

Visual COBOL 7.0J for Windows

Amazon Aurora PostgreSQL 動作検証 検証結果報告書

2022年5月19日作成

昨今、様々な業界でクラウドを利用したシステムの開発・運用が進められており、COBOL で開発された基幹システムにおいても、クラウド利用の波は押し寄せています。

クラウドを利用することによるハードウェアコストの削減だけではなく、クラウドが提供する各種サービスの活用も利点の1つです。例えば、マネージド型のデータベースサービスは、容易に、データベースの稼働・バックアップなどが行えるサービスであり、クラウド上で稼働するシステムで多く利用されています。

では、データベース連携を伴う COBOL で開発されたシステムのクラウド利用を考える際、このようなマネージド型のデータベースサービスを利用できるのでしょうか。実は、マネージド型のデータベースサービスとは、ODBC, JDBC といった標準仕様に基づいた接続が行えるため、COBOL で開発されたシステムでも、これまでオンプレミス環境で利用してきた接続方式で、マネージド型のデータベースサービスを利用することができます。

この文書は、Micro Focus Visual COBOL が提供する各種データベース接続機能を利用し、マネージド型のデータベースサービスの1つである、Amazon Web Services (AWS) が提供する Amazon Aurora との接続性を検証した結果報告書です。

目次

1. 検証概要、目的及びテスト方法	1
1) 検証概要	1
2) 目的及びテスト方法	1
2. 検証環境	2
3. テスト内容	3
1) ODBC 接続	3
2) JDBC 接続	3
3) XA リソースを介した ODBC 接続	3
4. 結果	4
1) インストール	4
2) サンプルアプリケーションの作成	5
3) サンプルアプリケーションの実行結果	6
5. テスト結果及び考察	6
6. 付録 サンプルアプリケーション実行結果	7
1) ODBC 接続	7
2) JDBC 接続	12
3) XA リソースを介した ODBC 接続	16

1. 検証概要、目的及びテスト方法

1) 検証概要

Amazon Aurora は、AWS (Amazon Web Services) が開発したフルマネージド型のリレーショナルデータベースエンジンです。昨今のトレンドである各種クラウドサービスを活用したシステム構築・運用によって、Amazon Aurora を用いた業務システムの運用も増加しています。

Visual COBOL に付属する OpenESQL プリプロセッサは、COBOL プログラムに記述された埋め込み SQL 文より ODBC ドライバ、JDBC ドライバ、ADO.NET データプロバイダを経由した様々なリレーショナルデータベースアクセスを提供します。このリレーショナルデータベースのアクセス先には、既存のデータベース製品だけではなく、Amazon Aurora といったクラウドが提供するマネージドデータベースサービスも含まれます。本稿では、この OpenESQL を使って ODBC 並びに JDBC 経由で、埋め込み SQL 文を含む COBOL プログラムから Amazon Aurora PostgreSQL へアクセスできることを動作検証しました。併せて、ODBC 用の XA スイッチモジュールを使って Visual COBOL に付属する COBOL 専用のアプリケーションサーバーで COBOL アプリケーションがコンテナ管理サービスとして Amazon Aurora PostgreSQL と連携できることも動作検証しました。

2) 目的及びテスト方法

Micro Focus Visual COBOL は最新鋭の COBOL 言語開発・実行環境を提供しています。COBOL 言語への埋め込み SQL 処理系を標準装備しており、ODBC ドライバ、JDBC ドライバ、ADO.NET データプロバイダを経由した様々なデータベースへのアクセスを可能とする OpenESQL プリプロセッサを搭載しています。OpenESQL を使うことで、ODBC については ODBC 3.x 仕様に、JDBC であれば JDBC 4.0 仕様に準拠したデータソースに対して設計上問題なくアクセスすることができます。本稿執筆時点における Micro Focus Visual COBOL の最新版 7.0J で動作保証されている PostgreSQL 11.15 互換の Amazon Aurora PostgreSQL に対して COBOL プログラムより日本語を含むデータを正しく操作できることを検証しました。ODBC 経由のアプリケーションについては、Windows のネイティブコードにコンパイルされた動的ロードモジュールより処

理を実行しています。更に同様の処理をする動的ロードモジュールを COBOL 専用のアプリケーションサーバーである Enterprise Server インスタンスにコンテナ管理のサービスとして配備し、ODBC を経由した XA 準拠リソースマネージャを介して Amazon Aurora PostgreSQL を操作できることを確認しています。JDBC 経由で接続するアプリケーションは、Java バイトコードにコンパイルし、JVM クラスとして実行して動作を確認しています。

2. 検証環境

> [EC2 インスタンス \(Windows クライアント\) 環境](#)

OS	Windows Server 2019 Base
インスタンスタイプ	t2.large (2 vCPU, 8GiB)
COBOL 開発環境製品	Micro Focus Visual COBOL 7.0J for Windows Patch Update 7
JDK/JRE	Adopt OpenJDK 8.0.212.03-hotspot
ソフトウェア	PostgreSQL ODBC ドライバ 64bit 版 11.01.00.00 Simple-JNDI 0.11.4.1 PostgreSQL JDBC ドライバ 42.3.5 JBoss Application Server 6 Version: 7.3.7 データベース内の登録データ参照目的で以下を利用 pgAdmin4-5.0

> [RDS \(Amazon Aurora PostgreSQL\)](#)

インスタンスクラス	db.r6g.large
エンジンバージョン	11.15
ネットワーク	パブリックアクセスなし EC2 環境から PostgreSQL 通信のみ許可

3. テスト内容

以下の内容が実施したテストの概要です。

1) ODBC 接続

COBOL プログラム中に CREATE TABLE 文を埋め込み SQL 文として記述し、テスト用のテーブルを作成します。続いて、INSERT 文によるデータの登録、UPDATE 文によるデータの編集を行います。INSERT 文、UPDATE 文の後には COMMIT 文を入れそれぞれのトランザクションを確定させます。扱うデータには日本語を含めます。反映したデータは CURSOR - FETCH して取り出し、データを確認します。最後に DROP TABLE 文を使って作成したテーブルを削除します。これにより、DDL 文、DML 文、DCL 文の正常動作並びに日本語データの正常なハンドリングを検証します。

2) JDBC 接続

1) で利用するプログラムソースの接続方式を JDBC に変更したプログラムを利用し、JDBC 接続上でも、同じロジックが正常に処理できることを検証します。

3) XA リソースを介した ODBC 接続

Java EE アプリケーションより EJB として呼び出される COBOL プログラムにて SELECT 文及び DML 文を発行し、正常に Amazon Aurora PostgreSQL データベースを操作できることを検証します。COBOL プログラムは COBOL 専用のアプリケーションサーバー Enterprise Server インスタンスに配備し、Java EE アプリケーションより JCA の技術を使って EJB 呼び出しさせます。この COBOL プログラムはコンテナ管理のサービスとしてエクスポートさせます。そのため、プログラム中に CONNECT 文、DISCONNECT 文、DCL 文は記述せず、これらの管理は Transaction Manager(この場合は、Enterprise Server インスタンス) に委譲します。アプリケーションの処理完了後は、正しくトランザクションが完結していることを別途確認します。更に、DML 文実行後、敢えて実行時エラーを発生させるようなコードを埋め込んだ COBOL プログラムに差し替え、実行時エラー発生時は正しく Rollback されることを確認します。

4. 結果

1) インストール

- Amazon Aurora PostgreSQL

今回は動作検証が目的であるため、以下のように「開発/テスト」テンプレートを選択して構築しました。

テンプレート
お客様のユースケースに合わせてサンプルテンプレートを選択します。

<input type="radio"/> 本番稼働用 高い可用性と、高速で安定したパフォーマンスのために、デフォルト値を使用します。	<input checked="" type="radio"/> 開発/テスト このインスタンスは本番稼働環境ではない開発で使用します。
--	---

- PostgreSQL ODBC ドライバ

以下のリンク先よりソースをダウンロードし、インストールしました。

ダウンロード元(2022/5/19 リンク検証) :

<http://www.postgresql.org/ftp/odbc/versions/msi/>

- PostgreSQL JDBC ドライバ

以下のリンク先よりソースをダウンロードし、インストールしました。

ダウンロード元(2022/5/19 リンク検証) :

<https://jdbc.postgresql.org/download.html>

- Simple-JNDI

以下のリンク先よりダウンロードし、インストールしました。

ダウンロード元(2022/5/19 リンク検証) :

<https://mvnrepository.com/artifact/simple-jndi/simple-jndi/0.11.4.1>

- JBoss Enterprise Application Server 7.3.7

以下のリンク先¹よりダウンロードしてインストールしました。

ダウンロード元(2022/5/19 リンク検証) :

<https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?downloadType=distributions&product=appplatform&version=7.3>

¹ アクセスには Red Hat アカウントが必要です。

2) サンプルアプリケーションの作成

本検証で用意したプログラムの処理フローを以下に記します。実際のプログラムコードを、Micro Focus のウェブサイト上に本報告書と共に公開しています。

- ODBC および JDBC 接続検証に使用したプログラムフロー
 - ① Amazon Aurora PostgreSQL データベースに接続
 - ② CREATE TABLE 文にてテスト用のテーブルを作成
 - ③ INSERT 文にて日本語を含まないデータを挿入
 - ④ INSERT 文にて日本語を含むデータを挿入
 - ⑤ COMMIT 文を発行してデータ挿入のトランザクションをコミット
 - ⑥ UPDATE 文にて日本語を含むデータを編集
 - ⑦ COMMIT 文を発行してデータの変更をコミット
 - ⑧ DECLARE CURSOR 文にてテスト用のテーブルを参照するカーソルを定義
 - ⑨ FETCH 文にてデータを取得
 - ⑩ DROP TABLE 文にてテスト用に作成したテーブルを削除
 - ⑪ Amazon Aurora PostgreSQL データベースとの接続を切断
- XA リソースを介した ODBC 接続検証に使用したアプリケーションの処理フロー
 - [パターン1：正常処理]
 - ① Amazon Aurora PostgreSQL データベースに接続
 - ② COBOL サービスを呼び出し(Java EE)
 - ③ SELECT 文を実行し変更対象のデータを取得(COBOL)
 - ④ 変更対象のデータを受け取ったパラメータに基づき更新(COBOL)
 - ⑤ UPDATE 文を実行(COBOL)
 - ⑥ SELECT 文を実行(COBOL)
 - ⑦ SELECT 文で取得したデータを戻りパラメータにセット(COBOL)
 - ⑧ COBOL より返された値をブラウザに表示(Java EE)
 - [パターン2：エラー処理]
 - ① Amazon Aurora PostgreSQL データベースに接続
 - ② COBOL サービスを呼び出し(Java EE)
 - ③ SELECT 文を実行し変更対象のデータを取得(COBOL)
 - ④ 変更対象のデータを受け取ったパラメータに基づき更新(COBOL)
 - ⑤ UPDATE 文を実行(COBOL)

⑥ 実行時エラー発生(COBOL)

それぞれの処理実行前後でテスト対象のレコードを確認します。

3) サンプルアプリケーションの実行結果

Windows のネイティブアプリケーション並びに JVM クラスとして生成したサンプルアプリケーションを正常に実行できることを確認しました。また、XA リソースを介した ODBC 接続においても Enterprise Server インスタンスが Transaction Manager として正しく接続及びトランザクションを管理し、COBOL プログラムも正しく連携できていることを確認しました。検証の実行手順等の詳細は付録の通りとなります。

5. テスト結果及び考察

Visual COBOL を使用してネイティブコードにコンパイルした、埋め込み SQL 文を含む COBOL プログラムから ODBC 経由で Amazon Aurora PostgreSQL データベースに接続し、DDL 文、DML 文、DCL 文を発行してデータベースが操作できることを検証しました。Java バイトコードにコンパイルし、JVM クラスとした場合も同様に JDBC 経由にて正しく操作できることを確認しました。日本語を含んだデータについても両者ともに正常に扱えることも検証できました。

XA リソースを介した ODBC 接続においても手続き型の COBOL プログラムを維持したまま Java EE アプリケーションの一部として正しく動作し、Enterprise Server インスタンスに接続及びトランザクションの管理を委譲できることを確認しました。この結果より、Amazon Aurora PostgreSQL データベースと連携する場合であっても、Java の開発者は Java 側のロジックを、COBOL の開発者は COBOL のビジネスロジックの構築に専念するという従来からの弊社製品の利用者が採用する COBOL - Java EE 連携における開発スタイルを適用できることが裏付けられました。

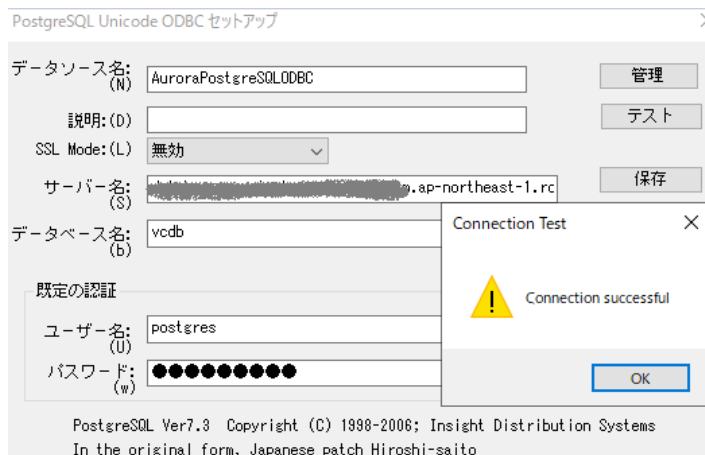
以上

6. 付録 サンプルアプリケーション実行結果

1) ODBC 接続

I. Amazon Aurora PostgreSQL 用の ODBC データソースを用意

- ① スタートメニューより ODBC データソースアドミニストレータ(64 ビット) を起動
- ② [追加] ボタンを押下
- ③ 「PostgreSQL Unicode(x64)」を選択し [完了] ボタンを押下
- ④ サーバー名、データベース名、ユーザ名、パスワード等必要な項目を入力
- ⑤ [テスト] ボタンを押下し、正常に構成できていることを確認



- ⑥ [保存] ボタンを押下し、構成内容を反映

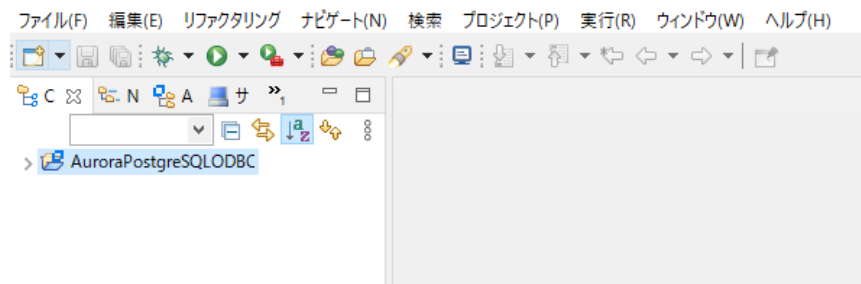
II. スタートメニューより Eclipse を起動

III. COBOL プロジェクトを作成

[ファイル]メニュー > [新規] > [COBOL プロジェクト]

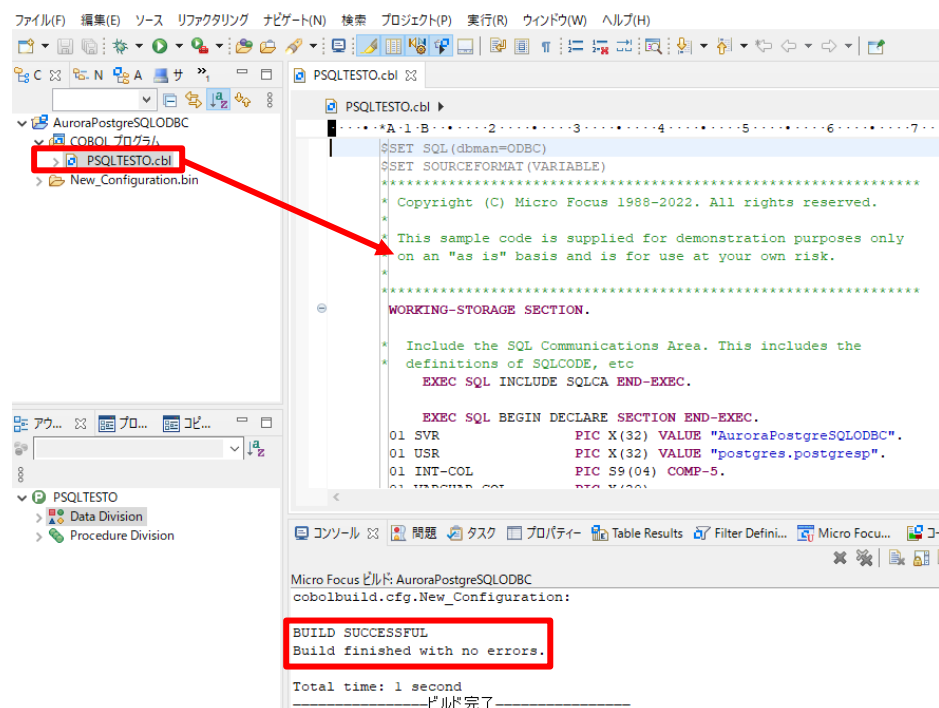
なお、プロジェクトテンプレートには、64 ビットを選択しました。

プロジェクト作成後の画面：



IV. サンプルプログラム PSQLTESTO.cbl をプロジェクトにインポート

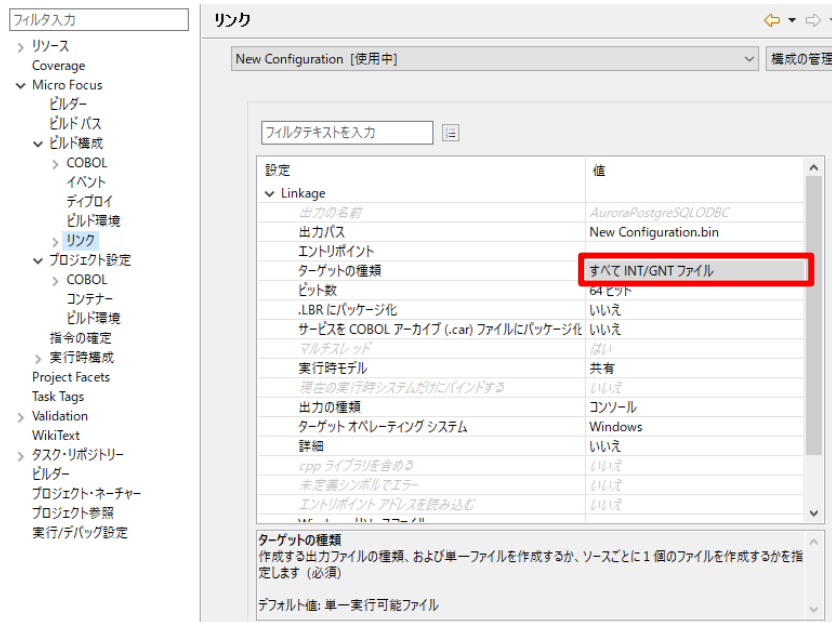
- ① COBOL エクスプローラーにてプロジェクトを右クリックし、
[インポート] > [インポート] をクリック
- ② [一般] > [ファイル・システム] を選択し [次へ] ボタンを押下
- ③ [参照] ボタンを押下し、サンプルプログラムが格納されているフォルダへ移動
- ④ サンプルプログラムにチェックを入れ、[完了] ボタンを押下
プログラムがプロジェクトに追加されると同時にビルド処理がキックされ、コンパイルされます。



V. ビルドのターゲットを動的ロードモジュール(.gnt)に変更

- ① COBOL エクスプローラーにてプロジェクトを右クリックし、
[プロパティ] を選択
- ② [Micro Focus] > [ビルド構成] > [リンク] へとナビゲート

③ ターゲットの種類を「すべて INT/GNT ファイル」に設定

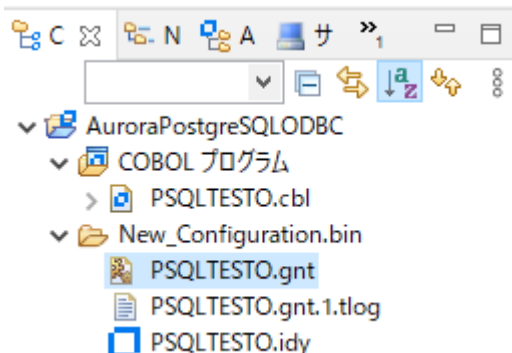


④ [Micro Focus] > [プロジェクト設定] > [COBOL] へとナビゲート

⑤ .GNT にコンパイルを「はい」に設定

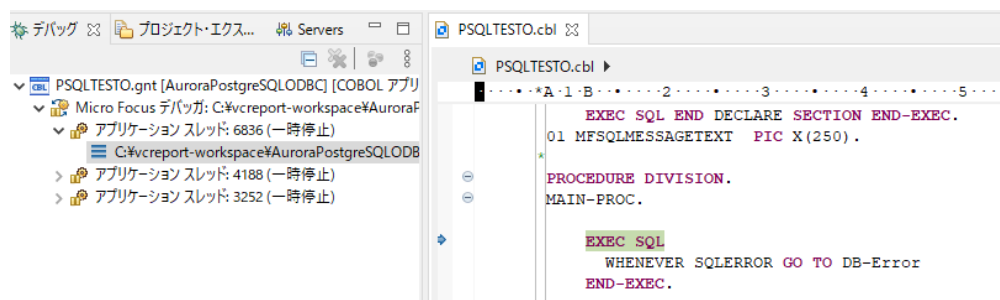


VI. 生成されたモジュールを確認

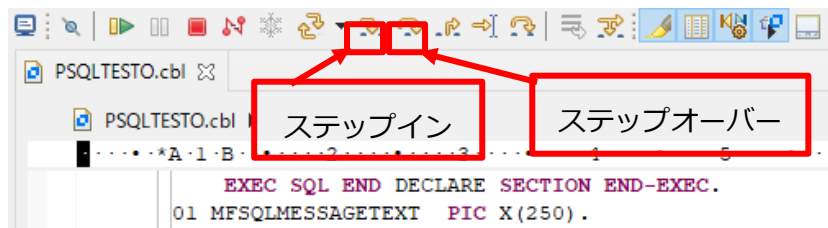


VII. 生成された動的ロードモジュール PSQLTESTO.gnt をデバッグ実行

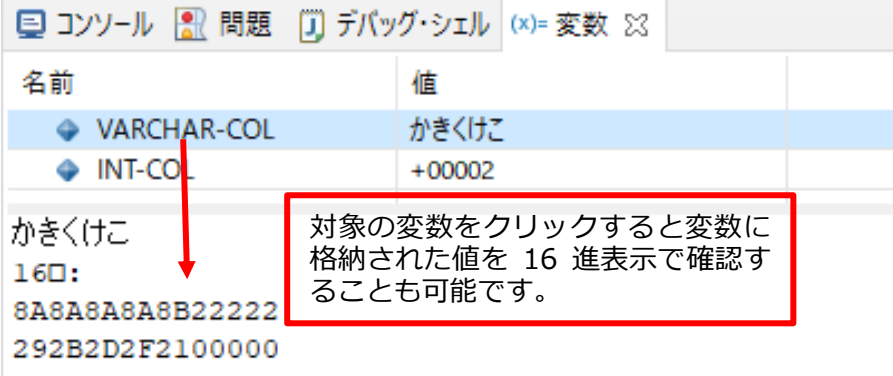
- ① COBOL エクスプローラーにて PSQLTESTO.gnt を右クリックし、
[デバッグ] > [COBOL アプリケーション] を選択
- ② 「パースペクティブ切り替えの確認」のプロンプトには [はい] を選択
- ③ デバッグパースペクティブに切り替わり最初の COBOL 文の実行前で処理が
止まっています。



- ④ COBOL 用に作りこまれたデバッガの機能を駆使してデバッグ実行
ステップイン(CALL 文や、PERFORM 文で実行する先の中までステップを進
める機能)、ステップオーバー(CALL 文や PERFORM 文の先までステップを
進めず、それらを 1 ステップとして実行)を使って、ステップ単位で処理を進
めます :



変数ビューでは COBOL の変数及びホスト変数に格納された値をウォッチできます：

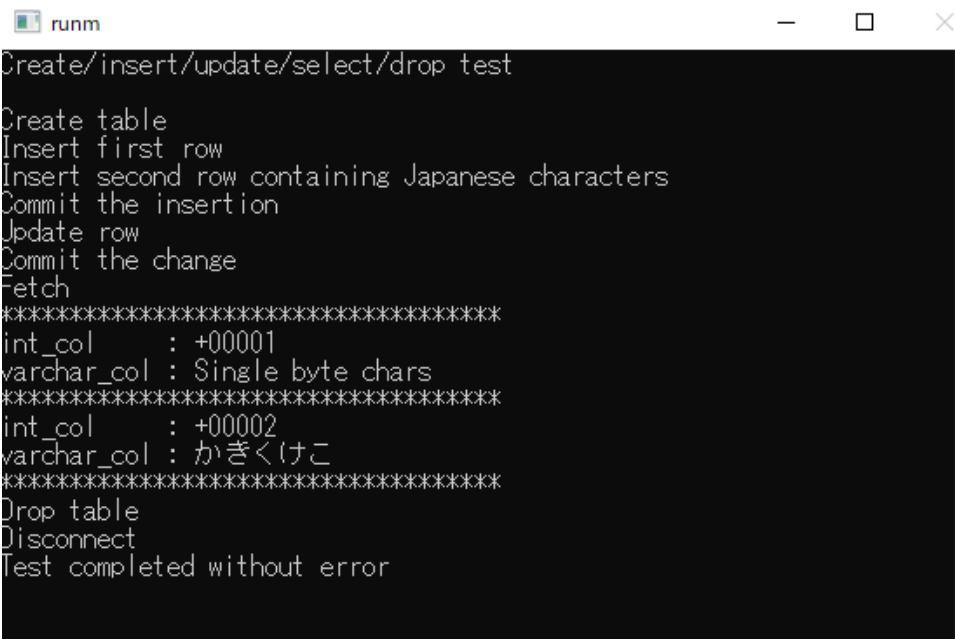


名前	値
◆ VARCHAR-COL	かきくけこ
◆ INT-COL	+00002

かきくけこ
160:
8A8A8A8A8B222222
292B2D2F2100000

対象の変数をクリックすると変数に格納された値を 16 進表示で確認することも可能です。

⑤ サンプルアプリケーションが正常に実行されたことを確認



```
runm
Create/insert/update/select/drop test
Create table
Insert first row
Insert second row containing Japanese characters
Commit the insertion
Update row
Commit the change
Fetch
*****
int_col      : +00001
varchar_col  : Single byte chars
*****
int_col      : +00002
varchar_col  : かきくけこ
*****
Drop table
Disconnect
Test completed without error
```

2) JDBC 接続

I. jndi.properties ファイルを作成

```
C:¥SimpleJNDI>type jndi.properties
java.naming.factory.initial=org.osjava.sj.SimpleContextFactory
org.osjava.sj.root=C:/SimpleJNDI/AurorapostgreSQL

C:¥SimpleJNDI>
```

※osjava.sj.root には jndi.properties が格納されるフォルダを指定

II. AmazonAuroraPostgreSQL を利用する Data Source 用の properties ファイルを作成

```
C:¥SimpleJNDI¥AurorapostgreSQL>type pg.properties
type=javax.sql.DataSource
url=jdbc:postgresql://AuroraPostgreSQLRDS.ap-northeast-1.rds.amazonaws.com:5432/vcdb
driver=org.postgresql.Driver
user=postgres
password=passwordOfPostgreSQL

C:¥SimpleJNDI¥AurorapostgreSQL>
```

※url には AmazonAuroraPostgreSQL のエンドポイントを指定

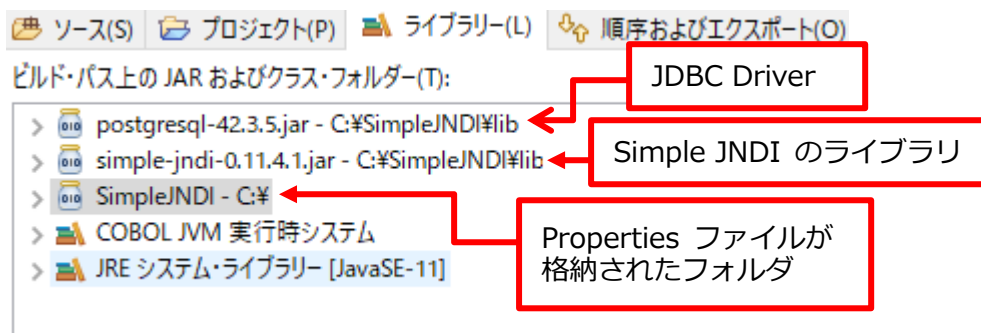
III. COBOL JVM プロジェクトを作成

IV. サンプルプログラム PSQLTESTJ.cbl をプロジェクトにインポート

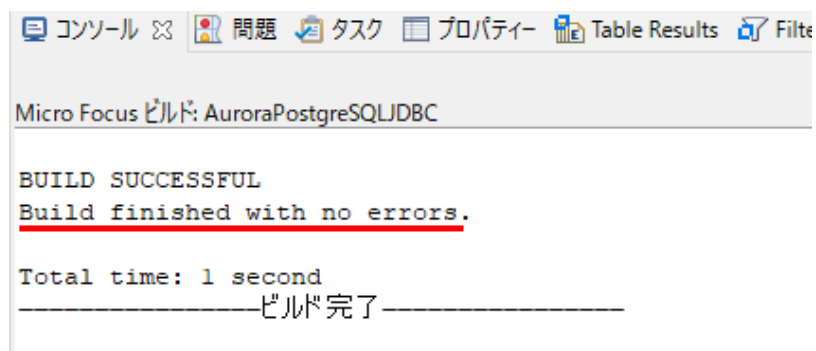
- ① COBOL エクスプローラーにて src フォルダを右クリックし、
[インポート] > [インポート] へとナビゲート
- ② [一般] > [ファイル・システム] を選択し [次へ] ボタンを押下
- ③ [参照] ボタンを押下し、付録 1 で利用したサンプルプログラムが格納されているフォルダへ移動
- ④ サンプルプログラムにチェックを入れ、[完了] ボタンを押下

V. CLASSPATH に JDBC Driver 及び Simple JNDI のライブラリを追加

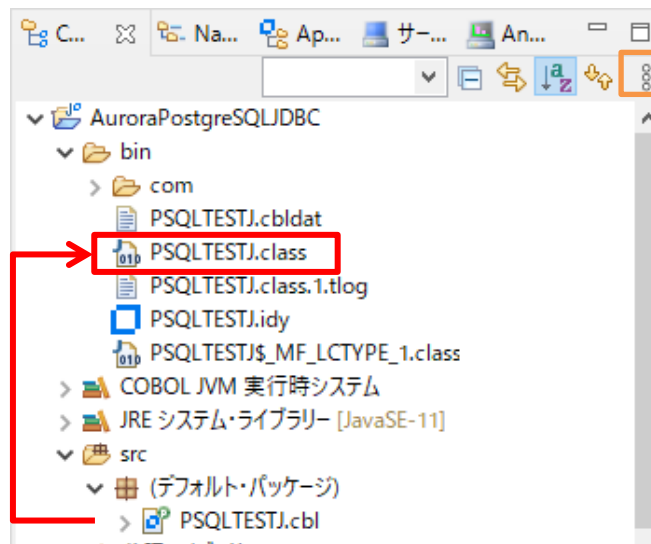
- ① COBOL エクスプローラーにてプロジェクトを右クリックから [プロパティ] を選択
- ② [Micro Focus] > [JVM ビルドパス] へとナビゲート
- ③ [ライブラリー] タブを選択
- ④ [外部 JAR の追加] ボタンを押下し、インストールした JDBC ドライバを選択
- ⑤ [外部 JAR の追加] ボタンを押下し、Simple JNDI のライブラリを選択
- ⑥ [外部クラス・フォルダーの追加] ボタンを押下し、I. で用意した properties ファイルが格納されたフォルダを追加



- ⑦ [OK] ボタンを押下しアプリケーションをビルド



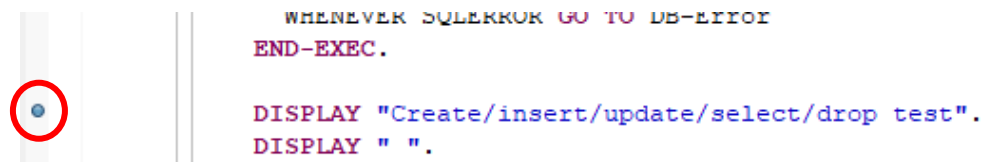
VI. COBOL エクスプローラーにて COBOL プログラムに対するクラスファイルが生成されていることを確認



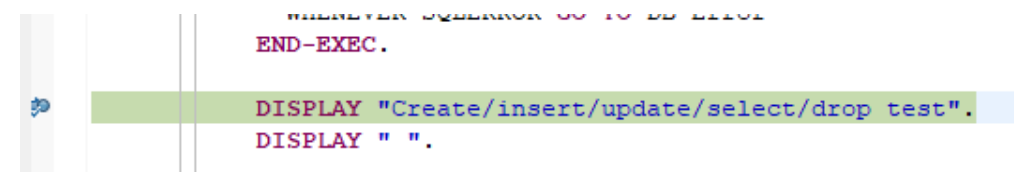
※ bin フォルダを表示するために、橙箇所のアイコンより[フィルタとカスタマイズ]を選択し、“COBOL JVM 出力フォルダ” のチェックを外す必要があります。

VII. 生成されたクラスファイルを Java アプリケーションとしてデバッグ実行

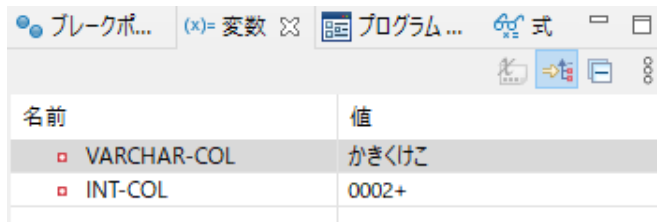
- ① 最初の COBOL 文にブレークポイントを指定



- ② COBOL エクスプローラーにて対象のプログラム PSQLTESTJ.cbl を右クリックから [デバッグ] > [Java アプリケーション] を選択
- ③ パースペクティブの切り替えの確認には [はい] を選択
ブレークポイントを指定した文の実行前で停止しています :

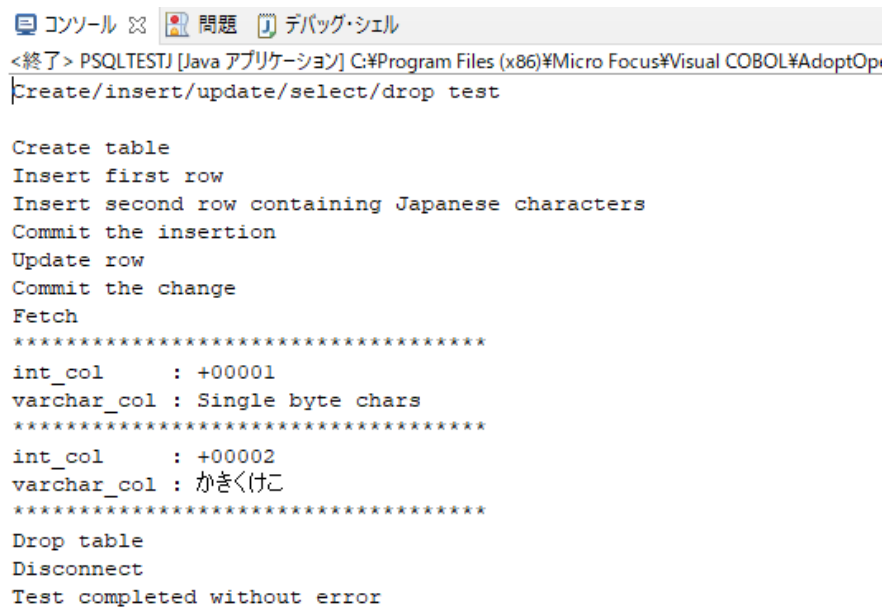


- ④ ODBC 接続と同じ要領でデバッガを使ってデバッグ実行
ここでも変数のウォッチ機能等を利用できます：



名前	値
VARCHAR-COL	かきくけこ
INT-COL	0002+

- ⑤ 正常に処理されたことをコンソールビューで確認



```
コンソール 問題 デバッグ・シェル
<終了> PSQLTESTJ [Java アプリケーション] C:\Program Files (x86)\Micro Focus\Visual COBOL\AdoptOp
Create/insert/update/select/drop test

Create table
Insert first row
Insert second row containing Japanese characters
Commit the insertion
Update row
Commit the change
Fetch
*****
int_col      : +00001
varchar_col  : Single byte chars
*****
int_col      : +00002
varchar_col  : かきくけこ
*****
Drop table
Disconnect
Test completed without error
```

3) XA リソースを介した ODBC 接続

I. ODBC 用の XA スイッチモジュールを作成

- ① スタートメニューより「Visual COBOL コマンドプロンプト(64-bit)」を起動
- ② <製品のインストールフォルダ>%src%enterpriseserver
配下の xa フォルダを任意のフォルダへコピー
- ③ XA スイッチモジュールをビルド

```
C:%xa>build odbc
Building 64-bit switch module...
Micro Focus COBOL
Version 7.0 (C) Copyright 1984-2022 Micro Focus or one of its
affiliates.
* チェック終了 : エラーはありません- コード生成を開始します
* Generating ESODBCXA
* Data:           16      Code:           56482      Literals:
4128
Micro Focus COBOL - CBLINK utility
Version 7.0.0.98 (C) Copyright 1984-2022 Micro Focus or one of its
affiliates.
Microsoft (R) Incremental Linker Version 14.16.27045.0
Copyright (C) Microsoft Corporation. All rights reserved.
ESODBCXA.obj
cbllds0000187C.obj
ライブラリ ESODBCXA.lib とオブジェクト ESODBCXA.exp を作成中
Microsoft (R) Manifest Tool
Copyright (c) Microsoft Corporation.
All rights reserved.
1 個のファイルをコピーしました。
Micro Focus COBOL
Version 7.0 (C) Copyright 1984-2022 Micro Focus or one of its
affiliates.
* チェック終了 : エラーはありません- コード生成を開始します
```

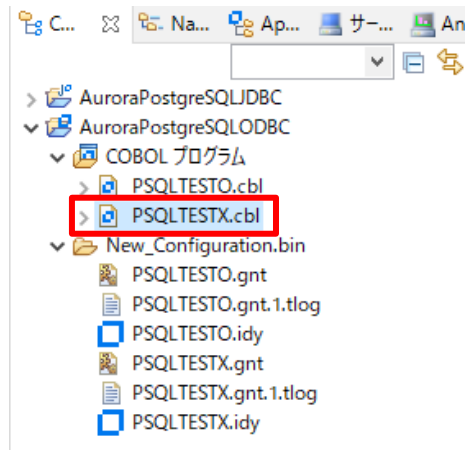
```
* Generating ESODBCXA_D
* Data:          16      Code:          63796      Literals:
4400
Micro Focus COBOL - CBLLINK utility
Version 7.0.0.98 (C) Copyright 1984-2022 Micro Focus or one of its
affiliates.
Microsoft (R) Incremental Linker Version 14.16.27045.0
Copyright (C) Microsoft Corporation. All rights reserved.

ESODBCXA_D.obj
cbllds00001BE8.obj
ライブラリ ESODBCXA_D.lib とオブジェクト ESODBCXA_D.exp を作成
中
Microsoft (R) Manifest Tool
Copyright (c) Microsoft Corporation.
All rights reserved.

C:¥xa>
```

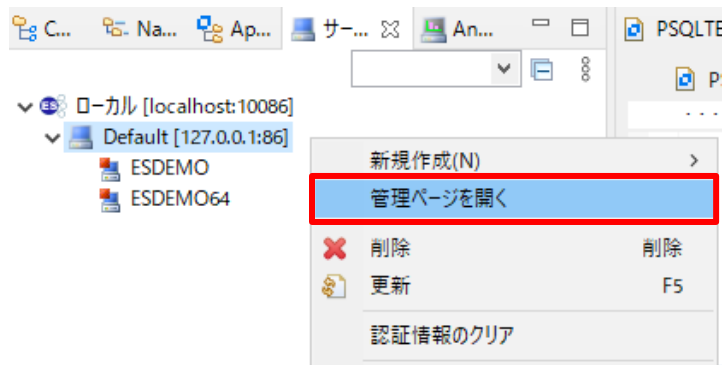
II. ODBC 接続で利用したプロジェクトに本検証で利用するサンプルプログラム
PSQLTESTX.cbl をインポート

III. 正しくコンパイルされ動的ロードモジュールにビルドされていることを確認



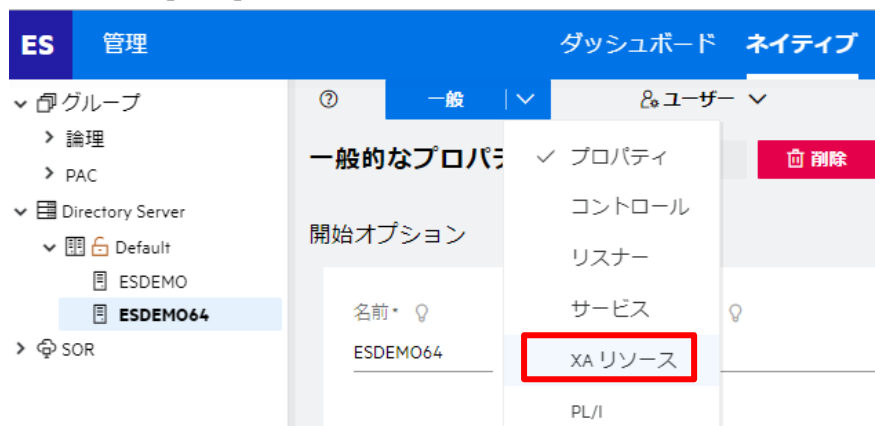
- IV. サーバーエクスプローラーにて Default [127.0.0.1:86] を右クリック
[管理ページを開く] を選択し

Enterprise Server Common Web Administration(ESCWA) ページを開く



- V. Enterprise Server インスタンスに XA リソースを構成

- ① ESDEMO64 をクリック
- ② 画面上部の [一般] ドロップダウンをクリックし、“XA リソース” を選択



- ③ [* 新規作成] ボタンを押下
- ④ 構成情報を入力し、[保存] ボタンを押下

XA リソースの構成

ID* 名前*

モジュール* I. で作成したスイッチモジュール

有効 再接続試行

OPEN 文字列 ODBC データソース名

CLOSE 文字列

保存 戻る

- ⑤ 正しく追加されていることを確認

XA リソース |

ID	名前	有効	再接続試行	OPEN 文字列
AURORAPO	AuroraPost...	✓	1	DSN=Auror...

VI. 動的デバッグを受け付けるよう Enterprise Server インスタンスを構成

- ① ESCWA 画面上部の [一般] ドロップダウンメニューをクリック
- ② [動的デバッグを許可] にチェックし、[適用] ボタンを押下

一般的なプロパティ |

開始オプション

名前 システムディレクトリ

共有メモリ ページ数 ページ数(4K) 共有メモリ クッション ページ数(4K)

SEP数

コンソール ログサイズ k

ローカルコンソールを表示 動的デバッグを許可 システム起動時に開始する

VII. Enterprise Server インスタンスを起動

- ① ESCWA 画面上部の [一般] ドロップダウンメニューをクリック

「Visual COBOL コマンドプロンプト(64-bit)」にて以下のコマンドを実行：

```
C:¥xa>casstart /rESDEMO64
..
CASCD0167I ES Daemon successfully auto-started 17:26:48
CASCD1005I C:¥Users¥Administrator¥Documents¥Micro Focus
User¥Visual COBOL¥WORKAREA¥ESDEMO64¥console.log 17:26:48
CASCD0050I ES "ESDEMO64" initiation is starting 17:26:48

C:¥xa>
```

正常に起動した場合の ESCWA 画面上から確認できるコンソールログ：

```
220516 17265078      6700 ESDEMO64 CASSI1600I SEP initialization completed successfully 17:26:50
220516 17265083      6700 ESDEMO64 CASSI1600I SEP initialization completed successfully 17:26:50
220516 17265083      6700 ESDEMO64 CASSI1600I SEP initialization completed successfully 17:26:50
220516 17265083      6700 ESDEMO64 CASX00015I AURORAPO XA interface initialized successfully 17:26:50
Registration Mode(Static) 17:26:50
220516 17265086      6700 ESDEMO64 CASX00015I AURORAPO XA interface initialized successfully 17:26:50
220516 17265086      6700 ESDEMO64 CASSI4104I PLTPI Phase 1 starting XA recovery program 17:26:50
220516 17265086      6700 ESDEMO64 CASSI4106I PLTPI Phase 1 starting File initialisation program 17:26:50
220516 17265086      6700 ESDEMO64 CASSI5040I Active SEP memory strategy set to x'00000001', retain count 100 17:
26:50
220516 17265180      184 ESDEMO64 CASSI1600I SEP initialization completed successfully 17:26:51
```

XA リソースが正常に初期化
された旨のメッセージ

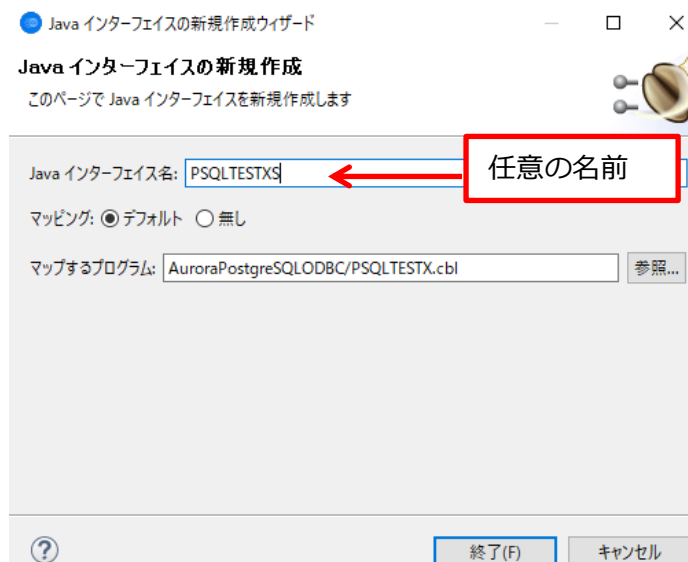
コンソールログの表示方法：

画面上部の [モニター] > [ログ] > [コンソールログ] をクリック



VIII. Java インターフェイスのプロファイル情報を追加

- ① COBOL エクスプローラーにて PSQLTESTX.cbl を右クリックし、
[新規作成] > [Java インターフェイス] を選択
- ② Java インターフェイス名を入力して、[終了] ボタンを押下



IX. COBOL パラメータのデータ型と Java 側のデータ型との変換マッピングを定義

- ① COBOL エクスプローラーにて
[<対象のプロジェクト>] > [Java インターフェイス] > [PSQLTESTXS] > [PSQLTESTX] ヘナビゲートし [PSQLTESTX] をダブルクリック
- ② アプリケーションの内容に合わせて方向を調整
対象の変数を右クリックから [プロパティ] を選択し変更
調整後の変換マッピング定義：

LINKAGE SECTION:		PSQLTESTX オペレーション - インターフェイス フィールド:			
名前	PICTURE	名前	方向	型	OCC...
OP-CODE	X	OP_CODE_io	入力	String	
MOD-VAL	9(5) comp-3	MOD_VAL_io	入力	int	
LNK-EMPNO	S9(9) comp-5	LNK_EMPNO_io	入力	int	
LNK-EMPDEPT		LNK_EMPDEPT_io	出力		
LNK-ENAME	X(10)	LNK_ENAME_io		String	
LNK-JOB	X(9)	LNK_JOB_io		String	
LNK-SAL	9(5)V9(2) comp-3	LNK_SAL_io		BigDecimal	
LNK-DNAME	X(14)	LNK_DNAME_io		String	

X. Java インターフェイスのプロファイル情報を構成

- ① COBOL エクスプローラーにて
[<対象のプロジェクト>] > [Java インターフェイス] > [PSQLTESTXS] を
右クリックし、[プロパティ] を選択
- ② [ディプロイメントサーバー] タブを選択
- ③ [変更] ボタンを押下し、7) で起動した Enterprise Server インスタンスを
指定
- ④ [トランザクション管理] 欄にて [コンテナ管理] を選択

一般 デプロイメントサーバー アプリケーションファイル **EJB 生成**

Enterprise Server 名:
ESDEMO64 (localhost:49894)

Enterprise Server 実行時環境の使用
Enterprise Server 実行時環境の構成...

EJB ステートフル サービスの場合、一部の値は無視されます

サービス名:
PSQLTESTX

トランザクション管理
 アプリケーション管理
 コンテナ管理

ディプロイする場合はユーザー名/パスワードが必要

- ⑤ [アプリケーションファイル] タブを選択
- ⑥ [レガシーアプリケーションをディプロイする] にチェック
- ⑦ [ファイルを追加] ボタンを押下し、対象のモジュール及びデバッグ情報ファイルを追加

一般 デプロイメントサーバー アプリケーションファイル EJB 生成

レガシーアプリケーションをディプロイ済みか、またはサーバーにディプロイする必要があるかを指定してください。

レガシーアプリケーションは既にディプロイ済み
ディプロイされたアプリケーションのパス: _____

レガシーアプリケーションをディプロイする

アプリケーションファイル:

New_Configuration.bin/PSQLTESTX.ant
New_Configuration.bin/PSQLTESTX.idy

ファイル追加
ファイル削除

- ⑧ [EJB 生成] タブを選択
- ⑨ [アプリケーションサーバー] 欄にて
「JEE 7」、「JBoss EAP 7.3」を指定
- ⑩ [J2EE クラスパス] 欄にて
%JBOSS_HOME%\modules\system\layers\base フォルダ配下の
 - javax¥ejb¥api¥main¥jboss-ejb-api_3.2_spec-2.0.0.Final-redhat-00001.jar
 - javax¥resource¥api¥main¥jboss-connector-api_1.7_spec-

2.0.0.Final-redhat-00001.jar

- javax¥servlet¥api¥main¥jboss-servlet-api_4.0_spec-2.0.0.Final-redhat-00001.jar

を指定

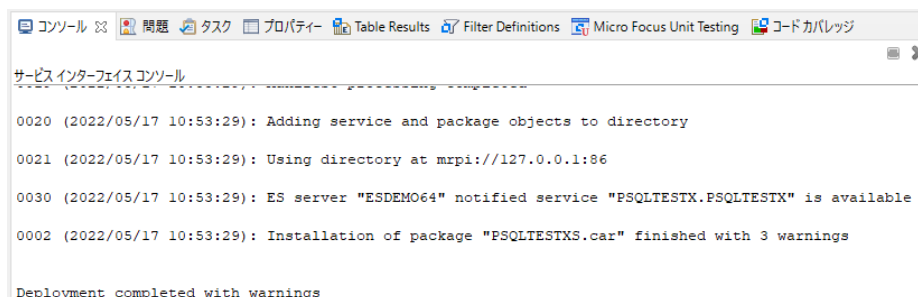
⑪ [OK] ボタンを押下

XI. Java インターフェイスをデプロイ

COBOL エクスプローラーにて

[<対象のプロジェクト>] > [Java インターフェイス] > [PSQLTESTXS]
を右クリックし [デプロイ] を選択

デプロイが完了した旨をコンソールビューより確認できます：



```
サービス インターフェイス コンソール
-----
0020 (2022/05/17 10:53:29): Adding service and package objects to directory
0021 (2022/05/17 10:53:29): Using directory at mrpi://127.0.0.1:86
0030 (2022/05/17 10:53:29): ES server "ESDEMO64" notified service "PSQLTESTX.PSQLTESTX" is available
0002 (2022/05/17 10:53:29): Installation of package "PSQLTESTXS.car" finished with 3 warnings

Deployment completed with warnings
```

同様に ESCWA 画面のサービス一覧からも確認できます：

☒ ▼ PSQLTESTX

🔗 .PSQLTESTX Available Web Serv... PSQLTESTX MFRHBINP created 10:...

XII. JBoss にリソースアダプタを配備

<製品のインストールフォルダ> ¥javaee¥javaee7¥jboss73EAP¥mfcobol-xa.rar を

%JBASS_HOME%¥standalone¥deployments

へコピー

下記の製品マニュアルを基に、JBoss の設定ファイルを編集

<https://www.microfocus.co.jp/manuals/VC70/Eclipse/vc70indx.html?#HHADTHDPOY71.html>

編集対象ファイル：

%JBOSS_HOME%\standalone\configuration\standalone.xml

編集内容 :

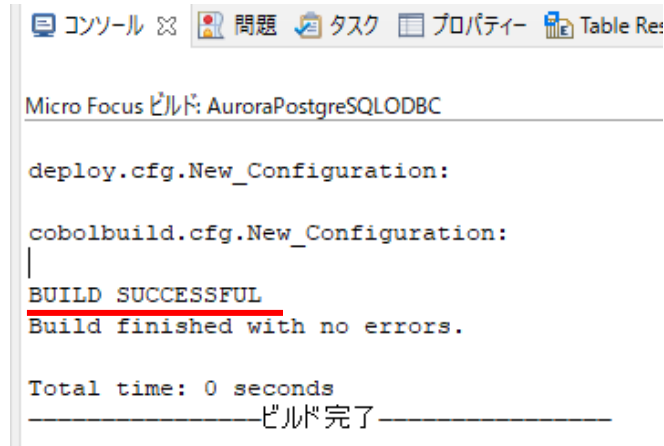
- jca サブシステムを無効にする
- mfcobol-xa.rar リソース アダプターを追加する

XIII. JBoss Application Server を起動

スタートメニューより、[JBoss プラットフォーム] > [サーバーの起動 (スタンドアロン)] をクリック

```
Calling "C:\JOSSEAP-7.3\bin\standalone.conf.bat"
    Setting      JAVA      property      to      "C:\Program
    Files\AdoptOpenJDK\jdk-8.0.212.03-hotspot\bin\java"
=====
=====
JBoss Bootstrap Environment
JBOSS_HOME: "C:\JOSSEAP-7.3"
JAVA:      "C:\Program      Files\AdoptOpenJDK\jdk-8.0.212.03-
hotspot\bin\java"
. .
11:01:40,726 INFO    [org.jboss.as] (Controller Boot Thread)
WFLYSRV0025: JBoss EAP 7.3.7.GA (WildFly Core 10.1.20.Final-
redhat-00001) は 28615 ミリ秒で開始しました - サービス 1011 個の
うち 792 個を開始しました (388 のサービスはレイジー、パッシブ、また
はオンデマンドです)。
11:01:40,726 INFO    [org.jboss.as] (Controller Boot Thread)
WFLYSRV0060: http://127.0.0.1:9990/management 上でリッスンす
る HTTP 管理インターフェース
11:01:40,726 INFO    [org.jboss.as] (Controller Boot Thread)
WFLYSRV0051: 管理コンソールは http://127.0.0.1:9990 をリッスン
しています。
```

- XIV. JBoss にデプロイするスタブクライアントアプリケーションを生成
COBOL エクスプローラーにて
[<対象のプロジェクト>] > [Java インターフェイス] > [PSQLTESTXS]
を右クリックし [クライアント生成...] を選択
正常にビルドされるとコンソールビューにその旨のメッセージが出力されま
す :



```
Micro Focus ビルド: AuroraPostgreSQLODBC  
  
deploy.cfg.New_Configuration:  
  
cobolbuild.cfg.New_Configuration:  
|  
BUILD SUCCESSFUL  
Build finished with no errors.  
  
Total time: 0 seconds  
-----ビルド完了-----
```

XV. スタブクライアントアプリケーションを JBoss にデプロイ

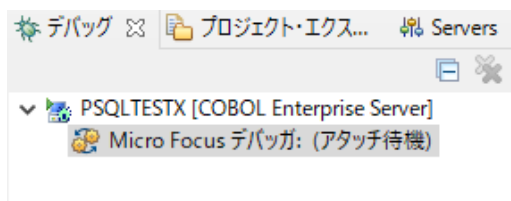
- ① <プロジェクトフォルダ> %repos%PSQLTESTXS.deploy
配下に生成された PSQLTESTXS.ear を JBoss のデプロイフォルダ
%JBASS_HOME%\standalone\deployments
にコピー
- ② XIII. で使用したプロンプト画面にて正しくデプロイされたことを確認



```
サーバの起動 (スタンドアロン)  
java:app/PSQLTESTXS.jar/PSQLTESTXSEJB!com.mypackage.PSQLTESTXS.PSQLTESTXSLocal  
java:module/PSQLTESTXSEJB!com.mypackage.PSQLTESTXS.PSQLTESTXSLocal  
java:global/PSQLTESTXS/PSQLTESTXS.jar/PSQLTESTXSEJB  
java:app/PSQLTESTXS.jar/PSQLTESTXSEJB  
java:module/PSQLTESTXSEJB  
11:22:20,271 INFO [io.jaegertracing.internal.JaegerTracer] (MSC service thread 1-3) No shutdown hook  
registered: Please call close() manually on application shutdown.  
11:22:20,333 INFO [io.jaegertracing.internal.JaegerTracer] (MSC service thread 1-2) No shutdown hook  
registered: Please call close() manually on application shutdown.  
11:22:20,380 INFO [org.jboss.weld.Version] (MSC service thread 1-4) WELD-000900: 3.1.6 (redhat)  
11:22:20,981 INFO [io.smallrye.metrics] (MSC service thread 1-3) MicroProfile: Metrics activated (Sm  
allRye Metrics version: 2.4.0.redhat-00004)  
11:22:21,803 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 44) WFLYUT0021: 登  
録された web コンテキスト: '/PSQLTESTXS' (サーバー 'default-server' 用)  
11:22:22,065 INFO [org.jboss.as.server] (DeploymentScanner-threads - 2) WFLYSRV0010: "PSQLTESTXS.ear  
(runtime-name: "PSQLTESTXS.ear") をデプロイしました。
```

XVI. COBOL Enterprise Server デバッグを起動

- ① COBOL エクスプローラーにてプロジェクトを右クリックし
[デバッグ] > [デバッグの構成] を選択
- ② [COBOL Enterprise Server] をダブルクリック
- ③ 任意の名前を入力した後、[Enterprise Server] 欄にて [参照] ボタンを押下し、利用中の Enterprise Server インスタンスを選択
- ④ デバッグの種類にて [Java] タブを選択
- ⑤ [デバッグ] ボタンを押下
- ⑥ [パースペクティブの切り替えの確認] には [はい] を選択
デバッグパースペクティブにてデバッガが待機状態となります：



XVII. 本検証で利用するデータの確認

Query Editor Query History

```
1 select empno,ename,job,sal,dname from emp inner join dept on emp.deptno=dept.deptno
2 order by empno
3
```

Data Output Explain Messages Notifications

	empno integer	ename character varying (10)	job character varying (12)	sal numeric (7,2)	dname character varying (14)
1	7369	SMITH	CLEARK	800.00	RESEARCH
2	7499	ALLEN	SALESPERSON	1600.00	SALES
3	7521	WARD	SALESPERSON	1250.00	SALES
4	7566	JONES	MANAGER	3000.00	RESEARCH
5	7654	MARTIN	SALESPERSON	1250.00	SALES
6	7698	BLAKE	MANAGER	2850.00	SALES
7	7782	CLARK	MANAGER	2450.00	ACCOUNTING
8	7788	SCOTT	ANALYST	3000.00	RESEARCH
9	7839	KING	PRESIDENT	5000.00	ACCOUNTING
10	7844	TURNER	SALESPERSON	1500.00	SALES
11	7876	ADAMS	CLEARK	1100.00	RESEARCH
12	7900	JAMES	CLEARK	950.00	SALES
13	7902	FORD	ANALYST	3000.00	RESEARCH
14	7934	MILLER	CLEARK	1300.00	ACCOUNTING

XVIII. スタブクライアントアプリケーションを起動

- ① ブラウザを起動し、スタブクライアントアプリケーション URL を入力
本検証で利用するアプリケーションの場合は下記を指定します。

<http://localhost:8080/PSQLTESTXS/PSQLTESTX.jsp>

- ② テストデータを入力

Test client for PSQLTESTXS.PSQLTESTX

[Back](#)

Perform the test by entering values:

PSQLTESTX_OP_CODE_io :

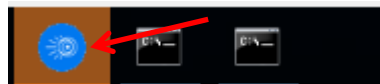
PSQLTESTX_MOD_VAL_io :

PSQLTESTX_LNK_EMPNO_io :

本例では、EMPNO = 7566 のレコードにおける SAL 列に 10 を加算します。

- ③ [Go!] ボタンを押下

Eclipse のデバッガに処理が引き込まれます :



XIX. Eclipse 上でデバッグ実行

これまで見てきたように変数に格納された値のウォッチ、ステップ単位の実行、ブレークポイント等といった機能をここでも駆使して効率的にデバッグできます。

XX. 実行結果の確認

ブラウザ上で Java EE アプリケーション側に戻ってきた結果を確認します：

PSQLTESTX_OP_CODE_io :

PSQLTESTX_MOD_VAL_io :

PSQLTESTX_LNK_EMPNO_io :

Result:

Variable	Value
LNK_EMPDEPT_io.LNK_ENAME_io	JONES
LNK_EMPDEPT_io.LNK_JOB_io	MANAGER
LNK_EMPDEPT_io.LNK_SAL_io	3010
LNK_EMPDEPT_io.LNK_DNAME_io	RESEARCH

3000 に 10 を加えた 3010 が返ってきています。



XXI. データベース内のレコードが更新されていることを確認

Query Editor Query History

```

1 select empno,ename,job,sal,dname from emp inner join dept
2 on emp.deptno=dept.deptno
3 where empno=7566
    
```

Data Output Explain Messages Notifications

	empno integer	ename character varying (10)	job character varying (12)	sal numeric (7,2)	dname character varying (14)
1	7566	JONES	MANAGER	3010.00	RESEARCH

XXII. 実行時エラーを引き起こすロジックが PSQLTESTX.cbl 中の UPDATE 文の後に埋め込まれたサンプルプログラム PSQLTESTXE.cbl を上記の要領でプロジェクトへ追加、Enterprise Server インスタンスへデプロイ、対応するスタブクライアントを JBoss にデプロイ

PSQLTESTXE.cbl の抜粋：

```

                                :
01 DUMMY-ARR                    OCCURS 5 TIMES PIC X(10).
    
```

```
01 DUMMY-IDX          PIC 9(1) VALUE 6.
                        :
EXEC SQL
  UPDATE EMP
    SET SAL = :HV-SAL
    WHERE EMPNO = :HV-EMPNO
END-EXEC.

MOVE ALL'A' TO DUMMY-ARR(DUMMY-IDX).
                        :
```

XXIII. COBOL Enterprise Server デバッグを起動

XXIV. 実行時エラーになるロジックが埋め込まれた COBOL アプリケーション
をスタブクライアントアプリケーションより実行
ここでは上と同じパラメータを指定します：

Test client for PSQLTESTXES.PSQLTESTXE

[Back](#)

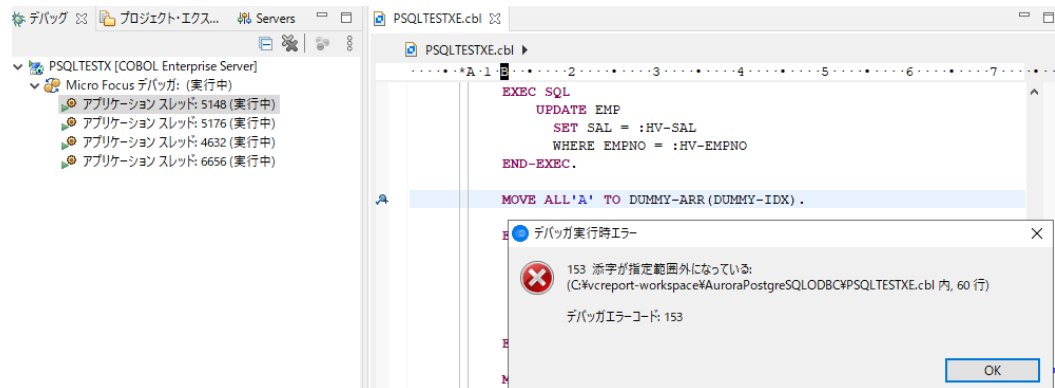
Perform the test by entering values:

PSQLTESTXE_OP_CODE_io :

PSQLTESTXE_MOD_VAL_io :

PSQLTESTXE_LNK_EMPNO_io :

XXV. Eclipse のデバッガで UPDATE 文を実行した次の MOVE 文で
COBOL の実行時エラーが発生することを確認



XXVI. Transaction Manager(Enterprise Server インスタンス) にて
UPDATE 文の変更が Rollback され、レコードの sal 値が実行前と
変わっていないことを確認

Query Editor Query History

```
1 select empno,ename,job,sal,dname from emp inner join dept
2 on emp.deptno=dept.deptno
3 where empno=7566
```

Data Output Explain Messages Notifications

	empno integer	ename character varying (10)	job character varying (12)	sal numeric (7,2)	dname character varying (14)
1	7566	JONES	MANAGER	3010.00	RESEARCH

以上