ORACLE
DATABASE

Oracle Database 12c Release 2の アプリケーション・コンティニュイティ

Oracleホワイト・ペーパー | 2017年3月

概要

アプリケーション開発者にとって、データベース・セッション（インスタンス、ノード、ストレージ、ネットワーク、またはその他の関連コンポーネント）の停止を隠すことは大変な作業です。その結果、多くの場合エラーやタイムアウトがエンドユーザーに返され、ユーザーの不満や生産性の欠如、機会の損失につながります。

アプリケーション・コンティニュイティでは、停止後に影響を受けるデータベース・セッションで実行中の作業がリカバリされるため、停止がエンドユーザーやアプリケーションからマスクされます。アプリケーション・コンティニュイティはこのリカバリをアプリケーションの後方で実行するため、アプリケーションにとってこの停止は短時間の実行遅延のように見えます。

アプリケーション・コンティニュイティは、リカバリ可能なエラー（一般的には下層のソフトウェア、フォアグラウンド、ハードウェア、通信、ネットワーク、またはストレージの各レイヤーに関連するエラー）につながる停止に対して動作します。アプリケーション・コンティニュイティを使用すると、計画外停止および計画メンテナンスを実施する際、ユーザー・エクスペリエンスが向上します。アプリケーション・コンティニュイティは、Oracleデータベースを利用するシステムおよびアプリケーションのフォルト・トレランスを強化します。

Oracle Database 12c Release 2では、Java、OCI、ODP.NET Unmanaged Providerを使った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利用することができます。

- 非XAデータソースとXAデータソース用Oracle WebLogic Server
- スタンドアロンで、またはIBM WebSphereやApache Tomcatなど、サード・パーティのアプリケーション・サーバーのデータソースとして使用されるOracle Universal Connection Pool
- JDBC `PooledConnection`インタフェースを使用したJDBCアプリケーション
- Oracle JDBC Thin再実行ドライバ
- 非XAデータソース用Oracle Tuxedo
- Oracle OCIセッション・プール
- Oracle ODP.NET Unmanaged Provider
- Oracle SQL*Plus

「アプリケーション・コンティニュイティが今、アプリケーションにもたらしているもの、それは、Oracle Database 12cのアプリケーション・コンティニュイティ機能によって多数の障害シナリオが自動的に対処されるという安心感のもとで、アプリケーションをクラスタ環境で実行できることです」

— Pythian、ATCG Principal Consultant Oracle、Marc Fielding

概要

リカバリ可能なエラーによってデータベース・セッションが使用できなくなった場合、アプリケーション・コンティニュイティによって、データベース・リクエストを中断なく迅速に再実行できます。リクエストには、データベースへのトランザクション・コールと非トランザクション・コール、およびクライアントや中間層でローカルに実行されるコールが含まれる場合があります。リクエストの再実行が成功した後は、データベース・セッションが停止した時点からアプリケーションを再開できます。ユーザーが、振替、航空券予約、購入などで何が起きているのか分からずに疑念を持ったまま取り残されることはもうありません。また、管理者が中間層のマシンを再起動し、障害の発生したセッションによって発生するログオン・ストームからリカバリする必要もありません。アプリケーション・コンティニュイティにより、多くの計画停止および計画外停止がマスクされ、アプリケーション開発者がリクエストのリカバリを試行する必要がなくなるため、エンドユーザー・エクスペリエンスが向上します。

アプリケーション・コンティニュイティがないと、次の理由により、アプリケーションが手順に沿って安全に停止をマスクすることがほぼ不可能になる場合があります。

- データベース・セッション内に反映された更新処理が消失し、クライアントの状態はその時点のまま、入力データ、返されたデータ、および変数がキャッシュされた状態で維持されます。
- コミットが発行済みの場合、コミット・メッセージは永続的ではありません。また、失われたリクエストのステータスを確認しても、それが将来的にコミットされないという保証はありません。
- アプリケーションで操作する必要のある非トランザクション・データベース・セッションの状態は消失します。
- リクエストを継続できる場合、データベースとデータベース・セッションを同期する必要があります。

単純なアプローチに振り回されないでください。再送信しようとして、単一の鍵を使ってハングし、不適切なデータベースを使って同期から外れ、インデックスを管理してダウンタイムを招き、実行時に拡張して機能不全に陥ることは避けたいものです。

アプリケーション・コンティニュイティがこの作業をアプリケーション開発者に代わって遂行します。アプリケーション・コンティニュイティは、安全にマスクできるデータベース関連の停止をマスクしようとすることで、開発者の生産性を向上させます。

以下のケースについては、アプリケーション側でエラー処理を行う必要があります。

- 無効な入力データなど、リカバリできないエラー。アプリケーション・コンティニュイティはリカバリ可能なエラーに対して適用されます。
- リカバリ可能なエラーのうち、oracle.sql具象クラスがアプリケーション内で使用されているといった制約など、記載の制約のいずれかが再実行によって検出された場合。あるいは、再実行により、ユーザーに表示される結果が、これまでにアプリケーションで設定された可能性のある元の内容にリストアできない場合など。

アプリケーション・コンティニュイティ導入以前のエクスペリエンス

アプリケーション・コンティニュイティがないと、ネットワーク停止、インスタンス障害、ハードウェア障害、ネットワーク障害、修復、構成変更、パッチなどによる停止がデータベース・リカバリによってマスクされません。

計画外停止の以前のエクスペリエンス

図1に、以前のエクスペリエンスの例を示します。この例では、リクエストが完了している場合でもエンドユーザーにエラーが返されます。

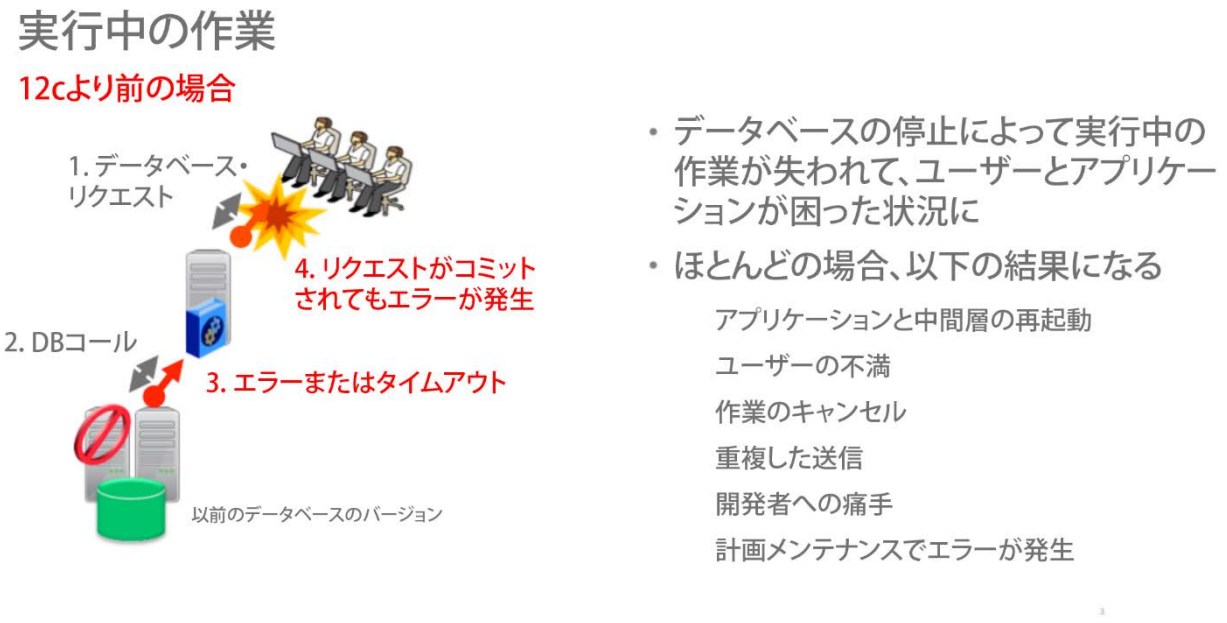

図1：以前の計画外停止のエクスペリエンス エンドユーザーは、データベース・セッションの停止によって以下のような影響を受けます。

混乱：ユーザーは、振替や注文、支払、予約、預入などのアプリケーションのリクエストに何が起きているのかが分かりません。

ユーザビリティの低下：エラーやコミットされていないデータの消失が発生して、ユーザーが再ログインしてデータの再入力や再送信を行う必要が生じ、結果的にトランザクションが重複し、不満がつのり、サポート・コストが上昇することとなります。

中断：場合によっては、DBAが中間層マシンを再起動して、ロードバランシングや受信したログオン・ストームに対処し、修復時に再びバランスを調整する必要があります。このような状況は絶対に起こってはならないことです。

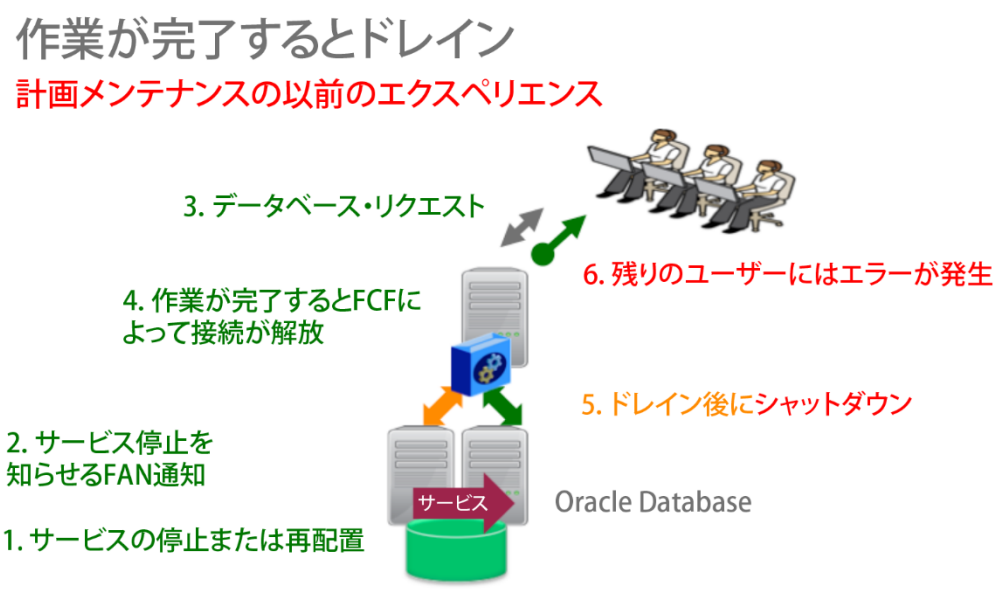

図2：計画メンテナンスの以前のエクスペリエンス

計画メンテナンスは、計画外停止より高頻度

Oracle Database 10g以降、Oracle Real Application Clusters（Oracle RAC）、Oracle RAC One、Oracle Restart、Oracle Data Guard、Oracle Active Data Guard、Oracle Guard（FANあり）をOracle接続プールとドライバ（Oracle WebLogic Server接続プール、Oracle Universal Connection Pool、ODP.NET Managed Provider/Unmanaged Provider、Oracle OCIセッション・プールとOracle Tuxedo、および12cR2の場合はJDBCドライバ）と併用しているアプリケーションの場合、計画メンテナンスは認識されません。IBM WebSphereやApache Tomcatなどのサード・パーティのアプリケーション・サーバーがある場合は、Oracle Universal Connection Poolをデータソースとして使用することを検討してください。スタンドアロンJavaアプリケーションの場合も同様です。

12cR2では、推奨されるNet接続文字列を使用する場合、高速接続フェイルオーバー（FCF）はデフォルトでオンになっています。Fast Application Notification（FAN）は、Real Application Clustersのインストールとアップグレード時に、サーバーで事前構成されます。

SRVCTLまたはGDSCTLを使用して、インスタンスのサービスを再配置します。または、UNIFORMサービス、pr all preferredを使用している場合は、インスタンス上のサービスを停止します。Oracle 12cR2以降、drain-timeoutを設定するとセッションをドレインするための時間を設けることができ、stop_optionを設定するとドレイン後に即座に強制終了させることができます。

FANは、あらかじめ計画した作業のためにサービスがダウンすることを通知します。

このFANイベントを接続プールまたはドライバが受信すると、ダウンするサービスに対するアイドル状態の接続が削除され、そのインスタンスに対する以降の接続が許可されなくなります。接続をプールに返すアプリケーション、または12cR2の場合はJava接続を確認するアプリケーションでは、高速接続フェイルオーバーによって接続が安全な場所で自動的に終了します。アプリケーションやユーザーがエラーを認識することはありません。その他のインスタンス上にある既存の接続は使用できる状態で維持され、必要に応じてそれらのインスタンスに対する新しい接続がオープンされます。12cR2でdrain_timeoutサービスを設定すると、ターゲット・インスタンスですでに実行中の作業が中断しないようにドレインが正常に行われます。

すべてのアプリケーションが接続をプールに返すか、接続を確認するか、またはその両方を実行するはずですが、現実的にはそうとは限りません。接続を返さないアプリケーションでは、インスタンスの停止時にエラーが発生して、ユーザー・エクスペリエンスが損なわれる可能性があります。

アプリケーション・コンティニュイティを使用する場合のOracle Database 12cのエクスペリエンス

図3に、アプリケーション・コンティニュイティを使用するアプリケーションにおける、Oracle Database 12cで実現されるユーザー・エクスペリエンスの向上について示します。

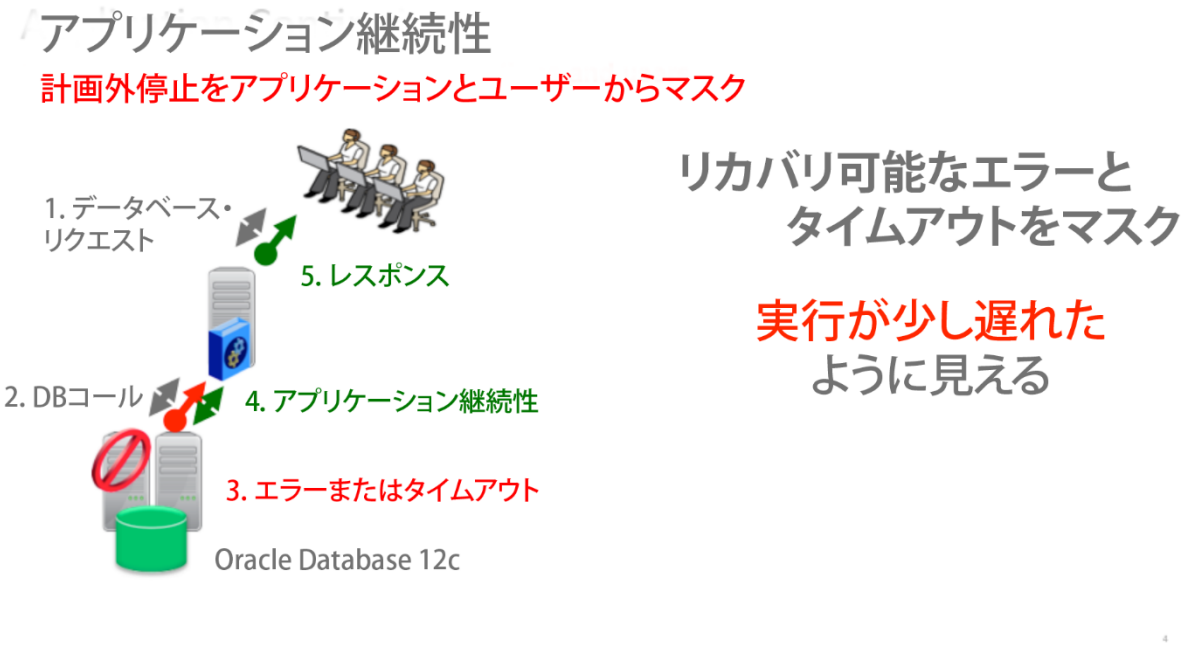

図3：アプリケーション・コンティニュイティを使用した場合のユーザー・エクスペリエンス

再実行が成功すると、アプリケーション・コンティニュイティにより、リカバリ可能なデータベース停止の多くがアプリケーションやユーザーから認識されなくなります。これにより、データベース・セッション、セッション（セッションの状態、カーソル、変数を含む）全体、実行中であった最後のトランザクション（存在する場合）をリストアすることで、マスキングが実現されます。

リカバリ可能なエラーのためにデータベース・セッションが利用できなくなった場合、アプリケーション・コンティニュイティによりセッションの再構築と、オープンなトランザクションの正常な状態へのリストアが試みられます。

トランザクションが正常にコミットされた場合、正常なステータスがアプリケーションに返されます。

再実行が成功した場合は、重複のリスクなく安全、透過的にリクエストが再開します。

再実行が失敗した場合は、データベースによりその再実行が拒否され、アプリケーションに元のエラーが返されます。再実行が成功するには、クライアントが以前にリクエストで受信した（アプリケーションによって決定された可能性がある）データと完全に同じデータが、クライアントに返される必要があります。

Oracle Database 12cによるアプリケーション・コンティニュイティ

アプリケーション・コンティニュイティのサポート対象

Oracle Database 12cのアプリケーション・コンティニュイティでは、以下のクライアント機能とサーバー機能をサポートしています。

Oracle Database 12cクライアント

» Oracle JDBC Replay Driver 12c以降。これは、Oracle Database 12cに付属する、アプリケーション・コンティニュイティのためのJDBCドライバ機能です。これ以降は“再実行ドライバ”と呼びます。

» Oracle Universal Connection Pool（UCP）

» Oracle WebLogic Server 12cおよびサード・パーティ

» Oracle JDBC - Replay Driver 12c以降を使うJava接続プールまたはスタンドアロンJavaアプリケーション

» Oracle Tuxedo（バージョン不明）

» OCI Session Pool 12c Release 2以降

» SQL*Plus 12c Release 2以降

» ODP.NET Unmanaged Provider 12c Release 2以降

Javaベースのサード・パーティ製アプリケーション・サーバーを使用している場合に、アプリケーション・コンティニュイティの機能を設定なしですぐに使用するには、次の3つの方法があります。

- もっとも効果的な方法は、データソースをUCPに置き換える方法です。このアプローチは、IBM WebSphereやApache Tomcatなど、多くのアプリケーション・サーバーでサポートされています。UCPをデータソースとして使用すると、高速接続フェイルオーバー、ランタイム・ロードバランシング、アプリケーション・コンティニュイティといった、完全に認定されたUCPの機能を使用できます。
- Oracle JDBC 12102より、リクエストの境界はJDBC `PooledConnection`インタフェースに組み込まれています。したがって、このインタフェースをネイティブ・プールで使用するサード・パーティのJava接続プールとスタンドアロンJava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使用できます。
- アプリケーションおよびサード・パーティも、リクエストの境界を追加できます。データベース・リクエストの境界が、JDBC APIの`beginRequest`と`endRequest`を使用して、データベースで特定されます。

Oracle Database 12c Server

アプリケーション・コンティニュイティはデータベースによって制御されます。データベースは、どのコールが再実行可能か、どのように再実行されるかを判断し、検証を構築し、可変値を保存します。また、キャプチャと再実行の処理方法をクライアントに指示します。

すべての一般的なデータベースのコール（SELECT、PL/SQL、ALTER SESSION、DML、DDL、COMMIT、ROLLBACK、SAVEPOINT、JDBCとOCIのRPC、ローカルのJDBCとOCIのコール）が再実行されます。

すべての一般的なトランザクション・タイプ（ローカル、パラレル、リモート、分散、PL/SQLに埋め込まれたトランザクション）が再実行されます。

シーケンス、日時、GUIDなどのオラクルのmutable関数は再実行のために維持されます（「付録 - アプリケーション・コンティニュイティの新しいデータベース概念」を参照）。

ハードウェア・アクセラレーションは、IntelチップとSPARCチップを使用したハードウェアに、Software in Siliconを使って組み込まれています。

アプリケーション・コンティニュイティの処理フェーズ

表1に示すように、アプリケーション・コンティニュイティでは3つの個別の処理フェーズが使用されます。この処理は、アプリケーション下層のドライバおよびデータベース内で、アプリケーションに対して透過的に実行されます。

表1：アプリケーション・コンティニュイティの処理フェーズ

通常実行	再接続	再実行
• データベース・リクエストにタグ付けする • サーバーとクライアントが、再実行可能なリクエストと不可能なリクエストを判別する • クライアントは指示に従って、元のコールを検証とともに保持する	• リクエストが再実行可能であることを確認する • タイムラインを確認する • 新しい接続を作成する • ターゲット・データベースを検証する • Transaction Guard を使用して、最後にコミットされた結果を適用する	• 保持していたコールを再実行する • 再実行中に、ユーザーに表示される結果が元の結果と一致していることを確認する • 再実行が成功した場合はリクエストを再開する • 再実行が失敗した場合は元の例外をスローする

通常実行 - 通常実行時には、各データベース・リクエストにリクエストの開始マーカーと終了マーカーを示すタグが追加されます。SQL*PLUSを使って、Oracle WebLogic Server接続プール、Oracle Universal Connection Pool、Oracle OCIセッション・プール、Tuxedo、またはODP.NETからのチェックアウトまたはこれらへの再チェックインを行うことでタグが追加されます。または、独自のアプリケーションか独自のJava接続プールの場合は、接続のチェックアウトとチェックイン時にリクエストの開始タグと終了タグを追加します。Universal Connection Poolはスタンドアロンとして、およびIBM WebSphereやApache Tomcatなど多くのサード・パーティ製アプリケーション・サーバーでサポートされます。

Oracle 12cクライアント・ドライバとOracle Database 12cの連携により、リクエスト内のどのコールが再実行可能であるかが判別されます。再実行可能なコールはデータベースから受け取った検証とともに、12cドライバによって長時間保持されます。ドライバは、データベース・リクエストが終了するまで、または再実行が無効化されるまで、これらのコールと検証情報を保持します。再実行は、限定されたコール、コミット（デフォルト・モードの場合（特別なモードを参照））、リクエスト終了、またはアプリケーションによる明示的な指定によって無効化されます。

再接続 - アプリケーション・コンティニュイティの再接続フェーズは、リカバリ可能なエラーが発生したときにドライバによってトリガーされます。このフェーズでは、リクエストの再実行がまだ有効であるかどうかがチェックされます。また、再実行を許可する期間を示すタイムアウト（再実行起動タイムアウト）がチェックされ、再実行の期限が切れていないことが確認されます。これらのチェックに合格した場合に、データベースへの新しい接続が確立されます。サービスの再構築が必要になる場合、データベースへの再接続には時間がかかるため、DBA は REPLAY_COUNT と REPLAY_DELAY の NET 構成値、サービスの属性である FAILOVER_DELAYとFAILOVER_TIMEOUTを確認する必要があります。NETの設定は、フェイルオーバーと新しい着信接続要求の両方に使用されるため、必要不可欠です。

ドライバは、データベースへの接続を確立した後、有効なデータベース・ターゲットに接続しているか、および最後のトランザクション（存在する場合）が正常にコミットされたかをチェックします。論理的に異なるデータベースに接続している場合や、同じデータベースに接続しているがそのデータベースのトランザクションが消失している場合は、再実行は行われません。たとえば、データベースが過去の時点にリストアされているような場合です。ドライバがコミット済みのトランザクションを再送信することはありません。Transaction Guardを使用して、送信は必ず最大1回となるようにします。

再実行 - 再実行フェーズは、データベースへの新しい接続が確立されてから開始します。データベースの命令により、ドライバが保持しているすべてのコールが再実行されます。再実行中、観測された結果の中にユーザーに表示される変更が含まれている場合は、再実行は無効化されます。これは、検証によって判定されます。再実行では、再実行フェーズ時にコミットは許可されません。許可されるのは、最後のコール（つまり、リカバリ可能なエラーが発生したコール）のコミットです。再実行が成功した後は、障害が発生した時点からリクエストが再開します。アプリケーション側から見ると、やや長い実行になります。

アプリケーション・コンティニュイティを使用する場合の制約

アプリケーション・コンティニュイティを使用する際には、グローバル、ローカル、データベースという3つのレベルで、以下の制約が適用されます。

表2：アプリケーション・コンティニュイティの制約

グローバル	リクエスト	ターゲット・データベース
サポート対象外： • デフォルトのデータベース・サービスとデフォルトの PDB サービス • 推奨されない Oracle JDBC 具象クラス • サード・パーティのステートメント・キャッシュ	• Java ストリームの場合、再実行は“ベスト・エフォート”方式 • アプリケーション・レベルのデータベース・リンクのある Oracle Active Data Guard • 2 フェーズ XA • 以下ではリクエスト・レベルは無効 — ALTER SYSTEM — ALTER DATABASE — 一部の ALTER SESSION	サポート対象外： • ロジカル・スタンバイ • Oracle GoldenGate • サード・パーティのレプリケーション

グローバル - この制約により、どのリクエストに対してもアプリケーション・コンティニュイティを有効にしたり、使用したりできなくなります。

データベース・サービス（DB_NAMEまたはDB_UNIQUE_NAMEに対応するデフォルト・サービス）を使用する接続については、再実行はサポートされません。デフォルト・データベース・サービスの有効化、無効化、またはフェイルオーバーは不可能であるため、このサービスを高可用性アプリケーションで使用することは想定されていません。

Oracle JDBCドライバを使用するアプリケーションについては、推奨されないoracle.sql具象クラス（BLOB、CLOB、BFILE、OPAQUE、ARRAY、STRUCT、またはORADATA）はサポートされません（MOS Note 1364193.1を参照）。アプリケーションに問題がないことを確認するには、ORAchk acchkの「Clean Up Concrete Classes for Application Continuity」を使用します。

Oracle Database 12c Release 2（12.2.0.1）のOCIとODP.NETの場合、OCIドライバのアプリケーション・コンティニュイティは、ADT、アドバンスト・キュー、動的バインド、一部のLOB APIを除外します。

アプリケーション・サーバー・レベルのステートメント・キャッシュ（サード・パーティ製アプリケーション・サーバーのステートメント・キャッシュなど）が有効化されている場合、再実行を使用するときは無効化する必要があります。代わりに、アプリケーション・コンティニュイティをサポートするJDBCまたはOCIステートメント・キャッシュを構成します。JDBCとオラクル向けに最適化されているため、適切に機能します。例：oracle.jdbc.implicitstatementcachesize=nnnを使用します。

リクエスト - この制約により、アプリケーション・コンティニュイティがデータベース・リクエスト部分に対して無効化されます。再実行は、次のリクエストで自動的に有効になります。

JDBCストリーム引数については、再実行は“ベスト・エフォート”方式です。たとえば、アプリケーションで物理アドレスが使用されている場合、アドレスは停止によって有効ではなくなり、再配置できません。

OCIのACについては、12.2.0.1は、AQ、ADT、および一部のLOBの操作が送られてくるとリクエストの残りを無効にします。

Oracle Database 12 Release 2（12.2）以降、再実行は、JavaおよびODP.NET Unmanaged DriverのXAデータソースでサポートされます。再実行は、ローカル・トランザクションのみサポートします。2フェーズが使用されている場合、再実行は自動で無効になり、次のリクエストで自動的に再び有効になります。これにより、アプリケーション・コンティニュイティは、昇格可能なXA、およびXAデータソースを使用して2フェーズ・コミットをほとんど使用しないアプリケーションと非常にスムーズに連携することができます。

アプリケーション・レベルで別のデータベースへのデータベース・リンクがあるOracle Active Data Guardを使用している場合、再実行はサポートされません。データベース・リンクはData Guard自体でサポートされ、アプリケーションとは別のサービスを使用するAWRでサポートされます。

ALTER SYSTEM文またはALTER DATABASE文を実行するリクエストの場合、再実行は無効化されます。たとえば、再実行では表領域の追加やデータベースの起動、シャットダウンは行われません。再実行は、コミット動作を変更する、イベントを設定するといった一部のALTER SESSION文でも無効化されます。

ターゲット・データベース - 現在アプリケーション・コンティニュイティでは、Oracleロジカル・スタンバイやOracle GoldenGate、サード・パーティ製レプリケーション・ソリューションを含む、論理的に異なるデータベースへのフェイルオーバーはサポートされません。再実行はトランザクションの消失がなく、祖先に相違がないことが確認された同じデータベースに適用されるという厳格な要件があります。各エンド・ポイントのデータベースのDBIDは異なる必要があります。各データベースのDBIDを取得するには、V$DATABASEを使用します。

リクエストが再実行の対象とならない状況

以下のイベントでは、データベース・リクエストの再実行が無効化される可能性があります。無効化された場合、次のリクエストの開始時に再実行が自動的に再び有効になります。再実行を無効化しても、通常の実行やその他のリクエストは影響を受けません。検証と記録が単に停止し、保持されたコールがドライバによって解放されます。無効化された再実行については、そのリクエストは保護されません。

表3：アプリケーションが再実行の対象とならない状況

通常実行	再接続	再実行
以下の後に発生する同じリクエスト内のすべてのコール • 動的モード（デフォルト）でのコミット成功 • 限定されたコール • 再実行無効化 API • XA への昇格	• エラーがリカバリ不可能 • タイムアウト — 再実行起動タイムアウト — 最大接続再試行数 — インシデントあたりの最大試行数 • ターゲット・データベースの再実行が無効である • 最後のコールが動的モードでコミット済み	• 検証により異なる結果が検出される

通常実行 - `begin Request`タグでコールのキャプチャが有効になります。キャプチャは、動的モードで`COMMIT`が成功するまで、元のすべてのコールに対して実行されます。ただし、限定されたコールが検出された場合、またはアプリケーションで明示的に`disableReplay`（Javaの場合）またはOCIRequestDisableReplay（OCIの場合）がコールされた場合を除きます。それ以外では、`endRequest`タグが検出されるまでキャプチャが続行します。

再接続 - エラーがリカバリ不可能な場合、タイムアウトを超過した場合（「アプリケーション・コンティニュイティ向けのサービスの構成」を参照）、ターゲット・データベースが元のデータベースと同じデータベースまたはその祖先データベースではない場合、またはリクエストがコミットされ、アプリケーション・コンティニュイティで動的モードが使用されている場合、再実行は行われません。

再実行 - 再実行されるコールの検証が合格しない場合に、再実行は中断され、元のエラーが返されます。再実行で返されてユーザーに表示される結果は、アプリケーションでそれまでに認識し、場合によっては設定してきた結果と同じものである必要があります。

可能性のある副次的作用

自律型トランザクションや外部PL/SQLおよびサーバー・サイドJavaのコールアウトでは、メインのデータベース・リクエストとは異なる副次的作用が発生します。副次的作用をもたらすデータベース・コールには、DBMS_ALERTコールを使用する電子メールと通知、DBMS_PIPEコールおよびRPCコールを使用したファイルのコピー、UTL_FILEコールを使用したテキスト・ファイルの書込み、UTL_HTTPコールを使用したHTTPコールアウト、UTL_MAILコールを使用した電子メール送信、UTL_SMTPコールを使用したSMTPメッセージ送信、UTL_TCPコールを使用したTCPメッセージ送信、UTL_URLコールを使用したURLアクセスなどがあります。

副次的作用のあるコールは、永続的な結果を残す可能性があります。たとえば、ユーザーが何らかの作業の途中でコミットせずにその場を離れ、セッションがタイムアウトするか、ユーザーが[Ctrl+C]キーを押した場合、フォアグラウンドまたはコンポーネントは失敗します。この際に、メイン・トランザクションはロールバックしますが、副次的作用がすでに適用されている可能性があります。

アプリケーションで別途指定しない限り、副次的作用は再実行されます。外部のアクションを使用するアプリケーションの場合は、副次的作用のあるリクエストを再実行すべきかどうかについて検討し、判断する必要があります。たとえば、アプリケーションが電子メールを再送信したり、再び監査したり、ファイルを再送信したりする必要があるかを確認します。ほとんどの場合、副次的作用を再実行した方が望ましいでしょう。ただし、そうしない方がよいこともあります。再実行すべきではない外部アクションがリクエストに含まれる場合は、アプリケーション・コンティニュイティが有効になっていない接続をそのリクエストで使用するか、`disableReplay API`（Javaの場合）またはOCIRequestDisableReplay（OCIの場合）を使用して、そのリクエストの再実行を無効にする必要があります。その他のすべてのリクエストは引き続き再実行されます。

アプリケーション・コンティニュイティを使用する上での評価手順

アプリケーションの評価手順

アプリケーションに対してアプリケーション・コンティニュイティを有効化する前に、表4に示す評価手順を実行します。

表4：評価手順

決定事項	タスク
リクエストの境界	リクエストの間に、接続を Oracle プールに返す。サード・パーティの Java 接続プールを使用している場合は、UCP または PooledConnection を使用するか、リクエスト境界を追加する。
JDBC 具象クラス	推奨されない JDBC 具象クラスを Java インタフェースに置き換える。アプリケーションに問題がないことを確認するには、OraChk acchk を使用し、問題がある場合はその箇所がどこかを調べる。
無効化の判断	リクエストに再実行すべきではないコールが含まれている場合は、別の接続を使用するか、Replay API を無効にする。
初期状態	アプリケーションで初期状態が設定されている場合は、FAILOVER_RESTORE=LEVEL1 を設定する。FAILOVER_RESTORE によってリストアされていない初期状態を使用する場合は、コールバックを登録する。 WebLogic Server Active GridLink または UCP のラベル付けを使用している場合は、何も行わない。
mutable 関数の許可	元の可変値を保持することを許可する。
保護レベルの測定	OraChk acchk を使用して、AC を使った保護のレベル、保護されていないリクエストとその理由を報告する。

リクエストの境界の確認

リクエストの境界は、データベース・リクエストの開始と終了をマークするタグです。Oracle 12c Release 2クライアントの時点で、境界が組み込まれた接続プールには、Oracle Universal Connection Pool、すべてのWebLogic Serverデータソース、Tuxedo、OCI、ODP.NET Unmanaged Provider、サード・パーティの標準的なアプリケーション・サーバーとスタンドアロンのJavaプール（Oracle 12 JDBCドライバの`PooledConnection`インタフェース、およびSQL*PLUSも使用）があります。

リクエストの境界が設定されているかどうかを判別するには、次の手順を実行します。

1. まず、アプリケーションが、上記のプールのいずれかから接続を取得し、返しているかを調査します。その条件を満たし、リクエストとリクエストの間で接続を返している場合、リクエストの境界が埋め込まれます。

   （SQL*Plusの場合、-acスイッチを使用）。

2. アプリケーションがこれらのプールのいずれかを使用していて、接続を解放していない場合は、多くの場合、接続の解放を設定するためのアプリケーション・プロパティが存在します。接続を解放する方が接続を解放しない場合よりもスケーラビリティが大幅に向上します。接続の解放により、その他の変更を加えなくても、リクエストの境界が自動的にマークされます。この構成を変更してください。リクエストの境界のその他の変更は不要です。

3. アプリケーションが、Oracle JDBC `PooledConnection`インタフェースを使わないサード・パーティのJava接続プールを使用している場合は、以下の3つの選択肢があります。

   - データソースをUCPに置き換えます。この簡単なデータソースの置き換えについては、Oracle JDBC下の各種サード・パーティ製アプリケーション・サーバーに関するOTCでホワイト・ペーパーを参照してください。この方法を使用することで、サード・パーティのアプリケーション・サーバーでは、FAN、ランタイム・ロードバランシング、XAアフィニティ、アプリケーション・コンティニュイティなどのUCP機能をすべて使用できるようになります。この方法は、IBM WebSphere、Apache TomCat、RedHat Spring向けに強く推奨するソリューションです。
   - 接続プールの作成と管理には、Oracle JDBC `PooledConnection`インタフェースを使用してください。
   - Javaアプリケーションの場合は、`beginRequest`および`endRequest` APIを追加して、データベース・リクエストの境界を特定します。これらのAPIは、パフォーマンスへ影響することなく、独立的にカスタムJDBCプールへのチェックアウトおよびチェックインを実行します。これらは、取得するときと返すときに追加する必要があります。APIをリクエストの開始と終了以外の場所に配置すると、リクエストが部分的になり、サポートされません。必ず他の方法のいずれかを使用することをお薦めします。

推奨されないOracle JDBC具象クラスの除外

Javaアプリケーションの場合のみ、推奨されないOracle JDBC具象クラスをアプリケーションで使用するかどうかを決定します。

Javaでアプリケーション・コンティニュイティを使用するには、推奨されないOracle JDBC具象クラスを置き換える必要があります。具象クラスの非推奨化に関する情報（アプリケーションで推奨されない具象クラスを使用している場合の対策を含む）については、My Oracle Support Note 1364193.1（https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=1364193.1）を参照してください。

アプリケーションで具象クラスが使用されているかどうかを判別するには、ACチェック（ORAchkのacchkと呼ばれる）を使用します。このチェックで具象クラスを確認する際に、12cドライバまたはデータベースを使ったり、ソース・コードを含めたりする必要はありません。アプリケーションの高可用性を計画しながら、事前にアプリケーションの検証を実行できます。

具象クラスのACチェックを制御する値は3つあります。コマンドラインまたはシェル環境変数で設定できます（または両方を使って設定することも可）。値は次のとおりです。

表5：具象クラスのACチェックの使用

コマンドライン引数	シェル環境変数	使用方法
–asmhome jarfilename	RAT_AC_ASMJAR	asm-all-5.0.3.jar 以上のバージョン（ダウンロード元：http://asm.ow2.org/）を参照する必要があります。
-javahome JDK8dirname	RAT_JAVA_HOME	JDK8 環境の場合は、JAVA_HOME ディレクトリを参照する必要があります。
-appjar dirname	RAT_AC_JARDIR	Oracle 具象クラスへの参照についてアプリケーション・コードを分析するには、コードの親ディレクトリ名を参照する必要

		があります。.class、.jar、.ear、.war、.rar の各ファイルがディレクトリ・ツリーを介して再帰的に分析されます。

使用例：

$ ./orachk -asmhome /tmp/asm-all-5.0.3.jar -javahome /tmp/jdk1.8.0_40 -appjar /tmp/appdir 出力

例：

停止のタイプ	ステータス	メッセージ
具象クラスのチェック		**Total :19845 Passed :19610 Warning :0 Failed :235 (Failed check count is one per file)**
	FAILED	[oracle/jdbc/driver/ArrayDataResultSet][[CAST] desc= oracle/sql/STRUCT method name=getObject, lineno=1477]
	FAILED	[oracle/jdbc/driver/ArrayDataResultSet][[CAST] desc= oracle/sql/NCLOB method name=getNClob, lineno=1776]
	FAILED	[oracle/jdbc/driver/BfileAccessor][[Method] name=getBFILE, desc=(I)Loracle/sql/BFILE;, lineno=99]
	FAILED	[oracle/jdbc/driver/BlobAccessor][[Method] name=getBLOB, desc=(I)Loracle/sql/BLOB;, lineno=129]
	FAILED	[oracle/jdbc/driver/ClobAccessor][[Method] name=getCLOB_, desc=(I[B)Loracle/sql/CLOB;, lineno=230]
	FAILED	[oracle/jdbc/driver/ClobAccessor][[Method] name=getCLOB, desc=(I)Loracle/sql/CLOB;, lineno=226]
	FAILED	[oracle/jdbc/driver/ClobAccessor][[Method] name=getNCLOB, desc=(I)Loracle/sql/NCLOB;, lineno=390]
	FAILED	[oracle/jdbc/driver/ClobAccessor][[CAST] desc= oracle/sql/NCLOB method name=getNCLOB, lineno=398]
	FAILED	[oracle/jdbc/driver/ClosedConnection][[Method] name=createTemporaryBlob, desc=(Ljava/sql/Connection;ZI)Loracle/sql/BLOB;, lineno=974]

具象クラスのACチェックを使用したその他の例については、以下のサイトを参照してください。
https://blogs.oracle.com/WebLogicServer/entry/using_orachk_to_clean_up

無効化すべきリクエストの有無の判断

アプリケーションに再実行すべきではないリクエストが含まれているかどうかを判別してから、デプロイしてください。リクエストで、外部のPL/SQLメッセージング・アクションのいずれか、またはサーバー側のOJVMを使用して外部へのコールを実行する場合、または自律型トランザクションを使用する場合は、そのリクエストをチェックします。再実行すべきかどうかを判断し、再実行すべきでない場合は無効にします。画面が再ロードされたか、リクエストが再送信されると、電子メールの送信や監査などのアクションが繰り返されます。その方が多くのアプリケーションにとって好都合です。しかし、リクエストの再ロードや再実行を許可できない場合があります（「12cR2における再実行による可能性のある副次的作用」のドキュメントを参照）。

アプリケーションの関数を再実行すべきではない場合、この関数に対しアプリケーション・コンティニュイティなしでサービスへの接続を使用します。または、disableReplay API（`Java`の場合は`disableReplay`、`OCI`の場合は`OCIRequestDisableReplay`）を、再実行すべきではないリクエストに埋め込みます。他のすべてのリクエストは再実行されます。

無効にすることは特に、UTL_HTTPパッケージまたは自律型プロシージャを使用するリクエスト、シリアライズを実行するリクエスト、または中間層の実際の経過時間に依存するリクエストにおいて重要になります。再実行により、セッションが同時並行的に、または個別にリカバリされます。別の方法でそれぞれのデータベース・セッションが同期されるとアプリケーション・ロジックで想定している場合は、再実行を無効化します。たとえば、アプリケーションで、コミット、ロールバック、またはセッション損失まで保持されるユーザー・ロックや外部デバイスなどの変化しやすいリソースを使用してセッションを同期する場合は、再実行を使用しないでください。

リクエストの実行ロジック内で中間層の時間に依存している場合は、再実行を無効化します。時刻T1で実行された文が時刻T2で再実行されることはないとリクエストで想定している場合は、このリクエストの再実行を無効化してください。再実行ドライバは、中間層のロジックの実行は繰り返さず、このロジックの一部として実行されるデータベース・コールを繰り返します。

フェイルオーバーの前にアプリケーションで初期状態を再確立させる必要があるかどうかを判断

多くのアプリケーションはステートレスで、リクエストが自身の状態を管理します。ステートレス・アプリケーションは、接続がプールから取得された際、接続の状態を要求しません。このようなアプリケーションは、リクエストの状態を使用でき、接続が次に取得されたときにその状態を要求しません。以下を選択します。

- FAILOVER_RESTORE=NONE（サービスで設定）（これがデフォルトです）。

一部のアプリケーションと中間層アプリケーションは接続プールに色付けしているので、すべての接続がたとえば、事前設定された言語またはタイムゾーンになっています。セッションの状態がリクエスト外部の接続に意図的に設定されている場合、リクエストからこの状態が要求されます。再実行するには、その前にこの状態を再び作り出す必要があります。以下のいずれかのオプションを選択します。

- Oracle Universal Connection PoolまたはOracle WebLogic Serverの接続ラベル付け
- FAILOVER_RESTORE=LEVEL1（サービスで設定）
- 接続初期化コールバック（Javaの場合）またはTAFコールバック（OCIの場合）

Oracle WebLogic Server接続プールまたはOracle Universal Connection Poolを使用する場合は、接続ラベル付けを使用することを推奨します。接続ラベル付けを使用すれば、ラベルがアプリケーション・コンティニュイティによって自動的に使用されます。変更作業は必要ありません。

それ以外の場合、サービスでFAILOVER_RESTOREをLEVEL1に設定することで、もっとも一般的な状態が自動的にリストアされます。表6を参照してください。

表6：FAILOVER_RESTORE=LEVEL1でリストアされた状態

NLS_CALENDAR	NLS_NUMERIC_CHARACTER	MODULE
NLS_CURRENCY	NLS_SORT	ACTION
NLS_DATE_FORMAT	NLS_TERRITORY	CLIENT_ID
NLS_DATE_LANGUAGE	NLS_TIME_FORMAT	ECONTEXT_ID
NLS_DUAL_CURRENCY	NLS_TIME_TZ_FORMAT	ECONTEXT_SEQ
NLS_ISO_CURRENCY	NLS_TIMESTAMP_FORMAT	DB_OP
NLS_LANGUAGE	NLS_TIMESTAMP_TZ_FORMAT	AUTOCOMMIT (Java and SQLPLUS)
NLS_LENGTH_SEMANTICS	TIME_ZONE (OCI, ODP.NET 12201)	CONTAINER and SERVICE (OCI, ODP.NET 12201)
NLS_NCHAR_CONV_EXCP	CURRENT_SCHEMA	

初期状態が必要で、FAILOVER_RESTOREまたはラベルが含まれていない場合は、コールバックが必要です。Oracle WebLogic管理コンソール、Oracle Universal Connection Pool、またはJDBC再実行ドライバでコールバックを登録するか、OCIとODP.NETの場合はTAFコールバックを使用します。アプリケーション・コンティニュイティにより、再実行時にコールバックが再実行されます。リクエストで確立されていない状態がアプリケーションに必要な場合、アプリケーションが接続ラベルを実装していない場合、LEVEL1に設定したFAILOVER_RESTOREを使って状態をリストアできない場合にだけ、コールバックを使用します。

可変値を保持する権限の付与

mutable関数とは、実行されるたびに新しい値を返す可能性のある関数です。mutable関数の元の結果を保持する機能は、SYSDATE、SYSTIMESTAMP、SYS_GUID、sequence.NEXTVAL向けに提供されています。元の値が保持されず、再実行時に異なる値がアプリケーションに返される場合は、再実行が拒否されます。

元の可変値の保持がアプリケーションに適合するかどうかを判断し、適合する場合は、可変値を再実行用に保持します。元の可変値を保持するかどうかの判断の際には、アプリケーション・コンティニュイティがやや遅れて実行されるものと考えます。アプリケーションが日時やシーケンスなどを受信した場合、システムは2、3秒スヌーズしてから復帰し、アプリケーションはこれらの値を受け取って処理を続行します。アプリケーション・コンティニュイティも同様です。アプリケーションにはすでに可変値があるため、サーバーは、停止が発生せずスヌーズしただけかのように、再実行時に同じ値を使って処理を続行します。

アプリケーション・ユーザーにはGRANT KEEPを使用し、シーケンス所有者にはKEEP句を使用して、可変オブジェクトを構成します。KEEP権限が付与されると、再実行時に元の関数の結果が適用されます。

例：

```
SQL> GRANT [KEEP DATE TIME | KEEP SYSGUID].. [to USER]

SQL> GRANT KEEP SEQUENCE.. [to USER] on [sequence object];

SQL> ALTER SEQUENCE.. [sequence object] [KEEP|NOKEEP];
```

アプリケーションのシーケンスはKEEP属性を使用して、シーケンス所有者のsequence.NEXTVALの元の値を維持できるため、再実行中、鍵が一致します。ほとんどのアプリケーションは、再実行時にシーケンス値を維持する必要があります。以下の例では、シーケンスのKEEP属性を設定します（この場合、文を実行するユーザーが所有するシーケンス。それ以外の場合はGRANT KEEP SEQUENCEを使用）。

```
SQL> CREATE SEQUENCE my_seq KEEP;

SQL> -- または、シーケンスが存在するが、KEEPがない場合：

SQL> ALTER SEQUENCE my_seq KEEP;
```

可変アプリケーションはローカル・データベースに適用されます。データベース・リンクを辿ることはありません。また、パラレル問合せのスレーブにプッシュ・ダウンされる場合、SYS_GUIDには適用されません。

カバレッジの測定

破壊的テストは有効なテストなので、実行することをお奨めします。ただし、障害はどの時点で発生するかはわかりません。アプリケーションがすべてのテストで適切にフェイルオーバーできても、本番環境のどこかで障害が発生し、一部のリクエストがなぜかフェイルオーバーしないことがあります。

アプリケーション・コンティニュイティのカバレッジ分析を使用すると、アプリケーション・コンティニュイティによって完全に保護されたリクエストの割合、完全に保護されていないリクエストの割合、それらのリクエストの特定とその場所が事前に報告されるので、このような事態を回避することができます。デプロイの前、アプリケーションの変更後にカバレッジ・チェックを使用してください。開発者とマネジメントは、基盤となるインフラストラクチャの障害からアプリケーション・リリースがどれほどよく保護されているか実感するでしょう。問題がある場合は、カバレッジのレベルを把握しながら、アプリケーションをリリース前に修正するか、撤回することができます。

カバレッジ・チェックの実行は、SQL_TRACEの使用に似ています。まず、サーバー側でアプリケーション・コンティニュイティ・トレースをオンにした状態で、代表的なテスト環境でアプリケーションを実行します。トレースは、標準的なRDBMSユーザー・トレース・ディレクトリ内のユーザー・トレース・ファイル内に収集されます。次に、このディレクトリを入力としてOracle ORAchkに渡して、実行されていたアプリケーション関数のカバレッジを報告します。このチェックではアプリケーション・コンティニュイティが使用されるため、データベースとクライアントでOracle 12cを使用する必要があります。アプリケーションは、アプリケーション・コンティニュイティとともにリリースする必要はありません。チェックの目的は、リリースを支援することです。

以下は、カバレッジ分析の概要です。

» アプリケーション・コンティニュイティが有効にされているときに、ラウンドトリップがデータベース・サーバーに対して行われ、キャプチャ・フェーズ中に戻ってきた場合、これは保護されたコールとしてカウントされます。

» アプリケーション・コンティニュイティのキャプチャが無効にされているときに、ラウンドトリップがデータベース・サーバーに対して行われた場合（リクエスト内ではない、または制限されたコールかdisableReply APIが呼び出されたあと）、これは保護されていないものとしてカウントされます。

» キャプチャのために無視されたラウンドトリップ、保護レベルの統計で無視された再実行。

各トレース・ファイルの処理の終了時に、データベースに送られたコールの保護レベルが計算されます。

トレースごとの判定は次のようになります：合格（>= 75）、警告（25 <= 値 <75）、不合格（< 25）。

カバレッジ・レポートの取得手順

1. セッション・レベルまたはRDBMSレベルのいずれかでアプリケーション・コンティニュイティのトレースを有効にします。

チェックする必要のある1つのアプリケーション関数のトレースを有効にします（再実行がイベントの設定によって無効にならないように、beginRequestの前またはコールバック内にトレースを設定します）。

```
alter session set events 'trace [progint_appcont_rdbms]' ;
```

テスト中に実行されているすべてのセッションのトレースを有効にします。

```
alter system set event='trace[progint_appcont_rdbms]' scope = spfile ;
```

2. アプリケーション関数を順々に実行します。アプリケーション関数についてレポートを出すには、関数を実行する必要があります。実行するアプリケーション関数の数が多いほど、カバレッジ・レポートの情報が詳細になります。
3. Oracle ORAchkを使って、収集したデータベース・トレースを分析し、保護レベルを報告します。保護されていない場合は、リクエストが保護されない理由を報告します。

ORAchkによるカバレッジのチェックを制御する値は4つあります。コマンドラインまたはシェル環境変数で設定できます（または両方を使って設定することも可）。値は次のとおりです。

表7：保護レベルに対するACチェックの使用

コマンドライン引数	シェル環境変数	使用方法
asmhome jarfilename	RAT_AC_ASMJAR	asm-all-5.0.3.jar のバージョン（ダウンロード元：fromhttp://asm.ow2.org/）を参照する必要があります。
-javahome JDK8dirname	RAT_JAVA_HOME	JDK8 環境の場合は、JAVA_HOME ディレクトリを参照する必要があります。
-apptrc dirname	RAT_AC_TRCDIR	カバレッジを分析するには、データベース・サーバー・トレース・ファイルが 1 つ以上あるディレクトリ名を指定します。トレース・ディレクトリは通常、$ORACLE_BASE/diag/rdbms/$ORACLE_SID/trace です。
None	RAT_ACTRACEFILE_WINDOW	このオプションの値はトレース・ディレクトリのスキャン時に、指定された最近の日数以内に作成されたファイルへのスキャンに対する分析を制限します。

使用例：

```
$ ./orachk -asmhome /tmp/asm-all-5.0.3.jar -javahome /tmp/jdk1.8.0_40 - apptrc
$ORACLE_BASE/diag/rdbms/$ORACLE_SID/trace 3
```

4. レポートを読む

カバレッジ・チェックにより、orachk_<uname>_<date>_<time>という名前のディレクトリが生成されます。このレポートでは、カバレッジが要約され、WARNINGSまたはFAILのステータスのあるトレース・ファイルが一覧表示されます。レポート・ディレクトリのPASSレポート（acchk_scorecard_pass.html）もチェックして、すべてのリクエストがPASS（Coverage(%) = 100）になっているか確認してください。すべての詳細を確認するには、`outfile`サブディレクトリ内でreports/acchk_scorecard_pass.htmlを見つけてください。

出力には、データベース・サービス名、モジュール名（v$session.programより。これは、Javaの接続プロパティ、たとえば、oracle.jdbc.v$session.programを使ってクライアント・サイドで設定可能）、ACTIONとCLIENT_ID（それぞれ"OCSID.ACTION"および"OCSID.CLIENTID"とともにsetClientInfoを使って設定可能）が含まれます。

出力例：格納場所は`ORAchk_.....html#acchk_scorecard`

停止のタイプ	ステータス	メッセージ
カバレッジ・チェック		**TotalRequest = 2139** **PASS = 2133** **WARNING = 0** **FAIL = 6**
	FAIL	[FAIL] Trace file name = SAMPLE_ora_1234.trc Row number = 2222 SERVICE NAME = (SAMPLE_WEB_SERVICE.OCS.QA) MODULE NAME = (DEBIT) ACTION NAME = null CLIENT ID = null Coverage(%) = 50 ProtectedCalls = 4 UnProtectedCalls = 4
	FAIL	[FAIL] Trace file name = SAMPLE_ora_5678.trc Row number = 7653 SERVICE NAME = (SAMPLE_WEB_SERVICE.OCS.QA) MODULE NAME = (PAYMENTS) ACTION NAME = null CLIENT ID = null Coverage(%) = 20 ProtectedCalls = 1 UnProtectedCalls = 4
	FAIL	[FAIL] Trace file name = SAMPLE_ora_90123.trc Row number = 15099 SERVICE NAME = (SAMPLE_WEB_SERVICE.OCS.QA) MODULE NAME = (PAYMENTS) ACTION NAME = null CLIENT ID = null Coverage(%) = 60 ProtectedCalls = 3 UnProtectedCalls = 2
	FAIL	[FAIL] Trace file name = SAMPLE_ora_4747.trc Row number = 789 SERVICE NAME = (SAMPLE_WEB_SERVICE.OCS.QA) MODULE NAME = (ACCOUNT) ACTION NAME = null CLIENT ID = null Coverage(%) = 50 ProtectedCalls = 2 UnProtectedCalls = 2

セッションの状態の整合性

セッションの状態は一貫して、非トランザクション状態がリクエストの間に変化する推移を示します。

セッションの状態の例には、NLS設定、オプティマイザのプリファレンス、イベント設定、PL/SQLグローバル変数、一時表、アドバンスト・キュー、LOB、結果キャッシュがあります。コミットされたトランザクションで非トランザクション値が変更される場合は、デフォルト値のDynamic（session_state_consistencyはサービス・レベル属性で、そのデフォルト値はDynamic）を使用します。

動的モードを使用していてCOMMITを実行したあと、状態がそのトランザクションで変更された場合、セッションが失われると、トランザクションを再実行してその状態を再確立することはできません。初期セットアップ後のセッションの状態が静的または動的か、COMMIT操作を引き続き渡すことが適切かどうかに応じてアプリケーションをカテゴリ化することができます。

動的モードはほぼすべてのアプリケーションで適切です。確信が持てない場合は、動的モードを使用します。顧客またはユーザー側でアプリケーションを変更できる場合は、動的モードを使用する必要があります。

ステートレス・アプリケーションの場合にのみ、session_state_consistencyをSTATICに設定できます。この設定により、COMMITの実行後、再実行を続行できます。アプリケーション・コンティニュイティは、コミットされたトランザクションを消去します。静的モードは、FAILOVER_RESTOREをLEVEL1に設定するか、コールバックを使用することでセッションの状態を完全にリストアできる、データベースに依存しないアプリケーションだけに使用します。このモードはTAFに似ています。接続の初期化後、状態を変更することはサポートされていません。このモードでは、リクエストが長時間実行され、プールに戻る頻度が少ない場合があります。

アプリケーションによって状態が変更されるかどうかが不明な場合、またはアプリケーションが外部コンサルタントによってカスタマイズ可能な場合は、session_state_consistencyを設定しないでください。これは、TAFのルールに非常によく似ています。

操作モード

デフォルトの動的モード時のアプリケーション・コンティニュイティは、リクエストを開始したときに再実行を有効にし、コミットが成功したときに無効にします。これはアプリケーション・コンティニュイティが、コミットされた作業を再実行できないためです。アプリケーションは、接続をプールに戻す必要があります。接続の取得と保持は拡張できません。

長時間実行されるステートレス・モードの場合、アプリケーション・コンティニュイティは、コミットされたトランザクションを消去し、処理を続行します。限定されたコールによって無効にされたあと、次のリクエストが始まるまで再び有効にされることはありません。

キャプチャは両方のモードで、明示的な無効化または限定されたコールによって無効にされます。キャプチャは、次のリクエストの開始時に再び自動的に有効になります。このすべてが自動的に実行されます。

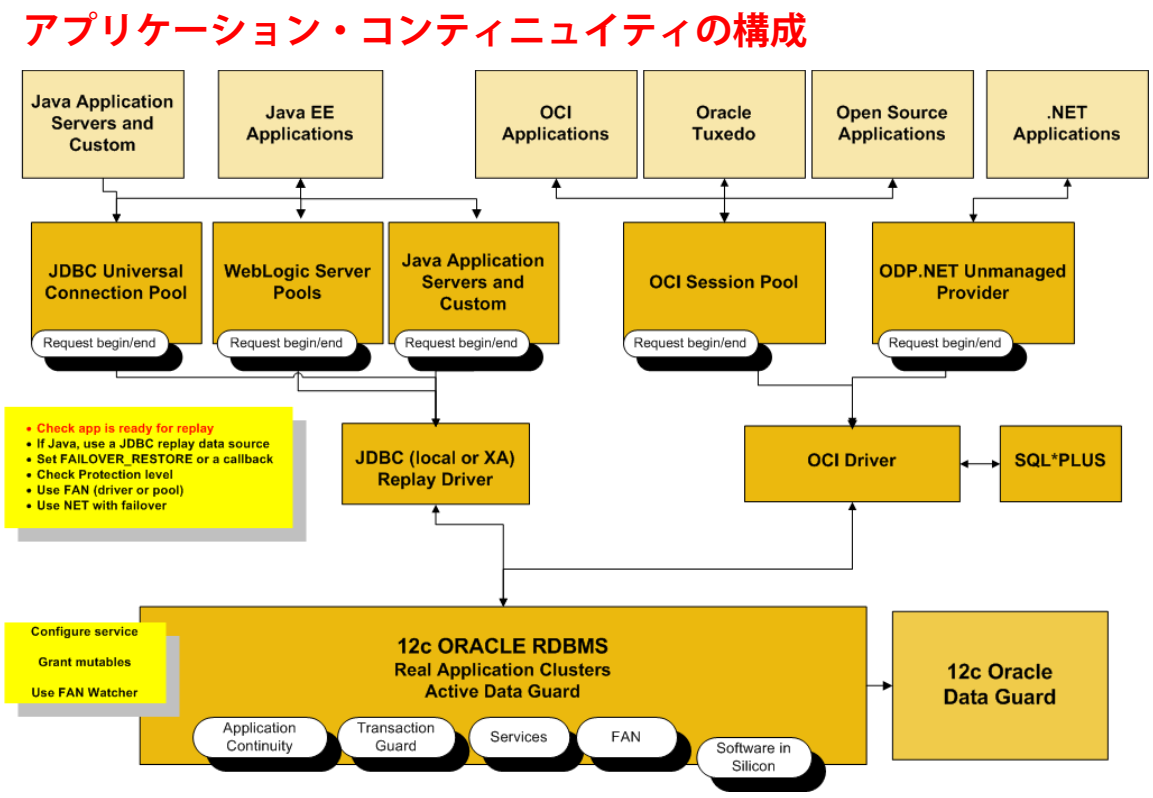

図4： アプリケーション・コンティニュイティの構成

Javaアプリケーションの場合のOracle JDBC 12c Replay Driverの構成

構成に応じて、次のいずれかのオプションを選択します。

Oracle Universal Connection Pool 12cの構成

UCP `PoolDataSource`に対して、Oracle JDBC 12c再実行データソースをコネクション・ファクトリとして構成します。

```
setConnectionFactoryClassName(“oracle.jdbc.replay.OracleDataSourceImpl”);
```

または

```
setConnectionFactoryClassName(“oracle.jdbc.replay.OracleDataSourceImpl”);
```

Oracle WebLogic Server 12cの構成

以下の図5に示すように、ローカルの再実行ドライバまたはXA再実行ドライバを使って、Oracle WebLogic Server管理コンソールでOracle 12c JDBC再実行データソースを構成します。

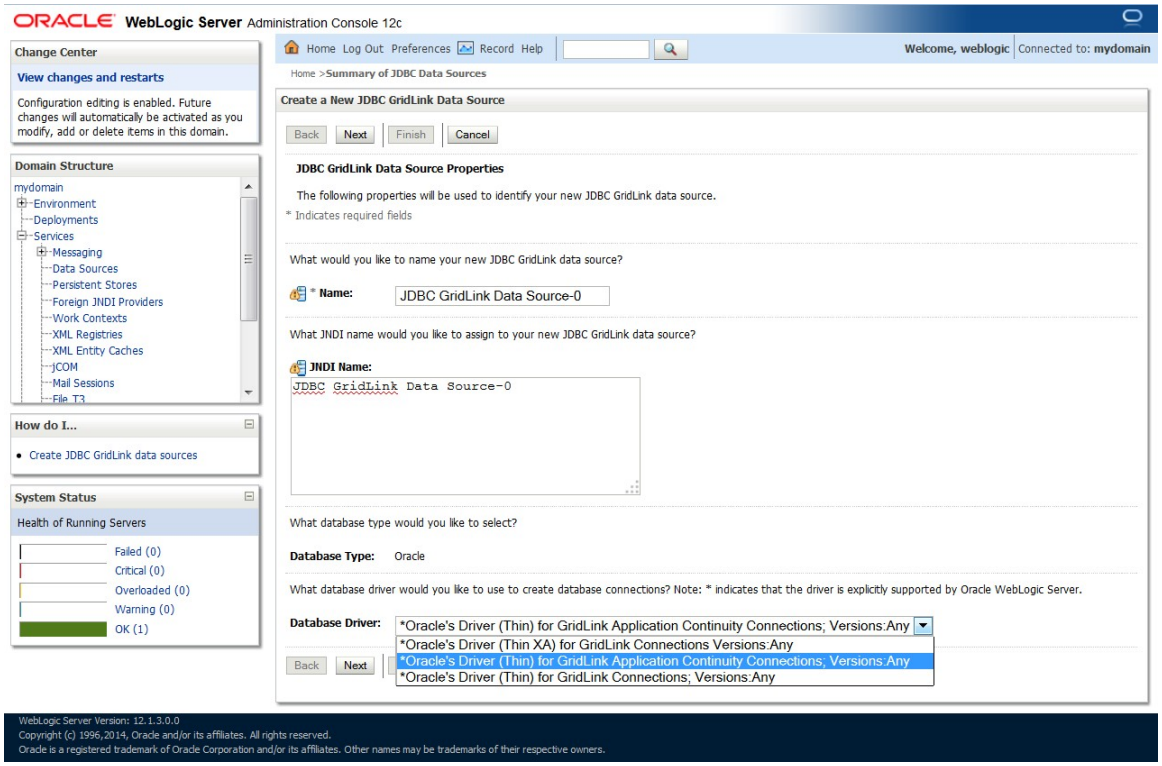

図5：JDBC Replayを有効化してWebLogic GridLinkデータソースを構成

スタンドアロンJavaアプリケーションまたはサード・パーティ接続プールの構成

プロパティ・ファイル内またはシンJDBCアプリケーション内で、Oracle JDBC 12c再実行データソースを構成します。

```
replay datasource=oracle.jdbc.replay.OracleDataSourceImpl (for non-XA)
```

または

```
replay datasource= oracle.jdbc.replay.OracleXADataSourceImpl (for XA)
```

SQL*PLUSの構成

SQL*Plusアプリケーションの場合は、-acスイッチを使用します。–acフラグはセッションの状態に対して標準的な動的モードを使用し、コミット時に無効になります。静的モードは、静的アプリケーションのみで使用できます。

```
sqlplus –ac user/password@ACservice
```

Oracle OCIセッション・プールの構成

OCIのアプリケーション・コンティニュイティには、Oracle Database 12cR2以降が必要です。アプリケーション・コンティニュイティは、サービスに対して有効になっていればOCIセッション・プールに対しても有効になります。

フェイルオーバーが実行される前に、停止を検出する必要があります。FANのONS転送が自動的に構成されるように、推奨されるTNS形式を使用します。oraccess.xmlでevents=-trueと設定します。oraccess.xmlでevents=-falseになっているかイベントが指定されていない場合、FANの使用が無効になります。oraccess.xmlの使用時にFANとSQL*Plus、PHPを維持するには、events=-trueも設定します。

```
<oraaccess> xmlns="http://xmlns.oracle.com/oci/oraaccess"
    xmlns:oci="http://xmlns.oracle.com/oci/oraaccess"
    schemaLocation="http://xmlns.oracle.com/oci/oraaccess
    http://xmlns.oracle.com/oci/oraaccess.xsd">
  <default_parameters>
    <events>true</events>
  </default_parameters>
```

ODP.NET Unmanaged Providerの構成

ODP.NET Unmanaged Providerのアプリケーション・コンティニュイティには、Oracle Database 12cR2以降が必要です。アプリケーション・コンティニュイティは、サービスに対して有効になっていれば有効になります。サービスでAQ_HA_NOTIFICATION (-notification)がTRUEに設定されている場合、FANもデフォルトで有効にされます。FANの自動構成のために、推奨されるTNS形式を使用します。推奨されるTNSを使用していない場合は、推奨されるTNS形式を使用してください。

ODP.NETには独自のCONNECT_TIMEOUTがあります。その値は（(RETRY_COUNT+1)×RETRY_DELAY）より大きい必要があります。

TNSまたはURLにおける高可用性の構成

フェイルオーバー、スイッチオーバー、フォールバック、基本的な起動時に接続を成功させるには、次のTNS/URL構成を推奨します。

TNSnamesまたはURLでRETRY_COUNT、RETRY_DELAY、CONNECT_TIMEOUT、TRANSPORT_CONNECT_TIMEOUTの各パラメータを設定して、接続リクエストがサービスを待機し、接続に成功できるようにします。

CONNECT_TIMEOUTを上限に設定して、ログイン・ストームを回避します。低い値の場合、アプリケーションまたはプールのキャンセルや再試行により、‘大量の’ログインが発生することがあります。

(RETRY_COUNT+1)×RETRY_DELAYまたはCONNECT_TIMEOUTを応答時間のSLAより大きい値に設定しないでください。アプリケーションは応答時間のSLA内で接続するか、エラーを受信することになります。

これらは、接続に高可用性を構成する場合の一般的な推奨事項です。Easy*Connectには高可用性機能はないため、使用しないでください。

これは、12.2用のすべてのOracleドライバに推奨されるTNSです。

```
Alias (or URL) = (DESCRIPTION =
    (CONNECT_TIMEOUT= 120)(RETRY_COUNT=20) RETRY_DELAY=3)
(TRANSPORT_CONNECT_TIMEOUT=3)
    (ADDRESS_LIST =
        (LOAD_BALANCE=on)
        ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
    (ADDRESS_LIST =
        (LOAD_BALANCE=on)
        ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-scan)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME = gold-cloud)))"
```

データベースのREMOTE_LISTENER設定に、クライアント接続に使用するすべてのURLに関するADDRESS_LIST内のアドレスが含まれている必要があります。

- SCAN名を使用しているURLがある場合は、**REMOTE_LISTENERS**にそのSCAN名が含まれている必要があります。
- ホストVIPの**ADDRESS_LIST**を使用しているURLがある場合は、**REMOTE_LISTENERS**に、すべてのSCAN VIPとすべてのホストVIPを含むADDRESSリストが含まれている必要があります。

アプリケーション・コンティニュイティ向けのサービスの構成

アプリケーション・コンティニュイティを使用するには、システム構成に応じて、SRVCTL、GDSCTL、DBMS_SERVICEのいずれかを使用してサービス属性を設定します。

FAILOVER_TYPE：**TRANSACTION**に設定すると、 アプリケーション・コンティニュイティが有効になります。

COMMIT_OUTCOME：**TRUE**に設定すると、Transaction Guardが有効になります。

また、以下のサービス属性を確認します。

REPLAY_INITIATION_TIMEOUT：再実行の開始を許可する期間（秒）を設定します（180、300、1800秒など。この期間が過ぎると再実行がキャンセルされます）。このタイマーはbeginRequestの時点で開始します（デフォルト：300秒）。

FAILOVER_RETRIES：再実行の各試行における接続の再試行回数を指定します（デフォルトは**18**回、再実行ドライバに適用されます）。

FAILOVER_DELAY：接続の再試行から次の再試行までの遅延（秒）を指定します（デフォルトは**10**秒。再実行ドライバに適用されます）。

FAILOVER_RESTORE ：**LEVEL1**を設定すると、再実行が始まる前に一般的に初期状態が自動でリストアされます（デフォルトは**NONE**）（表**6**、**12cR2**以降を参照）。

例

SRVCTLを使用してサービス属性を変更するには、以下のようなコマンドを使用します。ここで、EMEAはOracleデータベース名、GOLDはサービス名です。

```
srvctl modify service -db EMEA -service GOLD -failovertype TRANSACTION
-replay_init_time 300 -failoverretry 30 -failoverdelay 3 -commit_outcome TRUE –failover_restore LEVEL1 -
drain_timeout 60 –stop_option immediate
```

`Note In 12201 only, for GDSCTL set failover_restore using` DBMS_SERVICE.

DBMS_SERVICEパッケージを使用するには、以下の方法でサービス属性を変更します。

```
declare
params dbms_service.svc_parameter_array;
begin
params('FAILOVER_TYPE'):='TRANSACTION';
params('REPLAY_INITIATION_TIMEOUT'):=300;
params('FAILOVER_DELAY'):=3;
params('FAILOVER_RETRIES'):=30;
params('FAILOVER_RESTORE'):='LEVEL1';
params('commit_outcome'):='true';
params('drain_timeout'):=60;
params('stop_option'):='immediate';
dbms_service.modify_service('[your service]',params);
end;
```

Transaction Guardへのアクセス許可

アプリケーション・コンティニュイティを使用してフェイルオーバーするデータベース・ユーザーに、Transaction Guardパッケージ、DBMS_APP_CONTへの権限を付与します。

```
GRANT EXECUTE ON DBMS_APP_CONT TO <user-name>;
```

リソース割当てのチェック

必要となるメモリ・リソースとCPUリソースがシステムに搭載されていることを確認します。

メモリ：再実行では、データベース・リクエストの終了までコールが保持されるため、クライアントでより多くのメモリが消費されます。保持されるコール数が少ない場合、メモリ消費量は、無効化されたACとほぼ同じです。リクエストの終了時に、コールは解放されます。このアクションは、コールのクローズ時に解放される、無効化されたアプリケーション・コンティニュイティとは異なります。サイジングは、透過的アプリケーション・フェイルオーバー（TAF）と似ています。

Javaで良好なパフォーマンスを得る上で十分なメモリがある場合は、4～8GB（またはそれ以上）のメモリを仮想マシン（VM）に割り当ててください。たとえば、4GBの場合は、- Xms4096mと設定します。

CPU：ドライバでは、プロキシ・オブジェクトの作成（Java）、キューの管理、ガベージ・コレクションのために追加のCPUが使用されます。アプリケーション・コンティニュイティの検証では、Software in Siliconが使用されます。最新のIntelチップまたはSPARCチップを搭載したプラットフォームの場合、データベース側でのCPUオーバーヘッドは減少します。

タイマーの調整

タイマーの設定は非常に重要です。アプリケーションでリクエストのタイムアウトを基盤システムの検出時間とリカバリ時間より低い値に設定すると、基盤システムのリカバリと再実行の時間がないまま、リクエストがキャンセルされます。アプリケーションのタイムアウトは、RACおよびオブザーバの検出とクラスタ再構成、Data Guardのフェイルオーバーの時間を勘案する必要があります。時間を適切に調整しないと、タイマーが切れ、システムが回復する前に再実行が開始されます。その結果、回復する前に複数の再実行が始まることがあります。また、それより悪いことに、正常なノードのリクエストを含めたすべてのリクエストがタイムアウトし、再実行されることがあります。FANを使用する場合、検出時間コンポーネントは不要になります。

アプリケーションのタイムアウトが非常に厳しいシステムの場合は、Oracle Exadata上のFNDD（2秒）など、クラスタ再構成時間が非常に短いハードウェアを使用してノード・エビクションに備えることを検討する必要があるかもしれません。

データベースのリカバリも、FAST_START_MTTR_TARGETを設定して調整する必要があります。この値はもっとも低い1に設定できます。多くのシステムは、10～30秒を使用しています。FAST_START_MTTR_TARGETを低く設定すると、バッファ・キャッシュが維持されます。

アプリケーションでREAD_TIMEOUTまたはHTTP_REQUEST_TIMEOUTまたはカスタム・タイムアウトを設定するとします。この例では、READ_TIMEOUTを使用します。他の選択肢にも同じルールが適用されます。

タイマーのルール

» Read_timeout >> Oracle Exadataの特別なノード・エビクション（12.1.0.2で2秒）

» Read_timeout >> Misscount（デフォルトは30秒、変更可能）

» Read_timeout >> Data Guard Observer、FastStartFailoverThreshold（デフォルトは30秒、変更可能）
および FastStartFailoverThreshold >>> Misscount（少なくとも2回）

» Read_timeout >> FAST_START_MTTR_TARGET

» Read_timeout >> NETレベル（RETRY_COUNT+1）×RETRY_DELAY、

および

» Read_timeout << Replay_Initiation_Timeout（サービスで変更可能、デフォルトは300秒）

リクエストの早すぎるキャンセルを避けるため、アプリケーションのタイムアウトは、`(MISSCOUNT(or FDNN)+ FAST_START_MTTR_TARGET), (FastStartFailoverThreshold + FAST_START_MTTR_TARGET + OPEN TIME)`の最大値より大きい値にする必要があります。

トラブルシューティング

セッションがフェイルオーバーしない原因のチェックリスト

1. アプリケーション・コンティニュイティが無効になっている - orachkカバレッジ・レポートを実行してエラーの箇所と理由を確認する
2. フェイルオーバー時に環境が変更されていた
   - 可変値が維持されなかった（可変値が維持されるようにする）
   - 自動コミットが、実行時のときと同じ値に設定されていなかった
   - 他の状態がリクエスト外部で設定され、コールバックまたはラベルが使用されなかった
3. フェイルオーバー時に結果が変更されていた
   - SQLでAS OFが実行されず、異なる結果を返した
   - DMLによって何か別の処理が行われた
4. テスト・アプリケーションでV$INSTANCEまたはV$DATABASE、または同様のものが使用された
   - 実際のアプリケーションはこれらを使用しない（V$は変化するため）
5. アプリケーションが再実行時にコミットしようとした（異なるコード・パス）このような処理は許可されていない
6. 十分な時間がなくて再接続できなかった
   - 推奨されるTNSとRETRY_COUNTおよびRETRY_DELAYを使用する
7. アプリケーションのタイムアウトがリカバリ時間より短い
   - リカバリする前にアプリケーションのタイムアウトが切れた

他の手段がすべてうまくいかなかったら、1つのセッションを実行時にトレースして再実行します。サーバー側で次のイベントを使用します。

alter system set event='10602 trace name context forever, level 28:trace[progint_appcont_rdbms]:10702 trace name context forever, level 16' scope = spfile ;

管理

計画メンテナンス

計画メンテナンスの場合は、Oracle接続プールとOracleドライバのFAN、およびこれらのプールを使用するサード・パーティによって開始されたリクエストをドレインするアプローチが推奨されます。割り当てられた時間内に完了しないリクエストに対しては、ドレインとアプリケーション・コンティニュイティを併用します。

Oracle Database 12c Release 2以降、ドレインにかけることができる時間を指定するdrain_timeout、およびドレイン後（通常は）すぐに適用される停止オプションを事前に構成できます。停止、再配置、スイッチーオーバーの各操作には、必要に応じて設定値をオーバーライドするために、ドレイン・タイムアウトと停止モードが含まれています。

ドレイン・タイムアウトを指定してRELOCATEコマンドまたはSTOPコマンドを使用することで、FAN計画停止イベントによってアイドル状態のセッションが即座にクリアされ、リクエストが完了したとき、またはドライバ・レベルで次の接続チェックが発生したときに、アクティブなセッションが解放されるようにマークされます。このFANイベントにより、設定されたドレイン・タイムアウトの間に、作業を中断することなく、インスタンスからセッションをドレインすることができます。一部のセッションがチェックインしないままドレイン・タイムアウトに達した場合は、停止モードが適用されてサービスが即座に停止します。アプリケーション・コンティニュイティが構成されている場合は、これらの残りのセッションのリカバリが試行されます。

再実行なしのセッション停止または切断

アプリケーション・コンティニュイティが構成されていて、DBAがセッションを停止するか切断した場合、アプリケーション・コンティニュイティによってそのセッションのリカバリが試行されます。ただし、セッションの再実行が不要な場合は、以下のようにNOREPLAYキーワードを使用してください。

個々のセッションを再実行なしで停止

```
alter system kill session 'sid, serial#, @inst' noreplay;

alter system disconnect session 'sid, serial#, @inst' noreplay;

execute DBMS_SERVICE.DISCONNECT_SESSION('[service name]', DBMS_SERVICE.NOREPLAY) ;
```

個々のサービスを再実行なしで停止

`srvctl stop service`を使用して、停止するインスタンスまたはノードを指定できます。再実行が不要な場合は、`–force`および`–noreplay`オプションを使用すると再実行されません。次に例を示します。

```
srvctl stop service -db orcl -instance orcl2 -service orcl_pdb38 –force –stop_option immediate –noreplay

srvctl stop service -db orcl -node rws3 -service orcl_pdb38 –force –stop_option immediate –noreplay
```

複数のサービスをグループとして同時に停止

データベース・レベル、インスタンス・レベル、またはノード・レベルでグループとして実行できるサービスをすべて停止します。noreplayが必要な場合は、`noreplay`オプションを使用します。セッションが強制終了する前に、drain_timeoutを使ってドレインします。次に例を示します。

```
srvctl stop service -db orcl –drain_timeout 60 –force –stop_option immediate

srvctl stop service -db orcl –pdb orcl30 –drain_timeout 60 –force –stop_option immediate

srvctl stop service -node rws2 –drain_timeout 60 –force –stop_option immediate
```

結論

アプリケーション・コンティニュイティは、リカバリ可能な停止の後に不完全なデータベース・リクエストを再実行することで、停止をアプリケーションやエンドユーザーに認識させません。多くの停止と計画メンテナンスの操作が認識されません。そのため、アプリケーションがユーザーにエラーを送信し、ユーザーは何が起きているか分からないまま置き去りにされたり、データの再入力を求められたり、さらに悪いことに管理者が中間層サーバーを再起動して障害に対応するといった状況が回避されます。

アプリケーション・コンティニュイティは、Oracleデータベースを利用するシステムおよびアプリケーションのフォルト・トレランスを強化します。

付録 - アプリケーション・コンティニュイティの新しいデータベース概念

アプリケーション・コンティニュイティでは以下の用語および概念が使用されます。

リカバリ可能なエラー

リカバリ可能なエラーとは、実行中のアプリケーション・セッションのロジックとは関係なく、外部のシステム障害によって発生するエラーです。リカバリ可能なエラーは、フォアグラウンド、ネットワーク、ノード、ストレージ、データベースの計画停止および計画外停止の後に発生します。アプリケーションはエラー・コードを受け取りますが、この方法では最後に送信した操作のステータスをアプリケーションで把握できない場合もあります。リカバリ可能なエラーの機能はOracle Database 12cで強化されており、より多くのエラーに対応し、OCI用のパブリックAPIが追加されています。アプリケーションのコード内でエラー番号の一覧を記述する必要はもうありません。アプリケーション・コンティニュイティは、リカバリ可能なエラーのエラー・コードに従って起動されます。

信頼できるコミット結果

クライアントの視点では、トランザクションは、トランザクションREDOの書込み後に生成されたOracleメッセージ（コミット結果と呼びます）をクライアントが受け取ったときに、トランザクションがコミットされたと認識されます。しかし、COMMITメッセージは永続的ではありません。アプリケーション・コンティニュイティではOracle Database 12c Transaction Guardを使用して、リカバリ可能なエラーの後にコミット結果が消失した可能性がある場合に、信頼できる方法でコミット結果を取得します。

データベース・リクエスト

データベース・リクエストとは、アプリケーションから送信された作業単位のことです。通常、データベース・リクエストはSQLおよびPL/SQL、データベースRPCコール、および1つのデータベース接続上で発生する1つのWebリクエストへのローカル・クライアント側のコールに対応します。一般的には、データベース接続の接続プールからのチェックアウトと接続プールへのチェックインのために実行されるコールによって、データベース・リクエストの境界が定められます。リカバリ可能なエラーについては、アプリケーション・コンティニュイティはデータベース・セッションを再確立して、コミットされていないデータベース・リクエストを安全に繰り返します。

通常、JDBCを使用するデータベース・リクエストはある標準的なパターンに従います。設計されているデータベース・リクエストの数を表示するコード・スニペットは以下のようになります。

1. データベース・リクエストは、PoolDatasourceに対するgetConnectionコールから始まります。
2. アプリケーションのロジックが実行されます。SQL、PL/SQL、RPC、またはローカル・コールが実行される可能性があります。
3. トランザクションがコミットされます。
4. 接続が接続プールに返されたときに、データベース・リクエストが終了します。

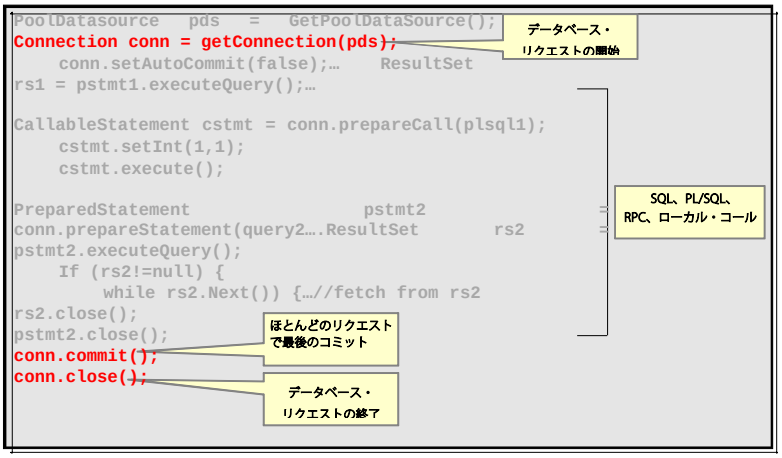

図6：データベース・リクエストの例

mutable関数

mutable関数とは、コールされるたびに結果が変わる可能性のある関数です。mutable関数によって、クライアントに表示される結果が再実行時に変わるために再実行が拒否される可能性があります。

キー値によく使用されるsequence.NEXTVALについて考えてみましょう。主キーがシーケンス値を使用して組み立てられており、この主キーが後で外部キーやその他のバインド内で使用される場合、同じ関数の結果を再実行時にも返す必要があります。

アプリケーション・コンティニュイティでは、GRANT KEEPまたはALTER..KEEPが構成されている場合に、再実行時にOracle関数コールの可変値を置き換えます。元の可変値を保持できるデータベース関数（equence.NEXTVAL、SYSDATE、SYSTIMESTAMP、SYS_GUIDなど）がコール内で使用されている場合、関数の実行により返された元の値を保存して、再実行時に再適用できます。アプリケーションで可変値機能の権限を付与していない状態で、再実行時に異なる結果がクライアントに返される場合は、これらのリクエストの再実行が拒否されます。

Oracle Corporation, World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065, USA

著者：Michael Ramchand、Peter Wilson、Martien Ouwens

海外からのお問い合わせ窓口

電話：+1.650.506.7000

ファクシミリ：+1.650.506.7200

Hardware and Software, Engineered to Work Together

アプリケーション・コンティニュイティ
2014年7月
著者：Carol Colrain
共著者：Tong Zhou、Nancy Ikeda、Stefan Roesch、Jean de Lavarene、Kevin Neel
オラクルの情報を発信しています。

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com