SYBASE®
An SAP® Company

管理者ガイド

Sybase Event Stream Processor 5.0

ドキュメント ID： DC01740-01-0500-01
改訂： 2011 年 12 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

第 1 章　基本的な管理作業

基本的な管理作業では、ユーザのために最小限の Sybase® Event Stream Processor 環境を準備します。

サーバの起動など、スタジオ内からアクセスできる管理作業もありますが、スタジオは、モニタ・ツールとしてではなく、開発環境として使用することを目的としています。したがって、Sybase Event Stream Processor のほとんどの管理では、コマンド・ライン・ユーティリティを使用するか、ファイルを直接操作することが必要です。

ユーザのプロビジョニング

ユーザのプロビジョニングは標準的な管理作業です。Event Stream Processor は既存のセキュリティ・インフラストラクチャと統合してユーザを認証し、ユーザにプロジェクトへのアクセス権を付与します。

ユーザが Event Stream Processor で認証されるのは ESP クラスタが設定されている場合のみです。プロジェクトをスタンドアロン・モードでコマンド・ラインから実行することもできますが、これはおすすめしません。スタンドアロン・モードではセキュリティが適用されません。

Event Stream Processor クラスタがサポートする認証メカニズムは、次のとおりです。

- セキュリティなし
- RSA
- Kerberos
- LDAP

参照：

- *第 3 章、「Event Stream Processor のセキュリティ」* (41 ページ)

ロギング

ログ・ファイルはクラスタ・サーバ・レベルとプロジェクト・レベルで設定します。1 つのクラスタ・ノードには複数のプロジェクトを入れることができ、プロジェクトごとに独自のプロジェクト・ログがあります。

クラスタ・ノード・ログの設定ファイル

ノード・ログの設定ファイルは `cluster.log.properties` という **log4j** プロパティ・ファイルです。

クラスタ内のノードごとにクラスタ・ノード・ログの設定ファイルを 1 つ作成します。このファイルは、通常ノードが起動されるディレクトリに作成します。たとえば、ノードのベース・ディレクトリは `%ESP_HOME%¥cluster¥projects¥<cluster-name>` (Windows) または `$ESP_HOME/cluster/projects/<cluster-name>` (Linux/Solaris) です。

インストール時にクラスタを設定した場合、デフォルトでは、クラスタ・ノード・ログの設定ファイルは `<ESP_HOME>/cluster/nodes/<node-name>` にあります。そのクラスタでプロジェクトを実行すると、ベース・ディレクトリは `<ESP_HOME>/cluster/projects/<node-name>` に変更されます。追加クラスタを設定する場合は、クラスタ・ノードの保存先のベース・ディレクトリを変更できます。クラスタ・ノード・ログの設定ファイルはノードの設定ファイルと同じフォルダに置くようにします。

次に、`cluster.log.properties` のサンプル・ファイルを示します。

```
com.sybase.esp.cluster.logfile=cluster.log

log4j.rootLogger=info, A

log4j.appender.A=org.apache.log4j.RollingFileAppender
log4j.appender.A.File=${com.sybase.esp.cluster.logfile}
log4j.appender.A.MaxFileSize=1MB
log4j.appender.A.MaxBackupIndex=5
log4j.appender.A.layout=org.apache.log4j.EnhancedPatternLayout
log4j.appender.A.layout.ConversionPattern=%d{MMM dd yyyy
HH:mm:ss.SSS} %p %t %c - %m%n


log4j.logger.com.sybase.esp=info
log4j.logger.com.sybase.esp.cluster.applications=info

log4j.additivity.com.sybase.esp.cluster.applications=false


.level=INFO
handlers=com.sybase.esp.cluster.impl.Log4JHandler
com.sybase.esp.cluster.impl.Log4JHandler.level=FINEST
```

クラスタ・ノードのログはファイル `node_one.log` に書き込まれます。このサンプル設定では、ログ・ファイルはサイズが 1MB に達したらファイルの内容をバックアップするように設定されています。`MaxBackUpIndex` オプションで、作成するバックアップ・ファイルの数を指定します。

rootLogger オプションと logger.com.sybase.esp オプションは error または info に設定できます。info の場合、生成されるログ情報は最小です。通常の状況下では、rootLogger オプションはデフォルト値の info に設定したままにしておきます。そうしないと、ログはそのサイズのためにほとんど読み取ることができなくなります。logger.com.sybase.esp を使用すれば、サードパーティのデバッグ・コンポーネントを使用せずにノードをデバッグできます。log4j.logger.com.sybase.esp.cluster.applications プロパティは変更しないでください。この場合は info 値が必要です。

サポートされるプロパティと設定手順の詳細については、**log4j** の資料を参照してください。

プロジェクト・ログ

プロジェクト・ログを設定して、実行中のプロジェクトのエラーを収集します。クラスタ内の 1 つまたは複数のプロジェクトのログを設定できます。

Event Stream Processor では、プロジェクトはローカル・クラスタとリモート・クラスタで実行されます。プロジェクト・ログは、配備するクラスタのタイプによって異なるディレクトリに格納されます。

ローカル・クラスタは ESP スタジオでテスト環境として使用されます。プロジェクト・ログ・ファイル (esp_server.log) を含め、ローカル・クラスタ内のプロジェクトで生成されたファイルは <ESP_HOME>¥SybaseESP¥5.0¥workspace¥<workspace-name>.<project-name>.<instance-number> にあります。

リモート・クラスタ・ノードには、ノード固有のベース・ディレクトリが必要です。ノードのベース・ディレクトリは <ESP_HOME>/cluster/projects/<node-name> ですが、このパスは変更できます。これはプロジェクト作業ディレクトリの親ディレクトリで、各プロジェクト固有のプロジェクト・ログ・ファイル (esp_server.log) があります。CCL に指定された相対パスはすべて、プロジェクト作業ディレクトリを基準とします。

プロジェクトのロギング・レベルはプロジェクト設定ファイル (.ccr) で変更するか、スタジオのプロジェクト設定エディタを使用して変更します。詳細については、『スタジオ・ユーザーズ・ガイド』を参照してください。

参照：

- *クラスタの設定* (29 ページ)

ロギング・レベル

ロギング・レベルの範囲は 0 ～ 7 で、値が大きいほど重大度は低くなります。プロジェクトのデフォルトのロギング・レベルは 4 です。

名前	レベル	説明
LOG_EMERG	0	/* system is unusable */
LOG_ALERT	1	/* action must be taken immediately */
LOG_CRIT	2	/* critical conditions */
LOG_ERR	3	/* error conditions */
LOG_WARNING	4	/* warning conditions */
LOG_NOTICE	5	/* normal but significant condition */
LOG_INFO	6	/* informational */
LOG_DEBUG	7	/* debug-level messages */

データのバックアップとリストア

データのオフラインまたはオンライン・バックアップを実行します。ご使用のシステムに基づいてデータを表示し、リストアします。

オフライン・バックアップがデータ・バックアップの優先方法です。ただし、プロジェクト・サーバの実行中もバックアップできます。

バックアップ・ファイルの内容を表示するには、**tar -tvf backup.tar** コマンドまたは **pkunzip -v backup.zip** コマンドを使用します。

Linux、Solaris、Windows の各システムでは、バックアップ・ファイルからデータをリストアします。

参照：

- *ログ・ストアのサイズ設定* (8 ページ)

データのバックアップ

バックアップを実行してデータを管理し、保護します。Sybase Event Stream Processor の管理対象データを定期的にバックアップします。バックアップは、Event Stream Processor がオフライン (推奨) でもオンラインでも実行できます。

バックアップするデータは、次のとおりです。

データ	説明
プロジェクト	`.ccl` ファイルと `.ccr` ファイルをバックアップします。`.ccx` ファイルと `.cclnotation` ファイルは再生成できます。
ログ・ストア	ストリームの内容をログ・ストアに書き込むようにストリームを設定します。バックアップしておけば、システム障害の発生後にデータをリストアできます。

オフライン・バックアップを使用することをおすすめします。ただし、プロジェクト・サーバの実行中もプロジェクトとログ・ストアをバックアップできます。これをオンライン・バックアップと呼びます。オフライン・バックアップは、すべてのマシン上で実行され、オンライン・バックアップより短時間のため、オフライン・バックアップをおすすめします。オンライン・バックアップの場合のように、データをリストアするときにファイル名とファイルの拡張子を個別に変更する必要はありません。

注意： すべてのファイルが確実にバックアップ・セットに追加されるようにします。一般に、これらのファイルは名前は同じですが拡張子が異なり、すべて同じディレクトリに格納されます。プロジェクトと関連ログ・ストアがすべて確実にバックアップされるようにします。

参照：
- *データのリストア*(7 ページ)
- *ログ・ストアのサイズ設定*(8 ページ)

オフライン・バックアップの実行
プロジェクト・サーバが実行していない間にログ・ストアのバックアップを実行します。

前提条件
Event Stream Processor をシャットダウンする前に、プロジェクト・ファイルのロケーションと、ストリームごとに定義されているストアのタイプを確認します。プロジェクトをパブリッシュしたりサブスクライブしているデータ・ストリームがないことを確認します。

手順

1. 次のコマンドを使用してプロジェクトを停止します。
   ```
   $ESP_HOME/bin/esp_cluster_admin --uri=esp://cluster_server:19011
   --username=me --password=sybase --stop project <workspace-name>/
   <project-name> [<instance-index>]
   ```

port には、ご使用の Event Stream Processor インストール環境で使用されているポート番号を入力します。 *espuser* と *password* には、ユーザ・クレデンシャルを入力します。

2. 各プロジェクトの `.ccl` と `.ccr` の各ファイルと、その関連するログ・ストアをバックアップします。Linux システムと Solaris システムの場合は、**tar** システム・ユーティリティを使用します。Windows システムの場合は、**pkzip** フリーウェア・ユーティリティまたは同等のものを使用します。パス `<base-directory>/<workspace-name>.<project-name>.<instance-number>` を使用して、個々のログ・ストアをバックアップします。`<base-directory>/<workspace-name>.<project-name>.<instance-number>` フォルダを使用して、プロジェクト内のすべてのログ・ストアをバックアップします。

参照：
- *データのリストア*(7 ページ)
- *ログ・ストアのサイズ設定*(8 ページ)

オンライン・バックアップの実行

プロジェクト・サーバの実行中にログ・ストアのバックアップを実行します。オンライン・バックアップに、配備済みプロジェクトは含まれません。

注意： オンライン・バックアップのときにプロジェクト・サーバを停止する必要はありませんが、バックアップ・ファイルの作成中は操作がサスペンドされ、短い中断が発生することがあります。この中断の長さは、ログ・ストアに蓄積されているデータの量によって異なります。オンライン・バックアップは、短い中断が発生しても問題ない場合のみ実行してください。

コマンド・プロンプトで **esp_client** ユーティリティを使用して、ログ・ストア・ファイルのバックアップ・コピーを作成します。例を示します。

```
$ESP_HOME/bin/esp_client -p <host>:<port>/<workspace-name>/<project-name> -c espuser[:password] backup
```

<host>:<port> には、ご使用の Event Stream Processor インストール環境で使用されているホスト名とポート番号を入力します。 *espuser* と *password* には、ユーザ・クレデンシャルを入力します。
これで、ログ・ストア・ディレクトリに一連のバックアップ・ファイルが作成されます。各ファイルの拡張子は `.bak` です。ストアの現在の内容のみがコピーされます。

参照：
- *データのリストア*(7 ページ)
- *ログ・ストアのサイズ設定*(8 ページ)

バックアップ・ファイルの表示

バックアップ・ファイルを表示するには、**tar -tvf backup.tar** または **pkunzip -v backup.zip** を使用します。

- Linux システムと Solaris システムの場合は、**tar -tvf backup.tar** コマンドを使用します。
- Windows システムの場合は、**pkunzip -v backup.zip** コマンドを使用します。

参照：

- *データのリストア*(7 ページ)
- *ログ・ストアのサイズ設定* (8 ページ)

データのリストア

データをリストアするには、バックアップ・ファイルからログ・ストアの内容を抽出します。

オンライン・バックアップ時に作成されたファイルをリストアするには、`.bak` ファイルの名前を変更する必要があります。 オフライン・バックアップの場合は、リストアするファイルの名前を変更する必要はありません。

Linux システムと Solaris システムの場合は、**tar -xvf** コマンドを使用してバックアップ・ファイルを抽出します。Windows システムの場合は、**WinZip** コマンドまたは **pkunzip** コマンドのどちらかを使用します。

参照：

- *データのバックアップ*(4 ページ)
- *ログ・ストアのサイズ設定* (8 ページ)

Linux システムと Solaris システムでのバックアップ・ファイルのリストア

Linux システムと Solaris システムでバックアップ・ファイルからファイルをリストアするには、**tar -xvf** コマンドを使用します。

1. バックアップ・ファイルからリストアするファイルを抽出します。
   たとえば、`backup.tar` からファイルを抽出するには、次のコマンドを使用します。

   ```
   cd ¥
   tar -xvf backup.tar
   ```

2. リストア対象のファイルがオンライン・バックアップ・プロセスを使用して作成したものである場合は、すべての `.bak` ファイルの名前を `.log` ファイルに変更します。例を示します。

```
for i in fastraid1/algorithmic1/*.bak; do mv $i ${i%%.bak}.log;
done
```

この例では、ksh シェルまたは bash シェルを使用し、すべてのファイルに同じロケーションを使用していることを前提とします。

3. プロジェクト・サーバを再起動します。

参照：

- *データのバックアップ*(4 ページ)
- *ログ・ストアのサイズ設定*(8 ページ)

Windows システムでのバックアップ・ファイルのリストア

Windows システムでバックアップ・ファイルからファイルをリストアするには、**WinZip** コマンドまたは **pkunzip** コマンドを使用します。

前提条件

Event Stream Processor を停止します。

手順

1. バックアップ・ファイルからリストアするファイルを抽出します。
   たとえば、backup.zip からファイルを抽出するには、次のコマンドを使用します。

```
c:
cd ¥
pkunzip backup.zip
```

2. Event Stream Processor を再起動します。

参照：

- *データのバックアップ*(4 ページ)
- *ログ・ストアのサイズ設定*(8 ページ)

ログ・ストアのサイズ設定

必要なログ・ストアのサイズを計算します。ログ・ストアは小さすぎても大きすぎてもパフォーマンスの問題を引き起こす可能性があるので、ログ・ストアのサイズを正しく設定することは重要です。

1. レコードの数とボリュームの両方の点から、ログ・ストアに収集するデータの最大量をバイト単位で見積もります。ソース・ストリームに入ってくるレコードの数とレコードのサイズの両方がわかっていれば、単に計算するだけです。

わからない場合は、プレイバック機能を使用して実際のデータを記録して再生することで、格納する必要があるデータの量を適切に把握できます。

ログ・ストアは、プロジェクトの終了時 (ログ・レベルが 3 以上の場合) と圧縮後 (ログ・レベルが 6 以上の場合) にサーバ・ログに "`liveSize`" を報告します。

注意： サーバ・ログの通知メッセージで "`liveSize`" が報告され、インデックス作成のオーバヘッドが既に含まれている場合は、手順 2 を省略します。

2. 基本的なインデックス作成のオーバヘッドを計算するには、レコードの数に 96 バイトを乗算します。その結果をボリュームに加算します。
3. **reservePct** パラメータの値を選択します。予約領域を含む、必要なストア・サイズ (バイト単位) は、次のように計算します。

   storeBytes = dataBytes * 100 / (100 - **reservePct**)

   この計算結果をメガバイト単位で切り上げます。
4. チェックポイントが設定されていないデータによって予約領域の容量を超過しないようにします。

   すべてのストリーム (ソース・ストリームを除く) の入力キューが満杯になったときに作成されるチェックポイントが設定されていないデータの最大量を見積もります。 配置された順番が早いキュー内のレコードはプロジェクトを通して処理されるので、それらによって生成されるレコードと合わせて数を数えます。 入力レコードごとにストリームによって生成される出力レコードの数を含めます。

   この例では、次のような順序の 4 つのストリームがあるログの、深さがデフォルトの 1024 に設定されたストリーム・キューを示します。

   ```
   source --> derived1 --> derived2 --> derived3
   ```

   a) 各ストリームによって生成されるレコードの数を次のように算出します。これらのレコードはそのキューの内容を消費します。
      - `derived1` の入力キュー内のレコード数は最大 1024 個になる可能性があります。キューが 1 つの入力レコードにつき 1 つの出力レコードを生成するとすれば、1024 個のレコードが生成されます。
      - `derived2` の入力キューのレコード数は最大 2048 個 (そのキューに既に収集されている 1024 個と、`derived1` からの追加の 1024 個) になる可能性があります。`derived2` がジョインで、入力レコードごとに平均 2 つの出力レコードを生成するとすれば、4096 個のレコード ([1024 + 1024] * 2) が生成されます。

- `derived3` のレコード数は最大 5120 個 (そのキューにある 1024 個と、`derived2` からの 4096 個) になる可能性があります。パススルー率を 1 とすると、`derived3` では 5120 個のレコードが生成されます。

プロジェクトのトポロジが線形でない場合は、すべての分岐を考慮します。データの親ストリームが異なると、パススルー率も異なる可能性があります。 すべての入力パスからのデータを加算します。 各ストリームの入力キューは 1 つのみなので、接続先の親ストリームの数にかかわらず、その深さは不変です。 ただし、各キュー内のレコードの組み合わせは異なる可能性があります。 最大量の出力を生成するレコードでキュー全体が構成されることを前提とします。 入力ストリームには、いったんロードされると通常の作業中は絶対に変更されない静的なデータが含まれることもあります。 これらの入力を数える必要はありません。この例では、`derived2` はジョイン・ストリームで、その 2 番目の入力として静的なデータがあります。

b) レコードの総数にそのストリームの平均レコード・サイズを乗算して、必要な領域を計算します。
たとえば、レコードの平均サイズが `derived1` では 100 バイト、`derived2` では 200 バイト、 `derived3` では 150 バイトの場合、計算式は次のようになります。

(1024 * 100) + (4096 * 200) + (5120 * 150) = 1,689,600

プロジェクト全体を通して、ソース・ストリームからログ・ストア内の少なくともすべてのストリームまで、レコード数をトレースします。 ログ・ストアにあるストリームから、サイズが設定されたデータのみを合計します。

c) レコード数に 96 バイトを乗算してインデックス作成のオーバヘッドを計算し、その結果をボリューム (バイト単位) に加算します。

(1024 + 4096 + 5120) * 96 = 983,040

1,689,600 + 983,040 = 2,672,640

この結果が予約領域の 1/4 以下であることを確認します。

uncheckpointedBytes < storeBytes * (**reservePct** / 4) / 100

この結果が予約領域の 1/4 より大きい場合は、予約率を上げて、ストア・サイズの計算を繰り返します。チェックポイントが設定されていないデータは、主にストアが小さいほど問題になります。クリーニング・サイクルによってデータが削除されたり圧縮されたりするため、チェックポイントが設定されていないデータのサイズ以外は、このオーバヘッドがストア・サイズの計算に著しく影響することはありません。

参照：
- *データのバックアップ*(4 ページ)
- *データのリストア*(7 ページ)

ログ・ストア

プロジェクトの XML ファイルにログ・ストアのサイズを指定します。

メモリ・ストアと異なり、ログ・ストアは自動的に拡張しません。ログ・ストアのサイズを正しく設定することが重要です。ストアのサイズが小さすぎると、クリーニング・サイクルの回数を増加する必要があるため、パフォーマンスが大幅に低下します。最悪の場合、ログ・ストアがオーバフローし、処理が停止する可能性もあります。ストアのサイズが大きすぎると、メモリとディスクのフットプリントが大きくなるため、この場合もパフォーマンスの問題が生じます。しかし、これらの問題はログ・ストアが小さすぎる場合ほど深刻ではありません。

***reservePct** パラメータ*
予約領域は、ストアの定期的なクリーニング時とストアの適正なサイズ変更を実行する場合に使用する中間領域として確保されます。

注意： 予約領域が小さすぎる場合にストアがデータで満杯になるまでプロジェクトを実行すると、サイズ変更が試みられ、その結果、ストアが反応しなくなることがあります。 つまり、予約領域はサイズ変更できません。予約領域からデータを取り出せるのは Sybase サポート・センタのみです。 予約領域は小さすぎるより大きすぎる方が安全です。ほとんどの状況ではデフォルトの 20% で十分です。ギガバイト・レベルのストアでは 10% まで下げた値を使用することもできます。ストアが小さく (30MB 以下)、特に複数のストリームが含まれている場合は、予約率の増加 (最大 40%) が必要になることもあります。40% でも足りないと思われる場合は、ストアのサイズを拡大します。

Event Stream Processor は必要な予約領域のサイズを自動的に見積もり、小さすぎると拡大します。通常、この操作の影響を受けるのは小さいストアのみです。

注意： 予約領域を拡大すると、データ用の残りの領域が少なくなります。新しいプロジェクトを開始する場合は、自動調整についてのサーバ・ログ・メッセージをモニタします。これらのメッセージが表示された場合は、ストアの拡大が必要である可能性があります。

ストアが稼働していれば、空き領域が予約領域より少なくなるまでさらにレコードがストアに書き込まれます。空き領域が予約領域より少なくなると、ソース・ストリームが一時的に停止します。ストリームはクワイス状態になり、チェックポイントとクリーニング・サイクルが実行されます。 ストリームはすぐにはクワイス状態になりません。 まず、入力キューに収集されたすべてのデータを処理します。クワイス状態の間に生成されたデータはすべてストアに追加されます。つまり、予約領域は、このデータを格納できるくらい大きいだけでなく、クリーニ

ング・サイクルを実行できる余裕も必要です。 このデータの量が予約領域を超えると、ストアはクリーニング・サイクルを実行できなくなるため、反応しなくなります。予約領域の自動計算では、チェックポイントが設定されていないデータは考慮されません。

ログ・ストア・サイズの警告
ストア内のデータの量が増加し、空き領域が 10% を下回る (予約領域を除く) と、Event Stream Processor はサーバ・ログに "`log store is nearing capacity`" という報告を開始します。データがストアから一度に大量に削除される場合は (たとえば、データが日中に収集され、1 週間より古いデータが 1 日の終わりに破棄される場合は)、古いデータがフラッシュされた後もこれらのメッセージが断続的に表示されることがあります。 削除されたデータがクリーニング・サイクルによってロールオーバされると、メッセージは表示されなくなります。

ログ・ストアが非常に小さい場合を除き、これらの警告はストアが領域不足になる前に表示されます。これらが表示された場合は、適宜に Event Stream Processor を停止して、ストアを拡大します。 拡大しないと、プロジェクトの空き領域が予約領域より少なくなったときに Event Stream Processor がアボートします。

ストアのサイズを正しく設定しておかないと、予約領域が満杯になる、つまり「反応しなくなる」可能性があり、予約領域をサイズ変更することも、予約領域にコンテンツを保存することもできなくなります。ストア・ファイルを削除して、ストアを空にした状態で Event Stream Processor を再起動します。ストア・ファイルを削除する前にバックアップを作成しておけば、Sybase サポート・センタがコンテンツを抽出できる可能性があります。プロジェクトでストアのサイズを変更すると、再開時にストアはサイズ変更されます。 ストアのサイズを小さくすることはできません。 ストアをサイズ変更してからプロジェクトを再開した場合は、新たに追加された空き領域がクリーニング・サイクルによって取り込まれるまで、空き領域が予約領域より少ないことを示すサーバ・ログ・メッセージが生成される可能性があります。

ストリームとログ・ストア
ストリーム (フレックス・ストリームなど) は、そのロジックでローカル変数またはグローバル変数のコンテキストを使用している場合、一般にメモリ・ストアを使用します。そうしないと、Event Stream Processor を再起動したときにストリームのストアは保持されますが、変数の値はリセットされます。 これらの変数をユニーク・キーの作成に使用すると、キーはユニークではなくなります。

一般的に、ソース・ストリームのみをログ・ストアに格納するか、ソース・ストリームとそのソース・ストリームが (直接的または間接的な) 派生元であるすべてのストリームを同じログ・ストアに格納することをおすすめします。 複数のストアが処理のシーケンスに混在すると、突然の停止と再起動によって、重複したキーを持つ不正なレコードに関するメッセージが再起動時に表示されることがあ

ります。 ローカル変数またはグローバル変数があると、再起動によってさらに大きな不整合が生じることもあります。

変更頻度が大幅に異なるストリームには別のログ・ストアを使用します。 大量でもほぼ静的なストリームと少量でも非常に変更頻度の高いストリームが 1 つのログ・ストアにあると、クリーニング・サイクルごとに静的なストリームの大量のデータを処理しなければなりません。 ストリームを分けておけばクリーニング・サイクルが最適化されます。 これは、ソース・ストリームとそのソース・ストリームを派生元とするすべてのストリームを同じログ・ストアに格納する説明とは矛盾しますが、ソース・ストリームのみをログ・ストアに格納し、派生ストリームはメモリ・ストアに格納することをおすすめします。

***ckcount** パラメータ*

ckcount (チェックポイント設定カウント) パラメータは、チェックポイントが設定されていないデータのサイズに影響します。このカウントは、中間インデックス・データの作成前に更新される可能性があるレコードの数を示します。 大きい値に設定すると、オーバヘッドは数多くのレコードに分散されてほぼ均一化され、レコードあたり平均 96 バイトになります。 小さい値に設定すると、オーバヘッドは増加します。ゼロに設定すると、トランザクションが終わるたびにインデックス・データが作成されます。シングルトランザクション・レコードのオーバヘッドは次のようになります。

$96 + 32 * \text{ceiling}\ (\log_2(\text{number\_of\_records\_in\_the\_stream}))$

ストリームが小さい (たとえば、レコード数が 1000 個未満) 場合、各レコードのオーバヘッドは次のようになります。

$96 + 32 * \text{ceiling}\ (\log_2(1000)) = 96 + 32 * 10 = 416$

多くの場合、レコード自体はオーバヘッドの 416 バイトより小さいサイズです。影響は対数的なので、大きいストリームはあまり影響を受けません。 100 万個のレコードを含むストリームの対数は 20 なので、レコードあたりのオーバヘッドは 736 バイトです。 オーバヘッドが増加すると、余分なデータが作成されてストア・クリーニングの回数が増えることによって、パフォーマンスに影響します。

***sweepamount** パラメータ*

sweepamount パラメータでは、クリーニングのたびに「スイープ」するログ・ファイルの容量を指定します。 **fullsize** パラメータの 5 ～ 20% の間の値を指定します。スイープ・サイズの下限はストレージ配列の書き込みキャッシュの半分にすることをおすすめします。 通常、これは 512 ～ 1024 メガバイトのスイープ・サイズを示します。 スイープ・サイズが小さいほど遅延時間のスパイクは最小限に抑えられますが、平均遅延時間は増加します。 値が大きいほど平均遅延時間は短くなりますが、領域の再利用時のスパイクは増加します。

sweepamount パラメータの値が小さすぎると、システムはクリーニングを過剰に実行します。このため、ログ・ストアはクリーニング時に十分な領域を解放できないこともあります。

スイープのサイズは、クリーニング・サイクルの開始時に予約領域に残っている空き領域の容量によっても制限されます。 予約領域がスイープの容量より小さく設定され、未使用のデータがスイープであまり見つからないと、再配置されたライブ・データで予約領域が満杯になった場合にスイープは停止します (スイープによって新たにクリーニングされた領域は次回のサイクルで新しい予約領域になります)。 優先される要因が他にない限り、スイープのサイズと予約領域のサイズは近い値にしておくことをおすすめします。**reservePct** はパーセント単位で指定するのに対し、**sweepamount** はメガバイト単位で指定します。

ログ・ストアのサイズとファイル・ロケーション
すべてのログ・ストア・ファイルの合計サイズがマシンで使用可能な RAM のサイズを超えないようにします。RAM のサイズを超えると、マシンでのデータの処理時間が長くなるため、すべてのモニタ・ツールで、各ストリームの CPU 利用率が低いことが示され、**vmstat** などの標準的な UNIX コマンドでは、システム・ページングのためにディスク使用率が高いことが示されます。

ログ・ストアを使用してデータをローカルに格納する場合は、高速ストレージ・デバイス (たとえば、RAID アレイまたは SAN で、できれば動的 RAM キャッシュが大きいもの) を使用することをおすすめします。スループットがやや低い場合は、ログ・ストアのバッキング・ファイルを単一ディスク・ドライブ (SAS、SCSI、IDE、または SATA) に置きます。

参照：
- *データのバックアップ*(4 ページ)
- *データのリストア*(7 ページ)

メモリ使用量

パフォーマンスを最大限に発揮するには、プロジェクト・サーバでレコードが重複しないようにします。さらにパフォーマンスを向上させるには、各シェル・プロセスが使用できるメモリの容量を制限します。

プロジェクト・サーバに、マシン上の RAM 使用量を直接設定したり制御したりする設定はありません。ただし、プロジェクト・サーバはシステム内のレコード数を数え、各種のストリームでレコードが重複しないようにしています。メモリ使用量はプロジェクトに格納されているレコードの数に正比例します。

プロジェクト設定ファイル (`.ccr`) に Java 仮想マシン (VM) の Java ヒープ・サイズを設定します。

プロジェクト・サーバのモニタ

コマンド・ライン・インタフェースを使用して、プロジェクト・サーバの実行インスタンスのパフォーマンスをモニタします。

esp_monitor ツールはプロジェクト・サーバの実行インスタンスからパフォーマンス・データを読み取り、標準出力でそのデータを表示します。 データをモニタできるのは、プロジェクト設定 (CCR) ファイルまたはスタジオを使用して時間粒度オプションが設定されている場合のみです。

この時間粒度オプションでは、実行中の Event Stream Processor からパフォーマンス・レコード (ストリームあたり 1 つとゲートウェイ接続あたり 1 つ) のセットを取得する頻度を秒単位で指定します。 デフォルトでは、すべてのプロジェクトの時間粒度が無効です。ユーザは、モニタが不要な場合はこの時間粒度プロジェクト・オプションを無効にしておくことで、パフォーマンスを向上させることができます。このプロジェクト・オプションは `.ccr` ファイルとスタジオで設定します。スタジオでのプロジェクトの設定方法の詳細については、『スタジオ・ユーザーズ・ガイド』を参照してください。

注意： **esp_clients_monitor** ストリームには接続クライアントに関する基本情報がありますが、パフォーマンス関連のフィールドにデータが挿入されるのはモニタ・オプションを設定した場合のみです。

RPC ポート 31415 を使用しているホスト "myhost.sybase.com" にマネージャ・ノードがあるクラスタ内で実行されているプロジェクトをモニタするには、次のコマンドを使用します。

```
esp_monitor -p myhost.sybase.com:31415/workspace-name/project-name
```

次に、簡単な例のみを示します。**esp_monitor** ツールの詳細については、『ユーティリティ・ガイド』を参照してください。

参照：

- *プロジェクト配備オプション* (21 ページ)
- *プロジェクト設定ファイルのサンプル* (38 ページ)
- *クラスタの設定* (29 ページ)

第 2 章 クラスタの管理

Event Stream Processor ではクラスタの拡張がサポートされ、必要に応じてクラスタにノードを追加できます。クラスタを使用することでユーザは複数のプロジェクトを同時に実行できます。また、可用性とフェールオーバが実現され、中央集中型のセキュリティとサポートを適用してクラスタ接続を管理できます。

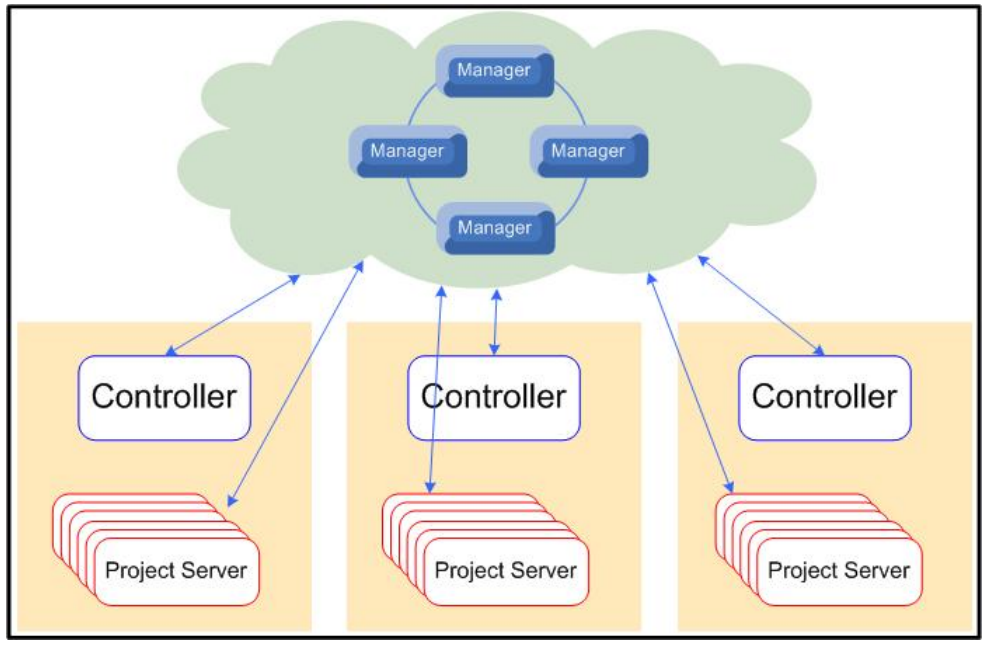
クラスタ・アーキテクチャ

Event Stream Processor のクラスタは、起動した後の管理者による操作を簡素化し、その必要性を最小限に抑えるように設計されています。

クラスタはノードのグループで構成されます。ノードとは、ホスト上で実行されるプロセスです。クラスタ内のノードは 1 つでも複数でもかまいません。シングルノード・クラスタは、マルチノード・クラスタを作成したり調整したりする場合の便利な起点となります。

クラスタは、マネージャ専用ノード、コントローラ専用ノード、またはマネージャでありコントローラでもあるノードで構成されます。クラスタはオンデマンドでプロジェクト・サーバを起動し、プロジェクト・ライフ・サイクルを管理します。 次の図で、コンテナはクラスタ内で実行されているプロジェクトを表します。

図 1 : クラスタ・アーキテクチャ



シングルノード・クラスタとは、1 つのマネージャ・ノードを含むクラスタのことです (このノードはマネージャとコントローラの両方の役目を果たします)。開発環境とテスト環境では、シングルノード・クラスタで十分です。シングルノード・クラスタに複数のプロジェクトを配備してプロジェクトの状況をモニタできます。また、配備されたプロジェクトにフェールオーバが設定されていれば、障害が発生したプロジェクトを再開できます。ただし、Event Stream Processor 環境を開発したり調整したりする場合は、クラスタにかかる負担が増加します。そのため、クラスタを拡大してノードを追加できます。必要に応じて、クラスタを追加することもできます。

クラスタに複数のマネージャ・ノードが含まれている場合、そのクラスタのことをマルチノード・クラスタと呼びます。マルチノード・クラスタでは、すべてのマネージャ・ノードがプライマリと見なされるため、クラスタ内に単一障害点はありません。ただし、複数のマネージャに対し設定されているコントローラが 1 つのみであると、そのコントローラが単一障害点になる可能性があります。

プロジェクトは、クラスタに配備されると、ノードにあるマネージャの 1 つを使用してハートビートを維持します。マネージャ・ノードはプロジェクトでハートビートが連続して 3 つ欠落していることを検出すると、プロジェクト障害と見なし、**STOP** コマンドを発行します。配備されたプロジェクトにフェールオーバが設定されていれば、プロジェクトは再開されます。CPU 使用率が 100% に達すると、プロジェクト・サーバはクラスタ・マネージャにハートビートを送信できなくな

り、プロジェクトを停止することがあります。マルチノード・クラスタでは、配備先のマネージャ以外の別のマネージャがプロジェクトのモニタを引き継ぐことができます。

マネージャ・ノードは共有キャッシュを介して他のマネージャとペアになります。あるマネージャ・ノードでプロジェクトの開始後に障害が発生すると、共有キャッシュを使用している他のマネージャ・ノードが、その障害が発生したマネージャ・ノードでこれまで監視していたプロジェクトの管理を引き継ぐことができます。

ファイルとディレクトリのインフラストラクチャ

クラスタ設定、プロジェクト配備、永続性は、クラスタ設定ファイルとプロジェクト設定ファイルを使用して管理します。

クラスタ管理では、次のファイルとディレクトリを使用します。

- `<nodename>.xml` - クラスタ設定ファイルは、インストール時に設定されたクラスタに対しては自動的に設定されます。インストール後にクラスタを作成した場合は手動で設定できます。クラスタ設定ファイルには、マネージャまたはコントローラのどちらかとしてのノードの定義、セキュリティの設定、ポート通信の有効化、キャッシュ、高可用性プロジェクト、永続性に必要なすべてのパラメータがあります。
- `<projectname>.ccr` - プロジェクト設定ファイルには、高可用性モード (アクティブ／アクティブ) でのプロジェクトの配備、コントローラとインスタンスの各結び付きの定義、フェールオーバの間隔についてのパラメータがあります。プロジェクトをアクティブ／アクティブ・モードで配備できるのは、クラスタ設定ファイルで、プロジェクトが実行されているノードに対し高可用性が有効に設定されている場合のみです。
- `<base-directory>` - クラスタ設定ファイルでは、ワークスペースとプロジェクト・インスタンスの格納先のベース・ディレクトリを設定できます。このパラメータはオプションです。
- `<persistence-directory>`- クラスタ設定ファイルでは、永続性を有効に設定し、クラスタのノードを開始したり停止したりした後もアプリケーションとワークスペースの設定を維持するディレクトリを定義できます。 クラスタ内の 1 つのマネージャを永続性対応にする場合は、そのクラスタ内のすべてのマネージャを永続性対応にします。また、すべてのマネージャが同じディレクトリを共有する必要があります。ディレクトリ名は変数です。
- `<security>`- クラスタ設定ファイルは Event Stream Processor のセキュリティ・ディレクトリを参照します。このディレクトリには、RSA 保護サーバ・インスタンス用の Java キーストア・ファイル、アクセス制御用の LDAP ポリシー・ファイル、他の認証モードが指定されていない場合に ESP サーバにオープン・セキュリティを提供する `csi_no_security.xml` ファイルなど、Event Stream Processor のセキュリティ・ファイルが格納されています。クラスタ内のすべて

のコンポーネントに、クラスタ設定ファイルに定義されているセキュリティ・ルールが適用されます。

クラスタのガイドライン

複数ノード設定のベスト・プラクティスについて説明します。

クラスタ内に複数のノードを設定する場合は、次のようにします。

- クラスタ内のすべてのノードで、キャッシュ名とキャッシュにアクセスするためのパスワードを同一にする必要があります。
- すべてのノードに対し、クラスタ設定ファイルでユニークな名前を指定します。名前の大文字と小文字は区別されません。
- 各クラスタで、ホストごとに定義するマネージャは 1 つのみとします。
- プロジェクトに共通のベース・ディレクトリを設定します。これで、すべてのプロジェクト・ログ・ストア・ファイルを共通の場所に保存できます。
- クラスタ内のすべてのマネージャ間に共通の永続性ディレクトリを設定します。クラスタ内の 1 つのマネージャ・ノードを永続性対応にする場合は、すべてのマネージャを永続性対応にします。
- 共通のセキュリティ・ファイルを参照します。すべてのノードのセキュリティ設定を同一にします。認証モードにかかわらず、すべてのノードにキーストアが必要です。キーストア・ファイルは、クラスタ内のすべてのノード間で共有します。

注意： 新しいユーザ・キーを配備すると、deploy コマンドの送信先のノードがキーストアを更新し、その他のノードはそのファイルを再ロードします。配備キーが正常に動作しているかどうかをテストするには、新しいキーでクラスタにログインしますが、異なるノードを使用します。

複数のプロジェクトのサポート

クラスタでは、ユーザはローカル・クラスタとリモート・クラスタを使用して複数のプロジェクトを同時に実行できます。

ローカル・クラスタは、ユーザがプロジェクトを開始したときに、スタジオによって自動的に作成されます。ローカル・クラスタは単一のユーザのマシンに存在し、他のユーザはアクセスできません。ローカル・クラスタには別々のライセンスが必要です。ローカル・クラスタではユーザ独自のプロジェクトを作成してテストできますが、運用環境をサポートするようには設計されていません。

リモート・クラスタは、管理者がネットワーク・サーバ上に作成できます。また、管理者はこれらのリモート・クラスタへのアクセス権を持つユーザを制御できます。**esp_server** で起動されるクラスタはすべて、スタジオと同じホスト上で実行されているものを含め、スタジオの観点でも、リモートと見なされます。クラスタは、Event Stream Processor のインストール・プロセス時でも、インストールの完了後でも作成できます。

ユーザは、複数のプロジェクトを 1 つのクラスタ・ノードで実行することも、別々のホスト・マシン上の複数のノードに分散して実行することもできます。プロジェクトをクラスタで実行すれば、プロジェクト障害が発生した場合にリカバリできます。

高可用性

Event Stream Processor では、サーバ・クラスタによって障害のリカバリ性とデータの冗長性が向上します。Event Stream Processor のアクティブ／アクティブ・モードと呼ばれるプロジェクト・レベルでは、高可用性が向上します。

シングルノード・クラスタでは、プロジェクト・レベルで障害のリカバリが実行されます。つまり、プロジェクトが実行を停止したことが検出され、プロジェクトは自動的に再開されます。ただし、シングルノード・クラスタはサーバ障害を防ぐことはできません。

複数ノード・クラスタはサーバ障害を防ぐことができます。さまざまなマシン上にクラスタ・マネージャを作成し、それらを 1 つのクラスタにまとめます。複数ノード・クラスタを配備する場合、ユーザはノードごとにプロジェクトの結び付きを定義できます。

また、ユーザはアクティブ／アクティブ・モードでもプロジェクトを配備できます。この場合は、クラスタ内で同じプロジェクトの 2 つのインスタンスが実行されます (別々のマシンで実行することをおすすめします)。プロジェクトの一方のバージョンがプライマリ・インスタンスとして指定され、もう一方がセカンダリ・インスタンスとして指定されます。クラスタの外部 (アダプタ、クライアント、スタジオ) からの接続はすべてプライマリ・プロジェクト・サーバに送られます。プライマリ・インスタンスで障害が発生すると、接続はすべて自動的にセカンダリ・インスタンスに送られます。

データはプライマリ・インスタンスとセカンダリ・インスタンス間で同期され続けます。まず、プライマリ・インスタンスが各メッセージを受信します。冗長性を維持するため、プライマリ・インスタンスが処理を開始する前にセカンダリ・インスタンスもメッセージの受信を確認します。

参照：

- *プロジェクト配備オプション* (21 ページ)

プロジェクト配備オプション

プロジェクト配備オプションでは、クラスタへのプロジェクトの配備方法と実行時のプロジェクトの動作方法を指定します。プロジェクト・オプション、アクティブ／アクティブ・インスタンス、フェールオーバ間隔、プロジェクト配備タ

イプ・オプションを含め、これらのパラメータは、CCR ファイルに手動で設定するか、ESP スタジオで設定します。

プロジェクト・オプション
プロジェクト・オプションは、プロジェクト設定エディタの [Advanced] タブで設定できます。プロジェクト・オプションはプロジェクトのランタイム・パラメータとして使用され、使用可能なオプション名の事前定義リストが含まれます。これには、ほとんどのコマンド・ライン・エントリが反映されています。各プロジェクト・オプションには、ユーザが入力できる値フィールドもあります。オプションを実装できるのは 1 回のみです。実装した後は、ドロップダウン・リストで選択できなくなります。

次の表に、ESP スタジオの [Project Configuration] ビューを使用して設定できる使用可能なすべてのプロジェクト・オプションをまとめます。

プロジェクト・オプション	説明
on-error-discard-record	true の場合、計算エラーが発生すると、計算対象のレコードは破棄されます。false に設定した場合、計算対象外のすべてのカラムに NULL が埋め込まれ、レコード処理は続行します。デフォルト値は true です。 **注意：** キー・カラムの計算エラーが発生した場合は、このオプションの設定にかかわらずレコードは破棄されます。
on-error-log	true に設定した場合、計算エラーが発生するとエラー・メッセージに記録されます。デフォルト値は true です。
java-classpath	Java クラス・パスを設定します。値は classpath ファイルのファイル・パスです。
java-max-heap	プロジェクトの最大 Java ヒープを設定します。デフォルト値は 256 メガバイトです。
utf8	サーバ上の UTF-8 機能を有効にします (デフォルトでは、この機能はオフです)。デフォルト値は false です。true に設定すると有効になります。
precision	プロジェクトで数字を表す場合の小数点以下の桁数を設定します。デフォルト値は 6 です。
command-port	コマンド・ポート番号を設定します。一般に、この詳細オプションは設定しないでください。ポートが 0、または 1 ～ 65535 の範囲外の場合は、プログラムによって任意のポートが選択されます。特定のポートを定義するには、1 ～ 65535 の値を設定します。

プロジェクト・オプション	説明
sql-port	SQL ポート番号を設定します。一般に、この詳細オプションは設定しないでください。ポートが 0、または 1 ～ 65535 の範囲外の場合は、プログラムによって任意のポートが選択されます。特定のポートを定義するには、1 ～ 65535 の値を設定します。
gateway-port	ゲートウェイ・ポート番号を設定します。一般に、この詳細オプションは設定しないでください。ポートが 0、または 1 ～ 65535 の範囲外の場合は、プログラムによって任意のポートが選択されます。特定のポートを定義するには、1 ～ 65535 の値を設定します。
time-granularity	プロジェクト内の時間粒度を定義します。このオプションでは、実行中の Event Stream Processor からパフォーマンス・レコード (ストリームあたり 1 つとゲートウェア接続あたり 1 つ) のセットを取得する頻度を秒単位で指定します。デフォルトでは、時間粒度は 5 に設定されます。このオプションを 0 に設定すると、モニタは無効になります。この場合は、パフォーマンスも最適化されます。
debug-level	プロジェクトをデバッグする場合のロギング・レベルを 0 ～ 7 の範囲で設定します。各番号の意味は、次のとおりです。 • 0：LOG_EMERG - システムが使用不可能である • 1：LOG_ALERT - アクションを直ちに実行しなければならない • 2：LOG_CRIT - 重大な状態 • 3：LOG_ERR - エラー状態 • 4：LOG_WARNING - 警告状態 • 5：LOG_NORMAL - 正常だが重要な状態である • 6：LOG_INFO - 通知 • 7：LOG_DEBUG - デバッグ・レベル・メッセージ
memory	プロジェクトのメモリ使用の上限を設定します。デフォルトは 0 で、無制限を意味します。
optimize	冗長なストア更新を実行しません。デフォルト値は false で、true に設定すると有効になります。
ignore-config-topology	これを有効にすると、プロジェクト間のトポロジが無視されます。デフォルトは false で、true に設定すると有効になります。

プロジェクト・オプション	説明
time-interval	ウィンドウでのローの最大表示時間を指定する、定数の間隔式を設定します。デフォルトでは、秒単位で 0 に設定されています。これは、タイマなしを意味します。

アクティブ／アクティブの配備
アクティブ／アクティブの配備は、CCR ファイルにプロジェクトを `ha-project` として定義した場合のみ使用できます。アクティブ／アクティブの配備とは、プロジェクトの 2 つのインスタンスをクラスタ内で同時に実行することです。プロジェクトの 2 つのインスタンスは 2 つの異なるホストでクラスタ・マネージャによって開始されます。

プロジェクトの一方のインスタンスがプライマリ・インスタンスとして選択されます。インスタンスの一方が既にアクティブな場合、そのインスタンスがプライマリ・インスタンスです。障害が発生したインスタンスが再開すると、そのインスタンスはセカンダリ・ポジションとなり、現行のインスタンスで障害が発生するか現行のインスタンスが停止するまで、そのポジションを維持します。

セカンダリ・プロジェクト・サーバの起動時にプライマリ・プロジェクト・サーバが見つからないと、セカンダリ・プロジェクト・サーバはプライマリ・サーバへの接続を 30 秒間再試行します。既存のプライマリ・サーバに正常に接続できなければ、プライマリ・サーバの役割を引き継ぎます。

ユーザがアクティブ／アクティブの設定を追加するには、高可用性タイプを定義し、さらに結び付きパラメータを追加します。アクティブ／アクティブの設定は、CCR Project Deployment Editor を使用して定義できます。

次に、アクティブ／アクティブの配備に合わせて設定された CCR ファイルの例を示します。

```
<Deployment>
  <Project ha="true">
   <Options>
    <Option name="debug-level">1</Option>
   </Options>
   <Instances>
    <Instance name="primary">
     <Affinities>
      <!-- By default no need to put affinity.  -->
     <Affinity type="controller" charge="positive" strength="strong"
value="node1"/>
      <Affinity type="instance" charge="negative" strength="strong"
value="secondary"/>
     </Affinities>
    </Instance>
    <Instance name="secondary">
```

```
     <Affinities>
      <!-- By default no need to put affinity.  -->
      <Affinity type="controller" charge="positive" strength="weak"
value="node2"/>
      <Affinity type="instance" charge="negative" strength="strong"
value="primary"/>
     </Affinities>
     <Failover enable="true">
      <FailureInterval>120<FailureInterval> // numbers in sec
      <FailuresPerInterval>4<FailuresPerInterval> // counter
     </Failover>
    </Instance>
   </Instances>
  </Project>
 </Deployment>
```

インスタンス
使用可能なインスタンスの数は、ユーザが選択した配備タイプ ([HA] (高可用性) または [Non-HA]) によって異なります。プロジェクトが HA (アクティブ／アクティブ) モードで設定されている場合は、プライマリとセカンダリの 2 つのインスタンスが作成されます。フェールオーバ間隔、間隔あたりの障害数など、インスタンスごとに結び付きとコールド・フェールオーバに関するオプションを設定できます。

結び付き
Affinities (結び付き) では、クラスタ内でプロジェクトを実行するかどうかを制限します。結び付きには次の 2 つのタイプがあります。

- コントローラ - アクティブ／アクティブと非アクティブ／アクティブの設定に使用します。各コントローラには複数の結び付きを割り当てることができますが、strong positive として設定できるコントローラの結び付きは 1 つのみです。
- インスタンス - アクティブ／アクティブの設定にのみ使用します。インスタンスの場合は、それぞれ別々のプロジェクト・サーバに適用できる 2 つの結び付きが作成されます。

結び付きごとに定義するパラメータは、次のとおりです。

フィールド	説明
name	結び付きのオブジェクトの名前、つまり結び付きの設定対象であるコントローラ名またはインスタンス名を入力します。インスタンスの結び付きの場合、1 つのインスタンスの結び付きは 2 つ目のインスタンスを参照します。

フィールド	説明
strength	[strong] または [weak]。strong の場合、プロジェクトは特定のコントローラで実行する必要があります。その他のコントローラは使用できません。weak の場合、プロジェクトは定義されたコントローラで優先して開始されます。ただし、そのコントローラを使用できない場合は、別の使用可能なコントローラで開始できます。
charge	[positive] または [negative]。positive の場合、プロジェクトはコントローラで実行されます。negative の場合、プロジェクトはコントローラで実行されません。

フェールオーバ
プロジェクトは、適切に実行されなかったり、適切に実行が停止されなかったりすると、障害が発生します。障害が発生したプロジェクトまたはサーバが別のサーバに切り替わって処理を続行した場合、フェールオーバが発生します。フェールオーバによってプロジェクトが再開されることもあります (定義されている場合)。再開は、障害の間隔と間隔あたりの再開数の定義に基づいて制限できます。フェールオーバ・オプションにアクセスするにはインスタンス設定を使用します。次のようなオプションがあります。

フィールド	説明
Failover	[enabled] または [disabled]。[disabled] の場合、プロジェクトのフェールオーバによる再開は実行できません。[enabled] の場合、[Failure interval] と [Failures per interval] の各フィールドにアクセスでき、再開を実行できます。
Failures per interval	プロジェクトが所定の間隔内で再開を試行できる回数を指定します。この数は、プロジェクトを手動で開始したり、障害の発生時点からの期間が間隔より長いために障害がリストから消去されたりした場合に、ゼロにリセットできます。
Failure interval	(オプション) ここには、間隔の時間を秒単位で指定します。ブランクのままにすると、間隔の時間は無制限に設定されます。

参照：
- *高可用性* (21 ページ)

クラスタの永続性とキャッシュ

クラスタの永続性とキャッシュはクラスタ設定ファイルに設定します。

クラスタ内のノードを開始したり停止したりした後もアプリケーションとワークスペースの設定を維持するには、クラスタの永続性を有効に設定します。 永続性を有効にした場合、クラスタの設定を維持するディレクトリを変更できます。クラスタ内のすべてのノードが同じ永続性ディレクトリを参照します。

クラスタ・キャッシュはインメモリの分散キャッシュで、クラスタの状態と設定を内部で共有する目的で使用されます。マネージャ・ノードはキャッシュのメンバであるのに対し、コントローラ・ノードはキャッシュのクライアントです。キャッシュのプロパティはすべてのノードで設定する必要がありますが、コントローラ専用ノードのポート情報を指定する必要はありません。

中央集中型セキュリティ

Event Stream Processor のセキュリティ・オプションはクラスタを介してローカルで設定されます。

認証モードと SSL (Secure Sockets Layer) 接続はクラスタ設定ファイルに設定されます。Event Stream Processor では、Kerberos、RSA、LDAP のセキュリティ・プロバイダがサポートされています。クラスタ内で実行されているすべてのプロジェクトに、そのクラスタに対して定義されたセキュリティ・ルールが適用されます。

参照：

- *第 3 章、「Event Stream Processor のセキュリティ」* (41 ページ)

クラスタの起動

クラスタは、コマンド・ラインから、またはスタジオを使用して起動します。

前提条件

ESP_HOME 環境変数を設定します。

手順

クラスタを起動するときは、マネージャ・ノードを起動してからコントローラ・ノードを起動します。クラスタが実行しているときに、ユーザはノードにプロジェクトを配備できます。

1. コマンド・ラインから、次のように実行します。

   ```
   cd %ESP_HOME%¥cluster¥nodes¥node1
   %ESP_HOME%¥bin¥esp_server.exe --cluster-node node1.xml
   ```

   Windows システムの場合は、上記のようになります。

   ```
   cd $ESP_HOME/cluster/nodes/node1
   $ESP_HOME/bin/esp_server --cluster-node node1.xm
   ```

   Linux システムと Solaris システムの場合は、上記のようになります。

   注意： ノードの起動元のディレクトリがノードの作業ディレクトリになります。ノードはこの作業ディレクトリで `cluster.log.properties` ファイルを検索します。

2. ワークスペースとプロジェクトを取得して開始します。

   ```
   esp_cluster_admin --uri=esps://<host>:<port> --
   username=<user> --password=<pass>
   ```

   クラスタの URI とクレデンシャルを指定してコマンドを実行し、クラスタ管理コマンドの処理を開始します。

   注意： URI プロトコル `esps` は、クラスタが SSL 対応であることを示します。SSL 対応以外のクラスタの URI はプロトコル `esp` を使用します。

参照：

- *クラスタの設定* (29 ページ)

ESP スタジオからサーバへの接続

スタジオには、サーバに接続したりサーバを起動したりするためのグラフィカル・インタフェースがあります。これはクラスタ・マネージャとも呼ばれます。

デフォルトでは、インストール後に、ローカル・クラスタ・マネージャ接続が [Run-Test] パースペクティブに既に定義されています。このローカル・クラスタはテストに使用できます。また、リモート・クラスタ・マネージャへの新しいサーバ接続も定義できます。デフォルトのサーバ接続とユーザ定義のサーバ接続のどちらも [Run-Test] パースペクティブの [Server View] ウィンドウに表示されます。

サーバ接続がまだ定義されていない場合は、[Run-Test] パースペクティブの [Server View] で [New Server URL] を選択して、新しい接続を作成します。

ESP スタジオのクラスタ・セットアップの詳細は、`$ESP_HOME/studio/esp-studio/clustercfg/localnode.xml` にあります。このファイルは変更しないでください。ESP スタジオからリモート・クラスタへ接続できなくなる可能性があります。

1. [Run-Test] パースペクティブがまだ表示されていない場合は、ウィンドウ上部の [Run-Test] タブをクリックするか、[ウィンドウ] - [Open Perspective] - [Run-Test] を選択します。
2. [Server View] ウィンドウが表示されていない場合は、[Run-Test] パースペクティブで [ウィンドウ] - [Show View] - [Server View] と選択します。
3. 次のどちらかを行います。
   - [New Server URL] を選択し、接続先のクラスタのホスト名とポートを指定して、新しいリモート・クラスタ接続を作成します。
   - 接続先のサーバを右クリックし、[Connect Server] を選択します。

   接続が成功すると、サーバ・フォルダにサーバ・ストリームが表示されます。
4. リストされているすべてのサーバに接続するには、[Server View] ウィンドウの右上隅の [Reconnect All] アイコンを選択します。

   [Filter Metadata Streams] の選択を解除すると、すべてのメタデータ・ストリームが [Server View] に表示されます。

クラスタの設定

クラスタを設定すると、処理作業が複数のサーバ間で分割されることによってパフォーマンスが向上します。

インストール時にクラスタを設定しなかった場合や、新しいクラスタを作成して設定する場合は、次の手順に従います。

クラスタの設定では、ノードのすべてのコンポーネントを設定します。クラスタ内の各ノードには、コントローラ、マネージャ、RPC、キャッシュの 4 つの基本的な設定セクションがあります。

```
[...]
- <Controller enabled="true">
  </Controller>
  <Manager enabled="true" />
- <Rpc>
  <Port>19011</Port>
  </Rpc>
- <Cache>
        <Host>thehostname</Host>
  <Port>19001</Port>
  <Name>test-name-1</Name>
  <Password>test-password-1</Password>
        <Managers enabled="true">
            <Manager>localhost:19001</Manager>
            <Manager>localhost:19002</Manager>
         <Manager>localhost:19003</Manager>
        </Managers>
        <Persistence enabled="true"
            <Directory>${ESP_STORAGE}</Directory>
```

```
        </Persistence>
  </Cache>
[...]
  </Node>
```

ノードをコントローラまたはマネージャのどちらかとして有効にするか、その両方として有効にするかに応じて、設定は異なります。

注意： Event Stream Processor の UNIX ベースのインストールには、この作業に使用するサンプル・ファイルがあります。

クラスタを設定する場合、ホスト名の値にはデフォルトで "localhost" が使用されます。他のマシンのクラスタ・クライアントが接続できるようにするには、ホスト名をそのクラスタ・ノードが実行されているマシンのホスト名に変更します。

1. UNIX インストールの場合は `${ESP_HOME}/cluster/nodes/<node-name>/<node-name>.xml` から、Windows インストールの場合は `%{ESP_HOME}%¥cluster¥nodes¥<node-name>¥<node-name>.xml` から、設定ファイルを開きます。
2. クラスタ内のノードにユニークな名前を割り当てます。
   ```
   <Name>node1</Name>
   ```
   注意： ノード名で大文字と小文字は区別されません。
3. (オプション) マクロの名前とタイプの各要素を設定します。

   マクロとは、繰り返し実行する設定を中央で管理したり、環境からプロパティを取得したりするための設定ファイルのショートカットです。

   使用できるマクロ・タイプ項目は、次のとおりです。
   - envar - マクロの値で定義されている環境変数から値を抽出します。
   - sysproperty - マクロの値で定義されている Java システム・プロパティから値を抽出します。
   - value - 指定された値が使用されます。
   - prompt - クラスタの起動時に、ユーザに値の入力を求めるプロンプトが表示されます。

   ```
   <Macros>
   <Macro name="ESP_HOME" type="envar">ESP_HOME</Macro>
   </Macros>
   ```
4. (オプション) システム・プロパティを設定します。

   この手順には、マクロ展開によってマクロの値をリテラル値に置換する操作を含めることができます。
5. コントローラを次のように有効にします。
   ```
   <Controller enabled="true">
   ```
6. コントローラを有効にしたノードで、コントローラ・セクション内に既存のアプリケーションを設定します。アプリケーションごとに名前を指定します。ま

た、アプリケーションを true に設定して有効にし、アプリケーション・クラスを定義します。

次の例はプロジェクト・アプリケーションを示しています。このアプリケーションが使用するベース・ディレクトリ、ホスト名、サービス設定ファイル、セキュリティ・ディレクトリが定義されています。これらのディレクトリは変更できますが、クラスタ内のすべてのプロジェクトで同一とします。

```
- <ApplicationTypes>
- <ApplicationType name="project" enabled="true">
  <Class>com.sybase.esp.cluster.plugins.apptypes.Project</Class>
  <StandardStreamLog enabled="true" />
- <Properties>
  <Property name="esp-home">${ESP_HOME}</Property>
  <Property name="hostname">${ESP_HOSTNAME}</Property>
  <Property name="ld-preload" />
  <Property name="services-file">${ESP_HOME}/bin/service.xml</
Property>
  <Property name="base-directory">${ESP_HOME}/cluster/projects/
test-name-1</Property>
  <Property name="ssl-key-file">${ESP_HOME}/security</Property>
  </Properties>
  </ApplicationType>
```

Event Stream Processor に付属の JRE (Java Runtime Environment) とは異なる別の JRE を使用している場合は、**ld-preload** パラメータを変更します。別の JRE を使用している場合は、ご使用のランタイム環境に付属の jsig ライブラリ・ファイルを指定します。

7. マネージャを次のように有効にします。
   ```
   <Manager enabled="true" />
   ```
8. (オプション) RPC ポートのホスト・ノードを次のように定義します。

   特に複数のネットワーク・カードを使用する環境では、このプロパティを使用して、クラスタで使用するネットワーク・インタフェースを指定します。
   ```
   <Host>thehostname</Host>
   ```
9. RPC ポート値を指定します。

   (オプション) `Port SSL` 値を true に設定して、クラスタで SSL を有効にします。
   ```
   <Port ssl="true">19011</Port>
   ```
10. マネージャを有効にしたノードで、キャッシュ・ポート値を指定します。
    ```
    <Port>19011</Port>
    ```
11. (オプション) キャッシュのホスト・ノードを定義します。

    特に複数のネットワーク・カードを使用する環境では、このプロパティを使用して、クラスタで使用するネットワーク・インタフェースを指定します。
    ```
    <Host>thehostname</Host>
    ```

12. キャッシュのユニークな名前とパスワードを定義します。このクラスタに追加されたすべてのノードは、キャッシュを操作する場合に同じ名前とパスワードを使用します。

```
  <Name>test-name-1</Name>
  <Password>test-password-1</Password>
```

13. (オプション) マネージャを有効にしたノードでマルチキャスト配信を有効にするには、enabled 値を `true` に設定し、`Group` と `Port` の各値を入力します。

```
<Multicast enabled="true">
            <Group>224.2.2.7</Group>
            <Port>54323</Port>
        </Multicast>
```

マルチキャストを有効にしない場合は、マネージャ・ノードを有効にして定義します。

```
<Multicast enabled="false">
 [...]
</Multicast>
<Managers enabled="true">
  <Manager>localhost:19001</Manager>
</Managers>
```

すべてのノードのマルチキャスト・ステータスは同一とします。

14. 永続性を有効または無効にします。

デフォルトでは、この値は `false` に設定されています。永続性を有効にした場合、永続性情報の格納先のディレクトリを変更できます。1 つのクラスタ内のすべてのノードが同じ永続性ディレクトリを参照する必要があります。

```
<Persistence enabled="true">
    <Directory>${ESP_STORAGE}</Directory>
</Persistence>
```

15. セキュリティの値を定義します。

`File`、`Policy`、`Keystore` など、セキュリティの値は既存のセキュリティ設定と整合しなければなりません。

```
<Security>
        <Csi>
            <File>${ESP_HOME}/security/csi_ldap.xml</File>
            <!--Policy>${ESP_HOME}/security/policy.xml</Policy-->
        </Csi>
        <Keystore>
            <Type>JKS</Type>
            <File>${ESP_HOME}/security/keystore.jks</File>
            <Password>pass123</Password>
            <KeyPassword>pass123</KeyPassword>
            <Algorithm>RSA</Algorithm>
        </Keystore>
    </Security>
```

注意： `<Policy>` パラメータは LDAP 認証にのみ適用されます。

参照：

- *クラスタの起動* (27 ページ)

クラスタ管理ツール

クラスタ管理ツールはクラスタ管理用の補助ツールです。プロジェクトとワークスペースの追加と削除、既存のプロジェクトのクエリ、開始、停止を実行します。

クラスタ管理ツールは、対話型モードまたはコマンド・ライン・モードで動作します。対話型モードでは、クラスタ・マネージャにいったん接続すれば、終了するまでコマンドを実行できます。コマンド・ライン・モードでは、各コマンドの実行後にユーティリティからログアウトされます。したがって、コマンドを指定するたびにクラスタ・マネージャに接続します。

注意： サポートされるコマンドを除き、パラメータの大文字と小文字は区別されません。

セキュリティなしを設定した状態でクラスタ・マネージャに接続するには、次のようにします。

```
esp_cluster_admin --uri=esp://<host>:<port>
```

RSA 認証を設定した状態でクラスタ・マネージャに接続するには、次のようにします。

```
esp_cluster_admin --uri=esp(s)://<host>:<port> --keyalias=
<keyalias> --storepass=<storepass> --keystore=<keystore>
```

Kerberos 認証または LDAP 認証を設定した状態でクラスタ・マネージャに接続するには、次のようにします。

```
esp_cluster_admin --uri=esp(s)://<host>:<port> --username=<user> --
password=<password>
```

表 1 : サポートされるコマンド

コマンド	機能
対話型モード： **get managers** コマンド・ライン・モード： **--get_managers**	クラスタ内のマネージャの host-name:rpc-port のペアが返されます。
対話型モード： **get controllers** コマンド・ライン・モード： **--get_controllers**	クラスタ内のコントローラのリストが返されます。

コマンド	機能
対話型モード： **get workspaces** コマンド・ライン・モード： **--get_workspaces**	クラスタ内のワークスペースの名前が返されます。
対話型モード： **get projects** コマンド・ライン・モード： **--get_projects**	引数リストと一致するプロジェクトとその状態のリストが返されます。
対話型モード： **get project <workspace-name>/<project-name>** コマンド・ライン・モード： **--get_projectdetail**	関連するワークスペース内のプロジェクトが表示されます。
対話型モード： **get streams <workspace-name>/<project name>** コマンド・ライン・モード： **--get_streams**	関連するワークスペース内のストリームが表示されます。
対話型モード： **get schema <workspace-name>/<project-name> <stream-name>** コマンド・ライン・モード： **--get_schema**	関連するストリームのスキーマが表示されます。
対話型モード： **add workspace <workspace-name>** コマンド・ライン・モード： **--add_workspace**	ワークスペースを追加します。
対話型モード： **add project <workspace-name>/<project-name> <ccx> [<ccr>]** コマンド・ライン・モード： **--add_project**	プロジェクトを追加します。 コントローラ名はオプションです。指定しない場合、アプリケーションの起動時にコントローラがランダムに選択されます。 ランダムに選択されたコントローラの場合は、起動後に弱い結び付きになります。 コントローラ名を指定した場合、デフォルトではコントローラの結び付きは弱い結び付きになります。

コマンド	機能
対話型モード： **remove workspace <workspace-name>** コマンド・ライン・モード： **--remove_workspace**	ワークスペースを削除します。
対話型モード： **remove project <workspace-name>/<project-name>** コマンド・ライン・モード： **--remove_project**	プロジェクトを削除します。ユーザは最初にプロジェクトを停止します。
対話型モード： **start project <workspace-name>/<project-name> [timeout (sec)] [<instance-index>]** コマンド・ライン・モード： **--start_project**	プロジェクトを開始します。コントローラとの強い結び付きを設定してプロジェクトを追加した場合にコントローラが使用できないと、起動は失敗します。
対話型モード： **stop project <workspace-name>/<project-name> [timeout (sec)] [<instance-index>]** コマンド・ライン・モード： **--stop_project**	プロジェクトを停止します。
対話型モード： **stop node <node-name>** コマンド・ライン・モード： **--stop_node**	ノードを停止します。
対話型モード： **encrypt <clear-text>** コマンド・ライン・モード： **--encrypt_text**	プレーン・テキスト・データを暗号化します。設定ファイル内のパスワードを暗号化する場合に使用します。
対話型モード： **deploykey <new-username> <keystore> <storepass> <key-alias> [<store-type>]** コマンド・ライン・モード： **--deploy_key**	新しいユーザ・キーをキーストアに配備することによって新しいユーザを追加します。

コマンド	機能
対話型モード： **reload policy** コマンド・ライン・モード： **--reload_policy**	実行中のクラスタに `policy.xml` ファイルを再ロードします。既存のポリシー・ファイルを最近更新した場合は、再ロード時に新しいポリシー設定に照らし合わせてクラスタが再検証されます。
対話型モード： **connect**	プロジェクトをクラスタに接続または再接続します。このコマンドは対話型モードのみです。
対話型モード：**quit** または **exit**	対話型モードからログアウトされます。ユーティリティに再びアクセスするには、認証を設定しないことを選択していない限り、ユーザ名とパスワードを指定します。
対話型モード： **> help** コマンド・ライン・モード： **--help**	ユーティリティのコマンドと使用方法に関する説明がプレーン・テキストで表示されます。

次の対話型モードのコマンドでは、一部のパラメータとコマンドの使用方法を示します。

```
esp_cluster_admin --uri=esp://cluster_server:19011 --username=me --
password=sybase get managers
esp_cluster_admin --uri=esp://cluster_server:19011 --username=me --
password=sybase get workspaces
esp_cluster_admin --uri=esp://cluster_server:19011 --username=me --
password=sybase get projects
```

次のコマンド・ラインでは、一部のパラメータとコマンドの使用方法を示します。

```
esp_cluster_admin --uri=esp://cluster_server:19011 --username=me --
password=sybase --get_managers
esp_cluster_admin --uri=esp://cluster_server:19011 --username=me --
password=sybase --get_workspaces
esp_cluster_admin --uri=esp://cluster_server:19011 --username=me --
password=sybase --get_projects
```

クラスタ環境でのログ・ストアの使用

CREATE LOG STORE 文の `filename` プロパティでパスを指定することによって、ログ・ストアの相対パスまたは絶対パスを定義します。

クラスタ環境内では、**CREATE LOG STORE** 文の `filename` プロパティでファイル・パスを指定します。プロジェクトあたり 1 つのログ・ストアを作成します。ログ・ストア定義の filename プロパティでは相対パスを使用してください。ログ・ファイルの宛先としては、プロジェクト・ファイルが格納されているベース・ディレクトリをおすすめします。

推奨される相対ファイル・パスの代わりにログ・ストアの絶対ファイル・パスを定義することもできますが、この方法はおすすめしません。ただし、プロジェクトを別のマシンで再開する場合は、絶対パスも有効でなければなりません。コントローラの結び付きを strong positive に設定して、プロジェクトを常に同じマシンで開始するように設定します。

1. CCL エディタで、次のように **CREATE LOG STORE** 文を使用してログ・ストアを作成します。

```
CREATE [DEFAULT] LOG STORE storename
PROPERTIES
filename='filepath'
    [sync={ true | false},]
    [sweepamount=size,]
    [reservepct=size,]
    [ckcount=size,]
    [maxfilesize=filesize];
```

2. `filename` プロパティに、ログ・ストアのロケーションの相対ファイル・パス (推奨) または絶対ファイル・パスのどちらかを入力します。

(推奨) 相対パス	相対パスはベース・ディレクトリを基準とします。相対パスを使用する場合、ログ・ストアは自動的にベース・ディレクトリを参照します。相対パスはディレクトリ・スタックを参照しません。つまり、パスの先頭はドライブ文字でもスラッシュ (/) でもありません。
(非推奨) 絶対パス	絶対パスは、現在の作業ディレクトリ (ベース・ディレクトリ) にかかわらず、マシン上の任意のロケーションを参照します。Windows システムの場合、絶対パスの先頭はドライブ文字です。UNIX システムと Solaris システムの場合、絶対パスの先頭はスラッシュ (/) です。

相対パスのロケーションはすべてのクラスタ・ノードがアクセスできる共有ディスクにしてください。ログ・ストアのパスは、ログ・ストア定義の `filename` プロパティに指定したパスを基準とするパスです。相対パスを使用すると、ログ・ストアは自動的に `<base-directory>/<workspace-name>.<project-name>.<instance-number>.<logstore-path>` に配置されます。ベース・ディレクトリの定義はクラスタ設定ファイル (`<node-name>.xml`) の controller セクションにあります。

絶対パスも使用できますが、相対パスをおすすめします。絶対パスを使用するには、まず、すべてのクラスタ・ノードが、指定した絶対パスとの間で読み取りと書き込みを実行できることを確認します。つまり、ロケーションはすべてのクラスタ・ノードで同じでなければなりません。また、2 つのプロジェクトがログ・ストアのロケーションに同じパスを使用していないことも確認しま

す。共有ディスクの使用に対応していない場合は、強い結び付きを使用して、プロジェクトが常に同じクラスタ・ノードで開始するように設定できます。

3. **CREATE LOG STORE** 文のその他のプロパティに適切な値を入力します。
4. [Compile] をクリックします ([F7] キーを押します)。
5. [Run Project] をクリックします。

参照：
- *プロジェクト・ログ* (3 ページ)
- *ログ・ストアのサイズ設定* (8 ページ)

プロジェクト設定ファイルのサンプル

このサンプルのプロジェクト設定 CCR ファイルを使用して、XML ベースのプロジェクト設定ファイルを作成および変更します。

スタジオのプロジェクト設定エディタを使用するだけでなく、テキスト・エディタで XML を直接編集することによってプロジェクト設定 CCR ファイルを作成および変更することもできます。

注意： テキスト・エディタで CCR ファイルを開くには、システムでファイルを見つけて、任意のエディタでそのファイルを開くか、[File Explorer] ペインで CCR ファイルを右クリックし、[Open With] > [Text Editor] と選択します。

スタジオのご使用のバージョンには次のサンプルが添付されています。クラスタ、バインディング、パラメータ、アダプタ、プロジェクト設定など、設定対象に従ってセクションに分割されています。

```
<Configuration>
    <Runtime>

        <Clusters>
          <!-- we need this only if we have a project/stream binding
-->
            <Cluster name="cluster1" type="local">
                <Username>atest</Username>
                <Password>secret</Password>
            </Cluster>
            <Cluster name="cluster2" type="local">
                <Username>user2</Username>
                <Password>Pass1234</Password>
                <!-- if cluster is remote, we need the Managers
section. if cluster local managers are retrieved internally from the
container -->
            </Cluster>
           <Cluster name="cluster4" type="local"> <!-- this is local
like cluster2 above, but with a different set of credentials -->
                <Username>user4</Username>
                <Password>Pass12345</Password>
```

```
                <!-- if cluster is remote, we need the Managers
section. if cluster local managers are retrieved internally from the
container -->
            </Cluster>
        </Clusters>

        <Bindings>
            <Binding name="BaseInput">
              <Cluster>cluster1</Cluster> <!-- this is always needed
-->
                <Workspace>ws1</Workspace>
                <Project>project1</Project>
               <BindingName>BaseInputBinding</BindingName> <!-- this
is for plat-in adapter name-->
                <RemoteStream>BaseInput</RemoteStream>
            </Binding>
            <Binding name="localStreamName2">
              <Cluster>cluster2</Cluster> <!-- this is always needed
-->
                <Workspace>ws2</Workspace>
                <Project>prj2</Project>
              <BindingName>Stream2Binding</BindingName> <!-- this is
for plat-in adapter name-->
                <RemoteStream>remoteStreamName2</RemoteStream>
            </Binding>
            <Binding name="localStreamName3">
              <Cluster>cluster2</Cluster> <!-- this is always needed
-->
                <Workspace>ws3</Workspace>
                <Project>prj3</Project>
              <BindingName>Stream3Binding</BindingName> <!-- this is
for plat-in adapter name-->
                <RemoteStream>remoteStreamName3</RemoteStream>
            </Binding>
            <Binding name="localStreamName4">
              <Cluster>cluster4</Cluster> <!-- this is always needed
-->
                <Workspace>ws4</Workspace>
                <Project>prj4</Project>
              <BindingName>Stream4Binding</BindingName> <!-- this is
for plat-in adapter name-->
                <RemoteStream>remoteStreamName4</RemoteStream>
            </Binding>
            <Binding name="localStreamName5">
              <Cluster>cluster1</Cluster> <!-- this is always needed
-->
                <Workspace>ws1</Workspace>
                <Project>prj2</Project>
              <BindingName>Stream5Binding</BindingName> <!-- this is
for plat-in adapter name-->
                <RemoteStream>remoteStreamName5</RemoteStream>
            </Binding>
        </Bindings>

        <Parameters>
            <Parameter name="myparam1">foo</Parameter>
```

```
            <Parameter name="myparam2">1234</Parameter>
            <Parameter name="myparam3">true</Parameter>
        </Parameters>

        <Adapters>
            <Adapter>
                <Properties name="datalocation1">
                    <Property name="Host">myhost1</Property>
                    <Property name="Port">5555</Property>
                </Properties>
            </Adapter>
            <Adapter>
                <Properties  name="datalocation2">
                    <Property name="Host">myhost2</Property>
                    <Property name="Port">6666</
Property>
                </Properties>
            </Adapter>
        </Adapters>

    </Runtime>

    <Deployment>

        <Project>
            <Options>
                <Option name="ignore-config-topology">true</Option>
                <Option name="jvm-heap">512MB</Option>
            </Options>
        </Project>

        <Cluster>
            <Failover></Failover>
            <Affinities>
                <Affinity type="controller"
charge="positive">myController</Affinity>
            </Affinities>
        </Cluster>

    </Deployment>

</Configuration>
```

参照：

- *プロジェクト配備オプション* (21 ページ)
- *クラスタ・アーキテクチャ* (17 ページ)
- *クラスタの設定* (29 ページ)

第 3 章 Event Stream Processor のセキュリティ

Event Stream Processor のセキュリティはクラスタ・マネージャによって中央で管理されます。クラスタ内で実行されているすべてプロジェクトに、そのクラスタに対して定義されたセキュリティ・ルールが適用されます。

Event Stream Processor は既存の Kerberos と LDAP の各セキュリティ・システムと統合します。また、組み込みの RSA 証明書ベースのセキュリティも用意されています。セキュリティに RSA を使用する場合、サーバ接続を要求するユーザは、有効な RSA キー・エイリアス、プライベート・キーがあるキーストア、キーストアのパスワードを指定する必要があります。セキュリティに LDAP または Kerberos を使用する場合、ユーザは LDAP または Kerberos のユーザ名とパスワードを指定する必要があります。

LDAP は役割ベースのポリシー設定をサポートしています。役割に基づき、プロジェクトまたはそのリソースへのアクセス権を提供するか制限するポリシーを定義して、セキュリティ実装をさらに絞り込みます。たとえば、すべてのプロジェクトへのフル・アクセスを許可する管理者役割を定義できます。

認証

一連のクレデンシャルを使用してユーザの ID を確認します。

ユーザがサーバにログインするときに、そのユーザのクレデンシャルがセキュリティ・プロバイダで確認されます。LDAP と Kerberos では、ユーザ名とパスワードの組み合わせがクレデンシャルと見なされます。RSA 認証では、キーのエイリアス、プライベート・キーがあるキーストア、キーストアのパスワードが必要です。サーバで設定できる認証のタイプは一度に 1 つのみです。セキュリティなしの使用も選択できます。

ユーザがサーバに適切なセキュリティ・クレデンシャルを提供した場合、サーバはそのユーザを認証します。認証が成功すると、サーバはそのユーザを有効なクライアントと見なし、ログインが完了します。ユーザはセッション ID を受け取ります。それ以降の通信では、クライアントはそのセッション ID を使用して自分自身を確認します。

アクセス制御とパーミッションは LDAP でのみサポートされます。

セキュリティなしのサーバの設定

ログイン時にどのようなユーザ名とパスワードのペアでも受け入れるようにサーバを設定します。

セキュリティなしモードでは、ユーザはログインする必要はありますが、どのようなユーザ名とパスワードのペアを使用してもログインできます。

1. 任意のテキスト・エディタを使用して `$ESP_HOME/cluster/nodes/<node-name>/<node-name>.xml file` ファイルを開きます。
2. Security セクションの CSI フィールドを`<File>csi_no_security.xml</File>` に変更します。

RSA 認証

RSA 認証には、プライベート・キーとパブリック・キーの組み合わせと、ユーザ名の役目を果たすエイリアスが必要です。

RSA 認証では、パスワードの代わりにパブリック・キーとプライベート・キーを使用して ESP サーバで認証し、セキュアなログイン・システムを作成します。各キーには、ログイン・ユーザ名の役目を果たすエイリアスが割り当てられます。ユーザは、クラスタ・マネージャに接続するときに、キーのエイリアス、プライベート・キーがあるキーストア、キーストアのパスワードを指定します。メッセージに署名するにはプライベート・キーが必要です。また、対応するパブリック・キーをサーバに提供します。

サーバは、ユーザ名または証明書のエイリアスを使用して、キーストアからパブリック・キーを取得します。パブリック・キーによって、クライアント／ユーザから送信された署名付きメッセージを確認します。パブリック・キーはサーバに配備します。

Java の keytool による RSA キーの生成

クライアントが Java の場合は、Java の keytool を使用して RSA 認証のパブリック・キーとプライベート・キーを作成します。

RSA 認証では、パスワードの代わりにパブリック・キーとプライベート・キーを使用して ESP サーバで認証します。クライアントが Java の場合は、`Java keytool` ユーティリティを使用して RSA キーを生成します。

1. コマンド・プロンプトまたは端末を開きます。
2. ESP_JAVA_HOME を Java のインストール先に設定します。
3. パスに `$ESP_JAVA_HOME/bin` を追加します。
4. ユーザが指定したエイリアスでプライベート／パブリック・キーを作成するには、次のように入力します。

```
keytool -genkey -keyalg RSA -alias <alias/username> -
keystore keystore.jks -storepass <password> -keypass
<password>
```

<alias/username> には、プライベート・キーとパブリック・キーに対するユーザ選択のエイリアスを指定します。エイリアスは、RSA を使用してログインする場合にユーザ名の役目を果たします。*<password>* には、エイリアスに関連付けられたプライベート・キーにアクセスする場合に必要なユーザ選択のパスワードを指定します。

5. クラスタ管理ツールを使用してパブリック・キーをサーバに配備します。

```
esp_cluster_admin --uri=esp[s]://host-name:port
--keystore=<keystore>
--storepass=<storepass>
--keypass=<keypass>
--key-alias=<alias>
```

これで、クラスタ・マネージャはパブリック・キーを使用できるようになります。このキーが、クラスタ・マネージャが使用するパブリック・キーになり、このキーによって、認証プロセス時にクライアントのプライベート・キーから送信された署名メッセージが確認されます。

RSA に合わせたサーバの設定

RSA 認証を使用するようにサーバを設定するには、`server.xml`、`csi.xml`、`csi_rsa.xml` の各ファイルを変更します。

セキュリティ設定情報はすべてのクラスタ・マネージャがアクセスできる XML ベースの設定ファイルに保持されます。

注意： Sybase Event Stream Processor で適切なセキュリティを実現するには、RSA 認証と SSL の両方を使用します。RSA を単独では使用しないでください。

1. 次のファイルを開きます。
   `$ESP_HOME/cluster/<node-name>/<node-name>.xml`

   このファイルは任意のテキスト・エディタを使用して編集できます。
2. クラスタ設定ファイルの Security セクションの CSI フィールドに、次のように入力します。

```
<File>csi_rsa.xml</File>
```

Kerberos に合わせたサーバの設定

Kerberos 認証を使用するようにサーバを設定するには、`csi.xml` ファイルと `csi_kerberos.xml` ファイルを変更します。

Event Stream Processor では Kerberos のチケットベースの認証をサポートしていないため、Kerberos のサポートは限定されています。Kerberos 認証を実行するには、

ユーザ名とパスワードを指定します。セキュリティ設定情報はすべてのクラスタ・マネージャがアクセスできる XML ベースの設定ファイルに保持されます。

1. テキスト・エディタを使用して `$ESP_HOME/cluster/<node-name>/<node-name>.xml` ファイルを開きます。このファイルには、次の行があります。
   ```
   <Property name="java.security.krb5.realm">REALM_PLACEHOLDER</
   Property>
           <Property name="java.security.krb5.kdc">KDC_PLACEHOLDER</
   Property>
   ```
2. 次の行を `$ESP_HOME/cluster/<node-name>/<node-name>.xml` ファイルに追加します。
   ```
   <Csi>
           <File> csi_kerberos.xml</File>
   </Csi>
   ```
3. サーバを再起動します。すべてのクラスタ・マネージャを再起動します。

LDAP 認証

LDAP はユーザ名とパスワードによる認証オプションで、アクセス制御にも対応しています。

UNIX のログイン・クレデンシャルを使用して LDAP にサインインします。

LDAP 認証は、ユーザ・アクセスをユーザの役割または権限レベルに基づいて特定の機能に制限できる、Event Stream Processor での唯一の認証形式です。

LDAP に合わせたサーバの設定

LDAP 認証を使用するようにサーバを設定するには、`csi.xml` ファイルまたは `csi_ldap.xml` ファイルを変更します。

インストール時に、LDAP 認証とアクセス制御の両方に対して同時に LDAP を設定できます。

セキュリティ設定情報はすべてのクラスタ・マネージャがアクセスできる XML ベースの設定ファイルに保持されます。

1. 任意のテキスト・エディタを使用して `$ESP_HOME/cluster/<node-name>/<node-name>.xml` ファイルを開きます。
2. 次の行をファイルに追加します。
   ```
   <Csi>
           <File> csi_ldap.xml</File>
   </Csi>
   ```
3. まだ設定されていない場合は、`csi_ldap.xml`(または `csi_openldap.xml`)ファイルを編集します。

4. すべてのクラスタ・マネージャを含め、サーバを再起動します。

アクセス制御
LDAP はアクセス制御の形式で追加レベルのセキュリティをサポートしています。アクセス制御では、サーバはユーザの役割に基づきユーザ・アクセスを制限できます。

アクセス制御はクラスタ・マネージャに実装し、`policy.xml` ファイルに設定します。管理者は、アクセス対象のリソース、属する役割、実行できるアクション間の関係を指定できます。1 つの設定をすべてのクラスタ・マネージャで使用できます。

役割、リソース、アクション
アクセス制御システムを使用してユーザ・アクセスを制限するには、定義済みの役割をユーザに割り当てます。この役割をリソースとリソースの権限設定アクションに関連付けます。

役割
役割はグループ名と同等で、セキュリティ・プロバイダ・サーバに定義されます。アクセス制御プロセスで、セキュリティ・プロバイダ・サーバはユーザが特定のグループに属するかどうかを判断します。属すると判断された場合、そのグループがユーザの役割と見なされ、ユーザがアクセスできる使用可能なリソースとアクションが制限されます。

*any 役割というオプションもあります。これは、すべてのユーザがその役割に属することを意味します。*any 役割が使用された場合は、ユーザがその役割に属するかどうかをチェックするためにセキュリティ・プロバイダ・サーバが呼び出されることはありません。

リソース
リソースにはクラスタとプロジェクト (プロジェクト・サーバ) の 2 つのタイプがあります。クラスタ・リソースは次のカテゴリーに分類されます。

- アプリケーション - プロジェクトの追加、削除、開始、停止、取得
- ワークスペース - ワークスペースの追加、削除、取得
- セキュリティ - RSA ユーザを追加する場合と、ポリシー・ファイルを再ロードする場合
- ノード - コントローラ・ノードとマネージャ・ノードの取得、ノードの停止

ストリーム、ウィンドウなどのリソースは (プロジェクト・サーバで実行される) プロジェクトのリソースと見なされます。ポリシー・ファイル内のリソースは、"/" を使用して子を示すツリーのような形式で定義されます。たとえば、workspace1/project1 というプロジェクトがあり、このプロジェクトに stream1 要素

と window1 要素がある場合、ポリシー・ファイルではこれらのリソースを次のようにさまざまな方法で定義できます。

- workspace1
- workspace1/project1
- workspace1/project1/stream1
- workspace1/project1/window1

Event Stream Processor は階層形式のリソース使用権利をサポートしています。つまり、リソース workspace1 に対してアクションを実行する権限がユーザに付与されている場合は、workspace1 下のすべてのリソースに対して同じアクションを実行する権限がそのユーザに自動的に付与されます。

*any オプションはリソースの一部としても使用できます。これはクラスタ内のすべてのリソースを指します。*any リソース・オプションは、workspace1/*any など、細分化方式では定義できません。

アクション
すべてのリソースで使用できるアクション・タイプ (アクセス方法) は次の 4 つです。

アクション・タイプ	説明
READ	特定のリソースのオープン、取得、サブスクライブを実行できます。ただし、変更は加えられません。
WRITE	特定のリソースの作成、追加、削除、更新を実行できます。
START	プロジェクトを開始できます。
STOP	プロジェクトまたはノードを停止できます。

アクセス制御のシナリオ
クライアントがログイン・コールを実行すると、セキュリティ・サービスによってユーザが認証されます。役割 A のユーザがリソース B にアクセスしようとすると、検証によって、そのリソースにアクセスして希望のアクションを実行する権限がユーザに付与されていることが確認されます。

ポリシー・ファイルには、役割 A のユーザがリソース B にアクセスしてアクション READ を実行できることが設定されています。役割 A のユーザがリソース B でアクション WRITE を実行しようとした場合、ユーザにその権限はありません。ただし、ユーザがリソース B で READ を実行しようとした場合は、このアクションの権限が付与されます。

アクセス制御の設定
XML ポリシー・ファイルを編集して、役割、リソース、アクション間の関係を作成します。

アクセス制御の管理にはクラスタ・マネージャを使用します。役割とリソースとの関係は 1 つの XML ポリシー・ファイルに保持され、このファイルをプロジェクト内のすべてのクラスタ・マネージャが使用します。このファイルは、クラスタ・マネージャを起動すると自動的にロードされます。実行時にポリシー・ファイルを再ロードするには、クラスタ管理ツールを使用します。

1. 任意のテキスト・エディタを使用して XML ポリシー・ファイルを開きます。
2. <Policies/> タグを追加して、作成するすべてのポリシーを入れます。
   <Policies/> タグの中には複数のポリシーを入れることができます。
3. 新しいポリシーを開始するには、<Policy/> タグを追加します。
4. ポリシー・タイプとして Project または Cluster を指定します。
   ```
   <Policy type="Project">
   ```
5. ポリシーの新しい役割を作成するには、<Subjects/> タグの中に <Role/> タグを追加します。
   <Subjects/> タグの中には複数の役割を入れることができますが、<Subjects/> タグ内の役割には、すべてのリソースとアクションが関連付けられます。役割を別のリソースとアクションに関連付けるには、<Policy> タグを使用して別個のポリシーを作成します。
6. <Role/> タグの中に作成する新しい役割にグループまたは役割を追加します。
7. リソースを役割に関連付けるには、<Resource/> タグで各リソースを指定し、それらのリソースを <Resources/> タグで囲みます。
8. アクションをリソースに関連付けるには、<Action/> タグで各アクションを指定し、これらのアクションを <Actions/> タグで囲みます。

ポリシー・ファイルのサンプルを次に示します。investment の役割では、2 つのリソースの読み取り、書き込み、開始、停止を実行できます。

```
<Policies>
 <Policy type= "Project">
       <Subjects>
          <Role>investment</Role>
       </Subjects>
       <Resources>
         <Resource>Default/PassThrough/vwapTrades</Resource>
         <Resource>Default/Pass1</Resource>
       </Resources>
       <Actions>
         <Action>read</Action>
         <Action>write</Action>
         <Action>stop</Action>
```

```
          <Action>start</Action>
        </Actions>
  </Policy> </Policies>
```

参照：
- *アクセス制御に合わせたサーバの設定* (48 ページ)

アクセス制御に合わせたサーバの設定
アクセス制御のセキュリティ・プロバイダとして使用できるのは LDAP のみです。

役割、リソース、アクション間の関係は XML ポリシー・ファイルを使用して管理します。アクセス制御を使用するようにサーバを設定するには、CSI ファイルを使用します。

1. LDAP 認証を使用するようにセキュリティを設定します。
2. CSI ファイルを編集する場合は、次のようにポリシー・ファイルを指定します。
   ```
   <Csi>
          <Policy>policy.xml</Policy>
          <File>csi_openldap.xml</File>
   </Csi>
   ```
3. `csi_ldap.xml` ファイルを必要に応じて編集します。
4. ポリシー・ファイルに役割、リソース、アクションを設定します。

クライアントがログイン・コールを実行すると、セキュリティ・プロバイダによってユーザが認証されます。ユーザがリソースでアクションを実行しようとすると、サーバは、そのユーザの役割がそのアクションとリソースへのアクセスを許可されているかどうかを判断します。許可されている場合、リソースに対してアクションを実行する権限がユーザに付与されます。それ以外の場合、アクションは拒否されます。

参照：
- *アクセス制御の設定* (47 ページ)

アクセス制御

ユーザの認証された ID と役割に基づき、データとサービスへのアクセスを制限します。

SSL (Secure Sockets Layer) 接続

Event Stream Processor はネットワークの SSL 接続をサポートして、クライアント・アプリケーションと ESP サーバ間の通信のプライバシーを確保しています。

SSL はクラスタ内のリモート・プロシージャ・コール (XMLRPC) でサポートされています。クラスタ内のノードは HTTP または HTTPS のどちらかをサポートしますが、両方を同時にはサポートしません。SSL がクラスタで有効であれば、クラスタ内のコンポーネントもすべて SSL 対応になります。

Event Stream Processor で、SSL はデフォルトで有効に設定されています。ユーザは Event Stream Processor をインストールするときに SSL を無効にできます。インストール時にクラスタを設定しなかった場合や新しいクラスタを作成する場合は、クラスタ設定ファイルで SSL を有効に設定できます。

参照：
- *クラスタの設定* (29 ページ)

Java キーストアの生成

Java キーストアは、X.509 証明書とプライベート・キーを格納し配備するための便利なメカニズムです。

プライベート・キーを作成するには、`$JAVA_HOME/bin` ディレクトリにあるキーストア・ツールを使用します。Event Stream Processor ユーティリティを呼び出す場合は、これらのプライベート・キーを使用する必要があります。たとえば、**esp_cluster_admin** には自己署名プライベート・キーが必要です。

注意： 手順 2 ～ 9 ではサンプル値を使用します。実際に入力する値は異なる可能性があります。

1. コマンド・ラインから、次のスクリプトを実行して自己署名キーを生成します。

   ```
   keytool -genkey -alias username -keyalg RSA
   -keysize 1024 -keystore filename.jks
   ```

   注意： コマンドに必須の username と keystore filename は変数です。

   [Return] キーを押します。
2. 新しいキーストアのパスワードを入力します。

   ```
   Enter keystore password: testpass
   ```

注意： セキュリティ上の理由から、入力したパスワードは表示されません。

[Return] キーを押します。

3. 新しいキーストアのパスワードを再入力します。

```
Re-enter new password: testpass
```

注意： セキュリティ上の理由から、入力したパスワードは表示されません。

[Return] キーを押します。

4. 姓名を入力します。

```
What is your first and last name?
  [Unknown]:  john smith
```

[Return] キーを押します。

5. 組織単位の名前を入力します。

```
What is the name of your organizational unit?
  [Unknown]:  business
```

[Return] キーを押します。

6. 組織の名前を入力します。

```
What is the name of your organization?
  [Unknown]:  company name
```

[Return] キーを押します。

7. 市町村の名前を入力します。

```
What is the name of your City or Locality?
  [Unknown]:  new york
```

[Return] キーを押します。

8. 都道府県の名前を入力します。

```
What is the name of your State or Province?
  [Unknown]:  new york
```

[Return] キーを押します。

9. 2 文字の国コードを入力します。

```
What is the two-letter country code for this unit?
  [Unknown]:  us
```

[Return] キーを押します。

10. yes または y と入力して、情報が正しいことを確認します。

```
Is CN=john doe, OU=business, O=company name, L=new york, ST=new
york, C=us correct?
  [no]:  y
```

[Return] キーを押します。

11. `<ceptest>` のキー・パスワードを入力し、[Return] キーを押します。キーのパスワードとキーストアのパスワードが同じ場合は、単に [Return] キーを押して、必要な値を指定します。

```
Enter key password for <ceptest>
        <RETURN if same as keystore password>:
```

注意： セキュリティ上の理由から、入力したパスワードは表示されません。

新しいキーストア・ファイルが作成されます。

pem 形式のプライベート・キーの生成

pem 形式のプライベート・キーを生成するように Java の keytool を変換します。

前提条件

ご使用のマシンに JDK 1.6 がインストールされていることを確認します。

手順

esp_client、**esp_convert**、**esp_upload**、**esp_subscribe**、**esp_cnc**、**esp_query** など、複数の Event Stream Processor ユーティリティで pem 形式のプライベート・キーが必要です。

キーストア・ツールは `$JAVA_HOME/bin` ディレクトリにあります。

1. コマンド・ラインから、次のスクリプトを実行して Java キーストアを OpenSSL で使用される PKCS12 形式のキーストアにエクスポートします。

```
                keytool -importkeystore -srckeystore
filename.jks  -destkeystore exportfilename.p12 -
deststoretype PKCS12
```

注意： このコマンドで必須の filename と exportfilename は変数です。

[Return] キーを押します。

2. 次のコマンドを実行して、PKCS12 形式のキーストアを pem 形式のプライベート・キーに変換します。

```
                openssl pkcs12 -in filename.p12 -out
username.private.pem -nodes
```

注意： このコマンドに必須の username は変数です。

[Return] キーを押します。

セキュリティ・ログの表示

セキュリティ・ログ・ファイルには、パスワード変更など、これまでに加えられたすべてのセキュリティ変更が記述されます。

クラスタ・ログ・ファイル内のサーバ・ログを表示します。

第 4 章　設定ファイルのパスワード暗号化

Event Stream Processor の **esp_cluster_admin** ユーティリティは、内部アダプタ、サービス、プロジェクトの各設定ファイルのパスワード暗号化をサポートします。また、Event Stream Processor には、外部アダプタ設定ファイルのパスワードを暗号化するための `encrypt.sh` スクリプトも用意されています。

esp_cluster_admin の呼び出し

esp_cluster_admin ユーティリティへのログインと終了を実行します。

前提条件
環境変数を設定します。

手順
ユーティリティとそのサポート・コマンドの詳細については、『ユーティリティ・ガイド』を参照してください。

1. **esp_cluster_admin** を呼び出します。
   ```
   $ESP_HOME/bin/esp_cluster_admin --uri=esp://<host>:<port> --
   username=<user-name> --password=<password>
   ```
   ユーティリティにログインするにはユーザ名とパスワードを指定します。ログインのクレデンシャルは、選択した認証方法によって異なります。
2. 必要に応じて、コマンドを連続して実行します。
3. **exit** または **quit** を使用してユーティリティを終了します。

設定ファイルのパスワードの暗号化

esp_cluster_admin には、設定ファイルのパスワードに暗号化を適用するための **encrypt** コマンドが含まれています。

前提条件
ESP_HOME 環境変数を設定します。

手順

アダプタの `.cnxml` とデータベース・サービス設定ファイルは、プロジェクト環境のセットアップ時にのみ変更します。ただし、セットアップ以降もプロジェク

ト設定ファイルへのアクセスは必要なので、スタジオにはプロジェクト・ファイルのプロパティを変更する環境が用意されています。スタジオでのプロジェクト・ファイルの設定方法の詳細については、『スタジオ・ユーザーズ・ガイド』を参照してください。

1. テキスト・エディタを使用して、目的の設定ファイルを開きます。
2. **esp_cluster_admin** を呼び出し、クレデンシャルを指定します。
3. 設定ファイル内のパスワードに対して **encrypt** コマンドを実行するには、テキスト・エディタからパスワード・テキストを選択してコピーします。
   次のサンプル設定ファイルでは、パスワードは "Pass1234" です。
   ```
   <?xml version="1.0" ?>
   - <Services>
   - <Service Name="MyDBService" Type="DB">
     <Parameter Name="DriverType">JDBCASE</Parameter>
     <Parameter Name="Host">localhost</Parameter>
     <Parameter Name="Port">5000</Parameter>
     <Parameter Name="User">testID</Parameter>
     <Parameter Name="Password" encrypted="false">Pass1234</
   Parameter>
     </Service>
     </Services>
   ```
4. ユーティリティで、**encrypt** コマンドを入力し、その横に設定ファイルのパスワード・テキストを貼り付けます。
   ```
   --encrypt Pass1234
   ```
5. コマンドを実行します。
   このアクションによって、隠されたパスワードを含む暗号化されたテキストの文字列が次のように生成されます。
   ```
   OJ5f+g5FmzcEdcbonmSREyIHPoAf3O3o5LAK9drQp7J5a5snY4luj/
   kdnc61LHNARLA7fOQbp2x20PFMRyti2RTl5qgoUxMjIptDXBm3GIOvXso6AoPBG/
   RUaA1dV8giMySEK/GJfnxSSsURfAJm5OHSK8pdt7OBmil0CaSUZdc=
   ```
6. 暗号化されたテキストをユーティリティからコピーし、設定ファイルが表示されているテキスト・エディタに貼り付けます。Password パラメータにある元の **password** を **encrypted** テキストに置き換え、パラメータの暗号化属性を作成して true に設定します。
   この属性によって、サーバはパスワードを暗号化されたテキストとして認識し、実行時に復号化します。属性が false に設定されると、サーバはパスワードを暗号化されたテキストとして認識しないため、復号化せずにパスワードを処理しようとするのでエラーが発生します。

注意： 次の例では、データベース・サービス設定ファイルを使用します。

```
<?xml version="1.0" ?>
- <Services>
- <Service Name="MyDBService" Type="DB">
  <Parameter Name="DriverType">JDBCASE</Parameter>
  <Parameter Name="Host">localhost</Parameter>
```

```
  <Parameter Name="Port">5000</Parameter>
  <Parameter Name="User">testID</Parameter>
  <Parameter Name="Password" encrypted="true">OJ5f
+g5FmzcEdcbonmSREyIHPoAf3O3o5LAK9drQp7J5a5snY4luj/
kdnc61LHNARLA7fOQbp2x20PFMRyti2RTl5qgoUxMjIptDXBm3GIOvXso6AoPBG/
RUaA1dV8giMySEK/GJfnxSSsURfAJm5OHSK8pdt7OBmil0CaSUZdc=</
Parameter>
  </Service>
  </Services>
```

7. 設定ファイルを保存します。

アダプタの暗号化と復号化のスクリプト

Event Stream Processor には、外部アダプタの設定値を暗号化したり、暗号化された値の復号化をテストしたりする場合に役立つ 2 つのスクリプトが用意されています。

`encrypt.sh/bat` と `decrypt.sh/bat` は `$ESP_HOME/adapters` で入手できます。これらは、任意の独立したキーストアを使用して暗号化／復号化を実行できる独立ユーティリティです。

指定する必要がある値には、キーストア、エイリアス、キーストアのパスワードが含まれます。

Java 外部アダプタのパスワードの暗号化

独立したキーストアを使用して、外部アダプタ設定ファイルのパスワードを暗号化し、その暗号化された値を実行時に復号化するように ESP サーバに指示します。

前提条件

ESP_HOME 環境変数を設定します。Event Stream Processor は Java Runtime Environment 1.6.0.22 以降をサポートしています。

手順

Java 外部アダプタ設定ファイルには、ESP サーバが復号化を承認する場合に使用する暗号化アルゴリズムがあります。

1. 任意のテキスト・エディタを使用して、目的の外部アダプタ設定ファイルを開きます。
2. `encrypt.sh` スクリプトを呼び出します。

```
$JAVA_HOME/bin/java -cp jar/adapterapi.jar:jar/commons-
codec-1.3.jar com.sybase.esp.adapter.api.CryptUtils encrypt
<password> <alias/user> RSA <keystorepath>/keystore.jks
<keystorepassword>
```

a) 外部アダプタ設定ファイルからパスワード文字列をコピーし、`encrypt.sh` スクリプトの `<password>` 変数の位置に貼り付けます。
b) `<alias/user>` 変数をストアキーのエイリアス (ユーザ名) に置き換えます。
c) 外部アダプタが使用している認証方法の名前を指定します。デフォルトは RSA です。
d) `<keystorepath>` 変数を `keystore.jks` ファイルのファイル パスに置き換えます。
e) `<keystorepassword>` 変数をキーストアのパスワードに置き換えます。
f) スクリプトを実行します。
このアクションによって、隠されたパスワードを含む暗号化されたテキストの文字列が次のように生成されます。

```
ilNkDIv7MK99CvRHkVmDunuAvErHEyNdGZ
+VTe63PBMEbyZ2CfZf6iHhCtDXD6fR9jPYIT/
3FcyHmX2VL5xEeDL29KJP4xPS6d9/
TUIozJvJb9YhA8yyHUGv9iGUmtJdcN4vvQ1XJPSGHD84vIKSHQOfz8UlZKl07u
Jl54b47JXi+hIt1X3hZtGAaKuNt9BDo3KIgD4McehJFH2eT0vYmLHjWAL
+JoO4V0/+e9ZlgF4hzjpVkYaO5zik7WyWbvVzLcv4sT4A77CGq4/uo
+ZsJlGdBQ/qlSXDBUKBacHhmYBV1j5xZgxLPu2feEl1OGP/+27126/Lz0M/
JVeShDOw==
```

注意： `decrypt.sh` スクリプトを使用して、暗号化されたテキストを検証します。暗号化されたテキストに対して decrypt コマンドを実行するには、`decrypt.sh` スクリプトを呼び出し、`encrypt.sh` スクリプトの場合と同じクレデンシャルを指定します。

g) 暗号化されたテキストをスクリプトからコピーし、設定ファイルが表示されているテキスト・エディタに貼り付けます。**espPassword** パラメータにある元のパスワードを、暗号化されたテキストに置き換えます。次に、パラメータの **encrypted** 属性を作成して true に設定します。
true に設定した場合、この属性によってサーバはパスワードを暗号化されたテキストと認識するため、実行時にそのパスワードを復号化できます。属性を false に設定した場合、サーバはパスワードを暗号化されたテキストと認識しません。そのため、復号化せずにパスワードを処理しようとするのでエラーが発生します。

```
<espPassword
encrypted="true">ilNkDIv7MK99CvRHkVmDunuAvErHEyNdGZ
+VTe63PBMEbyZ2CfZf6iHhCtDXD6fR9jPYIT/
3FcyHmX2VL5xEeDL29KJP4xPS6d9/
TUIozJvJb9YhA8yyHUGv9iGUmtJdcN4vvQ1XJPSGHD84vIKSHQOfz8UlZKl07u
Jl54b47JXi+hIt1X3hZtGAaKuNt9BDo3KIgD4McehJFH2eT0vYmLHjWAL
+JoO4V0/+e9ZlgF4hzjpVkYaO5zik7WyWbvVzLcv4sT4A77CGq4/uo
+ZsJlGdBQ/qlSXDBUKBacHhmYBV1j5xZgxLPu2feEl1OGP/+27126/Lz0M/
JVeShDOw==</espPassword>
```

3. 外部アダプタ設定ファイルには `<espConnection>` セクションがあります。ここには、**esp_server** への接続に必要なパラメータが含まれています。

espHost と **espPort** の各値を指定します。クラスタの場合は、**espConnection** でクラスタの URI を指定します。

```
<!-- Streaming platform settings -->
 <esp>
   <espConnection>
    <espHost>localhost</espHost>
    <espPort>22000</espPort>
<!--    <espProjectUri>esp://localhost:19011/ws1/p1</
espProjectUri> -->
   </espConnection>
```

4. <espSecurity> セクションには、ユーザ名、パスワードなど、外部アダプタの認証を有効にする場合に必要なパラメータがあります。**espAuthType** には認証タイプを指定します。

認証タイプ	必要な値
Kerberos	user_password
LDAP	user_password
キーストア、キーストアのパスワード	server_rsa
なし	認証なしで接続

Kerberos 認証の値を使用する例：

```
<espAuthType>user_password</espAuthType>
```

5. 選択した認証タイプに基づき、その他の必須フィールドに値を指定します。

認証タイプにかかわらず、パスワードを暗号化する場合は、**espRSAKeyStore** と **espRSAKeyStorePassword** の値を定義します。

```
<!--    <espRSAKeyFile>/keyfilepath/espuser.private.der</
espRSAKeyFile>   -->
    <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
    <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
    <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
```

6. 必要に応じて、**espEncryptionAlgorithm** に指定した認証タイプを変更します。デフォルト値は RSA です。もう 1 つの選択肢は DSA です。
7. 設定ファイルを保存します。

RTView アダプタ・パスワードの暗号化

isEncrypted プロパティが true に設定されている場合は、RTView アダプタ・パスワードを暗号化します。

1. キーストアを作成します。作成方法の詳細については、「Java キーストアの生成」を参照してください。
2. RTV プロジェクト・フォルダに `keystore.jks` ファイルを入れます。

   サンプル・プロジェクトの場合は、example フォルダにキーストア・ファイルを入れます。
3. 指定されたプレーン・テキスト形式のパスワードに対する暗号化パスワードを生成するように、`%ESP_HOME%¥adapters` にある `encrypt.bat` スクリプトを変更します。
4. RTView Builder を使用して ESP 接続を削除します。
5. 新しい ESP 接続を作成します。
6. [Password] フィールドに、暗号化パスワードを指定します。
7. 接続を保存します。
8. [No] を選択して既存の `.ini` ファイルを置換します。

   [Yes] を選択すると、`%RTV_HOME%¥lib` にあるファイルが更新されます。通常、暗号化パスワードには適切にエスケープしてエンコードする必要がある = と ¥ の各文字が含まれているので、

注意： `.ini` ファイルを直接更新すると動作しなくなる可能性があります。この処理にはグラフィカル・ユーザ・インタフェースを使用することをおすすめします。

第 5 章　外部データベースへのアクセス

サーバは `service.xml` 設定ファイルに定義されているデータベース・サービスを使用して外部データベースにアクセスします。

サーバが外部データベースと通信するには、次のようにします。

- 適切な JDBC ドライバまたは ODBC ドライバを使用して、インストールされている希望の外部データベースの効果的な JDBC 接続または ODBC 接続を確立します。
- `service.xml` ファイルを適切な設定情報で変更します。

サーバは、次の外部データベースに接続するための JDBC ドライバの使用をサポートしています。

- Adaptive Server® Enterprise
- IBM DB2
- Oracle
- Kx Systems KDB+
- Microsoft SQL Server

Event Stream Processor サーバは、次の外部データベースに接続するための ODBC ドライバの使用をサポートしています。

- Adaptive Server Enterprise
- Sybase IQ
- SQL Anywhere®
- IBM DB2
- Oracle
- Microsoft SQL Server
- TimesTen
- MySQL 5.x
- PostgreSQL

サービス設定ファイル

サービス設定ファイル (`service.xml`) には、データベース・サービスの定義があります。これにはデータベース接続に必要なプロパティとパラメータがすべて含まれます。

これらのプロパティを使用してサービスのデータベース接続を作成します。データベース・アクセスを必要とするアダプタは、接続の作成対象のサービスを指定

することによってデータベース・マネージャからの接続を取得します。たとえば、JDBC を介して Adaptive Server Enterprise データベースに接続するサービスと、ODBC を介して SQL Server に接続するサービスを定義できます。したがって、ソース・ストリームにアタッチされるデータベース (DB) アダプタで、JDBC を介して Adaptive Service Enterprise データベースから入力データを取得するプロジェクトを作成する場合は、Adaptive Server Enterprise のサービス名をアダプタ設定の一部 (**service** プロパティ) として指定します。実行時にアダプタはサービス定義のプロパティに基づきデータベース・マネージャからの接続を取得し、その接続を介してクエリを実行します。

各 <Service> ブロックは <Service> ノードの 2 つの属性 (Name と Type) を使用して 1 つのサービス定義項目を表します。サービスが DB サービス・タイプであることを示すには、Type 属性を "DB" に設定します。<Service> ノードには複数の <Parameter> タグがあります。各タグは 1 つのプロパティまたは設定を表します。<Parameter> タグは Name 属性と実際のパラメータ値で構成されます。Event Stream Processor の DB ドライバはこの情報を解析し、その解析結果に従って接続を設定します。

サービス設定ファイルのサンプル

`service.xml` 設定ファイルのサンプルは `ESP_HOME/bin` にあります。このファイルを参考にして、カスタムのサービス設定ファイルを作成できます。

```
<?xml version="1.0"?>
<Services>
        <Service Name="SampleJDBCService" Type="DB">
                <Parameter
Name="DriverLibrary">esp_db_jdbc_sybase_lib</Parameter>
                <Parameter Name="Host">localhost</Parameter>
                <Parameter Name="Port">12345</Parameter>
                <Parameter Name="User">user</Parameter>
                <Parameter Name="Password">password</Parameter>
                <Parameter Name="Database">db</Parameter>
        </Service>
        <Service Name="SampleODBCService" Type="DB">
                <Parameter Name="DriverLibrary">esp_db_odbc_lib</
Parameter>
                <Parameter Name="DSN">dsn</Parameter>
                <Parameter Name="User">user</Parameter>
                <Parameter Name="Password">password</Parameter>
        </Service>
</Services>
```

外部データベースへの接続の設定

`service.xml` ファイルを使用して、データベース接続に必要な接続プロパティを格納するサービス定義を格納します。

Event Stream Processor サーバの接続先の外部データベースごとに別々のサービス定義を作成します。

カスタムの `service.xml` ファイルを作成する場合の基準として、`ESP_HOME/bin` にあるサービス設定ファイルのサンプルを使用できます。実行中のプロジェクトのファイルを使用するには、プロジェクトの実行環境であるノードのクラスタ設定ファイルで **services-file** パラメータを変更します。これで、プロジェクトは `service.xml` ファイルを見つけられます。たとえば、`<Property name="services-file">${ESP_HOME}/bin/service.xml</Property>` というようにします。

Sybase Event Stream Processor 内からこのタイプの接続を設定するには、次の手順に従います。

1. **Service Name** パラメータをデータベース・サービスのユニークなサービス名に設定します。この名前の特徴は、次のとおりです。
   - 大文字と小文字の区別があります。
   - ファイル名の先頭は文字です。
   - 文字、数字、下線、ドット、コロンのいずれかで構成される文字列を使用できます。

   このサービス名は、外部データベースにアクセスする、データベース・アダプタなどのコンポーネントに対して指定する値です。
2. Service パラメータの **Type** 属性を `DB` に設定します。
3. (オプション) **Description** パラメータにサービス項目の説明を追加します。
4. **DriverLibrary** パラメータを、接続先のデータベースの Event Stream Processor ライブラリに設定します。

   JDBC ドライバ：

データベース	DriverLibrary 値
Adaptive Server Enterprise	esp_db_jdbc_sybase_lib
Microsoft SQL Server	esp_db_jdbc_mssql_lib
IBM DB2	esp_db_jdbc_db2_lib

データベース	DriverLibrary 値
Oracle	esp_db_jdbc_oracle_lib
Kx Systems KDB+	esp_db_jdbc_kdb_lib

ODBC ドライバ：

データベース	DriverLibrary 値
すべてのデータベース	esp_db_odbc_lib または esp_db_odbc64_lib

注意： 32 ビットの ODBC ドライバ・マネージャを使用している場合は **DriverLibrary** パラメータを esp_db_odbc_lib に設定し、64 ビットのマネージャを使用している場合は esp_db_odbc64_lib に設定します。

5. **User** パラメータを、外部データベースと通信する場合に使用するユーザ名に設定します。
   この値は暗号化されません。`services.xml` へのアクセス権があれば、誰でもこのユーザ名を読み取れます。
6. **Password** パラメータをユーザ名のパスワードに設定します。
   このパスワードを暗号化するには、パスワード値の後に `encrypted="true"` 属性を追加し、**esp_cluster_admin** ユーティリティを使用して、暗号化されたテキストを生成します。
7. (ODBC ドライバのみ) **DSN** パラメータを、サービスが使用するデータ・ソース名に設定します。
   このデータ・ソースは事前に ODBC ドライバ・マネージャで設定しておいてください。
8. (Oracle と TimesTen の ODBC ドライバのみ) **WriteBigIntAsChar** パラメータを true に設定し、ODBC ドライバ・プラグインによって bigint 型のデータを char 型としてデータベースに挿入するように設定します。有効な値は true または false です。
   たとえば、Oracle の ODBC ドライバは SQL_C_SBIGINT/SQL_C_UBIGINT パラメータをサポートしないため、データベース・アウトプット・アダプタが `long` と `interval` の Event Stream Processor 型を `bigint` 型のカラムに書き込もうとするとエラーが発生します。この問題を回避するため、Oracle と TimesTen の ODBC ドライバまたはテーブルで bigint 型のカラムを使用する場合は、このパラメータを `true` (`<Parameter Name="WriteBigIntAsChar">true</Parameter>`) に設定します。
9. (JDBC ドライバのみ) 次のように設定します。

パラメータ	説明
Host	データベース・サーバのホスト名。
Port	データベース・サーバのポート番号。
Database	データベース名。このパラメータは Oracle と KDB の場合は必要ありません。

または

パラメータ	説明
ConnectString	(オプション) これは、ホスト、ポート、データベースに関する情報を含む JDBC スタイルの接続文字列です。このパラメータは、**Host**、**Port**、**Database** の 3 つの別々のパラメータに優先します。 ドライバと外部サーバ間のパスワード暗号化を有効にするには、**EncryptPassword** プロパティを接続文字列に追加し、true に設定します。例を示します。 `<Parameter Name="ConnectString>jdbc:sybase:Tds:localhost:5000/cep?ENCRYPT_PASSWORD=true</Parameter>`

重要： Oracle JDBC ドライバの場合は、**Database** パラメータを **Instance** パラメータに置き換えます。たとえば、`<Parameter Name="Instance">orcl</Parameter>` のように記述します。

参照：

- *第 4 章、「設定ファイルのパスワード暗号化」*(53 ページ)

付録 A メタデータ・ストリーム

メタデータ・ストリームは Event Stream Processor によって自動的に作成されます。これらのストリームに対しクエリを実行してサブスクライブすることで、現在実行中のプロジェクトに関する重要な正常性情報とパフォーマンス情報を取得します。

メタデータ・ストリームの特徴は、次のとおりです。

- 予約名が付けられます。他のオブジェクトはこれらの名前を使用できません。
- レコードは ESPMetadataStore という特殊なストアに格納されます。他のストリームはこのストアを使用できません。

次の 14 個のメタデータ・ストリームがあります。

- _esp_columns
- _esp_keycolumns
- _esp_subscriptions
- _esp_subscriptions_ext
- _esp_runupdates
- _esp_clockupdates
- _esp_clients
- _esp_clients_monitor
- _esp_config
- _esp_connectors
- _esp_streams
- _esp_tables
- _esp_streams_topology
- _esp_streams_monitor

注意： プロジェクト内のストリームは、メタデータ・ストリームを入力とすることはできません。

_esp_columns

すべてのストリームのすべてのカラムに関する情報があります。

カラム	タイプ	説明
usename	`string`	"user" としてハードコードされています。

カラム	タイプ	説明
relname	`string`	このローで記述されているカラムがあるストリームの名前。
attname	`string`	このローで記述されているカラムの名前。
attypid	`integer`	このカラムのタイプを表す内部の PostgreSQL 値。有効な値は次のとおりです。 • integer 型の場合 - 23 • long 型の場合 - 20 • money 型の場合 - 701 • float 型の場合 - 701 • date 型の場合 - 1114 • timestamp 型の場合 - 1114 • string 型の場合 - 1043
attnum	`integer`	0 を先頭とする、スキーマ内のこのカラムの位置。

_esp_keycolumns

すべてのストリームのプライマリ・キー・カラムに関する情報があります。ストリームにプライマリ・キーがある場合、そのキーを構成するカラムがこのストリームにリストされます。

カラム	タイプ	説明
table	`string`	このローで記述されているカラムが属するストリームの名前。
field	`string`	このローで記述されているカラムの名前。
type	`integer`	このカラムのタイプを表す内部の PostgreSQL 値。有効な値は次のとおりです。 • integer 型の場合 - 23 • long 型の場合 - 20 • money 型の場合 - 701 • float 型の場合 - 701 • date 型の場合 - 1114 • timestamp 型の場合 - 1114 • string 型の場合 - 1043

_esp_subscriptions

現在アクティブなすべてのサブスクリプションに関する情報があります。すべてのサブスクリプションを追跡します。削除された接続は、サブスクライブ先のすべてのものからサブスクライブ解除されたと見なされます。

カラム	タイプ	説明
stream_handle	long	サーバがサブスクライブ先のストリームに割り当てるハンドル。
conn_handle	long	サーバがサブスクライブ先の接続に割り当てるハンドル。

_esp_subscriptions_ext

現在アクティブなすべてのサブスクリプションに関する情報があります。

新しいサブスクリションが追加されたり既存のストリームが削除されたりした場合に、このストリームの更新で遅延が生じることがあります。

カラム	タイプ	説明
stream_handle	long	サーバがサブスクライブするストリームのハンドル。
conn_handle	long	サーバがサブスクライブする接続のハンドル。
stream_name	string	ストリームの名前。
stream_user	string	ストリームの所有者のユーザ名。
subscriber_user	string	サブスクリプションを所有しているユーザ・アカウントのログイン名。
ip	string	クライアント・マシンの IP アドレス。
host	string	クライアント・マシンの記号ホスト名 (使用可能な場合)。使用できない場合は、クライアント・マシンの IP アドレスになります。
port	integer	このサブスクリプションが属している接続の開始元の TCP ポート番号。
login_time	timestamp	サブスクリプションが属している接続をサーバが受け入れた時間。

_esp_runupdates

デバッグ時に変更を通知します。サーバはプロジェクトがトレース・モードの場合のみ通知を送信します。

カラム	タイプ	説明
key	`string`	更新のタイプ。次の表を参照してください。
value	`integer`	key カラムで指定された更新に関連付けられた番号。
stream	`string`	更新によって個々のストリームに関するイベントを通知する場合はストリームの名前。それ以外の場合は NULL。
info	`string`	更新に関連付けられた追加情報。このフォーマットは更新のタイプによって異なります。

次の表に、サーバがデバッグ・コマンドとして送信する更新のタイプを示します。この表の「値」列と「ストリーム」列は _esp_runupdates ストリームの「カラム」下の「value」行と「stream」行に対応します。

キー	値	ストリーム	説明
TRACE	0 または 1	なし	有効 (1) または無効 (0)。
RUN	0 または 1	なし	プラットフォームが一時停止している (0) か、実行中 (1) です。
STEP	<count>	なし	プロジェクトは手動または自動のいずれかでシングル・ステップでした。値には実行されたステップの数が入ります。停止されたストリームに関する詳細は提供されません。
BREAK	<bp-id>	<stream-name>	ID <bp-id> のブレークポイントがストリーム <stream-name> でトリガーされました。これらの更新は、対応する更新 "RUN 0" の前または後のどちらかに入る可能性があります。
NO BREAK	<bp-id>	<stream-name>	ストリーム <stream-name> 上の ID <bp-id> のブレークポイントは、leftToTrigger の数は減少しましたが、まだトリガーされていません。

キー	値	ストリーム	説明
EXCEPTION	なし	<stream-name>	ストリーム <stream-name> で例外が発生しました。これらの更新は、対応する更新 "RUN 0" の前または後のどちらかに入る可能性があります。
REQUESTEXIT	なし	なし	受信したプロジェクトをシャットダウンする要求。
EXIT	なし	なし	すべてのユーザ・ストリームが終了したので、プロジェクトはシャットダウンしようとしています。

_esp_clockupdates

プロジェクトの論理クロックの変更を通知します。

カラム	タイプ	説明
キー	`string`	更新のタイプ。現時点では "CLOCK" です。
Rate	`float`	実際の時間を基準とする論理クロックのレート。
Time	`float`	UNIX エポックから現在の時刻までの経過秒数。
Real	`integer`	リアルタイム・フラグ。論理クロックがシステム時間と一致する場合は 1、一致しない場合は 0。
stop_depth	`integer`	時間の流れを再開するために clock resume コマンドを呼び出す回数。クロックが実行中の場合、stop_depth は 0 です。
max_sleep	`integer`	すべてのスリーパが物理的なクロック・レートまたはクロック時間の変化を検出することを保証する時間 (実際のミリ秒単位)。

_esp_clients

現在アクティブなすべてのゲートウェイ・クライアント接続に関する情報があります。

カラム	タイプ	説明
Handle	`long`	接続のユニークな整数 ID。

カラム	タイプ	説明
user_name	`string`	接続にログインするユーザ名。ユーザが認証されると表示されます。
IP	`string`	クライアント・マシンのアドレス。
host	`string`	クライアント・マシンの記号ホスト名 (使用可能な場合)。使用できない場合、host はクライアント・マシンの IP アドレスです。
port	`integer`	接続の開始元の TCP ポート番号。
login_time	`timestamp`	サーバが接続を受け入れた (が認証していない) 時刻 (GMT)。
conn_tag	`string`	ユーザ設定の記号接続タグ名。ユーザが設定していない場合、conn_tag は NULL です。

_esp_clients_monitor

現在アクティブなすべてのゲートウェイ・クライアント接続のパフォーマンスに関する情報があります。_esp_clients ストリームのデータのコピーが含まれます。データをモニタできるのは、プロジェクト設定 (CCR) ファイルに **time-granularity** オプションが設定されている場合のみです。

カラム	タイプ	説明
Handle	`long`	接続のユニークな整数 ID。
user_name	`string`	接続にログインするユーザ名。ユーザが認証されると表示されます。
IP	`string`	文字列としての、クライアント・マシンのアドレス。
host	`string`	クライアント・マシンの記号ホスト名 (使用可能な場合)。使用できない場合、host はクライアント・マシンの IP アドレスです。
port	`integer`	接続の開始元の TCP ポート番号。
login_time	`timestamp`	サーバが接続を受け入れた (が認証していない) 時刻 (GMT)。
conn_tag	`string`	ユーザ設定の記号接続タグ名。ユーザが設定していない場合、conn_tag は NULL です。

カラム	タイプ	説明
cpu_pct	float	前回の更新以降のクライアント・ゲートウェイ・スレッドの CPU 使用率。
last_update	date	現時点の更新時刻。
subscribed	integer	ストリームのサブスクリプションの状況。サブスクライブされている場合は 1、それ以外の場合は 0 です。
sub_trans_per_sec	float	前回の更新以降にクライアントが受信した 1 秒あたりのトランザクション数で表すクライアントのパフォーマンス。
sub_rows_per_sec	float	前回の更新以降にクライアントが受信した 1 秒あたりのデータ・ロー数で表すクライアントのパフォーマンス。
sub_inc_trans	long	前回の更新以降にクライアントが受信したトランザクション、エンベロープ、またはメッセージの数。
sub_inc_rows	long	前回の更新以降にクライアントが受信したデータ・ローの数。
sub_total_trans	long	クライアントが受信したトランザクション、エンベロープ、またはメッセージの総数。
sub_total_rows	long	クライアントが受信したデータ・ローの総数。
sub_dropped_rows	long	クライアントによって速やかに読み取られなかったためにゲートウェイで削除されたデータ・ローの総数。損失を伴うサブスクリプション用です。
sub_accum_size	integer	アキュムレータで収集されて次のパルスで送信される現在のローの数。パルス・サブスクリプション用です。
sub_queue	integer	クライアントに転送するためにキューに入れられているローの数。
sub_queue_fill_pct	float	キュー・サイズの上限を基準とする現在の sub_queue の比率。sub_queue_fill_pct が 100 パーセントに達すると、これ以降、このクライアントへデータをポストする試みはブロックされ、フロー制御はポスト元に戻されます。

カラム	タイプ	説明
sub_work_queue	integer	適切なキューからソケット・バッファに転送される、クライアントへの転送対象のロー数。ローはエンベロープを基準に再グループ化できます。
pub_trans_per_sec	float	前回の更新以降にクライアントから送信された 1 秒あたりのトランザクション数で表すクライアントのパフォーマンス。エンベロープとサービス・メッセージがトランザクションとしてカウントされます。
pub_rows_per_sec	float	前回の更新以降にクライアントから送信された 1 秒あたりのデータ・ロー数で表すクライアントのパフォーマンス。
pub_inc_trans	long	前回の更新以降にクライアントから送信されたトランザクション、エンベロープ、またはメッセージの数。
pub_inc_rows	long	前回の更新以降にクライアントから送信されたデータ・ローの数。
pub_total_trans	long	クライアントから送信されたトランザクション、エンベロープ、またはメッセージの総数。
pub_total_rows	long	クライアントから送信されたデータ・ローの総数。
pub_stream_id	long	クライアントが現在データをパブリッシュしようとしているストリームの数値 ID。通常、pub_stream_id は -1 です。これは、クライアントは現在データのパブリッシュを試みていないことを示します。

_esp_config

プロジェクトの現在の CCX があります。

カラム	タイプ	説明
key	string	"XML" としてハードコードされています。
value	string	現在の CCX のテキスト。

_esp_connectors

プロジェクトに定義されているすべてのインプロセス・アダプタに関する情報があります。

カラム	タイプ	説明
name	`string`	プロジェクトに定義されている、アダプタの名前。
stream	`string`	アダプタが定義されているストリームの名前。
type	`string`	**ATTACH ADAPTER** 文に定義されているアダプタのタイプ。
input	`integer`	値は、InConnection の場合は 1、OutConnection の場合は 0 です。
ingroup	`string`	このコネクタが属する StartUp グループ。
state	`string`	アダプタの状態。次のいずれかです。 • READY – 起動する準備が整っています。 • INITIAL – 起動と初期化を実行しています。 • CONTINUOUS – リアルタイムのデータを受信し続けています。 • IDLE – 現時点ではデータを受信していませんが、データ・ソースまたはリンクへの再接続を試みています。 • DONE – 入力データも出力データも残っていません。アダプタは終了しようとしています。 • DEAD – アダプタ・スレッドは終了しました。再起動が明示的に要求されるまでアダプタはこの状態を維持します。
total_rows	`long`	入力データで認識されたデータ・レコードの総数。
good_rows	`long`	正常に処理されたデータ・レコードの数。
bad_rows	`long`	エラーが発生したデータ・レコードの数。total_rows、good_rows、bad_rows の各フィールドは、オーバヘッドを削減するために数秒に 1 回更新されます。

_esp_streams

すべてのストリーム、デルタ・ストリーム、ウィンドウに関する情報があります。

カラム	タイプ	説明
user_name	string	"user" としてハードコードされています。
stream_name	string	このローで記述されるストリームの名前。
handle	long	ストリームの数値 ID。
type	string	ストリームのタイプ。"stream"、"deltastream"、"window"、"metadata" があります。
visibility	string	ストリームの可視性。"input"、"output"、"local"、"intermediate" があります。
target	string	visibility の値が "intermediate" であるストリームのターゲット・ストリームの名前。visibility の値が "intermediate" 以外のストリームでは、target は stream_name カラムと同じです。

_esp_tables

_ESP_Streams ストリームにある情報のコピーを含む内部ストリーム。

注意： このストリームは使用しないでください。代わりに _esp_streams ストリームを使用してください。

カラム	タイプ	説明
relname	string	このローで記述されるストリームの名前。
username	string	"user" としてハードコードされています。
remarks	string	ストリームの数値 ID。ASCII 文字列としての 10 進数。

_esp_streams_topology

直接接続される 2 つのストリームの名前があります。

カラム	タイプ	説明
src_stream	string	接続元ストリームの名前。
dst_stream	string	接続先ストリームの名前。

_esp_streams_monitor

ストリームのパフォーマンスに関する情報があります。_ESP_Streams ストリームのデータのコピーが含まれます。データをモニタできるのは、プロジェクト設定 (CCR) ファイルに **time-granularity** オプションが設定されている場合のみです。

カラム	タイプ	説明
stream	string	ストリームの名前。
target	string	ターゲット要素の名前。
cpu_pct	float	前回の更新以降のストリーム・スレッドの CPU 使用率。
trans_per_sec	float	前回の更新以降のストリームのパフォーマンス (1 秒あたりのトランザクション数)。
rows_per_sec	float	前回の更新以降のストリームのパフォーマンス (1 秒あたりのロー数)。
inc_trans	long	前回の更新以降にサーバで処理されたトランザクションの数。
inc_rows	long	前回の更新以降にサーバで処理されたローの数。
queue	integer	現在の入力キューのサイズ。
store_rows	long	ストリームのストア内の現在のレコード数。
last_update	date	現時点の更新時刻。
sequence	long	現時点の更新のシーケンス番号。

カラム	タイプ	説明
posting_to_client	long	ストリームが現在データをパブリッシュしようとしているクライアント接続の数値 ID。通常、posting_to_client は -1 です。これは、ストリームは現在データのパブリッシュを試みていないことを示します。

付録 B タイム・ゾーン

タイム・ゾーンは、一般的にローカル時間と呼ばれる同じ標準時間を使用する地域を表します。

ほとんどの場合、隣接したタイム・ゾーンとは、1 時間の時差があります。慣例によって、すべてのタイム・ゾーンは、ローカル時間を GMT/UTC からのオフセットとして計算します。グリニッジ標準時 (GMT) は歴史上の用語で、元々、英国の王立グリニッジ天文台での平均太陽時を意味していました。GMT は、原子時計を基準にする協定世界時 (UTC) に置き換えられています。すべての Sybase Event Stream Processor の使用目的で、GMT と UTC は等価です。政治的または地理的な実用性によって、タイム・ゾーン特性が時間経過と共に変化することがあります。たとえば、夏時間の開始日と終了日は年によって異なることがあり、新しい国が誕生すると新しいタイム・ゾーンが導入されることがあります。

内部的には、Event Stream Processor は日付／時刻型の情報を、データ型に応じて 1970 年 1 月 1 日午前 0 時 UTC からの秒数、ミリ秒数、またはマイクロ秒数として常に格納します。タイム・ゾーン指示子が使用されていない場合は、UTC 時間が適用されます。

夏時間
夏時間は、そのタイム・ゾーンで夏時間が使用されており、指定されたタイムスタンプが夏時間の期間内にあるときに考慮されます。夏時間の開始日と終了日は、C++ ライブラリに格納されています。

特定のタイム・ゾーンが指定されており、そのタイム・ゾーンで夏時間が使用されている場合、Event Stream Processor はこれらの日付を考慮して日付／時刻のデータ型を調整します。たとえば、次の例では、米国太平洋標準時 (PST) は夏時間の期間内にあるので、タイムスタンプが調整されます。

```
to_timestamp('2002-06-18 13:52:00.123456 PST','YYYY-MM-DD
HH24:MI:SS.ff TZD')
```

標準時から夏時間への移行とその逆の移行
夏時間への移行時と夏時間からの移行時には、特定の時間がありません。たとえば、米国の場合、標準時から夏時間への移行時に、時計は 01:59 から 03:00 に変化し、02:00 がありません。逆に、夏時間から標準時への移行時には、1 晩に 01:00 ～ 01:59 が 2 回発生します。これは、夏時間の終了時に時間が 2:00 から 1:00 に変化するためです。

ただし、これらの未定義の時間に受信データの入力が発生する可能性があり、エンジンは何らかの方法で対処する必要があります。夏時間への移行時に、Event

Stream Processor は 02:59 PST を 01:59 PST として解釈します。標準時に戻るときには、Event Stream Processor は 02:00 PDT を 01:00 PST として解釈します。

索引

さ

す

せ

た

て

と

は

ふ

ほ

め

ゆ

ろ