Oracle® Universal Connection Pool
開発者ガイド
19c

F16163-03(原本部品番号:E96473-04)

2021年12月

このガイドでは、Oracle Universal Connection Pool APIの使用方法について説明します。APIはJDBCドライバ・アグノスティックです。

タイトルおよび著作権情報

Oracle Universal Connection Pool開発者ガイド, 19c

F16163-03

原本著者: Tulika Das

原本協力著者: Tanmay Choudhury、Joseph Ruzzi、Tong Zhou、Yuri Dolgov、Paul Lo、Kuassi Mensah、Frances Zhao

原本協力者: Rajkumar Irudayaraj

目次

はじめに

Oracle Universal Connection Pool(UCP)は、データベース接続を管理するためのフル機能搭載の接続プールです。データベース集約型のJavaアプリケーションでは、この接続プールを使用することでパフォーマンスが向上し、システム・リソースをより効率的に利用します。

このガイドでは、UCP APIの使用方法を詳しく説明し、様々な使用例を示します。ただし、Oracle JDBCドライバ、Oracle DatabaseまたはSQLの使用については、UCPを理解する上で必要な場合を除き、詳しく説明しません。

対象読者

このガイドは、UCPを使用してJavaアプリケーション用のデータベース接続を作成および管理する方法を学習しようとするアプリケーション開発者およびシステム設計者を主に対象としています。このガイドを使用するにあたって、JavaおよびJDBCに精通している必要があります。UCPの一部の機能を使用する際には、Oracle Databaseの概念(Oracle RACやONSなど)についての知識が必要です。

ドキュメントのアクセシビリティについて

Oracleのアクセシビリティについての詳細情報は、Oracle Accessibility ProgramのWebサイト(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc)を参照してください。

Oracleサポートへのアクセス

サポートをご購入のOracleのお客様は、My Oracle Supportにアクセスして電子サポートを受けることができます。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info)か、聴覚に障害のあるお客様は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

関連ドキュメント

Oracle DatabaseでJavaを使用する方法の詳細は、Oracle Databaseドキュメント・セットの次のマニュアルを参照してください。

- 『Oracle Database JDBC開発者ガイド』
- 『Oracle Database 2日でJava開発者ガイド』
- 『Oracle Database Java開発者ガイド』

表記規則

このマニュアルでは次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック体	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。

規則	意味
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

このリリースのOracle Universal Connection Pool開発者ガイドにおける変更

この章の内容は次のとおりです。

Oracle Database 19cでの変更点

Oracle Database 19cのOracle Universal Connection Pool開発者ガイドにおける変更は次のとおりです。

新機能

この項では、このリリースでの新機能について説明します。

- リアクティブ・ストリームの収集のサポート

  リアクティブ・ストリームの収集のためのJavaライブラリのサポートを参照してください

- 接続検証タイムアウトの設定

  接続検証タイムアウトの設定を参照してください

- IAMトークンベース認証のサポート

  「Oracle Cloud InfrastructureでのIAMデータベース・アクセス・トークンを使用した認証」を参照してください

1 UCPの概要

この章の内容は次のとおりです。

- 接続プールの概要
- ユニバーサル接続プールの概要

1.1 接続プールの概要

接続プールは、データベース接続オブジェクトのキャッシュです。オブジェクトは、アプリケーションでデータベースへの接続に使用できる物理的なデータベース接続を表します。実行時に、アプリケーションはプールに接続をリクエストします。リクエストを満たすことができる接続がプールにある場合、その接続がアプリケーションに戻されます。接続が見つからない場合は、新しい接続が作成されてアプリケーションに戻されます。アプリケーションは、その接続を使用してデータベースで処理を実行した後、オブジェクトをプールに返します。その接続は次の接続リクエストに使用できます。

接続プールは、接続オブジェクトが再利用されるようにして、接続オブジェクトが作成される回数を減らします。接続プールにより、データベース集約型のアプリケーションのパフォーマンスは大幅に向上します。これは、接続オブジェクトの作成には時間とリソースの両面でコストがかかるためです。ネットワーク通信、接続文字列の読取り、認証、トランザクション参加、メモリー割当てなどのタスクはすべて、接続オブジェクトの作成に必要な時間およびリソースの一因となります。また、接続がすでに作成されているので、アプリケーションが接続を取得するための待機時間が短くなります。

多くの場合、接続プールには、プールのパフォーマンスを最適化するために使用されるプロパティが用意されています。これらのプロパティにより、プールで許容される最小および最大接続数や、接続がプールに返されるまでアイドル状態でいられる時間などの動作が制御されます。最適に構成された接続プールでは、短いレスポンス時間とプール内の接続を維持するために消費されるメモリーとのバランスがとられます。多くの場合、特定のアプリケーションに対して最適なバランスを実現するまでに、様々な設定を試す必要があります。

1.2 接続プールの使用のメリット

一般に、データベース集約型のアプリケーションに最も効果があるのは、接続プールです。データベース使用率がアプリケーションのパフォーマンスに影響を及ぼすことがわかっている場合は常に、方針としてアプリケーションで接続プールを使用する必要があります。

接続プールには、次のメリットがあります。

- 新しい接続オブジェクトが作成される回数を減らします。
- 接続オブジェクトの再利用を促します。
- 接続の取得プロセスを短縮します。
- 接続オブジェクトを手動で管理するために必要な労力を削減します。
- 失効した接続数を最小限にします。
- 接続の維持に消費されるリソース量を制御します。

1.3 ユニバーサル接続プールの概要

UCPは、JDBC接続をキャッシュするための接続プールを実装します。データベース集約型のJavaアプリケーションでは、この接続プールを使用することでパフォーマンスが向上し、システム・リソースをより効率的に利用します。

UCP JDBC接続プールでは任意のJDBCドライバを使用して物理接続を作成することができ、作成した接続はプールで管理されます。このプールは構成可能で、アプリケーションのパフォーマンス要件や可用性要件に基づいてプールの動作を最適化するために使用するプロパティが一式用意されています。さらに高度なアプリケーションでは、UCPのプール・マネージャを使用して、プール・インスタンスを管理できます。

また、プールは、Oracle Real Application Clusters (Oracle RAC)データベースを通して使用できる多くの高可用性およびパフォーマンス機能を利用します。これらの機能には、高速接続フェイルオーバー(FCF)、実行時接続ロード・バランシング(RLB)および接続アフィニティがあります。

ノート:

Oracle Database 11g リリース 2 から、単一インスタンス・データベースでの FCF も Oracle Restart でサポートされます。Oracle Restart は、独立サーバー用の Oracle Grid Infrastructure とも呼ばれます。

関連項目:

Oracle Restartの詳細は、『Oracle Database管理者ガイド』を参照してください。

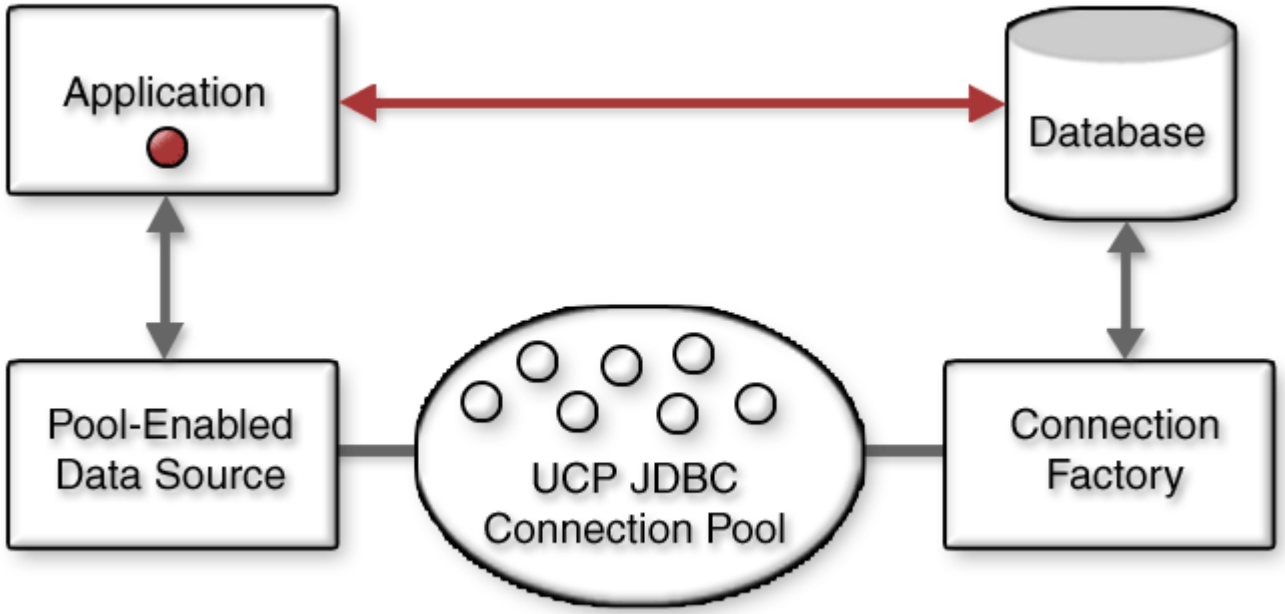
1.3.1 概念アーキテクチャ

アプリケーションは、UCPプール対応のデータソースを使用して、UCP JDBC接続プール・インスタンスから接続を取得します。PoolDataSourceデータソースは標準接続(java.sql.Connection)の取得に、PoolXADataSourceデータソースはXA (eXtended API)接続(javax.sql.XAConnection)の取得に使用されます。XAと非XAのどちらのUCP JDBC接続プールにも同じプール機能があります。

プール対応のデータソースは、コネクション・ファクトリ・クラスを利用して、プールで保持される物理的な接続を作成します。アプリケーションは、ConnectionオブジェクトまたはXAConnectionオブジェクトを作成できるファクトリ・クラスの使用を選択できます。プール対応のデータソースには、コネクション・ファクトリ・クラスを設定するためのメソッドの他、ファクトリ・クラスでデータベースへの接続に使用されるデータベースURLおよびデータベース資格証明書を設定するためのメソッドがあります。

アプリケーションは、接続ハンドルをプールから流用してデータベースで処理を実行します。処理が完了すると、接続はクローズされ、接続ハンドルはプールに返されて再利用できるようになります。次の図では、アプリケーションとUCP JDBC接続プール間のやりとりの概念図を示します。

図1-1 UCP JDBC接続プールの概念図



関連トピック

- UCPでのデータベース接続の取得

1.3.2 接続プールのプロパティ

UCP JDBC接続プールのプロパティは、プール対応のデータソースで使用可能なメソッドを使用して構成されます。プールのプロパティは、プール・サイズの制御や失効した接続の処理、接続がプールに返されるまで流用された状態でいられる時間に関する自律的な決定に使用されます。プール・プロパティの最適な設定は、アプリケーションおよびハードウェアのリソースによって決まります。通常、アプリケーションが接続を取得するために必要な時間と、一定のプール・サイズの維持に必要なメモリー量との間にはトレードオフが存在します。多くの場合、特定のアプリケーションについて目標のパフォーマンスを実現する最適なバランスを見つけるには、実験が必要です。

関連トピック

- ユニバーサル接続プールの動作の最適化

1.3.3 接続プール・マネージャ

UCPには、接続プールの管理統制を必要とするアプリケーションで使用される接続プール・マネージャがあります。このマネージャは、プールのライフ・サイクルの明示的な制御およびプールのメンテナンスに使用されます。また、マネージャを使用すると、管理コンソールを介してアプリケーションのプールとその管理の容易さを公開できます。

関連トピック

- 接続プール・マネージャの使用方法

1.3.4 高可用性およびパフォーマンスのシナリオ

UCP JDBC接続プールには、接続の高可用性およびパフォーマンスを確保するために使用される多くの機能があります。プールのリフレッシュや接続の検証など、これらの機能の多くは一般的なものであり、あらゆるドライバおよびデータベースの実装で機能します。実行時接続ロード・バランシング、接続アフィニティなど、これらの機能の一部は、Oracle JDBCドライバおよびOracle RACデータベースを使用する必要があります。

関連トピック

- Oracle RAC機能の使用方法

1.3.5 リアクティブ・ストリームの収集のためのJavaライブラリのサポート

Oracle Databaseリリース19cでは、リアクティブ・ストリームの収集をサポートするJavaライブラリが導入されています。これにより、ユーザーはデータをOracle Databaseに効率的にストリーミングできます。新しいJavaライブラリを使用すると、Javaアプリケーションは、大規模なクライアント・グループからデータを継続的に受信して収集できます。Oracle Databaseのダイレクト・パス・ロード・メソッドを使用してデータを挿入すると、収集プロセスがブロックされず、非常に高速になります。

このJavaライブラリは既存のUCP APIの拡張を使用しています。これにより、表パーティション、Oracle RAC接続アフィニティおよびシャーディングのサポートなどのデータベースの高可用性機能およびスケーラビリティ機能を収集プロセスに与えることができます。

関連項目:

リアクティブ・ストリームの収集のためのJavaライブラリの概要

2 スタート・ガイド

この章の内容は次のとおりです。

- UCPを使用するための要件
- UCPでの基本的な接続のステップ
- UCP APIの概要
- UCPを使用した基本的な接続の例

2.1 UCPを使用するための要件

UCPには、次の設計時および実行時の要件があります。

- JRE 1.5以上
- JDBCドライバまたはjava.sql.Connectionオブジェクトおよびjavax.sql.XAConnectionオブジェクトを戻すことができるコネクション・ファクトリ・クラス

> ノート:
>
> リリース 10.1 以上の Oracle ドライバがサポートされます。Oracle RAC や高速接続フェイルオーバーなどの Oracle Database の拡張機能には、Oracle Client ソフトウェアに同梱されている Oracle Notification Service ライブラリ(ons.jar)が必要です。

- ucp.jarライブラリ(アプリケーションのクラスパスに指定)
- ojdbc8.jarライブラリ(アプリケーションのクラスパスに指定)

> ノート:
>
> サード・パーティのデータベースおよびドライバとともに UCP を使用する場合でも、Oracle ojdbc8.jar ライブラリを使用する必要があります(UCP がこのライブラリに依存しているため)。

- SQL対応のデータベース。Oracle RACや高速接続フェイルオーバーなどの拡張機能には、Oracle Databaseが必要です。

2.2 UCPでの基本的な接続のステップ

UCPには、UCP JDBC接続プールから接続を流用するためにアプリケーションで使用されるプール対応のデータソースが用意されています。最も基本的な使用例では、接続プールは明示的に定義されません。かわりに、接続が流用される際に、デフォルトの接続プールが暗黙的に作成されます。

次のステップでは、データベースにアクセスするために、UCPプール対応のデータソースから接続を取得する方法について説明します。完全な例を例2-1に示します。

1. UCPのデータソース・ファクトリ(oracle.ucp.jdbc.PoolDataSourceFactory)を使用し、getPoolDataSourceメソッドを使用してプール対応のデータソースのインスタンスを取得します。データソース・インスタンスの型はPoolDataSourceである必要があります。次に例を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();
```

2. データベースへの物理的な接続の取得に必要な接続プロパティを設定します。これらのプロパティはデータソース・インスタンスに設定され、データベースに接続するためのURL、ユーザー名およびパスワード、物理的な接続の取得に使用されるコネクション・ファクトリなどがあります。これらは、JDBCドライバおよびデータベースに固有のプロパティです。次に例を示します。

```
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("<user>");
pds.setPassword("<password>");
```

3. 接続プールのデフォルトの動作を上書きするために、プール・プロパティを設定します。プール・プロパティはデータソース・インスタンスに設定されます。次に例を示します。

```
pds.setInitialPoolSize(5);
```

4. データソース・インスタンスを使用して接続を取得します。戻される接続は、データソースの接続プール内にある物理的な接続への論理的なハンドルです。次に例を示します。

```
Connection conn = pds.getConnection();
```

5. 接続を使用して、データベースで処理を実行します。

```
Statement stmt = conn.createStatement ();
stmt.execute("SELECT * FROM foo");
```

6. 接続をクローズし、プールに返します。

```
conn.close();
```

2.2.1 UCPでの認証

UCPは、透過的に認証します。つまり、接続の認証時に、PoolDataSourceはOracleDataSourceと同じように動作します。

UCPは、JDBC ThinドライバまたはJDBC OCIドライバが提示する次のすべての認証方式をサポートしており、基礎となるドライバに認証アクションを委譲します。

- Oracleウォレットに格納されているパスワードによる認証
- Kerberosを使用した認証
- SSL証明書による認証
- Lightweight Directory Access Protocol (LDAP)を使用した認証

2.2.2 Oracle Cloud InfrastructureでのIAMデータベース・アクセス・トークンを使用した認証

Oracle Databaseリリース19.13 (19.13.0.0.1)では、JDBC ThinドライバはIdentity and Access Management (IAM) Cloud Serviceによって生成されたデータベース・アクセス・トークンを使用して、共有Exadataインフラストラクチャ上のOracle Autonomous Databaseにアクセスできます。UCPはPoolDataSource.setTokenSupplier(Supplier)メソッド

を使用して、この認証タイプをサポートします。

関連項目:

Oracle Cloud InfrastructureでのIAMトークンベース認証のサポート

2.3 UCP APIの概要

次の項では、UCP APIの最も一般的に使用されるパッケージの簡単な概要について説明します。

関連項目:

APIの詳細は、『Oracle Universal Connection Pool Java API Reference』を参照してください。

oracle.ucp.jdbc

このパッケージには、JDBC接続および接続プールを使用して処理を実行するためにアプリケーションで使用される様々なインタフェースおよびクラスが含まれます。このパッケージに含まれるインタフェースの中でPoolDataSourceおよびPoolXADataSourceデータソース・インタフェースが、接続の取得と接続プールのプロパティの取得および設定のために使用されます。これら2つのインタフェースを実装するデータソース・インスタンスは、接続プールを自動的に作成します。

oracle.ucp.admin

このパッケージには、接続プール・マネージャを使用するためのインタフェースが含まれます。また、ユーザーがJMX操作を使用し、接続プールと接続プール・マネージャの操作および属性にアクセスできるようにするMBeanを使用するためのインタフェースも含まれます。インタフェースの中でも、UniversalConnectionPoolManagerインタフェースが、接続プール・インスタンスを作成およびメンテナンスするためのメソッドを提供します。

oracle.ucp

このパッケージには、接続プール機能の実装に使用される必須および任意のコールバック・インタフェースが含まれます。たとえば、ConnectionAffinityCallbackインタフェースは、接続アフィニティを有効または無効にするコールバックの作成に使用されますが、接続アフィニティの動作のカスタマイズにも使用できます。また、このパッケージには、統計クラス、UCP固有の例外クラス、およびデータソースを使用しないでUCPを直接使用するためのロジックも含まれます。

2.4 UCPを使用した基本的な接続の例

次の例では、データベースに接続して処理を実行し、終了するプログラムを示します。この例は単純で、場合によってはあまり実用的ではありません。しかし、データベースにアクセスするために、UCPプール対応のデータソースから接続を取得するのに必要な基本的なステップを示しています。

例2-1 基本的な接続の例

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

public class BasicConnectionExample {
   public static void main(String args[]) throws SQLException {
      try
```

```
      {
         //Create pool-enabled data source instance.

         PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

         //set the connection properties on the data source.

         pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
         pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
         pds.setUser("<user>");
         pds.setPassword("<password>");

         //Override any pool properties.

         pds.setInitialPoolSize(5);

         //Get a database connection from the datasource.

         Connection conn = pds.getConnection();

         System.out.println("¥nConnection obtained from " +
          "UniversalConnectionPool¥n");

         //do some work with the connection.
         Statement stmt = conn.createStatement();
         stmt.execute("select * from foo");

         //Close the Connection.

         conn.close();
         conn=null;

         System.out.println("Connection returned to the " +
          "UniversalConnectionPool¥n");

      }
      catch(SQLException e)
      {
         System.out.println("BasicConnectionExample - " +
          "main()-SQLException occurred : "
              + e.getMessage());
      }
   }
}
```

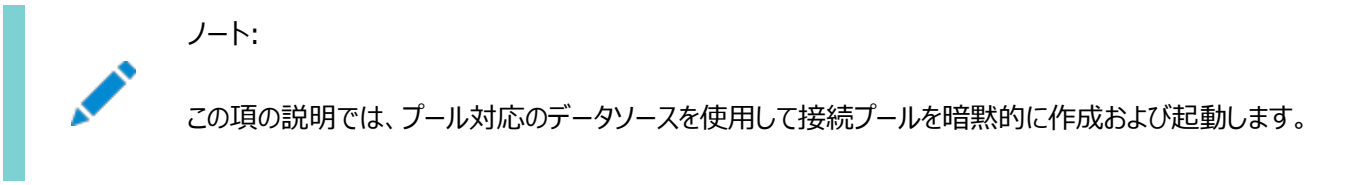
3 UCPでのデータベース接続の取得

この章の内容は次のとおりです。

3.1 UCPからの接続の流用について

アプリケーションは、プール対応のデータソースを使用して接続を流用します。この項では、接続の流用に関する次の概念について説明します。

- UCPからの接続の流用の概要
- プール対応のデータソースの使用方法
- プール対応のXAデータソースの使用方法
- 接続プロパティの設定
- JNDIを使用した接続の流用
- 接続初期化コールバックについて



ノート:

この項の説明では、プール対応のデータソースを使用して接続プールを暗黙的に作成および起動します。

3.1.1 UCPからの接続の流用の概要

UCP APIは、2つのプール対応のデータソースを備えています。1つは標準接続を流用するためのものであり、もう1つはXA接続を流用するためのものです。これらのデータソースは、UCP JDBC接続プール機能にアクセスでき、接続の流用に使用される一連のgetConnectionメソッドを備えています。XAと非XAのどちらのUCP JDBC接続プールにも同じプール機能があります。

UCP JDBC接続プールは、使用可能な接続と流用された接続の両方を保持します。アプリケーションが使用可能な接続と一致する接続の流用をリクエストすると、プールの接続が再利用されます。プール内の使用可能な接続がリクエストされた接続と一致しない場合は、新しい接続が作成されます。使用可能な接続と流用された接続の数は、プール・サイズ、タイムアウト間隔および検証ルールなどのプールのプロパティによって異なります。

3.1.2 プール対応のデータソースの使用方法

UCPは、データベースに接続するために使用されるプール対応のデータソース(oracle.ucp.jdbc.PoolDataSource)を提供します。oracle.ucp.jdbc.PoolDataSourceFactoryファクトリ・クラスは、プール対応のデータソース・インスタンスを作成するgetPoolDataSource()メソッドを備えています。次に例を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();
```

プール対応のデータソースには、実際の物理的な接続を取得するためにコネクション・ファクトリ・クラスが必要です。通常、コネクション・ファクトリはJDBCドライバの一部として提供され、データソースそのものにすることができます。UCP JDBC接続プールでは任意のJDBCドライバを使用して物理接続を作成することができ、作成した接続はプールで管理されます。setConnectionFactoryClassName(String)メソッドは、プール対応のデータソース・インスタンスにコネクション・ファクトリを定義するために使用されます。次の例では、JDBCドライバに付属するOracleのoracle.jdbc.pool.OracleDataSourceコネクション・ファクトリ・クラスを使用しています。他のベンダーのJDBCドライバを使用している場合は、該当するコネクション・ファクトリ・クラスについてベンダーのドキュメントを参照してください。

```
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
```

コネクション・ファクトリ・クラスに加えて、プール対応のデータソースには、データベースへの接続に使用されるURL、ユーザー名およびパスワードが必要です。プール対応のデータソース・インスタンスは、これらの各プロパティを設定するメソッドを備えています。次の例では、Oracle JDBC Thinドライバの構文を使用しています。他のベンダーのJDBCドライバを使用している場合は、使用する適切なURL構文についてベンダーのドキュメントを参照してください。

```
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("user");
pds.setPassword("password");
```

関連項目:

Oracle URL構文の使用方法は、『Oracle Database JDBC開発者ガイド』を参照してください。

最後に、プール対応のデータソースは、一連のgetConnectionメソッドを備えています。メソッドには次のものがあります。

- getConnection(): データベースへの接続に使用されたユーザー名およびパスワードと関連付けられている接続を戻します。
- getConnection(String username, String password): 指定されたユーザー名およびパスワードと関連付けられている接続を戻します。
- getConnection(java.util.Properties labels): 指定されたラベルと一致する接続を戻します。
- getConnection(String username, String password, java.util.Properties labels): 指定されたユーザー名およびパスワードと関連付けられ、指定されたラベルと一致する接続を戻します。

アプリケーションは、getConnectionメソッドを使用して、java.sql.Connectionタイプの接続ハンドルをプールから流用します。リクエストされた接続と一致する(同じURL、ユーザー名およびパスワード)接続ハンドルがすでにプール内に存在する場合は、その接続ハンドルがアプリケーションに戻されます。存在しない場合は、新しい接続が作成され、新しい接続ハンドルがアプリケーションに戻されます。OracleとMySQLの両方の例を示します。

Oracleのサンプル

次の例では、JDBC Thinドライバを使用する場合の接続の流用を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("<user>");
pds.setPassword("<password>");

Connection conn = pds.getConnection();
```

MySQLの例

次の例では、MySQLのConnector/J JDBCドライバを使用する場合の接続の流用を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionFactoryClassName("com.mysql.jdbc.jdbc2.optional.
   MysqlDataSource");
pds.setURL("jdbc:mysql://host:3306/dbname");
pds.setUser("<user>");
pds.setPassword("<password>");
```

```
Connection conn = pds.getConnection();
```

3.1.3 プール対応のXAデータソースの使用方法

UCPは、分散トランザクションに参加できるXA接続に使用されるプール対応のXAデータソース(oracle.ucp.jdbc.PoolXADataSource)を提供します。UCP JDBC XAプールには、非XAのUCP JDBCプールと同じ機能があります。oracle.ucp.jdbc.PoolDataSourceFactoryファクトリ・クラスは、プール対応のXAデータソース・インスタンスを作成するgetPoolXADataSource()メソッドを備えています。次に例を示します。

```
PoolXADataSource  pds = PoolDataSourceFactory.getPoolXADataSource();
```

プール対応のXAデータソース・インスタンスには、非XAデータソース・インスタンスと同様、実際の物理的な接続を取得するためにコネクション・ファクトリ、URL、ユーザー名およびパスワードが必要です。これらのプロパティは、非XAデータソース・インスタンスと同様の方法(前述参照)で設定されます。しかし、XA接続を取得するには、XA固有のコネクション・ファクトリ・クラスが必要です。通常、XAコネクション・ファクトリはJDBCドライバの一部として提供され、データソースそのものにすることができます。次の例では、JDBCドライバに付属するOracleのoracle.jdbc.xa.client.OracleXADataSource XAコネクション・ファクトリ・クラスを使用しています。他のベンダーのJDBCドライバを使用している場合は、該当するXAコネクション・ファクトリ・クラスについてベンダーのドキュメントを参照してください。

```
pds.setConnectionFactoryClassName("oracle.jdbc.xa.client.OracleXADataSource");
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("user");
pds.setPassword("password");
```

最後に、プール対応のXAデータソースは、一連のgetXAConnectionメソッドを備えています。これらのメソッドは、javax.sql.XAConnectionタイプの接続ハンドルをプールから流用するために使用されます。getXAConnectionメソッドは、前述のgetConnectionメソッドと同じです。次の例では、XA接続の流用を示します。

```
PoolXADataSource  pds = PoolDataSourceFactory.getPoolXADataSource();

pds.setConnectionFactoryClassName("oracle.jdbc.xa.client.OracleXADataSource");
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("<user>");
pds.setPassword("<password>");

XAConnection conn = pds.getXAConnection();
```

関連トピック

- UCPでの接続のラベル付け

3.1.4 接続プロパティの設定

Oracleのコネクション・ファクトリでは、特定の機能を使用して接続を構成するプロパティがサポートされます。UCPプール対応のデータソースは、setConnectionProperties(Properties)メソッドを備えています。このメソッドは、指定されたコネクション・ファクトリにプロパティを設定するために使用されます。次の例では、OracleのJDBCドライバの接続プロパティの設定を示します。他のベンダーのJDBCドライバを使用している場合は、ベンダー固有のドキュメントを参照して、この方法でのプロパティの設定がサポートされているかどうか、またどのプロパティが使用できるかを確認してください。

```
Properties connProps = new Properties();
connProps.put("fixedString", false);
connProps.put("remarksReporting", false);
connProps.put("restrictGetTables", false);
connProps.put("includeSynonyms", false);
```

```
connProps.put("defaultNChar", false);
connProps.put("AccumulateBatchResult", false);

pds.setConnectionProperties(connProps);
```

UCP JDBC接続プールは、プールの作成後および使用中にsetConnectionPropertiesがコールされた場合、作成済の接続を削除しません。

関連項目:

接続を構成するためのサポートされているプロパティの詳細なリストは、Oracle Database JDBC Java APIリファレンスを参照してください。たとえば、自動コミット・モードを設定するには、OracleConnection.CONNECTION_PROPERTY_AUTOCOMMITプロパティを使用できます。

3.1.5 JNDIを使用した接続の流用

接続は、プール対応のデータソースのJNDIルックアップを実行し、戻されたオブジェクトでgetConnection()をコールすることで、接続プールから流用できます。まず、プール対応のデータソースをJNDIコンテキストおよび論理名にバインドする必要があります。これは、オブジェクト参照の登録および検索が可能な、ネーミングおよびディレクトリ・サービス用のサービス・プロバイダ・インタフェース(SPI)をアプリケーションが実装していることを前提とします。

次の例では、Sun社のファイル・システムJNDIサービス・プロバイダを使用しています。このサービス・プロバイダは、次のJNDIソフトウェア・ダウンロード・ページからダウンロードできます。

http://www.oracle.com/technetwork/java/index.html

この例では、初期コンテキストを作成した後、MyPooledDataSourceという名前にバインドされているプール対応のデータソースのルックアップを実行しています。その後、戻されたオブジェクトを使用して接続プールから接続を流用しています。

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.fscontext.RefFSContextFactory");
env.put(Context.PROVIDER_URL, "file:/tmp");

ctx = new InitialContext(env);

PoolDataSource jpds = (PoolDataSource)ctx.lookup(MyPooledDataSource);
Connection conn = jpds.getConnection();
```

この例では、MyPoolDataSourceをコンテキストにバインドする必要があります。次に例を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("<user>");
pds.setPassword("<password>");

ctx.bind(MyPooledDataSource, pds);
```

3.1.6 接続初期化コールバックについて

接続初期化コールバックにより、アプリケーションおよびフレームワークはユニバーサル接続プールから取得される接続を初期化で

きます。プールからの接続のチェックアウトのたびに、またファイルオーバー中に再接続が成功するたびに、初期化コールバックが実行されます。

次の項で、初期化コールバックについて説明します。

- 接続初期化コールバックの概要
- 初期化コールバックの作成
- 初期化コールバックの登録
- 初期化コールバックの削除または登録解除

3.1.6.1 接続初期化コールバックの概要

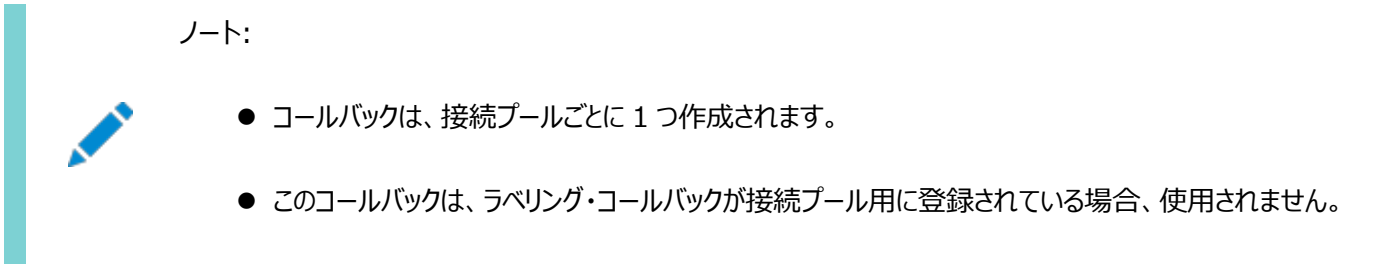
変更できないため、アプリケーションで接続ラベリングを使用しない場合、このようなアプリケーションには接続初期化コールバックが提供されます。

登録されると、接続がプールから流用されるたびに、またリカバリ可能なエラー後に再接続が成功するたびに、初期化コールバックは実行されます。実行時とリプレイ時の両方に同じコールバックを使用すると、元のセッションの確立時に使用されたのとまったく同じ初期化が実行時に確実に再設定されます。コールバックの起動が失敗した場合、リプレイはその接続で無効になります。

3.1.6.2 初期化コールバックの作成

UCP接続初期化コールバックを作成するために、アプリケーションは
oracle.ucp.jdbc.ConnectionInitializationCallbackインタフェースを実装しています。このインタフェースには、次のメソッドがあります。

```
void initialize(java.sql.Connection connection) throws SQLException;
```



ノート:

- コールバックは、接続プールごとに 1 つ作成されます。
- このコールバックは、ラベリング・コールバックが接続プール用に登録されている場合、使用されません。

例

次の例では、単純な初期化コールバックの作成方法を示します。

```
import oracle.ucp.jdbc.ConnectionInitializationCallback;
class MyConnectionInitializationCallback implements ConnectionInitializationCallback
{
  public MyConnectionInitializationCallback()
  {
    ...
  }
  public void initialize(java.sql.Connection connection) throws SQLException
  {
     // Reset the state for the connection, if necessary (like ALTER SESSION)
  }
}
```

3.1.6.3 初期化コールバックの登録

UCPには、接続初期化コールバックを登録するために、oracle.ucp.jdbc.PoolDataSourceインタフェースにregisterConnectionInitializationCallbackメソッドが用意されています。

```
public void registerConnectionInitializationCallback (ConnectionInitializationCallback cbk) throws
SQLException;
```

コールバックは、各接続プール・インスタンスに1つ登録できます。

3.1.6.4 初期化コールバックの削除または登録解除

UCPには、接続初期化コールバックを登録解除するために、oracle.ucp.jdbc.PoolDataSourceインタフェースにunregisterConnectionInitializationCallbackメソッドが用意されています。

```
public void unregisterConnectionInitializationCallback (ConnectionInitializationCallback cbk) throws
SQLException;
```

関連項目:

詳細は、『Oracle Universal Connection Pool Java API Reference』を参照してください

3.2 UCP接続プールのプロパティの設定

UCP JDBC接続プールは、接続プールのプロパティを使用して構成します。プロパティには、プール対応のデータソース・インスタンスで使用可能なgetメソッドとsetメソッドがあります。これらのメソッドは、プールをプログラムで構成するための便利な手段です。プールのプロパティが設定されていない場合、接続プールはデフォルトのプロパティ値を使用します。

次の例では、接続プールのプロパティの構成を示します。この例では、接続プール名およびプールで許容される最大/最小接続数を設定しています。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("JDBC_UCP");
pds.setMinPoolSize(4);pds.setMaxPoolSize(20);
```

UCP JDBC接続プールのプロパティは任意の順序で設定できます。また、実行時に動的に変更できます。たとえば、setMaxPoolSizeはいつでも変更できます。プールはその新しい値を認識し、その値に適応します。

関連トピック

- ユニバーサル接続プールの動作の最適化

3.3 UCPでの接続の検証の概要

接続は、接続の流用時にプールのプロパティを使用して検証できます。また、ValidConnectionインタフェースを使用してプログラムで検証することもできます。この項では、これら2つの方法について詳しく説明します。無効な接続は、アプリケーションのパフォーマンスおよび可用性に影響を及ぼす可能性があります。

3.3.1 流用時の検証

接続プールから接続を流用する際に、接続に対してSQL文を実行することで、接続を検証できます。接続の検証を有効にするには、次の2つの接続プールのプロパティを組み合せて使用します。

- setValidateConnectionOnBorrow(boolean)： 接続プールから接続を流用する際に、接続を検証するかどうかを指定します。このメソッドにより、プールから流用されるすべての接続に対して検証が有効になります。false値は、検証を実行しないことを意味します。デフォルト値はfalseです。
- setSQLForValidateConnection(String)： プールから接続を流用する際に、接続に対して実行されるSQL文を指定します。

ノート:

setSQLForValidateConnection プロパティは、Oracle JDBC ドライバを使用している場合には使用しないでください。Oracle JDBC ドライバを使用している場合、UCP は内部 ping を実行します。このメカニズムは SQL 文の実行よりも高速ですが、このプロパティを設定すると上書きされます。かわりに、setSQLForValidateConnection プロパティを使用せずに、setValidateConnectionOnBorrow プロパティを true に設定してください。

次の例では、プールから接続を流用する際の接続の検証を示します。この例では、MySQLのConnector/J JDBCドライバを使用しています。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionFactoryClassName("com.mysql.jdbc.jdbc2.optional.
   MysqlDataSource");
pds.setURL("jdbc:mysql://host:3306/mysql");
pds.setUser("<user>");
pds.setPassword("<password>");

pds.setValidateConnectionOnBorrow(true);
pds.setSQLForValidateConnection("select * from mysql.user");

Connection conn = pds.getConnection();
```

関連項目:

接続リクエスト遅延の最小化

3.3.2 setSecondsToTrustIdleConnection()メソッドを使用した接続検証の最小化

UCPでは、setValidateConnectionOnBorrow(boolean)メソッドの値をtrueすると、各接続がチェックアウト中に検証されます。この検証では、データベース接続を頻繁にチェックアウトするアプリケーションの大幅なオーバーヘッドが発生する場合があります。

頻繁な接続検証の影響を最小限に抑えるには、最近使用したまたは最近テストしたデータベース接続を信頼するために適切な値を使用したsetSecondsToTrustIdleConnection(int)メソッドを設定できます。この値を設定すると、接続検証テストがスキップされ、アプリケーション・パフォーマンスが大幅に向上します。

次の表に、この機能を使用するためにOracle Database 19cリリースで使用できる新しいメソッドを示します。

メソッド	説明
setSecondsToTrustIdleConnection(int secondsToTrustIdleConnection)	最近使用したまたは最近テストしたデータベース接続を信頼するために秒数を設定し、接続チェックアウト中の検証テストをスキップします。
getSecondsToTrustIdleConnection()	setSecondsToTrustIdleConnection(int)メソッドを使用して設定された値を取得します。

setSecondsToTrustIdleConnection(int)メソッドを正の値に設定すると、接続がsecondsToTrustIdleConnection(int)メソッドで指定された時間内に使用された場合に接続検証がスキップされます。デフォルト値は、機能が無効であることを意味する0秒です。

ノート:

setValidateConnectionOnBorrow(boolean)メソッドが true に設定されている場合のみ、setSecondsToTrustIdleConnection(int)メソッドは動作します。
setValidateConnectionOnBorrow(boolean)メソッドを true に設定せずにsetSecondsToTrustIdleConnection(int)メソッドをゼロ以外の値に設定すると、UCP は次の例外をスローします。

```
Invalid seconds to trust idle connection value or usage.
```

3.3.3 接続の有効性のチェック

oracle.ucp.jdbc.ValidConnectionインタフェースは、2つのメソッドisValidおよびsetInvalidを備えています。isValidメソッドは接続が使用可能かどうかを戻し、setInvalidメソッドはプール・インスタンスから接続を削除する必要があることを示すために使用されます。

isValidメソッドは、SQL例外がスローされた後でも接続が使用可能かどうかをチェックするために使用されます。また、いつでも流用された接続が有効であるかどうかをチェックするために使用できます。このメソッドは、Oracle RAC停止イベント後にトリガーされる高速接続フェイルオーバー・アクションなどの再試行メカニズムと組み合せると、特に便利です。

ノート:

- isValid メソッドは、プール・インスタンスおよび Oracle JDBC ドライバを調べて、接続がまだ有効かどうかを判断します。プールとドライバの両方で接続がまだ有効であることがレポートされた場合にかぎり、isValid メソッドによってデータベースへのラウンドトリップが発生します。このラウンドトリップを使用して、プールまたはドライバですぐには検出されないデータベース障害がないかをチェックします。
- Oracle Database リリース 18c 以降、ピンポン・プロトコルを使用してデータベースへの完全なラウンドトリップを行うメソッドの旧バージョンとは異なり、空のパケットをデータベースに送信する isValid メソッドの新しいバリアントがあります。

関連項目:

『Oracle Database JDBC開発者ガイド』

また、isValidメソッドは、接続タイムアウト機能および接続獲得機能と組み合せて使用しても便利です。これらの機能は、アプリケーションで接続がまだ保持されているときに、接続をプールに返すことができます。このような場合、isValidメソッドはfalseを戻し、アプリケーションが新しい接続を取得できるようにします。

次の例では、isValidメソッドの使用方法を示します。

```
try {  conn = poolDataSouorce.getConnection  ...}catch (SQLException sqlexc)
{
   if (conn == null || !((ValidConnection) conn).isValid())

   // take the appropriate action

...
conn.close();
}
```

XAアプリケーションでは、isValid()メソッドをコールする前に、PoolXADataSourceから取得されるXAConnectionをValidConnectionにキャストする必要があります。XAConnection.getConnection()メソッドをコールして取得されるConnectionをValidConnecionにキャストすると、例外がスローされることがあります。

関連トピック

- Oracle RAC機能の使用方法
- UCPからの接続の削除

3.4 UCPへの流用された接続の返却

流用された接続のうち、もう使用しないものはプールに返して、次の接続リクエストで使用できるようにする必要があります。closeメソッドにより、接続をクローズしてプールに自動的に返します。closeメソッドは、プールから接続を物理的に削除しません。

クローズしない場合、流用された接続は流用されたままになります。つまり、使用可能な接続がない場合に、後続の接続リクエストによって新しい接続が作成されます。この動作は、多くの接続が作成される原因となり、システム・パフォーマンスに影響を及ぼす可能性があります。

次の例では、接続のクローズおよびプールへの返却を示します。

```
Connection conn = pds.getConnection();

//do some work with the connection.

conn.close();
conn=null;
```

3.5 UCPからの接続の削除

ValidConnectionインタフェースのsetInvalidメソッドは、接続のクローズ時に、接続プールから接続を削除する必要があることを示します。通常、このメソッドは、例外の後やValidConnectionインタフェースのisValidメソッドがfalseを戻した場合など、接続が使用できなくなったときに使用されます。また、接続の状態が悪いとアプリケーションが判断した場合にも使用できます。
次の例では、setInvalidメソッドを使用した接続のクローズおよびプールからの削除を示します。

```
Connection conn = pds.getConnection();
...

((ValidConnection) conn).setInvalid();
...

conn.close();
conn=null;
```

3.6 サード・パーティ製品とのUCP統合

ミドルウェア・プラットフォームやフレームワークなどのサード・パーティ製品では、UCPを使用してアプリケーションおよびサービスに接続プーリング機能を提供できます。UCP統合には、スタンドアロン・アプリケーションで使用できるものと同じ接続プール機能があり、Oracle Databaseとの緊密な統合を提供します。

2つのデータソース・クラスPoolDataSourceImpl(非XA接続プール用)とPoolXADataSourceImpl(XA接続プール用)が、UCPとの統合点として使用できます。どちらのクラスもoracle.ucp.jdbcパッケージにあります。これらのクラスは、それぞれPoolDataSourceインタフェースおよびPoolXADataSourceインタフェースの実装で、デフォルトのコンストラクタがあります。

関連項目:

実装クラスの詳細は、Oracle Universal Connection Pool Java APIリファレンスを参照してください。

これらの実装では、接続プール・インスタンスを明示的に作成し、接続を返すことができます。次に例を示します。

```
PoolXADataSource pds = new PoolXADataSourceImpl();

pds.setConnectionFactoryClassName("oracle.jdbc.xa.client.OracleXADataSource");
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("user");
pds.setPassword("password");

XAConnection conn = pds.getXAConnection();
```

サード・パーティ製品では、これらのデータソース実装クラスをインスタンス化できます。また、これらのインタフェースのメソッドは、JavaBean設計パターンに準拠しているため、リフレクションを使用して接続プールのプロパティをクラスに設定するために使用できます。たとえば、Oracle JDBCコネクション・ファクトリおよびデータベースを使用するUCPデータソースは、次のように定義して、JNDIレジストリにロードできます。

```
<data-sources>
   <data-source
      name="UCPDataSource"
      jndi-name="jdbc/UCP_DS"
      data-source-class="oracle.ucp.jdbc.PoolDataSourceImpl">
      <property name="ConnectionFactoryClassName"
                value="oracle.jdbc.pool.OracleDataSource"/>
      <property name="URL" value="jdbc:oracle:thin:@//localhost:1521:oracle"/>
      <property name="User" value"user"/>
      <property name="Password" value="password"/>
      <property name="ConnectionPoolName" value="MyPool"/>
      <property name="MinPoolSize" value="5"/>
      <property name="MaxPoolSize" value="50"/>
   </data-source>
</data-sources>
```

リフレクションを使用する場合、name属性はプロパティの設定に使用されるsetterメソッドの名前(大/小文字を区別)と一致します。つまり、次のようにして、アプリケーションでデータソースを使用できます。

```
Connection connection = null;
try {
   InitialContext context = new InitialContext();
   DataSource ds = (DataSource) context.lookup( "jdbc/UCP_DS" );
```

```
connection = ds.getConnection();
...
```

4 ユニバーサル接続プールの動作の最適化

この章の項目は次のとおりです。

4.1 接続プールの最適化

この項では、プーリング動作を最適化するための接続プールのプロパティの設定方法について説明します。作成時に、UCP JDBC接続プールはデフォルト設定で事前構成されます。デフォルト設定により、一般的な汎用の接続プールとなります。しかし、アプリケーションにはそれぞれ異なるデータベース接続要件があり、接続プールのデフォルト動作の変更が必要な場合があります。プール・サイズや接続タイムアウトなどの動作を構成して、接続プール全体のパフォーマンスだけでなく接続の可用性も向上させることができます。多くの場合、特定のアプリケーションに合せて接続プールをチューニングする最善の方法は、最適なパフォーマンスおよびスループットを達成するまで、様々な値を使用して様々なプロパティの組合せを試すことです。

接続プールのプロパティの設定

接続プールのプロパティが設定されるのは、プール対応のデータソースを介して接続を取得するときか、接続プール・マネージャを使用して接続プールを作成するときです。

次の例では、プール対応のデータソースを介した接続プールのプロパティの設定を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("JDBC_UCP");
pds.setMinPoolSize(4);pds.setMaxPoolSize(20);
...
```

次の例では、接続プール・マネージャを使用して接続プールを作成する場合の接続プールのプロパティの設定を示します。

```
UniversalConnectionPoolManager mgr = UniversalConnectionPoolManagerImpl.
getUniversalConnectionPoolManager();

pds.setConnectionPoolName("JDBC_UCP");
pds.setMinPoolSize(4);pds.setMaxPoolSize(20);
...

mgr.createConnectionPool(pds);
```

4.2 UCPでのプール・サイズの制御について

UCP JDBC接続プールには、プール・サイズの制御に使用される一連のプロパティがあります。これらのプロパティを使用すると、要求の増減につれて、プール内の接続数を増減できるようになります。この動的な動作は、不要な接続の維持に浪費される場合のあるシステム・リソースの節約に役立ちます。

この節では、以下のトピックについて説明します。

- 初期プール・サイズの設定
- 最小プール・サイズの設定
- 最大プール・サイズの設定

4.2.1 初期プール・サイズの設定

初期プール・サイズのプロパティは、接続プールの初回作成時または再初期化時に作成される使用可能な接続の数を指定します。通常、このプロパティは、プールを最適なサイズにすることで、発生する起動時間を削減するために使用されます。

値0は、接続を事前作成しないことを示します。デフォルト値は0です。次の例では、初期プール・サイズの構成を示します。

```
pds.setInitialPoolSize(5);
```

初期プール・サイズのプロパティが最大プール・サイズのプロパティより大きい場合は、最大数の接続のみが初期化されます。

初期プール・サイズのプロパティが最小プール・サイズのプロパティより小さい場合は、初期数の接続のみが初期化され、最小プール・サイズ値を満たす十分な接続が作成されるまで維持されます。

4.2.2 最小プール・サイズの設定

最小プール・サイズのプロパティは、プールが保持する使用可能な接続および流用された接続の最小数を指定します。接続プールがまだ最小サイズに達していない場合を除いて、常に指定された最小プール・サイズに戻ろうとします。たとえば、最小限度が10に設定され、まだ2つの接続しか作成および流用されていない場合は、この数値が最小プール・サイズを下回っているため、プールで保持される接続数は2のままです。

このプロパティを使用すると、要求が減少するにつれてプール内の接続数を減らすことができます。同時に、システム・リソースが不要な接続の維持のために浪費されないようにします。

デフォルト値は0です。次の例では、最小プール・サイズの構成を示します。

```
pds.setMinPoolSize(2);
```

4.2.3 最大プール・サイズの設定

最大プール・サイズのプロパティは、プールが保持する使用可能な接続および流用された(使用中の)接続の最大数を指定します。最大数の接続が流用された場合、プールに返されるまで接続は利用できません。

このプロパティを使用すると、要求が増加するにつれてプール内の接続数を増やすことができます。同時に、プールがシステムのリソースを使い果し、最終的にアプリケーションのパフォーマンスや可用性に影響を及ぼすほど大きくならないようにします。

値0は、プールで保持される接続がないことを示します。接続を取得しようとすると例外が発生します。デフォルト値では、最大でInteger.MAX_VALUE(デフォルトでは2147483647)まで接続を作成し続けることができます。次の例では、最大プール・サイズの構成を示します。

```
pds.setMaxPoolSize(100);
```

4.3 静的接続プールを使用したReal-World Performanceの最適化について

Real-World Performanceグループが調査するオンライン・トランザクション処理(OLTP)のパフォーマンス問題のほとんどは、アプリケーションで使用される接続方法に関連します。このため、健全な接続方法を設計することはシステム・パフォーマンスにおいて重要で、特に、規模を拡大して増加する要件を満たす必要のある企業環境では重要です。

ほとんどのアプリケーションは、データベースで開いたままにする最小数の接続とデータベースに行うことができる最大数の接続を構成したデータベースへの動的プールの接続を使用します。アプリケーションでデータベースへの接続を必要とする場合、プールから要求します。使用できる接続がないと、接続の最大数に達していない場合は新しい接続が作成されます。接続が指定された期間使用されていない場合、使用できる接続の最小数を超えると接続が閉じられます。

この構成は、アプリケーションが必要とするアクティブな接続数を維持する場合のみ、システム・リソースを節約します。実際、この構成では、接続ストームおよびデータベース・システムのCPUオーバーサブスクリプションが有効になり、システムがすぐに不安定になります。データベース接続が必要なアプリケーション・サーバーの多くのアクティビティが存在する場合、接続ストームが発生する可能性があります。すべてのリクエストを満たすデータベースへの十分な接続がない場合、アプリケーション・サーバーは新しい接続を開きます。データベースの新しい接続の作成は、リソース消費型アクティビティです。多くの接続が短期間に行われる場合、データベース・システムのCPUリソースが過負荷になる可能性があります。

そのため、静的接続プールを作成するには、データベース・システムへの接続数は、システムで使用できるCPUコアに基づく必要があります。CPUコアごとに1-10個の接続をお薦めします。理想的な数値は、アプリケーションおよびシステム・ハードウェアによって異なります。ただし、値はその範囲内です。Real-World Performanceグループでは、接続の最小数と最大数を同じ値に設定して、データベースへの接続の静的プールを作成することを推奨します。これにより、データベース接続の数を事前定義された値に維持して接続ストームを防ぎます。

たとえば、データベース・サーバーに2 CPU、CPUごとに12コア、コアごとに2スレッドがある場合、24コアを使用可能で、データベースへの接続数を12から120の間にする必要があります。CPUコアのみが命令を実行できるため、スレッド数は考慮されません。システムに複数のデータベースがある場合、この数値は、システムに接続しているすべてのアプリケーションおよびすべてのデータベースに対して累積されます。2つのアプリケーション・サーバーがある場合、接続の最大数(たとえば、ここでは120)を分割する必要があります。システムで実行している2つのデータベースがある場合、接続の最大数(120個の接続)を分割する必要があります。

関連項目:

- https://www.youtube.com/watch?v=Oo-tBpVewP4
- https://www.youtube.com/watch?v=XzN8Rp6glEo

4.4 UCPでの失効した接続

失効した接続とは、使用可能であるか流用中であるにもかかわらず、使用されなくなった接続です。流用されたままの失効した接続は、接続の可用性に影響を及ぼすことがあります。また、失効した接続があると、使用されていない接続を長期間維持するためにリソースが浪費されることから、システム・リソースに影響を及ぼすことがあります。この項で説明するプール・プロパティを使用して、失効した接続を制御します。

この節では、以下のトピックについて説明します。

- 接続再利用とは
- 中止接続タイムアウトの設定
- TTL接続タイムアウトの設定
- 接続待機タイムアウトの設定
- 非アクティブ接続タイムアウトの設定
- 問合せタイムアウトの設定
- タイムアウト・チェック間隔の設定

ノート:

アプリケーションで不要になった接続はすべてクローズすることをお薦めします。接続をクローズすると、流用されたままの失効した接続の数を少なくすることができます。

4.4.1 接続再利用とは

接続再利用機能を使用すると、一定の時間が経過した後または接続が一定の回数使用された後に、接続を適切にクローズして接続プールから削除できます。また、使用できない接続の維持に浪費されることになるシステム・リソースを節約します。

4.4.1.1 最大接続再使用時間の設定

最大接続再使用時間を使用すると、一定の時間使用された後に、接続を適切にクローズしてプールから削除できます。このプロパティのタイマーは、接続が物理的に作成されると開始します。流用された接続はプールに返された後にしかクローズされないため、再使用時間を超過します。

通常、この機能は、ファイアウォールがプール層とデータベース層の間に存在し、時間制限に基づいて接続をブロックするように設定されている場合に使用されます。ブロックされた接続は、使用できないにもかかわらずプールに残存します。このような場合、接続再使用時間をファイアウォールのタイムアウト・ポリシーより小さい値に設定します。

ノート:

最大接続再使用時間は、TTL 接続タイムアウトとは異なります。TTL 接続タイムアウトは、接続がプールから流用されると開始します。一方、最大接続再使用時間は、接続が物理的に作成されると開始します。また、TTL タイムアウトでは、流用期間中にタイムアウトに達すると、接続をクローズして再利用のためにプールに返します。最大接

続再使用時間では、タイムアウトに達すると、接続をクローズしてプールから破棄します。

最大接続再使用時間の値は秒単位です。値0は、この機能が無効であることを示します。デフォルト値は0です。次の例では、最大接続再使用時間の構成を示します。

```
pds.setMaxConnectionReuseTime(300);
```

関連トピック

- TTL接続タイムアウトの設定

4.4.1.2 最大接続再使用数の設定

最大接続再使用数を使用すると、一定の回数流用された後に、接続を適切にクローズして接続プールから削除できます。通常、このプロパティは、メモリー・リークなどの問題を解消するために接続を定期的にリサイクルするために使用されます。

値0は、この機能が無効であることを示します。デフォルト値は0です。次の例では、最大接続再使用数の構成を示します。

```
pds.setMaxConnectionReuseCount(100);
```

4.4.2 接続検証タイムアウトの設定

接続検証タイムアウトは、プールから流用された接続が検証される期間を指定します。これは接続検証操作の最大時間です。この期間中に検証が完了しなかった場合、接続は無効とみなされます。

接続検証タイムアウトの値は秒単位です。デフォルト値は15に設定されます。次の例では、接続検証タイムアウトの構成を示します。

```
pd.setConnectionValidationTimeout(55);
```

4.4.3 中止接続タイムアウトの設定

中止接続タイムアウト(ACT)を使用すると、流用された接続が一定時間使用されなかった場合に、接続プールに戻すことができます。中止の決定は、データベースへのコールを監視することで行われます。このタイムアウト機能は、接続再利用が最大限になるようにし、使用されていない流用された接続の維持に浪費されることになるシステム・リソースを節約します。

ノート:

UCP では、再利用のために接続を回収する前に、ローカル・トランザクションが保留中である接続を取り消すかロールバックします。

ACT値は秒単位です。値0は、この機能が無効であることを示します。デフォルト値は0に設定されます。次の例では、中止接続タイムアウトの構成を示します。

```
pds.setAbandonedConnectionTimeout(10);
```

すべての接続が一定時間後にリープされます。ACTに達したとき、またはACTを免れた場合は、その免除の期限が切れるとリープされます。プールにACTを設定した場合は、次のようになります。

- 文に対してStatement.setQueryTimeoutメソッドをコールせずにその文を実行した場合、サーバーが問合せに応答するのを接続が待機していても、ACTを超えると接続はリープされます。
- Statement.setQueryTimeoutメソッドをコールして文を実行した場合、問合せタイムアウトおよびACTに達した後に接続はリープされます。問合せタイムアウトの待機中に、接続はリープされません。問合せタイムアウトの満了は、ACTタイマーをリセットするイベントです。問合せタイムアウトの満了時に発生するcancelアクションの待機中にACTに達した場合、接続はリープされます。
- 1つの接続に、問合せタイムアウトが設定されたs1と問合せタイムアウトが設定されていないs2の2つの文がある場合、s1が問合せタイムアウトを待機している間、ACTによって接続はリープされませんが、s2がハングした場合、接続はリープされます。

  2つの文は、JDBCの要件に基づいて順番に実行されることに注意してください。

4.4.4 TTL接続タイムアウトの設定

TTL接続タイムアウトを使用すると、流用された接続を一定時間流用されたままにした後で、接続をプールに回収できます。このタイムアウト機能は、接続が最大限に再利用されるようにし、想定される使用時間よりも長く接続を維持するために浪費されることになるシステム・リソースの節約に役立ちます。

ノート:

UCP では、再利用のために接続を回収する前に、ローカル・トランザクションが保留中である接続を取り消すかロールバックします。

TTL接続タイムアウトの値は秒単位です。値0は、この機能が無効であることを示します。デフォルト値は0に設定されます。次の例では、TTL接続タイムアウトの構成を示します。

```
pds.setTimeToLiveConnectionTimeout(18000)
```

4.4.5 接続待機タイムアウトの設定

接続待機タイムアウトは、プールに接続がなくなった場合にアプリケーション・リクエストが接続を取得するために待機する時間を指定します。プール内の接続がすべて使用されている(流用されている)場合、およびプール・サイズが最大プール・サイズのプロパティで指定されている最大接続許容数に達している場合、接続プールには接続がなくなります。タイムアウト値に達すると、リクエストはSQL例外を受け取ります。その場合、アプリケーションは接続の取得を再試行できます。このタイムアウト機能により、アプリケーションがブロックされる時間を最小限にすることでアプリケーション全体の有用性が向上し、適切なリカバリを実行できます。

接続待機タイムアウトの値は秒単位です。値0は、この機能が無効であることを示します。デフォルト値は3秒に設定されます。次の例では、接続待機タイムアウトの構成を示します。

```
pds.setConnectionWaitTimeout(10);
```

4.4.6 非アクティブ接続タイムアウトの設定

非アクティブ接続タイムアウトは、クローズしてプールから削除されるまでの、使用可能な接続がアイドル状態でいられる時間を指定します。このタイムアウト・プロパティは、使用可能な接続にのみ適用でき、流用された接続には作用しません。このプロパ

ティは、使用されなくなった接続の維持に浪費されることになるリソースの節約に役立ちます。非アクティブ接続タイムアウトを(最大プール・サイズとともに)使用すると、アプリケーション・ロードが変化するにつれて接続プールを拡大または縮小できます。

非アクティブ接続タイムアウトの値は秒単位です。値0は、この機能が無効であることを示します。デフォルト値は0に設定されます。次の例では、非アクティブ接続タイムアウトの構成を示します。

```
pds.setInactiveConnectionTimeout(60);
```

4.4.7 問合せタイムアウトの設定

Oracle Database 12cリリース2 (12.2.0.1)では、UCPにqueryTimeoutプロパティが導入されました。このプロパティでは、UCPがStatementオブジェクトの実行を待機する秒数を指定します。この制限を超えると、DatabaseExceptionがスローされます。次の方法でこのプロパティを設定するsetQueryTimeoutメソッドを使用します。

```
...
PoolDataSourceImpl pds = new PoolDataSourceImpl();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL(<url>);
pds.setUser("scott");
pds.setPassword(<password>);
pds.setConnectionPoolName("my_pool");
pds.setQueryTimeout(60); // 60 seconds to wait on query
...
```

4.4.8 タイムアウト・チェック間隔の設定

タイムアウト・チェック間隔プロパティは、タイムアウト・プロパティ(中止接続タイムアウト、TTL接続タイムアウトおよび非アクティブ接続タイムアウト)が適用される頻度を制御します。タイムアウトした接続は、タイムアウト・チェック・サイクルの実行時に回収されます。つまり、接続のタイムアウト時に、実際には接続がプールに回収されないことがあります。接続のタイムアウトと実際の接続の回収の間のラグ・タイムは、タイムアウト・チェック間隔の長さによっては非常に大きい場合があります。

タイムアウト・チェック間隔プロパティは秒単位です。デフォルト値は30に設定されます。次の例では、プロパティ・チェック間隔の構成を示します。

```
pds.setTimeoutCheckInterval(60);
```

関連項目:

Oracle Net Servicesの詳細は、『Oracle Database Net Services管理者ガイド』を参照してください。

4.5 UCPでの接続の獲得について

接続獲得機能を使用すると、接続プールが指定された使用可能な接続数に達したときに、指定された数の流用された接続を回収できます。この項では、次の概念について説明します。

- UCPでの接続の獲得の概要
- 獲得可能への接続の設定
- 獲得トリガー数の設定
- 獲得最大数の設定

4.5.1 UCPでの接続の獲得の概要

この機能は、プール内で一定数の接続を常に使用可能な状態にし、パフォーマンスの最大化に役立ちます。特に、アプリケーションで接続ハンドルをキャッシュする場合に便利です。通常、キャッシングはパフォーマンス上の理由から実行されます。これは、キャッシングにより、接続がトランザクションに参加するために必要となる状態の再初期化が最小限になるためです。

たとえば、接続はプールから流用され、必要なセッション状態で初期化された後、コンテキスト・オブジェクト内に保持されます。この方法で接続を保持することが、接続プールに使用可能な接続がなくなる原因になる可能性があります。接続獲得機能では、該当する場合に、流用された接続を回収して再利用できるようにします。

接続獲得は、HarvestableConnectionインタフェースを使用して制御し、接続獲得トリガー数と接続獲得最大数の2つのプール・プロパティを使用して構成または有効にします。接続獲得機能を実装するときは、これらのインタフェースおよびプロパティを併用します。

4.5.2 獲得可能への接続の設定

oracle.ucp.jdbc.HarvestableConnectionインタフェースのsetConnectionHarvestable(boolean)メソッドは、接続を獲得するかどうかを制御します。このメソッドは、接続獲得が有効である場合にロック・メカニズムとして使用されます。たとえば、トランザクション内で接続が使用されていて接続の獲得を禁止する場合、このメソッドを接続に対してfalseに設定します。トランザクションの完了後、このメソッドを接続に対してtrueに設定すると、必要に応じて接続を獲得できます。

> ノート:
>
> 接続獲得機能が有効である場合、デフォルトではすべての接続が獲得可能です。この機能が有効である場合、接続が獲得可能かどうかを明示的に制御するため、setConnectionHarvestable メソッドを常に使用する必要があります。

次の例では、接続獲得機能で接続を獲得しようとしても接続が獲得不可であることを示す、setConnectionHarvestableメソッドの使用例を示しています。

```
Connection conn = pds.getConnection();

((HarvestableConnection) conn).setConnectionHarvestable(false);
```

4.5.3 獲得トリガー数の設定

接続獲得トリガー数は、接続獲得をトリガーする使用可能な接続のしきい値を指定します。たとえば、接続獲得トリガー数を10に設定した場合、プール内の使用可能な接続数が10まで減少すると、接続獲得がトリガーされます。

値Integer.MAX_VALUE(デフォルトでは2147483647)は、接続獲得が無効であることを示します。デフォルト値はInteger.MAX_VALUEです。

次の例では、接続獲得トリガー数を構成して接続獲得を有効にしています。

```
pds.setConnectionHarvestTriggerCount(2);
```

4.5.4 獲得最大数の設定

接続獲得最大数プロパティは、獲得トリガー数に達した場合に、プールに返す必要がある流用された接続の数を指定します。実際に獲得される接続数は、0から接続獲得最大数の値までのいずれかです。最も長い間使用されていない接続から先に獲得されるため、非常にアクティブなユーザー・セッションが最大限に接続を保つことができます。

獲得最大数の値の範囲は、0から最大接続数プロパティの値までです。デフォルト値は1です。範囲外の値を指定すると、SQL例外がスローされます。

次の例では、接続獲得最大数の構成を示します。

```
pds.setConnectionHarvestMaxCount(5);
```

ノート:

- 接続獲得機能および中止接続タイムアウト機能が同時に有効になっている場合、タイムアウト処理では、獲得できないとして指定された接続を回収しません。
- 接続獲得機能および TTL 接続タイムアウト機能が同時に有効になっている場合、タイムアウト処理では、獲得できないとして指定された接続を回収します。

関連トピック

- 再利用可能な接続の動作の制御

4.6 UCPでのSQL文のキャッシングについて

この項では、次の項でUCPでSQL文をキャッシュする方法について説明します。

- UCPでの文キャッシングの概要
- UCPでの文キャッシングの有効化

4.6.1 UCPでの文キャッシングの概要

文キャッシングにより、文の処理はより効率的になります。文キャッシングでは、繰り返し使用される実行可能な文をキャッシングすることでパフォーマンスが向上し、プログラマがコンパイル済の文を明示的に再利用する必要がなくなります。また、繰り返されるカーソルの作成、繰り返される文の分析および作成によるオーバーヘッドを解消し、アプリケーションとデータベース間の通信のオーバーヘッドを削減します。文キャッシングと再利用は、アプリケーションに対して透過的です。各文キャッシュは、物理的な接続に関連付けられます。つまり、物理的な接続はそれぞれ独自の文キャッシュを保有します。

キャッシュされた文の一致条件は次のとおりです。

- 文のSQL文字列は、キャッシュ内のものと同一(大/小文字を区別)である必要があります。
- 文の種類は、キャッシュ内のものと同一(preparedまたはcallable)である必要があります。
- 文によって生成される結果セットのスクロール可能タイプは、キャッシュ内のものと同一(forward-onlyまたはscrollable)である必要があります。

文キャッシングは、JDBCドライバのベンダーによって異なる方法で実装および有効化されます。この項の説明は、OracleのJDBCドライバ固有のものです。他のベンダーのドライバでの文キャッシングは、コネクション・ファクトリで接続プロパティを設定することで構成できます。JDBCベンダーのドキュメントを参照して、文キャッシングがサポートされているかどうか、接続プロパティとして設定できるかどうかを確認してください。UCPでは、JDBCベンダーが文プーリングをサポートする場合は、文プーリングを有効にするためのJDBC 4.0(JDK16)APIがサポートされません。

関連トピック

- 接続プロパティの設定

4.6.2 UCPでの文キャッシングの有効化

最大文数プロパティは、接続ごとにキャッシュする文の数を指定します。このプロパティは、Oracle JDBCドライバにのみ適用できます。このプロパティを設定しない場合、または0に設定する場合、文キャッシングは無効になります。デフォルトでは、文キャッシングは無効です。文キャッシングを有効にすると、文キャッシュは、接続プールで保持される物理的な接続のそれぞれに関連付けられます。1つの文キャッシュは、すべての物理的な接続で共有されません。

次の例では、文キャッシングの有効化を示します。

```
pds.setMaxStatements(10);
```

文キャッシュ・サイズの決定

キャッシュ・サイズは、アプリケーションがデータベースに対して発行する個々の文の数に設定する必要があります。アプリケーションがデータベースに対して発行する文の数が不明な場合、JDBCパフォーマンス・メトリックを使用して文キャッシュ・サイズの決定に役立てます。

文キャッシュ・サイズのリソース問題

接続はそれぞれ独自の文キャッシュに関連付けられます。接続の文キャッシュ内に保持される文が、データベース・リソースを保持し続ける場合があります。オープンされた接続の数と各接続のキャッシュされた文の数の合計が、データベースで許容されるオープン・カーソルの限度を超える可能性があります。この問題は、キャッシュ内で許容される文の数を減らすか、データベースで許容されるオープン・カーソルの限度を増やすかすることで、回避できます。

5 UCPでの接続のラベル付け

この章では、以下のトピックについて説明します。

- UCPでの接続のラベル付けの概要
- UCPでのラベリング・コールバックの実装
- UCPでの接続ラベルの適用
- UCPからのラベル付けされた接続の流用
- UCPでの不一致ラベルのチェック
- DRCPを使用したUCPの統合
- UCPでの接続ラベルの削除

5.1 UCPでの接続のラベル付けの概要

多くの場合、アプリケーションは、接続プールから取得した接続を使用する前に初期化します。初期化は一様ではなく、アプリケーション・コード内でメソッド・コールを必要とする単純な状態の再初期化や、ネットワーク上でのラウンドトリップを必要とするデータベース操作などがあります。このような初期化のコストは非常に高い場合があります。

接続のラベル付けを使用すると、アプリケーションが接続に任意の名前/値のペアを付けることができます。アプリケーションは、必要なラベルが付いた接続を接続プールにリクエストできます。特定のラベルと特定の接続状態を関連付けることで、すでに初期化されている接続をプールから取得し、再初期化の時間とコストを回避できます。接続ラベリング機能は、ユーザー定義キーまたは値に意味を与えません。ユーザー定義キーおよび値の意味は、アプリケーションでのみ定義されます。

接続ラベリングの例には、ロール、NLS言語設定、トランザクション分離レベル、ストアド・プロシージャ・コール、またはリソースによる処理の実行の前に接続上で必要となるその他のコストのかかる状態の初期化があります。

接続ラベリングはアプリケーション駆動型で、2つのインタフェースを使用する必要があります。
oracle.ucp.jdbc.LabelableConnectionインタフェースは、接続ラベルの適用および削除と、接続に設定されているラベルの取得に使用されます。oracle.ucp.ConnectionLabelingCallbackインタフェースは、リクエストされたラベルが付いた接続がすでに存在するかどうかを判断するラベリング・コールバックの作成に使用されます。接続が存在しない場合、このインタフェースを使用して、現行の接続を必要に応じて構成できます。これらのインタフェースのメソッドについては、この章全体を通して詳しく説明します。

5.2 UCPでのラベリング・コールバックの実装

UCPは、単一のラベルまたは複数のラベルの使用に関係なく、データベース常駐接続プーリング(DRCP)・タグ付けインフラストラクチャを使用して、UCPのラベリングをサポートします。ただし、UCPのみのかわりにUCPおよびDRCPの組合せを使用する場合、複数のラベルを使用した動作は少し異なる可能性があります。

このセクションのトピックは次のとおりです:

- UCPでのラベリング・コールバックの使用時期
- UCPでのラベリング・コールバックの作成
- UCPでのラベリング・コールバックの登録
- UCPでのラベリング・コールバックの削除

関連項目:

「DRCPを使用したUCPの統合」

5.2.1 UCPでのラベリング・コールバックの使用時期

ラベリング・コールバックは、接続プールでラベル付けされた接続を選択する方法を定義するために使用され、アプリケーションに戻す前に、選択された接続の構成ができます。接続ラベリング機能を使用するアプリケーションは、コールバックを実装する必要があります。

ラベリング・コールバックは、ラベル付けされた接続がリクエストされていながら、リクエストされたラベルと一致する接続がプールにない場合に使用されます。コールバックは、リクエストされたラベルと一致するように再構成するために必要となる作業量が最も少ない接続を特定した後、アプリケーションに戻す前に接続ラベルを更新できるようにします。この節では、以下のトピックについて説明します。

5.2.2 UCPでのラベリング・コールバックの作成

ラベリング・コールバックを作成するには、アプリケーションにoracle.ucp.ConnectionLabelingCallbackインタフェースを実装します。コールバックは、接続プールごとに1つ作成されます。インタフェースでは、次の2つのメソッドが提供されています。

- costメソッド
- configureメソッド

costメソッド

このメソッドは、ラベルマッチングの相違を考慮に入れて、接続の構成コストを見積ります。接続リクエストが発生すると、接続プールはこのメソッドを使用して、最も構成コストが低い接続を選択します。

```
public int cost(Properties requestedLabels, Properties currentLabels);
```

configureメソッド

このメソッドは、アプリケーションに戻す前に、選択された接続に対して接続プールによってコールされます。接続の状態を設定し、接続に対してラベルの適用または削除を行うために、このメソッドが使用されます。

```
public boolean configure(Properties requestedLabels, Connection conn);
```

接続プール内で使用可能な各接続に対して繰り返します。接続ごとにcostメソッドをコールします。costメソッドの結果は、接続を必要な状態に再構成するために必要なコストの見積りを表すintegerです。値が大きいほど、接続の再構成にはコストがかかります。接続プールは、常に最も低いコスト値の接続を戻します。アルゴリズムは次のとおりです。

- costメソッドがある接続について0を戻した場合、その接続が適合となります。接続プールは、検出された接続に対してconfigureメソッドをコールせず、その接続をそのまま戻します。
- costメソッドが0より大きい値を戻した場合、コスト値が0の接続を検出するか、使用可能な接続がなくなるまで繰り返します。
- すべての使用可能な接続に対して繰り返して、接続の最低コストがInteger.MAX_VALUE(デフォルトでは2147483647)となった場合、プール内に接続リクエストを満たす接続はありません。プールは新しい接続を作成して戻します。プールが最大プール・サイズに達している(新しい接続を作成できない)場合は、SQL例外をスローするか、接続待機タイムアウト属性が指定されていれば待機するかのいずれかです。
- すべての使用可能な接続に対して繰り返して、接続の最低コストがInteger.MAX_VALUEよりも低い場合、その接続に対してconfigureメソッドをコールし、その接続を戻します。複数の接続がInteger.MAX_VALUEを下回る場合は、最低コストの接続を戻します。

ノート:

コスト 0 は、requestedLabels と currentLabels が等しいという意味ではありません。

5.2.2.1 UCPでのラベリング・コールバックの例

次の例では、costとconfigureの両メソッドを実装する単純なラベリング・コールバックの実装を示します。このコールバックは、特定のトランザクション分離レベルで初期化されるラベル付けされた接続の検出に使用されます。

```
class MyConnectionLabelingCallback
  implements ConnectionLabelingCallback {

  public MyConnectionLabelingCallback()
  {
  }

  public int cost(Properties reqLabels, Properties currentLabels)
  {
    // Case 1: exact match
    if (reqLabels.equals(currentLabels))
    {
      System.out.println("## Exact match found!! ##");
      return 0;
    }

    // Case 2: some labels match with no unmatched labels
    String iso1 = (String) reqLabels.get("TRANSACTION_ISOLATION");
    String iso2 = (String) currentLabels.get("TRANSACTION_ISOLATION");
    boolean match =
      (iso1 != null && iso2 != null && iso1.equalsIgnoreCase(iso2));
    Set rKeys = reqLabels.keySet();
    Set cKeys = currentLabels.keySet();
    if (match && rKeys.containsAll(cKeys))
    {
```

```
      System.out.println("## Partial match found!! ##");
      return 10;
    }

    // No label matches to application's preference.
    // Do not choose this connection.
    System.out.println("## No match found!! ##");
    return Integer.MAX_VALUE;
  }

  public boolean configure(Properties reqLabels, Object conn)
  {
    try
    {
      String isoStr = (String) reqLabels.get("TRANSACTION_ISOLATION");
      ((Connection)conn).setTransactionIsolation(Integer.valueOf(isoStr));
      LabelableConnection lconn = (LabelableConnection) conn;

      // Find the unmatched labels on this connection
      Properties unmatchedLabels =
       lconn.getUnmatchedConnectionLabels(reqLabels);

      // Apply each label <key,value> in unmatchedLabels to conn
      for (Map.Entry<Object, Object> label : unmatchedLabels.entrySet())
      {
        String key = (String) label.getKey();
        String value = (String) label.getValue();
        lconn.applyConnectionLabel(key, value);
      }
    }
    catch (Exception exc)
    {
      return false;
    }
    return true;
  }
}
```

5.2.3 UCPでのラベリング・コールバックの登録

プール対応のデータソースは、ラベリング・コールバックを登録するための registerConnectionLabelingCallback(ConnectionLabelingCallback callback)メソッドを備えています。1つの接続プールに登録できるコールバックは1つのみです。次の例では、MyConnectionLabelingCallbackクラスに実装されているラベリング・コールバックの登録を示します。

```
MyConnectionLabelingCallback callback = new MyConnectionLabelingCallback();
pds.registerConnectionLabelingCallback( callback );
```

5.2.4 UCPでのラベリング・コールバックの削除

プール対応のデータソースは、ラベリング・コールバックを削除するためのremoveConnectionLabelingCallback()メソッドを備えています。次の例では、ラベリング・コールバックの削除を示します。

```
pds.removeConnectionLabelingCallback( callback );
```

5.3 DRCPを使用したUCPの統合

DRCPは、重みのない単一のラベルである接続のタグ付けをネイティブにサポートします。そのため、DRCPでUCPを使用する場合、単一のラベルを使用したラベル付けは透過的に動作します。

ノート:

UCP の最大プール・サイズは、DRCP のサイズより小さくしておくことをお薦めします。UCP プール・サイズが DRCP サイズより大きい場合は、setValidateConnectionOnBorrow プロパティを off に設定する必要があります。そうしないと、UCP は、その時点では DRCP に関連付けられていない接続の無効化とクローズを維持し、新しい接続の作成を継続します。

関連項目:

UCPでの接続の検証の概要

複数のラベルのUCP接続は動作しますが、次の動作変更があります。

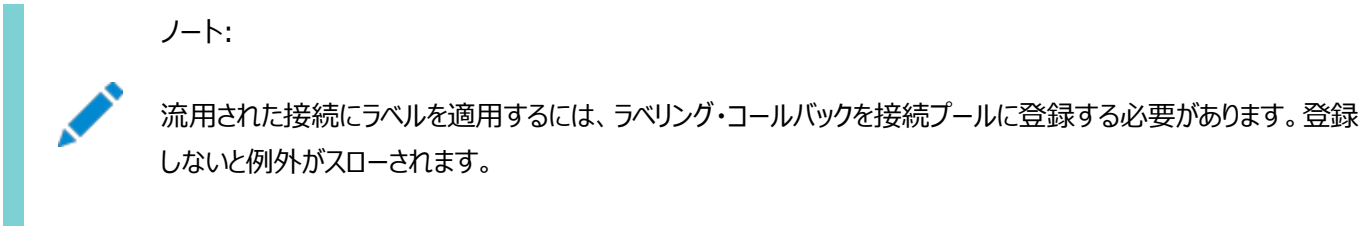
- 接続ラベリングを使用してDRCPでUCPを使用する場合、ConnectionLabelingCallback APIのcostメソッドは起動しません。
- UCPは、DRCP構成なしではなくDRCP構成ありでConnectionLabelingCallback APIのconfigureメソッドを起動できます。

関連項目:

DRCPの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください。

5.4 UCPでの接続ラベルの適用

ラベルは、LabelableConnectionインタフェースのapplyConnectionLabelメソッドを使用して、流用された接続に適用されます。通常、このメソッドは、ラベリング・コールバックのconfigureメソッドからコールされます。任意の数の接続ラベルを流用された接続に累積的に適用できます。ラベルが接続に適用されるたびに、指定されたキー/値のペアが、すでに接続に適用されているラベルのコレクションに追加されます。最後に適用された値のみがどのキーに対しても保持されます。



ノート:

流用された接続にラベルを適用するには、ラベリング・コールバックを接続プールに登録する必要があります。登録しないと例外がスローされます。

次の例では、トランザクション分離レベルで接続を初期化した後、ラベルを接続に適用しています。

```
String pname = "property1";
String pvalue = "value";
Connection conn = pds.getConnection();

// initialize the connection as required.

conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);

((LabelableConnection) conn).applyConnectionLabel(pname, pvalue);
```

特定のキーを適用済の接続ラベルのセットから削除するには、削除するキーおよびnull値のラベルを適用します。この方法は、接続ラベルのセットから特定のキー/値ペアをクリアするために使用できます。

関連トピック

- UCPでのラベリング・コールバックの実装

5.5 UCPからのラベル付けされた接続の流用

プール対応のデータソースは、ラベル付けされた接続をプールから流用するために使用される2つのgetConnectionメソッドを備えています。これらのメソッドを次に示します。

```
public Connection getConnection(java.util.Properties labels )
   throws SQLException;

public Connection getConnection( String user, String password,
                                 java.util.Properties labels )
   throws SQLException;
```

これらのメソッドには、getConnectionメソッドにPropertiesオブジェクトとして渡すラベルが必要です。次の例では、property1, valueというラベルが付いた接続の取得を示します。

```
String pname = "property1";
String pvalue = "value";
Properties label = new Properties();
label.setProperty(pname, pvalue);

Connection conn = pds.getConnection(label);
```

5.6 UCPでの不一致ラベルのチェック

1つの接続は複数のラベルを保有できます。各ラベルは目的とする条件に基づいて一意に接続を識別します。getUnmatchedConnectionLabelsメソッドは、リクエストされたラベルから、どの接続ラベルが一致してどの接続ラベルが一致しなかったかを検証するために使用されます。このメソッドは、複数のラベルを持つ接続が接続プールから流用された後に使用され、通常、ラベリング・コールバックで使用されます。次の例では、不一致ラベルのチェックを示します。

```
String pname = "property1";
String pvalue = "value";
Properties label = new Properties();
label.setProperty(pname, pvalue);

Connecion conn = pds.getConnection(label);
Properties unmatched = ((LabelableConnection)
   connection).getUnmatchedConnectionLabels (label);
```

5.7 UCPでの接続ラベルの削除

removeConnectionLabelメソッドは、接続からラベルを削除するために使用されます。このメソッドは、ラベル付けされた接続が接続プールから流用された後に使用されます。次の例では、接続ラベルの削除を示します。

```
String pname = "property1";
String pvalue = "value";
Properties label = new Properties();
label.setProperty(pname, pvalue);
Connection conn = pds.getConnection(label);
((LabelableConnection) conn).removeConnectionLabel(pname);
```

6 再利用可能な接続の動作の制御

この章では、次のインタフェースについて説明します。

- AbandonedConnectionTimeoutCallbackインタフェース
- TimeToLiveConnectionTimeoutCallbackインタフェース

6.1 AbandonedConnectionTimeoutCallbackインタフェース

AbandonedConnectionTimeoutCallbackコールバック・インタフェースは中止接続タイムアウト機能に使用します。この機能により、アプリケーションは中止接続をカスタマイズ処理できます。コールバック・オブジェクトは論理接続プロキシのいずれかを使用するか、または各プール接続に登録されます。これにより、特定の接続が中止されたとプールによってみなされた場合、アプリケーションはカスタマイズ処理を実行できます。流用された接続が中止されたとユニバーサル接続プールによってみなされた場合、handleTimedOutConnectionメソッドが起動されます。アプリケーションは接続に対して次の操作のいずれかを実行できます。

- プール処理プロセスの完全な上書き
- 追加の処理アクションの起動
- デフォルトのプール処理の想定

JDBCアプリケーションは、handleTimedOutConnectionメソッド内でcancel、closeおよびrollbackメソッドを中止接続に対して起動できます。

ノート:

同じ接続に複数の AbandonedConnectionTimeoutCallback インタフェースを登録しようとすると、例外が発生します。この例外は、プール・レイヤーにおける UniversalConnectionPoolException か、JDBC、JCA などの UCP アダプタのタイプに固有の java.sql.SQLException のいずれかです。

6.2 TimeToLiveConnectionTimeoutCallbackインタフェース

TimeToLiveConnectionTimeoutCallbackコールバック・インタフェースはTTL接続タイムアウト機能に使用します。これにより、アプリケーションはTTLタイムアウト接続をカスタマイズ処理できます。

コールバック・オブジェクトは論理接続プロキシのいずれかを使用するか、または各プール接続に登録されます。これにより、特定のTTL接続がタイムアウトになった場合、アプリケーションはカスタマイズ処理を実行できます。

流用された接続がTTLタイムアウトになったとユニバーサル接続プールによって検出された場合、handleTimedOutConnectionメソッドが起動されます。アプリケーションは接続に対して次の操作のいずれかを実行できます。

- プール処理プロセスの完全な上書き
- 追加の処理アクションの起動
- デフォルトのプール処理の想定

JDBCアプリケーションは、handleTimedOutConnectionメソッド内でcancel、closeおよびrollbackメソッドを中止接続に対して起動できます。

ノート:

同じ接続に複数の TimeToLiveConnectionTimeoutCallback インタフェースを登録すると、例外が発生します。この例外は、プール・レイヤーにおける UniversalConnectionPoolException であるか、JDBC、JCA などの UCP アダプタのタイプに固有の java.sql.SQLException です。

7 接続プール・マネージャの使用方法

この章の内容は次のとおりです。

- UCPマネージャの使用の概要
- JMXベース管理の概要

7.1 UCPマネージャの使用の概要

ユニバーサル接続プール(UCP)マネージャはUCPインスタンスを作成し維持します。新しいプールが作成されるたびに、プール・インスタンスがプール・マネージャに登録されます。この項の内容は次のとおりです。

- 接続プール・マネージャについて
- UCPの接続プール・マネージャの作成
- 接続のライフ・サイクル状態
- ユニバーサル接続プールのメンテナンス

7.1.1 接続プール・マネージャについて

アプリケーションは、接続プール・マネージャを使用してUCP JDBC接続プールを明示的に作成および管理します。アプリケーションでマネージャを使用するのは、接続プールの作成、起動、停止、破棄などのライフ・サイクルを完全に制御できるためです。また、マネージャを使用して、接続プール内の接続のリフレッシュ、リサイクル、パージなどの定期的なメンテナンスを実行します。最後に、管理ツールおよびコンソールの集中統合ポイントにできることから、接続プール・マネージャを使用します。

7.1.2 UCPの接続プール・マネージャの作成

接続プール・マネージャは、oracle.ucp.adminパッケージにあるUniversalConnectionPoolManagerインタフェースのインスタンスです。マネージャは、JVMごとに複数の接続プールを管理するために使用されるシングルトン・インスタンスです。このインタフェースは、接続プール・マネージャとやりとりするためのメソッドを備えています。UCPには、接続プール・マネージャ・インスタンスの取得に使用される実装があります。次の例では、その実装を使用した接続プール・マネージャ・インスタンスの作成を示します。

```
UniversalConnectionPoolManager mgr = UniversalConnectionPoolManagerImpl.
getUniversalConnectionPoolManager();
```

7.1.3 接続のライフ・サイクル状態

アプリケーションは、接続プール・マネージャを使用して接続プールのライフ・サイクルを明示的に制御します。マネージャは、接続プールの作成、起動、停止および破棄に使用されます。ライフ・サイクル・メソッドは、UniversalConnectionPoolManagerインタフェースの一部として含まれます。

ライフ・サイクルの状態について

接続プールのライフ・サイクルの状態は、接続プールに対して実行できるマネージャ操作に影響を与えます。プールのライフ・サイクルを明示的に制御するアプリケーションでは、プールが適切な状態にある場合にのみ、マネージャの操作が使用されるようにする必要があります。ライフ・サイクルの制約については、この項全体を通して説明します。

プールのライフ・サイクルの状態を次に示します。

- 起動中: 接続プールの起動メソッドがコールされ、起動中であることを示します。
- 実行中: 接続プールが起動され、接続の割当て準備ができていることを示します。
- 停止中: 接続プールが停止中であることを示します。
- 停止済: 接続プールが停止していることを示します。
- 失敗: 起動、停止または実行中に、接続プールで障害が発生したことを示します。

7.1.3.1 接続プールの作成

接続マネージャのCreateConnectionPoolメソッドは、接続プールを作成および登録します。マネージャは、接続プール・アダプタを使用してプールを作成し、プール対応のデータソースを利用してプール・プロパティを構成します。アプリケーションで接続プールを暗黙的に起動してから、createConnectionPoolメソッドを使用して明示的に同じ名前のプールを作成しないでください。

次の例では、マネージャを使用した接続プール・インスタンスの作成を示します。

```
UniversalConnectionPoolManager mgr = UniversalConnectionPoolManagerImpl.
getUniversalConnectionPoolManager();

PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionPoolName("mgr_pool");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin:@//localhost:1521/XE");
pds.setUser("<user>");
pds.setPassword("<password>");

mgr.createConnectionPool((UniversalConnectionPoolAdapter)pds);
```

管理のためマネージャを使用してプールを作成する必要はありません。暗黙的に作成され(プール対応のデータソースの使用時に自動的に作成され)、プール名を使用して構成されたプールも、プール・マネージャによって自動的に登録および管理されます。暗黙的なプールの作成をお薦めします。

プール命名規則

接続プール名は、構成の一部として定義する必要があります。プール名は、マネージャとやりとりする場合に特定のプールを参照する方法を提供します。接続プール名は一意である必要があり、複数の接続プールで使用することはできません。同じ名前の接続プールがすでに存在する場合、マネージャはpool already exists例外をスローします。

JBossとの互換性

JBossユーザーは、oracle.ucp.destroyOnReload JVMシステム・プロパティをtrueに設定して、JBoss固有のサイレント・リロード機能を使用できます。oracle.ucp.destroyOnReloadプロパティがtrueに設定されている場合、JBoss固有の動作では、同じ名前の新しいプール・インスタンスを作成する前に、古いプール・インスタンスを自動的に破棄します。このシステム・プロパティが設定されていないか、falseに設定されている場合、UCPはpool already exists例外をスローします。

7.1.3.2 接続プールの起動

マネージャのstartConnectionPoolメソッドで接続プールが起動されます。起動するプールを特定するために、プール名がパラメータとして使用されます。プール名は、プール・プロパティとしてプール対応のデータソースで定義されます。

次の例では、接続プールの起動を示します。

```
mgr.startConnectionPool("mgr_pool");
```

アプリケーションでは、常にマネージャのcreateConnectionPoolメソッドを使用して接続プールを作成してからプールを起動する必要があります。また、アプリケーションですでに起動されているプールを起動しようとした場合、つまりプールが停止済または失敗以外のステータスである場合は、ライフ・サイクルの状態例外が発生します。

7.1.3.3 接続プールの停止

マネージャのstopConnectionPoolメソッドで接続プールが停止されます。停止するプールを特定するために、プール名がパラメータとして使用されます。プール名は、プール・プロパティとしてプール対応のデータソースで定義されます。接続プールを停止すると、すべての使用可能な接続および流用された接続はクローズされます。

次の例では、接続プールの停止を示します。

```
mgr.stopConnectionPool("mgr_pool");
```

アプリケーションでは、マネージャを使用して暗黙的または明示的に起動された接続プールを停止できます。存在しないプールを停止しようとした場合、またはプールが起動済または起動中以外の状態である場合は、エラーが発生します。

7.1.3.4 接続プールの破棄

マネージャのdestroyConnectionPoolメソッドで接続プールは停止され、接続プール・マネージャから削除されます。破棄するプールを特定するために、プール名がパラメータとして使用されます。プール名は、プール・プロパティとしてプール対応のデータソースで定義されます。

次の例では、接続プールの破棄を示します。

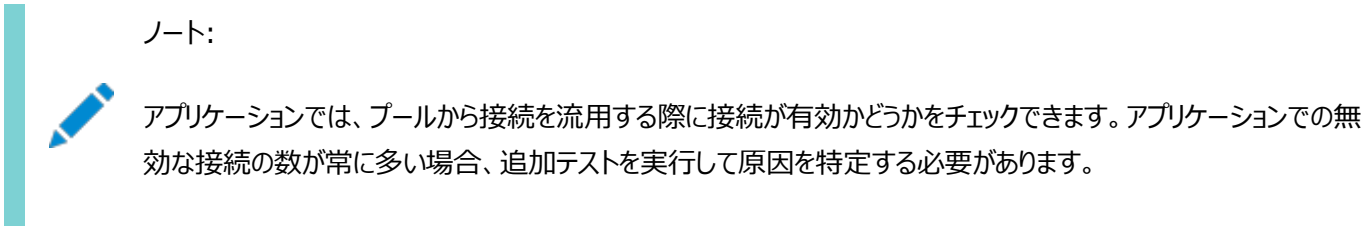
```
mgr.destroyConnectionPool("mgr_pool");
```

アプリケーションでは、破棄された接続プールは起動できません。新しい接続プールを明示的に作成および起動する必要があります。

7.1.4 ユニバーサル接続プールのメンテナンス

アプリケーションでは、接続プール・マネージャを使用して接続プールでメンテナンスを実行します。メンテナンスには、接続プールのリフレッシュ、リサイクルおよびパージがあります。メンテナンス・メソッドは、UniversalConnectionPoolManagerインタフェースの一部として含まれます。

通常、メンテナンスは無効な接続を削除および置換し、有効な接続の高い可用性を確保するために実行します。一般に、無効な接続はデータベースへの接続に使用できませんが、プールで維持され続けます。このような接続はシステム・リソースを浪費し、プールの最大接続数制限に直接影響を及ぼします。最終的には、多すぎる無効な接続によってアプリケーション・パフォーマンスが悪化します。



ノート:

アプリケーションでは、プールから接続を流用する際に接続が有効かどうかをチェックできます。アプリケーションでの無効な接続の数が常に多い場合、追加テストを実行して原因を特定する必要があります。

関連トピック

- UCPでの接続の検証の概要

7.1.4.1 接続プールのリフレッシュ

接続プールをリフレッシュすると、プール内のすべての接続が新しい接続に置き換えられます。現在流用されている接続には削除マークが付けられ、接続がプールに返された後にリフレッシュされます。マネージャのrefreshConnectionPoolメソッドで接続プールがリフレッシュされます。リフレッシュするプールを特定するために、プール名がパラメータとして使用されます。プール名は、プール・プロパティとしてプール対応のデータソースで定義されます。

次の例では、接続プールのリフレッシュを示します。

```
mgr.refreshConnectionPool("mgr_pool");
```

7.1.4.2 接続プールのリサイクル

接続プールをリサイクルすると、プール内の無効な接続のみが新しい接続に置き換えられ、流用された接続は置き換えられません。マネージャのrecycleConnectionPoolメソッドで接続プールがリサイクルされます。リサイクルするプールを特定するために、プール名がパラメータとして使用されます。プール名は、プール・プロパティとしてプール対応のデータソースで定義されます。

Oracle以外のドライバを使用する場合は、setSQLForValidateConnectionプロパティを設定する必要があります。UCPではこのプロパティを使用して、接続をリサイクルする前に接続が有効かどうかを判断します。

次の例では、接続プールのリサイクルを示します。

```
mgr.recycleConnectionPool("mgr_pool");
```

関連トピック

- UCPでの接続の検証の概要

7.1.4.3 接続プールのパージ

接続プールをパージすると、すべての接続(使用可能な接続および流用された接続)が接続プールから削除され、接続プールを空の状態にします。後続の接続リクエストにより、新しい接続が作成されます。マネージャのpurgeConnectionPoolメソッドで接続プールがパージされます。パージするプールを特定するために、プール名がパラメータとして使用されます。プール名は、プール・プロパティとしてプール対応のデータソースで定義されます。

次の例では、接続プールのパージを示します。

```
mgr.purgeConnectionPool("mgr_pool");
```

ノート:

minPoolSize や initialPoolSize などの接続プールのプロパティは、接続プールのパージ後に実行できないことがあります。

7.2 UCPでのJMXベース管理の概要

JMX(Java Management Extensions)は、アプリケーション、システム・オブジェクト、デバイス、サービス指向のネットワーク、JVM(Java仮想マシン)を管理および監視するツールを提供するJavaテクノロジです。JMXでは、特定のリソースは、MBean(マネージドBean)と呼ばれる1つ以上のJavaオブジェクトで構成されます。MBeanは、MBeanインタフェースおよびクラスで構成されます。MBeanインタフェースは、すべての公開された属性および操作のメソッドを示します。クラスはこのインタフェースを実装し、構成されたリソースの機能を提供します。

MBeanは、管理エージェントとして機能するコア・マネージド・オブジェクト・サーバー(MBeanサーバーと呼ばれる)に登録され、Javaプログラミング言語に対応するほとんどのデバイスで実行できます。JMXエージェントは、MBeanが登録されているMBeanサーバーと、MBeanを処理する一連のサービスで構成されます。

関連項目:

- https://docs.oracle.com/javase/tutorial/jmx/mbeans/standard.html
- 『Oracle Universal Connection Pool Java API Reference』

UCPには、プール管理サポートのために次の2つのMBeanがあります。

- UniversalConnectionPoolManagerMBean
- UniversalConnectionPoolMBean

ノート:

UniversalConnectionPoolManager.isJmxEnabled メソッドが true を返した場合にのみ、すべての MBean 属性および操作を使用できます。このフラグのデフォルト値は true です。このデフォルト値は、UniversalConnectionPoolManager.setJmxEnabled メソッドをコールすることによって変更できます。MBeanServer を使用できない場合、自動的に false が jmxFlag に設定されます。

7.2.1 UniversalConnectionPoolManagerMBean

UniversalConnectionPoolManagerMBeanは、従来の接続プール・マネージャの機能がすべて含まれているマネージャMBeanです。UniversalConnectionPoolManagerMBeanは次の機能を提供します。

- プールMBeanの登録および登録解除
- プールの起動、停止、リフレッシュなどのプール管理操作
- DMS統計の起動と停止
- ロギング

7.2.2 UniversalConnectionPoolMBean

UniversalConnectionPoolMBeanは、プールのプロパティおよびプールの統計情報の動的な構成に対応します。

UniversalConnectionPoolMBeanは次の機能を提供します。

- サイズ、タイムアウトなどのプールのプロパティの属性の構成
- プールのリフレッシュ、リサイクルなどのプール管理操作
- プールの統計情報およびライフ・サイクルの状態の監視

8 マルチテナント・データソースの共有プール・サポート

Oracle Database 12cリリース2(12.2.0.1)以降、マルチテナント・データソースの複数のデータソースでは、UCPで接続の共通プールを共有し、必要に応じて共通接続プールの接続を再利用できます。この項では、新しい共有プール機能に関連する次の概念について説明します。

ノート:

- JDBC Thin ドライバのみ、JDBC OCI ドライバではなく共有プール機能をサポートします。
- この機能を使用するには、XML 構成ファイルを使用する必要があります。

- 共有プール・サポートの概要
- 共有プールをサポートするための前提条件
- 共有プールの構成
- 共有プール・サポートのAPI
- 共有プールのサンプルのXML構成ファイル

関連トピック

- 共有プールのサンプルのXML構成ファイル

8.1 共有プール・サポートの概要

UCPでは、プール・インスタンスはデータソースと1対1のマッピングがあります。同じデータベースおよびサービスへの接続を内部的に作成してキャッシュしても、各データソースは固有の接続プール・インスタンスを作成するため、そのインスタンスは別のデータソースによってアクセスできず、共有されません。このアーキテクチャでは、多くの分離された接続プールが作成され、データベースが特定の数の接続のみスケール・アップできるため、スケーラビリティの問題が発生します。

Oracle Database 12cリリース2(12.2.0.1)以降、UCPは、同じ接続プールを共有する同じデータベースに接続された複数のデータソースをサポートします。この共通接続プールは共有プールと呼ばれます。共有プールは、Oracle Databaseマルチテナント環境のマルチテナントJavaアプリケーションのスケーラビルなデプロイメントのためにシステム・リソースを最適化します。各データソースで均一でない負荷が存在する場合、この機能はさらに柔軟性を提供します。データソースごとの個別のプールが作成される場合、アイドル状態の接続プールから負荷状態にアイドル状態のリソースを移動できません。ただし、共有プールを使用する場合、データソース間の接続を共有して再利用し、効率的な方法で接続を利用できます。そのため、この機能により、データベース接続の合計数が減り、データベース・サーバーでのリソース使用、診断能力、管理性およびスケーリングが向上します。

この機能を実装できる2つのシナリオは、次のとおりです。

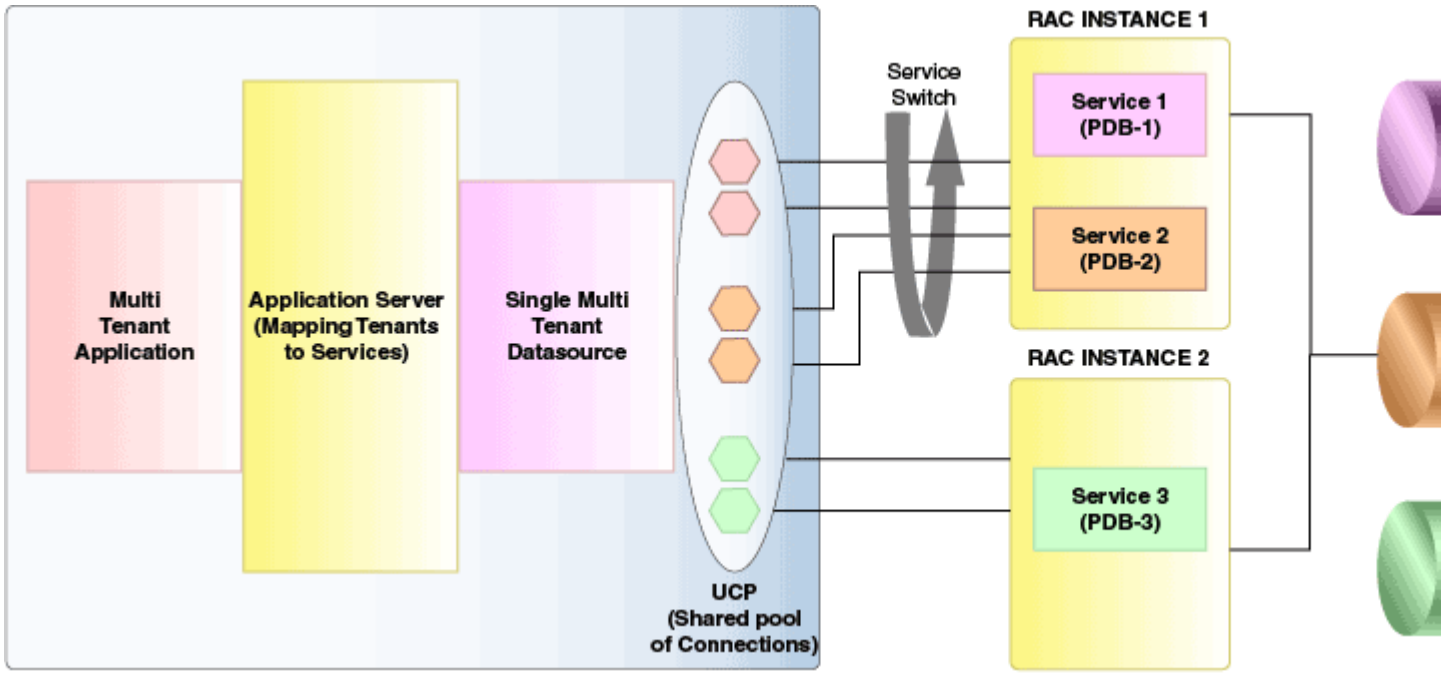
- 共有プールを使用した単一マルチテナント・データソース
- 共有プールを使用したテナントごとの1つのデータソース

共有プールを使用した単一マルチテナント・データソース

この構成を使用する場合、次の図に示すように、複数のテナントが共通データソースおよび共通プールを使用して、各テナント

に適用される異なるサービスの接続を提供します。

図8-1 共有プールを使用した単一マルチテナント・データソース



次のコードでは、この機能の動作方法について説明します。

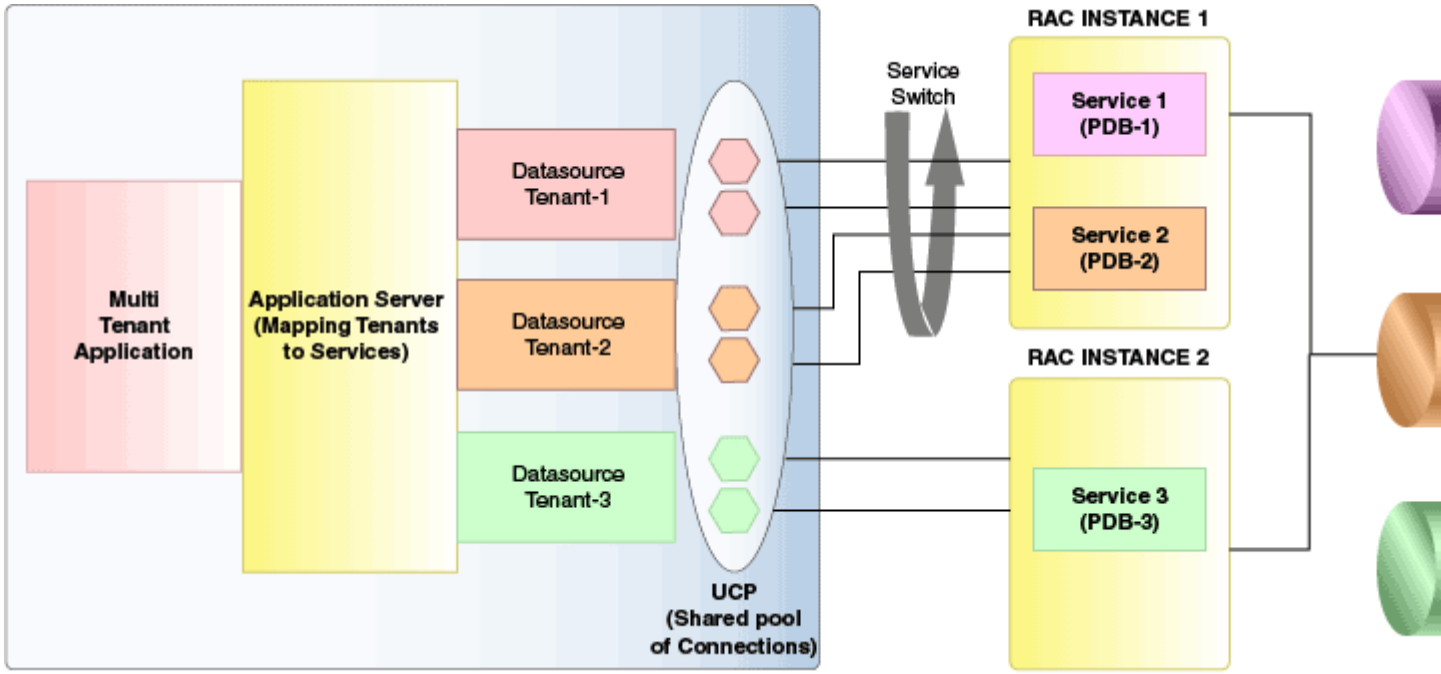
```
PoolDataSource multiTenantDS = PoolDataSourceFactory.getPoolDataSource();

//common user for the CDB
multiTenantDS.setUser("c##common_user");
multiTenantDS.setPassword("password");

//Points to the root service of the CDB
multiTenantDS.setURL("jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)"
    + "(HOST=myhost)(PORT=5521))(CONNECT_DATA=(SERVICE_NAME=root.oracle.com)))");

// password enabled role for tenant-1
Properties tenant1Roles = new Properties();
tenant1Roles.put("tenant1-role", "tenant1-password");

//Create Connection to Tenant-1 and apply the tenant specific PDB roles.
Connection tenant1Connection =
    multiTenantDS.createConnectionBuilder()
                 .serviceName("tenant1Svc.oracle.com")
                 .pdbRoles(tenant1Roles)
                 .build();

// password enabled role for tenant-2
Properties tenant2Roles = new Properties();
tenant1Roles.put("tenant2-role", "tenant2-password");

//Create Connection to Tenant-2 and apply the tenant specific PDB roles.
Connection tenant2Connection =
    multiTenantDS.createConnectionBuilder()
                 .serviceName("tenant2Svc.oracle.com")
                 .pdbRoles(tenant2Roles)
                 .build();
```

新しいAPIの詳細は、「共有プール・サポートのUCP API」を参照してください。

共有プールを使用したテナントごとの1つのデータソース

この構成を使用する場合、マルチテナント・アプリケーションには、テナントごとの個別のデータソースおよび接続のための共通共有プールがあります。これにより、次の図に示すように、テナント固有のサービス情報で構成されて共通プールを共有する個別のデータソースが発生します。

図8-2 共有プールを使用したテナントごとの1つのデータソース



次のコードでは、この機能の動作方法について説明します。

```
    // UCP XML configuration file path in case of Unix
    String file_URI = "file:/user/app/sharedpool/initial-shared-pool-config.xml";

    // UCP XML configuration file path in case of Windows
    String file_URI = "file:/D:/user/app/sharedpool/initial-shared-pool-config.xml";

    // Java system property to specify XML configuration file location
    System.setProperty("oracle.ucp.jdbc.xmlConfigFile",<file_URI>);

    // Get the datasource instance, named as "pds1" in XML configuration file(initial-shared-pool-
config.xml)
    PoolDataSource pds1 = PoolDataSourceFactory.getPoolDataSource("pds1");
    Connection pds1Conn = pds1.getConnection();

    // Get the datasource instance, named as "pds2" in XML configuration file(initial-shared-pool-
config.xml)
    PoolDataSource pds2 = PoolDataSourceFactory.getPoolDataSource("pds2");
    Connection pds2Conn = pds2.getConnection();

    // Reconfigure datasource(pds1) using the new properties
    Properties newProps = new Properties();
    newProps.put("serviceName", <newServiceName>);
    pds1.reconfigureDataSource(newProps);

    // Configure a new datasource(pds3) to running pool using the new data source properties
    Properties dataSourceProps = new Properties();
    dataSourceProps.put("serviceName", <serviceName>);
    dataSourceProps.put("connectionPoolName", <poolName>);
    dataSourceProps.put("dataSourceName", <dataSourceName>);
```

```
    PoolDataSource pds3 = PoolDataSourceFactory.getPoolDataSource(dataSourceProps);

    // Reconfigure connection pool("pool1") using the new properties

    Properties newPoolProps = new Properties();
    newPoolProps.put("initialPoolSize", <newInitialPoolSizeValue>);
    newPoolProps.put("maxPoolSize", <newMaxPoolSizeValue>);
    UniversalConnectionPoolManager ucpMgr =
UniversalConnectionPoolManagerImpl.getUniversalConnectionPoolManager();
    ucpMgr.reconfigureConnectionPool("pool1", newPoolProps);
```

ノート:

- UCP は、この機能を実装するためにサービス・スイッチを使用します。ただし、共有プールのサービス・スイッチは、同種のサービスでのみサポートされます。共有プールの異機種間サービス(Transaction Guard およびアプリケーション・コンティニュイティなどのサービス属性の点からの異種性)はサポートされていません。
- コード・スニペットで使用されている XML 構成ファイルは、共有プール・サポートに必要な XML 構成ファイルに関する項を参照してください。

8.2 共有プールをサポートするための前提条件

共有プールを使用するマルチテナント・データソースの前提条件は、次のとおりです。

- XML構成ファイルを使用して、共有プールの初期構成を提供する必要があります。システム・プロパティoracle.ucp.jdbc.xmlConfigFileを使用して、UCPの初期XML構成ファイルを指定できます。初期XML構成ファイルの場所は、URIとして指定する必要があります。たとえば、file:/user_directory/ucp.xmlなどです。

  configuration.xsdスキーマ・ファイルは、参照用のucp.jarファイルに含まれます。UCP XML構成ファイルの作成中に、このファイルを参照します。
- 共有プールの再構成中に、再構成APIを介して更新されたプール・プロパティが提供されます。
- 共有プールおよび個別のテナント・データソース固有サービスに使用されるサービスのアプリケーション・サービスを常に使用します。管理サービスまたはデフォルトのPDBサービスが使用される場合、接続は再利用されません。
- 共有プールを介してアクセスする様々なサービスは同種である必要があります。つまり、アプリケーション・コンティニュイティ(AC)、データベース常駐接続プーリング(DRCP)などに関して、類似したプロパティを持つ必要があります。
- 共有プールは単一のユーザーで構成する必要があります。このユーザーは、CDBで構成された共通ユーザーである必要があります。共通ユーザーには次の権限が必要です - CREATE SESSION、ALTER SESSIONおよびSET CONTAINER。また、共通ユーザーには、DBMS_SERVICE_PRVTパッケージの実行権限も必要です。

ノート:

- 共通ユーザーにテナントごとの特定のロールまたはパスワード対応のロールが必要な場合、これらのロールは、各テナント・データソース・プロパティで指定する必要があります。
- SET CONTAINER 文の利点は、別の PDB への既存の接続があれば、プールは PDB への新しい

接続を作成する必要がないことです。プールでは、既存の接続を使用でき、SET CONTAINER 文を通して、目的の PDB に接続できます。

- プールの接続の合計数が接続再利用しきい値(プールで構成されている場合)および最小プール・サイズに達する場合のみ、共有プールの様々なテナント接続の接続再利用が発生します。
- XML構成ファイルの共有プールに指定されたURLは、サービス名を明示的に指定したLONG形式である必要があります。短い形式または簡易接続URLはサポートされていません。

8.3 共有プールの構成

次の項では、共有プール構成について説明します。

- プールの初期構成
- プールの再構成

プールの初期構成

プールの初期構成では、XML構成ファイルを使用してデータソース・インスタンスを取得し、そのデータソースを使用して共有プールから接続を取得します。

```
    // Get the data source instance, named as "pds1" in the XML configuration file(initial-shared-
pool-config.xml)
    PoolDataSource pds1 = PoolDataSourceFactory.getPoolDataSource("pds1");
    Connection pds1Conn = pds1.getConnection();
```

プールの再構成

- 次のコードでは、プールの初期構成中に取得したデータソースの再構成方法を示します。

```
 // Reconfigure datasource(pds1) using the new propertries for reconfiguration

Properties newProps = new Properties();
newProps.put("serviceName",<newServiceName>);
pds1.reconfigureDataSource(newProps);
```

- 次のコードでは、新しいデータソースをすでに実行されている共有プールに追加する方法を示します。

```
// Configure a new datasource(pds3) to the running pool using the new data source properties

Properties dataSourceProps = new Properties();
dataSourceProps.put("serviceName", <serviceName>);
dataSourceProps.put("connectionPoolName", <poolName>);
dataSourceProps.put("dataSourceName", <dataSourceName>);
PoolDataSource pds3 = PoolDataSourceFactory.getPoolDataSource(dataSourceProps);
```

- 次のコードでは、接続プールの再構成方法を示します。

```
// Reconfigure connection pool("pool1") using the new properties

Properties newPoolProps = new Properties();
newPoolProps.put("initialPoolSize", <newInitialPoolSizeValue>);
newPoolProps.put("maxPoolSize", <newMaxPoolSizeValue>);
UniversalConnectionPoolManager ucpMgr =
UniversalConnectionPoolManagerImpl.getUniversalConnectionPoolManager();
ucpMgr.reconfigureConnectionPool("pool1", newPoolProps);
```

8.4 共有プール・サポートのUCP API

PoolDataSourceインタフェースの新しいメソッド

次のメソッドがoracle.ucp.jdbc.PoolDataSourceインタフェースで導入されました。

- reconfigureDataSource(Properties configuration)
- getMaxConnectionsPerService()
- getServiceName()
- getPdbRoles()
- getConnectionRepurposeThreshold()
- setConnectionRepurposeThreshold(int threshold)

PoolDataSourceFactoryクラスの新しいメソッド

次のメソッドがoracle.ucp.jdbc.PoolDataSourceFactoryクラスで導入されました。

- getPoolDataSource(String dataSourceName)
- getPoolDataSource(Properties configuration)
- getPoolXADataSource(String dataSourceName)
- getPoolXADataSource(Properties configuration)

oracle.ucp.admin.UniversalConnectionPoolManagerインタフェースの新しいメソッド

次のメソッドがoracle.ucp.admin.UniversalConnectionPoolManagerインタフェースで導入されました。

```
reconfigureConnectionPool(String poolName , Properties configuration)
```

oracle.ucp.admin.UniversalConnectionPoolインタフェースの新しいメソッド

次のメソッドがoracle.ucp.admin.UniversalConnectionPoolインタフェースで導入されました。

- isShareable()
- getMaxConnectionsPerService()
- setMaxConnectionsPerService(int maxConnectionsPerService)

関連項目:

詳細は、『Oracle Universal Connection Pool Java API Reference』を参照してください。

8.5 共有プールのサンプルのXML構成ファイル

initial-shared-pool-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ucp-properties>
    <connection-pool
        connection-pool-name="pool1"
        connection-factory-class-name="oracle.jdbc.pool.OracleDataSource"

url="jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=host_name)(PORT=1521)(PROTOCOL=tcp))( CONNECT_DATA=
(SERVICE_NAME=myorcldbservicename)))"
        user="C##CommonUser"
```

```
        password=password
        initial-pool-size="10"
        min-pool-size="5"
        max-pool-size="20"
        connection-repurpose-threshold="13"
        max-connections-per-service="15"
        validate-connection-on-borrow="true"
        sql-for-validate-connection="select 1 from dual"
        shared="true"
    >

        <connection-property name="oracle.jdbc.ReadTimeout" value="2000"/>
        <connection-property name="oracle.net.OUTBOUND_CONNECT_TIMEOUT" value="2000"/>

        <data-source
            data-source-name="pds1"
            service=pdb1_service_name
            description="pdb1 data source"/>

        <data-source
            data-source-name="pds2"
            service=pdb2_service_name
            description="pdb2 data source"/>


    </connection-pool>
</ucp-properties>
```

9 Oracle RAC機能の使用方法

この章の内容は次のとおりです。

- Oracle RAC機能の概要
- 高速接続フェイルオーバーについて
- ランタイム接続ロード・バランシングについて
- 接続アフィニティについて
- グローバル・データ・サービス

9.1 Oracle RAC機能の概要

UCP JDBC接続プールは、様々なOracle Real Application Cluster (Oracle RAC)データベース機能と緊密に統合されています。これらの機能には、高速接続フェイルオーバー(FCF)、実行時接続ロード・バランシングおよび接続アフィニティがあります。これらの機能には、Oracle JDBCドライバ、Oracle RACデータベース、Oracle Clientソフトウェアに同梱されているOracle Notification Serviceライブラリ(ons. jar)を使用する必要があります。

アプリケーションはOracle RAC機能を使用して、接続のパフォーマンスおよび可用性を最大化し、接続の問題による停止時間を軽減します。アプリケーションの可用性およびパフォーマンスの要件は一様ではないため、それに応じてOracle RAC機能を実装する必要があります。

ノート:

Oracle Database 11g リリース 1(11.2)から、単一インスタンス・データベースの FCF も Oracle Restart でサポートされます。Oracle Restart は、単一インスタンスの高可用性(SIHA)としてすでに知られています。

関連項目:

- これらのテクノロジの詳細は、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照してください。
- Oracle Restartの詳細は、『Oracle Database管理者ガイド』を参照してください。

一般的な高可用性およびパフォーマンスの機能

UCP APIおよび接続プールのプロパティには、Oracle RACデータベースを必要としない、多くの高可用性およびパフォーマンスの機能があります。これらの機能は、OracleとOracle以外のどちらの接続とも連携して有効に機能します。これらの機能については、このガイド全体を通して説明します。たとえば、流用時の接続の検証、タイムアウト・プロパティの設定、最大再利用プロパティの設定、接続プール・マネージャ操作はすべて、高いレベルの接続の可用性および最適なパフォーマンスを確保するために使用されます。

ノート:

一般的な高可用性およびパフォーマンスの機能は、Oracle 接続を使用する場合の方がわずかによく機能します。これは、UCP が Oracle JDBC の内部 API を利用するためです。

Oracle RACのデータベースのバージョン互換性

次の図では、様々なOracle RAC機能でサポートされているデータベースのバージョンを示します。

表9-1 Oracle RACのバージョン互換性

機能	サポートされているデータベースのバージョン

機能	サポートされているデータベースのバージョン
高速接続フェイルオーバー	Oracle Database 10.1.x 以上のバージョン
実行時接続ロード・バランシング	Oracle Database 10.2.x 以上のバージョン
Web セッション・アフィニティ	Oracle Database 11.1.x 以上のバージョン
トランザクションベースのアフィニティ	Oracle Database 10.2.x 以上のバージョン(Oracle Database 11.1.x を推奨)

Oracle RAC用のOracle JDBCドライバのバージョン互換性

Oracle JDBCドライバ10.1.x以上のバージョンがOracle RAC機能でサポートされています。

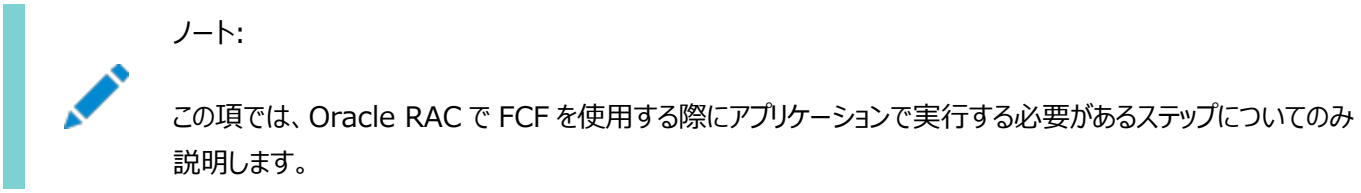
9.2 高速接続フェイルオーバーについて

この項には次のサブセクションが含まれます:

- 高速接続フェイルオーバーの概要
- 高速接続フェイルオーバーとは
- 高速接続フェイルオーバーの前提条件
- 高速接続フェイルオーバーの構成の例
- 高速接続フェイルオーバーの有効化
- ONSとは
- 接続URLの構成

9.2.1 高速接続フェイルオーバーの概要

高速接続フェイルオーバー(FCF)機能は、接続プールを通して実装される高速アプリケーション通知(FAN)クライアントです。この機能には、Oracle JDBCドライバと、Oracle RAC、Single RestartまたはActive Data Guardなどの高可用性(HA)データベース構成を使用する必要があります。



ノート:

この項では、Oracle RAC で FCF を使用する際にアプリケーションで実行する必要があるステップについてのみ説明します。

関連項目:

詳細は、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照してください。

FCFは次のシナリオで有用です。

- 計画外停止: FCFは停止した接続を迅速に検出し、終了してプールから削除します。接続を削除するには、システムが応答しなくなることを防ぐために、迅速にサーバーソケットの接続を切断する必要があります。流用された接続および使用中の接続には、計画外停止の場合にのみ割込みが発生します。
- 計画済停止: FCFは、作業が完了し接続の制御がプールに戻されるまで、流用された接続または使用中の接続を中断およびクローズしません。
- 致命的な接続エラーおよび例外: FCFは、再試行を確実かつ効果的に行うため、致命的な接続エラーおよび例外を `isValid` APIにカプセル化します。
- 新規ノードのクラスタへの追加: FCFは、Oracle RACクラスタに加わる新規ノードを認識し、新しい接続をそのノードに適切に配置して、アプリケーションの実行時に最高品質のサービスを提供します。これにより、Oracle RACノードの追加とアプリケーション層からの作業リクエスト・ルーティングの中間層の統合が容易になります。
- 実行時作業リクエスト: FCFは、実行時作業リクエストをすべてのアクティブなOracle RACインスタンスに配布します。

計画外停止のシナリオ

FCFでは、Oracle RACクラスタに対する失効した接続を検出および削除することで、計画外停止のシナリオをサポートします。

失効した接続には、サービス停止およびノード停止イベントのために、Oracle RACクラスタのインスタンスで利用できるサービスのない接続が含まれます。流用された接続と、使用可能な接続のうち失効しているものが検出され、それらのネットワーク接続が切断されてプールから削除されます。これらの削除された接続は、プールで置き換えられません。かわりに、アプリケーションでは接続を使用して処理を実行する前に、接続を再試行する必要があります。

ノート:

流用された接続は、計画外停止のシナリオ時にただちに終了およびクローズされます。実行中のトランザクションは、即座に例外を受け取ります。

計画停止のシナリオ

FCFでは、Oracle RACサービスを適切に停止できる計画停止のシナリオをサポートします。このシナリオの場合、失効している流用された接続はマークが付けられ、プールに返された後に終了および削除されます。実行中のトランザクションに変化はなく、完了するまで続行されます。

計画外停止と計画停止のシナリオの主な違いは、流用された接続の処理方法です。プール内でアイドル状態の(流用されていない)失効した接続は、計画外停止のシナリオと同じ方法で削除されます。

UCPは、計画停止したOracle RACインスタンスからの正常な接続の排出もサポートします。影響を受ける、流用されている接続は、プールに戻るとすぐに削除されるのではなく、猶予期間にスムーズに削除されます。これはサービスの再配置中のスループットの影響とログオン・ストームを避ける際に役立ちます。

Oracle RACインスタンスの再追加および新しいインスタンスのシナリオ

FCFでは、Oracle RACクラスタが関係するサービスを提供するインスタンスを追加するシナリオをサポートします。インスタンスは、クラスタにとって新しいものでも、停止イベント後に再起動されたものでもかまいません。どちらの場合でも、UCPは新しいインスタンスを認識し、必要に応じてノードへの接続を作成します。

関連トピック

- 接続の有効性のチェック
- 高速接続フェイルオーバーの有効化

9.2.2 高速接続フェイルオーバーとは

高速接続フェイルオーバーを使用可能に設定すると、メカニズムは自動なので、アプリケーションによる操作は必要ありません。この項では、次の項の接続フェイルオーバーがアプリケーションに提示される仕組みと、アプリケーションによるリカバリのステップを説明します。

- アプリケーションによる認識
- FCFの特長

9.2.2.1 アプリケーションによる認識

Oracle RACサービス障害がJDBCアプリケーションに伝播される前に、データベースはローカル・トランザクションをすでにロールバックしています。次に、キャッシュ・マネージャがすべての無効な接続をクリーン・アップします。無効な接続を保持しているアプリケーションがその接続を通して機能しようとすると、「SQLException, ORA-17008, クローズされた接続です。」を受け取ることがあります。

アプリケーションが「クローズされた接続です。」エラー・メッセージを受信した場合、次のことを実行する必要があります。

1. 接続要求を再試行します。古い接続はオープンでないため、これは必須です。

2. トランザクションを再実行します。接続がクローズする前に実行された作業はすべて失われています。

ノート:

アプリケーションによってトランザクションをロールバックしないでください。アプリケーションが例外を受信した時点で、トランザクションはすでにデータベースでロールバックされています。

9.2.2.2 FCFの特長

高速接続フェイルオーバーでは、キャッシュ内の各接続はサービス、インスタンス、データベースおよびホスト名へのマッピングを維持します。

データベースがOracle RACイベントを生成すると、そのイベントはJDBCが稼働中であるJVMに転送されます。JVM内のデーモン・スレッドがOracle RACイベントを受信し、それをConnection Cache Managerに転送します。次に、Connection Cache ManagerはOracle RACイベントの影響を受けるアプリケーションに対してSQL例外を発行します。

一般的なフェイルオーバーの例を次に示します。

1. データベース・インスタンスで障害が発生し、キャッシュ内にいくつかの失効した接続が残ります。
2. データベース内のOracle RACメカニズムがOracle RACイベントを生成し、そのイベントはJDBCが含まれているJVMに送信されます。
3. JVM内のデーモン・スレッドはOracle RACイベントの影響を受けるすべての接続を検出し、SQL例外によって接続がクローズされたことを通知し、オープンなトランザクションがあればすべてロールバックします。
4. 各接続はSQL例外を受信し、再試行します。

9.2.3 高速接続フェイルオーバーの前提条件

高速接続フェイルオーバーを使用するには、次の前提条件を満たしている必要があります。

- ユニバーサル接続プールが有効です。

  高速接続フェイルオーバーは、JDBC接続キャッシュ・メカニズムとの組合せで動作します。これにより、アプリケーションは接続を管理して高可用性を確保します。

- アプリケーションがデータベースへの接続にサービス名を使用していること

  アプリケーションはサービス識別子を使用できません。

- 基礎となるデータベースには、Oracle Database 12cリリース1 (12.1)以降のReal Application Clusters (Oracle RAC)機能、または単一インスタンス・データベースかOracle RACのいずれかで構成されたOracle Data Guardがあること。

  フェイルオーバー・イベントが伝播しないと、接続フェイルオーバーは発生しません。

- JDBCが稼働しているノードでOracle Notification Service(ONS)が構成されており、使用可能であること

  JDBCは、データベース・イベントの伝播およびJDBCへのイベントの通知を行うために、ONSに依存しています。

- JDBCインスタンスが稼働しているJava仮想マシン(JVM)で、oracle.ons.oraclehomeがORACLE_HOMEを指すように設定されていること

9.2.4 高速接続フェイルオーバーの構成の例

次の例では、FCF機能を使用する接続プールを示します。FCFは、プール対応のデータソースを使用して構成されます。この例では、FCFの有効化、Oracle Notification Service(ONS)の構成および接続URLの構成が行われます。これらの項目については、例の後で説明します。

通常、oracle.ucp.jdbc.ValidConnectionインタフェースのisValidメソッドはFCF機能とともに使用され、Oracle RAC停止イベントによってSQL例外がスローされた後でも流用された接続が使用可能かどうかをチェックするために使用されます。次に例を示します。

```
try {  conn = pds.getConnection;  ...}catch (SQLException sqlexc)
{
   if (conn == null || !((ValidConnection) conn).isValid())

   // take the appropriate action

...
conn.close();
}
```

例9-1 高速接続フェイルオーバーの構成の例

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("FCFSamplePool");
pds.setFastConnectionFailoverEnabled(true);
pds.setONSConfiguration("nodes=racnode1:4200,racnode2:4200¥nwalletfile=
/oracle11/onswalletfile");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin@(DESCRIPTION= "+
   "(LOAD_BALANCE=on)"+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=racnode1) (PORT=1521))"+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=racnode2) (PORT=1521))"+
   "(CONNECT_DATA=(SERVICE_NAME=service_name)))");
...
```

関連トピック

- 接続の有効性のチェック

9.2.5 高速接続フェイルオーバーの有効化

FCFプール・プロパティを使用して、FCFを有効または無効にします。デフォルトではFCFは無効になっています。次の例では、例9-1に示されているFCFの有効化を示します。

```
pds.setFastConnectionFailoverEnabled(true);
```

ノート:

Oracle Database 12c リリース 1 (12.1.0.2)以降、UCP は oracle.ucp.PlannedDrainingPeriod システム・プロパティをサポートします。計画停止により影響を受ける、流用されている接続をプールがスムーズに排出する猶予期間を指定します(整数の秒数)。同じデータベース・サービスが、停止中のものとは別のインスタンスで使用可能になると、排出が開始されます。

このプロパティが設定されていない、または 0 に設定されている場合、プールは影響を受ける、流用されている接続がプールに戻されるとすぐにクローズします。

高速接続フェイルオーバーのステータスの問合せ

アプリケーションは、高速接続フェイルオーバーが使用可能になっているかどうかを判断するために、OracleDataSource.getFastConnectionFailoverEnabledをコールします。フェイルオーバーが使用可能な場合はtrueが、それ以外の場合はfalseが戻されます。

ノート:

FCF は、実行時接続ロード・バランシングおよび接続アフィニティを使用する場合にも有効にする必要があります。これらの機能については、この章で後述します。

9.2.6 ONSとは

FCFは、Oracle Notification Service(ONS)を利用して、接続プールとOracle RACデータベース間でデータベース・イベントを伝播します。接続プールは、実行時にONS環境を設定できる必要があります。ONS(ons.jar)はOracle Clientソフトウェアに同梱されています。ONSは、リモート構成またはクライアント側のONSデーモン構成を使用して構成できます。リモート構成は、スタンドアロンのクライアント・アプリケーションに適した構成です。このセクションのトピックは次のとおりです:

- ONS構成ファイルの概要
- ONSのリモート構成
- クライアント側のONSデーモンの構成

9.2.6.1 ONS構成ファイルの概要

ONS構成は、ONSの構成ファイルORACLE_HOME/opmn/conf/ons.configを使用して制御します。このファイルは、ONSデーモンにその動作方法を示します。ons.config内の構成情報は、単純な名前と値のペアで定義されます。

ons.configファイルのパラメータには必須のものとオプションのものがあります。表9-2は必須のONS構成パラメータを示し、表9-3はオプションのONS構成パラメータを示します。ons.configファイルの更新後に、ONSをリフレッシュする必要があります。

表9-2 必須のONS構成パラメータ

パラメータ	説明
localport	ローカル・クライアントと通信するために、ONS がローカル・ホスト・インタフェースでバインドするポートを指定します。 たとえば、localport=4100 です
remoteport	他の ONS デーモンと通信するために、ONS がすべてのインタフェースでバインドするポートを指定します。

パラメータ	説明
	たとえば、remoteport=4200 です
nodes	通信相手である他の ONS デーモンのリストを指定します。ノード値は、ホスト名または IP アドレスにポートを追加したもののカンマ区切りのリストとして指定します。提供するポート値は、各 ONS インスタンスがリスニングしているリモート・ポートです。全ノードで同一のファイルを維持するために、現行の ONS ノードの host:port をノード・リストに含めることも可能です。リストの読取り時には、これは無視されます。 たとえば、nodes=myhost.example.com:4200,123.123.123.123:4200 です。 ノード行にリスト表示されているノードは、Oracle RAC インスタンス内の個別のノードに対応します。ノードをリストに含めることにより、中間層ノードは Oracle RAC ノードと確実に通信できます。最低 1 つの中間層ノードと、1 つの Oracle RAC インスタンス内のノードを構成しないと、互いが認識されません。それぞれの側で最低 1 つのノードが互いを認識していると、すべてのノードが認識されます。各 Oracle RAC ノードの ONS 構成ファイルでは、すべての単一クラスタおよび中間層ノードをリスト表示する必要はありません。特に、1 つの ONS 構成ファイル・クラスタ・ノードが中間層を認識している場合、クラスタ内のすべてのノードが中間層を認識しています。

表9-3 オプションのONS構成パラメータ

パラメータ	説明
logcomp	記録する ONS コンポーネントを指定します。書式は次のとおりです。 <component>[<subcomponent>,...];<component>[<subcomponent>,...];... サブコンポーネントを指定する必要がない場合、コンポーネント名の後にカッコ([])を含まないでください。サブコンポーネントからメッセージを除外するには、サブコンポーネント名の前に感嘆符(!)が必要です。たとえば、topology サブコンポーネントからメッセージを除外するには、次の書式を使用します。 [all,!topology] メッセージを除外するサブコンポーネントを指定する前に、サブコンポーネントがメッセージを含むことを最初に確認する必要があります。 次に、コンポーネントの有効な値を示します。 ● internal ● ons internal としてコンポーネントを指定する場合、サブコンポーネントの有効な値はありません。ons としてコンポーネントを指定する場合、サブコンポーネントの次の値を指定できます。 ● all: すべてのメッセージを指定します

パラメータ	説明
	● ons: ONS ローカル情報 ● listener: ONS リスナー情報 ● discover: ONS 検出(サーバーまたはマルチキャスト)情報 ● servers: クラスタに接続されていて現在稼働中の ONS リモート・サーバー ● topology: ONS の現在のクラスタ全体のサーバー接続トポロジ ● server: ONS リモート・サーバー接続情報 ● client: ONS クライアント接続情報 ● connect: ONS の全般的な接続情報 ● subscribe: ONS クライアント・サブスクリプション情報 ● message: ONS の通知の受信および処理情報 ● deliver: ONS 通知配信情報 ● special: ONS の特殊な通知処理 ● internal: ONS 内部リソース情報 ● secure: ONS の SSL 操作情報 ● workers: ONS ワーカー・スレッド 次の例は、secure サブコンポーネントを除く ons の下のすべてのサブコンポーネントのメッセージを記録する場合を示します。 logcomp=ons[all,!secure]
logfile	ONS がメッセージの記録に使用するログ・ファイルを指定します。ログ・ファイルのデフォルト値は $ORACLE_HOME/opmn/logs/ons.log です。 たとえば、logfile=/private/oraclehome/opmn/logs/myons.log です。
walletfile	Oracle Secure Sockets Layer(SSL)が SSL 証明書の格納に使用するウォレット・ファイルを指定します。ONS でウォレット・ファイルが指定されると、他の ONS インスタンスとの通信時に SSL を使用し、接続しようとするすべての ONS インスタンスから SSL 証明書の認証を要求します。つまり、1 つの ONS インスタンスで SSL を有効にする場合、このインスタンスと接続されているすべてのインスタンスでも有効にする必要があります。この値は、ewallet.p12 ファイルが存在するディレクトリを指すように設定します。

パラメータ	説明
	たとえば、walletfile=/private/oraclehome/opmn/conf/ssl.wlt/default です。
useocr	ONS がすべての Oracle RAC ノードおよびポート番号を、ONS 構成ファイルでなく Oracle Cluster Registry (OCR)に格納するかどうかを示す、サーバー側での使用のために予約されている値を指定します。useocr=on の場合、すべての Oracle RAC ノードおよびポート番号が Oracle Cluster Registry (OCR)に格納されます。 このオプションは、クライアント側では使用しないでください。
allowgroup	localport に接続するユーザー・グループを示す ONS 設定を指定します。true に設定すると、ONS は同じ OS グループ内のユーザーにローカル・ポートへの接続を許可します。false に設定すると、ONS は ONS デーモンを実行している同じユーザーにローカル・ポートのアクセスを許可します。このパラメータのデフォルト値は false です。リモート ONS 構成を使用する場合、このパラメータを設定する必要はありません。

ons.configファイルでは、ナンバー記号(#)で始まる行に空白行およびコメントを使用できます。

9.2.6.2 ONSのリモート構成

UCPでは、ONSConfigurationプール・プロパティを使用したONSのリモート構成をサポートします。ONSConfigurationプール・プロパティ値は、ons.configファイルの内容に酷似している文字列です。この文字列は、改行文字(¥n)で区切られたname=valueペアのリストからなります。このプール・プロパティは、次の2つの方法で設定できます。

- 名前は、nodes、walletfile、walletpasswordのいずれかです。パラメータ文字列では、少なくともONS構成のnodes属性をカンマ区切りのhost:portペアのリストとして指定する必要があります。walletfile属性がOracleウォレット・ファイルとして指定される場合は、SSLが使用されます。

  次の例では、例9-1に示されているONS構成文字列を示します。

  ```
  ...
  pds.setONSConfiguration("nodes=racnode1:4200,racnode2:4200¥nwalletfile=/oracle11/onswalletfile
  ");
  ...
  ```

- 名前は、propertiesfileにしかできません。この値は、ONS固有のJavaプロパティ・ファイルの場所です。このファイルには、oracle.ons.nodesプロパティと、次のONS Javaプロパティの一方または両方が含まれている必要があります。
  - oracle.ons.walletfile
  - oracle.ons.walletpassword

    次の例では、このようなONSConfiguration文字列を示します。

    ```
    pds.setONSConfiguration("propertiesfile=/usr/ons/ons.properties");
    ```

    Javaプロパティ・ファイルons.propertiesの内容の例を次に示します。

    ```
    oracle.ons.nodes=racnode1:4200,racnode2:4200
    oracle.ons.walletfile=/oracle11/onswalletfile
    ```

ノート:

構成文字列内のパラメータは、Oracle RAC Database のパラメータと一致する必要があります。さらに、Oracle Application Server を使用する場合、サーバーに適用可能な手順を使用して ONS を構成する必要があります。

スタンドアロン Java アプリケーションの場合、setONSConfiguration メソッドを使用して ONS を構成する必要があります。ただし、アプリケーションが次の要件を満たす場合、FCF を有効化するために、setONSConfiguration メソッドを呼び出す必要はなくなります。

- アプリケーションが Oracle Database 12c リリース 1 (12.1)以降の UCP および Oracle RAC Database 12c リリース 1 (12.1)を使用している
- アプリケーションで ONS ウォレットまたはキーストアを必要としない

9.2.6.3 クライアント側のONSデーモンの構成

クライアント側のONSデーモン構成は、Oracle Application Serverなどの中間層サーバーで実行される典型的なアプリケーションです。このシナリオでのクライアントは、ons.configファイルを更新することでONSを直接構成します。ファイルの場所は、プラットフォームによって異なります。例9-2では、例9-1用のons.configファイルを示します。

ノート:

クライアント側の ONS デーモン構成では、接続プールを起動するオペレーティング・システム(OS)ユーザーと、クライアント側のデーモンを起動する OS ユーザーが異なる場合、どちらのユーザーも同じ OS グループに属する必要があります。また、allowgroup パラメータの値は、ons.config ファイルで true に設定する必要があります。

ONSを構成した後、onsctlコマンドでONSデーモンを起動します。ONSデーモンが常に実行中であることを確認する必要があります。

onsctlコマンドの使用

構成後、ORACLE_HOME/opmn/bin/onsctlを使用してONSデーモンの起動、停止、再構成および監視を行います。表9-4は、onsctlでサポートされているコマンドのサマリーを示しています。

表9-4 onsctlコマンド

コマンド	効果	出力
start	ONS デーモンを起動	onsctl: ons started
stop	ONS デーモンを停止	onsctl: shutting down ons daemon...
ping	ONS デーモンが稼働中であるかどうかを	ons is running ...

コマンド	効果	出力
	検証	
reconfig	ONS デーモンをシャットダウンせずに ONS 構成のリロードをトリガー	
help	onsctl のヘルプ・サマリー・メッセージを出力	
detailed	onsctl の詳細ヘルプ・メッセージを出力	

関連項目:

『Oracle Real Application Clusters管理およびデプロイメント・ガイド』

ノート:

- JDBC インスタンスが実行されている Java 仮想マシン(JVM)では、アプリケーションの起動前に oracle.ons.oraclehome システム・プロパティを ORACLE_HOME の場所に設定する必要があります。次に例を示します。

```
java -Doracle.ons.oraclehome=$ORACLE_HOME ...
```

- UCP に対しては ONS のリモート構成をお薦めします。

ノート:

Oracle RAC 12.1.0.2.0 では、デフォルトで、サーバーのインストールに walletfile ONS パラメータの値の設定が必要で、すべての ONS 接続に SSL の使用が強制されます。

ONS リモート構成文字列またはローカル構成ファイルの walletfile パラメータをすでに使用している UCP アプリケーションがある場合、要件は、同じトポロジでは、クライアント側のウォレット・ファイルにサーバー側のウォレット・ファイルと同じ内容が含まれる必要があることだけです。サーバー側ファイルのコピーを作成し、クライアント側で使用できるようにすることができます。

walletfile パラメータを設定せずに Oracle RAC 機能を使用している UCP アプリケーションの場合、次のいずれかを実行する必要があります。

- 例 9-1 に示すように、walletfile パラメータの設定を ONS リモート構成文字列またはローカル構成ファイルに追加します。同じトポロジでは、クライアント側のウォレット・ファイルには、Oracle RAC サーバー側のウォレット・ファイルと同じ内容が含まれる必要があることに注意してください。

- 次のコマンドを実行して walletfile パラメータ設定をクライアントおよびサーバーの ONS 構成文字列およびローカル構成ファイルから削除します。

```
srvctl modify nodeapps -clientdata
```

セキュアな通信のために、Oracle RAC 12.1.0.2.0 を最初にインストールする、またはパッチを適用すると、Oracle RAC 12.1.x の ONS 自動構成は機能しなくなります。かわりに、アプリケーションは明示的 ONS 構成(リモートまたはローカル)を使用し、前述のいずれかの変更を行う必要があります。

例9-2 サンプルのons.configファイルの例

```
# This is an example ons.config file
#
# The first three values are required
localport=4100
remoteport=4200
nodes=racnode1.example.com:4200,racnode2.example.com:4200
```

9.2.7 接続URLの構成

コネクション・ファクトリの接続URLでは、FCFを使用する際、サービス名の構文を使用する必要があります。サービス名は、接続プールをサービスにマップするために使用されます。また、ファクトリ・クラスはOracleファクトリ・クラスである必要があります。次の例では、例9-1に示されている接続URLの構成を示します。

```
...
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin@//host:port/service_name");
...
```

ノート:

FCF が有効である場合、サービス識別子(SID)を接続 URL に指定すると、例外がスローされます。

次の例では、Oracle RACデータベースに接続する際の有効な接続URLの構文を示します。Oracle JDBC Thinドライバと Oracle OCIドライバの両方の例が含まれています。URLを使用してOracle RACノード間のロード・バランシングを明示的に有効にできることに注意してください。

有効な接続URLの使用方法

```
pds.setURL("jdbc:oracle:thin@//host:port/service_name");

pds.setURL("jdbc:oracle:thin@//cluster-alias:port/service_name");

pds.setURL("jdbc:oracle:thin:@(DESCRIPTION= "+
   "(LOAD_BALANCE=on)"+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=host1)(PORT=1521))"+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))"+
   "(CONNECT_DATA=(SERVICE_NAME=service_name)))");

pds.setURL("jdbc:oracle:thin:@(DESCRIPTION= "+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=cluster_alias) (PORT=1521)) "+
   "(CONNECT_DATA=(SERVICE_NAME=service_name)))");
```

```
pds.setURL("jdbc:oracle:oci:@TNS_ALIAS");

pds.setURL("jdbc:oracle:oci:@(DESCRIPTION= "+
   "(LOAD_BALANCE=on) "+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=host1) (PORT=1521)) "+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521)) "+
   "(CONNECT_DATA=(SERVICE_NAME=service_name)))");

pds.setURL("jdbc:oracle:oci:@(DESCRIPTION= "+
   "(ADDRESS=(PROTOCOL=TCP)(HOST=cluster_alias) (PORT=1521)) "+
   "(CONNECT_DATA=(SERVICE_NAME=service_name)))");
```

9.3 ランタイム接続ロード・バランシングについて

この項には次のサブセクションが含まれます:

- 実行時接続ロード・バランシングの概要
- 実行時接続ロード・バランシングの設定

9.3.1 実行時接続ロード・バランシングの概要

Oracle Real Application Clusters環境では、接続は関連サービスを提供するインスタンスに属しています。すべてのインスタンスが等しく実行され、接続がキャッシュからランダムに取り出されるのが最も理想的です。しかし、あるインスタンスのパフォーマンスが他のインスタンスを上回る場合、接続をランダムに選択すると効率が悪くなります。実行時接続ロード・バランシング機能を使用すると、最適なパフォーマンスを提供するインスタンスに作業要求がルーティングされるため、作業を再配置する必要が最小限になります。

UCP JDBC接続プールは、Oracle RACデータベースが提供するロード・バランシング機能を利用します。実行時接続ロード・バランシングには、Oracle JDBCドライバおよびOracle RACデータベースを使用する必要があります。

関連項目:

Oracle Real Application Clusters管理およびデプロイメント・ガイド

実行時接続ロード・バランシングは、次の場合に便利です。

- 従来のワークロード・バランシングが最適でない場合
- クラスタ・データベース内のリソースを最大限に利用するようにリクエストをルーティングする必要がある場合
- クラスタ内の許容量が異なり、時間とともに変化することが予想される場合
- 低速なノード、ハングアップしたノードおよびデッド・ノードに作業を送信しないようにする要件が必須である場合

UCPは、Oracle RACロード・バランシング・アドバイザを使用します。このアドバイザを使用して、Oracle RACインスタンス間の作業のバランスをとり、最高のパフォーマンスを発揮するインスタンスを特定します。アプリケーションは、最高のパフォーマンスを発揮するインスタンスから透過的に接続を受け取ります。速度が落ちたインスタンス、レスポンスを返さないインスタンスまたは障害が発生したインスタンスからは、ただちに接続リクエストが変更されます。

実行時接続ロード・バランシングには次の利点があります。

- 高いパフォーマンスおよびスケーラビリティのためにプールされた接続を管理します。
- データベース・インスタンスにルーティングする作業の比率の推奨値を継続的に受け取ります。
- CPU性能またはレスポンス時間など、様々なバックエンド・ノードの能力に基づいて作業の分散を調整します。
- クラスタ構成の変更、アプリケーション・ワークロード、過度に使用されているノード、ハングアップにすばやく対応します。
- Oracle RACロード・バランシング・アドバイザからメトリックを受け取ります。パフォーマンスが良好なインスタンスへの接続が最もよく使用されます。パフォーマンスが低いインスタンスへの新しい未使用の接続は、時間とともに使用されなくなります。分散メトリックを受け取らない場合、接続はランダムに選択されます。

9.3.2 実行時接続ロード・バランシングの設定

実行時接続ロード・バランシングには、FCFが有効であり、適切に構成されていることが必要です。

また、ロード・バランシングを有効にする各サービスのサービス・レベルの目標を使用してOracle RACロード・バランシング・アドバイザを構成する必要があります。

- サービス目標は、次のいずれかに設定する必要があります。
  - DBMS_SERVICE.SERVICE_TIME
  - DBMS_SERVICE.THROUGHPUT

  サービス目標はgoalパラメータを使用して、接続バランシング目標はclb_goalパラメータを使用して設定できます。
- 接続バランシング目標はSHORTに設定する必要があります。次に例を示します。

```
EXECUTE DBMS_SERVICE.MODIFY_SERVICE (service_name => 'sjob' -, goal =>
   DBMS_SERVICE.GOAL_THROUGHPUT -, clb_goal => DBMS_SERVICE.CLB_GOAL_SHORT);
```

  または

```
EXECUTE DBMS_SERVICE.MODIFY_SERVICE (service_name => 'sjob' -, goal =>
   DBMS_SERVICE.GOAL_SERVICE_TIME -, clb_goal => DBMS_SERVICE.CLB_GOAL_SHORT);
```

DBMS_SERVICE.create_serviceプロシージャをコールして、接続バランシング目標も設定できます。

ノート:

接続バランシング目標は、LONG に設定することができます。ただし、これはクローズされたワークロードに対して、つまり完了した作業の割合と新しく始める作業の割合が等しいときに、最も有効です。

関連トピック

- 高速接続フェイルオーバーについて

関連項目:

『Oracle Real Application Clusters管理およびデプロイメント・ガイド』

9.4 接続アフィニティについて

この項には次のサブセクションが含まれます:

- 接続アフィニティの概要
- 接続アフィニティの設定

9.4.1 接続アフィニティの概要

UCP JDBC接続プールは、Oracle RACデータベースが提供するアフィニティ機能を利用します。接続アフィニティには、Oracle JDBCドライバとリリース11.1.0.6以上のOracle RACデータベースを使用する必要があります。

接続アフィニティは、特定のOracle RACインスタンスに送られる接続を接続プールで選択できるようにするパフォーマンス機能です。プールが実行時接続ロード・バランシング(構成されている場合)を使用し、Oracle RACインスタンスを選択して最初の接続を作成すると、後続の接続は同じインスタンスへのアフィニティを使用して作成されます。

関連項目:

- 「厳密なアフィニティ・モード」
- Oracle RACデータベースの設定の詳細は、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照してください。

UCP JDBC接続プールは、次の3つのタイプの接続アフィニティをサポートします。

- トランザクションベースのアフィニティ
- Webセッション・アフィニティ
- Oracle RACデータ・アフィニティ

9.4.1.1 トランザクションベースのアフィニティ

トランザクションベースのアフィニティは、クライアント・アプリケーションまたは障害イベントによって解放できるOracle RACインスタンスへのアフィニティです。通常、アプリケーションがこのタイプのアフィニティを使用するのは、Oracle RACインスタンスへの存続期間が長いアフィニティが望ましい場合、または新しいOracle RACインスタンスへのリダイレクトのコストが(パフォーマンスの点で)高い場合です。分散トランザクションは、トランザクションベースのアフィニティのよい例です。分散トランザクションに参加しているXA接続は、トランザクション中にOracle RACインスタンスへのアフィニティを保持します。この場合、分散トランザクション中に接続が別のOracle RACインスタンスにリダイレクトされると、アプリケーションで非常に高いパフォーマンス・コストが発生します。

9.4.1.2 Webセッション・アフィニティ

Webセッション・アフィニティは、インスタンス、クライアント・アプリケーションまたは障害イベントによって解放できるOracle RACインスタンスへのアフィニティです。Oracle RACインスタンスは、インスタンスでアフィニティが有効か無効かに関係なく、ヒントを使用して接続プールと通信します。Oracle RACインスタンスは、パフォーマンスやロードなどの様々な要素に基づいてアフィニティを無効にできます。Oracle RACインスタンスでアフィニティがサポートされなくなると、プール内の接続は新しいインスタンスを使用するためにリフレッシュされ、アフィニティが再設定されます。

通常、アプリケーションがこのタイプのアフィニティを使用するのは、Oracle RACインスタンスへの存続期間が短いアフィニティが望

ましい場合、または新しいOracle RACインスタンスへのリダイレクトのコストが(パフォーマンスの点で)最も低い場合です。たとえば、メール・クライアント・セッションでは、Oracle RACインスタンスへのWebセッション・アフィニティを使用してパフォーマンスを向上させますが、接続が別のインスタンスにリダイレクトされても相対的に影響を受けません。

9.4.1.3 Oracle RACデータ・アフィニティ

データ・アフィニティは、関連するキャッシュ・エントリ・グループが単一のキャッシュ・パーティションに含まれることを保証する概念を表します。

Oracle Databaseリリース18c以降、UCPではOracle RACデータ・アフィニティがサポートされています。Oracle RACデータベースでデータ・アフィニティを有効にすると、アフィニティ設定された表のデータはパーティション化され、表の特定のパーティションまたは行のサブセットが、特定のOracle RACデータベース・インスタンスにアフィニティ設定されます。アフィニティによって、キャッシュ局所性の改善、ノード間の同期化の削減、RACインスタンス間のpingブロックにより、アプリケーションのパフォーマンスおよびスケーラビリティが向上します。

関連項目:

カスタムのパーティション割当て戦略の有効化

Oracle RACデータ・アフィニティ機能を使用するには、UCPを使用してデータベースにアクセスするクライアントは、接続リクエストでデータ・アフィニティ・キーを指定する必要があります。アフィニティ対応RACデータベースの接続をプールする際に、UCPには次の機能があります。

1. UCPはプールの起動時に、Oracle RACインスタンス間のデータ・パーティションのデータ・アフィニティを含むトポロジを認識します。
2. Oracle RACデータ・アフィニティ機能を利用する必要があるUCP接続リクエストによって、シャーディング・キー・ビルダーを使用してデータ・アフィニティ・キーが指定され、接続ビルダーが次のように使用されます。

```
  PoolDataSource pds = new PoolDataSourceImpl();
   // configure the datasource with the database connection properties

/* Builds the RAC data affinity key using the sharding key builder API
and gets a connection from the pool using UCP connection builder */
  OracleShardingKey dataAffinityKey =  pds.createShardingKeyBuilder()
          .subkey(1000, OracleType.NUMBER)
          .build();

  Connection connection = pds.createConnectionBuilder()
          .shardingKey(dataAffinityKey)
          .build();
```

ノート:

データ・アフィニティ・キーを指定せずに、有効化された Oracle RAC データ・アフィニティへの接続リクエストを行うこともできます。ただし、この場合は、Oracle RAC データ・アフィニティ機能のメリットはありません。

3. UCPは、リクエストで指定されたシャード・キーに対してアフィニティ設定されたインスタンスを判別し、そのインスタンスの接続がプールに存在するかどうかを確認します。接続が存在する場合は、その接続がリクエストを処理するために使用されます。一致する接続がプールに存在しない場合、実行時ロード・バランシングへのフォールバックにより、リクエストの接続が選択されて処理されます。リクエストを処理するために新しい接続を作成する必要がある場合、リクエストは指

定したシャード(データ・アフィニティ)キーに対応する、アフィニティ設定されたインスタンスにルーティングされます。

4. HAイベントがある場合、またはOracle RACでデータ・パーティションのアフィニティに変更がある場合、UCPは、データ・パーティションのトポロジがサーバー側と同期するようにします。

9.4.2 接続アフィニティの設定

次のステップを実行して、接続アフィニティを設定します。

- FCFを有効にします。

  関連項目:

  「高速接続フェイルオーバーについて」

- 実行時接続ロード・バランシングを有効にします。

  関連項目:

  「ランタイム接続ロード・バランシングについて」

- 接続アフィニティ・コールバックを作成します。
- コールバックを登録します。

> ノート:
>
> トランザクションベースのアフィニティは、アプリケーション/中間層および UCP 間で厳密に有効範囲が指定されます。したがって、トランザクションベースのアフィニティは、setFastConnectionFailoverEnabled プロパティの true への設定のみが必要であり、完全な FCF 構成は必要ありません。
>
> また、トランザクションベースのアフィニティは、技術的には実行時接続ロード・バランシングを必要としません。しかし、パフォーマンスの向上に役立つため、通常は有効にしておきます。実行時接続ロードバランシングが無効である場合、接続プールはランダムに接続を選択します。

この項には次のサブセクションが含まれます:

- 接続アフィニティ・コールバックの作成
- 接続アフィニティ・コールバックの登録
- 接続アフィニティ・コールバックの削除

9.4.2.1 接続アフィニティ・コールバックの作成

接続アフィニティには、コールバックを使用する必要があります。コールバックは、oracle.ucpパッケージにあるConnectionAffinityCallbackインタフェースの実装です。コールバックは、接続アフィニティ・コンテキストを設定および取得するために接続プールで使用されます。また、アフィニティ・ポリシー・タイプ(トランザクションベースまたはWebセッション)の設定にも

使用されます。

次の例では、コールバックの実装でのアフィニティ・ポリシーの設定を示します。また、アフィニティ・コンテキストの手動による設定も示します。通常、接続プールがアプリケーション内部でアフィニティ・コンテキストを設定します。しかし、アフィニティの動作をカスタマイズしたりアフィニティ・コンテキストを直接制御するアプリケーションのために、アフィニティ・コンテキストを手動で設定する機能があります。

```
public class AffinityCallbackSample
   implements ConnectionAffinityCallback {

   Object appAffinityContext = null;
   ConnectionAffinityCallback.AffinityPolicy affinityPolicy =
   ConnectionAffinityCallback.AffinityPolicy.TRANSACTION_BASED_AFFINITY;

   //For Web session affinity, use WEBSESSION_BASED_AFFINITY;

   public void setAffinityPolicy(AffinityPolicy policy)
   {
      affinityPolicy = policy;
   }

   public AffinityPolicy getAffinityPolicy()
   {
      return affinityPolicy;
   }

   public boolean setConnectionAffinityContext(Object affCxt)
   {
      synchronized (lockObj)
      {
         appAffinityContext = affCxt;
      }
      return true;
   }

   public Object getConnectionAffinityContext()
   {
      synchronized (lockObj)
      {
         return appAffinityContext;
      }
   }
}
```

9.4.2.2 接続アフィニティ・コールバックの登録

接続アフィニティ・コールバックは、registerConnectionAffinityCallbackメソッドを使用して接続プールに登録されます。コールバックは、接続プールの作成時に登録されます。接続プールごとに登録できるコールバックは1つのみです。

次の例では、接続アフィニティ・コールバックの実装の登録を示します。

```
ConnectionAffinityCallback callback = new MyCallback();

PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("AffinitySamplePool");
pds.registerConnectionAffinityCallback(callback);
...
```

9.4.2.3 接続アフィニティ・コールバックの削除

接続アフィニティ・コールバックは、removeConnectionAffinityCallbackメソッドを使用して接続プールから削除されます。次に例を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("AffinitySamplePool");
pds.removeConnectionAffinityCallback();
...
```

9.4.2.4 厳密なアフィニティ・モード

デフォルトでは、アフィニティはヒントにすぎません。必要なインスタンスの接続が見つからない場合、接続プールは、接続の新しいOracle RACインスタンスを選択します。厳密なアフィニティ・モードをオンに切り替えて、この動作を変更できます。必要なインスタンスの接続が見つからない場合、厳密なアフィニティ・モードはUCP例外をスローします。

次のプール・プロパティを使用して、厳密なアフィニティ・モードをオンに切り替えます。

- useStrictWebSessionAffinityプロパティ

  厳密なWebセッション・アフィニティ・モードをオンまたはオフに切り替えるには、useStrictWebSessionAffinityプロパティをtrueまたはfalseに設定します。

- useStrictXAAffinityプロパティ

  厳密なトランザクションベースのアフィニティ・モードをオンまたはオフに切り替えるには、useStrictXAAffinityプロパティをtrueまたはfalseに設定します。

UniversalConnectionPoolMBeanを使用して、これらのプロパティを処理できます。

関連トピック

- UniversalConnectionPoolMBean

9.5 グローバル・データ・サービス

この項では、ユニバーサル接続プールで使用できる新しいグローバル・データ・サービス(GDS)機能について説明します。

- Global Data Servicesの概要
- GDSを使用するためのアプリケーションの構成

9.5.1 Global Data Servicesの概要

グローバル・データ・サービス(GDS)機能は、Oracle Database 12cリリース1 (12.1)以降で使用できます。この機能により、Oracle RACでのみ使用できた高速接続フェイルオーバー機能、実行時接続ロード・バランシング機能および接続アフィニティ機能は、共通サービスを提供する一連の複製データベースにまで拡張されました。

一連のデータベースには、Data Guard、GoldenGate、他のレプリケーション・テクノロジによって相互接続されたOracle RACデータベースや単一インスタンスのOracleデータベースが含まれます。複数のデータベースによって提供できるデータベース・サービスは、グローバル・サービスと呼ばれるため、単一データベースによってのみ提供できる従来のサービスとは区別できます。この組合せにより、このグローバルに分散された構成内の任意の場所にサービスをデプロイできるため、ロード・バランシング、高可用性、データベース・アフィニティなどがサポートされます。

関連項目:

Oracle Database Global Data Services概要および管理ガイド

9.5.2 GDSを使用するためのアプリケーションの構成

UCPは、Oracle RACのローカル・サービスに接続するのと同じ方法でグローバル・データ・サービスに接続します。接続文字列のサービス名は、グローバル・サービスの名前である必要があります。エンドポイントは、データベースのローカル・リスナー、リモート・リスナーまたはSCANリスナーのエンドポイントではなく、GDSリスナーのエンドポイントである必要があります。

クライアントは、接続文字列のREGIONパラメータでリージョンを指定する必要があります。これは、新しいGDSに関する要件です。GDSの場合、実行時ロード・バランシング・アドバイザが特定のリージョン用にカスタマイズされるため、リージョン名は必須です。一般的な接続文字列の例を次に示します。

```
(DESCRIPTION=
  (ADDRESS=(GDS_protocol_address_information))
  (CONNECT_DATA=
   (SERVICE_NAME=global_service_name)
   (REGION=region_name)))
```

ローカル・サービスと同様、UCPは、リスナー・フェイルオーバーまたはロード・バランシング(あるいはその両方)用に複数のGDSリスナーを同じ接続文字列に指定できます。

ノート:

SCAN は GDS リスナーに対してサポートされていないため、各リスナーのエンドポイントを指定する必要があります。

```
(DESCRIPTION=
```

(ADDRESS_LIST= (LOAD_BALANCE=ON) (FAILOVER=ON)
(ADDRESS=(GDS_protocol_address_information))
(ADDRESS=(GDS_protocol_address_information))) (CONNECT_DATA=
(SERVICE_NAME=global_service_name) (REGION=region_name)))

ローカル・リージョンのグローバル・サービス・マネージャのみがクライアント接続文字列に指定されている場合、REGIONパラメータはオプションです。これは、GDS構成に1つのリージョンのみが存在する場合か、または複数のリージョンがある場合に該当する可能性があります。ただし、単一データベースで動作するよう設計されている既存のクライアントの接続文字列を変更することはできません。REGIONパラメータが指定されていない場合、クライアントのリージョンは、グローバル・サービスに接続するために使用されるグローバル・サービス・マネージャのリージョンであると想定されます。

ノート:

REGION パラメータが接続文字列に指定されないかぎり、単一のリージョンを持つ GDS 構成とともに 12c 以前のシン JDBC クライアントのみ使用できます。

前の例のGDSリスナーはすべて、UCPが実行されている同じリージョン(ローカル・リージョン)に属しています。高可用性を提供するため、ローカル・リージョンのすべてのGDSを使用できない場合、追加のADDRESS_LIST記述子のバディ・リージョンのGDSリスナーを指定できます。

```
(DESCRIPTION=
  (FAILOVER=on)
  (ADDRESS_LIST=
    (LOAD_BALANCE=ON)
    (ADDRESS=(global_protocol_address_information))
    (ADDRESS=(global_protocol_address_information)))
  (ADDRESS_LIST=
    (LOAD_BALANCE=ON)
    (ADDRESS=(global_protocol_address_information))
    (ADDRESS=(global_protocol_address_information)))
  (CONNECT_DATA=
   (SERVICE_NAME=global_service_name)
   (REGION=region_name)))
```

UCPリージョン用に最適にカスタマイズされるONS接続情報をUCPがGDSから自動的に取得するため、手動ONS構成は必要ありません。

ノート:

- GDS に対して自動 ONS 構成を有効にするには、UCP で高速接続フェイルオーバー(FCF)を有効にする必要があります。
- 自動 ONS 構成は、Oracle GDS および Oracle RAC でのみ動作します。単一インスタンスの Oracle Database で動作しません。

  自動 ONS 構成は、ONS ウォレットまたはキーストア・パラメータをサポートしません。アプリケーションがこれ

らのパラメータのいずれかを必要とする場合、次のいずれかの方法で明示的に ONS を構成する必要があります。

- PoolDataSource.setONSConfiguration(String)メソッドのコール
- ローカルの ONS 構成ファイルの ONS ウォレットまたはキーストア・パラメータの追加

10 アプリケーション・コンティニュイティの保証

この章では、Oracle Databaseのアプリケーション・コンティニュイティ機能に関連する次の概念について説明します。

- UCPを使用したアプリケーション・コンティニュイティの有効化の概要
- アプリケーション・コンティニュイティのデータソースの構成
- アプリケーション・コンティニュイティの接続ラベリングの使用
- アプリケーション・コンティニュイティの接続初期化コールバックの使用

10.1 UCPを使用したアプリケーション・コンティニュイティの有効化の概要

Oracle Database 12cリリース1 (12.1)では、アプリケーション・コンティニュイティ機能を導入し、アプリケーションに依存しない汎用のインフラストラクチャを提供しています。アプリケーション・コンティニュイティを使用すると、修復、構成変更またはパッチ適用後のシステム、通信またはハードウェアに関連している可能性がある計画停止または計画外停止の発生後に、アプリケーションの面から作業のリカバリが可能になります。

アプリケーション・コンティニュイティを使用するには、最初にデータ・ソースを構成する必要があります。その後に、次の2つの機能のいずれかを使用して、Universal Connection Pool (UCP)を使用するアプリケーションでアプリケーション・コンティニュイティを実装します。

- アプリケーション・コンティニュイティの接続ラベリングの使用
- アプリケーション・コンティニュイティの接続初期化コールバックの使用

関連トピック

- アプリケーション・コンティニュイティのデータソースの構成
- アプリケーション・コンティニュイティの接続ラベリングの使用
- アプリケーション・コンティニュイティの接続初期化コールバックの使用

10.2 アプリケーション・コンティニュイティのデータソースの構成

プール対応のデータソースでアプリケーション・コンティニュイティ機能を使用にするには、アプリケーションで oracle.ucp.jdbc.PoolDataSourceインタフェースに対して次のコールを実行する必要があります。

```
// pds is a PoolDataSource
pds.setConnectionFactoryClassName("oracle.jdbc.replay.OracleDataSourceImpl");
```

SIDを使用するかわりに、常にサービスに接続します。SID構文で接続する場合、アプリケーション・コンティニュイティはサポートされません。

Oracle Real Application Clusters(Oracle RAC)またはData Guardに対して実行する場合、次のコード・スニペットに示されている高速接続フェイルオーバー(FCF)も有効にする必要があります。

```
pds.setFastConnectionFailoverEnabled(true);
```

10.3 アプリケーション・コンティニュイティの接続ラベリングの使用

接続ラベリングを使用すると、アプリケーションが接続に任意の名前/値のペアを付けることができます。アプリケーションは、必要なラベルが付いた接続を接続プールにリクエストできます。

接続ラベリングは、接続リクエストごとに初期状態を設定します。アプリケーションが接続ラベリングを使用したり、接続のラベル付けを利用する場合、フェイルオーバー時にクリーン接続を初期化するには、ラベリング・コールバックをアプリケーション・コンティニュイティ用に登録する必要があります。

アプリケーション・コンティニュイティが基礎となるデータソースから新しい接続を取得するたびに、ラベリング・コールバックが実行されます。コールバックは、通常の接続チェックアウト中およびリプレイ中に実行されます。そのため、実行時に作成される状態は、リプレイ時に確実に再作成されます。初期化は冪等にする必要があります。

コールバック呼出しの最後にトランザクションが完了(コミットまたはロールバック)するかぎり、コールバックでトランザクションを実行することは有効です。アプリケーション・コンティニュイティは、このようなトランザクションを含め、コールバック実装内にコーディングされたアクションを繰り返します。停止がUCPラベリング・コールバックの実行中に発生する場合、アプリケーション・コンティニュイティがリプレイ試行の一部として複数回コールバックを実行することがあります。また、コールバック・アクションを同等にすることが重要です。

関連トピック

- UCPでの接続のラベル付け

10.4 アプリケーション・コンティニュイティの接続初期化コールバックの使用

変更できないため、アプリケーションで接続ラベリングを使用しない場合、このようなアプリケーションには接続初期化コールバックが提供されます。

登録されると、接続がプールから流用されるたびに、またリカバリ可能なエラー後に再接続が成功するたびに、初期化コールバックは実行されます。

関連トピック

- 接続初期化コールバックについて

11 シャード・データベースの共有プール

この章では、次の項でシャード・データベースのUCP共有プールについて説明します。

- データベース・シャーディングのUCP共有プールの概要
- シャード・データベースの接続リクエストの処理について
- UCPを使用した中間層ルーティング
- データベース・シャーディングのサポート用UCP API
- 中間層ルーティング・サポートのためのUCP API
- UCPシャーディングの例
- UCPを使用した中間層ルーティングの例

11.1 データベース・シャーディングのUCP共有プールの概要

Oracle Database 12cリリース2(12.2.0.1)以降、ユニバーサル接続プール(UCP)はデータベース・シャーディングをサポートします。UCPは指定されたシャーディング・キーを識別し、特定のシャードに接続します。シャーディングはグローバル・データ・サービス(GDS)を使用します。GDSは可用性、負荷、ネットワーク待機時間およびレプリケーション・ラグなどの様々なパラメータに基づいて、クライアント・リクエストを適切なデータベースにルーティングします。

関連項目:

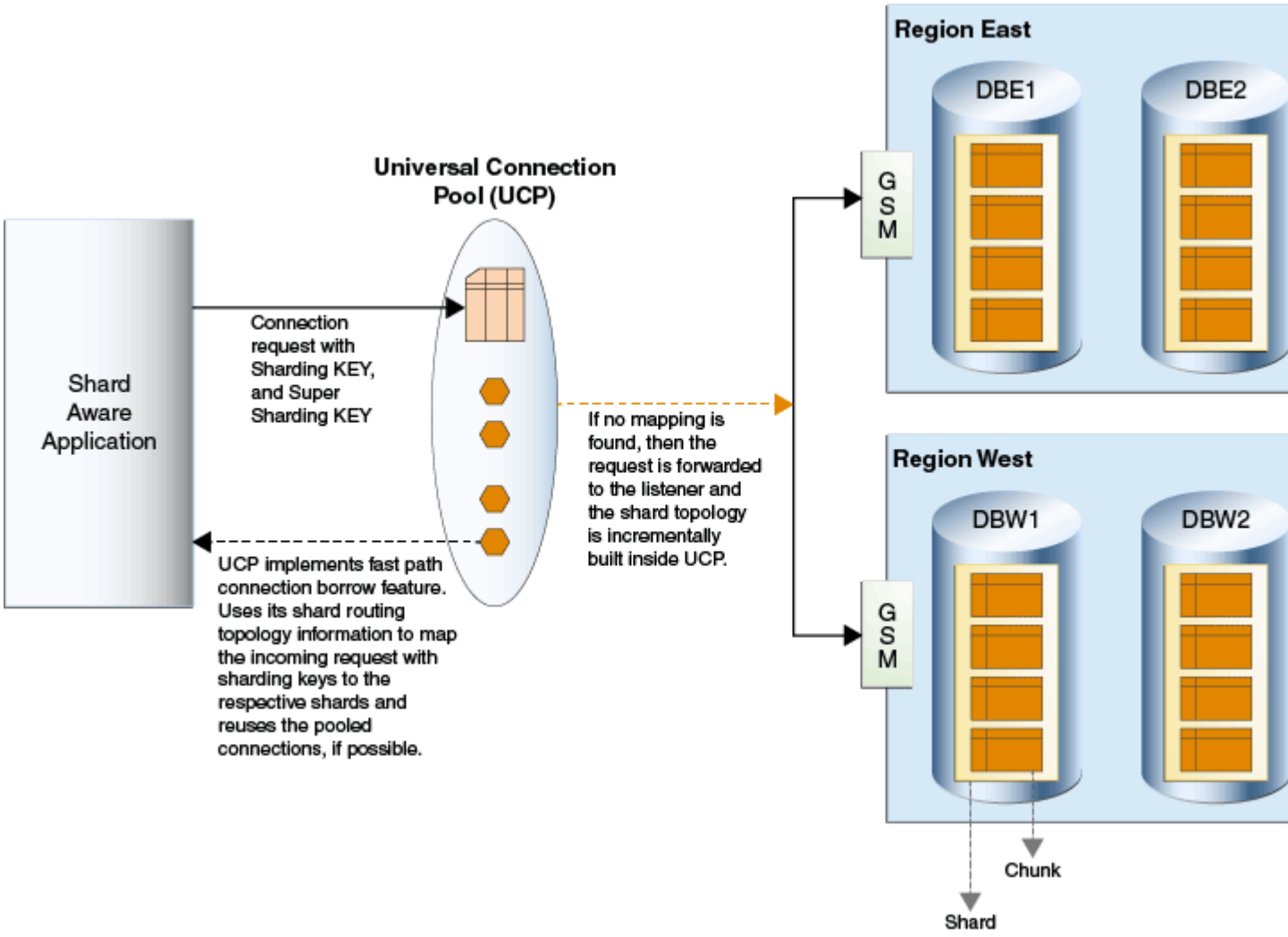
- 『Oracle Database JDBC開発者ガイド』
- Oracle Database管理者ガイド

データベース・シャーディングのUCP共有プールの使用例

この項では、データベース・シャーディングのUCP共有プールの使用例について説明します。使用例では、シャード・データベースに接続しているアプリケーションはUCPを使用して、同じ共有プール内のシャードGDSデータベースの異なるシャードおよびチャンクへの接続を格納します。アプリケーションでは、接続リクエスト中にシャーディング・キーをUCPに提供する必要があります。シャーディング・キーに基づいて、プールは接続リクエストを正しいシャードにルーティングします。データベースのシャードおよびチャンク間でデータが分散されても、それはユーザーには意識されません。UCPは再シャーディングおよびチャンク移動を透過的に処理し、エンド・ユーザーの影響を最小限に抑えます。

次の図に、使用例を示します。

図11-1 シャード・データベース・アーキテクチャを使用したユニバーサル接続プール(UCP)



関連トピック

- グローバル・データ・サービス

11.2 シャード・データベースの接続リクエストの処理について

次の項では、ユニバーサル接続プール(UCP)がシャード・データベースの接続リクエストを処理する方法について説明します。

- シャーディング・キーの作成について
- 既知のシャーディング・キーを使用したプールからの接続のチェックアウト方法
- シャーディング・キーを提供しない接続のチェックアウトについて
- クロス・シャード問合せのカタログ・データベースへの接続について
- シャードごとの接続数の構成について
- 接続チェックアウト中のプール接続選択アルゴリズム
- UCPでのエラー処理のフェイルオーバーまたは再シャーディング

11.2.1 シャーディング・キーの作成について

シャード認識アプリケーションは、シャードされたデータベースへの接続の確立に必要なシャーディング・キーおよびスーパー・シャーディング・キーを識別して作成する必要があります。これを実行するには、シャード認識アプリケーションでOracleShardingKeyおよびOracleShardingKeyBuilderインタフェースを使用する必要があります。

OracleShardingKeyBuilderは、データ型が異なる複合キーをサポートするために次のビルダー・メソッドを使用します。

```
subkey(Object subkey, java.sql.SQLTYPE subkeyDataType)
```

各サブキーのデータ型が異なる可能性がある複合シャーディング・キーを作成するには、ビルダーでsubkeyメソッドを複数回起動します。データ型は、oracle.jdbc.OracleType列挙またはjava.sql.JDBCTypeを使用して定義できます。

例11-1 シャーディング・キーの作成

次の例は、シャーディング・キーの作成方法を示します。

```
import java.sql.Connection;
import java.sql.Date;
import java.sql.SQLException;
import java.sql.Statement;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;

  public class ShardExample
  {
    public static void main(String[] args) throws SQLException
    {
      String url =
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)(PORT=3216)(PROTOCOL=tcp))(CONNECT_DATA=(SERVICE
_NAME=myservice)(REGION=east)))";
      String user="testuser1";
      String pwd = "password";

      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setURL(url);
      pds.setUser(user);
      pds.setPassword(pwd);
```

```
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setInitialPoolSize(5);
      pds.setMinPoolSize(5);
      pds.setMaxPoolSize(20);

      // build the sharding key object
      Date shardingKeyVal = new java.sql.Date(0L);
      OracleShardingKey sdkey = pds.createShardingKeyBuilder()
                                   .subkey(shardingKeyVal, OracleType.DATE)
                                   .build();

      Connection conn = pds.createConnectionBuilder()
                           .shardingKey(sdkey)
                           .build();

      Statement stmt = conn.createStatement();
      stmt.execute("... SQL statement here ...");
      stmt.close();
      conn.close();
    }
  }
```

次のコードでは、Stringデータ型とDateデータ型で構成される複合シャーディングの作成方法を示します。

```
...
Date shardingKeyVal = new java.sql.Date(0L);
...
OracleShardingKey shardingKey = datasource.createShardingKeyBuilder()
                                          .subkey("abc@xyz.com", JDBCType.VARCHAR)
                                          .subkey(shardingKeyVal, OracleType.DATE)
                                          .build();
...
```

ノート:

- 有効でサポートされているデータ型の固定のセットがあります。サポートされていないデータ型をキーとして使用すると、例外が発生します。次のリストに、サポートされているデータ型を示します。
  - OracleType.VARCHAR2/JDBCType.VARCHAR
  - OracleType.CHAR/JDBCType.CHAR
  - OracleType.NVARCHAR/JDBCType.NVARCHAR
  - OracleType.NCHAR/JDBCType.NCHAR
  - OracleType.NUMBER/JDBCType.NUMERIC
  - OracleType.FLOAT/ JDBCType.FLOAT
  - OracleType.DATE/ JDBCType.DATE
  - OracleType.TIMESTAMP/JDBCType.TIMESTAMP
  - OracleType.TIMESTAMP_WITH_LOCAL_TIME_ZONE

- OracleType.RAW

- データベースで指定されている NLS 書式に準拠したシャーディング・キーを指定する必要があります。

11.2.2 シャーディング・キーを使用したプールからの接続のチェックアウト方法

接続がUCPから流用される場合、シャード認識アプリケーションでは、PoolDataSourceクラスに存在する新しい接続ビルダーを使用して、シャーディング・キーおよびスーパー・シャーディング・キーを提供できます。シャーディング・キーが存在しない場合またはデータベース・メタデータで指定されたデータ型にマップされない場合、IllegalArgumentExceptionがスローされます。次のコードでは、シャーディング・キーを使用した接続のチェックアウト方法を示します。

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
...
Connection conn = pds.createConnectionBuilder()
// Establish a connection using sharding key and super sharding key
                     .shardingKey(shardingKey)
                     .superShardingKey(superShardingKey)
                     .build();

OracleShardingKey shardKey = pds.createShardingKeyBuilder()
// Build a compound sharding key with email address and customer ID as the two sharding keys
                                 .subkey(<email>, OracleType.VARCHAR2)
                                 .subkey(<custid>, OracleType.NUMBER)
                                 .build();

OracleShardingKey superShardKey = pds.createShardingKeyBuilder()
// Build a super sharding key with the customer region
                                      .subkey(<cust_region>, OracleType.VARCHAR2)
                                      .build();
```

ノート:

接続チェックアウト中にシャーディング・キーを指定する必要があります。それ以外の場合、エラーまたは例外がアプリケーションにスローされます。競合状態も接続使用中に例外になります。

11.2.3 シャーディング・キーを提供しない接続のチェックアウトについて

シャード・データベースに接続するためにUCPデータ・ソースを使用する場合、接続ビルダーAPIを介して接続リクエストのシャーディング・キーを提供することは必須です。シャーディング・キーを提供しない場合、例外がユーザーにスローされます。

11.2.4 複数のシャード問合せのシャード・カタログまたはコーディネータへの接続について

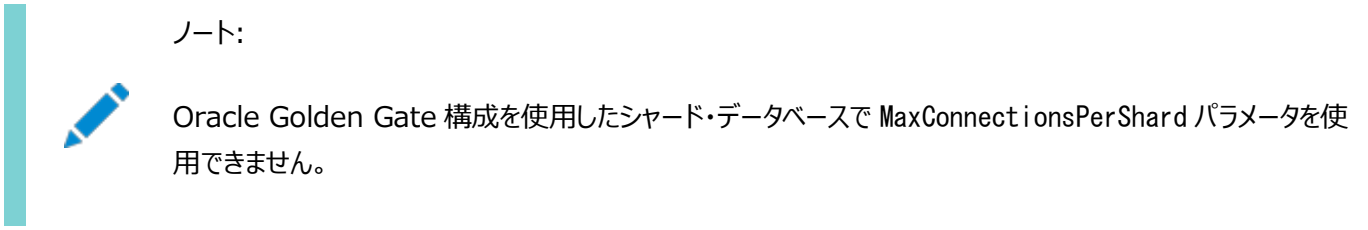
複数のシャード問合せを実行するためにシャード・カタログまたはコーディネータに接続する場合、新しいPoolDataSourceインスタンスを使用して、個別のプールを作成することをお薦めします。コーディネータ・サービスで作成されるデータソースから取得した接続で複数のシャード問合せを実行できます。コーディネータの接続リクエストには、接続ビルダーAPIのシャーディング・キーが含まれません。

11.2.5 シャードごとの接続数の構成について

UCPがシャード・データベースのプール接続に使用される場合、プールには、異なるシャードの接続が含まれます。そのため、接続を取得する場合、すべてのシャードのプール容量の相当な使用が接続されていることを確認するために、UCPはMaxConnectionsPerShardパラメータを使用します。これは、シャード・データベースの各シャードに適用されるグローバル・パラメータで、接続の合計数を指定された制限を下回るシャードに制限するために使用されます。

次の表に、このパラメータを設定および取得するAPIを示します。

メソッド	説明
poolDatasource.setMaxConnectionsPerShard(<max_connections_per_shard_limit>)	シャード当たりの最大接続数を設定します。
poolDatasource.getMaxConnectionsPerShard()	setMaxConnectionsPerShard(<max_connections_per_shard_limit>)メソッドを使用して設定された値を取得します。



ノート:

Oracle Golden Gate 構成を使用したシャード・データベースで MaxConnectionsPerShard パラメータを使用できません。

11.2.6 接続チェックアウト中のプール接続選択アルゴリズム

シャード・データベースの異なるシャードにUCPを介して新しい接続が作成される場合は常に、プールは内部的にシャード・ルーティング・キャッシュを追加的に学習して作成します。

ルーティング・キャッシュは、シャーディング・キーをキーが存在する各シャードにマップします。特定のシャーディング・キーを使用して接続リクエストに対してプールの接続を検索する場合、UCPはキャッシュを使用してリクエストを正しいシャードにリダイレクトします。高速パス接続流用と呼ばれるこの機能により、リクエストされたシャーディング・キーに基づいてプールの接続の効率的な再利用を可能にします。また、この機能により、リクエストをルーティングするためにシャード・データベースに移動することを回避できます。

11.2.7 UCPでのエラー処理のフェイルオーバーまたは再シャーディング

再シャーディングまたはフェイルオーバー・イベントの後、UCPシャード・ルーティング・キャッシュとサーバーのデータを同期しようとします。データベースの様々な変更のためにONS通知へサブスクライブすることで、キャッシュは最新のままです。

11.3 UCPを使用した中間層ルーティング

Oracle Databaseリリース18c以降では、Oracle Universal Connection Pool (UCP)に中間層ルーティング機能が導入されています。この機能は、シャーディング機能を使用するOracleのユーザーが、クライアント・アプリケーションからシャード・データベースへの専用の中間層を使用できるようにします。

通常、中間層接続プールは、データベース・リクエストを特定のシャードにルーティングします。このようなルーティングでは、各中間層接続プールから各シャードへの接続が確立されるため、データベースへの接続が多くなりすぎます。中間層ルーティング機能は、各データ・センターまたはクラウド専用の中間層(Webサーバーまたはアプリケーション・サーバー)を持ち、クライアント・データ(クライアント・シャーディング・キーに対応する)を含むシャードが存在する関連する中間層にクライアント・リクエストを直接ルーティングすることで、この問題を解決します。

UCPのOracleShardRoutingCacheクラスは、クライアント・リクエストを適切な中間層にルーティングするために使用できる中間層ルーティングAPIを提供しています。このクラスのインスタンスは、UCPの内部シャード・ルーティング・キャッシュを表します。これは、シャーディング・カタログのユーザー、パスワード、URLなどの接続プロパティを指定することで作成できます。Oracle Databaseリリース19c以降では、新しい接続プロパティserviceNameも指定する必要があります。これはグローバル・サービスの名前です。

ルーティング・キャッシュは、シャーディング・カタログに接続し、シャード・マッピング・トポロジへのキーを取得してキャッシュに格納します。getShardInfoForKey（shardKey, superShardKey）メソッドは、UCPのルーティング・キャッシュを使用して、指定されたシャーディング・キーのシャードに関する情報を取得します。ShardInfoインスタンスは、シャードの一意のシャード名と優先度をカプセル化します。中間層APIを使用するアプリケーションは、返された一意のシャード名値を、指定されたシャードへの接続がある中間層にマップできます。

ルーティング・キャッシュは、各ONSイベントをサブスクライブすることによって、チャンクの移動または分割時に自動的にリフレッシュまたは更新されます。

11.4 データベース・シャーディングのサポート用UCP API

UCPConnectionビルダー・クラス

UCPConnectionBuilderクラスは、username、passwordおよびlabel以外の追加パラメータで接続オブジェクトを作成するために使用されます。ビルダーを使用するには、接続リクエストの一部である必要がある各パラメータの対応するビルダー・メソッド、buildメソッドの順にコールする必要があります。ビルダー・メソッドをコールする順序は重要ではありません。ただし、同じビルダー属性を複数回適用する場合、最新の値のみが接続の作成時に考慮されます。

構文

```
public abstract class UCPConnectionBuilder<S> implements
OracleConnectionBuilder<UCPConnectionBuilder<S>,S>
```

UCPConnectionBuilderクラスは、validateメソッドおよび特定のユーザーのデータを設定するための複数のコンストラクタも提供します。

例11-2 接続ビルダーの作成

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
..//set the required properties on the datasource
ShardingKey superShardingKey = ds.createShardingKeyBuilder()

                                .subkey("EASTERN_REGION", JDBCType.VARCHAR)

                                .build();
ShardingKey superShardingKey = ds.createShardingKeyBuilder()

                                .subkey("PITTSBURGH_BRANCH", JDBCType.VARCHAR)

                                .build();
Connection conn = pds.createConnectionBuilder()
                                                        .shardingKey(superShardingKey)
                                                        .superShardingKey(superShardingKey)
                                                        .build();
```

PoolDataSourceインタフェースの新しいメソッド

次のメソッドがoracle.ucp.jdbc.PoolDataSourceインタフェースで導入されました。

```
/**
 * Creates a new UCPConnectionBuilder instance.
 *
 * @param <S>
 * Connection type for this ConnectionBuilder
 * @param <B>
 * Builder type to use
 * @return The OracleConnectionBuilder instance that was created
 */
 public UCPConnectionBuilder createConnectionBuilder();


/**
 * Creates a new OracleShardingKeyBuilder instance
 *
 * @return The OracleShardingKeyBuilder instance that was created
 */
 public default OracleShardingKeyBuilder createShardingKeyBuilder() {
```

```
return new OracleShardingKeyBuilderImpl();
}
```

PoolXADataSourceインタフェースの新しいメソッド

次のメソッドがoracle.ucp.admin.UniversalConnectionPoolManagerインタフェースで導入されました。

```
/**
 * Creates a new XAConnectionBuilder instance.
 *
 * @return The XAConnectionBuilder instance that was created
 */
 public UCPXAConnectionBuilder createXAConnectionBuilder();
```

11.5 中間層ルーティング・サポートのためのUCP API

OracleShardRoutingCacheクラス

このクラスは、UCPの内部シャード・ルーティング・キャッシュを拡張し、WebLogic Server、中間層ルーターまたはロード・バランサで基本ルーティング・キャッシュ機能を使用できるようにします。

publicクラスOracleShardRoutingCacheはShardRoutingCacheを拡張します

このクラスは、OracleShardRoutingCache(Properties dataSourceProps)メソッドおよびSet<ShardInfo> getShardInfoForKey(OracleShardingKey key, OracleShardingKey superKey)メソッドを提供します。

ShardInfoインタフェース

ShardInfoインタフェース・インスタンスは、一意のシャード名と優先度をカプセル化します。一意のシャード名は、特定のシャードに接続する中間層サーバーにマップできます。

11.6 UCPシャーディングの例

例

次のコードでは、UCPシャーディングAPIの使用方法を示します。

例11-3 UCPシャーディングの例

```
PoolDataSource pds = new PoolDataSourceImpl();
pds.setURL(url);
pds.setUser("system");
pds.setPassword("manager");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");

OracleShardingKey employeeNamekey =
    pds.createShardingKeyBuilder()
       .subkey("Mary", JDBCType.VARCHAR)  // First Name
       .subkey("Claire", JDBCType.VARCHAR)  // Last Name
       .build();

OracleShardingKey locationKey = pds.createShardingKeyBuilder()
                                   .subkey("US", JDBCType.VARCHAR)//Location
                                   .build();

 OracleConnection connection = pds.createConnectionBuilder()
                                  .shardingKey(employeeNamekey)
                                  .superShardingKey(locationKey)
                                  .build();
```

11.7 UCPを使用した中間層ルーティングの例

次の例は、UCPの中間層ルーティングAPIの使用について説明しています。

例11-4 UCPを使用した中間層ルーティングの例

```
import java.sql.SQLException;
import java.util.Properties;
import java.util.Random;
import java.util.Set;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.ucp.UniversalConnectionPoolException;
import oracle.ucp.routing.ShardInfo;
import oracle.ucp.routing.oracle.OracleShardRoutingCache;

/**
 * The code example illustrates the usage of the middle-tier routing feature of UCP.
 * The API accepts sharding key as input and returns the set of ShardInfo
 * instances mapped to the sharding key. The ShardInfo instance encapsulates
 * unique shard name and priority. The unique shard name then can be mapped
 * to a middle-tier server that connects to a specific shard.
 *
 */
public class MidtierShardingExample {

  private static String user = "testuser1";
  private static String password = "testuser1";

  // catalog DB URL
  private static String url = "jdbc:oracle:thin:@//hostName:1521/catalogServiceName";
  private static String region = "regionName";

  public static void main(String args[]) throws Exception {
    testMidTierRouting();
  }

  static void testMidTierRouting() throws UniversalConnectionPoolException,
      SQLException {

    Properties dbConnectProperties = new Properties();
    dbConnectProperties.setProperty(OracleShardRoutingCache.USER, user);
    dbConnectProperties.setProperty(OracleShardRoutingCache.PASSWORD, password);
    // Mid-tier routing API accepts catalog DB URL
    dbConnectProperties.setProperty(OracleShardRoutingCache.URL, url);

    // Region name is required to get the ONS config string
    dbConnectProperties.setProperty(OracleShardRoutingCache.REGION, region);

    OracleShardRoutingCache routingCache = new OracleShardRoutingCache(
        dbConnectProperties);

    final int COUNT = 10;
    Random random = new Random();

    for (int i = 0; i < COUNT; i++) {
      int key = random.nextInt();
      OracleShardingKey shardKey = routingCache.getShardingKeyBuilder()
```

```
            .subkey(key, OracleType.NUMBER).build();
        OracleShardingKey superShardKey = null;

        Set<ShardInfo> shardInfoSet = routingCache.getShardInfoForKey(shardKey,
            superShardKey);

        for (ShardInfo shardInfo : shardInfoSet) {
          System.out.println("Sharding Key=" + key + " Shard Name="
              + shardInfo.getName() + " Priority=" + shardInfo.getPriority());
        }
      }

    }
}
```

12 接続プールの診断

次のパラメータは、ユニバーサル接続プール(UCP)を診断するために使用されます。

- プールの統計情報
- ダイナミック・モニタリング・サービス・メトリック
- Oracle RACの統計情報の表示について
- UCPでのロギングの概要
- 例外とエラー・コード

12.1 プールの統計情報

Universal Connection Pool (UCP)では、接続プールの実行時統計情報を提供します。これらの統計情報は次の2つのカテゴリに分かれます。

- 非累積

  これらの統計情報は現在実行中の接続プール・インスタンスにのみ適用されます。

- 累積

  これらの統計情報はプールの起動または停止の複数のサイクルにわたって収集されます。

oracle.ucp.UniversalConnectionPoolStatisticsインタフェースは、接続プール統計情報を問い合せるために使用されるメソッドを備えています。このインタフェースのメソッドは、oracle.ucp.jdbc.PoolDataSource.getStatisticsメソッドを使用して、プール対応のデータソースおよびプール対応のXAデータソースからコールできます。次に例を示します。

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
...
...
int totalConnsCount = pds.getStatistics().getTotalConnectionsCount();
System.out.println("The total connetion count in the pool is "+ totalConnsCount +".");
```

oracle.ucp.jdbc.PoolDataSource.getStatisticsメソッドは、それ自体でコールすることも可能であり、接続プールの全統計情報を1つのStringとして戻します。

12.2 ダイナミック・モニタリング・サービス・メトリック

UCPは、すべてのプールの統計情報をダイナミック・モニタリング・サービス(DMS)メトリックの形式にするようにサポートしています。これらのDMSメトリックを収集し利用するには、アプリケーションのクラス・パスにdms. jarファイルを含める必要があります。

UCPは、プール・マネージャ・インタフェースとプール・マネージャMBeanの両方でDMSメトリックの収集をサポートしています。UnversalConnectionPoolManager.startMetricsCollectionメソッドを使用すると、特定の接続プール・インスタンスに対してDMSメトリックの収集を開始でき、UnversalConnectionPoolManager.stopMetricsCollectionメソッドを使用すると、DMSメトリックの収集を停止できます。メトリックの更新間隔は、UnversalConnectionPoolManager.setMetricUpdateIntervalメソッドを使用して指定できます。プール・マネージャMBeanは同様の操作をエクスポートします。

12.3 Oracle RACの統計情報の表示について

UCPは、一連のOracle RACの実行時統計情報を提供します。この統計情報は、接続プールがOracle RAC機能をどの程度利用しているか判断するために使用されます。また、Oracle RAC機能を使用するために接続プールが適切に構成されているかどうかの判断に役立てるためにも使用されます。統計情報には、FCF処理情報、実行時接続ロード・バランスの成否率、アフィニティ・コンテキストの成否率がレポートされます。

oracle.ucp.jdbc.oracleパッケージにあるOracleJDBCConnectionPoolStatisticsインタフェースは、Oracle RACの統計情報を接続プールに問い合せるために使用されるメソッドを備えています。このインタフェースのメソッドは、データソースのgetStatisticsメソッドを使用して、プール対応のデータソースおよびプール対応のXAデータソースからコールできます。次に例を示します。

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();
...

Long rclbS = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getSuccessfulRCLBBasedBorrowCount();
System.out.println("The RCLB success rate is "+rclbS+".");
```

データソースのgetStatisticsメソッドは、それ自身でコールすることも可能であり、接続プールの全統計情報を1つのStringとして戻し、Oracle RACの統計情報を組み込みます。

12.3.1 高速接続フェイルオーバーの統計情報

getFCFProcessingInfoメソッドは、最新の高速接続フェイルオーバー(FCF)の試行に関する情報をStringの形式で提供します。通常、FCFの情報は、FCFの問題の診断に役立てるために使用されます。この情報は、各FCFの試行結果(成功または失敗)、関連するOracle RACインスタンス、クリーンアップされた接続数、FCFの試行の失敗をトリガーした例外などで構成されます。次の例では、getFCFProcessingInfoメソッドの使用方法を示します。

```
Sting fcfInfo = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getFCFProcessingInfo();
System.out.println("The FCF information: "+fcfInfo+".");
```

次の例では、getFCFProcessingInfo()メソッドの出力文字列を示します。

```
    Oct 28, 2008 12:34:02 SUCCESS <Reason:planned> <Type:SERVICE_UP> ¥
      <Service:"svvc1"> <Instance:"inst1"> <Db:"db1"> ¥
      Connections:(Available=6 Affected=2 FailedToProcess=0 MarkedDown=2 Closed=2) ¥
      (Borrowed=6 Affected=2 FailedToProcess=0 MarkedDown=2 MarkedDeferredClose=0 Closed=2) ¥
      TornDown=2 MarkedToClose=2 Cardinality=2
    ...
    Oct 28, 2008 12:09:52 SUCCESS <Reason:unplanned> <Type:SERVICE_DOWN> ¥
      <Service:"svc1"> <Instance:"inst1"> <Db:"db1"> ¥
      Connections:(Available=6 Affected=2 FailedToProcess=0 MarkedDown=2 Closed=2) ¥
      (Borrowed=6 Affected=2 FailedToProcess=0 MarkedDown=2 MarkedDeferredClose=0 Closed=2)
    ...
    Oct 28, 2008 11:14:53 FAILURE <Type:HOST_DOWN> <Host:"host1"> ¥
      Connections:(Available=6 Affected=4 FailedToProcess=0 MarkedDown=4 Closed=4) ¥
      (Borrowed=6 Affected=4 FailedToProcess=0 MarkedDown=4 MarkedDeferredClose=0 Closed=4)
```

ロギングを有効にしている場合、前述の情報はUCPログでも利用でき、FCFの結果を検証できます。

12.3.2 実行時接続ロード・バランスの統計情報

実行時接続ロード・バランスの統計情報は、接続プールがOracle RACの実行時接続ロード・バランシング機能を効率的に利用しているかどうかの判断に使用されます。この統計情報には、実行時接続ロード・バランシングのアルゴリズムを利用できたリクエストの数と、アルゴリズムを利用できなかったリクエストの数がレポートされます。getSuccessfulRCLBBasedBorrowCountメソッドとgetFailedRCLBBasedBorrowCountメソッドが、それぞれの統計情報の取得に使用されます。次の例では、getFailedRCLBBasedBorrowCountメソッドの使用方法を示します。

```
Long rclbF = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getFailedRCLBBasedBorrowCount();
System.out.println("The RCLB failure rate is: "+rclbF+".");
```

失敗率が高い場合は、Oracle RACロード・バランシング・アドバイザまたは接続プールが適切に構成されていないことを示している可能性があります。

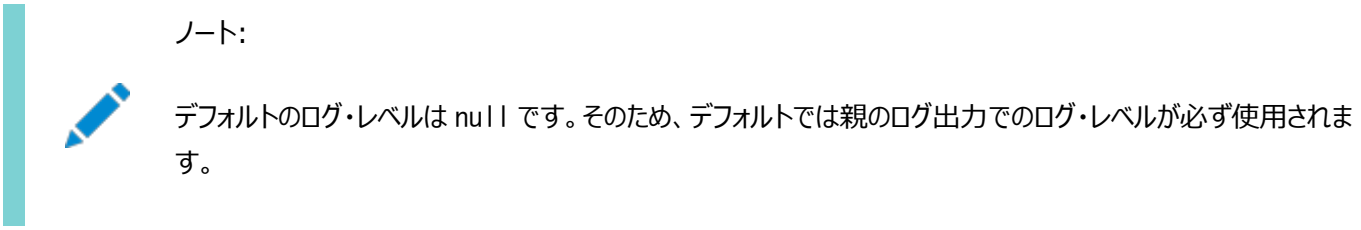
12.3.3 接続アフィニティの統計情報

接続アフィニティの統計情報は、接続プールが接続アフィニティを効率的に利用しているかどうかの判断に使用されます。この統計情報には、アフィニティ・コンテキストと一致した流用リクエストの数と、アフィニティ・コンテキストと一致しなかったリクエストの数がレポートされます。getSuccessfulAffinityBasedBorrowCountメソッドとgetFailedAffinityBasedBorrowCountメソッドが、それぞれの統計情報の取得に使用されます。次の例では、getFailedAffinityBasedBorrowCountメソッドの使用方法を示します。

```
Long affF = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getFailedAffinityBasedBorrowCount();
System.out.println("The connection affinity failure rate is: "+affF+".");
```

12.4 UCPでのロギングの概要

UCPは、JDKロギング機能(java.util.logging)を利用しています。ロギングは、デフォルトでは無効になっており、ログ・メッセージを出力するために構成する必要があります。ロギングは、ログ構成ファイルを使用するか、APIレベルの構成を使用して、構成できます。



ノート:

デフォルトのログ・レベルは null です。そのため、デフォルトでは親のログ出力でのログ・レベルが必ず使用されます。

12.4.1 ロギング・プロパティ・ファイルの使用方法

ロギングは、プロパティ・ファイルを使用して構成できます。プロパティ・ファイルの場所は、ロギング構成ファイル・プロパティのJavaプロパティとして設定する必要があります。次に例を示します。

```
java -Djava.util.logging.config.file=log.properties
```

ロギング・プロパティ・ファイルは、ログの書込みに使用するハンドラ、ログの書式設定に使用するフォーマッタ、デフォルトのログ・レベルの他、特定のパッケージおよびクラスのログ・レベルを定義します。次に例を示します。

```
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

oracle.ucp.level = FINEST
oracle.ucp.jdbc.PoolDataSource = WARNING
```

カスタム・フォーマッタは、UCPに同梱されており、フォーマッタ・プロパティの値として入力できます。次に例を示します。

```
java.util.logging.ConsoleHandler.formatter = oracle.ucp.util.logging.UCPFormatter
```

Oracle Technology Network(OTN)から、UCPに用意されているucpdemos.jarファイルをダウンロードすることもできます。このファイルにはサンプルのロギング・プロパティ・ファイルのリストがあります。たとえば、このファイルには、高速接続フェイルオーバー(FCF)機能のトラブルシューティングに使用できるロギング・プロパティ・ファイルがあります。

12.4.2 UCPおよびJDK APIの使用

ロギングは、UCP APIまたはJDK APIのいずれかを使用して動的に構成できます。UCP APIを使用する場合は、接続プール・マネージャを使用してロギングを構成します。JDKを使用する場合は、java.util.logging実装を使用してロギングを構成します。

次の例では、UCP APIを使用してロギングを構成しています。

```
UniversalConnectionPoolManager mgr = UniversalConnectionPoolManagerImpl.
getUniversalConnectionPoolManager();

mgr.setLogLevel(Level.FINE);
```

次の例では、JDKロギング実装を直接使用しています。

```
Logger.getLogger("oracle.ucp").setLevel(Level.FINEST);
Logger.getLogger("oracle.ucp.jdbc.PoolDataSource").setLevel(Level.FINEST);
```

12.4.3 実行時における機能固有のロギングの有効化または無効化

Oracle Database 12cリリース2 (12.2.0.1)以降、UCPでは実行時に選択した機能に対してロギングの有効化または無効化がサポートされています。たとえば、ロード・バランシング機能のみのロギングを有効にして、UCPの他の機能のロギングを無効にすることができます。また、同じ実行で高速フェイルオーバー機能のロギングを有効にして、ロード・バランシング機能のロギングを無効にすることもできます。

すべての機能のロギングはデフォルトで有効になっています。

UCPのロギングの切替え機能はOracleDiagnosabilityMBeanに含まれています。このBeanを使用するには、JConsoleを起動してアプリケーションに接続します。

サポートされている機能の表示

サポートされている機能のリストを表示するには、次のメソッドを使用します。

```
getTraceController().getSupportedFeatures()
```

有効になっている機能の表示

現在有効になっている機能のリストを表示するには、次のメソッドを使用します。

```
getTraceController().getEnabledFeatures()
```

機能のロギングの有効化

特定の機能またはすべての機能のロギングを有効にするには、次のようにtraceメソッドを使用します。

```
trace(boolean enable, String feature_name)
trace(boolean enable, ALL)
```

機能のロギングの無効化

特定の機能またはすべての機能のロギングを無効にするには、次のようにtraceメソッドを使用します。

```
trace(boolean disable, String feature_name)
trace(boolean disable, ALL)
```

ロギングの一時停止および再開

ロギングを一時停止および再開するには、それぞれ、次のメソッドを使用します。

```
suspend()
resume()
```

12.4.4 機能固有のロギングのロギング・プロパティ・ファイルの使用について

Oracle Database 12cリリース2 (12.2.0.1)以降、ロギング・プロパティ・ファイルにプロパティを追加することにより、特定の機能のロギングを有効または無効にすることができます。ロギングはデフォルトですべての機能について有効になっています。そうでない場合、次の構文(clio.feature.all = on)を使用してすべての機能のロギングを有効にすることができます。ロギングの機能固有の有効化では、次の項で示されているようにプロパティを使用できます。

機能ベース粒度のサポートされている機能

```
clio.feature.pool_statistics          = on
```

```
clio.feature.check_in                 = on
clio.feature.check_out                = on
clio.feature.labeling                 = on
clio.feature.conn_construction        = on
clio.feature.conn_destruction         = on
clio.feature.high_availability        = on
clio.feature.load_balancing           = on
clio.feature.transaction_affinity     = on
clio.feature.web_affinity            = on
clio.feature.data_affinity           = on
clio.feature.conn_harvesting          = on
clio.feature.ttl_conn_timeout         = on
clio.feature.abandoned_conn_timeout   = on
clio.feature.admin                    = on
clio.feature.sharding                 = on
```

12.4.5 サポートされるログ・レベル

次に、JDBCでサポートされる各ログ・レベルを示します。FINEよりも低いレベルでは、ユーザーにとって重要とはかぎらない出力が生成されます。FINERよりも低いレベルでは、非常に大量の出力が生成されます。

- INTERNAL_ERROR – 内部エラー
- SEVERE – SQL例外
- WARNING: SQL警告およびその他の隠れた問題
- INFO: パブリック・イベント(接続の試行やOracle RACイベントなど)
- CONFIG – SQL文
- FINE – パブリックAPI
- TRACE_10: 内部イベント
- FINER – 内部API
- TRACE_20: 内部デバッグ
- TRACE_30: 大量の内部API
- FINEST: 大量の内部デバッグ

12.5 例外とエラー・コード

多くのUCPメソッドは、例外チェーンがサポートされたUniversalConnectionPoolExceptionをスローします。スローした例外で、printStackTraceメソッドをコールすると、例外の根本原因を特定できます。UniversalConnectionPoolExceptionには、45000から45499までの範囲の標準のOracleエラー・コードが含まれています。getErrorCodeメソッドを使用すると、例外のエラー・コードを取得できます。

A エラー・コード・リファレンス

この付録では、ユニバーサル接続プール(UCP)のエラー・メッセージ、接続プール・レイヤー用のUCPエラー・メッセージおよびJDBCデータソースと動的プロキシ用のUCPエラー・メッセージの一般構造について簡単に説明します。この付録の構成は次のとおりです。

- UCPエラー・メッセージの一般構造
- 接続プール・レイヤーのエラー・メッセージ
- JDBCデータソースおよび動的プロキシのエラー・メッセージ

2つのメッセージ・リストはエラー・メッセージ番号順にソートされています。

A.1 UCPエラー・メッセージの一般構造

UCPエラー・メッセージの一般構造は、次のようにメッセージの末尾にコロンを付けて、実行時情報を追加できます。

```
<error_message>:<extra_info>
```

たとえば、closed statementエラーは次のように表示されることがあります。

```
Closed Statement:next
```

これは(結果セット・オブジェクトの)nextメソッドのコール中に例外がスローされたことを示します。

場合によっては、スタック・トレースに同様の情報が見つかることがあります。

A.2 接続プール・レイヤーのエラー・メッセージ

この項では、接続プール・レイヤーのUCPエラー・メッセージを示します。

表A-1 接続プール・レイヤーのエラー・メッセージ

エラー・メッセージ番号	メッセージ
UCP-45001	ユニバーサル接続プールの内部エラー
UCP-45002	ユニバーサル接続プールに使用可能な接続がありません
UCP-45003	ユニバーサル接続プールがすでに存在します
UCP-45004	接続取得情報が無効です
UCP-45005	コールバックはすでに登録されています
UCP-45006	ユニバーサル接続プールの構成が無効です
UCP-45051	非アクティブ接続タイムアウトのタイマーのスケジュールに失敗しました

エラー・メッセージ番号	メッセージ
UCP-45052	中止接続タイムアウトのタイマーのスケジュールに失敗しました
UCP-45053	TTL 接続タイムアウトのタイマーのスケジュールに失敗しました
UCP-45054	ユニバーサル接続プールは NULL にできません
UCP-45055	使用可能な接続の削除中にエラーが発生しました
UCP-45057	AvailableConnections オブジェクトは NULL にできません
UCP-45058	Failoverable オブジェクトは NULL にできません
UCP-45059	MaxPoolsize は、0(ゼロ)に設定されます。返す接続がありません
UCP-45060	ライフサイクルの状態が無効です。ユニバーサル接続プールの状態を確認してください
UCP-45061	ユニバーサル接続プールが起動していません。ユニバーサル接続プールを起動してからアクセスしてください
UCP-45062	使用可能な接続の収集は、ユニバーサル接続プールが初期化状態の場合にのみ設定可能です
UCP-45063	接続の取得試行中にユニバーサル接続プールが停止しました
UCP-45064	ユニバーサル接続プールのすべての接続が使用中です
UCP-45065	接続流用が NULL を返しました
UCP-45091	接続ラベリング・コールバックはすでに登録されています
UCP-45092	ラベリング・コールバックが登録されていないラベル付き接続を流用しています
UCP-45093	ラベルなし接続をリクエストしましたが、ラベル付き接続を流用しています
UCP-45097	接続獲得タイマーのスケジュールに失敗しました
UCP-45100	ConnectionFactoryAdapter が NULL を返しました
UCP-45103	ConnectionFactoryAdapter は DataSourceConnectionFactoryAdapter のインスタンスである必要があります

エラー・メッセージ番号	メッセージ
UCP-45104	ConnectionFactoryAdapter オブジェクトは NULL にできません
UCP-45105	ConnectionFactoryAdapter は ConnectionPoolDataSourceConnectionFactoryAdapter のインスタンスである必要があります
UCP-45106	ConnectionFactoryAdapter は XADataSourceConnectionFactoryAdapter のインスタンスである必要があります
UCP-45150	UniversalPooledConnection は NULL にできません
UCP-45152	UniversalPooledConnectionStatus オブジェクトは NULL にできません
UCP-45153	接続ラベル・キーは NULL または空の文字列にできません
UCP-45154	接続ラベリング操作はクローズされた接続では起動できません
UCP-45155	接続獲得コールバックはすでに登録されています
UCP-45156	中止接続タイムアウト・コールバックはすでに登録されています
UCP-45157	TTL 接続タイムアウト・コールバックはすでに登録されています
UCP-45201	接続ラベル・キーは NULL または空の文字列にできません
UCP-45202	ConnectionRetrievalInfo オブジェクトのクローニングに失敗しました
UCP-45203	接続リクエスト情報が NULL です
UCP-45251	ConnectionPoolDataSource は NULL にできません
UCP-45252	ConnectionRetrievalInfo オブジェクトが無効です
UCP-45253	ConnectionPoolDataSource からの PooledConnection の取得中に SQLException が発生しました
UCP-45254	接続タイプが無効です。javax.sql.PooledConnection である必要があります
UCP-45255	PooledConnection のクローズ中に SQLException が発生しました

エラー・メッセージ番号	メッセージ
UCP-45256	データ・ソースは NULL にできません
UCP-45257	データ・ソースから接続を取得できません
UCP-45258	接続タイプが無効です。java.sql.Connection である必要があります
UCP-45259	プロキシへの接続は java.sql.Connection のインスタンスである必要があります
UCP-45260	XADatasource は NULL にできません
UCP-45261	XADataSource からの XAConnection の取得中に SQLException が発生しました
UCP-45262	接続タイプが無効です。javax.sql.XAConnection である必要があります
UCP-45263	XAConnection のクローズ中に SQLException が発生しました
UCP-45264	接続は NULL にできません
UCP-45265	プロキシへの接続は java.sql.Statement のインスタンスである必要があります
UCP-45266	プロキシへの文は java.sql.ResultSet のインスタンスである必要があります
UCP-45267	プロキシへの接続は javax.sql.XAConnection のインスタンスである必要があります
UCP-45268	Driver 引数は null にできません
UCP-45269	URL 引数は null にできません
UCP-45301	フェイルオーバー情報への接続を取得できません
UCP-45302	フェイルオーバー情報を取得する SQL 問合せを実行できません
UCP-45303	フェイルオーバー情報の取得中に SQLException が発生しました
UCP-45304	イベント・タイプは NULL にできません
UCP-45305	イベント・タイプが無効です。イベント・タイプは database/event/host または database/event/service である必要があります

エラー・メッセージ番号	メッセージ
UCP-45306	フェイルオーバー・イベント・タイプが無効です。OracleFailoverEvent である必要があります
UCP-45307	アフィニティ・コンテキストが無効です。OracleConnectionAffinityContext である必要があります
UCP-45308	リモート ONS サブスクリプションのフェイルオーバーの有効化中に例外が発生しました
UCP-45350	ユニバーサル接続プールはユニバーサル接続プール・マネージャにすでに存在しています。ユニバーサル接続プールをユニバーサル接続プール・マネージャに追加できません
UCP-45351	ユニバーサル接続プール・マネージャにユニバーサル接続プールが見つかりません。ユニバーサル接続プール・マネージャにユニバーサル接続プールを登録してください
UCP-45352	ユニバーサル接続プール・マネージャのインスタンスを取得できません
UCP-45353	ユニバーサル接続プール・マネージャの MBean インスタンスを取得できません
UCP-45354	MBean ObjectName の形式が正しくありません。正しい形式を使用して MBean の ObjectName を構成してください
UCP-45355	MBean の登録または登録解除中に MBean 例外が発生しました
UCP-45356	MBean は MBeanServer にすでに存在しています。別の名前を使用して MBean を登録してください
UCP-45357	JMX に準拠していない MBean オブジェクトの MBean Server への登録中に例外が発生しました
UCP-45358	指定された MBean はリポジトリに存在しません
UCP-45359	無効なターゲット・オブジェクト・タイプが指定されています。管理リソースを確認してください
UCP-45360	無効な MBean 記述子が指定されています。ユニバーサル接続プール・マネージャの MBean 記述子を確認してください
UCP-45361	ユニバーサル接続プール・マネージャの MBean の MBeanInfo の作成中にランタイム例外が発生しました
UCP-45362	ユニバーサル接続プール・マネージャの MBean のコンストラクタ情報の作成中にランタイム例外が発生しました

エラー・メッセージ番号	メッセージ
UCP-45363	ユニバーサル接続プール・マネージャの MBean の属性情報の作成中にランタイム例外が発生しました
UCP-45364	ユニバーサル接続プール・マネージャの MBean の操作情報の作成中にランタイム例外が発生しました
UCP-45365	ユニバーサル接続プールは ConnectionConnectionPool または OracleConnectionConnectionPool のインスタンスである必要があります
UCP-45366	無効な MBean 記述子が指定されています。JDBC ユニバーサル接続プールの MBean 記述子を確認してください
UCP-45367	JDBC ユニバーサル接続プールの MBean の MBeanInfo の作成中にランタイム例外が発生しました
UCP-45368	JDBC ユニバーサル接続プールの MBean のコンストラクタ情報の作成中にランタイム例外が発生しました
UCP-45369	JDBC ユニバーサル接続プールの MBean の属性情報の作成中にランタイム例外が発生しました
UCP-45370	JDBC ユニバーサル接続プールの MBean の操作情報の作成中にランタイム例外が発生しました
UCP-45371	ユニバーサル接続プールの MBean の属性情報の作成中にランタイム例外が発生しました
UCP-45372	ユニバーサル接続プールの MBean の操作情報の作成中にランタイム例外が発生しました
UCP-45373	無効な MBean 記述子が指定されています。ユニバーサル接続プールの MBean 記述子を確認してください
UCP-45374	ユニバーサル接続プールの MBean の MBeanInfo の作成中にランタイム例外が発生しました
UCP-45375	UCP メトリック収集を停止できません。メトリック収集の停止中か、ナウンまたはセンサーの破棄中に例外が発生しました。
UCP-45376	メトリック更新タイマー・タスクのスケジュールに失敗しました
UCP-45377	UCP メトリック・センサーの更新中に問題が発生しました
UCP-45378	ユニバーサル接続プールが OracleJDBCConnectionPool のインスタンスではないため、ONSConfiguration プロパティにアクセスできません

エラー・メッセージ番号	メッセージ
UCP-45379	ユニバーサル接続プールの MBean で接続プール名を設定できません。重複しないように接続プール名を確認してください
UCP-45380	MBean オブジェクトが NULL です
UCP-45381	MBean オブジェクト名が NULL です
UCP-45382	MBean 表示名が NULL です
UCP-45383	ユニバーサル接続プール・マネージャでプール作成用のアダプタが無効です
UCP-45384	ユニバーサル接続プール・マネージャの MBean でプール作成用のアダプタが無効です
UCP-45385	ユニバーサル接続プール・マネージャでプールを作成中にエラーが発生しました
UCP-45386	ユニバーサル接続プール・マネージャの MBean でプールを作成中にエラーが発生しました
UCP-45401	待機スレッドの LO 水位標は負にできません
UCP-45402	待機スレッドの HI 水位標は負にできません
UCP-45403	ワーカー・スレッド合計の上限は負にできません
UCP-45404	キューのポーリング・タイムアウトは負にできません
UCP-45405	待機スレッドの HI 水位標は LO 水位標より低くできません
UCP-45406	ワーカー・スレッド合計の上限は待機スレッドの上限よりも大きくできません
UCP-45407	エラー番号が範囲外です
UCP-45408	ログ出力が NULL であるため、操作は無効です

A.3 JDBCデータソースおよび動的プロキシのエラー・メッセージ

この項では、JDBCデータソースのUCPエラー・メッセージおよび動的プロキシのエラー・メッセージを示します。

表A-2 JDBCデータソースおよび動的プロキシのエラー・メッセージ

エラー・メッセージ番号	メッセージ

エラー・メッセージ番号	メッセージ
SQL-0	ユニバーサル接続プールを起動できません
SQL-1	ユニバーサル接続プールを作成できません
SQL-2	最小プール・サイズが無効です
SQL-3	最大プール・サイズが無効です
SQL-4	非アクティブ接続タイムアウトが無効です
SQL-5	接続待機タイムアウトが無効です
SQL-6	TTL 接続タイムアウトが無効です
SQL-7	中止接続タイムアウトが無効です
SQL-8	タイムアウト・チェック間隔が無効です
SQL-9	フェイルオーバーの有効化に失敗しました
SQL-10	maxStatements 値の設定に失敗しました
SQL-11	検証用の SQL 文字列の設定に失敗しました
SQL-12	接続獲得トリガー数が無効です
SQL-13	接続獲得最大数が無効です
SQL-14	ユニバーサル接続プールはすでに作成されています。ユニバーサル接続プールは再作成できません
SQL-15	ユニバーサル接続プールの破棄中に例外が発生しました
SQL-16	操作は Oracle の接続プールにのみ適用されます
SQL-17	ONS 構成文字列の設定中に例外が発生しました
SQL-18	ラベリング・コールバックの登録に失敗しました
SQL-19	ラベリング・コールバックの削除に失敗しました

エラー・メッセージ番号	メッセージ
SQL-20	アフィニティ・コールバックの登録に失敗しました
SQL-21	アフィニティ・コールバックの削除に失敗しました
SQL-22	ユニバーサル接続プールの構成が無効です
SQL-23	指定されたファクトリ・クラス名を持つファクトリ・クラス・インスタンスの作成に失敗しました
SQL-24	ユーザーの設定に失敗しました
SQL-25	パスワードの設定に失敗しました
SQL-26	URL の設定に失敗しました
SQL-27	ファクトリ・クラスはデータ ソースのインスタンスである必要があります
SQL-28	接続を作成できません。使用可能な接続がありません
SQL-29	接続の取得中に例外が発生しました
SQL-30	ユニバーサル接続プールが起動されていません
SQL-31	接続はクローズされています
SQL-32	ラベルの適用中にエラーが発生しましたラ
SQL-33	接続ラベルの削除中にエラーが発生しました
SQL-34	ラベルの取得中にエラーが発生しました
SQL-35	不一致ラベルの取得中にエラーが発生しました
SQL-36	獲得可能な接続の設定中にエラーが発生しました
SQL-37	獲得中のコールバックの登録中にエラーが発生しました
SQL-38	獲得中のコールバックの削除中にエラーが発生しました
SQL-39	中止接続コールバックの登録中にエラーが発生しました

エラー・メッセージ番号	メッセージ
SQL-40	中止接続コールバックの削除中にエラーが発生しました
SQL-41	TTL 接続コールバックの登録中にエラーが発生しました
SQL-42	TTL 接続コールバックの削除中にエラーが発生しました
SQL-43	結果セットはクローズされています
SQL-44	文はクローズされています
SQL-45	接続プール名を設定できません。重複しないように接続プール名を確認してください
SQL-46	SQL 文字列が NULL です
SQL-47	接続の無効化設定中にエラーが発生しました
SQL-48	接続プロパティを設定できません
SQL-49	データベース・サーバー名を設定できません
SQL-50	データベース・ポート番号を設定できません
SQL-51	データベース名を設定できません
SQL-52	データ・ソース名を設定できません
SQL-53	データ・ソースの説明を設定できません
SQL-54	データ・ソース・ネットワーク・プロトコルを設定できません
SQL-55	データ・ソース・ロール名を設定できません
SQL-56	最大接続再使用時間が無効です
SQL-57	最大接続再使用数が無効です
SQL-58	メソッドは無効化されています
SQL-59	接続ファクトリのプロパティを設定できません

索引

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

X