WebSphere Application Server V8.5.5

最新情報ワークショップ

～次世代WASランタイム Liberty で変わるWebシステム～

Libertyプロファイルの構築・運用管理

Disclaimer

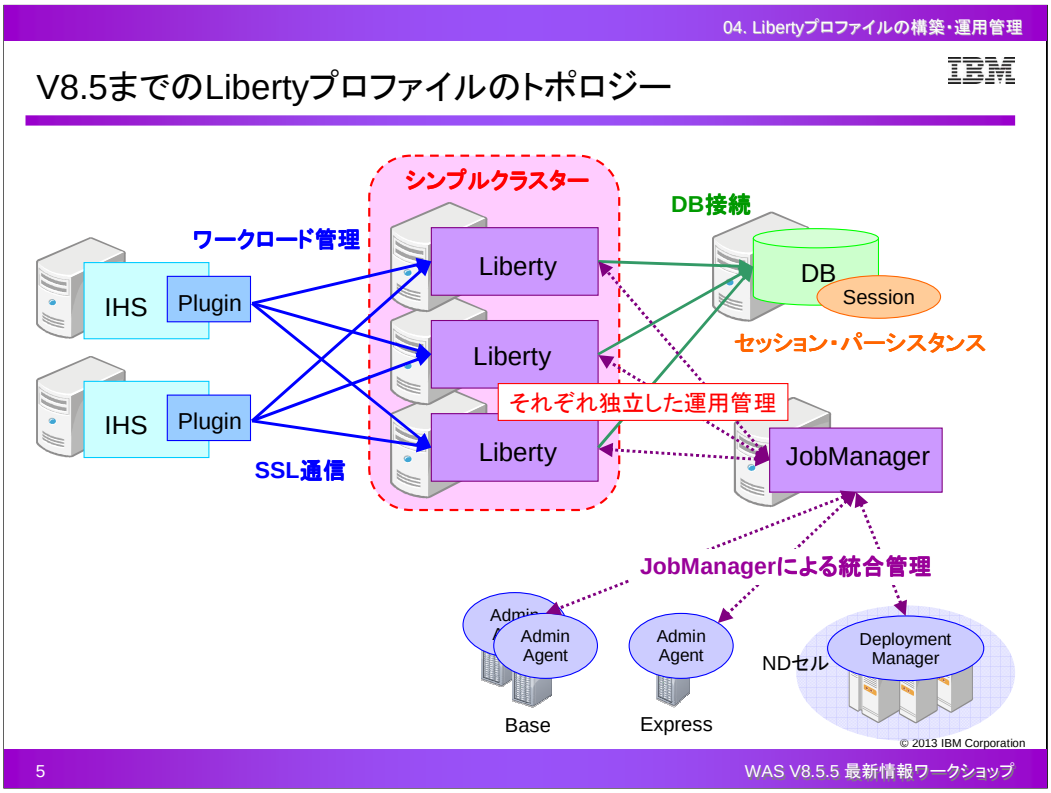
- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2013年7月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- IBM、IBMロゴおよびibm.comは、世界の多くの国で登録されたInternational Business 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。

目次

IBM

トポロジー

V8.5までのトポロジー
V8.5.5でのトポロジー
Liberty Collective
Libertyクラスター
トポロジー選択指針



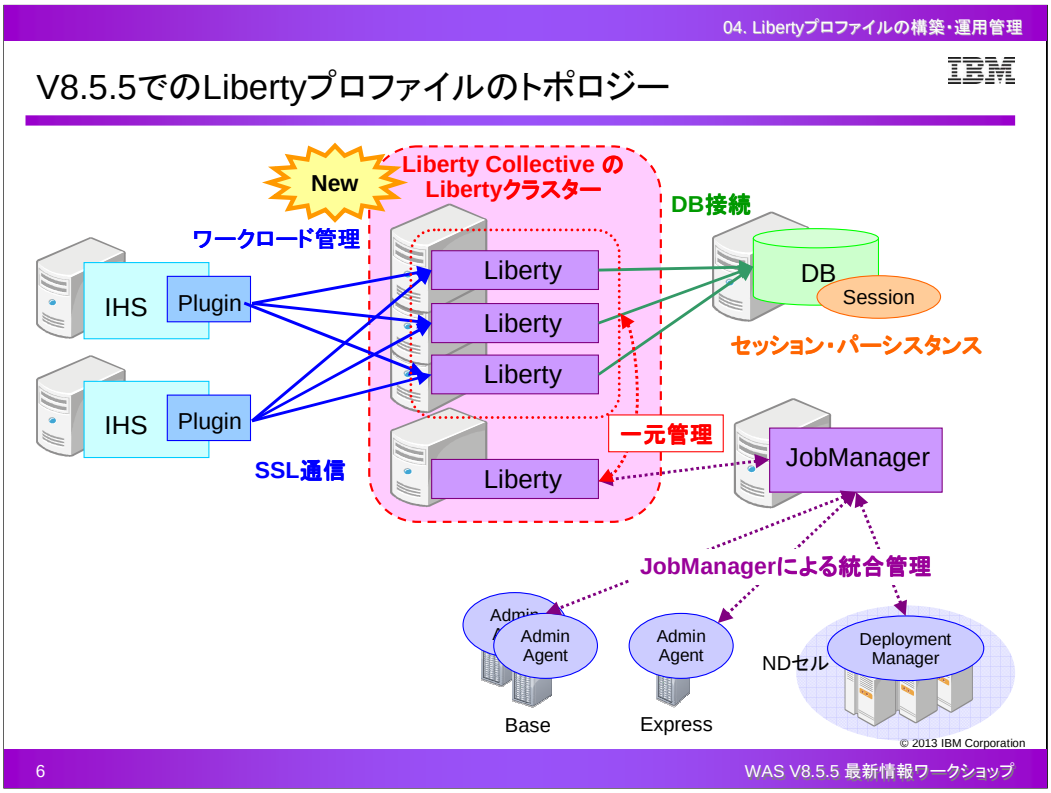
V8.5までのLibertyプロファイルの基本トポロジーは、フルプロファイルの基本トポロジーとほぼ同じです。

Libertyプロファイルに関しても、同じアプリケーションを複数のノードや複数のサーバー上で稼働させるために並列に配置することができます。ただし、WASのNDエディションで提供されている高機能クラスターとは異なり、Libertyプロファイルでは、Expressエディションや Baseエディションでもサポートされているシンプルクラスターを構成することになります。したがって、それぞれのLibertyプロファイルに対して独立した管理や操作を行う必要があります。

Libertyプロファイルの前段にIHSおよびプラグインを配置することで、IHSから複数のLibertyプロファイルへのリクエストの割り振りを行うことができます。IHSからLibertyプロファイルへの割り振りでは、セッション・アフィニティーやサーバー・フェールオーバーなどのワークロード管理を行うこともでき、SSLを使用した通信を行うこともできます。

Libertyプロファイルから後段のDBサーバーに対して、フルプロファイルと同様にデータソースを使用した接続を行うこともできます。また、セッション情報の永続化のためにセッション・パーシスタンスの構成もサポートしています。

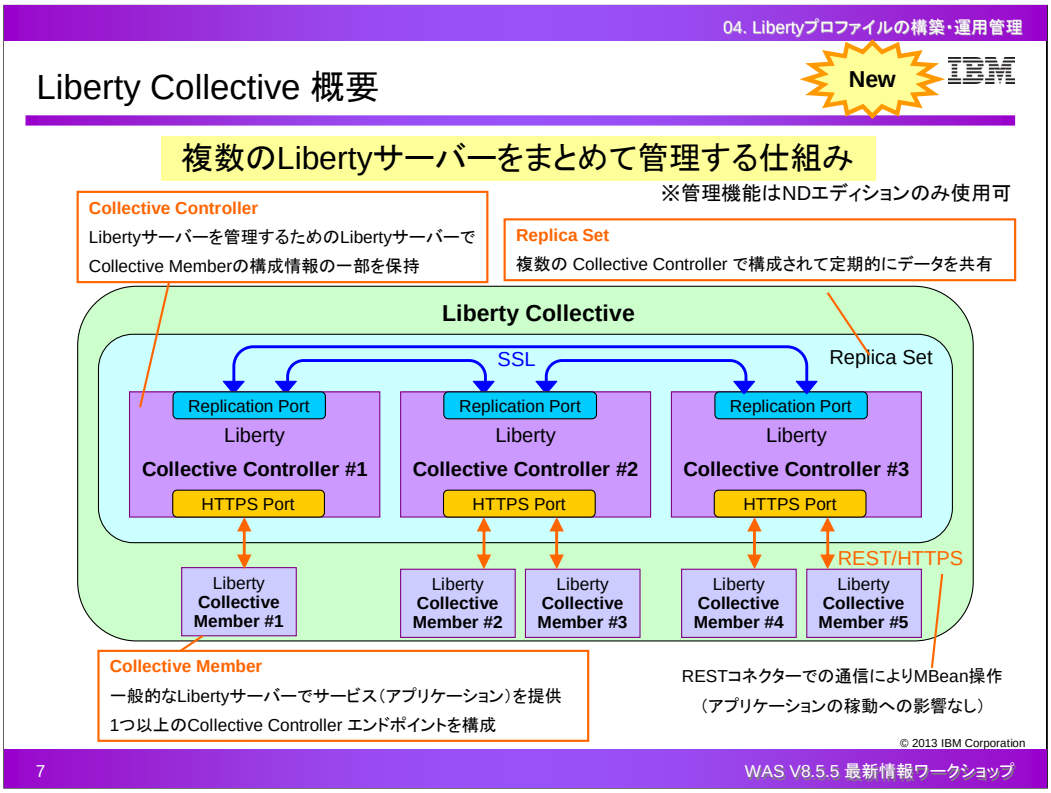
運用管理面では、複数のLibertyプロファイルをJobManagerから統合的に管理し、ジョブを実行することができます。



V8.5.5でのLibertyプロファイルの基本トポロジーでは、V8.5.5の新機能であるLiberty Collective（Liberty集合）を構成することができるようになりました。

Liberty Collective とは、複数のLibertyサーバーを管理用のLibertyサーバーを使用して一元管理するという仕組みです。異なるアプリケーションが稼動するそれぞれ独立したスタンドアロンのLibertyサーバーをまとめて管理することもできますし、シンプルクラスターのように同じアプリケーションを複数のノードや複数のサーバー上で稼働させるためのLibertyクラスターを構成し、それらをまとめて管理することもできます。

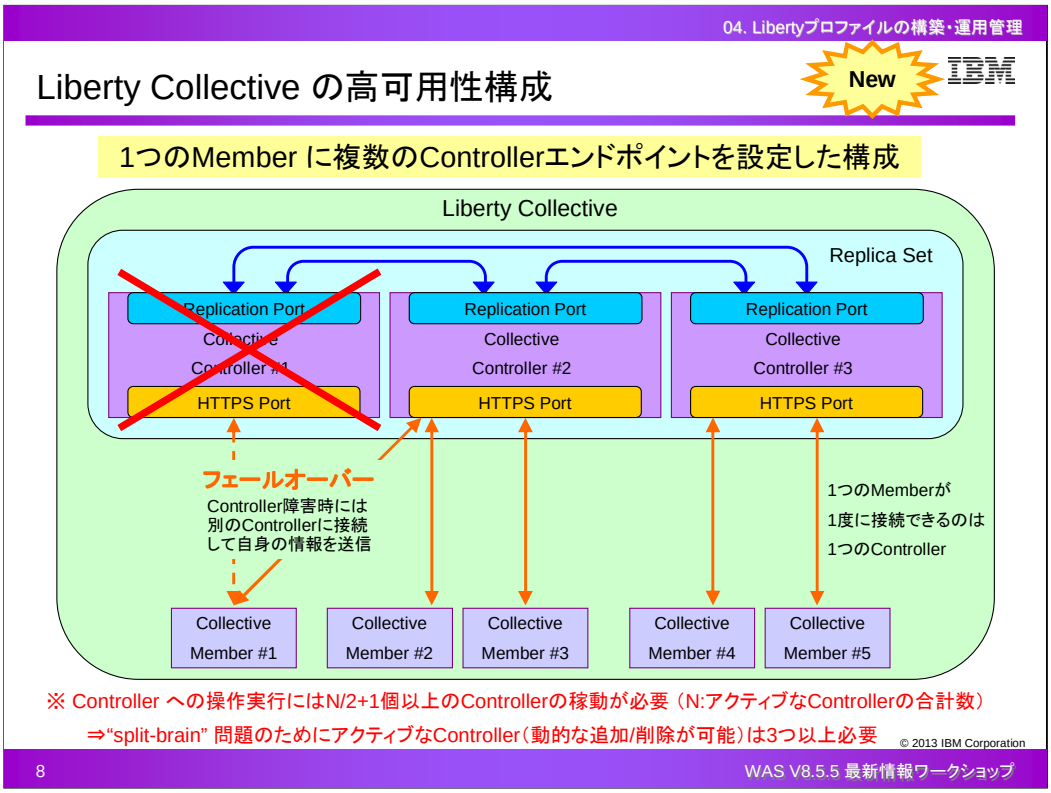
IHSからLibertyプロファイルへ割り振りを行う際のワークロード管理やSSL通信、Libertyプロファイルからの DB接続やセッション・パーシスタンス構成、JobManagerによる統合管理といった、クラスター構成以外の部分に関しては、V8.5.5でも同様の機能は提供されています。V8.5.5では、新たにメッセージング・サービスへの接続機能も提供されています。



Liberty Collective とは、複数のLibertyサーバーをまとめて管理する仕組みのことです。管理される側の機能は、Liberty CoreやBaseといったエディションでも使用することができますが、管理する側の機能はWASのNDエディションでのみ使用することができます。

こちらの図は、Liberty Collective のアーキテクチャーで、Liberty Collective は1つの管理/操作ドメインの単位を指します。Liberty Collective は、Collective Controller とCollective Member の2種類のLibertyサーバーで構成されます。Collective Member とは、実際にアプリケーションを稼動させるための一般的なLibertyサーバーで、エンドポイントとして構成されたCollective Controllerに接続して、自身の情報を送信します。Collective Controller とは、Collective Member を管理するためのLibertyサーバーで、Collective Member から受け取った構成情報やランタイム状況などの一部の情報を保持しており、Liberty Collective を操作するためのMBeanを提供します。Collective Member とCollective Controller の通信は、常にJMX Restコネクターを使用したMBean操作の形式で行われ、この通信の状況は、Collective Member のアプリケーションの稼動（サービス）には影響を与えません。Replica Set とは同じLiberty Collective 内で情報を共有し合う1つ以上のCollective Controller で構成された単位で、定期的に専用のReplication Port を使用して通信およびデータの共有を行います。

Liberty Collective は1つ以上のCollective Controller で構成されますが、複数のCollective Controller を配置することにより、可用性を持たせることができます。



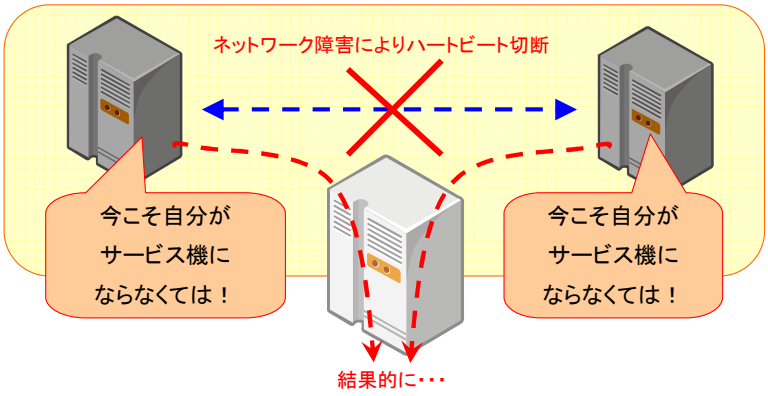
Liberty Collective では、1つのCollective Member が1度に接続できるのは1つのCollective Controller のみですが、Collective Member のエンドポイントにフェールオーバー用に複数の Collective Controller エンドポイントを設定しておくことにより、Liberty Collective を高可用性構成にすることができます。

具体的には、あるCollective Member が接続しているCollective Controller にプロセス障害が発生した場合、Collective Member は、設定されている別のCollective Controller へと接続し直すフェールオーバー機能が提供されています。

1つのLiberty Collective において、Collective Controller への操作を実行する際は、N/2+1個以上（N:アクティブなCollective Controller の合計数）のCollective Controller が稼動している必要があります。したがって、Collective Controller が2つでは”split-brain”問題に該当し、ネットワーク上の問題でCollective Controller 同士が通信できなくなった場合などに条件を満たせずCollective Controller への操作が実行できなくなってしまうため、最低でも3つのCollective Controller を構成しておく必要があります。Replica Set へのCollective Controller の追加（アクティブ化）/削除（非アクティブ化）は動的に行うことができますので、例えば3つ構成していたCollective Controller のうち1つにプロセス障害が発生した場合、動的に1つのCollective Controller を追加して合計3つに構成することにより、Collective Controller への操作を継続して行うことができます。

【参考】split-brain問題

スプリット・ブレイン・シンドローム　または　ネットワークパーティション問題



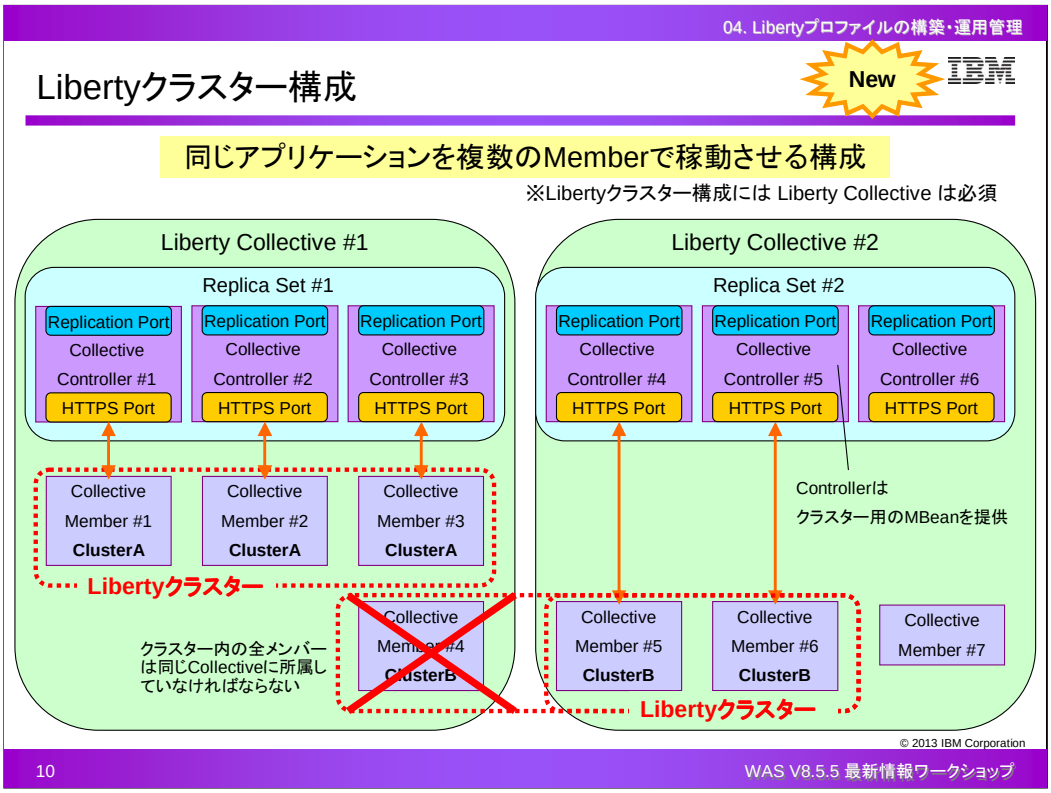
ファイルシステムのマウントで競合
データベースへの更新で競合
リソースの取得で競合
　などなど

サービス不能

”split-brain”問題とは、スプリット・ブレイン・シンドロームやネットワーク・パーティション問題とも言われています。

例えば2台のサーバーがアクティブ・スタンバイのクラスタ構成を組んで、両者間でお互いの監視のためにハートビートを送信し合っているようなケースにおいて、サーバーの障害ではなくネットワーク障害などによりハートビートが切断されると、両者が相手のサーバーがダウンしたと見なし、アクティブになろうとしてしまい、結果的にサービス不能に陥ってしまうことがあります。

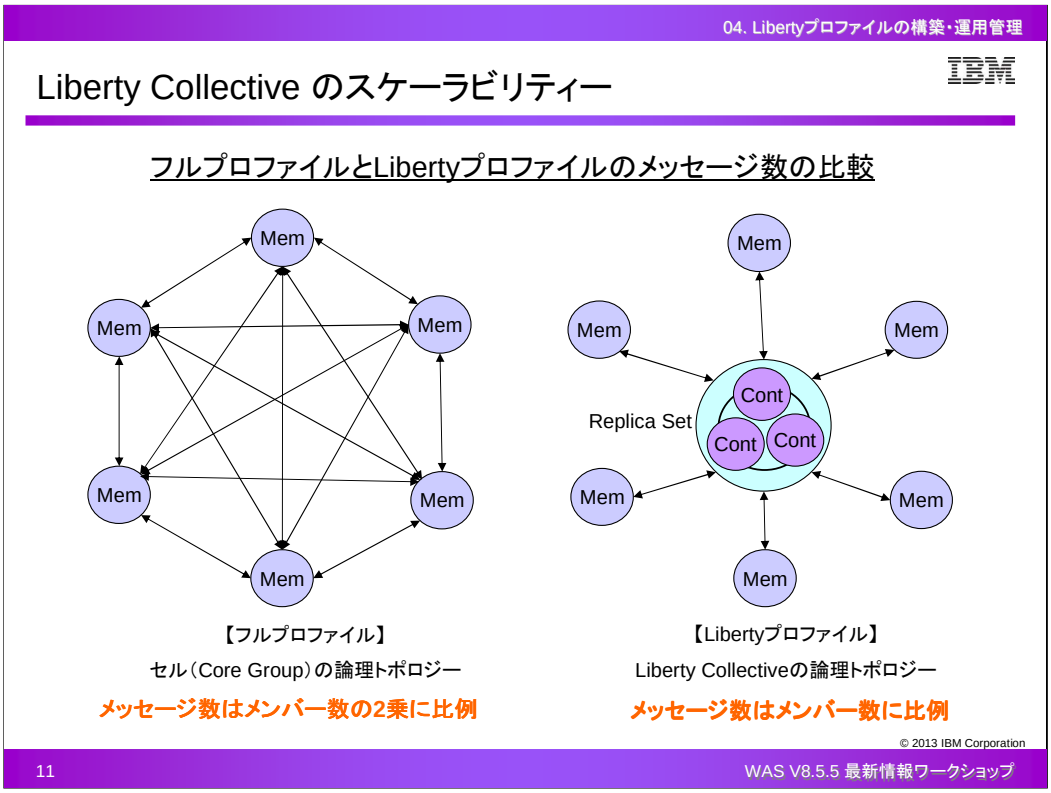
Liberty Collective においても、Collective Controller が2つで構成されている場合、定期的にReplication処理を行なっているところでネットワーク障害等が発生すると、それぞれのCollective Controller は、2つ構成されているうち自分1つだけがアクティブであると認識し、N/2+1という条件を満たせず、操作ができなくなってしまいます。



Liberty Collective を使用すると、同じアプリケーションを複数のアプリケーション・サーバー（Collective Member）で稼動させるためのLibertyクラスターを構成することもできます。

Libertyクラスターを構成すると、Collective Controller はクラスター用のMBeanを提供し、個々のLibertyサーバーに対する操作だけでなく、Libertyクラスターに属するLibertyサーバー全体に対する操作を行うことができるようになります。例えば、Libertyクラスター名の取得、Libertyクラスターの起動/停止や稼働状況の確認、といった操作を行うことができます。

Libertyクラスターを構成する場合、全てのクラスターメンバーは同じLiberty Collective に属していなければなりません。同じLiberty Collective の中には、Libertyクラスターに属しているLibertyサーバーとスタンドアロンのLibertyサーバーが共存するように構成することもできます。

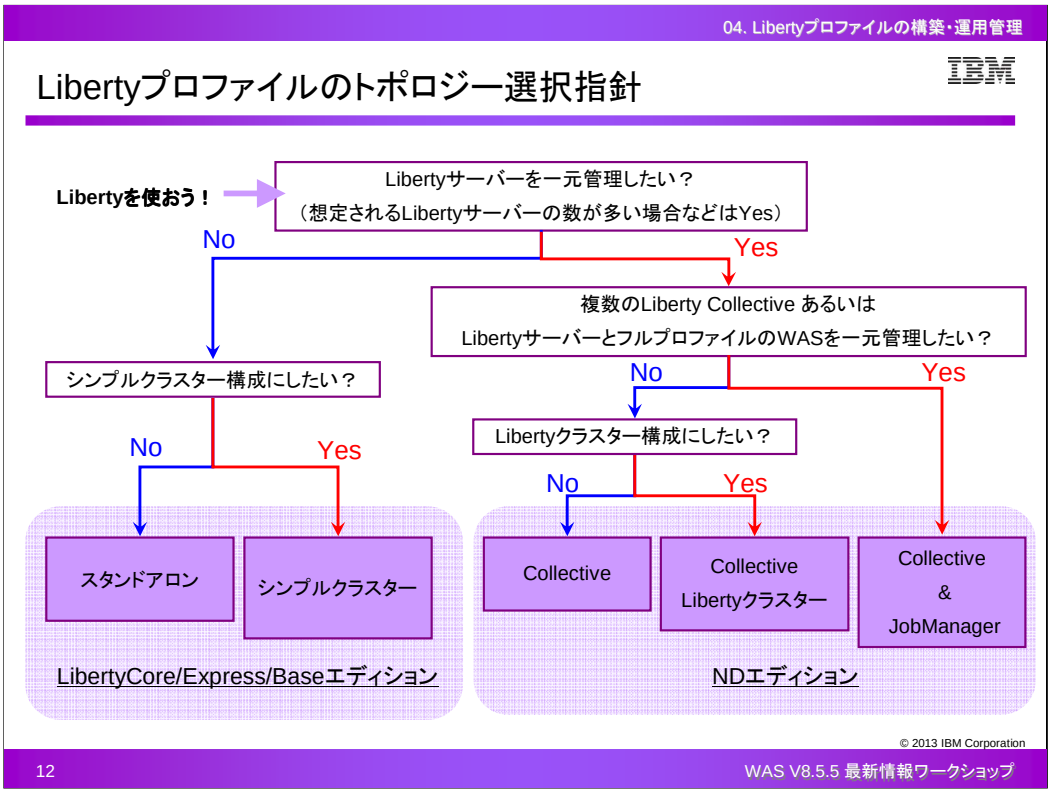


Liberty Collective のスケーラビリティーは、フルプロファイルと比較して優位であるといえます。

フルプロファイルのセル（Core Group）では、アプリケーションを稼動させるアプリケーション・サーバーやクラスターメンバーといったJVMの数が増えると、JVM同士の通信でやりとりされるメッセージの数はメンバー数の2乗に比例して増えていきます。

一方、LibertyプロファイルのLiberty Collective では、アプリケーションを稼動させるアプリケーション・サーバーやクラスターメンバーといったJVM（Collective Member）の数が増えると、JVM同士の通信はCollective Controller とCollective Member でやりとりされる分が増えるだけなので、メッセージの数はメンバー数に比例して増えていきます。

したがって、アプリケーションを稼動させるメンバーの数を増やしていった場合、フルプロファイルよりもLibertyプロファイルの方が内部通信によるオーバーヘッドは少なくなります。



V8.5.5のLibertyプロファイルを使用する際、どのような要件の場合にどのトポロジーを選択すべきか、その選択指針についてまとめます。

まず初めに、複数のLibertyサーバーを一元管理したいかどうかを検討します。想定されるLibertyサーバーの数が多く、1台1台に対してアクセスして管理や操作を行うといった負荷が大きい場合は一元管理を行います。Libertyサーバーの一元管理を行う場合は、Liberty Collective の機能を使用する必要があるため、WASのNDエディションを採用することになります。想定されるLibertyサーバーの数が少ない、あるいは特に一元管理する必要がないということであれば、WASのLibertyCore/Express/Baseエディションを採用することになります。

Libertyサーバーを一元管理しない場合、同じアプリケーションを複数のサーバーで稼動させる要件があればシンプルクラスター構成、そのような要件がなければ各Libertyサーバーが独立した状態のスタンドアロン構成になります。LibertyCore/Express/Baseのエディションの選択は、各エディションの機能的な違いやライセンス上の制約を確認して決める必要があります。

Libertyサーバーを一元管理する場合、複数のLiberty Collective（管理ドメイン）あるいはLibertyサーバーとフルプロファイルのWASなど全て含めて一元的に管理する必要があれば、Liberty Collective だけでなくJobManagerを使用した統合管理を行なう構成になります。

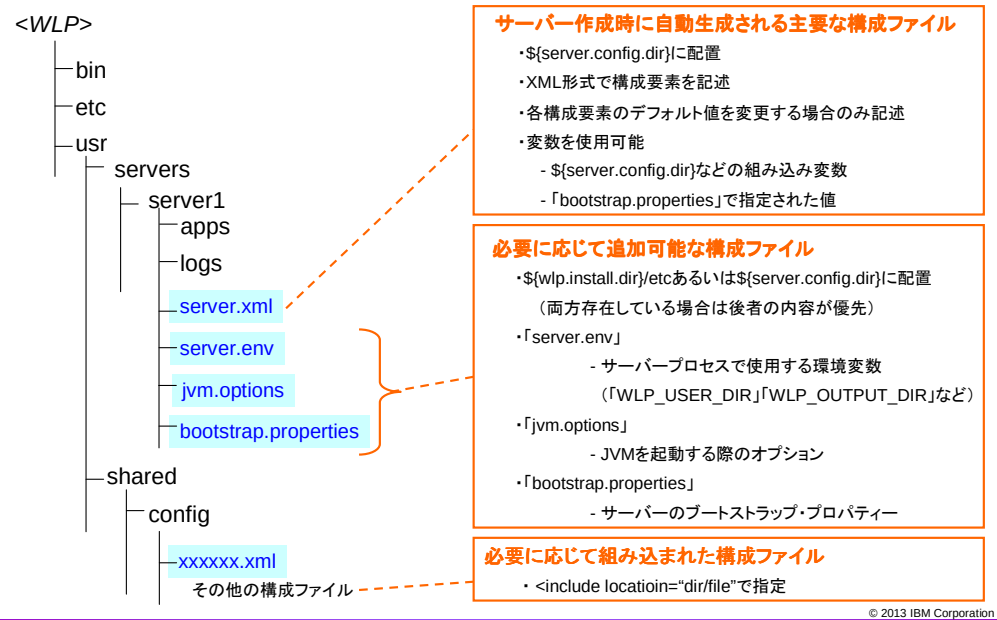
Liberty Collective の単位だけで一元管理できれば良いという場合は通常のLiberty Collective 構成、更に同じアプリケーションを複数のサーバーで稼動させる要件があればLiberty Collective を使用したLibertyクラスター構成になります。

IBM

構成情報の管理

Libertyプロファイルの構成情報
Liberty Collective の構成情報
Libertyプロファイルの構成管理とランタイム操作
Liberty Collective の構成管理とランタイム操作

Libertyプロファイルの構成情報

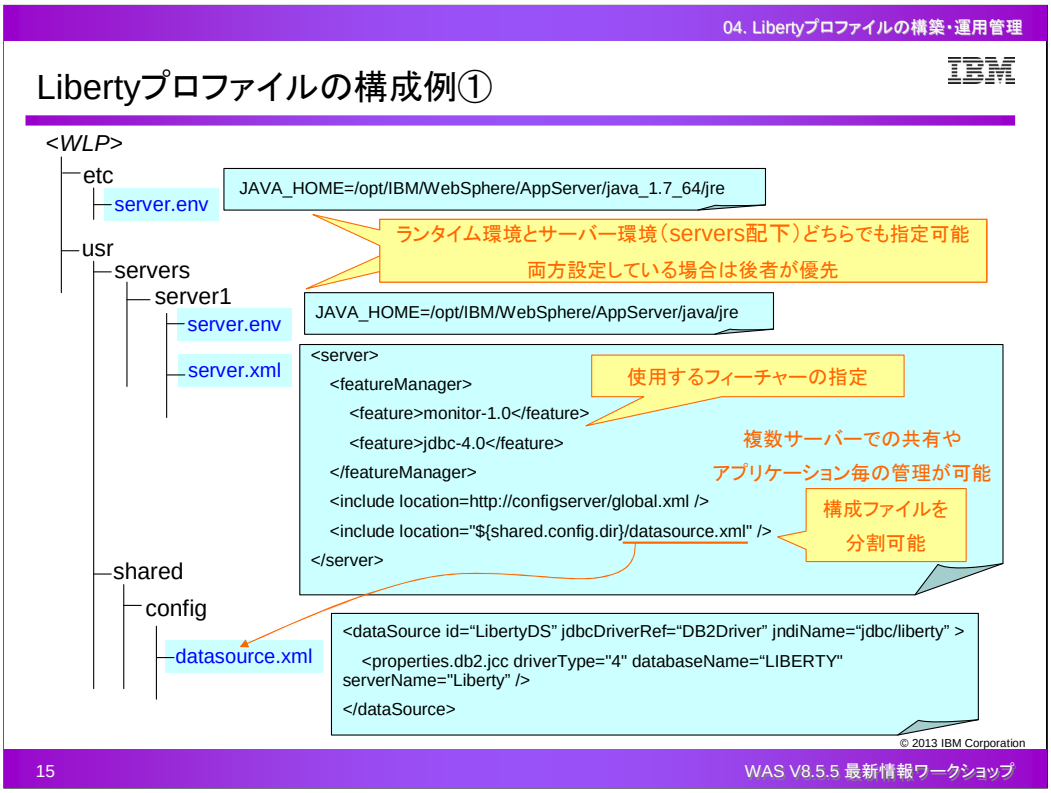


Libertyプロファイルの構成情報は全て構成ファイルに格納されています。

サーバー作成時に自動生成される主要なファイルとして、server.xmlがあります。これはサーバーの稼動に必須となる構成ファイルです。

必要に応じて追加することのできる構成ファイルとして、server.env、jvm.options、bootstrap.propertiesがあります。

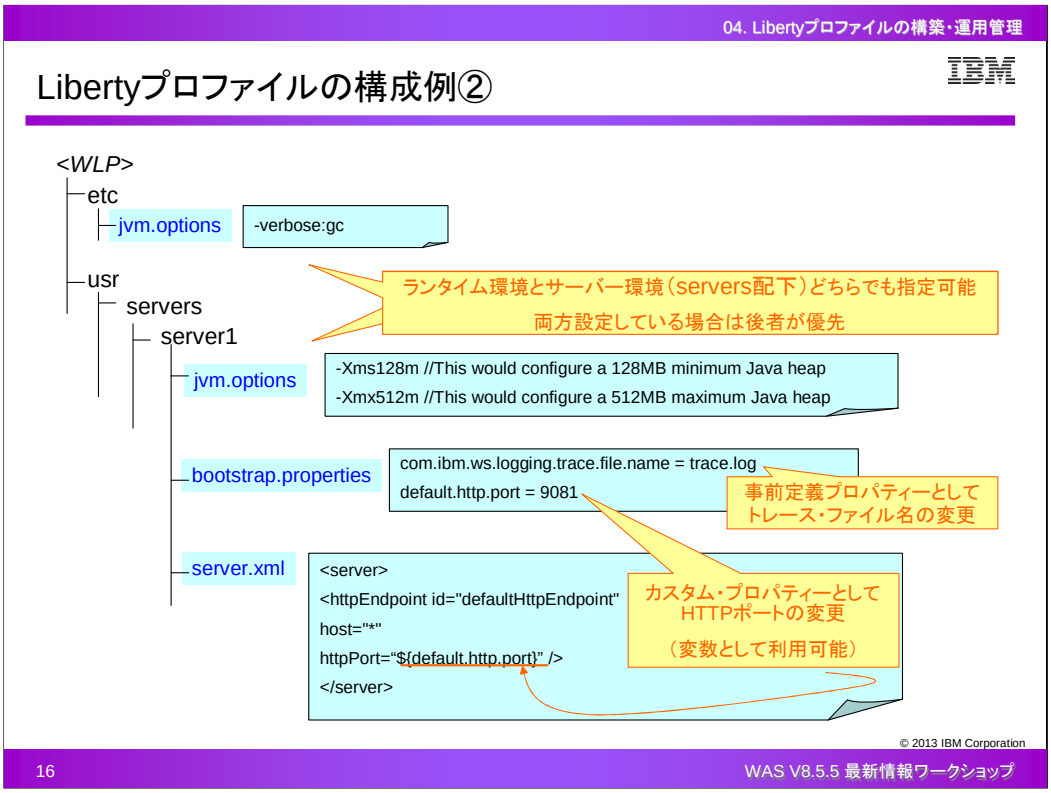
また、server.xmlの中で<include>エレメントを記述することにより、更に別のXML形式の構成ファイルに分割することができます。



構成例の1つ目を紹介します。

環境変数のカスタマイズを行うserver.envでは、Libertyプロファイルの稼働環境となるJREを指定することができます。server.envは、ランタイム環境としてLibertyプロファイルのetcディレクトリ配下にも、サーバー環境としてLibertyプロファイルのusrディレクトリの各サーバーディレクトリ配下にも配置することができます。両方に配置して同じ内容を設定している場合は、サーバー環境の設定が優先されます。

構成情報を記載するserver.xmlでは、使用するフィーチャーの指定や各種設定を行います。全ての構成を1つのserver.xml内に記載することもできますが、<include>エレメントを使用することにより、構成ファイルを分割して他のディレクトリやWebサーバーなど外部に配置しておくことができます。例えば、特定のサーバーに固有の変数を含むserver.xmlは各サーバーで保持し、メイン構成用の構成ファイルを個別ファイルにして複数のサーバーで共有するなど、環境に合わせた構成ファイルの構造を作成することができます。また、アプリケーション毎の構成をそれぞれ個別ファイルにすることで、アプリケーション毎に運用やバージョン管理を行うことができます。この例では、server1というサーバーの構成ファイルserver.xmlにおいて<include>エレメントを使用し、データソースに関する設定部分をdatasource.xmlという名称の個別ファイルに分割しています。



構成例の2つ目を紹介します。

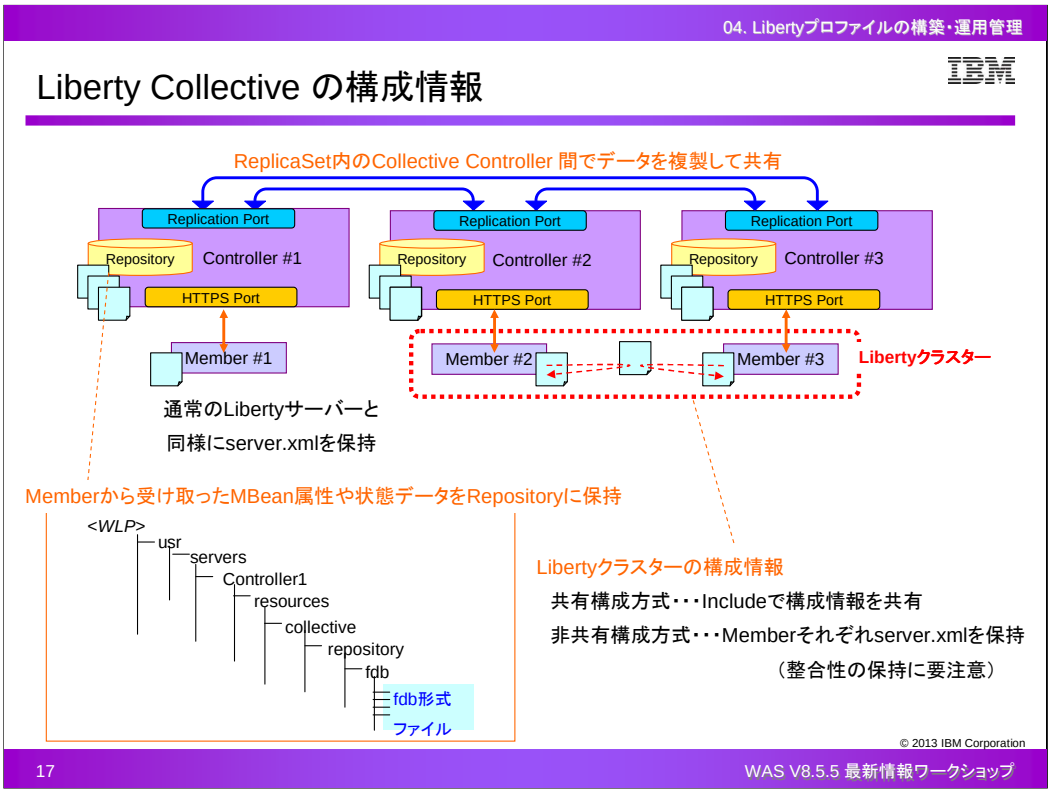
JVMのオプションのカスタマイズを行うjvm.optionsでは、ヒープサイズやその他のJVMオプションを指定することができます。jvm.optionsは、ランタイム環境としてLibertyプロファイルのetcディレクトリ配下にも、サーバー環境としてLibertyプロファイルのusrディレクトリの各サーバーディレクトリ配下にも配置することができます。両方に配置して同じ内容を設定している場合は、サーバー環境の設定が優先されます。この例では、ランタイム環境としてverbosegcログ出力の設定を行い、server1という特定のサーバー環境としてヒープサイズの設定を行っています。Libertyプロファイルでは、デフォルトのJVMオプションとして「-Xms4m」「-Xmx488m」「-XX:MaxPermSize=256m（Solarisのみ）」が設定されています。

ランタイム環境の初期化に使用されるbootstrap.propertiesには、事前定義プロパティーやカスタム・プロパティーを設定することができます。bootstrap.propertiesで設定したカスタム・プロパティーは、server.xml内で変数化して使用することができます。

参考：

Setting generic JVM arguments in the WebSphere Application Server V8.5 Liberty profile

http://www-01.ibm.com/support/docview.wss?uid=swg21596474



Liberty Collective 構成の場合、各Collective Member は通常のLibertyサーバーと同様に構成情報としてserver.xmlを保持しています。そして、Collective Controller は、Collective Memberから構成情報の一部やランタイム状況を受け取り、更にReplica Set のデータ複製によって受け取った情報も含めて、各Collective Controllerが保持するRepositoryに格納します。Repositoryの実態はファイル群であり、Collective Controller のサーバー情報が格納されているディレクトリ配下にfdb形式で保管されています。

Libertyクラスター構成の場合、構成情報の保持の仕方として、共有構成方式と非共有構成方式があります。非共有構成方式の場合は、それぞれのLibertyクラスターメンバーで構成情報を保持することになるので、Libertyクラスター全体に反映させたい設定を行なう際は整合性が保持されるように注意が必要です。

Libertyプロファイルの構成変更とランタイム操作

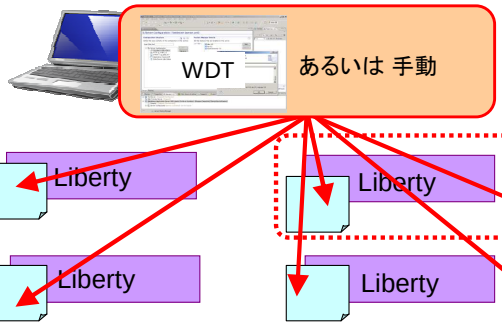
WDT（WebSphere Application Server Developer Tools）

- ・GUIベースで変更したものがserver.xmlに反映
- ・GUIベースで起動/停止実行

手動

- ・直接手動でserver.xmlを編集/置き換え
- ・サーバー単位で起動/停止コマンド実行

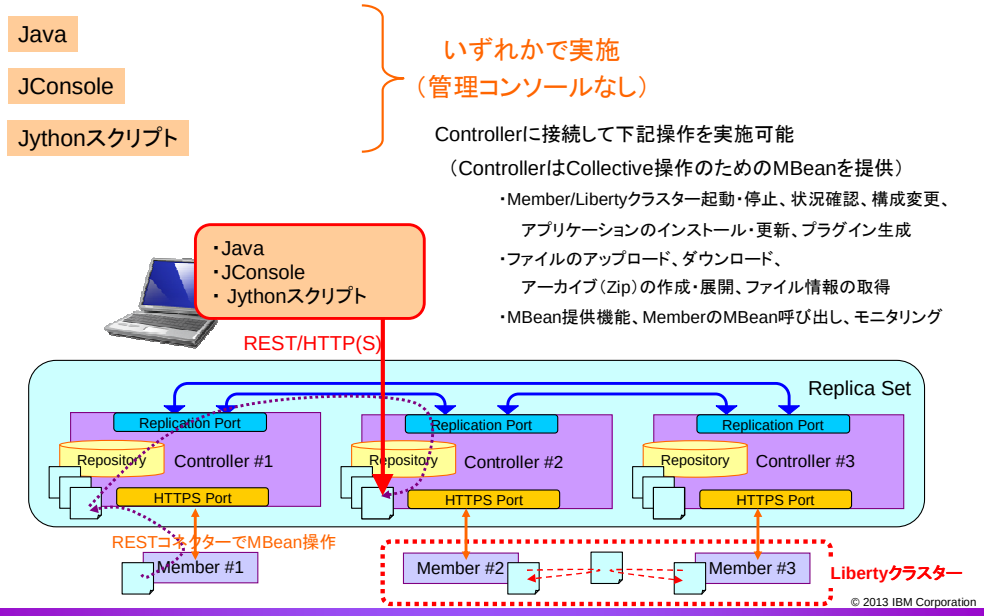
いずれかで実施
（管理コンソールなし）



Libertyプロファイルには管理コンソールが存在しないため、構成変更やランタイム操作は、WDT（WebSphere Application Server Developer Tools）か手動いずれかで実施する必要があります。

いずれの方法の場合においても、Liberty Collective でない構成であれば、Libertyサーバーそれぞれに対して接続を行い、操作を実施する必要があります。

Liberty Collective の構成変更とランタイム操作

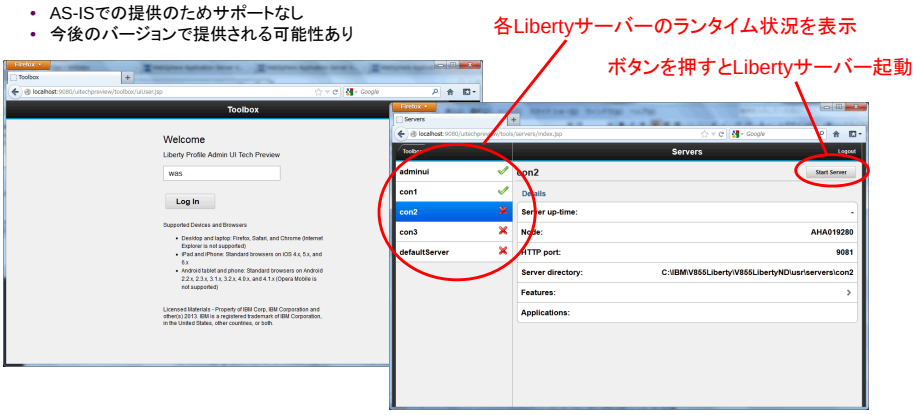


Liberty Collective の場合においても管理コンソールは存在せず、構成変更やランタイム操作には、JMXを使用したJavaアプリケーション、Jconsole、Jythonスクリプトいずれかの方法で実施する必要があります。

いずれの方法の場合においても、構成されているCollective Controller のうちいずれか1つに対して接続すれば、Collective Controller がCollective操作のためのMbeanを提供しているため、操作を行なうことができます。

【参考】Admin UI の紹介

- Admin UI
  - Libertyサーバー用の管理ユーザーインターフェース
  - 1つのアプリケーション（war）として提供
  - Libertyサーバーにデプロイしてブラウザからアクセスすることで実行
  - Libertyサーバーの起動/停止/リスト表示などのオペレーションをGUIで実行可能
  - Tech Preview として提供
    - 開発部門で開発中の技術・機能の事前公開
    - AS-ISでの提供のためサポートなし
    - 今後のバージョンで提供される可能性あり



IBM

構築

導入前提
server/collectiveコマンド
Liberty Collective 構成
IHS/プラグイン構成
セッション・パーシスタンス構成
Package機能
JobManager連携
PushOut構築モデル

導入前提

サポートしているJDK

Java6以上であればどのJDKもサポート

最小サポート・レベル
IBM JDK : 6.0 (J9 2.6) SR1
Oracle JDK : Java6 Update 26

分散プラットフォーム
32bit/64bitのJava共にサポート

z/OSプラットフォーム
64bitのJavaのみサポート

ただし・・・

Liberty Collective を使用する場合
WindowsのMemberに対してサーバーの起動や停止などの
リモート操作を行うためには**ControllerはIBM JDK で実行する必要あり**
（サード・パーティーJDKには必要なセキュリティー・クラスが含まれず）

導入方法

新規導入方法 → 詳細は前のセッション「Libertyプロファイルによる開発と構成」参照
①IIM（IBM Installation Manager）
②自己展開圧縮ファイル実行
③WDT（WebSphere Application Server Developer Tools）からダウンロード

Libertyプロファイルを稼働させるためのJDKは、Java6以上であればどの種類のJDKでもサポートされます。ただし、いくつかの制約があります。JDKの種類によっては最小サポート・レベルが決められており、IBM JDK は6.0(J9 2.6) SR1、Oracle JDK はJava6 SR26 となっています。また、分散プラットフォームでは32bitと64bitのJavaを共にサポートしていますが、z/OSプラットフォームでは64bitのJavaのみをサポートしています。

Liberty Collective を使用する場合は、上記チャートに記載されている制約がありますので、注意が必要です。

Libertyプロファイルの新規導入方法としては、①IIM（IBM Install Manager）②自己展開圧縮ファイル実行③WDTからダウンロードするという3つの方法があります。それぞれの詳細につきましては、「Libertyプロファイルによる開発・構成」をご参照ください。

Libertyプロファイルの基本的なコマンド

server コマンド ・・・ Libertyプロファイルを操作するコマンド（<WLP>/bin配下）

【基本書式】 server <Action> <Server_Name> [Options]

【Actionの種類】

- ・create ： サーバーの作成
- ・run/start/stop/debug ： サーバーの起動・停止
- ・dump/javadump(V8.5.0.1以降) ： 問題判別情報の収集
- ・package ： サーバー環境のパッケージング
- ・status/version/help ： 情報の表示

collective コマンド ・・・ Liberty Collectiveを操作するコマンド（<WLP>/bin配下） New

【基本書式】 collective <Action> <Server_Name> [Options]

【Actionの種類】

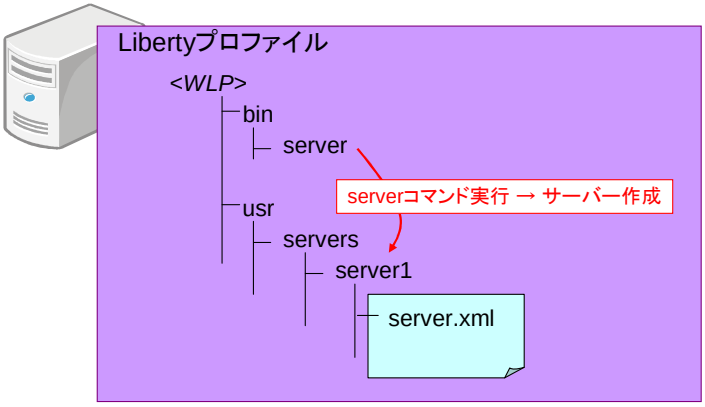
- ・create ： Controller の作成
- ・join ： LibertyサーバーをMemberとしてCollectiveに結合
- ・replicate ： LibertyサーバーをControllerとしてCollectiveに結合
- ・remove ： LibertyサーバーをCollectiveから除去
- ・registerHost/updateHost ： ホストをCollectiveに登録/登録されたホストの認証情報を更新
- ・unregisterHost ： CollectiveからホストおよびホストにIに関連づけられた全てのLibertyサーバーを登録抹消

Libertyプロファイルを構築する上で必要となる基本的なコマンドを紹介します。

1つ目はserverコマンドで、Libertyプロファイルを作成したり操作する際に使用します。指定できるActionとしては、サーバーの作成、起動・停止、問題判別情報の収集、サーバー環境のパッケージング、情報の表示、があります。

2つ目はcollectiveコマンドで、Liberty Collective を作成したり操作する際に使用します。指定できるActionとしては、Controllerの作成、MemberとしてCollectiveに結合、ControllerとしてCollectiveに結合、サーバーをCollectiveから除去、ホストをCollectiveに登録、登録されたホストの認証情報を更新、CollectiveからホストおよびホストIに関連づけられた全てのサーバーの登録抹消、があります。

serverコマンドによるサーバー作成



serverコマンド実行結果

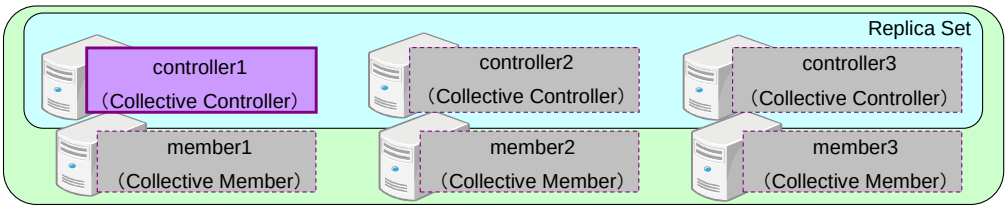
```
<WLP>¥bin>server create server1
サーバー server1 が作成されました。
```


Libertyプロファイルを導入すると、<WLP>/bin配下にserverコマンドがあります。

Actionとして「create」を指定してserverコマンドを実行することで、数秒でLibertyサーバーを作成することができます。Libertyサーバーを作成すると、<WLP>/usr/servers/<servername>配下に主要な構成ファイルであるserver.xmlが作成されます。

collectiveコマンドによるLiberty Collective構成（1/3）

使用するフィーチャー ： collectiveController-1.0



① serverコマンドでControllerになるLibertyサーバー「controller1」を作成

```
<WLP>¥bin>server create controller1
```

② collectiveコマンドで「controller1」をControllerとして構成

```
<WLP>¥bin>collective create controller1 --keystorePassword=con1password
```

③ ②の実行結果で出力される内容をserver.xmlにコピーして一部編集
フィーチャーや鍵ストアやトラストストアの情報など

④ Controller「controller1」を起動

```
<WLP>¥bin>server start controller1
```

```
<WLP>¥bin>collective create controller1 --keystorePassword=con1password
集合を確立するために必要な証明書を作成しています... ・・・・<略>・・・
controller1 の Collective Controller 構成を正常にセットアップしました
使用可能にするには、server.xml に次の行を追加します。
  <featureManager>
    <feature>collectiveController-1.0</feature>
  </featureManager>
  ・・・<略>・・・
```


Liberty Collective を構成する手順を紹介します。

ここでは、仮に、Collective Controller として「controller1」、「controller2」、「contorller3」、Collective Member として「member1」、「member2」、「member3」というLibertyサーバーを構成していくと想定します。

まず初めに、Collective Controller として「controller1」を構成します。

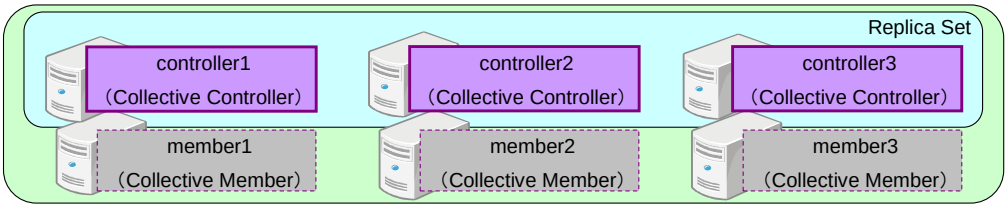
①serverコマンドのcreateアクションを実行してLibertyサーバー「controller1」を作成します。

②作成した「controller1」をcollectiveコマンドのcreateアクションを実行してCollective Controller として構成します。

③②の実行結果として、「使用可能にするには、server.xmlに次の行を追加します」というメッセージが出力されるので、その後に出力されている情報をコピーし、「controller1」のserver.xmlに添付し、パスワード情報など必要に応じて一部修正をします。

④serverコマンドのstartアクションを実行して「controller1」を起動すると、Collective Controller としての「controller1」が起動します。

collectiveコマンドによるLiberty Collective構成（2/3）



⑤ serverコマンドでControllerになるLibertyサーバー「controller2」「controller3」を作成

⑥ collectiveコマンドで「controller2」「controller3」をControllerとして結合（Replica Setへの追加）

```
<WLP>¥bin>collective replicate controller2 --host=localhost --port=9443 --user=admin
--password=adminpwd --keystprePassword=con2password
```

```
<WLP>¥bin>collective replicate controller3 --host=localhost --port=9443 --user=admin
--password=adminpwd --keystprePassword=con3password
```

⑦ ⑥の実行結果で出力される内容をserver.xmlにコピーして一部編集
フィーチャーや鍵ストアやトラストストアの情報など

```
<WLP>¥bin>collective replicate controller2 --host=localhost ･･･<略>･･･
ターゲット collective controller localhost:9443 を複製しています... ･･･<略>･･･
サーバー controller2 としてコントローラーを正常に複製しました
使用可能にするには、server.xml に以下の行を追加します。
    <featureManager>
        <feature>collectiveController-1.0</feature>
</featureManager>        ･･･<略>･･･
```

⑧ Controller「controller2」「controller3」を起動

⑨ RepositoryConfiguration MBeanの
addReplicaオペレーションを実行（Repositoryへの追加）

次に、「controller1」のReplica Set に「controller2」と「controller3」を追加し、構成します。

⑤serverコマンドのcreateアクションを実行してLibertyサーバー「controller2」と「controller3」を作成します。

⑥作成した「controller2」と「controller3」をcollectiveコマンドのreplicateアクションを実行してReplica Set に追加します。

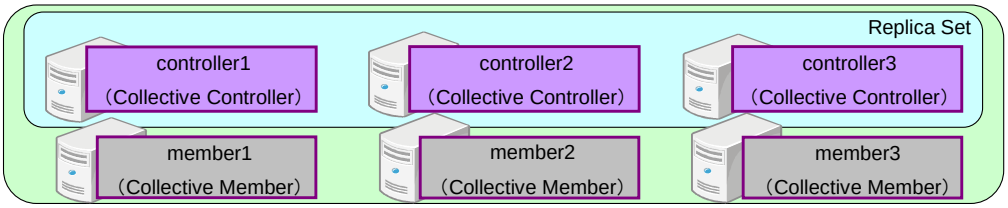
⑦⑥の実行結果として、「使用可能にするには、server.xmlに次の行を追加します」というメッセージが出力されるので、その後に出力されている情報をコピーし、「controller2」と「controller3」のserver.xmlにそれぞれ添付し、パスワード情報など必要に応じて一部修正をします。

⑧serverコマンドのstartアクションを実行して「controller2」と「controller3」を起動すると、Collective Controller としての「controller2」と「controller3」が起動します。

⑨JConsoleなどを使用して、RepositoryConfiguration MBean のaddReplicaオペレーションを実行することで、「controller2」と「controller3」をRepositoryに追加してアクティブ化します。

collectiveコマンドによるLiberty Collective構成（3/3）

使用するフィーチャー ： collectiveMember-1.0



⑩ serverコマンドでMemberになるLibertyサーバー「member1」「member2」「member3」を作成

⑪ collectiveコマンドでMemberとして結合

```
<WLP>¥bin>collective join member1 --host=localhost --port=9443 --user=admin --
password=adminpwd --keystorePassword=mem1password
```

⑫ ⑪の実行結果で出力される内容をserver.xmlにコピーして一部編集

フィーチャーや鍵ストアやトラストストアの情報など

```
<WLP>¥bin>collective join member1 --host=localhost ･･･<略>･･･
集合をターゲット・コントローラー localhost:9443 と結合しています...　･･･<略>･･･
サーバー member1 の集合を正常に結合しました
使用可能にするには、server.xml に次の行を追加します。
  <featureManager>
    <feature>collectiveMember-1.0</feature>
  </featureManager>
  ･･･<略>･･･
```

⑬ Member「member1」「member2」「member3」を起動

次に、作成したLiberty Collective にCollective Member として「member1」「member2」「member3」を構成します。

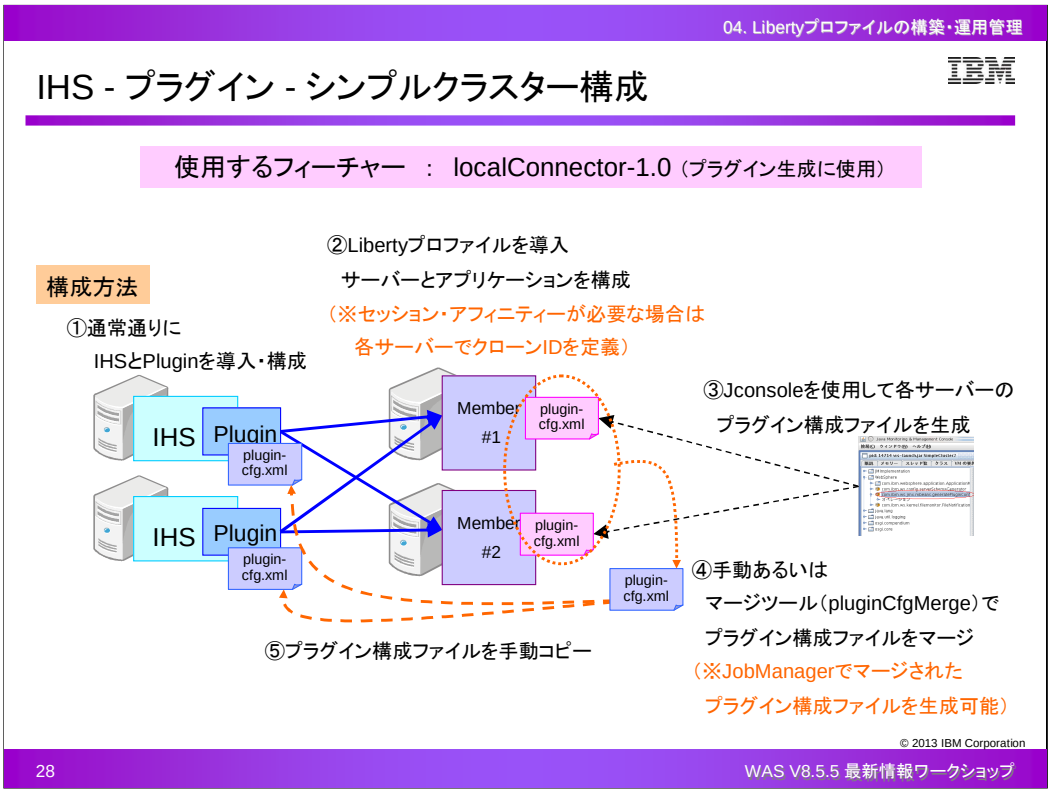
⑩serverコマンドのcreateアクションを実行してLibertyサーバー「member1」「member2」「member3」を作成します。

⑪作成した「member1」「member2」「member3」 をcollectiveコマンドのjoinアクションを実行してCollectiveに結合します。

⑫⑪の実行結果として、「使用可能にするには、server.xmlに次の行を追加します」というメッセージが出力されるので、その後に出力されている情報をコピーし、「member1」「member2」「member3」のserver.xmlにそれぞれ添付し、パスワード情報など必要に応じて一部修正をします。

⑬serverコマンドのstartアクションを実行して「member1」「member2」「member3」 を起動すると、Collective Memeber としての「member1」「member2」「member3」 が起動します。

以上の手順により、Liberty Collective が構成されます。



Libertyプロファイルでの IHS−プラグイン−シンプルクラスター構成方法は、まず、通常通りにIHSのノードに対してIHSとプラグインを導入して構成します。そして、複数のLibertyプロファイルのノードにLibertyプロファイルを導入し、サーバーとアプリケーションを構成します。そのとき、各サーバーのクローンIDを定義する必要があるため、各サーバーのserver.xml上に任意のクローンIDを記載します。次に、各サーバーのプラグイン構成ファイルをそれぞれJconsoleを使用して生成します。プラグイン構成ファイルはLibertyプロファイルの各サーバーディレクトリにそれぞれ作成されるので、それらを手動あるいはWASのフルプロファイルで提供されているマージツール（pluginCfgMergeコマンド）を使用してマージします。最後に、そのマージしたプラグイン構成ファイルをIHSノードに手動でコピーし、プラグインが読み込めるようにします。

以上の作業により、IHS−プラグイン−シンプルクラスターが構成されます。

IHS - プラグイン - Libertyクラスター構成

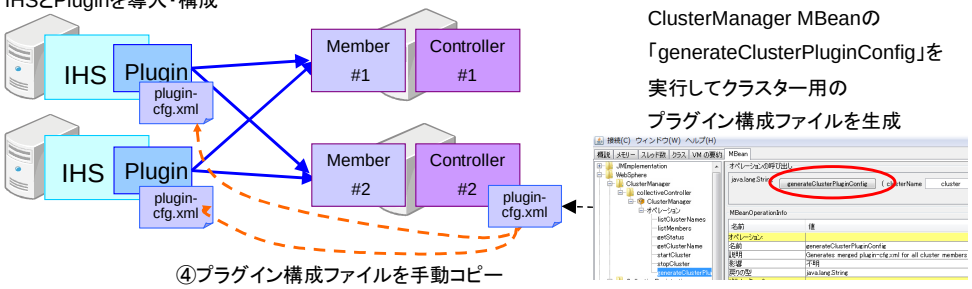
使用するフィーチャー ： clusterMember-1.0
localConnector-1.0（プラグイン生成に使用）

構成方法

①通常通りに
IHSとPluginを導入・構成

②Libertyプロファイルを導入
サーバーとアプリケーションを構成

③Jconsoleを使用して
ClusterManager MBeanの
「generateClusterPluginConfig」を
実行してクラスター用の
プラグイン構成ファイルを生成



④プラグイン構成ファイルを手動コピー

プラグイン構成ファイル生成時のControllerのログ

```
[13/07/05 11:19:14:460 JST] 00000075 com.ibm.ws.http.plugin.merge.internal.PluginMergeToolImpl  I Merged plugin config file written to
C:/IBM/V855Liberty/V855LibertyND/usr/servers/controller2/pluginConfig/cluster-plugin-cfg.xml

[13/07/05 11:19:14:460 JST] 00000075 com.ibm.ws.http.plugin.merge.internal.PluginMergeToolImpl  I Merge Complete
```

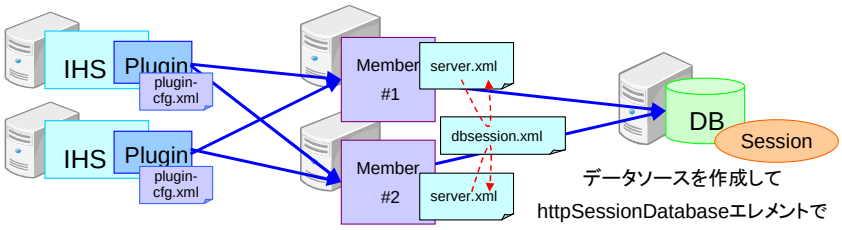

LibertyプロファイルでのIHS-プラグイン-Libertyクラスター構成方法は、まず、通常通りにIHSのノードに対してIHSとプラグインを導入して構成します。そして、複数のLibertyプロファイルのノードにLibertyプロファイルを導入し、Liberty Collective を構成します。次に、いずれかのCollective Controller に接続してクラスター用のプラグイン構成ファイルをJconsoleを使用して生成します。クラスター用のプラグイン構成ファイルは「cluster-plugin-cfg.xml」というファイル名で、接続先であるCollective Controller のサーバーディレクトリに作成されます。最後に、その作成されたクラスター用のプラグイン構成ファイルをIHSノードに手動でコピーし、プラグインが読み込めるようにします。

以上の作業により、IHS-プラグイン-Libertyクラスターが構成されます。

セッション・パーシスタンス構成

使用するフィーチャー ： sessionDatabase-1.0

構成方法



データソースを作成して
httpSessionDatabaseエレメントで
セッションDBを設定

server.xml

```
<pluginConfiguration webserverPort="80" webserverSecurePort="443" />
<include location="${shared.config.dir}/dbsession.xml" />
```

dbsession.xml

```
<feature>sessionDatabase-1.0</feature>
<fileset id="DB2Files" includes="*.jar" dir="${shared.resource.dir}" />
<library id="DB2Lib" filesetRef="DB2Files" />
<jdbcDriver id="DB2Driver" libraryRef="DB2Lib" />
<dataSource id="SessionDS" jdbcDriverRef="DB2Driver" jndiName="jdbc/sessions">
<properties.db2.jcc driverType="4" databaseName="SESSION" serverName="LibertyDB"
portNumber="50000" user="root" password="rootpassword" />
</dataSource>
<httpSessionDatabase id="SessionDB" dataSourceRef="SessionDS" />
```

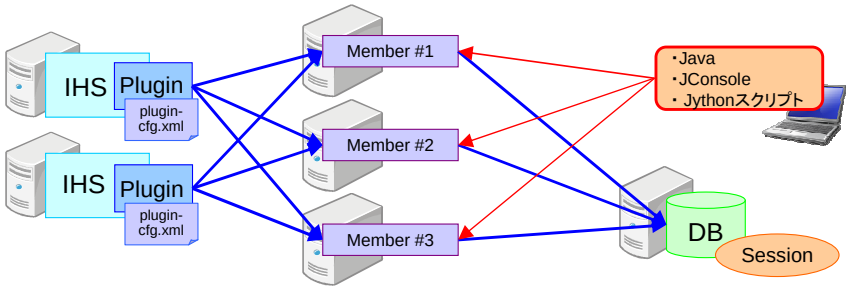

Libertyプロファイルでセッション・パーシスタンス構成を行う場合、sessionDatabase-1.0というフィーチャーを使用します。

構成方法は、シンプルクラスターあるいはLibertyクラスターを構成したら、あとはserver.xml上にセッション情報格納先に関する構成エレメントを追加するだけです。

以上の作業によりセッション・パーシスタンス構成が行われ、特定のサーバー障害によりメモリ上のセッション情報が失われても、外部保管しているセッション情報からセッションのフェールオーバーを実施することができるようになります。

Libertyプロファイルでは、メモリ間複製はサポートされていません。

新規構築例①：シンプルクラスターの新規構築の流れ



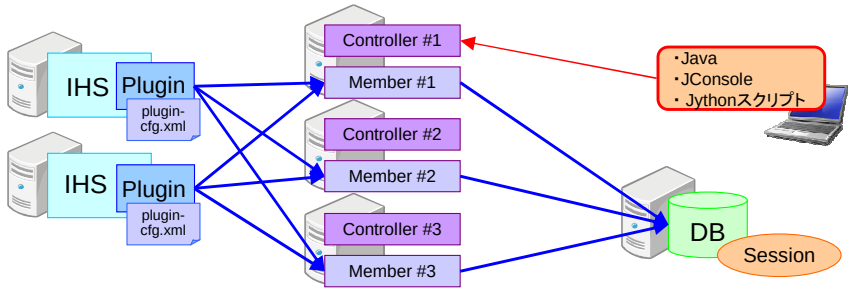
- 構成
  - IHS2台、Libertyシンプルクラスターメンバー3台、セッションパーシスタンス有
- 構築の流れ
  - ①通常通りIHSとPluginを導入・構成、DBを導入・構成
  - ②3台のノードにWAS-Libertyを導入
  - ③各ノードに1つずつLibertyサーバーを作成
  - ④Libertyサーバーを構成
    - Includeを使用して構成ファイルを共有
    - アプリケーション配置場所を共有
    - 構成ファイルserver.xmlでクローンIDを定義
    - セッションパーシスタンス構成
  - ⑤アプリケーションをデプロイ
  - ⑥JConsole等を使用して各Libertyサーバーのプラグイン生成
  - ⑦プラグイン構成ファイルのマージ
  - ⑧プラグイン構成ファイルの手動配布

シンプルクラスターの新規構築の流れについてまとめます。

ここでは、IHSおよびIHSプラグインのノード2台、Libertyシンプルクラスターメンバーのノード3台、セッションパーシスタンス有効、というシステムを構築すると想定します。

構築の流れは、上記チャートの通りとなります。

新規構築例②：Libertyクラスターの新規構築の流れ

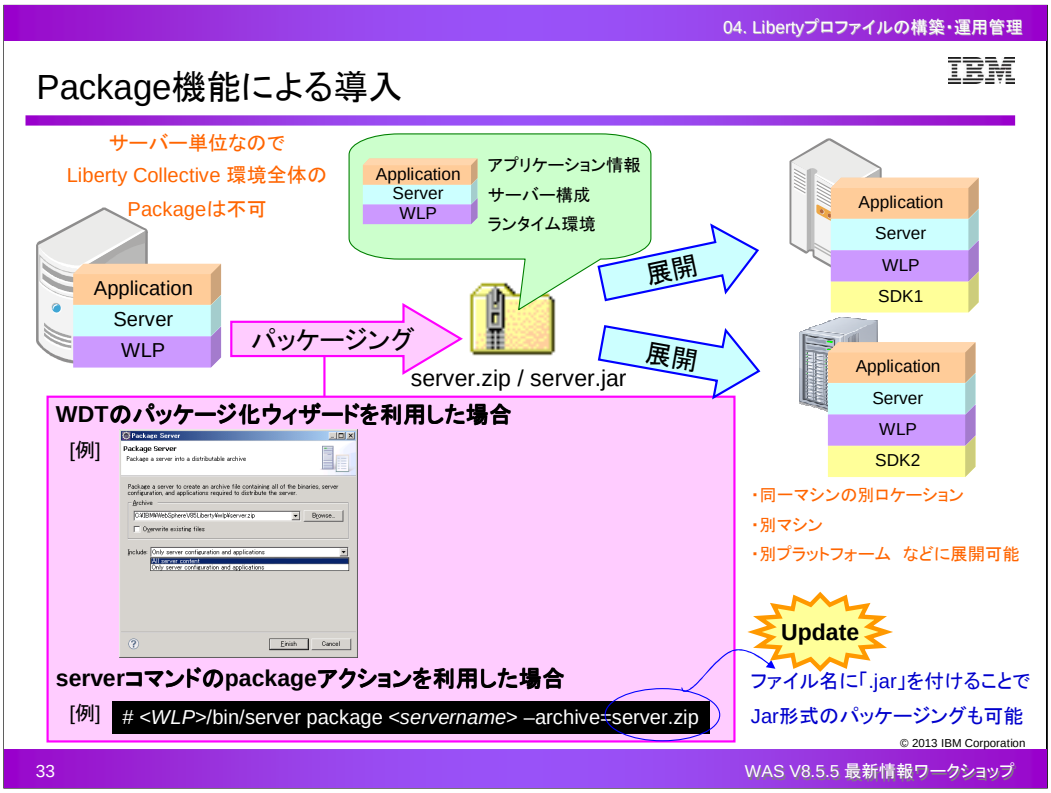


- 構成
  - IHS2台、Libertyクラスターメンバー3台、セッションパーシスタンス有
- 構築の流れ
  - ①通常通りIHSとPluginを導入・構成、DBを導入・構成
  - ②3台のノードにWAS-Libertyを導入
  - ③各ノードに2つずつLibertyサーバーを作成
  - ④Liberty Collectiveを構成
    - Liberty Collective 構成（create → replicate → join）
    - Includeを使用して構成ファイルを共有
    - アプリケーション配置場所を共有
    - ClusterGroupを定義
    - 高可用性のために複数Controllerを設定
    - セッションパーシスタンス構成
  - ⑤アプリケーションをデプロイ
  - ⑥JConsole等を使用してLibertyクラスターのプラグイン生成
  - ⑦プラグイン構成ファイルの手動配布

Libertyクラスターの新規構築の流れについてまとめます。

ここでは、IHSおよびIHSプラグインのノード2台、Libertyクラスターメンバーのノード3台、セッションパーシスタンス有効、というシステムを構築すると想定します。

構築の流れは、上記チャートの通りとなります。



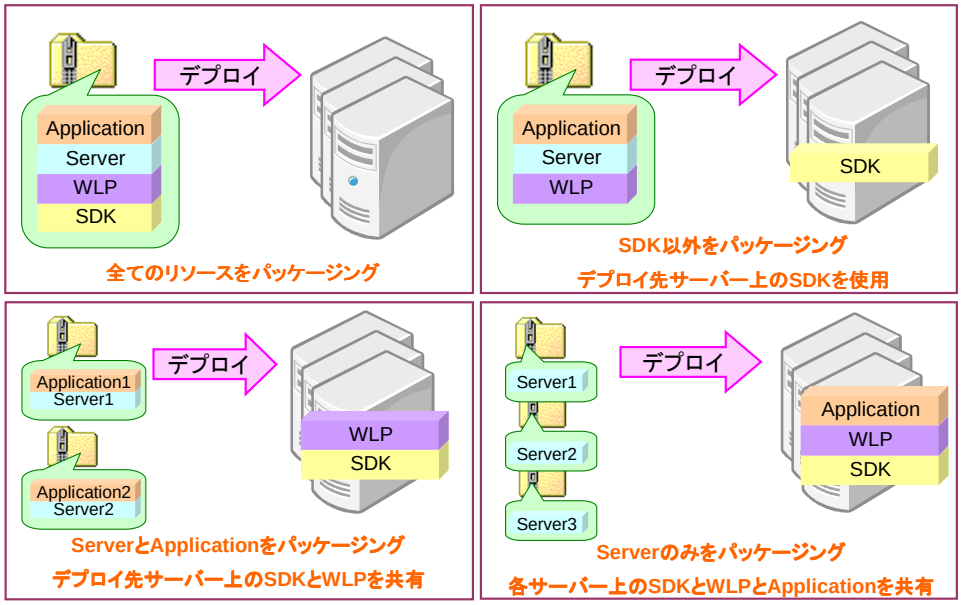
Package機能によるインストールとは、一度構築したランタイム環境、サーバー構成、アプリケーション情報を全て1つのzip形式の圧縮ファイルにパッケージングし、それをzip展開することによってLibertyプロファイルを導入するという方法です。

パッケージングを行う機能はLibertyプロファイルが提供しており、WDTのパッケージ化ウィザードを利用したGUIベースで実行することも、serverコマンドのpackageアクションを利用したコマンドベースで実行することもできます。

zip展開先としては、パッケージングを行った同一マシンの別ロケーションでも、別マシン上でも、別プラットフォームのマシン上でも可能です。別プラットフォームのマシンにzip展開する場合、ディレクトリ名やファイル名の大文字小文字区別の要否などに関しては考慮する必要があります。

V8.5では、Package機能によりパッケージングされた後のファイルはzip形式の圧縮ファイルでしたが、V8.5.5の新機能として、実行時にファイル名に「.jar」を付けることにより、Jar形式の圧縮ファイルにすることもできるようになりました。

Package機能による導入パターン



Package機能を利用したインストール・パターンとして、ここでは4つのパターンを紹介します。

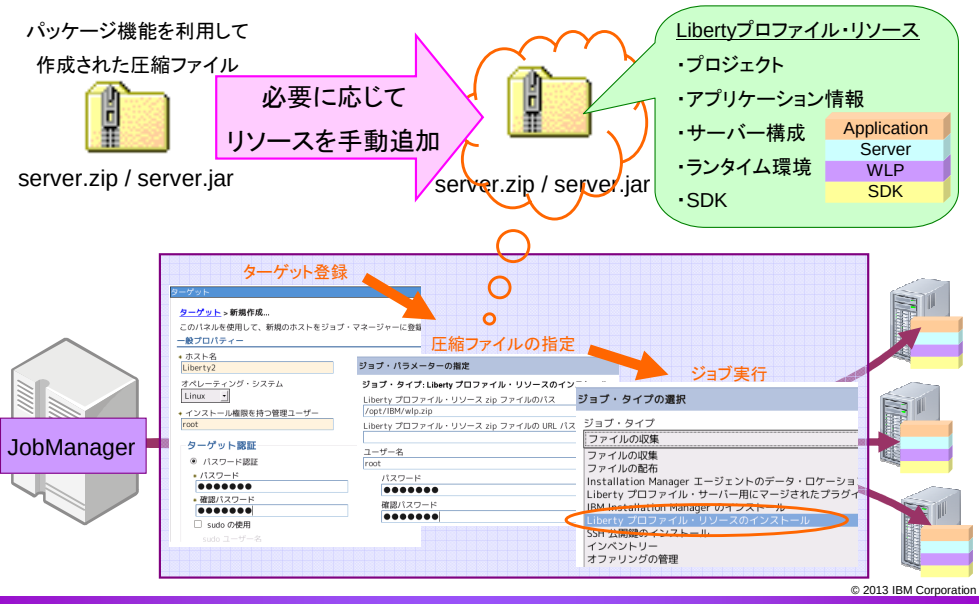
1つ目は、全てのリソースをパッケージングし、デプロイ先となるサーバー上で展開してそのまま使用するというパターンです。デプロイ先のサーバーには特に準備や設定は不要です。

2つ目は、SDK以外のリソースをパッケージングし、デプロイ先となるサーバー上のSDKを使用するというパターンです。デプロイ先のサーバーにSDKだけ準備されていれば、Libertyプロファイルのインストール作業やサーバーの構成、アプリケーションのデプロイといった作業は不要です。

3つ目は、サーバー構成とアプリケーション情報をパッケージングし、デプロイ先となるサーバー上のSDKとLibertyプロファイルを使用するというパターンです。デプロイ先のサーバーにSDKとLibertyプロファイルが準備されていれば、さまざまなサーバー構成とアプリケーション情報の組み合わせをデプロイし、SDKとLibertyプロファイルを共有することができます。

4つ目は、サーバー構成のみをパッケージングし、デプロイ先となるサーバー上のSDKとLibertyプロファイルとアプリケーション情報を使用するというパターンです。デプロイ先のサーバーにSDKとLibertyプロファイルとアプリケーション情報が準備されていれば、さまざまなサーバー構成をデプロイしてそれらを共有することができます。

JobManagerによる導入



JobManagerによる導入方法では、事前にLibertyプロファイル・リソースと呼ばれるzip形式の圧縮ファイルを作成しておき、それをターゲットとして登録したサーバーに対してジョブとして実行することで、ファイルが展開され、インストールが行われます。

Libertyプロファイル・リソースとは、プロジェクト、アプリケーション情報、サーバー構成、ランタイム環境、SDKの情報がパス名やグループ化などルールに則ったディレクトリ構造で配置されたzipファイルです。手動で新規にzipファイルを作成して必要なリソースを追加していくことも可能ですが、パッケージ機能を利用してzipファイルを作成し、そこに必要に応じてリソースを追加していくことでLibertyプロファイル・リソースを作成することもできます。

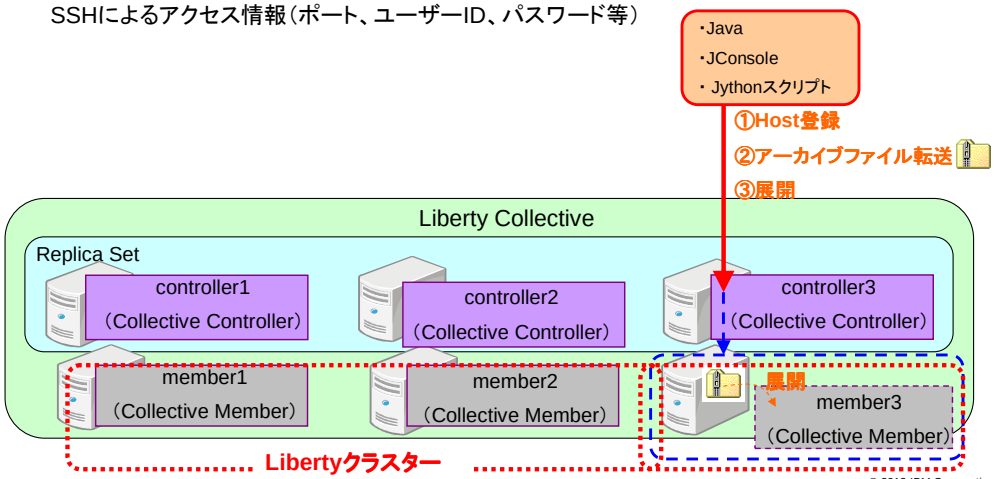
Libertyプロファイル・リソースの中のプロジェクトとは、リソースのコンテナー（オプション）です。関連するリソースを同じプロジェクトの下でグループ化することによって、管理を容易にし、他のプロジェクトからのリソースとの名前の競合を避けることができます。また、Libertyプロファイル・リソースの中には、Libertyプロファイルのサーバーを実行するためのSDKも追加することができます。

Liberty Collective のPushOut構築モデル

PushOut構築

Collectiveに登録されたLibertyの存在しないHostに対して管理クライアントからMemberを構築

SSHによるアクセス情報（ポート、ユーザーID、パスワード等）

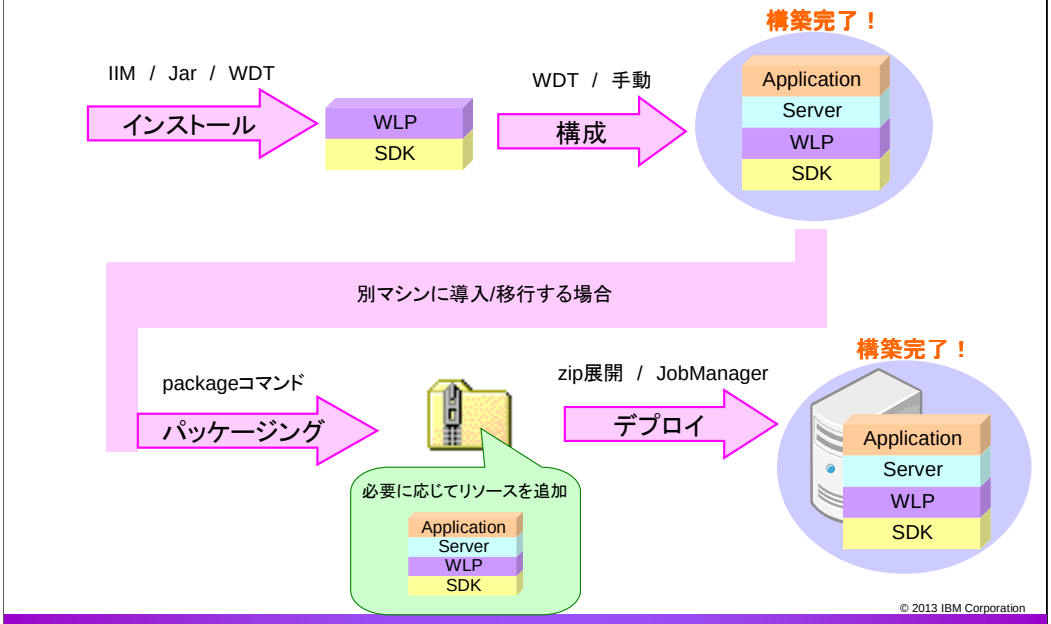


Liberty Collective のPushOut構築モデルという構築方法では、JobManagerを使用することなく、事前にPackageコマンドで作成しておいたzip形式の圧縮ファイルをリモートのサーバーに対して送信してそのままインストールを行なう、ということができます。

管理クライアントからCollective Controller に接続し、Libertyプロファイルが導入されていないリモートのノードをLiberty Collective にHostとして登録することができます。そして、事前に作成しておいたzip形式の圧縮ファイルを指定することで、そのファイルがHostに送信され、展開され、そのままLibertyプロファイルが導入され、Liberty Collective として構成されます。

このLibertyプロファイルが導入されていない段階でのHostに対しては、SSHを使用して接続されます。

Libertyプロファイル構築の流れ

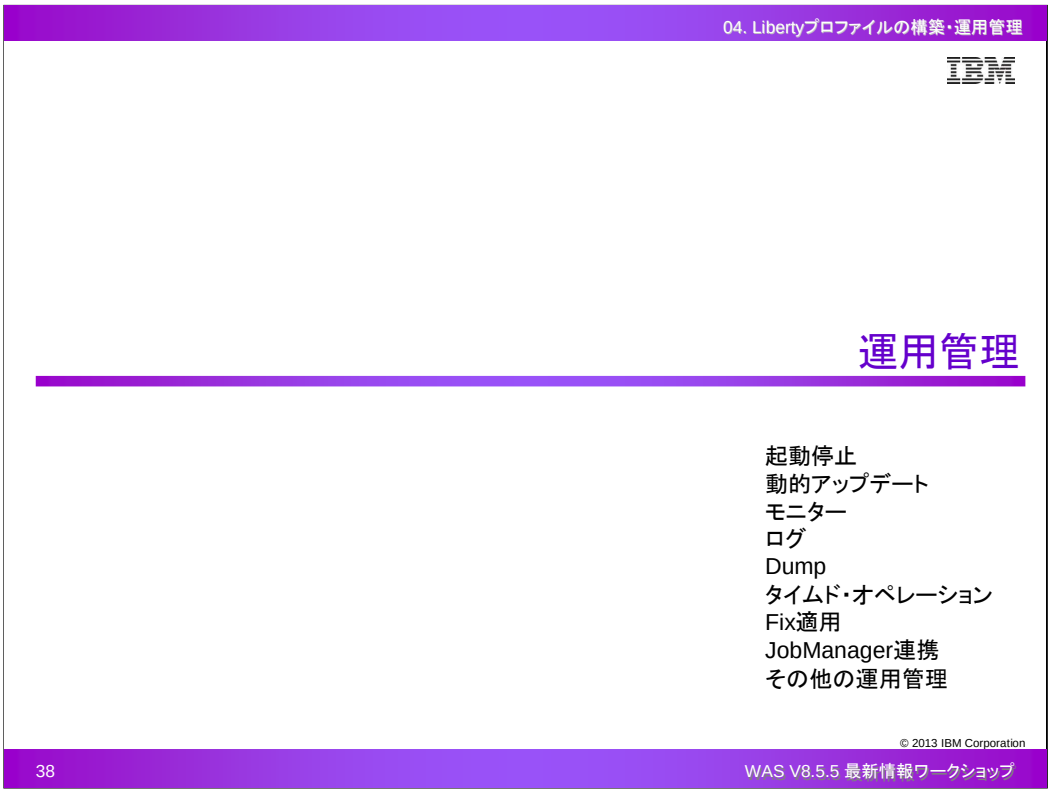


Libertyプロファイルを本番環境に導入し、構築する場合の全体の流れをまとめます。

まず初めに、実行環境となるSDKを準備し、IIM/Jar/WDTいずれかの方法によりLibertyプロファイルをインストールします。次に、WDT/手動いずれかの方法によりアプリケーションの実行に必要なサーバーの構成を行い、アプリケーションのデプロイを行います。これで、Libertyプロファイルの導入と構築は完了です。

この構成をそのまま別マシンに導入/移行する必要がある場合は、パッケージ機能によるzipファイルの作成および必要に応じてリソースの追加を行い、それをzip展開/JobManagerいずれかの方法によりターゲットとなるサーバー上に導入します。これで、別マシンへのLibertyプロファイルの導入と構築は完了です。

例えば、開発環境でLibertyプロファイルの環境を全て構築した後に、パッケージ機能を利用して本番環境へ移行すれば、新たに本番環境においてインストールや構成の作業を行う必要がなく、開発環境との設定の差異なども発生することはありません。


04. Libertyプロファイルの構築・運用管理
IBM
運用管理
起動停止
動的アップデート
モニター
ログ
Dump
タイムド・オペレーション
Fix適用
JobManager連携
その他の運用管理
© 2013 IBM Corporation
38
WAS V8.5.5 最新情報ワークショップ

起動停止

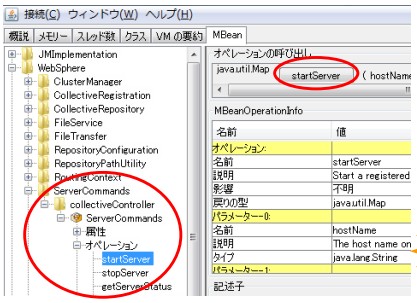
serverコマンドでの起動停止

```
<WLP>¥bin>server start server1
サーバー server1 を始動中です。
サーバー server1 が始動しました。
```

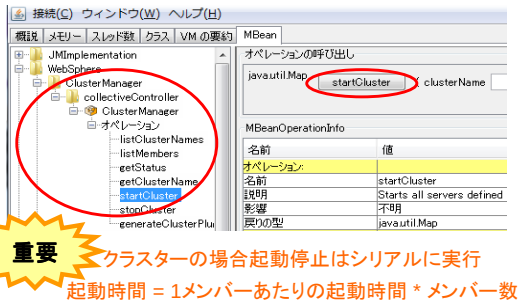
```
<WLP>¥bin>server stop server1
サーバー server1 を停止中です。
サーバー server1 は停止しました。
```

Liberty Collective の起動停止

【Liberty Collective 構成の場合】
Controllerから提供される ServerCommands MBean
の実行によりMemberの起動停止が可能



【Libertyクラスター構成の場合】
Controllerから提供される ClusterManager MBean
の実行によりクラスターの起動停止が可能

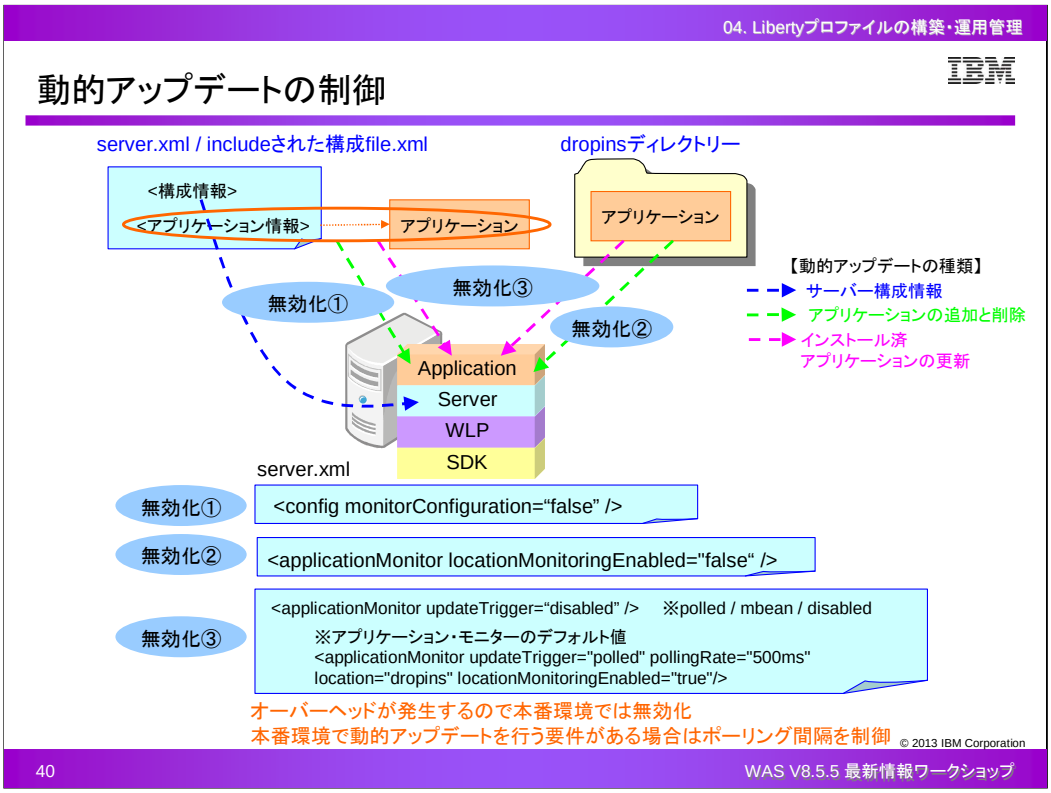


重要 クラスターの場合起動停止はシリアルに実行
起動時間 = 1メンバーあたりの起動時間 * メンバー数

通常のLibertyサーバーは、serverコマンドのstartアクション・stopアクションを実行して起動・停止を行うことができます。

Liberty Collective 構成の場合、Collective Controller で提供されるServerCommands MBean の startServer・stopServerのオペレーションを実行することで、Collective Member の起動・停止を行います。

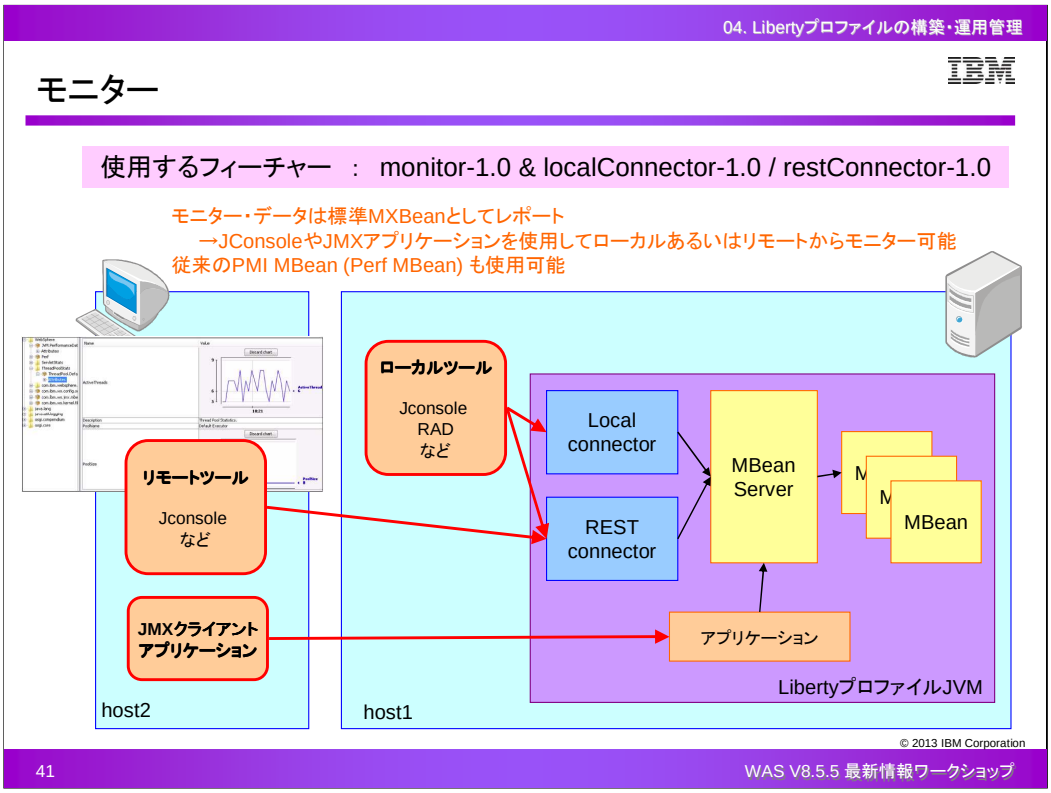
Libertyクラスター構成の場合、Collective Controller で提供されるClusterManager MBean の sartCluster・stopClusterのオペレーションを実行することで、Libertyクラスター全体の起動・停止を行います。ここで注意が必要なのは、Libertyクラスター構成の場合、起動・停止は各Memberシリアルに実行されるため、Libertyクラスター全体の起動・停止時間は、1メンバーあたりにかかる起動・停止時間とメンバー数の積になり、Member数が多ければそれだけ多くの時間がかかることになります。



Libertyプロファイルは、サーバー構成やアプリケーション情報を動的にアップデートすることができますが、その機能を必要に応じて制御することができます。

動的アップデートの種類としては、サーバー構成情報、アプリケーションの追加と削除、インストール済アプリケーションの更新、の3つがあります。サーバー構成情報は、server.xmlあるいはincludeされた構成ファイルから動的アップデートを行うことができます。アプリケーションの追加と削除およびインストール済アプリケーションの更新は、server.xmlあるいはincludeされた構成ファイルやアプリケーション配置ディレクトリー、dropinsディレクトリーから動的アップデートを行うことができます。これらの動的アップデート機能に関して、それぞれの方法で無効化あるいはモニターのポーリング間隔の変更を行うことができます。

構成情報をモニターする構成サービスやアプリケーション・モニターによるポーリングはオーバーヘッドが発生するので、本番環境では無効化します。本番環境で動的アップデートを行う要件がある場合は、ポーリング間隔を長く設定するなどして、できるだけオーバーヘッドが少なくなるように対応します。



Libertyプロファイルのサーバー・ランタイム環境をモニターするためにはmonitor-1.0フィーチャーを使用します。そして、ローカルツールからモニター・データを確認するためにはlocalConnector-1.0フィーチャー、リモートツールからモニター・データを確認するためにはrestConnector-1.0フィーチャーを使用します。

monitor-1.0フィーチャーを有効にするとモニターが開始され、モニター・データは標準MXBeanとしてレポートされます。モニター用のMXBeanとして、「WebSphere:type=JvmStats」(JVM)、「WebSphere:type=ServletStats,name=*」(スレッドプール)、「WebSphere:type=ThreadPoolStats,name=DefaultExecutor」(Webアプリケーション)などがあります。また、従来のPMI MBean(Perf MBean)も使用できますので、必要に応じてモニター・データを確認することができます。

localConnector-1.0フィーチャーでは、ローカルJMXコネクターが提供されます。このコネクターはJVMに組み込まれ、同一ホスト・マシン上で同一ユーザー ID、同一JDKで実行しているユーザーのみが使用することができます。Jconsoleなどの JMXクライアントやAttach API を使用するその他のJMX クライアント、RADなどによるローカル・アクセスが可能になります。

restConnector-1.0フィーチャーでは、セキュアなJMXコネクターが提供されます。このコネクターは、任意のJDKを使用してローカルまたはリモートで使用することができます。RESTベース・コネクター経由のJMX クライアントによるリモート・アクセスが可能になります。SSLおよび基本ユーザー・セキュリティー構成が必要になります。

【参考】モニター・データ（1/2）

JVM

- Heap: 現行JVMに使用されているヒープ・サイズ
- FreeMemory: 現行JVMに使用可能な空きヒープ
- UsedHeap: 現行JVMの使用済みヒープ
- ProcessCPU: JVMプロセスで使用されたCPUのパーセンテージ
- GcCount: JVMの始動以降にGCが行われた回数
- GcTime: GC時間の合計累算値
- UpTime: JVM の始動以降の時間(ミリ秒単位)

スレッドプール

- PoolName: スレッドプール名
- AcitiveThreads: 要求を処理中のアクティブ・スレッド数
- PoolSize: スレッドプールのサイズ

Webアプリケーション

- AppName: アプリケーションの名前
- ServletName: サーブレットの名前
- RequestCount: このサーブレットに対するヒット数
- ResponseTime: 平均応答時間 (ナノ秒)
- Description: カウンターの説明
- RequestCountDetails: 最終タイム・スタンプなど、RequestCount の詳細
- ResponseTimeDetails: 取得したスナップショットの数、最小値、最大値など、ResponseTime の詳細

【参考】モニター・データ(2/2)

IBM

JAX-WS

- AvgResponseTime: 平均応答時間 (ミリ秒)
- MaxResponseTime: 最大応答時間 (ミリ秒)
- MinResponseTime: 最小応答時間 (ミリ秒)
- NumInvocations: このエンドポイントまたは操作への呼び出しの数
- NumChekedApplicationFaults: 検査されたアプリケーション障害の数
- NumLogicalRuntimFaluts: 論理実行時障害の数
- NumRuntimeFaults: 実行時障害の数
- NumUnCheckedApplicationFaults: 未検査のアプリケーション障害の数
- TotalHandlingTime: 合計応答処理時間 (ミリ秒)

New

セッション

- CreateCount: 作成されたセッションの総数
- LiveCount: ライブ・セッションの総数
- ActiveCount :アクティブ・セッションの総数
- InvalidatedCount :無効化されたセッションの総数
- InvalidatedCountbyTimeout :タイムアウトにより無効化されたセッションの総数

New

**データソース
コネクションプール**

- CreateCount:作成された接続の総数
- DestroyCount: 破棄された接続の総数
- ManagedConnectionCount :使用中の ManagedConnection オブジェクトの数
- WaitTime :接続が認可されるまでの平均待ち時間 (ミリ秒)
- ConnectionHandleCount :使用中の Connection オブジェクトの数
- FreeConnectionCount :プール内の空き接続の数

New

モニター対象のフィルタリング

New

IBM

フィルタリング機能

動的にモニター対象を変更可能

server.xml

```
<monitor filter="JVM,ThreadPool,WebContainer,Session,ConnectionPool"/>
```

<monitor>が記載されていない場合　→全てモニター対象
<monitor>が記載されて指定がない場合　→モニター対象なし

＜例＞JVMとコネクションプールのみモニターしたい場合

```
<monitor filter="JVM,ConnectionPool"/>
```

※　V8.5.5でフィルタリングできるのはコンポーネントレベルのみでカウンターレベルでは不可

V8.5.5の新機能として、Libertyプロファイルのモニターにフィルタリングができるようになりました。

モニター対象は、server.xmlに<monitor filter>エレメントを記述することで動的に変更することができます。特に何の記述もない場合、monitor-1.0フィーチャーを有効にした時点で全ての項目のモニターが行われます。<monitor filter>エレメントの中に項目を記述すると、その項目のみがモニターされるようになります。また、<monitor filter>エレメントを記述して中をブランクにすると、どの項目もモニターされないようになります。

V8.5.5のフィルタリング機能では、コンポーネントレベルでの設定のみであり、カウンターレベルでフィルタリングすることはできません。

Libertyプロファイルの主なログは、console.log、messages.log、trace_<timestamp>.log、ffdc_<timestamp>.logなどがあり、デフォルトではLibertyプロファイルの各サーバーディレクトリ配下のlogsディレクトリに出力されます。

console.logには、基本的なサーバー状況およびオペレーション・メッセージが出力されます。タイムスタンプは出力されません。JVMオプションでverbosegcを有効にした場合のログは、console.logに出力されます。

messages.logには、SystemOutやSystemErrおよびその他のメッセージが出力されます。タイムスタンプと発行元スレッドIDが出力されますので、障害時の問題判別の際などは、こちらのログを確認します。

trace_<timestamp>.logには、現行トレース構成で設定された詳細レベルのトレースやメッセージが出力されます。タイムスタンプは出力されます。

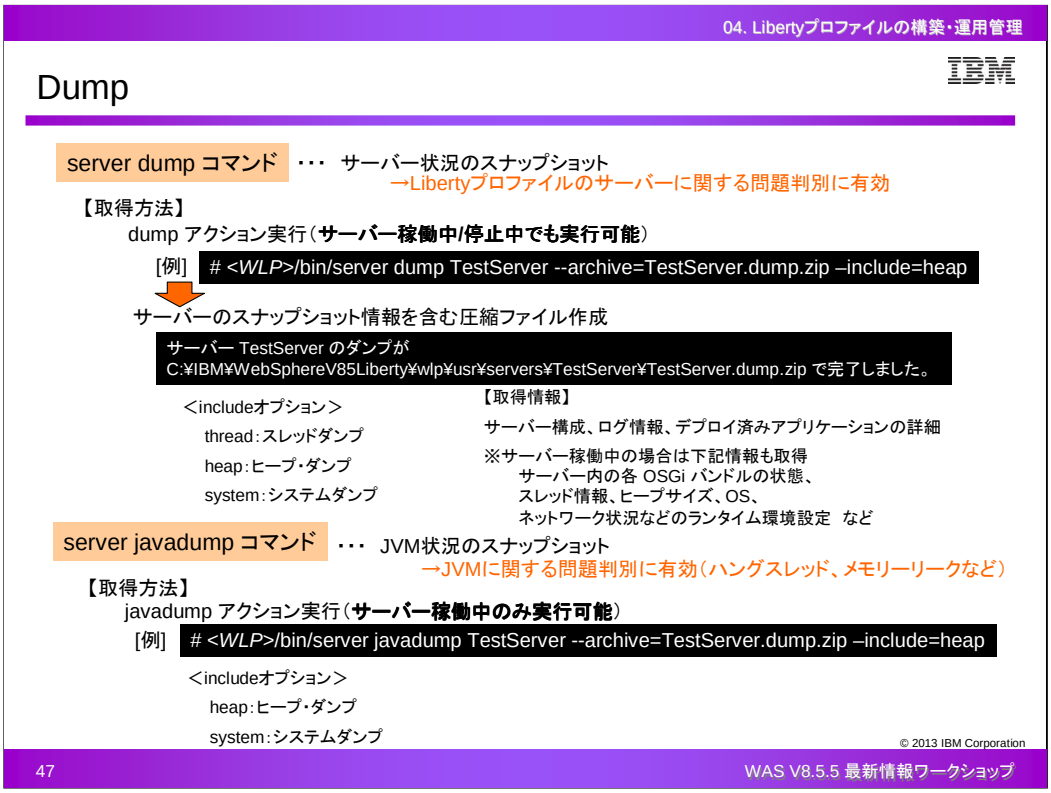
ffdc_<timestamp>.logには、初期障害データ・キャプチャー機能（FFDC）のメッセージが出力されます。通常は、要求されたオペレーションの失敗に関連する診断データの指定域ダンプが含まれます。

ログファイルの出力先や最大サイズや出力内容などの詳細は、ロギング・プロパティーとしてserver.xmlあるいはbootstrap.propertiesで設定することができます。

04. Libertyプロファイルの構築・運用管理

【参考】ロギング・プロパティー

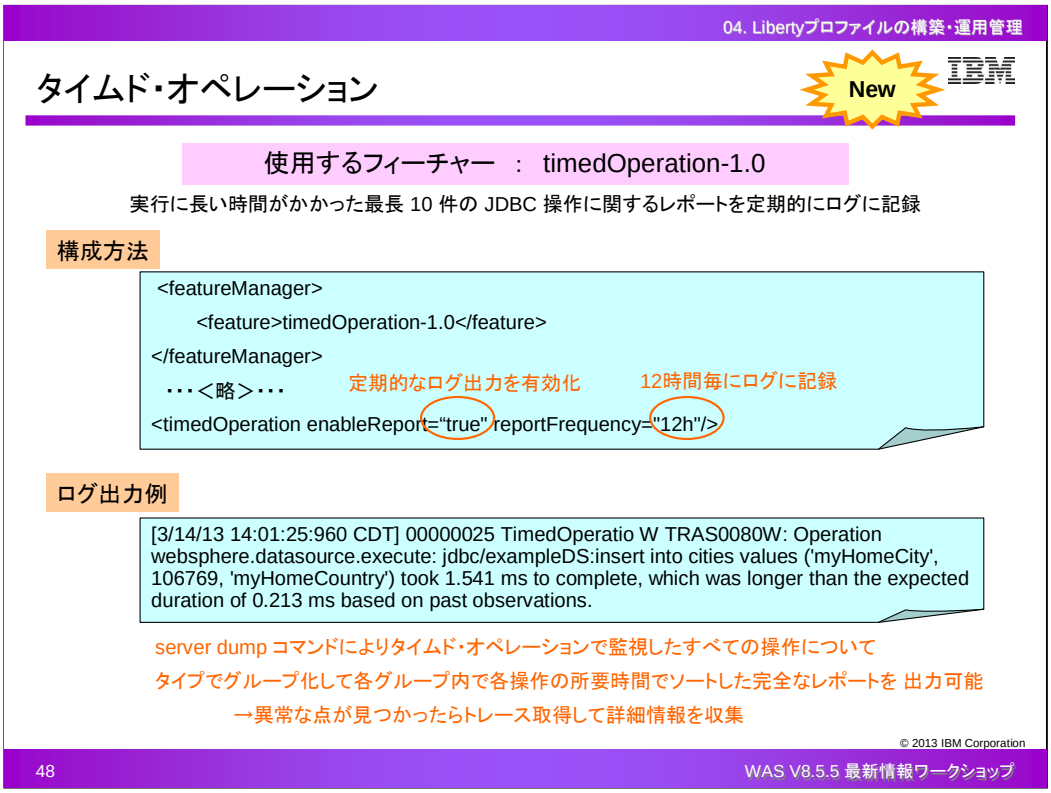
server.xmlで設定する属性	bootstrap.propertiesで設定するプロパティー	説明
logDirectory	com.ibm.ws.logging.log.directory	FFDC を含むすべてのログ・ファイルのディレクトリーを設定します。
maxFileSize	com.ibm.ws.logging.max.file.size	ログ・ファイルの許容最大サイズ(MB)。これを超えるとローテートされます。これを無効にするには、値を 0 に設定します。
maxFiles	com.ibm.ws.logging.max.files	最大ファイル・サイズが有効な場合に、この設定を使用して、保持する各ログ・ファイルの数を決定します。
consoleLogLevel	com.ibm.ws.logging.console.log.level	console.log ファイルに入れるメッセージの細分度を制御します。有効な値は INFO、AUDIT、WARNING、ERROR、および OFF です。デフォルトのレベルは AUDIT です。
copySystemStreams	com.ibm.ws.logging.copy.system.streams	true の場合、System.out はシステム出力ストリーム に書き込まれ、System.err はシステム・エラー・ストリームに書き込まれます。false の場合、System.out および System.err は、構成されたログ (messages.log や trace.log など) に書き込まれますが、システム・ストリームには書き込まれません。デフォルト値は true です。
messageFileName	com.ibm.ws.logging.message.file.name	メッセージ・ログのデフォルト名は messages.log です。このファイルは常に存在し、System.out と System.err に加えて、INFO とその他 (AUDIT、WARNING、ERROR、FAILURE) のメッセージが含まれます。このログには、タイム・スタンプと発行元スレッド ID も含まれます。
suppressSensitiveTrace		サーバー・トレースは、ネットワーク接続経由で受信したバイトなどの型のないデータをトレースする場合に、機密データを公開してしまう可能性があります。この属性を true に設定すると、機密である可能性のある情報がログ・ファイルおよびトレース・ファイルで公開されないようにします。デフォルト値は false です。
traceFileName	com.ibm.ws.logging.trace.file.name	追加トレースまたは詳細トレースが有効な場合にのみ作成されます。
traceSpecification	com.ibm.ws.logging.trace.specification	トレース・ストリングを使用して、選択的にトレースを有効にします。デフォルトは *=info=enabled です。
traceFormat	com.ibm.ws.logging.trace.format	トレース・ログのフォーマットを制御します。Liberty プロファイル のデフォルトのフォーマットは ENHANCED です。フルプロファイルのように BASIC と ADVANCED のフォーマットも使用できます。



Libertyプロファイルでは、サーバー状況あるいはJVM状況のスナップショットを取得する機能が提供されています。取得方法は、serverコマンドのdumpアクションあるいはjavadumpアクションを使用します。

dumpアクションはサーバー稼働中でも停止中でも実行することができ、実行すると、そのサーバーのスナップショット情報を含む圧縮ファイルがLibertyプロファイルの各サーバーディレクトリ配下に作成されます。取得情報は、サーバー構成、ログ情報、デプロイ済みアプリケーションの詳細などです。サーバー稼働中にdumpアクションを実行した場合は、それに加えて、サーバー内の各 OSGi バンドルの状態、登録されたOSGiサービスの情報、スレッド情報、JVM、ヒープサイズ、オペレーティング・システム、ネットワーク状況などのランタイム環境設定などの詳細な情報が取得できます。このサーバーのスナップショット情報は、WDTのユーティリティー機能で取得することもできます。また、コマンド実行時にincludeオプションを使用することにより、圧縮ファイル内にスレッドダンプ（javacore）やヒープ・ダンプ、システムダンプも含めることができます。

Javadumpアクションはサーバー稼働中のみ実行することができ、実行すると、そのサーバーのスレッドダンプ（javacore）がLibertyプロファイルの各サーバーディレクトリ配下に作成されます。また、コマンド実行時にincludeオプションを使用することにより、同時にヒープ・ダンプやシステム・ダンプも作成されます。



V8.5.5では、実行に長時間かかったJDBC処理を監視するタイムド・オペレーションという新機能が提供されています。

タイムド・オペレーションを有効にすると、アプリケーションのJDBC処理のうち、実行に長時間かかったものを監視し、そのレポートを定期的にmessages.logに出力させることができます。

構成方法は、server.xmlに<timedOperatoin>エレメントを記述し、定期的なログ出力の有効無効や出力の時間間隔を指定します。

また、ログに出力させるだけでなく、serverコマンドのdumpアクションを実行することにより、タイムド・オペレーションで監視した全ての操作について、タイプでグループ化して各グループ内で各操作の所要時間でソートした完全なレポートを出力させることができます。

通常のmessages.logに出力させた情報を確認し、問題がありそうであればserverコマンドのdumpアクションでレポートを確認し、更に詳細な情報が必要であればトレースを出力させるといった対応により、問題判別を行います。

Fix適用

IBM

V8.5.5のFixPack1は
2013年10月28日リリース予定

IIMを使用してLibertyプロファイルを導入した場合

IIMを使用してFixPackおよびiFixを適用

自己展開圧縮ファイルを使用してLibertyプロファイルを導入した場合

【FixPack】新しいランタイム環境として提供されるFixPack圧縮ファイル（Jar）を実行

1. 新しいロケーションにFixPack圧縮ファイル（Jar）を適用
   - ・既存ランタイム環境のサーバー停止は不要
   - ・同じロケーションの場合はサーバー停止および既存ランタイム環境のバックアップが必要
2. ユーザー・データとサーバー構成をマイグレーション
   - ・環境変数を設定している場合は新しいランタイム環境も既存と同じロケーションを使用
     →マイグレーションは不要
   - ・環境変数を設定していない場合は新しいランタイム環境のロケーションを使用
     →マイグレーションは必要
3. 新しいサーバーを起動

server.envでの環境変数の設定

```
WLP_USER_DIR=C:¥wlp¥user
WLP_OUTPUT_DIR=C:¥wlp¥output
```

【iFix】修正モジュールとして提供されるiFix圧縮ファイル（Jar）を実行

詳細な手順は次ページ参照

LibertyプロファイルのFixPack適用は、フルプロファイルとは大きく異なります。

IIMを使用してLibertyプロファイルを導入した場合は、同様にIIMを使用してFixPackやiFixを適用するという、フルプロファイルと同様の手順になります。

自己展開圧縮ファイルを使用してLibertyプロファイルを導入した場合は、修正モジュールだけを含んだFixPackではなく、FixPackが適用された状態のLibertyプロファイルのランタイム環境としてJar形式の圧縮ファイルが提供されるため、それを実行し、ランタイム環境ごと入れ替えることになります。その際、ユーザー・データとサーバー構成は手動でマイグレーションを行う必要があります。それらのディレクトリをserver.envで環境変数としてLibertyプロファイルのランタイムとは異なる別のロケーションとして指定している場合は、マイグレーションを行う必要はなく、ランタイムを置き換えた後もそのロケーションを参照するように構成します。

iFixに関しては、修正モジュールだけを含んだJar形式の圧縮ファイルが提供されるため、それを実行することで修正内容だけを適用するという手順になります。

【参考】iFix適用方法

iFix適用方法

1. サーバー停止「server stop <*servername*>」
2. Jarファイルで提供されるiFixモジュール展開（Libertyプロファイルのインストール先指定）
3. 適用後は「--clean」オプションをつけてサーバー起動「server start <*servername*> --clean」
   （※）「--clean」オプションによりworkareaディレクトリ（<*WLP*>/usr/servers/<*servername*>/workarea）にある
   OSGiキャッシュやplatformキャッシュを削除（再作成）

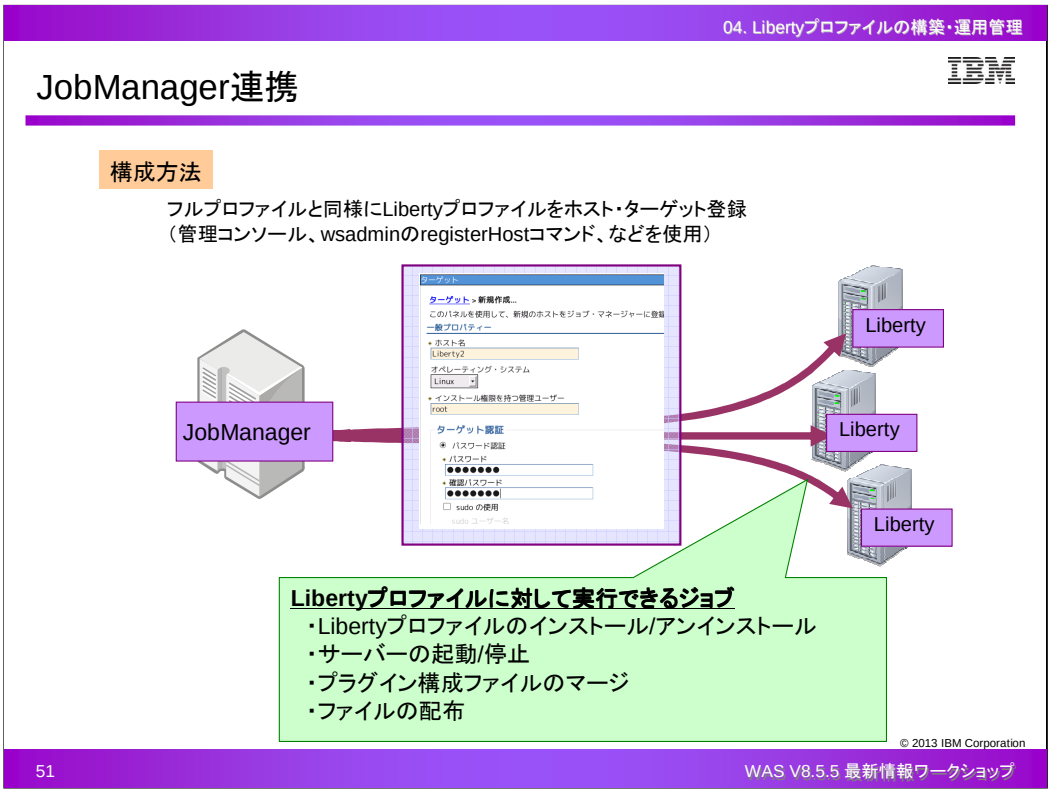
iFix適用後のサーバー起動時のメッセージ出力 （iFixに関連するフィーチャーを有効にしている場合のみ出力）
「[監査] CWWKF0014W: サーバーには次のテスト修正がインストールされています。PMxxxxx。」

fixesディレクトリにxmlファイル自動作成 （fix適用の履歴）
<*WLP*>/lib/fixes配下

```
2012/08/02  13:39    <DIR>          .
2012/08/02  13:39    <DIR>          ..
2012/08/02  13:39             1,508 8.5.0.0-WS-WAS_WLPArchive-TFPM69790_8.5.0.20120731_1104.xml
```

iFix削除方法

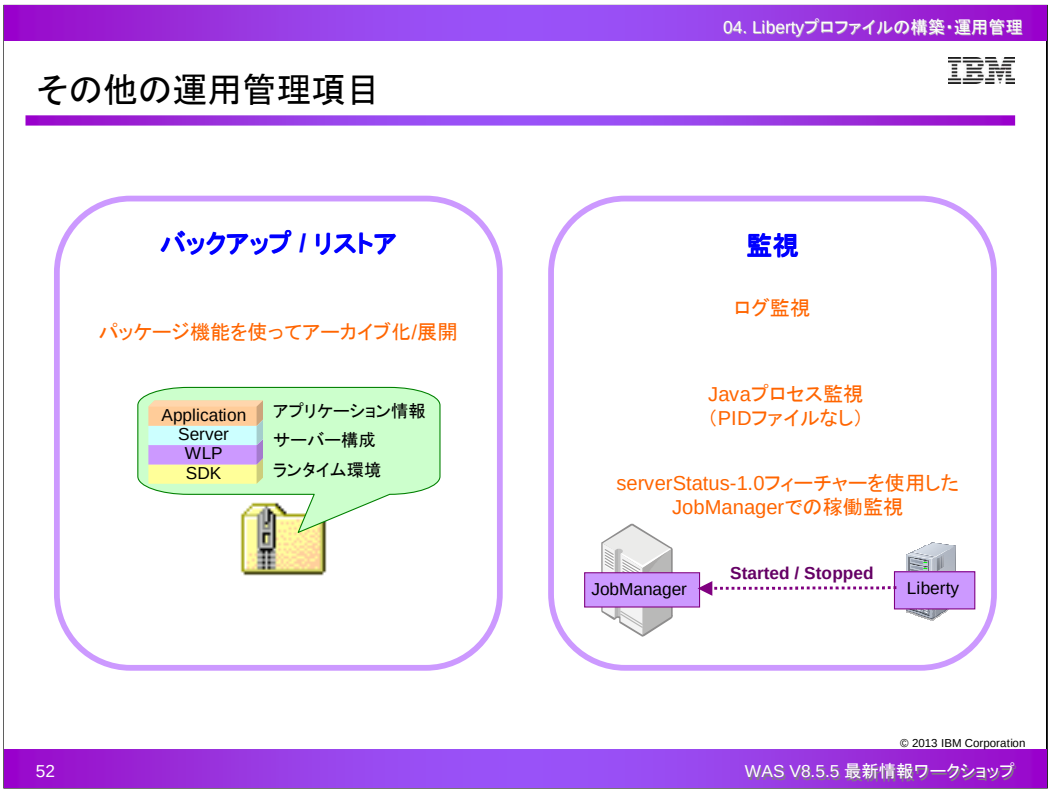
1. サーバー停止「server stop <*servername*>」
2. 以下のファイル（例）を手動で削除
   ・<*WLP*>/lib/com.ibm.ws.jndi_1.0.0.20120731-1104.jar （iFix適用によって追加されたファイル）
   ・<*WLP*>/lib/fixes/8.5.0.0-WS-WAS_WLPArchive-TFPM69790_8.5.0.20120731_1104.xml
3. 削除後は「--clean」オプションをつけてサーバー起動「server start <*servername*> --clean」



JobManagerを使用すると、フルプロファイルのNDセルやExpress/Baseエディションのサーバーを統合管理するのと同様に、複数のLibertyプロファイルも統合的に集中管理することができます。

LibertyプロファイルをJobManagerと連携させるには、フルプロファイルの場合と同様で、Libertyプロファイルのサーバーを管理コンソールやwsadminのregisterHostコマンドなどを使用してJobManagerのホスト・ターゲットとして登録します。

JobManagerからLibertyプロファイルに対して実行できるジョブは種類が限られており、Libertyプロファイルのインストール/アンインストール、サーバーの起動/停止、プラグイン構成ファイルのマージ、ファイルの配布があります。



その他にも、本番環境ではさまざまな運用管理項目がありますが、Libertyプロファイルにおける対応については、以下の通りです。

バックアップ/リストアに関しては、Libertyプロファイルで提供しているPackage機能を使用してアーカイブ化することで、ランタイム環境からアプリケーション情報まで全ての情報を容易にバックアップ取得することができ、zip展開することで容易にリストアすることができます。

監視に関しては、フルプロファイルで一般的に行われているのと同様に、Javaプロセス監視を行うことができます。Libertyプロファイルではpidファイルは存在しないため、pidファイルの監視を行うことはできません。また、serverStatus-1.0フィーチャーを使用すると、Libertyプロファイルのサーバーは JobManagerに対して自動的にランタイム状況を通知することができるようになりますので、それで稼働状況を監視することもできます。


04. Libertyプロファイルの構築・運用管理
IBM
まとめ
© 2013 IBM Corporation
53
WAS V8.5.5 最新情報ワークショップ

まとめ（1/2） フルプロファイル vs Libertyプロファイル

比較項目	フルプロファイル	Libertyプロファイル
Java EEサポート	◎ Full Profileをサポート	○ Web Profile＋その他主要仕様のみ
管理コンソール・ツール	◎ 管理コンソールが付属	△ JMXクライアント作成が必要
軽量・コンパクト	△	◎
スケーラビリティー	△ セルのサーバー数には一定の限界あり	◎ サーバー数とリソース負荷はほぼ比例
管理コンポーネントHA	△ DMのHA構成は共用ディスク・ODR必要	○ Replica SetはControllerのHA構成
環境構築スピード	△ インストール・構成に長時間を要する	○インストール・構成は短時間
構成変更容易性	△ 再起動が必要	◎ 動的変更が可能、環境のコピーも容易
HAマネージャー	○ シングルトン・フェイルオーバーが可能	×
静的クラスター	○	○
動的クラスター	○ ODRと組合せて利用可能	×
（状況）利用実績	◎ 実績豊富	△ 実績少ない
（状況）既存スキルの活用	◎	△ 従来WASと異なるため学習が必要

（注意）上記比較はWAS V8.5.5 NDエディションの機能に基づきます

まとめ（2/2）

IBM

Libertyプロファイルの構築・運用管理はシンプル

軽量、シンプルな構成、容易な構成管理、動的アップデート、容易な移行

V8.5.5の新機能「Liberty Collective」の登場

複数のLibertyサーバーの一元管理が可能

フルプロファイルとは異なるアーキテクチャーで高いスケーラビリティー

WASを使用するシステム構成の選択肢が広がった

開発段階では積極的にLibertyプロファイルの使用を検討するのに対して
本番環境への適用にあたってはLibertyプロファイルの特徴を理解した上で検討

＜Libertyプロファイルを本番環境で利用するのに有効なケース＞

- これまでTomcatで本番運用していた場合
- サーバーリソースが少ない場合
- 動的アップデートや短時間サーバー起動の要件がある場合
- ベンダーによる長期サポートが求められる場合
- アプリケーション・サーバー数の多い大規模なWASシステムの場合

などなど

IBM

END