Oracle Database 12c Release 2で提供されている JavaおよびJavaScript開発者向け機能

Oracle Cloud およびオンプレミス

Oracle ホワイト・ペーパー | 2017 年 4 月

免責事項

下記事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。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下さい。オラクルの製品に関して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。

概要

Web アプリケーションは、モノリシックなアーキテクチャから、サービスやマイクロサービスに基づいた俊敏性とスケーラビリティに優れたアーキテクチャへと進化しています。プログラミング・モデルやプログラミング・スタイルは、手続き型から関数型やリアクティブ型に進化しています。データ・モデルと永続化要件も、単一モデルからポリグロット永続化モデルへと進化しています。これらの進化に伴って、パフォーマンス、マルチテナント、高可用性、スケーラビリティ、セキュリティなどに関してデータベース・エンジンに新たな要件が生じています。Oracle Database 12c Release 2 は、リリース 12.1[1]の強力な機能の上に構築されており、これらの要件に対処する豊富な機能を備えています。このホワイト・ペーパーでは、Java 開発者と JavaScript[2]開発者に対して、個々のアプリケーションでこれらの新機能を利用する方法を概説することを目的としています。

クラウド・ファースト

Oracle Database 12c Release 2 は、Oracle Cloud[3]で最初にリリースされており、さまざまな開発要件やデプロイメント要件に対応しています。Oracle JDBC ドライバと UCP（Oracle Java 接続プール）を使用して、オンプレミスとクラウド・サービスの Java アプリケーションと IDE をさまざまな Oracle Database Cloud Service にシームレスに接続できます。詳しくは、以下のページを参照してください。

http://www.oracle.com/technetwork/jp/database/application-development/jdbc/overview/jdbc-eecloud-3089380-ja.html
http://www.oracle.com/technetwork/jp/database/application-development/jdbc/index-3093936-ja.html
http://www.oracle.com/technetwork/jp/database/application-development/jdbc/overview/default-3396167-ja.html

最新のJava標準のサポート

Java SE 8 では、数ある機能の中でも特に、関数プログラミングの基盤としてのラムダ式、java.util.streams パッケージ、非同期処理をサポートする java.util.concurrent.ompletableFuture、強化されたセキュリティなどが提供されています。Java 8 のすべての新機能の概要については、http://www.oracle.com/technetwork/jp/java/javase/8-whats-new-2157071-ja.html を参照してください。

Oracle Database 12c Release 2 では、JDBC ドライバ、Java 接続プール（UCP）、および組込み JVM（Oracle JVM）のすべてが Java SE 8 および JDBC 4.2 をサポートしています。

JavaScriptのサポート

また、Java SE 8 では、パフォーマンスと機能を強化するために、Rhino の置換えとして Nashorn JavaScript エンジンが提供されています。Java 開発者および JavaScript 開発者は、次の簡単な手順

[1] http://www.oracle.com/technetwork/database/application-development/12cdb-java-perf-scal-ha-security-1963442.pdf
[2] プレーンな JavaScript。Node.js については、Oracle OTN の Node.js Developer Center を参照してください。
@http://www.oracle.com/technetwork/database/database-technologies/scripting-languages/node_js/oracle-node-js-2399407.html
[3] https://cloud.oracle.com/database

で Nashorn を使用できます。

1. スクリプト・エンジン・マネージャをインスタンス化します。

2. JavaScript エンジンを作成します。

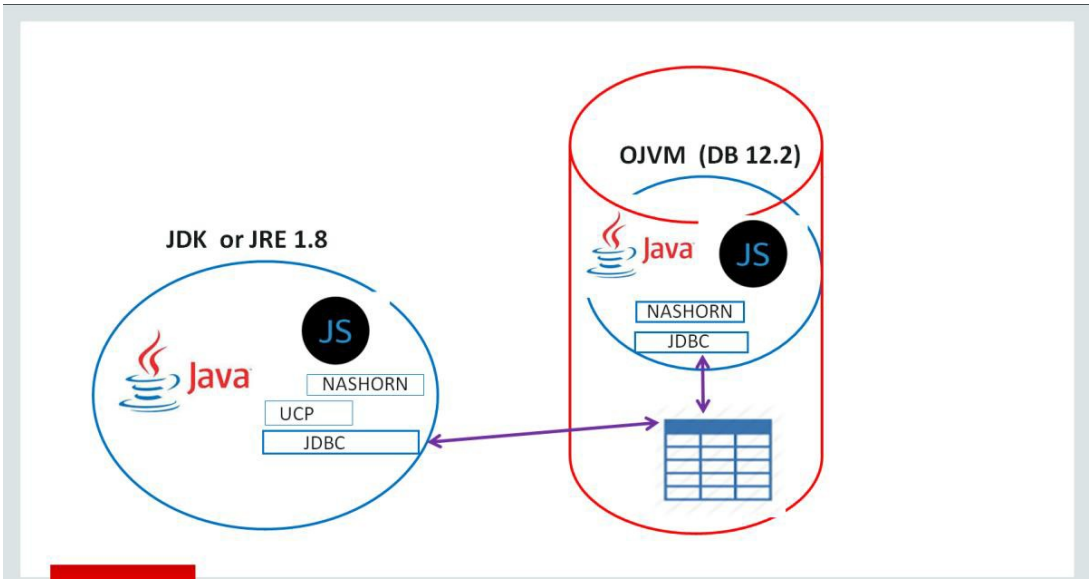
3. リソース・ストリーム・リーダーをエンジンの eval メソッドに引数として渡します。

以下のコード部分は、これらの手順に対応しています。

```
import javax.script.*;
import java.net.*;
import java.io.*;
...
// create a script engine manager ScriptEngineManager
factory = new ScriptEngineManager();
// create a JavaScript engine
ScriptEngine engine =
    factory.getEngineByName("javascript");
// create schema resource URL
URL url = Thread.currentThread()
    .getContextClassLoader().getResource("hello.js");
engine.eval(new InputStreamReader(url.openStream()));
```

JavaScriptからのデータベース・アクセス

JVM（JDK、JRE、および Oracle JVM）で JavaScript を実行すると、Java と JavaScript 間でシームレスな相互運用性を実現できます。たとえば、JavaScript に標準のデータベース・アクセス API がない場合、JavaScript から JDBC 呼出しを実行できます。以下の図は、Java と JavaScript による、JDBC を使用した、Oracle データベース外部の Java クライアントと JavaScript クライアントおよび Oracle データベース内部の Java と JavaScript からの表アクセスを示しています。



データベースでのJavaScript

Oracle database 12.2 の Oracle JVM では Nashorn エンジンがサポートされているため、データバインドの JavaScript 関数またはプロシージャをデータベースで直接実行できます。データ近くで実行できるため、パフォーマンスが向上し（ネットワーク待機時間が発生しない）、スキル、ライブラリ、コードを再利用できます。前述したように、Nashorn によって、JavaScript から JDBC 呼出しを実行してデータにアクセスできます。

以下の JavaScript コードは、データベース組込みの JVM（Oracle JVM）で動作します。

```
var selectQuery = function(id)
{
   var Driver = Packages.oracle.jdbc.OracleDriver;
   var oracleDriver = new Driver();
   var url = "jdbc:default:connection:";
   var query = "";
   var output = "";

   if(id == 'all') {
 query ="SELECT a.data FROM employees a";
   } else {
 query ="SELECT a.data FROM employees a WHERE a.data.EmpId=" + id;
   }
   var connection = oracleDriver.defaultConnection();

   // Prepare statement
 var preparedStatement = connection.prepareStatement(query);

   // execute Query
 var resultSet = preparedStatement.executeQuery();

   // display results
 while(resultSet.next()) {
  output = output + resultSet.getString(1) + " ";
 }

   // cleanup
    resultSet.close();
    preparedStatement.close();
    connection.close();
    return output;
}
```

実際に、Oracle データベースでどのように JavaScript を実行するのでしょうか。

1. まず、loadjavaユーティリティを使用してデータベース・スキーマにJavaScriptコードをJavaリソースとしてロードします。
2. 次に、次の3つのいずれかの呼出しを使用します。

   (i) SQL> でSQLまたはPL/SQLからdbms_javascript.run()を呼び出す

   (ii) データベース内でJavaからDbmsJavaScrpt.Run()を呼び出す

   (iii) javax.script APIを使用する

各方法の詳細については、ブログ記事（http://db360.blogspot.com/2016/11/javascript-in-oracle-database-12c.html ）を参照してください。

GitHugb の oracle マスター・リポジトリに JavaScript/Nashorn のコード・サンプルを掲載する予定です。

https://github.com/oracle/oracle-db-examples/tree/master/javascript

パフォーマンスとスケーラビリティ

このリリースでは、JDBC、UCP、Oracle JVM によって、Java アプリケーションおよび JavaScript アプリケーションで新しいパフォーマンス拡張機能とスケーラビリティ拡張機能を利用できます。

Oracle JVM

Java、JRuby、Jython、Closure、Scala、JavaScript などの PL/SQL 言語および JVM 言語をデータベースで実行する理由は、パフォーマンスです。このリリースでは、パフォーマンスを向上するために、組込みの JVM（Oracle JVM）が強化されており、JIT とメモリ管理の効率が改善されています。私の友人である Marcelo Ochoa が DB 12.2 の Oracle JVM をテストし、ブログ記事（http://marceloochoa.blogspot.com.ar/2016/11/12cr2- versus-12cr1-scan-time-using-java.html ）にリリース 12.2 で Oracle JVM のパフォーマンスが向上していることを紹介しています。

JDBCとUCP

Oracle JDBC ドライバが、転送時のデータ圧縮をサポートするようになったため、WAN 経由のクライアント/サーバー通信の応答が向上しています。この機能を有効にするには、以下のコード部分に示すように oracle.net.networkCompression プロパティを設定します。

```
// Enabling Network Compression
   prop.setProperty("oracle.net.networkCompression","on");
// Optional configuration for setting the client compression threshold.
   prop.setProperty("oracle.net.networkCompressionThreshold","1024");
   ds.setConnectionProperties(prop);
   ds.setURL(url);
   Connection conn = ds.getConnection();
```

Oracle Universal Connection Pool（UCP）は、待機時間なしの多次元 KD ツリー検索 [4]を用いて再設計されており、パフォーマンスを低下させずに 1 秒あたり 30 万を超えるプール処理を維持できます。

また、DB 12.2 の UCP では、接続状態チェックの頻度を構成できるため、接続チェックアウト時の体系的な状態チェックのオーバーヘッドが軽減されます。

以下に、Java API を示します。

```
public void setSecondsToTrustIdleConnection(int secondsToTrustIdleConnection)
                                throws java.sql.SQLException
```

secondsToTrustIdleConnection を 30 に設定すると、接続は以降の 30 秒間有効であると見なされます。

[4] 関連する JavaOne セッションを参照してください。 https://www.youtube.com/watch?v=PEwBrjkJfrk&t=352s&list=PLPIzp-E1msrYicmovyeuOABO4HxVPlhEA&index=59

シャード・データベースのJava共有プール

Oracle Database 12c Release 2 ではシャード・データベース・アーキテクチャが導入されており、数百ものデータベースに渡って水平方向でデータをパーティション化できるため、スケーラビリティに制限がなくなります。シャード・ディレクタが、シャードとシャーディング・キーのマップを保持します。Oracle JDBC と UCP の両方が、新しい API によってシャード・データベース・アーキテクチャをサポートしています。以下の Java コード部分に示すように、この新しい API を使用してシャーディング・キーとスーパー・シャーディング・キーを作成します。

```
// A sharding key make of 2 sub-keys
   OracleShardingKey shardingKey =
                    dataSource.createShardingKeyBuilder()
                   .subkey("Customer_EMAIL",oracle.jdbc.OracleType.VARCHAR2)
                   .subkey("1234", oracle.jdbc.OracleTypes.NUMBER)
                   .build();
```

シャード対応の Java アプリケーションで、特定のシャードに対して接続を要求できるようになりました。

```
 // Request a connection to a shard
    Connection conn =
                    pds.createConnectionBuilder()
                       .shardingKey(shardingKey)
                       .superShardingKey (superShardingKey)
                       .build();
```

UCP が Oracle Database シャード・ディレクタからシャーディング・トポロジを取得し、適切なシャードに接続要求を直接ルーティングするため、追加のホップ（シャード・ディレクタへの転送）が回避されます。

マルチテナント・データベースのJava共有プール

このリリースでは、UCP が共有プールを提供し、プラガブル・データベース（PDB）にアタッチされている空き接続を、他のプラガブル・データベースで使用できるように再割当てします。1 つの接続プールを、それぞれ固有の PDB を持つ多数のテナントが共有できるようになりました。数千ものテナントに、それぞれ固有のデータソースまたはデータベース・サービスを設定し、大規模にスケーリングできるようになりました。以下に示す Java コード部分では、各テナントがそれぞれのデータベース・サービスを介して接続されています。

```
PoolDataSource multiTenantDS = PoolDataSourceFactory.getPoolDataSource();
multiTenantDS.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
// Set the common user for the CDB and point to the root service
multiTenantDS.setUser("c##1");
multiTenantDS.setPassword("password");
multiTenantDS.setURL("jdbc:oracle:thin…");
// Create a connection to Tenant-1
Connection pdb1_conn = multiTenantDS.createConnectionBuilder()
                        .serviceName("pdb1")
                        .build();

// Get the password enabled role for tenant-2, if configured on the database
Properties pdb2Roles = new Properties();
pdb2Roles.put("pdb2-role", "pdb2-password");

// Create a connection to Tenant-2 and apply the tenant specific PDB roles.
Connection pdb2_conn = multiTenantDS.createConnectionBuilder()
                         .serviceName("pdb2")
                         .pdbRoles(pdb2Roles)
                         .build();
```

PL/SQLコールバック・インタフェース

アプリケーションでは、実行時にセッション状態に NLS_CURRENCY や CURRENT_SCHEMA などの新しい値を割り当てることが必要になる場合があります。新しい PL/SQL コールバック・メカニズムにより、データベースでラウンドトリップを発生させずに、これらの状態に新しい値を設定できます。Java アプリケーションはこのようなコールバックを RDBMS サーバーに実装し、JDBC 呼出しによってコールバックを登録する必要があります。実装例については、Oracle JDBC のガイドを参照してください。

Oracle更新バッチの非推奨化

Oracle 更新バッチは Oracle Database リリース 12.1 で非推奨になっており、このリリースでサポート対象外になっています。そのため、バッチ・サイズの設定が無効になっており、デフォルトで 1 に設定されます。Oracle 更新バッチ API を引き続き使用するアプリケーションは、パフォーマンスが低下します（一度に 1 行ずつになります）。代わりに、標準の JDBC バッチ API を使用することをお勧めします。

高可用性 - 継続的運用

JDBC -UCP

- 高可用性と継続的運用は、Web アプリケーションとクラウド・サービスにおいて重要です。このリリースでは、高可用性に対応するように JDBC と UCP のサポートが拡張されています。新しい機能拡張には、次のものが含まれます。
- `SimpleFan.jar` で FAN イベント UP と DOWN、およびロードバランシング・アドバイザリを処理するように Java API を拡張。これらの API を使用すると、サード・パーティの Java コンテナまたはフレームワークをおもな FAN HA イベントにサブスクライブして、サービス/ノードの障害や追加を管理できます。
- Oracle JDBC ドライバにより計画メンテナンスをサポート（DOWN イベントの reason=User）。ドライバは FAN DOWN イベントを受信すると、メンテナンスがスケジュールされているノードまたはサービスにアタッチされているすべての接続を"*安全な場所*"で終了させます。Java アプリケーションが接続オブジェクトに対してメソッド `isValid()`、`isUsable()`、または `isClosed()`を起動する必要があります。あるいは、アプリケーションが SQL 呼出し `SELECT 1 FROM DUAL /*+ CLIENT_CONNECTION_VALIDATION */`を実行することもできます。詳細については、JDBC のドキュメントを参照してください。
- トランザクション・ガード（TG）を XA データソースに拡張。Java 用のトランザクション・ガードは、アプリケーションと RDBMS 間の通信が中断された場合に COMMIT 文の結果を確定できるように、Oracle Database 12c Release 1 で導入された HA API です。このリリースでは、1 フェーズ・コミットの最適化、読取り専用の最適化、および昇格可能な XA により、トランザクション・ガードが XA トランザクションに拡張されています。
- アプリケーション・コンティニュイティ（AC）を XA データソースに拡張。Java 用のアプリケーション・コンティニュイティは、データベース、サービス、ノードの計画外停止が発生した場合にサービスを継続するために、Oracle Database 12c Release 1 で導入された HA ソリューションです。JDBC 再実行データソースを使用するとともに、データベース・サービスを適切に構成する必要があります。このリリースでは、AC は XA データソースもサポートしています。
- シャード・トポロジを取得してシャード・ディレクタとして機能することで、シャード・ディレクタが停止した場合でも、UCP はシャード対応の Java アプリケーションが処理を実行して特定のシャードに対して接続を要求できるようにします。

ベスト・プラクティス

高可用性を実現するためのベスト・プラクティスには、以下のものがあります。

a) **計画メンテナンスの場合：**Oracle Database HA 構成（Oracle RAC、ADG、GDS、マルチテナント）を使用する。シームレスな HA をサポートする Oracle ドライバまたは Oracle 接続プールを使用する。位置の透過性を実現するサービスを使用する。最新の機能を利用して接続文字列を適切に構成する。ドライバまたはプールによってドレインがトリガーされるように FAN イベントを自動構成および自動有効化（12.2 の機能）する。`drain_timeout`（秒単位）や `stopoption`（`immediate`、`transactional`）などのサービス属性を適切に設定する。

b) **計画外停止の場合：**上記の計画メンテナンス場合のベスト・プラクティスに従います。さらに、次のベスト・プラクティスに従います：Application Continuity を構成する。Oracle 接続プール（UCP）を使用する、またはデータベース作業ユニットにリクエスト境界を追加する（リクエスト）。使用されていない接続をプールに戻す。`ORAChk` を実行して JDBC 具体クラスがアプリケーションに使用されているかどうかを特定する。アプリケーションによって使用されるユーザー・スキーマに対して、seq.nextval、sysdate、systimestamp などの可変値を維持する権限を付与する。

セキュリティ

以下のセキュリティ強化機能は、このリリースの新機能です。

DB 12.2 の JDBC ドライバは、デフォルトで SSL v1.2（TLS v1.2）をサポートしています。

暗号化とセキュリティ強化を適用するには、Oracle Database Exadata Express Cloud Service への Java 接続に Java キーストア（JKS）ファイルが必要です。

Oracle JVM で Nashorn を使用してデータベースで JavaScript を実行するには、データベース・スキーマに DBJAVASCRIPT ロールを付与する必要があります。また、`javax.script` API（JSR223）の使用は、権限が最小限となるサンドボックス・モデルのみに制限されています。

Web サービス・コールアウト（後述）では、データベース・スキーマに OJVMWCU ロールを付与する必要があります。新しい Web サービス・コールアウト・ユーティリティ（後述）は、HTTP 基本認証に加えて SSL ベースの Web サービスもサポートしています

管理性、操作性、診断機能

このリリースの新機能には、次のものがあります。Oracle JVM による Web サービス・コールアウトのサポート（REST と SOAP）。機能レベルでの実行時ロギング（JDBC）。Oracle JVM による長い識別子のサポート。JDBC による PL/SQL ブールのサポート。JDBC による機能レベルのデバッグのサポート。Oracle JVM で実行中の Java コードをデバッグするための機能拡張。データベース常駐接続プール（DRCP）の機能拡張。これらの機能拡張について、以降の項でいくつか簡単に説明します。

Oracle JVM - Webサービス・コールアウト

リモート/外部の SOAP Web サービスを起動してデータを取得する機能は広く使用されており、JPublisher によって可能になっていました。DB 12.1 でこの機能がサポート対象外となり、DB 12.2 では新しいユーティリティが使用可能になりました。また、RESTful Web サービス・コールアウトのサポートが追加されています。

新しい Web サービス・コールアウト・ユーティリティは、WSDL（SOAP Web サービス）と WADL（Restful Web サービス）および追加のパラメータを受け入れ、Web サービスのクライアント・プロキシを取得してデータベースにロードし、SQL 呼出しおよび PL/SQL 呼出しのラッパーを生成します。

詳細については、ドキュメント（http://docs.oracle.com/database/122/JJDEV/database-as-web-service-consumer.htm#JJDEV13475 ）を参照してください。

Oracle JVM - 長い識別子とデバッガ

Oracle JVM が長い識別子をサポートするようになり、SQL 識別子の最大長が 128 バイトになりました。

JDWP インタフェースを使用してデータベースで Java をデバッグする場合、接続へのデバッグ・セッションのアタッチ、ブレーク・ポイントの設定またはクリア、Java コードのステップ実行、変数値の設定および変更、式の評価、ウォッチ・ポイントの設定またはクリアが可能です。

JDBC – UCP

また、このリリースでは、データベース常駐接続プール（DRCP）の新しい拡張機能を利用でき、これらの拡張機能として、複数プロパティのラベル付け、パフォーマンス監視とチューニング用の新しい統計ビューと AWR レポート、進行中のトランザクションを含むプールされたサーバー用の新しい MAX_TXN_THINK_TIME プロパティ、セッション状態の修正のための PL/SQL コールバック、プロキシ・セッションの共有機能があります。

JDBC で、OracleDiagnosabilityMBean を使用した機能レベルのデバッグがサポートされるようになりました。これにより、選択された機能のロギングを実行時に有効化および無効化できます。たとえば、高速接続フェイルオーバー機能のロギングを有効化し、ロードバランシング機能を無効化できます。デフォルトでは、実行時ロギングはすべての機能に対して有効になります。詳しくは、http://docs.oracle.com/database/122/JJDBC/JDBC-diagnosability.htm#JJDBC-GUID-193E4D19-40D4-40D8-89C5-1A792E15F538 を参照してください。

以下のコード部分は、JDBC による PL/SQL ブール（パラメータ・バインド変数として）のサポートを示しています。

```
cstmt.registerOutParameter(1, OracleTypes.PLSQL_BOOLEAN);
  // Execute the callable statement
   cstmt.execute();
   boolean   TF = cstmt.getBoolean(1);
```

結論

このホワイト・ペーパーでは、Oracle Database 12c Release 2 で Java 開発者および JavaScript 開発者が得られるおもな利点について説明しました。クラウド内のデータベースでの Java 接続、JDBC、UCP、および Oracle JVM での最新の Java 標準のサポート、JavaScript と Nashorn による JDBC を使用したデータベース・アクセス、Oracle JVM と Nashorn によるデータベースでの JavaScript の実行、JDBC、UCP、および Oracle JVM でのパフォーマンスとスケーラビリティの強化、およびセキュリティ、管理性、操作性の強化について説明しました。

クラウドおよびオンプレミスで Oracle Database 12c Release 2 を使用すると、Java 開発者および JavaScript 開発者は、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、管理性、診断機能を確保しながら、関数型プログラミング・モデルやリアクティブ・プログラミング・モデルを使用して、中間層コンポーネントやデータベース組込みコンポーネントを備えた最新の Web アプリケーションを設計およびデプロイできます。

Oracle Corporation, World 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**
ファクシミリ：**+1.650.506.7200**

CONNECT WITH US

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

Integrated Cloud Applications & Platform Services