Oracle® Endeca Information Discovery Integrator

Integratorサーバー・ガイド

リリース3.1.0　2013年10月

ORACLE

著作権および免責事項

OracleおよびJavaはOracle Corporationおよびその関連企業の登録商標です。その他の名称は、それぞれの所有者の商標または登録商標です。UNIXは、The Open Groupの登録商標です。

このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複写、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アセンブル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。

ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクル社までご連絡ください。

このソフトウェアまたは関連ドキュメントを、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供する場合は、次の通知が適用されます。

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations.As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs.No other rights are granted to the U.S. Government.

このソフトウェアもしくはハードウェアは様々な情報管理アプリケーションでの一般的な使用のために開発されたものです。このソフトウェアもしくはハードウェアは、危険が伴うアプリケーション(人的傷害を発生させる可能性があるアプリケーションを含む)への用途を目的として開発されていません。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用する際、安全に使用するために、適切な安全装置、バックアップ、冗長性(redundancy)、その他の対策を講じることは使用者の責任となります。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用したことに起因して損害が発生しても、オラクル社およびその関連会社は一切の責任を負いかねます。

このソフトウェアまたはハードウェア、そしてドキュメントは、第三者のコンテンツ、製品、サービスへのアクセス、あるいはそれらに関する情報を提供することがあります。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスに関して一切の責任を負わず、いかなる保証もいたしません。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

Clover Serverに対するOracle Endecaの補足

この補足では、Clover ServerをOracle Endeca Integrator Serverとして使用する場合のサポートと制限事項に関する特定の情報を示します。

サポートされるコンテナ

Oracle Endeca Integrator Serverは、次のコンテナでのみサポートされます。

- Apache Tomcat
- Oracle WebLogic

Clover Serverは他のコンテナにインストール可能ですが、Oracle Endeca Integrator Serverでは、他のコンテナへのインストールはサポートされません。

CloverETL Server

リファレンス・マニュアル

www.cloveretl.com

www.javlininc.com

Varecha Martin [FAMILY Given]
Waller Tomas [FAMILY Given]

CloverETL Server: リファレンス・マニュアル

www.cloveretl.com
www.javlininc.com
: Varecha Martin [FAMILY Given] 、 Waller Tomas [FAMILY Given]

このリファレンス・マニュアルでは、CloverETL Serverのリリース3.4.xについて説明しています。

リリース3.4

フィードバックの宛先:

このドキュメントに関してご意見またはご提案がある場合は、docs@cloveretl.comに電子メールをお送りください。

Javlin

目次

図一覧

表一覧

第1章 CloverETL Serverの概要

CloverETL Serverは、CloverETLデータ統合スイートのエンタープライズ・ランタイム、監視および自動化システムです。大規模で複雑なプロジェクトのデータ統合プロセスをデプロイ、監視、スケジュール、統合および自動化するために必要なツールを提供します。

CloverETL ServerのHTTPおよびSOAP WebサービスAPIは、CloverETL Serverを既存のアプリケーション・ポートフォリオおよびプロセスに統合するための追加の自動化制御を提供します。

CloverETL Serverは、J2EE標準に作成されたJavaアプリケーションです。Apache Tomcat、Jetty、IBM Websphere、Sun Glassfish、JBoss、Oracle Weblogicなど幅広いアプリケーション・サーバーをサポートしています。

表1.1 CloverETL ServerとCloverETL Engineの比較

	CloverETL Server	実行可能ツールとしてのCloverEngine
グラフ実行の機能	http（またはJMX、その他）のAPIをコール(詳細は、14章「単純なHTTP API」を参照してください。)	外部プロセスを実行、またはJava APIをコール
エンジンの初期化	サーバーの起動時	グラフを実行するたびにinitがコールされます
スレッドとメモリーの最適化	スレッドのリサイクル、グラフ・キャッシュ、その他	未実装
スケジューリング	タイムテーブル、ワンタイム・トリガー、ロギングなどによりスケジューリング	外部ツール(Cronなど)を使用可
統計	各グラフを実行するたびに独自のログ・ファイルが作成されて結果ステータスが格納され、CSによりトリガーされた各イベントが記録されます	未実装
監視	グラフが失敗した場合、イベント・リスナーに通知されます。電子メールを送信し、シェル・コマンドを実行するか、他のグラフを実行する場合があります。7章「グラフ・イベント・リスナー」を参照してください。また、サーバーはサーバー/グラフ・ステータスの監視に使用できる各種のAPI (HTTPおよびJMX)を実装します。	グラフの実行中にJMX mBeanを使用できます
グラフと関連ファイルの格納	グラフはサーバー・ファイルシステム上で、いわゆるサンドボックスに格納されます	
セキュリティと認可のサポート	CSはユーザー/グループの管理をサポートするため、各サンドボックスには独自のアクセス権セットを指定できます。すべてのインタフェースは認証を必要とします。3章「サーバー側ジョブ・ファイル - サンドボックス」を参照してください。	ユーザーが入力したパスワードを暗号化できます
統合機能	CSには、HTTPなどの共通プロトコルを使用してコールできるAPIがあり	CloverEngineライブラリを、クライアントのJavaコードの組込みライブ

	CloverETL Server	実行可能ツールとしてのCloverEngine
	ます。14章「単純なHTTP API」を参照してください。	ラリとして使用することも、グラフごとに個別のOSプロセスとして実行することもできます。
グラフの開発	CSは1つのプロジェクト(サンドボックス)に対するチーム作業をサポートします。今後のバージョンでは、CloverETL DesignerがCSに統合されます。	
スケーラビリティ	CSは変換リクエストの水平方向のスケーラビリティと、データ・スケーラビリティを実装します。24章「クラスタリング」を参照してください。また、CloverEngineの実装にはネイティブで垂直方向のスケーラビリティがあります。	Clover Engineが垂直方向のスケーラビリティを実装します
ジョブフロー	CSは各種のジョブフロー・コンポーネントを実装します。詳細は、CloverETLのマニュアルを参照してください。	Clover Engine自体が限定的にジョブフローをサポートします。

第2章 インストール

次の項では、2つの異なるインストール・タイプについて説明します。評価サーバーというタイトルの項では、構成なしの最も迅速で単純なインストールについて詳細に説明し、エンタープライズ・サーバーというタイトルの項では、選択したアプリケーション・コンテナおよびデータベースに対するテストと本番について詳細に説明します。

評価サーバー

CloverETL Serverのデフォルト・インストールでは、追加のデータベース・サーバーは必要ありません。組込みのApache Derby DBが使用されます。また、後続の構成も不要です。CloverETL Serverは、最初の起動時に自動的に構成されます。データベース表と必要なレコードの一部が、最初の起動時に空のデータベース上に自動的に作成されます。Web GUIのサンドボックスセクションで、サンドボックスとデモ用のいくつかのグラフが作成されたことを確認できます。

注記

CloverETL Serverコンソールのデフォルトのログイン資格証明:

ユーザー名: clover

パスワード: clover

なんらかの構成、たとえば電子メールの送信、LDAP認証、クラスタリングなどの変更を必要とするCloverETl Serverの機能を評価する必要がある場合、またはTomcat以外のアプリケーション・コンテナでCloverETL Serverを評価する必要がある場合は、「エンタープライズ・サーバー」で説明されている一般的なインストールに進んでください。

Apache Tomcatのインストール

CloverETL Serverを実行するには、Apache Tomcatバージョン6.0.xが必要です。

Apache Tomcatがすでにインストールされている場合は、この項をスキップできます。

1. http://tomcat.apache.org/download-60.cgiから、バイナリのディストリビューションを含むZIPをダウンロードします。Tomcatは、Windows OSのサービスとしてインストールすることもできますが、ファイルシステムへのアクセスに問題が生じる可能性があり、評価の目的用としてはお薦めしません。
2. ダウンロードしたzipファイルを解凍します。
3. **[tomcat_home]/bin/startup.sh**（またはWindows OSの場合は**[tomcat_home]/bin/startup.bat**）で、Tomcatを実行します。
4. URL: http://localhost:8080/で、Tomcatが実行中かどうかを確認します。Apache Tomcatの情報ページが開きます。
5. Apache Tomcatがインストールされます。

詳細なインストール手順が必要な場合は、http://tomcat.apache.org/tomcat-6.0-doc/setup.htmlを参照してください。

CloverETL Serverのインストール

1. 次の前提条件を満たしているかどうかを確認します。
   - Oracle JDKまたはJRE v. 1.6.x以上
   - **JAVA_HOME**または**JRE_HOME**環境変数が設定されていること。

- Apache Tomcat 6.0.xがインストールされていること。詳細は、「Apache Tomcatのインストール」を参照してください。

2. メモリー制限、その他のスイッチを設定します。詳細は、「メモリー設定」の項を参照してください。

   setenvファイルを作成します。

   UNIXライクなシステムの場合: **[tomcat]/bin/setenv.sh**

   ```
   export CATALINA_OPTS="$CATALINA_OPTS -XX:MaxPermSize=512m -Xms128m -Xmx2048m"
   export CATALINA_OPTS="$CATALINA_OPTS -Dderby.system.home=$CATALINA_HOME/temp -server"
   echo "Using CATALINA_OPTS: $CATALINA_OPTS"
   ```

   Windowsシステムの場合: **[tomcat]/bin/setenv.bat**

   ```
   set "CATALINA_OPTS=%CATALINA_OPTS% -XX:MaxPermSize=512m -Xms128m -Xmx1024m"
   set "CATALINA_OPTS=%CATALINA_OPTS% -Dderby.system.home=%CATALINA_HOME%/temp -server"
   echo "Using CATALINA_OPTS: %CATALINA_OPTS%"
   ```

3. Apache Tomcat用のCloverETL Serverを含むWebアーカイブ・ファイル(**clover.war**)と、有効なライセンス・ファイルを含む**clover-license.war**をダウンロードします。

4. 両方のWARファイル(**clover.war**および**clover-license.war**)を**[tomcat_home]/webapps**ディレクトリにデプロイします。

   デプロイメント上の問題を回避するために、コピー中はTomcatを停止してください。

5. **[tomcat_home]/bin/startup.sh**（またはWindows OSの場合は**[tomcat_home]/bin/startup.bat**)で、Tomcatを実行します。

6. 次のURLで、CloverETL Serverが実行中かどうかを確認します。

   - Webアプリケーション・ルート

     http://[host]:[port]/[contextPath]

     httpコネクタのデフォルトのTomcatポートは8080で、CloverETL Serverのデフォルトの**contextPath**は"clover"です。したがって、デフォルトのURLは次のようになります。

     http://localhost:8080/clover/

   - Web GUI

     http://[host]:[port]/[contextPath]/gui http://localhost:8080/clover/gui

     デフォルトの管理者資格証明を使用してWeb GUIにアクセスします。ユーザー名は"clover"、パスワードは"clover"です。

7. CloverETL Serverがインストールされ、基本的な評価の準備ができます。各種の変換デモがインストールされた2つのサンドボックスがあります。

エンタープライズ・サーバー

この項では、各種のアプリケーション・サーバーにおけるCloverETL Serverのインストールの詳細を説明し、またサーバーの構成方法についても説明します。簡単にCloverETL Serverの機能を評価する場合、構成が必要なく、評価インストールが適しています。「評価サーバー」を参照してください。

エンタープライズ環境用のCloverETL Serverは、Webアプリケーション・アーカイブ(WARファイル)として出荷されます。そのため、アプリケーション・サーバーにWebアプリケーションをデプロイする標準的な方法を使用できます。ただし、アプリケーション・サーバーごとに動作と機能はそれぞれ異なります。それぞれのインストールと構成の詳細は、次の各項に記載されています。

対応するコンテナのリスト:

- 「Apache Tomcat」
- 「Jetty」
- 「IBM Websphere」
- 「Glassfish / Sun Java System Application Server」
- 「JBoss」
- 「Oracle WebLogic Server」

インストール中に問題が発生した場合は、「インストール中に想定される問題」を参照してください。

Apache Tomcat

Apache Tomcatのインストール

CloverETL Serverを実行するには、Apache Tomcatバージョン6.0.xが必要です。

Apache Tomcatがすでにインストールされている場合は、この項をスキップできます。

1. http://tomcat.apache.org/download-60.cgiからバイナリ・ディストリビューション・フォームをダウンロードします。
2. ダウンロードしたzipファイルを解凍します。
3. **[tomcat_home]/bin/startup.sh**（またはWindows OSの場合は**[tomcat_home]/bin/startup.bat**）で、Tomcatを実行します。
4. URL: http://localhost:8080/で、Tomcatが実行中かどうかを確認します。Apache Tomcatの情報ページが開きます。
5. Apache Tomcatがインストールされます。

詳細なインストール手順が必要な場合は、http://tomcat.apache.org/tomcat-6.0-doc/setup.htmlを参照してください。

CloverETL Serverのインストール

1. Apache Tomcat用のCloverETL Serverを含むWebアーカイブ・ファイル(**clover.war**)をダウンロードします。
2. 次の前提条件を満たしているかどうかを確認します。
   - Oracle JDKまたはJRE v. 1.6.x以上
   - **JAVA_HOME**または**JRE_HOME**環境変数が設定されていること。
   - Apache Tomcat 6.0.xがインストールされていること。CloverETL Serverは、Apache Tomcat 6.0.xコンテナに対応して開発とテストが行われています(想定されていない他のバージョンでも動作する場合があります)。詳細は、「Apache Tomcatのインストール」を参照してください。

- **heapとperm genのメモリー領域に対するデフォルトの制限は変更することをお薦めします。**

  詳細は、「メモリー設定」の項を参照してください。

  "Xms"と"Xmx"のJVMパラメータを調整して、メモリー・ヒープの最小サイズと最大サイズを設定できます。**[TOMCAT_HOME]/bin/setenv.sh**ファイルの環境変数**JAVA_OPTS**を設定して、TomcatのJVMパラメータを設定できます(存在しない場合は、作成できます)。

  setenvファイルを作成します。

  UNIXライクなシステムの場合: **[tomcat]/bin/setenv.sh**

```
export CATALINA_OPTS="$CATALINA_OPTS -XX:MaxPermSize=512m -Xms128m -Xmx1024m"
export CATALINA_OPTS="$CATALINA_OPTS -Dderby.system.home=$CATALINA_HOME/temp -
server"
echo "Using CATALINA_OPTS: $CATALINA_OPTS"
```

  Windowsシステムの場合: **[tomcat]/bin/setenv.bat**

```
set "CATALINA_OPTS=%CATALINA_OPTS% -XX:MaxPermSize=512m -Xms128m -Xmx1024m"
set "CATALINA_OPTS=%CATALINA_OPTS% -Dderby.system.home=%CATALINA_HOME%/temp -server"
echo "Using CATALINA_OPTS: %CATALINA_OPTS%"
```

  前述の設定でもわかるように、**-server**というスイッチもあります。パフォーマンス上の理由により、コンテナはサーバー・モードで実行することをお薦めします。

3. (Tomcat用にビルドされた) **clover.war**を**[tomcat_home]/webapps**ディレクトリにコピーします。

   コピーはアトミック操作ではありません。Tomcatの実行中は、コピー・プロセスの長さに注意してください。コピー時間が長すぎると、デプロイ中に失敗の原因になることがあります。これは、Tomcatが不完全なファイルのデプロイを試行するためです。かわりに、Tomcatが実行中でないときにファイルを操作してください。

4. Tomcatを再起動しなくても、WARファイルは自動的に検出されてデプロイされます。

5. 次のURLで、CloverETL Serverが実行中かどうかを確認します。

   - Webアプリケーション・ルート

     http://[host]:[port]/[contextPath]

     httpコネクタのデフォルトのTomcatポートは8080で、CloverETL Serverのデフォルトの**contextPath**は"clover"です。したがって、デフォルトのURLは次のようになります。

     http://localhost:8080/clover/

   - Web GUI

     http://[host]:[port]/[contextPath]/gui

     httpコネクタのデフォルトのTomcatポートは8080で、CloverETL Serverのデフォルトの contextPathは"clover"です。したがって、デフォルトのURLは次のようになります。

     http://localhost:8080/clover/gui

     デフォルトの管理者資格証明を使用してWeb GUIにアクセスします。ユーザー名は"clover"、パスワードは"clover"です。

Apache TomcatでのCloverETL Serverの構成

デフォルトのインストール(構成なし)をお薦めするのは評価を目的とする場合のみです。本番では、少なくともDBデータ・ソースとSMTPサーバーの構成をお薦めします。

指定した場所のプロパティ・ファイル

このようなファイルの例をあげます。

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

プロパティ・ファイルは、システム・プロパティまたは環境変数**clover_config_file**(**clover.config.file**)で指定された場所からロードされます。

Apache Tomcatのシステム・プロパティは、**[TOMCAT_HOME]/bin/setenv.sh**ファイル(存在しない場合は作成できます)で設定できます。単に**JAVA_OPTS="$JAVA_OPTS -Dclover_config_file=/path/to/cloverServer.properties"**を追加します。

CloverETL Serverライセンスのインストール

グラフを実行するには、CloverETL Serverに有効なライセンスが必要です。ライセンスがなくてもCloverETL Serverはインストールできますが、グラフは実行できません。

Tomcatにライセンスをインストールするには、2つの方法があります。簡単なのは、別のWebアプリケーション**clover-license.war**を使用する方法です。ただしクラスタ環境の場合は、プレーン・ライセンス・ファイルの構成を実行する必要があります(すべてのアプリケーション・コンテナで共通)。

a) 個別のライセンスWAR

1. Webアーカイブ・ファイル**clover-license.war**をダウンロードします。

2. **clover-license.war**を**[tomcat_home]/webapps**ディレクトリにコピーします。

3. Tomcatを再起動しなくても、WARファイルは自動的に検出されてデプロイされます。

4. 次のURLで、ライセンスのWebアプリケーションが実行中かどうかを確認します。

   http://[host]:[port]/clover-license/ (注意: contextPath **clover-license**は必須であり、変更できません)

b) license.fileのプロパティ

または、サーバーの"license.file"プロパティを構成します。その値を、**license.dat**ファイルのフルパスに設定します。

注記

CloverETLのライセンスは、**clover-license.war**を再デプロイすることによって、いつでも変更できます。後から、ライセンスを変更したことをCloverETL Serverに認識させる必要があります。

- サーバーのWeb GUI → 構成 → CloverETL情報 → ライセンスに移動します。
- ライセンスの再ロードをクリックします。
- あるいは、CloverETL Serverアプリケーションを再起動する方法もあります。

警告: WARファイルを再デプロイする際には、ディレクトリ**[tomcat_home]/webapps/[contextPath]**を削除する必要があります。Tomcatが実行中の場合は自動的に行われます。その場合でも手動で確認するようにしてください。そうしないと、変更が反映されません。

IBM AS/400 (iSeries)のApache Tomcat

iSeriesプラットフォームでCloverETL Serverを実行する場合は、いくつか追加の設定があります。

1. 32ビット版のJava 6.0を使用していることを宣言します
2. パラメータ**-Djava.awt.headless=true**を指定してJavaを実行します

これを構成するには、次の内容が含まれているファイル**[tomcat_home]/bin/setenv.sh**を変更/作成します。

```
JAVA_HOME=/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit
```

```
JAVA_OPTS="$JAVA_OPTS -Djava.awt.headless=true"
```

Jetty

CloverETL Serverのインストール

1. Jetty用に作成されたCloverETL Serverアプリケーションを含むWebアーカイブ・ファイル(**clover.war**)をダウンロードします。
2. 次の前提条件を満たしているかどうかを確認します。
   - Oracle JDKまたはJREバージョン1.6.x以上
   - Jetty 6.1.x (サポートされるのはこの特定のバージョンのみ)

     jetty-6リリースはすべて、http://jetty.codehaus.org/jetty/から入手できます。Jetty 7はサポートされていません(Jetty 7は、Eclipse Foundationによって運営されているためディストリビューション・パッケージに大きい違いがあります)。
   - メモリー割当て設定

     これに関係するのは、JVMパラメータの-Xms -Xmx (ヒープ・メモリー)と、-XX:MaxPermSize (クラスローダ・メモリー制限)です。詳細は、「メモリー設定」の項を参照してください。

     パラメータを設定するには、次の行

     ```
     JAVA_OPTIONS='$JAVA_OPTIONS -Xms128m -Xmx1024m -XX:MaxPermSize=256m'
     ```

     を**[JETTY_HOME]/bin/jetty.sh**に追加します。

3. **clover.war**を**[JETTY_HOME]/webapps**にコピーします。

4. コンテキスト・ファイル**clover.xml**を**[JETTY_HOME]/contexts**に作成し、次の行を入力します。

```
<?xml version="1.0"  encoding="ISO-8859-1"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN"
"http://www.eclipse.org/jetty/configure.dtd">
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
<Set name="contextPath">/clover</Set>
<Set name="war"><SystemProperty name="jetty.home" default="."/>/webapps/clover.war
</Set>
</Configure>
```

   Jettyによって**clover.xml**が検出され、アプリケーションが自動的にロードされます。

5. **[JETTY_HOME]/bin/jetty.sh start**（またはWindows OSの場合は**[JETTY_HOME]/bin/Jetty-Service.exe**）を実行します。

最後に、たとえば、http://localhost:8080/test/でサーバーが稼働しているかどうかを確認します。

JettyでのCloverETL Serverの構成

デフォルトのインストール(構成なし)をお薦めするのは評価を目的とする場合のみです。本番では、少なくともDBデータ・ソースとSMTPサーバーの構成をお薦めします。

構成プロパティを設定するには、いくつかの方法がありますが、最も一般的なのは、指定した場所のプロパティ・ファイルです。

指定した場所のプロパティ・ファイル

このようなファイルの例をあげます。

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

共通プロパティ・ファイルは、環境/システム・プロパティ**clover_config_file**または**clover.config.file**によって指定された場所からロードされます。これが、Jettyを構成する推奨の方法です。

Jettyのシステム・プロパティは、**[JETTY_HOME]/bin/jetty.sh**ファイルで設定できます。次を追加します。

```
JAVA_OPTIONS="$JAVA_OPTIONS -Dclover_config_file=/path/to/cloverServer.properties"
```

CloverETL Serverライセンスのインストール

グラフを実行するには、CloverETL Serverに有効なライセンス・ファイルが必要です。ライセンスがなくてもCloverETL Serverはインストールできますが、グラフは実行できません。

1. **license.dat**ファイルを取得します。

2. CloverETL Serverの**license.file**パラメータを、**license.dat**のパスに設定します。

   - "license.file"プロパティを構成プロパティ・ファイルに追加します(「JettyでのCloverETL Serverの構成」の説明を参照)。その値を、**license.dat**ファイルのフルパスに設定します。

   - Jettyを再起動します。

   ライセンスを構成するには、いくつかの方法があります。すべての方法の詳細は、18章「構成」を参照してください。

注記

CloverETLのライセンスは、**license.dat**ファイルを置き換えることによって、いつでも変更できます。後から、ライセンスを変更したことをCloverETL Serverに認識させる必要があります。

- サーバーのWeb GUI → 構成 → CloverETL情報 → ライセンスに移動します。

- ライセンスの再ロードをクリックします。

- あるいは、CloverETL Serverアプリケーションを再起動する方法もあります。

IBM Websphere

CloverETL Serverのインストール

1. Websphere用に作成されたCloverETL Serverアプリケーションを含むWebアーカイブ・ファイル(**clover.war**)をダウンロードします。

2. 次の前提条件を満たしているかどうかを確認します。

   - IBM JDKまたはJREバージョン1.6.x以上

   - IBM Websphere 7.0 (http://www.ibm.com/developerworks/downloads/ws/was/を参照)

   - メモリー割当て設定

     これに関係するのは、JVMパラメータの-Xmsと-Xmxです。詳細は、「メモリー設定」の項を参照してください。

     ヒープ・サイズは、IBM WebsphereのIntegrated Solutions Consoleで設定できます(デフォルトでは**http://[host]:10003//ibm/console/**でアクセスできます)

     - サーバー → Application Server → [server1] (またはサーバーの別の名前) → プロセス管理 → Java仮想マシンに移動します。

     - 最大ヒープ・サイズというフィールドがあります。デフォルト値は256 MBですが、ETL変換には不十分です。

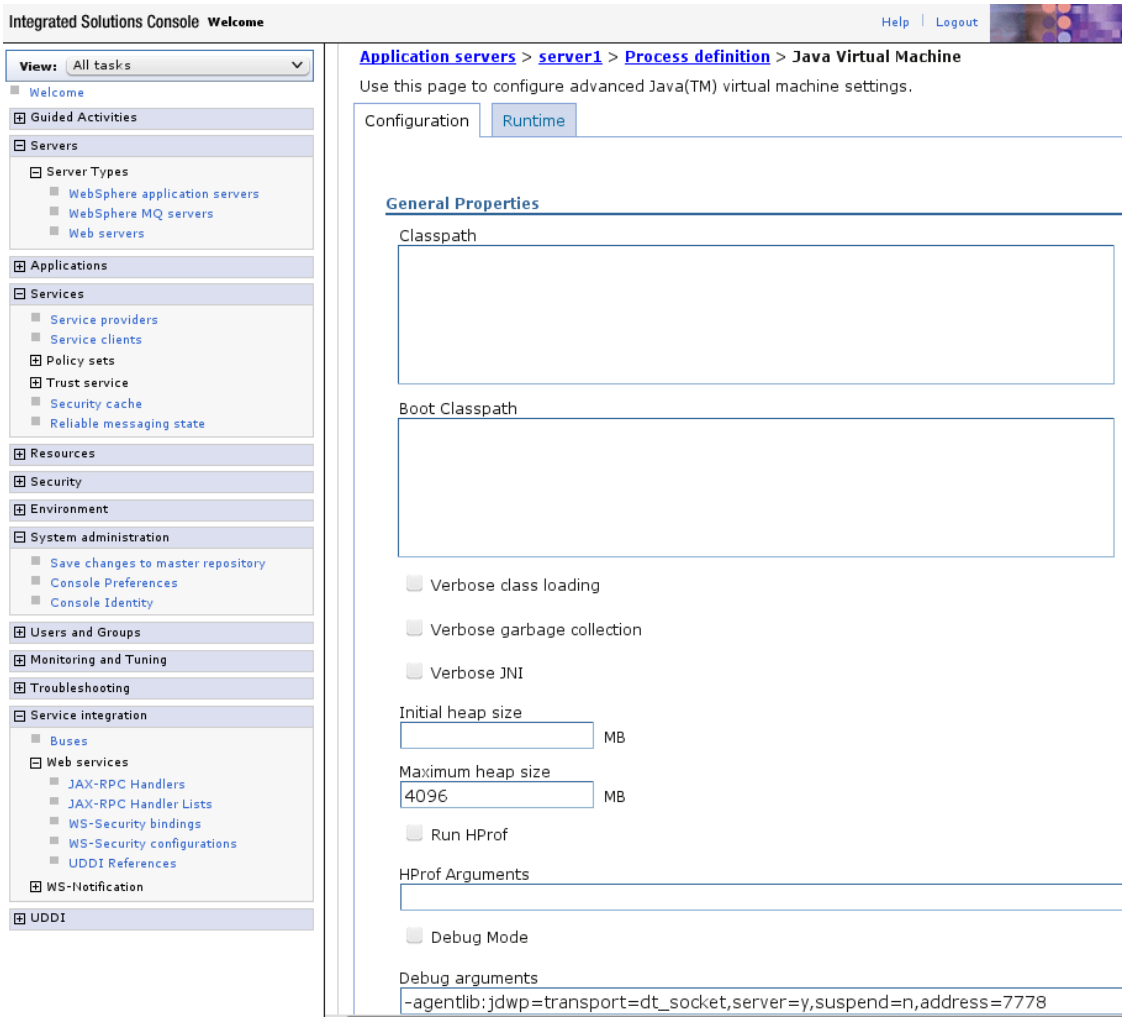


図2.1 最大ヒープ・サイズ制限の調整

- 同じページに、汎用のJVM引数があります。次のような恒久領域の制限を追加します。

  -XX:MaxPermSize=512M

- サーバーを再起動して変更を確定します。

3. WARファイルをデプロイします

- Integrated Solutions Consoleに移動します。

  (http://localhost:9060/ibm/console/)

- アプリケーション → 新規アプリケーション → 新しいエンタープライズ・アプリケーションに移動します。

4. ロギングを構成します

デフォルトでは、Websphereログ出力はlog4jを使用しません。これが原因でCloverETL Serverのロギングが正しく構成されない場合があります。したがって、CloverETL Engineの一部のメッセージではグラフ実行ログが欠落しています。そのため、log4jを使用するようにWebsphereを正しく構成することをお薦めします。

- 構成ファイルをWebsphereディレクトリ **AppServer/profiles/AppSrv01/properties/commons-logging.properties** に追加します

- 次の行をファイルに挿入します。

```
priority=1
org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
org.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger
```

- jarファイルを**clover.war/WEB-INF/lib**アーカイブから**AppServer/lib**ディレクトリにコピーします。**commons-logging-*.jar**および**log4j-*.jar**のようなファイルをすべてコピーします。

5. サーバーが稼働しているかどうかを確認します

"clover"コンテキスト・パスで実行されるアプリケーションとして**clover.war**を設定していると仮定します。次のようにポート番号が変更されています。

http://localhost:9080/clover

注記

サード・パーティのライブラリを使用した一部のCloverETL機能は、IBM Websphereで正しく機能しません

- HadoopではOracle Java 1.6+での実行のみが保証されていますが、Hadoop開発者はOracle/Sun固有のコードを削除するように努めています。Hadoop Wikiの『Hadoop Java Versions』を参照してください。
- OSGiフレームワーク(構成および初期化されている場合)は、WebServiceClientおよびEmailReaderコンポーネントの誤動作の原因になります。詳細は、組込みOSGiフレームワークの項を参照してください。

IBM WebsphereでのCloverETL Serverの構成

デフォルトのインストール(構成なし)をお薦めするのは評価を目的とする場合のみです。本番では、少なくともDBデータ・ソースとSMTPサーバーの構成をお薦めします。

構成プロパティを設定するには、いくつかの方法があります。最も一般的なのは、指定した場所のプロパティ・ファイルです。

指定した場所のプロパティ・ファイル

このようなファイルの例をあげます。

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

構成プロパティ・ファイルを示すシステム・プロパティ(または環境変数)**clover_config_file**を設定します。

- Integrated Solutions Consoleに移動します。

  (http://localhost:9060/ibm/console/)

- サーバー → Application Server → [server-name] → Javaおよびプロセス管理 → プロセス定義 → 環境エントリに移動します。

- **clover_config_file**という名前のシステム・プロパティを作成します。この値が、たとえば**cloverServer.properties**など、ファイルシステム上の構成ファイルのフルパスです。
- この変更には、IBM Websphereの再起動が必要です。

CloverETL Serverライセンスのインストール

CloverETL Serverでグラフを実行するには、有効なライセンスが必要です。ライセンスがなくてもCloverETL Serverはインストールできますが、グラフは実行できません。

1. **license.dat**ファイルを取得します。
2. CloverETL Serverの**license.file**パラメータを、**license.dat**ファイルのパスに設定します。
   - 「IBM WebsphereでのCloverETL Serverの構成」に説明されているように、"license.file"プロパティを構成プロパティ・ファイルに追加します。プロパティの値は、**license.dat**ファイルのフルパスである必要があります。
   - CloverETL Serverを再起動します。

   これを行うには、他の方法もあります。最も直接的なのは、システム・プロパティまたは環境変数**clover_license_file**を設定する方法です。(すべての方法の詳細は、18章「構成」を参照してください)。

注記

適切に構成されたCloverETLのライセンスは、ファイル**license.dat**を置き換えることによって、いつでも変更できます。この場合、ライセンスを変更したことをCloverETL Serverに認識させる必要があります。

- Web GUI → 構成 → CloverETL情報 → ライセンスに移動します。
- ライセンスの再ロード・ボタンをクリックします。
- あるいは、CloverETL Serverアプリケーションを再起動する方法もあります。

Glassfish / Sun Java System Application Server

CloverETL Serverのインストール

1. Glassfish (Tomcat)用に作成されたCloverETL Server Webアーカイブ・ファイル(**clover.war**)を取得します。
2. 次の前提条件を満たしているかどうかを確認します
   - Oracle JDKまたはJREバージョン1.6.x以上
   - Glassfish (CloverETL ServerはV2.1で動作確認済)
   - メモリー割当て設定

     JVMパラメータ**-Xms -Xmx and -XX:MaxPermSize**に関係します。詳細は、「メモリー設定」の項を参照してください。

     ヒープ・サイズと恒久領域は、XMLファイル**[glassfish]/domains/domain1/config/domain.xml**で設定できます。次の下位要素を<java-config>に追加します。

```
<jvm-options>-XX:MaxPermSize=512m</jvm-options>
<jvm-options>-Xmx2048m</jvm-options>
```

この変更には、Glassfishの再起動が必要です。

3. WARファイルをデプロイします

- WARファイルをサーバー・ファイルシステムにコピーします。CloverETL Serverは、約100 MBのWARファイルとして圧縮されているので、管理コンソールを使用してローカル・ファイルシステムから直接アップロードすることはできません。
- アプリケーション名とコンテキスト・ルートの属性に"clover"を入力します。サーバー・ファイルシステムでWARファイルのパスを指定します。
- Glassfish Admin Consoleに移動します

  デフォルトではhttp://localhost:4848/でアクセスでき、デフォルトのユーザー名/パスワードは、admin/adminadminです
- アプリケーション → Webアプリケーションに移動し、デプロイをクリックします。
- フォームを発行します

GlassfishでのCloverETL Serverの構成

デフォルトのインストール(構成なし)をお薦めするのは評価を目的とする場合のみです。本番では、少なくともDBデータ・ソースとSMTPサーバーの構成をお薦めします。

構成プロパティを設定するには、いくつかの方法があります。最も一般的なのは、指定した場所のプロパティ・ファイルです。

指定した場所のプロパティ・ファイル

このようなファイルの例をあげます。

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

構成プロパティ・ファイルを示すシステム・プロパティ(または環境変数)**clover_config_file**を設定します。

- Glassfish Admin Consoleに移動します

  デフォルトではhttp://localhost:4848/でアクセスでき、ユーザー名/パスワードは、admin/adminadminです
- 構成 → システム・プロパティに移動します。
- **clover_config_file**という名前のシステム・プロパティを作成します。この値が、たとえば**cloverServer.properties**など、ファイルシステム上のファイルのフルパスです。
- この変更には、Glassfishの再起動が必要です。

CloverETL Serverライセンスのインストール

CloverETL Serverでグラフを実行するには、有効なライセンスが必要です。ライセンスがなくてもCloverETL Serverはインストールできますが、グラフは実行できません。

ライセンスの構成は、WebSphereとほぼ同様です。

1. **license.dat**ファイルを取得します。

2. CloverETL Serverの**license.file**パラメータを、**license.dat**のパスに設定します。

   - 「指定した場所のプロパティ・ファイル」に説明されているように、"license.file"プロパティを構成プロパティ・ファイルに追加します。その値を、**license.dat**ファイルのフルパスに設定します。

   - CloverETL Serverを再起動します。

   これを行うには、他の方法もあります。最も直接的なのは、システム・プロパティまたは環境変数**clover_license_file**を設定する方法です。(すべての方法の詳細は、18章「構成」を参照してください)。

注記

適切に構成されたCloverETLのライセンスは、**license.dat**を置き換えることによって、いつでも変更できます。次に、ライセンスを変更したことをCloverETL Serverに認識させる必要があります。

- Web GUI → 構成 → CloverETL情報 → ライセンスに移動します。

- ライセンスの再ロードをクリックします。

- あるいは、CloverETL Serverを再起動する方法もあります。

JBoss

CloverETL Serverのインストール

1. JBoss用に作成されたCloverETL Server Webアーカイブ・ファイル(**clover.war**)を取得します。

2. 次の前提条件を満たしているかどうかを確認します

   - Oracle JDKまたはJREバージョン1.6.x以上

   - JBoss 6.0またはJBoss 5.1 - http://www.jboss.org/jbossas/downloadsを参照

   - jboss javaプロセスのメモリー設定。詳細は、「メモリー設定」の項を参照してください。

     メモリー制限は**[jboss-home]/bin/run.conf**で設定できます(Windows OSでは**run.conf.bat**)。

     ```
     JAVA_OPTS="$JAVA_OPTS -XX:MaxPermSize=512m -Xms128m -Xmx2048m"
     ```

     Windowsの場合も、前述と同様の手順を実行します。

3. 別のJBossサーバー構成を作成します

   この手順はオプションで、事前定義されたサーバー構成("default"など)を使用できます。ただし、次の状況でCloverETLを実行する必要がある場合は、特定のJBossサーバー構成を使用すると便利な場合があります。

   - 他のJBossアプリケーションから分離されている

   - 異なるサービス・セットがある

   - 他のアプリケーションとは異なるライブラリがクラスパスにある

JBossサーバー構成の詳細は、JBossのマニュアルを参照してください。 JBoss Server Configurations Start the Server With Alternate Configuration

4. DBデータ・ソースの構成

   JBossは組込みのDerby DBでは動作しないため、常にDB接続を構成する必要があります。この例では、MySQLを使用しました。

   - データソース構成ファイル**[jboss-home]/server/[serverConfiguration]/deploy/mysql-ds.xml**を作成します

```
<datasources>
    <local-tx-datasource>
        <jndi-name>CloverETLServerDS</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/cloverServerDB</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>root</user-name>
        <password></password>
    </local-tx-datasource>
</datasources>
```

注記

XMLファイルで特殊文字を使用する場合は、XMLエンティティとして入力する必要があります。たとえば、アンパサンド"&"を表す"&"などです。

   JNDI名は、正確に"CloverETLServerDS"とする必要があります。ここで行うのは、作成したデータベースへのDB接続パラメータ(**connection-url**、**driver-class**、**user-name**および**password**)の設定です。データベースは、最初に実行する前に空である必要があり、サーバーが自動的に表を作成します。

   JNDIデータ・ソースは、JBossでCloverETL ServerのDB接続を構成する唯一の手段です。

   - 使用しているDBに対応するJDBCドライバを、アプリケーション・サーバーのクラスパスに指定します。JDBCドライバ**mysql-connector-java-5.1.5-bin.jar**を**[jboss-home]/server/[serverConfiguration]/lib**にコピーしました

5. 次の項の説明に従って、CloverETL Serverを構成します。

6. WARファイルをデプロイします

   clover.warを**[jboss-home]/server/[serverConfiguration]/deploy**にコピーします

7. **[jboss-home]/bin/run.sh**(または、Windows OSの場合は**run.bat**)を介してjbossを起動します。特定のサーバー構成でJBossを実行する場合は、**[jboss-home]/bin/run.sh -c [serverConfiguration]**のようなパラメータとして指定されている必要があります。serverConfigurationが指定されていない場合は、"default"が使用されます。

   すべてのアプリケーションが開始されるまで数分かかる場合があります。

8. JBossの応答とCloverETL Serverの応答を確認します

   - デフォルトでは、JBoss管理コンソールにはhttp://localhost:8080/でアクセスできます。デフォルトのユーザー名/パスワードは、admin/adminです

   - デフォルトでは、http://localhost:8080/でCloverETL Serverにアクセスできます。

9. 必要な場合、デフォルトとサンプルのサンドボックス(**temp**ディレクトリに自動的に作成される)をファイルシステム上で適切なディレクトリに移動します。

- これらのサンドボックスは、最初のデプロイ時に自動的に作成され、**web-app**ディレクトリに配置されます。このディレクトリは、特定のデプロイごとに固有です。なんらかの理由でWebアプリケーションをデプロイした場合に、このディレクトリが作成されます。このため、サンドボックスは変化しない場所に移動することをお薦めします。

JBossでのCloverETL Serverの構成

デフォルトのインストール(構成なし)をお薦めするのは評価を目的とする場合のみです。本番では、少なくともDBデータ・ソースとSMTPサーバーの構成をお薦めします。

構成プロパティを設定するには、いくつかの方法があります。最も一般的なのは、指定した場所のプロパティ・ファイルです。

指定した場所のプロパティ・ファイル

- **cloverServer.properties**を適切なディレクトリに作成します

```
datasource.type=JNDI datasource.jndiName=java:/CloverETLServerDS
jdbc.dialect=org.hibernate.dialect.MySQLDialect
license.file=/home/clover/config/license.dat
```

  datasource.typeおよび**datasource.jndiName**プロパティは変更せず、DBサーバー(18章「構成」)に従って適切なJDBC言語を設定します。また、ライセンス・ファイルのパスも設定します。

- システム・プロパティ(または環境変数) **clover_config_file**を設定します。

  これには、前の手順で作成した**cloverServer.properties**ファイルのフルパスを指定する必要があります。

  最も簡単なのは、次のように**[jboss-home]/bin/run.sh**でJavaのパラメータを設定する方法です。

```
export JAVA_OPTS="$JAVA_OPTS -Dclover_config_file=/home/clover/config/
cloverServer.properties"
```

  JAVA_OPTSプロパティの他の設定、つまり前述したメモリー設定はオーバーライドしないでください。

  Windows OSの場合は、**[jboss-home]/bin/run.conf.bat**を編集し、JVMにオプションが渡されるセクションに次の行を追加します。

```
set JAVA_OPTS=%JAVA_OPTS% -Dclover_config_file=C:\JBoss6\cloverServer.properties
```

- この変更には、JBossの再起動が必要です。

CloverETL Serverライセンスのインストール

CloverETL Serverでグラフを実行するには、有効なライセンスが必要です。ライセンスがなくてもCloverETL Serverはインストールできますが、グラフは実行できません。

1. **license.dat**ファイルを取得します。

   clover_license.warのみがある場合は、一般的なzipアーカイブとして抽出し、**WEB-INF**サブディレクトリで**license.dat**を検索します

2. CloverETL Serverのパラメータ**license.file**に、**license.dat**のパスを指定します。

   ライセンスを構成するには、前の項で説明されているように**cloverServer.properties**ファイルで**license.file**プロパティを設定するのが最も確実です。

   これを行うには、他の方法もあります。(すべての方法の詳細は、18章「構成」を参照してください)。

3. 構成を変更するには、アプリケーション・サーバーの再起動が必要です。

注記

CloverETLのライセンスは、**license.dat**ファイルを置き換えることによって、いつでも変更できます。この場合、ライセンスを変更したことをCloverETL Serverに認識させる必要があります。

- Web GUI → 構成 → CloverETL情報 → ライセンスに移動します。
- ライセンスの再ロードをクリックします。
- あるいは、CloverETL Serverアプリケーションを再起動する方法もあります。

Oracle WebLogic Server

CloverETL Serverのインストール

1. WebLogic用に作成されたCloverETL Server Webアーカイブ・ファイル(**clover.war**)を取得します。

2. 次の前提条件を満たしているかどうかを確認します

   - Oracle JDKまたはJREバージョン1.6.x以上
   - WebLogic (CloverETL ServerはWebLogic Server 11g (10.3.6)およびWebLogic Server 12c (12.1.1)で動作確認済です。http://www.oracle.com/technetwork/middleware/ias/downloads/wls-main-097127.htmlを参照してください)

     WebLogicが稼働している必要があり、ドメインを構成する必要があります。これは、http://hostname:7001/console/ (7001はHTTPのデフォルト・ポート)で管理コンソールに接続して確認できます。ユーザー名とパスワードは、インストール中に指定します。

   - メモリー割当て設定

     これに関係するのは、JVMパラメータの-Xms -Xmxと、-XX:MaxPermSizeです

     詳細は、「メモリー設定」の項を参照してください。

     これは、次の内容を追加して設定できます。

     ```
     export JAVA_OPTIONS='$JAVA_OPTIONS -Xms128m -Xmx2048m -XX:MaxPermSize=512m' to the
      start script
     ```

     この変更には、ドメインの再起動が必要です。

3. HTTP基本認証の構成を変更します

   - HTTPリクエストで"Authentication"ヘッダーが検出されると、WebLogicはそれ自体のレルムでユーザーを検索しようとします。CloverETLでユーザーを認証するためには、この動作を停止する必要があります。

- 構成ファイル**[domainHome]/config/config.xml**を変更します。**<enforce-valid-basic-auth-credentials>false</enforce-valid-basic-auth-credentials>**という要素を、**<security-configuration>**という要素(終了タグの直前)に追加します。

4. WARファイル(またはアプリケーション・ディレクトリ)をデプロイします

   - WebLogicバージョンに適したclover.warを取得します。
   - WebLogic Server管理コンソールを使用して、clover.warをデプロイします。詳細は、『Oracle Fusion Middleware管理者ガイド』(http://docs.oracle.com/cd/E23943_01/core.1111/e10105/toc.htm)を参照してください。

5. ライセンス(およびその他の構成プロパティ)を構成します

   - 後述の別項を参照してください

6. CloverETL ServerのURLを確認します

   - Webアプリケーションは、デプロイ後に自動的に起動するため、稼働していることを確認できます。

     デフォルトでは、http://host:7001/cloverでCloverETL Serverにアクセスできます。ポート7001は、デフォルトのWebLogic HTTPコネクタ・ポートです。

WeblogicでのCloverETL Serverの構成

デフォルトのインストール(構成なし)をお薦めするのは評価を目的とする場合のみです。本番では、少なくともDBデータ・ソースとSMTPサーバーの構成をお薦めします。

構成プロパティを設定するには、いくつかの方法があります。最も一般的なのは、指定した場所のプロパティ・ファイルです。

指定した場所のプロパティ・ファイル

cloverServer.propertiesを適切なディレクトリに作成します。

構成ファイルには、DBデータソースの構成、SMTP接続の構成などが含まれます。

構成プロパティ・ファイルを示すシステム・プロパティ(または環境変数)
clover_config_fileを設定します

- WebLogicドメイン起動スクリプト**[domainHome]/startWebLogic.sh**で、JAVA_OPTIONS変数を設定します

```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dclover_config_file=/path/to/clover-config.properties
```

- この変更には、Weblogicの再起動が必要です。

CloverETL Serverライセンスのインストール

CloverETL Serverでグラフを実行するには、有効なライセンスが必要です。ライセンスがなくてもCloverETL Serverはインストールできますが、グラフは実行できません。

1. **license.dat**ファイルを取得します。

   clover_license.warのみがある場合は、一般的なzipアーカイブとして抽出し、**WEB-INF**サブディレクトリで**license.dat**を検索します

2. CloverETL Serverのパラメータ**license.file**に、**license.dat**ファイルのパスを指定します

ライセンスを構成するには、前の項で説明されているように**cloverServer.properties**ファイルで**license.file**プロパティを設定するのが最も確実です。

これを行うには、他の方法もあります。(すべての方法の詳細は、18章「構成」を参照してください)。

3. 構成を変更するには、アプリケーション・サーバーの再起動が必要です。

注記

適切に構成されたCloverETLのライセンスは、ファイル**license.dat**を置き換えることによって、いつでも変更できます。この場合、ライセンスを変更したことをCloverETL Serverに認識させる必要があります。

- Web GUI → 構成 → CloverETL情報 → ライセンスに移動します。
- ライセンスの再ロードをクリックします。
- あるいは、CloverETL Serverアプリケーションを再起動する方法もあります。

インストール中に想定される問題

CloverETL Serverは、様々なアプリケーション・サーバー、データベースおよびJVM実装で動作するユニバーサルJEEアプリケーションとして位置付けられているため、インストール中に問題が発生する可能性があります。それらは、サーバー環境の適切な構成によって解決できます。この項では、構成に関するヒントについて説明します。

Derbyにおけるメモリーの問題

これまで正常に稼働していたサーバーが突然、大量のリソース(CPU、メモリー)を消費し始めた場合は、内部Derby DBの実行が原因になっている可能性があります。代表的な原因は、Apache Tomcatの不正または不完全なシャットダウンと、Apache Tomcatのパラレル(再)起動です。

解決方法: 標準(スタンドアロン)データベースに移行します。

修正方法:CloverETL Serverを再デプロイします。

1. Apache Tomcatを停止し、他のインスタンスが実行中でないことを確認します。実行中の場合は、強制終了します。
2. **config.properties**を**webapps/clover/WEB-INF**および**clover/WEB-INF/sandboxes** からバックアップします(ここにデータがある場合)。
3. **webapps/clover**ディレクトリを削除します。
4. Apache Tomcatサーバーを起動します。自動的にClover Serverが再デプロイされます。
5. DesignerからもWebからも接続できることを確認します。
6. Apache Tomcatをシャットダウンします。
7. **config.properties** をリストアし、通常のデータベースを指定します。
8. Apache Tomcatを起動します。

JAVA_HOMEまたはJRE_HOME環境変数が定義されていない

アプリケーション・サーバー(大部分はTomcat)を起動しようとしてこのエラー・メッセージが表示された場合は、次の処理を実行してください。

Linuxの場合:

次の2つのコマンドを使用して、サーバー上の変数のパスを設定します。

- **[root@server /] export JAVA_HOME=/usr/local/java**
- **[root@server /] export JRE_HOME=/usr/local/jdk**

最後の手順として、アプリケーション・サーバーを再起動します。

Windows OSの場合:

JAVA_HOMEをJDKインストール・ディレクトリ(**C:\Program Files\java\jdk1.6.0**など)に設定します。オプションで、**JRE_HOME**もJREのベース・ディレクトリ、たとえば**C:\Program Files\java\jre6**に設定します。

重要

JREのみインストールしている場合は、**JRE_HOME**のみを指定します。

効果のないApache Tomcatコンテキスト・パラメータ

Tomcatは、コンテキスト・パラメータの一部を無視することがあります。これにより、CloverETL Serverが構成されているように見えても一部のみしか構成されていないため、CloverETL Serverの異常動作の原因になります。一部のパラメータは受け入れられますが、一部は無視されます。通常は正常に動作しますが、一部の環境でこの状況が発生することがあります。このような動作は一貫性があるため、再起動後も同じ状況になります。おそらく、Tomcatの問題(https://issues.apache.org/bugzilla/show_bug.cgi?id=47516およびhttps://issues.apache.org/bugzilla/show_bug.cgi?id=50700)に関連しています。この問題を回避するには、コンテキスト・パラメータのかわりにプロパティ・ファイルを使用してCloverETL Serverを構成してください。

Tomcatのログ・ファイルcatalina.outがWindows上で欠落

Windows用のTomcat起動バッチ・ファイルは、アプリケーションの標準出力を含むcatalina.outファイルを作成するように構成されていません。Tomcatをコンソールで起動せず、なんらかの問題が発生した場合は、catalina.outが重要になることがあります。あるいは、Tomcatをコンソールで実行している場合でも、エラー・メッセージの表示後、すぐ自動的に閉じることがあります。

次の手順を実行して、catalina.outの作成を有効にしてください。

- [tomcat_home]/bin/catalina.batを変更します。"_EXECJAVA"変数が設定されている行に、パラメータ"/B"を追加します。該当する行は2行です。次のようになります。

  set _EXECJAVA=start /B [the rest of the line]
  パラメータ/Bを指定すると、"start"コマンドで新しいコンソール・ウィンドウが開かず、コマンドをそれ自身のコンソール・ウィンドウで実行します。

- 次の1行のみを含む新しい起動ファイル、たとえば[tomcat_home]/bin/startupLog.batを作成します。

  catalina.bat start > ..\logs\catalina.out 2<&1
  これでTomcatは通常のように実行されますが、標準出力がコンソールではなくcatalina.outファイルに変更されます。

次に、[tomcat_home]/bin/startup.batのかわりに新しい起動ファイルを使用します

JVMを待機中にタイムアウト

Jettyアプリケーション・サーバーが正常に起動し、Clover Serverを起動できない場合は、ラッパーのJVMに対する待機時間が長すぎた可能性があります(これはメモリー不足の問題とみなされま

す)。**[JETTY_HOME]\logs\jetty-service.log**で、次のような行があるかどうかを調べてください。

```
Startup failed: Timed out waiting for signal from JVM.
```

見つかった場合は**[JETTY_HOME]\bin\jetty-service.conf**を編集し、次の行を追加します。

```
wrapper.startup.timeout=60
wrapper.shutdown.timeout=60
```

これで問題が解決しない場合は、両方の値を120に設定してみます。デフォルトのタイムアウトは、どちらも30です。

clover.warがWebsphereでデフォルトのコンテキスト(Windows OS)

コンテキスト・パスを指定せずに**clover.war**をIBM Websphereサーバーにデプロイしている場合は、コンテキスト・ルートで他のアプリケーションが実行中でないかを確認する必要があります。WebsphereでClover Serverを起動できない場合は、ログを確認して次のようなメッセージがあるかどうか調べます。

```
com.ibm.ws.webcontainer.exception.WebAppNotLoadedException:
Failed to load webapp: Failed to load webapp: Context root /* is already bound.
Cannot start application CloverETL
```

見つかった場合は、ここで説明する場合に該当しています。問題を解決する最も簡単な方法は、他の(サンプル)アプリケーションをすべて停止し、サーバーで**clover.war**のみが実行されている状態にすることです。これによって、これ以降はコンテキスト・ルート(例: http://localhost:9080/)でサーバーが稼働するようになります。

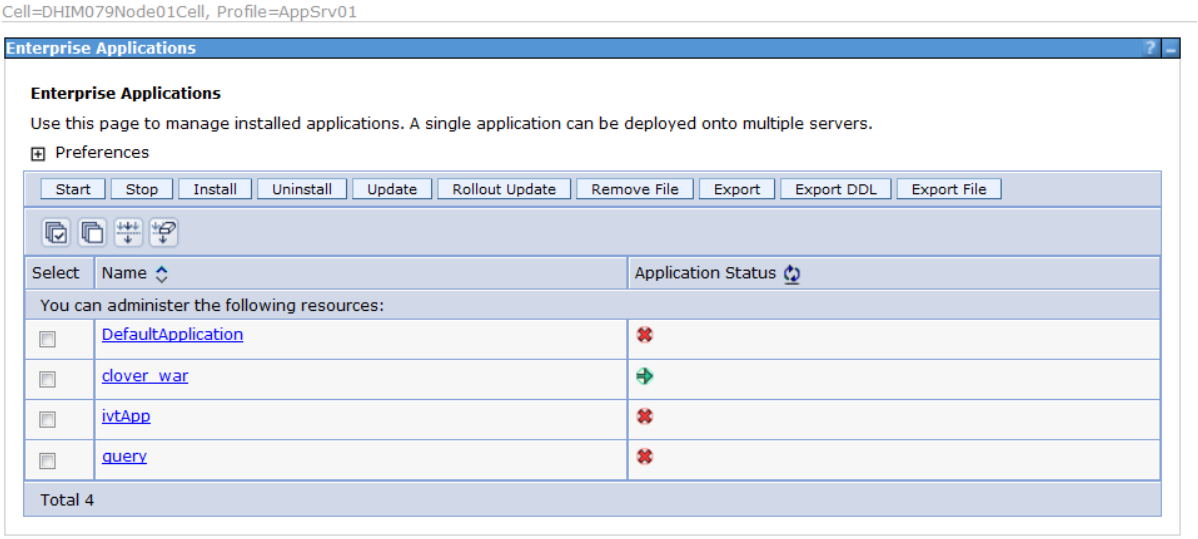


図2.2 IBM Websphereで稼働しているアプリケーションがClover Serverのみの場合

Linux上のTomcat 6.0 -デフォルトDB

Linuxで内部(デフォルト)データベースを使用している場合、Clover Serverが特に理由もなく最初の起動時にエラーになることがあります。可能性としては、**/var/lib/tomcat6/databases**ディレクトリが作成されていない場合があります(親フォルダに対するアクセス権が原因)。

解決方法: 自分でディレクトリを作成し、サーバーを再起動してみます。単純な修正ですが、Tomcat Web管理者を介して**WAR**ファイルとしてデプロイされたClover Serverに対しては有効であることが確認されています。

derby.system.homeにアクセスできない

サーバーを起動できない場合は、ログに次のようなメッセージがあります。

```
java.sql.SQLException: Failed to start database 'databases/cloverserver'
```

この場合、次の例外で詳細を確認します。その後、**derby.system.home**システム・プロパティの設定を確認します。そこでアクセスできないディレクトリが指定されているか、ファイルが他のプロセスによってロックされている可能性があります。特定のディレクトリをシステム・プロパティで設定してください。

複数の環境変数と複数のCloverETL Serverインスタンスが1つのマシン上に存在

clover_license_fileまたは**clover_config_file**などの環境変数を設定している場合、複数のCloverETL Serverは実行できないことに注意してください。したがって、複数のインスタンスを同時に実行する必要がある場合は、別の方法でパラメータを設定してください(すべての方法の詳細は、18章「構成」を参照)。これは、環境変数はすべてのアプリケーションによって共有され、構成も共有するため、予期しないエラーが発生するためです。環境変数のかわりに、システム・プロパティ(-D接頭辞を含むパラメータ-Dclover_config_fileを使用してアプリケーション・コンテナ・プロセスに渡されます)を使用できます。

パスに特殊文字とスラッシュ

サーバーを操作する際には、通常より厳格にフォルダの命名規則に従う必要があります。サーバー・パスに特殊文字は使用しないでください。空白、アクセント記号、発音符号などはすべて非推奨です。Windowsシステムでは命名によく使用される文字です。ただしこれは問題の原因になり、しかも特定が困難です。奇妙なエラーが発生し、その原因を特定できない場合は、アプリケーション・サーバーを次のように確実な場所にインストールしてみてください。

C:\JBoss6

同様に、たとえばClover Serverライセンス・ファイルを指定する場合などで、***.properties**ファイル内部のパスにはスラッシュを使用し、円記号は使用しないでください。誤って円記号を使用するとエスケープ文字と解釈され、サーバーが正常に動作しないことがあります。次に正しいパスの例を示します。

license.file=C:/CoverETL/Server/license.dat

JAXBおよびそれ以前のバージョンのJVM 1.6

バージョン1.3以降のCloverETL Serverには、jaxb 2.1のライブラリが含まれています。これが、jaxb 2.0を含むJVM 1.6の初期バージョンでは競合の原因になることがあります。ただし、JDK6 Update 4リリースでは最終的にjaxb 2.1が含まれるようになったため、これより新しいバージョンのJVMに更新すれば、競合の可能性は解決されます。

ファイルシステムの権限

アプリケーション・サーバーは、ファイルシステム上で適切な読取り/書込み権限を持つOSユーザーが実行する必要があります。アプリケーション・サーバーが初めてルート・ユーザーによって実行された場合に問題が発生することがあり、ログおよびその他の一時ファイルはルート・ユーザーによって作成されます。同じアプリケーション・サーバーを別のユーザーが実行すると、ルートのファイルには書き込めないためエラーになります。

JMS APIとJMSのサード・パーティ製ライブラリ

JMSライブラリが欠落していてもサーバーの起動には失敗しませんが、アプリケーション・サーバーへのデプロイ時には問題になるため、この項で取り上げることにします。

clover.war自体にはjms.jarが含まれていないので、アプリケーション・サーバーのクラスパスに指定する必要があります。ほとんどのアプリケーション・サーバーにはデフォルトでjms.jarがありますが、Tomcatなどにはjms.jarがありません。そのため、JMS機能が必要な場合は、jms.jarを明示的に追加する必要があります。

JMSタスク機能を使用する場合、サーバーのクラスパスにはサード・パーティのライブラリも必要です。JMS Reader/Writerコンポーネントで外部ライブラリの指定が許可される場合でも、同じアプローチが推奨されます。これらのライブラリにはたびたびメモリー・リークが存在し、それが原因で"OutOfMemoryError: PermGen space"が発生するためです。

メモリー設定

Java仮想マシンの現在の実装では、JVMシステム・プロセスに対してはグローバルなメモリー構成のみが可能です。そのため、アプリケーション・サーバー全体が、そこで実行されるWARやEARまで含めて、1つのメモリー領域を共有します。

JVMメモリーのデフォルト設定は、CloverETL Serverとアプリケーション・コンテナを実行するには小さすぎます。IBM Websphereなど一部のアプリケーション・サーバーは独自にJVMのデフォルトを増やしますが、それでもまだメモリー設定は十分ではありません。

最適なメモリー上限は多くの条件、たとえばCloverETLで実行される変換などにより異なります。最大の上限は、恒久的に割り当てられるメモリー量ではなく、超えてはならない上限です。この上限を超えた場合、OutOfMemoryErrorが発生します。

"Xms"と"Xmx"のJVMパラメータを調整して、メモリー・ヒープの最小サイズと最大サイズを設定できます。使用するアプリケーション・コンテナによって、設定の変更方法は様々です。

変換に必要なメモリー量がわからない場合は、最大1から2 GBのヒープ・メモリーをお薦めします。変換の作成中に**OutOfMemoryError**が発生した場合は、この上限をさらに増やすことができます。

クラスをロードするためのメモリー領域(いわゆる"PermGen領域")は、ヒープ・メモリーと区別されているため、JVMパラメータ"-XX:MaxPermSize"で設定できます。デフォルトでは、これが64 MBしかなく、エンタープライズ・アプリケーションには十分ではありません。ここでも、適切なメモリー上限は様々な条件で異なりますが、512 MBを指定しておくとほとんどの場合に十分です。PermGen領域の最大値が低すぎる場合は、**OutOfMemoryError: PermGen space**が発生することがあります。

設定方法の詳細は、個々のコンテナに関する項を参照してください。

新しいバージョンへのサーバーのアップグレード

アップグレード時の一般的な注意事項

- CloverETL Serverのアップグレードには停止時間が必要です。メンテナンス・ウィンドウを計画してください
- 正常なアップグレードには約30分必要です。ロールバックには30分必要です
- 本番環境に移行する前に、最初に開発/テスト環境で次の手順を実行します

アップグレードの前提条件

- 新しいCloverETL Server Webアプリケーション・アーカイブ(使用されるアプリケーション・サーバーに適した**clover.war**)と使用可能なライセンス・ファイルがあること

- 使用可能な特定のCloverETLバージョンのrelease notes(および現在のバージョンとアップグレード先のバージョン間のすべてのバージョン)があること
- 特定のCloverETLバージョンのKnown Issues & Compatibilityについてグラフとジョブを更新およびテストすること。
- デフォルトの場所から外部化されたCloverETL Server構成プロパティ・ファイルがあること。「構成ソースとその優先度」を参照してください
- CloverETL Serverが構成を格納するスタンドアロン・データベース・スキーマ。「DB接続構成の例」を参照してください
- CloverETL Serverが正常に実行され、ジョブの実行に対応することを検証するためにいつでも実行できるテスト・グラフ付きの別のサンドボックスがあること。

アップグレード手順

1. すべてのサンドボックスを一時停止し、実行中のグラフの処理の終了を待ちます
2. CloverETL Serverアプリケーション(または、クラスタ・モードで実行している場合はすべてのサーバー)を停止します
3. 既存のCloverETLデータベース・スキーマをバックアップします(データベース・スキーマに対する変更が必要な場合は、新しいサーバーを最初に起動したとき自動的にその変更が行われます)
4. 既存のCloverETL Webアプリケーション・アーカイブ(**clover.war**)およびライセンス・ファイル(すべてのノード上)をバックアップします
5. 既存のCloverETLサンドボックス(すべてのノード上)をバックアップします
6. CloverETL Server Webアプリケーションを再デプロイします。その方法はアプリケーション・サーバーによって異なります。サポートされているすべてのアプリケーション・サーバーのインストールの詳細は、「エンタープライズ・サーバー」を参照してください。再デプロイすると、新しいサーバーが以前のバージョンの構成に基づいて構成されます。
7. 新しいライセンス・ファイルをインプレースでコピーします(すべてのノード上)。ライセンス・ファイルは、一意の文字列セットを含むテキストとして出荷されます。次の場合に応じて操作してください。
   - 新しいライセンスをファイル(***.dat**)として受け取った場合は、単に古いライセンス・ファイルを上書きします。
   - ライセンス・テキストを電子メールなどで送信された場合は、ライセンスの内容(**Company**と**END LICENSE**の間の全テキスト)を**clover-license.dat**という新規ファイルにコピーします。次に、古いライセンス・ファイルを新しいライセンス・ファイルで上書きします。
8. CloverETL Serverアプリケーションを起動します(すべてのノード上)
9. CloverETL Serverコンソールのすべてのタブのコンテンツ、特にスケジュールとイベント・リスナーが正常であることを確認します
10. 特定のバージョンのCloverETL Serverと互換性のあるグラフを更新します(これは事前に準備およびテストする必要があります)
11. テスト・サンドボックスを再開し、テスト・グラフを実行して機能を検証します
12. すべてのサンドボックスを再開します

ロールバックの手順

1. CloverETL Serverアプリケーションをシャットダウンします

2. CloverETL Server Webアプリケーション(**clover.war**)およびライセンス・ファイル(すべてのノード上)をリストアします

3. CloverETL Serverデータベース・スキーマをリストアします

4. CloverETLサンドボックス(すべてのノード上)をリストアします

5. CloverETL Serverアプリケーションを起動します(すべてのノード上)

6. テスト・サンドボックスを再開し、テスト・グラフを実行して機能を検証します

7. すべてのサンドボックスを再開します

重要

評価バージョンの場合 – ライセンスをアップグレードするだけでは十分ではありません。評価サーバーからエンタープライズ・サーバーに移行する場合は、デフォルトの構成とデータベースは使用しないでください。かわりに、時間はかかりますが本番環境に応じてClover Serverを構成してください。

第3章 サーバー側ジョブ・ファイル – サンドボックス

サンドボックスは、すべてのプロジェクトの変換グラフ・ファイル、ジョブフロー、データおよびその他のリソースを格納する場所です。これは、Designerプロジェクトに相当するサーバー側の機能です。サーバーは、ユーザー権限管理やグローバルのサンドボックスごとの構成オプションなどの追加機能をサンドボックスに追加します。

サーバーおよびDesignerが統合されているため、Designerワークスペースのサーバー・プロジェクトを使用してサーバー・サンドボックスに接続できます。このようなプロジェクトはリモート・ファイル・システムのように動作し、すべてのデータがサーバーに格納されてリモートでアクセスされます。しかし、ローカル・プロジェクトと同じ方法で、ファイルのコピー・アンド・ペースト、グラフの作成、編集およびデバッグなどのすべての操作をサーバー・プロジェクトで実行できます。サーバーへの接続の構成の詳細は、CloverETL Designerのマニュアルを参照してください。

技術的には、サンドボックスはサーバー・ホスト・ファイル・システム上の専用ディレクトリであり、そのコンテンツはサーバーによって管理されます。パーティション化されたサンドボックスなどの高度なタイプのサンドボックスには、分散パラレル処理を可能にするために複数の場所があります(詳細は、24章「クラスタリング」を参照してください)。サンドボックスには、別のサンドボックスを含めることができません。これはプロジェクトの単一ルート・パスです。

すべてのサンドボックスをCloverETL Serverインストールの外部のフォルダに入れることをお薦めします(/var/cloveretl/sandboxesなど)。ただし、各サンドボックスは、必要に応じて他から独立したファイル・システムに配置できます。格納するフォルダとそのすべてのコンテンツは、CloverETL Server/アプリケーション・サーバーを実行しているユーザーに対する読取り/書込み権限が必要です。

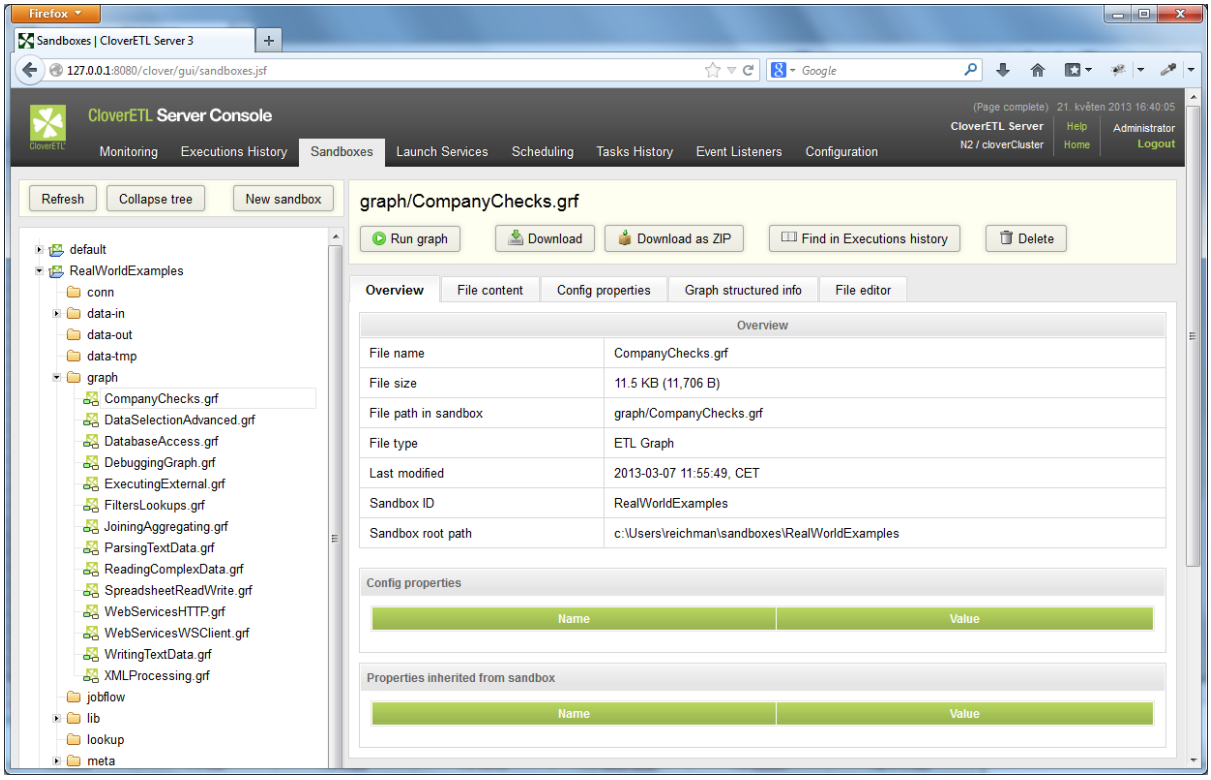


図3.1 CloverETL Server Web GUIのサンドボックスセクション

各サンドボックスは、次の属性で定義されます。

表3.1 サンドボックスの属性

サンドボックスID	サンドボックスの一意な名前。サンドボックスを識別するためにサーバーAPIで使用されます。識別子の一般的なルールを満たしている必要があります。サンドボックスを作成する際にユーザーによって指定され、後から変更も可能です。注意: いずれかのCS APIクライアントですでに使用されている可能性があるため、変更はお薦めしません。
サンドボックス	表示用にのみ使用されるサンドボックス名。サンドボックスを作成する際にユーザーによって指定され、後から変更も可能です。
サンドボックスのルート・パス	サーバー側ファイルシステムにおけるサンドボックス・ルートの絶対パス。サンドボックスを作成する際にユーザーによって指定され、後から変更も可能です。この属性はスタンドアロン・モードでのみ使用されます。クラスタ・モードの詳細は、24章「クラスタリング」を参照してください。
所有者	サンドボックスを作成する際に自動的に設定されます。後から変更も可能です。

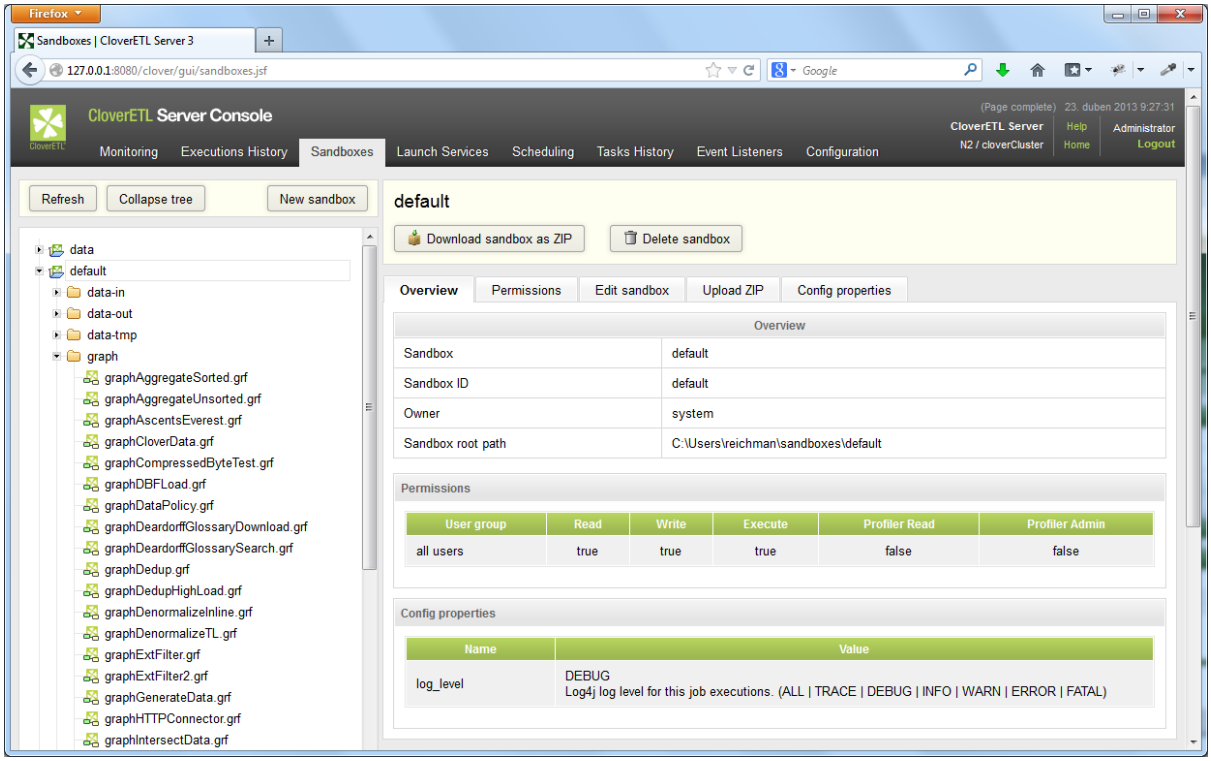


図3.2 CloverETL Server Web GUIのサンドボックスの詳細

ETLグラフまたはジョブフローからのファイルの参照

一部のコンポーネントでは、ファイルシステム上のリソースへの参照として、ファイルのURL属性を指定できます。また、ファイルシステム上のファイルへの参照として、外部メタデータ、ルックアップまたはDB接続定義も指定されます。CloverETL Serverでは、この関係を指定する方法がいくつかあります。

- 相対パス

  グラフにおける相対パスはすべて、ジョブ・ファイル(ETLグラフまたはジョブフロー)を含む同じサンドボックスのルートへの相対パスとみなされます。

- sandbox:// URLs

サンドボックスのURLを指定すると、スタンドアロンのCloverETL Serverまたはクラスタで異なるサンドボックスからリソースを参照できます。クラスタ環境では、特定のクラスタ・ノードでしかリソースにアクセスできない場合に、CloverETL Serverがリモート・ストリーミングを透過的に管理します。

サンドボックスURLの詳細は、「コンポーネント・データ・ソースとしてのサンドボックス・リソースの使用」を参照してください。

サンドボックスの内容のセキュリティと権限

各サンドボックスには、サンドボックスの作成時に所有者が設定されます。このサンドボックスに対して、所有者であるユーザーと管理者が無制限の権限を持ちます。サンドボックスの設定によっては、他のユーザーがアクセス権を持つ場合もあります。

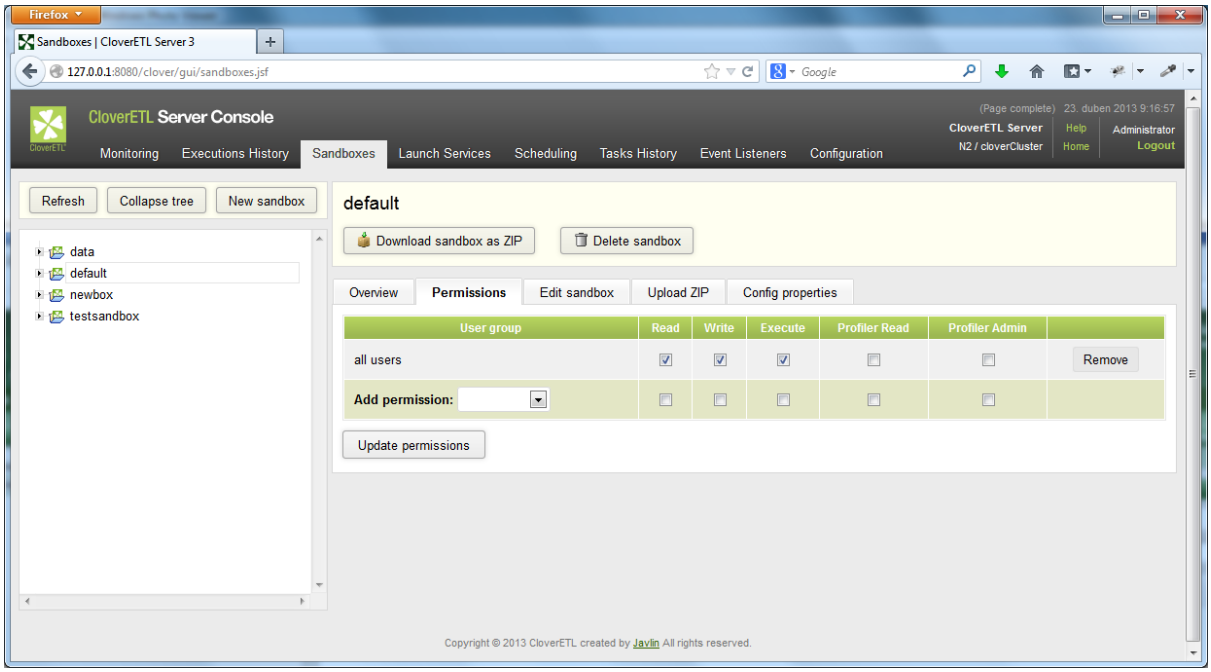


図3.3 CloverETL Server Web GUIのサンドボックスの権限

特定のサンドボックスに対する権限は、サンドボックスの詳細の権限タブで変更が可能です。このタブで、指定したユーザー・グループに特定の操作の実行を許可できます。

3タイプの操作があります。

表3.2 サンドボックスの権限

読取り	ユーザーは、サンドボックスのリストでこのサンドボックスを表示できます。
書込み	ユーザーは、CS APIを介してサンドボックスの中のファイルを変更できます。
実行	ユーザーは、このサンドボックスの中のジョブを実行できます。注意: グラフ・イベント・リスナーおよび同様の機能によって実行されるジョブは、実際には、イベントのソースであるジョブと同じユーザーによって実行されます。詳細は、グラフ・イベント・リスナーを参照してください。スケジュール・トリガーによって実行されるジョブは、実際にはスケジュール所有者によって実行されます。6章「スケジューリング」を参照してください。ジョブがサンドボックスからのファイル(メタデータなど)を必要とする場合、ユーザーには読取り権限も必要で、読取り権限がないと実行が失敗します。

プロファイラ読取り	ユーザーは、サンドボックスから実行されたプロファイラ・ジョブの結果を表示できます。
プロファイラ管理者	ユーザーは、サンドボックスから実行されたプロファイラ・ジョブの結果を管理できます。

これらの権限によって、特定のサンドボックスの内容に対するアクセス権が変更されます。また、サンドボックスの構成に関する操作、たとえばサンドボックスの作成、編集、削除などを実行する権限を構成することもできます。詳細は、5章「ユーザーとグループ」を参照してください。

サンドボックスの内容

サンドボックスには、ジョブフロー、グラフ、メタデータ、外部接続および関連するすべてのファイルが含まれます。ファイル、特にグラフやジョブフローのファイルは、サンドボックス・ルートからの相対パスで識別されます。したがって、特定のジョブ・ファイルを識別するには、サンドボックスとサンドボックスにおけるパスの2つの値が必要です。ジョブフローまたはETLグラフのパスを、通常はジョブ・ファイルと呼びます。

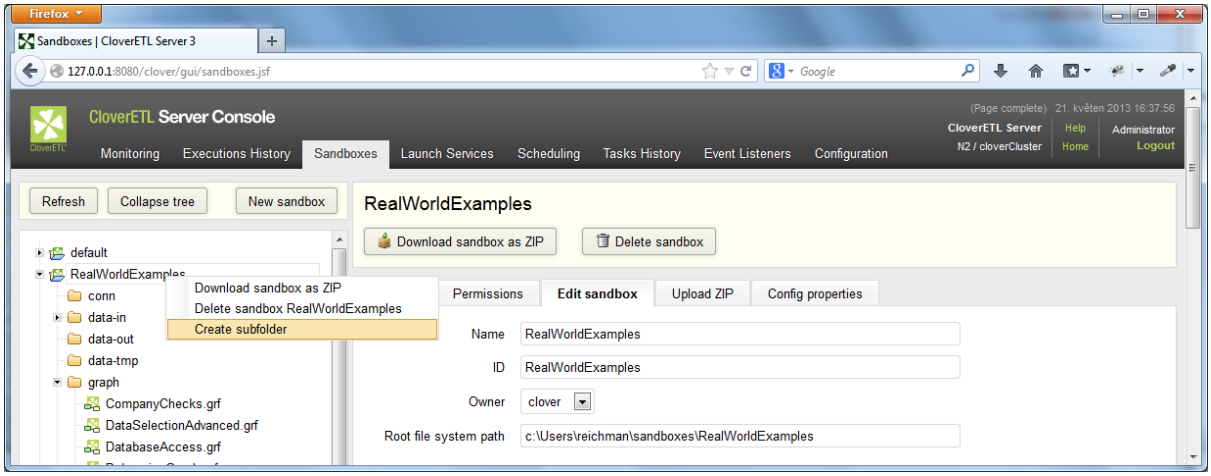


図3.4 Web GUI - サンドボックスセクション - サンドボックスのコンテキスト・メニュー

Web GUIのサンドボックスセクションはファイル・マネージャではありませんが、サンドボックス管理に便利な機能があります。

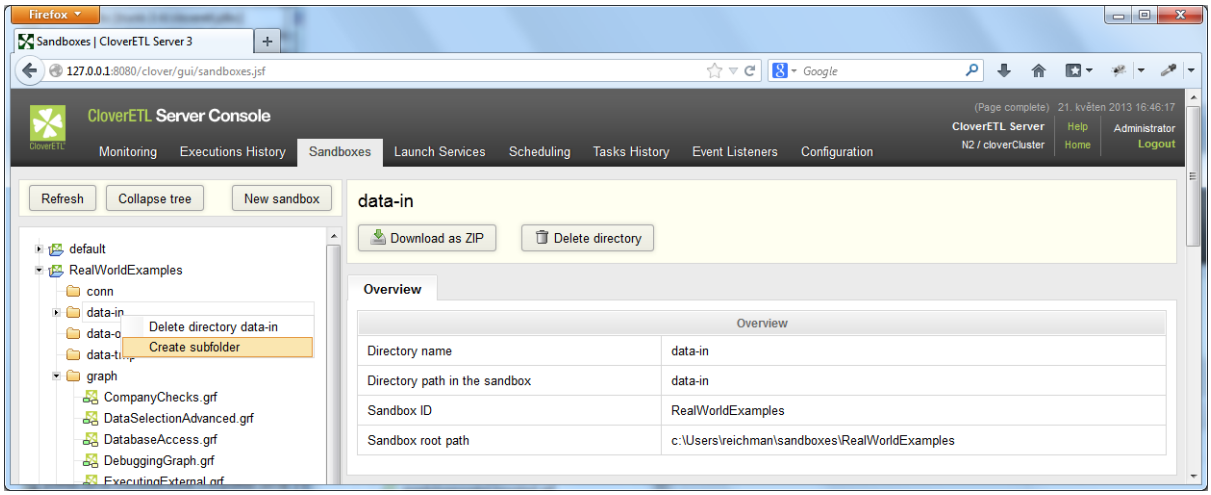


図3.5 Web GUI - サンドボックスセクション - フォルダのコンテキスト・メニュー

ZIPとしてのサンドボックスのダウンロード

左パネルでサンドボックスを選択すると、Web GUI右側のツールバーに、ZIPとしてのサンドボックスのダウンロード・ボタンが表示されます。

作成したZIPには、ファイルシステムと同じ階層にある読取り可能なサンドボックス・ファイルがすべて含まれます。このZIPファイルを使用して、同じサンドボックス、または別のサーバー・インスタンスにある他のサンドボックスにファイルをアップロードできます。

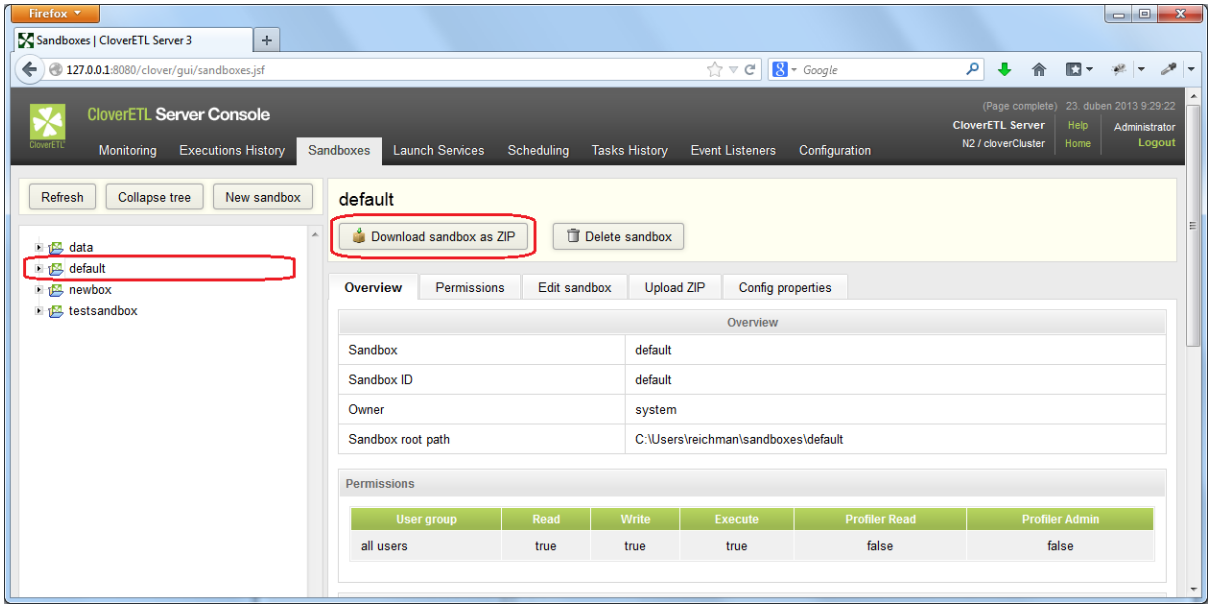


図3.6 Web GUI - ZIPとしてのサンドボックスのダウンロード

サンドボックスへのZIPのアップロード

左パネルでサンドボックスを選択します。選択したサンドボックスに対して、書込み権限が必要です。次に、右パネルでZIPのアップロード・タブを選択します。ZIPのアップロードにはスイッチとなるパラメータがいくつかあり、それについては後述します。+ ZIPのアップロード・ボタンで、共通ファイル選択ダイアログを開きます。ZIPファイルを選択すると、すぐサーバーにアップロードされて、結果のメッセージが表示されます。結果メッセージの各行に、アップロードされたファイルごとの説明が示されます。選択したオプションに応じて、ファイルがスキップ、更新、作成または削除されます。

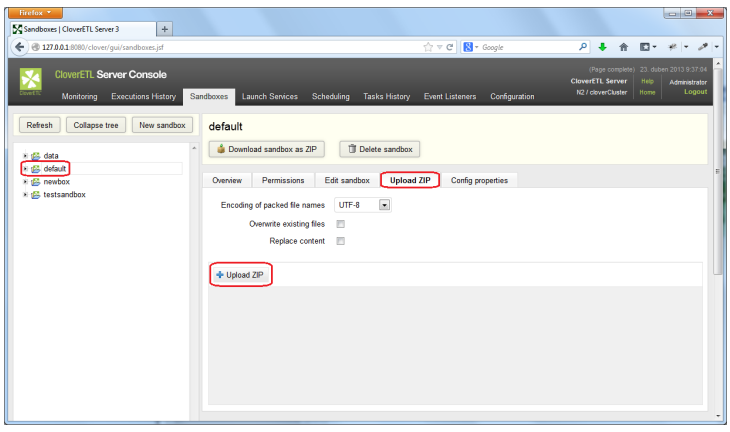


図3.7 Web GUI - サンドボックスへのZIPのアップロード

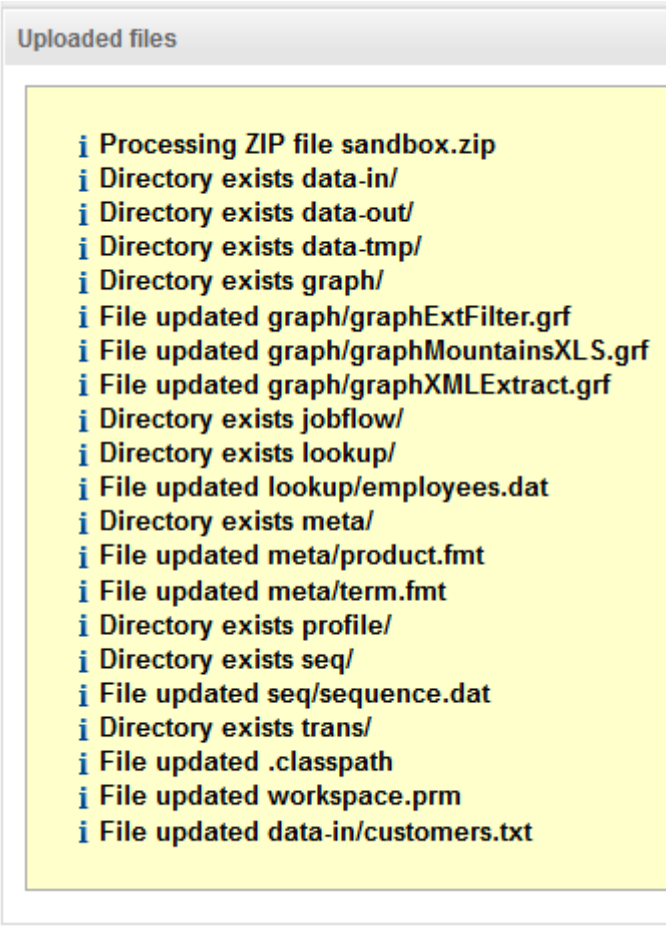


図3.8 Web GUI - ZIPのアップロードの結果

表3.3 ZIPアップロード・パラメータ

ラベル	説明
圧縮ファイル名のエンコード	特殊文字(非ASCII文字)を含むファイル名がエンコードされます。この選択ボックスでファイル名が適切にデコードされるように正しいエンコードを選択します。
既存ファイルの上書き	このスイッチを選択すると、サンドボックスの同じパスに同じファイル名で格納しようとした場合に、既存のファイルが新規ファイルで上書きされます。
内容の置換	このオプションを選択すると、アップロードしたZIPファイルに含まれず、アップロード先のサンドボックスに存在するファイルがすべて削除されます。このオプションを使用するとデータが失われる可能性があるので、これを有効にするには、特別な権限アップロードしたZIPに含まれないファイルを削除できるが必要です。

ZIPでのファイルのダウンロード

左パネルでファイルを選択すると、Web GUI右側のツールバーに、ZIPとしてのファイルのダウンロード・ボタンが表示されます。

作成されるZIPには、選択したファイルのみが含まれます。この機能は、Web GUIで直接表示できない大きいファイル(入力ファイルや出力ファイルなど)で便利です。ユーザーはそれをダウンロードできます。

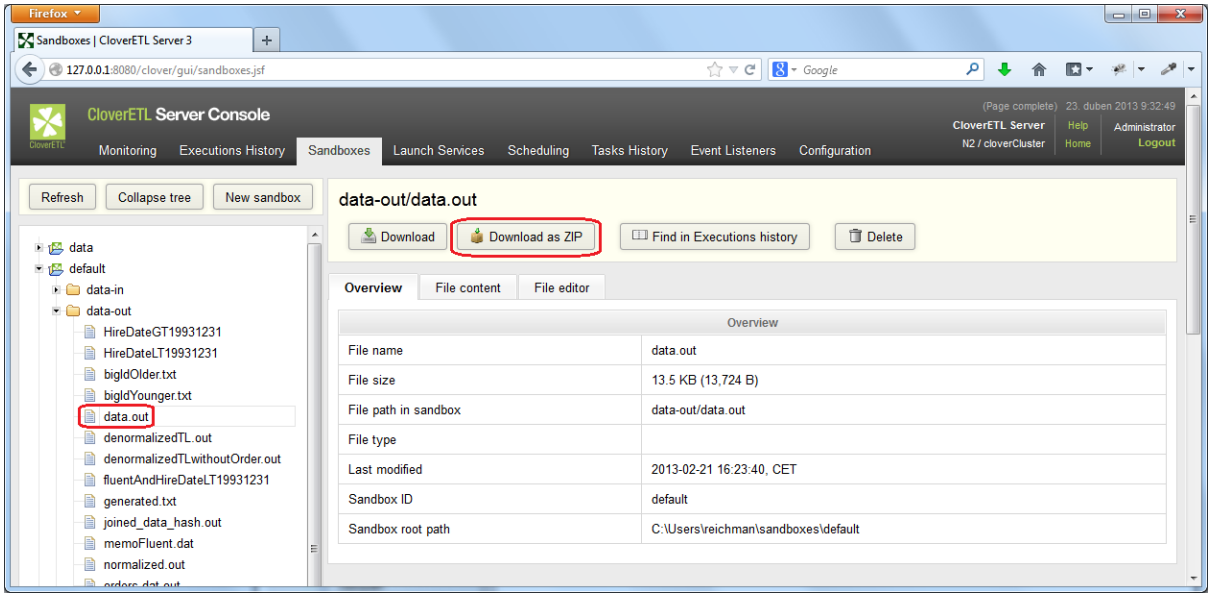


図3.9 Web GUI - ZIPとしてのファイルのダウンロード

HTTP APIファイルのダウンロード

単純なHTTP GETのリクエストによってダウンロード・サーブレットにアクセスするサンドボックス・ファイルをダウンロードまたは表示できます。

```
http://[host]:[port]/[Clover Context]/downloadFile?[Parameters]
```

サーバーにはHTTP基本認証が必要です。Linuxコマンドラインの HTTPクライアント"wget"の場合は、次のようになります。

```
wget --user=clover --password=clover
http://localhost:8080/clover/downloadFile?sandbox=default\&file=data-out/data.dat
```

アンパサンド文字は円記号でエスケープします。そうしないと、コマンドライン・システムでプロセスのフォークを示す演算子として解釈されます。

URLのパラメータ

- sandbox - サンドボックス・コード。必須パラメータです。
- file - サンドボックス・ルートからのファイルの相対パス。必須パラメータです。
- zip - "true"に設定すると、ファイルがZIPとして返され、レスポンスのコンテンツ・タイプは"application/x-zip-compressed"になります。デフォルトではfalseで、レスポンスのコンテンツ・タイプはfileです。

ジョブ構成プロパティ

ETLグラフまたはジョブフローごとに構成プロパティを設定でき、それが実行中に適用されます。プロパティは、Web GUIのサンドボックスセクションで編集可能です。ジョブ・ファイルを選択し、構成プロパティ・タブに移動します。

各サンドボックスに対して同じ構成プロパティを編集することもできます。サンドボックスに指定した値はサンドボックスの各ジョブにも適用されますが、ジョブに対して指定した構成プロパティより優先度は低くなります。

サンドボックスにもジョブにも構成プロパティが指定されない場合は、メインのサーバー構成のデフォルトが適用されます。ジョブ構成プロパティに関連するグローバルな構成プロパティには、"execution"という接頭辞が付きます。たとえば、サーバー・プロパティ"executor.classpath"は、ジョブ構成プロパティ"classpath"のデフォルトです。(詳細は、18章「構成」を参照してください)

また、追加のジョブ・パラメータを指定でき、これはジョブのXMLでプレースホルダとして使用されます。このプレースホルダは、XMLファイルのダウンロードおよび解析中に解決されるので、このようなジョブを保存することはできません。

表3.4 ジョブ構成パラメータ

プロパティ名	デフォルト値	説明
tracking_interval	2000	変換の実行時にノード・ステータスをサンプリングする間隔(ミリ秒単位)。
max_running_concurrently	unlimited	この変換の同時実行インスタンス数。
enqueue_executions	false	ブール値。trueの場合、max_running_concurrentlyを超える実行はエンキューされ、falseの場合、max_running_concurrently超える実行は失敗します。
log_level	INFO	このグラフ実行のLog4jログ・レベル(ALL | TRACE | DEBUG | INFO | WARN | ERROR | FATAL)。低レベル(ALL、TRACE、DEBUG)については、ルート・ログ出力レベルも低レベルに設定する必要があります。ルート・ログ出力のログ・レベルはデフォルトでINFOなので、ジョブ構成パラメータ"log_level"が適切に設定されている場合、変換の実行ログにINFOを超える詳細メッセージは含まれません。log4j構成の詳細は、21章「ロギング」を参照してください。
max_graph_instance_age	0	サーバーのキャッシュで、変換のインスタンスが持続する時間を表す間隔(ミリ秒単位)。0の場合、実行するたびに変換が初期化され解放されます。変換は、場合によってはプールに格納して再利用できません(変換は、動的に指定されたパラメータを使用してプレースホルダを使用します)
classpath		ジョブ・ファイルで使用される外部クラス(変換、ジェネレータ、JMSプロセッサ)を含むパスまたはjarファイルのリスト。指定されたすべてのリソースは、変換ジョブのランタイム・クラスパスに追加されます。すべてのClover Engineライブラリおよびアプリケーション・サーバーのクラスパス上のライブラリは自動的にクラスパスに指定されます。セパレータは、エンジンのプロパティ"DEFAULT_PATH_SEPARATOR_REGEX"で指定します。ディレクトリ・パスは常にスラッシュ文字"/"で終了する必要があります。そうしないと、ClassLoaderはそのディレクトリを認識しません。サーバーは常にジョブのサンドボックスの"trans"サブディレクトリを追加するので、明示的に追加する必要はありません。

プロパティ名	デフォルト値	説明
compile_classpath		ジョブ・ファイルで使用される外部クラス(変換、ジェネレータ、JMSプロセッサ)を含むパスまたはjarファイルのリストおよびそれらのコンパイル用の関連ライブラリ。アプリケーション・サーバーのクラスパス上のライブラリは自動的には含められません。セパレータは、エンジンのプロパティ"DEFAULT_PATH_SEPARATOR_REGEX"で指定します。ディレクトリ・パスは常にスラッシュ文字"/"で終了する必要があります。そうしないと、ClassLoaderはそのディレクトリを認識しません。サーバーは常に"SANDBOX_ROOT/trans/"ディレクトリおよびすべてのJARを自動的に"SANDBOX_ROOT/lib/"ディレクトリに追加するため、これらを明示的に追加する必要はありません。
skip_check_config	デフォルト値はエンジン・プロパティから取得	変換を実行する前に構成チェックを実行するかどうかを指定するスイッチです。
password		エンコードされたDB接続パスワードのデコードに使用するパスワード。
verbose_mode	true	TRUEの場合、ジョブ実行の詳細なログが生成されます。
use_jmx	true	TRUEの場合、ジョブ・エグゼキュータは実行中の変換のJMX mBeanを登録します。
debug_mode	false	TRUEの場合、デバッグを有効にしたエッジがデバッグ・ディレクトリのファイルにデータを格納します。プロパティ"graph.debug_path"を参照してください。 明示的に設定しない場合、サーバー統合に伴うDesignerからグラフを実行するとdebug_modeがTRUEに設定されます。一方、サーバー・コンソールからグラフを実行するとdebug_modeはfalseに設定されます。
executor.use_local_context_url	TRUEの場合、実行中のジョブのコンテキストURLはローカルの"file:" URLになります。そうでない場合は、"sandbox:" URLが使用されます。	false
executor.jobflow_token_tracking	FALSEの場合、ジョブフロー実行時におけるトークンのトラッキングが無効になります。	true

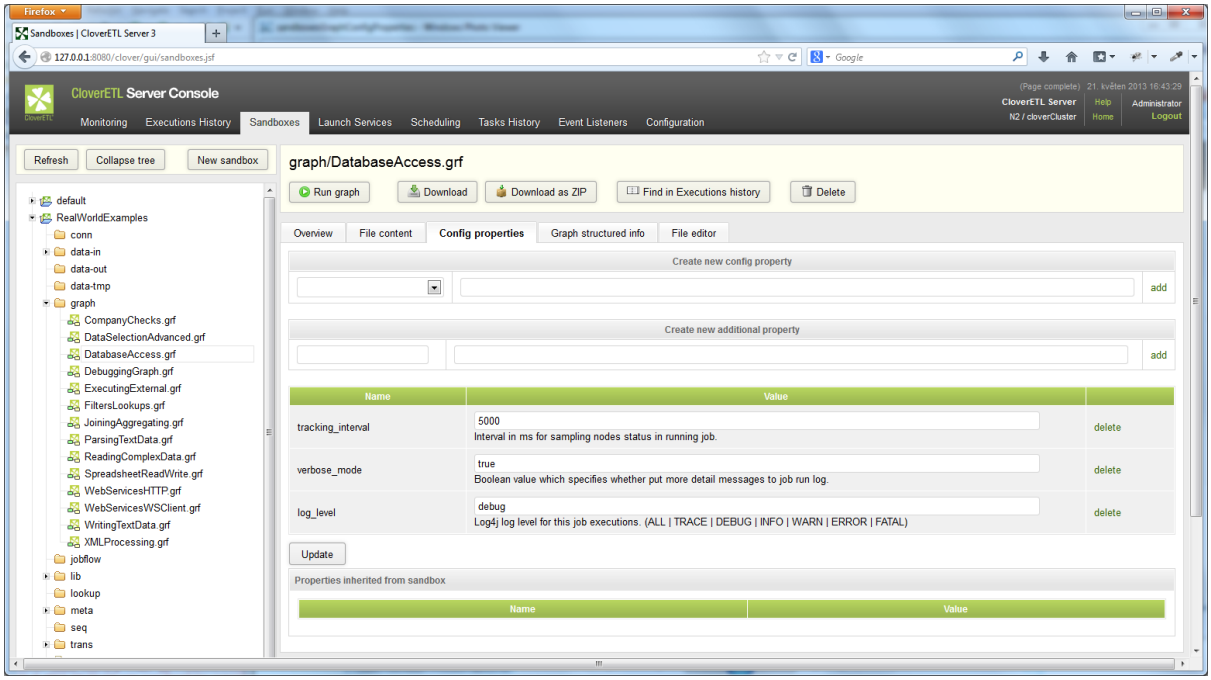


図3.10 ジョブ構成プロパティ

第4章 ジョブ実行の表示 - 実行履歴

実行履歴には、サーバーが実行したすべてのジョブ(変換グラフ、ジョブフローおよびデータ・プロファイラ・ジョブ)の履歴が表示されます。これを使用して、ジョブが失敗した理由を調べることができます。特定の実行に使用されたパラメータなどを参照してください。

表には、実行ID、ジョブ・ファイル、現在のステータス、実行時刻、その他の便利なリンクなど、ジョブに関する基本的な情報が表示されます。リスト内のジョブ名をクリックすると、追加の詳細(関連ログ・ファイル、パラメータ値、トラッキングなど)も表示されます。

一部のジョブは実行履歴リストに表示されないことがあります。これらは、パフォーマンスを向上させるために持続性が無効になっています。たとえば、一部の起動サービスは、サービスのレスポンス時間を向上させるために実行情報の格納を無効にします。

フィルタリングと並替え

フィルタパネルを使用してビューをフィルタ処理します。デフォルトでは、親タスクのみ表示されます(実行の子の表示) - たとえば、クラスタのマスター・ノードとそれらのワーカーはデフォルトで非表示になります。

表ヘッダーの上下の矢印を使用して、リストをソートします。デフォルトでは、最新のジョブが最初にリストされます。

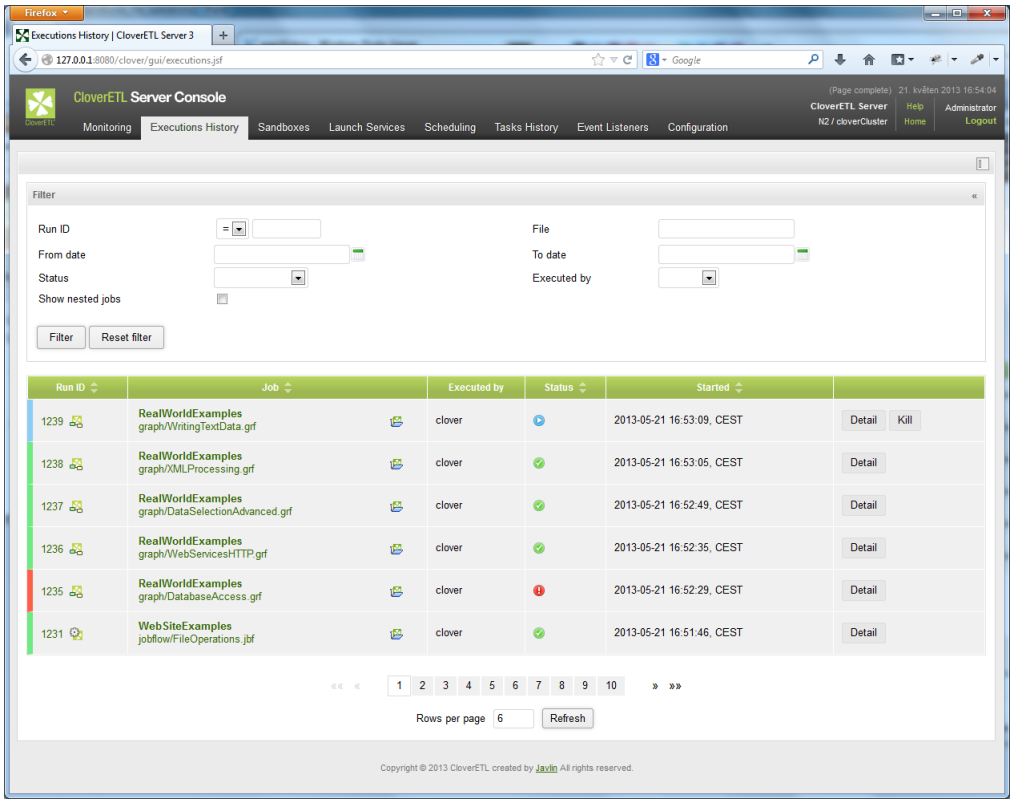


図4.1 実行履歴 - 実行処理表

表でジョブの実行を選択すると、右側に詳細情報が表示されます。

表4.1 永続実行レコードの属性

属性	説明
実行ID	ジョブの実行を識別する一意の番号。サーバーAPIは、通常はこの番号を実行リクエストの単純なレスポンスとして返します。これは、ジョブ実行の仕様の後続のコールのパラメータとして役立ちます。

属性	説明
実行タイプ	サーバーによって認識されるジョブのタイプ。ETLグラフの場合はSTANDALONE、ジョブフローの場合はJOBFLOW、プロファイラの場合はPROFILER_JOB、クラスタ内のパーティション化された実行のメイン・レコードの場合はMASTER、クラスタ内のパーティション化された実行のワーカー・レコードの場合はPARTITIONED_WORKER
親実行ID	親ジョブの実行ID。通常は、このジョブを実行したジョブフローまたはこのワーカー実行をカプセル化するマスター実行です。
ルート実行ID	ルート親ジョブの実行ID。別の親ジョブによって実行されなかったジョブ実行です。
実行グループ	ジョブフロー・コンポーネントは、この属性を使用してサブジョブをグループ化できます。詳細は、ジョブフロー・コンポーネントの説明を参照してください。
ネストされたジョブ	このジョブ実行に子実行があるかどうかを示します。
ノード	クラスタ・モードでは、この実行が実行されたクラスタ・ノードのIDを示します。
実行者	ジョブを実行したユーザー。一部のAPI/GUIを使用して直接またはスケジュールかイベント・リスナーを使用して間接的に実行されます。
サンドボックス	ジョブ・ファイルを含むサンドボックス。実行リクエストとともに送信されるためにサーバー・サイトにジョブ・ファイルが存在しないジョブの場合は、デフォルト・サンドボックスに設定されます。
ジョブ・ファイル	サンドボックスのルートの相対ジョブ・ファイル・パス。実行リクエストとともに送信されるためサーバー・サイトにジョブ・ファイルが存在しないジョブの場合は、生成された文字列に設定されます。
ジョブ・バージョン	ジョブ・ファイルのリビジョン。CloverETL Designerによって生成され、ジョブ・ファイルに格納される文字列です。
ステータス	ジョブ実行のステータス。READY - 実行の開始を待機しています。RUNNING - ジョブを処理中です。FINISHED OK - ジョブがエラーなしで終了しました。ABORTED - 一部のAPI/GUIを使用して直接または親ジョブフローによってジョブが中止されました。ERROR - ジョブに失敗しました。N/A (使用不可) - サーバー・プロセスが突然停止し、ジョブを正常に中止できなかったため、再起動後に不明なステータスのジョブがN/Aとして設定されます
起動済	実行開始のサーバー日時(およびタイムゾーン)。
終了	実行終了のサーバー日時(およびタイムゾーン)。
期間	実行期間
コンポーネントIDのエラー	コンポーネントのエラーによりジョブが失敗した場合、このフィールドにはコンポーネントのIDが入ります。
コンポーネント・タイプのエラー	コンポーネントのエラーによりジョブが失敗した場合、このフィールドにはコンポーネントのタイプが入ります。
エラー・メッセージ	ジョブに失敗した場合、このフィールドにはエラーの説明が入ります。
例外	ジョブに失敗した場合、このフィールドにはエラー・スタック・トレースが入ります。
入力パラメータ	ジョブに渡される入力パラメータのリスト。パラメータはジョブ・ファイルからのロード中に適用されるため、ジョブ・ファイルはキャッシュできません。デフォルトではジョブ・ファイルはキャッシュされません。
入力ディクショナリ	ジョブに渡されるディクショナリ要素のリスト。ディクショナリは、ジョブ・ファイル・キャッシングで個別に使用されます。
出力ディクショナリ	ジョブが終了する瞬間のディクショナリ要素のリスト。

子の実行があるジョブ、たとえばパーティション化されたジョブやジョブフローの場合、実行階層も表示されます。

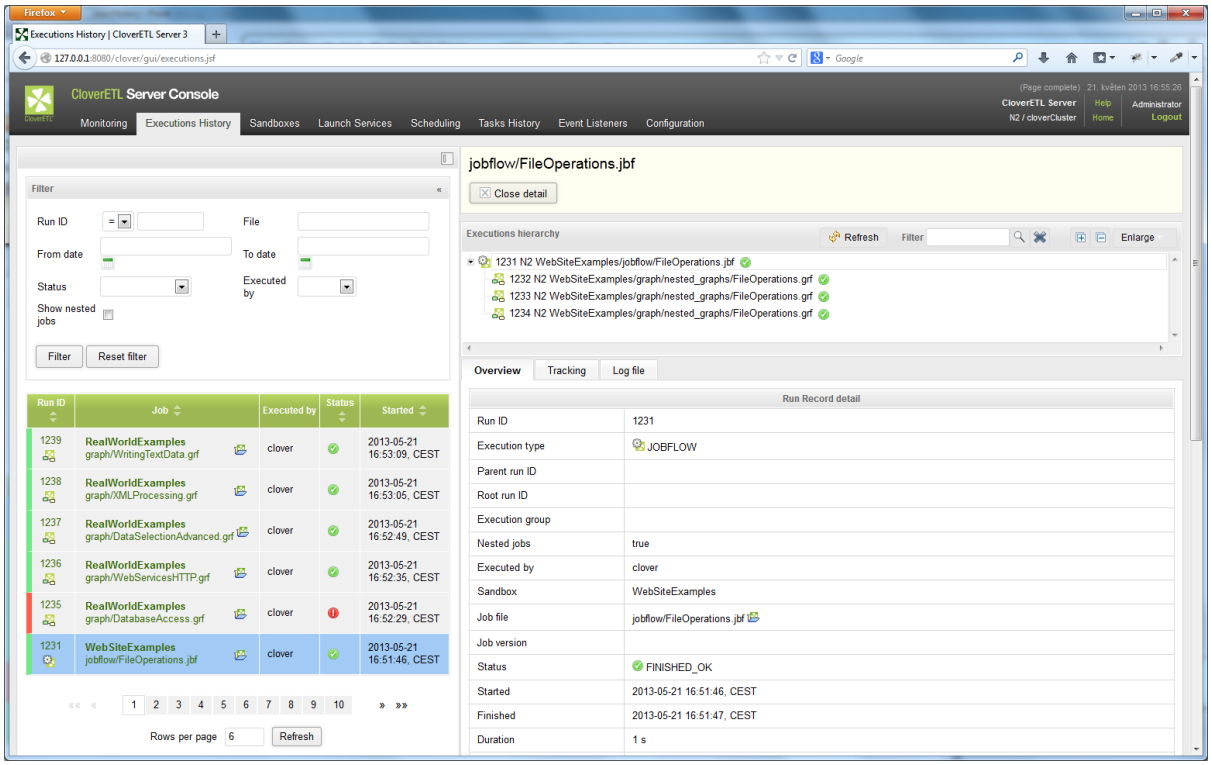


図4.2 実行履歴 – 全体の概要

詳細パネルとジョブのログは大きくなる場合があるので、詳細パネルを広げられるように、左側の表を非表示にすると便利です。リスト・パネルの上部で最小化アイコンをクリックすると、パネルが非表示になります。リスト・パネルを再表示するには、左にあるパネルの展開アイコンをクリックします。

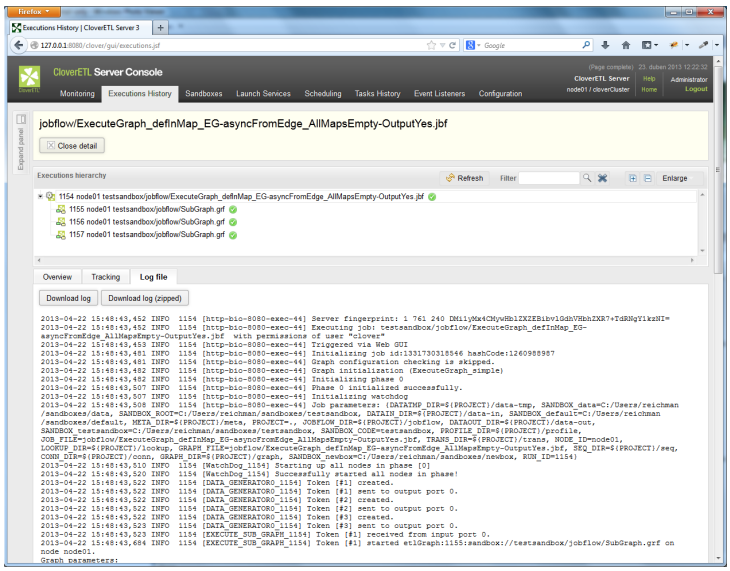


図4.3 実行階層とドッキング表示されたジョブ・リスト

実行階層は多少複雑になる場合があるので、全文フィルタでツリーの内容をフィルタできます。ただし、フィルタを使用すると、選択した実行は階層構造ではなくなります。

第5章 ユーザーとグループ

CloverETL Serverにはセキュリティ・モジュールが組み込まれており、これでユーザーとグループが管理されます。ユーザー・グループは、サンドボックスへのアクセス権限およびユーザーがサーバー上で実行できる、サーバーAPI関数への認証されたコールなどの操作を制御します。単一のユーザーが複数のグループに属することができます。

LDAPまたはActive Directoryをサーバーに構成して、グローバル・ディレクトリからユーザーを認証し、有効グループ(および権限)を割り当てることができます。

ユーザーとユーザー・グループは、構成/ユーザーとグループで管理できます。これには、ユーザーのリスト権限が必要です。

セキュリティ・モジュールはグローバルにオフにできます(18章「構成」を参照)。

LDAP認証

3.2から、ユーザー認証にLDAPサーバーを使用するようにCloverETL Serverを構成できるようになりました。そのため、任意のCloverETL Serverインタフェース(APIまたはGUI)への認証に、LDAPに登録されているユーザーの資格証明を使用できます。

ただし、認可(サンドボックスの内容に対するアクセス・レベルと操作の権限)は引き続きCloverのセキュリティ・モジュールによって処理されます。各ユーザーは、LDAP認証を使用してログインした場合でも、CloverETLセキュリティ・モジュールに独自のユーザー・レコード(および関連グループ)が必要です。したがって、同じユーザー名とドメインのユーザーを"LDAP"に設定する必要があります。そのようなユーザー・レコードが存在しない場合は、CloverETLの構成に従って自動的に作成されます。

LDAPユーザーを認証するためのCloverETLの動作

1. Web GUIのログイン・フォームで、LDAPの資格証明を指定します。
2. CloverETL ServerがLDAPに接続し、ユーザーが存在するかどうかを確認します(指定した検索を使用してLDAPをルックアップします)。
3. LDAPにユーザーが存在する場合、CloverETL Serverは認証を実行します。
4. 成功すると、CloverETL ServerはLDAPユーザーのグループを検索します。
5. CloverETL Serverは、Cloverにログインできる LDAPグループにそのユーザーが割り当てられているかどうかを確認します。
6. Cloverのユーザー・レコードは、現在のLDAP値に従って作成/更新されます。
7. Cloverユーザーは、LDAPグループへの現在の割当てに従ってCloverグループに割り当てられます。
8. ユーザーがログインします

注記

ドメインの切替え:

- ユーザーをLDAPで作成してからcloverドメインに切り替えた場合は、パスワードの変更タブでそのユーザーのパスワードを設定します。
- ユーザーをcloverとして作成してから、LDAPドメインに切り替えた場合は、cloverドメインのパスワードがありますが、LDAPのパスワードでオーバーライドされま

す。cloverドメインに戻ると、元のパスワードが再利用されます。必要な場合(パスワードを忘れた場合など)は、パスワードの変更タブでリセットできます。

構成

デフォルトでは、CloverETL Serverで認証に使用できるのは、独自の内部メカニズムのみです。LDAPでの認証を有効にするには、構成プロパティ"security.authentication.allowed_domains"を適切に設定します。これは、認証に使用されるユーザー・ドメインのリストです。

現在、"LDAP"と"clover"の2つの認証メカニズムが実装されています("clover"はCloverETLの内部認証の識別子で、security.default_domainプロパティによって変更できますが、ホワイトレーベル化の目的に限られます)。LDAP認証を有効にするには、値を"LDAP" (LDAPのみ)または"clover,LDAP"に設定します。どちらのドメインのユーザーもログインできます。LDAPが適切に構成されるまでは、両方のメカニズムを同時に許可しておくことをお薦めします。管理者ユーザーは、LDAP接続が適切に構成されていない場合でもWeb GUIに引き続きログインできます。

LDAP接続の基本的なプロパティ

```
# Implementation of context factory
security.ldap.ctx_factory=com.sun.jndi.ldap.LdapCtxFactory
# timeout for all queries sent to LDAP server
security.ldap.timeout=5000
# limit for number of records returned from LDAP
security.ldap.records_limit=50

# URL of LDAP server
security.ldap.url=ldap://hostname:port
# Some generic UserDN which allows lookup for the user and groups.
security.ldap.userDN=
# Password for the user specified above security.ldap.password=
```

ユーザー・ルックアップの構成

指定した値は、次の特定のLDAPツリーで有効です。

- dc=company,dc=com
  - ou=groups
    - cn=admins
      (objectClass=groupOfNames,member=(uid=smith,dc=company,dc=com),member=(uid=jones,dc=company,dc=com)
    - cn=developers (objectClass=groupOfNames,member=(uid=smith,dc=company,dc=com))
    - cn=consultants (objectClass=groupOfNames,member=(uid=jones,dc=company,dc=com))
  - ou=people
    - uid=smith (fn=John,sn=Smith,mail=smith@company.com)
    - uid=jones (fn=Bob,sn=Jones,mail=jones@company.com)

LDAPユーザーをそのユーザー名でルックアップする場合は、次のプロパティが必須です。(前述のログイン・プロセスのステップ[2]を参照)

```
# Base specifies the node of LDAP tree where the search starts
security.ldap.user_search.base=dc=company,dc=eu
# Filter expression for searching the user by his username.
# Please note, that this search query must return just one record.
# Placeholder ${username} will be replaced by username specified by the logging user.
security.ldap.user_search.filter=(uid=${username})
# Scope specifies type of search in "base".
There are three possible values: SUBTREE | ONELEVEL | OBJECT
# http://download.oracle.com/javase/6/docs/api/javax/naming/directory/SearchControls.html
 security.ldap.user_search.scope=SUBTREE
```

次のプロパティは、定義した検索の属性の名前です。Cloverセキュリティ・モジュールによるユーザー・レコードの作成/更新が必要になった場合に、LDAPユーザーに関する基本的な情報を取得するために使用されます(前述のログイン・プロセスのステップ[6]を参照)。

```
security.ldap.user_search.attribute.firstname=fn
security.ldap.user_search.attribute.lastname=sn
security.ldap.user_search.attribute.email=mail
# This property is related to the following step "searching for groups".
# Groups may be obtained from specified user's attribute, or found by filter (see next
 paragraph)
# Please leave this property empty if the user doesn't have such attribute.
security.ldap.user_search.attribute.groups=memberOf
```

次のステップでは、ユーザーが割り当てられているグループをcloverが検索しようとします。(前述のログイン・プロセスのステップ[4]を参照)。ユーザーが割り当てられているグループのリストを取得するには、2つの方法があります。ユーザーとグループの関係をユーザー側で指定します。ユーザー・レコードは、グループのリストに関する属性を持ちます。通常は"memberOf"属性です。または、ユーザーとグループの関係をグループ側で指定します。グループ・レコードは、割り当てられているユーザーのリストに関する属性を持ちます。通常は"member"属性です。

関係をユーザー側で指定する場合は、適切なプロパティを指定してください。

```
security.ldap.user_search.attribute.groups=memberOf
```

そうでない場合は、空のままにします。

関係をグループ側で指定する場合は、検索のプロパティを指定してください。

```
security.ldap.groups_search.base=dc=company,dc=com
# Placeholder ${userDN} will be replaced by user DN found by the search above
# If the filter is empty, searching will be skipped.
security.ldap.groups_search.filter=(&(objectClass=groupOfNames)(member=${userDN}))
security.ldap.groups_search.scope=SUBTREE
```

そうでない場合は、security.ldap.groups_search.filterプロパティを空のままにして、検索をスキップするようにします。

Cloverユーザー・レコードは、検索(または属性)で見つかったLDAPグループに従ってcloverグループに割り当てられます。(グループ・チェックは、ログインごとに実行されます)

```
# Value of the following attribute will be used for lookup for the Clover group by its
 code.
# So the user will be assigned to the Clover group with the same "code"
security.ldap.groups_search.attribute.group_code=cn
```

Cloverにログインできるグループを指定することもできます。(前述のログイン・プロセスのステップ[5]を参照)

```
# Semicolon separated list of LDAP group DNs (distinguished names).
# LDAP user must be assigned to one or more of these groups,
otherwise new clover user can't be created.
# Special value "_ANY_" disables this check and basically any LDAP user may login.
# If the LDAP group DNs are configured, also security.ldap.groups_search.*
properties must be configured.
# value could be e.g. "cn=developers,dc=company,dc=com;cn=admins,dc=company,dc=com"
security.ldap.allowed_ldap_groups=_ANY_
```

Web GUIのユーザーセクション

これは、ユーザー管理のためのセクションです。このセクションの機能はユーザーの権限に依存します。したがって、ユーザーによっては、このセクションを表示できても何も変更できません。あるいは、変更は可能でも新規のユーザーは作成できません。

ユーザーセクションで使用できる機能

- 新規ユーザーの作成
- 基本データの変更
- パスワードの変更
- ユーザーの有効化/無効化
- グループへのユーザーの割当て - グループに割り当てると、ユーザーに適切な権限が付与されます。

表5.1 前述の空のDBのデフォルト・インストール後に作成される2つのユーザー

ユーザー名	説明
clover	cloverユーザーは管理権限を持ちます。したがって、デフォルトのパスワード"clover"は、インストール後に必ず変更してください。
system	systemユーザーは、他のユーザーを使用できない場合、たとえばセキュリティがグローバルにオフになっている場合に、共通ユーザーではなくアプリケーションによって使用されます。このユーザーは削除できず、このユーザーでログインすることもできません。

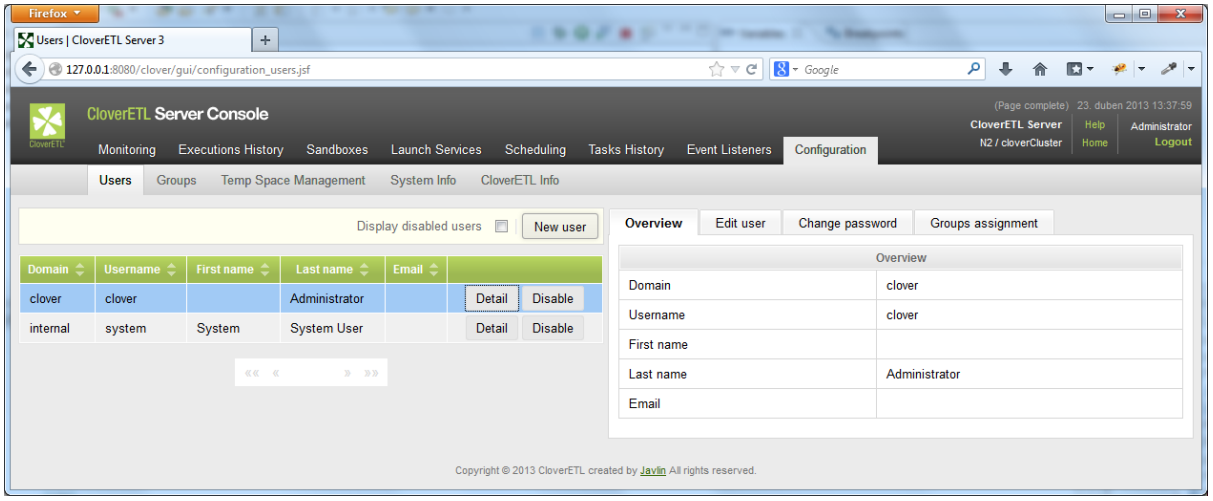


図5.1 Web GUI -構成のユーザーセクション

表5.2 ユーザーの属性

属性	説明
ドメイン	ユーザーの発生元であるドメイン。現在使用可能な値は"clover"または"ldap"の2つです。
ユーザー名	共通ユーザーの識別子。一意である必要があり、空白や特殊文字は使用できません。英数字のみです。
パスワード	大/小文字が区別されるパスワード。ユーザーがパスワードを忘れた場合は、新しいパスワードを設定する必要があります。セキュリティ上の理由でパスワードは暗号化して格納されます。したがって、データベースから取り出すことはできず、その操作を行う適切な権限のあるユーザーが変更する必要があります。
名	
姓	
電子メール	CloverETL管理者またはCloverETLサーバーが自動通知に使用できる電子メール。詳細は、「タスク - 電子メールの送信」を参照してください。

ユーザー・レコードの編集

ユーザーの作成権限またはユーザーの編集権限のあるユーザーは、このフォームを使用して基本的なユーザー・パラメータを設定できます。

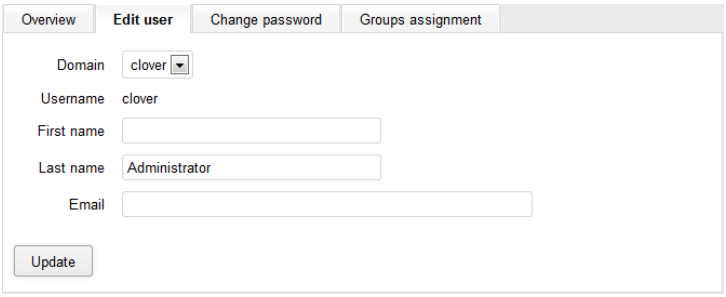


図5.2 Web GUI - ユーザーの編集

ユーザー・パスワードの変更

ユーザーがパスワードを忘れた場合は、新しいパスワードを設定する必要があります。パスワードの変更権限のあるユーザーは、このフォームを使用してパスワードを変更できます。

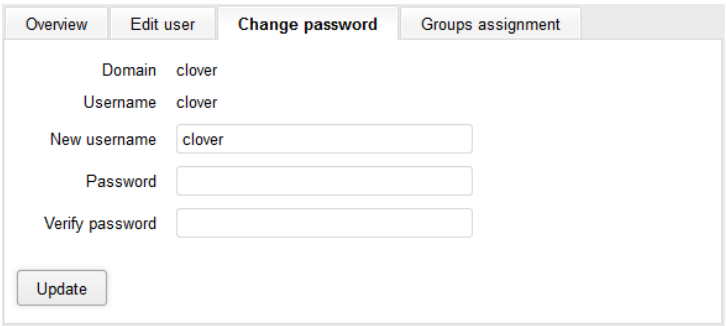


図5.3 Web GUI - パスワードの変更

グループの割当て

グループに割り当てると、ユーザーに適切な権限が付与されます。このフォームを使用して、ユーザーを割り当てるグループを指定できるのは、グループの割当て権限でログインしているユーザーのみです。権限の詳細は、「Web GUIのグループセクション」を参照してください。

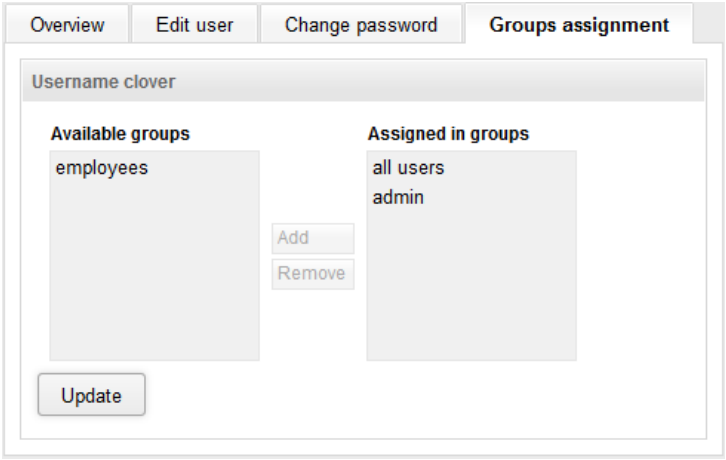


図5.4 Web GUI - グループの割当て

ユーザーの有効化/無効化

ユーザー・レコードは、ログおよび履歴レコードと様々に関係しているため、削除はできません。かわりに、無効化することができます。つまり、基本的にレコードはリストに表示されず、ユーザーはログインもできません。

ただし、無効化したユーザーは再度有効化できます。無効化したユーザーはグループから削除されるので、再度有効化してから必要に応じてグループに割り当てる必要があります。

Web GUIのグループセクション

グループとはユーザーの抽象セットであり、グループに割り当てられたユーザーは一定の権限を付与されます。したがって、個々のユーザーに権限を指定することは必ずしも必要ではありません。

CloverETL Serverには、複数レベルの権限が実装されています。

- サンドボックスでの読取り/書込み/実行権限 - サンドボックスの所有者は、グループごとに異なる権限を指定できます。詳細は、「サンドボックスの内容のセキュリティと権限」を参照してください。

- 一定の操作を実行する権限 - 操作権限権限の割当てを持つユーザーは、特定の権限を既存のグループに割り当てることができます。

- 特定のサービスを起動する権限 - 詳細は、17章「起動サービス」を参照してください。

表5.3 インストール中に作成されるデフォルトのグループ

グループ名	説明
admins	このグループはすべての操作権限、つまり無制限の権限を持ちます。デフォルトのユーザー"clover"はこのグループに割り当てられ、管理者となります。

グループ名	説明
all users	CloverETLの各ユーザーは、デフォルトでこのグループに割り当てられます。このグループからユーザーを削除することは可能ですが、推奨はされません。このグループは、サンドボックスまたは一定の操作の権限など、例外なくすべてのユーザーに付与する権限があるときに便利です。

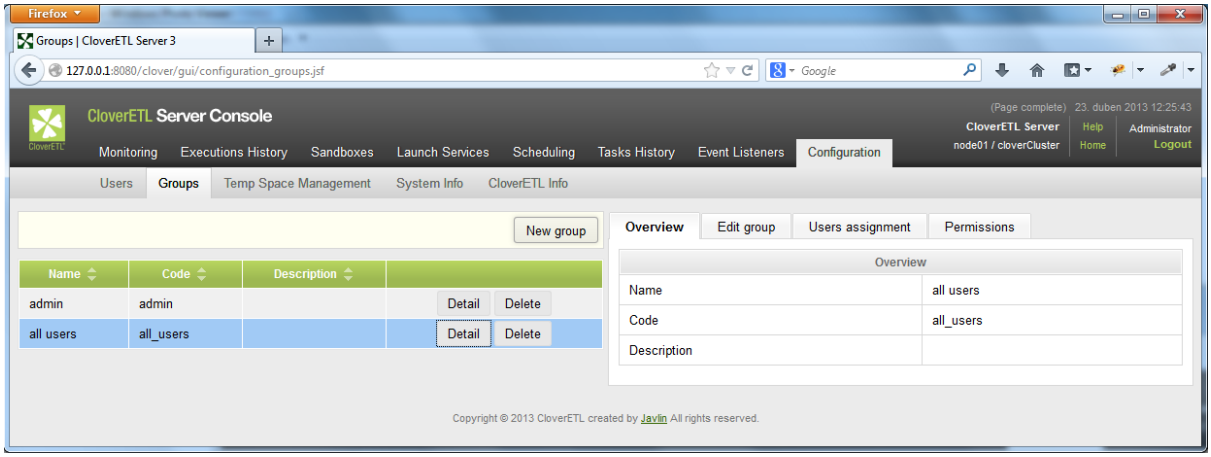


図5.5 Web GUIのグループセクション

ユーザーの割当て

ユーザーとグループの関係はN:Mです。つまり、グループをユーザーに割り当てる場合でも、ユーザーをグループに割り当てる場合でも、方法は変わりません。

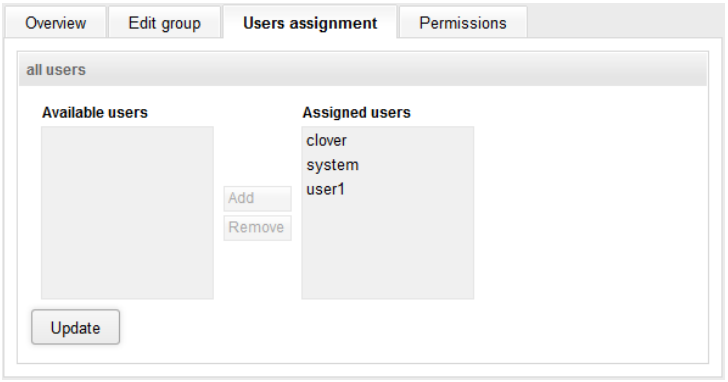


図5.6 Web GUI - ユーザーの割当て

グループ権限

グループ権限はツリーとして構造化され、権限はルートからリーフへと継承されます。つまり、ある権限(ツリー・ノード)が有効(青い丸)である場合、サブツリーの権限もすべて自動的に有効(白い丸)になります。赤い×印の権限は無効です。

"admin"グループにはすべて権限が割り当てられるので、サブツリーの個別の権限は自動的に割り当てられます。

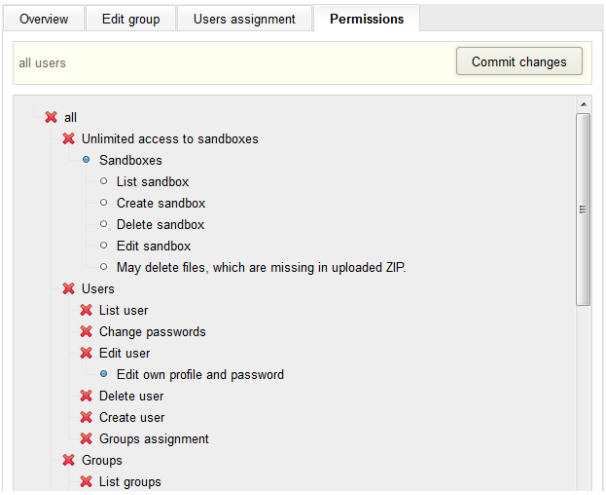

Overview
Edit group
Users assignment
Permissions
all users
Commit changes
all
Unlimited access to sandboxes
Sandboxes
List sandbox
Create sandbox
Delete sandbox
Edit sandbox
May delete files, which are missing in uploaded ZIP.
Users
List user
Change passwords
Edit user
Edit own profile and password
Delete user
Create user
Groups assignment
Groups
List groups

図5.7 権限のツリー

第6章 スケジューリング

スケジューリング・モジュールでは、反復的または適切なタイミングでトリガーする必要のある操作の時間スケジュールを作成できます。

UNIXシステムの"cron"と同様に、各スケジュールは別々の時間スケジュール定義と実行するタスクを表します。

クラスタでは、ノードIDパラメータを使用してスケジュールされたタスクを実行する必要のあるノードを明示的に指定できます。ただし、設定されていない場合、ノードはすべての使用可能なノードから自動的に選択されます(ただし、常に1つのみです)。

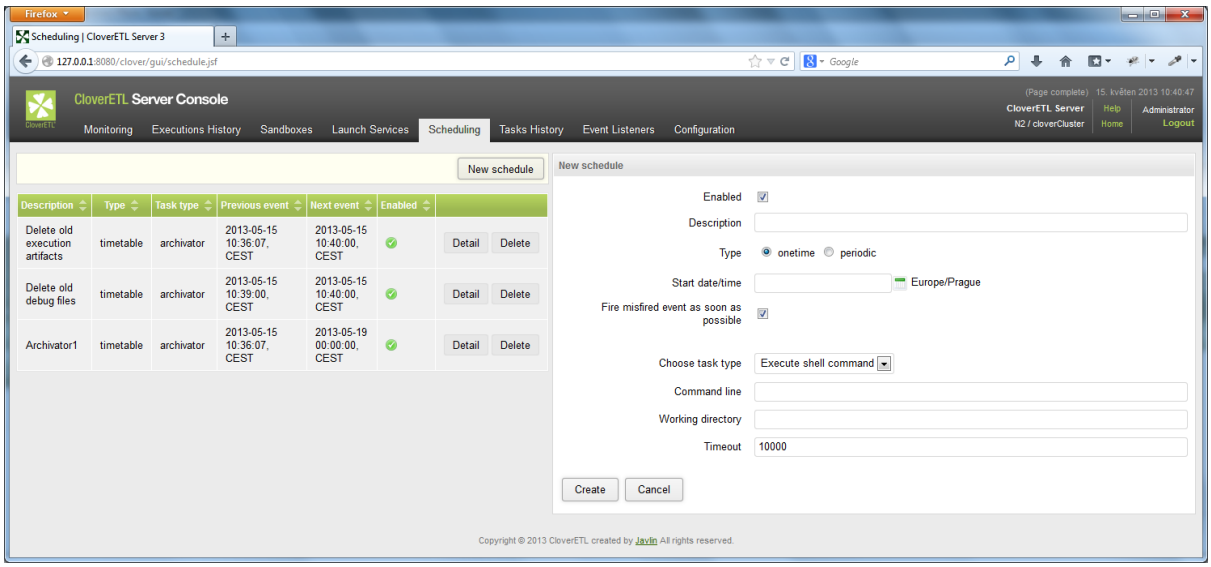


図6.1 Web GUI - スケジューリングセクション - 新規作成

タイムテーブル設定

この項では、スケジュールをいつトリガーする必要があるかを指定する方法について説明します。正確なトリガー時間が保証されるわけではないことに注意してください。数秒の遅れが生じる可能性があります。スケジュール自体は様々な方法で指定できます。

- 「ワンタイム・スケジュール」
- 「間隔別の定期的スケジュール」
- 「タイムテーブル別の定期的スケジュール(cron式)」

ワンタイム・スケジュール

このスケジュールが1回のみトリガーされることは明白です。

表6.1 ワンタイム・スケジュールの属性

タイプ	ワンタイム
開始日付/時間	分単位の精度で指定した日付および時間。
正しく起動していないイベントを即時起動するスイッチ	選択すると、トリガー時間がなんらかの理由により欠落している場合(サーバーの再起動など)、可能であれば、即時トリガーされ

	ます。それ以外の場合は、無視され、次のスケジュール時にトリガーされます。

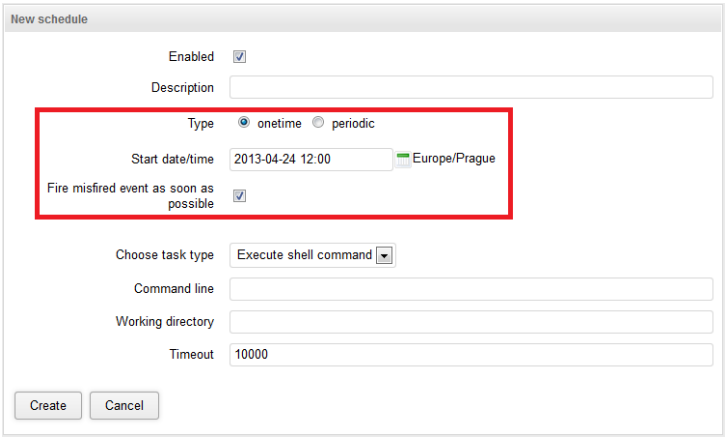


図6.2 Web GUI - ワンタイム・スケジュール・フォーム

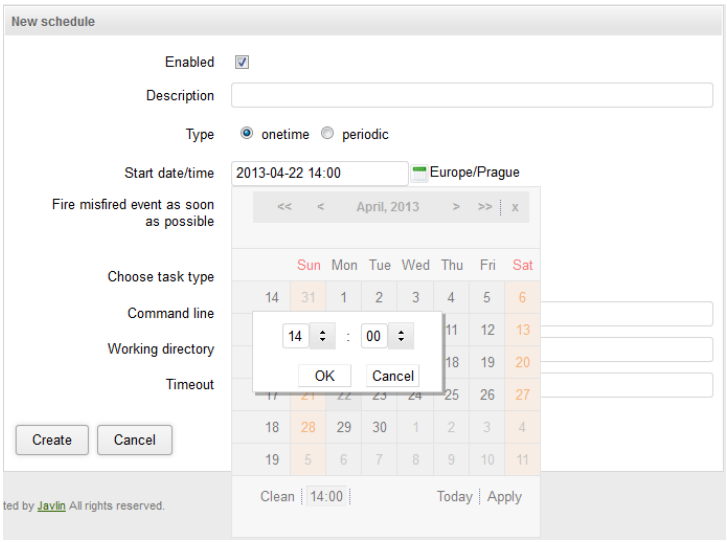


図6.3 Web GUI - スケジュール・フォーム - カレンダ

間隔別の定期的スケジュール

このタイプのスケジュールは、最も簡単な定期的タイプです。トリガー時間は、次の属性によって指定されます。

表6.2 定期的スケジュールの属性

タイプ	定期的
周期性	間隔
次の日時より前はアクティブにならない	分単位の精度で指定した日付および時間。
次の日時より後はアクティブにならない	分単位の精度で指定した日付および時間。
間隔(分)	2つのトリガー時間の間隔を指定します。次のタスクは、前のタスクがまだ実行中でもトリガーされます。

正しく起動していないイベントを即時起動するスイッチ	選択すると、トリガー時間がなんらかの理由により欠落している場合(サーバーの再起動など)、可能であれば、即時トリガーされます。それ以外の場合は、無視され、次のスケジュール時にトリガーされます。

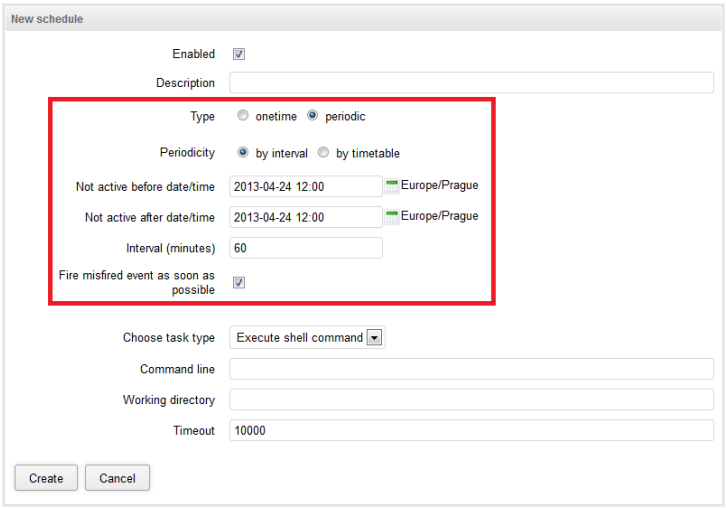


図6.4 Web GUI - 定期的スケジュール・フォーム

タイムテーブル別の定期的スケジュール(cron式)

タイムテーブルは、強力な(ただし少々扱いにくい) cron式によって指定します。

表6.3 cron定期的スケジュールの属性

タイプ	定期的
周期性	間隔
日時より前はアクティブにならない	分単位の精度で指定した日付および時間。
日時より後はアクティブにならない	分単位の精度で指定した日付および時間。
cron式	cronは、スケジューリング用として独自の書式を使用する強力なツールです。この書式は、UNIX管理者の間で広く知れ渡っています(たとえば、0 0/2 4-23 * * ?は、4:00amと11:59pmの間で2分ごとを意味します)。
正しく起動していないイベントを即時起動するスイッチ	選択すると、トリガー時間がなんらかの理由により欠落している場合(サーバーの再起動など)、可能であれば、即時トリガーされます。それ以外の場合は、無視され、次のスケジュール時にトリガーされます。

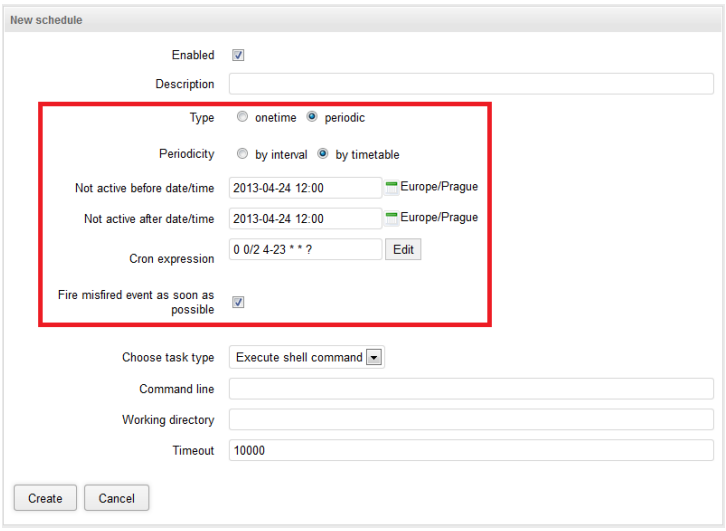


図6.5 cron定期的スケジュール・フォーム

タスク

基本的に、タスクにより、トリガー時に何を行うかを指定します。スケジュール用およびグラフ・イベント・リスナー用として、次のような複数のタスクが実装されています。

- 「タスク - グラフの実行」
- 「タスク - ジョブフローの実行」
- 「タスク - ジョブの中断」
- 「タスク - シェル・コマンドの実行」
- 「タスク - 電子メールの送信」
- 「タスク - Groovyコードの実行」
- 「タスク - アーカイバ」

タスク - グラフの実行

このタスク・タイプの動作は、「タスク - ジョブフローの実行」とほとんど同じです

表6.4 グラフ実行タスクの属性

タスク・タイプ	グラフの開始
サンドボックス	この選択ボックスには、ログ出力ユーザーによる読取りが可能なサンドボックスが含まれます。実行するグラフが含まれるサンドボックスを選択します。
グラフ	この選択ボックスには、選択したサンドボックスでアクセス可能なすべてのグラフ・ファイルが入力されます。
パラメータ	実行対象ジョブにパラメータとして渡されるキーと値のペア。また、このタスクがジョブ(グラフまたはジョブフロー)イベントによってトリガーされる場合、ソース・ジョブ・パラメータを指定できます。これらのパラメータは、ソース・ジョブから実行対象ジョブに渡されます。たとえば、イベント・ソースには、

	次のパラメータがあるとします: 値"val2"を持つparamName2、値"val3"を持つparamName3、値"val5"を持つparamName5。タスクでは、パラメータ属性が次のように設定されています。 `paramName1=paramValue1 paramName2= paramName3 paramName4` このため、実行対象ジョブは、次のパラメータおよび値を取得します: 値"paramValue1"を持つparamName1 (タスク構成で明示的に指定されています)、値""を持つparamName2 (タスク構成で明示的に指定されている空の文字列により、イベント・ソース・パラメータがオーバーライドされます)、値"val3"を持つparamName3 (値はイベント・ソースから取得されます)。次のパラメータは渡されません: paramName4は、イベント・ソース内に値がないため、渡されません。paramName5は、渡すパラメータとしてタスク内に指定されていないため、渡されません。 "EVENT_RUN_RESULT"や"EVENT_RUN_ID"などのイベント・パラメータは、制限なしで実行対象ジョブに渡されます。

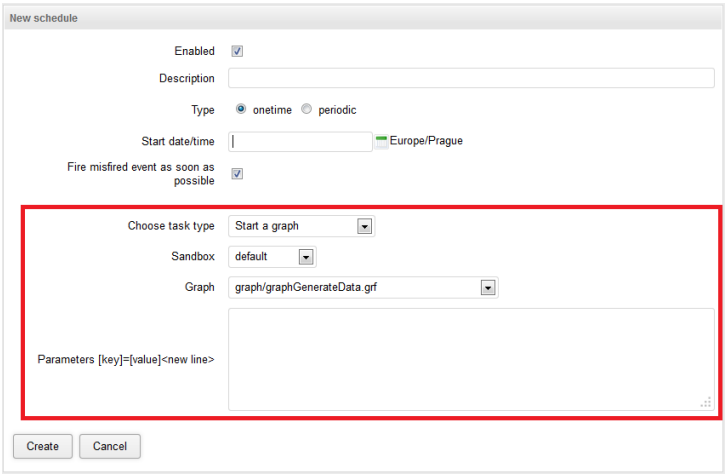


図6.6 Web GUI - グラフ実行タスク

タスク - ジョブフローの実行

このタスク・タイプの動作は、「タスク - グラフの実行」とほとんど同じです

表6.5 ジョブフロー実行タスクの属性

タスク・タイプ	ジョブ・フローの開始
サンドボックス	この選択ボックスには、ログ出力ユーザーによる読取りが可能なサンドボックスが含まれます。実行するジョブフローが含まれるサンドボックスを選択します。
ジョブフロー	この選択ボックスには、選択したサンドボックスでアクセス可能なすべてのジョブフロー・ファイルが入力されます。
パラメータ	実行対象ジョブにパラメータとして渡されるキーと値のペア。また、このタスクがジョブ(グラフまたはジョブフロー)イベントによってトリガーされる場合、ソース・ジョブ・パラメータを指定できます。これらのパラメータは、ソース・ジョブから実行対象ジョブに渡されます。たとえば、イベント・ソースには、次のパラメータがあるとします: 値"val2"を持つparamName2、

	値"val3"を持つparamName3、値"val5"を持つparamName5。タスクでは、パラメータ属性が次のように設定されています。 `paramName1=paramValue1 paramName2= paramName3 paramName4` このため、実行対象ジョブは、次のパラメータおよび値を取得します: 値"paramValue1"を持つparamName1（タスク構成で明示的に指定されています)、値""を持つparamName2（タスク構成で明示的に指定されている空の文字列により、イベント・ソース・パラメータがオーバーライドされます)、値"val3"を持つparamName3(値はイベント・ソースから取得されます)。次のパラメータは渡されません: paramName4は、イベント・ソース内に値がないため、渡されません。paramName5は、渡すパラメータとしてタスク内に指定されていないため、渡されません。 "EVENT_RUN_RESULT"や"EVENT_RUN_ID"などのイベント・パラメータは、制限なしで実行対象ジョブに渡されます。

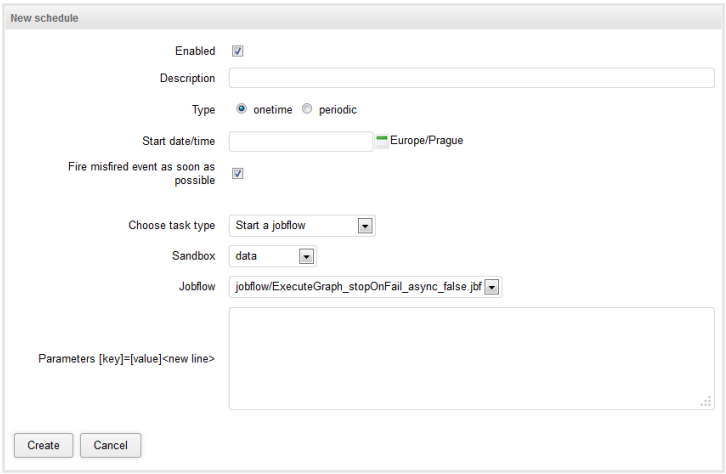


図6.7 Web GUI - ジョブフロー実行タスク

タスク - ジョブの中断

このタスクをアクティブ化すると、指定したジョブ(ETLグラフまたはジョブフロー)が現在実行されている場合、このジョブを強制終了/中断します。

表6.6 ジョブの中断タスクの属性

タスク・タイプ	ジョブの中断
イベントのソースの強制終了	このスイッチがオンである場合、このタスクをアクティブ化した、イベントのソースであるジョブが強制終了されます。属性サンドボックスおよびジョブは無視されます。
サンドボックス	強制終了するジョブが含まれるサンドボックスを選択します。この属性が機能するのは、イベントのソースの強制終了スイッチがオフである場合のみです。
ジョブ	この選択ボックスには、選択したサンドボックスでアクセス可能なすべてのジョブが入力されます。現在実行されている、選択したジョブのすべてのインスタンスが強制終了されます。この属性が機能するのは、イベントのソースの強制終了スイッチがオフである場合のみです。

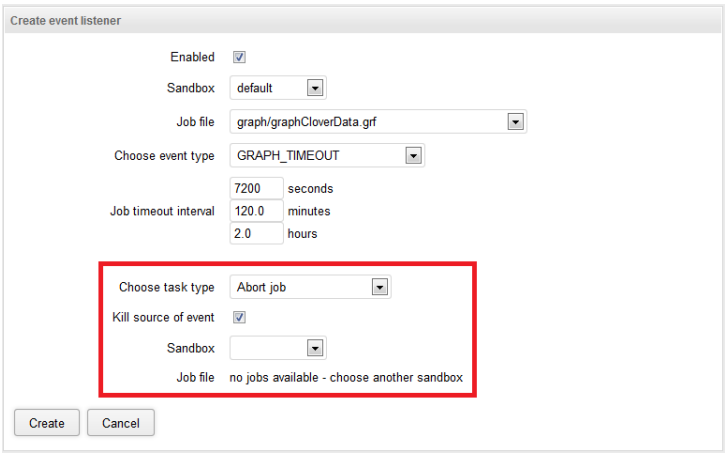


図6.8 Web GUI - "ジョブの中断"

タスク - シェル・コマンドの実行

表6.7 シェル・コマンドの実行タスクの属性

タスク・タイプ	シェル・コマンドの実行
コマンドライン	外部プロセスを実行するためのコマンドライン。
作業ディレクトリ	プロセスの作業ディレクトリ。設定されていない場合は、アプリケーション・サーバー・プロセスの作業ディレクトリが使用されます。
タイムアウト	タイムアウト(ミリ秒)。この数値によって指定した時間が経過した後、外部プロセスが終了し、すべての結果がログに記録されます。

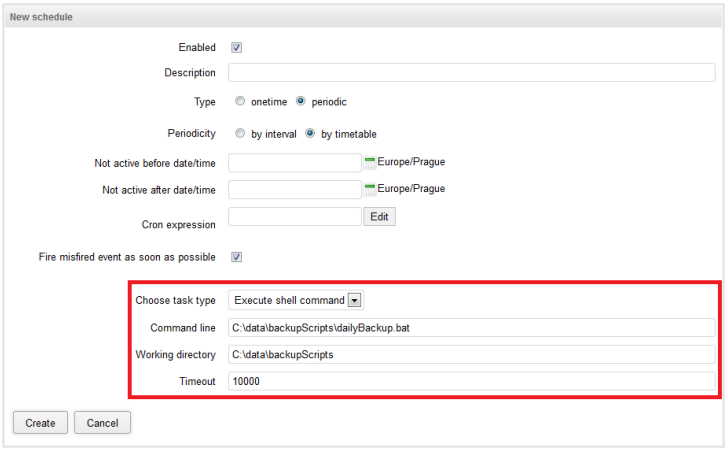


図6.9 Web GUI - シェル・コマンド

タスク - 電子メールの送信

このタスクは非常に便利ですが、現在はグラフ・イベントに対するレスポンスとしてのみ使用されています。これは、監視用として非常に強力な機能です。(このタスク・タイプの説明は、7章「グラフ・イベント・リスナー」を参照してください)。

注意: 電子メールを定期的に送信するのは無意味であるように思われますが、現在のサーバー・ステータスや日次の概要を送信できます。これらの機能は将来のバージョンで実装されます。

タスク - Groovyコードの実行

このタスク・タイプを使用すると、スクリプト言語Groovyで作成されたコードを実行できます。いくつかの変数も使用できます。このタスクの唯一のパラメータは、Groovyで作成されたソース・コードです。

CloverETL ServerにはGroovyバージョン2.0.0が含まれています

表6.8 Groovyコードで使用可能な変数のリスト

変数	クラス	説明	可用性
event	com.cloveretl.server.events.AbstractServerEvent		毎回
task	com.cloveretl.server.persistent.Task		毎回
now	java.util.Date	現在時	毎回
parameter	java.util.Properties	タスクのプロパティ	毎回
user	com.cloveretl.server.persistent.User	event.getUser()と同様	毎回
run	com.cloveretl.server.persistent.RunRecord		イベントがGraphServerEventのインスタンスである場合
tracking	com.cloveretl.server.persistent.TrackingGraph	run.getTrackingGraph()と同様	イベントがGraphServerEventのインスタンスである場合
sandbox	com.cloveretl.server.persistent.Sandbox	run.getSandbox()と同様	イベントがGraphServerEventのインスタンスである場合
schedule	com.cloveretl.server.persistent.Schedule	((ScheduleServerEvent)event).getSchedule()と同様	イベントがScheduleServerEventのインスタンスである場合
servletContext	javax.servlet.ServletContext		毎回
cloverConfiguration	com.cloveretl.server.spring.CloverConfiguration	CloverETL Serverの構成値	毎回
serverFacade	com.cloveretl.server.facade.api.ServerFacade	ファサード・インタフェースの参照。CloverETL Serverコアをコールする際に役立ちます。 WARファイルには、ファサードAPIのJavaDocが含まれます。このファイルは、URL: http://host:port/clover/javadoc/index.htmlからアクセスできます	毎回
sessionToken	String	イベントを所有するユーザーの有効なセッショ	毎回

変数	クラス	説明	可用性
		ン・トークン。ファサード・インタフェースの認可用として役立ちます。	

変数run、trackingおよびsandboxが使用可能なのは、イベントがGraphServerEventクラスのインスタンスである場合のみです。変数scheduleは、イベント変数クラスとしてのScheduleServerEventに対してのみ使用可能です。

Groovyスクリプトの使用例

この例は、終了したグラフを記述するテキスト・ファイルを作成するスクリプトを示しています。これは、run変数の使用を示しています。

```
import com.cloveretl.server.persistent.RunRecord;
String dir = "/tmp/";
RunRecord rr = (RunRecord)run;

String fileName = "report"+rr.getId()+"_finished.txt";

FileWriter fw = new FileWriter(new File(dir+fileName));
fw.write("Run ID       :"+rr.getId()+"\n");
fw.write("Graph ID     :"+rr.getGraphId()+"\n");
fw.write("Sandbox      :"+rr.getSandbox().getName()+"\n");
fw.write("\n");
fw.write("Start time   :"+rr.getStartTime()+"\n");
fw.write("Stop time    :"+rr.getStopTime()+"\n");
fw.write("Duration     :"+rr.getDurationString()+"\n");
fw.write("Final status :"+rr.getFinalStatus()+"\n");
fw.close();
```

タスク - アーカイバ

名前が示すように、このタスクでは、DBから不要になったレコードをアーカイブ(または削除)できます。

表6.9 アーカイバ・タスクの属性

タスク・タイプ	アーカイバ
次より古い	時間(分): 不要と評価されたレコードを指定します。指定した間隔より古いレコードがアーカイブに格納されます。
アーカイバ・タイプ	使用可能な値は"archive"または"delete"の2つです。削除する場合、UNDO操作は一切不可能な状態でレコードを削除します。アーカイブする場合、レコードがDBから削除されますが、削除されたデータが含まれるCSVファイルを使用してZIPパッケージが作成されます。
アーカイブの出力パス	この属性が有効なのは、アーカイブタイプに対してのみです。
実行履歴を含める	
ステータスのある実行レコード	ステータスが選択されている場合、指定したステータスを持つ実行レコードのみがアーカイブされます。これが役立つのは、たとえば、正常に終了したジョブのレコードを削除するが、失敗したジョブを将来の調査用として保持する場合などです。
一時ファイルを含める	選択すると、アーカイバにより、次より古い属性に定義されている特定のタイムスタンプより古いグラフ一時ファイルがすべて削除されます。一時ファイルは、グラフ・デバッグ・データ、ディクショナリ・ファイルおよびグラフ・コンポーネントにより作成されたファイルです。

レコード・ステータスのある一時ファイル	ステータスが選択されている場合、選択したステータスを持つ実行レコードに関連する一時ファイルのみがアーカイブされます。これが役立つのは、たとえば、正常に終了したジョブのファイルを削除するが、失敗したジョブを将来の調査用として保持する場合などです。
タスク履歴を含める	選択すると、実行レコードがアーカイバに含まれます。グラフ実行のログ・ファイルも含まれます。
タスク・タイプ	このタスク・タイプを選択すると、選択したタスク・タイプのログのみがアーカイブされます。
タスク結果マスク	タスク・ログ結果属性にマスクが適用されます。結果がこのマスクと一致するレコードのみがアーカイブされます。ワイルドカードなしで文字列を指定します。指定した文字列が結果属性に含まれる各タスク・ログが削除/アーカイブされます。大/小文字が区別されるかどうかは、データベース照合によって異なります。
プロファイラ実行を含む	選択すると、プロファイラ・ジョブ結果がアーカイバに含まれます。

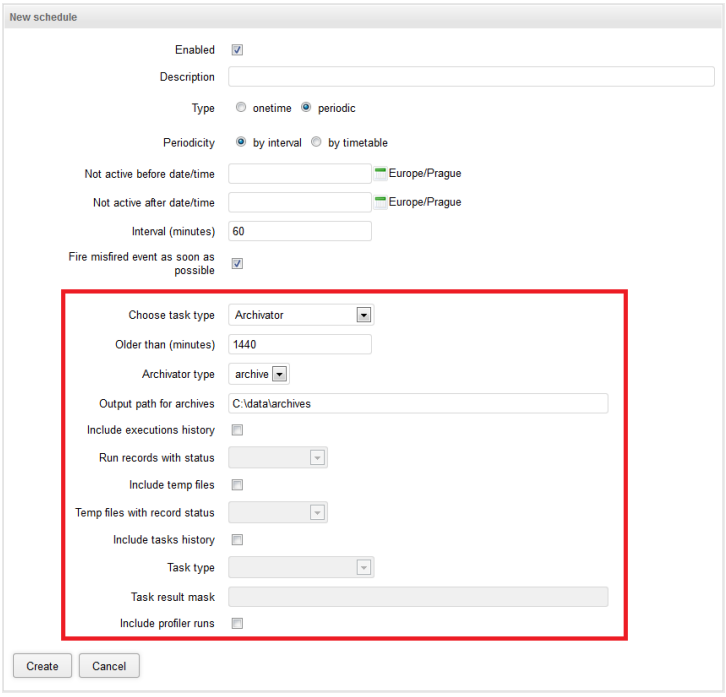


図6.10 Web GUI - レコードのアーカイブ

第7章 グラフ・イベント・リスナー

グラフ・イベント・リスナーでは、サーバーが特定のジョブ(変換グラフ)の成功または失敗に応じて実行するタスクを定義できます。

各リスナーは特定のグラフにバインドされており、(手動、スケジュール済、APIコール経由のいずれかにかかわらず)グラフが実行されるたびに評価されます。

リスナーを使用して、複数のジョブをチェーン(行の次のジョブを開始する成功リスナーを作成)できます。ただし、ジョブフローは開発および監視機能が向上しているため、ジョブフローを使用して複雑なプロセスを自動化することをお薦めします。

グラフ・イベント・リスナーは、ジョブフロー・イベント・リスナー(8章「ジョブフロー・イベント・リスナー」)に似ています。CloverETL Serverにとっては、どちらも単なるジョブです。

クラスタでは、イベントおよび関連タスクはジョブが実行されたのと同じノード上で実行されます。グラフが分散している場合、タスクはマスター・ワーカー・ノードで実行されます。ただし、タスク定義でノードIDを明示的に指定することで、タスクが実行される場所をオーバーライドできます。

グラフ・イベント

各イベントには、イベントのソースであるグラフのプロパティが設定されています。イベント・リスナーが指定されている場合、タスクではこれらのプロパティを使用できます。つまり、チェーン内の次のグラフでは、チェーン内の最初のグラフをアクティブ化する"EVENT_FILE_NAME"プレースホルダを使用できます。グラフ実行(つまり、RUN_ID)ごとに明確に設定されているグラフ・プロパティは、最後のグラフによってオーバーライドされます。

現時点では、次のタイプのグラフ・イベントが存在します。

- 「グラフ開始」
- 「グラフ・フェーズ終了」
- 「グラフ正常終了」
- 「グラフ・エラー」
- 「グラフ中断」
- 「グラフ・タイムアウト」
- 「グラフ・ステータス不明」

グラフ開始

ETLグラフ実行が正常に開始された場合、このタイプのイベントが作成されます。

グラフ・フェーズ終了

グラフ・フェーズが終了し、すべてのノードがステータス**FINISHED_OK**で終了するたびに、このタイプのイベントが作成されます。

グラフ正常終了

グラフのすべてのフェーズおよびノードがステータス**FINISHED_OK**で終了した場合、このタイプのイベントが作成されます。

グラフ・エラー

なんらかの理由によりグラフを実行できない場合またはグラフの任意のノードに障害が発生した場合、このタイプのイベントが作成されます。

グラフ中断

グラフが明示的に中断された場合、このタイプのイベントが作成されます。

グラフ・タイムアウト

指定した間隔より長くグラフが実行された場合、このタイプのイベントが作成されます。このため、グラフ・タイムアウト・イベントのリスナーごとにジョブ・タイムアウト間隔属性を指定する必要があります。この間隔は、秒、分または時単位で指定できます。

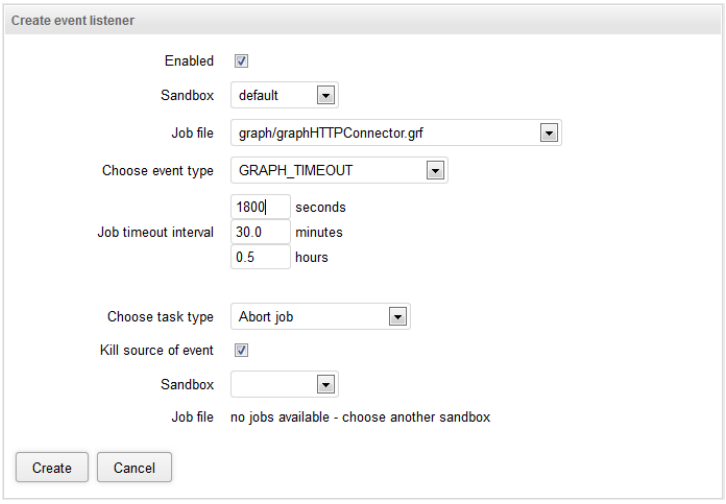


図7.1 Web GUI - グラフ・タイムアウト・イベント

グラフ・ステータス不明

起動時に、実行履歴内で未定義のステータスの実行レコードが検出された場合、このタイプのイベントが作成されます。未定義のステータスとは、グラフの実行時にサーバーが強制終了されたことを意味します。グラフの状態は自動的に使用不可に変更され、グラフ・ステータス不明イベントが送信されます。このように機能するのは、実行履歴内に永続レコードがある実行の場合のみです。実行履歴内に永続レコードがなくても変換を実行することはできますが、これは通常、パフォーマンスを向上させたり変換を高速に実行する場合などです(たとえば、起動サービスを使用する場合など)。

リスナー

指定したイベント・タイプおよびグラフ(またはサンドボックス内のすべてのグラフ)に対してリスナーを作成できます。リスナーは実際に、グラフ・イベントとタスク間の接続として機能します。この場合、グラフ・イベントによっていつ実行するかが指定され、タスクによって何を実行するかが指定されます。

このため、処理は次のように進行します。

· イベントが作成されます

· このイベントのリスナーが通知されます

・ 各リスナーによって関連タスクが実行されます

タスク

"シェル・コマンドの実行"、"グラフの実行"および"アーカイバ"の各タスク・タイプの詳細は、スケジューリングの項に説明があります。これらのタスク・タイプの詳細は、この項を参照してください。特にグラフ・イベント・リスナーの場合に役立つタスク・タイプがもう1つあるため、これについてもここで説明します。これは、電子メールの送信タスク・タイプです。

注意: スケジューリングとグラフ・イベント・リスナーには両方とも、任意のタイプのタスクを使用できます。タスク・タイプの説明は、最も明白なユースケースを示すために2つの項に分けられています。

・ 「タスク - 電子メールの送信」

・ 「タスク - JMSメッセージ」

タスク - 電子メールの送信

このタスク・タイプは、グラフ実行の結果に関する通知用として役立ちます。つまり、指定したサンドボックス内の失敗または特定のグラフの失敗ごとに通知するリスナーをこのタスク・タイプを使用して作成できます。

表7.1 電子メールの送信タスクの属性

タスク・タイプ	電子メール
電子メール・パターン	この選択ボックスは、ユーザーが新規レコードを作成している場合にのみ使用できます。これには、事前定義された電子メール・パターンがすべて含まれます。これらのいずれかを選択すると、この下のすべてのフィールドにパターンの値が入力されます。
宛先	受信者の電子メール・アドレス。複数のアドレスをカンマで区切って指定できます。また、プレースホルダを使用することもできます。詳細は、「プレースホルダ」を参照してください。
Cc	Ccは、カーボン・コピーを表します。これらのアドレスに電子メールのコピーが送信されます。複数のアドレスをカンマで区切って指定できます。また、プレースホルダを使用することもできます。詳細は、「プレースホルダ」を参照してください。
Bcc	Bccは、ブラインド・カーボン・コピーを表します。これはCcと同じですが、これらの受信者が電子メールのコピーを受信していることが他の受信者にはわかりません。
返信先(送信者)	送信者の電子メール・アドレス。これは、SMTPサーバーに応じて有効なアドレスである必要があります。また、プレースホルダを使用することもできます。詳細は、「プレースホルダ」を参照してください。
件名	電子メールの件名。また、プレースホルダを使用することもできます。詳細は、「プレースホルダ」を参照してください。
テキスト	プレーン・テキスト形式の電子メールの本文。電子メールはマルチパートとして作成されるため、HTMLの本文が優先されます。プレーン・テキストの本文は、HTMLを表示しない電子メール・クライアント用です。また、プレースホルダを使用することもできます。詳細は、「プレースホルダ」を参照してください。
HTML	HTML形式の電子メールの本文。電子メールはマルチパートとして作成されるため、HTMLの本文が優先されます。プレーン・テキストの本文は、HTMLを表示しない電子メール・クライアント

	用です。また、プレースホルダを使用することもできます。詳細は、「プレースホルダ」を参照してください。
添付としてのログ・ファイル	このスイッチを選択すると、電子メールには、関連するグラフ実行がパックされたログ・ファイルが添付されます。

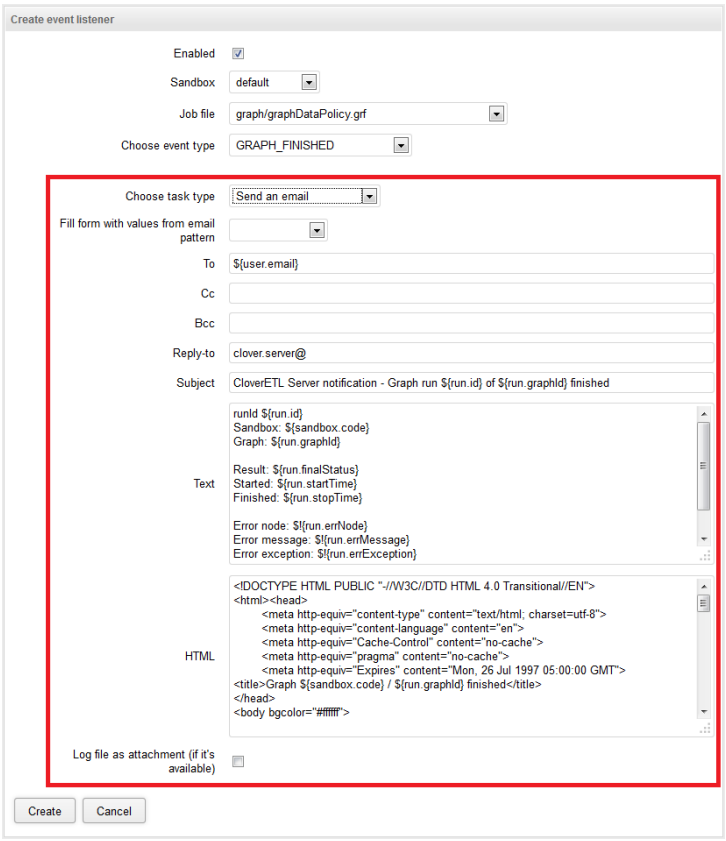


図7.2 Web GUI - 電子メールの送信

注意: SMTPサーバーへの接続を必ず構成してください(18章「構成」を参照してください)。

プレースホルダ

プレースホルダは、タスクの各種フィールドで使用できます。これらは特に、コンテキスト変数に応じて電子メールのコンテンツを生成できる、電子メール・タスクで便利です。

注意: ほとんどの場合、電子メール・パターンを使用することによってこれを回避できます(詳細は、電子メール・タスクを参照)。

これらのフィールドは、Apache Velocityテンプレート・エンジンによって事前に処理されます。構文の詳細は、VelocityプロジェクトのURL (http://velocity.apache.org/)を参照してください。

プレースホルダで使用できる他に、ループおよび条件の作成にも使用できるコンテキスト変数が複数用意されています。

- event
- now
- user
- run
- sandbox

これらの一部は、フィールドが処理される状況によっては空である場合があります。つまり、タスクがグラフ・イベントのために処理される場合、runおよびsandbox変数には関連データが含まれ、それ以外の場合は空になります。

表7.2 電子メール・テンプレートで役立つプレースホルダ

変数名	内容
now	現在の日付/時間
user	このイベントを引き起こしたユーザー。これは、スケジュールの所有者である場合や、グラフを実行したユーザーである場合があります。ドット表記法(例: ${user.email})を使用してアクセス可能なサブプロパティ(email、username、firstName、lastName、groups (値リスト))が含まれます
run	1回のグラフ実行を表すデータ構造。ドット表記法(例: ${run.graphId})を使用してアクセス可能なサブプロパティ(graphId、finalStatus、startTime、stopTime、errNode、errMessage、errException、logLocation)が含まれます

tracking	グラフ実行時のコンポーネントのステータスを表すデータ構造。ループおよび条件のVelocity構文を使用してアクセス可能なサブプロパティが含まれます。
	`#if (${tracking})`
	`<table border="1" cellpadding="2" cellspacing="0">`
	`#foreach ($phase in $tracking.trackingPhases)`
	`<tr><td>phase: ${phase.phaseNumber}`
	`</td> <td>${phase.execTime} ms</td>`
	`<td></td><td></td><td></td></tr>`
	`#foreach ($node in $phase.trackingNodes)`
	`<tr><td>${node.nodeName}</td>`
	`<td>${node.result}</td>`
	`<td></td><td></td><td></td></tr>`
	`#foreach ($port in $node.trackingPorts)`
	`<tr><td></td><td></td>`
	`<td>${port.portType}:${port.index}</td>`
	`<td>${port.totalBytes} B</td>`
	`<td>${port.totalRows} rows</td></tr>`
	`#end`
	`#end`
	`#end`
	`</table>`
	`#end`
	`}`

sandbox	実行したグラフが含まれるサンドボックスを表すデータ構造。ドット表記法(例: ${sandbox.name})を使用してアクセス可能なサブプロパティ(name、code、rootPath)が含まれます
schedule	このタスクをトリガーしたスケジュールを表すデータ構造。ドット表記法(例: ${schedule.description})を使用してアクセス可能なサブプロパティ(description、startTime、endTime、lastEvent、nextEvent、fireMisfired)が含まれます

タスク - JMSメッセージ

このタスク・タイプは、グラフ実行の結果に関する通知用として役立ちます。つまり、指定したサンドボックス内の失敗または特定のグラフの失敗ごとに通知するグラフ・イベント・リスナーをこのタスク・タイプを使用して作成できます。

JMSメッセージングには、JMS API (jms.jar)およびサード・パーティのライブラリが必要です。これらすべてのライブラリは、アプリケーション・サーバーのクラスパスで使用可能である必要がありま

す。一部のアプリケーション・サーバーにはこれらのライブラリがデフォルトで含まれますが、一部のアプリケーション・サーバーには含まれません。このため、これらのライブラリは明示的に追加する必要があります。

表7.3 JMSメッセージ・タスクの属性

タスク・タイプ	JMSメッセージ
初期コンテキスト・クラス名	javax.naming.InitialContext実装の完全クラス名。JMSプロバイダごとに独自の実装があります。たとえば、Apache MQの場合は"org.apache.activemq.jndi.ActiveMQInitialContextFactory"です。空である場合、デフォルトの初期コンテキストが使用されます
コネクション・ファクトリJNDI名	コネクション・ファクトリのJNDI名。JMSプロバイダによって異なります。
宛先JNDI名	サーバー上のメッセージ・キュー/トピックのJNDI名
ユーザー名	JMSメッセージ・ブローカに接続するためのユーザー名
パスワード	JMSメッセージ・ブローカに接続するためのパスワード
URL	JMSメッセージ・ブローカのURL
JMSパターン	この選択ボックスは、ユーザーが新規レコードを作成している場合にのみ使用できます。これには、事前定義されたJMSメッセージ・パターンがすべて含まれます。これらのいずれかを選択すると、この下のテキスト・フィールドにパターンの値が自動的に入力されます。
テキスト	JMSメッセージの本文。また、プレースホルダを使用することもできます。詳細は、電子メールの送信タスクの「プレースホルダ」を参照してください。

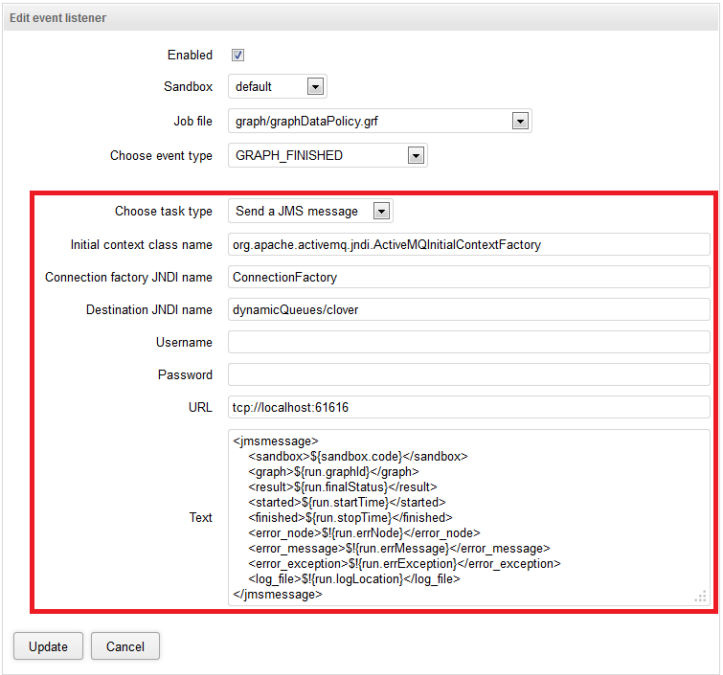


図7.3 Web GUI - タスクJMSメッセージのエディタ

ユースケース

使用可能なユースケースは、次のとおりです。

- 「チェーン内のグラフの実行」
- 「グラフの失敗に関する電子メール通知」
- 「グラフの成功に関する電子メール通知」
- 「グラフによって処理されたデータのバックアップ」

チェーン内のグラフの実行

たとえば、グラフBを実行するのは、別のグラフAがエラーなしで終了した場合であるとします。このため、これらのグラフ間にはなんらかの関係があります。このような動作を実現するには、グラフ・イベント・リスナーを作成します。グラフAのイベント**graph finished OK**に対してリスナーを作成し、グラフBが実行対象として指定された状態でタスク・タイプ**execute graph**を選択します。これで完了です。グラフCが実行対象として指定された状態でタスク**execute graph**を選択してグラフBに対してリスナーを作成すると、グラフのチェーンとして機能します。

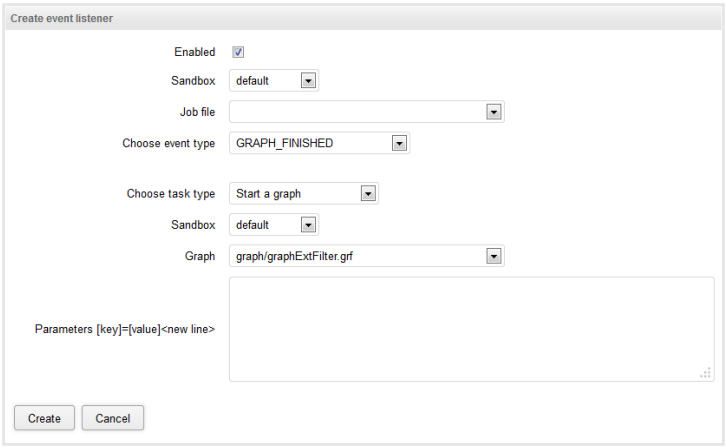


図7.4 イベント・ソース・グラフが指定されていないため、指定したサンドボックス内のすべてのグラフに対してリスナーが機能

グラフの失敗に関する電子メール通知

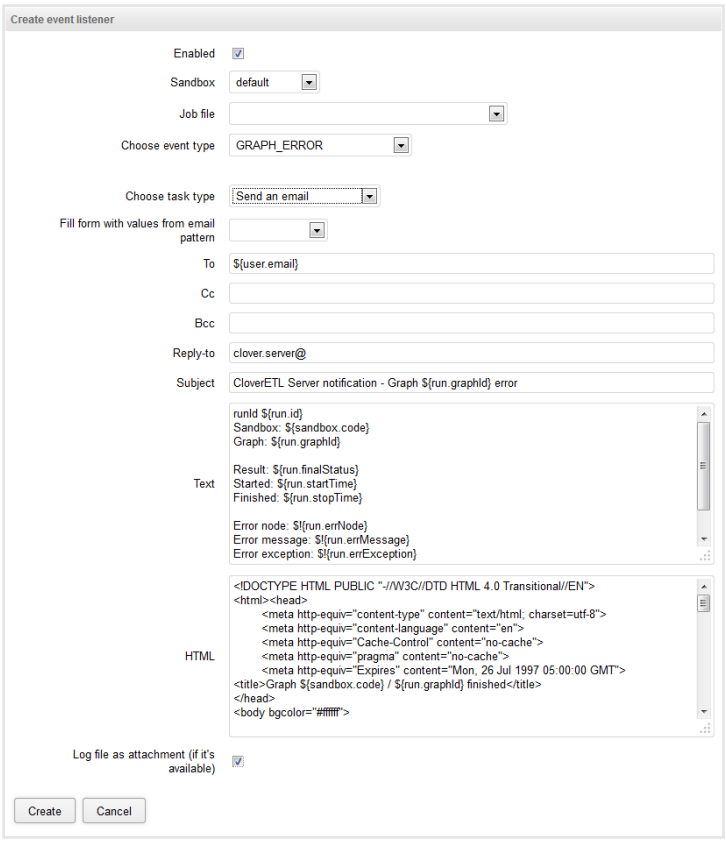


図7.5 Web GUI - グラフの失敗に関する電子メール通知

グラフの成功に関する電子メール通知

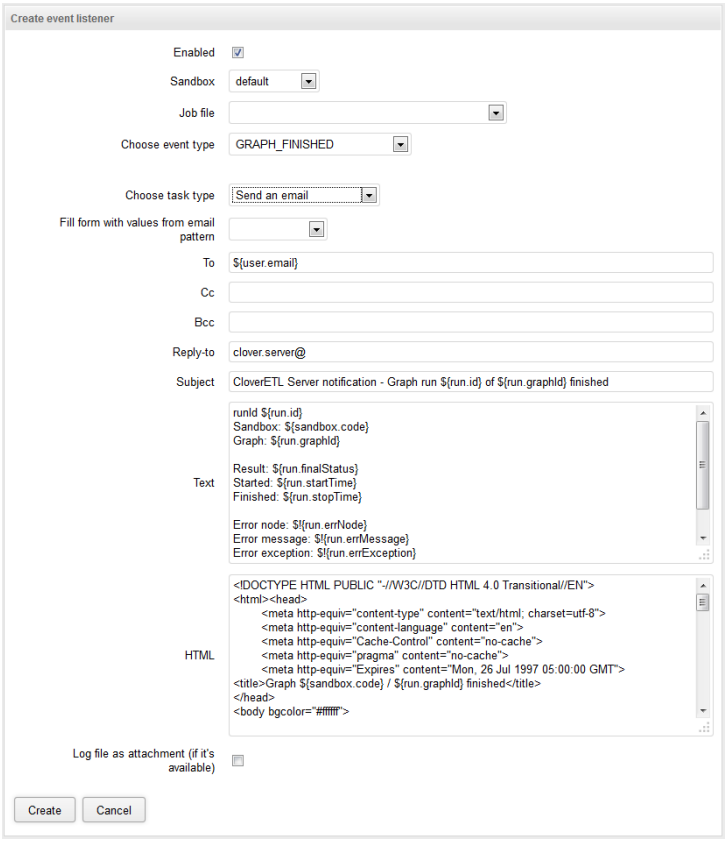


図7.6 Web GUI - グラフの成功に関する電子メール通知

グラフによって処理されたデータのバックアップ

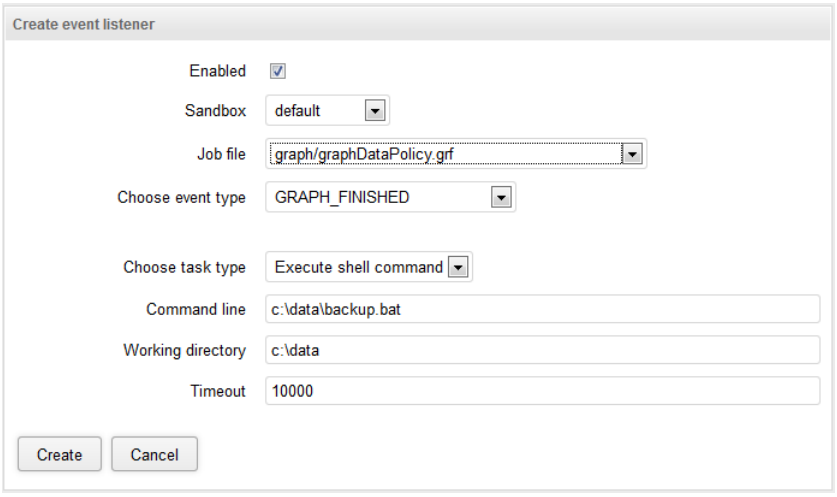


図7.7 Web GUI - グラフによって処理されたデータのバックアップ

第8章 ジョブフロー・イベント・リスナー

ジョブフロー・イベント・リスナーでは、サーバーが特定のジョブ(ジョブフロー)の成功または失敗に応じて実行するタスクを定義できます。

各リスナーは特定のジョブフローにバインドされており、(手動、別のジョブフローを使用、スケジュール済、APIコール経由のいずれかにかかわらず)ジョブフローが実行されるたびに評価されます。

ジョブフロー・イベント・リスナーの機能は、多くの用途においてグラフ・イベント・リスナーと非常に類似しています(タスクの項)。これは、ETLグラフとジョブフローは、CloverETLサーバーの観点からは両方ともジョブであるためです。

クラスタでは、イベントおよび関連タスクはジョブが実行されたのと同じノード上で実行されます。ジョブフローが分散している場合、タスクはマスター・ワーカー・ノードで実行されます。ただし、タスク定義でノードIDを明示的に指定することで、タスクが実行される場所をオーバーライドできます。

ジョブフロー・イベント

各イベントには、イベント・ソース・ジョブのプロパティが含まれます。イベント・リスナーが指定されている場合、タスクではこれらのプロパティを使用できます。たとえば、チェーン内の次のジョブでは、チェーン内の最初のジョブをアクティブ化する"EVENT_FILE_NAME"プレースホルダを使用できます。実行(たとえば、RUN_ID)ごとに明確に設定されているジョブ・プロパティは、最後のジョブによってオーバーライドされます。

次のタイプのジョブフロー・イベントがあります。

- 「ジョブフロー開始」
- 「ジョブフロー・フェーズ終了」
- 「ジョブフロー正常終了」
- 「ジョブフロー・エラー」
- 「ジョブフロー中断」
- 「ジョブフロー・タイムアウト」
- 「ジョブフロー・ステータス不明」

ジョブフロー開始

ジョブフロー実行が正常に開始された場合、このタイプのイベントが作成されます。

ジョブフロー・フェーズ終了

ジョブフロー・フェーズが終了し、すべてのノードがステータス**FINISHED_OK**で終了するたびに、このタイプのイベントが作成されます。

ジョブフロー正常終了

ジョブフローのすべてのフェーズおよびノードがステータス**FINISHED_OK**で終了した場合、このタイプのイベントが作成されます。

ジョブフロー・エラー

なんらかの理由によりジョブフローを実行できない場合またはジョブフローの任意のノードに障害が発生した場合、このタイプのイベントが作成されます。

ジョブフロー中断

ジョブフローが明示的に中断された場合、このタイプのイベントが作成されます。

ジョブフロー・タイムアウト

指定した間隔より長くジョブフローが実行された場合、このタイプのイベントが作成されます。このため、ジョブフロー・タイムアウト・イベントのリスナーごとにジョブ・タイムアウト間隔属性を指定する必要があります。この間隔は、秒、分または時単位で指定できます。

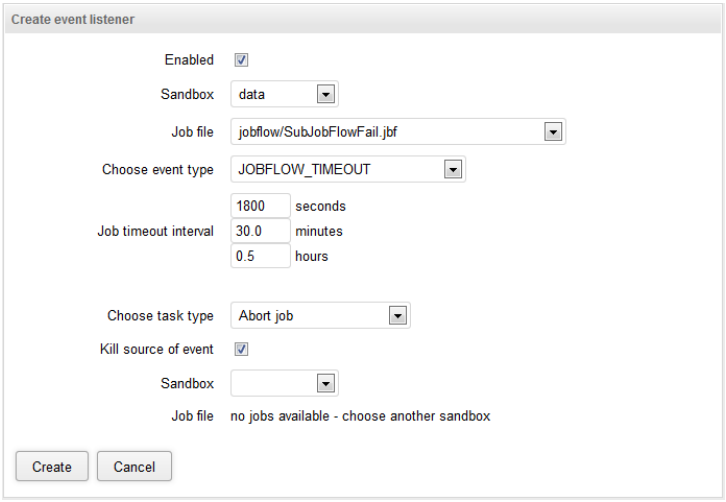


図8.1 Web GUI - ジョブフロー・タイムアウト・イベント

ジョブフロー・ステータス不明

起動時に、実行履歴内で未定義のステータスの実行レコードが検出された場合、このタイプのイベントが作成されます。未定義のステータスとは、ジョブフローの実行時にサーバーが強制終了されたことを意味します。ジョブフローの状態は自動的に使用不可に変更され、ジョブフロー・ステータス不明イベントが送信されます。このように機能するのは、実行履歴内に永続レコードがある実行の場合のみです。実行履歴内に永続レコードがなくても変換を実行することはできますが、これは通常、パフォーマンスを向上させたり変換を高速に実行する場合などです(たとえば、起動サービスを使用する場合など)。

リスナー

指定したイベント・タイプおよびジョブフロー(またはサンドボックス内のすべてのジョブフロー)に対してリスナーを作成できます。リスナーは実際に、ジョブフロー・イベントとタスク間の接続として機能します。この場合、ジョブフロー・イベントによっていつ実行するかが指定され、タスクによって何を実行するかが指定されます。

このため、処理は次のように進行します。

- イベントが作成されます
- このイベントのリスナーが通知されます

· 各リスナーによって関連タスクが実行されます

タスク

タスクにより、トリガーされたイベントに対する対応として実行する必要がある操作を指定します。

タスク・タイプは、「タスク」および「タスク」で説明されています。

注意: ジョブフロー・イベント・リスナーには、任意のタイプのタスクを使用できます。タスク・タイプの説明は、最も明白なユースケースを示すために2つの項に分けられています。

第9章 JMSメッセージ・リスナー

JMSメッセージ・リスナーでは、受信JMSメッセージをリスニングできます。メッセージのソース(JMSトピックまたはJMSキュー)および各受信メッセージに対して実行されるタスクを指定します。

JMSメッセージングには、JMS API (jms.jar)および特定のサード・パーティのライブラリが必要です。これらすべてのライブラリは、アプリケーション・サーバーのクラスパスで使用可能である必要があります。アプリケーション・サーバーには、デフォルトでこれらのライブラリが含まれているものと含まれていないものがあります。このような場合、CloverETL Serverを起動する前にライブラリを明示的に追加する必要があります。

JMSは、このドキュメントの範囲を超えた複雑なトピックです。JMSの詳細は、Oracle Webサイト(http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html)を参照してください。

JMS実装は、CloverETL Serverが実行されているアプリケーション・サーバーに依存します。

クラスタでは、どのノードがJMSをリスニングするかしないかを明示的に指定できます。未指定の場合は、すべてのノードがリスナーとして登録されます。JMSトピックの場合は、すべてのノードがメッセージを取得してタスク(複数インスタンス)をトリガーし、JMSキューの場合は、ランダム・ノードがメッセージを消費し、タスク(1つのインスタンスのみ)を実行します。

表9.1 JMSメッセージ・タスクの属性

属性	説明
イベントを処理するためのノードID	この属性が有効なのは、クラスタ環境の場合のみです。これは、リスナーを初期化する必要があるノードIDです。設定しない場合、リスナーはクラスタ内のすべてのノードで初期化されます。
初期コンテキスト・クラス名	javax.naming.InitialContext実装の完全クラス名。JMSプロバイダごとに独自の実装があります。たとえば、Apache MQの場合は"org.apache.activemq.jndi.ActiveMQInitialContextFactory"です。空である場合、デフォルトの初期コンテキストが使用されます。指定したクラスは、web-appクラスパスまたはapplication-serverクラスパス上にある必要があります。これは通常、特定のJMSブローカ・プロバイダごとにJMS APIが実装された1つのライブラリ内にあります。
コネクション・ファクトリJNDI名	コネクション・ファクトリのJNDI名。JMSプロバイダによって異なります。
宛先JNDI名	サーバー上のメッセージ・キュー/トピックのJNDI名
ユーザー名	JMSメッセージ・ブローカに接続するためのユーザー名
パスワード	JMSメッセージ・ブローカに接続するためのパスワード
URL	JMSメッセージ・ブローカのURL
恒久サブスクライバ(トピック専用)	FALSEである場合、メッセージ・コンシューマは非恒久としてブローカに接続されるため、接続がアクティブであるときに送信されたメッセージのみを受信します。他のメッセージは失われます。TRUEである場合、コンシューマは恒久としてサブスクライブされるため、接続が非アクティブであるときに送信されたメッセージも受信します。このようなメッセージは、送信可能になるまで、または有効期限が切れるまで格納されます。このスイッチが有効なのはトピック宛先の場合のみです。これは、キュー宛先の場合、メッセージは送信可能になるまで、または有効期限が切れるまで常に格納されるためです。サーバーの再起動中や、JMSメッセージ・リスナーが更新される間の一瞬、コンシューマは非アクティブになるため、再度初期化する必要があります。このため、コンシューマに恒久サブスクリプションが設定されていない場合、このような合間にトピック内のメッセージが失われる可能性があります。 サブスクリプションが恒久である場合、クライアントには"ClientId"を指定する必要があります。この属性は、JMSプロバイダに応じて様々な方

属性	説明
	法で設定できます。たとえば、ActiveMQの場合は、URLパラメータtcp://localhost:1244?jms.clientID=TestClientIDとして設定されます。
メッセージ・セレクタ	この問合せ文字列は、受信メッセージをフィルタするための条件の仕様として使用できます。構文の詳細は、Java EE APIのWebサイト(http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html)を参照してください。これは、コンシューマのタイプ(キュー/トピック)に応じて動作が異なります。キューの場合、フィルタで除外されたメッセージはキュー内に残ります。トピックの場合、トピック・サブスクライバのメッセージ・セレクタによってフィルタで除外されたメッセージがサブスクライバに送信されることはありません。サブスクライバの観点からは、これらは存在しません。
Groovyコード	Groovyコードは、追加メッセージ処理やメッセージの拒否用として使用できます。これら両方の機能については次の説明を参照してください。

オプションのGroovyコード

Groovyコードは、追加メッセージ処理やメッセージの拒否用として使用できます。

- 追加メッセージ処理 Groovyコードにより、コンテナ"properties"および"data"に格納されている値を変更/追加/削除できます。

- メッセージの拒否/承認 GroovyコードからBoolean.FALSEが戻されると、メッセージは拒否されます。それ以外の場合、メッセージは承認されます。拒否されたメッセージを再送信することもできますが、JMSブローカにより、メッセージの再送信に関する制限が構成されている必要があります。Groovyコードによって例外がスローされた場合、コーディング・エラーとみなされ、これが原因でJMSメッセージは拒否されません。このため、例外によってメッセージの拒否が指示された場合、これはGroovyで処理する必要があります。

表9.2 Groovyコードでアクセス可能な変数

タイプ	キー	説明
javax.jms.Message	msg	JMSメッセージのインスタンス
java.util.Properties	properties	詳細は、次の説明を参照してください。メッセージから読み取られた値(文字列または文字列に変換されたもの)が含まれます。これは、タスクに渡され、タスクによってなんらかの方法で使用されます。たとえば、グラフの実行タスクの場合、これらのパラメータを実行対象グラフに渡します。
java.util.Map<String, Object>	data	詳細は、次の説明を参照してください。メッセージ・インスタンスから読み取られたかプロキシされた値(オブジェクト、ストリームなど)が含まれます。これは、タスクに渡され、タスクによってなんらかの方法で使用されます。たとえば、グラフの実行タスクの場合、これをディクショナリ・エントリとして実行対象グラフに渡します。
javax.servlet.ServletContext	servlet Context	ServletContextのインスタンス
javax.jms.Message	msg	JMSメッセージのインスタンス

タイプ	キー	説明
com.cloveretl.server.api.ServerFacade	serverFacade	CloverETL Serverコア機能をコールする場合に使用可能なserverFacadeのインスタンス。
String	sessionToken	serverFacadeメソッドのコールに必要なsessionToken

追加処理に使用可能なメッセージ・データ

JMSメッセージは処理され、これに含まれるデータはプロパティの2つのデータ構造に格納されます。

表9.3 プロパティ要素

キー	説明
JMS_PROP_[property key]	各メッセージ・プロパティについて、"key"が接頭辞"JMS_PROP_"およびプロパティ・キーから構成される1つのエントリが作成されます。
JMS_MAP_[map entry key]	メッセージがMapMessageのインスタンスの場合、各マップ・エントリについて、"key"が接頭辞"JMS_MAP_"およびマップ・エントリ・キーから構成される1つのエントリが作成されます。値は文字列に変換されます。
JMS_TEXT	メッセージがTextMessageのインスタンスの場合、このプロパティにはメッセージの内容が含まれます。
JMS_MSG_CLASS	メッセージ実装のクラス名
JMS_MSG_CORRELATIONID	相関IDは、プロバイダ固有のメッセージIDまたはアプリケーション固有の文字列値です
JMS_MSG_DESTINATION	JMSDestinationヘッダー・フィールドには、メッセージが送信される宛先が含まれます。
JMS_MSG_MESSAGEID	JMSMessageIDは、履歴リポジトリ内のメッセージを識別するための一意キーとして機能する必要のある文字列値です。一意性の正確な範囲がプロバイダによって定義されています。少なくとも、インストールが接続されたメッセージ・ルーターのセットである、プロバイダの特定のインストールのすべてのメッセージに対応する必要があります。
JMS_MSG_REPLYTO	このメッセージへの返信を送信する必要のある宛先。
JMS_MSG_TYPE	メッセージが送信されたときにクライアントによって提供されるメッセージ・タイプ識別子。
JMS_MSG_DELIVERYMODE	このメッセージに対して指定されたDeliveryMode値。
JMS_MSG_EXPIRATION	メッセージの有効期限。これは、クライアントが指定する存続時間と送信時点のGMTの合計です。
JMS_MSG_PRIORITY	JMS APIは、最低の優先度が0で最高の優先度が9である10レベルの優先度値を定義します。また、クライアントは優先度0～4を標準優先度の段階、優先度5～9を緊急優先度の段階とみなす必要があります。
JMS_MSG_REDELIVERED	このメッセージが再配信される場合は"TRUE"です。
JMS_MSG_TIMESTAMP	メッセージが送信のためにプロバイダに渡された時刻。トランザクションまたはその他のクライアント側メッセージ・キューにより実際の送信は後に行われることがあるため、これはメッセージが実際に送信された時刻ではありません。

プロパティ構造のすべての値はString型として格納されますが、これらは数値またはテキストです。

下位互換性を保つために、リストされているプロパティはすべて、小文字のキーを使用してアクセスすることもできます。ただし、この方法はお薦めしません。

表9.4 データ要素

キー	説明
JMS_MSG	javax.jms.Messageのインスタンス
JMS_DATA_STREAM	java.io.InputStreamのインスタンス。TextMessage、BytesMessage、StreamMessage、ObjectMessage（ペイロード・オブジェクトがStringのインスタンスである場合のみ)の場合のみアクセスできます。文字列はUTF-8でエンコードされます。
JMS_DATA_TEXT	Stringのインスタンス。TextMessageおよびObjectMessage（ペイロード・オブジェクトがStringのインスタンスである場合)のみが対象です。
JMS_DATA_OBJECT	java.lang.Objectのインスタンス - メッセージ・ペイロード。ObjectMessageのみが対象です。

データ・コンテナは、その実装に応じてそれを使用するタスクに渡されます。たとえば、グラフの実行タスクの場合、これをディクショナリ・エントリとして実行対象グラフに渡します。

クラスタでは、タスクを実行する明示的なノードを指定できます。ただし、"data"ペイロードがシリアライズ可能でなく、受信ノードと実行ノードが異なる場合は、クラスタが"data"を実行ノードに渡せないためエラーがスローされます。

グラフまたはジョブフロー内では、ディクショナリ・エントリとして渡されるデータは一部のコンポーネント属性として使用できます。たとえば、ファイルURL属性は、プロキシ・ストリームを使用した受信JMSメッセージからの直接データ読取りの場合は"dict:JMS_DATA_STREAM:discrete"のようになります。

下位互換性を保つために、リストされているディクショナリ・エントリはすべて、小文字のキーを使用してアクセスすることもできます。ただし、この方法はお薦めしません。

第10章 ユニバーサル・イベント・リスナー

2.10以降

ユニバーサル・イベント・リスナーでは、イベントがいつトリガーされたかを制御し、その後で事前定義のタスクを実行するGroovyコードの一部を作成できます。Groovyコードは定期的に実行され、TRUEを返した場合にタスクが実行されます。

表10.1 ユニバーサル・メッセージ・タスクの属性

属性	説明
イベントを処理するためのノードID	(クラスタの場合のみ)。リスナーを初期化する必要のあるノードIDを指定します。設定しない場合、リスナーはクラスタ内のすべてのノードで初期化されます。
チェックインの間隔(秒)	Groovyコード実行の周期性。
Groovyコード	TRUE (タスクを実行)またはFALSE (アクションなし)に評価されるGroovyコード。詳細は、次の説明を参照してください。

Groovyコード

Groovyの一部は繰り返し実行および評価されます。結果に基づいて、イベントがトリガーされ、タスクが実行されるかアクションが実行されません。

たとえば、グラフを開始する前に重要なデータ・ソースを継続的に確認できます。または、実行グラフの複雑な確認を行い、たとえば必要に応じてこれを強制終了することも決定できます。ServerFacadeインタフェースを使用してCloverETL Serverコア機能をコールすることもできます。Javadocのhttp://host:port/clover/javadoc/index.htmlを参照してください。

評価基準

GroovyコードからBoolean.TRUEが戻された後に、イベントがトリガーされ、関連タスクが実行されます。それ以外の場合、何も行われません。

Groovyコードによって例外がスローされた場合、コーディング・エラーとみなされ、イベントはトリガーされません。このため、例外はGroovyコードで適切に処理する必要があります。

表10.2 Groovyコードでアクセス可能な変数

タイプ	キー	説明
java.util.Properties	properties	Groovyコード内のString-String型のキーと値のペアによって入力できる空のコンテナ。これは、タスクに渡され、タスクによってなんらかの方法で使用されます。たとえば、グラフの実行タスクの場合、これらのパラメータを実行対象グラフに渡します。
java.util.Map<String, Object>	data	Groovyコード内のString-Object型のキーと値のペアによって入力できる空のコンテナ。これは、タスクに渡され、タスクによって実装に応じてなんらかの方法で使用されます。たとえば、グラフの実行タスクの場合、これをディクショナリ・

タイプ	キー	説明
		エントリとして実行対象グラフに渡します。これはシリアライズ可能ではありません。このため、タスクがこれに依存している場合、これを正しく処理できるのは、同じクラスタ・ノード上である場合のみです。
javax.servlet.ServletContext	servlet Context	ServletContextのインスタンス
com.cloveretl.server.api.ServerFacade	serverFacade	CloverETL Serverコア機能をコールする場合に使用可能なserverFacadeのインスタンス。
String	sessionToken	serverFacadeメソッドのコールに必要なsessionToken

第11章 手動タスク実行

3.1以降

手動タスク実行では、イベントを定義およびトリガーすることなく、即時に影響のあるタスクを直接呼び出すことができます。

ファイル・リスナーやグラフ/ジョブフロー・リスナーなど、通常はトリガー・イベントに関連付けられているタスク・タイプがいくつかあります。これらのタスクはいずれも手動で実行できます。

また、通常はタスクをトリガーするソース・イベントをシミュレートするためのタスク・パラメータを指定できます。次の図は、ファイル・イベントをシミュレートする方法を示しています。様々なイベント・ソースのパラメータは、グラフ・パラメータの項を参照してください。

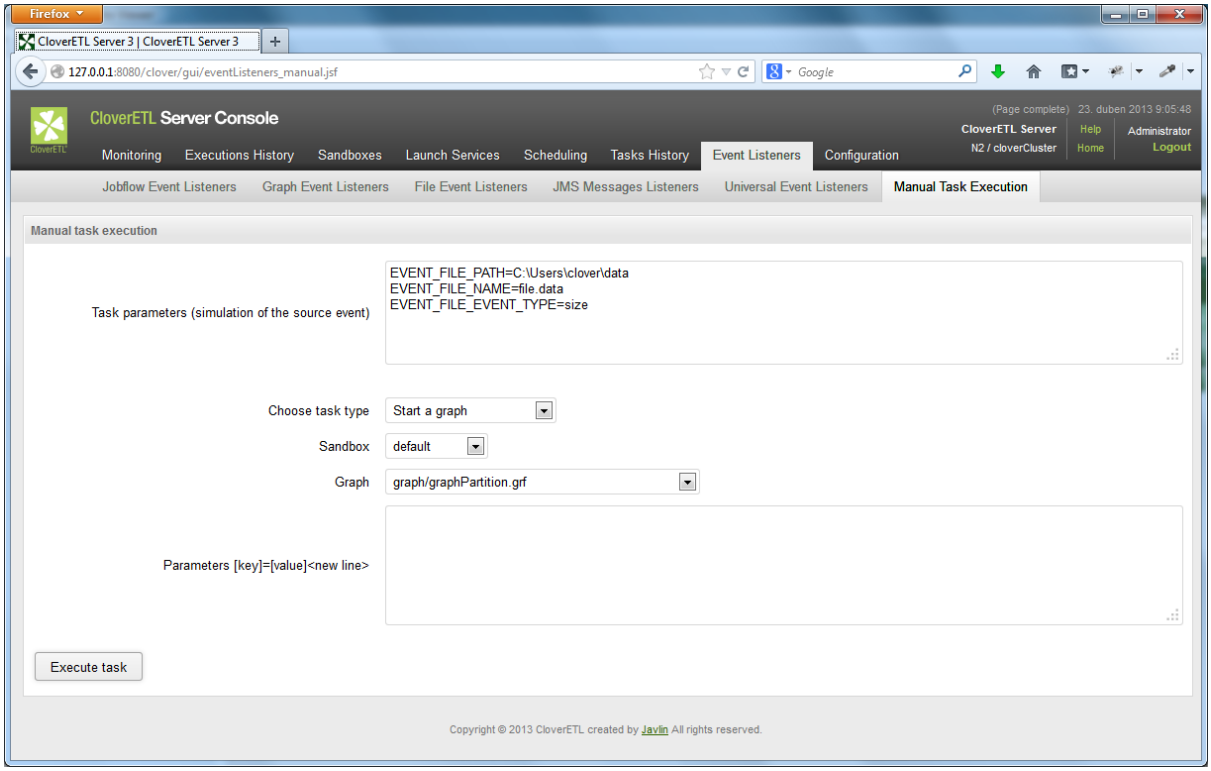


図11.1 Web GUI - 手動タスク実行セクション

第12章 ファイル・イベント・リスナー

1.3以降

ファイル・イベント・リスナーでは、フォルダに表示される新しいファイルなど、特定のファイル・システム・パスの変更を監視し、事前定義されたタスクでこのようなイベントに対応できます。

正確なパスを指定するかワイルドカードを使用して、秒単位のチェック間隔を設定し、最終的にイベントを処理するタスクを定義できます。

構成に必要な場合に変更できるグローバルな最小チェック間隔
("clover.event.fileCheckMinInterval"プロパティ)があります。

クラスタでは、各イベント・リスナーに、ローカル・ファイル・システムでチェックを実行するクラスタ・ノードを指定するノードID属性があります。スタンドアロン環境では、ノードID属性は無視されます。

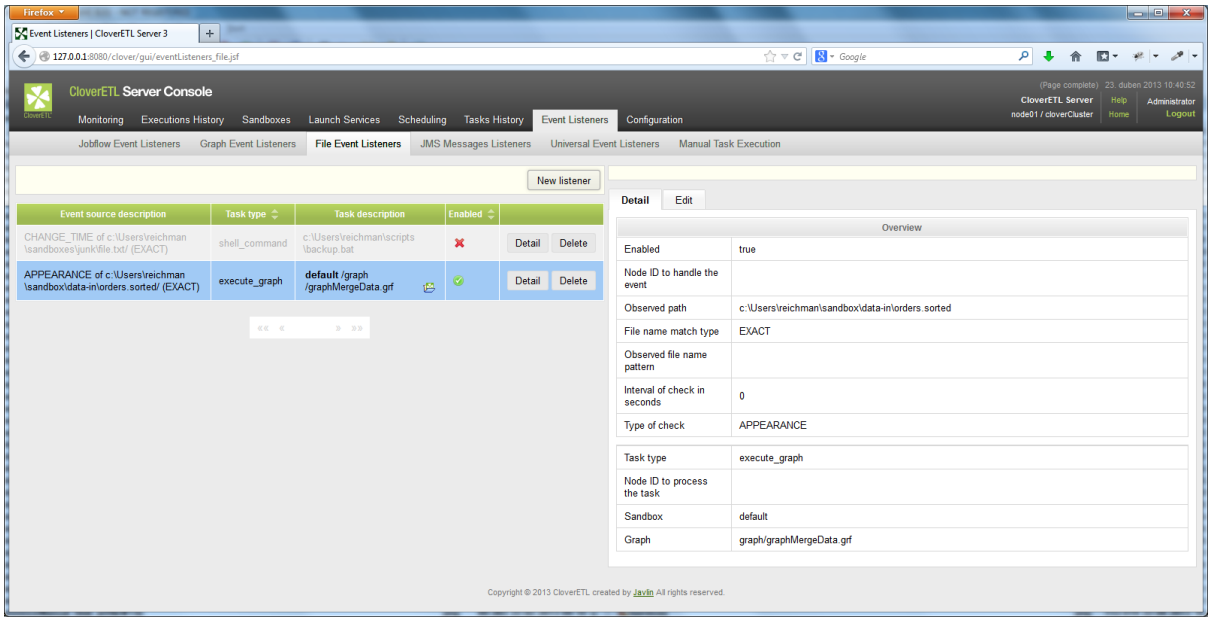


図12.1 Web GUI - ファイル・イベント・リスナー・セクション

監視対象ファイル

監視対象ファイルは、ディレクトリ・パスおよびファイル名パターンによって指定します。

厳密に1つのファイル名を指定することも、指定したディレクトリ内の複数の一致ファイルを監視するためにファイル名パターンを指定することもできます。パターンと一致する変更済ファイルが複数存在する場合、これらのファイルごとに個別イベントがトリガーされます。

監視対象ファイルのファイル名パターンを指定する方法は3通りあります

- 「完全一致」
- 「ワイルドカード」
- 「正規表現」

完全一致

監視対象ファイルの厳密な名前を指定します。

ワイルドカード

ほとんどのオペレーティング・システムで一般的なワイルドカード(*や?など)を使用できます。

- * - 任意の文字の0以上のインスタンスと一致します
- ? - 任意の1文字のインスタンスと一致します
- [...] - 大カッコで囲まれた任意の文字と一致します
- ¥ - エスケープ文字

例

- *.csv - すべてのCSVファイルと一致します
- input_*.csv - input_001.csv、input_9.csvなどと一致します
- input_???.csv - input_001.csvとは一致しますが、input_9.csvとは一致しません

正規表現

例

- (.*?)¥.(jpg|jpeg|png|gif)$ - イメージ・ファイルと一致します

注意

- 絶対パスを使用することをお薦めします。相対パスを使用することもできますが、作業ディレクトリはアプリケーション・サーバーに依存します。
- MS Windows OSの場合でも、ファイル・セパレータとしてフォワード・スラッシュを使用してください。バックスラッシュはエスケープ・シーケンスとみなされる可能性があります。

ファイル・イベント

リスナーごとに、該当するイベント・タイプを指定する必要があります。

次の4つのファイル・イベントがあります。

- 「ファイルAPPEARANCE」
- 「ファイルDISAPPEARANCE」
- 「ファイルSIZE」
- 「ファイルCHANGE_TIME」

ファイルAPPEARANCE

このタイプのイベントは、監視対象ファイルが2回の確認の間に別の場所から作成またはコピーされたときに発生します。このタイプのイベントは、新規ファイルが完成したかどうかとは関係なく、検出されると同時に発生することに注意してください。このため、ファイルがまだ不完全であっても完全なファイルを必要とする可能性があるタスクが実行されます。したがって、ファイルを別の場所に保存し、ファイルが完成したら、CloverETL Serverが検出できる監視対象場所に移動/名前変更することをお薦めします。ファイルの移動/名前変更はアトミック操作である必要があります。

このタイプのイベントは、ファイルが2回の確認の間に更新(タイムスタンプまたはサイズの変更)されている場合は発生しません。APPEARANCE (出現)とは、ファイルは前回の確認時には存在しなかったが、現在の確認時には存在することを意味します。

ファイルDISAPPEARANCE

このタイプのイベントは、監視対象ファイルが2回の確認の間に削除されたか別の場所に移動されたときに発生します。

ファイルSIZE

このタイプのイベントは、監視対象ファイルのサイズが2回の確認の間に変更されたときに発生します。このタイプのイベントは、ファイルが作成または削除された場合は発生しません。ファイルは両方の確認時に存在する必要があります。

ファイルCHANGE_TIME

このタイプのイベントは、監視対象ファイルの変更時間が2回の確認の間に変更されたときに発生します。このタイプのイベントは、ファイルが作成または削除された場合は発生しません。ファイルは両方の確認時に存在する必要があります。

確認間隔、タスクおよびユースケース

- 2回の確認の最小限の間隔を指定できます。これは秒単位で指定します。
- リスナーごとにタスクが定義され、これらのタスクはファイル・イベントに応じて処理されます。すべてのタスク・タイプおよびその属性の詳細は、スケジューリングおよびグラフ・イベント・リスナーの項を参照してください
- · グラフ実行、入力データがアクセス可能な場合
  - グラフ実行、入力データが更新された場合
  - グラフ実行、生成データがあるファイルが削除され、再作成する必要がある場合

タスク処理時にイベントのソースを使用する方法

イベントの原因となる(イベントのソースとみなされる)ファイルは、タスク処理時に使用できます。つまり、グラフ変換のリーダー・コンポーネント/ライター・コンポーネントは、特別なプレースホルダによってこのファイルを参照できます。**${EVENT_FILE_PATH}**は、イベント・ソースが含まれるディレクトリへのパスです。**${EVENT_FILE_NAME}**は、イベント・ソースの名前です。

前のバージョンでは、小文字のプレースホルダを指定していました。バージョン3.3以降、プレースホルダは大文字です。ただし、下位互換性のために小文字も機能します。

イベント・ソースが**/home/clover/data/customers.csv**の場合、プレースホルダには EVENT_FILE_PATH - **/home/clover/data**、EVENT_FILE_NAME - **customers.csv**が含まれます。

グラフ実行タスクの場合、これが機能するのは、グラフがプールされていない場合のみです。このため、保持プール間隔は0 (デフォルト値)に設定する必要があります。

第13章 WebDAV

3.1以降

WebDAV APIでは、標準WebDAV仕様を使用して、サンドボックスのコンテンツをアクセスおよび管理できます。

具体的には次の操作が可能です。

- ディレクトリ構造の参照
- ファイルの編集
- ファイル/フォルダの削除
- ファイル/フォルダの名前変更
- ファイル/フォルダの作成
- ファイルのコピー
- ファイルの移動

WebDAV インタフェースは、URL: "http://[host]:[port]/clover/webdav"からアクセスできます。

注意: このURLは一般的なブラウザで開くことができますが、これらのほとんどはリッチWebDAVクライアントではありません。そのため、アイテムのリストの表示のみが可能で、ブラウザではディレクトリ構造は参照できません。

WebDAVクライアント

様々なオペレーティング・システムを対象としたWebDAVクライアントは多数存在し、一部のOSはWebDAVをネイティブでサポートしています。

LinuxライクなOS

Linuxシステム上で動作する優れたWebDAVクライアントはKonquerorです。URL (webdav:// [host]:[port]/clover/webdav)にある別のプロトコルを使用してください。

MS Windows

MS Windowsの最後のディストリビューション(Win XP以降)は、WebDAVをネイティブでサポートしています。ただし、これは多少不安定であるため、無料または市販のWebDAVクライアントを使用することをお薦めします。

- 弊社でテストした最良のWebDAVクライアントはBitKinex (http://www.bitkinex.com/webdavclient)です
- もう1つのオプションは、Total Commander (http://www.ghisler.com/index.htm)をWebDAVプラグイン (http:// www.ghisler.com/plugins.htm#filesys)とともに使用する方法です

Mac OS

Mac OSは、WebDAVをネイティブでサポートしており、この場合、問題は発生しません。finderアプリケーションを使用し、Connect to the server ...メニュー項目を選択し、HTTPプロトコルでURL "http://[host]:[port]/clover/webdav"を使用します。

WebDAV認証/認可

CloverETL Server WebDAV APIでは、デフォルトでHTTP基本認証を使用します。ただし、HTTPダイジェスト認証を使用するよう再構成することもできます。詳細は、構成の項を参照してください。

一部のWebDAVクライアントはHTTP基本認証とは機能せず、ダイジェスト認証でのみ機能するため、ダイジェスト認証が役立つ場合があります。

HTTPダイジェスト認証は、バージョン3.1に追加された機能です。古いCloverETL Serverディストリビューションをアップグレードした場合、アップグレードの前に作成されたユーザーは、パスワードをリセットするまではHTTPダイジェスト認証を使用できません。このため、パスワードをリセット(または管理者がかわりにパスワードをリセット)すると、新規ユーザーと同様にダイジェスト認証を使用できるようになります。

第14章 単純なHTTP API

単純なHTTP APIは、単純なHTTPコールを使用して外部アプリケーションからサーバーを制御できる基本的なサーバー自動化ツールです。

すべての操作は、HTTPのGETメソッドを使用してアクセスでき、プレーン・テキストを戻します。このため、リクエストおよびレスポンスの両方を、非常に単純なツール(wget、grepなど)を使用して便利に送信および解析できます。

グローバル・セキュリティがオン(デフォルトではオン)である場合は、HTTP基本認証が使用されます。認証された操作では、対応する権限を持つ有効なユーザー資格証明が必要です。

ETLグラフ関連操作"graph_run"、"graph_status"および"graph_kill"は、ジョブフローおよびデータ・プロファイラ・ジョブに対しても機能します。

リクエストURLの汎用パターン:

```
http://[domain]:[port]/[context]/[servlet]/[operation]?
[param1]=[value1]&[param2]=[value2]...
```

wgetクライアントの場合、次のコマンドラインを使用できます。

```
wget --user=$USER --password=$PASS -O ./$OUTPUT_FILE $REQUEST_URL
```


help操作

パラメータ

なし

戻り値

使用可能な操作およびパラメータ(説明付き)のリスト

例

```
http://localhost:8080/clover/request_processor/help
```

graph_run操作

この操作をコールし、指定したジョブの実行を開始します。この操作は下位互換性のためにgraph_runと呼ばれますが、ETLグラフ、ジョブフローまたはプロファイラ・ジョブを実行できます。

パラメータ

表14.1 graph_runのパラメータ

パラメータ名	必須	デフォルト	説明
graphID	はい	-	サンドボックスのルートの相対ジョブ・ファイルのファイル・パス。
sandbox	はい	-	サンドボックスのテキストID
additional job parameters	いいえ		"param_"接頭辞が付いたURLパラメータは、実行対象ジョブに渡され、変換XMLでプレースホルダとして使用できますが、"param_"接頭辞がない場合、たとえば、URLで指定された"param_FILE_NAME"は、XML内で${FILE_NAME}として使用できます。これらのパラメータが解決されるのはXMLのロード中のみであるため、プールできません。
nodeID	いいえ	-	クラスタ・モードの場合は、ジョブを実行する必要があるノードのID。ただし、これは最終ノードではありません。グラフが分散している場合や、ノードが切断されている場合、グラフは別のノードでも実行できます。
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

実行ID: 各実行リクエストを識別する増分番号

例

```
http://localhost:8080/clover/request_processor/graph_run?graphID=graph/
graphDBExecute.grf&sandbox=mva
```

graph_status操作

この操作をコールし、指定したジョブ実行のステータスを取得します。この操作は下位互換性のためにgraph_statusと呼ばれますが、ETLグラフまたはジョブフローのステータスを戻すことができます。

パラメータ

表14.2 graph_statusのパラメータ

パラメータ名	必須	デフォルト	説明
runID	はい	-	各グラフ実行のID
returnType	いいえ	STATUS	STATUS | STATUS_TEXT | DESCRIPTION | DESCRIPTION_XML
waitForStatus	いいえ	-	待機対象のステータス・コード。指定すると、この操作はグラフが必要なステータスになるまで待機します。
waitTimeout	いいえ	0	waitForStatusが指定されている場合、指定したミリ秒数のみ待機します。デフォルトの0は永久を意味しますが、アプリケーション・サーバーの構成によって異なります。指定したタイムアウトの期限が切れたときに、グラフ実行がまだ必要なステータスになっていない場合、コード408 (リクエスト・タイムアウト)が戻されます。アプリケーション・サーバーのHTTPリクエスト・タイムアウトの期限が先に切れた場合も、408コードが戻される可能性があります。
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

指定したグラフのステータス。これは、オプションのパラメータreturnTypeに応じて、数値コード、テキスト・コードまたは複雑な説明である場合があります。説明は、セパレータとしてパイプが使用されたプレーン・テキストまたはXMLとして戻されます。XMLレスポンスのXML書式を表すスキーマには、CloverETL ServerのURL (http://[host]:[port]/clover/schemas/ executions.xsd)でアクセスできます。waitForStatusパラメータに応じて、結果は即時戻されるか、指定したステータスになってから戻されます。

例

```
http://localhost:8080/clover/request_processor/graph_status ->
-> ?runID=123456&returnType=DESCRIPTION&waitForStatus=FINISHED&waitTimeout=60000
```

```
http://localhost:8080/clover/request_processor/graph_status?
runID=123456&returnType=DESCRIPTION&waitForStatus=FINISHED&waitTimeout=60000
```

graph_kill操作

この操作をコールし、ジョブ実行を中断/強制終了します。この操作は下位互換性のためにgraph_killと呼ばれますが、ETLグラフ、ジョブフローまたはプロファイラ・ジョブを中断/強制終了できます。

パラメータ

表14.3 graph_killのパラメータ

パラメータ名	必須	デフォルト	説明
runID	はい	-	各グラフ実行のID
returnType	いいえ	STATUS	STATUS | STATUS_TEXT | DESCRIPTION
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

指定したグラフを強制終了しようとした後のグラフのステータス。これは、オプションのパラメータに応じて、数値コード、テキスト・コードまたは複雑な説明である場合があります。

例

```
http://localhost:8080/clover/request_processor/graph_kill?
runID=123456&returnType=DESCRIPTION
```

server_jobs操作

パラメータ

なし

戻り値

現在実行されているジョブのrunIDのリスト。

例

```
http://localhost:8080/clover/request_processor/server_jobs
```

sandbox_list操作

パラメータ

なし

戻り値

すべてのサンドボックスのテキストIDのリスト。次のバージョンでは、アクセス可能なもののみが戻されます。

例

```
http://localhost:8080/clover/request_processor/sandbox_list
```

sandbox_content操作

パラメータ

表14.4 sandbox_contentのパラメータ

パラメータ名	必須	デフォルト	説明
sandbox	はい	-	サンドボックスのテキストID
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

指定したサンドボックスのすべての要素のリスト。各要素は、サンドボックスのルートの相対ファイル・パスとして指定できます。

例

```
http://localhost:8080/clover/request_processor/sandbox_content?sandbox=mva
```

executions_history操作

パラメータ

表14.5 executions_historyのパラメータ

パラメータ名	必須	デフォルト	説明
sandbox	はい	-	サンドボックスのテキストID
from	いいえ		実行開始の日時の下限。この日時以降のレコードのみが戻されます。書式: "yyyy-MM-dd HH:mm" (URLエンコードする必要があります)。
to	いいえ		実行開始の日時の上限。この日時以前のレコードのみが戻されます。書式: "yyyy-MM-dd HH:mm" (URLエンコードする必要があります)。
stopFrom	いいえ		実行停止の日時の下限。この日時以降のレコードのみが戻されます。書式: "yyyy-MM-dd HH:mm" (URLエンコードする必要があります)。
stopTo	いいえ		実行停止の日時の上限。この日時以前のレコードのみが戻されます。書式: "yyyy-MM-dd HH:mm" (URLエンコードする必要があります)。
status	いいえ		現在の実行ステータス。指定したSTATUSを持つレコードのみが戻されます。意味のある値は、RUNNING | ABORTED | FINISHED_OK | ERRORです。
sandbox	いいえ		サンドボックス・コード。指定したサンドボックスのグラフのレコードのみが戻されます。
graphId	いいえ		指定したサンドボックス内で一意であるテキストID。サンドボックスのルートの相対ファイル・パス
orderBy	いいえ		リスト順序の属性。使用可能な値: id | graphId | finalStatus | startTime | stopTime。デフォルトでは、順序はありません。
orderDescend	いいえ	true	昇順または降順を指定するスイッチ。true (デフォルト)である場合、順序は降順です。
returnType	いいえ	ID	使用可能な値: IDs | DESCRIPTION | DESCRIPTION_XML
index	いいえ	0	レコード・セット全体で最初に戻されるレコードの索引。(開始元
records	いいえ	infinite	戻されるレコードの最大数。
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

フィルタ基準に応じた実行のリスト。

returnType==IDsである場合、runIDの簡単なリストを戻します(改行デリミタ付き)。

returnType==DESCRIPTIONの場合、選択した実行の現在のステータス、フェーズ、ノードおよびポートを表す複合レスポンスを戻します。

```
execution|[runID]|[status]|[username]|[sandbox]|[graphID]|[startedDatetime]|
[finishedDatetime]|
[clusterNode]|[graphVersion]
phase|[index]|[execTimeInMilis]
node|[nodeID]|[status]|[totalCpuTime]|[totalUserTime]|[cpuUsage]|[peakCpuUsage]|
[userUsage]|
[peakUserUsage]
port|[portType]|[index]|[avgBytes]|[avgRows]|[peakBytes]|[peakRows]|[totalBytes]|
[totalRows]
```

リクエストの例

```
http://localhost:8080/clover/request_processor/executions_history -> -> ?
from=&to=2008-09-16+16%3A40&status=&sandbox=def&graphID=&index=&records=&returnType=
 DESCRIPTION
```

```
http://localhost:8080/clover/request_processor/executions_history?
 from=&to=2008-09-16+16%3A40&status=&sandbox=def&graphID=&index=&records=&returnType=
 DESCRIPTION
```

DESCRIPTION（プレーン・テキスト）レスポンスの例

```
execution|13108|FINISHED_OK|clover|def|test.grf|2008-09-16 11:11:19|2008-09-16 11:11:58|
nodeA|2.4 phase|0|38733
node|DATA_GENERATOR1|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Output|0|0|0|0|0|130|10
node|TRASH0|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|5|0|130|10
node|SPEED_LIMITER0|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|0|0|130|10
port|Output|0|0|0|5|0|130|10
execution|13107|ABORTED|clover|def|test.grf|2008-09-16 11:11:19|2008-09-16 11:11:30
phase|0|11133
node|DATA_GENERATOR1|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Output|0|0|0|0|0|130|10
node|TRASH0|RUNNING|0|0|0.0|0.0|0.0|0.0 port|Input|0|5|0|5|0|52|4
node|SPEED_LIMITER0|RUNNING|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|0|0|130|10
port|Output|0|5|0|5|0|52|4
```

returnType==DESCRIPTION_XMLの場合、選択した1つ以上の実行を表す複合データ構造をXML書式で戻します。XMLレスポンスのXML書式を表すスキーマには、CloverETL ServerのURL (http://[host]:[port]/clover/schemas/ executions.xsd)でアクセスできます。

suspend操作

サーバーまたはサンドボックスを一時停止します(指定されている場合)。一時停止とは、一時停止されているサーバー/サンドボックスでグラフを実行できないことを意味します。

パラメータ

表14.6 suspendのパラメータ

パラメータ名	必須	デフォルト	説明
sandbox	いいえ	-	一時停止するサンドボックスのテキストID。指定しない場合、サーバー全体を一時停止します。

パラメータ名	必須	デフォルト	説明
atonce	いいえ		このパラメータがTRUEに設定されている場合、一時停止されているサーバー(またはサンドボックスのみ)から実行されているグラフが中断されます。それ以外の場合、一般的な方法で終了するまで実行できます。

戻り値

結果メッセージ

resume操作

パラメータ

表14.7 resumeのパラメータ

パラメータ名	必須	デフォルト	説明
sandbox	いいえ	-	再開するサンドボックスのテキストID。指定しない場合、サーバーは再開されます。
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

結果メッセージ

sandbox_create操作

この操作により、指定したサンドボックスが作成されます。これが"partitioned"または"local"タイプのサンドボックスである場合、"sandbox_add_location"操作によって場所も作成されます。

パラメータ

表14.8 sandbox_createのパラメータ

パラメータ名	必須	デフォルト	説明
sandbox	はい	-	作成するサンドボックスのテキストID。
path	いいえ	-	サーバーがスタンドアロン・モードで実行されている場合におけるサンドボックスのルートのパス。
type	いいえ	shared	サンドボックス・タイプ: shared | partitioned | local。スタンドアロン・サーバーの場合、デフォルトの"shared"が使用されるため、空のままである可能性があります。
createDirs	いいえ	true	サンドボックスのディレクトリ構造を作成するかどうかのスイッチ(クラスタ環境のスタンドアロン・サーバーまたは"shared"サンドボックスの場合のみ)。
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

結果メッセージ

sandbox_add_location操作

この操作により、指定したサンドボックスに場所が追加されます。partitionedまたはlocalタイプのサンドボックスに対してのみ使用可能です。

パラメータ

表14.9 sandbox_add_locationのパラメータ

パラメータ名	必須	デフォルト	説明
sandbox	はい	-	場所の追加先のサンドボックス。
nodeId	はい	-	場所属性 - この場所に直接アクセスできるノード。
path	はい	-	場所属性 - 指定したノード上の場所のルートへのパス。
location	いいえ	-	場所属性 - 場所記憶域ID。指定しない場合、新しい場所が生成されます。
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

結果メッセージ

sandbox_remove_location操作

この操作により、指定したサンドボックスから場所が削除されます。partitionedまたはlocalタイプのサンドボックスに対してのみ場所を関連付けることができます。

パラメータ

表14.10 sandbox_add_locationのパラメータ

パラメータ名	必須	デフォルト	説明
sandbox	はい	-	指定したサンドボックスから場所を削除します。
location	はい	-	場所記憶域ID。指定した場所が指定したサンドボックスにアタッチされていない場合、サンドボックスは変更されません。
verbose	いいえ	MESSAGE	MESSAGE | FULL - 可能性のあるエラー・メッセージの冗長性の程度。

戻り値

結果メッセージ

Cluster status

この操作により、クラスタのノードのリストが表示されます。

パラメータ

なし

戻り値

クラスタのノードのリスト。

第15章 JMX mBean

CloverETL ServerのJMX mBeanは、サーバーの内部ステータスの監視用として使用できるAPIです。

MBeanは、次の名前で登録されています。

```
com.cloveretl.server.api.jmx:name=cloverServerJmxMBean
```

JMX構成

アプリケーションのJMX MBeanには、デフォルトでは、JVMの外部からアクセスできません。これらにアクセスできるようにするには、アプリケーション・サーバー構成を変更する必要があります。

この項では、開発およびテスト用としてJMXコネクタを構成する方法について説明します。これにより、認証が無効になる可能性があります。本番デプロイメントの場合、認証を有効にする必要があります。これを実現する方法については、追加ドキュメント(http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html#auth)を参照してください。

構成および可能性のある問題:

- 「Apache TomcatでのJMXの構成方法」
- 「GlassfishでのJMXの構成方法」
- 「Websphere 7」
- 「可能性のある問題」

Apache TomcatでのJMXの構成方法

TomcatのJVMは、次の自明のパラメータを使用して実行する必要があります。

1. **-Dcom.sun.management.jmxremote=true**
2. **-Dcom.sun.management.jmxremote.port=8686**
3. **-Dcom.sun.management.jmxremote.ssl=false**
4. **-Dcom.sun.management.jmxremote.authenticate=false**
5. **-Djava.rmi.server.hostname=your.server.domain (necessary only for remote JMX connections)**

UNIXライクなOSの場合、環境変数CATALINA_OPTSを次のように設定します。

```
export CATALINA_OPTS="-Dcom.sun.management.jmxremote=true
                      -Dcom.sun.management.jmxremote.port=8686
                      -Dcom.sun.management.jmxremote.ssl=false
                      -Dcom.sun.management.jmxremote.authenticate=false
                      -Djava.rmi.server.hostname=your.server.domain.com"
```

ファイルTOMCAT_HOME/bin/setenv.sh (存在しない場合は作成できます)またはTOMCAT_HOME/bin/catalina.sh

Windowsの場合は扱いにくい可能性があるため、各パラメータを個別に設定する必要があります。

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote=true
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.port=8686
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.authenticate=false
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.ssl=false
set CATALINA_OPTS=%CATALINA_OPTS% -Djava.rmi.server.hostname=your.server.domain
```

ファイルTOMCAT_HOME/bin/setenv.bat（存在しない場合は作成できます）またはTOMCAT_HOME/bin/catalina.bat

これらの値とともに、URLを使用して

```
service:jmx:rmi:///jndi/rmi://localhost:8686/jmxrmi
```

JVMのJMXサーバーに接続できます。ユーザー/パスワードは必要ありません

GlassfishでのJMXの構成方法

Glasfishの管理コンソール(デフォルトでは、http://localhost:4848でユーザー/パスワードとしてadmin/adminadminを使用してアクセスできます)に移動します

セクションConfiguration→Admin Service→systemに移動し、次のように属性を設定します。

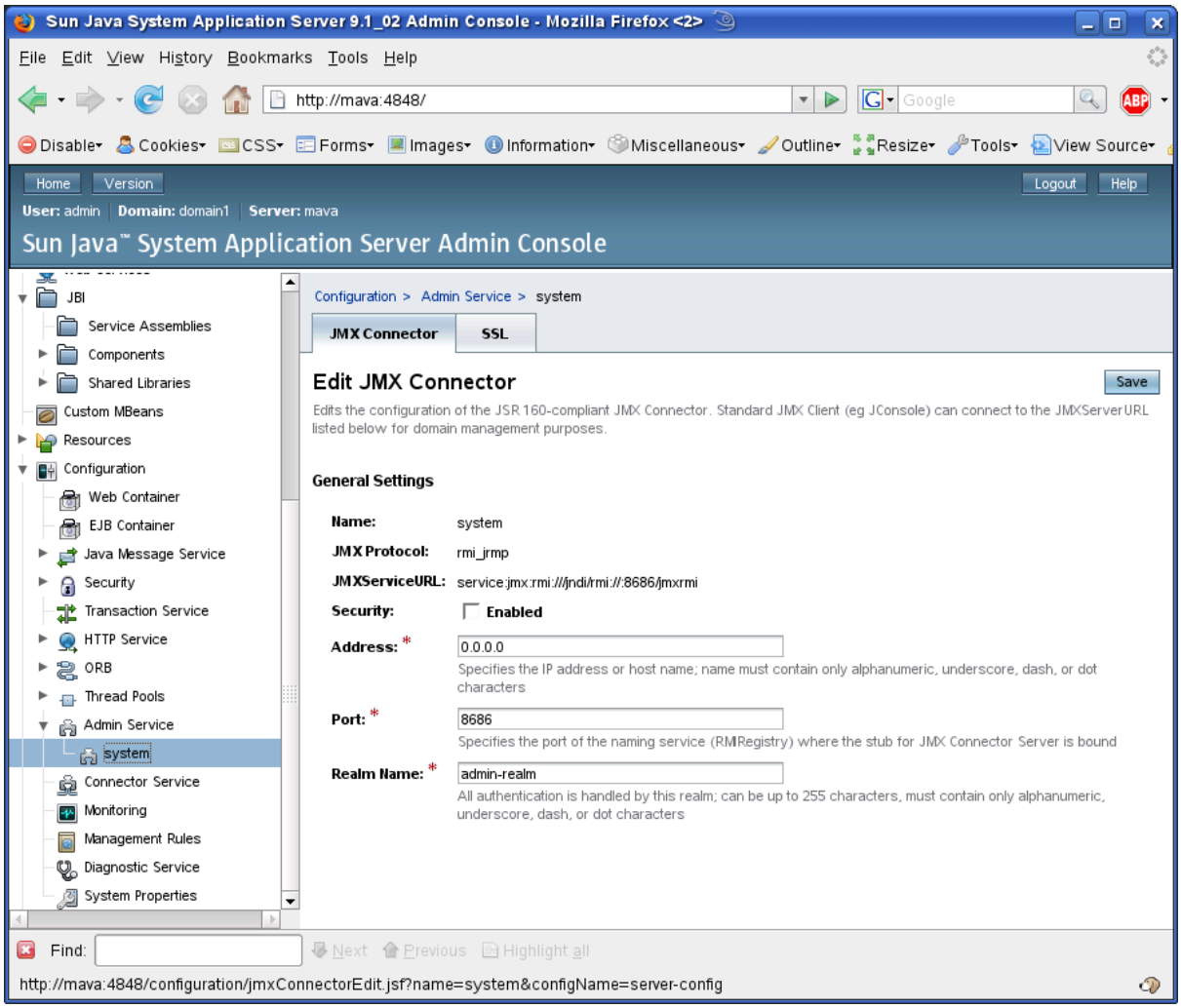


図15.1 Glassfish JMXコネクタ

これらの値とともに、URLを使用して

```
service:jmx:rmi:///jndi/rmi://localhost:8686/jmxrmi
```

JVMのJMXサーバーに接続できます。

ユーザー/パスワードとしてadmin/adminadminを使用します。(admin/adminadminはデフォルトのglassfish値です)

WebsphereでのJMXの構成方法

Websphereでは特別な構成は必要ありませんが、cloverのMBeanは、アプリケーション・サーバーの構成に依存する名前とともに登録されています。

```
com.cloveretl.server.api.jmx:cell=[cellName],name=cloverServerJmxMBean,node=[nodeName],
process=[instanceName]
```

```
server.api.jmx:cell=[cellName],name=cloverServerJmxMBean,node=[nodeName],process=
[instanceName]
```

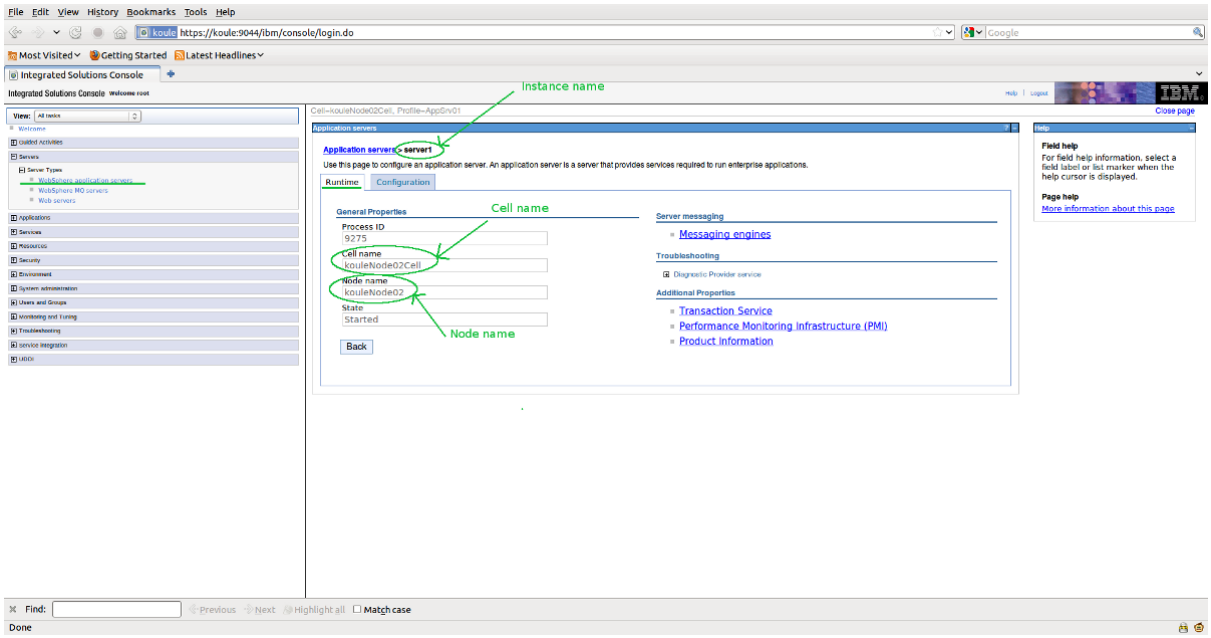


図15.2 Websphere構成

Websphere 7

JMXサーバーに接続するためのURLは、次のとおりです。

```
service:jmx:iiop://[host]:[port]/jndi/JMXConnector
```

hostは接続先のホスト名、portはRMIポート番号です。 デフォルトのWebsphereがインストールされている場合、1つのシステムにインストールされているサーバーの数、および接続先の特定のサーバーに応じて、JNDIポート番号は2809、2810、...のようになります。確実を期すには、次のような行がダンプされるので、ログを確認します

```
0000000a RMIConnectorC A  ADMC0026I: The RMI Connector is available at port 2810
```

Websphere7でのJMXの構成方法

Websphereでは特別な構成は必要ありませんが、cloverのMBeanは、アプリケーション・サーバーの構成に依存する名前とともに登録されています。

```
com.cloveretl.server.api.jmx:cell=[cellName],name=cloverServerJmxMBean,node=[nodeName],
 process=[instanceName]
```

```
server.api.jmx:cell=[cellName],name=cloverServerJmxMBean,node=[nodeName],process=
 [instanceName]
```

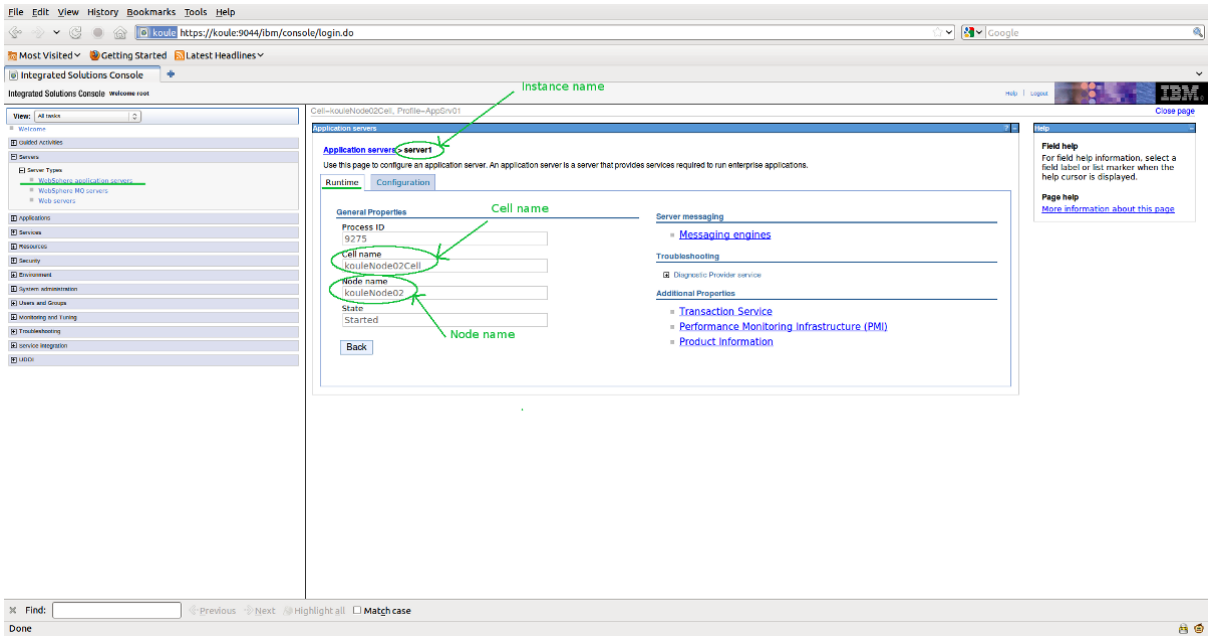


図15.3 Websphere7構成

JMXサーバーに接続するためのURLは、次のとおりです。

```
service:jmx:iiop://[host]:[port]/jndi/JMXConnector
```

hostは接続先のホスト名、portはRMIポート番号です。 デフォルトのWebsphereがインストールされている場合、1つのシステムにインストールされているサーバーの数、および接続先の特定のサーバーに応じて、JNDIポート番号は2809、2810、...のようになります。確実を期すには、次のような行がダンプされるので、ログを確認します

```
0000000a RMIConnectorC A   ADMC0026I: The RMI Connector is available at port 2810
```

また、クラスパスで、Websphereのホーム・ディレクトリのjarファイル:

/runtimes/com.ibm.ws.admin.client_7.0.0.jar
/runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar
/runtimes/com.ibm.ws.orb_7.0.0.jar
を設定する必要があります。

可能性のある問題

- デフォルトのJMX mBeanサーバーでは、トランスポート・プロトコルとしてRMIを使用します。場合によっては、ピアの1つでJavaバージョン1.6が使用されているときに

RMIがリモートで接続できない可能性があります。解決するのは簡単で、2つのシステム・プロパティ**-Djava.rmi.server.hostname=[hostname or IP address] -Djava.net.preferIPv4Stack=true**を設定するだけです。

操作

操作の詳細は、MBeanインタフェースのJavaDocを参照してください。

JMX API MBeanのJavaDocは、URL (http://[host]:[port]/[contextPath]/javadoc-jmx/index.html)で、実行されているCloverETL Serverインスタンスからアクセスできます。

第16章 SOAP WebService API

CloverETL Server SOAP Webサービスは、単純なHTTP APIのかわりに自動化を提供する高度なAPIです。ほとんどの(ただしすべてではない) HTTP API操作はSOAPインタフェースでも使用できますが、SOAP APIはサンドボックスや監視などを操作するための追加の操作を提供します。

SOAP APIサービスは、次のURLでアクセスできます。

```
http://[host]:[port]/clover/webservice
```

SOAP APIサービス・ディスクリプタは、次のURLでアクセスできます。

```
http://[host]:[port]/clover/webservice?wsdl
```

プロトコルHTTPは、Webサーバーの構成に基づいてセキュアなHTTPSに変更できます。

SOAP WSクライアント

公開されているサービスは、様々なプログラミング言語のライブラリで広くサポートされている最も一般的なバインディング・スタイル"document/literal"を使用して実装されています。

このAPIのクライアントを作成するには、WSDLドキュメント(前述のURLを参照)とともに、プログラミング言語および開発環境に応じてなんらかの開発ツールが必要です。

関連するすべてのクラスを含むWebServiceインタフェースのJavaDocは、URL (http://[host]:[port]/[contextPath]/javadoc-ws/index.html)で、実行されているCloverETL Serverインスタンスからアクセスできます。

WebサーバーでHTTPSコネクタが構成されている場合、クライアントは、Webサーバーの構成に応じてセキュリティ要件も満たす必要があります。(クライアントの信頼性および正しく構成されたキーストアなど)

SOAP WS API認証/認可

公開されているサービスはステートレスであるため、認証"sessionToken"をパラメータとして各操作に渡す必要があります。クライアントは、"login"操作をコールすることによって認証sessionTokenを取得できます。

第17章 起動サービス

起動サービスでは、変換グラフまたはジョブフローをWebサービスとして公開できます。起動サービスでは、CloverETL変換を公開して、他のアプリケーション用にまたはユーザーに直接、リクエスト-レスポンス・ベースのデータ・インタフェース(検索、複雑な参照など)を提供できます。

起動サービスの概要

起動サービスのアーキテクチャは比較的単純であり、Webブラウザを利用する複数層アプリケーションの基本設計に準拠しています。

たとえば、ユーザーが入力するユーザーフレンドリなフォームを作成し、処理のためにCloverETL Serverに送信できます。

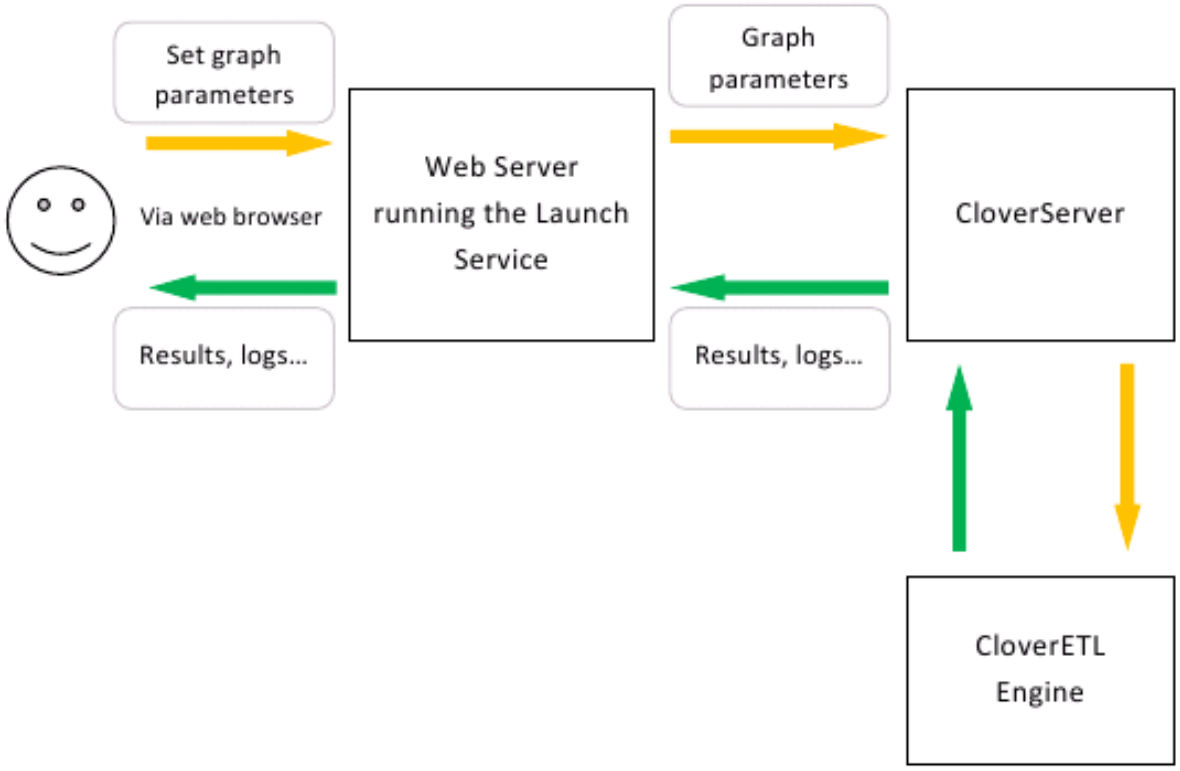


図17.1 Webアプリケーションのバックエンドとしての起動サービスおよびCloverETL Server

起動サービスでのグラフのデプロイ

起動サービスとして公開するためのグラフを準備するには、設計プロセス中に次のことを念頭に置きます。

1. パラメータ化コールを作成するためにグラフ/ジョブフロー・リスナーを定義できます。パラメータはディクショナリ・エントリとしてグラフに渡されるため、ディクショナリをパラメータ(ファイル名、検索用語など)の入力/出力として使用するようにグラフを設計します。

2. グラフは、ディクショナリ・エントリへのパラメータの構成およびバインドを提供する起動サービス・セクションで公開する必要があります。

起動サービスのETLグラフ/ジョブフローでのディクショナリの使用

サービスとして公開されるグラフまたはジョブフローは、通常はコール元がリクエスト・データ(パラメータまたはデータ)を送信し、変換がそれを処理して結果を返すことを意味します。

起動サービス定義では、サービスのパラメータをディクショナリ・エントリにバインドできます。これらは変換で事前定義する必要があります。

ディクショナリは、実行中の変換とコール元の間のキー/値の一時データ・インタフェースです。通常、ディクショナリは実行された変換との間でのパラメータの受渡しに使用されますが、これに限定されません。

ディクショナリの詳細は、CloverETL Designerのユーザーズ・ガイドのディクショナリに関する項を参照してください。

CloverETL ServerのWeb GUIでのジョブの構成

各起動サービス構成は、名前、ユーザーおよびグループ制限によって識別されます。ユーザーまたはグループ制限が異なるかぎり、同じ名前で複数の構成を作成できます。

同じ起動構成(名前)を使用していても、ユーザー制限を使用して、異なるユーザーに異なるジョブを起動できます。たとえば、開発者はジョブのデバッグ・バージョンを使用でき、エンド・ユーザーは本番ジョブを使用できます。また、ユーザー制限を使用して、特定のユーザーが起動構成をまったく実行できないようにすることもできます。

同様に、グループ制限を使用して、ユーザーのグループ・メンバーシップに基づいてジョブを区別することもできます。

複数の構成が現在のユーザー/グループおよび構成名と一致する場合、最も適切な構成が選択されます。(グループ名よりユーザー名の方が優先されます)

新規起動構成の追加

起動サービスボタンを使用して、新規起動サービスを作成します。名前はサービスの識別子であり、サービスURLで使用されます。次に、サンドボックスと、公開する変換グラフまたはジョブフローを選択します。

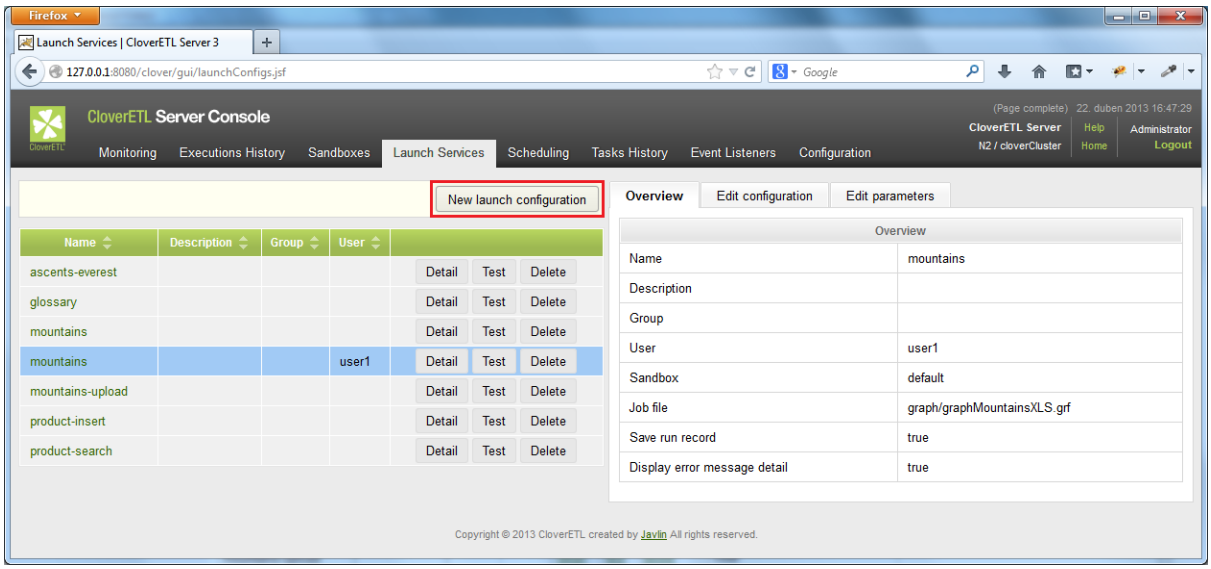


図17.2 起動サービス・セクション

新規起動サービスを作成した後で、次のような追加属性を設定できます。

1. ユーザーおよびグループ・アクセス制限および追加構成オプション(構成の編集)

2. ディクショナリ・エントリへの起動サービス・パラメータのバインド(パラメータの編集)

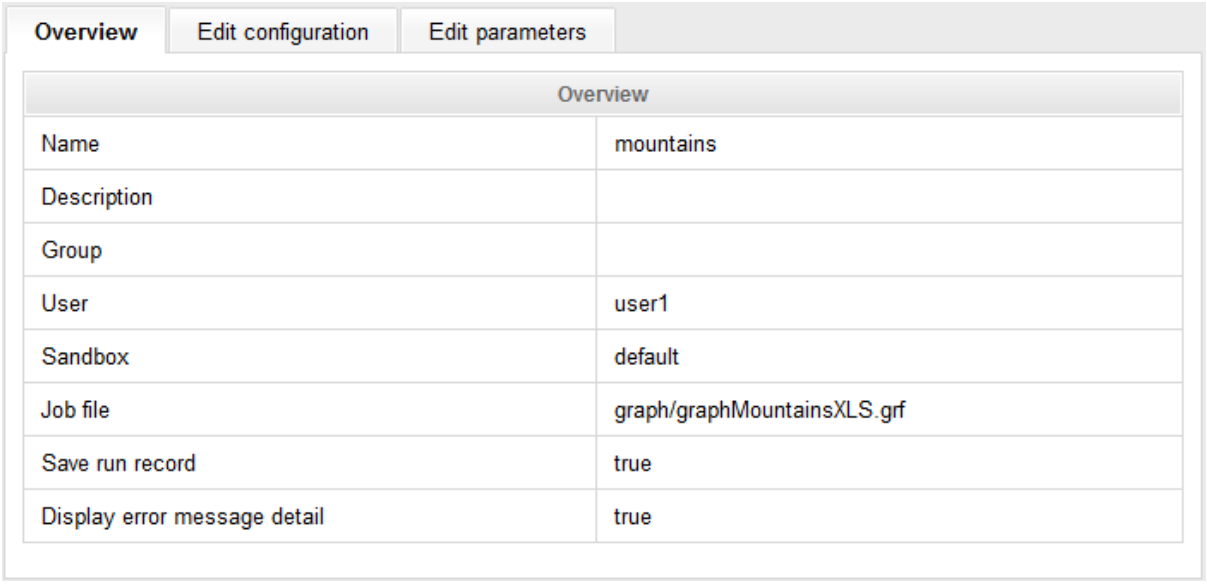


図17.3 概要タブ

概要タブには、起動構成に関する基本的な詳細が表示されます。これらは、構成の編集タブで変更できます。

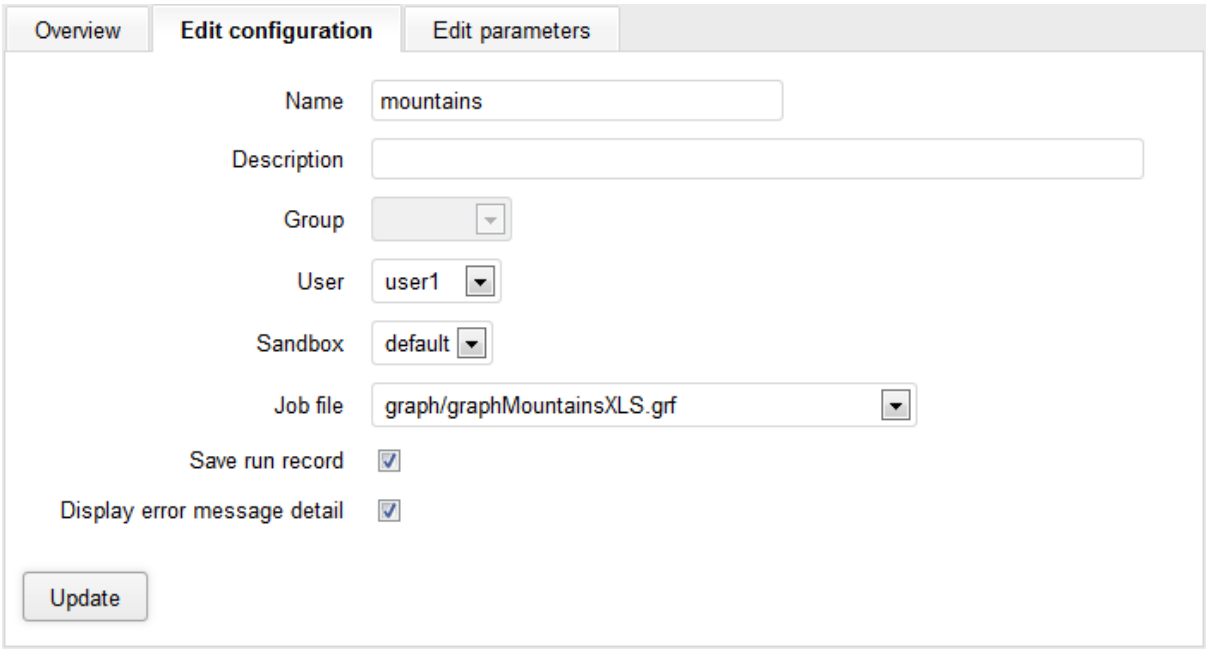


図17.4 構成の編集タブ

構成の編集:

・ 名前 - Webから構成にアクセスする際の名前(識別子)。

- 説明 – 構成の説明。
- グループ – 構成を特定のユーザーのグループに制限します。
- ユーザー – 構成を特定のユーザーに制限します。
- サンドボックス – 構成を起動するCloverETL Sandbox。
- ジョブ・ファイル – 実行するジョブを選択します。
- 実行レコードの保存 – 選択すると、起動構成に関する詳細が実行履歴に格納されます。パフォーマンスを向上させる必要がある場合は、選択を解除します。実行レコードを格納すると、高頻度コールのレスポンス時間が悪化します。
- エラー・メッセージ詳細の表示 – 起動が失敗した場合に詳細なメッセージを表示する場合は、これを選択します。

パラメータの編集タブを使用して、起動構成のパラメータ・マッピングを構成できます。起動リクエストで送信された値に基づいてパラメータ値を正しく割り当てるには、マッピングが必要です。

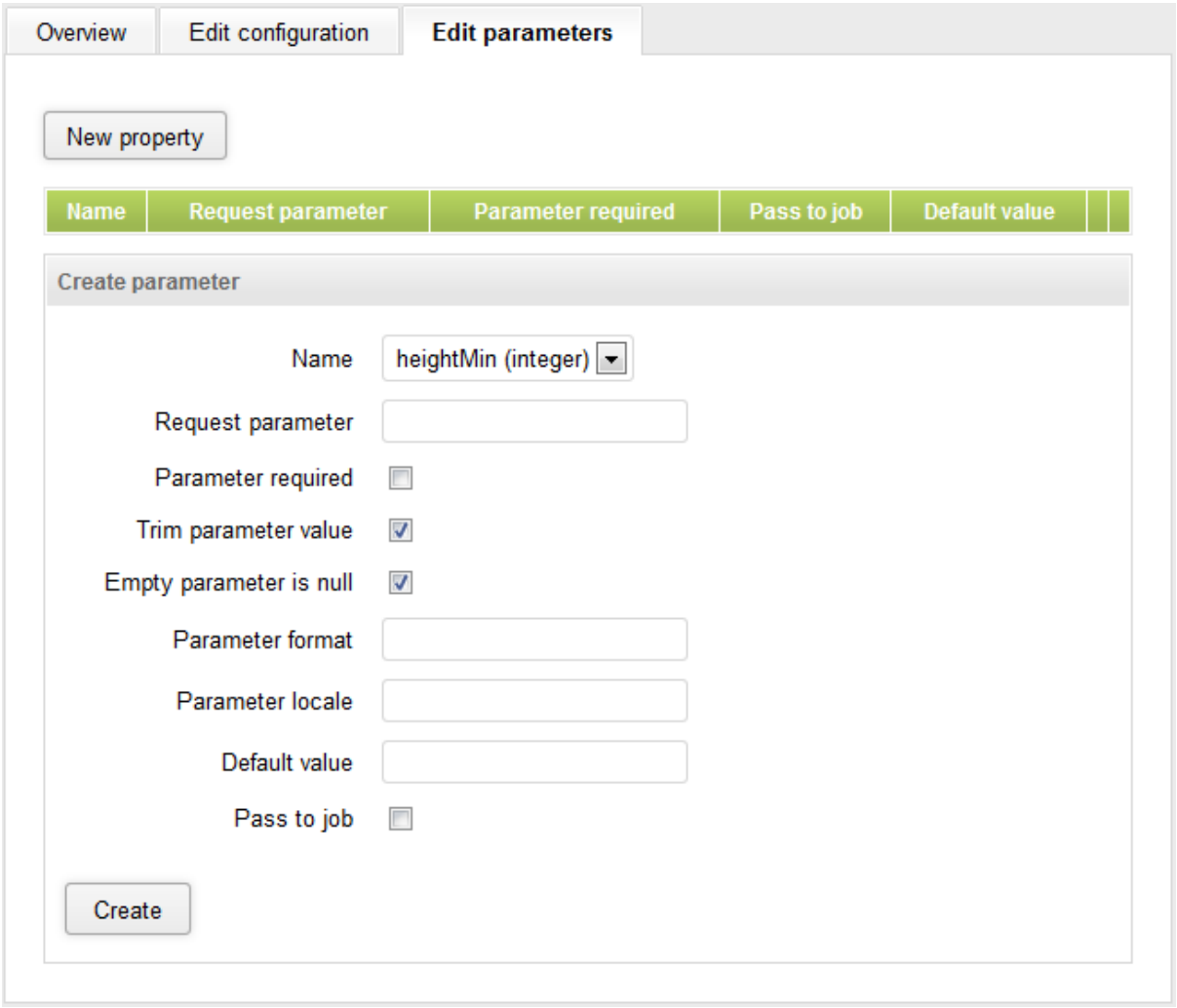


図17.5 新規パラメータの作成

新規パラメータ・バインドを追加するには、新規プロパティ・ボタンをクリックします。必要になるたびに、ジョブで定義されたグラフ/ジョブフロー・リスナー・プロパティをここで作成する必要があります。

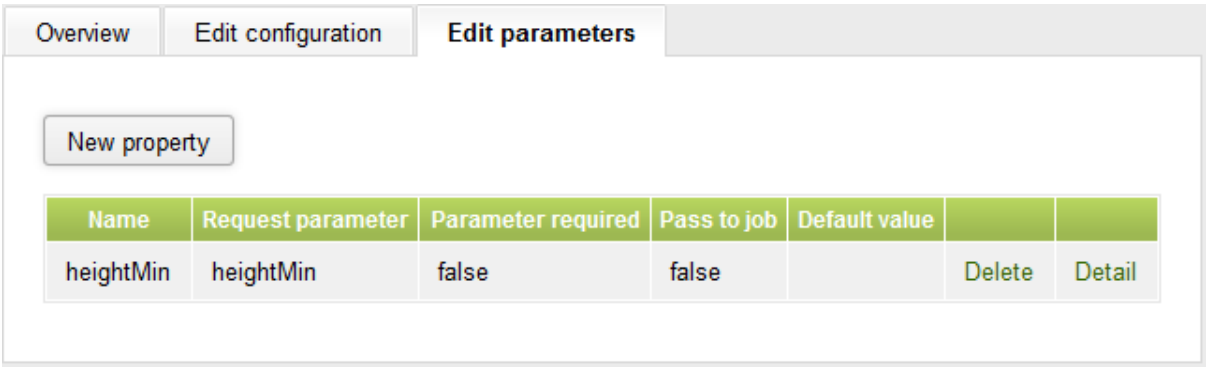


図17.6 パラメータの編集タブ

プロパティごとに次のフィールドを設定できます。

- 名前 – バインドするグラフ/ジョブフローで定義されたディクショナリ・エントリの名前。
- リクエスト・パラメータ – 公開されたサービスでこのプロパティが表示される際の名前。この名前は名前とは異なる名前にできます。
- 必須パラメータ – 選択すると、パラメータが必須になり、パラメータが省略されるとエラーがレポートされます。
- ジョブに渡す – 選択すると、プロパティ値は、パラメータ(${ParameterName}で、ParameterNameは名前と同じ)としてもジョブに渡されます。これにより、ディクショナリをサポートする場所だけでなく、ジョブ定義の任意の場所でパラメータを使用できます。ただし、パラメータはジョブ初期化中に評価されます。このため、このようなジョブはプールできず、サービスへの高頻度の反復コールのパフォーマンスが低下します。この場合、かわりにディクショナリを使用してプーリングを可能にするように変換を再設計することを検討してください。
- デフォルト値 – パラメータが起動リクエストで省略されたときに適用されるデフォルト値。

起動サービスへのデータの送信

起動リクエストは、HTTP GETまたはPOSTメソッドを介して送信できます。起動リクエストは、ジョブに渡す必要があるすべてのパラメータの値が含まれる単なるURLです。リクエストURLは、複数の部分で構成されています。

(ログイン画面からアクセス可能な起動サービス・テスト・ページを使用して、起動サービスの駆動をテストします。)

```
[Clover Context]/launch/[Configuration name]?[Parameters]
```

- [Clover Context]は、CloverETL Serverが実行されているコンテキストに対するURLです。通常、これは、CloverETL Serverに対する完全なURLです(たとえば、CloverETL Demo Serverの場合、これはhttp://server-demo.cloveretl.com:8080/cloverのようになります)。
- [Configuration name]は、構成の作成時に指定された起動構成の名前です。この例では、これは"mountains"に設定されます(大/小文字は区別されます)。
- [Parameters]は、構成が問合せ文字列として必要とするパラメータのリストです。これは、"&"文字で分離された名前=値ペアのURLエンコードされた[RFC 1738]リストです。

前述の設定に基づいて、mountainsを使用したこの例における起動リクエストの完全なURLは、http://server-demo.cloveretl.com:8080/clover/launch/NewMountains?heightMin=4000のようになります。前述のリクエストでは、heightMinプロパティの値は4000に設定されます。

グラフ実行の結果

ジョブが終了すると、結果がHTTPレスポンスのコンテンツとしてHTTPクライアントに戻されます。

出力パラメータはジョブのディクショナリとして定義されます。"Output"としてマークされたすべてのディクショナリ・エントリはレスポンスの一部です。

出力パラメータの数に応じて、次の出力がHTTPクライアントに送信されます。

- 出力パラメータなし - 概要ページのみが戻されます。ページには、ジョブが開始された時間、終了した時間、ユーザー名などの詳細が表示されます。概要ページの書式はカスタマイズできません。
- 1つの出力パラメータ - この場合、ディクショナリ内のプロパティ・タイプによってMIMEコンテンツ・タイプが定義されているHTTPレスポンスの本文として出力がクライアントに送信されます。
- 複数の出力パラメータ - この場合、各出力パラメータがマルチパートHTTPレスポンスの一部としてHTTPクライアントに送信されます。レスポンスのコンテンツ・タイプは、HTTPクライアントに応じてmultipart/relatedまたはmultipart/x-mixed-replaceになります(クライアントは完全に自動検出されます)。multipart/relatedタイプはMicrosoft Internet Explorerに基づくブラウザに対して使用され、multipart/x-mixed-replaceはGeckoまたはWebkitに基づくブラウザに送信されます。

起動リクエストは、CloverETL Server構成内のlaunch.log.dirプロパティによって指定されているディレクトリ内のログ・ファイルに記録されます。起動構成ごとに、[Configuration name]#[Launch ID].logという名前のログ・ファイルが1つ作成されます。このファイルでは、起動リクエストごとに、次のタブ区切りフィールドを持つ1行のみが含まれます。

(プロパティlaunch.log.dirが指定されていない場合、ログ・ファイルは一時ディレクトリ[java.io.tmpdir]/cloverlog/launch内に作成されます。ここで、"java.io.tmpdir"はシステム・プロパティです)

- 起動開始時間
- 起動終了時間
- ログイン・ユーザー名
- 実行ID
- 実行ステータス: FINISHED_OK、ERRORまたはABORTED
- クライアントのIPアドレス
- HTTPクライアントのユーザー・エージェント
- 起動サービスに渡される問合せ文字列(現在の起動のパラメータの完全なリスト)

構成が有効でない場合、同じ起動詳細が同じディレクトリ内の_no_launch_config.logファイルに保存されます。未認証リクエストはすべて同じファイルに保存されます。

第18章 構成

デフォルトのインストール(構成なし)をお薦めするのは評価を目的とする場合のみです。本番使用の場合は、専用データベースを構成し、通知を送信するようにSMTPサーバーを正しく構成することをお薦めします。

構成ソースとその優先度

構成プロパティには複数のソースがあります。プロパティが設定されていない場合、アプリケーションのデフォルトが使用されます。

警告: 次に示すソースを結合しないでください。構成が混乱し、メンテナンスが困難になります。

コンテキスト・パラメータ(Apache Tomcatで使用可能)

アプリケーション・サーバーによっては、WARファイルを変更せずにコンテキスト・パラメータを設定できます。この方法による構成が可能で推奨されるのは、Tomcatの場合です。

Apache Tomcatは一部の環境で一部のコンテキスト・パラメータを無視することがあるため、この方法はお薦めしません。ほとんどの場合、プロパティ・ファイルを使用するのが便利ではるかに信頼性の高い方法です。

Apache Tomcatの例

Tomcatでは、コンテキスト構成ファイルでコンテキスト・パラメータを指定できます。CloverETL Server Webアプリケーションのデプロイメント後に自動的に**[tomcat_home]/conf/Catalina/localhost/clover.xml**が作成されます。

次の要素を追加することにより、プロパティを指定できます。

```
<Parameter name="[propertyName]" value="[propertyValue]" override="false" />
```

環境変数

接頭辞**clover.**を使用して環境変数を設定します(**clover.config.file**)。

一部のオペレーティング・システムではドット文字を使用できないことがあるため、ドット(_)のかわりに下線(.)を使用することもできます。このため、**clover_config_file**も有効です。

システム・プロパティ

接頭辞**clover.**を使用してシステム・プロパティを設定します(**clover.config.file**)。

また、ドット(.)のかわりに下線(_)を使用することもできるため、**clover_config_file**も有効です。

デフォルトの場所のプロパティ・ファイル

ソースは、共通プロパティ・ファイル(キーと値のペアを持つテキスト・ファイル)です。

```
[property-key]=[property-value]
```

デフォルトでは、CloverETLでは、構成ファイル**[workingDir]/cloverServer.properties**の検索が試行されます。

指定した場所のプロパティ・ファイル

前述の場合と同じですが、プロパティ・ファイルはデフォルトの場所からロードされません。これは、その場所が環境変数またはシステム・プロパティ**clover_config_file**または**clover.config.file**によって指定されるためです。アプリケーション・サーバーでコンテキスト・パラメータを設定できない場合は、この構成方法をお薦めします。

web.xml内のコンテキスト・パラメータの変更

clover.warを解凍し、ファイル**WEB-INF/web.xml**を変更し、次のコードを追加します。

```
<context-param>
<param-name>[property-name]</param-name>
<param-value>[property-value]</param-value>
</context-param>
```

この方法はお薦めしませんが、前述のいずれの方法も不可能である場合は役立つ場合があります。

構成ソースの優先度

構成ソースには、次の優先度があります。

1. コンテキスト・パラメータ(アプリケーション・サーバー内に指定されるか、**web.xml**に直接指定されます)
2. CSによってこの順序で検索が試行される外部構成ファイル(これらの1つのみがロードされます):
   - コンテキスト・パラメータ**config.file**によって指定されるパス
   - システム・プロパティ**clover_config_file**または**clover.config.file**によって指定されるパス
   - 環境変数**clover_config_file**または**clover.config.file**によって指定されるパス
   - デフォルトの場所(**[workingDir]/cloverServer.properties**)
3. システム・プロパティ
4. 環境変数
5. デフォルト値

DB接続構成の例

スタンドアロン・デプロイメント(非クラスタ化)では、組込みApache Derby DBがデフォルトで使用され、評価の目的には十分であるため、DB接続の構成はオプションです。ただし、本番デプロイメントの場合は、外部DB接続の構成をお薦めします。一般的なJDBC DB接続属性(URL、ユーザー名、パスワード)またはDB DataSourceのJNDIの場所を指定することもできます。

クラスタ・デプロイメントでは、クラスタ内の少なくとも1つのノードにDB接続が構成されている必要があります。その他のノードには(同じDBへの)直接接続があるか、永続的操作のプロキシとして別のノードを使用する場合がありますが、スケジューラは直接接続されたノードでのみアクティブです。この機能の詳細は、クラスタリングの項を参照してください。この項では、直接DB接続構成についてのみ説明します。

DB構成とその変更は、次のようになります。

組込みApache Derby

Apache Derbyの組込みDBは、デフォルトのCloverETL Serverインストールとともに使用されます。デフォルトでは、データの永続性のために作業ディレクトリが記憶域ディレクトリとして使用されます。この場合、一部のシステムでは問題が発生する可能性があります。Derby DBへの接続に関して問題が発生した場合は、外部DBへの接続を構成するか、少なくともDerbyのホーム・ディレクトリを指定することをお薦めします。

システム・プロパティ**derby.system.home**を設定し、アプリケーション・サーバーからアクセスできるパスを設定します。このシステム・プロパティは、次のJVM実行パラメータによって指定できます。

-Dderby.system.home=[derby_DB_files_root]

構成用のプロパティ・ファイルを使用する場合、**jdbc.driverClassName**, **jdbc.url**, **jdbc.username**, **jdbc.password**, **jdbc.dialect**. という各パラメータを指定します。次はその例です。

```
jdbc.driverClassName=org.apache.derby.jdbc.EmbeddedDriver
jdbc.url=jdbc:derby:databases/cloverDb;create=true
jdbc.username=
jdbc.password=
jdbc.dialect=org.hibernate.dialect.DerbyDialect
```

jdbc.urlパラメータに注目してください。**databases/cloverDb**の部分は、DBデータのサブディレクトリを意味します。このサブディレクトリは、**derby.system.home**として設定されているディレクトリ(または、**derby.system.home**が設定されていない場合は作業ディレクトリ)内に作成されます。値**databases/cloverDb**はデフォルト値であり、変更できます。

MySQL

CloverETL Serverには、MySql 5.xが必要です

構成用のプロパティ・ファイルを使用する場合、**jdbc.driverClassName**, **jdbc.url**, **jdbc.username**, **jdbc.password**, **jdbc.dialect**. という各パラメータを指定します。次はその例です。

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://127.0.0.1:3306/clover?useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=
jdbc.dialect=org.hibernate.dialect.MySQLDialect
```

3.0 JDBCドライバはCloverETL ServerのWebアーカイブに含まれないため、アプリケーション・サーバーのクラスパスに追加する必要があります。

次のように、正しいキャラクタ・セットを使用してDBを作成します。

```
CREATE DATABASE IF NOT EXISTS clover DEFAULT CHARACTER SET 'utf8';
```

DB2

Linux/Windows上のDB2

構成用のプロパティ・ファイルを使用する場合、**jdbc.driverClassName, jdbc.url, jdbc.username, jdbc.password, jdbc.dialect.** という各パラメータを指定します。次はその例です。

```
jdbc.driverClassName=com.ibm.db2.jcc.DB2Driver
jdbc.url= jdbc:db2://localhost:50000/clover
jdbc.username=usr
jdbc.password=pwd
jdbc.dialect=org.hibernate.dialect.DB2Dialect
```

可能性のある問題

不正なページ・サイズ

データベースcloverは、適切な**PAGESIZE**を使用して作成する必要があります。DB2には、このプロパティに使用可能な値として4096、8192、16384または32768という複数の値があります。

CloverETL Serverは、**PAGESIZE**が16384または32768に設定された状態でDBを処理する必要があります。**PAGESIZE**値が正しく設定されていない場合、CloverETL Serverの起動に失敗した後ログ・ファイルにエラー・メッセージが表示されます。

```
ERROR:
DB2 SQL Error: SQLCODE=-286, SQLSTATE=42727, SQLERRMC=16384;
ROOT, DRIVER=3.50.152
```

SQLERRMCには、**PAGESIZE**に適した値が含まれます。

次のように、正しい**PAGESIZE**を使用してデータベースを作成できます。

```
CREATE DB clover PAGESIZE 32768;
```

表が再編成保留状態

ALTER TABLEコマンドの後、一部の表が"再編成保留状態"にある場合があります。この動作は、DB2に固有のものです。ALTER TABLE DDLコマンドが実行されるのは、新規CloverETL Serverバージョンの初回起動時のみです。

この問題のエラー・メッセージは、次のようになります。

```
Operation not allowed for reason code "7" on table "DB2INST2.RUN_RECORD"..
SQLCODE=-668, SQLSTATE=57016
```

または、次のようになります

```
DB2 SQL Error: SQLCODE=-668, SQLSTATE=57016, SQLERRMC=7;DB2INST2.RUN_RECORD,
 DRIVER=3.50.152
```

この場合、"RUN_RECORD"は"再編成保留状態"にある表名であり、"DB2INST2"はDBインスタンス名です。

これを解決するには、DB2コンソールに移動し、コマンドを(表run_recordに対して)実行します。

```
reorg table run_record
```

DB2コンソールの出力は、次のようになります。

```
db2 => connect to clover1
Database Connection Information

Database server        = DB2/LINUX 9.7.0
SQL authorization ID   = DB2INST2
Local database alias   = CLOVER1

db2 => reorg table run_record
DB20000I  The REORG command completed successfully.
db2 => disconnect clover1
DB20000I  The SQL DISCONNECT command completed successfully.
```

"clover1"はDB名です

DB列の長さを切り捨てるALTER TABLEをDB2で使用不可。

この問題はDB2の構成に依存し、一部のAS400でのみ発生することがこれまでに判明しています。CloverETL Serverでは、アプリケーションのアップグレード後の初回インストール時に一連のDPパッチが適用されます。これらのパッチの一部により、テキスト列の長さを切り捨てる列変更が適用される場合があります。これらの変更によってデータが切り捨てられることはありません。ただし、DB2では、一部のデータが切り捨てられる可能性があるため、この処理は許可されません。DB2では、空のDB表内でもこれらの変更は拒否されます。解決策としては、データ切捨てに関するDB2の警告を無効にし、パッチを適用するCloverETL Serverを再起動してから、DB2の警告を再度有効にします。

AS/400上のDB2

AS/400での接続は若干異なる可能性があります。

構成用のプロパティ・ファイルを使用する場合、**jdbc.driverClassName, jdbc.url, jdbc.username, jdbc.password, jdbc.dialect.** という各パラメータを指定します。次はその例です。

```
jdbc.driverClassName=com.ibm.as400.access.AS400JDBCDriver
jdbc.username=javlin
jdbc.password=clover
jdbc.url=jdbc:as400://host/cloversrv;libraries=cloversrv;date format=iso
jdbc.dialect=org.hibernate.dialect.DB2400Dialect
```

jdbc.usernameおよび**jdbc.password**に対してはOSユーザーの資格証明を使用します。

前述の**jdbc.url**の**cloversrv**は、DBスキーマの名前です。

スキーマはAS/400コンソールで作成できます。

- コマンド**STRSQL**を実行します(SQLコンソール)
- **CREATE COLLECTION cloversrv IN ASP 1**を実行します
- **cloversrv**はDBスキーマの名前であり、最大で10文字にすることができます

アプリケーション・サーバーのクラスパス内に正しいJDBCドライバが含まれる必要があります。

サーバー上の**/QIBM/ProdData/Java400**にあるJDBCドライバ**jt400ntv.jar**を使用します。

jt400ntv.jar JDBCドライバを使用します。

TomcatのクラスパスにはJDBCドライバとともにjarを必ず追加してください。

Oracle

構成用のプロパティ・ファイルを使用する場合、**jdbc.driverClassName, jdbc.url, jdbc.username, jdbc.password, jdbc.dialect.** という各パラメータを指定します。次はその例です。

```
jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@host:1521:db
jdbc.username=user
jdbc.password=pass
jdbc.dialect=org.hibernate.dialect.Oracle9Dialect
```

アプリケーション・サーバーのクラスパスにはJDBCドライバとともにjarを必ず追加してください。

CloverETL Serverバージョン1.2.1以降、言語**org.hibernate.dialect.Oracle10gDialect**は使用できません。かわりに**org.hibernate.dialect.Oracle9Dialect**を使用してください。

CloverETL Serverで使用されるスキーマに付与する必要がある権限は、次のとおりです。

```
CONNECT CREATE SESSION CREATE/ALTER/DROP TABLE CREATE/ALTER/DROP SEQUENCE

QUOTA UNLIMITED ON <user_tablespace>; QUOTA UNLIMITED ON <temp_tablespace>;
```

MS SQL

Ms SQLには、DBサーバーの構成が必要です。

- TCP/IP接続の許可:
- ツールSQL Server Configuration Managerを実行します
- クライアント プロトコルに移動します
- TCP/IP (デフォルトのポートは1433です)をオンにします
- ツールSQL Server Management Studioを実行します
- データベースに移動し、DB cloverを作成します
- セキュリティ/ログインに移動し、ユーザーを作成し、このユーザーをDB cloverの所有者として割り当てます
- セキュリティに移動し、SQL Server 認証モードと Windows 認証モードを選択します

構成用のプロパティ・ファイルを使用する場合、**jdbc.driverClassName, jdbc.url, jdbc.username, jdbc.password, jdbc.dialect.** という各パラメータを指定します。次はその例です。

```
jdbc.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
jdbc.url=jdbc:sqlserver://localhost:1433;databaseName=clover
jdbc.username=user
jdbc.password=pass
jdbc.dialect=org.hibernate.dialect.SybaseDialect
```

TomcatのクラスパスにはJDBCドライバとともにjarを必ず追加してください。

Postgre SQL

構成用のプロパティ・ファイルを使用する場合、**jdbc.driverClassName, jdbc.url, jdbc.username, jdbc.password, jdbc.dialect.** という各パラメータを指定します。次はその例です。

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://localhost/clover?charSet=UTF-8
jdbc.username=postgres
jdbc.password=
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
```

TomcatのクラスパスにはJDBCドライバとともにjarを必ず追加してください。

JNDI DB DataSource

サーバーは、アプリケーション・サーバーまたはコンテナ内で構成されるJNDI DB DataSourceに接続できます。ただし、設定が必要なCloverETLパラメータがいくつか存在します。これらを設定しない場合、予期しない動作が行われる可能性があります。

```
datasource.type=JNDI
# type of datasource; must be set, because default value is JDBC datasource.jndiName=
# JNDI location of DB DataSource; default value is java:comp/env/jdbc/clover_server
# jdbc.dialect=
# Set dialect according to DB which DataSource is connected to.The same dialect as in
 sections above.
#
```

前述のパラメータは、(プロパティ・ファイルまたはTomcatコンテキスト・ファイル内の)他のパラメータと同じ方法で設定できます

Apache TomcatでのDataSource構成の例を示します。次のコードをコンテキスト・ファイルに追加します。

```
<Resource name="jdbc/clover_server" auth="Container"
type="javax.sql.DataSource" driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://192.168.1.100:3306/clover?useUnicode=true&amp;characterEncoding=utf8"
username="root" password="" maxActive="20" maxIdle="10"
maxWait="-1"/>
```

注記

コンテキスト・ファイルで特殊文字を入力する場合は、XMLエンティティとして指定する必要があります。アンパサンド"&"を表す"&"などです。

プロパティのリスト

表18.1 一般構成

キー	説明	デフォルト
config.file	CloverETL Server構成ファイルの場所	[working_dir]/cloverServer.properties
license.file	CloverETL Serverライセンス・ファイル(license.dat)の場所	
engine.config.file	CloverETLエンジン構成のプロパティ・ファイルの場所	CloverETLとともに圧縮されたプロパティ・ファイル
private.properties	CloverETL Serverコードによってのみ使用されるサーバー・プロパティのリスト。このため、これらのプロパティは、ServerFacadeの外部からはアクセスできません。デフォルトでは、パスワードが含まれる可能性があるプロパティがすべてリスト内に存在します。これは基本的に、これらの値がWeb GUIユーザーには表示されないことを意味します。各値は単一の星印"*"に置き換えられています。このリストを変更すると、一部のサーバーAPIで予期しない動作が行われる可能性があります。	jdbc.password、executor.password、security.ldap.password、clover.smtp.password
engine.plugins.src	このプロパティには、CloverETLエンジンの追加プラグインのソースに対する絶対パスがある場合があります。これらのプラグインは、WARに圧縮されているプラグインの代替ではありません。ソースは、ディレクトリでもzipファイルでもかまいません。ディレクトリとzipファイルには両方とも、各プラグインのサブディレクトリが含まれる必要があります。ディレクトリまたはZIPファイルの変更が適用されるのは、サーバーの再起動時のみです。詳細は、拡張性 - エンジンのプラグインの項を参照してください。	空
datasource.type	CloverETL ServerでJNDIデータソースを使用してDBに接続する必要がある場合、これを明示的にJNDIに設定します。この場合、パラメータ"datasource.jndiName"および"jdbc.dialect"を正しく設定する必要があります。使用可能な値: JNDI | JDBC	JDBC
datasource.jndiName	DB DataSourceのJNDIの場所。これが適用されるのは、"datasource.type"が"JNDI"に設定されている場合のみです。	java:comp/env/jdbc/clover_server
jdbc.driverClassName	JDBCドライバ名のクラス名	
jdbc.url	データを格納するためにCloverETL Serverによって使用されるJDBCのURL	
jdbc.username	JDBCデータベースのユーザー名	
jdbc.password	JDBCデータベースのユーザー名	
jdbc.dialect	ORMで使用するHibernate言語	

キー	説明	デフォルト
quartz.driver DelegateClass	Quartz用のSQL言語。値は、"jdbc.dialect"プロパティ値から自動的に導出されます。	
security_enabled	true | false。FALSEに設定された場合、認証は必要なく、任意のユーザーに管理権限が付与されます。	true
security.session. validity	セッションの有効期間(ミリ秒)。ログインしたユーザー/クライアントのリクエストが検出された場合は、有効期間が自動的に延長されます。	14400000
security.default_ domain	すべての新規ユーザーが含まれるドメイン。データベース内のユーザーのレコードに格納されます。"clover"をホワイトレーベル化する必要がないかぎり、変更しないでください。	clover
security.basic_ authentication. features_list	HTTPを使用してアクセスでき、HTTP基本認証によって保護する必要がある機能のセミコロン区切りのリスト。各機能は、サーブレット・パスによって指定します。	/request_processor;/ simpleHttpApi;/ launch;/launchIt;/ downloadStorage;/ downloadFile;/ uploadSandboxFile;/ downloadLog
security.basic_ authentication.realm	HTTP基本認証のレルム文字列。	CloverETL Server
security.digest_ authentication. features_list	HTTPを使用してアクセスでき、HTTPダイジェスト認証によって保護する必要がある機能のセミコロン区切りのリスト。各機能は、サーブレット・パスによって指定します。HTTPダイジェスト認証はバージョン3.1に追加された機能であることに注意してください。古いCloverETL Serverディストリビューションをアップグレードした場合、アップグレードの前に作成されたユーザーは、パスワードをリセットするまではHTTPダイジェスト認証を使用できません。このため、パスワードをリセット(または管理者がかわりにパスワードをリセット)すると、新規ユーザーと同様にダイジェスト認証を使用できるようになります。	/webdav
security.digest_ authentication.realm	HTTPダイジェスト認証のレルム文字列。これが変更されると、すべてのユーザーがパスワードをリセットする必要があります。リセットしない場合、HTTPダイジェスト認証によって保護されているサーバー機能にアクセスできなくなります。	CloverETL Server
security.digest_ authentication.nonce_ validity	秒単位で指定されたHTTPダイジェスト認証の有効間隔。この間隔が過ぎると、サーバーからクライアントに対して新規認証が求められます。これはほとんどのHTTPクライアントで自動的に行われます。	300
clover.event. fileCheckMinInterval	ファイル・チェックの間隔(ミリ秒)。詳細は、12章「ファイル・イベント・リスナー」を参照してください。	1000
clover.smtp.transport. protocol	SMTPサーバー・プロトコル。使用可能な値は"smtp"または"smtps"です。	smtp
clover.smtp.host	SMTPサーバーのホスト名またはIPアドレス	
clover.smtp.port	SMTPサーバーのポート	

キー	説明	デフォルト
clover.smtp.authentication	true/false。falseの場合、ユーザー名およびパスワードは無視されます。	
clover.smtp.username	SMTPサーバーのユーザー名	
clover.smtp.password	SMTPサーバーのパスワード	
clover.smtp.additional.*	接頭辞"clover.smtp.additional."の付いたプロパティは、メーラーに渡されるプロパティ・インスタンスに(接頭辞なしで)自動的に追加されます。一部のプロトコル固有パラメータで役立つ場合があります。接頭辞は削除されます。	
logging.project_name	製品名を指定する必要があるときにログ・メッセージで使用されます	CloverETL
logging.default_subdir	すべてのサーバー・ログのデフォルトのサブディレクトリ名。システム・プロパティ"java.io.tmpdir"によって指定されるパスの相対パスです。絶対パスは指定しないでください。絶対パス用として使用されるプロパティを使用してください。	cloverlogs
launch.log.dir	サーバーが起動リクエスト・ログを格納する必要がある場所。詳細は、起動サービスの項を参照してください。	${java.io.tmpdir}/[logging.default_subdir]/launch。${java.io.tmpdir}はシステム・プロパティです
graph.logs_path	サーバーがグラフ実行ログを格納する必要がある場所。詳細は、ロギングの項を参照してください。	${java.io.tmpdir}/[logging.default_subdir]/graph。${java.io.tmpdir}はシステム・プロパティです
temp.default_subdir	サーバーの一時ファイルのデフォルトのサブディレクトリ名。システム・プロパティ"java.io.tmpdir"によって指定されるパスの相対パスです。	clovertmp
graph.debug_path	サーバーがグラフ・デバッグ情報を格納する必要がある場所。	${java.io.tmpdir}/[temp.default_subdir]/debug。${java.io.tmpdir}はシステム・プロパティです
graph.pass_event_params_to_graph_in_old_style	3.0以降。グラフ・イベントによって実行されるグラフにパラメータを渡すための下位互換性用のスイッチ。3.0より前のバージョンでは、すべてのパラメータが実行対象グラフに渡されていました。3.0以降は、指定したパラメータのみが渡されます。詳細は、「タスク - グラフの実行」を参照してください。	false
threadManager.pool.corePoolSize	常にアクティブ(実行中またはアイドル中)であるスレッドの数。サーバー・イベントを処理するためのスレッド・プールに関連しています。	4

キー	説明	デフォルト
threadManager.pool.queueCapacity	スレッドを待機中のタスクが含まれるキュー(FIFO)の最大サイズ。サーバー・イベントを処理するためのスレッド・プールに関連しています。これは、"queueCapacity"より多い待機中のタスクがないことを意味します。たとえば、queueCapacity=0の場合、待機中のタスクはありません。各タスクは、使用可能なスレッドまたは新規スレッド内で即時実行されます。queueCapacity=1024の場合、キュー内で最大1024のタスクが"corePoolSize"から使用可能なスレッドを待機できます。	12
threadManager.pool.maxPoolSize	アクティブなスレッドの最大数。コア・プール内に使用可能なスレッドがなく、キュー容量を超えている場合、最大"maxPoolSize"の新規スレッドがプール内に作成されます。maxPoolSizeを超える同時タスクが存在する場合、スレッド・マネージャによってその実行が拒否されます。このため、queueCapacityまたはmaxPoolSizeは十分大きい値に設定してください。	1024
task.archivator.batch_size	1つのバッチ内で削除されるレコードの最大数。アーカイブされた実行レコードを削除するために使用されます。	50
launch.http_header_prefix	起動サービスによってHTTPレスポンスに追加されるHTTPヘッダーの接頭辞。	X-cloveretl
task.archivator.archive_file_prefix	アーカイバによって作成されるアーカイブ・ファイルの接頭辞。	cloverArchive_
license.context_names	ライセンスが含まれる可能性があるweb-appコンテキストのカンマ区切りのリスト。これらはそれぞれ先頭にスラッシュを付ける必要があります。Apache Tomcatの場合のみ機能します。	/clover-license、/clover_license
license.display_header	サーバーのWeb GUIにライセンス・ヘッダーを表示するかどうかを指定するスイッチ。	false

表18.2 ジョブ実行構成のデフォルト - 詳細はジョブ構成プロパティの項を参照

キー	説明	デフォルト
executor.tracking_interval	実行中のグラフの現在のステータスをスキャンする間隔(ミリ秒)。間隔が短いほど、ログ・ファイルは大きくなります。	2000
executor.log_level	グラフ実行のログ・レベル。TRACE | DEBUG | INFO | WARN | ERROR	INFO
executor.max_running_ concurrently	同時に存在(または実行)可能なグラフ・インスタンスの数。0は、制限がないことを意味します	0
executor.max_graph_instance_age	間隔(ミリ秒)。グラフ・インスタンスをメモリーから解放するまでアイドル状態にする時間を指定します。0は、制限がないことを意味します。このプロパティは2.8以降名前が変更されています。元の名前はexecutor.maxGraphInstanceAgeでした	0
executor.classpath	グラフで使用される変換/プロセッサ・クラスのクラスパス。ディレクトリ[sandbox_root]/trans/は、グラフ実行クラスパスに自動的に追	

キー	説明	デフォルト
	加されるため、ここにリストする必要はありません。	
executor.skip_check_config	グラフ構成の確認を無効にします。グラフ実行のパフォーマンスが向上します。グラフ開発時に役立つ場合があります。	true
executor.password	エンコードされたDB接続パスワードのデコードに使用するパスワード。	
executor.verbose_mode	TRUEの場合、グラフ実行の詳細なログが生成されます。	true
executor.use_jmx	TRUEの場合、グラフ・エグゼキュータは実行中のグラフのJMX mBeanを登録します。	true
executor.debug_mode	TRUEの場合、デバッグを有効にしたエッジがデバッグ・ディレクトリのファイルにデータを格納します。プロパティgraph.debug_pathを参照してください	false

プロパティの詳細は、クラスタリングの項を参照してください。

第19章 グラフ/ジョブフローのパラメータ

CloverETL Serverでは、グラフやジョブフローの実行ごとにパラメータ・セットが渡されます。

プレースホルダ(パラメータ) ${paramName}が解決されるのは初期化中(XML定義ファイルのロード中)のみであるため、ジョブの実行ごとにパラメータを解決する必要がある場合、ジョブをプールできないことに注意してください。ただし、次のように、現在のパラメータ値には常にインラインJavaコードによってアクセスできます。

```
String runId = getGraph().getGraphProperties().getProperty("RUN_ID");
```

プロパティは、次のように、追加または置換できます。

```
getGraph().getGraphProperties().setProperty("new_property", value );
```

CloverETL Serverによって常に設定されるパラメータ・セットは、次のとおりです。

表19.1 グラフ実行構成のデフォルト - 詳細はグラフ構成プロパティの項を参照

キー	説明
SANDBOX_CODE	実行対象グラフが含まれるサンドボックスのコード。
JOB_FILE	サンドボックスのルート・パスの相対ファイル・パス。
SANDBOX_ROOT	サンドボックスのルートの絶対パス。
RUN_ID	グラフ実行のID。スタンドアロン・モードまたはクラスタ・モードでは、常に一意です。実行レコードが永続でない場合は、値0より小さくなる可能性があります。詳細は、起動サービスを参照してください。

実行のタイプに応じた別のパラメータ・セット

グラフの実行方法に応じたパラメータは他にもあります。

Web GUIからの実行

パラメータなし

起動サービスの呼出しによる実行

属性Pass to graphが有効であるサービス・パラメータは、ディクショナリ入力データとしてだけではなく、グラフ・パラメータとしてもグラフに渡されます。

HTTP API実行グラフ操作の呼出しによる実行

"param_"接頭辞が付いたURLパラメータは、実行対象グラフに渡されますが、この場合は"param_"接頭辞は使用されません。たとえば、URLで指定された"param_FILE_NAME"は、"FILE_NAME"という名前のプロパティとしてグラフに渡されます。

RunGraphコンポーネントによる実行

3.0以降、"paramsToPass"属性によって指定されたパラメータのみが"parent"グラフから実行対象グラフに渡されます。ただし、一般的なプロパティ(RUN_ID、PROJECT_DIRなど)は新しい値によって上書きされます。

WS APIメソッドexecuteGraphの呼出しによる実行

値のあるパラメータを、実行リクエストによってグラフに渡すことができます。

スケジューラによるグラフの実行タスクによる実行

表19.2 渡されるパラメータ

キー	説明
EVENT_SCHEDULE_EVENT_TYPE	スケジュールのタイプ SCHEDULE_PERIODIC | SCHEDULE_ONETIME
EVENT_SCHEDULE_LAST_EVENT	前のイベントの日付/時間
EVENT_SCHEDULE_DESCRIPTION	Web GUIに表示されるスケジュールの説明
EVENT_USERNAME	イベントを所有するユーザー。スケジュールの場合、これは、スケジュールを作成したユーザーです
EVENT_SCHEDULE_ID	グラフをトリガーしたスケジュールのID

JMSリスナーからの実行

受信メッセージのタイプに応じて、多くのグラフ・パラメータおよびディクショナリ・エントリが渡されます。詳細は、9章「JMSメッセージ・リスナー」を参照してください。

グラフ/ジョブフロー・イベント・リスナーによるグラフの開始タスクによる実行

3.0以降、デフォルトでは、ソース・ジョブから指定したプロパティのみが実行対象ジョブに渡されます。この動作を変更し、ソース・ジョブからすべてのパラメータが実行対象ジョブに渡されるようにできるサーバー構成プロパティ"graph.pass_event_params_to_graph_in_old_style"も用意されています。このスイッチは、下位互換性のために実装されています。デフォルトの動作については、グラフ・イベント・リスナーのエディタで渡すパラメータのリストを指定できます。詳細は、タスク - グラフの実行の項を参照してください。

ただし、現在の値を持つ次のパラメータは常にターゲット・ジョブに渡されます

表19.3 渡されるパラメータ

キー	説明
EVENT_RUN_SANDBOX	イベントのソースであるグラフが含まれるサンドボックス
EVENT_JOB_EVENT_TYPE	GRAPH_STARTED | GRAPH_FINISHED | GRAPH_ERROR | GRAPH_ABORTED | GRAPH_TIMEOUT | GRAPH_STATUS_UNKNOWN（ジョブフロー・イベント・リスナーのJOBFLOW_*と類似)。
EVENT_RUN_JOB_FILE	イベントのソースであるジョブのjobFile
EVENT_RUN_ID	イベントのソースであるグラフ実行のID。

キー	説明
EVENT_TIMEOUT	タイムアウトの間隔を指定するミリ秒数。"timeout"グラフ・イベントの場合のみ有効です。
EVENT_RUN_RESULT	イベントのソースである実行の結果(または現在のステータス)。
EVENT_USERNAME	イベントを所有するユーザー。グラフ・イベントの場合は、グラフ・イベント・リスナーを作成したユーザーです

ファイル・イベント・リスナーによるグラフの実行タスクによる実行

表19.4 渡されるパラメータ

キー	説明
EVENT_FILE_PATH	イベントのソースであるファイルのパス。ファイル名は含まれません。末尾はファイル・セパレータではありません。
EVENT_FILE_NAME	イベントのソースであるファイルのファイル名。
EVENT_FILE_EVENT_TYPE	SIZE | CHANGE_TIME | APPEARANCE | DISAPPEARANCE
EVENT_FILE_PATTERN	ファイル・イベント・リスナーで指定されるパターン
EVENT_FILE_LISTENER_ID	
EVENT_USERNAME	イベントを所有するユーザー。ファイル・イベントの場合は、ファイル・イベント・リスナーを作成したユーザーです

別のグラフ・パラメータの追加方法

追加のグラフ構成パラメータ

選択したグラフまたは選択したサンドボックス内のすべてのグラフに対してWeb GUIのセクションサンドボックスで、いわゆる追加パラメータを追加できます。詳細は、「ジョブ構成プロパティ」を参照してください。

タスクexecute_graphのパラメータ

グラフの実行タスクは、スケジュール、グラフ・イベント・リスナーまたはファイル・イベント・リスナーによってトリガーできます。タスク・エディタを使用すると、実行対象グラフに渡されるキーと値のペアを指定できます。

第20章 変換開発者向けの推奨事項

app-serverクラスパスへの外部ライブラリの追加

接続(JDBC/JMS)には、サード・パーティのライブラリが必要な場合があります。これらのライブラリをapp-serverクラスパスに追加することをお薦めします。

CloverETLを使用すると、これらのライブラリをグラフ定義に直接指定できます。これにより、CloverETLでは、これらのライブラリを動的にロードできるようになります。ただし、この場合、外部ライブラリを使用すると、メモリー・リークが発生し、"java.lang.OutOfMemoryError: PermGen space"が発生する可能性があります。

また、app-serverのクラスパスにはJMS APIが必要ですが、多くの場合、サード・パーティのライブラリにはこのAPIもバンドルされています。このため、これらのライブラリが同じクラスローダによってロードされていない場合、クラスロードの競合が発生する可能性があります。

RunGraphコンポーネントによって実行される別のグラフは同じJVMインスタンスでのみ実行可能

サーバー環境では、すべてのグラフが同じVMインスタンスで実行されます。RunGraphコンポーネントの属性"same instance"をFALSEに設定することはできません。

第21章 ロギング

メイン・ログ

CloverETL Serverでは、ロギング用としてlog4jライブラリが使用されます。WARファイルには、デフォルトのlog4j構成が含まれます。

デフォルトでは、ログ・ファイルは、システム・プロパティ"java.io.tmpdir"によって指定されるディレクトリ内の"cloverlogs"サブディレクトリ内に作成されます。

通常、"java.io.tmpdir"には、一般的なシステム一時ディレクトリ"/tmp"が含まれます。Tomcatの場合は通常、これは"[TOMCAT_HOME]/temp"です

デフォルトのロギング構成は、システム・プロパティ"log4j.configuration"によってオーバーライドされる可能性があります。このプロパティには、log4j構成ファイルのURLが含まれる必要があります。

```
log4j.configuration=file:/home/clover/config/log4j.xml
```

このような構成によってデフォルトの構成がオーバーライドされるため、グラフ実行ログに影響が及ぶ可能性があります。このため、グラフ実行ログを保持するために独自のログ構成に次のコード片が含まれる必要があります

```
<logger name="Tracking" additivity="false"> <level value="debug"/> </logger>
```

これらのシステム・プロパティにより、HTTPリクエスト/レスポンスをstdoutに記録できるようになります。

クライアント側:

`com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true`（詳細は、CloverETL Designerのユーザーズ・ガイドのIntegrating CloverETL Designer with CloverETL Serverの章を参照してください）

サーバー側:

`com.sun.xml.ws.transport.http.HttpAdapter.dump=tru`

グラフ実行ログ

グラフまたはジョブフローの実行ごとに独自のログ・ファイルがあります。これは、たとえばサーバー・コンソールのセクション実行履歴でアクセスできます。

デフォルトでは、これらのログ・ファイルは、システム・プロパティ"java.io.tmpdir"によって指定されるディレクトリ内のサブディレクトリcloverLogs/graph内に保存されます。

CloverETLプロパティ"graph.logs_path"により、これらのログに対して別の場所を指定することもできます。このプロパティは、メイン・サーバー・ログには影響しません。

第22章 CloverETL Server APIの拡張性

CloverETL Serverでは、APIの拡張が実装されるため、カスタムAPIを使用して追加機能を公開できます。

GroovyコードAPIとOSGiプラグインの2つを使用できます。

GroovyコードAPI

3.3以降

CloverETL ServerのGroovyコードAPIを使用すると、クライアントでは、HTTPリクエストによってサーバーに格納されているGroovyコードを実行できます。実行対象コードでは、ServerFacade、インスタンスのHTTPリクエストおよびHTTPレスポンスにアクセスできるため、GroovyコードでカスタムCloverETL Server APIを実装できます。

コードを実行するには、次のURLをコールします。

```
http://[host]:[port]/clover/groovy/[sandboxCode]/[pathToGroovyCodeFile]
```

プロトコルHTTPは、Webサーバーの構成に応じてセキュアなHTTPSに変更できます。

サーバーでは、構成に応じて基本認証またはダイジェスト認証が使用されます。そのため、ユーザーは、認可されているとともに、指定したサンドボックス内で実行する権限およびGroovyコードAPIをコールする権限を持っている必要があります。

GroovyコードAPIをコール(および編集)する権限は非常に強力な権限です。これは通常、Groovyコードは、Javaコードと同じ処理を実行でき、アプリケーション・コンテナと同じシステム・プロセスとして実行されるためです。

Groovyコードでアクセス可能な変数

デフォルトでは、これらの変数はGroovyコードからアクセス可能です。

表22.1 Groovyコードでアクセス可能な変数

タイプ	キー	説明
javax.servlet.http.HttpServletRequest	request	コードをトリガーしたHTTPリクエストのインスタンス。
javax.servlet.http.HttpServletResponse	response	スクリプトの終了時にクライアントに送信されるHTTPレスポンスのインスタンス。
javax.servlet.http.HttpSession	session	HTTPセッションのインスタンス。
javax.servlet.ServletConfig	servletConfig	ServletConfigのインスタンス
javax.servlet.ServletContext	servletContext	ServletContextのインスタンス
com.cloveretl.server.api.ServerFacade	serverFacade	CloverETL Serverコア機能をコールする場合に使用可能なserverFacadeのインスタンス。 WARファイルには、ファサードAPIのJavaDocが含まれます。このファイルは、URL: http://host:port/clover/javadoc/index.htmlからアクセスできます

タイプ	キー	説明
String	sessionToken	serverFacadeメソッドのコールに必要なsessionToken

コード例

コードによりHTTPレスポンスのコンテンツとして戻される文字列が戻されたり、コードにより出力ライターに対する出力自体が構成される場合があります。

次のスクリプトの場合、独自の出力が作成されますが、何も戻されません。このため、基礎となるサーブレットによって出力は変更されません。この方法の利点は、出力を高速で構成し、クライアントに継続的に送信できることです。ただし、出力ストリーム(またはライター)がオープンされた場合、スクリプト処理時にエラーが発生してもエラー説明は送信されません。

```
response.getWriter().write("write anything to the output");
```

次のスクリプトの場合、文字列が戻されるため、基礎となるサーブレットにより、この文字列が出力に追加されます。この方法の利点は、コード処理時にエラーが発生した場合、完全なスタックトレースが戻されるため、スクリプトを修正できることです。ただし、構成された出力によってある程度のメモリーが消費される可能性があります。

```
String output = "write anything to the output"; return output;
```

次のスクリプトは、少々複雑になっています。この場合、構成されているすべてのスケジュールのリストとともにXMLが戻されます。これらのスケジュールをリストするには、権限が必要です。

```
// uses variables: response, sessionToken, serverFacade
import java.io.*;
import java.util.*;
import javax.xml.bind.*;
import com.cloveretl.server.facade.api.*;
import com.cloveretl.server.persistent.*;
import com.cloveretl.server.persistent.jaxb.*;

JAXBContext jc =
JAXBContext.newInstance( "com.cloveretl.server.persistent:com.cloveretl.server.
persistent.jaxb" ); Marshaller
m = jc.createMarshaller();
m.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
m.setProperty(Marshaller.JAXB_SCHEMA_LOCATION, "/clover/schemas/cs.xsd");

Response<List<Schedule>> list = serverFacade.findScheduleList(sessionToken, null);
SchedulesList xmlList = new SchedulesList();
xmlList.setSchedulesList(list.getBean());
m.marshal(xmlList, response.getWriter());
```

組込みOSGiフレームワーク

3.0以降

CloverETL Serverには、プラグイン(OSGiバンドル)の実装を可能にする、組込みOSGiフレームワークが含まれます。これにより、新しいAPI操作を追加したり、サーバー・コンソールのUIを拡張したりできます。これは、標準のclover.warから独立したものです。

CloverETL自体は、OSGiテクノロジに基づくものではありません。OSGiは、サーバーAPIを拡張する目的にのみ、オプションで使用されます。OSGiフレームワークはデフォルトでは完全に無効になっており、プロパティ"plugins.path"が次のように設定された場合にのみ有効になります。

組込みOSGiフレームワークEquinoxでは、いくつかの侵入型手法が使用されるため、他のアプリケーションや一部のCloverETL機能と干渉する場合があります。通常は、前述のGroovy APIを使用してください。ただし場合によっては、OSGiプラグインを使用したほうが望ましいケースもあります。たとえば、カスタムAPIで複数の異なるライブラリを使用した後、サーバー・クラスパス上のライブラリを使用する必要がある場合などです。Groovyでは同じクラスパスがCloverETLとして使用されるのに対し、OSGiプラグインには分離された独自のクラスパスがあります。したがって、このようなケース(OSGiプラグインを使用する必要があるケース)では、CloverETLをできるだけ単純な環境にデプロイしてください(たとえば、TomcatやJettyなどの軽量なコンテナ上に、Cloverを唯一のアプリケーションとしてデプロイするなど)。このようなデプロイメントにより、干渉の可能性を最小限に抑えることができます。

組込みOSGiフレームワークとの干渉の例(フレームワークが明示的に構成および初期化されている場合のみ)

- OSGiフレームワークが原因で、IBM Websphere上のCloverETLコンポーネントWebServiceClientおよびEmailReaderが正常に機能しない
- OSGiフレームワークが原因で、Oracle JRockit JVM上で実行されているWeblogic 10.3.5上のCloverETLが正常に起動しない
- OSGiフレームワークがWeblogic 12上で適切に初期化できない

プラグインの可用性

プラグインは基本的に、起動サービス、HTTP APIまたはWebサービスAPIと同様に新規サーバーAPIとして機能します。これは、単純なJSP、HttpServletまたは複雑なSOAP Webサービスとしてのみ使用できます。プラグインにHTTPサービスが含まれる場合、起動時に指定したURLでリスニングするために登録され、受信HTTPリクエストはWebコンテナからプラグインにブリッジされます。プラグイン自体は、"ServerFacade"と呼ばれるCloverETL Serverの内部インタフェースにアクセスできます。ServerFacadeには、グラフの実行、グラフ・ステータスおよび実行履歴の取得、スケジューリング、リスナー、構成の操作などの多数の方法が用意されています。このため、APIは、特定のデプロイメントのニーズに応じてカスタマイズできます。

OSGiバンドルのデプロイ

OSGiフレームワークに関するCloverETL Serverの構成プロパティは2つあります。

- plugins.path - すべてのプラグイン(JARファイル)が含まれるディレクトリへの絶対パス。
- plugins.autostart - カンマ区切りのプラグイン名のリストです。これらのプラグインは、サーバーの起動時に起動されます。理論的には、OSGiフレームワークはOSGiバンドルをオンデマンドで起動できますが、サーブレット・コンテナへのサーブレット・ブリッジが使用される場合、このバンドルは信頼性が低くなります。このため、すべてのプラグインを指定することをお薦めします。

プラグインをデプロイするには、2つの構成プロパティを設定し、"plugins.path"によって指定されるディレクトリにプラグインをコピーし、サーバーを再起動します。

第23章 拡張性 - CloverETLエンジンのプラグイン

3.1.2以降

CloverETL Serverでは、指定したソースからロードされる外部エンジン・プラグインを使用できます。ソースは、構成プロパティ"engine.plugins.src"をコピーすることによって指定します。

CloverETL構成の可用性の詳細は、18章「構成」を参照してください。

このプロパティは、追加のCloverETLエンジン・プラグインが含まれるディレクトリまたはzipファイルの絶対パスである必要があります。ディレクトリとzipファイルには両方とも、各プラグインのサブディレクトリが含まれる必要があります。これらのプラグインは、WARに圧縮されているプラグインの代替ではありません。ディレクトリまたはZIPファイルの変更が適用されるのは、サーバーの再起動時のみです。

各プラグインには、デフォルトではparent-first計画を使用する独自のクラスローダがあります。プラグインのクラスローダの親は、web-appクラスローダ([WAR]/WEB-INF/libのコンテンツ)です。プラグインにサード・パーティのライブラリが使用されている場合、親クラスローダのクラスパス上のライブラリとの競合が発生する可能性があります。これらは一般的な例外/エラーであり、クラスロードに問題があることを示しています。

- java.lang.ClassCastException
- java.lang.ClassNotFoundException
- java.lang.NoClassDefFoundError
- java.lang.LinkageError

このような競合を解消する方法はいくつかあります。

- 競合しているサード・パーティのライブラリを削除し、親クラスローダ(web-appまたはapp-serverクラスローダ)上のライブラリを使用します
- プラグインに対して異なるクラスロード計画を使用します。
  - プラグイン記述子plugin.xmlで、要素"plugin"に属性greedyClassLoader="true"を設定します
  - これは、プラグイン・クラスローダでself-first計画が使用されることを意味します
- 選択したJavaパッケージに対して逆のクラスロード計画を設定します。
  - プラグイン記述子plugin.xmlで、要素"plugin"に属性"excludedPackages"を設定します。
  - これは、パッケージの接頭辞のカンマ区切りのリストです。例: excludedPackages="some.java.package,some.another.package"
  - 前の例では、"some.java.package"、"some.another.package"およびこれらのすべてのサブパッケージが逆のクラスロード計画を使用してロードされてから、プラグインのクラスパス上の残りのクラスがロードされます。

前述の提案は組み合せることもできます。これらの競合に対して最適な解決策を見つけることは容易ではなく、このような解決策は、app-serverクラスパス上のライブラリによって異なります。

より効率的にデバッグを行うには、関連するクラスローダにTRACEログ・レベルを設定することが有用です。

```
<logger name="org.jetel.util.classloader.GreedyURLClassLoader">
  <level value="trace"/>
</logger>
<logger name="org.jetel.plugin.PluginClassLoader">
  <level value="trace"/>
</logger>
```

サーバーのlog4j構成のオーバーライドの詳細は、ロギングの項を参照してください。

第24章 クラスタリング

クラスタには、高可用性とスケーラビリティという2つの一般的な機能があります。これらは両方ともCloverETL Serverによって異なるレベルで実装されます。この項では、CloverETLクラスタリングの基礎について詳細を説明します。

CloverETL Serverは、ユーザーのライセンスによって許可されている場合のみ、クラスタ内で動作します。

高可用性

バージョン3.0以降、CloverETL Serverでは、クラスタ・ノード間の差異は認識されません。このため、マスター・ノードやスレーブ・ノードは存在しません。つまり、すべてのノードが実質的に対等です。CloverETLクラスタ自体には単一点障害(SPOF)はありません。ただし、SPOFは入力データや他の一部の外部要素内に存在する可能性があります。

クラスタリングにより、HTTPを介してアクセスできるすべての機能の高可用性(HA)が実現されます。これには、サンドボックスの参照、サービス構成(スケジューリング、起動サービス、リスナー)の変更および主としてジョブ実行が含まれます。任意のクラスタ・ノードが受信HTTPリクエストを受け入れ、これらをそのノード自体が処理するか、別のノードに委任できます。

すべてのノードが対等であるため、任意のクラスタ・ノードがほとんどすべてのリクエストを処理できます。

- ジョブ・ファイル、メタデータ・ファイルなどはすべて共有サンドボックス内にあります。このため、すべてのノードがこれらにアクセスできます。共有ファイルシステムはSPOFである場合があるため、レプリケートされたファイルシステムをかわりに使用することをお薦めします。

- データベースは、すべてのクラスタ・ノードによって共有されます。また、共有DBもSPOFである場合がありますが、これもクラスタ化できます。

ただし、まだなお、ノード自体がリクエストを処理できない可能性があります。このような場合、このリクエストは、これを処理できるノードに完全かつ透過的に委任されます。

1つ(または複数)の特定ノードに制限されるリクエストは、次のとおりです。

- パーティション化されたサンドボックスまたはローカル・サンドボックスの内容に対するリクエスト。これらのサンドボックスはすべてのクラスタ・ノード間では共有されません。このようなリクエストは任意のクラスタ・ノードに向けられる可能性があります。この場合、このリクエストはターゲット・ノードに委任されますが、このターゲット・ノードが起動し実行中である必要があります。

- パーティション化されたサンドボックスまたはローカル・サンドボックスを使用するようジョブが構成されています。これらのグラフには、必要なサンドボックスに物理的にアクセスできるノードが必要です。

- ジョブには、特定のクラスタ・ノードによって指定された割当てがあります。割当ての概念については、次の項で説明します。

これにより、アクセスできないクラスタ・ノードがあるとリクエストが失敗する可能性があるため、可能な場合は、特定のクラスタ・ノードの使用、または特定のクラスタ・ノードからのみアクセス可能なリソースの使用は避けることをお薦めします。

CloverETL自体には、ジョブを実行するためのロード・バランサが実装されています。このため、特定のノードを対象として構成されていないジョブをクラスタ内の任意の場所で実行でき、CloverETLのロード・バランサにより、このジョブを処理するノードがリクエストと現在の負荷に応じて決定されます。この処理はクライアント側に対してすべて透過的に実行されます。

HAを実現するには、独立したHTTPロード・バランサを使用することをお薦めします。独立したHTTPロード・バランサを使用すると、HTTPリクエストに対して透過的なフェイルオーバーが可能になります。これにより、実行中のノードにリクエストが送信されます。

スケーラビリティ

実装されるスケーラビリティには独立した2つのレベルがあります。変換リクエスト(および任意のHTTPリクエスト)のスケーラビリティとデータのスケーラビリティ(パラレル・データ処理)です。

これらのスケーラビリティ・レベルは両方とも水平です。水平スケーラビリティとは、クラスタにノードを追加することを意味します。一方、垂直スケーラビリティとは、単一ノードにリソースを追加することを意味します。垂直スケーラビリティは、CloverETLエンジンによってネイティブにサポートされており、ここでは説明しません。

変換リクエスト

基本的に、クラスタ内のノードが多いほど、一度に処理できる変換リクエスト(または一般的にはHTTPリクエスト)が多くなります。このタイプのスケーラビリティは、増大するクライアント数をサポートするためのCloverETL Serverの機能です。この機能は、前の項で説明したHTTPロード・バランサの使用と密接に関連しています。

パラレル・データ処理

このタイプのスケーラビリティは現在、ETLグラフに対してのみ使用可能です。ジョブフロー・ジョブとプロファイラ・ジョブをパラレルに実行することはできません。

変換がパラレルに処理される場合、グラフ全体(またはその一部)が複数のクラスタ・ノード上でパラレルに実行されます。この場合、各ノードによってデータの一部のみが処理されます。

このため、クラスタ内のノードが多いほど、指定した時間内に処理できるデータが多くなります。

データは、グラフ実行の前に分割(パーティション化)することも、高速でグラフ自体が分割することもできます。この結果生成されるデータは、パーティションに格納することも、収集して1つのデータ・グループとして格納することもできます。

スケーラビリティの曲線図は、変換のタイプによって異なる場合があります。これはほぼ直線になる場合があり、ほとんどの場合、これが理想的な形です。ただし、データ・ソースが単一であるために複数のリーダーによる読取りが不可能であることが原因でそれ以上のデータ変換の速度が制限される場合は例外です。このような場合、実際には入力データを待機することになるため、パラレル・データ処理を行う利点がなくなります。

ETLグラフ割当て

クラスタ環境で実行される各ETLグラフは、自動的に変換分析の対象となります。この分析の主な目的は、いわゆるETLグラフ割当てを検出することです。グラフ割当てとは、クラスタ環境での変換の実行方法に関する一連の指示のことです。パラレル・データ処理がどのように機能するかをより深く理解するには、グラフ分析とその結果の割当てに関するより詳細な情報を取得する必要があります。

分析ではまず、各コンポーネントに対する割当てを検出する必要があります。コンポーネント割当てとは、そのコンポーネントの実行先となる一連のクラスタ・ノードのことです。コンポーネント割当てを指定するにはいくつかの方法があります(ドキュメントの次の項を参照してください)。ただし、ここで重要なことは、コンポーネントを複数のインスタンスで実行するようにリクエストできるということと、それがパラレル・データ処理に必要だということです。分析の次の手順は、すべてのコンポーネント割当てに対応する最適なグラフ分解を検出して、グラフ・インスタンス間のリモート・エッジ数を最小化することです。

分析結果では、実行する必要があるグラフのインスタンス数と、それらのインスタンスの実行先となるクラスタ・ノード、そして各インスタンスに配置されるコンポーネントが示されます。言い換え

れば、1つの実行対象グラフは多数のインスタンスで実行でき、各インスタンスは任意のクラスタ・ノードで処理でき、さらに各インスタンスには最適なコンポーネントのみを含めることができます。

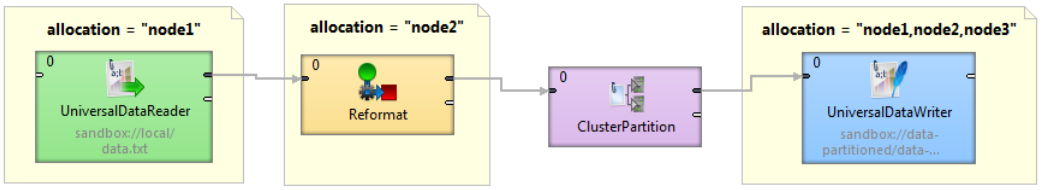


図24.1 コンポーネント割当ての例

この図は、いくつかのコンポーネントに様々なコンポーネント割当てが指定されたサンプル・グラフです。1つ目のコンポーネントであるUniversalDataReaderはnode1で実行するようにリクエストされ、2つ目のReformatコンポーネントはクラスタ・ノード2で実行されます。ClusterPartitionコンポーネントは、相互接続された2つのコンポーネントの割当てのカーディナリティを変更できるようにする特殊なコンポーネントです(クラスタのパーティション化と収集については、後の項で詳しく説明します)。最後のUniversalDataWriterコンポーネントでは、3つのクラスタ・ノード(node1、node2およびnode3)上で実行するように要求されます。変更分析を視覚化すると次の図のようになります。3つのワーカー(グラフ)がそれぞれ異なるクラスタ・ノードで実行されます(複数のワーカーを1つのノードに関連付けることも可能なので、これは必須ではありません)。クラスタノードのnode1のワーカーには、UniversalDataReaderと、UniversalDataWriterコンポーネントの3つのインスタンスから最初の1つのみが含められます。これらのコンポーネントはどちらも、リモート・エッジによって、node2で実行されているコンポーネントに接続されます。node3で実行されているワーカーには、node2で実行されているClusterPartitionerからリモートに転送されたデータによってフィードされた、UniversalDataWriterのみが含まれます。

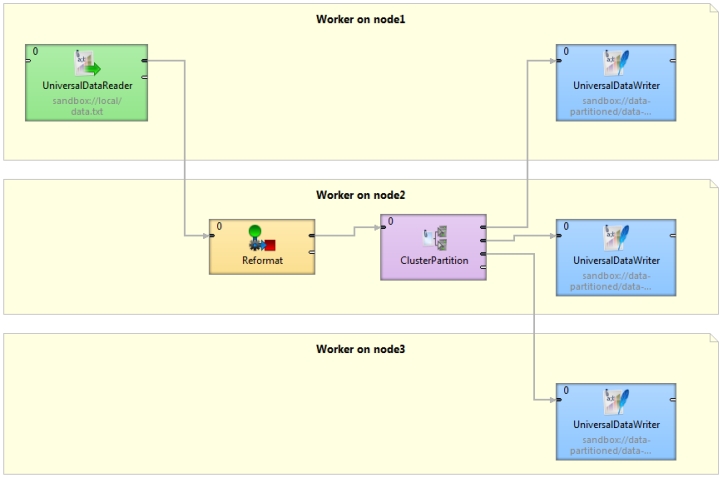


図24.2 コンポーネント割当てに基づくグラフ分解

コンポーネント割当て

1つのコンポーネントの割当ては、複数の方法で導出できます(リストは所定の優先順位でソートされます)。

- 明示的定義 – すべてのコンポーネントに共通の属性割当てがあります。CloverETL Designerでは、使いやすいダイアログを使用できます。

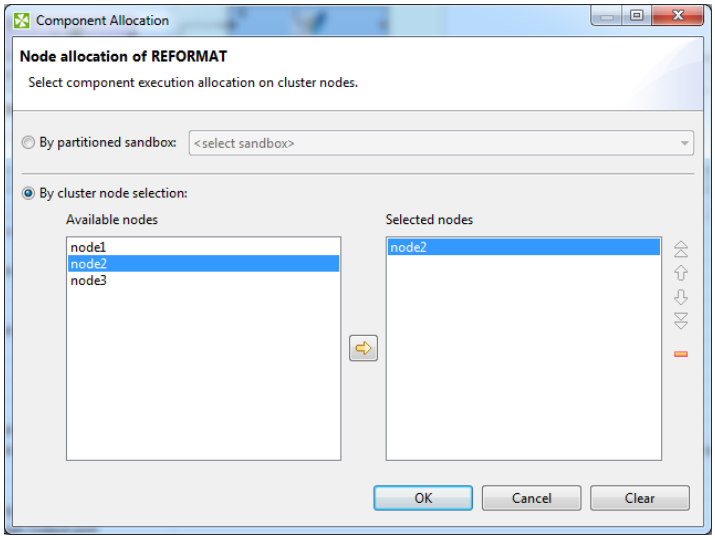


図24.3 コンポーネント割当てダイアログ

明示的割当て定義では、次の3つのアプローチを使用できます。

- ワーカー数に基づく割当て – コンポーネントは、CloverETLクラスタによって優先付けられた一部のクラスタ・ノード上にあるリクエストされたインスタンスで実行されます。サーバーでは、データ処理を最速化するために組込みのロード・バランシング・アルゴリズムを使用できます。
- パーティション化されたサンドボックスでの参照に基づく割当て – コンポーネント割当ては、指定のパーティション化されたサンドボックスの場所に対応します。パーティション化されたサンドボックスごとに場所のリストがあり、それぞれが特定のクラスタ・ノードにバインドされています。したがって、割当ては場所のリストに対応したものになります。詳細は、「パーティション化されたサンドボックスおよびローカル・サンドボックス」のパーティション化されたサンドボックスを参照してください。
- クラスタ・ノード識別子のリストによって定義された割当て(単一のクラスタ・ノードをより頻繁に使用できます)

- パーティション化されたサンドボックスへの参照 UniversalDataReader、UniversalDataWriterおよびParallelReaderコンポーネントは、自身の割当てをfileURL属性から導出します。URLがパーティション化されたサンドボックス内のファイルを参照している場合、コンポーネント割当ては自動的にパーティション化されたサンドボックスの場所から自動的に導出されます。そのため、パーティション化されたサンドボックス内のファイルを使用してこれらのコンポーネントのいずれかを操作する場合は、適切な割当てが自動的に使用されます。
- 隣接するコンポーネントからの採用 デフォルトでは、割当ては隣接するコンポーネントから継承されます。左側のコンポーネントが優先されます。再帰的割当て継承については、クラスタ・パーティショナとクラスタ収集が特性バインドとなります。

データのパーティション化/収集

前述のとおり、データは複数の方法でパーティション化および収集できます。これは、グラフ実行の前に準備することも、高速でパーティション化することもできます。

高速でのパーティション化/収集

ClusterPartition、ClusterLoadBalancingPartition、ClusterSimpleCopy、ClusterSimpleGather、ClusterMergeおよびClusterRepartitionという6つの考慮が必要な特別なコンポーネントがあります。いずれも非クラスタ版と同様に機能します。ただし、それぞれの分割または収集の特性はデータ・フロー割当ての変更に使用されるため、ワーカー間でのデータ配布の変更に使用される場合もあります。

ClusterPartitionとClusterLoadBalancingPartitionは共通のパーティショナと同様に機能し、データ割当てを1からNに変更します。ClusterPartitionerの前のコンポーネントは1つのノードで実行されますが、ClusterPartitionerの後のコンポーネントはノード割当てに応じてパラレルに実行されます。ClusterSimpleCopyコンポーネントは類似する複数の場所で使用できます。このコンポーネントはデータ・レコードを配布しませんが、すべての出力ワーカーにコピーします。

ClusterGatherとClusterMergeは、正反対に機能します。これらは、データ割当てを1からNに変更します。収集/マージの前のコンポーネントはパラレルに実行され、収集の後のコンポーネントは1つのノードでのみ実行されます。

外部ツールによるデータのパーティション化/収集

高速でデータをパーティション化すると、場合によっては不要なボトルネックが発生する可能性があります。低レベルのツールを使用してデータを分割したほうが、スケーラビリティの観点からは効率的である場合があります。最適なのは、実行中のワーカーごとに独立したデータ・ソースからデータが読み取られる状態です。このため、ClusterPartitionerコンポーネントが存在する必要はなく、グラフは最初からパラレルに実行されます。

または、グラフ全体がパラレルに実行されていてもかまいませんが、結果はパーティション化されます。

ノード割当ての制限事項

前述したように、各コンポーネントには独自のノード割当てが指定されている場合があり、その場合、いくつかの干渉が発生する可能性があります。

- 隣接するコンポーネントのノード割当てはカーディナリティが同じである必要があります したがって、同じ割当てである必要はありませんが、カーディナリティは同じである必要があります。たとえば、あるETLグラフにDataGeneratorおよびTrashという2つのコンポーネントがあるとします。nodeAに割り当てられたDataGeneratorはnodeBに割り当てられたTrashにデータを送信できます。nodeAに割り当てられたDataGeneratorがnodeAに割り当てられたTrashにデータを送信すると失敗します。

- ClusterGatherおよびClusterMergeの後のノード割当ては、カーディナリティが1である必要があります したがって、どのような割当てでもかまいませんが、カーディナリティは1である必要があります。

- ClusterPartition、ClusterLoadBalancingPartitionおよびClusterSimpleCopyの前のコンポーネントのノード割当ては、カーディナリティが1である必要があります

パーティション化されたサンドボックスおよびローカル・サンドボックス

パーティション化されたサンドボックスおよびローカル・サンドボックスについては、前述の項で説明しました。これらの新しいサンドボックス・タイプはバージョン3.0で導入されたもので、これらはパラレル・データ処理には不可欠です。

共有サンドボックスとあわせて、合計で3つのサンドボックス・タイプがあります。

共有サンドボックス

このタイプのサンドボックスは、すべてのクラスタ・ノードでアクセスできるすべてのデータに対して使用する必要があります。これには、HA（前述参照）をサポートする必要があるすべてのグラフ、グラフのメタデータ、接続、クラスおよび入力/出力データが含まれます。すべての共有サンドボックスは、すべてのクラスタ・ノード間で適切に共有されるディレクトリ内に配置される必要があります。オペレーティング・システムとファイルシステムに応じて、適切な共有/レプリケーティング・ツールを使用できます。

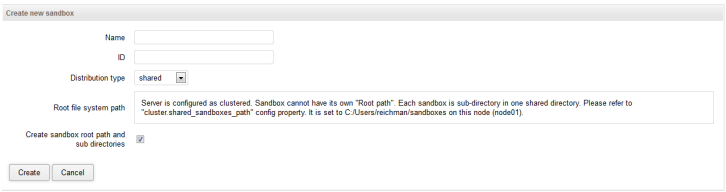


図24.4 新規共有サンドボックスを作成するためのダイアログ・フォーム

前述のスクリーンショットからわかるように、ファイルシステムで任意のルート・パスを指定できるわけではありません。共有サンドボックスは、"cluster.shared_sandboxes_path"によって指定されるディレクトリ内に格納されます。各共有サンドボックスの内部には、サンドボックスIDによって名前が付けられた独自のサブディレクトリがあります。

ローカル・サンドボックス

このサンドボックス・タイプは、特定のクラスタ・ノードによってのみアクセスできるデータを対象としています。これには、大量の入力/出力ファイルが含まれる可能性があります。これには次のような目的があります。つまり、任意のクラスタ・ノードがこのタイプのサンドボックスの内容にアクセスできるが、ローカル(高速)アクセスが可能なノードは1つのみであり、このノードはデータ提供のために起動し実行中である必要があります。グラフでは、物理的に異なるノードに格納されている複数のサンドボックスのリソースを使用できます。これは、クラスタ・ノードでは、リソースがローカル・ファイルであるかのようにネットワーク・ストリームを透過的に作成できるためです。詳細は、「コンポーネント・データ・ソースとしてのサンドボックス・リソースの使用」を参照してください。

ローカル・サンドボックスは一般的なプロジェクト・データ(グラフ、メタデータ、接続、ルックアップ、プロパティ・ファイルなど)には使用しないでください。予期できない動作が発生する可能性があります。かわりに共有サンドボックスを使用してください。

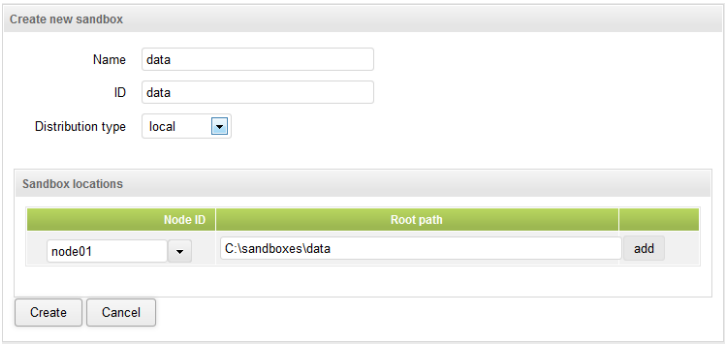


図24.5 新規ローカル・サンドボックスを作成するためのダイアログ・フォーム

パーティション化されたサンドボックス

このタイプのサンドボックスは実際には、通常は異なるクラスタ・ノード上に存在する一組の物理的な場所用の抽象ラッパーです。ただし、同じノード上に複数の場所が存在する場合もあります。パーティション化されたサンドボックスには2つ目的があり、これらは両方ともパラレル・データ処理と密接に関連しています。

1. ノード割当て仕様 - パーティション化されたサンドボックスの場所により、グラフまたはグラフの一部を実行するワーカーが定義されます。このため、物理的な場所ごとに実行する単一のワーカーが実行されます。このワーカーは実際にデータをその場所に格納する必要はありません。これは、これらのノードでETLグラフのこの部分をパラレルに実行するようCloverETL Serverに指示するための単なる手段です。

2. パラレル・データ処理時の一部のデータの記憶域。物理的な場所ごとに一部のデータのみが含まれます。通常の使用方法の場合、より多くの入力ファイルに入力データを分割するため、各ファイルを異なる場所に配置し、ワーカーごとに独自のファイルが処理されます。

前述のスクリーンショットからわかるように、パーティション化されたサンドボックスの場合、異なるクラスタ・ノードで物理的な場所を1つ以上指定できます。

パーティション化されたサンドボックスは一般的なプロジェクト・データ(グラフ、メタデータ、接続、ルックアップ、プロパティ・ファイルなど)には使用しないでください。予期できない動作が発生する可能性があります。かわりに共有サンドボックスを使用してください。

コンポーネント・データ・ソースとしてのサンドボックス・リソースの使用

共有サンドボックス、ローカル・サンドボックスまたはパーティション化されたサンドボックス(またはスタンドアロン・サーバー上の通常のサンドボックス)であるかどうかとは関係なく、サンドボックス・リソースは、グラフ内で次のようないわゆるサンドボックスURLとしてfileURL属性の下に指定されます。

sandbox://data/path/to/file/file.dat

"data"はサンドボックスのコードであり、"path/to/file/file.dat"はサンドボックス・ルートからリソースへのパスです。URLはグラフの実行時にCloverETL Serverによって評価され、コンポーネント(リーダーまたはライター)により、サーバーからオープンされたストリームが取得されます。これは、ローカル・ファイルに対するストリームである場合や、他のリモート・リソースに対するストリームである場合があります。このため、グラフは、リソースに対してローカルでアクセスできるノードで実行する必要はありません。グラフ内でより多くのサンドボックス・リソースが使用され、これらの各リソースが異なるノード上にある場合もあります。このような場合、CloverETL Serverでは、リモート・ストリームを最小限に抑えるために最もローカルなリソースを持つノードが選択されます。

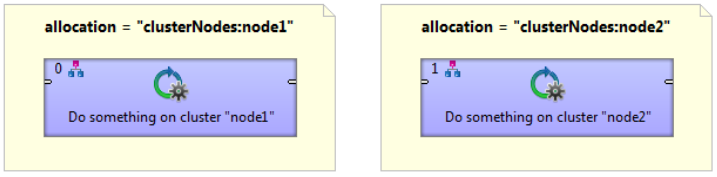
サンドボックスURLには、パラレル・データ処理用の特別な用途があります。パーティション化されたサンドボックス内のリソースがあるサンドボックスURLが使用される場合、グラフ/フェーズの一部は、パーティション化されたサンドボックスの場所のリストで指定されるノード割当てに応じてパラレルに実行されます。したがって、各ワーカーには独自のローカル・サンドボックス・リソースがあります。CloverETL Serverにより、各ワーカーのサンドボックスURLが評価され、ローカル・リソースに対するオープン・ストリームがコンポーネントに提供されます。

サンドボックスURLは、スタンドアロン・サーバーでも使用できます。グラフによって様々なサンドボックスからリソースが参照される場合、これは最適です。これには、メタデータ、ルックアップ定義または入力/出力データなどがあります。ただし、参照されるサンドボックスは、グラフを実行するユーザーがアクセスできる必要があります。

グラフ割当ての例

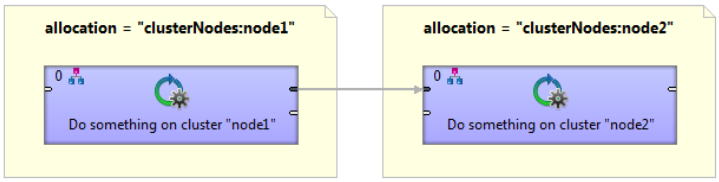
基本的なコンポーネント割当て

この例は2つのコンポーネントのグラフを示したものです。割当てにより、1つ目のコンポーネントがクラスタノードのnode1で実行され、2つ目のコンポーネントがクラスタノードのnode2で実行されるようになっています。



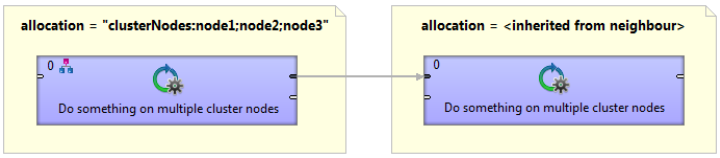
リモート・データ転送を使用した基本的なコンポーネント割当て

エッジで接続された2つのコンポーネントに、それぞれ異なる割当てを指定できます。1つ目はnode1で実行され、2つ目はnode2で実行されます。クラスタ環境により、リモート・データ・レコード転送が自動的に可能になります。



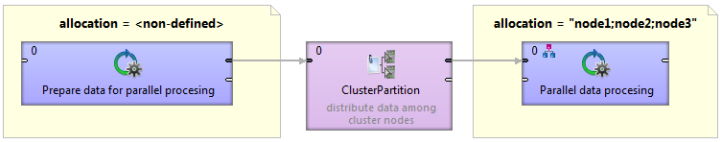
複数実行

複数ノード割当てを使用したグラフはパラレルで実行されます。この例では、両方のコンポーネントに同じ割当てが指定されているため、3つの同じ変換がクラスタノードのnode1、node2およびnode3で実行されます。



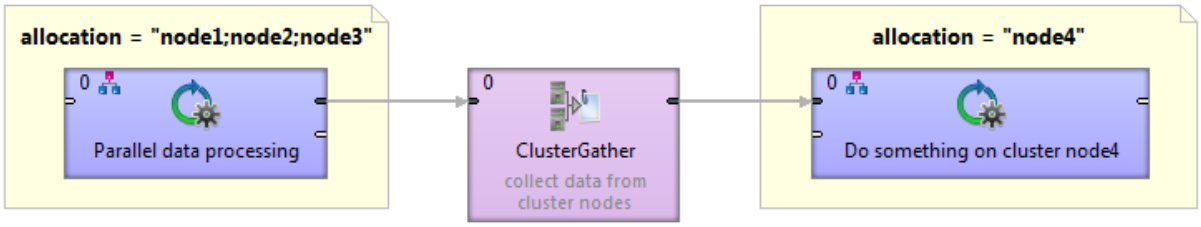
クラスタ・データ・パーティショニング

2つの割当てを使用したグラフです。1つ目のコンポーネントには未指定の単一のノード割当てがあり、自動的に導出されてリモート・エッジ数を最小化します。ClusterPartitionコンポーネントでは、レコードを配布してクラスタ・ノードのnode1、node2およびnode3でさらにデータを処理します。



クラスタ・データ収集

2つの割当てを使用したグラフです。1つ目のコンポーネントでパラレル・データ処理の結果に生じたデータ・レコードは、ClusterGatherコンポーネントで収集され、クラスタノードのnode4へと渡されてさらに単一ノードが処理されます。



クラスタ・デプロイメントに関する推奨事項

1. クラスタ内のすべてのノードでシステム日時を同期化する必要があります。
2. すべてのノードで、共有ファイルシステムまたはレプリケートされたファイルシステムに格納されたサンドボックスが共有されます。すべてのノード間で共有されるファイルシステムは単一点障害です。したがって、レプリケートされたファイルシステムを使用することをお薦めします。
3. すべてのノードでDBが共有されるため、トランザクションのサポートが必要になります。MySQL表エンジンのMyISAMはトランザクション型ではないため、このエンジンを使用すると予期しない動作が発生する可能性があります。

4. すべてのノードでDBが共有される場合、これは単一点障害になります。クラスタ化されたDBを使用することをお薦めします。

5. Tomcatでは、このプロパティのライセンスは"license.file"として構成してください。**clover_license.war**は使用しないでください。Tomcatでは、web-appが予期しない順序でロードされるため、クラスタについては、ライセンスをCloverETL Server自体の前にロードする必要があります。

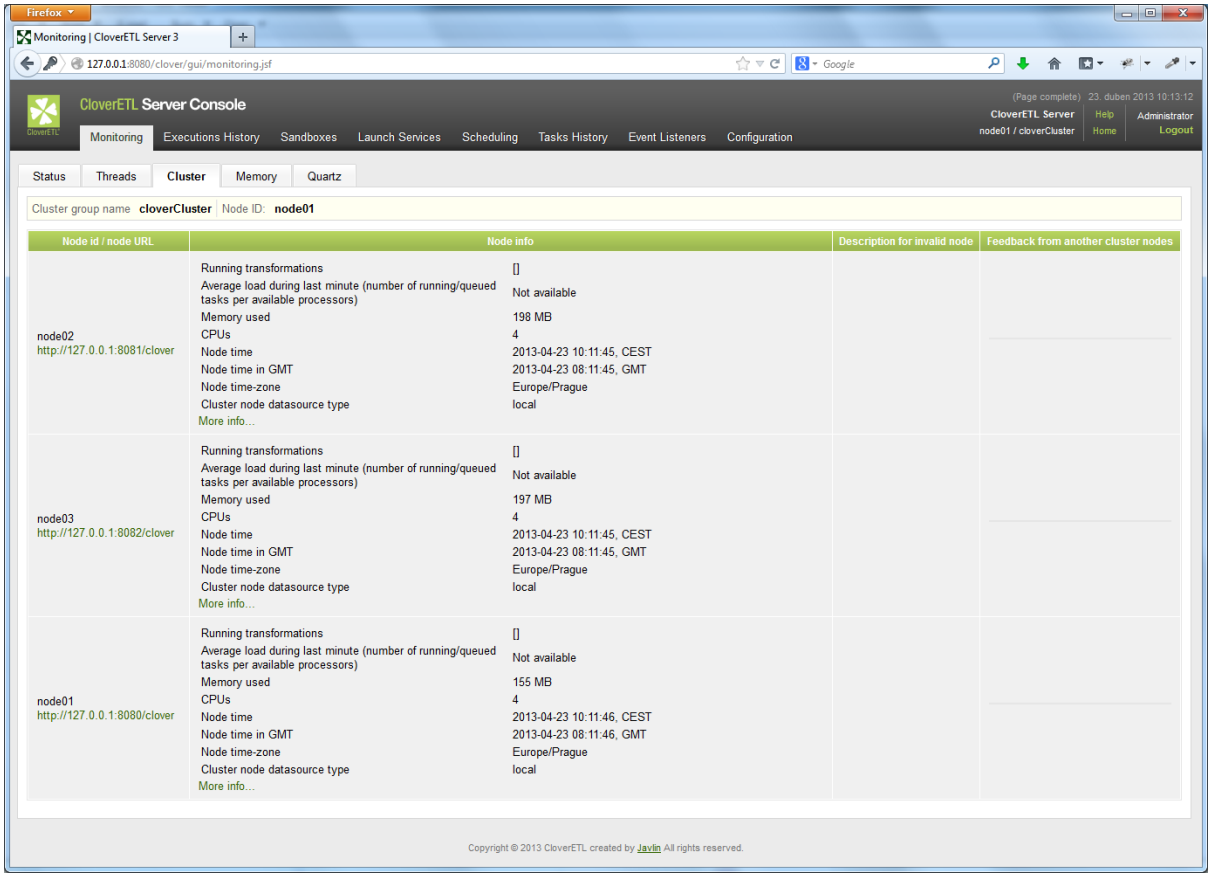
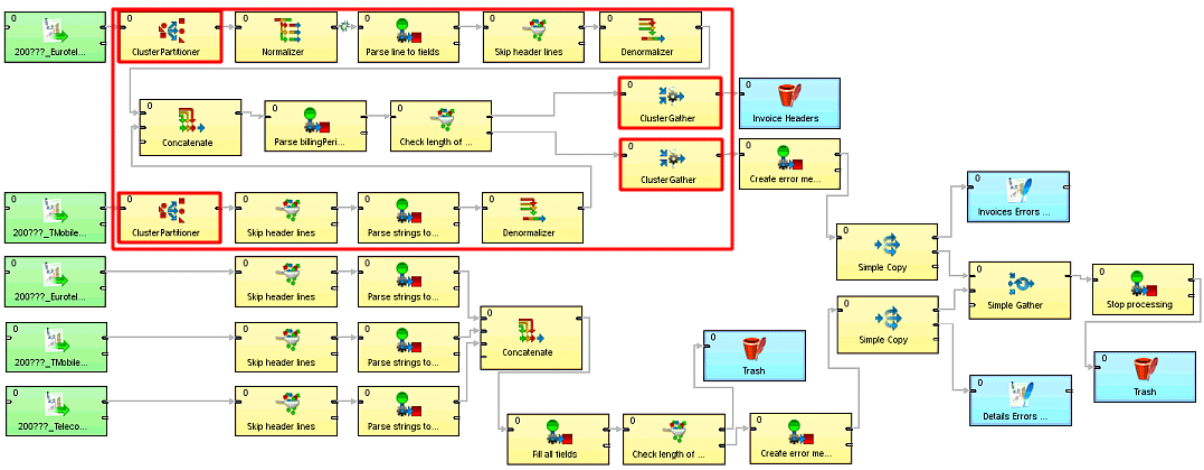


図24.6 クラスタに結合されたノードのリスト

分散実行の例

次の図は、チェコ共和国内の複数の携帯電話ネットワークのプロバイダによって生成される請求書の解析に使用される変換グラフを示しています。



これらの入力ファイルのサイズは最大で数GBになる可能性があるため、クラスタ環境で動作するようグラフを設計することは非常に有用です。

変換設計例の詳細

グラフ内には4つのクラスタ・コンポーネントがあります。これらのコンポーネントによって、変更ポイントであるノード割当てが定義されるため、これらのコンポーネントにより区切られるグラフ部分は赤い四角で強調表示されています。これらのコンポーネントの割当てはパラレルで実行される必要があります。つまり、点線の四角形の内側にあるコンポーネントには、効率的な割当てが指定される必要があります。グラフの残りの部分は、1つのノード上でのみ実行されます。

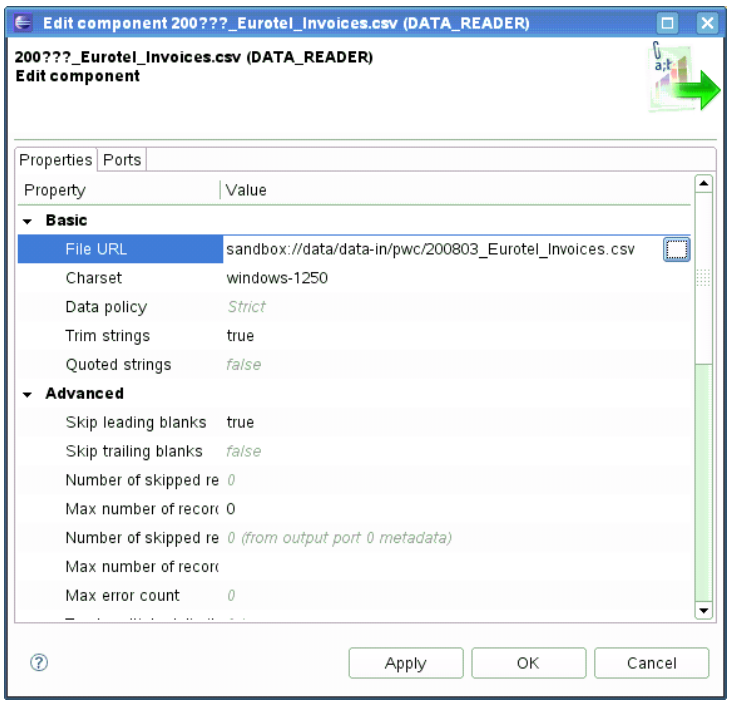
ノード割当ての指定

グラフでは、次の2つのノード割当てが使用されています。

- パラレルで実行されているコンポーネントのノード割当て(4つのクラスタ・コンポーネントによる境界の区切り)。

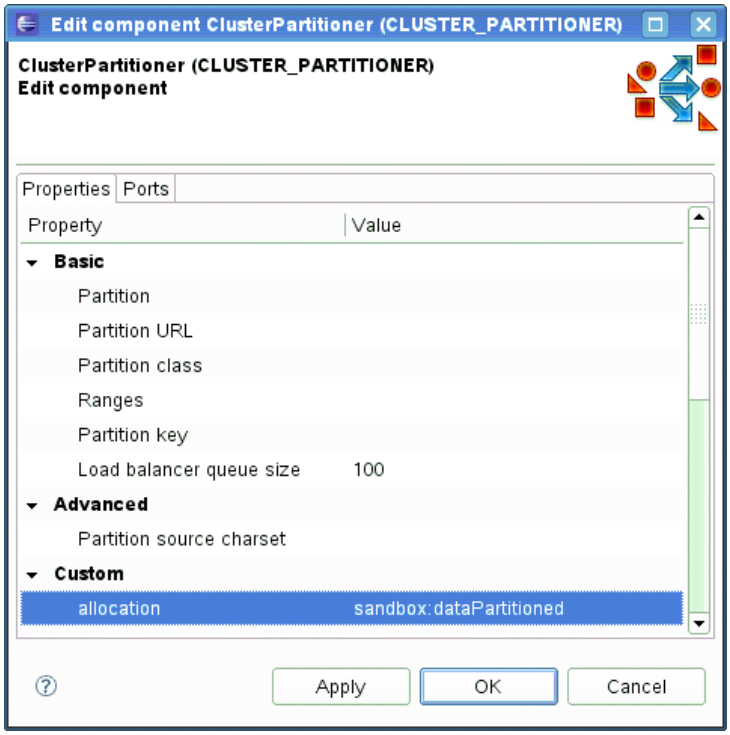
- 単一ノードで実行される、グラフの外側部分のノード割当て。

単一ノードは、入力データのURLで使用されるサンドボックス・コードによって指定されます。次のダイアログでは、ファイルURLの値に"sandbox://data/path-to-csv-file"が示されています。"data"は、指定したファイルが置かれているサーバー・サンドボックスのIDです。この"data"のローカル・サンドボックスにより単一ノードが定義されます。



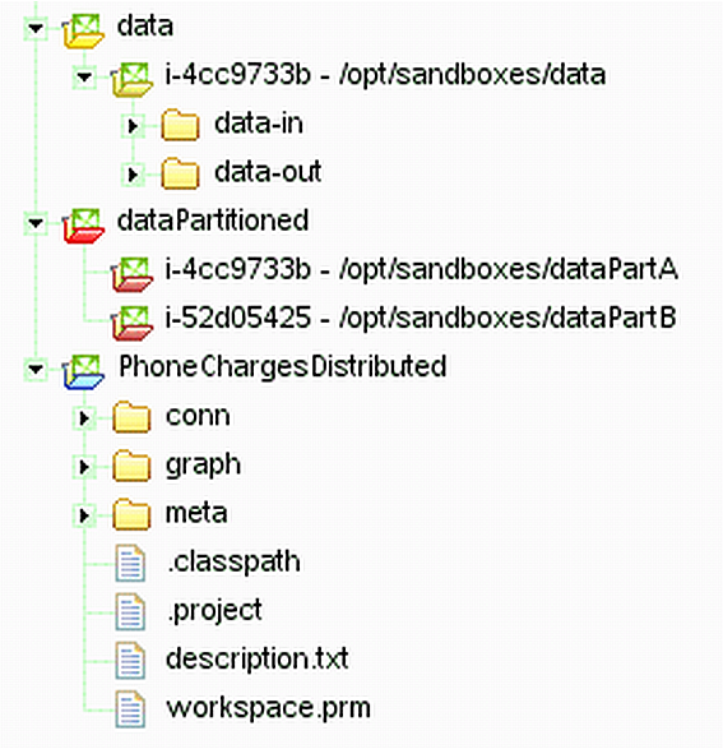
4つのクラスタ・コンポーネントによって区切られたグラフ部分では、ファイルのURL属性によって割当てを指定することもできますが、この部分はファイルを使用して機能するのではないため、ファイルURLは存在しません。したがって、ノード割当て属性を使用します。各コンポーネントは隣接コンポーネントから割当てを採用できるため、割当ては1つのコンポーネントに対してのみ設定すれば十分です。

次のダイアログの"dataPartitioned"もサンドボックスIDです。



これらのサンドボックスについて詳しく見てみましょう。このプロジェクトには、"data"、"dataPartitioned"および"PhoneChargesDistributed"という3つのサンドボックスが必要です。

- data
  - 入力および出力データが含まれます
  - ローカル・サンドボックス(黄色いフォルダ)であるため、物理的な場所が1つのみ含まれます
  - 指定したパス内のノード"i-4cc9733b"でのみアクセスできます
- dataPartitioned
  - パーティション化されたサンドボックス(赤いフォルダ)であるため、異なるノード上に物理的な場所のリストがあります
  - データが含まれず、グラフではこのサンドボックスで読取りまたは書込みが行われないため、ノード割当ての定義用としてのみ使用されます
  - 次の図では、2つのクラスタ・ノードに対して割当てが構成されています
- PhoneChargesDistributed
  - グラフ・ファイル、メタデータおよび接続を含む一般的なサンドボックス
  - 共有サンドボックス(青いフォルダ)であるため、すべてのクラスタ・ノードが同じファイルにアクセスできます



前の図のサンドボックス構成を使用してグラフが実行された場合、ノード割当ては次のようになります。

- 単一ノード上でのみ実行されるコンポーネントは、"data"サンドボックスの場所に応じて"i-4cc9733b"ノードでのみ実行されます。
- "dataPartitioned"サンドボックスに応じた割当てを持つコンポーネントは、"i-4cc9733b"および"i-52d05425"ノード上で実行されます。

変換例のスケーラビリティ

変換例は、Amazon Cloud環境ですべての実行に対して次の条件を使用してテストされています。

- 同じマスター・ノード
- 同じ入力データ: 1、2GBの入力データ、2700万のレコード
- ノード割当てごとに3回の実行
- 2回の実行ごとにノード割当ての変更
- ノードはすべて"c1.medium"タイプ

ノード割当てのカーディナリティは、1つのノードから始めて最大8ノードまでテストしました。

次の図は、クラスタ内のノード数に対するランタイムの機能の依存性を示しています。

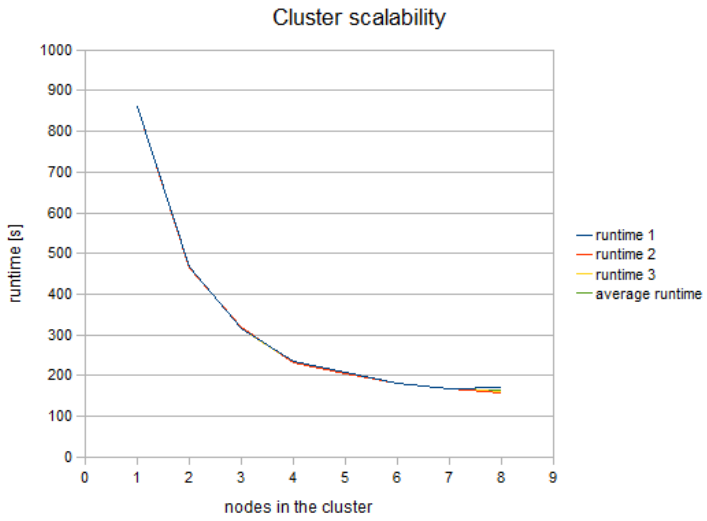


図24.7 クラスタのスケーラビリティ

次の図は、クラスタ内のノード数に対する加速要因の依存性を示しています。加速要因は、1つのクラスタ・ノードを使用した場合の平均ランタイムとx個のクラスタ・ノードを使用した場合の平均ランタイムの比率です。つまり、次のようになります。

`speedupFactor = avgRuntime(1 node) / avgRuntime(x nodes)`

この結果から、最大4ノードが効率的であることがわかります。ノードを追加するたびにクラスタのパフォーマンスは向上しますが、向上の効果は減少します。9個以上のノードを使用すると、マイナスの影響も現れています。これは、これらのノードの管理によるオーバーヘッドのためにパフォーマンス上のメリットが失われるためです。

これらの結果は、変換ごとに固有のものであるため、変換の機能曲線が改善する場合や、あるいは悪化する場合もあります。

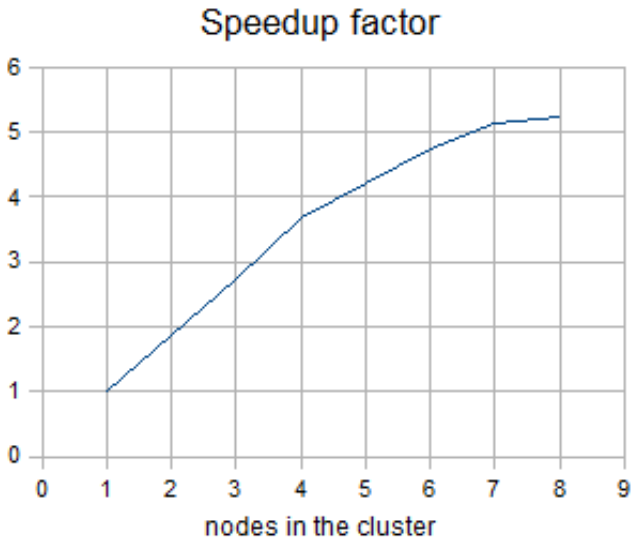


図24.8 加速要因

ランタイムの測定結果表:

ノード数	ランタイム1 [s]	ランタイム2 [s]	ランタイム3 [s]	平均ランタイム [s]	加速要因
1	861	861	861	861	1

ノード数	ランタイム1 [s]	ランタイム2 [s]	ランタイム3 [s]	平均ランタイム [s]	加速要因
2	467	465	466	466	1.85
3	317	319	314	316.67	2.72
4	236	233	233	234	3.68
5	208	204	204	205.33	4.19
6	181	182	182	181.67	4.74
7	168	168	168	168	5.13
8	172	159	162	164.33	5.24

クラスタ構成

クラスタが正しく機能するのは、各ノードが正しく構成されている場合のみです。クラスタリングが有効であり、各ノードのnodeIDが一意であり、すべてのノードが共有DB(直接接続または別のクラスタ・ノードによるプロキシ)および共有サンドボックスにアクセスでき、ノード間の協調性に関するすべてのプロパティがネットワーク環境に応じて設定されていることが必要です。

プロパティおよび使用可能な構成は、次のとおりです。

- 「必須プロパティ」
- 「オプションのプロパティ」
- 「2ノードのクラスタ構成の例」
- 「ロード・バランシングのプロパティ」

必須プロパティ

必須クラスタ・プロパティ以外にも、license.fileおよびクラスタ環境とは特に関係のない他の必須プロパティを設定する必要があります。データベース接続の構成も必須ですが、直接接続のほかに、別のクラスタ・ノード(単一または複数)を使用してプロキシを構成することもできます。詳細は、"cluster.datasource.type"プロパティを参照してください。

表24.1 必須プロパティ - クラスタの各ノードで正しく設定する必要があるプロパティ

プロパティ	タイプ	デフォルト
cluster.enabled	boolean	false
説明:	サーバーがクラスタに接続されるかどうかを切り替えます	
cluster.node.id	String	node01
説明:	各クラスタ・ノードには一意のIDが必要です	
cluster.shared_sandboxes_path	String (パス)	
説明:	パス(すべての共有サンドボックスがこのノードに格納されます)。クラスタが有効な場合、ジョブ・ファイル(グラフ、メタデータ、接続構成など)を含んでいるすべてのサンドボックスは共有される必要があるため、サンドボックスごとに個別の"rootPath"を指定するのではなく、"cluster.shared_sandboxes_path"で指定したファイル・システム・フォルダにすべての共有サンドボックスが格納されます。サンドボックスのルート・ディレクトリへのパスは[shared_sandboxes_path]/[sandboxID]のように構築されま	

プロパティ	タイプ	デフォルト
	す。また、共有が適切に構成される必要があるため、ディレクトリの内容にはすべてのクラスタ・ノードのファイルシステムからアクセスできます。	
cluster.jgroups.bind_address	String (IPアドレス)	127.0.0.1
説明:	別のクラスタ・ノードとの通信に使用される、イーサネット・インタフェースのIPアドレス。ノード間のメッセージングに必要です。	
cluster.jgroups.start_port	int (ポート)	7800
説明:	jGroupsサーバーがノード間のメッセージをリスニングするポート。	
cluster.jgroups.tcpping.initial_hosts	String (形式: "IPaddress1[port1],IPaddress2[port2]")	127.0.0.1[7800]
説明:	ノードを実行およびリスニングするIPアドレス(およびポート)のリスト。これは、"bind_address"および"start_port"という別のノードのプロパティに関連付けられます。例: bind_address1[start_port1],bind_address2[start_port2],... クラスタのすべてのノードをリストする必要はありませんが、リストされたhost:portの少なくとも1つが実行されている必要があります。ノード間のメッセージングに必要です。	
cluster.http.url	String (URL)	http://localhost:8080/clover
説明:	CloverETLクラスタ・ノードのURL。Webアプリケーションのルートへの HTTP/HTTPS URLである必要があるため、通常は、"http://[hostname]:[port]/clover"です。これは主に、他のクラスタ・ノードからの同期ノード間通信に使用されます。クライアント・ブラウザやCloverETL Designerからアクセスできるように、完全修飾ホスト名またはIPアドレスを使用することをお薦めします。	

オプションのプロパティ

表24.2 オプションのプロパティ - クラスタ構成に必須ではないプロパティ(デフォルト値で十分)

プロパティ	タイプ	デフォルト	説明
cluster.node.sendinfo.interval	int	5000	ミリ秒単位の時間間隔; 各ノードにより、それ自体に関する情報が別のノードに送信されます。この間隔により、情報が設定される頻度が指定されます
cluster.node.remove.interval	int	15000	ミリ秒単位の時間間隔; この間隔内にノード情報が受信されない場合、ノードは失われたとみなされ、クラスタから削除されます
cluster.max_allowed_time_shift_between_nodes	int	2000	ノード間の最大許容時間シフト。すべてのノードでシステム時間が同期化されている必要があります。そうでない場合、クラスタが正しく機能しない可能性があります。このため、このしきい値を超えると、ノードは無効として設定されます。
cluster.group.name	String	clover Cluster	クラスタごとに一意のグループ名があります。同じネットワーク環境内に2つのクラ

プロパティ	タイプ	デフォルト	説明
			スタが必要な場合、これらのクラスタごとに独自のグループ名が設定されます。
cluster.datasource.type	String	local	ノードにCloverETL Serverデータベースへの直接接続がない場合は、このプロパティを"remote"に変更してください。これにより、他のクラスタ・ノードをプロキシとして使用して永続操作を処理するようになります。その場合は、プロパティ "cluster.datasource.delegate.nodeIds" も適切に構成する必要があります。Properties jdbc.*は無視されます。なお、スケジューラは直接接続を持つノードでのみアクティブになります。
cluster.datasource.delegate.nodeIds	String		永続操作を処理するためにこのノードがプロキシとして使用できるクラスタ・ノードIDのリスト(カンマ","区切り)。リストされたノードIDの少なくとも1つが実行されている必要があります(そうでない場合、このノードは失敗します)。リストされたすべてのノードIDに、CloverETL Serverデータベースへの直接接続が適切に構成されている必要があります。プロパティ"cluster.datasource.delegate.nodeIds"はデフォルトでは無視されます。 プロパティ"cluster.datasource.type"は、この機能を有効にするために"remote"に設定される必要があります。

2ノードのクラスタ構成の例

この項には、CloverETLクラスタ・ノード構成の例が含まれます。また、次を構成することが必要です。

- ディレクトリ/home/clover/nfs_shared/sandboxesの共有またはレプリケーション
- 両ノードからの同じデータベースに対する接続
- HTTPロード・バランサ

192.168.1.131でのノードの構成

```
jdbc.dialect=org.hibernate.dialect.MySQLDialect
datasource.type=JNDI
datasource.jndiName=java:comp/env/jdbc/clover_server

cluster.enabled=true
cluster.node.id=node01
cluster.shared_sandboxes_path=/home/clover/nfs_shared/sandboxes

license.file=/home/clover/license/license.dat

cluster.group.name=cloverCluster
cluster.jgroups.bind_address=192.168.1.131
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.13[7800]

cluster.http.url=http://192.168.1.131:8080/clover
```

192.168.1.13でのノードの構成

```
       # this node will use node01 as proxy for its persistent operations above shared
DB
       cluster.datasource.type=remote cluster.datasource.delegate.nodeIds=node01

          # direct connection to the shared DB isn't configured
          #jdbc.dialect=org.hibernate.dialect.MySQLDialect #datasource.type=JNDI
          #datasource.jndiName=java:comp/env/jdbc/clover_server

       cluster.enabled=true
       cluster.node.id=node02
       cluster.shared_sandboxes_path=/home/clover/nfs_shared/sandboxes

       license.file=/home/clover/license/license.dat

       cluster.group.name=cloverCluster
       cluster.jgroups.bind_address=192.168.1.13
       cluster.jgroups.start_port=7800
       cluster.jgroups.tcpping.initial_hosts=192.168.1.131[7800]

       cluster.http.url=http://192.168.1.13:8080/clover
```

ロード・バランシングのプロパティ

ロード・バランシング基準の乗法子です。ロード・バランサにより、グラフを実行するクラスタ・ノードが決定されます。つまり、任意のノードが実行のためのリクエストを処理できますが、グラフはノードの現在の負荷およびこれらの乗法子に応じて同じノードまたは異なるノード上で実行されます。

数値が大きいほど、決定に対する関連性が高くなります。乗法子はすべて0より大きい必要があります。

クラスタのノードごとにロード・バランシングのプロパティが異なる場合があります。任意のノードが変換実行のための受信リクエストを処理でき、各ノードが独自の構成に応じて異なる方法でロード・バランシングの基準を適用できます。

これらのプロパティはクラスタ構成に必須ではありません。デフォルト値で十分です

表24.3 ロード・バランシングのプロパティ

プロパティ	タイプ	デフォルト	説明
cluster.lb.balance.running_graphs	float	3	ロード・バランシング用の実行グラフの重要度を指定します。
cluster.lb.balance.memused	float	0.5	ロード・バランシング用の使用メモリーの重要度を指定します。
cluster.lb.balance.cpus	float	1.5	ロード・バランシング用のCPU数の重要度を指定します。
cluster.lb.balance.request_bonus	float	2	ノードが実行のためのリクエストを処理するものと同じノードである、という事実の重要度を指定します。これは、グラフを実行する場所を決定する同じノードです。この乗法子に十分大きい値を指定すると、実行のためのリクエストを処理するノードと同じノードでグラフが常に実行されるようになります。
cluster.lb.balance.node_bonus	float	1	構成されたノードの総比率ボーナス。1より大きい値を指定すると、

プロパティ	タイプ	デフォルト	説明
			ノードがロードバランサによって選択される可能性が高まります。値1は、ボーナスもペナルティもないことを意味します。0は、そのノードがロードバランサによって一切選択されなくなることを意味します。ただし、グラフを実行することはできます(たとえば、クラスタ内に他のノードがない場合や、グラフがそのノードで実行されるように設計されている場合)。

第25章 一時領域管理

CloverETL Serverで使用可能なコンポーネントの多くが正しく機能するには、一時ファイルまたはディレクトリが必要です。一時領域は、これらのファイルまたはディレクトリが作成および保持されるローカル・ファイルシステム上の物理的な場所です。

概要

CloverETL Serverで定義されている一時領域の概要は、構成→一時領域管理→概要で参照できます。

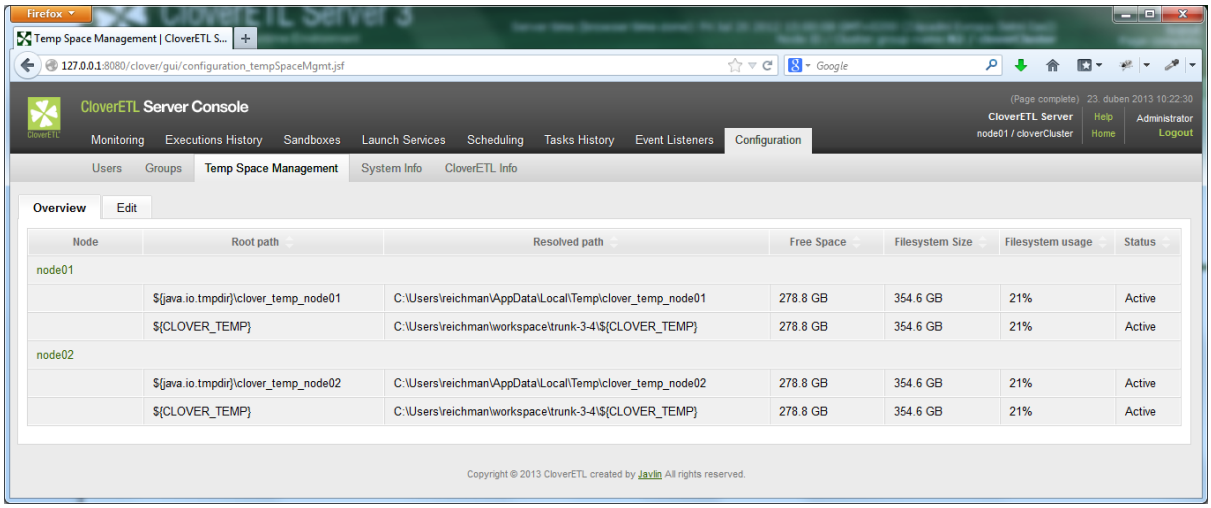


図25.1 構成された一時領域の概要 – クラスタ・ノードごとに1つのデフォルト一時領域

設定

一時領域管理には、一時領域を追加、一時停止、再開および削除するためのインタフェースが用意されています。これは、構成→一時領域管理→編集からアクセスできます。

画面は、グローバル構成とノード構成当たりという2つのドロップダウン領域に分かれています。グローバル構成では、スタンドアロン・サーバーの一時領域を管理します。また、サーバー・クラスタの場合は、すべてのノード上の一時領域を管理します。ノード構成当たりを使用すると、ノードごとに一時領域を個別に保持できます。

初期化

CloverETL Serverが起動すると、一時領域構成が確認されます。一時領域が構成されていない場合、**java.io.tmpdir**システム・プロパティが指し示すディレクトリ内にデフォルトの一時領域が新しく作成されます。ディレクトリの名前は、次のとおりです。

- **${java.io.tmpdir}/clover_temp**（スタンドアロン・サーバーの場合）
- **${java.io.tmpdir}/clover_temp_<node_id>**（サーバー・クラスタの場合）

一時領域の追加

新規一時領域を定義するには、表内の最後の行の下にあるテキスト・フィールドにパスを入力し、追加リンクをクリックします。パス内では環境変数およびシステム・プロパティを使用できます(例: **${VARIABLE_NAME}/temp_space**)。入力したディレクトリが存在しない場合は作成されます。

注記

環境変数は、同じ名前のシステム・プロパティより優先されます。変数が含まれるパスが解決されるのは、新規一時領域が追加された後のサーバーの起動中です。変数値が変更されている場合は、サーバーを再起動してこのような変更を有効にする必要があります。

ヒント

一時領域を追加する主な目的は、システム・スループットを高めることにあります。したがって、I/Oのパフォーマンスを最大限に高めるには、入力したパスが別の物理デバイス上にあるディレクトリを指し示している必要があります。

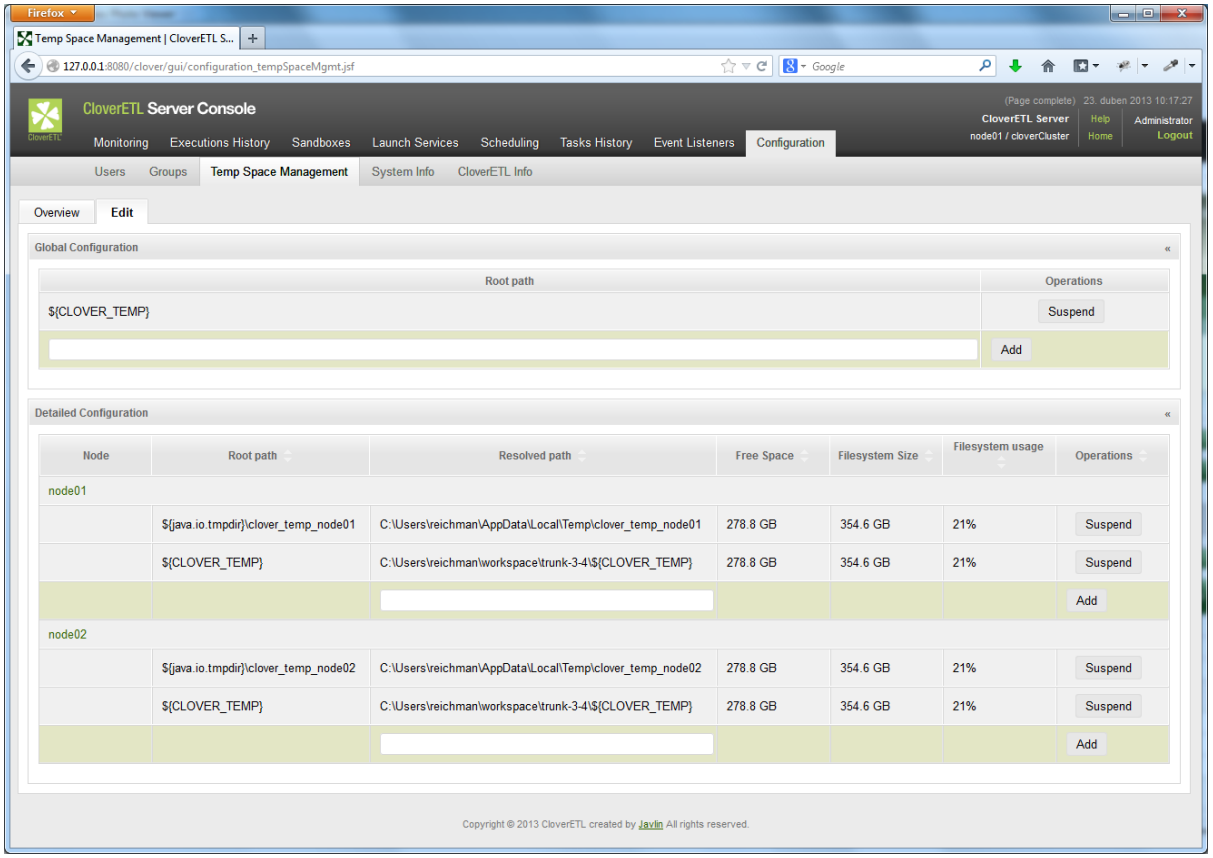


図25.2 両ノードに設定された環境プロパティを使用して新しく追加されたグローバル一時領域。

一時領域の一時停止

一時領域を一時停止するには、パネル内の一時停止リンクをクリックします。前または現在のグラフ実行から残されたファイルがある場合、通知が表示されます。一時領域が一時停止された後は、一時領域で一時ファイルが新規作成されなくなりますが、すでに作成されているファイルはそれまでどおり実行ジョブによる使用が可能です。

注記

一時領域は少なくとも1つがアクティブである(一時停止されていない)よう保持されます。

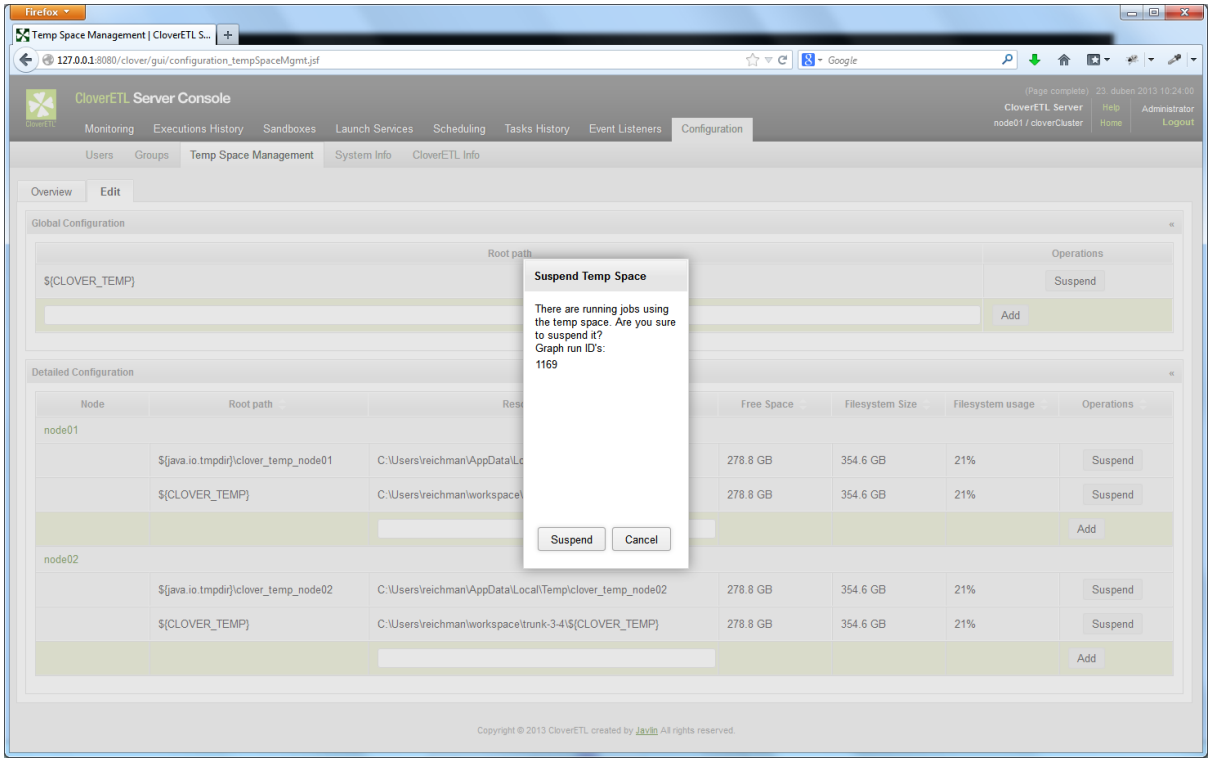


図25.3 実行中のジョブのデータが存在する場合に一時停止操作によって求められる確認。

一時領域の再開

一時領域を再開するには、パネル内の再開リンクをクリックします。再開された一時領域はアクティブです。つまり、一時ファイルおよびディレクトリの作成用として使用できます。

一時領域の削除

一時領域を削除するには、パネル内の削除リンクをクリックします。一時停止されている一時領域のみを削除できます。一時領域を使用している実行中のジョブが存在する場合、削除は許可されません。一時領域ディレクトリ内にファイルが残されている場合、表示された通知パネルでこれらを削除できます。使用可能なオプションは、次のとおりです。

- 削除 – システムから一時領域を削除しますが、内容は保持します
- 削除および抹消 – システムから一時領域とその内容を削除します
- 取消 – 操作を続行しません

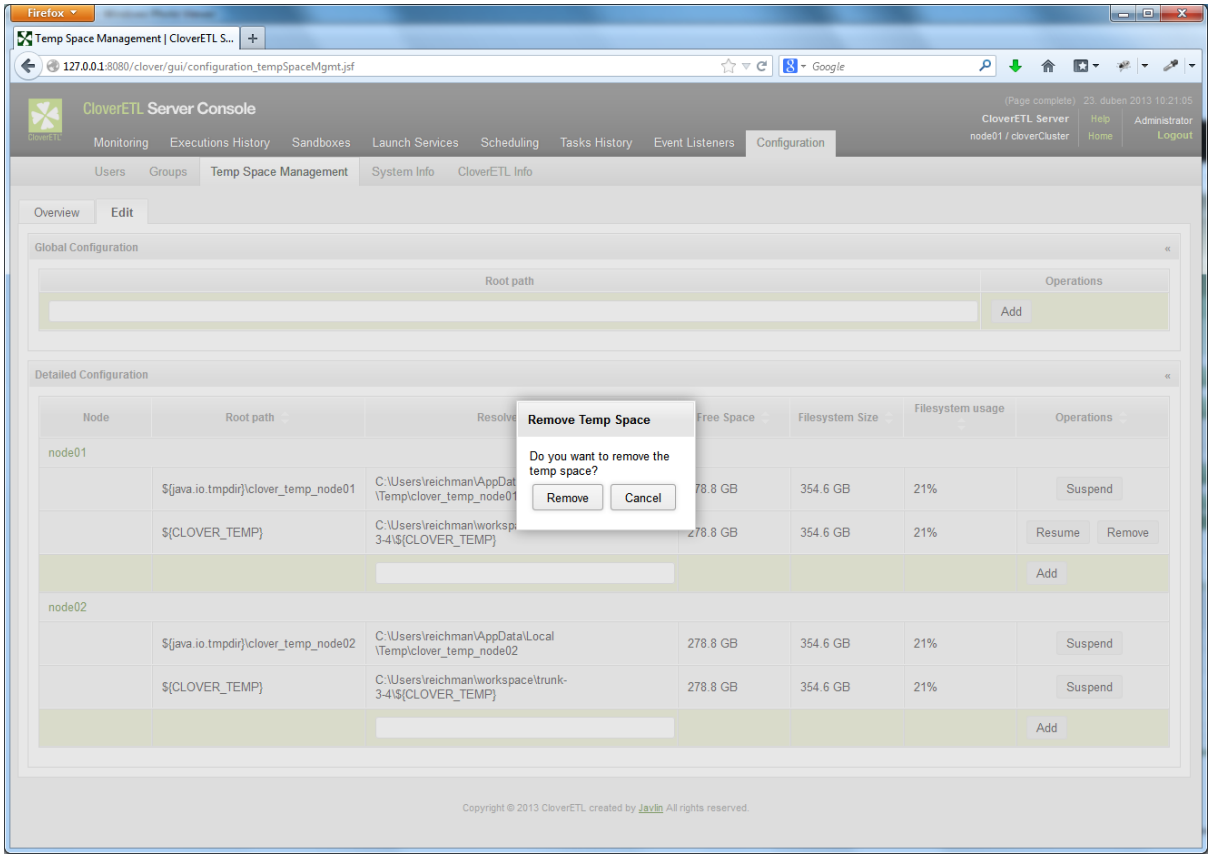


図25.4 一時領域内にデータが存在する場合に削除操作によって求められる確認。