ORACLE

Oracle Databaseの アプリケーション・コンティニュイティ

Oracle Real Application ClustersとOracle Active Data Guardを使用した計画メンテナンス時と計画外停止時のアプリケーションの継続的な可用性の利点

バージョン [1.0]

公開

本書の目的

本書では、Oracle Database 19cリリースの機能の概要と強化された点が説明されています。本書は、19cへのアップグレードに関するビジネス上の利点の評価と、説明した製品機能の実装およびアップグレードの計画を支援することのみを目的としています。

免責事項

本文書には、ソフトウェアや印刷物など、いかなる形式のものも含め、オラクルの独占的な所有物である占有情報が含まれます。この機密文書へのアクセスと使用は、締結および遵守に同意したOracle Software License and Service Agreementの諸条件に従うものとします。本文書と本文書に含まれる情報は、オラクルの事前の書面による同意なしに、公開、複製、再作成、またはオラクルの外部に配布することはできません。本文書は、ライセンス契約の一部ではありません。また、オラクル、オラクルの子会社または関連会社との契約に組み込むことはできません。

本書は情報提供のみを目的としており、記載した製品機能の実装およびアップグレードの計画を支援することのみを意図しています。マテリアルやコード、機能の提供をコミットメント（確約）するものではなく、購買を決定する際の判断材料になさらないでください。本文書に記載されている機能の開発、リリース、時期および価格については、弊社の裁量により決定されます。製品アーキテクチャの性質上、本書に記述されているすべての機能を安全に組み込むことができず、コードの不安定化という深刻なリスクを伴う場合があります。

ORACLE

目次

はじめに

アプリケーションは、データベース層の計画メンテナンス、計画外停止、ロードの不均衡が不可視化されたときに継続的サービスを達成します。アプリケーションのベスト・プラクティス、シンプルな構成の変更、MAAのベスト・プラクティスを使用してデプロイされたOracle Databaseの組合せにより、アプリケーションの継続的な可用性が確保されます。

次のチェックリストは、アプリケーション・コンティニュイティをまだ使用していない場合でも、アプリケーションとデータベースを準備する際に役立ちます。ここに記載されている内容は、継続的な可用性をサポートするためのシステムを準備し、計画メンテナンス作業と計画外停止が発生した場合に起こる可能性がある停止時間を短縮する上で重要な価値をもたらします。

アプリケーションの継続的な可用性を維持するための機能セット

アプリケーションは、データベース層の計画メンテナンス、計画外停止、ロードの不均衡が不可視化されたときに継続的な可用性を達成します。オラクルでは、計画イベント時、計画外停止時、ロードの不均衡時にアプリケーションの可用性を維持するために選択できる機能セットを提供しています。これらの機能は、サービスの中断からアプリケーションを保護する保険証券のようなものであると考えることができます。最善の機能とは、アプリケーション開発者がインフラストラクチャに対してではなく、機能の構築に集中できるようにアプリケーションに対して完全に透過的であり、アプリケーションが将来的に変化したときにアプリケーションを継続的に保護できる機能です。オラクルでは、これを未来の保証と呼んでいます。

この機能セットから開始してください。

計画メンテナンスに向けたセッションのドレイニングとリバランシング

計画メンテナンスが開始されると、インスタンス、PDB、またはデータベースからドレインする必要があるセッションはドレイニング対象としてマークされ、アイドル・セッションが段階的にリリースされます。アクティブ・セッションは、このセッション内で実行される作業が完了したときにドレインされます。セッションのドレイニングは、高速アプリケーション通知（FAN）用に構成されたOracle接続プールおよび中間層で広範に使用されています。Oracle Database 18c以降、PDBとインスタンスが停止または再配置されたときにセッションがドレインされます。ドレイニングは常に、計画メンテナンスを不可視化するための最善のソリューションです。アプリケーション・コンティニュイティなどのフェイルオーバー・ソリューションは、割り当てられた時間内に作業がドレインされないときのフォールバック機能です。

透過的アプリケーション・フェイルオーバー（TAF）

TAFは、Oracle8iまで遡る機能です。インスタンスに障害が発生した後、TAFにより、新しいセッションが作成され、オンデマンドでSELECTモードを使用する場合、障害が発生する前の時点まで戻って問合せが再実行されます。Oracle Database 12.2以降、TAFには、問合せが再実行される前のセッションの初期状態をリストアするために、アプリケーション・コンティニュイティと一致するFAILOVER_RESTOREが用意されています。カーソルは、最初に再確立された状態を使用して再実行されます。
TAFを使用したアプリケーションでは、このセッション内で、後でセッションの状態（PLSQL、一時表、一時LOB、sysコンテキストなど）を変更してはなりません。なぜなら、このセッションの状態はリストアされないからです。

アプリケーション・コンティニュイティ（AC）

アプリケーション・コンティニュイティは、停止を不可視化します。Oracle Database 12.1以降は、Javaベースのシン・アプリケーションで使用でき、Oracle Database 12.2.0.1以降は、OCIおよびODP.NETベースのアプリケーションで使用できます。アプリケーション・コンティニュイティを有効にすると、セッションの状態やトランザクションの状態などが分かっている時点からセッションがリカバリされて再構築されます。処理中のすべての作業がアプリケーション・コンティニュイティによって再構築されます。フェイルオーバーが行われると実行時間がわずかに長くなりますが、アプリケーションはそれまでと同様に動作し続けます。アプリケーション・コンティニュイティの標準モードは、OLTPスタイルのプール済みアプリケーションに適しています。

透過的アプリケーション・コンティニュイティ（TAC）

Oracle Database 18c以降、セッションとトランザクションの状態を透過的に追跡および記録する透過的アプリケーション・コンティニュイティ（TAC）機能が導入され、リカバリ可能な停止の後にデータベース・セッションをリカバリできるようになりました。このリカバリには、アプリケーションの知識もコード変更も必要ありません。そのため、現行のアプリケーションの透過的アプリケーション・コンティニュイティを有効にできます。アプリケーションを透過的にフェイルオーバーするため、アプリケーションからユーザー・コールが発行されたときのセッション状態の使用状況を取得して分類した状態追跡情報が使用されます。

どのソリューションを使用するべきか

	TAC	AC	TAF	ドレイニング
アプリケーションの実装方法がわからない	有	無	無	有
アプリケーションがトランザクションを実行する	有	有	無 （計画外トランザクション切断の場合のみ）	有
アプリケーションがOracleの状態（一時LOB、PL/SQL、一時表）を使用する	有	有 無（静的使用の場合）	無	有
アプリケーションが接続プールを使用しない	有	無	有	有
アプリケーションに副次的作用（ファイル転送）がある	有 副次的作用は再実行されない	カスタマイズ可能	無	有
アプリケーションで初期状態をリストアする必要がある	有、および カスタマイズ可能	有、および カスタマイズ可能	有、および カスタマイズ可能	有
アプリケーションが変更されても未来が保証される	有	無	無	有

アプリケーション構成の手順

ドレイニングを使用した透過的アプリケーション・コンティニュイティは、継続的な可用性を実現するために推奨されるソリューションです。12cドライバを使用している場合、または初期状態または副次的作用のためにカスタマイズが必要である場合、アプリケーション・コンティニュイティを使用する必要があります。TAFは、引き続き使用可能であり、完全にサポートされます。

ソリューションを実装する際は、次の指示に従ってください。

1. デフォルトのデータベース・サービス（デフォルトのサービスの名前はデータベースやPDBの名前と同じです）ではない、Oracle Clusterwareによって管理されるサービスを使用してください。作成したサービスにより、位置の透過性とHA機能が提供されます。
2. タイムアウト、再試行、遅延が組み込まれた推奨される接続文字列（本書でこの後に説明します）を使用することで、停止時に着信接続要求でエラーが発生しないようにします。
3. 高速アプリケーション通知（FAN）は、ドレイニングを開始し、障害から抜け出し、サービスの再開時とロードの不均衡の発生時にセッションをリバランスするために不可欠なコンポーネントです。ノード障害やネットワーク障害などの停止については、クライアントがFANによって中断されていない場合、アプリケーションのフェイルオーバーは行われません。FANは、すべてのフェイルオーバー・ソリューションに適用されます。FANを構成する場合は、ONSの自動構成を使用してください。推奨されるTNSを厳密に使用してください。この形式は変更しないでください。（例外：クライアントが12c以前である場合、FANを手動で構成してください。）
4. メンテナンスが開始される前に、メンテナンスの対象であるインスタンスまたはノードから作業をドレインしてください。Oracle接続プールまたは接続テスト（あるいはこの両方）を使用してFANを有効化してください。FANを使用したOracle接続プールが最善のソリューションです。なぜなら、プールによってセッション移動の完全なライフサイクルが実現されるからです。つまり、作業のドレイニングとリバランスがメンテナンスとして進行します。FANを使用する場合、接続をプールに返してください。サーバーのドレイニング（代替計画については本書でこの後に説明します）を使用しており、テストが標準テストではない場合、DBMS_APP_CONT_ADMINを使用してテストをサーバーに追加してください。DRAIN_TIMEOUT内にドレインされないセッションは、フェイルオーバーされます。
5. セッションのフェイルオーバー用の標準ソリューションは、透過的アプリケーション・コンティニュイティ（TAC）です。Oracle Database 12c Release 2を使用している場合、または副次的作用またはコールバックありでカスタマイズしたい場合、または一時表などの状態を使用し、クリーンアップしないアプリケーションがある場合は、アプリケーション・コンティニュイティ（AC）を使用してください。アプリケーションが読取り専用であり、初期設定後にセッション内のOracleセッションの状態が変更されない場合は、透過的アプリケーション・フェイルオーバー（TAF）を使用してください。

アプリケーションの継続的な可用性のための構成要素

サービスの使用

サービスとは、作業を管理するためのロジックを抽象化したものです。サービスを使用すると、アプリケーションはMAAシステムの冗長要素の信頼性を享受できます。サービスは、単一のシステム・イメージによって作業を管理するため、基盤システムの複雑さはクライアントからは分かりません。

サービスには次のような特徴があります。

- 管理の単位：少数のサービスは管理できますが、ノード、インスタンス、リスナー、ネットワーク・インタフェースは管理できません。サービスは、サイトとデータベースの位置の透過性を実現します。
- 可用性の単位：リソースは、ソフトウェア・スタック全体を起動する必要はなく、サービスごとに迅速に、個別に、並行してリカバリされます。
- パフォーマンスの単位：作業は、サービスの品質と優先順位に応じてMAAシステム全体で透過的にルーティングされます。サービスは、サービス・レベルのしきい値を基準にして測定され、違反は、AWR内のアドバイス付きのソリューションと一緒にマネジメントに報告されます。

FAN、接続識別子、TAC、AC、スイッチオーバー、コンシューマ・グループ、その他多くの機能と操作が、サービスの使用を前提としています。デフォルトのデータベース・サービスは使用しないでください。なぜなら、これは無効化、再配置、または制限できないため、高可用性がサポートされないからです。使用するサービスは、Data Guard環境内の特定のプライマリ・ロールまたはスタンバイ・ロールに関連付けられています。アプリケーションの使用には、初期化パラメータservice_namesを使用しないでください。

サービスの例：

ATP-Dから、PDBデータベースごとに、事前構成された5つのサービスが選択対象として提供されます。すべてのサービスにFANとドレイニングが用意されています。

サービス名	説明	ドレイニング	FAN	TAC
TPURGENT	OLTPの最高の優先順位	有	有	有
TP	OLTPの一般的な優先順位 メイン・サービスとしての使用を推奨	有	有	有
HIGH	報告またはバッチ 最高の優先順位	有	有	
MEDIUM	報告またはバッチ 中程度の優先順位	有	有	
LOW	報告またはバッチ 最低の優先順位	有	有	

接続文字列またはURLにおける高可用性の構成

オラクルが提供する接続文字列はすべて、次の推奨事項に準拠しています。オラクルが提供するウォレットを使用する場合、何も行う必要がありません。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する場合、次のTNS/URL構成を推奨します。

tnsnames.oraファイルまたはURLで、RETRY_COUNT、RETRY_DELAY、CONNECT_TIMEOUT、TRANSPORT_CONNECT_TIMEOUTパラメータを設定し、サービスと接続が正常に行われるまで接続リクエストが待機できるようにします。接続試行および接続再試行は、Oracle Database Net Servicesによって管理されます。

CONNECT_TIMEOUTを高い値に設定して、90秒や120秒などのログイン・ストームを回避します。値を低くすると、アプリケーションまたはプールのキャンセルや接続の再試行のために、ログイン合戦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。EZCONNECTには高可用性の能力がないため、クライアントで簡易接続ネーミングを使用しないでください。

次で指定するstandby-scanは、（アクティブな）Data Guard構成に指定されているSTANDBYサイトで使用可能なSCANアドレスを参照することに注意してください。最初にPRIMARYサイトへの接続が試みられ、サービスを使用できない場合、STANDBYサイトでサービスへの接続が試みられます。サービスの位置が判明したら、Oracleドライバ12.2以降は、提供されたサービスと一緒にaddress_listを記憶し、このサービスが次回移動するまでは、これを最初に選択します。

standby-scanをTNS接続記述子に追加し、standby-scanに透過的にフェイルオーバーする処理はオプションです。同じリージョン内のスタンバイ・データベースにフェイルオーバーする場合はたいてい、許容できるパフォーマンスを得られます。一方、別のリージョン内のスタンバイ・データベースにフェイルオーバーする場合は、追加のネットワーク待機時間が生じる結果、許容できない応答時間のパフォーマンスになる可能性があります。後者の場合、サイトのフェイルオーバー操作が必要になり、この際、中間層リソースとスタンバイ・データベースが含まれる別のリージョンへのDNSのフェイルオーバーも必要になります。

Oracleドライバのバージョン12.2以降すべてに対してこの接続文字列を使用してください。

```
Alias (or URL) =
  (DESCRIPTION =
  (CONNECT_TIMEOUT= 90)(RETRY_COUNT=50)(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=standby-scan)(PORT=1521)))
    (CONNECT_DATA=(SERVICE_NAME = YOUR SERVICE)))
```

12.1以前のJDBC接続の場合、次を使用する必要があります。この接続文字列は、より低いCONNECT_TIMEOUTを使用します。なぜなら、Oracle Database 12.2までは、Javaシン・ドライバに対してTRANSPORT_CONNECT_TIMEOUTを使用できないからです。RETRY_DELAYには、11.2.0.4クライアント用のパッチが必要です。

```
Alias (or URL) =
  (DESCRIPTION =
  (CONNECT_TIMEOUT= 15)(RETRY_COUNT=50)(RETRY_DELAY=3)
    (ADDRESS_LIST =
      (LOAD_BALANCE=on)
      (ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
    (ADDRESS_LIST =
      (LOAD_BALANCE=on)
      (ADDRESS = (PROTOCOL = TCP)(HOST=standby-scan)(PORT=1521)))
    (CONNECT_DATA=(SERVICE_NAME = YOUR SERVICE)))
```

高速アプリケーション通知（FAN）

FANを使用する必要があります。FANは、アプリケーションのフェイルオーバーを中断する場合の必須コンポーネントです。FANがない場合、ハードウェアやネットワークに障害が発生した後にアプリケーションがTCP/IPタイムアウト時に停止し、リソースが再開されたときにリバランスを怠る可能性があります。すべてのOracleプールとすべてのOracleアプリケーション・サーバーがFANを使用します。サード・パーティ製JAVAアプリケーション・サーバーは、Oracle UCPを使用してFANを有効化できます。FANを使用するためにアプリケーションを変更する必要はありません。変更する必要があるのは構成のみです。

計画メンテナンス時の継続的なサービスの場合、FANと一緒に次を使用します。

- Oracleプールまたは
- サード・パーティ製JDBCアプリケーション・サーバーを使用したOracle UCPまたは
- 最新のOracleクライアント・ドライバ

計画外停止時の継続的なサービスの場合、FANと一緒に次を使用します。

- アプリケーション・コンティニュイティまたは
- 透過的アプリケーション・コンティニュイティ

上記のADDRESS_LISTの形式は、複数の理由により重要です。その1つは、この形式を使用すると、ONSの自動構成が可能になることです。ONSは、FANイベントを中間層プールおよびクライアントに伝播するために使用されます。データベース接続が確立されると、データベース層のONS情報が中間層に送信され、中間層がONS通信経路を自動的に確立できるようになります。つまり、構成を動的に行うことが可能になり、中間層で保持する必要がなくなります。ONS接続は、プライマリ・サイトとスタンバイ・サイトの両方から行われます。

FANのカバレッジ

FANイベントは、次と統合されます。

- Oracle Fusion MiddlewareとOracle WebLogic Server
- Oracle Data Guard Broker
- JDBCのシン・インタフェースとOCIインタフェースの両方用のOracle JDBC Universal Connection PoolまたはOracle JDBC Driver
- 管理対象外プロバイダと管理対象プロバイダ用のODP.NET接続プール
- Oracle Tuxedo
- SQL*Plus
- PHP
- Oracle Global Data Services
- Oracle JDBC Universal Connection Poolを使用したサード・パーティ製JDBCアプリケーション・サーバー
- リスナー

クライアントでFANを有効にするには：

前に説明したTNSエイリアスまたはURLを使用します。Oracle Database 12c以降のクライアント・ドライバを使用する場合、この接続文字列を使用して、FANイベントを受信するためにクライアント側でOracle Notification Service（ONS）のサブスクリプションを自動構成します。これより古いドライバの場合、構成の詳細については、付録のFANに関するホワイトペーパーを参照してください。ONSは、データベース層とクライアント層の間のセキュアな通信経路を提供します。これにより、クライアントは、サービスの可用性（コンポーネントの停止または起動）やランタイム・ロードバランシングに関するアドバイスの通知を受けて、通常の操作時の作業配置の効率を高めることができます。

1. 提供されたTNSエイリアスまたはURLを使用してください。12c以降のドライバを使用する場合、この接続文字列により、FANイベントを受信するためにクライアント側でONS（自動ONS）のサブスクリプションが自動構成されます。これより古いドライバの場合、FANに関するホワイトペーパーを参照してください。
2. クライアントによっては、次のようにアプリケーション構成プロパティでFANを有効化してください。
    - Universal Connection PoolまたはJDBCシン・ドライバ（12.2以降） プロパティFastConnectionFailoverEnabledを設定します。

- Oracle RAC向けのWebLogic Active GridLink
  FANと高速接続フェイルオーバーはデフォルトで有効になります。
- Oracle WebLogic Server、IBM WebSphere、IBM Liberty、Apache Tomcat、Red Hat WildFly（JBoss）、JDBCアプリケーション
  接続プールの交換用としてUniversal Connection Poolを使用します。
- ODP.Netクライアント（管理対象プロバイダと管理対象外プロバイダ）
  ODP.Net 12.1以前を使用する場合、接続文字列内の"HA events = true;pooling=true"を設定します。
- Oracle Call Interface（OCI）クライアントとOCIベースのドライバ
  ネイティブ設定のないOracle Call Interface（OCI）クライアントは、oraacces.xmlを使用し、イベントをtrueに設定できます。

  Python、Node.js、およびPHPではネイティブ・オプションを使用できます。PythonとNode.jsでは、接続プールの作成時にイベント・モードを設定できます。PHPでは、php.iniを編集し、エントリoci8.events=onを追加します。

  SQL*Plusでは、FANはデフォルトで有効になります。

3. TLS（ウォレットベース）認証を使用するようにONSを構成できます。JDBCアプリケーションの場合：
   a) アプリケーションのCLASSPATH内に次のJARファイルが存在することを確認します：ons.jar、osdt_cert.jar、osdt_core.jar、oraclepki.jar。Oracle ATP-Dサービスは常にTLSを使用します。Oracle UCP XML構成ファイルを使用して宣言的に設定します。

```
<?xml version="1.0"?>
<ucp-properties>
<connection-pool
connection-pool-name="UCP_pool1"
user="dbuser"
password="dbuserpasswd"
connection-factory-class-name="oracle.jdbc.pool.OracleDataSource"
initial-pool-size="10"
min-pool-size="5"
max-pool-size="15"
url="jdbc:oracle:thin:@(RECOMMENDED TNS)
fastConnectionFailoverEnabled="true"
onsConfiguration="nodes=primary-scan:6200,secondary-
scan:6200\nwalletfile=/net/host/path/onswallet\nwalletpassword=myWalletPasswd">
</connection-pool>
</ucp-properties>
```

   b.) FAN用のウォレットを指定します。次のいずれかを実行します。
   AUTO-ONSをウォレットと一緒に使用するには、アプリケーションで次のJavaシステム・プロパティを設定する必要があります。

“-Doracle.ons.walletfile=/replace_this_with_host_path/onswallet”および
“-Doracle.ons.walletpassword=myONSwalletPassword”
これは、プール単位または接続単位基準で設定することはできません。

明示的なONS構成（AUTO-ONSではない）を使用するには、次のいずれかを実行します。
i) プログラムによってOracle UCP内で、次のように、setONSConfiguration()を呼び出します。

```
pds.setONSConfiguration(“nodes=primary-scanhost:6200,secondary-
scanhost:6200\nwalletfile=/replace_this_with_host_path/onswallet\nwalletpass
word=myWalletPassword”);
```

ii) Oracle UCP XML構成ファイルを使用して宣言的に設定します。（Oracle Call Interface（OCI）アプリケーションの場合、Oracleクライアント・ドライバ12.2以降が必要です）。
oraaccess.xmlファイルの<default_parameters>セクションに次を追加します。

```
<default_parameters>
（その他の設定がこのセクションに含まれる場合があります）
<events>
True
</events>
<ons>
<auto_config>true</auto_config>
<wallet_location>/path/onswallet</wallet_location>
</ons>
</default_parameters>
```

<wallet_location>パスは、ウォレットを含むディレクトリの名前である必要があります。<hosts>、<max_connections>、<subscription_wait_timeout>を含むその他のパラメータが、oraaccess.xmlのonsセクションに設定される場合があります。ネイティブ・イベント設定制御をサポートしているドライバは、<events>セクションを省略し、代わりにドライバ設定を使用する場合があります。デフォルトでは、ONSが失敗したとしても、データベースへの接続が確立されます。このシナリオで接続が失敗する方がよい場合は、<events>および<ons>と同じレベルにセクションを追加できます。

```
<fan>
    <subscription_failure_action> error</subscription_failure_action>

</fan>
```

tnsnames.oraおよびsqlnet.oraネットワーク・ファイルと同じディレクトリにoraaccess.xmlファイルを配置します。たとえば、Oracle Instant Clientを使用する場合、これらのファイルはデフォルトのディレクトリnetwork/admin内にある可能性があります。または、すべてのネットワーク構成ファイルを別のアクセス可能なディレクトリに配置することもできます。配置したら、環境変数TNS_ADMINをそのディレクトリの名前に設定します。

アプリケーションに影響しないメンテナンス

メンテナンス前のセッションのドレイニング

計画メンテナンスを行う場合は、メンテナンスを開始する前に、進行中の作業を完了させる時間を確保することを推奨します。これを行うには、Oracle接続プールとOracleドライバのFANまたは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します。また、Oracle Databaseも作業を直接ドレインします。Oracle Databaseに接続するサービスは、接続テストを行うように構成し、ドレイニングに使用できる時間をdrain_timeoutに指定し、ドレイニング・タイムアウトの終了後に適用されるstopoptionにIMMEDIATEを指定します。停止、再配置、スイッチ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、drain_timeoutとstopoptionが含まれています。

計画メンテナンスが開始されると、FANをサブスクライブしているすべてのアプリケーションに対してFANの計画停止イベントが公開されます。このFANイベントを受信すると、FAN対応のプールは、アイドル状態のセッションをクリアし、リクエストが完了したとき、または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を解放対象としてマークすることで対応します。このFANイベントにより、drain_timeoutによって指定された時間中、作業を中断することなく、インスタンスからセッションがドレインされます。一部のセッションがチェックインしないままdrain_timeoutに達した場合は、stopoptionが適用されて（IMMEDIATEを使用して）サービスが停止します。Oracle Database 18c以降、ドレインするセッションがマークされます。

マークされたら、アプリケーションを中断することなく、セッションをドレインするための安全な場所を見つけるためのルールが適用されます。ドレイニングはdrain_timeout時間中、続けられます。

ドレイニング用に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、選択したフェイルオーバー・ソリューションと組み合わせてドレイニングを使用します。割り当てられた時間内にドレイニングが完了しなかったセッションについては、フェイルオーバー・ソリューションによってリカバリが試行されます。ベスト・プラクティスに従って計画メンテナンスを行う場合は、アプリケーション・サーバーを再起動する必要はありません。

計画ドレイニング用の構成

1つまたは両方の**手段**を使用することで、計画メンテナンスは、Oracle Databaseに接続されたアプリケーションから不可視化されます。計画メンテナンスの前に、データベース・インスタンスでのデータベース・セッションをドレインまたはフェイルオーバーし、アプリケーションの作業が中止されないようにします。

推奨手段

計画メンテナンスを不可視化するために推奨されるソリューションは、FANに対応したOracle接続プールを使用する方法です。Oracleプールは、ドレイニング、再接続、MAAシステム内でのリバランシングといったライフサイクル全体に対応しています。メンテナンスの進行に合わせてセッションの移動とリバランシングが行われ、完了するとまた移動とリバランシングが行われます。アプリケーションでOracleプールとFANが使用され、リクエスト間でプールに接続が戻される場合、ユーザーに影響はありません。FANを使用するためにアプリケーション変更などを行う必要は一切ありません。

Oracleプールは、FANイベントを受信すると、ドレインするインスタンスに接続されているすべての接続をマークします。即座に、チェックインされた接続がクローズされ、これらは再使用できなくなります。使用中の接続は、接続プールに次回チェックインされるときにクローズ対象としてマークされたままになります。これらの接続は、プールに返されるときに、段階的にリリースされます。

アプリケーションでの使用法としては、必要なときに接続をチェックアウトし、現在の操作が完了したら接続をプールにチェックインするのがベスト・プラクティスです。これは、アプリケーションを最高のパフォーマンスで実行するため、および実行時、メンテナンス時、フェイルオーバー・イベントの発生時に作業をリバランスするために重要です。

サード・パーティのJavaベースのアプリケーション・サーバーを使用している場合、もっとも効果的にドレイニングやフェイルオーバーを行うには、プールされたデータソースの代わりにOracle UCPを使用します。このアプローチは、Oracle WebLogic Server、IBM WebSphere、IBM Liberty、Apache Tomcat、Red Hat WildFly（JBoss）、Spring、Hibernate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。オラクルとIBMなどのその他のベンダーの両方による技術概要で、Oracle UCPをこれらのアプリケーション・サーバーと一緒に使用する方法が説明されています。Oracle 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、十分に動作保証されたOracle UCPの機能を使用できます。oracle.comの「アプリケーション・コンティニュイティ」と「JDBC」について説明されたテクノロジー・ページで、多くの顧客事例研究を参照できます。

代替手段 – Oracle Database 19cのドレイニングまたはOracleドライバ19cのドレイニング

OracleプールとFANを使用できない場合、Oracle Database自体がセッションをドレインします。このデータベースは、接続をリリースするのに安全な場所を探し始めます。このデータベースは、一連の拡張可能なルールと経験則を使用して、データベース・セッションを取り除くタイミングを計ります。ドレイニングが開始されたら、ルールが満たされるまでデータベース・セッションがデータベース上で継続されます。このルールには、次が含まれます。

- 接続の有効性を検証する標準アプリケーション・サーバー接続テスト
- 有効性を検証するカスタムSQLテスト
- リクエスト境界の有効化と現在のリクエストの終了

接続テストについては、接続プールから取得される場合、プールに返される場合、バッチがコミットされる場合に、アプリケーション・サーバー、プール済みアプリケーション、ジョブ・スケジューラなどで接続がテストされるのが標準的です。ドレイニング時間中、データベースは接続テストをインターセプトし、接続をクローズし、失敗したテストのステータスを返します。接続テストを発行するアプリケーション・レイヤーは、失敗した返却のステータスを処理する準備が整っており、異なる接続を取得するための別のリクエストを発行します。このアプリケーションは中断されません。

帯域内FANを使用したOracle Driver 19cのドレイニング

Oracle Database 18c以降、ドライバは、Oracle Database、Driver 18c以降と一緒に、計画停止用の帯域内FANイベントを受信し、ドレイニングを可能にします。ドライバは、接続を安全にクローズするためにリクエストのステータスと接続テストの終了を探します。ドライバは、connection.isValid()、CheckConStatus、またはOCI_ATTR_SERVER_STATUSなど、SQLやPINGベースを使用しない接続テストを使用します。リクエスト境界の使用は、Oracle Database 12c以降のプール、またはJDK9以降を使用しているJavaプールを利用します。アプリケーションは、ベスト・プラクティスに従い、リクエストが完了したときにこれらのプールに接続を返す必要があります。

接続テスト

pingとリクエストの終了：

停止に応じて、アプリケーションは、高速接続フェイルオーバー（FCF）が処理される前に、接続が取得される猶予期間内に古い接続を受信する場合があります。これは、たとえば、着信接続要求と一致してソケットがクローズされるときにクリーン・インスタンスで発生する場合があります。アプリケーションがエラーを受信するのを阻止するには、接続プールで接続チェックを有効化する必要があります。

JDBC Universal Connection Pool

> setValidateConnectionOnBorrow() – 接続プールから接続を取得するときに接続を検証するかどうかを指定します。このメソッドは、プールから取得したすべての接続の検証を可能にします。デフォルト値はfalseです。値をtrueに設定し、検証が実行されるようにします。

JDBCドライバ接続テスト

> 接続プールを使用する場合、connection.isValid()またはSQL接続テストをプロパティ・ファイルに設定できます。connection.isValid()は、12.2以降のローカル・コールであり、接続に関するリクエストをデータベースに送信する前に、すべてのJavaベースの接続プールから接続を取得する場合、およびジョブ・スケジューラまたは同様のツールを使用する場合も、デフォルトでその使用を有効化することを推奨します。

OCI接続

> サーバーへの接続がFANまたはOCI_ERRORのどちらによって切断されたかを検証するには、アプリケーションでサーバー・ハンドル内の属性OCI_ATTR_SERVER_STATUSの値をコード・チェックする必要があります。FANイベントに応じて、OCIレイヤーは、サービスが停止している場合、OCI_ATTR_SERVER_STATUS属性をOCI_SERVER_NOT_CONNECTEDに設定します。スケジュールされたメンテナンスの場合、作業が完了するまでの猶予期間を設けるために、このステータスは接続を削除せずに設定されます。FANを使用して計画メンテナンスを報告する場合、アプリケーションは、取得の前かつプールへの返却の後にOCI_ATTR_SERVER_STATUSを確認し、これらの安全な場所のみのセッションを削除します。取得の前かつ返却の後にクローズされた接続を削除すると、計画メンテナンス時にアプリケーション・エラーが発生することなく、優れたユーザー・エクスペリエンスを実現できます。FANを使用して計画外停止を報告する場合、アプリケーションは、即座にエラーを受信します。これが、コードを必要とする唯一の接続テストです。その他すべての接続テストは構成可能です。

ODP.NETプロバイダ

> CheckConStatusはデフォルトで有効になっています。このプロパティを使用して、接続をODP.NET接続プールに返す前に、接続のステータスを確認することを可能にします。

SQL接続テスト

すべてのアプリケーション・サーバーには、各接続プール内の接続の有効性をテストするための機能があります。これは、接続プロパティによって設定されるか、管理コンソールで設定されます。テストの目的は、使用不可の接続をアプリケーションに提供することを防止したり、使用不可の接続が検出されたときに、これがプールにリリースされる際に削除したりすることです。

さまざまなアプリケーション・サーバー全体で、テストの名前は似ています。提供されたテストにはさまざまな手段が使用されていますが、最も一般的な手段はSQL文です。オラクルでは、Javaアプリケーション・サーバーに標準Javaコールconnection.isValid()を使用することを推奨します。Oracle Database 18c以降、データベースをドレインするためにすべてのテストが使用されます。また、Oracle Database 18c以降、データベースは、セッションの安全なドレイニング・ポイントを検査することで、FANを使用せずにセッションをドレインします。サービスが停止または再配置されると、検査が自動的に開始されます。サーバーのドレイニングはすべてのドライバをサポートし

ています。
デフォルトですべてのデータベース・サービスとプラガブル・データベース・サービスに追加されたSQL接続テストは4つあるため、アプリケーションがこれらのSQLテストを使用する場合、これらはすでに対象になっています。

- SELECT 1 FROM DUAL;
- SELECT COUNT(*) FROM DUAL;
- SELECT 1;
- BEGIN NULL;END;

サーバーでのドレイニング用の接続テストの追加構成

サービス、プラガブル・データベース、非コンテナ・データベースに対する接続テストを追加、削除、有効化、または無効化できます。追加および有効化したこれらの接続テストは、ビューDBA_CONNECTION_TESTSを使用して確認します。次に例を示します。

デフォルトでは、ドライバはpingテストを使用します。ドライバ向けに有効化する必要はありません。pingを使用する任意のテスト（isValid、isUsable、OCIping、またはconnection.statusなど）をデータベースが使用するようにしたい場合、次のようなSQL文を使用します。

SQL> execute dbms_app_cont_admin.enable_connection_test(dbms_app_cont_admin.ping_test);

プラガブル・データベースまたは非コンテナ・データベースに新しいサーバー側のSQL接続テストを追加するには、非コンテナ・データベースにログオンし、次のようなSQL文を使用します。

SQL> execute dbms_app_cont_admin.add_sql_connection_test('select * from dual;',[SERVICE]);

透過的アプリケーション・コンティニュイティ

透過的アプリケーション・コンティニュイティは、Oracle Database 18c以降のアプリケーション・コンティニュイティのモードで、セッションとトランザクションの状態を透過的に追跡および記録し、リカバリ可能な停止の後にデータベース・セッションをリカバリできるようにします。これは安全に行われ、DBAがアプリケーションに関する知識を持つ必要も、アプリケーション・コードを変更する必要もありません。透過性は、アプリケーションからデータベースに対してコールが発行されたときのセッション状態の使用状況を分類する状態追跡インフラストラクチャを使用することで実現できます。状態追跡を使用することで、アプリケーションまたは環境が変化するときに透過的アプリケーション・コンティニュイティを使用するアプリケーションの未来を保証します。

透過的アプリケーション・コンティニュイティのカバレッジ

Oracle Autonomous Database向けの透過的アプリケーション・コンティニュイティは、次のクライアントをサポートしています。最新のクライアント・ドライバを使用することを強く推奨します。Oracle Database 19c以降のクライアント・ドライバは、TACを完全にサポートしています。

- Oracle JDBC Replay Driver 18c以降これは、Oracle Database 18c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。
- Oracle Universal Connection Pool（UCP）18c以降
- Oracle WebLogic Server 18c、およびOracle UCPを使用するサード・パーティ製JDBCアプリケーション・サーバー
- リクエスト境界のあるOracle JDBC Replay Driver 12c以降を使用した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
- OCIセッション・プール19c以降
- SQL*Plus 19c以降
- ODP.NET Unmanaged Provider 18c以降
- 19cのOCIドライバ以降を使用したOracle Call Interfaceベースのアプリケーション

サード・パーティ製のJavaベースのアプリケーション・サーバーを使用している場合、もっとも効果的に高可用性を実現するには、データ

ソースの代わりにOracle UCPを使用します。このアプローチは、Oracle WebLogic Server、IBM WebSphere、IBM Liberty、Apache Tomcat、Red Hat WildFly（JBoss）、Spring、Hibernate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。オラクルとIBMなどのその他のベンダーの両方によるホワイト・ペーパーで、Oracle UCPをこれらのアプリケーション・サーバーと一緒に使用する方法が説明されています。Oracle 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、十分に動作保証されたOracle UCPの機能を使用できます。

透過的アプリケーション・コンティニュイティを使用する場合の手順

サポートされているクライアントの使用（上記のカバレッジを参照） 接続プールへの接続の返却
アプリケーションが次の接続を使用する場合、リクエスト境界を特定するために変更を行う必要はありません。

- Oracle接続プールから、または
- Oracle JDBC Replay Driver 18c以降から、または
- 19c以降を使用したOCIベースのアプリケーションから

接続プールが説明されているとおりに機能するには、アプリケーションが必要なときに接続を取得し、使用していないときに接続をリリースする必要があります。この方が、接続を保持するよりスケーラビリティが向上し、使用するメモリ量も少なくなり、リクエスト境界が透過的に提供されます。ベスト・プラクティスは、必要とする場合にのみアプリケーションが接続をチェックアウトすることです。使用していないときに接続を保持することは、勧められる方法ではありません。

JDBCシン・ドライバのリリース18cを使用してTACを使用している場合、および19c以降のOCIベースのアプリケーションを使用している場合、リクエスト境界は透過的に検出および拡張されます。つまり、リソース使用率が削減され、リカバリが高速になります。なぜなら、リクエスト境界が自動的に拡張され、トランザクションとセッションの状態に貢献していない文が必要なくなったときに消去されるからです。Oracleプールを使用し、リクエスト間で接続をOracleプールに返すのが、依然としてベスト・プラクティスです。

FAILOVER_RESTOREの使用

TACは、デバイス上でFAILOVER_RESTOREをAUTOに設定することで、事前設定の状態を自動的にリストアします。

標準セットに加えて事前設定されたセッションの状態が必要である場合、コールバックまたはUCPラベルを登録し、これらの状態をリストアできます。複雑なセッションの状態（一時表またはsys_contextなど）をリストアする必要がある場合、この柔軟性を備えたアプリケーション・コンティニュイティを使用してコールバックを使用してください。

アプリケーションでのmutable使用の有効化

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。Oracle Database 19ではSQLの可変値が自動的に保持されます。

PL/SQLに可変値が必要である場合、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、関数の元の結果が再実行時に適用されます。たとえば、次のとおりです。

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

```
SQL> GRANT KEEP SEQUENCE mySequence to myUser on sequence.object
```

副次的作用の無効化

通常の実行時、透過的アプリケーション・コンティニュイティでは、副次的作用が検出され、これらは再実行されません。副次的作用とは、メールの送信、ファイルの転送、TCPの使用などの外部アクションです。副次的作用のタイプは、アプリケーションのロジックに関連するものと、データベースのハウスキーピングに関連する内部的なものの間で区別されます。副次的作用がある文を使用するアプリケーションの場合、文の実行時はキャプチャが無効になります。キャプチャは、次の検出済みリクエスト境界または明示的なリクエスト境界で再び

自動的に有効になります。キャプチャは、暗黙的リクエスト境界をサポートしているため、可能な場合、Java 18c再実行ドライバおよびOCI 19cドライバに対してTACによって再び自動的に有効になります。副次的作用を再実行したい場合、この柔軟性を備えたアプリケーション・コンティニュイティを使用してください。

制限事項

制限事項の最新のリストについては、ドキュメントを参照してください。

アプリケーション・コンティニュイティのカバレッジ

アプリケーション・コンティニュイティはカスタマイズできるため、副次的作用（メールなど）を再実行したり、TACでは許可されないフェイルオーバー時の複雑なコールバックを追加したりすることを選択できます。リリース12.2を使用している場合、または副次的作用またはコールバックありでカスタマイズしたい場合、またはセッション持続期間一時表などの状態を使用し、クリーンアップしないアプリケーションがある場合は、ACを使用してください。

アプリケーション・コンティニュイティのカバレッジ

Oracle Database 18cのアプリケーション・コンティニュイティでは、次のクライアンをサポートしています。

- 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。
- Oracle Universal Connection Pool（UCP）12c以降
- Oracle WebLogic Server 12c、およびOracle UCPを使用するサード・パーティ製JDBCアプリケーション・サーバー
- リクエスト境界のあるOracle JDBC Replay Driver 12c以降を使用した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 Call Interface Session Pool 12c Release 2以降を使用したアプリケーションと言語ドライバ
- SQL*Plus 19c以降
- ODP.NET Unmanaged Provider 12c Release 2以降（“Pooling=true”、“Application Continuity=true”は12.2以降はデフォルト）

サード・パーティ製のJavaベースのアプリケーション・サーバーを使用している場合、もっとも効果的に高可用性を実現するには、データソースの代わりにOracle UCPを使用します。このアプローチは、Oracle WebLogic Server、IBM WebSphere、IBM Liberty、Apache Tomcat、Red Hat WildFly（JBoss）、Spring、Hibernate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。Oracle 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、動作保証されたOracle UCPの機能を使用できます。

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手順

サポートされているクライアントの使用（上記のカバレッジを参照） 接続プールへの接続の返却
アプリケーションは、リクエストごとに接続を接続プールに返却します。ベスト・プラクティスは、必要とする場合にのみアプリケーションが接続をチェックアウトすることです。使用していないときに接続を保持することは、勧められる方法ではありません。したがって、アプリケーションでは、接続をチェックアウトし、その後処理が完了したら、すぐにその接続をチェックインする必要があります。続いて接続は、後続の他のスレッドまたは再び必要になったときに自分のスレッドで使用できるようになります。この手法に従うことにより、キャプチャを再開および終了するのに安全な場所を識別するためにアプリケーション・コンティニュイティで使用する明示的なリクエスト境界も組み込まれます。ACでは、TACの場合のように、検出された追加のリクエスト境界が提供されることはありません。

FAILOVER_RESTOREの使用

一部のアプリケーションと中間層アプリケーションでは、たとえばすべての接続の言語またはタイムゾーンが事前設定されるように、接続プールを構成します。リクエスト以外で接続にセッションの状態を意図的に設定し、この状態を想定してリクエストを行う場合は、再実行の前にこの状態を作成し直して再実行する必要があります。

もっとも一般的な状態は、FAILOVER_RESTOREをLEVEL1に設定することによって自動的にリストアされます。標準セットに加えてセッションの状態を事前設定する場合、コールバックまたはUCPラベルを登録し、これらの状態をリストアする必要があります。本書のこの後にある追加資料内のアプリケーション・コンティニュイティの技術概要を参照してください。

追加のセッションの状態が必要な場合は、次のいずれかのオプションを使用してください。

- 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（古い）TAFコールバック（OCIの場合）
- Oracle Universal Connection PoolまたはOracle WebLogic Serverの接続ラベル付け

アプリケーションでのmutable使用の有効化

mutableに関するTACのセクションを参照してください。

再実行時に副次的作用を繰り返すかどうかの決定

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メールを再送信するか、またはファイルを再送信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。すべての副次的作用を再実行しない場合は、透過的アプリケーション・コンティニュイティを使用します。

透過的アプリケーション・フェイルオーバー

透過的アプリケーション・フェイルオーバー（TAF）は、OCI、OCCI、Java Database Connectivity（JDBC）OCIドライバ、ODP.NETのクライアント側の機能です。この機能は、接続の初期設定後にOracleの状態を変更しないアプリケーション用の接続またはSELECT文をフェイルオーバーします。接続をフェイルオーバーするにはFAILOVER_TYPEをBASICに設定し、SELECT文を再接続および再実行するにはSELECTに設定します。TAFの事前接続は、推奨されるオプションではありません。なぜなら、実際の状況下でアプリケーションが事前接続されないことが原因で、フェイルオーバー時に遅延が発生するからです。

Oracle Database向けの透過的アプリケーション・フェイルオーバーは、次のクライアントをサポートしています。

- Oracle Call Interface（OCI）
- Oracle C++ Call Interface（Oracle CCI）
- Oracle JDBC OCIドライバ（一般的にシック・ドライバは推奨されません）
- ODP.Net Unmanaged Provider
- OCIを使用したOracle Tuxedo
- OCIセッション・プール

透過的アプリケーション・フェイルオーバーを使用する場合の手順

サポートされているクライアントの使用（上記の対象範囲を参照） FAILOVER_RESTOREの使用
Oracle Database 12.2以降、アプリケーションが接続の値を事前設定しているかどうかをチェックします。一部のアプリケーションと中間層アプリケーションでは、たとえば、すべての接続の言語またはタイムゾーンが事前設定されるように、接続プールを構成します。リクエスト以外で接続にセッションの状態を意図的に設定し、この状態を想定してリクエストを行う場合は、再実行の前にこの状態を作成し直して再実行する必要があります。

もっとも一般的な状態は、FAILOVER_RESTOREをLEVEL1に設定することによって自動的にリストアされます。標準セットに加えてセッションの状態を事前設定する場合、TAFコールバックを登録し、これらの状態をリストアする必要があります。

次のオプションを一緒に使用します。

FAILOVER_RESTORE=LEVEL1（サービスで設定）
TAFコールバック（OCIの場合）

Oracle RDBMS 12.2以降では、コールバックがまだない場合、FAILOVER_RESTORE=LEVEL1が推奨される方法です。

トランザクション・ガードを使用したTAF

Oracle Database 12.2以降、フェイルオーバー時に、透過的アプリケーション・フェイルオーバー（TAF）は新しい接続を取得し、トランザクション・ガードが有効になっている場合は、トランザクション・ガードを起動してコミット結果を強制します。トランザクション・ガードがコミット済みおよび完了を返した場合、TAFは処理を続行し、アプリケーションにエラーは返されません。トランザクション・ガードが未コミットまたはコミット済みを返したものの、完了を返さなかった場合、TAFはアプリケーションにTAFエラーを返します。TAFは新しい接続を維持します。TAFとトランザクション・ガードの両方を使用する場合、開発者はTAFエラー・コード（ORA-25402、ORA-25408、ORA-25405）を使用して、トランザクションを再送信するか、またはユーザーにコミットされていないことを示すメッセージを返すかを決定できます。TAFとトランザクション・ガードの両方が有効になっている場合にのみ、これらのエラー・コードに基づき安全に再送信する、または未コミットを返すことができます。TAFのみが有効になっている場合は、再送信する、または未コミットを返すのは、安全ではありません。アプリケーション・コンティニュイティがこの再送信を開発者に代わって遂行します。

この後で説明するトランザクション・ガードの技術概要を参照してください。

TACまたはACの使用時の保護レベルの把握

保護レベルの統計情報

リクエスト境界と保護レベルに関する統計情報を使用してカバレッジのレベルを監視します。アプリケーション・コンティニュイティを使用する場合はシステム、セッション、サービスから統計情報が収集されるため、保護レベルを監視できます。統計情報はV$SESSTAT、V$SYSSTATで確認でき、Oracle Database 19c以降のバージョンではV$SERVICE_STATSで確認できます。これらの統計情報は自動ワークロード・リポジトリに保存され、自動ワークロード・リポジトリ・レポートで確認できます。

統計	合計	1秒あたり	トランザクションあたり
累積開始リクエスト数	1,500,000	14,192.9	2.4
累積終了リクエスト数	1,500,000	14,192.9	2.4
リクエスト内の累積ユーザー・コール数	6,672,566	63,135.2	10.8
保護された累積ユーザー・コール数	6,672,566	63,135.2	10.8

ヒント：保護レベルをPDB別に、または履歴データを使用して報告するには、付録（使用するSQLなど）を参照してください。出力は次のようになります。
次の場合、TACまたはACは有効であり、アプリケーションを保護しています。

- リクエスト内の累積ユーザー・コール数 = 保護された累積ユーザー・コール数
- 上記の数字がゼロと等しくない

この保護レベルは、データベース内で測定されます。この保護レベルを実現するためにFAILOVER_RESTOREの対象とならない状態がこの処理に含まれる場合、クライアントは、問合せにORDER BY句を使用し、初期のセッションの状態を事前設定することが必要な場合があります。

上記の例は、開始リクエスト数と終了リクエスト数が増加していることを示しています。この数自体は、アプリケーションが接続プールからチェックアウトまたは接続プールにチェックインする頻度、またはTACの使用時にデータベースが検出できるリクエスト境界の内容によって異なります。これらの値の増加率は、リクエストの送信率によって異なります。保護されたユーザー・コールの比率は、次の式を使用して計算できます。

保護されたコールの比率 = 保護された累積ユーザー・コール数 / リクエスト内の累積ユーザー・コール数 * 100
保護されたユーザー・コールの比率が100 %未満になることはあり得ます。JDBC具象クラスを使用している可能性があるか、副次的作用が無効であるか、リカバリ不可能な使用されている可能性があるか、またはアプリケーションが特定のリクエストに対してアプリケーション・コンティニュイティを無効にすることを選択する可能性があります。アプリケーションが100 %保護されない場合、ORAchkコンポーネント

acchkを独自のサイトで使用し、アプリケーション内のどの部分のカバレッジが100 %を下回っているかを表示できます。マネジメントは、アドバイザに従うか、または影響を評価して何の措置も講じないかを決定できます。

クライアントの構成

JDBCシン・ドライバのチェックリスト

1. すべての推奨パッチがクライアントに適用されていることを確認します。MOS Note『Client Validation Matrix for Application Continuity』（Doc ID 2511448.1）を参照してください。
2. プロパティ・ファイルまたはコンソールでOracle JDBC再実行データソースを構成します：
   setConnectionFactoryClassName("oracle.jdbc.replay.OracleDataSourceImpl");または
   setConnectionFactoryClassName("oracle.jdbc.replay.OracleXADataSourceImpl");（XAの場合）
3. カバレッジとパフォーマンスのためにJDBCステートメント・キャッシュを使用します。
   最善のカバレッジとパフォーマンスを実現するには、アプリケーション・サーバーのステートメント・キャッシュの代わりに、JDBCドライバのステートメント・キャッシュを使用します。これにより、ドライバはステートメントがクローズされることを認識できるようになるため、リクエストの終了時にメモリを解放できます。
   JDBCステートメント・キャッシュを使用するには、接続プロパティoracle.jdbc.implicitStatementCacheSize
   （OracleConnection.CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE）を使用します。キャッシュ・サイズの値はopen_cursorsの数と同じです。例：oracle.jdbc.implicitStatementCacheSize=nnn。この場合、nnnは通常、10と100の間であり、アプリケーションが維持しているオープン・カーソルの数と等しいです。
4. ガベージ・コレクタをチューニングします。多くのアプリケーションでは、ガベージ・コレクタのデフォルト・チューニングで十分です。大量のデータを返すアプリケーションや保持するアプリケーションの場合は、2 G以上といった高い値を使用できます。例：java -Xms3072m -Xmx3072m。Javaの初期ヒープ・サイズ（ms）のメモリ割当てと最大ヒープ・サイズ（mx）のメモリ割当てを同じ値に設定することを推奨します。この設定により、メモリ・ヒープの増加および縮小で、システム・リソースが使用されなくなります。
5. コミット

   JDBCアプリケーションでAUTOCOMMITを使用する必要がない場合は、アプリケーションまたは接続のプロパティでAUTOCOMMITを無効にします。Apache Tomcat、IBM WebSphere、IBM Liberty、Red Hat WildFly（JBoss）などのサード・パーティ製アプリケーション・サーバーにOracle UCPまたは再実行ドライバが埋め込まれている場合に、この設定は重要です。

   Oracle UCPのPoolDataSource接続プロパティを使用して、autoCommitをfalseに設定します。
   connectionProperties="{autoCommit=false}"
6. JDBC具象クラス
   JDBCアプリケーションについては、非推奨のoracle.sql具象クラス（BLOB、CLOB、BFILE、OPAQUE、ARRAY、STRUCT、またはORADATA）はオラクルでサポートされません（MOS Note 1364193.1『New JDBC Interfaces』を参照）。アプリケーションに問題がないことをクライアントで認識するには、ORAchk -acchkを使用します。Oracle JDBCシン・ドライバのバージョン18c以降、JDBC Replay Driverの制限付き具象クラスのリストは次のとおりに削減されています。oracle.sql.OPAQUE、oracle.sql.STRUCT、oracle.sql.ANYDATA。
7. 高速接続フェイルオーバー（FCF）を構成します。

   クライアント・ドライバ12c以降の場合

   - auto-onsで推奨されているURLを使用します
   - ons.jar（およびオプションのWALLET jar、osdt_cert.jar、osdt_core.jar、oraclepki.jar）がCLASSPATHに存在することを確認します
   - プール・プロパティまたはドライバ・プロパティfastConnectionFailoverEnabled=trueを設定します
   - サード・パーティ製のJDBCプールの場合は、UCPが推奨されます
   - ONS用にポート6200を開きます（6200はデフォルト・ポートです。別のポートが選択されている場合もあります）

   12c以前のクライアント・ドライバの場合、提供されているアドレスを使用します。

oracle.ons.nodes =XXX01:6200, XXX02:6200, XXX03:6200を設定します。

OCI（Oracle Call Interface）ドライバのチェックリスト

1. すべての推奨パッチがクライアントに適用されていることを確認します。MOS Note『Client Validation Matrix for Application Continuity』（Doc ID 251148.1）を参照してください。
2. OCIStmtPrepareをOCIStmtPrepare2に置き換えます。OCIStmtPrepare()は12.2以降、非推奨になっています。すべてのアプリケーションでOCIStmtPrepare2()を使用する必要があります。TACとACではOCIStmtPrepareを使用できますが、この文は再実行しないでください。詳しくは、こちらを参照してください。
3. OCIベースのアプリケーションにFANを使用するには、以下を実行します。
    - ATP-Dがサービスでaq_ha_notificationsを事前設定します
    - auto-onsで推奨される接続文字列を使用します
    - oraaccess.xmlでauto_config、events、wallet_location（オプション）を設定します（付録を参照）
    - アプリケーションをO/Sクライアント・スレッド・ライブラリにリンクさせます
    - ONS用にポート6200を開きます（6200はデフォルト・ポートです。別のポートが選択されている場合もあります）

12c以前のクライアント・ドライバの場合、oraccess.xmlで提供されているアドレスを使用します。

ODP.NET管理対象外のプロバイダ・ドライバのチェックリスト

1. すべての推奨パッチがクライアントに適用されていることを確認します。MOS Note『Client Validation Matrix for Application Continuity』（Doc ID 251148.1）を参照してください。
2. OCIベースのアプリケーションにFANを使用するには、以下を実行します。
    - ATP-Dがサービスでaq_ha_notificationsを事前設定します
    - auto-onsで推奨される接続文字列を使用します
    - oraaccess.xmlでonsConfigとwallet_location（オプション）を設定します（付録を参照）
    - ONS用にポート6200を開きます（6200はデフォルト・ポートです。別のポートが選択されている場合もあります）
    - 接続文字列でFANを設定します

    "user id=oracle; password=oracle; data source=HA; pooling=true; HA events=true;“
    - （オプション）次の接続文字列で、ランタイム・ロードバランシングを設定します

    "user id=oracle; password=oracle; data source=HA; pooling=true; HA events=true; load balancing=true;“

まとめ

Oracle Database 19cは、ユーザーに代わって高可用性を実現するために構成および管理されます。ユーザーによる追加の構成または管理は必要ありません。

アプリケーションに対して継続的な可用性を実現するためのシンプルな手順がいくつか用意されています。

- SLAに適したデータベース・サービスを選択します
- 高速アプリケーション通知（FAN）を構成します
- アプリケーションで推奨されている接続文字列を使用します
- ドレイニングを最適化するためのアプリケーションのベスト・プラクティスを使用します
- 継続的なサービスのための透過的アプリケーション・コンティニュイティまたはアプリケーション・コンティニュイティを使用します

これら5つのシンプルな手順に従うことで、計画メンテナンス・アクティビティには停止が必要なくなります。また、計画外イベントの結果として、大半の事例でトランザクションが失敗したりユーザー向けサービスが中断したりしなくなります。

付録：サービスの構成

Oracle Databaseを使用する場合、透過的アプリケーション・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、またはTAFを使用するOracle RAC上でサービスを作成できます。サービスがActive Data Guardまたはプライマリ・データベース上でアクティブであるかを区別するためにロールを使用できます。次の例では、これを説明するためにGOLDという名前のサービスを追加しています。

透過的アプリケーション・コンティニュイティ

```
$ srvctl add service -db mydb -service GOLD –preferred inst1 -available serv2 - failover_restore AUTO -failoverretry 1 -failoverdelay 3 -commit_outcome TRUE -failovertype AUTO -replay_init_time 600 -retention 86400 -notification TRUE -drain_timeout 300 - stopoption IMMEDIATE
```

または、複数のアクティブ・インスタンスを持つサービスが必要である場合（およびこの場合、使用可能なインスタンスはありません）

```
$ srvctl add service -db mydb -service GOLD -preferred inst1,inst2 -failover_restore AUTO - failoverretry 1 -failoverdelay 3 -commit_outcome TRUE -failovertype AUTO -replay_init_time 600 -retention 86400 -notification TRUE -drain_timeout 300 -stopoption IMMEDIATE
```

透過的アプリケーション・フェイルオーバー

```
$ srvctl add service -db mydb -service BRONZE -preferred inst1 -available inst2 - failover_restore LEVEL1 -failoverretry 1 -failoverdelay 3 - commit_outcome TRUE - failovertype SELECT -retention 86400 -notification TRUE -drain_timeout 300 -stopoption IMMEDIATE
```

Oracle Data Guardのロールを使用して追加する場合、TACの例を次に示します

```
$ srvctl add service -db mydb -service GOLD -preferred inst1 -available inst2 - failover_restore AUTO -failoverretry 1 -failoverdelay 3 - commit_outcome TRUE -failovertype AUTO -replay_init_time 600 -retention 86400 -notification TRUE -role PHYSICAL_STANDBY - drain_timeout 300 -stopoption IMMEDIATE
```

推奨されるデータベースの接続文字列またはURL（前述）が使用されている場合、再試行はOracle Database Net Servicesによって管理されます。

計画ドレイニング – サーバー側の操作

企業の多くは多数のサービスを実行しています。これは、単一のデータベースまたはインスタンスが提供している多数のサービスであろうと、多数のデータベースが提供している同じノード上で実行されている数個のサービスであろうと関係ありません。Oracle Database 12c Release 2以降、個々のサービスごとにSRVCTLコマンドを実行する必要がなくなり、影響を受けるすべてのサービスに関するノード名、データベース名、プラガブル・データベース名、またはインスタンス名を指定するだけですみます。

DRAIN_TIMEOUTとSTOPOPTIONは両方とも、サービスを追加するとき、またはサービスを作成後に変更するときに定義できるサービス属性です。また、SRVCTLを使用してこれらの属性を指定することもできます。これは、サービスに定義されている内容よりも優先されます。サービスのグループの操作全体にわたって、このグループの最大のDRAIN_TIMEOUTが適用されます。次の例を参照してください。

RAC上でデータベース、ノード、またはPDB別にすべてのサービスを再配置する

```
$srvctl relocate service -database <db_unique_name> -oldinst <old_inst_name> [-newinst <new_inst_name>] -drain_timeout <timeout> -stopoption <stop_option>

$srvctl relocate service -database <db_unique_name> -currentnode <current_node> [- targetnode <target_node>] -drain_timeout <timeout> -stopoption <stop_option>

$srvctl relocate service -database <db_unique_name> -pdb <pluggable_database> {-oldinst <old_inst_name> [-newinst <new_inst_name>] | -currentnode <current_node> [-targetnode <target_node>]} -drain_timeout <timeout> -stopoption <stop_option>
```

inst1という名前のインスタンス（任意のインスタンス）でGOLDという名前のサービスを停止する

$srvctl stop service –db myDB –service GOLD –instance inst1 -drain_timeout <timeout> - stopoption <stop_option>

すべてのインスタンスでGOLDという名前のサービスを停止する

$srvctl stop service –db myDB –service GOLD -drain_timeout <timeout> -stopoption <stop_option>

inst1という名前のインスタンス（任意のインスタンス）でGOLDという名前のサービスを開始する

$srvctl start service –db myDB –service GOLD –instance inst1

1つのノードですべてを開始/停止する

$srvctl stop service -node <node_name> -drain_timeout <timeout> -stopoption <stop_option>

$srvctl stop service -db <db_unique_name> [-node <node_name> | -instance <inst_name>] - drain_timeout <timeout> -stopoption <stop_option>

$srvctl stop service -db <db_unique_name> [-node <node_name> | -instance <inst_name>] - pdb <pluggable_database> -drain_timeout <timeout> -stopoption <stop_option>

Data Guardのスイッチオーバー

switchover to <db_resource_name> [wait [xx]];

付録：具象クラスをチェックするためのacchkの使用

このチェックは、Javaアプリケーションにのみ適用されます。このチェックを使用して、推奨されないOracle JDBC具象クラスをJava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、アプリケーション・コンティニュイティ・チェック（Oracle ORAchkのacchkと呼ばれる）を使用します。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を検証します。

JDBCドライバ・バージョン12.2.0.2以前の場合、アプリケーション・コンティニュイティでは、変数型、キャスト、メソッドの戻りタイプ、またはコンストラクタの呼出し用として、形式ARRAY、BFILE、BLOB、CLOB、NCLOB、OPAQUE、REF、またはSTRUCTのoracle.sqlの推奨されない具象クラスを使用するトランザクションを再実行することはできません。

JDBCドライバ・バージョン18c以降の場合、アプリケーション・コンティニュイティでは、変数型、キャスト、メソッドの戻りタイプ、またはコンストラクタの呼出し用として、形式OPAQUE、REF、またはSTRUCTのoracle.sqlの推奨されない具象クラスを使用するトランザクションを再実行することはできません。

アプリケーション・コンティニュイティでアプリケーションを保護するには、推奨されないJavaを削除する必要があります。

Oracle具象クラスに関するアプリケーション・コンティニュイティ・チェックを制御する値は、4つあります。これらの値は、コマンドラインで、またはシェル環境変数を介して、またはこれら両方を使用して設定します。これらの値は次のとおりです。

表2-1 具象クラスのアプリケーション・コンティニュイティ・チェック

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	これは、asm-all-XXX.jarのバージョン（ダウンロード元：ASMのホームページ）を参照する必要があります。

-javahome JDK8dirname	RAT_JAVA_HOME	JAVA_HOMEディレクトリを参照する必要があります。
-appjar dirname	RAC_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。このプログラムは、.classファイルを分析し、.jarファイルとディレクトリを繰り返し分析します。
-jdbcver	RAC_AC_JDBCVER	カバレッジ・チェックのターゲット・バージョン

アプリケーション・コンティニュイティでの具象クラス・チェックの概要例

次のコマンドは、Oracle具象クラスに関するアプリケーション・コンティニュイティをチェックします。

$ ./orachk -acchk -asmhome /path/orachk/asm-5.0.3/lib/all/asm-all-5.0.3.jar -javahome /usr/lib/jvm/jre-1.8.0-openjdk.x86_64 -jdbcver 19.1 -appjar /scratch/nfs/tmp/jarfiles

停止のタイプ	ステータス	Message
具象 クラス・チェック		Total :114 Passed :110 Warning :0 Failed :2 (Failed check count is one per file)
	Failed	[ac/workload/lobsanity/AnydataOut][[CAST]desc= oracle/sql/ANYDATAmethodname=getDataInfo,lineno= 38]

付録：アプリケーション・コンティニュイティのためのacchkの使用

破壊的テストは有効なテストです。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。

AC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、完全には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・リリースを保護する方法を把握しています。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、標準的なデータベース・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、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は12c以降である必要があります。アプリケーションは、必ずしもアプリケーション・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。このチェックは、リリース前に役に立ちます。

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のキャプチャが有効であるときに、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であるときに、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が呼び出された後）、これは保護されていないコールとしてカウントされます。
- キャプチャと再実行によって無視されたラウンドトリップは、保護レベルの統計で無視されます。

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・レポートの実行

データベース・レベルでトレースをオンにします。

ワークロードの実行前に、必要な情報がトレース・ファイルに含まれるように、テスト用のOracle Databaseサーバー上で以下の文をDBAとして実行します。

SQL> alter system set event='10602 trace name context forever, level 28: trace[progint_appcont_rdbms]:10702 trace name context forever, level 16';
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作成するには、アプリケーション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、カバレッジ分析の情報が詳細になります。

Oracle ORAchkを使用して、収集したデータベース・トレースを分析し、保護レベルをレポートします。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レポートします。

保護レベルに対するアプリケーション・コンティニュイティ・チェックの使用

カバレッジ・レポートの例

$ ./orachk -acchk -javahome /tmp/jdk1.8.0_40 -apptrc
$ORACLE_BASE/diag/rdbms/$ORACLE_SID/trace

-javahome JDK8dirname	RAT_JAVA_HOME	JAVA_HOMEディレクトリを参照する必要があります。
-appjar dirname	RAC_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。このプログラムは、.classファイルを分析し、.jarファイルとディレクトリを繰り返し分析します。

カバレッジ・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。すべてのリクエストがPASS（Coverage(%) = 100）であることを確認するには、レポート・リポジトリのPASSレポートacchk_scorecard_pass.htmlをチェックしてください。

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を使用してクライアント側で設定可能）、ACTIONとCLIENT_ID（それぞれOCSID.ACTIONおよびOCSID.CLIENTIDとともにsetClientInfoを使用して設定可能）が含まれます。

出力例：ORACHK_ HTML#ACCHK_SCORECARD内

カバレッジ・チェック		TotalRequest = 1088 PASS = 1082 WARNING = 1 FAIL = 5.
	FAIL	Trace file name = orcl1_ora_30467.trc Line number of Request start = 1409 Request number =6 SERVICE NAME = (srv_auto_pdb1) MODULE NAME = (SQL*Plus) ACTION NAME = () CLIENT ID= () Coverage(%) = 12 Protected Calls = 1 Unprotected Calls =7
	WARNING	Trace file name = ATPCDB12_ora_321597.trc Line number of Request start = 653 Request number = 1 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (JDBC Thin Client) ACTION NAME = () CLIENT ID = () Coverage(%) = 25 Protected Calls = 1 Unprotected Calls = 3
	FAIL	Trace file name = ATPCDB12_ora_292714.trc Line number of Request start = 1598 Request number = 7 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (SQL*Plus) ACTION NAME = () CLIENT ID = () Coverage(%) = 16 Protected Calls = 1 Unprotected Calls = 5

	FAIL	Trace file name = ATPCDB12_ora_112022.trc Line number of Request start = 1167 Request number = 3 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (JDBC Thin Client) ACTION NAME = () CLIENT ID = () Coverage(%) = 0 Protected Calls = 0 Unprotected Calls = 1
	FAIL	Trace file name = ATPCDB12_ora_112022.trc Line number of Request start = 1689 Request number = 5 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (JDBC Thin Client) ACTION NAME = () CLIENT ID = () Coverage(%) = 0 Protected Calls = 0 Unprotected Calls = 1
	PASS	Report containing checks that passed: [Full Path]_060219_184513/reports/acchk_scorecard_pass.html

付録：PDB、サービス、履歴別に保護を報告するためのSQL

PDB別に保護を報告するには、次の例を使用します。

```
set lines 85
col Service_name format a30 trunc heading "Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests"
col Total_protected format 9,999,9999 heading "Calls Protected"
col Protected format 999.9 heading "Protected %"
select con_id, total_requests,
total_calls,total_protected,total_protected*100/NULLIF(total_calls,0) as Protected
from(
select * from
(select s.con_id, s.name, s.value
FROM GV$CON_SYSSTAT s, GV$STATNAME n
WHERE s.inst_id = n.inst_id
AND s.statistic# = n.statistic#
AND s.value != 0 )
pivot( sum(valu
e)
for name in ('cumulative begin requests' as total_requests, 'cumulative end requests' as
Total_end_requests, 'cumulative user calls in requests' as Total_calls, 'cumulative user
calls protected by Application Continuity' as total_protected)
))
order by con_id;
```

サービス別に保護を報告するには、次の例を使用します。

```
set pagesize 60
set lines 120
col Service_name format a30 trunc heading "Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests"
col Total_protected format 9,999,9999 heading "Calls Protected"
col Protected format 999.9 heading "Protected %"
select con_id, service_name,total_requests,
total_calls,total_protected,total_protected*100/NULLIF(total_calls,0) as Protected
from(
select * from
(select a.con_id, a.service_name, c.name,b.value
```

```
FROM gv$session a, gv$sesstat b, gv$statname c
WHERE a.sid = b.sid
AND a.inst_id = b.inst_id
AND b.value != 0
AND b.statistic# = c.statistic#
AND b.inst_id = c.inst_id
AND a.service_name not in ('SYS$USERS','SYS$BACKGROUND'))
pivot(
sum(value)
for name in ('cumulative begin requests' as total_requests, 'cumulative end requests' as
Total_end_requests, 'cumulative user calls in requests' as Total_calls, 'cumulative user
calls protected by Application Continuity' as total_protected) ))
order by con_id, service_name;
```

過去3日間の保護履歴に関して報告するには、次の例を使用します。

```
set lines 85
col Service_name format a30 trunc heading"Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests"
col Total_protected format 9,999,9999 heading "Calls Protected"
col Protected format 999.9 heading "Protected %"
set lines 85
col Service_name format a30 trunc heading"Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests"
col Total_protected format 9,999,9999 heading "Calls Protected"
col Protected format 999.9 heading "Protected %"
select a.instance_number,begin_interval_time,total_requests,
total_calls, total_protected, total_protected*100/NULLIF(total_calls,0) as Protected
from(
select * from
(select a.snap_id, a.instance_number,a.stat_name, a.value
FROM dba_hist_sysstat a
WHERE a.value != 0 )
pivot(
sum(value)
for stat_name in ('cumulative begin requests' as total_requests, 'cumulative end requests' as
Total_end_requests, 'cumulative user calls in requests' as Total_calls, 'cumulative user calls protected by
Application Continuity' as total_protected)
)) a,
dba_hist_snapshot b
where a.snap_id=b.snap_id
and a.instance_number=b.instance_number
and begin_interval_time>systimestamp - interval '3' day
order by a.snap_id,a.instance_number;
```

追加の技術概要

高速アプリケーション通知

http://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/learnmore/fastapplicationnotification12c-2538999.pdf

JAVAアプリケーション・サーバーにUCPを埋め込む方法

WLS UCP Datasource, https://blogs.oracle.com/weblogicserver/wls-ucp-datasource

『Design and Deploy WebSphere Applications for Planned, Unplanned Database Downtimes and Runtime Load Balancing with UCP』（http://www.oracle.com/technetwork/database/application-development/planned-unplanned-rlb-ucp-websphere-2409214.pdf）

『Reactive programming in microservices with MicroProfile on Open Liberty 19.0.0.4』（https://openliberty.io/blog/2019/04/26/reactive-microservices-microprofile-19004.html#oracle）

『Design and deploy Tomcat Applications for Planned, Unplanned Database Downtimes and Runtime Load Balancing with UCP』（http://www.oracle.com/technetwork/database/application-development/planned-unplanned-rlb-ucp-tomcat-2265175.pdf）

『Using Universal Connection Pool with JBoss AS』（https://blogs.oracle.com/dev2dev/using-universal-connection- pooling-ucp-with-jboss-as）

アプリケーション・コンティニュイティ

http://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/overview/application-continuity-wp-12c-1966213.pdf

『Ensuring Application Continuity』（https://docs.oracle.com/en/database/oracle/oracle-database/19/racad/ensuring-application-continuity.html#GUID-C1EF6BDA-5F90-448F-A1E2-DC15AD5CFE75）

透過的アプリケーション・フェイルオーバー

https://docs.oracle.com/en/database/oracle/oracle-database/23/adfns/high-availability.html#GUID-96599425-9BDA-483C-9BA2-4A4D13013A37

トランザクション・ガード

http://www.oracle.com/technetwork/database/database-cloud/private/transaction-guard-wp-12c- 1966209.pdf

『GRACEFUL APPLICATION SWITCHOVER IN RAC WITH NO APPLICATION INTERRUPTION』

My Oracle Support (MOS) Note：Doc ID 1593712.1

Connect with us

+1.800.ORACLE1までご連絡いただくか、**oracle.com**をご覧ください。北米以外の地域では、**oracle.com/contact**で最寄りの営業所をご確認いただけます。

blogs.oracle.com facebook.com/oracle twitter.com/oracle