実際にLibertyプロファイルを利用する場合、特に本番環境で利用する場合は、性能や可用性といった非機能要件を満たすためにLibertyサーバーを冗長化することが検討されます。

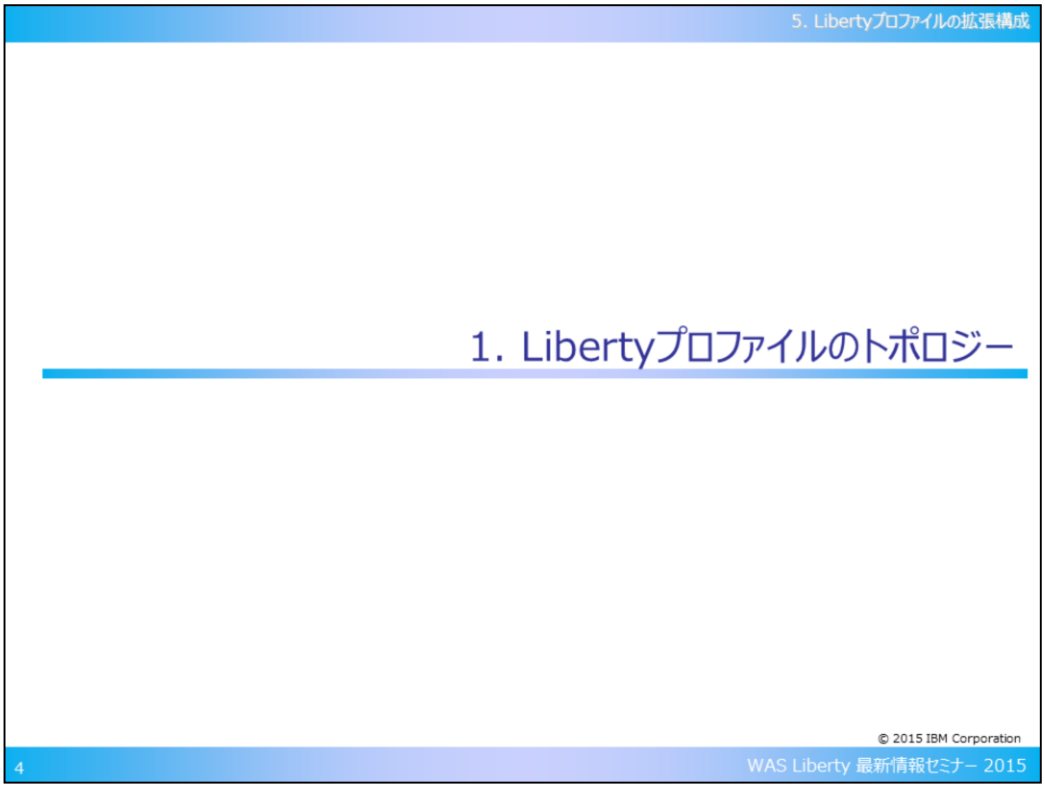
当セッションでは、Libertyプロファイルを冗長化するためにどのようなトポロジー構成があるのかご紹介します。また、それぞれのトポロジーにおける構成方法や運用管理方法、考慮事項についてご説明します。

Disclaimer

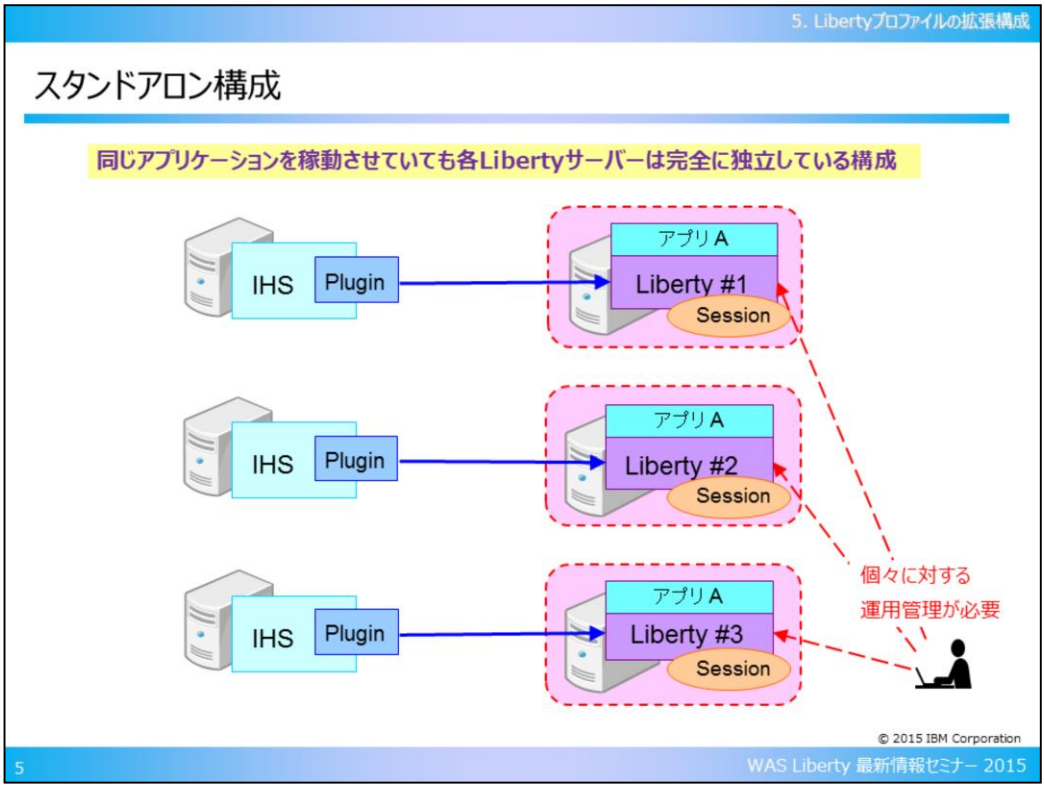
- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2015年7月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- ＩＢＭ、ＩＢＭロゴおよびibm.comは、世界の多くの国で登録されたInternational Business 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

- 1. Libertyプロファイルのトポロジー
- 2. シンプル・クラスター構成
- 3. Liberty Collective と Libertyクラスター構成
- 4. まとめ



1章では、Libertyプロファイルで構成することのできるトポロジーをシンプルなものから順にご紹介します。

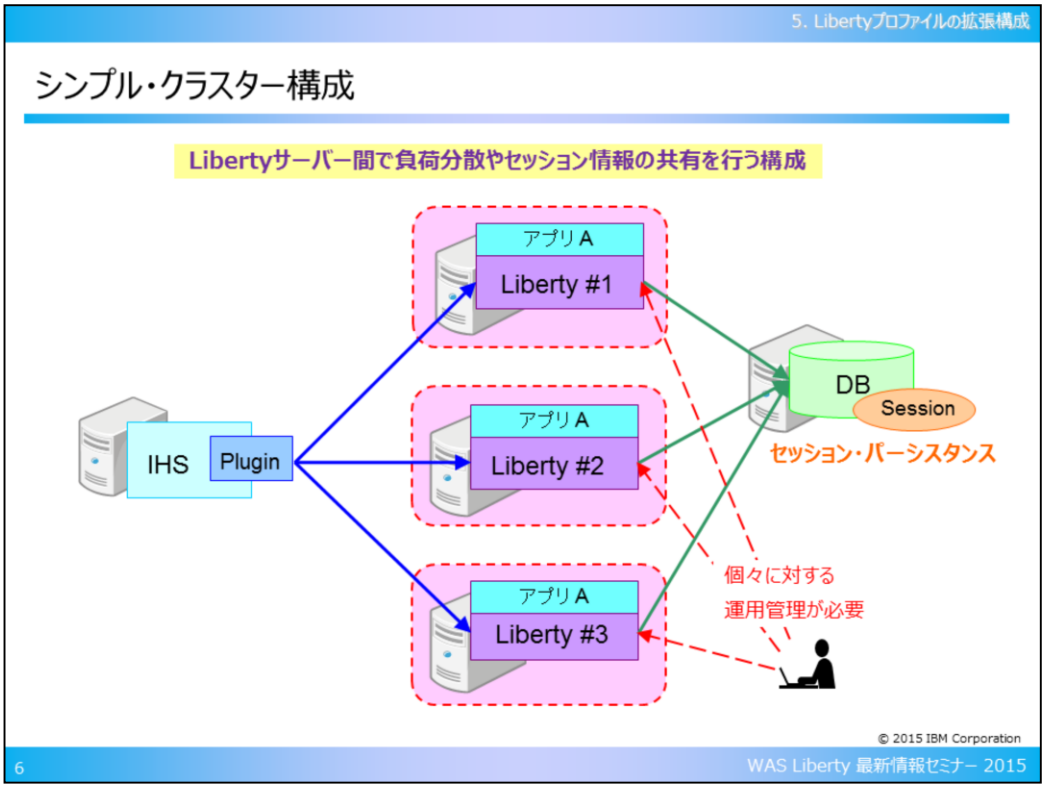


スタンドアロン構成とは最もシンプルなトポロジーで、同じアプリケーションを稼動させていても各Libertyサーバーは完全に独立している構成です。

IHSとLibertyサーバーは1:1の構成で、IHSからLibertyサーバーにリクエスト転送する際に負荷分散やLibertyサーバー障害時のフェールオーバーは行われません。

セッション情報も各Libertyサーバー毎に保持していて共有していないため、Libertyサーバー障害時のセッション情報のフェールオーバーも行われません。

起動・停止・構成変更といった運用管理は各Libertyサーバーに接続して個々に実施する必要があります。

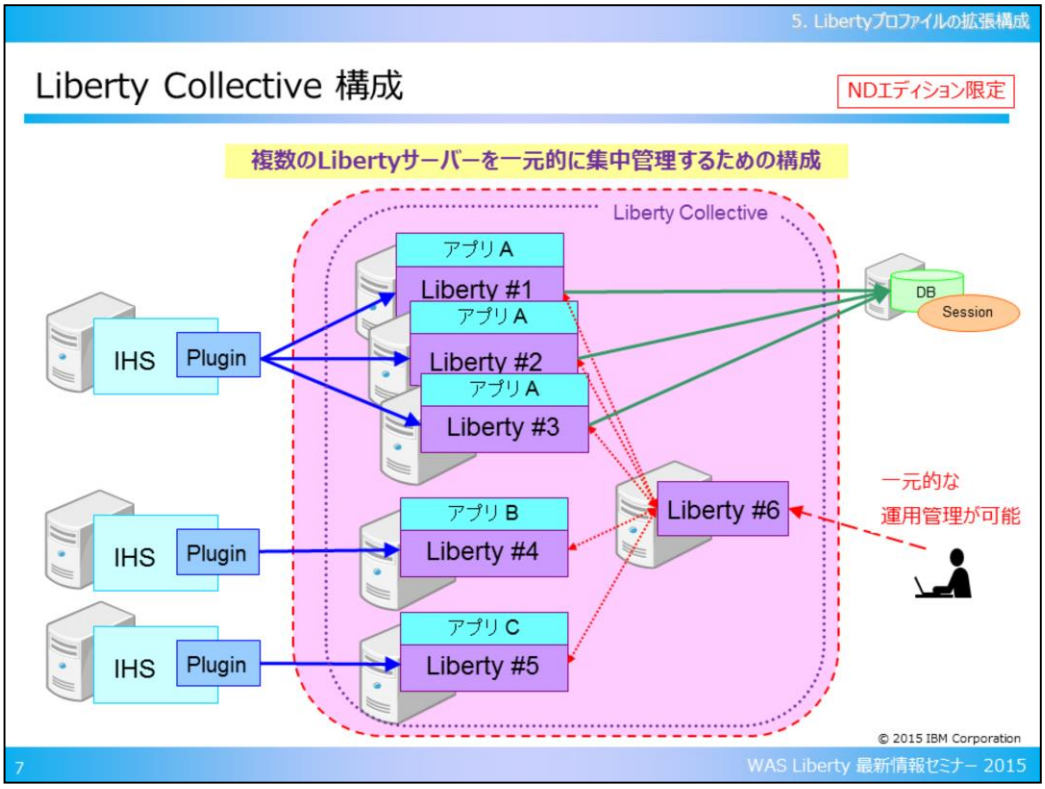


シンプル・クラスター構成とはスタンドアロン構成に耐障害性を持たせたトポロジーで、Libertyサーバー間で負荷分散やセッション情報の共有を行う構成です。

IHSとLibertyサーバーは1:N（あるいはN:N）の構成で、IHSからLibertyサーバーにリクエスト転送する際に負荷分散やLibertyサーバー障害時のフェールオーバーが行われます。

セッション情報はセッション・パーシスタンス機能によりLibertyサーバー同士で共有していないため、Libertyサーバー障害時のセッション情報のフェールオーバーも行われます。

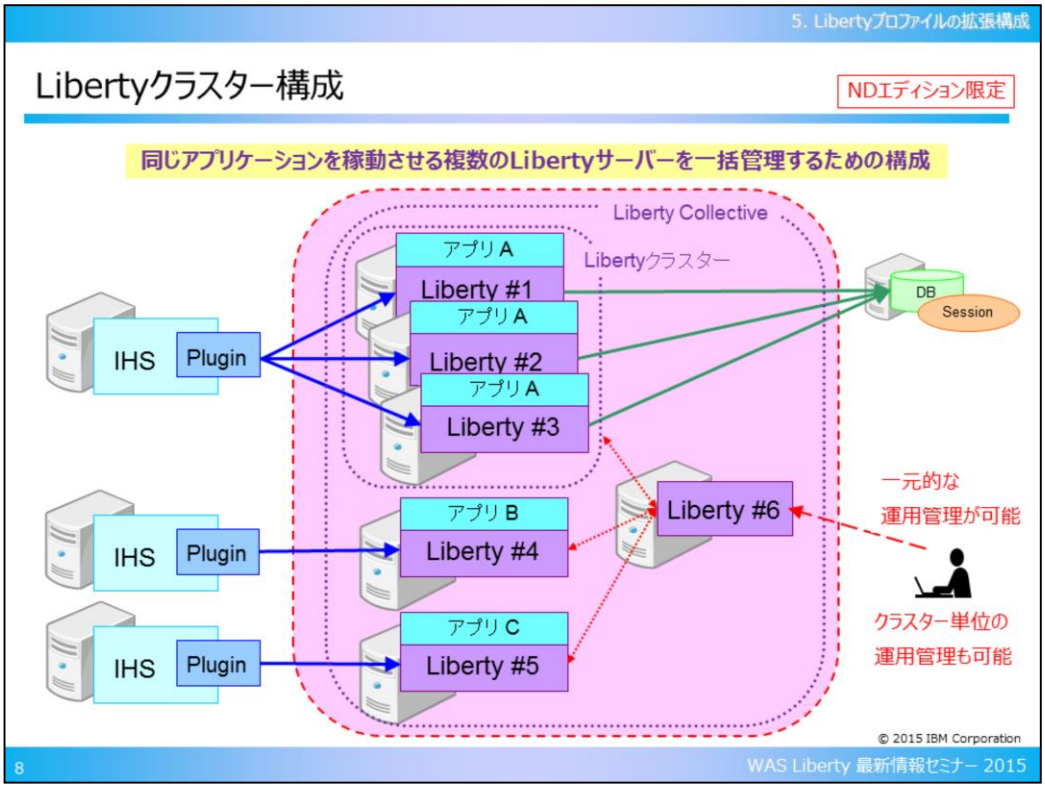
起動・停止・構成変更といった運用管理は各Libertyサーバーに接続して個々に実施する必要があります。



Liberty Collective 構成とはNDエディションのLibertyプロファイルが提供するLiberty Collective という機能を使用したトポロジーで、複数のLibertyサーバーを一元的に集中管理するための構成です。

Liberty Collective は、業務アプリケーションを稼動させるためのLibertyサーバーとそれらLibertyサーバーを管理するためのLibertyサーバーで構成されています。

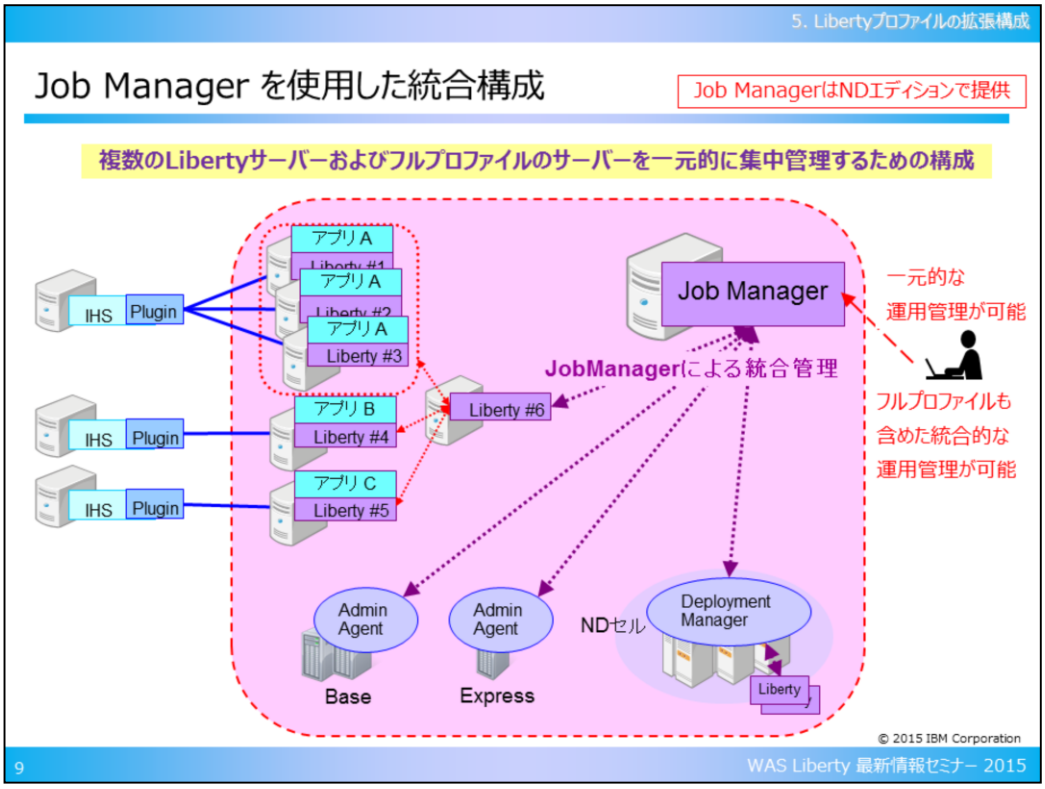
起動・停止・構成変更といった運用管理は、1つの管理用のLibertyサーバーに接続することにより、Liberty Collective 内に存在する複数のLibertyサーバーに対してまとめて実施することができます。



Libertyクラスター構成とはLiberty Collective 内で構成できるトポロジーで、同じアプリケーションを稼動させる複数のLibertyサーバーを一括管理するための構成です。

Liberty Collective 内に存在する複数のLibertyサーバーをLibertyクラスターと定義することでグループ化することができます。

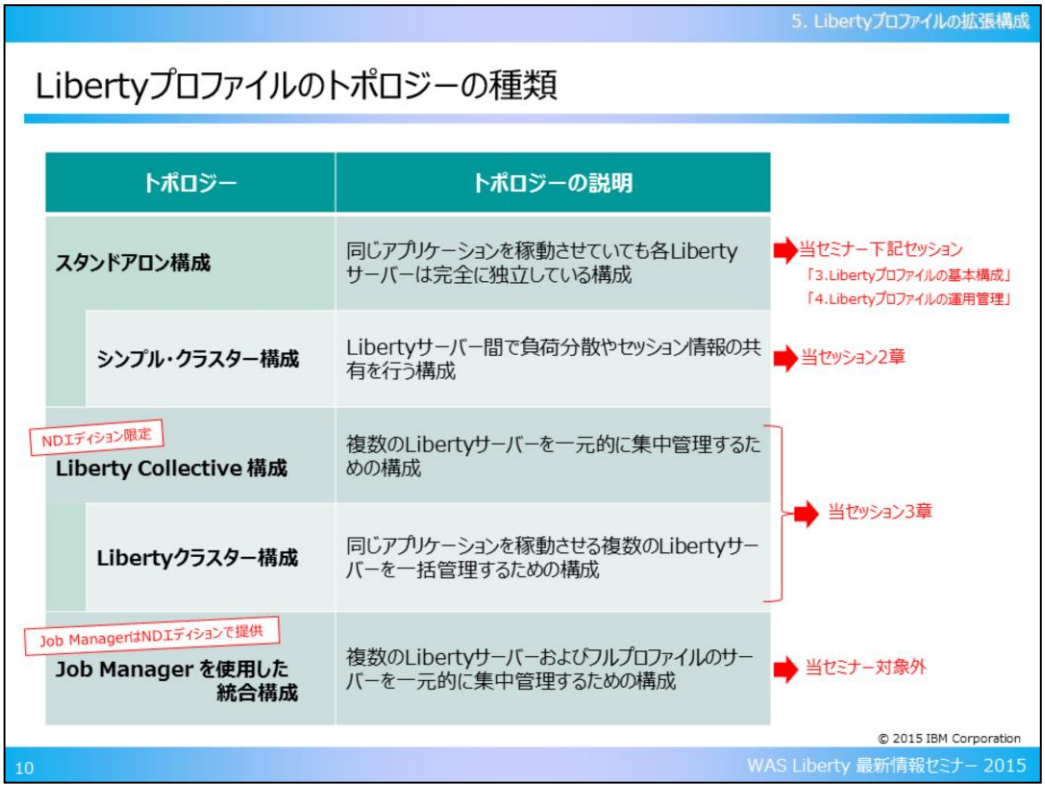
起動・停止・構成変更といった運用管理は、1つの管理用のLibertyサーバーに接続することにより、クラスター単位で（クラスター内の全メンバーに対して一括で）実施することができます。



Job Manager を使用した統合構成とはWASフルプロファイルで提供されているジョブスケジューラー機能を持ったJob Manager を使用したトポロジーで、複数のLibertyサーバーおよびフルプロファイルのサーバーを一元的に集中管理するための構成です。

Job Manager を使用すると、これまでご紹介してきたLibertyプロファイルのトポロジーに加えて、フルプロファイルのBase/Express/NDセルなども含めて統合化することができます。

起動・停止・構成変更といった運用管理は、Job Manager に接続することにより、全てのLibertyプロファイルおよびフルプロファイルのサーバーに対して実施することができます。



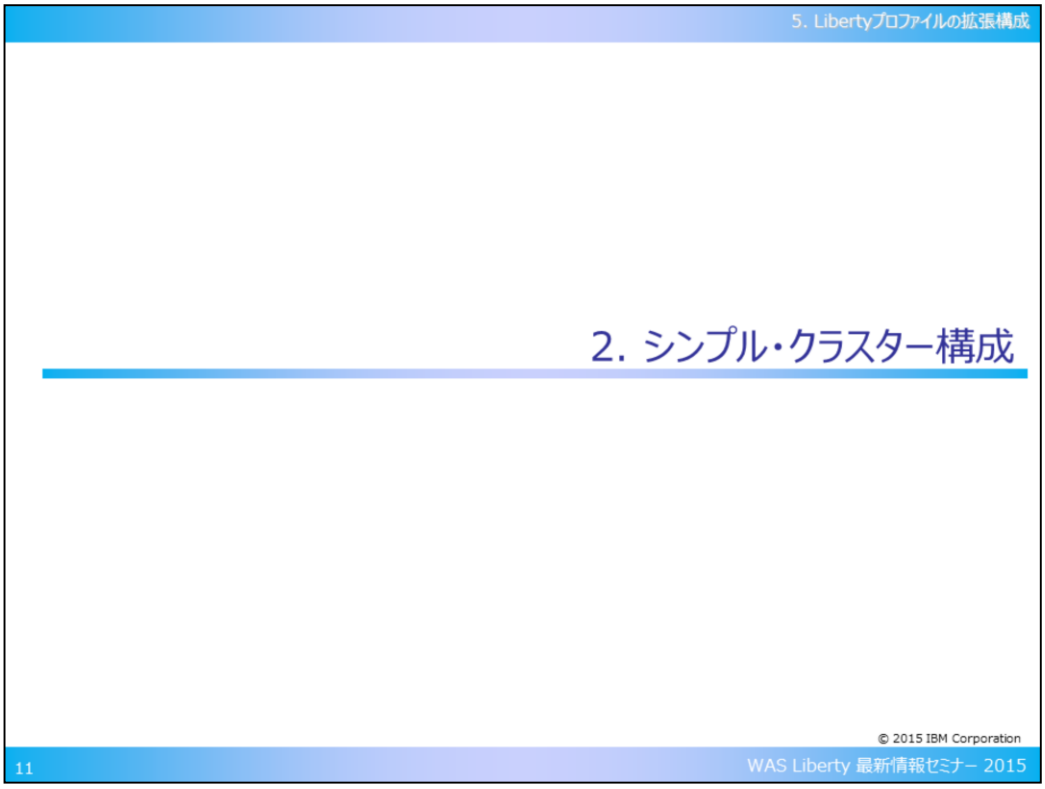
これまでご紹介してきたLibertyプロファイルのトポロジーをまとめた表です。

スタンドアロン構成に関する構成方法や運用管理に関しては、当セミナーの「3. Libertyプロファイルの基本構成」および「4. Libertyプロファイルの運用管理」のセッションにてご説明しています。

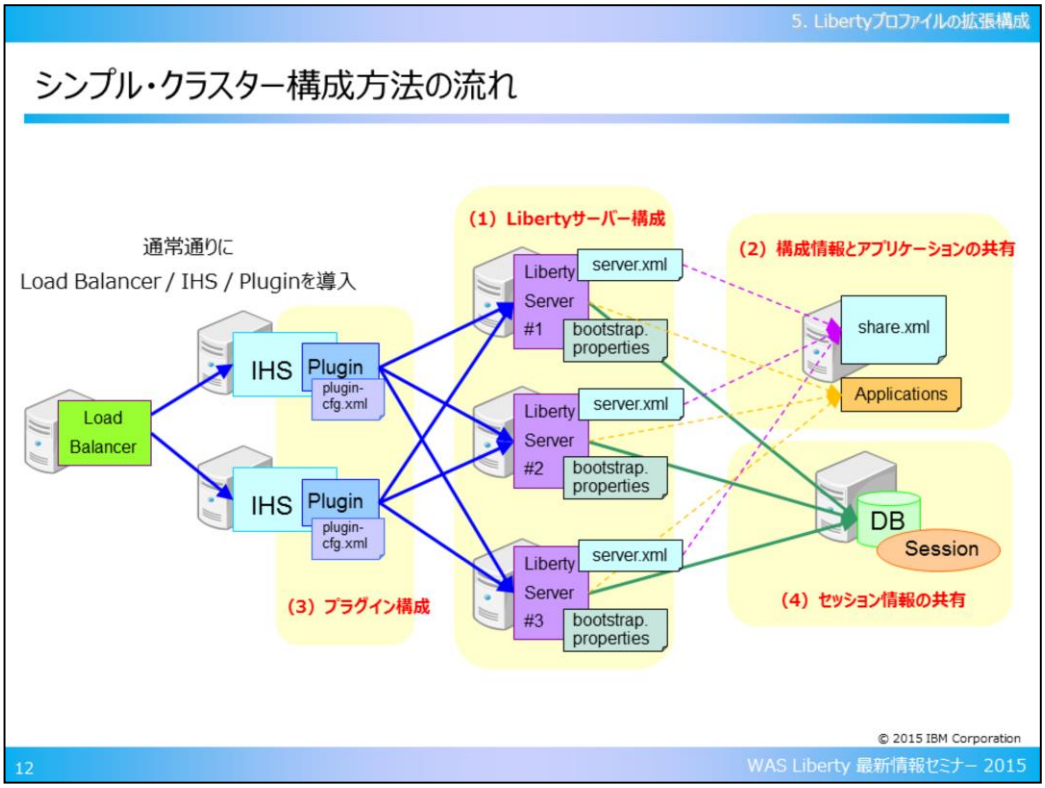
シンプル・クラスター構成に関する構成方法や運用管理に関しては、当セッションの2章でご説明します。

Liberty Collective 構成およびLibertyクラスター構成に関する構成方法や運用管理に関しては、当セッションの3章でご説明します。

Job Manager を使用した統合構成に関する構成方法や運用管理に関しては、フルプロファイルで提供されているJob Manager の機能に依存しますので、当セミナー対象外としています。



2章では、Libertyプロファイルのシンプル・クラスターに関する構成方法や運用管理についてご紹介します。



Libertyプロファイルでシンプル・クラスターを構成する場合の構成方法についてご説明します。

まず、フルプロファイルと同様、通常通りにLoad Balancer/IHS/PluginといったLibertyサーバー前段のコンポーネントを導入します。

以降の大まかな作業の流れは以下の通りです。

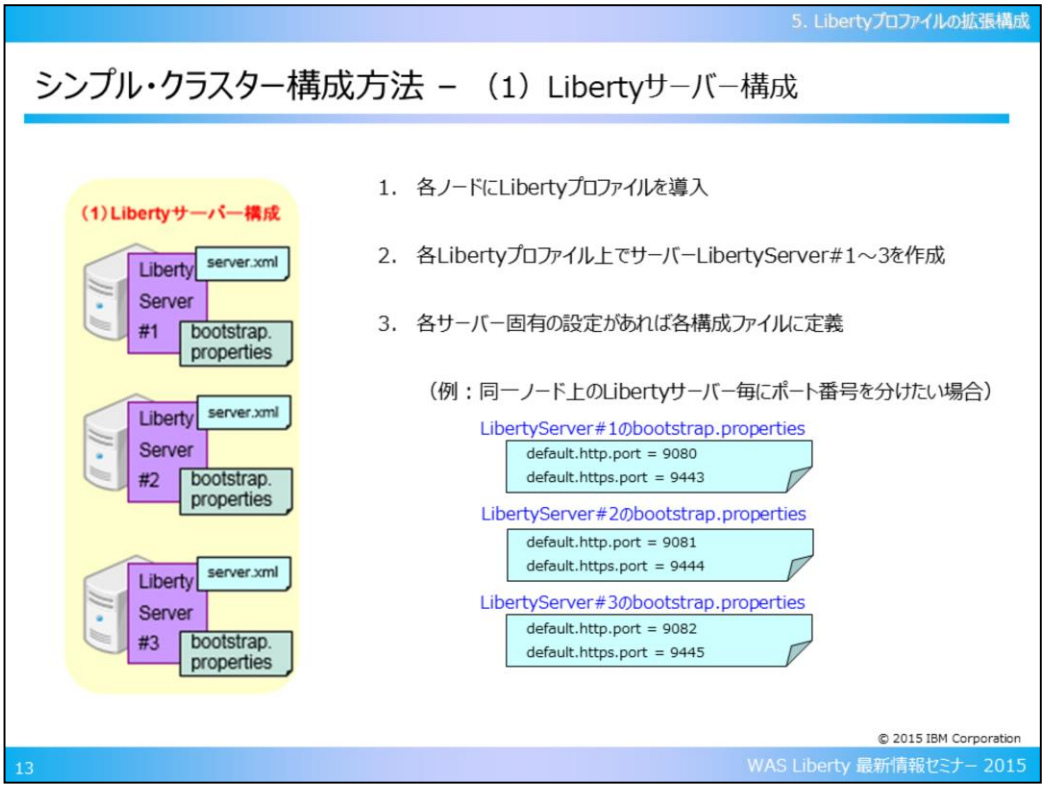
（1）Libertyサーバーを稼動させるノード上にLibertyプロファイルを導入してLibertyサーバーを作成し、サーバー固有の設定をします。

（2）構成変更時の管理対象を最小化するために、Libertyサーバー共通となる構成情報とアプリケーションを共有させるように構成します。

（3）各Libertyサーバーへの負荷分散やフェールオーバーが行えるように、プラグインを構成します。

（4）セッション情報を取り扱うアプリケーションの場合は、Libertyサーバー同士でセッション情報を共有するように構成します。

これらの各ステップに関する詳細な手順は次ページ以降でご説明します。



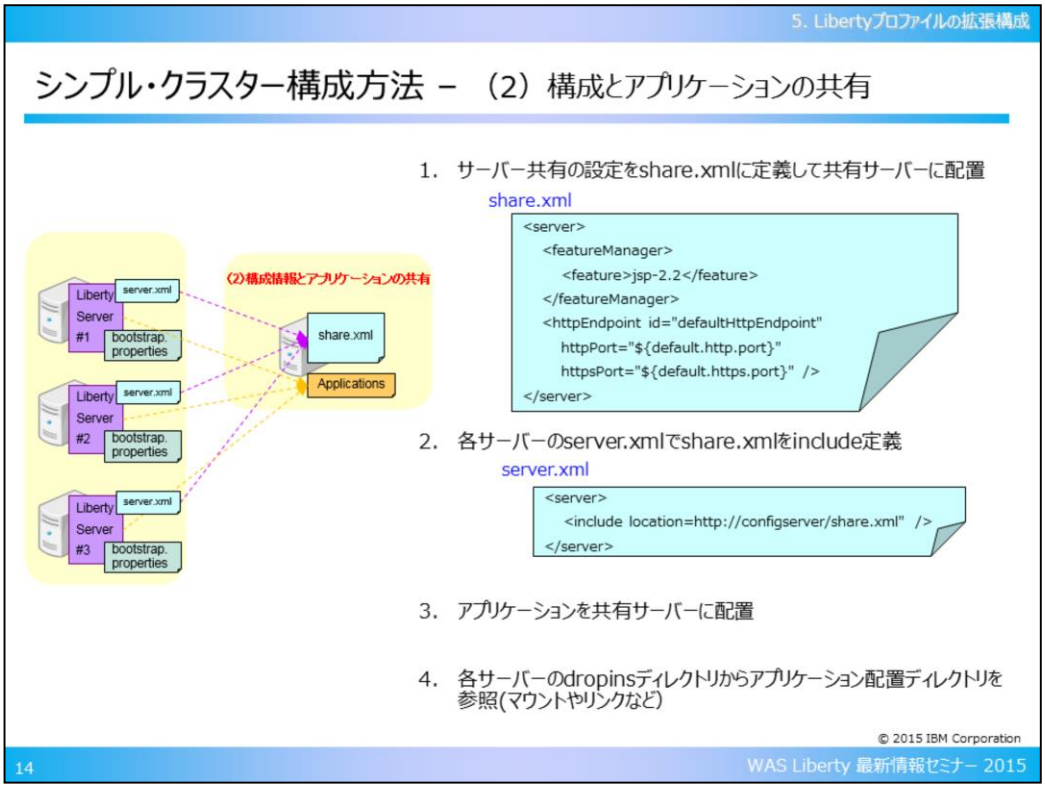
こちらは、Libertyサーバーを稼動させるノード上にLibertyプロファイルを導入してLibertyサーバーを作成し、サーバー固有の設定をする手順の例です。

1. シンプル・クラスターを構成する全てのノード上にLibertyプロファイルを導入します。

2. 各ノードのLibertyプロファイル上で、LibertyサーバーとしてLibertyServer#1～3を作成します。

3. シンプル・クラスター内のLibertyサーバーで共通的な設定は別の構成ファイルに定義するため、ここでは固有の設定があれば各Libertyサーバーの構成ファイルに定義します。ここでは、例えば同一ノード上に3つのLibertyサーバーを作成している場合、HTTPトランスポートおよびHTTPSトランスポート番号を分ける必要があるため、各Libertyサーバーのbootstrap.propertiesファイルにポート番号を定義している例を挙げています。

bootstrap.propertiesファイルは必須ではないため、作成しない限り存在しません。このファイルはサーバー・ディレクトリーに作成する必要があり、サーバー・ディレクトリーには、構成ルート・ファイル server.xml も含まれています。デフォルトでは、サーバー・ディレクトリーは usr/servers/server_name で、サーバー・ディレクトリーは変更できます。



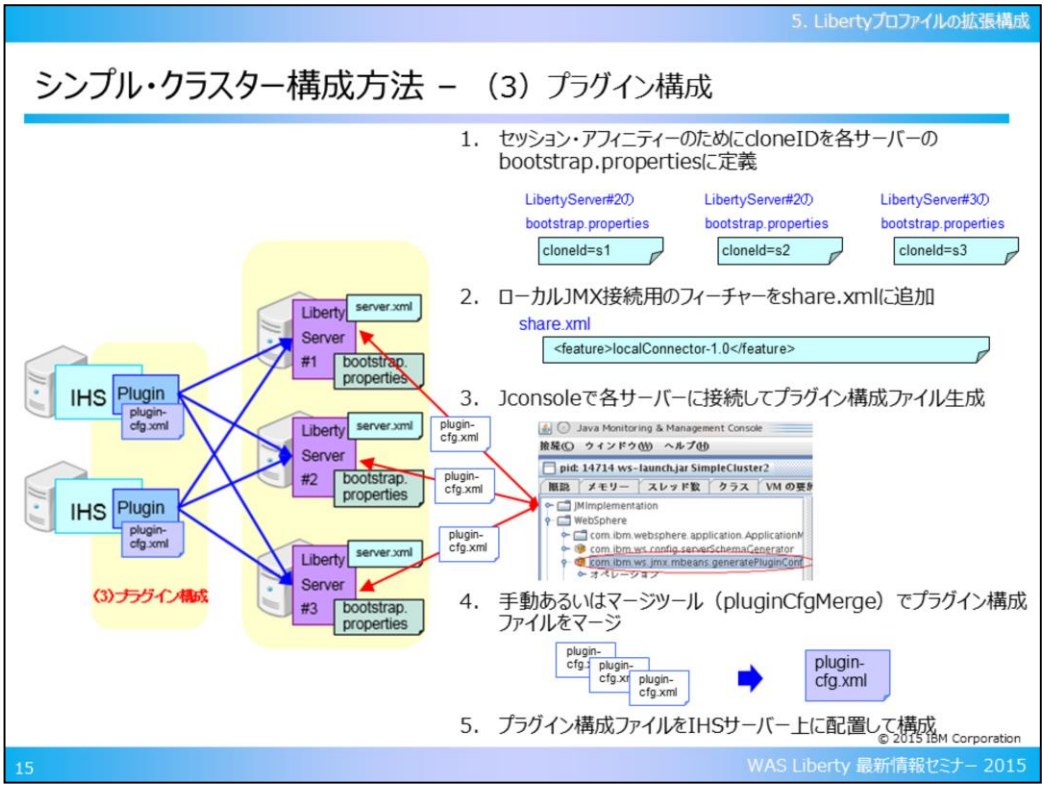
こちらは、構成変更時の管理対象を最小化するために、Libertyサーバー共通となる構成情報とアプリケーションを共有させるように構成する手順の例です。

1. シンプル・クラスター内のLibertyサーバーで共通となる設定をshare.xmlなどserver.xmlとは別の構成ファイルに定義し、ファイルサーバーやWebサーバーなど共有サーバーに配置します。

2. 各Libertyサーバーのserver.xml内にshare.xmlをincludeするように定義します。

3. アプリケーションも共有サーバーに配置します。

4. 各Libertyサーバーのdropinsディレクトリからアプリケーションを配置したディレクトリを参照させるようにOSのマウント等の機能で構成します。



こちらは、各Libertyサーバーへの負荷分散やフェールオーバーが行えるように、プラグインを構成する手順の例です。

1. セッション・アフィニティーの機能を持たせるために、各Libertyサーバーのbootstrap.propertiesファイルにcloneIDを定義します。

2. プラグイン構成ファイルを生成するために各LibertyサーバーにJMX接続を行う必要があるため、share.xmlにローカルJMX接続用のフィーチャー「localConnector-1.0」を追加します。

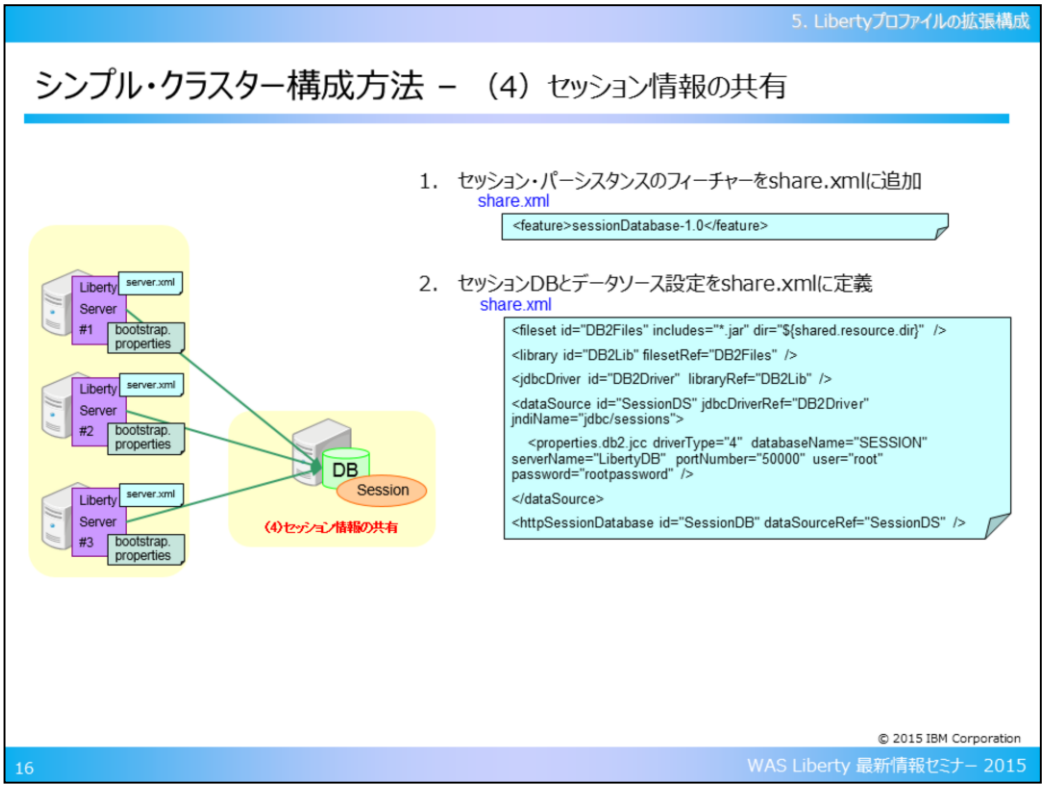
3. Jconsoleで各Libertyサーバーに接続し、プラグイン構成ファイルを生成するオペレーションを実行します。

4. 各Libertyサーバーで生成されたプラグイン構成ファイルを手動あるいはフルプロファイルで提供されているマージツールpluginCfgMergeを使用して1つにマージします。

5. マージされたプラグイン構成ファイルをIHSプラグインが読み込めるようにIHSサーバー上に配置します。

＜参考＞

http://www-01.ibm.com/support/docview.wss?uid=swg21674883

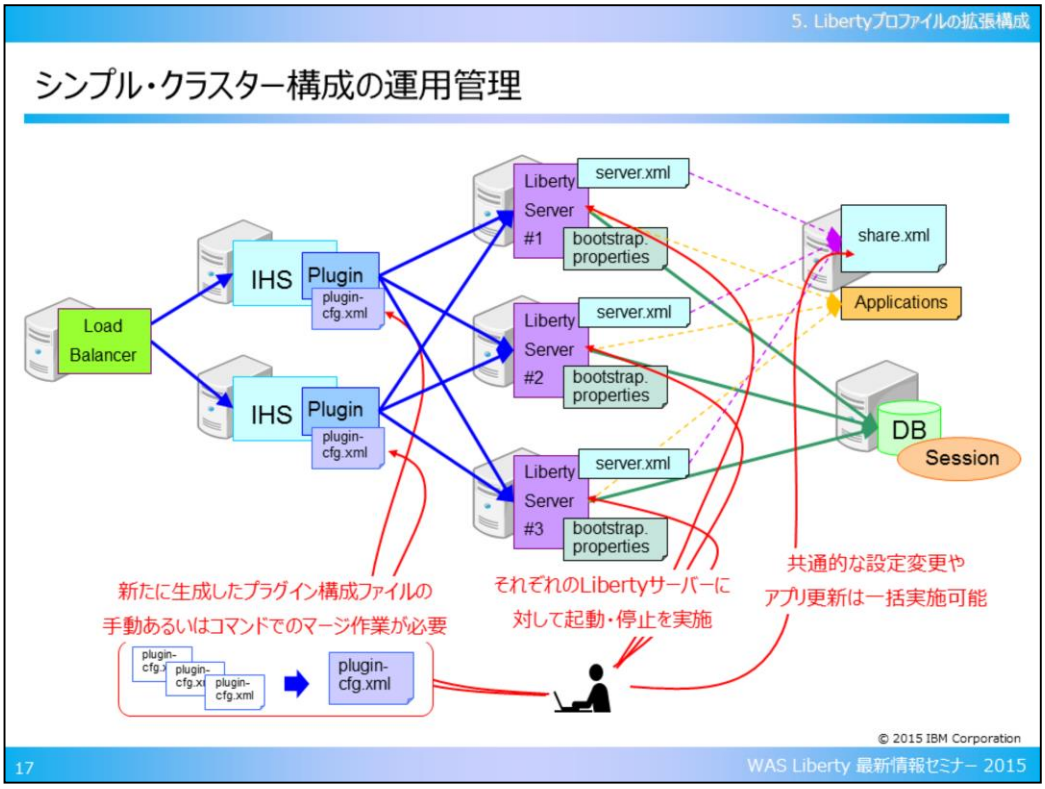


こちらは、セッション情報を取り扱うアプリケーションの場合は、Libertyサーバー同士でセッション情報を共有するように構成する手順の例です。

1. セッション情報をDBに保管するセッション・パーシスタンスを有効にするために、share.xmlファイルにフィーチャー「sessionDatabase-1.0」を追加します。

2. share.xmlファイルに、セッションDBおよびセッションDBに接続するためのデータソースを定義します。

Libertyプロファイルでは、セッション・パーシスタンスのセッション保管先として、DBの他にもWebSphere eXtreme Scale やIBM WebSphere DataPower Appliance XC10 V2 caching appliance が選択できます。

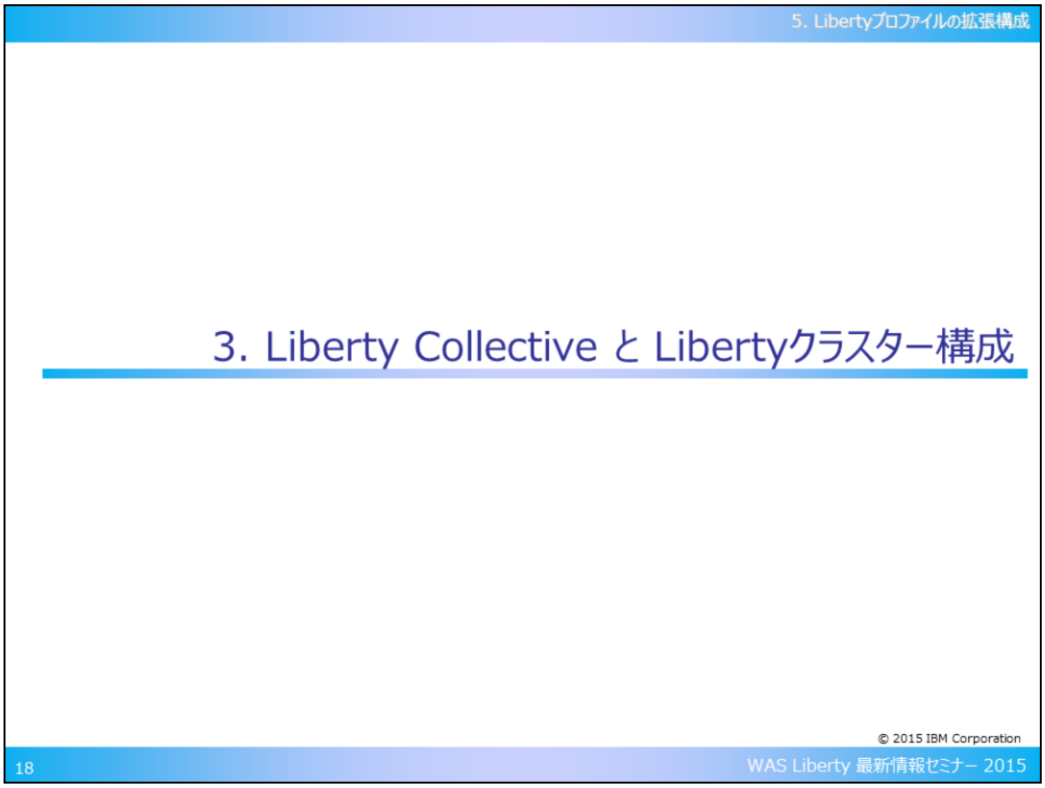


シンプル・クラスターを構成した場合の運用管理の特徴をご説明します。

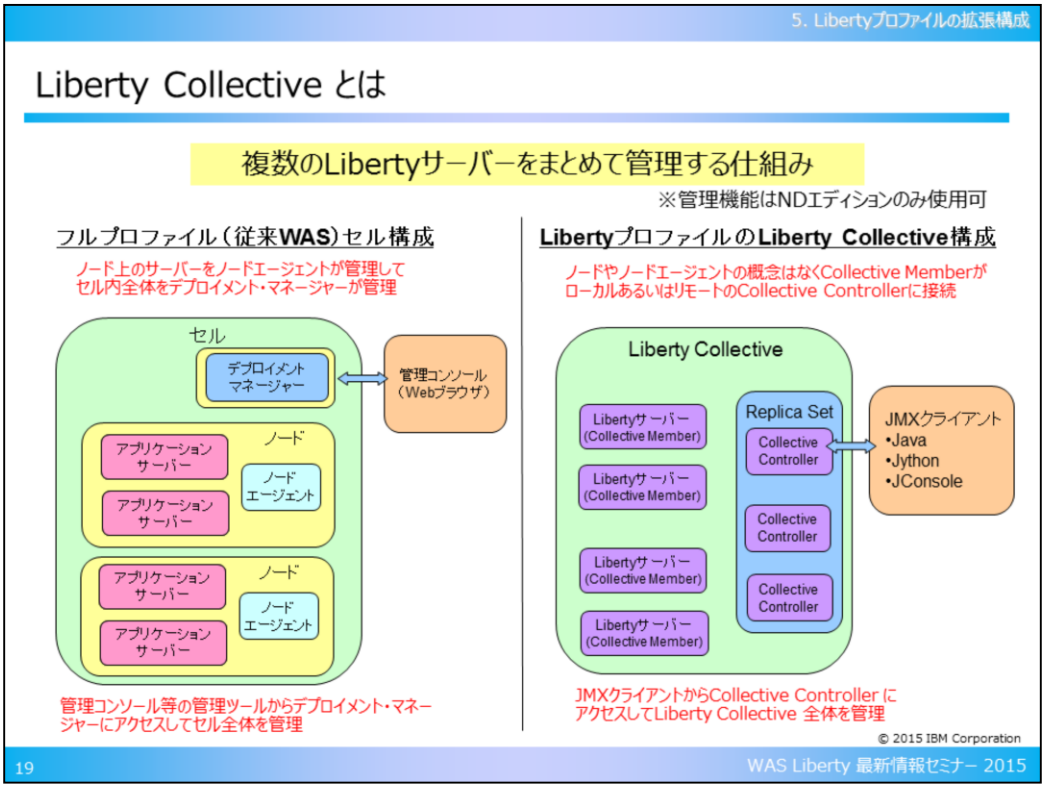
まず、共通的な設定やアプリケーション情報は1箇所に構成しましたので、構成変更やアプリケーション更新の際は、共有サーバー上の1箇所を更新することにより、全てのLibertyサーバーに情報を反映させることができます。

ただ、Libertyサーバーの起動・停止といったオペレーションは、Libertyサーバーそれぞれに対して実施する必要があります。

また、構成やアプリケーションの変更に伴ってプラグイン構成ファイルを新たに生成するたびに、手動あるいはフルプロファイルで提供されているコマンドを使用したマージ作業が必要になります。



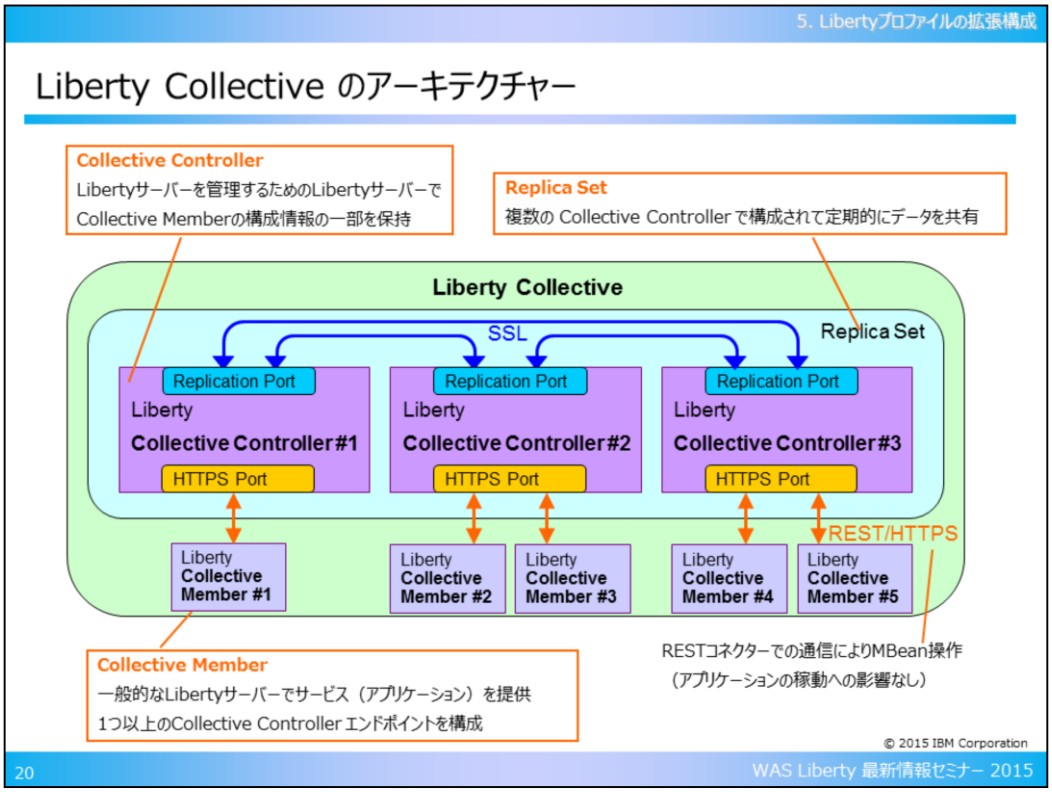
3章では、LibertyプロファイルのLiberty Collective およびLibertyクラスターに関する構成方法や運用管理についてご紹介します。



LibertyプロファイルではLiberty Collective という機能が提供されており、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。

フルプロファイルのセル構成では、ノード上に定義されている複数のサーバーはノードエージェントが管理し、セル内の全てのノードをデプロイメント・マネージャーが管理するというトポロジーです。管理者は管理コンソール等の管理ツールからデプロイメント・マネージャーに対してアクセスし、セル内を管理します。

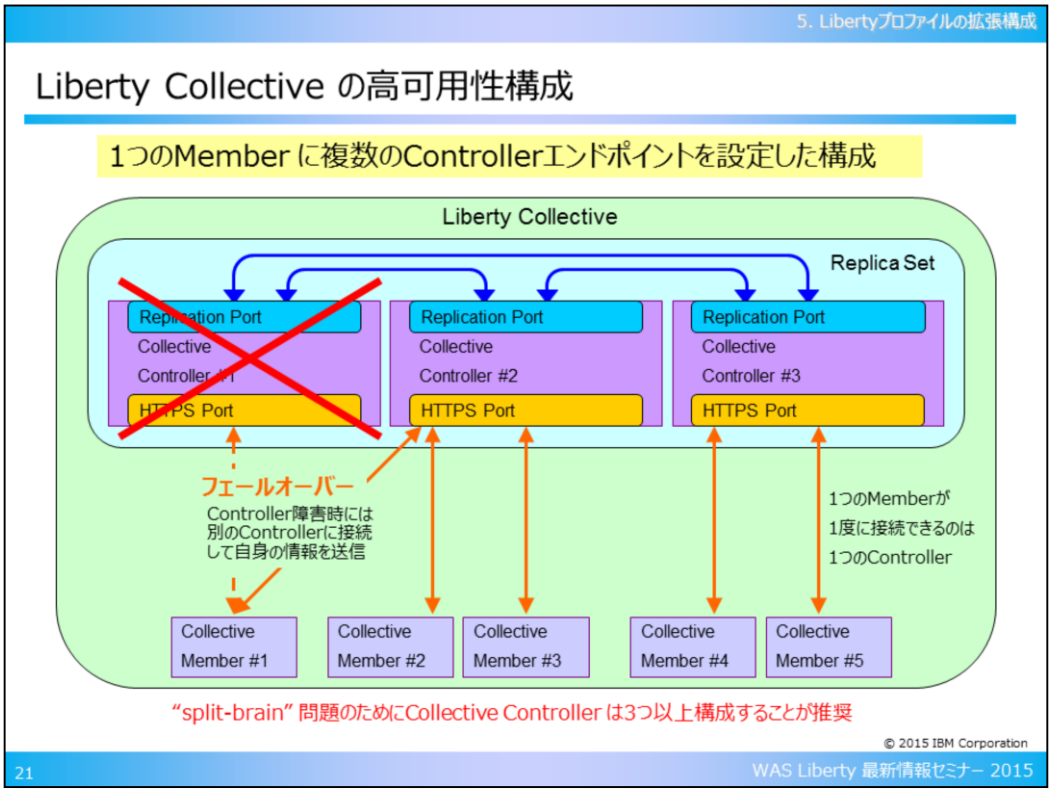
LibertyプロファイルのLiberty Collective 構成では、ノードやノードエージェントといった概念はなく、業務アプリケーションを処理するLibertyサーバーであるCollective Member と管理用のLibertyサーバーであるCollective Controller で構成されたトポロジーになります。Collective Member がローカルあるいはリモートのCollective Controller に接続することでCollective Controller がCollective Member を管理できるようになります。管理者はJMXクライアントからCollective Controller に対してアクセスし、Liberty Collective 内を管理します。



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。

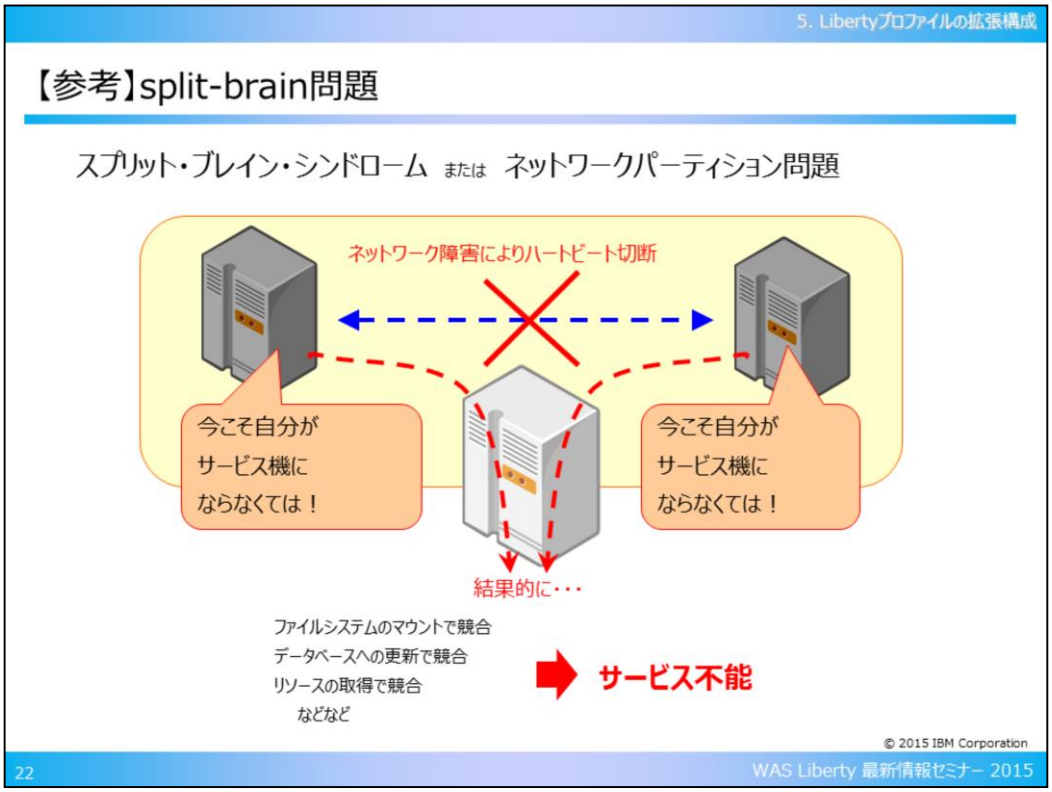
Liberty Collective は1つ以上のCollective Controller で構成されますが、複数のCollective Controller を配置することにより、可用性を持たせることができます。



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。

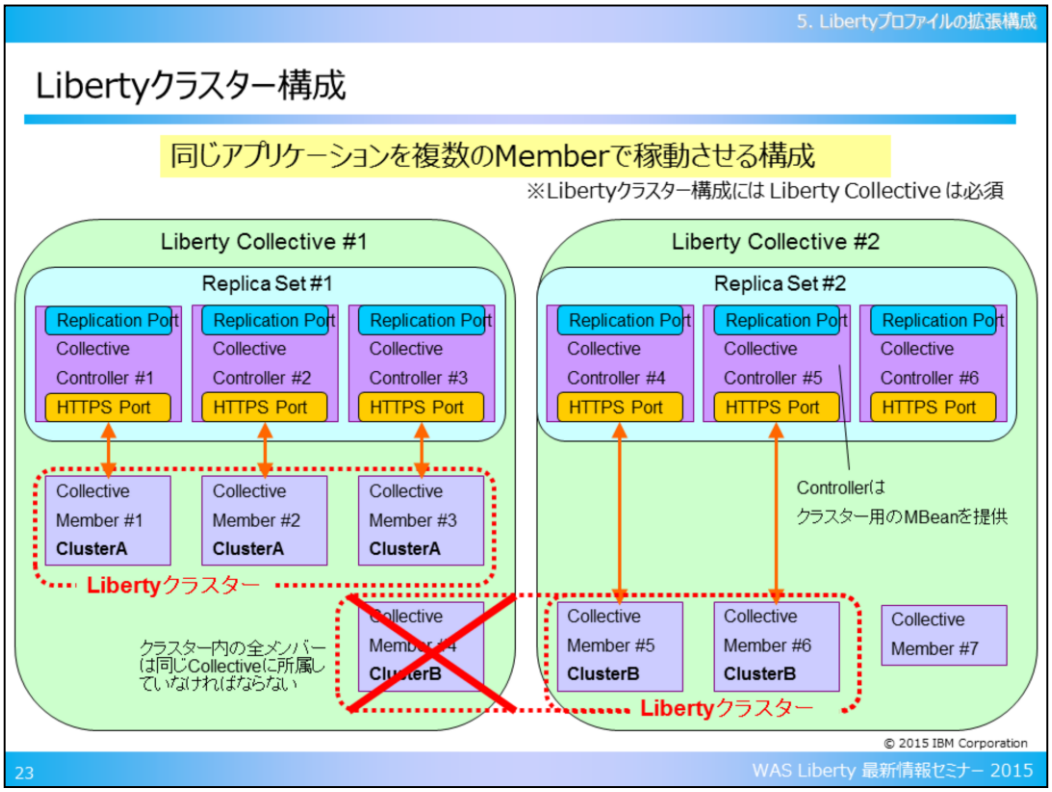
１つのLiberty Collective において、”split-brain”問題のためにCollective Controller は３つ以上構成することが推奨されています。



”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。

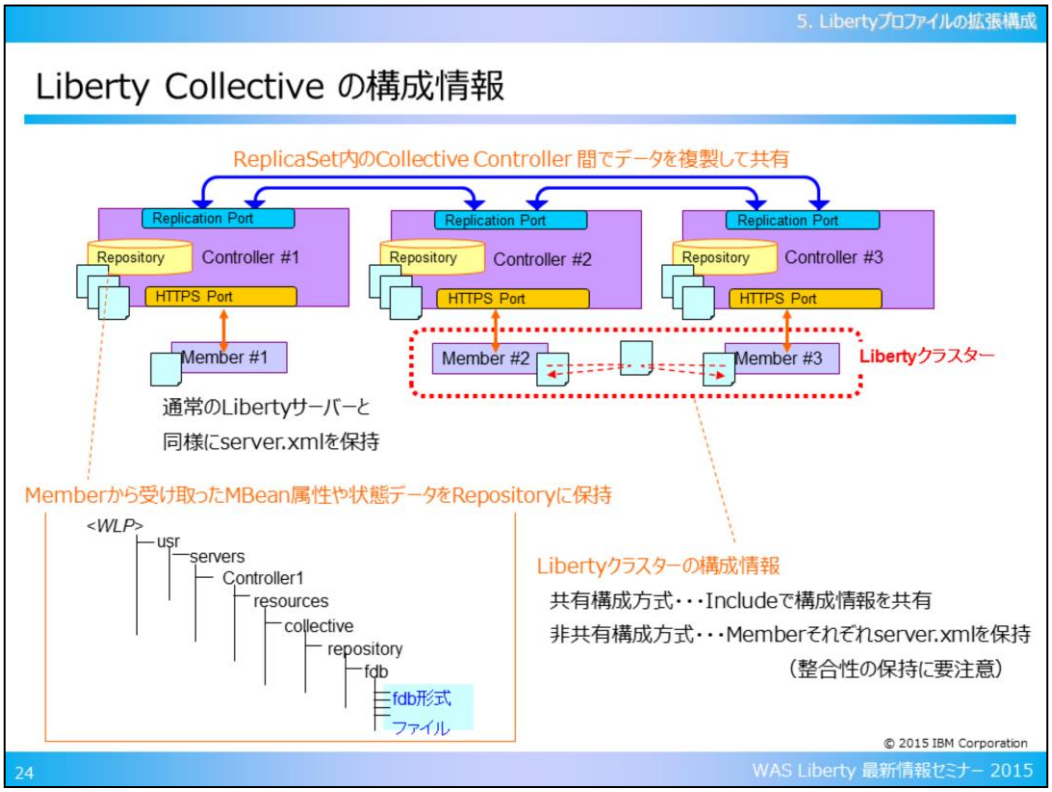
Liberty Collective においても、Collective Controller が2つで構成されている場合、定期的にReplication処理を行なっているところでネットワーク障害等が発生すると、それぞれのCollective Controller は、2つ構成されているうち自分1つだけがアクティブであると認識し、N/2+1という条件を満たせず、操作ができなくなってしまいます。



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。

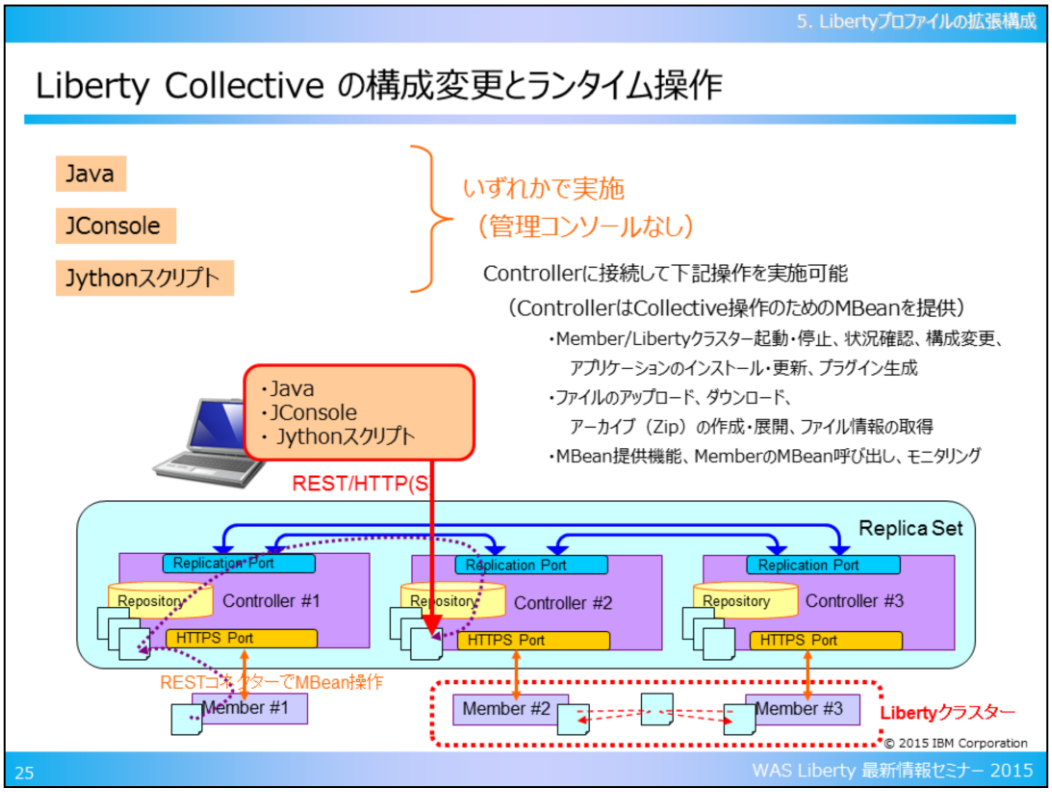
Libertyクラスターを構成すると、Collective Controller はクラスター用のMBeanを提供し、個々のLibertyサーバーに対する操作だけでなく、Libertyクラスターに属するLibertyサーバー全体に対する操作を行うことができるようになります。例えば、Libertyクラスター名の取得、Libertyクラスターの起動/停止や稼働状況の確認、といった操作を行うことができます。

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。



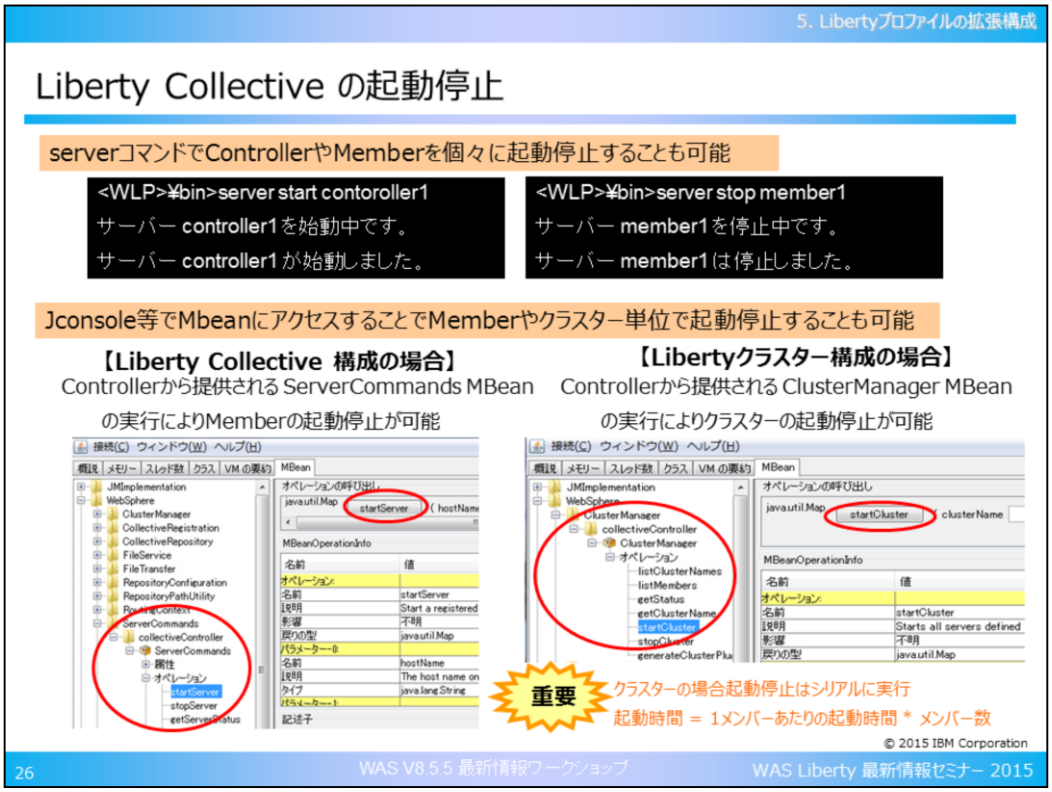
Liberty Collective 構成の場合、各Collective Member は通常のLibertyサーバーと同様に構成情報としてserver.xmlを保持しています。そして、Collective Controller は、Collective Memberから構成情報の一部やランタイム状況を受け取り、更にReplica Set のデータ複製によって受け取った情報も含めて、各Collective Controllerが保持するRepositoryに格納します。Repositoryの実態はファイル群であり、Collective Controller のサーバー情報が格納されているディレ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

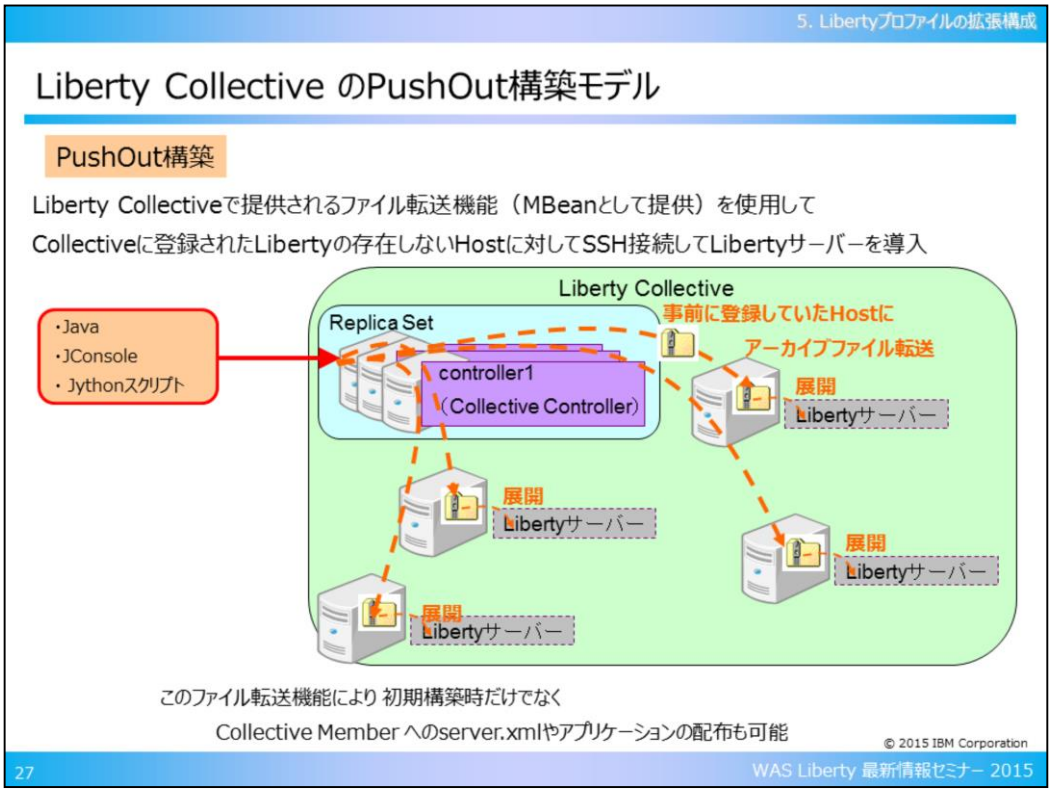
いずれの方法の場合においても、構成されているCollective Controller のうちいずれか1つに対して接続すれば、Collective Controller がCollective操作のためのMbeanを提供しているため、操作を行なうことができます。



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 Member 個々に対して同様に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るのに加えて、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・停止を行うこともできます。

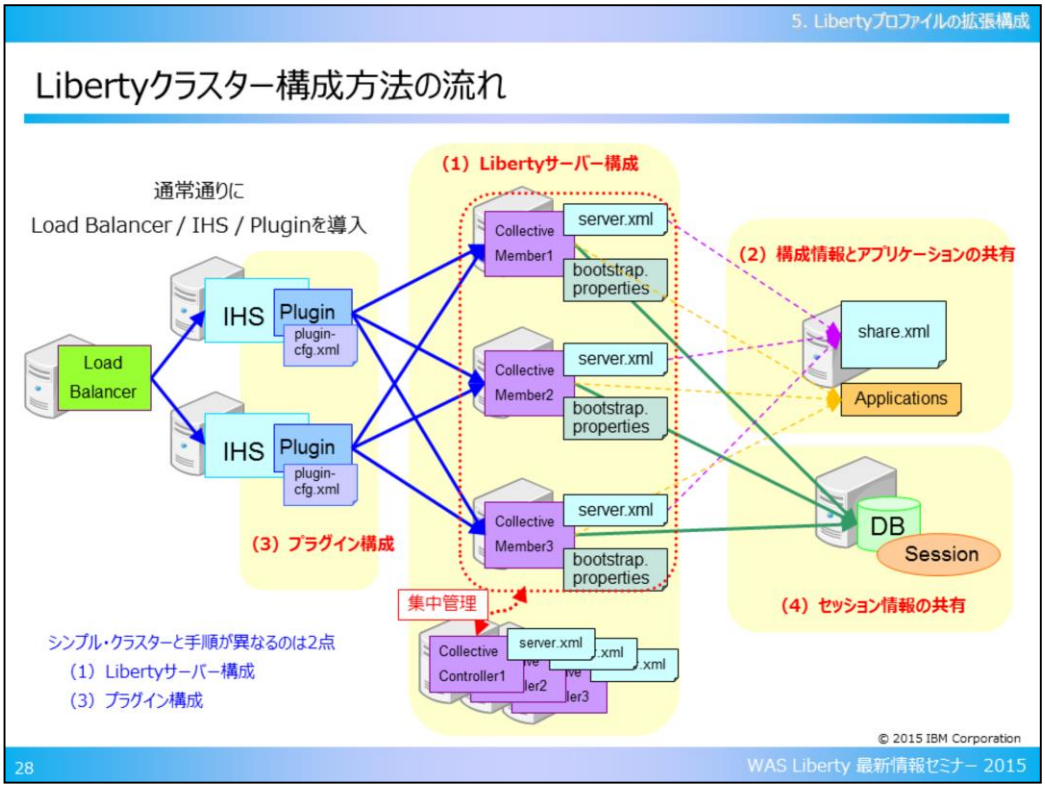
Libertyクラスター構成の場合、Collective Controller で提供されるClusterManager MBean のsartCluster・stopClusterのオペレーションを実行することで、Libertyクラスター全体の起動・停止を行います。ここで注意が必要なのは、Libertyクラスター構成の場合、起動・停止は各Memberシリアルに実行されるため、Libertyクラスター全体の起動・停止時間は、1メンバーあたりにかかる起動・停止時間とメンバー数の積になり、Member数が多ければそれだけ多くの時間がかかることになります。



Liberty 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等OSレベルの機能を使用して接続されます。



LibertyプロファイルでLibertyクラスターを構成する場合の構成方法についてご説明します。

まず、フルプロファイルと同様、通常通りにLoad Balancer/IHS/PluginといったLibertyサーバー前段のコンポーネントを導入します。

以降の大まかな作業の流れは以下の通りです。

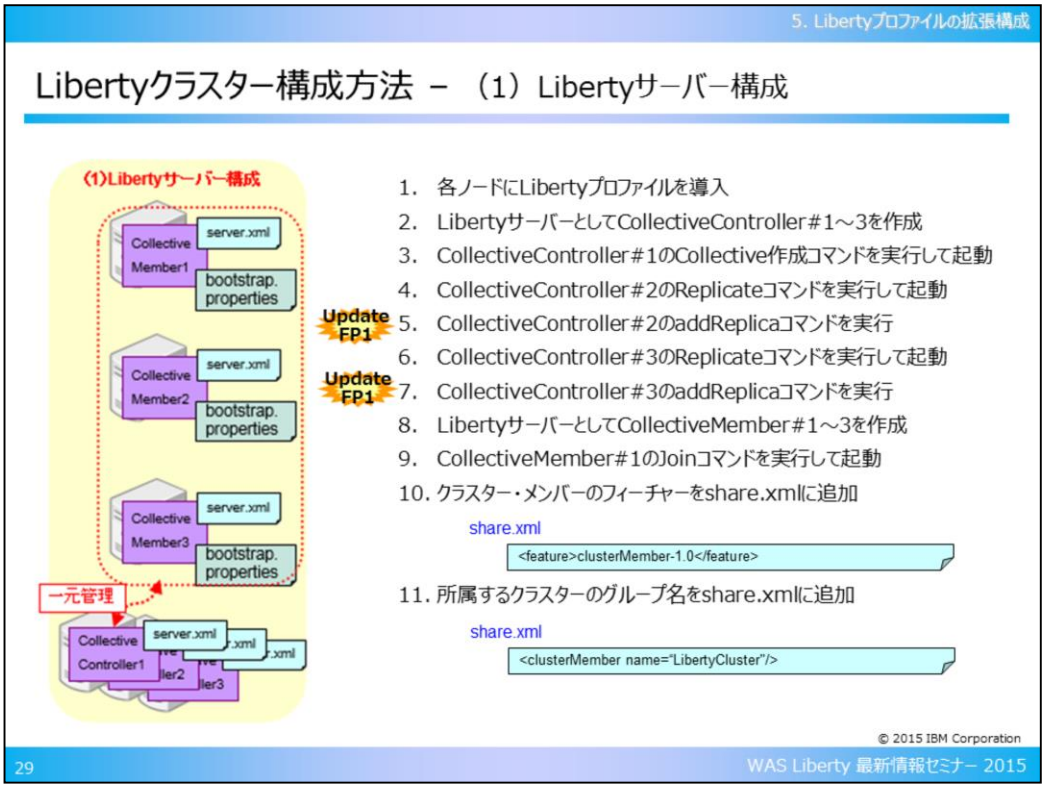
（1）Libertyサーバーを稼動させるノード上にLibertyプロファイルを導入してLibertyサーバーを作成し、Liberty Collective を構成してサーバー固有の設定をします。

（2）構成変更時の管理対象を最小化するために、Libertyサーバー共通となる構成情報とアプリケーションを共有させるように構成します。

（3）各Libertyサーバーへの負荷分散やフェールオーバーが行えるように、プラグインを構成します。

（4）セッション情報を取り扱うアプリケーションの場合は、Libertyサーバー同士でセッション情報を共有するように構成します。

これらのステップのうち、シンプル・クラスターの構成手順と異なるのは（1）と（3）のみですので、それらに関する詳細な手順を次ページ以降でご説明します。



こちらは、Libertyサーバーを稼動させるノード上にLibertyプロファイルを導入してLibertyサーバーを作成し、Liberty Collective を構成してサーバー固有の設定をする手順の例です。

1. Liberty Collective を構成する全てのノード上にLibertyプロファイルを導入します。

2. Collective Controller となるLibertyサーバーとしてCollectiveController#1～3を作成します。

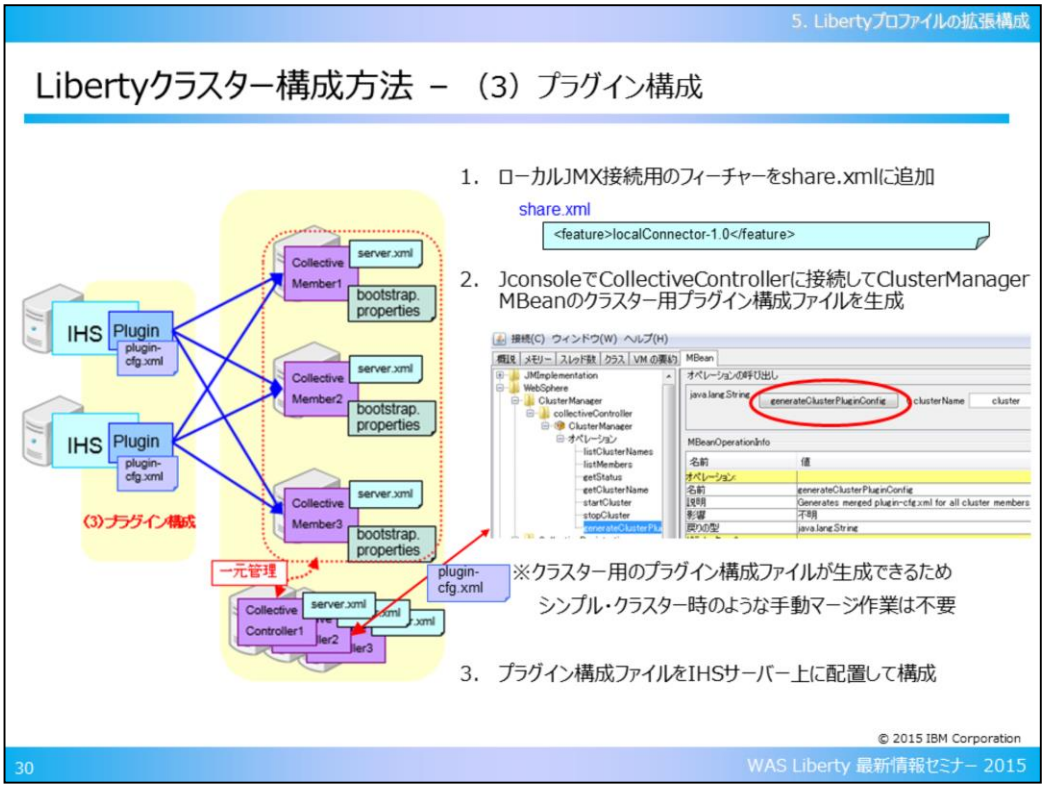
3～7. Libertyプロファイルで提供されているCollective Controller を構成するためのCollective、Replicate、addReplicaコマンドを使用してCollective Controller とReplica Set を構成します。

8. Collective Member となるLibertyサーバーとしてCollectiveMember#1～3を作成します。

9. Libertyプロファイルで提供されているCollective Member を構成するためのJoinコマンドを使用してCollective Member を構成します。

10. 共通設定を共有するためのshare.xmlファイルに「clusterMember-1.0」フィーチャーを追加します。

11. 更にshare.xmlにCollective Member が所属するクラスターのグループ名を定義します。

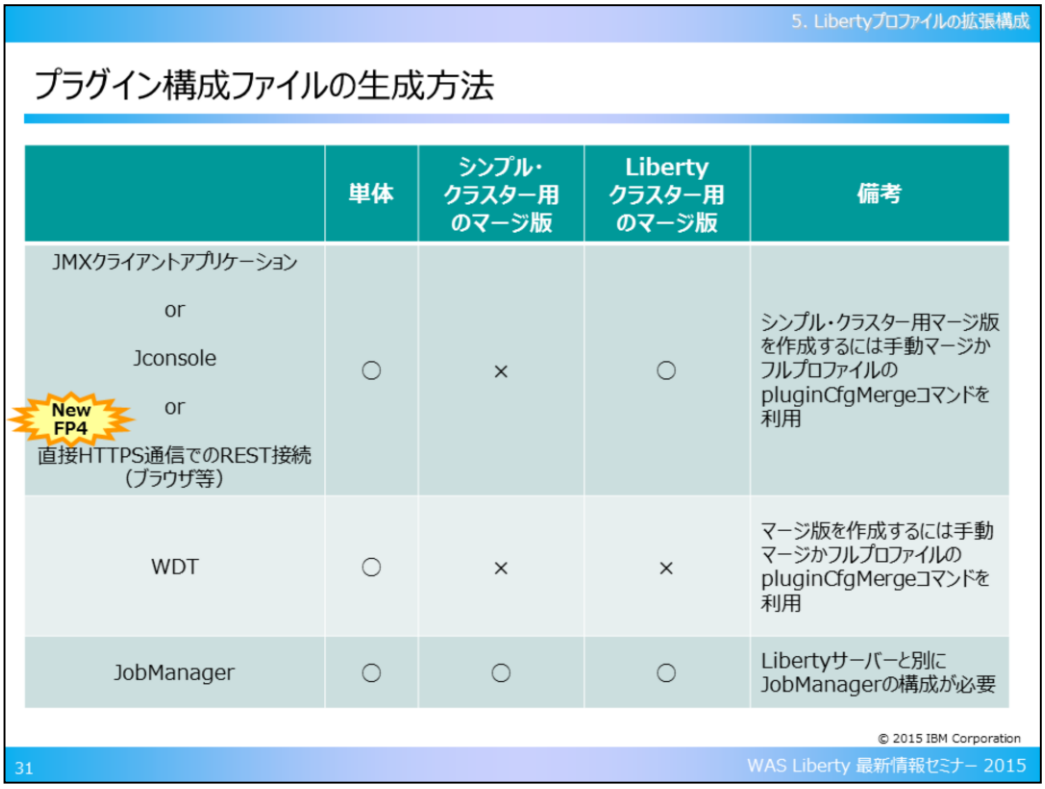


こちらは、各Libertyサーバーへの負荷分散やフェールオーバーが行えるように、プラグインを構成する手順の例です。

1. プラグイン構成ファイルを生成するために各LibertyサーバーにJMX接続を行う必要があるため、share.xmlにローカルJMX接続用のフィーチャー「localConnector-1.0」を追加します。

2. Jconsoleでいずれかの Collective Controller に接続し、ClusterManager Mbeanのプラグイン構成ファイルを生成するオペレーションを実行します。ここでは、既にクラスターのメンバー分の定義情報を含んだマージ版のプラグイン構成ファイルが生成されますので、手動でのマージ作業は不要です。

3. 生成されたプラグイン構成ファイルをIHSプラグインが読み込めるようにIHSサーバー上に配置します。

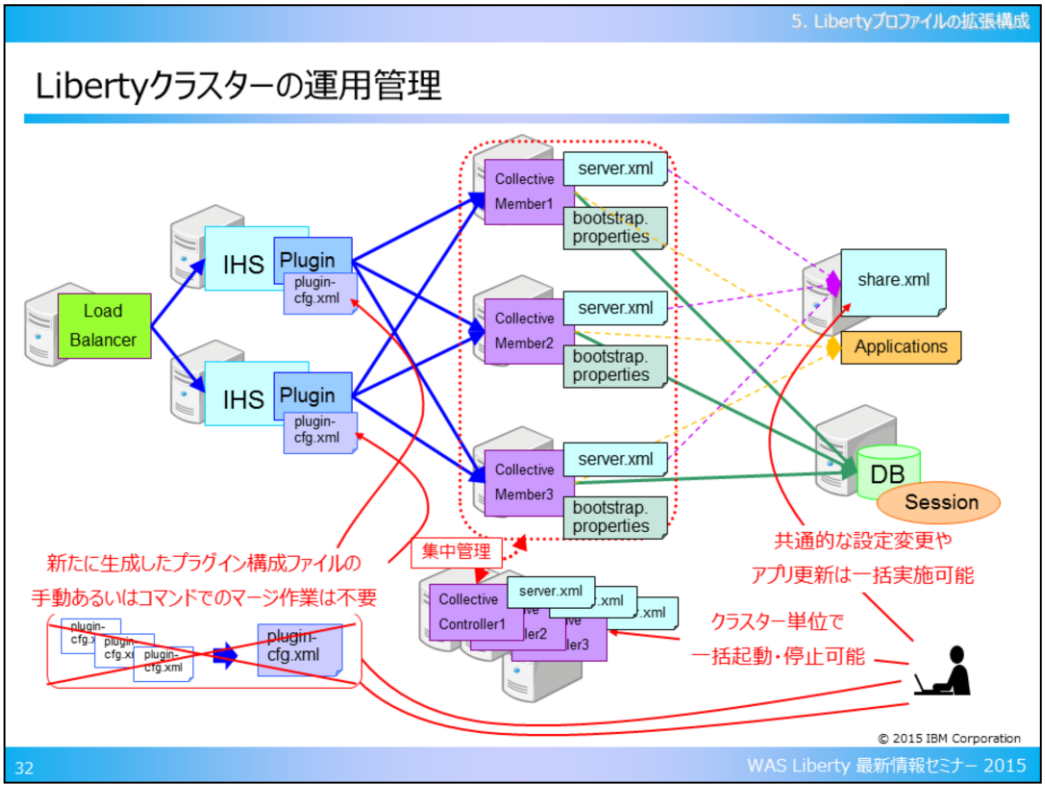


Libertyプロファイルにおけるプラグイン構成ファイルの生成方法には、JMXクライアントを使用する方法やブラウザなどから直接HTTPS通信でREST接続する方法、WDTを使用する方法、Job Manager を使用する方法などがあります。

方法によって、プラグイン構成ファイルのマージ版の生成可否が異なりますので、必要に応じて手動あるいはフルプロファイルで提供されているマージツールpluginCfgMergeを使用して1つにマージします。

＜参考＞

http://www-01.ibm.com/support/docview.wss?uid=swg21674883

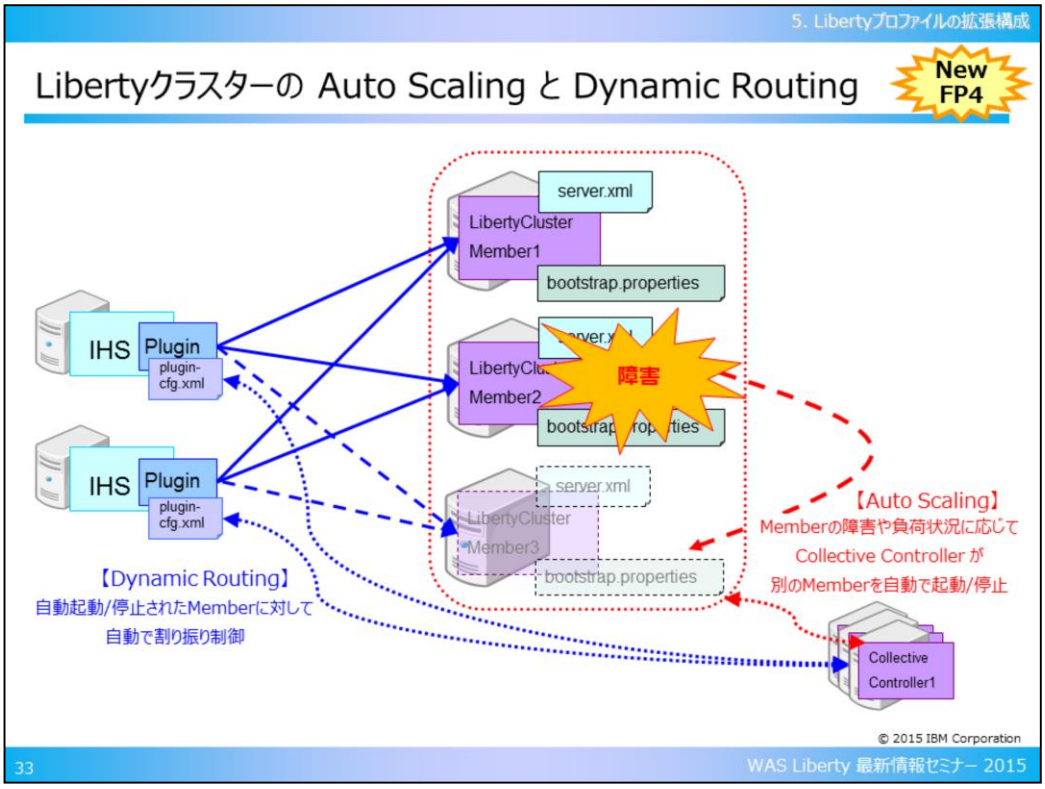


Libertyクラスターを構成した場合の運用管理の特徴をご説明します。

まず、共通的な設定やアプリケーション情報は1箇所に構成しましたので、構成変更やアプリケーション更新の際は、共有サーバー上の1箇所を更新することにより、全てのLibertyサーバーに情報を反映させることができます。

また、Libertyサーバーの起動・停止といったオペレーションは、Libertyサーバーそれぞれに対して実施する必要はなく、Liberty Controller 経由でクラスター単位で実施することができます。

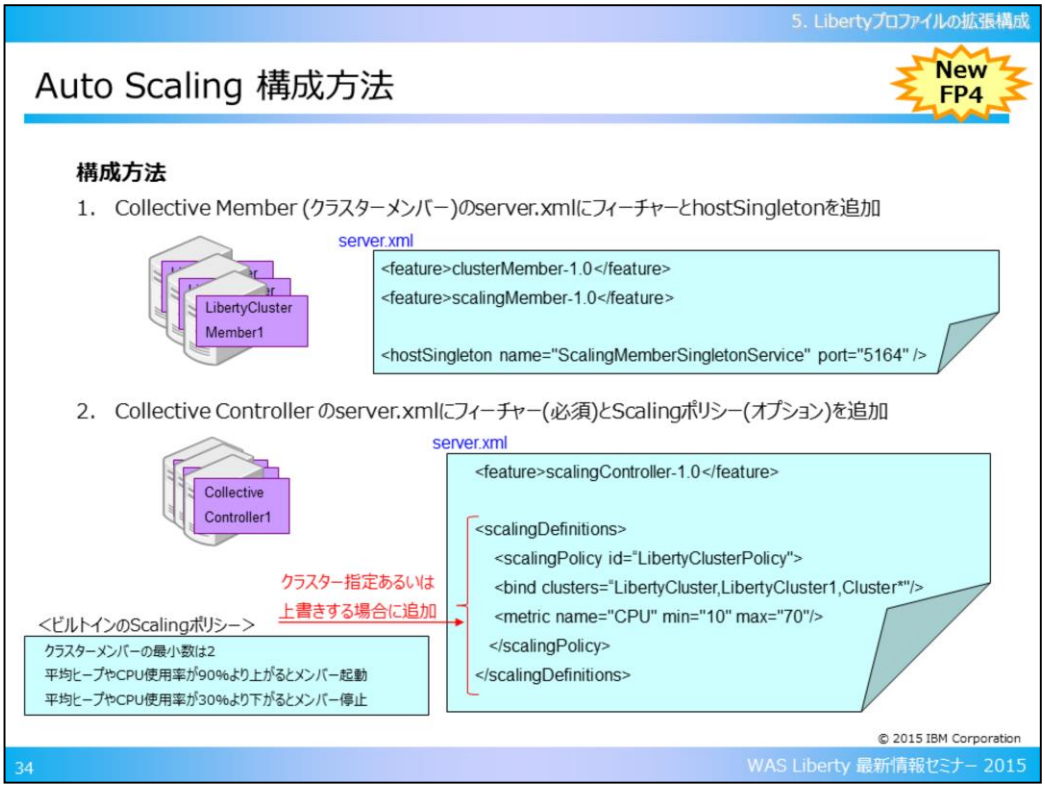
構成やアプリケーションの変更に伴ってプラグイン構成ファイルを新たに生成するたびに、手動あるいはフルプロファイルで提供されているコマンドを使用したマージ作業を行う必要はなく、Libertyクラスター用のマージ版を生成することができます。



Libertyクラスターでは、FixPack4（V8.5.5.4）からAuto Scaling とDynamic Routing という機能が提供されています。

Auto Scaling とは、LibertyクラスターのMemberの障害あるいは負荷状況に応じてCollective Controller が別のMemberを自動で起動/停止し、スケール・アウトあるいはスケール・インを行う機能です。

Dynamic Routing とは、Auto Scaling によって自動的に起動/停止されたMemberの稼働状況をCollective Controller から受け取り、自動的に割り振り制御を行う昨日です。



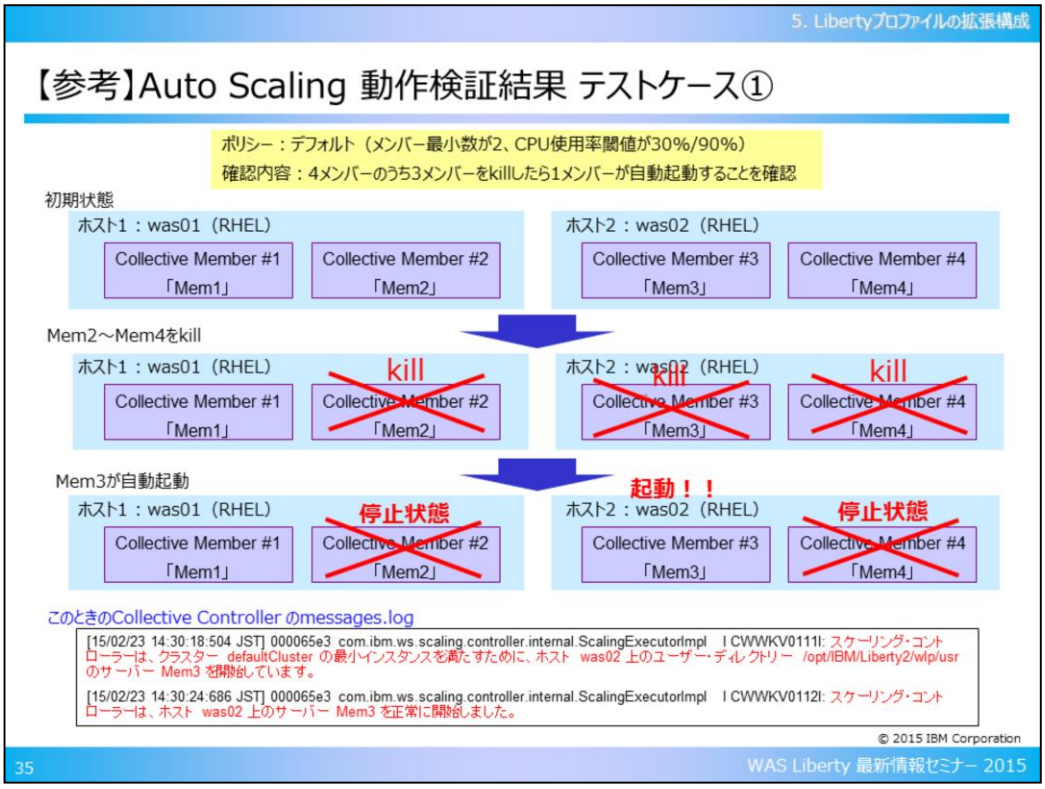
Auto Scaling は、Collective Member 側とCollective Controller 側両方の構成ファイルに定義を追加することで構成することができます。

1. Collective Member のserver.xmlファイルにAuto Scaling のための「scalingMemeber-1.0」フィーチャーとhostSingletonの定義を追加します。

2. Collective Controller のserver.xmlファイルに「scalingController-1.0」フィーチャー」と必要に応じてScalingポリシーの定義を追加します。Auto Scaling の動作条件を定義するためのScalingポリシーはビルトインで定義されていますが、それを上書きたい場合に構成ファイルに定義を追加します。

＜参考＞

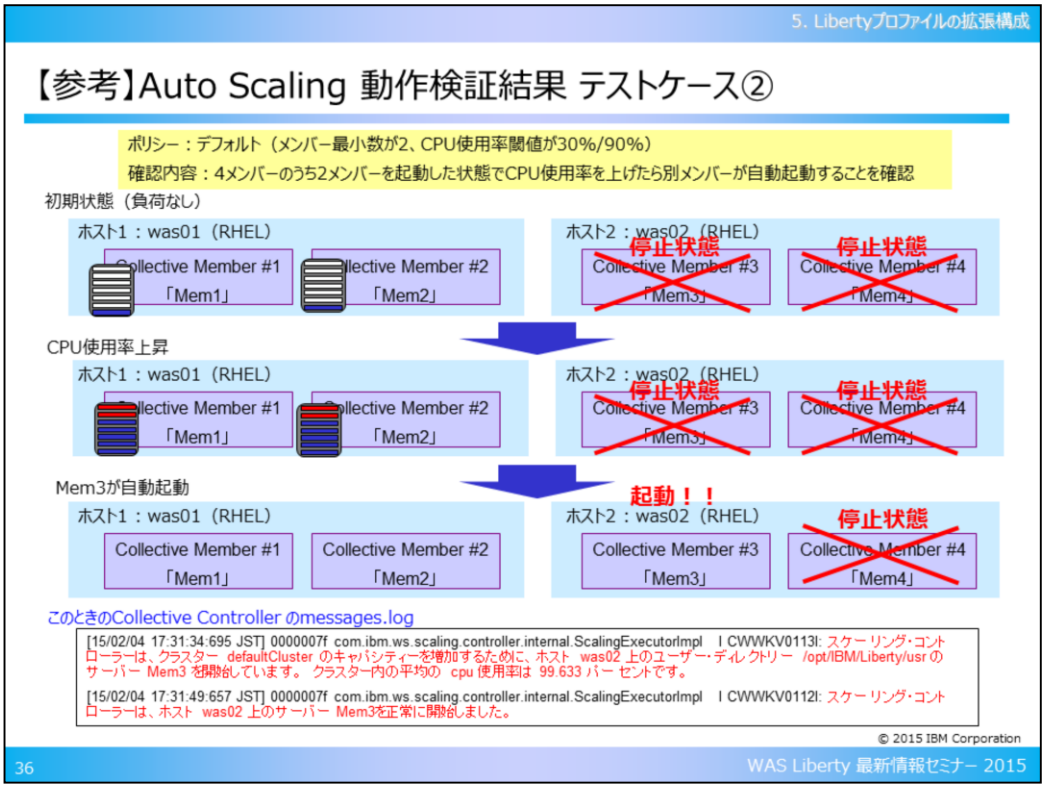
http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_wve_autoscaling.html



こちらは、実際にAuto Scaling の動作検証を行った際の検証結果です。

Scalingポリシーはデフォルトのまま、4つのMemberが稼動状態のときに3Memberをkillした際の動作を確認しました。

デフォルトのScalingポリシーではMemberの最小数は2となっているため、killした3Memberのうち1Memberだけが自動で起動してきました。その際、Collective Controller のmessages.logには最小インスタンスを満たすためにMemberを1つ起動した旨が記録されていました。

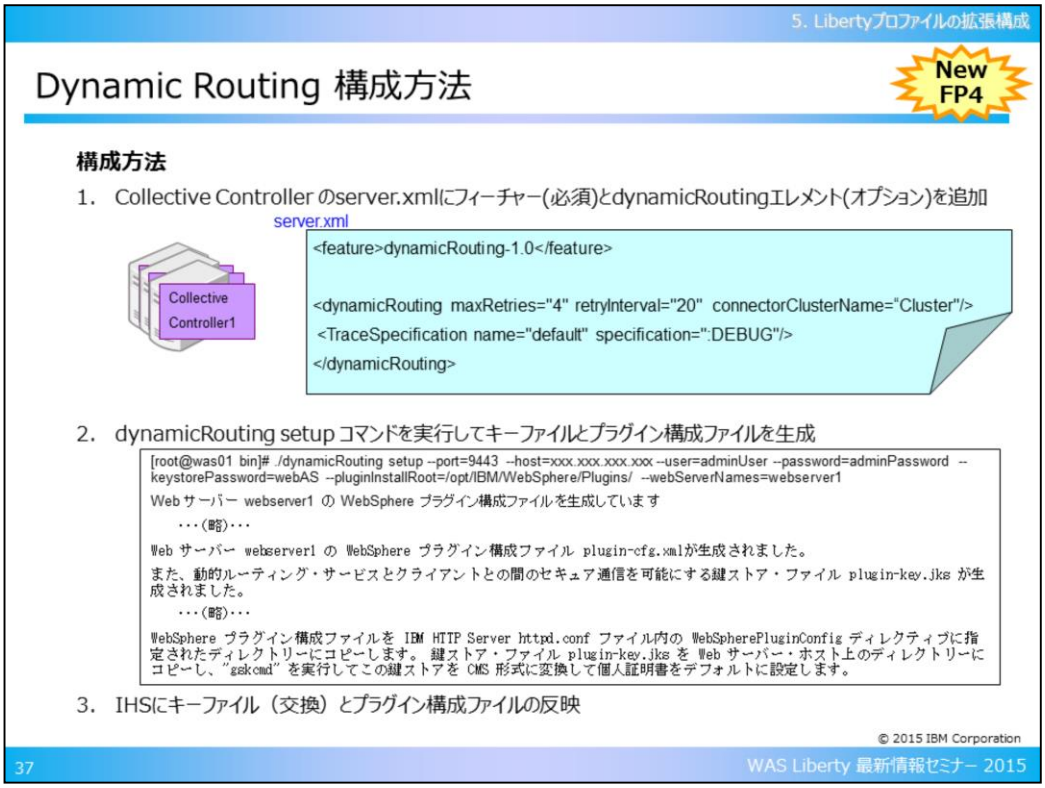


こちらは、実際にAuto Scaling の動作検証を行った際の検証結果です。

Scalingポリシーはデフォルトのまま、2つのMemberが稼動状態のときにCPU使用率を100%近く上げた際の動作を確認しました。

デフォルトのScalingポリシーではCPU使用率の閾値が30%/90%となっているため、1Memberだけが自動で起動してきました。その際、Collective Controller のmessages.logにはクラスターのキャパシティーを満たすためにMemberを1つ起動した旨が記録されていました。

また、Auto Scaling を構成している際は、1つのMemberをserverコマンドにより正常停止した場合においても、Scalingポリシーに該当する場合は別のMemberが起動されてくるという動作が確認できました。



Dynamic Routing は、Collective Controller 側の構成ファイルに定義を追加し、プラグイン構成ファイルを生成して配置することで構成することができます。

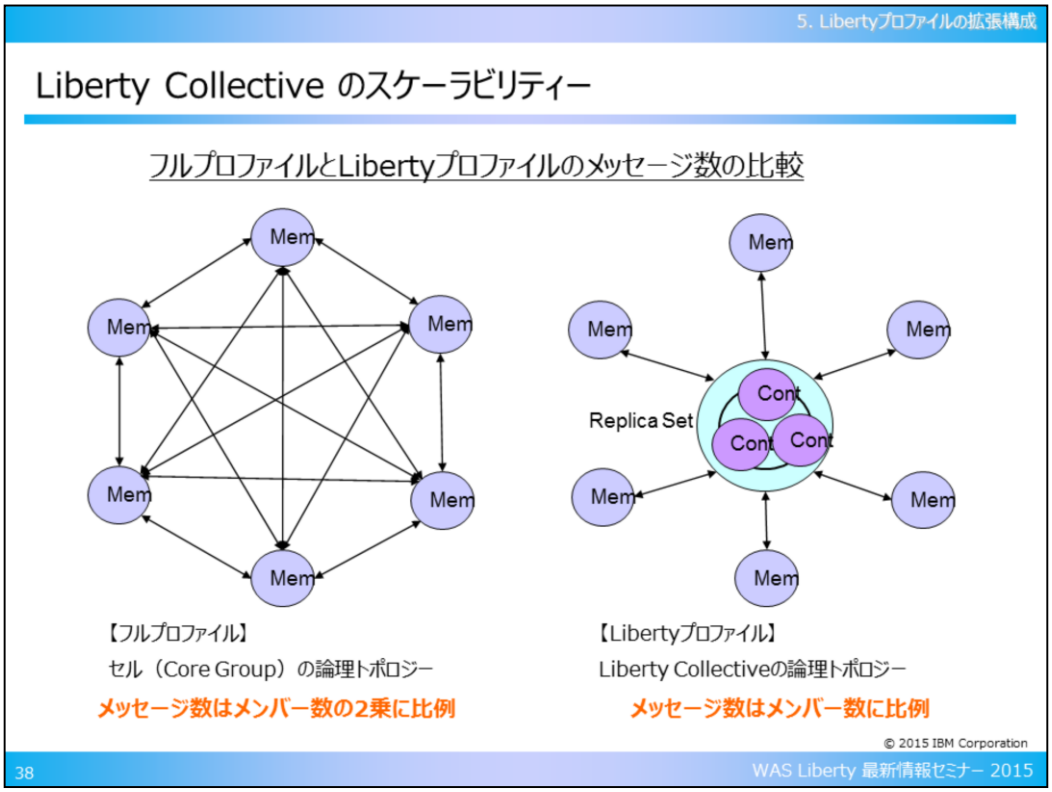
1. Collective Controller のserver.xmlファイルにDynamic Routing のための「dynamicRouting-1.0」フィーチャーと必要に応じてdynamicRouting の定義を追加し、最大リトライ回数やリトライ間隔等を設定します。

2. Libertyプロファイルで提供されているdynamicRouting setup コマンドを実行してキーファイルとプラグイン構成ファイルを生成します。

3. IHSとLibertyサーバー間のSSL通信のためにキーファイル交換を行い、生成したプラグイン構成ファイルをIHSに配置します。

＜参考＞

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_wve_enabledynrout.html

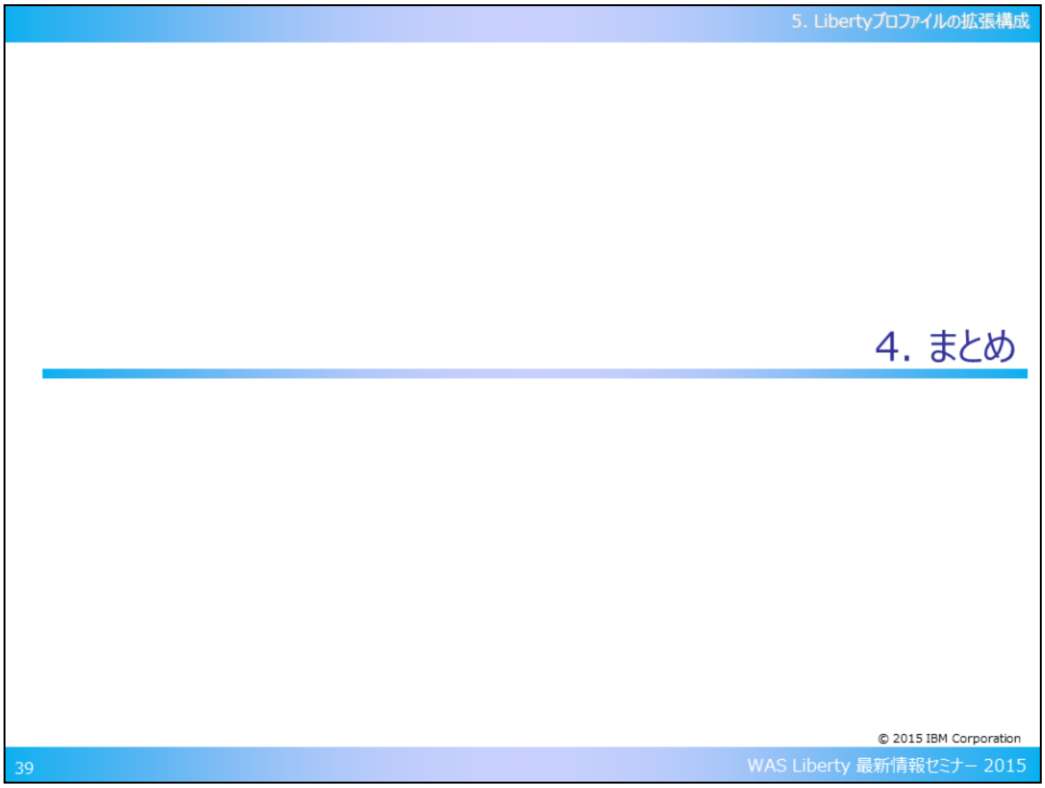


Liberty Collective のスケーラビリティーは、フルプロファイルと比較して優位であるといえます。

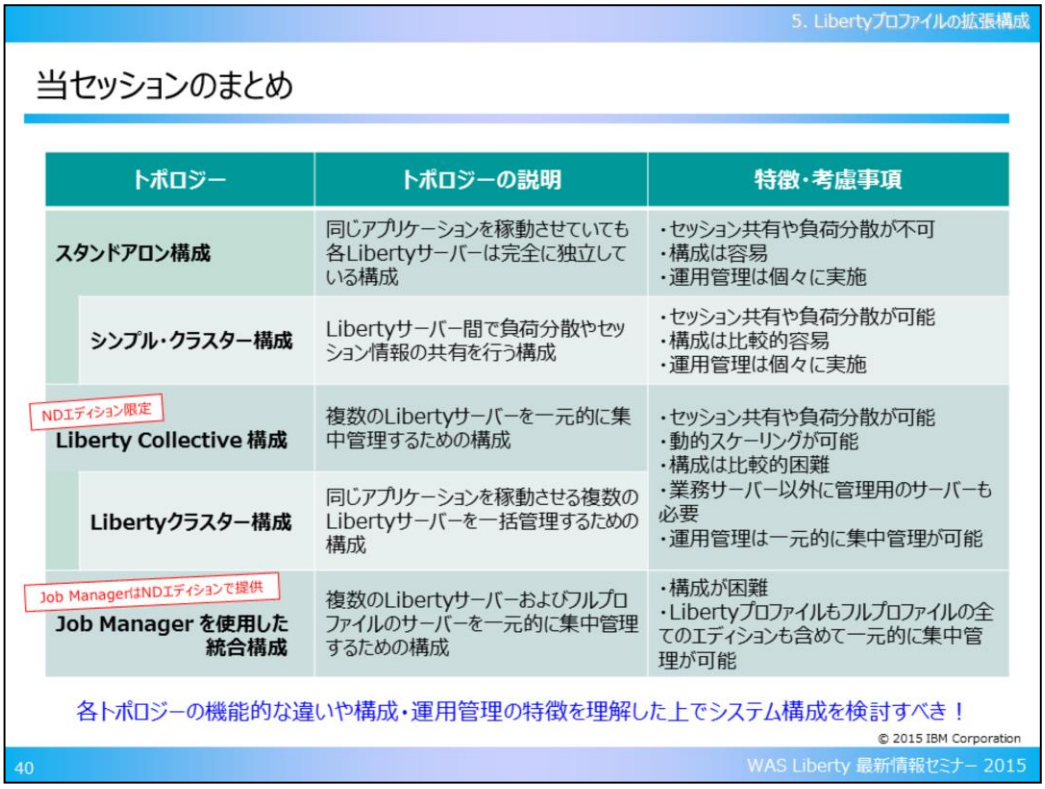
フルプロファイルのセル（Core Group）では、アプリケーションを稼動させるアプリケーション・サーバーやクラスターメンバーといったJVMの数が増えると、JVM同士の通信でやりとりされるメッセージの数はメンバー数の2乗に比例して増えていきます。

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。



4章では、これまでの内容をまとめます。



当セッションでは、Libertyプロファイルのトポロジーとして、スタンドアロン構成、シンプル・クラスター構成、Liberty Collective 構成、Libertyクラスター構成、Job Manager を使用した統合構成をご紹介し、それぞれの構成方法や特徴などについてご説明しました。

Libertyプロファイルを採用したシステムにおいてどのようなトポロジーにするか検討する際は、それぞれのエディションや機能的な違いだけでなく、構成方法や運用管理の容易性や特徴などを理解した上で最適な選択をしていただく必要があります。

Liberty Collective に関しては、業務サーバーとは別に管理用のLibertyサーバー（推奨は3つ以上）を構成し管理する必要がありますので、業務サーバー数が少ないとその分のオーバーヘッドが大きくなるため、業務サーバー数が比較的多い場合にご利用いただくのが良いのではないかと考えられます。


5. Libertyプロファイルの拡張構成
THANK
YOU
© 2015 IBM Corporation
41
WAS Liberty 最新情報セミナー 2015