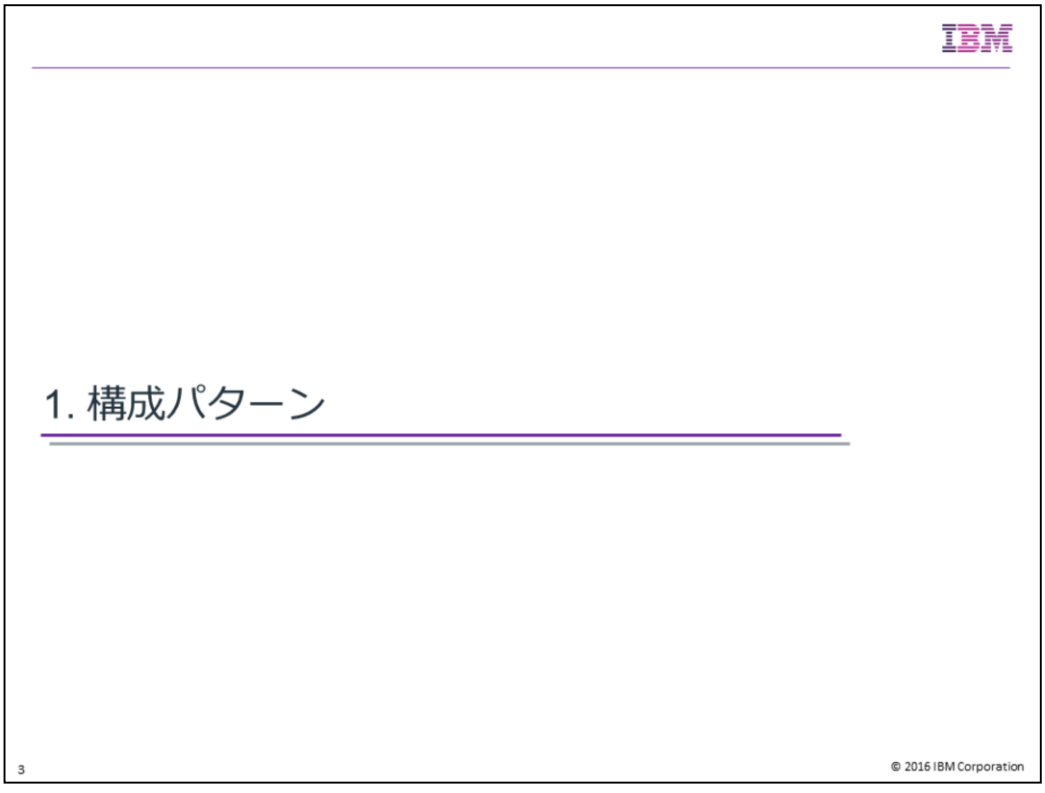
このセッションでは、Libertyランタイムを使った場合のトポロジー・パターンや、各トポロジーを構成する上で前提となるあるいは利用できる機能について説明します。また、それぞれのパターンにおける障害時動作についても例を挙げ、トポロジーを検討する上での指針をまとめています。

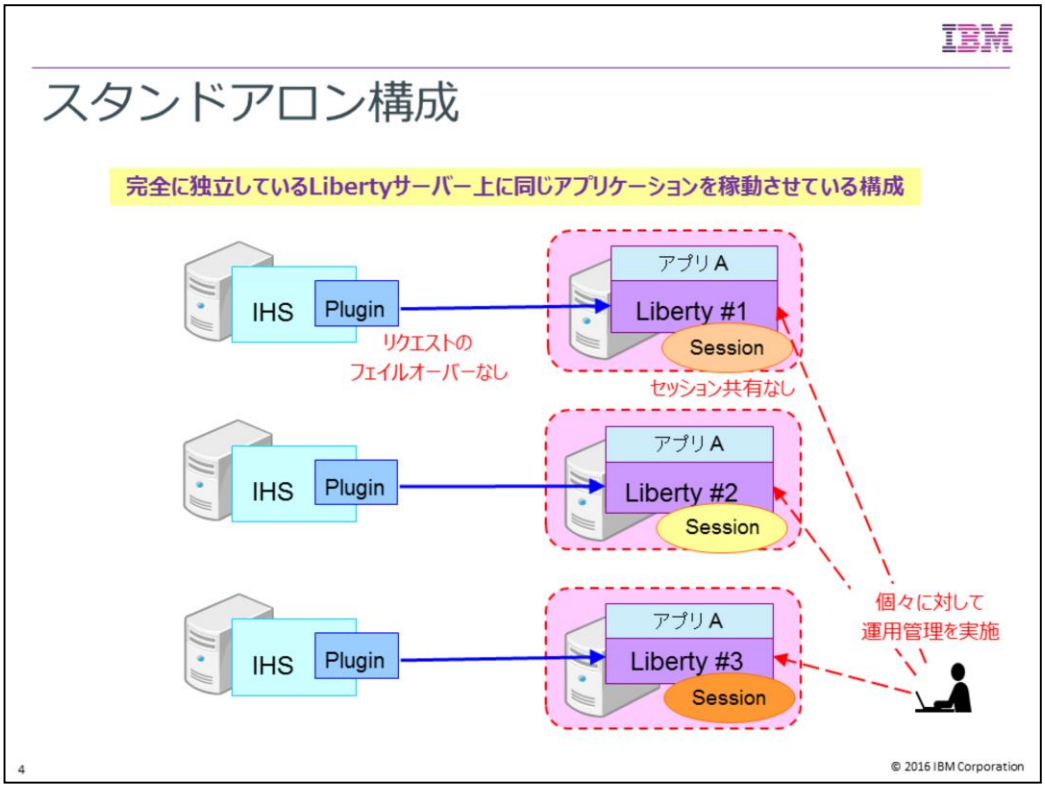
IBM

アジェンダ

- 1. 構成パターン
- 2. 基本機能
- 3. 高可用性機能
- 4. 構成パターン別障害時動作
- 5. トポロジー選択指針
- まとめ



この章では、Libertyを使った場合の構成を3パターンに分類し、それぞれの特徴ついて説明します。

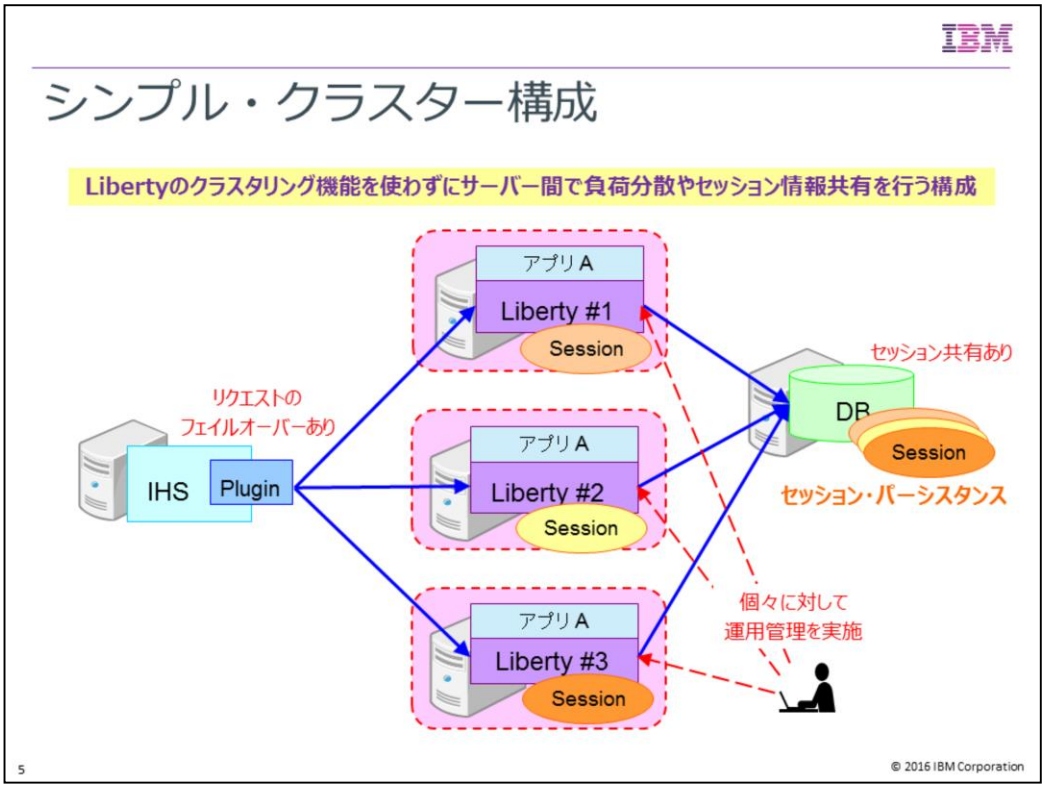


スタンドアロン構成とは最もシンプルなトポロジーで、完全に独立しているLibertyサーバー上で同じアプリケーションを稼動させてる構成です。

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・稼働状況確認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。

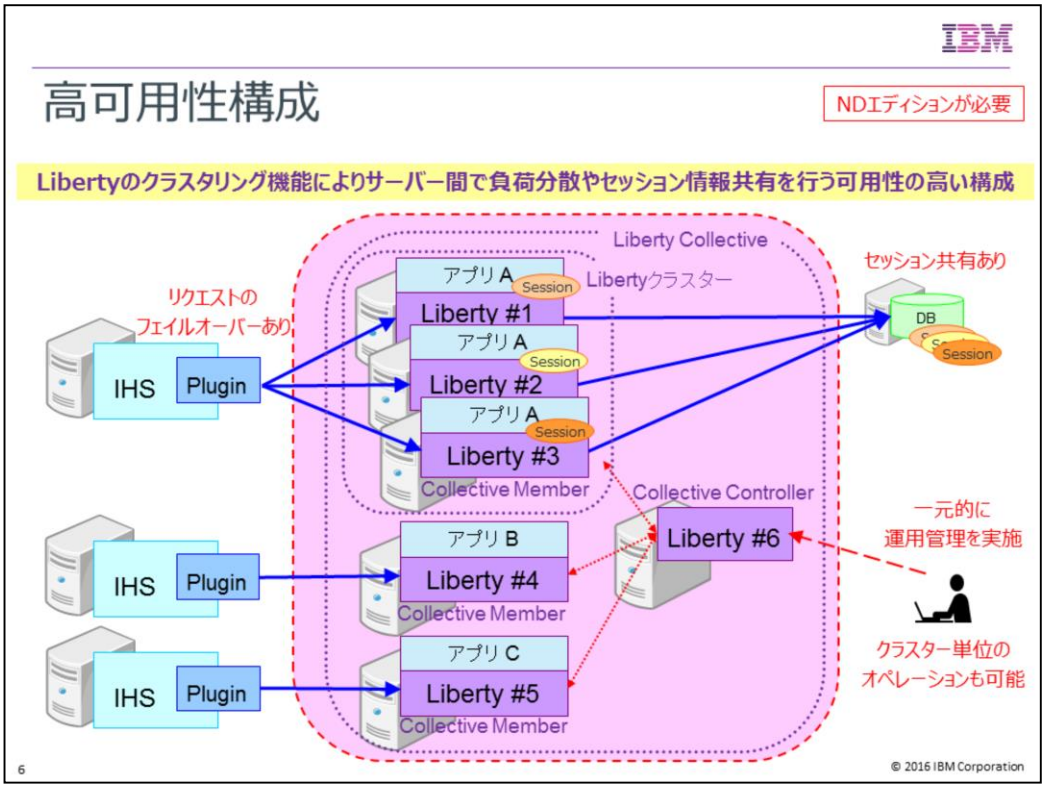


シンプル・クラスター構成とはスタンドアロン構成に耐障害性を持たせたトポロジーで、Libertyのクラスタリング機能を使わずにでLibertyサーバー間で負荷分散やセッション情報の共有を行う構成です。

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るため、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・稼働状況確認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。



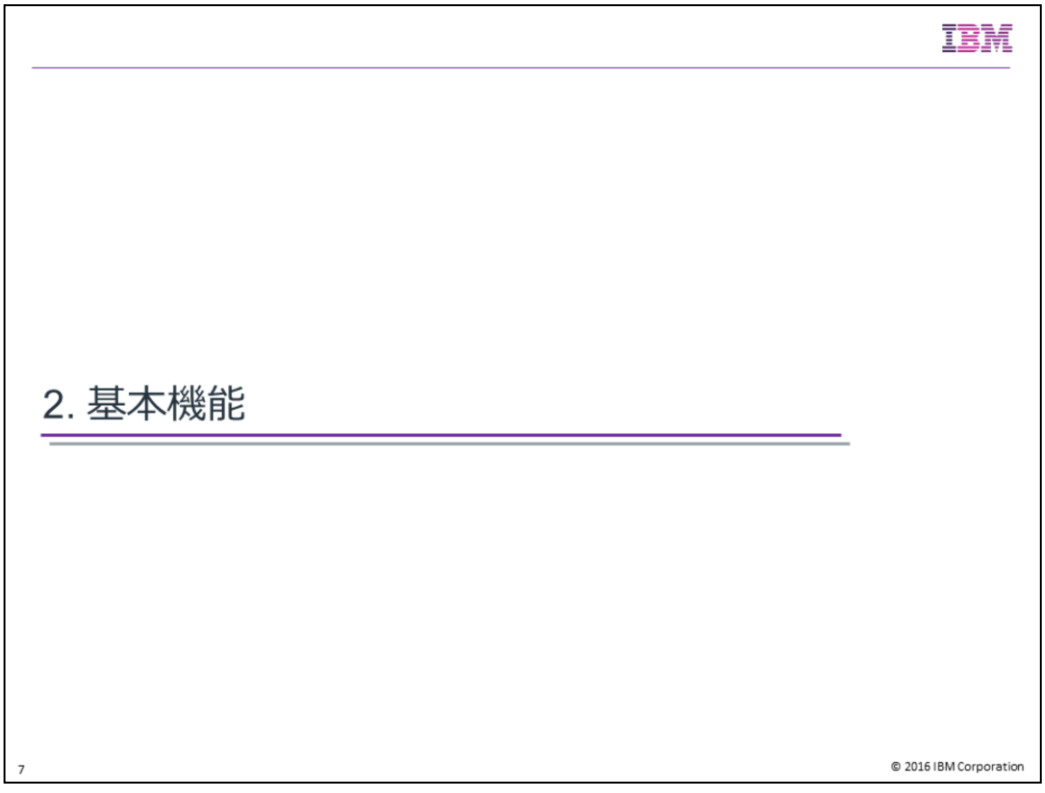
高可用性構成とは、Libertyのクラスタリング機能を使ってサーバー間で負荷分散やセッション情報共有を行い、更に複数のLibertyサーバーを一括管理する構成です。

Libertyのクラスタリング機能は、Liberty Collective という名称で、業務アプリケーションを稼動させるためのCollective Member と管理用のCollective Controller で構成されています。

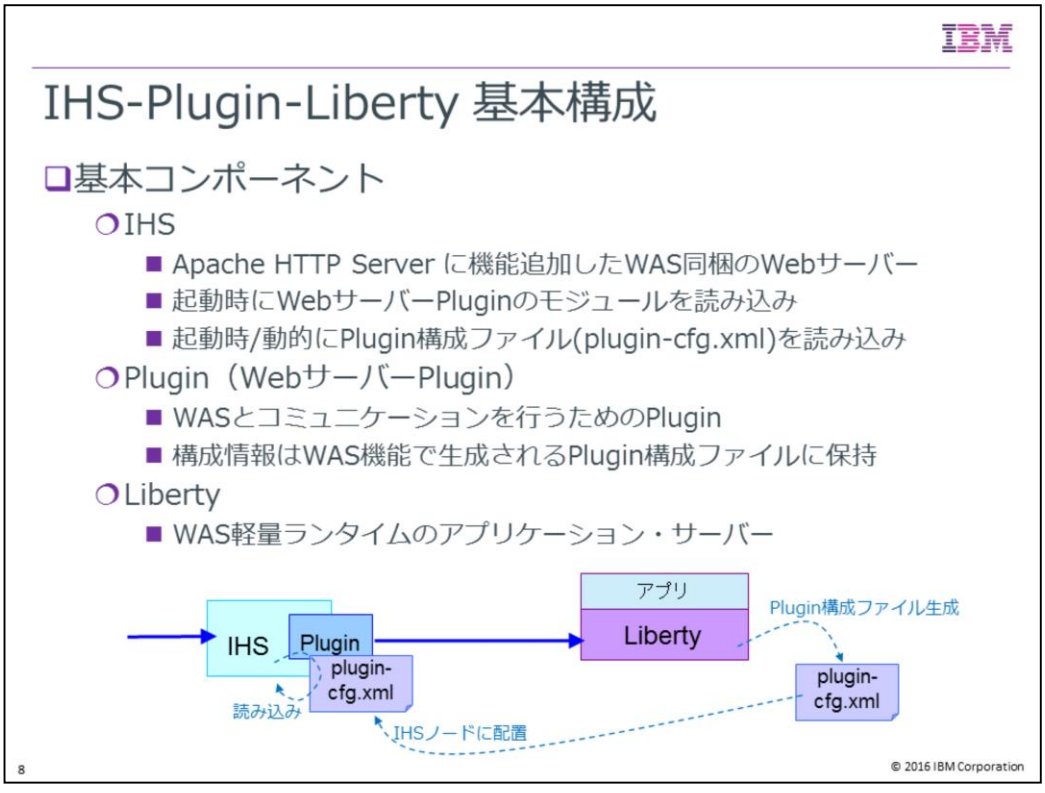
Libertyクラスター構成とはLiberty Collective 内で構成できるトポロジーで、同じアプリケーションを稼動させる複数のCollective Member をLibertyクラスターと定義することでグループ化することができます。

Liberty Collective を構成するにはWASのNDエディションが必要となります。上図のうち、Collective Controller やLibertyクラスターのメンバーであるLiberty#1、2、3、6に関しては、NDエディションで提供されるLibertyフィーチャーを使用するため、NDエディションでなければなりません。上図のうち、Libertyクラスターのメンバーではないコレクティブ・メンバー であるLiberty#4、5に関しては、Liberty Coreエディションや Baseエディションでも構成できます。

起動・停止・稼働状況確認といった運用管理は、管理用のCollective Controller となるLibertyサーバーに接続することにより、Liberty Collective 内のCollective Member あるいはLibertyクラスター単位で実施することができます。



この章では、Libertyのトポロジーを検討する上で理解しておく必要がある基本機能について説明します。



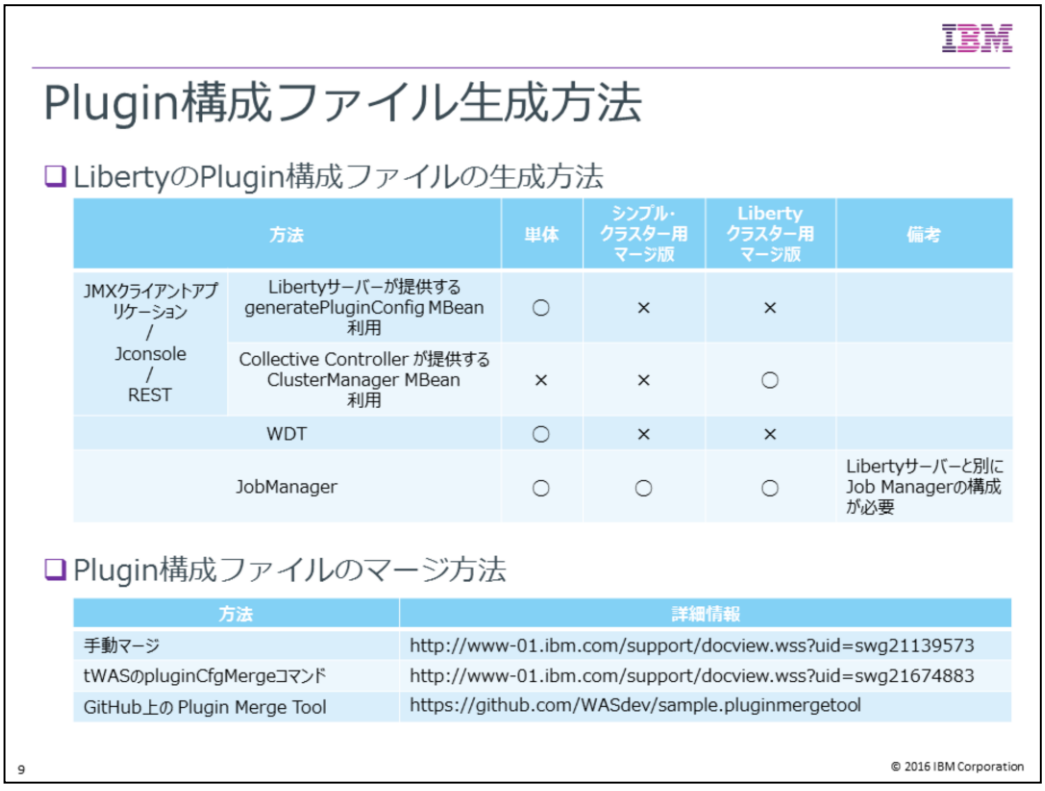
一般的な構成となるIHS-Plugin-Libertyの基本コンポーネントについて説明します。

IHSとは、Apache HTTP Server に機能追加したWASに同梱されたWebサーバーです。IHS起動時にWebサーバーPluginのモジュールおよびPlugin構成ファイル(plugin-cfg.xml)を読み込むことで、後段のLibertyサーバーにリクエストを転送します。

Plugin(WebサーバーPlugin)とは、WASとコミュニケーションを行うためにWebサーバーに組み込まれるPluginであり、構成情報はWAS機能によって生成されるPlugin構成ファイルに保持しています。

Libertyとは、WAS軽量ランタイムのアプリケーション・サーバーです。

IHSとPluginの機能や構成方法は従来のWAS traditional (以降、tWAS)と何も変わりません。ただ、IHSノードに配置させるPlugin構成ファイルの生成の方法がtWASとLibertyでは異なります。



Libertyランタイムにおける Plugin構成ファイルの生成方法には、JMXクライアント/Jconsole/RESTを使用してLibertyが提供しているMbeanにアクセスして生成する方法、WDTを使用する方法、Job Manager を使用する方法などがあります。

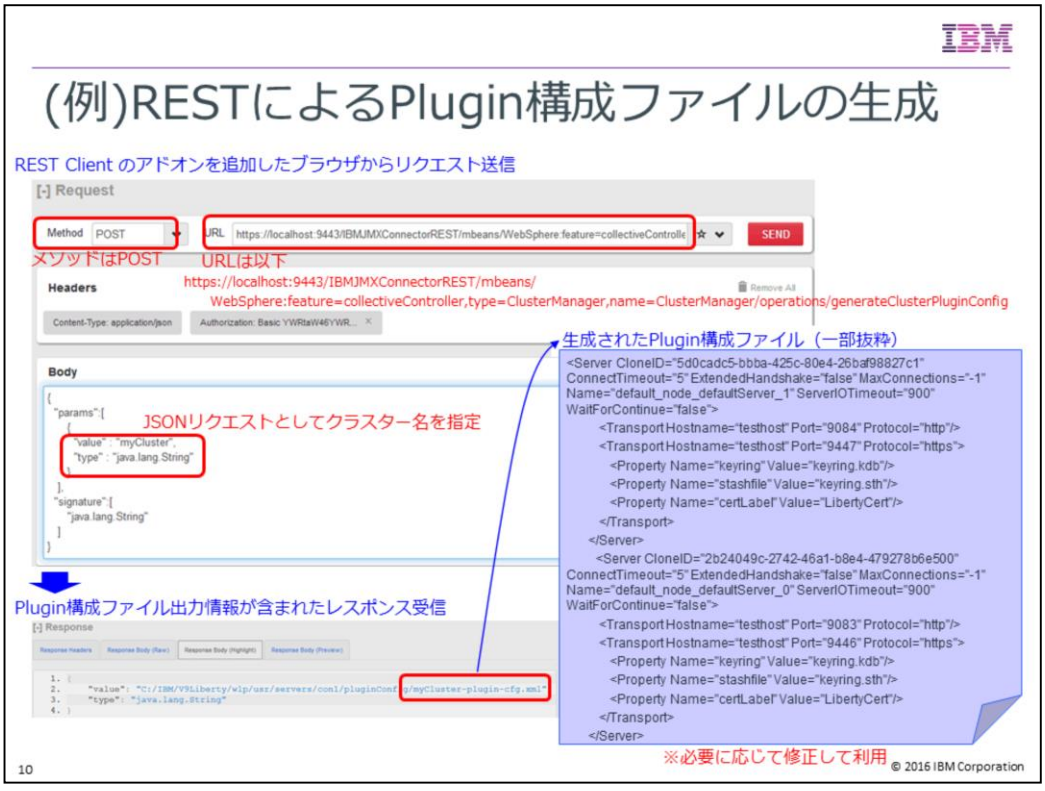
上表の通り、方法によって生成できるPlugin構成ファイルの種類が異なります。Libertyサーバー単体で提供されるgeneratePluginConfig MBean にアクセスすると単体のPlugin構成ファイルが生成され、Collective Controller で提供されるClusterManager Mbean にアクセスするとLibertyクラスターのマージされたPlugin構成ファイルが生成されます。

単体のLibertyサーバーから生成された複数のPlugin構成ファイルをマージする方法としては、手動編集する方法、tWASで提供されているpluginCfgMergeコマンドを使用する方法、GitHub上で提供されているPlugin Merge Tool を使用する方法、があります。それぞれの詳細な手順につきましては、リンク先の情報を参照ください。

＜参考＞

Knowledge Center 「plugin-cfg.xml ファイルの生成」

http://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.doc/ae/twlp_generate_plugin_cfg_xml.html

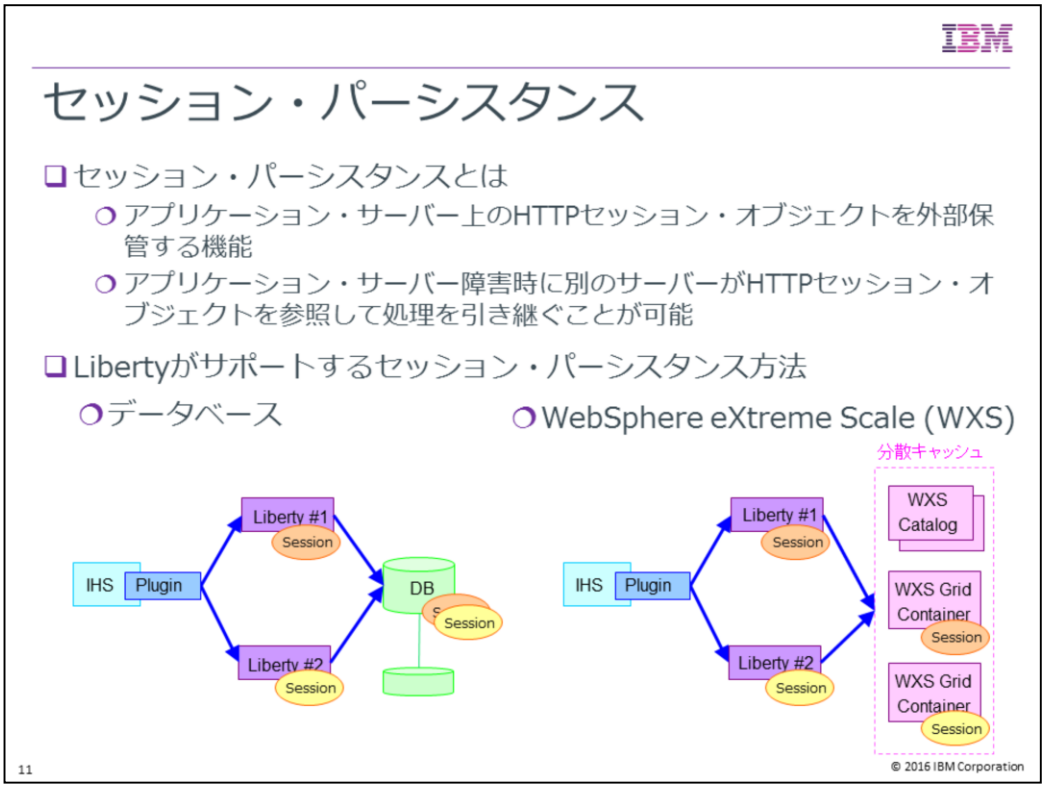


一例として、RESTを使ってCollective Controller のClusterManager Mbean にアクセスしてLibertyクラスターのPlugin構成ファイルを生成する方法を紹介します。

REST Client のアドオンを追加したブラウザからRESTリクエストを送信します。宛先URLはCollective Controller のHTTPSポートを指定して「https://<hostname>:<httpsport>/IBMJMXConnectorREST/mbeans/WebSphere:feature=collectiveController,type=ClusterManager,name=ClusterManager/operations/generateClusterPluginConfig」とします。メソッドはPOSTでBodyにはJSONリクエストとして上記のようにLibertyクラスター名を指定します。Headersとして「Content-Type:application/json」と「Authorization:Basic」を追加します。

RESTのリクエストに対するレスポンスとして、Plugin構成ファイルが出力されるパスが返されます。

ファイルシステム上のそのパスにLibertyクラスター用のマージされたPlugin構成ファイルが生成されますので、必要に応じて修正して利用することができます。



セッション・パーシスタンスとは、アプリケーション・サーバーのメモリ上にあるHTTPセッション・オブジェクトを外部保管する機能です。セッション・パーシスタンスの機能を利用すると、あるアプリケーション・サーバーに障害が発生した際に、別のアプリケーション・サーバーが外部保管されたHTTPセッション・オブジェクトを参照して処理を引き継ぐことができます。

セッション・パーシスタンスの機能や効果はtWASとLibertyでは変わりありませんが、Libertyでサポートされるセッション・パーシスタンス方法はデータベースとWebSphere eXtreme Scale (WXS)だけであり、tWASで提供されているメモリ間複製はサポートされていません。

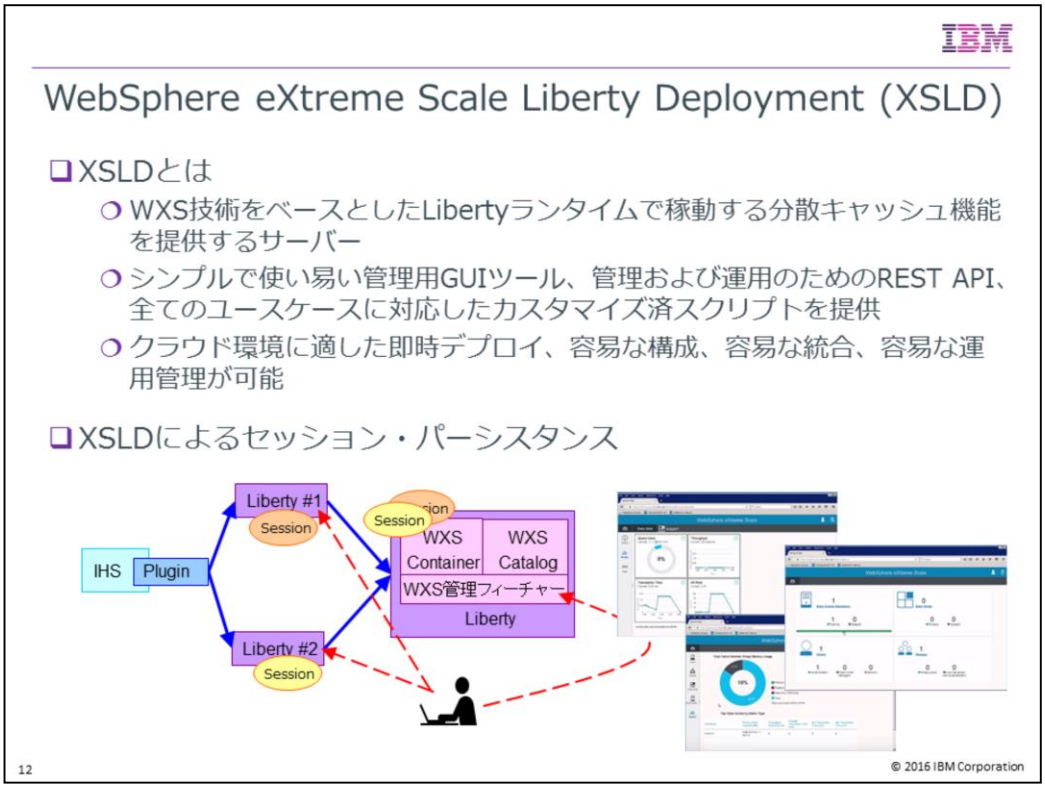
＜参考＞

Knowledge Center 「Liberty のセッション・パーシスタンスの構成」

https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.doc/ae/twlp_admin_session_persistence.html

Knowledge Center 「ビデオ: Configure session cache management with Liberty and WebSphere eXtreme Scale (Liberty および WebSphere eXtreme Scale でのセッション・キャッシュ管理の構成)」

https://www.ibm.com/support/knowledgecenter/SSAW57_liberty/com.ibm.websphere.wlp.nd.doc/ae/video_transcript_xs_session_cache.html



WebSphere eXtreme Scale Liberty Deployment (XSLD)とは、WXS技術をベースとしたLibertyランタイムで稼動する分散キャッシュ機能を提供するサーバーです。

Libertyランタイムで稼動するため軽量であるというだけでなく、シンプルで使い易い管理用GUIツールや運用管理のためのREST APIが提供されています。更に、全てのXSLDのユースケースに対応したカスタマイズ済スクリプトも提供しているため、構成の負荷も軽減されます。

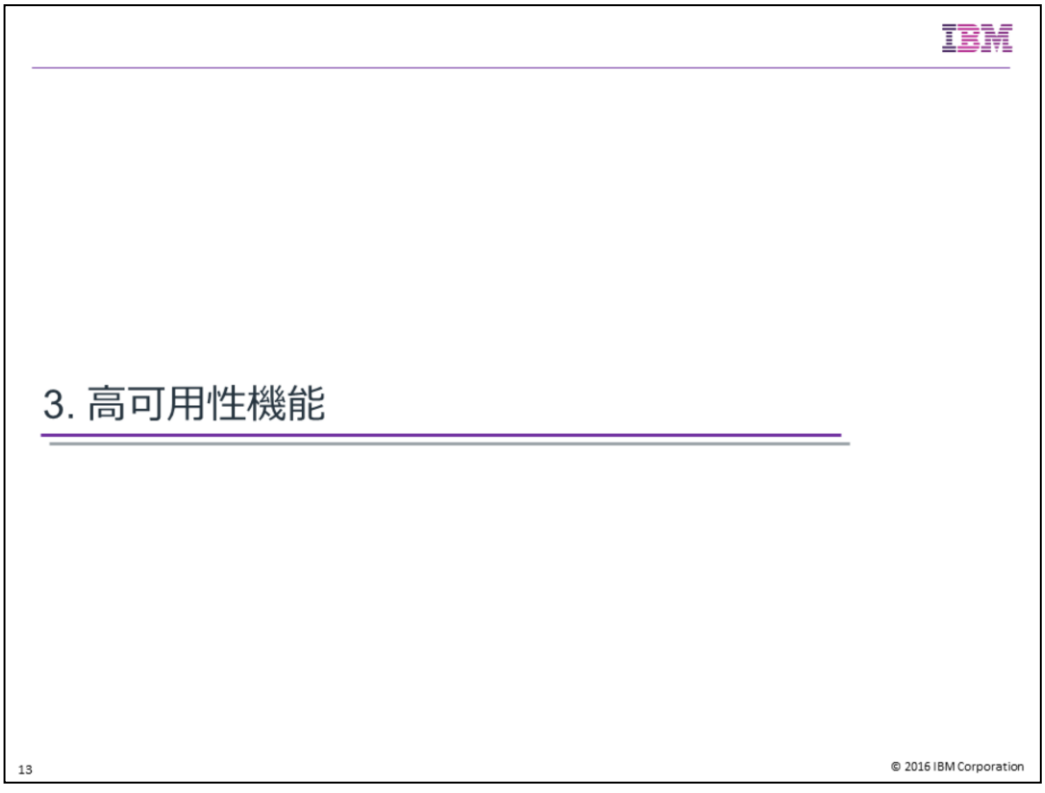
XSLDを使用することにより、クラウド環境に適した即時デプロイ、容易な構成、容易な統合、容易な運用管理を行うことができるようになります。

また、Libertyランタイムで稼動するため、他の業務サービスを提供するLibertyサーバーと同じ方法で運用管理を行うこともできます。

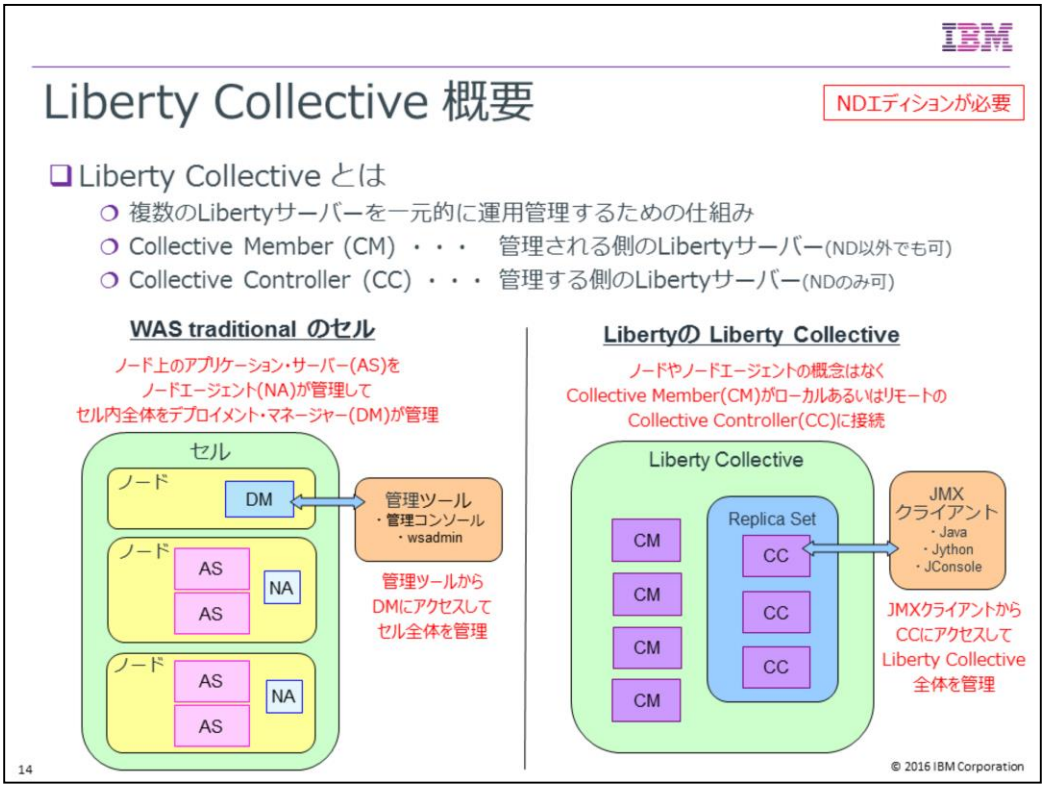
＜参考＞

Knowledge Center 「WebSphere eXtreme Scale Liberty Deployment」

https://www.ibm.com/support/knowledgecenter/SSTVLU_8.6.1/com.ibm.websphere.extremescale.doc/cxsgetstartxsld.html



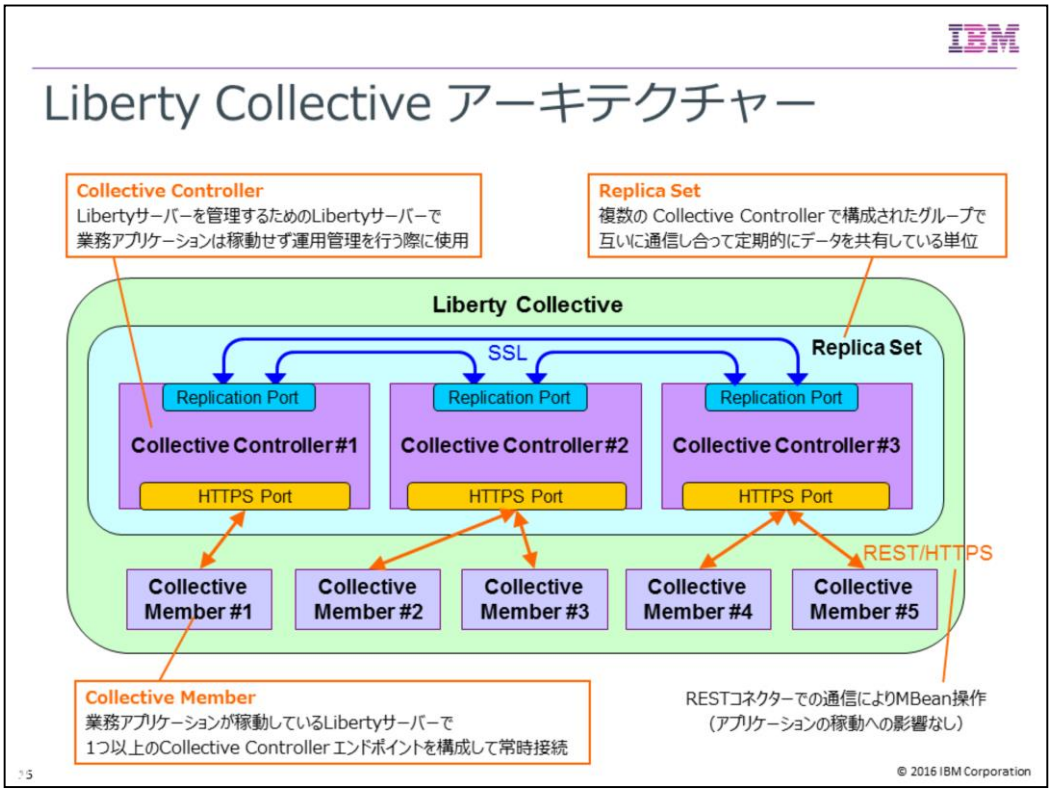
この章では、Libertyのトポロジーを検討する上で理解しておく必要がある高可用性機能について説明します。



LibertyランタイムではLiberty Collective という機能が提供されており、複数のLibertyサーバーをまとめて管理することができます。

tWAS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。

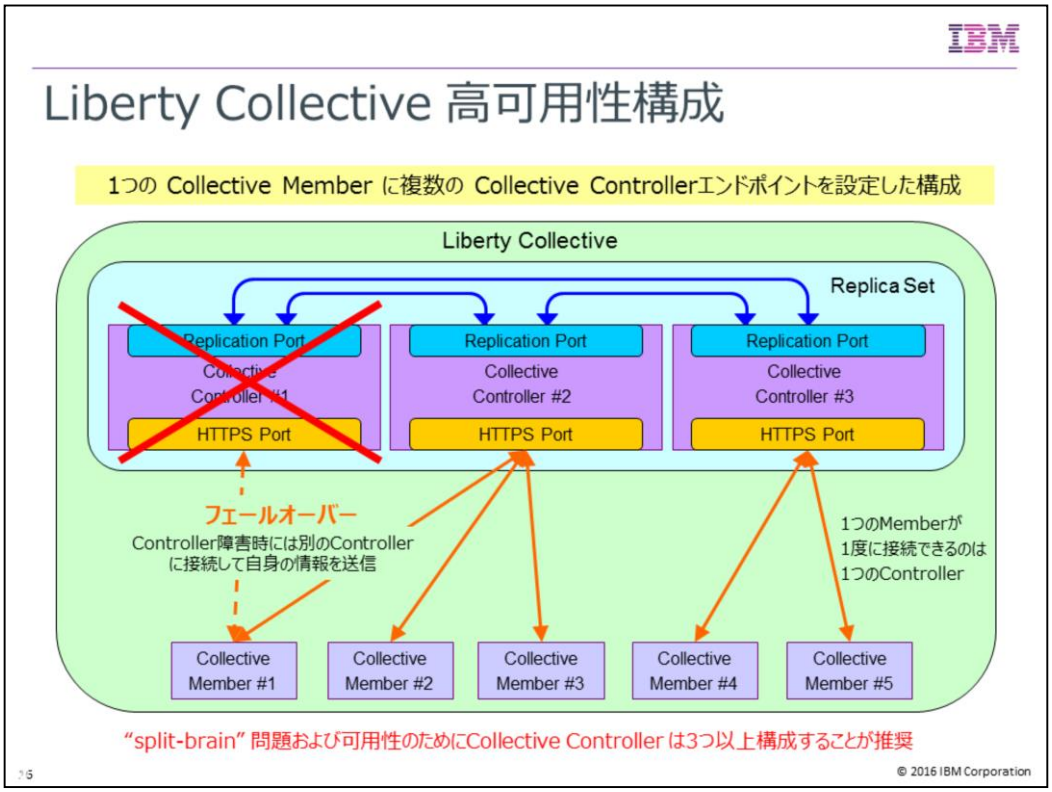
LibertyランタイムのLiberty Collective 構成では、ノードやノードエージェントといった概念はなく、業務アプリケーションを処理するLibertyサーバーであるCollective Member と管理用のLibertyサーバーであるCollective Controller で構成されたトポロジーになります。Collective Member がローカルあるいはリモートのCollective Controller に接続することでCollective Controller がCollective Member を管理できるようになります。管理者はJMXクライアントからCollective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

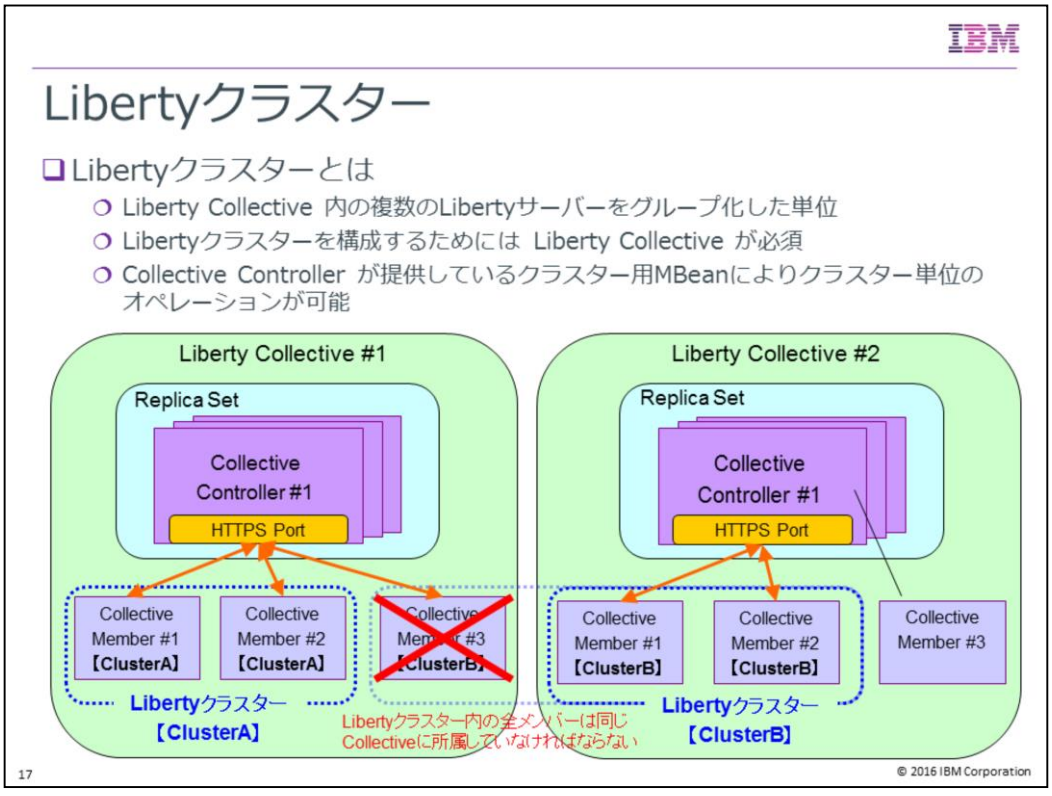
Liberty Collective は1つ以上のCollective Controller で構成されますが、複数のCollective Controller を配置することにより、可用性を持たせることができます。



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。

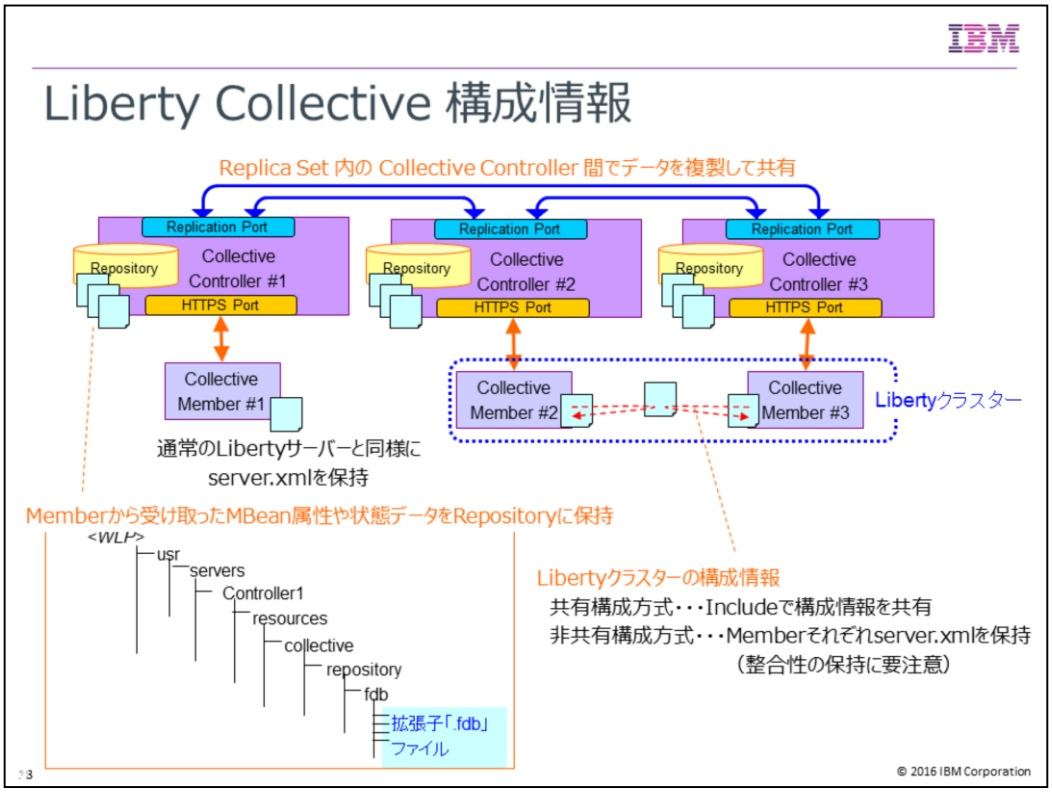
1つのLiberty Collective において、”split-brain”問題のためにCollective Controller は３つ以上構成することが推奨されています。



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。

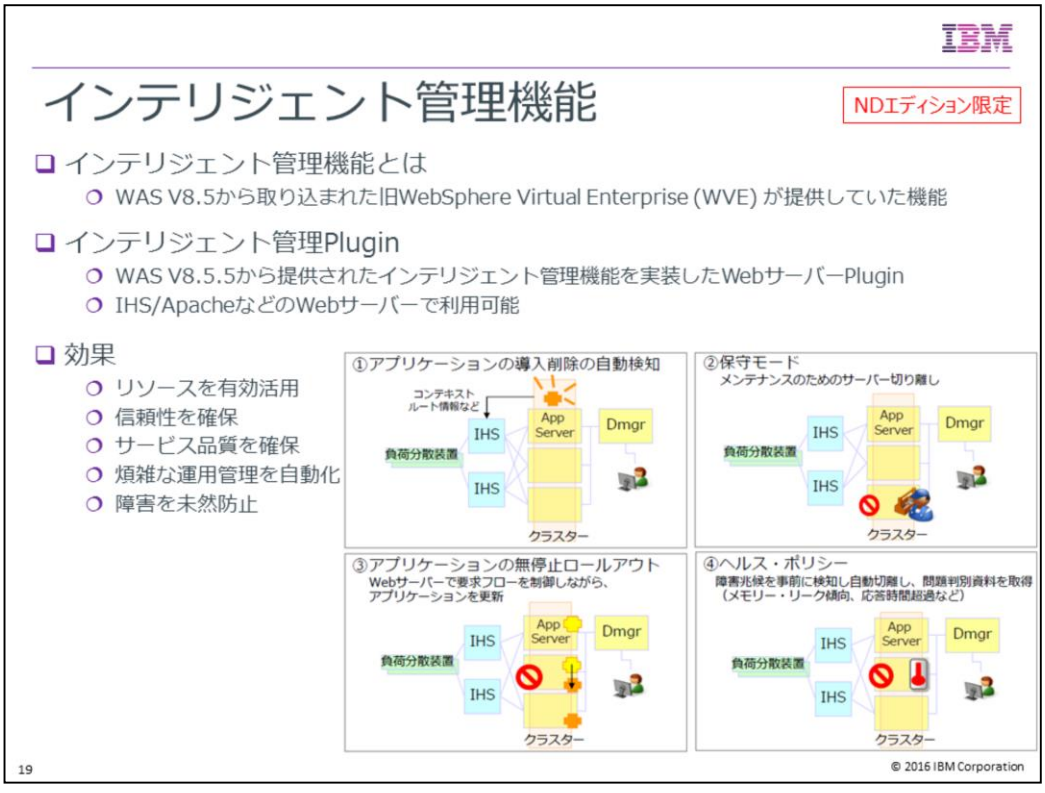
Libertyクラスターを構成すると、Collective Controller はクラスター用のMBeanを提供し、個々のLibertyサーバーに対する操作だけでなく、Libertyクラスターに属するLibertyサーバー全体に対する操作を行うことができるようになります。例えば、Libertyクラスター名の取得、Libertyクラスターの起動/停止や稼働状況の確認、といった操作を行うことができます。

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からランタイム状況などを受け取り、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拡張子「fdb」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。

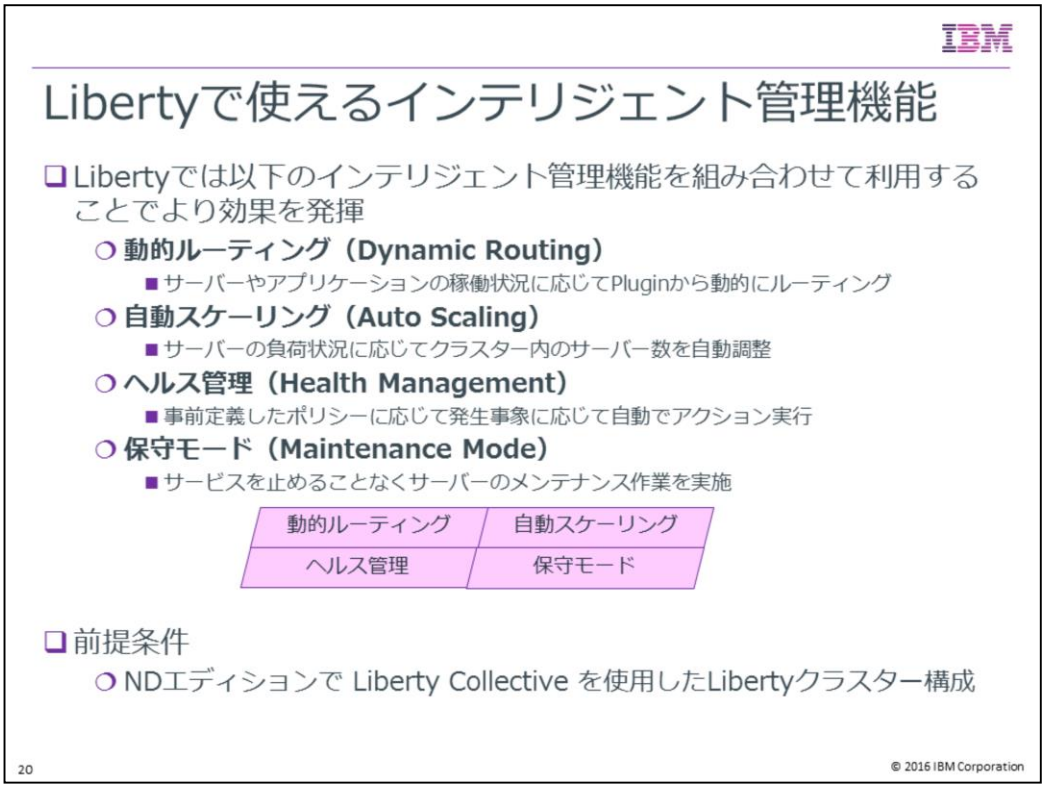


近年のITトレンドの変化に応えるべく、WASのNDエディションではインテリジェント管理機能を提供しています。このインテリジェント管理機能は、WAS V8.5から旧WebSphere Virtual Enterprise (WVE) で提供されていた機能がWAS本体に統合されたものです。

また、WAS V8.5.5からはインテリジェント管理機能を実装したWebサーバーPluginが提供されており、IHS/ApacheなどのWebサーバーで利用することができます。

具体的には、①アプリケーションをクラスター・メンバー上に導入したり削除した際にそれを自動検知する機能、②業務サービス提供中に特定のサーバーをメンテナンスするために保守モードとして切り離す機能、③業務サービス提供中にアプリケーションを更新する際にリクエストの割り振りを制御しながら全てのクラスター・メンバーに反映させる機能、④障害の兆候を事前に検知して問題判別資料を収集するといった特定のアクションを実行させる機能、などが提供されています。

これらの機能を利用することにより、リソースの有効活用、信頼性の確保、サービス品質の確保、煩雑な運用管理の自動化、障害の未然防止、といった効果があります。



Libertyランタイムで利用できるインテリジェント管理機能としては、動的ルーティング、自動スケーリング、ヘルス管理、保守モードがあり、これら単体でも利用できますが、複数を組み合わせることによってより効果を発揮します。

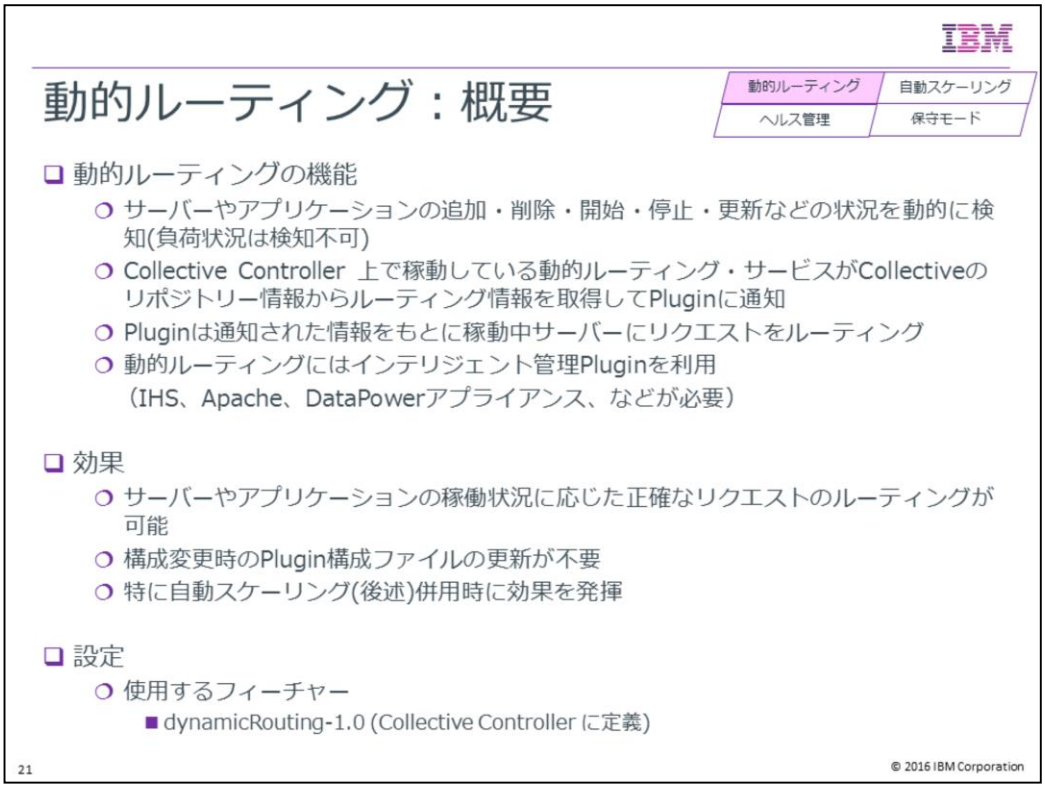
動的ルーティングとは、アプリケーション・サーバーやアプリケーションの起動・停止といった稼働状況を検知して、動的にPluginからのルーティングを制御できるという機能です。

自動スケーリングとは、アプリケーション・サーバーのCPU使用率やメモリ使用状況といった負荷状況に応じてクラスター内のメンバー数を自動で縮小したり拡張したりすることができるという機能です。

ヘルス管理とは、事前定義しておいたポリシーに応じて、発声事象に応じて自動でアクションを行うことができるという機能です。

保守モードとは、業務サービスを停止することなく特定のサーバーのメンテナンス作業を行うために切り離しができるという機能です。

Libertyランタイムにおいてこれらのインテリジェント管理機能を利用する前提条件として、Libertyサーバー側はNDエディションで提供されるLiberty Collectiveを使用してLibertyクラスターを構成しておく必要があります。



動的ルーティングの機能では、アプリケーション・サーバーやアプリケーションの追加・削除・開始・停止・更新などの状況を動的に検知してルーティングすることができます。動的ルーティングの機能では各サーバーの負荷状況までは検知できません。

動作としては、Collective Controller 上で稼動している動的ルーティング・サービスがCollective Controller のリポジトリー情報から各Collective Member の稼働状況を取得してそれをPluginに通知します。

Pluginは通知された情報をもとに起動状態であるアプリケーション・サーバーにリクエストをルーティングします。

動的ルーティングを行うためには、インテリジェント管理Pluginを利用する必要があります。

動的ルーティングの機能を利用すると、稼働状況に応じた正確なリクエストのルーティングが行える他に、構成変更時に手動でPlugin構成ファイルを更新する必要がないといった利点もあります。

また、後述する自動スケーリングの機能と併用することで特に効果を発揮します。

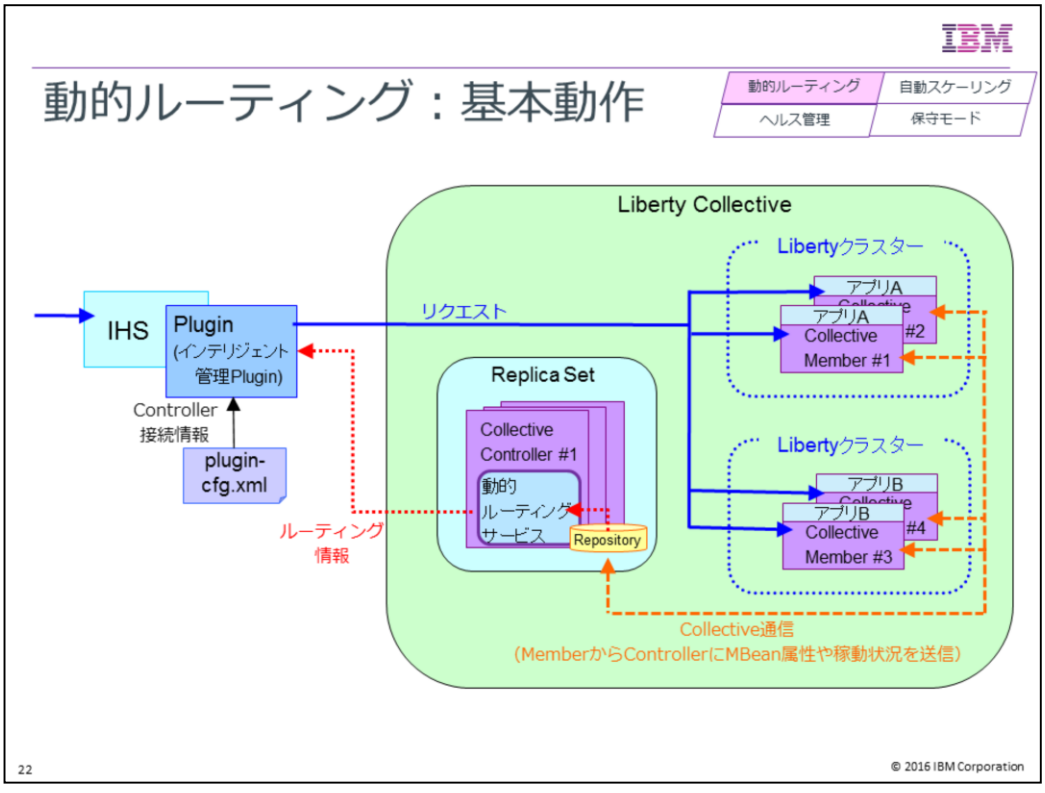
動的ルーティングを有効にするには、dynamicRouting-1.0というLibertyフィーチャーをCollective Controller で定義します。

＜参考＞

Knowledge Center 「Liberty 集合に対する動的ルーティングのセットアップ」

https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ib

m.websphere.wlp.nd.doc/ae/twlp_wve_enabledynrout.html

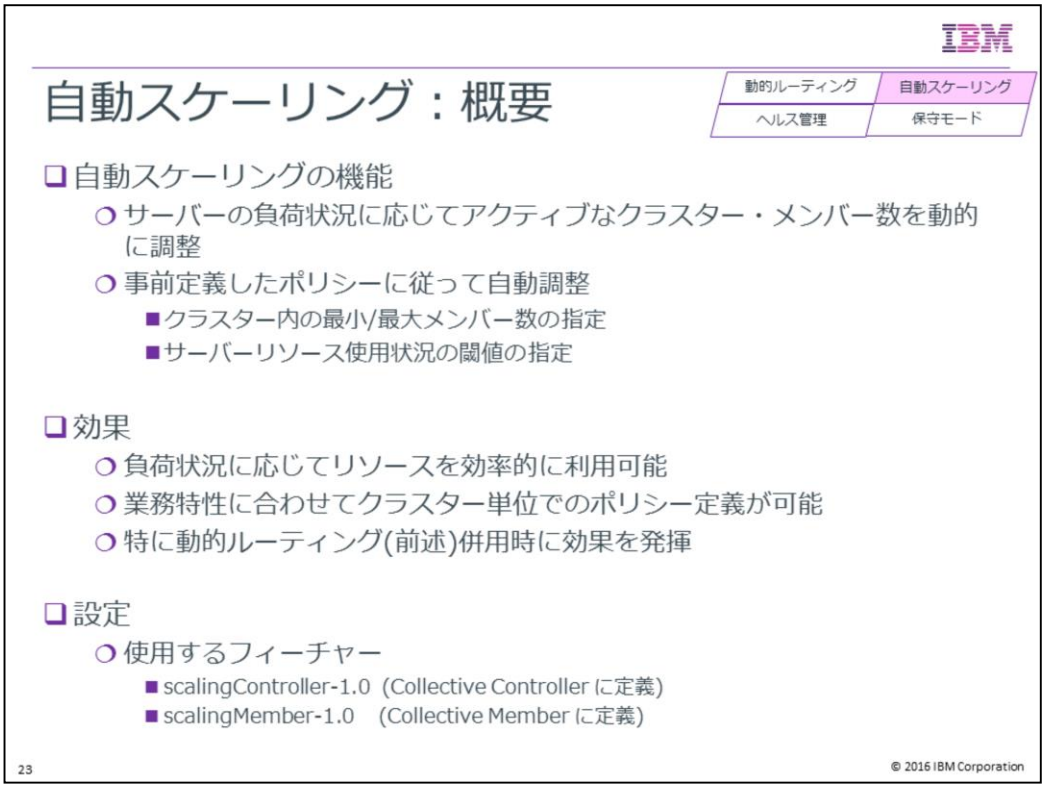


動的ルーティングの基本動作は上図の通りです。

インテリジェント管理PluginはPlugin構成ファイルからCollective Controller の接続情報を取得します。

Collective Controller 上で稼動している動的ルーティング・サービスがCollective Controller のリポジトリー情報から各Collective Member の稼働状況を取得してそれをPluginに通知します。

Pluginは通知された情報をもとに起動状態であるアプリケーション・サーバーにリクエストをルーティングします。



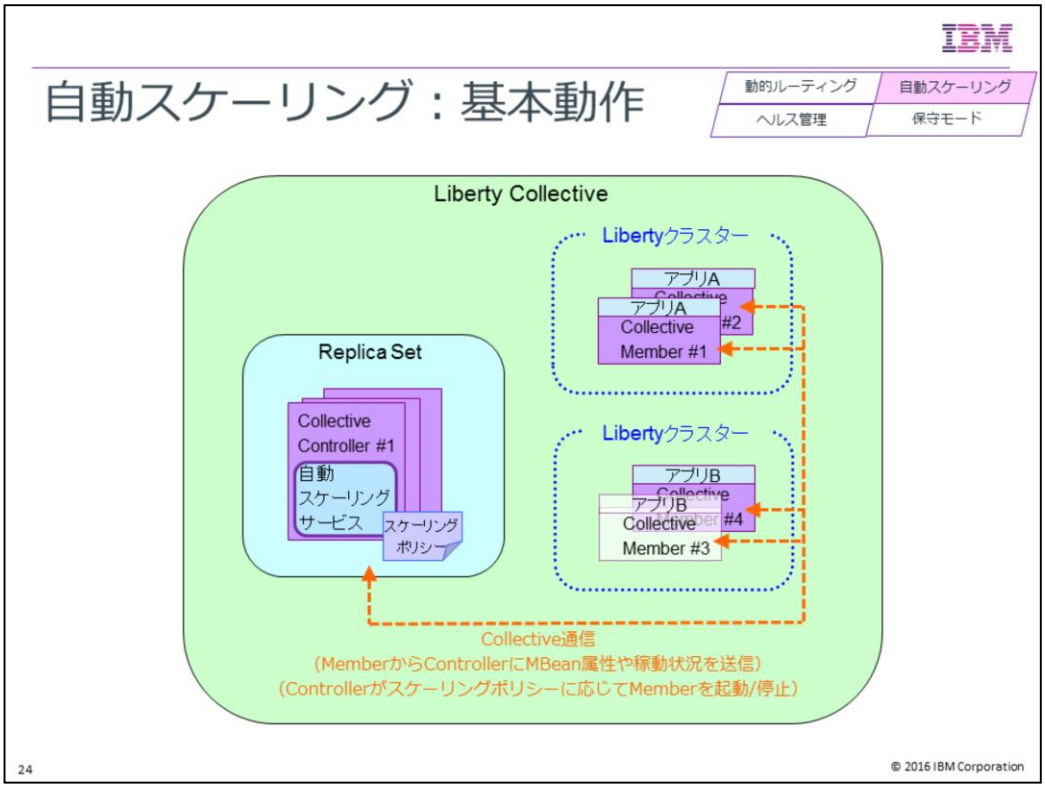
自動スケーリングの機能では、アプリケーション・サーバーの負荷状況に応じてアクティブなクラスター・メンバー数を動的に調整することができます。

事前に、クラスター内の最小/最大メンバー数やサーバーリソース使用状況の閾値などをポリシーとして定義しておくと、そのポリシーに従って自動調整されます。

自動スケーリングの機能を利用すると、負荷状況に応じてリソースを効率的に利用できたり、業務特性に応じてクラスター単位でポリシーを定義することができます。

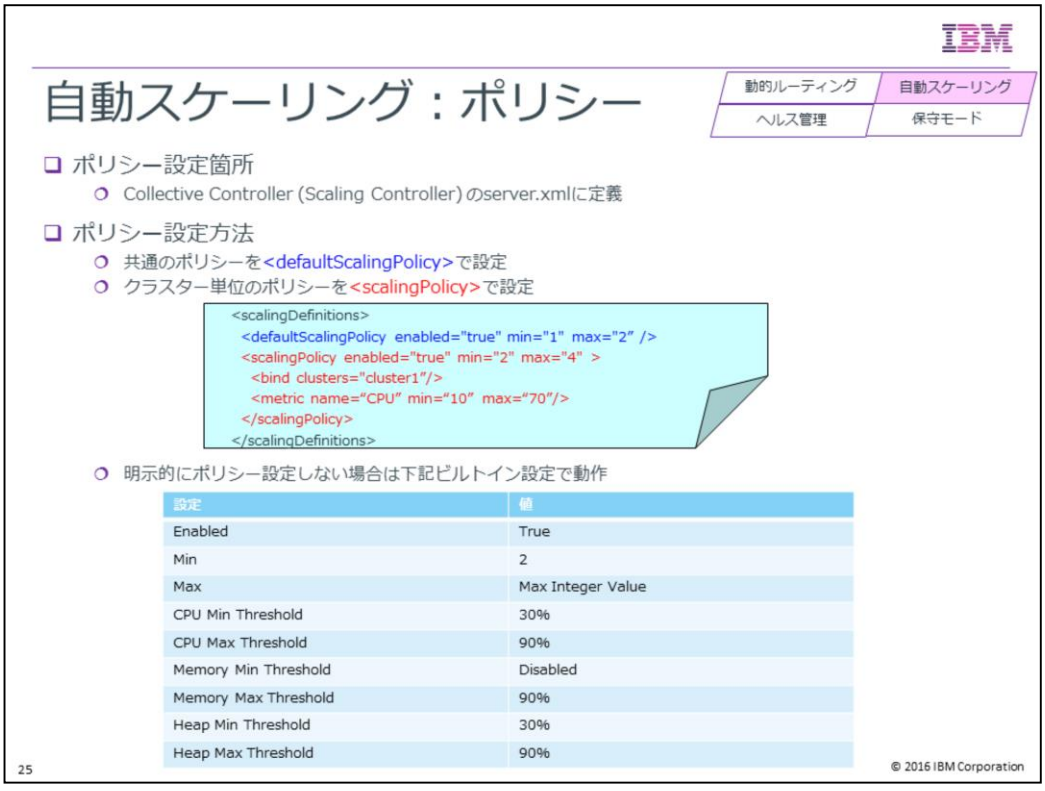
特に、既述の動的ルーティングと併用することでより効果を発揮します。

自動スケーリングを有効にするには、Collective Controller にscalingController-1.0というLibertyフィーチャーを、Collective Member にscalingMember-1.0 というLibertyフィーチャーを定義します。



自動スケーリングの基本動作は上図の通りです。

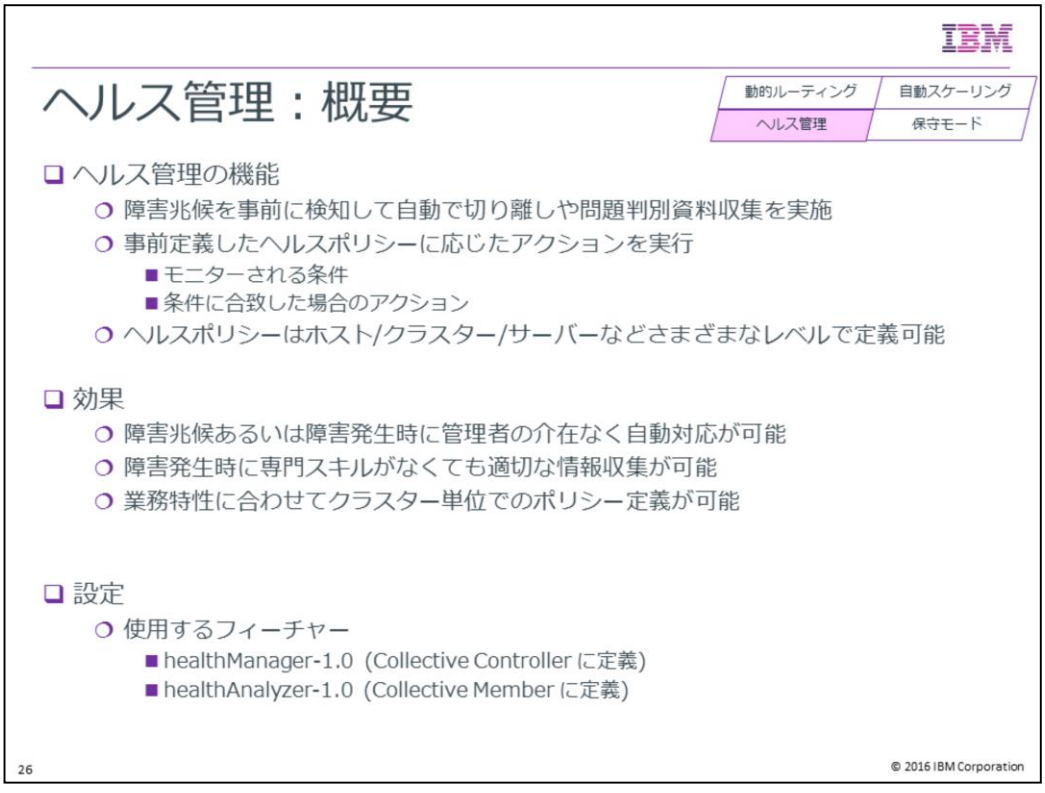
Collective Controller 上では自動スケーリング・サービスが稼動しており、事前定義したスケーリング・ポリシーに応じた動作をします。



スケーリング・ポリシーは、Collective Controller (Scaling Controller) の server.xml上に定義します。

ポリシー設定方法として、全てのクラスターに対して適用したいデフォルトのポリシーを<dafaultScalingPolicy>で設定し、アプリケーション特性やトランザクション量などから特定のLibertyクラスターに対してのみ適用したいポリシーを <scalingPolicy>でLibertyクラスター名を指定して設定します。

自動スケーリングの機能を有効にして、明示的にポリシーを設定していない場合は、上記表のようにビルトイン設定で動作します。



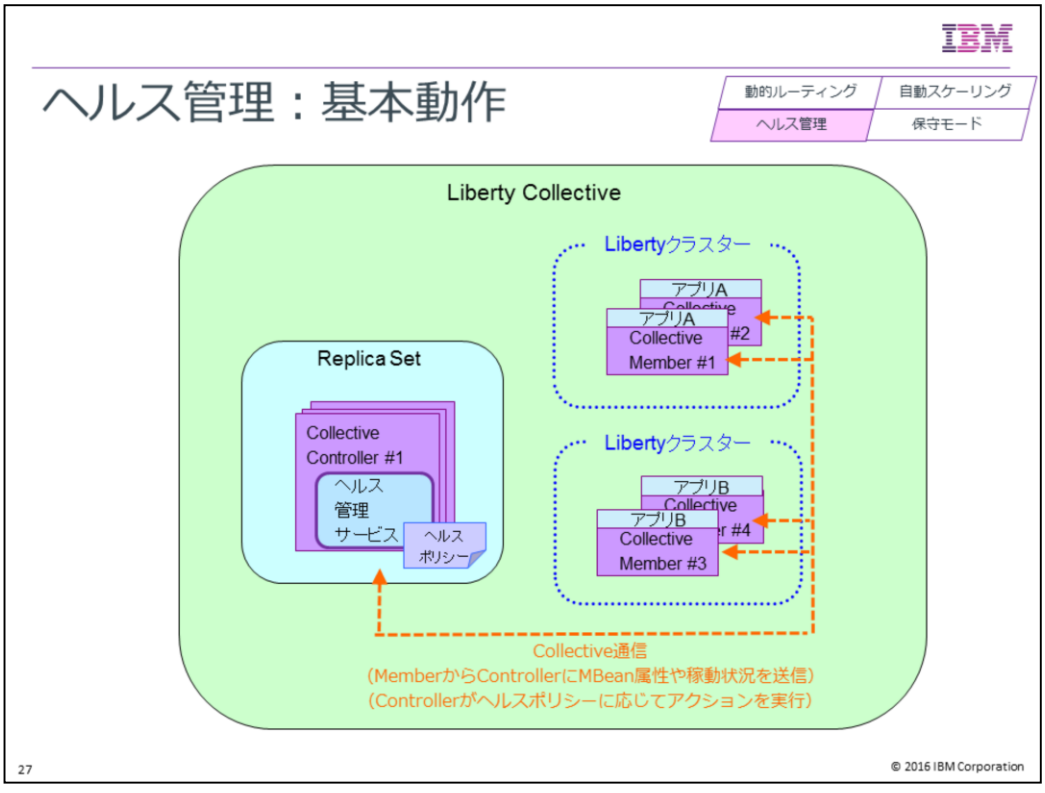
ヘルス管理の機能は、障害の兆候を事前に検知して自動での切り離しや問題判別資料収集を実施することができます。

事前に、モニターされる条件や条件に合致した場合のアクションをヘルスポリシーとして定義しておくと、あとは定義したポリシーに応じて自動で動作するようになります。

ヘルスポリシーは、ホスト、クラスター、サーバーといったレベルで定義することができます。

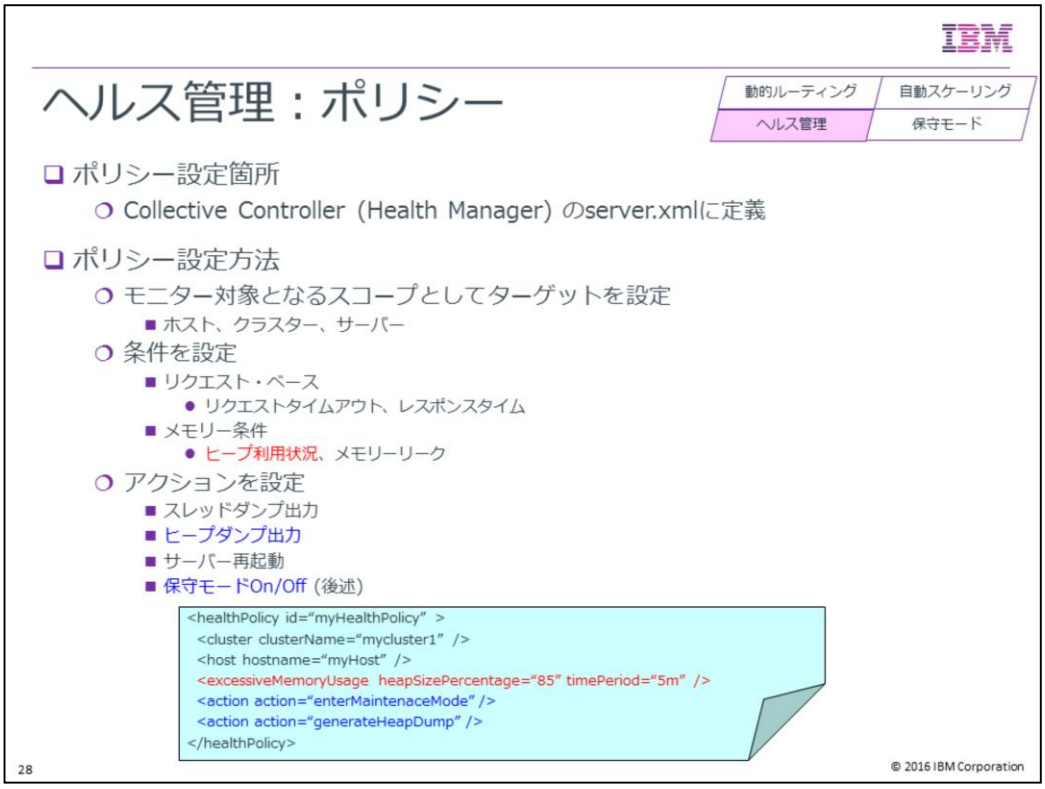
ヘルス管理機能を利用することにより、障害兆候あるいは障害発生時に管理者の介在なく自動対応が可能になり、障害発生時に専門スキルがなくても適切な情報収集が可能になります。

ヘルス管理を有効にするには、Collective Controller にhealthManager-1.0というLibertyフィーチャーを、Collective Member にhealthAnalyzer-1.0 というLibertyフィーチャーを定義します。



ヘルス管理の基本動作は上図の通りです。

Collective Controller 上ではヘルス管理サービスが稼動しており、事前定義したヘルスポリシーに応じた動作をします。

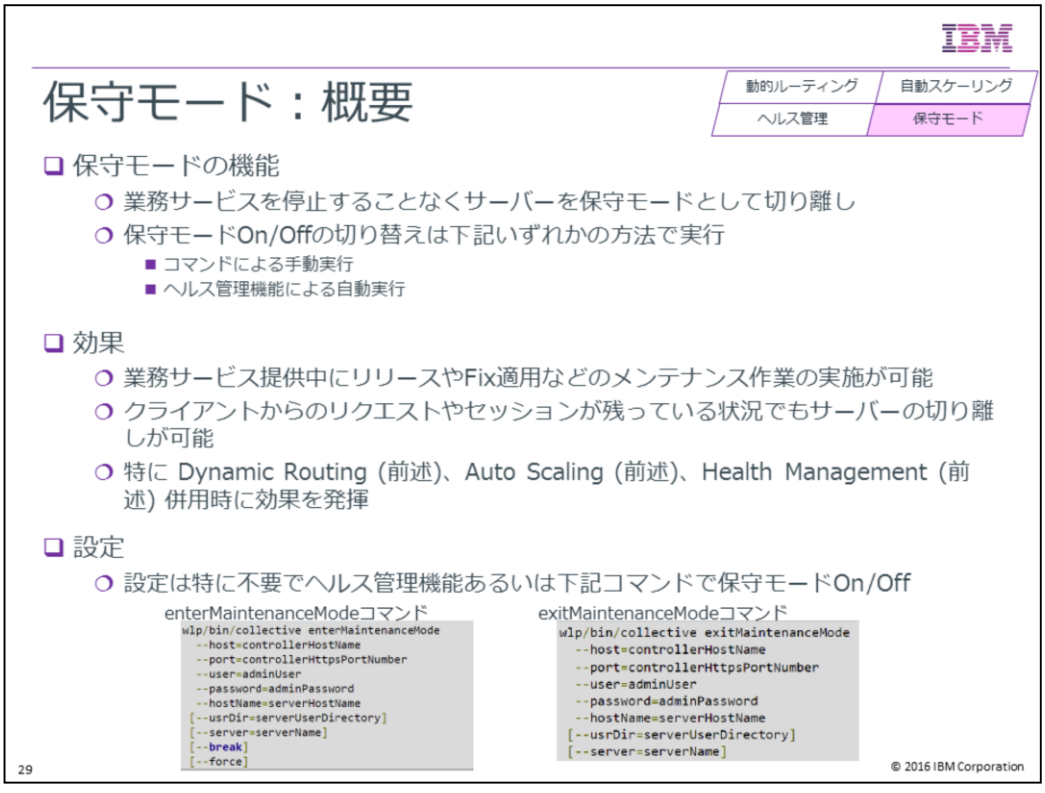


ヘルス・ポリシーは、Collective Controller (Health Manager) のserver.xml上に定義します。

ポリシー設定方法として、モニター対象となるスコープとして、ホスト、クラスター、サーバー、いずれかのターゲットを設定します。

条件として、リクエストタイムアウトやレスポンスタイムといったリクエスト・ベース、ヒープ利用状況やメモリーリークといったメモリー条件などを設定します。

アクションとして、スレッドダンプ出力、ヒープダンプ出力、サーバー再起動、保守モードOn/Offなどを設定します。

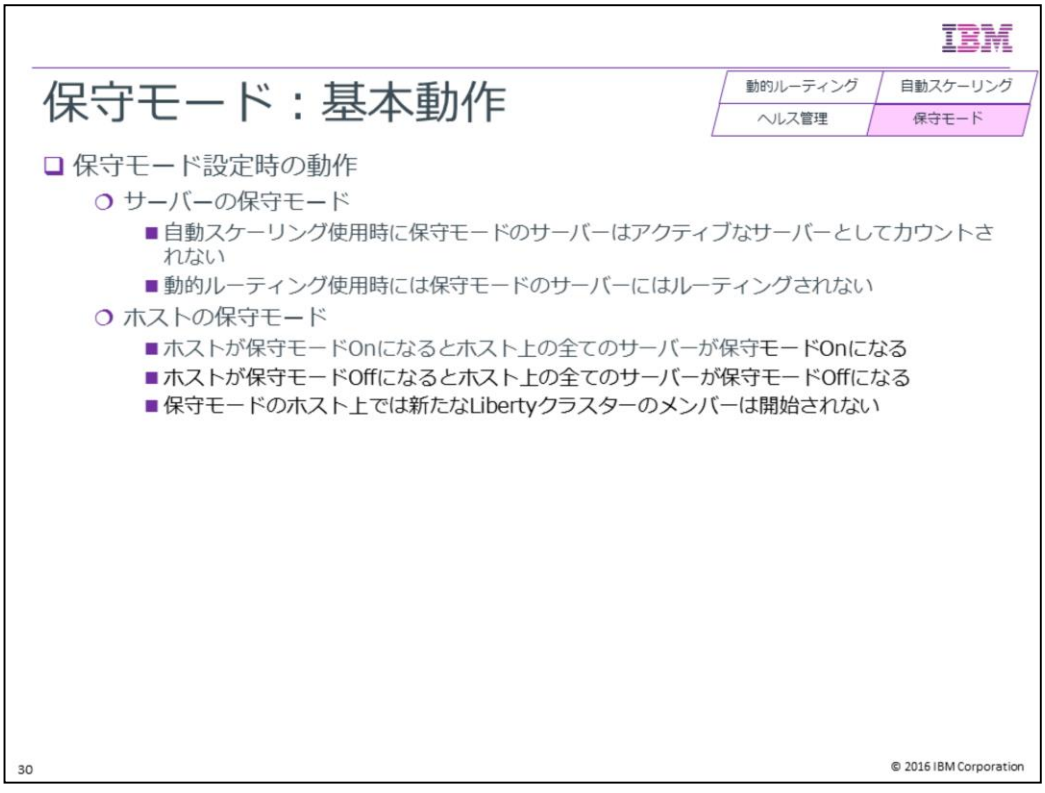


保守モードの機能は、業務サービスを停止することなくLibertyサーバーを保守モードとして切り離すことができます。

保守モードOn/Offの切り替え方法は、コマンドによる手動実行か既述のヘルス管理機能のアクションとしての自動実行になります。

保守モードを利用することにより、業務サービス提供中であっても、リリースやFix適用などといったメンテナンス作業を行うことができます。また、クライアントからのリクエストやセッションが残っている状況であってもLibertyサーバーを切り離すことができます。

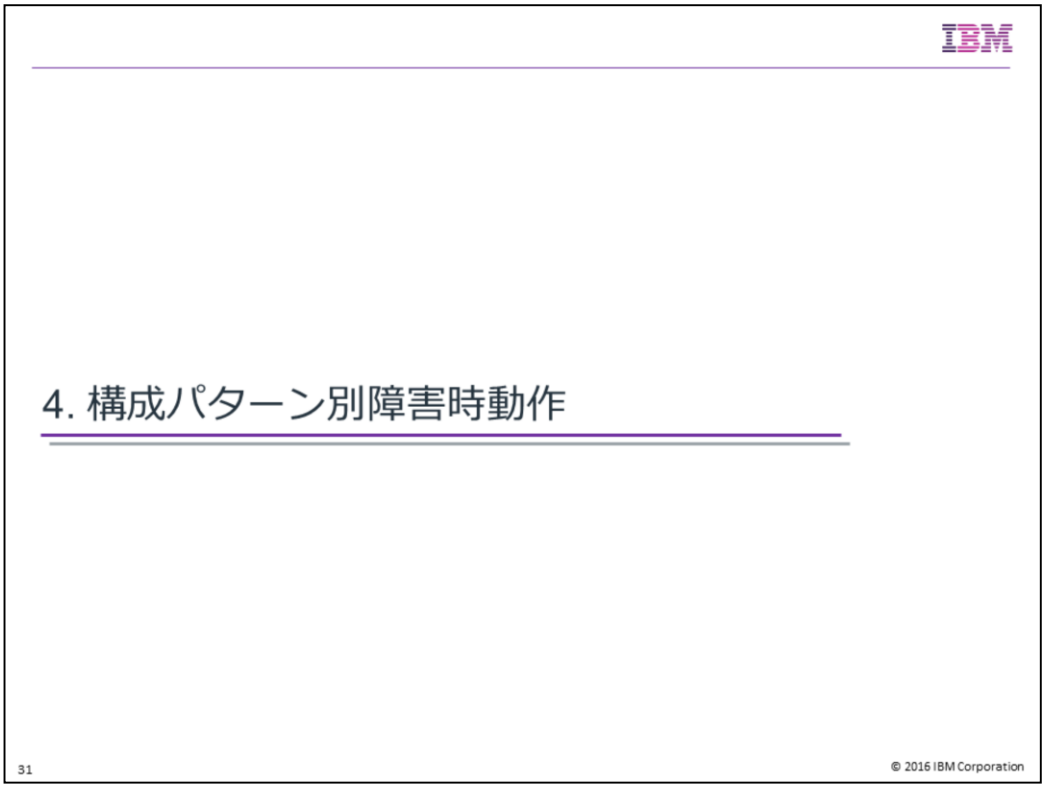
保守モードOn/Offの切り替えには、ヘルス管理機能のアクションとして実行するか、「enterMaintenanceMode」コマンドや「exitMaintenanceMode」コマンドを使用します。



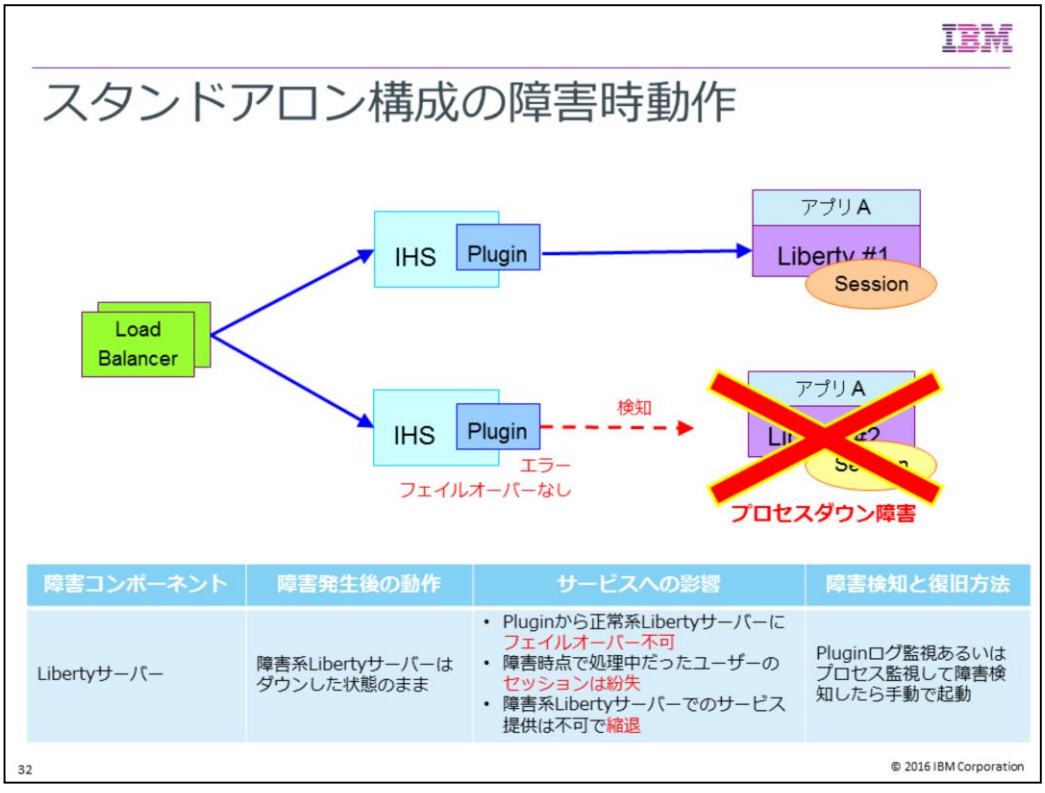
保守モード設定時の基本動作について説明します。

Libertyサーバーの保守モードに関して、自動スケーリング使用時に保守モードOn状態のサーバーはアクティブなサーバーとしてカウントされません。また、動的ルーティング使用時に保守モードOn状態のサーバーにはルーティングされません。

ホストの保守モードに関して、ホストが保守モードOn状態になると、そのホスト上に存在する全てのLibertyサーバーが保守モードOnの状態となり、ホストが保守モードOff状態になると、そのホスト上に存在する全てのLibertyサーバーが保守モードOffの状態となります。また、保守モードOn状態のホスト上では新たなLibertyクラスターのメンバーはアクティブになりません。



この章では、Libertyの構成パターン別に、例を挙げて障害時動作を説明します。



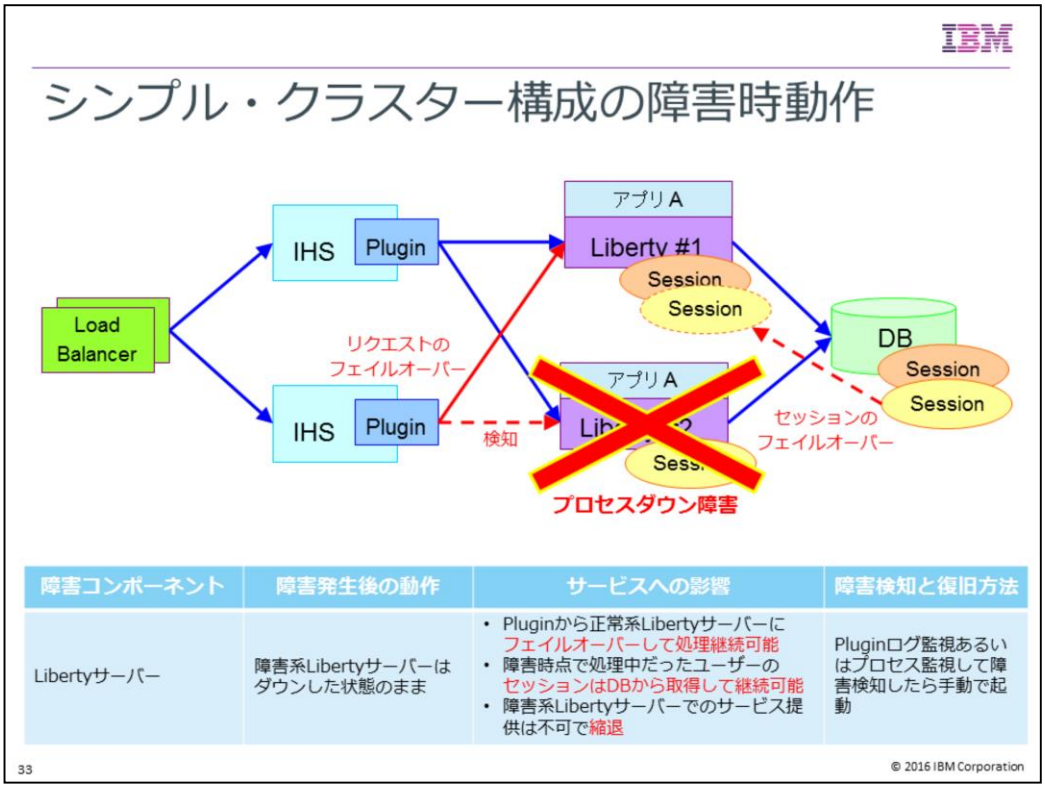
Libertyランタイムのスタンドアロン構成の障害時動作です。

ここでは、例として、2台のうち1台のLibertyサーバーにプロセスダウン障害が発生した場合の動作です。

Libertyサーバーにプロセスダウン障害が発生すると、障害したLibertyサーバーはそのままダウン状態となります。

クライアントからのリクエストがPluginまで届くと、Pluginはルーティング先としては障害系サーバーしか認識していないため、リクエストのフェイルオーバーはできません。また、障害発生時点で処理中だったユーザーのセッションも引き継がれずに紛失となり、クライアントが再度リクエストを送信してきても新規リクエストとして扱われます。障害系のLibertyサーバーはダウンしたままでサービス提供はできませんので、システム全体としては縮退での運用となります。

Libertyサーバーがダウンしたことは、Pluginログ監視あるいはLibertyサーバーのプロセス監視によって検知することができるため、対応としては手動で再起動することになります。



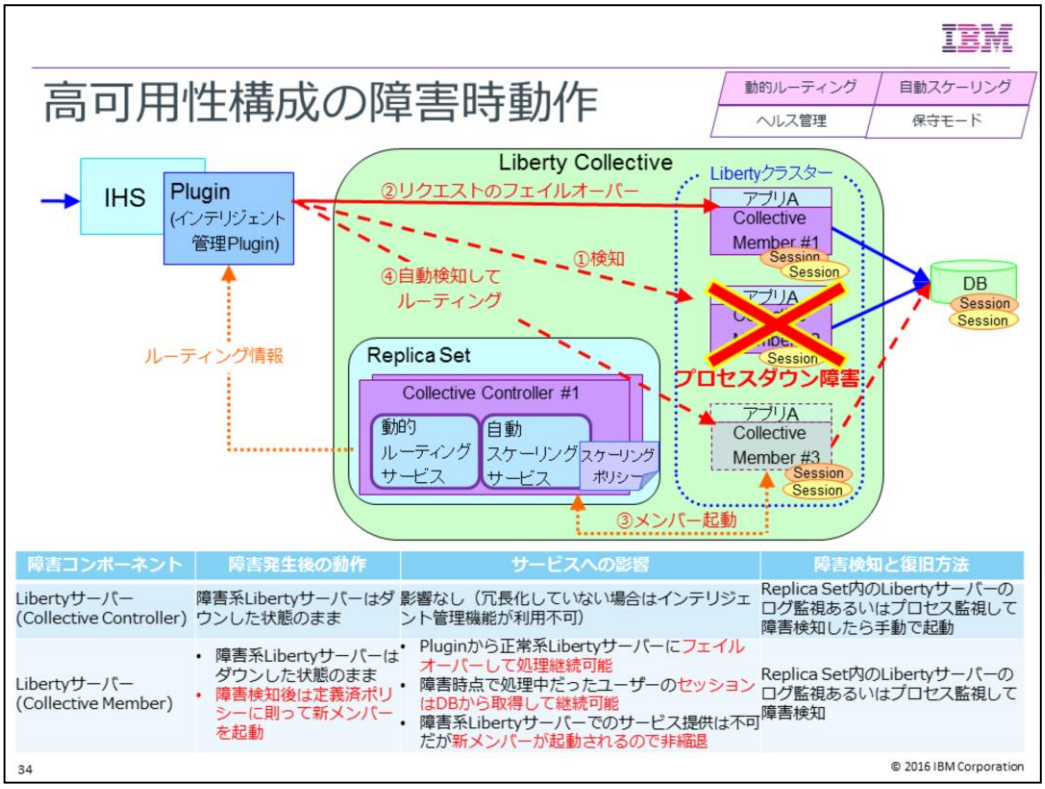
Libertyランタイムのシンプル・クラスター構成の障害時動作です。

ここでは、例として、2台のうち1台のLibertyサーバーにプロセスダウン障害が発生した場合の動作です。

Libertyサーバーにプロセスダウン障害が発生すると、障害したLibertyサーバーはそのままダウン状態となります。

クライアントからのリクエストがPluginまで届くと、Pluginは障害を検知して正常系Libertyサーバーにリクエストのフェイルオーバーをします。また、障害発生時点で処理中だったユーザーのセッションはセッション・パーシスタンス機能によって引き継がれ、クライアントが再度リクエストを送信してきてもセッションのフェイルオーバーによって既存リクエストとして処理が継続されます。障害系のLibertyサーバーはダウンしたままでサービス提供はできませんので、システム全体としては縮退での運用となります。

Libertyサーバーがダウンしたことは、Pluginログ監視あるいはLibertyサーバーのプロセス監視によって検知することができるため、対応としては手動で再起動することになります。

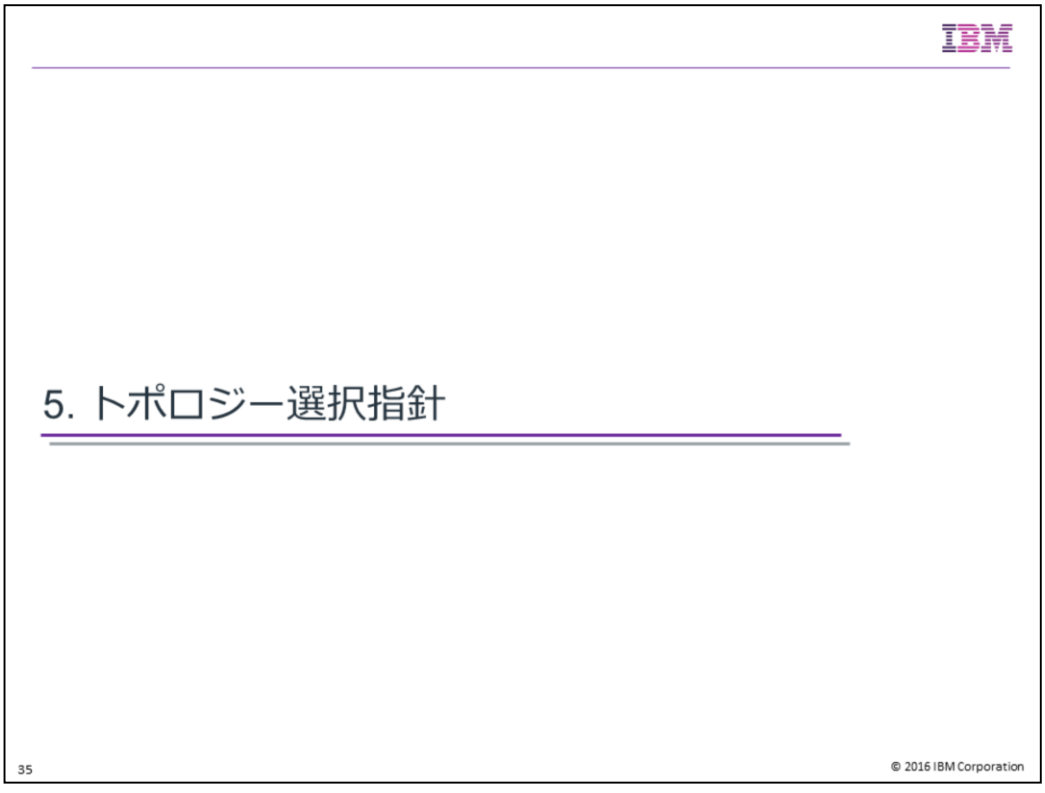


Libertyランタイムの高可用性構成の障害時動作です。

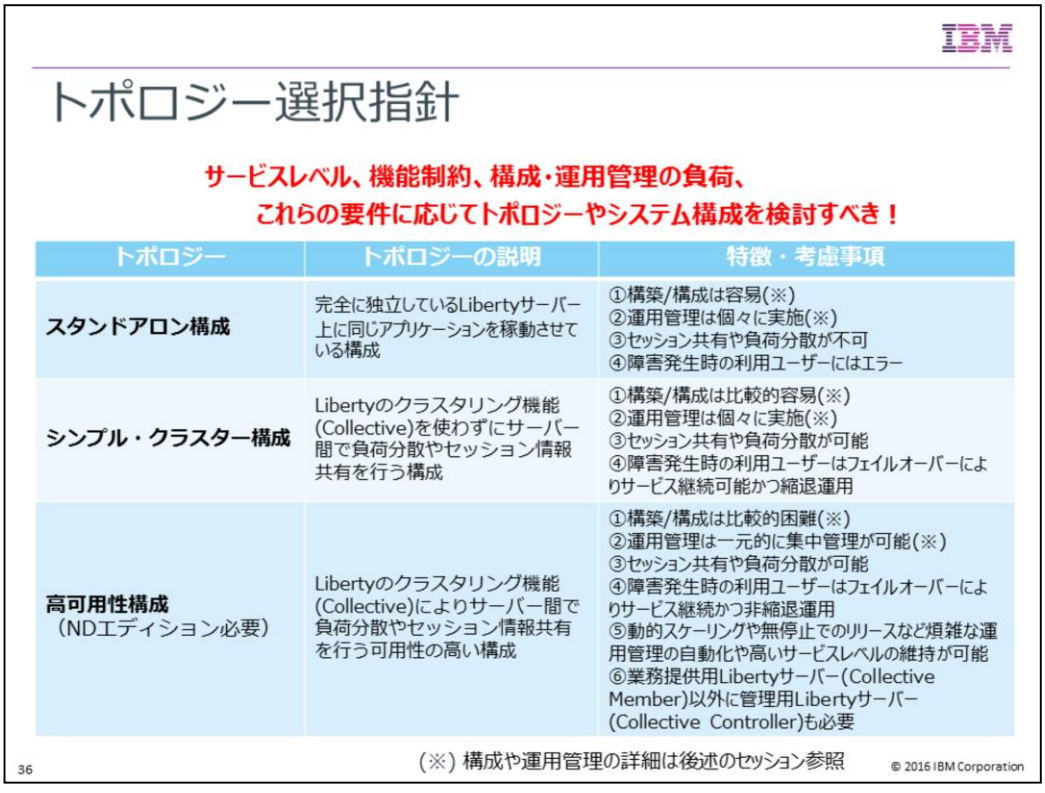
ここでは、例として、Collective Controller であるLibertyサーバーにプロセスダウン障害が発生した場合の動作とCollective Member であるLibertyサーバーにプロセスダウン障害が発生した場合の動作です。

Collective Controller としてのLibertyサーバーにプロセスダウン障害が発生すると、障害したLibertyサーバーはそのままダウン状態となります。Collective Controller を冗長化していない場合あるいは稼動しているCollective Controller が存在しない場合は、インテリジェント管理機能が利用できない状態となります。Libertyサーバーがダウンしたことは、Libertyサーバーのログ監視あるいはプロセス監視によって検知することができるため、対応としては手動で再起動することになります。

Collective Member としてのLibertyサーバーにプロセスダウン障害が発生すると、障害したLibertyサーバーはそのままダウン状態となります。インテリジェント管理機能の自動スケーリングを利用している場合、障害検知後にポリシーに従って新たなメンバーが起動されます。クライアントからのリクエストがPluginまで届くと、Pluginは障害を検知するか動的ルーティングによってルーティング情報を入手し、正常系Libertyサーバーにリクエストのフェイルオーバーをします。また、障害発生時点で処理中だったユーザーのセッションはセッション・パーシスタンス機能によって引き継がれ、クライアントが再度リクエストを送信してきてもセッションのフェイルオーバーによって既存リクエストとして処理が継続されます。障害系のLibertyサーバーはダウンしたままですが、自動スケーリング機能によって新たに起動されるLibertyサーバーでもサービス提供を開始しますので、システム全体としては縮退しない運用となります。Libertyサーバーがダウンしたことは、Libertyサーバーのログ監視あるいはプロセス監視によって検知することができます。



この章では、Libertyのトポロジー選択指針をまとめます。



トポロジー選択指針

サービスレベル、機能制約、構成・運用管理の負荷、
これらの要件に応じてトポロジーやシステム構成を検討すべき！

トポロジー	トポロジーの説明	特徴・考慮事項
スタンドアロン構成	完全に独立しているLibertyサーバー上に同じアプリケーションを稼動させている構成	①構築/構成は容易(※) ②運用管理は個々に実施(※) ③セッション共有や負荷分散が不可 ④障害発生時の利用ユーザーにはエラー
シンプル・クラスター構成	Libertyのクラスタリング機能(Collective)を使わずにサーバー間で負荷分散やセッション情報共有を行う構成	①構築/構成は比較的容易(※) ②運用管理は個々に実施(※) ③セッション共有や負荷分散が可能 ④障害発生時の利用ユーザーはフェイルオーバーによりサービス継続可能かつ縮退運用
高可用性構成 （NDエディション必要）	Libertyのクラスタリング機能(Collective)によりサーバー間で負荷分散やセッション情報共有を行う可用性の高い構成	①構築/構成は比較的困難(※) ②運用管理は一元的に集中管理が可能(※) ③セッション共有や負荷分散が可能 ④障害発生時の利用ユーザーはフェイルオーバーによりサービス継続かつ非縮退運用 ⑤動的スケーリングや無停止でのリリースなど煩雑な運用管理の自動化や高いサービスレベルの維持が可能 ⑥業務提供用Libertyサーバー(Collective Member)以外に管理用Libertyサーバー(Collective Controller)も必要

(※) 構成や運用管理の詳細は後述のセッション参照

当セッションでは、Libertyランタイム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、NDエディションで提供されるLiberty Collective を使った高可用性構成を紹介しました。上記表には、これまで説明してきた各トポロジーにおける説明や特徴、考慮事項をまとめました。

Libertyランタイムを採用したシステムにおいてトポロジーを検討する際は、システムに求められるサービスレベル、機能制約、構成・運用管理の負荷などを考慮する必要があります。

上記表に記載された①構築/構成と②運用管理に関する詳細は、後述の各該当セッションの資料をご参照ください。

まとめ

- 基本機能
  - IHS-Plugin-Liberty 基本構成　・・・LibertyではPlugin構成ファイル生成方法が特有
  - セッション・パーシスタンス　・・・LibertyではDBとWXSをサポート
- 高可用性機能
  - Liberty Collective 機能　・・・Collective Controller と Collective Member で構成
  - インテリジェント管理機能
    - 動的ルーティング　・・・稼働状況に応じた動的なルーティングが可能
    - 自動スケーリング　・・・負荷状況に応じたクラスターメンバー数の自動調整が可能
    - ヘルス管理　・・・ポリシーに応じた障害兆候の検知や自動での対応が可能
    - 保守モード　・・・業務サービス提供中における特定サーバーの切り離しが可能
- 構成パターン
  - スタンドアロン構成　・・・独立しているLibertyサーバー上に同じアプリケーションを稼動させる構成
  - シンプル・クラスター構成　・・・Liberty Collective を使わないクラスター構成
  - 高可用性構成　・・・Liberty Collective やインテリジェント管理機能を使ったクラスター構成

WebSphere Application Server Liberty