WebSphere Application Server Version 8.0へのマイグレーションガイド

2012/03/27 Version 4.6

日本IBM株式会社
ソフトウェア事業
WebSphere Client Technical Professionals

WebSphere software

© 2011 IBM Corporation

おことわり

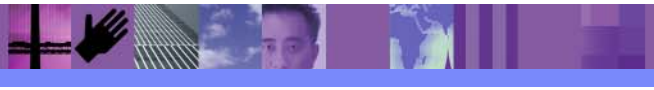
- このマイグレーションガイドは，WebSphere Application Server（以下WAS）の
  マイグレーションを行うさいの手順の全体を俯瞰し，
  その概要を把握していただくことを目的として作成されています
- この資料は以下の内容を含んでおります
  - マイグレーションの全体的な流れ
  - 多くのお客様で必要な一般的な作業内容
  - 過去のマイグレーションプロジェクトでの経験事例
- この資料はIBMの正式なレビューを受けていません
  - 内容については正確を期すように勤めておりますが，本資料を参照することにより生じた
    不利益，損害等については一切責任を負いかねますので，あらかじめご了承下さい
- このガイドは，マイグレーションにあたって問題となる事象の完全なリストではありません
  - 実際のマイグレーション作業においては，
    この資料に記述のない問題が発生する可能性もあります
- 実際の作業の手順などについては記述しておりません
  - 実際にマイグレーションプロジェクトを行うにあたっては，
    末章のリファレンスにありますInfoCenterやマイグレーションのRedBooks，
    各種テクニカルノートなどをご参照いただくようにお願いいたします

Agenda

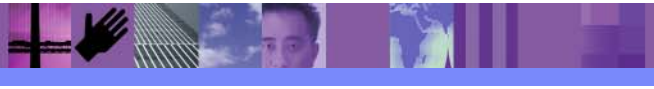
- WebSphere Application Serverの進化
- マイグレーションのロードマップ
- アプリケーションのマイグレーション
- 実行環境のマイグレーション
- 参照リンク・リファレンス

改変略歴

- Version 1.x
  - WebSphere 6.1ご紹介セミナー用の説明資料として作成
- Version 2.x
  - 2007/10/25　Version 2.0　　最初の版
- Version 3.x
  - 2008/11/10　Version 3.1　　WAS 7.0へのマイグレーションガイドに改題
- Version 4.0
  - 2011/6/18　Version 4.0　　WAS 8.0へのマイグレーションガイドに改題
  - 2011/6/28　Version 4.1　　細部を修正
  - 2011/8/05　Version 4.2　　Edge Componentの情報を追加・細部を修正
  - 2011/10/14　Version 4.3　　大きなEAR/WARのデプロイについてを追加
  - 2011/11/29　Version 4.4　　各種情報を追加・更新
  - 2012/02/23　Version 4.5　　「ハードニングによる影響」ほかを追加・更新
  - 2012/03/27　Version 4.6　　EJB 3.0の互換性，仕様外の動作（4）を追加

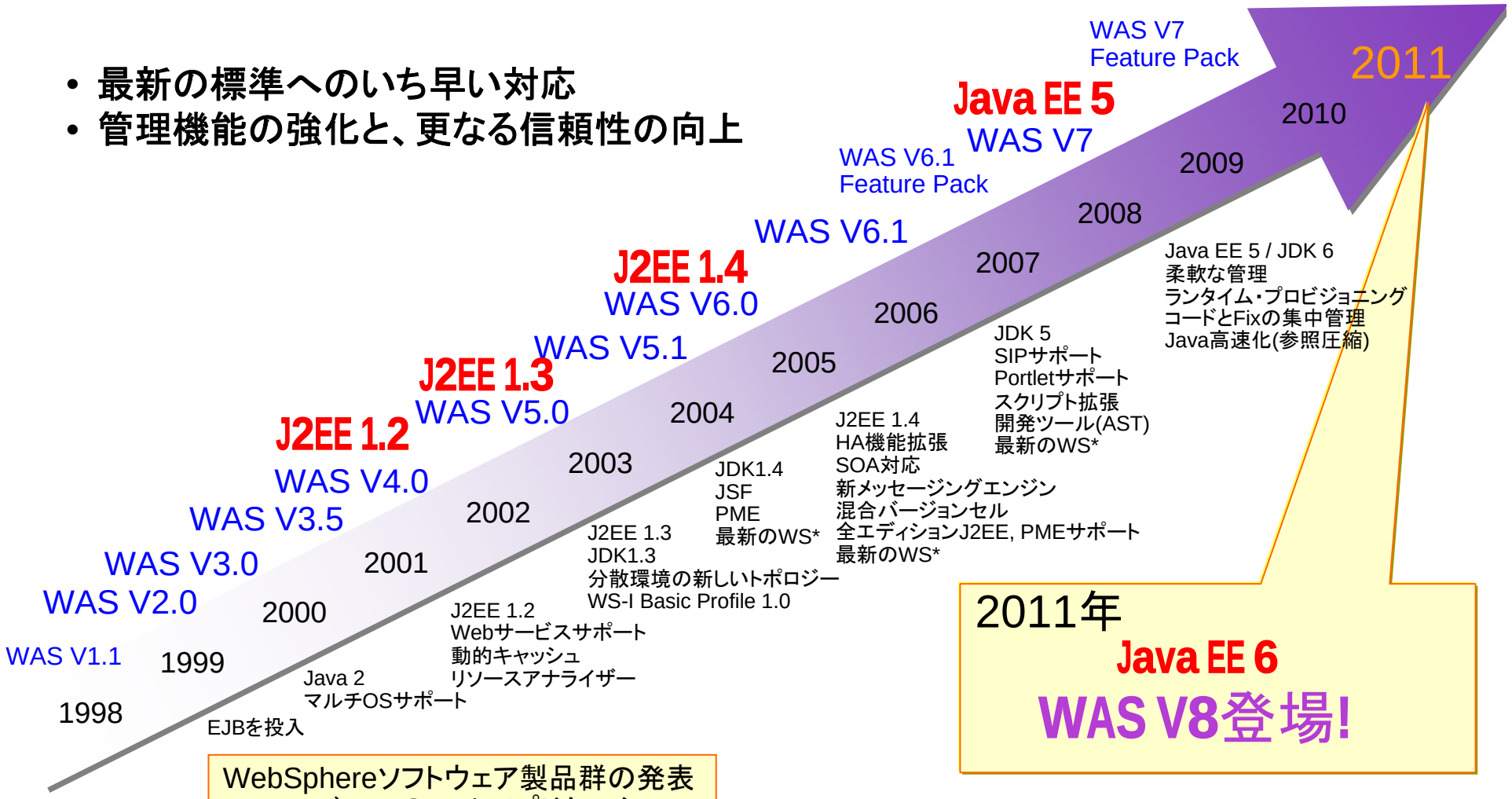


WebSphere Application Serverの進化



WebSphere Application Serverの歴史

- 最新の標準へのいち早い対応
- 管理機能の強化と、更なる信頼性の向上



WebSphereソフトウェア製品群の発表
JavaベースのWebアプリケーション
の開発・展開・管理

WAS V4.0の新機能（抜粋）

- J2EE 1.2（最初のJ2EE）に完全対応
  - Servlet API 2.2 / JSP 1.1 / EJB 1.1など
- J2EE標準のアプリケーション開発スタイルを採用
  - 「アプリ開発者」と「サーバー管理者」のロール（役割）を定義
  - EAR/EJB Jar/WARによるアプリケーション構築
  - IDE（統合開発環境）の使用を前提とした仕様群
- Webサービス機能に対応
- 各種パフォーマンス機能
  - DynaCacheによるコンテンツの再利用
  - パフォーマンス・モニタリング機能のフレームワークの実装（PMI）
  - GCのマルチスレッド化
- Eclipseベースの統合開発ツールを同時リリース

WAS V5.0の新機能（抜粋）

- J2EE 1.3への完全対応
  - 大幅に機能強化されたServlet API 2.3 / JSP 1.2
    国際化対応、サーブレットフィルター、イベントリスナー、等
  - 完成された仕様となったEJB 2.0
    CMRのサポート、Message Driven Beanの採用、等
- 管理概念の大幅変更
  - XMLファイルによる構成情報保持＋ブラウザベースの管理
    （V4.0まではRDBMSでのリポジトリDB＋Javaアプリケーションの管理）
  - J2EE仕様のJMXを使用したコンポーネント間連携
  - 稼動時のコンポーネント間の依存関係の解消
- JMS（メッセージング機能）の統合
- Edge Componentの統合

WAS V5.1の新機能（抜粋）

- J2SE 1.4への対応
- 要求メトリック
- Web Service機能を中心とした大幅なパフォーマンス向上

WAS V6.0の新機能（抜粋）

- J2EE 1.4への完全対応
  - Servlet API 2.4 / JSP 2.0 / EJB 2.1など
  - Webサービスの標準化（JAX-RPC/JAXR）
  - JSF、SDOなどの新技術も先行して対応
  - J2EEを強化するPME（Programming Model Extension）を標準サポート
- アプリケーション管理機能の強化
  - Fine Grained Update / Rollout Update
- HA（High Availability）機能の強化による「止まらないサービス」の実現
- Javaベースの新メッセージングエンジンを搭載
- SOAへの対応
  - SOA（サービス指向アーキテクチャー）に基づくESB（エンタープライズ・サービス・バス）機能を提供

WAS V6.1の新機能（抜粋）

- J2SE 5.0 への対応
  - 開発生産性の向上、JDK 5.0のパフォーマンスの向上
- 最新のAPIへの対応、
  - JSR116 SIPサーブレット、JSR168 ポートレット、 最新Webサービス仕様
- システム管理機能の拡張
  - コマンド支援、コンソールのカストマイズ、導入・構築の容易性、メッセージング拡張
- セキュリティー
  - 設定の簡略化、柔軟なセキュリティーの設定が可能に
- 使いやすさの向上
  - 新しいツールの提供：IBM Support Assistant (ISA) / 診断プロバイダー
- Application Server Toolkit (AST)
  - パッケージングだけでなく、開発機能を提供
- Installation Factory
  - WASの導入・構築を簡略化

WAS V7.0の新機能（抜粋）

- Java EE 5をはじめとした最新仕様への対応
  - EoD：単純なことを簡単に
  - しかし，複雑なことも実現可能
- セキュリティ
  - マルチ・セキュリティ・ドメイン，Kerberos 認証とシングル・サインオンのサポート
  - システム監査機能
- 高度な管理機能
  - 大規模な分散環境で管理のオーバーヘッドを軽減するフレキシブル・マネジメント
  - 複数EAR/WAR/ライブラリで構成されるアプリケーションを統合管理するBLA
  - 集中インストール・マネージャー
  - プロパティ・ファイルベース構成ツール
- パフォーマンスの向上
  - IBM Java SE 6 による共有クラス・キャッシュ機能の向上
  - 64bit WASにて、32bit WASと同等のメモリー使用量とパフォーマンスを実現
  - ランタイム・プロビジョニング，ランタイム・コンポーネントの非同期スタート
  - Webサービスの大幅なパフォーマンス向上

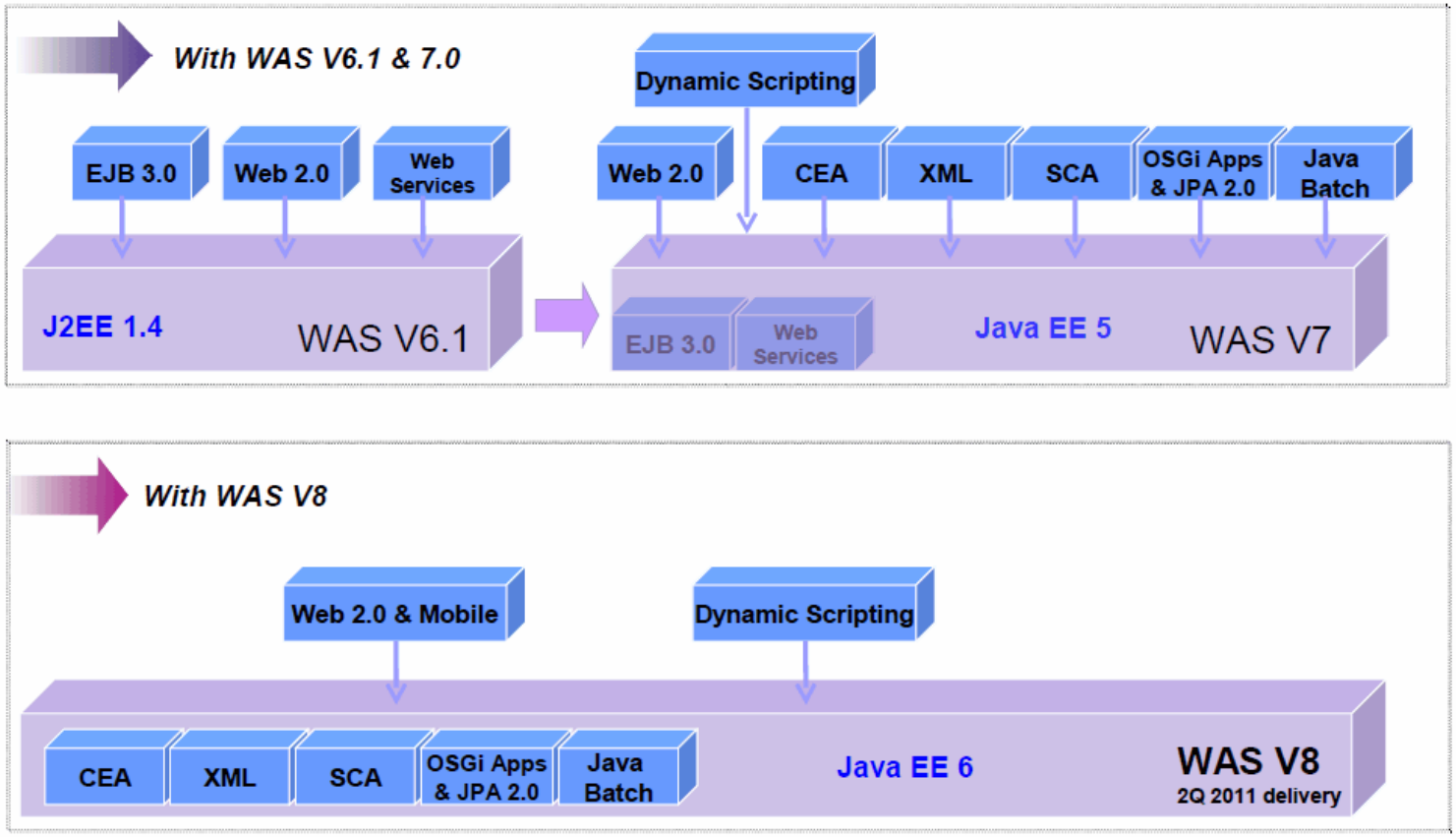
WAS V8.0の新機能（抜粋）

- 多くの最新のプログラミングモデルに対応
  - Java EE 6（Servlet 3.0，JSF 2.0，JAX-RS，CDI，JPA 2.0）
  - OSGi（Enterprise OSGiのサポート/バンドルの動的更新）
  - Javaバッチ
  - SCA（Service Component Architecture）
  - Web 2.0 & Mobile（Dojo Toolkit 1.6/Dojo Mobile）
- パフォーマンスの向上
- 管理機能・セキュリティの強化・改善

最新のテクノロジーを提供する新しい方法　Feature Pack

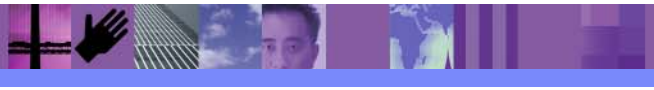
- WAS V6.1よりFeature Packにて**新機能**を提供。
- WASのライセンスをお持ちのお客様に**無償**で提供
- Passport Advantageによる**正式サポート**対象

新バージョンのリリースを待たずに
最新のテクノロジーを
入手可能！



問題判別機能の強化

- JVM（Java Virtual Machine）の問題判別機能の改善
  - Javacoreによる障害時の情報取得
  - Heap Dumpによるメモリ中のオブジェクトの情報の取得
  - 各種トレース機能による問題判別
- WASのモニタリング・問題判別機能
  - ハングスレッドの検出（WAS 5.1.1以降）／メモリリークの検出（WAS 6.0.2以降）
  - PMIによる，各種リソースの使用状況・パフォーマンス情報の取得（WAS 4.0以降）
  - 要求メトリックによるServlet/EJB/JDBCの応答時間の記録（WAS 5.1以降）
  - Collectorによる資料の自動収集（WAS 5.0以降）
  - HPEL（High Performance Extensible Logging）によるログ・トレースの高速化（WAS 8.0以降）
- ISA（IBM Support Assistant）の提供（WAS 6.1以降）
  - 様々なサイトを対象とした串刺し検索
  - 問題判別のための各種ツールの実行
  - 問題報告のための資料収集をサポート



WASサポート・ポリシーの変更：5年＋3年にサポート延長

通常の3年＋有償2年のサポート・ポリシーを、WAS V6から5年＋3年に延長

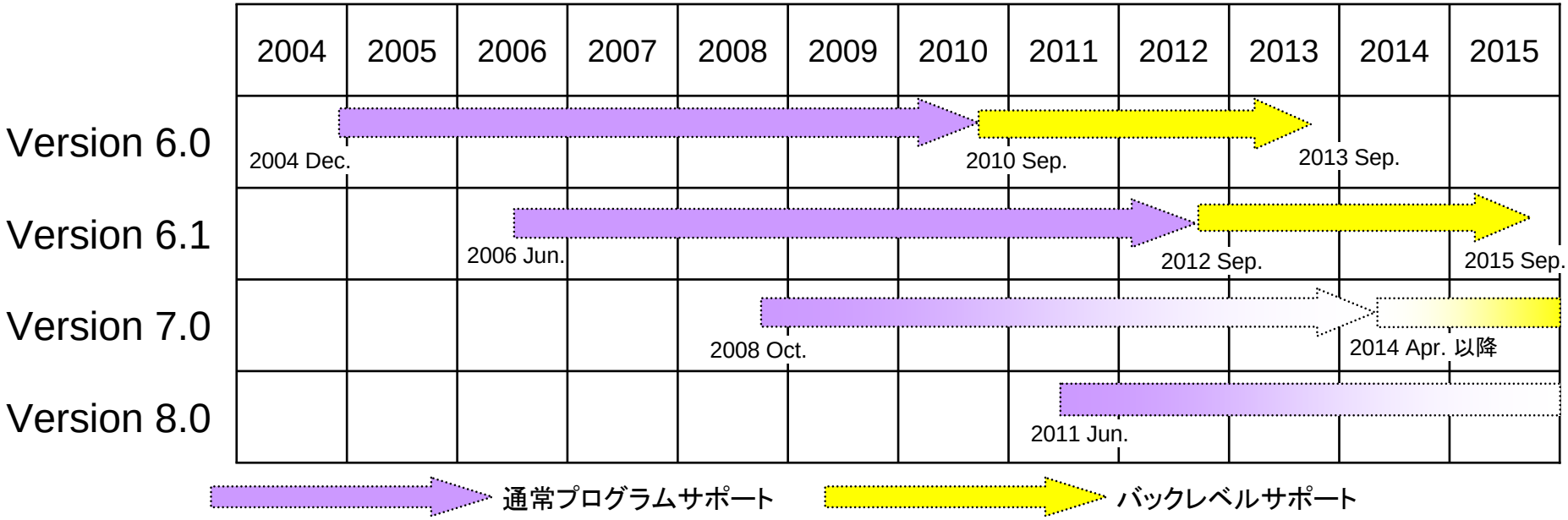
- 対象製品：
  WebSphere Application Server
- 対象OS：
  AIX, Windows, Linux, Linux pSeries, Linux zSeries,
  OS/390, OS/400, i5/OS, z/OS
  （その他のOSについては現在Platform提供ベンダーと交渉中）
- 対象バージョン：
  V6.0, V6.1, V7.0, V8.0
- 対象エディション：
  Base, Developer, Express, Network Deployment

- サポート期間が長くなることによるメリット
  - 開発、テスト後の運用期間を同じバージョンで長く保つことができる
  - 次バージョン/リリースへのアップグレードの準備を余裕を持って進められるように

詳細についてはこちらの文書を参照
「WebSphere Application Server サポート・ポリシーの変更：“5年＋3年”サポート・ポリシーの新たな発表 」
- http://www.ibm.com/jp/domino01/mkt/websphere.nsf/doc/000A26AE

WASのサポートポリシー

- 「Enhanced IBM Support Lifecycle Policy」にもとづいたサポートを提供
  - 製品出荷から最低5年後までのプログラムサポート、
    最低8年後までのバックレベル・プログラム支援サービスが提供されます。
  - 詳細は以下のURLでご確認下さい
    http://www.ibm.com/software/support/lifecycle/lc-policy.html
- サポート終了日は，遅くともサポート終了1年以上前に発表レターにて通知します
  - 発表済みの製品については以下のページで確認できます
    http://www.ibm.com/software/support/lifecycle/

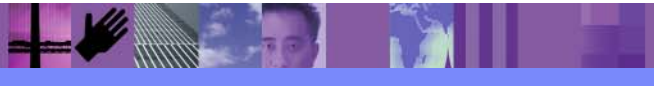


【参照】V6.0～V8.0での新機能

- V6.0での新機能
  - InfoCenter “このリリースの新機能”

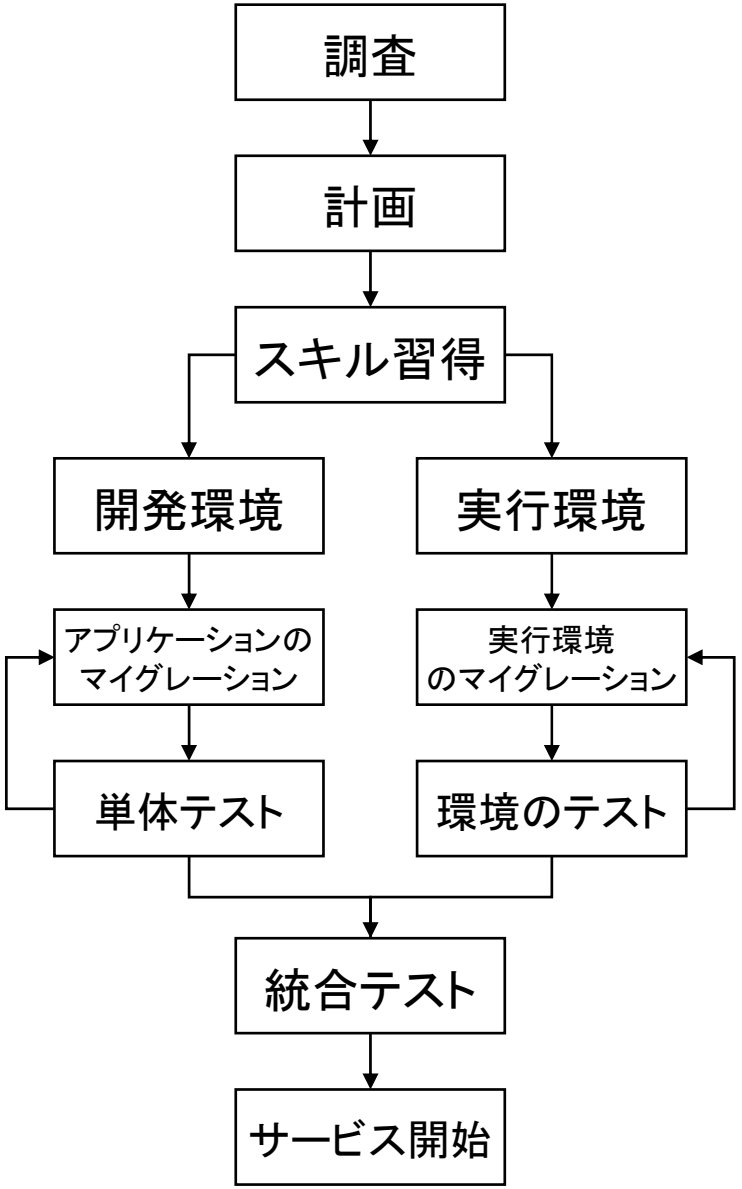
    http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc_newinrelease.html
  - WAS V6.0 アナウンスメントワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/was/was6_ws/
- V6.1での新機能
  - InfoCenter “What is new in this release”
    http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc_newinrelease.html
- V7.0での新機能
  - InfoCenter "New Feature"
    http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6topnew.html
  - WAS V7.0 アナウンスメント・ワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/was/was7_ws/
- V8.0での新機能
  - InfoCenter "New Feature"
    http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6topnew.html

マイグレーションのロードマップ



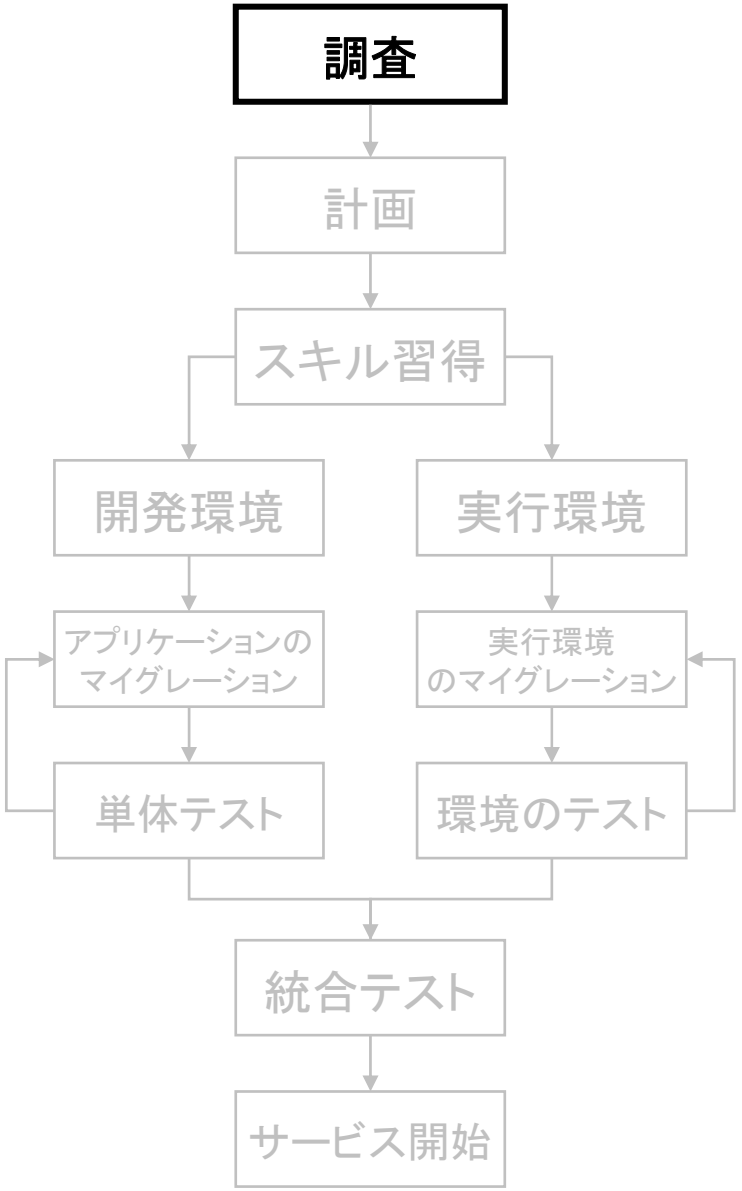
マイグレーションのロードマップ

- 調査
- 計画
- スキルの習得
- 開発環境
  - 開発環境の更新
  - アプリケーションのマイグレーション
  - 単体テスト
- 実行環境
  - 実行環境のマイグレーション
  - 実行環境のテスト
- 統合テスト
- サービス開始



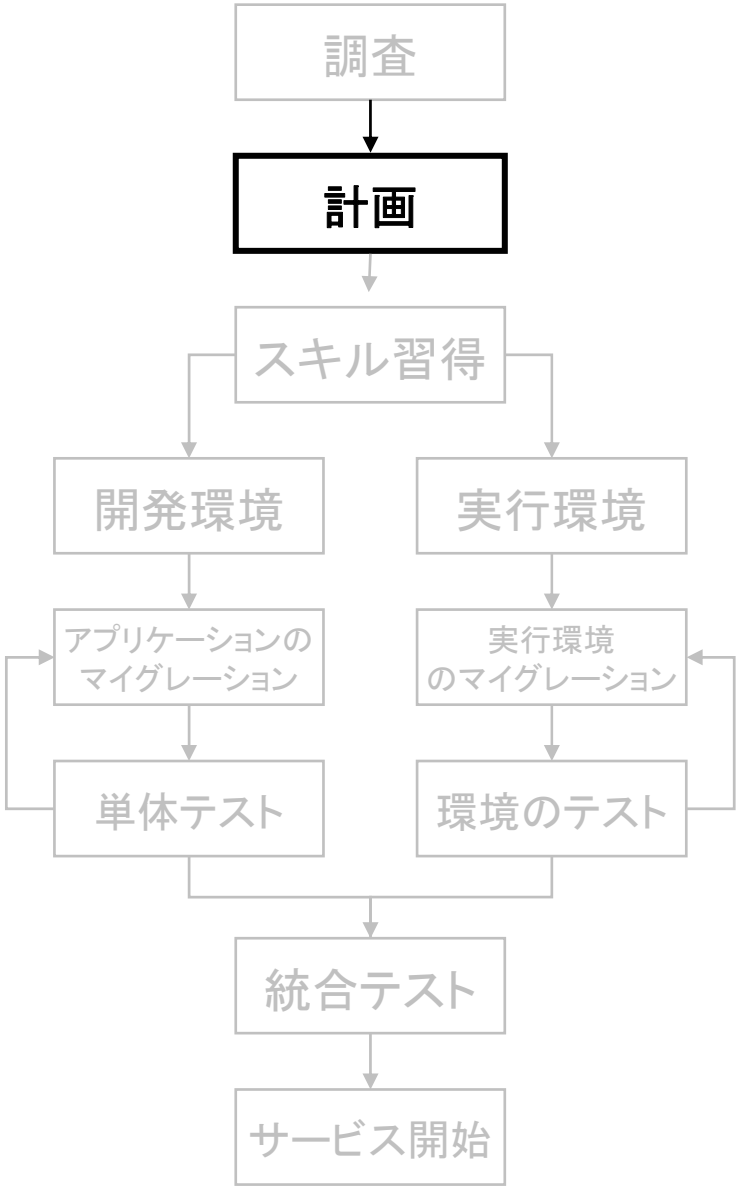
ロードマップ：調査

- 現環境の調査
  - S/Wの種類・バージョン
  - アプリケーションの数と依存関係
  - 使用しているAPI・仕様
  - 開発ツール・手法
- 要件の洗い出し
  - 現環境の問題点
  - 新環境で想定される要件
  - サービスに要求される可用性
- 最新情報の収集
  - 新バージョンのS/W
  - 使用できる機能・仕様
  - サポートされる組合わせ
  - 最新の開発ツール・手法
- 体制
  - 参画できる開発者・運用担当者
  - 必要なスキル習得項目
  - 提供されるサービス



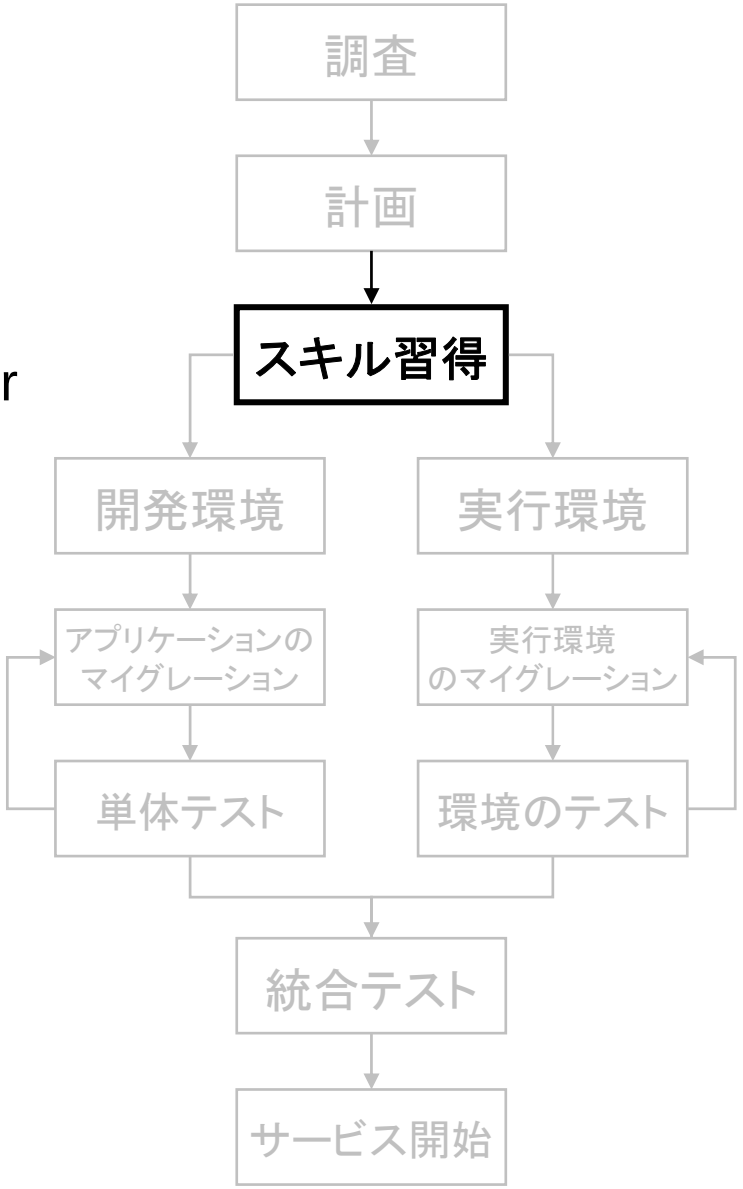
ロードマップ：計画

- 更新内容の決定
  - WASや関連ソフトのバージョン
  - 使用するH/W構成
  - 更新するアプリケーション
- 新環境のトポロジー
  - 必要とされる可用性
  - 要求されるパフォーマンス
- アプリケーション
  - 使用するAPI・仕様
  - 開発に使用するツール・手法
  - 更新の基本方針
- テスト内容の検討
- スケジュールの策定
  - スケジュール・タイムライン
  - 体制
  - リスク項目の洗い出し
  - 戻し手段の確保



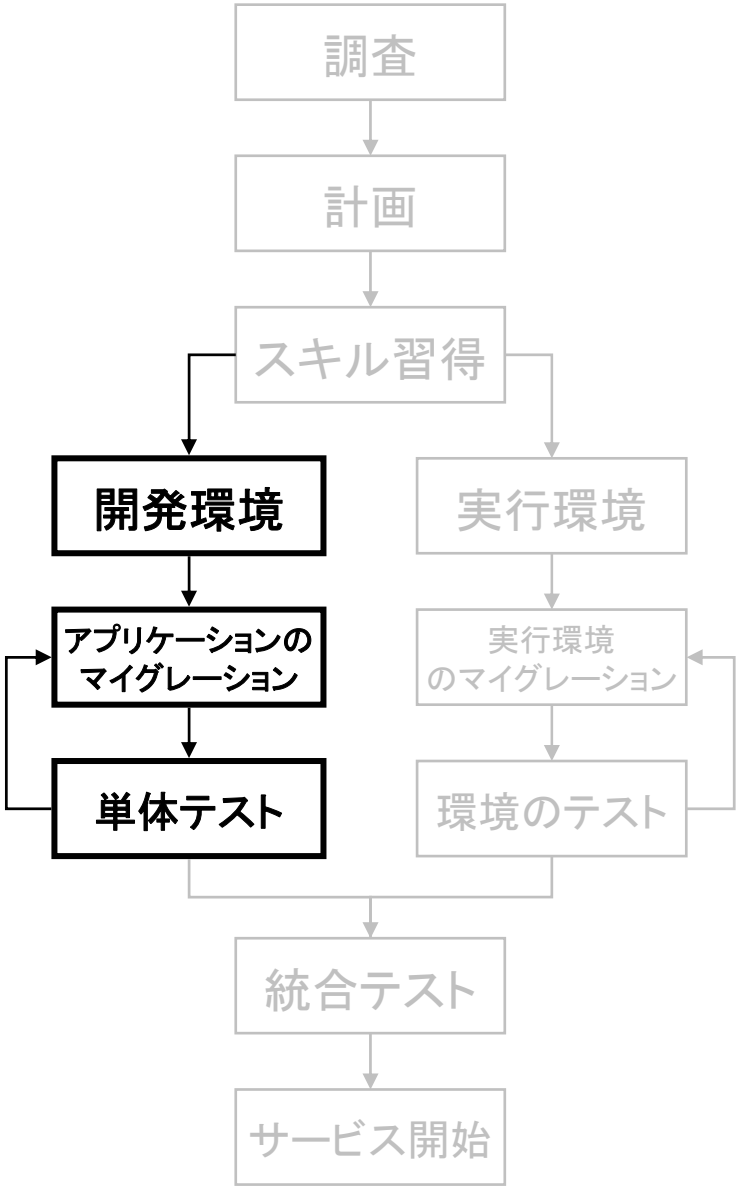
ロードマップ：スキル習得

- スキル習得
  - 新しい開発ツール
  - 最新のWebSphere Application Server
  - 最新仕様
  - フレームワーク
  - WebSphere管理モデルの変更点
- パイロットプロジェクトの策定



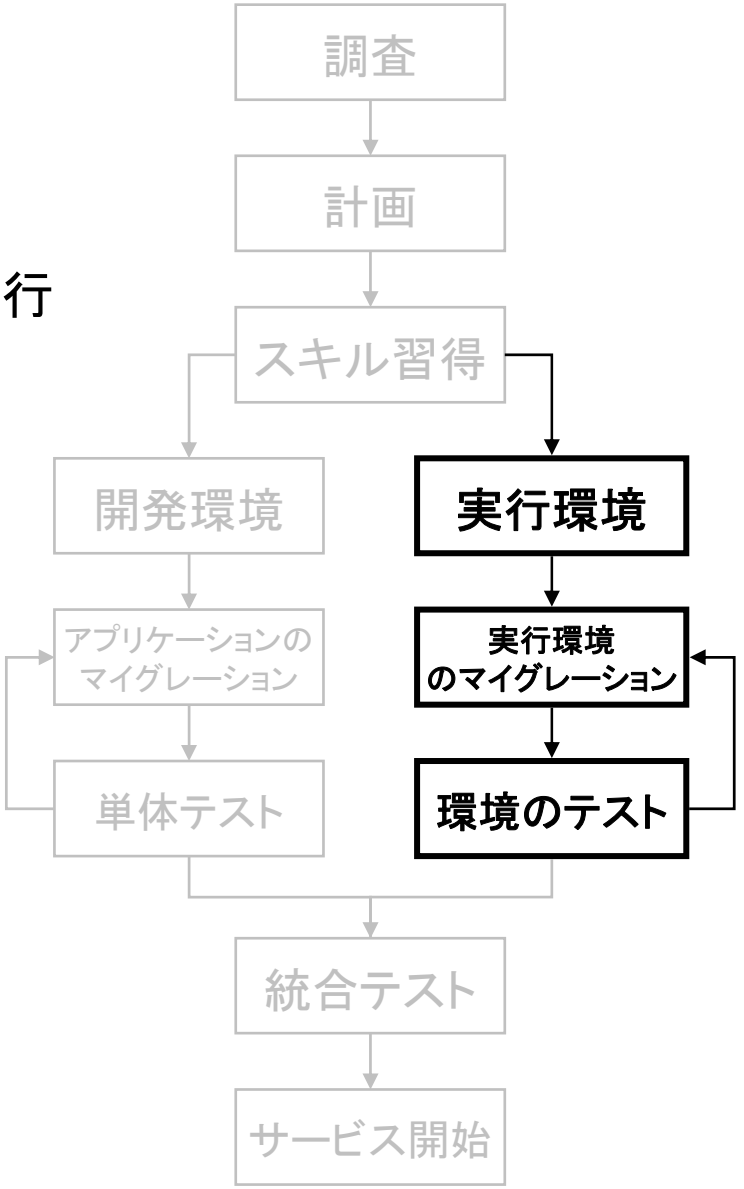
ロードマップ：開発

- 開発に使用するIDE／ツールの更新
- アプリケーションに互換性がある場合
  - 既知の問題についての調査を実施
  - 問題がなければテストを実行
- 更新が必要である場合
  - 移行ツールによる更新（P27参照）
  - 手動の更新
  - 追加開発の実施
- 非推奨機能を使用した部分の洗い出し
  - 更新の検討
- 単体テスト



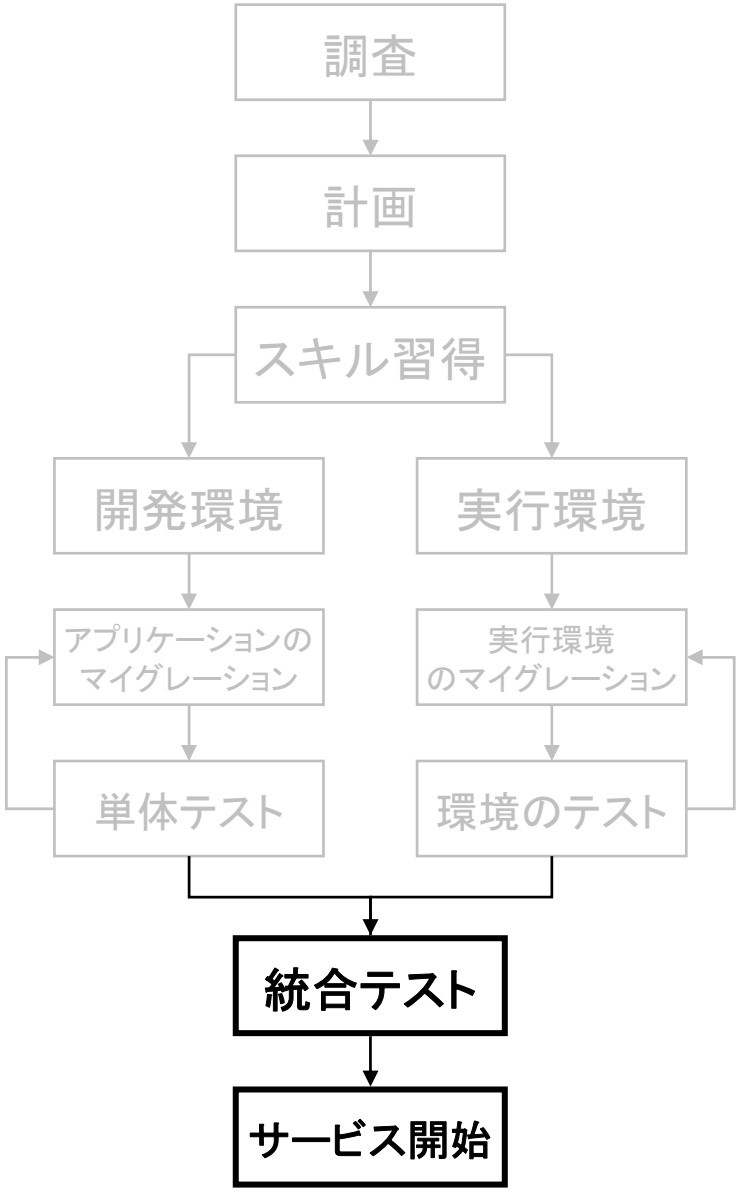
ロードマップ：実行環境

- ツールによる移行が可能な場合
  - マイグレーションツールによる移行の実行
- 手動で移行する場合
  - ソフトウェアの新規導入
  - 新しい構成の実行
- 各種設定・パラメーターの決定
- 運用スクリプト・監視ツールの移行
- テストアプリケーションによる稼動確認
- 障害発生時のフェールオーバー確認



ロードマップ：

- 統合テストの実施
  - アプリケーションの稼動確認
  - テスト実行・結果評価の自動化（P31-34を参照）
- パフォーマンス測定
  - 想定アクセス量の正常処理を確認
  - 限界アクセス量の確認
- トラブル時に元の環境に戻せる準備を



マイグレーション対象のエディション

- WAS 4.0以前からのマイグレーション
  - 単一のサーバーでアプリケーションを実行
  - 障害発生時のフェールオーバーは不要 / サーバーの異常発生時には再起動で対応
    → WebSphere Application Server（Baseエディション）を使用する
  - 複数サーバーでのクラスタリング機能が必要
  - 万が一の障害発生の場合にもサービスをとめることができない
    → WebSphere Application Server Network Deploymentを使用する
- WAS 5.0以降からのマイグレーション
  - 基本的に現在使用しているものと同エディションを選択すればよい
  - Enterpriseエディションを使用していた場合，
    WebSphere Process Serverなどのプロセスインテグレーション製品への移行を検討する

各バージョンでのエディション構成

WAS 3.0/3.5	WAS 4.0	WAS 5.x	WAS 6.x/7.0/8.0
▪Standard Edition ▪Advanced Edition ▪Enterprise Edition	▪Advanced Single Server Edition ▪Advanced Edition ▪Enterprise Edition	▪Express ▪(Base) ▪Network Deployment ▪Enterprise	▪Express ▪(Base) ▪Network Deployment

WAS V8のエディション構成

内容	WAS Express	WAS (Base)	WAS ND (Network Deployment)	WAS for Developer
導入可能プロセッサー数	PVU 480まで	無制限	無制限	無制限
課金体系	PVU課金	PVU / ソケット課金	PVU課金	ユーザー課金
64bit対応	×	○	○	×
Java EE 6サポート (EJB 含む)	○	○	○	○
Feature Pack サポート	○	○	○	○
IBM HTTP Server /Web Server plug-inの同梱	○	○	○	○
Edgeコンポーネント同梱	×	×	○	×
Proxy Server / Secure Proxy Server 同梱	×	×	○	×
複数サーバーの集中管理	×	×	○	n/a
クラスター構成とフェイルオーバー				
セッションのフェイルオーバー	2サーバーまで○ 方法：DB	5サーバーまで○ 方法：DB	○ 制限なし 方法：DB or メモリー間複製	n/a
2PCコミットにおけるトランザクションのフェイルオーバー	×	×	○	n/a
メッセージングのフェイルオーバー	×	×	○	n/a
Job Manager同梱	× Job Managerの管理対象にはなれる	× Job Managerの管理対象にはなれる	○	n/a
同梱ツール(アセンブリー＆デプロイ)	IADT	IADT	IADT	IADT
DB2 同梱 *	○	○	○	○
Tivoli Directory Server 同梱 *	○	○	○	○
Tivoli Access Manager同梱 *	×	○	○	×

IADT: IBM Assembly and Deploy Tools for WebSphere Administration

*同梱されるDB2、Tivoli製品には使用に制限があります。

関連ソフトウェアの更新

- ベンダーからのサポート状況や，組み合わせのサポート可否などの条件により，WASと同時に使用しているS/Wなどについても，更新が必要な場合もある
  - プラットフォーム（OS，H/W）
  - データベース
  - メッセージング製品　　等
- WASとの連携がサポートされているS/Wについては，以下のURLで確認可能
  - http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006921

例）WASとの組み合わせがサポートされている製品のバージョン（WAS出荷時点）

	WAS 5.1.1	WAS 6.0	WAS 6.1	WAS 7.0	WAS 8.0
Windows	2000/2003	2000/2003	2000 /2003/2008	2003/2008	2003 SP2 /2008
AIX	5.1/5.2/5.3	5.1/5.2/5.3	5.2/5.3/6.1	5.3/6.1	6.1/7.1
Sun Solaris	8/9	8/9/10	9/10	9/10	10
Apache	1.3/2.0	2.0	2.0/2.2	2.0/2.2	2.2
DB2	7.2/8.1/8.2/9	8.1/8.2/9	8.1/8.2/9/9.5	8.1/8.2/9/9.5	9/9.5/9.7
Oracle	8i/9i/10g	8i/9i/10g	9i/10g	10g/11g	10g/11g

開発環境の更新

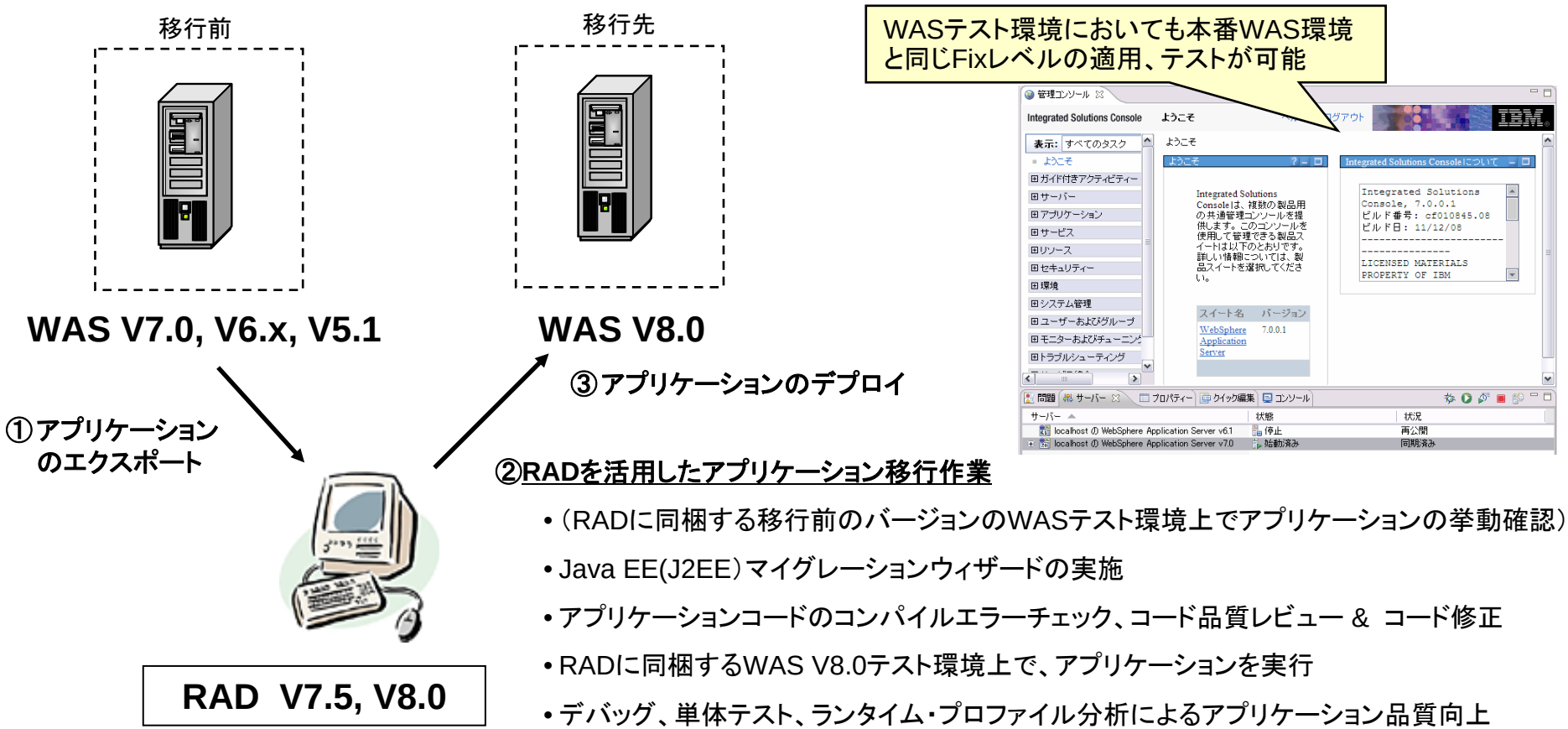
- IDEを使用していなかった場合，可能な限り使用する様にする
  - J2EE仕様自体がIDEを前提としたものとなっており，構成ファイルを手で編集することは現実的ではない
  - コードの変更項目を洗い出すには，IDEの構文チェック機能を使用するのが最適
  - デプロイメントディスクリプタのバージョンの更新（J2EE 1.2→1.4）などが自動で行える
  - 後述するWebSphere Application Server Migration Toolkitが利用できる
- VisualAge for JavaのリポジトリやWebSphere Studio 5.0のワークスペースを Rational Application Developerに直接取り込むことはできない
  - ソースコード等をExportしたうえで，個別に取り込む
- WebSphere Studio 5.1やRational Application Developer 6.0からは，最新の環境へ直接マイグレーションすることが可能

標準的な開発ツール

WAS 4.0	WAS 5.0/5.1	WAS 6.0	WAS 6.1	WAS 7.0	WAS 8.0
WebSphere Studio 4.0/5.0	WebSphere Studio 5.1	Rational Application Developer 6.0	Rational Application Developer 7.0	Rational Application Developer 7.5	Rational Application Developer 8.0

Rational Application Developer (RAD) を使用したWASマイグレーション支援

- WebSphere Application Server (WAS) 上のアプリケーション開発を強力にサポート
  - RADV7.5には、WAS V7.0, V6.1, V6.0のローカルテスト環境が同梱
  - RADV8.0には、WAS V8.0, V7.0, V6.1のローカルテスト環境が同梱
  - WAS上のアプリケーションの開発、実行、デバッグ、単体テスト、ランタイム・プロファイル



②RADを活用したアプリケーション移行作業

- （RADに同梱する移行前のバージョンのWASテスト環境上でアプリケーションの挙動確認）
- Java EE(J2EE)マイグレーションウィザードの実施
- アプリケーションコードのコンパイルエラーチェック、コード品質レビュー & コード修正
- RADに同梱するWAS V8.0テスト環境上で、アプリケーションを実行
- デバッグ、単体テスト、ランタイム・プロファイル分析によるアプリケーション品質向上

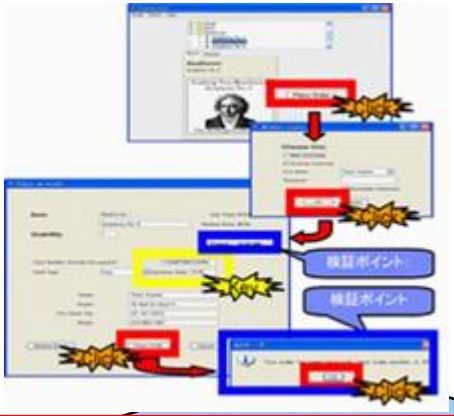
機能テストの繰り返し
Rational Functional Tester (RFT) による自動機能テスト

Web, Java, Flex などの
アプリケーションに対応

- 記録した操作を再生し、画面上のデータや隠れたプロパティーを自動検証

テストシナリオ設定 ⇒ テスト実施 ⇒ 操作の記録 ⇒ スクリプト生成 ⇒ 操作の再生 ⇒ 結果の検証

対象アプリケーション
画面遷移テストシナリオ





マウス操作



キー入力

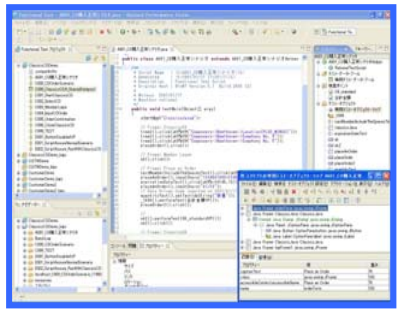
画面操作を記録

テストスクリプト
自動生成

テストスクリプト
(Java)

必要に応じて拡張可能
(Java)

記録した操作を再生



データプールにより、スクリプトを変更せずテストケース数を増やすことが可能

	カード番号::java.lang....	有効期限::java.lang.:
0	1234 1234 1234 1234	12/07
1	0000 0000 0000 0000	04/11
2	8888 9999 0000 1111	07/13

WASマイグレーションなどの稼働環境の変更時やアプリケーションの
修正/変更時に一度記録したスクリプトで繰り返しテストを実行

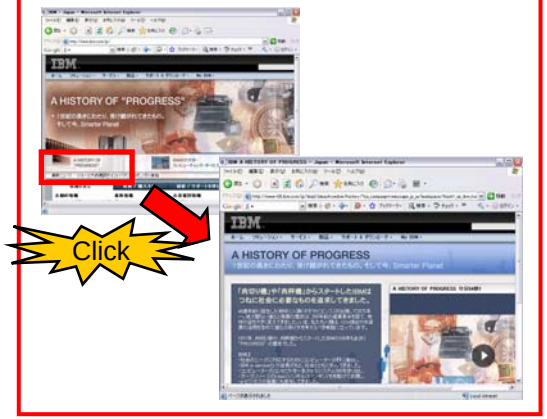
パフォーマンス測定（負荷テスト）
Rational Performance Tester (RPT) による負荷テスト

負荷テスト実行時のサーバー・リソースの収集も可能

- HTTPプロトコルのやり取りを記録し、大量の仮想ユーザーで負荷テストを実行

テストシナリオ設定 ⇒ 操作の記録 ⇒ スクリプト生成 ⇒ 負荷設定 ⇒ テストの実行 ⇒ 結果の検証

対象アプリケーション
画面遷移テストシナリオ





マウス操作



キー入力

プロトコルのやり取りを記録

テストスクリプト自動生成

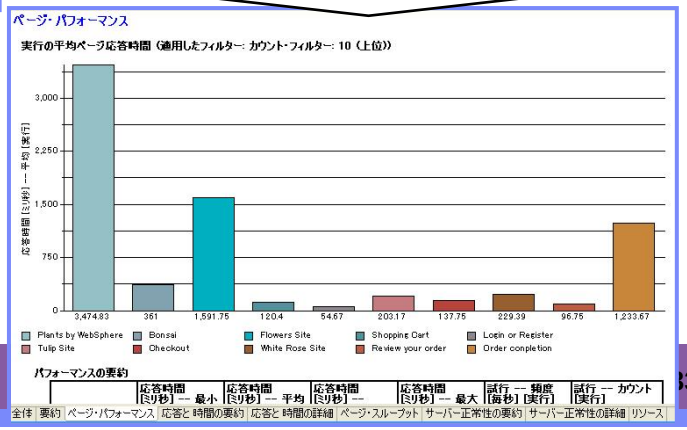


何人のユーザーがどのようなエミュレートを行うか

テストシナリオ作成



スケジュール実行とレポートの生成

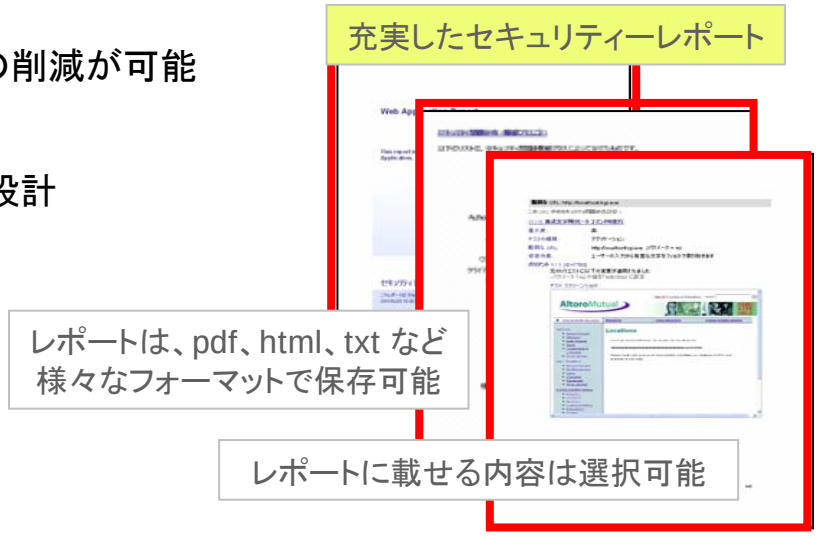


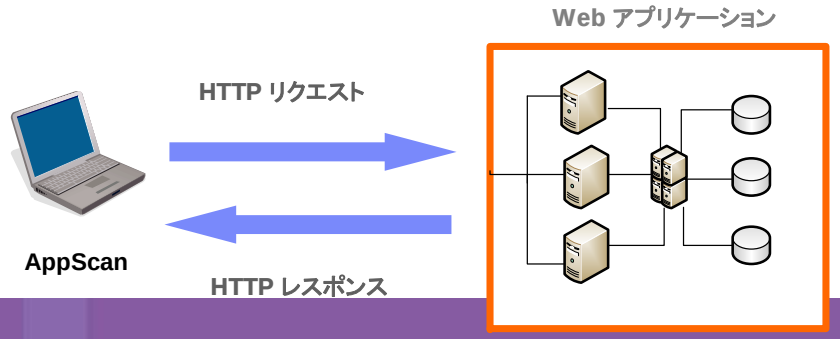
WASマイグレーション時にサーバーへ
大量の負荷を与え最適なチューニングを行う

Rational AppScan によるWebアプリケーションセキュリティー・テスト

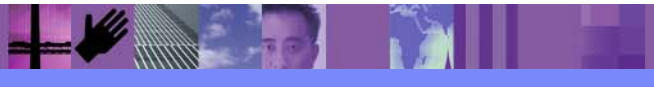
- **開発者からセキュリティ専門家まで、幅広く使えるスタンドアローンの Webアプリケーション セキュリティ テスト ツール**
  - Webアプリケーションの脆弱性と、インフラ(Webサーバー等)の設定ミス、既知の問題を検知
  - Web アプリケーションをブラックボックスとして検査
  - テストを自動化し、手作業に比べて圧倒的な時間とコストの削減が可能
  - 脆弱性の指摘、修正方法の提示、レポートの作成
  - セキュリティ専門家も利用する高機能ながら、使いやすい設計







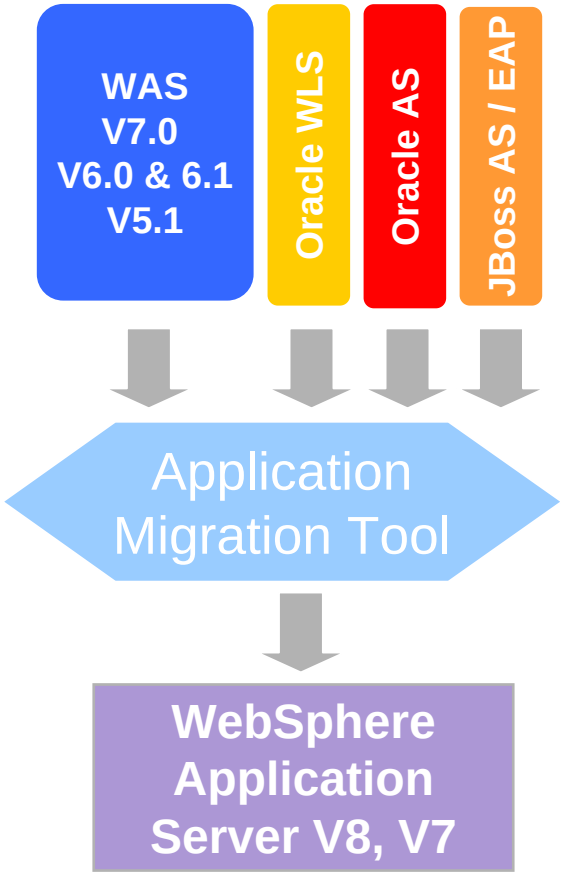
アプリケーションのマイグレーション



Application Migration Tool

- Application Server Migration Toolkit
  - Competive Application Migration Tool
    - WebLogic, JBoss, Oracle ASなどからの移行を支援するツール
  - WebSphere Application Migration Tool
    - WASの旧バージョンからの移行を支援するツール
- 無償でダウンロードいただいて使用することができます
- EclipseやRational Application Developerに組み込んで使用
- Rational Software Analyzerの技術を使用してJavaコード, JSP, デプロイメント記述子の調査
- 修正が必要な箇所をリストアップし修正方法についてガイドを表示

移行の負荷を大幅に削減



http://www.ibm.com/developerworks/jp/websphere/library/was/was7_migration_toolkit/

アプリケーションのマイグレーションの難易度

From＼To	3.0	3.5	4.0	5.0	5.1	6.0	6.1	7.0	8.0
7.0									Easiest
6.1								Easiest	Easiest
6.0							Easier	Easier	Easier
5.1						Easy	Easy	Easy	Easy
5.0					Easy	Easy	Easy	Easy	Easy
4.0				Easy *	Easy *	Easy *	Easy *	Easy *	Easy *
3.5			**Difficult**	Difficult	Difficult	Difficult	Difficult	Difficult	Difficult
3.0		Easy *	**Difficult**	Difficult	Difficult	Difficult	Difficult	Difficult	Difficult
2.0	Easy *	Easy *	**Difficult**	Difficult	Difficult	Difficult	Difficult	Difficult	Difficult

* 難しくなるケースもしばしばある

WASの各バージョンが対応しているJava EE/Java SE仕様

WAS V5.0	**J2EE 1.3** Servlet 2.3/JSP 1.2 EJB 2.0			**J2SE 1.3**
WAS V5.1	**J2EE 1.3** Servlet 2.3/JSP 1.2 EJB 2.0	JSP/Servlet 仕様の変更		**J2SE 1.4**
WAS V6.0	**J2EE 1.4** Servlet 2.4/JSP 2.0 EJB 2.1		Java 5対応	**J2SE 1.4**
WAS V6.1	**J2EE 1.4** Servlet 2.4/JSP 2.0 EJB 2.1			**J2SE 5.0**
WAS V7.0	**J2EE 1.4** Servlet 2.4/JSP 2.0 EJB 2.1	**Java EE 5** Servlet 2.5/JSP 2.1 EJB 3.0		**Java SE 6**
WAS V8.0	**J2EE 1.4** Servlet 2.4/JSP 2.0 EJB 2.1	**Java EE 5** Servlet 2.5/JSP 2.1 EJB 3.0	**Java EE 6** Servlet 3.0/JSP 2.2 EJB 3.1	**Java SE 6**

J2EE/Java EEアプリケーションのマイグレーションの概要

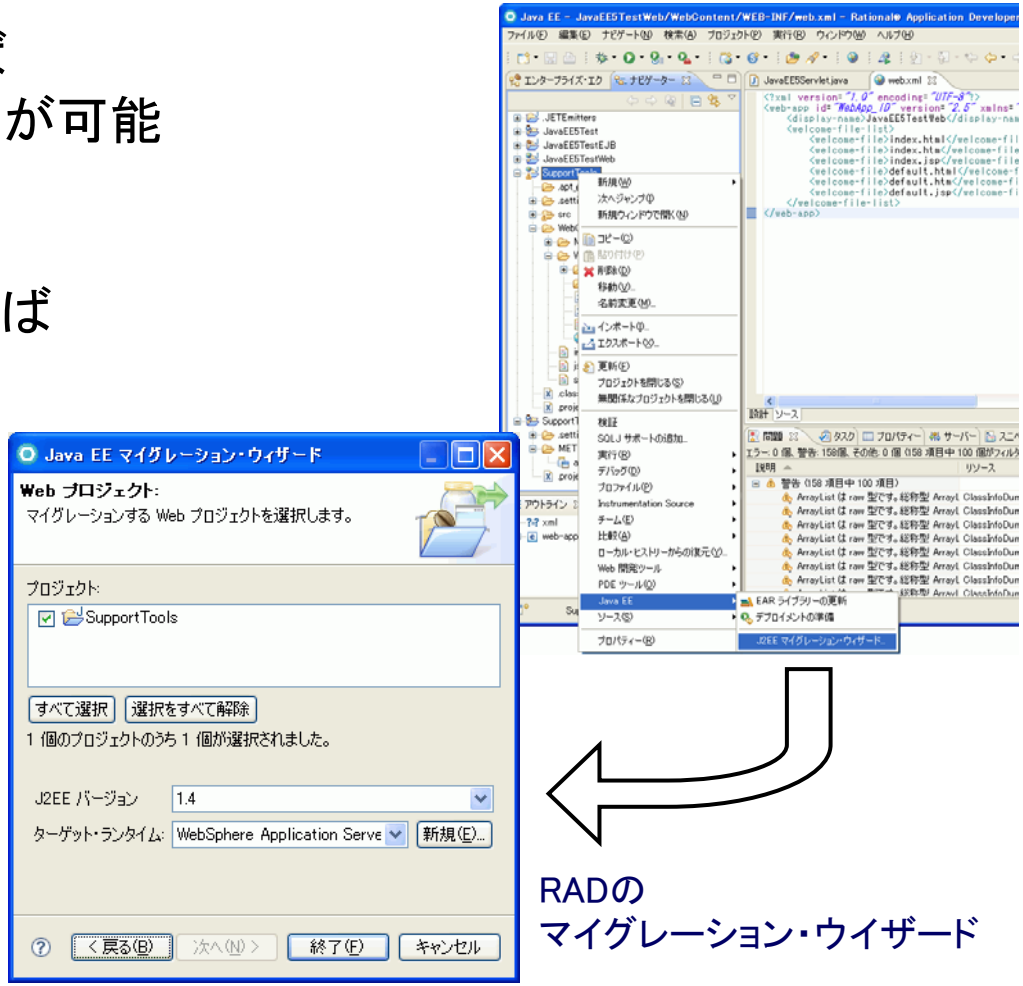
- J2EE仕様以前のアプリケーションは大規模な書き換えが必要
  - EAR/WARなどのアプリケーションパッケージ仕様に移行する必要がある
  - JSP 0.91やIBM Connection Managerなど根本的に書き換えが必要なものも
- J2EE 1.2からJ2EE 1.3以降へは基本的な上位互換が保たれている
  - ただし，J2EE 1.3から国際化対応の機能が導入されたため，
    日本語環境のアプリケーションでは何らかの対応が必要となるケースが多い
  - 細かい仕様の変更があるため，個別に対応が必要となるケースも
- J2EE 1.3からJ2EE 1.4以降へは高いレベルで上位互換が保たれている
  - 多くのアプリケーションが変更無しに稼動する
  - アプリケーションによっては若干の手直しが必要となるケースもある
- 基本的にJ2EE 1.4アプリケーションであれば，WAS V8.0でそのまま動作可能
- Webサービス
  - J2EE 1.3以前では標準化されていないため，書き換えが必要となるケースが多い
- IBM独自機能を使用している部分については，書き換えが必要となるケースも

J2EE 1.2/1.3アプリケーションのJ2EE 1.4への移行

- J2EE 1.4アプリケーションであれば
  WAS V8.0でそのまま実行することが可能

- RADなどの開発ツールを使用すれば
  DDなどの構成ファイルを
  自動的に書き換えることが可能

- 既存のアプリケーションを
  Java EE 5/6形式に書き換える
  必要はない
  - 新規アプリケーション開発で
    Java EE 5/6を使用する



RADの
マイグレーション・ウイザード

Java SE仕様（J2SE仕様）のマイグレーション

- JDK 1.1以前のアプリケーションは，書き換えることが望ましい
  - 今日では非推奨となったAPIが使用されていることがおおい
  - 当時のAPIは貧弱なため，最新のAPIを使用した方が，よりよいコードに
- JDK 1.2/1.3/1.4からJDK 5.0/6.0への移行は
  - バイトコードレベルで上位互換が保たれている
    - 以前の環境でコンパイルしたClassファイルはほとんどそのまま使用できる
  - ソースコードレベルでは書き換えが必要なケースも
    - 追加された予約語「assert」「enum」の対応
    - Genericsへの対応
    - 強化されたコードチェックによる，エラー・警告への対応
    - java.lang.reflect.Proxyと同名のjava.net.Proxyが追加
    - java.util.Loggerのコンストラクタの変更

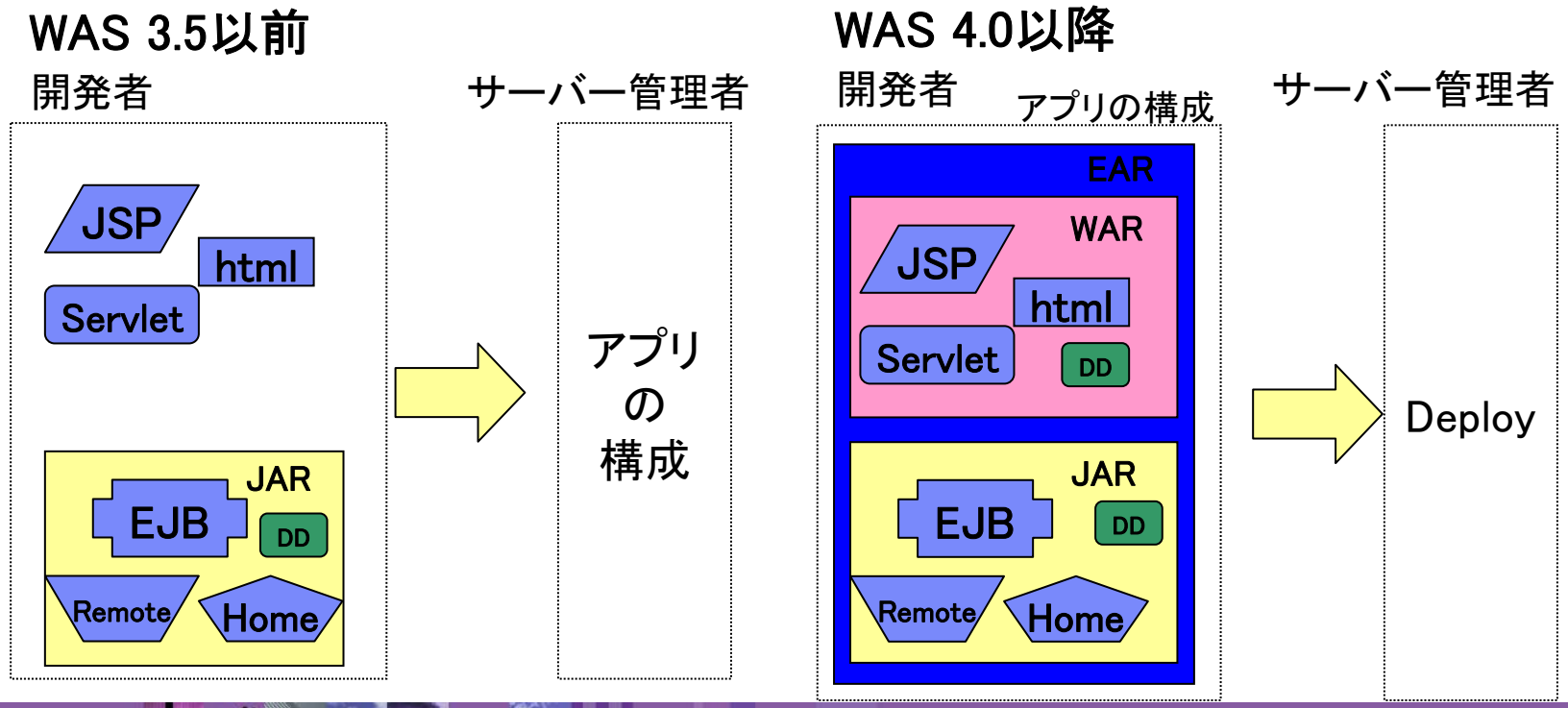
    等
  - 詳細については，以下の文書を参照
    http://java.sun.com/j2se/1.5.0/compatibility.html
- JDK 5.0からJDK 6.0については，極めて高いレベルで上位互換性がとられている
  - 詳細については，以下の文書を参照
    http://java.sun.com/javase/ja/6/webnotes/compatibility.html

JDKのバージョンによる挙動の違い

- 内部実装クラスへの依存
  - JDK/JRE内部で使用されている非公開クラスの実装はバージョンごとに変更されることが多々あります
  - 「JavaプラットフォームAPI仕様」に載っていないJDK/JREのクラスを直接使用している場合，移行に当たって修正が必要となるケースがあります
  - 基本的に，アプリケーションを作成する際には，これらの内部クラスを直接使用しないように実装してください
- file.encodingのコマンドライン引数（-D）による変更
  - デフォルトのファイルエンコーディングを指定するfile.encodingシステムプロパティをコマンドライン引数で変更した場合の挙動は，JDKのバージョンによって異なります
    - 変更が有効になるものと有効にならないものがあります
  - デフォルトのファイルエンコーディングに依存し，「 file.encoding」でデフォルトのエンコーディングを変更した上で実行しているアプリケーションは，移行に当たって修正が必要となるケースがあります
  - アプリケーションを作成する際には，文字の変換処理では必ずエンコーディングを指定して，デフォルトのエンコーディングによって挙動が変わらないように実装してください

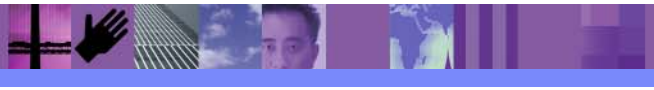
WAS 3.5以前からのマイグレーション（1）

- J2EEのアプリケーションパッケージングへの移行
  - デプロイメント・ディスクリプタ（DD）によるアプリケーション構成
  - 関連ファイルをDDとともにEAR/WARファイルへのパッケージング
  - サーバー管理者はパッケージされたアプリケーションをサーバーに配備（Deploy）



WAS 3.5以前からのマイグレーション（2）

- Servlet API 2.0（JSDK 2.0）は最新のServlet APIに書き換え
- JSP 0.91 APIを使用したページは，ほぼ全面的に書き換え
- EJB 1.0は，最新のEJB仕様/JPA仕様にしたがって書き換え
- JNDIによるリソースの取得は，リソース参照を経由したものに変更する
  （もしくは，アノテーションを使用したDIに変更する）
- IBM Connection Managerを使用したDBアクセスコードはJDBCに書き換え
- 他，廃止されたIBM独自機能を使用している部分の書き換え
  - PageListServlet，Data Access Bean，JSPのtsxタグ，・・・
  - com.ibm.ejs.ns.jndi.CNInitialContextFactory
    - Deprecated（非推奨）に変更
      代わりにcom.ibm.websphere.naming.WsnInitialContextFactoryを使用
  - com.ibm.ejs.cm.portability.StaleConnectionException
    - Deprecated（非推奨）に変更
      代わりにcom.ibm.websphere.ce.cm.StaleConnectionExceptionを使用



WAS 3.5以前からのマイグレーション（3）

- 階層化されたクラスローダー
  - WAS 4.0以降では，コンポーネント（EAR / WAR / EAR）ごとに複数のクラスローダーが用意されている
  - 階層の親子関係，参照可否にたいする考慮が必要（詳細は後述）
  - JNI（Java Native Interface）を使用したライブラリには特別の考慮が必要
- HttpSessionの仕様変更への対応
  - 有効範囲の変更（Webアプリケーション内に限定）
    アプリケーションをまたいでのHttpSessionによる情報の共有は不可に
  - HttpSession#setAttribute(String name, Object attr)では，
    name，attrともにnull値が禁止に
  - HttpSession#setValue()系のメソッドは全てDeprecatedに
    HttpSession#setAttribute()に置き換えることを推奨

非推奨となったMethod	代替のMethod
getValue()	getAttribute()
getValueNames()	getAttributeNames()
removeValue()	removeAttribute()
putValue()	setAttribute()

WAS 4.0から：国際化機能への対応

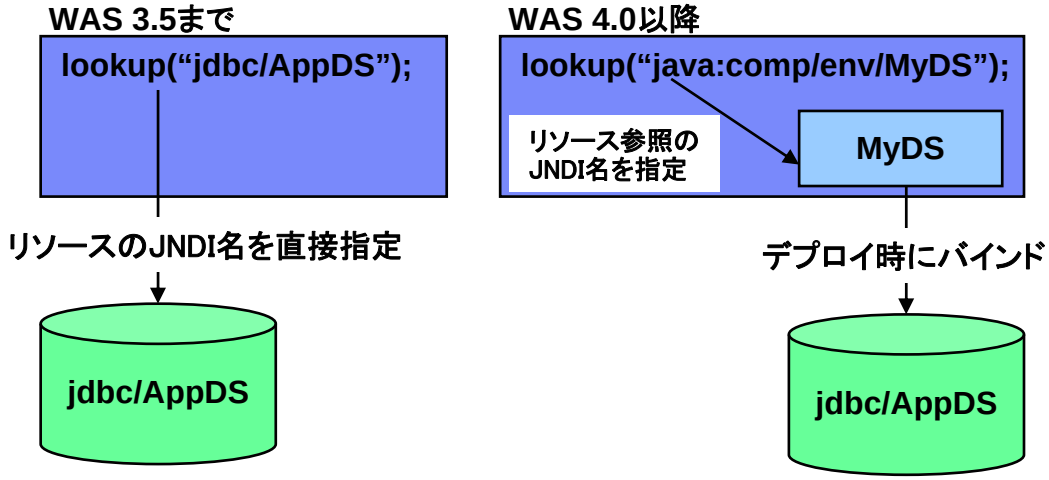
- WAS 5.0（J2EE 1.3）以降で，文字コードを指定する仕様が多数追加された
  - WAS 4.0（J2EE 1.2）以前では，サーバーの構成で指定していた
- クライアントからのリクエストパラメーターの文字コード
  - ServletRequest#setCharacterEncodingによる指定が必須に
- Servlet/JSPからのレスポンスの文字コード指定
  - デフォルトcharsetが8859_1に
  - ServletResponse#setContentTypeや
pageディレクティブのcontentType属性にcharsetの指定が必須に
- JSPのソースコードの記述文字コード
  - pageディレクティブのpageEncoding属性の指定が必要
  - デプロイメントディスクリプタ（web.xml）での一括指定も可能（J2EE 1.4以降）

WAS 4.0から:EJBの更新

- 互換モードにより，J2EE 1.2（EJB 1.1）
  アプリケーションもWAS V8.0上で稼動はする
  - リソースの使用に制限
    （WAS V4.0データソースしか使用できない）
- EJBのEntity Beanは書き換えを推奨
  - DBへの永続化の仕組みがJPAとして分離されたため
  - Java EE 6では，Entity Beanは非推奨
- BMT Session Beanは，
  CMT Session Beanへの書き換えを推奨
- CMT Session Beanについては，再コンパイル・デプロイコードの再作成で対応可能な場合が多い

WAS 4.0から:リソース参照

- DataSourceなどのリソースは，必ずリソース参照を経由して取得
  - WAS 3.5までは，JNDIでDataSourceなどを取得する際に，リソースの名前（グローバル名）を直接指定してlookupしていた
  - WAS 4.0以降では，アプリケーションに構成したリソース参照の名前（ローカル名）を指定してlookupをおこなう



- リソース参照には，認証方法（コンテナ認証/アプリケーション認証），共用設定（J2EE 1.3以降），トランザクション属性（IBM独自）などが指定可能
  - リソース参照で指定しなかったときのデフォルトのトランザクション属性がWAS 4.0とWAS 5.0以降で変更
  - アプリケーションによっては共用属性を正しく設定しないと問題が発生する可能性もある

WAS 4.0から：スレッド

- WAS 5.0以降で段階的に制限が厳格に
（仕様外の動作がエラーになるケースがある）
  - EJBコンテナ上では，ユーザースレッドの作成は禁止
  - トランザクションに関わるオブジェクト（DBコネクションなど）は
スレッド間で共有不可
  - Webコンテナ提供のオブジェクトの
サーブレット実行スレッド外での使用不可
    - HttpServletRequest/HttpServletResponseのインスタンス
    - HttpSessionのインスタンス　　等
- JMSのListenerは，EJB MDBに移行
  - 参照：
http://publib.boulder.ibm.com/infocenter/wasinfo/v5r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/tmb_migmb.html
- MQのBase Javaのクラスの中には，内部で独自のスレッドを生成するものがあり，
移行に当たって問題となるケースがある

WAS 4.0から:その他

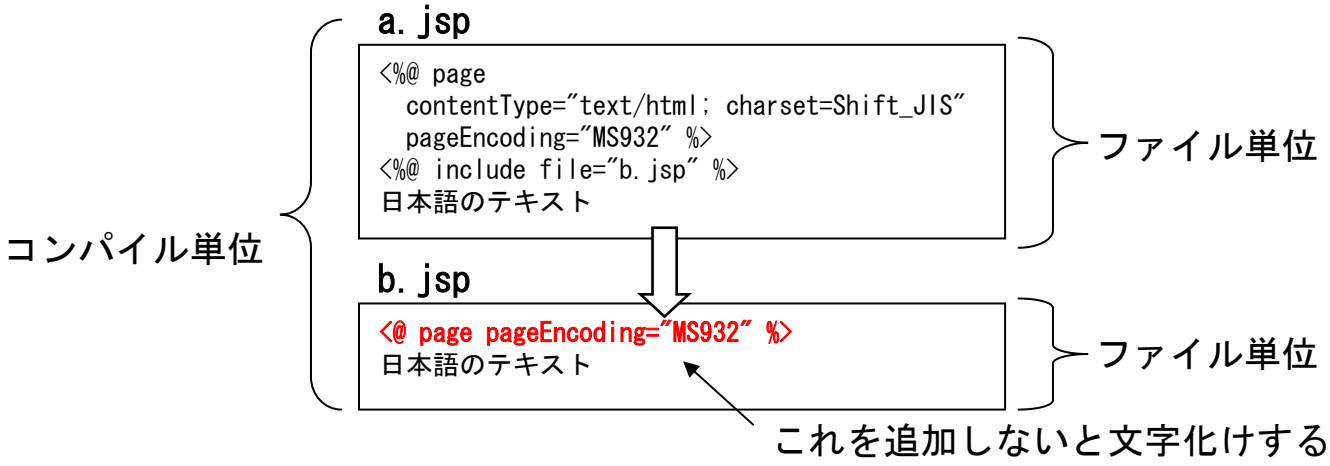
- 分散ネーミングサービスへの対応
  - Provider URLの変更(iiop:～ → corbaloc:～)
    - 障害時のフェールオーバー先を指定可能に
  - EJBのJNDI名の変更
    - WAS 3.5/4.0
      "EJBHome01"
    - WAS 5.0～(Single Server)
      "cell/nodes/h01Node/servers/server1/EJBHome01"
    - WAS 5.0～(Cluster)
      "cell/cluster/cluster01/EJBHome01"
- HttpServletResponse.sendRedirectの挙動の変更
  - /webapp/examplesのサーブレットからsendRedirect("me.jsp")を実行したとき
    - WAS 4.0
      http://localhost/webapp/examples/me.jsp にリダイレクト
    - WAS 5.0以降
      http://localhost/me.jspにリダイレクト
  - JVMシステムパラメーターの指定で変更することが可能
  - 詳細は以下の文書を参照
    http://publib.boulder.ibm.com/infocenter/wasinfo/v5r1/topic/com.ibm.websphere.base.doc/info/aes/ae/xrun_jvm_sendredirect.html

WAS 5.0から：ServletErrorReport

- com.ibm.websphere.servlet.error.ServletErrorReport
  - Error JSPで発生したエラーの詳細を取得するためのクラス
  - Exceptionのサブクラス
  - JDK 1.4よりgetStackTrace()がjava.lang.Throwableで定義された
  - ServletErrorReportで定義されていたpublic String getStackTrace()は public String getStackTraceAsString()に変更
  - getStackTrace()の返り値はjava.lang.Throwableのものに変更に public StackTraceElement[] getStackTrace()

WAS 5.0/5.1から：JSP 2.0の仕様変更への対応

- JSP 2.0からEL（Expression Language）が仕様に追加された
  - JSP中に「${ ～ }」「#{ ～ }」という文字列が存在している場合ELと解釈される
  - これらの文字列が含まれている場合，適宜エスケープ（¥記号の追加）をおこなう
- JSP 1.2から2.0でpageEncodingの指定方法が変更に
  - JSP 1.2：　コンパイル単位でEncodingを指定
  - JSP 2.0：　ファイル単位でEncodingを指定
- includeディレクティブで他のJSPファイルを読み込んでいる場合に、pageディレクティブとpageEncodingを追加
- アプリ内の全てのJSPファイルが全て同じエンコードの場合にはweb.xmlで一括指定が可能
  - <jsp-config>要素の配下の<page-encoding>要素で指定する



WAS 5.0/5.1から：32bit WASから64bit WASへの移行

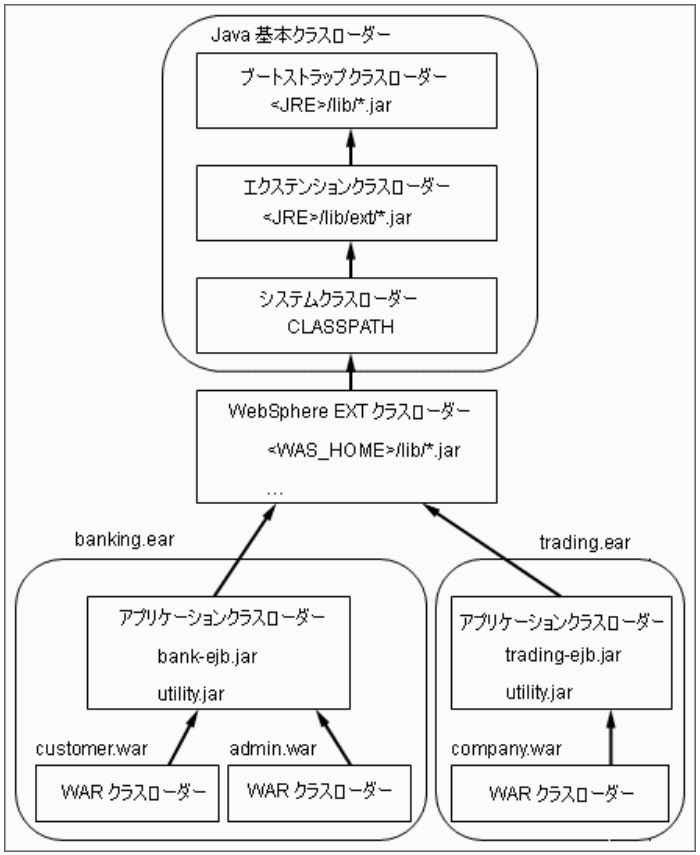
- WAS V6.0.1から徐々に、64bit ネイティブ環境で動くWASを提供
- WAS V7.0/8.0では，全てのプラットフォームで64bitエディションを提供
  - ただしExpressエディションは32bit版のみの提供
- WAS V7から，使用されているIBM JDKで参照圧縮の機能が搭載されたため64bitエディションの欠点（ヒープメモリがより多く消費される）が解消
- 32bit環境で動いていたアプリケーションは、そのまま64bit環境へ移行可能
- ただし、JNI (Java Native Interface) を使っているアプリケーションでは、読み込むライブラリーを64bitにしてからリコンパイルを行う

WAS 6.0/6.1/V7.0から

- J2EE 1.4/Java EE 5アプリケーションの多くは，ほとんど修正することなくそのままWAS V8.0上で実行することが可能です。
- Java EE仕様レベルで非推奨となったものがあります
  - EJB Entity Bean → JPAに移行を推奨
  - JAX-RPC → JAX-WSに移行を推奨
- WAS V6.0からのマイグレーションについては，JDKの仕様変更への対応が必要になるケースがあります。
- WAS V7.0からV8.0は，高いレベルで上位互換性が取られています。
  - WAS V6.1以前のアプリケーションのマイグレーションを行う場合，ターゲットがV7.0であってもV8.0であっても，工数や必要な行程はほぼ同じです。
  - 既存のWAS V7.0向けのマイグレーション情報の大部分は，WAS V8.0へのマイグレーションでもそのまま使用できます。

クラスローダー階層に関する考慮点

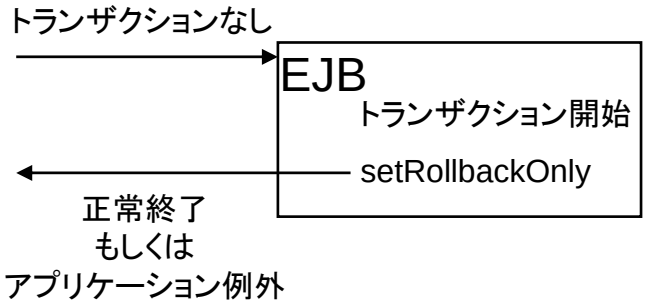
- WAS 3.5以前からの場合，階層化された
  クラスローダーに対する対応が必要
  - 親のクラスローダーからロードされたクラスからは
    子のクラスローダーからのクラスは参照不可
- WAS 4.0以降ではクラスローダー間の
  優先順位の設定に注意が必要
  - 「親が最初（Parent First）」
    WASのクラスがアプリのクラスより優先
  - 「親が最後（Parent Last）」
    アプリのクラスがWASのクラスより優先
- 「親が最初」で動いているアプリケーションは，
  意図せずWASのもつライブラリに依存している
  可能性がある
  - その場合，WASのバージョンアップの際に
    問題が発生する可能性がある
  - 可能な限り「親が最後」で使用する

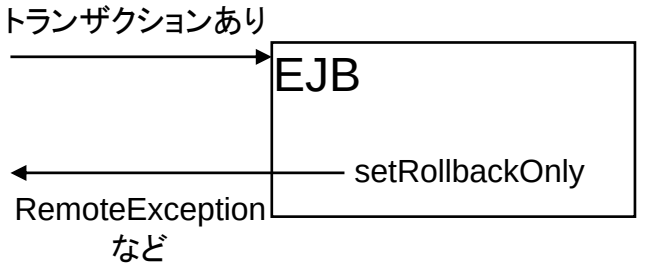


クラスローダーの詳細については以下の文章を参照
「クラスローダーとJ2EEパッケージング戦略を理解する」
http://www.ibm.com/developerworks/jp/websphere/library/java/j2ee_classloader/

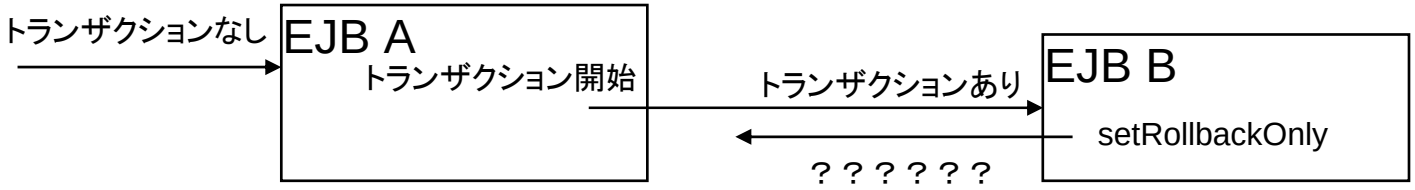
EJB 3.0のsetRollbackOnly動作変更への対応

- EJB内でsetRollbackOnlyが呼ばれたときの動作は，EJB呼び出し前にトランザクションが開始されていたかによって異なります。
  - EJB内でトランザクションが開始されたときには正常に終了し（またはアプリケーション例外がthrowされ）すでにトランザクションが開始されていたときには例外が投げられるとEJB仕様で規定されています





- EJBからEJBが呼ばれたときの挙動が，WAS V6.0/6.1とV7.0以降で異なります
  - WAS V6.0/6.1では，EJB Bからは例外が投げられます
  - EJB 3.0の互換性テストスイートでは，このシナリオでは例外が投げられないことが前提となっているため，WAS V7.0以降ではEJB Bは正常終了ないしアプリケーション例外が投げられるように変更になっています。



- WAS V7.0以降で以前と同じ動作に戻すためには，JVMシステムプロパティ（com.ibm.websphere.ejbcontainer.limitSetRollbackOnlyBehaviorToInstanceFor ）の設定が必要です。
  - 詳細については，以下のURLを参照してください
    http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/tejb_version_specific.html

サイズの大きいEAR/WARのデプロイ

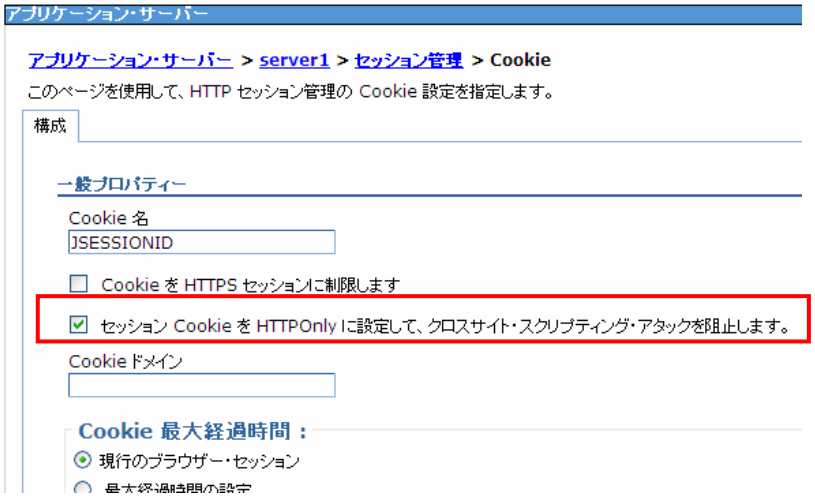
- サイズが大きい（数十Mバイト～数百Mバイト以上）アプリケーションをWAS V7.0/8.0に
  マイグレーションして利用すると，デプロイに非常に時間がかかることがあります
- 原因
  - Java EE 5/6では，構成情報をアノテーションを使用してClassファイルに埋め込めるようになっています
  - デプロイを行う際には，以下の全てのClassファイルについてアノテーションの検索が実行されます
    - WARのWEB-INF/classesの下
    - WARのWEB-INF/libの下のJarファイル
- 対策
  - Java EE 5/6の新機能を使用しない場合は，J2EE 1.4アプリケーションのまま使用する
  - アノテーションを使用しない場合は，デプロイメントディスクリプタ（web.xml）を作成し
    metadata-complete="true"属性をつけてアノテーションの検索を抑止する

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0" metadata-complete="true">
```

  - WARのWEB-INF/libの下のJarファイルを，
    可能な限り，共有ライブラリとしてWARファイル外に配置する
  - 以下の文章も参照して下さい
    http://www.ibm.com/support/docview.wss?rs=180&uid=swg21381764

ハードニングによる影響

- WAS　V8.0から，セキュリティを向上させるためのハードニングとして，
  いくつかのパラメーターのデフォルト値が変更されています
  - この影響でアプリケーションの動作に影響が出る可能性があります
- WASの利用するCookieへのHtttpOnlyオプションのデフォルトの付加
  - WASがセッション維持に使用しているCookie，認証情報を維持しているCookieに，
    デフォルトでHttpOnlyオプションがつくようになりました
    - JSESSIONIDやLTPAなど
  - HttpOnlyが付加されたCookieは，
    JavaScriptからのアクセスが制限されるほか，
    ブラウザによっては
    Webページに埋め込まれた
    Java Appletやフラッシュなどからも
    使用出来なくなります
    - アプリケーションがこれらの用途でCookieに
      アクセスしている場合，動作の不良が
      発生する可能性があります
  - 問題が発生した場合は，
    管理コンソールから設定を外して下さい



Webサービスを使用したアプリケーション

- 多くの場合で，Webサービスプロキシなどの再作成が必要となる
  - Apache SOAP、WebSphere JAVA.SOAPを使用している場合など
- JAX-RPC
  - Java EE 6で，将来廃止予定と規定された
  - 可能な限りJAX-WSに移行する
- Web Service Gateway
  - Apache SOAP channelはSOAP HTTP channelへの置き換えを推奨
  - Web Services Gateway カスタマイズAPIは，JAX-RPCハンドラへの置き換えを推奨
- UDDIで非推奨となった機能
  - UDDI Version 2形式のUDDI RegistryのEJBインターフェース
  - uddi4jv2.jarで提供されているUDDI4J Version 2クラスライブラリ
  - Low-level UDDI Utility Tools (UUT) API
    - BusinessStub、ServiceStubなど
- セキュリティ
  - Apache SOAP実装をベースとしたSOAP-Security(XMLデジタル署名)
    - JSR-109 Web service実装とWS-Security (Web Service Security) 1.0実装への移行を推奨
  - WS-Security (Web Service Security) Draft 13レベルのサポート
    - WS-Security 1.0仕様に準拠した形に書き換える必要がある
    - 例）com.ibm.wsspi.wssecurity.config.KeyLocator は com.ibm.wsspi.wssecurity.keyinfo.KeyLocatorに置き換え

非推奨および削除となった機能・安定化された機能（1）

- 削除された機能（Removed features）
  - そのバージョンから使用できなくなった機能
  - 使用している場合は，必ず移行が必要
- 非推奨となった機能（Deprecated features）
  - 将来のバージョンで削除が予定される機能
  - 基本的には，3年間、または、メジャー・リリースが2つ上がるまでは（そのどちらかの長い期間）サポートされる。
    - たとえば，V5.0.xで非推奨となった機能については，V5.1およびV6.0の間はサポートされるが，その後は製品から削除される予定。
  - 稀に，2リリースよりも早く削除されることも（その場合は、Information Centerに記述）
- 安定化された機能（Stabilized features）
  - 将来のバージョンでの削除は予定されていないが，代替機能があるため機能改善や新機能の追加は行われないもの。
  - 移行の必要はないが，代替機能の検討はおこなう

非推奨および削除となった機能・安定化された機能（2）

- V5.0からV8.0の間に非推奨および削除となった機能の詳細については Information Centerの以下の章を参照

  “非推奨のフィーチャー、安定化されたフィーチャー、および除去された フィーチャー”

  http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/rmig_deprecationlist.html

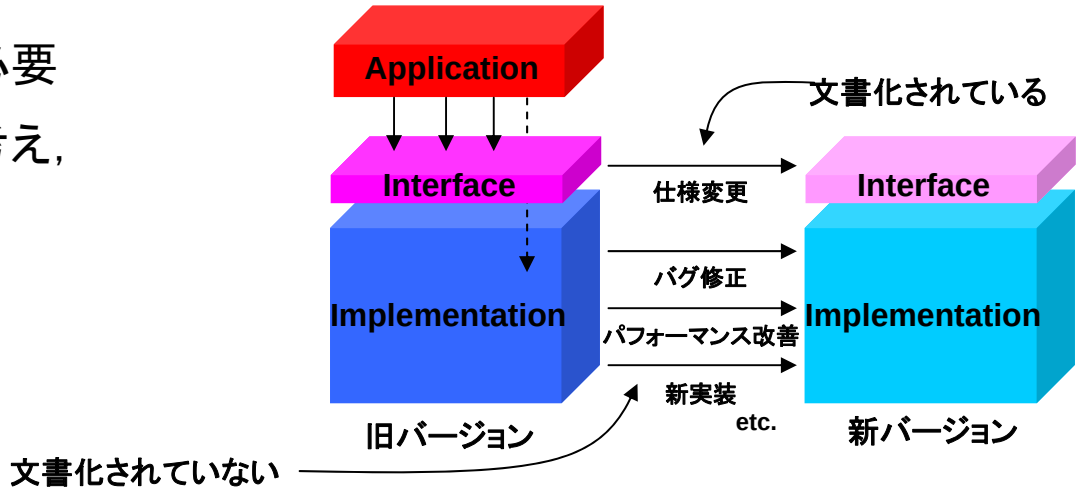
- V7.0から「安定化された機能」のカテゴリが追加されたため，
  以前は「非推奨」となっていた機能の一部が「安定化」されたものもある

その他、移行に当たって対処が必要な例

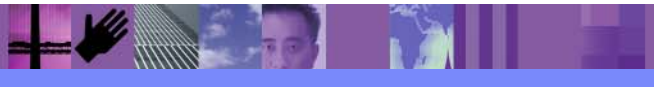
- サーバーAPIの仕様外の挙動の違い（バグが治ったことによる副作用など）
  - JSPでnullオブジェクトを出力したとき
  - CookieがないときのHttpServletRequestでgetCookies()の戻り値
  - JDBCで、executeQueryによる結果を返さないSQLの実行
  - RequestDispatcherでforwardからreturnした後の出力
  - includeされたServlet/JSPでのaddHeader/setHeaderしたとき
  - ユーザースレッドを作成し、スレッド間でDB接続を共有している
  - <jsp:useBean>のclass属性でJavaBean仕様に反するクラスを指定
  - etc.
- 非公開スペックへの依存
  - WASがセットするCookieの文字列への依存
  - HttpSession#getIdで取得される文字列の内容への依存
  - サーバーのローカルディレクトリ構造への依存
  - WebSphere内部クラスの使用
  - etc.

単体テスト・統合テストの重要性

- 前ページのような問題は，事前のコードレビューで発見することが難しい
  - 「仕様の変更」は文書化されているが，
    「仕様外の動作の変更」はほとんど文書化されていない
  - 開発環境のコードチェック機能でも発見できないケースが多い
  - 問題が発生するパターンが無数に考えられ，事前検証で網羅することが不可能
- テストを実行して発見することが最良の手段
  - 最低限必要な修正を加えたら、まずは新環境でテストを行い問題の洗い出しを行う
  - 十分な期間のテストを
    スケジュールしておくことが必要
  - 運用後のパッチ適用なども考え，
    テストの自動化も検討する



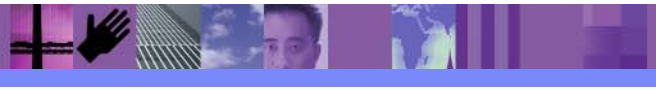
実行環境のマイグレーション



実行環境のマイグレーションの難易度

From＼To	3.0	3.5	4.0	5.0	5.1	6.0	6.1	7.0	8.0
7.0									Easier
6.1								Easier	Easier
6.0							Easier	Easier	Easier
5.1						Easier	Easier	Easier	Easier *
5.0					Easier	Easier	Easier	Easier *	Easier *
4.0				**Difficult**	Difficult	Difficult	Difficult	Difficult	Difficult
3.5			Easier	**Difficult**	Difficult	Difficult	Difficult	Difficult	Difficult
3.0		Easier	Easier	**Difficult**	Difficult	Difficult	Difficult	Difficult	Difficult
2.0	Difficult	Difficult	Difficult	**Difficult**	Difficult	Difficult	Difficult	Difficult	Difficult

* WAS 5.xから8.0は，ツールを使用した自動マイグレーションはできない

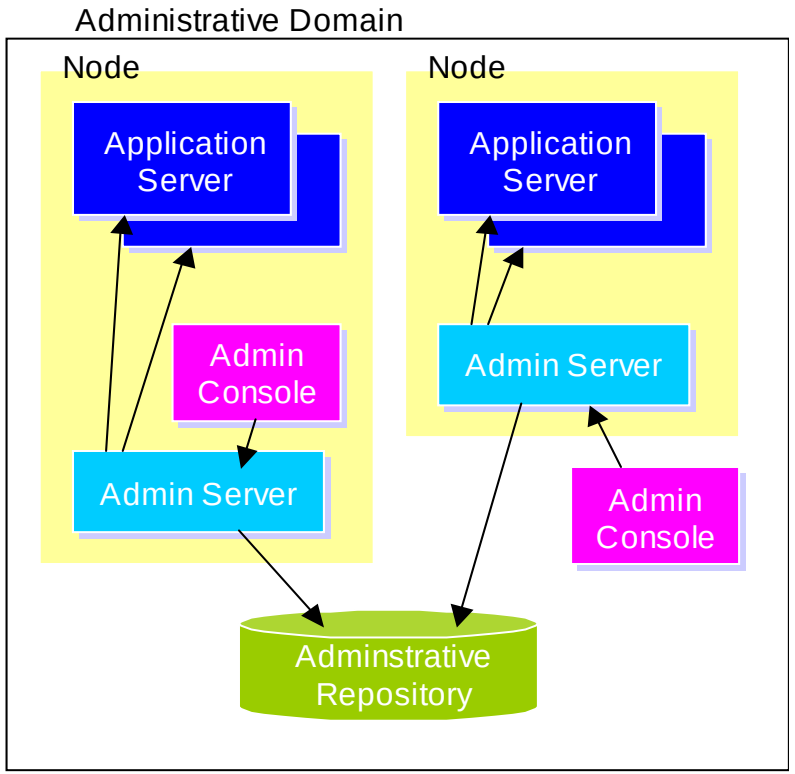


実行環境のマイグレーションの概要

- WAS 5.0で管理概念が大きく変更された
  - それ以前のバージョンとの互換性がほとんどない
    - プロセスの種類，管理コマンドの種類・書式，管理スクリプトの文法，・・・
    - ログの種類，形式，出力内容，・・・
    - 設定情報の格納場所，WASのコンポーネント間の通信方法，・・・
  - WAS 4.0以前からの移行では，基本的にサーバー環境の再設計が必要
- WAS 5.0以降では，管理概念は大きく変わってはいない
  - WAS 6.0以降なら，基本的にはツールによる8.0への移行が可能
- V6.0より前のバージョンからの移行では，
  HA機能への対応，プロファイルへの対応が必要
- V6.1より前のバージョンからの移行では，
  改良されたセキュリティ機能への対応が必要

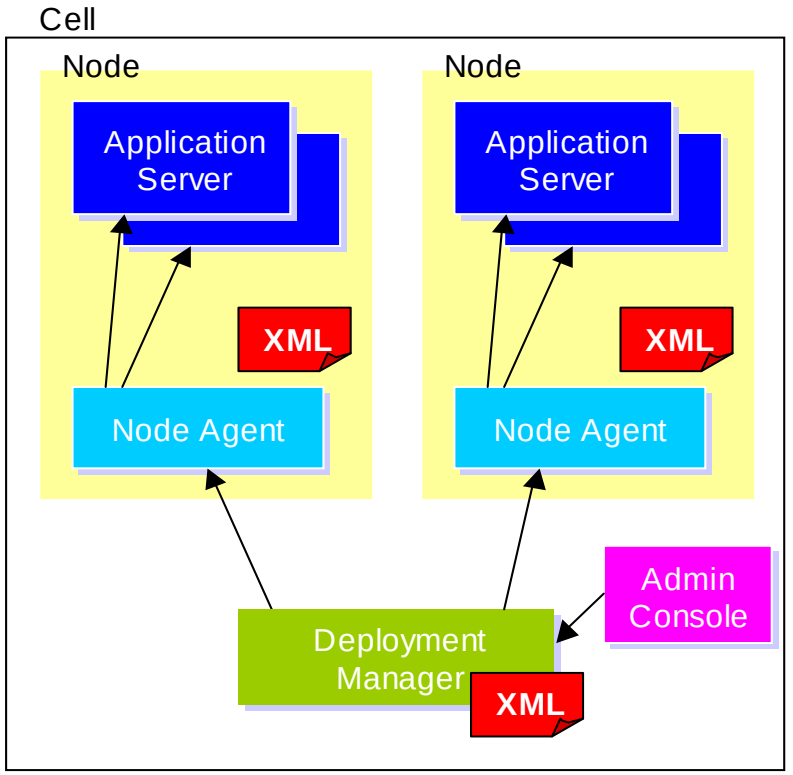
WAS 3.5 / WAS 4.0AEのサーバー管理トポロジー

- 管理モデル
  - 全てのノードに管理サーバー（AdminServer）が必要
  - 各アプリケーションサーバーは管理サーバー経由で起動
  - 構成情報はDBMS上に保存
  - アプリケーションサーバーは管理サーバー経由で構成情報を入手
- 管理ツール
  - Javaアプリケーションベースの管理コンソール
  - XMLConfig
  - WebSphere Control Program (WSCP)



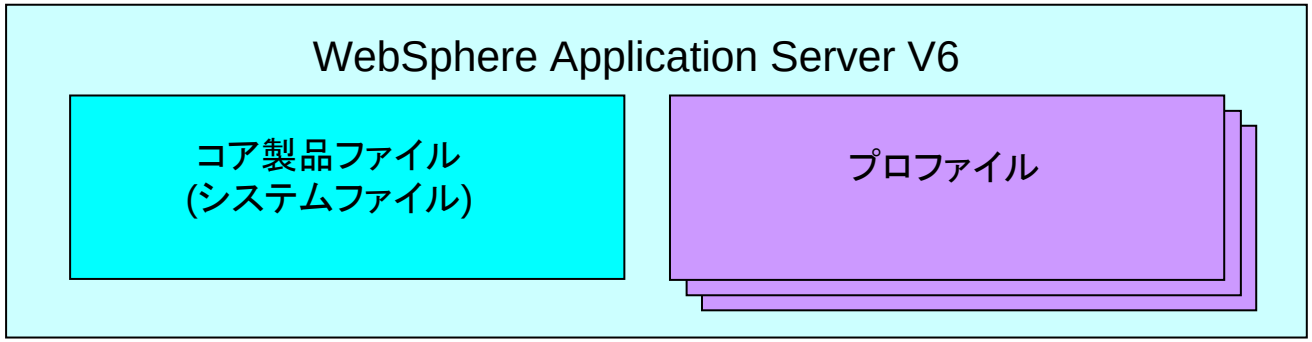
WAS 5.0以降のサーバー管理トポロジー

- 管理モデル
  - –各NodeにNode Agentが必要
  - –構成情報はXMLファイルで保存
  - –各Application Serverは個別に構成ファイルを参照
  - –Deployment Managerがマスターの構成を保持し、Node Agent経由で各Nodeのファイルを同期
  - –Deployment Managerと各Node Agentは独立して起動可能（構成の同期などだけが不可に）
- 管理ツール
  - –Webブラウザベースの管理コンソール
  - –wsadminコマンド



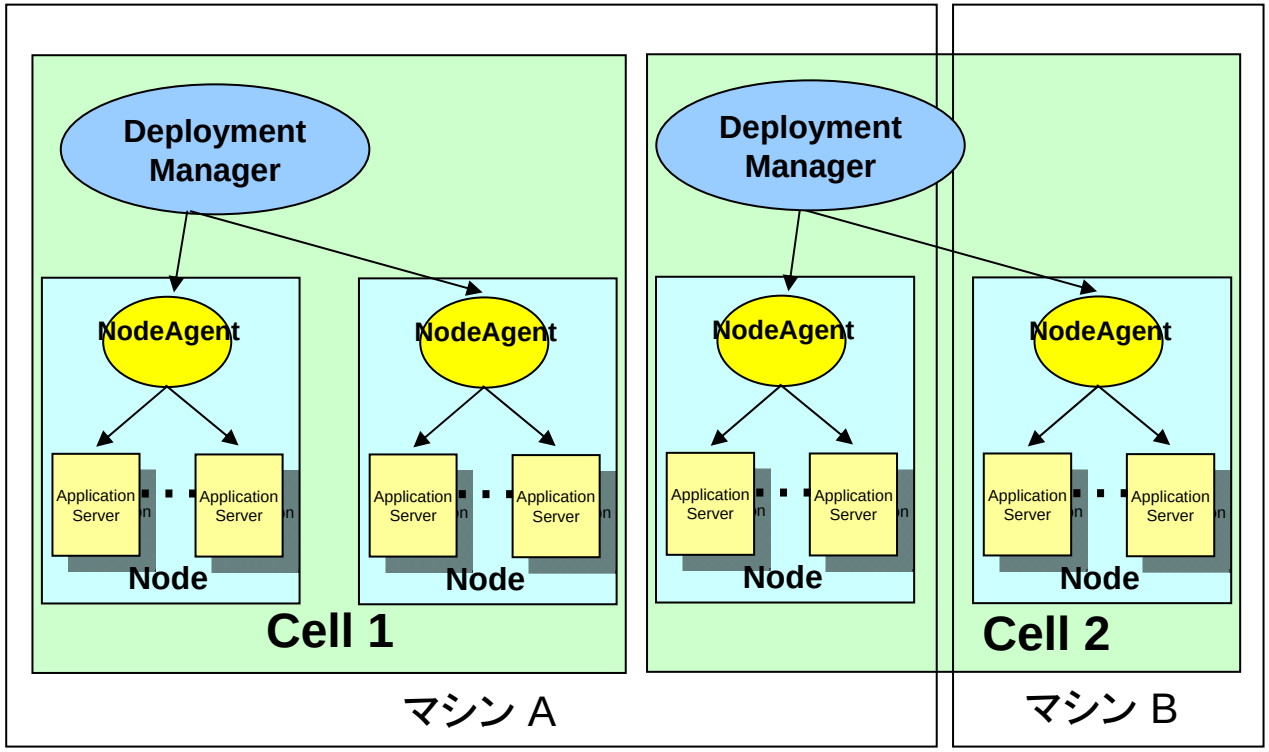
WAS 6.0から:プロファイル機能の新設

- プロファイルはWASのシステムファイルからユーザー固有の環境情報を分離したもの
  - プロファイル構成情報，アプリケーション構成情報，アプリケーション実行ファイル等
  - プロファイル操作コマンド群
  - ログファイル，一時ファイル，トランザクションログなど
- 一つのシステムファイル上で，複数のプロファイルを定義することが可能
- プロファイルには次の3つのタイプを選択可能
  - デプロイメントマネージャー・プロファイル
    - デプロイメントマネージャー（DM）・ノードのためのプロファイル
  - アプリケーションサーバー・プロファイル
    - アプリケーションサーバー・ノードのためのプロファイル
  - カスタム・プロファイル
    - DMが管理するノードのためのプロファイル



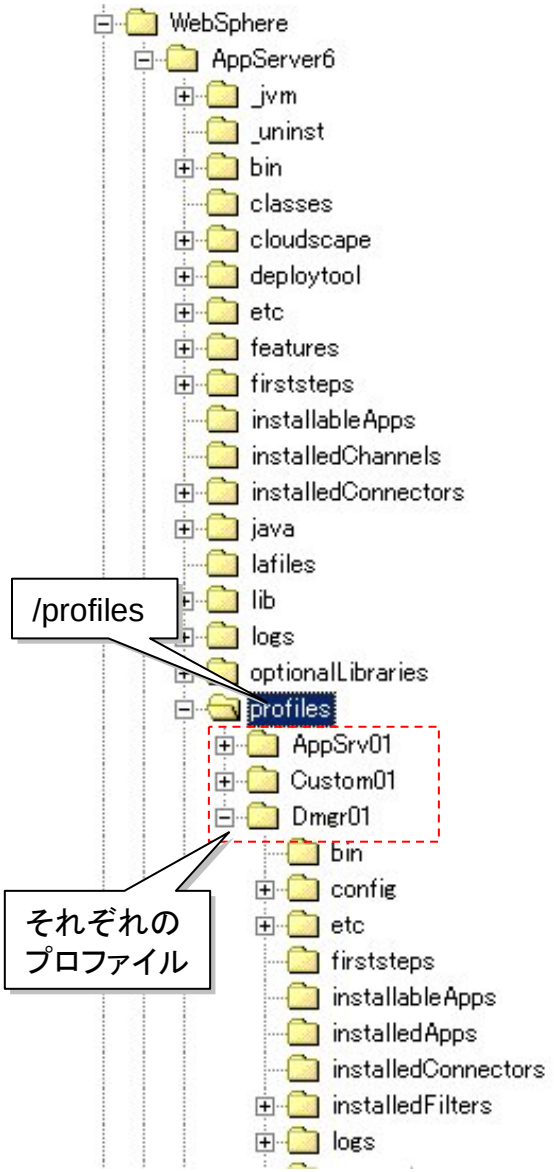
WAS 6.0から:プロファイルの効果

- 柔軟なトポロジー構成
  - 単一のサーバーに二つのノードを構成
  - 複数のCellを管理するDeployment Managerを単一のマシンに集約　　等
- バックアップの作成が容易に



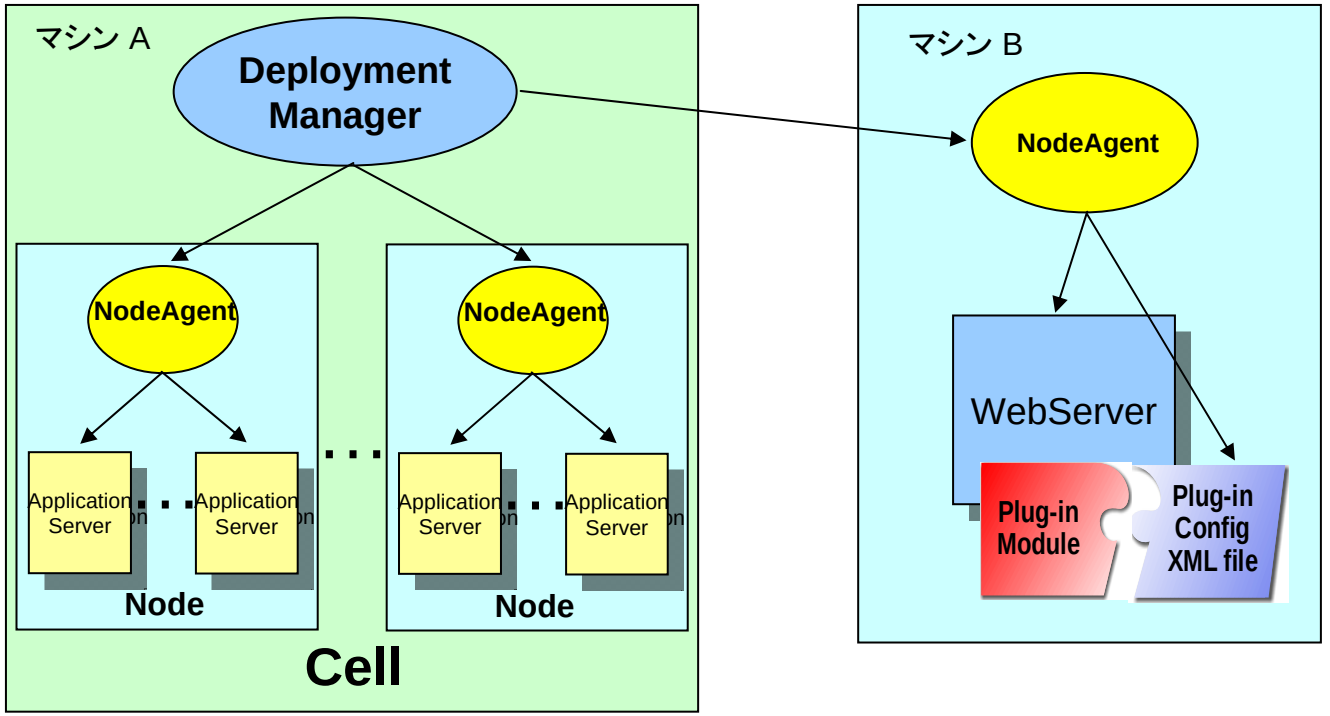
WAS 6.0から:プロファイルへの対応

- コマンド・ログディレクトリの変更
- デフォルトではprofilesディレクトリの下に
  プロファイル毎にディレクトリが作成され
  その下に各種のファイルが格納される
- WAS V5.1以前のバージョンでは,
  導入ディレクトリ直下にあった
  bin/config/etc/logsディレクトリなどは,
  大部分がプロファイル・ルートの下に
  移動になった
- 監視ツールや運用ツールは
  対象ディレクトリの変更が必要



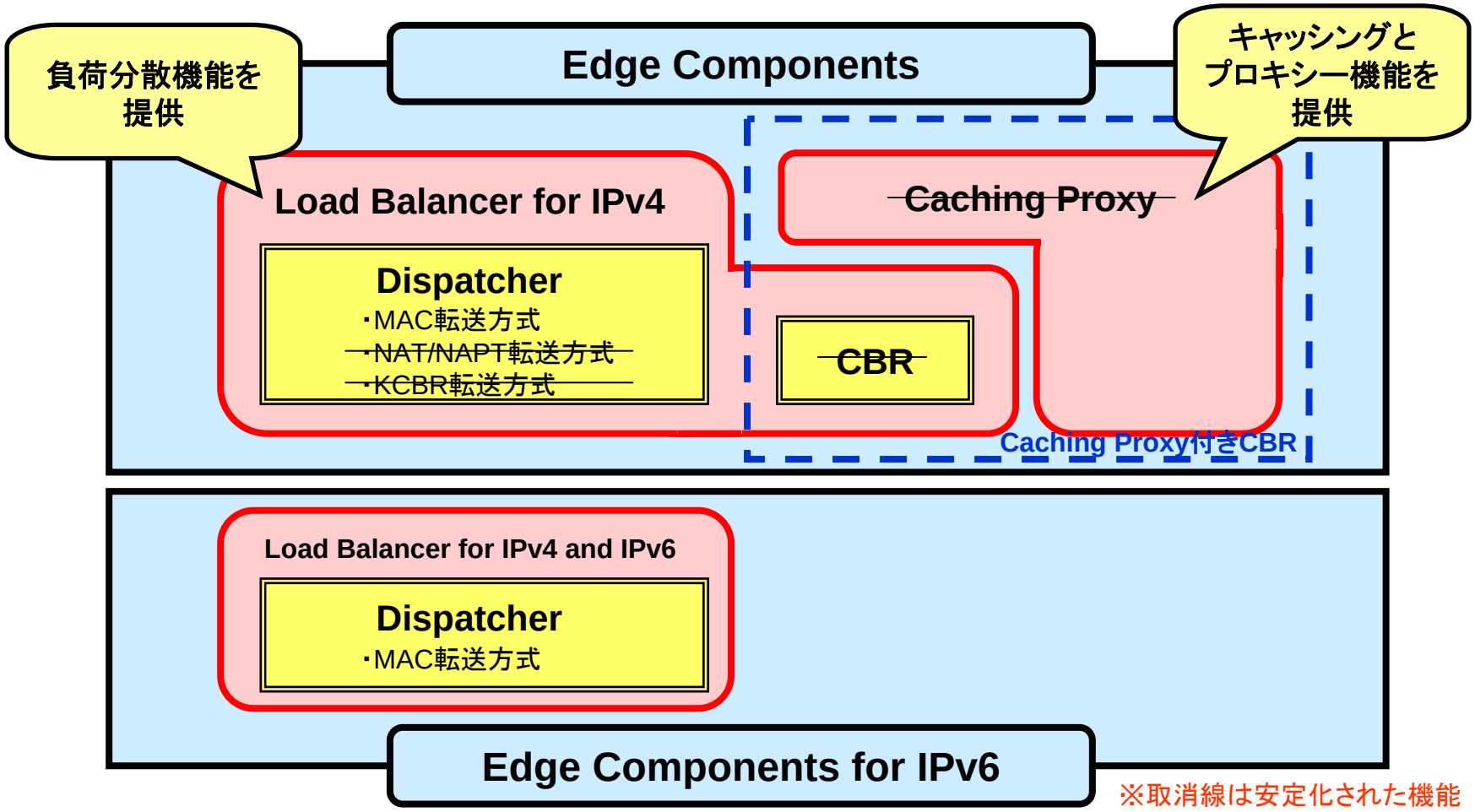
WAS 6.0から：Webサーバーの管理

- リモートのWebサーバーマシン上にノードエージェントを作成
- 以下の管理オペレーションをWAS管理コンソールで実施可能に
  - プラグイン構成の生成
  - プラグイン構成の伝播
  - Webサーバーの開始・停止、ログファイルの表示、構成ファイルの編集
  - WAS内部で使用しているSSL証明書の同期



WAS 6.0から：安定化となったEdge Component

- WAS V6.1/7.0以降，Edge Componentの多くが「安定化された機能」に



WAS 6.0から：Edge Component／LLBからULBへ

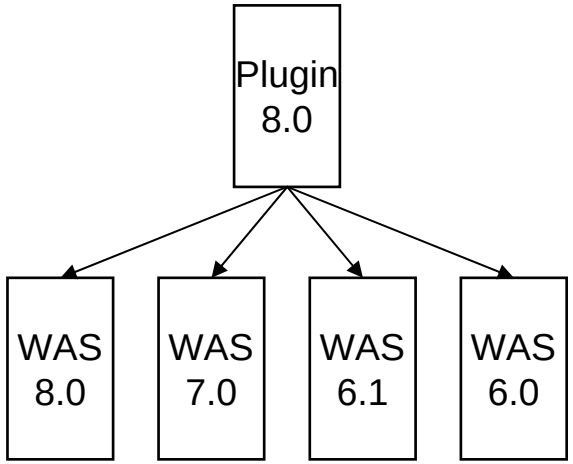
- 従来提供されていたカーネルエクステンションベースの
  Load Balancer for IPv4（LLB：Legac LoadBalancer）は
  「安定化された機能」に
  - 新規のプラットフォーム追加は行われないため，実質的に使用できないケースも
- WAS V6.0から同梱されている
  Load Balancer for IPv4 and IPv6（ULB：Userland LoadBalancer）の使用を推奨
- LLBとULBの機能差に注意が必要
  - LLBではできた「自サーバーへのDispatch」がULBではできない
  - LoadBalancerに，いわゆる「Sorry Server」を兼任させていた場合，
    トポロジーの変更が必要になるケースがある

WAS 6.0から: Edge Component／LLBとULBの違い

		LLB	ULB
①	転送方式	MAC、NAT、KCBR、CP+CBR	MAC転送、カプセル化転送
②	区切り文字	:	@
③	アフィニティー方式	IP Sticky、Cookie	IP Sticky
④	ルール	10種類	2種類
⑤	dscontrol server downコマンド	使用可能	使用不可
⑥	同一筐体内への転送	可能	不可
⑦	テイクオーバー時の接続/アフィニティー情報の引き継ぎ	引き継ぎ可能	引き継ぎ不可
⑧	goActiveスクリプトによるクラスター・アドレスの設定	必須	自動
⑨	serverDownスクリプトの挙動	従来通り	変更
⑩	インストールディレクトリ	<EDGE_ROOT>/lb	<EDGE_ROOT>/ulb
⑪	ワイルドカード・クラスター/ポート	使用可能	使用不可
⑫	ICMPの転送	しない	する

WAS 5.0以降：異なるバージョンの混在（1）

- WebサーバーPluginの下位互換サポート
  - WAS 8.0のPluginは
    WAS 4.0～8.0への振り分けが可能
  - WAS 7.0のPluginは
    WAS 4.0～7.0への振り分けが可能



参照：http://www.ibm.com/support/docview.wss?rs=180&uid=swg21160581

- 異なるバージョンのWASの共存
  - WAS 5.0以降は，
    導入ディレクトリを自由に設定可能
  - 既存のバージョンを認識し，異なるポート番号を自動で割り当てることが可能

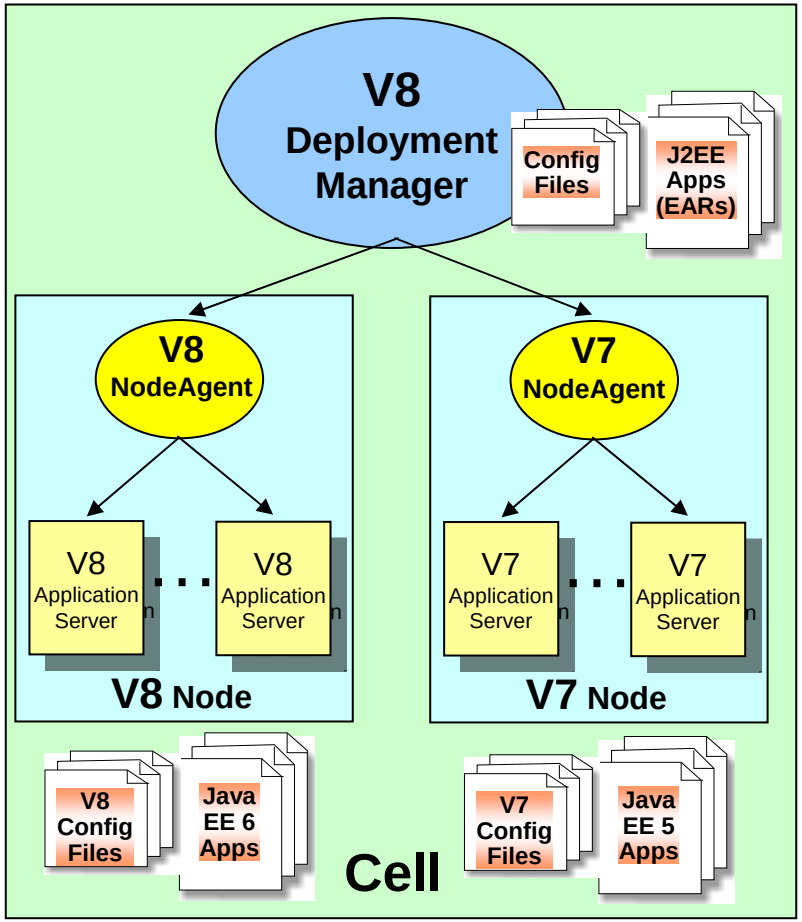
同一筐体の例

/opt/WebSphere/AppServer80/
WAS 8.0の環境

/opt/WebSphere/AppServer70/
WAS 7.0の環境

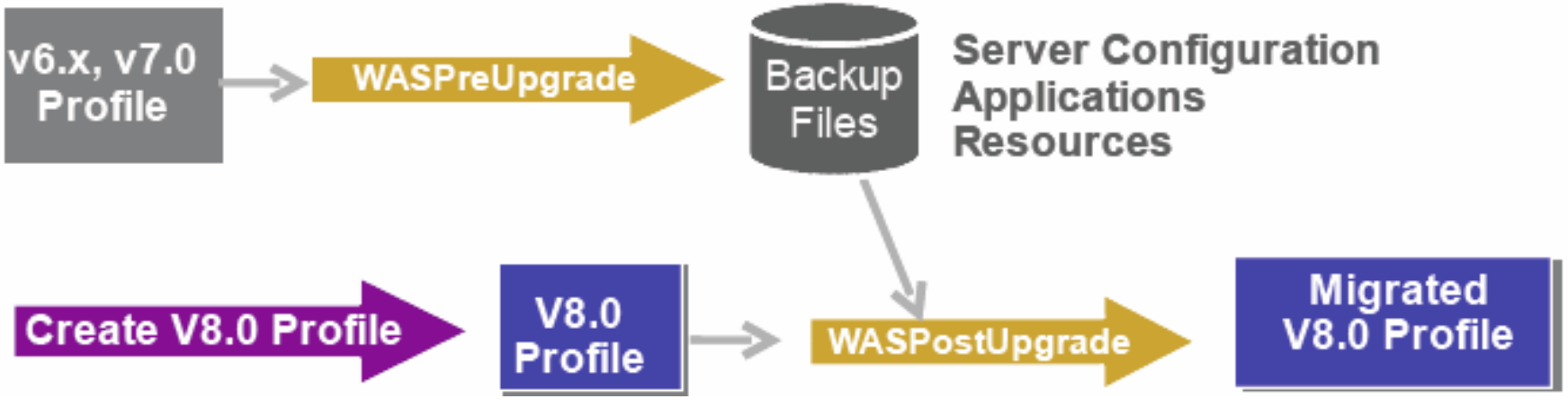
WAS 5.0以降：異なるバージョンの混在（2）

- 1つのCell内で
  V8.0，V7.0，V6.1とV6.0の
  ノードを混在することが可能
- DMはより新しいバージョンで
  なければならない
- 旧ノードは異なるOSでも可能



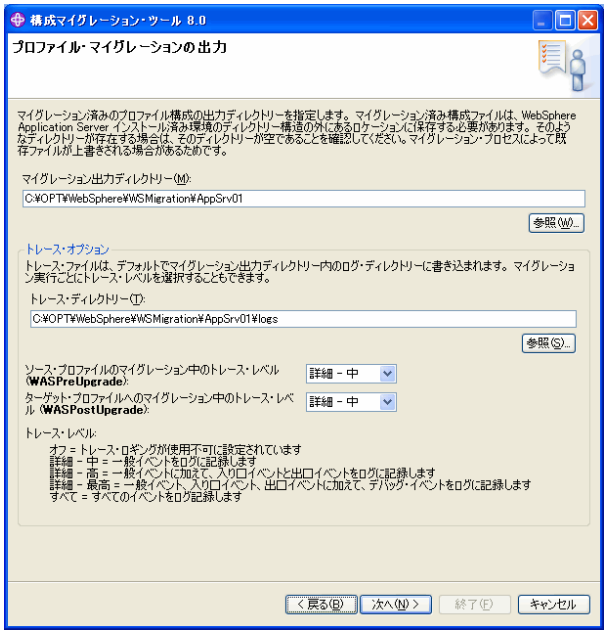
WAS V6.0以降の環境からのマイグレーション

- ツールによるマイグレーションが可能
  - 既存のV6，V7環境から構成情報を抜き出し，V8環境に取り込む
  - 旧環境は削除されていないので，切り戻すことも可能
- Deployment Managerを含むトポロジーでの段階移行では使用が必須
  - V7では，
    500以上のノードを含む環境の移行での使用実績もあり
- 導入されているアプリケーションを含めて移行するかどうかは
  選択することが可能



マイグレーションツール

- コマンドラインツール
  - –WASPreUpgrade
  - –WASPostUpgrade
- GUIツール
  - –構成マイグレーションツール
    - WAS Customization Toolboxから起動
    - コマンドラインツールのフロントエンド・プログラム
    - 指定されたパラメーターで WASPreUpgradeとWASPostUpgradeが実行される



WAS V5環境からのマイグレーションの注意点（1）

- V6.0から、<WAS_ROOT>/profiles/　以下にプロファイル環境が作成される
  - V5.xで<WAS_ROOT>にあったコマンドやログファイルの場所は、各プロファイルのディレクトリー以下に変更となった
  - 現在スクリプトでコマンドの場所やログの場所を指定している場合には修正が必要
    - V5.x での例
      - サーバーの起動　<WAS_ROOT>/bin/startServer.sh
      - サーバーのログ　<WAS_ROOT>/logs/server1/SystemOut.log
    - V6.x での例
      - サーバーの起動　<WAS_ROOT>/profiles/<profile名>/bin/startServer.sh
      - サーバーのログ　<WAS_ROOT>/profiles/<profile名>/logs/server1/SystemOut.log

WAS V5環境からのマイグレーションの注意点（2）

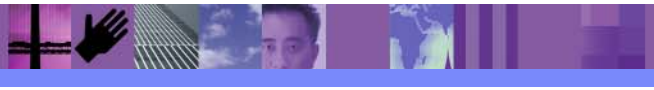
- WAS 6.0からHAマネージャー機能が実装された
  - 使用するポート番号が増加しているので，
    WASのコンポーネント間にF/Wが存在している場合には，
    通信の許可を追加する
  - core groupの設定を行う
    - core group内のJavaプロセス間では，
      ハートビートによる互いの死活監視が行われている
    - core groupのサーバー数が20～30を超えるような場合，
      パフォーマンス上の問題が発生する可能性が高いので，
      適切にcore groupを分割する

ツールによるマイグレーションの注意点（1）

- アプリケーションが多数導入された大規模なCellのマイグレーションでは，通信タイムアウトによりWASPostUpgradeの実行に失敗することがある
  - <WAS Root>/properties/ssl.client.props を編集し，com.ibm.SOAP.requestTimeout の値をデフォルトの180秒から10倍程度に増やす
- 旧環境の動いているOSがWAS V8でサポートされない場合
- 異なる物理筐体にWAS V8を導入する場合
  - WAS V8環境にて<AppServer_Root>/bin/migration/bin/createRemoteMigrJar.bat(sh)を実行
    - 作成されたJARファイルを旧環境にコピーして，それを展開するとbinディレクトリ配下にWASPreUpgradeコマンドがあります。
  - ここからWASPreUpgradeコマンドを実行して現環境をバックアップする
  - 新しいOSを導入した環境にWAS V8をセットアップする
  - バックアップされたファイルを新環境に転送し，WASPostUpgradeを実行

ツールによるマイグレーションの注意点（2）

- Cell環境のマイグレーション
  - 移行後のセル名・ノード名は，移行前と同じにしておく必要がある
- SSL証明書および鍵管理
  - 使用していたSSL証明書はそのままインポートされる
  - WAS 7.0から採用された階層付きの証明書の構成にはならない
    - WAS 6.1までのデフォルトの環境では自己署名証明書が使用されている
    - 必要であれば，wsadminのconvertSelfSignedCertificatesToChainedタスクを使用して，自己署名証明書を階層付き証明書に変換する
  - 変換しない場合，証明書の自動更新によるアクセス障害などに注意する
    - WAS 7.0では，署名者証明書（Root証明書）がデフォルトで15年間の有効期限を持つため，個人証明書が1年ごとに更新されても，証明書の検証失敗による接続障害は起こらない
- 非rootユーザーで導入された環境
  - いったんrootユーザーのプロファイルへマイグレーションする
  - その後，非rootユーザーへファイルの所有権・アクセス権を変更する



管理スクリプト

- コマンドライン管理ツールwsadminでは
  二つのスクリプト言語が使用可能
  - Jacl
  - Jython
- 既存のJaclスクリプトも移行の必要はなし
  - Jaclスクリプトは「安定化された機能」に変更となり
    新規の機能追加は行われないが
    将来のバージョンでの廃止も予定されていない
- 新規のスクリプト開発はJythonでおこなう

パフォーマンスチューニングについて

- WASのパフォーマンスにかかわる内部実装やパラメーターは変更になっている部分が多い
  - WAS 5.0
    - Databaseアクセスの内部実装の変更
      （デフォルト分離レベルの変更，PreparedStatementキャッシュの実装等）
    - HttpSessionマネージャーの機能拡張
  - WAS 6.0
    - HAマネージャーの実装，DRSによるサーバー間情報共有
    - HTTPトランスポートでのチャネルフレームワークの採用
  - WAS 6.1
    - 新しいIBM JVM（J9VM）の採用
  - WAS 7.0
    - JITの高速化／POWER7環境への対応／ 64bit版JVMにおける参照圧縮機能
  - WAS 8.0
    - 世代別GCのデフォルト化／Subpool GC Policyの非推奨　　等
- 通常の環境であればデフォルトのパラメーターで運用することも可能
- 高い負荷のかかるシステムについては，移行後のシステムにおいて，
  あらためてパフォーマンスチューニングを行うことが必要になるケースも
  - サーバー環境の移行に当たっては負荷テストの実施が必要不可欠

参照リンク・リファレンス

マイグレーションに関する情報（1）

- WAS V8へのマイグレーションガイド（この資料）
  - http://www.ibm.com/developerworks/jp/websphere/library/was/was8_migration/
- WASの前提条件
  - http://www.ibm.com/software/webservers/appserv/was/requirements/
- WAS V8.0 Information Center (オンライン・マニュアル)
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp
    - マイグレーション、共存、および相互運用
      http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6topmigrating.html
    - 非推奨のフィーチャー、安定化されたフィーチャー、および除去されたフィーチャー
      http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/rmig_deprecationlist.html
- Redbook 「WebSphere Application Server V7 Migration Guide」
  - http://www.redbooks.ibm.com/abstracts/redp4635.html
- WebSphere Application Server V8.0 導入ガイド
  - http://www.ibm.com/developerworks/jp/websphere/library/was/was8_install/
- Knowledge Collection: Migration planning for WebSphere Application Server
  - http://www.ibm.com/support/docview.wss?rs=180&uid=swg27008724

マイグレーションに関する情報（2）

- Rational トップページ
  - http://www.ibm.com/jp/software/rational/
- developerWorks Japan Rational (Rational技術情報）
  - http://www.ibm.com/developerworks/jp/rational/
    - RAD V7.0 マイグレーション・ガイド
      http://www.ibm.com/developerworks/jp/rational/library/am/rsdp/rad/7/migration_guide/index.html
    - RAD V7.5 へのマイグレーション
      http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.etools.rad.migration.doc/topics/c_migrationoverview.html
    - Rational Functional Tester 記事一覧
      http://www.ibm.com/developerworks/jp/rational/library/qm/index.html#rft
    - Rational Performance Tester 記事一覧
      http://www.ibm.com/developerworks/jp/rational/library/qm/index.html#rpt
    - Rational AppScan 記事一覧
      http://www.ibm.com/developerworks/jp/rational/library/qm/index.html#appscan
- Rational製品評価版の入手方法およびダウンロード先
  - http://www.ibm.com/jp/software/rational/download

Servlet API 2.0（JSDK 2.0）の移行

- Servlet API 2.0（JSDK 2.0）の移行
  - com.sun.server.http.HttpServiceRequest, com.sun.server.http.HttpServiceResponseは移行が必要
    - com.sun.server.http.HttpServiceRequest.setAttribute() を javax.servlet.http.HttpServletRequest.setAttribute() に移行する
    - com.sun.server.http.HttpServiceResponse.callPage() を javax.servlet.RequestDispatcher に移行する

移行前

```
// Code example of using the old HttpServiceResponse.callPage()
public void doGet (HttpServletRequest req, HttpServletResponse res)
     throws ServletException, IOException
{
  String message = "TIHS is a test";
  ((com.sun.server.http.HttpServiceRequest)req).setAttribute("message", message);
  ((com.sun.server.http.HttpServiceResponse)res).callPage("/Update.jsp", req);
}
```

移行後

```
// Code example of using the new RequestDispatcher
public void doGet (HttpServletRequest req, HttpServletResponse res)
     throws ServletException, IOException
{
  String message = "TIHS is a test";
  req.setAttribute("message", message);
  RequestDispatcher rd =
    getServletContext().getRequestDispatcher("/Update.jsp");
  rd.forward(req, res);
}
```

JSP 0.91の更新

- JSP 1.0以降と書式が完全に異なるため，全面的な書き換えが必要

標準 JSP タグ		
説明	**JSP 0.91**	**JSP 2.0**
別の JSP ファイルまたはサーブレットを起動するタグ	<SERVLET>	<jsp:forward> または <jsp:include>
JavaBean を起動またはインスタンス化するタグ	<BEAN>	<jsp:useBean>
クラス全体のメソッドおよび変数	<SCRIPT>	<%! code %>
JSP ディレクティブで使用されるタグ (構文は若干変更されています)	JSP 0.91 ディレクティブを参照	JSP 2.0 ディレクティブを参照
インライン Java コード・スクリプトレットで使用されるタグ (構文は若干変更されています)	JSP 0.91 スクリプトレットを参照	JSP 2.0 スクリプトレットを参照
データベース・アクセスのための IBM 拡張 タグ		
説明	**JSP 0.91**	**JSP 2.0**
データベースからの変数データを挿入するタグ	<INSERT>	式言語 (EL)を使用する
データベース結果照会データ・セットを繰り返すために使用するタグ	<REPEAT>	<c:forEach>
データベースへの接続に使用するタグ	<DBCONNECT>	<sql:setDataSource>
データベースにアクセスするための ユーザー ID およびパスワードを表すタグ	<USERID> および <PASSWD>	<sql:setDataSource>のuser,password属性
データベース照会を実行するために使用するタグ	<DBQUERY>	<sql:query>
データベースを変更するために使用するタグ	<DBMODIFY>	<sql:update>

PageListServletのHttpServletへの移行

- .servletファイル内の設定情報は，全てサーブレット初期化パラメータへ移行させる
- サーブレットクラスの継承元をjavax.servlet.http.HttpServletに変更し，PageListServletの独自メソッドをユーザーコードで実装する
  - HttpServletを継承した独自のクラスを作成し，これを継承するのが効率的
- 移行例：

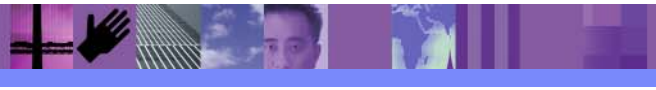
メソッド	移行前	移行後

callPage	callPage(getPageNameFromreq(req), req, res);	String nextPage = req.getParameter("nextPage");
		if(nextPage == null)
		{
		  nextPage = getInitParameter("default_page");
		}
		RequestDispatcher dispatch = req.getRequestDispatcher(nextPage);
		dispatch.forward(req, res);

callPageNamed	callPageNamed("customPageName", req, res);	RequestDispatcher dispatch =
		req.getRequestDispatcher("customPageName");
		dispatch.forward(req, res);

setRequestAttribute	setRequestAttribute(String name, Object o, req);	req.setAttribute(String name, Object o);

getParameter	String name = getParameter(req, "name", true, true, false, "alt_name");	String name = (String)req.getParameter("name");
		if (name == null)
		{
		  name = "alt_name";
		}



IBM Connection ManagerのJDBCへの移行

- JDBCのDataSourceをつかったコネクション取得に書き換える

	IBM Connection Manager	JDBC
Import文	import java.sql.*; import com.ibm.servlet.connmgr.*;	import java.sql.*; import javax.sql.*; import javax.naming.*;

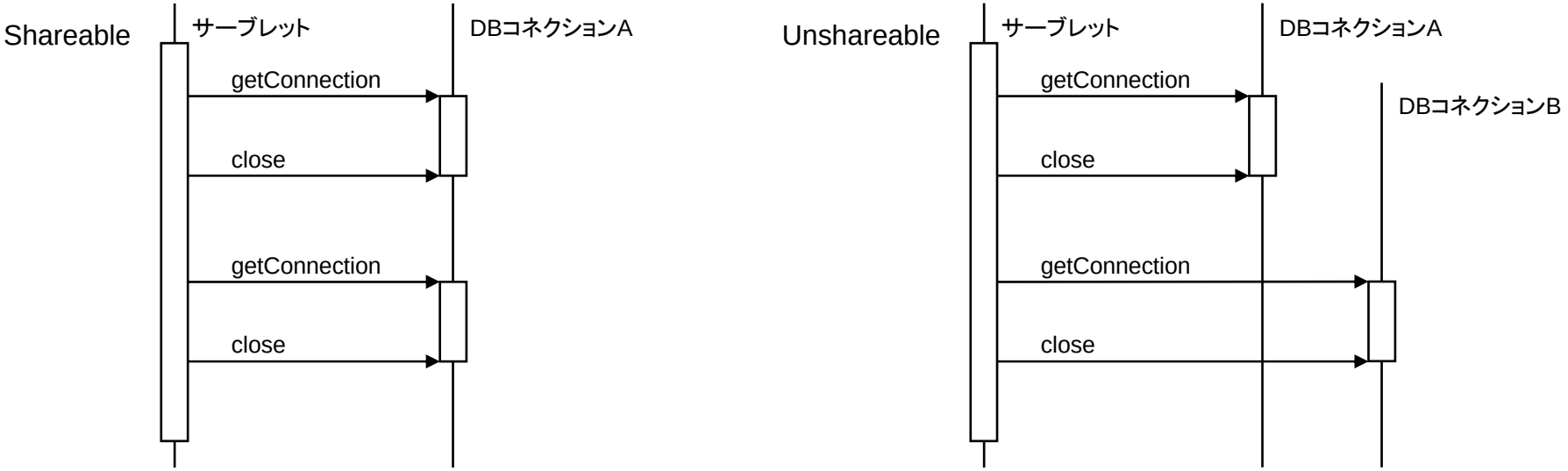
初期化	// create specification for desired connection
	IBMConnSpec spec =
	new IBMJdbcConnSpec("poolname",
	true,"COM.ibm.db2.jdbc.app.DB2Driver",
	"jdbc:subprotocol:database","userid", "password");
	// establish connection manager access to use its facilities
	IBMConnMgr connMgr = IBMConnMgrUtil.getIBMConnMgr();
		// create parameter list to access naming system
		Hashtable parms = new Hashtable();
		parms.put(
		Context.INITIAL_CONTEXT_FACTORY,
		"com.ibm.websphere.naming.WsnInitialContextFactory");
		// access naming system
		Context ctx = new InitialContext(parms);
		// get DataSource factory object from naming system
		DataSource ds = (DataSource)ctx.lookup("jdbc/sample");

コネクションの取得	// use spec to get connection manager connection
	IBMJdbcConn cmConn = (IBMJdbcConn)connMgr.getIBMConnection(spec);
	// use connection manager connection to get data server connection
	Connection conn = cmConn.getJdbcConnection();
		// use DataSource factory to get data server connection
		Connection conn = ds.getConnection("userid", "password");

コネクションのクローズ	// release connection manager connection
	cmConn.releaseIBMConnection();
	// do not issue conn.close();
		// placing it back in the connection pool
		conn.close();

リソース参照の共用可能属性

- 共用可能（Shareable）に設定されたリソースは，サーブレット/EJB実行中に複数回リソース取得が行われた場合，同一のリソースインスタンスが返される
- 共用不能（Unshareable）に設定されたりソースは，リソース取得を行うたびに，別のインスタンスが返される

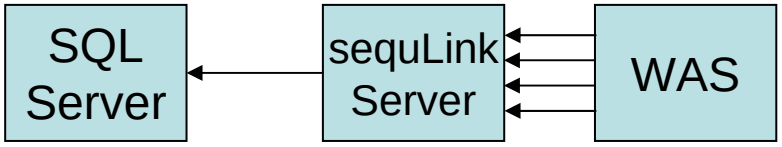


- アプリケーション中で複数のリソースをプールから同時に取得し，
  それらが異なるインスタンスであることを前提としているアプリケーションは，
  Shareableの設定では正常に動かないことがある
  - DBへのConnectionを二つ取得し，それぞれ別のトランザクションでSQLを並行実行しているような場合

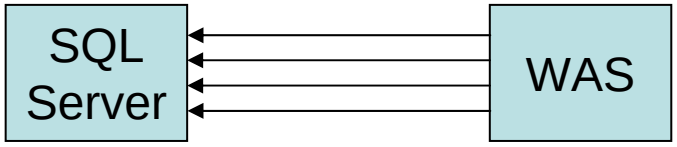
データベースアクセスについての注意点

- 分離レベルの設定
  - WAS 4.0以前とWAS 5.0以降では，デフォルトの分離レベルが変更に
    - デフォルトを使用しているアプリケーションでは，デッドロックなどの問題が発生することも
  - デフォルトは使用せず，アプリケーションで必要な分離レベルを明示的に指定する
    - リソース参照の分離レベル設定
    - Connection#setTransactionIsolation()メソッド
- 使用するJDBCドライバを変更することによる挙動の違い
  - 例）「DataDirect SequeLink JDBC 5.5 (type 3)」から
    「DataDirect Connect JDBC 3.5 (type 4)」に変えた場合
    - DB上でのSQLの同時実行が格段に増える
    - 並行実行性に問題あるSQLがある場合，Type 3環境では問題なく動いてしまうが，Type 4環境に移行したとたん動かなくなる，というようなケースも

- DataDirect sequLink type 3 JDBC driver for MS SQL Server



- DataDirect ConnectJDBC type 4 driver for MS SQL Server等

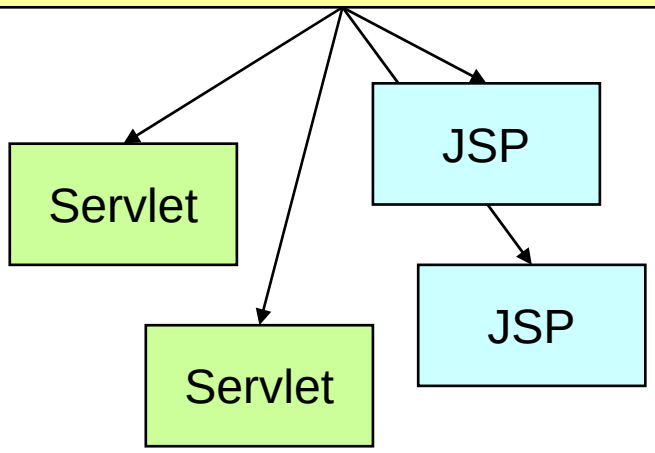


リクエストパラメーターのエンコード

- 「VALUE=日本語」というパラメーターがブラウザから送られてきたとき
  - Shift_JISでは
    VALUE=%93%FA%96%7B%8C%EA
  - UTF-8では
    VALUE=%E6%97%A5%E6%9C%AC%E8%AA%9E
  - エンコードを指定しないと,
    正しくパラメーターを受け取ることができない
- WAS 4.0以前(J2EE 1.2以前)では,
  サーバー構成で指定していた
  - 「default.client.encoding」システムプロパティで設定
- WAS 5.0以降(J2EE 1.3以降)では, アプリケーションでエンコードを指定できるようになった
  - 「ServletRequest#setCharacterEncoding」メソッドで指定
- サーブレットフィルターを使用すると, 一括して指定することが可能

ServletFilter

```
public void doFilter(
    ServletRequest req,
    ServletResponse resp,
    FilterChain chain)
throws IOException, ServletException
{
    req.setCharacterEncoding("UTF-8");
    chain.doFilter(req, resp);
}
```



V5.0/5.1で非推奨となった機能

- V5.0
  - Web Service GatewayでのApache SOAP channel
    - SOAP HTTP channelへの置き換えを推奨
  - Apache SOAP、WebSphere JAVA.SOAP
    - soap.jar、wssoap.jarで提供されているクラス・ライブラリー
  - 両者ともWAS V8.0で削除された機能に
- V5.1
  - Common Connector Framework
    - ccf.jar、ccf2.jar、recjava.jar、eablib.jarで提供されるライブラリ群
    - J2Cへの置き換えを推奨
    - V6.1でCCFは削除された機能に
  - Web Services Gateway
    - Web Services Gateway カスタマイズAPI
      - JAX-RPCハンドラへの置き換えを推奨

V6.0で非推奨となった機能（1）

- JSPのtsx拡張タグ
  - 以下のIBM独自拡張タグが非推奨に
  - JSTL（JSP Standard Tag Library）へ移行を推奨
    - WAS 6.0では，<tsx:repeat>タグなど，挙動が変更となっているタグがあるため注意が必要（PK26741/PK31208を参照）

tsx tag	JSTL tag
tsx:repeat	c:forEach
tsx:dbconnect	sql:setDataSource
tsx:userid	sql:setDataSourceタグのuser属性
tsx:passwd	sql:setDataSourceタグのpassword属性
tsx:dbquery	sql:query
tsx:dbmodify	sql:update
tsx:getProperty	式言語 (EL)を使用する

- databeans.jarで提供されるData Access Beans
  - Service Data Objects (SDO)への移行を推奨
- JRas Extensions API
  - java.util.logging パッケージ (JSR47)への移行を推奨

V6.0で非推奨となった機能（2）

- J2EE Connector Architecture (J2C)の以下のメソッドやインターフェース
  - com.ibm.ws.management.descriptor.xml.ConnectionFactory.xmlのgetPoolContents/getAllPoolContentsメソッド
    - showPoolContents/whoAllPoolContentsに置き換え
  - com.ibm.websphere.j2c.ConnectionManagerインターフェース
    - J2C 1.5のLazyAssociatableConnectionManagerインターフェースに置き換え
  - com.ibm.websphere.j2c.ConnectionEventListenerインターフェース
    - J2C 1.5のLazyEnlistableConnectionManagerインターフェースに置き換え
- CMPのメソッド・レベル・アクセス・インテント
  - beanレベルのアクセス・インテントを使用する
- データソースにおけるコンテナ管理認証別名とマッピング構成別名
  - リソース参照認証設定を使用する



V6.0で非推奨となった機能（3）

- セキュリティ
  - Apache SOAP実装をベースとしたSOAP-Security(XMLデジタル署名)
    - JSR-109 Web service実装とWS-Security (Web Service Security) 1.0実装への移行を推奨
  - WS-Security (Web Service Security) Draft 13レベルのサポート
    - WS-Security 1.0仕様に準拠した形に書き換える必要がある
    - 例）com.ibm.wsspi.wssecurity.config.KeyLocator は com.ibm.wsspi.wssecurity.keyinfo.KeyLocatorに置き換え
  - Security Authentication Service (SAS) IIOP プロトコル
    - Common Secure Interoperability Version 2 (CSIv2)へ移行
  - Security Authentication Service (SAS) CORBAセキュリティー・プログラミングAPI
    - Java Authentication and Authorization Service (JAAS)へ移行
- UDDI
  - UDDI Version 2形式のUDDI RegistryのEJBインターフェース
  - uddi4jv2.jarで提供されているUDDI4J Version 2クラスライブラリ
  - Low-level UDDI Utility Tools (UUT) API
    - BusinessStub、ServiceStubなど

V6.0で非推奨となった機能（4）

- com.ibm.websphere.servlet.filterパッケージのAPI
  - 標準Servlet APIのFilter機能への移行を推奨
- MIMEフィルターによるサーブレット起動
  - WAS 3.5より実装されているIBM独自機能
  - 通常のServlet Filter機能への移行を推奨
  - 非推奨ではなく、使用できなくなっている
- com.ibm.websphere.product 以下のパッケージのいくつかのクラス・メソッド
  - WASのバージョンや導入ディレクトリなどの情報を取得するクラス群
  - いくつかの仕様変更が加えられている
- WAS 4.0形式のPMI Client API
  - JMXをベースとしたWAS 5.0以降のPMI Client APIへの移行を推奨
- com.ibm.websphere.runtime.ServerName のinitializeメソッド
  - V6.0.2で非推奨に
  - WASのランタイムが使用するものなのでアプリケーションからCallしないこと

V6.1で非推奨となった機能（1）

- JVMのcom.ibm.websphere.sendredirect.compatibility プロパティー
  - アプリケーションを修正して、Servlet 2.4の仕様に合わせる
    / で始まる場合は、Webアプリケーションのコンテキストルートからの相対パスではなく、Webサーバー・ルートからの相対パスを指定すること
- 次のクラスによるWebSphere Studio tools runtimeのサポート
  - com.ibm.webtools.runtime.AbstractStudioServlet
  - com.ibm.webtools.runtime.BuildNumber
  - com.ibm.webtools.runtime.NoDataException
  - com.ibm.webtools.runtime.StudioPervasiveServlet
  - com.ibm.webtools.runtime.TransactionFailureException
  - com.ibm.webtools.runtime.WSUtilities
    - アプリケーションを再構築して、J2EE標準のコーディングにする
- 次のクラスを含むPageList Servlet custom extension
  - com.ibm.servlet.ClientList
  - com.ibm.servlet.ClientListElement
  - com.ibm.servlet.MLNotFoundException
  - com.ibm.servlet.PageListServlet
  - com.ibm.servlet.PageNotFoundException
    - アプリケーションを再構築して、J2EE標準のコーディングにする

V6.1で非推奨となった機能（2）

- com.ibm.websphere.rsadapter.DataStoreHelperMetaData クラスの setDatabaseDefaultIsolationLevel(int) メソッド
  - public final void setDatabaseDefaultIsolationLevel (int helperDefaultLevel, int cusDefinedWasDefaultIsoLevel) に修正
- Mediation Framework の com.ibm.websphere.sib.mediation.handler.SIMessageContextException クラス、com.ibm.websphere.sib.mediation.messagecontext.SIMediationBeanMessageContext インターフェース
  - .SIMessageContextException クラスは、com.ibm.websphere.sib.mediation.handler.MessageContextException クラスに修正
  - SIMediationBeanMessageContext インタフェースも同等のインターフェースに修正（WASでは提供なし）
- com.ibm.etools.util プラグイン、logutil.jar ファイル
  - Java Logging APIやTPTPのサービスへ移行

V6.1で非推奨となった機能（3）

- WAS 6.0.x以前で使用されていたメッセージIDのフォーマット
  - convetlogコマンドを使用
- JVMPI (JVM Profiler Interface)
  - JVMTI (JVM Tool Interface) へ移行する
- DB2 Legacy CLI-based Type 2 JDBC Driver provider
  - DB2 Universal JDBC Driver Providerを使用する
- Cloudscape datasource Helper
  - マイグレーション・ユーティリティーが自動的にDerbyのHelperに修正
- データソースのカスタム・プロパティー
  - dbFailOverEnabled → validateNewConnection に置き換え
  - connRetriesDuringDBFailover → validateNewConnectionRetryCount に置き換え
  - connRetryIntervalDuringDBFailover →validateNewConnectionRetryInterval に置き換え
- generic SQL databaseに対するCMPのデプロイ
  - WASで使用するデータベース・ベンダー / バージョンを指定してデプロイする

V6.1で非推奨となった機能（4）

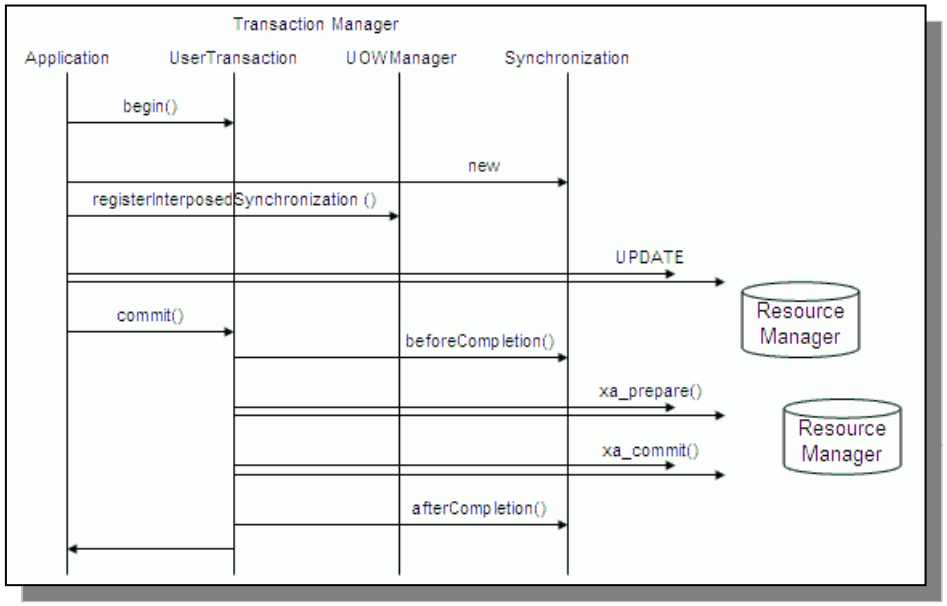
- WebContainer のstartTransports、stopTransports、restartWebApplication MBean
  - Channel Frameworkが提供するTransportChannelService MBeanを使用する
- トランザクション・サービスのDISABLE_FILE_LOCKING カスタム・プロパティー
  - 管理コンソールの「トランザクション・サービス」で「ファイル・ロックを使用可能にする」のチェックを外す
- SWAN (Simple WebSphere Authentication Mechanism)
  - LTPAへ移行する
- Edge ComponentsのCBR、NAT転送、KCBR転送など、MAC転送以外のコンポーネントや機能
  - MAC転送と IBM HTTP Server プラグイン、または、Secure Proxy serverに移行する
- Edge Componentsのキャッシング・プロキシー
  - MAC転送と IBM HTTP Server プラグイン、または、Secure Proxy serverに移行する

WAS V7で非推奨となった機能（サーブレット関連）

- Proxyオブジェクト
  - com.ibm.websphere.servlet.request.HttpServletRequestProxyおよびcom.ibm.websphere.servlet.response.HttpServletResponseProxy
    - Servlet APIのHttpServletRequestWrapper/HttpServletResponseWrapperを使用する
- セッションマネージャー（HttpSession）
  - Shared Session Contextを使用したWAR/EAR間のHttpSessionの共有
  - Servlet21SessionCompatibilityプロパティを使用したHttpSessionの共有
    - BLA内で情報を共有するIBMApplicationSessionを使用する
  - SSL IDによるセッショントラッキング
    - CookieもしくはURLリライティングによるトラッキングに置き換える
  - システムプロパティ，Webコンテナカスタムプロパティによる設定
    - セッションマネージャーのカスタムプロパティで設定する
- WebSphere Application Serverに同梱されていた以下のライブラリ
  - JavaServer Faces Widget Library (JWL) のJarファイル
    - Rationalツールに付属のものを使用する
  - Apache Struts 1.1, 1.2.4, and 1.2.7
    - Apache Strutsのサイトから最新のものをダウンロードして使用する

WAS V7で非推奨となった機能（トランザクション）

- トランザクション関連
  - com.ibm.websphere.jtaextensions.ExtendedJTATransactionインターフェースのregisterSynchronizationCallbackForCurrentTranメソッド
  - com.ibm.ws.extensionhelper.TransactionControlインターフェース
- トランザクションの制御には新しく追加されたJTA 1.1の機能を使用する
  - UOWManager#registerIterposedSynchronization()
    /javax.transaction.TransactionSynchronizationRegistry等



WAS V7で非推奨となった機能（DBアクセス）1

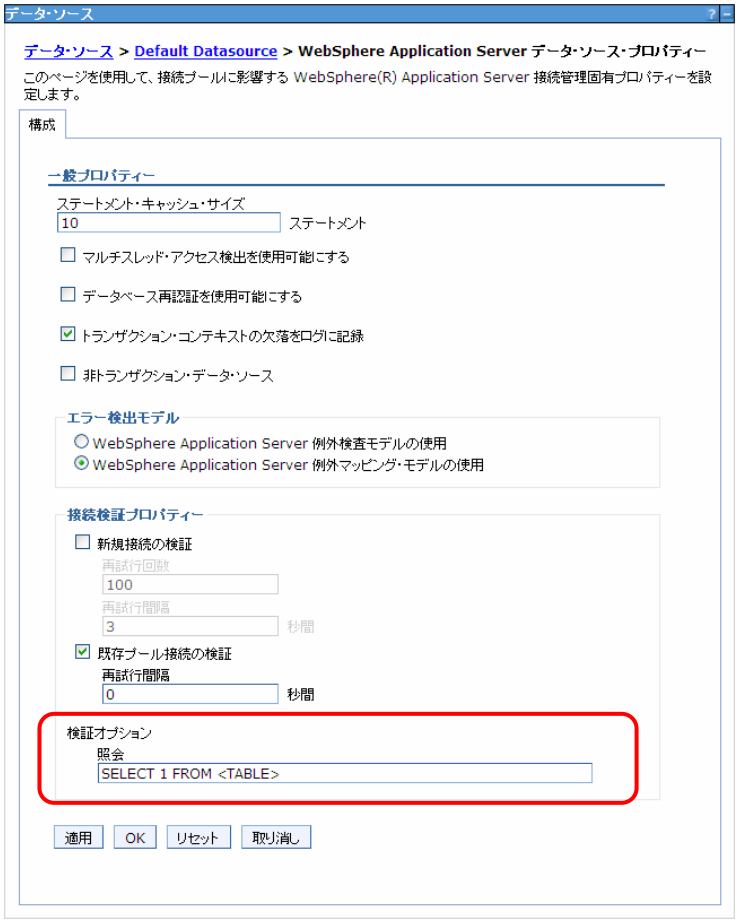
- DB接続関連クラス
  - com.ibm.websphere.rsadapterのクラス
    - JdbcAccessorImpl/OracleDataStoreHelper
  - com.ibm.websphere.rsadapterのインターフェース
    - Beginnable/HandleStates/Reassociateable/WSNativeConnectionAccessor
  - com.ibm.websphere.rsadapterのクラスのメソッド
    - WSCallHelperのsetConnectionError(Object conn)/call
    - WSConnectionのgetClientInformation/setClientInformation
  - com.ibm.ws.rsadapter.cciのクラスのメソッド
    - WSResourceAdapterBaseのgetNativeConnection(javax.resource.cci.Connection) /getNativeConnection(com.ibm.ws.rsadapter.jdbc.WSJdbcConnection)
  - com.ibm.ws.rsadapter.jdbcのクラスのメソッド
    - WSJdbcUtilのgetNativeConnection(com.ibm.ws.rsadapter.jdbc.WSJdbcConnection)
  - com.ibm.websphere.rsadapterのクラスのフィールド
    - WSConnectionの CLIENT_ACCOUNTING_INFO/CLIENT_APPLICATION_NAME/CLIENT_ID/CLIENT_LOCATION /CLIENT_OTHER_INFOCLIENT_TYPE
- getNativeConnectionはJDBC 4.0で追加されたメソッドを使用する
  - その他はInformation Centerの記述にしたがって対処

WAS V7で非推奨となった機能（DBアクセス）2

- SQLストリングによる妥当性検査
  - WebSphere Application Server データ・ソース・プロパティーに設定
    - 「検証オプション」の「照会」に設定する
  - JDBC 4.0で追加された Connection#isValid()メソッドを 使用することにより，SQL実行前に 有効性確認を行うことが可能になった

```
Connection conn = ds.getConnection();
if (!conn.isValid(10)) {
    // リトライ処理
}
Statement s = conn.createStatement();
s.executeUpdate(sql);
s.close();
```

JDBC 4.0の機能を利用した例



参考：DB接続障害のアプリケーション対応

- 従来はSQL実行時するまで，Connectionが有効かを確認できなかった
  - StaleConnectionExceptionの発生で無効接続の検知をおこなっていた

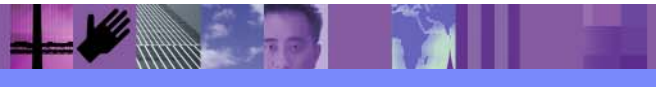
- JDBC 4.0で追加されたConnection#isValid()メソッドを使用することによりSQL実行前に有効性確認を行うことが可能になった
  - これからは，可能な限りこちらの方式を利用する

```
Connection conn = ds.getConnection();
while (true) {
    try {
        Statement s = conn.createStatement();
        s.executeUpdate(sql);
        s.close();
        break;
    } catch (StaleConnectionException e) {
        // リトライ処理
    }
}
```

従来の方法の例

```
Connection conn = ds.getConnection();
if (!conn.isValid(10)) {
    // リトライ処理
}
Statement s = conn.createStatement();
s.executeUpdate(slq);
s.close();
```

JDBC 4.0の機能を利用した例



WAS V7で非推奨となった機能（Messageリソース）

- J2C（J2EE Connector Architecture）1.5に準拠していないJMSプロバイダーの使用
  - WAS V5でデフォルトで使用されていたJMSプロバイダーはWAS 6.1で非推奨となっている
  - WebSphere MQで提供されるJMSプロバイダーは，WMQ V7以降でJ2C 1.5対応に
- JMSリスナーポートによるMDBの使用
  - リスナーポートを構成・管理するAdminConfigのコマンドやMBeanなども非推奨
  - JMS activation specificationによるMDBに置き換える

WAS V7で非推奨となった機能（Webサービス）1

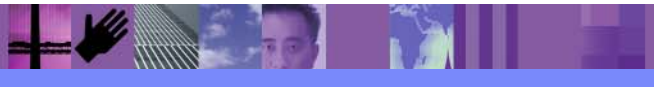
- Java API for XML Web Services (JAX-WS) 2.0で使用されていた WS-Addressingエンドポイントリファレンス
  - com.ibm.websphere.wsaddressing.jaxws.W3CEndpointReference
  - com.ibm.websphere.wsaddressing.jaxws.SubmissionEndpointReference
  - com.ibm.websphere.wsaddressing.jaxws.EndpointReferenceConverter
  - 以下のものに置き換える
    - javax.xml.ws.wsaddressing.W3CEndpointReference
    - com.ibm.websphere.wsaddressing.jaxws21.SubmissionEndpointReference
    - com.ibm.websphere.wsaddressing.jaxws21.EndpointReferenceConverter
- ‘2006/02’WS-Addressing WSDLネームスペースのサポート
  - ‘2006/05’ネームスペースを使用する
- Web Services Distributed Management (WSDM) インターフェース
  - WAS標準のMBeanを使用した管理機能を利用する

WAS V7で非推奨となった機能（Webサービス）2

- IBM独自実装のSOAP/JMSを利用したJAX-WS/JAX-RPCアプリケーション
  - 標準のSOAP/JMSを利用したアプリケーションへ書き換える
    - Webサービス・プロバイダー側
      - ボトムアップ開発の場合：
        BeanやEJBからプロバイダーを再作成する
      - トップダウン開発の場合：
        WSDLから再度スケルトンを生成し，従来のロジックをコピーする
    - Webサービス・リクエスター側
      - WSDLからクライアント・スタブを再作成する

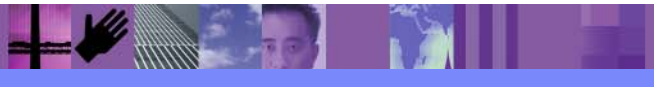
WAS V7で非推奨となった機能（SIBus）

- Websphere Common Configuration Model(WCCM)の以下の型
  - SIBJMSProvider/SIBJMSConnectionFactory/SIBJMSQueueConnectionFactory /SIBJMSTopicConnectionFactory/SIBJMSQueue/SIBJMSTopic
  - スクリプト中でこれらのクラスが使用されている場合には，AdminTaskで提供される同等のメソッド（例：AdminTask.listSIBJMSQueues()など）に置き換える
- wsadmin内で使用できる以下のSIBusのセキュリティ機能
  - createSIBus, modifySIBusコマンドの-secureオプション
    - -busSecurityオプションを使用する
  - listInheritSenderForTopic, listInheritReceiverForTopic, listInheritDefaultsForDestinationコマンド
    - isInheritSenderForTopic, isInheritReceiverForTopic, isInheritDefaultsForDestinatonコマンドに置き換える
  - Inter-engine authentication alias
    - createSIBus, modifySIBusコマンドから-interEngineAuthenticationAliasオプションを削除する



WAS V7で非推奨となった機能（サーバー構成・管理）

- disablePK54589プロパティ
  - 「disablePK54589=true」を設定している場合には「isConnectionSharingBasedOnCurrentState=false」に置き換える
- Core GroupトランスポートのUnicast/Multicast
  - チャネルトランスポートに移行する
- Collectorツール（collector.bat, collector.sh）
  - IBM Support Assistant（ISA）に統合されているAutoPDツールを使用する
  - ISAについては，以下のサイトを参照
    http://www.ibm.com/software/support/isa/
    http://www.ibm.com/support/docview.wss?rs=3455&uid=swg27013569
- WebSphere Touchpoint機能
  - 標準のWAS管理機能を使用する
- AdminTaskオブジェクトのSecureConversationコマンドグループ
  - WS-Security分散キャッシュ構成を管理するためにはWSSCacheManagementを使用する



WAS V7で非推奨となった機能（IBM HTTP Server）

- mod_file_cacheモジュール
  - mod_mem_cacheやmod_cacheなどのキャッシング機能に置き換える
- mod_ibm_ldapモジュール
  - mod_ldapに置き換える
- mod_mime_magic・mod_proxy_ftpモジュール
  - これらの使用は推奨されないので，関連ディレクティブを削除する
- AFPA機能（mod_afpa_cacheモジュール）
  - AFPA（Adaptive Fast Path Architecture）機能は，
    ほとんどの環境でパフォーマンス上の効果が無く，
    様々なトラブルの原因ともなるため，使用は推奨されません

WAS V7で削除された機能

- Java Virtual Machine Profiler Interface (JVMPI)
- Java Virtual Machine Debug Interface (JVMDI)
  - Java Virtual Machine Tool Interface (JVMTI)を使用する
- com.ibm.websphere.servlet.filterパッケージのクラス
  - サーブレットAPIで提供されるServletFilterに移行する
- Integrated Cryptographic Services Facility (ICSF)認証機構
  - Lightweight Third-Party Authentication (LTPA)認証機構を使用する
- mb2mdbツール
  - 後継機能は無し
- Web services gateway customization API
  - JAX-RPC handlerおよびSIBusメディエーションへ移行する
- com.ibm.websphere.servlet.session.UserTransactionWrapper
  - UserTransactionを直接HttpSessionに格納する
- com.ibm.websphere.rsadapter.DataDirectDataStoreHelper
  - com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelperを使用する
- com.ibm.websphere.rsadapter.MSSQLServerDataStoreHelper
  - com.ibm.websphere.rsadapter.MicrosoftSQLServerDataStoreHelperを使用する

WAS V7で削除された機能（JDBCサポート）

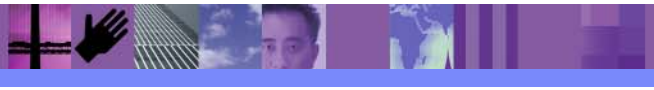
- 以下のJDBC Driverのサポート
  - WebSphere Connect JDBC driver
  - Microsoft SQL Server 2000 Driver for JDBC
  - WebSphere SequeLink JDBC driver for Microsoft SQL Server
- DB2® Legacy CLI-based Type 2 JDBC Driverのサポート
- DB2® Legacy CLI-based Type 2 JDBC Driver (XA)のサポート
  - DB2 Universal JDBC Driverを使用する

WAS V8で非推奨となった機能（1）

- com.ibm.websphere.product パッケージ
  - 以下のクラス
    - com.ibm.websphere.product.WASProduct.
    - com.ibm.websphere.product.WASProductException
    - com.ibm.websphere.product.history.WASHistory
    - com.ibm.websphere.product.history.WASHistoryException
      com.ibm.websphere.product.metadata.WASComponent
    - com.ibm.websphere.product.metadata.WASComponentUpdate
    - com.ibm.websphere.product.metadata.WASMaintenancePackage
  - com.ibm.websphere.product.WASDirectory の以下のメソッド
    - getInstalledComponentByName(String componentName)
    - getInstalledComponentList()
    - isComponentInstalled(String componentName)
    - getHistoryMaintenancePackageList()
    - getInstalledMaintenancePackageByID(String mpID)
    - getInstalledMaintenancePackageList()
    - isMaintenancePackageInstalled(String ID)
  - com.ibm.websphere.product.WASDirectory の以下のフィールド
    - ID_BASE / ID_EXPRESS / ID_ND
  - com.ibm.websphere.product.WASDirectory
    com.ibm.websphere.product.utils.WASDirectoryHelperの以下のフィールド
    - ID_PME / ID_WBI / ID_JDK / ID_EMBEDDED_EXPRESS / ID_XD / ID_CLIENT / ID_PLG / ID_IHS / ID_WXD / ID_NDDMZ / ID_UPDI

WAS V8で非推奨となった機能（2）

- historyInfoコマンドの以下のオプション
  - -components/-maintenancePackageID
- versionInfoコマンドの以下のオプション
  - -componentDetail/-components/-maintenancePackageDetail/-maintenancePackages
- pmtコマンドで起動するProfile Management Tool
- migrationコマンドで起動するMigration Management Tool
- 管理スクリプト
  - AdminTaskオブジェクトのcreateServerTypeコマンド
  - AdminTaskオブジェクトのManagedNodeGroupコマンドグループの以下のコマンド
    - createManagedNodeGroup/deleteManagedNodeGroup/addMemberToManagedNodeGroup deleteMemberFromManagedNodeGroup/queryManagedNodeGroups getManagedNodeGroupMembers/getManagedNodeGroupInfo/modifyManagedNodeGroupInfo
  - AdminTaskオブジェクトのJobManagerNodeコマンドグループの以下のコマンド
    - cleanupManagedNode/queryManagedNodes getManagedNodeProperties/modifyManagedNodeProperties/getManagedNodeKeys
- activity.logの使用
- IBM JVMのSubpool GC Policy



WAS V8で削除された機能

- Web Services GatewayにおけるApache SOAP channel
- Apache SOAP, WEBSJAVA.SOAP（soap.jar，wssoap.jar）
- 以下のデータベース・リソースアダプタークラス／フィールド
  - com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper
  - com.ibm.websphere.rsadapter.OracleDataStoreHelper
  - com.ibm.websphere.rsadapter.DataStoreHelper.ORACLE_10G_HELPER
  - com.ibm.websphere.rsadapter.DataStoreHelper.ORACLE_HELPER
- zOS版で，リクエストのトランザクションクラス割当てを記述したファイルを指定する以下のパラメーター
  - protocol_http_transport_class_mapping_file

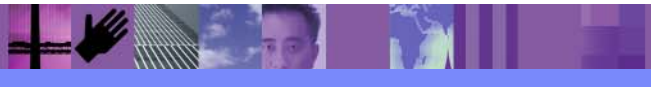
BigDecimalの仕様変更

- JSR 13: Decimal Arithmetic Enhancement のサポート
  - JDK 1.4のAPIドキュメントなどで予告されていたとおり，
    Java標準APIで提供されているクラスBigDecimalの仕様が，JDK 5.0より変更
  - 多くの新機能の追加
  - 仕様の制限の撤廃，ANSI規格との整合性の向上
- これにともない，BigDecimalのtoString()メソッドの動作が変更された
  - scale部の大きな（絶対値が1より非常に小さい）BigDecimalにおいて，
    従来は非指数表記の数値が返されていたケースで，
    「E」を含んだ指数表記で値が返されることがある

```
System.out.println(new BigDecimal("0.00000001"));
```

JDK 1.4の出力：“0.00000001”　　**JDK 5.0の出力**：“1E-8”

- String文字列化されたBigDecimalにたいして処理を行っているアプリケーションでは
  対応が必要なケースがある
  - 必要に応じてJDK 1.4と同じ結果を返すtoPlainString()に書き換える



BigDecimalの仕様変更のJDBC Driverへの影響について

- WAS 6.1/JDK 5.0上で
  DB2のJDBC Driver（Type2/4）を使用した場合に影響が報告されている
  - PreparedStatement#setBigDecimal()を使用した際にSQLExceptionが発生する障害が発生することがある
  - リリースごとに修正が作成されているので，それを適用する
    - V8.2
      APAR IZ10805
    - V9.1
      APAR IZ10807
    - V9.5
      APAR IZ10808
- Oracle JDBC Driverでも同様の問題が発生し，V10.2.0.4で修正されている
  - Bug Number4711863:
    setBigDecimal corrupts number for negative values

JDKのバグが直ったことによる問題の例

- 西暦1000年より前の日付をセットしたjava.sql.Dateの文字列表現には，以前のバージョンのJDKでは年号が4桁より少なくなるというバグが存在した
  - 「System.out.println(new Date(-1899,0,1));」というコードは，「0001-01-01」という文字列を出力するのが正しいが，以前は「1-01-01」と出力されてしまっていた
  - このバグはSun JDKでは1.3で，IBM JDKでは遅れて1.4から修正された
  - WAS 5.1以前からのマイグレーションで問題となるケースがある

```
if (date.toString().equals(“1-01-01”)) {
        // Special case
} else {
        // normal operations
}
```

JDK/JREの実装が変わったことによる問題の例

- 以下のコードは，Javaのコードとしては間違ったコード

```
Hashtable table;
        ...
Iterator it = (Iterator)table.keys();
```

  - HashtableのkeysメソッドのEnumerationで，Iteratorにキャストはできない

- SunのJDK/JREでは，keysで返される内部クラスのインスタンスは，Iteratorも実装しているものとなっていたため，このコードは「たまたま」動いてしまっていた
- IBM J9VMの提供する内部クラスはIteratorは実装していなかったため，このコードは実行エラーとなった

WASのバグが直ったことによる問題の例

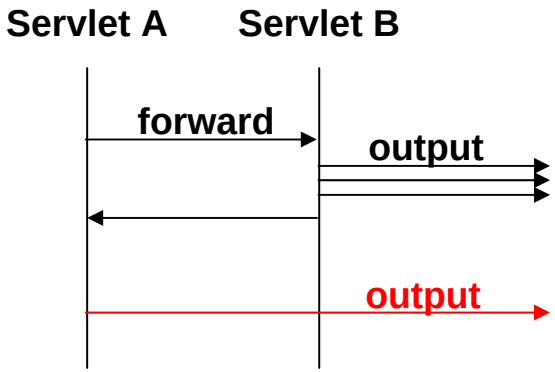
- JSPでnullオブジェクトを出力したとき
  - 「<% String s = null; %> <%= s %>」というJSPコードは「null」という文字列を表示するのが正しい挙動
  - WASの3.5.2より前のバージョンでは，何も表示されないというバグがあった
  - このバグがAPAR PQ43560で修正された結果，画面に大量の「null」という文字が表示されたアプリケーションがあった
- executeQueryによる結果を返さないSQLの実行
  - JDBCの仕様では，UPDATE/INSERT/DELETEなど結果を返さないSQLはexecuteUpdate，結果を返すSELECTはexecuteQueryで実行することになっている
  - DB2の古いJDBC Driverでは，UPDATE/INSERT/DELETEをexecuteQueryで実行しても正常に動いてしまっていた
  - これが仕様どおりにSQLExceptionを返すように修正されたときに，大量のアプリケーションが正常稼動しなくなった

WASのバグが直ったことによる問題の例（2）

- HttpServletRequestの「getCookies()」の戻り値
  - クライアントからCookieが送られていない場合，nullを返すのが仕様
  - WAS 3.5以前のバージョンでは，長さ0の配列が返ってしまっていた
  - 戻り値のnullチェックをしていなかったアプリケーションでエラーが発生
- includeされたServlet/JSPでのaddHeader/setHeaderの実行
  - 他のServlet/JSPにincludeされたServlet/JSP内部では，ヘッダの操作はできないのが仕様
    - サーブレットの中でRequestDispatcherを使用したり，JSPの中で<jsp:include page=“ “ />タグを使用した場合が該当
    - 実際に実行しても無視される
    - JSPで<%@ include file=“ “ %>を使用した場合は，実際には一つのサーブレットになるので，読み込まれる側でaddHeader/setHeaderは実行できる
  - WAS 5.1以前では，includeされた側でヘッダの操作ができてしまうという問題が存在
  - WAS 5.1.0.4以降でこの問題が修正されたため，アプリケーションによっては問題が発生した

WASのバグが直ったことによる問題の例（3）

- RequestDispatcher#forward()
  - forward()の呼び出しを行った場合，
    呼び出し前/後にはクライアントへの出力ができないのが仕様
    - クライアントへの出力は，呼び出されたServlet/JSP中でおこなう
  - WAS 3.5以前には，forward()呼び出しから戻った後の出力が，
    クライアントに送信されるというバグがあった
  - これが修正されたところ，画面の下半分が消えてしまった，
    というアプリケーションの事例があった



- HttpSession#isNew()の戻り値
  - JSPからJSPをincludeしたとき，
    isNew()の値が正しくなくなるバグが修正された（APAR PQ81416）

test1.jsp

```
<%@page contentType="text/plain" session="true" %>
<%= session.isNew() %>
<jsp:include page="test2.jsp" />
```

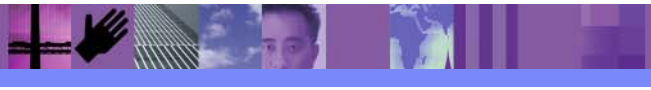
test2.jsp

```
<%@page session="true" %>
<%= session.isNew() %>
```

  - test1.jspに最初にアクセスしたときの戻り値
    - 修正以前　　“true” “false”
    - 修正後：　　“true” “true”

WASのバグが直ったことによる問題の例（4）

- include/forwardされたサーブレット/JSPでHttpServletRequest#getQueryString() を実行した結果
  - Servlet 2.3仕様では，RequestDispatcherで処理をincludeないしforwardで転送した場合，転送に使用したURLでQuery Stringが更新されるべきであると定義されている
  - WAS V5.0.2.10，V5.1.1.5およびV6.0.2よりも前のバージョンでは，転送されたサーブレット/JSPからgetQueryString()を実行した際に，返される値が転送前から更新されていないというバグが存在していました
  - 上記のバージョンおよびWAS V6.1以降では，正しい値（更新された値）が返されるようになる
- HttpSession#getLastAccessedTime()の戻り値
  - WAS V5.0/5.1では，getLastAccessedTime()が前回アクセスしたセッションの時刻ではなく，現在のセッションの時刻を返すバグがあった
  - WAS V6.0以降ではこれが正しく前回のセッションの時刻を返すように修正されているため，V5.xの挙動に依存したアプリケーションでは修正が必要になるケースがある



仕様外の動作による問題の例（1）

- <jsp:useBean>のclass属性
  - class属性にはJavaBean仕様をみたしたクラスを指定しなければいけない
    - 「Serializableである」「デフォルトコンストラクタを持つ」など
  - <jsp:useBean>では，指定されたscopeに既存のBeanが見つからない場合，class属性のBeanがデフォルトコンストラクタを使用して新規に生成される
  - 指定したクラスがデフォルトコンストラクタを持たない場合
    - WAS 5.0以前
      Beanが見つからない場合に実行時エラー
      → 見つかれば正常実行
    - WAS 5.1以降
      JSPが最初に実行されたときにコンパイルエラー
      → いかなる場合も実行不能

仕様外の動作による問題の例（2）

- JSPの中に，対応する開きタグがない閉じタグ（</tsx:repeat>等）が存在している場合
  - WAS V5.1まで
    余分な閉じタグは無視されていました
  - WAS V6.0以降
    JSPのコンパイル時にエラーが出力されます
  - このエラーを無視するオプションを追加するAPARがWAS 6.0/6.1を対象として作成されています（PK30879）
- 異なる型を引数とする複数のSetterを定義したJSPカスタムタグの挙動
  - カスタムタグの属性（attribute）は，定義されたSetterの型に自動的に変換されて，タグハンドラ・クラスにわたされます
    - <mytag:CustomTag option="true"> → CustomTagHandler.setOption(true) 等
  - タグハンドラ・クラスに複数のSetterが指定されていた場合，どちらのSetterの型に変換されるかは環境依存
    - setOption(String opt)とsetOption(boolean opt)の両方が定義されていた場合など
    - バージョンアップを行うと，呼ばれるSetterがかわり，動作がかわる場合がある
  - 同名の複数のSetterを定義しないようにする必要があります

仕様外の動作による問題の例（3）

- JSPファイル中でpageContext.includeを使用して読み出しているJSPファイルが存在していないときの挙動
  - WAS V6.1.0.23以降，もしくはPK65408適用後，およびWAS V7.0
    - 「SRVE0190E: File not found」がログに記録され，エラーページが呼び出される
  - WAS V6.1.0.23以降，もしくはPK65408適用後およびWAS V7.0で，Webコンテナのカスタムプロパティとして「com.ibm.ws.webcontainer.throwMissingJspException」が設定されている
    - FileNotFoundExceptionがthrowされる
  - WAS V8.0
    - FileNotFoundExceptionがthrowされる
  - それ以前
    - 無視される
- JSPファイル中で<jsp:include>タグで読み出しているJSPファイルが存在していないときの挙動
  - WAS V5.1以前
    - FileNotFoundExceptionがthrowされる
  - WAS V6.0/6.1およびWAS V7.0
    - 「JSPG0036E: Failed to find resource」がログに記録され，エラーページが呼び出される
  - WAS V6.1.0.17以降，もしくはV6.0.2.29以降，もしくはPK57843適用後およびWAS V7.0で，Webコンテナのカスタムプロパティとして「com.ibm.ws.webcontainer.throwMissingJspException」が設定されている
    - FileNotFoundExceptionがthrowされる
  - WAS V8.0
    - FileNotFoundExceptionがthrowされる
- いずれのバージョンも，存在しないファイルのincludeを修正することを推奨

仕様外の動作による問題の例(4)

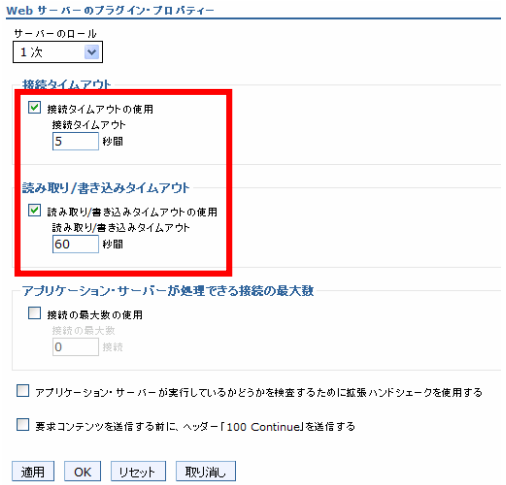
- Stringのコンストラクタで文字セットを指定してバイト列を文字に変換する際に,
  バイト列に文字セットで扱えない不正なバイト列が含まれている場合
  - 不正なバイト列を含んだ入力を変換した場合の挙動がJDKのバージョンによって
    以下のように異なる
    - 不正なバイト列の部分だけが無視される
      - 以前のバージョンのIBM JDK
      - システムプロパティ「-Dcom.ibm.IgnoreMalformedInput=true」が設定されたIBM JDK
    - バイト列全体がU+FFFDに変換される
      - WAS V6.0/6.1/7.0/8.0でIZ80871/IZ80870/PM18989が適用されたIBM JDK
    - 不正なバイト列の部分だけがU+FFFDに変換される
      - Sun/Oracle/HP JDK
      - WAS V6.0/6.1/7.0/8.0でIV02528/IV06610/PM46284が適用されたIBM JDK
  - それぞれのAPARが含まれるFixpackやSRのバージョンなど,
    詳細については以下の文章を参照
    http://www.ibm.com/support/docview.wss?uid=jpn1J1008815
  - Java標準API仕様では, 結果は「不定」とされている
    - 不正なバイト列の変換を含めた厳密な変換結果が要求される場合には,
      java.nio.charset.CharsetDecoderを使用するようにする

その他，WASのバージョン間で実装や挙動が異なる例（1）

- クラスが格納されているJARファイルのディレクトリ・ファイル名
  - WASのバージョンが異なると，同名のクラスであっても，格納されているJARファイルのファイル名が異なっていたり，配置されているディレクトリが異なっているケースがあります
  - Rationalなどの開発ツール上ではなく，WAS環境のJARを使用してantなどでアプリケーションのコンパイルを行っている場合には，前提クラスを読み込むパスを変更する必要があるケースがあります
- MSIE 6の問題KB812935をWAS側で回避するためのパラメーターが，バージョンによって異なっています
  - 詳細については以下の文章をご参照ください
    http://www.ibm.com/jp/domino01/mkt/cnpages1.nsf/page/default-0005A39E

その他，WASのバージョン間で実装や挙動が異なる例（2）

- HTTPサーバープラグインのタイムアウト値のデフォルト値の変更
  - 「接続タイムアウト」「読み取り／書き取りタイムアウト」のデフォルト値がWSA V7で変更（5分→1分）されています
  - 応答時間が1分を超えるようなアプリケーションではタイムアウト値の変更が必要です
- JDKのデフォルトのGCポリシーの変更
  - WAS V6.1/7.0のIBM JDK
    - -Xgcpolicy:optthruputがデフォルト
  - WAS V8.0のIBM JDK
    - -Xgcpolicy:genconがデフォルト
  - Javaヒープのチューニングを行っていた場合，設定値の変更が必要です



その他，WASのバージョン間で実装や挙動が異なる例（3）

- WAS V8より，
  セッションマネジメント用Cookie（JSESSIONID）にHttpOnlyオプションがデフォルトで付加
  - HttpOnlyオプションが指定されたCookieは，多くのブラウザでJavaScriptから不可視となる
  - XSSによるセッションハイジャッキング等を防止することが可能
  - この影響で，JSESSIONIDをJavaScriptで直接操作しているアプリケーションがエラーになることがある
    - この場合，セッション管理で「セッションCookieをHttpOnlyに設定」を無効にする

