19c ORACLE Database

継続的可用性

Application Continuity for the Oracle Database

Oracleホワイト・ペーパー / 2020年8月27日

ORACLE

目次

概要 3
アプリケーションの継続的可用性を維持するための機能セット 4
アプリケーションの構成手順 5
アプリケーションの継続的可用性を支える要素 6
高速アプリケーション通知（FAN） 8
アプリケーションに影響を与えないメンテナンス 12
透過的アプリケーション・コンティニュイティ 15
アプリケーション・コンティニュイティ 17
透過的アプリケーション・フェイルオーバー 18
TACまたはACを使用する場合の保護レベルの把握 20
クライアントの構成 20
結論 23
付録：サービスの構成 24
付録：ACCHKを使用した、具象クラスのチェック 26
付録：ACCHKを使用したアプリケーション・コンティニュイティのカバレッジ 28
付録：SQLを使用した、PDB、サービス、および履歴による保護のレポート 31
その他のホワイト・ペーパー 34

概要

計画メンテナンスや計画外停止、データベース層での負荷の不均衡に気づかれなければ、アプリケーションのサービスを継続できていると言えます。アプリケーションのベスト・プラクティスを取り入れ、簡単な構成変更を加え、MAAベスト・プラクティスを使用してデプロイしたOracle Databaseを使用することで、アプリケーションの可用性が確実に継続します。

アプリケーションとデータベースの準備に役立つチェックリストを以下に示します。これはアプリケーション・コンティニュイティを使用していない場合でも役に立ちます。このホワイト・ペーパーに記載されている内容は、継続的可用性に対応したシステム構成を構築する準備を行い、計画メンテナンスの作業中や計画外停止中に発生し得る停止時間を短縮するうえで大いに役立ちます。

アプリケーションの継続的可用性を維持するための機能セット

計画メンテナンスや計画外停止、データベース層での負荷の不均衡に気づかれなければ、アプリケーションの可用性を継続できていると言えます。オラクルは、計画的イベント中や、計画外停止や負荷の不均衡の発生時にもアプリケーションの可用性を維持できる一連の機能を提供し、個別に選択できるようにしています。これらの機能は、万が一のサービス中断からアプリケーションを保護する保険と考えることができます。もっとも優れている機能は、アプリケーションに対して完全に透過的であるためアプリケーション開発者をインフラストラクチャではなく機能の構築に専念させることができる機能、そして将来アプリケーションが変更されてもアプリケーションを保護し続ける機能です。これをオラクルでは未来対応と呼んでいます。

まずは、一連の機能について説明します。

計画メンテナンス時のセッションのドレイニングとリバランシング

計画メンテナンスを開始すると、インスタンス、PDB、データベースからドレインする必要があるセッションにドレイン対象のマークが付けられます。アイドル状態のセッションは徐々に解放されます。アクティブなセッションは、そのセッションで実行中の作業が完了した時点でドレインされます。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接続プールと中間層で広く使用されています。Oracle Database 18cからは、PDBやインスタンスが停止または再配置されると、データベースによるセッションの自己ドレインが行われます。気づかれずに計画メンテナンスを行うには、どのような場合でもドレイニングを行うのがもっともよい方法です。割り当てた時間内に作業のドレインが完了しそうにない場合は、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を実行します。

透過的アプリケーション・フェイルオーバー（TAF）

TAFはOracle8iから搭載されている機能です。インスタンス障害が発生すると新しいセッションが作成され、SELECTモードを使用している場合は、要求に応じて障害発生前の位置まで戻って問合せが再実行されます。Oracle Database 12.2以降、`FAILOVER_RESTORE`という、アプリケーション・コンティニュイティに相当する機能がTAFで提供されるようになり、問合せを再実行する前にセッションの初期状態をリストアできるようになりました。カーソルは、再確立された初期状態を使用して再実行されます。PLSQL、一時表、一時LOB、sys_contextといったセッション状態はリストアされないため、TAFを使用しているアプリケーションのセッションでは、後からセッション状態を変更してはいけません。

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、Javaベースのシン・アプリケーションの場合はOracle Database 12.1以降、OCIおよびODP.NETベースのアプリケーションの場合はOracle Database 12.2.0.1以降で使用できるようになった機能で、これを使用すると停止の発生に気づかれなくすることができます。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わ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きに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、プールされるOLTP型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 Database18c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からデータベース・セッションをリカバリできるようになりました。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するためには、アプリケーションの知識もアプリケーションのコード変更も必要ありません。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。

使用すべきソリューション

	TAC	AC	TAF	ドレイニング
アプリケーションの実装方法が分からない場合	はい	いいえ	いいえ	はい
トランザクションを実行するアプリケーションの場合	はい	はい	計画外の場合は「いいえ」 トランザクションの切断のみ	はい
Oracleの状態（一時LOB、PL/SQL、一時表）を使用するアプリケーションの場合	はい	はい いいえ （静的モードの場合）	いいえ	はい
接続プールを使用しないアプリケーションの場合	はい	いいえ	あり	はい
次的影響（ファイル転送など）があるアプリケーションの場合	はい 副次的影響は再実行されません	カスタマイズ可能	なし	はい
初状態をリストアする必要があるアプリケーションの場合	はい、かつカスタム	はい、かつカスタム	はい、かつカスタム	はい
アプリケーションの変更に備えた未来対応の場合	はい	いいえ	なし	はい

ドレイニングによる透過的アプリケーション・コンティニュイティは、継続的可用性を実現するための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や、初期状態や副次的影響に対してカスタマイズが必要な場合は、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は引き続き利用でき、完全にサポートされています。

アプリケーションの構成手順

ソリューションの実装手順を以下に示します。

1. Oracle Clusterwareによって管理される、デフォルト・データベース・サービス以外のサービスを使用します（デフォルト・サービスはデータベースまたはPDBと名前が同じです）。位置の透過性機能とHA機能を備えたサービスを作成します。
2. タイムアウト、再試行、遅延が組み込まれた推奨の接続文字列（このホワイト・ペーパーで後述します）を使用し、停止中に着信した接続要求がエラーにならないようにします。
3. 高速アプリケーション通知（FAN）は、サービスが再開したときや負荷の不均衡が発生したときのドレイニングの開始、障害からの離脱、セッションのリバランシングに必須の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では、クライアントがFANによって中断されない場合、アプリケーションの高速フェイルオーバーは実行されません。どのフェイルオーバー・ソリューションでもFANを使用します。FANを構成するときはONSの自動構成を使用します。TNS形式は推奨のものをそのまま使用し、変更はしないでください（ただし、クライアントが12cより前の場合は手動でFANを構成します）。

4. メンテナンスを開始する前に、メンテナンス対象のインスタンスやノードから作業をドレインします。Oracle接続プールまたは接続テスト（または両方）を使用してFANを有効にします。最適なソリューションはFANを使用するOracle接続プールです。これは、セッション移動のライフサイクル全体が提供されるためです。つまり、メンテナンスの進行に合わせて作業のドレイニングやリバランシングが行われるということです。FANを使用する場合はプールに接続を返します。サーバーによるドレイニング（この代替計画については後述します）を使用していて、テストが標準テストでない場合は、`DBMS_APP_CONT_ADMIN`を使用してサーバーにテストを追加します。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. セッションをフェイルオーバーする場合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、副次的影響またはコールバックを使用してカスタマイズする場合、または一時表などの状態を使用してクリーンアップを行わないアプリケーションを使用する場合、アプリケーション・コンティニュイティ（AC）を使用します。

   アプリケーションが読取り専用であり、最初のセットアップ後にセッション内でOracleセッションの状態が変更されない場合、透過的アプリケーション・フェイルオーバー（TAF）を使用します。

アプリケーションの継続的可用性を支える要素

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、MAAシステムの冗長部分の信頼性から得られるメリットをアプリケーションで享受できます。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、表面上は分かりません。

サービスは次のように定義できます。

- 管理の単位：少数のサービスは管理できますが、多数のノード、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、サイトおよびデータベースの位置の透過性を提供します。
- 可用性の単位：各サービスのリソースは独立かつ並行して迅速にリカバリされ、ソフトウェア・スタック全体を起動する必要はありません。
- パフォーマンスの単位：サービスの質と優先順位に従ってMAAシステム内で透過的に作業がルーティングされます。サービスはサービス・レベルしきい値を基準にして測定され、違反があれば、AWRで推奨されたソリューションと併せて経営陣にレポートされます。

サービスを使用する場合は、FAN、接続識別子、TAC、AC、スイッチオーバー、コンシューマ・グループなど多数の機能と操作が必要です。デフォルトのデータベース・サービスは無効化、再配置、制限などができず、したがって高可用性には対応しないため、使用しないでください。使用するサービスはData Guard環境にある特定のプライマリ・ロールまたはスタンバイ・ロールと関連付けられます。使用するアプリケーションに初期化パラメータ`service_names`を使用しないでください。

サービスの例：

ATP-Dからは、各PDBデータベースに5つのサービスがあらかじめ構成されており、この中から選択できるようになっています。どのサービスもFANとドレイニングに対応しています。

サービス名	説明	ドレイニング	FAN	TAC
TPURGENT	OLTP優先順位：最高	はい	はい	はい
TP	OLTP優先順位：普通 メイン・サービスとしての使用を推奨	はい	はい	はい
HIGH	レポートまたはバッチ 優先順位：最高	はい	はい	
MEDIUM	レポートまたはバッチ	はい	はい	

	優先順位：中			
低	レポートまたはバッチ 優先順位：最低	はい	はい	

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どれも、以下に示す勧告に準拠しています。オラクルが提供するウォレットを使用する場合は何もする必要がありません。フェイルオーバー、スイッチオーバー、フォールバック、基本的な起動時の接続には、次のTNS/URL構成を使用することを推奨します。

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。接続の試行および再試行はOracle Database Net Servicesによって管理されます。

CONNECT_TIMEOUTを上限に設定して、90sや120sなどのログイン・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ログインが大量に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には高可用性機能がないため、クライアントでは簡易接続ネーミングを使用しないでください。

なお、下で指定しているstandby-scanの参照先は（Active）Data Guard構成で指定したSTANDBYサイトで使用可能なSCANアドレスです。最初にプライマリ・サイトへの接続が試行され、サービスが使用可能でない場合はスタンバイのこのサービスへの接続が試行されます。サービスの場所がわかると、そのサービスの提供に使用されたaddress_listが12.2以降のOracleドライバに記憶され、次にサービスの場所が変わるまでこれが最初に選択されます。

*standby-scan*に透過的にフェイルオーバーされるようにする場合は、任意で*standby-scan*をTNS接続記述子に追加します。同じリージョン内にあるスタンバイ・データベースにフェイルオーバーする場合はパフォーマンスが許容範囲内になることがほとんどですが、異なるリージョンでネットワーク待機時間が発生する場所にあるスタンバイ・データベースにフェイルオーバーする場合は、応答時間のパフォーマンスが許容範囲を超えることになる場合もあります。後者のケースでは、サイト・フェイルオーバー操作が必要となり、中間層のリソースとスタンバイ・データベースが保持されている別のリージョンにDNSをフェイルオーバーさせることも必要になります。

Oracleドライバ・バージョン12.2以降の場合はすべて、次の接続文字列を使用します。

```
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接続には次のTNSを使用する必要があります。Thin Javaドライバに TRANSPORT_CONNECT_TIMEOUTを使用できるのはOracle Database 12.2以降の場合に限られるため、この接続文字列では低いCONNECT_TIMEOUTを使用しています。RETRY_DELAYを使用するには11.2.0.4クライアント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は必ず使用してください。アプリケーションのフェイルオーバーを中断するための必須コンポーネントです。FANを使用しなければ、ハードウェア障害やネットワーク障害が発生した場合にアプリケーションがTCP/IPタイムアウトでハングし、リソースが再開されたときにリバランスが行われない可能性があります。OracleプールおよびOracleアプリケーション・サーバーはすべて、FANを使用します。サード・パーティのJAVAアプリケーション・サーバーでは、UCPを使用してFANを有効にできます。FANを使用するためにアプリケーションの変更は不要です。構成の変更のみが必要となります。

計画メンテナンス中にサービスを継続させるには、FANを以下のいずれかと併用します。

- Oracleプール
- UCPとサード・パーティJDBCアプリケーション・サーバー
- 最新のOracleクライアント・ドライバ

計画外停止中にサービスを継続させるには、FANを以下のいずれかと併用します。

- 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

上述したADDRESS_LISTは形式が重要です。理由はいくつかありますが、1つは、この形式を使用するとONSの自動構成ができることです。ONSは、中間層のプールやクライアントにFANイベントを伝播するときに使用されます。データベース接続が確立すると、データベース層のONS情報が中間層に送信され、中間層でONSの通信経路を自動的に確立できるようになります。つまり、動的な構成が可能になり、中間層でのメンテナンスが不要になるということです。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のサポート対象

FANのイベントを統合できるものは次のとおりです。

- Oracle Fusion MiddlewareおよびOracle WebLogic Server
- Oracle Data Guard Broker
- JDBC ThinインタフェースおよびOCIインタフェース向けOracle JDBC Universal Connection PoolまたはOracle JDBCドライバ
- 非管理対象プロバイダおよび管理対象プロバイダ向けODP.NET接続プール
- Oracle Tuxedo
- SQL*Plus
- PHP
- Global Data Services
- Oracle JDBC Universal Connection Poolを使用するサード・パーティの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を有効にする手順：

前述した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は、FANイベントを受信するためのOracle Notification Service（ONS）サブスクリプションが、この接続文字列によりクライアントで自動構成されます古いドライバを使用する場合、構成の詳細については、「その他のホワイト・ペーパー」に掲載されているFANのホワイト・ペーパーを参照してください。ONSによってデータベース層とクライアント層間にセキュアな通信パスが提供され、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をクライアントに通知できるようになるため、通常運用時に作業の配置が改善されます。

クライアントでFANを有効にする手順：

1. 指定されたTNSエイリアスまたはURLを使用します。12c以降のドライバを使用している場合は、FANイベントを受信するためのONSサブスクリプションが、この接続文字列によりクライアントで自動構成されます（AUTO-ONS）。古いドライバを使用する場合は、FANのホワイト・ペーパーを参照してください。

2. クライアントに応じて、次のようにアプリケーション構成プロパティでFANを有効にします。Universal Connection PoolまたはJDBC Thinドライバ（12.2以降）

プロパティ`FastConnectionFailoverEnabled`を設定します。

WebLogic Active GridLink for Oracle RAC

FANと高速接続フェイルオーバーはデフォルトで有効化されます。

Oracle WebLogic Server、IBM WebSphere、IBM Liberty、Apache Tomcat、Red Hat WildFly（JBoss）、JDBCアプリケーション

接続プールの代わりに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および管理対象外プロバイダ）

ODP.Net 12.1以前を使用する場合、接続文字列に“`HA events = true;pooling=true`”を設定します。

Oracle Call Interface（OCI）クライアントおよびOCIベースのドライバ

ネイティブ設定を使用しないOracle Call Interface（OCI）クライアントでは、oraacces.xmlを使用し、events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および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`を編集してエントリ`oci8.events=on`を追加します。

SQL*PlusではFANがデフォルトで有効になります。

3. TLS（ウォレットベースの）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にJARファイル`ons.jar`、`osdt_cert.jar`、`osdt_core.jar`、`oraclepki.jar`が存在することを確認します。Oracle ATP-Dサービスでは常にTLSが使用されます。

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) 次のいずれかを実行してFAN用のウォレットを指定します。

- ウォレットでAUTO-ONSを使用する場合は、次のJavaシステム・プロパティをアプリケーションで設定する必要があります。

"-Doracle.ons.walletfile=/replace_this_with_host_path/onswallet" and
"-Doracle.ons.walletpassword=myONSwalletPassword"

これは、プール単位または接続単位では設定できません。

- （AUTO-ONSではなく）明示的なONS構成を使用する場合は、次を実行します。

i) プログラムを使用してUCP内で`setONSConfiguration()`を呼び出します。たとえば、次のようにします。

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

ii) UCP XML構成ファイルを使用して宣言的に設定します。

ii) Oracle Call Interface（OCI）アプリケーションの場合は、Oracleクライアント・ドライバ12.2以降が必要です。

a) oraaccess.xmlファイルの<default_parameters>セクションに以下を追加します。

```
<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>パスには、ウォレットが含まれているディレクトリの名前を指定する必要があります。

<hosts>、<max_connections>、<subscription_wait_timeout>などのその他のパラメータは、oraaccess.xmlのonsセクションで設定できます。

ネイティブ・イベントの設定制御をサポートしているドライバでは、<events>セクションを省略して、代わりにドライバ設定を使用できます。

デフォルトでは、ONSが失敗した場合で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を失敗させる必要がある場合は、<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>
```

ネットワーク・ファイルtnsnames.oraおよびsqlnet.ora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、これらのファイルをデフォルト・ディレクトリnetwork/adminに配置できます。または、アクセス可能な別のディレクトリにすべてのネットワーク構成ファイルを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。

この時点で、計画メンテナンスに対応したアプリケーションを構成できます。

アプリケーションに影響を与えないメンテナンス

メンテナンス前のセッションのドレイン

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・パーティのFANによって開始されたリクエストをドレイニングすることで実行できます。Oracle Databaseでも作業のドレインが直接行われます。

Oracle Databaseに接続するサービスは、接続テストを行い、ドレイニングに使用できる時間をdrain_timeoutに指定し、ドレイン・タイムアウトの終了後に適用されるstopoptionにIMMEDIATEを指定して構成します。停止、再配置、スイッチオーバーの各操作にはdrain_timeoutとstopoptionが含まれており、設定済みの値を必要に応じてオーバーライドできます。

計画メンテナンスを開始すると、FANにサブスクライブしているすべてのアプリケーションにplanned-downイベントが送信されます。FANに対応したプールでこのFANイベントが受信されると、プールが応答し、アイドル状態のセッションはクリアされます。また、アクティブなセッションは、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解放する対象としてマークされます。このFANイベントにより、drain_timeoutで指定した期間は、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できます。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からは、ドレインするセッションのマーキングはデータベースで自動的に行われます。マーキングが完了すると、アプリケーションを妨害せず安全にセッションをドレインできる場所を探すためのルールが適用されます。drain_timeoutの期間が終わるまでドレイニングは続行されます。

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、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ンが完了しなかったセッションについては、選択したフェイルオーバー・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。

計画ドレイニングの構成

片方または両方のアプローチを使用すると、Oracle Databaseに接続しているアプリケーションには計画メンテナンスが認識されません。計画メンテナンスの前に、データベース・インスタンスでデータベース・セッションのドレインまたはフェイルオーバーを行い、アプリケーションの作業が中断されないようにします。

推奨アプローチ

計画メンテナンスを認識させないためのソリューションとしては、FANに対応したOracle接続プールを使用する方法を推奨します。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。OracleプールとFANをアプリケーションで使用している場合はユーザーへの影響がなく、リクエストとリクエストの間はプールに接続が戻されます。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プールでFANイベントが受信されると、Oracleプールにより、インスタンスのすべての接続にドレイン対象のマークが付けられます。チェックイン済みの接続は、再利用されないよう即座にクローズされます。使用中の接続は、次に接続プールにチェックインされたときにクローズする対象としてマークが付けられたままになります。これらがプールに返されるに連れて、接続が徐々に解放されていきます。

アプリケーションの用法としては、必要なときに接続をチェックアウトし、現在のアクション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に重要ですが、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も重要です。

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。これらのアプリケーション・サーバーでUCPを使用する方法については、オラクルや他のベンダー（IBMなど）が発行しているホワイト・ペーパーを参照してください。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UCPの機能を使用できます。oracle.comのテクノロジーのページでは、顧客事例を多数紹介し、“アプリケーション・コンティニュイティ”や“JDBC”について説明しています。

代替アプローチ – Oracle Database 19cのドレイニングまたは Oracle Driver 19cのドレイニング

OracleプールとFANの併用ができない場合でも、Oracleデータベースで自動的にセッションのドレインが行われます。接続を解放できる安全な場所はデータベースで自動的に検索されます。データベース・セッションを解放するタイミングは、さまざまなルールと経験則に基づいてデータベースにより検出されます。
ドレイニングが始まっても、ルールに適合しない限りデータベース・セッションはデータベースからドレイニングされません。ルールには次のものが含まれます。

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

接続テストとしてアプリケーション・サーバー、プールされるアプリケーション、ジョブ・スケジューラなどで通常行われるのは、接続プールからの借用時、プールへの返却時、バッチ・コミット時の接続テストです。ドレインの期間中は、データベースが接続テストを妨害し、接続をクローズし、テストが失敗したことを示すステータスを返します。接続テストの発行元であるアプリケーション・レイヤーでは、失敗のステータスが返されると即座に反応し、さらにリクエストを発行して別の接続を取得します。つまり、アプリケーションは妨害されません。

Oracle Driver 19cのドレイニングと帯域内FAN

Oracle Database 18cおよびOracle Database Drivers 18c以降、計画停止のFAN帯域内イベントをドライバが受信するとドレイニングが有効化されるようになりました。ドライバがリクエストの終了ステータスと接続テストを探して接続を安全にクローズします。ドライバは、SQLを使用しない、PINGベースの接続テスト（`connection.isValid()`、`CheckConStatus`、`OCI_ATTR_SERVER_STATUS`など）を使用します。リクエストの境界を使用する場合、Oracle Database 12c以降のプール、またはJDK9以降を使用しているJavaプールが必要です。ベスト・プラクティスに沿ったアプリケーションを作成し、リクエストが完了したらプールに接続を返却する必要があります。

接続テスト

Pingと終了リクエスト：

高速接続フェイルオーバー（FCF）が処理される前に接続が借用されると、障害の種類によっては、古くなった接続をアプリケーションがレース期間中に受信することがあります。こうした事態は、着信接続要求と同時にソケットがクローズされるクリーン・インスタンス停止などで発生する可能性があります。アプリケーションがエラーを受信しないようにするには、接続プールで接続チェックを有効にする必要があります。

JDBC Universal Connection Pool

> `setValidateConnectionOnBorrow()` – 接続プールから接続を借用するときに接続の検証を行うかどうかを指定します。プールから借用されるすべての接続の検証がこのメソッドにより有効化されます。デフォルト値はfalseです。検証を実行する場合は値をtrueに設定します。

JDBC Driver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、すべてのJavaベースの接続プールから接続を借用する場合、ジョブ・スケジューラを使用する場合、または同様に、データベースへの接続リクエストを送信する前に使用するように、デフォルトで有効にすることを推奨します。

OCI接続

> サーバーへの接続がFANとOCI_ERRORのいずれで終了されたのかを確認するには、サーバー・ハンドルの属性OCI_ATTR_SERVER_STATUSの値をチェックするコードをアプリケーションに組み込む必要があります。

サービスが停止している場合は、FANイベントへのレスポンスとしてOCIレイヤーでOCI_ATTR_SERVER_STATUS属性がOCI_SERVER_NOT_CONNECTEDに設定されます。スケジュール・メンテナンスの場合は、作業を完了するための猶予期間を確保するために、接続を削除せずにこのステータスが設定されます。

FANを使用して計画メンテナンスのレポートを行うと、借用前とプールへの返却後にOCI_ATTR_SERVER_STATUSがアプリケーションでチェックされ、安全な場所でのみセッションがドロップされます。クローズされた接続を借用前と返却後にドロップすると、計画メンテナンス中にアプリケーション・エラーを受信することがなくなり、ユーザー・エクスペリエンスが良好になります。FANを使用して計画外停止のレポートを行うと、即座にエラーがアプリケーションで受信されます。

コーディングが必要な接続テストはこれだけです。他の接続テストはすべて構成が可能です。

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、接続がODP.NET接続プールに返される前の接続ステータス・チェックを有効にします。

SQL接続テスト

どのアプリケーション・サーバーにも、それぞれの接続プールにある接続の妥当性をテストする機能が付属していて、プロパティを構成するか管理コンソールを使用して設定するようになっています。このテストの目的は、使用不能な接続をアプリケーションで借用しないようにするとともに、使用不能の接続が検出された場合はプールに解放するときに削除することです。

さまざまなアプリケーション・サーバーがありますが、テストの名前はどれも類似しています。付属しているテストではさまざまなアプローチが使用されていますが、もっとも多いのはSQL文です。オラクルが推奨するのは、標準のJavaコール`connection.isValid()`を使用するJavaアプリケーション・サーバーです。Oracle Database 18c以降、データベースのドレインにすべてのテストが使用されるようになりました。また、Oracle Database 18c以降は、セッションの安全なドレイニング・ポイントを調査することで、FANを使用せずにセッションがドレインされるようになりました。サービスが停止するか再配置されると調査が自動的に開始されます。サーバー・ドレイニングはすべてのドライバに対応しています。

SQL接続テストは4つあり、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・サービスにデフォルトで追加されるため、アプリケーションでこれらのSQLテストを使用していれば、それらはすでにカバーされています。

```
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に対応したその他の接続テストの構成

サービス、プラガブル・データベース、非コンテナ・データベースへ接続テストを追加、削除、有効化、無効化できます。追加したテストや有効化したテストは、ビューDBA_CONNECTION_TESTSで確認できます。次に例を示します。

デフォルトでは、ドライバではpingテストが使用されます。ドライバに対して有効にする必要はありません。ping（isValid、isUsable、OCIping、connection.statusなど）を使用するテストをデータベースで使用する場合は、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・データベースのサーバー側SQL接続テストを新しく追加する場合は、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]);
```

次はフェイルオーバー・ソリューションを選択します

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（TAC）はOracle Database18cで追加されたアプリケーション・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てリカバリ可能な停止からデータベース・セッションをリカバリできるようにする機能です。これは安全な処理で、DBAがアプリケーションの知識を持っている必要もなければ、アプリケーションのコードを変更する必要もありません。処理を透過的に行うために、状態を追跡するインフラストラクチャを使用して、アプリケーションからデータベースに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します。状態追跡を使用することで、アプリケーションや環境が変化しても、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は未来対応になります。

透過的アプリケーション・コンティニュイティのサポート対象

Oracle Autonomous Databaseの透過的アプリケーション・コンティニュイティは、次のクライアントに対応しています。

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UCPを使用しているサード・パーティのJDBCアプリケーション・サーバー
- Oracle JDBC - Replay Driver 12c以降とリクエスト境界を使う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
- OCI Session Pool 19c以降
- SQL*Plus 19c以降
- ODP.NET Unmanaged Provider 18c以降
- 19c以降のOCIドライバを使用した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。これらのアプリケーション・サーバーでUCPを使用する方法については、オラクルや他のベンダー（IBMなど）が発行しているホワイト・ペーパーを参照してください。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 サポート対象のクライアントの使用（前述した対象範囲を参照）
接続プールへの接続の返却
アプリケーションで以下のいずれかの接続を使用している場合は、アプリケーションを変更しなくてもリクエストの境界を識別できます。

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

説明どおりに接続プールを機能させるためには、アプリケーションで必要になったときに接続を取得し、使用していない接続は解放する必要があります。これにより、接続を維持する場合と比較して、スケーラビリティが向上し、メモリ使用が低減します。また、リクエストの境界が透過的になります。

アプリケーションで必要になったときだけ接続をチェックアウトするのがベスト・プラクティスです。使用していない接続を保持しているのは適切ではありません。

JDBC-Thinドライバ・リリース18c、および19c以降のOCIベースのアプリケーションとTACを併用している場合は、リクエストの境界の検出と拡張が透過的に行われます。つまり、リクエストの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影響を与えない文は不要になると削除されるため、リソース使用率が低下し、リカバリが高速化されるということです。それでも、Oracleプールを使用することと、リクエストとリクエストの間は接続をOracleプールに返すことがベスト・プラクティスであることに変わりありません。

FAILOVER_RESTOREの使用

TACでは自社のサービスの`FAILOVER_RESTORE`を`AUTO`に設定すると、事前設定した状態が自動的にリストアされます。

標準設定の他に事前設定のセッション状態が必要になった場合は、コールバック（UCPラベル）を登録してその状態をリストアすることができます。一時表やsys_contextといった複雑なセッション状態をリストアすることが必要になった場合は、コールバックとアプリケーション・コンティニュイティを併用すると、この柔軟性が得られます。

アプリケーションでの可変値の使用の有効化

可変関数とは、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。`SYSDATE`、`SYSTIMESTAMP`、`SYS_GUID`、`sequence.NEXTVAL`については、可変関数の元の結果を保持できるようになっています。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 19cではSQLの可変値が自動的に保持されます。

PL/SQLの可変値が必要な場合は可変オブジェクトを構成しますが、このときアプリケーション・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。

例：

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

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

副次的影響の無効化

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により検出された副次的影響は再実行されません。副次的影響とは、メールの送信、ファイルの転送、TCPの使用といった外部アクションです。副次的影響の種類は、アプリケーションのロジックに関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とに区別されます。副次的影響を伴う文を使用するアプリケーションの場合は、その文が実行されているときは取得が無効化されます。キャプチャは、次の検出された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。TACによるJava 18c Replay DriverおよびOCI 19cドライバでは、暗黙的なリクエスト境界がサポートされるため、可能な場合はキャプチャが再び自動的に有効になります。副次的影響の再実行が必要な場合は、このような柔軟性のあるアプリケーション・コンティニュイティを使用します。

制限事項

制限事項の最新の一覧については、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

アプリケーション・コンティニュイティはカスタマイズが可能で、メールなどの副次的影響の再実行を選択したり、TACでは実行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できます。12.2の場合は、副次的影響またはコールバックを使用してカスタマイズするか、セッション期間中の一時表などの状態を使用しクリーンアップを行わないアプリケーションを使用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のサポート対象

Oracle Database 18c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UCPを使用しているサード・パーティのJDBCアプリケーション・サーバー。
- Oracle JDBC - Replay Driver 12c以降とリクエスト境界を使う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 Call Interface Session Pool 12cリリース2以降を使用するアプリケーションおよび言語ドライバ
- SQL*Plus 19c以降
- ODP.NET Unmanaged Provider 12cリリース2以降（12.2以降では“Pooling=true”、“Application Continuity=true”がデフォルト）

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、もっとも効果的に高可用性を実現するには、データソースの代わりに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。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

サポート対象のクライアントの使用（前述したサポート対象を参照）
接続プールへの接続の返却

アプリケーションで使用する接続は、リクエストのたびに接続プールに返却する必要があります。アプリケーションで必要になったときだけ接続をチェックアウトするのがベスト・プラクティスです。使用していない接続を保持しているのは適切ではありません。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、処理が完了したらすぐにその接続をチェックインします。そうすれば、後続の他のスレッドや、自分のスレッドで再び必要になったときに接続を使用できます。このようにすると明示的なリクエスト境界も埋め込まれ、キャプチャの再開と終了を安全に実行できる場所をアプリケーション・コンティニュイティで識別できるようになります。ACの場合はTACとは異なり、リクエストの境界が追加で検出されることはありません。

FAILOVER_RESTOREの使用

アプリケーションや中間層では、すべての接続がたとえば、事前設定した言語やタイムゾーンになるように接続プールを構成する場合があります。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に設定すると自動的にリストアされます。標準設定の他にセッション状態を事前設定する場合は、コールバック（UCPラベル）を登録してその状態をリストアすることが必要になります。下の「その他のホワイト・ペーパー」に掲載されているアプリケーション・コンティニュイティのホワイト・ペーパーを参照してください。

他のセッション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ます。

- 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可変値の使用の有効化

TACの項の可変値に関する説明を参照してください。

再実行時に副次的影響を再現する必要性の有無の判断

再実行を無効にしない限り、副次的影響は再実行されます。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、副次的影響のあるリクエストを再実行する意味があるかどうか検討して判断する必要があります。たとえば、電子メールの再送信やファイルの再送信が必要かどうかを確認します。ほとんどの場合、副次的影響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ことができます。その他のリクエストはすべて引き続き再実行されます。すべての副次的影響を再実行する必要がある場合は、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、およびODP.NETのクライアント側の機能で、接続の初期セットアップ後にOracleの状態を変更しないアプリケーションの接続またはSELECT文をフェイルオーバーします。フェイルオーバーする必要がある接続にはFAILOVER_TYPEをBASICに設定し、接続し直して再実行するSELECT文にはSELECTを設定します。実際の状況ではアプリケーションが事前接続されないことからフェイルオーバーの遅延の原因になるため、TAFの事前接続というオプションは推奨されません。

Oracle Databaseの透過的アプリケーション・フェイルオーバーでは、以下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（OCCI）
- Oracle JDBC OCIドライバ（通常、シック・ドライバは推奨されません）
- ODP.Net Unmanaged Provider
- OCIを使用する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

サポート対象のクライアントの使用（前述したサポート対象を参照）

FAILOVER_RESTOREの使用

12.2以降のOracle Databaseでは、接続時にアプリケーションによる値の事前設定が行われているかどうかをチェックします。アプリケーションや中間層では、すべての接続がたとえば、事前設定した言語やタイムゾーンになるように接続プールを構成する場合があります。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に設定すると自動的にリストアされます。標準設定の他にセッション状態を事前設定する場合は、TAFコールバックを登録してその状態をリストアすることが必要になります。

次のオプションを併せて使用します。

FAILOVER_RESTORE=LEVEL1（サービスで設定）

OCIのTAFコールバック

まだコールバックを登録していない場合、RDBMS 12.2以降ではFAILOVER_RESTORE=LEVEL1にすることを推奨します。

トランザクション・ガードを使用したTAF

Oracle Database 12.2以降では、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）が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、あるい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代わりに再送信します。

下に示すトランザクション・ガードのホワイト・ペーパーを参照してください。

TACまたはACを使用する場合の保護レベルの把握

保護レベルの統計情報

リクエストの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、より新しい19cバージョンではV$SERVICE_STATSで確認できます。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。

統計情報	合計	1秒あたり	トランザクションあたり
cumulative begin requests	1,500,000	14,192.9	2.4
cumulative end requests	1,500,000	14,192.9	2.4
cumulative user calls in request	6,672,566	63,135.2	10.8
cumulative user calls protected	6,672,566	63,135.2	10.8

ヒント：保護レベルをPDBによって、または履歴データを使用してレポートするには、付録のSQL使用例を参照してください。出力は次のようになっています。
以下の場合に、TACまたはACが有効になってアプリケーションを保護します。

- Cumulative user calls in requestの値とcumulative user calls protectedの値が等しい
- かつ、上記の値がゼロでない

この保護レベルはデータベース内部で測定されます。このレベルの保護を達成するには、クライアントで問合せにORDER BY句を使用し、FAILOVER_RESTOREの対象ではない状態が含まれている場合は、セッションの初期状態を事前設定することが必要になる場合があります。

上記の例は、開始リクエストと終了リクエストの数が増加していることを示しています。数自体は、アプリケーションが接続プールに対してチェックアウトおよびチェックインを実行する頻度や、TACの使用時にデータベースで検出可能なリクエスト境界によって異なります。これらの値が増加する速度は、リクエストが送信される速度によって異なります。以下を使用すると、保護対象のユーザー・コールの割合を計算できます。

保護対象のコールの割合 = cumulative user calls protected / cumulative user calls in request × 100

保護対象のコールの割合が100%未満になる可能性があります。JDBC具象クラスを使用している、副次的影響が無効化されている、リカバリ不能な状態が使用されている、または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化するようにアプリケーションで選択されている場合があります。アプリケーションの保護が100 %でない場合は、サイトでORAchkコンポーネントのacchkを使用し、アプリケーション内のどの箇所で保護が100 %未満になっているのかを把握できます。影響を評価してから、アドバイザに従うのか措置を何も実施しないのかをマネジメントが判断できます。

クライアントの構成

JDBC Thinドライバの場合のチェックリスト

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. 保護および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。

最大の保護とパフォーマンスを達成するには、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、リクエストの終了時にステートメントがクローズされてメモリが解放されることを認識できます。

JDBCの文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。以下に例を示します。

`oracle.jdbc.implicitStatementCacheSize=`*`nnn`* ここで、*nnn*は通常、10〜100の値であり、アプリケーションで維持されているオープン・カーソルの数と同じになります。

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、2G以上といった高い値を使用できます。以下に例を示します。

```
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。これにより、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ません。

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UCPまたは再実行ドライバが埋め込まれている場合、この作業は重要です。

UCP `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、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 *『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 Thinドライバ・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に推奨されている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・パーティのJDBCプールの場合は、UCPを推奨します。
- 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）。

クライアント・ドライバが12cより以前の場合は、以下のアドレスを使用します。

oracle.ons.nodesをXXX01:6200、XXX02:6200、XXX03:6200となるように設定します。

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

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およびACではOCIStmtPrepareを使用できますが、このステートメントは再実行されません。

   https://docs.oracle.com/en/database/oracle/oracle-database/19/lnoci/deprecated-oci-functions.html#GUID-FD74B639-8B97-4A5A-BC3E-269CE59345CA

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でサービスにaq_ha_notificationsを事前設定します。
- auto-onsに推奨されている接続文字列を使用します。
- `oraaccess.xml`で`auto_config`、`events`、および`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）。

クライアント・ドライバが12cより前のものである場合は、`oraccess.xml`で指定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・ドライバの場合のチェックリスト

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でサービスに`aq_ha_notifications`を事前設定します。
- auto-onsに推奨されている接続文字列を使用します。
- `oraaccess.xml`で`onsConfig`および`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）。
- 次の接続文字列で、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、ラインタイム・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; loadbalancing=true;"
```

結論

Oracle Database 19cは、高可用性を実現するようにユーザーに代わって構成および管理されています。ユーザーが追加の構成や管理を行う必要はありません。

以下の簡単な手順を実行することで、アプリケーションで継続的可用性を達成できます。

- それぞれのSLAに適したデータベース・サービスを選択する
- 高速アプリケーション通知（FAN）を構成する
- アプリケーションに推奨されている接続文字列を使用する
- ドレイニングを最適化するアプリケーション・ベスト・プラクティスを使用する
- 継続的なサービスを達成するために、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する

上記の5つの簡単な手順を実行することで、ほとんどの場合、計画メンテナンス作業時に停止が不要になり、計画外のイベントが発生した場合もトランザクションの失敗やユーザー・サービスの中断がなくなります。

付録：サービスの構成

Oracle Databaseを使用する場合は、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を使用するサービスをOracle RACに作成できます。Active Data Guardとプライマリ・データベースのどちらでサービスがアクティブなのかは、ロールを使用して区別できます。以下の例では、このことを示すために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、アクティブ・インスタンスが1つ以上のサービスが必要な場合（この場合、*使用可能な*インスタンスがありません）

```
$ 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
```

アプリケーション・コンティニュイティ

```
$ srvctl add service -db mydb -service SILVER -preferred inst1 -available
inst2 -failover_restore LEVEL1 -failoverretry 1 -failoverdelay 3 -
commit_outcome TRUE -failovertype TRANSACTION -replay_init_time 1800 -
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
```

Data Guardロールを使用して追加する場合のTACの例：

```
$ 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接続文字列またはURL（このホワイト・ペーパーで前述しています）が使用される場合、再試行の試みは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

多くの企業が大量のサービスを実行していますが、単一のデータベースまたはインスタンスで多数のサービスを提供していることもあれば、少しのサービスを提供するデータベースが同じノードで多数稼働していることもあります。Oracle Database 12c Release 2以降では、個々のサービスについて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のノード名、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よくなりました。

DRAIN_TIMEOUTとSTOPOPTIONは両方とも、サービスを追加するときまたは作成した後でサービスを変更するときに定義できるサービス属性です。これらの属性はSRVCTLを使用して指定することもでき、その場合はサービスに定義されている属性より優先されます。サービスのグループのもっとも大きいDRAIN_TIMEOUTがグループ操作に適用されます。次の例を参照してください。

RAC上のデータベース、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つのノードであらゆるものを開始/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：ACCHKを使用した、具象クラスのチェック

このチェックは、Javaアプリケーションのみに適用されます。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（推奨されない具象クラスをアプリケーションで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、Oracle ORAchkのアプリケーション・コンティニュイティ・チェック（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、形式がARRAY、BFILE、BLOB、CLOB、NCLOB、OPAQUE、REF、またはSTRUCTの推奨されないoracle.sql具象クラスを変数タイプ、キャスト、メソッドの戻り型、またはコンストラクタの呼出しとして使用する、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、形式がOPAQUE、REF、またはSTRUCTの推奨されないoracle.sql具象クラスを変数タイプ、キャスト、メソッドの戻り型、またはコンストラクタの呼出しとして使用する、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によってアプリケーションを保護するために、推奨されないJavaを削除する必要があります。

Oracle具象クラスに対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。これらの値を、コマンドライン、シェル環境変数、またはこの両方を使用して設定します。以下に値を示します。

表2-1 具象クラス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

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-XXX.jarのバージョン（ASMホームページからダウンロード）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	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。プログラムによって.classファイルが分析され、.jarファイルとディレクトリが再帰的に分析されます。
-jdbcver	RAT_AC_JDBCVER	カバレッジ・チェックの対象バージョン

アプリケーション・コンティニュイティによる具象クラスのチェック例のまとめ

以下のコマンドで、Oracle具象クラスに対するアプリケーション・コンティニュイティ・チェックを確認します。

```
$ ./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
```

出力例：アプリケーション・コンティニュイティのまとめ

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 :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：ACCHKを使用したアプリケーション・コンティニュイティのカバレッジ

破壊的テストは有効なテストです。ただし、障害が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チェック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することができます。カバレッジ・チェックは、デプロイの前やアプリケーションの変更後に使用します。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・リリースをどのように保護するかを把握します。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レポートします。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がOracle 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を使用してリリースする必要はありません。チェックを行うと、リリース前に役立ちます。

以下は、カバレッジ分析のサマリーです。

- 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か、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コールとしてカウントされます。
- キャプチャや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次々と実行します。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。

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・レポート例

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

コマンドライン引数	シェル環境変数	使用方法
-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。
-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、以下になります。 $ORACLE_BASE/diag/rdbms/{DB_UNIQUE_NAME}/$ORACLE_SID/trace

カバレッジ・レポートの確認

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）になっていることを確認するには、reportsディレクトリのPASSレポートacchk_scorecard_pass.htmlを確認します。

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を使って設定可能）が含まれます。

出力例：FOUND IN ORACHK_ .....HTML#ACCHK_SCORECARD

カバレッジ・チェック	FAIL	**TotalRequest = 1088** **PASS = 1082** **WARNING = 1** **FAIL = 5** 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：SQLを使用した、PDB、サービス、および履歴による保護のレポート

PDBによる保護のレポートを作成するには、以下の例を使用します。

```
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(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;
```

サービスによる保護のレポートを作成するには、以下の例を使用します。

```
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レポートを作成するには、以下の例を使用します。

```
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' a
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;
```

その他のホワイト・ペーパー

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）

『アプリケーション・コンティニュイティの確保』（*https://docs.oracle.com/cd/E82638_01/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/18/adfns/high-availability.html#GUID-96599425-9BDA-483C-9BA2-4A4D13013A37

トランザクション・ガード

トランザクション・ガード（https://www.oracle.com/technetwork/jp/database/database-cloud/private/transaction-guard-wp-12c-1966209-ja.pdf）

Graceful Application Switchover in RAC with No Application Interruption

My Oracle Support（MOS）Note：Doc ID 1593712.1

ORACLE CORPORATION

Worldwide Headquarters
500 Oracle Parkway, Redwood Shores, CA 94065 USA

海外からのお問い合わせ窓口
電話　+ 1.650.506.7000+ 1.800.ORACLE1
FAX　+ 1.650.506.7200
oracle.com

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

Integrated Cloud Applications & Platform Services

Copyright © 2020, Oracle and/or its affiliates.All rights reserved.本文書は情報提供のみを目的として提供されており、ここに記載されている内容は予告なく変更されることがあります。本文書は、その内容に誤りがないことを保証するものではなく、また、口頭による明示的保証や法律による黙示的保証を含め、商品性ないし特定目的適合性に関する黙示的保証および条件などのいかなる保証および条件も提供するものではありません。オラクルは本文書に関するいかなる法的責任も明確に否認し、本文書によって直接的または間接的に確立される契約義務はないものとします。本文書はオラクルの書面による許可を前もって得ることなく、いかなる目的のためにも、電子または印刷を含むいかなる形式や手段によっても再作成または送信することはできません。

OracleおよびJavaはOracleおよびその子会社、関連会社の登録商標です。その他の名称はそれぞれの会社の商標です。

IntelおよびIntel XeonはIntel Corporationの商標または登録商標です。すべてのSPARC商標はライセンスに基づいて使用されるSPARC International, Inc.の商標または登録商標です。AMD、Opteron、AMDロゴおよびAMD Opteronロゴは、Advanced Micro Devicesの商標または登録商標です。UNIXは、The Open Groupの登録商標です。0820

2020年8月
著者：Troy Anthony、Carol Colrain 共著者：
Burt Clouse、Ian Cookson

Oracle is committed to developing practices and products that help protect the environment