています。

【参考】計算主体型プログラミング・モデル

- 計算主体バッチの実装クラスは、インターフェースCIWorkを実装
- 開始時に、メソッドrun() が呼び出され、メソッド終了とともにジョブ・ステップが完了



これも、参考までに、計算主体型の場合に実装するインターフェースです。3階層になっていますが、Java EE非同期の仕組みcommonj.workを拡張したシンプルなプログラミング・モデルで、開発者は、実質的にrun()メソッドのみを実装すれば良いことになります。



残る、もうひとつの基本型が、ネイティブ実行型です。

ジョブ中のジョブ・ステップの一環として、ネイティブのコマンドを実行するCommandRunnerユーティリティを使用できます。つまり、1ジョブの中で、トランザクショナル・バッチや、計算主体型バッチと連携する形で、ネイティブの実行を行うことができます。

例えば、最初のジョブ・ステップが、トランザクショナル・バッチで、DBのテーブルの各行を処理してファイルを出力し、

次に、CommandRunnerを使って、LinuxのSORTコマンドを呼び出して、ソートを行って、後続のトランザクショナル・バッチに渡すといった処理を1つのジョブとして行うことができます。

ちなみに、ネイティブ・ジョブの実行は簡単です。基本的に実行ファイルをxJCLに記述するだけです。オプションで、引数を与えたり環境変数を設定することもできます。

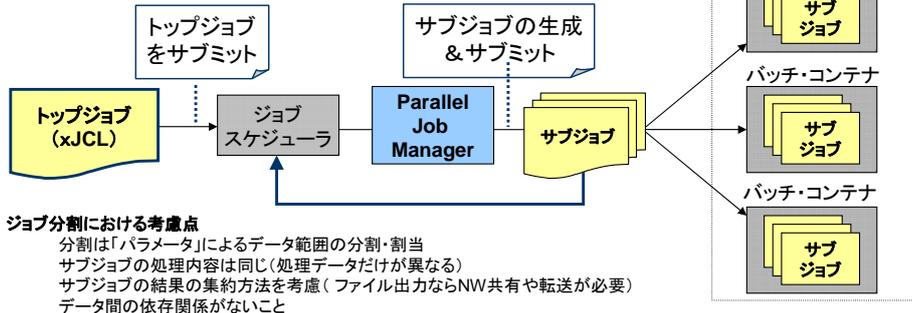


ジョブの分割並列処理(パラレル・ジョブ・マネージャー)

- トップ・ジョブから、多数のサブジョブを生成し実行依頼
- 大規模ジョブを分割&並列処理し、リソースを有効活用し全体処理時間を短縮
- 障害時は、サブ・ジョブ単位で再実行が可能

■ パラレル・ジョブ・マネージャーの機能

- ◆ サブジョブ用xJCLを生成しスケジューラにサブミット
- ◆ サブジョブの初期化・完了処理、状況のモニター
- ◆ 戻り値の集約及び、ログの集約
- ◆ 障害発生時の各サブジョブのリスタート処理



© 2012 IBM Corporation

15

開発編の最後に、パラレル・ジョブ・マネージャーによるジョブの分割並列処理について簡単にご紹介します。

大規模なバッチ処理を行う場合、分割・並列処理による高速化が、当然必要となってきます。これを自動化する機能が、Parallel Job Manager (PJM)です。PJMは、トップジョブがサブミットされると、xJCLに記載されたパラメータを分割するためのクラスを呼び出し、指定された分割数のサブジョブ用xJCLを生成して、サブミットします。さらに、PJMは、これらのジョブ実行をモニターし、完了すると、戻り値、及び、ログを集約して、ひとつのジョブとして完了します。サブ・ジョブが失敗した場合でも、トップジョブを再始動することで、該当するサブジョブだけを再サブミットします。

このように、PJMは、一元的にジョブの分割・実行・モニターの機能を提供します。

【参考】プログラミング・モデルの拡張

New

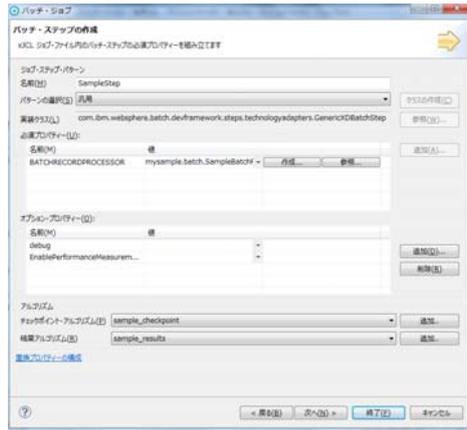
- OSGiバッチ・アプリケーション
 - ◆ バッチをOSGiアプリケーションとしてパッケージ化してデプロイ可能
- ジョブ・リスナー
 - ◆ ジョブ及びステップに関して初期設定とクリーンアップを追加する
com.ibm.websphere.batch.listener.JobListener インターフェースの実装を提供
- ジョブ・ステップ・コンテキスト
 - ◆ 現行バッチ・ジョブ・ステップが実行されるコンテキストを一意的に識別する情報(ジョブID等)へのアクセス
 - ◆ アプリケーション固有情報を、バッチ・プログラミング・フレームワーク・メソッド間で受け渡せるユーザー・データ域
 - ◆ ステップをまたがってアプリケーション固有情報を渡すことができる一時的なユーザー・データ域
 - ◆ チェックポイント/再始動をまたがって、アプリケーション固有情報を保管する永続的なユーザー・データ域
- トランザクショナル・バッチの拡張
 - ◆ レベル・スキップ処理:レコードの読み取り、及び、書き込みエラーをスキップ
 - ◆ ステップ再試行処理:processJobStepメソッドにエラーが発生した場合に、ジョブ・ステップを再試行
 - ◆ トランザクション・モード:グローバル・トランザクション・モードかローカル・トランザクション・モードかを指定
 - ◆ バッチ・データ・ストリームのタイムアウト:トランザクション・タイムアウト(秒)を指定可能

参考までに、ここまでに触れなかったプログラミング・モデルや開発関連の強化項目をリストしています。時間の都合で、説明は割愛させていただきます。

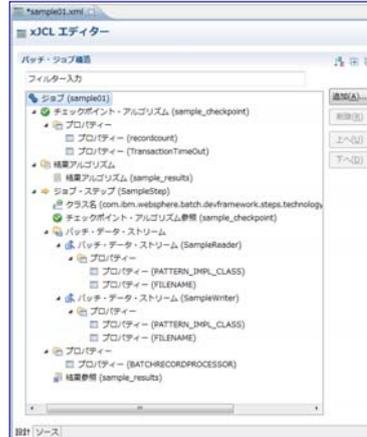
【参考】Rational Application Developer Javaバッチ開発

- ジョブ構成と必要なコンポーネントの作成を支援するジョブ作成ウィザード
- グラフィカルにxJCLを編集できるxJCLEディター
- 作成したバッチを、その場で稼働確認可能

ジョブ作成ウィザード: バッチ・ステップの作成画面



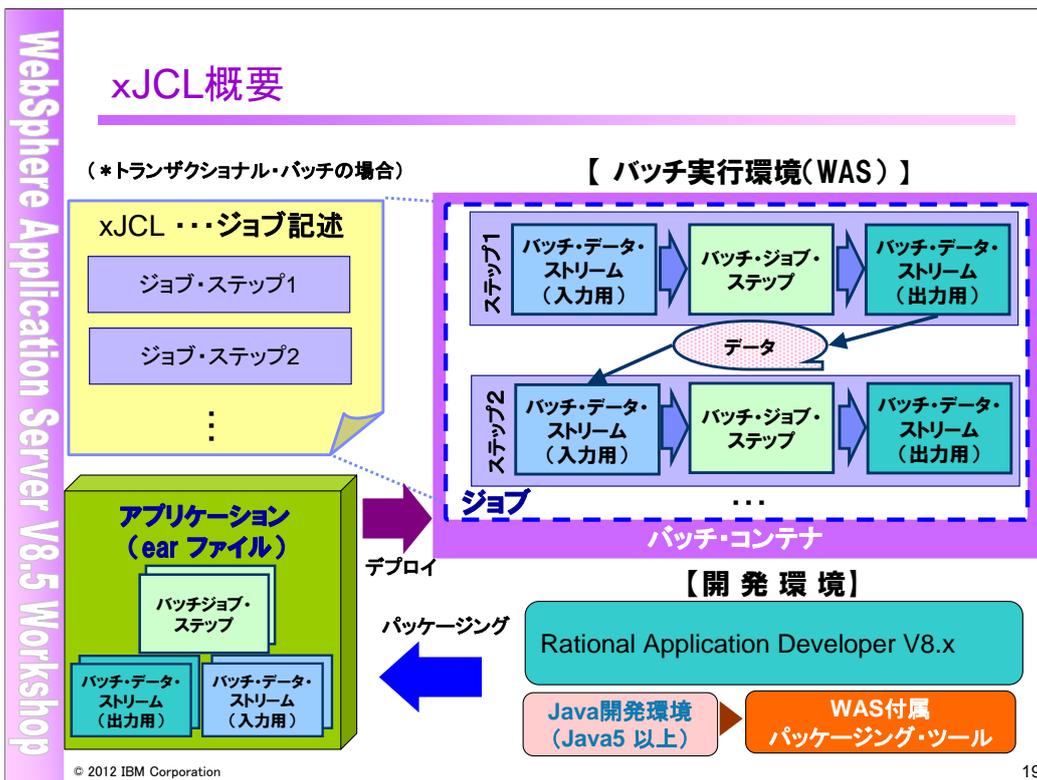
xJCLEディター



開発ツールの説明も割愛します。参考までに、Rational Application Developer for WebSphereの画面イメージを添付します。

3. ジョブの実行と管理

- xJCL概要
- バッチ実行環境の概要
- ジョブ管理コンソール
- 日時指定によるジョブ実行
- ジョブ・クラス
- ジョブ優先度制御
- 外部スケジューラ連携
- ロギング機能



バッチ基盤のランタイムの説明の前に、バッチ実行の制御のための言語であるxJCLについてご説明します。

開発したバッチ実行のためのコンポーネント、Javaクラスは、Rational Application Developer、或いは、WAS付属のツールで、earファイルとしてパッケージングし、WASで稼動するバッチコンテナにデプロイします。

デプロイした時点では、どのようなバッチ処理を、行うかはわかりません。処理の内容はxJCLで記述し、バッチ・コンテナに指示します。

xJCL では、どのコンポーネントをどう組み合わせ、どういった順番で実行するかを記述します。トランザクショナル・バッチであれば、幾つかのバッチ・データストリームと、バッチ・ジョブ・ステップを組み合わせ、一つのジョブ・ステップを構成します。このようなジョブ・ステップが1つ以上集まって、一つのジョブとなります。

例えば、最初のジョブ・ステップでは、DBからデータを取り出して処理を行い結果をファイルに出力し、次のジョブ・ステップでは、このファイルを入力として、1行ずつ処理を行い、結果を、更に別のファイルに出力します。更に3番目のジョブ・ステップが、このファイルを読み込んで処理を行い。。。といったようにバッチ処理が進行していきます。

全てのジョブ・ステップが完了した時点でジョブ全体は完了し、結果がスケジューラに通知されます。

xJCL概要:構成

- 柔軟なモジュールの組み合わせ & 再利用が可能
- パラメータの受け渡し



© 2012 IBM Corporation

20

xJCLにどのような情報を記述するのか、トランザクショナル・バッチの場合について紹介します。
 先にお話したように、ジョブの内容は、ジョブ・ステップに分解できます。
 各ジョブ・ステップでは、バッチ・ジョブ・ステップとバッチ・データ・ストリームのクラス名や、チェック・ポイント・アルゴリズムの指定、或いは、これらに付随した各種パラメータを記述します。

xJCL概要: 代替プロパティ

- 代替プロパティ値は実行時に変更可能 (xJCL中で指定)
- 実行毎にパラメータ値を変更する場合に便利 (xJCLの使い回しが可能)

xJCL

```

<substitution-props>
  ...
  <prop name="sample.input.file" value="/tmp/indata.txt" />
  <prop name="sample.output.file" value="/tmp/outata.txt" />
</substitution-props>

  ...
  デフォルト値
  <prop name="IMPLCLASS"
    value="mysample.HelloBatchFileReader" />
  <prop name="FILENAME" value="{sample.input.file}" />
</props>
  ...
        
```

ジョブ管理コンソール: サブミット画面

代替プロパティの更新にチェックして、「サブミット」をクリックすると、代替プロパティの設定画面が表示される

© 2012 IBM Corporation

21

パラメータの変更があるたびに、xJCLを作成するとなると、似たようなxJCLを大量に管理しなくてはなりません。

或いは、例えば、ジョブに渡すファイル名に日付が入る場合など、そのたびにxJCLを変更することになりかねません。

そこで、xJCLをなるべく共用し、実行時に必要なパラメータだけを変更するための機能が、「代替プロパティ」です。

この図のように、\$で始まる変数を記入することで、代替プロパティとして宣言したことになります。

その上で、各代替プロパティのデフォルト値を、<substitution-props>セクションに記載します。

代替プロパティは、実行時に値を指定することができます

例えば、ジョブ管理コンソールから、ジョブをサブミットする際に、「代替プロパティの変更」にチェックを入れると、

次の画面で、代替プロパティの値を変更することができます。

実は、この代替プロパティの機能は、先に紹介したParallel Job managerによるパラメータの分割でも利用されています。

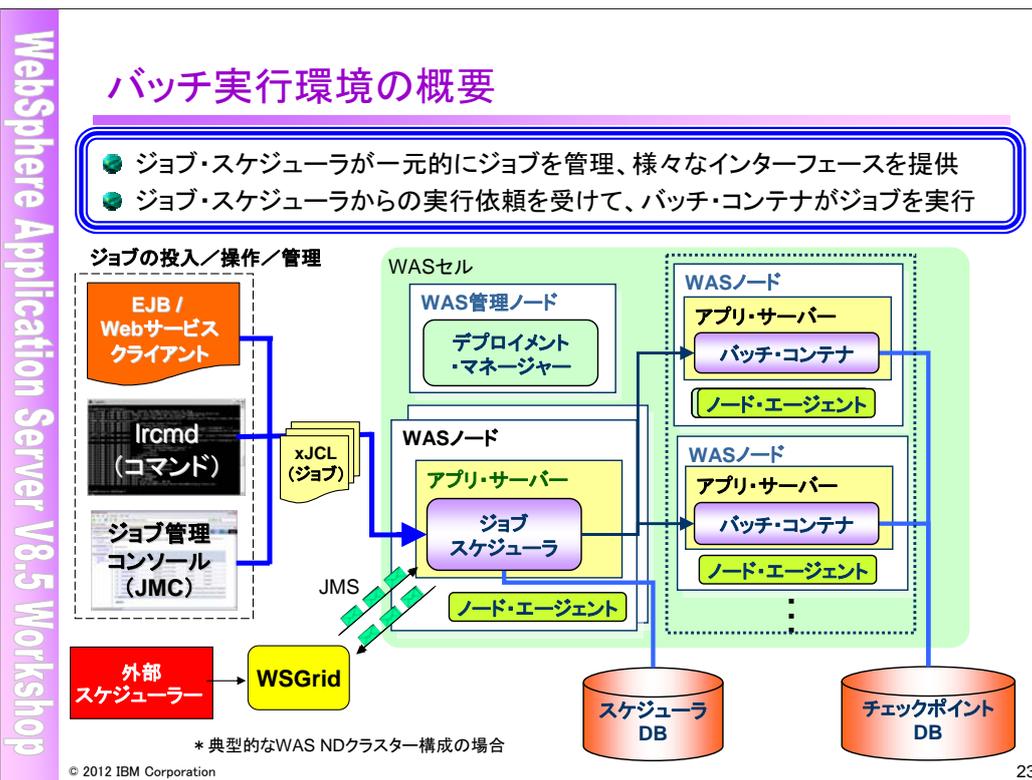
【参考】xJCLサンプル(一部)

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample" default-application-name="SampleBatchApp" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jndi-name>ejb/com.ibm/ws/batch/SampleBatchAppBatchController</jndi-name>
  <step-scheduling-criteria>
    <scheduling-mode>sequential</scheduling-mode>
  </step-scheduling-criteria>
  <checkpoint-algorithm name="recordbased">
    <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</classname>
    <props>
      <prop name="recordcount" value="5" />
    </props>
  </checkpoint-algorithm>
  <results-algorithms>
    <results-algorithm name="jobsum">
      <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
    </results-algorithm>
  </results-algorithms>
  <substitution-props>
    <prop name="sample.input.file" value="/tmp/indata.txt" />
    <prop name="sample.output.file" value="/tmp/outdata.txt" />
  </substitution-props>
  <job-step name="Step1">
    <classname>com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep</classname>
    <checkpoint-algorithm-ref name="recordbased" />
    <results-ref name="jobsum" />
    <batch-data-streams>
      <bds>
        <logical-name>inputStream</logical-name>
        <props>
          <prop name="IMPLCLASS" value="mysample.HelloBatchFileReader" />
          <prop name="FILENAME" value="{sample.input.file}" />
        </props>
        <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileReader</impl-class>
      </bds>
    </batch-data-streams>
  </job-step>
  . . . . .
</job>
```

© 2012 IBM Corporation

22

xJCLの具体的なサンプルです。後半は割愛しています。



バッチ・ジョブ・ステップやバッチ・データストリームといったクラスを開発して、バッチ・コンテナにデプロイし、ジョブ処理の内容を記載したxJCLを作成したら、いよいよバッチ・ジョブの実行です。

ジョブの実行、及び、制御のインターフェースは、3つあります。

1つは、ジョブ管理コンソールで、グラフィカルなWeb UIによる操作環境を提供します。

2つめは、コマンド・ラインツール、Ircmdです。xJCLや各種パラメータを指定して実行することで、サブミットができます。また、サブミット時のリターンに含まれるジョブIDを元に、色々な操作を行うことが可能です。

更に、このようなジョブ操作をプログラムから行いたい場合のために、EJB、或いは、WebサービスによるAPIを提供しています。

これらのインターフェースについて共通する点として、ジョブのサブミットが成功した時点で、制御がクライアント側に戻る点です。

ジョブの実行には、数時間かかるケースもありますから、これは当然の動作です。

しかし、いわゆるエンタープライズ・スケジューラ製品、例えば、当社製品であればTivoli Workload Schedulerなどでは、実行が同期で行われることを前提としています。つまり、制御が戻った時点で、「ジョブが終了した」と見なして、次の実行を開始してしまいます。

そのため、通常のスケジューラとのジョブ連携のためのユーティリティとしてWSGridが提供されています。外部のスケジューラは、このWSGridを通じて、WASのバッチと連携することができ、企業のジョブ・ネットに、WebSphereバッチを組み込むことが可能になります。WSGridの詳細は、後ほど説明します。

ジョブを実行するWAS側の構成ですが、まず、ジョブを一元的に管理するジョブ・スケジューラがあります。ジョブ・スケジューラは、ジョブ情報をDBに入れて管理しています。

ジョブ・スケジューラは、ジョブを受付けると、アプリケーション・サーバー上で稼動する、いずれかのバッチ・コンテナに、そのジョブの実行を依頼します。バッチ・コンテナは、実際のジョブを実行し、例えば、それがトランザクショナル・バッチの場合は、一定のタイミングで取得したチェック・ポイント情報をDBに書き込みます。

ジョブ管理コンソール

- グラフィカルなWebユーザー・インターフェースによるジョブの管理
- ジョブのサブミット／操作／保存／リスト表示／スケジュール管理

ナビゲーション・ペイン



ジョブの表示画面

ジョブの表示画面のスクリーンショット:

ジョブ・スケジュールにサブミットされたすべてのジョブのリストを表示します。ジョブ・オペレーションを実行するには、ジョブを選択し、「アクションの選択」メニューからアクションを選択し、「適用」をクリックします。フィルター・コントロールを使用して、ジョブ・リストの表示を精製します。

設定

---アクションの選択--- 適用

これまでにサブミットされたジョブのリスト

選択	ジョブ ID	実行依頼者	最終更新	状態	ノード	アプリケーション・サーバー
<input type="checkbox"/>	BatchSample:00001		2012-08-20 07:13:30.787	終了	host31Node02	DC01_host31Node02
<input type="checkbox"/>	BatchSample:00002		2012-08-20 07:20:06.967	終了	host31Node02	DC01_host31Node02
<input type="checkbox"/>	[SampleScheduler]BatchSample:00003		2012-08-20 08:30:09.372	終了	host31Node02	DC01_host31Node02

フィルター対象: 3 合計: 3

---アクションの選択---

- キャンセル
- 除去
- 再始動
- 再開
- 停止
- 中断

ジョブに対する操作

© 2012 IBM Corporation

24

ジョブ管理コンソールの、「ジョブの表示」の画面です。この画面では、ジョブのリストが、表示され、これらのジョブに関する各種操作を行うことができます。操作は、リストから選択できます。また、ジョブ ID をクリックすると、ジョブ・ログが表示されます。ジョブ・ログについては、後ほど説明します。ジョブ管理コンソールで行える操作は、インターフェース、コマンド・ライン・ツール Irmcd、或いは、EJB 及び Web サービスの API から行うことができます。

日時指定によるジョブ実行

- サブミット遅延: サブミット時に、開始時刻を指定することが可能
- スケジュール: 開始日時と間隔(日・週・月)を指定して定時実行

ジョブ管理コンソール:スケジュール作成

スケジュールの作成

作成するスケジュールの名前を指定します。ジョブを

* 名前:
SampleSchedule

* 開始日 (yyyy-MM-dd)
2012 - 09 - 18

* 開始時刻 (HH:mm:ss)
14 : 30 : 00

* 間隔:
毎月

< 戻る 次へ > 終了 キャンセル

毎日、毎週、毎月
から選択

ジョブ管理コンソール:サブミット遅延

ジョブのサブミット

ジョブのサブミット
ジョブとしてサブミットするジョブ定義を指定します。ジョブ定義の元として、ローカル
それらを指定するようプロンプトが出されます。

ジョブ定義
● ローカルファイルシステム
* xJCL へのパスの指定
C:\Projects\Batch\Sample\Batchsample.jcl02 参照...

○ ジョブリポジトリ
* ジョブ名の指定

代替プロシエーの更新
サブミット遅延

* 開始日 (yyyy-MM-dd)
2012 - 09 - 18

* 開始時刻 (HH:mm:ss)
14 : 30 : 00

サブミット キャンセル

実行時刻を設定

© 2012 IBM Corporation

25

通常、ジョブがサブミットされると、実行可能なバッチ・コンテナがあれば、ジョブは即時に実行を開始します。

一方、ジョブの実行日時を指定したい場合があります。

例えば、今、ジョブをサブミットして、後で遅延で一度だけ実行したい場合です。このような遅延実行は、サブミット画面の「サブミット遅延」を選択し、時刻を指定してサブミットすることで行えます。

また、もう一つは、バッチ実行を繰り返し定期的に行いたい場合です。ジョブ実行の「スケジュール」を作成することで、1日、1週間、或いは、一ヶ月の周期で定期的にジョブ実行することができます。

ジョブ・クラス

● ジョブの性質を記述した一連の設定の集まりを予めジョブ・クラスとして定義して、利用

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="SimpleCIEar" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" class="JC_Silver">
  <jndi-name>ejb/com.ibm/ws/ci/SimpleCIEJB</jndi-name>
  <job-step name="Step1">
    <classname>com.ibm.websphere.ci.samples.SimpleCIWork</classname>
    <props>
      <prop name="calculationTimeInSecs" value="30" />
      <prop name="outputFileName" value="C:/yam/02test/SimpleCI-output.txt" />
    </props>
  </job-step>
</job>
```

ジョブ・スケジューラ > ジョブ・クラス > JC_Silver
このジョブ・クラスの設定を指定します。

構成

一連プロパティ

名前
JC_Silver

実行時間と並行数の制限

最大実行時間
600 秒

最大同時ジョブ数
10

ジョブ・ログ制限数

最大クラス スペース
1000

最大ファイル存続期間
30 日

出力キュー制限数

最大ジョブ数
10

最大ジョブ存続期間
30 日

最大実行時間を
越えるとジョブは
中断される
(突き抜け防止)

最大同時ジョブ数
を超えてディス
パッチされない

ジョブログの消しこ
み制御

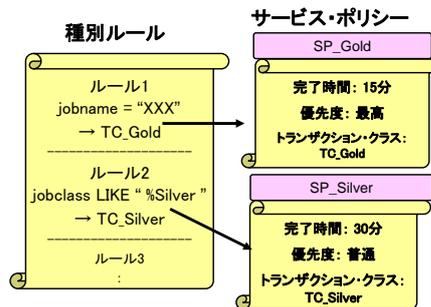
DBのジョブ実行結
果ログ記録レコー
ド消しこみの制御

一連の性質が似通ったジョブをグルーピングするための機能として、メインフレームでお馴染みのジョブ・クラスを設定し利用することができます。最大実行時間や最大同時実行ジョブ数、後述するジョブ・ログの消去のタイミング等々をジョブ・クラスに対して設定できます。

ジョブのxJCL中で、このジョブ・クラスを指定すれば、適切なスケジューリングのための情報をスケジューラに与えることができます。

ジョブ優先度制御 New

- サービス・ポリシー
 - 「目標完了時間」と「重要度」を設定
 - ジョブのディスパッチ先の選択、及び、実行順序の優先度に反映
 - 各ジョブとサービス・ポリシーとの紐付けは、種別ルールに記述
- 種別ルール
 - ジョブ実行依頼者やジョブクラス、ジョブ名等のルールによって、各ジョブをトランザクション・クラスに対応付け
 - トランザクション・クラスに対応するサービス・ポリシーが適用される



© 2012 IBM Corporation

27

ジョブのスケジューリングに対して、優先度を設定することもできます。

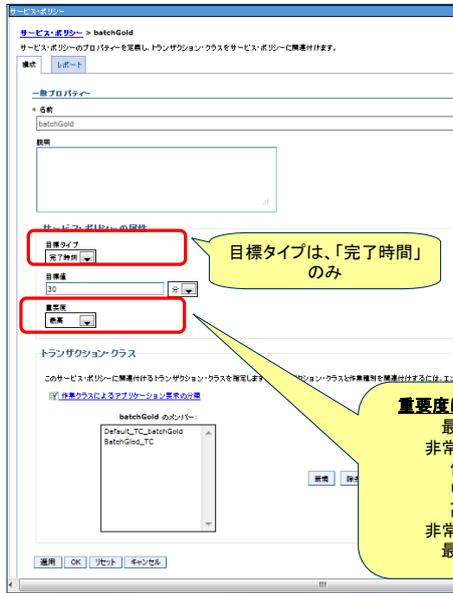
優先度や目標完了時間といった情報をサービス・ポリシーとして定義します。

サブミットされたジョブがどのサービス・ポリシーに従って実行されるかは、種別ルールに記述します。

例えば、ジョブ名や、ジョブ・クラス、或いは、ジョブをサブミットしたユーザーIDなども指定することができます。

【参考】ジョブ優先度制御関連の管理コンソール画面

管理コンソール: サービス・ポリシー画面



管理コンソール: 種別ルール画面

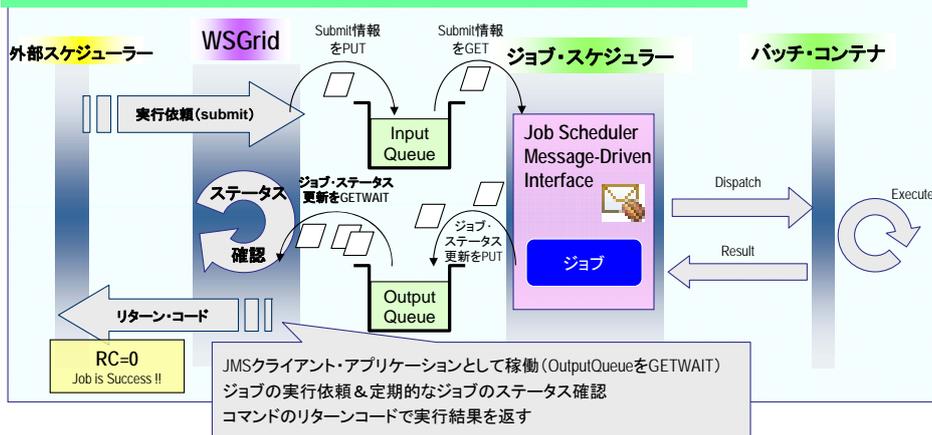


左側は、サービス・ポリシーの設定画面、右側が、種別ルールの設定画面になります。これらの設定は、ジョブ管理コンソールではなく、WASの管理コンソールで行います。

外部スケジューラー連携 New

- 外部スケジューラーと連携してより複雑なジョブ・スケジューリングや既存の企業内ジョブ・ネットと連携
- WSGrid は非同期のジョブ投入/完了を同期に見せ外部スケジューラーと連携

WSGridユーティリティによる外部スケジューラー連携の仕組み



© 2012 IBM Corporation

29

先にお話したように、外部スケジューラーと連携するために、ジョブ実行の完了を待つ制御を戻すためのWSGridユーティリティが提供されています。

ここでは、その大まかな動作概要です。

WSGridは、実行依頼とともに起動されて、サブミット情報を送信側のJMSキューにPUTします。ジョブ・スケジューラーは、その情報を取り出して、バッチ・コンテナに実行依頼します。この間、WSGridは、受信側のJMSキューをGET Waitしています。ジョブ・スケジューラーからジョブ完了のメッセージを受け取った時点で、WSGridは実行を終了して外部スケジューラーに制御を戻します。このときに、正常終了であれば、リターンコード0を返します。

ロギング機能

■ ジョブログ

- ◆ スケジューラー及びバッチ・コンテナが出力するジョブ実行に関する詳細なログ
- ◆ ジョブ管理コンソールから参照及びダウンロードが可能
- ◆ ジョブからの出力を制御できるジョブログ・フィルターSPI
- ◆ 次のディレクトリー下2つのファイル(part.0.log & part.1.log)に出力
 <WASプロファイル・ディレクトリー>/joblogs/<JOB_ID>/<yyyymmdd_HHMMSS>/
 (yyyy: 西暦年/mm:月/dd:日/HH:時刻/MM:分/SS:秒)

New

■ 使用状況の記録(チャージバック・アカウント機能)

- ◆ 課金用の情報をDBに記録
- ◆ ジョブ毎に、消費CPUを 10^{-9} 秒単位で記録
- ◆ 直接SQLで情報を取り出すことも可能
- ◆ zOS版では、SMFへの記録も可能

ログや記録に関する2つの機能を紹介します。

1つ目は、ジョブ・ログで、実行しているxJCLをはじめ、詳細の実行状況のログを出力します。ジョブ・ログの内容は、すでに述べたように、ジョブ管理コンソールから参照できます。また、必要に応じて、ジョブ管理コンソールの画面から、ジョブ・ログファイルをダウンロードすることも可能です。次ページに、この画面のキャプチャを添付してありますので、参照ください。

2つ目は、いわゆる課金のための、使用状況を記録するための、チャージバック・アカウント機能です。各ジョブで使用したCPU時間を 10^{-9} 秒単位でDBに記録します。情報は、直接SQLで取り出すことも可能です。zOS版では、お馴染みのSMFへ記録することも可能です。

【参考】ジョブ管理コンソールでのジョブ・ログ参照

ジョブ管理コンソールから一括してジョブ・ログを確認可能

ジョブ・ログを表示 (Page1: グリッド・スケジューラーの出力)

```

CWRB6721 07/19/07 17:11:28 875 JST Processing w3ci symbolic was
CWRB64501 07/19/07 17:11:28 875 JST List of substitution-props:
CWRB6701 07/19/07 17:11:28 875 JST pjobstep_loop_max=10
CWRB6701 07/19/07 17:11:28 875 JST checkpoint=recordbased
CWRB64541 07/19/07 17:11:28 875 JST List of properties passed to
CWRB6701 07/19/07 17:11:28 875 JST pjobstep_loop_max=10
CWRB6701 07/19/07 17:11:28 875 JST checkpoint=recordbased
CWRB6701 07/19/07 17:11:28 875 JST checkpointInterval=1
CWRB64501 07/19/07 17:11:28 875 JST List of properties to be app
CWRB6701 07/19/07 17:11:28 875 JST pjobstep_loop_max=10
CWRB6701 07/19/07 17:11:28 875 JST checkpoint=recordbased
CWRB6701 07/19/07 17:11:28 875 JST checkpointInterval=1
CWRB64501 07/19/07 17:11:28 875 JST Job w3ci before symbolic was
1 <?xml version="1.0" encoding="UTF-8"?>
2 <job name="XD61BatchTest01" xmlns="http://www.w3.org/2001
3 <!--
4 <!--
5 <!--
6 <!--
7 <!--
8 <!--
9 <!--
10 <!--
11 <!--
12 <!--
13 <!--
14 <!--
15 <!--
16 <!--
17 <!--
18 <!--
19 <!--
20 <!--
21 <!--
22 <!--
23 <!--
24 <!--
25 <!--
26 <!--
27 <!--
28 <!--
29 <!--
30 <!--
31 <!--
32 <!--
33 <!--
34 <!--
35 <!--
36 <!--
37 <!--
38 <!--
39 <!--
40 <!--
41 <!--
42 <!--
43 <!--
44 <!--
45 <!--
46 <!--
47 <!--
48 <!--
49 <!--
50 <!--
51 <!--
52 <!--
53 <!--
54 <!--
55 <!--
56 <!--
57 <!--
58 <!--
59 <!--
60 <!--
61 <!--
62 <!--
63 <!--
64 <!--
65 <!--
66 <!--
67 <!--
68 <!--
69 <!--
70 <!--
71 <!--
72 <!--
73 <!--
74 <!--
75 <!--
76 <!--
77 <!--
78 <!--
79 <!--
80 <!--
81 <!--
82 <!--
83 <!--
84 <!--
85 <!--
86 <!--
87 <!--
88 <!--
89 <!--
90 <!--
91 <!--
92 <!--
93 <!--
94 <!--
95 <!--
96 <!--
97 <!--
98 <!--
99 <!--
100 <!--
101 <!--
102 <!--
103 <!--
104 <!--
105 <!--
106 <!--
107 <!--
108 <!--
109 <!--
110 <!--
111 <!--
112 <!--
113 <!--
114 <!--
115 <!--
116 <!--
117 <!--
118 <!--
119 <!--
120 <!--
121 <!--
122 <!--
123 <!--
124 <!--
125 <!--
126 <!--
127 <!--
128 <!--
129 <!--
130 <!--
131 <!--
132 <!--
133 <!--
134 <!--
135 <!--
136 <!--
137 <!--
138 <!--
139 <!--
140 <!--
141 <!--
142 <!--
143 <!--
144 <!--
145 <!--
146 <!--
147 <!--
148 <!--
149 <!--
150 <!--
151 <!--
152 <!--
153 <!--
154 <!--
155 <!--
156 <!--
157 <!--
158 <!--
159 <!--
160 <!--
161 <!--
162 <!--
163 <!--
164 <!--
165 <!--
166 <!--
167 <!--
168 <!--
169 <!--
170 <!--
171 <!--
172 <!--
173 <!--
174 <!--
175 <!--
176 <!--
177 <!--
178 <!--
179 <!--
180 <!--
181 <!--
182 <!--
183 <!--
184 <!--
185 <!--
186 <!--
187 <!--
188 <!--
189 <!--
190 <!--
191 <!--
192 <!--
193 <!--
194 <!--
195 <!--
196 <!--
197 <!--
198 <!--
199 <!--
200 <!--
201 <!--
202 <!--
203 <!--
204 <!--
205 <!--
206 <!--
207 <!--
208 <!--
209 <!--
210 <!--
211 <!--
212 <!--
213 <!--
214 <!--
215 <!--
216 <!--
217 <!--
218 <!--
219 <!--
220 <!--
221 <!--
222 <!--
223 <!--
224 <!--
225 <!--
226 <!--
227 <!--
228 <!--
229 <!--
230 <!--
231 <!--
232 <!--
233 <!--
234 <!--
235 <!--
236 <!--
237 <!--
238 <!--
239 <!--
240 <!--
241 <!--
242 <!--
243 <!--
244 <!--
245 <!--
246 <!--
247 <!--
248 <!--
249 <!--
250 <!--
251 <!--
252 <!--
253 <!--
254 <!--
255 <!--
256 <!--
257 <!--
258 <!--
259 <!--
260 <!--
261 <!--
262 <!--
263 <!--
264 <!--
265 <!--
266 <!--
267 <!--
268 <!--
269 <!--
270 <!--
271 <!--
272 <!--
273 <!--
274 <!--
275 <!--
276 <!--
277 <!--
278 <!--
279 <!--
280 <!--
281 <!--
282 <!--
283 <!--
284 <!--
285 <!--
286 <!--
287 <!--
288 <!--
289 <!--
290 <!--
291 <!--
292 <!--
293 <!--
294 <!--
295 <!--
296 <!--
297 <!--
298 <!--
299 <!--
300 <!--
301 <!--
302 <!--
303 <!--
304 <!--
305 <!--
306 <!--
307 <!--
308 <!--
309 <!--
310 <!--
311 <!--
312 <!--
313 <!--
314 <!--
315 <!--
316 <!--
317 <!--
318 <!--
319 <!--
320 <!--
321 <!--
322 <!--
323 <!--
324 <!--
325 <!--
326 <!--
327 <!--
328 <!--
329 <!--
330 <!--
331 <!--
332 <!--
333 <!--
334 <!--
335 <!--
336 <!--
337 <!--
338 <!--
339 <!--
340 <!--
341 <!--
342 <!--
343 <!--
344 <!--
345 <!--
346 <!--
347 <!--
348 <!--
349 <!--
350 <!--
351 <!--
352 <!--
353 <!--
354 <!--
355 <!--
356 <!--
357 <!--
358 <!--
359 <!--
360 <!--
361 <!--
362 <!--
363 <!--
364 <!--
365 <!--
366 <!--
367 <!--
368 <!--
369 <!--
370 <!--
371 <!--
372 <!--
373 <!--
374 <!--
375 <!--
376 <!--
377 <!--
378 <!--
379 <!--
380 <!--
381 <!--
382 <!--
383 <!--
384 <!--
385 <!--
386 <!--
387 <!--
388 <!--
389 <!--
390 <!--
391 <!--
392 <!--
393 <!--
394 <!--
395 <!--
396 <!--
397 <!--
398 <!--
399 <!--
400 <!--
401 <!--
402 <!--
403 <!--
404 <!--
405 <!--
406 <!--
407 <!--
408 <!--
409 <!--
410 <!--
411 <!--
412 <!--
413 <!--
414 <!--
415 <!--
416 <!--
417 <!--
418 <!--
419 <!--
420 <!--
421 <!--
422 <!--
423 <!--
424 <!--
425 <!--
426 <!--
427 <!--
428 <!--
429 <!--
430 <!--
431 <!--
432 <!--
433 <!--
434 <!--
435 <!--
436 <!--
437 <!--
438 <!--
439 <!--
440 <!--
441 <!--
442 <!--
443 <!--
444 <!--
445 <!--
446 <!--
447 <!--
448 <!--
449 <!--
450 <!--
451 <!--
452 <!--
453 <!--
454 <!--
455 <!--
456 <!--
457 <!--
458 <!--
459 <!--
460 <!--
461 <!--
462 <!--
463 <!--
464 <!--
465 <!--
466 <!--
467 <!--
468 <!--
469 <!--
470 <!--
471 <!--
472 <!--
473 <!--
474 <!--
475 <!--
476 <!--
477 <!--
478 <!--
479 <!--
480 <!--
481 <!--
482 <!--
483 <!--
484 <!--
485 <!--
486 <!--
487 <!--
488 <!--
489 <!--
490 <!--
491 <!--
492 <!--
493 <!--
494 <!--
495 <!--
496 <!--
497 <!--
498 <!--
499 <!--
500 <!--
501 <!--
502 <!--
503 <!--
504 <!--
505 <!--
506 <!--
507 <!--
508 <!--
509 <!--
510 <!--
511 <!--
512 <!--
513 <!--
514 <!--
515 <!--
516 <!--
517 <!--
518 <!--
519 <!--
520 <!--
521 <!--
522 <!--
523 <!--
524 <!--
525 <!--
526 <!--
527 <!--
528 <!--
529 <!--
530 <!--
531 <!--
532 <!--
533 <!--
534 <!--
535 <!--
536 <!--
537 <!--
538 <!--
539 <!--
540 <!--
541 <!--
542 <!--
543 <!--
544 <!--
545 <!--
546 <!--
547 <!--
548 <!--
549 <!--
550 <!--
551 <!--
552 <!--
553 <!--
554 <!--
555 <!--
556 <!--
557 <!--
558 <!--
559 <!--
560 <!--
561 <!--
562 <!--
563 <!--
564 <!--
565 <!--
566 <!--
567 <!--
568 <!--
569 <!--
570 <!--
571 <!--
572 <!--
573 <!--
574 <!--
575 <!--
576 <!--
577 <!--
578 <!--
579 <!--
580 <!--
581 <!--
582 <!--
583 <!--
584 <!--
585 <!--
586 <!--
587 <!--
588 <!--
589 <!--
590 <!--
591 <!--
592 <!--
593 <!--
594 <!--
595 <!--
596 <!--
597 <!--
598 <!--
599 <!--
600 <!--
601 <!--
602 <!--
603 <!--
604 <!--
605 <!--
606 <!--
607 <!--
608 <!--
609 <!--
610 <!--
611 <!--
612 <!--
613 <!--
614 <!--
615 <!--
616 <!--
617 <!--
618 <!--
619 <!--
620 <!--
621 <!--
622 <!--
623 <!--
624 <!--
625 <!--
626 <!--
627 <!--
628 <!--
629 <!--
630 <!--
631 <!--
632 <!--
633 <!--
634 <!--
635 <!--
636 <!--
637 <!--
638 <!--
639 <!--
640 <!--
641 <!--
642 <!--
643 <!--
644 <!--
645 <!--
646 <!--
647 <!--
648 <!--
649 <!--
650 <!--
651 <!--
652 <!--
653 <!--
654 <!--
655 <!--
656 <!--
657 <!--
658 <!--
659 <!--
660 <!--
661 <!--
662 <!--
663 <!--
664 <!--
665 <!--
666 <!--
667 <!--
668 <!--
669 <!--
670 <!--
671 <!--
672 <!--
673 <!--
674 <!--
675 <!--
676 <!--
677 <!--
678 <!--
679 <!--
680 <!--
681 <!--
682 <!--
683 <!--
684 <!--
685 <!--
686 <!--
687 <!--
688 <!--
689 <!--
690 <!--
691 <!--
692 <!--
693 <!--
694 <!--
695 <!--
696 <!--
697 <!--
698 <!--
699 <!--
700 <!--
701 <!--
702 <!--
703 <!--
704 <!--
705 <!--
706 <!--
707 <!--
708 <!--
709 <!--
710 <!--
711 <!--
712 <!--
713 <!--
714 <!--
715 <!--
716 <!--
717 <!--
718 <!--
719 <!--
720 <!--
721 <!--
722 <!--
723 <!--
724 <!--
725 <!--
726 <!--
727 <!--
728 <!--
729 <!--
730 <!--
731 <!--
732 <!--
733 <!--
734 <!--
735 <!--
736 <!--
737 <!--
738 <!--
739 <!--
740 <!--
741 <!--
742 <!--
743 <!--
744 <!--
745 <!--
746 <!--
747 <!--
748 <!--
749 <!--
750 <!--
751 <!--
752 <!--
753 <!--
754 <!--
755 <!--
756 <!--
757 <!--
758 <!--
759 <!--
760 <!--
761 <!--
762 <!--
763 <!--
764 <!--
765 <!--
766 <!--
767 <!--
768 <!--
769 <!--
770 <!--
771 <!--
772 <!--
773 <!--
774 <!--
775 <!--
776 <!--
777 <!--
778 <!--
779 <!--
780 <!--
781 <!--
782 <!--
783 <!--
784 <!--
785 <!--
786 <!--
787 <!--
788 <!--
789 <!--
790 <!--
791 <!--
792 <!--
793 <!--
794 <!--
795 <!--
796 <!--
797 <!--
798 <!--
799 <!--
800 <!--
801 <!--
802 <!--
803 <!--
804 <!--
805 <!--
806 <!--
807 <!--
808 <!--
809 <!--
810 <!--
811 <!--
812 <!--
813 <!--
814 <!--
815 <!--
816 <!--
817 <!--
818 <!--
819 <!--
820 <!--
821 <!--
822 <!--
823 <!--
824 <!--
825 <!--
826 <!--
827 <!--
828 <!--
829 <!--
830 <!--
831 <!--
832 <!--
833 <!--
834 <!--
835 <!--
836 <!--
837 <!--
838 <!--
839 <!--
840 <!--
841 <!--
842 <!--
843 <!--
844 <!--
845 <!--
846 <!--
847 <!--
848 <!--
849 <!--
850 <!--
851 <!--
852 <!--
853 <!--
854 <!--
855 <!--
856 <!--
857 <!--
858 <!--
859 <!--
860 <!--
861 <!--
862 <!--
863 <!--
864 <!--
865 <!--
866 <!--
867 <!--
868 <!--
869 <!--
870 <!--
871 <!--
872 <!--
873 <!--
874 <!--
875 <!--
876 <!--
877 <!--
878 <!--
879 <!--
880 <!--
881 <!--
882 <!--
883 <!--
884 <!--
885 <!--
886 <!--
887 <!--
888 <!--
889 <!--
890 <!--
891 <!--
892 <!--
893 <!--
894 <!--
895 <!--
896 <!--
897 <!--
898 <!--
899 <!--
900 <!--
901 <!--
902 <!--
903 <!--
904 <!--
905 <!--
906 <!--
907 <!--
908 <!--
909 <!--
910 <!--
911 <!--
912 <!--
913 <!--
914 <!--
915 <!--
916 <!--
917 <!--
918 <!--
919 <!--
920 <!--
921 <!--
922 <!--
923 <!--
924 <!--
925 <!--
926 <!--
927 <!--
928 <!--
929 <!--
930 <!--
931 <!--
932 <!--
933 <!--
934 <!--
935 <!--
936 <!--
937 <!--
938 <!--
939 <!--
940 <!--
941 <!--
942 <!--
943 <!--
944 <!--
945 <!--
946 <!--
947 <!--
948 <!--
949 <!--
950 <!--
951 <!--
952 <!--
953 <!--
954 <!--
955 <!--
956 <!--
957 <!--
958 <!--
959 <!--
960 <!--
961 <!--
962 <!--
963 <!--
964 <!--
965 <!--
966 <!--
967 <!--
968 <!--
969 <!--
970 <!--
971 <!--
972 <!--
973 <!--
974 <!--
975 <!--
976 <!--
977 <!--
978 <!--
979 <!--
980 <!--
981 <!--
982 <!--
983 <!--
984 <!--
985 <!--
986 <!--
987 <!--
988 <!--
989 <!--
990 <!--
991 <!--
992 <!--
993 <!--
994 <!--
995 <!--
996 <!--
997 <!--
998 <!--
999 <!--
1000 <!--

```

ジョブ・ログの表示 (Page2: グリッド実行環境の出力)

```

CWRB65881 07/19/07 17:11:32 859 JST Setting up 32ee job XD61BatchTest01 42 for execution in Grid
CWRB62101 07/19/07 17:11:32 875 JST Job setup manager been in setting up job XD61BatchTest01 42
CWRB64901 07/19/07 17:11:32 203 JST No match found in job status table entry using key: [bjeeaaa
CWRB67401 07/19/07 17:11:34 375 JST Job [XD61BatchTest01 42] is in job setup.
CWRB64701 07/19/07 17:11:35 421 JST Creating abstract resources required by the job.
CWRB67401 07/19/07 17:11:35 607 JST Job [XD61BatchTest01 42] is submitted for execution.
CWRB62301 07/19/07 17:11:36 750 JST Job setup manager been completed job XD61BatchTest01 42 setu
CWRB64901 07/19/07 17:11:38 210 JST Initializing for step dispatch using scheduling mode: sequen
CWRB64901 07/19/07 17:11:38 328 JST Created job step status table entry using key [jobid XD61Bat
CWRB64901 07/19/07 17:11:38 328 JST Found job results table entry matching on key: [jobid XD61Ba
CWRB620301 07/19/07 17:11:38 328 JST Initialization for sequential step dispatch is complete.
CWRB64901 07/19/07 17:11:38 343 JST Dispatching job XD61BatchTest01 42: job contains 1 step(s)
CWRB64901 07/19/07 17:11:38 343 JST Job [XD61BatchTest01 42] Step [Step1] is in step setup
CWRB64221 07/19/07 17:11:40 578 JST Loading job step beans for step Step1 using jndi name = ejb-Mi
System out: 07/19/07 17:11:40 697 JST BatchTestPOJ0Step1: setProperties() is called
System out: 07/19/07 17:11:40 697 JST BatchTestPOJ0Step1: setProperties() this xjcsprops = {pjob
System out: 07/19/07 17:11:40 697 JST BatchTestPOJ0Step1: BatchJobStepInterfaceFormatJobStep(1)
System out: 07/19/07 17:11:40 697 JST BatchTestPOJ0Step1: setProperties() is called
System out: 07/19/07 17:11:40 697 JST set pjobStep_loop_max=10
CWRB65941 07/19/07 17:11:40 774 JST Step Step1 setup is complete ended normally
CWRB64401 07/19/07 17:11:41 796 JST Job [XD61BatchTest01 42] Step [Step1] is dispatched
System out: 07/19/07 17:11:42 812 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 812 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 828 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 828 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 844 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 844 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 859 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 859 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 876 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
System out: 07/19/07 17:11:42 890 JST BatchTestPOJ0Step1: BatchJobStepInterfaceProcessJobStep(
CWRB64401 07/19/07 17:11:42 890 JST Job [XD61BatchTest01 42] Step [Step1] is in step breakdown.
CWRB64601 07/19/07 17:11:43 906 JST Destroying job step Step1
System out: 07/19/07 17:11:43 906 JST BatchTestPOJ0Step1: destroyJobStep() is called
CWRB64801 07/19/07 17:11:43 906 JST Job step Step1 destroy completed with rc: 0
CWRB64801 07/19/07 17:11:43 906 JST Firing Step1 results algorithm com.ibm.waspi.batch.results
CWRB6441 07/19/07 17:11:43 921 JST Stopping step Step1 recordbased checkpoint. User transactio

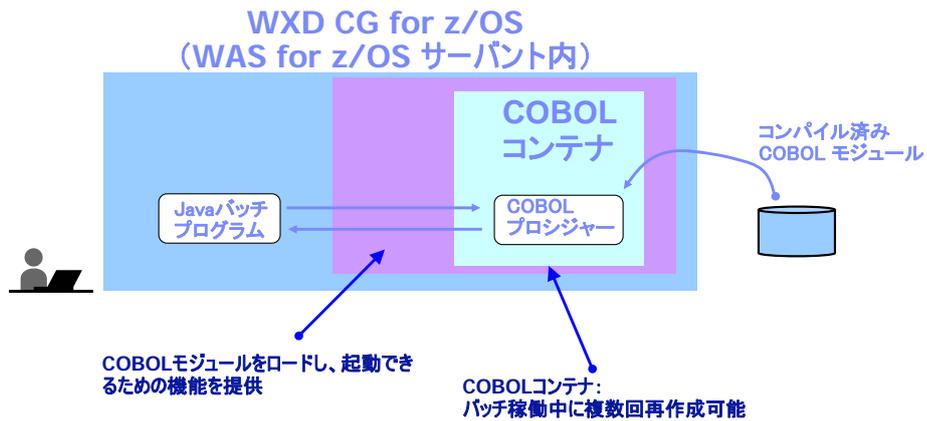
```

ローカルへのダウンロードも可能

ジョブ管理コンソールから、ジョブログを参照している画面です。画面下の「ダウンロード」ボタンをクリックすることで、ダウンロードできます。

【参考】COBOLコンテナ ★ New

- 同一プロセスを使用した「コール・レベル」での統合を実現
- WebSphereバッチ環境で、同一プロセス上にCOBOLのモジュールをロードし、Javaから起動



zOS版固有の機能として、COBOLのモジュールを、バッチコンテナと同じプロセス内で実行することが可能です。詳細は割愛します。

まとめ

まとめ

- **本格的なJava/バッチ環境をWAS標準で提供**
 - Java EE OLTPと実装モジュール、及び、サーバー環境を共有
- **プログラミング・モデルとフレームワークに基づくアプリケーション開発**
 - チェックポイント&リスタート機能を提供するトランザクショナル・バッチ
 - バッチ開発を強かにサポートするフレームワーク及び開発ツール
 - ネイティブ実行による、既存実行モジュールとの連携
 - 大規模ジョブの分割・並列処理を可能にするParallel Job Manager
- **高度なスケジューリング及び、ジョブ連携が可能なジョブ実行環境**
 - ジョブ・スケジューラによる一元的&柔軟なジョブ管理
 - 多様なジョブ管理インターフェース
 - 日時指定や、ジョブ・クラス、優先度に基づくスケジューリング機能
 - エンタープライズ・スケジューラとの連携による、ジョブ・ネット全体との連携が可能

2章でお話したように、WASのバッチ機能は、しっかりとしたプログラミング・モデルに基づき、チェック・ポイント&リスタートや分割・並列処理機能、及び、ネイティブ実行といった機能を提供しています。また、開発を強かにサポートするフレームワークを用意しています。

ランタイム環境では、信頼性の高いバッチ・コンテナに加えて、優先度や、ジョブ・クラスに基づくスケジューリング機能を備えています。

また、スケジューリング機能を補完するとともに、WASバッチを他環境のジョブと連携したジョブ・ネットに組み込むことができる、ジョブ連携機能も準備しています。

このように、WebSphereバッチは、企業内の本格的なバッチ処理を行うために必要十分と思われる機能を提供しています。

参考文献

- WebSphere ライブ WAS道場「知っていますか？ WASでのJavaバッチ処理」
http://www.ibm.com/software/jp/websphere/events/livestream/was_ondemand.html#sec3
- DeveloperWorks記事「作ってみようJavaバッチ ～ WebSphere Application Serverのバッチ機能を使う」
http://www.ibm.com/developerworks/jp/websphere/library/was/was_batch/
- WebSphere XD V6アップデート&デザインワークショップ資料：J2EEバッチシステムのつぼ
http://www.ibm.com/developerworks/jp/websphere/library/wxd/wxd6_updatews/
- DeveloperWorks Japan: WebSphere XD Compute Grid
<http://www.ibm.com/developerworks/jp/websphere/category/wxd/cg.html>
- WebSphere Application Server V8.5 InfoCenter
<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/>
- WebSphere XD Compute Grid V8 InfoCenter
<http://publib.boulder.ibm.com/infocenter/wasinfo/cgwas80/index.jsp>

ワークショップ、セッション、および資料は、IBMまたはセッション発表者によって準備され、それぞれ独自の見解を反映したものです。それらは情報提供の目的のみで提供されており、いかなる参加者に対しても法的またはその他の指導や助言を意図したものではありません。またそのような結果を生むものでもありません。本講演資料に含まれている情報については、完全性と正確性を期するよう努力しましたが、「現状のまま」提供され、明示または暗示にかかわらずいかなる保証も伴わないものとします。本講演資料またはその他の資料の使用によって、あるいはその他の関連によって、いかなる損害が生じた場合も、IBMは責任を負わないものとします。本講演資料に含まれている内容は、IBMまたはそのサプライヤーやライセンス交付者からいかなる保証または表明を引き出すことを意図したもので、IBMソフトウェアの使用を規定する適用ライセンス契約の条項を変更することを意図したものでなく、またそのような結果を生むものでもありません。

本講演資料でIBM製品、プログラム、またはサービスに言及しても、IBMが営業活動を行っているすべての国でそれらが使用可能であることを暗示するものではありません。本講演資料で言及している製品リリース日付や製品機能は、市場機会またはその他の要因に基づいてIBM独自の決定権をもっていつでも変更できるものとし、いかなる方法においても将来の製品または機能が使用可能になると確約することを意図したものではありません。本講演資料に含まれている内容は、参加者が開始する活動によって特定の販売、売上高の向上、またはその他の結果が生じることを述べ、または暗示することを意図したもので、またそのような結果を生むものでもありません。パフォーマンスは、管理された環境において標準的なIBMベンチマークを使用した測定と予測に基づいています。ユーザーが経験する実際のスループットやパフォーマンスは、ユーザーのジョブ・ストリームにおけるマルチプログラミングの量、入出力構成、ストレージ構成、および処理されるワークロードなどの考慮事項を含む、数多くの要因に応じて変化します。したがって、個々のユーザーがここで述べられているものと同様の結果を得られると確約するものではありません。

記述されているすべてのお客様事例は、それらのお客様がどのようにIBM製品を使用したか、またそれらのお客様が達成した結果の実例として示されたものです。実際の環境コストおよびパフォーマンス特性は、お客様ごとに異なる場合があります。

IBM、IBM ロゴ、ibm.com、Rational、WebSphere、およびz/OSは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。現時点での IBM の 商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。

JavaおよびすべてのJava関連の商標およびロゴは Oracleやその関連会社の米国およびその他の国における商標または登録商標です。