



Hinemos HULFT

連携ノウハウと適用例 公開日：2012/11/5

目次

1	はじめに	6
2	免責事項	6
3	目指す姿	7
3.1	背景	7
3.2	連携の目的	8
3.3	連携による解決方法	8
4	事前準備	9
4.1	Hinemosの必須設定項目	9
4.2	HULFTの必須設定項目	10
4.3	その他の事前準備	10
5	連携シナリオ	11
5.1	シナリオの想定	11
5.2	シナリオ1：ファイル転送設定の登録	12
5.2.1	シナリオ概要	12
5.2.2	手順1：ファイル転送ジョブユニットの登録	13
5.2.3	手順2：マスタファイル作成ジョブの登録	13
5.2.4	手順3：配信管理情報登録ジョブの登録	13
5.2.5	手順4：ファイル転送ジョブの登録	15
5.2.6	手順5：スケジュールの登録	17
5.3	シナリオ2：ファイル転送結果の閲覧	18
5.3.1	シナリオ概要	18
5.3.2	ファイル転送に成功した場合	18
5.3.3	ファイル転送に失敗した場合	20
5.4	シナリオ3：ファイル転送失敗時の再配信指示	22
5.4.1	シナリオ概要	22
5.4.2	再配信指示	22
6	付録A	24
6.1	サンプルプログラムの構成	24
6.2	サンプルプログラムで使用するHULFTのエラーコード・ファイル	25
6.2.1	HULFTのエラーコード	25
6.2.2	HULFTのファイル	26
6.3	HulftSend.java	27
6.3.1	プログラム概要	27
6.3.2	配信管理情報のパラメータファイル作成	28
6.3.3	配信管理情報の登録	29
6.4	HulftFileTransferJob.java	30
6.4.1	プログラム概要	30
6.4.2	集信管理情報のパラメータファイル作成	35
6.4.3	集信管理情報の登録	35
6.4.4	送信要求コマンドの実行	35
6.4.5	エラーコードを取得	35
6.4.6	詳細メッセージの表示	35

7	付録B	37
7.1	連携にあたっての検討内容	37

本ソフトウェアは独立行政法人情報処理推進機構(IPA)の2004年度下期オープンソースソフトウェア活用基盤整備事業の委託を受けて開発しました。テーマ名は「分散ファシリティ統合マネージャの開発」です。

<http://www.ipa.go.jp/software/open/2004/result.html>

本ドキュメントについては、GPLの下で配布されていません。

本ドキュメントの使用には、以下の条件が適用されます。

- 内容の変更、編集は許可されません。
- 例えば印刷物の販売や出版物に本ドキュメントの一部を記載する場合は、事前に株式会社NTTデータとの契約・合意を必要とします。

1 はじめに

本ドキュメントでは、統合運用管理ソフトウェアHinemosとファイル転送・データ連携ツールHULFTの連携手順について説明します。本ドキュメントで用いたソフトウェア、OSを下記に記載します。

【ソフトウェア】

- Hinemos ver4.0.0
- HULFT 7

【OS】

- Windows Server 2008 R2
- Red Hat Enterprise Linux 6.2

2 免責事項

本ドキュメントでの設定は一例であり、実際に使用される際はご利用の環境の構成、各ソフトのバージョン、運用設計等に沿って設定を変更していただけない場合、適用できません。本ソフトウェア、ドキュメント及びサンプルプログラムの使用により生じたいかなる損害に対しても、弊社は一切の責任を負いません。

3 目指す姿

3.1 背景

1つの業務要件を実現するため複数のシステム間でデータを連携する環境では、Hinemosだけでは機能不足であるため、安全にデータ転送を行う機能を持ったHULFTを利用することがよくあります。

一方で 現在HinemosとHULFTの両方を利用する環境では、以下のような課題があります。

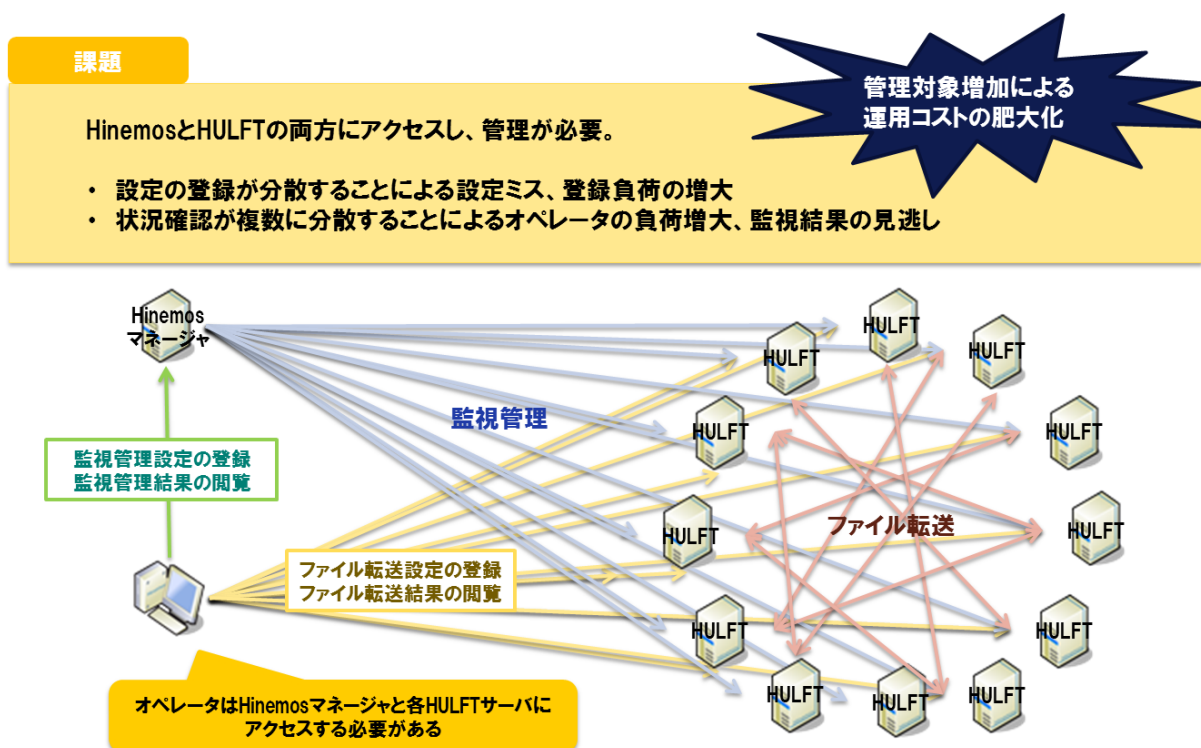


図1, HinemosとHULFTの両方を利用する環境の課題

3.2 連携の目的

本連携により、HinemosからHULFTによるシステム間のデータ連携機能をシームレスに利用可能となることで、運用対象の単一化によるコスト低減を実現します。

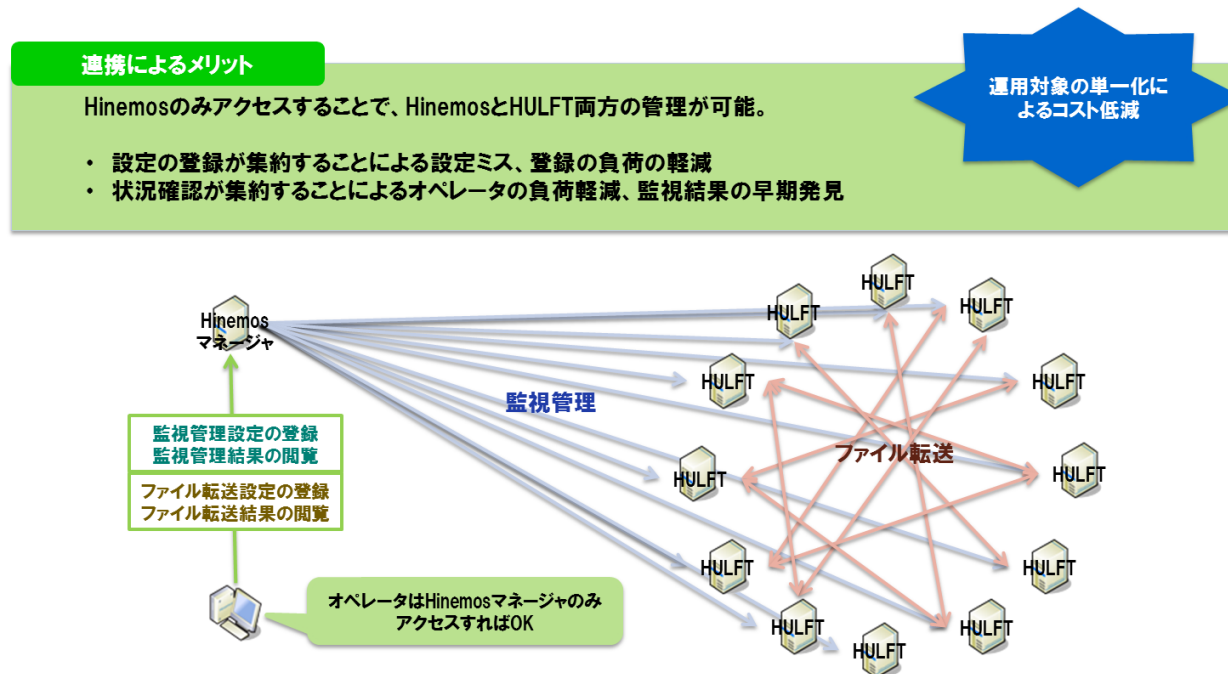


図2, Hinemos-HULFT連携によるメリット

3.3 連携による解決方法

Hinemosのジョブ管理機能によりジョブネットやジョブスケジュールの登録を行うことで、HinemosとHULFTの連携を実現します。本連携では、エラーメッセージの確認もHinemosの画面上で行えます。

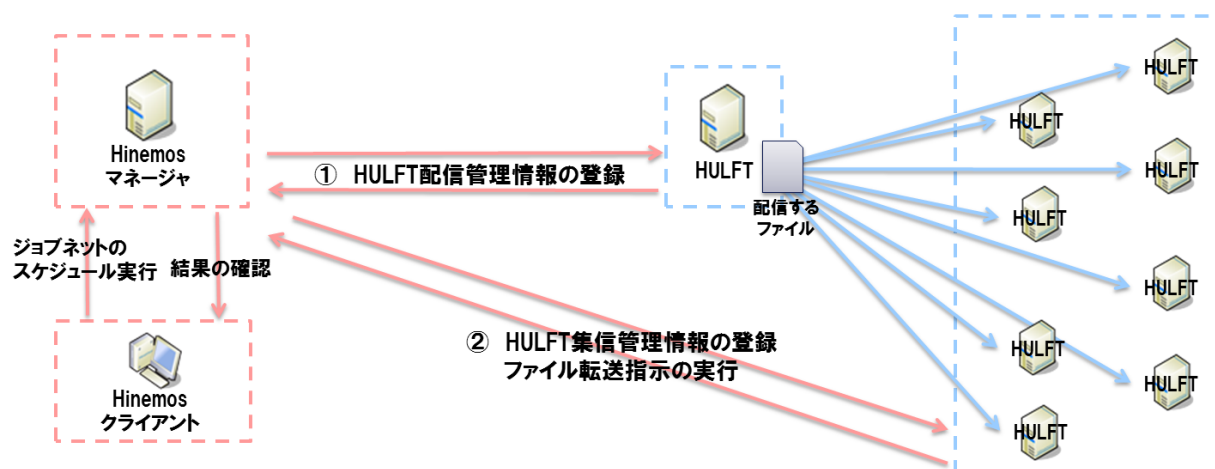


図3, 連携による解決方法(Hinemosジョブ管理機能の活用)

4 事前準備

4.1 Hinemosの必須設定項目

まず、Hinemosマネージャを運用管理サーバ、Hinemosエージェントをファイルの送信元と送信先、Hinemosクライアントを運用管理端末にインストールします。

インストールの詳細は、下記マニュアルをご参照ください。

- ・「Hinemos ver4.0 インストールマニュアル 第1.1版」

Hinemosのインストール後、事前に設定が必要な項目は下記のとおりです。

1. ノード登録

ファイルの送信先、送信元をあらかじめノードとして登録します。

ノード登録の詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 3.4.1 ノード情報の作成」

2. スコープ登録

OSごとにジョブの設定が異なるため、OSごとにスコープを登録します。

スコープ登録の詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 3.6.1 スコープの作成」

3. スコープへのノード割当て

登録したスコープに、対応するOSのノードを割り当てます。

スコープへのノード割当ての詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 3.7.1 ノードの割当て」

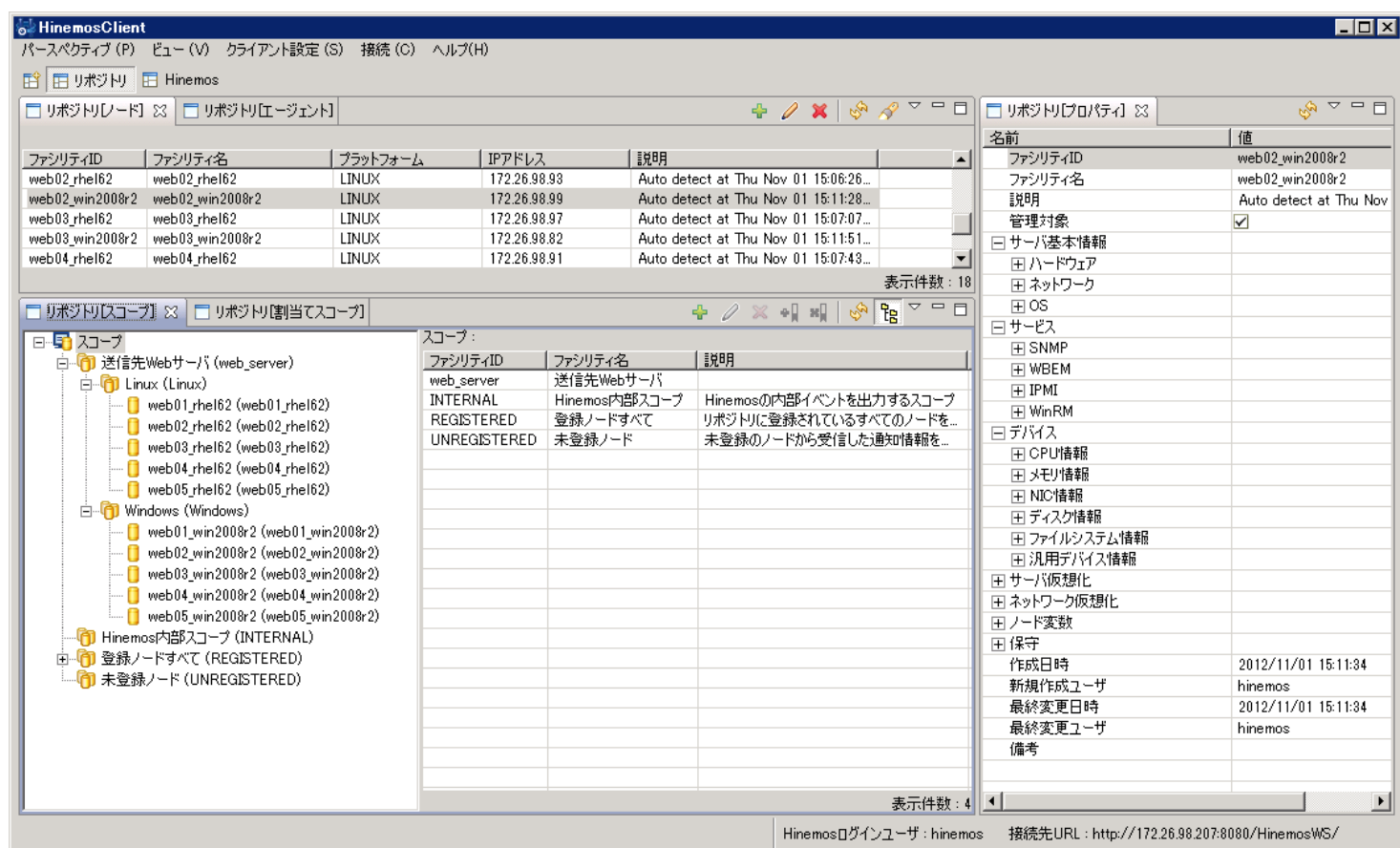


図4, スコープ登録、ノード割当て時の例

4.2 HULFTの必須設定項目

まず、HULFTをファイルの送信元と送信先にそれぞれインストールします。

インストールの詳細は、下記マニュアルをご参照ください。

- Windowsの場合：「HULFT7 Windows 導入マニュアル」
- Linuxの場合：「HULFT7 UNIX/Linux 導入マニュアル」

HULFTのインストール後、事前に設定が必要な項目は下記のとおりです。

1. 操作ログ出力選択の設定

ファイル転送結果の取得時にコマンド実行ログ(huloplcmd.csv)を使用するため、コマンド操作ログを出力する設定に変更する必要があります。デフォルトの設定ではログは出力されません。

操作ログ出力選択の詳細は、下記マニュアルをご参照ください。

Windowsの場合：「HULFT7 Windows アドミニストレーション・マニュアル 3.4 システム動作環境の設定について」

Linuxの場合：「HULFT7 UNIX/Linux アドミニストレーション・マニュアル 3.4 システム動作環境の設定について」

2. 詳細ホスト情報登録

ファイルの送信元で、送信要求を受け付けるホスト(ファイルの送信先)を登録します。また、ファイルの送信先で、送信要求を送るホスト(ファイルの送信元)を登録します。

詳細ホスト情報登録の詳細は、下記マニュアルをご参照ください。

Windowsの場合：「HULFT7 Windows オペレーション・マニュアル 2.1.4 詳細ホスト情報」

Linuxの場合：「HULFT7 UNIX/Linux オペレーション・マニュアル 2.1.4 詳細ホスト情報」

3. 転送グループ登録

ファイルの送信元で、ホストを指定して転送グループを登録します。登録した転送グループIDは、ファイルの送信元に配信管理情報を登録する際、利用します。

転送グループ登録の詳細は、下記マニュアルをご参照ください。

Windowsの場合：「HULFT7 Windows オペレーション・マニュアル 2.1.5 転送グループ情報」

Linuxの場合：「HULFT7 UNIX/Linux オペレーション・マニュアル 2.1.5 転送グループ情報」

4. エラー定義ファイルの配置

ファイルの送信先がWindowsの場合、ファイルの送信先で、UNIX/Linux版エラー定義ファイル(UX.dat)を任意のフォルダに配置します。なお、Windows版では、Windows版用エラー定義ファイル(NT.dat)は、インストール時にあらかじめ配置されています。また、ファイルの送信先がLinuxの場合、ファイルの送信先で、Windows版用エラー定義ファイル(NT.dat)と、UNIX/Linux版エラー定義ファイル(UX.dat)の両方を任意のフォルダに配置します。

エラー定義ファイルは、HULFTのエラーコードに対するエラーメッセージを格納したファイルです。HULFTの保守契約者様向けに公開しており、下記「HULFT技術サポートサイト」よりダウンロードいただけます。

<https://tech.hulft.com/support/login/login.jsp>

なお、HULFTの詳細ホスト情報のホスト名とHinemosのノードのファシリティIDは別物のため、同一の名称にしても互換性がない点を意識して命名する必要があります。

4.3 その他の事前準備

その他、サンプルプログラムを使用する場合、事前に必要となる準備は下記のとおりです。

1. サンプルプログラムの配置

ジョブ実行のため、サンプルプログラムをコンパイルし、事前に配置します。ファイルの送信元には、HulftSend.class (Windowsの場合は、HulftSend.batも)、ファイルの送信先には、HulftFileTransferJob.class (Windowsの場合は、HulftFileTransferJob.batも) を任意のフォルダに配置します。

5 連携シナリオ

5.1 シナリオの想定

本ドキュメントにおける、シナリオの想定は下記のとおりです。

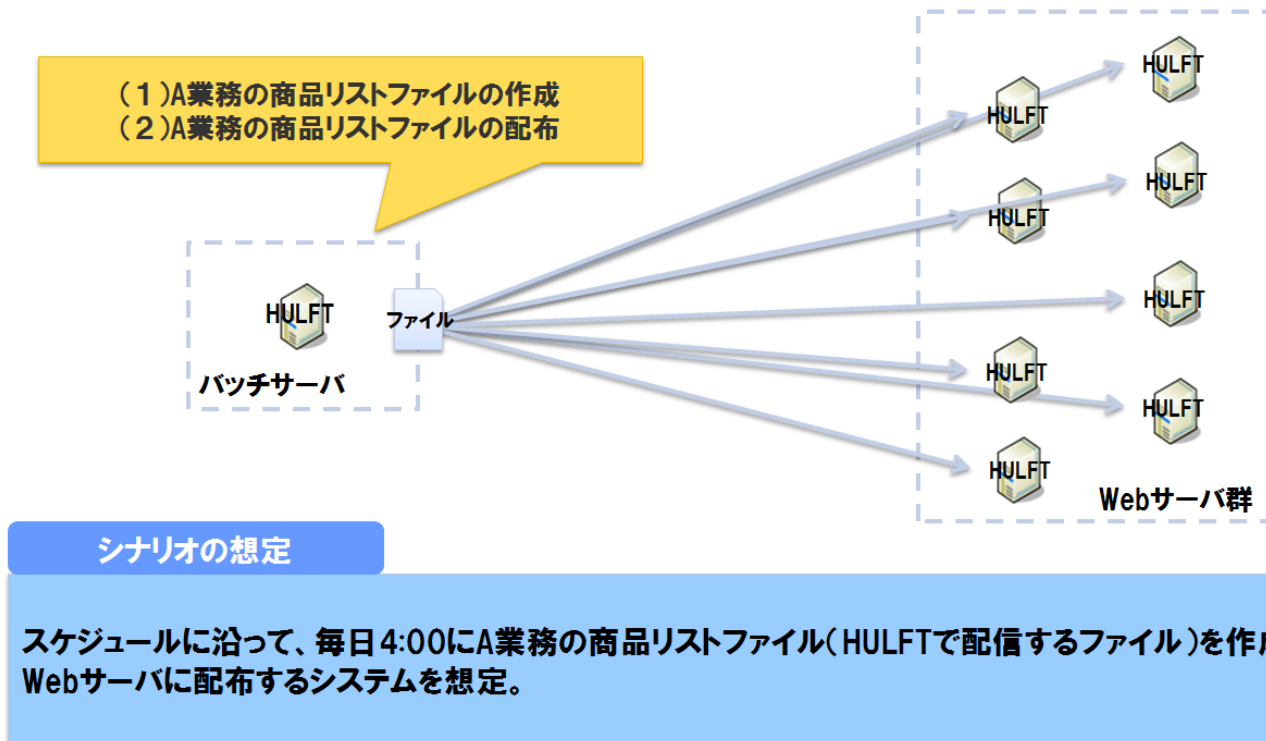


図5, シナリオの想定

シナリオの想定に沿うよう、下記1～3の具体的なシナリオで連携を実施します。

- ・ シナリオ1：ファイル転送設定の登録
Hinemosのジョブ管理機能で、ファイル転送のジョブユニットを登録し、毎日4:00に実行されるようスケジュールの登録をします。
- ・ シナリオ2：ファイル転送結果の閲覧
スケジュールで実行されたファイル転送のジョブユニットについて、実行結果の確認をします。
- ・ シナリオ3：ファイル転送失敗時の再配信指示
ファイル転送のジョブユニットの実行に失敗した場合、再配信指示をします。

シナリオ1～3について、次章以降説明していきます。

5.2 シナリオ1：ファイル転送設定の登録

5.2.1 シナリオ概要

シナリオ1：ファイル転送設定の登録では、ファイル転送のジョブユニットを登録し、毎日4:00に実行されるようスケジュールの登録をします。

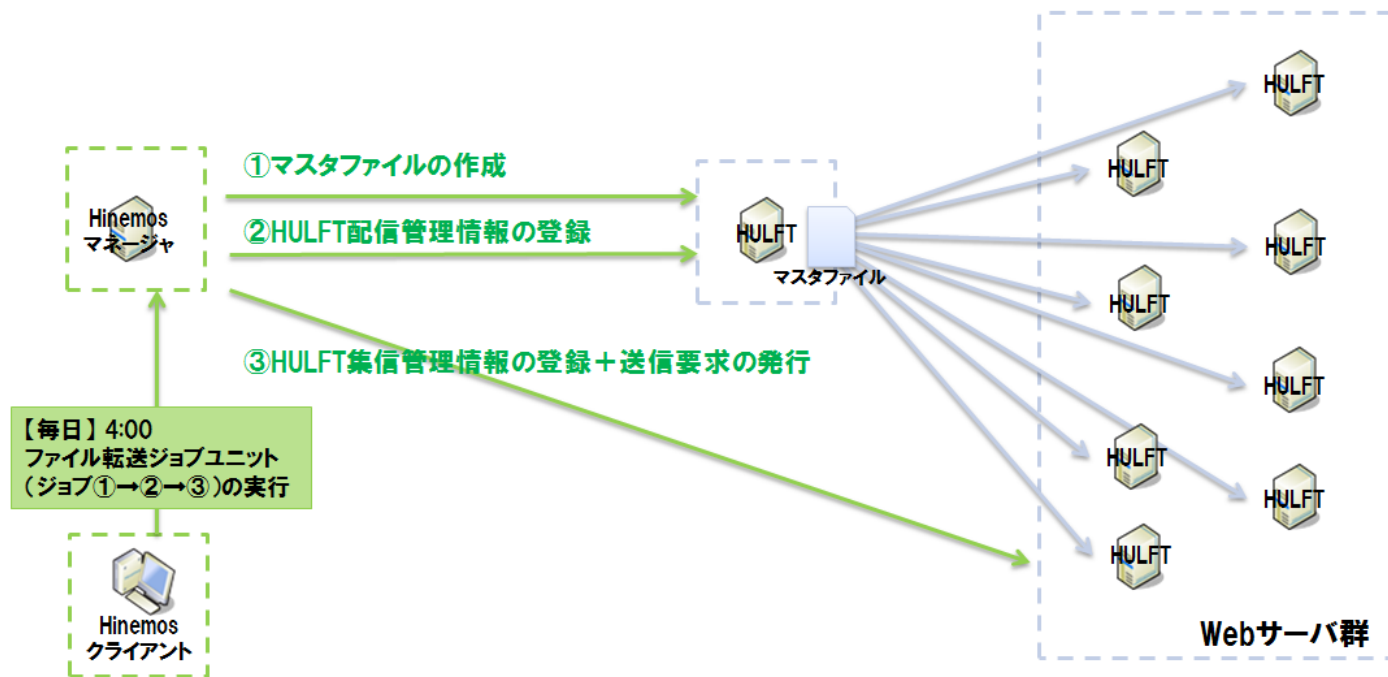


図6, シナリオ1：ファイル転送設定の登録

Hinemosジョブ管理機能による設定手順は、下記のとおりです。

- ・ 手順1：ファイル転送ジョブユニットの登録
- ・ 手順2：マスタファイル作成ジョブ(図中のジョブ①)の登録
- ・ 手順3：配信管理情報登録ジョブ(図中のジョブ②)の登録
- ・ 手順4：ファイル転送ジョブ(図中のジョブ③)の登録
- ・ 手順5：スケジュールの登録

手順1～4終了時のファイル転送ジョブユニットは下記となります。

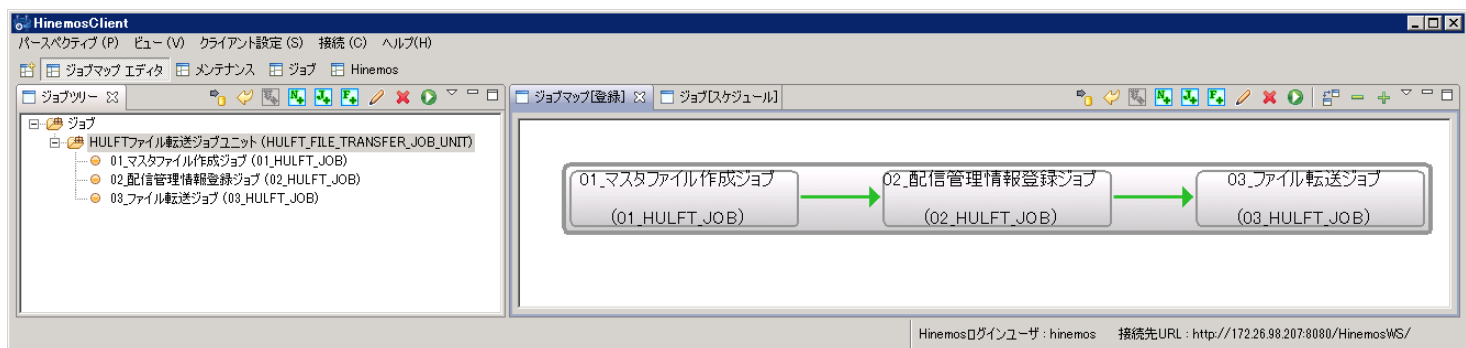


図7, ファイル転送ジョブユニット

各設定手順は、次節以降をご覧ください。

5.2.2 手順1：ファイル転送ジョブユニットの登録

ファイル転送ジョブユニットの作成・登録をします。

ジョブユニット作成の詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 9.4.1 ジョブユニットの作成・変更」

	終了値	終了値の範囲
正常:	0	0 - 0
警告:	1	1 - 1
異常:	-1	(正常・警告以外)

図8, ファイル転送ジョブユニットの例

5.2.3 手順2：マスタファイル作成ジョブの登録

ファイル転送ジョブユニット配下に、配信するマスタファイルを作成する、マスタファイル作成ジョブの作成・登録をします。

ジョブ作成時の注意点は下記のとおりです。

- ・ スコープは、マスタファイルを作成するノード(バッチサーバ)を指定します。
- ・ 起動コマンドは、マスタファイルを作成するコマンドまたはスクリプトを指定します。なお、マスタファイル作成スクリプトは、必要に応じて各プロジェクトにてご用意ください。ここでは、バッチサーバがWindowsの場合は「C:\hulft_testtest02.txt」、Linuxの場合は「/root/hulft_test/test01.txt」を作成したと仮定して、手順3以降を進めていきます。

ジョブ作成の詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 9.4.3 ジョブの作成・変更」

5.2.4 手順3：配信管理情報登録ジョブの登録

ファイル転送ジョブユニット配下に、配信管理情報登録ジョブの作成・登録をします。

ジョブ作成に際して、以下の処理を行うプログラムを用意します。

- ・ マスタファイルを作成したノード(バッチサーバ)に、HULFTの配信管理情報のパラメータファイルを作成する
- ・ マスタファイルを作成したノード(バッチサーバ)に、HULFTの配信管理情報を登録する

サンプルプログラムを使用したジョブ作成時の注意点は下記のとおりです。

- ・ 待ち条件は、マスタファイル作成ジョブが終了状態「正常」で終了した場合を指定します。
- ・ スコープは、マスタファイルを作成したノード(バッチサーバ)を指定します。
- ・ 起動コマンドは、下記のとおり指定します。
 - ・ バッチサーバがWindowsの場合
[HulftSend.batのパス] [登録するファイルID] [マスタファイルのパス] [転送グループID]
(例)「C:hulft_testjobsHulftSend.bat FILE02 C:hulft_testtest02.txt HINEMOS2」
 - ・ バッチサーバがLinuxの場合
[HulftSendの実行コマンド] [登録するファイルID] [マスタファイルのパス] [転送グループID]
(例)「/usr/bin/java -cp /root/hulft_test/jobs/ HulftSend FILE01 /root/hulft_test/test01.txt HINEMOS1」

ジョブ作成の詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 9.4.3 ジョブの作成・変更」

ジョブID: 02_HULFT_JOB

ジョブ名: 02_配信管理情報登録ジョブ

説明:

待ち条件 | 制御 | **コマンド** | 開始遅延 | 終了遅延 | 終了状態 | 通知先の指定

スコープ

☐ ジョブ変数: #[FACILITY_ID]

☒ 固定値: Webサーバ(Linux) 参照

スコープ処理

☒ 全てのノードで実行

☐ 正常終了するまでノードを順次リトライ

起動コマンド: /usr/bin/java -cp /root/hulft_test/jobs/ HulftSend FILE01

停止コマンド: /hulft_test/jobs/HulftSend_stop.sh

実効ユーザ: root

☒ コマンド実行失敗時に終了する

終了値: -1

OK(O) キャンセル(O)

図9, 配信管理情報登録ジョブの例

5.2.5 手順4：ファイル転送ジョブの登録

ファイル転送ジョブユニット配下に、ファイル転送ジョブの作成・登録をします。

ジョブ作成に際して、以下の処理を行うプログラムを用意します。

- ファイルの送信先に、HULFTの集信管理情報のパラメータファイルを作成する
- ファイルの送信先に、HULFTの集信管理情報を登録する
- ファイルの送信先で、バッチサーバに対し、送信要求コマンドの実行をする
- ファイル転送に失敗した場合、ファイルの送信先でエラーコードの取得・表示をする

サンプルプログラムを使用したジョブ作成時の注意点は下記のとおりです。

- ・待ち条件は、配信管理情報登録ジョブが終了状態「正常」で終了した場合を指定します。
- ・スコープは、ファイルの送信先となるスコープを指定します。
- ・起動コマンドは、下記のとおり指定します。
 - ・ファイルの送信先がWindowsの場合
[HulftFileTransferJob.batのパス] [登録したファイルID] [配置先のファイル名] [送信元のホスト名]
(例) 「C:hulft_testjobsHulftFileTransferJob.bat FILE01 C:hulft_test\$SNDFILE master_win_server」
 - ・ファイルの送信先がLinuxの場合
[HulftFileTransferJobの実行コマンド] [登録したファイルID] [配置先のファイル名] [送信元のホスト名]
(例) 「/usr/bin/java -cp /root/hulft_test/jobs/ HulftFileTransferJob FILE02 /root/hulft_test/'\$SNDFILE' master_linux_server」

ジョブ作成の詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 9.4.3 ジョブの作成・変更」

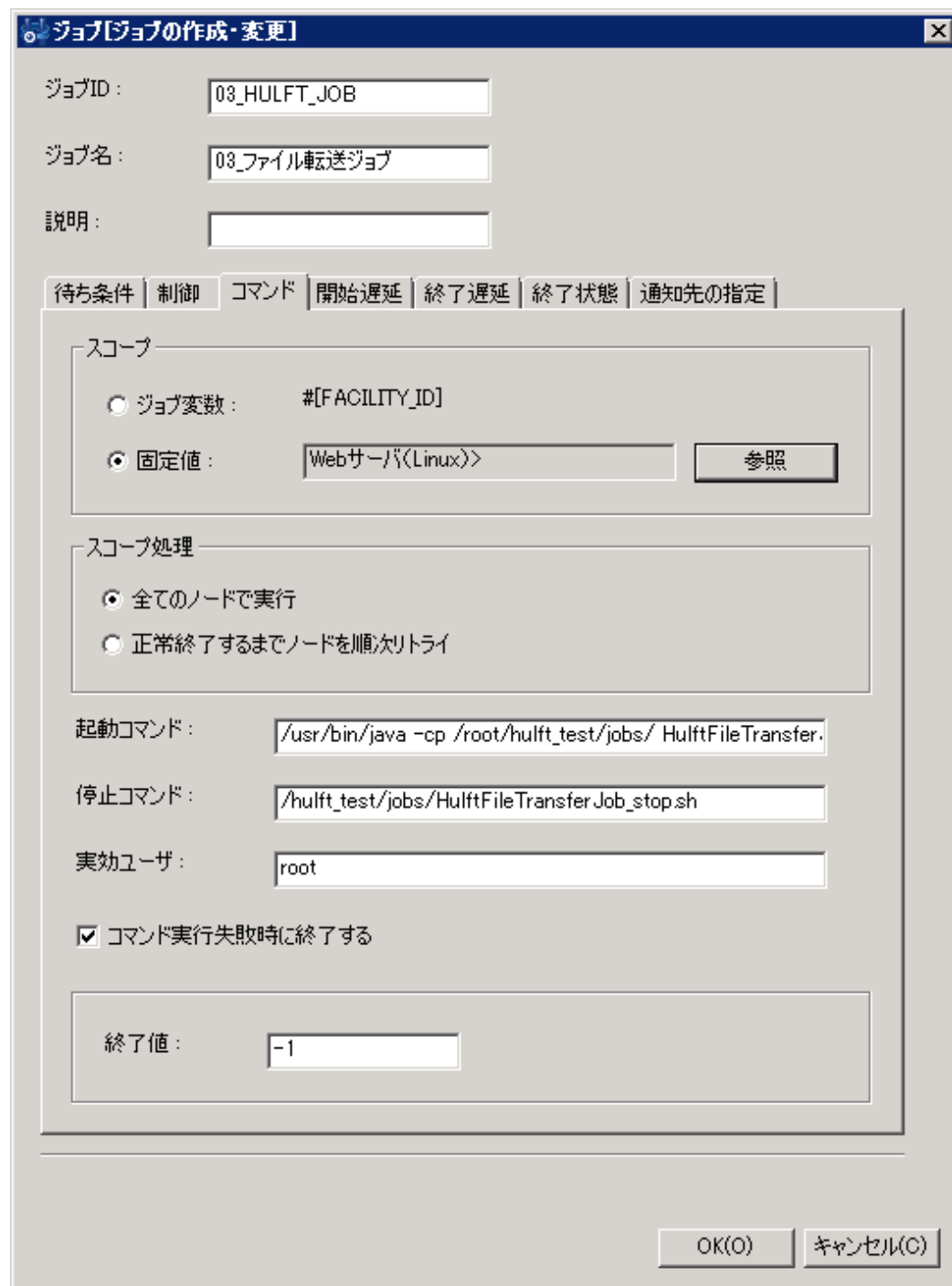


図10, ファイル転送ジョブの例

5.2.6 手順5：スケジュールの登録

ファイル転送ジョブユニットに対して、スケジュールの作成・登録をします。スケジュールの設定例を下記に記載します。

- ・ジョブIDは、ファイル転送ジョブユニットを指定します。
- ・スケジュールは、「月」「日」は空欄のまま、「04 時 00 分」を指定します。

スケジュール作成の詳細については、下記マニュアルを参照してください。

「Hinemos ver4.0 ユーザマニュアル 第1.1版 9.6.3 ジョブのスケジュール実行」

ジョブ[スケジュールの作成・変更]

スケジュールID: hulft_schedule_01

スケジュール名: HULFT毎日4時実行スケジュール

ジョブID: HULFT_FILE_TRANSFER_JOE 参照

ジョブ名: HULFTファイル転送ジョブユニット

カレンダーID:

スケジュール

☒ 月 日 04 時 00 分

☐ 曜日 時 分

有効/無効

☒ 有効 ☐ 無効

登録 キャンセル(C)

図11, スケジュールの例

5.3 シナリオ 2：ファイル転送結果の閲覧

5.3.1 シナリオ概要

シナリオ 2：ファイル転送結果の閲覧では、スケジュールで実行されたファイル転送のジョブユニット(図中のジョブ①～③)の実行結果を確認します。

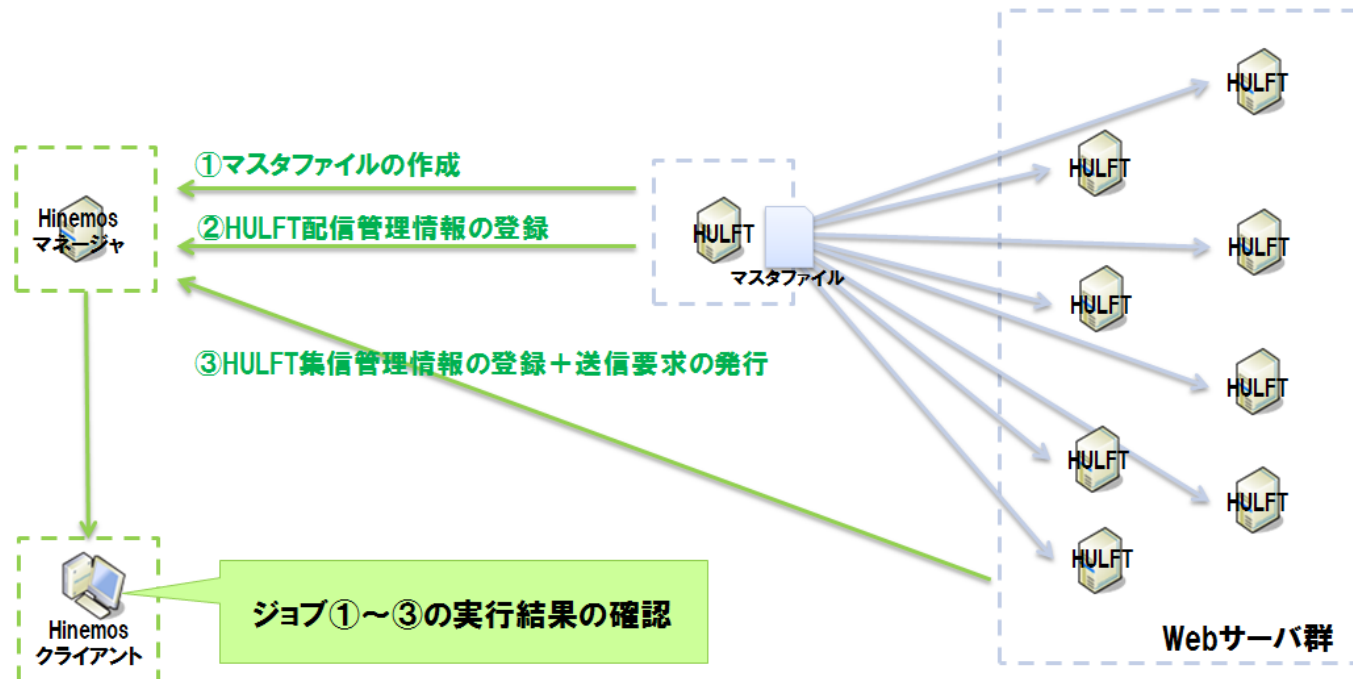


図12, シナリオ 2：ファイル転送結果の閲覧

ファイル転送に成功した場合と失敗した場合について、実行結果の確認方法を見ていきます。

5.3.2 ファイル転送に成功した場合

Hinemosクライアントのジョブマップビューアパースペクティブより、ジョブ[履歴]ビュー、ジョブマップ[履歴]ビュー、ジョブ[ノード詳細ビュー]を確認します。

ファイル転送に成功した場合、ファイル転送のジョブユニットは終了状態「正常」で終了します。（本連携では、ジョブユニット・ジョブの終了状態にデフォルト値を使用しています。）

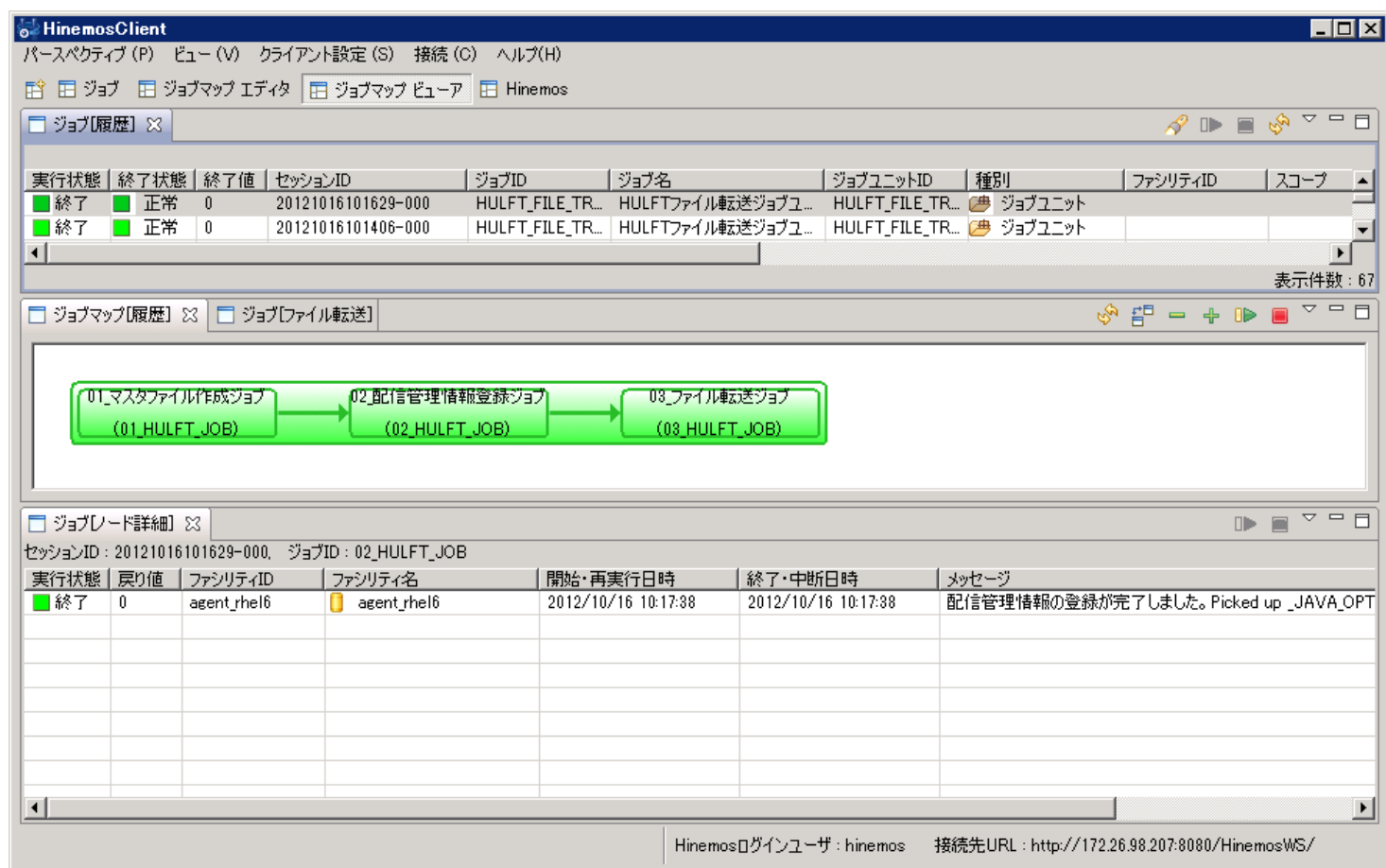


図13, ファイル転送に成功した場合

サンプルプログラムを使用した場合、配信管理情報登録ジョブ、ファイル転送ジョブのジョブ[ノード詳細ビュー]の戻り値は「0」となります。メッセージは下記のとおりです。

- ・ 配信管理情報登録ジョブのメッセージ
 - ・ マスタファイル作成ノード(バッチサーバ)がWindowsの場合
配信管理情報の登録が完了しました。
 - ・ マスタファイル作成ノード(バッチサーバ)がLinuxの場合
配信管理情報の登録が完了しました。
Picked up _JAVA_OPTIONS: -XX:OnOutOfMemoryError='/usr/bin/jmap -F -histo %p >> /opt/hinemos_agent/var/log/agent_outofmemory_histgram.log'
- ・ ファイル転送ジョブのメッセージ
 - ・ マスタファイル作成ノード(バッチサーバ)がWindowsの場合
集信管理情報の登録が完了しました。
ファイルの転送が完了しました。
 - ・ マスタファイル作成ノード(バッチサーバ)がLinuxの場合
集信管理情報の登録が完了しました。
ファイルの転送が完了しました。
Picked up _JAVA_OPTIONS: -XX:OnOutOfMemoryError='/usr/bin/jmap -F -histo %p >> /opt/hinemos_agent/var/log/agent_outofmemory_histgram.log'

なお、マスタファイル作成ジョブの戻り値、メッセージは、ファイル作成のスクリプト内でプロジェクトの要件に合わせて設定する必要があります。本連携では、戻り値「0」の場合、ジョブの終了状態が「正常」となるよう設定しています。

5.3.3 ファイル転送に失敗した場合

Hinemosクライアントのジョブマップビューアパースペクティブより、ジョブ[履歴]ビュー、ジョブマップ[履歴]ビュー、ジョブ[ノード詳細ビュー]を確認します。

サンプルプログラムを使用した場合、ファイル転送に失敗した場合は、ファイル転送のジョブユニットは終了状態「警告」または「異常」で終了します。（本連携では、ジョブユニット・ジョブの終了状態にデフォルト値を使用しています。）

エラーメッセージは、ジョブ[ノード詳細ビュー]のメッセージから確認します。

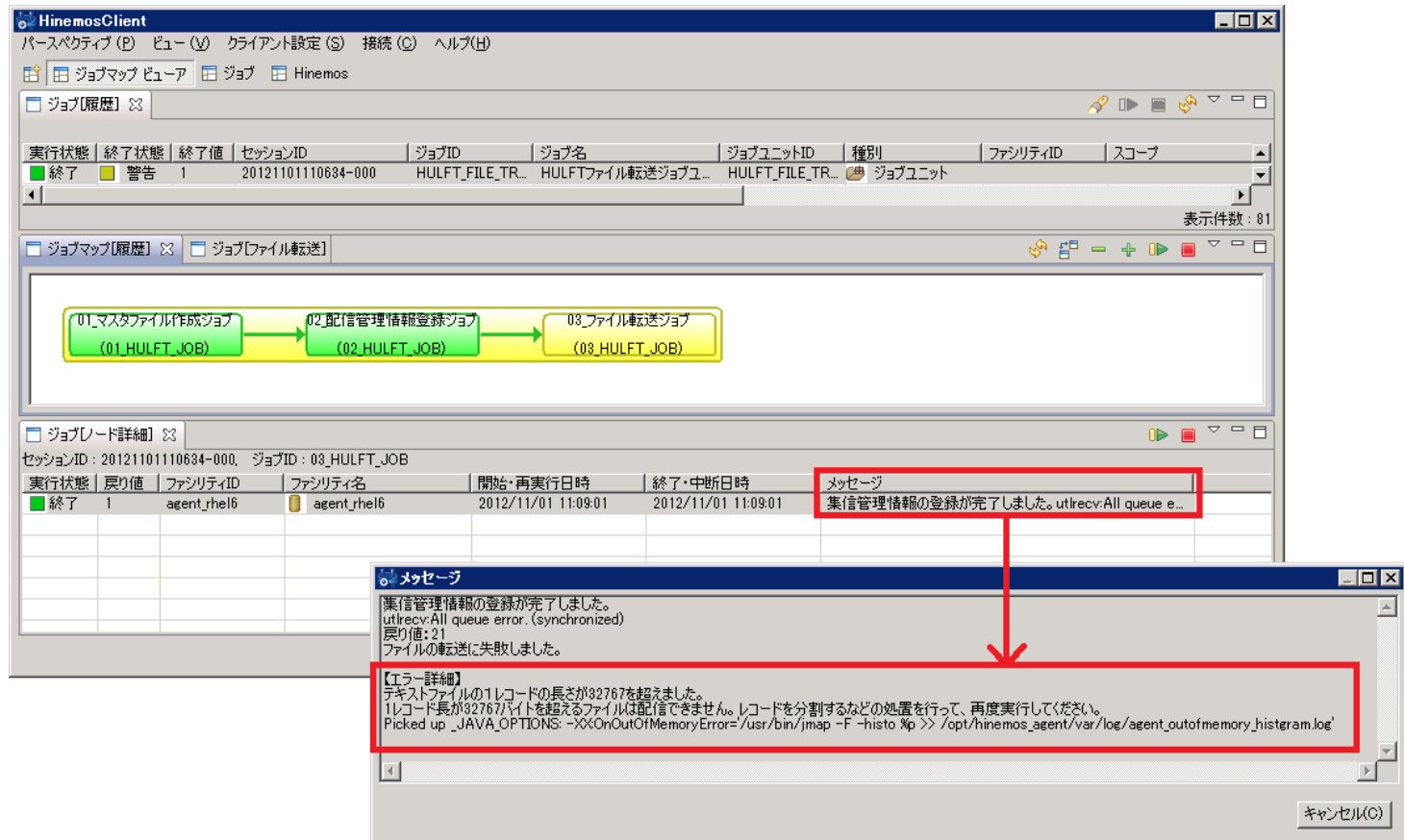


図14, ファイル転送に失敗した場合(ファイル転送ジョブの実行に失敗した例)

配信管理情報登録ジョブの実行に失敗した場合の実行結果は以下のとおりです。

- ・ジョブマップ[履歴]ビューのジョブの実行状態は「異常」となります。
- ・ジョブ[ノード詳細ビュー]の戻り値には、リターンコード「2」が入ります。
- ・ジョブ[ノード詳細ビュー]のメッセージで、エラーメッセージを確認します。

サンプルプログラムを使用した場合、ファイル転送ジョブの実行に失敗するパターンとして、集信管理情報の登録の失敗した場合、送信要求の実行に失敗した場合の2パターンが考えられます。実行結果は以下のとおりです。

【集信管理情報の登録の失敗した場合】

- ・ジョブマップ[履歴]ビューのジョブの実行状態は「異常」となります。
- ・ジョブ[ノード詳細ビュー]の戻り値には、リターンコード「2」が入ります。
- ・ジョブ[ノード詳細ビュー]のメッセージで、エラーメッセージを確認します。

【送信要求の実行に失敗した場合】

- ジョブマップ[履歴]ビューのジョブの実行状態は「警告」となります。
- ジョブ[ノード詳細ビュー]の戻り値には、リターンコード「1」が入ります。
- ジョブ[ノード詳細ビュー]のメッセージで、エラーメッセージを確認します。以下に例を挙げます。
 - ファイルの配信先スコープがWindowsの場合の例
集信管理情報の登録が完了しました。
utlrecv: 要求先ホストでエラーが発生しました。ホスト名=hulft_rhel6232 要求先エラーコード=21
utlrecv: 送信要求処理に失敗しました。
戻り値: 17
ファイルの転送に失敗しました。
【エラー詳細】
1レコードのデータ長が最大値を超えています。
テキスト転送の場合、1レコード（改行まで）の最大値は3 2 7 6 8 バイトです。配信ファイルを確認してください。
 - ファイルの配信先スコープがLinuxの場合の例
集信管理情報の登録が完了しました。
utlrecv: All queue error. (synchronized)
戻り値: 21
ファイルの転送に失敗しました。
【エラー詳細】
テキストファイルの1レコードの長さが32767を超えました。
1レコード長が32767バイトを超えるファイルは配信できません。レコードを分割するなどの処置を行って、再度実行してください。
Picked up _JAVA_OPTIONS: -XX:OnOutOfMemoryError='/usr/bin/jmap -F -histo %p >> /opt/hinemo
s_agent/var/log/agent_outofmemory_histgram.log'

なお、マスタファイル作成ジョブの戻り値、メッセージは、ファイル作成のスクリプト内でプロジェクトの要件に合わせて設定する必要があります。

5.4 シナリオ3：ファイル転送失敗時の再配信指示

5.4.1 シナリオ概要

シナリオ3：ファイル転送失敗時の再配信指示では、シナリオ2でファイル転送に失敗した場合、失敗した原因を取り除いた後、対象のジョブを手動で実行します。

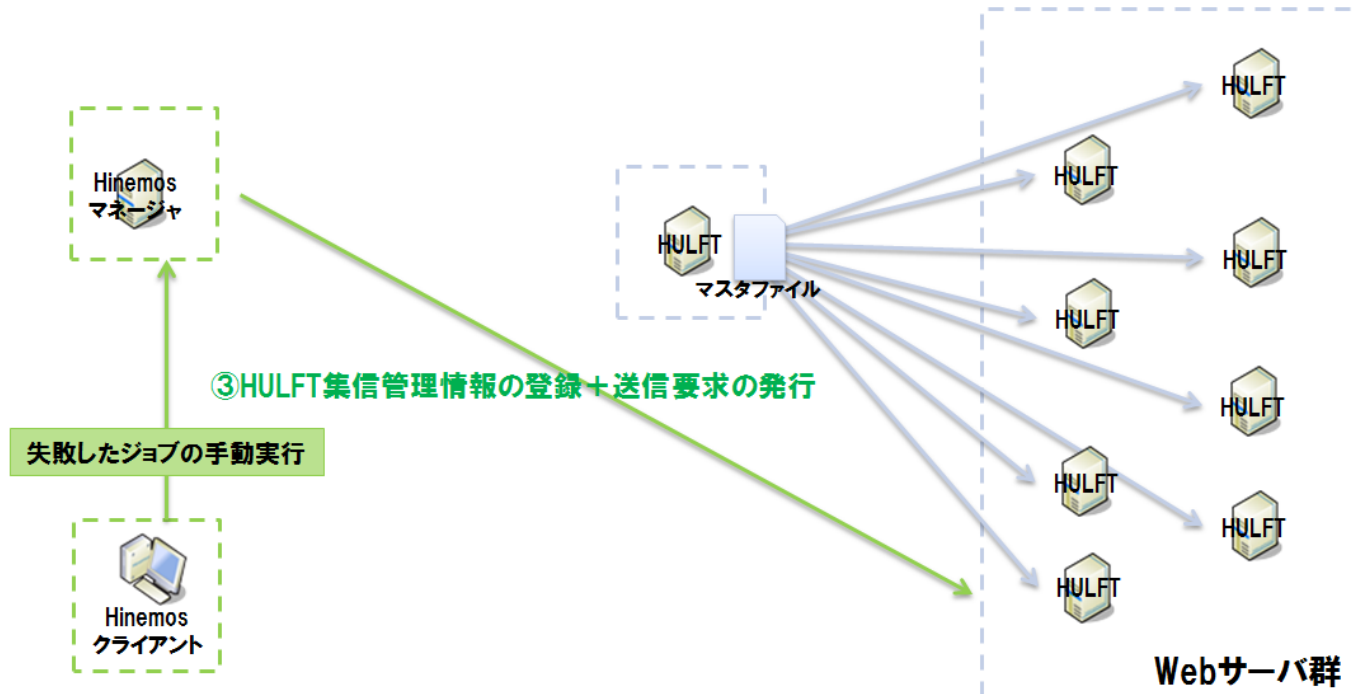


図15, シナリオ3：ファイル転送失敗時の再配信指示

5.4.2 再配信指示

ジョブの再配信指示は、失敗した原因を取り除いた後、ジョブマップ[履歴]ビューまたはジョブ[ノード詳細]ビューから手動で再実行をします。

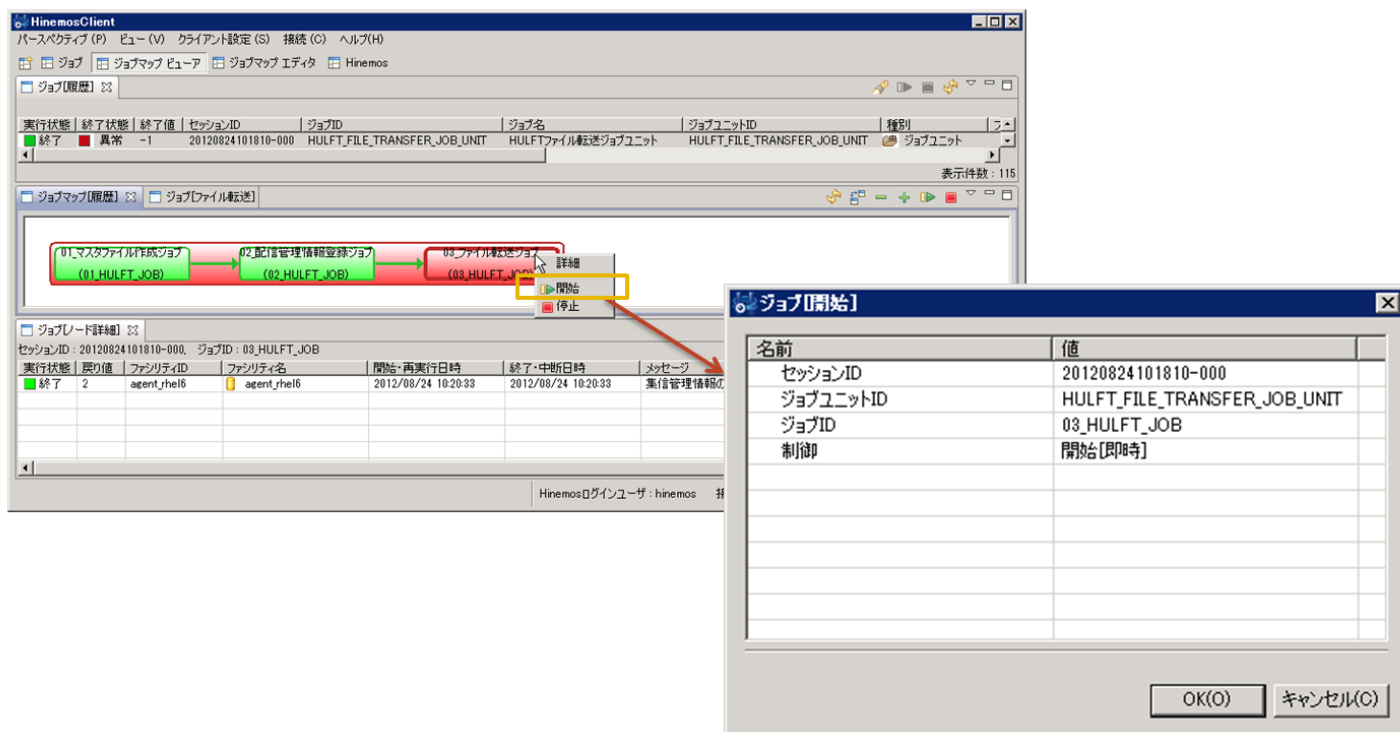


図16, 実行に失敗したジョブに再配信指示を行う場合の例

実行に失敗したジョブに再配信指示を行う場合は、ジョブマップ[履歴]ビューで対象のジョブを右クリックし、開始[即時]を行います。

また、実行に失敗したジョブのうち、特定のノードのみに再配信指示を行う場合は、ジョブ[ノード詳細]ビューで対象のジョブを右クリックし、開始[即時]を行います。

6 付録A

6.1 サンプルプログラムの構成

本連携では、下記2つのサンプルプログラムを用意しています。

- HulftSend.java

配信管理情報を登録するサンプルプログラムです。シナリオ1：ファイル転送設定の登録で、手順3：配信管理情報登録ジョブの登録に使用しています。

- HulftFileTransferJob.java

集信管理情報の登録、送信要求の実行をするサンプルプログラムです。シナリオ1：ファイル転送設定の登録で、手順4：ファイル転送ジョブの登録に使用しています。

なお、プログラムの実行環境がWindowsの場合、ジョブ作成時の起動コマンドに"Program Files"などの空白を含むファイルパスを指定できないため、バッチファイルを起動コマンドで指定し、バッチファイルからサンプルプログラムを呼び出す必要があります。

下記にHulftSend.javaを呼び出すHulftSend.batと、HulftFileTransferJob.javaを呼び出すHulftFileTransferJob.batを記載します。

HulftSend.bat

```
@echo off  
  
"C:\Program Files (x86)\Hinemos\Agent4.0.0\jre6\bin\java.exe" -cp C:\hulft_test\jobs HulftSend %1 %2 %3
```

HulftFileTransferJob.bat

```
@echo off  
  
"C:\Program Files (x86)\Hinemos\Agent4.0.0\jre6\bin\java.exe" -cp C:\hulft_test\jobs HulftFileTransferJob %1 %2 %3
```


6.2 サンプルプログラムで使用するHULFTのエラーコード・ファイル

6.2.1 HULFTのエラーコード

本連携のサンプルプログラム中で、エラーメッセージ取得時に使用するHULFTの履歴情報のエラーコードは、完了コードと詳細コードで構成されています。

- Windowsの場合

完了コード6桁(下3桁のみ参照)と、詳細コード5桁で構成されます。

000000(00000) (完了コード(詳細コード))

エラーコードの詳細は、下記マニュアルをご参照ください。

「HULFT7 Windows エラーコード・メッセージ 1.1 履歴情報エラーコード概要」

- Linuxの場合

完了コード4桁と、詳細コード4桁で構成されます。

0000-0000 (完了コード-詳細コード)

エラーコードの詳細は、下記マニュアルをご参照ください。

「HULFT7 UNIX/Linux エラーコード・メッセージ 1.1 履歴情報エラーコード概要」

6.2.2 HULFTのファイル

本連携のサンプルプログラム中で、エラーメッセージ取得時に使用するHULFTのファイルは下記のとおりです。

- コマンド実行ログ(Windows/Linux共通：huloplcmd.csv)

コマンド実行ログは、HULFTの操作コマンド実行履歴が記載されるログファイルです。送信要求コマンドを実行後、コマンド実行に紐付く処理識別子を取得するために使用します。

ファイルの詳細は、下記マニュアルをご参照ください。

- Windowsの場合

「HULFT7 Windows アドミニストレーション・マニュアル 付1.1 操作ログファイルフォーマット」

- Linuxの場合

「HULFT7 UNIX/Linux アドミニストレーション・マニュアル 付1.1 操作ログファイルフォーマット」

- 集信履歴ファイル(Windowsの場合：hulrcvlg.dat / Linuxの場合：hulrcvlog.dat)

集信履歴ファイルは、ファイル集信時のファイルID、時刻、エラーコード(完了コード、詳細エラーコード)などの履歴情報が記載されるファイルです。コマンド実行ログから取得した処理識別子と一致する行を探し、エラーコード取得するために使用します。

ファイルの詳細は、下記マニュアルをご参照ください。

- Windowsの場合

「HULFT7 Windows オペレーション・マニュアル 付1.1.2 集信履歴ファイル(hulrcvlg.dat) のフォーマット」

- Linuxの場合

「HULFT7 UNIX/Linux オペレーション・マニュアル 付1.1.2 集信履歴ファイル(hulrcvlg.dat) のフォーマット」

- エラー定義ファイル(Windowsの場合：NT.dat / Linuxの場合：UX.dat)

エラー定義ファイルは、HULFTのエラーコードに対するエラーメッセージを格納したファイルです。集信履歴ファイルから取得したエラーコードと一致する行を探し、エラーメッセージを取得するために使用します。エラーコードは、NT.datの場合は完了コードのみ、UX.datの場合は完了コードと詳細コードの両方を使用します。

ファイルの詳細は、本ドキュメントの「HULFTの必須設定項目」をご参照ください。

6.3 HulftSend.java

6.3.1 プログラム概要

配信管理情報を登録するサンプルプログラムです。

```
package com.sample;

import java.io.*;

public class HulftSend {
    /**
     * 配信管理情報の登録
     */
    public static final String[] pre_env = { "HULEXEP=/usr/local/HULFT/bin", "PATH=/usr/local/HULFT/bin:$PATH",
                                              "HULPATH=/usr/local/HULFT/etc" }; //HULFTの環境変数

    public static void main(String[] args) {
        String os = osCheck(); //実行環境のOSをチェック
        makeFile(os, args[0], args[1], args[2]); //配信管理情報のパラメータファイルを作成
        addSendInfo(os); //配信管理情報の登録
    }

    //実行環境のOSをチェック
    public static String osCheck() {
        String osname = System.getProperty("os.name");
        return osname;
    }

    //配信管理情報のパラメータファイルを作成
    public static void makeFile(String os, String fileid, String filename, String groupid) {
        FileWriter fw = null;
        try {
            String newline = ""; //改行
            if (os.indexOf("Windows") != -1) { //Windowsの場合
                newline = "\r\n";
            } else { //Linuxの場合
                newline = "\n";
            }
            String [] snd = {"SNDFILE="+fileid, "FILENAME="+filename, "INTERVAL=0", "BLOCKLEN=4096", "BLOCKCNT=3",
                            "COMP=NO", "COMPSIZE=0", "TRANSPRTY=50", "TRANSTYPE=TEXT", "CODESET=A", "KJCHNGE=S",
                            "SHIFTTRANSACT=Y", "CLEAR=K", "GRPID="+groupid, "END"}; //配信管理情報のパラメータ
            File fl = new File("./hulftsnd.txt"); //パラメータファイル
            fl.createNewFile(); //ファイルが存在しない場合、ファイルを生成
            fw = new FileWriter(fl);
            for (int i = 0; i < snd.length; i++) { //ファイルにパラメータを書き込む
                fw.write(snd[i] + newline); //各項目のパラメータ+改行
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                fw.close();
            } catch (IOException e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
        }
    }
}
```

```
// 配信管理情報の登録
public static void addSendInfo(String os) {
    try {
        String command = ""; // 実行コマンド
        if (os.indexOf("Windows") != -1) {
            command = "\"C:\\HULFT Family\\hulft7\\binnt\\utliupdt.exe\" -f ./hulftsnd.txt -r"; // 配信管理情報登録コマンド
            Runtime rt = Runtime.getRuntime();
            Process process = rt.exec(command); // コマンドの実行
            checkError(process); // コマンドの実行結果をチェック
        } else {
            String[] env = pre_env; // HULFTの環境変数
            command = "/usr/local/HULFT/bin/utliupdt -f ./hulftsnd.txt -r"; // 配信管理情報登録コマンド
            Runtime rt = Runtime.getRuntime();
            Process process = rt.exec(command, env); // コマンドの実行
            checkError(process); // コマンドの実行結果をチェック
        }
        System.out.println("配信管理情報の登録が完了しました。");
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

// 標準エラー出力と戻り値を表示
public static void checkError(Process process) {
    InputStream stream = null;
    BufferedReader br = null;
    try {
        int ret = process.waitFor(); // コマンドの戻り値
        stream = process.getInputStream();
        br = new BufferedReader(new InputStreamReader(stream));
        if (ret != 0) { // コマンドの戻り値が0でない場合
            String line;
            while ((line = br.readLine()) != null) { // 標準エラーがある限り読み込む
                System.out.println(line); // 標準エラーの出力
            }
            System.out.println("戻り値: " + ret); // 戻り値の出力
            System.out.println("配信管理情報の登録に失敗しました。");
            System.exit(2); // リターンコード「2」で終了
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    } catch (InterruptedException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    } finally {
        try {
            stream.close();
            br.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
}
```

プログラムの処理については次節以降をご覧ください。

6.3.2 配信管理情報のパラメータファイル作成

配信管理情報の登録に必要なパラメータファイルを作成します。配信管理情報登録ジョブの起動コマンドで引数に指定した値に応じて、「SNDFILE」にはファイルID、「FILENAME」には転送するファイル名、「GRPID」には転送グループIDが入力されます。

6.3.3 配信管理情報の登録

作成した配信管理情報のパラメータファイルから、配信管理情報を登録します。このとき、utliupdtコマンドを使用しますが、コマンドの戻り値が0以外の場合、標準エラーと戻り値を出力し、リターンコード「2」で終了します。

6.4 HulftFileTransferJob.java

6.4.1 プログラム概要

集信管理情報の登録、送信要求の実行をするサンプルプログラムです。

```
package com.sample;

import java.io.*;
import java.util.*;

public class HulftFileTransferJob {
    /**
     * 集信管理情報の登録、ファイル転送の実行
     */
    public static final String[] pre_env = { "HULEXEP=/usr/local/HULFT/bin", "PATH=/usr/local/HULFT/bin:$PATH",
        "HULPATH=/usr/local/HULFT/etc" }; //HULFTの環境変数

    public static final String pre_win_huloplcmd_path = "C:\\HULFT Family\\hulft7\\etc\\opl\\huloplcmd.csv"; //Windowsのコマンド実行ログのパス
    public static final String pre_win_hulrcvlog_path = "C:\\HULFT Family\\hulft7\\etc\\hulrcvlg.dat"; //Windowsの集信履歴ファイルのパス
    public static final String pre_linux_huloplcmd_path = "/usr/local/HULFT/etc/opl/huloplcmd.csv"; //Linuxのコマンド実行ログのパス
    public static final String pre_linux_hulrcvlog_path = "/usr/local/HULFT/etc/hulrcvlog.dat"; //Linuxの集信履歴ファイルのパス
    public static final String pre_win_NT = "C:\\HULFT Family\\hulft7\\etc\\errfile\\NT.dat"; //WindowsのNT.datのパス
    public static final String pre_win_UX = "C:\\HULFT Family\\hulft7\\etc\\errfile\\UX.dat"; //WindowsのUX.datのパス
    public static final String pre_linux_NT = "/root/hulft_test/jobs/NT.dat"; //LinuxのNT.datのパス
    public static final String pre_linux_UX = "/root/hulft_test/jobs/UX.dat"; //LinuxのUX.datのパス

    public static void main(String[] args) {
        String os = osCheck(); // 実行環境のOSをチェック
        makeFile(os, args[0], args[1]); // 集信管理情報のパラメータファイルを作成
        addReceiveInfo(os); // 集信管理情報の登録
        fileTransferJob(os, args[0], args[2]); // 送信要求コマンドの実行
        String error = getErrorCode(os); // エラーコードを取得
        getMessage(os, error); // 詳細メッセージを表示
    }

    //実行環境のOSをチェック
    public static String osCheck() {
        String osname = System.getProperty("os.name");
        return osname;
    }

    //集信管理情報のパラメータファイルを作成
    public static void makeFile(String os, String fileid, String filename) {
        FileWriter fw = null;
        try {
            String newline = ""; //改行
            if (os.indexOf("Windows") != -1) { //Windowsの場合
                newline = "\r\n";
            } else { //Linuxの場合
                newline = "\n";
            }
            String[] rcv = { "RCVFILE=" + fileid, "FILENAME=" + filename,
                "OWNER=root", "GROUP=root", "PERM=777", "CODESET=A",
                "TRANSMODE=REP", "ABNORMAL=KEEP", "RCVTYPE=S", "GENCTL=NO",
                "JOBWAIT=T", "DATAVERIFY=0", "END" }; //集信管理情報のパラメータ
            File fl = new File("./hulft_rcv.txt"); //パラメータファイル
            fl.createNewFile(); //ファイルが存在しない場合、ファイルを生成
            fw = new FileWriter(fl);
            for (int i = 0; i < rcv.length; i++) { //ファイルにパラメータを書き込む
                fw.write(rcv[i] + newline); //各項目のパラメータ+改行
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }

        e.printStackTrace();
    } finally {
        try {
            fw.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
//集信管理情報の登録
public static void addReceiveInfo(String os) {
    try {
        String command = ""; //実行コマンド
        int ret = 0; //コマンドの戻り値
        if (os.indexOf("Windows") != -1) {
            command = "\"C:\\HULFT Family\\hulft7\\binnt\\utliupdt.exe\" -f ./hulftrecv.txt -r"; //集信管理情報登録コマンド
            Runtime rt = Runtime.getRuntime();
            Process process = rt.exec(command); //コマンドの実行
            ret = checkError(process); //コマンドの実行結果をチェック
        } else {
            String[] env = pre_env; //HULFTの環境変数
            command = "/usr/local/HULFT/bin/utliupdt -f ./hulftrecv.txt -r"; //集信管理情報登録コマンド
            Runtime rt = Runtime.getRuntime();
            Process process = rt.exec(command, env); //コマンドの実行
            ret = checkError(process); //コマンドの実行結果をチェック
        }
        if (ret != 0) {
            System.out.println("集信管理情報の登録に失敗しました。");
            System.exit(2); //リターンコード「2」で終了
        } else {
            System.out.println("集信管理情報の登録が完了しました。");
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

//送信要求コマンドの実行
public static void fileTransferJob(String os, String fileid, String hostname) {
    try {
        String command = ""; //実行コマンド
        int ret = 0; //コマンドの戻り値
        if (os.indexOf("Windows") != -1) { //送信先がWindowsの場合
            command = "\"C:\\HULFT Family\\hulft7\\binnt\\utlrecv.exe\" -f "
                + fileid + " -h " + hostname + " -sync"; //送信要求コマンド
            Runtime rt = Runtime.getRuntime();
            Process process = rt.exec(command); //コマンドの実行
            ret = checkError(process); //コマンドの実行結果をチェック
        } else { //送信先がLinuxの場合
            String[] env = pre_env; //HULFTの環境変数
            command = "/usr/local/HULFT/bin/utlrecv -f " + fileid + " -h "
                + hostname + " -sync"; //送信要求コマンド
            Runtime rt = Runtime.getRuntime();
            Process process = rt.exec(command, env); //コマンドの実行
            ret = checkError(process); //コマンドの実行結果をチェック
        }
        if (ret != 0) {
            System.out.println("ファイルの転送に失敗しました。");
        } else {
            System.out.println("ファイルの転送が完了しました。");
            System.exit(0); //リターンコード「0」で終了
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

//エラーコードを取得
public static String getErrorCode(String os) {
    String error = ""; //エラーコード
```

```
BufferedReader br = null;
FileInputStream fis = null;
BufferedInputStream bis = null;
```

```

try {
    String huloplcmd_path = ""; // コマンド実行ログのパス
    String hulrcvlog_path = ""; // 集信履歴ファイルのパス
    String line = ""; // 該当ファイルの1行分
    String lastline = ""; // 該当行
    String id = ""; // 処理識別子
    byte[] buf; // 1行を1バイトずつ格納

    if (os.indexOf("Windows") != -1) {
        huloplcmd_path = pre_win_huloplcmd_path;
        hulrcvlog_path = pre_win_hulrcvlog_path;
        buf = new byte[1280]; // 集信履歴ファイルの1行あたりのバイト数
    } else {
        huloplcmd_path = pre_linux_huloplcmd_path;
        hulrcvlog_path = pre_linux_hulrcvlog_path;
        buf = new byte[1088]; // 集信履歴ファイルの1行あたりのバイト数
    }
    File csv = new File(huloplcmd_path);
    br = new BufferedReader(new FileReader(csv)); // コマンド実行ログを読み込む
    while ((line = br.readLine()) != null) { // 1行ずつ最終行まで読み込む
        lastline = line;
    }
    StringTokenizer st = new StringTokenizer(lastline, "\\,\\"); // 最終行を「,」で区切る
    int i = 0;
    while (st.hasMoreTokens()) {
        id = st.nextToken();
        if (i == 7) { // 8番目のトークンである処理識別子を取得
            break;
        }
        i++;
    }
    fis = new FileInputStream(hulrcvlog_path);
    bis = new BufferedInputStream(fis); // 集信履歴ファイルを読み込む
    int len = bis.read(buf);
    String matchline;
    String [] errorcode = new String [8];
    while ( ( len = bis.read(buf) ) != -1 ) { // 1行ずつ読み込み、処理識別子と一致するか確認
        matchline = new String(buf, 0, len); // 読み込んだ行を文字列に変換
        if (matchline.indexOf(id) != -1) { // 処理識別子と一致したら終了
            break;
        }
    }
    if (os.indexOf("Windows") != -1) { // Windowsの場合
        for (i = 161; i <= 166; i++) { // 16進数で161~164バイト目を取得
            errorcode[i-161] = String.format("%1$x ", buf[i-1]);
            errorcode[i-161] = errorcode[i-161].length() == 2 ? "0" + errorcode[i-161] : errorcode[i-161];
        }
        errorcode[6] = errorcode[3].trim() + errorcode[2].trim() + errorcode[1].trim() + errorcode[0].trim();
        errorcode[7] = errorcode[5].trim() + errorcode[4].trim(); // 詳細コード
        int errorcode1 = Integer.parseInt(errorcode[6], 16); // 10進数に変換
        int errorcode2 = Integer.parseInt(errorcode[7], 16); // 10進数に変換
        errorcode1 = errorcode1%1000; // 完了コード
        errorcode[6] = String.format("%03d", errorcode1); // 3桁に整形
        errorcode[7] = String.format("%05d", errorcode2); // 5桁に整形
        error = errorcode[6] + errorcode[7]; // 完了コード + 詳細コード
    } else {
        for (i = 161; i <= 164; i++) { // 16進数で161~164バイト目を取得
            errorcode[i-161] = String.format("%1$x ", buf[i-1]);
            errorcode[i-161] = errorcode[i-161].length() == 2 ? "0" + errorcode[i-161] : errorcode[i-161];
        }
        errorcode[4] = errorcode[1].trim() + errorcode[0].trim(); // 完了コード
        errorcode[5] = errorcode[3].trim() + errorcode[2].trim(); // 詳細コード
        int errorcode1 = Integer.parseInt(errorcode[4], 16); // 10進数に変換
        int errorcode2 = Integer.parseInt(errorcode[5], 16); // 10進数に変換
    }
}

```



```
        errorcode[4] = String.format("%04d", errorcode1);//4桁に整形
        errorcode[5] = String.format("%04d", errorcode2);//4桁に整形
        error = errorcode[4] + errorcode[5];//完了コード+詳細コード
    }
} catch (FileNotFoundException e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
} catch (IOException e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
} finally {
    try{
        br.close();
        fis.close();
        bis.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
return error;
}

//標準エラー出力と戻り値を表示
public static int checkError(Process process) {
    int ret = 0;//コマンドの戻り値
    InputStream stream = null;
    BufferedReader br = null;
    try {
        ret = process.waitFor();
        stream = process.getInputStream();
        br = new BufferedReader(new InputStreamReader(stream));
        if (ret != 0) { //コマンドの戻り値が0でない場合
            String line;
            while ((line = br.readLine()) != null) { //標準エラーがある限り読み込む
                System.out.println(line); //標準エラーの出力
            }
            System.out.println("戻り値 : " + ret); //戻り値の出力
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    } catch (InterruptedException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    } finally {
        try{
            stream.close();
            br.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
    return ret;
}

//詳細メッセージを表示
public static void getMessage(String os, String errorcode){
    String [] filepath = new String[2]; //エラーファイルのパス
    String fileerrorcode = ""; //エラーファイルと一致させるエラーコード
    if(os.indexOf("Windows")!=-1){ //送信先がWindowsの場合
        filepath[0] =pre_win_NT; //NT.datのパス
        filepath[1] =pre_win_UX; //UX.datのパス
    }
```

```

int winerrorcode = Integer.parseInt(errorcode);
if((winerrorcode/100000)==406){//完了コードが「xxx406(00xxx)」の場合
    fileerrorcode = Integer.toString(winerrorcode%1000);
    searchErrorcode(filepath[0], fileerrorcode);//送信元がWindowsの場合メッセージを出力
    fileerrorcode = "0" + Integer.toString(winerrorcode%1000);
    searchErrorcode(filepath[1], fileerrorcode);//送信元がLinuxの場合メッセージを出力
}else{//完了コードが「xxx406(00xxx)」以外の場合
    fileerrorcode = Integer.toString(winerrorcode/100000);
    searchErrorcode(filepath[0], fileerrorcode);
}
} else {//送信先がLinuxの場合
    filepath[0] =pre_linux_UX;//UX.datのパス
    filepath[1] =pre_linux_NT;//NT.datのパス
    int linuxerrorcode = Integer.parseInt(errorcode);
    if((linuxerrorcode/10000)==209){//完了コードが「02090xxx」の場合
        fileerrorcode = Integer.toString(linuxerrorcode%1000);
        searchErrorcode(filepath[1], fileerrorcode);//送信元がWindowsの場合メッセージを出力
        fileerrorcode = "0" + Integer.toString(linuxerrorcode%1000);
        searchErrorcode(filepath[0], fileerrorcode);//送信元がLinuxの場合メッセージを出力
    }else{//完了コードが「02090xxx」以外の場合
        fileerrorcode = errorcode;
        searchErrorcode(filepath[0], fileerrorcode);
    }
}
}
System.exit(1);//リターンコード「1」で終了
}

//エラーファイルを探索・出力
public static void searchErrorcode(String filepath, String errorcode) {
    BufferedReader br = null;
    try {
        String line = "";//該当ファイルの1行分
        String matchline = "";//該当行
        String message = "";//エラー詳細
        int errorcount = 0;//エラー表示件数
        File fp = new File(filepath);
        br = new BufferedReader(new FileReader(fp));
        while ((line = br.readLine()) != null) { // 最終行まで読み込む
            matchline = line;
            StringTokenizer st = new StringTokenizer(matchline, ",");//読み込んだ行を「,」で区切る
            int i = 0;
            while (st.hasMoreTokens()) {
                message = st.nextToken();
                if (i == 1){
                    if(message.startsWith(errorcode)) { // 2番目のトークンがエラーコードと前方一致した場合
                        errorcount++;
                        System.out.println();
                        System.out.println("【エラー詳細】 ");
                    }else{// 2番目のトークンがエラーコードと一致しなかった場合終了、次の行に進む
                        break;
                    }
                }
                }else if(i>1){//エラーメッセージを表示
                    System.out.println(message);
                }
                i++;
            }
        }
        if(errorcount >= 2){//エラー表示件数が2件以上の場合
            System.out.println();
            System.out.println("上記エラー詳細のいずれかが、原因として考えられます。");
        }
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

```

```
    } catch (IOException e) {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    } finally {  
        try{  
            br.close();  
        } catch (IOException e) {  
            System.out.println(e.getMessage());  
            e.printStackTrace();  
        }  
    }  
}  
}
```

プログラムの処理については次節以降をご覧ください。

6.4.2 集信管理情報のパラメータファイル作成

集信管理情報の登録に必要なパラメータファイルを作成します。ファイル転送ジョブの起動コマンドで引数に指定した値に応じて、「RCVFILE」には登録したファイルID、「FILENAME」には配置先のファイル名が入力されます。

6.4.3 集信管理情報の登録

作成した集信管理情報のパラメータファイルから、集信管理情報を登録します。このとき、utliupdtコマンドを使用しますが、コマンドの戻り値が0以外の場合、標準エラーと戻り値を出力し、リターンコード「2」で終了します。

6.4.4 送信要求コマンドの実行

ファイル転送ジョブの起動コマンドで引数に指定したファイルID、送信元のホスト名を引数として送信要求を実行します。送信先がLinuxの場合は、コマンド実行時にHULFTの環境変数を渡します。コマンドの戻り値が0の場合、リターンコード「0」で終了します。

6.4.5 エラーコードを取得

送信要求コマンドの実行に失敗した場合、エラーコード(完了コード+詳細コード)を取得します。コマンド実行ログ(huloplcmd.csv)の最終行から処理識別子を取得します。次に、集信履歴ファイル(hulrcvlog.dat)で取得した処理識別子に一致する行を探索し、完了コードと詳細コードを取得します。最終的には、ファイルの転送先がWindowsの場合は「000(完了コード)+00000(詳細コード)」、Linuxの場合は「0000(完了コード)+0000(詳細コード)」の形で、いずれも8桁の文字列として整形します。

6.4.6 詳細メッセージの表示

取得したエラーコード(完了コード+詳細コード)に応じて、エラー定義ファイル(NT.dat、UX.dat)より、エラーメッセージを取得し、詳細メッセージとして出力します。詳細メッセージ出力後は、リターンコード「1」で終了します。エラーコードの構成については、本ドキュメントの「サンプルプログラムで使用するHULFTのエラーコード・ファイル」をご参照ください。

エラーコードによる振り分け方法は、下記の図をご覧ください。

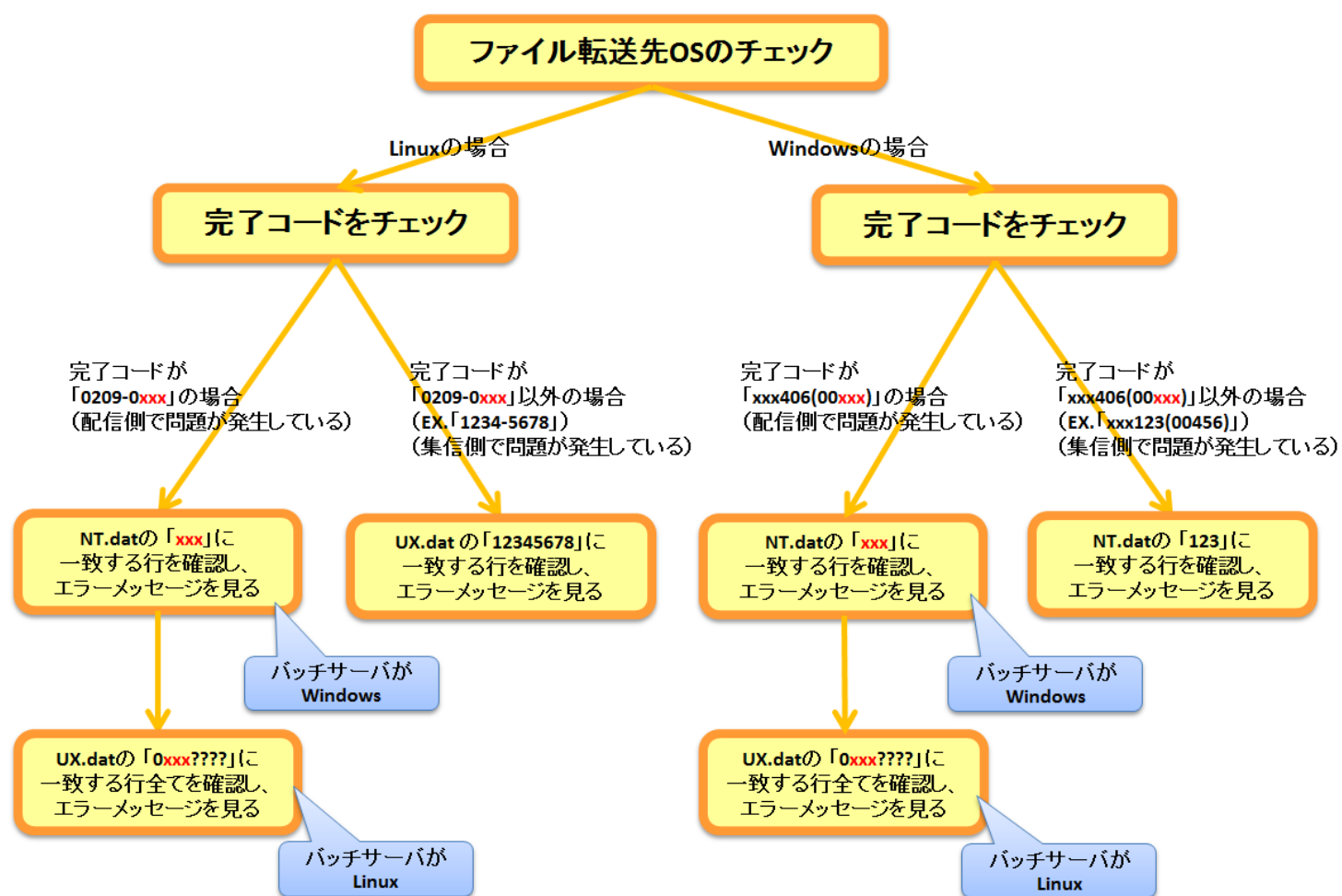


図17, エラーコードによる振り分け方法

7 付録B

7.1 連携にあたっての検討内容

本連携では、HULFTのファイル転送時に送信元から送信先に対して「配信要求」を実行するのではなく、送信先から送信元に対して「送信要求」を実行しています。

配信要求を選択した場合、ジョブは送信元に対して実行することになります。

この場合、ジョブ作成時のスコープには送信元のマスタサーバを指定するため、送信先のWebサーバを指定するには、配信要求時に指定したファイルIDの配信管理情報と紐付いた転送グループIDを利用する必要があります。また、ジョブの実行結果も送信元のサーバに集約するため、送信先それぞれの結果を確認することが困難となります。

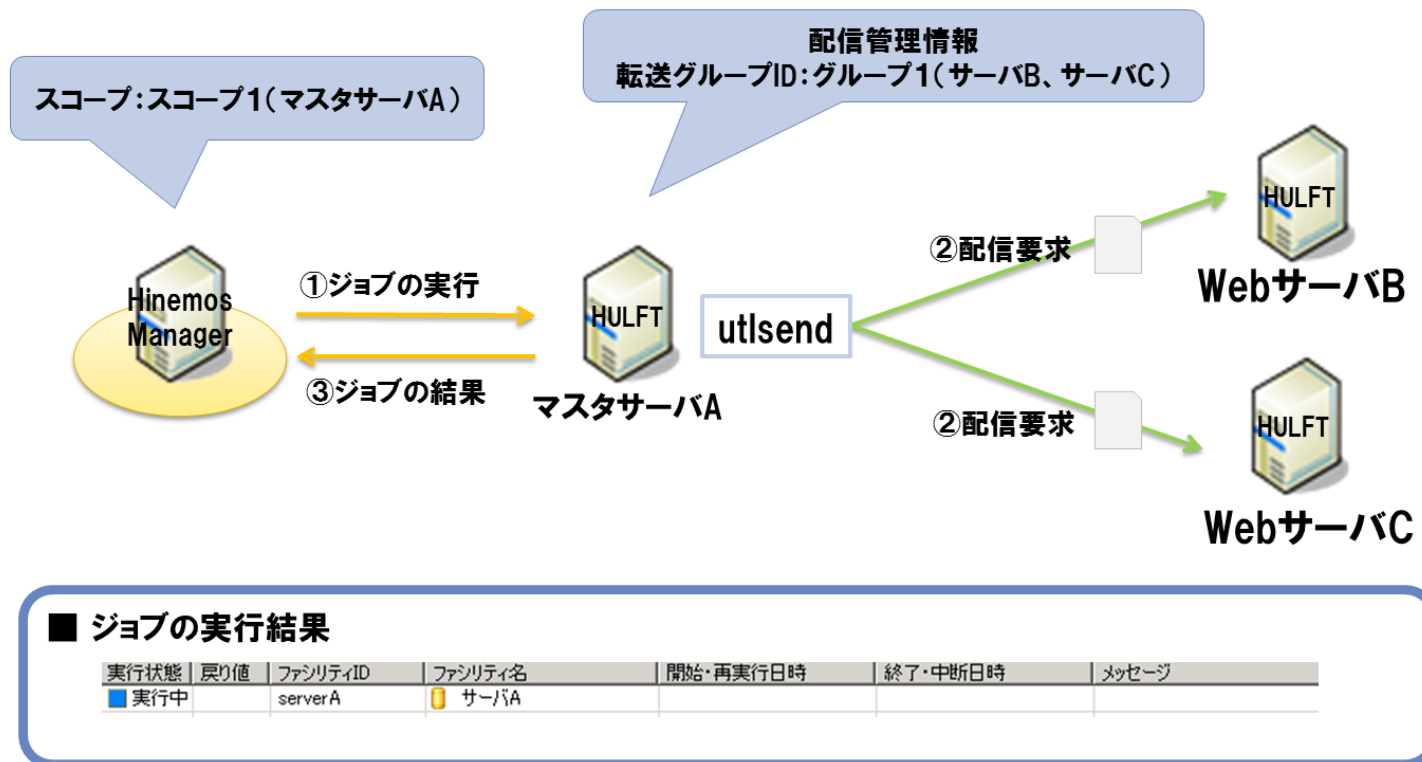


図18, 配信要求を選択した場合

一方、送信要求を選択した場合、ジョブは送信先に対して実行することになります。

この場合、ジョブ作成時のスコープには、Hinemosのスコープを利用して送信先のWebサーバ群を指定できます。送信要求コマンド「utlrecv」の引数に送信元ホスト名を記載することで、送信元のマスタサーバも指定可能です。また、ジョブの実行結果も送信先それぞれの結果を確認することが可能となります。

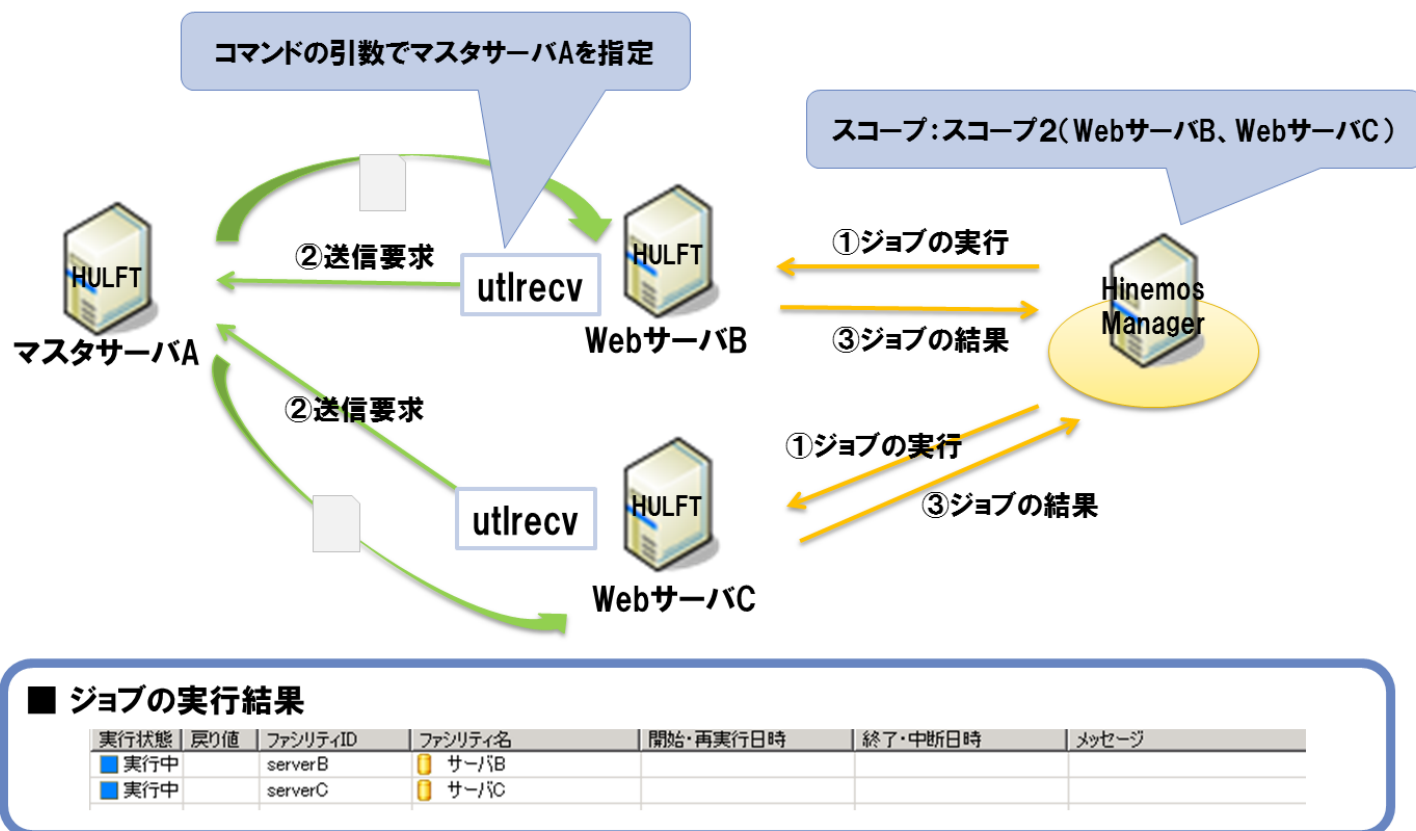


図19, 送信要求を選択した場合

配信要求と送信要求を検討した結果、本連携では1サーバから複数のサーバにファイルを配布するシナリオに適応できる送信要求を選択しました。

Hinemos HULFT 連携ノウハウと適用例 公開日：2012/11/5

非売品

- 禁無断複製
- 禁無断転載
- 禁無断再配布

Hinemosは（株）NTTデータの登録商標です。

Linuxは、Linus Torvalds氏の米国およびその他の国における登録商標または商標です。

その他、本書に記載されている会社名、製品名は、各社の登録商標または商標です。

なお、本文中にはTM、Rマークは表記しておりません。