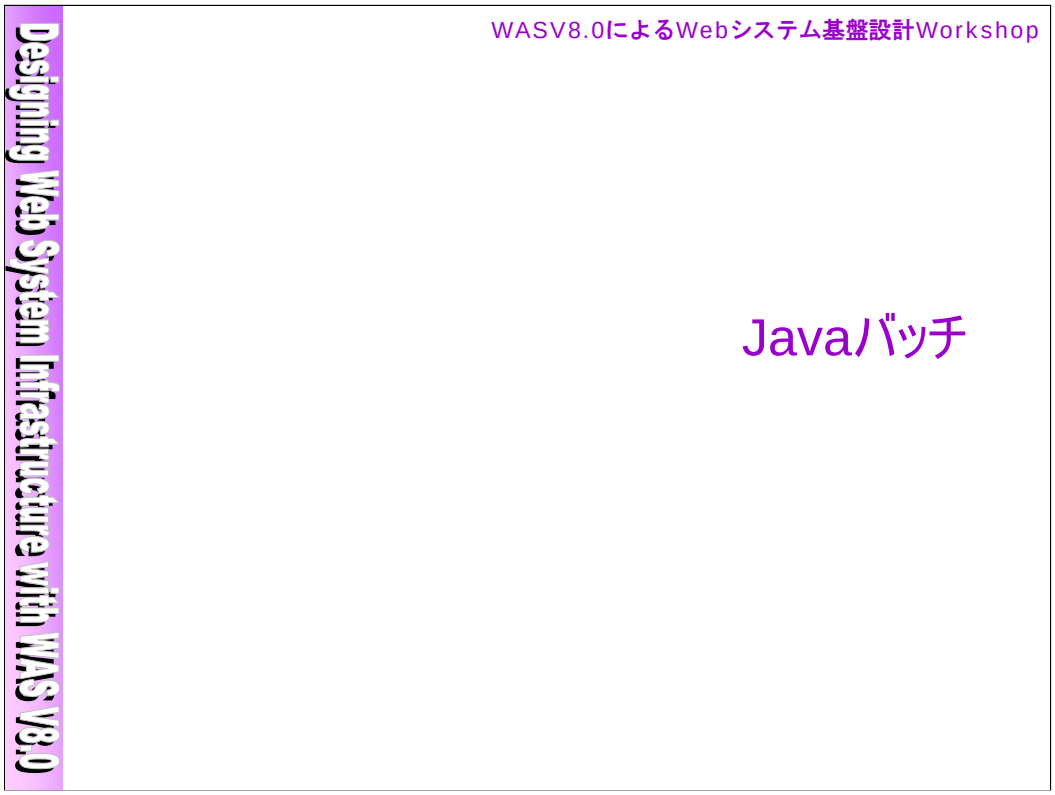

Designing Web System Infrastructure with WAS V8.0
WASV8.0によるWebシステム基盤設計Workshop
Javaバッチ

Designing Web System Infrastructure with WAS V8.0

当セッションの目的

- ミッション・クリティカルな環境におけるJavaバッチ設計に必要な知識の習得
  - WASのJavaバッチ機能を理解する。

© 2012 IBM Corporation

2

Designing Web System Infrastructure with WAS V8.0

Agenda

1. バッチ処理とは

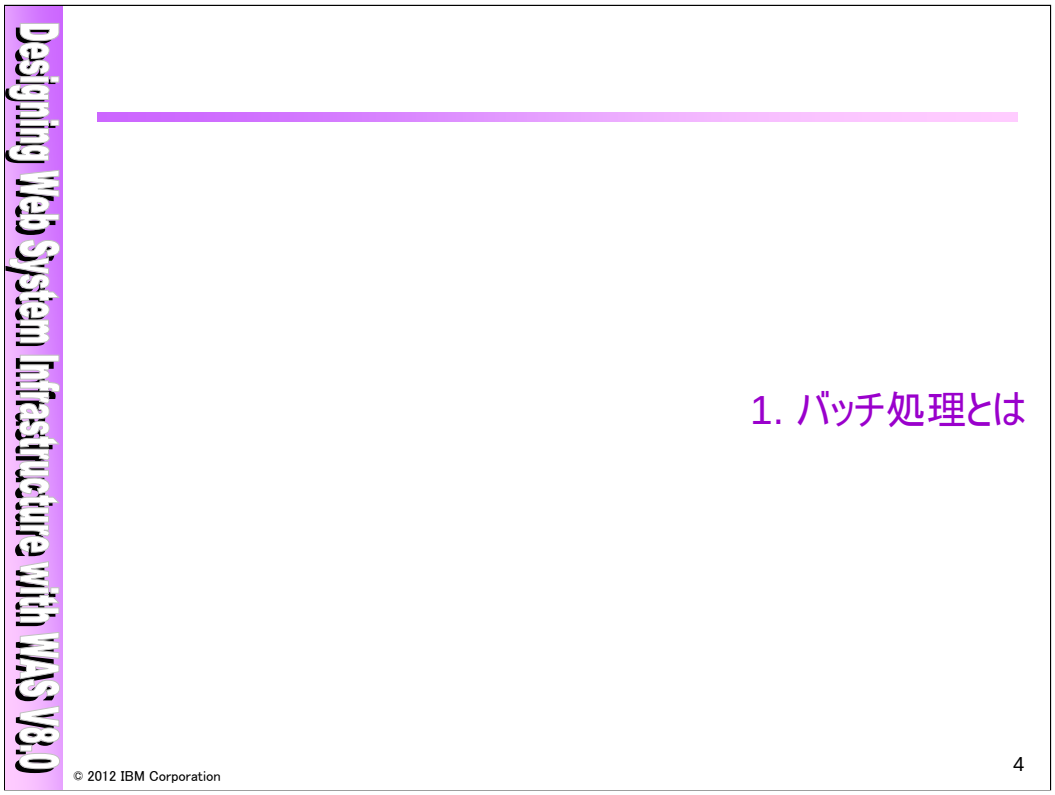
2. バッチ機能概要

3. バッチ機能の特徴

まとめ・参考文献

© 2012 IBM Corporation

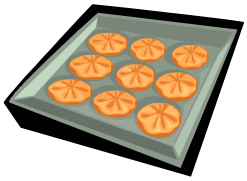
3


Designing Web System Infrastructure with WAS V8.0
1. バッチ処理とは
© 2012 IBM Corporation
4

Designing Web System Infrastructure with WAS V8.0

バッチ処理とは

- 「batch [名詞] 」の意味
  - パンなどの焼き物の1回分の焼き分量
  - 1回で作成される物などの量
  - 1回で処理される物や人の集まり



- バッチ処理 の定義
  - “纏まったデータを一度に処理する”こと
  - データが纏まっていればバッチ処理
- なぜ、”纏めて”処理するのか？
  - 即時性を必要としない処理である
    - 即時性は必要としない → ある程度纏まってから処理できる
  - 纏めて処理する必要がある
    - ある時点にならないとデータが揃わない → 揃ってから纏めて処理する必要がある
    - データが纏まって到着する（連携される） → 纏めて処理する必要がある
  - 纏めて処理した方が効率が良い
    - データが発生する都度処理するのは非効率である → 纏めた方が効率的である

© 2012 IBM Corporation

5

WASは、オンラインのWebアプリケーション実行環境として発展してきましたが、ITインフラではオンライン・アプリケーションだけでなくバッチ処理も重要な役割を担っています。このセッションではWASが提供するバッチ処理機能をご紹介します。

バッチ処理の例

外部イベント契機の非同期・長時間処理

受注を受けて、多数の工程を含む生産計画を作成する（製造業）

工業製品のデザイン変更の影響を、数時間かけてシミュレーション（製造業）

価格変動の膨大な履歴を入力して、数時間かけてリスク分析計算（金融業）

帳票の作成・印刷

定時起動による定型処理（日次/月次/期末の処理など）

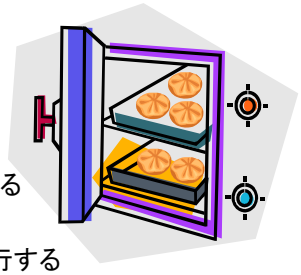
金融機関における振り込み処理

クレジット会社の請求書作成業務

Designing Web System Infrastructure with WAS V8.0

バッチ処理の特性

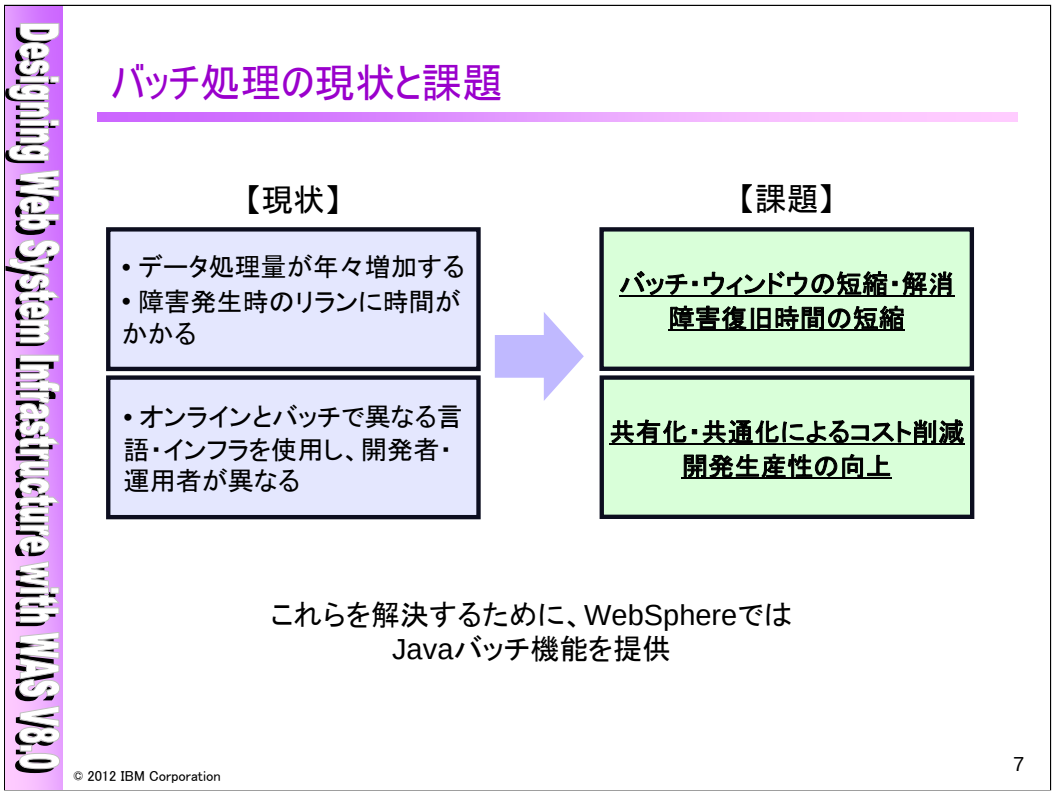
- “データが纏まっている”ことから、バッチ処理ならではの長所と考慮点がある
- バッチ処理の長所
  - 再実行が容易
    - 入力データが消失しないので、再実行が可能である
  - 処理効率を改善できるポイントが多い
    - コミット件数を調整することで、コミット処理のオーバーヘッドを削減できる
      オンライン処理は都度コミットだが、バッチ処理は調整ができる
    - データを処理しやすい順序にソートしてから処理できる
    - データを分類･分割し、並行処理できる
- 考慮点
  - 再実行が容易な単位で、分割処理（または、コミット）する
  - 再実行が容易になるように、処理ステップを分割する
  - バッチ処理ウィンドウに収まるように、分割して並列実行する



© 2012 IBM Corporation

6

バッチ処理とは、データを纏めて処理することです。
データを纏めることによる長所と考慮点を確認しておきましょう。



バッチ処理の重要性とあわせて、バッチ処理が抱える課題もあります。

・バッチウィンドウの短縮・課題

バッチのデータ量増大に伴いバッチ処理時間も増大する一方、インターネット化/グローバル化でオンライン停止時間帯の確保が困難となっています

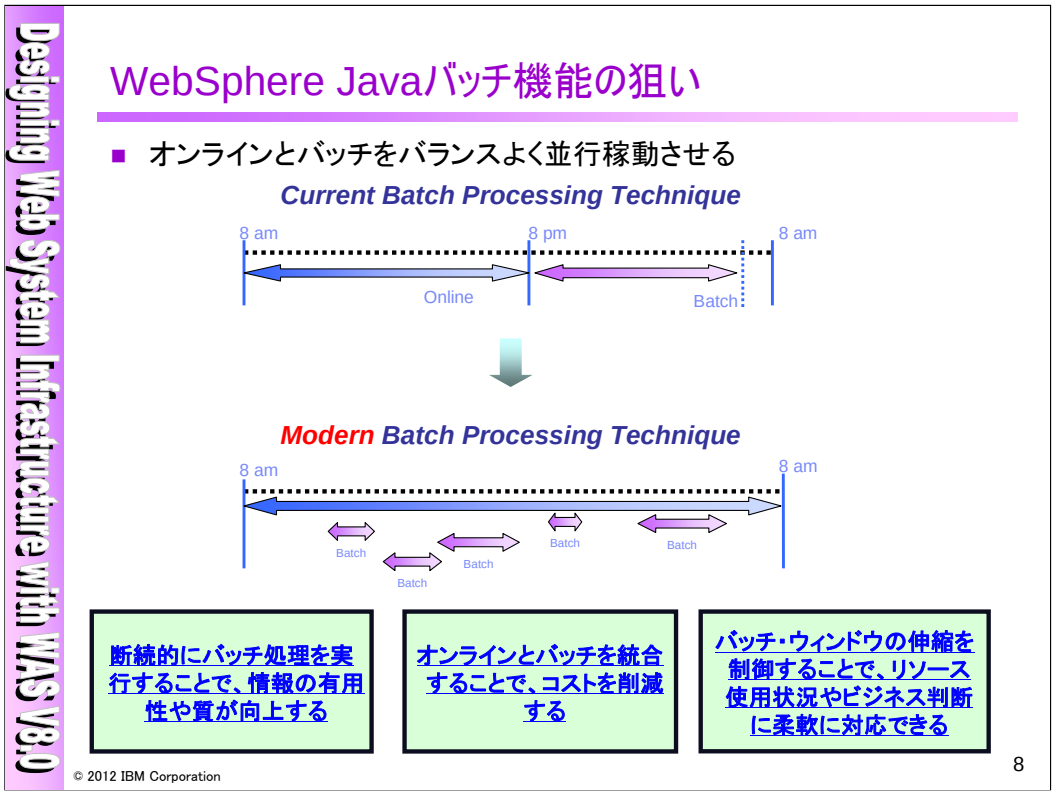
・障害復旧時間の短縮

バッチ処理量が増大すれば障害時の復旧時間も長時間化してしまいます。しかし、バッチ処理が遅延するとオンライン業務へ影響してしまうため、短時間で復旧する必要があります

・共有化・共通化によるコスト削減、開発生産性の向上

既存資産に対応可能な技術者の減少によるスキル共通化や、オンラインとバッチ環境の資源共有によるコスト削減が求められています

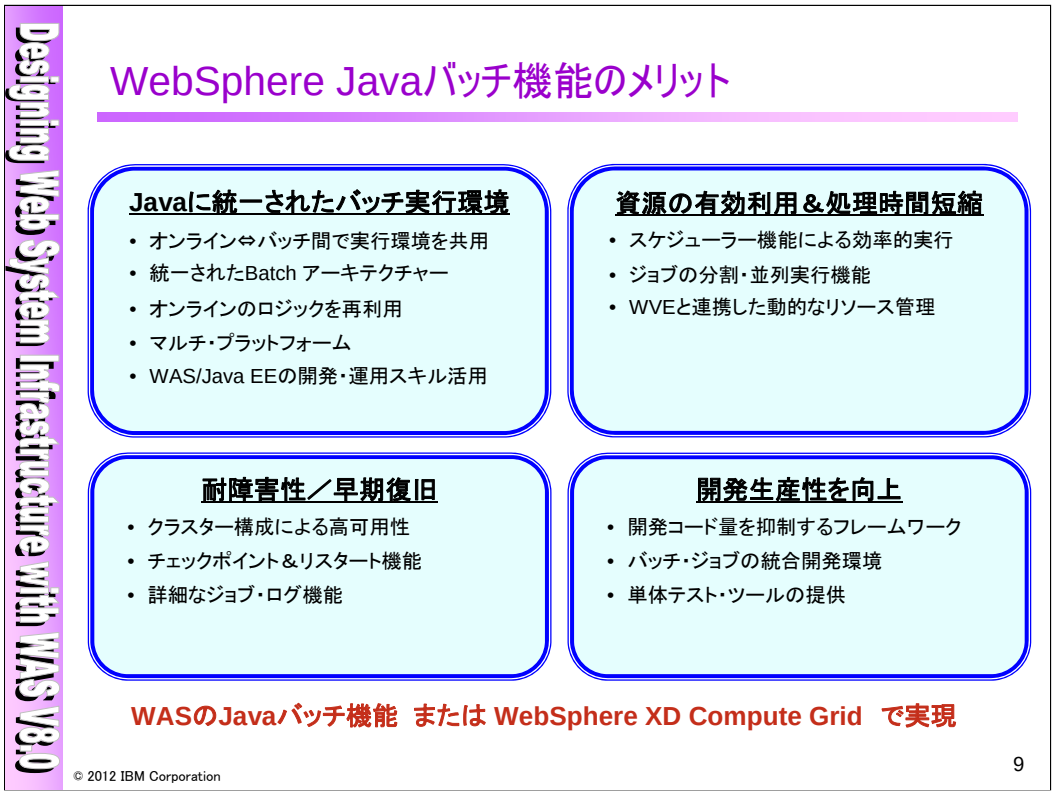
これらの課題を解決するために、WASはJavaバッチ機能を提供しています



そのコンセプトはオンライン処理とバッチ処理をバランスよく並行稼動させることです。

従来のようにオンライン処理時間帯とバッチ処理時間帯を分けるのではなく、24時間365日オンラインのサービスを提供しつつ、バッチ処理を柔軟に行うことを目指しています。

これをモダンバッチという言い方をすることがあります。



WebSphereのバッチ機能を使用することで以下のようなメリットが期待できます。

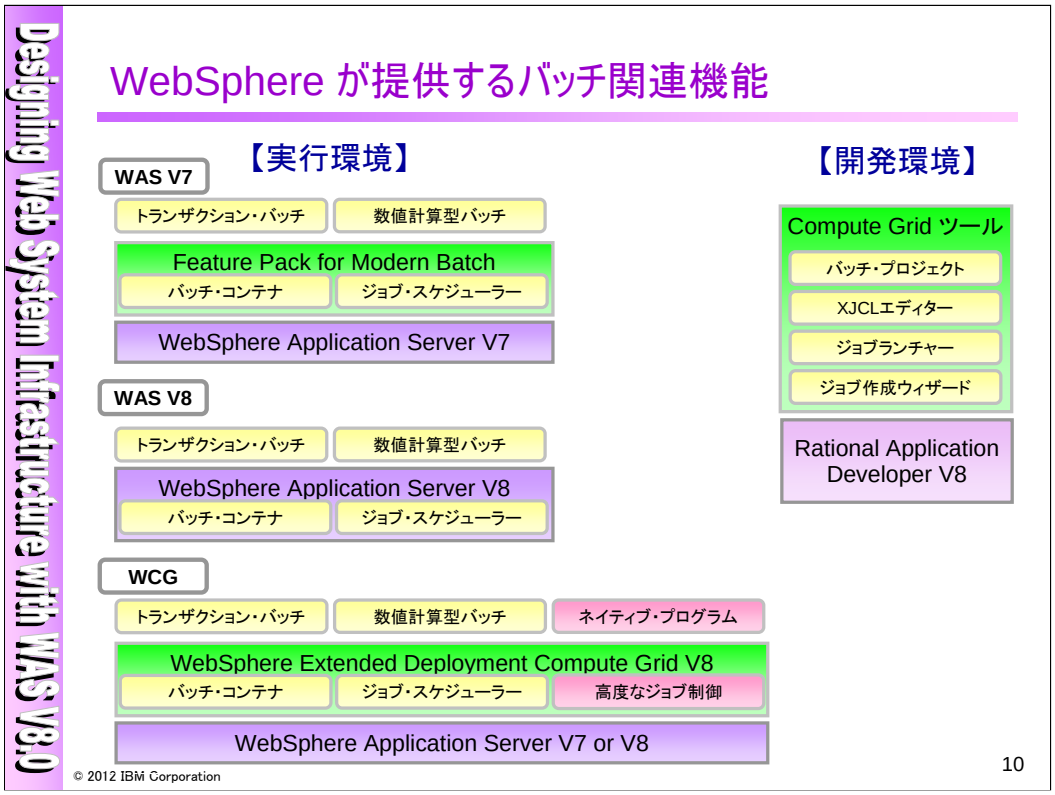
・実績あるJavaEE基盤であるWASの機能をバッチ処理に活用できます。

・Java EEオンラインのロジックをバッチでも利用できるため、ロジックを共通化して重複開発を削減することができます。

・オンラインとバッチ間でサーバー資源を共有できます。

・WASやJava EEの開発・運用スキルをバッチへの適用することができます。

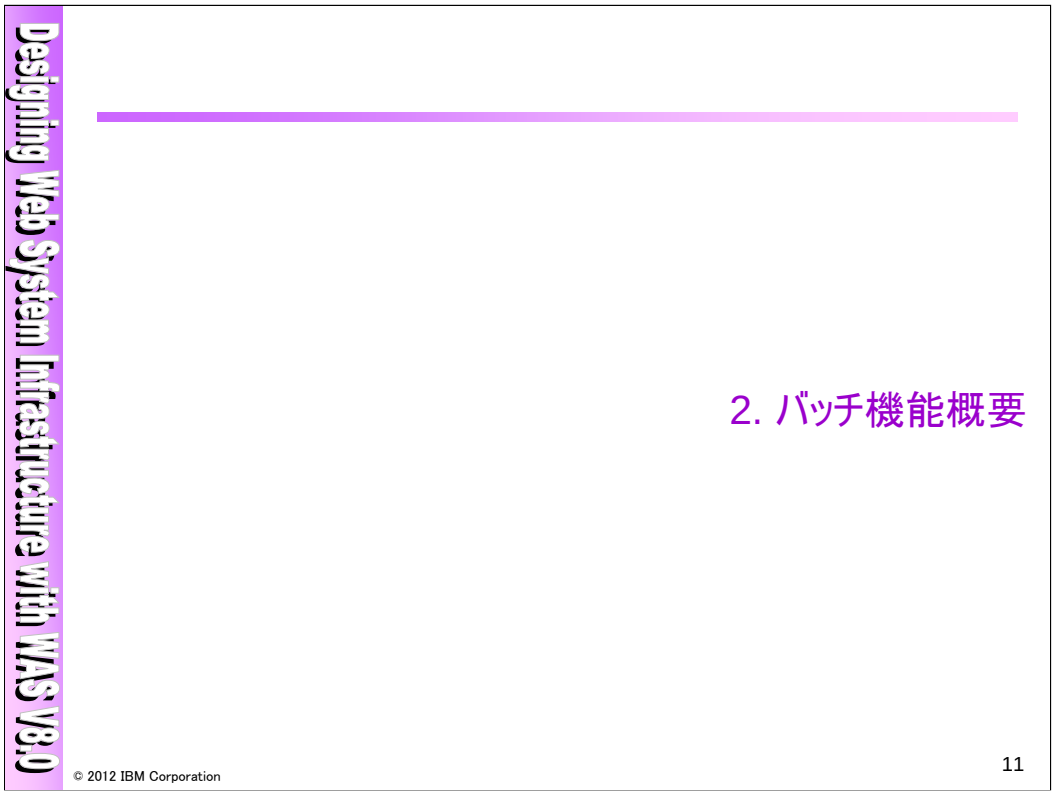
WebSphereのJavaバッチはWASと、WASの上位製品であるWCG（WebSphere XD Compute Grid）で実現できます。

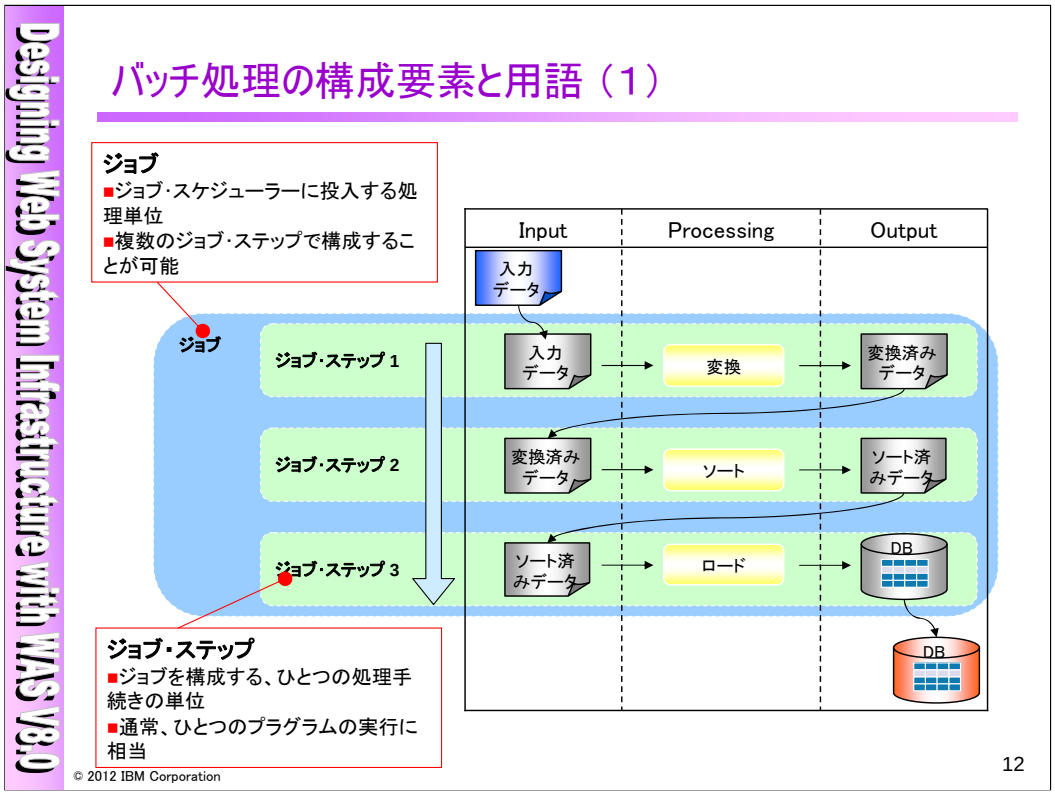


WASのJavaバッチ機能はWAS V7で無料のフィーチャー(FeaturePack for Modern Batch)として提供され、WAS V8ではその機能が標準搭載されました。

WCGはWAS V7またはV8に追加インストールして、さらに高度なバッチ機能を提供します。

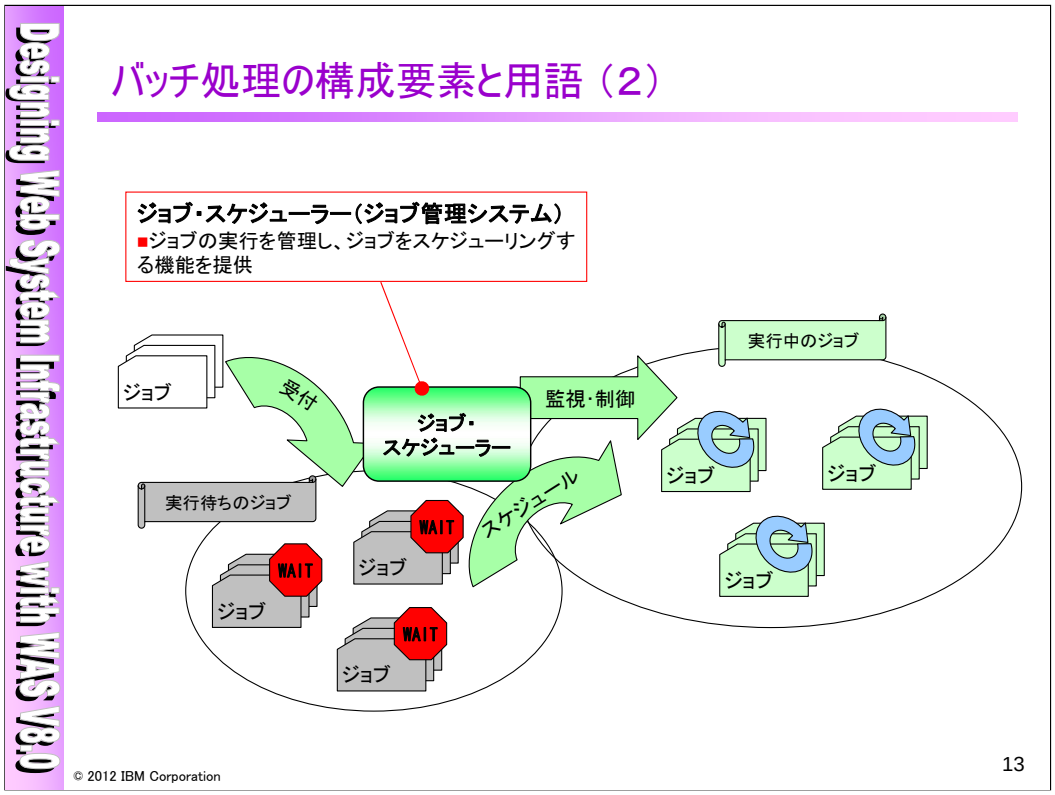
開発環境としてはRAD V8.0.1から、Compute Gridツールを使用して、より効率的にJavaバッチ・アプリケーションを開発できる環境が提供されています。


Designing Web System Infrastructure with WAS V8.0
2. バッチ機能概要
© 2012 IBM Corporation
11

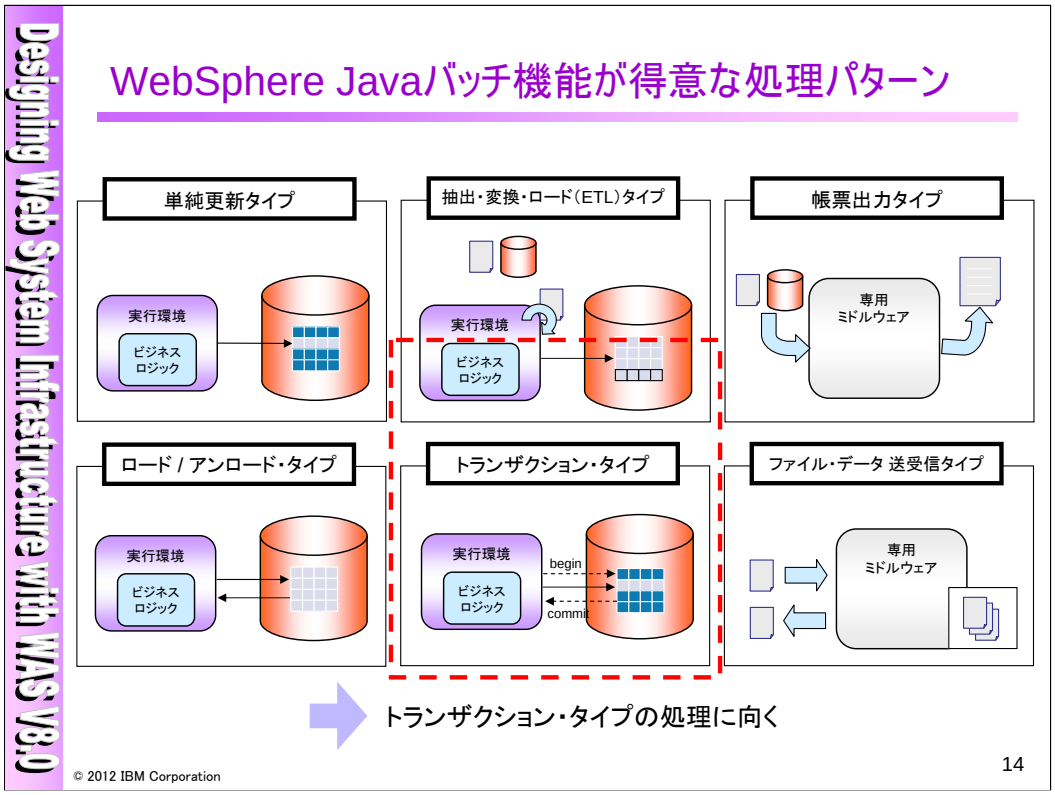


この章では、まずバッチ処理の用語を確認しましょう。

バッチ処理はひとつまたは複数のジョブから構成され、ジョブは複数のジョブ・ステップで構成されます。ジョブとジョブ・ステップの粒度は各プロジェクト毎に異なります。



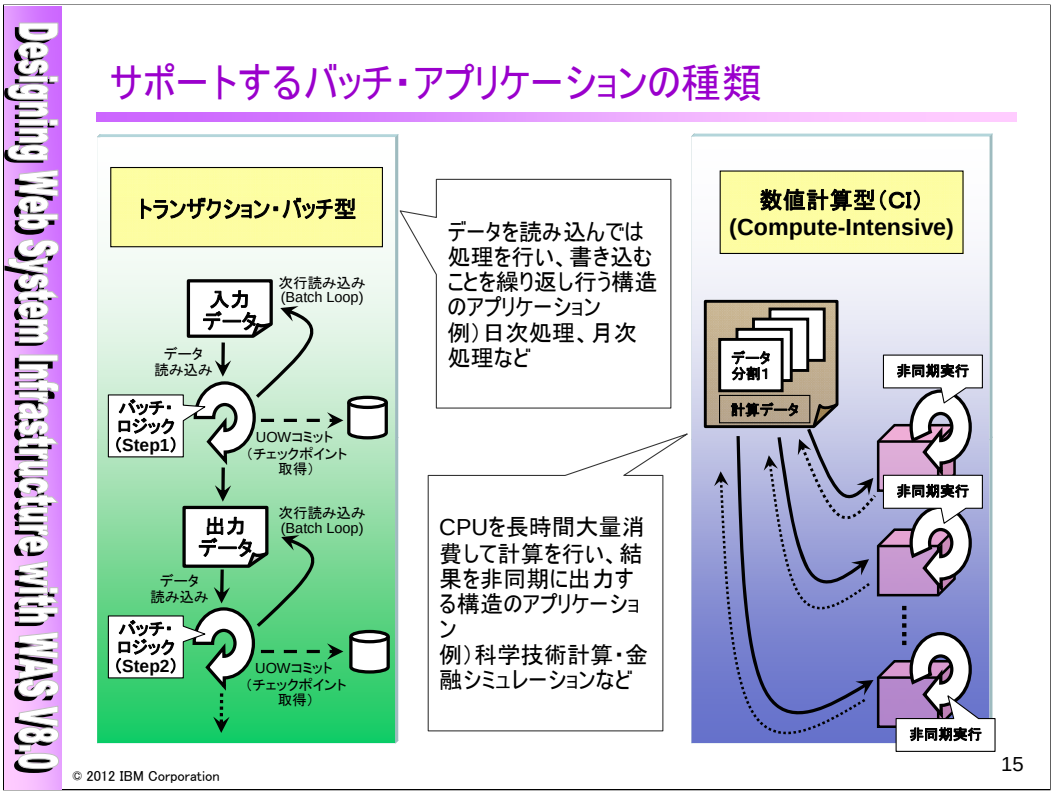
ジョブは、ジョブ・スケジューラーによって実行されます。
ジョブ・スケジューラーは、ジョブを受付け、指定されたスケジュールに従ってそのジョブを実行します。



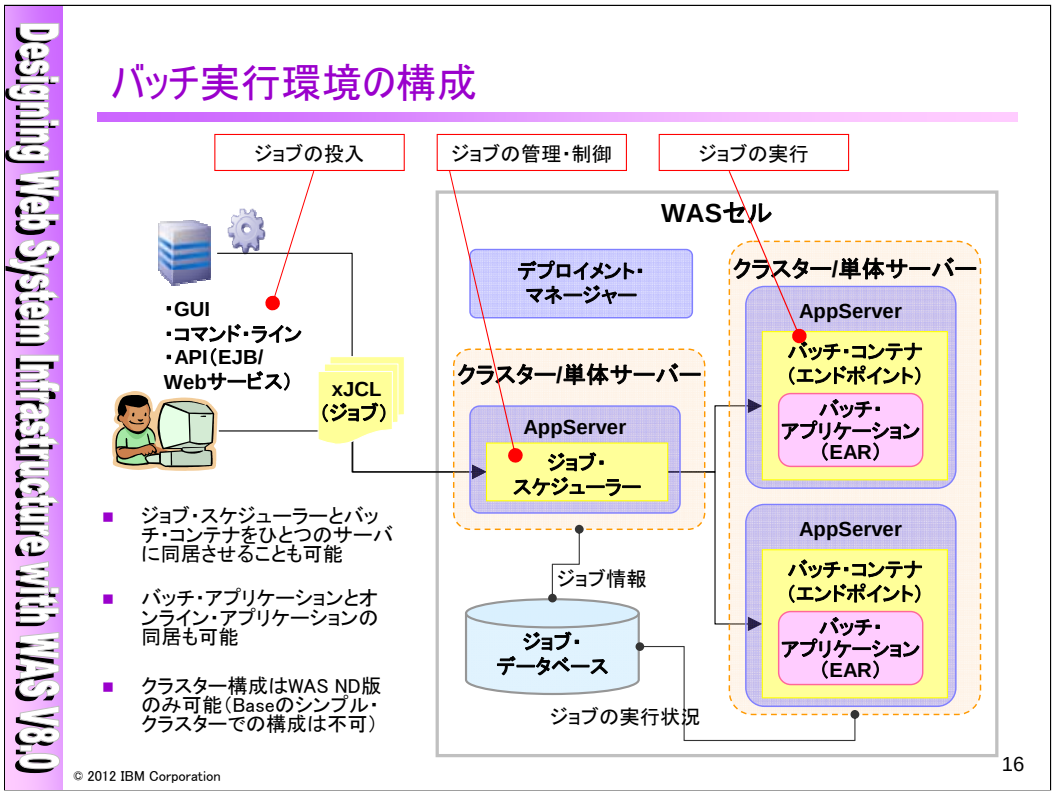
バッチ処理の種別を確認しましょう。一言でバッチ処理といっても、様々な処理形態があり、それぞれの処理に対する実装方法も向き不向きがあります。

たとえば、ファイルのデータをデータベースへロードしたり、データベースのデータをファイルへアンロードするような処理はデータベースのユーティリティーを使うのが現実的です。

WASのバッチ処理は、インプット・データから何らかのトランザクションを発生させて処理するようなトランザクション・タイプのバッチ処理が得意です。



WAS がサポートするバッチ・アプリケーション・タイプは2種類になります。
①トランザクション・バッチ型
入力データを読み込んでは処理を行い、処理結果データをファイルもしくはDBへ書き込むというトランザクションを繰り返し行うタイプのアプリケーションです。日次処理、月次処理などに代表されるいわゆるバッチ処理がこれにあたります。
②数値計算型（CI）（Compute-Intensive）
CPUを大量消費して長時間にわたる計算処理を行い、その結果を非同期に出力するタイプのアプリケーションです。科学技術計算・金融シミュレーションなどのようなものに代表され、入力データを分割して大量のサーバーで計算処理を実行させるいわゆるGrid的なアプリケーションがこれにあたります。



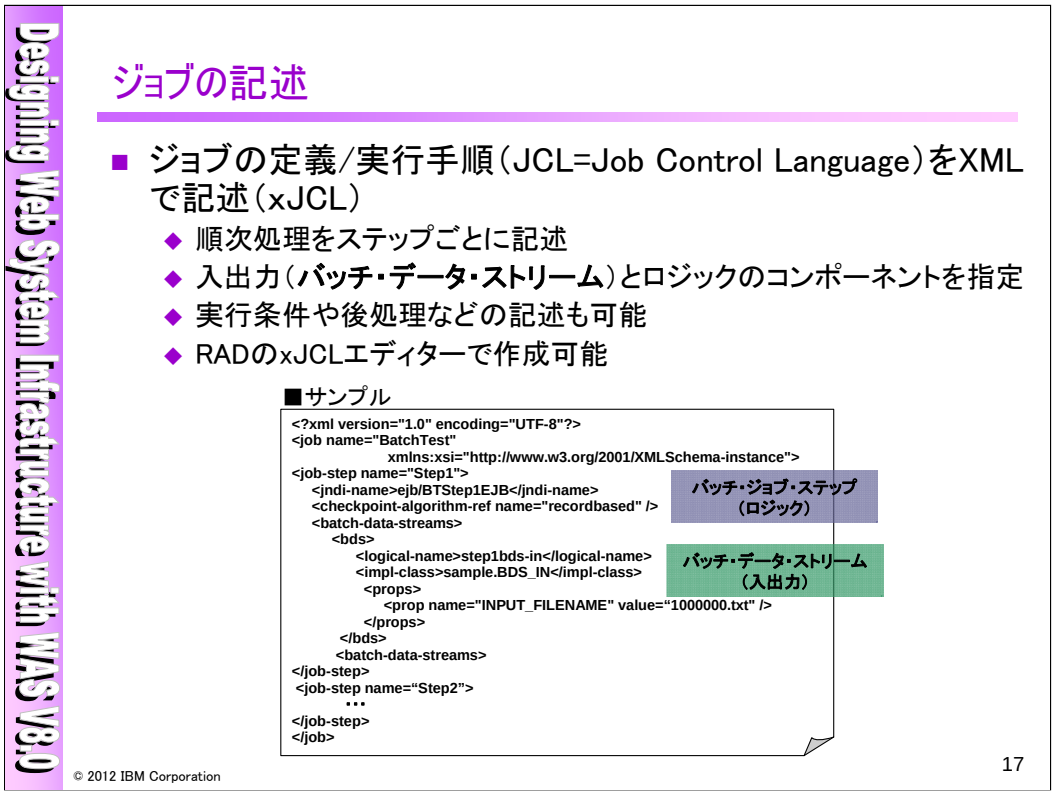
基本のコンポーネントは上記のようになります。リクエストの受付・ディスパッチングを行うジョブ・スケジューラー、ジョブ実行エンジンであるバッチ・コンテナ（エンドポイント）はそれぞれ単体サーバーもしくはクラスターに構成できます。典型的な構成はジョブ・スケジューラーが単体もしくは2台程度のクラスターで、グリッド・エンドポイントがクラスター構成となります。

ジョブはxJCLというファイルで定義し、ジョブ・スケジューラーではジョブ実行制御機能を提供しています。管理インターフェースからxJCLファイルと共にジョブを受付け、バッチ・コンテナに対してジョブを割り振ります。受け付けたジョブの情報や状態はジョブ・データベースに保存します。

WAS管理コンソールからジョブ・スケジューラーをホストさせるクラスター/単体サーバーを指定することで、該当するサーバーがジョブ・スケジューラーとして稼動します。

バッチ・コンテナではバッチ・アプリケーションの実行環境を提供します。トランザクション・バッチ型と数値計算型のアプリケーション稼動が可能ですが、通常のJavaEEアプリケーションも同一サーバー上で稼動させることができます。

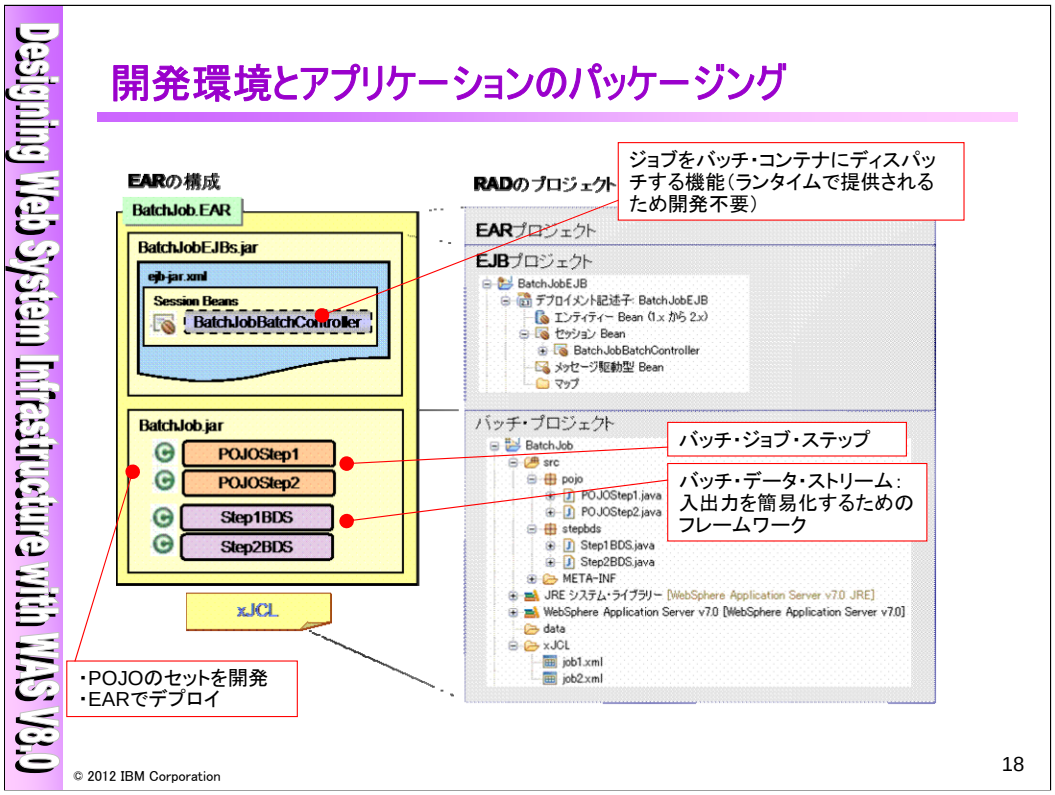
バッチ・コンテナで稼動するバッチ・アプリケーションはチェックポイント情報（ジョブの実行状況）をジョブ・データベースに保存します。



ホストをご存知の方であればなじみがあると思いますが、ホストでは実行ジョブの定義と実行手順を記述したものにJob Control Language（JCL）があります。
WAS では、ジョブの定義と実行手順はxJCLとよばれる XMLベースのJCLで記述します。

xJCLへは、ジョブの名前、JNDI名と実装クラス名、ジョブの実行順序、そのほかジョブの実行制御に関わるパラメーターとプロパティーを記述します。
以下のように、ジョブのサブミット、ジョブの実行環境へのディスパッチ、ジョブの実行は、すべてxJCLに書かれている定義に従って実施されることになります。
①クライアントは、ジョブ・スケジューラーに対してxJCLをもってジョブを実行依頼します。
②ジョブ・スケジューラーは、サブミットされたxJCLの定義に従い、該当のジョブを処理可能なバッチ・コンテナーを選択し、ディスパッチします。
③バッチ・コンテナーは、xJCLの定義に従い、サブミットされたジョブに該当するアプリケーションを実行します。

バッチ・アプリケーションの種類によりサポートされるxJCLのエレメント、すなわちXMLのSchemaは異なります。



WASのバッチ実行環境を使用するためには、バッチ・アプリケーションを開発する必要があります。

バッチ・アプリケーションは、EAR形式でのパッケージングとなり、一つのEARが一つのジョブに相当します。一つのジョブは単数または複数のジョブ・ステップから構成されますが、一つのジョブ・ステップが一つのバッチ・ジョブ・ステップ・クラスに相当します。

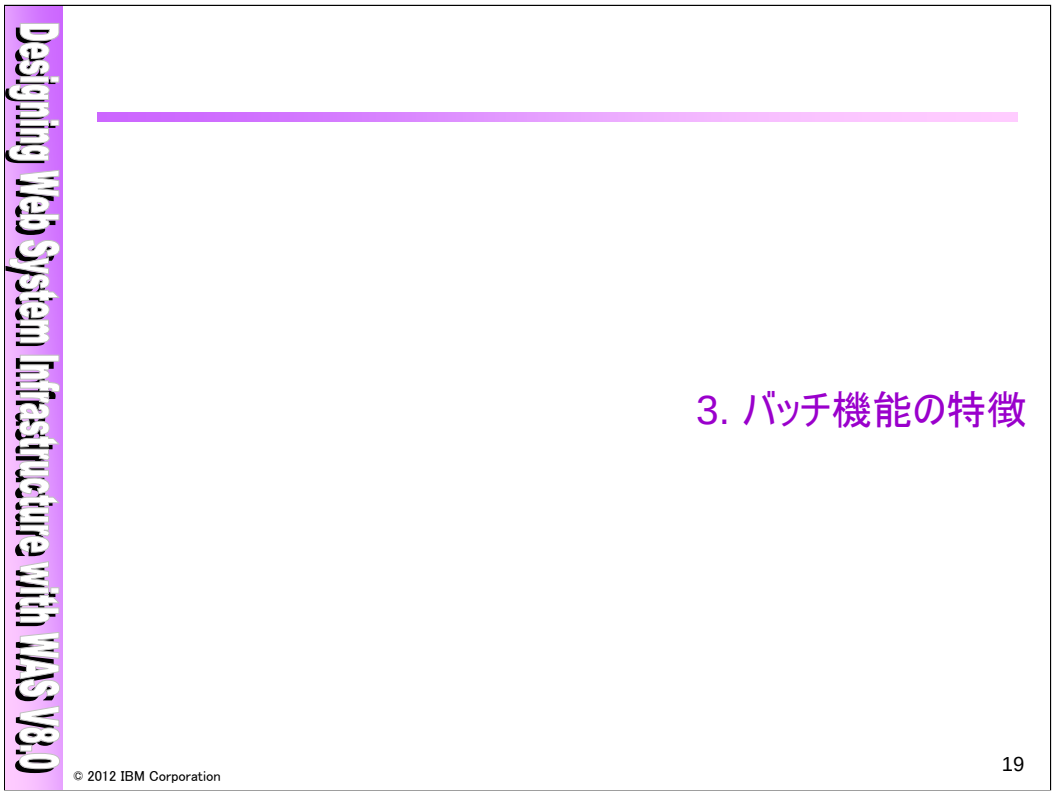
EARに含まれる主な要素は以下になります。

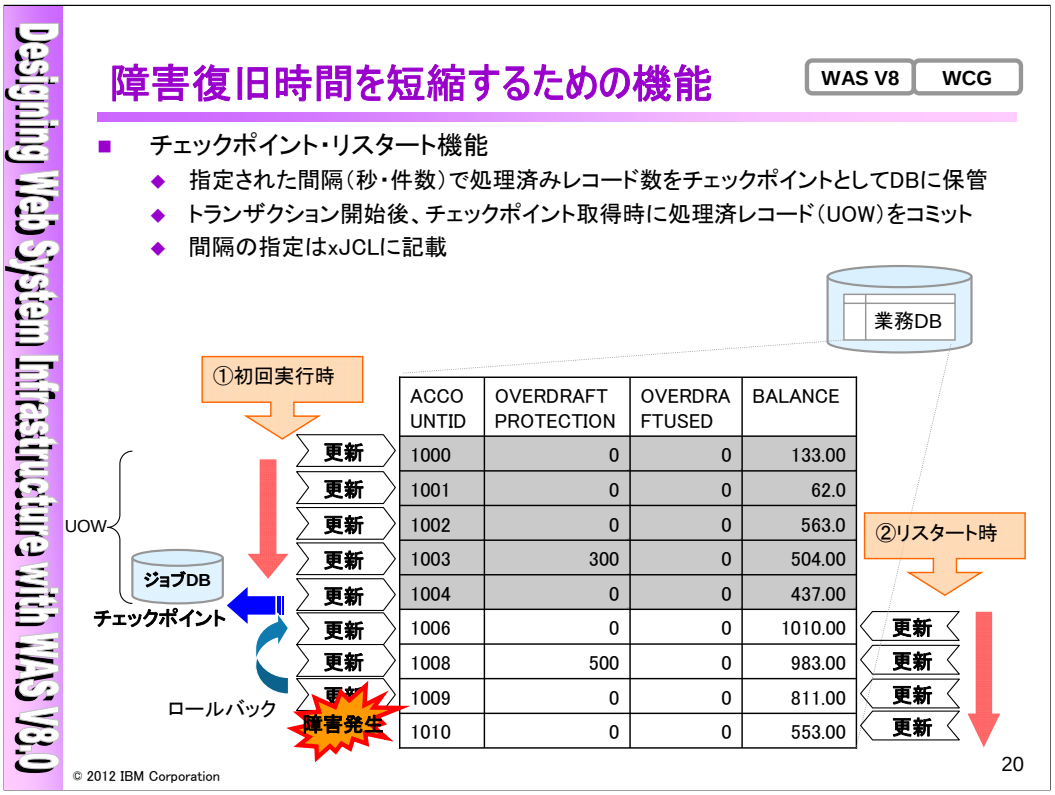
・バッチ・ジョブ・ステップ：実際のバッチのビジネス・ロジックをPOJOで実装するクラスです。ジョブ・ステップごとに生成します。

・BDS（バッチ・データ・ストリーム）：バッチ・ジョブ・ステップとファイルやDBといった入出力データとの間を仲介するコンポーネントです。BDSはフレームワーク化されており、入出力を簡単に制御することができます。

・バッチ・コントローラーBean：バッチ・ジョブ・ステップのコントローラーで、一つのEARに一つ定義されます。実態はWAS実行環境が提供するため開発の必要はありません。

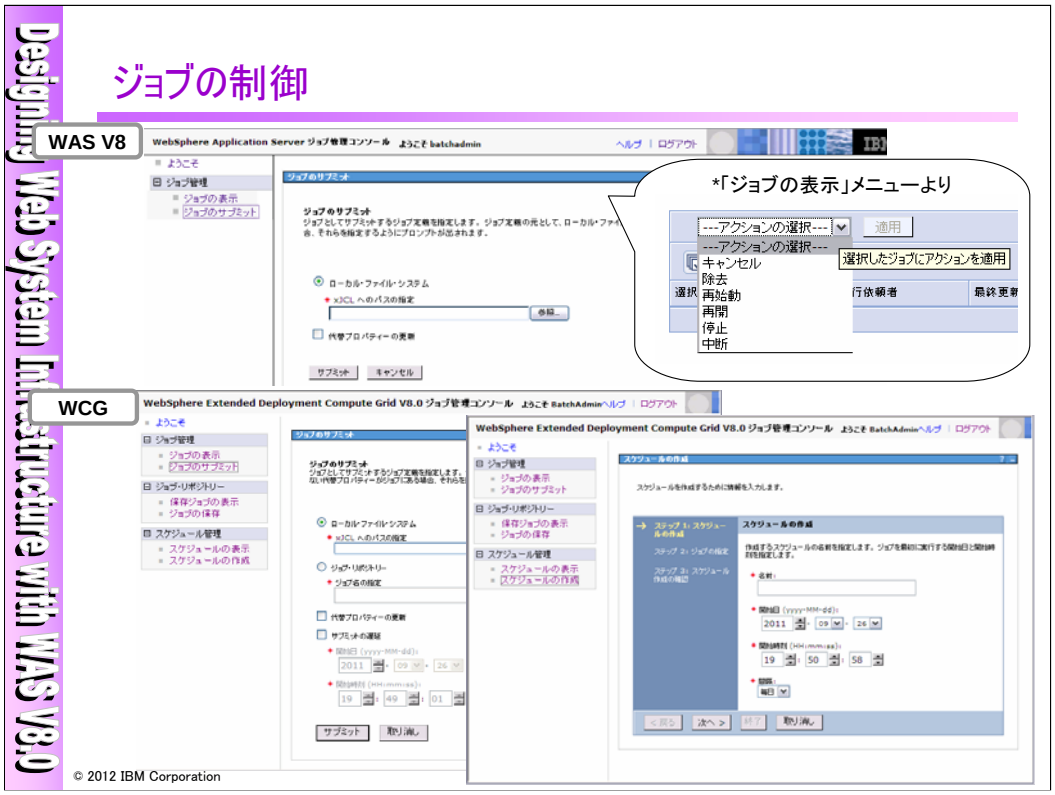
バッチ・アプリケーションには多くのコンポーネントが含まれますが、基本的な機能を使用する上で開発者がコーディングを必要とするのはバッチ・ジョブ・ステップのみで、多くのコードはxJCLファイルを元に開発ツールによって自動生成することができます。


Designing Web System Infrastructure with WAS V8.0
3. バッチ機能の特徴
© 2012 IBM Corporation
19



チェックポイント/リスタート機能について解説します。チェックポイント機能とは、バッチが障害などの理由で途中停止した場合に備えて指定タイミングで定期的にコミットを取り、その進捗を記録することで、回復時に再実行可能な状態にしておく機能です。リスタート機能はチェックポイントの情報を読み取り、再実行された際に最後にコミットしたレコードの次の行からの処理を行う機能です。

WASでは製品機能として提供されています。チェックポイントのコミット・タイミングの指定は、xJCL内部のcheckpoint-algorithmで指定します。指定可能なアルゴリズムは“recordbased”（処理行数）の他に、“timebased”（処理秒数）指定が可能です。ジョブ・ステップの単位で異なるチェックポイント・アルゴリズムを指定することも出来ます。しかし、チェックポイント指定の単位、すなわち最大UOWの単位はジョブではなくステップですので、三つのステップを内在するジョブでは最小でも三つのUOWに分かれることに注意してください。上記の例は、チェックポイントをrecordbasedで5行単位と指定し、障害が起きた場合の挙動を示しています。



WASで提供されている呼び出し用インターフェースでは、ジョブ実行以外にもジョブ・ステータスの確認などを行う管理インターフェースとなり、ジョブの制御ができます。

GUIツールとして、Job 管理コンソール（JMC）があり、コマンドライン・ツールとして、lrcmd があります。また、リモートからのEJBコールとWebサービス・コールが可能なAPIも提供されています。

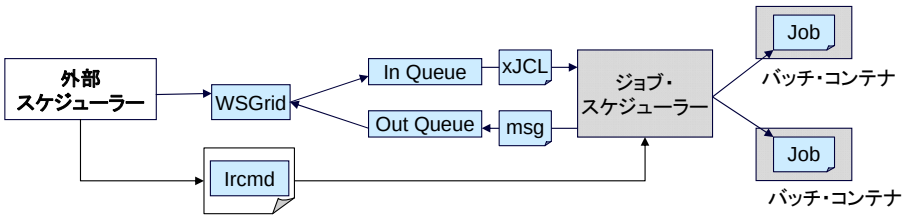
実行中（完了）のジョブの状態を表示し、「キャンセル/除去/再始動/再開/停止/中断」のアクションを施すことができます。

WCGでは、さらにジョブをスケジューリングすることも可能です。

Designing Web System Infrastructure with WAS V8.0

スケジューラーとの連携

WAS V8 | WCG



- **Ircmd（WAS V8）**
  - ジョブの実行依頼をしてもジョブ完了まで待機しない
    - スケジューラーとの連携には工夫が必要
  - コマンド実行毎にJVMが起動
- **WSGrid（WCG）**
  - ジョブ実行からジョブ完了まで待機する
  - スケジューラーからの直接起動、OSシェルからの起動、zOSのJCLからの起動、などが可能

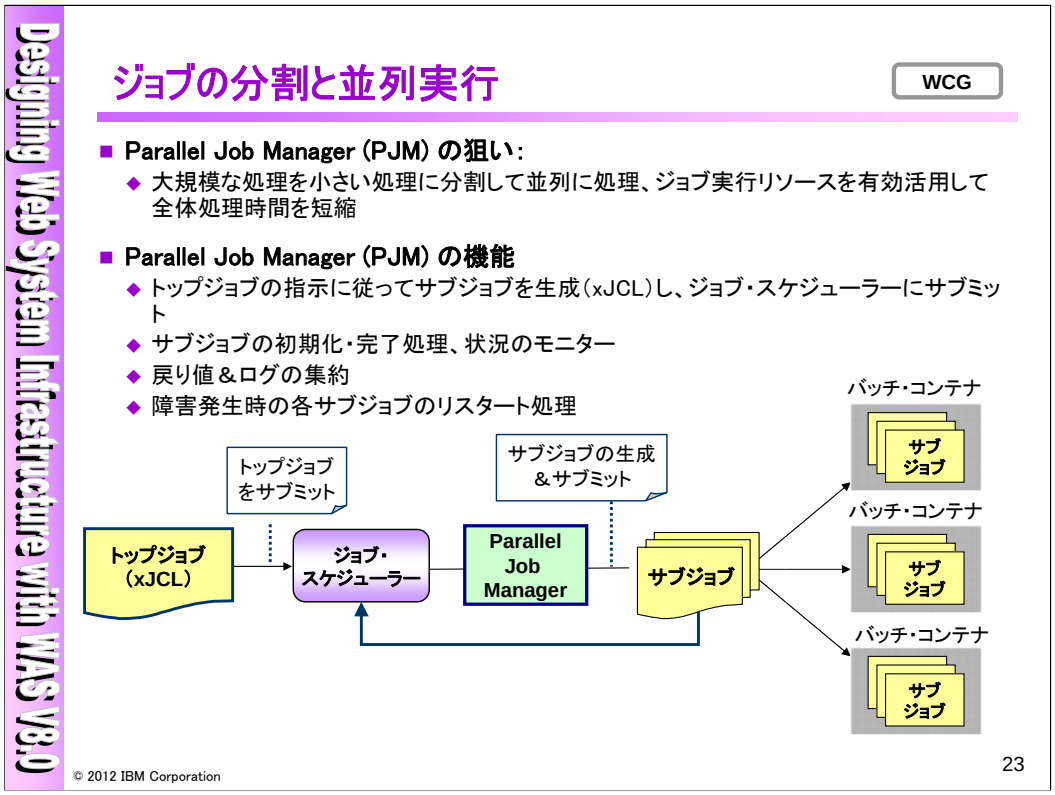
© 2012 IBM Corporation

22

WASのジョブ・スケジューラーは主にジョブ・ディスパッチャーが主な役割であり、通常のスケジューラーが持つ「様々な起動条件での稼動」や「稼動状況のモニタリング、異常時の通知」機能はありません。そこで、外部スケジューラー機能とどう連携するかを検討する必要があります。

WASではIrcmdというジョブの実行および管理を行うコマンド・ライン・ツールが提供されており、ジョブ管理に関わる一通りの操作が可能です。

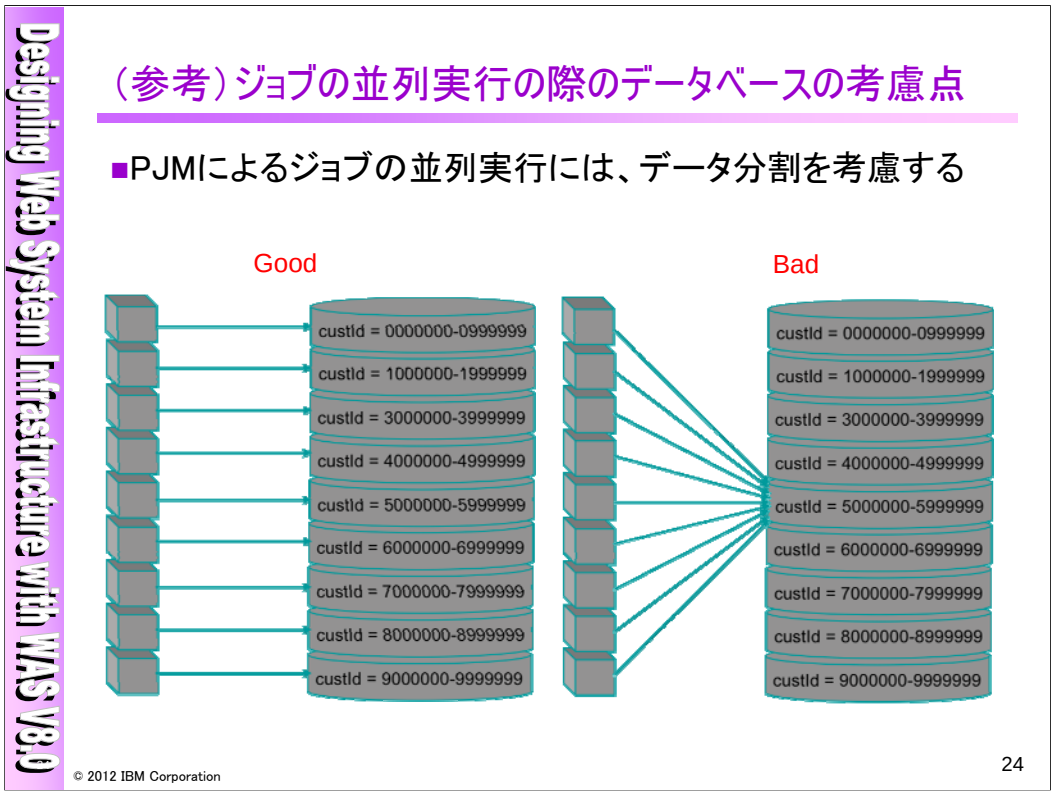
IrcmdとTWSの連携例については参考資料をご参照ください。



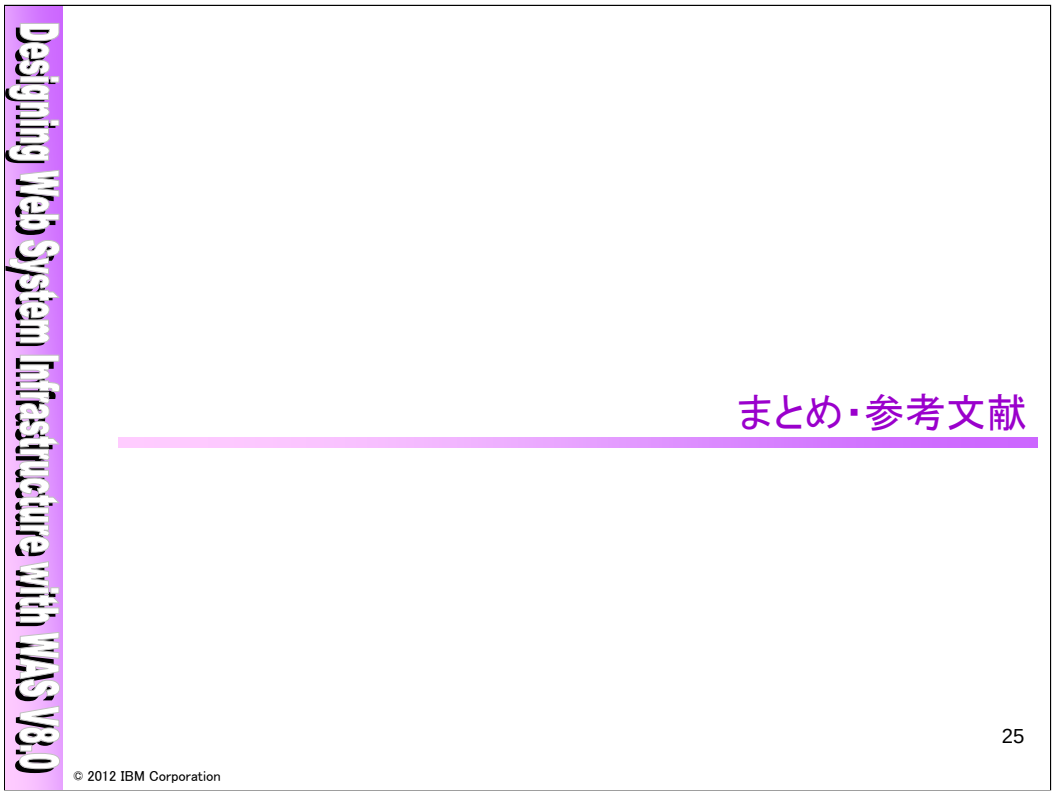
WCGでは、処理をサブジョブに分割して並列実行できる機能が提供されています。

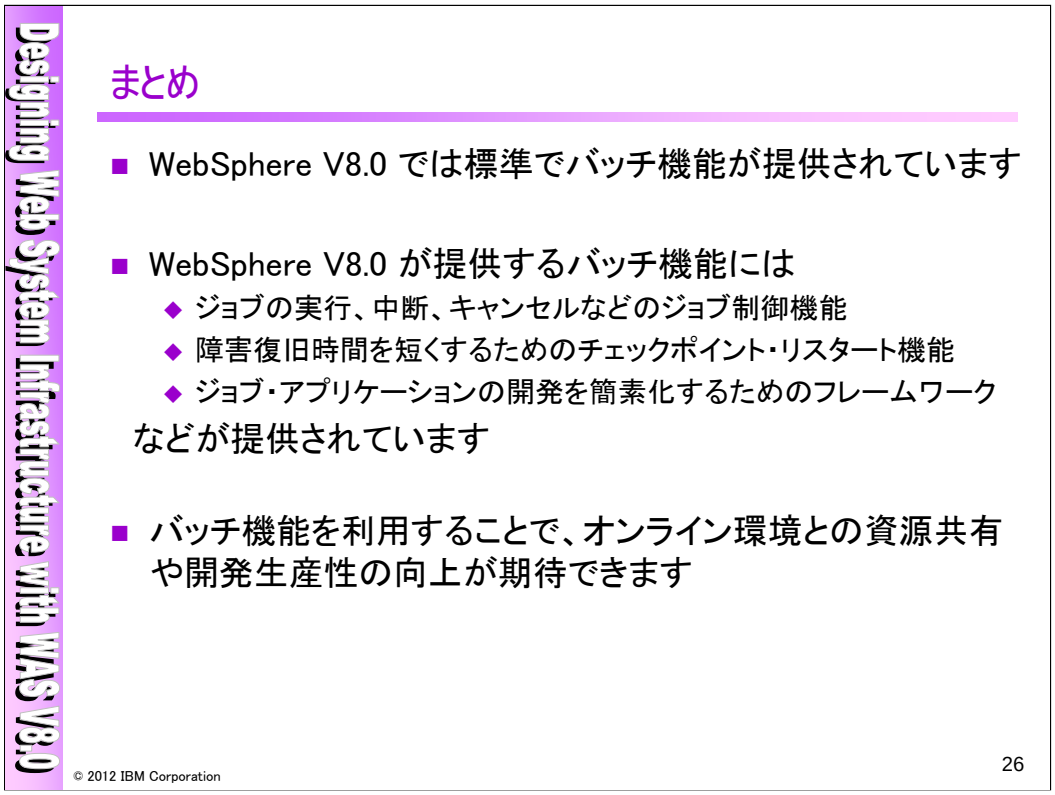
パラレル・ジョブ・マネージャーはジョブをサブ・ジョブに分割し、各バッチ・コンテナへディスパッチし、その結果を集約します。

サブ・ジョブは処理内容は同じで、扱う処理データが異なります。



パラレル・ジョブ・マネージャーの並列実行を考える場合は、サブ・ジョブが扱うデータも分割できることを考慮する必要があります。


Designing Web System Infrastructure with WAS V8.0
まとめ・参考文献
25
© 2012 IBM Corporation



WASではJavaバッチ実行環境のコアとなる部分を提供しています。
ユーザーはバッチ・プログラミング・モデルに基づいてPOJOでの開発が可能です。
さらに高度なバッチ機能を利用する場合はWCGの利用についてもご検討いただければと思います。

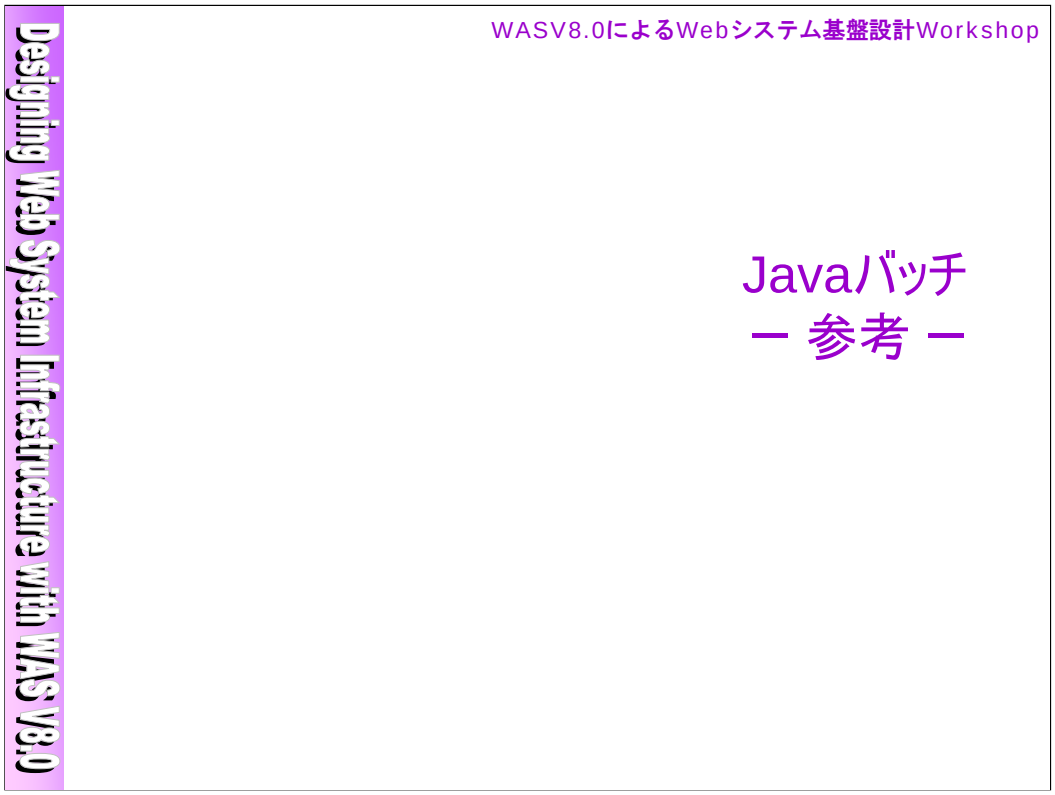
Designing Web System Infrastructure with WAS V8.0

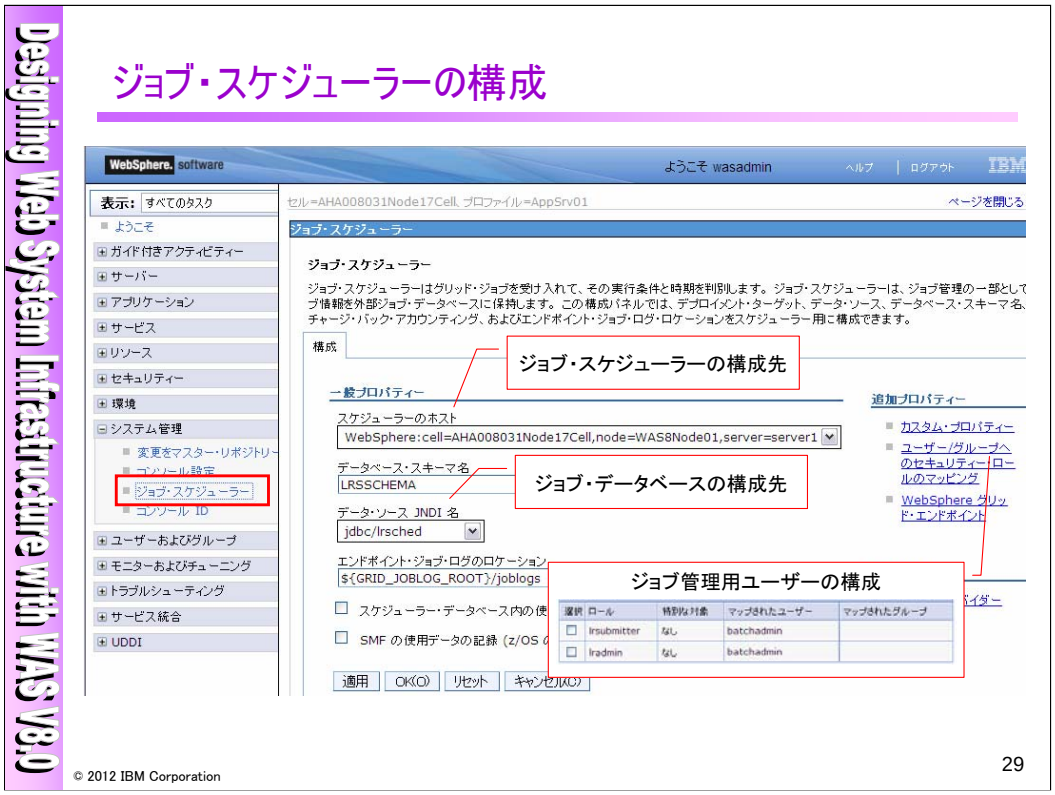
参考文献

- WAS V8 Information Center
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp
- ワークショップ資料
  - WebSphere Application Server V8 アナウンスメント・ワークショップ資料
    - http://www.ibm.com/developerworks/jp/websphere/library/was/was8_ws/
- Redpaper
  - Batch Processing with WebSphere Compute Grid: Delivering Business Value to the Enterprise
    - http://www.redbooks.ibm.com/abstracts/redp4566.html
- developerWorks
  - 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/
  - バッチ開発ガイド
    - https://www.ibm.com/developerworks/mydeveloperworks/wikis/home/wiki/WebSphere%20Application%20Server%20Wiki%20-%20Japan?lang=ja
  - 処理量急増や即時性を高めるJava&COBOL資産の統合バッチ環境
    - http://www.ibm.com/software/jp/websphere/events/impact2011/pdf/impact2011_e4.pdf

© 2012 IBM Corporation

27

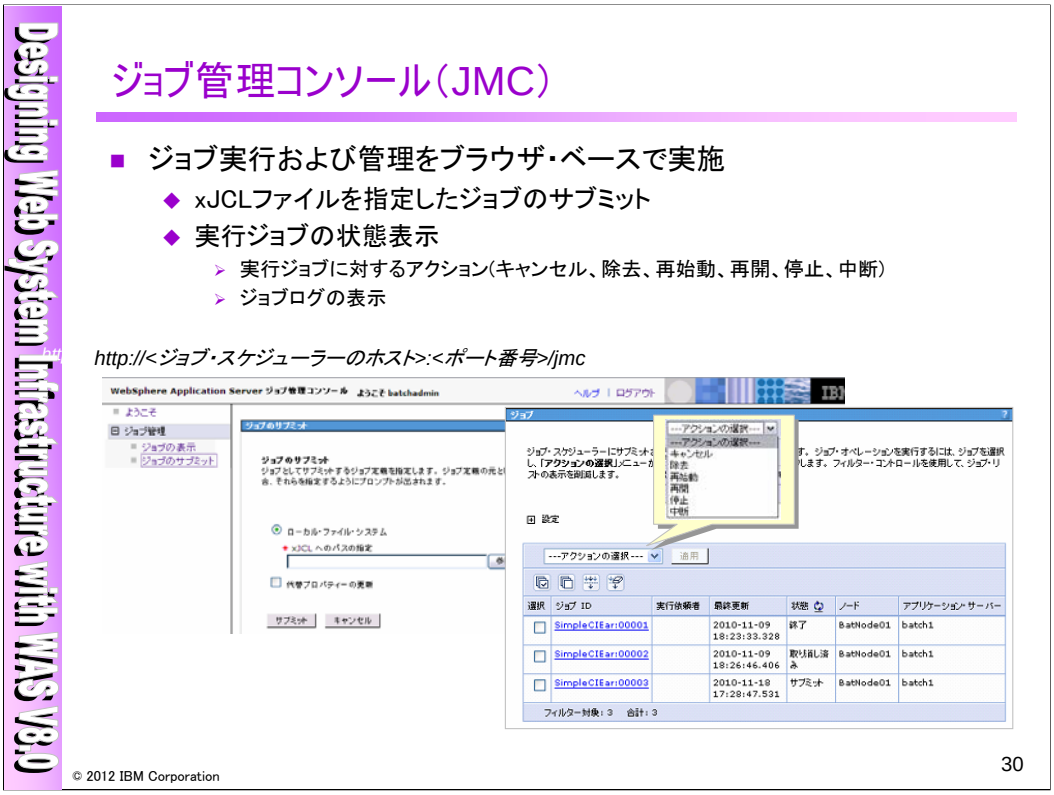

Designing Web System Infrastructure with WAS V8.0
WASV8.0によるWebシステム基盤設計Workshop
Javaバッチ
ー 参考 ー



ジョブ・スケジューラーは管理コンソールから設定します。

ジョブスケジューラーの構成先となるクラスター/単体サーバーの設定や、ジョブ・データベースのあて先となるJNDI名を指定します。

また、ジョブ・スケジューラーを稼動させるサーバーのアプリケーション・セキュリティーをOnにすれば、ジョブ管理用ロールを設定することができます。

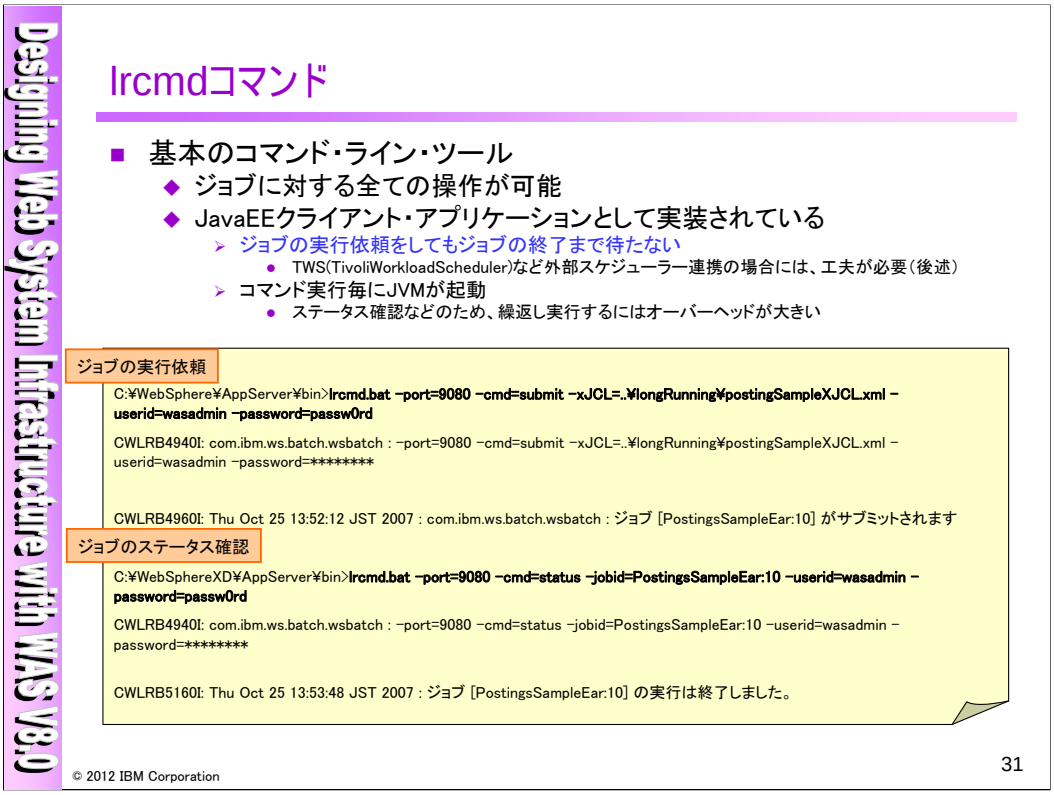


ジョブ管理コンソールでは、ジョブの実行及び管理をブラウザ・ベースで実施することができます。WASの管理コンソールとは別に提供されており、以下のURLにアクセスします。ジョブ管理コンソールは、ジョブ・スケジューラー上で稼動するアプリケーションなので、ホスト名/ポート番号にはジョブ・スケジューラーの値を指定します。

http://<*host_name*>:<*port#*>/jmc/console.jsp

ジョブ管理コンソールではあらかじめ用意したxJCLファイルと共にジョブのサブミットを実行することができます。xJCLファイルに変数が使用されている場合は、値入力も行います。

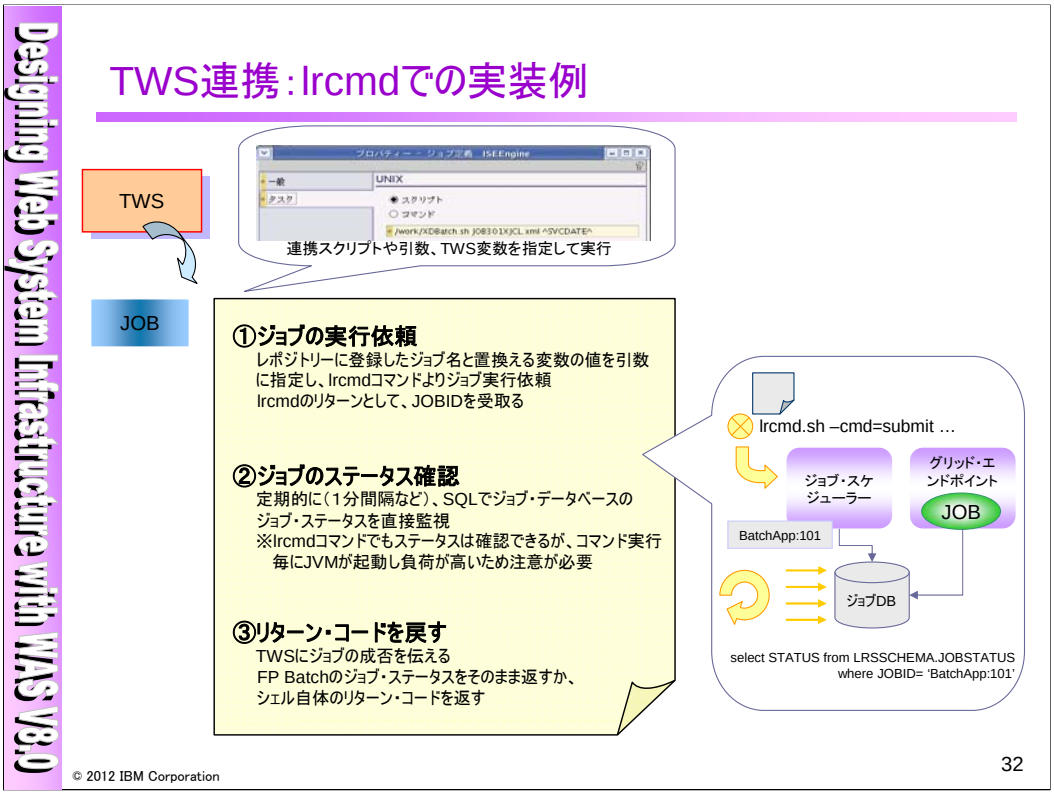
グリッド・エンドポイントで実行中（完了）のジョブの状態を表示し、「キャンセル/除去/再始動/再開/停止/中断」のアクションを施すことができます。



lrcmdはジョブの実行および管理を行うコマンド・ライン・ツールです。ジョブ管理に関わる一通りの操作が可能です。

lrcmdコマンドはジョブのサブミットを行っても、当該ジョブのサブミット処理が終了するとジョブIDを返して直ぐに抜けてしまい、ジョブの成否を待ちません。したがって、ジョブの成否は改めてlrcmdからステータス確認のコマンドを実行するか、ジョブ管理コンソールから確認する必要があります。

lrcmdはJavaEEクライアント・アプリケーションとして実装されており、コマンドの実行毎にJVMが起動/停止します。lrcmdのステータス確認コマンドをポーリングなどの用途に使用するには実行するにはオーバーヘッドが大きくなり、注意が必要です。コマンドの実行毎にJVMが起動しますので、外部スケジューラーにより同時に複数のlrcmdコマンドがキックされた際にも、複数のJVMが同時に起動し、マシン・リソースを浪費してしまうことがありますので注意が必要です(連携シェル内で一度に起動できるlrcmdコマンドの数を制限するなどが現実的な対応です)。



lrcmdでのTWS連携の実装例です。

変数が使えることでxJCLの生成処理は不要であり、xJCLは事前にレポジトリーに登録しておくことができます。

ジョブの実効依頼は、レポジトリーに登録したジョブ名と置換え変数値を引数に渡して行います。

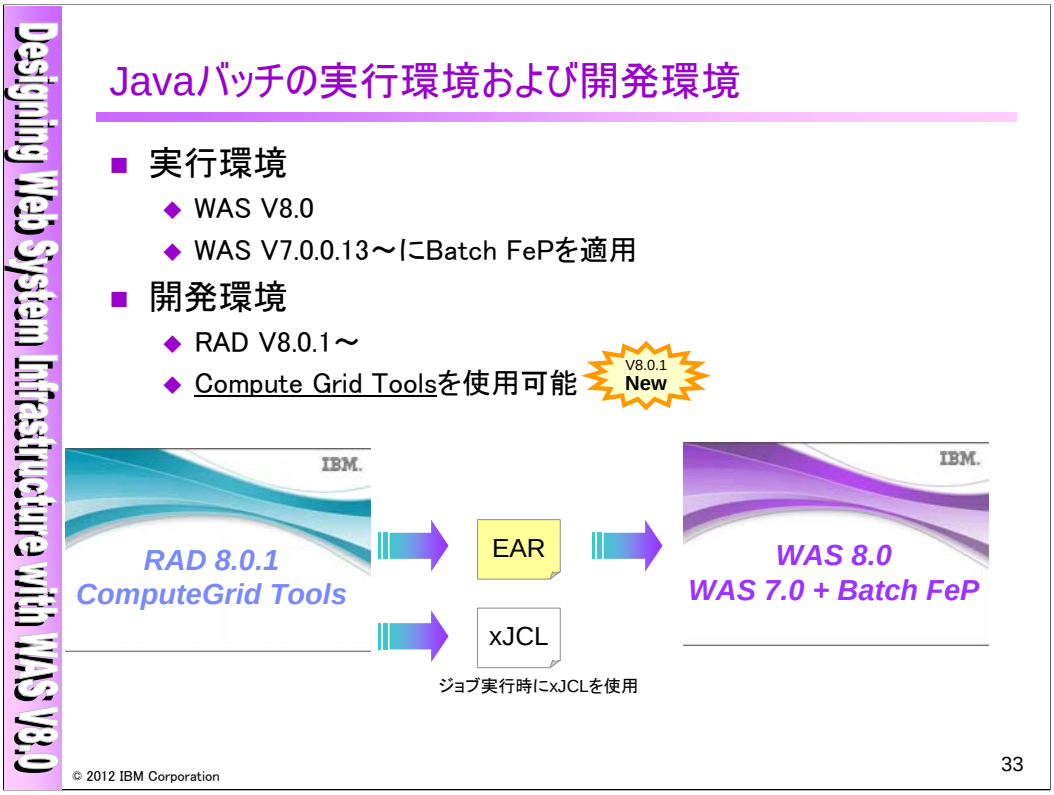
lrcmdコマンドを使用してTWS連携を実装する場合、TWSとバッチジョブの連携のためのスクリプトに必要な機能は以下のようになります。

（オプション）実行用xJCLの生成：xJCL内で変数が使用可能なので基本的には実行時にxJCLを生成するといった作業は不要です。変数を利用することができず、実行時に毎回xJCLを生成する必要がある場合のみの処理です。

①ジョブの実行依頼：ジョブ・スケジューラーに対して、ジョブの実行依頼を行います。

②ジョブのステータス監視、終了検知：TWSにバッチ・ジョブの実行結果を戻すために、ジョブのステータスを確認する必要があります。lrcmdコマンドはジョブの終了を待機しないため、連携スクリプトの中で独自にポーリングの実装を盛り込む必要があります。

③ジョブの成否をリターン・コードでTWSへ返す：最終的なジョブの実行結果をTWSにリターン・コードとして返す必要があります。

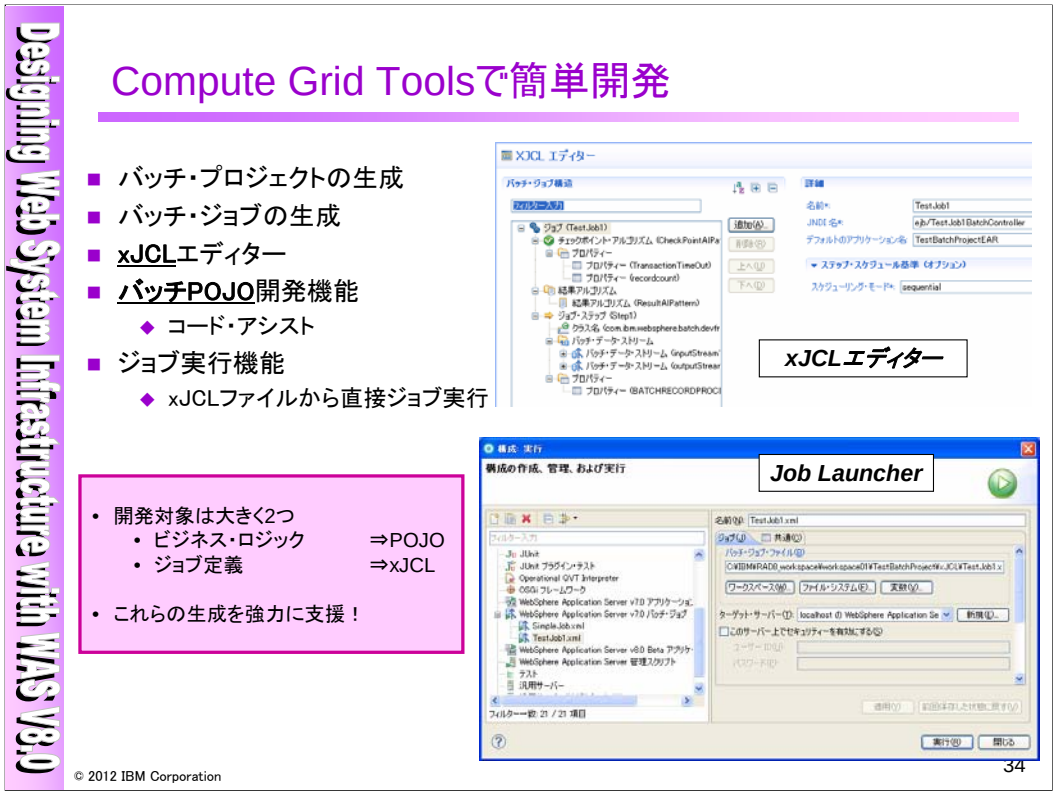


Javaバッチの実行環境および開発環境について説明します。

実行環境はWAS V8またはWAS V7.0.0.13以上にバッチ・フィーチャー・パックを適用します。WAS V7の場合はフィーチャー・パックの導入後、Javaバッチ機能が利用可能な専用のプロファイルを作成する必要があります。

開発環境はRAD V8.0.1からはバッチ開発用として新たにCompute Grid Toolsが提供されるようになりました。

RAD V8.0.1で開発したバッチ・アプリケーションはEAR形式でエクスポートし、WAS環境にデプロイするという流れになります。xJCLファイルは実行時に使用しますが、RAD V8.0.1で開発しエクスポートしておきます。



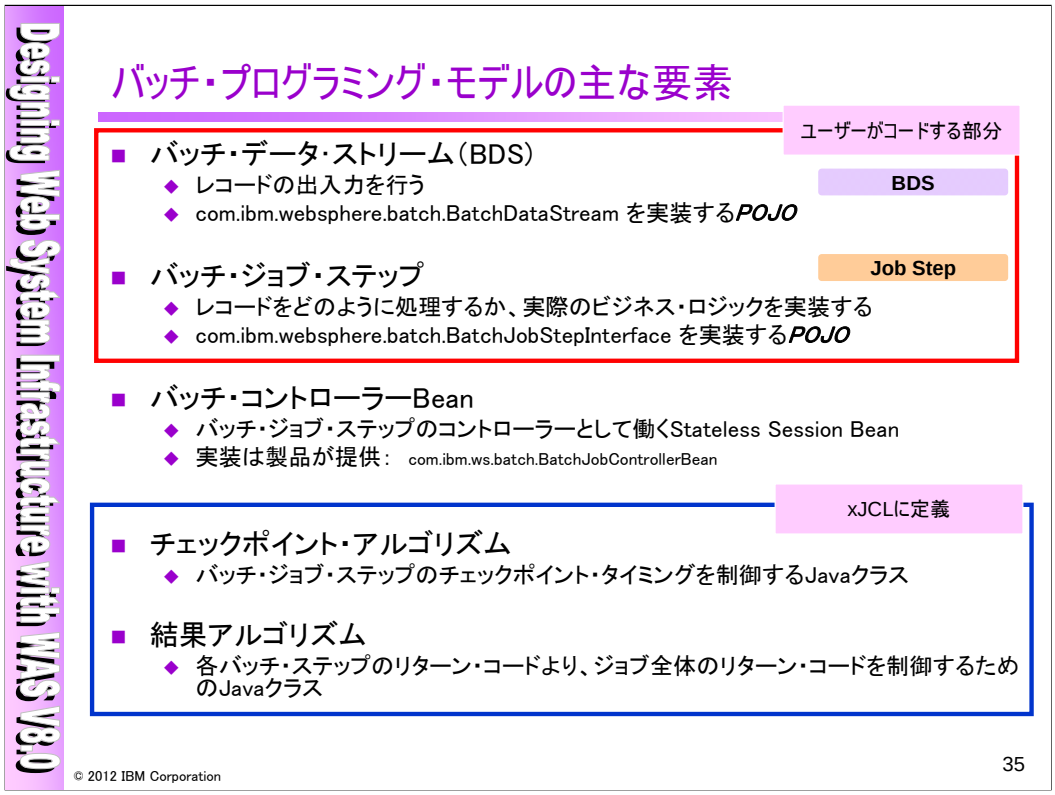
Compute Grid Toolsで提供されているバッチ・アプリケーション開発機能は以下になります。

・バッチ・プロジェクトの生成/バッチ・ジョブの生成：JavaEEアプリケーションでWebプロジェクト生成のウィザードが提供されているように、「バッチ・プロジェクト」および「バッチ・ジョブ」を生成するためのウィザードが提供されています。

・xJCLエディター：XMLベースのバッチ情報を含んだxJCLファイルを設定ベースで生成するエディターが提供されています。

・バッチ実行部分のPOJO開発を行う際のコード・アシスト機能を提供しています。

・ジョブ実行機能：ワークスペースでxJCLファイルから直接ジョブ実行を行うJob Launcherが提供されています。

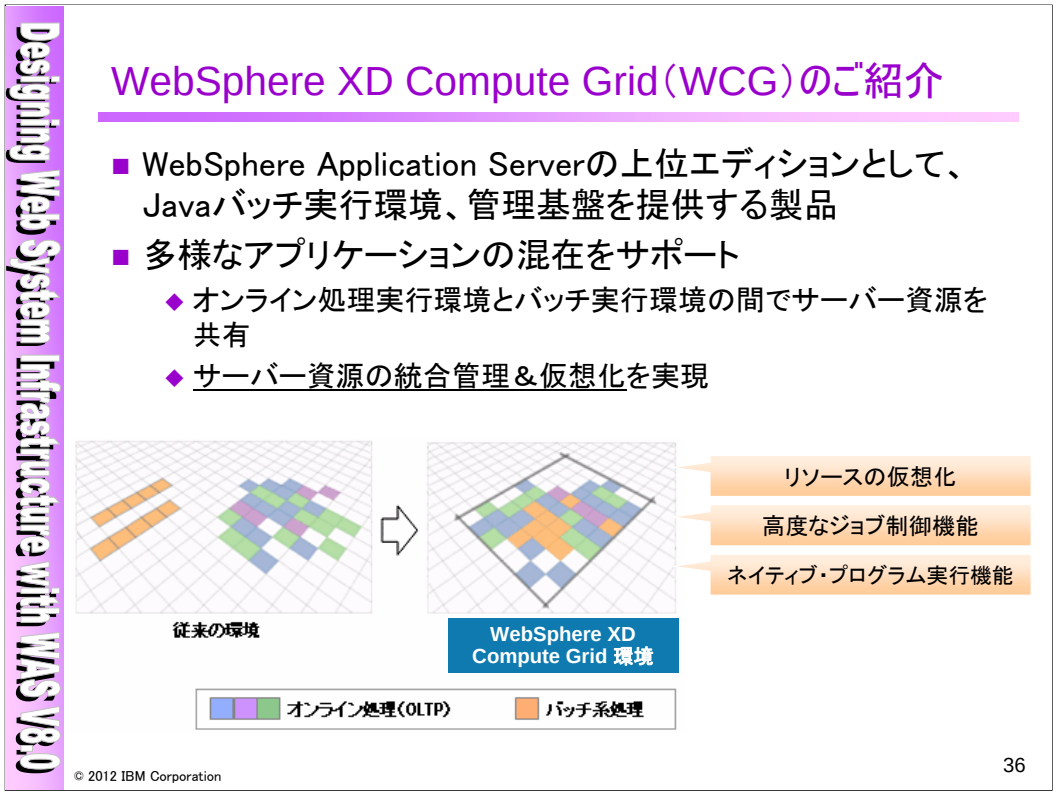


WASのJavaバッチ機能では、チェック・ポイント&リスタート機能やxJCLパラメーターの取得などのために、バッチ・コンテナとのインターフェースが定義されています。バッチ・プログラミング・モデルでは、このインターフェースを使うことで、開発者はバッチ基盤機能との連携を図りながら、さまざまなケースに対応できます。また、処理や入出力には多くの場合に適合する一定のパターンがあります。WASでは、このような典型的なパターンに適合する場合に、インターフェースの実装を省き、開発を省力化するためのフレームワークを提供しています。

バッチ・プログラミング・モデルで、ユーザーがコードを書かなくてはいけない部分は、ビジネス・ロジックの部分（ジョブ・ステップ）と、データ入出力（BDS）に関連する部分になります。それぞれフレームワークが提供するインターフェースを実装してコードを記述します。

その他、チェックポイントを取得するアルゴリズムであったり、リターン・コードを制御する部分はウィザードで設定可能です。

チェックポイント取得は特に、件数ベース、時間ベースのどちらかの方式を選択することになります。



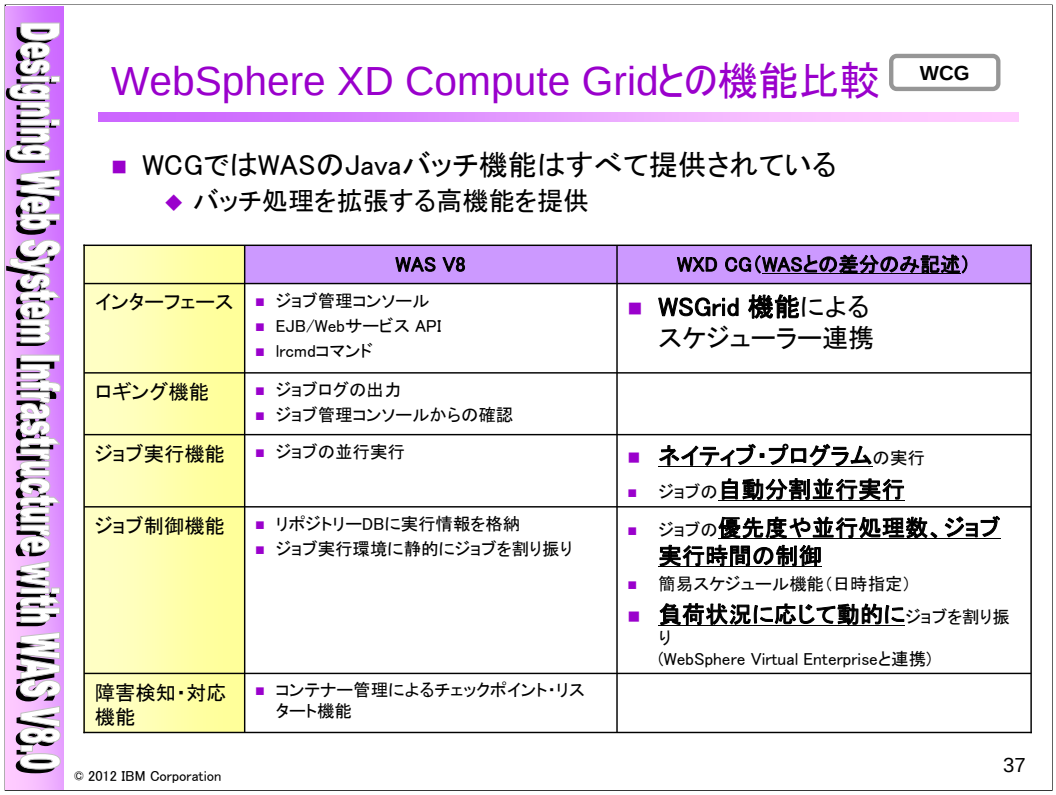
WebSphere XD ComputeGridについて説明します。

WASのJavaバッチ機能はWCGのサブセットにあたります。

WCGはWASの上位エディションの位置づけにあり、Javaバッチ実行環境、管理基盤を提供する製品です。

WASのJavaバッチ機能と同様にオンライン処理実行環境とバッチ実行環境の間でサーバー資源の共有を実現します。

WCGではさらに、リソースの仮想化、高度なジョブ制御機能、ネイティブ・プログラム実行機能を提供しています。



WCGではWASが提供するJavaバッチ機能はすべて提供されています。プログラミング・モデルなども一緒です。

バッチ処理を更に拡張する高機能を提供していますが、その機能比較を表一覧に示します。

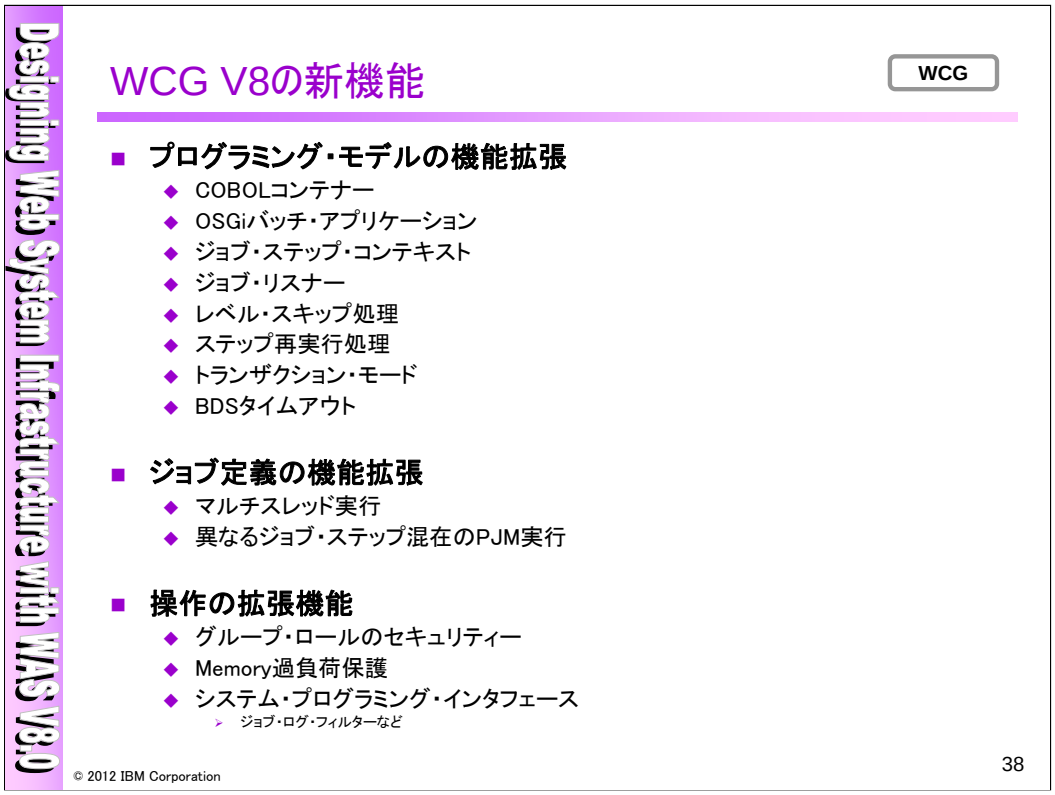
スケジューラーと連携する場合、WASのlrcmdコマンドは、考慮が必要である旨説明しましたが、WSGridというツール、インターフェースを用いることで作りこみなしで連携することが可能です。

また、ネイティブ・プログラムの実行もサポートしています。これはたとえば帳票印刷の処理やファイル送受信処理などComputeGridの外で動く処理を実行出来ると言う事です。

WVEと連携することで、サーバーの負荷状況に応じて、動的にジョブを割り振りしてくれるという強力な機能もあります。

優先度の高いジョブを、負荷の少ないサーバーで優先的に処理する、その際にはオンラインとの割り振り状況も加味するといったことが可能になります。

いわば、オンラインもバッチも、1つのサーバープールとして扱い、プールの中でのリソース配分は、リクエスト、ジョブの優先度に応じて自動的に最適化されるという、アプリケーションの仮想化を実現できることになります。



その他にWCG V8では様々な新機能が提供されました。詳細はWCGのInfoCenterをご参照ください。

Designing Web System Infrastructure with WAS V8.0

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律的またはその他の指導や助言を意図したものではなく、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だすことを意図したものでも、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もなく、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いても、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と述べる、または暗示することを意図したものでも、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、COBOL、Rational、WebSphere、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。

© 2012 IBM Corporation

39