ORACLE

Oracle Database 23aiで提供されているJava開発者向け機能

2024年6月、バージョン1.0

ORACLE

本書の目的

本書では、リリース23aiの機能と強化された点の概要が説明されています。本書は、23ai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。本書は、ユーザーとのライセンス同意書の一部をなすものではなく、またオラクルやその子会社および関連会社とのいかなる契約上の合意事項にも含まれるものではありません。

本書は情報提供のみを目的としたものであり、ここで説明する製品の機能を実装およびアップグレードする際の資料として使用され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

ORACLE

目的

本書では、リリース23aiの機能と強化された点の概要が説明されています。本書は、23ai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。

ORACLE

はじめに

Java開発者やJavaアーキテクトが本書を読むべき理由は何でしょうか。それには、いくつかの理由があります。この技術概要では、Oracle Database 23aiのJava機能の概要を説明します。新機能は、JavaによるOracle AI Vector Searchのサポート、容易な開発、マルチクラウド、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、ミッション・クリティカルなデプロイメント、セキュリティ、可用性の領域に対応します。

AI Vector SearchとVectorデータ型

ベクトルとは、1つまたは複数の数値、つまり、整数（...、-2、-1、0、1、2、...）または小数（...、-2.2、-1.1、0.0、1.1、1.1、...）の配列です。ベクトル埋込みは、オブジェクトの意味と関係を数学的に表現したものです。ベクトル埋込みは、モデルによって生成されます。

AI Vector Searchを使用すると、質問応答や類似度検索、つまりオブジェクト（単語、フレーズ、画像など）間のセマンティックな関係の検索が可能になります。

VECTORデータ型は、ベクトル埋込みを格納して索引付けすることで迅速な取得と類似度検索を実現するために導入されました。VECTORの可能な数値型、INT8、FLOAT32、FLOAT64になります。

```
CREATE TABLE my_vectors (id NUMBER, embedding VECTOR(768, INT8)) ;
```

この例では、各ベクトルに768のディメンションがあり、各ディメンションはINT8です。

JDBCによるVectorデータ型のサポート

JDBCによるVectorデータ型のサポートにより、開発者は、人工知能に焦点を当てて堅牢でスケーラブルな高パフォーマンスのJavaアプリケーションを構築できます。

INT8型の値は8ビットの2の補数で、Javaの"byte"に対応します。FLOAT32型の値は32ビットの浮動小数点値で、Javaの"float"に対応します。FLOAT64型の値は64ビットの浮動小数点値で、Javaの"double"に対応します。

JDBC API

Oracle JDBCドライバは、SQLType、DatabaseMetaData、ResultSetMetaDataとParameterMetaData、VectorMetaData、Java to SQL Conversions with PreparedStatement と CallableStatement 、SQL to Java Conversions with CallableStatement、SQL to Java Conversions with CallableStatmentとResultSet、VECTOR Datumクラスなど、Javaアプリケーション用のAI Vector Searchのサポートに必要なコンポーネントを実装します。

これらのAPIを使用することで、テキスト・センテンスのベクトル埋込みを使用して類似度検索を実行するインタラクティブなコマンドライン・プログラムを実装できます。

```
PreparedStatement query = connection.prepareStatement(
        "SELECT info"
          + " FROM my_data"
          + " ORDER BY VECTOR_DISTANCE(v, ?, COSINE)"
          + " FETCH APPROX FIRST ?ROWS ONLY");
```

詳しくは、Oracle JDBCのドキュメントを参照してください。

容易な開発、マルチクラウド

このリリースでは、容易な開発とマルチクラウドのための新機能が追加されています。

ORACLE

容易な開発

容易な開発の機能には、Oracle JVMによるJava SE 11およびJavaモジュールのサポート、JDBC-Thinによるネイティブ・ブール・タイプのサポート、JDBC-Thinによるリレーショナル JSON二面性ビューのサポート、Oracle JVM Webサービス・コールアウトの拡張機能、JDBC-ThinによるSQLアノテーションのサポート、マルチクラウドのサポートが含まれます。

JDBCによるJSONペイロードの配列エンキュー/デキューのサポート

このリリースでは、JDBCは、JSONペイロードを含むエンキューおよびデキュー呼び出しの配列オブジェクトの正確性に関するバグを修正します。

Oracle JVMでのJDK 11とJavaモジュールのサポート

このDB 23.4.0.24.05リリースでは、データベース常駐JVM（Oracle JVMとも呼ばれる）がJDK 11およびJavaモジュールをサポートするようになりました。
Project Jigsawは、Java 9で導入されたJavaモジュール・システムにつながりました。設計の目標は次のようなものでした。

- 開発者によるライブラリや大規模なアプリケーションの構築と保守を容易にする
- Java SEプラットフォームの実装全般、特にJDKのセキュリティと保守性を向上させる
- アプリケーション・パフォーマンスを改善できるようにする
- Java SEプラットフォームとJDKを、小型コンピューティング・デバイスや高密度のクラウド・デプロイメントで使用するためにスケールダウンできるようにする

Oracle JVMでのJavaモジュールのサポートは次のように機能します。

1. Oracle JVMシステムを構築するJava SEモジュールは、モジュールのルート・セットに自動的に含められます。
2. Javaクラスのmainクラスがモジュールのメンバーである場合、モジュールのルート・セットに追加されます。
3. 次のオプションのいずれかを使用して他のモジュールを追加できます。
    a. loadjava --add-modulesオプション：mainクラスがすでにロードされている場合。これはJDKのjava –-add-modulesコマンドライン引数に似ています。
    b. oracle.aurora.addmods：システム・プロパティを指定する場合。設定のプロパティについては、こちらを参照してください。
4. モジュールの最終セットは、Oracle JVMセッションの開始時に一貫性と完全性が検査されます。

JDBC-Thinによるリレーショナル JSON二面性ビューのサポート

リレーショナル JSON二面性ビューはOracle Database 23aiの新機能であり、これにより、開発者はリレーショナル側（領域管理の効率性、問合せ可能性、一貫性、SQLの強力な分析およびレポート機能）と、JSON側（自己記述的、自己完結型、プログラムによるJSONデータへのアクセスなどのスキーマレスの容易な開発、階層データ、共通交換形式、バイナリJSON、Javaタイプへの容易な変換など）の両方の利点を手にすることができます。

詳しくは、私のブログの記事を参照してください。

JDBCによるネイティブのBooleanデータ型のサポート

Oracle JDBCは、ISO SQL規格に準拠したBOOLEANデータタイプを、次のAPIを使用してoracle.jdbc.OracleTypeでサポートします。

INSERT

...

```
String query = "INSERT INTO BoolTable values (?) ";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setBoolean(1, true);
pstmt.execute();
```

...

FETCH

...

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from BoolTable");
while(rs.next()) {
    System.out.print("Value: " + rs.getBoolean("booleanColumn"));
```

...

}

ResultSetMetaData should return getColumnType = 16 and getColumnTypeName = BOOLEAN

私の同僚の@juarezjuniorさんは、彼のブログの記事において、BOOLEANデータタイプと、論理ブール値としてNUMBERおよびVARCHARデータタイプを使用する表列との互換性について広範囲に調査しています。

JDBC接続プロパティsendBooleanAsNativeBoolean

新規プロパティsendBooleanAsNativeBooleanにより、JDBCドライバの以前のリリースとの下位互換性が確保されます。

falseに設定すると（デフォルトはtrue）、このプロパティにより、整数（0/1）をBooleanデータ型として送信する古い動作がリストアされます。この機能により、Booleanデータ型の古い動作に依存するJavaアプリケーションに互換性が提供されるため、既存のJavaアプリケーションを中断することなく、最新のJDBCドライバへのアップグレードが簡素化されます。

Oracle JVM Webサービス・コールアウトの拡張機能

データベースを使用して外部RESTまたはSOAP Webサービスを起動したりコールアウトしたりする機能は、一般的なユースケースです。このリリースではOracle JVM Webサービス・コールアウト・ユーティリティにwadl2javaツールが組み込まれています。そのツールを別途ダウンロードする必要はなく、loadjavaユーティリティの-t <wadl2java tool location>オプションを使用してその場所を指定する必要もありません。

詳しくは、こちらを参照してください。

JDBC-ThinによるSQLアノテーションのサポート

データベースの表、ビュー、または列をアプリケーションのメタデータまたはアノテーションに関連付けることができるため、変更を一元管理できます。JDBC-Thinには、アノテーションを取得するための新しいAPIがあります。

```
getAnnotations(java.lang.String objectName, java.lang.String domainName,
java.lang.String domainOwner) throws java.sql.SQLException
```

getAnnotations(java.lang.String objectName, java.lang.String columnName, java.lang.String domainName, java.lang.String domainOwner) throws java.sql.SQLException

クラウド・コンピューティングとマルチクラウド

このセクションでは、Oracle Cloud Infrastructureを使用してクラウド・コンピューティングをサポートする新機能について説明します。マルチクラウドの分野では、オラクルとMicrosoftは協力して、JavaアプリがAzure（アプリ）とOracle Cloud（Autonomous Database）全体でシームレスに実行できるようにしてきました。新機能には、一元化されたAppConfigとリソース・プロバイダが含まれます。これらの機能の一部は、オンプレミスおよびプライベート・クラウドで使用できます。

AppConfig、リソース、およびトレース・イベント・プロバイダ

クラウド・コンピューティングでは、アプリケーションの構成やリソースをアプリケーション・コードとは別にクラウド・ストレージやVaultに保存し、それらを安全に取得する必要性が高くなります。一元化されたAppConfigを使用すると、Javaアプリケーションに変更を加えることなく、実行時に構成を変更できます。

このリリースでは、Oracle JDBCは、標準のサービス・プロバイダ・インタフェースに基づく新しい拡張性またはプラグイン・メカニズムを通じて、一元化されたアプリケーション構成とリソース・プロバイダをサポートします。

Java開発者は、アプリケーション構成全体や個別のデータ（例：データベース・ユーザー名、データベース・パスワード、データベース・アクセス・トークン、TLS/SSL構成）を、Oracle Cloud Infrastructure Object Storage、Azure App config、JSON httpsサーバーまたはオンプレミス（例：ファイル・システム）で保存および取得できるようになりました。データベース接続パスワード、データベース・アクセス・トークンなどの機密データは、シークレットとしてVaultに保存されます。
一元化により、Javaアプリケーションのコードを変更せずにシームレスな構成更新が可能になります。 https://github.com/oracle-samples/ojdbc-extensions

一元化された構成プロバイダ

JDBC URLの先頭が*jdbc:oracle:thin:@**config-<provider>***である場合（例：*jdbc:oracle:thin:@config-azure*）、ドライバは***config-<provider.jar>***（例：config-azure.jar）を登録済みのサービス・プロバイダのリストからロードしようとします。

現在サポートされている構成プロバイダのリストには、組込みプロバイダ、Azure App Config、OCI Object Storage、OCIデータベース・ツール接続、およびユーザー定義のカスタム・プロバイダが含まれています。JDBCプロバイダのGithubリポジトリを参照してください。

組込みプロバイダ：ファイル・システムとhttpsサーバー：

jdbc:oracle:thin:@config-file:config.json

jdbc:oracle:thin:@config-https://server/config/myappconfig?key=dev

1) Azure App Config connect_descriptorとオプションのユーザー。パスワードとwallet_locationもオプションですが、Azure Key Vaultのシークレットとして指定されます。
   *URL= jdbc:oracle:thin:@config**-azure**:{appconfig-name}[?key=prefix&label=value&option1=value1…]*
   例：jdbc:oracle:thin:@config-azure:myappconfig?key=sales_app1&label=dev
   図1で例示しているとおり、app-config名（例：*myappconfig*）、接頭辞（例：*/sales_app1*）、およびラベル（例：*dev*）はapp config内で指定された鍵と値のペアを取得するために組み合わせて使用されます。
   特定のOracle JDBCプロパティを、<prefix>/jdbcを使用して指定および取得することもできます。

Key	Value	Label
/sales_app1/user	scott	dev
/sales_app1/password	{"uri":"https://mykeyvault.vault.azure.net/secrets/passwordsalescrm"}	dev
/sales_app1/wallet_location	{"uri":"https://mykeyvault.vault.azure.net/secrets/walletcrm"}	dev
/sales_app1/connect_descriptor	(description=(retry_count=20)(retry_delay=3)(address=(protocol=tcps)(port=1521)(host=adb.us-phoenix-1.oraclecloud.com))(connect_data=(service_name=gebqqvpozhjbqbs_dbtest_medium.adb.oraclecloud.com))(security=(ssl_server_dn_match=yes)(ssl_server_cert_dn="CN=adwc.uscom-east-1.oraclecloud.com, OU=Oracle BMCS US, O=Oracle Corporation, L=Redwood City, ST=California, C=US")))	dev
/sales_app1/jdbc/autoCommit	false	dev

図1 - App config

2) OCI Object Storage connect_descriptorとオプションのユーザー。パスワードとwallet_locationはオプションであるか、またはAzure Key Vaultのシークレットとして指定されます。
URL = jdbc:oracle:thin:@config-ociobject:{object-url}[?key=name&option1=value1...]
connect_descriptorは、「Object Storage」/「Buckets」/「Object」→「Object Details」の下に保存されます。
例：jdbc:oracle:thin:@config-ociobject:https://objectstorage.us-phoenix-1.oraclecloud.com/n/oracleonpremjava/b/bucket1/o/payload_ojdbc_objectstorage.json

```
{"connect_descriptor": "(description=(retry_count=20)(retry_delay=3)(address=(protocol=tc
ps)(port=1521)(host=adb.us-phoenix-
1.oraclecloud.com))(connect_data=(service_name=gebqqvpozhjbqbs_dbtest_medium.adb.oraclecl
oud.com))(security=(ssl_server_dn_match=yes)))",
   "user": "scott", "password": {
      "type": "vault-oci",
      "value": "ocid1.vaultsecret.oc1.phx.amaaaaaxxxx", "authentication": {
         "method":"OCI_INSTANCE_PRINCIPAL"
      }
   }
```

3) OCIデータベース・ツール接続と、オプションでシークレット用のOCI Vaultへの参照。
各構成には、使用される接続の識別に使用されるOCIDがあります。これにはconnectionString、userName、userPassword、keyStores、advancedPropertiesが含まれます。
URL = jdbc:oracle:thin:@config-ocidbtools:ocid1.databasetoolsconnection.oc1.phx.ama ...

4) カスタム・プロバイダ：Java開発者は独自のプロバイダを、*oracle.jdbc.spi.OracleConfigurationProvider*インタフェースを実装することによって構築できます。そのインタフェースはJDBCドライバのjar内にあります。組込みプロバイダがそれを実装します。これらのプロバイダは名前を定義する必要があり、*java.util.Properties*を返します。

Javaアプリケーションは次のことが必要です。

- プロバイダJARファイルをクラスパスに含めるか、プロバイダ参照をPOMファイルに含める。
- 接続URLに必要な値を設定する。

リソース・プロバイダ

リソース・プロバイダは、データベース接続文字列、データベース・ユーザー名、データベース・パスワード、データベース・アクセス・トークン、TLS/SSL構成、トレース・イベント・リスナーなどの単一リソースをOracle JDBCに提供します。リソース・プロバイダは、oracle.jdbc.provider.connectionString、oracle.jdbc.provider.username、oracle.jdbc.provider.password、oracle.jdbc.provider.accessToken、oracle.jdbc.provider.tlsConfiguration、およびoracle.jdbc.provider.traceEventListenerなどのリソース・プロバイダの名前を識別する接続プロパティによって構成されます。

次のコード・スニペットでは、パスワード・プロバイダの構成に接続プロパティが使用されます。 oracle.jdbc.provider.password=password-provider oracle.jdbc.provider.password.vaultId=9999-8888-7777 "oracle.jdbcprovider.password"プロパティはパスワード・プロバイダの名前を構成し、"oracle.jdbc.provider.password.vaultId"プロパティはパスワード・プロバイダによって認識されるvaultIdを構成します。

トレース・イベント・リスナー・プロバイダ - OpenTelemetry

Oracle JDBCドライバは、問合せ実行中のデータベース・ラウンドトリップ、接続確立中のIPアドレスの再試行、データベース停止からのアプリケーション・コンティニュイティ（AC）リカバリの開始、ACリカバリの成功などのイベントを生成する場合があります。

JDBCドライバは、Oracle Database JDBCドライバからアプリケーションおよびシステム・トレース・イベントを受信するリスナーを定義します。詳しくは、oracle.jdbc.TraceEventListener javadocを参照してください。

JDBCドライバは、これらのイベントをOpenTelemetryに公開したりカスタムのTraceEventListenerを登録したりするためのリスナーの登録に使用できる、oracle.jdbc.spi.TraceEventListenerProviderサービス・プロバイダ・インタフェースも定義します。

1. OracleConnectionBuilder.traceEventListener(TraceEventListener)を使用してプログラムによって実装できます。
2. または、OracleResourceProvider実装を使用します。これはユーザー独自のものか、またはOracleのOpenTelemetry向けオープンソース・プロバイダのいずれかです。新しいTraceEventListener OpenTelemetryプロバイダは、TraceEventListenerProviderインタフェースの実装であり、こちら（https://github.com/oracle-samples/ojdbc-extensions/tree/main/ojdbc-provider-opentelemetry）で公開されています。
3. oracle.jdbc.provider.traceEventListener接続プロパティを使用してトレース・イベント・リスナー・プロバイダを識別します

   oracle.jdbc.provider.traceEventListener=example-provider
   oracle.jdbc.provider.traceEventListener.traceLevel=INFO

SpringBootの例は次のとおりです

spring.datasource.url=jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:1521/xyz.adb.oraclecloud.com?oracle.jdbc.provider.traceEventListener=open- telemetry-trace-event-listener-provider

その他のJDBCプロバイダについては、こちらを参照してください。https://github.com/oracle-samples/ojdbc-extensions将来のリリースでは、より多くの組込みプロバイダとオープンソース・プロバイダが導入される予定です。

Javaのパフォーマンスとスケーラビリティ

パフォーマンスとスケーラビリティのための新しい機能には次のものがあります。True Cacheデータソース、パイプライン化されたデータベース操作に対するJDBCのサポート、Reactive Streams Ingestionライブラリ（RSI）の新しいDataLoadモード、DRCPのマルチプール・サポート、UCPによるシャーディング分割パーティション・セットのサポート、UCPによるシャーディング時のXAトランザクションのサポート、UCPによる接続作成コンシューマとコールバックのサポート、UCPによる最大接続再利用時間のサポート、UCPへのリアクティブ拡張、JDBC-ThinによるBequeathプロトコルのサポート、executeBatch()/andexecuteLargeBatch()のパフォーマンス強化、およびUCP接続借用の強化です。

ORACLE

True Cacheデータソース

True Cacheインスタンスは、プライマリOracleデータベースの完全に機能する読取り専用レプリカです。インメモリで、ほとんどはディスクレスです。これは中間層に存在し、アプリケーションと同じ場所に置かれます。新しいoracle.jdbc.useTrueCacheDriverConnectionプロパティをtrueに設定することで、JDBCレベルで有効になります。True Cacheデータソースを有効にすると、True Cacheデータベース・インスタンスまたはプライマリ・データベースに対して使用できる論理接続が作成されます。

接続を読取り専用としてマークするために、True Cacheデータソースは標準のjava.sql.Connection.setReadOnly(boolean)およびjava.sql.Connection.isReadOnly()メソッドを使用します。デフォルトでは、接続の読取り専用モードはfalseに設定されています。次の例は、True Cacheの使用方法を示しています。https://gist.github.com/Kuassim/350e775c6dcf7b4448418c920b4f9425

パイプライン化されたデータベース操作

データベース・パイプラインは、操作間のレスポンスを待たずに送信される複数のデータベース・リクエストのシーケンスで構成されます。データベースは、各問合せの結果が準備できるとレスポンスを送信します。パイプライン・データベース操作は、非同期プログラミング・モデルを促進します。このモデルでは、ユーザー・スレッドは、実行のためにSQL文を送信すると、その実行とResultSetを待たずにすぐに戻ります。

Java開発者は、Oracle JDBC Reactive Extensions、Reactive Streamsライブラリ（R2DBC、Reactor、RxJava、Akka Streams、Vert/xなど）、Java仮想スレッドを通じてデータベース・パイプライン処理を活用します。
これらの各ユースケースと、標準的なJDBCバッチ処理の透過的なパイプライン操作サポートを示すコード・サンプルについては、私のブログの記事を参照してください。

Reactive Streams Ingestion向けの新しいDataLoadモード

Reactive Streams Ingestionライブラリ（RSI）を使用すると、ダイレクト・パス・ロードと Reactive Streamsメカニズムを使用して、Oracleデータベースにデータを高速に取り込むことができます。これは、Java接続プール（UCP）、表パーティション、Oracle RAC接続アフィニティ、およびOracle Globally Distributed Database（以前のOracle Database Sharding）を利用します。

このリリースでは、新しいDataLoadモードがデフォルトのストリーム・モードに追加されました。ストリーム・モードでは、ワーカー・スレッドはJDBC接続のプールを共有し、取り込まれたデータは頻繁にコミットされます。一方でDataLoadモードでは、接続数が大きくなる可能性があり、取り込まれたデータはRSIインスタンスのクローズ時にのみコミットされます。

```
ReactiveStreamsIngestion.Builder rsiBuilder = ReactiveStreamsIngestion.builder()
        .useDataLoadMode()
        .username("<user_name>")
        .password("<password>")
        .url("jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
(HOST=myhost.com)(PORT=5521))(CONNECT_DATA=(SERVICE_NAME=myservice.com)))")
        .table("customers")
        .columns (new String[] { "id", "name", "region" });
// try-with-resource文を使用して文の最後に
// RSIインスタンスが閉じられるようにします。
try (ReactiveStreamsIngestion rsi = rsiBuilder.build()){
     // レコードを公開します。
}
```

詳細とコード・サンプルについては、『Oracle Database JDBC開発者ガイド、23ai』を参照してください。

UCPによるデータベース・シャーディング時のXAトランザクションのサポート

WebLogic ServerのUCPネイティブ・データソースを使用してOracleシャード・データベースに接続するJavaアプリケーションは、WebLogic Transaction Manager（TM）によって管理されるJTA/XAトランザクションに参加できます。

ORACLE

UCPによる接続作成コンシューマとコールバックのサポート

この機能により、Javaアプリケーションは特定のPoolDataSourceオブジェクトに対して"接続作成コンシューマ"を登録できます。そのコンシューマは、明示的な（つまり、Javaアプリケーションによる）または暗黙的な（つまり、サイジング設定を調整するUCPによる）接続の作成時に通知（つまり、コールバック）されます。

詳細と、接続作成コンシューマの登録、登録解除、ステータスの確認などのコード・サンプルについては、UCP開発者ガイドの第4章を参照してください。

UCPによる最大接続再利用時間のサポート

この機能の一般的なユースケースは、中間層とデータベース層がファイアウォールによって分離されている場合です。その場合、一部の接続がファイアウォールによってブロックされ、プール内で長時間アイドル状態のままになる可能性があります。
最大接続再利用時間（秒単位）をファイアウォールのタイムアウトよりも小さい値に設定すると、このような状況を回避できます。

pds.setMaxConnectionReuseTime(300);

新しいシステム・プロパティoracle.ucp.timersAffectAllConnectionsをTRUEに設定すると、定期的なポーリングが可能になり、接続可能なすべての接続をチェックして最大接続再利用時間を確認できます。

詳しくは、『Universal Connection Pool開発者ガイド』のセクション5.4.1.1を参照してください。

DRCPでのマルチプール・サポート

データベース常駐接続プール（DRCP）は、プラガブル・データベース（PDB）レベル、またはコンテナ・データベース（CDB）とも呼ばれる管理インフラストラクチャ・レベルのいずれかにあるRDBMS側プールです。Javaアプリケーションは、JDBC接続文字列で(SERVER=POOLED)を使用してDRCPを参照できます。

新しいマルチプール機能を使用すると、接続文字列内で(POOL_NAME= <pool_name>)を使用してサブパーティションに名前を付けることで、複数のアプリケーション間でDRCPをサブパーティション化できます。

DRCPとマルチプールを使用した接続文字列の例を次に示します。

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=)(PORT=))
                (CONNECT_DATA=(SERVER=POOLED)(POOL_NAME=)))
```

dbms_connection_poolパッケージのadd_pool ()およびremove_pool()プロシージャを使用して、マルチプールにプールを追加したり、マルチプールからプールを削除したりできます。

```
exec dbms_connection_pool.add_pool('mypool')

exec dbms_connection_pool.remove_pool('mypool')
```

UCPへのリアクティブ拡張

以前のOracle Database 21cリリースでは、Java.util.concurrent.Flowインタフェースの実装であるリアクティブ拡張機能がOracle JDBCに導入されました。今回のリリースでは、Java接続プール（UCPとも呼ばれる）がリアクティブ拡張機能で拡張され、非同期接続借用リクエストを発行できるようになりました。
これは次のように機能します。

1. UCPConnectionBuilderまたはUCPXAConnectionBuilderのいずれかをインスタンス化します。
2. CompletableFuture<Connection>またはPublisher<Connection>のいずれかを使用して、UCPConnectionBuilderで普通の接続を非同期的に要求します。あるいはCompletableFuture<XAConnection>またはPublisher<XAConnection>のいずれかを使用して、UCPXAConnectionBuilderでXA接続を非同期に要求することもできます。
3. Javaコードにjava.util.concurrent.Executorインタフェースを実装していない場合、非同期借用操作はデフォルトのForkJoinPoolエグゼキュータで実行されます。
4. 借用操作が完了すると、CompletableFutureまたはパブリッシャに通知されます。
5. その後、接続オブジェクトに対して操作を実行できます。

『Universal Connection Pool開発者ガイド』の第11章にあるコード・サンプルを参照してください。

JDBC-ThinによるBequeathプロトコルのサポート

このプロトコルにより、同じLinuxホスト上にあるデータベース・クライアント（例：Java/JDBCアプリケーション）とデータベース・サーバー・プロセスが、ネットワーク・レイヤーやネットワーク・リスナーを介さずに直接通信できるようになります。このプロトコルを使用してOracleデータベースに接続するには、ORACLE_HOME変数とORACLE_SID変数の値を、接続URL（後述）またはアプリケーション環境変数として設定する必要があります。

jdbc:oracle:thin:@(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq))(ENVS=ORACLE_HOME=/var/lib/oracle/dbhome,ORACLE_SID=oraclesid))

executeBatch() executeLargeBatch()への拡張

このリリースでは、PreparedStatement.executeBatch()およびPreparedStatement.executeLargeBatch()の応答時間は、2 MBより大きなバッチの1回のラウンドトリップを通じて大幅に改善されました。前のリリースでは、JDBCドライバは各DMLのバッチ操作でデータベースを複数回呼び出していました。

UCP接続借用の強化

UCPがリクエストを満たすために新しい接続を作成する必要がある場合、接続の借用には、*connectionWaitTimeout*で指定された値よりも長い時間がかかることがあります。このリリースでは、接続が別のスレッドによって解放されている間は、UCPは別の接続が作成されるのを待つのではなく、解放されたばかりの接続を割り当てます。

ミッション・クリティカルなデプロイメント、セキュリティ、可用性

このセクションでは、Javaアプリケーションのミッション・クリティカルなデプロイメント、セキュリティ、可用性（ゼロダウンタイム）をサポートする新機能について説明します。

ミッション・クリティカルなデプロイメント

このセクションでは、新しいApp Stack for Javaと、新しい可観測性機能の概要を説明します。

IaC - App Stack for Java

Javaアプリケーションをクラウドにデプロイする場合、開発者は、仮想ネットワークの構成、デプロイメント・プラットフォームとデータベースのプロビジョニング、JDBCデータソースおよびロード・バランサとDNSの構成、CI/CDパイプラインのビルドとデプロイメントの生成など、いくつかの問題点に直面します。Infrastructure as Code（IaC）フレームワーク（App Stack for Java）を使用すると、クラウド・インフラストラクチャ・リソースのプロビジョニングに関連するほとんどのタスクを自動化できるため、それらの問題点は解消されます。

App Stack for Javaは、技術的にはOracle Database 23aiリリースの一部ではありません。このブログ投稿で取り上げたので、ぜひ確認してください。

可観測性

可観測性とは、アプリケーションのログ、メトリック、トレースを監視し、キャプチャし、動的に分析して、問題をリアルタイムで診断することを指します。これは、最新のサービスベースのアプリケーションの開発、デプロイメント、DevOpsなどにとって重要な要件です。このリリースでは、強化されたロギング、新しいデバッグ（最初の障害時の診断）、および新しいトレース機能が追加されています。

SQL文のID[1]

パフォーマンス上の問題を診断するためにSQL文のハッシュ値を取得する必要はあるでしょうか。

[1] https://docs.oracle.com/en/database/oracle/oracle-database/23/jajdb/oracle/jdbc/OracleStatement.html#getSqlId()を参照してください。

oracle.jdbc.OracleStatementインタフェースの新しいgetSqlid()メソッドは、対象の文オブジェクトのSQL IDを返します。

すべてのユースケースに単一のJar

まず、うれしいニュースとして問題を調査するために本番環境の*jar*とデバッグ*jar*を切り替える必要はなくなりました。

すべてのユースケース（本番、デバッグ、メトリック）に、単一のojdbc jar（例：*ojdbc8.jar*、*ojdbc11.jar*）で対応できます。言い換えれば、デバッグの場合の*ojdbc8_g.jar*または*ojdbc11_g.jar*、Oracle Dynamic Monitoring Service（DMS）メトリックの場合の*ojdbc8dms.jar*または*ojdbc11dms.jar*、DMSデバッグの場合の*ojdbc8dms_g.jar*または*ojdbc11dms_g.jar*は必要なくなりました。

最初の障害時に診断（自己主導型診断機能）
この機能は、Javaアプリケーションで最初に発生した障害を診断します。重要な実行状態をメモリに記録し、エラーについての記録をダンプします。これは常にON（デフォルト）ですが、-Doracle.jdbc.diagnostic.enableDiagnoseFirstFailure=false または DiagnosticMBeansインタフェースを使用して無効にできます。最初の障害時の診断で診断出力を取得するには、java.util.loggingを構成する必要があります。

詳しくは（特に機密データの取り扱いについては）、ブログ投稿を参照してください。

ロギング

コアJDBC Jar（例：*ojdbc8.jar*または*ojdbc11.jar*）には、次のプロパティを使用して有効にする必要があるロギング機能が含まれています。
-Doracle.jdbc.diagnostic.enableLogging=true.
-Djava.util.logging.config.file=./logging.config
ハンドラとロギングの粒度も指定する必要があります。

分散化トレース - OpenTelemetry

9ページの**トレース・イベント・リスナー・プロバイダ - OpenTelemetry**を参照してください。

DMSメトリック

Oracle Dynamic Monitoring Service（DMS）メトリックは、*dms.jar*がクラスパス内にある場合に記録されます。

セキュリティ

Javaの新しいセキュリティ機能には次のものがあります。
サムプリントベースの証明書の選択、LDAP/LDAPSの簡易接続のサポート、JDBCによるDRCPを使用したTLSのサポート、OJVMによるFIPSのサポート、より長いパスワードのサポート、Oracle Cloud Infrastructure（OCI）Identity and Access Management（IAM）トークンを使用したOCI IAMのトークンベースの認証、OAuth 2.0アクセス・トークンを使用したAzure Active Directory（AD）のトークンベースの認証、RADIUSチャレンジ/レスポンス認証（2FAとも呼ばれる）、Kerberosサポートの拡張、Oracle JVMによるHTTPおよびTCPのサポート。

サムプリントベースの証明書の選択

JDBCシン・ドライバは、KeyStoreに複数の証明書が存在する場合はエイリアスに基づいて証明書を選択します。選択された証明書とそのサムプリントは、TLSハンドシェイクによるクライアント認証に使用されます。

LDAP/LDAPSの簡易接続のサポート

構文は、LDAP URL構文とEasy Connect構文を次のように組み合わせたものになります。
ldap[s]://host[:port]/name[,context]?[parameter=value{&parameter=value}]
例：

sqlplus
"<user_name>/<password>@ldaps://<host_name>/test?DIRECTORY_SERVER_TYPE=oid&WALLET_LOCATION

=/oracle/network/admin&AUTHENTICATE_BIND=true&AUTHENTICATE_BIND_METHOD=LDAPS_SIMPLE_AUT H"

詳しくは、Oracle JDBCのドキュメントを参照してください。

OJVMによるFIPSのサポート

データベース常駐JVM（OJVMとも呼ばれる）では、FIPS 140-2 Javaクラスをインストールできるようになり、JsafeJCEがデフォルトの暗号化プロバイダになりました。Javaクラスをインストールして FIPSを有効にする手順は、OJVMのドキュメントのFIPSのセクションで説明されています。

より長いパスワードのサポート

JDBCは、透過的に、つまりAPIの変更なしで、最大1024バイトの長いパスワードをサポートするようになりました。

OCI IAMおよびAzure ADのトークンベースの認証

Oracle Cloud Infrastructure（OCI）Identity and Access Management（IAM）やAzure Active Directory（AD）などのクラウド・ディレクトリ・サービスに対するOracle JDBCサポート。

OCI IAMのトークンベースの認証

JDBC-Thinドライバは、Oracle Cloud Infrastructure（OCI）Identity and Access Management（IAM）のサポートを強化します。
手順は以下のとおりです

1) Javaアプリケーションは、以下で説明するいずれかの方法でデータベース・トークンを提供します。
2) データベースは、認証サービスから取得した公開鍵を使用してトークンを検証し、ユーザー認証を実行します。
3) Javaアプリケーションは、トークンに埋め込まれた公開鍵とペアになっている秘密鍵を所有していることを証明するヘッダーを送信します。
4) トークンと署名の両方が有効で、IAMユーザーとデータベース・ユーザーの間にマッピングが存在する場合、データベースへのアクセスがJDBCアプリケーションに許可されます。

この認証方式は次のように使用できます。ファイル・システム上のデータベース・トークンに使用する、oracle.jdbc.accessToken接続プロパティを使用してトークン値を渡す、OracleConnectionBuilder.accesTokenメソッドを使用して認証サービスからトークンを取得する、OracleDataSourceクラスのOracleCommonDataSource.setTokenSupplier(AccessToken accessTokenのサプライヤ関数を使用する。

詳しくは、Oracle JDBCドキュメントのクライアント側のセキュリティの章を参照してください。

Azure ADのトークンベースの認証

JDBC-Thinドライバは、Azure Active DirectoryのOAuth2 アクセス・トークンをサポートします。手順は以下のとおりです

1. Javaアプリケーションは、以下で説明するいずれかの方法でデータベース・トークンを提供します。
2. データベースは、認証サービスから取得した公開鍵を使用してトークンを検証し、ユーザー認証を実行します。
3. この認証方式は次のように使用できます。ファイル・システム上のデータベース・トークンに使用する、oracle.jdbc.accessToken接続プロパティを使用してトークン値を渡す、OracleConnectionBuilder.accesTokenメソッドを使用して認証サービスからトークンを取得する、OracleDataSourceクラスのOracleCommonDataSource.setTokenSupplier(AccessToken accessTokenのサプライヤ関数を使用する。

詳しくは、Oracle JDBCドキュメントのクライアント側のセキュリティの章を参照してください。

RADIUSチャレンジ/レスポンス認証（2FAとも呼ばれる）

1) Javaアプリケーションは、ユーザー名とパスワードを使用して最初のレベルの認証を実行します。

2) RADIUSサーバーはJavaアプリケーションにチャレンジを送信します。

3) Javaアプリケーションはハンドラを使用してチャレンジに応答します。

ハンドラはoracle.net.radius_challenge_response_handler接続プロパティまたは ConnectionBuilder.radiusChallengeResponseHandlerメソッドのいずれかを使用して構成されます。

コード・サンプルを含め、詳しくは、Oracle JDBCドキュメントのセクション9.9.4を参照してください。

Kerberosの機能拡張

このリリースでは、CredentialCache内のチケット許可チケットの要件を削除するか、KerberosLoginModuleをインスタンス化することにより、Kerberos認証が簡素化されました。

Kerberosプリンシパルとパスワードのプロパティの構成

Kerberosプリンシパルを使用してOracle Databaseに接続する場合
接続文字列内でPASSWORD_AUTHパラメータをKERBEROS5に設定します。oracle.jdbc.passwordAuthentication接続プロパティを使用して、PASSWORD_AUTHをKERBEROS5に設定することもできます。JDBC Thinドライバはアプリケーションの KerberosLoginModuleを初期化し、これによりKerberos認証が簡素化されます。

次のコード例を参照してください。

https://gist.github.com/Kuassim/3a628317a4501a0004b5e21fde829696

JAAS構成を使用するKerberos認証

デフォルトでは、JDBC-ThinドライバはOracle JDKにバンドルされているデフォルトのKerberosログイン・モジュール（com.sun.security.auth.module.Krb5LoginModule）を使用します。ただし代わりにJAAS構成を使用することも選択できます。

次のコード例を参照してください。

https://gist.github.com/Kuassim/fe6774588556f1f443334de57b408a99

Oracle JVMによるHTTPおよびTCPのサポート

データベースに組み込まれたJVM（OJVMとも呼ばれる）で、その他のOS呼び出しを無効化したままHTTPおよびTCP操作を有効化または無効化できるようになりました。

詳しくは、マルチテナント環境のデータベース・セキュリティを参照してください。

可用性

透過的アプリケーション・コンティニュイティ（TAC）により、Javaアプリケーションの高可用性は可能な限り透過的になっています。

透過的アプリケーション・コンティニュイティの機能強化

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）は、Javaアプリケーションからデータベース・インスタンスまたはネットワークの障害を隠す、Oracleデータベースの高可用性およびゼロダウンタイム機能です。これらの機能は、RDBMSサーバー、JDBCドライバ、およびUCPにまたがっています。ACおよびTACは、各リリースにおいて、ほとんどの設定をデータベース・クライアントおよびアプリケーションからRDBMSサーバーにプッシュします。

このリリースでは、JDBCドライバは再開可能なカーソルをサポートします。これらはセッションの長時間実行カーソルであり、複数のトランザクションをまたいで開いた状態を保ちます。再開可能なカーソルを使用すると、TACは暗黙的に、より頻繁にアプリケーション・リクエスト境界を確立し、TACの適用範囲をより広範なものにします。

全体像と詳細を把握するには、Javaのアプリケーション・コンティニュイティに関する章全体をお読みください。

まとめ

結論は次のようになります。この技術概要を通じて、容易な開発、クラウド・コンピューティング、マルチクラウド、パフォーマンスとスケーラビリティ、ミッション・クリティカルなデプロイメント、セキュリティ、可用性の領域における、Oracle Database 23ai（23.4.0.24.05）の新しいJava機能の全体的な概要を説明しました。これらの機能は、最新のJavaアプリケーションの設計とデプロイメントに間違いなく役立ちます。

@kmensah, #javaOracleDB, http://oracle.com/jdbc

ORACLE

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