Oracle® Database
2 日で Java 開発者ガイド
リリース 19c

F16966-01(原本部品番号:E96442-01)

2019年2月

タイトルおよび著作権情報

Oracle Database 2日でJava開発者ガイド, リリース19c

F16966-01

原本著者: Tanmay Choudhury

原本協力著者: Tulika Das

原本協力者: Kuassi Mensah、Nirmala Sundarappa

な契約が定められている場合を除いて、オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

目次

- 表一覧
- タイトルおよび著作権情報
- はじめに
    - 対象読者
    - 関連ドキュメント
    - 表記規則
- 1 この本の目的と目標
    - アプリケーションのアーキテクチャ
    - コンポーネントおよびリポジトリ
    - 1日目の目標とタスク
    - 2日目の目標とタスク
- 2 JDBC、UCPおよびデータベースにおけるJavaの簡単な紹介
    - Java Database Connectivityドライバ(JDBC)
    - ユニバーサル接続プール
    - データベースにおけるJava(OJVM)
- 3 HR Webアプリケーションの概要
    - HR Webアプリケーションの機能
- 4 アプリケーション開発の開始
    - インストールする必要があるもの
        - Oracle Database 12c リリース2 (12.2)
            - JDBCアプリケーションのHRスキーマのロック解除
        - JDK 8
        - JDeveloper IDE
        - J2SEまたはJDK
        - 統合開発環境
        - Webサーバー
    - Oracle Database 12c リリース2 (12.2)のインストールの検証
    - Githubリポジトリの詳細
    - JDeveloperでのアプリケーションのインポート
    - JDeveloperでのアプリケーションのコンパイル
    - Mavenを使用したコンパイルと、任意のJava EEコンテナでのアプリケーションの実行
- 5 全従業員をリスト
    - 従業員のJava Beanエンティティの作成
    - JDBC接続のためのJava Beanインタフェースの作成
    - JDBC接続のためのJava Bean実装の作成
    - リクエストを処理するサーブレットの作成
    - 結果を表示するHTMLページの作成
    - CSSファイルの作成
- 6 従業員IDによる検索
    - Employee Java Bean
    - リクエストを処理するコードのサーブレットへの追加
    - 従業員IDによる検索のための新規HTMLの作成

- 7 従業員レコードの更新
    - EmployeeBean.javaでの新しいメソッドgetEmployeeByFn(String)の宣言
    - 新しいメソッドupdateEmployee(Employee)の宣言
    - 従業員名による検索のための新しいメソッドgetEmployeebyFn()の実装
    - 新しいメソッドupdateEmployee(Employee)の実装
    - サーブレット(WebController.java)へのコードの追加
    - 新しいHTML(incrementSalary.html)の作成
    - Tomcatでのログイン・ユーザーの作成
- 8 ベスト・プラクティス
- 9 トラブルシューティングおよびデバッグ
- 索引

表一覧

- 1-1　Webアプリケーションのアーキテクチャ
- 1-2　アプリケーションに必要なコンポーネント
- 4-1　Githubリポジトリの詳細

はじめに

ここでは、*Oracle Database 2*日で*Java*開発者ガイドの対象読者と表記規則について説明します。また、詳細情報が記載されているオラクル社の関連ドキュメントのリストも示します。

対象読者

このマニュアルは、Javaを使用してOracle Databaseデータへのアクセスおよび変更を行うアプリケーション開発者を対象としています。このマニュアルでは、これらのタスクを簡単なJava Database Connectivity (JDBC)アプリケーションを使用して実行する方法を説明します。このマニュアルでは、Oracle JDeveloper統合開発環境(IDE)を使用してアプリケーションを作成します。このマニュアルは、Javaプログラミングに関心のあるすべての読者を対象にしていますが、次に関するいくらかの予備知識があることを前提にしています。

- Java
- Oracle PL/SQL
- Oracleデータベース

関連ドキュメント

詳細は、Oracle Databaseのドキュメント・セットに含まれる次のドキュメントを参照してください。

- 『*Oracle Database JDBC*開発者ガイド』
- 『*Oracle Database Java*開発者ガイド』
- *Oracle Universal Connection Pool*開発者ガイド

表記規則

このドキュメントでは次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック	イタリックは、ユーザーが特定の値を指定するプレースホルダー変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

1 このマニュアルの目標と目的

Javaは、様々なエンタープライズ・ソリューションを構築するために使用されており、開発者の間で一般的な言語です。

このガイドを通じて、Javaアプリケーションを作成するために使用されるあらゆるJava製品を理解できます。MVCデザイン・パターン、Oracle JDBC Thinドライバ、ユニバーサル接続プール(UCP)およびデータベースにおけるJava (埋込みOJVMを使用)を使用してJava Webアプリケーションをモデリングする方法について学習します。

次のいくつかの章では、Java Webアプリケーション「HR Webアプリケーション」を作成します。このアプリケーションは、AnyCo CorporationのHRチームによる、特定の従業員またはすべての従業員の詳細の参照または変更、従業員の削除、またはすべての従業員への昇給の適用を支援します。

アプリケーションには2人のユーザーがいます

hrstaff

hradmin

各ユーザーには、異なるロールと権限のセットがあります。

この章の項目は次のとおりです。

Webアプリケーションのアーキテクチャ

アプリケーションのコンポーネント

1日目のタスク

2日目のタスク

アプリケーションのアーキテクチャ

Webアプリケーションのアーキテクチャ

HR Webアプリケーションは、MVC (Model、View、Controller)アーキテクチャと最新のツールおよびテクノロジを使用します。モデル・ビュー・コントローラ(MVC)は、使用しやすいデザイン・パターンです。これはWebアプリケーションを3つの単純な部分(モデル、ビュー、コントローラ)に分割します。

モデルには、Webアプリケーションが操作するデータまたは情報が格納されます。ユーザー・インタフェースに関する情報は含まれません。

ビューには、ユーザー・インタフェース(UI)のすべての要素が含まれます。これにはボタン、表示ボックス、リンク、入力ボックスなどが含まれます。

コントローラは**モデル**と**ビュー**を接続します。

ユーザーには、アプリケーションにログインした後にHTMLページのJSPページになる可能性のあるインタフェース(ビュー)が表示されます。コントローラ(Javaサーブレット)は、ログイン中、またはその他のフロー中に正しいビューをレンダリングします。データまたはデータの更新を要求すると、コントローラは、表またはビューに関するデータを表すモデルを起動し、データをレンダリングします。モデルは、通常、Oracle Databaseまたは他のデータベースに格納されているユーザー・データを表します。

コントローラはこのデータを**ビューに**渡し、表示可能な形式でユーザーに表示します。

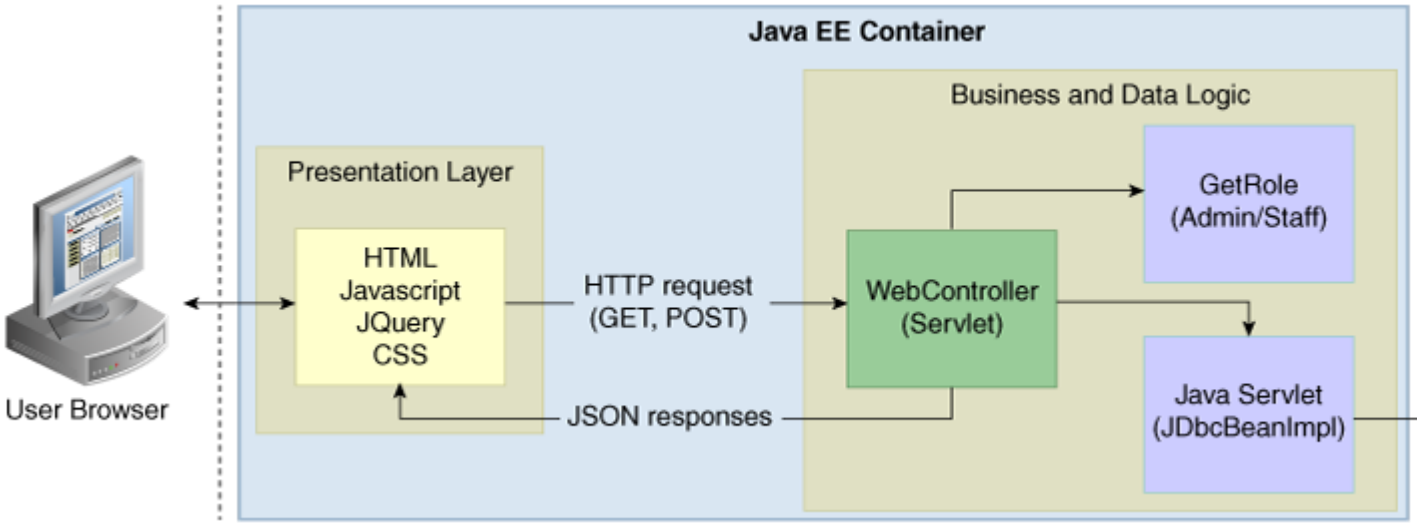
表1-1 Webアプリケーションのアーキテクチャ

名称	使用されているテクノロジ	説明
モデル	Oracle Database、Java Beans	アプリケーションが操作する情報またはデータを表します。
ビュー	HTML、JavaScript、JQuery、CSS	エンド・ユーザーにモデルをレンダリングするユーザー・インタフェース。これには、ボタン、リンク、入力など、ユーザーに表示されるすべての要素が含まれます。
コントローラ	Java サーブレット	コントローラはユーザー・アクションを処理して応答します。ユーザー入力に基づいてフローを編成します。また、モデルとビューを接続してユーザーに出力をレンダリングします。

次に示すのは、Webアプリケーションへのリンクです。

HRスキーマと従業員を使って、Webアプリケーションのフローを理解します。

図1-1



コンポーネントおよびリポジトリ

次の表で、アプリケーションに必要なすべてのコンポーネントをリストして説明します。

表1-2 アプリケーションに必要なコンポーネント

パッケージ名	説明
src	ソース・ファイルが含まれます
ターゲット	クラス・ファイルが含まれます
src/main/java/com/oracle/jdbc/samples	-
/bean/JdbcBean.java	従業員の詳細を属性として定義します
/bean/JdbcBeanImpl.java	EmployeeBean の実装クラス

パッケージ名	説明
src/main/java/com/oracle/jdbc/samples	-
entity/Employee.java	すべての従業員の属性とそれに定義されてているデータ型で構成されます
/web/WebController.java	アプリケーション・フローを制御するサーブレット
/web/GetRole.java	アプリケーションの HRStaff ロールと HRAdmin ロールを作成します
src/main/resources	-
SalaryHikeSP.java	給与の増額を処理するためにデータベースにおける Java から呼び出される Java クラス
SalaryHikeSP.sql	給与範囲に基づいて従業員の給与を引き上げるプロシージャが記載された SQL ファイル
src/main/webapp	-
about.html	HR Web アプリケーションに関する詳細が含まれます
login.html	HR Web アプリケーションのログイン・ページが含まれます
login-failed.html	ログインが失敗した場合に表示するページ
index.html	HR Web アプリケーションのランディング・ページ
listAll.html	すべての従業員レコードを表示する HTML ページ
listByName.html	名前で従業員を検索したときに結果を表示する HTML ページ
listById.html	従業員 ID で従業員を検索したときに結果を表示する HTML ページ
incrementSalary.html	給与の増額後に結果を表示する HTML ページ
src/main/webapp	-
css/app.cs	HR Web アプリケーションで使用されるすべてのスタイルおよびフォントの詳細が含まれます
src/main/webapp	-
WEB-INF/web.xml	HR Web アプリケーションのコントローラ

1日目の目標とタスク

1日目の最後に、次のことができるようになります。

a.JDBC、UCPおよびデータベースにおけるJavaを理解し、単純なJavaプログラムを実行してこれらの製品を理解します。

b.すべての従業員の詳細をリストする機能"listAll"を実装します。

1 **JDBC、UCPおよびデータベースにおけるJavaの概要:** 製品、関連バイナリおよびパッケージについて、サンプル・コードを使用して理解します。

2 **HR Webアプリケーションの概要:** この章では、HRのWebアプリケーションを詳細に説明し、Webアプリケーションのフローと、アプリケーションの一部として作成するパッケージおよびファイルを紹介します。

3 **アプリケーション開発の開始:** アプリケーションを開発する前提条件と、環境を準備する方法を理解します。まず、クラウド

のOracle Database Serviceにサブスクライブするか、オンプレミスのOracle Database 12cリリース2を社内にインストールします。その後、アプリケーションを構築するIDEであるJDeveloperをインストールします。JDeveloperまたはTomcat Java EEコンテナに統合されているWeb Logic Serverを使用して、アプリケーションをデプロイおよび実行します。

この章では、アプリケーションの構築に役立つMavenなどの他のツールをダウンロードすることもできます。

4 **全従業員のリスト:** この章では、すべてのコンポーネントを組み合せ、Oracle Databaseに接続する初期機能を作成して、データベースから従業員の詳細情報を取得する方法について説明します。

2日目の目標とタスク

第II部では、第2日に終えるすべてのタスクについて説明します。ユニバーサル接続プール(UCP)と、データベースにおけるJava (OJVM)の使用方法を学びます。また、次の方法についても学習します。

1 従業員IDによる検索: この章では、「従業員IDによる検索」の機能を実装する方法について説明します。

2 従業員レコードの更新: この章では、従業員レコードを更新する方法について説明します。これは2段階のプロセスです。最初に、従業員名に基づいて従業員のレコードを検索します。必要な結果を取得すると、給与、ジョブID、名、姓およびその他の詳細を更新できます。

3 **従業員レコードの削除:** この章では、2段階のプロセスで従業員レコードを削除する方法について説明します。

4 **全従業員の給与の増額:** 表にリストされた従業員の給与について、「データベースにおけるJava」を使用して増額を指定する方法を学びます。

5 **アプリケーション・ユーザーの作成:** この章では、TomcatおよびJDeveloperで'hradmin'および'hrstaff'ユーザーを作成する方法について説明します。

6 **まとめ:** この章では、2日間で学んだことをすべてまとめます。また、アプリケーションの使用を拡張するための適切な参照情報とリンクも示します。

2 JDBC、UCPおよびデータベースにおけるJavaの簡単な紹介

Oracle Databaseは、データの格納、変更および使用に使用できるリレーショナル・データベースです。

Javaアプリケーションでは、リレーショナル・データベースのデータのアクセスおよび操作に、Java Database Connectivity(JDBC)標準が使用されます。

業界標準のアプリケーション・プログラミング・インタフェース(API)であるJDBCを使用すると、JavaからSQLを使用してRDBMSにアクセスできます。JDBCは、JDBCエスケープ標準のエントリ・レベルに準拠しています。ベンダー各社はそれぞれ独自の拡張を含めたJDBC仕様を実装しています。

ユニバーサル接続プール(UCP)は、接続を再利用するためにデータベース接続オブジェクトをキャッシュするために使用する接続プールです。これにより、パフォーマンスが向上します。

データベースにおけるJava(OJVM)により、Javaデータ・ロジックを使用してSQL操作をグループ化し、インプレース処理用にデータベースにロードできます。

関連項目:

http://www.oracle.com/technetwork/java/overview-141217.html

この章では、Oracle Database 12cリリース2(12.2)とともにJDBCドライバ、ユニバーサル接続プール(UCP)およびデータベースにおけるJava(OJVM)について説明します

- Java Database Connectivityドライバ(JDBC)
- ユニバーサル接続プール(UCP)
- データベースにおけるJava(OJVM)

Java Database Connectivityドライバ(JDBC)

JDBCは、ユーザーがデータベースに接続してSQL文を実行し、データベースに問合せできるようにするデータベース・アクセス・プロトコルです。JDBCドライバは、最新のJDBC仕様に準拠して実装されています。Javaアプリケーションは、クラスパスにJDK8と互換性のあるojdbc8. jarが必要です。

コアJavaクラス・ライブラリには、JDBC APIのjava. sqlとjavax. sqlが用意されています

次の項では、JDBC標準に対するOracleサポートについて説明します。

- Oracle JDBC Thinドライバ
- Oracle JDBCパッケージ

Oracle JDBC Thinドライバ

ほとんどの場合、JDBC Thinドライバを使用することをお薦めします。JDBC Thinドライバは、適切なJava仮想マシンを搭載したシステムであれば、どのシステム上でも機能します。(JVM)Oracleが提供する他のクライアント・ドライバとして、JDBC Thinドライバ、Oracle Call Interface (OCI)ドライバ、サーバー側Thinドライバ、サーバー側内部ドライバなどがあります。

JDBC Thinドライバは、Pure JavaのType IVドライバです。JDBCドライバ・バージョン(ojdbc8.jar)には、JDK 8のサポー

トが含まれています。

JDBC Thinドライバは、データベースにアクセスするためにSQL*Netを使用してサーバーと通信します。

関連項目:

『*Oracle Database JDBC*開発者ガイド』

アクション・アイテム1: DB_URLを変更してデータベースを指すようにします。ヘルプが必要な場合は、Githubの DataSourceSample.javaを参照してください。

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.DatabaseMetaData;

import oracle.jdbc.pool.OracleDataSource;
import oracle.jdbc.OracleConnection;

public class DataSourceSample {
  // The recommended format of a connection URL is the long format with the
  // connection descriptor.
  // final static String DB_URL=
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)(PORT=1521)(PROTOCOL=tcp))(CONNECT_DATA=(SERVICE
_NAME=myorcldbservicename)))";

  final static String DB_URL=
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=localhost)(PORT=5521)(PROTOCOL=tcp))(CONNECT_DATA=(SERV
ICE_NAME=service name)))";
  final static String DB_USER = "username";
  final static String DB_PASSWORD = password;

  public static void main (String args[]) throws SQLException {

    OracleDataSource ods = new OracleDataSource();
    ods.setURL(DB_URL);
    ods.setUser(DB_USER);
    ods.setPassword(DB_PASSWORD);

    // With AutoCloseable, the connection is closed automatically.
    try (OracleConnection connection = (OracleConnection)
      ods.getConnection()) {
      // Get the JDBC driver name and version
      DatabaseMetaData dbmd = connection.getMetaData();
      System.out.println("Driver Name: " + dbmd.getDriverName());
      System.out.println("Driver Version: " +
        dbmd.getDriverVersion());
      System.out.println("Database Username is: " +
        connection.getUserName());
    }
  }
}
```

ユニバーサル接続プール

接続プールは、接続オブジェクトを再利用して、接続オブジェクトが作成される回数を減らすことで、パフォーマンスを向上させます。

Oracle Universal Connection Pool (UCP)は、Oracle Database構成との密接な統合を利用して、高可用性、スケーラビリティ、ロード・バランシングとともに接続プール機能を提供する、機能豊富なJava接続プールです。

UCPを使用するには、Javaアプリケーションまたはコンテナのクラスパスにucp.jarとojdbc8.jar (JDK8)が必要です。

関連項目:

*Oracle Universal Connection Pool*開発者ガイド

アクション・アイテム2: DB_URLを変更してデータベースを指すようにします。

```
Import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

public class UCPSample {
  // final static String DB_URL=
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)(PORT=1521)(PROTOCOL=tcp))(CONNECT_DATA=(SERVICE
_NAME=myorcldbservicename)))";

  final static String DB_USER = "username";
  final static String DB_PASSWORD = "pwd";
  final static String CONN_FACTORY_CLASS_NAME = "oracle.jdbc.pool.OracleDataSource";

  /*
   * The sample demonstrates UCP as client side connection pool.
   */
  public static void main(String args[]) throws Exception {
    // Get the PoolDataSource for UCP
    PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();

    // Set the connection factory first before all other properties
    pds.setConnectionFactoryClassName(CONN_FACTORY_CLASS_NAME);
    pds.setURL(DB_URL);
    pds.setUser(DB_USER);
    pds.setPassword(DB_PASSWORD);
    pds.setConnectionPoolName("JDBC_UCP_POOL");

    // Default is 0. Set the initial number of connections to be
    // created when UCP is started.
    pds.setInitialPoolSize(5);

    // Default is 0. Set the minimum number of connections
    // that is maintained by UCP at runtime.
    pds.setMinPoolSize(5);

    // Default is Integer.MAX_VALUE (2147483647). Set the maximum
    // number of connections allowed on the connection pool.
    pds.setMaxPoolSize(20);

    // Default is 30secs. Set the frequency in seconds to enforce
    // the timeout properties. Applies to
    // inactiveConnectionTimeout(int secs),
    // AbandonedConnectionTimeout(secs)&
    //TimeToLiveConnectionTimeout(int secs).
```

```
    // Range of valid values is 0 to Integer.MAX_VALUE.
    pds.setTimeoutCheckInterval(5);

    // Default is 0. Set the maximum time, in seconds, that a
    // connection remains available in the connection pool.
    pds.setInactiveConnectionTimeout(10);

    System.out.println("Available connections before checkout: "
        + pds.getAvailableConnectionsCount());
    System.out.println("Borrowed connections before checkout: "
        + pds.getBorrowedConnectionsCount());
    // Get the database connection from UCP.
    try (Connection conn = pds.getConnection()) {
      System.out.println("Available connections after checkout: "
          + pds.getAvailableConnectionsCount());
      System.out.println("Borrowed connections after checkout: "
          + pds.getBorrowedConnectionsCount());
      // Perform a database operation
      printEmployees(conn);
    } catch (SQLException e) {
      System.out.println("UCPSample - " + "SQLException occurred : "
          + e.getMessage());
    }
    System.out.println("Available connections after checkin: "
        + pds.getAvailableConnectionsCount());
    System.out.println("Borrowed connections after checkin: "
        + pds.getBorrowedConnectionsCount());
  }
 /*
  * Displays first_name and last_name from the employees table.
  */
  public static void printEmployees(Connection connection)
     throws SQLException {
    // Statement and ResultSet are AutoCloseable and closed
    // automatically.
    try (Statement statement = connection.createStatement()) {
      try (ResultSet resultSet = statement
          .executeQuery("select first_name, last_name from
            employees")) {
        System.out.println("FIRST_NAME" + "  " + "LAST_NAME");
        System.out.println("----------------------");
        while (resultSet.next())
          System.out.println(resultSet.getString(1) + " "
              + resultSet.getString(2) + " ");
      }
    }
  }
}
```

データベースにおけるJava(OJVM)

Oracle Databaseには、サーバーに常駐するJava仮想マシン(JVM)があります。それによって、サーバー上のOracle JVM内で実行中のJavaアプリケーションから、同じシステムおよび同じプロセスに存在するデータにアクセスできます。

データ処理が大きいアプリケーションには、データベースにおけるJavaをお薦めします。JVMは、基礎となるOracle RDBMSライブラリを直接使用する機能があり、JavaコードとSQLデータの間にネットワーク接続が必要ありません。そのため、パフォーマンスと実行が向上します。データ・アクセスのために、サーバー上でJavaコードを実行するとき、Oracle Databaseはサーバー側内

部ドライバを使用します。

アクション・アイテム3: SQLPlusを介してデータベースに接続し、ServersideConnect.javaを起動する前にServersideConnect.sqlを実行します。詳細は、Githubのサンプルを参照してください。

ServersideConnect.sql

```
Rem NAME
Rem ServersideConnect.sql
Rem
Rem DESCRIPTION
Rem SQL for invoking the method which gets a server side connection to
rem Reads the content of the Java source from ServersideConnect.java
rem then compiles it
connect username/pwd
CREATE OR REPLACE AND COMPILE JAVA SOURCE NAMED ServersideConnect_src AS
@ ServersideConnect.java
/
show error
rem A wrapper (a.k.a. Call Spec), to invoke Java
rem function in the database from SQL, PL/SQL, and client applications
CREATE OR REPLACE PROCEDURE ServersideConnect_proc AS
LANGUAGE JAVA NAME 'ServersideConnect.jrun ()';
/
rem running the sample
connect username/pwd
SET SERVEROUTPUT ON SIZE 10000
CALL dbms_java.set_output (10000);

execute ServersideConnect_proc;


InternalT2Server.java
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import oracle.jdbc.driver.OracleDriver;
import oracle.jdbc.pool.OracleDataSource;

public class ServersideConnect {

  static public void jrun() throws SQLException {
    // For testing ServersideConnect
    // test("jdbc:oracle:kprb:@");
    method1("jdbc:default:connection");
    method2();
  }
 /*
  * Shows using the server side Type 2 driver a.k.a KPRB driver
  */
 static public void method1 (String url) throws SQLException {
    Connection connection = null;
    try {
     // Method 1: Using OracleDataSource
      OracleDataSource ods = new OracleDataSource();
      ods.setURL(url);
      connection = ods.getConnection();
```

```
      System.out.println("Method 1: Getting Default Connection "
          + "using OracleDataSource");
      // Perform database operation
      printEmployees(connection);
    }
}

static public void method2 () throws SQLException {
   Connection connection = null;
   try {
        OracleDriver ora = new OracleDriver();
        connection = ora.defaultConnection();
        System.out.println("Method 2: Getting Default Connection "
          + "using OracleDriver");
        // Perform database operation
        printEmployees(connection);
   }
}

 /*
  * Displays employee_id and first_name from the employees table.
  */
  static public void printEmployees(Connection connection)
     throws SQLException {
    ResultSet resultSet = null;
    Statement statement = null;
    try {
      statement = connection.createStatement();
      resultSet = statement.executeQuery("SELECT employee_id, first_name"
         +    " FROM employees order by employee_id");
      while (resultSet.next()) {
        System.out.println("Emp no: " + resultSet.getInt(1) + " Emp name: "
            + resultSet.getString(2));
      }
    }
    catch (SQLException ea) {
      System.out.println("Error during execution: " + ea);
      ea.printStackTrace();
    }
    finally {
      if (resultSet != null) resultSet.close();
      if (statement != null) statement.close();
    }
  }
}
```

3 HR Webアプリケーションの概要

HR Webアプリケーションは、AnyCo Corporationという架空の企業の全従業員に関連する情報にアクセスすることを目的としています。

このアプリケーションにアクセスできるユーザーは、次の2種類です。

- HRStaff (HRスタッフ)
- HRAdmin (HR管理者)

HRStaffアカウントとHRAdminアカウントは、権限が異なります。

HRStaffは、読取り専用アクセスでアプリケーションを使用でき、従業員レコードを更新/削除する権限はありません。HRStaffに実行できるのは、従業員をリストすることと、従業員IDで検索することだけです。

HRAdminはアプリケーションを完全に制御でき、読取りおよび書込み権限を持っています。HRAdminは従業員レコードの更新/削除など、アプリケーションのすべての機能にアクセスでき、すべての従業員の給与増額の指定もできます。

この章の内容は、次のとおりです。

- HR Webアプリケーションの機能
- パッケージ

HR Webアプリケーションの機能

AnyCo Corporationに関連する情報にアクセスする機能のリストを次に示します。

hrstaffを介して次の機能を実行できます。

- **全従業員をリスト**

  List All Employeesオプションを使用して、従業員情報を取得します。この関数は、Employee_ID、First_Name、Last_Name Email、Phone_Number,Job_Id、およびSalaryなどの情報をリストします。

- **従業員IDによる検索**

  primary key (これが従業員ID)を使用して、特定の従業員を検索します。

  hradminユーザーを介して次の機能を実行できます。

  hradminユーザーはアプリケーションを完全に制御し、読取り権限と更新権限の両方を持っています。

- **従業員レコードの更新**

  Update Employee Record関数を使用して、従業員レコードを更新できます。最初に、従業員の名前に基づいて従業員を検索します。UPDATE関数を使用して、first_name、last_name、email phone_number、job_id、salaryなど、レコードの従業員の詳細情報を更新できます。

  データベースから従業員レコード全体を削除するには、DELETE関数を使用します。

- **給与の増額**

  増分増額タブを使用して、hikeの給与のパーセントを変更(増減)できます。

- **情報**

  このページでは、HRアプリケーションの概要を示し、そこで提供される様々な機能について説明します。

4 アプリケーション開発の開始

Oracle Database 12c リリース2 (12.2)に接続するJavaアプリケーションを開発するには、必要に応じていくつかのコンポーネントをインストールしておく必要があります。この章の構成は、次のとおりです。

- インストールする必要があるもの
- Oracle Database 12c リリース2 (12.2)のインストールの検証
- Oracle JDeveloperまたはany Java IDE(Eclipse、NetBeans、Intellij)のインストール

インストールする必要があるもの

サンプル・アプリケーションを開発するには、次の製品およびコンポーネントをインストールする必要があります。

- Oracle Database 12cリリース2 (12.2)
- J2SEまたはJDK
- Apache Maven
- JDeveloper IDE
- Web Server (Tomcat)
- Oracle Database 12c リリース2 (12.2)

  Java Webアプリケーションを作成するには、データベースに通常付属しているHRスキーマで動作するOracle Database 12cリリース2 (12.2)サーバーのインストールが必要です。Oracle Database12cリリース2 (12.2)をインストールするには、2つの方法があります。

  Oracle Database Services on Cloud

  Oracle Database Services on Cloudは、オラクル社が提供する計算処理能力、物理記憶域およびメンテナンス操作と管理操作のためのツールを利用することで単一のOracle Databaseへのアクセスを提供します。わずか数分で動作可能なデータベースにアクセスできます。Oracleクラウドにデータベースのインスタンスを作成すると、その機能と、EECSオプションを除くOracle Databaseで使用可能な操作への完全なアクセスが可能になります。

  Oracle Cloudでは、ニーズに適したコストとレベルで開始する柔軟性が提供されます。後で、変化する要件にあわせて適応できます。

  データベース・クラウド・サービス・ページにログインして、使用可能な様々なオプションを確認し、サインアップできます。簡単に参照できるように、いくつかのオプションを次に示します。

  Oracle Database Cloud Service (DBCS)

  Oracle Database Cloud Service on Oracle Cloud Infrastructure (CS)

  Oracle Database Exadata Express Cloud Service (EECS) - 完全管理

  Oracle Database Exadata Cloud Service (ExaCS)

  Oracle Database Exadata Cloud Machine (ExaCM)

  Oracle Database Cloud Service (DBCS)にアカウントを作成します。これにより、HRスキーマにもアクセスできます。

  
  注意:

OTN ページ Oracle Database Cloud Service (DBCS)での Java アプリケーションおよび IDE の使用に、データベース・インスタンスを作成し、このガイドの最初で提供される JDBC および UCP コード・サンプルを使用して接続を試行する手順が説明されています。

手順1:

OTNで使用可能なOracle Databaseのインストール

代替オプションとして、Oracle Database 12cリリース2をオンプレミスにインストールすることもできます。リンクに従ってOracle Databaseをインストールし、リリース・ノートを確認します。

- Oracle Database 12c Release 2 for Linux
- Oracle Database 12c Release 2 for Windows

注意:

JDK 8 のダウンロード - Java Development Kit (JDK 8)をダウンロードして Java アプリケーションを作成およびコンパイルします。Java のインストールの詳細は、http://www.oracle.com/technetwork/java/javase/downloads/index.html を参照してください。

JDBC APIの詳細は、http://www.oracle.com/technetwork/java/overview-141217.htmlを参照してください。

Oracle Database 12c リリース2 (12.2)

Javaアプリケーションを作成するには、HRスキーマ(データベースに付属)で動作するOracle Database 12c リリース2 (12.2) Serverのインストールが必要です。インストールにより、Oracle Database 12c リリース2 (12.2)インスタンスが作成され、このデータベースを管理するためのツールが提供されます。

Oracle Database Services on Cloud

Oracle Database Services on Cloudは、オラクル社が提供する計算処理能力、物理記憶域およびメンテナンス操作と管理操作のためのツールを利用することで単一のOracle Databaseへのアクセスを提供します。わずか数分で動作可能なデータベースにアクセスできます。Oracleクラウドにデータベースのインスタンスを作成すると、その機能と、EECSオプションを除くOracle Databaseで使用可能な操作への完全なアクセスが可能になります。

Oracle Cloudでは、ニーズに適したコストとレベルで開始する柔軟性が提供されます。後で、変化する要件にあわせて適応できます。

データベース・クラウド・サービス・ページにログインして、使用可能な様々なオプションを確認し、サインアップできます。簡単に参照できるように、いくつかのオプションを次に示します。

Oracle Database Cloud Service (DBCS)

Oracle Database Cloud Service on Oracle Cloud Infrastructure (CS)

Oracle Database Exadata Express Cloud Service (EECS) - 完全管理

Oracle Database Exadata Cloud Service (ExaCS)

Oracle Database Exadata Cloud Machine(ExaCM)

Oracle Database Cloud Service (DBCS)にアカウントを作成します。これにより、HRスキーマにもアクセスできます。

OTNページOracle Database Cloud Service (DBCS)でのJavaアプリケーションおよびIDEの使用の順を追った説明に従ってデータベース・インスタンスを作成し、このガイドの最初で提供されるJDBCおよびUCPコード・サンプルを使用して、そのインスタンスに接続することもできます。

手順1:

OTNで使用可能なOracle Databaseのインストール

代替オプションとして、Oracle Database 12cリリース2をオンプレミスにインストールすることもできます。リンクに従ってOracle Databaseをインストールし、リリース・ノートを確認します。

詳細は、次に示すOracle Database 12*c* リリース2 (12.2)のインストレーション・ガイドおよびリリース・ノートを参照してください。

- 『*Oracle Database*インストレーション・ガイド *for Linux*』
- 『*Oracle Database*インストレーション・ガイド*for Microsoft Windows*』

JDBCアプリケーションのHRスキーマのロック解除

HRユーザー・アカウント(このマニュアルのJavaアプリケーションで使用するサンプルHRスキーマの所有者)は、最初はロックされています。HRとしてログインするには、まず管理権限を持つユーザー(SYS)としてログインし、アカウントのロックを解除する必要があります。

データベースがローカルにインストールされている場合は、**SQLコマンドラインの実行**を使用して、次のようにアカウントのロックを解除します。

1. **「SQLコマンドラインの実行」**にアクセスするには、**「スタート」**メニューから**「プログラム」**(または「すべてのプログラム」)→**「Oracle Database 12c Release 2 (12.2)」**を選択し、**「SQLコマンドラインの実行」**をクリックします。DBA権限を持つユーザーとしてログインします。次に例を示します。

   ```
   > CONNECT SYS AS SYSDBA;
   Enter password: password
   ```

2. 次のコマンドを実行します。

   ```
   > ALTER USER HR ACCOUNT UNLOCK;
   ```

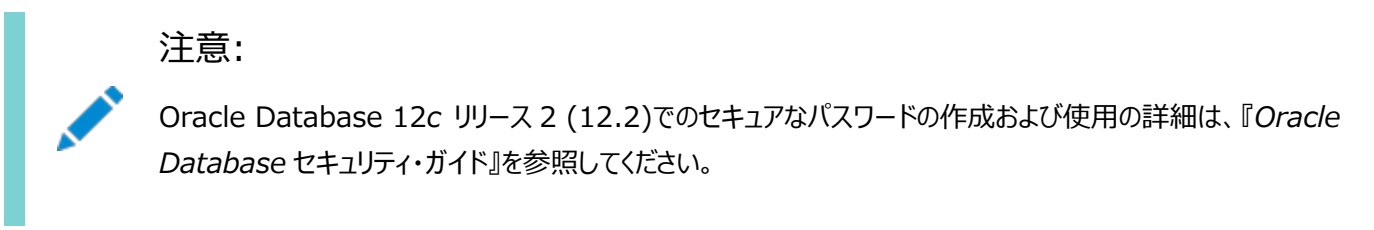
   または、

   ```
   > ALTER USER HR IDENTIFIED BY HR;
   ```

3. 次のように接続をテストします。

   ```
   > CONNECT HR
   Enter password: password
   ```

データベースに接続したことを示すメッセージが表示されます。



また、HRスキーマにある制約およびトリガーの一部は、このマニュアルで作成するJavaアプリケーションの目的に合っていません。次のSQL文を使用して、これらの制約およびトリガーを削除する必要があります。

```
DROP TRIGGER HR.UPDATE_JOB_HISTORY;
DROP TRIGGER HR.SECURE_EMPLOYEES;
DELETE FROM JOB_HISTORY;
```

JDK 8

Javaアプリケーションを作成およびコンパイルするには、JDK8 (Java Development Kit)が必要になります。

注意:

Oracle Database 12c リリース 2 (12.2)の JDBC ドライバは JDK8 をサポートしています。Java のインストールの詳細は、http://www.oracle.com/technetwork/java/javase/downloads/index.html を参照してください。

JDeveloper IDE

このガイドでは、Oracle JDeveloperリリース12cが、サンプルJava Webアプリケーションの作成に使用される統合開発環境(IDE)です。JDeveloperは、ソース・コードの編集、アプリケーションのコンパイル、WARファイルの作成、およびwarファイルを統合WebLogic Server (WLS)にデプロイするために使用します。

J2SEまたはJDK

Javaアプリケーションを作成およびコンパイルするには、Java 2 Platform, Standard Edition, Software Development Kit(J2SE SDK)(以前のJava Development Kit(JDK))がすべて必要です。

注意:

Oracle Database 12c リリース 2 (12.2)は、JDK 8 をサポートしています。

関連項目:

- Javaのインストールの詳細は、http://www.oracle.com/technetwork/java/javase/downloads/index.htmlを参照してください。
- JDBC APIの詳細は、http://www.oracle.com/technetwork/java/overview-141217.htmlを参照してください。

統合開発環境

アプリケーションの開発を簡単にするため、統合開発環境(IDE)でアプリケーションを開発することができます。このマニュアルでは、Oracle JDeveloperを使用して、このアプリケーションのファイルを作成します。

Webサーバー

このマニュアルで開発するサンプル・アプリケーションは、JavaServer Pages (JSP)テクノロジを使用して情報を表示し、ユーザーからの入力を受け入れます。これらのページをデプロイするには、サーブレットおよびJSPコンテナを使用するWebサーバー(Apache Tomcatアプリケーション・サーバーなど)が必要です。

このマニュアルでは、JSPページのデプロイに、JDeveloperの Oracle WebLogicサーバーという埋込みサーバーを使用します。Oracle JDeveloperをインストールしない場合でも、任意のWebサーバーを使用してJSPページをデプロイできます。

これらのサーバーの詳細は、ベンダー固有のドキュメントを参照してください。

Oracle Database 12*c* リリース2 (12.2)のインストールの検証

Oracle Database 12*c* リリース2 (12.2)のインストールはプラットフォーム固有です。サンプル・アプリケーションの作成に進む前に、インストールが成功したことを検証する必要があります。この項では、Oracle Database 12*c* リリース2 (12.2)のインストールを検証する手順について説明します。

インストールの検証には、次の作業があります。

Githubリポジトリの詳細

表4-1 Githubリポジトリの詳細

名前と場所	詳細
2DaysJavaGuide-ワークスペース	この zip には、HR Web アプリケーションを構築するための Java クラスの概要が含まれています。zip をダウンロードし、任意の場所に解凍して指示に従います。
HRWebApp	このリポジトリには、アプリケーションの完全なコード・サンプルが含まれています。これは、問題または例外が発生した場合に参照してください。

JDeveloperでのアプリケーションのインポート

JDeveloperにアプリケーションをインポートするには、次の手順を実行します。

1. HRWebApp_Workspaceが抽出された場所に移動します。
2. アプリケーションの名前を選択します。
3. HR Webアプリケーションの構築に必要なすべてのファイルを含むプロジェクトが作成されます。これで、編集を開始できます。

JDeveloperでのアプリケーションのコンパイル

JDeveloperでmavenコマンドを使用して.warファイルをコンパイルまたは作成できます。

1. プロジェクトを右クリックします。
2. Mavenを実行します
3. compile/package/cleanに移動します

Mavenを使用したコンパイルと、任意のJava EEコンテナでのアプリケーションの実行

a.Mavenを使用したコンパイル:

HR Webアプリケーションは、Mavenのコマンドを使用して簡単にコンパイルすることもできます。Oracle mavenリポジトリを使用している場合は、必要なすべての詳細が簡単にアクセスできるようにsettings.xmlファイルを用意します。HRWebApp_workspaceをダウンロードして必要なコードを追加した後、次のコマンドを使用してソース・コードを消去、コンパイルおよびパッケージ化します。

次のコマンドを実行します。

```
mvn -s settings.xml clean
mvn -s settings.xml compile
mvn -s settings.xml package
```

b.任意のJava EEコンテナへの.warファイルのデプロイ

mavenコマンド"mvn-s settings.xml package"を使用してソース・コードがパッケージ化されると、warファイルはフォルダ"target/JdbcWebSamples.war"の下に配置されます。このwarファイルをTOMCAT_HOME/webapps/に配置し、tomcatサーバーを起動します。tomcatが起動したら、URL http://localhost:8080/JdbcWebSamples/を使用してHR Webアプリケーションにアクセスします。

ログイン/ログアウト・モジュールが追加された場合は、**hradmin**または**hrstaff**ユーザーのいずれかでログインします。

JDeveloperでのアプリケーションの実行

JDeveloperは、統合されたWebLogic Serverを使用してWebアプリケーションを実行します。

全従業員をリスト

HR Webアプリケーションにはいくつかの機能があります。「すべてリスト」は、Employee_id、First_name、Last_Name、Email、Phone_number、Job_id、Salaryなどの従業員の詳細がEmployees表から取得され、Webページに表示される機能です。「すべてリスト」機能が表示されているスクリーンショットを参照してください。

5 全従業員をリスト

HR Webアプリケーションにはいくつかの機能があります。「すべてリスト」は、Employee_id、First_name、Last_Name、Email、Phone_number、Job_id、Salaryなどの従業員の詳細がEmployees表から取得され、Webページに表示される機能です。「すべてリスト」機能が表示されているスクリーンショットを参照してください。

従業員のJava Beanエンティティの作成

クラス名: src/main/java/com/oracle/jdbc/samples/entity/Employee.java

Githubの場所: Employee.java

説明: これは、従業員のすべての属性のgetterメソッドおよびsetterメソッドを含むクラスです。例: First_name、Last_Name、Employee_Idなどにはgetterメソッドとsetterメソッドがあります。

実行する手順:

1. サンプルに示すように、コンストラクタ*Employee()*を作成します。

2. 従業員のすべての属性のgetterメソッドおよびsetterメソッドを作成し、Webアプリケーションに表示します。

Employee.javaの作成:

1. クラスEmployee.javaのパッケージを宣言します。

```
package com.oracle.jdbc.samples.entity;
```

2. Employeeクラスに必要な次のパッケージをインポートします

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
```

3. Employeeクラスを宣言します。開きカッコ({)および閉じカッコ(})を追加します。カッコの間にカーソルを置きます。

```
public class Employee {
```

4. 従業員の属性ごとに次の変数を宣言します。

```
private int Employee_Id;
private String First_Name;
private String Last_Name;
private String Email;
private String Phone_Number;
private String Job_Id;
private int Salary;
```

5. ResultSetを入力として受け取り、SQLExceptioをスローするEmployeeのコンストラクタを作成します。このコンストラクタで、Employeeの属性のすべての値を設定します。

```
public Employee(ResultSet resultSet) throws SQLException {
  this.Employee_Id = resultSet.getInt(1);
  this.First_Name = resultSet.getString(2);
  this.Last_Name = resultSet.getString(3);
  this.Email = resultSet.getString(4);
  this.Phone_Number = resultSet.getString(5);
  this.Job_Id = resultSet.getString(6);
  this.Salary = resultSet.getInt(7);
}
```

getterメソッドおよびsetterメソッドを作成する手順:

GetterメソッドおよびSetterメソッドは、カプセル化を実現するXの値を取得および設定するために使用します。Employeeのすべての属性(employee_id、first_name、last_name、salaryなど)に対するgetXおよびsetXメソッドを作成します。

1. 次に示すように、Employee_Idのgetterメソッドおよびsetterメソッドを作成します

```
public int getEmployee_Id() {
   return Employee_Id;
}
  public void setEmployee_Id(int Employee_Id) {
     this.Employee_Id = Employee_Id;
  }
```

2. 従業員のFirst_Nameのgetterおよびsetterメソッドを作成します

```
public String getFirst_Name() {
  return First_Name;
}
public void setFirst_Name(String First_Name) {
  this.First_Name = First_Name;
}
```

3. 従業員のLast_Nameのgetterおよびsetterメソッドを作成します

```
public String getLast_Name() {
  return Last_Name;
}
public void setLast_Name(String Last_Name) {
  this.Last_Name = Last_Name;
}
```

4. 従業員のEmailのgetterおよびsetterメソッドを作成します

```
public String getEmail() { return Email; } public void setEmail(String Email) { this.Email =
Email; }
```

5. 従業員のPhone Numberのgetterおよびsetterメソッドを作成します

```
public String getPhone_Number() {

return Phone_Number; }

public void setPhone_Number(String Phone_Number) {

this.Phone_Number = Phone_Number;

}
```

6. 従業員のJobIdのgetterおよびsetterメソッドを作成します

```
public String getJob_Id() { return Job_Id;}public void setJob_Id(String Job_Id) { this.Job_Id =
Job_Id;}
```

7. 従業員のSalaryのgetterおよびsetterメソッドを作成します

```
public int getSalary() {
  return Salary;
}
public void setSalary(int Salary) {
  this.Salary = Salary;
}
```

JDBC接続のためのJava Beanインタフェースの作成

1．クラスEmployeeBean.javaのパッケージを宣言します

```
package com.oracle.jdbc.samples.bean;
```

2．従業員の詳細を含むEmployeeエンティティ・クラスをインポートします

```
import com.oracle.jdbc.samples.entity.Employee;
```

3．インタフェースEmployeeBeanクラスを宣言します。開きカッコ({)および閉じカッコ(})を追加します。カッコの間にカーソルを置きます。

```
public interface EmployeeBean {
```

4．新しい行で、EmployeeオブジェクトのListを返すメソッドgetEmployees()を宣言します。

```
public List<Employee> getEmployees();
```

JDBC接続のためのJava Bean実装の作成

クラス名: src/main/java/com/oracle/jdbc/samples/bean/EmployeeBeanImpl.java

Githubの場所: EmployeeBeanImpl.java

説明: これは実装クラスです。EmployeeBean.javaで宣言されたすべてのメソッドがこのクラスに実装されます。各機能に関連する新しいメソッドが、次の各章に追加されます。最初に、"ListAll"機能に必要なメソッドの実装を追加します。

実行する手順:

手順4: メソッド*getConnection()*を作成して、データベースへの接続を確立します。データベースを指すように、接続URL、DBユーザー名およびDBパスワードを更新します。

手順5: 従業員表から従業員のリストを取得するメソッドgetEmployees()を作成します。Employees表から目的の列を選択して、使用するSELECT問合せを更新します。

手順4: getConnection()メソッドを作成する手順

1. EmployeeBean.javaのパッケージを宣言します。

```
package com.oracle.jdbc.samples.bean;
```

2. 従業員の詳細を含むEmployeeクラスをインポートします。

```
Import com.oracle.jdbc.samples.entity.Employee;
```

3. 次に示すように、他の依存クラスをインポートします。特定のクラスがインポートされない場合、JDeveloperでは、必要なパッケージのインポートを促すメッセージが表示されます。[Alt]キーを押しながら[Enter]キーを押してクラスをインポートします。

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import java.sql.PreparedStatement;
import oracle.jdbc.OracleStatement;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.driver.OracleDriver;
import oracle.jdbc.OracleTypes;
import java.sql.PreparedStatement;
import oracle.jdbc.OracleStatement;
import oracle.jdbc.OracleConnection;

import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;
```

4. 次に示すクラス宣言EmployeeBeanImplを追加します。これがEmployeeBeanを実装します。開きカッコ({)および閉じカッコ(})を追加します。カッコの間にカーソルを置きます。

```
public class EmployeeBeanImpln implements EmployeeBean {
```

5. staticメソッドgetConnection()を宣言して接続を確立します。開きカッコ({)および閉じカッコ(})を追加します。カッコの間にカーソルを置きます。

```
public static Connection getConnection() throws SQLException {
```

6. 次に示すようにドライバを登録します。

```
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
```

7. データベースURLおよびデータベースのユーザー名とパスワードを渡して、接続を取得します。

```
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@//myorclhost:5521/myorcldbservice", "hr", "hr");
```

8. データベース接続を返します。

```
return connection;
```

getEmployees()メソッドを作成する手順:

1. メソッドgetEmployees()を宣言します。開きカッコ({)および閉じカッコ(})を追加します。カッコの間にカーソルを置きます。

```
Public List<Employee> getEmployees()throws SQLException {
```

2. List<Employee>型の戻り値の変数を宣言します

```
List<Employee> returnValue = new ArrayList<>();
```

3. tryブロックを開始します。ソース・コードはJDK8でコンパイルされ、自動クローズ可能文を使用するため、catchブロックとfinallyブロックを明示的に指定する必要はありません。最初のtryブロックは、getConnection()メソッドを呼び出してデータベース接続を取得するためのものです。データベース接続を確立するために可変接続を宣言します。

```
try (Connection connection = getConnection()) {
```

4. Statementを作成するために別のtryブロックを開始します。

```
try (Statement statement = connection.createStatement()) {
```

5. ResultSetの別のtryブロックを開始します。実行する必要がある問合せを含めます。問合せで従業員の必須フィールドをすべて取得するようにします。

```
try (ResultSet resultSet = statement.executeQuery("SELECT Employee_Id, First_Name, Last_Name,
Email, Phone_Number, Job_Id, Salary FROM EMPLOYEES")) {
```

6. whileループを開始して、ResultSetから取得した従業員のリストを取得します。

```
while(resultSet.next()) {
  returnValue.add(new Employee(resultSet));
}
```

7. すべてのtryブロックのカッコを閉じます。

8. SQLExceptionを捕捉し、次に示すようにロガーにメッセージを記録します。

```
catch (SQLException ex) {
  logger.log(Level.SEVERE, null, ex);
  ex.printStackTrace();
}
```

9. メソッドgetEmployees()から従業員のListを返します

```
return returnValue;
```

リクエストを処理するサーブレットの作成

次のコードは、リクエストを処理するサーブレットを作成するために必要な手順です。

クラス名: src/main/java/com/oracle/jdbc/samples/web/WebController.java

Githubの場所: WebController.java

説明: これは、アプリケーションのすべてのフローを制御するメイン・サーブレットです。アプリケーションの新しい機能ごとに、新しいリクエストおよびレスポンスを処理するコードをdoPost()およびprocessResponse()にそれぞれ追加します。

実行する手順:

6. サーブレットの*WebController.java*および*reportError()*メソッドを作成します

7. メソッド*processRequest()*を作成します。このメソッドはGETとPOSTの両方のHTTPリクエストを処理します。

8. メソッド*doGet()*を作成します。詳細を追加し、データベースから従業員の詳細を取得して結果をJSONに表示します。JSONは、HTMLに表示される結果の出力形式になります。

9. サーブレットに関するいくつかの一般情報を表示するメソッドgetServletInfo()を作成します。

10. 例外およびその他のエラー・メッセージを記録するロガーを作成します。

手順6: WebController.javaおよびreportError()メソッドを作成する手順

1. WebController.javaのパッケージを宣言します。

```
package com.oracle.jdbc.samples.web;
```

2. 従業員の詳細とEmployeeBeanImplも含むEmployeeクラスをインポートします。

```
import com.oracle.jdbc.samples.entity.Employee;
   import com.oracle.jdbc.samples.bean.EmployeeBean;
   import com.oracle.jdbc.samples.bean.EmployeeBeanImpl;
```

3. Employeeの結果を表示するためにGSON (Google GSON)をインポートします。

```
import com.google.gson.Gson; import com.google.gson.reflect.TypeToken;
```

4. 次に示すように、他の依存クラスをインポートします。特定のクラスがインポートされない場合、JDeveloperでは、必要なパッケージのインポートを促すメッセージが表示されます。[Alt]キーを押しながら[Enter]キーを押してインポートします。

```
import javax.servlet.ServletException;
        import javax.servlet.annotation.WebServlet;
        import javax.servlet.http.*;
        import java.io.IOException;
        import java.io.PrintWriter;
        import java.io.BufferedReader;
        import java.util.ArrayList;
        import java.util.List;
        import java.util.Map;

        import java.util.logging.Logger;
```

5. サーブレットに注釈を追加します。

```
@WebServlet(name = "WebController", urlPatterns = {"/WebController"})
```

6. HttpServletを拡張する次のクラス宣言WebControllerを追加します。開きカッコ({)および閉じカッコ(})を追加します。カッコの間にカーソルを置きます。

```
public class WebController extends HttpServlet {
```

7. EmployeeBeanImpl型のオブジェクト"employeeBean"を宣言します。これはグローバル変数になり、reportError()、processRequest()およびdoGet()などのすべてのメソッドで使用できます。

8. Gson型のオブジェクト"gson"を宣言します。これはグローバル変数になり、reportError()、processRequest()およびdoGet()などのすべてのメソッドで使用できます。

```
Gson gson = new Gson();
```

9. 次に示すように、メソッドreportErrorを宣言します。これは、エラーを取得してページに表示するためです。

```
private void reportError(HttpServletResponse response, String message) throws ServletException,
IOException {
```

10. 次に示すように、レスポンス・コンテンツ・タイプを"text/html"およびcharset=UTF-8に設定します。

```
response.setContentType("text/html;charset=UTF-8");
```

11. PrintWriterオブジェクトを作成し、示されたとおりにエラー・メッセージを出力します。

try (PrintWriter out = response.getWriter()) {

```
try (PrintWriter out = response.getWriter()) {
out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet WebController</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>" + message + "</h1>");
out.println("</body>");
out.println("</html>");
}
```

手順7: プロセス・リクエストを作成する手順

1. processRequest(req,res)に次のメソッド宣言を追加します。開きカッコと閉じカッコ({、})を追加し、カッコの間にカーソルを置きます。

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
```

2. Employeeオブジェクトを含む*List*型の変数*employeeList*を宣言します。 Gsonオブジェクトを処理する変数gsonを宣言します。

```
List<Employee> employeeList = null;
```

3. EmployeeBeanのgetEmployeesメソッドを呼び出して、employeeListオブジェクトをインスタンス化します。

```
employeeList = employeeBean.getEmployees();
```

4. employeeListがNULLでないかを確認します

```
if (employeeList != null) {
```

5. コンテンツ・タイプを"application/json"に設定します

```
response.setContentType("application/json");
```

6. メソッドtoJson(…)を呼び出してemployeeListをJSONに変換します。

```
gson.toJson(employeeList,
            new TypeToken<ArrayList<Employee>>() {}.getType(),
response.getWriter());
```

7. if条件の終了。カッコを閉じます(})

8. employeeListが空の場合のエラー・シナリオを対象にするelse条件を追加します。

```
else {
  response.setStatus(HttpServletResponse.SC_NOT_FOUND);
}
```

手順8: doGet()を作成する手順:

1. 次のdoGet()のメソッド宣言を追加します。開きカッコと閉じカッコ({、})を追加し、カッコの間にカーソルを置きます。

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
```

2. HTTPServletRequestおよびHTTPServletResponseレスポンス・オブジェクトを渡して、メソッドprocessRequest(request, response)を呼び出します。

```
processRequest(request, response);
```

手順9: getServletInfo()を作成する手順

1. 次のgetServletInfo()のメソッド宣言を追加します。開きカッコと閉じカッコ({、})を追加し、カッコの間にカーソルを置きます。

```
public String getServletInfo() {
```

2. return文で、サーブレットに関するメッセージを設定します。

```
return "JdbcWebServlet: Reading Employees table using JDBC and transforming it as a JSON. ";
```

手順10: Logger ()を作成する手順:

1. クラスWebController.javaのカッコを閉じる前に、クラスの最後にLogger型の可変ロガーを作成します。

```
private static final Logger logger =
Logger.getLogger(WebController.class.getName());
```

結果を表示するHTMLページの作成

クラス名:

src/main/webapp/listAll.html

Githubの場所: listAll.html

説明: データベースから取得した結果を表示するHTMLページです。

実行する手順:

手順11: HTMLページのtitle、stylesheetおよびbodyを作成します

手順12: <script>タグを開始してGETリクエストを処理します

手順13: JSONレスポンスを処理して結果をHTMLページに表示するメソッドprocessResponse()を作成します。

手順11: HTMLページのtitle、stylesheetおよびbodyを作成する手順:

1. HTMLページのtitle、stylesheetおよびbodyを作成します。

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>List all Employees</title>
```

<link rel="stylesheet" type="text/css" href="css/app.css" > </head> <body>

手順12: GETリクエストを処理する手順:

1. <script>タグを開始し、URLおよびHTTPRequestの変数を宣言します。

<script>

var xmlhttp = new XMLHttpRequest();

var url = "WebController";

2. リクエストの送信時、つまり各機能のリンクが選択された場合のアクションを定義します。

```
xmlhttp.onreadystatechange=function() {
     if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
       processResponse(xmlhttp.responseText);
     }
   }
xmlhttp.open("GET", url, true);
xmlhttp.send();
```

手順13: processResponse()リクエストを処理する手順:

1. HTMLページにJSON結果を表示するには、関数processResponse()を作成します。

```
function processResponse(response) {
// Process the JSON response into an array.
var arr = JSON.parse(response);
var i;
var out - "<table>";
keys = Object.keys(arr[0]);
//Print headers
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
```

```
out += "<th>"+keys [i]+"</th>"
}
out += "</tr>";
// Print values
for(j = 0; j < arr.length; j++) {
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<td>"+arr[j][keys[i]]+"</td>"
}
out += "</tr>"
}
out += "</table>";

document.getElementById("id-emp").innerHTML = out;

}

}
```

CSSファイルの作成

次のコードは、JSONを処理してHTML上で表示するJavaスクリプト内でメソッドprocessResponse()を作成します。

クラス名: src/main/webapp/css/app.css

Githubの場所: app.css

説明: スタイルシートには、ページ上のボタン、サイド・ナビゲーション、メイン・ページ、リンクなどすべてのUI要素に対する色、フォントおよびスタイルが含まれています。

実行する手順:

手順14: app.cssをダウンロードして、アプリケーションで使用します。

6 従業員IDによる検索

「従業員IDによる検索」は、主キーである従業員IDに基づいて特定の従業員を検索できる機能です。ユーザーは従業員IDを入力してリクエストを送信する必要があります。

Employee Java Bean

クラス名: src/main/java/com/oracle/jdbc/samples/bean/EmployeeBean.java

Githubの場所: EmployeeBean.java

説明: IDにより従業員を検索する新しいメソッドgetEmployee(int)です。

実行する手順:

手順1: getEmployee(int)メソッドの宣言

手順1: getEmployee(int)メソッドを宣言する手順

1. ファイルEmployeeBeanは、1日目に"ListAll"機能ですでに作成されています。同じクラスを使用して、各機能に対して新しいメソッドを追加できます。

2. 新しい行で、EmployeeIdをパラメータとして使用するgetEmployee(int)メソッドを宣言します。

```
public List<Employee> getEmployee(int empId);
```

新しいメソッドgetEmployee(int)をEmployeeBeanImpl.javaに実装します。

Githubの場所: EmployeeBeanImpl.java

説明: IDにより従業員を検索する新しいメソッドgetEmployee(int)を実装します。このメソッドは、従業員IDを入力パラメータとして使用してEmployee型のオブジェクトを返します。

実行する手順:

手順2: 新しいメソッドgetEmployee(int)を実装します。

手順2: getEmployee(int)メソッドを作成する手順:

1. Employee型のオブジェクトのリストを返す*getEmployee(int)*メソッドを宣言します。

```
public List<Employee> getEmployee(int empId) {
```

2. List<Employee>型の戻り値の変数を宣言します

```
List<Employee> returnValue = new ArrayList<>();
```

3. 最初のtryブロックは、メソッドgetConnection()を呼び出してデータベース接続を取得するためのものです。データベース接続を確立するために可変接続を宣言します。

```
try (Connection connection = getConnection()) {
```

4. PreparedStatementを作成するために別のtryブロックを開始します。PreparedStatementには、従業員IDに基づいて従業員を選択するために実行する必要がある問合せが含まれます。

```
try (PreparedStatement preparedStatement = connection.prepareStatement("SELECT Employee_Id, First_Name, Last_Name, Email, Phone_Number, Job_Id, Salary FROM EMPLOYEES WHERE Employee_Id = ?")) {
```

5. 問合せの従業員IDである入力パラメータを設定します。

```
preparedStatement.setInt(1, empId);
```

6. ResultSetの別のtryブロックを開始します。

```
try (ResultSet resultSet = preparedStatement.executeQuery()) {
```

7. 返されたものがあるかどうかを確認します。はいの場合は、returnValueに追加し、それ以外の場合は例外をスローします。

```
if(resultSet.next()) {
    returnValue.add(new Employee(resultSet));
  } else {
    throw new SQLException("No recrods found");
  }
```

8. すべてのtryブロックのカッコを必ず閉じてください。合計3つのtryブロックを閉じる必要があります。

9. SQLExceptionを捕捉し、次に示すようにロガーにメッセージを記録します。

```
catch (SQLException ex) {
  logger.log(Level.SEVERE, null, ex);
  throw ex;
}
```

10. メソッドgetEmployee(int)から従業員のリストを返します。

```
return returnValue;
```

リクエストを処理するコードのサーブレットへの追加

クラス名: src/main/java/com/oracle/jdbc/samples/web/WebController.java

Githubの場所: WebController.java

説明: このサーブレットは、1日目ですでに作成されています。従業員IDによる検索に関連するコードを追加します。

実行する手順:

手順3: 従業員IDで検索するコードをメソッドprocessRequest()に追加します。

手順3: processRequest()へのコードの追加手順:

1. 従業員IDを取得する変数ID_KEYを宣言します。これはグローバル変数であるため、メソッドprocessRequest()外で宣言する必要がありますが、WebControllerクラス内で宣言する必要があります。

```
private static final String ID_KEY = "id";
```

2. メソッドprocessRequest()は、"ListAll"機能ですでに作成されています。「従業員IDによる検索」機能を実装するコードを追加します。

3. ユーザーからの入力を取得する文字列型の変数値を宣言します。

```
String value = null;
```

4. "ListAll"機能に追加されたものに加えて、新しい機能を処理するif条件を追加します。ユーザーが入力した従業員IDを取得し、メソッドgetEmployee(int)を起動して従業員レコードが存在するかどうかを確認します。

```
if ((value = request.getParameter(ID_KEY)) != null) {
      int empId = Integer.valueOf(value).intValue();
      employeeList = employeeBean.getEmployee(empId);
   } else {
      // Previously used getEmployees() method for Listall feature
      employeeList = employeeBean.getEmployees();
   }
```

従業員IDによる検索のための新規HTMLの作成

クラス名:

src/main/webapp/listById.html

Githubの場所: listById.html

説明: これは、従業員IDを入力する入力ボックスを示すHTMLです。従業員レコードが見つかった場合は、その従業員の詳細がページに表示されます。それ以外の場合は、エラー・メッセージが表示されます。

実行する手順:

手順4: HTMLページのtitle、headおよびstylesheetを作成します。

手順5: 入力フィールド(従業員IDなど)を送信する関数を作成します

手順6: JSONレスポンスを処理して結果をHTMLページに表示するメソッドprocessResponse()を作成します。

1. HTMLページのtitle、stylesheetおよびbodyを作成します。

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>List Employee by Id</title>
<!-- Specify the stylesheet here -->
<link rel="stylesheet" type="text/css" href="css/app.css" >
<!-- Bootstrap JS for the UI -->
<link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
</head>
```

手順5: 入力フィールドを処理する手順:

1. 従業員IDを取得するための<body>タグと<input>タグを開始します。

```
<body>
<div><label>Employee Id: </label>
<input id="empId" type="textfield"
onkeypress="return waitForEnter(event)"¥>
</div>
<br/>
<br/>
<script>
function waitForEnter(e) {
  if (e.keyCode == 13) {
    var tb = document.getElementById("empId");
    fetchElementById(tb.value)
    return false;
  }
}
<script>
var xmlhttp = new XMLHttpRequest();
var url = "WebController";
```

3. リクエストの送信時、つまり各機能のリンクが選択された場合のアクションを定義します。

```
xmlhttp.onreadystatechange=function() {
     if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
       processResponse(xmlhttp.responseText);
     }
```

```
   }
xmlhttp.open("GET", url, true);
xmlhttp.send();
```

手順6: processResponse()メソッドを作成する手順

1. HTMLページにJSON結果を表示するには、関数processResponse()を作成します。

```
function processResponse(response) {
//Process the JSON respnse into an array.
var arr = JSON.parse(response);
     var i;
var out = "<table>";
keys = Object.keys(arr[0]);

// Print Headers
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<th>"+keys[i]+"</th>"
}
out += "</tr>";
// Print values
for(j = 0; j < arr.length; j++) {
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<td>"+arr[j][keys[i]]+"</td>"

}
out += "</tr>"
}
out += "</table>";
document.getElementById("id-emp").innerHTML = out;
}
```

7 従業員レコードの更新

hradminには、従業員レコードを更新する権限があります。**hrstaff**ユーザーにこの権限はありません。

まず、レコード内の従業員を検索する必要があります。従業員に関連する情報を取得すると、従業員に関連する詳細を変更するための「編集」および「削除」オプションが表示されます。

この章では、作成する必要があるクラスと、従業員IDによる検索機能を作成するために追加する必要があるコードについて説明します。

この章では、次の方法について学習します。

1. *EmployeeBean.java*での新しいメソッド*getEmployeeByFn(String)*の宣言
2. *EmployeeBean.java*での新しいメソッド*updateEmployee(int)*の宣言
3. *EmployeeBeanImpl.java*での新しいメソッド*getEmployeeByFn(String)*の実装
4. *EmployeeBeanImpl.java*での新しいメソッド*updateEmployee(int)*の実装
5. リクエストおよびレスポンスを処理するコードの*WebController.java*への追加
6. 結果を表示するHTMLページ(*listByName.html*)の作成

クラス名:

`src/main/java/com/oracle/jdbc/samples/entity/Employee.java`

この例で作成したEmployee.javaファイルを使用します。

EmployeeBean.javaでの新しいメソッド*getEmployeeByFn(String)*の宣言

クラス名: src/main/java/com/oracle/jdbc/samples/bean/EmployeeBean.java

Githubの場所: EmployeeBean.java

説明: メソッドgetEmployeeByFn(String)は、名に基づいて従業員を検索するのに役立ちます。

従業員の詳細を変更するには、**hradmin**は最初に名で従業員を検索する必要があります。

実行する手順:

手順1: メソッドgetEmployeeByFn(String)の宣言

手順1: getEmployeeByFn(String)メソッドを宣言する手順

1. 演習の1日目で作成したクラスEmployeeBeanを使用します。各機能に新しいメソッドを追加できます。
2. パラメータとして名を使用するメソッドgetEmployeeByFn(String)メソッドを宣言します。

```
public List<Employee> getEmployeeByFn(String fn)throws SQLException;
```

新しいメソッド*updateEmployee(Employee)*の宣言

クラス名: src/main/java/com/oracle/jdbc/samples/bean/EmployeeBean.java。

Githubの場所: EmployeeBean.java

説明: この方法で、名、姓、給与、job_idなどの従業員の属性を更新できます。

実行する手順:

手順2: メソッド*updateEmployee(Employee)*の宣言

updateEmployee(Employee)メソッドを宣言する手順

1. 'ListAll'機能に対して1日目からの演習で作成したEmployeeBeanファイルを使用します。同じクラスを使用して新しい機能に新しいメソッドを追加できます。

2. パラメータとしてEmployeeオブジェクトを使用するメソッドupdateEmployee(Employee)を宣言します。

```
public String updateEmployee(Employee employee) throws SQLException;
```

従業員名による検索のための新しいメソッドgetEmployeebyFn()の実装

クラス名:

src/main/java/com/oracle/jdbc/samples/bean/JdbcBeanImpl.java

Githubの場所: EmployeeBeanImpl.java

説明: 従業員IDで検索するための新しいメソッドgetEmployeeByFn(String)を実装します。このメソッドは、従業員IDを入力パラメータとして使用し、Employee型のオブジェクトを返します。

実行する手順: メソッドgetEmployeeByFn(String)の実装

getEmployeeByFn(String)**メソッドを実装する手順**:

1. Employeeオブジェクトを返すgetEmployeeByFn(String)を宣言します。

```
public List<Employee> getEmployeeByFn(String fn) throws SQLException {
```

2. List<Employee>型の戻り値の変数を宣言します。

```
List<Employee> returnValue = new ArrayList<>();
```

3. tryブロックを作成します。tryブロックを使用すると、getConnectionメソッドを呼び出してデータベース接続を作成できます。

```
try (Connection connection = getConnection()) {
```

4. PreparedStatementを作成する別のtryブロックを作成します。従業員IDに基づいて従業員を選択するために実行される問合せを追加します。

```
try (PreparedStatement preparedStatement = connection.prepareStatement("SELECT Employee_Id, First_Name,
Last_Name, Email, Phone_Number, Job_Id, Salary FROM EMPLOYEES WHERE First_Name LIKE ?")) {
```

5. 入力パラメータ(従業員の名)を設定します。メソッドの入力パラメータは、PreparedStatementのINパラメータとして設定されます。

```
preparedStatement.setString(1, fn + '%');
```

6. ResultSetのために別のtryブロックを作成します。実行する必要がある問合せを含めます。

```
try (ResultSet resultSet = preparedStatement.executeQuery()) {
```

7. 結果をループするwhile文を含めます。レコードが見つかった場合は、returnValueに追加します。

```
while(resultSet.next()) {
    returnValue.add(new Employee(resultSet));
  }
```

8. すべてのtryブロックのカッコを閉じます。

9. SQLExceptionを捕捉し、ロガーにメッセージを記録します。

```
catch (SQLException ex) {
  logger.log(Level.SEVERE, null, ex);
  throw ex;
}
```

10. メソッドgetEmployee(int)から従業員のリストを返します

```
return returnValue;
```

新しいメソッド*updateEmployee(Employee)*の実装

クラス名: src/main/java/com/oracle/jdbc/samples/bean/EmployeeBeanImpl.java。

Githubの場所: EmployeeBeanImpl.java

説明: メソッドupdateEmployee(Employee)を使用すると、従業員レコードにあるfirst_name、last_nameなどの従業員の詳細を更新できます。

実行する手順: メソッドupdateEmployee(Employee)の実装

UpdateEmployee(Employee)メソッドを実装する手順:

1. メソッドupdateEmployee(Employee)を宣言します。

```
public String updateEmployee(Employee employee) throws SQLException {
```

2. 更新されたレコード数を取得するための変数を宣言して初期化します。

```
int updateCount = 0;
```

3. getConnection()メソッドを呼び出してデータベース接続を確立するtryブロックを作成します。

```
try (Connection connection = getConnection()) {
```

4. PreparedStatementを作成する別のtryブロックを作成します。従業員IDに基づいて従業員を選択するために実行する必要がある問合せを含めます。

```
try (PreparedStatement preparedStatement = connection.prepareStatement("UPDATE employees SET
FIRST_NAME = ?, LAST_NAME = ?, EMAIL = ?, PHONE_NUMBER = ?, SALARY = ? WHERE EMPLOYEE_ID = ?")) {
```

5. ユーザーが属性ごとに入力した新しい値を設定し、prepapredStatementを実行します。

```
preparedStatement.setString(1, employee.getFirst_Name());
preparedStatement.setString(2, employee.getLast_Name());
preparedStatement.setString(3, employee.getEmail());
preparedStatement.setString(4, employee.getPhone_Number());
preparedStatement.setInt(5, employee.getSalary());
   preparedStatement.setInt(6, employee.getEmployee_Id());
updateCount = preparedStatement.executeUpdate();
```

6. すべてのtryブロックのカッコを閉じます。

7. SQLExceptionを捕捉し、次に示すようにロガーにメッセージを記録します。

```
catch (SQLException ex) {
  logger.log(Level.SEVERE, "Unable to update record", ex);
  throw new SQLException("Alert! Record could not be updated, "
    +ex.getMessage(), ex);}
```

8. 更新されたレコードの数とともにメッセージをロガーに記録します。

```
logger.fine("Update count: " +updateCount);
```

9. 更新されたレコードがない場合は、次のようにメッセージを入力します。

```
if (updateCount != 1) {
  logger.severe("Unable to update record");
  throw new SQLException("Alert! Record could not be updated");
}
```

10. レコードが更新された場合は、成功メッセージを返します。

```
return "Success: Record updated";
```

サーブレット(*WebController.java*)へのコードの追加

ここから項目が開始します。

クラス名: src/main/java/com/oracle/jdbc/samples/web/WebController.java

Githubの場所: WebController.java

説明: サーブレットは1日目の演習で作成しました。この項では、すべての従業員に昇給を与えるための関連コードを追加します。

手順6: ユーザーが入力パーセントを入力した後に、給与予算を計算するコードを追加します。これはdoPost()メソッドで処理されます。

コードを**doPost(req,res)**に追加する手順:

1. このメソッドは「従業員レコードの更新」の項ですでに作成されています。

2. 入力に基づいて呼び出される機能を確認する別のifブロックを作成します。

```
if ("incrementSalary".equals(value)) {
  if ((value = request.getParameter(INCREMENT_PCT)) != null)    {
       try {
   System.out.println("increment% = " +value);
   response.setContentType("application/json");
List<Employee> employeeList =    employeeBean.incrementSalary(Integer.valueOf(value));
   System.out.println("incrementSalary, employeeList: " +employeeList.toString());
   gson.toJson(employeeList,
new TypeToken<ArrayList<Employee>>() {}.getType(),
response.getWriter());
  }catch (Exception ea) {          response.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
   }
} else {
response.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
      }
}
```

3. すべてのカッコが正しく閉じていることを確認します。

新しいHTML(*incrementSalary.html*)の作成

クラス名: src/main/webapp/incrementSalary.html。

Githubの場所: incrementSalary.html

説明: このHTMLには、給与予算を計算するためのパーセントを入力する入力ボックスが表示されます。

手順7: HTMLページのtitle、headおよびstylesheetの作成

手順8: 入力フィールド(従業員名など)を送信する関数の作成

手順9: JSONレスポンスを処理して結果をHTMLページに表示するメソッドprocessResponse()を作成します。

手順7: HTMLページのtitle、stylesheetおよびbodyを作成する手順:

1. HTMLページのtitle、stylesheetおよびbodyを作成します。

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Increment Salary</title>
<link rel="stylesheet" type="text/css" href="css/app.css" >
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
<script    src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js">script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
</head>
```

手順8: 入力フィールドを処理する手順

1. 昇給率を取得するための<body>タグと<input>タグを開始します。

```
<body>
<div> Enter the percentage increase in salary<inputid='incrementField' type="number" max="100"
min="3">%
</div>
<div id="UpdateButton"> <button type="button" class="btn btn-info btn-lg"
onclick='javascipt:confirmUpdate()'> Increment Salaries</button> <button type="button" class="btn btn-
default btn-lg" onclick='javascipt:cancelUpdate()'>Cancel</button></div>
<div id="status" class="none"></div>
<div id="id-emp"></div>
<script>
function showStatus(c, message) {
     $('#status').text(message);
     $('#status').attr('class', c);
       }

function confirmUpdate() {
  var increment = $('#incrementField').val();
  var res = confirm("Do you really want to Increment Salary by " +increment +"%?");
  if(res == true) {
    console.log("Salary record");
    $('#UpdateButton').hide();
    showStatus("alert alert-info", "Updating records, processing request");
    var xmlhttp = new XMLHttpRequest();
    var url = "WebController?op=incrementSalary&incrementPct=" +increment;
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        processResponse(xmlhttp.responseText);
        showStatus("alert alert-success", "Updating records, successfully updated");
      }
      else {
        showStatus("alert alert-danger", "Updating records, failure, could not update records");
      }
    }
    xmlhttp.open("POST", url, true);
    xmlhttp.send();
    showStatus("alert alert-info", "Updating records, request sent");

  }
```

```
  else {
    console.log("Salary not updated");
    showStatus("alert alert-warning", "Updating records, attempt cancelled");

  }
}

</script>
```

手順9: processResponse()メソッドを作成する手順:

1. HTMLページにJSON結果を表示するprocessRequest()関数を作成します。

```
unction processResponse(response) {
  var arr = JSON.parse(response);
  var i;
  var out = "<table>";
keys = Object.keys(arr[0]);

  // Print headers
  out += "<tr>"
for(i = 0; i < keys.length; ++i) {
    out += "<th>"+keys[i]+"</th>"
}
out += "</tr>";

// Print values
for(j = 0; j < arr.length; j++) {
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<td>"+arr[j][keys[i]]+"</td>"
}
out += "</tr>"
}
out += "</table>";
document.getElementById("id-emp").innerHTML = out;
}
```

Tomcatでのログイン・ユーザーの作成

HR Webアプリケーションには2人のユーザー(hradminとhrstaff)があります。

資格証明を使用してホーム画面にログインすると、ランディング・ページがWebアプリケーションの詳細とともに表示されます。hradminおよびhrstaffには、異なる権限と異なる機能へのアクセス権があります。

この章では、作成する必要がある必要なクラスと、Tomcatでログイン機能を構築する方法を示します。

- ログイン機能用のXMLファイル(*tomcat-users.xml)*を作成します
- ユーザーをログインするHTMLページ(*login.*html)の作成
- エラー・メッセージを表示するHTMLページ(*login-failed.html*)の作成
- ログイン中にユーザーを認証する*web.xml*の作成
- アプリケーションの詳細を表示するHTMLページ(about.html)の作成
- ランディング・ページ(index.html)の作成と、リダイレクト用のhtmlページの定義
- ログアウトを処理するコードのサーブレット(*WebController.java*)への追加

ログイン機能用のXMLファイル(*tomcat-users.xml)*の作成

クラス名: /java/HRWebApp/tomcat-users.java

Githubの場所: tomcat-users.xml

説明: XMLファイルtomcat-users.xmlを作成し、アクセスを許可するログイン・ユーザーをリストします。ユーザーごとにユーザー名とパスワードの両方を指定します。

手順1: 新しいファイルtomcat-user.xmlの作成

tomcat-users.xmlを作成する手順:

1. ファイルtomcat-users.xmlを作成します。このファイルをTOMCAT_HOME/conf /tomcat-users.xmlの下に配置します。

```
<?xml version='1.0' encoding='utf-8'?> z
<tomcat-users xmlns="http://tomcat.apache.org/xml"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd" version="1.0">
<role rolename="manager"/>
<role rolename="staff"/>
<user username="hradmin" password="welcome" roles="manager,staff"/>
<user username="hrstaff" password="welcome" roles="staff"/>
</tomcat-users>
```

新しいHTML (*login.html*)の作成

クラス名: src/main/webapp/login.html

Githubの場所: login.html

説明: ログイン・ページは、Webアプリケーションのメイン・ページを起動すると表示されます。ログイン・ページには、ユーザー名とパスワードを取得するフィールドが表示されます。

手順2: *login.html*のtitle、headおよびstylesheetの作成

手順3: 入力フィールドを取得してフォームを送信する<body>の作成

手順2: login.htmlのtitle、headおよびstylesheetを作成する手順:

1. ファイルtomcat-users.xmlを作成します。このファイルTOMCAT_HOME/conf/tomcat-users.xmlを配置します。

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Login to Jdbc Web Sample application</title>
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
<style>
#cent {
    position:absolute;
    top:50%;
    left:50%;
    margin-top:-50px; /* this is half the height of your div*/
    margin-left:-100px; /*this is half of width of your div*/
  }
   td {
     height: 30px;
   }

</style>
</head>
```

手順3: <body>を作成してフォームを送信する手順:

1. ユーザーが入力したログイン資格証明を発行する<body>および<form>を作成します。

```
<body>
<div id="cent">
<form method="POST" action="j_security_check">
<table>
<tr>
<td colspan="2">Login to the Jdbc Web Sample application:</td>
</tr>
<td>Name:</td>
<td><input type="text" name="j_username" /></td>
</tr>
<tr>
<td>Password:</td>
<td><input type="password" name="j_password"/></td>
</tr>
<tr>
<td colspan="2"><input type="submit" value="Go" /></td>
</tr>
</table>
</form>
</div>
</body>
```

新しいHTML (*login-failed.html*)の作成

クラス名: src/main/webapp/login-failed.html

Githubの場所: login-failed.html

説明: これは、ログインが失敗した場合に表示されるhtmlページです。

手順4: ログインしようとして失敗したときにメッセージを表示する新しいページ*login-failed.html*の作成

新しいHTMLページlogin-failed.htmlを作成する手順。

1.次に示すようにlogin-failed.htmlを作成します。

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Login Failed</title>
</head>
<body>
<p>
Sorry, login failed!
</p>
</body>
</html>
```

ログイン中にユーザーを認証する*web.xml*の作成

クラス名: src/main/webapp/WEB-INF/web.xml

Githubの場所: web.xml

説明: web.xmlファイルは、ログイン・ページがユーザーに表示されるときにユーザーを認証する記述子で構成されます。

手順: 次に示すように、web.xmlファイルを作成します。

web.xmlを作成する手順

1. web.xmlファイル管理者を作成するには、次の手順を使用します。

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
<display-name>Jdbc Web Sample</display-name>
<security-role>
<role-name>manager</role-name>
</security-role>
<security-role>
<role-name>staff</role-name>
</security-role>
<security-constraint>
<web-resource-collection>
<web-resource-name>Wildcard means whole app requires
authentication</web-resource-name>
<url-pattern>/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>manager</role-name>
</auth-constraint>
<user-data-constraint>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
</security-constraint>
<security-constraint>
<web-resource-collection>
<web-resource-name>Wildcard means whole app requires
authentication</web-resource-name>
<url-pattern>/*</url-pattern>
<http-method>GET</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>staff</role-name>
</auth-constraint>
<user-data-constraint>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
  </security-constraint>
<login-config>
<auth-method>FORM</auth-method>
<form-login-config>
<form-login-page>/login.html</form-login-page>
<form-error-page>/login-failed.html</form-error-page>
</form-login-config>
</login-config>
</web-app>
```

Webアプリケーションを記述するHTMLページ*(about.html)*の作成

クラス名: src/main/webapp/about.html

Githubの場所: about.html

説明: about.htmlファイルには、HRアプリケーション、ユーザーおよび機能に関する情報が表示されます。

手順6: about.htmlをダウンロードして、アプリケーションで使用します。

ランディング・ページ(index.html)の作成と、リダイレクト用のページの定義

クラス名: src/main/webapp/index.html

Githubの場所: index.html

説明: index.htmlファイルは、HR Webアプリケーションに関するすべての詳細で構成されます。ユーザーおよび機能の詳細を説明します。

手順7: *index.html*のtitle、headおよびstylesheetの作成

手順8: リクエストをリダイレクトするHTMLページを呼び出す<body> の作成

7.index.htmlのtitle、headおよびstylesheetを作成する手順:

1. index.htmlファイルを作成します。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Employee table listing</title>
<link rel="stylesheet" type="text/css" href="css/app.css" >
<style>
iframe:focus {
outline: none;
}
iframe[seamless] {
display: block;
}
</style>
</head>
<body>
```

手順8: ボタンに対する<body>、リダイレクトおよびアクションを作成する手順:

1. ナビゲーション・リンクとログアウトを使用して、機能のアクションと<body>を作成します。

```
<body>
<div id="sideNav" class="sidenav">

<a href="javascript:void(0)" class="closebtn" onclick="closeNav()" class="staff">×</a>
<a href="javascript:switchSrc('listAll.html')" class="staff">List All</a>
<a href="javascript:switchSrc('listById.html')" class="staff">Search By Id</a>

<a href="javascript:switchSrc('listByName.html')" class="manager">Update Employee Record</a>
<a href="javascript:switchSrc('incrementSalary.html')" class="manager">Increment Salary</a>
<a href="javascript:switchSrc('about.html')">About</a>
</div>
<div id="main">
<div align="right">
<div
id="myrole"
        style="display:inline; color:#393318; display: block; background-color:#eff0f1;position:
absolute; top: 20px; right: 8%;"
    >myrole</div>
<a href="javascript:void(0)"
       onclick="logout()"
       class="staff"
       style="display: block; position: absolute; top: 20px; right: 1%">Logout</a>
```

```
</div>
<div>
<span style="font-size:30px;cursor:pointer" onclick="openNav()"> Java 2 Days HR Web Application
</span>
</div>
<div>
<iframe id="content"
src="about.html"
frameborder="0"
style="overflow:hidden; height:100%; width:100%"
height="100%"
width="100%"></iframe>
</div>
</div>
<script>
function openNav() {
  document.getElementById("sideNav").style.width = "256px";
  document.getElementById("main").style.marginLeft = "256px";
}

function closeNav() {
  document.getElementById("sideNav").style.width = "0";
  document.getElementById("main").style.marginLeft= "0";
}

function switchSrc(src) {
  document.getElementById('content').src = src;
}

function logout() {

  var xmllogout = new XMLHttpRequest();
  xmllogout.open("GET", "WebController?logout=true", true, "_", "_");
  xmllogout.withCredentials = true;
  // Invlalid credentials to fake logout
  xmllogout.setRequestHeader("Authorization", "Basic 00001");
  xmllogout.send();

  xmllogout.onreadystatechange = function() {
    window.location.replace("index.html");
  }

 return true;
}

var xmlhttp = new XMLHttpRequest();
var url = "getrole";

xmlhttp.onreadystatechange = function() {
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    role = xmlhttp.responseText;
    console.log("role: " +role);
    if (role == "staff") {
      console.log ("disabling manager");
      var x = document.getElementsByClassName('manager');
      for(i = 0; i < x.length; ++i) {
        x[i].style.display = 'none';
      }
    }
    document.getElementById('myrole').innerHTML = ' '+role+' ';
```

```
  }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
</script>
</body>
```

ログアウトを処理するコードのサーブレット(*WebController.java*)への追加

クラス名: src/main/java/com/oracle/jdbc/samples/web/WebController.java

Githubの場所: WebController.java

説明: このサーブレットは1日目の演習で作成しました。この手順では、ログアウトおよびログインを実装する関連コードを追加します。

手順9: ログ・アウト機能に対するメソッドprocessRequest(req, res) **の更新**

メソッド processRequest (req, res)を更新する手順:

1. これまでの手順で"ListAll"機能にメソッドを作成しました。この手順では、logout機能のコードを追加します。

2. 入力に基づいて呼び出す機能を検証するifブロックを作成します。入力値が'LOGOUT'であるかどうかを確認します。次に、関連するメソッドを呼び出してユーザーをログ・アウトします。

```
if ((value = request.getParameter(LOGOUT)) != null) {
/* Getting session and then invalidating it */

 HttpSession session = request.getSession(false);
if (request.isRequestedSessionIdValid() && session != null)    {
  session.invalidate();
}
handleLogOutResponse(request,response);
   response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
return;
}

private void handleLogOutResponse(HttpServletRequest request,
HttpServletResponse response) {
    Cookie[] cookies = request.getCookies();
    for (Cookie cookie : cookies) {
      cookie.setMaxAge(0);
      cookie.setValue(null);
      cookie.setPath("/");
      response.addCookie(cookie);

    }
  }
```

3. すべてのカッコが正しく閉じられていることを確認します。

8 ベスト・プラクティス

1.クラウド上のデータベース・サービスを使用します。

Oracle Database Service on Cloud (DBCS)を使用して、クラウド上にデータベースを作成します。DBCSには、組込みHRスキーマと、HR Webアプリケーションの構築時に使用できる表が付属しています。

注意:

最新の Oracle Database の特性および機能を使用するには、Oracle Database Enterprise Edition 12.2.0.1 を使用します。

2.JDBCドライバ、UCP:

最新の12.2.0.1バージョンのJDBCドライバおよびUCPを使用することをお薦めします。

注意:

12.2.0.1 JDBCドライバおよびUCPから最新のJDBCドライバおよびUCPをダウンロードします

3.JDK 8

JDK 8に準拠する最新バージョンのOracle JDBCドライバ12.2.0.1を使用することをお薦めします。

4.自動クローズ可能文

JDK7以降、明示的なcatch文なしでデフォルトで閉じる自動クローズ可能文が導入されました。

5.Statementオブジェクトのかわりに PreparedStatementを使用します:

JDBCの文は、DDL(ALTER、CREATE、GRANTなど)での使用のためにローカライズする必要があります。これらのコマンドは、バインド変数を受け入れることができないためです。

他のタイプの文には、PreparedStatementまたはCallableStatementを使用します。これらの文ではバインド変数を使用できます。

9 トラブルシューティングおよびデバッグ

1. Tomcatログ・ファイル:

アプリケーションのデプロイ後に、TOMCAT_HOME/logs/catalina.outでエラーを確認します。

2. 追加ロギング:

Tomcatのロギングを有効にして、コード内のロギング・メッセージを検索します。

注意:

詳細は、https://tomcat.apache.org/tomcat-8.0-doc/logging.html を参照してください。

UI関連の問題のデバッグ

1. ブラウザのバージョン:

このアプリケーションは、Firefox (バージョン52)およびChrome (バージョン58)で正常にテストされています。

2. ブラウザ・コンソール:

ブラウザ・コンソールで、問題を検索してデバッグするエラーを探します。

3. ネットワーク・トラフィックの監視:

ネットワーク・トラフィックを追跡して、作成されているリクエストおよびリクエストに対するレスポンスを探します。400を超える戻りステータスはエラーを示します。400から500の範囲でエラーが見つかった場合は、デバッグのためにTomcatログ・ファイルを調べます。

バックエンド・サーバーへの様々なコールからのJSONレスポンスを調べます。

4.追加ロギング:

パッケージ化されたHTMLファイルを編集し、console.log(...)を追加してロギングを追加します。

索引

E H I J O S W

E

- SQL-92のエントリ・レベル 1

H

- HRアカウント
  - テスト 1
- HRユーザー・アカウント
  - サンプル・アプリケーション 1
  - ロック解除 1

I

- IDE 1
  - Oracle JDeveloper 1
- インストール
  - データベースでの検証 1
- 統合開発環境 1

J

- J2SE 1
  - インストール 1
- Java Database Connectivity 1
- JavaServer Pages 1
- JDBC 1
- JSP 1
- JSPページ
  - デプロイ 1

O

- Oracle Database 12cリリース2 1
  - インストール 1
  - インストレーション・ガイド 1
  - リリース・ノート 1
  - 検証 1

    - インストールの検証 1
- Oracle Database 12c リリース2のインストール
    - プラットフォーム固有 1
- Oracle WebLogic Server 1

S

- サンプル・アプリケーション
    - HRユーザー・アカウント 1

W

- Webサーバー 1
    - Apache Tomcat 1
    - サーブレット・コンテナ 1