ORACLE
Autonomous
Database Cloud

Oracle Database 19c によるJavaプログラミング

オンプレミス、Oracle Cloud、データタイプ、
セキュリティ、パフォーマンス/スケーラビリティ、
停止時間ゼロ

ホワイト・ペーパー/2020年1月9日

ORACLE

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。本書に記載されている機能の開発、リリース、および時期については、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により、コードが大幅に不安定化するリスクなしに、本書に記載されているすべての機能を安全に含めることができない場合があります。

はじめに

Oracle Database Release 19cによりJavaの開発者とアーキテクトは、Oracle JDBCドライバ、Oracle Universal Connection Pool（Oracle UCP）、データベース組込みJVM（別名、Oracle JVM）[1]を使用して、最新で、保護された、レジリエンスがある高速アプリケーションを設計およびデプロイすることができます。

このホワイト・ペーパーでは、Oracle JDBCの標準、オンプレミスまたはOracle Cloudにおけるデータベースへの接続、新しくサポートするデータタイプ、セキュリティの拡張機能、パフォーマンスとスケーラビリティ、停止時間ゼロの領域における最新の改良された点とAPIについて説明します。

Oracle JDBCの標準

Oracle JDBCでは、標準のJDBC 4.3の改良された点に対応しています。

オンプレミスおよびOracle CloudでのDB 19cへのJava接続

このセクションでは、データベース・エイリアス、データベース・サービス、簡易接続ネーミングの仕組み、Oracle Cloudでのデータベースへの接続について説明します。

ドライバjarファイル

このリリースで核となるOracle JDBCドライバは次のとおりです。

- Java SE 8でコンパイルされたojdbc8.jar（JDBC 4.2）、Java 11で使用可能
- Java SE 10でコンパイルされたojdbc10.jar（JDBC 4.3）、Java 11で使用可能

Metalink Doc ID 2482279.1では、Oracle JDBCのリリースの方向性が述べられています。
必要とされるすべてのjarファイルの一覧は、Oracle MavenとOTNのダウンロード・ページで確認できます。

データベースのエイリアスとサービス

Oracle Databaseに接続する場合、Javaプログラムでは、Oracle JDBC接続文字列に含まれるOracle Net Namingエイリアス（jdbc:oracle:thin:@dbaliasなど）を使用します。
Oracle Net Servicesのエイリアスは、プロトコル、ホスト、ポート、サービス名を含む完全記述に拡張されます。tnsnames.oraとして知られる構成ファイル、またはLDAPとディレクトリ・ネーミング・サービス・リポジトリldap.ora（大規模デプロイメントの場合）を使用します。Java開発者またはデータベース管理者が、構成ファイルまたはディレクトリ・ネーミング・サービス・リポジトリでのエイリアスと完全記述のマッピングを定義します。
以下に、tnsnames.oraのエントリ例を示します

```
dbalias =
        (DESCRIPTION=
            (ADDRESS=
                (PROTOCOL=tcp)
                (HOST=sales-server)
                (PORT=1521))
```

1 詳しくは、Oracle JVMランディング・ページ（https://www.oracle.com/jp/database/technologies/appdev/ojvm.html）を参照してください（Githubコードのサンプルを含む）。

```
        (CONNECT_DATA= (SERVICE_NAME=dbservice)))
```

Oracle JDBCはロングURL形式に対応しています。この形式では、tnsnames.oraエントリの完全記述が接続文字列内で直接指定されるため、構成ファイルやディレクトリ・ネーミング・サービスを使用する必要がありません。

```
jdbc:oracle:thin:@ DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=sales- server)
(PORT=1521))(CONNECT_DATA= (SERVICE_NAME=dbservice)))
```

データベース・サービス名は、データソースのJNDI名に類似しており、データベース・サービス名によってデータベースが仮想化されます。関連付けられたデータベースは変更されることがあり、たとえば、Oracle RAC環境でサービスがインスタンスから別のインスタンスに再配置されたり、コード変更なしでオンプレミスからOracle Cloudに移動されたりします。
次の記述は、クラスタ化データベース（つまり、Oracle RAC）環境とディザスタ・リカバリ（つまり、Oracle Active Data Guard）環境に対応する推奨データベース接続文字列です。

```
dbalias =
        (DESCRIPTION =
            (CONNECT_TIMEOUT=120)(RETRY_COUNT=20)(RETRY_DELAY=3)
            (TRANSPORT_CONNECT_TIMEOUT=3)
            (ADDRESS_LIST =
                (LOAD_BALANCE=on)
                (ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
            (ADDRESS_LIST =
                (LOAD_BALANCE=on)
                (ADDRESS = (PROTOCOL = TCP)(HOST=standby-scan )(PORT=1521)))
            (CONNECT_DATA=(SERVICE_NAME = service name)))
```

上記の指定には、接続の再試行、同じデータベースにアクセスするすべてのインスタンスでのワークロードのロードバランシング（Oracle RACのテクノロジー）、ディザスタ・リカバリ（Oracle Active Data Guardのテクノロジー）の場合における別のデータセンターへのフェイルオーバーに関する規定が含まれています。

構成ファイルの場所

tnsnames.ora構成ファイルのデフォルトの場所は、TNS_ADMINによりシステム・プロパティoracle.net.tns_adminとして指定されます。Oracle Database Release 18c以降のOracle JDBCでは、次に示すように、TNS_ADMINを環境変数としてまたはURLにおいて定義できるようになっています。

```
jdbc:oracle:thin:@//myhost:1521/orcl?TNS_ADMIN=/home/oracle/network/ admin/
```

JDBC接続文字列内のプロパティ

このリリース以降は、Oracle JDBCのプロパティを接続文字列内で設定可能です。次の例では、暗黙的な文のキャッシュ・サイズ値を60に設定しています。

```
jdbc:oracle:thin:@(description=
  (address=(protocol=tcps)(port=1521)(host=example1.com))
  (connect_data=(service_name=example2.com)))?
oracle.jdbc.implicitStatementCacheSize=60
```

プロパティ・ファイル

Oracle Database 18c以降のOracle JDBCドライバは、Oracle Cloudとともにオンプレミス接続を簡素化するため、プロパティ・ファイルの仕組みに対応しています。デフォルトのプロパティ・ファイルはojdbc.propertiesで、そのデフォルトの場所はTNS_ADMINの値（Oracle JDBCのURLに含まれるか、システム・プロパティ（oracle.net.tns_admin）または環境変数を介して設定される、どちらかの値セット）によって定義され、
$TNS_ADMIN/ojdbc.propertiesのように記述します。

データベースのエイリアスojdbc_<dbalias>_properties（ojdbc_orcl_propertiesなど）から命名した追加ファイルを使用できます。デフォルトとデフォルト以外の両方のファイルが存在する場合は、デフォルト以外のファイルが優先されます。プロパティ・ファイルについて詳しくは、オンラインのOracle JDBC Javadocで説明されています。

ウォレットの場所のプロパティ

Oracle Database Release 18c以降のOracle JDBCドライバは、次のようにウォレットの場所を指定するため新しいプロパティ
my_wallet_directory に対応しています。

```
dbaccess =
    (DESCRIPTION=
          (PROTOCOL=tcps)
          (Host=hostname)
          (Port=1522))
      (CONNECT_DATA=
            (SERVICE_NAME=myservicename))
            (Security=(my_wallet_directory=$TNS_ADMIN/jnetadmin_c/)))
```

ウォレットの場所は、次のようにしてシステム・プロパティoracle.net.wallet_locationによって設定することもできます。

```
java –cp .:oraclepki.jar:ojdbc10.jar –D oracle.net.wallet_location= file:/ path/to/wallet/cwallet.sso MyApp
```

Oracle Easy Connect Plus

Oracle Database Release 11以来使用可能になった簡易接続ネーミングの仕組みでは、エイリアスを必要とせずに（よって、構成またはディレクトリ・ネーミング・サービスも不要）、ポート、プロトコル、サービスのデフォルト値を使用して接続文字列を動的に拡張するようになっていますが、デフォルトのサービス名はDEFAULT_SERVICE_listener.oraファイルで定義する必要があります。
一般的なEasy Connect構文は次のとおりです。
[//]host[:port][/service_name][:server][/instance_name]

次のJava接続文字列 jdbc:oracle:thin:@sales-server は、jdbc:oracle:thin@//sales-server:1521に拡張されます。

このリリースでは簡易接続ネーミングが改良され、接続文字列において、TCPS、複数のホストまたはポート、グローバル・データベース・サービス名、オプションのウォレットの場所、オプションのデータベース・サーバー識別名、名前/値ペアとしての接続プロパティの受渡しに対応しています。

一般的なEasy Connect Plus構文は次のとおりです。

```
"[[protocol:]//]host1[,host12,host13][:port1][,host2:port2][/service
_name][:server][/instance_name][?[wallet_location=dir][&ssl_server_c ert_dn=dn],...]"
```

クエスチョン・マーク（?）は名前/値ペアの始まりを示し、アンパサンド（&）は名前/値ペア間のデリミタです。

次のようなEasy Connect Plusベースの接続文字列があるとします。

```
jdbc:oracle:thin:@tcp://salesserver1:1521,salesserver2, salesserver3:1522/sales.us.example.com
```

これは、次のように拡張されます。

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
        (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver1)(PORT=1521))
        (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver2)(PORT=1522))
        (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver3)(PORT=1522)))
          (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```

すべてのOracle JDBCの接続プロパティ（サーバーのドメイン名とその他多くを含む）は、Oracle Easy Connect Plus構文を使用してURL内で指定可能です。

たとえば、次の接続文字列があるとします。

```
jdbc:oracle:thin:@tcp://myorclhostname:1521/myorclservicename?
oracle.jdbc.implicitStatementCacheSize=100
```

これは、次のように拡張されます。

```
jdbc:oracle:thin:@(description=(address=(protocol=tcp)(port=1521)(ho
st=myorclhostname))(connect_data=(service_name=myorclservicename)))?
oracle.jdbc.implicitStatementCacheSize=100
```

詳しくは、Oracle JDBCのドキュメントのセクション8.2.5を参照してください。

Oracle Cloudでのデータベースへの Java接続

Javaアプリケーションを自律型クラウド・データベース・サービス（Oracle Autonomous Transaction Processing（Oracle ATP-S、Oracle ATP-D）またはOracle Autonomous Data Warehouse Cloud（Oracle ADW）に接続するには、次の簡単な構成ステップに従う必要があります。

- もっとも重要なステップはOracle Cloudコンソールからクライアントの資格証明を取得することです。これは、tnsnames.ora、sqlnet.ora、ojdbc.properties、 keystore.jks、truststore.jks、cwallet.ora、ewallet.p12などの構成ファイルをおもに格納したzipファイルです。詳しくは、こちらを参照してください（このドキュメントの前提条件に含まれるステップ2を参照）。
- 最新のJDK8（JDK8u163以降）、JDK9、JDK10またはJDK11を入手します。
- 最新のOracle JDBC jarおよびUCP jarファイルをオラクルのMavenリポジトリまたはOracle JDBCのダウンロード・ページから入手します。19.3のOracle JDBCドライバはCentral Mavenで入手できます。
- ojdbc.propertiesプロパティ・ファイル（およびojdbc_<aliasname>.properties）は、標準でOracle Walletと連携動作するように事前に構成されていますが、次のようにしてJKS用に構成する必要があります。

```
oracle.net.ssl_server_dn_match=true
javax.net.ssl.trustStore=${TNS_ADMIN}/truststore.jks
javax.net.ssl.trustStorePassword=welcome1
javax.net.ssl.keyStore=${TNS_ADMIN}/keystore.jks                ***Not needed for ATP-D
javax.net.ssl.keyStorePassword=welcome1.                        ***Not needed for ATP-D
```

このブログ・ポストはOracle ATP-Dをクローズアップしています。詳しくは、「クラウドのデータベース・サービスへのJava接続」ページを参照してください。

データタイプのサポート

JSONデータタイプ検証

Oracle Database 18c以降では、返された列がJSONデータタイプであることを確認するため、isColumnJSON()メソッドが使用されていることがあります。

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

```
ResultSetMetaData rsmd = rs.getMetaData();
OracleResultSetMetaData orsmd = (OracleResultSetMetaData)rsmd;
...
boolean json = orsmd.isColumnJSON(i);
```

INバインド変数としてのREF CURSOR

GitHubの例を参照してください。

セキュリティの改良された点

以前のリリースのOracle Databaseにおいて、Oracle JDBCドライバは、厳密認証のサポート、データの暗号化と整合性、SSL、Kerberos、RADIUS、セキュアな外部パスワード・ストア、Oracle Advanced Securityなどのいくつかのセキュリティ・メカニズムに対応してきました。
Javaアプリケーションの最新のセキュリティ機能には、サーバー・ドメイン名の検証、自動SSL認証、HTTPプロキシ構成などがあります。

サーバー・ドメイン名の検証

Oracle Database 18c以降のOracle JDBCドライバは、oracle.net.ssl_server_cert_dn接続プロパティまたはJDBC URL内のssl_server_cert_dnのいずれかでDNが指定されていれば、サーバーを自動的に認証するようになっています。

```
oracle.net.ssl_server_cert_dn="CN=test.us1.oracletest.com,OU=ST,O=Or acle,ST=California,C=US"
```

注：URLで設定されている値は、プロパティで設定されている値より優先されます。

自動SSL認証

公開鍵インフラストラクチャの自動解決

オラクルでは、公開鍵と証明書を使用するための公開鍵インフラストラクチャ（PKI）を提供していますが、Oracle PKIプロバイダを登録する必要があります。Oracle Database Release 18c以降のOracle JDBCドライバは、プロバイダの実装（すなわち、oraclepki.jar）がCLASSPATH上にある場合、またはoracle.net.wallet_location接続プロパティまたはシステム・プロパティが設定されている場合には、OraclePKIProviderをロードし、PKIプロバイダを自動的に解決（つまり、登録が不要）します。osdt_core.jarとosdt_cert.jarをCLASSPATHに含める必要があります。これらの追加のjarファイルは、Central Mavenで19.3 JDBCドライバと一緒にダウンロードされます。または、ojdbc8-full.zipまたはojdbc10-full.zipをダウンロードしてこれらのjarファイルを入手することもできます。

ウォレットの場所は、次のいずれかの形式で指定できます。

- file:/path/ewallet.sso"または"file:/path/cwallet.p12"または"file:/path/to/directory/
- (SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=/path/to/directory))
)
  -Doracle.net.wallet_location= '(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY= ...)))'

Javaと一緒にOracle WalletのSSL認証を使用する場合にも、osdt_core.jarとosdt_cert.jar をCLASSPATHに含める必要があります。

```
java -cp
../ojdbc10.jar:./oraclepki.jar:./osdt_core.jar:./osdt_cert.jar:.
StatementSample
```

自動キーストア・タイプ解決

Oracle Database Release 18c以降のOracle JDBCドライバは、キーストア（javax.net.ssl.keyStoreプロパティの値）ファイルの拡張子とトラストストア（javax.net.ssl.trustStore プロパティの値）ファイルの拡張子に基づいてキーストア・タイプを解決できるようになっています。

a) ファイル拡張子.jksはJKSとしてjavax.net.ssl.keyStoreTypeに解決される
b) ファイル拡張子.ssoはSSOとしてjavax.net.ssl.keyStoreTypeに解決される
c) ファイル拡張子.p12はPKCS12としてjavax.net.ssl.keyStoreTypeに解決される
d) ファイル拡張子.pfxはPKCS12としてjavax.net.ssl.keyStoreTypeに解決される

キーストアまたはトラストストアがkss://を含むURIの場合、これはKSSタイプにマップされます。

HTTPSプロキシ構成のサポート

HTTPSプロキシを使用すると、クライアント側のファイアウォールでアウトバウンド・ポートを開く必要がなくなるため、パブリック・クラウド・データベース・サービスへのアクセスが可能になります。
Oracle Database 18c以降のOracle JDBCドライバは、以下に示すように、接続文字列でのHTTPSプロキシ構成に対応しています。

```
(DESCRIPTION=
    (ADDRESS=
            (HTTPS_PROXY=salesproxy)
            (HTTPS_PROXY_PORT=8080)
            (PROTOCOL=TCPS)
            (HOST=sales2-svr)
            (PORT=443))
      (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```

パフォーマンスとスケーラビリティ

自律型データベース（Oracle ATPとOracle ADW）のクラウド・サービスとオンプレミスのOracle Databaseを使用してJavaアプリケーションのパフォーマンスとスケーラビリティを最大限に引き出すことには、データベース接続の高速化、SQL文の処理の高速化、ネットワーク・トラフィックの最適化、インプレース処理、Javaワークロードのスケールアウトなど、こちらのブログ・ポストで説明されている操作が含まれます。
Oracle Database Release 19cと18cでのJavaアプリケーションのパフォーマンスとスケーラビリティにおいて改良された最新の点には、Memoptimized Rowstore、Oracle RAC環境におけるデータ・アフィニティ、Connection Manager in Traffic Director Mode（CMAN-TDM）、シャード・ルーティングAPIなどがあります。

Memoptimized Rowstore

Oracle Database 19cで導入されたこの新しい機能により、高速取込み（すなわち、相当数のクライアントから同時に少量のデータを高速取込み）と高速検索（すなわち、非常に高頻度なデータ問合せ）に対応できます。

次のオプションを指定して表が作成されているとします。

```
CREATE TABLE customers (
                    id NUMBER(20,0),
                    name VARCHAR2(90 BYTE),
                    region VARCHAR2(10 BYTE)
            )
            segment creation immediate
            memoptimize  or write
            ;
```

Oracle JDBCを介しての次のSQL INSERT文は、データの高速取込みを実行します（すなわち、非常に高速に返されるユーザー・コール）。

```
INSERT /*+ MEMOPTIMIZE_WRITE */ INTO CUSTOMERS VALUES (2, 'JOHN DOE', 'NORTH');
```

Memoptimized Rowstoreについて詳しくは、データベース・パフォーマンス・チューニングのドキュメントのセクション12.5を参照してください。

Oracle Connection Manager in Traffic Director Mode（CMAN-TDM）

CMAN-TDMは、Oracle Connection Manager（Oracle CMAN）の改良版で、文キャッシュ、行プリフェッチ、結果セット・キャッシュ、接続多重化などのパフォーマンス機能を透過的に提供するデータベース・プロキシです。またCMAN-TDMは、セキュリティと高可用性の透過的なメカニズムも備えています。Oracle Database Release 18c以降のOracle JDBCドライバはCMAN-TDMに対応しています。Javaアプリケーションはその利点を透過的に得ています。

CMAN-TDMについて詳しくは、こちらのWebページを参照してください。

Oracle RACのデータ・アフィニティ

Oracle RAC環境では、パーティションがOracle RACシステムの特定のインスタンスに"関係付け"られるような方法で、表内のデータがパーティション化（すなわち、行のサブセット化）されます。データの"関係付け"をOracle RACインスタンスにローカライズすると、スケーラビリティが向上します。
Oracle Database 18c以降、Oracle Universal Connection Pool（Oracle UCP）として知られるOracle Java Connectionプールは、次のようにしてOracle RACのデータ・アフィニティに対応しています。

1) UCPでデータ・パーティションのトポロジを収集する
2) Oracle RACのデータ・アフィニティを利用する必要がある接続リクエストにより、データベースのシャーディング・キーとコネクション・ビルダーを次のようにしてアフィニティ・キーを提供する

```
PoolDataSource pds = new PoolDataSourceImpl();
  // configure the datasource with the database connection properties.
  OracleShardingKey dataAffinityKey = pds.createShardingKeyBuilder()
  .subkey(1000, OracleType.NUMBER)
  .build();

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

詳しくは、Oracle Universal Connection Poolのドキュメントを参照してください。

シャード・ルーティングAPI

Java 9には、シャーディング・キー・オブジェクトを示すShardingKeyインタフェースが導入されており、すべての中間層接続プールはシャーディング・キーに基づいて接続リクエストを特定のシャードにルーティングします。Oracle JDBCドライバは、標準のシャーディング・キーAPIに対応しています。

ただし、各中間層で任意のシャードにアクセスすることが可能なため、中間層ごとにすべてのシャードされたデータベースのトポロジ全体がキャッシュされ、それらのシャードへの接続がプーリングされることになります。
そのようなシステム・リソース（すなわち、データベース接続）の無駄を省くため、Oracle Universal Java Connection Pool（Oracle UCP）は、各中間層（または中間層のいくつか）を特定のシャードと"連結する"ルーティングAPIを備えています。このパターンは"スイム・レーン"と呼ばれ、各レーンがシャードのインスタンスに届くまでWebサーバーをJavaアプリケーション・サーバーに"接続"します。
中間層とシャードの間でアフィニティを作成する利点は、各中間層で確立する必要がある接続の数が減り、システムの全体的スケーラビリティが向上することです。
以下にパブリック・インタフェースとパブリック・クラスを示します。

```
/**
  *  This class extends the UCP's internal shard routing cache
  *
  */
public class OracleShardRoutingCache extends ShardRoutingCache {
...
/**
      *  Creates an instance of a Shard Routing cache to be used
      *  by a mid-tier that needs to do shard-based routing.
      *  Once this cache is created,getShardInfoForKey can be used for
      *  every Sharding Key to know which shard needs to be used.
      */
public OracleShardRoutingCache(Properties dataSourceProps) throws
            UniversalConnectionPoolException {

    }
    ...
    /**
      *  Gets the information of each sharded database that maps to
      *  the sharding keys.
      */
public Set<ShardInfo> getShardInfoForKey(OracleShardingKey key, OracleShardingKey superKey) {

    }
}

/**
  *  When the routing cache is queried for shard information, a set of
  *  objects of this type is returned.
  *  Each object furnishes the required information about one of the shards
  *  referred to by the sharding keys.
  */
public interface ShardInfo {

    /**
      *  Returns the shard name.
      *
      *  @return shard name
      */
    String getName();

    /**
      *  Returns the shard's priority.
      *
      *  @return shard priority
      */
    int getPriority();

}
```

詳細と実際の例については、こちらのブログ・ポストを参照してください。

停止時間ゼロ

Javaアプリケーションの高可用性または停止時間ゼロは、Oracle Databaseの計画および計画外停止を持続する能力に依存します。このセクションでは、停止時間ゼロに寄与するOracle Database 19cおよび18cの最新の高可用性メカニズムを中心に説明します。

これらのメカニズムには、組込みJVM（別名、Oracle JVM）のローリング・パッチ適用、透過的アプリケーション・コンティニュイティ（TAC）、エータベース常駐接続プール（別名、DRCP）でのアプリケーション・コンティニュイティ（AC）への対応、TAC/ACでのレガシー・タイプのOracle JDBCへの対応などが含まれます。Oracle RAC、Oracle RAC One Node、Oracle Active Data GuardなどのOracle Databaseの高可用性構成は、必要ではあってもJava開発者には関係がないため、このセクションでは扱いません。このセクションでは、停止時間ゼロを実現するために高可用性メカニズムが果たす役割について説明します。

以前のリリースでは、ここでは説明していないものの、内部で使用されているかこのリリースで拡張されている、トランザクション・ガード機能とアプリケーション・コンティニュイティ機能が導入されました。詳しくは、Oracle Database Release 19cのOracle JDBCのドキュメントを参照してください。

組込みJVM（Oracle JVM）のローリング・パッチ適用

Release 8i以降のOracle RDBMSには、SQLおよびPL/SQLと同じメモリ領域とプロセスでJavaコードを実行するため、Java VM（別名、Oracle JVM）が組み込まれています。Oracle JVMのGITHUBの例を参照してください。

JVMは、同じデータベースにアクセスするすべてのセッション間で共有されるJavaシステム・クラスで構成されています。クラスタ化データベース構成では、Oracle Real Application Clusterテクノロジーにより、いくつかのRDBMSインスタンスが組込みJava VMシステム・クラスを含む同じデータベースに同時にアクセスすることができます。

Oracle JVMクラスにパッチを適用する場合の課題は、すべてのRDBMSインスタンスを停止する必要があるために、JavaおよびJava以外のアプリケーションもすべて停止しなければならいことです。

Release 18c（18.4）以降、Oracle JVMへのパッチ適用は計画メンテナンスの実施中にローリング方式で実行可能になっており、そのためJavaおよびJava以外のアプリケーションの停止時間ゼロ[2]が実現されています。

Oracle RAC環境の場合、Oracle JVMのローリング・パッチ適用は以下の3つのステップで実行されます。Oracle Cloudまたはデータベースの管理者がこの操作を行うため、Java開発者に求められるのは、確実に接続文字列内のRETRY_COUNTおよびRETRY_DELAYで短い一時停止時間に対応することだけです。

ステップ1：同じデータベースを共有しているRDBMSインスタンスのセットを2つのグループに分割します。

ステップ2：グループ1（すなわち、最初の半分）で、（Oracle JVM部を含む）Oracleバイナリを停止してパッチを適用します。JavaおよびJava以外のアクティビティは別の半分に再配置する必要があります。DBAは、Javaサービスの再配置方法を認識しています。Java接続プール（少なくともOracle UCPとJavaコンテナ）では、パッチ適用がスケジュールされているインスタンスから分岐した接続を停止する方法を認識しています（「*データベースの計画停止時間の維持*」を参照）。バイナリのパッチ適用後にこれらのシステムを再始動します。ただし、パッチが適用されたOracleバイナリはJVMシステム・クラス（これからパッチ適用される）と同期した状態ではなくなっているので、Java処理は再開しないようにする必要があります。これらのパッチ適用済みシステムからJavaの使用を試みても、ステップ3が完了するまではブロックされます。

ステップ3：残りの半分で、Oracle JVMシステム・クラスにパッチを適用します（言い換えると、古くなったOracle JVMクラスを新しいクラスと交換します）。このプロセスには、数秒から10〜15秒の時間（これは、クラスタ全体でのJavaの一時停止時間）を要します。
Oracle JVMシステム・クラスへのパッチ適用に続いて、グループ1でJava処理を再開できます（ブロックされた処理が再開します）**が、パッチが適用されていないOracleバイナリの第2グループでは新しいOracle JVMシステム・クラスを使用できません。これらのシステムにもパッチを適用する必要があり、その後に使用可能になります。**JavaおよびJava以外の処理はグループ2でも再開できます。

Oracle Cloud管理者またはDBAがそのようなOracle JVMのローリング・パッチ適用を実施しますが、Java開発者とアーキテクトは、推奨される接続文字列で示されているOracle JVMのパッチ適用プロセスと規定（retry- count、retry-period）のオーケストレートについて認識しておく必要があります。

[2] クラスタ全体で、Java処理のため2、3秒の一時停止時間が発生します。

高速アプリケーション通知

従来、Javaアプリケーションは、処理中のデータベース操作に関連付けられたタイムアウトの期限切れ時にデータベース・サービスが使用不可になったことについて通知されますが、タイムアウトを即時に把握できず、予測もできません。一方、高速アプリケーション通知（FAN）は即時通知機能を備えています。FANのイベントおよび通知は、次のイベントによってトリガーされます。Node Down、Public Network Down、Instance Down、Instance Up、Service Down、Service Up、Service Member Down、Service Member Up、Database Down、Database Up。

simplefan.jarには、Oracle JDBCドライバ、Java接続プール、JavaコンテナによってFANイベントをサブスクライブおよび管理するためのJava APIが組み込まれています。以前のリリース、サード・パーティの接続プール、コンテナの場合は、明示的に有効化および登録してFANイベントを受信する必要があります。Oracle JDBC Release 18cおよび19c以降のOracle JDBCドライバは、classpathに（FANイベントの送信システムをサポートする）simplefan.jarおよびons.jarの両方が含まれていると、高可用性データベース環境（すなわち、Oracle RAC、Oracle ADG）でFANを自動的に有効化します。

透過的アプリケーション・コンティニュイティ（TAC）

Oracle Database Release 12.1で導入されたアプリケーション・コンティニュイティ（AC）は、Oracle Databaseの高可用性メカニズム（取得と再実行）で、これによりOracle JDBCのドライバとRDBMSはドライバ・メモリ内で連携動作して、ドライバ・メモリ、1単位の処理（別名、"リクエスト"、通常、Javaトランザクション）で行われるすべてのデータベース・コール（SQL文、パラメータの割当て、セッション状態、トランザクション状態など）を取得し、計画外停止時（ネットワークの停止、ホストの停止、RBDMSインスタンスの停止）または計画メンテナンス中のセッションの強制停止時に、良好な状態のデータベース・インスタンスに対してそれらのコールしを再実行できるようにします。

Oracle Database Release 18cで導入された透過的アプリケーション・コンティニュイティ（TAC）は、アプリケーション・コンティニュイティ（AC）の改良版です。透過的アプリケーション・コンティニュイティの最終的な目標はゼロ・クライアント側構成であるため、データベースの新しいリリースごとに、クライアント側からサーバー側へ設定をプッシュし、取得と再実行時におけるセッション状態のカバレッジを拡げています。

このホワイト・ペーパーではJava開発者の観点からTACについてのみ扱います。ACおよびサーバー側構成については取り上げませんが、それらについて詳しくは、こちらのアプリケーション・コンティニュイティのホワイト・ペーパーで説明されています。

ACの場合、コールの取得の境界は、処理単位の開始（すなわち、プールからの接続のチェックアウト、明示的なconn.beginRequestコール）と終了（すなわち、プールへの接続の再チェックイン、COMMIT、ROLLBACK、または明示的なconn.endRequestコール）によって定められます。

TACの場合、そのような境界はOracle JDBCのドライバによって暗黙的または透過的に有効化されます。状態追跡インフラストラクチャによりOracle JDBCのドライバは、実行中のJavaアプリケーション・コード内に"リクエスト"境界を散在させることができます。暗黙的なリクエスト境界のメカニズムは、oracle.jdbc.beginRequestAtConnectionCreationシステム・プロパティをfalseに設定することによって無効化できます。

再実行データソース

TAC/ACを使用することが前提のJavaアプリケーションでは、以降に示すように、普通のOracle JDBCデータソースの代わりに再実行データソース（oracle.jdbc.replay.OracleDataSourceImpl）を使用して接続を取得する必要があります。
Oracle JDBCのドライバの再実行データソースは1単位の処理の開始時点でデータベース・コールの記録を開始します。通常は、新しいトランザクション、すなわち接続チェックアウト（getConnection()）の開始時、またはOracleConnectionオブジェクトでのbeginRequest()への明示的なコールの開始時です。新しいトランザクションには論理トランザクション識別子（LTXID）が関連付けられます。LTXIDは、実行中の処理単位がコミットされたかどうかをチェックするため、Oracle Transaction Guardが使用します（TAC/ACのさまざまな構成要素のオーケストレートについては、「*計画外データベース停止の維持*」を参照）。

処理単位の終わり（すなわち、COMMIT、ROLLBACK）に、接続がプールに再びチェックインされます（conn.close()、conn.endRequest）。再実行データソースが停止し、データベース・コールの記録を削除します。

```
...
import java.sql.Connection;
import javax.sql.PooledConnection;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.replay.OracleDataSourceFactory;
import oracle.jdbc.replay.OracleDataSource;
import oracle.jdbc.replay.OracleConnectionPoolDataSource;

…
OracleDataSource rds = OracleDataSourceFactory.getOracleDataSource();
Connection conn = rds.getConnection();
((OracleConnection) conn.beginRequest(); // Explicit request begin

…
OracleConnectionPoolDataSource rcpds =
OracleDataSourceFactory.getOracleConnectionPoolDataSource();
PooledConnection pc = rcpds.getPooledConnection(); Connection conn2 =
pc.getConnection(); // Implicit request beginRequest
```

セッション状態

TACでは、データベース・サービスのFAILOVER_TYPE属性をAUTOに設定する必要があります。そのため、FAILOVER_RESTORE属性もAUTOに設定します。FAILOVER_TYPEがAUTOに設定されていると、JDBCドライバは接続のチェックアウト時に新しい処理単位を自動的に開始します。oracle.jdbc.enableImplicitRequestsプロパティをFALSEに設定する（デフォルトではTRUE）と、暗黙的リクエスト境界をオフにすることができます。

FAILOVER_RESTOREがAUTOに設定されていると、TACでは再実行時に次のセッション状態がリストアされます。NLS_CALENDAR、NLS_CURRENCY、NLS_DATE_FORMAT、NLS_DATE_LANGUAGE、NLS_DUAL_CURRENCY、NLS_ISO_CURRENCY、NLS_LANGUAGE、NLS_LENGTH_SEMANTICS、NLS_NCHAR_CONV_EXCP、NLS_SORT、NLS_NUMERIC_CHARACTER、NLS_TERRITORY、NLS_TIME_FORMAT、NLS_TIME_TZ_FORMAT、NLS_TIMESTAMP_FORMAT、NLS_TIMESTAMP_TZ_FORMAT、CURRENT_SCHEMA、MODULE、ACTION、CLIENT_ID、ECONTEXT_ID、ECONTEXT_SEQ、DB_OP、AUTOCOMMIT states、ERROR_ON_OVERLAP_TIME、EDITION、SQL_TRANSLATION_PROFILE、ROW ARCHIVAL VISIBILITY、ROLEs、CLIENT_INFO。
注：Javaの場合、NLS_COMPとCALL_COLLECT_TIME はリストアされません。

ほとんどのJavaアプリケーションではこれらの状態をリストアするだけで十分ですが、再実行時に上記に含まれていないカスタム・セッション状態をリストアする場合、Java開発者は、"Oracle JDBCの接続初期化コールバック"または"UCP接続ラベル付けコールバック"のメカニズムを使用することがあります。これらのコールバックと初期状態リストアのメカニズムは両立しません（コールバックが状態リストアのメカニズムより優先されます）。

```
// Example of JDBC initialization callback implementation
import oracle.jdbc.replay.ConnectionInitializationCallback;
class MyConnectionInitializationCallback implements
ConnectionInitializationCallback
{
        public MyConnectionInitializationCallback()
        {
            ...
          }
```

```
        public void initialize(java.sql.Connection connection) throws SQLException
        {
                // Reset the state for the connection, if necessary
              ...
          }
 }
```

注：初期化コールバックは、接続がプールから借用されるときに毎回か、または再実行時に実行されます。これにはべき等性があり、実行中のトランザクションをコミットまたはロールバックしないようにする必要があります。

データベースの可変値関数

SYSDATE、SYSTIMESTAMP、SYS_GUID、sequence.NEXTVALなどのデータベースの可変値関数は、コールされるたびに新しい値を返します。その結果、データベース・サービスの計画外停止に続く実行中処理単位のTACまたはAC再実行時に結果セットが元の値と異なる値になり、システムはそれを再実行に失敗したとみなします。

Javaアプリケーションでこれらのいずれかの可変値関数を使用する場合は、再実行時にこれらの関数から返される値が（元のコールの場合と）変わらないようにするため、Oracle CloudまたはデータベースサービスのDBA管理者に、データベース・ユーザーに対してKEEP権限を付与するよう依頼する必要があります。

副次的影響

クライアント側のJavaアプリケーションからのデータベース・コールにより、HTTPコールアウト、FTPコールアウト（ファイル転送）、電子メール送信などのデータベース・セッション内から外部システム（RPC）に対してコールが行われることがあります。これらの外部コール（または副次的影響）もACによって再実行されます。一方TACには、これらの副次的影響を検出して自動的に無効化する機能があります。使用するJavaアプリケーションで副次的影響を再実行する必要がある場合は、ACにフォールバックし、oracle.jdbc.replay.ReplayableConnectionインタフェースのdisableReplay()コールを使用する必要があります。

```
if (connection instanceof oracle.jdbc.replay.ReplayableConnection)
{
  (( oracle.jdbc.replay.ReplayableConnection)connection).disableReplay();

}
```

レガシーOracle JDBCタイプ

以前のリリースのOracle JDBCには、具象Javaクラスとしてレガシー・データタイプが実装されており、インタフェースベース実装のjava.sqlタイプとは異なり、再実行時にプロキシ処理できませんでした（そのため、Metalink Note 1364193.1に従って非推奨となりました）。
Oracle Database Release 18c以降のOracle JDBCドライバは、アプリケーション・コンティニュイティ（AC）および透過的アプリケーション・コンティニュイティ（TAC）において次の具象クラスに対応しています。
oracle.sql.CLOB、oracle.sql.NCLOB、oracle.sql.BLOB、oracle.sql.BFILE、
oracle.sql.STRUCT、oracle.sql.REF、oracle.sql.ARRAY

具象クラスoracle.sql.OPAQUE、oracle.sql.STRUCT、oracle.sql.ANYDATAには対応していません。

DRCPでのアプリケーション・コンティニュイティ（AC）のサポート

Oracle Database Release 18c以降のOracle JDBCドライバは、サーバー側でデータベース常駐接続プール（DRCP）が構成されている場合にはアプリケーション・コンティニュイティに対応しています。
次の例に示すように、Java接続文字列によってDRCPを使用できるかどうかが決まります。

```
...
String url = "jdbc:oracle:thin:@(DESCRIPTION =
  (TRANSPORT_CONNECT_TIMEOUT=3000)
  (RETRY_COUNT=20)(RETRY_DELAY=3)(FAILOVER=ON)
  (ADDRESS_LIST =(ADDRESS=(PROTOCOL=tcp)
```

```
  (HOST=CLOUD-SCANVIP.example.com)(PORT=5221))
  (CONNECT_DATA=(SERVICE_NAME=ac-service)
(SERVER=POOLED)))";
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource ");
// Set DataSource Property
pds.setUser("HR");
pds.setPassword("hr");
System.out.println ("Connecting to " + url);
pds.setURL(url); pds.setConnectionPoolName("HR-Pool1");
pds.setMinPoolSize(2); pds.setMaxPoolSize(3);
pds.setInitialPoolSize(2);
Properties prop = new Properties();
prop.put("oracle.jdbc.DRCPConnectionClass", "HR-Pool1");
pds.setConnectionProperties(prop);
...
```

停止時間ゼロのJavaチェックリスト

- Oracle JDBCドライバの再実行データソース（oracle.jdbc.replay.OracleDataSourceImpl）を使用します（上記の「TAC」セクションを参照）。将来のリリースではこのような要件を取り除くことを検討中です。
- 最新のOracle JDBCドライバを入手し、推奨されるパッチを適用してTAC/AC[3]に対応します。
- CONNECT_TIMEOUT、RETRY_COUNT、RETRY_DELAY、TRANSPORT_CONNECT_TIMEOUTで、推奨されるJava接続文字列を適切な値を指定して使用して、データベース・サービスが一時的に使用不可になったときに（数十秒または数分の間）Javaアプリケーションがタイムアウトするのを回避します。
- Oracle JDBCの文キャッシュを有効にし、Javaコンテナの1つで文キャッシュを無効にします。これによりドライバでは、"リクエスト"の終わりに閉じられる文を識別し、メモリを解放することができます。
- *simplefan.jar*と*ons.jar*がclasspath内に必ず含まれるようにすることにより、高速アプリケーション通知（FAN）を有効化します。
- 以降で説明する計画メンテナンスまたは計画外停止を維持するためのステップに従います。

データベースの計画停止時間の維持

OSまたはRDBMSのパッチ適用、またはハードウェア・アップグレードの場合には、データベース・サーバーの計画停止が必要になることがあります。Oracle Cloudまたはデータベースの管理者は、これらの停止のスケジュールと通知を管理します。これらの停止からJavaアプリケーションを保護し、コンティニュイティを確保するため、Javaフレームワーク（ドライバ、接続プール、コンテナ）とユーザーJavaコードについて、以降で概要を示すいくつかのステップを実行する必要があります。

Oracle Cloudまたはデータベースの管理者のステップ

1. データベース・サービスが単一システムにおいてのみ実行可能な場合は、メンテナンス対象システムから別のシステムにそのサービスを再配置します。具体的なコマンドはJava開発者には関係しません。
    - このアクションにより、"Planned Maintenance FAN event"通知が、Oracle JDBCドライバ、またはそれらのイベントを受信するようにサブスクライブしたJava Connection Pool（UCP）やJavaコンテナに送信されます。

3 **TAC/AC用のDB19cのOracle JDBCバグ修正：29150338、29195279、29313347。**TAC/AC用のDB18cのOracle JDBCバグ修正：27748210、29313347、28381686、28504351。
TAC/AC用のDB19cのUCPバグ修正：27423500、29128935。TAC/AC用のDB18cのUCPバグ修正：27103398、27290134、27423500、27479395、29325356。
**TAC/AC用のWebLogicバグ修正：24919627、26336757

2. あるいは、他のシステムでもそのデータベース・システムを実行可能な場合は、メンテナンス対象のシステムで特定の方法に従ってデータベースを停止します。Java開発者には関係しないので、ここでは具体的なコマンドについては説明しません。
    - このアクションにより、"Planned Maintenance FAN event"通知が、Oracle JDBCドライバ、またはそれらのイベントを受信するようにサブスクライブしたJava Connection Pool（UCP）やJavaコンテナに送信されます。
3. Javaアプリケーションが実行中の処理を完了するまで待機し、メンテナンス対象システムに繋がっているデータベース接続を放棄します。この時間はドレイン期間と呼ばれます。
4. メンテナンス対象システムですべてのアプリケーションがそのアクティビティを停止したら、データベース・インスタンスまたはシステムを停止します。
    - 実行中の処理を完了できなかったり、その接続を放棄できなかったりするバッチ・ジョブまたはJavaアプリケーションに関連付けられたデータベース・セッションについては、強制終了が必要になることがあります。
5. メンテナンスが完了したら、そのサービスを元の場所に再配置するか、メンテナンスが終了したシステムで再起動することができます。

Javaインフラストラクチャ とJava開発者のステップ

"Planned Maintenance FAN event"通知を通知すると、Java Connection Pool（UCP）やJavaコンテナ（WebLogic、WebSphere、JBoss）では次のアクションが実行されます。

a) メンテナンス対象システムで実行されているデータベース・サービスのインスタンスに繋がっている分岐接続を停止します。

b) プールをクリーニング、すなわちメンテナンス対象システムで実行されているデータベース・サービスのインスタンスに繋がっているアイドル状態の接続を除去します。

c) Oracle Cloudのサービス管理者またはDBAが現実的なドレイン期間を指定したと仮定して、実行中の処理を完了させます。Java開発者も、oracle.ucp.PlannedDrainingPeriodシステム・プロパティで0（ゼロ）以外の値を指定してドレイン期間を指定することが可能です。

“Planned Maintenance FAN event”通知を受信すると、Oracle JDBCドライバは、以降で説明する"セルフドレインAPI"のいずれかが呼び出されたときに使用中の接続を閉じ、それらの接続が安全に除去されるようにします。

- java.sql.Connection.isValid(int timeout)
- oracle.jdbc.OracleConnection.pingDatabase()
- oracle.jdbc.OracleConnection.pingDatabase(int timeout)
- oracle.jdbc.OracleConnection.endRequest()
- Statement.execute（最初のコメント以外のトークンとして次のヒントを含む任意のSQLコマンド）

  /*+ CLIENT_CONNECTION_VALIDATION */

  例：/*+ CLIENT_CONNECTION_VALIDATION */ SELECT 1 FROM DUAL

Tomcat、WebLogic、WebSphere、Wildfly、JBossなどのJavaコンテナには、TestConnectionOnReserve、PreTest Connection、check- valid-connection-sql、TestonBorrowなどの特定の接続検証オプションがあります（データソース構成で指定）。

これらのAPIの役割は、APIのいずれかの呼出しの結果がFalseであった場合に、Javaアプリケーションが何らかの処理を実行する規定を作成したということをドライバに対して示すことです。接続の有効性をテストするということは、計画メンテナンスの準備において、ドライバまたは接続プールによって接続が無効と宣言された場合に、適切なアクションを実行する準備ができているということを意味します。

組込みJVM（別名、Oracle JVM）を使用してデータベースにおいてインプレース処理（Javaストアド・プロシージャその他）を実行するユーザーJavaコードは、通常、最上位レベルのコール（すなわち、JDBC CallableStatement）または中間層ツールやユーティリティから呼び出されます。これらのインデータベース・ユーザーJavaコードは、最上位Javaの計画メンテナンスの対象となります。

要約すると、計画メンテナンスの維持は、以下の条件が満たされていれば、通常はJavaアプリケーションに対して透過的です。つまり、Java開発者が最新のjarファイル（ojdbc10.jarまたはojdbc8.jar、ucp.jar、simplefan.jar、ons.jar）を構成しており、一般的なJavaのベストプラクティス（すなわち、使用中でない場合は接続をプールに戻す）とともにHAの推奨事項（すなわち、高可用性（上記参照）に対応する推奨接続文字列を構成し、長時間実行中のトランザクションまたはバッチ・ジョブの"セルフドレインAPI"を呼び出す）が実装されている場合です。

長時間実行中のトランザクションやバッチ・ジョブは、ベスト・プラクティスが実装されていない場合には終了可能です。その場合は、透過的アプリケーション・コンティニュイティなどの計画外停止メカニズムが作動し、サービスの使用可能なメンバーでコミットされていない実行中の処理が実行されます（または、単一インスタンスのサービスの再開時）。

データベースの計画外停止の維持

全般的機能

データベースの計画外停止は、データベース・システムまたはホストが停止する、データベース・システムを接続しているパブリック・ネットワークが停止する、データベース・インスタンスが停止する（システム全体ではない）、Javaアプリケーションが使用しているデータベース・サービスが停止する（データベース・インスタンス全体ではない）、サービスのメンバーが停止する（サービス全体ではない）などの状況のいずれかに当てはまります。

これらのいずれかの状況が発生すると、Oracle RACクラスタ内、またはOracle Active Data Guard構成のディザスタ・リカバリ・サイト内で動作を継続しているインスタンスまたはサービス・メンバーによって、対応するFANイベントが即時にパブリッシュされます。Javaドライバ、接続プール、またはコンテナなどのサブスクライバは即時に通知を受信し、ユーザーJavaコード（プレーンJava、microservice、Servlet、Beanなど）に対して適切なアクションを透過的に実行します。

Javaアプリケーションまたはフレームワークで最新のjarファイル（ojdbc10.jarまたはojdbc8.jar、ucp.jar、simplefan.jar、ons.jar）が構成されており、Javaアプリケーションにより一般的なJavaのベストプラクティス（すなわち、使用中でない接続を推奨される接続文字列（上記参照）とともにプールに戻す）を実装していれば、FANイベントがパブリッシュされサブスクライバが受信するとすぐ、透過的アプリケーション・コンティニュイティ（TAC）が起動します。

通常、UCP（など）は、ドライバおよびデータベースと連携して次のアクションを実行します。

1. アプリケーションからSQL例外を隠す（（リカバリ可能な例外であると想定して隠します。詳しくは、Oracle JDBCのドキュメントを参照）。
2. 動作停止したインスタンス/ノード/データセンターに繋がっている孤立接続を削除することによってプールをクリーンアップします。
3. 動作状態が良好なインスタンス/ノード/データセンターへの一時接続を作成してから、TAC/ACによって維持されているセッション状態を自動的にリストアするか、ユーザー実装の初期化またはラベル付けのコールバックを呼び出します。また、失敗した処理単位での設定に従い、対象のJava接続に対応するデータベース・セッションにKEEP権限が付与されていると想定して、可変値関数の値を再キャストします。
4. 内部でトランザクション・ガード（TG）をトリガーし、トランザクションの開始時に割り当てられた論理トランザクション識別子（LTXID）を使用して実行中の処理がコミットされた（またはロールバックした）かどうかをチェックします。データベース・セッションでCOMMITまたはROLLBACK演算子を受け取った後には、データベースとの接続損失が発生することがあります。トランザクション・ガードにより、TGチェックの後にコミットされていないトランザクションがコミットすることのないよう保証されるため、安全な再実行が実現します。
5. 実行中の処理単位の記録を再実行します。再実行データソースを有効化した接続ごとの記録は、ドライバのメモリに保管されます。
6. 再実行時、結果が同一（すなわち、結果セット・チェックサム）の場合は、TAC/ACが正常に機能しており、コントロールがアプリケーションに戻されて続行されます。同一でない場合は、例外が再キャストされ、（そのような例外を処理する規定があると仮定して）ユーザーJavaコードが表示されます。

その他の考慮事項

TAC/ACの機能はほとんどのJavaアプリケーションで使用できますが、TAC/ACで正常に再実行できない状況が発生する可能性があります。ORAchkユーティリティを実行することをお奨めします。My Oracle Support Note 1268927.2を参照してください。

結論

このホワイト・ペーパーでは、Java開発者またはアーキテクトを対象に、オンプレミスまたはOracle CloudのOracle Database 19cから最大限のパフォーマンス、スケーラビリティ、高可用性、セキュリティを引き出すためのベスト・プラクティスを示しました。

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

オラクルの情報を発信しています
+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

ホワイト・ペーパーOracle Database 19cでのJavaプログラミングJavaおよびOracle Database 19c Javaプログラミング Oracle Database 19c Javaプログラミング Oracle Database 19c
2020年1月
著者：Kuassi Mensah