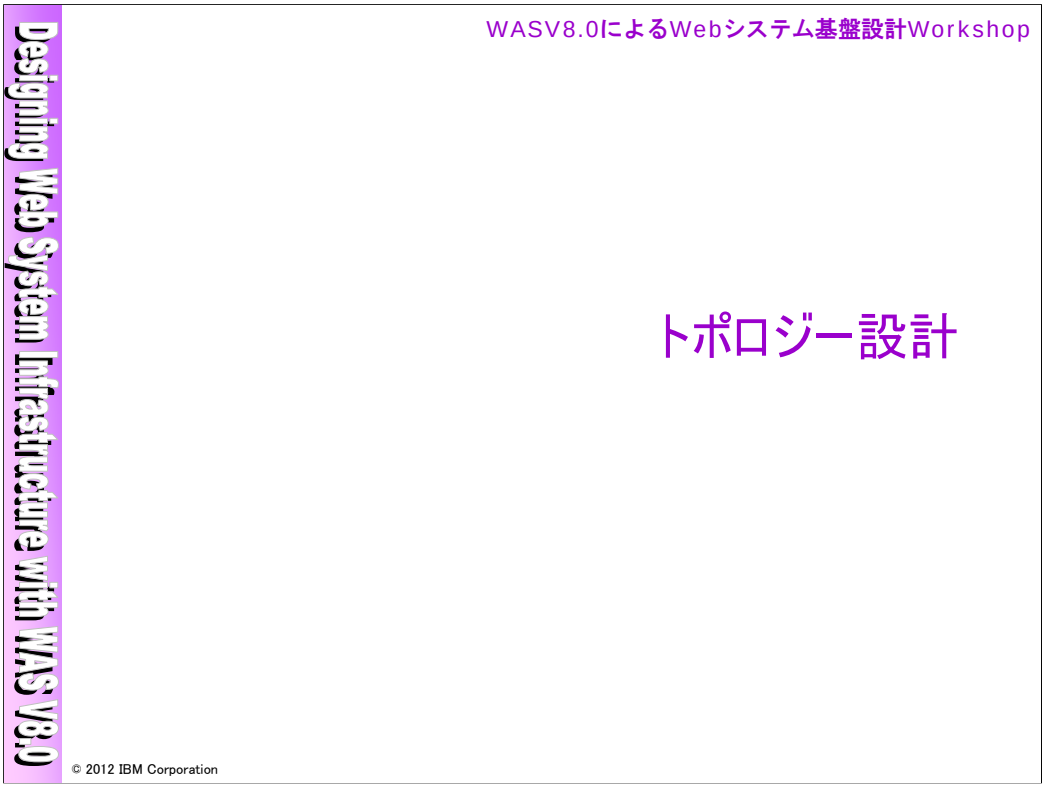

Designing Web System Infrastructure with WAS V8.0
WASV8.0によるWebシステム基盤設計Workshop
トポロジー設計
© 2012 IBM Corporation

Designing Web System Infrastructure with WAS V8.0

当セッションの目的

- ミッション・クリティカルな環境におけるWASトポロジー設計に必要な知識の習得
  - Webシステムの各コンポーネントの役割と、各コンポーネントが提供する高可用性・高パフォーマンス機能を理解する。
  - パフォーマンス、可用性、セキュリティー要件、運用要件に基づいたWebシステム・トポロジーの選択基準、メリット・デメリットを理解する。

© 2012 IBM Corporation

Designing Web System Infrastructure with WAS V8.0

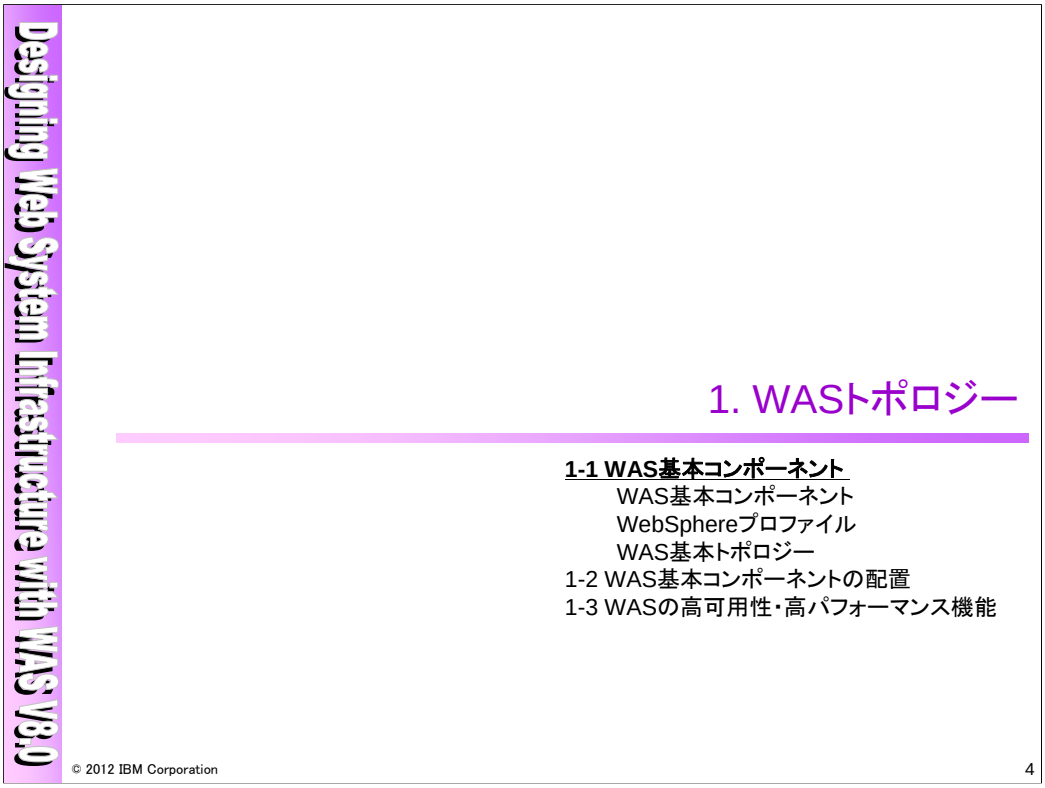
Agenda

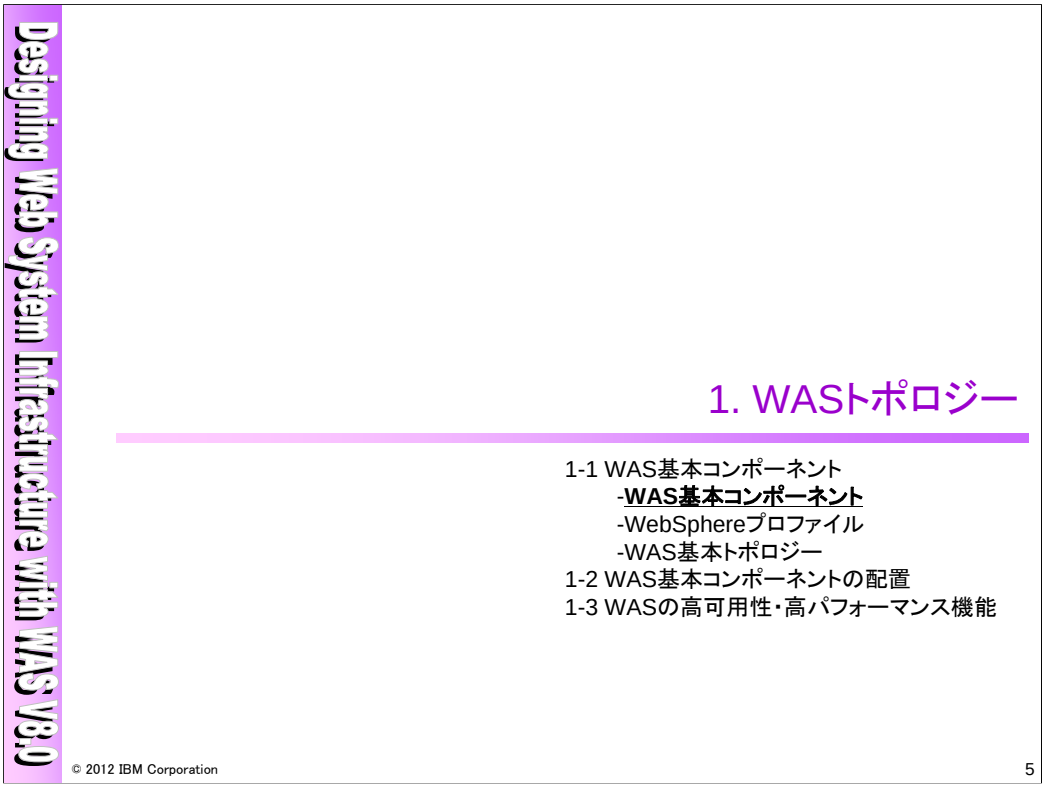
1. WASトポロジー

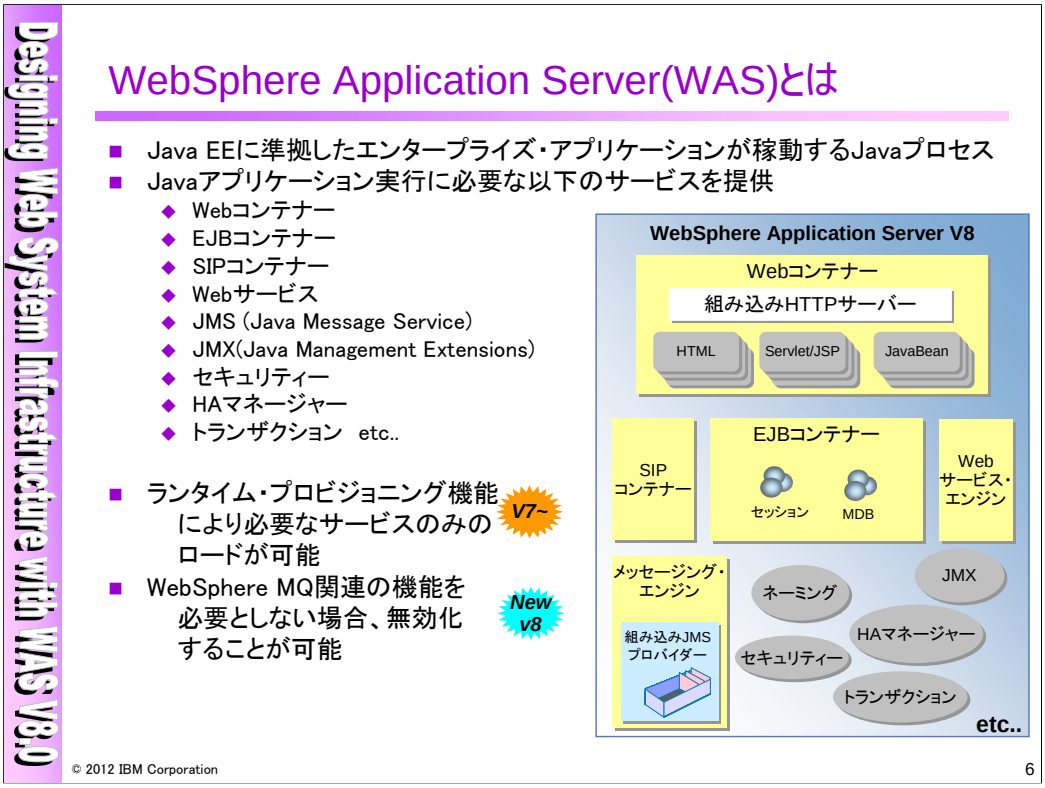
2. 負荷分散と障害検知

まとめ・参考文献

© 2012 IBM Corporation

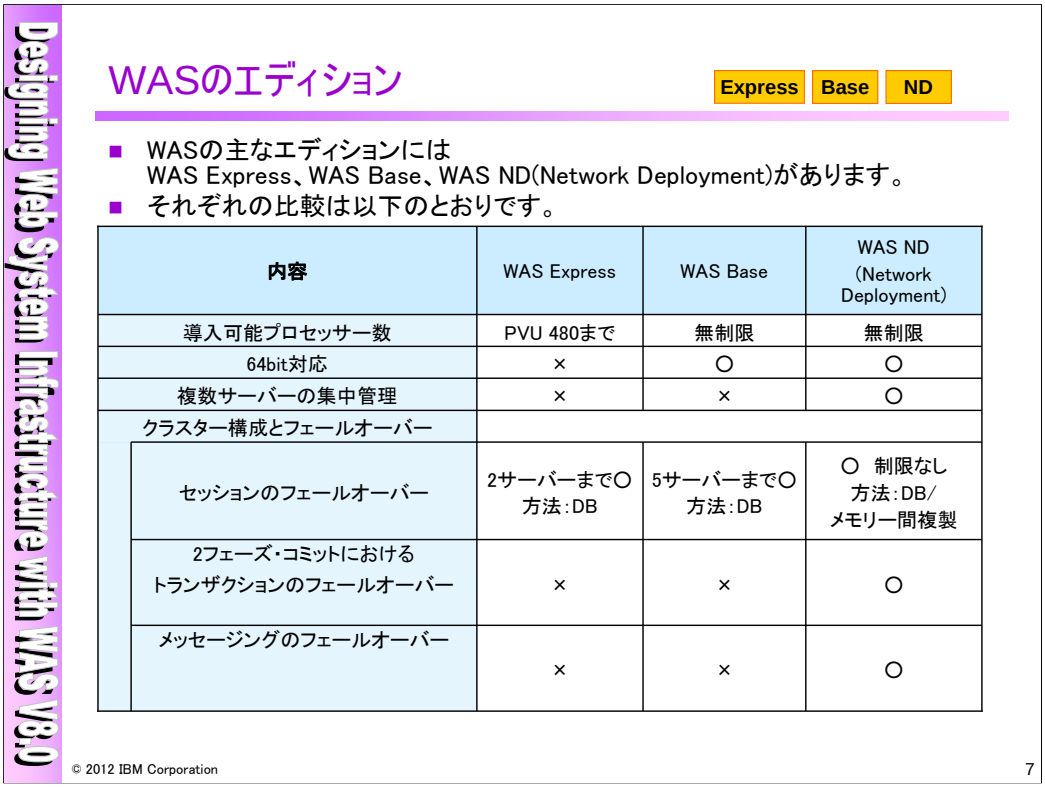

Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
WAS基本コンポーネント
WebSphereプロファイル
WAS基本トポロジー
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
4


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
-WAS基本コンポーネント
-WebSphereプロファイル
-WAS基本トポロジー
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
5

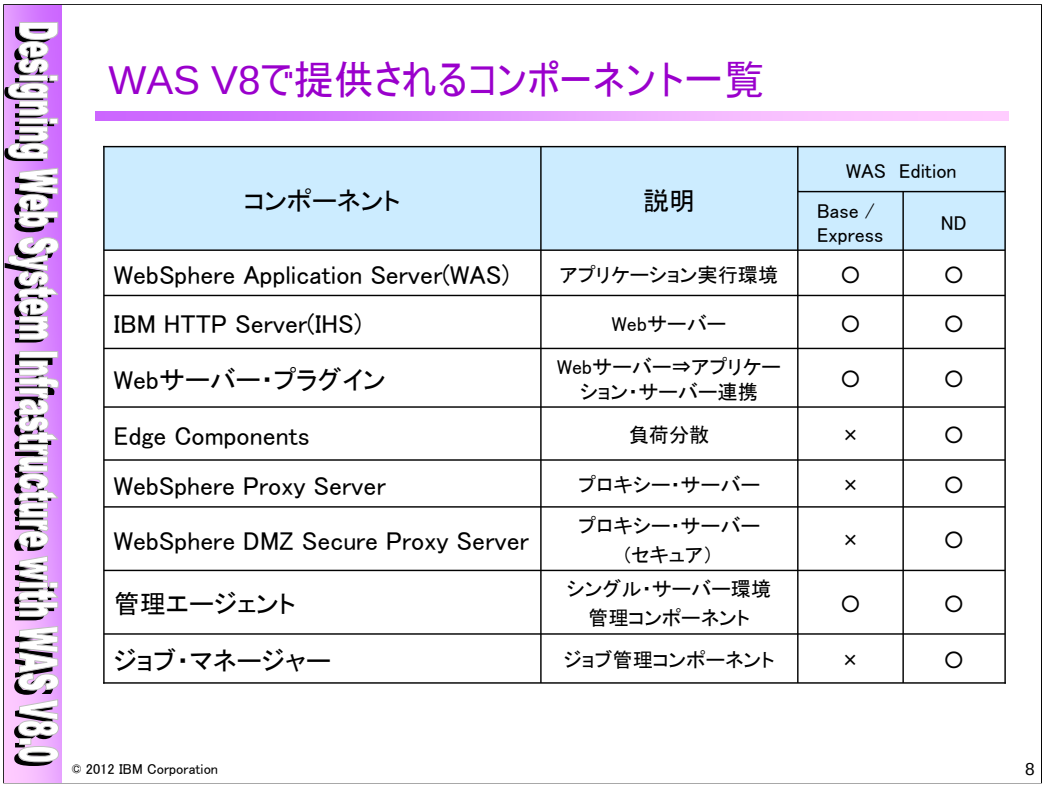


Java EEアプリケーションはサーブレットやJSP, EJBなどによって構成されます。WebSphere Application Serverはこうした Java EE準拠のアプリケーションを稼動させるためのJavaプロセスで、サーブレットやJSPのランタイム環境であるWebコンテナー、EJBランタイム環境であるEJBコンテナーなど、Java EEの仕様に合わせて複数のサービスを提供します。

数多くサービスを提供しますが、例えばEJBを使用しないアプリケーションの場合はEJBコンテナー・サービスが不要であるなど、使用するアプリケーションによって不要なサービスは少なからず存在してしまいます。こうしたことに対し、V7.0よりランタイム・プロビジョニングと呼ばれる、アプリケーション・サーバー上のアプリケーションが必要とする、最小限のサービス・コンポーネントのみをロードし、不要なサービスのロードを抑止する機能が追加されました。また、V8.0よりWebSphere MQに関連する機能(JMSプロバイダー機能)を不要な場合無効化できる機能が提供され、これまでのバージョンよりもさらに不要なサービスによるリソース消費を抑えることが可能になりました。

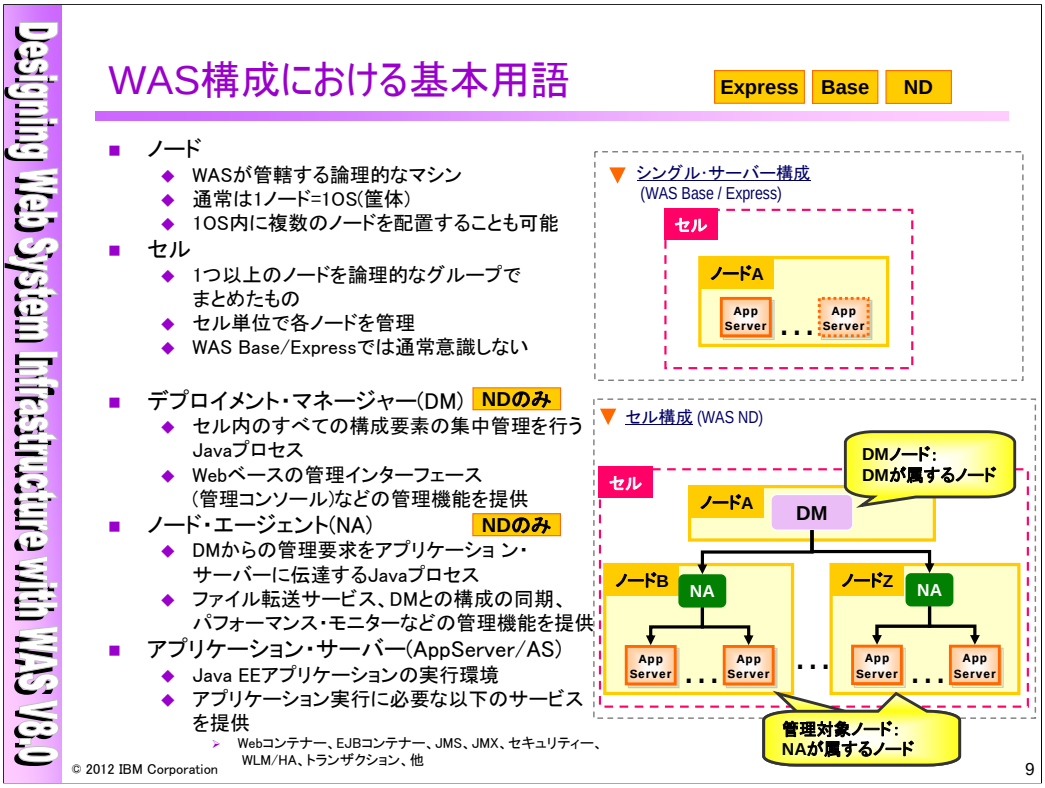


WASの主なエディションにはWAS Express、WAS Base、WAS ND(Network Deployment)があります。
それぞれの比較は表のとおりです。



コンポーネント	説明	WAS Edition	
		Base / Express	ND
WebSphere Application Server(WAS)	アプリケーション実行環境	○	○
IBM HTTP Server(IHS)	Webサーバー	○	○
Webサーバー・プラグイン	Webサーバー⇒アプリケーション・サーバー連携	○	○
Edge Components	負荷分散	×	○
WebSphere Proxy Server	プロキシー・サーバー	×	○
WebSphere DMZ Secure Proxy Server	プロキシー・サーバー（セキュア）	×	○
管理エージェント	シングル・サーバー環境管理コンポーネント	○	○
ジョブ・マネージャー	ジョブ管理コンポーネント	×	○

WAS V8.0が提供するコンポーネントは上記の8種類あります。 これらのコンポーネントはエディションによって同梱される・されないがありますので、システム構成を考える場合にはご注意いただく必要があります。



WASの基本構成をご説明します前に、WAS構成の基本用語であるセル/ノード/デプロイメント・マネージャー/ノード・エージェント/アプリケーション・サーバーについてまずご説明いたします。

セル

セルは、WAS の1つ以上のノードを論理的なグループにまとめたものです。

WAS ND構成で使用されるものであり、WAS Base(Express)では通常は意識しません。

ノード

ノードは、WASが管轄する論理的なマシンです。多くの場合、ネットワーク上で識別可能な IP ホスト・アドレスを持つ物理的なマシンに一致します。

NDの複数台構成では、あるノードをあるセル配下に構成することを、「統合する」といいます。各ノードで実行するアプリケーション・サーバーの管理は、DMがノード・エージェントと通信することにより行います。

デプロイメント・マネージャー(DM)

DMは、セル内のすべての構成要素の集中管理を行います。DMは、それ自体が専用のアプリケーション・サーバー上で実行されていて、Webベースの管理インターフェース(管理コンソール)などの管理機能を提供しています。

ノード・エージェント

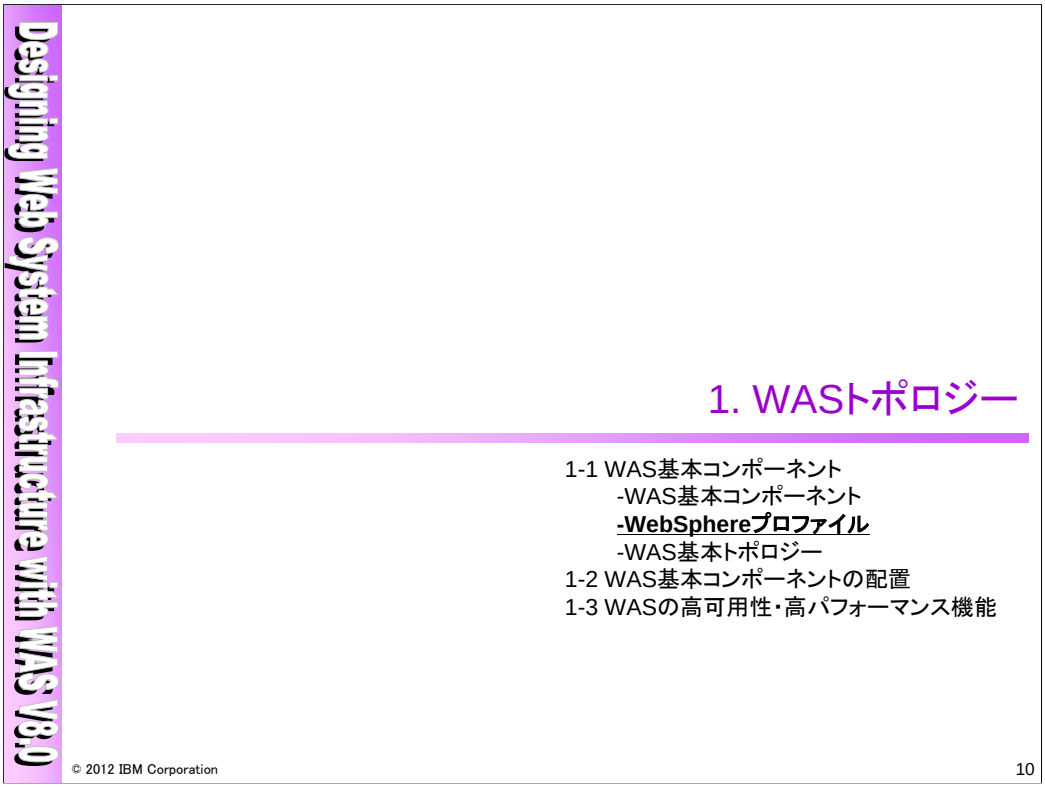
ノード・エージェントはDMからの管理要求をアプリケーション・サーバーに伝達するJavaプロセスで、セルに統合されているすべてのノードで実行されます。また、ファイル転送サービス、構成の同期、およびパフォーマンス・モニターなどの管理機能を提供します。ノード・エージェントは単なる管理エージェントで、エンタープライズ・アプリケーションが提供する機 能には関与しません。ノード・エージェントは、ノードをセルに追加すると、自動的に作成されます。逆に、セルに追加するまではノード・エージェントは作成 されません。

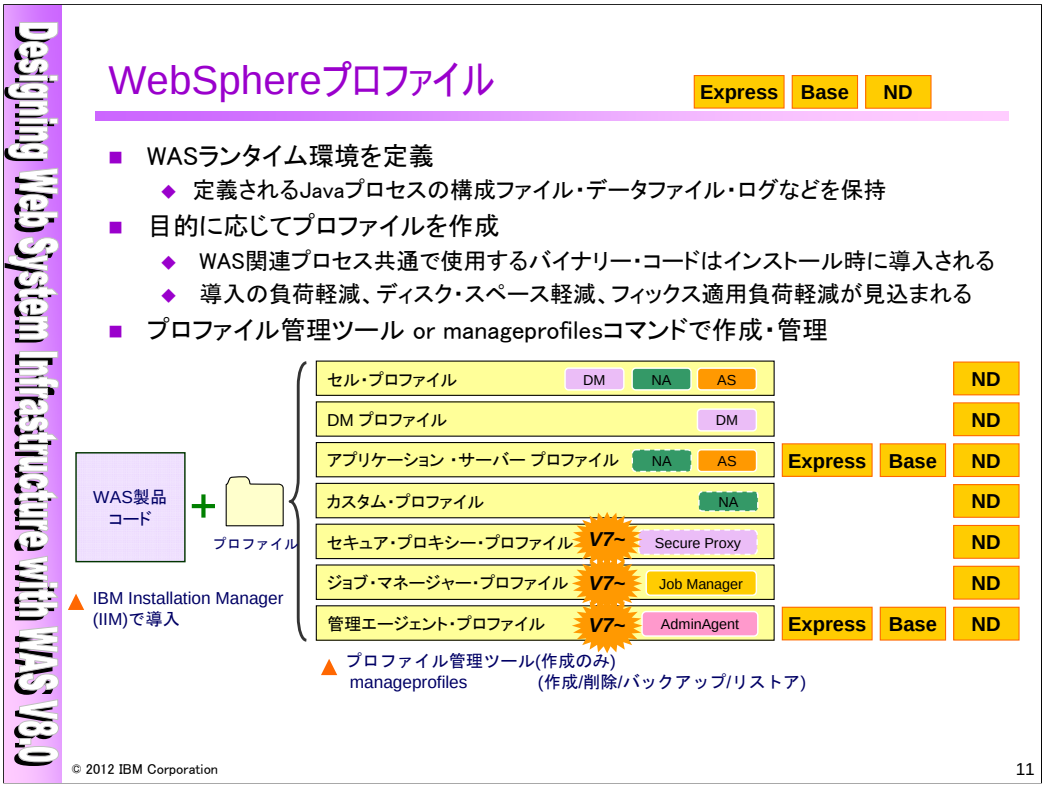
アプリケーション・サーバー

Javaプロセスとして稼動し、JavaEEアプリケーション実行環境を提供します。

アプリケーション・サーバーでは主に次のようなサービスを提供します。

ーWebコンテナー、EJBコンテナー、JMS、JMX、セキュリティー、WLM/HA、トランザクション


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
-WAS基本コンポーネント
-WebSphereプロファイル
-WAS基本トポロジー
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
10

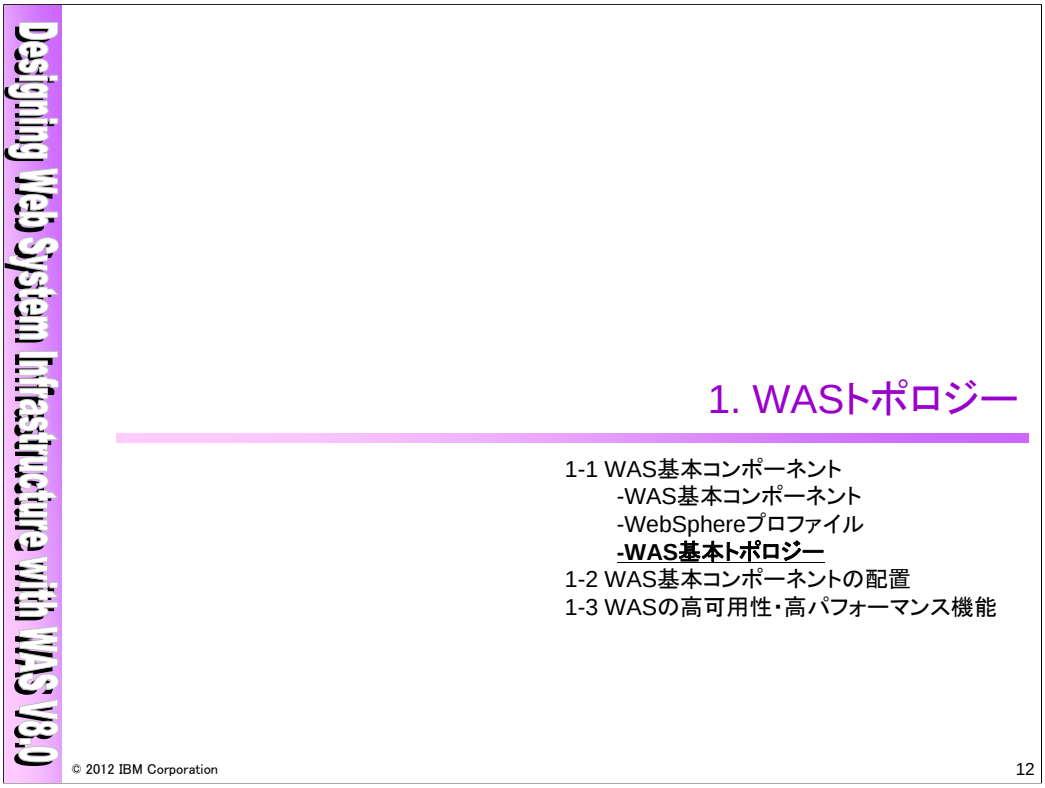


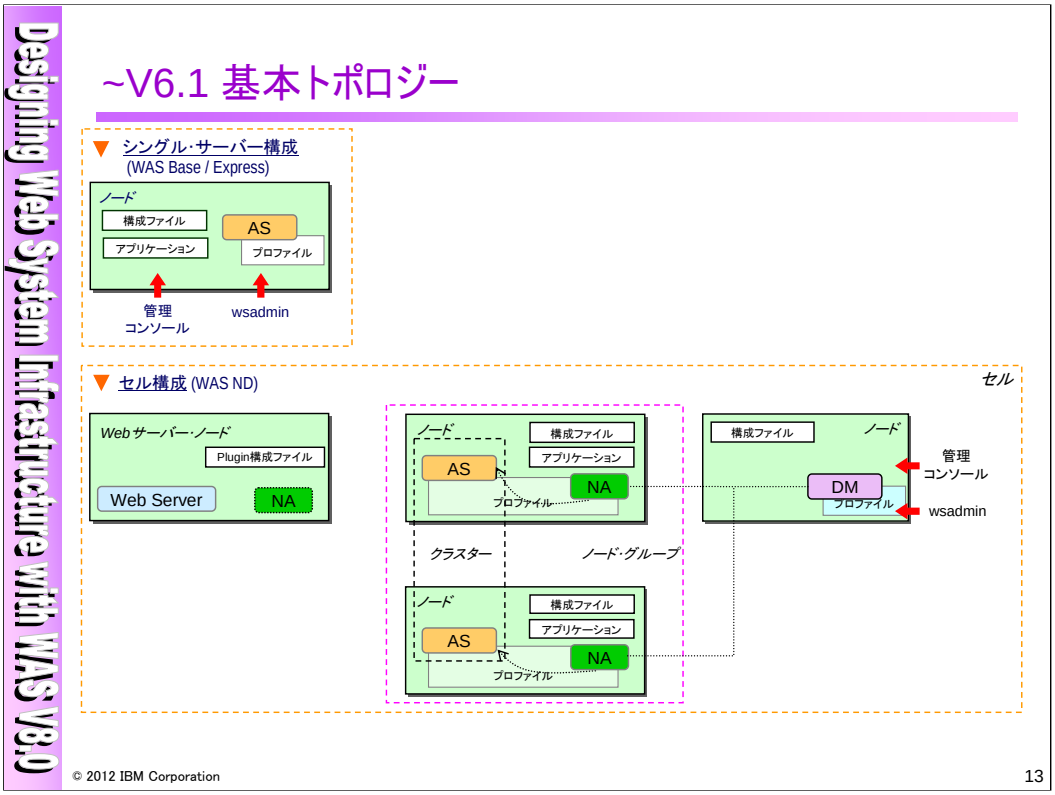
WASのノードを実際に定義するには、構成に応じたプロファイルを作成する必要があります。

プロファイルは、1ノード上のJavaプロセスの実行環境の定義です。WAS関連のプロセスが共通で使用するバイナリー・コードとJavaプロセスの定義情報を分けて導入・構成することで、セルに対しノードを追加する場合など、目的に応じてプロファイルを作成するのみで済み、不要にバイナリコードを導入する必要がなくなりますので、環境作成・変更にかかる負荷の軽減やディスク・スペースの削減が見込まれます。

プロファイルに関しては、プロファイル管理ツールあるいはmanageprofilesコマンドで作成・管理する必要があります。

プロファイル管理ツールはV7までは32bit版のみの提供でしたが、V8では64bit版も提供されています。


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
-WAS基本コンポーネント
-WebSphereプロファイル
-WAS基本トポロジー
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
12

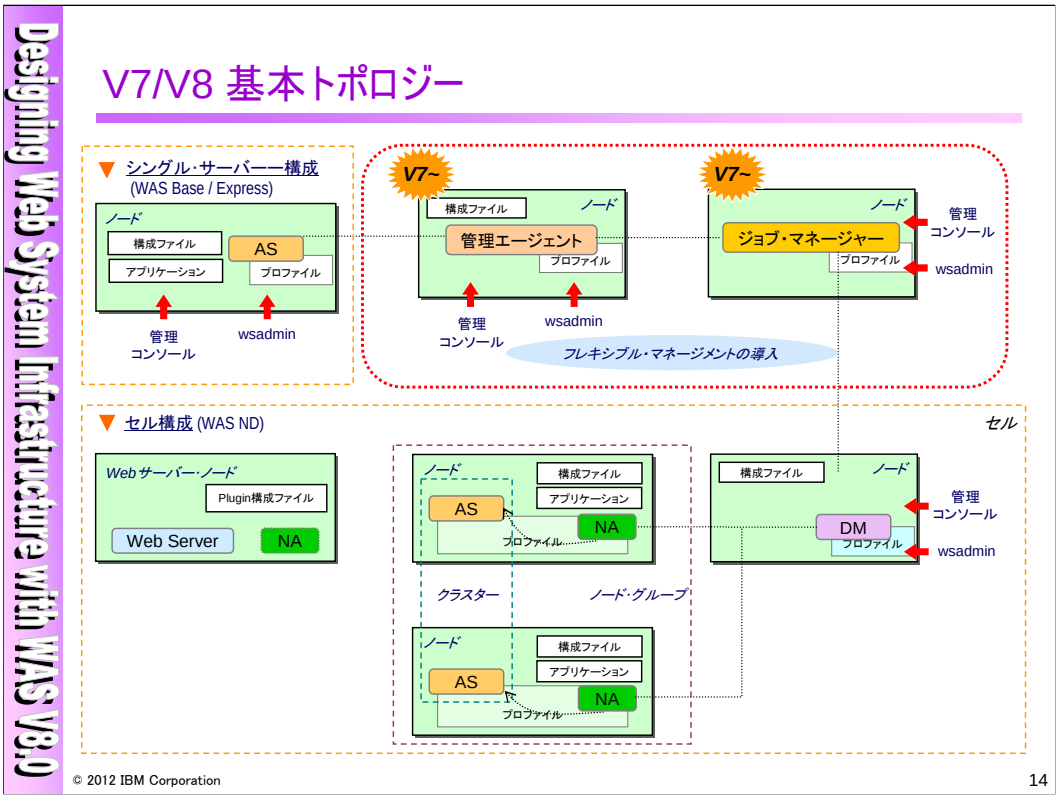


WAS V6.1までは、管理構成としてWAS BaseもしくはExpress版を使用したシングル・サーバー・ノードと、WAS ND (Network Deployment)版を使用したセル構成の2パターンをとることが出来ます。

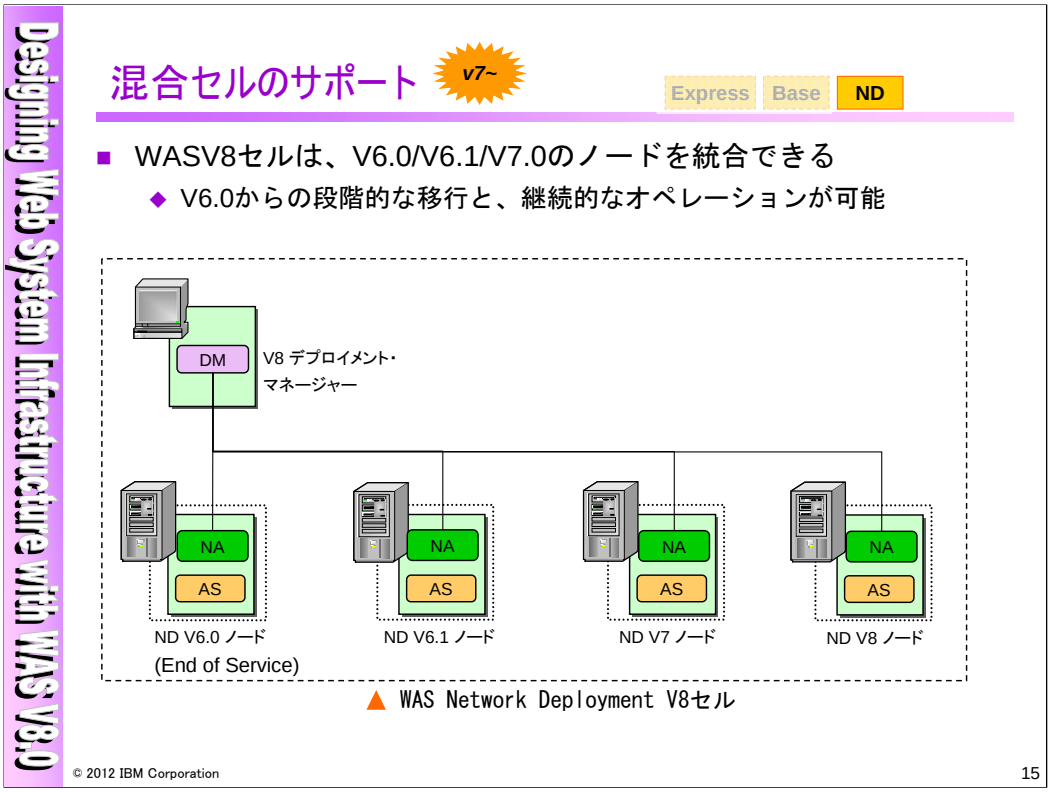
重要な管理対象としては大まかに分けて二つ、configディレクトリ配下に置かれるXMLで記載された構成ファイル(JVMプロセスの定義情報やJDBCデータソースといったリソース情報など)と、installedAppsディレクトリ配下に置かれるアプリケーション・サーバー(AS)上で稼動するエンタープライズ・アプリケーションです。これらを管理するオペレーションは、管理コンソールと言うブラウザー・インターフェースのツールと、wsadminというコマンドラインツールによって行うことが可能です。

シングル・サーバー・ノードでは、管理は個々のノードに対して行います。管理コンソール自体もアプリケーション・サーバー上に導入されるエンタープライズ・アプリケーションなので、ノード内の一つのアプリケーション・サーバー上に自動的に導入されます。

セル構成ではセル内のマシンを個別に管理していては負荷がかかりますので、管理ポイントを集約しています。これがデプロイメント・マネージャー(DM)というプロセスです。セルに参加するアプリケーション・サーバーノードは、addNodeというコマンド(もしくは管理コンソールの『ノードの追加』オペレーション)でデプロイメント・マネージャー管理下に配置されます。これはノード内にノード・エージェント(NA)というプロセスを起動し、デプロイメント・マネージャーとの構成情報やアプリケーションの同期を取ることが出来る構成です。ノード・エージェントには自ノードで稼動するアプリケーション・サーバーを定期的に監視し、異常停止の場合再起動するという機能もあります。

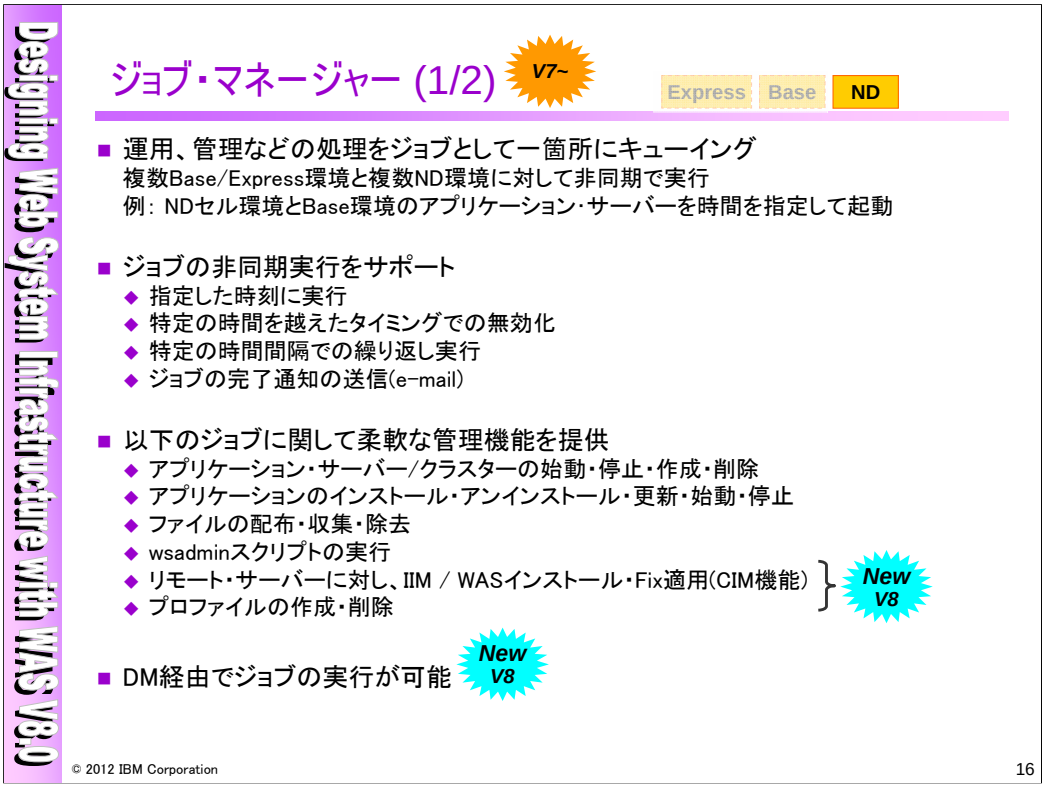


こちらはV7以降の基本トポロジーです。基本的にはV6.1までのトポロジーと同じですが、変更点としては新しいコンポーネントとしてジョブ・マネージャーと管理エージェントが加わりました。管理エージェントはWAS Base構成/Express構成を集中管理するコンポーネントです。ジョブ・マネージャーは複数のNDセルやシングル・サーバー・ノードを統一的に運用管理するコンポーネントです。管理エージェントとジョブ・マネージャーはプロファイルを使用して作成されます。また、この二つの管理コンポーネントによってフレキシブル・マネージメントという新しい概念が実現されます。



WASV8セルは、V6.0/V6.1/V7.0のノードを統合することが可能です。これにより、V6.0からの段階的な移行と、継続的なオペレーションが可能です。ただしこの構成を取る場合にはいろいろと考慮事項がありますので、詳細は「運用設計」セッションのマイグレーションの章をご参照ください。

ただし、V7.0のセルと違い、V5.xのセルは統合できないことにご注意下さい。



ジョブ・マネージャーとは運用、管理などの処理をジョブとして一箇所にキューイングし、複数Base/Express環境と複数NDセル環境に対して非同期で実行する仕組みを提供する管理コンポーネントです。アプリケーション・サーバーを時間を指定して起動させるなど、運用・管理の処理をスケジューリングして実行させたい場合にはジョブ・マネージャーを使用します。また、複数NDセル環境や複数Base/Express環境が混在する環境においてもジョブ・マネージャーを使用することで統一的な運用管理が可能になります。

V8.0において、ジョブ・マネージャーが実行できるジョブにリモートサーバーに対するIIMやWASのインストール・Fix適用といった集中インストール・マネージャー(CIM)機能や、プロファイルの作成・削除が追加され、よりジョブ・マネージャーが利用できるケースが増えました。さらに、V8.0ではDMからジョブ・マネージャーに対し、ジョブを登録し実行させることも可能になりました。

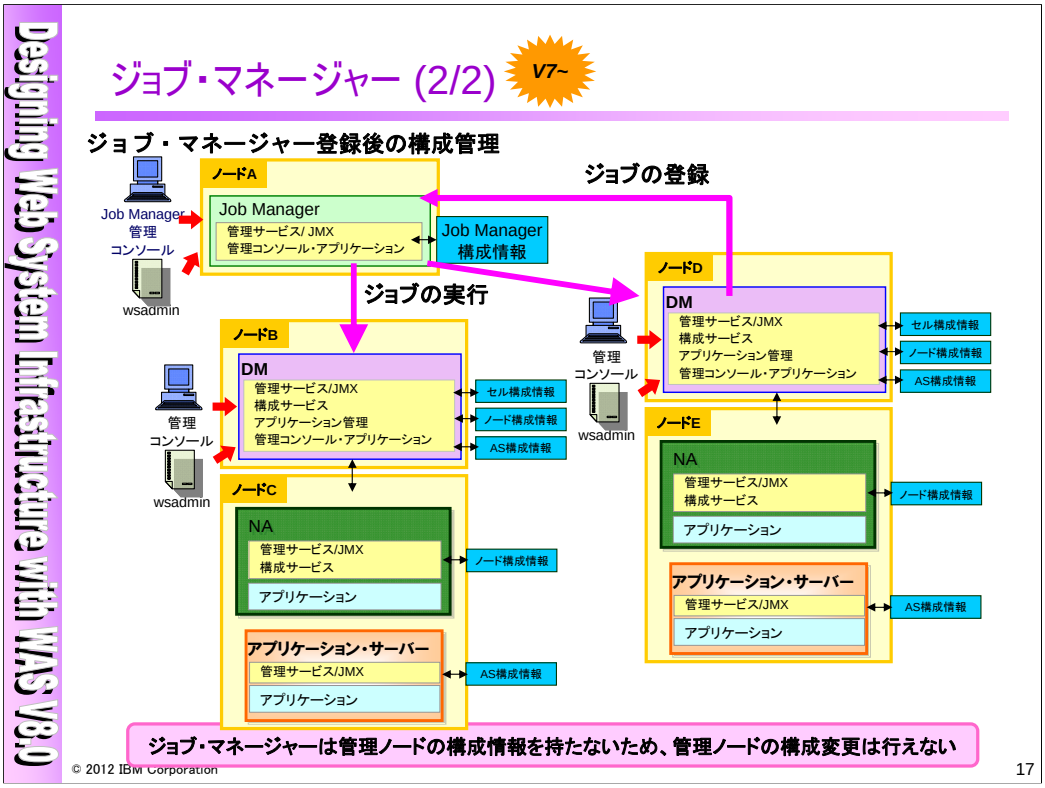
ジョブの管理はジョブ・マネージャー管理コンソールもしくはwsadminで可能です。wsadminでのジョブの管理については以下のInfoCenterの記載をご参照ください。

Information Center「wsadmin スクリプトを使用する管理ジョブ・タイプ」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rxml_7jobtypes.html

Information Center「wsadmin スクリプトによる柔軟な管理環境でのジョブの管理」

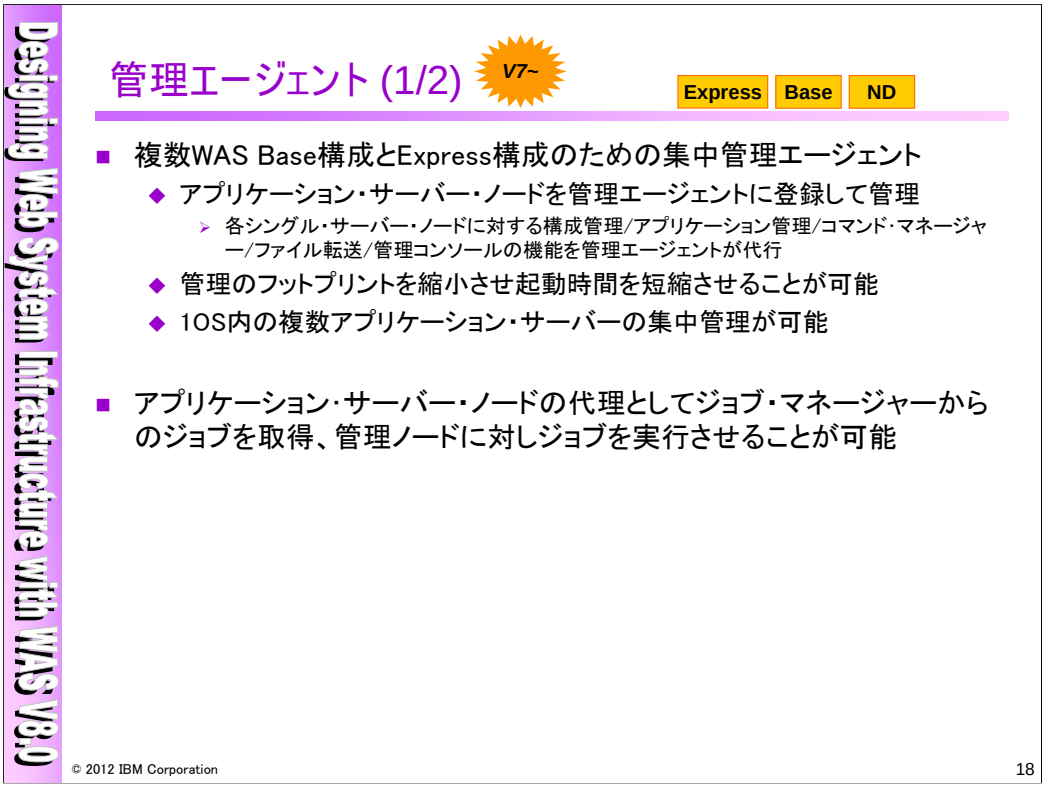
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/txml_7flexiblemgmt.html



上図がジョブ・マネージャーにNDセル環境を登録後の構成管理です。

NDセル環境をジョブ・マネージャーに登録しても、ジョブ・マネージャーではNDセル環境の構成情報は持たず、ジョブ・マネージャー自身の構成情報しか持ちません。そのためジョブ・マネージャーでは配下の管理ノードの構成変更を行うことは出来ないため、ジョブ・マネージャーに登録しているノードの構成変更をおこなう際は、NDセル環境の場合はデプロイメント・マネージャーに、シングル・サーバー・ノードの場合は管理エージェントに対して管理コンソールもしくはwsadminコマンドで接続し、構成変更をおこないます。

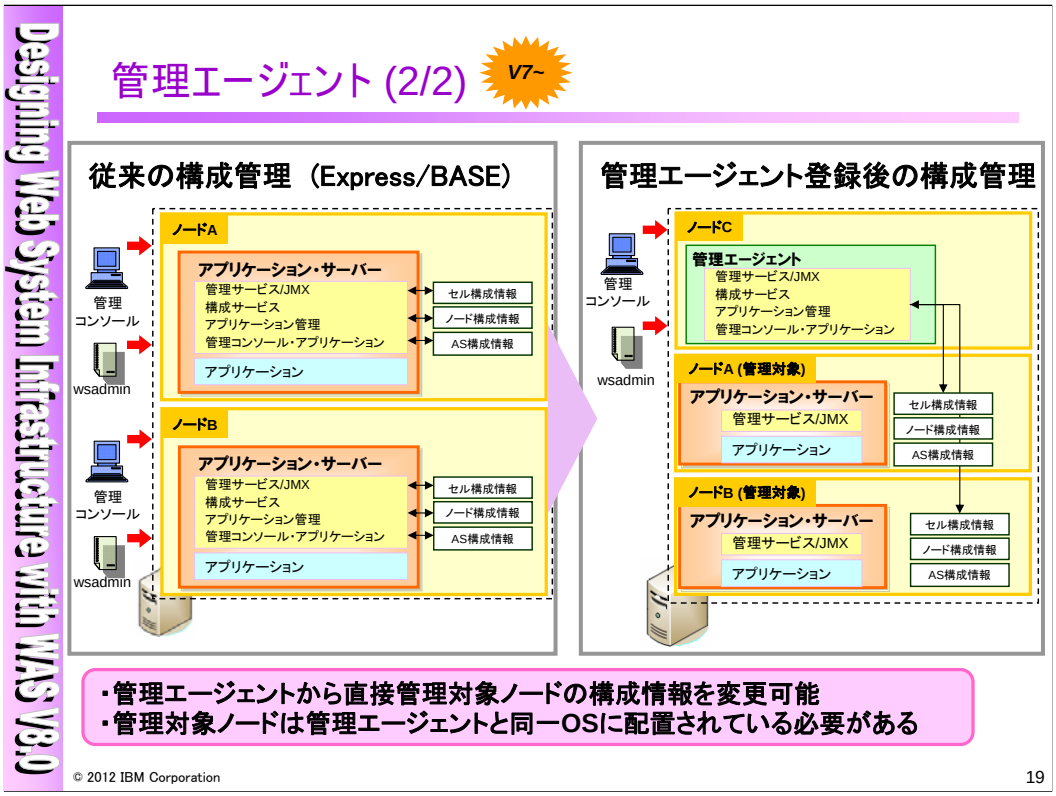
またジョブ・マネージャーはデプロイメント・マネージャーやアプリケーション・サーバーと同一筐体に配置することも別筐体に配置することも可能です。ただしジョブ・マネージャーも一つのJavaプロセスですので、同一筐体に配置した場合はジョブ・マネージャーの分だけアプリケーション・サーバーで使用できるリソースは減りますのでご注意ください。



管理エージェントとはWAS Base構成とExpress構成を管理するためにV7.0より新しく加わった管理コンポーネントで、管理エージェントにより1OS内の複数のBase構成とExpress構成を集中管理することが可能になりました。

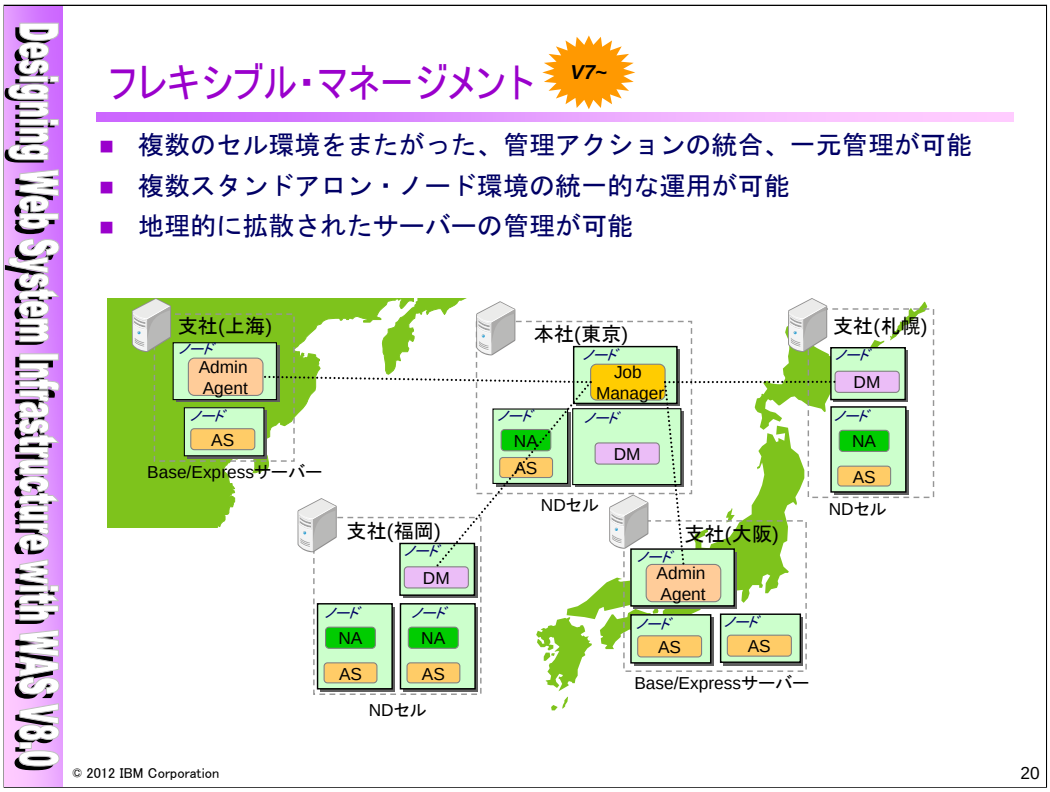
アプリケーション・サーバー・ノードが登録されるとそのノードは管理対象ノードとして扱われます。管理エージェントの管理対象ノードになると構成管理/アプリケーション管理/コマンド・マネージャー/ファイル転送/管理コンソールの機能が管理エージェントに移行され、アプリケーション・サーバーのフットプリントを縮小させ起動時間を短縮させることができます。

また、管理エージェントは管理エージェントの管理対象ノードであるアプリケーション・サーバー・ノードの代理としてジョブ・マネージャーからのジョブを取得し、各アプリケーション・サーバー・ノードに対してジョブを実行させることが可能です。そのためには管理エージェントの管理対象ノードをジョブ・マネージャーに登録する必要があります。



管理エージェントの基本構成は右図の通りで、1OS内にあるノードを管理エージェントは管理することが可能です。管理エージェントに登録された段階で、登録されたアプリケーション・サーバー・ノードにある管理コンソールは使用不可能になり、管理エージェントが提供する管理コンソールからの各ノードの構成変更が可能になります。

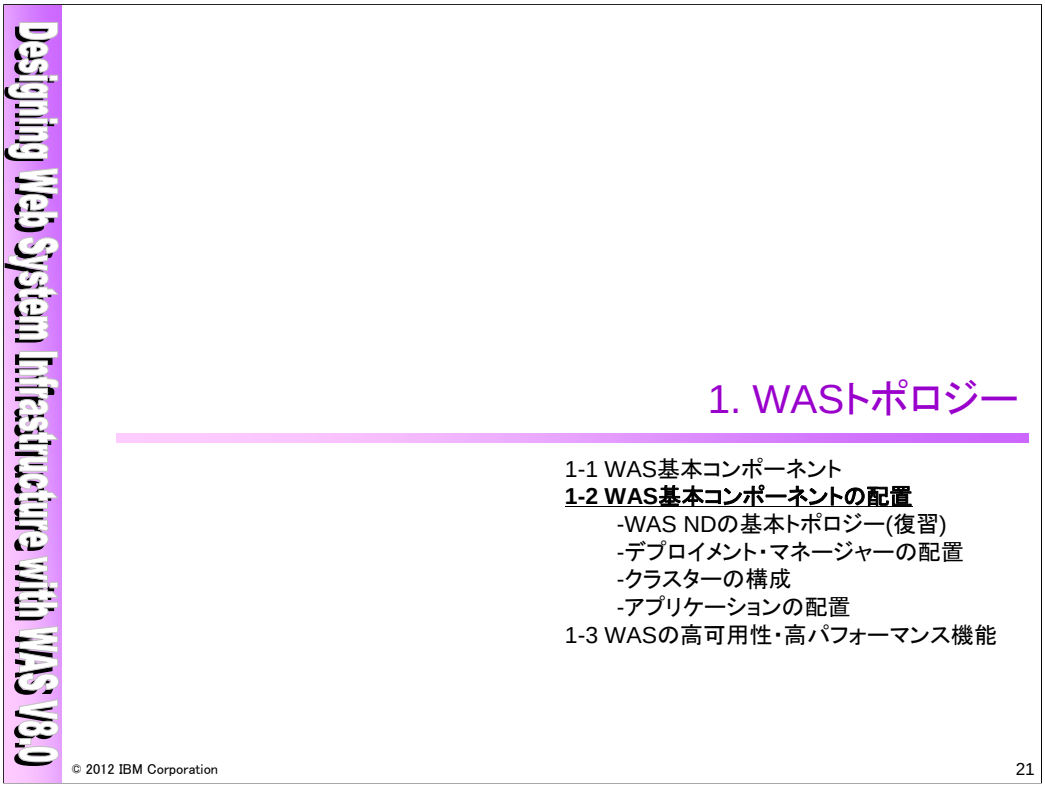
注意点として、複数のOSを跨って管理エージェントにノードを登録できないことが挙げられます。例えば、OS区画#1 / #2にアプリケーション・サーバーが配置されていて、OS区画#1あるいは#2、もしくは異なるOS区画#3に管理エージェントを配置して2台のアプリケーション・サーバーを集中管理する、という構成をとることはできません。また、WAS NDのセル構成のように、管理エージェントに登録したノードの構成情報を同時に出力させることはできませんのでご注意ください。

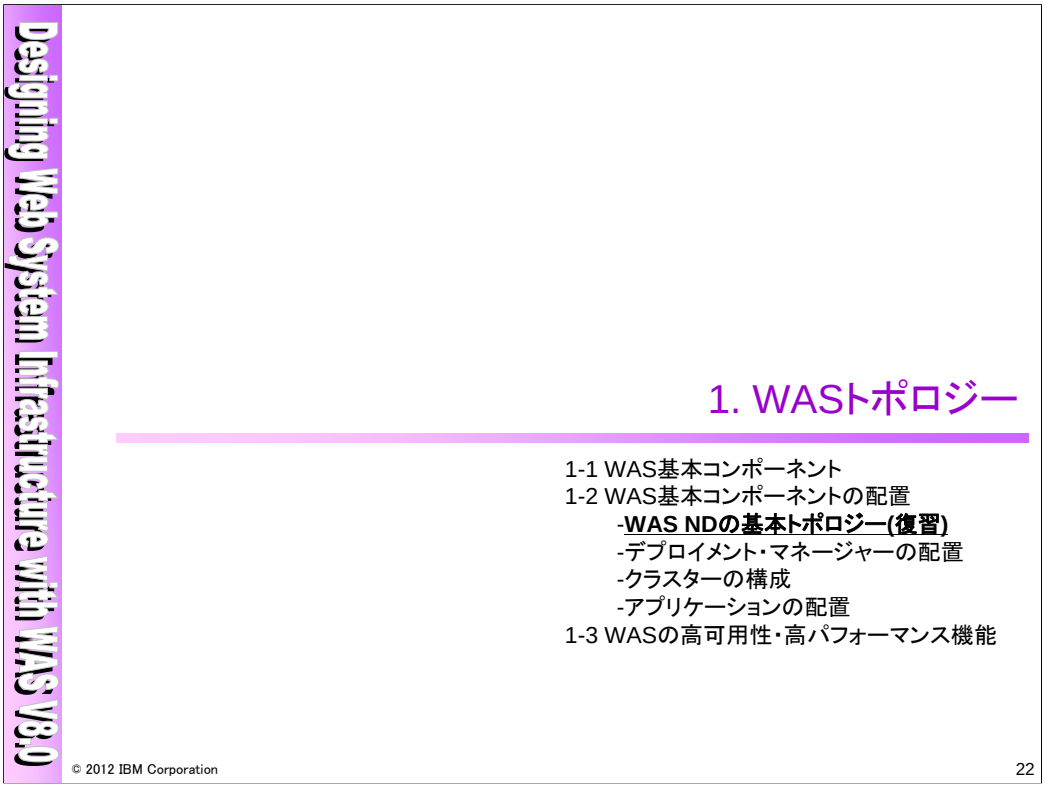


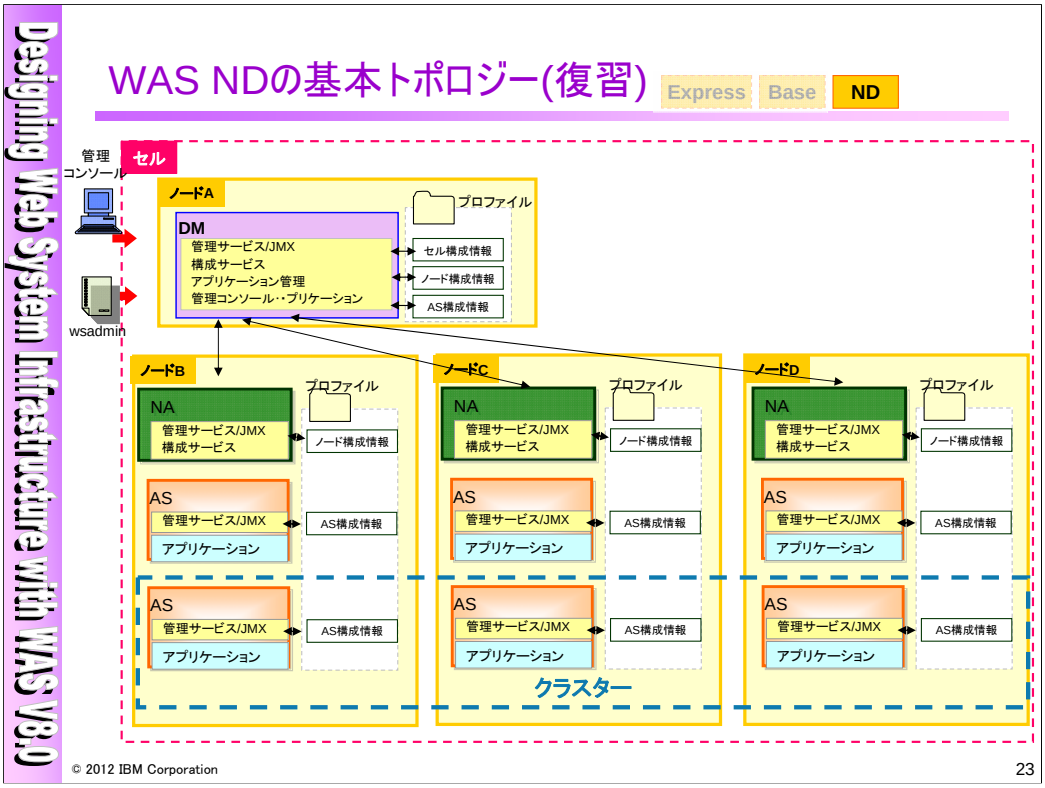
こうした管理系コンポーネントであるジョブ・マネージャーと管理エージェントを用いることで、フレキシブル・マネージメント構成をとることが可能になります。

従来、スタンドアロン環境では各サーバーを個々に管理し、またセル環境ではデプロイメント・マネージャーがノード・エージェントを通じてノード内のサーバーを管理していました。管理対象は個々のサーバーレベルのため、複数スタンドアロン環境や複数セル環境が存在する場合には、それぞれの環境を個別に管理する必要があり、大規模環境での運用性には限界がありました。

そこで登場したフレキシブル・マネージメントを使用することにより、複数セル環境が存在する場合には、複数のデプロイメント・マネージャーの一元管理や管理アクションの統合が可能になりました。複数のスタンドアロン環境が存在する場合には、統一的な運用が可能になりました。また、管理対象のノードは、地理的に拡散されていても非同期でジョブを実行するため、ネットワーク負荷を軽減することができます。


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
-WAS NDの基本トポロジー(復習)
-デプロイメント・マネージャーの配置
-クラスターの構成
-アプリケーションの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
21


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
-WAS NDの基本トポロジー(復習)
-デプロイメント・マネージャーの配置
-クラスターの構成
-アプリケーションの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
22

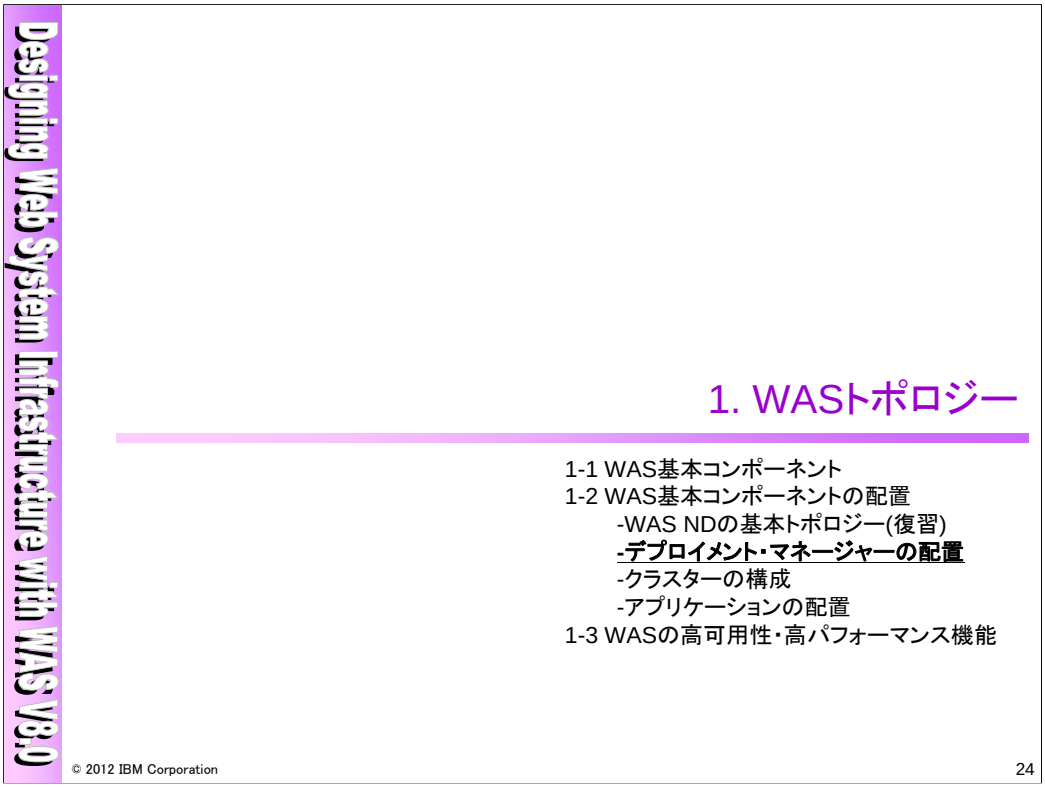


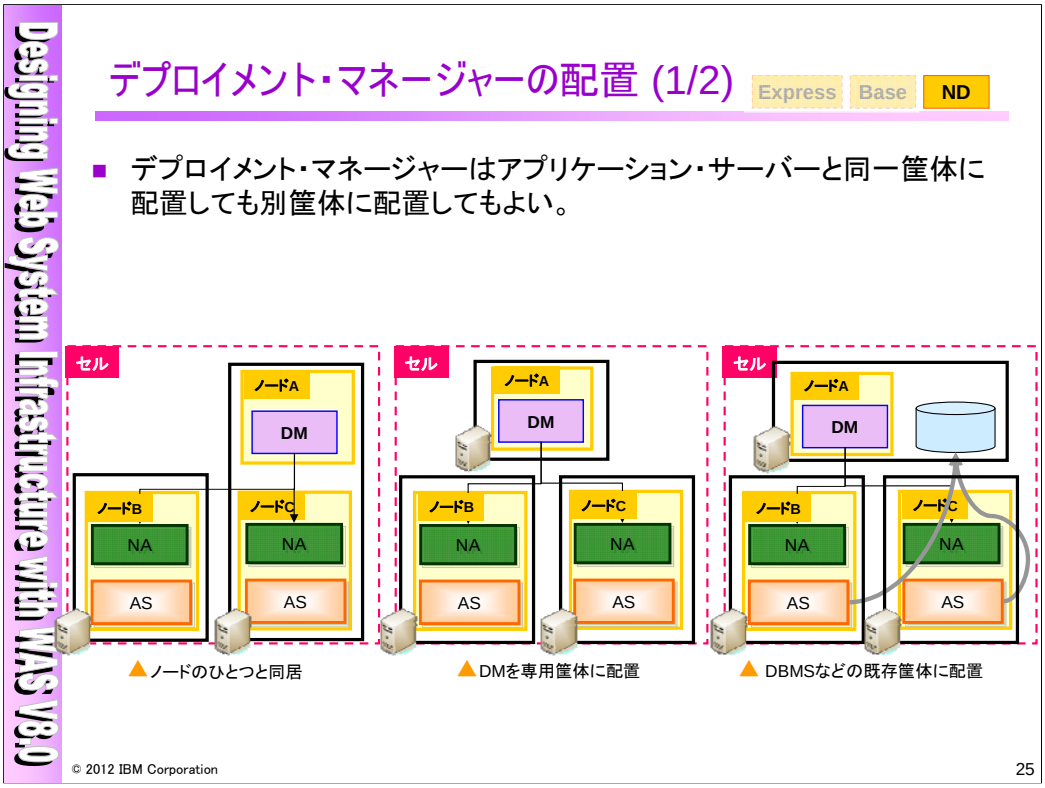
WAS NDの基本トポロジーについて、復習します。

WAS NDではセル構成と呼ばれる、DMで各ノードの構成要素を集中管理する構成が基本の構成になります。アプリケーション・サーバー・ノード構成と違い、アプリケーション・サーバーをノード内に複数たてることも可能ですし、ノードを追加する場合もDMにノードを登録しさえすれば、常に1つの管理コンソールで管理を行うことが可能です。

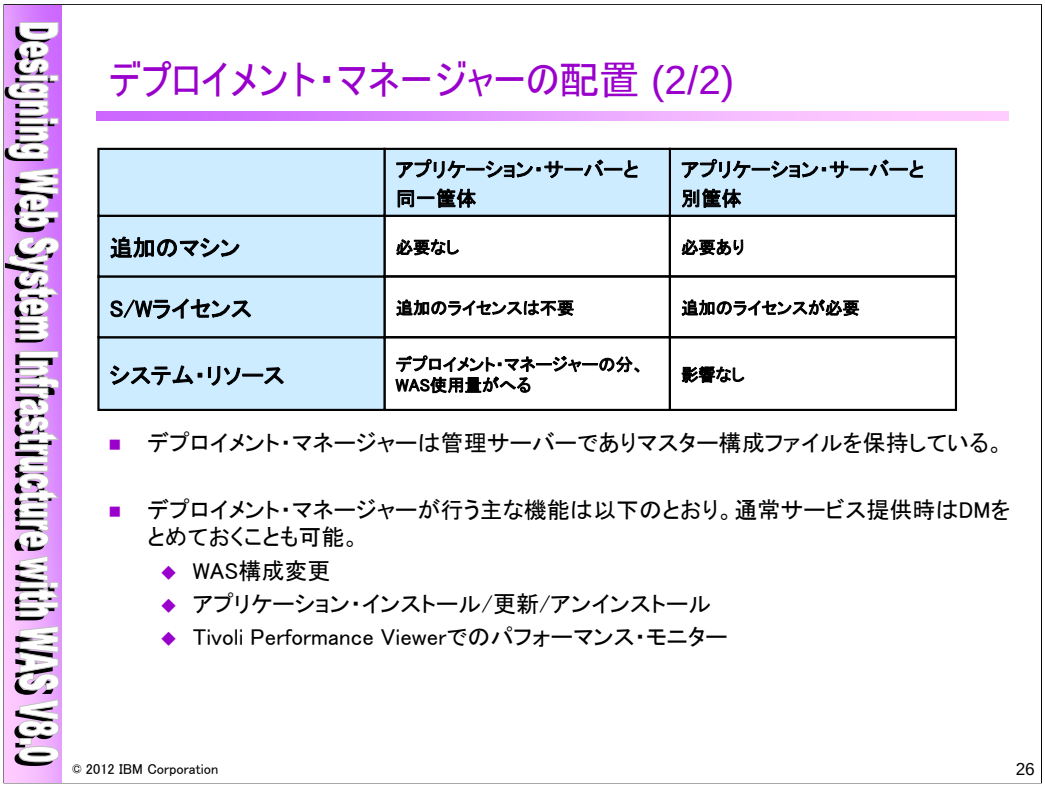
また、複数のアプリケーション・サーバーのグループであるクラスターを作成することもできます。

業務アプリケーションは個々のアプリケーション・サーバーもしくはクラスターを指定してインストールします。

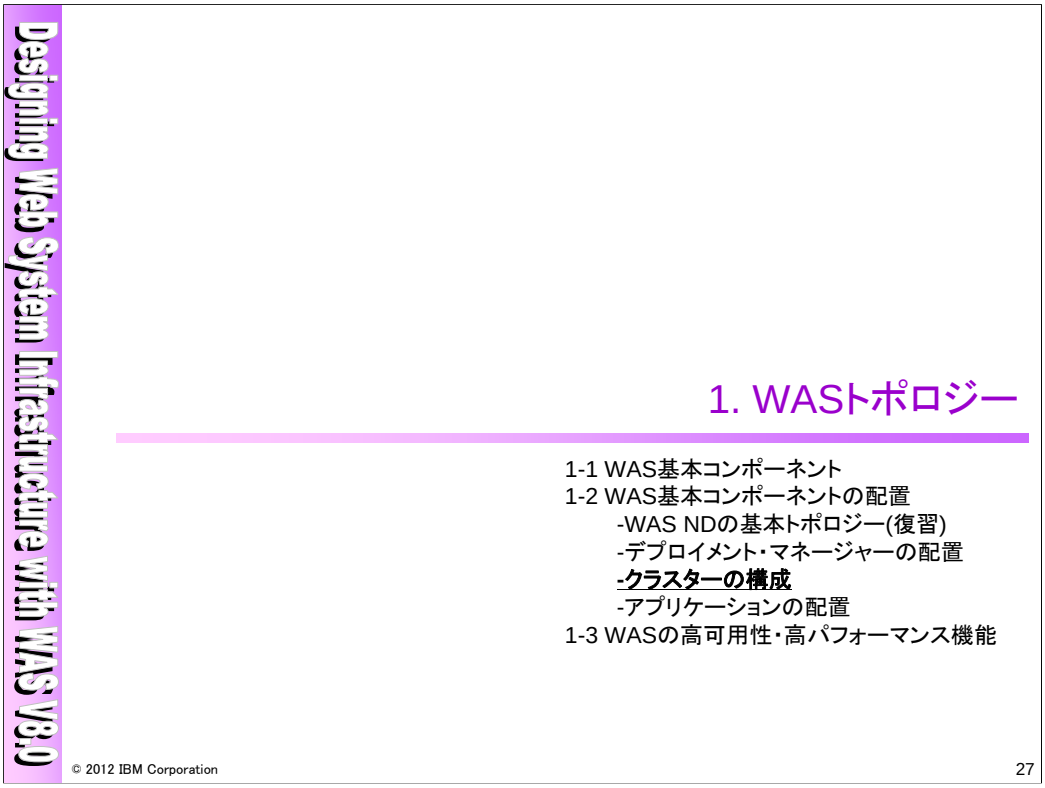

Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
-WAS NDの基本トポロジー(復習)
-デプロイメント・マネージャーの配置
-クラスターの構成
-アプリケーションの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
24

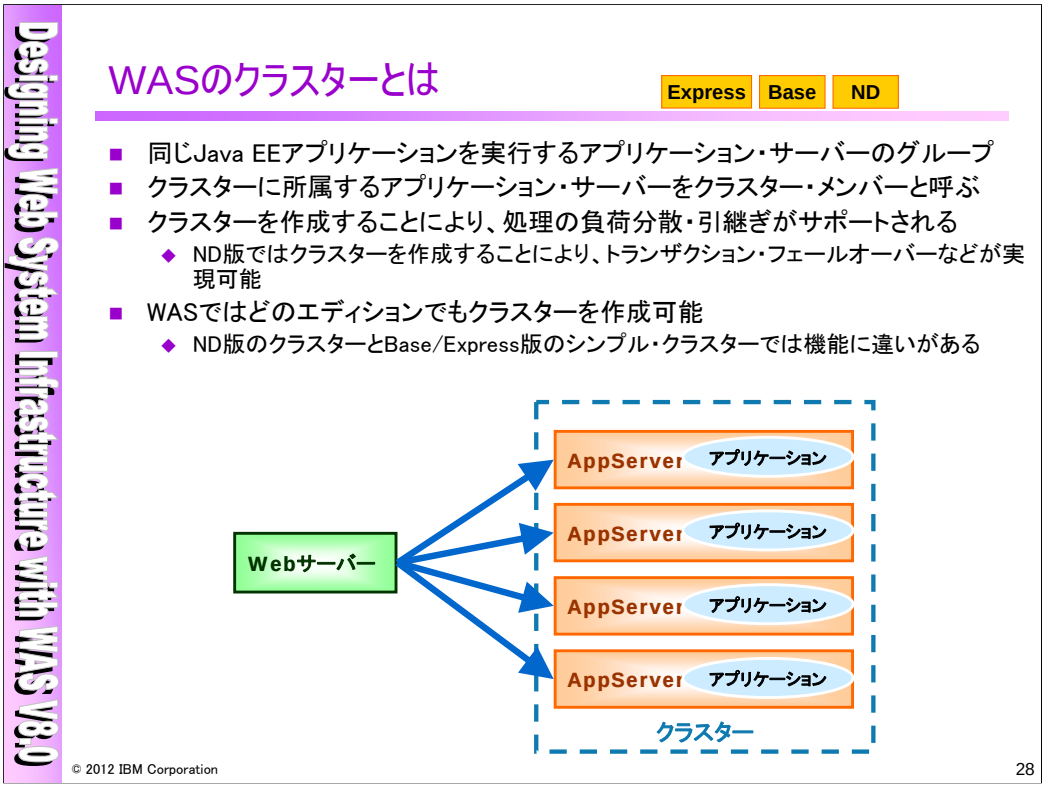


WAS NDではアプリケーション・サーバーの構成の全体の取りまとめと行うデプロイメント・マネージャーが必要です。デプロイメント・マネージャーはアプリケーション・サーバーと同一の筐体に配置する方法と、デプロイメント・マネージャーを他のマシンへ配置する方法があります。同一ノードに配置することで必要なマシンの数やライセンス数を減らすことができますが、デプロイメント・マネージャー自体もJavaプロセスであり、マシンリソースを消費します。同一筐体に配置した場合と、別筐体に配置した場合の比較表を次ページで説明します。



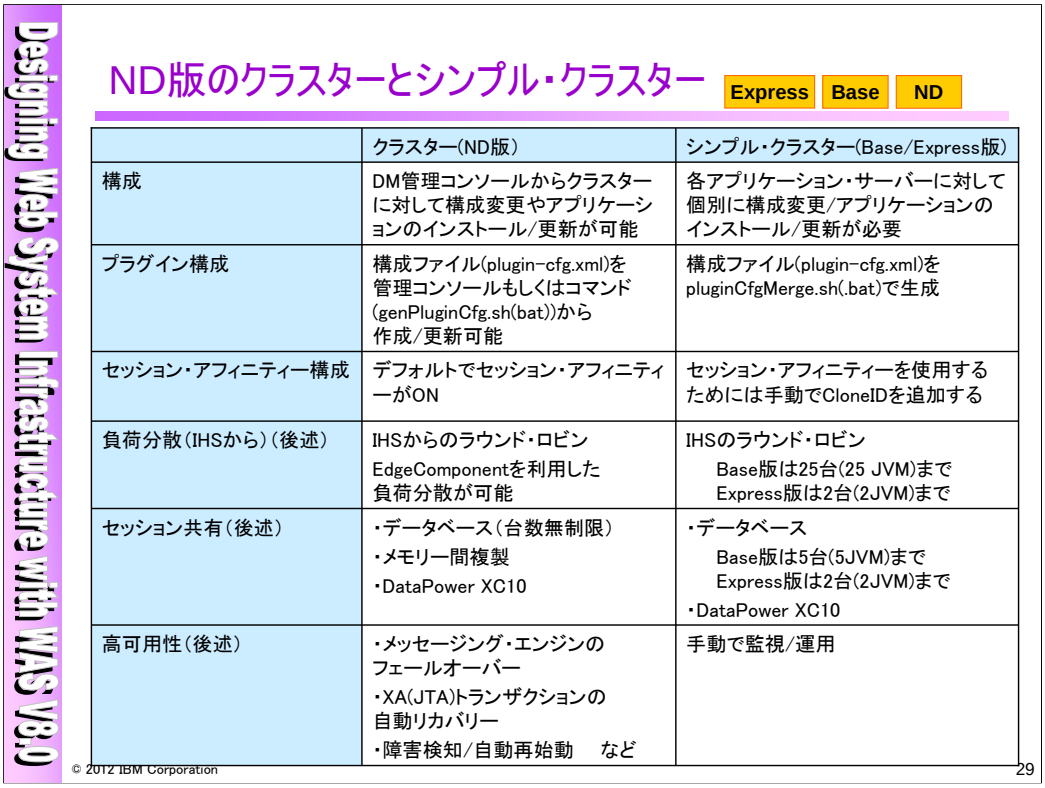
同一筐体に配置しても、デプロイメント・マネージャーを停止しておくことでリソースの使用を抑えることもできます。アプリケーションの更新や構成の変更や、Tivoli Performance Viewerでのパフォーマンス・モニタリングの際に、デプロイメント・マネージャーの起動が必要になります。WAS V6からはEJB WLMに関して、ルーティングテーブルの更新はHAマネージャーが行いますので、デプロイメント・マネージャーを起動させる必要はありません。


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
-WAS NDの基本トポロジー(復習)
-デプロイメント・マネージャーの配置
-クラスターの構成
-アプリケーションの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
27



アプリケーションに対するパフォーマンス・可用性を向上させるため、複数のアプリケーション・サーバーに同一のアプリケーションを稼動させる場合が多いです。

クラスターを構成することで、クライアントからの処理の負荷を分散させるのみならず、クラスターに属するサーバー(クラスター・メンバー)間でのセッションの引継ぎやメッセージングのフェールオーバーなど様々な高可用性機能を使用することが可能になります



WAS Base/Expressにおいてもシンプル・クラスターと呼ばれるクラスター構成をとることが可能です。

但し、WAS NDの持つ高可用性機能の多くが使用できず、割り振り台数にライセンス上の制限が設けられていることに注意が必要です。

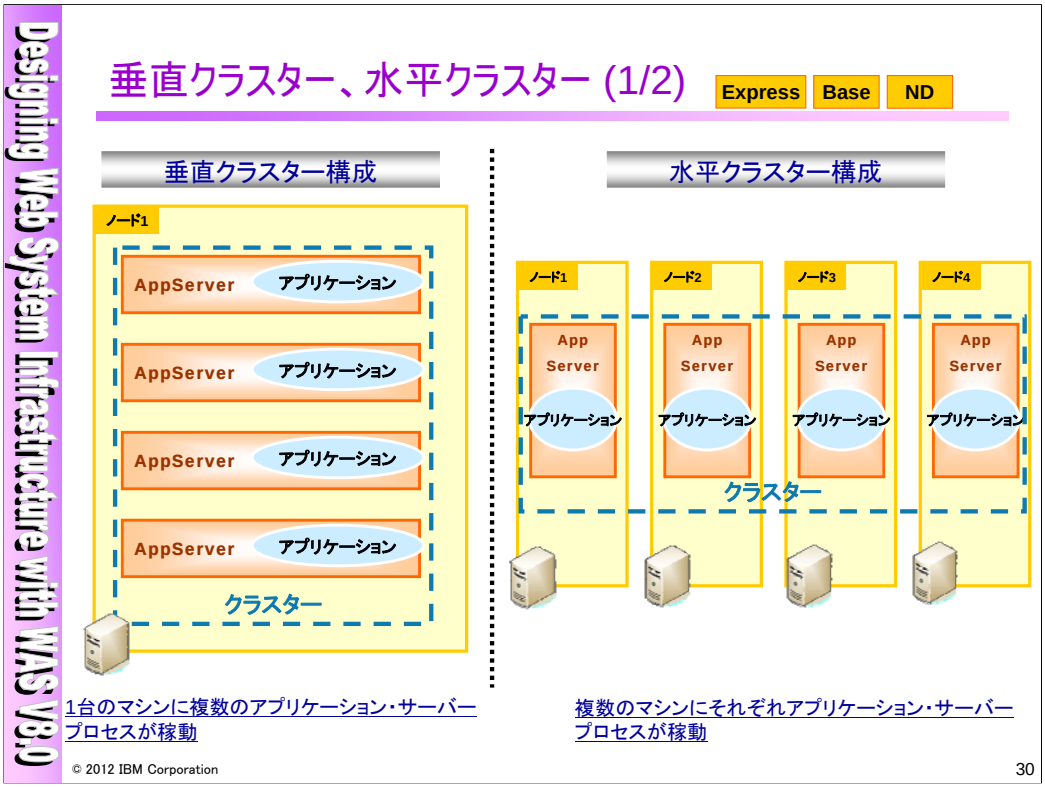
WAS Baseの場合、IHS (IBM HTTP Server) プラグインからBaseサーバーへの割り振りに関し、25台 (25 JVM) まで割り振り制限、セッション・フェールオーバーを行うためのDBへのパーシスタンスは5台までの制限があります(DB2のライセンスに関しては、WASのライセンスにセッション管理の用途でDB2が同梱されているため、追加のライセンスは発生しません)。

WAS Expressに関しては、2台までのプラグイン割り振りと2台までのセッション・パーシスタンスになります。

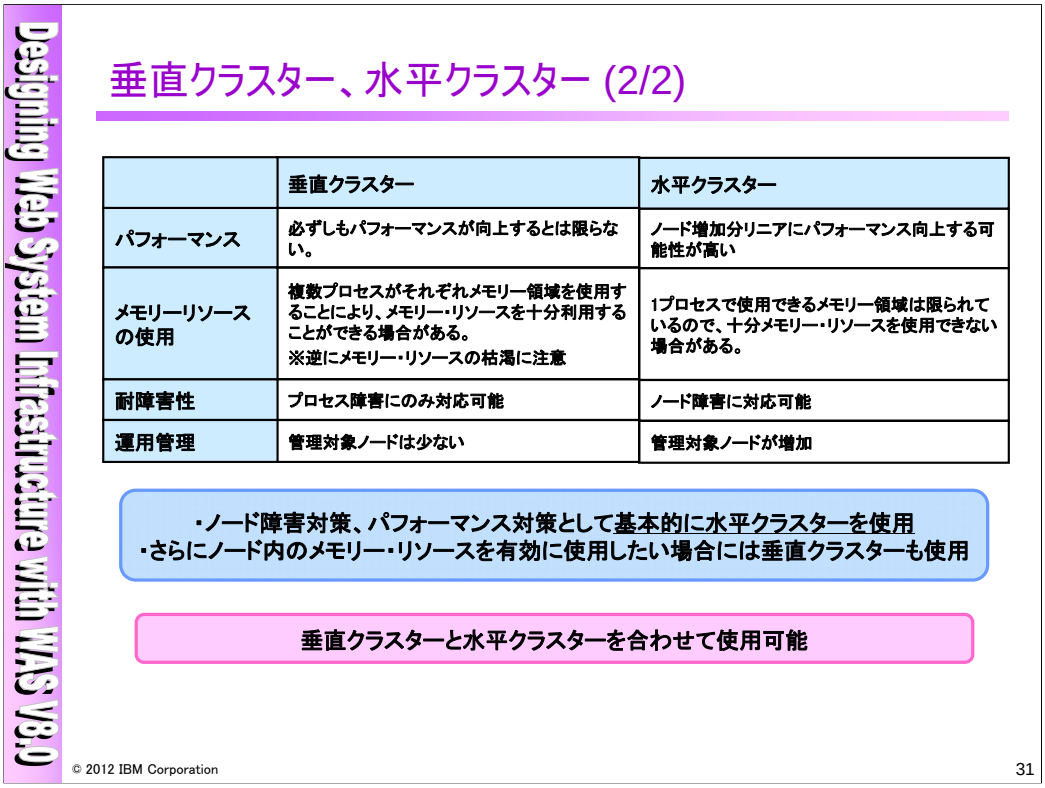
さらに、シンプル・クラスターの場合、構成変更・管理は各アプリケーション・サーバー毎に実施する必要があり、プラグインからの割り振りに対しIHS:WASを1:Nの関係に定義するために手動で構成ファイルを書き換えるか、pluginCfgMergeスクリプトを実行することで構成ファイルを生成する必要が生じることにも注意が必要です。

NDクラスターとシンプル・クラスターの比較がこちらの表になります。

違いを検討し、シンプル・クラスターの制限が許容できる場合、シンプル・クラスター構成も検討します。



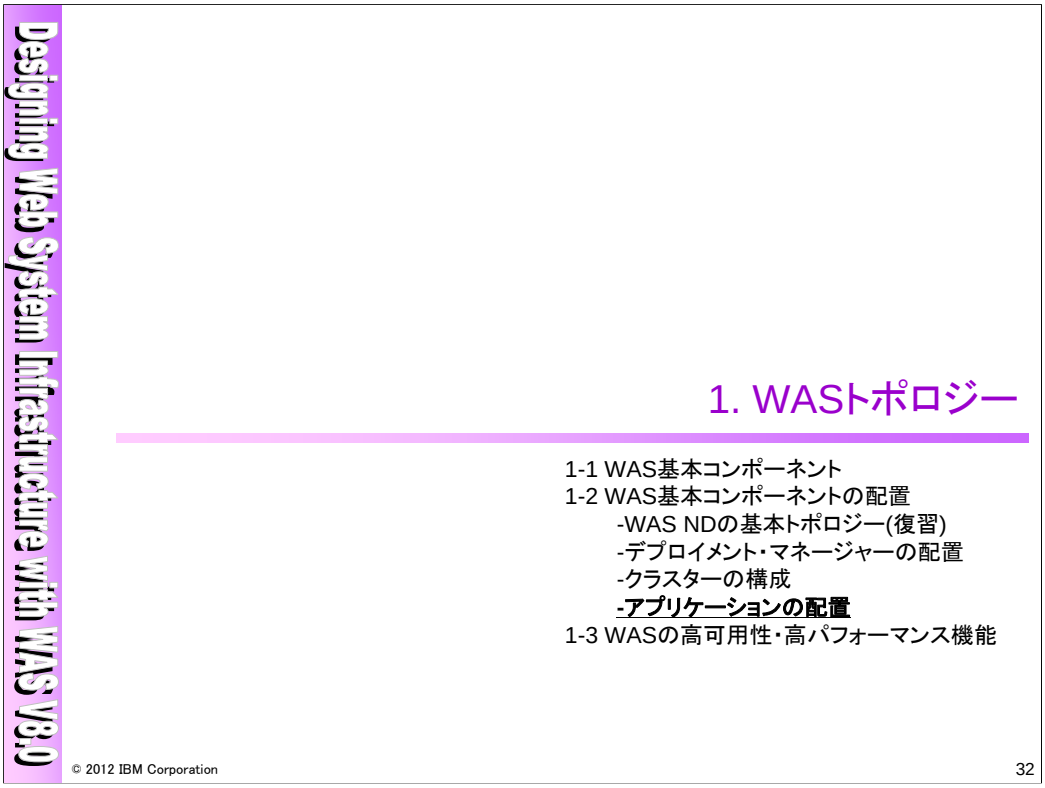
WASでは、パフォーマンス要件を満たすための複数アプリケーション・サーバーの構成方法として、垂直クラスター構成と水平クラスター構成があります。垂直クラスター構成では、一台のマシンに複数のアプリケーション・サーバーを構成します。水平クラスター構成では複数のマシンにそれぞれアプリケーション・サーバーを構成します。

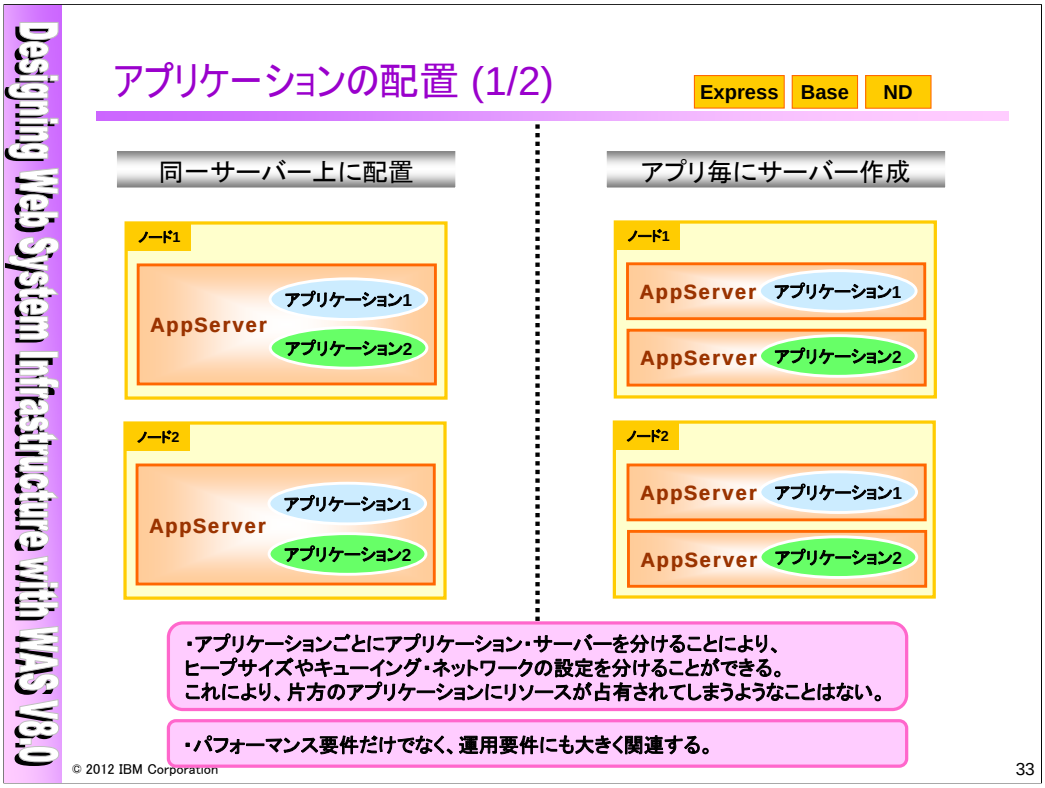


垂直クラスターの場合には1台のマシン上に複数のアプリケーション・サーバーのプロセスを稼動させるため、1台のマシンリソースを共有することになり、必ずしもパフォーマンスが向上するとは言い切れません。一方、水平クラスターは各マシンにそれぞれ各アプリケーション・サーバーのプロセスを稼動させるため、ノードの増加とともにほとんどリニアにパフォーマンスが向上する可能性が高いです。よってパフォーマンスとしては垂直クラスターよりも水平クラスターの方が効果的です。ただし、1台のマシンリソースを十分に利用したい場合には垂直クラスターにより有効に利用することが可能です。

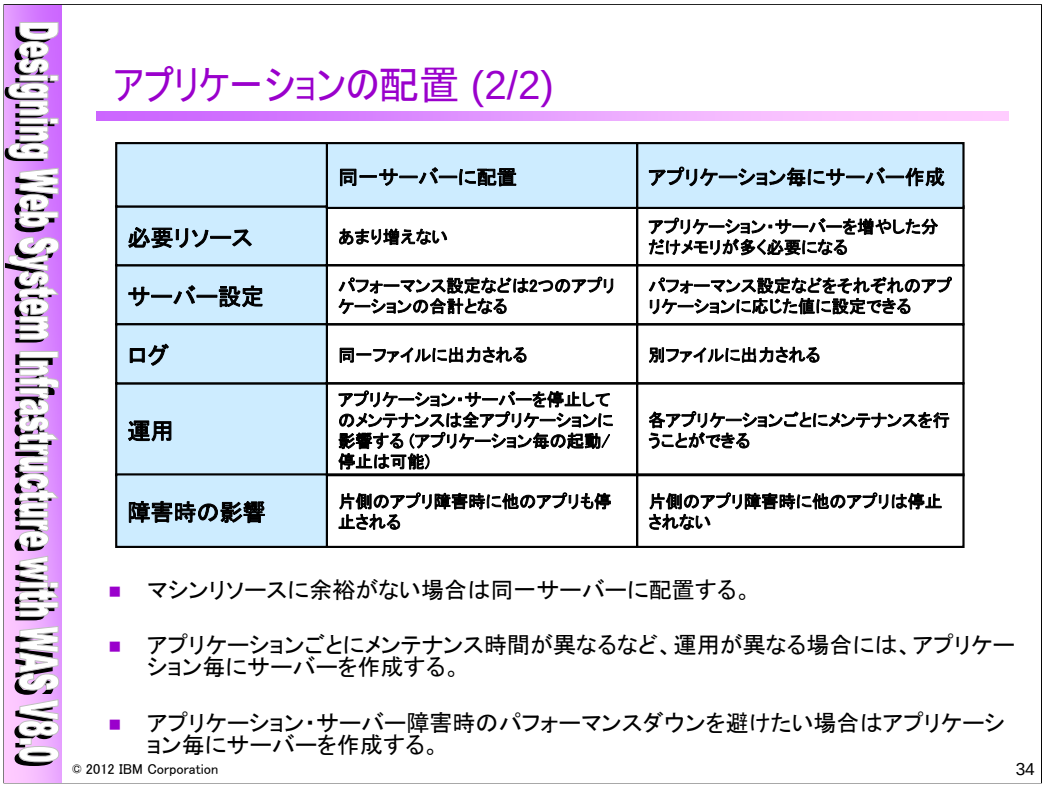
耐障害性においては垂直クラスターはノード障害には対応できず、プロセス障害のみに対応可能です。水平クラスターではノード障害にも対応可能です。

また運用管理については複数マシンをまたがったノード構成をとることは出来ないため、水平クラスターの場合にはマシンの数だけノード数が増加し、その分管理対象ノードも増加します。

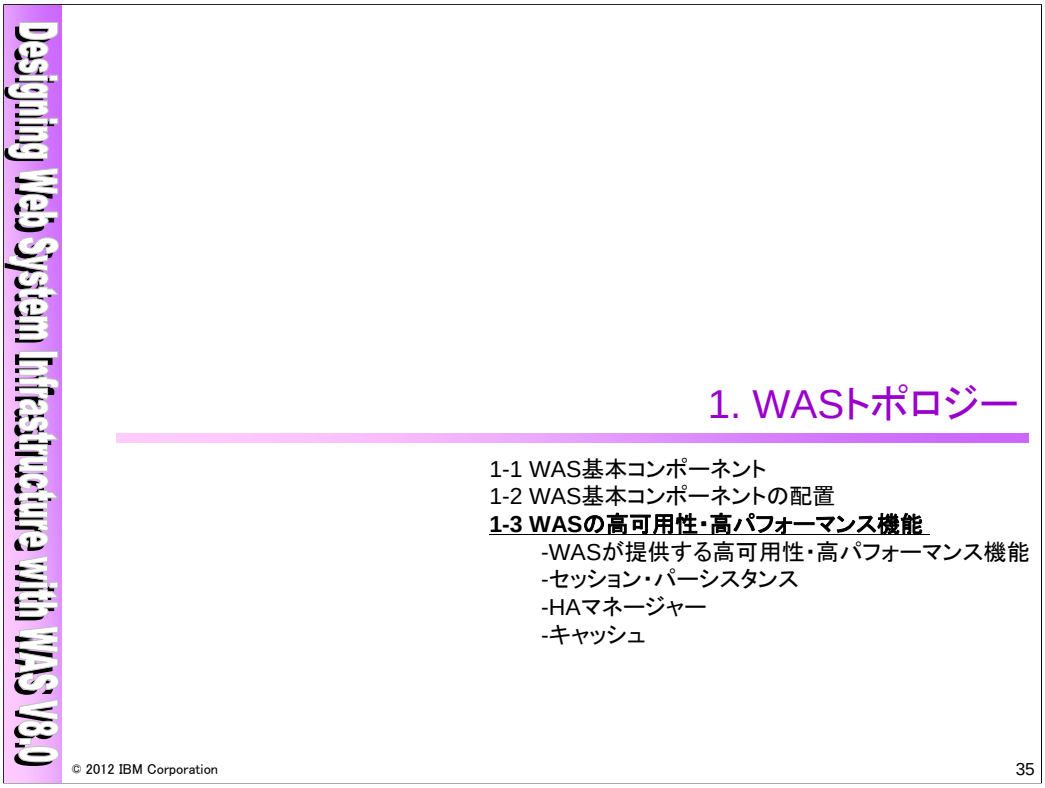

Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
-WAS NDの基本トポロジー(復習)
-デプロイメント・マネージャーの配置
-クラスターの構成
-アプリケーションの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
32

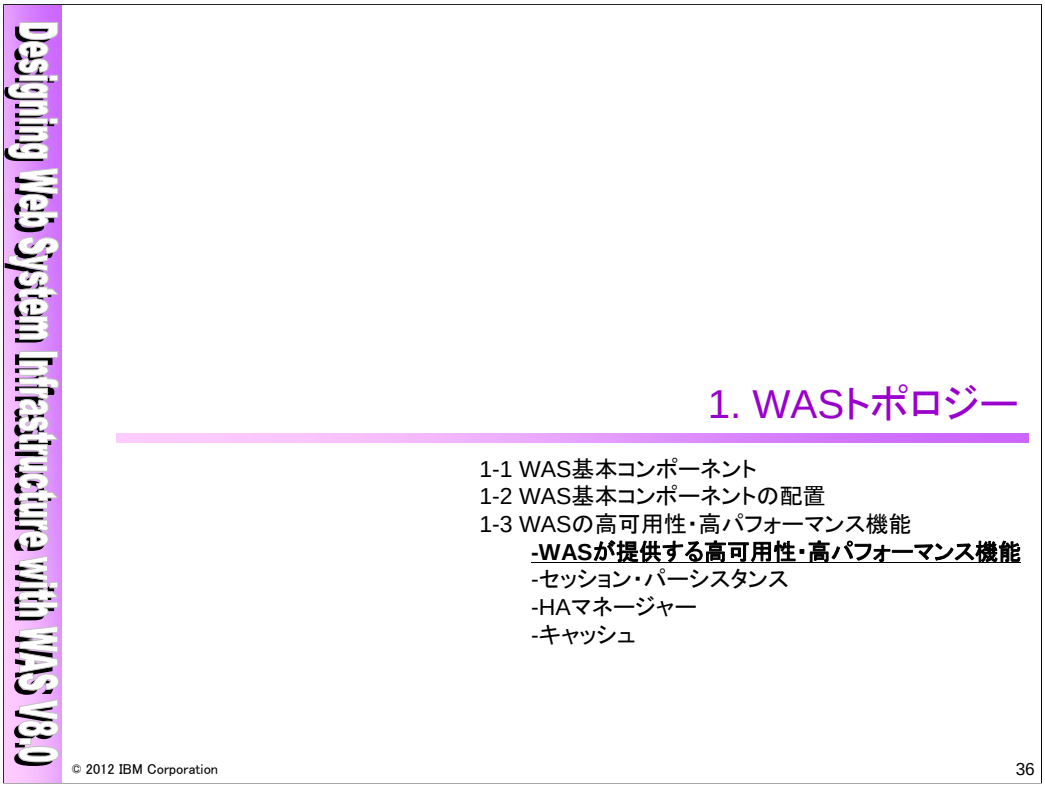


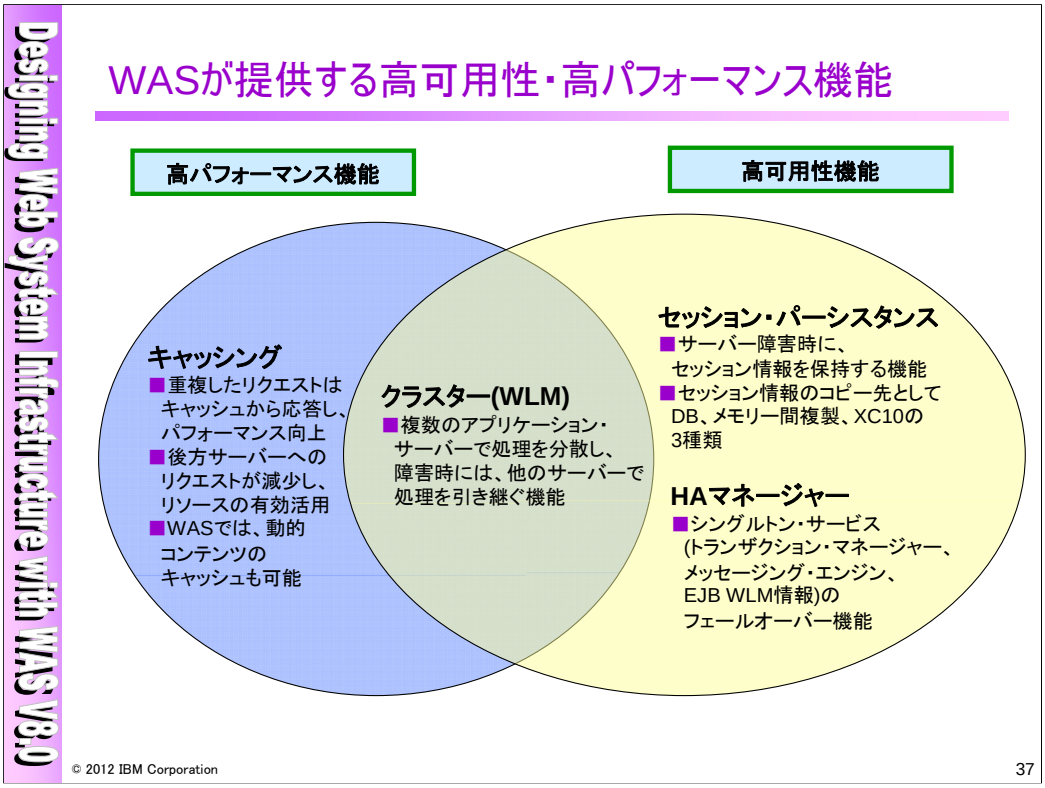
複数のアプリケーションがある場合、一つのアプリケーション・サーバーへデプロイする方法と、それぞれ別のアプリケーション・サーバーへデプロイする方法があります。アプリケーション毎に運用が違う場合や、アプリケーション・サーバーの設定をアプリケーション毎に分けたい場合、アプリケーションの種類や傾向が大きく異なる場合などは、アプリケーション毎にサーバーを作成することで対応できますが、その反面リソースの消費は大きくなりますのでご注意下さい。



アプリケーション毎にサーバーを分けた場合、サーバーのヒープサイズの設定やパフォーマンス・パラメータをアプリケーション毎に分けて設定することが可能です。またパフォーマンス要件だけでなく、運用要件もアプリケーションの配置が影響してきます。例えばアプリケーション毎にメンテナンス時間が異なる場合には同一サーバー上に全てのアプリケーションを配置している場合にサーバーを停止すると全てのアプリケーションに影響が出てしまうため、サーバーを分けた方がメンテナンスがしやすくなります。またSystemOut.logなどのサーバーのログに関しても、アプリケーション毎にサーバーを分けた場合にはアプリケーション毎にログも出力されるため、障害時などのログの解析も行いやすくなります。


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
-WASが提供する高可用性・高パフォーマンス機能
-セッション・パーシスタンス
-HAマネージャー
-キャッシュ
© 2012 IBM Corporation
35

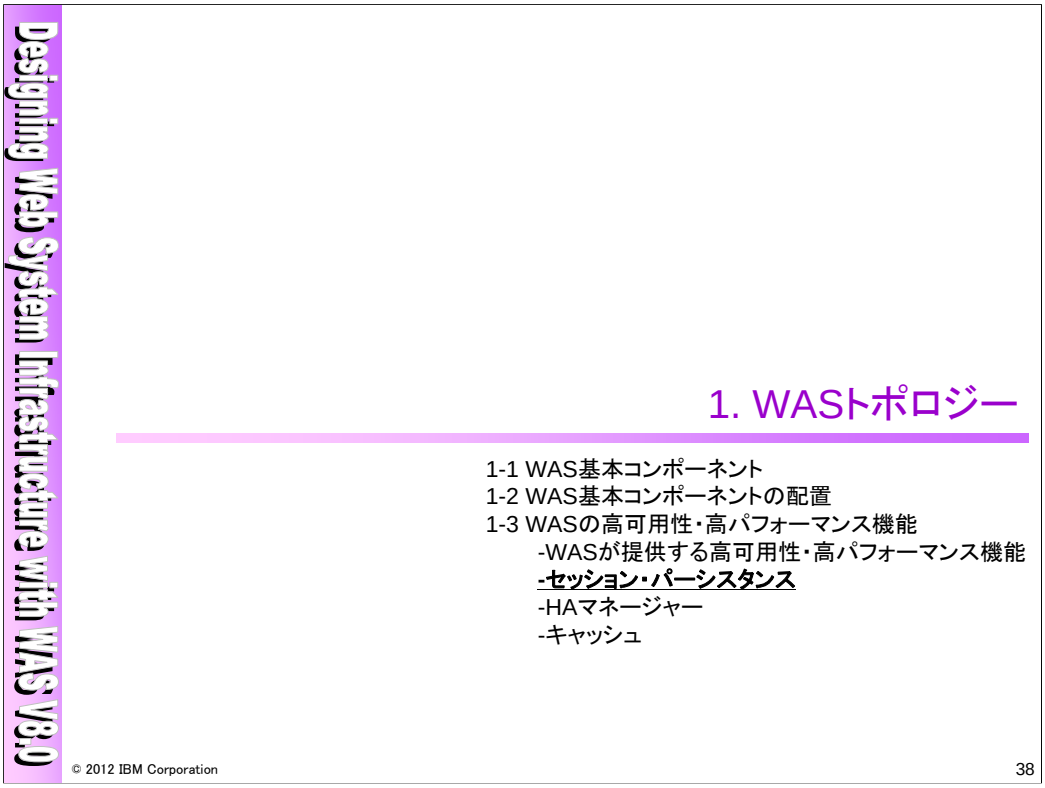

Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
-WASが提供する高可用性・高パフォーマンス機能
-セッション・パーシスタンス
-HAマネージャー
-キャッシュ
© 2012 IBM Corporation
36

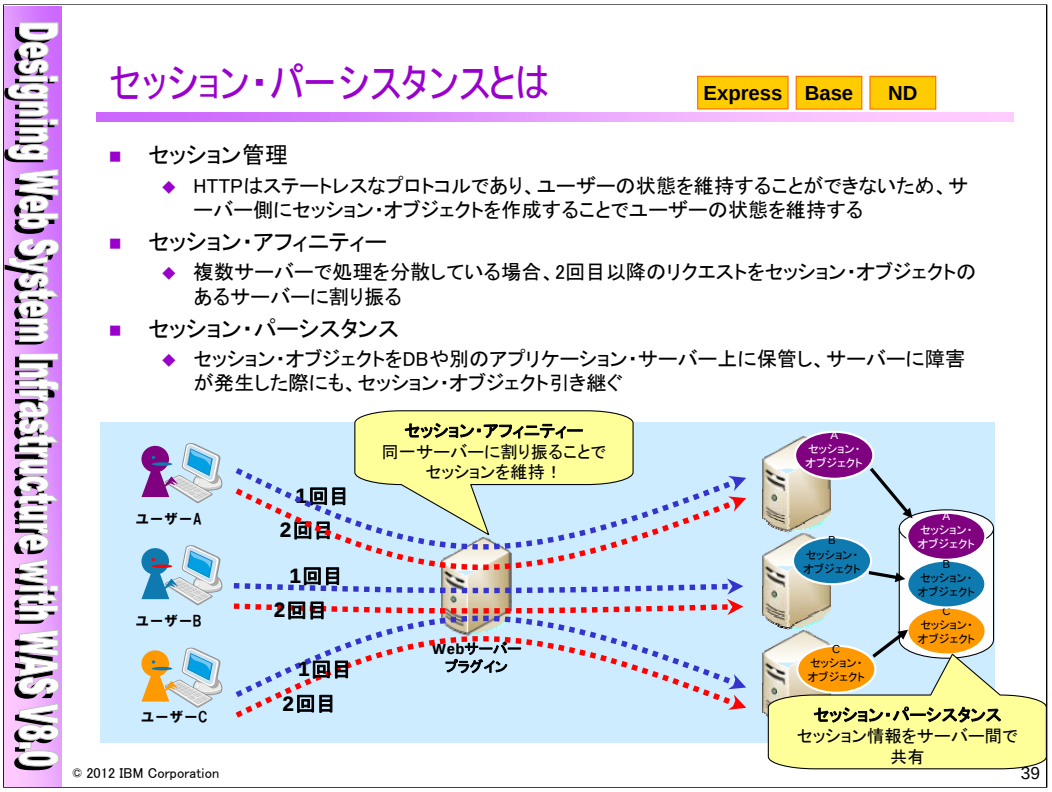


図はWASが提供する高可用性機能、高パフォーマンス機能をまとめた図になります。まず高可用性、高パフォーマンス両方を実現する機能としてクラスター機能があります。複数のアプリケーション・サーバー上で同じアプリケーションを稼働させることにより、1台がダウンしても業務提供可能であり、また複数台で処理をおこなうことにより負荷分散も実現することができます。

その他高可用性機能としては、セッション情報を別の場所に保管させることにより障害時でもセッションの維持が可能なセッション・パーシスタンス機能、シングルトン・サービスに対してフェールオーバー機能を提供するHAマネージャーがあります。

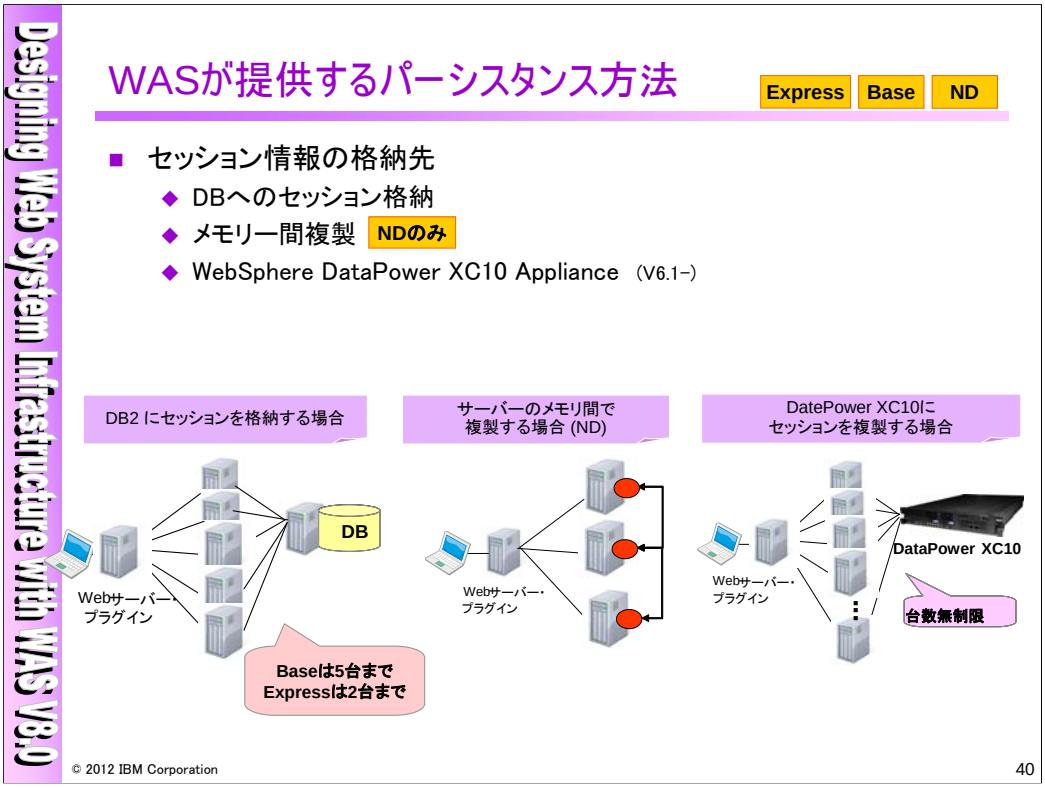
また、高パフォーマンスに対するその他機能としては、サーブレットやJSPといった動的ページに対してもキャッシュ保持可能なキャッシング機能があります。


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
-WASが提供する高可用性・高パフォーマンス機能
-セッション・パーシスタンス
-HAマネージャー
-キャッシュ
© 2012 IBM Corporation
38



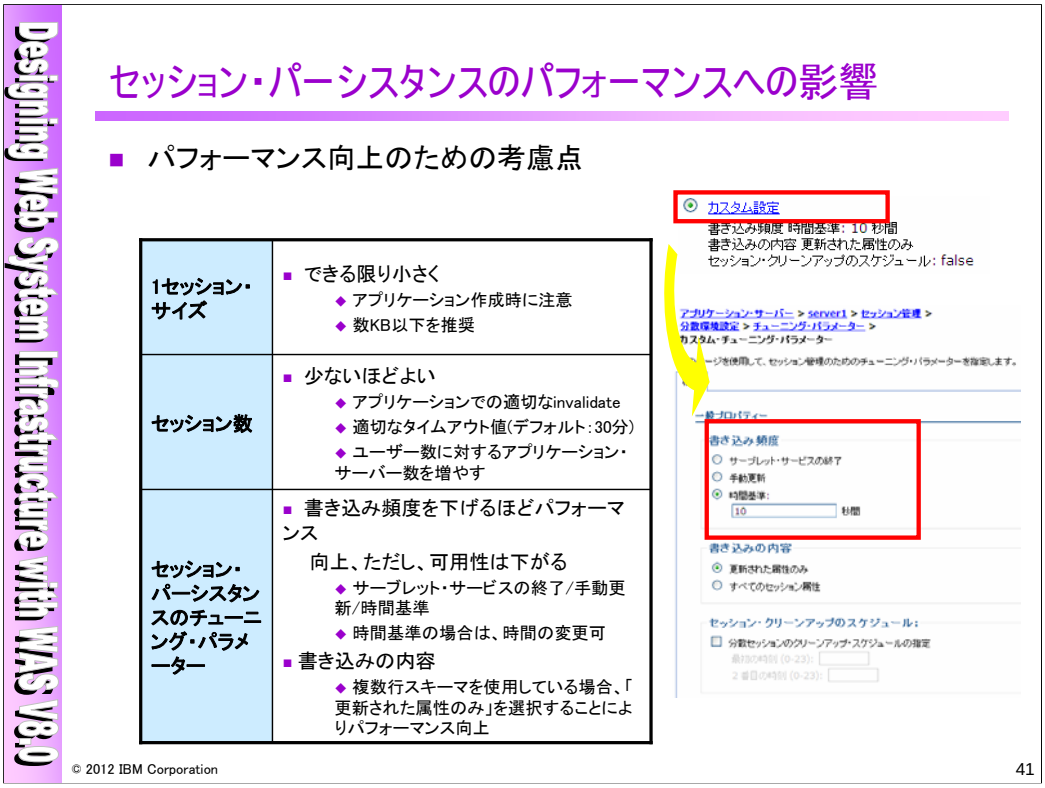
Webアプリケーションでは、例えばショッピング・サイトの買い物かごなど、ユーザーの状態を保持したいという要件がある場合が多いです。しかし、HTTPはステートレスなプロトコルであるため、ユーザーの状態を保持することができません。そこで、Webアプリケーションでは、サーバー側にセッション・オブジェクトを作成しユーザーと関連付けることでユーザーの状態を保持します。このような仕組みをセッションを管理と呼びます。

セッション・パーシスタンスとは、セッション情報(セッション・オブジェクト)のコピーを異なる場所に保管することで、アプリケーション・サーバーの障害時にもセッションの維持を可能にする機能です。WASV7では、セッション情報をDBへ保管するSession DBと、他のアプリケーション・サーバーのメモリー上へセッション情報を保管するメモリー間複製の2つの方法を選択可能です。



セッションのパーシスタント/共有先としては、DB、メモリー間複製、DataPower XC10の3つの方法があります。

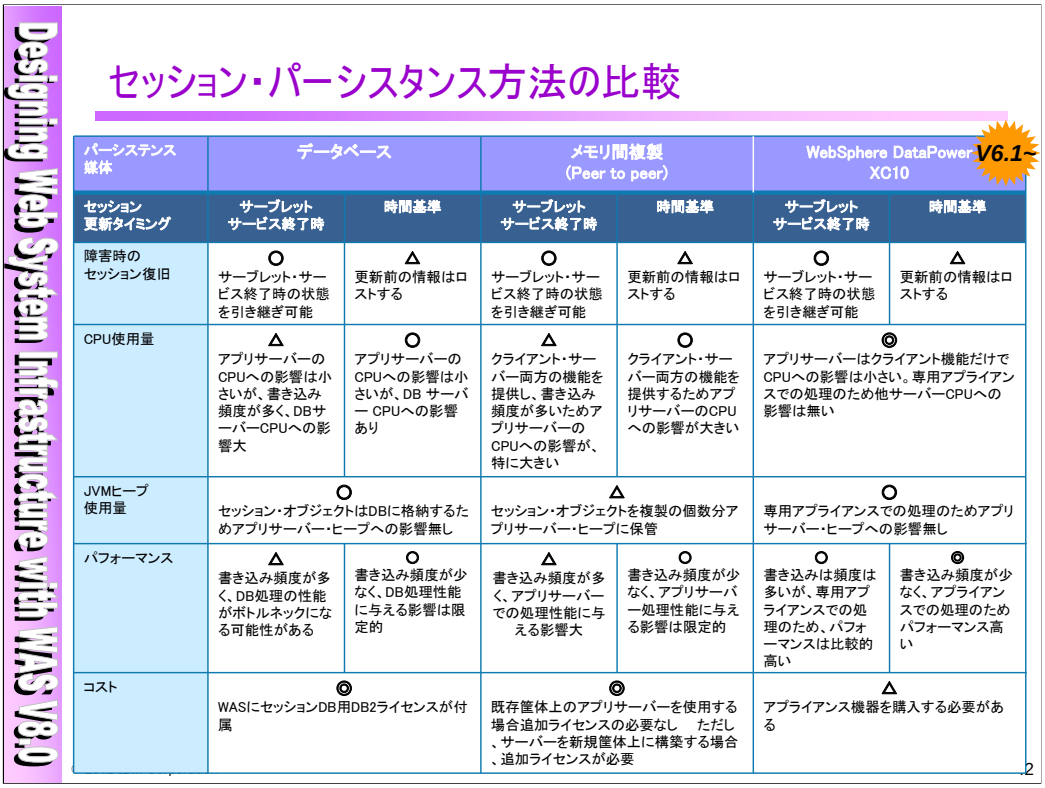
それぞれの方法の比較についてはこの後説明します。



パフォーマンスを向上するためには、適切なセッション管理を行い、セッション・オブジェクトをなるべく小さく、そして少なくする必要があります。

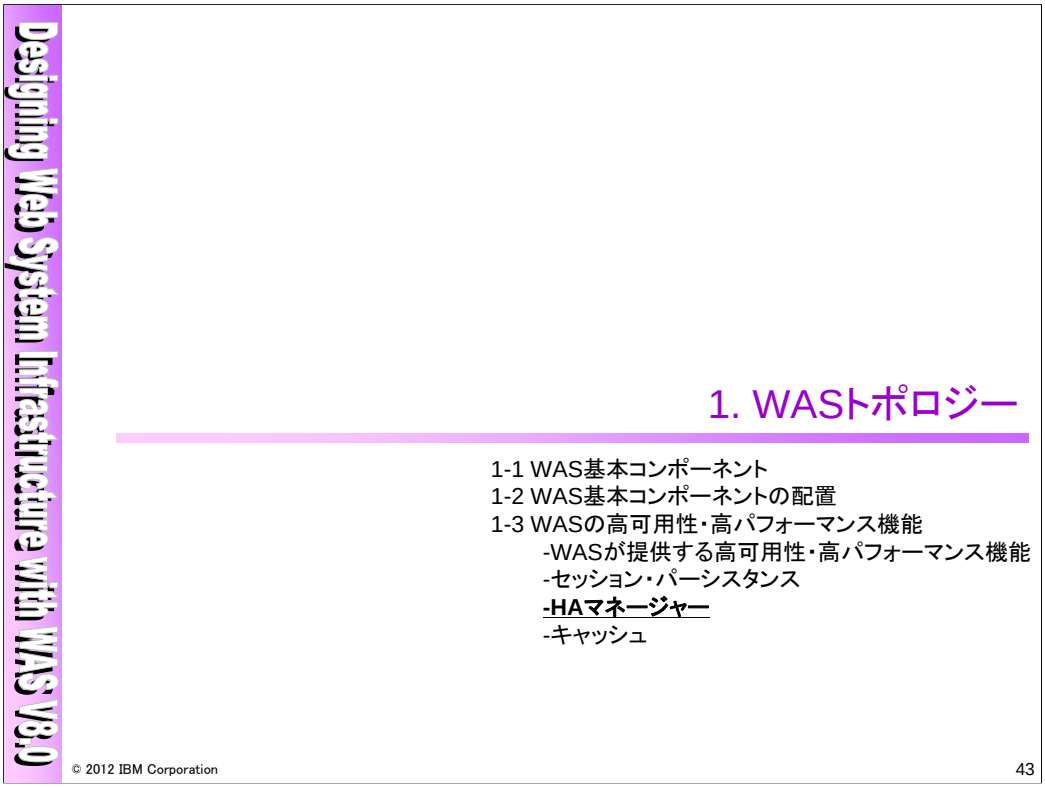
またセッション・パーシスタンスの書き込み頻度を低くすればパフォーマンスが向上しますが、可用性は低下しますので、要件にあった設定を実施して下さい。

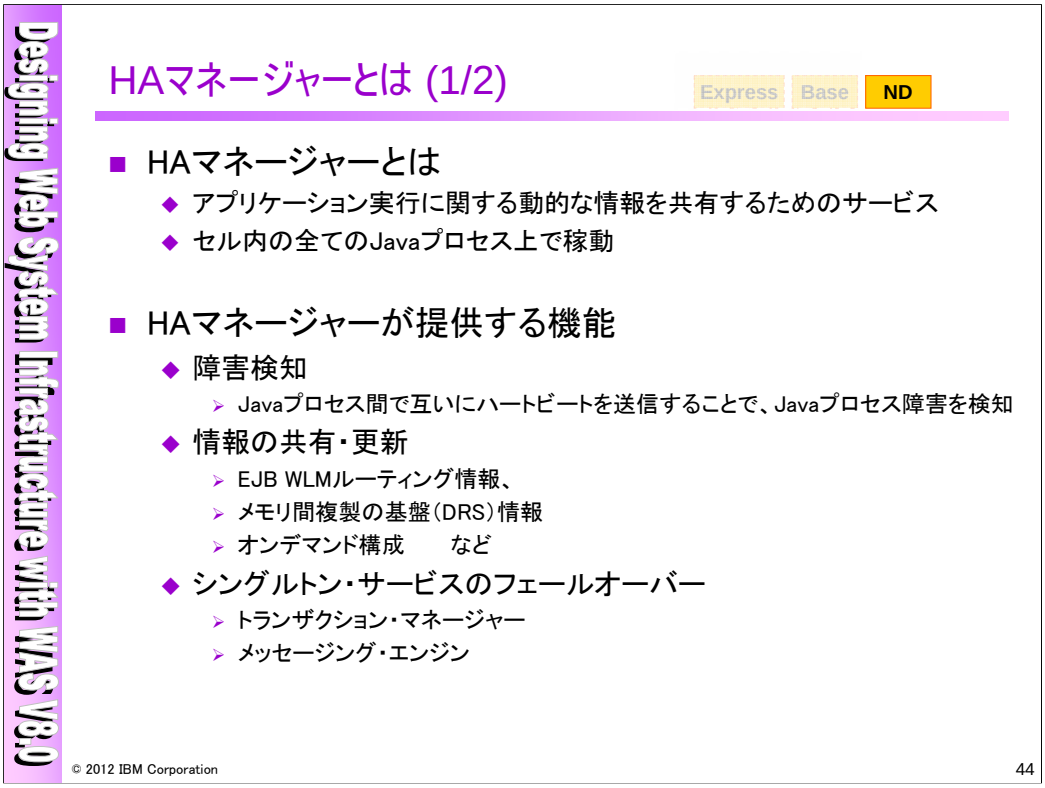
またSession DBでセッション・パーシスタンスを構成する場合、WASで指定できるデータベースの構成パラメータとして単一行スキーマと複数行スキーマがあります。デフォルトの設定では単一行スキーマを使用する構成になっており，1つのセッション・オブジェクトのデータはセッションIDをキーとするデータベーステーブルの1行に格納されます。複数行スキーマを使用する場合には、セッション・オブジェクト内の各アトリビュートはそれぞれ別々の行に格納され，各セッション当たりの保管データ量を増やすことができます。通常複数行スキーマはセッションに格納するオブジェクトのサイズが2MBを超えた場合に使用する設定でありましたが、セッションのアトリビュート(属性)の数が1つだけの場合にはセッション・オブジェクトのサイズに関わらず、複数行スキーマを使用したほうがパフォーマンスが良いことがわかっています。よってセッション・パーシスタンスを使用する際のチューニング項目として、こちらもご考慮ください。



パーシステンス媒体	データベース		メモリ間複製 (Peer to peer)		WebSphere DataPower XC10	
セッション更新タイミング	サーブレットサービス終了時	時間基準	サーブレットサービス終了時	時間基準	サーブレットサービス終了時	時間基準
障害時のセッション復旧	○ サーブレット・サービス終了時の状態を引き継ぎ可能	△ 更新前の情報はロストする	○ サーブレット・サービス終了時の状態を引き継ぎ可能	△ 更新前の情報はロストする	○ サーブレット・サービス終了時の状態を引き継ぎ可能	△ 更新前の情報はロストする
CPU使用量	△ アプリサーバーのCPUへの影響は小さいが、書き込み頻度が多く、DBサーバーCPUへの影響大	○ アプリサーバーのCPUへの影響は小さいが、DB サーバー CPUへの影響あり	△ クライアント・サーバー両方の機能を提供し、書き込み頻度が多いためアプリサーバーのCPUへの影響が、特に大きい	○ クライアント・サーバー両方の機能を提供するためアプリサーバーのCPUへの影響が大きい	◎ アプリサーバーはクライアント機能だけでCPUへの影響は小さい。専用アプライアンスでの処理のため他サーバーCPUへの影響は無い	
JVMヒープ使用量	○ セッション・オブジェクトはDBに格納するためアプリサーバー・ヒープへの影響無し		△ セッション・オブジェクトを複製の個数分アプリサーバー・ヒープに保管		○ 専用アプライアンスでの処理のためアプリサーバー・ヒープへの影響無し	
パフォーマンス	△ 書き込み頻度が多く、DB処理の性能がボトルネックになる可能性がある	○ 書き込み頻度が少なく、DB処理性能に与える影響は限定的	△ 書き込み頻度が多く、アプリサーバーでの処理性能に与える影響大	○ 書き込み頻度が少なく、アプリサーバー処理性能に与える影響は限定的	○ 書き込みは頻度は多いが、専用アプライアンスでの処理のため、パフォーマンスは比較的高い	◎ 書き込み頻度が少なく、アプライアンスでの処理のためパフォーマンス高い
コスト	◎ WASにセッションDB用DB2ライセンスが付属		◎ 既存筐体上のアプリサーバーを使用する場合追加ライセンスの必要なし ただし、サーバーを新規筐体上に構築する場合、追加ライセンスが必要		△ アプライアンス機器を購入する必要がある	

各パーシスタンス方法による違いを項目毎にまとめました。
それぞれ考慮点、利点などがありますので、この表を参考に最適な方法を選択してください。
いずれの方法を使用しても、使用しない場合に比較してパフォーマンスに影響が生じます。


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
-WASが提供する高可用性・高パフォーマンス機能
-セッション・パーシスタンス
-HAマネージャー
-キャッシュ
© 2012 IBM Corporation
43



HAマネージャーは、アプリケーション実行に関する動的な情報を共有するためのサービスで、セル内の全てのJavaプロセス上で稼動します。HAマネージャーの機能は大きく分けて3つです。

(1)障害検知

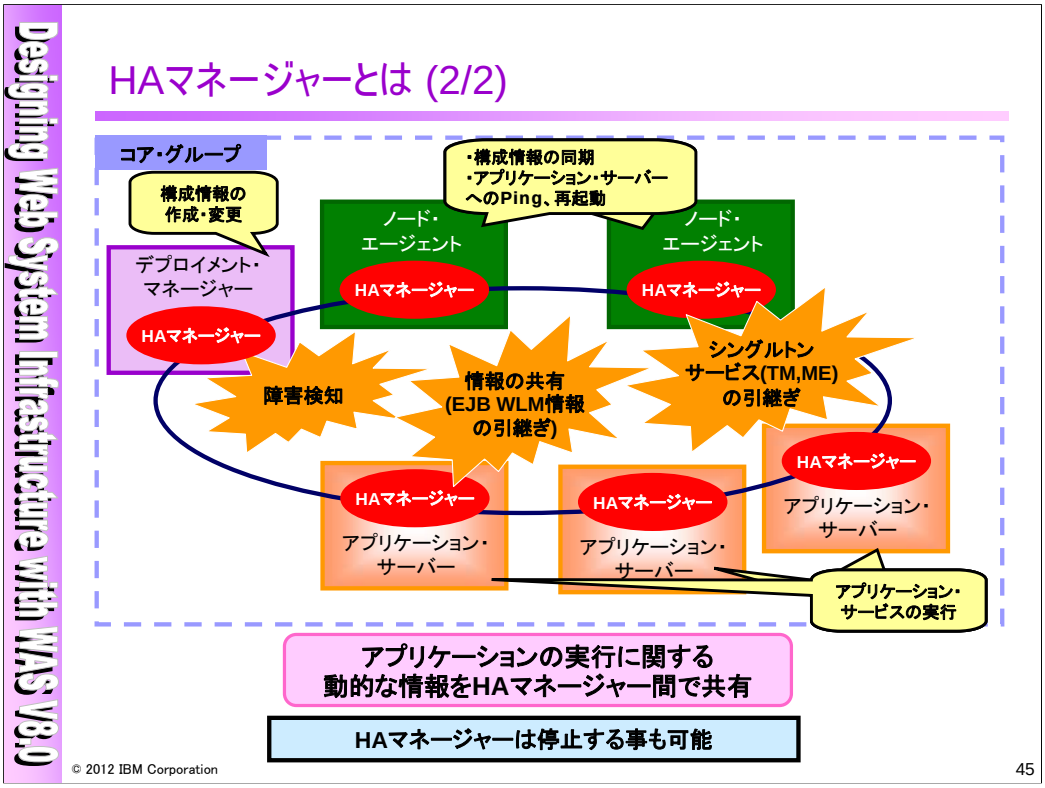
各Javaプロセス間で互いにハートビートを送信することで、Javaプロセス障害を検知し特定することができます。

(2)情報の共有・更新

アプリケーション稼働に必要な、EJB WLMルーティング情報、メモリ間複製の基盤であるDRS情報、ODC(オンデマンド構成)情報等を共有し、必要な場合は即時更新されます。

(3)シングルトン・サービスのフェールオーバー

基本的に1台のサーバー・インスタンスで実施しなければならないサービスのフェールオーバーをサポートします。具体的には、トランザクション・ログに対応するトランザクション・マネージャー・サービスのフェールオーバーによるインダウトトランザクションのピア・リカバリーと、メッセージング・エンジンのフェールオーバーがサポートされます。



コア・グループとはセル内のJVMをグルーピングしたもので、HA機能の設定の単位となります。コア・グループ内でJVMプロセスを監視するため、全メンバー間のネットワーク通信が必要となり、デフォルトでは"DefaultCoreGroup"が使用されます。

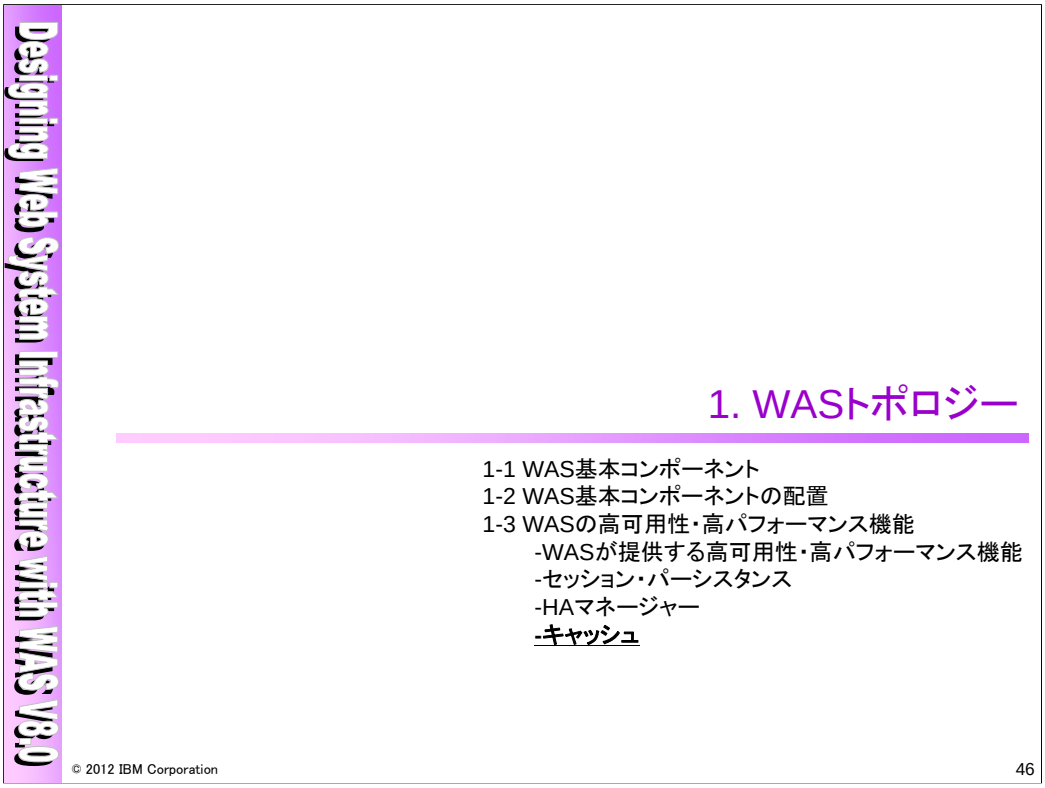
コア・グループの中のすべてのJVMはお互いを監視し、JVM障害の検知、情報の共有・更新、シングルトン・サービスのフェールオーバー機能を実現します。

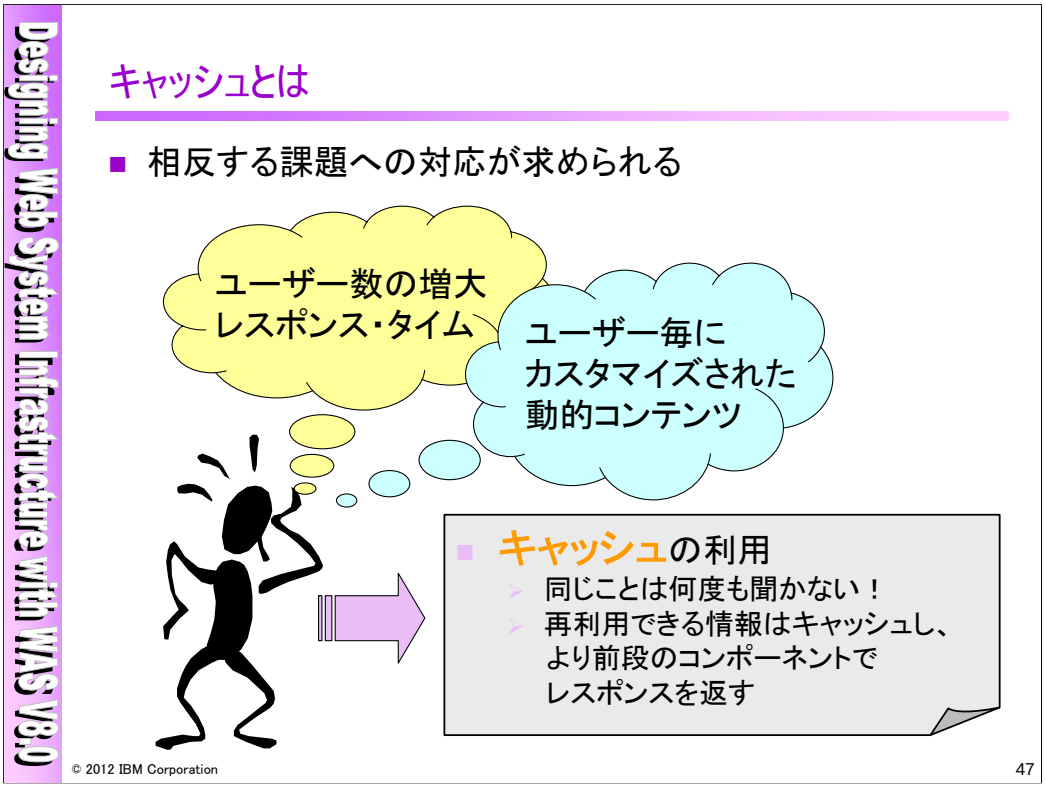
まだ、HAマネージャーの機能は停止することも可能です。

詳細は下記のページをご参照ください。

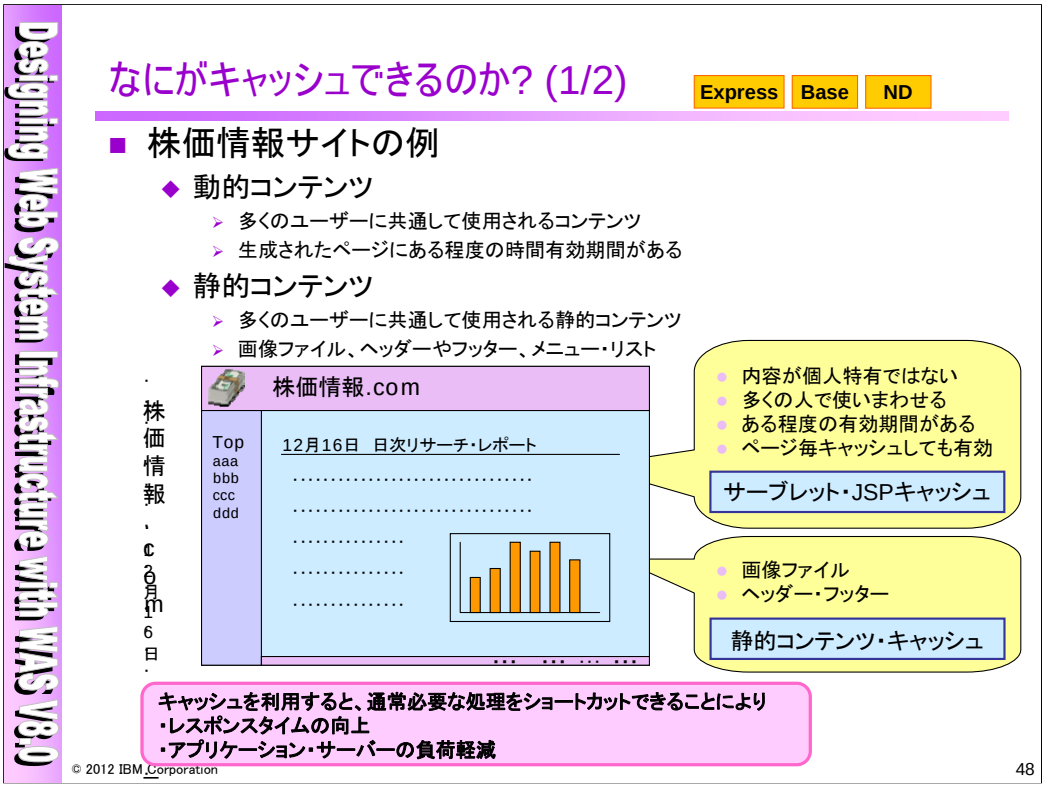
大規模なセル環境におけるHAマネージャー設定の注意点

http://www.ibm.com/support/docview.wss?uid=jpn1J1001545


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
-WASが提供する高可用性・高パフォーマンス機能
-セッション・パーシスタンス
-HAマネージャー
-キャッシュ
© 2012 IBM Corporation
46



高パフォーマンス機能として、キャッシュについて説明します。Webシステムにおけるパフォーマンスの代表的な指標はスループット(単位時間あたりの処理数)とレスポンス・タイム(応答時間)になります。キャッシュを使用することで、リクエストを前段のコンポーネントで返すことが可能です。これによりユーザー数増加に伴うWAS、DBなどの負荷を軽減することが可能となり(スループットの向上)、またレスポンス・タイムも向上します。

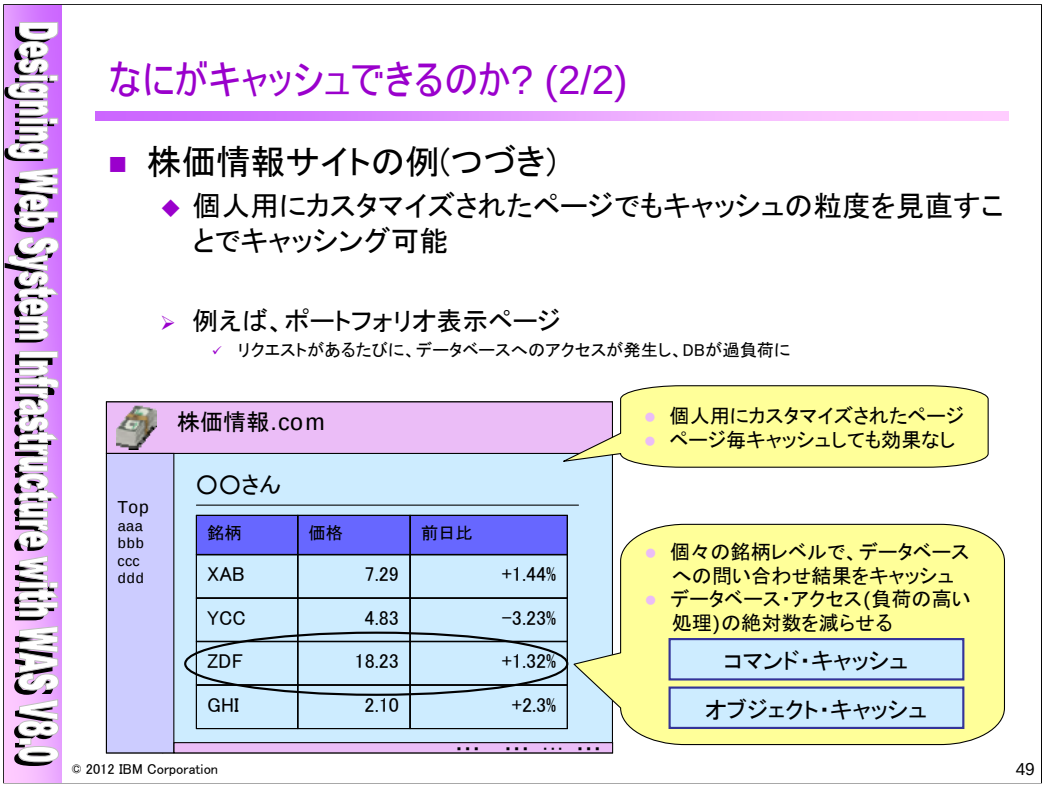


動的コンテンツでキャッシュの対象となるものは、多くのユーザーに対し共通なコンテンツやある程度有効期間を持つコンテンツになります。

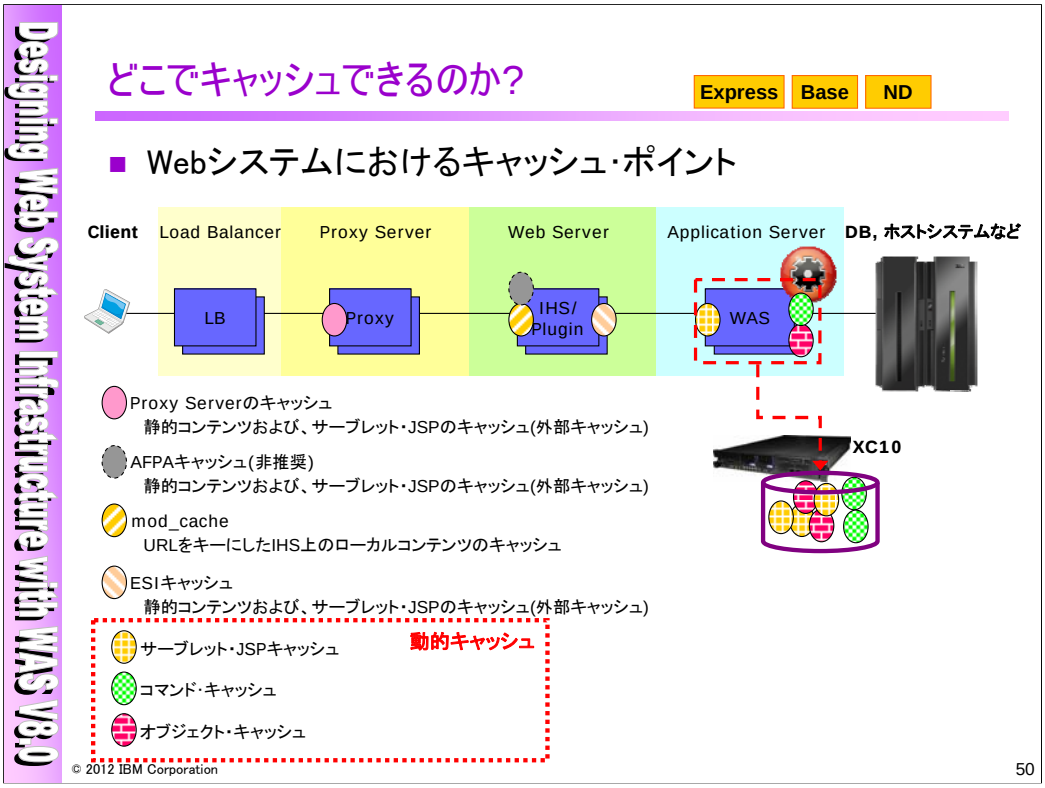
WASの動的キャッシュでは、サーブレット・キャッシュ、コマンド・キャッシュ、オブジェクト・キャッシュの3つの種類があります。

サーブレット・キャッシュでは、GETやPOST要求のURLやパラメーター、Cookie情報などを基にサーブレットやJSPの実行結果をキャッシュします。キャッシュは外部キャッシュとして、IHSやProxy Serverへ配置することが可能です。

サーブレット・キャッシュは、コマンド・キャッシュ、オブジェクト・キャッシュに比べ設定が容易です。どのページをどの期間保管するかを決定し、cachespec.xmlを設定する必要はありますが、プログラムを変更する必要はありません。

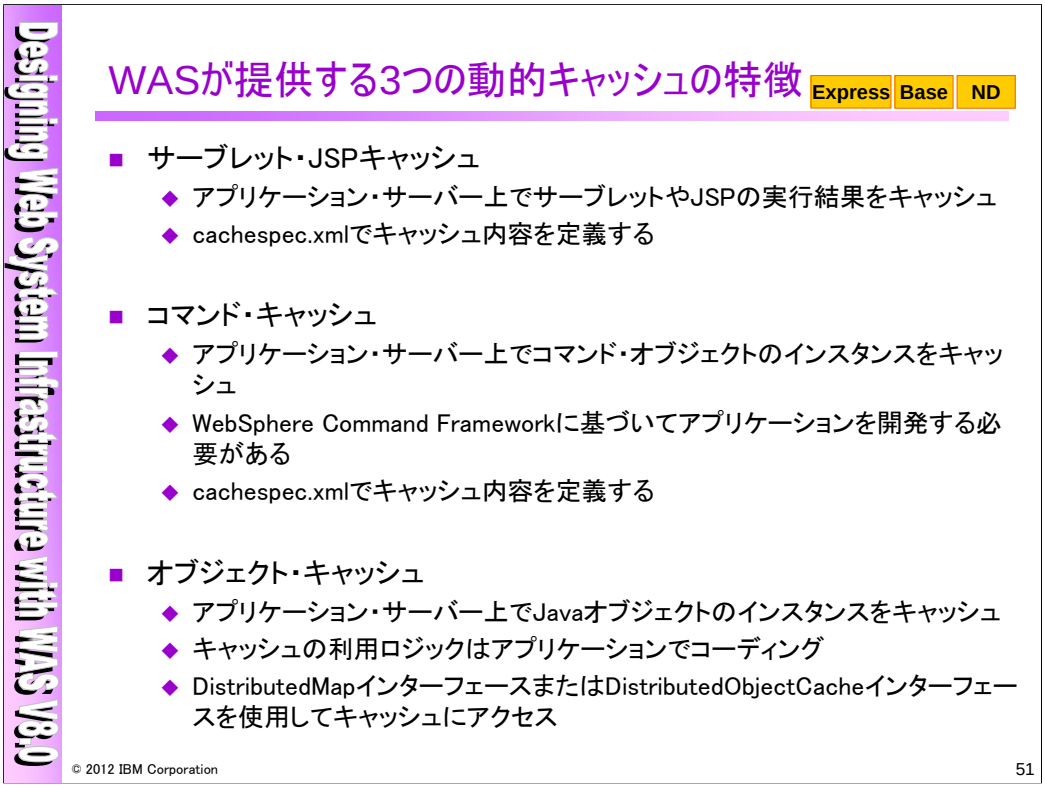


WASではページ単位のキャッシュだけではなく、コマンド・キャッシュやオブジェクト・キャッシュといった、個々の結果についてもキャッシュが可能です。例えば、個人向けの株価のページであっても、株価の値をキャッシュすることで、他の人のページでも同様の株価を表示する際にキャッシュから返す事が可能です。ただしサーブレット・キャッシュはアプリケーション側での対応が必要ないのに対し、コマンド・キャッシュおよびオブジェクト・キャッシュではこれらを使用することを前提にWebSphere独自APIを使用したアプリケーションのプログラミングを行う必要があります。



WASを使用したWebシステムにおいては、Caching Proxy、IHS、Webコンテナー、EJBコンテナーなど、さまざまなポイントでキャッシュが可能です。基本的には、よりクライアント側に近いコンポーネントにキャッシュさせることで、レスポンスタイムが向上します。ここではWAS上のキャッシュであるサーブレット・JSPキャッシュ、コマンド・キャッシュ、オブジェクト・キャッシュについて取り上げます。XC10はこのWAS上のキャッシュのリモートキャッシュとして機能します。

他にも、プロキシ・サーバーやWebサーバー上にもキャッシュを配置することができます。静的コンテンツだけでなく、サーブレット・JSPキャッシュを配置することができ、これは外部キャッシュと呼ばれます。



WASの動的キャッシュでであるサーブレット・キャッシュ、コマンド・キャッシュ、オブジェクト・キャッシュの特徴をまとめます。

サーブレット・JSPキャッシュ：

サーブレットやJSPの実行結果を保存します。サーブレット・JSPキャッシュでは同じサーブレットに対するリクエストであっても、指定したパラメータやCookie情報毎に区別してキャッシュを生成することができます。管理コンソールおよびcachespec.xmlでの設定で使
用可能です。

コマンド・キャッシュ：

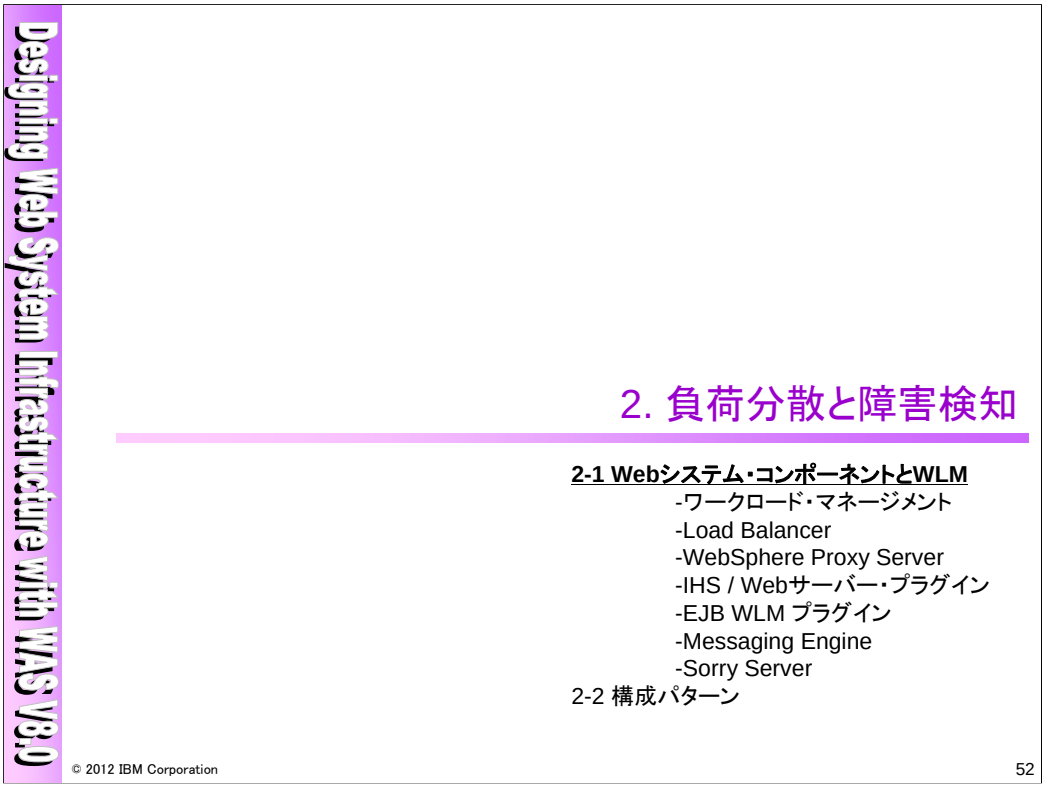
コマンド・オブジェクトの実行結果をキャッシュします。

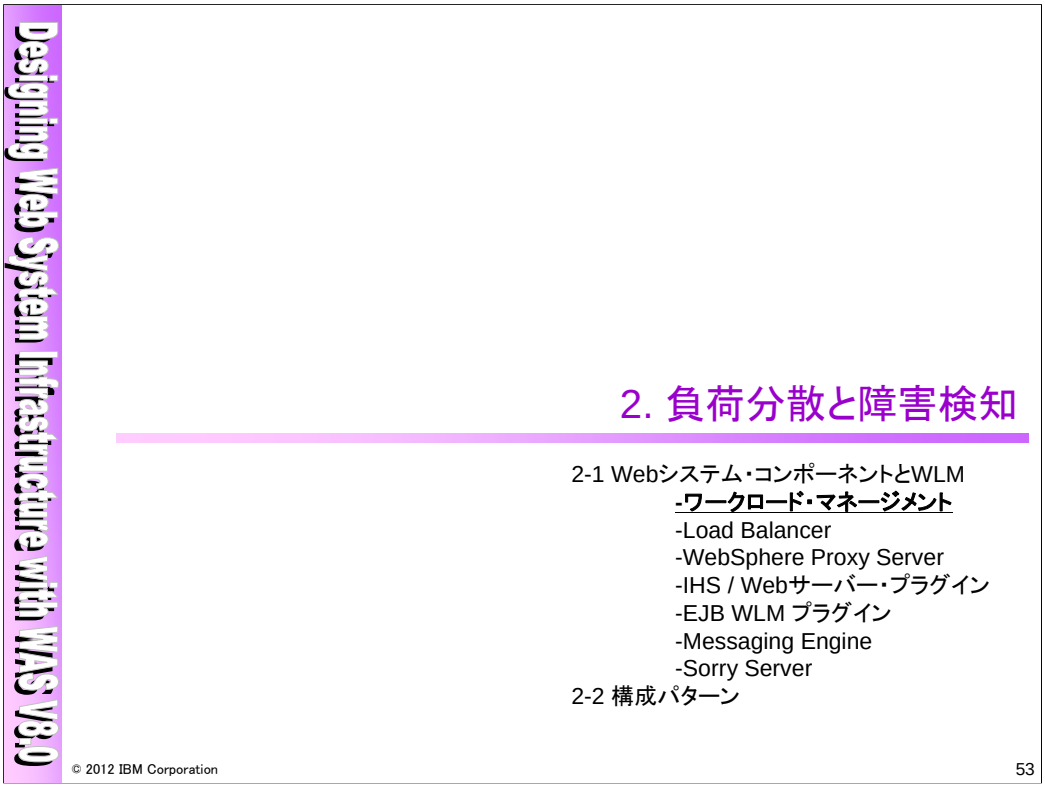
WebSphere Command Frameworkに基づいてアプリケーションを実装しcachecpec.xmlの設定によりキャッシュを行います。

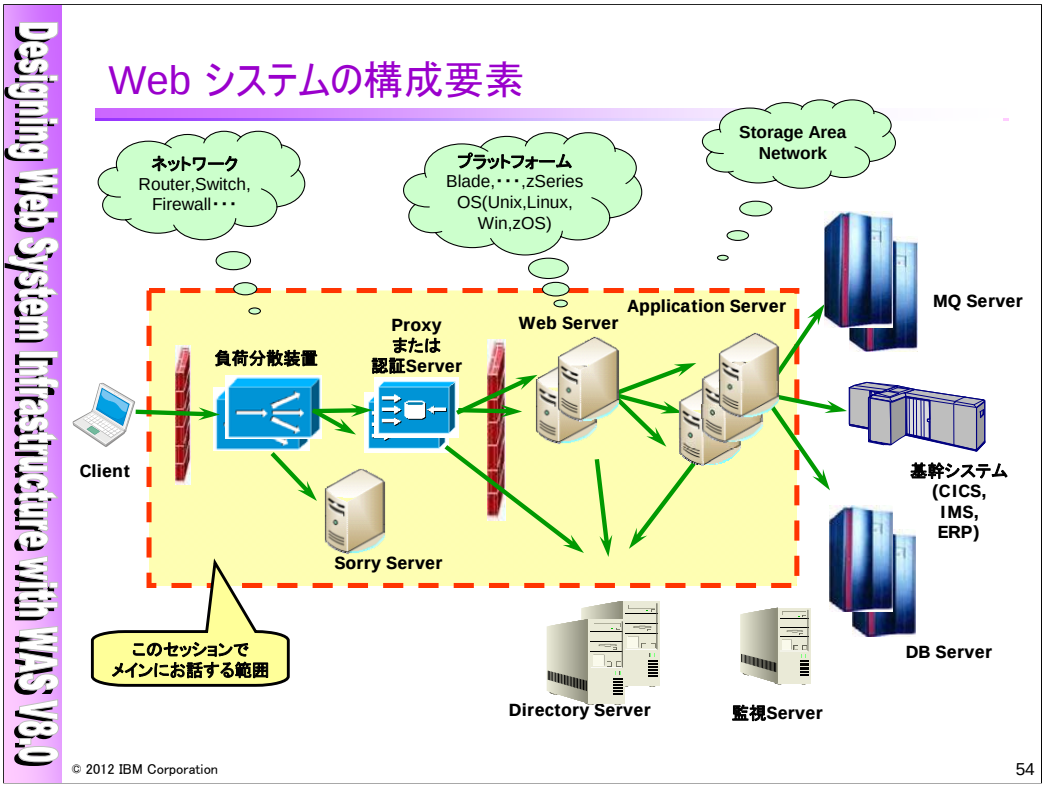
オブジェクト・キャッシュ：

オブジェクト・キャッシュはユーザーがアプリケーション・ロジックの中でキャッシュを利用するようにロジックを書いて利用します。

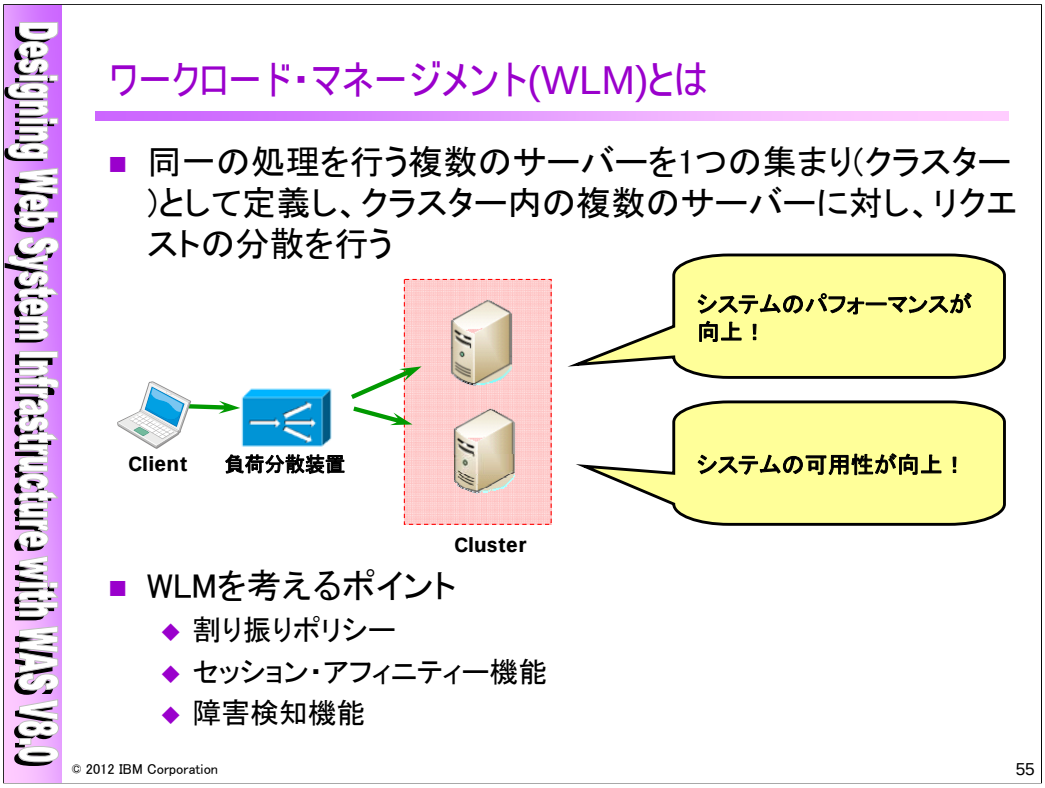
com.ibm.websphere.cache.DistributedMapインターフェースを使用してキャッシュ・エントリーの保管および取出しを行います。このDistributedMapインターフェースはjava.util.Mapインターフェースを拡張していますので、使い方が非常に分かりやすいものになっています。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
-ワークロード・マネージメント
-Load Balancer
-WebSphere Proxy Server
-IHS / Webサーバー・プラグイン
-EJB WLM プラグイン
-Messaging Engine
-Sorry Server
2-2 構成パターン
© 2012 IBM Corporation
52


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 WebシステムコンポーネントとWLM
-ワークロード・マネージメント
-Load Balancer
-WebSphere Proxy Server
-IHS / Webサーバー・プラグイン
-EJB WLM プラグイン
-Messaging Engine
-Sorry Server
2-2 構成パターン
© 2012 IBM Corporation
53



図はWebシステム設計において考慮する必要のあるコンポーネントをまとめたものになります。本セッションでは赤線で囲っているLoad Balancer、プロキシー・サーバー、Webサーバー、アプリケーション・サーバーの設計項目について学びます。前半の章では、主にアプリケーション・サーバーについて学びました。後半は主にそれ以外のコンポーネントについて説明します。



Webシステムのトポロジーを考えるとき、ワークロード・マネージメント(WLM)という考えが重要になります。

複数のサーバーを一つの集まりとして定義したものをクラスターと呼びます。クラスター内の複数のサーバーに大してリクエストの負荷分散を行うことをWLMと呼びます。WLMの効果は主に以下の2つになります。

・負荷を分散し、システムのパフォーマンスを向上させる

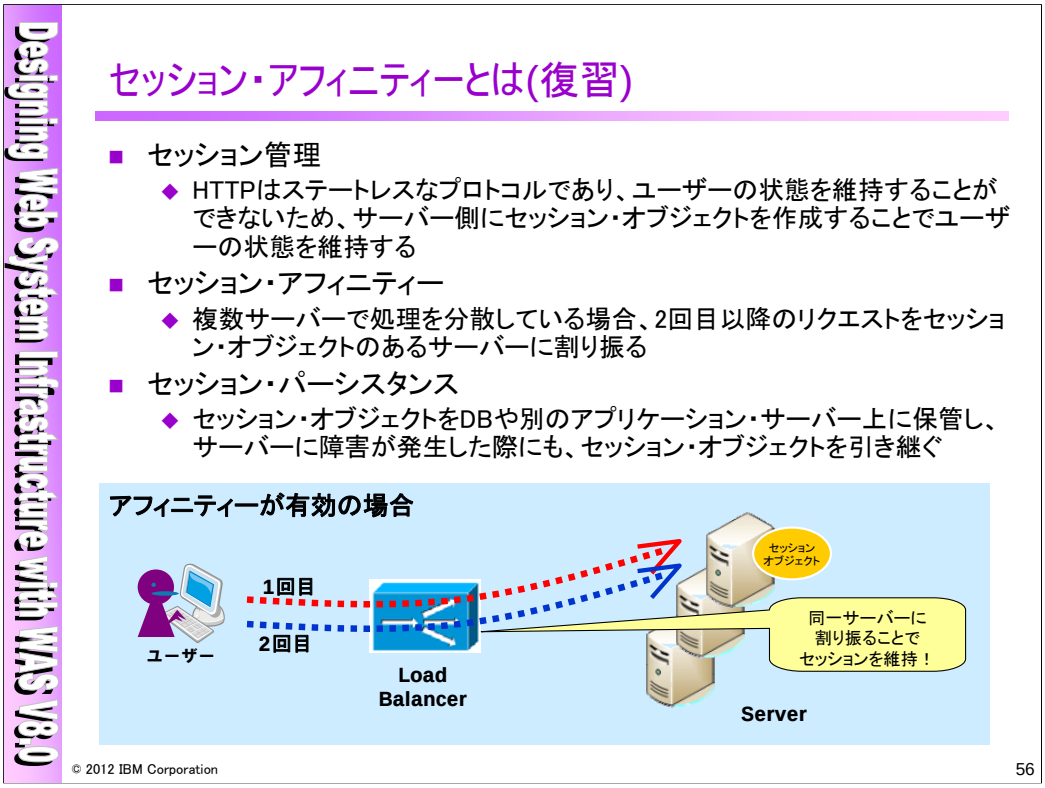
・障害を検知し、正常系へ割り振ることで可用性を向上させる

WLMを考えるとき、重要なポイントして、以下の3つがあります。

・割り振りポリシー

・セッション・アフィニティー機能

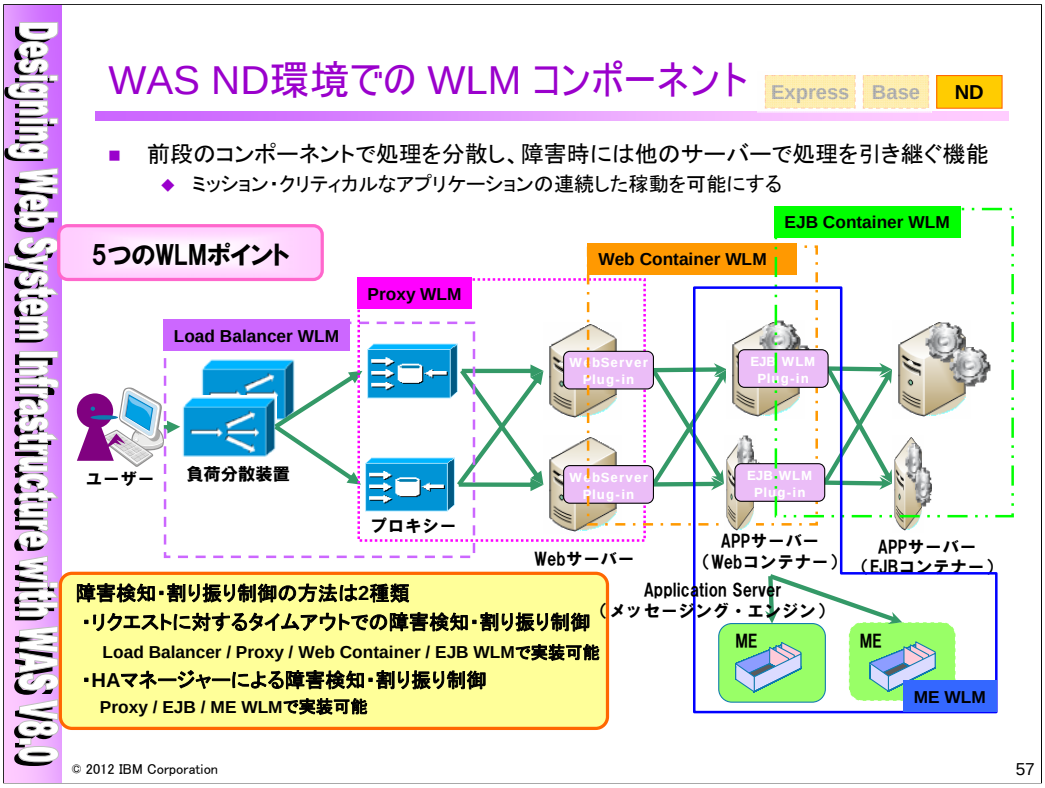
・障害検知機能



Webアプリケーションでは、例えばショッピング・サイトの買い物かごなど、ユーザーの状態を保持したいという要件がある場合が多いです。しかし、HTTPはステートレスなプロトコルであるため、ユーザーの状態を保持することができません。そこで、Webアプリケーションでは、サーバー側にセッション・オブジェクトを作成しユーザーと関連付けることでユーザーの状態を保持します。このような仕組みをセッションを管理と呼びます。

しかし、クラスターを構成し複数サーバーで処理を分散している場合、2回目以降のリクエストがセッション・オブジェクトのないサーバーに割り振られてしまうかもしれません。もしセッション・オブジェクトがないサーバーに割り振られてしまうと、買い物かごは空になってしまいます。そこで、2回目以降のリクエストを確実にセッション・オブジェクトのあるサーバーに割り振る仕組みが必要であり、このような仕組みをセッション・アフィニティーと呼びます。

なお、このセッションの前半では、「セッション・パーシスタンス」という言葉も登場しました。これら3つは似ているようですが、意味が異なりますので注意して下さい。



WAS ND環境でのWLMのポイントとして、以下があります。

・負荷分散装置 WLM

負荷分散装置からWebサーバー（Proxy Server）へのWLMです。

・Proxy WLM

Proxy ServerからWebサーバー（アプリケーション・サーバー）へのWLMです。

・Web Container WLM

WebサーバーからWebコンテナーへのWLMです。

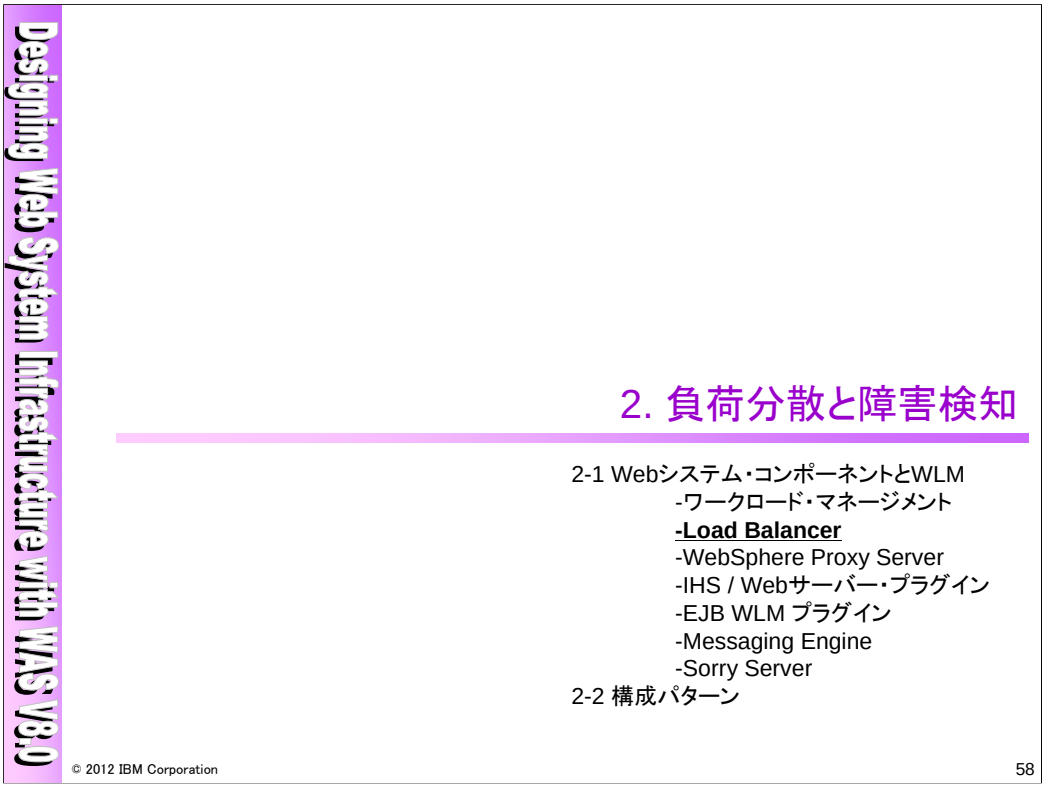
・EJB Container WLM

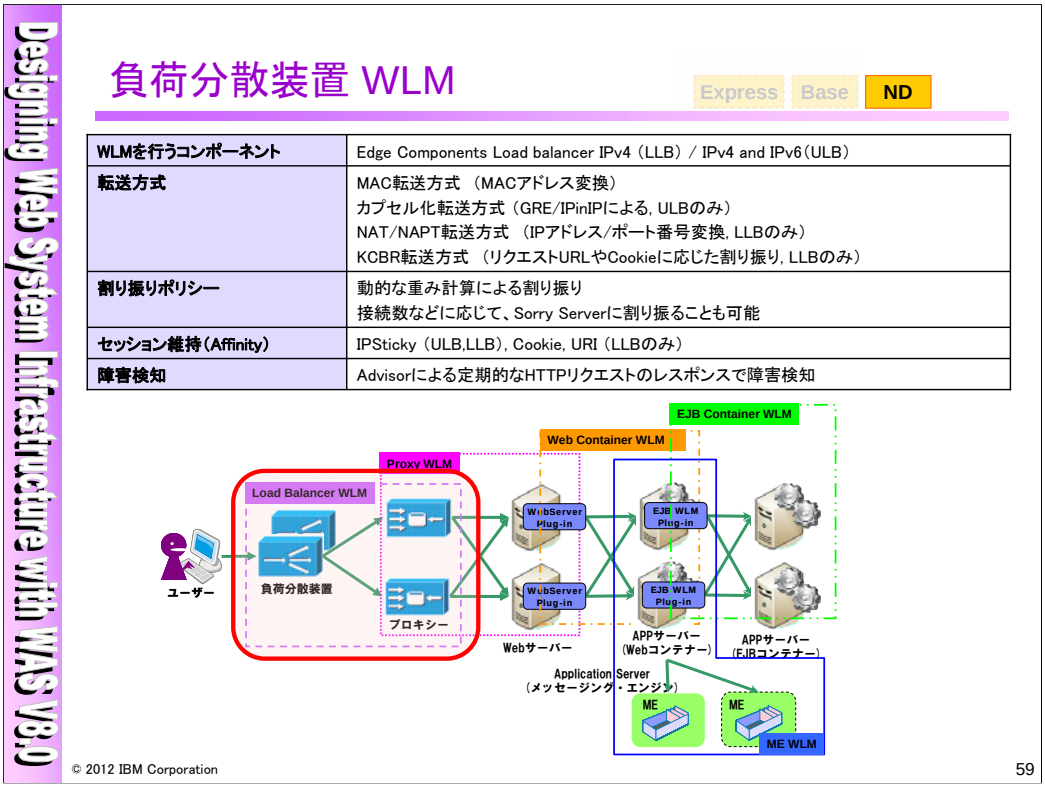
Webコンテナー（EJBクライアント）からEJBコンテナーへのWLMです。

・Message Engine WLM

WebコンテナーからMessage Engineへの WLMです。

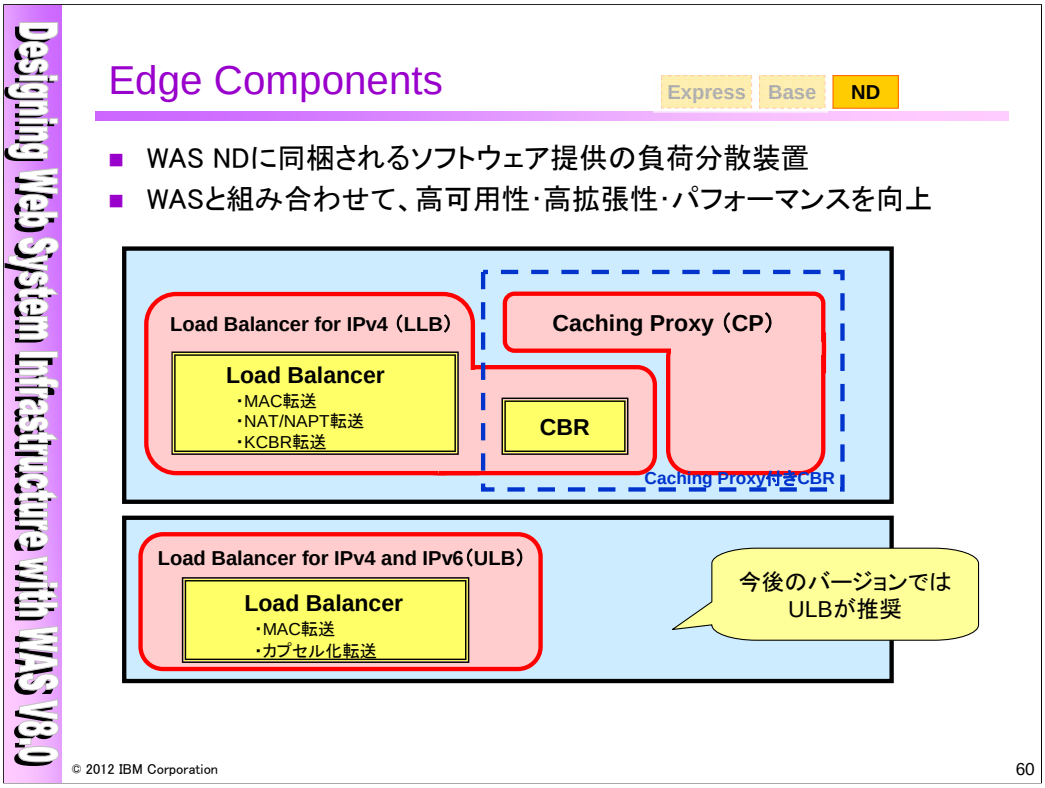
各WLMでの障害検知・割り振り制御の方法は大きく分けて2通りで、1つは実リクエストを投げてそのレスポンスにかかる時間で検知・割り振り制御する方法で、もう1つはProxy WLM / EJB WLM / ME WLMで使用できる、HAマネージャーにより障害検知・割り振りを制御する方法です。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
-ワークロード・マネージメント
-Load Balancer
-WebSphere Proxy Server
-IHS / Webサーバー・プラグイン
-EJB WLM プラグイン
-Messaging Engine
-Sorry Server
2-2 構成パターン
© 2012 IBM Corporation
58



負荷分散装置は、クライアントからのリクエストをWebサーバーに対してWLMを行うコンポーネントです。このコンポーネントは、WASV8.0 NDに含まれるEdge Componentsが該当します。

WASV8.0では、転送方式はMAC転送方式、セッション維持（Affinity）は接続元IPアドレスによるセッション固定方式であるIPStickyが推奨となります。また、割り振りポリシーは、Webサーバーへの接続数やレスポンスタイムを動的に計算し割り振り先を決定します。障害検知に関しては、Advisorによる定期的なHTTPリクエストのレスポンスを見て検知します。



Edge Componentsは、WAS ND に同梱されるソフトウェア提供の負荷分散装置で、主に

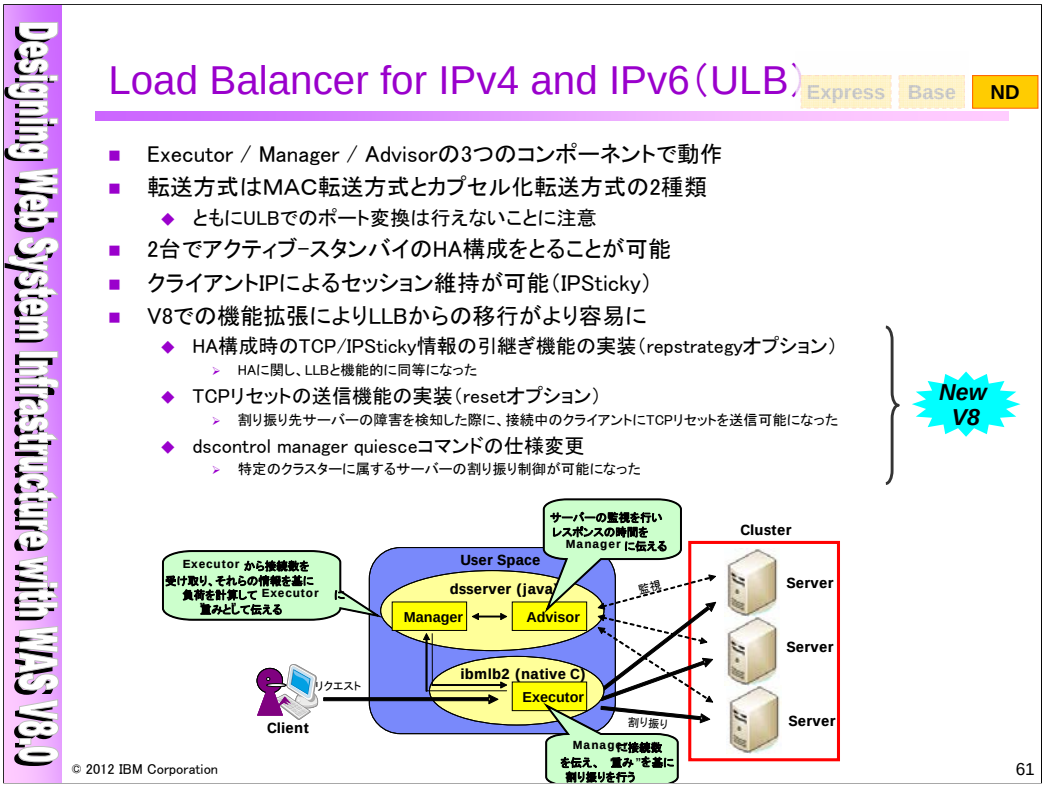
・カーネルスペースで動作する負荷分散装置のLoad Balancer for IPv4（Legacy Load balancerとも呼ばれます。以下LLBと記載します）

・主にLLBのCBRコンポーネントと組み合わせて使用するプロキシー・コンポーネントのCaching Proxy（以下CPと記載します）

・V7.0から提供されたユーザースペースで動作する負荷分散装置のLoad Balancer for IPv4 and IPv6（Userspace Load balancerとも呼ばれます。以下ULBと記載します。）

の3つのコンポーネントを有します。

V8.0で重要なアップデートとして、V6.1より非推奨であったLLBとCPが安定化に移行したこと、ULBで機能拡張が行われたことが挙げられます。



以下、LLB/CPからの移行が推奨されているULBについてご説明いたします。

ULBはExecutor/Manager/Advisorの3つのコンポーネントから成るシンプルな負荷分散装置で、定期的な監視リクエストをAdvisorから送信し、その結果をManagerが判断、Managerからの割り振り指示を受けてExecutorがリクエストを転送するという一連の動作をとります。アクティブ－スタンバイのHA機能をULBで有しており、ULBの冗長化も容易に行えます。

V8.0において重要なポイントとしては、

・TCP/HTTPセッションの引継ぎの実装

・TCPリセットの送信機能の実装

・dscontrol manager quiesceコマンドの仕様変更

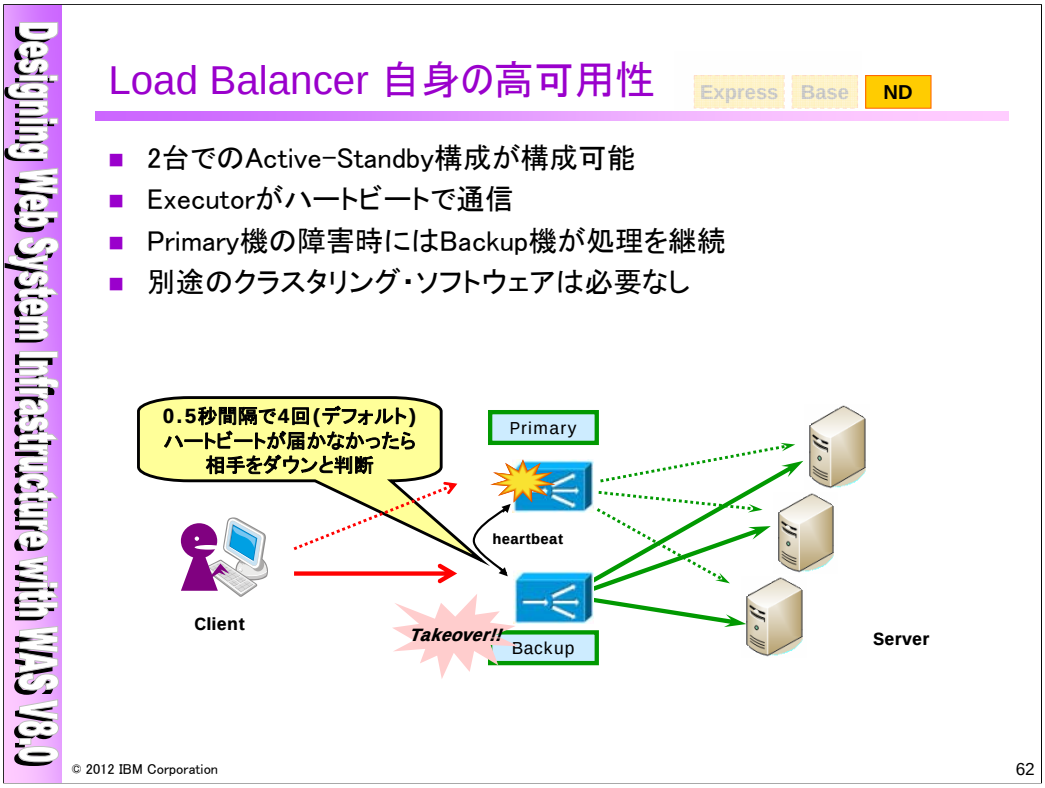
が挙げられ、これらの機能拡張によりLLBからULBの細かな機能差が解消され、LLBからの移行がより容易になりました。

ULBとLLBの差異につきましては、V7.0の資料になりますが、以下のリンク先資料をご参照ください。

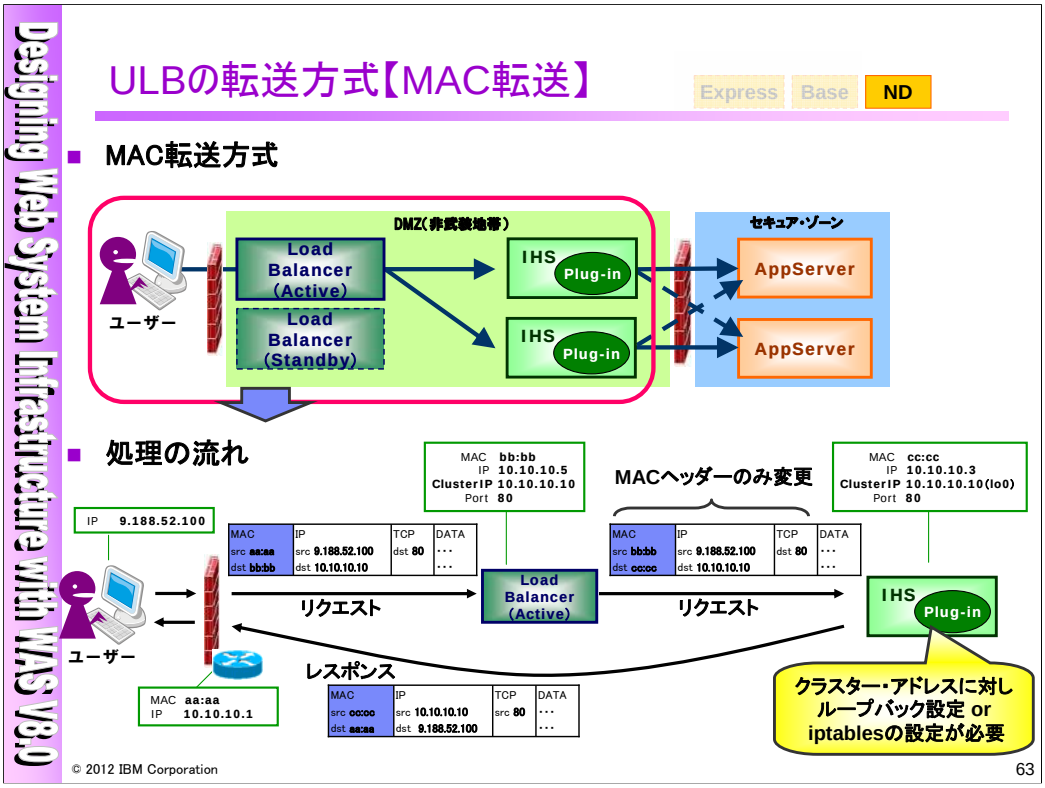
Edge Components V7 ガイド －「1.Overview」

http://www.ibm.com/developerworks/jp/websphere/library/was/edge7_guide/

V8からはデフォルトの導入ディレクトリが変わっています。



Load Balancerは割り振り先サーバーのクラスタリングを行うために必要なソフトウェアですが、Load Balancerが1台構成ではLoad Balancer自身がSPOFとなってしまいます。そこで、Load Balancerは自身の可用性のため、ソフトウェアレベルでのHigh Availability機能を実装しています。この機能により、Load BalancerはActive-Standby構成を取ることができます。Primary機とBackup機の間では、お互いに0.5秒間隔でハートビートを投げあい、相手マシンからハートビートが届かなかった場合は(デフォルト2秒間)、相手マシンに障害が発生したと判断し、自身がActiveマシンへと移行します。



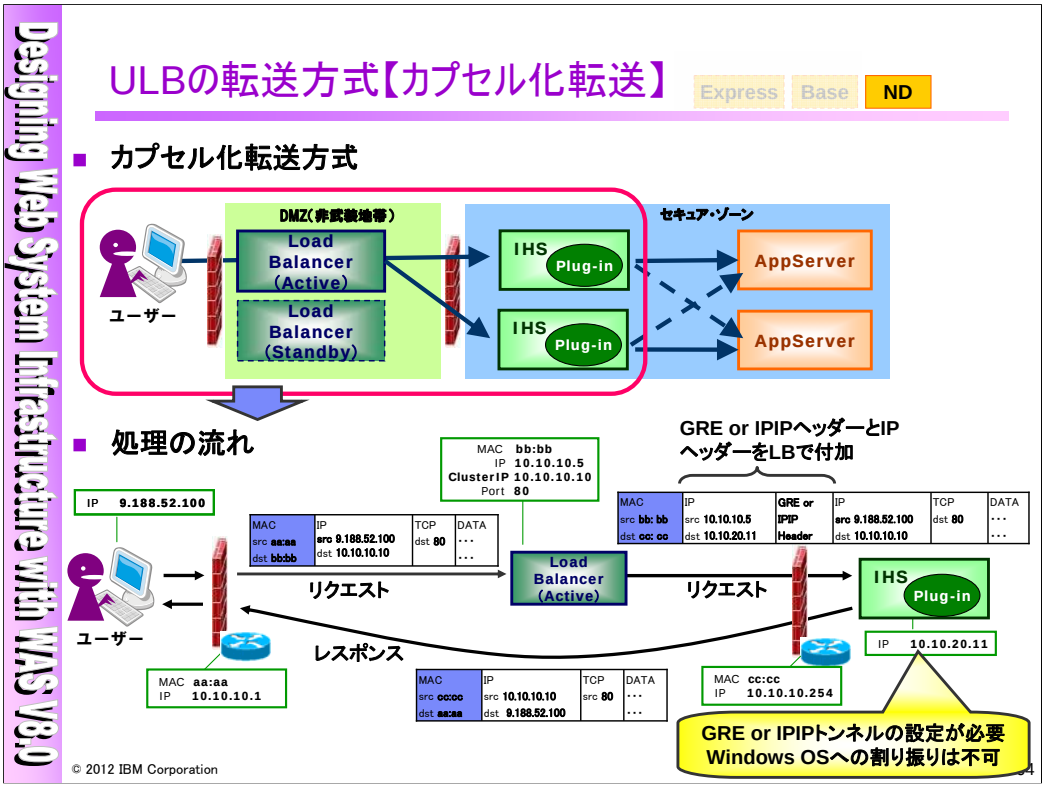
ULBでの転送方式であるMAC転送方式とカプセル化転送方式での基本トポロジーを以下説明します。

MAC転送方式での基本トポロジーは上記の通りです。MAC転送方式はユーザーのリクエストに対し宛先MACアドレスを変換して転送する方式ですので、同一セグメント内での割り振りに限定されます。ULBとアプリケーション・サーバーのセグメントを分けたい場合は、通常IHS+Plug-insをULBと同じセグメントに配置し、リモート構成をとる必要があります。

割り振り先サーバーからのレスポンスはLoad Balancerを経由せずにユーザーへ返る動作(DSR: Direct Server Return, 直接サーバー返答)になります。

割り振り先サーバーに対しては、ループバック・インターフェースにクラスター・アドレスをエイリアス、あるいはLinuxの場合はiptablesでクラスターアドレスに対しリダイレクトの設定を行う必要があります。

ULBの導入されているOSに割り振り先サーバー・プロセスを同居させる同居構成をとることはできないことにご注意ください。

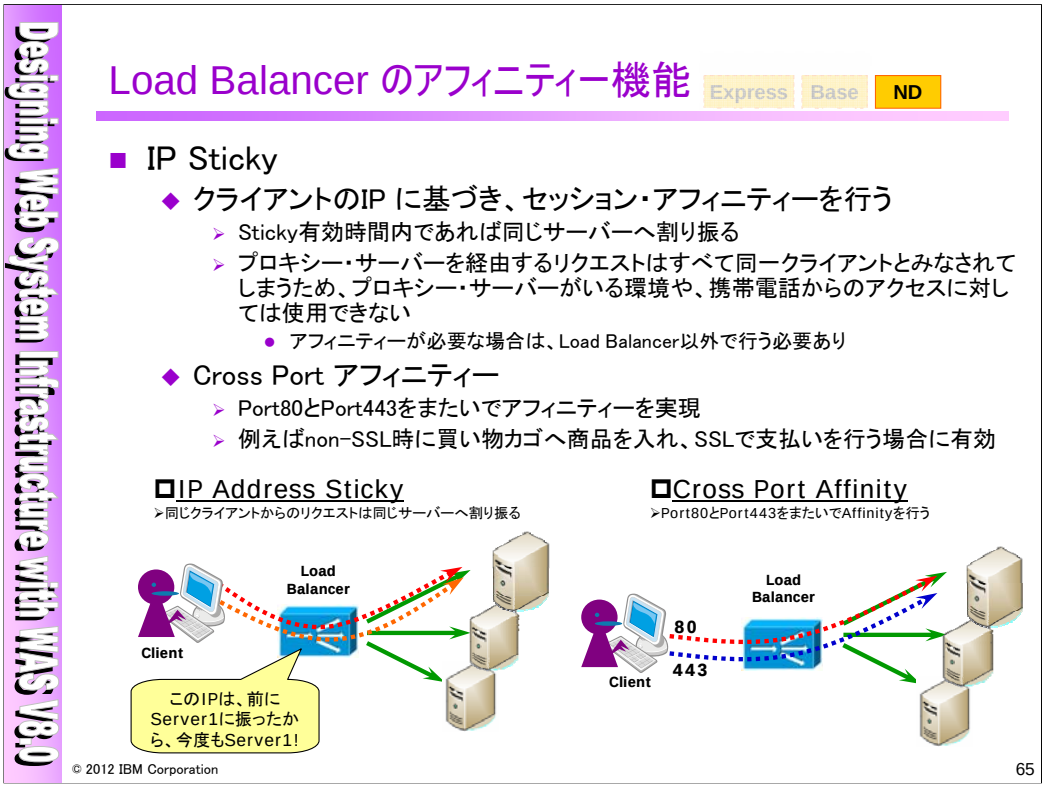


セグメントを越えた割り振りをULBで実施したい場合は、カプセル化転送方式を使用することで実現可能です。

カプセル化転送方式はRFCで定義されているIP in IPまたはGRE（Generic Routing Encapsulation ）の仕様に基づいてユーザーからの既設パケットとは別にIPヘッダーを加えて転送します。割り振り先サーバーでは、転送されたパケットを受け取り処理するために、GREあるいはIP in IPトンネルの設定を実施しておく必要があります。MAC転送方式と同じくLoad Balancerを経由せずにユーザーにレスポンスを返します。

カプセル化転送方式の考慮点として、Windows系OSのサーバーを割り振り先に指定することができないことが挙げられます。

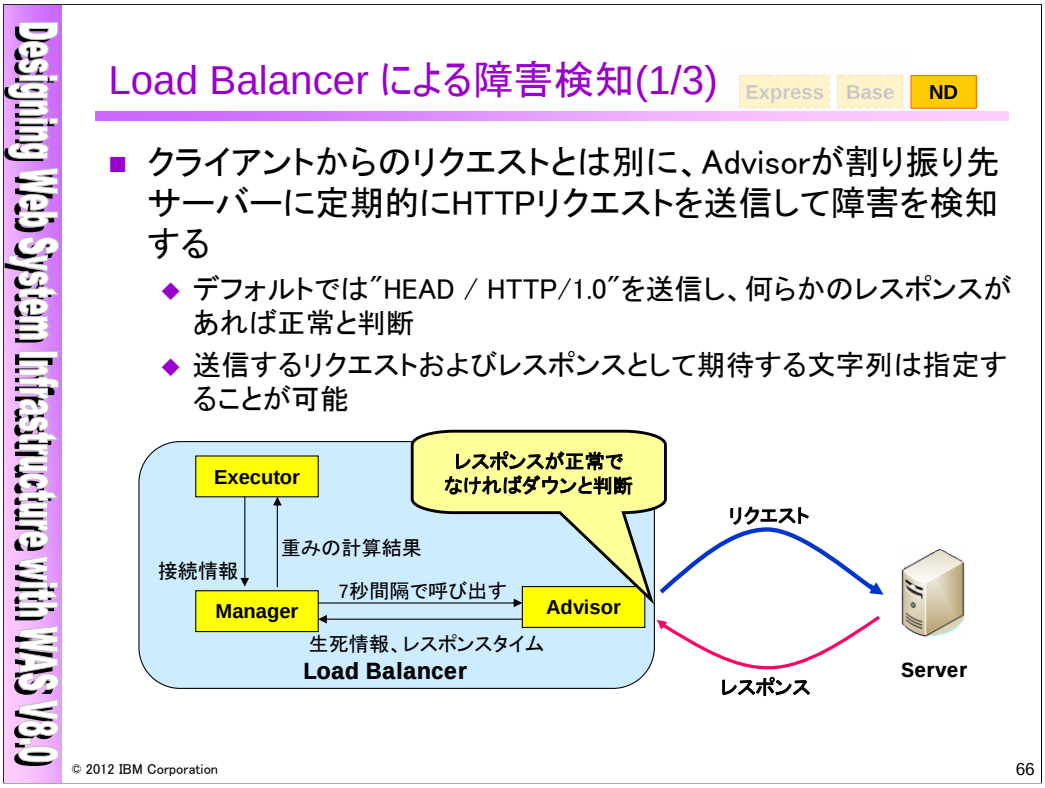
（参考）IP in IP Tunneling
http://www.ietf.org/rfc/rfc1853.txt
（参考）Generic Routing Encapsulation（GRE）
http://www.ietf.org/rfc/rfc2784.txt



ここからは、Load Balancerが提供するWLM機能を説明していきます。まずアフィニティー機能です。

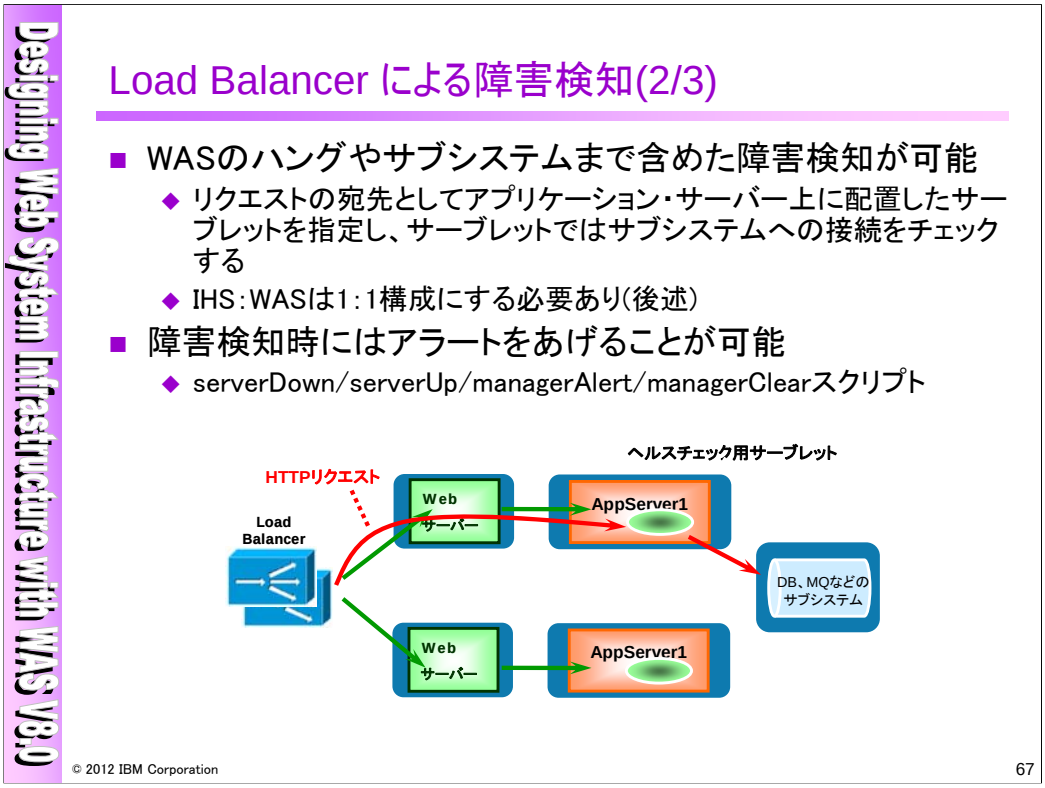
Load Balancerが提供するアフィニティー機能はIP Stickyと呼ばれる機能です。これはLoad Balancerがクライアントの IPアドレス毎にどの割り振り先サーバーに割り振ったかを記憶し、Sticky有効時間内であれば同一のサーバーに割り振り続けるというものです。IP Stickyの注意としては、Load Balancerの前段にプロキシー・サーバーがいる環境では、プロキシー・サーバーを経由するリクエストがすべて同じクライアントに見えてしまうために、割り振りが偏ってしまうことです。このため、このような環境ではIP Stickyは使用できません。また、携帯電話からのリクエストは、ラウンド・ロビン・プロキシーを経由してくるため、IP Stickyを用いてもセッション・アフィニティーを行うことができません。

関連する機能として、Cross Portアフィニティー機能があります。Load Balancerは基本的にはSticky情報を〈クラスター・アドレス〉:〈ポート〉の組み合わせの単位で持っています。このためLoad BalancerでStickyを効かせていても、80ポートへのリクエストと443ポートへのリクエストでは別々のサーバーに割り振られてしまう可能性があります。例えば、HTTPで商品を選択し精算時のみSSL通信を行なうようなアプリケーションの場合には、商品選択の間はセッションが維持できるが、購入ボタンを押すとセッション・オブジェクトのないサーバーに割り振られてログイン・ページに戻されてしまう可能性があります。この問題を解消するのがCross Portアフィニティです。複数のポート間でSticky情報を共有し、ポートをまたいでアフィニティを維持します。



ここからLoad Balancerによる障害検知の仕組みについて説明します。

Load Balancerの障害検知の方法は、Advisorが行います。Advisorは割り振り先サーバーにHTTPリクエストを送信します。デフォルトでは7秒毎に監視リクエストを送信し、レスポンスが正常に戻れば割り振り先を正常と判断します。デフォルトでは"HEAD / HTTP/1.0"を送信し、何らかのレスポンスがあれば正常と判断します。送信するリクエストおよびレスポンスとして期待する文字列は指定することが可能です。



Load Balancerによる障害検知の特徴として、WASのハングやサブシステムの障害まで含め、システムを一気通貫で監視できることです。 このような構成をとるためには、Advisorが送信するリクエストの宛先として、WAS上に配置したヘルスチェック用のサーブレットを指定します。このサーブレットでは、DBなどへの接続をチェックし、全てが正常な場合にのみAdvisorが期待するレスポンスを返すようにします。なお、このような構成をとるためには、IHSとWASを1:1に構成する必要があります。IHSとWASが1:Nで構成されている場合、サーブレットにリクエストを投げたとしても、どのWASにリクエストが振られるかわからないため、正しく割り振り先サーバーの障害を検知することができません。1:N構成の場合には、Load Balancerによる監視はIHSまでにするようにしてください。

もうひとつの特徴として、Load Balancerが割り振り先サーバーの障害を検知した際には、アラートをあげることができます。Load Balancerでは、障害を検知した際に以下の4つのスクリプトが実行されます。

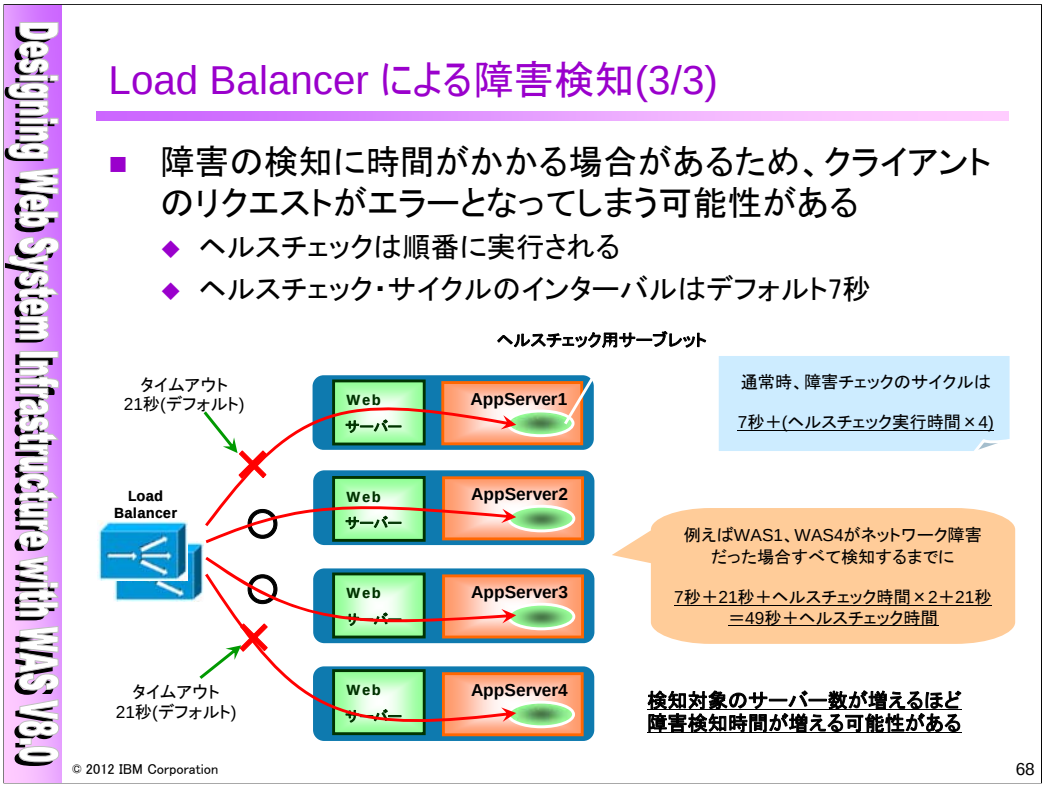
・serverDown・・・サーバーのダウンを検知した際に実行される

・serverUp・・・サーバーの復旧を検知した際に実行される

・managerAlert・・・クラスター内の全てのサーバーがダウンした際に実行される

・managerClear・・・クラスター内の全てのサーバーがダウンした状態から、1台でもサーバーが復旧した際に実行される

これらのスクリプトには任意のロジックを書くことが可能です。スクリプトの中に監視ソフトウェアへの通知を行ったり、管理者へメールするといった記述を行ってください。



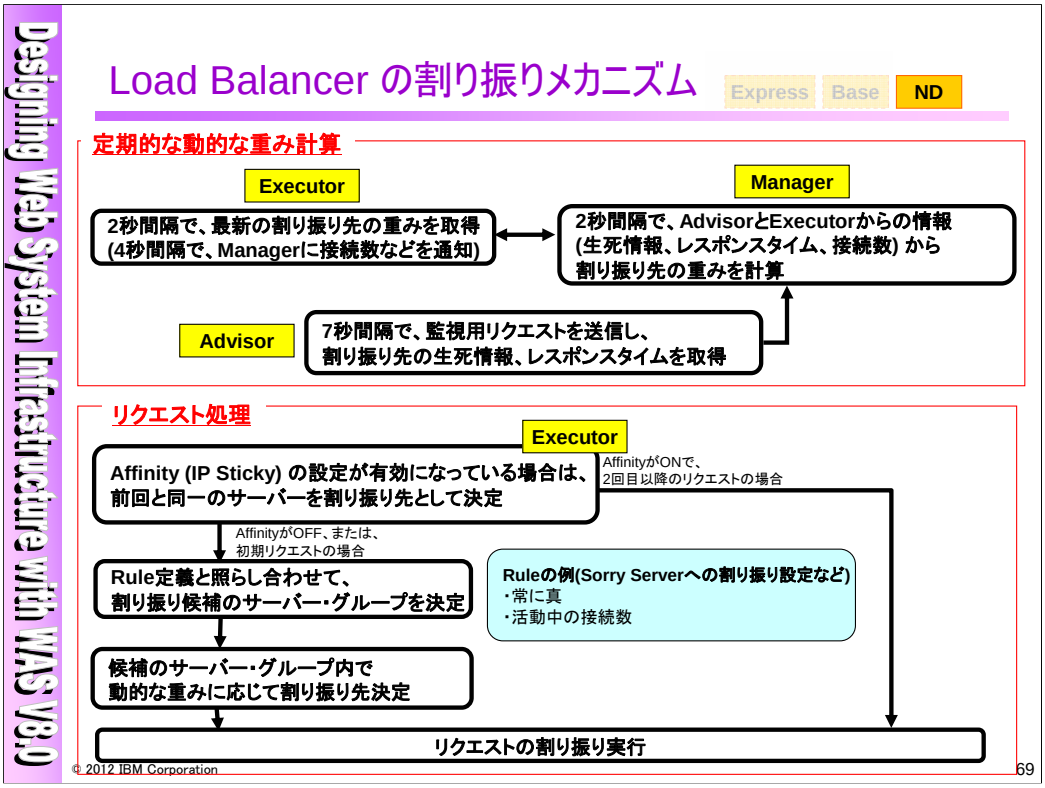
Load Balancerによる障害検知の最後に、注意点を説明します。

監視はクライアントのリクエストとは別に、定期的に行われているため、障害の検知に時間がかかる場合があります。割り振り先サーバーが複数ある場合、ヘルスチェックは順番に実行され、このヘルスチェック・サイクルのインターバルがデフォルトでは7秒となります。このため、例えばこの7秒の間に障害が発生した場合、 障害の検知が遅れクライアントのリクエストがダウンしているサーバーに割り振られ、結果エラーとなってしまう可能性があります。

ヘルスチェックの際、ネットワーク障害により応答がないケースと、サーバーのハングなどで応答がない場合にも障害を判断するために、Advisorの監視パラメータとして2つのタイムアウトを設定することができますが、これらのパラメータのデフォルト値は21秒です。監視は順番に実行されるため、1台のWASからの応答が遅延している場合、その間は他のサーバーの監視が行われません。このため障害の検知がさらに遅れる可能性があります。

・connecttimeout・・・Advisorが割り振り先サーバーへの接続を確立するまでのタイムアウト

・receivetimeout・・・Advisorがリクエストを送信してから接続を受け取るまでのタイムアウト

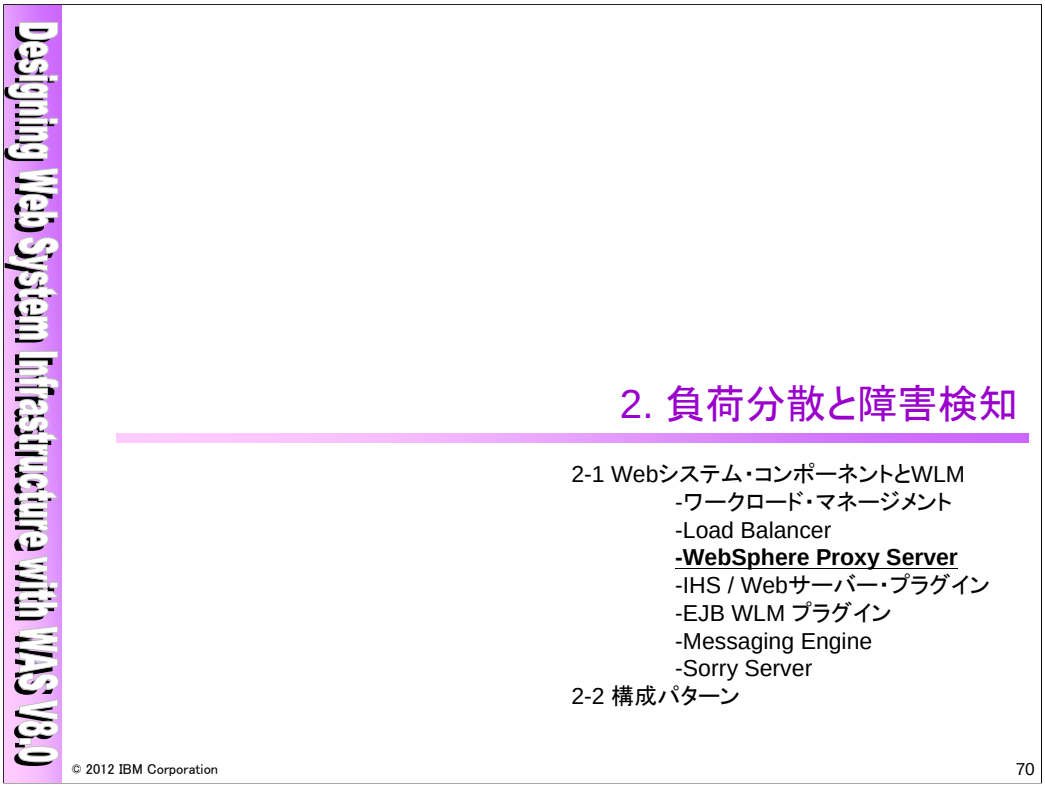


ここでは、Load Balancerの割り振りのメカニズムの詳細を説明します。

定期的な動的な重み計算・・・Advisorは7秒間隔(デフォルト)で監視リクエストを割り振り先サーバーに投げ、生死情報およびレスポンスタイムをManagerに渡します。ManagerはAdvisorの情報およびExecutorから受け取る接続数の情報をもとに割り振り先サーバーの重み付けを行います。ExecutorはManagerから重みを取得し、その重みをもとに割り振りを実施します。

リクエスト処理・・・リクエストがくると、Executorが割り振りを行います。まずアフィニティーが有効な場合、2回目以降のリクエストであれば前回と同様のサーバーへ割り振りを行います。初回リクエストの場合は、予め定義されたルールに従い割り振り先を決定します。

ルールとは、現在の接続数、時間、クライアントのIPアドレスなどにもとづいて、割り振り先を制御することが可能なLoad Balancerの機能です。

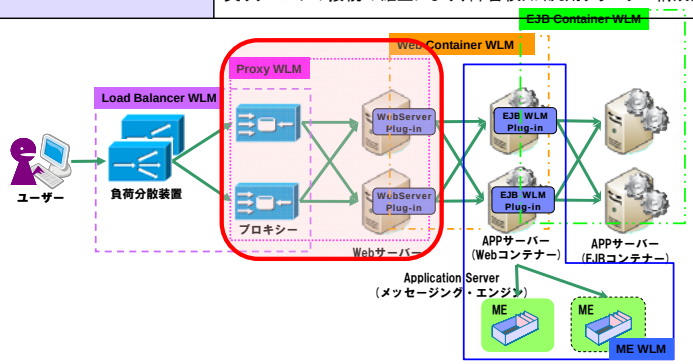

Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
-ワークロード・マネージメント
-Load Balancer
-WebSphere Proxy Server
-IHS / Webサーバー・プラグイン
-EJB WLM プラグイン
-Messaging Engine
-Sorry Server
2-2 構成パターン
© 2012 IBM Corporation
70

Designing Web System Infrastructure with WAS V8.0

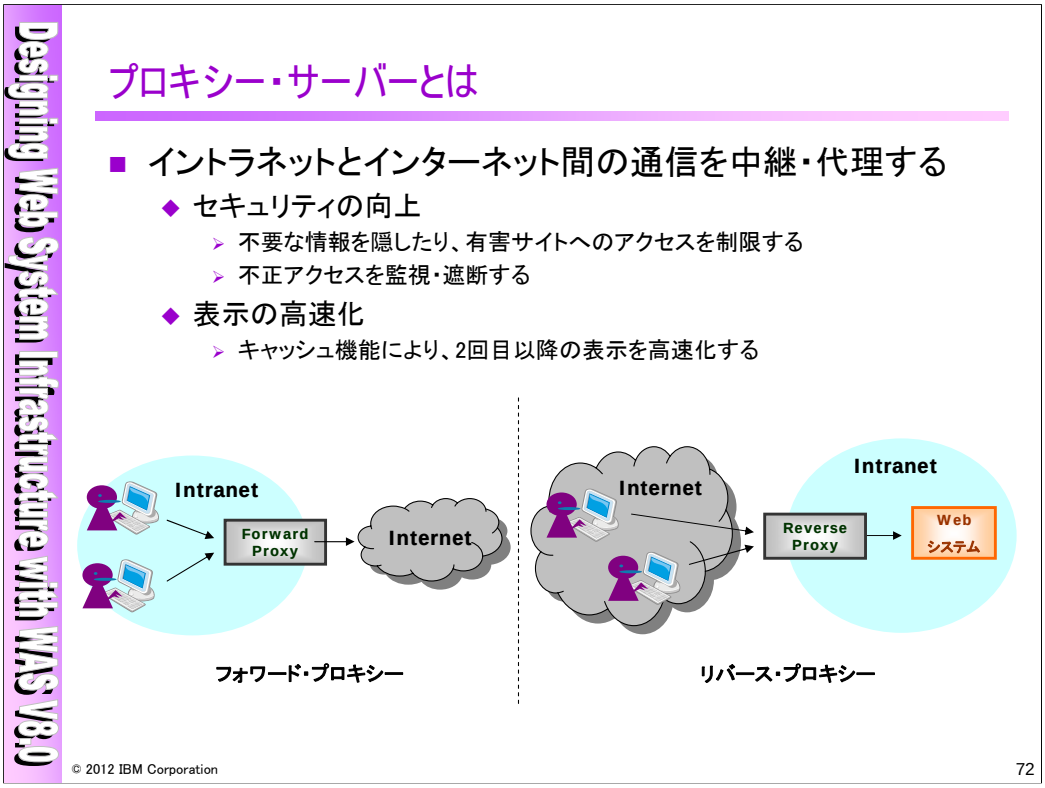
Proxy Server WLM

Express Base **ND**

WLMを行うコンポーネント	WebSphere Proxy Server
割り振りポリシー	重み付きラウンド・ロビン（デフォルト） ランダム（WASV7から追加）
セッション維持（Affinity）	SSL ID URL再書き込み Cookie （Active / Passive）
障害検知	HAマネージャーによる障害検知（ODC：オンデマンド構成） 実リクエストの接続の確立により、障害検知（汎用クラスター構成）

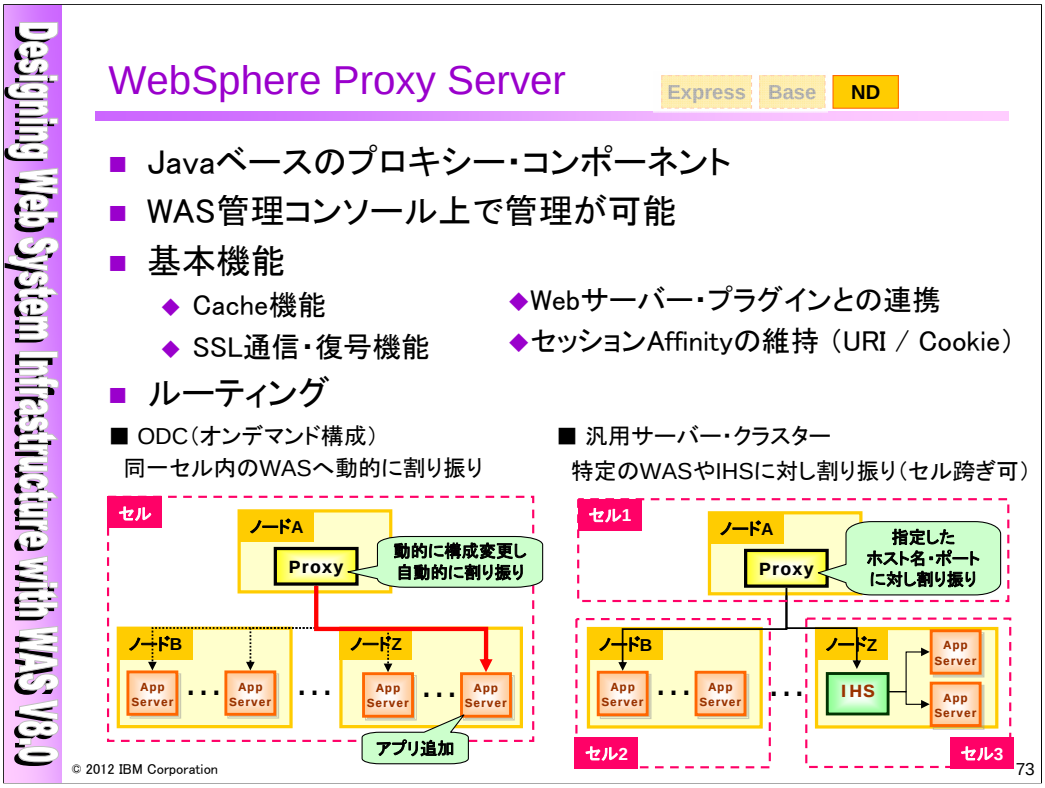


© 2012 IBM Corporation



プロキシサーバーとは、イントラネットとインターネット間のホスト通信を中継・代理する機能/コンポーネントです。プロキシー・サーバーはフォワードプロキシーとリバース・プロキシーに分けられます。

このうち、次ページにてご説明するWebSphere Proxy Serverは、リバース・プロキシーに該当します



まず、WebSphere Proxy ServerはJavaベースのプロキシー・コンポーネントで、WASのセル内に構成し、管理コンソール上でアプリケーション・サーバーと合わせて管理することが可能です。主な基本機能として、

・Cache機能

静的コンテンツ、動的コンテンツ、SSLコンテンツのキャッシュが可能です。

・SSL通信・復号機能

SSLの復号が可能です。クライアント→Proxy間をSSL通信、Proxy Server以降は通常のHTTP通信を行うことも、クライアント→Proxy ServerをSSL通信にし、Proxy ServerでSSLを復号してURLのチェックを行った後、再び暗号化を行い、後方へSSL通信を行うことも可能です。

・Webサーバー・プラグインとの連携

Webサーバー・プラグインから要求をProxy Serverに割り振ることが可能です。

・セッションAffinityの維持

Proxy ServerにてURIやCookieによるAffinity制御が可能です。

が挙げられます。

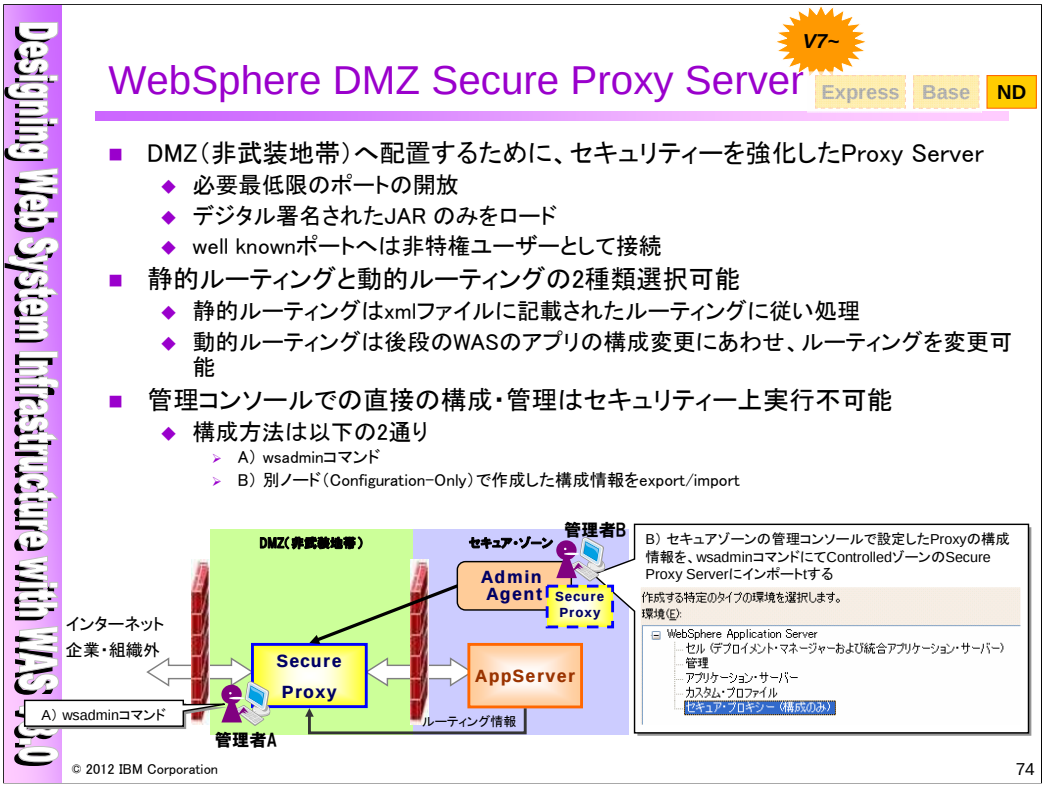
また、ルーティング方式は以下の2種類です：

・ODC（オンデマンド構成）によるルーティング

同一セル内のサーバーに対して追加構成をすることなく、動的に構成変更し割り振ることが可能なルーティング設定です。

・汎用サーバー・クラスターによるルーティング

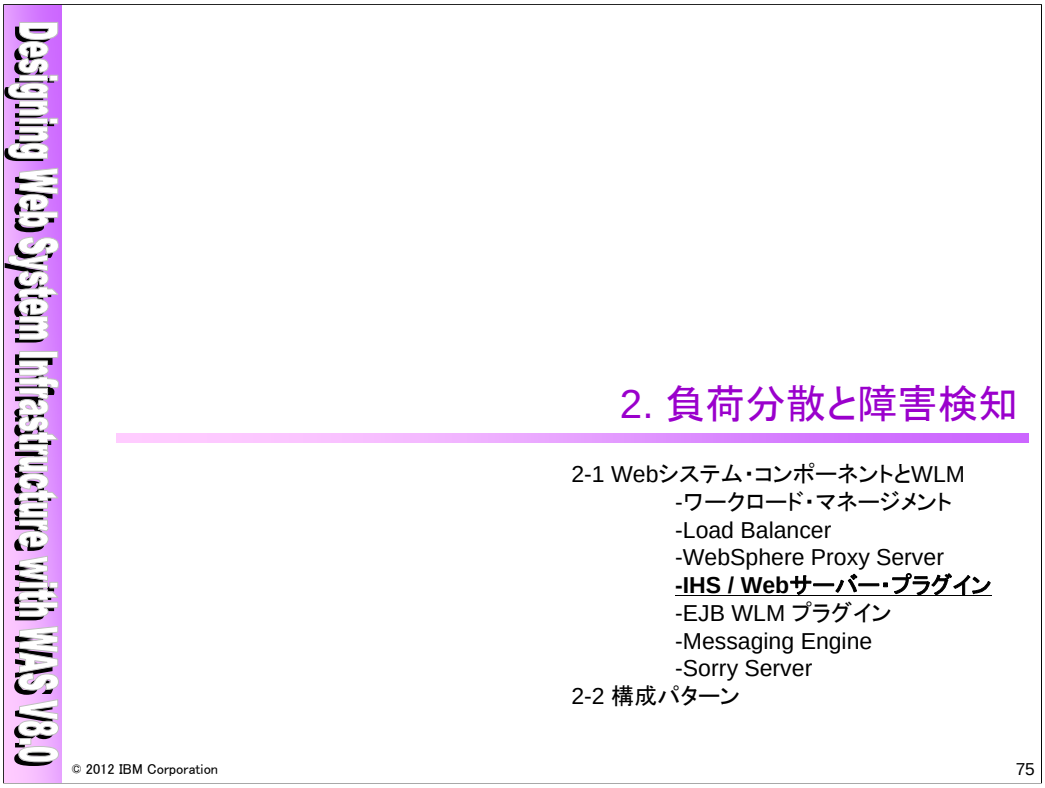
特定のアプリケーション・サーバーやWebサーバー（セル跨ぎも可）に対して割り振る場合に使用するルーティング設定です。

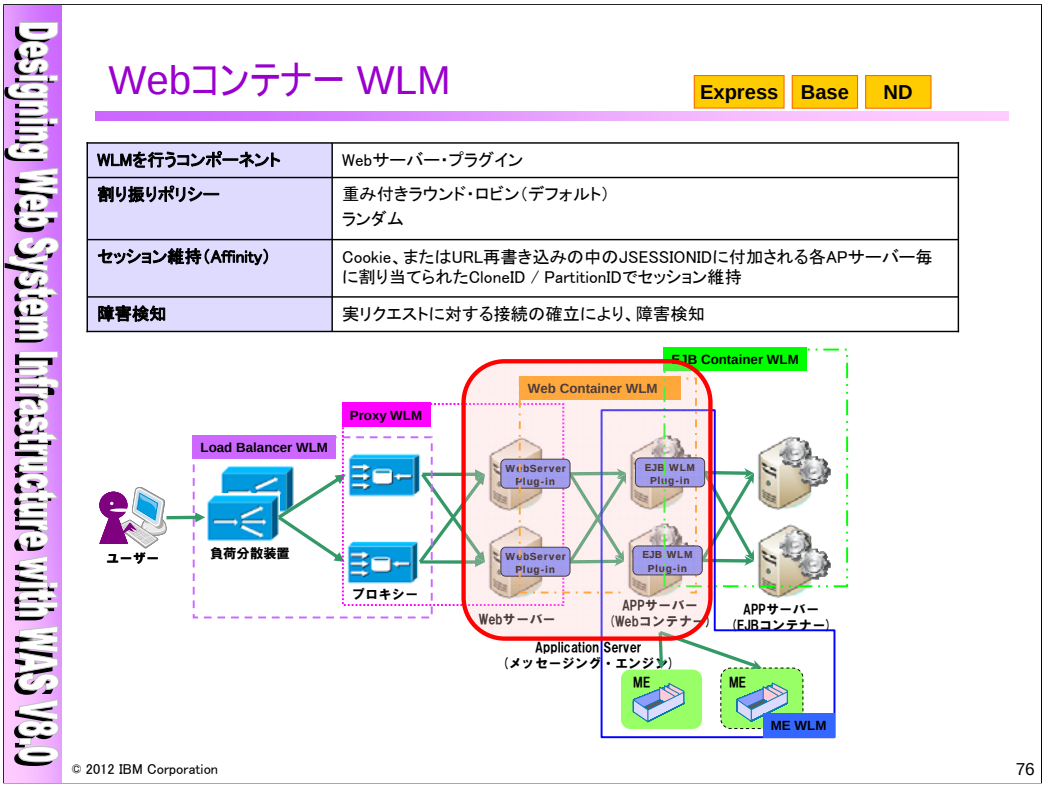


続きまして、WebSphere DMZ Secure Proxy Server（以下Secure Proxy）ですが、こちらはDMZ（非武装地帯）へ配置するためにセキュリティーを強化したProxy Serverです。具体的には、必要最低限のポートの開放、デジタル署名されたJARのみをロード、well knownポートへは非特権ユーザーとして接続する等によりセキュリティーが強化されています。

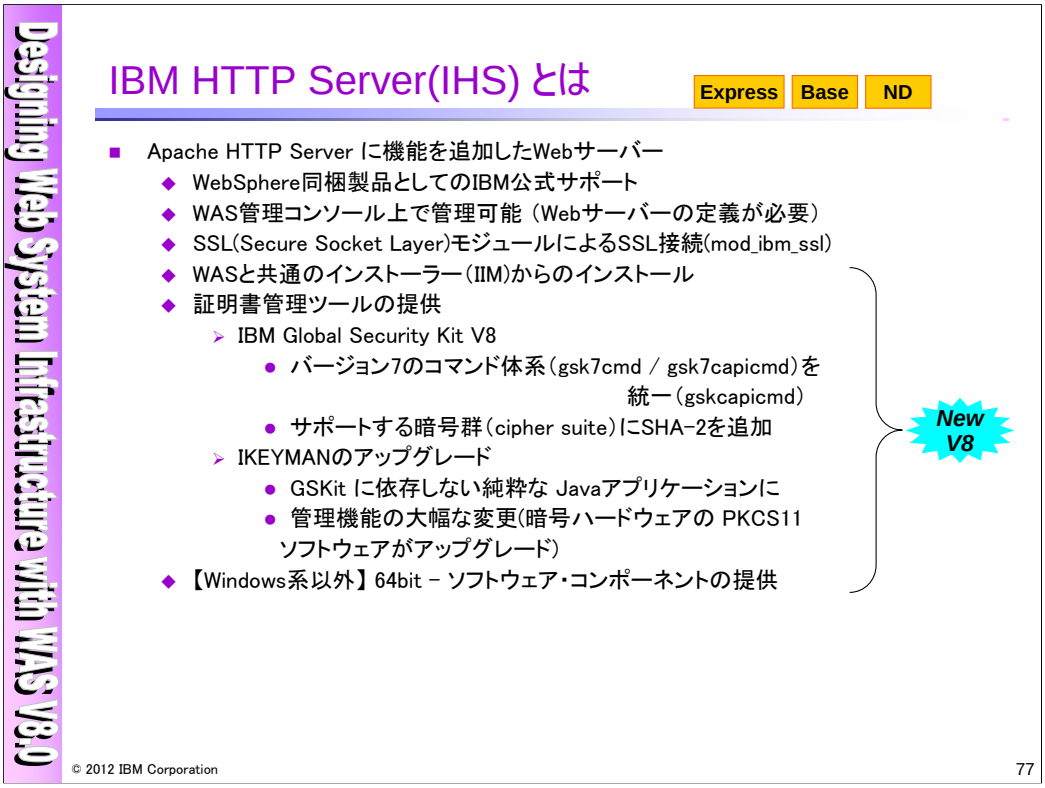
Secure ProxyはWebSphere Proxy Serverと異なり、直接管理コンソールでの管理・運用を行うことができない点が大きな特徴です。設定変更するためにはwsadminコマンド（ローカル、Flexible Management経由）もしくは、別ノードでConfuguration-Onlyプロファイルを作成し、そのノード上の管理コンソールから設定したProxyの構成情報を、wsadminコマンドにてimport/exportする必要があります。

ルーティングに関しては、xmlファイルに基づき割り振りを行う静的ルーティングと、WebSphere Proxy ServerのODC（オンデマンド構成）と同様にWASのアプリケーションの構成変更に合わせて動的にルーティングを制御する動的ルーティングが使用可能です。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
-ワークロード・マネージメント
-Load Balancer
-WebSphere Proxy Server
-IHS / Webサーバー・プラグイン
-EJB WLM プラグイン
-Messaging Engine
-Sorry Server
2-2 構成パターン
© 2012 IBM Corporation
75



Webサーバーからアプリケーション・サーバーのWebコンテナーへのWLMは、Web サーバー・プラグインが行います。割り振りポリシーは、重み付きラウンド・ロビンとランダムが設定可能です。セッションAffinityは、CookieとURL再書き込みが設定可能です。障害検知は、実リクエストの接続の確立により行われます。



IBM HTTP Serverは、2.2系のApache HTTP ServerにIBM独自のモジュール（WASの管理コンソールからの管理を可能にするモジュールやSSL関連のセキュリティー・モジュールなど）を実装したWebサーバーです。

管理コンソールから開始、停止などの管理をすることができます。詳細は次ページで説明します。

V8.0になり、証明書管理ツールであるGSKitのバージョンアップも行われ、特に今まで2系統あったコマンド群が1つに集約され、OSに依存せず統一した手順で証明書の運用管理を行うことが可能になりました。また、サポートする暗号群もより強度なものが加わり、より安全なSSL通信をクライアントに提供することが可能になりました。また、GUIベースの証明書管理ツールであるIKEYMANも機能強化され、GSKitに依存しない純粋なJavaアプリケーションとなりました。ユーティリティー内でエラー報告の大幅な機能強化、暗号ハードウェアの PKCS11 ソフトウェアがアップグレードされ、管理機能の大幅な変更が行われました。

さらに、V7.0まで一部OSを除いて32bit-ソフトウェア・コンポーネントの提供でしたが、V8.0よりWindowsを除き64bit-ソフトウェア・コンポーネントが提供されるようになりました。

（参考）IHS 32bit版とIHS 64bit版の注意点（WAS-08-042）

http://www.ibm.com/jp/domino01/mkt/cnpages1.nsf/page/default-0004B65F

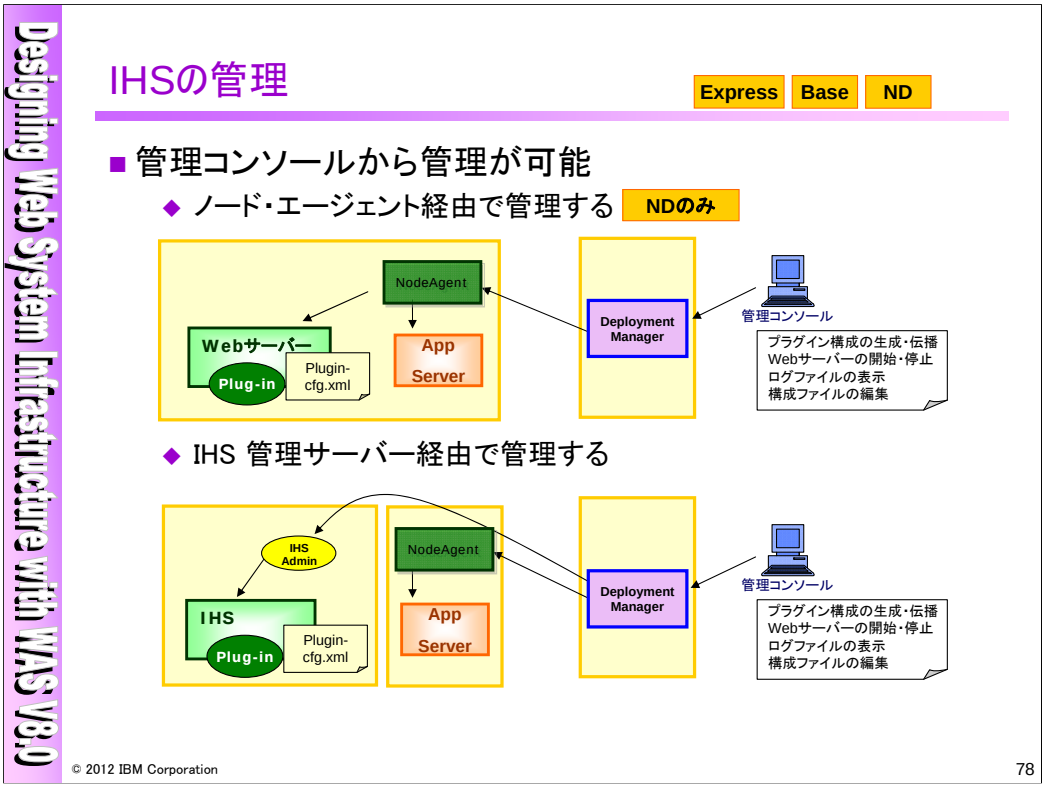
その他詳細についてはInfoCenterをご覧ください。

Information Center「このリリースの新機能」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.ihs.doc/info/ihs/ihs/cihs_newfunction.html

Information Center「Apache HTTP Server との主な相違」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.ihs.doc/info/ihs/ihs/cihs_changes.html



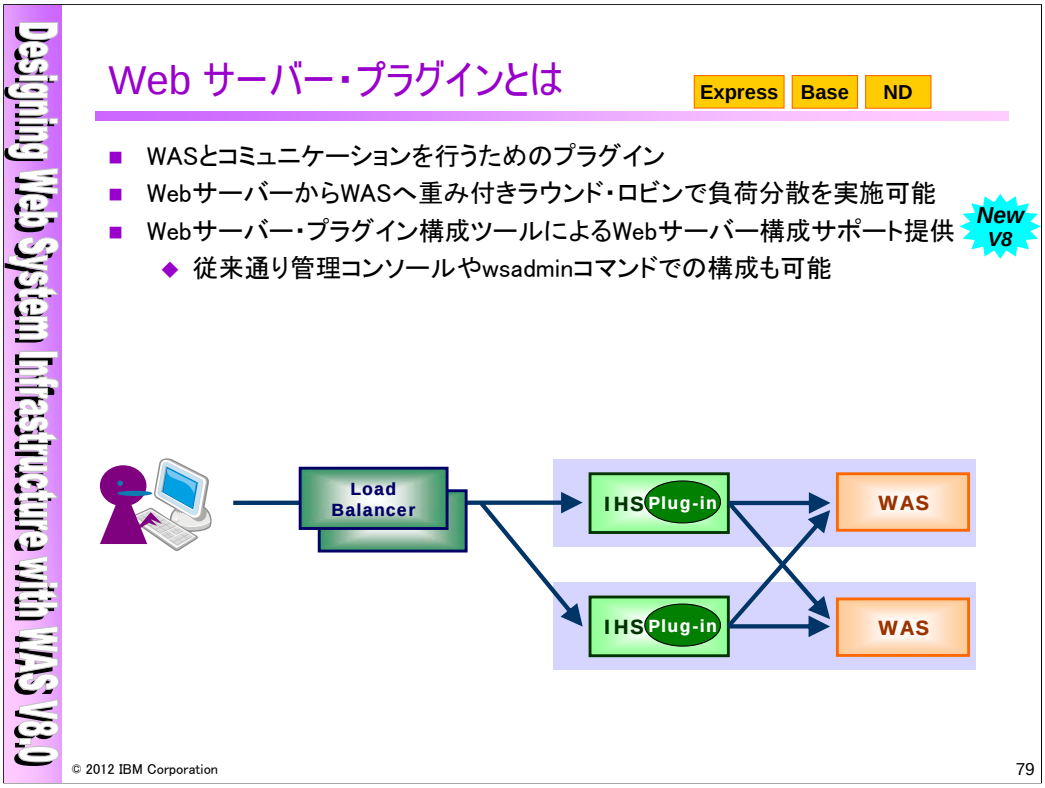
IHSはWASの管理コンソールから管理することができます。管理を行うために、経由するプロセスによって2通りのパターンがあります。

まず、ノード・エージェントを経由する方法です。デプロイメント・マネージャーはノード・エージェントと通信を行い、ノード・エージェントがIHSを管理します。IHSとWASを同一筐体に配置する場合など、IHSが導入されている筐体に既にノード・エージェントが存在する場合はこちらの構成をとることをお勧めします。

次に、IHS管理サーバーを使用する方法です。こちらはIHSとWASを別の筐体に配置する場合に使用します。この場合管理コンソールでは、IHSが配置されるノードを「非管理対象ノード」として構成します。デプロイメント・マネージャーはIHS管理サーバーと通信を行い、IHS管理サーバーがIHSを管理します。IHS管理サーバーの実態はWebサーバー・プロセスであり、デプロイメント・マネージャーとのやり取りはHTTPで行われます。

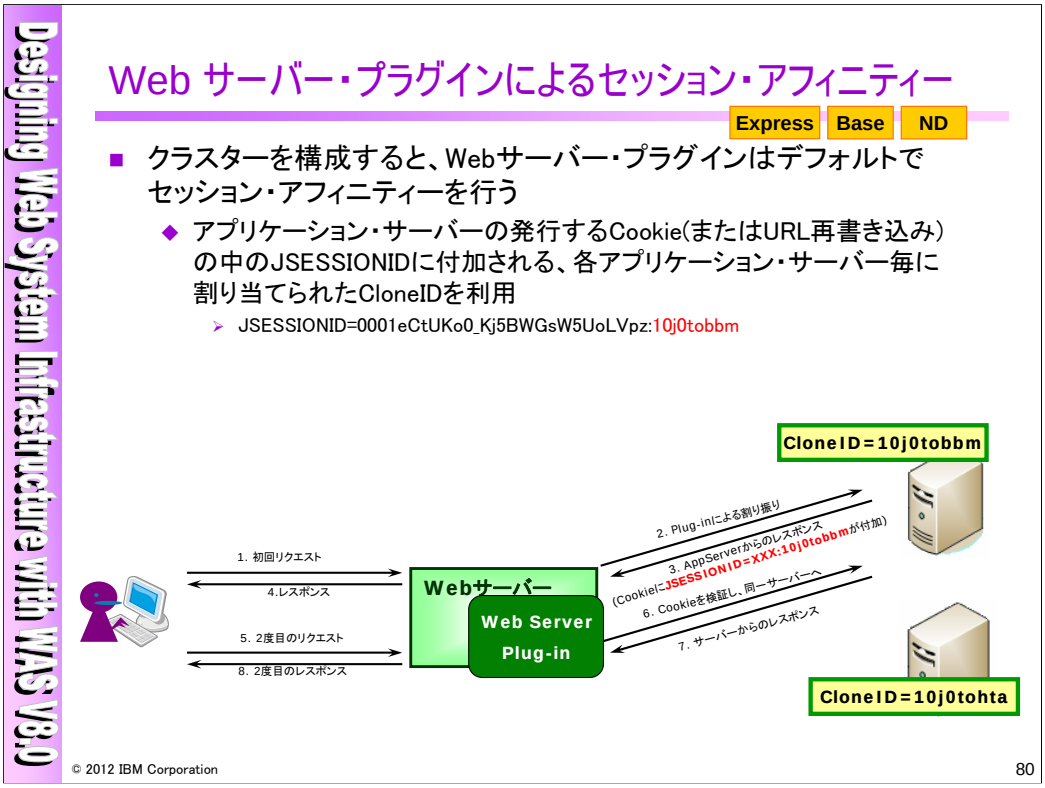
WASV8でのWebサーバーは、これまで同様に、アプリケーション・サーバーと同一筐体でも別筐体でも配置可能です。

いずれの構成の場合も、WAS管理コンソールから、プラグイン構成の生成・伝播 / Webサーバーの開始・停止 / ログファイルの表示 / 構成ファイルの編集が実施可能です。

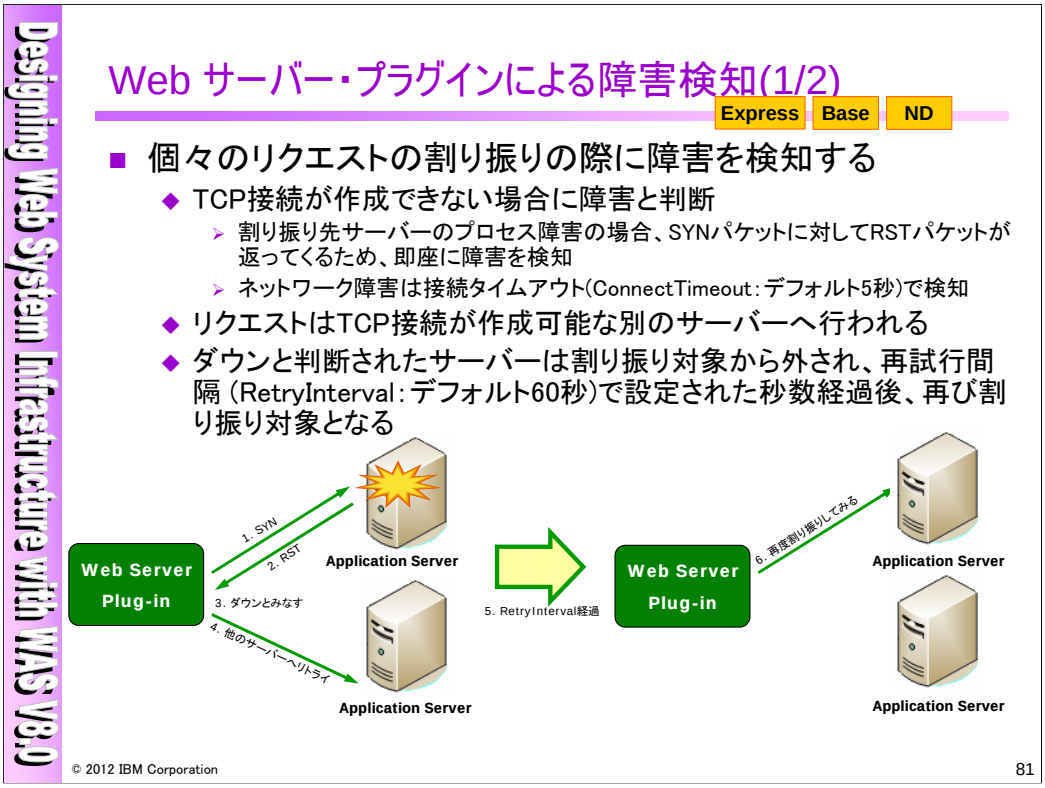


Web Server Plug-insはIHS等WebサーバーとWebSphere Application Serverとの間のコミュニケーションを行うために必要なプラグインで、Webサーバー1台からWAS複数台へ重み付きのラウンド・ロビンでリクエストの負荷分散を行うことが可能です。

V8.0よりWeb Server Plug-insの構成定義をWebサーバー・プラグイン構成ツールによりGUIで行うことが可能になりました（従来通り管理コンソールからも構成はできます）。



クラスターを構成すると、Webサーバー・プラグインはデフォルトの設定でセッション・アフィニティーを行います。これはWASが発行するCookie (or URL再書き込み) の中のJSESSIONIDに付与したサーバー固有の値であるCloneIDを使用して行われます。



Web サーバー・プラグインによる障害検知は、実際に個々のリクエストをWebコンテナーへ割り振る際に行われます。リクエストを割り振るために、まずTCP接続を作成しようとします。割り振り先サーバーにプロセス障害が発生している場合、SYNパケットに対してRSTパケットが返ってくるため、即座に障害が検知されます。ネットワーク障害の場合には応答が返ってきませんが、この場合は接続タイムアウトで障害を検知します。接続タイムアウトのデフォルト値はV6.1までは無制限でしたが、V7.0より5秒に変更されました。

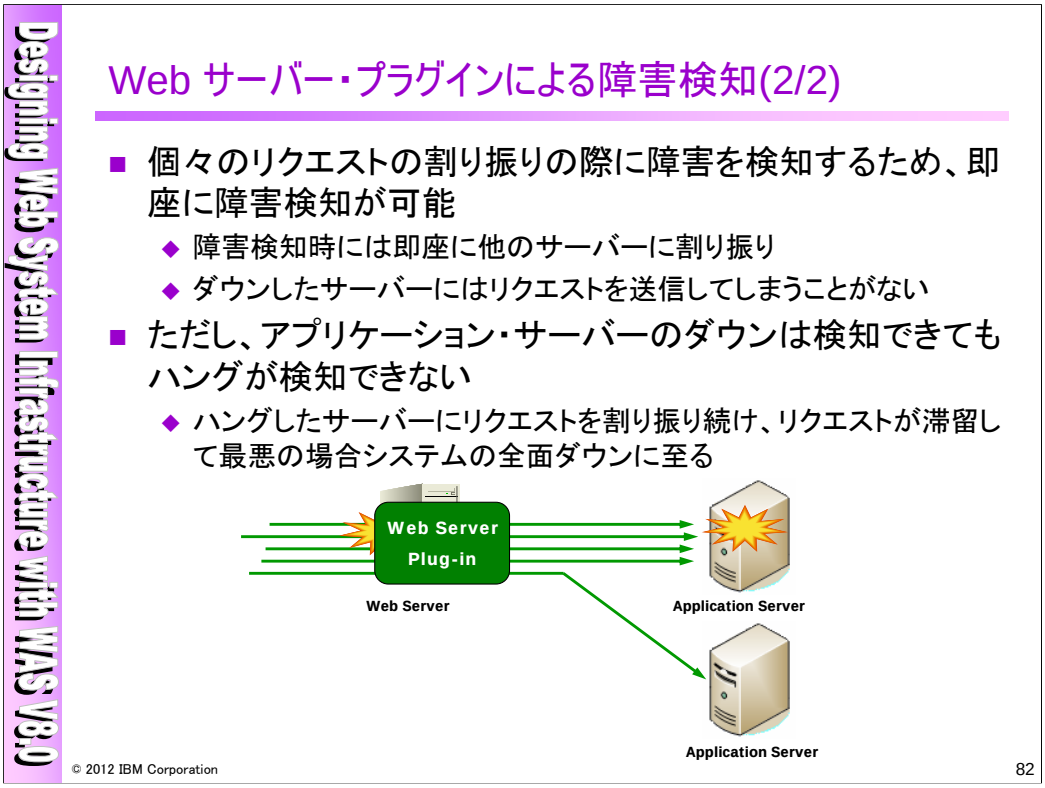
障害を検知すると、リクエストはほかのサーバーに転送されます。

一度ダウンと判断されたサーバーは割り振り対象からはずされますが、再試行間隔で設定された秒数が経過すると、再び割り振りの対象となります。

Webサーバー・プラグインのタイムアウト値設定方法

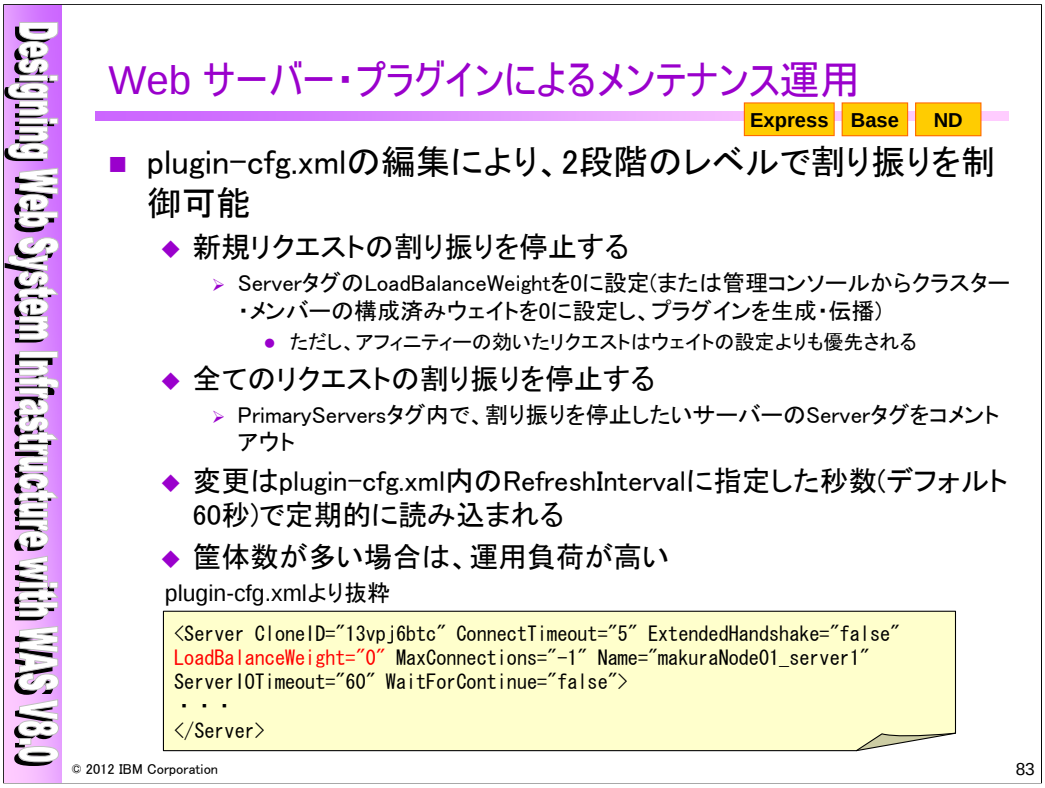
・plugin-cfg.xml を編集

・管理コンソールの「サーバー > アプリケーション・サーバー > server > Web サーバーのプラグイン・プロパティー」より、それぞれ「接続タイムアウト」、「読み取り/書き込みタイムアウト」



Web サーバー・プラグインによる障害検知の特徴は、個々のリクエストの転送の際に毎回TCP接続の確立による障害検知をおこなうため、即座に障害の検知が可能であることです。このため、ダウンしたサーバーにリクエストを転送してしまうことがありません(Load Balancerの場合はユーザーリクエストと異なる監視用のリクエストを定期的に送信するという方法での障害検知であるため、障害の検知にタイムラグが生じます)。

しかし、このTCPレベルの監視では、アプリケーション・サーバーのダウンは検知できても、ハングや関連するサブシステムの障害を検知することができません。ハングが検知できないために、ハングしたサーバーにリクエストを割り振り続けてしまい、最悪の場合システムが全面ダウンするという障害が起こりえるため、ハングに対しては考慮しておく必要があります。

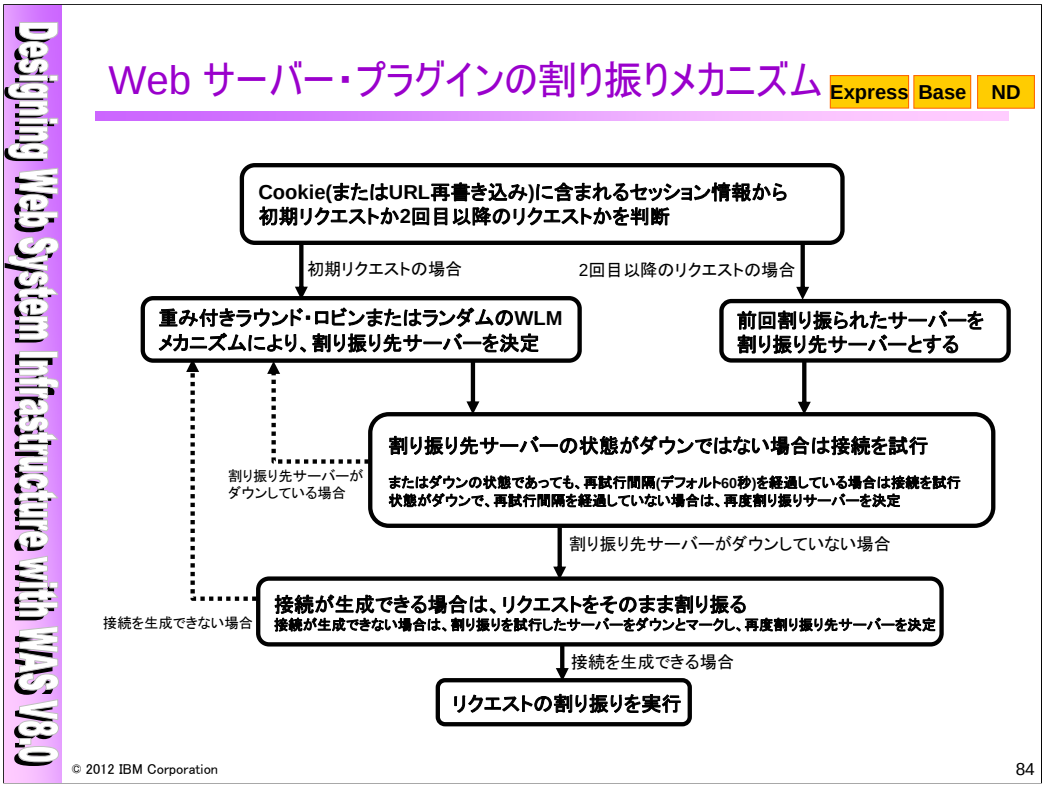


IHS:WASが1:N構成の場合は、Load Balancerで割り振りの制御ができませんので、IHSのPluginで割り振りの制御を行う必要があります。

具体的にはplugin-cfg.xmlファイルを編集し、動的に読み込ませることによって行います。plugin-cfg.xmlはplugin-cfg.xmlの中で指定したRefreshIntervalの間隔でplugin-cfg.xmlを読み込み、動的にルーティング情報の変更を行っています。

plugin-cfg.xmlファイルの割り振り制御も2段階のレベルで割り振りを制御することが可能です。新規リクエストを割り振らなくするには、割り振りを停止したいサーバーのServerタグ内属性であるLoadBalanceWeightを「0」に設定します。これにより新規リクエストは割り振られなくなりますが、アフィニティーの効いたリクエストは引き続き割り振られます。一定時間経過し完全に割り振りを止めたい場合は、PrimaryServersタグ内にある割り振りを止めたいサーバーのServerタグをコメントアウトします。

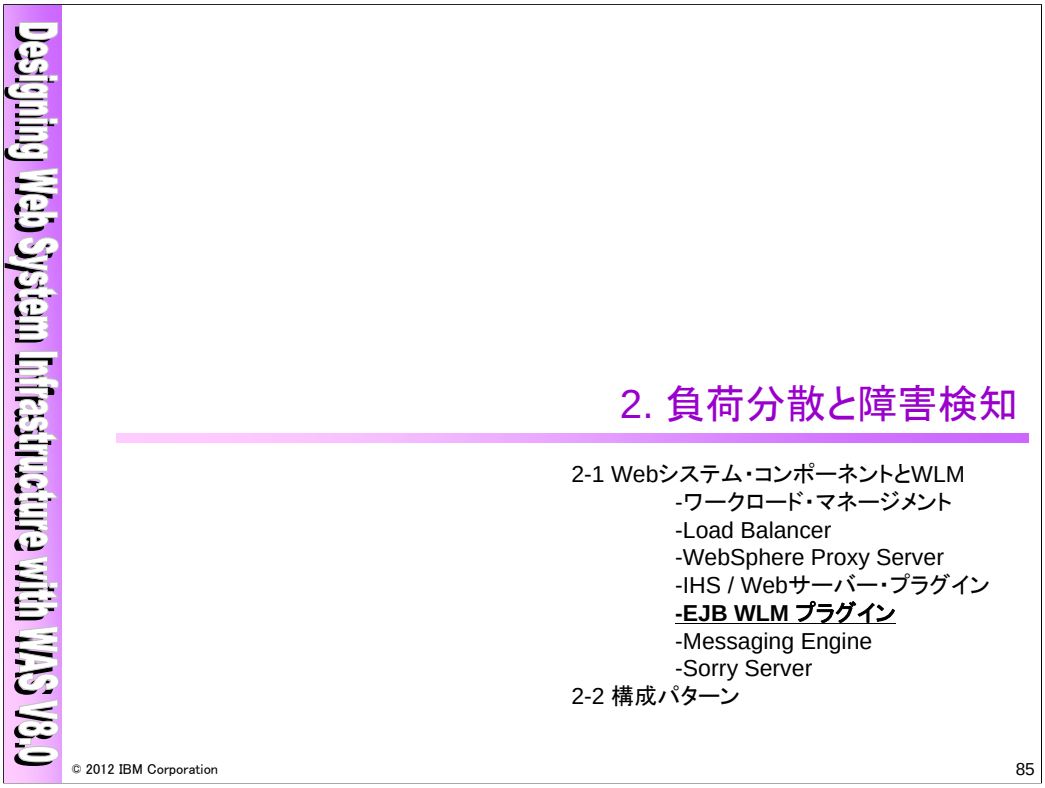
plugin-cfg.xmlファイルの編集による割り振り制御は、全てのIHSノードでそれぞれ作業を行う必要がありますので筐体数が多い場合は運用負荷が高くなります。あらかじめ複数のplugin-cfg.xmlを別名で用意しておき、シェルである程度自動化しておけば編集ミスなどの問題が発生しにくくなります。

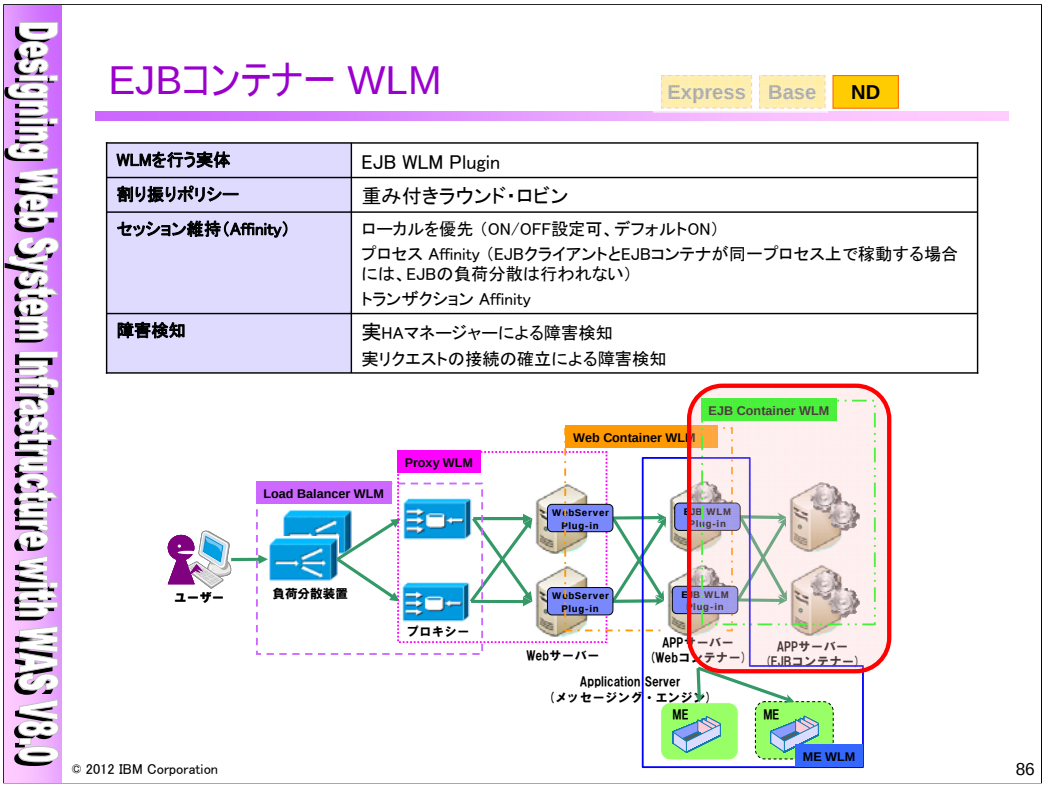


WebコンテナーWLMの割り振り先決定メカニズムは次のとおりです。割り振り先の決定処理フローは、初期リクエストの場合と2回目以降のリクエストで異なります。

初期リクエストが来た場合、重み付きラウンド・ロビンもしくはランダムのWLMメカニズムにより割り振り先が決定されます。割り振り先が正常に稼動している場合はそのままリクエストを割り振ります。割り振り先がダウンしていた場合、再度割り振り先を決定します。

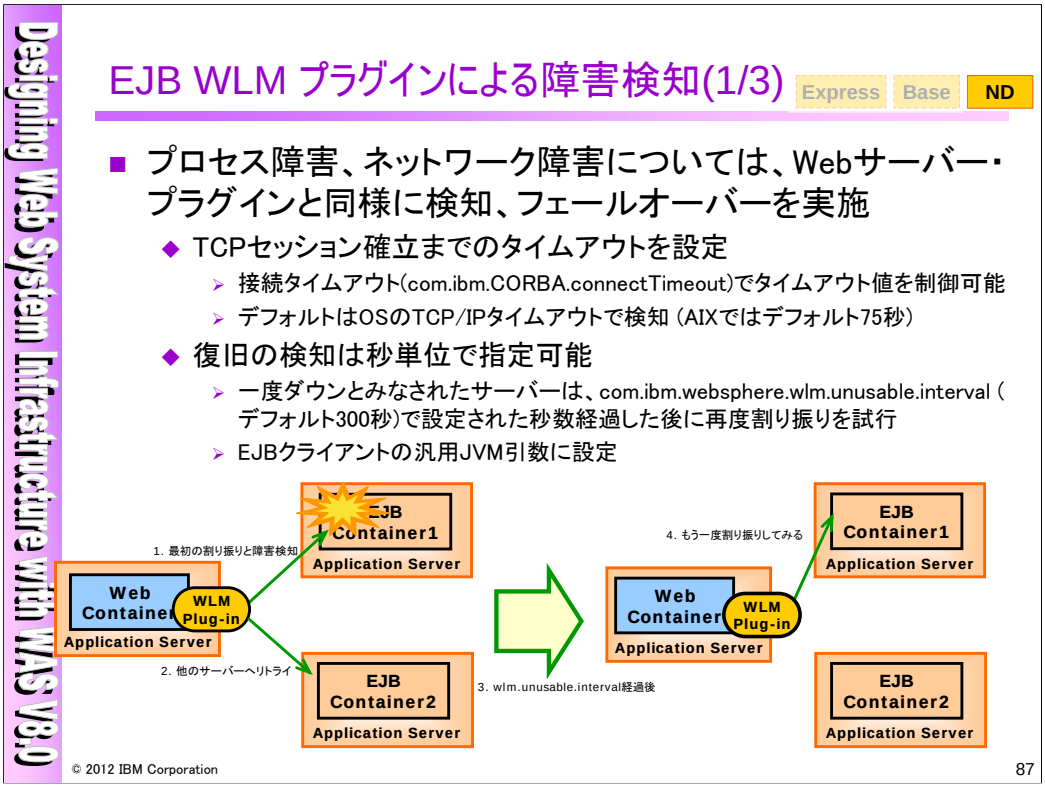
2回目以降のリクエストが届いた場合、CookieもしくはURLに含まれるセッション情報から前回割り振ったサーバーへと割り振りが行われます。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
-ワークロード・マネージメント
-Load Balancer
-WebSphere Proxy Server
-IHS / Webサーバー・プラグイン
-EJB WLM プラグイン
-Messaging Engine
-Sorry Server
2-2 構成パターン
© 2012 IBM Corporation
85



EJBコンテナーへのWLMは、Webコンテナー上に存在するEJB WLM プラグインが実施します。割り振りポリシーは、Webコンテナーと同様に重み付きラウンド・ロビンを採用しています。

Affinityは、"ローカルを優先"、"プロセスAffinity"、"トランザクションAffinity" の3つがあり、"重み付けラウンド・ロビン"よりも優先されます。"ローカルを優先"を設定するとEJBクライアントと同一ノード上のEJBコンテナーへのリクエストが優先され、ネットワーク通信が行われませんので、パフォーマンスの向上が期待できます。EJBクライアントとEJBコンテナが同一プロセス上で稼動する場合は、プロセスAffinity機能により、全てのEJBクライアントのリクエストは同一プロセスのEJBコンテナーに割り振られます。トランザクションAffinityは、すべての割り振りポリシーおよびAffinityより優先され、同一のトランザクション・スコープ内のリクエストは、必ず前回と同じEJBコンテナーにリクエストが割り振られます。障害検知は、実リクエストの接続の確立、もしくはHAマネージャーにより行われます。



EJBへのWLMでは、Web サーバー・プラグインと同様にプロセス障害およびネットワーク障害を検知し、正常なEJB コンテナーへと割り振りを行います。このタイムアウト値はEJBクライアントの汎用JVM引数com.ibm.CORBA.connectTimeoutで指定することが可能です。

復旧はcom.ibm.websphere.wlm.unusable.intervalのデフォルト300秒後に接続を試行することで検知できます。

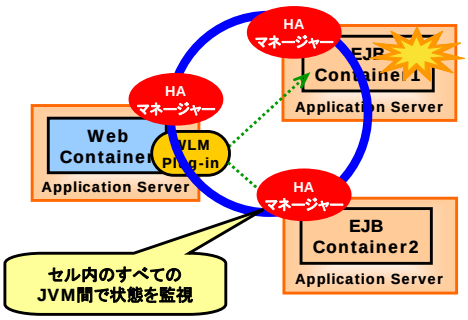
※設定の詳細は、InfoCenter「ワークロード管理構成の調整」の記載を参照下さい。

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/trun_wlm_tuning.html

Designing Web System Infrastructure with WAS V8.0

EJB WLM プラグインによる障害検知(2/3)

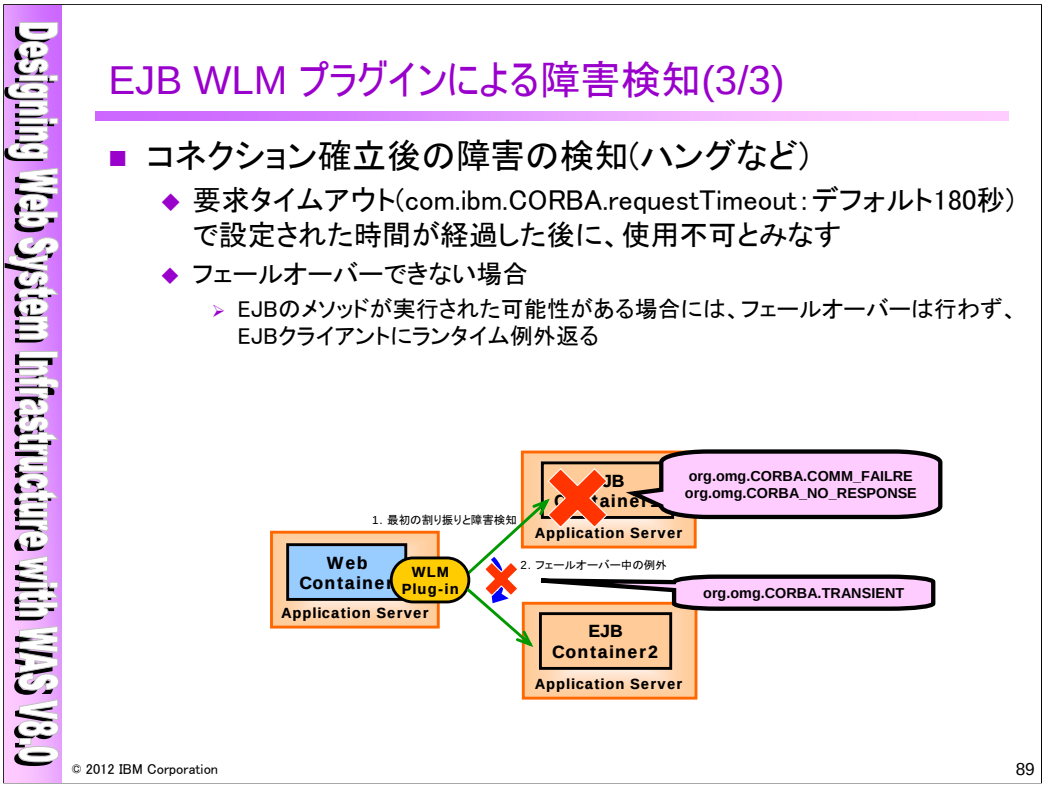
- HAマネージャーによる障害検知
  - すべてのJVMプロセスで稼動するHAマネージャーによる相互のSocket監視とHeartbeatによる監視により、リクエスト試行前にプロセス障害、ネットワーク障害を検知可能



その情報は、EJBクライアントとEJBコンテナが同一セル内の場合は、直接通知される。
EJBクライアントとEJBコンテナが同一セルでない場合は、EJBレスポンスに含めて通知される。

© 2012 IBM Corporation

EJBコンテナーの障害については、HAマネージャーによりリクエスト試行前に検知が可能です。HAマネージャーはアプリケーション・サーバーやノード・エージェントといったプロセス上で稼働しています。各プロセス間のHAマネージャーはHeartbeat通信を行い、プロセスの稼働状況をチェックしています。EJBコンテナー用プロセスがダウンした場合、HAマネージャーによりプロセスダウンが検知され、その結果EJB WLMプラグインのルーティングテーブルが更新されます。EJB WLMプラグインはこのルーティングテーブルを見て割り振りを行うため、実リクエストをEJBコンテナーに対し投げる前にプロセスダウンを検知でき、該当のEJBコンテナーには割り振りがおこなわれなくなります。EJBコンテナーが障害から回復した際は、同様にHAマネージャーが検知を行い、EJB WLMプラグインのルーティングテーブルが更新され、割り振りが行われるようになります。



EJBコンテナーに対しリクエストが投げられた後の障害については、要求タイムアウト(デフォルト180秒)により検知が可能です。その際、EJBコンテナーから返されるExceptionの結果から、EJBのメソッドが実行されていないことが保障されている場合には、EJB WLMプラグインにより別のEJBコンテナーに対しリクエストが再送されます。しかし、EJBのメソッドが実行されている可能性がある場合は、フェールオーバーは行われず、以下のランタイム例外が返ります。

・org.omg.CORBA.COMM_FAILURE

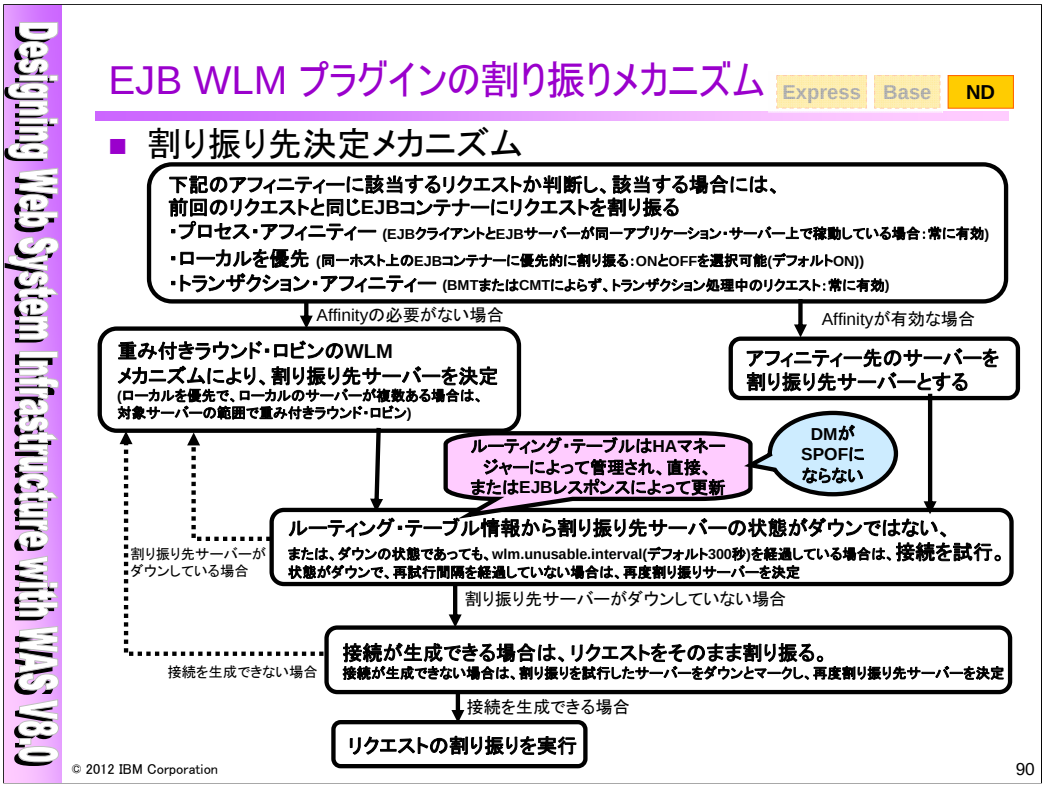
クライアントからの要求送信後、サーバーからの応答がクライアントに返る前に、処理の実行中に通信が失われた場合にスローされる

・org.omg.CORBA.NO_RESPONSE

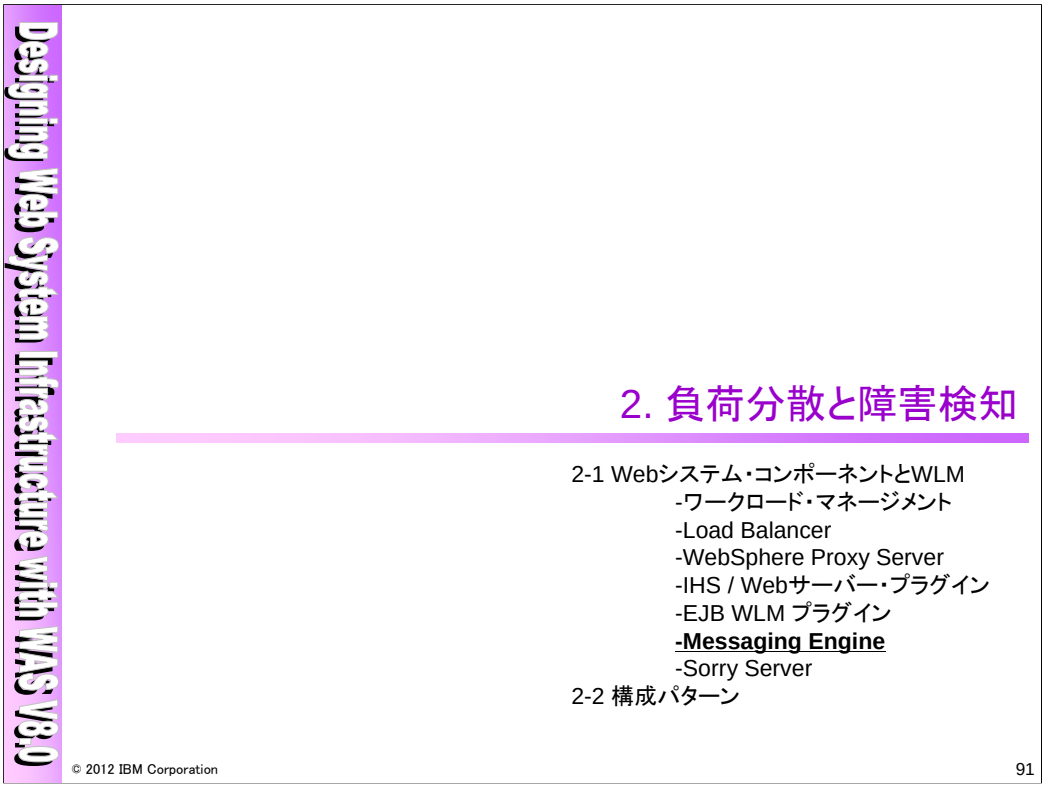
クライアントが遅延同期呼び出しの結果を取得する際に、要求応答がまだ使用できない場合にスローされる

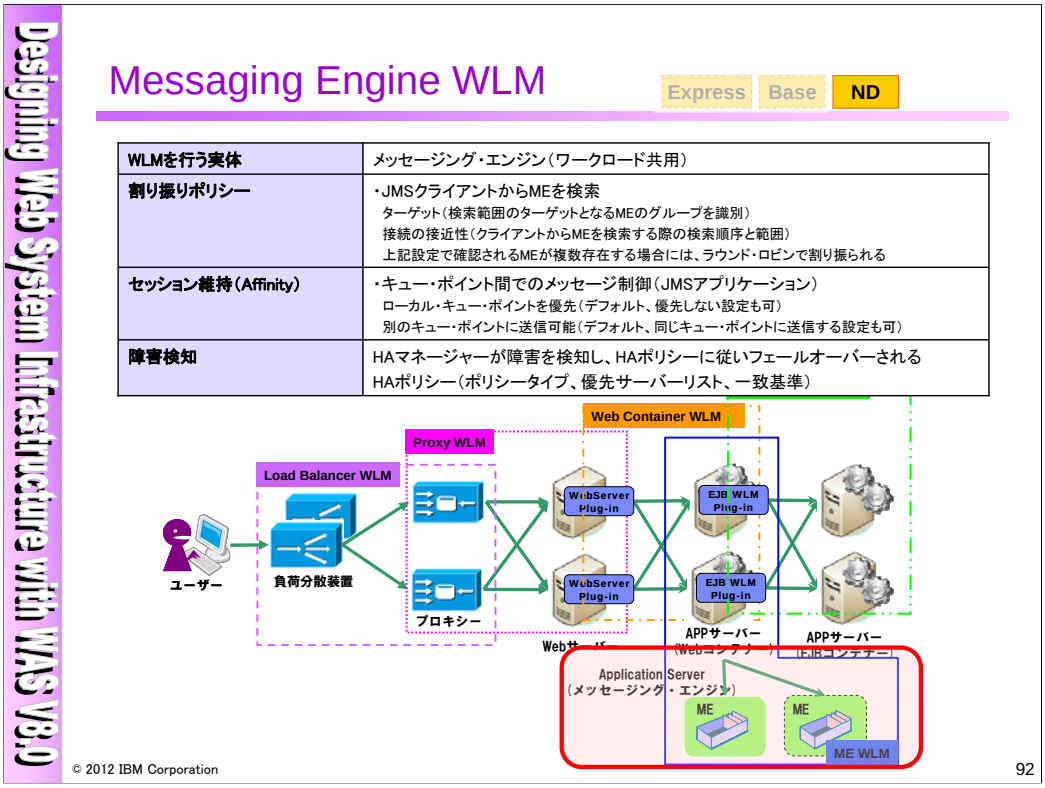
・org.omg.CORBA.TRANSIENT

ORBがオブジェクトに到達しようとして失敗した場合にスローされる



EJBリクエストでは、リクエストが来たとき、アフィニティーの必要があるかどうかチェックを行い、必要であればアフィニティー先のサーバーへと割り振りを実施します。そして割り振り先がダウンしている場合およびアフィニティーが必要ない場合は、重み付きラウンド・ロビンにて割り振り先が決定されます。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
-ワークロード・マネージメント
-Load Balancer
-WebSphere Proxy Server
-IHS / Webサーバー・プラグイン
-EJB WLM プラグイン
-Messaging Engine
-Sorry Server
2-2 構成パターン
© 2012 IBM Corporation
91



メッセージング・エンジンでは、SIBusメンバー内に複数のMEを起動し処理を分散させることができ、これを「ワークロード共用」と呼びます。そして、JMSクライアントから、SIBus内のアクティブなMEを検索する順番は、ターゲットおよび接続の接近性にて設定します。この設定で複数のMEが確認された場合には、ラウンド・ロビンで割り振られます。障害検知は、HAマネージャーの機能により、HAポリシーに従ってメッセージング・エンジンがフェール・オーバーされます。

Information Center – 「ワークロード共有」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cjt0013_.html

Information Center – 「デフォルト以外のブートストラップ・サーバーへの接続の構成」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/tjn0033_.html

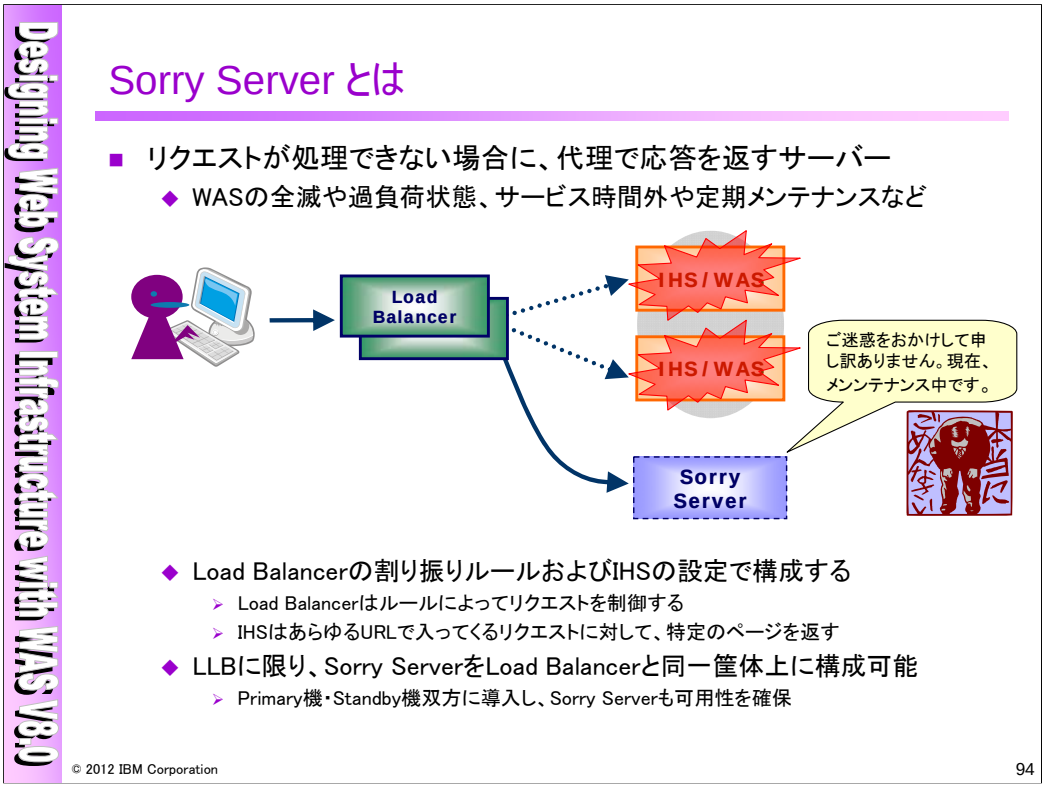
Designing Web System Infrastructure with WAS V8.0

2. 負荷分散と障害検知

2-1 Webシステム・コンポーネントとWLM
- -ワークロード・マネージメント
- -Load Balancer
- -WebSphere Proxy Server
- -IHS / Webサーバー・プラグイン
- -EJB WLM プラグイン
- -Messaging Engine
- **-Sorry Server**

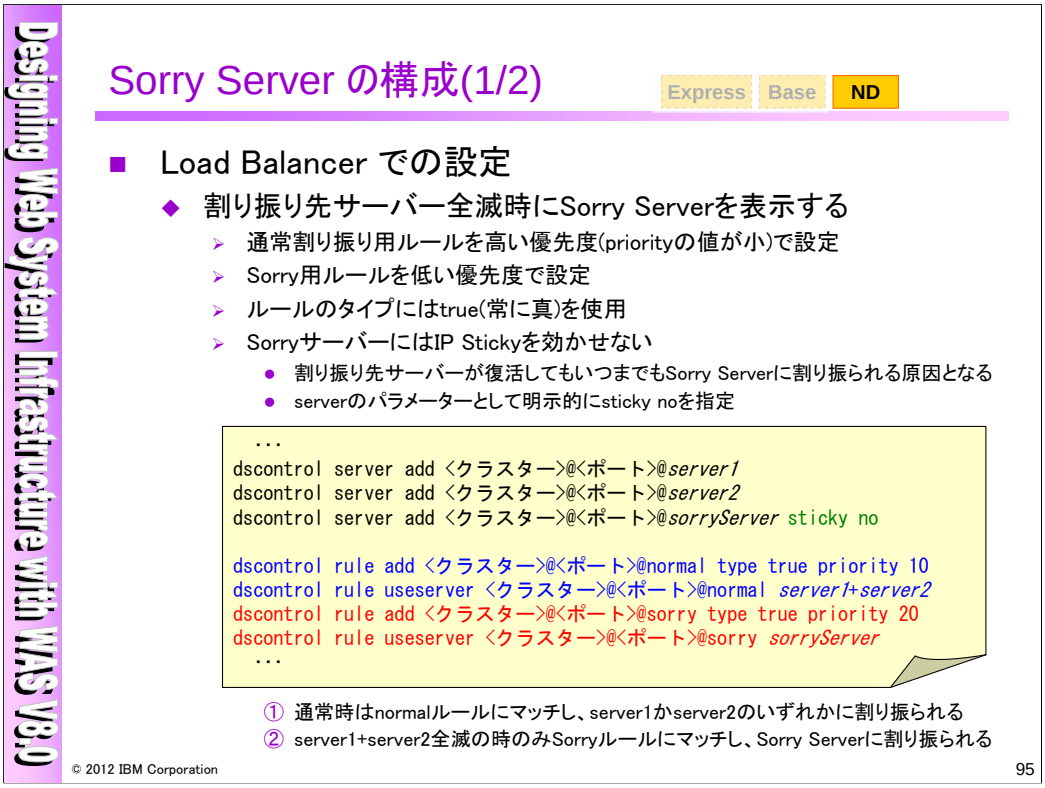
2-2 構成パターン

© 2012 IBM Corporation



Sorry Serverとはアプリケーション・サーバーの障害時や過負荷状況、またはメンテナンスなど、なんらかの理由でサービスが提供できない場合に、サービスが提供できない旨の代替のコンテンツをレスポンスとして返すサーバーになります。

通常Sorry ServerはLoad Balancerでのルール設定とWebサーバーの設定を変更することによって構成します。なおLegacy Load Balancerの場合はSorry ServerをLoad Balancerと同一筐体上に配置することが可能ですので、筐体数を増やすことなくSorry Serverを構築することが可能です。またLoad BalancerのPrimary機・Standby機双方にSorry Serverを構築することでSorry Server自体の可用性をあげることができます。



Sorry ServerのLoad Balancerでの構成方法です。

最初にLoad Balancerにおけるルールの定義方法について説明します。Load Balancerではルールの定義に二つのコマンドを使用します。まず、dscontrol rule add <クラスター>@<ポート>@<ルール名>コマンドを使用してルールを定義します。このルールの定義では、ルールの優先度や、ルールの条件を記述します。次にdscontrol rule useserver <クラスター名>:<ポート>:<ルール名> <サーバー名>コマンドを使用し、ルールにマッチした場合に割り振るサーバーを定義します。

ルールの評価はリクエストごとに行われます。ルールは優先度の高いものから評価され、ルールにマッチすると、そのルールに対してdscontrol rule useserverコマンドで定義されているサーバーに割り振りが行われます。ここで、もし割り振り先のサーバーが全てダウンしていた場合は、次のルールの評価に移ります。

最終的にいずれのルールにもマッチしなかった場合は、リクエストはラウンド・ロビンで割り振られます。もっともシンプルな例をみてみます。

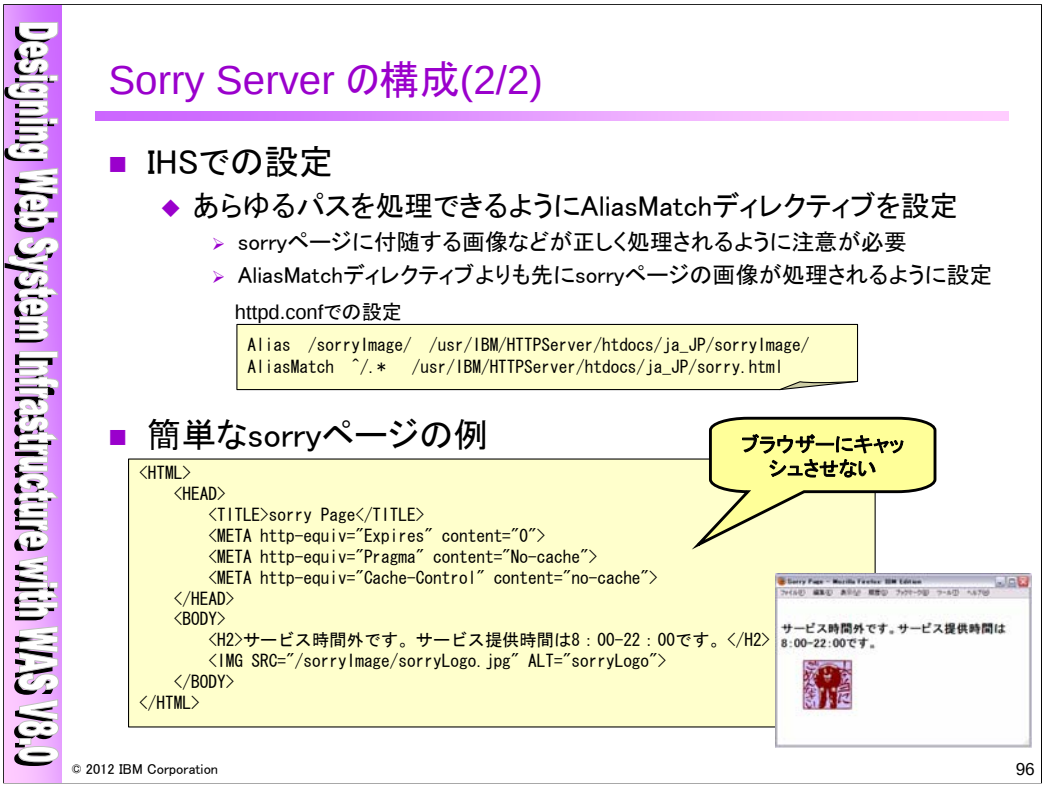
また、ルールは定義されているものの、ルールにマッチした場合の割り振り先サーバーが定義されていない場合は、リクエストはドロップされます。

Sorry Serverを構成するもっともシンプルな構成では、type trueのルールを使用してスライドにあるような定義をします。このルールは以下のような挙動をしますので、Sorry Serverへの割り振りが実現できます。

①通常時はnormalにマッチし、server1かserver2のいずれかに割り振られる

②server1+server2全滅の時のみsorryにマッチし、sorryServerに割り振られる

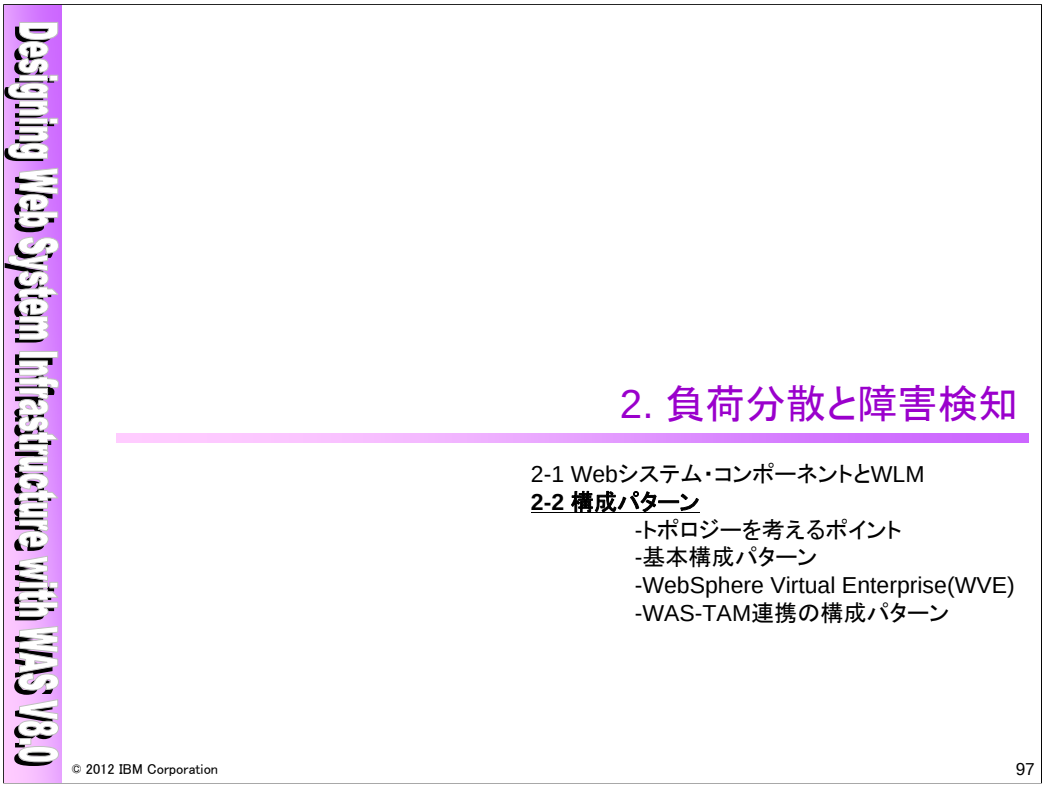
なおIP Stickyを使用している場合はルールよりStickyが優先されますので、Sorry ServerまでStickyが有効になっていると通常時用サーバーが復旧した場合でもいつまでもsorryサーバーに割り振られるといったことになってしまいます。IP Stickyを使用している場合には、sticky noと明示的にSorry Serverに対するStickyを無効にしてください。

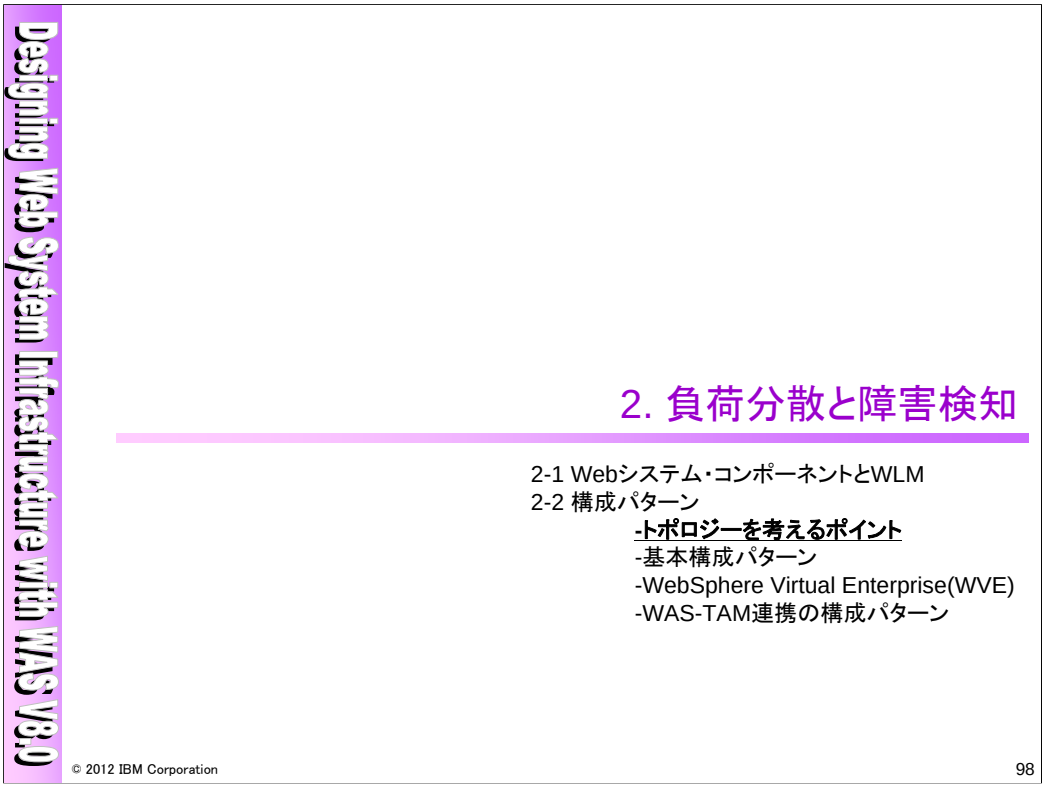


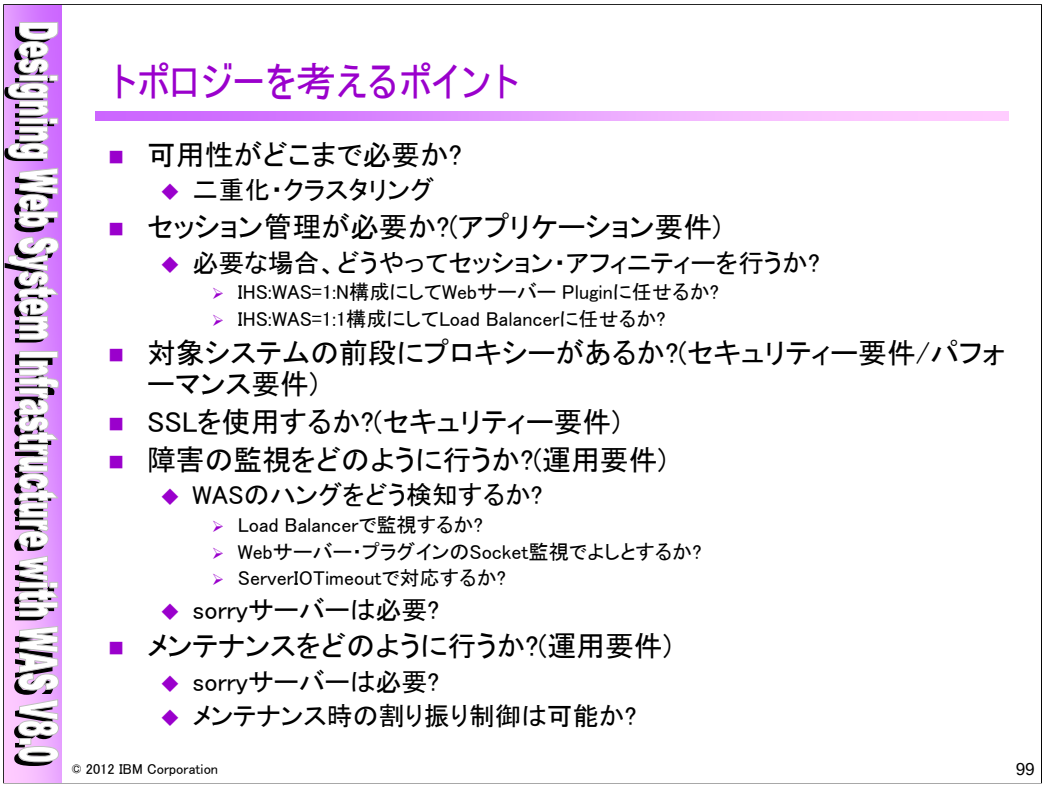
Sorry Serverに対してはどのようなURLでリクエストが飛んでくるか分かりません。どのようなリクエストに対してもコンテンツが返すことができるようにIHSではAliasMatchディレクティブを用いてコンテンツを指定します。上記のAliasMatch例では先頭が(^)/で始まり、任意の一文字(.)が0回以上繰り返される(＊)場合、つまりあらゆるURLにマッチングします。sorryページ内の画像までこのAliasMatchディレクティブで返されてしまってはまずいので、sorryページ内の画像に関わるディレクティブはAliasMatchディレクティブの前に記述します。

下にあげてあるHTMLページは簡単なsorryページの例です。sorryページがブラウザーにキャッシュされてしまいますと、システムが復旧したにも関わらず、クライアントがいつまでもブラウザーのsorryページのキャッシュを見ているといったことになりかねません。これを防ぐために、HTMLコンテンツのHEAD部分にブラウザー側でのキャッシュを防ぐためのヘッダー情報を付加します。

但し一部のブラウザーには、これらのヘッダーを無視してキャッシュを行ってしまうブラウザーもありますので、sorryページのコンテンツ内に「いつまでもこのページが表示される場合は、画面をリロードして下さい」といった記述が必要なケースがあります。

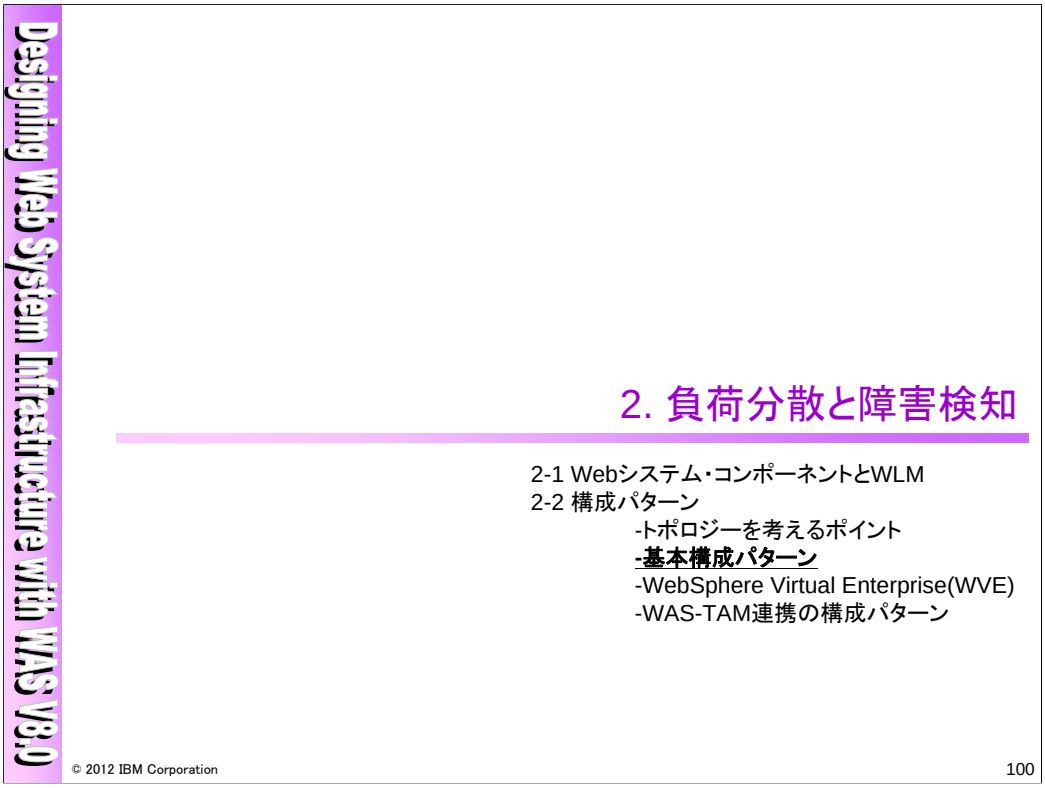

Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
2-2 構成パターン
-トポロジーを考えるポイント
-基本構成パターン
-WebSphere Virtual Enterprise(WVE)
-WAS-TAM連携の構成パターン
© 2012 IBM Corporation
97

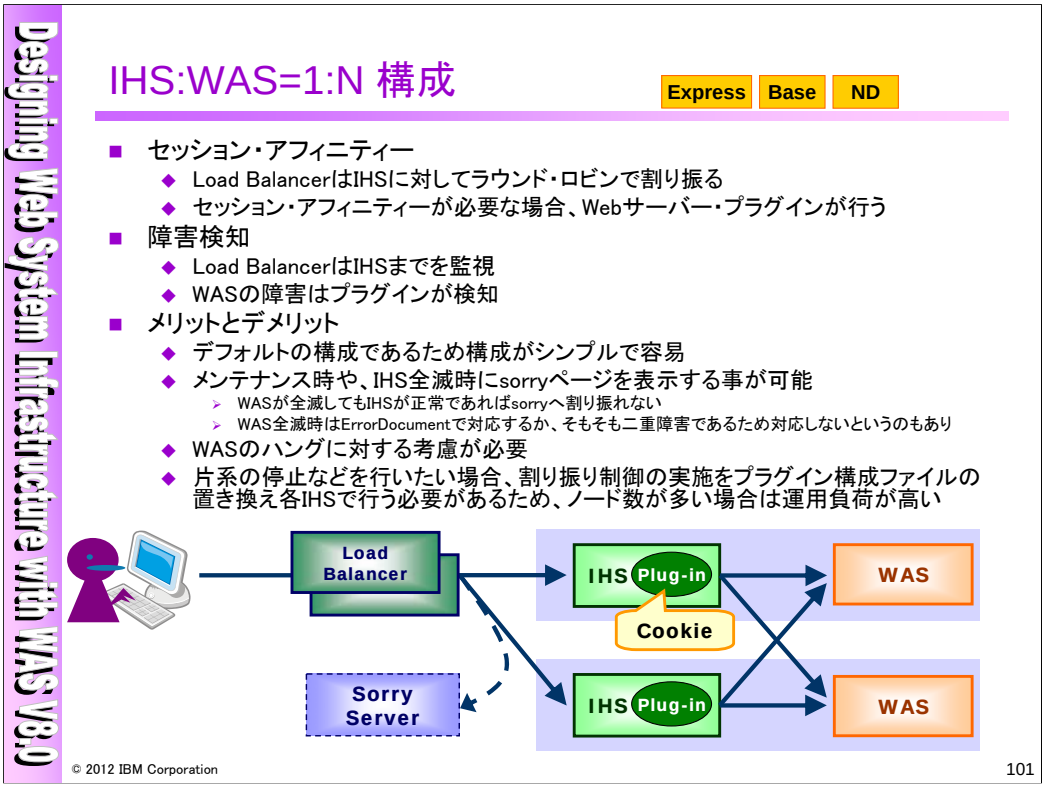

Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 WebシステムとコンポーネントとWLM
2-2 構成パターン
-トポロジーを考えるポイント
-基本構成パターン
-WebSphere Virtual Enterprise(WVE)
-WAS-TAM連携の構成パターン
© 2012 IBM Corporation
98



ここでは、Webシステムのトポロジーを考える際に検討すべき主要なポイントをあげてみました。

これらのポイントの全てにおいて、全ての要件を満たすことは困難ですので、それぞれの要件の重要度を鑑みたうえで、どの要件を優先させて構成を組んでいくか検討することになります。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 WebシステムとコンポーネントとWLM
2-2 構成パターン
-トポロジーを考えるポイント
-基本構成パターン
-WebSphere Virtual Enterprise(WVE)
-WAS-TAM連携の構成パターン
© 2012 IBM Corporation
100

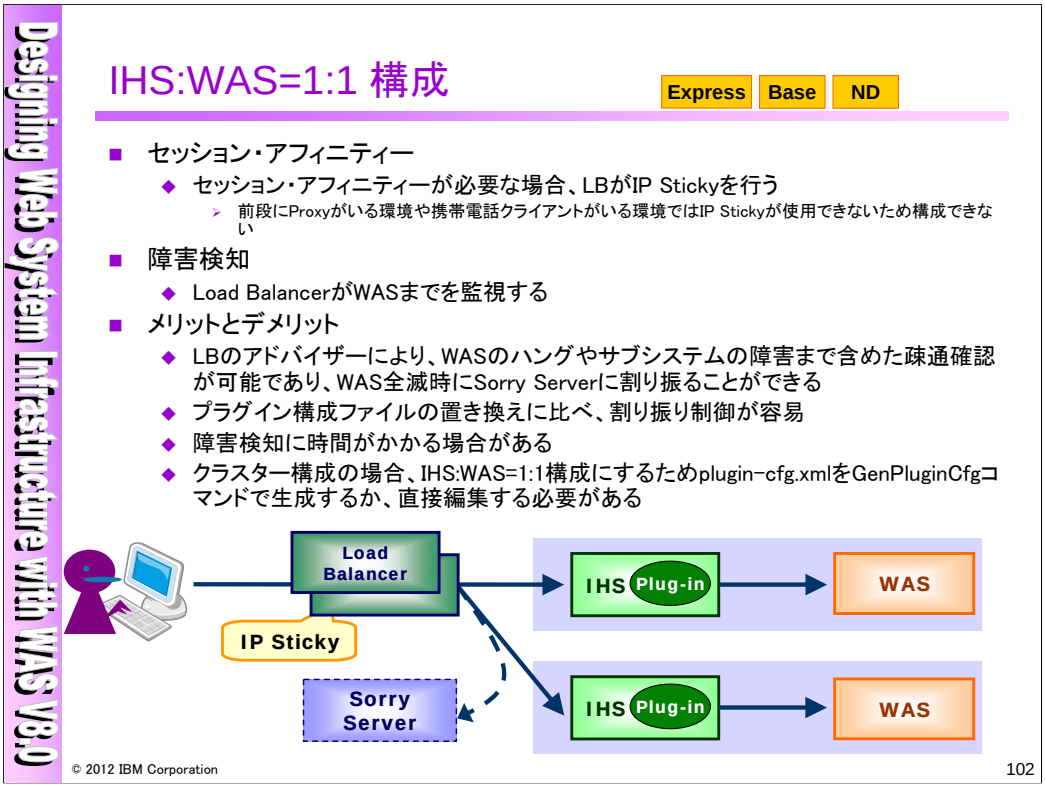


まず最も基本的なトポロジーとして、IHS:WASを1:Nとする構成を考えます。

この構成では、Load BalancerはMAC転送方式を用いて、特にアフィニティーを意識せずに割り振りを行います。セッション・アフィニティーはWebサーバー・プラグインがcloneidを見てアフィニティーの維持を行います。この構成の場合、Load Balancerから検知できるのはIHSの障害までです。WASの障害検知はIHS上のWebサーバー・プラグインによって行われます。

この構成のメリットとしては、デフォルトの構成であるために構成がシンプルです。セッション・アフィニティーもデフォルトの設定で行われますので、特に追加での設定は必要ありません。WAS側の設定でJSESSIONIDのやり取りにCookieではなくURL再書き込みを用いれば、携帯電話のクライアントにも対応可能です。

この構成のデメリットとしては、WASの障害検知がWebサーバー・プラグインによって行われるため、アプリケーション・サーバーのハングに対して考慮が必要なことがあります。Load Balancerで行えるようなアプリケーション・サーバーと連携するサブ・コンポーネントまで含めた障害検知を行うこともできません。アプリケーション・サーバーのダウンを検知しても、特に通知をあげる方法がありませんので、別途Tivoli製品を用いてプロセス監視などを行うなど通知の仕組みを考える必要があります。また、Sorry Serverを使用する場合、Load Balancerが監視するのはWASではなくIHSであるため、WASに障害が発生してもIHSが正常であればSorry Serverに割り振ることができません。そのため、WASの障害時にはErrorDocumentで対応するか、このあとのスライドで説明しますが、1:N構成でもLoad Balancerを使ってWASまでの監視を行う方法を検討する必要があります。また、メンテナンスのために例えば片系のみを停止しようとする場合、各IHSノードでプラグイン構成ファイルの置き換え作業が必要になりますので、Load Balancerで割り振り制御する場合と比べ運用負荷が高くなります。

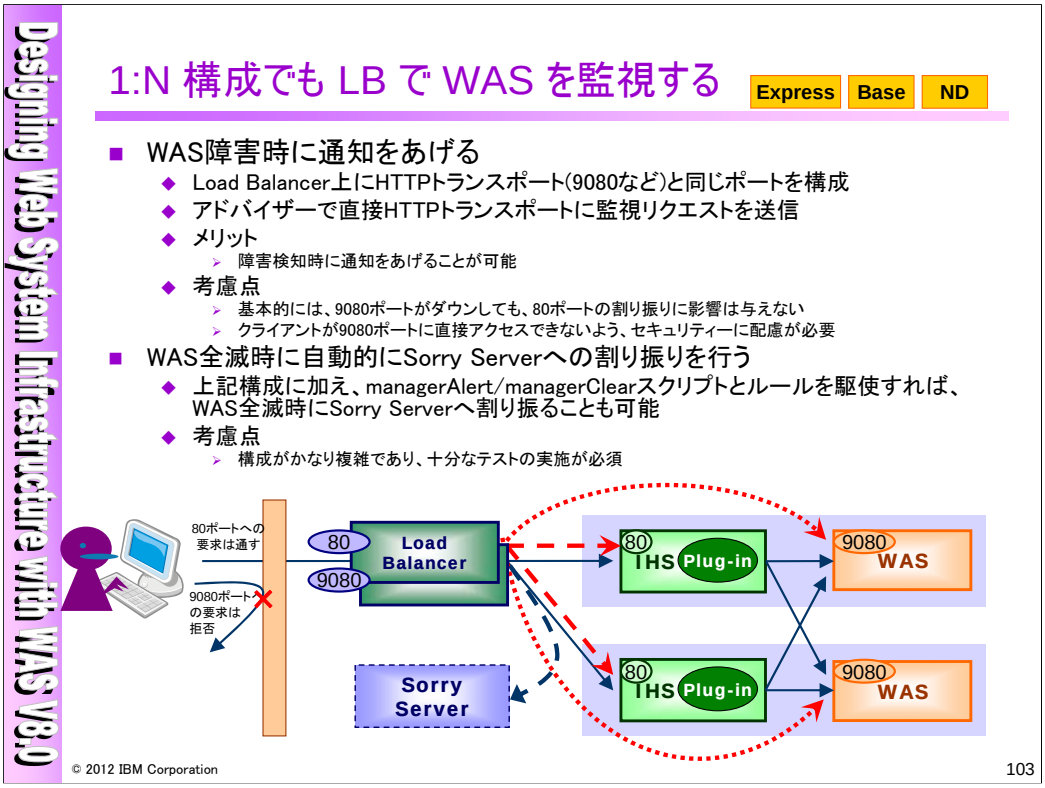


続いて、IHS:WASを1:1とする構成を考えます。

この構成では、Load Balancerは1:N構成の場合と同様にMAC転送方式で割り振りを行いますが、セッション・アフィニティー必要な場合にはLoad BalancerでIP Stickyを用いてセッション・アフィニティーを行います。IP Stickyは前段にプロキシー・サーバーが存在する環境では使用することができないため、そういった環境ではそもそもこの構成を取ることができません。

この構成のメリットとしては、障害検知をLoad Balancerが行うため、アプリケーション・サーバーおよび連携するサブコンポーネントまでの障害検知をLoad Balancerで行うことができる点です。プラグインでは考慮が必要であったアプリケーション・サーバーのハング検知が可能であり、障害検知時に通知をあげたり、Sorry Serverへ割り振ることも可能です。

この構成のデメリットとしては、デフォルトではIHS:WASは1:N構成となりますので、1:1構成に設定するためにplugin-cfg.xmlの手動編集またはGenPluginCfg.shを使用してのplugin-cfg.xmlの編集が必要になる点です。台数が少なければそれほど煩雑ではありませんが、筐体数が多い場合には台数分の編集作業が発生しますので初期構成時の作業負荷が高くなります。また、Load Balancerの障害検知では、障害の検知までにタイムラグがあることにも注意が必要です。

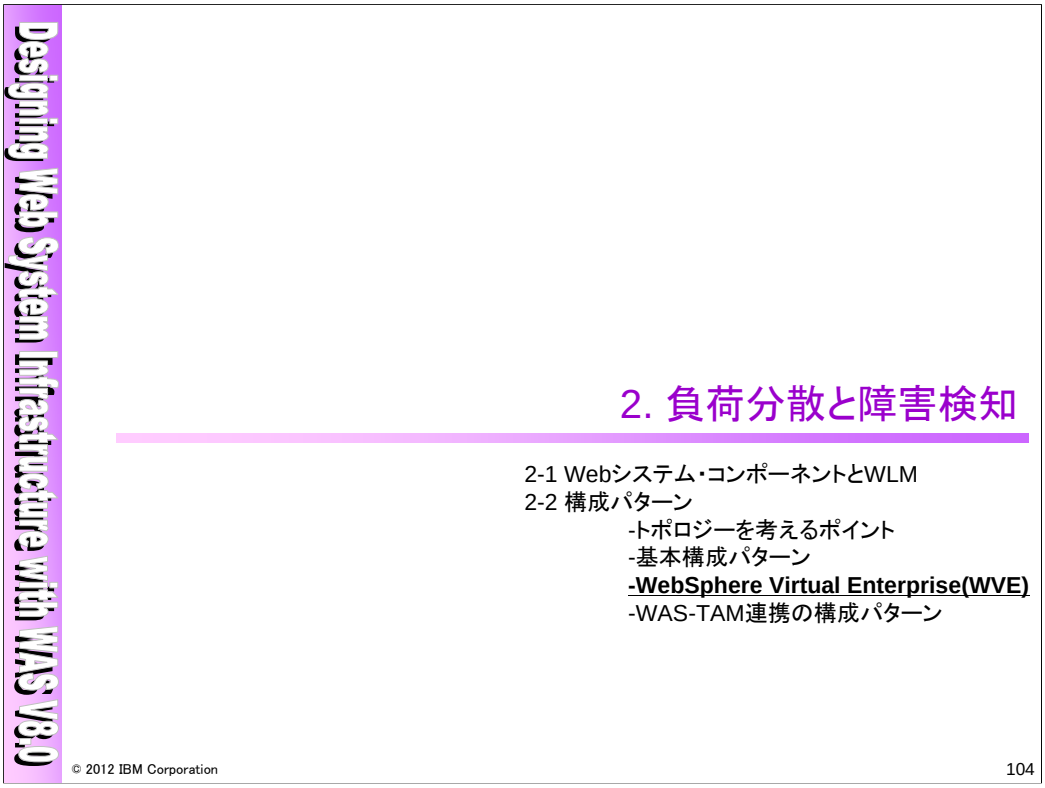


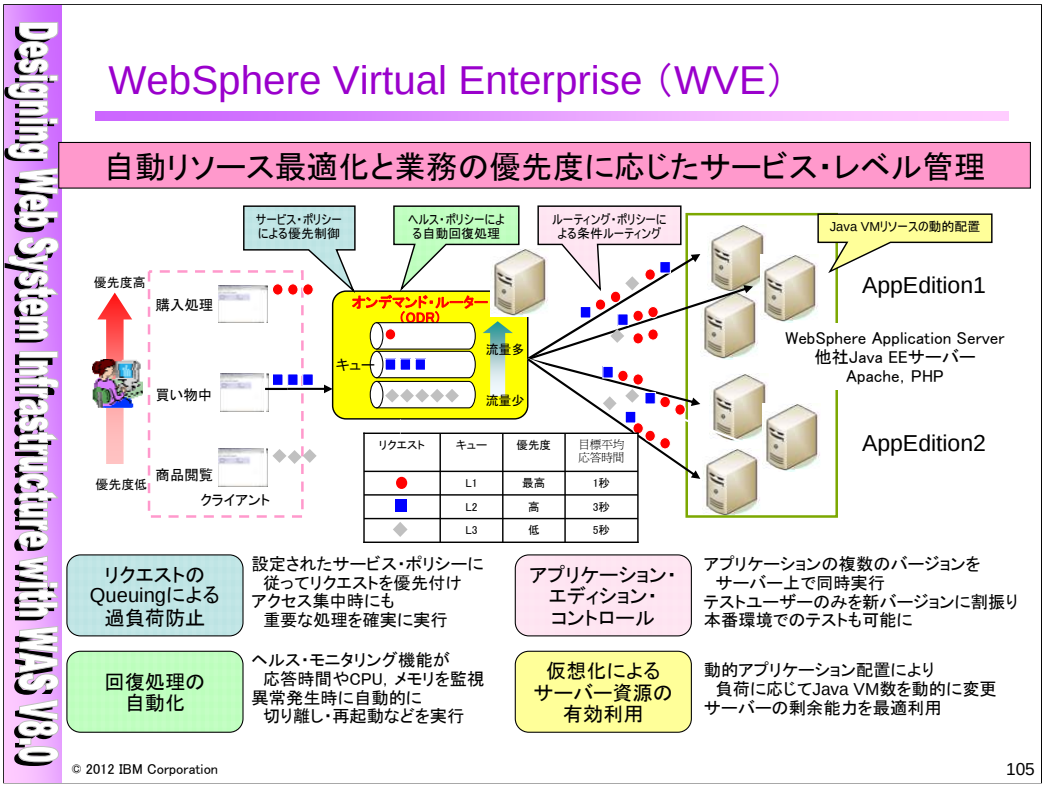
トポロジーとしてIHS:WASが1:N構成を採用した場合に、Load BalancerでWASの障害を検知する方法を紹介します。この構成を取ることにより、WASの障害をLoad Balancerで検知し、アラートをあげることが可能になります。

この構成では、Load Balancer上にHTTPトランスポートと同じポートを定義して監視目的に利用します。Load Balancerから9080ポートに対して直接アドバイザーの監視リクエストが飛びますので、WASの障害検知に利用することができますし、障害検知時には通知を上げることも可能になります。

考慮点として、たとえ9080ポートでWASの障害を検知したとしても、通常のクライアント・リクエストが処理される80ポートの割り振りにはまったく影響を与えません。あくまで監視・通知目的でLoad Balancerが使用できるのみです。またLoad Balancer上の9080ポートにクライアントからアクセスされてしまうと、Load BalancerはWASポートに直接割り振りを行ってしまいます。アクセス・ログも残らないままクライアントにWASへとアクセスされてしまうことになりますので、セキュリティ・ホールとならないよう、前段のFirewallなどでLoad Balancer上の9080ポートにはクライアントから直接アクセスできないようにしておくことが必須です。

もし、通知をあげるだけでなくWAS全滅時に自動的にSorry Serverへ割り振りを行いたい場合、managerAlert/managerClearスクリプトとルール機能を駆使することで実現可能です。この場合、managerAlertが呼ばれた際に、このスクリプトからdscontrol server downコマンドを2台のIHSに対して発行し、それによってSorry Serverへの割り振りを行うようにします。ただし構成がかなり複雑になる上、managerAlert/managerClear/serverDown/serverUpコマンドの挙動には癖があるため、十分なテストの実施が必須となります。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 WebシステムコンポーネントとWLM
2-2 構成パターン
-トポロジーを考えるポイント
-基本構成パターン
-WebSphere Virtual Enterprise(WVE)
-WAS-TAM連携の構成パターン
© 2012 IBM Corporation
104



複数アプリケーションがあり、それらの処理の優先度をつけたい等、よりインテリジェントな要件がある場合、WebSphere製品のWebSphere Virtual Enterpriseで提供されるODRと呼ばれるプロキシー・コンポーネントを使用することで、そうした要件を実現させることが可能です。

WebSphere Virtual Enterpriseに関して、詳しくは下記リンク先資料をご参照ください。

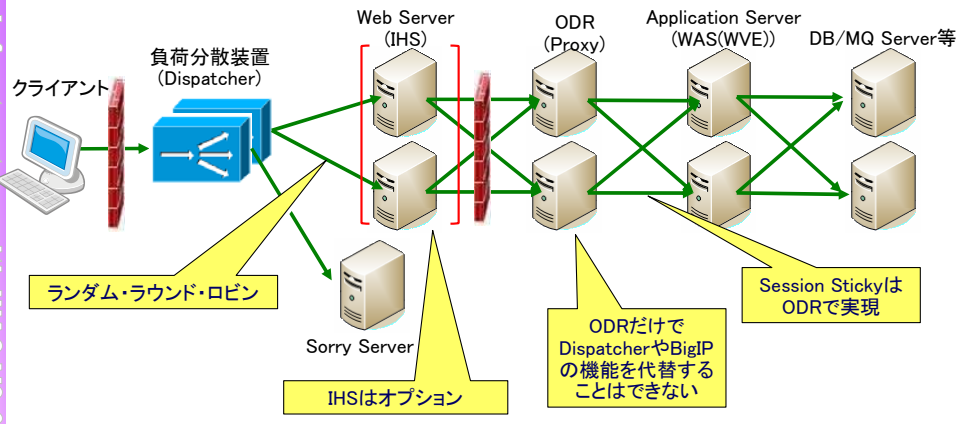
developerWorks: WebSphere Virtual Enterprise

http://www.ibm.com/developerworks/jp/websphere/category/wxd/wve.html

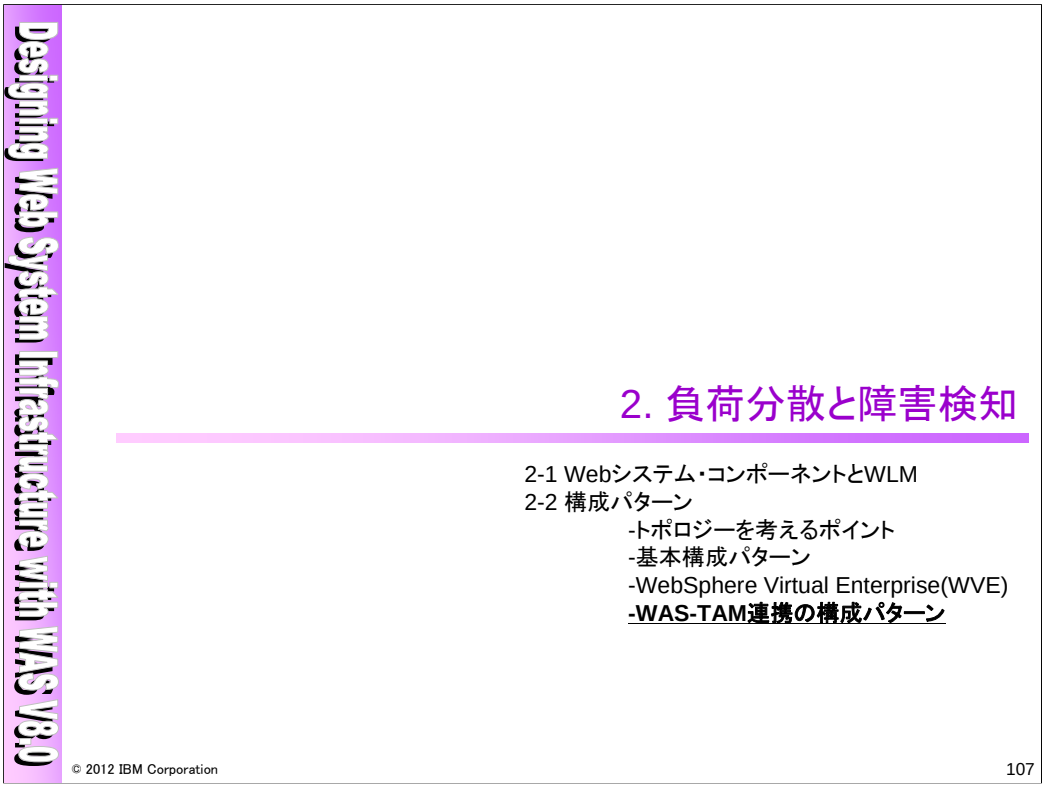
Designing Web System Infrastructure with WAS V8.0

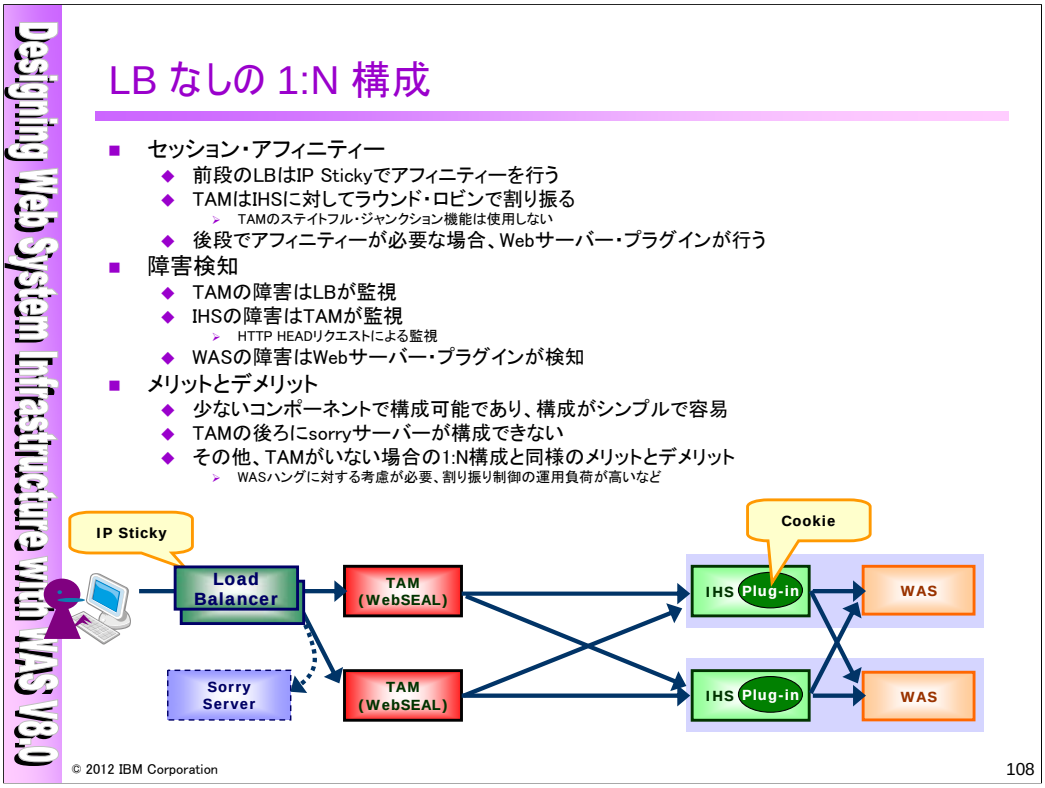
LB+WVE ODR構成

- WVE ODRによる負荷分散
  - 構成パターン(1)にODRを追加したパターン(LBはEdge以外でも可)
  - ODRによるポリシー・ベースの自動ルーティングと流量制御
  - ODRはProxy Serverの機能を拡張したものであり、WAS同様にActive-Active構成で可用性を担保
- Sessionアフィニティーは ODRの機能で実現
- SSLの解除はIHSもしくはODRで行う



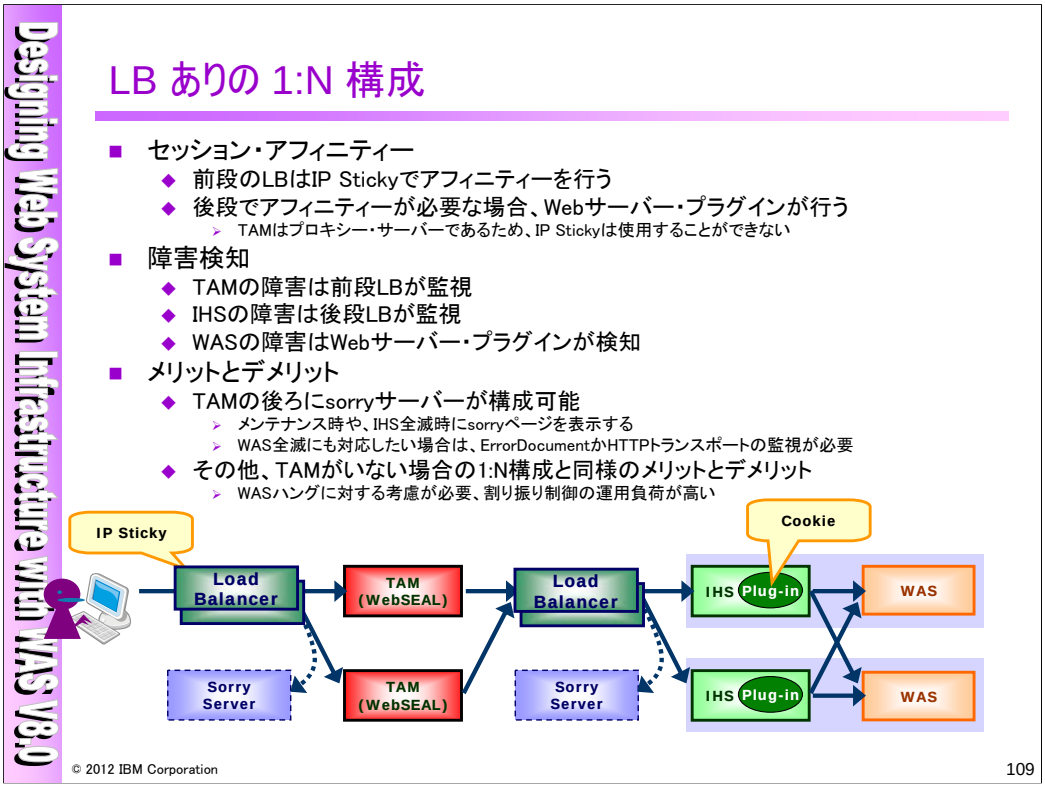
© 2012 IBM Corporation
106


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 WebシステムコンポーネントとWLM
2-2 構成パターン
-トポロジーを考えるポイント
-基本構成パターン
-WebSphere Virtual Enterprise(WVE)
-WAS-TAM連携の構成パターン
© 2012 IBM Corporation
107



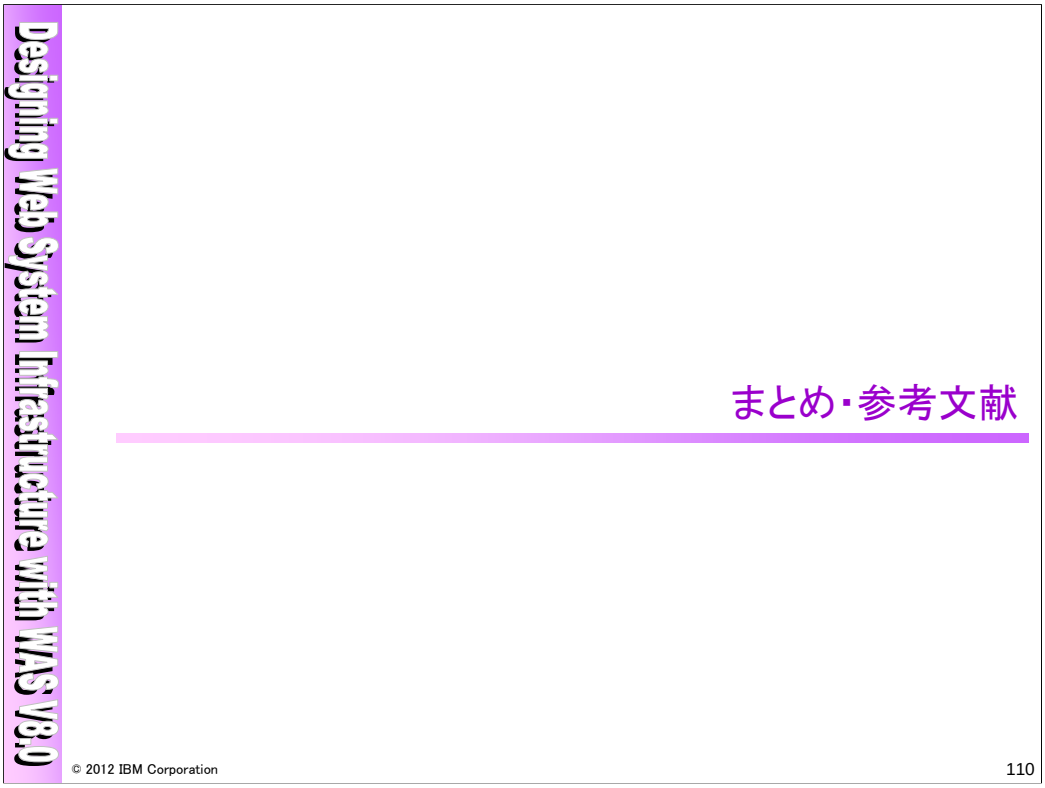
ここからは、シングル・サインオンを実現するIBM製品であるTivoli Access Manager for e-business(TAM)とWASを連携する場合のトポロジーについて説明します。TAMではリバース・プロキシーとして認証・アクセス制御・シングルサインオンを実現するコンポーネントとしてWebSEALというソフトウェアがあります。

まず、最も基本的な構成として、TAMの前段のみにLoad Balancerを配置する構成があります。



つづいて、TAMとIHSの間にもLoad Balancerを配置する構成です。この構成では、TAMがプロキシーとして動作するため、IHSとWASは1:Nとして構成し、Webサーバー・プラグインでアフィニティーを行う必要があります。

この構成のメリットはTAMの後ろにSorry Serverが配置できることです。TAMから複数のシステムやアプリケーションに対して割り振りを行っていて、システム毎に運用が異なるケースでは、このSorry Serverは大変有用となります。


Designing Web System Infrastructure with WAS V8.0
まとめ・参考文献
© 2012 IBM Corporation
110

Designing Web System Infrastructure with WAS V8.0

まとめ

- WASトポロジー
  - WAS基本コンポーネント
    - Webシステムの各コンポーネントの役割
  - WAS基本コンポーネントの配置
    - 各コンポーネントを配置する上での考慮事項
  - WASの提供する高可用性・高パフォーマンス機能
    - 高可用・高パフォーマンスを実現するための機能紹介
- 負荷分散と障害検知
  - Webシステム・コンポーネントとWLM
    - WebシステムのコンポーネントとWLMの仕組み
  - 構成パターン
    - Webシステムのトポロジー・パターンの紹介と選択基準

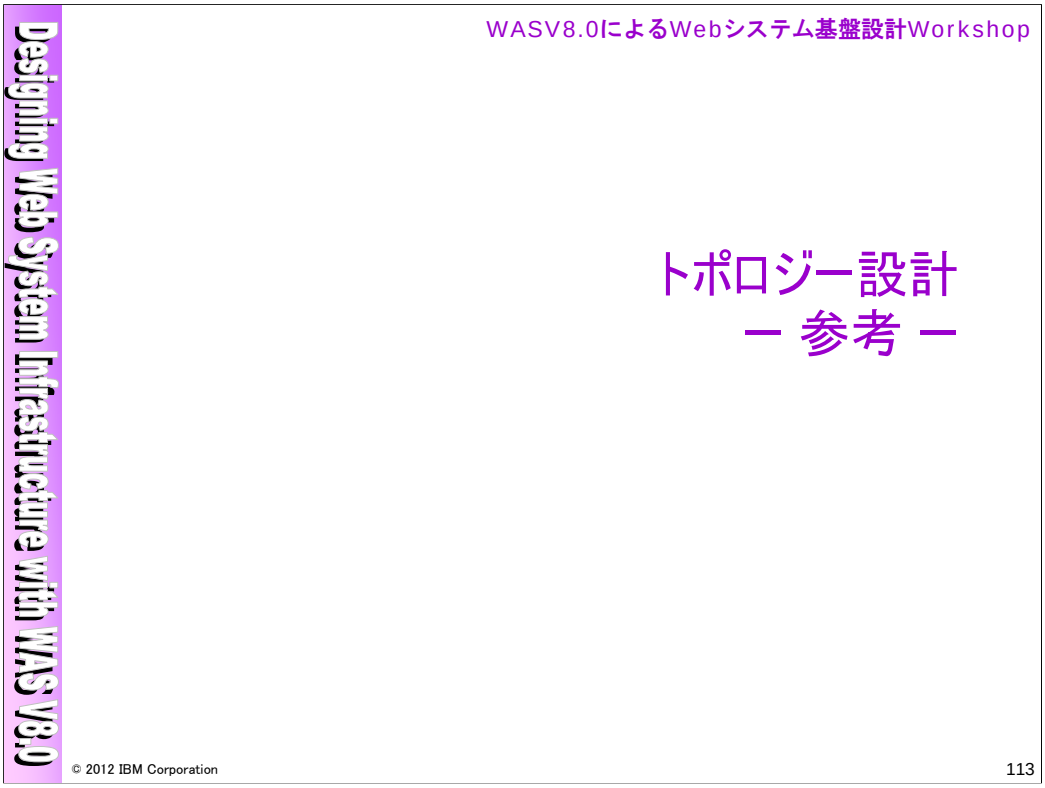
© 2012 IBM Corporation

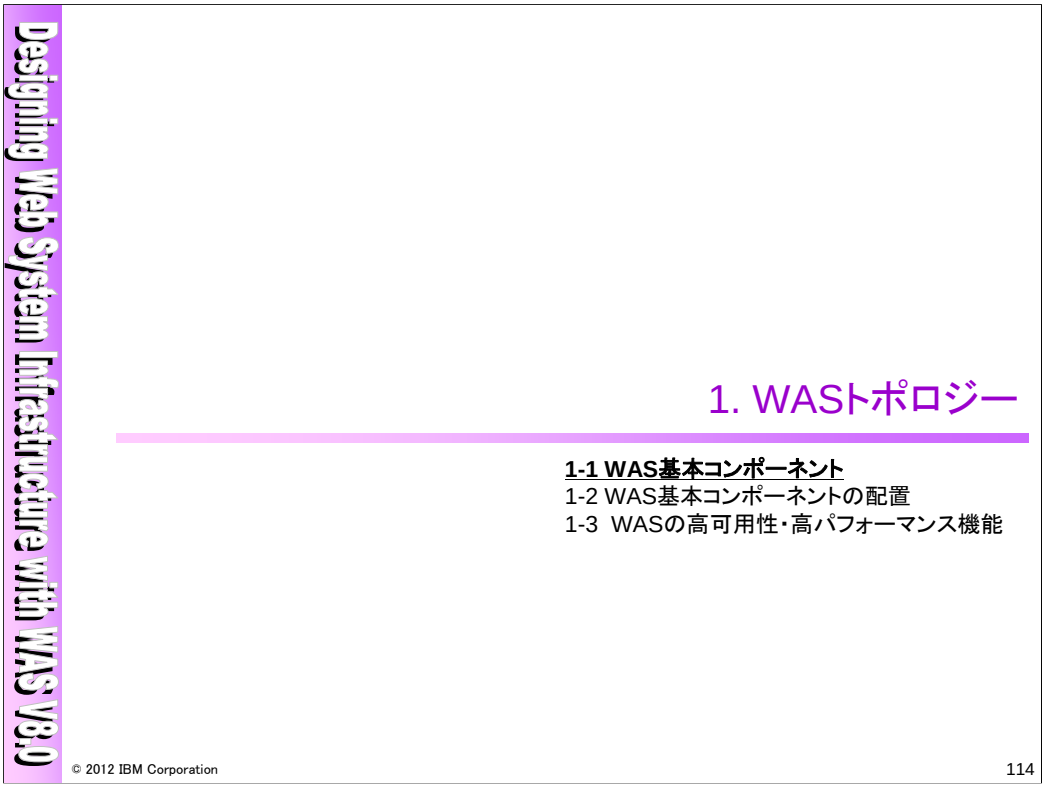
Designing Web System Infrastructure with WAS V8.0

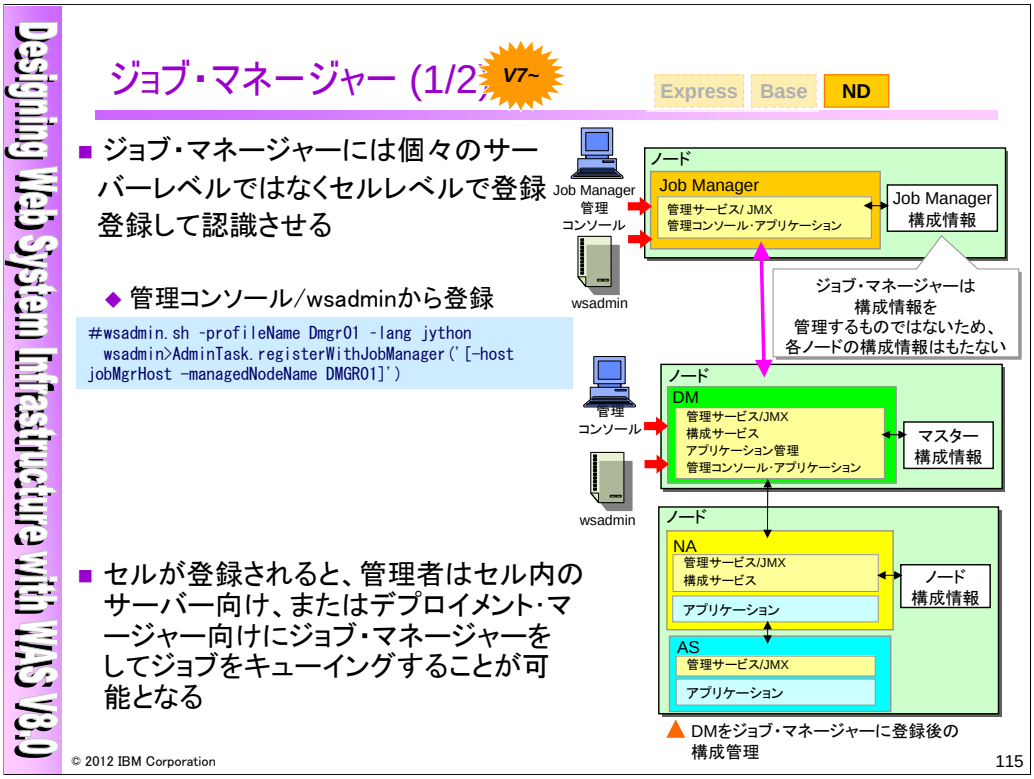
参考文献

- Information Center
  - WebSphere Application Server V8.0
    - http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp
- ワークショップ資料
  - WebSphere Application Server V8 アナウンスメント・ワークショップ
    - http://www.ibm.com/developerworks/jp/websphere/library/was/was8_ws/
  - WebSphere Application Server V7.0によるWebシステム基盤設計ワークショップ
    - http://www.ibm.com/developerworks/jp/websphere/library/was/was7_guide/index.html
  - DataPower XC10 キャッシングアプライアンス ワークショップ資料
    - http://www.ibm.com/developerworks/jp/websphere/library/esb/dpxc10_ws/

© 2012 IBM Corporation


Designing Web System Infrastructure with WAS V8.0
WASV8.0によるWebシステム基盤設計Workshop
トポロジー設計
ー 参考 ー
© 2012 IBM Corporation
113


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
114



NDセル環境をジョブ・マネージャーの管理対象にするにはデプロイメント・マネージャー・ノードを登録する必要があります。デプロイメント・マネージャー・ノードを登録する場合には、デプロイメント・マネージャーの管理コンソール、もしくはwsadminから行います。管理コンソールから行う場合には、ナビゲーションツリーの[システム管理]>[デプロイメント・マネージャー]>[ジョブ・マネージャー]をクリックし、『ジョブ・マネージャーに登録』をクリックしてください。登録を解除する場合には『ジョブ・マネージャーの登録抹消』をクリックしてください。wsadminから登録する場合にはAdminTaskオブジェクトのregisterWithJobManagerコマンドオプションを使用します。-portでは管理セキュリティを設定していない場合にはジョブ・マネージャーrのWC_adminhost(デフォルト9960)を指定してください。管理セキュリティを設定している場合にはWC_adminhost_secure(デフォルト9943)を指定してください。またジョブ・マネージャーからデプロイメント・マネージャーの登録を解除するにはAdminTaskオブジェクトでunregisterWithJobManagerコマンドオプションを使用してください。

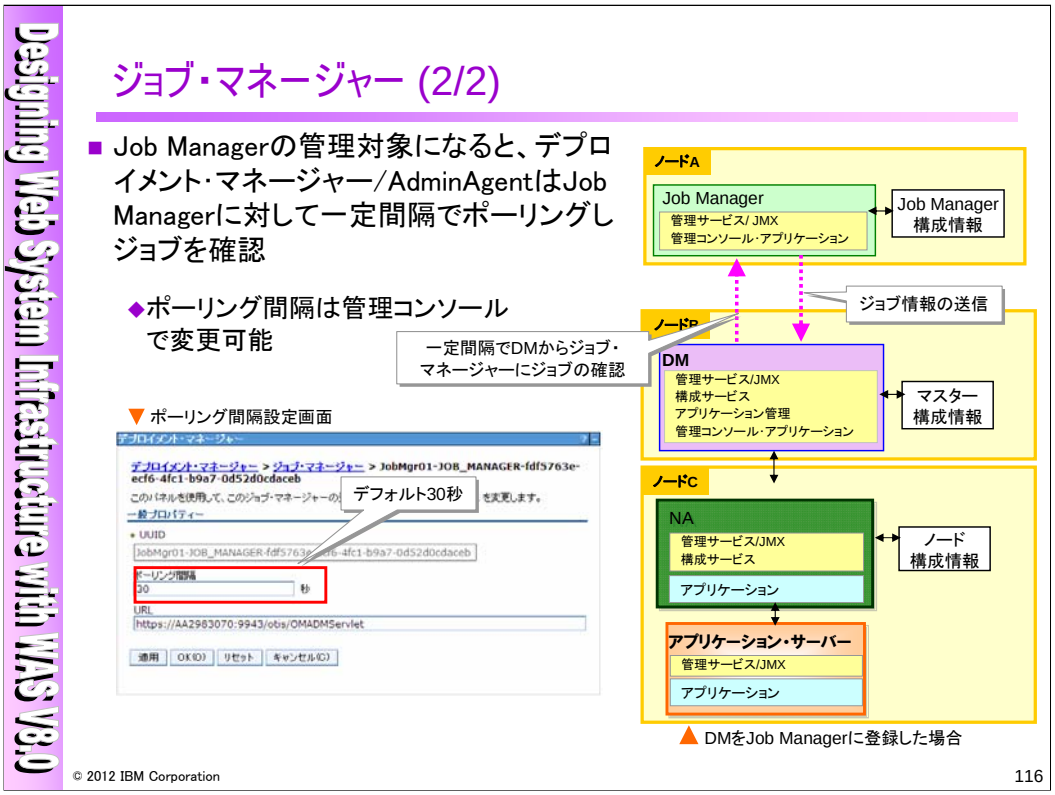
コマンドオプションの引数の詳細などは以下のInfoCenterの記載をご参照ください。

WAS V8 InfoCenter「wsadmin スクリプトを使用する AdminTask オブジェクトの ManagedNodeAgent コマンド・グループ」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rxml_7managednodeagent.html

ジョブ・マネージャーにデプロイメント・マネージャー・ノードを登録すると、ジョブ・マネージャーはデプロイメント・マネージャーに向けてジョブを非同期に実行することが可能です。ただしジョブ・マネージャーは構成情報を管理するものではないため、各ノードの構成情報は持ちません。

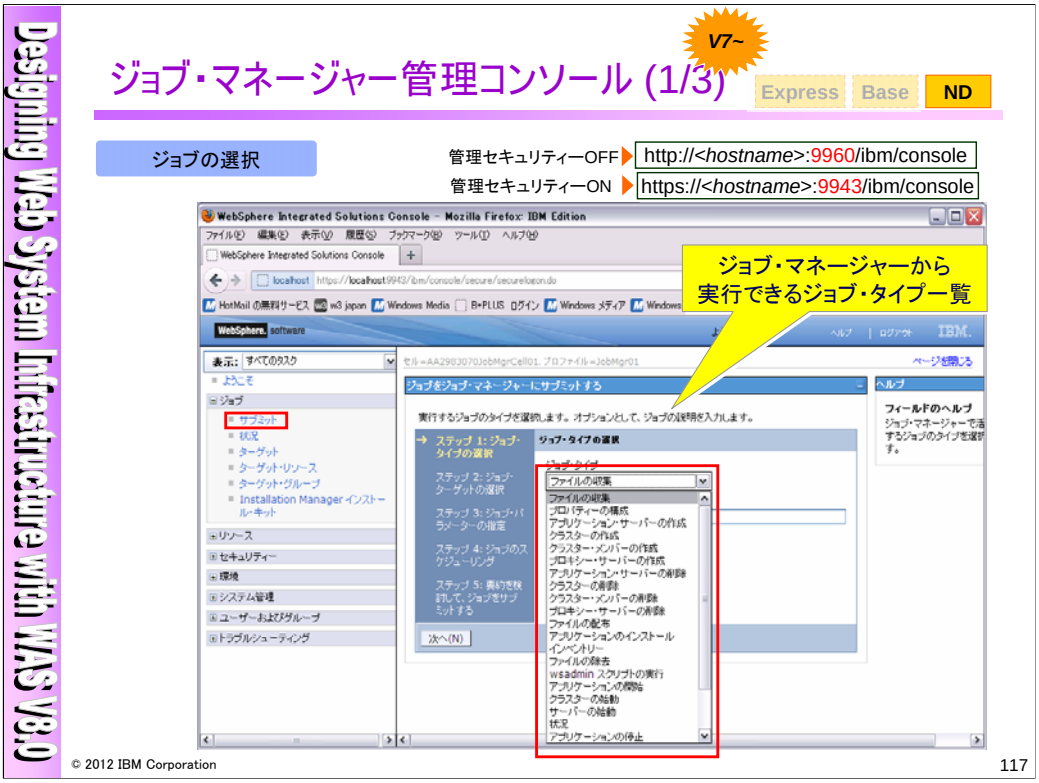
ジョブ・マネージャーへのBase/Express環境のシングル・サーバー・ノードの登録については後述します。



ジョブ・マネージャーに登録して管理対象になると、デプロイメント・マネージャーと管理エージェントはジョブ・マネージャーに対してジョブの有無を確認するために一定間隔でポーリングを行います。デフォルトではこの間隔は30秒です。この間隔はデプロイメント・マネージャー管理コンソールもしくは管理エージェント管理コンソールで変更可能です。

デプロイメント・マネージャー管理コンソールから変更する場合はナビゲーション・ツリーの[システム管理]>[デプロイメント・マネージャー]>[ジョブ・マネージャー]>[*ジョブ・マネージャーUUID*]をクリックし、ポーリング間隔を変更してください。

管理エージェント管理コンソールから変更する場合は[システム管理]>[管理エージェント]>[ノード]>[*ノード名*]>[ジョブ・マネージャー]>[*ジョブ・マネージャーUUID*]をクリックし、変更してください。



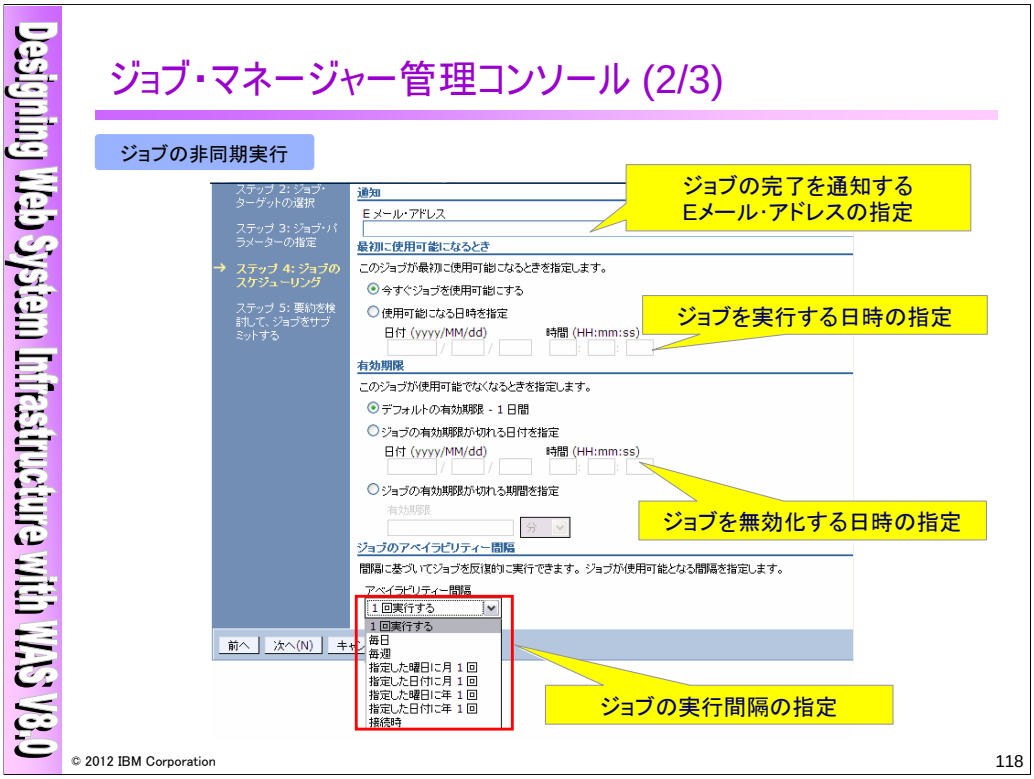
こちらはジョブ・マネージャーの管理コンソール画面です。ジョブ・マネージャー管理コンソールからはジョブのサブミットやジョブのステータス確認 が出来るほか、ジョブ・マネージャー自身のセキュリティ設定やログの設定を行うことが出来ます。また、管理対象のノードのリソース(アプリケーション・サーバー/クラスター/アプリケーション)名と起動・停止状況を閲覧することができます。ただし、管理対象ノードのリソースに対して起動・停止の操作や構成変更を行うことは出来ません。ジョブ・マネージャー管理コンソールを使用する場合には、http://<*hostname*>:<ポート番号>/ibm/consoleにアクセスして下さい。この時のポート番号は、該当のジョブ・マネージャーのWC_adminhost(デフォルト9960)を指定してください。管理セキュリティを設定している場合はWC_adminhost_secure(デフォルト9943) を指定して下さい。

ジョブ・マネージャーからジョブを実行する場合にはナビゲーション・ツリーから[ジョブ]>[サブミット]をクリックします。そこから実行するジョブ・タイプを選択してください。

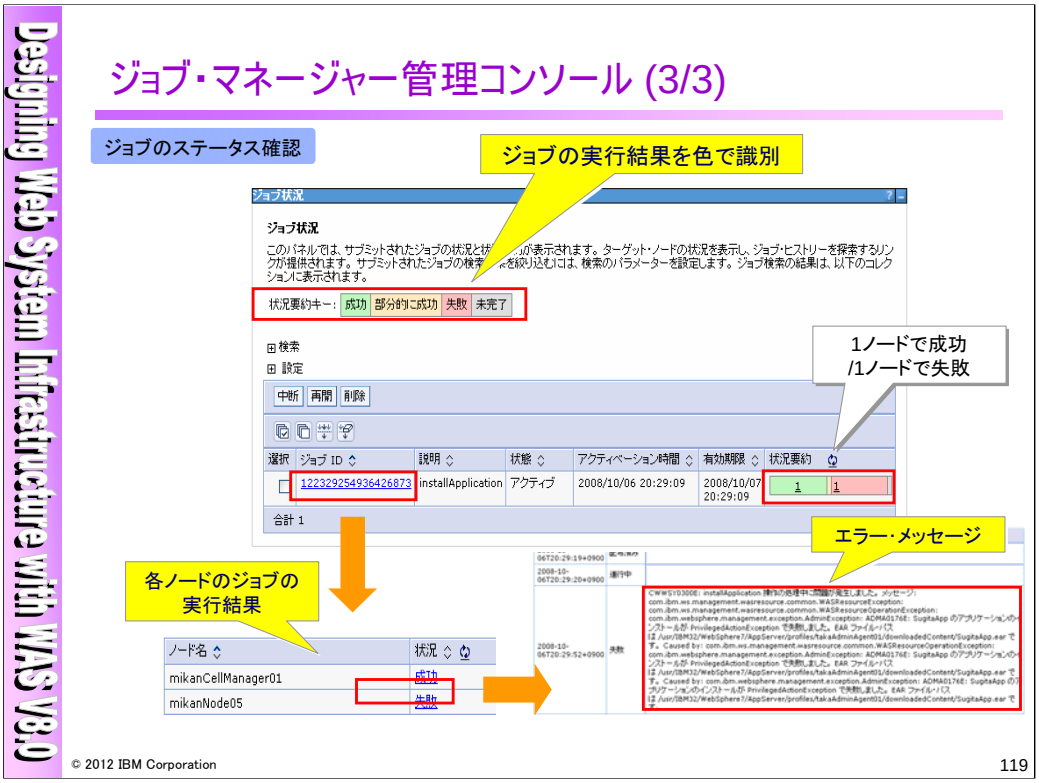
ジョブのサブミット手順の詳細は下記のInfoCenterの詳細をご参照ください。

WAS V8 InfoCenter「ジョブのサブミット」

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



こちらはジョブの実行日時や、ジョブを繰り返し実行する場合には実行間隔を指定できる画面です。ジョブは日時を指定するのではなく、今すぐ実行させることも可能です。また、ジョブを無効化するためのジョブの有効期限を指定することも可能です。デフォルトでは有効期限は1日間です。これらはジョブをサブミットするステップの途中で指定できます。



こちらはジョブをサブミットした後に、ジョブのステータスを確認する画面です。ナビゲーション・ツリーから[ジョブ]＞[状況]をクリックします。画面にはジョブID/説明/状態/アクティベーション時間/有効期限/状況要約が表示されます。ジョブIDはサブミットしたジョブを特定するIDです。説明はサブミットしたジョブのタイプを表します。状態ではジョブの実行結果ではなく、ジョブが実行されたかどうかを表すものです。状態には[アクティブ/中断状態/保留/期限切れ]の4種類あります。アクティベーション時間はジョブが実行可能になる時間を示します。有効期限はジョブを無効化する日時を示します。状況要約ではジョブがサブミットされたノード数とジョブの実行結果が表示されます。ジョブの実行結果は色で識別します。実行結果の色分けは以下になります。

［緑色:成功/黄色:部分的に成功/赤色:失敗/灰色:未完了］

上記画面の例では状況要約で緑色の枠に1、赤色の枠に1と表示されていますので、ジョブが成功したノード数は1、失敗したノード数は1と確認できます。

ジョブのステータス確認方法の詳細については下記のInfoCenterの記載もご参照ください。

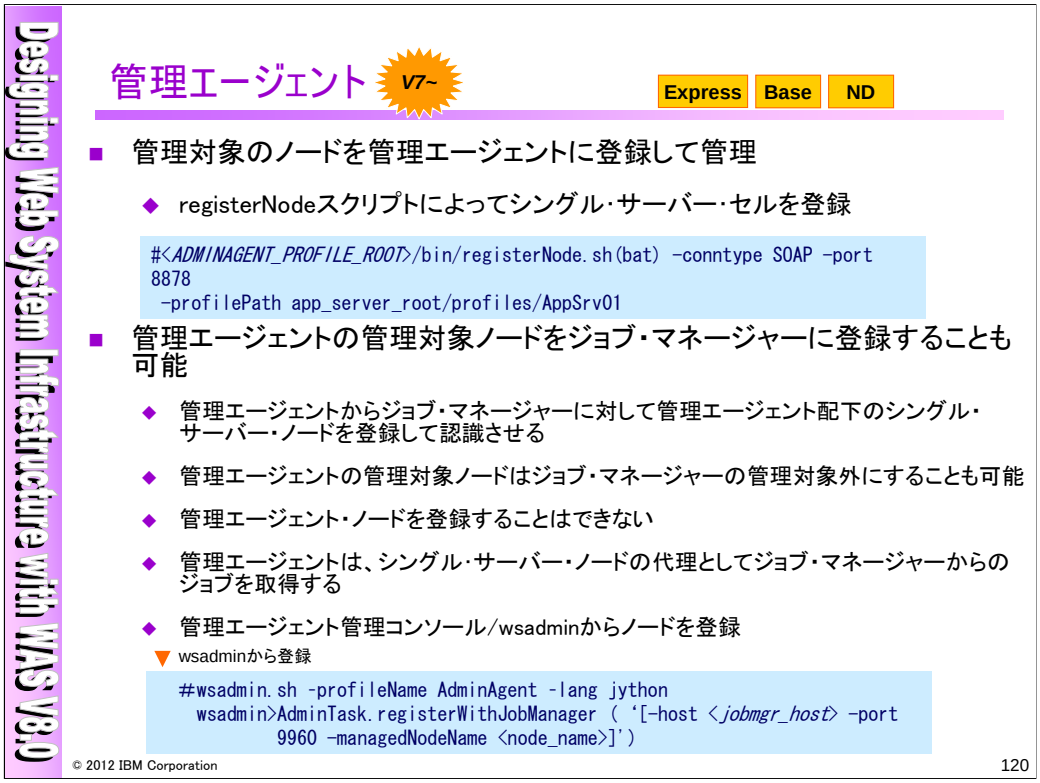
WAS V8 InfoCenter「ジョブ状況の確認」

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

WAS V8 InfoCenter「ジョブ状況コレクション」

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

ジョブIDをクリックしますと、ジョブをサブミットした各ノードでのジョブの実行結果が確認できます。



シングル・サーバー・ノードのプロファイルを管理エージェントに登録するにはregisterNodeスクリプトを使用します。registerNodeスクリプトは<*ADMINAGENT_PROFILE_ROOT*>/binディレクトリにあります。管理エージェントからノードの登録を解除する場合にはderegisterNodeスクリプトを使用します。

コマンドの引数の詳細などは以下のInfoCenterの記載をご参照ください。

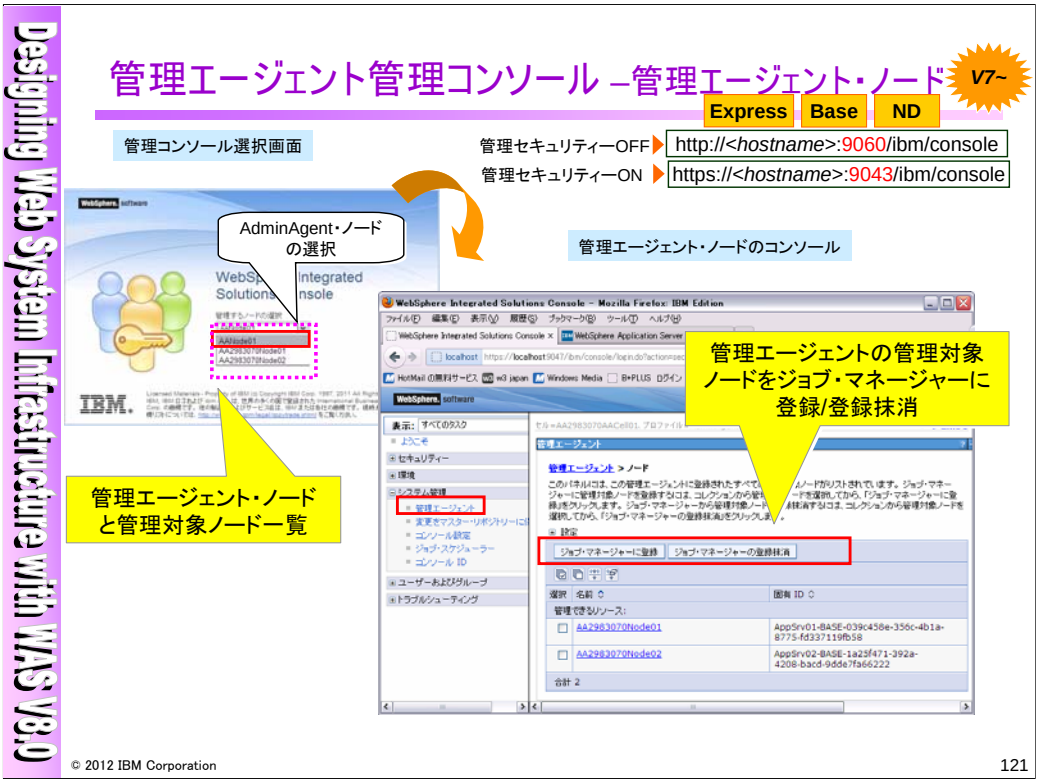
WAS V8 InfoCenter「registerNode コマンド」

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

WAS V8 InfoCenter「deregisterNode コマンド」

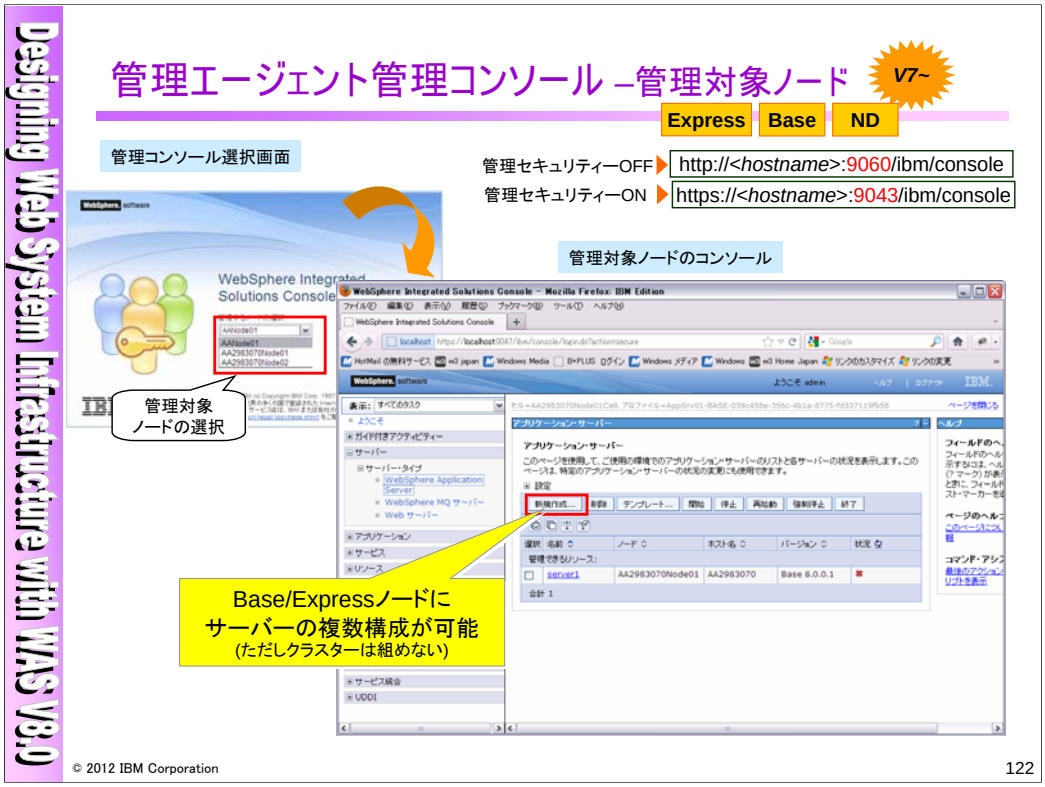
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/ragt_deregisterNode.html

また管理エージェントの管理対象ノードをジョブ・マネージャーに登録することで、管理エージェントは管理対象のシングル・サーバー・ノードの代理としてジョブ・マネージャーからジョブを受け取り、そのノードに対してジョブを実行することが可能です。管理エージェントノードはジョブ・マネージャーに登録できないのでご注意ください。管理エージェントの管理対象ノードをジョブ・マネージャーに登録するには管理コンソールもしくはwsadminから登録が可能です。管理コンソールからの登録については後ほど管理エージェント管理コンソールの項で解説いたします。wsadminで登録する場合にはAdminTaskオブジェクトのregisterWithJobMangerコマンドオプションを使用して登録します。登録を解除する際にはAdminTaskオブジェクトのunregisterWithJobManagerコマンドオプションを使用します。



こちらは管理エージェント・ノードの管理コンソールです。管理エージェント・ノードの管理コンソールでは管理エージェント自身のセキュリティ設定やログの設定を行うことが可能です。また、管理エージェントの管理対象ノードをジョブ・マネージャーに登録、もしくは登録抹消することが可能です。

上記画面は管理エージェントの管理対象ノードをジョブ・マネージャーに登録する、もしくは登録から抹消する画面です。ナビゲーション・ツリーから[システム管理]＞[管理エージェント]＞[ノード]をクリックしてください。該当のノードにチェックを入れ、『ジョブ・マネージャーに登録』もしくは『ジョブ・マネージャーの登録抹消』をクリックしてください。

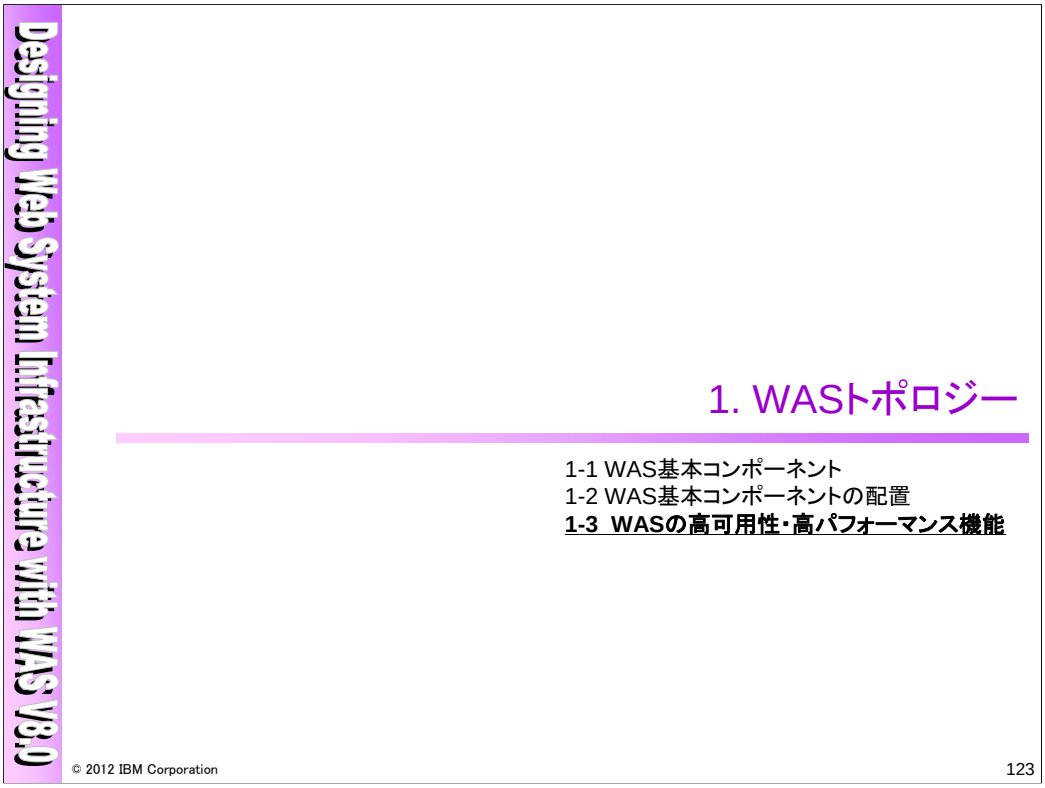


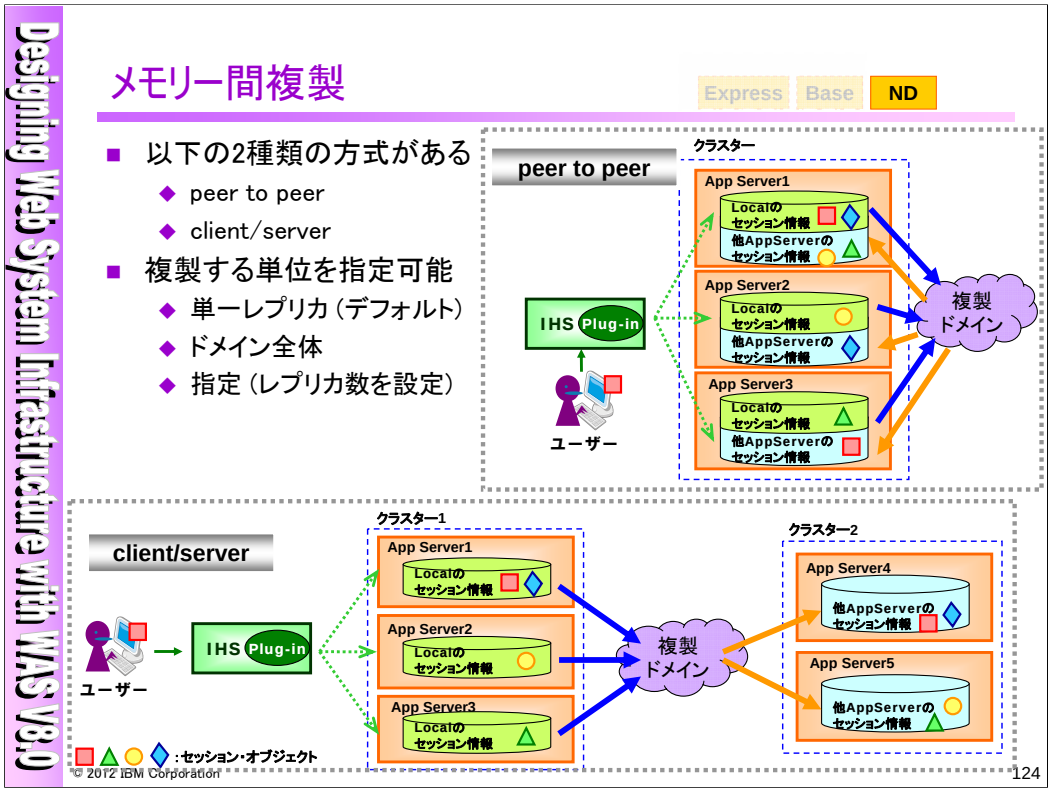
シングル・サーバー・ノードが管理エージェントに登録されて管理対象ノードになると、直接管理対象ノードの管理コンソールにアクセス出来なくなります。そのため、管理対象ノードの管理コンソールを使用するには管理エージェントの管理コンソールにアクセスする必要があります。

管理エージェント管理コンソールは、http://<*hostname*>:<ポート番号>/ibm/consoleにアクセスして下さい。この時のポート番号は、該当の管理エージェントのWC_adminhost(デフォルト9060)を指定してください。管理セキュリティを設定している場合はWC_adminhost_secure(デフォルト9043) を指定して下さい。

アクセスすると、管理エージェント・ノードと管理エージェントの管理対象ノードを選択する画面が表示されます。そこで該当の管理対象ノードを選択すると、管理対象ノードの管理コンソールが使用できます。

管理対象ノードの管理コンソールは従来のシングル・サーバー・ノードの管理コンソールと同じように、アプリケーション・サーバーやアプリケーションの構成情報の変更、アプリケーション・サーバーの起動・停止などの状態の変更や、ほとんどのWAS構成情報の変更を行うことができます。追加機能として、管理エージェントの管理対象ノードになると、管理コンソールからノード上に複数のアプリケーション・サーバーを構成することが可能になります。ただし、クラスターを組むことは出来ません。


Designing Web System Infrastructure with WAS V8.0
1. WASトポロジー
1-1 WAS基本コンポーネント
1-2 WAS基本コンポーネントの配置
1-3 WASの高可用性・高パフォーマンス機能
© 2012 IBM Corporation
123



メモリー間複製では複製モードとして、peer to peer もしくは client/server を選択できます。

・peer to peerモード

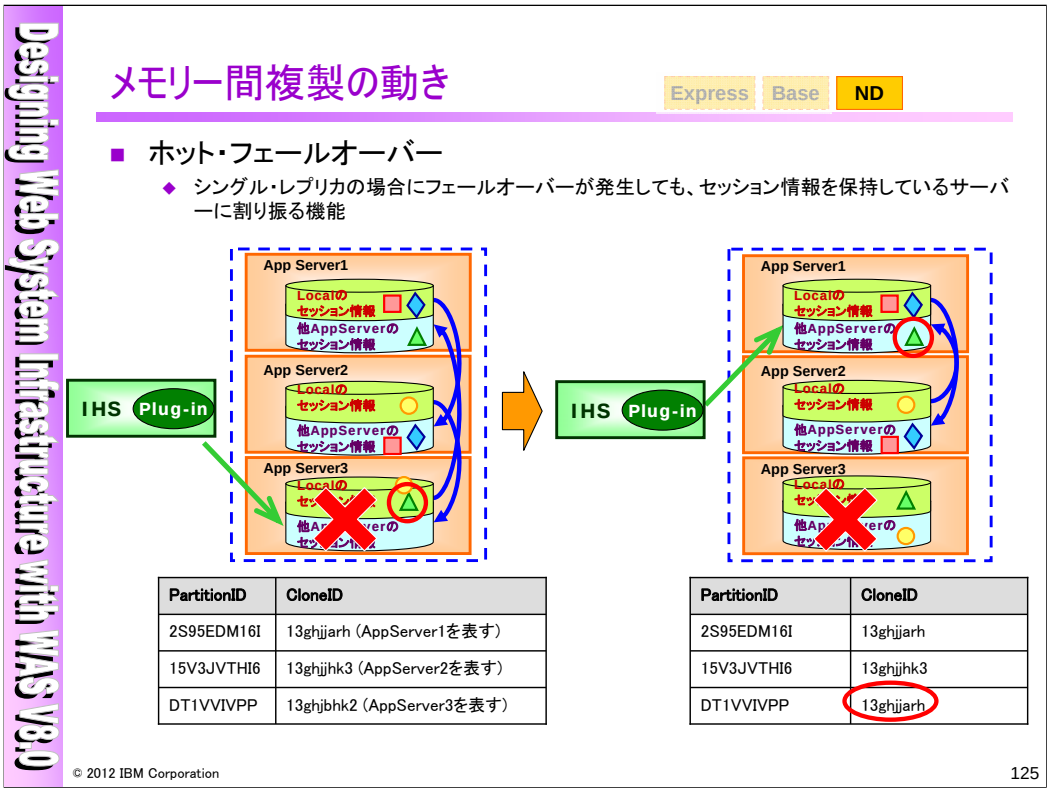
リクエスト処理を行うクラスターのアプリケーション・サーバー間でセッションを保持する方法です。デフォルトでは1つのセッション・データに関するレプリカが他サーバー上に1つ存在します。

・client/server

リクエスト処理を行うクラスターとは別に、セッション・オブジェクトを保持するためのクラスターを構成する方法です。

・ホット・フェールオーバー

アプリケーション・サーバーに障害が発生すると、WebコンテナーWLM（Webサーバー・プラグイン）がレプリカのあるアプリケーション・サーバーに割り振る機能です。



peer to peerの場合、Webサーバー・プラグインはPartitionIDを使用してWLMを行います。PartitionIDにはアプリケーション・サーバー固有の値であるCloneIDが紐付いており、このマッピングテーブルをWebサーバー・プラグインが保持します。

シングル・レプリカを使用している場合、セッション・オブジェクトは同じクラスター内の他の一台のアプリケーション・サーバー上で複製されます。サーバーがダウンした場合は、セッション・オブジェクトが複製されているサーバーへと処理が割り振られます。

・ サーバーダウン後の最初のリクエスト

1. Webサーバー・プラグインは、既存のマッピング・テーブルに従い、リクエストを割り振る

2. 割り振り先がダウンしている場合には、異なるサーバーに割り振る

3. アプリケーション・サーバーでは、まず、ローカルのセッション情報を問い合わせ、存在しない場合には、リモートのサーバーにセッション情報を問い合わせを行う

4. 割り振り先が変更されたリクエストのCookie情報(URL再書き込み情報)には、新しいPartitionIDが付加される

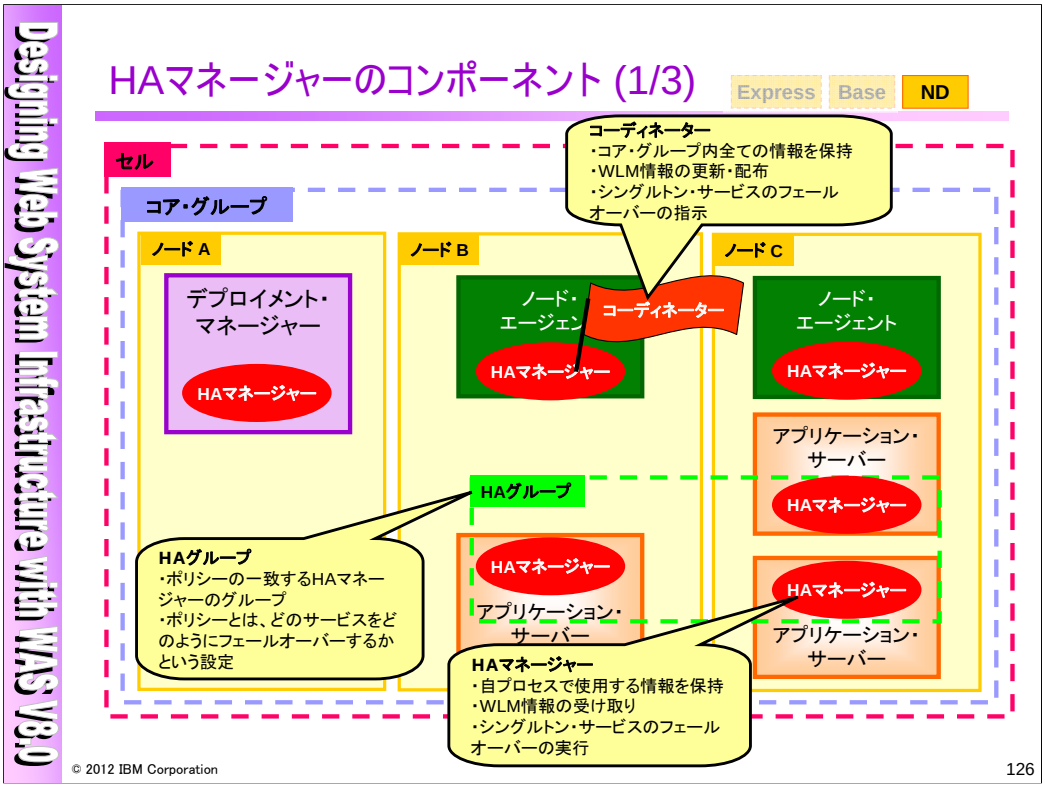
5. アプリケーション・サーバーは、レスポンスに新しいマッピング・テーブル情報を追加し、Webサーバー・プラグインに通知する

6. Webサーバー・プラグインは、マッピング・テーブル情報を取得する

・サーバーダウン後の2回目以降のリクエスト

1. 新しいマッピング・テーブルを使用し、セッション情報が存在するサーバーに割り振る

2.割り振り先が変更されたリクエストのCookie情報(URL再書き込み情報)には、新しいPartitionIDを付加される



HAマネージャー関連のコンポーネントとしては、HAマネージャーとコーディネーターの2つがあります。また、論理的なグループとしてコア・グループとHAグループがあります。

コア・グループ内ではHAマネージャーとコーディネーターが稼動します。コーディネーターはコア・グループメンバーの情報(グループ名、グループメンバーやポリシーの情報)や、コア・グループメンバーの状況(停止、開始、失敗)をトラッキングしています。また、サーバー障害時に適切なアプリケーション・サーバー内のHAマネージャーにフェールオーバーの指示を出します。コーディネーターの存在するJVMプロセス障害発生に備えて、全てのJVMでコーディネーターを引き継ぐ準備がされており、Single Point of Failureではありません。コーディネーターは、アクティブになるコーディネーターの数や優先的にコーディネーターを起動するサーバーを指定することができます。またコーディネーターからの指示により、他サーバーのシングルトン・サービスを自アプリケーション・サーバーに引き継がせます。

どのサーバー・プロセスを優先コーディネーターとして指定するか決定する場合は、以下の事項を考慮してください。

－十分なJVMヒープサイズが確保できるサーバーである

コーディネーターが稼働するサーバー・プロセスは、HAグループ情報や各メンバーの状態情報を保管するため、追加のヒープ領域が必要になります。

－高負荷がかかるサーバーではない

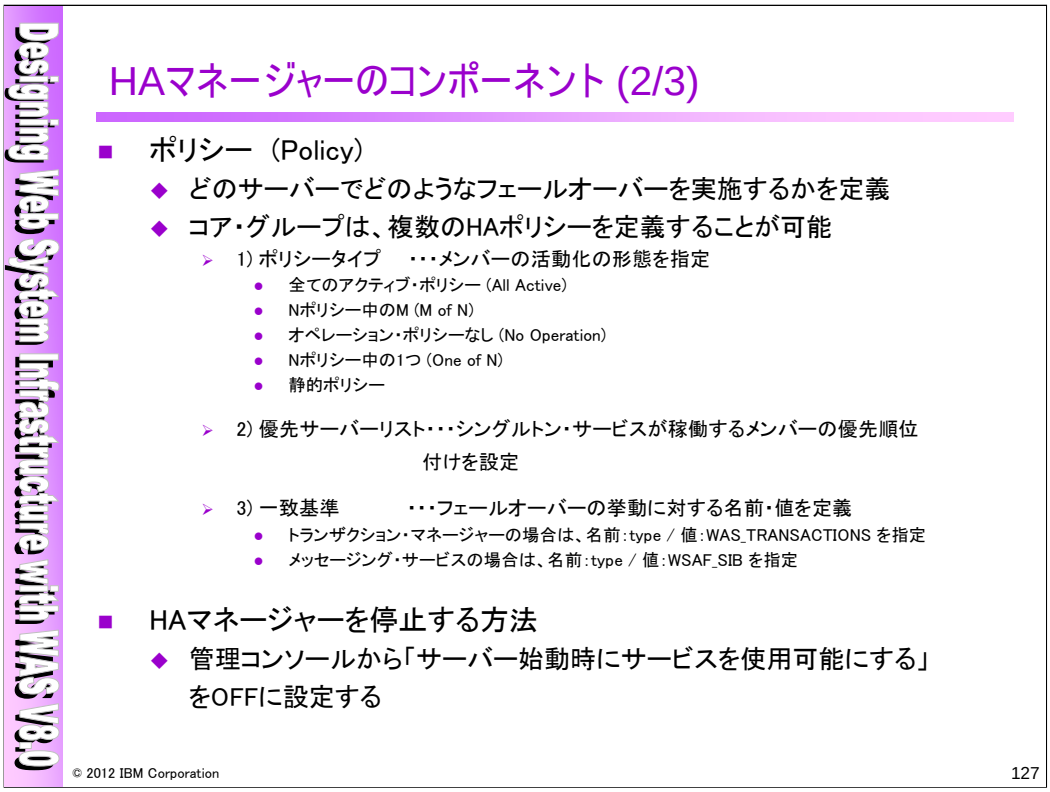
コア・グループ内のイベントに迅速に反応するために、コーディネーターはCPUリソースが必要になります。

－起動、停止が繰り返されないサーバーである

コーディネーターのフェールオーバーが発生するとコア・グループのメンバー全てで追加のCPUリソースを消費しますので、あまり頻繁に停

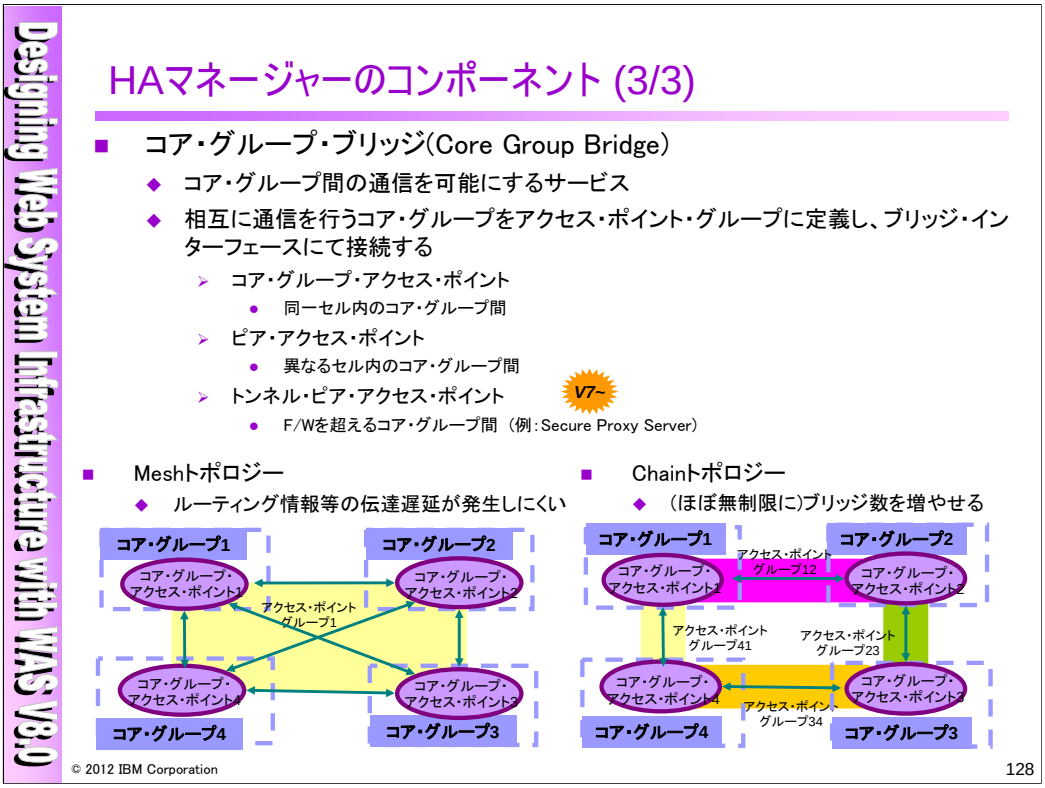
止されないサーバー上でコーディネーターを稼働させることが推奨されます。

また、コア・グループ内のサーバー(コア・グループ・メンバー)を、さらに動的にグルーピングすることができます。これを"高可用性グループ"といい、name=valueの組のセットで構成される一致基準をもとにグルーピングされます。この一致基準を使用して、サービス統合のバス内のサーバーをグルーピングします。



ポリシーとはシングルトン・サービスの引き継ぎに対して、どのサービスをどのようにフェールオーバーするかという設定です。メッセージング・エンジンのフェールオーバーの設定もポリシーで定義します。また、コア・グループは複数のポリシーを定義することが可能です。ポリシーでは、ポリシータイプ/優先サーバーリスト/一致基準を設定します。

HAマネージャーにより様々な高可用サービスが提供されますが、その反面、CPU、ヒープ・メモリー、ソケットなどのシステム・リソースを消費します。HAマネージャー関連のコンポーネント が消費するリソースの量は、コア・グループのサイズの増加とともに指数関数的に増大します。WASV6.xでの実績より、コア・グループ内に50個以上のプロセスが存在する場合は、HAマネージャーを停止する、もしくはコア・グループを分割することを検討してください。コア・グループを分割した場合にはコア・グループ・ブリッジ(後述)を使用することでコア・グループ間の通信が可能です。デフォルトでは、1つのセルには、1つのデフォルト・コア・グループが作成され、すべてのプロセスがこのグ ループのメンバーとなります。

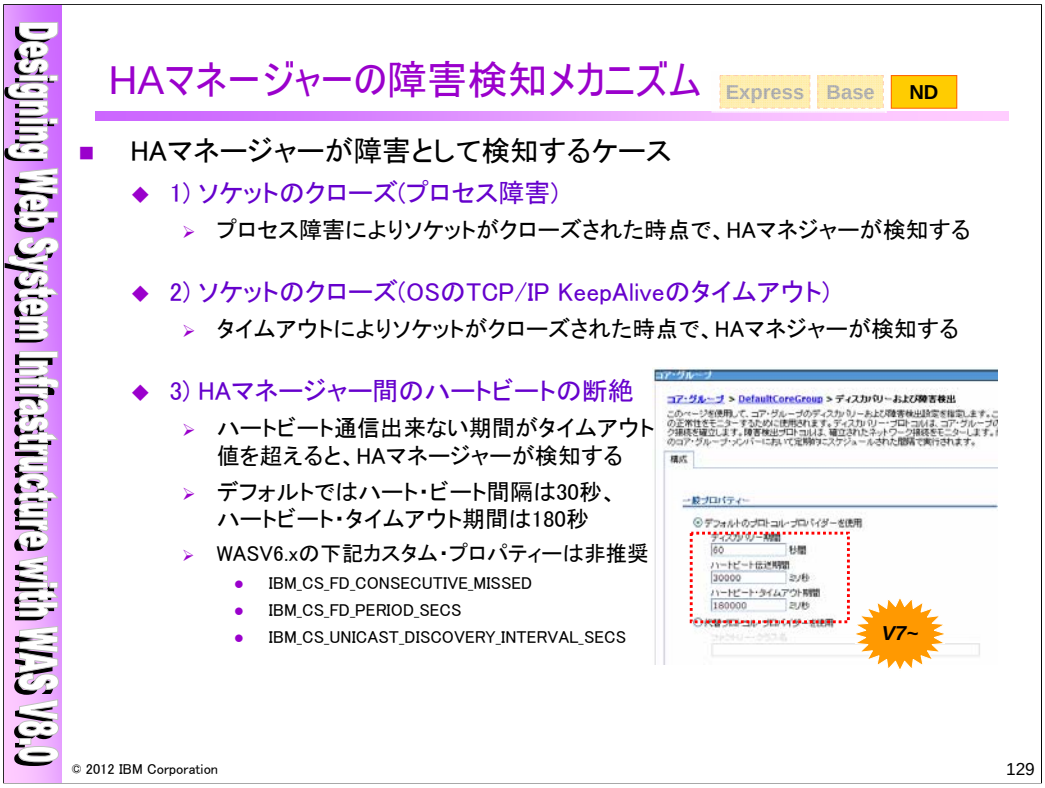


コア・グループ間の通信を可能にするサービスがコア・グループ・ブリッジです。そして、複数コア・グループ間の通信にはアクセス・ポイント・グループを定義し、ブリッジ・インターフェースと呼ばれる機能にてお互いを接続します。WASV7からは、アクセス・ポイントとして、F/Wを超えるコア・グループ間を接続できる"トンネル・ピア・アクセス・ポイント"が追加されました。

ブリッジ・インターフェースはコーディネーターとは異なり、明確に指定する必要があります。可用性を考慮すると各コア・グループ毎に2つのブリッジ・インターフェースを指定することが推奨されます。またコーディネーターやコア・グループ・ブリッジの役割を持たせるプロセスは、可能であれば、両者の機能を持つ専用のプロセスを作成し、最大ヒープサイズを1024MBに設定することが推奨されています。専用のプロセスを用意しない場合には、ノード・エージェントに役割を割り当てて下さい。

＜参考:コア・グループ・ブリッジの設定が必要なケース例＞

・バージョンの異なるシステム間でHAマネージャーの情報を共有する
・外部セルに対するJMSリクエスト
・EJBのバックアップクラスターが外部セルに配置されている
・Proxy Serverを使用して外部セルにリクエストを割り振る
・ODRを使用して外部セルにリクエストを割り振る



HAマネージャーが障害として検知するケースは、以下の3ケースです。この中で最も早く検知したケースにてサーバーがダウンしたと判断します。

1.ソケットのクローズ(プロセス障害)

プロセス障害によりソケットがクローズされた時点で、HAマネジャーが検知します。

2.ソケットのクローズ(OSのTCP/IP KeepAliveのタイムアウト)

タイムアウトによりソケットがクローズされた時点で、HAマネジャーが検知します。

3.HAマネージャー間のハートビートが断絶した場合

ハートビート通信出来ない期間がタイムアウト値を超えると、HAマネージャーが検知します。デフォルトのハートビート・タイムアウト期間は180秒です。

WASV6.xの下記カスタムプロパティは、WASV7では非推奨になりました。WASV6.xとWASV7の混合セル環境でのみご使用下さい。

・IBM_CS_FD_CONSECUTIVE_MISSED

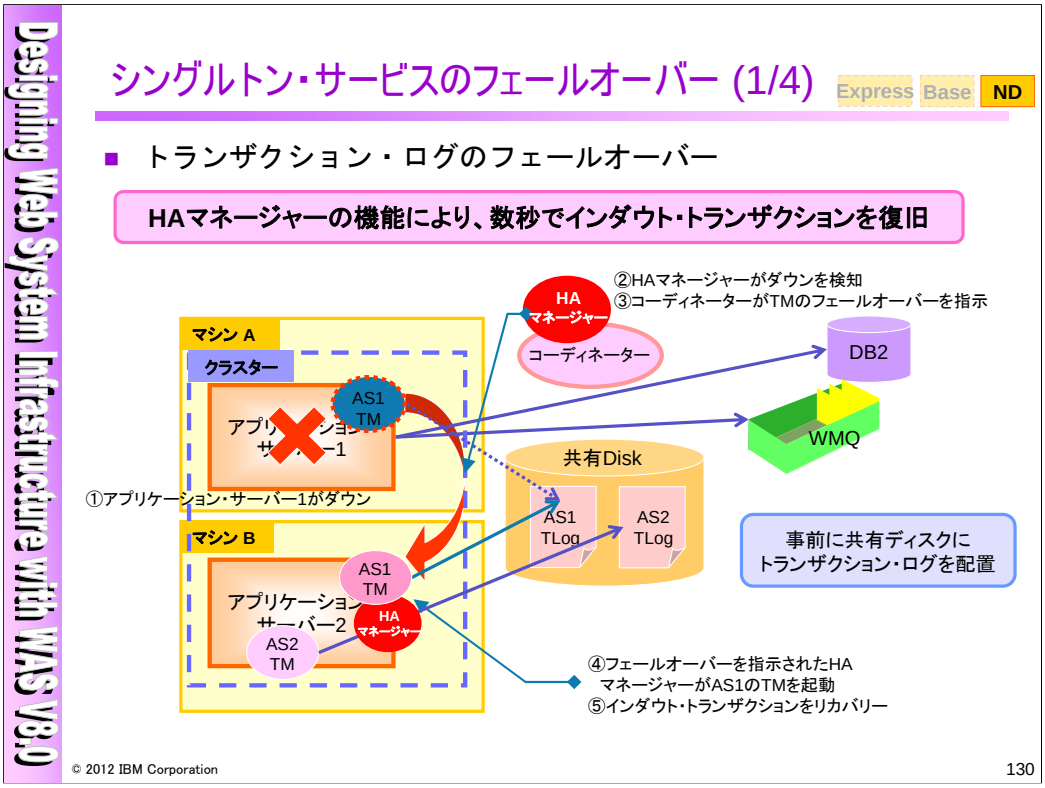
コア・グループメンバーが失敗したと判断されるハートビートの欠落数(デフォルト6回)

・IBM_CS_FD_PERIOD_SECS

コア・グループのネットワーク接続が検査される頻度(デフォルト30秒)

・IBM_CS_UNICAST_DISCOVERY_INTERVAL_SECS

ディスカバリー・プロトコルが実行される間隔(V6.0/V6.0.1:デフォルト15秒、V6.0.2以降:デフォルト30秒)



WASV8ではこれまでと同様に、事前に共有ディスクにトランザクション・ログを配置することで、障害が発生してもトランザクション・マネージャーのフェールオーバーが自動で実行され、処理を復旧する事が可能です。

HAマネージャーによるトランザクションの復旧手順は以下になります。

(1)アプリケーション・サーバー1の障害をコア・グループ内の他のサーバーのHAマネージャーが検知します。

(2)コア・グループ・コーディネーターがアプリケーション・サーバー1のトランザクション・マネージャーをアプリケーション・サーバー2に

フェールオーバーすることを決定し、アプリケーション・サーバー2のHAマネージャーにフェールオーバーを指示します。

(3)指示を受けたアプリケーション・サーバー2のHAマネージャーが通常アプリケーション・サーバー1で稼働しているトランザクション・マ

ネージャーをアプリケーション・サーバー2で起動します。

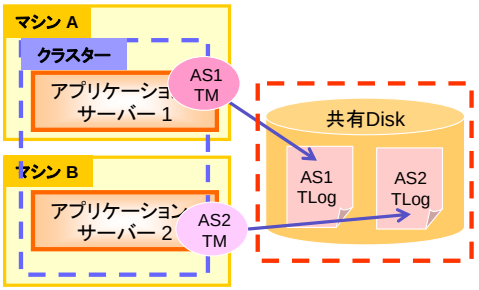
(4)アプリケーション・サーバー2で起動したアプリケーション・サーバー1のトランザクション・マネージャーがインダウト・トランザクション

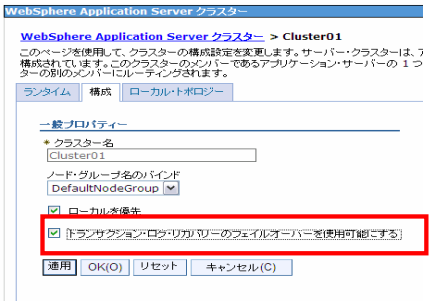
のリカバリーを行います。

Designing Web System Infrastructure with WAS V8.0

シングルトン・サービスのフェールオーバー (2/4)

- トランザクション・マネージャーのHA設定
  - 共有Diskの準備
    - ・各クラスター・メンバーから同時にアクセスできるディスクが必要
    - ・共有ディスクにトランザクション・ログを配置
  - WASの設定
    - ・アプリケーション・サーバーの設定
      トランザクション・ログ・ディレクトリーの設定変更
    - ・クラスターの設定
      トランザクション・マネージャーのフェールオーバーを使用可能にする
      (デフォルト:OFF)

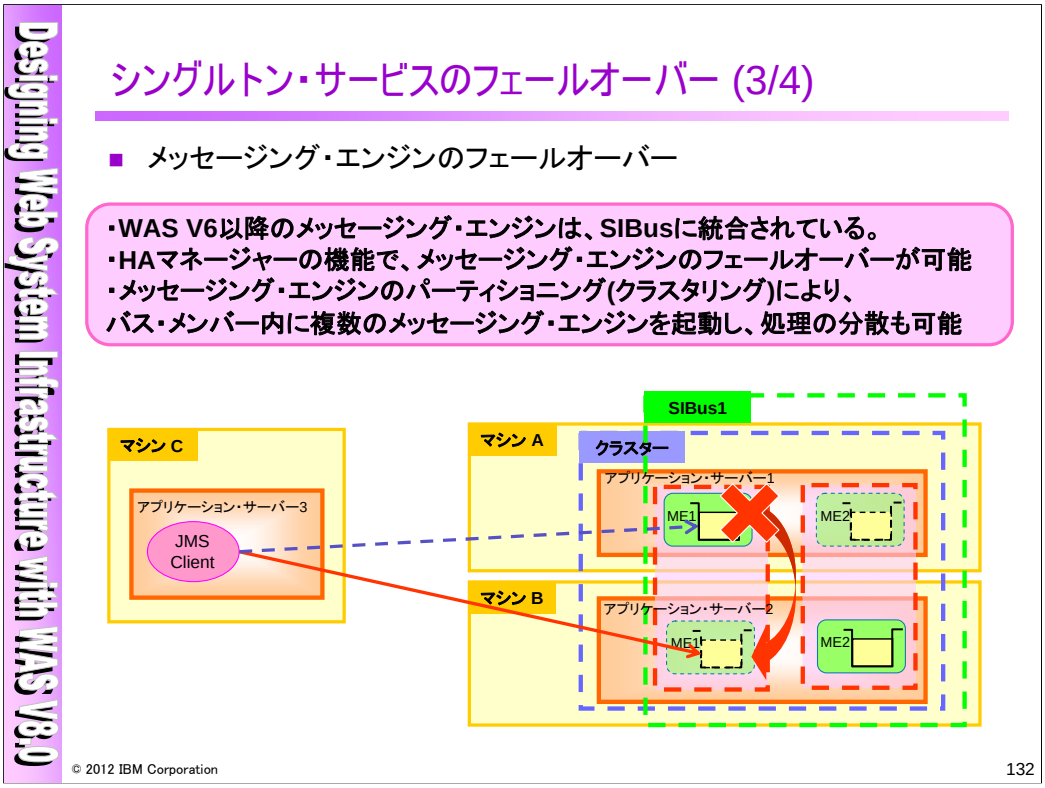




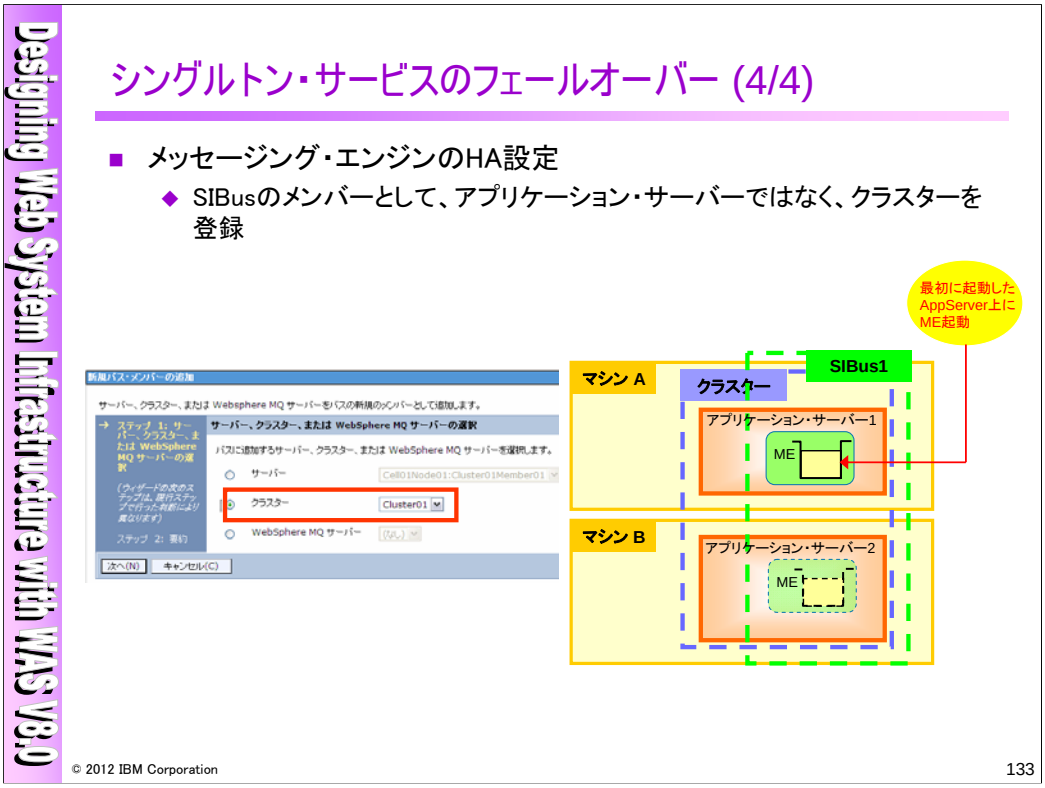
© 2012 IBM Corporation

トランザクション・マネージャーのHAを設定するには、トランザクション・ログを共有Disk上に配置する必要があります。

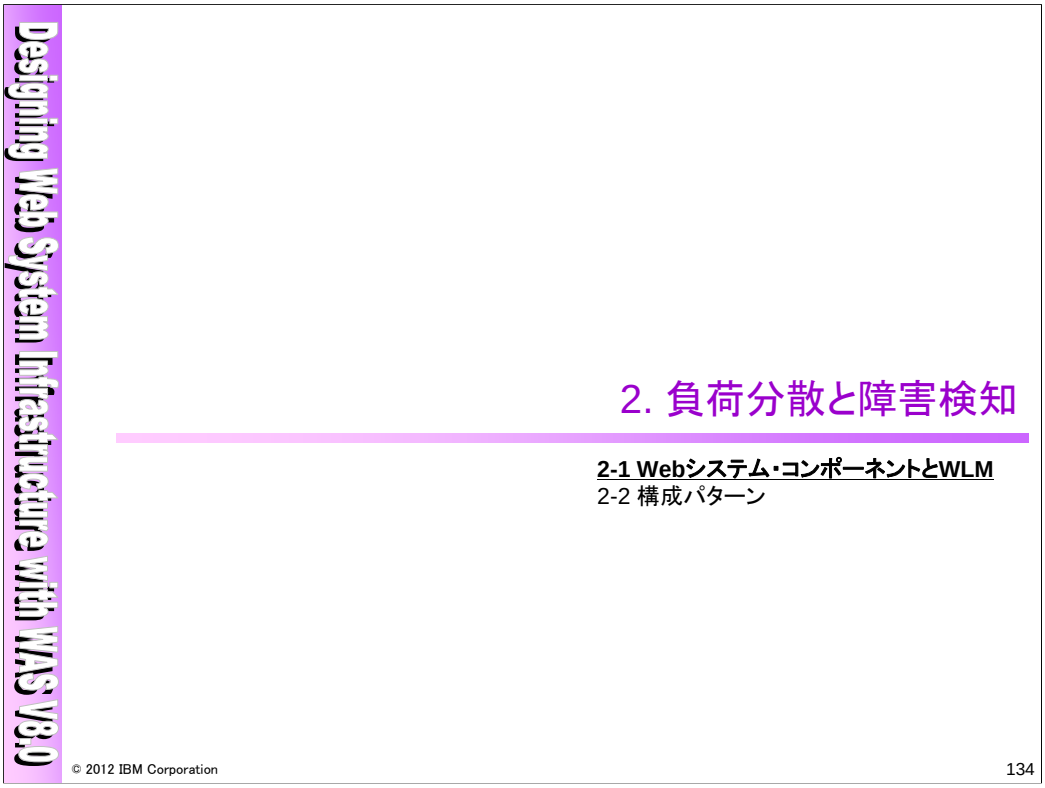
WASの設定としては、トランザクション・ログのディレクトリーの設定、およびトランザクション・マネージャーのフェールオーバーを有効にする設定が必要です。



WAS6.0以降では、SIBusに統合されたメッセージング・エンジンがメッセージング・サービスを提供します。メッセージング・エンジンはアプリケーション・サーバー上で稼働します。そのため、アプリケーション・サーバーでクラスター構成をおこなうことにより、メッセージング・エンジンはクラスター間でのフェールオーバーが可能になります。また、HAマネージャーから若干話がそれますが、メッセージング・エンジンの高可用性、高パフォーマンス機能としてパーティショニング構成を行うことにより複数稼働させることも可能です。



メッセージング・エンジンのHAの設定は、SIBusのメンバーとして、登録先をアプリケーション・サーバーではなく、クラスターに設定します。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
2-2 構成パターン
© 2012 IBM Corporation
134

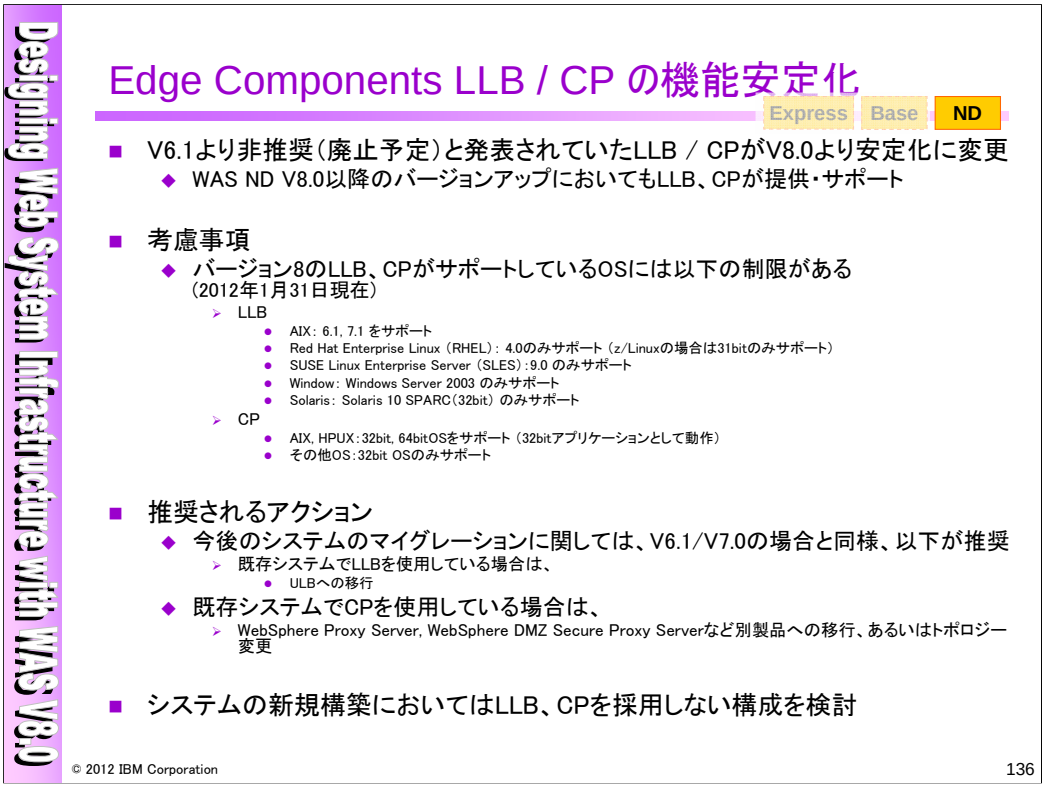


メンテナンスを行う際に利用できるLoad Balancerでの割り振り制御コマンドとしては2種類あります。

まず、新規リクエストの割り振りを停止したい場合、dscontrol manager quiesceコマンドを使用します。このコマンドを実行すると、指定したサーバーの重みが「0」となり、新規の接続が割り振られなくなります。活動中の接続についてはその終了を最後まで待ちます。なおIP Stickyを使用している場合には、Sticky時間内のクライアントからのリクエストは引き続き割り振られてしまいます。このためSticky時間が長い場合にはなかなか割り振り先サーバーへの接続が完全に終了しない可能性もあります。Stickyを使用している場合にSticky時間内のクライアントからの接続であっても、新規接続の割り振りを停止したい場合は、nowオプションを指定してdscontrol manager queisceコマンドを実行してください。

続いて、活動中の接続も含めて、全てのリクエストの割り振りを停止するコマンドとして、dscontrol server downコマンドがあります。このコマンドを実行すると、指定したサーバーの重みが「-1」となり、全てのリクエストの割り振りが停止します。

なお、dscontrol server downコマンドは"特定クラスターの特定ポートのこのサーバー"のように明示的に割り振り制御するサーバーを指定できますが、dscontrol manager queisceコマンドの場合は、サーバー名しか指定できません。このため、例えば80ポートおよび443ポートに同一サーバー名でサーバーが登録されていた場合には、dscontrol manager queisceを発行した場合、80ポート/443ポート双方のポートに対する割り振りが停止してしまいます。443ポートは割り振りを止めたいけど80ポートは稼動させたままにしたいというような要件がある場合は、物理的に同じサーバーであっても、ポートによってサーバーの定義名を変えてサーバーを登録してください。



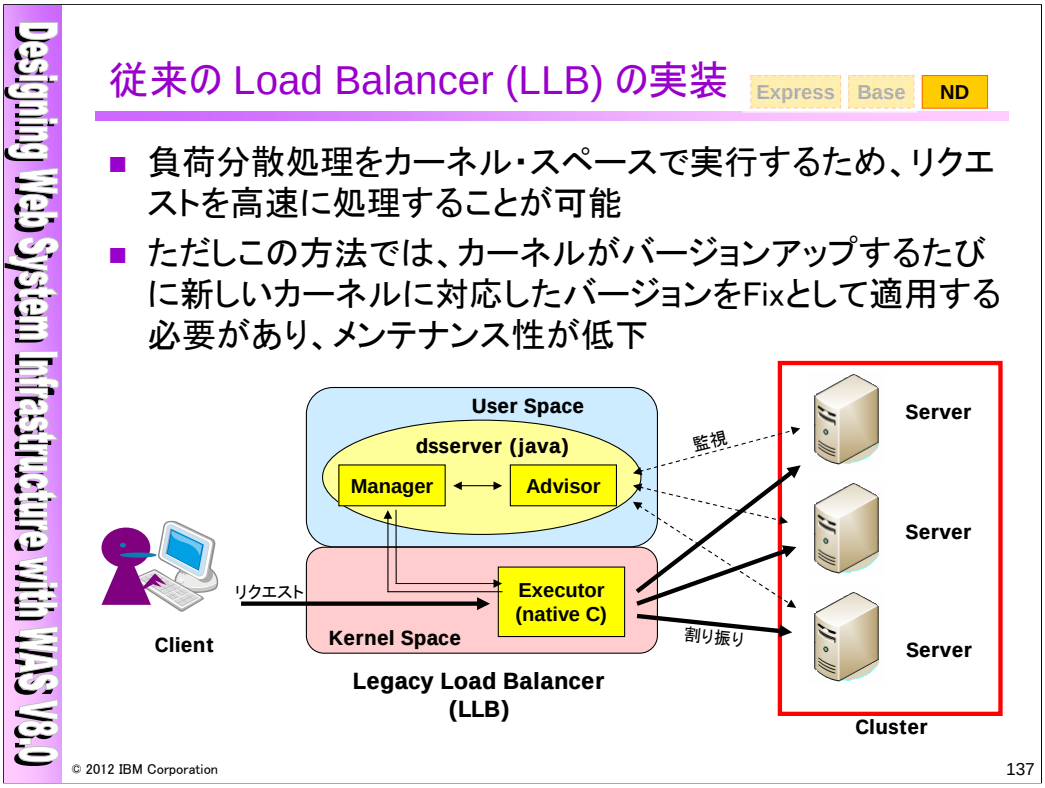
まず、LLBとCPの安定化について、ご説明いたします。

V6.1よりLLB/CPは非推奨（廃止予定）となり、当初はV8.0をもってLLBとCPの提供は終了予定でしたが、今後製品として機能拡張は行われない代わりに、今後のバージョンアップにおいても製品が提供・サポートされ続ける予定へと変更になりました。

Information Center －「安定化されたフィーチャー」

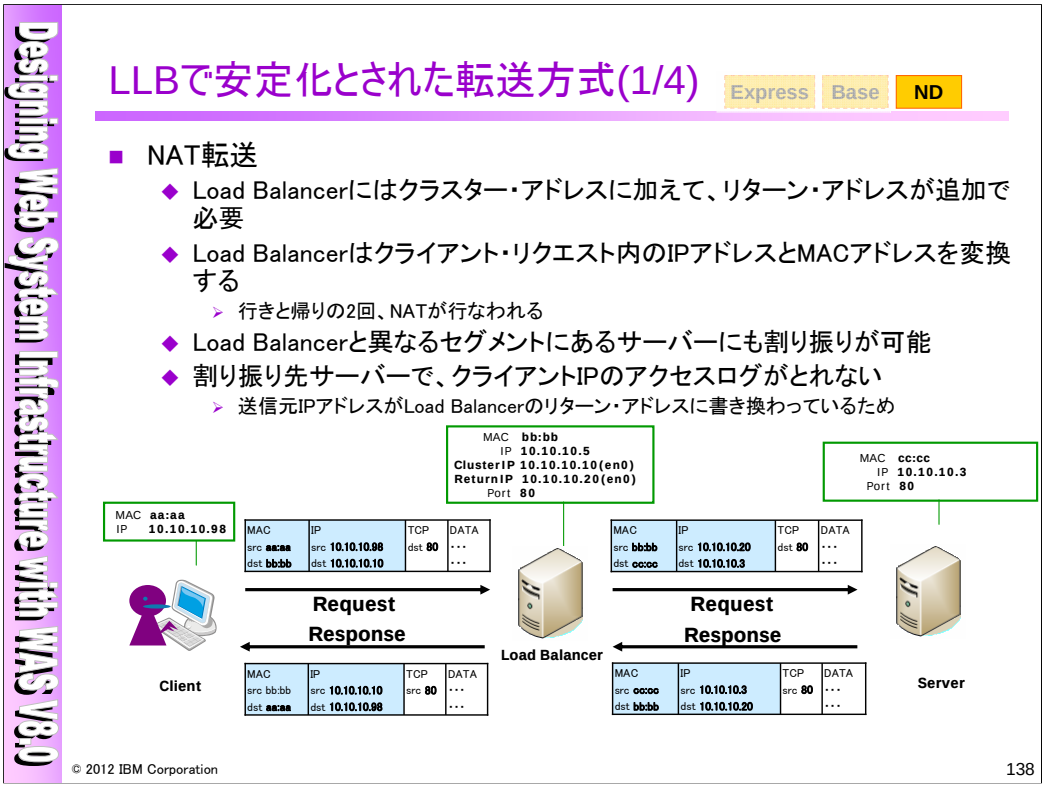
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rmig_stabfeat.html

ただし、V8.0でのLLB、CPがサポートしているOSには制限があります。今後はLLB/CPを使用せず、ULBを使用する構成をとることが推奨されます。



新しいLoad Balancerについて説明する前に、従来のLoad Balancerの実装について説明します。従来のLoad Balancerでは、Executerはカーネル・スペースで動作します。カーネル上で動作しているため、Executorはプロセスとして確認することができません。カーネル上で動作することにより、リクエストを高速に処理することができましたが、この方法では、カーネルがバージョンアップするたびに新しいカーネルに対応したバージョンをFixとして適用する必要があり、メンテナンス性が低下するという問題がありました。

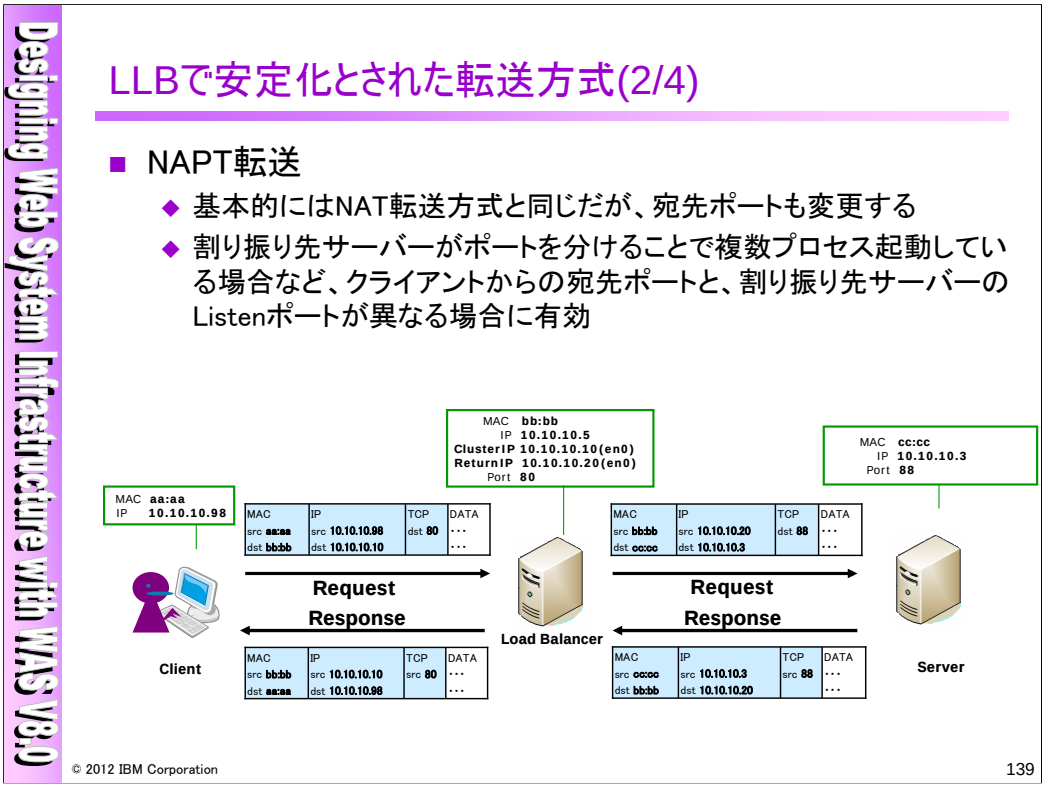
ManagerとAdvisorについては、dsserverという一つのjavaプロセス上で動作します。管理者はdscontrolコマンドまたはGUIを使用し、このdsserverに対して操作を行います。



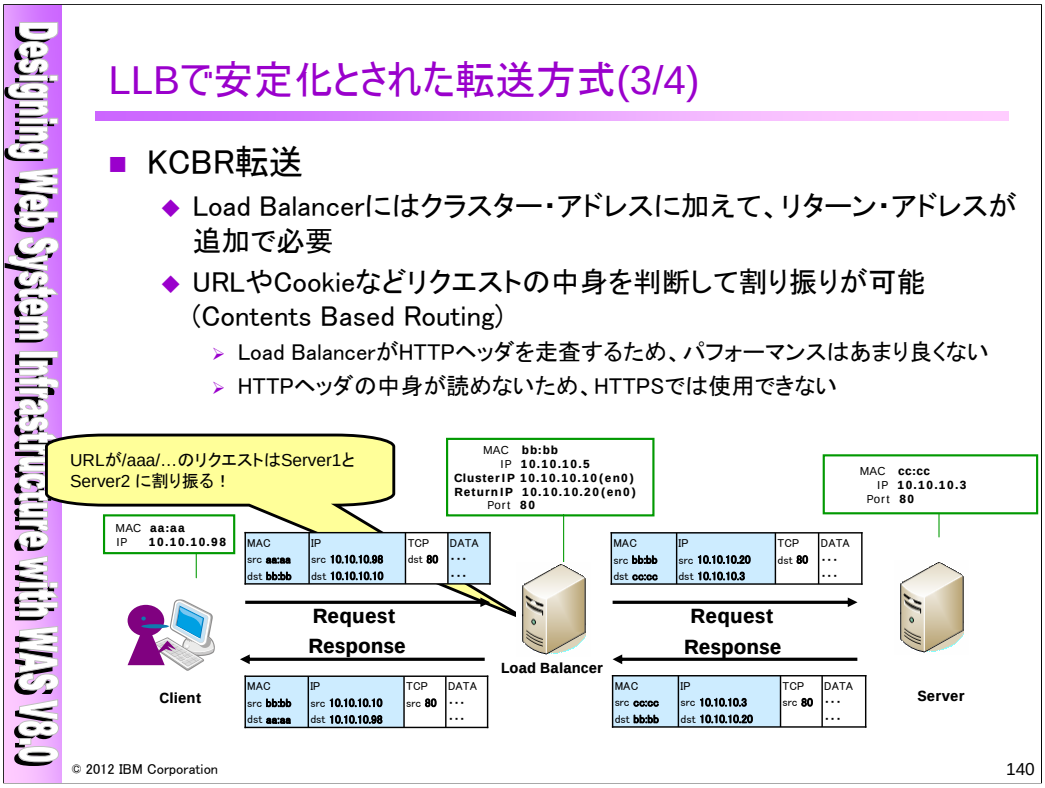
NAT転送方式では、クライアントがアクセスするクラスター・アドレスに加え、割り振り先サーバーへのリクエスト送信に使われるリターン・アドレスが必要になります。

この転送方式では、Load Balancerで行きと帰りの2回NAT(IPアドレスの変換)が発生します。IPアドレスの書換えまで行なっていますので、Load Balancerと別ネットワークに存在するサーバーに対しても割り振りを行なうことができます。

具体的なパケット処理の流れは次のとおりです。クラスター・アドレスに対して飛んできたリクエストを受け取ったLoad Balancerは、送信元IPアドレスをリターン・アドレス、宛先アドレスを割り振り先サーバーのアドレスに書き変えてリクエストを転送します。リクエストを受け取った割り振り先サーバーは、処理を行い、リクエストの送信元IPアドレスに対してレスポンスを返します。これによりLoad Balancer上に定義されたリターン・アドレスにレスポンスが返されます。さらにLoad Balancerはこのレスポンスの宛先をクライアントのIPアドレスに書き換えてクライアントに転送します。この際、送信元IPアドレスはクラスター・アドレスに書換えられますので、クライアントに割り振り先サーバーのIPが漏れることはありません。



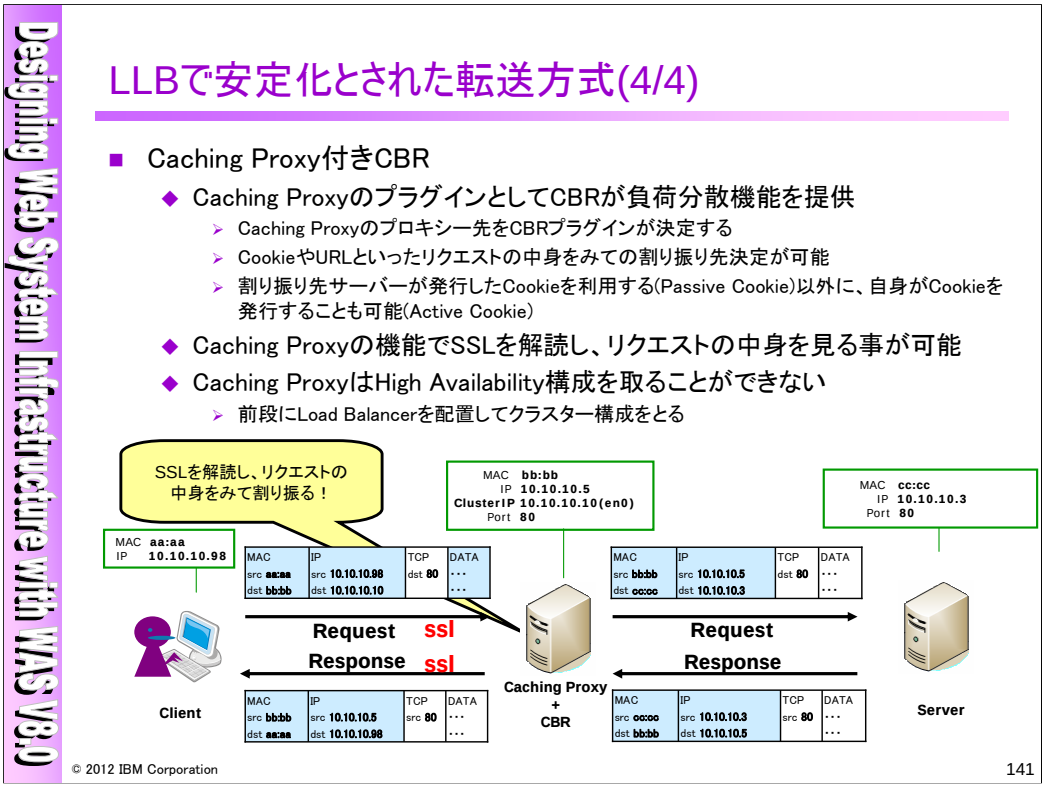
NAPT転送は基本的にはNAT転送方式と同じですが、IPに加えポートまで変換する点がNAT転送方式と異なります。これはクライアントがLoad Balancerに対して送るリクエストの宛先ポートと割振り先サーバーのListenしているポートが異なる場合に有効です。



次にLoad BalancerのDispatcherによるCBR転送方式です。CBRプラグインによるCBR転送と区別するため、Kernel CBR(KCBR)転送と呼びます。クラスター・アドレスに加えて、リターン・アドレスが必要になる点や、IPアドレスまでの書き換えが行なわれる点はNAT転送方式と同様ですが、KCBR転送方式はContent Based Routingの名前の通り、コンテンツの中身を判断してリクエストの割り振りを行なえる点が大きく異なります。KCBR転送方式ではコンテンツの中身を見ることができますので、Cookieの中身を判断してのセッション・アフィニティやURLを見ての割り振りといったことが可能です。

しかし、SSLの複合化を行なうことができないため、SSL通信の場合はコンテンツの中身を見て割り振りを行なうことはできません。SSL通信を行なっていて、かつCookieの中身を見て割り振りを行いたいといった要件の場合には、次に紹介するCBRコンポーネントを用いる必要があります。

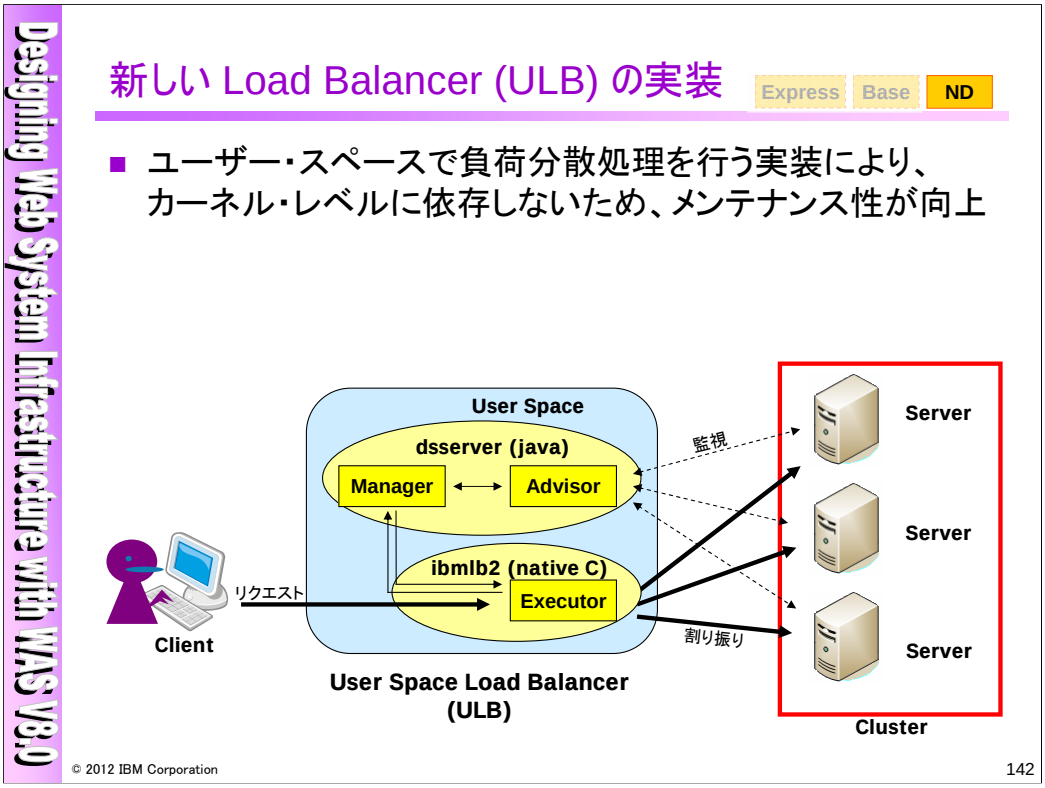
なおNAT転送方式とKCBR転送方式では、IPアドレスが変換される点だけを考えると同様のものに見えますが、厳密には違いがあります。NAT転送方式ではLoad Balancer受け取ったパケットを即座に転送するため、コネクションはクライアントと割り振り先サーバー間で張られているような動作をします。一方、KCBR転送方式ではHTTPヘッダを解析するためにパケットをバッファする必要があり、そのため、クライアント-Load Balancer間とLoad Balancer-割り振り先サーバー間で別々の2つのコネクションが張られているような動作となります。



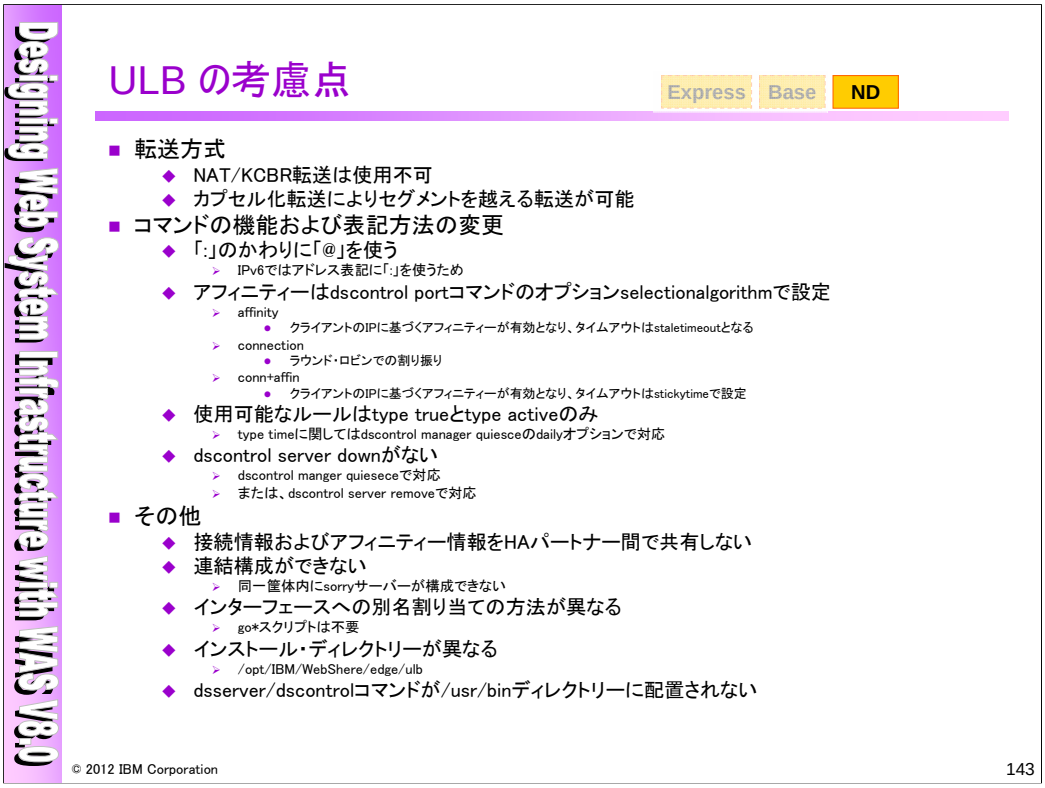
次にCBR転送方式です。CBRはCaching Proxyのプラグインとして働き、割り振り先を決定する役割を果たします。実際に割り振りを行うのはCaching Proxyです。このため、CBR転送方式は、Caching Proxy付きCBRと呼ばれる事があります。

KCBR転送と異なる点として、Caching Proxy付きCBRではSSLを複合化する事ができます。このため、HTTPSがある環境でも、リクエストの中身を解読して割り振りを行うことができます。CBRからの情報を参照して、割り振りを行います。

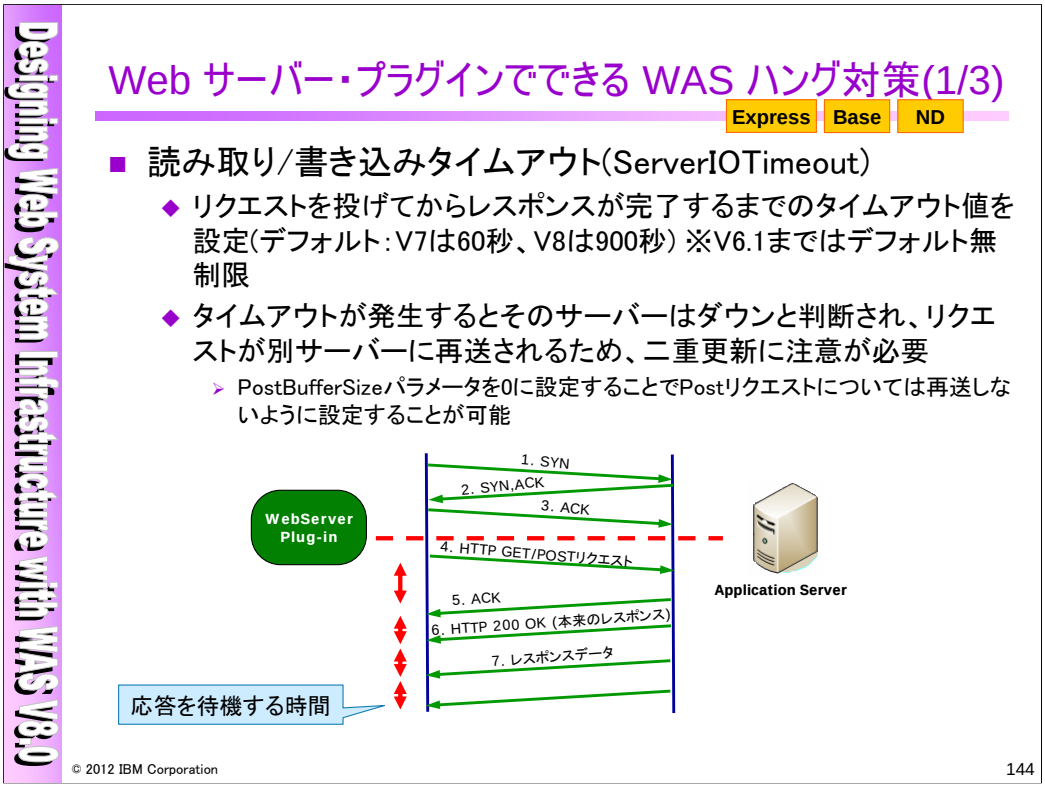
なお、Caching Proxy付きCBRはHigh Availability構成を取ることが出来ないため、可用性を確保するためには前段にLoad Balancerを配置して負荷分散を行う必要があります。



新しい実装では、Executorもユーザー・スペースで動作するようになりました。これにより、カーネル・レベルに依存することがなくなり、メンテナンス性が向上しました。Executorはibmlb2という一つのプロセスとして確認することができます。ManagerとAdvisorについては従来のLoad Balancerと同様にdsserverというjavaプロセス上で動作します。新しい実装のLoad BalancerをUser Space Load Balancer (ULB)、従来のLoad BalancerをLegacy Load Balancer (LLB)またはKernel Load Balancerと呼ぶことがあります。



ULBとLLBでは機能に差があるため注意が必要です。ここでは特に注意が必要な点についてまとめました。



WASのハングを検知するための対策の一つ目として、読み取り/書き込みタイムアウトがあります。

このパラメーターを指定すると、HTTPリクエストを投げてからレスポンスを受け取るまでのタイムアウト値を指定できます。これによりWASがハングしていてもタイムアウトにより障害と判断することが可能です。但し、実際の処理に時間がかかるようなアプリケーションでは、その時間を考慮してタイムアウト値を設定する必要があります。読み取り/書き込みタイムアウトのデフォルト値はV6.1では無制限でしたが、V7.0では60秒になりました。

読み取り/書き込みタイムアウトにより障害が検知された場合、リクエストは他のサーバーへリトライされます。プロセス障害や接続タイムアウトの場合には、TCPセッション確立の段階での障害検知であり、未だリクエストを送信していませんでしたので、リクエストをリトライすることには何も問題はありませんでしたが、読み取り/書き込みタイムアウトの場合には既にリクエストを送信した後であるため、更新リクエストであれば、二重更新となってしまわないように注意する必要があります。

プラグインのPostBufferSizeパラメータを0に設定することで、Postリクエストに関してはリトライさせないことができますが、Getリクエストに関してはリトライを停止することはできません。詳細は下記のテクニカルフラッシュの記載を参照下さい。

「WAS NDにおいてWebサーバ・プラグインでWLMを実施する場合の注意点 」

http://www.ibm.com/jp/domino01/mkt/cnpages1.nsf/page/default-000867DE

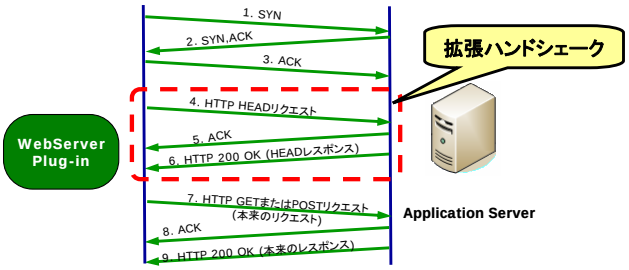
「【障害情報】POSTリクエストを再送させない設定時、プラグインがタイムアウト発生時にダウンと検知しない問題の解決策 (WAS-09-006)」

http://www.ibm.com/jp/domino01/mkt/cnpages1.nsf/page/default-00083294

Designing Web System Infrastructure with WAS V8.0

Web サーバー・プラグインでできる WAS ハング対策(2/3)

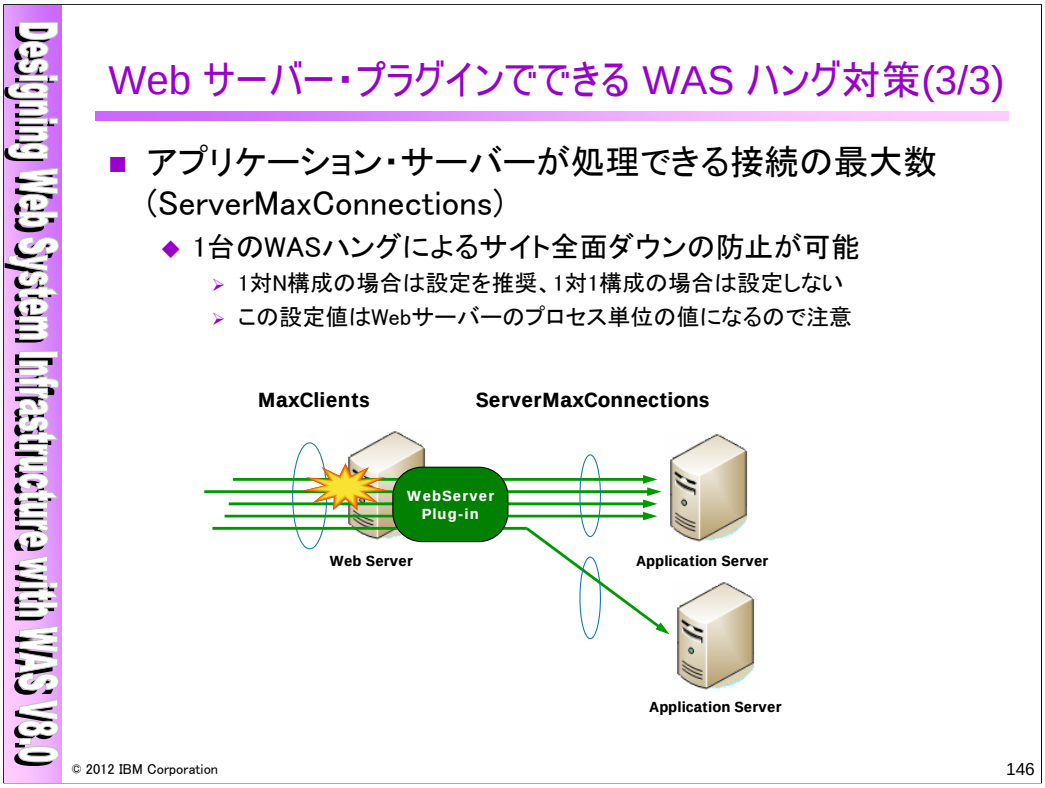
- 拡張ハンドシェーク(ExtendedHandshake)
  - 実リクエストの送信前にHEADリクエストを送信(デフォルト:OFF)
    - Webコンテナーがレスポンスを返せない場合には検知可能
  - パフォーマンスに若干のデメリット
  - リクエストを送信する前に障害検知が可能であり、リクエストの再割り振り、復旧の検知はプロセス障害の場合と同様



© 2012 IBM Corporation

Web サーバー・プラグインの拡張ハンドシェークを有効にすると、実リクエストを投げる前にHEADリクエストを投げて、障害検知を行います。ただし、実リクエスト毎にHEADリクエストを投げることになるため、パフォーマンスに若干影響を与えます。

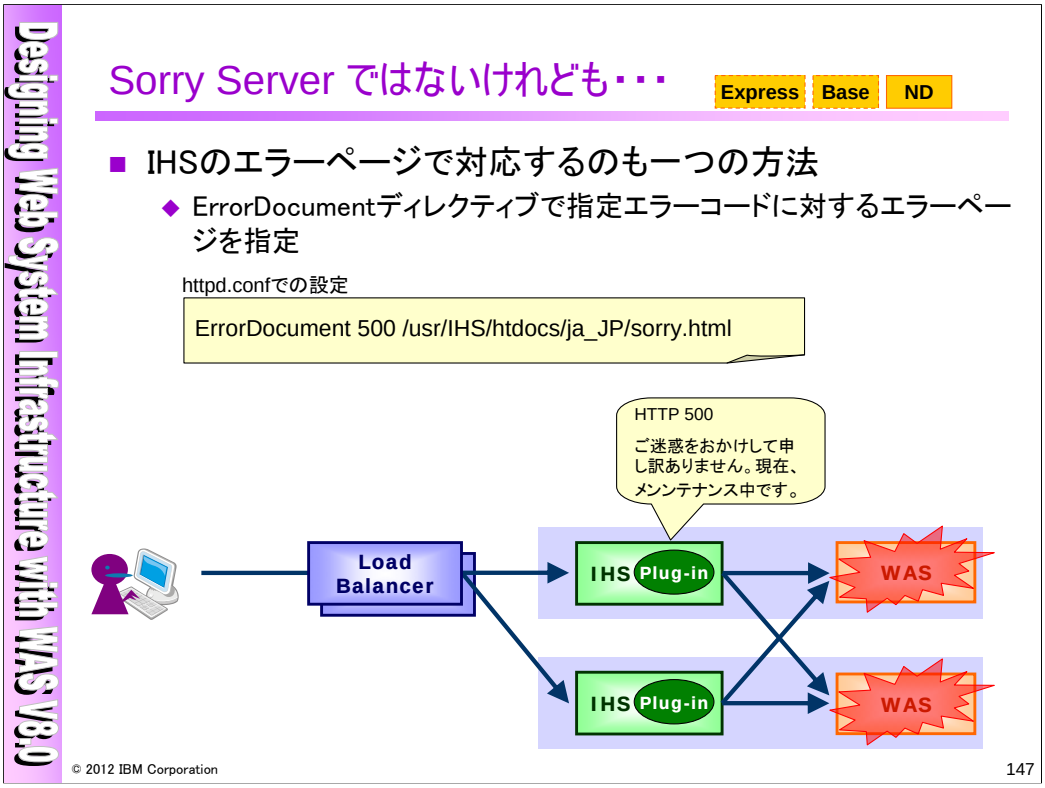
拡張ハンドシェイクを有効にした場合も、TCPセッションのタイムアウト値はConnectTimeoutで、HTTPリクエストタイムアウト値はServerIOTimeoutです。HEADリクエストに応答がない場合、ServerIOTimeoutの設定がされていないとOSレベルのTCP/IPタイムアウトまで応答を待ち続けることになり、有効なハング検知になりませんので、あわせてServerIOTimeoutも設定するようにしてください。



WASがハングした場合、ハングしたWAS上に接続が滞留します。ハングが検知できていない場合はリクエストが割り振られ続け、ハングしたWASへの接続数がIHSのMaxClients値に達し、IHSがダウンと判断されてしまう可能性があります。IHSが複数ある場合でも、同様の現象で次々にIHSがダウンとなり、最終的にシステムが全面ダウンとなってしまう危険性があります。Webサーバー・プラグインの「接続の最大数」を設定することで、WASとの接続数に制限を設定する事が可能となり全面ダウンを防ぐことが出来ます。

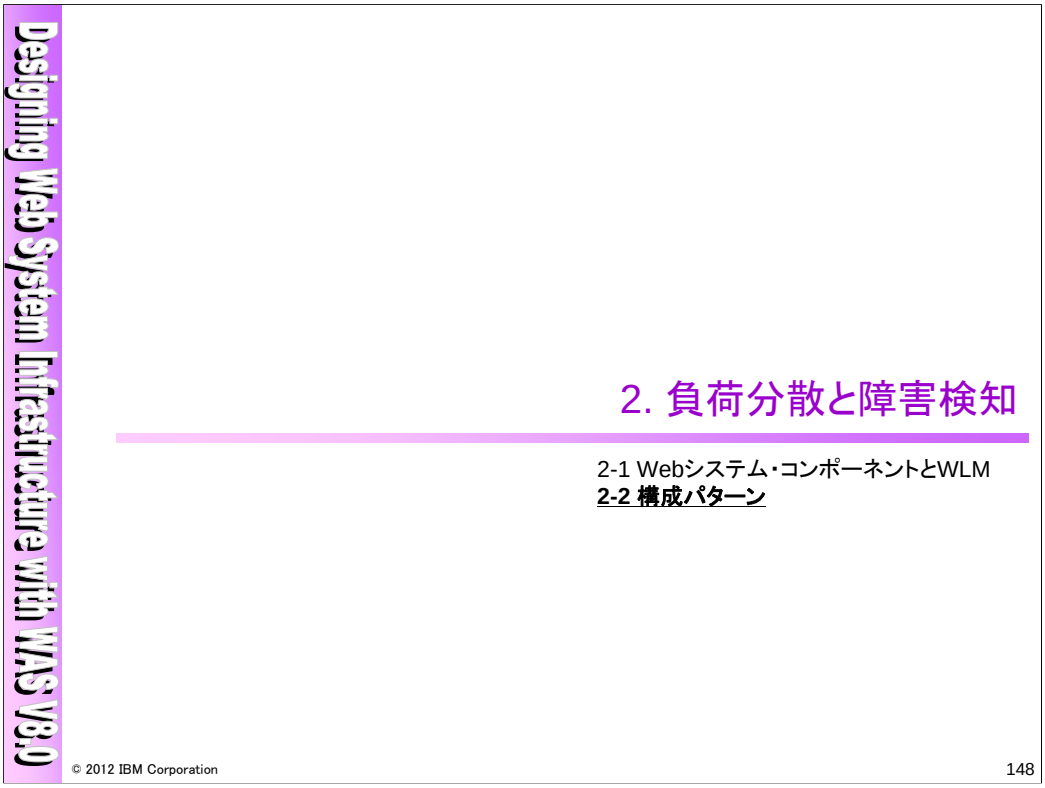
※プラグインの各設定値の詳細は、InfoCenter「Web サーバー・プラグイン用の Application Server プロパティー設定」の記載を参照下さい。

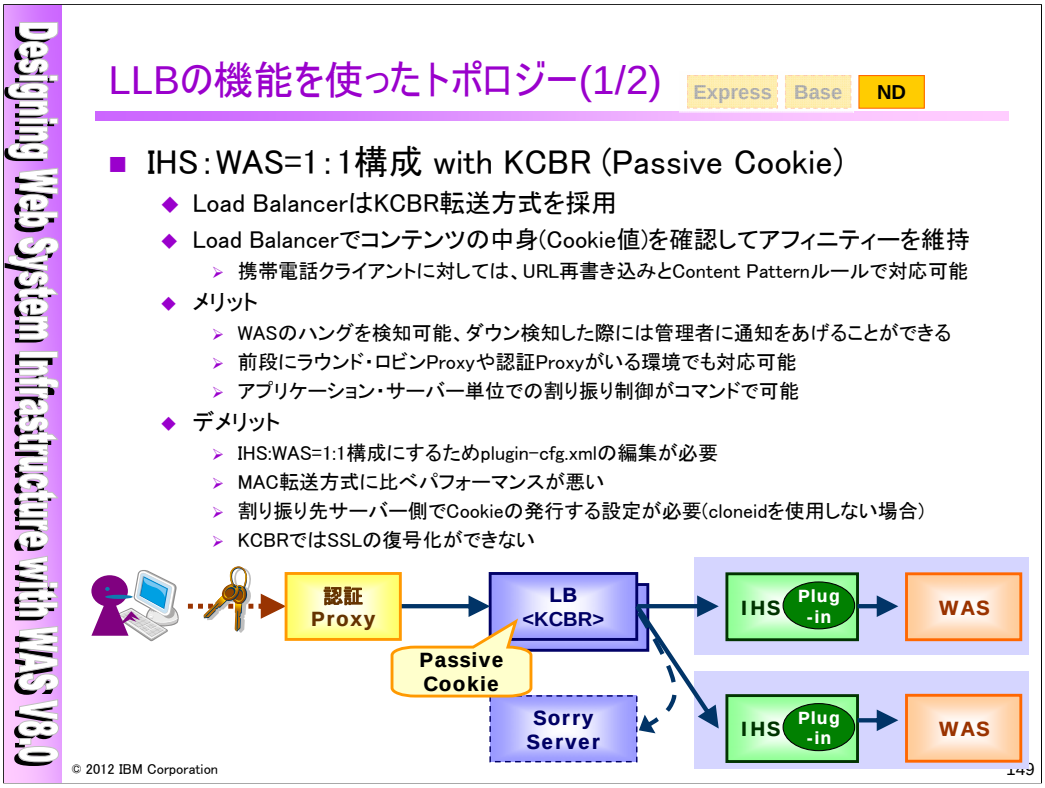
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/uwsv_plugin_props5.html



Sorry Serverではありませんが、sorryページを返す方法として、ErrorDocumentディレクティブを使う方法もあります。WASがダウンしている場合、IHSは500エラーを返しますので、ErrorDocumentディレクティブを使ってカスタマイズしたエラーページを表示させます。

この構成はIHS:WASが1:1となっている構成で、WASが全滅時にエラーを表示するための方法として有効です。


Designing Web System Infrastructure with WAS V8.0
2. 負荷分散と障害検知
2-1 Webシステム・コンポーネントとWLM
2-2 構成パターン
© 2012 IBM Corporation
148

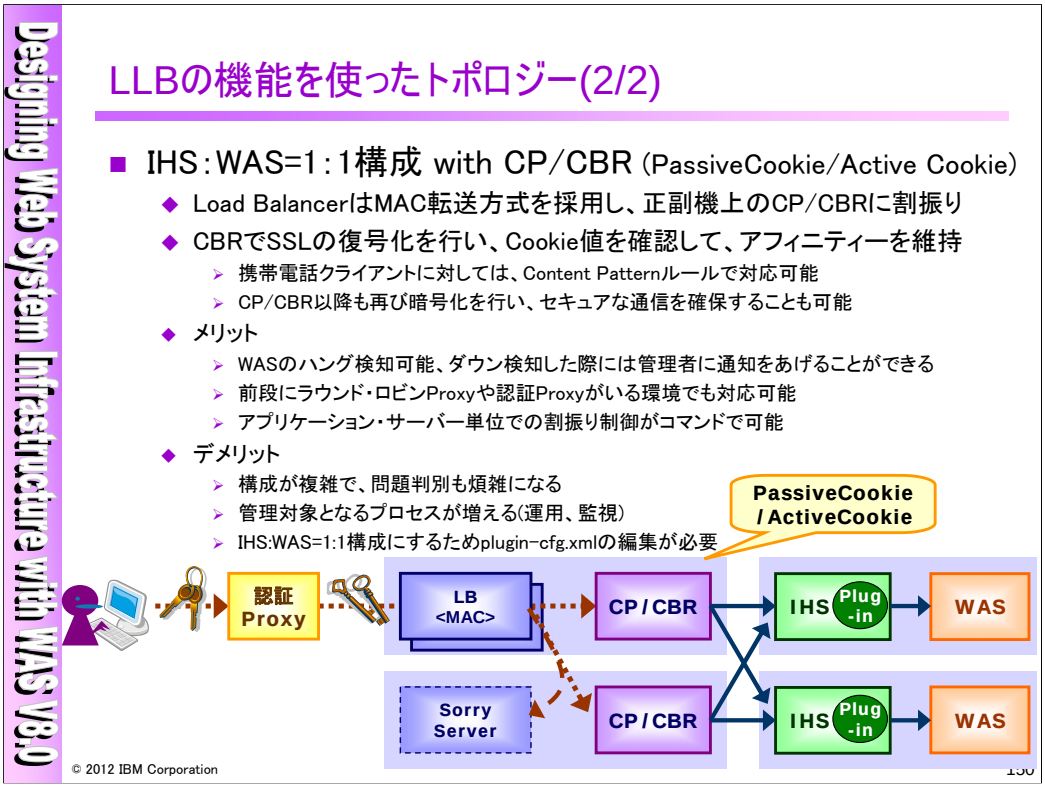


前段にプロキシーが存在する場合にも、IHS:WASを1:1に構成するためにはKCBR転送が必要になります。

この構成ではIHS:WASを1:1構成にし、Load BalancerでWASまでの障害検知を行いますが、セッション・アフィニティの維持方法としてPassiveCookieを用います。Cookieを用いることにより、たとえLoad BalancerからクライアントのIPアドレスが1つしか見えなくても負荷の偏りなく負荷分散が可能になります。コンテンツの中身(Cookie)を見て割り振る必要があるため、処理がシンプルであるMAC転送方式などと比べ、パフォーマンスの限界性能が低くなります。Passive Cookieは動的な文字列を含むCookieにも対応していますので、JSESSIONIDに含まれるcloneidを見て割り振りを行うこともできますが、JSESSIONIDを使用せずIHSやアプリケーションで独自に発行したCookieを使用する場合には、その分の設定が必要となります。

また、Load BalancerではSSLの復号化ができませんので、SSL通信の場合にはCookieを見て割り振りを行うことができません。

なお、携帯電話のクライアントに対しては、WAS側でCookieを発行する代わりにURL再書き込みを用い、Load BalancerでPassive Cookieの代わりにContent Patternルールを用いることで、このトポロジーで対応可能です。

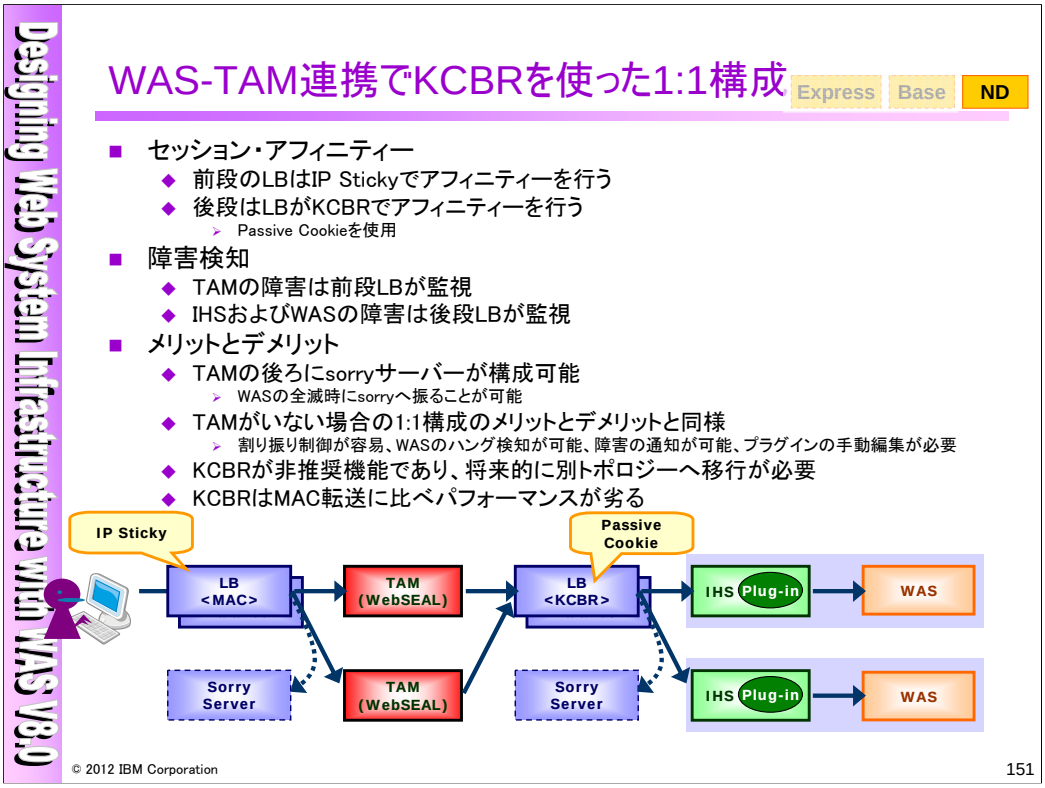


KCBRではSSL通信の際にCookieを用いてセッションの維持を行えないという制限を克服したのが、このトポロジーとなります。

この構成では、SSL通信を解除しCookieを見てセッション・アフィニティの維持を行うためCP付きCBRを採用しています。CBRコンポーネントはHA構成がとれないため、さらに前段にLoad Balancer(MAC転送方式)を配置し、複数のCP/CBRを配置することで可用性を確保しています。(なおLoad BalancerとCP/CBRは同一筐体上に配置することができるため、筐体数は増やす必要はありません。)

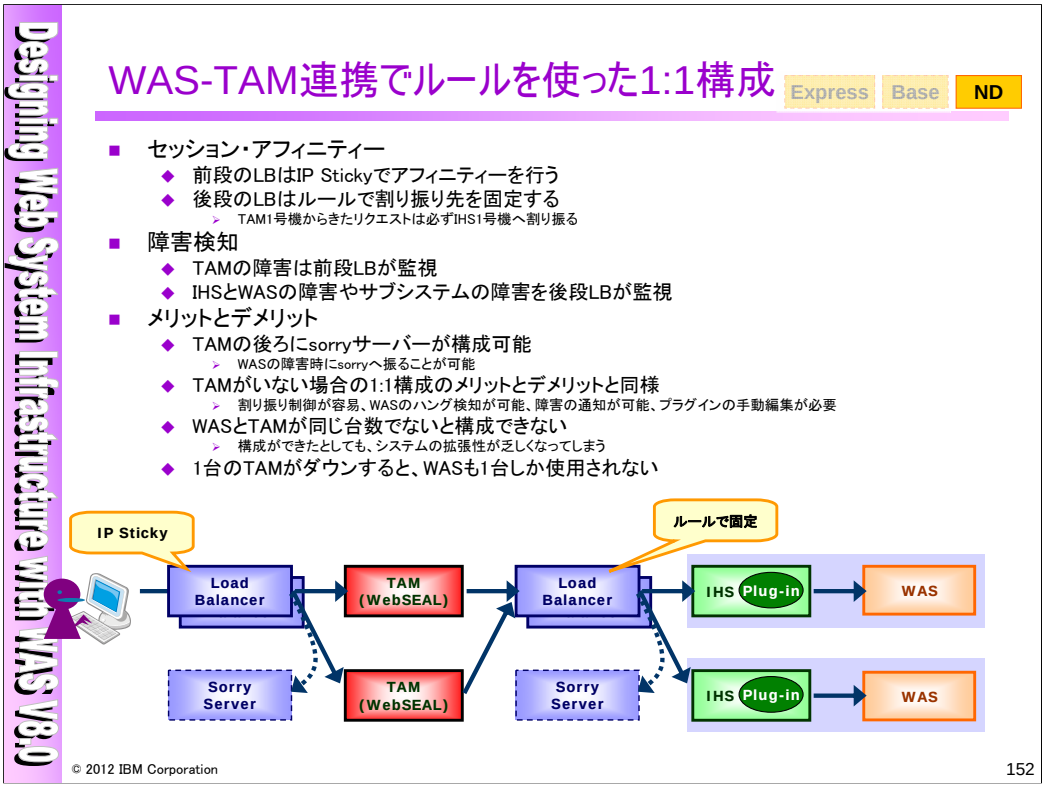
この構成のメリットはこれまでと同様、IHS:WAS＝1:1構成のメリットは享受できますし、前段にProxyが存在する場合や携帯電話クライアントの場合も、それぞれCookieやURL再書き込みを利用してアフィニティの維持が行えます。

デメリットとしては見ての通り、構成が複雑になる上、管理対象となるプロセスが増えます。ですので、初期のシステム構築負荷も高くなりますし、運用負荷も高くなります。また問題判別も複雑になってしまいます。



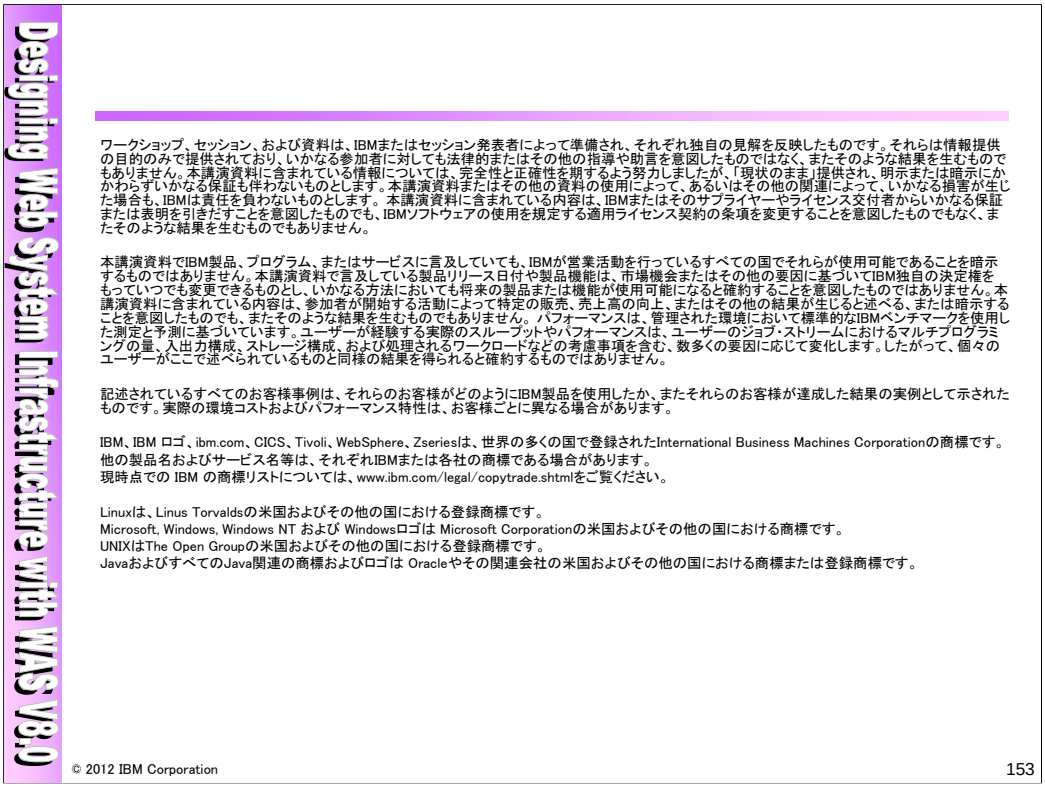
TAMから後ろを1:1構成としたい場合の構成方法として、非推奨機能であるKCBR転送を使う方法があります。

この構成ではWASのJSESSIONIDやアプリケーションの発行するCookieを利用して、Load Balancerがアフィニティーを行います。Load BalancerでWASまでの障害の検知を行えるなど、1:1構成の多くのメリットを得ることができますが、KCBRが非推奨機能であるために、将来的には別トポロジーへの移行が必要になります。



TAMから後ろを1:1構成としたい場合の構成方法として、ルールを使う方法があります。この構成では、TAM1号機からのリクエストを必ずIHS1号機へ、TAM2号機からのリクエストをTAM2号機へ割り振るように構成します。

1:1構成にはLoad BalancerでWASまでの障害の検知を行えたり、割り振りの制御が細やかにできるといったメリットがあります。


Designing Web System Infrastructure with WAS V8.0
ワークショップ、セッション、および資料は、IBMまたはセッション発表者によって準備され、それぞれ独自の見解を反映したものです。それらは情報提供の目的のみで提供されており、いかなる参加者に対しても法律的またはその他の指導や助言を意図したものではなく、またそのような結果を生むものでもありません。本講演資料に含まれている情報については、完全性と正確性を期するよう努力しましたが、「現状のまま」提供され、明示または暗示にかかわらずいかなる保証も伴わないものとします。本講演資料またはその他の資料の使用によって、あるいはその他の関連によって、いかなる損害が生じた場合も、IBMは責任を負わないものとします。本講演資料に含まれている内容は、IBMまたはそのサプライヤーやライセンス交付者からいかなる保証または表明を引きだすことを意図したものでも、IBMソフトウェアの使用を規定する適用ライセンス契約の条項を変更することを意図したものでもなく、またそのような結果を生むものでもありません。
本講演資料でIBM製品、プログラム、またはサービスに言及していても、IBMが営業活動を行っているすべての国でそれらが使用可能であることを暗示するものではありません。本講演資料で言及している製品リリース日付や製品機能は、市場機会またはその他の要因に基づいてIBM独自の決定権をもっていつでも変更できるものとし、いかなる方法においても将来の製品または機能が使用可能になると確約することを意図したものではありません。本講演資料に含まれている内容は、参加者が開始する活動によって特定の販売、売上高の向上、またはその他の結果が生じると述べる、または暗示することを意図したものでも、またそのような結果を生むものでもありません。パフォーマンスは、管理された環境において標準的なIBMベンチマークを使用した測定と予測に基づいています。ユーザーが経験する実際のスループットやパフォーマンスは、ユーザーのジョブ・ストリームにおけるマルチプログラミングの量、入出力構成、ストレージ構成、および処理されるワークロードなどの考慮事項を含む、数多くの要因に応じて変化します。したがって、個々のユーザーがここで述べられているものと同様の結果を得られると確約するものではありません。
記述されているすべてのお客様事例は、それらのお客様がどのようにIBM製品を使用したか、またそれらのお客様が達成した結果の実例として示されたものです。実際の環境コストおよびパフォーマンス特性は、お客様ごとに異なる場合があります。
IBM、IBM ロゴ、ibm.com、CICS、Tivoli、WebSphere、Zseries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。
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。
Microsoft, Windows, Windows NT および Windowsロゴは Microsoft Corporationの米国およびその他の国における商標です。
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。
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。
© 2012 IBM Corporation
153