NEC CLUSTERPRO による
Micro Focus Enterprise Server
クラスター化の検証報告

平成 18 年 4 月 26 日

マイクロフォーカス株式会社

目次

はじめに

本書では、Micro Focus Enterprise Server（以下、Enterprise Server)で提供するアプリケーションサービスをNEC CLUSTERPRO SE（以下、CLUSTERPRO)により共有ディスクを利用したクラスタ上で動作させた検証を報告します。

Enterprise Server は Windows、Unix、Linux 環境で COBOL サービスをトランザクショナルに管理・実行するアプリケーション サーバです。Enterprise Server は Micro Focus Net Express および Server Express COBOL コンパイラ ツール群の運用部分です。Enterprise Server はメインフレームのトランザクション環境と同等の OLTP 機能を、J2EE アプリケーションサーバと協調実行できるよう設計されています。ユーザにとっては、ハイエンドのトランザクション システムで、機能およびパフォーマンスと同様に重要なことは、その環境がきわめて可用性、信頼性、保守性に優れていることです。

CLUSTERPROはHAクラスタを構築するためのクラスタリングソフトウェアです。
障害発生時には、業務再起動や、冗長化されたサーバへの業務引継ぎ（フェイルオーバ）を行い、継続したサービスを提供します。

次ページ以降、CLUSTERPROとEnterprise Serverを組み合わせ、如何に信頼性・可用性の高いCOBOLサービスシステムを提供していくか、その方法について検証します。

使用環境

使用したハードウェア

(サーバ)
Express5800/120Lh　　2台
　CPU　Intel(R) Xeon(TM) 3.00GHz×4
　MEM　約4.45GB(4669152KB)
　HD　36GB×5
　OS　Red Hat Enterprise Linux AS release 4

(ストレージ)
ST12000　　1台
　HD　8GB×4

(クライアント)
Express5800/58Ta　　1台
　CPU　Celeron (mendcio) 465.594MHz
　MEM　約192MB(191276KB)
　HD　36GB×5
　OS　Red Hat Enterprise Linux AS release 4

使用したソフトウェア

(アプリケーションサーバ)
Micro Focus Enterprise Server 4.0J Service Pack 2 for Intel x86 Linux

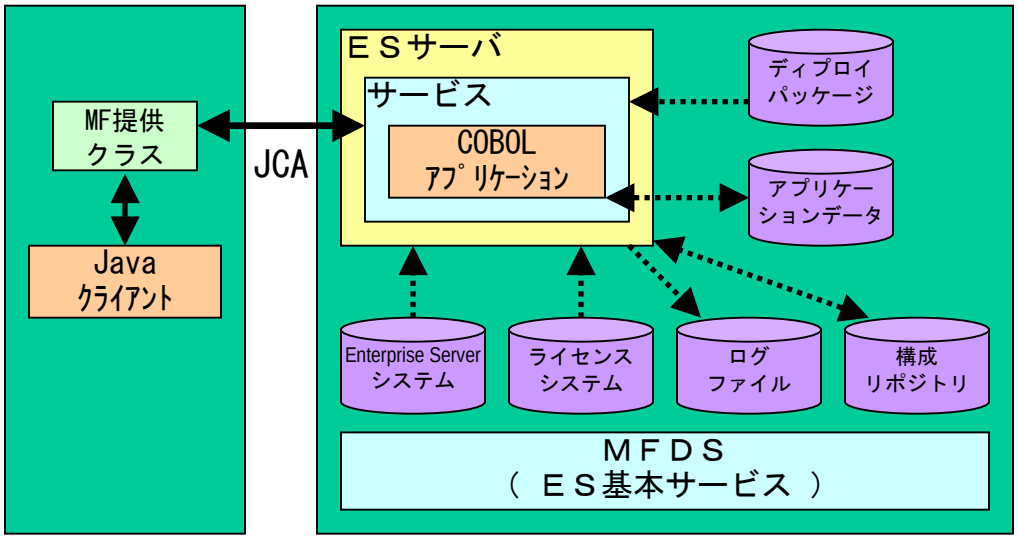
(クラスタリング)
CLUSTERPRO SE for Linux Ver3.1-7

(言語)
Micro Focus Server Express 4.0J Service Pack 2 for Intel x86 Linux
Java(TM) 2 SDK, Standard Edition 1.4.2_8

概要

Enterprise Server とテストシステム

本検証では Enterprise Server に実装するサンプルモデルを用意しました。



Java クライアントから要求されたリクエストは Enterprise Server 上にディプロイされたサービスの COBOL アプリケーションに引き渡されます。この COBOL アプリケーションは索引ファイルであるアプリケーションデータから連番を取得し、この連番をカウントアップして書き込みします。そして日時、プログラム名、実行ホスト名、連番を返します。Java クライアントはこれを表示して終了します。

以下に関連する Enterprise Server の構成要素について説明します。

（プロセス関連）

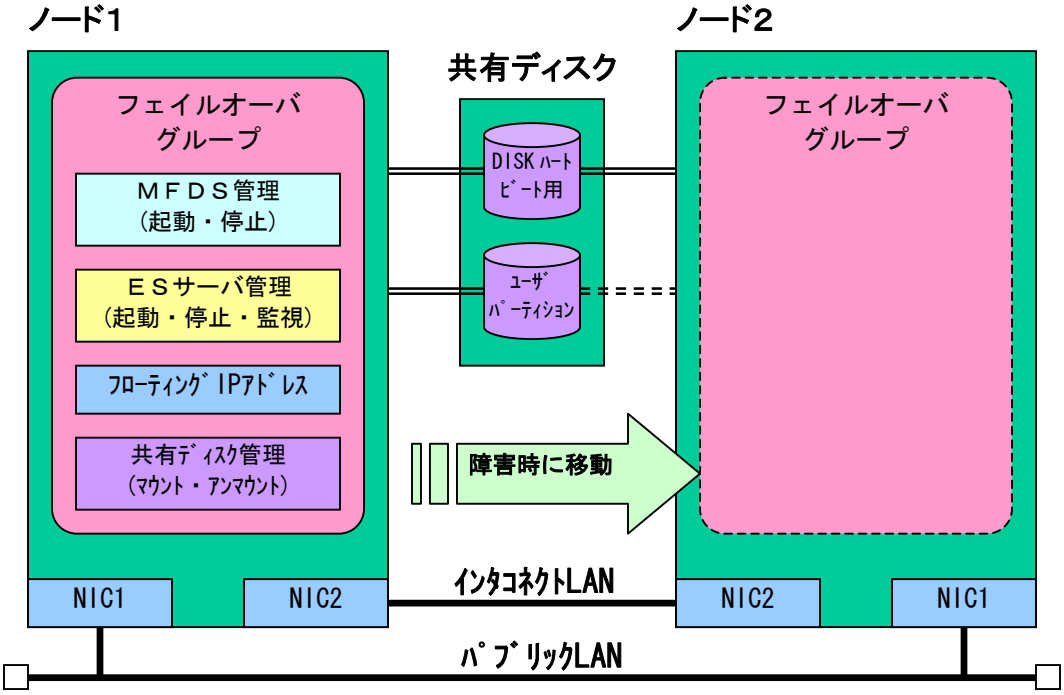
- ・MFDS　Enterprise Server の基本サービスの一つ。1 ノードに 1 つだけ起動が必要
- ・ES サーバ　Enterprise Server の管理単位。1 つのノードに複数定義でき、複数のサービスを含むことができる。
- ・サービス　Enterprise Server で提供する COBOL サービスであり、ディプロイの単位。
- ・COBOL ｱﾌﾟﾘｹｰｼｮﾝ　COBOL プログラムが提供する業務プロセス。
- ・MF 提供クラス　クライアントから Enterprise Server に接続するために利用
- ・Java ｸﾗｲｱﾝﾄ　コマンドライン起動で Enterprise Server にリクエストを出し受け取った結果を表示する。

（ファイル関連）

- ・ES システム　Enterprise Server のシステムコマンド、ライブラリなどのファイルを含む。$COBDIR 環境変数がポイントするディレクトリパス。共有可。
- ・ﾗｲｾﾝｽｼｽﾃﾑ　Enterprise Server のライセンス管理の製品のライセンスが投入されるデータベースを保管する。このディレクトリをディスクコピーなどで移動・復元することはでない。共有不可。
- ・ログファイル　Enterprise Server のログ。ES サーバ単位で管理され、再起動によって自動的に新しいファイルに書き換えられる。共有可。
- ・構成ﾘﾎﾟｼﾞﾄﾘ　Enterprise Server の設定情報を格納したファイル群。共有可。

・ﾃﾞｨﾌﾟﾛｲﾊﾟｯｹｰｼﾞ　car ファイル。ディプロイに必要なファイル群。実装する際のディプロイパッケージは Interfase Mapping Tool Kit(IMTK)により COBOL ソースから容易に生成することが出来る。Server Express によってコンパイルされた実行形式のファイルを含む。（ソースサンプルは巻末に記載）共有可。
・ｱﾌﾟﾘｹｰｼｮﾝﾃﾞｰﾀ　COBOL プログラムが利用するユーザファイル。共有可。
・Java ｸﾗｲｱﾝﾄ　Java プログラム。（ソースサンプルは巻末に記載）
・MF 提供クラス　クライアントプログラムが利用する Enterprise Server 接続用の Java クラス。

クラスタリング



CLUSTERPRO ではフェイルオーバする一連の操作をフェイルオーバグループの単位で管理します。

フェイルオーバグループには、フローティングＩＰアドレス(仮想ＩＰアドレス)や共有ディスクパーティションの有効化、アプリケーションの起動・停止・監視の処理を設定します。
フローティングＩＰアドレスや共有ディスクパーティションはフェイルオーバグループが立ち上がると自動的に有効化され停止すると無効化します。
ＭＦＤＳやＥＳサーバなどの起動・停止はそれぞれのシェルにコマンドで処理を記述します。本運用ではこの起動・停止のシェルには障害時の状態回復・状態保存を行なう処理なども組み込む必要があります。
またアプリケーションの監視は、障害とする事象が発生するまで監視を繰り返す業務監視用のシェルをユーザが準備しこれを組み込む事で行ないます。

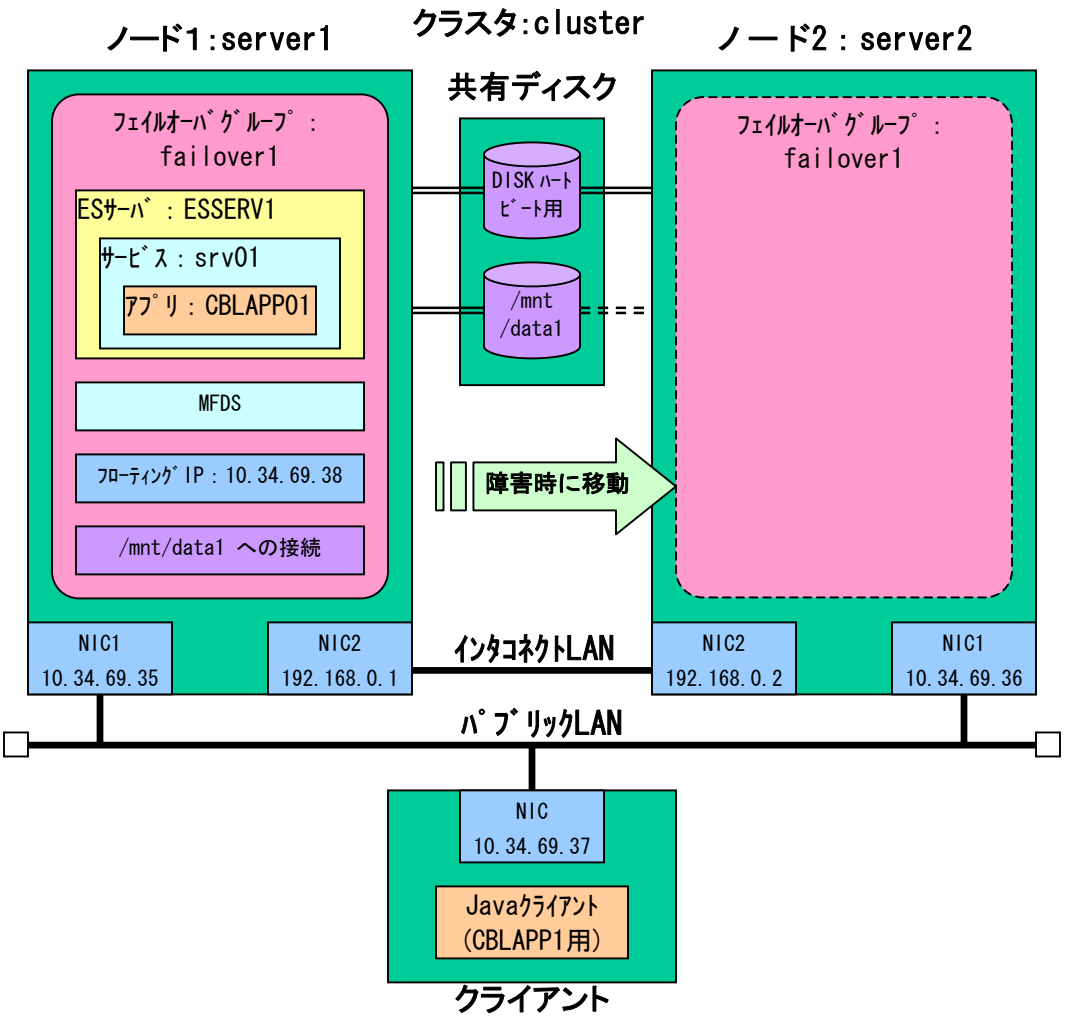
環境設定

下図のモデルを想定した場合の環境の CLUSTERPRO、MF 製品の設定例を記述します。

手順

１．事前準備
２．製品インストールと動作環境の設定
３．CLUSTERPRO の設定
４．COBOL コンポーネントの準備
５．Enterprise Server の設定
６．クライアントの作成

環境設定モデル



１．事前準備

各ノードに搭載された NIC の設定、共有ファイルの接続に関する事前の設定、その他の OS レベルの設定を済ませておきます。

２．製品のインストール

（クラスタ環境）
各ノード個別に下記２つの製品をインストールします。
・Micro Focus Enterprise Server 4.0J Service pack 2 for Intel x86 Linux
・CLUSTERPRO SE for Linux Ver3.1-7

（開発環境）
実装する COBOL コンポーネントの準備に COBOL・Java 開発環境が必要になります。このノード上もしくは別の同一 OS のマシン上に下記の製品をインストールします。
・Micro Focus Server Express 4.0J Service pack 2 for Intel x86 Linux
・Java(TM) 2 SDK, Standard Edition 1.4.2_8
・j2ee.jar

（クライアント環境）
本検証ではクライアントは Java プログラムからリクエストしますので、クライアントに Java 環境が必要になります。本検証では
・Java(TM) 2 SDK, Standard Edition 1.4.2_8
・j2ee.jar

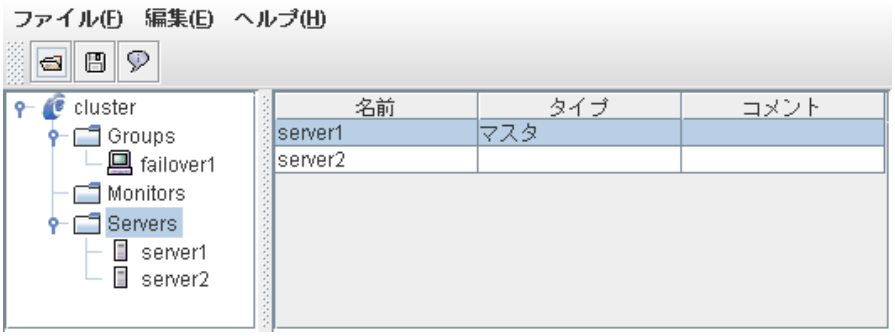
本検証での Enterprise Server のインストールでは、下記の様に配置しました。
ES システム　　：/opt/mf/ES40SP2
ライセンスシステム　：/var/mfaslmf（固定）
ログライル　　：/var/mfcobol/es
構成リポジトリ　　：/opt/mf/ES40SP2/etc/mfds/
/opt と/var は、いずれもローカルのディスクです。

またあらかじめ各製品毎の動作に必要な環境変数等の設定をしておきます。

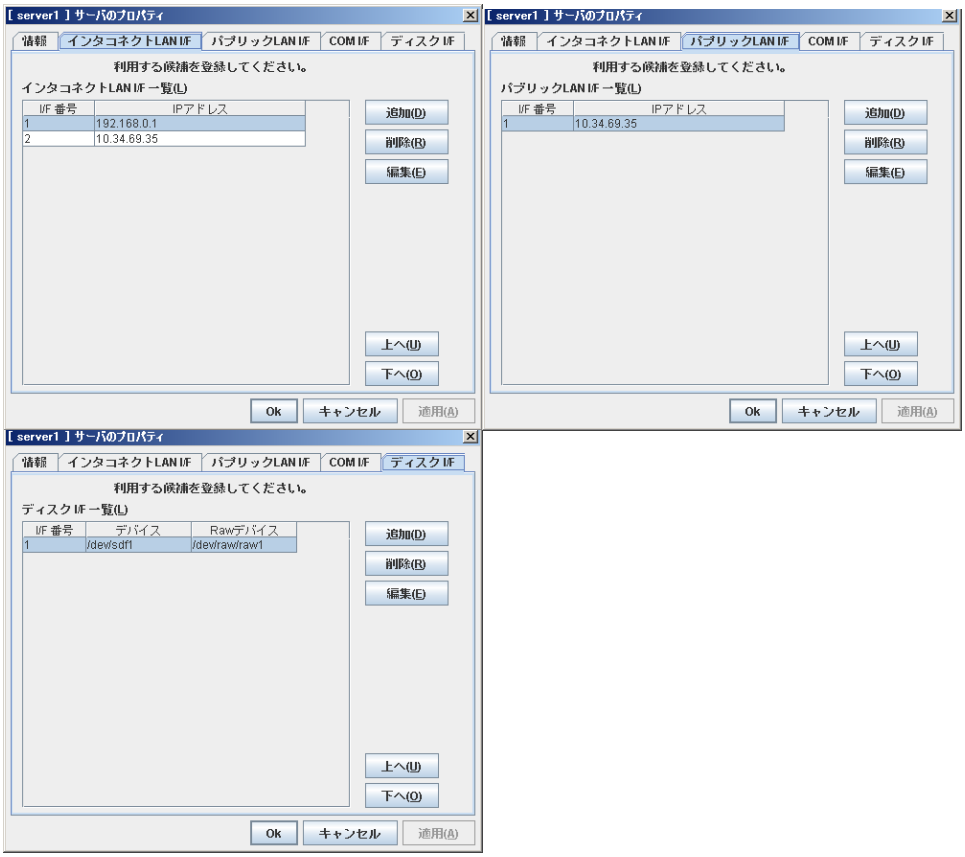
３．CLUSTERPRO の設定

本検証においては CLUSTERPRO の設定は製品に付属するトレッキングツールを使用しました。本ツールを使用するとリモート端末上の GUI 画面でクラスタ全体の構成ファイルを定義・配布が可能となります。
ここで設定するのはサーバの構成、フェイルオーバグループの構成、モニタリングの構成などです。

サーバの構成
構成するノード２つを登録します。ここでは server1 をマスタに設定しました。



ノード毎に各インタフェースの設定を行ないます。



フェイルオーバグループの構成

フェイルオーバグループを必要数定義します。本検証では、1 つのフェイルオーバグループに、フローティング IP、共有ディスク、フェイルオーバするリソースを設定し、テストパターンによっては監視プロセスなどを追加登録しました。

フェイルオーバグループのリソースタイプ execute resource ではアプリケーションの起動・停止をシェルで管理できます。Enterprise Server のコマンドは下記の様に記述します。

処理	コマンド例
mfds の初期化処理 （必要に応じて）	killall -9 mfds32
mfds の起動	mfds & Sleep 15　　注)sleep コマンド値は環境に依存します
mfds の停止	killall -9 mfds32
ES サーバの初期化処理 （必要に応じて）	para=<ES サーバ名> ps -ef | grep $para | grep -E "cassi|cascd|casjcp|casmgr|mfcs " > $$tmp.txt 2> /dev/null

	tmp=`cat $$tmp.txt \| grep -v "grep -E "\| awk '{print $2}'` if [".$tmp" != "."] ; then 　　kill -9 $tmp fi rm -f $$tmp.txt
ES サーバの起動	casstart /r<ES サーバ名>
ES サーバの停止	casstop /r<ES サーバ名>

フェイルオーバグループの起動シェルサンプル(start.sh)

```
#! /bin/sh
#***************************************
#*              start.sh               *
#***************************************

if [ "$CLP_EVENT" = "START" ]
then
        if [ "$CLP_DISK" = "SUCCESS" ]                    環境変数の設定
        then

                source /root/ES4OSP2
                mfds &                                    追記部分
                sleep 15
                casstart /rESSERV1


                echo "NORMAL1"
                if [ "$CLP_SERVER" = "HOME" ]
                then
                        echo "NORMAL2"
                else
                        echo "ON_OTHER1"
                fi
        else
                echo "ERROR_DISK from START"
        fi
elif [ "$CLP_EVENT" = "FAILOVER" ]
then
        if [ "$CLP_DISK" = "SUCCESS" ]                    環境変数の設定
        then

                source /root/ES4OSP2
                mfds &                                    追記部分
                sleep 15
                casstart /rESSERV1


                echo "FAILOVER1"
                if [ "$CLP_SERVER" = "HOME" ]
                then
                        echo "FAILOVER2"
                else
                        echo "ON_OTHER2"
                fi
        else
                echo "ERROR_DISK from FAILOVER"
        fi
else
        echo "NO_CLP"
fi
echo "EXIT"
exit 0
```

フェイルオーバグループの停止シェルサンプル

```
#! /bin/sh
#*************************************
#*              stop.sh              *
#*************************************

if [ "$CLP_EVENT" = "START" ]
then
        if [ "$CLP_DISK" = "SUCCESS" ]
        then
                source /root/ES4OSP2
                casstop /rESSERV1
                killall -9 mfds32

                echo "NORMAL1"
                if [ "$CLP_SERVER" = "HOME" ]
                then
                        echo "NORMAL2"
                else
                        echo "ON_OTHER1"
                fi

        else
                echo "ERROR_DISK from START"
        fi
elif [ "$CLP_EVENT" = "FAILOVER" ]
then
        if [ "$CLP_DISK" = "SUCCESS" ]
        then
                source /root/ES4OSP2
                casstop /rESSERV1
                killall -9 mfds32

                echo "FAILOVER1"
                if [ "$CLP_SERVER" = "HOME" ]
                then
                        echo "FAILOVER2"
                else
                        echo "ON_OTHER2"
                fi

        else
                echo "ERROR_DISK from FAILOVER"
        fi
else
        echo "NO_CLP"
fi
echo "EXIT"
exit 0
```

環境変数の設定

追記部分

環境変数の設定

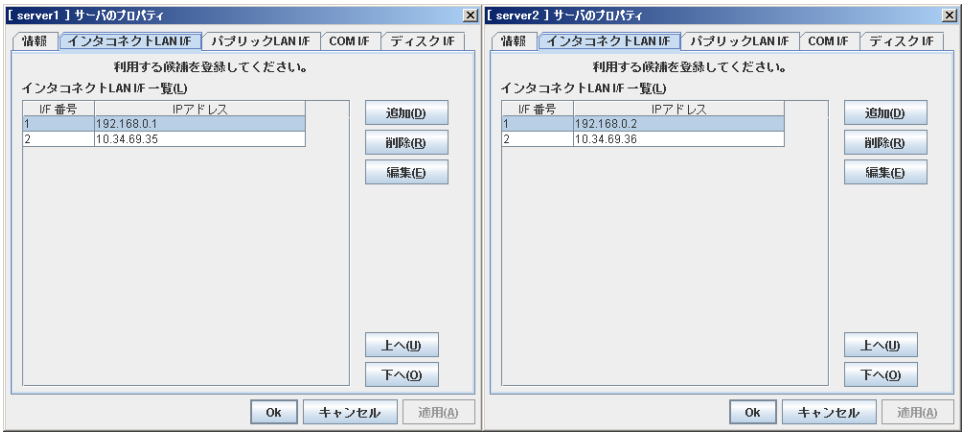
追記部分

これらのシェルは execute resource のプロパティ画面からエディタを起動して編集できます。

またフェイルオーバグループ毎に起動するサーバを優先順位をつけて設定します。

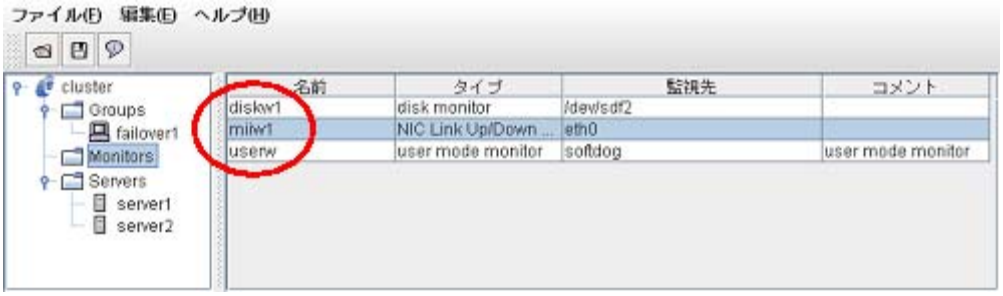
ハートビートの構成

ハートビートの構成では検知する箇所と優先順位を登録します。
例えばインタコネクト LAN で検知を行なうには、まず各サーバの設定で IP アドレスを指定してインタコネクト LAN を登録して I/F 番号を取得します。



次にクラスタの設定で、この I/F 番号と LAN タイプでハートビートの登録を行ないます。

モニタリングの構成
モニタリングの構成では障害を検知するモニタの登録を行ないます。



この検証では、ディスクモニタ、NIC Link Up/Down モニタ、ユーザ空間モニタ(デフォルト)を設定しました。テストパターンによっては pid モニタを追加登録しました。

４．COBOL コンポーネントの準備

Enterprise Server に実装するコンポーネントの準備を行ないます。
作業は開発環境にて行ないます。本検証では Server Express が提供する Interface Mapping Tool Kit を使って一連の作業を簡略化しています。
初めに COBOL ソースの元にサービスに対するデフォルトのマッピングを作成します。サービス名は任意に命名します。

```
  imtkmake -defmap src=cblapp01.cbl service=srv01 type=ejb
```

これで”srv01.xml”の名称でデフォルトのマッピング情報が生成されます。
デフォルトではこのサービスのパラメータはプログラム中のパラメータ名で全て”io”(入出力)として生成されています。本サンプルでは出力パラメータのみであるため実態に合わせ全て”out”に手修正しました。

更にここで同じ COBOL ソースを cob コマンドでコンパイルし、実行形式ファイル(本検証では gnt 形式を選択)を生成しておきます。
そして次のコマンドでディプロイパッケージを生成します。

```
  imtkmake -generate service=srv01 type=ejb cblapp01.gnt
```

これで”srv01.car”ファイルが生成されます。

５．Enterprise Server の設定

まず/etc/services に次の 2 行を追加します。

```
  mfcobol          86/tcp
  mfcobol          86/udp
```

次に mfds コマンドで Enterprise Server を起動します。
Web ブラウザから http://server1:86 でアクセスすると Enterprise Server Administration 画面が開きます。ここで多くの操作を実行できる様になります。

初めにデフォルトで生成されている ES サーバ”ESDEMO”を削除します。
次に新規に必要な ES サーバを作成します。名称以外はデフォルトの設定としました。

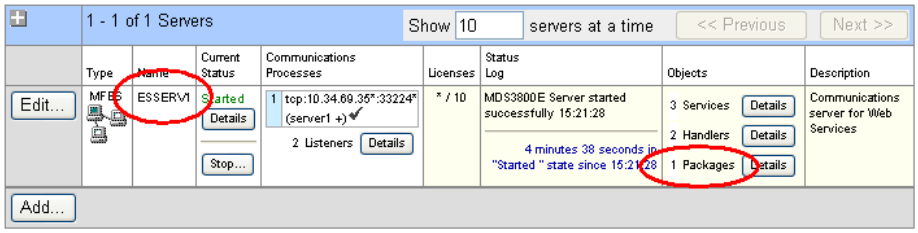
次にディプロイパッケージを使ってサービスの登録を行ないます。
<Enterprise Server ｲﾝｽﾄｰﾙﾃﾞｨﾚｸﾄﾘ>/deploy の下に任意のディプロイ用ディレクトリを作成し、その下に４で作成した car ファイルと実行形式ファイルをコピーします。
<Enterprise Server ｲﾝｽﾄｰﾙﾃﾞｨﾚｸﾄﾘ>/deploy/.mfdeploy を以下の様に書き換えます。

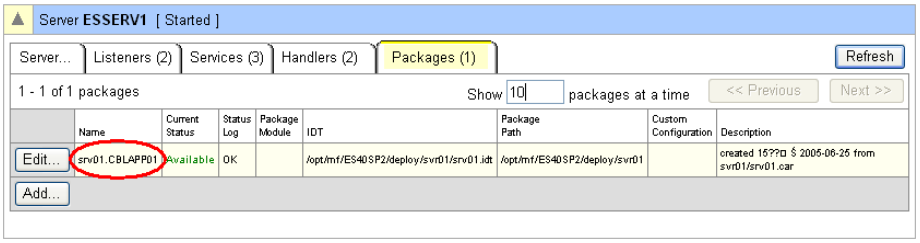
```
  MFDS=localhost
  MFUS_SERVER=ESSERV1
  MFUS_LISTENER=Web Services
```

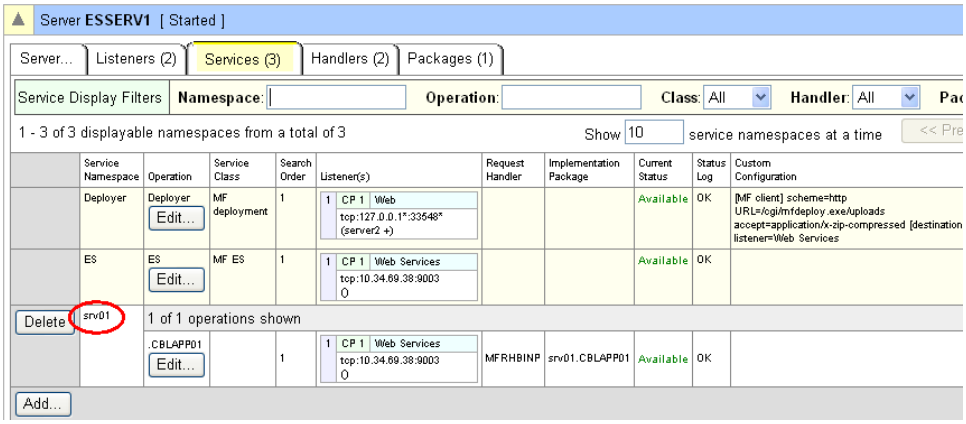
<Enterprise Server ｲﾝｽﾄｰﾙﾃﾞｨﾚｸﾄﾘ>/deploy の下にのディプロイ用ディレクトリに移動して下記のコマンドを実行します。

```
mfdepinst srv01.car
```

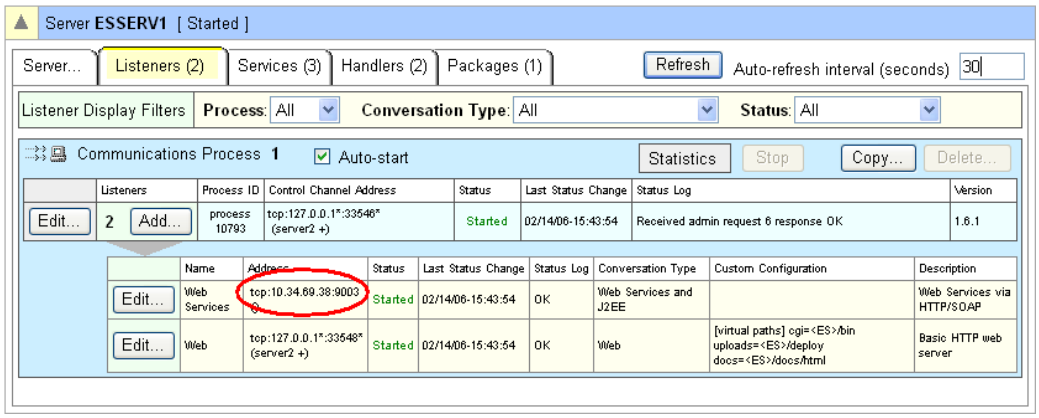
すると.mfdeploy に記載した ES サーバ上に car ファイル作成時に指定した名称でサービスが作成されます。登録された ES サーバは Start(開始)ボタンで手動で開始できます。







次にリスナーの設定をします。
ES サーバを停止後、リスナーの画面の Edit(編集)ボタンで編集画面を開き、”WebServices”の中に”*:*”で設定されている IP アドレス:ポート番号の IP アドレスを固定で設定します。
ここで設定する IP アドレスはフェイルオーバグループでフェイルオーバする場合にはフローティング IP(10.34.69.38)を、グループ化せずに単体でテストする場合には固定 IP(10.34.69.35)を設定すると便利です。



６．クライアントの作成

本検証で Enterprise Server 上にディプロイしたサービスにリクエストを出すのは Java のクライアントプログラムです。（ソースサンプルは巻末に記載）

このプログラムを利用するには$CLASSPATH に下記の 3 ファイルを指定する必要があります。

j2ee.jar
mfconnector.jar　　(<Enterprise Server ｲﾝｽﾄｰﾙﾃﾞｨﾚｸﾄﾘ>/lib に存在)
mfcobolpure.jar　　(<Enterprise Server ｲﾝｽﾄｰﾙﾃﾞｨﾚｸﾄﾘ>/lib に存在)

検証

テストパターン

本検証では以下の７パターンの確認をしました。

パターン	タイトル	説明
①	手動切り替え	CLUSTERPRO への操作による切り替えを確認
②	ｲﾝﾀｺﾈｸﾄ障害	ｲﾝﾀｺﾈｸﾄ LAN のケーブルを引き抜いて切り替えを確認
③	業務障害	業務監視用シェルの異常検出による切り替えを確認
④	OS のみで待機 （シングルスタンバイ）	待機状態のノードでは MFDS 起動せず切り替えを確認
⑤	MFDS 起動で待機 （高速スタンバイ 1）	待機状態のノードでは MFDS 起動して切り替えを確認
⑥	DS サーバ起動で待機 （高速スタンバイ 2）	待機状態のノードでは MFDS、DS サーバを起動状態で切り替えを確認
⑦	別の業務を提供しながら待機 （相互スタンバイ）	待機状態のノードでは MFDS 起動、切り替える DS サーバとは別の DS サーバを動作させせながら切り替えを確認。ノード 1⇔ノード 2 の双方向で確認

①～③はフェイルオーバのトリガーのテストです。代表的なトリガー3 種類のパターンで確認します。
フェイルオーバをステータスで確認します。

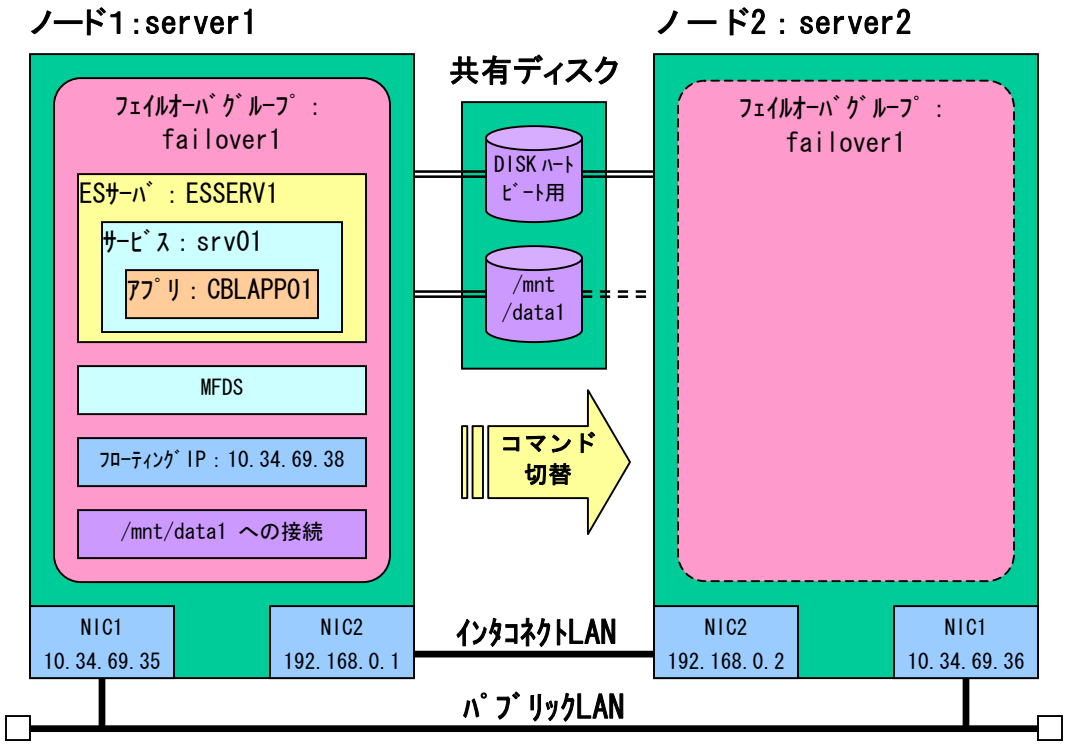
④～⑦は待機側のノードの状態を変えるテストです。本検証では Enterprise Server のプロセスの実行状態を変えて確認します。
Java クライアントから繰り返しリクエストを出し続け、フェイルオーバ後もサービスが提供され続ける事を確認します。

以降、パターン毎の設定と検証結果を記述します。

パターン① 手動切り替え

＜目的＞
通常の運用でも行なわれる動作の確認。
CLUSTERPRO のコマンドを実行し、フェイルオーバグループの移動が行なわれるか検証する。

＜構成＞



＜設定のポイント＞

failover1 の開始シェルに組み込む起動コマンド

```
mfds &
sleep 15
casstart /rESSERV1
```

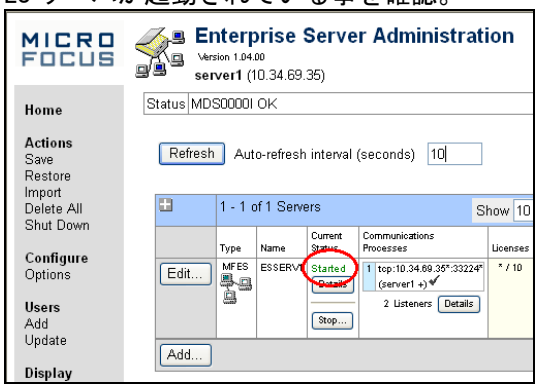
failover1 の終了シェルに組み込む停止コマンド

```
casstop /rESSERV1
killall -9 mfds32
```

＜検証＞
CLUSTERPRO コマンドにてフェイルオーバを行なう。

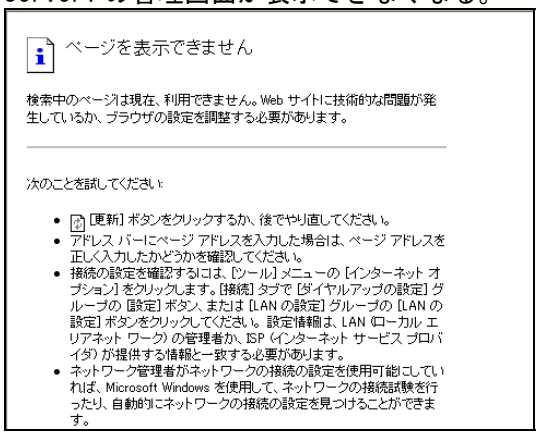
CLUSTERPRO が立ち上がると server1 でフェイルオーバグループ failover1 が開始する。

ES サーバが起動されている事を確認。

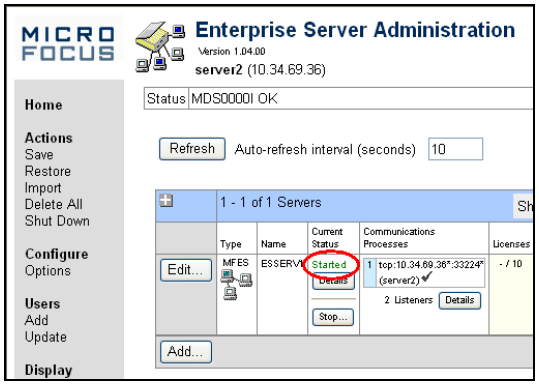


次に CLUSTERPRO コマンドにてフェイルオーバグループを移動する
　＞ clpgrp -m failover1

すると、フェイルオーバグループ failover1 の停止処理により mfds が終了するため server1 の管理画面が表示できなくなる。



そして server2 でフェイルオーバグループ failover1 が開始される。



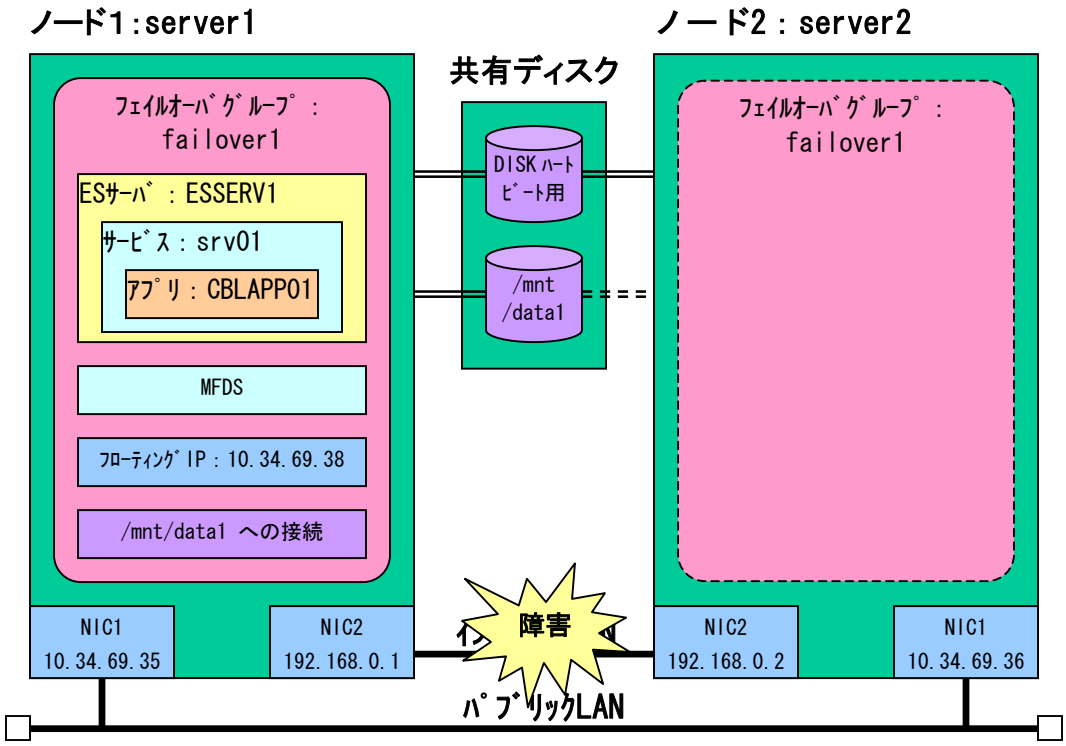
＜結果＞

CLUSTERPRO のコマンド clpgrp によるフェイルオーバグループの移動を確認。

パターン② インタコネクト障害

＜目的＞
CLUSTERPRO の機能のみで検知する障害パターンの確認
インタコネクト LAN の障害を起こし、フェイルオーバグループの移動が行なわれるか検証する。

＜構成＞



＜設定のポイント＞

failover1 の開始シェルに組み込む起動コマンド

```
mfds &
sleep 15
casstart /rESSERV1
```

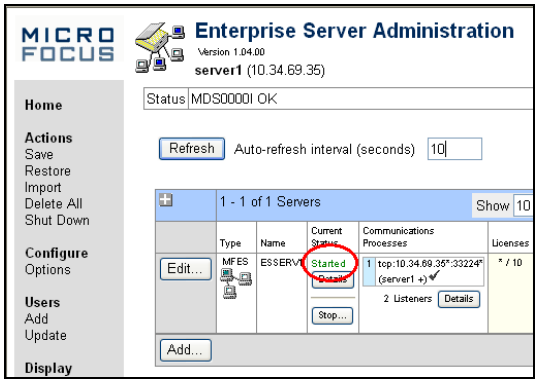
failover1 の終了シェルに組み込む停止コマンド

```
casstop /rESSERV1
killall -9 mfds32
```

クラスタの設定にインタコネクト LAN の登録をする。

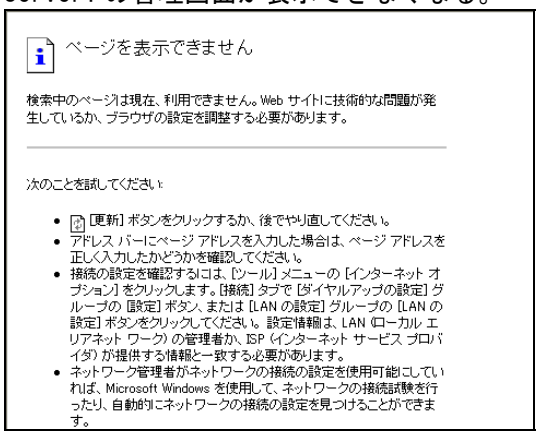
＜検証＞

server1 でフェイルオーバグループ failover1 を起動している初期状態にする。

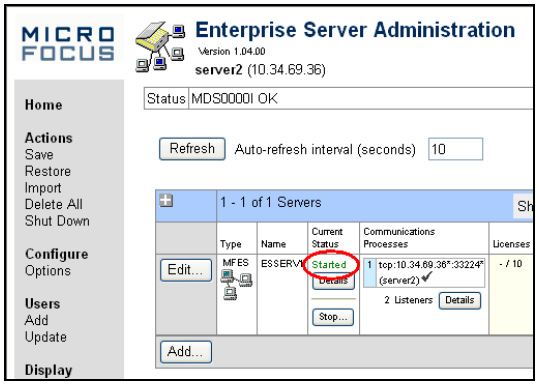


インタコネクト LAN のケーブルを引き抜く。

すると、フェイルオーバグループ failover1 の停止処理により mfds が止まるため server1 の管理画面が表示できなくなる。



そして server2 でフェイルオーバグループ failover1 が開始される。



この時の CLUSTERPRO の状態(clpstat コマンドの結果)

```
Every 2.0s: clpstat

======================  CLUSTER STATUS  ============================
 Cluster : cluster
 <server>
   server1 .........: Caution
     lanhb1         : Normal           LAN Heartbeat
     lanhb2         : Caution          LAN Heartbeat
     diskhb1        : Normal           DISK Heartbeat
  *server2 .........: Caution
     lanhb1         : Normal           LAN Heartbeat
     lanhb2         : Caution          LAN Heartbeat
     diskhb1        : Normal           DISK Heartbeat
 <group>
   failover1 .......: Online
     current        : server2
     disk1          : Online
     exec1          : Online
     fip1           : Online
 <monitor>
   diskw1           : Normal
   miiw1            : Caution
   userw            : Normal           user mode monitor
==================================================================
```

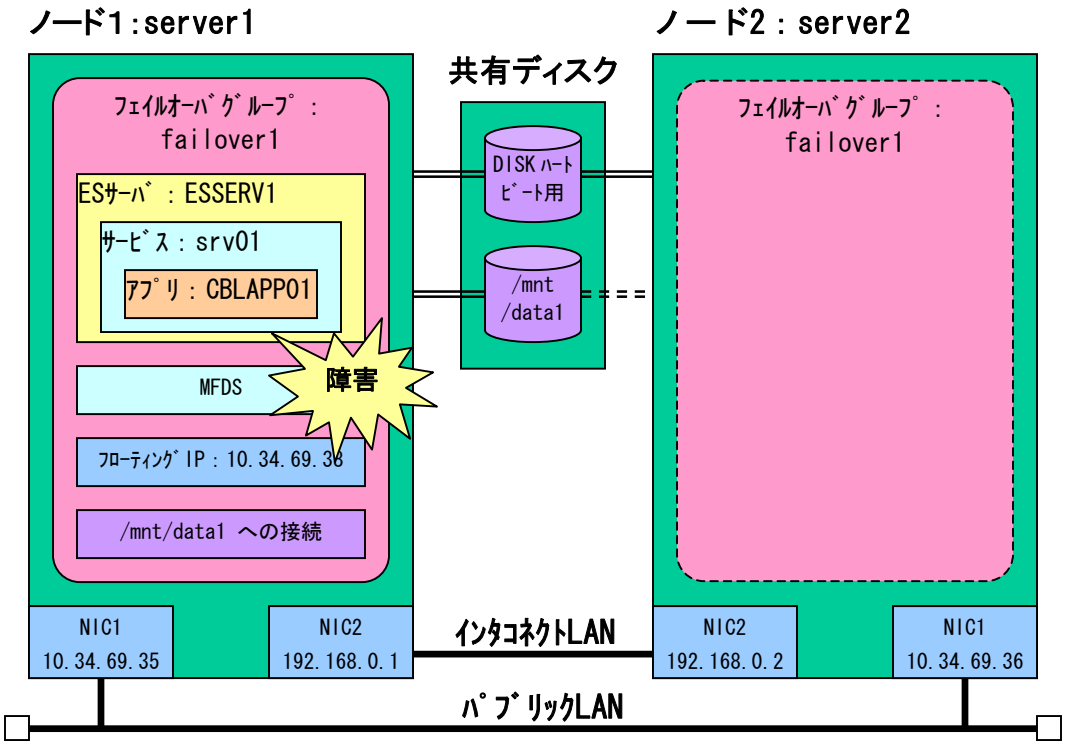
＜結果＞

インタコネクトの障害によるフェイルオーバグループの移動を確認。

パターン③ 業務障害

＜目的＞
ユーザ作成の監視シェルにより検知する障害パターンの確認
本検証では Enterprise Server のプロセス異常を想定し、主要サービスである MFDS を停止しすることでフェイルオーバグループの移動が行なわれるか検証する。

＜構成＞



＜設定のポイント＞

failover1 の開始シェルに組み込む起動コマンド

```
mfds &
sleep 15
casstart /rESSERV1
```

failover1 の終了シェルに組み込む停止コマンド

```
casstop /rESSERV1
killall -9 mfds32
```

ユーザ作成シェルで状態監視を行なうには、監視用のシェルを新たな execute resource として登録・起動し、このシェルの実行状態を CLUSTERPRO で監視して障害検知を行なう。

ユーザ作成シェルは自動的に監視対象をチェックし続け、異常を検知した時に終了するように作成する。

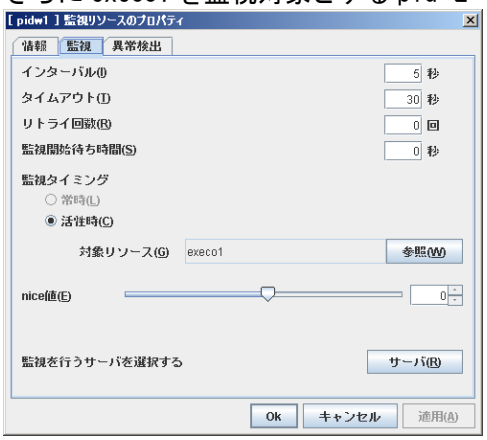
failover1 に監視用の execute resource、execo1 を追加

execo1 の起動シェル start.sh に監視処理を組み込む

```
sleep 10
mdump_func()
{
mdump localhost ESSERV1 > $$tmp.txt  2> /dev/null
web_srv_flg=0
cat $$tmp.txt | while read line
do
        tmp=`echo $line | fgrep "SERVER_DOWN"`
        if [ ".$tmp" != "." ] ; then
                return 3
        fi
        tmp=`echo $line | fgrep "CN:"`
        if [ ".$tmp" != "." ] ; then
                tmp=`echo $line | fgrep "Web Services"`
                if [ ".$tmp" != "." ] ; then
                        web_srv_flg=1
                else
                        web_srv_flg=0
                fi
        fi
        if [ $web_srv_flg -eq 1 ] ; then
                lisnr=`echo $line | fgrep "mfListenerStatus:"`
                if [ ".$lisnr" != "." ] ; then
                        lisnr=`echo $line | fgrep "Started"`
                        if [ ".$lisnr" != "." ] ; then
                                return 0
                        else
                                return 1
                        fi
                fi
        fi
done
}
if [ ".$host_name" = "."  ] || [ ".$es_name" = "." ] ; then
    host_name="localhost"
    es_name=""
fi
while : ; do
        # echo "monitoring"
        mdump_func
        res=$?
        rm $$tmp.txt
        if [ "$res" = "1" ]; then
                exit 0
        fi
        if [ "$res" = "3" ]; then
                exit 0
        fi
        sleep 5
done
;;
```

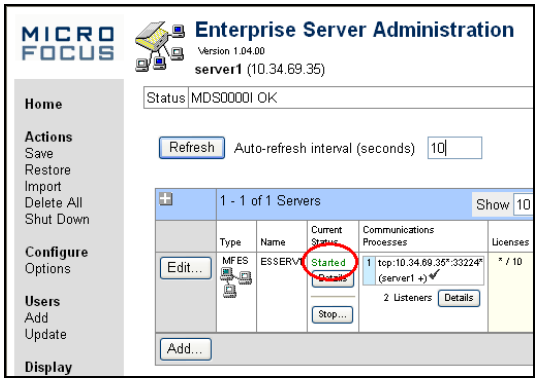
ここに記載した監視処理はマイクロフォーカスが提供する mdump ユーティリティの出力を利用して異常を検出する。(MFDS が正常に動作しておらず)mdump がステータス情報を正常に取得できない場合、及び、Web Service リスナーのステータスが” Started”以外にて異常とする検知する方式とした。

さらに execo1 を監視対象とする pid モニタを追加登録する。



＜検証＞

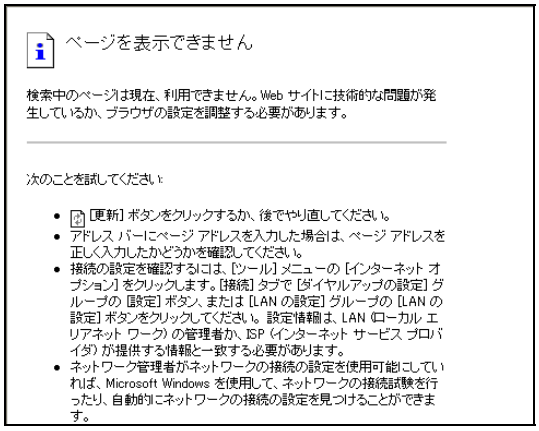
server1 でフェイルオーバグループ failover1 を起動している初期状態にする。



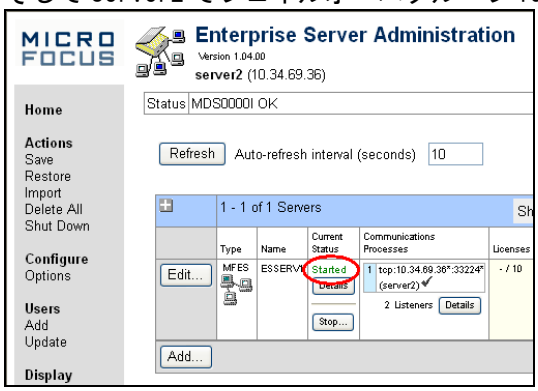
次に MFDS のプロセスを kill する。

```
> killall -9 mfds32
```

すると、mfds のプロセス異常により execo1 の監視シェルが障害を検知しファイルオーバが開始する。
フェイルオーバグループ failover1 の停止処理により mfds が止まるため server1 の管理画面が表示できなくなる。



そして server2 でフェイルオーバグループ failover1 が開始される。



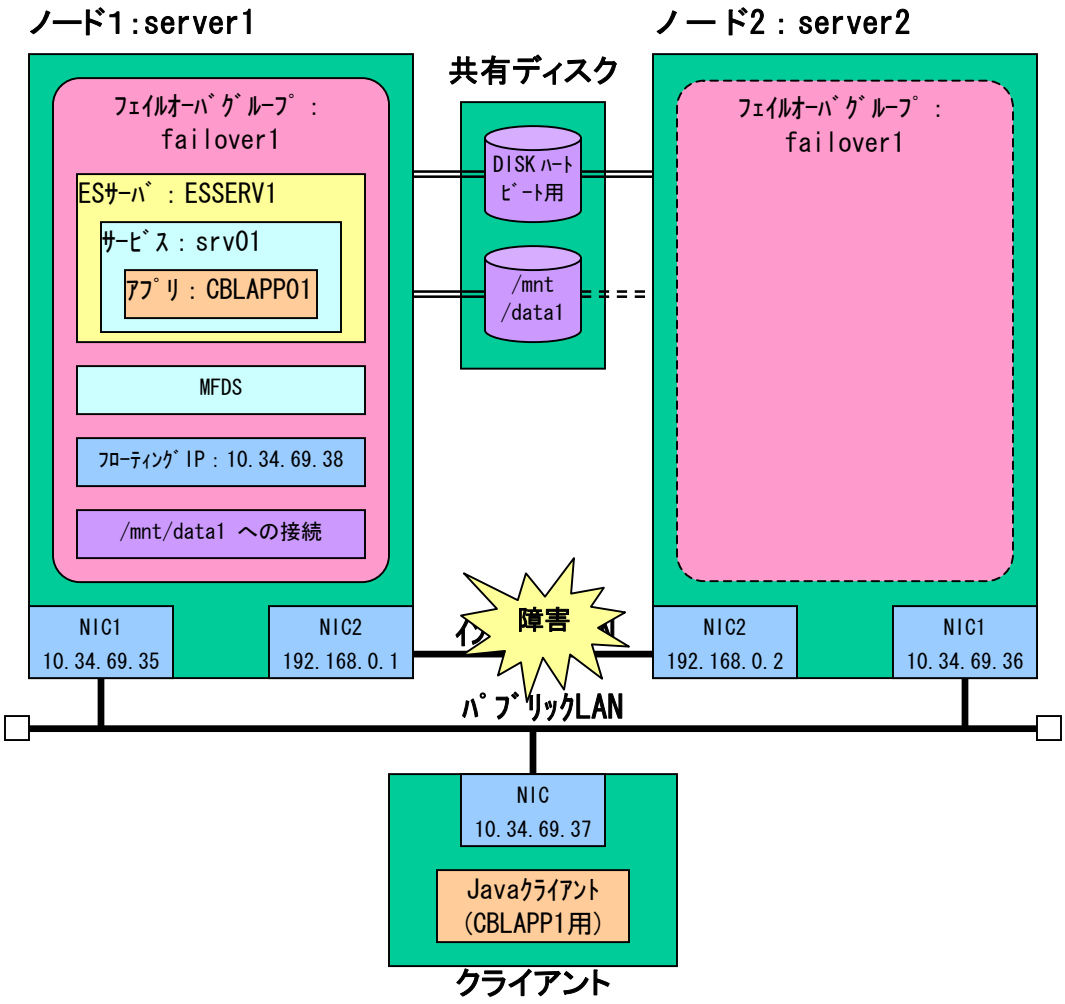
＜結果＞

ユーザ作成の監視シェルにより検知した障害によるフェイルオーバグループの移動を確認。

パターン④ OS のみで待機（シングルスタンバイ）

＜目的＞
待機側のノードで Enterprise Server の関連プロセスは起動しない状態からのフェイルオーバを確認する。
フェイルオーバ前と後で同じサービスが継続される事も確認する。

＜構成＞



＜設定のポイント＞

（パターン①～③の構成と同じ）

failover1 の開始シェルに組み込む起動コマンド

```
mfds &
sleep 15
casstart /rESSERV1
```

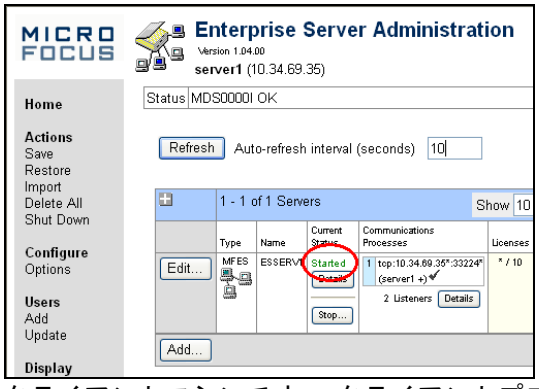
failover1 の終了シェルに組み込む停止コマンド

```
casstop /rESSERV1
killall -9 mfds32
```

クラスタの設定にインタコネクト LAN の登録をする。

＜検証＞

server1 でフェイルオーバグループ failover1 を起動している初期状態にする。

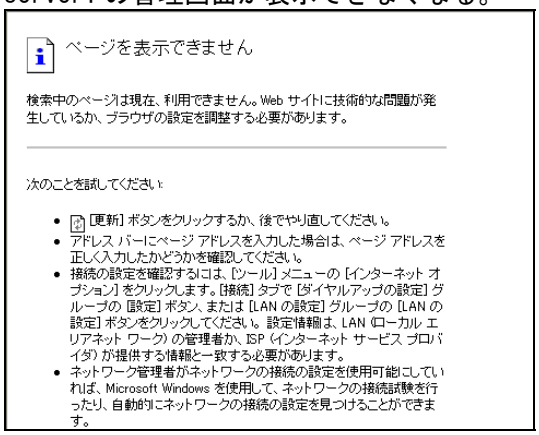


クライアントマシンで Java クライアントプログラムを定周期で実行を繰り返す。
シェル例(RunjavaTest.sh)

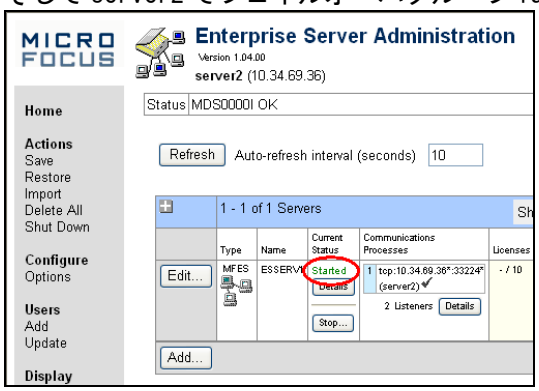
```
#!/bin/sh
while [ "1" == "1" ]; do
java -
classpath .:/home/clinux2/test/mfcobol.jar:/home/clinux2/test/mfconnector.jar:/home/clinux2/test/j2ee.jar clptest
#sleep 10
done
```

インターコネクト LAN のケーブルを引き抜く。

すると、フェイルオーバグループ failover1 の停止処理により mfds が止まるため server1 の管理画面が表示できなくなる。



そして server2 でフェイルオーバグループ failover1 が開始される。



クライアントプログラム実行結果

```
[root@gw test]# ./RunjavaTest.sh
06/03/14 15:42:18.07 CBLAPP01 server1 128
06/03/14 15:42:20.54 CBLAPP01 server1 129
06/03/14 15:42:22.87 CBLAPP01 server1 130
06/03/14 15:42:25.21 CBLAPP01 server1 131
06/03/14 15:42:27.53 CBLAPP01 server1 132
06/03/14 15:42:29.82 CBLAPP01 server1 133          server1 でサービスが稼動
06/03/14 15:42:32.12 CBLAPP01 server1 134
06/03/14 15:42:34.43 CBLAPP01 server1 135
06/03/14 15:42:36.72 CBLAPP01 server1 136
06/03/14 15:42:39.02 CBLAPP01 server1 137
06/03/14 15:42:41.35 CBLAPP01 server1 138
06/03/14 15:42:43.64 CBLAPP01 server1 139
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)        サービス中断中
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:452)
        at java.net.Socket.connect(Socket.java:402)
        at java.net.Socket.<init>(Socket.java:309)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:47)
                        :
                  (  繰り返し・中略  )
                        ;
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=45103.0)for 10.34.69.38:9003 (cause class:java.net.ConnectException,
cause:Connection refused) executing srv01.CBLAPP01
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:452)
        at java.net.Socket.connect(Socket.java:402)
        at java.net.Socket.<init>(Socket.java:309)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:47)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=268.0)for 10.34.69.38:9003 (cause class:java.net.ConnectException,
cause:Connection refused) executing srv01.CBLAPP01
06/03/14 15:43:55.76 CBLAPP01 server2 140
06/03/14 15:43:58.07 CBLAPP01 server2 141
06/03/14 15:44:00.37 CBLAPP01 server2 142
06/03/14 15:44:02.67 CBLAPP01 server2 143
06/03/14 15:44:04.98 CBLAPP01 server2 144
06/03/14 15:44:07.28 CBLAPP01 server2 145          server2 でサービスを引継ぎ。ﾃﾞｰﾀﾌｧｲﾙの連番も継続
06/03/14 15:44:09.59 CBLAPP01 server2 146
06/03/14 15:44:11.91 CBLAPP01 server2 147
06/03/14 15:44:14.19 CBLAPP01 server2 148
06/03/14 15:44:16.48 CBLAPP01 server2 149
06/03/14 15:44:37.26 CBLAPP01 server2 157

[root@gw test]#
```

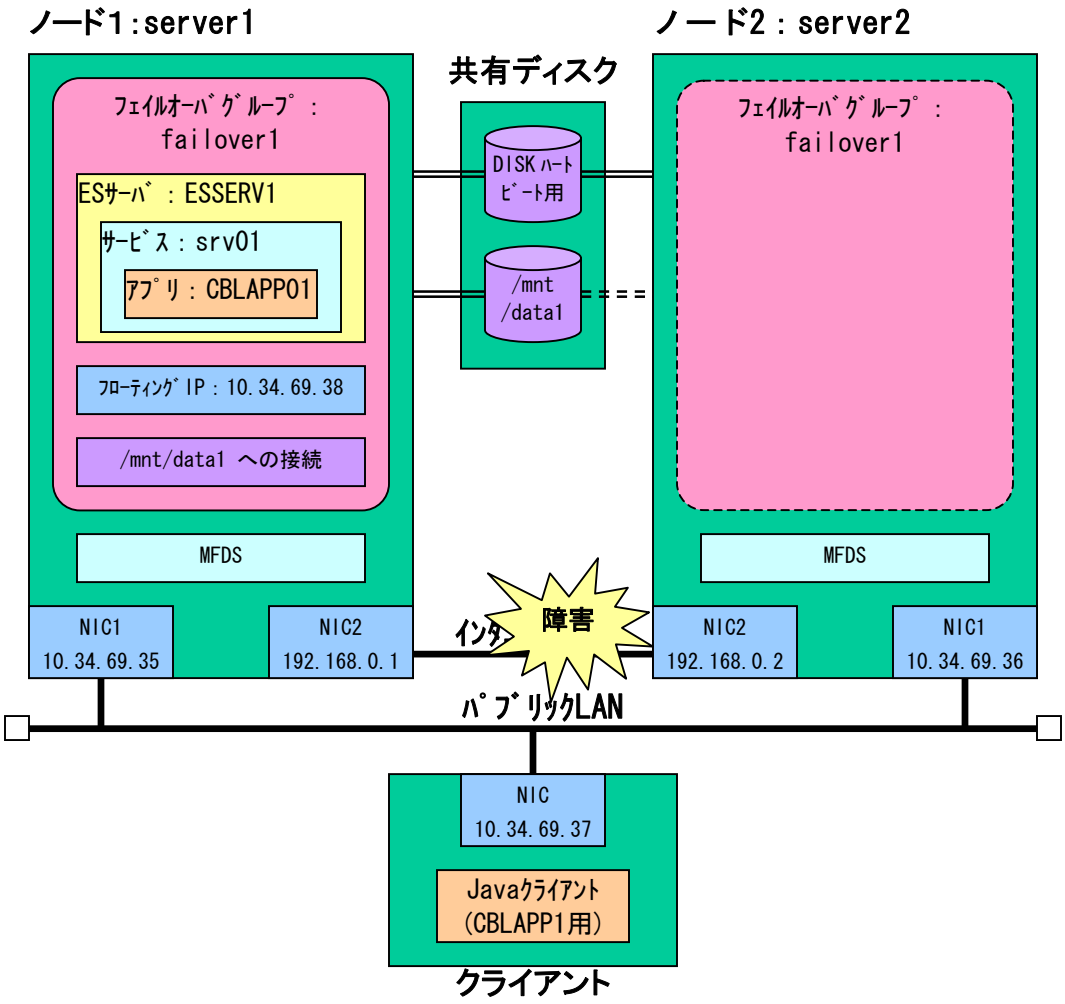
＜結果＞

待機側のノードで Enterprise Server の関連プロセスは起動しない状態(シングルスタンバイ)からのフェイルオーバグループの移動を確認。
また、一定時間のサービス中断後、同じサービスを待機側で提供できていることも確認。

パターン⑤ MFDS 起動で待機（高速スタンバイ１）

＜目的＞
待機側のノードで Enterprise Server の基本プロセス MFDS のみ起動した状態からのフェイルオーバを確認する。
フェイルオーバ前と後で同じサービスが継続される事も確認する。

＜構成＞



＜設定のポイント＞

それぞれ各ノードであらかじめ MFDS を立ち上げておく。
MFDS の制御はフェイルオーバグループに含めない。

failover1 の開始シェルに組み込む起動コマンド

```
casstart /rESSERV1
```

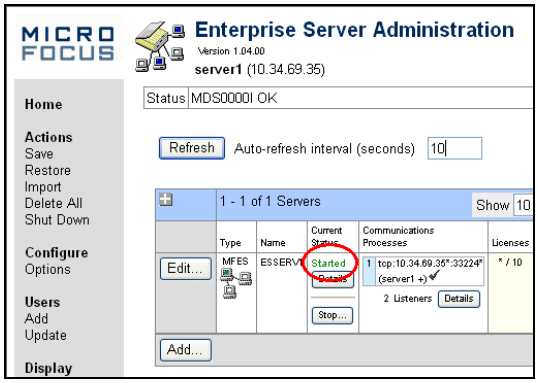
failover1 の終了シェルに組み込む停止コマンド

```
casstop /rESSERV1
```

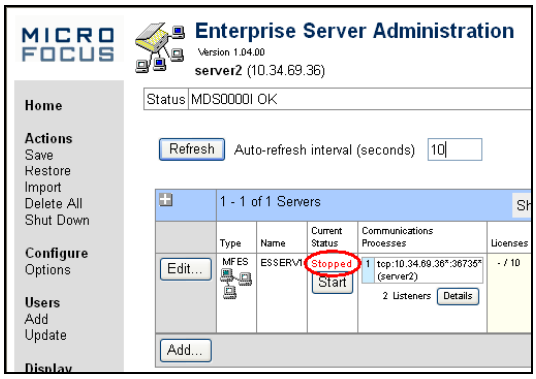
クラスタの設定にインタコネクト LAN の登録をする。

＜検証＞

server1 でフェイルオーバグループ failover1 を起動している初期状態にする。



server2 ではフェイルオーバグループ failover1(ES サーバ ESSERV1)は停止している。

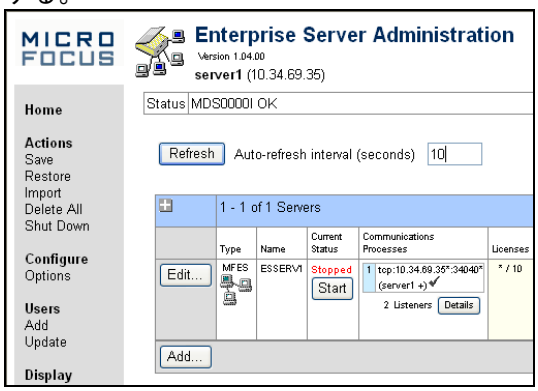


クライアントマシンで Java クライアントプログラムを定周期で実行を繰り返す。
シェル例(RunjavaTest.sh)

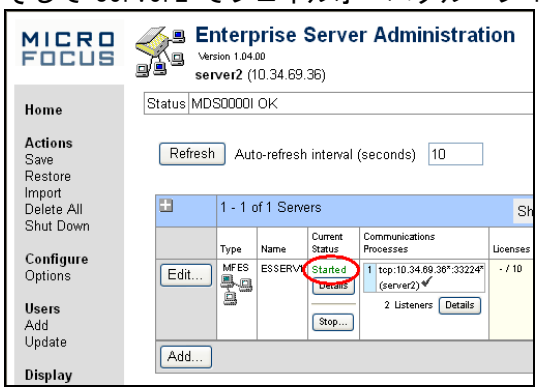
```
#!/bin/sh
while [ "1" == "1" ]; do
java -
classpath .:/home/clinux2/test/mfcobol.jar:/home/clinux2/test/mfconnector.jar:/home/clinux2/t
est/j2ee.jar clptest
#sleep 10
done
```

インターコネクト LAN のケーブルを引き抜く。

すると、フェイルオーバグループ failover1 の停止処理により ES サーバ ESSERV1 が停止する。



そして server2 でフェイルオーバグループ failover1(ES サーバ ESSERV1)が開始される。



クライアントプログラム実行結果

```
[root@gw test]# ./RunjavaTest.sh
06/03/14 15:42:18.07 CBLAPP01 server1 128
06/03/14 15:42:20.54 CBLAPP01 server1 129
06/03/14 15:42:22.87 CBLAPP01 server1 130
06/03/14 15:42:25.21 CBLAPP01 server1 131
06/03/14 15:42:27.53 CBLAPP01 server1 132
06/03/14 15:42:29.82 CBLAPP01 server1 133
06/03/14 15:42:32.12 CBLAPP01 server1 134
06/03/14 15:42:34.43 CBLAPP01 server1 135
06/03/14 15:42:36.72 CBLAPP01 server1 136
06/03/14 15:42:39.02 CBLAPP01 server1 137
06/03/14 15:42:41.35 CBLAPP01 server1 138
06/03/14 15:42:43.64 CBLAPP01 server1 139
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:452)
        at java.net.Socket.connect(Socket.java:402)
        at java.net.Socket.<init>(Socket.java:309)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:47)
                        :
                (  繰り返し・中略  )
                        ;
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=45103.0)for 10.34.69.38:9003 (cause class:java.net.ConnectException,
cause:Connection refused) executing srv01.CBLAPP01
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:452)
        at java.net.Socket.connect(Socket.java:402)
        at java.net.Socket.<init>(Socket.java:309)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
```

server1 でサービスが稼動

サービス中断中

```
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:47)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=268.0)for 10.34.69.38:9003 (cause class:java.net.ConnectException,
cause:Connection refused) executing srv01.CBLAPP01
06/03/14 15:43:55.76 CBLAPP01 server2 140
06/03/14 15:43:58.07 CBLAPP01 server2 141
06/03/14 15:44:00.37 CBLAPP01 server2 142
06/03/14 15:44:02.67 CBLAPP01 server2 143
06/03/14 15:44:04.98 CBLAPP01 server2 144
06/03/14 15:44:07.28 CBLAPP01 server2 145
06/03/14 15:44:09.59 CBLAPP01 server2 146
06/03/14 15:44:11.91 CBLAPP01 server2 147
06/03/14 15:44:14.19 CBLAPP01 server2 148
06/03/14 15:44:16.48 CBLAPP01 server2 149
06/03/14 15:44:37.26 CBLAPP01 server2 157

[root@gw test]#
```

server2 でサービスを引継ぎ。ﾃﾞｰﾀﾌｧｲﾙの連番も継続

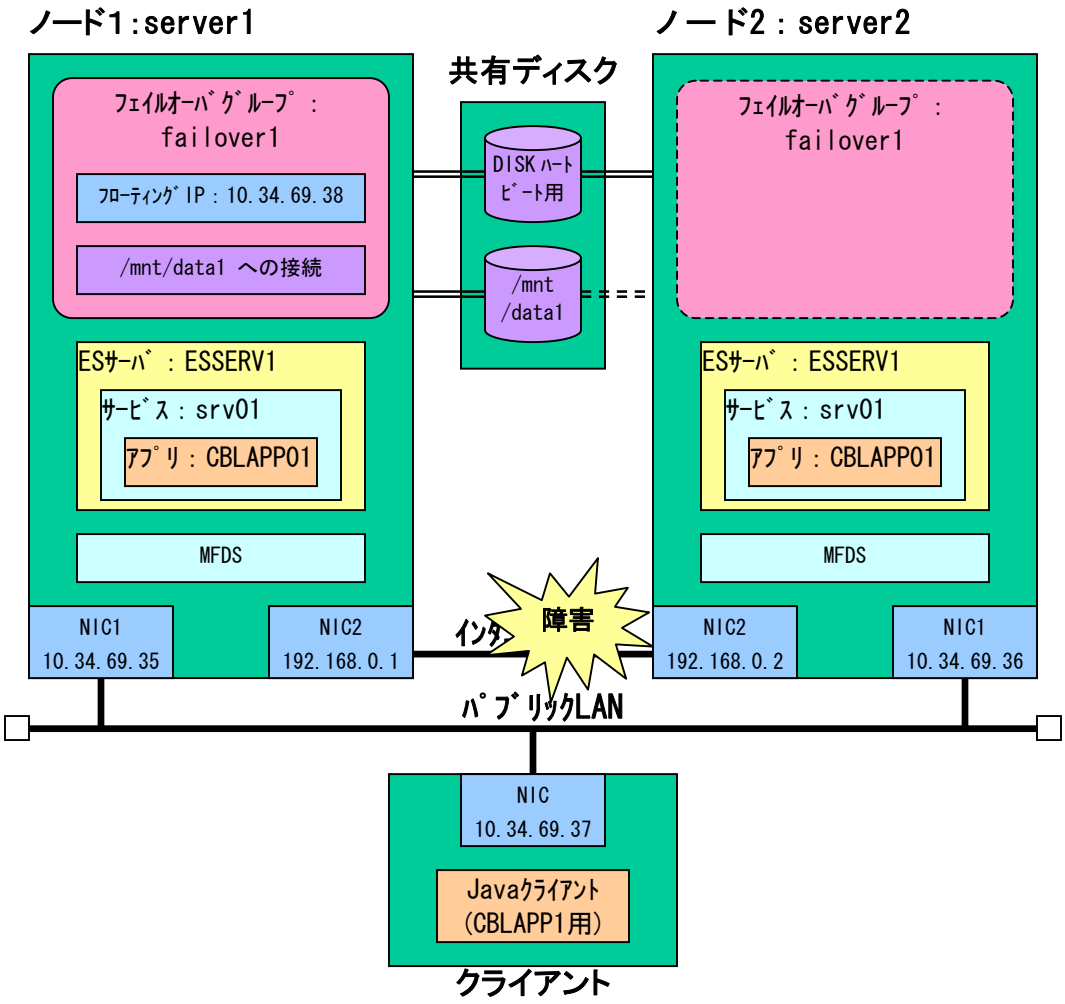
＜結果＞

待機側のノードで Enterprise Server の基本プロセス MFDS のみ起動した状態からのフェイルオーバグループの移動を確認。
また、一定時間のサービス中断後、同じサービスを待機側で提供できていることも確認。

パターン⑥ ES サーバ起動で待機（高速スタンバイ 2）

＜目的＞
待機側のノードで Enterprise Server の基本プロセス MFDS と ES サーバを起動した状態からのフェイルオーバを確認する。
フェイルオーバ前と後で同じサービスが継続される事も確認する。

＜構成＞

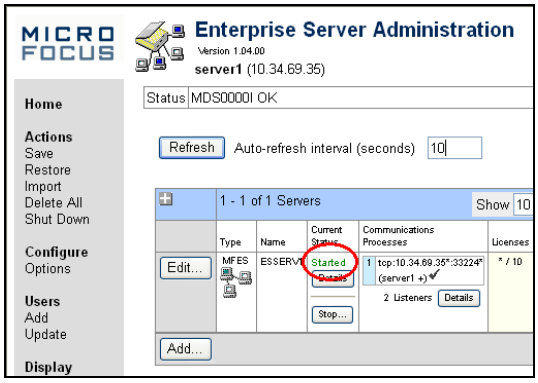


＜設定のポイント＞

それぞれ各ノードであらかじめ MFDS を立ち上げておく。
フェイルオーバグループではフローティング IP アドレスと共有ディスクの切り替えのみを行なう。

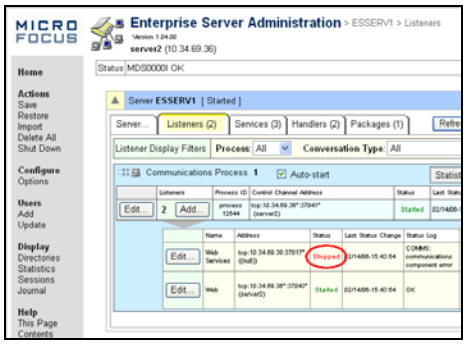
クラスタの設定にインタコネクト LAN の登録をする。

＜検証＞
server1 で手動で ESSERV1 を起動する。



server2 で手動で ESSERV1 を起動する。

<u>ところがserver2 で手動でESSERV1 を立ち上げようとするとリスナーが立ち上がらない。</u>



理由：Enterprise Server のリスナーは指定した IP アドレスが有効になっている必要がある。

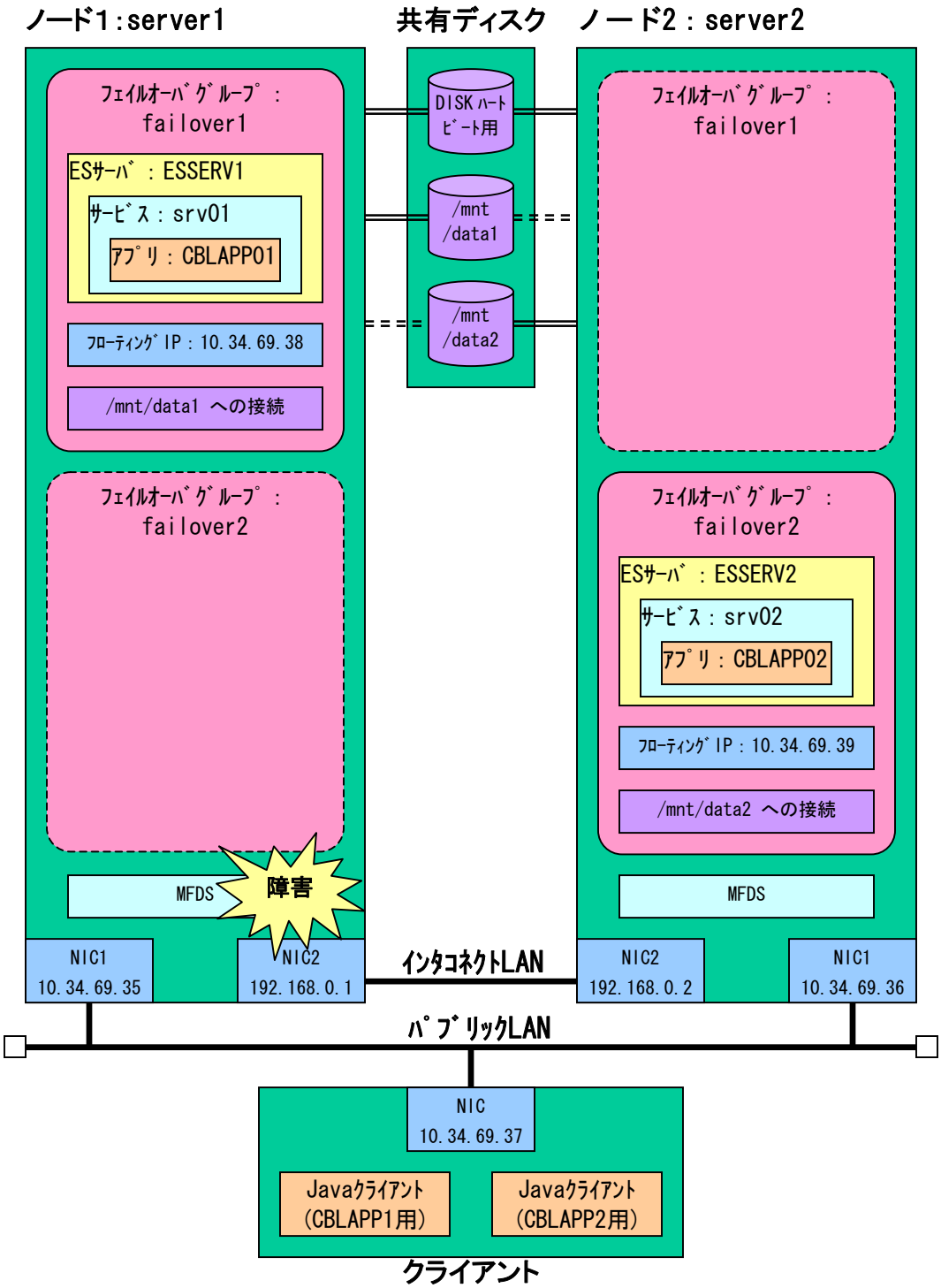
＜結果＞

Enterprise Server の制限事項により、待機側の ES サーバを正常に立ち上がらせることができないため、本パターンの前提条件を満たせなかった。
よって本パターンでの動作検証は不可。

パターン⑦ 別の業務を提供しながら待機（相互スタンバイ１）

＜目的＞
待機側のノードで Enterprise Server の MFDS を起動し、別の ES サーバによるサービスを起動した状態からのフェイルオーバを確認する。
フェイルオーバ前と後で同じサービスが継続される事も確認する。

＜構成＞



＜設定のポイント＞

それぞれ各ノードであらかじめ MFDS を立ち上げておく。
MFDS の制御はフェイルオーバグループに含めない。

failover1 の開始シェルに組み込む起動コマンド

```
casstart /rESSERV1
```

failover1 の終了シェルに組み込む停止コマンド

```
casstop /rESSERV1
```

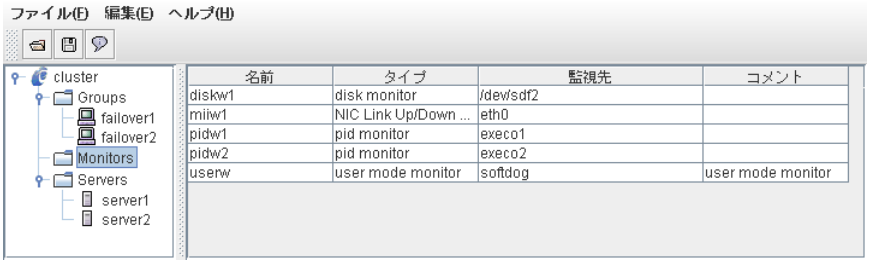
failover2 の開始シェルに組み込む起動コマンド

```
casstart /rESSERV2
```

failover1 の終了シェルに組み込む停止コマンド

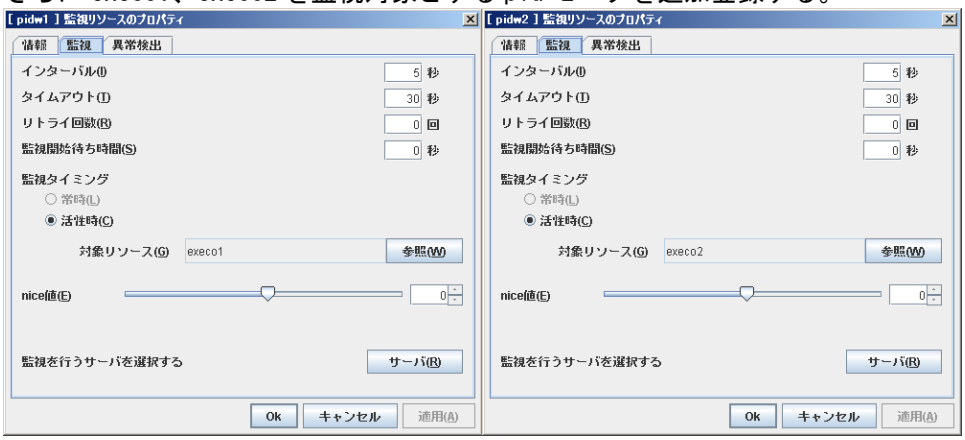
```
casstop /rESSERV2
```

failover1 に監視用の execute resource、execo1 と execo2 を追加



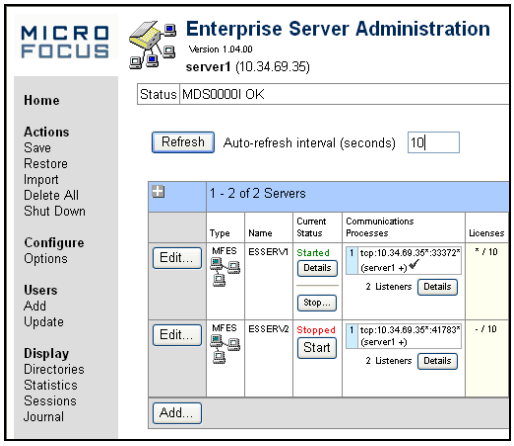
execo1 に登録するシェルはパターン③と同様。
execo2 のシェルは mdump コマンドの箇所の ESSERV1 を ESSERV2 に変更したもの。

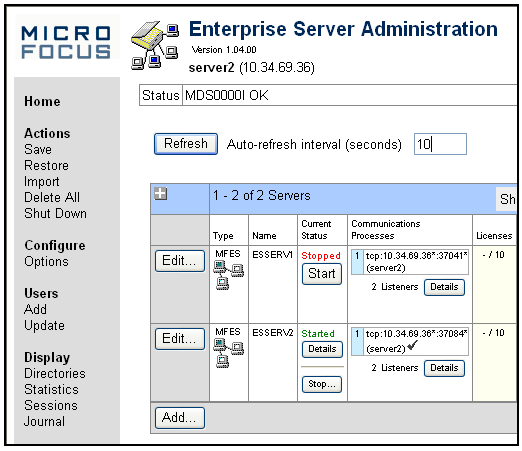
さらに execo1、execo2 を監視対象とする pid モニタを追加登録する。



＜検証＞

server1 でフェイルオーバグループ failover1 を、server2 でフェイルオーバグループ failover2 を開始しておく。





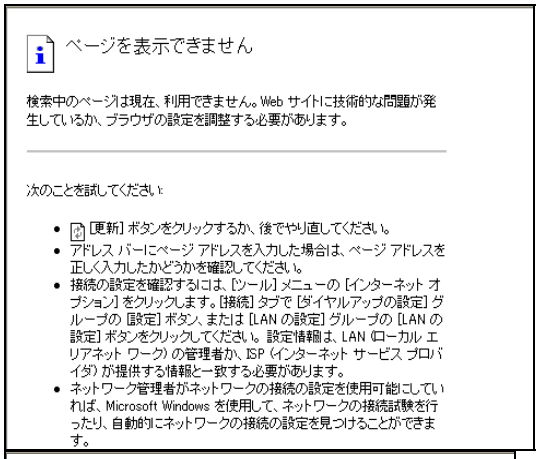
クライアントマシンで各サービスに対する Java クライアントプログラムを定周期で実行を繰り返す。(2 サービス同時に呼び出す)

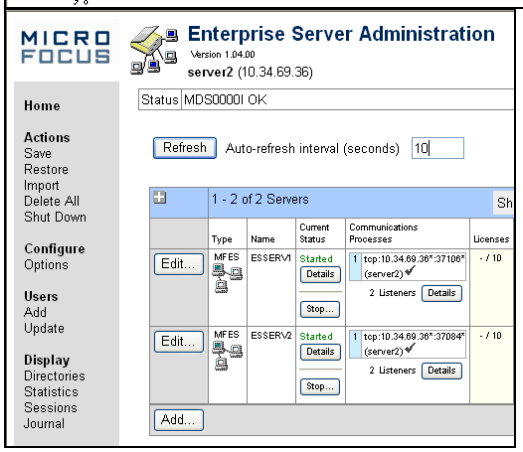
シェル例(RunjavaTest.sh)

```
#!/bin/sh
while [ "1" == "1" ]; do
java -
classpath .:/home/clinux2/test/mfcobol.jar:/home/clinux2/test/mfconnector.jar:/home/clinux2/test/j2ee.jar clptest
#sleep 10
done
```

次に server1 で MFDS を直接手動で停止する。
> killall -9 mfds32

すると、mfds のプロセス異常により exec01 の監視シェルが障害を検知しファイルオーバが開始する。
server1 ではサービスが停止し、server2 でフェイルオーバグループ failover1 が開始され ESSERV1 が起動する。


ページを表示できません
検索中のページは現在、利用できません。Web サイトに技術的な問題が発生しているか、ブラウザの設定を調整する必要があります。
次のことを試してください:
[更新] ボタンをクリックするか、後でやり直してください。
アドレス バーにページ アドレスを入力した場合は、ページ アドレスを正しく入力したかどうかを確認してください。
接続の設定を確認するには、[ツール] メニューの [インターネット オプション] をクリックします。[接続] タブで [ダイヤルアップの設定] グループの [設定] ボタン、または [LAN の設定] グループの [LAN の設定] ボタンをクリックしてください。設定情報は、LAN (ローカル エリアネット ワーク) の管理者か、ISP (インターネット サービス プロバイダ) が提供する情報と一致する必要があります。
ネットワーク管理者がネットワークの接続の設定を使用可能にしていれば、Microsoft Windows を使用して、ネットワークの接続試験を行ったり、自動的にネットワークの接続の設定を見つけることができます。


MICRO FOCUS
Enterprise Server Administration
Version 1.04.00
server2 (10.34.69.36)
Home
Actions
Save
Restore
Import
Delete All
Shut Down
Configure
Options
Users
Add
Update
Display
Directories
Statistics
Sessions
Journal
Status MDS0000I OK
Refresh
Auto-refresh interval (seconds) 10
1 - 2 of 2 Servers
Type
Name
Current Status
Communications Processes
Licenses
Edit...
MFES
ESSERV1
Started
Details
Stop...
1 tcp:10.34.69.36*:37106* (server2)
2 Listeners
Details
- / 10
Edit...
MFES
ESSERV2
Started
Details
Stop...
1 tcp:10.34.69.36*:37084* (server2)
2 Listeners
Details
- / 10
Add...

クライアントプログラム実行結果(CBLAPP01)

```
[root@gw test]# ./RunjavaTest.sh
06/03/14 17:57:25.73 CBLAPP01 server1 205
06/03/14 17:57:29.69 CBLAPP01 server1 206
06/03/14 17:57:34.30 CBLAPP01 server1 207
06/03/14 17:57:39.29 CBLAPP01 server1 208
06/03/14 17:57:43.50 CBLAPP01 server1 209
06/03/14 17:57:48.31 CBLAPP01 server1 210
06/03/14 17:57:53.05 CBLAPP01 server1 211
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:452)
        at java.net.Socket.connect(Socket.java:402)
        at java.net.Socket.<init>(Socket.java:309)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:47)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=449.0)for 10.34.69.38:9003 (cause class:java.net.ConnectException,
cause:Connection refused) executing srv01.CBLAPP01

                        :
                  (  繰り返し・中略  )
                        ;
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
        at java.net.Socket.connect(Socket.java:452)
        at java.net.Socket.connect(Socket.java:402)
        at java.net.Socket.<init>(Socket.java:309)
        at java.net.Socket.<init>(Socket.java:153)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:894)
        at com.microfocus.cobol.connector.transport.BINPStream.createSocketWithTimeout(BINPStream.java:825)
        at com.microfocus.cobol.connector.transport.BINPStream.<init>(BINPStream.java:97)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.open(BINPRemoteCall.java:176)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.connect(BINPRemoteCall.java:73)
        at com.microfocus.cobol.connector.transport.BINPRemoteCall.reconnect(BINPRemoteCall.java:145)
        at com.microfocus.cobol.connector.PureCobolBean.cobcall(PureCobolBean.java:285)
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:247)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:174)
        at clptest.main(clptest.java:47)
javax.resource.spi.EISSystemException: CobolException Exception throw during open (see getCause() for more
information) (elapsed time=513.0)for 10.34.69.38:9003 (cause class:java.net.ConnectException,
cause:Connection refused) executing srv01.CBLAPP01
06/03/14 17:58:58.36 CBLAPP01 server2 212
06/03/14 17:59:03.00 CBLAPP01 server2 213
06/03/14 17:59:07.97 CBLAPP01 server2 214
06/03/14 17:59:12.90 CBLAPP01 server2 215
06/03/14 17:59:17.31 CBLAPP01 server2 216
06/03/14 17:59:21.95 CBLAPP01 server2 217
06/03/14 17:59:26.79 CBLAPP01 server2 218
06/03/14 17:59:31.15 CBLAPP01 server2 219
06/03/14 17:59:35.30 CBLAPP01 server2 220
06/03/14 17:59:37.80 CBLAPP01 server2 221

[root@gw test]#
```

server1 でサービスが稼動

サービス中断中

server2 でサービスを引継ぎ。ﾃﾞｰﾀﾌｧｲﾙの連番も継続

クライアントプログラム実行結果(CBLAPP02)

```
[root@gw test]# ./RunjavaTest.sh
06/03/14 17:57:35.12 CBLAPP02 server2 7
06/03/14 17:57:39.78 CBLAPP02 server2 8
06/03/14 17:57:44.45 CBLAPP02 server2 9
06/03/14 17:57:49.29 CBLAPP02 server2 10
06/03/14 17:57:53.91 CBLAPP02 server2 11
06/03/14 17:57:58.59 CBLAPP02 server2 12
06/03/14 17:58:02.86 CBLAPP02 server2 13
06/03/14 17:58:07.94 CBLAPP02 server2 14
06/03/14 17:58:12.52 CBLAPP02 server2 15
06/03/14 17:58:16.86 CBLAPP02 server2 16
06/03/14 17:58:21.56 CBLAPP02 server2 17
06/03/14 17:58:26.20 CBLAPP02 server2 18
06/03/14 17:58:30.91 CBLAPP02 server2 19
06/03/14 17:58:35.69 CBLAPP02 server2 20
06/03/14 17:58:40.51 CBLAPP02 server2 21
06/03/14 17:58:45.39 CBLAPP02 server2 22
06/03/14 17:58:50.10 CBLAPP02 server2 23
06/03/14 17:58:54.73 CBLAPP02 server2 24
06/03/14 17:58:59.50 CBLAPP02 server2 25
06/03/14 17:59:04.21 CBLAPP02 server2 26
06/03/14 17:59:08.97 CBLAPP02 server2 27
06/03/14 17:59:13.52 CBLAPP02 server2 28
06/03/14 17:59:18.41 CBLAPP02 server2 29
06/03/14 17:59:23.06 CBLAPP02 server2 30
06/03/14 17:59:27.71 CBLAPP02 server2 31

[root@gw test]#
```

同様に同じ状態から server2 の MFDS を直接手動で停止し、server2 の ESSERV2 が server1 にフェイルオーバすることが確認する。

＜結果＞

待機側のノードで Enterprise Server の MFDS を起動し、別の ES サーバによるサービスを起動した状態からのフェイルオーバグループの移動を双方向で確認。
また、一定時間のサービス中断後、同じサービスを待機側で提供できていることも確認。

まとめ

本検証では代表的な異常検出例３パターンと待機状態例４パターンについて動作を確認しました。

ﾊﾟﾀｰﾝ	タイトル	テスト結果	説明
①	手動切り替え	○	CLUSTERPRO 標準機能でフェイルオーバ確認
②	ｲﾝﾀｰｺﾈｸﾄ障害	○	CLUSTERPRO 標準機能でフェイルオーバ確認
③	業務障害	○	MDUMP ユーティリティの出力を解析するシェルを自作してフェイルオーバ確認
④	OS のみで待機 （シングルスタンバイ）	○	MFDS 制御、ES サーバ制御をフェイルオーバグループに含めフェイルオーバ確認
⑤	MFDS 起動で待機 （高速スタンバイ 1）	○	ES サーバ制御をフェイルオーバグループに含めフェイルオーバ確認
⑥	DS サーバ起動で待機 （高速スタンバイ 2）	×	Enterprise Server のリスナーは定義する IP が立ち上げ時に実在しなければ立ち上がらないため、このパターンは実装不可
⑦	別の業務を提供しながら待機 （相互スタンバイ）	○	フェイルオーバグループを分けて別のサービスを継続的に提供しながらフェイルオーバ確認

異常検出のパターンでは、今回マイクロフォーカス提供の mdump ユーティリティにより
を使用し、ステータスを監視する事で異常を検出することが出来ました。
但し、本運用においては、ユーザプログラム又は関連ソフトウェアの監視なども含め、監視方法や監視対象を再検討する必要があります。

待機状態のパターンでは、待機系のノードでは ES サーバをあらかじめ立ち上げておく事ができない事が分かりました。
また MFDS は開始状態でも停止状態からもフェイルオーバは可能でした。
但し、複数のフェイルオーバグループで ES サーバを立ち上げるケース等では MFDS は ES サーバの起動前にあらかじめ起動しておく必要があるため、基本的には OS の起動時に MFDS を自動起動するように設定することをお奨めします。

付録

サーバアプリケーションソースサンプル(cblapp01.cbl)

```
       FILE-CONTROL.
           SELECT APPFILE ASSIGN TO EXTERNAL APPFILE01
                  ORGANIZATION INDEXED RECORD KEY RKEY
                  ACCESS MODE RANDOM
                  FILE STATUS IS AP-STATUS.
       DATA                DIVISION.
       FILE                SECTION.
       FD  APPFILE.
       01  AF-REC.
           05  RKEY        PIC X(6).
           05  AF-COUNTER   PIC 9(10).
       WORKING-STORAGE     SECTION.
       01  WK-DATE         PIC X(6).
       01  WK-TIME         PIC X(8).
       01  AP-STATUS       pic x(02).
       LINKAGE             SECTION.
       01  LK-DATETIME     PIC X(20).
       01  LK-PGMNAME      PIC X(10).
       01  LK-HOSTNAME     PIC X(20).
       01  LK-COUNTER      PIC 9(10).
       PROCEDURE           DIVISION
             USING LK-DATETIME LK-PGMNAME LK-HOSTNAME LK-COUNTER.
       1.
           PERFORM    GET-DATETIME.
           MOVE       'CBLAPP01  '   TO LK-PGMNAME.
           CALL       'gethostname' USING LK-HOSTNAME BY VALUE 20.
           OPEN       I-O            APPFILE.
           MOVE       SPACE          TO RKEY.
           READ       APPFILE        INVALID KEY CONTINUE.
           MOVE       AF-COUNTER     TO LK-COUNTER.
           ADD        1              TO AF-COUNTER.
           REWRITE    AF-REC         INVALID KEY CONTINUE.
           CLOSE      APPFILE.
           EXIT       PROGRAM.

       GET-DATETIME.
           ACCEPT  WK-DATE  FROM DATE.
           ACCEPT  WK-TIME  FROM TIME.
           STRING  WK-DATE(1:2) '/' WK-DATE(3:2) '/' WK-DATE(5:2) ' '
                   WK-TIME(1:2) ':' WK-TIME(3:2) ':' WK-TIME(5:2) '.'
                   WK-TIME(7:2) DELIMITED BY SIZE INTO LK-DATETIME.
```