



Red Hat JBoss Enterprise Application Platform 7.3

JBoss EAP でのトランザクションの管理

管理者が Red Hat JBoss Enterprise Application Platform トランザクションをトラブルシューティングするための手順と情報。

Red Hat JBoss Enterprise Application Platform 7.3 JBoss EAP でのトランザクションの管理

管理者が Red Hat JBoss Enterprise Application Platform トランザクションをトラブルシューティングするための手順と情報。

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2023 | You need to change the HOLDER entity in the en-US/Managing_Transactions_on_JBoss_EAP.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書には、管理者が JBoss EAP でトランザクションをトラブルシューティングするための情報が記載されています。

目次

第1章 JBOSS EAP のトランザクション	4
1.1. TRANSACTION サブシステム	4
1.2. トランザクションのプロパティ	4
1.3. トランザクションのコンポーネント	4
1.4. トランザクション管理の原則	5
1.4.1. XA トランザクションと非 XA トランザクション	5
第2章 トランザクションの設定	6
2.1. 一意なノード識別子	6
2.1.1. 一意なノード識別子の重要性	6
2.2. トランザクションマネージャーの設定	6
管理コンソールを使用したトランザクションマネージャーの設定	6
管理 CLI を使用したトランザクションマネージャーの設定	6
2.3. システムプロパティを使用したトランザクションマネージャーの設定	7
2.4. JAKARTA TRANSACTIONS を使用するようにデータソースを設定	8
前提条件	8
Jakarta Transactions を使用するようにデータソースを設定	8
2.5. JTS の ORB の設定	8
管理 CLI を使用した ORB の設定	9
セキュリティインターセプターの有効化	9
IIOP サブシステムでのトランザクションの有効化	9
Transactions サブシステムでの JTS の有効化	9
管理コンソールを使用した ORB の設定	9
第3章 トランザクションの管理	10
3.1. トランザクションの閲覧	10
ログストアの更新	10
準備済みトランザクションすべての表示	10
3.2. トランザクションの管理	10
トランザクションの属性の表示	10
トランザクション参加者の詳細の表示	10
トランザクション参加者の削除	11
トランザクション参加者のリカバリー	12
トランザクション参加者の状態の更新	12
3.3. トランザクション統計の表示	12
3.4. トランザクションオブジェクトストアの設定	13
JDBC データソースをトランザクションオブジェクトストアとして使用	14
トランザクション JDBC ストア属性	14
ActiveMQ ジャーナルオブジェクトストアの使用	15
第4章 トランザクションの監視	16
4.1. TRANSACTIONS サブシステムのロギング設定	16
管理コンソールを使用したトランザクションロガーの設定	16
管理 CLI を使用したトランザクションロガーの設定	16
4.1.1. TRACE ログレベルの有効化	16
4.1.2. トランザクションブリッジロガーの有効化	17
4.1.3. トランザクションログメッセージ	18
4.1.4. トランザクションログファイルのデコード	19
4.1.4.1. トランザクションの XID/UID の特定	19
4.1.4.2. トランザクション状態とリソースの特定	19
TransactionStatusConnectionManager	19
TransactionStatusManager	20

4.1.4.3. トランザクション履歴の表示	20
第5章 トランザクションマネージャー例外の処理	23
5.1. タイムアウトしたトランザクションのデバッグ	23
5.2. 新しい JBOSS EAP サーバーへのログの移行	24
5.2.1. ファイルベースのログストレージの移行	24
5.2.2. JDBC ストアベースのログストレージの移行	24
5.3. JBOSS EAP での XTS の有効化	24
5.4. 期限切れトランザクションの消去	25
5.5. ヒューリスティックな結果のリカバリー	26
5.5.1. ヒューリスティックな結果を判断するためのガイドライン	28
問題の検出	28
手作業によるトランザクションのコミットまたはロールバック	28
HEURISTIC_HAZARD 例外のリカバリー	28
HEURISTIC_ROLLBACK および HEURISTIC_COMMIT 例外のリカバリー	29
手動調整に失敗した場合の追加手順	29

第1章 JBOSS EAP のトランザクション

トランザクションは、すべてが成功または失敗しなければならない2つ以上の操作で設定されます。成功するとコミットされ、失敗するとロールバックされます。ロールバックでは、トランザクションのコミットが行われる前に各メンバーの状態が元に戻されます。

1.1. TRANSACTION サブシステム

transactions サブシステムは、タイムアウト値、トランザクションロギング、統計の収集、Jakarta Transactions を使用するかどうかなど、トランザクションマネージャー (TM) オプションの設定を可能にします。**transactions** サブシステムは、以下の4つの主要素で設定されます。

コア環境

コア環境には、管理されているリソースの代わりに JBoss EAP サーバーによるトランザクション境界の制御を可能にする TM インターフェイスが含まれています。トランザクションコーディネーターはトランザクションオブジェクトとの対話と、トランザクションに参加するリソースを管理します。

リカバリー環境

JBoss EAP トランザクションサービスのリカバリー環境は、システムがトランザクションの影響を受けるすべてのリソースに一貫してトランザクションの結果を確実に適用するようにします。アプリケーションのプロセスやそれをホストするマシンがクラッシュしたり、ネットワークの接続が切断されても、この操作は継続されます。

コーディネーター環境

コーディネーター環境は、デフォルトのタイムアウトやロギングの統計など、トランザクションのカスタムプロパティを定義します。

オブジェクトストア

JBoss EAP トランザクションサービスは、障害をリカバリーする目的で、オブジェクトストアを使用し、トランザクションの結果を永続的に記録します。リカバリーマネージャーは、リカバリーが必要となる可能性のあるトランザクションやリソースに対して、オブジェクトストアやその他の情報の場所をスキャンします。

1.2. トランザクションのプロパティ

適切に設計されたトランザクションの一般的な標準は ACID (アトミック性、一貫性、独立性、および永続性: atomic, consistent, isolated, and durable) です。

アトミック性

トランザクションのすべてのメンバーは、トランザクションのコミットまたはロールバックに関して同じ決定を行う必要があります。

一貫性

トランザクションは一貫した結果を生成し、アプリケーション固有のインバリアントを保持します。

独立性

トランザクションスコープ外のプロセスによるデータの変更を防ぐため、変更前に操作するデータをロックする必要があります。

永続性

致命的な障害が発生した場合を除き、コミットされたトランザクションの影響は失われません。

1.3. トランザクションのコンポーネント

トランザクションコーディネーター

コーディネーターはトランザクションの結果を左右します。これは、クライアントによって呼び出される web サービスが一貫した結果になるようにします。

トランザクションコンテキスト

トランザクションコンテキストは伝播されるトランザクションに関する情報で、トランザクションが複数のサービスにまたがることが可能になります。

トランザクション参加者

参加者は、参加者モデルを使用してトランザクションに登録されるサービスです。

トランザクションサービス

トランザクションサービスは基盤のトランザクションプロトコルのモデルをキャプチャーし、そのモデルに応じてトランザクションに関連する参加者と調整します。

トランザクション API

トランザクション API は、トランザクション境界と参加者登録のインターフェイスを提供します。

1.4. トランザクション管理の原則

1.4.1. XA トランザクションと非 XA トランザクション

非 XA トランザクションには1つのリソースのみが関与します。非 XA トランザクションにはトランザクションコーディネーターがなく、1つのリソースがすべてのトランザクションの作業を行います。ローカルトランザクションと呼ばれることもあります。

XA トランザクションには複数のリソースが関与します。XA トランザクションにはコーディネートを行うトランザクションマネージャーがあり、1つのトランザクションに1つ以上のデータベースまたは Jakarta Messaging などのその他のリソースがすべて参加します。グローバルトランザクションとも呼ばれます。

第2章 トランザクションの設定

2.1. 一意なノード識別子

一意なノード識別子は、特定のノード識別子のみと一致するトランザクションおよびトランザクション状態のリカバリーを可能にします。ノード識別子は **com.arjuna.ats.arjuna.nodeIdentifier** プロパティを使用して設定します。

2.1.1. 一意なノード識別子の重要性

XA リカバリーの実行時、JBoss EAP トランザクションによるリカバリーが可能な **Xid** タイプを設定する必要があります。各 **Xid** には一意なノード識別子がエンコードされており、JBoss EAP は特定のノード識別子と一致するトランザクションおよびトランザクション状態のみをリカバリーします。

ノード識別子を設定するには、**JTAEnvironmentBean.xaRecoveryNodes** プロパティを使用します。これには複数の値をリストで含めることができます。



警告

アスタリスク ("*") の値を指定すると、ノード識別子に関係なく、すべてのトランザクションのリカバリー (および可能性としてロールバック) が強制されます。そのため、注意して使用する必要があります。

com.arjuna.ats.jta.xaRecoveryNode プロパティの値は英数字である必要があります。また、**com.arjuna.ats.arjuna.nodeIdentifier** プロパティの値と一致する必要があります。

2.2. トランザクションマネージャーの設定

トランザクションマネージャーは、Web ベースの管理コンソールまたはコマンドライン管理 CLI を使用して設定できます。

管理コンソールを使用したトランザクションマネージャーの設定

以下の手順は、Web ベースの管理コンソールを使用してトランザクションマネージャーを設定する方法を示しています。

1. 画面上部の **Configuration** タブを選択します。
2. JBoss EAP を管理対象ドメインとして実行している場合は、変更する任意のプロファイルを選択します。
3. **Subsystem** リストから、**Transaction** を選択し、**表示** をクリックします。
4. 該当する設定のタブを選択します。たとえば、リカバリーオプションの場合は **Recovery** を選択します。
5. **編集** をクリックして必要な変更を行い、**保存** をクリックして変更を保存します。

管理 CLI を使用したトランザクションマネージャーの設定

管理 CLI で一連のコマンドを使用してトランザクションマネージャーを設定できます。これらのコマン

ドはすべて **/subsystem=transactions** (スタンドアロンサーバー向け) または **/profile=default/subsystem=transactions/** (管理対象ドメインの **default** プロファイル向け) で始まります。

トランザクションマネージャーのすべての設定オプションの詳細なリストについては、[トランザクションマネージャーの設定オプション](#) を参照してください。

2.3. システムプロパティを使用したトランザクションマネージャーの設定

管理コンソール、管理 CLI、またはシステムプロパティのいずれかを使用すると多くのトランザクションマネージャーオプションを設定できますが、以下のオプションはシステムプロパティを使用しのみ設定可能です。ただし、以下のオプションはシステムプロパティのみを使用して設定できます。管理 CLI または管理コンソールを使用して設定することはできません。

プロパティ名	説明
RecoveryEnvironmentBean.periodicRecoveryPeriod	リカバリーを試行する間隔 (秒単位)。 <ul style="list-style-type: none"> ● 正の整数である必要があります。 ● デフォルトは 120 秒 (2 分) です。
RecoveryEnvironmentBean.recoveryBackoffPeriod	最初と 2 度目のリカバリーパスの間隔 (秒単位)。 <ul style="list-style-type: none"> ● 正の整数である必要があります。 ● デフォルトは 10 秒です。
RecoveryEnvironmentBean.periodicRecoveryInitilizationOffset	最初のリカバリーパスより前の間隔 (秒単位)。 <ul style="list-style-type: none"> ● 0 または正の整数である必要があります。 ● デフォルトは 0 秒です。
RecoveryEnvironmentBean.expiryScanInterval	期限切れスキャンの間隔 (時間単位)。 <ul style="list-style-type: none"> ● 整数値である必要があります。 ● 0 はスキャンを無効にします。 ● 負の値は、最初の実行を遅延します。 ● デフォルトは 12 時間です。

以下は、**standalone.xml** サーバー設定ファイルでこれらのシステムプロパティを設定する方法の例になります。

```
<system-properties>
  <property name="RecoveryEnvironmentBean.periodicRecoveryPeriod" value="180"/>
  <property name="RecoveryEnvironmentBean.recoveryBackoffPeriod" value="20"/>
</system-properties>
```

```
<property name="RecoveryEnvironmentBean.periodicRecoveryInitializationOffset" value="5"/>
<property name="RecoveryEnvironmentBean.expiryScanInterval" value="24"/>
</system-properties>
```

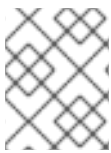
詳細は、[設定ガイド](#) のシステムプロパティを参照してください。

2.4. JAKARTA TRANSACTIONS を使用するようにデータソースを設定

このタスクは、データソースで Jakarta Transactions を有効にする方法を示しています。

前提条件

- データベースは Jakarta Transactions をサポートする必要があります。詳細は、データベースのベンダーのドキュメントを参照してください。
- [非 XA データソース](#) を作成します。手順は、[設定ガイド](#) を参照してください。



注記

XA データソースは、[設定ガイド](#) で説明されているように、デフォルトで機能する Jakarta Transactions です。

Jakarta Transactions を使用するようにデータソースを設定

- 以下の管理 CLI コマンドを使用して **jta** 属性を **true** に設定します。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:write-
attribute(name=jta,value=true)
```



注記

管理対象ドメインでは、このコマンドの前に **/profile=PROFILE_NAME** を付けます。

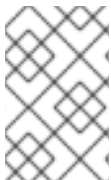
- 変更を反映するためにサーバーをリロードします。

```
reload
```

これで、データソースが Jakarta Transactions を使用するように設定されます。

2.5. JTS の ORB の設定

JBoss EAP のデフォルトインストールでは、トランザクションの ORB (Object Request Broker: オブジェクトリクエストブローカー) のサポートが無効になっています。管理 CLI または管理コンソールを使用して **iiop-openjdk** サブシステムの ORB を設定できます。



注記

iiop-openjdk サブシステムを利用できるのは、管理対象ドメインで **full** または **full-ha** プロファイルを使用している場合や、スタンドアロンサーバーの **standalone-full.xml** または **standalone-full-ha.xml** 設定ファイルを使用している場合です。

iiop-openjdk サブシステムの使用可能な設定オプションについては、[設定ガイド](#) の **IIOP サブシステム** の属性を参照してください。

管理 CLI を使用した ORB の設定

管理 CLI を使用して ORB を設定できます。これは、JTS と使用するために行う ORB の最低限の設定です。

以下の管理対象ドメインの管理 CLI コマンドは、**full** プロファイルを使用して設定できます。必要な場合は設定に応じてプロファイルを変更してください。スタンドアロンサーバーを使用している場合は、コマンドの **/profile=full** 部分を省略してください。

セキュリティーインターセプターの有効化

値を **identity** に設定し、**security** 属性を有効にします。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=security,value=identity)
```

IIOP サブシステムでのトランザクションの有効化

JTS に対して ORB を有効にするには、**transactions** 属性の値をデフォルトの **spec** ではなく **full** に設定します。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=transactions, value=full)
```

Transactions サブシステムでの JTS の有効化

```
/profile=full/subsystem=transactions:write-attribute(name=jts,value=true)
```



注記

JTS をアクティベートするにはリロードでは不十分なため、サーバーを再起動する必要があります。

管理コンソールを使用した ORB の設定

1. 管理コンソールの上部で、**Configuration** タブを選択します。管理対象ドメインを使用する場合は、変更するプロファイルを選択する必要があります。
2. **Subsystems** → **IIOP (OpenJDK)** と選択し、**表示** をクリックします。
3. **編集** をクリックし、必要に応じて属性を変更します。
4. **保存** をクリックして変更を保存します。

第3章 トランザクションの管理

3.1. トランザクションの閲覧

管理 CLI では、トランザクションレコードを参照および操作する機能がサポートされます。この機能は、TM と JBoss EAP の管理 API 間の対話によって提供されます。

トランザクションマネージャーは、待機中の各トランザクションとトランザクションに関連する参加者に関する情報を、オブジェクトストアと呼ばれる永続ストレージに格納します。管理 API は、オブジェクトストアを **log-store** と呼ばれるリソースとして公開します。**probe** 操作はトランザクションログを読み取り、各レコードに対してノードパスを作成します。**probe** コマンドは、**log-store** を更新する必要があるときに、いつでも手動で呼び出すことができます。トランザクションログが即座に表示され非表示になるのは、正常な挙動です。

ログストアの更新

以下のコマンドは、管理対象ドメインでプロファイル **default** を使用するサーバーグループに対してログストアを更新します。スタンドアロンサーバーの場合は、コマンドから **profile=default** を削除します。

```
/profile=default/subsystem=transactions/log-store=log-store:probe
```

準備済みトランザクションすべての表示

準備済みトランザクションをすべて表示するには、最初に [ログストアを更新](#) し、ファイルシステムの **ls** コマンドに類似した機能を持つ次のコマンドを実行します。

```
ls /profile=default/subsystem=transactions/log-store=log-store/transactions
```

または

```
/host=master/server=server-one/subsystem=transactions/log-store=log-store:read-children-names(child-type=transactions)
```

各トランザクションが一意的識別子とともに表示されます。個々の操作は、各トランザクションに対して実行できます。詳細は、[トランザクションの管理](#) を参照してください。

3.2. トランザクションの管理

トランザクションの属性の表示

Java Naming and Directory Interface 名、EIS 製品名およびバージョン、状態などのトランザクションに関する情報を表示するには、**read-resource** 操作を使用します。

```
/profile=default/subsystem=transactions/log-store=log-store/transactions=0\ffff7f000001\:-b66efc2\4f9e6f8f\9:read-resource
```

トランザクション参加者の詳細の表示

各トランザクションログには、**participants** (参加者) と呼ばれる子要素が含まれます。トランザクションの参加者の詳細を確認するには、この要素に **read-resource** 操作を使用します。参加者は、Java Naming および Directory Interface 名によって識別されます。

```
/profile=default/subsystem=transactions/log-store=log-store/transactions=0\ffff7f000001\:-b66efc2\4f9e6f8f\9/participants=java\:\JmsXA:read-resource
```

結果は以下のようになります。

```
{
  "outcome" => "success",
  "result" => {
    "eis-product-name" => "ActiveMQ",
    "eis-product-version" => "2.0",
    "jndi-name" => "java:/JmsXA",
    "status" => "HEURISTIC",
    "type" => "/StateManager/AbstractRecord/XAResourceRecord"
  }
}
```

ここで示された結果は **HEURISTIC** 状態であり、リカバリーが可能です。詳細は、[トランザクション参加者のリカバリー](#) を参照してください。

特別な場合では、ログに対応するトランザクションレコードがないオーファンレコード (XAResourceRecords) をオブジェクトストアに作成できます。たとえば、準備済みの XA リソースが TM の記録前にクラッシュし、ドメイン管理 API はアクセス不可能である場合などです。このようなレコードにアクセスするには、管理オプション **expose-all-logs** を **true** に設定する必要があります。このオプションは管理モデルには保存されず、サーバーが再起動されると **false** に戻ります。

```
/profile=default/subsystem=transactions/log-store=log-store:write-attribute(name=expose-all-logs,
value=true)
```

代わりに以下のコマンドを実行すると、トランザクション参加者 ID が集約され表示されます。

```
/host=master/server=server-one/subsystem=transactions/log-store=log-
store/transactions=0\:\ffff7f000001\:-b66efc2\4f9e6f8f\9:read-children-names(child-type=participants)
```

トランザクション参加者の削除

各トランザクションログは、トランザクションを表すトランザクションログを削除する **delete** 操作をサポートします。

```
/profile=default/subsystem=transactions/log-store=log-store/transactions=0\:\ffff7f000001\:-
b66efc2\4f9e6f8f\9:delete
```

これにより、トランザクションのすべての参加者も削除されます。



警告

通常、参加者のログ管理はリカバリーシステムまたはそれを所有するトランザクションログに任せますが、**delete** 操作はその操作を安全に実行できる場合に使用できます。XA リソースがヒューリスティックに完了すると、**forget** の呼び出しが発生し、XA リソースベンダーのログは適切に消去されます。この **forget** 呼び出しに失敗しても、デフォルトでは **delete** 操作は正常に行われます。この挙動をオーバーライドするには、**ObjectStoreEnvironmentBean.ignoreMBeanHeuristics** システムプロパティを **false** に設定します。

トランザクション参加者のリカバリー

トランザクションの各参加者は、**recover** 操作を使用したリカバリーをサポートします。

```
/profile=default/subsystem=transactions/log-store=log-store/transactions=0:ffff7f000001:-
b66efc2\4f9e6f8f\9/participants=2:recover
```

トランザクション参加者の状態が **HEURISTIC** である場合、**recover** 操作は状態を **PREPARE** に切り替え、周期リカバリープロセスにコミットを再実行するよう要求します。

コミットに成功すると、参加者はトランザクションログから削除されます。これを検証するには、**log-store** で **probe** 操作を実行し、参加者がリストされていないことを確認します。最後の参加者が削除されると、トランザクションも削除されます。

トランザクション参加者の状態の更新

トランザクションをリカバリーする必要がある場合は、リカバリーを試行する前に **refresh** 操作を使用して、トランザクションのリカバリーが必要であることを確認できます。

```
/profile=default/subsystem=transactions/log-store=log-store/transactions=0:ffff7f000001:-
b66efc2\4f9e6f8f\9/participants=2:refresh
```

注記

JBoss EAP 7.3 では、トランザクション失敗の例外はシリアライズされ、クライアントに渡されます。クラスパスに例外クラスがない場合は、クライアントに対して **ClassNotFoundException** が発生します。

JBoss EAP 7.3 には、**org.wildfly.common.rpc.RemoteExceptionCause** 例外が含まれます。クライアントはこれを **wildfly** ライブラリーからの例外として認識します。サーバーは元の例外をこの新しい例外としてクローンし、元の例外のすべてのフィールドを文字列フォームに配置し、例外のメッセージに追加します。その後、サーバーは **RemoteExceptionCause** の例外をクライアントに渡します。

3.3. トランザクション統計の表示

トランザクションマネージャーの統計が有効になっていると、トランザクションマネージャーによって処理されたトランザクションの統計を表示できます。トランザクションマネージャーの統計を有効にする方法については、JBoss EAP [設定ガイド](#) のトランザクションマネージャーの設定を参照してください。

管理コンソールまたは管理 CLI を使用して統計を表示できます。管理コンソールでは、**Runtime** タブから **Transaction** サブシステムを選択するとトランザクションの統計を表示できます。管理 CLI では、**read-resource** 操作に **include-runtime=true** を使用すると統計を表示できます。以下に例を示します。

```
/subsystem=transactions:read-resource(include-runtime=true)
```

以下の表は、管理コンソールの表示名、管理 CLI の属性、および各トランザクション統計の説明を示しています。

表3.1 Transactions サブシステムの統計

表示名	属性	説明
Aborted	number-of-aborted-transactions	中止したトランザクションの数。
Application Failures	number-of-application-rollback	失敗の原因がアプリケーションだった、タイムアウトを含む失敗したトランザクションの数。
Average Commit Time	average-commit-time	トランザクションコミットの平均時間 (ナノ秒単位)。クライアントがコミットを呼び出したときからトランザクションマネージャーがコミットの成功を判断するまで測定されます。
Committed	number-of-committed-transactions	コミットされたトランザクションの数。
Heuristics	number-of-heuristics	ヒューリスティック状態のトランザクションの数。
Inflight Transactions	number-of-inflight-transactions	開始済みであるが終了されていないトランザクション数。
Nested Transactions	number-of-nested-transactions	作成済みのネストされたトランザクションの総数。
Number of Transactions	number-of-transactions	ネストされたトランザクションを含む、作成済みのトランザクションの総数。
Resource Failures	number-of-resource-rollback	障害の原因がリソースであった失敗トランザクションの数。
System Failures	number-of-system-rollback	内部システムエラーが原因で、ロールバックされたトランザクションの数。
Timed Out	number-of-timed-out-transactions	タイムアウトが原因で、ロールバックされたトランザクションの数。

3.4. トランザクションオブジェクトストアの設定

トランザクションにはオブジェクトを保存する場所が必要です。オブジェクトストレージのオプションの1つがJDBC データソースです。特にパフォーマンスが気になる場合、JDBC オブジェクトストアはファイルシステムまたは ActiveMQ ジャーナルオブジェクトストアよりも速度が遅くなる場合があります。



重要

トランザクションログのストレージタイプとして Apache ActiveMQ Artemis ジャーナルを使用するよう **transactions** サブシステムが設定されている場合、JBoss EAP の 2 つのインスタンスは同じディレクトリを使用してジャーナルを保存することはできません。アプリケーションサーバーインスタンスは同じ場所を共有することはできず、アプリケーションサーバーインスタンスごとに一意な場所を設定する必要があります。



注記

トランザクションオブジェクトストアがないと、データの一貫性を保てなくなる可能性があります。そのため、オブジェクトストアを **安全な** ドライブに配置する必要があります。

JDBC データソースをトランザクションオブジェクトストアとして使用

JDBC データソースをトランザクションオブジェクトストアとして使用するには、以下の手順に従います。

1. データソース (例: **TransDS**) を作成します。非 XA データソースの手順は、JBoss EAP [設定ガイド](#) の **非 XA データソースの作成** を参照してください。JBoss EAP [設定ガイド](#) の説明どおり、オブジェクトストアが適切に動作するには、データソースの JDBC ドライバーを JAR デプロイメントとしてではなく、**コアモジュールとしてインストール** する必要があることに注意してください。
2. データソースの **jta** 属性を **false** に設定します。

```
/subsystem=datasources/data-source=TransDS:write-attribute(name=jta, value=false)
```

3. **jdbc-store-datasource** 属性を、使用するデータソースの Java Naming and Directory Interface 名に設定します (例: **java:jboss/datasources/TransDS**)。

```
/subsystem=transactions:write-attribute(name=jdbc-store-datasource, value=java:jboss/datasources/TransDS)
```

4. **use-jdbc-store** 属性を **true** に設定します。

```
/subsystem=transactions:write-attribute(name=use-jdbc-store, value=true)
```

5. JBoss EAP サーバーを再起動し、変更を反映します。

トランザクション JDBC ストア属性

以下の表は、JDBC オブジェクトストレージに関する利用可能な属性をすべて表しています。



注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-txn_4_0.xsd** のスキーマ定義ファイルで確認してください。

表3.2 トランザクション の JDBC ストア属性

プロパティ	説明
use-jdbc-store	トランザクションに対して JDBC ストアを有効にするには、 true に設定します。
jdbc-store-datasource	ストレージに使用される JDBC データソースの Java Naming および Directory Interface 名。
jdbc-action-store-drop-table	起動時にアクションストアテーブルをドロップおよび再作成するかどうか。デフォルトは false です。
jdbc-action-store-table-prefix	アクションストアテーブル名の接頭辞。
jdbc-communication-store-drop-table	起動時にコミュニケーションストアテーブルをドロップおよび再作成するかどうか。デフォルトは false です。
jdbc-communication-store-table-prefix	コミュニケーションストアテーブル名の接頭辞。
jdbc-state-store-drop-table	起動時にステートストアテーブルをドロップおよび再作成するかどうか。デフォルトは false です。
jdbc-state-store-table-prefix	ステートストアテーブル名の接頭辞。

ActiveMQ ジャーナルオブジェクトストアの使用

以下の手順に従って、ActiveMQ ジャーナルオブジェクトストアを使用します。

1. **use-journal-store** 属性を **true** に設定します。

```
/subsystem=transactions:write-attribute(name=use-journal-store,value=true)
```

2. JBoss EAP サーバーを再起動し、変更を反映します。

第4章 トランザクションの監視

4.1. TRANSACTIONS サブシステムのロギング設定

JBoss EAP の他のログ設定に依存せずにログに記録されたトランザクションに関する情報の量を制御できます。ログ設定は、管理コンソールまたは管理 CLI を使用して設定できます。

管理コンソールを使用したトランザクションロガーの設定

1. **Logging** サブシステム設定に移動します。
 - a. 管理コンソールで、**Configuration** タブをクリックします。管理対象ドメインを使用する場合は、適切なサーブプロファイルを選択する必要があります。
 - b. **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックします。
2. **com.arjuna** 属性を編集します。
Categories タブを選択します。**com.arjuna** エントリーがすでに存在します。**com.arjuna** を選択し、**編集** をクリックします。ログレベルを変更し、親ハンドラーを使用するかどうかを選択できます。
 - ログレベル:
トランザクションにより大量のロギング出力が生成されることがあるため、サーバーのログがトランザクション出力で満たされないようデフォルトのロギングレベルは **WARN** に設定されます。トランザクション処理の詳細を確認する必要がある場合は、トランザクション ID が表示されるよう **TRACE** ログレベルを使用します。
 - 親ハンドラーの使用:
親ハンドラーはロガーが出力を親ロガーに送信するかどうかを指定します。デフォルトの動作は **true** です。
3. **保存** をクリックして変更を保存します。

管理 CLI を使用したトランザクションロガーの設定

以下のコマンドを使用して管理 CLI からログレベルを設定します。スタンドアロンサーバーの場合は、コマンドから **/profile=default** を削除します。

```
/profile=default/subsystem=logging/logger=com.arjuna:write-attribute(name=level,value=VALUE)
```

4.1.1. TRACE ログレベルの有効化

TRACE ログレベルはログにデータを追加して、JBoss EAP で Jakarta コネクターの問題を診断できるようにします。以下の手順は、**org.jboss.jca**、**org.jboss.as.connector**、および **com.arjuna** クラスの **TRACE** レベルのロギングを有効にする方法を示しています。

前提条件

- JBoss EAP がインストールされている。

手順

1. 端末を開きます。
2. 管理 CLI を起動します。

3. 以下のいずれかのオプションを選択します。

- 管理対象ドメインで、**org.jboss.jca**、**org.jboss.as.connector**、**com.arjuna** クラスの **TRACE** ロギングレベルを有効にするには、以下のコマンドを使用します。

```
/profile=<PROFILE NAME>/subsystem=logging/logger=org.jboss.jca:add(level=TRACE)
/profile=<PROFILE NAME>/subsystem=logging/logger=org.jboss.as.connector:add(level=TRACE)
/profile=<PROFILE NAME>/subsystem=logging/logger=com.arjuna:write-attribute(name=level,value=TRACE)
```

<PROFILE NAME> を JBoss EAP プロファイル (**default**、**full**、または **full-ha**) に置き換えます。

- JBoss EAP をスタンドアロンサーバーとして実行する場合に、**org.jboss.jca**、**org.jboss.as.connector**、**com.arjuna** クラスの **TRACE** ロギングレベルを有効にするには、以下のコマンドを使用します。

```
/subsystem=logging/logger=org.jboss.jca:add(level=TRACE)
/subsystem=logging/logger=org.jboss.as.connector:add(level=TRACE)
/subsystem=logging/logger=com.arjuna:write-attribute(name=level,value=TRACE)
```

必要に応じて、次のコマンドを使用して、**console-handler** クラスで **TRACE** ログレベルを有効にします。

```
/subsystem=logging/console-handler=CONSOLE:write-attribute(name=level,value=TRACE)
```

コードスニペットは、適切な設定ファイルに追加されます。

```
<logger category="com.arjuna">
  <level name="TRACE"/>
</logger>

<logger category="org.jboss.jca">
  <level name="TRACE"/>
</logger>
<logger category="org.jboss.as.connector">
  <level name="TRACE"/>
</logger>
```

4.1.2. トランザクションブリッジロガーの有効化

トランザクションブリッジは XTS 上の層で、Transaction Manager の Jakarta Transactions または JTS コンポーネントの上にレイヤーです。これは JBoss EAP サーバーの他のコンポーネントと対話します。システムの操作の詳細を知るために、トランザクションマネージャーと対話するコンポーネントの詳細ロギングを有効にすることができます。

トランザクションブリッジは **logging** サブシステムを使用します。JBoss EAP サーバーの実行時、ロギングは **standalone-xts.xml** ファイルの **logging** サブシステム設定によって設定されます。トランザクションブリッジのロギングは、デバッグを行うときに便利です。

以下の管理 CLI コマンドを使用すると、**org.jboss.jbossts.txbridge** ロガーを設定し、トランザクションブリッジのロギングを有効にすることができます。

-

```
/subsystem=logging/logger=org.jboss.jbossts.txbridge:add(level=ALL)
```

これにより、サーバー設定ファイルに以下の XML が作成されます。

```
<logger category="org.jboss.jbossts.txbridge">
  <level name="ALL" />
</logger>
```



注記

デプロイメントの順番の問題により、**logging** サブシステムが完全に設定される前に、トランザクションブリッジを含むトランザクションマネージャーのコンポーネントがアクティブになることがあります。このような場合、デフォルトのロギングレベルが起動中に適用されるため、詳細なデバッグメッセージは出力されません。

以下の管理 CLI コマンドを使用すると、**com.arjuna** ロガーを設定し、詳細なロギングを有効にすることができます。

```
/subsystem=logging/logger=com.arjuna:write-attribute(name=level,value=ALL)
```

これにより、サーバー設定ファイルに以下の XML が作成されます。

```
<logger category="com.arjuna">
  <level name="ALL" />
</logger>
```

4.1.3. トランザクションログメッセージ

トランザクションロガーに **DEBUG** ログレベルを使用することにより、ログファイルを読み取り可能な状態に保ちつつトランザクションを追跡できます。詳細なデバッグの場合は、**TRACE** ログレベルを使用します。トランザクションロガーの設定に関する詳細については、[Transactions サブシステムのロギング設定](#) を参照してください。

TRACE ログレベルでログを記録するよう設定すると、トランザクションマネージャー (TM) は多くのロギング情報を生成できます。最も一般的なメッセージの一部は次のとおりです。このリストは包括的ではなく、他のメッセージが表示されることもあります。

表4.1 トランザクション状態の変更

トランザクションの開始	トランザクションが開始されたら、クラス com.arjuna.ats.arjuna.coordinator.BasicAction のメソッド Begin が実行され、メッセージ BasicAction::Begin() for action-id <transaction uid> でログに示されます。
トランザクションのコミット	トランザクションがコミットされたら、クラス com.arjuna.ats.arjuna.coordinator.BasicAction のメソッド Commit が実行され、メッセージ BasicAction::Commit() for action-id <transaction uid> でログに示されます。

トランザクションのロールバック	トランザクションがロールバックされたら、クラス com.arjuna.ats.arjuna.coordinator.BasicAction のメソッド Rollback が実行され、メッセージ BasicAction::Rollback() for action-id <transaction uid> でログに示されます。
トランザクションのタイムアウト	トランザクションがタイムアウトすると、 com.arjuna.ats.arjuna.coordinator.TransactionReaper のメソッド doCancellations が実行され、 Reaper Worker <thread id> attempting to cancel <transaction uid> とログに示されます。この結果、上記のように同じスレッドがトランザクションをロールバックすることが示されます。

4.1.4. トランザクションログファイルのデコード

4.1.4.1. トランザクションの XID/UID の特定

javax.transaction.TransactionManager インターフェイスは、トランザクション識別子を見つける方法を2つ提供します。

- **toString** メソッドを呼び出すと、識別子を含むトランザクションに関する完全な情報を出力することができます。
- この代わりに、**javax.transaction.Transaction** インスタンスを **com.arjuna.ats.jta.transaction.Transaction** にキャストし、**ArjunaCore Uid** を返す **get_uid** メソッドまたはブランチ識別子でないグローバル識別子の **Xid** を返す **getTxId** メソッドのいずれかを呼び出します。

```
com.arjuna.ats.jta.transaction.Transaction arjunaTM =
(com.arjuna.ats.jta.transaction.Transaction)tx.getTransaction();
System.out.println("Transaction UID" +arjunaTM.get_uid());
```

4.1.4.2. トランザクション状態とリソースの特定

TransactionStatusConnectionManager

TransactionStatusConnectionManager オブジェクトは、トランザクションの状態を取得するためにリカバリーモジュールによって使用されます。これは、アプリケーションプロセスで

TransactionStatusManager オブジェクトに接続する **TransactionStatusConnector** オブジェクトのテーブルを維持することで、**TransactionStatusManager** オブジェクトのプロキシのように動作します。

トランザクション **Uid** とトランザクションタイプ (ある場合) をパラメーターとして取る **getTransactionStatus** メソッドを使用すると、トランザクションの状態を取得することができます。

1. トランザクション **Uid** パラメーターのプロセス **Uid** フィールドは、トランザクションオブジェクトストアで目的の **TransactionStatusManagerItem** のホストとポートのペアを検索するために使用されます。
2. ホストとポートのペアは、**TransactionStatusConnector** オブジェクトを使用して目的の **TransactionStatusManager** オブジェクトへの TCP 接続を確立するために使用されます。
3. トランザクションの状態を取得するため、**TransactionStatusConnector** はトランザクション **Uid** とトランザクションタイプを **TransactionStatusManager** に渡します。

以下のコード例は、**TransactionStatusConnectionManager** を取得し、トランザクションの状態をチェックする方法を表しています。

```
// Transaction id
Uid tx = new Uid();
....
TransactionStatusConnectionManager tscm = new TransactionStatusConnectionManager();

// Check if the transaction aborted
assertEquals(tscm.getTransactionStatus(tx), ActionStatus.ABORTED);
```

TransactionStatusManager

TransactionStatusManager オブジェクトはリカバリーマネージャーのインターフェイスとして動作し、実行中のアプリケーションプロセスからトランザクションの状態を取得します。**com.arjuna.ats.arjuna.coordinator.TxControl** クラスによって、アプリケーションプロセスごとに1つの **TransactionStatusManager** が作成されます。TCP 接続は、リカバリーマネージャーと **TransactionStatusManager** との間の対話に使用されます。デフォルトでは、すべての空きポートが **TransactionStatusManager** によって使用されます。ただし、使用されるポートは、以下のプロパティを使用して修正できます。

```
$ EAP_HOME/bin/standalone.sh -
DRecoveryEnvironmentBean.transactionStatusManagerPort=NETWORK_PORT_NUMBER
```

1. 作成時、**TransactionStatusManager** は **TransactionStatusManagerItem** としてオブジェクトストアのホストと格納されるポートを取得します。
2. **Listener** スレッドが開始され、**TransactionStatusConnector** からの接続リクエストを待ちます。
3. 接続が確立されると、**AtomicActionStatusService** サービスを実行する **Connection** スレッドが作成されます。このサービスは、**TransactionStatusConnector** オブジェクトからトランザクション Uid およびトランザクションタイプ (ある場合) を受け入れます。
4. トランザクションの状態はローカルトランザクションテーブルから取得され、**TransactionStatusConnector** オブジェクトに返されます。

4.1.4.3. トランザクション履歴の表示

デフォルトでは、トランザクションサービスはトランザクションに関する履歴を維持しません。作成されたトランザクションの数と作成された各トランザクションの結果に関する情報をトランザクションサービスが維持するようにするには、**CoordinatorEnvironmentBean.enableStatistics** プロパティ変数を **true** に設定します。

以下の管理 CLI コマンドを使用して、統計を有効にします。

```
/subsystem=transactions:write-attribute(name=enable-statistics,value=true)
```

プログラムを使用してより詳細なトランザクション統計を取得するには、**com.arjuna.ats.arjuna.coordinator.TxStats** クラスを使用します。

TxStats クラスの例

```
public class TxStats
{
    /**
```



```
* @return the number of transactions (top-level and nested) created so far.
*/

public static int numberOfTransactions();

/**
 * @return the number of nested (sub) transactions created so far.
 */

public static int numberOfNestedTransactions();

/**
 * @return the number of transactions which have terminated with heuristic
 *         outcomes.
 */

public static int numberOfHeuristics();
/**
 * @return the number of committed transactions.
 */

public static int numberOfCommittedTransactions();

/**
 * @return the total number of transactions which have rolled back.
 */

public static int numberOfAbortedTransactions();

/**
 * @return total number of inflight (active) transactions.
 */

public static int numberOfInflightTransactions ();

/**
 * @return total number of transactions rolled back due to timeout.
 */

public static int numberOfTimedOutTransactions ();
/**
 * @return the number of transactions rolled back by the application.
 */

public static int numberOfApplicationRollbacks ();

/**
 * @return number of transactions rolled back by participants.
 */

public static int numberOfResourceRollbacks ();

/**
 * Print the current information.
 */
```

```
    public static void printStatus(java.io.PrintWriter pw);  
}
```

com.arjuna.ats.arjuna.coordinator.ActionManager クラスは、現在アクティブなトランザクションのリストを返す **getNumberOfInflightTransactions** メソッドを使用して、特定のアクティブなトランザクションに関する詳細を提供します。

第5章 トランザクションマネージャー例外の処理

5.1. タイムアウトしたトランザクションのデバッグ

トランザクションがタイムアウトする理由は多くあり、以下が含まれることがあります。

- サーバーのパフォーマンス遅延。
- 待機によるスレッドのスタック、またはハングアップ。
- スレッドが処理を完了するのに、設定したトランザクションのタイムアウト時間よりも多くの時間を必要とする。

以下のエラーメッセージのログを確認すると、タイムアウトしたトランザクションを特定できます。

```
WARN ARJUNA012117 "TransactionReaper::check timeout for TX {0} in state {1}"
```

{0} はトランザクションの Uid、{1} はトランザクションマネージャーの観点からのタイムアウトしたトランザクションの状態 {1} になります。

トランザクションマネージャーは、トランザクションタイムアウトのデバッグに以下のオプションを提供します。

- トランザクションのタイムアウト値を設定して、トランザクションのライフタイムを制御できます。コミットまたはロールバックが原因でトランザクションが終了する前にタイムアウト値が経過する場合、**transactions** サブシステムは transactions をロールバックします。
- **XAResource** インターフェイスの **setTransactionTimeout** メソッドを使用すると現在のトランザクションをリソースマネージャーに伝播できます。この操作がサポートされる場合、リソースマネージャーに関連するデフォルトのタイムアウトがオーバーライドされます。タイムアウトのオーバーライドは以下のような状況で役に立ちます。
 - 長期実行されているトランザクションのライフタイムがデフォルトよりも長い場合。
 - デフォルトのタイムアウトを使用すると、トランザクションの終了前にリソースマネージャーがロールバックし、トランザクションもロールバックする原因となる場合。
- タイムアウト値を指定しなかったり、**0** を値として使用すると、トランザクションマネージャーは実装固有のデフォルト値を使用します。JBoss EAP トランザクションマネージャーでは、**CoordinatorEnvironmentBean.defaultTimeout** プロパティはこの実装固有のデフォルト値を表します。デフォルト値は **300** 秒です。**0** を値として指定すると、デフォルトのトランザクションタイムアウトを無効にします。
以下の管理 CLI コマンドを使用するとデフォルトのトランザクションタイムアウトを変更できます。

```
/subsystem=transactions:write-attribute(name=default-timeout,value=VALUE)
```

管理対象ドメインで実行している場合、コマンドの前に **/profile=PROFILE_NAME** を付けて更新するプロファイルを指定する必要があります。

- JBoss EAP トランザクションマネージャーは、**XAResource** インスタンスでの **setTransactionTimeout** メソッドの呼び出しに、すべてか無かの選択をサポートします。**JTAEnvironmentBean.xaTransactionTimeoutEnabled** プロパティを **true** (デフォルト) に設定すると、すべてのインスタンスでメソッドを呼び出しできます。タイムアウトを無効

にしてトランザクションごとに設定する場合
は、**com.arjuna.ats.jta.common.JTAEnvironmentBean** クラスの
setXATransactionTimeoutEnabled メソッドを使用できます。

5.2. 新しい JBOSS EAP サーバーへのログの移行

前提条件

transactions サブシステムが新旧の JBoss EAP で同一に設定されるようにします。復元する必要のあるログはデータソースと通信する必要があるため、Jakarta Transactions データソースのリストが含まれる同じ設定が必要です。

5.2.1. ファイルベースのログストレージの移行

トランザクションマネージャーのログを新しい JBoss EAP サーバーに移行するには、ログを新しい JBoss EAP サーバーにコピーします。

以下のコマンドを使用するとファイルベースのログをコピーできます。

1. **EAP_HOME** ディレクトリーを参照します。
2. 以下のコマンドを使用してログのアーカイブを作成します。

```
$ tar -cf logs.tar ./standalone/data/tx-object-store
```

3. 以下のコマンドを使用して、アーカイブしたログを新しい **EAP_HOME** ディレクトリーで展開します。

```
$ tar -xf logs.tar -C NEW_EAP_HOME
```

5.2.2. JDBC ストアベースのログストレージの移行

- 新しい JBoss EAP サーバーを設定すると、[JDBC をトランザクションオブジェクトストアとして使用](#) に説明のある古いデータベースおよびテーブルを使用することができます。
- または、トランザクションログに使用するデータベースとテーブルを特定することもできます。その後、SQL ツールを使用してテーブルをバックアップし、新しいデータベースに復元できます。



注記

SQL クエリーツールは、JBoss EAP に同梱されている **h2** JAR ファイルにあります。

5.3. JBOSS EAP での XTS の有効化

トランザクションマネージャーの XML トランザクションサービス (XTS) コンポーネントは、ビジネストランザクションでのプライベートおよびパブリック web サービスのコーディネートをサポートします。XTS は、JBoss EAP サーバーでホストされる web サービスに WS-AT および WS-BA のサポートを提供します。これは任意のサブシステムで、**standalone-xts.xml** 設定を使用して有効にできます。

XTS が有効な状態で JBoss EAP サーバーを起動

1. JBoss EAP サーバーディレクトリーに移動します。

```
cd $EAP_HOME
```

2. XTS 設定ファイルのサンプルを **/configuration** ディレクトリーにコピーします。

```
cp docs/examples/configs/standalone-xts.xml standalone/configuration
```

3. **xts** 設定を指定して JBoss EAP サーバーを起動します。

Linux の場合:

```
bin/standalone.sh --server-config=standalone-xts.xml
```

Windows の場合:

```
bin\standalone.bat --server-config=standalone-xts.xml
```

5.4. 期限切れトランザクションの消去

期限切れトランザクションを消去できるプロパティーは次のとおりです。

ExpiryEntryMonitor

リカバリーマネージャーが期限切れスキャナースレッドを初期化すると、オブジェクトストアから期限切れのアイテムを削除するために使用される **ExpiryEntryMonitor** オブジェクトが作成されます。複数のスキャナーモジュールは動的にロードされ、特定タイプの期限切れのアイテムを削除します。

RecoveryEnvironmentBean.expiryScanners システムプロパティーを使用すると、プロパティーファイルのスキャナーモジュールを設定できます。スキャナーモジュールは初期化時にロードされます。

```
$ EAP_HOME/bin/standalone.sh -
DRecoveryEnvironmentBean.expiryScanners=CLASSNAME1,CLASSNAME2
```

expiryScanInterval

すべてのスキャナーモジュールは、期限切れのアイテムをスキャンするため、**ExpiryEntryMonitor** スレッドによって周期的に呼び出されます。この期間を時間単位で設定するには、以下の例のように **expiryScanInterval** システムプロパティーを使用します。

```
$ EAP_HOME/bin/standalone.sh -
DRecoveryEnvironmentBean.expiryScanInterval=EXPIRY_SCAN_INTERVAL
```

すべてのスキャナーモジュールは、**ExpiryScanner** インターフェイスから同じ動作を継承します。このインターフェイスは、以下を含むすべてのスキャナーモジュールによって実装されたスキャンメソッドを提供します。スキャナースレッドはこのスキャンメソッドを呼び出します。

ExpiredTransactionStatusManagerScanner

ExpiredTransactionStatusManagerScanner は期限切れの **TransactionStatusManagerItems** をオブジェクトストアから削除します。これらのアイテムは、削除される前に一定期間オブジェクトストアに保持されます (デフォルトでは 12 時間)。この期間を時間単位で設定するには、**transactionStatusManagerExpiryTime** システムプロパティーを以下の例のように使用します。

```
$ EAP_HOME/bin/standalone.sh -
```

```
DRecoveryEnvironmentBean.transactionStatusManagerExpiryTime=TRANSACTION_STATUS_
MANAGER_EXPIRY_TIME
```

AtomicActionExpiryScanner

AtomicActionExpiryScanner は、完了済みであると仮定された **AtomicActions** のトランザクションログを移動します。たとえば、参加者がコミットを指示された後、**transactions** サブシステムによるログの更新が可能になる前に障害が発生した場合、リカバリー時に JBoss EAP トランザクションマネージャーはコミットリクエストを再実行しようとします。しかし、これは失敗するため、ログは削除されません。破損やゼロ長などが理由で自動的にログをリカバリーできない場合、**AtomicActionExpiryScanner** も使用されます。すべてのログは、**/Expired** が付けられた以前の場所を基にして、特定の場所に移動されます。



注記

AtomicActionExpiryScanner はデフォルトで無効になっています。これをトランザクションマネージャーのプロパティファイルに追加すると有効にすることができます。破損したログに対応するために有効にする必要はありません。

5.5. ヒューリスティックな結果のリカバリー

トランザクションの更新をコミットまたはロールバックするために、分散トランザクションの完了段階中にトランザクションリソースが一方的な決定を行うと、ヒューリスティックな完了が発生します。これにより、分散データが不確定な状態のままになる可能性があります。可能性として、ネットワークの障害やリソースのタイムアウトがヒューリスティックな完了の原因となることがあります。ヒューリスティックな完了によって、以下のヒューリスティックな結果の例外の1つが発生します。

HEURISTIC_COMMIT

トランザクションマネージャーがロールバックの実行を決定するとこの例外が発生しますが、すべてのリソースはすでに独自にコミットしています。この場合、一貫して完了しているため、何もする必要はありません。

HEURISTIC_ROLLBACK

この例外は、トランザクションマネージャーからのコミットの決定が遅れたため、リソースがすべてロールバックしたことを意味します。一貫して完了しているため、**HEURISTIC_COMMIT** と同様に何もする必要はありません。

HEURISTIC_HAZARD

この例外は、更新の一部の処理が不明であるため発生します。不明なものはすべてコミットまたはロールバックされています。

HEURISTIC_MIXED

この例外は、トランザクションの一部がロールバックされ、残りの部分がコミットされると発生します。

この手順では、Jakarta Transactions を使用してトランザクションのヒューリスティックな結果を処理する方法を説明します。

1. トランザクションのヒューリスティックな結果の原因は、リソースマネージャーがコミットまたはロールバックの実行を約束したにも関わらず、約束を守らなかったことにあります。原因としては、サードパーティーコンポーネント、サードパーティーコンポーネントと JBoss EAP 間の統合レイヤー、または JBoss EAP 自体の問題が考えられます。

ヒューリスティックなエラーの最も一般的な2つの原因は、環境での一時的な障害と、リソースマネージャー対応時のコーディングエラーです。

2. 通常、環境内で一時的な障害が発生した場合は、ヒューリスティックなエラーを発見する前に気づくはずですが、原因としては、ネットワークの停止、ハードウェア障害、データベース障害、電源異常などが考えられます。
 ストレステストの実施中にテスト環境でヒューリスティックな結果が発生した場合は、テスト環境の脆弱性を意味します。



警告

JBoss EAP は、障害発生時にヒューリスティックな状態ではないトランザクションを自動的にリカバリーしますが、ヒューリスティックなトランザクションのリカバリーは実行しません。

3. 環境に明白な障害が発生していない場合や、ヒューリスティックな結果を簡単に再現できる場合は、おそらくコーディングエラーが原因です。サードパーティーベンダーに連絡して解決策があるかどうかを確認する必要があります。
 JBoss EAP のトランザクションマネージャー自体に問題があることを疑う場合は、サポートチケットを作成する必要があります。
4. 管理 CLI を使用して手動によるトランザクションのリカバリーを試すことができます。トランザクションを手作業でリカバリーする手順は、[トランザクション参加者のリカバリー](#)を参照してください。
5. トランザクションの結果を手作業で解決する処理は、障害の正確な状況によって異なります。環境に合わせて以下の手順を実行します。
 - a. 関係しているリソースマネージャーを特定します。
 - b. トランザクションマネージャーとリソースマネージャーの状態を調べます。
 - c. 関係するコンポーネントの1つまたは複数で、ログの消去とデータの照合を手動で強制します。
6. テスト環境である場合や、データの整合性を気にしない場合は、トランザクションログを削除して JBoss EAP を再起動すると、ヒューリスティックな結果はなくなります。デフォルトのトランザクションログの場所はスタンドアロンサーバーでは **EAP_HOME/standalone/data/tx-object-store/** ディレクトリー、管理対象ドメインでは **EAP_HOME/domain/servers/SERVER_NAME/data/tx-object-store/** ディレクトリーになります。管理対象ドメインの場合、**SERVER_NAME** は、サーバーグループに参加している個々のサーバー名を示します。



注記

トランザクションログの場所は、使用中のオブジェクトストアや、**object-store-relative-to** および **object-store-path** パラメーターに設定された値にも左右されます。標準のシャドーログや Apache ActiveMQ Artemis ログなどのファイルシステムログの場合、デフォルトディレクトリーの場所が使用されますが、JDBC オブジェクトストアを使用する場合は、トランザクションログはデータベースに保存されます。

5.5.1. ヒューリスティックな結果を判断するためのガイドライン

問題の検出

ヒューリスティックな決定は、トランザクションシステムで発生する可能性がある最も重大なエラーの1つです。これにより、トランザクションの一部がコミットされ、他の部分がロールバックされます。そのため、トランザクションのアトミックな特性に反し、データの整合性が保たれなくなる可能性があります。

リカバリー可能なリソースは、トランザクションマネージャーが要求するまで、ヒューリスティックな決定に関するすべての情報を安定したストレージに保持します。安定したストレージに保存される実際のデータは、リカバリー可能なリソースの種類によって異なり、標準化されていません。データを解析し、データ整合性の問題を修正するためにリソースを編集することも可能です。

ヒューリスティックな結果はサーバーログに保存され、リソースマネージャーとトランザクションマネージャーを使用して特定できます。

手作業によるトランザクションのコミットまたはロールバック

一般的に、トランザクションを手作業でコミットまたはロールバックすることはできません。JBoss EAP のトランザクション管理では、トランザクションを自動化リカバリーの保留リストに戻し、再度実行するか記録を削除します。以下に例を示します。

read-resource 操作を使用すると、トランザクションの参加者の状態を確認することができます。

```
/subsystem=transactions/log-store=log-store/transactions=0\:\ffff7f000001\:-
b66efc2\:4f9e6f8f\:9/participants=2:read-resource
```

結果は以下のようになります。

```
{
  "outcome" => "success",
  "result" => {
    "eis-product-name" => "ArtemisMQ",
    "eis-product-version" => "2.0",
    "jndi-name" => "java:/JmsXA",
    "status" => "HEURISTIC_HAZARD",
    "type" => "/StateManager/AbstractRecord/XAResourceRecord"
  }
}
```

この結果の状態は **HEURISTIC_HAZARD** で、リカバリーの対象となります。

HEURISTIC_HAZARD 例外のリカバリー

以下の手順は、**hazard** タイプのヒューリスティックの結果をリカバリーする方法の例になります。

1. リカバリーを開始するには、各リソースマネージャーを確認し、トランザクションマネージャーのツールから特定可能なさまざまなブランチの結果を確立する必要があります。しかし、リソースマネージャーによるコミットまたはロールバックを強制する必要はありません。リソースマネージャーを確認し、ヒューリスティックな例外の状態を認識してください。以下はさまざまなリソースマネージャーのヒューリスティックな結果をリストおよび解決するための参照リンクになります。



注記

これらのリンクは、参照のみを目的としており、変更する可能性があります。詳細はペンダーのドキュメントを参照してください。

- [Oracle でのインダウトトランザクションの手動解決](#)
- [DB2 でのインダウトトランザクションの手動解決](#)
- [PostgreSQL での 2 フェーズコミットの 準備済みトランザクション、コミット、ロールバック](#)
- [MySQL での XA トランザクション実装](#)
- [MariaDB での XA トランザクション実装](#)

2. 以下の例が示すように、recover 操作を実行する必要があります。

```
/subsystem=transactions/log-store=log-store/transactions=0\:\ffff7f000001\:-
b66efc2\:\4f9e6f8f\:\9/participants=2:recover
```

recover 操作を実行すると、トランザクションの状態が **PREPARE** に変更され、**commit** 操作を再実行してリカバリーが試行されます。リカバリーが正常に試行されると、参加者はトランザクションログから削除されます。

これを検証するには、再度 **log-store** 要素で **probe** 操作を実行します。参加者がリストされていないことを確認できるはずです。最後の参加者が削除されると、トランザクションも削除されます。

HEURISTIC_ROLLBACK および HEURISTIC_COMMIT 例外のリカバリー

ヒューリスティックな結果が **rollback** タイプである場合、以下を行います。

- リソースマネージャーが適切に実装されていれば、リソースはトランザクションをコミットできません。
- 残りのトランザクションを正常にコミットし、トランザクションストアから消去するために、**forget** 呼び出しを使用してリソースマネージャーからブランチを削除するべきかどうかを決定する必要があります。
- リソースマネージャーからブランチを削除しない場合は、トランザクションは永遠にトランザクションストアに保存されます。

しかし、ヒューリスティックな結果が **commit** タイプであった場合、ビジネスセマンティックを使用して一貫性のない結果に対応する必要があります。

手動調整に失敗した場合の追加手順

データベーストランザクションテーブルをチェックできます。これは、Oracle の **DBA_2PC_PENDING** テーブルです。しかし、これは特定のリソースマネージャーによって異なります。トランザクションマネージャーは、各リソースマネージャーで確認するブランチを提供することができます。

詳細は、このリソースマネージャーに関するベンダーのドキュメントを確認してください。サードパーティーのリソースマネージャーが問題の原因として疑われる場合は、その提供元とサポートチケットの作成を考慮してください。

