3 ワークロード管理、高可用構成

この章では、HTTP リクエスト、EJB リクエストを複数のアプリケーション・サーバーに負荷分散するワークロード管理 (WorkLoad Management 以下、WLM) と、システム全体の高可用性 (High Availability) 、特に HA マネージャーについて説明します。WLM は、負荷分散だけでなく、アプリケーション・サーバーが利用できない場合に、他の利用できるアプリケーション・サーバーにリクエストを割り振るフェイルオーバーの機能も提供します。

3-1 WLM 概要

クラスターを作成することによりクラスター・メンバー間での WLM が有効になります。Web コンテナーへのリクエストに対する WLM を Web コンテナーWLM と呼びます。Web コンテナーWLM は、Web コンテナー・サービスを実行するアプリケーション・サーバーをクラスター化することにより実現できます。EJB コンテナーへのリクエストに対する WLM を EJB コンテナーWLM と呼びます。EJB コンテナーWLM は、EJB コンテナー・サービスを実行するアプリケーション・サーバーをクラスター化することにより実現できます。

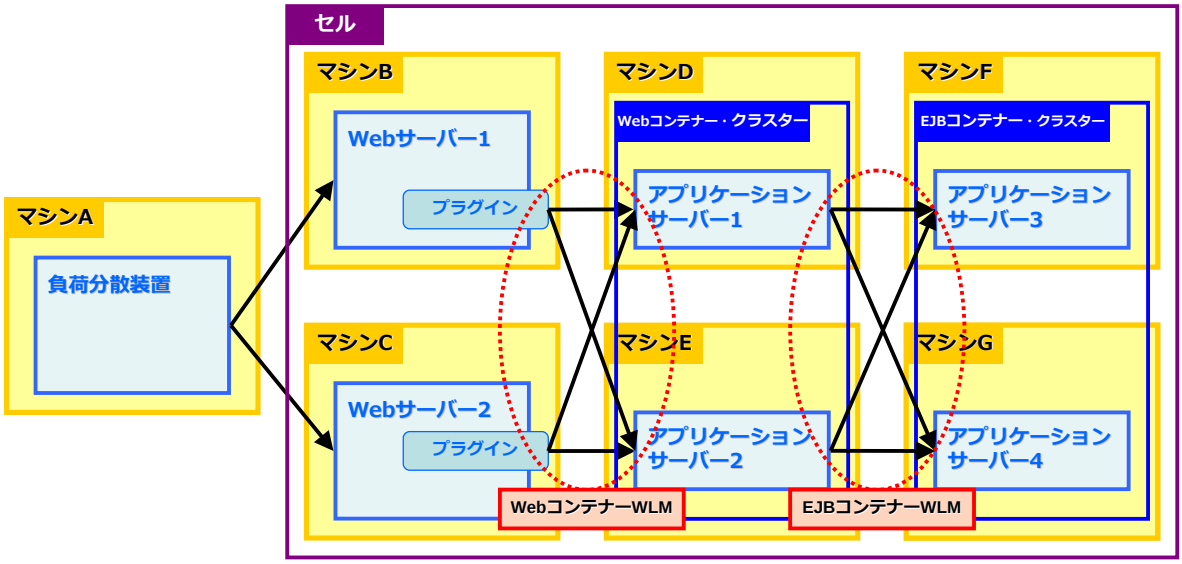

図 3-1 Web コンテナーWLM と EJB コンテナーWLM

WAS V8.5 からは、動的クラスターが使用可能です。動的クラスターとは、動的に拡張可能なクラスターであり、負荷状況や重要度に基づいて、製品が自立的にクラスター・メンバーの開始、停止を行うことが可能です。オンデマンド・ルーター (以下、ODR) を使用することで、割り振り可能なサーバーの重み付けを変更し動的に WLM を行うことが可能です。これを DWLM と呼びます。DWLM については「3-4-1 DWLM」を参照してください。

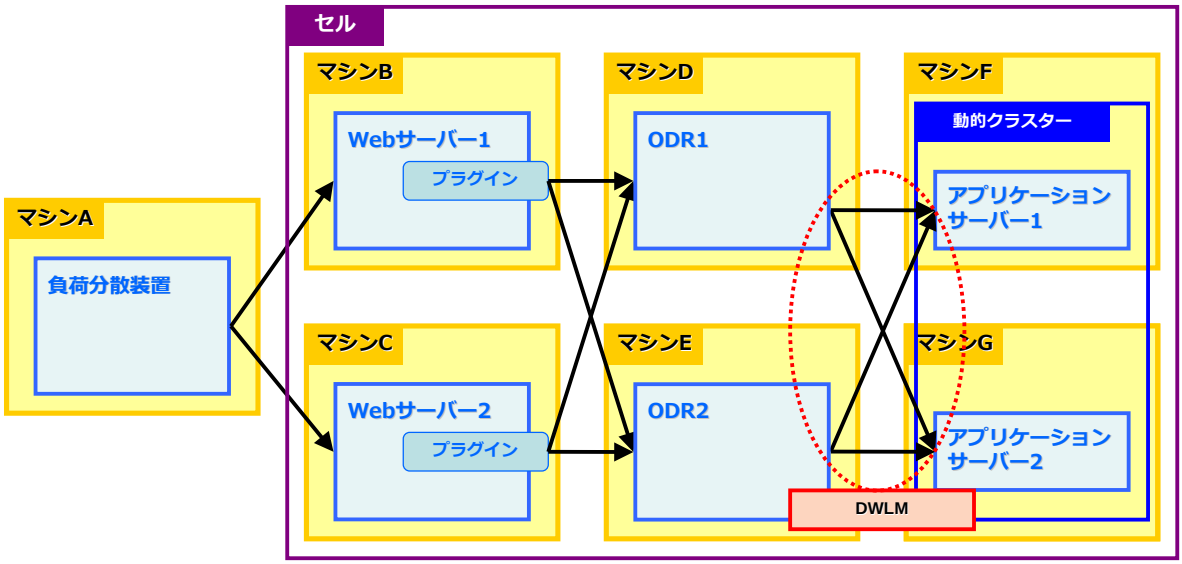

図 3-2 DWLM

3-2 Web コンテナーWLM

この節では、Web コンテナーWLM の構成方法を説明します。Web コンテナーWLM は Web コンテナー・サービスを実行するアプリケーション・サーバーをクラスター化することにより実現できます。具体的には、Web コンテナーWLM を実行するコンポーネントである Web サーバー・プラグインと Web コンテナーの設定とその設定に応じた動きを説明します。また、HTTP セッション ID の有無によるリクエストの処理フローの違いについて説明します。セッション・データをデータベースまたはメモリー間コピーによって他のアプリケーション・サーバーのメモリー上にコピーし、高可用性を向上するセッション・パーシスタンスの機能については、**「1-2-4-6 セッション・パーシスタンス・トポロジー」**を参照してください。

3-2-1 負荷分散メカニズム

Web サーバーが受け付けたリクエストを、どの Web コンテナーに割り振るかを決定する Web コンテナーWLM は、Web サーバー・プラグインにおいて行われます。Web コンテナーWLM の設定を行うには、まず、Web サーバー・プラグインがリクエストを割り振り先 Web コンテナーに転送する処理フローを理解することが重要です。Web サーバー・プラグインがリクエストを処理するフローの概要を**図 3-3**に示します。Web サーバー・プラグインは割り振り先の定義を記述したプラグイン構成ファイル（plugin-cfg.xml）を読み込み、その定義にしたがって割り振り先を決定します。尚、このプラグイン構成ファイルは、xml 形式のファイルであり内容を確認することが可能です。

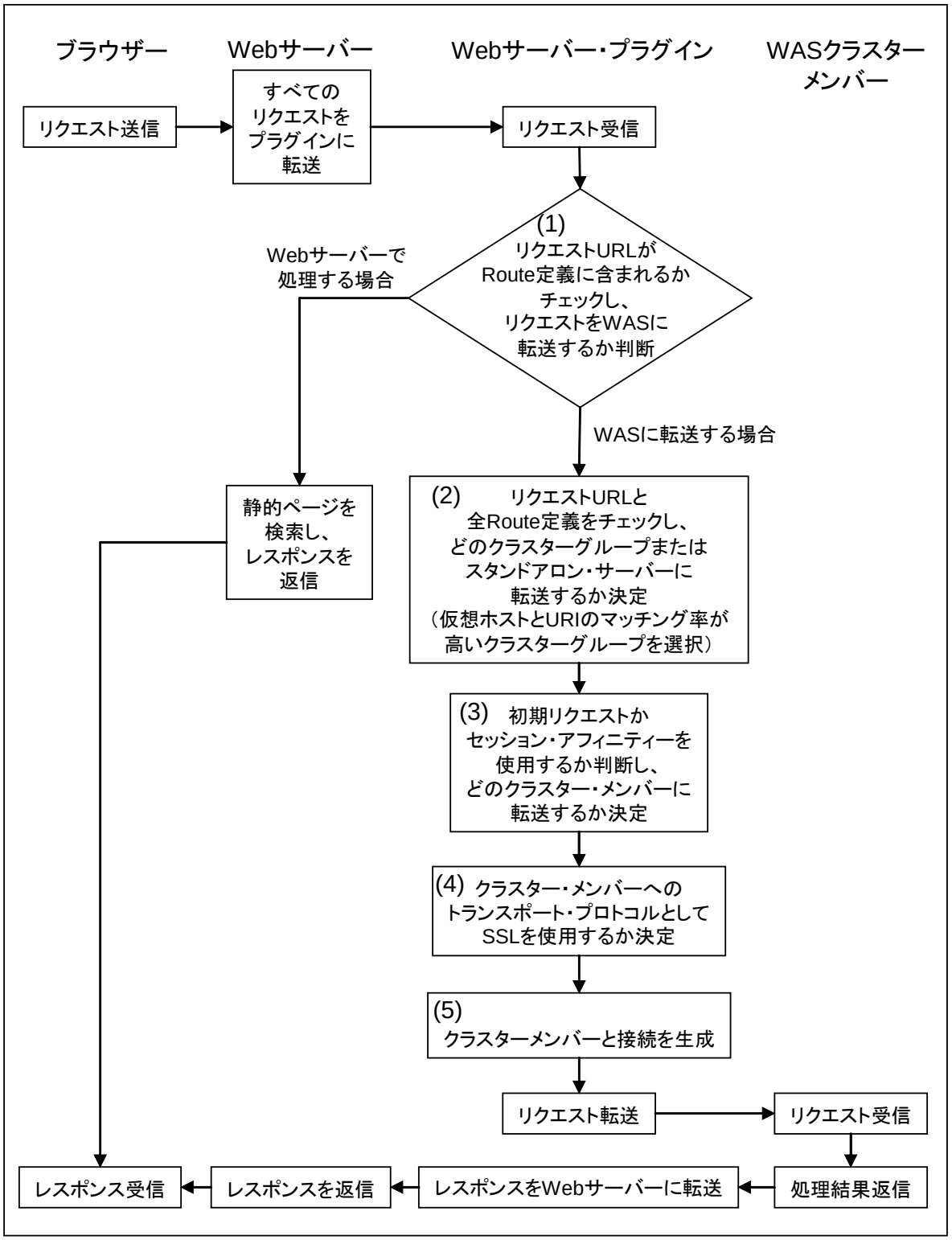

図 3-3 Web サーバー・プラグインのリクエスト処理フロー

図 3-3の処理フローについて、例として、http://host1.ise.ibm.com/hitcount のリクエストをブラウザーから Web サーバーに行った場合を使用して説明します。尚、hitcount のアプリケーションは、「DefaultApplication」に含まれるサンプルのアプリケーションです。今回例として説明するのに使用する

システム環境は**図 3-4**の構成です。また、Web サーバーが読み込むプラグイン構成ファイルの抜粋を**例 3-1**に示します。

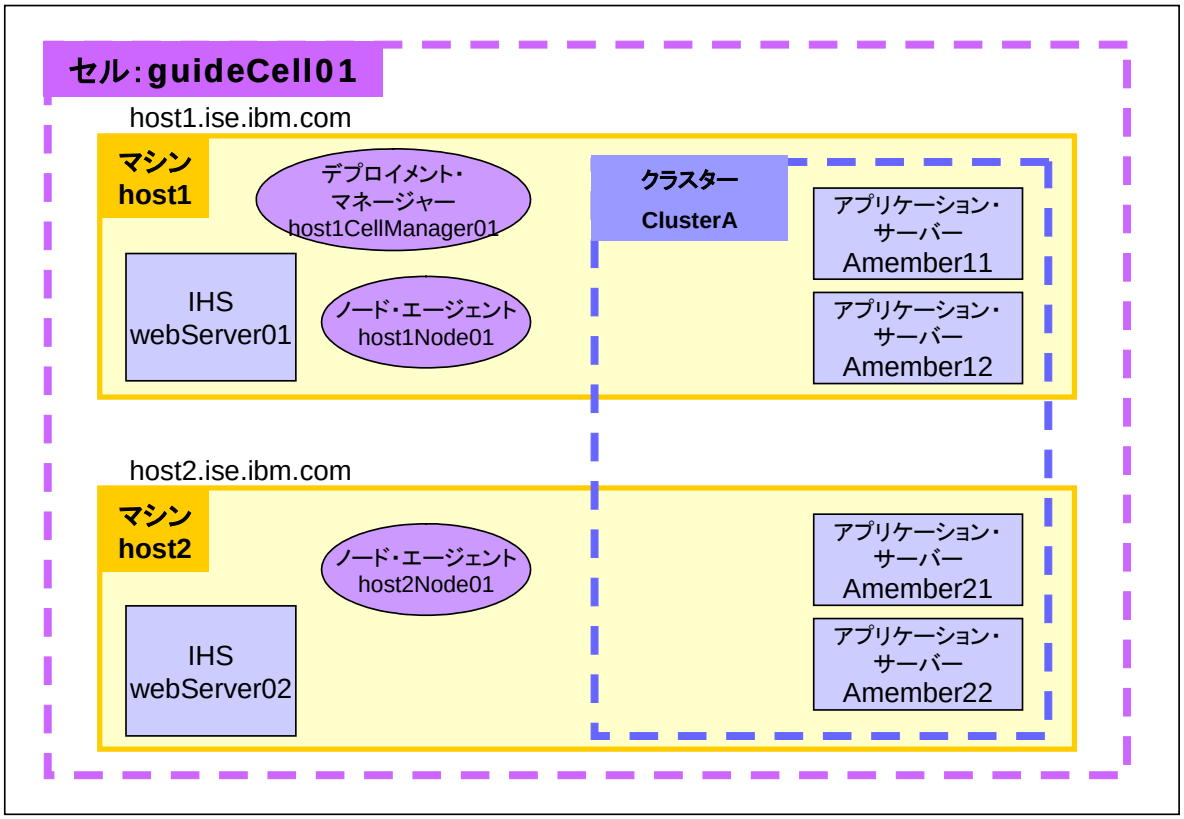

図 3-4 サンプルのシステム環境

例 3-1 プラグイン構成ファイル(plugin-cfg.xml)の例(一部抜粋)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Config ASDisableNagle="false" AcceptAllContent="false"
        AppServerPortPreference="hostHeader" ChunkedResponse="false"
        FIPSEnable="false" IISDisableNagle="false" IISPluginPriority="High"
        IgnoreDNSFailures="false" RefreshInterval="60"
        ResponseChunkSize="64" VHostMatchingCompat="false">
    <Log LogLevel="Error" Name="・・・・・/http_plugin.log"/>
    <Property Name="ESIEnable" Value="true"/>
    <Property Name="ESIMaxCacheSize" Value="1024"/>
    <Property Name="ESIInvalidationMonitor" Value="false"/>
    <VirtualHostGroup Name="default_host">
        <VirtualHost Name="*:9080"/>
        <VirtualHost Name="*:80"/>
```

```
        <VirtualHost Name="*:9443"/>
    </VirtualHostGroup>
    <ServerCluster CloneSeparatorChange="false" GetDWLMTable="false" IgnoreAffinityRequests="true"
        LoadBalance="Round Robin" Name="ClusterA" PostBufferSize="64" PostSizeLimit="-1"
        RemoveSpecialHeaders="true" RetryInterval="60">
        <Server CloneID="13l6924ja" ConnectTimeout="5" ExtendedHandshake="false"
        LoadBalanceWeight="2" MaxConnections="-1" Name="Amember11"
        ServerIOTimeout="60" WaitForContinue="false">
            <Transport Hostname="host1" Port="9080" Protocol="http"/>
            <Transport Hostname="host1" Port="9443" Protocol="https">
                <Property name="keyring" value="••••/plugin-key.kdb"/>
                <Property name="stashfile" value="••••/plugin-key.sth"/>
            </Transport>
        </Server>
        <PrimaryServers>
          <Server Name="host1Node01_Amember11" />
        </PrimaryServers>
    </ServerCluster>
    <UriGroup Name="default_host_ClusterA_URIs">
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/snoop/*" />
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hitcount" />
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsp" />
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsv" />
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsw" />
    </UriGroup>
    <Route ServerCluster="ClusterA" UriGroup="default_host_ClusterA_URIs"
        VirtualHostGroup="default_host" />
    <RequestMetrics armEnabled="false" loggingEnabled="false" rmEnabled="false" traceLevel="HOPS">
        <filters enable="false" type="URI">
            <filterValues enable="false" value=" /snoop"/>
            <filterValues enable="false" value="/hitcount"/>
        </filters>
        <filters enable="false" type="SOURCE_IP">
```

```
            <filterValues enable="false" value="255.255.255.255"/>
            <filterValues enable="false" value="254.254.254.254"/>
        </filters>
    </RequestMetrics>
</Config>
```

Web サーバーでは、受け取ったリクエストをすべてプラグインに転送します。たとえ、Web サーバー上の静的コンテンツに対しリクエストが行われた場合でも、一度 Web サーバー・プラグインに処理が渡されます。

1). Web サーバー・プラグインは、まずリクエスト URL が WAS で処理するものなのか、Web サーバーで処理するものなのかを判定します。プラグインは、受け取ったリクエストのホスト名とポート番号の組(この例では、host1.ise.ibm.com:80)と URI(この例では、/hitcount)が、プラグイン構成ファイルの各 Route ディレクティブの仮想ホストの設定と URI の設定に適合するものがあるか検索します。適合する Route ディレクティブが存在しない場合には、Web サーバーでリクエストを処理するものとして、Web サーバーに処理を戻します。適合する Route ディレクティブが存在する場合には、WAS に処理を転送するため、プラグインで後続の処理を実行します。

プラグイン・ログ(http_plugin.log)のログ・レベルをトレースに設定しログを取得すると下記のような情報が記録されます。尚、プラグイン・ログのトレース取得方法は、「3-2-5-2プラグイン・ログ」を参照ください。

```
[Mon Nov 10 15:30:05 2008] 0005b018 00000708 - TRACE: ws_common: websphereVhostMatch: Comparing '*:80' to 'host1.ise.japan.ibm.com:80' in VhostGroup: default_host

[Mon Nov 10 15:30:05 2008] 0005b018 00000708 - DEBUG: ws_common: websphereVhostMatch: Found a match '*:80' to 'host1.ise.japan.ibm.com:80' in VhostGroup: default_host with score 1, exact match 0

....

[Mon Nov 10 15:30:05 2008] 0005b018 00000708 - TRACE: ws_common: websphereFindServerGroup: trying to match a route for: vhost='host1.ise.japan.ibm.com'; uri='/hitcount'

[Mon Nov 10 15:30:05 2008] 0005b018 00000708 - DEBUG: ws_common: websphereUriMatch: Found a match '/hitcount' to '/hitcount' in UriGroup: default_host_cluster1_URIs with score 9, exact match 9
```

2). リクエストをどのクラスター・グループまたはスタンドアロン・サーバーに転送するかを決定します。1)の手順と同様に、各 Route ディレクティブで設定される仮想ホストと URI の設定と適合するものがあるか検索し、適合する仮想ホストと URI の文字数をカウントします。複数の Route ディレクティブの設定に適合する場合は、適合する仮想ホストと URI の文字数の最も多い Route ディレクティブのクラスター・グループまたはスタンドアロン・サーバーに転送します。プラグイン・ログのトレースには、下記のような情報が記録されます。

```
[Mon Nov 10 15:30:05 2008] 0005b018 00000708 - TRACE: ws_common: websphereFindServerGroup: Setting the server group: ClusterA; curScore of 10 greater than high of 0

[Mon Nov 10 15:30:05 2008] 0005b018 00000708 - DETAIL: ws_common: websphereFindServerGroup: Setting the server group: cluster1; highScore: 10; highExactMatch: 9; affinityCookie: JSESSIONID; affinityURL: jsessionid
```

3). リクエストを決定されたクラスター・グループ内のどのクラスター・メンバーに転送するかを決定します。プラグインは、リクエスト情報の中にセッション ID が含まれているかをチェックします。セッション ID が含まれている場合には、セッション・アフィニティーの機能を使用し、プラグインは、前回のリクエストと同一のクラスター・メンバーにリクエストを転送します。セッション ID が含まれて

いない場合には、WLM ポリシー（ラウンドロビンまたはランダム）にしたがって、リクエストを転送するクラスター・メンバーを決定します。セッション・アフィニティーのメカニズムと、WLM ポリシーによる、転送先クラスター・メンバーの決定メカニズムは後述します。

(i) リクエスト内にセッション識別子が含まれている場合はプラグイン・ログのトレースに下記のように記録されます。

```
[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_common: websphereHandleSessionAffinity: Checking for session affinity

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_common: websphereHandleSessionAffinity: Checking the SSL cookie affinity: SSLJSESSION

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: lib_htrequest: htrequestGetCookieValue: No cookie found for: 'SSLJSESSION'

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID', value='0000fRJ11RgyNc8J2GWDIWZK8C4:13maa91k3'

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_common: websphereHandleSessionAffinity: Checking the JSESSIONID in cookie: 0000fRJ11RgyNc8J2GWDIWZK8C4:13maa91k3

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - DEBUG: ws_common: websphereParseCloneID: Parsing clone ids from '0000fRJ11RgyNc8J2GWDIWZK8C4:13maa91k3'

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_common: websphereParseCloneID: Adding clone id '13maa91k3'

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_common: websphereParseCloneID: Returning list of clone ids

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_server_group: serverGroupFindClone: Looking for clone

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID '13maa91k3' to server clone id '13maa91k3'

[Thu Nov 13 16:19:46 2008] 000710c8 00000809 - TRACE: ws_server_group: serverGroupFindClone: Match for clone 'host1_Amember11'
```

(ii) リクエスト内にセッション識別子が含まれていない場合はプラグイン・ログのトレースに下記のように記録されます。

```
[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: ws_common: websphereHandleSessionAffinity: Checking for session affinity
```

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: ws_common: websphereHandleSessionAffinity: Checking the SSL cookie affinity: SSLJSESSION

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: lib_htrequest: htrequestGetCookieValue: No cookie found for: 'SSLJSESSION'

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: lib_htrequest: htrequestGetCookieValue: No cookie found for: 'JSESSIONID'

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: ws_common: websphereHandleSessionAffinity: Checking the url rewrite affinity: jsessionid

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: ws_common: websphereParseSessionID: Parsing session id from '/hitcount'

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: ws_common: websphereParseSessionID: No session found for jsessionid

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - TRACE: ws_common: websphereHandleSessionAffinity: Bypassing check for partitionID cookie affinity. No stored partition table.

[Thu Nov 13 16:19:03 2008] 000710c8 00000708 - DEBUG: ws_server_group: serverGroupNextRoundRobinServer: Round Robin load balancing

4). プラグインとクラスター・メンバー間の通信プロトコルとして SSL を使用するか否か(HTTP か HTTPS か)を決定します。今回の設定と同様にデフォルトの設定では、各クラスター・メンバーは HTTP と HTTPS の 2 つの通信プロトコルと関連付けられています。この場合、クライアントから Web サーバーへのアクセスが SSL を使用している場合は、プラグインからクラスター・メンバー間の通信も SSL を使用し、クライアントから Web サーバーへのアクセスが SSL を使用していない場合は、プラグインからクラスター・メンバー間の通信も SSL を使用せず、HTTP で通信します。HTTP または HTTPS のどちらかのみのプロトコルでプラグインとクラスター・メンバー間の通信を行う場合には、トランスポート・チェーンの設定[サーバー]→[サーバー・タイプ]→[アプリケーション・サーバー]→[ApplicationServer_name] →[コンテナー設定]→[Web コンテナー設定]→[Web コンテナー・トランスポート・チェーン] (図 3-5参照)において、[WCInboundDefault]または[WCInboundDefaultSecure]のうち使用するトランスポート・チェーンのみを使用可能に設定します。使用しないトランスポート・チェーンの[使用可能]のチェックボックスを OFF に設定します。尚、[WCInboundAdminDefault]と[WCInboundDefaultAdminSecure]は管理ツールが使用しますので、設定を変更する必要はありません。

図 3-5 トランスポート・チェーンの設定

5). プラグインとクラスター・メンバー間で物理的な接続を生成します。HTTP 1.1 では、KeepAlive 接続を行うことができるので、必ずしも新規の接続を生成するわけではなく、既存の接続を複数のリクエストが順に使用する場合もあります。

その後、リクエストは上記の接続を使用して WAS のクラスター・メンバーに転送されます。WAS で、レスポンスが生成され、レスポンスはプラグインに戻されます。プラグインが Web サーバーにレスポンスを転送し、最終的に Web サーバーがクライアントにレスポンスを返します。

3-2-2 負荷分散設定

「3-2-1負荷分散メカニズム」を制御するための設定変更の方法と挙動について説明します。

3-2-2-1 プラグイン WLM ポリシー

プラグイン WLM ポリシーは、プラグインがリクエストの転送先クラスター・メンバーを選択する方法を指定します。プラグインが負荷分散を行うアルゴリズムとして、以下の 2 つの方法があります。

- 重み付きラウンドロビン
- ランダム

デフォルト値は重み付きラウンドロビンです。この設定は、**[サーバー]→[サーバー・タイプ]→[Web サーバー]→[*WebServer_name*] →[追加プロパティー]→[プラグイン・プロパティー]→[追加プロパティー]→[要求ルーティング]**のページ(図 3-6参照)で変更することができます。尚、**「3-2-2-2セッション・アフ**

ィニティー」で説明するセッション・アフィニティーの機能は、プラグイン WLM ポリシーよりも優先されます。

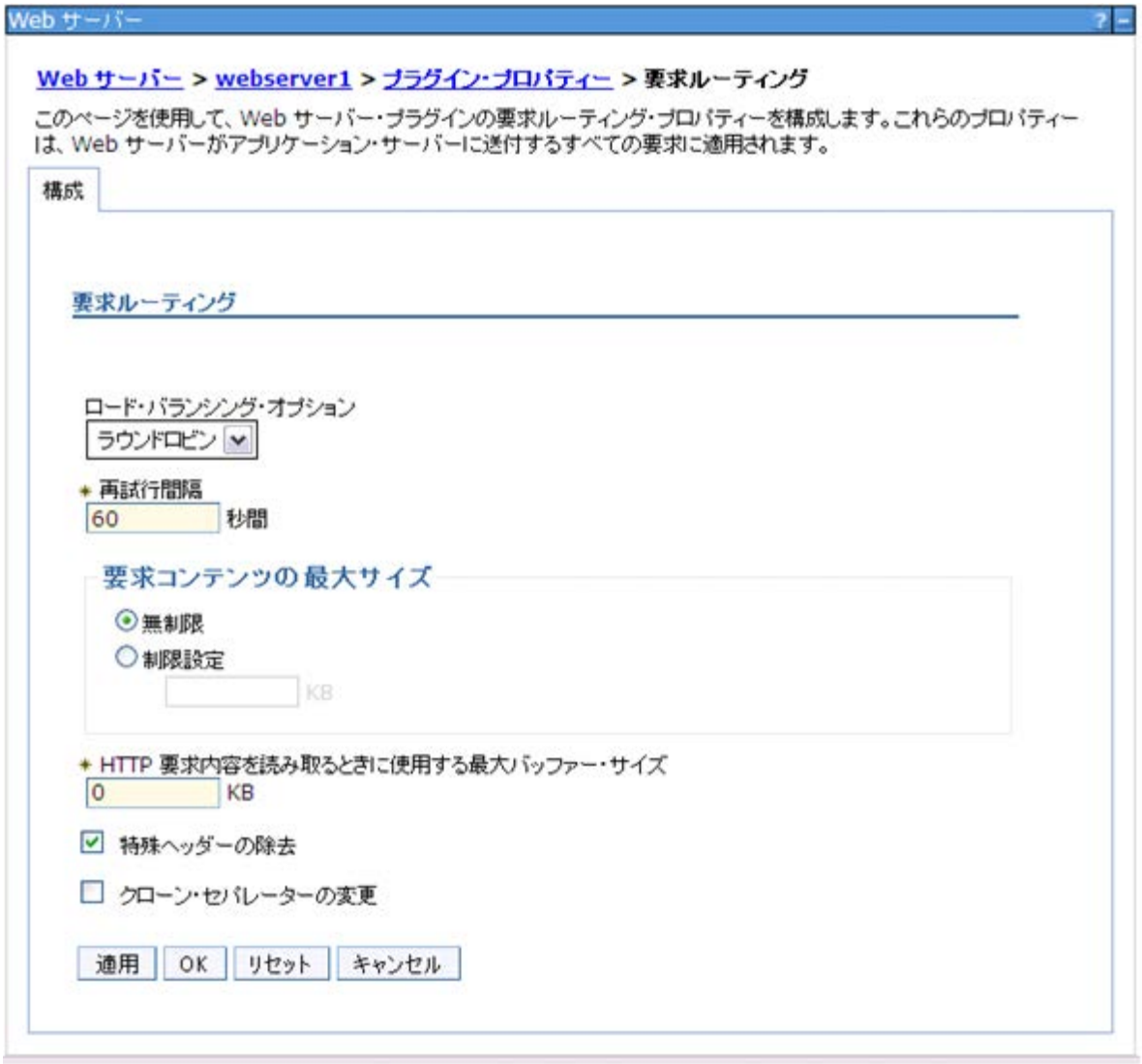

図 3-6 プラグイン WLM ポリシーの設定

重み付きラウンドロビン

クラスター・メンバーの重みは、管理コンソールでクラスター・メンバーのプロパティー（**[サーバー]→[クラスター]→[WebSphere Application Server クラスター]→[*Cluster_name*] →[追加プロパティー]→[クラスター・メンバー]→[*ClusterMember_name*]**（**図 3-7参照**））で確認・変更することができます。また、クラスターの新規作成やクラスター・メンバーの追加時にも変更することができます。デフォルトの重みは 2 です。有効な値の範囲は 0 から 100 の整数値です。

図 3-7 クラスター・メンバーの重み(ウェイト)設定

重み付きラウンドロビンのアルゴリズムを選択した場合、起動した最初、プラグインはランダムにリクエスト転送先クラスター・メンバーを選択します。そして、リクエストを転送したクラスター・メンバーの重みを 1 減じます。後続のリクエストは、ラウンドロビンのアルゴリズムで、異なるクラスター・メンバーに順番にリクエストが転送されます。重みが 0 になったクラスター・メンバーへのリクエストの転送は行われません。すべてのクラスター・メンバーの重みが 0(もしくは 0 以下)になると、重みはリセット(現在の値に設定している重みが加算)されます。

ただし、クライアントからのリクエストにセッション情報が存在する場合には、セッション・アフィニティーの機能が優先され、そのクライアントが前回アクセスしたクラスター・メンバーにリクエストが転送されます。セッション・アフィニティーの機能で、転送先クラスター・メンバーが決定された場合も、そのクラスター・メンバーの重みは 1 減じられます。これにより、クラスター・メンバーの重みが 0 より小さい値になることもあります。

表 3-1で具体的に重み付きラウンドロビンがどのように機能するか説明します。2 つのクラスター・メンバーが存在し、それぞれのクラスター・メンバーの重みが 3、2 とします。最初のリクエストは、ランダムに割り振られます。ここでは、クラスター・メンバー1 が選択されたとします。その後の新規リクエスト(リクエス

ト2～4）はラウンドロビンで交互にリクエストが割り振られます。セッション ID を含んだリクエスト（リクエスト5）の場合は、ラウンドロビンに優先して、セッション・アフィニティーの機能が有効になり、前回と同一のクラスター・メンバーに割り振られます。すべてのクラスター・メンバーの重みが 0 以下になると、重みはリセットされます。ここでは、設定された重みの 3 と 2 が、リクエスト 6 終了後の重みにそれぞれ加算されています。

表 3-1 重み付きラウンドロビンによるリクエスト転送先クラスター・メンバーの選択

	クラスター・メンバー1	クラスター・メンバー2
初期値	3	2
リクエスト 1	2	2
リクエスト 2	2	1
リクエスト 3	1	1
リクエスト 4	1	0
リクエスト 5 (クラスター・メンバー2 にアフィニティーのあるリクエスト)	1	-1
リクエスト 6	0	-1
リセット	3	1

尚、この WLM による割り振り先の決定ロジックの詳細は、下記のような注意点があります。

- この重みのテーブルは、各 Web サーバーの httpd プロセスごとに保持しています。
- Web サーバーの起動時に、プラグインはクラスター・メンバーの重みとして、それぞれの重みを最大公約数で割った値を使用します。つまり、クラスター・メンバーの重みとして 6 と 4 を設定した場合も、表 3-1と同様に 3 と 2 として動きます。これは、重みの小さなクラスター・メンバーが連続してアイドルしている時間を短くするためです。
- もし、特定のクラスター・メンバーへのアフィニティーが有効となるリクエストが多く、リセット前の重みが大きな負の値となっていた場合は、リセット時に全てのクラスター・メンバーの重みが正の値となるように、それぞれに設定された重みの数倍の値が加算されます。例えば、表 3-1の設定でリセット前の重みが 0 と-7 だった場合、リセット時には、クラスター・メンバー1 には、12(=3×4)が、クラスター・メンバー2 には、8(=2×4)が加算されます。

ランダム

リクエストは各クラスター・メンバーにランダムに割り振られます。重みの設定値は考慮されません。

3-2-2-2 セッション・アフィニティー

プラグインには、同一クライアントからのリクエストを特定のクラスター・メンバーに転送し続けるセッション・アフィニティーの機能があります。Servlet2.5 の仕様では、一度セッションが作成された後は、同一クライアントからのリクエストは同じアプリケーション・サーバーに割り振られる必要があることが定義されています。同一の HTTP セッションを持つリクエストを同一クラスター・メンバーに転送することで、セッション・マネージャーのキャッシュ効率が上がりパフォーマンス向上に寄与します。**表 3-2**の 3 種類のセッション管理の方法のいずれかを用います。ただし、SSL ID によるセッション管理は WAS V7.0 より非推奨となりました。セッション識別子がクライアントからのリクエストに含まれていれば、プラグインは識別子中のクローン情報に基づき、割り振り先のサーバーを選択します。このとき、アフィニティー先のアプリケーシ

ョン・サーバーがダウンしていると判断した場合は、他の利用可能なアプリケーション・サーバーへリクエストを転送(フェイルオーバー)します。

表 3-2 セッション管理の方法

セッション管理の方法	説明
Cookie	最も広く使われているセッション管理の方法
URL 再書き込み	Cookie に対応していないブラウザー(モバイル端末等)への対応に使用
SSL ID (V7.0 より非推奨)	セッション管理の方法として使用されるが、セッション・アフィニティーの機能を有効にするためには、Cookie または URL 再書き込みとの併用が必要

セッション識別子は、具体的には以下のような値が使用されます。

```
Cookie: JSESSIONID=0000fRJ11RgyNc8J2GWDIWZK8C4:13maa91k3
```

例 3-2 セッション識別子の例

この JSESSIONID の値は下記の表 3-3のように 4 つの部分に分けられます。

表 3-3 セッション識別子の構成

要素	値の例
Cache ID	0000
Session ID	fRJ11RgyNc8J2GWDIWZK8C4
Separator	:
Clone ID または Partition ID	13maa91k3 または M9UFFV7LL

プラグインは、セッション識別子中の Clone ID の値を、プラグイン構成ファイル(**例 3-1 プラグイン構成ファイル(plugin-cfg.xml)の例(一部抜粋)**)参照)の<Server>タグ中の CloneID 属性の値と比較し、どのクラスター・メンバーにリクエストを割り振るかを判断します。

ピアツーピアモードでのメモリー間複製によるセッション・パーシスタンスを設定した場合に、Partition ID によるアフィニティーが行われます。HA マネージャー(詳細は後述)がそれぞれのクラスター・メンバーに対応する Partition ID を生成し、Partition ID と Clone ID のマッピングを表すテーブルを保守します。このテーブルは、クライアントへのレスポンスの Header 情報の一部として、クラスター・メンバーからプラグインに送付されます。このときのプラグイン・ログのトレースを**例 3-3**に示します。この Header 情報はプラグインで除去され、クライアントへは送付されません。

クライアントからのリクエストにセッション識別子が含まれる場合、プラグインは、最初にその ID が Clone ID のいずれかに一致するかチェックし、一致する場合、そのクラスター・メンバーにリクエストを転送します。Clone IDに一致しない場合、引き続き、Partition IDのいずれかに一致するかをチェックし、一致する場合には、その Partition ID に対応する Clone ID のクラスター・メンバーにリクエストを転送します。Partition ID にも一致しない場合には、重み付きラウンドロビンもしくはランダムの WLM ポリシーにしたがって、転送先クラスター・メンバーが決定されます。

```
[Thu Nov 13 19:44:59 2008] 0003808c 00000708 - TRACE: ws_common: ParsePartitionIDs: Parsing partitionID pair from 'M9UFFV7LL:13maa91k3;1JCBPNLK1Q:13maa9htn;'

[Thu Nov 13 19:44:59 2008] 0003808c 00000708 - TRACE: ws_common: ParsePartitionIDs: Adding partitionID / clone pair 'M9UFFV7LL' : '13maa91k3'

[Thu Nov 13 19:44:59 2008] 0003808c 00000708 - TRACE: ws_common: ParsePartitionIDs: Adding partitionID / clone pair '1JCBPNLK1Q' : '13maa9htn'

[Thu Nov 13 19:44:59 2008] 0003808c 00000708 - TRACE: ws_common: ParsePartitionIDs: Returning partitionID / cloneid pair list
```

例 3-3 Partition ID のテーブルをプラグインが取得したときのプラグイン・ログのトレース

もし、プラグインが Partition ID のテーブルにしたがってリクエストを転送し、そのクラスター・メンバーがダウンしていた場合、プラグインは、最新の Partition ID のテーブルを取得するリクエストを別途送信します。その新しいテーブルを使用して、再度リクエストが転送されます。この Partition ID のメカニズムを使用することにより、ピアツーピアモードでのメモリー間複製によりセッション・パーシスタンスを行っているときに、プラグインは、セッション情報が既に保持されているクラスター・メンバーへのホット・フェイルオーバーが可能になります。

3-2-3 フェイルオーバー・メカニズム

プラグインはクラスター・メンバー間の負荷分散だけでなく、あるクラスター・メンバーに障害が発生した場合、その障害を検知し、他の正常なクラスター・メンバーにリクエストを割り振るフェイルオーバーもサポートします。

プラグインがリクエストの割り振り先を決定し、そのクラスター・メンバーとの通信を行うとき、その接続が切断されたり、通信が失敗したりすると、プラグインはそのクラスター・メンバーをダウンとマークし、異なる割り振り先を決定し、再度リクエストの割り振りを行います。

フェイルオーバー処理の概要は下記の**図 3-8**のようになります。

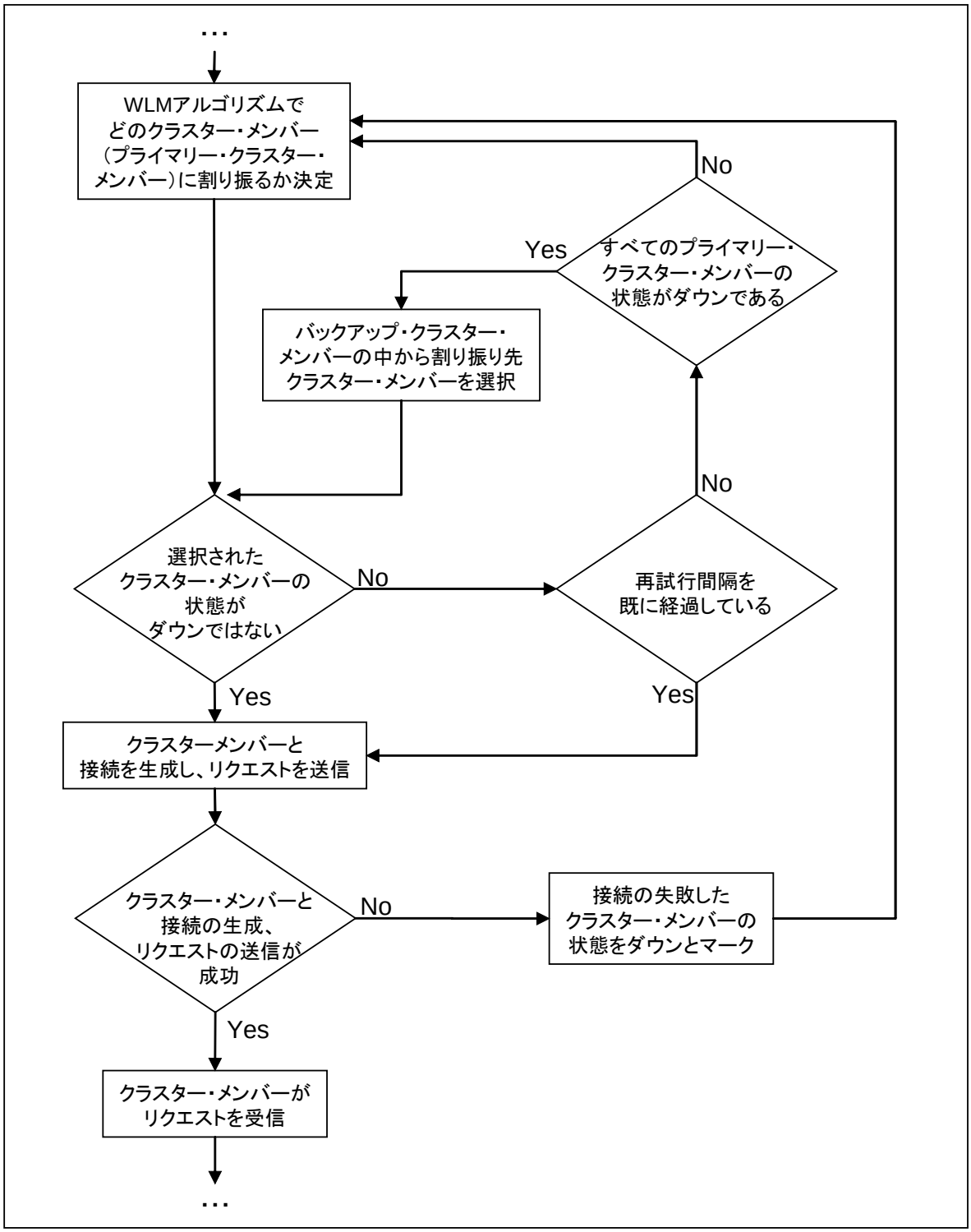

図 3-8 フェイルオーバーの処理フロー

このように、接続に失敗し障害とみなしたクラスター・メンバーへは割り振りを行いません。障害とみなしたクラスター・メンバーに対しては、プラグイン設定の再試行間隔の時間が経過後に、リクエストの割り振りを試みます。ここで、プライマリー・クラスター・メンバーとバックアップ・クラスター・メンバーの選択と再試行間隔の時間は変更することが可能です。設定方法はそれぞれ、**「3-2-4-5プライマリー、バックア**

ップ・サーバー」、「3-2-4-1再試行間隔」で説明します。デフォルトでは、すべてのクラウスター・メンバーがプライマリーであり、再試行間隔は 60 秒です。

クラスター・メンバーの障害はプラグインがリクエストを割り振った時点で検知されます。TCP 接続が失敗した場合に障害と判断し、割り振りを停止します。

アプリケーション・サーバーが稼働しているマシンの OS が稼働している状態で、意図的にアプリケーション・サーバーを停止した場合や、予期しない障害でアプリケーション・サーバーの Java プロセスがダウンしている場合、プラグインからの TCP 接続要求(SYN)に対して RST が送信され、即時に TCP 接続は失敗します。この場合、プラグインはクラスター・メンバーを即時に障害と判断し、割り振りを停止します。

アプリケーション・サーバーの稼働するマシンのシステム障害（OS 障害、システム停止、電源障害）やネットワーク障害の場合、TCP 接続要求は即座に失敗せず、プラグインの接続タイムアウトに従い、タイムアウト経過後にその接続を失敗とみなします。この接続タイムアウトのデフォルトの設定値は 5 秒です。V6.1 では、接続タイムアウトはデフォルトで 0（無制限（OS のタイムアウト値が使用される））が設定されていましたが、V7.0 ではこのデフォルト値が変更されています。接続タイムアウトは、「3-2-4-2接続タイムアウト、読み取り/書き込みタイムアウト」で説明します。

3-2-4 フェイルオーバー設定

「3-2-3フェイルオーバー・メカニズム」を制御するための設定の変更の方法とそのときの動きについて説明します。

3-2-4-1 再試行間隔

一度ダウンとマークされたクラスター・メンバーに対して、再度アクセスを試みるまでの時間を再試行間隔の時間で設定します。設定は、WLM ポリシーの設定と同様に、[サーバー]→[サーバー・タイプ]→[Web サーバー]→[*WebServer_name*] →[追加プロパティー]→[プラグイン・プロパティー]→[追加プロパティー]→[要求ルーティング]のページ（**図 3-6参照**）での[再試行間隔]の項目で変更することができます。

この時間を長くすることにより、不必要にダウンしているアプリケーション・サーバーに対してリクエストを試行することを避けることができます。逆に長くしすぎると、障害から復旧したサーバーに対して、リクエストが割り振られるようになるまでの時間が長くなってしまいます。この値のデフォルト値は、60 秒です。この値の特定の推奨値はなく、その環境に依存します。例えば、クラスター・メンバーの数が十分多く、1 台のクラスター・メンバーの停止が全体の負荷にあまり影響を及ぼさない場合には、再試行間隔を長くすることができます。また、アプリケーション・サーバーの起動時間も考慮してください。アプリケーション・サーバーの起動とアプリケーションのロードに時間がかかる場合には、再試行間隔もそれに応じて長くすることが推奨されます。

3-2-4-2 接続タイムアウト、読み取り/書き込みタイムアウト

プラグインとアプリケーション・サーバー間の TCP 接続におけるタイムアウトを OS の TCP タイムアウトでなく、プラグイン独自の設定を行うことができます。設定は、プライマリー、バックアップ・サーバーの設定と同様の箇所で、各アプリケーション・サーバーの Web サーバーのプラグイン・プロパティー（[サーバー]→[サーバー・タイプ]→[アプリケーション・サーバー]→[*ApplicationServer_name*] →[追加プロパティー]→[Web サーバーのプラグイン・プロパティー] （**図 3-9**））の[接続タイムアウト]の項目で変更することができます。

接続タイムアウトの設定値に 0 よりも大きい値で、正常に接続されるまでプラグインが待機する秒数を指定します（0 はプラグインの設定による接続タイムアウトの指定はないことを表します）。デフォルト値は

5 秒になります。その時間間隔を経過後も接続が行われなかった場合、プラグインは、サーバーに割り振り不可のマークを付けて、リクエストを別のアプリケーション・サーバーにフェイルオーバーします。

また、TCP 接続確立後、プラグインからアプリケーション・サーバーへの HTTP リクエストが応答完了するまでのタイムアウトもプラグイン独自の設定で行うことができます。この設定箇所も各アプリケーション・サーバーの Web サーバーのプラグイン・プロパティーの**[読み取り/書き込みタイムアウト]**の項目で変更することができます。

読み取り/書き込みタイムアウトの設定値に 0 よりも大きい値で、プラグインからアプリケーション・サーバーにリクエストを割り振り、応答が返るまでプラグインが待機する秒数を指定します（0 はプラグインの設定による読み取り/書き込みタイムアウトの指定はないことを表します）。デフォルト値は 60 秒になります。アプリケーション・サーバーがハングしていたなど、その時間間隔を経過後も応答が返らなかった場合、プラグインは、サーバーに割り振り不可のマークを付けて、再度リクエストを別のアプリケーション・サーバーにフェイルオーバーします。

ただし、接続タイムアウトおよび読み取り/書き込みタイムアウトの設定は各環境のネットワークスピードとピーク時の負荷を考慮して、ピーク時にもタイムアウトが発生しないような十分長い時間を設定してください。

図 3-9 Web サーバーのプラグイン・プロパティー

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、前述したようにデフォルト設定ではリクエストが他のアプリケーション・サーバーへ再送されます。これにより、障害発生時には同じ処理が二度実行される可能性があります。例えば、DB の参照処理だけであればこの再送機能は非常に有効ですが、DB の更新処理がある場合には二重実行によりデータの不整合が発生する可能性があります。

ここで、Web サーバー・プラグインの HTTP 要求内容を読み取るときに使用する最大バッファー・サイズを 0 に設定すると、読み取り/書き込みタイムアウトで障害検知されても、Post リクエストに関しては再送しなくなります。(Get リクエストに関しては再送しなくすることはできません。) この設定は、**[サーバー]→[サーバー・タイプ]→[Web サーバー]→[*WebServer_name*] →[追加プロパティー]→[プラグイン・プロパティー]→[追加プロパティー]→[要求ルーティング]**のページ（**図 3-6**参照）で変更することができます。

3-2-4-3 接続の最大数

プラグインとアプリケーション・サーバー間の接続数の最大数を設定することができます。設定は、接続タイムアウトの設定と同様の箇所で、各アプリケーション・サーバーの Web サーバーのプラグイン・プロパティー（**[サーバー]→[サーバー・タイプ]→[アプリケーション・サーバー]→[*ApplicationServer_name*] →[追加プロパティー]→[Web サーバーのプラグイン・プロパティー]** （**図 3-9参照**））の**[アプリケーション・サーバーが処理できる接続の最大数]**の項目で変更することができます。

デフォルト値は**[無制限]**です。この設定値を使用することにより、あるクラスター・メンバーがハングもしくは、高負荷によってレスポンスが遅い場合に、そのサーバーに無制限に接続が滞留してしまうことを避けることができます。しかし、小さな値を設定すると、この値を超過したリクエストは、異なるクラスター・メンバーに割り振られてしまいますので、高負荷時にセッション・アフィニティーが機能しないリクエストが発生する場合がありますので注意してください。

また、この設定は、Web サーバーのプロセスそれぞれから、特定のアプリケーション・サーバーの接続数の制限値になります。具体的には、あるクラスター・メンバーの接続の最大数の設定値が 50 で、このクラスター・メンバーは 5 台の IHS Web サーバーからリクエストが受け付け、それぞれの Web サーバーは 2 つのプロセスを開始する場合には、このクラスター・メンバーへは合計で、最大 500（= 50×5×2）のリクエストを受け付ける可能性があります。

3-2-4-4 拡張ハンドシェーク

拡張ハンドシェークを使用することで、ハングしたアプリケーション・サーバーへリクエストを送信することを防ぐことが可能です。設定は、接続タイムアウトの設定と同様の箇所で、各アプリケーション・サーバーの Web サーバーのプラグイン・プロパティー（**[サーバー]→[サーバー・タイプ]→[アプリケーション・サーバー]→[*ApplicationServer_name*]→[追加プロパティー]→[Web サーバーのプラグイン・プロパティー]** （**図 3-9**参照））の**[アプリケーション・サーバーが実行しているかどうかを検査するために拡張ハンドシェークを使用する]**のチェックボックスで設定することができます。

この設定を有効にした場合、アプリケーション・サーバーへリクエストを送信する前に HEAD リクエストを送信してアプリケーション・サーバーの生死を確認します。ただし、業務ロジックのハングのようにプロセスは実行されていて内部スレッドがハングしている状態では、HEAD リクエストに応答するのでリクエストは割り振られていまいます。

3-2-4-5 プライマリー、バックアップ・サーバー

クラスター・メンバーとなるアプリケーション・サーバーに対する設定で、そのサーバーをプライマリー・サーバー（管理コンソール上は「一次」と表示されます。）とするかバックアップ・サーバーとするかを選択できます。

プラグインは、全てのプライマリー・サーバーがダウンした場合にのみ、バックアップ・サーバーに対して割り振りを行います。バックアップ・サーバーが複数ある場合、重み付きの割り振りは行われず、単純にプラグイン構成ファイルの<BackupServers>にリストされるクラスター・メンバーの最初にリストされたサーバーに割り振りを試みます。最初にリストされたサーバーにアクセスできない場合には、2 番目以降にリストされたサーバーに割り振りを行います。また、バックアップ・サーバーに対してセッション・アフィニテ

ィーは有効になりません。プライマリー・サーバーが 1 台でも使用可能となると、その後のリクエストはプライマリー・サーバーに割り振られることになります。

＜注意＞

- セッションをピアツーピアモードのメモリー間複製の設定を行い、パーティション ID のロジックを使用している場合、プライマリー、バックアップ・サーバーの設定は使用できません。

アプリケーション・サーバーの役割の設定は、各アプリケーション・サーバーの Web サーバーのプラグイン・プロパティー（[**サーバー**]→[**サーバー・タイプ**]→[**アプリケーション・サーバー**]→[*ApplicationServer_name*] →[**追加プロパティー**]→[Web **サーバーのプラグイン・プロパティー**]（**図** 3-9 参照))の[**サーバーの役割**]のプルダウン・メニューで確認・変更することができます。[**一次**]がプライマリー・サーバーを表します。

3-2-5 その他のプラグイン設定

プラグインのその他の設定項目について説明します。

3-2-5-1 プラグインの再ロード間隔

プラグインがプラグイン構成ファイルをロードする間隔を設定することができます。デフォルトではこの間隔は 60 秒です。プラグイン構成ファイル再ロード間隔の設定変更は各 Web サーバーのプラグイン・プロパティー（[**サーバー**]→[**サーバー・タイプ**]→[Web **サーバー**]→[*WebServer_name*] →[**追加プロパティー**]→[**プラグイン・プロパティー**]（図 3-10参照））の[**構成最新表示間隔**]の項目で設定します。

自動的にプラグイン構成ファイルを再ロードする機能は負荷がかかるため、本番の実稼働環境が安定した場合は、必要最低限にしたほうがよいと考えられます。ただし、本番運用時に、プラグイン構成ファイルを読み込みなおすことによって、割り振り先を動的に変更する場合には、デフォルトの 60 秒間隔のように定期的に読み込む設定にしておく必要があります。

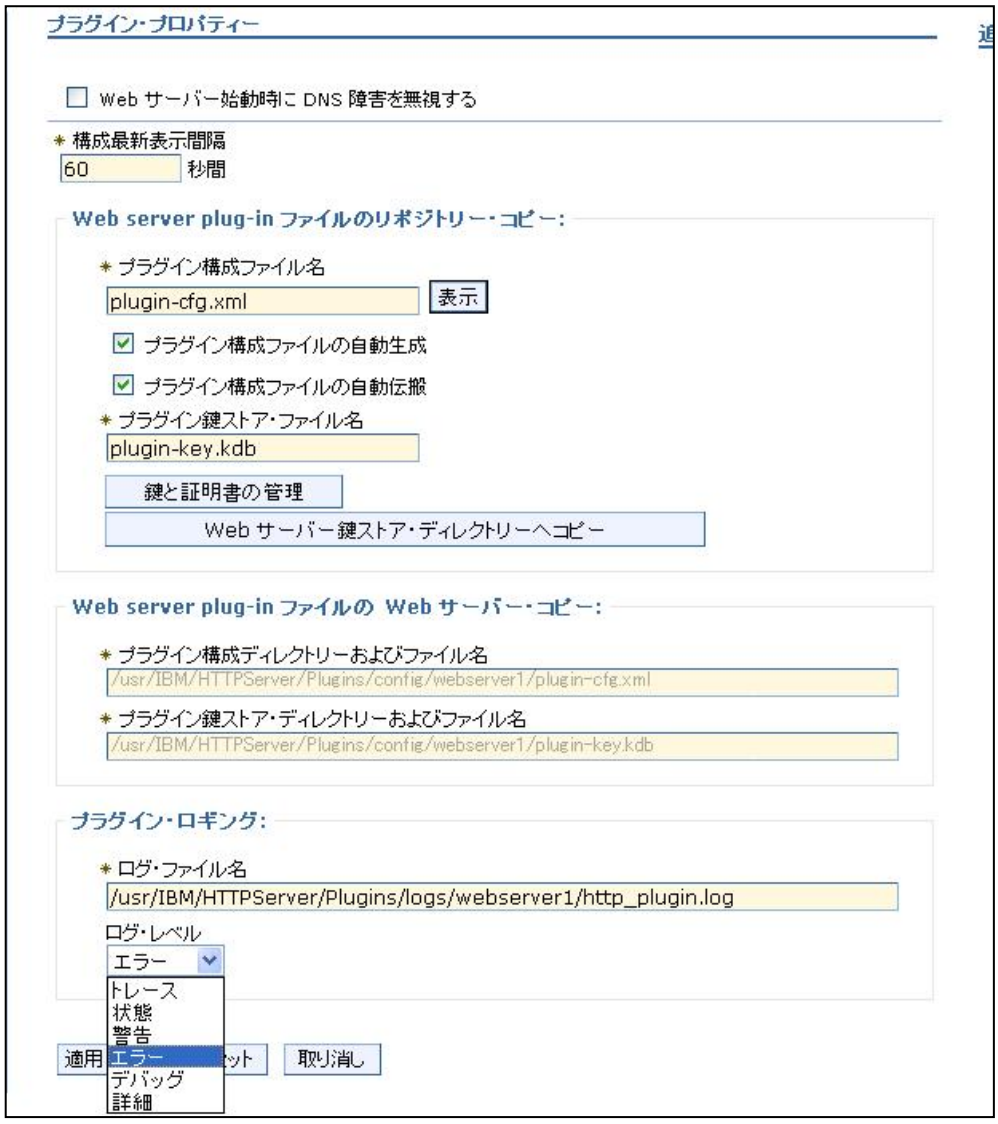

図 3-10 プラグイン・プロパティーの設定

3-2-5-2 プラグイン・ログ

プラグインの問題判別に有用なログとして、プラグイン・ログがあります。プラグイン・ログの設定変更は各 Web サーバーのプラグイン・プロパティー（**[サーバー]→[サーバー・タイプ]→[Web サーバー]→[*WebServer_name*]→[追加プロパティー]→[プラグイン・プロパティー]**（図 3-10参照））のプラグイン・ロギングの項目で設定します。ログ・ファイルのパス、ファイル名の設定とログ・レベルの設定を行うことができます。ログ・レベルは下記の 6 つのレベルから選択できます。

- トレース
- 状態
- 警告
- エラー
- デバッグ
- 詳細

エラーがデフォルト設定となり、トレースに設定することで情報量がもっとも多くなります。ログ・レベルをトレースに設定した場合、クライアントからのリクエストごとに非常に多くの情報をログに書き込み、ログ・ファイルのサイズが急速に大きくなり、また、パフォーマンスへの影響もあります。したがって、正常に稼働している本番環境では、ログ・レベルをトレースに設定すべきではありません。

3-2-5-3 静的コンテンツの Web サーバーへの配置

静的コンテンツ(HTML ファイルや GIF ファイル等)は Web サーバーに、動的コンテンツ(サーブレット、JSP 等)はアプリケーション・サーバーに分けて配置する方法を説明します。この設定は、アプリケーション・サーバーの設定ではなく、アプリケーションでの設定を変更します。これにより、プラグインに静的コンテンツの要求の場合にはアプリケーション・サーバーにリクエストを転送せず、Web サーバーで処理するリクエストと判断させます。設定変更の方法は、Rational Application Developer for WebSphere や RAD Assembly and Deploy で、Web デプロイメント記述子の画面で**[WebSphere 拡張機能を開く]**をクリックし、**[ファイルのサービス提供を使用可能にする]**のチェックボックスを無効にします(**図 3-11**参照)。この設定は、IBM 拡張の**[ibm-web-ext.xml]**ファイルに反映されます。デフォルトではファイル・サービスの使用は有効です。このように、ファイル・サービスの使用を無効化することにより、Web サーバーからアプリケーション・サーバーにリクエストを転送するかどうかを判断するプラグイン構成ファイルの URI の指定が、ワイルドカード(*)を使用した URI 指定ではなく、明示的なサーブレット・マッピングに従った指定になります。

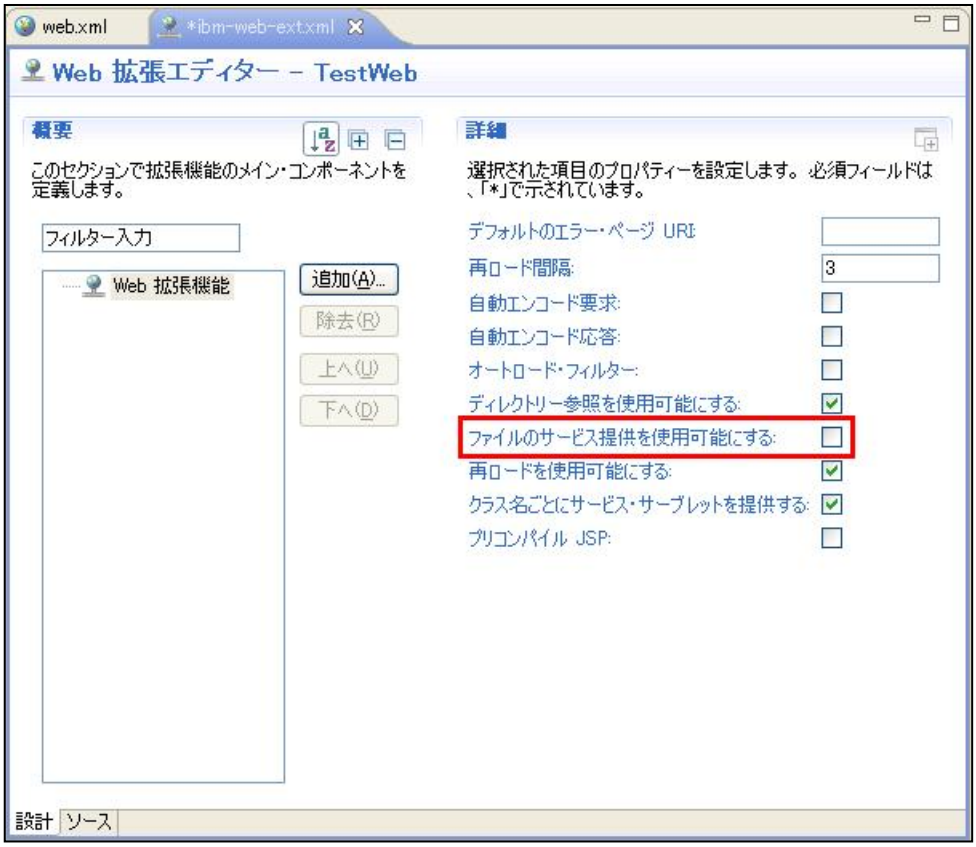

図 3-11 ファイル・サービスの設定

3-2-5-4 その他の設定の情報

プラグインの設定は、Web サーバー配下のプラグイン・プロパティーとアプリケーション・サーバー配下のプラグイン・プロパティーの 2 箇所にあります。これらを設定した後、プラグイン構成ファイルの再作成を行う必要があります。その他の設定に関しては下記の Information Center の記載を参照ください。

WAS V8.5 Information Center「Web サーバー・プラグインのプロパティー」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/uwsv_plugin_props.html
WAS V8.5 Information Center「Web サーバー・プラグインの要求ルーティング・プロパティー」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/uwsv_plugin_props3.html
WAS V8.5 Information Center「Web サーバー・プラグイン用の Application Server プロパティー設定」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/uwsv_plugin_props5.html

3-2-6 プラグイン構成ファイル

プラグイン構成ファイル（plugin-cfg.xml。**例 3-1 プラグイン構成ファイル（plugin-cfg.xml）の例（一部抜粋）** (p.6)」参照。）の主要なタグの内容について説明します。

表 3-4 プラグイン構成ファイルの主要なタグ

タグ	説明
VirtualHostGroup VirtualHost	クライアントがページリクエストを行うときに、HTTP Host ヘッダーに指定されるべき仮想ホスト名とポートの組を定義します。 仮想ホストを分けることにより、リクエストのホストヘッダーごとに異なるアプリケーションをマップすることができます。
UriGroup Uri	クライアントからのリクエストに含まれる URI は、Uri タグ内に定義されたすべての名前と比較され、仮想ホストの比較と合わせて、WAS が処理するリクエストか Web サーバーが処理するリクエストかを判断します。
Route	Route タグ内に定義された、VirtualHostGroup と UriGroup のマッチング結果から、どの ServerCluster のグループにリクエストを転送するか定義します。
ServerCluster Server	ServerCluster 内に含まれるクラスター・メンバーを定義します。WLM ポリシーやセッション・アフィニティーの機能で割り振り先を決定するときに使用します。
Transport	割り振り先のアプリケーション・サーバーを決定した後、SSL を使用した接続（HTTPS）を行うか HTTP で接続するかを定義します。

3-3 EJB コンテナーWLM

この節では、EJB コンテナーWLM の構成方法を説明します。EJB コンテナーWLM は EJB コンテナー・サービスを実行するアプリケーション・サーバーをクラスター化することにより実現できます。この節では、EJB コンテナーWLM を実行するコンポーネントである WebSphere ORB (Object Request Broker) WLM プラグインと EJB コンテナーの設定とその設定に応じた動きを説明します。

WAS ND では、クラスターを作成することにより、EJB の負荷分散は自動的に有効になります。ORB 内の WLM プラグインが複数のクラスター・メンバーへのリクエストの割り振りを実行します。

尚、EJB コンテナーWLM を使用する EJB クライアントは、EJB クラスター・メンバーとは別プロセスで稼働する必要があります。EJB WLM は EJB クラスター・メンバーと別プロセスで稼働する EJB クライアントに対して、負荷分散とフェイルオーバーの機能を提供します。EJB クライアントと EJB コンテナーが同一プロセス上で稼働する場合、プロセス・アフィニティー機能により、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。

下記の種類の EJB クライアントにおいて WLM が機能します。

- リクエスト EJB と異なるアプリケーション・サーバー上で稼働するクライアント（サーブレット、JSP、EJB）
- WebSphere クライアント・コンテナー上で稼働する Java アプリケーション
- WebSphere が提供する JRE（Java Runtime Environment）を使用する Java アプリケーション

3-3-1 EJB リクエスト・メカニズム

IBM ORB を使用した EJB クライアント（WAS の Web コンテナー、EJB コンテナー、クライアント・コンテナーまたは WebSphere が提供する JRE を使用する EJB クライアント）が EJB にアクセスするステップを説明します。

プログラム的には以下の手順で EJB クライアントから EJB を呼び出します。EJB3.0 以降ではアノテーションを使用したコーディングも可能ですが、ここではリクエストのステップを明確化するためにアノテーションを使用していません。

1). bootstrap プロセスの初期コンテキストを取り出します。
Web コンテナーまたは EJB コンテナー上のクライアントから、同一セル内の EJB にアクセスする場合には、下記のように、ローカル・プロセスのネーミング・サービスの初期コンテキストを取得します。

```
javax.naming.Context initialContext = new javax.naming.InitialContext();
```

セル外の EJB にアクセスする場合や、Java EE コンテナー外からアクセスする場合には、明示的にネームサーバー のアドレスを指定して初期コンテキストを取得します。

```
java.util.Properties props = new java.util.Properties();

props.put(Context.INITIAL_CONTEXT_FACTORY, “com.ibm.websphere.naming.WsnInitialContextFactory”);

props.put(Context.PROVIDER_URL, “corbaloc::host1.ise.ibm.com:2809”);

javax.naming.Context initialContext = new javax.naming.InitialContext(props);
```

2). クライアントは、JNDI 名を使用して、EJB ビジネスインターフェースを lookup します。

```
BeenThere beenthere = (BeenThere) initialContext.lookup(“java:comp/env/BeenThere”);
```

上記の例は Java EE コンテナー上で稼働する EJB クライアントの例です。ローカルの JNDI 名はデプロイメント・ディスクリプターまたは WAS 上でのアプリケーションの設定でグローバルの JNDI 名に置き換えられます。

3). EJB インスタンスのメソッドを実行します。

```
java.util.Properties envInfo = beenThere.getRuntimeEnvInfo();
```

上記の処理が実行されるときの、WAS のコンポーネントの呼び出し手順は図 3-12のようになります。

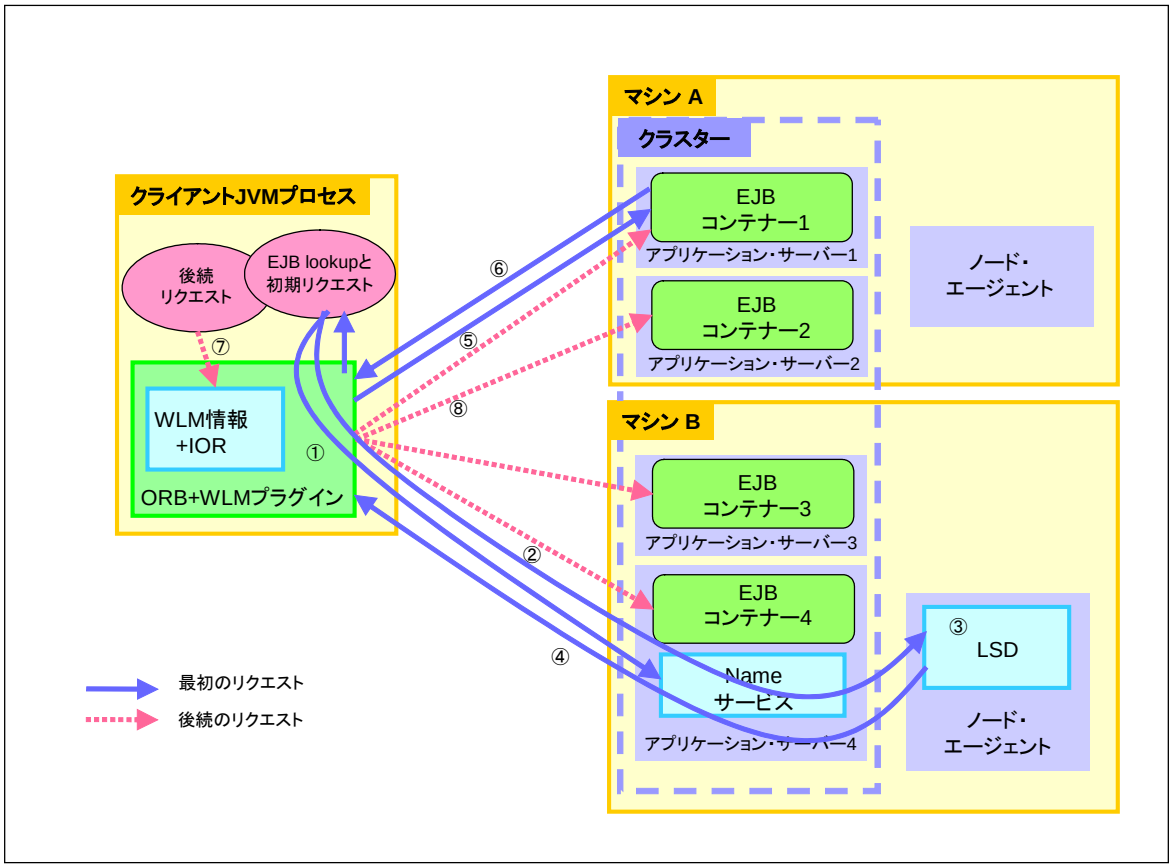

図 3-12 EJB 負荷分散処理フロー

① 初期コンテキストの作成のリクエストは、EJB クライアントのローカル JVM の ORB（Object Request Broker）を経由して、ネーミング・サービスを呼び出します。

② EJB ビジネスインターフェースの JNDI 名の lookup により戻されるオブジェクトは、間接的なオブジェクトの参照（indirect IOR（Interoperable Object Reference））であり、ネーミング・サービスが稼働するローカルのノード・エージェントの LSD（Location Service Daemon）をポイントします。

③ 最初の EJB リクエストは LSD を呼び出します。LSD は LSD が保持する WLM プラグインを使用してクラスター・メンバーの 1 つを選択します。

④ LSD は、その特定のクラスター・メンバーを表す直接的な参照（direct IOR）を ORB に返答します。

⑤ ORB は LSD が選択したクラスター・メンバーにリクエストを転送します。

⑥ EJB リクエストの結果とともに、WLM 情報（クラスターを構成するサーバー・グループの情報）がリプライ情報に含まれます。この WLM 情報は WLM プラグインに保管され、後続の EJB リクエストでは、その WLM 情報が使用されます。

続いて、後続のリクエスト呼び出しの手順は以下のようになります。

⑦ EJB クライアントからの後続のリクエストも、ORB を経由して行われます。
⑧ ORB は、リクエストを処理する EJB の参照を WLM プラグインに問い合わせます。ここで、負荷分散のルールにしたがって EJB の参照が戻されます。ORB はリモートの EJB クライアントのメソッドを呼び出します。

IBM ORB を使用しないクライアントの場合、クライアント側では WLM 情報を保持できないので、LSD のみがどの EJB コンテナーにリクエストを送信するか決定します。上記手順④の直接的な参照がその EJB クライアントからの後続のリクエストでも常に使用されます。

3-3-2 負荷分散ポリシー

WAS ND は EJB 負荷分散のポリシーとして以下の 4 つのルールによって割り振り先が決定されます。

- 重み付きラウンドロビン
- ローカルを優先
- プロセス・アフィニティー
- トランザクション・アフィニティー

上記 4 つのルールのうち、「重み付きラウンドロビン」のポリシーよりも「ローカルを優先」、「プロセス・アフィニティー」、「トランザクション・アフィニティー」のポリシーが優先されます。

以下で、それぞれの負荷分散のポリシーの動きとその設定方法を説明します。

3-3-2-1 重み付きラウンドロビン

EJB コンテナーWLM は Web コンテナーWLM と同様に、クラスター・メンバー設定におけるウェイトに基づいた重み付きラウンドロビンでリクエストを割り振ります。EJB コンテナーWLM ではクラスター単位でルーティング・テーブルを持ち、「ローカルを優先」としている場合は同じクラスターであっても、ローカルに割り振るためのルーティング・テーブルと、リモートに割り振るためのルーティング・テーブルを別々に管理します。これにより、EJB クライアントと同じマシンの全ての EJB コンテナーに障害が発生し、別マシンの EJB コンテナーにリクエストが割り振られた場合も WLM が機能します。

ルーティング・テーブルの値は、リクエストを割り振るたびに 1 減じられます。値が 0 または負のサーバーへは新しいリクエストは割り振られません。Web コンテナーWLM と同様に、アフィニティーが機能している場合には、0 または負のサーバーへも割り振りが行われます。しかし、Web コンテナーWLM と異なり、すべてのクラスターの値が 0 以下になり、値のリセットが行われるときには、設定した重みの値そのものが加算されます。Web コンテナーWLM では、設定した重みの倍数の値が加算され、すべてのメンバーの値が正の値にリセットされましたが、EJB コンテナーWLM では、設定した重みの値そのものが加算され、リセットされた後も、負の値を持つメンバーが存在することもありえます。

クラスター・メンバーの重みは、Web コンテナーWLM での設定と同じ設定が使用され、設定方法も同様です。管理コンソールでクラスター・メンバーのプロパティー（[サーバー]→[クラスター]→[WebSphere Application Server クラスター]→[*Cluster_name*]→[追加プロパティー]→[クラスター・メンバー]→[*ClusterMember_name*] (図 3-7参照)）で確認・変更することができます。また、クラスターの新規作成やクラスター・メンバーの追加時にも変更することができます。デフォルトの重みは 2 です。有効な値の範囲は 0 から 100 の整数値です。

3-3-2-2 ローカルを優先

ローカルを優先の設定はオン・オフを切り替えることができます。この設定を有効にすると、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ます。つまり、ローカル・ノードのすべて

のクラスター・メンバーが使用できない場合以外は、WLM プラグインは、同一ノードで稼働する EJB コンテナーへリクエストを割り振ります。

「ローカルを優先」を設定することにより、EJB クライアントと EJB が同一ノードに配置されている場合、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。

同一ノード上の EJB コンテナーが全て使用不可の場合には、他のノード上の EJB コンテナーへとリクエストは割り振られます。ローカル・ノードの EJB が使用可能になった場合は、リクエストはまたローカルの EJB に割り振られますが、トランザクション中のリクエストはそのままリモートに割り振られます。

ローカルを優先の設定を変更・確認はクラスターの設定ページで行います。管理コンソールで**[サーバー]→[クラスター]** →[WebSphere Application Server **クラスター**]→[*Cluster_name*] (**図** 3-13参照)を選択し、**[ローカルを優先]**のチェックボックスでオン・オフを設定します。デフォルトの設定はオンです。設定を反映するために、設定変更を行ったクラスター・メンバーの再起動が必要です。

図 3-13 ローカルを優先の設定

尚、プロセス・アフィニティーとトランザクション・アフィニティーはローカルを優先の設定より、つねに優先されます。

3-3-2-3 プロセス・アフィニティー

ウェイトやローカルを優先の設定に関係なく、EJB クライアントと EJB コンテナーが同一アプリケーション・サーバーで稼働する場合、リクエストは EJB コンテナーWLM による割り振りではなく、プロセス・アフィニティー機能により必ず同一アプリケーション・サーバーの EJB コンテナーに割り振られます。

3-3-2-4 トランザクション・アフィニティー

EJB コンテナーWLM により割り振られるリクエストの中でも、同一トランザクション・スコープ中のリクエストに対しては必ず前回と同じ EJB コンテナーへとリクエストは割り振られます。トランザクションの種類は、ユーザー管理トランザクション、コンテナー管理トランザクションのどちらでも有効になります。トランザクション・アフィニティーはすべてのルーティング・ポリシーに優先します。

3-3-3 フェイルオーバー

EJB クライアントが EJB リクエストを呼び出した場合、そのリクエストは、クラスター・メンバーのいずれかで処理されます。もし、そのクラスター・メンバーが使用不可だった場合、そのリクエストは自動的に他のクラスター・メンバーに再割り振りが行われます。

3-3-3-1 ブートストラップ・フェイルオーバー

EJB の高可用性を検討する場合には、EJB サーバーの冗長化だけでなく、EJB クライアントの冗長化やフェイルオーバーも検討する必要があります。

EJB クライアントは、最初に、EJB home インターフェース（EJB2.x の場合）またはビジネスインターフェース(EJB3.0 の場合)を lookup します。以下の 2 つの場合があります。

クラスター化された EJB クライアントからの呼び出し

クラスター化された Web コンテナーまたは EJB コンテナー上の EJB クライアントからの呼び出しで、サーブレットや EJB から EJB 呼び出しを行う場合です。この場合、一般的に EJB クライアントはデフォルトの URL で、自分自身のアプリケーション・サーバーをブートストラップ・サーバーとして使用します。ブートストラップに失敗した場合、EJB クライアントからの呼び出しは失敗しますが、EJB クライアントは冗長化されていますので、EJB クライアントの呼び出しをフェイルオーバーすることにより、高可用性が達成されます。

スタンドアロンの EJB クライアントからの呼び出し

スタンドアロンの Java クライアントや Java EE クライアントの場合、EJB クライアントをクラスター化することはできません。この場合、EJB クライアントがブートストラップ・サーバーとして一つのサーバーだけを指定していた場合、ブートストラップ・サーバーがダウンしているときは EJB クライアントからの呼び出しは失敗します。以下の例のようにブートストラップ・サーバーとして複数のサーバーを指定することが推奨されます。

```
java.util.Properties props = new java.util.Properties();

props.put(Context.INITIAL_CONTEXT_FACTORY, “com.ibm.websphere.naming.WsnInitialContextFactory”);

props.put(Context.PROVIDER_URL, “corbaloc::host1.ise.ibm.com:9810, :host2.ise.ibm.com:9810”);

javax.naming.Context initialContext = new javax.naming.InitialContext(props);
```

3-3-3-2 EJB コンテナー・フェイルオーバー

EJB コンテナーのフェイルオーバーは、EJB クライアントに存在する ORB の機能とノード・エージェントの LSD が保持するルーティング・テーブルによって提供されます。フェイルオーバーはルーティング・テーブルの修正と、クライアント ORB によるリクエストの再割り振りによって実現されます。

デプロイメント・マネージャーがダウンした場合

WAS V6 以降では、HA マネージャーが、単一のデプロイメント・マネージャー上ではなく、すべてのアプリケーション・サーバー、ノード・エージェント、デプロイメント・マネージャー上で稼働することにより、WLM 情報などのキーとなるサービスを実行します。HA マネージャーの詳細は「3-5 HA マネージャー」を参照ください。HA マネージャーの機能により、EJB WLM に関して、デプロイメント・マネージャーは単一障害ポイントになりません。

ノード・エージェントがダウンした場合

ノード・エージェントは、EJB WLM に関連するサービスとして LSD サービスを提供しています。LSD サービスは、WLM ルーティング情報を EJB クライアントに供給します。

EJB クライアントが最初にリクエストを行ったときに WLM 情報は取得されます。WLM ルーティング情報を一度取得した後にノード・エージェントがダウンした場合、ルーティング情報は EJB クライアントにキャッシュされるので、影響はありません。EJB クライアントが WLM ルーティング情報を取得する前にノード・エージェントがダウンすると、WLM ルーティング情報を取得できないので、「スタンドアロンの EJB クライアントからの呼び出し」などの方法を使用して、他の LSD サービスに WLM ルーティング情報を要求する必要があります。

クラスター・メンバーがダウンした場合

ルーティング・テーブル情報を取得する前の最初のリクエストでエラーが発生した場合には、ORB がエラーを受け取り、LSD に対して再度リクエストの割り振り先 EJB コンテナーを問い合わせます。EJB クライアントがルーティング・テーブル情報を取得した後に、EJB コンテナーに障害が発生した場合の挙動は大きく 2 つに分けられます。EJB クライアントが EJB コンテナーと同一セル内に存在し障害を HA マネージャーから通知される場合と、EJB クライアントがセル外に存在、または HA マネージャーでの障害検知前にリクエストを送信して、EJB クライアントがリクエストを送信することで障害を検知する場合です。

EJB リクエストによる障害検知

EJB リクエストを送信し、エラーが発生した場合には、WLM クライアントがエラーを受け取り、ルーティング・テーブルのサーバー・リストから障害の発生したサーバーを除去し、ルーティング・アルゴリズムにしたがって、異なるサーバーにリクエストを再送します。また正常なリクエストにおいて、WLM クライアントが保持するルーティング・テーブル情報が、EJB コンテナーが保持するルーティング・テーブル情報よりも古い場合には、新しいルーティング・テーブル情報が EJB レスポンスと同時に EJB クライアントに通知されます

EJB クライアントで障害を検知した場合は、検知したクライアントのみが障害を認識し、リクエストを割り振らなくなりますが、その情報は他の EJB クライアントには伝えられません。

EJB クライアントは EJB コンテナーの障害を検知すると、その EJB コンテナーへのリクエストは次のパラメーターで指定した秒数後に、再度割り振りを試みます。デフォルトは 300 秒です。

com.ibm.websphere.wlm.unusable.interval

設定方法の詳細は下記の Information Center の記載を参照ください。

WAS V8.5 Information Center「ワークロード管理構成の調整」

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/trun_wlm_tuning.html

ネットワーク障害を HA マネージャーが検知する前にリクエストを送信した場合の挙動は、メソッドコールをする前にネットワーク障害が発生した場合と、メソッドコール中にネットワーク障害が発生した場合で挙動は異なります。

メソッドコールをする前にネットワーク障害が発生した場合、EJB クライアント側の OS の TCP/IP タイムアウトにしたがい、タイムアウト経過後にその接続を失敗とみなします。その後、例外が発生するのではなく、他のサーバーにリクエストを割り振ります。TCP/IP タイムアウトは、AIX の場合、デフォルトは 75 秒（tcp_keepinit=150）で次のコマンドで変更可能です。

no -o tcp_keepinit = *180*（90 秒に変更）

OS のタイムアウトだけでなく、EJB WLM プラグイン独自の接続タイムアウトのパラメーターである CORBA 接続タイムアウト値でも制御できます。WAS8.0 以降のデフォルトの設定値は 10 秒に設定されています。

com.ibm.CORBA.ConnectTimeout

設定は、EJB クライアントがアプリケーション・サーバー上で稼働する場合には、ORB のカスタム・プロパティーで設定します。設定方法の詳細は下記の Information Center の記載を参照ください。

WAS V8.5 Information Center「オブジェクト・リクエスト・ブローカーのカスタム・プロパティー」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/rorb_setg.html

尚、CORBA 接続タイムアウト値が OS のタイムアウト値より長い場合は、OS のタイムアウト値が優先されます。

メソッドコール中にネットワーク障害が発生した場合や、EJB コンテナーで処理が滞留した場合、CORBA リクエストタイムアウトの秒数経過後に障害とみなします。この場合リクエストのリトライは発生せず、org.omg.CORBA.NO_RESPONSE 例外が発生します。CORBA リクエストタイムアウトは次のパラメーターで指定します。デフォルト値は 180 秒です。

com.ibm.CORBA.RequestTimeout

設定は、EJB クライアントがアプリケーション・サーバー上で稼働する場合には、ORB のプロパティーで設定します。設定方法の詳細は下記の Information Center の記載を参照ください。

WAS V8.5 Information Center「オブジェクト・リクエスト・ブローカー・サービス設定」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/uorb_rsetg.html

メソッドコール中に EJB コンテナーのプロセス障害が発生した場合には、即座に org.omg.CORBA.COMM_FAILURE 例外が発生します。既にメソッドは実行されていますのでリトライはされません。

Stateful Session Bean フェイルオーバーのサポート

WAS V7.0 以降では Stateful Session Bean のフェイルオーバーをサポートしています。Stateful Session Bean のフェイルオーバーは、WLM 機能とデータ複製サービス（Data Replication Service（DRS と呼ばれることもあります）機能によって行われます。データ複製サービスは、アプリケーション・サーバー間の一般的なデータ複製を行うための WAS の内部コンポーネントです。Stateful Session Bean の複製だけでなく、HTTP セッションのメモリー間複製や動的キャッシュの複製でも使用されます。

Stateful Session Bean のフェイルオーバーの有効化と無効化の設定は管理コンソールより行います。フェイルオーバーを有効または無効にするターゲットの範囲に応じて下記の 3 つのレベルで設定を行うことができます。

- EJB コンテナー
- エンタープライズ・アプリケーション
- EJB モジュール

Stateful Session Bean フェイルオーバーを設定すると、Stateful Session Bean の Passivate のタイミングで複製が行われます。通常、Stateful Session Bean の Passivate のタイミングはデプロイメント・ディスクリプターibm-ejb-jar-ext.xml (EJB3.0)または ibm-ejb-jar-ext.xmi(EJB2.x)ファイルに設定します。以下の 3 種類の設定があります。

- ONCE
- TRANSACTION
- ACTIVITY_SESSION

Stateful Session Bean フェイルオーバーを設定すると、強制的に[TRANSACTION]に設定されます。[TRANSACTION]設定では、トランザクション開始時に Activate され、終了時に Passivate が実行されます。トランザクション中に障害が発生した場合は、複製は行われていませんので例外が発生します。

Stateful Session Bean のフェイルオーバーの設定を EJB コンテナーレベルで変更・確認する場合、管理コンソールで**[サーバー]→[サーバー・タイプ]→[WebSphere Application Server]→[アプリケーション・サーバー]→[*ApplicationServer_name*]→[EJB コンテナー設定] →[EJB コンテナー]**（図 3-14参照）を選択し、**[Stateful Session Bean のフェイルオーバーを使用可能にするメモリー間の複製]**のチェックボックスでオン・オフを設定します。デフォルトの設定はオフです。このチェックボックスは、複製ドメインを定義するまでオンにすることはできません。

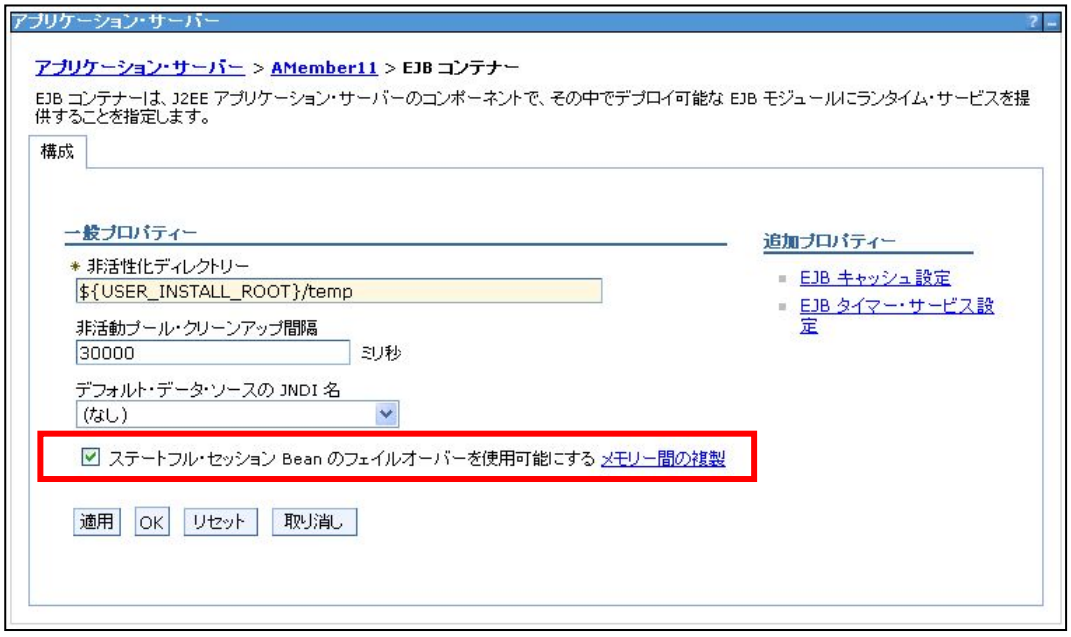

図 3-14 EJB コンテナーレベルでの Stateful Session Bean のフェイルオーバーの設定

Stateful Session Bean のフェイルオーバーの設定をアプリケーションレベルまたは EJB モジュールレベルで変更・確認する場合、管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]→[*Application_name*]→[Enterprise Java Bean プロパティー] →[ステートフル・セッション Bean のフェイルオーバー設定]**（図 3-15参照）または**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]→[*Application_name*]→[モジュール]→[モジュールの管理]→[*jarFile_name*]→[追加プロパティー]→[ステートフル・セッション Bean のフェイルオーバー設定]**を選択します。Stateful Session Bean のフェイルオーバーを使用可能にするには[メモリー間複製を使用して、ステートフル・セッション Bean のフェイルオーバーを使用可能にします。]のチェックボックスをオンにします。Stateful Session Bean のフェイルオーバーを使用不可にするには、このチェックボックスをオフにします。Stateful Session Bean のフェイルオーバーをオンにした場合の複製設定は 2 種類あります。アプリケーションレベルの設定では、[EJB コンテナーからの複製設定を使用します。]を選択した場合には、このアプリケーションに定義したすべての複

製設定が無視されます。つまり、EJB コンテナーレベルでの設定の上書きは行われません。[アプリケーション複製設定の使用]を選択した場合には、EJB コンテナーレベルでの設定が上書きされます。同様に、EJB モジュールレベルの設定では、[アプリケーションまたは EJB コンテナー複製設定の使用]を選択した場合には、この EJB モジュールに定義したすべての複製設定が無視されます。つまり、EJB コンテナーレベルでの設定とアプリケーションレベルでの設定の上書きは行われません。[EJB モジュール複製設定の使用]を選択した場合には、EJBコンテナーレベルでの設定とアプリケーションレベルでの設定が上書きされます。

図 3-15 アプリケーションレベルでの Stateful Session Bean のフェイルオーバー設定

尚、**[EJB コンテナーからの複製設定を使用します。]**を選択して、Stateful Session Bean の複製を行うには、メモリー間複製を EJB コンテナー・レベルで構成する必要があります。そうしないと、このパネルの設定は、サーバー始動中に EJB コンテナーによって無視され、EJB コンテナーは Stateful Session Bean フェイルオーバーがこのアプリケーションでは使用可能ではないことを示すメッセージがログに記録されます。

3-3-4 バックアップ・クラスター

バックアップ・クラスターの定義により、EJB クラスターの全体障害時に稼働するクラスターを設定することが可能です。バックアップ・クラスターは EJB リクエストに対してのみ有効です。バックアップとして指定するクラスターは別のセルにある必要があり、プライマリー・クラスターと同じクラスター名、アプリケーション、リソース名を使用しなければなりません。また、2 つのクラスターで情報のやり取りを行うために、各々のクラスターが属するコア・グループにてブリッジ設定を行う必要があります。

バックアップ・クラスターに対するリクエストはプライマリー・クラスターのメンバーが復活すると、自動的にプライマリーに割り振りが行われるようになります。

バックアップ・クラスターの作成方法の詳細は下記の Information Center の記載を参照ください。
WAS V8.5 Information Center「バックアップ・クラスターの作成」

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/trun_wlm_backup.html

3-4 Intelligent Management 環境

これまでの Web システムでは混雑時にあわせたサーバーを用意する事が一般的であり、想定以上のリクエストが来た場合はサーバーの過負荷状態を防ぐために前段の負荷分散装置で接続数等により流量制御を行うことで対応してきました。この方法ではアプリケーションやユーザー毎に優先度をつけた管理は行えず、混雑時にはどのサービスについても均等に遅延、または Sorry 振り分けが発生していました。

WAS V8.5 では、Intelligent Management 機能を使用することでこれらの問題を解決することができます。オートノミック要求フロー・マネージャー（ARFM）はサーバーの負荷状況やリクエストの応答時間をもとに流量制御を行うことができます。混雑時には重要度の低いリクエストをキューイングし、より重要度の高いリクエストを優先的に処理させます。リクエストの重要度や目標応答時間はサービス・ポリシーで設定します。動的ワークロードマネージメント（DWLM）はアプリケーション・サーバーの負荷状況から動的に重み付けを行い、ARFM が処理可能と判断したリクエストを割振ります。また、アプリケーション配置コントローラー（APC）はサーバーの負荷状況に合わせアプリケーション・サーバーを動的に起動し、サービスを適切なレベルに維持します。

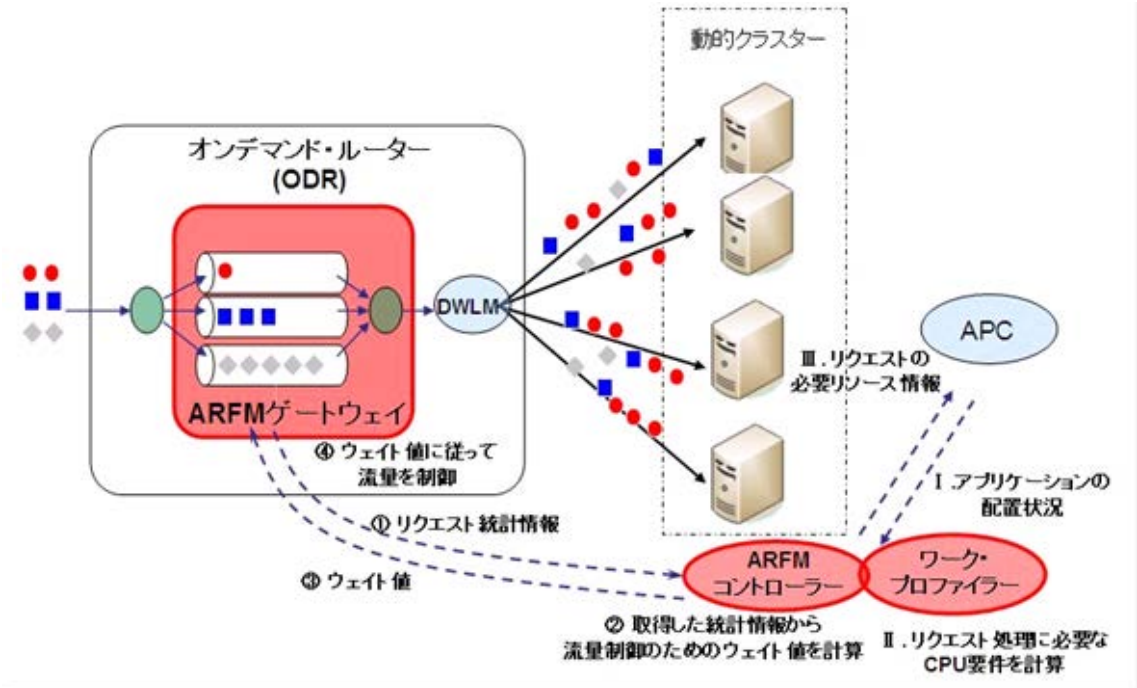

図 3-16 Intelligent Management Pack 概要

3-4-1 DWLM

DWLM はもともと WebSphere Virtual Enterprise（以下、WVE）の機能でしたが、WVE が WAS ND V8.5 に統合されたため、WAS ND V8.5 でこの機能を使用することが可能になりました。

DWLM ではシステム・リソースの使用状況に基づいて、ビジネス目標に対応できるよう最適な重み付けを行い、その動的に設定される重みに基づいた重み付きラウンドロビン方式でルーティングを行うことが可能です。この機能は静的クラスターおよび動的クラスターに適用が可能です。

3-4-1-1 DWLM の機能

DWLM は関連コンポーネント(3-4-2 ARFMを参照)から取得した情報をもとにリクエストのルーティングを行います。ウェイト値の計算は周期的に行われ、デフォルトでは 20 秒毎に実施されます。ウェイト値の計算には、サーバーに送られる要求に対する平均サービス時間と、各ノードのプロセッサー使用率が使用されます。一般的にノードのプロセッサー使用率が低い場合は負荷の均等分散が優先され、ノードのプロセッサー使用率が高くなるとパフォーマンスの均一化が優先されます。この比率は動的クラスターのカスタム・プロパティー"equalCPUFactor"で調整することも可能です。

3-4-1-2 DWLM の設定方法

この設定は、[サーバー]→[クラスター]→[cluster_name]→[動的ワークロード管理 (DWLM)]もしくは[サーバー]→[動的クラスター]→[cluster_name]→[動的ワークロード管理 (DWLM)]のページ（図3-17参照)で変更することができます。

図 3-17 DWLM 設定

- 動的ワークロード管理 (DWLM)

このチェックボックスを選択することで、システムが必要に応じて動的に重みを変更することのできる機能が有効となります。動的クラスターではデフォルトでオン、静的クラスターではデフォルトはオフです。

また、DWLM の動作は動的クラスターのカスタム・プロパティーで変更が可能です。

- equalCPUFactor

このカスタム・プロパティーを使用することで、サーバーの動的重みを計算する際に、サーバーに送られる要求の平均サービス時間、ノードのプロセッサー使用率の 2 つの指標について、どちらを優先させるかまたはそのバランスについて指定することができます。

平均サービス時間を優先させる場合は 0 を、ノードのプロセッサー使用率を優先させる場合は 1 を指定します。両方の指標を混用する場合は 0 から 1 までの少数値を設定します。0.4 に設定した場合、ノードのプロセッサー使用率を 0.4、平均サービス時間を 0.6 の割合で使用します。

このカスタム・プロパティーはセルまたは動的クラスター単位で設定を行い、デフォルト値は、非仮想化環境では 0、仮想化環境では 1 となります。

WAS V8.5 Information Center「動的クラスター・カスタム・プロパティー」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/rwve_odccustprop.html

3-4-2 ARFM

オートノミック要求フロー・マネージャー（ARFM）は、リクエストをサービス・ポリシーに基づきキューイングし、優先制御や流量制御を行うコンポーネントです。ARFM はゲートウェイ、コントローラー、ワーク・プロファイラーという 3 つのコンポーネントから構成されています。（図 3-16参照）

3-4-2-1 ARFM の機能

AFRM の 3 つのコンポーネントについて説明します。

ゲートウェイ

ゲートウェイは、サービス・ポリシーごとにキューを自動生成し、リクエスト・フローをインターセプトしてキューイングします。キューごとに最大接続数を持ち、同時に送り出すリクエスト数を制限します。ARFM 構成プロパティーの集計期間（図 3-18参照）毎に集計した統計データをブロードキャストします。

コントローラー

コントローラーは、ゲートウェイによってキューイングされたリクエスト・フローの最適化を担うためのコンポーネントであり、それまでに実行された処理の統計情報からゲートウェイからディスパッチする際のウェイト値を計算します。ARFM 構成プロパティーの制御サイクルの最低期間（図 3-18参照）毎にアクティブとなり、ゲートウェイ、APC（3-4-3 APC を参照）を制御します。

ワーク・プロファイラー

ワーク・プロファイラーは、APC からアプリケーションの配置状況を受け取り、動作ポリシーやリクエスト統計情報によりリクエストの処理に必要な CPU 要件を計算します。

3-4-2-2 ARFM の設定方法

この設定は、[動作ポリシー]→[オートノミック・マネージャー]→[オートノミック要求フロー・マネージャー]のページ（図 3-18参照）で変更することができます。

図 3-18 ARFM 設定

以下は InfoCenter 内でのパラメーター名を使用しています。管理コンソール上での表記が異なる場合括弧書きでパラメーター名を記載します。

- 集計期間（集約期間）

ARFM ゲートウェイごとに集計された統計データをブロードキャストする期間を指定します。デフォルト値は 5 秒です。

集計期間を設定する場合、十分な数のパフォーマンス・サンプルの収集に対応できる大きさに設定します。ゲートウェイでは、要求ごとにサンプルが収集され、適切な統計的測定を行うには数百のサンプルが必要となります。集計期間は1秒から1時間の間で設定が可能ですが、集約期間の値を小さく設定しすぎると信頼できないパフォーマンス測定メトリックとなります。反対に集約期間の値を長く設定しすぎると、制御設定の再計算の回数が少なくなり突然トラフィックが増えた場合などに製品の反応が悪くなります。

- 制御サイクルの最低期間(コントロール・サイクルの最小長)

このパラメーターは、コントローラーがアクティブになる頻度を定義します。デフォルト値は 59 秒です。ARFM コントローラーをアクティブにするプロセスは、新規統計が 1 つのゲートウェイから受信され、かつ、過去にアクティブになってから経過した時間が制御サイクルの最低期間以上、または起動時などコントローラーが今まで一度もアクティブになったことがない場合に開始されます。

- 平滑化ウィンドウ

このパラメーターは着信ゲートウェイ統計に対するコントローラーのリアクションの感度を定義します。ARFM コントローラーは直近の統計レポートの平均を使用しますが、平滑化ウィンドウで結合するレポートの数を制御します。デフォルト値は 12 です。

平滑化ウィンドウを低く設定するとコントローラーの感度が上がりますが、低すぎる場合データのノイズや異常に敏感に反応してしまうため注意が必要です。

平滑化ウィンドウと集計期間の積が実際の制御サイクルの最低期間とほぼ同じになるように設定することが推奨されます。

- キューの最大の長さ(最大キュー長)

このパラメーターは、各ゲートウェイのキューの長さを定義します。デフォルト値は 1000 です。
キューを短く設定した場合は、短期間にトラフィック・バーストが起こることによって、要求が拒否される可能性が高くなります。キューを長く設定した場合は、要求がキューに長く留まることになります。 キューに入れられた要求はメモリーを消費します。

- メモリー過負荷防止

各アプリケーション・サーバーで使用されるヒープ・サイズの最大パーセンテージを指定します。デフォルトは 100%です。

- CPU の最大使用率(最大 CPU 使用率)

ARFM は優先順位付け機能に加えて過負荷防止機能を持っています。ミドルウェア・ノードの最大 CPU 使用率を指定し、ノードの CPU 使用率がこのパーセンテージを超えるとそのノードは過負荷と見なされます。デフォルトは 80%です。

また、アプリケーション・サーバーの CPU 使用率が 50%を超えると DB が過負荷になることが分かっているケースでは 50%に設定するといったことが可能です。

- 要求拒否ポリシー

拒否ポリシーを設定することで、CPU 過負荷時にメッセージを拒否することが可能です。拒否ポリシーは以下の 3 つのオプションと拒否しきい値設定します。

- 着信メッセージの拒否なしでは、タイムアウトになったりサービス・ポリシーの目標の応答時間しきい値に違反する可能性があることには関係なく、すべてのメッセージをキューに入れることができる

この拒否ポリシーでは、着信メッセージを拒否しません。このオプションがデフォルトになります。

- メッセージの予測応答時間がそのサービス・ポリシーの目標の応答時間しきい値を超える場合、既存のダイアログまたはセッションの一部でない着信メッセージを拒否する

この拒否ポリシーでは、サービス・ポリシーの目標応答時間を超えるメッセージを拒否します。

- メッセージの予測応答時間がそのサービス・ポリシーの目標の応答時間しきい値を以下のパーセンテージよりも多く超える場合、既存のダイアログまたはセッションの一部でない着信メッセージを拒否する

この拒否ポリシーでは、拒否しきい値をパーセンテージで指定します。このオプションを選択した場合、サービス・ポリシー応答時間しきい値を基準に設定したパーセンテージを超えるメッセージを拒否します。

3-4-3 APC

アプリケーション配置コントローラー（APC）は、ARFM コントローラー及びワーク・プロファイラーからの情報に従い、アプリケーションを動的に配置、またアプリケーション・サーバー・インスタンスを動的に起動・停止するコンポーネントです。

3-4-3-1 APC の機能

APC は、サーバーの負荷状況を取得し ARFM に通知します。ARFM は、サーバーの追加が有効と判断した場合などに通知を行い、APC はこの通知をもとにアプリケーションの配置変更を決定し、その配置変更に基づいて動的クラスター・メンバーの起動や停止を行います。

APC は、以下の条件でサーバーを起動します。

- 動的クラスターに定義されたアプリケーション・インスタンスの最小数が満たされない場合。
- 要求が、ODR により、非アクティブな動的クラスターに対してルーティングされた場合。
- インスタンス追加が有効であると ARFM によって判断された場合。

APC は、以下の条件でサーバーを停止します。

- ノードにメモリー制約がある場合
  APC は、動的クラスターに対する最小、またはその動的クラスターに必要な容量とシステムのプロセッサー制約およびメモリー制約を識別します。あるノードの使用可能メモリーが低くなると、APC は、インスタンスを停止して、そのノードがスワッピングしないようにします。
- 動的クラスターが、アプリケーションの遅延スタートおよび積極的なアイドル停止用に構成され、その動的クラスターに対する要求がない場合
  アプリケーションの遅延スタートおよび積極的なアイドル停止用に構成された動的クラスターに対する要求がない場合、APC は、そのクラスターのインスタンスを停止し、非アクティブな動的クラスターのリソース消費をなくそうとします。

3-4-3-2 APC の設定方法

この設定は、[動作ポリシー]→[オートノミック・マネージャー]→[アプリケーション配置コントローラー]のページ（**図 3-19参照**）で変更することができます。

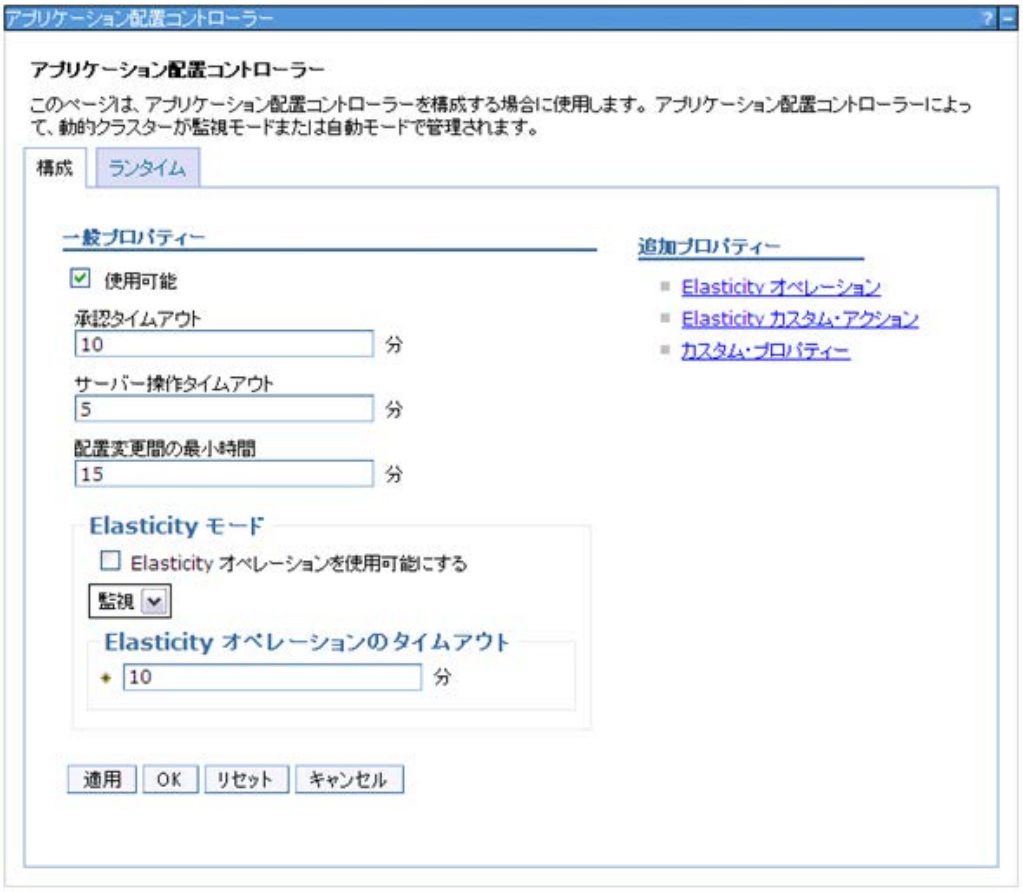

図 3-19 APC 設定

- 使用可能

APC を使用可能または使用不可にします。デフォルトは使用可能です。APC を使用不可に設定することは、すべての動的クラスターを静的クラスターに変更することと同じです。

- 承認タイムアウト

動的クラスターを監視モードで稼動させている場合、APC は、タスクを作成しますが、承認を待ってから変更を行います。APC は、タイムアウトになったランタイム・タスクをユーザーがリジェクトしたタスクとして扱います。

- サーバー操作タイムアウト

APC が操作の完了を待機する時間を設定します。サーバーの開始に失敗した場合、APC は問題が修正されて手動で開始されるまでそのサーバーの開始は試みません。

maintenanceModeOnOperationFail カスタム・プロパティーを true に設定することで、開始に失敗したサーバーを保守モードにすることが可能です。

また、unsetMaintenanceModeAfterStar カスタム・プロパティーを true に設定することで、正常に開始したサーバーの保守モードを解除することが可能です。

InfoCenter「アプリケーション配置カスタム・プロパティー」

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/rwve_odplacecustprop.html

- 配置変更間の最小時間

頻繁にアプリケーションの配置を再調整するように APC を構成すると、動的クラスターのサイズを再調整して当初得られたパフォーマンス向上の効果がオーバーヘッドの追加のために打ち消されます。配置変更間の最小時間を設定することで、前回の変更が完了するかタイムアウトしてから次の変更まで APC が待機する時間を設定します。

- Elasticity オペレーションを使用可能にする

このパラメーターを選択することで Elasticity オペレーションが使用可能になります。

Elasticity モードとは、動的にノードの追加や削除を行う設定です。サービス・ポリシーを満たすことが出来ず、全ての使用可能なサーバーが開始済みである場合にノードを追加したり、サービス・ポリシーを満たしている場合に不要なノードを削除したりすることも可能です。

Elasticity オペレーションのアクションについては追加プロパティーで構成します。

WAS V8.5 Information Center「Elasticity モード」

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/cwve_elasticity.html

以下は追加プロパティーの設定です。

- Elasticity オペレーション

ノードの追加を行う Add とノードの削除を行う Remove の二つのオペレーションが用意されています。

アクションの選択では、"事前定義 Elasticity オペレーション・アクション"と"カスタム Elasticity オペレーション・アクション"が選択できます。

Elasticity モードは IBM Workload Deployer（IWD）、PureApplication との親和性が高く、"事前定義 Elasticity オペレーション・アクション"は IWD など仮想化環境と組み合わせることで使用が可能です。"カスタム Elasticity オペレーション・アクション"は次に説明する Elasticity カスタム・アクションで設定したアクションを選択できます。

- Elasticity カスタム・アクション

ここで Elasticity オペレーションのアクションを設定することができます。

ノードの追加であれば、プロファイルの作成、セルへのノードの追加、動的クラスターのメンバーシップ・ポリシーに適合するように追加したノードの設定の書き換えといったアクションが必要になります。

WAS V8.5 Information Center「Elasticity モードの構成」

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/twve_cfgelastic.html

3-4-4 サービス・ポリシー

サービス・ポリシーは、作業をグループ化し、そのグループに対して重要度及び目標応答時間を定義したものです。サービス・ポリシーの概念を理解するためには、作業クラス（WorkClass）、トランザクション・クラスの理解が必要になります

作業クラスは、アプリケーション・サーバーによって実行される作業をグループにまとめたものです。ここで言う作業とは HTTP 要求、SIP メッセージ、IIOP 呼び出し、または JMS メッセージを指します。各作業クラスには、作業の処理方法を決定するためのルール・セットを指定し、関連するトランザクション・クラスを指定します。HTTP の場合は、リクエストを URI によってグループ化したもの、と言い換えても良いかもしれません。作業クラスはアプリケーション・モジュールまたはオンデマンド・ルーターに関連付け

られたパターンをもとに決定されます。デフォルトで、"/*"というパターンが存在しているため、全ての要求がこのデフォルトの作業クラスに一致することになります。

トランザクション・クラスは、作業クラスをさらにグルーピングしたもので、作業クラスはこのトランザクション・クラスを介してサービス・ポリシーに関連付けられます。

サービス・ポリシーは複数のトランザクション・クラスを持つことができますが、トランザクション・クラスは単一のサービス・ポリシーに紐付きます。また、トランザクション・クラスは複数の作業クラスを持つことができますが、作業クラスは単一のトランザクション・クラスに紐付きます。

3-4-4-1 サービス・ポリシーの機能

サービス・ポリシーには、重要度と目標応答時間を定義します。ARFM ゲートウェイではサービス・ポリシー毎にキューを持ちます。よって、サービス・ポリシーに設定された重要度と目標応答時間がキューイングを行う際の指標となります。以下にそれぞれの設定指針を記述します。

目標応答時間は、混雑時（既に CPU やメモリなどのリソースがいっぱいで、割振っても処理負荷がオーバーしてしまうようなケース）に最終的にどれくらいの応答時間までを許容できるか、という観点で設定を行います。この応答時間目標は、過負荷防止のため ODR がキューイングした際、リクエストが ODR でどれだけ待たせることが可能かといった指標にもなりますので、過負荷防止の発生する負荷状況時のサービス時間を基準にして設定する必要があります。逆に、通常時のアプリケーション処理時間を設定してしまうと、どのリクエストをキューイングしてもサービス目標に違反することにつながるため、キューイングについて適切な判断が出来なくなります。また、バックエンドのサービス時間も考慮に入れ、ARFM がサービス目標に違反せずキューイングできる値を設定する必要があります。

また、ARFM は同じサービス・ポリシー内で計測されたサービス時間の実績をもとに応答時間を予測し、一定時間に処理できるリクエスト数を決定します。そのため、1 つのポリシーに分類されるトラフィックの応答時間、CPU 使用率は同程度とする必要があります。

重要度は混雑時などリソースに制約がある場合、どの処理についてパフォーマンスの低下を容認できるかを決める際の指標として使用されます。より重要度が低いサービス・ポリシーに該当するリクエストから、キューイングを行います。

3-4-4-2 サービス・ポリシーの設定

サービス・ポリシーの設定は、[動作ポリシー]→[サービス・ポリシー]→[service_policy_name]のページ（図 3-20 参照）で設定します。サービス・ポリシーを新規に作成する場合は、[動作ポリシー]→[サービス・ポリシー]のページで[新規]をクリックします。

以下に、設定可能な項目に関する説明を記述します。

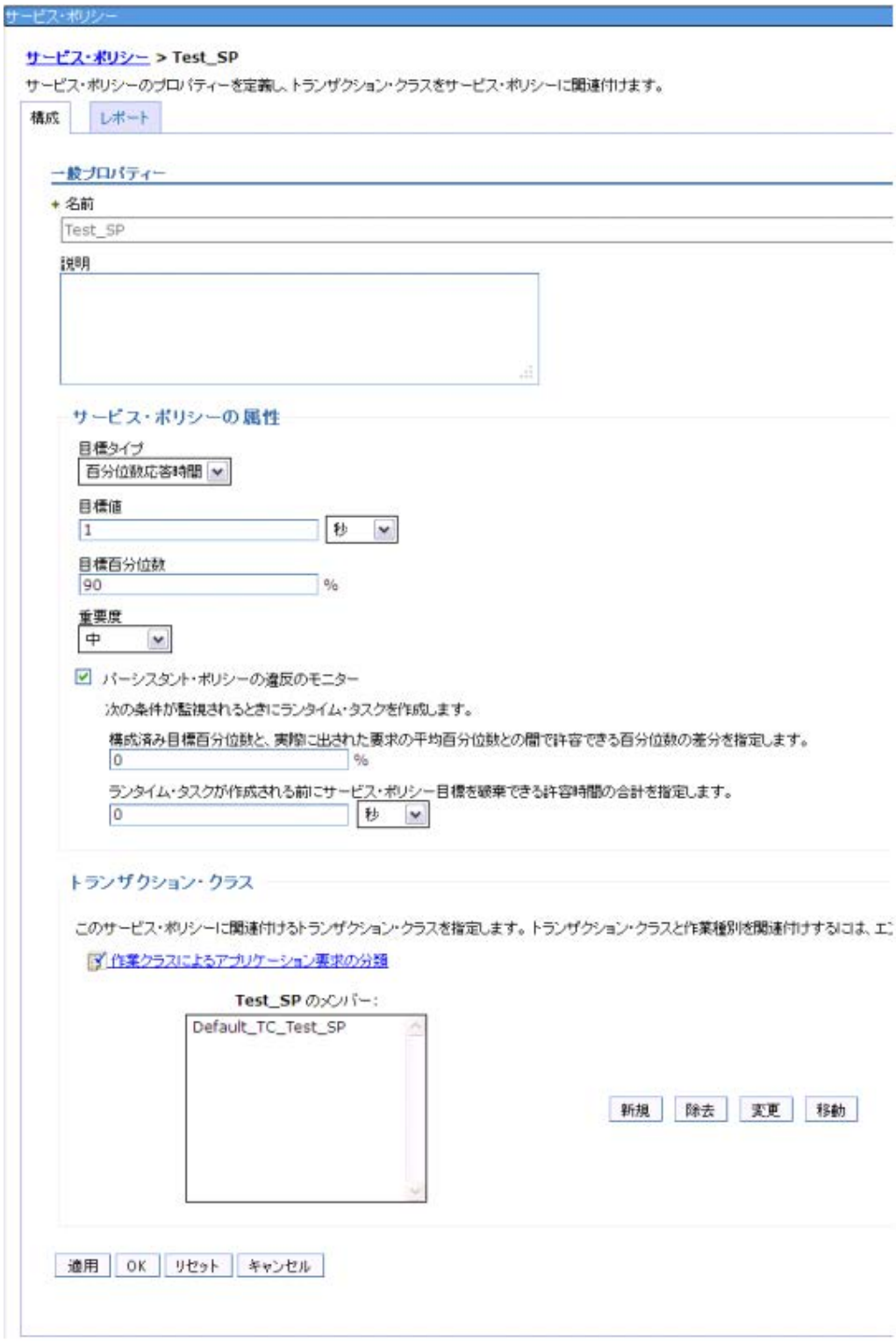

図 3-20 サービス・ポリシー設定

- 名前

サービス・ポリシーの名前を設定します。その重要度に基づきGOLD、SILVER、BRONZEといったキーワードを使用すると視認性が良くなります。

- 目標タイプ

このパラメーターは、任意、平均応答時間、百分位数応答時間のいずれかを設定します。

  - 任意
    任意がデフォルトになります。目標値や重要度を指定することができないため通常は使用しません。このタイプのリクエストはリソースに制約がある場合、パフォーマンスが低下する可能性があります。

  - 平均応答時間
    平均応答時間を目標値とします。パラメーターには目標値と重要度を指定します。

  - 百分位数応答時間
    全要求のうちの何パーセントが目標値の応答時間内に収まるようにする為に指定します。
    パラメーターは目標値と重要度の他に目標百分位数を指定します。

- 目標値

サービスの目標応答時間を指定しますが、通常時の応答時間ではなく、完了するために充てられる最大時間を指定します。

- 目標百分位数（目標百分位）

全要求のうちの何パーセントが目標値の応答時間内に収まるようにするかを指定します。

- 重要度

重要度のレベルを示します。最低から最高の7段階で指定が可能です。デフォルトは中です。

- パーシスタント・ポリシーの違反のモニター

ポリシー違反が発生したときに、ランタイム・タスクを発生させるかを設定します。目標タイプによって設定方法が異なります。

- サービス・ポリシー・メンバーシップの定義

ここでトランザクション・クラスとのマッピングを行います。必要に応じてトランザクション・クラスを新規に作成することが可能です。

図は目標タイプに百分位数応答時間を使用しています。目標タイプが平均応答時間の場合、目標百分位数のパラメーターは存在しません。

次に作業クラス定義の作成方法を紹介します。

この設定は、[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]→[application_name]→[サービス・ポリシー]のページ（図 3-21参照）で変更することができます。

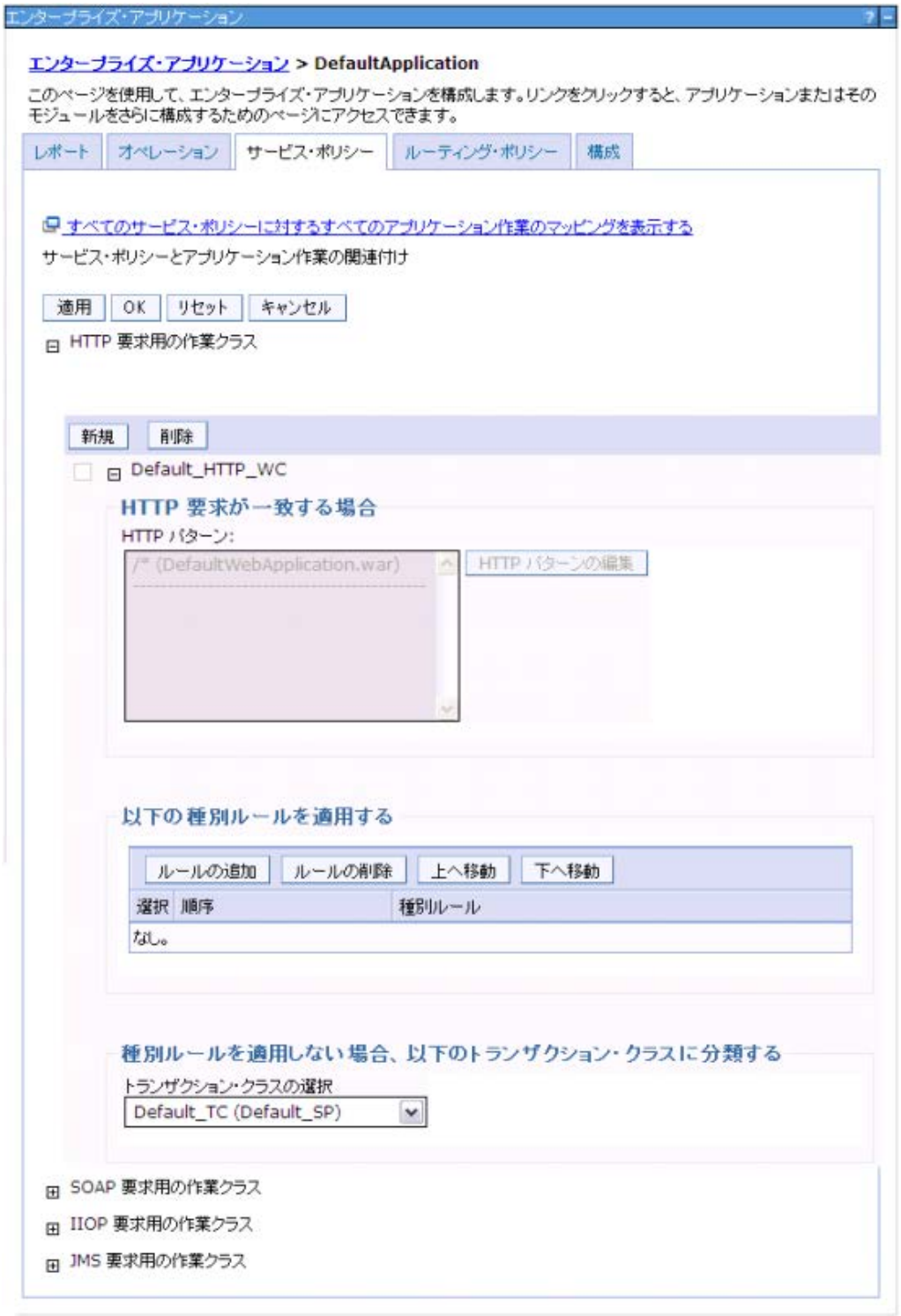

図 3-21 サービス・ポリシー設定

[サービス・ポリシー]タブで、ルールの作成対象となる作業要求タイプおよび作業クラスを展開し、[新規]をクリックします。この中で作業クラス名と、操作内容（HTTP 要求用であれば HTTP パターン）を設定し作業クラスを定義します。

最後に作業クラスをトランザクション・クラスへマッピングします。該当の作業クラスを展開し、"トランザクション・クラスの選択"でマッピングさせるトランザクション・クラスを選択します。

1. 該当の作業クラスを展開し、"以下の種別ルールを適用する"で[ルールの追加]→[ビルド副次式]をクリックします。
2. ビルド副次式ウィンドウで作成した生成済み副次式をルール・エディター（適用するルール）に貼り付けます。
3. 作成した種別ルール内の"トランザクション・クラスに分類する"でこの種別ルールにマッチした際に使用されるトランザクション・クラスを選択します。

この時、種別ルールを用いてさらに詳細なルールでトランザクション・クラスへマッピングさせることも可能です。種別ルールとは、作業クラスをさらに詳細な条件で細分化したものです。作業クラスに指定するルールとなり、複数定義することができます。上位のルールから順番にチェックが行われます。例えば以下のようなルールを作成することができます。

- HTTP ヘッダーの値の指定
- Cookie の値の指定
- 特定の IP からのリクエスト

詳細は、以下を参照してください
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wve.doc%2Fae%2Frwve_odrhttp.html

3-5 HA マネージャー

HA マネージャー・コンポーネントの主な機能は、単一障害ポイント（Single Point of Failure）を除去することです。HA マネージャーは、アプリケーション・サーバー環境を継続的にモニターしています。アプリケーション・サーバー・コンポーネントに障害が発生した場合、HA マネージャーは、障害が発生したサーバーから、シングルトン・サービスをフェイルオーバーします。このアクションにより、アプリケーション・サーバーの可用性が一段と向上します。HA マネージャーは、デプロイメント・マネージャーなどの特定のアプリケーション・サーバー・プロセスではなく、使用可能な任意のアプリケーション・サーバー・プロセス上でサービスを実行し、情報共有を行います。また、HA マネージャーは、セル内のシングルトン・サービスのフェイルオーバーを実行します。例えば、シングルトン・サービスには以下のようなサービスがあります。

- トランザクション・マネージャー
- メッセージング・エンジン

3-5-1 HA マネージャー概要

HA マネージャーは、セル内の全ての JVM 上（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）でサービスとして稼働し、HA 環境を実現します（図 3-22参照）。HA マネージャーという言葉は、各 JVM で稼働するコンポーネントを示す場合と、HA マネージャー同士が連携して

実現する高可用性機能全体を示す場合があります。HA マネージャーは全ての JVM 間での情報共有・障害検知・障害リカバリーのための仕組みを提供します。

HA マネージャーによる機能は大きく3つに分けることができます。

1). 障害検知
各 JVM 間でのハートビートにより、JVM 障害を検知し特定します。

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・DRS 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・サービスのフェイルオーバー
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・サービスのフェイルオーバーと、データベースに書き込まれた JMS メッセージに対する JMS メッセージング・サービスのフェイルオーバーを実施します。

なお、HA マネージャー機能は無効化することができます。詳細は以下の資料を参照ください。
WAS V8.5 Information Center「HA マネージャーの使用不可化と使用可能化」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.multiplatform.doc%2Fae%2Ftrun_ha_ham_enable.html

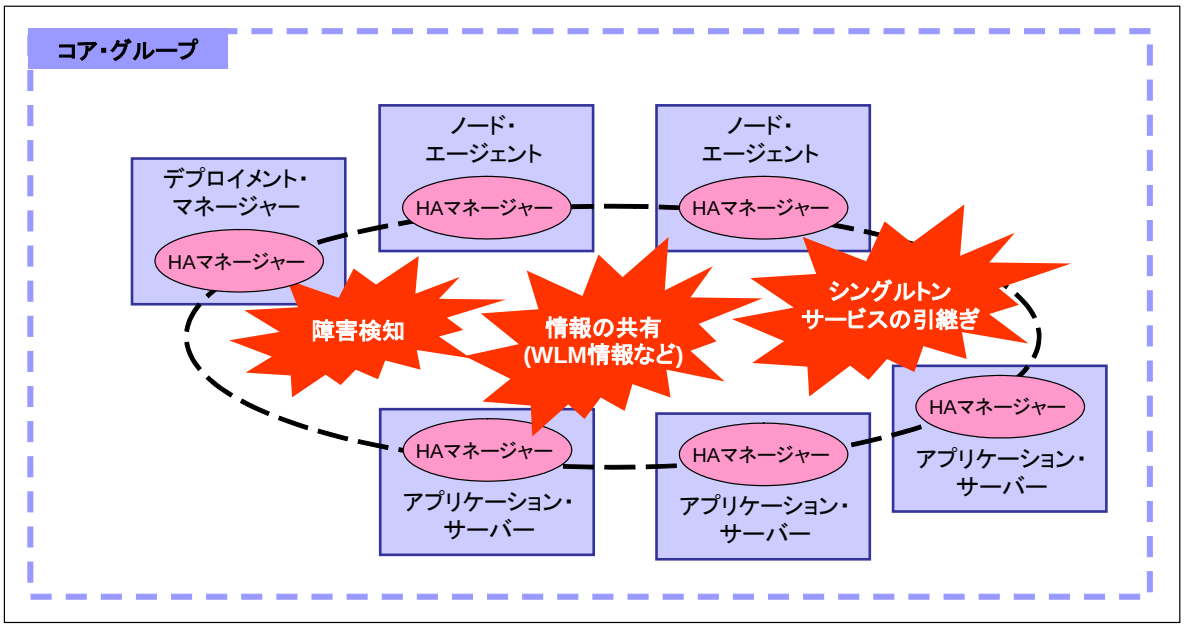

図 3-22 HA マネージャーの概要

3-5-2 HA 環境のコンポーネント

HA 環境を構成するコンポーネントを説明します。

図 3-23に示すように、HA 環境のコンポーネントとして、HA マネージャーとコーディネーターがあり、物理的または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。

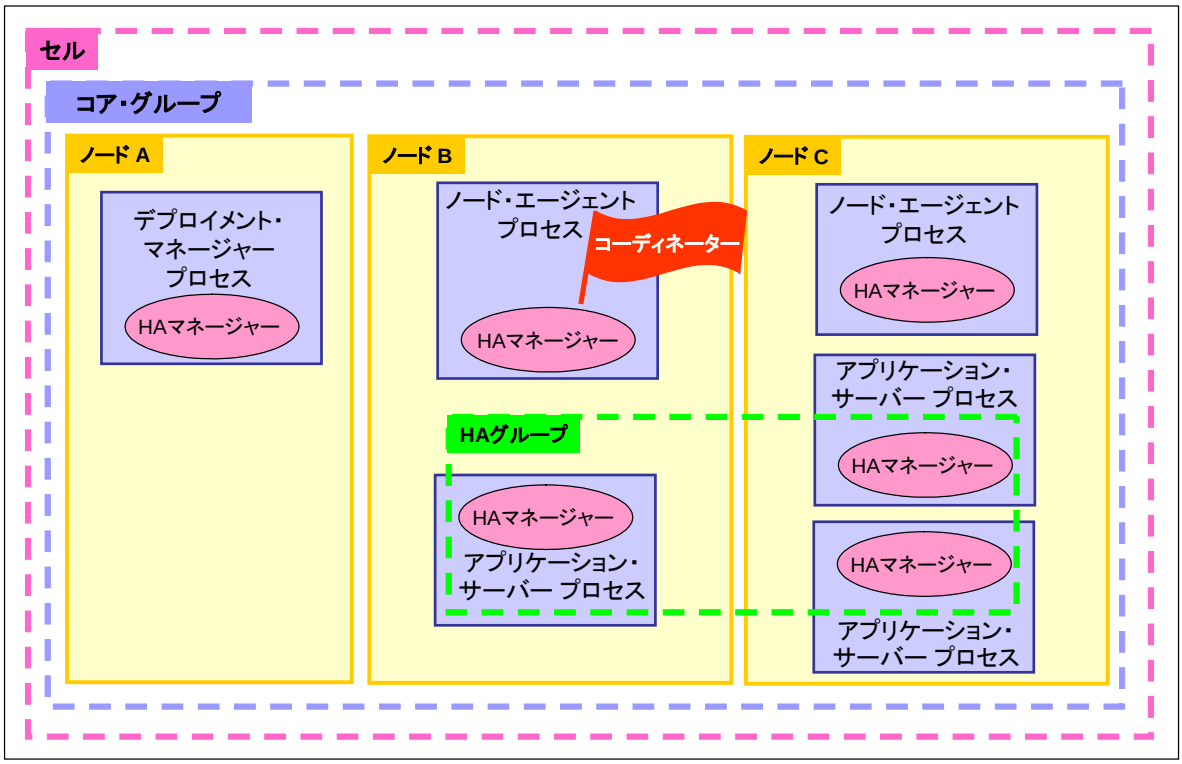

図 3-23 HA 環境のコンポーネント

コア・グループは障害検知、情報共有、フェイルオーバーの HA 機能を実施する範囲です。コア・グループ内ではコンポーネントとして、HA マネージャーとコーディネーターが稼働します。

コーディネーターはコア・グループ内の全ての情報を保持し、HA 機能全体の調整を実施します。コーディネーターは EJB WLM ルーティング情報等を更新し、配布します。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 マネージャーにシングルトン・サービスのフェイ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います。コーディネーターの存在するプロセスに障害が発生すると、自動選択もしくは指定された JVM プロセスがコーディネーターとして稼働することになります。

個々の HA マネージャーは自プロセスで使用する情報を保持します。EJB WLM 情報等の更新がされるとコーディネーターからその情報を受け取り、自分が持つ情報を更新します。またコーディネーターからの指示により、他の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・サーバーにフェイルオーバーします。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イルオーバーするかという設定が HA ポリシーと呼ばれます。各シングルトン・サービスに対して、そのシングルトン・サ

ービスがフェイルオーバー可能なアプリケーション・サーバーの論理的なグループが HA グループです。

3-5-2-1 コア・グループ

コア・グループはセル内で HA 環境の範囲を表します。コア・グループは JVM プロセスをグルーピングします。スタンドアロンのアプリケーション・サーバー、クラスター・メンバーのアプリケーション・サーバー、ノード・エージェント、デプロイメント・マネージャーの JVM プロセスが含まれます（
図 3-23参照）。

セル内には少なくとも 1 つのコア・グループが必要であり、セル内の各 JVM プロセスは、いずれかのコア・グループに所属する必要があります。デフォルトでは、DefaultCoreGroup と呼ばれるコア・グループが自動的に生成され、セル内のすべての JVM プロセスはこの DefaultCoreGroup のメンバーになります。追加のコア・グループは、管理コンソールを使用して作成することができます。

コア・グループとそれに含まれる JVM プロセスには、以下の規則があります。

- 1 つのセル内に複数のコア・グループを作成可能
- セルをまたがったコア・グループの作成はできない
- セル内の JVM プロセスはいずれかのコア・グループに所属しなければならない
- セル内の JVM プロセスは複数のコア・グループに所属することはできない
- あるクラスターの全メンバーは同じ 1 つのコア・グループに所属する必要がある
- 1 つのコア・グループに複数のクラスターの参加は可能

※ 過去のバージョンであった、コア・グループには少なくとも 1 つのデプロイメント・マネージャーまたはノード・エージェントのプロセスが含まれている必要があるという制約は V8.5 ではありません。

ほとんどの構成では、DefaultCoreGroup の使用で十分です。その環境にどうしても必要にならない限り、コア・グループを追加しないことを推奨します。また、特定の問題または状態を解決するため以外の目的では、デフォルトの構成を変更しないでください。追加のコア・グループが必要になるのは以下のような場合です。

- ファイアウォール内部のアプリケーション・サーバーとファイアウォール外部のアプリケーション・サーバーを 1 つのセルにまとめている場合コア・グループ内の通信のため FW 定義が大変になるため。
- 大規模なセル環境で、1つのコア・グループ内のメンバー数が多くなりすぎる場合。100 メンバーまでのコア・グループは問題なく動作しますが、200 メンバーを超えるコア・グループは推奨されません。

コア・グループのスケーリングについては以下の Information Center を参照ください。

WAS V8.5 Information Center「コア・グループのスケーリングについての考慮事項」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_cgscale.html

また、同一セル内の複数のコア・グループ間や異なるセル内のコア・グループとの間で、コア・グループ・ブリッジ・サービスを使用して、WLM 情報を共有することができます。

新しいコア・グループの作成方法、コア・グループ・ブリッジの作成方法の詳細は Information Center の記載を参照ください。

WAS V8.5 Information Center「新規コア・グループ（高可用性ドメイン）の作成」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/trun_ha_newcg.html

WAS V8.5 Information Center「コア・グループ・ブリッジ・サービスの構成」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/trun_ha_coregroupbridge.html

コア･グループの設定

コア・グループの設定では、コア・グループ内でアクティブになるコーディネーターの数とコア・グループ・メンバー間の通信方式の設定変更を行うことができます。

コア・グループの設定を行うには、管理コンソールで、[サーバー]→[コア・グループ]→[コア・グループ**設定**]→[*CoreGroup_name*]を選択します（**図 3-24**参照）。

図 3-24 コア･グループの設定

コーディネーターの数のデフォルト値は、1 です。たいていの環境では 1 つのコーディネーターで十分です。しかし、大規模なコア・グループの場合は複数のコーディネーターの設定が必要になる場合があります。すべての HA データが、コーディネーターのメモリーに保管される必要があります。そのため、コーディネーターが 1 つの場合、大規模なコア・グループではシステム内のメモリーが不足することがあります。そうなると、システムに不具合が生じる可能性があります。メモリーの使用状況は冗長ガーベッジ・コレクションを使用して調査してください。その他、優先コーディネーターで、サービスに利用されない NA や DM でコーディネーターが稼動するようコントロールすることが可能です。

コア・グループのメンバー間の通信方式として下記の 2 種類より選択します。

- チャネル・フレームワーク
  デフォルトの通信方式です。チャネル・チェーンと関連付けることができるため、SSL を使用した通信チェーンなどと関連付けすることも可能です。
- ユニキャスト（非推奨）

ユニキャストの通信は、意図されたメッセージが特定の受信側グループに送信される場合に最も適切と言えます。ただし、V7.0 以降では非推奨となっています。

尚、アプリケーション・サーバー単位で、コア・グループのプロパティーを設定する箇所もあります。HA マネージャー間の通信トランスポートのバッファー・サイズをデフォルトの 10MB から変更することが可能です。このバッファーは HTTP セッション複製や Stateful Session Bean 複製のデータ複製サービス（DRS）と共有しますので、HTTP セッション複製を大量に行う場合などには、バッファー・サイズの拡張が必要になります。設定箇所等は下記の Information Center の記載を参照ください。

WAS V8.5 Information Center「コア・グループ・サービスの設定」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/urun_ha_cgservice.html

3-5-2-2 コーディネーター

コア・グループ内の特定のメンバーがコーディネーターとして選出されます。コーディネーターは、メンバーの起動、停止または失敗の状態を追跡し、各 HA マネージャーに伝達します。トランザクション・マネージャーやメッセージング・エンジンのシングルトン・サービスをどのグループ・メンバー上で稼働するかを決定し、フェイルオーバーが必要な場合はその指示を該当する HA マネージャーに行います。コーディネーターは、Single Point of Failure ではありません。コーディネーターが稼働するサーバー・プロセスが停止した場合、別のサーバー・プロセスが新規コーディネーターとして選択され、これがコア・グループの管理などを引き継ぎます。

管理コンソールを使用して、特定のサーバーを優先コーディネーター・サーバーとして指定できます。優先コーディネーター・サーバーが指定されない場合は、その他のサーバー・プロセスから自動的に選択されます。コア・グループでのコーディネーターのフェイルオーバーを最小限にするには、特定のサーバーを優先コーディネーター・サーバーとして指定し、それらのサーバーの再始動の頻度を制限することをお勧めします。

優先コーディネーターの設定

デフォルトでは、ノード名/サーバー名の組み合わせ（例：host1node01/Amember11、host1node01/nodeagent、host1CellManager01/dmgr）の文字列のアルファベット順にサーバー・プロセスがコーディネーターとして選択されます（先ほどの例の中のサーバーでは、ノード名がもっとも若い host1CellManager01/dmgr が選択されます）。優先コーディネーターを設定することにより、特定のサーバー・プロセス上で優先して、コーディネーターのコンポーネントを稼働させることができます。優先コーディネーターを設定するには、管理コンソールで、**[サーバー]→[コア・グループ]→[コア・グループ設定]→[*CoreGroup_name*]→[追加プロパティー]→[優先コーディネーター・サーバー]**を選択します（**図 3-25**参照）。優先コーディネーターとして指定するサーバーを選択して**[追加]**ボタンをクリックします。複数のサーバーを優先コーディネーターとして指定した場合には、リスト上の上位のサーバーがより優先度が高くなります。優先コーディネーターのサーバーがすべて使用不可の時には、優先コーディネーターとしては指定されていない通常のサーバーの中からコーディネーターが選択されます。

図 3-25 優先コーディネーターの設定

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イ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。

コーディネーターとしての役割を実行するサーバー・プロセスの SystemOut.log には下記の情報が記録されます。

例 3-4 コーディネーターが稼働するサーバーの SystemOut.log メッセージ

```
[08/12/08 13:41:47:063 JST] 0000000e CoordinatorIm I    HMGR0206I: このコーディネーターは、コア・グループ DefaultCoreGroup のアクティブ・コーディネーターです。 アクティブ・コーディ-ネーター・セットは [guideCell01¥host1CellManager01¥dmgr] です。
```

3-5-2-3 HA グループ

HA グループは、コア・グループ内に生成される動的なコンポーネントです。1 つ 1 つの HA グループは、高可用シングルトン・サービスを表します。高可用シングルトン・サービスとしては、トランザクション・マネージャーとメッセージング・エンジンがあります。HA グループ内のメンバーのサーバー・プロセスのうちの 1 つで、シングルトン・サービスが実際に稼働(アクティブ)し、その他のメンバーではシングルトン・サービスの稼働を待機(アイドル状態)します。HA グループのメンバーを変更するには、次節で説明するポリシーを使用します。

HA グループのリストを表示するには、管理コンソールで、**[サーバー]→[コア・グループ]→[コア・グループ設定]→[*CoreGroup_name*]**をクリックし、**[ランタイム]**タブを選択します。**図 3-26**の画面が表示されます。グループ名プロパティー欄にアスタリスク(*)を指定して、**[グループの表示]**ボタンをクリックするとすべてのHAグループのリストが表示されます。グループ名プロパティー欄に一致基準(詳細は**「一致基準」**を参照ください)を入力することにより、特定の HA グループをリストすることができます。例えば、一致基準として、type=WAS_TRANSACTIONSと入力することにより、トランザクション・マネージャーのHAグループをリスト表示することができます。**[サーバーの表示]**ボタンをクリックした場合は、HA グループのアクティブ・メンバーとなっている、現在稼働しているサーバーの状況が表示されます。

図 3-26 HA グループの選択

[グループの表示]ボタンをクリックすると、**図 3-27**に示すように一致基準にマッチしたHAグループがリストされます。それぞれの HA グループは関連付けられた HA ポリシーとともに表示されます。

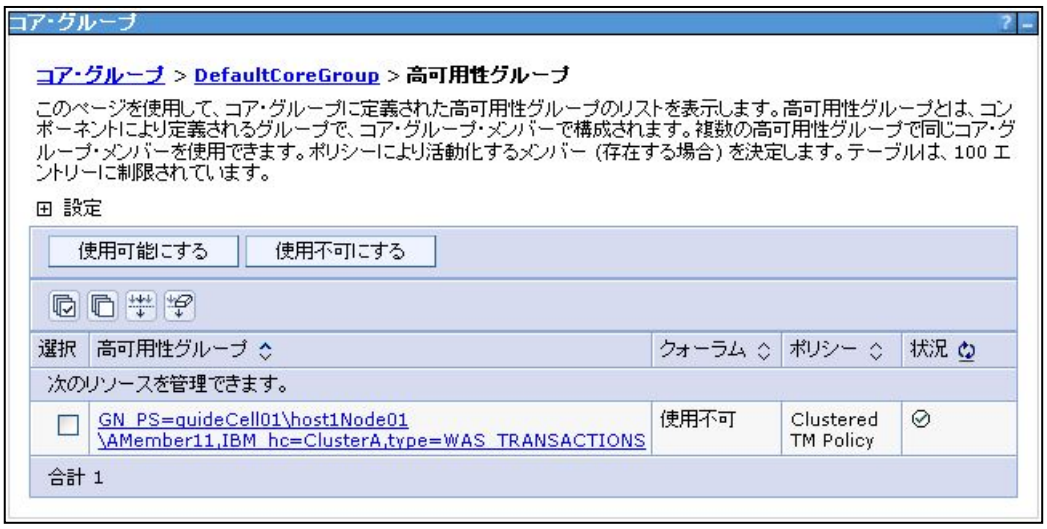

図 3-27 HA グループのリスト

HAグループのリスト内の特定のHAグループのハイパーリンクをクリックすると、**図 3-28**に示すようにHA グループに含まれるメンバーのリストとどのメンバーでアクティブとなっているかが表示されます。下

記の例では、AMember11 のクラスター・メンバー上でシングルトン・サービスが稼働していることが分かります。

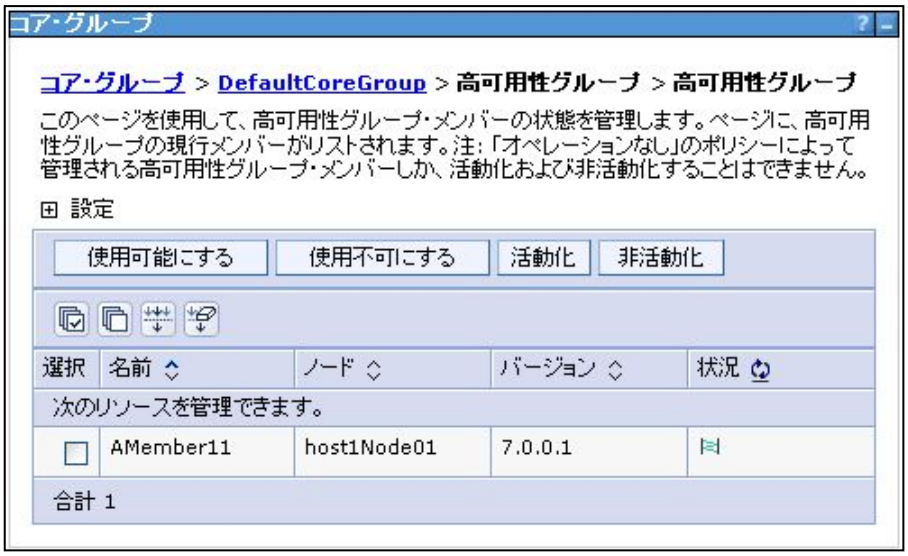

図 3-28 HA グループのメンバーのリスト

3-5-2-4 コア・グループ・ポリシー

コア・グループ・ポリシー（以下、ポリシーと表記）には、シングルトン・サービスがどのようにフェイルオーバーするかを定義します。HA グループ（つまり、シングルトン・サービス）は 1 つのポリシーと関連付けられ、ポリシーの定義にしたがってフェイルオーバー先のクラスター・メンバーやフェイルバックを実施するかを判断します。ある 1 つの HA グループが関連付けられるポリシーは 1 つですが、複数の HA グループが同一のポリシーに関連付けられる場合はあります。デフォルトでは、トランザクション・マネージャーの HA グループはすべて、[Clustered TM Policy]に、メッセージング・エンジンの HA グループはすべて[Default SIBus Policy]に関連付けられています。

ポリシーの新規作成またはポリシー設定の確認・変更を行うには、管理コンソールで、**[サーバー]→[コア・グループ]→[コア・グループ設定]→[*CoreGroup_name*]→[追加プロパティー]→[ポリシー]**を選択し、**[新規作成]**ボタンまたは[*Policy_name*]のハイパーリンクをクリックします。下記画面が表示されます。

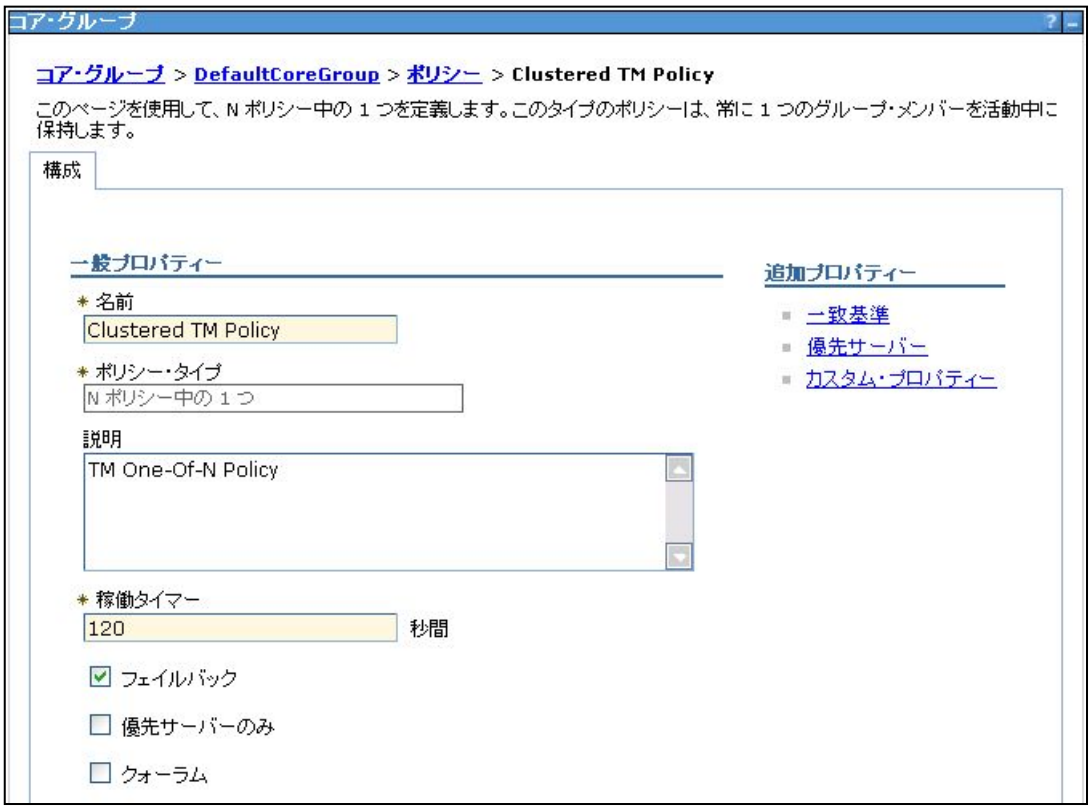

図 3-29 ポリシーの作成・編集

ポリシーの新規作成時は、ポリシー・タイプを選択する必要があります。以下で各項目について説明します。

ポリシー・タイプ

5 種類のポリシー・タイプが選択できますが、下記の制限があります。

- トランザクション・マネージャーのポリシーを設定する場合は、ポリシー・タイプに**[N ポリシー中の 1 つ]**を選択する必要があります。
- メッセージング・エンジンのポリシーを設定する場合は、ポリシー・タイプに**[N ポリシー中の 1 つ]**または**[静的ポリシー]**を選択する必要があります。

詳細は下記の Information Center の記載を参照ください。
WAS V8.5 Information Center「コア・グループ・ポリシー」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/urun_ha_policy.html
したがって一般的には、[N ポリシー中の 1 つ]または[静的ポリシー]のポリシー・タイプを設定します。

静的ポリシー

このポリシーは、シングルトン・サービスを特定の HA グループ・メンバー上で稼働させたい場合に設定します。その特定のメンバーが停止した場合には、シングルトン・サービスも停止します。自動的なフェイルオーバーは行われません。

N ポリシー中の 1 つ

このポリシーは、常に、HA グループのメンバーの中の 1 つのサーバー上でサービスを活動状態にします。このポリシーは、シングルトン・サービスのフェイルオーバーを希望するグループによって使用されます。障害が発生した場合、HA マネージャーが別のサーバーでシングルトン・サービスを始動します。このポリシーを使用する場合、HA グループ内のすべてのメンバー上で外部リソースが使用できる必要があります。具体的には、メッセージング・エンジンの場合には、メッセージング・エンジンのデータソースとしてのリモート・データベースにアクセスできる必要があり、トランザクション・マネージャーの場合には、そのトランザクション・ログが、NAS（Network Attached Storage）などの外部ディスクに保管され、HA グループ内のすべてのメンバーからアクセスできる必要があります。

[N ポリシー中の 1 つ]のポリシー・タイプを選択した場合には、フェイルオーバーの動きを制御する以下の追加オプションがあります。HA グループとポリシーの関連付けを設定する[一致基準]については次節で説明します。

- 優先サーバー
  シングルトン・サービスが稼働するメンバーの優先順位付けを設定することができます。ただし、フェイルバックの設定が有効になっていない場合、より優先順位が高いサーバーが後から起動（復旧）しても、シングルトン・サービスの稼働は、優先順位の低いサーバー上でそのまま稼働し続けますのでご注意ください。
- フェイルバック
  フェイルバックを有効にした場合、より優先度の高いサーバーが復旧した場合に、シングルトン・サービスの稼働はより優先度の高いサーバーに移動します。
- 優先サーバーのみ
  有効にした場合、優先サーバーのリストで指定したサーバー上でのみ、シングルトン・サービスが稼働するように制限します。
- 稼働タイマー
  HA マネージャーがトランザクション・マネージャーやメッセージング・エンジンなどのコンポーネントをコールバックし、稼働チェックを実施する間隔です。デフォルト値は 120 秒です。これにより、トランザクション・マネージャーやメッセージング・エンジン機能などのプログラムがハングした場合、HA マネージャーがハングを検出します。同様の設定項目がアプリケーション・サーバーごとのコア・グループの設定にありますが、ポリシーでの設定が 0 より大きな値の場合、このポリシーで設定した値の方が優先されます。
- クォーラム
  クォーラムが有効な場合、メンバーの過半数がオンラインでないとサービスが活動化されません。クォーラム・メカニズムは、障害によってグループが区画に分割される場合に、アプリケーション・サーバーのシャットダウンを可能にするハードウェア制御機能とともに作動するように設計されています。

オペレーション・ポリシーなし

このポリシーを使用すると、HA マネージャーは自動的にシングルトン・サービスの起動を行いません。このポリシーは外部のクラスタリング・ソフトウェアと合わせて使用することを意図しています。外部クラスタリング・ソフトウェアから HA マネージャーの MBean を操作することによって、シングルトン・サービスの起動を制御します。

一致基準

ポリシーが割り当てられた HA グループによってシングルトン・サービスのフェイルオーバーは管理されます。ポリシーと HA グループの関連付けを行うのが一致基準です。HA グループのプロパティーとポ

リシーごとに設定する一致基準を比較し、最も一致項目が多いポリシーが HA グループに割り当てられます。HA グループのプロパティーは、図 3-27の[高可用性グループ]の欄に表示される値です。

一致基準の値を表示・編集するには、管理コンソールで、[サーバー]→[コア・グループ]→[コア・グループ設定]→[*CoreGroup_name*]→[追加プロパティー]→[ポリシー]→[*Policy_name*]→[追加プロパティー]→[一致基準]を選択します。図 3-30の画面が表示されます。

図 3-30 一致基準の設定

ここで、名前と値の組み合わせを設定し、特定のHAグループにポリシーが割り当てられるようにします。特定のトランザクション・マネージャーとメッセージング・エンジンを指定するために使用する名前と値の組を以下の表に示します。

表 3-5 トランザクション・マネージャーを指定する一致基準

名前	値	ポリシーと一致する トランザクション・マネージャー
type	WAS_TRANSACTIONS	任意のトランザクション・マネージャー
IBM_hc	クラスターの名前	指定したクラスター上で稼働するすべてのトランザクション・マネージャー
GN_PS	トランザクション・マネージャーがホームとするサーバーの名前	特定のトランザクション・マネージャー

表 3-6 メッセージング・エンジンを指定する一致基準

名前	値	ポリシーと一致する メッセージング・エンジン
type	WSAF_SIB	任意のメッセージング・エンジン
WSAF_SIB_MESSAGING_ENGINE	メッセージング・エンジンの名前	特定のメッセージング・エンジン
WSAF_SIB_BUS	SIBus の名前	指定した SIBus 上のすべてのメッセージング・エンジン
IBM_hc	クラスターの名前	指定したクラスター上のすべてのメッセージング・エンジン

3-5-3 障害検知

HA マネージャーは、コア・グループのメンバーの JVM を監視し、必要に応じてフェイルオーバーを実行します。下記の 2 つの方法を使用して障害を検知します。

- ハートビート・シグナルによる障害検知
- ソケットのクローズによる障害検知

3-5-3-1 ハートビート・シグナルによる障害検知

ネットワーク障害やサーバーのハングにより、ハートビートの受信が、正常に行えない回数が一定回数を超えると HA マネージャーは対象プロセスがダウンしたと判断します。デフォルトの設定は、30 秒間隔でハートビートの送信を行い、180 秒間応答がない場合（つまり 6 回のハートビート通信の失敗）、ハートビート送信間隔およびタイムアウト値を設定するには、**[サーバー]→[コア・グループ]→[コア・グループ設定]→[*CoreGroup_name*]→[追加プロパティー]→[ディスカバリーおよび障害検出]**を選択します。下記画面の「ハートビート伝送期間」がハートビートの送信間隔で「ハートビート・タイムアウト期間」がハートビートのタイムアウト値になります。「ハートビート・タイムアウト期間」は「ハートビート伝送間隔」に通信失敗時のリトライ回数を掛けたものを設定してください。

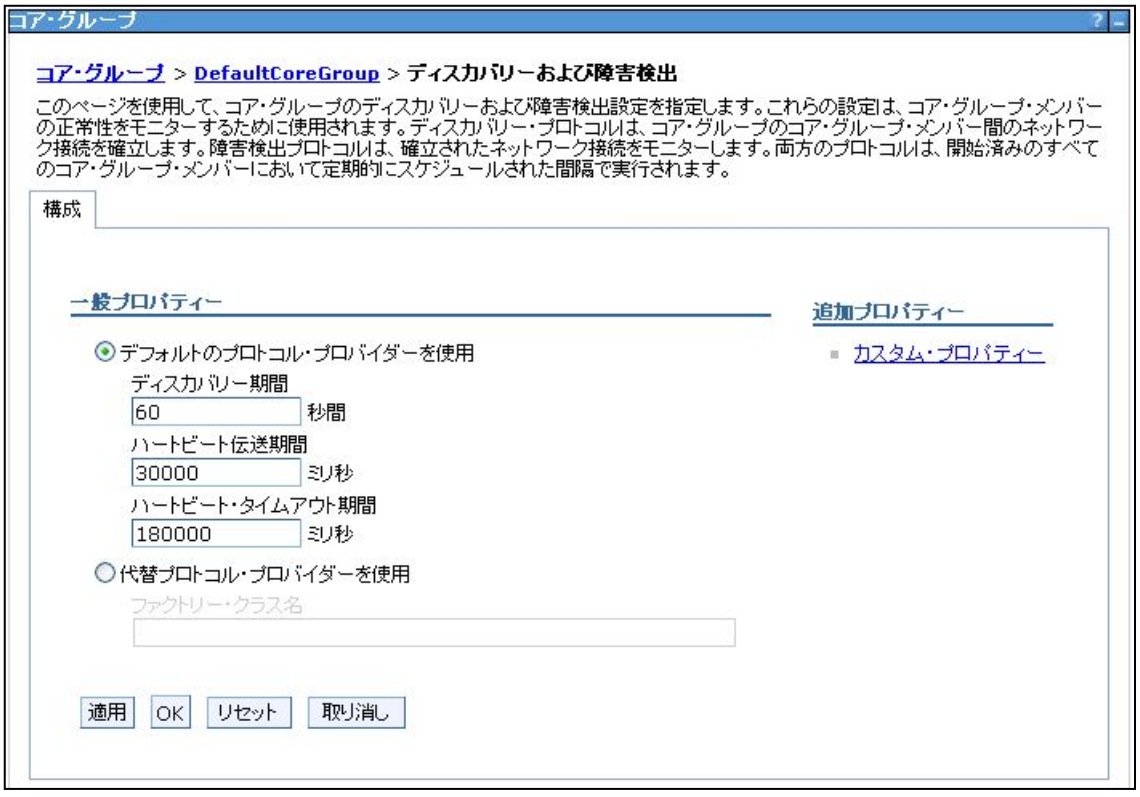

図 3-31 ハートビート・シグナルの障害検知の設定変更

1つのコア・グループ内に V6.x 以前のメンバーが混合して存在する場合は、上記の方法でハートビート設定の変更をすることはできません。この場合は V6.x の場合と同様にカスタム・プロパティーに値を設定することにより、変更が可能です。詳細については以下の Information Center をご参照ください。

WAS V8.5 Information Center「コア・グループに対するデフォルトの障害検出プロトコルの構成」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/trun_ha_cfg_faildetect.html

3-5-3-2 ソケットのクローズによる障害検知

JVM プロセス障害により、ソケットがクローズされると HA マネージャーにより即時に検知され対象プロセスがダウンしたと判断します。OS の TCP/IP KeepAlive のタイムアウトにより、ソケットがクローズされた場合も、プロセスダウンと判断します。

3-5-4 トランザクション・マネージャー・フェイルオーバー

WAS のトランザクション・マネージャーは 2 つ以上のリソースにアクセスする 2 フェーズコミットを行う場合に、トランザクション・ログを書き込みます。トランザクション・ログはディスクに保管され、システム停止やプロセスダウンが発生した場合に、処理途中のトランザクションを回復するために使用されます。

HA マネージャーによるトランザクション・マネージャーのフェイルオーバーを使用すると、あるクラスター・メンバーが 2 フェーズコミットのトランザクション処理の途中で障害が発生した場合に、同じクラスター内の異なるクラスター・メンバーが、アプリケーション・サーバーの再起動をすることなく、即座にトランザクションを回復することができます。トランザクションが回復されたときに、そのトランザクションに関連したデータベールのロックも解放されます。トランザクションの回復が遅れると、そのトランザクション処理が遅れるだけでなく、データベースのロックにより、その他のデータベースアクセスにも影響を及ぼす場合があります。

トランザクション・マネージャーのフェイルオーバーを有効にするには、各クラスター・メンバーから同時にアクセスできる共有ディスクのディレクトリーにトランザクション・ログを配置する必要があります。

3-5-4-1 トランザクション・マネージャー・フェイルオーバー・メカニズム

トランザクションの処理中に障害が発生した場合の、トランザクション・マネージャーのフェイルオーバーによるインダウト・トランザクションの回復メカニズム（トランザクションのピア・リカバリー）について説明します。

図 3-32に示すように、アプリケーション・サーバー1とアプリケーション・サーバー2から構成されるクラスターに対して、トランザクション・マネージャーのフェイルオーバーの設定が行われ、各アプリケーション・サーバーのトランザクション・ログがどちらのアプリケーション・サーバーからもアクセス可能な共有ディスクに保管されていたとします。トランザクション・マネージャーのフェイルオーバー設定の詳細は次節で説明します。アプリケーション・サーバー1で2フェーズコミットのトランザクションの処理中にアプリケーション・サーバー1 またはアプリケーション・サーバー1 が稼働するシステムに障害が発生した場合の、インダウト・トランザクションの HA マネージャーによる回復処理は以下のようになります。

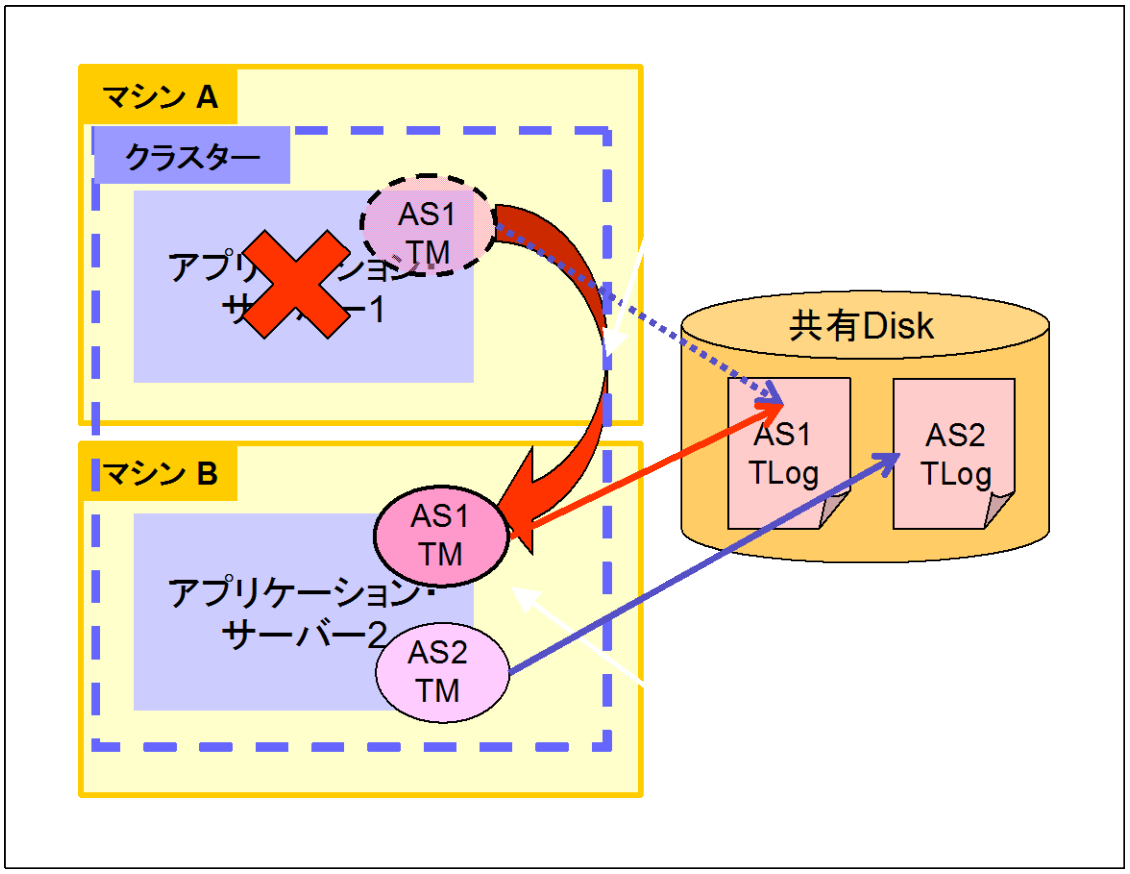

図 3-32 トランザクションのピア・リカバリー

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 にフェイ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イ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。

このとき、アプリケーション・サーバー2 の再起動は必要なく、HA マネージャーが障害を検知してから、アプリケーション・サーバー1 のインダウト・トランザクションを回復する時間は数秒で完了します。尚、アプリケーション・サーバー2 上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・マネージャーはリカバリー専用で起動しており、未解決のトランザクションのリカバリーのみを行います。アプリケーション・サーバー2 で稼働するアプリケーションのトランザクションの処理はアプリケーション・サーバー2 のトランザクション・マネージャーが処理を行います。

3-5-4-2 トランザクション・マネージャー・フェイルオーバー設定

トランザクション・マネージャーのフェイルオーバーを正しく機能させるには、以下の 2 つの設定を行う必要があります。

- トランザクション・ログを共有ディスクへ配置
- トランザクション・マネージャーのフェイルオーバーの有効化

トランザクション・ログを共有ディスクへ配置

トランザクション・ログを配置する共有ディレクトリーとして、下記のファイル・システムが使用できます。

- Network File System (NFS) version 4
- Windows Common Internet File System (CIFS)
- IBM TotalStorage SAN File System

尚、NFS バージョン 4 のサポートは AIX バージョン 5.3 からになりますのでご注意ください。

WAS のプログラムやプロファイルの作成ディレクトリーは共有ディスク上である必要はなく、トランザクション・ログの配置ディレクトリーのみを共有ディスク上に変更します。以下の手順で変更を行います。

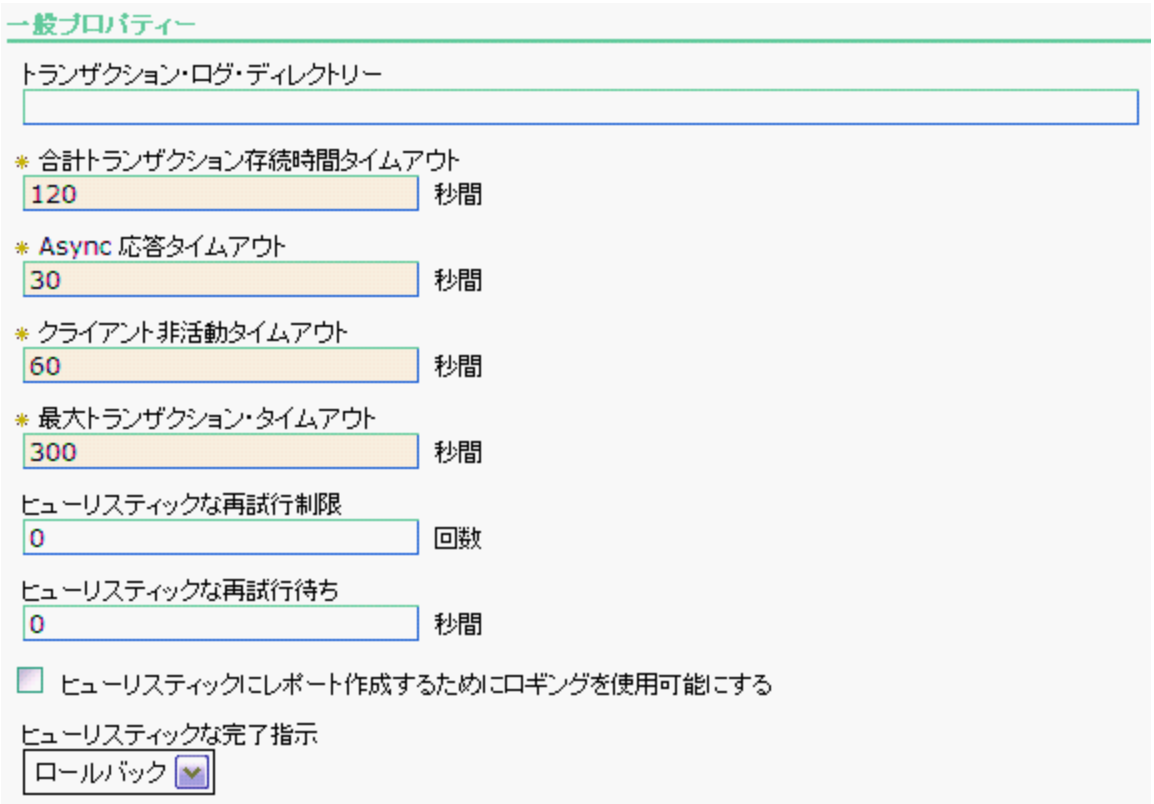

図 3-33 トランザクション・ログ・ディレクトリーの設定

1). クラスターを停止します。
2). すべてのクラスター・メンバーのトランザクション・ログ・ディレクトリーの設定を変更します。
管理コンソールで、[サーバー]→[サーバー・タイプ]→[WebSphere Application Server]→[*ApplicationServer_name*]→[コンテナー・サービス]→[トランザクション・サービス]を選択します。図 3-33の画面が表示されます。[トランザクション・ログ・ディレクトリー]の欄に、それぞれのクラスター・メンバーごとの共有ディスク上のディレクトリーパスを入力します。
3). 構成情報の保管と同期化を行います。

4). OS のコマンドを使用し、既存のトランザクション・ログを、指定した共有ディスク上のディレクトリーにコピーします。
尚、デフォルトのトランザクション・ログ・ディレクトリーは以下のとおりです。
/<*Custom_Profile_root*>/tranlog/*Cell_name*/*Node_name*/*ApplicationServer_name*

5). クラスターを始動します。

手順 2)と 4)は、クラスター・メンバー毎に繰り返します。

トランザクション・マネージャーのフェイルオーバーの有効化

トランザクション・マネージャーのフェイルオーバーの有効化は、クラスターのプロパティーで設定します。管理コンソールで、[サーバー]→[クラスター]→[*Cluster_name*]を選択します。図 3-34の画面が表示されます。[トランザクション・ログ・リカバリーのフェイルオーバーを使用可能にする]のチェックボックスを有効にします。デフォルトでは、無効に設定されています。設定変更後、クラスターの再起動が必要になります。

図 3-34 トランザクション・マネージャー・フェイルオーバーの有効化

フェイルオーバーを設定した各トランザクション・マネージャーのコア・グループ・ポリシーのポリシー・タイプが [N ポリシー中の 1 つ](デフォルトの[Clustered TM Policy]ポリシーのポリシー・タイプは[N ポリシー中の 1 つ]です)であることをご確認ください。その他のポリシーを設定している場合には、HA マネージャーによるトランザクション・マネージャーのフェイルオーバーは正しく機能しません。割り当てられているポリシーの確認は管理コンソールで、[サーバー]→[コア・グループ]→[コア・グループ設定]→

[*CoreGroup_name*]をクリックし、[**ランタイム**]タブを選択します。**図 3-26**の画面が表示されます。グループ名プロパティー欄にアスタリスク(*)を指定して、[**グループの表示**]ボタンをクリックし、すべての HA グループのリストを表示することにより確認できます。

3-5-5 メッセージング・エンジン・フェイルオーバー

メッセージング・エンジンのフェイルオーバーに関しては、「4 章 SIBus 構成」を参照ください。

3-6 高可用性デプロイメント・マネージャー構成（HA DM）

高可用デプロイメント・マネージャー(HA DM)は DM の高可用機能です。DM は管理作業を行わない間は停止させておくことが可能であり、多くのケースでは HA DM は必須の機能ではありません。しかし、アプリケーションの配布やサーバーの監視等の自動化を行っている場合や、視覚化データ・サービスのログを中断させたくない場合など、常に DM が起動していることが重要となるケースでは HA DM は有効です。

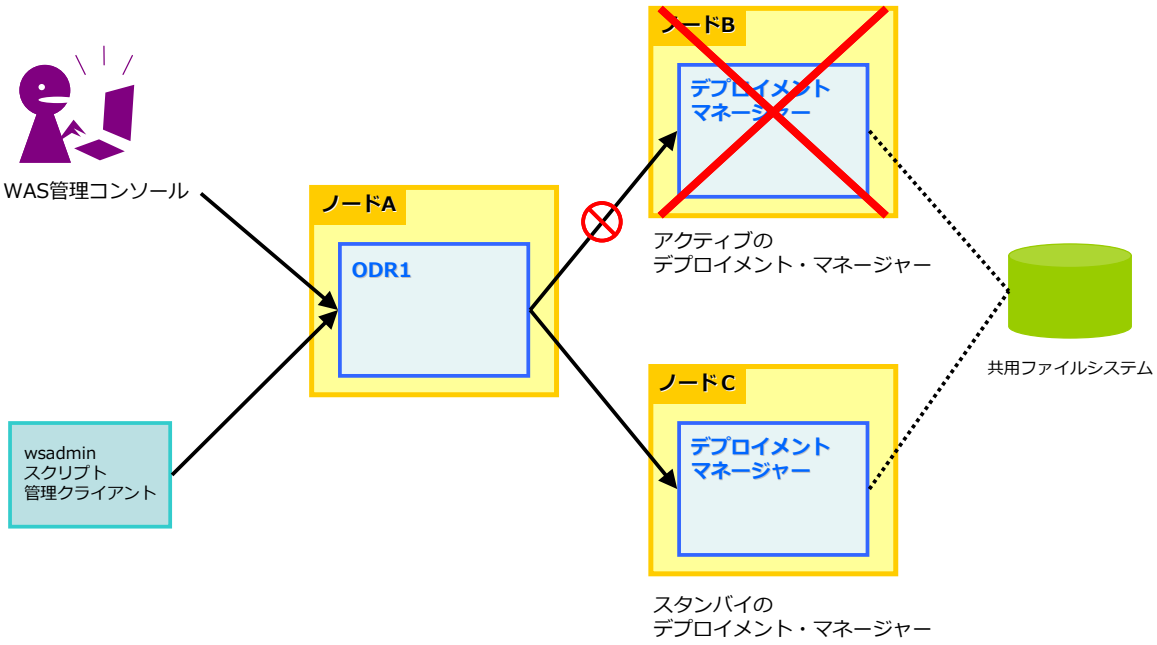

図 3-35 HA DM

3-6-1 HA DM の機能

HA DM を使用することのメリットは、DM の管理機能の単一障害点(SPOF)を排除することにあります。

HA DM では、セル内に 2 つ以上の DM を定義し、アクティブ-スタンバイ構成とします。各 DM はマスター構成リポジトリーとワークスペースの同一インスタンスを共有ファイル・システムで共有します。また、

テイクオーバーする際に、アクティブとスタンバイの両方が自身をアクティブとみなすことを避けるため、アクティブな DM は共有ファイル・システムにロックを取得します。そのため共有ファイル・システムは高速ロック・リカバリーをサポートしている必要があります。SAN FS ではこの時間を 10 秒まで短く設定することが可能です。IBM General Parallel File System（GPFS）が推奨されますが、NFS v4 以降も使用することができます。

アクティブな DM が停止した場合、テイクオーバーが発生してスタンバイ側の DM がアクティブになります。

その他の考慮点として、HA DM 構成では DM へのリクエストは一旦 ODR を経由するため、管理コンソールにアクセスする際のアドレスには DM のホスト名ではなく、ODR のホスト名となります。つまり、DM のホスト名が dmgr1 で、ODR のホスト名が odr1 であった場合、”http://dmgr1:9060/ibm/console”に代わって”http://odr1:9060/ibm/console”を用いることになります。直接アクティブ側の DM にアクセスすることも可能です。

3-6-2 HA DM の設定

HA DM を構成するには、HA DM 構成で新規のセルを作成する方法と、既存のセルを HA DM 構成に変換する方法があります。

HA DM 構成で新規セルを作成する場合は以下の手順となります。

1). WAS を共有ファイル・システムにインストールします。
2). IP アドレス A を用いて DM プロファイルを作成し、起動します。
3). IP アドレス C を用いてカスタムプロファイルを作成し、セルに統合します。このカスタムプロファイルに ODR を作成します。
4). IP アドレス B を用いて DM プロファイルを作成し、xd_HA DMgrAdd コマンドを実行します。これによりスタンバイ側の DM となります。

※ローカルに WAS をインストールし、DM プロファイルのみ共有ファイル・システムに置くことも可能です。

既存セルを HA DM 構成に変換するは以下の手順となります。

1). WAS を共有ファイル・システムにインストールします。
2). IP アドレス A、および既存の DM と同じセル名やノード名を用いて DM プロファイルを作成します。
3). tmsStorage フォルダーとその内容を、既存の DM プロファイルから、共有ファイル・システム上の DM プロファイルにコピーします。
4). セル内に ODR が無い場合は作成します。
5). IP アドレス B を用いて DM プロファイルを作成し、xd_HA DMgrAdd コマンドを実行します。これによりスタンバイ側の DM となります。

WAS V8.5 Information Center「高可用性デプロイメント・マネージャー環境の構成」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/twve_xdsoconfig.html

3-7 保守モード

保守モードを使用することで、ノードまたはサーバー稼動させたまま、リクエストの割り振りを止めることが可能です。保守モードは、ODR またはプロキシー・サーバーがトポロジーに組み込まれ、これらを経

由してリクエストを送信している場合に使用できます。サーバーを保守モードに設定しておいて、プロセスを稼動させたまま問題判別を行ったり、ダンプファイルなどの資料取得を行ったり、必要なチューニングを施したり、といったメンテナンス作業を行うことが可能となります。また、縮退の際にはアフィニティーの管理が重要なポイントとなりますが、保守モード機能を使用するとアフィニティーを維持しながら縮退する事も可能です。

保守モード設定対象のアプリケーション・サーバーに有効なセッションが存在している場合には、このセッションを使用するユーザーからのリクエストは継続して対象のアプリケーション・サーバーに割り振りを行います。新規ユーザーからのリクエストは他のアプリケーション・サーバーへ割り振りがされますので、保守モード設定対象のサーバーからは徐々に有効なセッションが無くなっていきます。

保守モードはアプリケーション・サーバー単位とノード単位で設定することが可能です。

保守モードには、以下のレベルがあります。

- 保守モード
  対象のアプリケーション・サーバーに有効なセッションがある場合には、そのセッションが終了もしくはタイムアウトするまではアフィニティーを維持して該当リクエストを割り振ります。(新規のリクエストは保守モードに設定されていないサーバーに割り振られます。)すべてのセッションが終了した後に、保守モードへ移行します。
- 保守モード - アフィニティー中断
  セッションの状態に関わらず、該当サーバーへの割り振りを中断して即時に保守モードに移行します。ノード単位での設定では選択できません。
- 保守モード - 即時停止
  アプリケーション・サーバー・プロセスを即時に停止した後に保守モードに移行します。
- 標準
  保守モードを解除する場合に選択します。

3-7-1 保守モードの設定方法

この設定は、[システム管理]→[ミドルウェア・ノード]もしくは[サーバー]→[すべてのサーバー]で変更することができます。

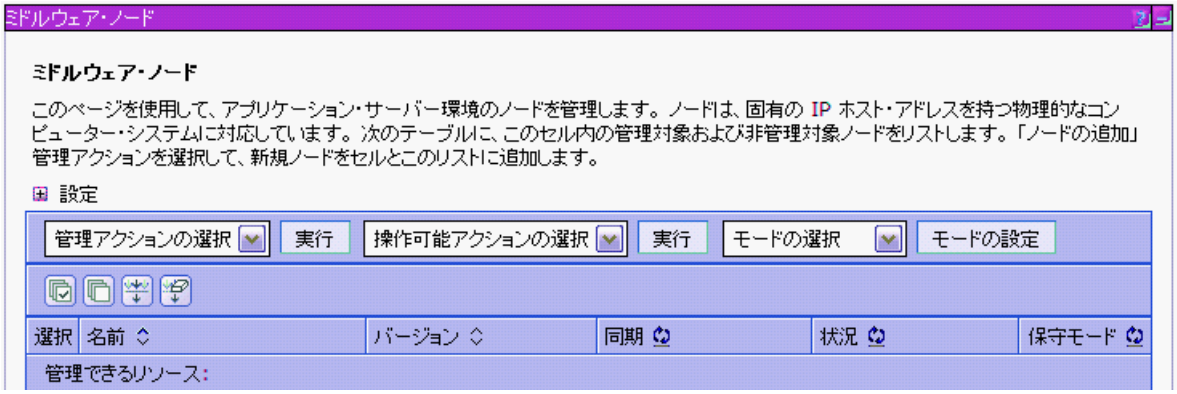

図 3-36 保守モードの設定(ノード)

対象のノード(もしくはサーバー)を選択し、保守モードを選択します。

WAS V8.5 Information Center「保守モードの設定」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/twve_mwmaint.html