Libertyプロファイルによる開発・構成のセッションです。当セッションでは、システム担当者・アプリケーション開発者の方が、実際にLibertyを導入してシステムの構成を行い、アプリケーションの開発を行えるところまで、ご説明していきたいと思います。

03. Libertyプロファイルによる開発・構成

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2013年7月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- IBM、IBMロゴおよびibm.com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。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- 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。
- Microsoft, Windows および Windowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。

© 2013 IBM Corporation

Agenda

- Libertyプロファイルの導入
  - 3つの導入方法
- Libertyプロファイルの基本構成と定義
  - 基本構成
  - ディレクトリ構成
  - server.xmlの定義と編集・構成
  - アプリケーションの導入方法
- Libertyプロファイルでのフィーチャーの利用
  - Libertyプロファイルのアーキテクチャ
  - フィーチャーの利用
- Libertyプロファイルでの開発
  - Infrastructure as Code にむけて

© 2013 IBM Corporation

当セッションのアジェンダです。

Libertyプロファイルの導入から、基本的な構成、アプリケーションの管理、様々なフィーチャーの利用といったアプリケーション開発に必要な内容を中心にご説明します。それぞれの機能を確認したあと、Libertyプロファイルにおける開発とはなにがうれしいのか、どういった潮流の中にこの製品を位置づけることができるのか考えます。

まず、Libertyプロファイルの導入です。

Libertyプロファイルの3通りの導入方法

1. IBM Installation Managerによる導入
   - 従来のWASと同様にInstallation Managerを利用して導入

2. 自己展開圧縮jarファイルを実行して展開
   - jarファイルをコマンド・ラインから実行
   - 既存の導入・構成済みの環境をパッケージして、独自の導入zipファイルを作成することも可能

3. Eclipse上のWDTを使用してダウンロード・導入
   - WASdev にアクセスして WebSphere Application Server Developer Tools (WDT)を導入
   - WDTを利用して、Libertyを導入

© 2013 IBM Corporation

Libertyプロファイルの導入方法には3種類あります。

従来のWebSphere Application Serverと同様、IBM Installation Managerによる導入です。導入コンポーネントの管理やFIXの管理が楽になるメリットがありますが、Installation Managerを別途導入しないといけないなどの手間がかかる側面もあります。

より簡便な導入方法として、jarファイル形式で提供される自己展開圧縮を実行・展開することで導入する方法があります。シェルなどでの環境構築の自動化が容易です。また次章でコメントがありますが、構成・稼動確認済みのLibertyプロファイル環境をコマンドでパッケージングして、自己展開ファイルとして独自の導入ファイルを構成することも可能です。

さらに、Eclipse環境でのシステム開発者に向けの導入方法がWDT (WebSphere Application Server Developer Tools)を利用した導入方法です。まずWDTを導入し、このWDTのウィザードに従って、Libertyを導入することが可能です。

これらについて、もう少し詳しく見ていきます。

まず、IBM Installation Manager (IIM) による導入です。

事前に別途IBM Installation Managerを導入していただき、IIMのリポジトリとして別途用意した導入イメージをポイントして、導入します。

WAS 8.5.0では従来型のWASの一部のように選択していましたが、WAS8.5.5からは独立してインストール可能なコンポーネントに昇格しています。なお、既にWAS8.5.0が導入されている環境に対して、そのままWAS8.5.5にアップグレードすることはできないため、WAS8.5.0を別名バックアップしたあと、別導入する形になります。

03. Libertyプロファイルによる開発・構成

2) 自己展開圧縮ファイルを実行して展開(1)

- 導入ファイルは実行可能Jarファイル
  - ファイル名は wlp-yyyyy-runtime-8.5.5.x.jar
    (「yyyyy」の部分はエディション、「8.5.5.x」の部分はバージョン番号が入る)
- 入手方法
  - 製品の導入メディアのWLPディレクトリーに格納
  - WASDev.netからダウンロード(要IBM ID登録)
- 導入手順
  1. コマンドラインから「java -jar wlp-yyyyy-runtime-8.5.5.x.jar」で実行
  2. ライセンス条項(プログラムのご使用条件)の表示またはスキップ
  3. ライセンス条項への同意
  4. 展開先ディレクトリーの指定
  5. 指定したディレクトリーにwlpディレクトリーが作成され製品が展開される
- 注意点
  - Java SE 1.6以降のJREで実行が可能
  - 展開先ディレクトリーにwlpディレクトリーがすでに存在している場合は失敗

© 2013 IBM Corporation

次に 自己展開圧縮可能jarファイルを展開して導入する方法です。ファイル名は wlp-developer-runtime-8.5.5.x.jar のような形で提供されます。

製品導入メディアから入手することも可能ですし、Developerエディションは WSDev.netからダウンロードすることが可能です。

03. Libertyプロファイルによる開発・構成

2) 自己展開圧縮ファイルを実行して展開(2)

1. Jarファイルを実行
2. ライセンスを表示・スキップ

```
C:¥Windows¥system32¥cmd.exe - java -jar wlp-developers-runtime-8.5.5.0.jar
C:¥tmp>java -jar wlp-developers-runtime-8.5.5.0.jar
IBM WebSphere Application Server for Developers V8.5.5
を使用、抽出、またはインストールする前に、プログラムのご使用条件 (保証適用外プロ
グラム用)
の条項と追加のライセンス情報に同意する必要があります。 以下の使用条件をよくお読
みください。

--viewLicenseAgreement オプションを使用して、ご使用条件を個別に表示できます。

ここでライセンス条項を表示する場合は Enter、スキップする場合は 'x' を押してくだ
さい。
```

3. ライセンス条項への同意

```
以下で「同意する」を選択すると、使用許諾契約の契約条件 (該当する場合は、 IBM 以外
の契約条件を含む) を受諾することになります。
同意しない場合は、「同意しない」を選択してください。

[1] 同意する、または [2] 同意しないを選択:  1
```

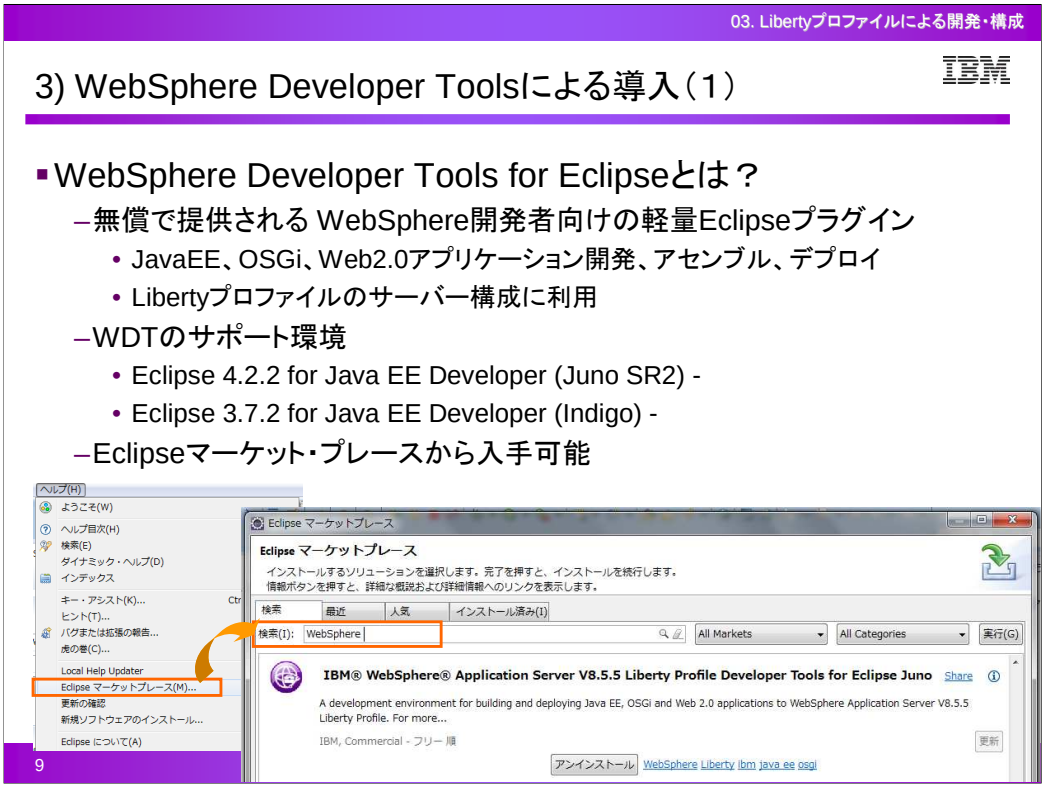
4. 展開先ディレクトリーの指定
   - カレント・ディレクトリが
     デフォルト

```
製品ファイルのディレクトリーを入力するか、ブランクのままにしてデフォルト値を受け
入れます。
デフォルトのターゲット・ディレクトリー: C:¥tmp

製品ファイルのターゲット・ディレクトリー? C:¥IBM¥LibertyBase
ファイルを C:¥IBM¥LibertyBase¥wlp に抽出しています。
すべての製品ファイルを正常に抽出しました。
```

© 2013 IBM Corporation

導入手順としては、java –jarコマンドでダウンロードしてきた jarファイルを実行します。ライセンスが表示されますので、ライセンスを確認して同意します。展開先ディレクトリの確認がありますので、導入したいディレクトリを指定して導入します。デフォルトでは、カレント・ディレクトリが指定されている点と、指定したディレクトリ下に wlp というディレクトリが作成されて入る点にご注意ください。

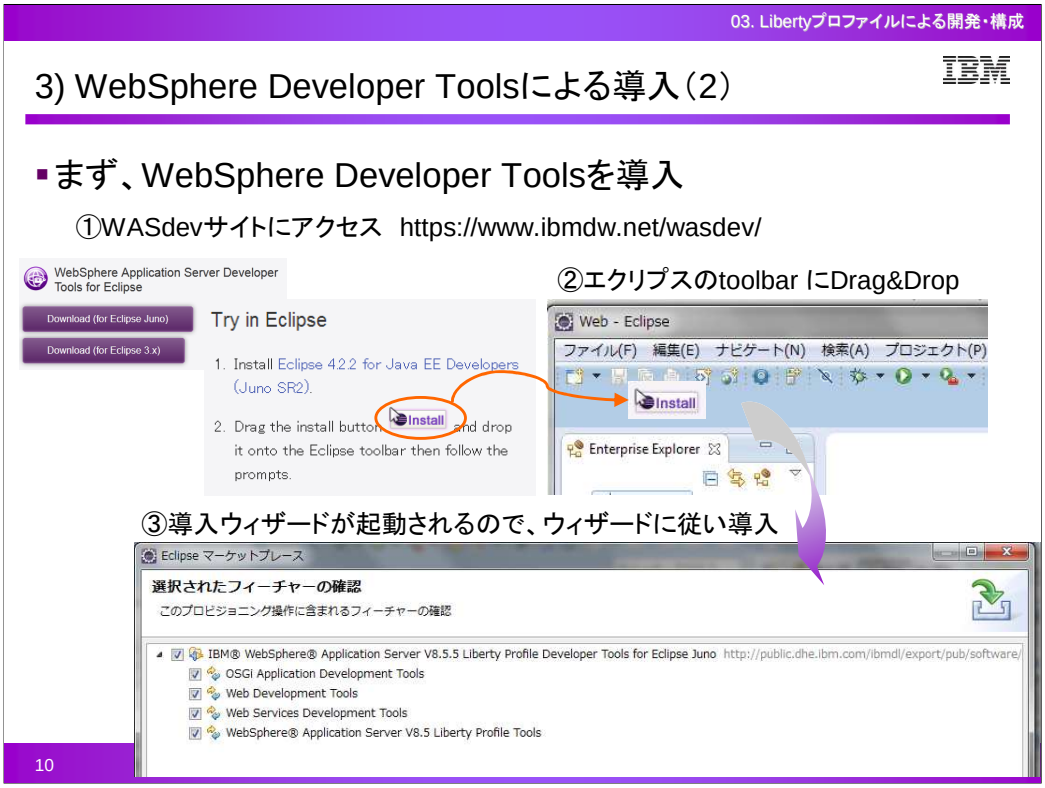


次に WebSphere Developer Tools を利用して、導入する方法です。

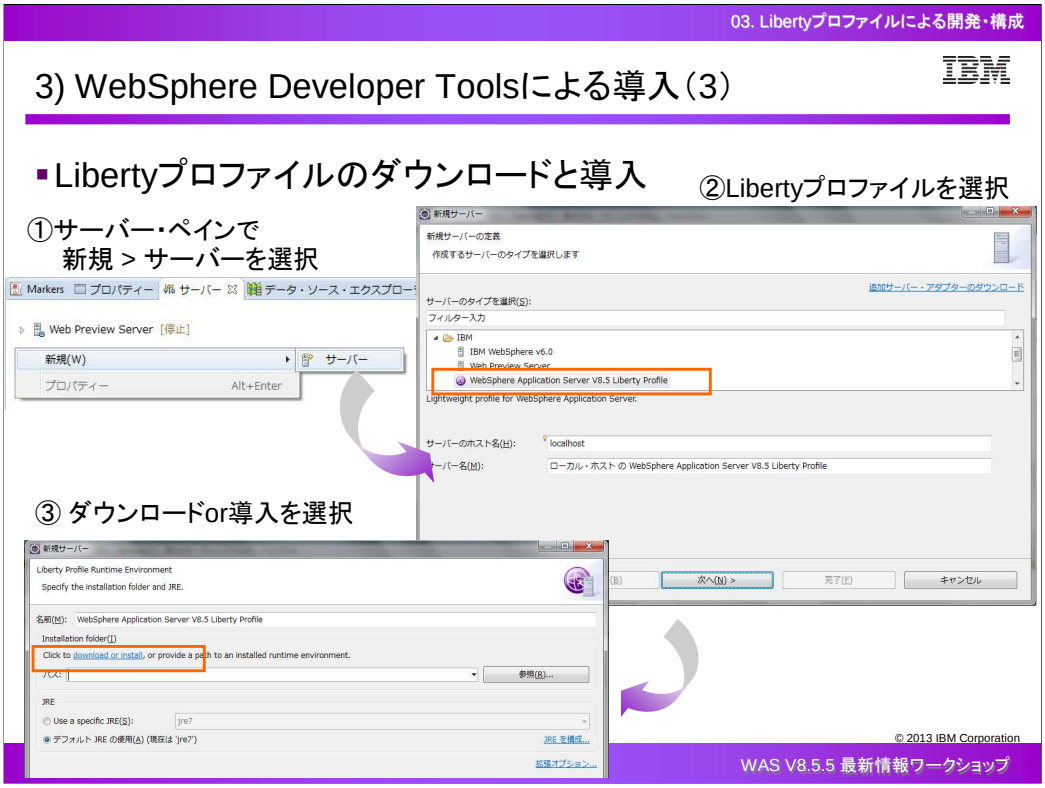
WebSphere Developer Toolsは、無償で提供されるWebSphere開発者向けの軽量Eclipseプラグインです。WebSphere向けて、Java EEの開発やアセンブルをヘルプする機能が提供されます。さらに今回のLibertyプロファイルのサーバー構成をサポートする機能が豊富に含まれています。

WDT導入方法としては、Eclipseのヘルプ→ Eclipseマーケット・プレースで WebSphereと検索して頂ければ見つけることが可能です。

またWASDevのサイトを利用した、より簡単な導入方法がありますので、そちらを紹介します。

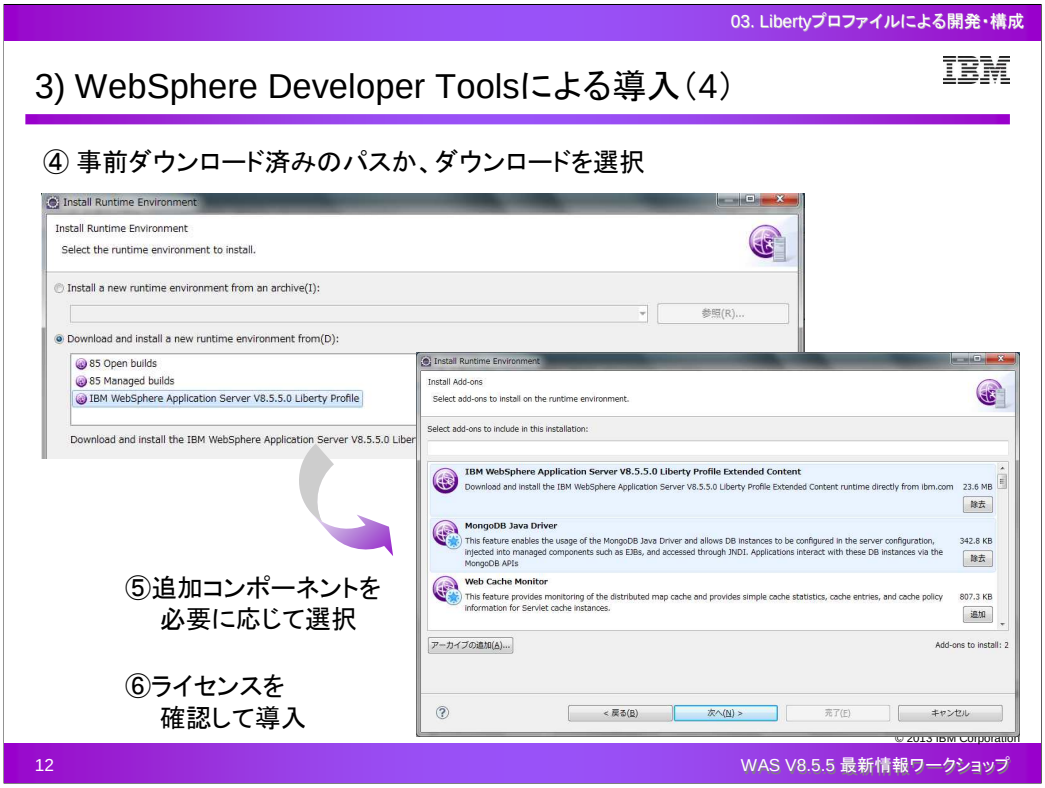


まずWASdevサイトにアクセスしてください。こちらのダウンロード・ページに WebSphere Application Server Developer Tools for Eclipseというリンクがありますので、こちらでご自身のEclipseのバージョンにあるリンクを開きます。開いた画面において、ブラウザーから「Install」のアイコンを、ご利用中のEclipseのツールバーにDrag&Dropしてください。自動で WebSphere Developer Toolsの導入ウィザードが開始されますので、ウィザードに従い導入を完了させます。

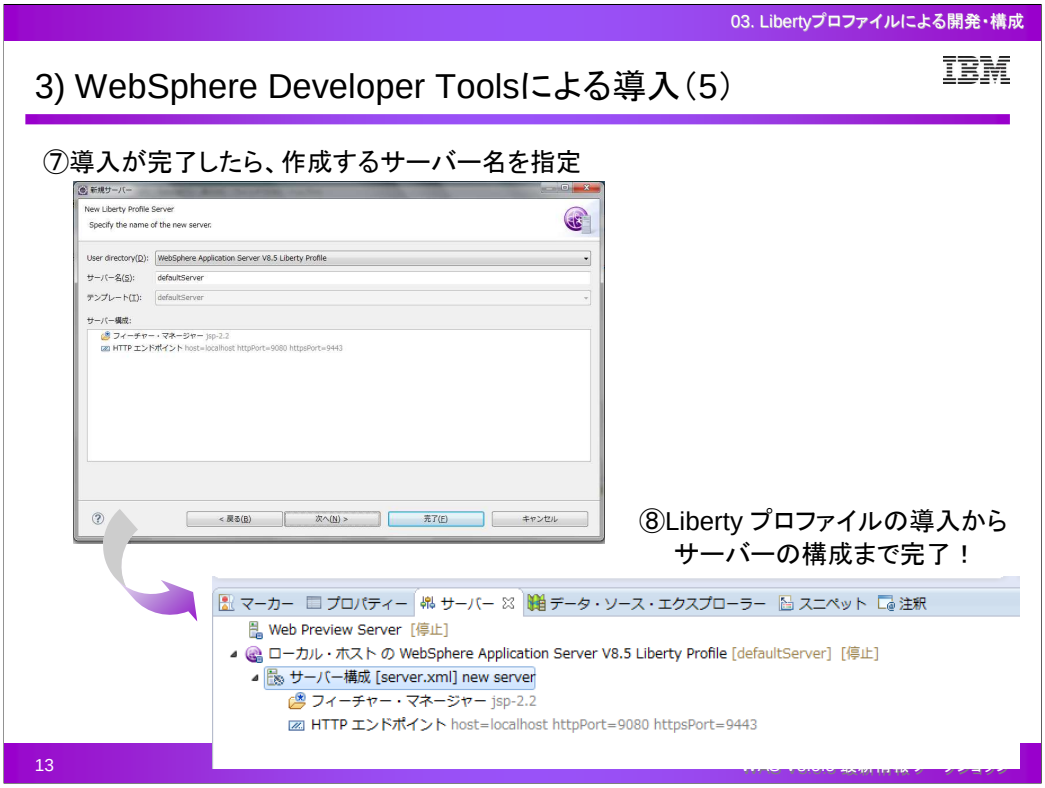


サーバー・ペインで「新規」> 「サーバー」を選択します。サーバーのタイプを選択する画面で WebSphere Application Server V8.5.5 Liberty Profileを選択し、「次へ」をクリックします。

次にLibertyのインストール・イメージを選択する画面になりますので、まだ未導入であれば「download or install」のクリックを選択します。

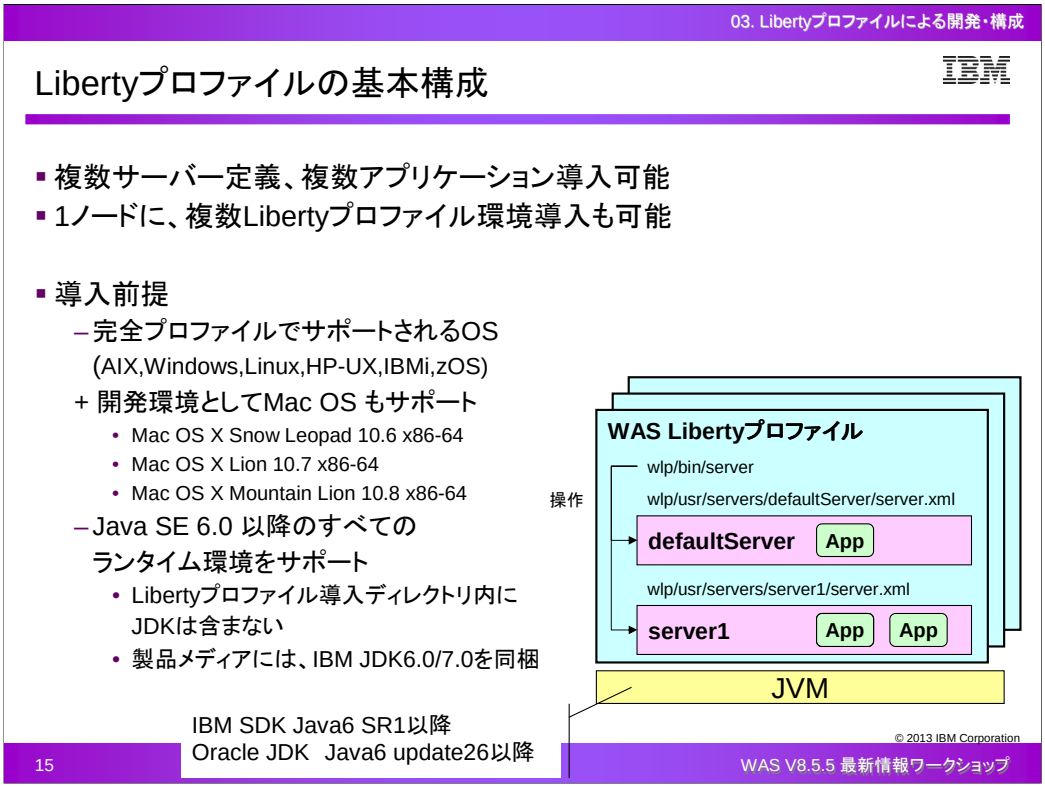


事前にインストール・イメージがある場合はそのパスを、まだイメージがない場合には「Download and Install」を選んで、Liberty Profileを選択します。「次へ」を押すと、追加コンポーネントを選択する画面が出ます。サンプル・アプリケーションなど必要に応じて追加コンポーネントを選択して「次へ」を押します。あとはライセンスに同意して導入します。



導入が完了したら作成するサーバー名が確認されますので、指定して構成します。デフォルトでは defaultServerという名前でサーバーが作成されます。

次に、Libertyプロファイルの基本構成と構成定義の編集方法についてみていきます。

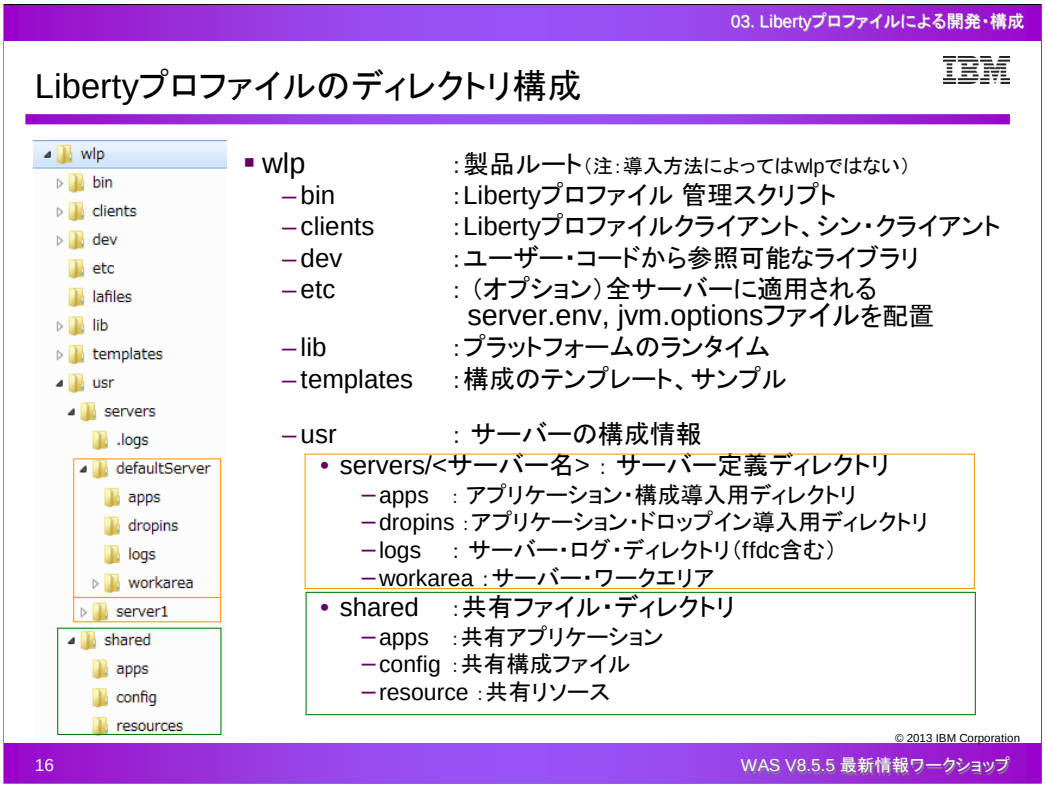


まずLibertyプロファイルの基本構成です。

Libertyプロファイルは、1ノードに複数導入も可能です。1つのLibertyプロファイルの中で、複数のサーバー・インスタンスを構成することももちろん可能ですし、1つのサーバー・インスタンスの中で、複数アプリケーションを稼動させることも可能です。

LibertyでサポートしているOS環境は、完全プロファイルでサポートされるOSに加えて、開発環境としてMacOSもサポートしています。

Libertyのインストール・イメージ自体にはJVMを含んでいません。Java SE6.0以降のランタイム環境をサポートしており、WindowsやLinux環境では IBM JDKに加えて、Oracle JDKを利用することもできます。ただし、製品メディアには IBM JDK6.0および 7.0を同梱しています。本番環境などでJVMまであわせて一貫したサポートが必要なお客様はIBM JDKをご利用いただくメリットがあります。また IBM JDKが提供されていない HP環境やMac環境ではOracleのJDKを利用することになります。



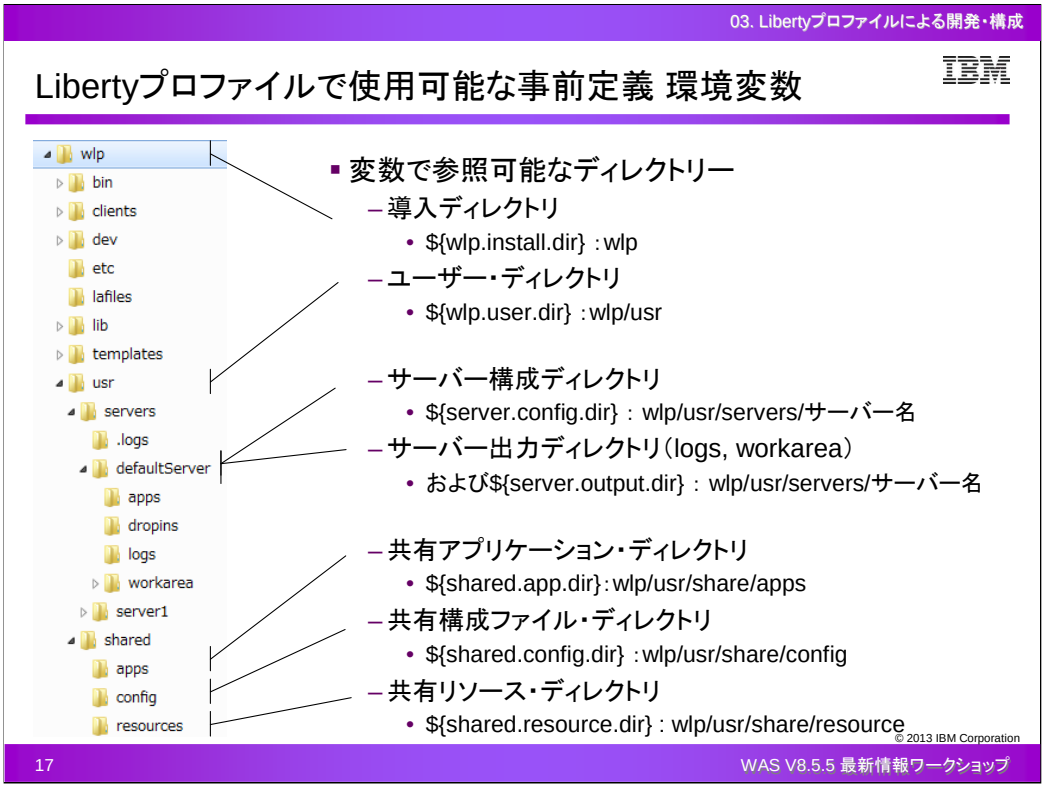
Libertyプロファイルのディレクトリ構成です。

製品rootディレクトリがwlpです(IIMで導入した場合はtopディレクトリとしてwlpは作成されませんので、導入時に指定したディレクトリ名になります)。その下にbinディレクトリがあり、サーバー管理に使うserverコマンドなどが含まれます。devディレクトリの下には apiやspiなど、アプリケーション・コードから参照可能なライブラリが含まれています。デフォルトでは存在しませんが、etcディレクトリを作成していただくと、全サーバーに共通して適用可能なserver.envファイルやjvm.optionsファイルを配置することが可能です。libの下に実際のLibertyランタイムのライブラリが含まれます。templatesの下にはサーバー構成のデフォルト・テンプレートが含まれます。

usrディレクトリ以下が、実際にユーザーが構成で使用するディレクトリです。

usrディレクトリの下のserversディレクトリ下にサーバーごとにディレクトリが作成されます。各サーバーには、アプリケーションの導入用ディレクトリ apps, dropinsディレクトリやlogsディレクトリが存在し、サーバー自体の作業ディレクトリとなるworkareaが存在します。

さらにsharedディレクトリ以下には、サーバー間で共有するファイルを配置しておけるsharedディレクトリがあります。共有構成ファイルや、サーバー間で共有するライブラリを配置する際に使用します。



Libertyプロファイルのサーバー構成ファイルで使用することができる事前定義変数です。

03. Libertyプロファイルによる開発・構成

<LIBERTY_ROOT>¥bin¥serverコマンド

- 基本書式

  server <アクション> <サーバー名> [Options]

- 起動・停止
  - create :新規サーバーインスタンスの作成
  - start :バック・グラウンドでサーバーを起動
  - run :フォア・グラウンドでサーバーを起動
  - debug :JVMデバッグオプションで、フォアグラウンドでサーバーを起動
  - stop :稼動しているサーバーを停止
- 管理
  - status :指定されたサーバーのステータスを確認
  - package :サーバー・ランタイムと、サーバー構成・アプリケーションをzipファイルにパッケージング
  - version :サーバー・ランタイムバージョンを表示
- 問題判別
  - dump :サーバーから問題判別資料をダンプし、アーカイブ
  - javadump :javacoreを出力
  - help :詳細オプション含め、コマンドヘルプを表示

© 2013 IBM Corporation

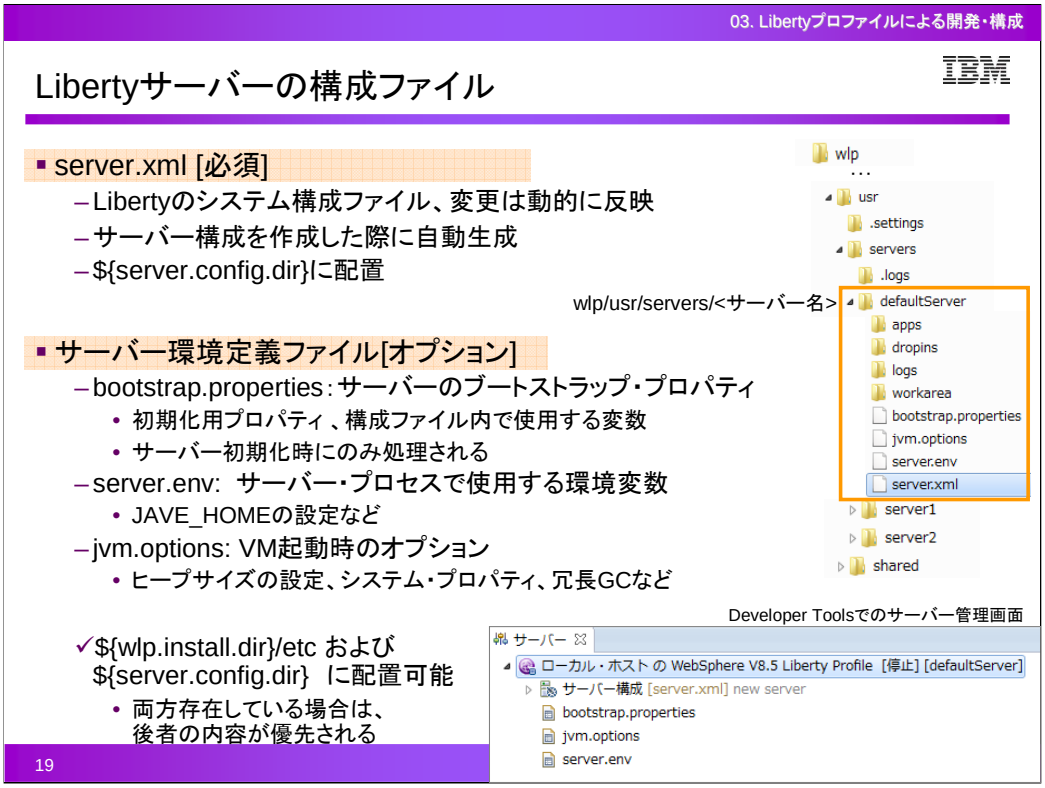
Libertyプロファイルを利用する際にもっとも基本的なコマンド群です。従来のフル・プロファイルのWebSphereではstartServer, stopServerなどコマンドが分かれていましたが、Libertyではserverコマンドに様々なコマンドが集約されています。

サーバー・インスタンスの作成は server create <serverName> のように、非常に簡単に作成することが可能です。

server startで起動するとサーバーはバック・グラウンドで起動、server runで起動するとサーバーはフォア・グラウンドで起動します。

server packageコマンドを利用することでLiberty環境をランタイム含めてパッケージすることができます。package機能の詳細については次のセッションで説明があります。

これまでフル・プロファイルではcollector.sh などで実施していた問題判別資料の収集も、製品の基本コマンドに統合されています。



それでは、Libertyプロファイルを構成する構成ファイルについてみていきましょう。

まず唯一必須となる構成ファイルが server.xmlです。シンプルな構成であればこのserver.xmlのみで構成が完了します。サーバー構成ディレクトリ直下に作成されます。

さらにその他として、追加で作成・利用可能なファイルが3つあります。1つ目が、サーバーの初期化時に読み込まれるbootstrap.propertiesがあります。構成ファイル内で使用する変数であったり、トレース・ファイル定義だったりを指定することが可能です。サーバー・プロセスが読み込む環境変数を定義する server.envファイルや、JVM起動時オプションを設定することができる jvm.optionsファイルがあります。ヒープサイズの設定やverbose:gcの設定はここで実施します。これらのファイルはサーバー起動時に読み込まれるため動的変更は行われません。V8.5まではこれらのオプショナル・ファイルは手で作成する必要がありましたが、V8.5.5からはWebSphere Developer Toolで作成することが可能です。

server.xmlの定義

- サーバー構成をXML形式で記述
  - <server>エレメントの子要素として設定を追加していく
    - <featureManager>
      - 利用するフィーチャーを定義
    - <httpEndpoint>
      - サーバーのホスト名、トランスポートを指定
  - システム・デフォルト設定が定義済み
    - デフォルトを変更する場合のみ記述を追加
    - 記述がない場合は、デフォルト値が使用される

デフォルトで生成されるserver.xml

```
<server description="new server">
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>
    <httpEndpoint id="defaultHttpEndpoint"
            host="localhost"
            httpPort="9080"
            httpsPort="9443"
    />
</server>
```

© 2013 IBM Corporation

server.xmlについて、より詳しく見ていきます。

右にあるXMLファイルが、デフォルトで作成される server.xmlファイルです。この構成ファイルでJSPとServlet稼動するWebコンテナーについて定義が完了しています。従来のWASの構成ファイルをご存知の方はそのシンプルさに驚くことと思います。

簡単にXMLの要素を見ていきます。<server>エレメントの下に<featureMaanger>エレメントがありこの中でJSPが有効化されています。さらに<httpEndpoint>エレメントがありWebコンテナーがListenするホスト名とポートが指定されています。

それ以外は全てシステムのデフォルト値が暗黙的に定義されています。デフォルトを変更する場合のみ明示的にserver.xmlに記載します。記載がない場合は、デフォルト値が有効になるため、構成ファイルをシンプルに保つことができています。

03. Libertyプロファイルによる開発・構成

server.xmlの定義：使用フィーチャーの定義

IBM

- <featureManager> および <feature>
  - <featureManager>下の<feature>で利用するフィーチャーを定義し、有効化
  - フィーチャーには依存関係がある
  - 前提となるフィーチャーは暗黙的に有効となる
    - 例）jsp-2.2はservlet-3.0が前提となっている
  - 有効となっていないフィーチャーは、アプリケーションは使用できない

server.xml

```
...
<!-- Enable features -->
<featureManager>
    <feature>jsp-2.2</feature>
    <feature>jdbc-4.0</feature>
</featureManager>
...
```

Developer Toolsのフィーチャー構成画面

<featureManager>の下には、<feature>エレメントを追加して構成していきます。

各<feature>には依存関係が定義されていて、前提となるフィーチャーは暗黙的に有効にされます。たとえば、jsp-2.2に関してはservlet-3.0が前提となっているため、jsp-2.2を有効化すると、暗黙的にservlet-3.0が有効化されます。

server.xmlでは、明示的に有効化したフィーチャーしか記述されませんが、WebSphere Developer Toolsのフィーチャー構成画面（後述）では、暗黙的に有効とされているフィーチャーに関しても表示することが可能です。

03. Libertyプロファイルによる開発・構成

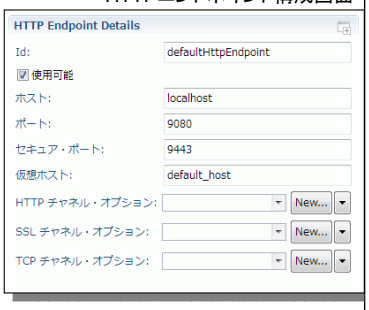
server.xmlの定義: HTTPエンドポイントの定義

- <httpEndpoint>
  - サーバーがLISTENするポート、提供するHTTPサーバー・サービスの定義
  - 詳細の設定は、子エレメントで指定可能
    - <accessLogging>
      - アクセス・ログのパス、出力フォーマット、ファイル・サイズ、ファイル数
    - <httpOptions>
      - 読込み/書込みタイムアウト、KeepAliveの設定
    - <sslOptions>
      - 使用するSSL定義（SSLレパートリー）の設定
    - <tcpOptions>
      - 非活動タイムアウト

server.xml

```
...
<httpEndpoint
 host="localhost"
 httpPort="9080"
 httpsPort="9443"
 id="defaultHttpEndpoint" />
...
```

Developer Toolsの
HTTPエンドポイント構成画面



© 2013 IBM Corporation

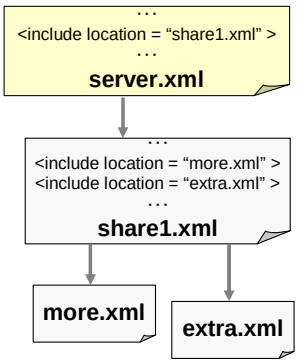
次に<httpEndpoint>です。サーバーのWebコンテナーがListenするポートを指定します。

デフォルトではホスト名とポートだけですが、その他の詳細な設定は、子エレメントで指定していきます。

たとえば、Webコンテナーのアクセスログのフォーマットやサイズなどは、<accessLogging>エレメントで指定していきます。他にもHTTPやTCPのタイムアウトなども子エレメントの中で指定していきます。

server.xmlの定義：その他Tips

- 柔軟な構成：構成ファイルのインクルードが可能
  - 複数サーバーで定義を共有する場合、チーム開発などで有効
    - サーバーインスタンスに依存しない定義（DB定義など）
  - <include location="dir/file">で組込むべきファイルを指定
    - dirを相対パスで指定した場合以下の順で検索される
      - ・<include>が書かれた構成ファイルのディレクトリー
      - ・${server.config.dir}
      - ・${shared.config.dir}



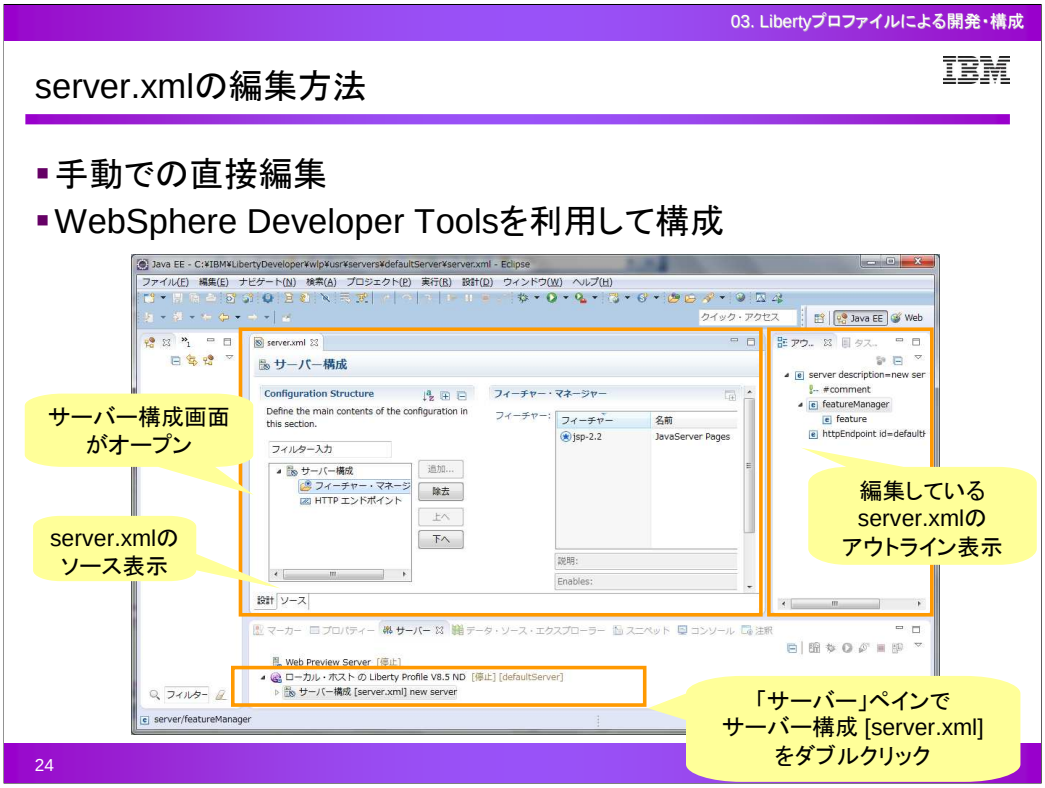
- 変数を使用した記述（${～}）が可能
  - 製品事前定義 変数（${server.config.dir}等）
  - bootstrap.propertiesで指定した値
- 時間を指定する属性値には時間単位をつける
  - 時間単位としては時間 (h)，分 (m)，または秒 (s) が使用可能
    - 一部の設定ではミリ秒（ms）も使用可能
    - たとえば30秒は「30s」と表す
    - 複数の時間単位を使用して「1m30s」というような表記も可能

© 2013 IBM Corporation

さらにserver.xmlは、構成ファイルの一部を別XMLファイルに外出しにすることが可能です。

たとえば、データ・ソースへのアクセスするための定義など、サーバー・インスタンスごとに変わらない共通する定義は、別のXMLファイルとして外出しし、<include>エレメントでserver.xmlに取り込むことが可能です。このように共通する構成ファイルをコンポーネント化することで管理の容易化を実現できます。またチーム開発などの場合も、サーバー・インスタンスに依存しない共通構成ファイルを配ることで、簡単に構成の共通化を実現することが可能です。

その他、bootstrap.propertiesで定義した変数を構成ファイルに利用することができます。従来のWebSphereではタイムアウトを指定する場合は単位を確認する必要がありましたが、Livbertyではタイムアウト関連パラメータは、hやm、sなど単位を指定します。たとえば3分は3mと定義することができますし180sと定義することもできます。



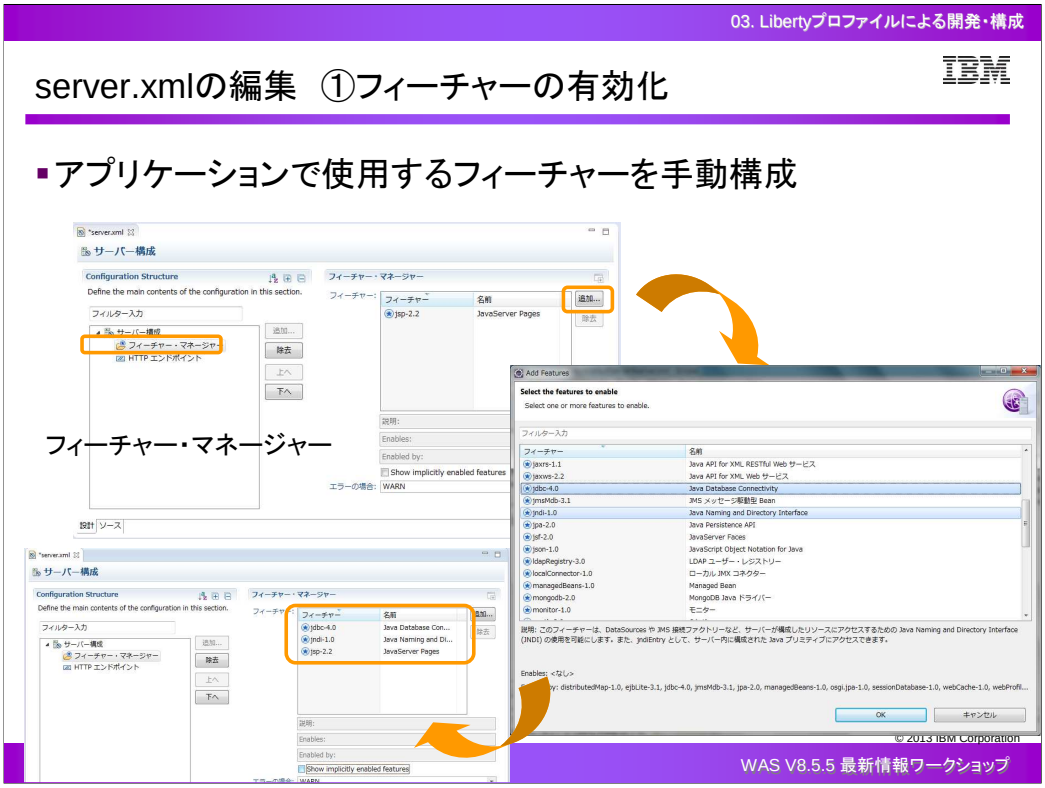
server.xmlの編集方法としては、2つあります。

特にツールが生成するIDといったものはserver.xmlには含まれませんので、手編集でserver.xmlを編集することもサポートされています。

ただ実際には全て手作業で編集することは辛いので、先ほどご紹介したWebSphere Developer Toolsを利用してsevrer.xmlを構成し、いったん作成したsever.xmlを手で編集してタイムアウト値を修正するといったシナリオが現実的かと思います。

WebSphere Developer Toolsの使い方としては、サーバー・ペインに表示されているLibertyプロファイルのサーバー・インスタンスのサーバー構成（server.xml）をダブル・クリックすると、サーバー構成ペインが開きますので、こちらを利用して編集していきます。

こちらでソース画面を表示させれば、編集を行っているserver.xmlのソースを表示させることもできますし、左のアウトラインのペインにはserver.xmlの大枠が表示されます。



server.xmlで編集・構成していくもとのしては、フィーチャーとサーバー構成の2つがあります。

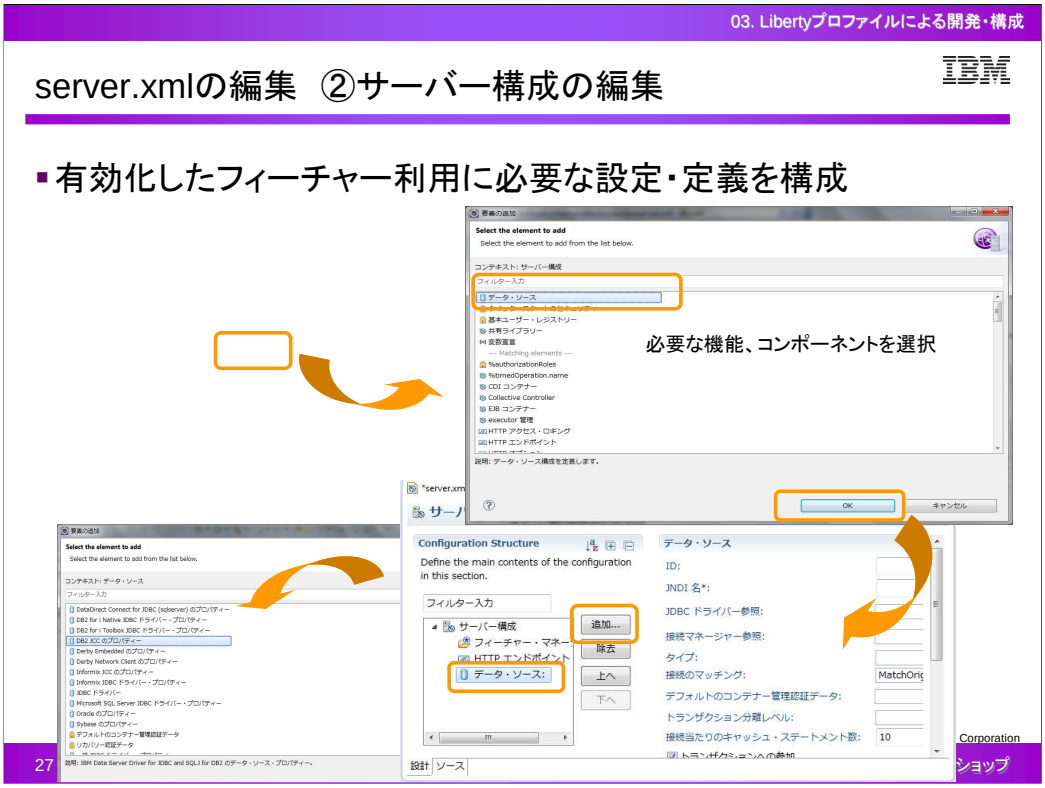
フィーチャーとは、ServletやJDBC、JMSなど使用するプログラミング・モデルや動的キャッシングのサービスです。

使用するプログラミング・モデルや機能を選び、自分自身でフィーチャー・マネージャーの追加ボタンから使用するプログラミング・モデル（フィーチャー）を追加していきます。

また、WebSphere Developer Toolsが入っている環境であれば、アプリケーションを導入したタイミングで、アプリケーションで使用されているプログラミング・モデルを解析し、server.xmlを自動的に編集しフィーチャーを自動的に追加することも行われます。

たとえば、上の例ではjspとservletのみのWebコンテナー機能しか有効となっていませんが、EJBアプリケーションをデプロイすることで、WDTが判断してserver.xmlにEJBLiteフィーチャーを追加し、ダイナミックにフィーチャーが有効化されます。

但し、自動構成が行われないフィーチャーもあるので、注意が必要です。



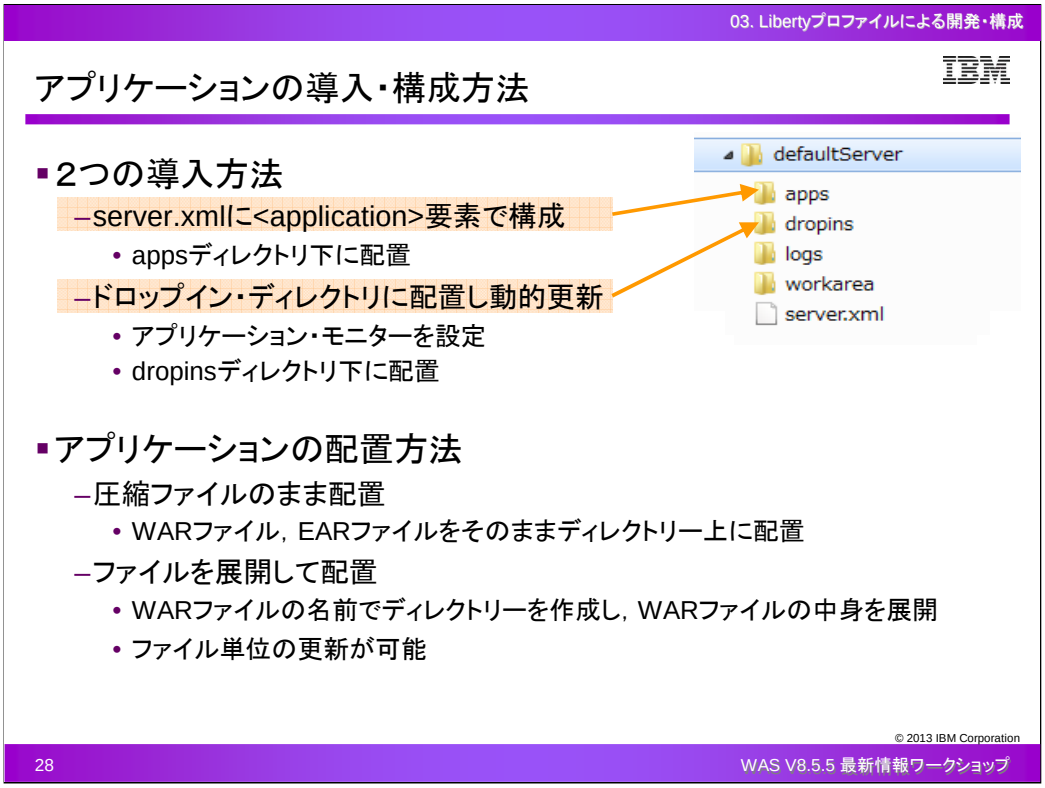
フィーチャーの次に、サーバーの構成です。

たとえば、Webコンテナーの設定をデフォルトから変更したり、JDBCのデータソースの設定を行ったりといった作業があります。

これらに関しては、環境に応じて設定していくものですので、自動的に構成はされません。明示的にコンポーネントを選びながら、追加していく形になります。

様々なフィーチャーとそれらに必要な構成については、次の節で見ていきます。

その前に、もう一つ大事なアプリケーションの導入と構成について確認していきます。



Libertyでのアプリケーションの導入方法には2つあります。

1つ目は、server.xmlに<application>エレメントを記載していく方法です。もう1つは、ドロップイン・ディレクトリに配置することで、アプリケーションの導入を行う方法です。

配置の方法ですが、圧縮ファイルのまま各ディレクトリに配置することもできますし、ファイルを展開して配置することも可能です。

03. Libertyプロファイルによる開発・構成

アプリケーション導入方法1: sevrer.xmlでの構成

- <application>要素により指定
  - location属性は必須。絶対パス・相対パスやURLで記述
  - 子要素としてセキュリティー・ロールのバインディングやライブラリー参照を指定することが可能
- location属性を相対パスで指定した場合，${server.config.dir}/apps，${shared.app.dir}から検索される

server.xml

```
<application
  id="SupportTools" location="SupportTools.war"
  name="Support Tools" type="war"
  context-root="/support" />
```

Developer Toolsのアプリケーション構成画面

<application>要素にネストして
<application-bnd>要素や
<classloader>要素を定義できる

© 2013 IBM Corporation

まず、<application>エレメントを指定して導入する方法です。

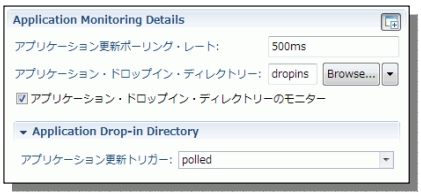
アプリケーションに関して、セキュリティ属性やクラスローダーや共有ライブラリの設定などが必要な場合に利用します。また頻繁な更新が発生しない本番環境ではこちらで明示的に構成を行ってアプリケーションを導入していきます。

アプリケーションのロケーションを指定する際に、相対パスで指定した場合、サーバー構成ディレクトリ下のアプリケーション・ディレクトリや共有アプリケーション・ディレクトリから検索されます。

03. Libertyプロファイルによる開発・構成

アプリケーション導入方法2:ドロップイン・ディレクトリ

- コード開発段階でのアプリケーションや、アプリケーション・セキュリティー・ロール等追加構成が必要ないアプリで利用可能
- アプリケーション・モニターを構成
  - <applicationMonitor>要素で指定
  - デフォルトは${server.config.dir}/dropins



- ドロップイン・ディレクトリーにファイルをコピーする
  - 拡張子により自動的にタイプ(EAR/WAR/WAB)を判別
    - WAR/WABは、拡張子を除いたファイル名がアプリケーションのコンテキストルートとなる
      例) Snoop.war は http://localhost:9080/Snoop/ でアクセス
- アプリケーションのアンインストール
  - ファイルを削除すると、自動的にアプリケーションがアンインストールされる

© 2013 IBM Corporation

次にドロップイン・ディレクトリで指定する方法です。さきほど見たようなセキュリティーなど詳細な設定が不要な場合は、こちらを利用できます。特に開発段階などで頻繁なアプリケーションの更新が発生する場合には、こちらの導入方法が適しています。

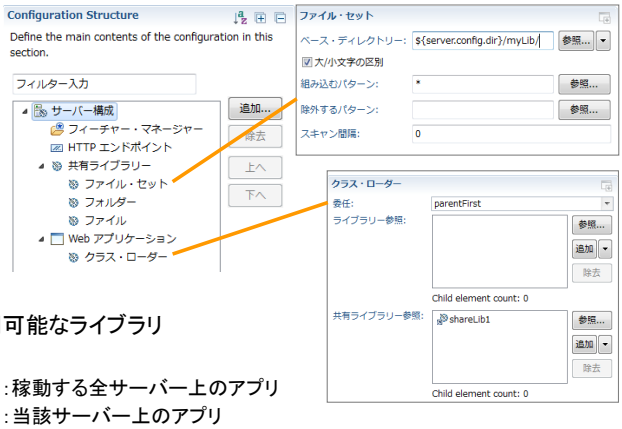
サーバーの構成の中で、アプリケーション・マネージャーというコンポーネントを有効にし、アプリケーションをドロップインするディレクトリを指定します。

ディレクトリから削除することでアプリケーションはアンインストールされます。

03. Libertyプロファイルによる開発・構成

アプリケーションから参照するライブラリの管理

- 共有ライブラリ
  - 明示的にライブラリを構成
    - ファイル・セット、フォルダー、ファイル
  - クラス・ローダーから明示的に関連づけ
- グローバル・ライブラリ
  - 全てのアプリケーションから使用可能なライブラリ
  - 以下の2箇所に配置可能
    - ${shared.config.dir}/lib/global　:稼動する全サーバー上のアプリ
    - ${server.config.dir}/lib/global　:当該サーバー上のアプリ
  - 以下の場合にライブラリを参照
    - 明示的にクラス・ローダー定義がない場合
    - 明示的にクラス・ローダー定義が存在し、関連付けている場合
  - BestPractice: クラスローダーの分離と依存性明確化のため、明示的にクラスローダー定義を構成



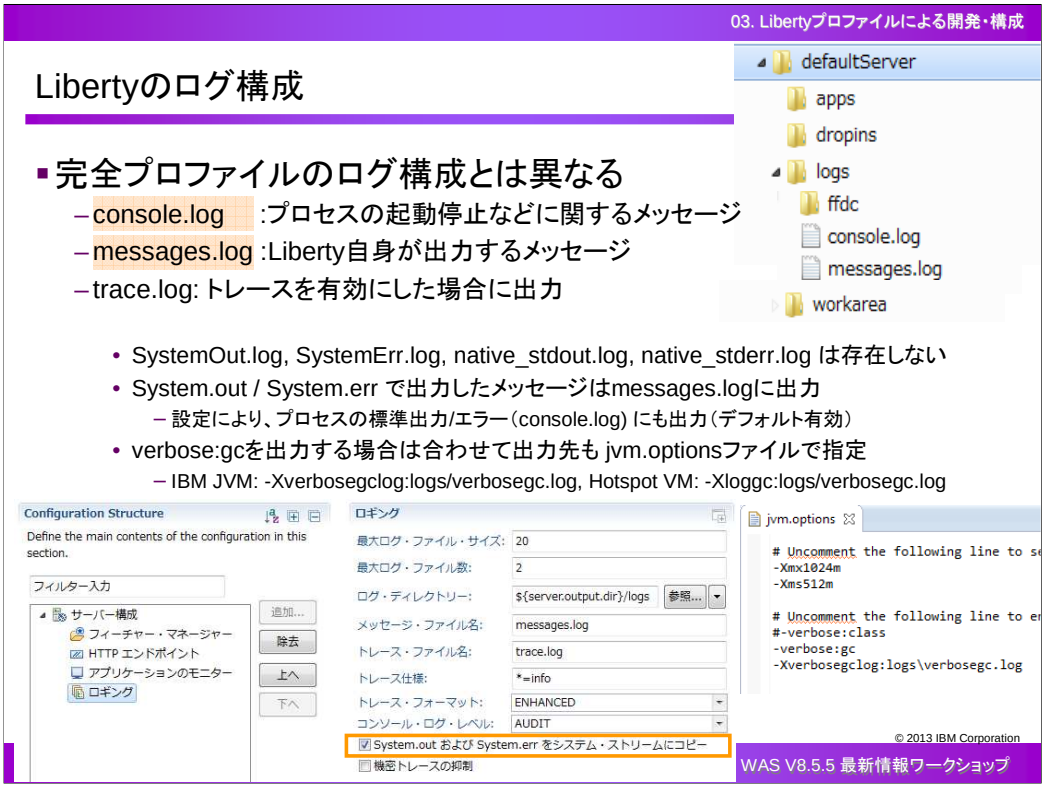
© 2013 IBM Corporation

アプリケーションが参照するライブラリの管理についてご説明します。

まず共有ライブラリです。明示的にライブラリとして構成し、それを共有ライブラリとしてクラスローダーの管理画面から関連付けます。

もう一つの定義方法が、グローバル・ライブラリです。これは全てのアプリケーションから使用可能となるライブラリであり、スコープにより2つの配置場所があります。稼動する全サーバー上のアプリで共有する場合は、共有構成ディレクトリ下の ${shared.config.dir}/lib/global に配置します。サーバー・インスタンス内で稼動するアプリケーション内で共有する場合には ${server.config.dir}/lib/global に配置します。このディレクトリに配置されたライブラリは、アプリケーション内に明示的にクラスローダー定義がないアプリケーションに関しては自動的に参照されます。また明示的なクラスローダー定義が存在する場合も、明示的に関連付けを行っている場合はライブラリを参照します。

なお暗黙的に参照はされますが、クラスローダーの分離を依存性の明確化のために、明示的にクラスローダー定義を構成して頂くことがBest Practiceとされています。



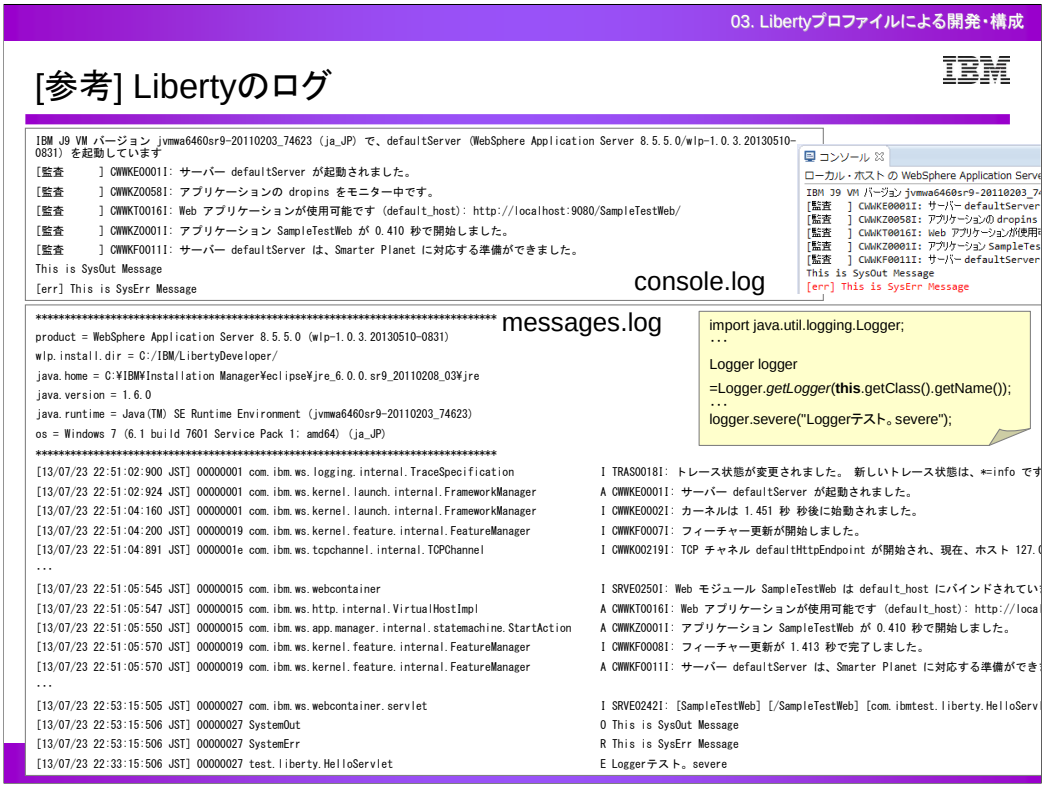
次にLibertyのログ構成についてみていきます。

従来の完全プロファイルのログ構成とは異なり、おなじみのSystemOut.log/SystemErr.log,native_stdout.log,native_stderr.logファイルは存在しません。

メインとなるのはconsole.logとmessage.logです。console.logはプロセスの起動停止など標準出力に出されるメッセージが出力されます。messages.logにこれまでSystemOut.logやSystemErr.logに出力されていた製品に関するメッセージが出力されます。

アプリケーションの中で、System.out.println()やSystem.err.println()で出力したメッセージはmessage.logに出力されます。設定によりプロセスの標準出力/標準エラー(console.log) にも出すことができます。デフォルトで有効になっていますので、標準出力/標準エラーに出す必要がない場合は、ロギングコンポーネントから設定を外してください。

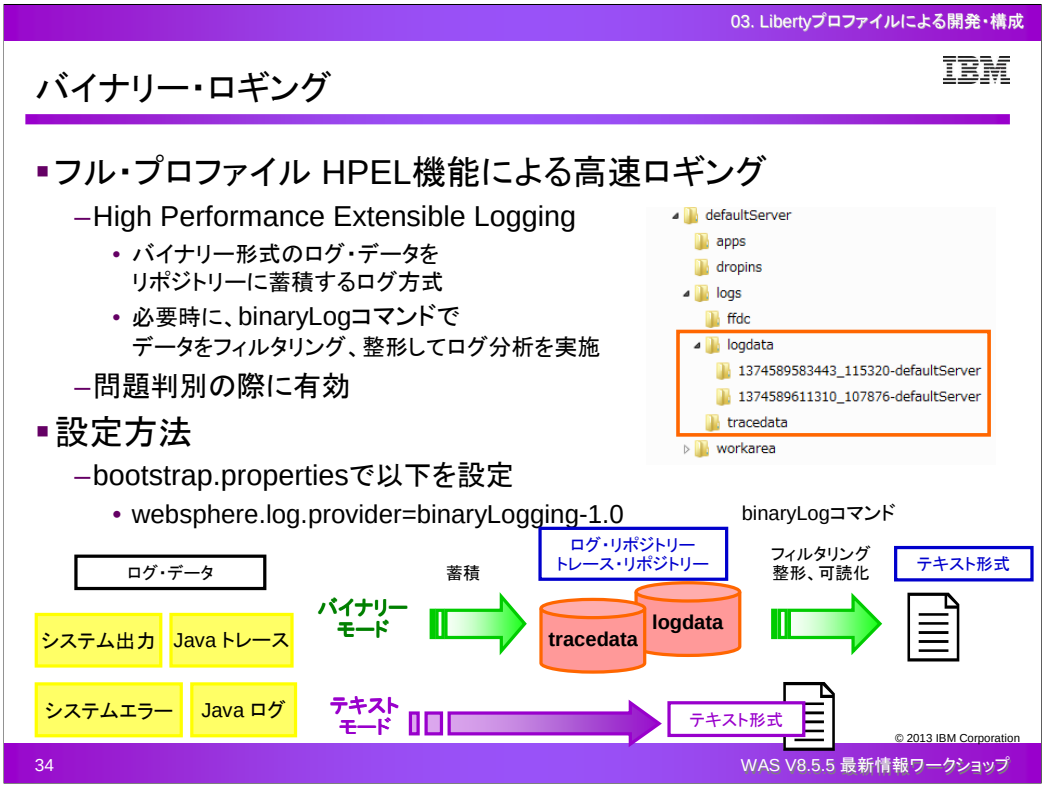
また多くのお客様で、パフォーマンス・チューニングのために冗長GCの取得をお願いしていますが、これはLibertyProfileではjvm.optionsファイルで明示的に指定して有効化します。この際に、各JVMに応じた方法でファイル名も指定してください。



こちらがLibertyのログのサンプルです。console.logはその名の通り、eclipseのconsole.logにも表示されます。

message.logには従来のSystemOutに出力されていたメッセージが出力されます。

Java Util Loggingのロガーを使って出力したメッセージは、このmessage.logに出力されます。



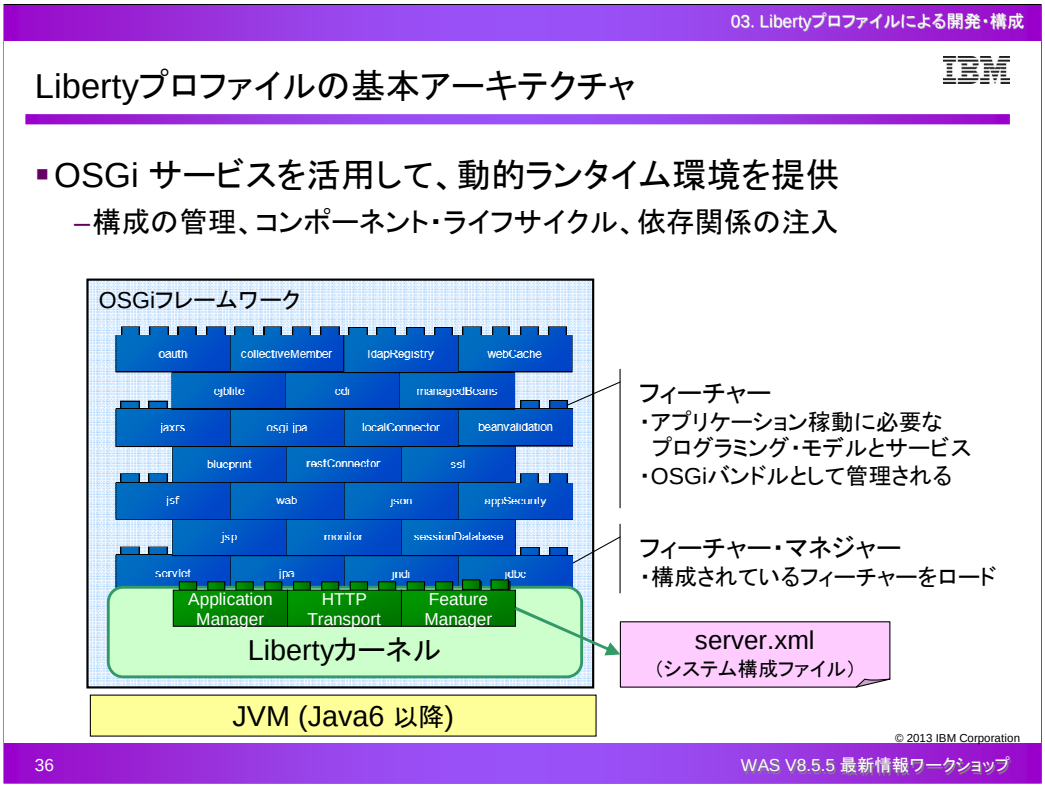
最後にバイナリー・ロギングの機能です。これは、フル・プロファイルで提供されている High Performance Extensible Logging機能をインポートした機能になります。

バイナリー形式でログ・データを出力することで、ハイ・パフォーマンスなログ出力が可能です。トレースを有効化すると、トレース処理が重くて問題が再現しなくなるなどの課題に対応可能です。

一旦、レポジトリーにバイナリー形式でログが出力されますので、binaryLogコマンドを利用してメッセージをフィルタリング・整形して利用します。

設定方法としては、bootstrap.propertiesファイルでwebsphere.log.provider=binaryLogging-1.0 を指定することで有効化します。

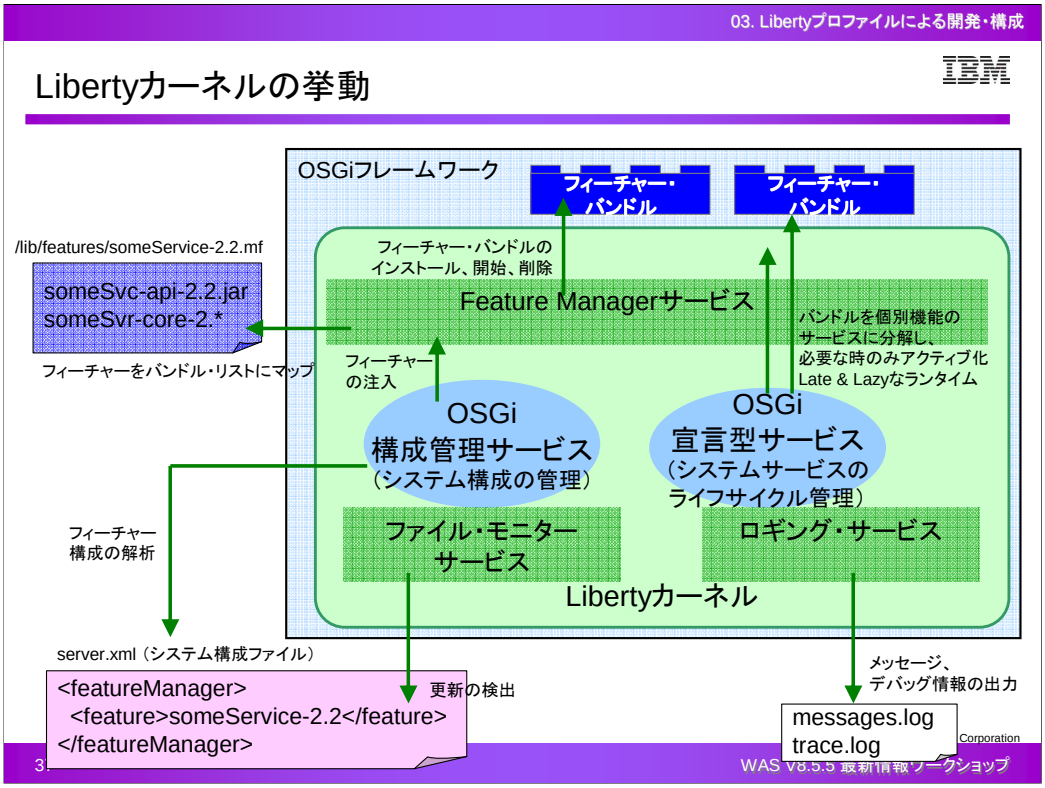
それでは、Libertyにおいて個々のフィーチャーをどのように構成するかみていきましょう。



まず、フィーチャーの利用を可能にしている基本的アーキテクチャーついて確認してきます。

まずLibertyカーネルが起動されると、OSGiのフレームワークが開始されます。OSGiは、古いJavaの世界では欠けていた動的なコンポーネントの追加や実行を可能にしているフレームワークであり、Eclipseなどでも利用されています。

Libertyカーネルの中ではFeature Managerと呼ばれるコンポーネントが稼動しており、server.xmlの内容を解析して、構成されているフィーチャーをロードします。ロードされる個々のフィーチャーはアプリケーション稼動に必要なプログラミング・モデルやサービスであり、OSGiのコンポーネントの単位であるOSGiバンドルとして管理されています。



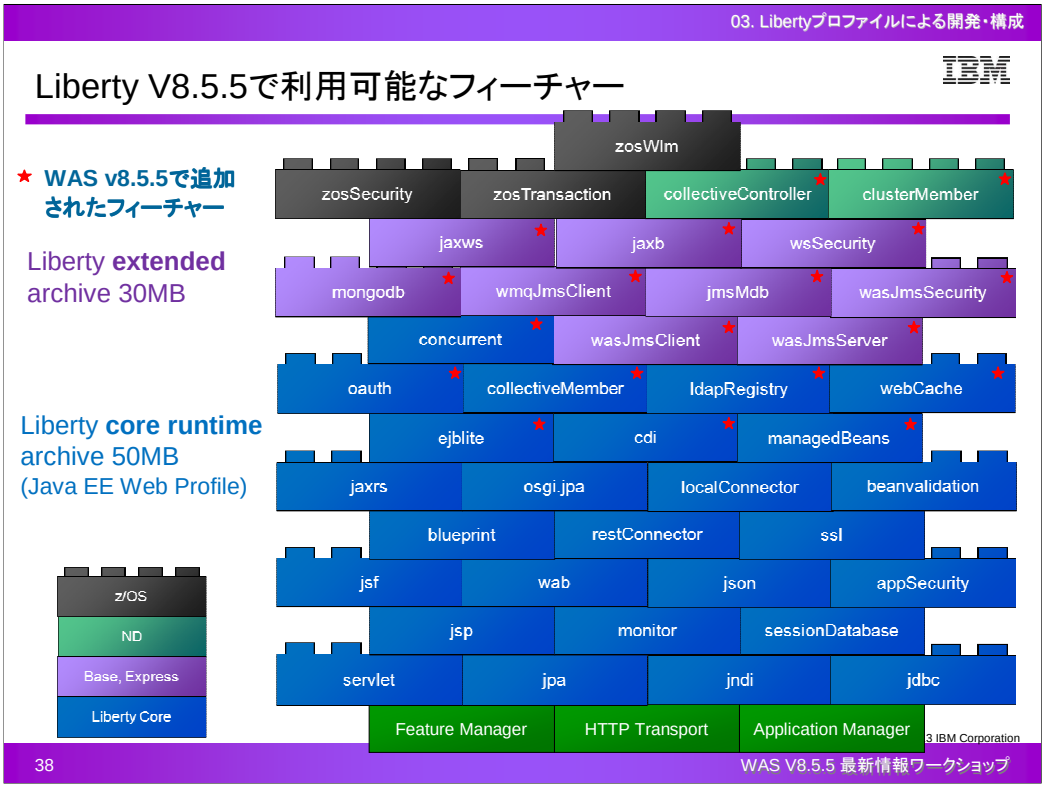
Libertyカーネルの挙動について、もう少しだけ詳しく見てみます。

Libertyカーネルで稼動するOSGi 構成管理サービス（OSGi Config Admin Service）がserver.xmlを読込み、必要とされるフィーチャーの解析を行い、FeatureManagerサービスに必要なフィーチャーを伝えます。FeatureMangerは伝えられたフィーチャーをOSGiのバンドルにマップし、そのフィーチャーのバンドルを動的にインストールします。

Libertyカーネル上では、ファイル・モニター・サービスも稼動しており、server.xmlの変更を監視していますので、server.xmlに変更が行われると、再びフィーチャーの解析が行われ、OSGiの構成管理コンポーネントによりフィーチャーが注入されます。

またOSGiの宣言型サービスを利用することにより、それぞれのサービスの依存性が明確化されています。これにより、バンドルの全てを起動時にアクティブにするのではなく（Late）、依存性のあるサービスのみを必要な時に有効化する（Lazy）ことで、軽量なランタイム環境を実現しています。

またLibertyカーネルにはロギング・サービスが統合されており、メッセージやデバイス情報の出力が行われます。



Libertyで利用可能なフィーチャーです。

青色のコンポーネントがLiberty Coreで提供されるLiberty Coreランタイムです。Coreランタイム全体で50MB程度しかありません。このCoreランタイムでは、JavaEE のWeb Profile機能を提供します。さらには、次のセッションで取り上げる集中管理のための仕組みCollectiveのメンバーになるためのフィーチャーも提供します。

紫色のコンポーネントが、Express, Baseで提供されるコンポーネントです。ここにはWASでメッセージングを利用する際に必要となるJMSの仕組みやJAXWSやJAXBなどWebサービスを利用するためのフィーチャーが含まれます。さらにドキュメント・データベースとしてmongodbクライアントとしてのフィーチャーも統合されています。緑色のコンポーネントが、Network Deployment版で提供されるコンポーネントです。集中管理のコントローラーとなるためのcollectiveController、またクラスターを構成するためのclusterMemberという管理のためのフィーチャーが提供されています。

zOS（黒色のコンポーネント）では、zOSのためにzOSWlmやzOSSecurityといったzOS用のコンポーネントが提供されています。

03. Libertyプロファイルによる開発・構成

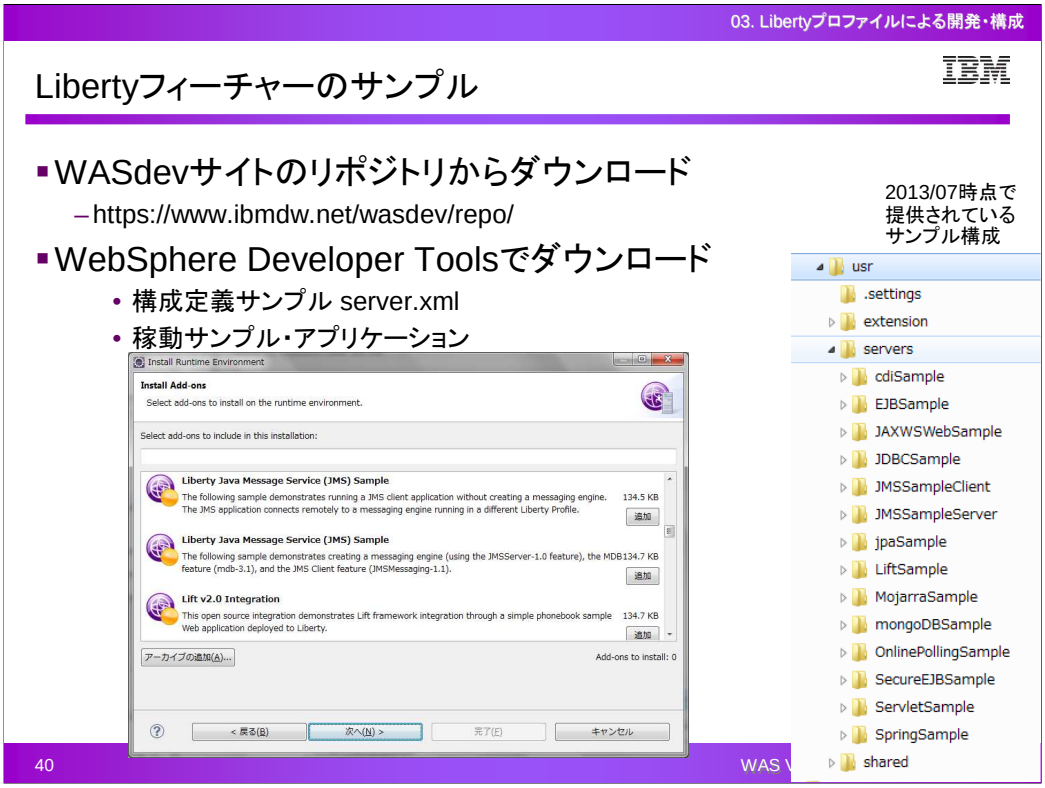
[参考] Liberty (ND版)で利用可能なフィーチャー一覧

IBM

<Liberty_ND>¥bin>productInfo featureInfo

appSecurity-1.0 [1.1.0]		
appSecurity-2.0 [1.0.0]	jmsMdb-3.1 [1.0.0]	restConnector-1.0 [1.0.0]
beanValidation-1.0 [1.0.0]	jndi-1.0 [1.0.0]	serverStatus-1.0 [1.0.0]
blueprint-1.0 [1.0.0]	jpa-2.0 [1.0.0]	servlet-3.0 [1.0.0]
cdi-1.0 [1.0.0]	jsf-2.0 [1.0.0]	sessionDatabase-1.0 [1.0.0]
clusterMember-1.0 [1.0.0]	json-1.0 [1.0.0]	ssl-1.0 [1.0.0]
collectiveController-1.0 [1.0.0]	jsp-2.2 [1.0.0]	timedOperations-1.0 [1.0.0]
collectiveMember-1.0 [1.0.0]	ldapRegistry-3.0 [1.0.0]	wab-1.0 [1.0.0]
concurrent-1.0 [1.0.0]	localConnector-1.0 [1.0.0]	wasJmsClient-1.1 [1.0.0]
distributedMap-1.0 [1.0.0]	managedBeans-1.0 [1.0.0]	wasJmsSecurity-1.0 [1.0.0]
ejbLite-3.1 [1.0.0]	mongodb-2.0 [1.0.0]	wasJmsServer-1.0 [1.0.0]
jaxb-2.2 [1.0.0]	monitor-1.0 [1.0.0]	webCache-1.0 [1.0.0]
jaxrs-1.1 [1.0.0]	oauth-2.0 [1.0.0]	webProfile-6.0 [6.0.0]
jaxws-2.2 [1.0.0]	osgi.jpa-1.0 [1.0.0]	wmqJmsClient-1.1 [1.0.0]
jdbc-4.0 [1.0.0]	osgiConsole-1.0 [1.0.0]	wsSecurity-1.1 [1.0.0]

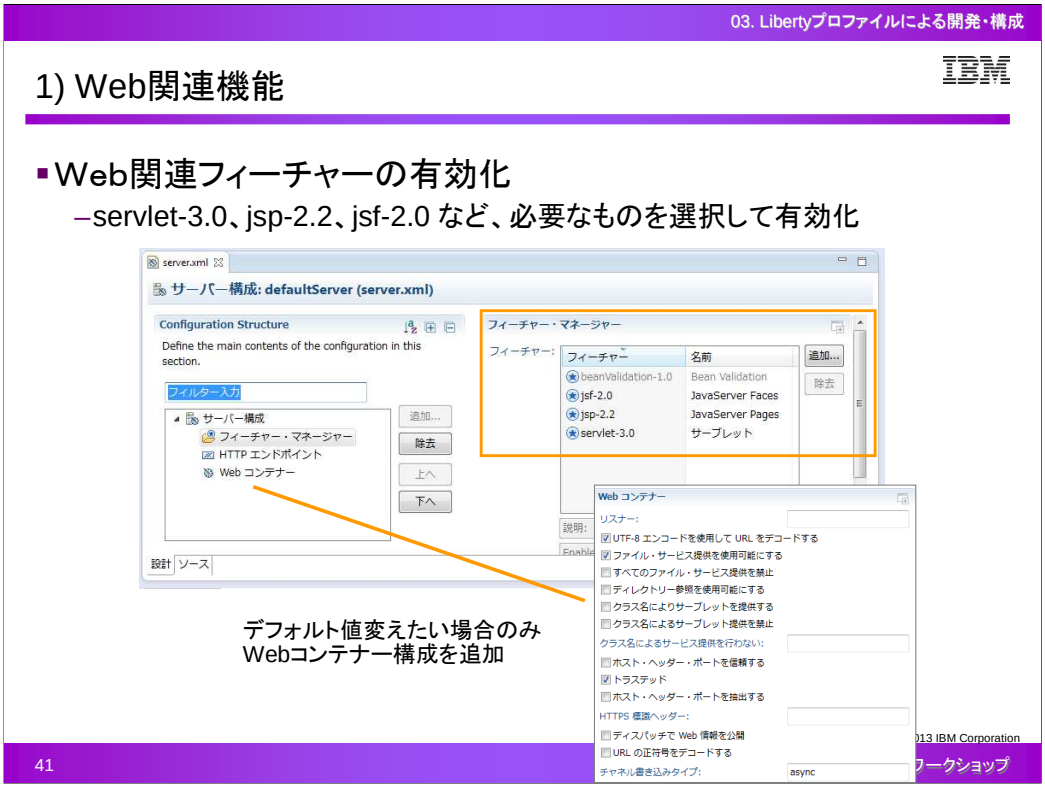
こちらがLibertyで利用可能なフィーチャーについて、LibertyのbinディレクトリにあるproductInfoコマンドをfetatureInfoオプションを指定して、出力した結果です。（Network Deployment版での出力結果です）



これから個々のフィーチャーについて見ていきますが、実際にご自身の環境でゼロから構成を作っていくのは、従来のWebSphereやJava EEの知識がないと少し難しく感じるかもしれません。

そのような場合には、多様なサンプルが提供されています。

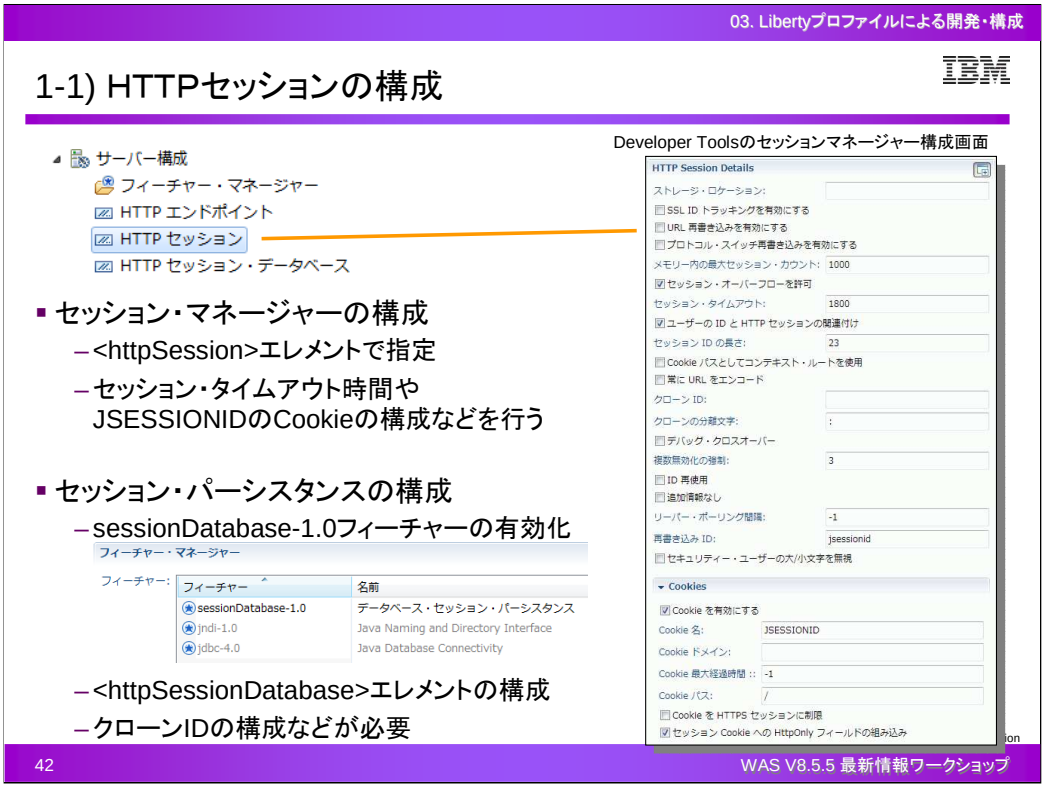
WebSphere Developer Toolsダウンロード時に追加選択できるコンポーネントとしても選択できますし、https://www.ibmdw.net/wasdev/repo/ からもダウンロードできます。これらのサンプルにはサーバーの構成定義のサンプルと、そのまま稼動させ、またコーディングの中身を確認することができるサンプル・アプリケーションが含まれます。使い方が分からないフィーチャーなどがあった場合には、このようなサンプルを確認して頂くことでフィーチャーの利用の仕方がクリアになるはずです。



それではここから、いくつかの基本的なフィーチャーについて構成の仕方について確認していきたいと思います。

まずLiberty Coreで提供されるWebコンテナー、DBアクセス、EJB、そしてExpress, Baseで提供されるメッセージング、Webサービス、mongodbの順に見ていきます。

まずWeb関連フィーチャーです。これはシンプルに必要なフィーチャー追加することで利用可能です。デフォルト値から設定を変更したい場合のみ、Webコンテナーをサーバー構成に追加して設定の変更を実施します。



Webコンテナーで、HTTPセッションを利用する場合にはHTTPセッション・マネージャーを構成に追加します。

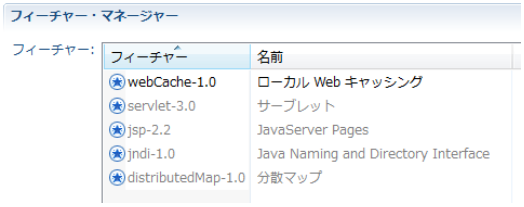
従来のWebSphere環境では最初からコンテナーの構成に統合されていましたが、Libertyでは構成に追加して使用します。個々の設定情報は従来のWebSphere環境とほぼ同じですので、理解しやすいものです。

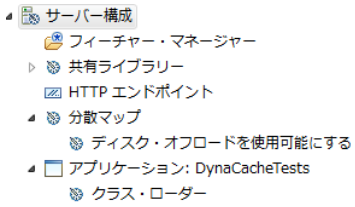
Liberty CoreやExpress, Baseなどで、シンプル・クラスターを行う場合には、手作業でクローンIDを設定することが必要です。またセッションDBによるセッション・パーシスタンスを利用するためには、httpSessionDatabaseエレメントを構成し、JDBCなどのコンポーネントの設定を行うことが必要になります。

03. Libertyプロファイルによる開発・構成

1-2) 動的キャッシング機能

- webCache-1.0 の有効化
  - サーブレット・キャッシュ
    および オブジェクト・キャッシュ
    - フル・プロファイル（従来型WAS）と異なり、いずれのキャッシュ・インスタンスとしても稼動
    - ディスク・オフロード先は、各サーバーのworkarea領域
  - 複数サーバーでのキャッシュ複製はサポートされない
  - キャッシュの設定方法 3種類
    - server.xml　での構成
    - アプリケーションのWEB-INFまたはMETA-INF内に、cacheinstances.properties / distributedmap.properties ファイルを配置
    - com.ibm.wsspi.cache.DistributedObjectCacheFactory クラスで作成





© 2013 IBM Corporation

次にWAS8.5.5でLibertyに機能追加された動的キャッシング機能です。webCache-1.0というコンポーネントで利用可能になります。

従来の完全プロファイルの動的キャッシュで提供されていたサーブレット・キャッシュおよびオブジェクト・キャッシュの機能を持っています。完全プロファイルではサーブレット・キャッシュとオブジェクト・キャッシュは別のインスタンスとして定義していましたが、Libertyではいずれのキャッシュ・インスタンスとしても同じフィーチャーで利用可能です。LibertyにはDRS（Data Replication Service）は提供されていませんので、複数サーバー間でのキャッシュ複製機能は提供されていません。キャッシュの設定方法は、右の構成画面のようにserver.xmlで定義する方法やアプリケーションのWEB-INF/META-INF内に定義ファイルを用意する方法、さらにプログラミングで構成する方法などがあります。またCache Monitorなどの動的キャッシュの管理アプリケーションも提供されていますので、サンプルを落として確認してください。

03. Libertyプロファイルによる開発・構成

2) データベース・アクセス

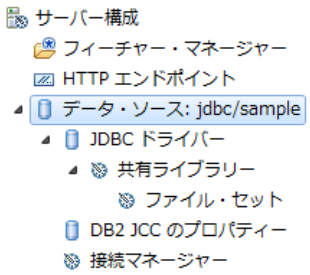
- JDBC 4.0の有効化
  - JNDIも暗黙的に有効化される

フィーチャー・マネージャー

フィーチャー:

フィーチャー	名前
jdbc-4.0	Java Database Connectivity
jndi-1.0	Java Naming and Directory Interface

- DBアクセスに必要な構成



✓JDBC リソースの構成を更新したら、サーバー再始動が必要

- データソースの定義
  - JNDI名、接続タイプ
  - 使用するJDBCドライバー、接続マネージャー
- JDBCドライバーの定義
  - JDBCドライバーのパス
- JDBCドライバー・プロパティの設定
  - JDBCドライバー・タイプ
  - 接続先ホスト名、ポート
- 接続マネージャーの設定
  - 接続プール、タイムアウト

© 2013 IBM Corporation

データベース・アクセスのための定義です。データベース・アクセスのためにはJDBC4.0を有効化します。JDBC4.0を有効化すると、暗黙的にデータソースのルックアップの際に利用するJNDIも暗黙的に有効化されます。

DBに必要な構成としては、まずデータソースの定義を作成します。データソースにはルックアップの際に使用するJNDI名を指定し、さらにJDBCドライバーや接続プールの接続マネージャーを関連付けます。これらについて詳しく見ていきます。

03. Libertyプロファイルによる開発・構成

2-1) データソースの定義

IBM

- データソース
  - JNDI名が必須
  - JDBCドライバーと接続マネージャーを関連付け
  - 接続のタイプ
    - 下表の種類から指定

未指定の場合は、以下の順序で検索	接続プール	2フェーズ対応
javax.sql.ConnectionPoolDataSource	○	×
javax.sql.DataSource	×	×
javax.sql.XADataSource	○	○

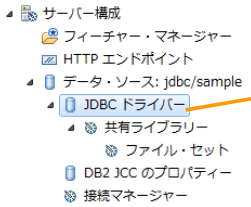
© 2013 IBM Corporation

データソースでは、ルックアップの際に使用するJNDI名を必須項目として指定します。さらに利用するJDBCドライバーと接続マネージャーを設定します。

従来の完全プロファイルのデータソースでは、非XAのデータソースとXAのデータソースは別のデータソースとして定義を行いましたが、Libertyではタイプを変更することで振舞いを切り替えることが可能です。接続プール機能を提供する非XAのデータソースがデフォルトです。次に接続プール機能を提供しない非XAのデータソース用のドライバーが検索されます。それでも適合するものがない場合は、接続プールを提供するXAのデータソース用のドライバーが検索されます。

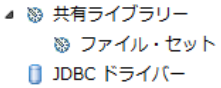
2-2) JDBCドライバーの定義

- JDBCドライバー
  - JDBCドライバーエレメント内で定義




```
<jdbcDriver>
    <library name="Lib_DB2_JCC_Driver">
        <fileset dir="${shared.resource.dir}/jdbc/db2"
                 includes="db2jcc4.jar, db2jcc_license_cu.jar"></fileset>
    </library>
</jdbcDriver>
```

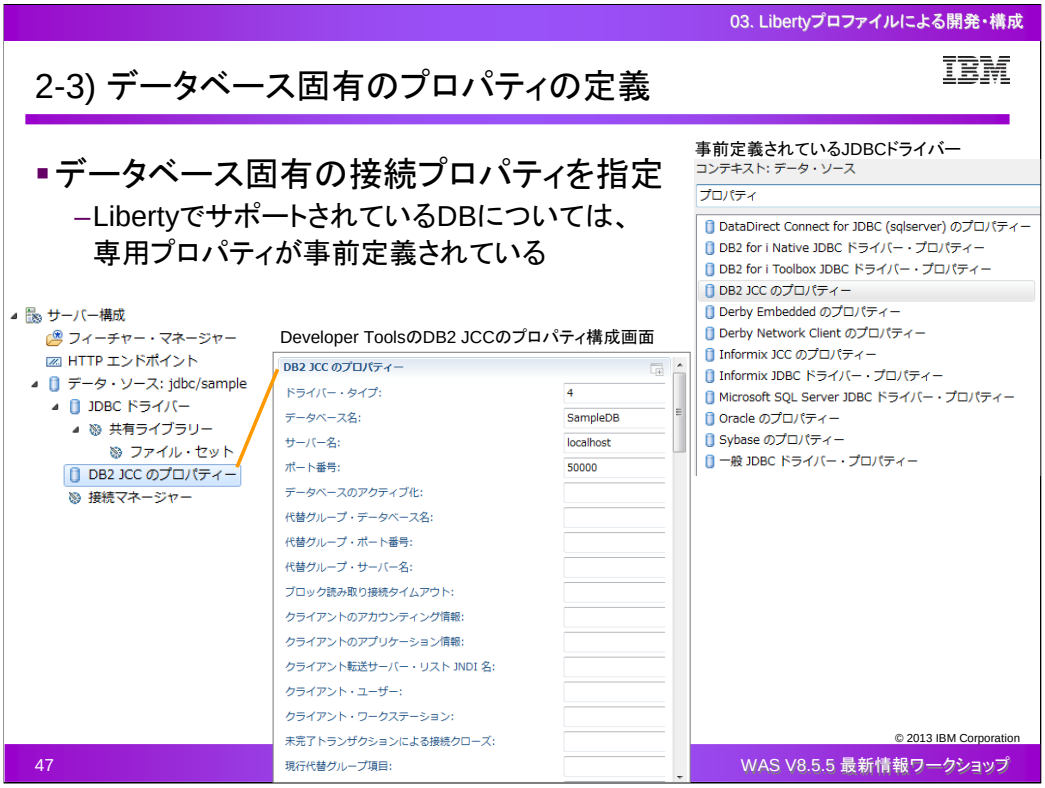
  - 別定義したライブラリーの参照を指定してのJDBCドライバー定義も可能




```
<library id="Lib_Derby_JDBC_Driver">
    <fileset dir="${shared.resource.dir}\jdbc\derby" </fileset>
</library>
<jdbcDriver libraryRef="Lib_Derby_JDBC_Driver" id="Derby JDBC Driver" />
```

© 2013 IBM Corporation

データソースに関連付けるものとして、実際にデータベースにアクセスする際に利用するJDBCドライバーがあります。事前定義していなければ、データソースに追加する形でJDBCドライバーのパスを指定することも可能ですし、別途事前にライブラリとして定義していれば、その参照をJDBCドライバーとして定義し、データソースに関連づけることも可能です。



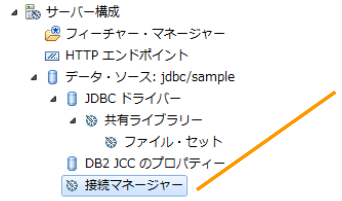
接続先のデータベース名やホスト名、ポートは、データベース固有プロパティとして設定します。

Libertyでサポートされているの DBについては、事前に専用プロパティのリストが用意されていますので、それを選択して構成します。サポートされないJDBCドライバーを利用する場合には、一般JDBCドライバーを利用します。

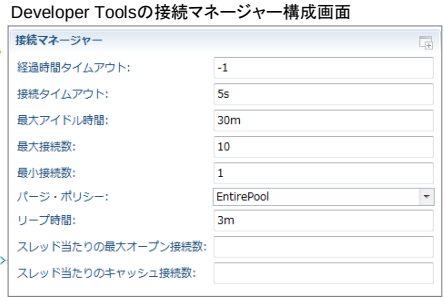
03. Libertyプロファイルによる開発・構成

2-4) 接続マネジャーの設定

- JDBCコネクション管理のカスタマイズ
  - DataSourceの子要素として定義するか、connectionManagerRefで指定
    - 全てデフォルトで使用する場合には明示的な構成は不要
    - タイムアウト系の設定は、0が即時、-1が無効（タイムアウトしない）



Developer Toolsの接続マネージャー構成画面

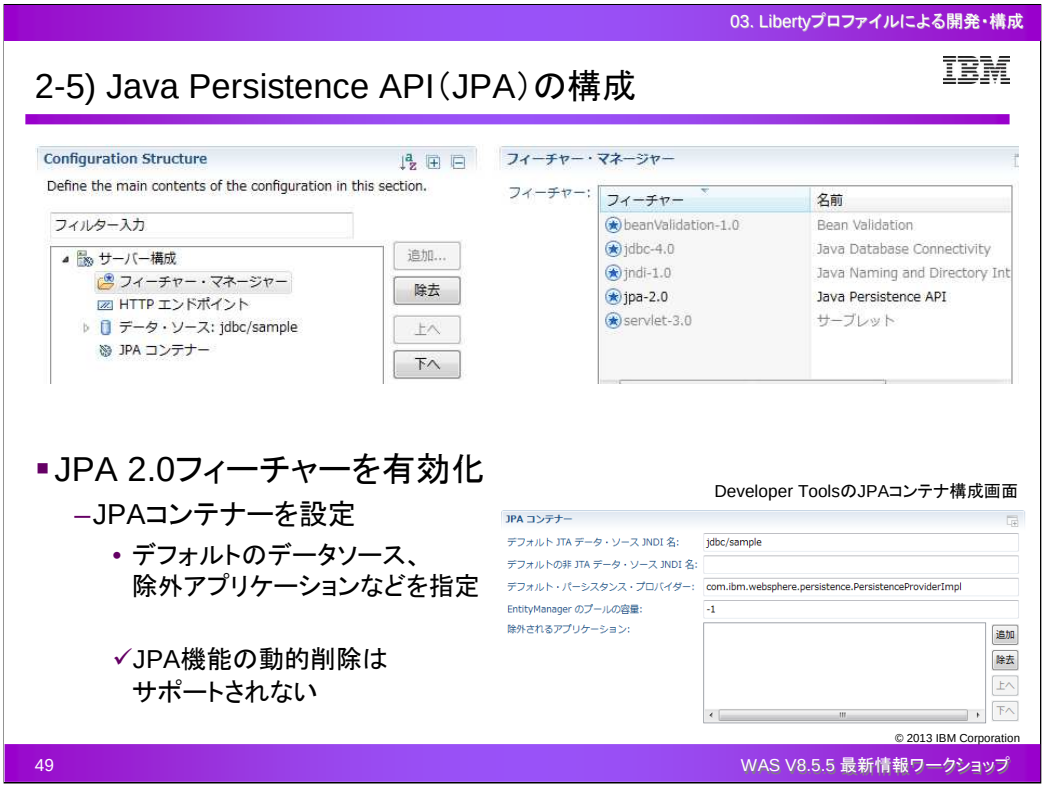



```
<dataSource jndiName="jdbc/sample" id="Sample DataSource">
<jdbcDriver>
        <library name="Lib_DB2_JCC_Driver">
            <fileset dir="${shared.resource.dir}/jdbc/db2"
                     includes="db2jcc4.jar, db2jcc_license_cu.jar">
        </library>
    </jdbcDriver>
<properties.db2.jcc databaseName="SampleDB"></properties.db2.jcc>
<connectionManager  maxPoolSize="10"
                    minPoolSize="1"
                    connectionTimeout="5s">
</connectionManager>
</dataSource>
```

© 2013 IBM Corporation

接続プールのマネージャーも設定を変更する場合には、明示的に構成に追加します。

なお、分散系DB2のデータソースでは、デフォルトでは、接続プールの最大数はデフォルトでは最大50、接続タイムアウトは30秒です。



最後にEJB3.0で導入されたJava Persistence APIの構成です。JPAを有効にすると、JDBCやJNDIなどが有効化されます。

構成にJPAコンテナーを追加し、JPAが利用するデフォルトのデータソースを設定します。なお、JPAフィーチャーの動的な削除はサポートされません。

3) EJB Liteの構成

- フルEJB3.1のサブセットを提供
  - セッションBean
    - stateless,statefull,singleton）
  - ローカル・インターフェース
- ejbLite-3.1の有効化

フィーチャー・マネージャー

フィーチャー:

フィーチャー	名前
ejbLite-3.1	Enterprise JavaBeans Lite
jndi-1.0	Java Naming and Directory Interface
servlet-3.0	サーブレット

EJB完全版

EJB Lite

Local Session Beans
Annotations / ejb-jar.xml
CMT / BMT
Declarative Security
Interceptors
JPA 2.0
JTA 1.1

Message Driven Beans
EJB Component
Web Service Endpoints
RMI-IIOP Interoperability
2.x/3.x Remote View
2.x Local View
Timer Service
CMP / BMP

  - ejb 3.x以前のEJBはサポートされない
  - ポータブルな名前空間を利用してください
    - java:global、java:app、java:module のいずれかのコンテキストにバインド
      - globalで一意： java:global[/app name]/module name/enterprise bean name
      - アプリ内で一意： java:app[/module name]/enterprise bean name
      - モジュール内で一意： java:module/enterprise bean name
    - (JVMスコープの）ejblocal の名前空間は検索されない

© 2013 IBM Corporation

次にEJBです。Libertyプロファイルでは、EJB3.1から提供されている EJBLiteをサポートしています。

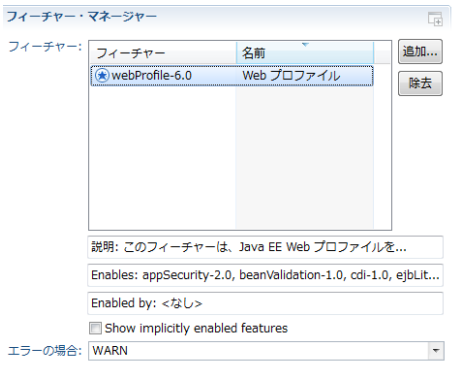
これはセッションBeanを中心としたEJB3.1フル仕様のサブセットです。リモート呼出しはサポートせず、ローカル呼出しのみをサポートしています。

EJB2.xのEJBやEntityBeanはサポートしません。JNDI名には、スコープにあわせて global,application, moduleのコンテキストの名前を指定します。

03. Libertyプロファイルによる開発・構成

4) JavaEE6 Webプロファイル

- JavaEE6 Webプロファイル
  - Webシステム開発において、利用頻度の高いAPIをまとめた軽量プロファイル
    - Servlet 3.0, JSP2.2, JSF 2.0
    - EJB Lite 3.1, JPA 2.0、JTA1.1 …



暗黙的に有効となっているフィーチャー

フィーチャー	名前
appSecurity-2.0	アプリケーション・セキュリティー
beanValidation-1.0	Bean Validation
cdi-1.0	Contexts and Dependency Injection
ejbLite-3.1	Enterprise JavaBeans Lite
jdbc-4.0	Java Database Connectivity
jndi-1.0	Java Naming and Directory Interface
jpa-2.0	Java Persistence API
jsf-2.0	JavaServer Faces
jsp-2.2	JavaServer Pages
managedBeans-1.0	Managed Bean
servlet-3.0	サーブレット
ssl-1.0	Secure Sockets Layer

© 2013 IBM Corporation

ここまでもろもろ見てきましたが、個別にフィーチャーを設定していくのではなく、Webシステム開発において利用頻度の高いAPIをまとめたJava EEのベストプラクティスともいえる軽量プロファイルWebプロファイルを利用することも可能です。

03. Libertyプロファイルによる開発・構成

5) メッセージングの構成

IBM

- メッセージング関連コンポーネント

メッセージ

フィーチャー	名前
wmqJmsClient-1.1	WebSphere MQ JMS クライアント
wasJmsServer-1.0	WebSphere Embedded Messaging サーバー
wasJmsClient-1.1	WebSphere Embedded Messaging JMS クライアント
jmsMdb-3.1	JMS メッセージ駆動型 Bean

- wasJmsServer-1.0
  - Liberty内部に組込みメッセージング・エンジンを構成
- wasJmsClient-1.1
  - wasJmsServerで構成したLiberty内の組込みメッセージング・エンジン、完全プロファイルのメッセージング・エンジン上のメッセージング・エンジンに、JMS1.1でアクセス

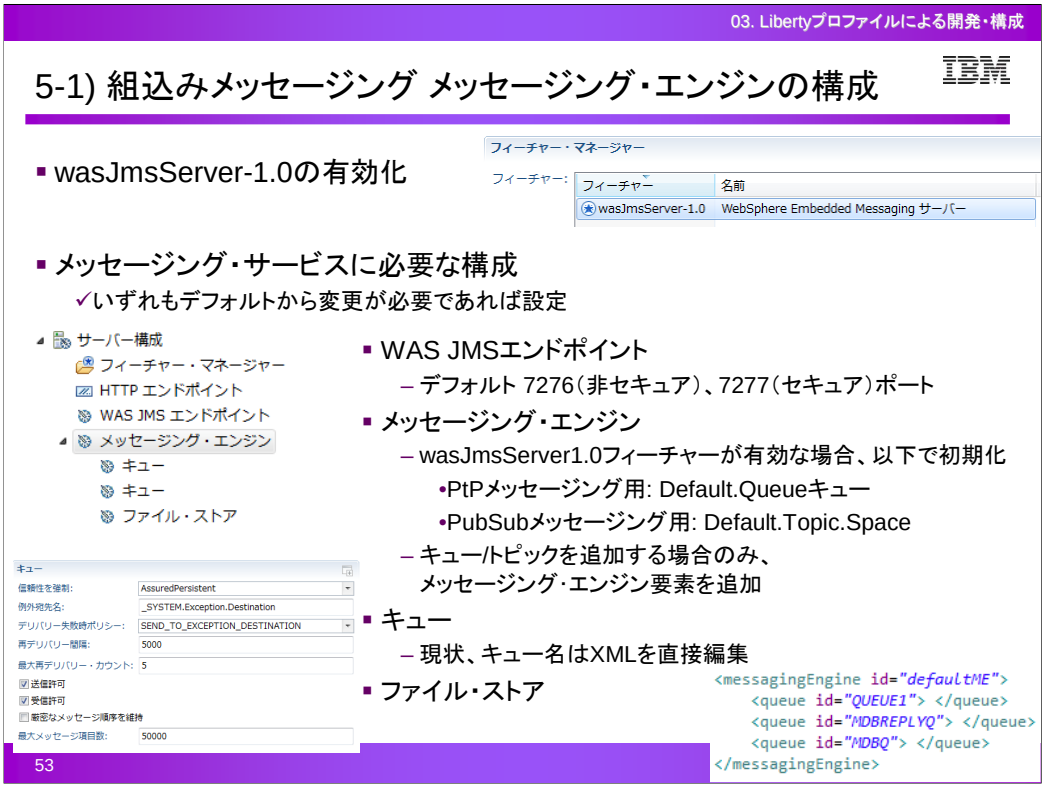
サービス統合バス（SIBus）と完全互換

- wmqJmsClient-1.1
  - 外部のWebSphere MQのQmgr上のメッセージング・キューにJMS 1.1でアクセス
- jmsMdb-3.1
  - メッセージ駆動型Bean（MDB）を構成

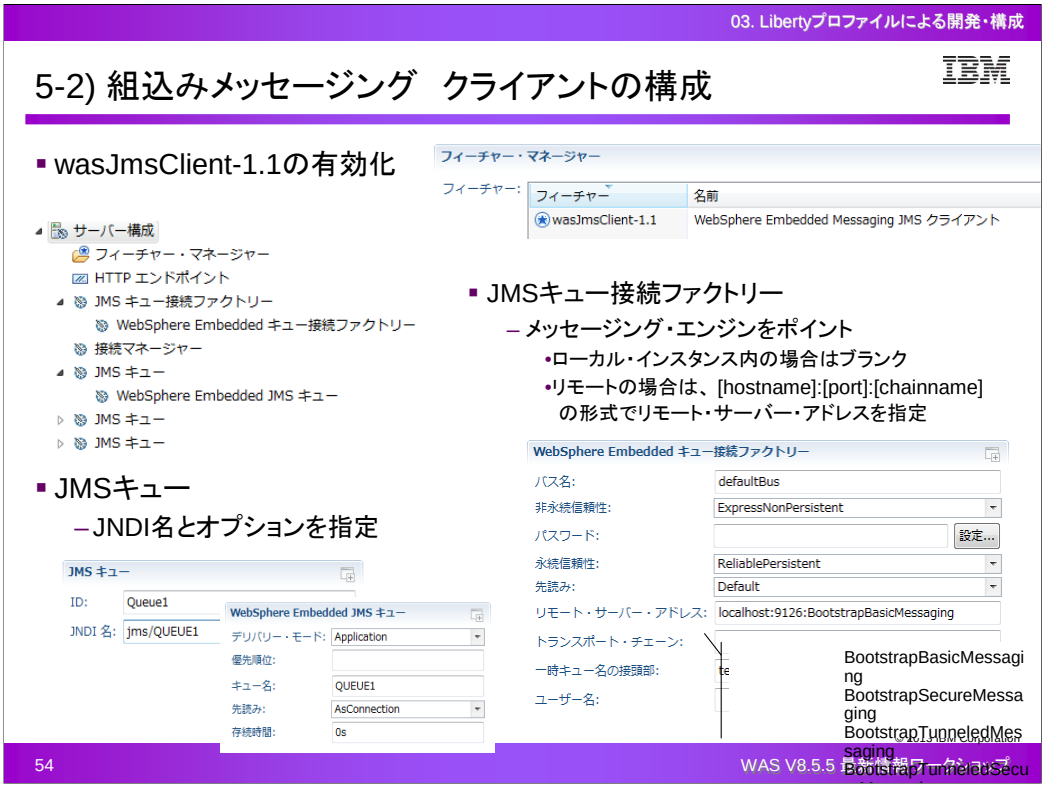
© 2013 IBM Corporation

次にExpress, Base, NetworkDeploymentで提供される機能についてみていきます。

まずメッセージングです。メッセージングに関連するフィーチャーは4つあります。Liberty内部にメッセージング・エンジンを構成するwasJMSServer1.0、Liberty内部に構成したメッセージング・エンジンまたは完全プロファイルのSIBusにアクセスする際のJMSクライアントとなるwasJMSClient-1.1、外部のWebSphere MQのQMGRにアクセスする際のJMSクライアントとなるwmqJmsClient1.1、さらにメッセージ駆動型Beanを提供するjmsMdb-3.1です。

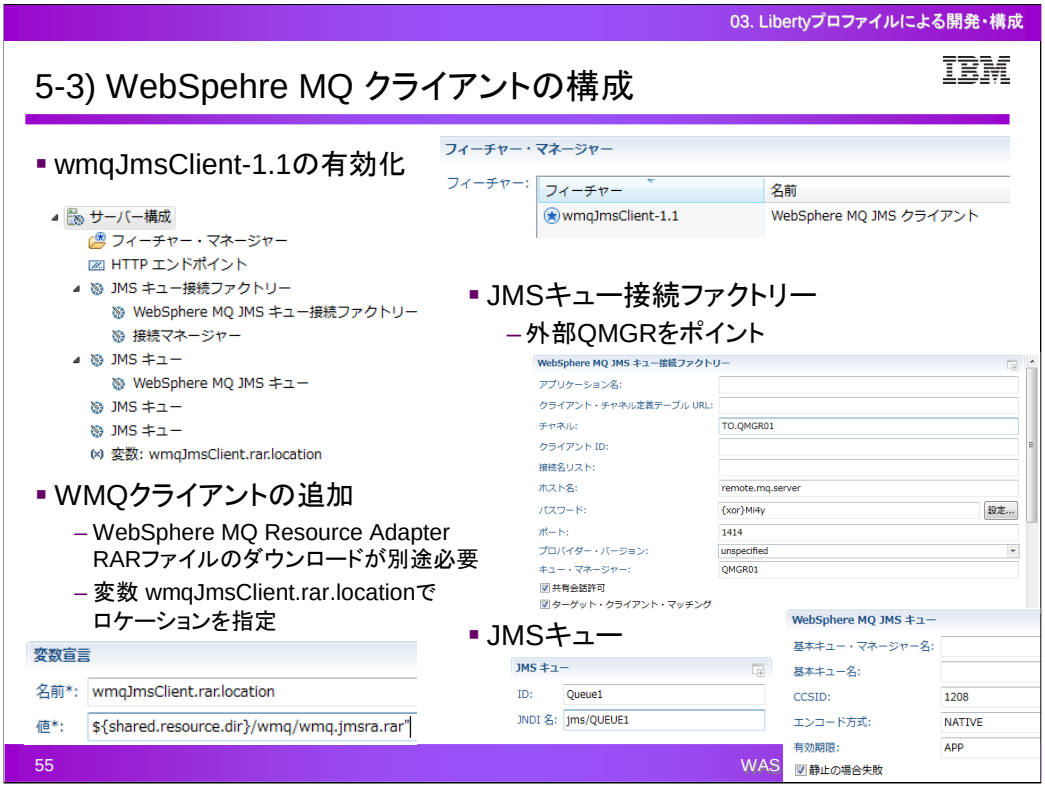


メッセージング・エンジンを構成するには、まずLiberty内部に組込みメッセージング・エンジンを構成するwasJmsServer-1.0をフィーチャーで有効化します。デフォルトでは、DefaultQueue名前で宛先キュー、およびDefautl Topic Spaceという名前でトピック・スペースが作成されますので、これを変更したい場合には、明示的にメッセージング・エンジンに宛先を追加していきます。メッセージの永続化のためにファイル・ストアのロケーションを変更する場合にも、明示的にファイル・ストアを関連付けます。



次にクライアント側のwasJmsClient-1.1です。

フィーチャーを有効化した後、接続先のメッセージング・エンジンを指定するためのJMSキュー接続ファクトリーを指定します。師弟方法はリモート・サーバー・アドレスに対してホスト名およびポートを指定します。さらにキューの宛先のJNDI名などはJMSキューとして定義します。

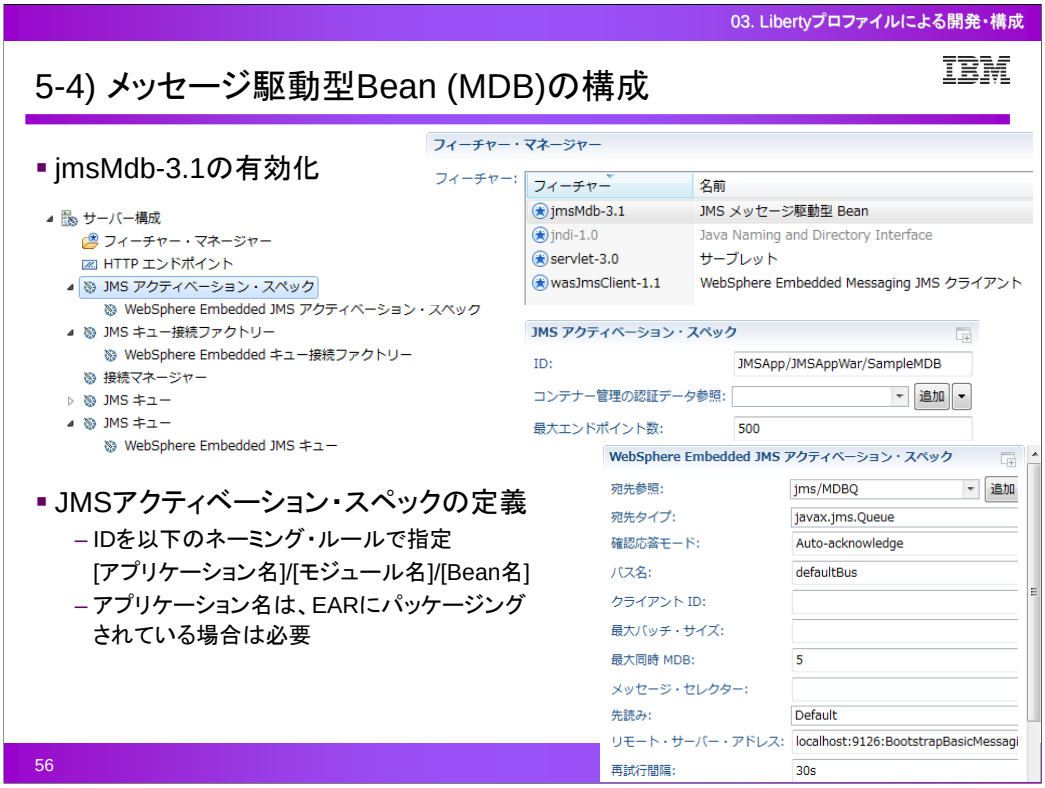


次にWebSphere MQのJMSクライアントです。

接続のために必要なJMSキュー接続ファクトリーやJMSキューは、WASの組込みメッセージング・エンジンと同様です。

Liberty自体には従来の完全プロファイルと異なり、WMQのクライアント・ライブラリであるWebSphere MQ Resource Adapterが同梱されていませんので、ご自身でダウンロードしていただき、変数としてWMQクライアントを定義する必要があります。定義方法については、以下のTechnoteのリンクをご参照ください。

http://www-01.ibm.com/support/docview.wss?uid=swg21633761



さらにメッセージ駆動型Bean（MDB）を使用する場合には、フィーチャーとしてJms-Mdb3.1を有効化した後、JMSアクティベーション・スペックを定義して、MDBで使用するメッセージング・エンジンやJMS宛先を指定します。

6) Webサービス (JAX-WS, JAX-RS)

- JAX-WS 2.2をサポート
  - Java API for XML Web Services
  - jaxws-2.2フィーチャーを有効化
    - jaxb-2.2も暗黙的に有効化される
  - ✓JAX-RPC はサポートしない
    - JavaEE6仕様として deprecated feature扱い

フィーチャー・マネージャー

フィーチャー:

フィーチャー	名前
jaxws-2.2	Java API for XML Web サービス
jaxb-2.2	Java Architecture for XML Binding

- JAX-RS 1.1をサポート
  - Java API for XML RESTful Web Services
    - RESTスタイルのWebサービスを提供
  - Jaxrs-1.1フィーチャーを有効化
    - あわせてjson-1.0も有効化される

フィーチャー・マネージャー

フィーチャー:

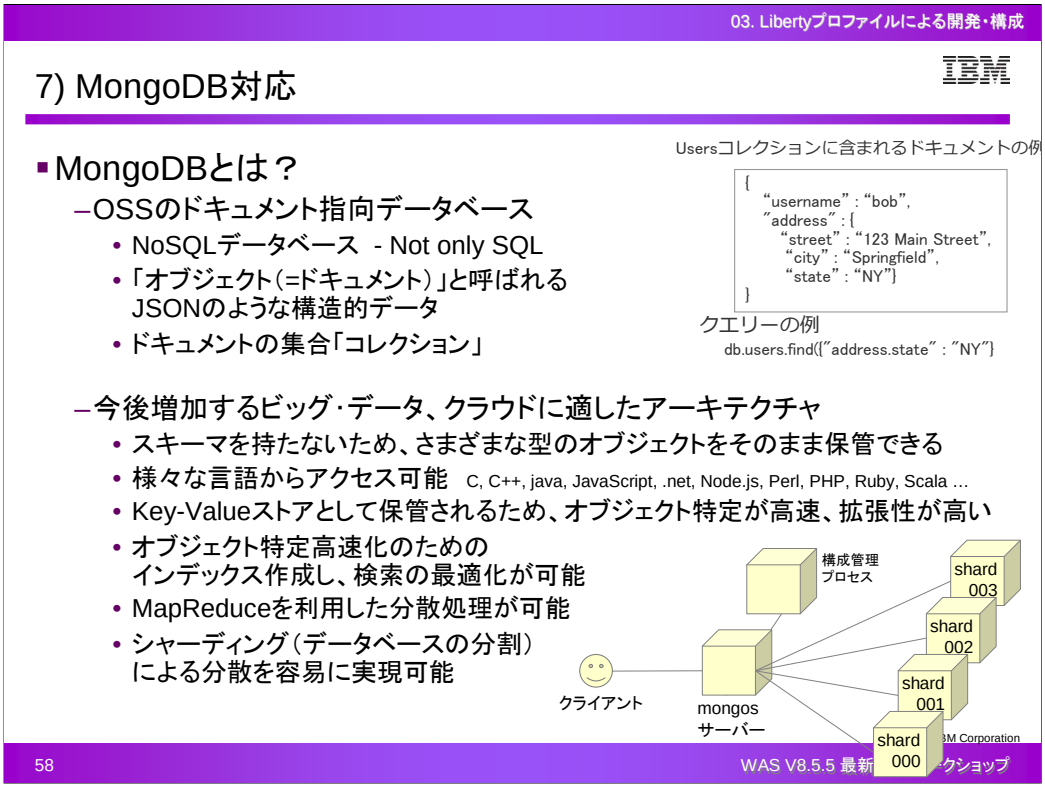
フィーチャー	名前
jaxrs-1.1	Java API for XML RESTful Web サービス
json-1.0	JavaScript Object Notation for Java

© 2013 IBM Corporation

次にWebサービスです。Webサービス自体は基本的にフィーチャーを有効化することで利用可能です。

LibertyでサポートされるWebサービスはJAX-WSです。JAX-RPCはLibertyではサポートされません。

またRESTスタイルのWebサービスを提供するJAX-RS1.1もサポートしています。



最後に WAS8.5.5で提供が開始されたMongoDBについて見ていきます。

前のセッションでも見たようにMongoDBは、Not only SQLと呼ばれるデータベースの代表的なものです。JSON形式のオブジェクト(ドキュメント)を保管するため、ドキュメント指向型データベースと呼ばれます。

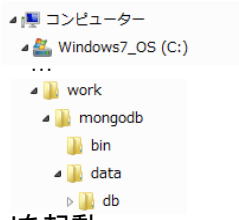
MongoDBが注目されているのは、今後増加すると考えれられるビッグ・データやクラウドのアプリケーションに適したアーキテクチャを持っているためです。スキーマを持たないため様々なデータをそのまま保管可能であり、KeyValueストアとして保管されるため、シャーディング(データベースの分割)により、保持可能なデータがスケールしやすいといったメリットがあります。MongoDBのMongoとはHumongo(ばかでかい)という言葉から名づけられたものです。また今後のシステム動向として多システム言語化していくと考えられていますが、そういったpolyglot環境にもMongoDBは対応しています。

このMongoDBに対するクライアント機能をLibertyは提供しています。

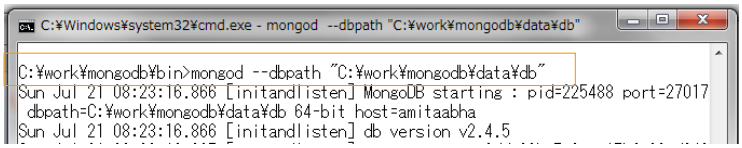
[参考] MongoDB クイック･スタート

1. http://www.mongodb.org/ を訪れ、環境に応じたファイルをダウンロード

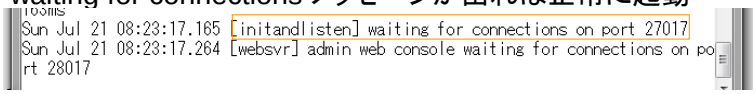
注） ライセンス、使用同意事項はご確認ください



2.zipファイルを任意の場所に展開
3.任意の場所に data/db ディレクトリを作成
4.--dbpathオプションでdbディレクトリのパスを指定して mongodを起動



5. waiting for connections メッセージが出れば正常に起動



© 2013 IBM Corporation

MongoDBの使い方ですが、非常にシンプルです。

MongoDBのサイトを訪れ、環境に応じたファイルをダウンロードします。落としてきたzipファイルを任意の場所に展開します。データの保持先であるdata/dbディレクトリを作成し、そのディレクトリを—dbpathオプションで指定して、mongodコマンドで起動します。

これだけで、mongodbを利用可能になります。（※詳しい使い方は、MongoDBのサイトをご確認ください）

このようなMongoDBのためにアクセス可能なクライアント機能をLibertyでは提供しています。利用方法としては、mongodb-2.0を有効化し、共有ライブラリとして、ダウンロードしてきたMongoDBのJavaドライバーを定義します。あとはMongoDB インスタンス接続のための認証情報やJNDI名の定義などを行うことで利用可能になります。

03. Libertyプロファイルによる開発・構成

7-2) アプリケーション内でのMongoDBの利用

```
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MongoException;

@WebServlet("/")
public class MongoServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Resource(name = "mongo/sampledb")
    protected DB db;

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        PrintWriter pw = resp.getWriter();
        if (db == null) {
            pw.println("Uh oh! Something went wrong. DB is null, please check the server logs.");
        } else {
            pw.println("HelloMongo!");
        }

        try {
            DBCollection col = db.getCollection("user");
            long count = col.getCount();
            pw.println("<br>[count] collection count = " + count + " from [sampledb]<br>");
            pw.println("[inserting] 10<br>");
            for (int i = 0; i < 10; i++)
                col.insert(new BasicDBObject("user", count + i));

            pw.println("[dump entire db] : <br>");
            DBCursor cursor = col.find();
            while (cursor.hasNext()) {
                DBObject obj = cursor.next();
                pw.println(obj+"<br>");
            }
```

MongoDB関連クラスをインポート

@Resouce で指定したJNDI名でDBインスタンスを取得

db.getCollection()でコレクションを取得

データをコレクションにインサート

取得したカーソルで結果をイテレートできる

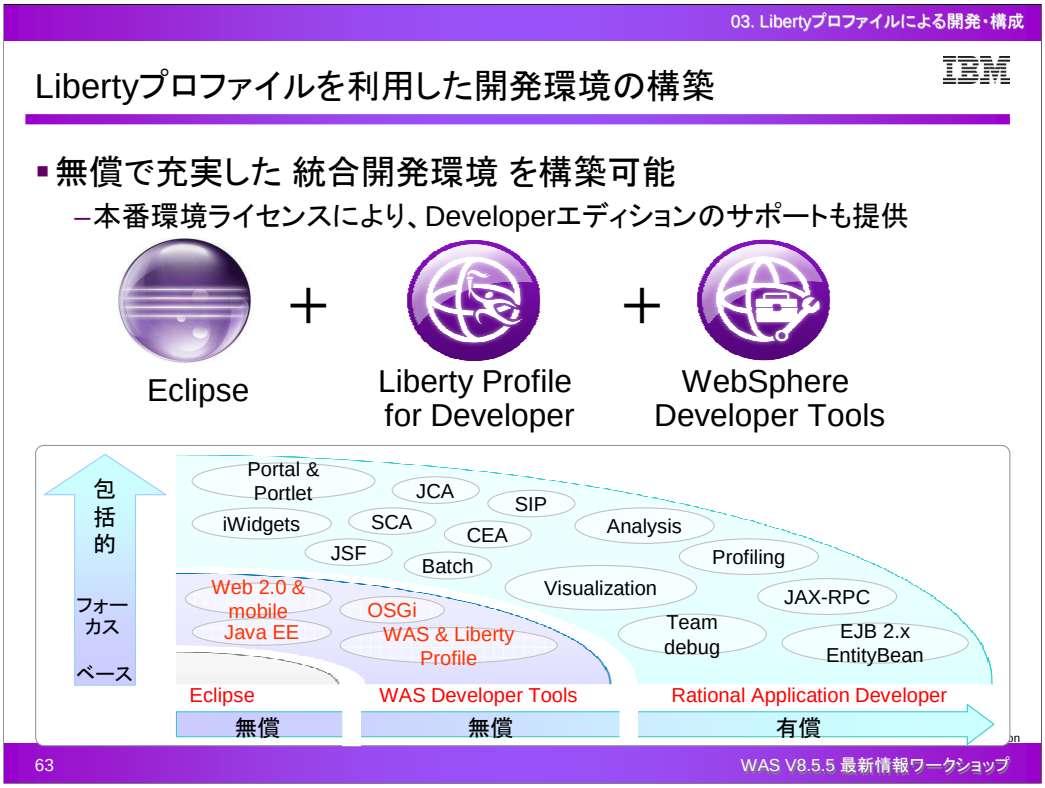
© 2013 IBM Corporation

MongoDBを利用したコードのサンプルです。

Mongodbのライブラリから関連するクラスをインポートして、@ResourceでMongoDBのインスタンスを取得しています。

取得したDBのインスタンスから、db.getCollectionでコレクションを取得して、データの挿入や参照を行っています。

最後にLibertyプロファイルを利用した開発について見ていきます。



これまで見てきたとおり、Eclipseに、Liberty ProfileのDeveloperエディションおよびWebSphere Developer Toolsを追加することで、無償で充実した統合開発環境を構築することが可能です。またWAS8.5.5からは、本番環境ライセンスによりDeveloperエディションのサポートも提供されるようになりました。

無償でJava EEやWeb2.0、OSGiのアプリケーションは開発できるようになりますが、Liberty ProfileおよびWDTではサポートされない、SIPやJava Batch、JSFといった仕様の開発や、JAX-RPCやEJB2.X、Entity Beanの開発はRational Application Developerを利用して従来通り開発します。

03. Libertyプロファイルによる開発・構成

開発者のエクスペリエンスの向上

- シンプルな構成ファイルで構築、容易な導入
  - サーバー環境の管理にかかる手間とコストが劇的に減少
  - 開発チームでの構成情報の共有や再利用も簡単
- 軽量なコンテナー
  - わずか数秒で起動、少ないメモリー・フットプリント
    - Eclipseなど統合開発環境と同時に起動していても、ストレス・フリー
  - ダイナミックに反映されるランタイム環境
    - 使用フィーチャーの追加・削除、設定変更でも、サーバー再起動必要なし
    - デフォルトでは、アプリケーション更新も動的に反映、稼動確認も容易

© 2013 IBM Corporation

Libertyを利用することで、従来の完全プロファイルのWebSphere ASと比べ、何が変わるのでしょうか？

まず、シンプルな導入方法、シンプルな構成方法で、開発環境の構築が非常に簡便になります。これによりサーバー環境の管理にかかる手間とコストを劇的に減少させることができます。また、includeファイルを利用することで、開発チーム内での構成情報の共有や最利用も容易になるといえます。

また、開発者自身へのメリットも多くあります。わずか数秒で起動する軽量のコンテナーという点も魅力ですし、少ないフットプリントで起動するため、必ずしも十分なリソースがないことの多い開発用のPCでもストレス・フリーでテストが可能です。また、ダイナミックにランタイム環境が更新されるため、パラメータや設定を変えるたびに再起動してといったわずらわしい作業から解放されます。こういった小さいように見える点も、開発時にはストレスとなり生産効率が落ちてしまう原因となっていましたが、この点が改善された点がおおきなメリットといえます。

最後に今一度、現在のシステム開発の潮流について振り返ってみましょう。

机上で綿密な計画を立てて、最初から完成品の大規模リリースを目指すこれまでの従来型のアプローチは、変化の激しい現在においてはリスクが高いという考え方が広まり、リーン・スタートアップという考え方が広がってきました。最初から大きな完成品を目指すのではなく、よりシステムが使われるようにするにはどうするか、よりユーザーが使いやすいシステムを実装するにはなにが必要なのかという仮説を立て、その仮説をより早く実証できるコードを開発し、計測を通じてシステムの改善につなげるアプローチが求められています。

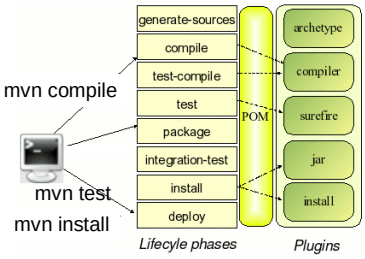
またネット企業で培われたWebアプリケーションの運用ノウハウから産まれたDevOpsという考え方を、企業システムにも取り込もうという考え方が広がってきています。ビジネスの要求から、開発、運用まで一体化してシステム運用を行っていきます。DevOpsでは、テストやビルドのサイクルを継続的に回す（継続インテグレーション）を前提として、継続的なアプリケーションの改修と改善、機能追加（継続的デリバリー）を行っていきます。

このような継続インテグレーションおよび継続デリバリーといった考え方に対応するためには、それを支える基盤といったものも変わらなければいけません。これまでのような開発環境の準備に週単位で時間がかかるのでは、システム開発のサイクルをスピード感をもって回していくことはできません。基盤に関しても、コードのようにバージョン管理され、再現性のある環境が求められており、それらはInfrastructure as Code呼ばれます。PureApplicationなどのシステムも同様の思想の元に出てきているシステムと考えることもできるでしょう。

03. Libertyプロファイルによる開発・構成

Apache Mavenとの統合 - Infrastructure as Codeの実現 -

- Apache Maven
  - Java用 プロジェクト・ライフサイクル管理ツール
    - プロジェクト(アプリケーション)情報の一元管理
    - 依存ライブラリの管理
    - ビルドプロセスの単純化、自動化
      - プロジェクトのコンパイル、ユニットテスト



  - Maven用プラグインを提供
    - WASDevに構成された プラグイン・リモート・レポジトリをpom.xmlに登録
      - http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/maven/repository/
    - Mavenから呼び出せる処理
      - サーバー作成
        mvn liberty:create-server -DserverHome=SVR_HOME -DserverName=SVR_NAME
      - アプリケーションのデプロイ、アンデプロイ
        mvn liberty:deploy -DserverName=SVR_NAME -DserverHome=SVR_HOME -DappArchive=APPLICATION_FILE
      - サーバーの起動、停止
        mvn liberty:start-server -DserverHome=SVR_HOME -DserverName=SVR_NAME
      - サーバーのパッケージング、導入
        mvn liberty:package-server -DserverName=SVR_NAME -DserverHome=SVR_HOME -DpackageFile=ARCHIVE_FILE

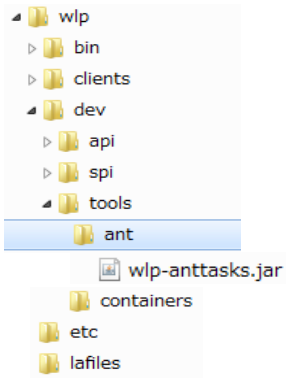
© 2013 IBM Corporation

こういった継続インテグレーションの中で継続的にテストをする際に利用できるのがApacheのMavenです。

Mavenは現在多くのプロジェクトで活用されているJavaプロジェクトのライフサイクル管理ツールです。MavenではJavaプロジェクト情報の一元管理のほか、依存ライブラリの管理、ビルドプロセスの自動化、ユニットテストの自動化など、一連のライフサイクルを管理することができます。このMavenの用のプラグインを利用して、MavenでのJavaアプリケーションの管理の中にLibertyを取り込むことが可能です。Mavenから呼び出せる処理は上図の通りです。

Apache Antとの統合 - Infrastructure as Codeの実現 -

- Apache Ant
  - Java用 ビルド・ツール
    - ビルドプロセスの単純化、自動化
  - wlp-anttasks.jar
    - wlp/dev/tools に存在
    - Ant環境から呼び出せるよう配置
  - 利用可能なAntタスク
    - サーバー構成、管理
      - Antタスク server: create, start, stop, status, package
    - アプリケーションのデプロイメント、アンデプロイ
      - Antタスク deploy, undeploy



© 2013 IBM Corporation

同様にJavaのビルド管理ツールであるAntとも統合するためのAntタスクが用意されています。

これらのタスクを含むwlp-anttasks.jar がwlp/dev/tools配下に配置されていますので、それをAnt環境から呼び出せるように配置することで、Antでの管理にLibertyのランタイム環境を組込むことが可能になります。

IBM

まとめ

© 2013 IBM Corporation

ここまで、Libertyの導入から基本的構成、開発にいたるまで見てきました。
Libertyの特色のうち、黄色字になっている箇所を中心にご説明させていただきました。

03. Libertyプロファイルによる開発・構成

まとめ

- 容易に導入・構築可能なアプリケーション・サーバー環境
- 軽量フットプリント、動的なランタイム構成
- WAS8.5.5で、対応プログラミング・モデルが拡充
- 今後のシステム開発により適したサーバー環境
  - 継続インテグレーション、継続デリバリー
  - Infrastructure as Codeの実現

© 2013 IBM Corporation

参考資料

- WebSphere Application Server InfoCenter
  - http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp
  - 「範囲」で Libertyに絞り込んで頂くと、探しやすくなります
- WASDev.net
  - https://www.ibmdw.net/wasdev/
  - Liberty Profile for Developers やWDTのダウンロード
- Libertyレポジトリ
  - https://www.ibmdw.net/wasdev/repo
  - 構成ファイルや、OSSソフトウェアとの統合サンプル、管理スクリプトのサンプルなどを提供

© 2013 IBM Corporation

03. Libertyプロファイルによる開発・構成

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、DB2、WebSphere、およびz/OSは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。
他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。
現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。

インテル, Intel, Intelロゴ, Intel Inside, Intel Insideロゴ, Intel Centrino, Intel Centrinoロゴ, Celeron, Intel Xeon, Intel SpeedStep, Itanium, およびPentium は Intel Corporationまたは子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。
Microsoft, Windows, Windows NT および Windowsロゴは Microsoft Corporationの米国およびその他の国における商標です。
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。

© 2013 IBM Corporation

03. Libertyプロファイルによる開発・構成

IBM

END

© 2013 IBM Corporation