Micro Focus Visual COBOL 2.2J for Windows

PostgreSQL データアクセス

動作検証 検証結果報告書

2014 年 5 月 2 日

マイクロフォーカス株式会社

1. 検証概要、目的及びテスト方法

1.1 検証概要

PostgreSQL はカルフォニア大学バークレー校で開発された POSTGRES, Version 4.2 をベースにしたオープンソースのリレーショナルデータベース管理システムです。PostgreSQL はオープンソースでありながら、商用リレーショナルデータベース管理システム製品に劣らない機能を装備しています。そのため、学習用途に限らず企業システムにおいても利用されるケースも往々にして確認されています。更に近年の企業システムへのオープンソースソフトウェア活用の流れに相まって一層注目を集めるリレーショナルデータベース管理システムです。

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。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 PostgreSQL へアクセスできることを動作検証しました。加えて、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で COBOL アプリケーションがコンテナ管理サービスとして PostgreSQL と連携できることも動作検証しました。

1.2 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ます。製品出荷時に弊社にて動作検証される RDBMS は Oracle、SQL Server、DB2 のみですが、OpenESQL を使えば 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対して設計上問題なくアクセスすることができます。今回、本稿執筆時点における最新版 PostgreSQL 9.3.4 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 Native 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 Enterprise Server に配備し、コンテナ管理のサービスとして ODBC 経由で PostgreSQL を XA 準拠のリソースマネージャとして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、javabyte 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

ソフトウェア

Windows 8 Enterprise

PostgreSQL Server 9.3.4

利用したデータベースの文字コード	
ENCODING	UTF8
LC_LOCATE	Japanese_Japan.932
LC_CTYPE	Japanese_Japan.932

PostgreSQL ODBC ドライバ 64bit 版 9.3-0210

Simple-JNDI 11.4.1

PostgreSQL JDBC ドライバ pgJDBC 9.3-1100

JBoss Application Server 6 Version: 6.1.0

Micro Focus Visual COBOL 2.2J for Windows

ハードウェア

機種： Dell OPTIPLEX7010

CPU： Intel Core2 i7-3770 3.40GHz

Memory： 3.00 Gbyte memory(ゲスト OS に割り当てたサイズ)

3. テスト内容

3.1 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作成します。続いて、INSERT 文によるデータの充填、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、中身を確認します。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。

3.2 JDBC 接続

3.1 で利用するプログラムソースの接続部分のみを JDBC 用に調整し、同じロジックが正常に処理できることを検証します。

3.3 XA インタフェースを介した ODBC 接続

JavaEE 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 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 Enterprise Server に配備し、JavaEE 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ズさせます。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) に委譲します。アプリケーションの処理完了後は、正しく Transaction が完結していることを別途、PostgreSQL に付属する管理ツールを使って確認します。更に、DML 文実行後、敢えて実行時エラーが発生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

4.1 インストール

> PostgreSQL Server
> The PostgreSQL Global Development Group のサイトにおける以下のページ中の案内に従い、EnterpriseDB 社のサイトよりインストーラをダウンロードし、インストールしました。
>
> The PostgreSQL Global Development Group のサイト(2014/4/15 リンク検証) :
> http://www.postgresql.org/download/windows/
>
> ダウンロード元(2014/4/15 リンク検証) :
> http://www.enterprisedb.com/products-services-training/pgdownload#windows

> 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(2014/4/15 リンク検証) :
> http://www.postgresql.org/ftp/odbc/versions/msi/

> PostgreSQL JDBC ドライバ
PostgreSQL Server インストール後に自動起動された Stack Builder 3.1.1 を使ってインストールしました。

> 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(2014/5/1 リンク検証)：
http://code.google.com/p/osjava/downloads/detail?name=simple-jndi-0.11.4.1.zip&can=2&q=

> JBoss Application Server 6.1.0
以下のリンク先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(2014/5/2 リンク検証)：
http://jbossas.jboss.org/downloads

4.2 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は、Micro Focus のウェブサイト上に本報告書と供に公開しています。

➢ 3.1 及び 3.2 の検証に使用したプログラムの処理フロー

① 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 PostgreSQL データベースとの接続を切断

- 3.3 の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 COBOL サービスを呼び出し(JavaEE)
② SELECT 文を実行し変更対象のデータを取得(COBOL)
③ 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)
④ UPDATE 文を実行(COBOL)
⑤ SELECT 文を実行(COBOL)
⑥ SELECT 文で取得したデータを戻りパラメータにセット(COBOL)
⑦ COBOL より返された値をブラウザに表示(JavaEE)

[パターン２：エラー処理]
① COBOL サービスを呼び出し(JavaEE)
② SELECT 文を実行し変更対象のデータを取得(COBOL)
③ 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)
④ UPDATE 文を実行(COBOL)
⑤ 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。

4.3 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 インターフェースを介した ODBC 接続においても Enterprise Server 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

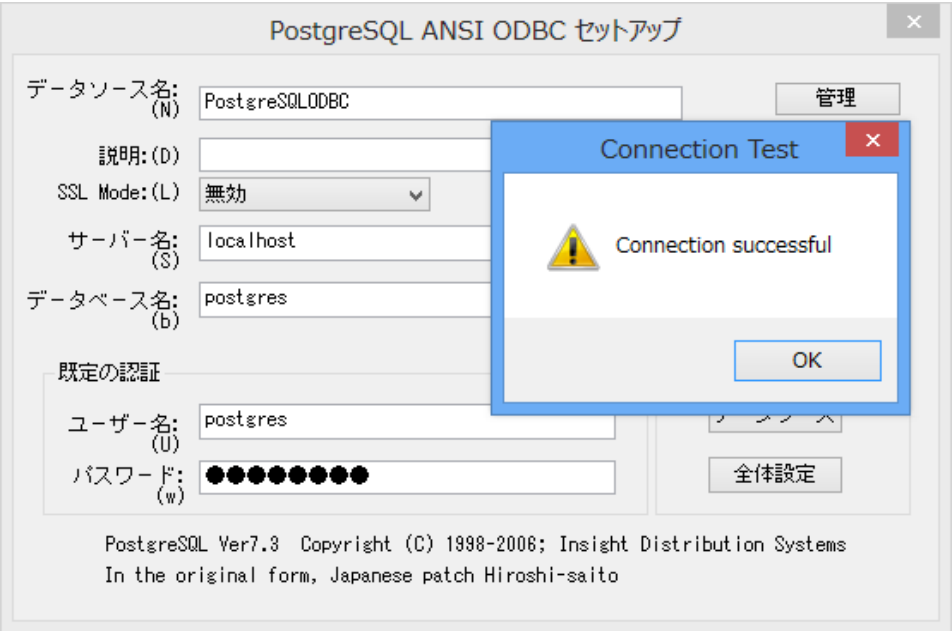
埋め込み SQL 文を含む COBOL プログラムを Visual COBOL を使ってNative コードにコンパイルしたアプリケーションより PostgreSQL データベースに ODBC 経由で接続して DDL 文、DML 文、DCL 文を発行してデータベースを操作できることを検証しました。javabyte コードにコンパイルし、JVM クラスとした場合も 同様に JDBC 経由にて正しく操作できることを確認しました。日本語を含んだデータについても両者ともに正常に扱えることも検証できました。

XA インターフェースを介した 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 JavaEE アプリケーションの一部として正しく動作し、Enterprise Server 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、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

付録 1. サンプルアプリケーションの実行 – ODBC 編

1) 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 ASNI(x64)」を選択し [完了] ボタンを押下
   ④ サーバ名、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

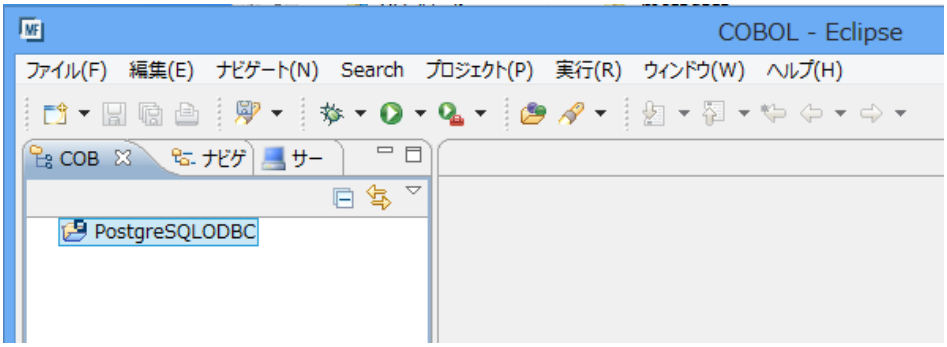


   ⑥ [保存] ボタンを押下し、構成内容を反映

2) スタートメニューより Eclipse を起動
3) 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]

プロジェクト作成後の画面：



4) 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 エクスプローラにてプロジェクトを右クリックし、

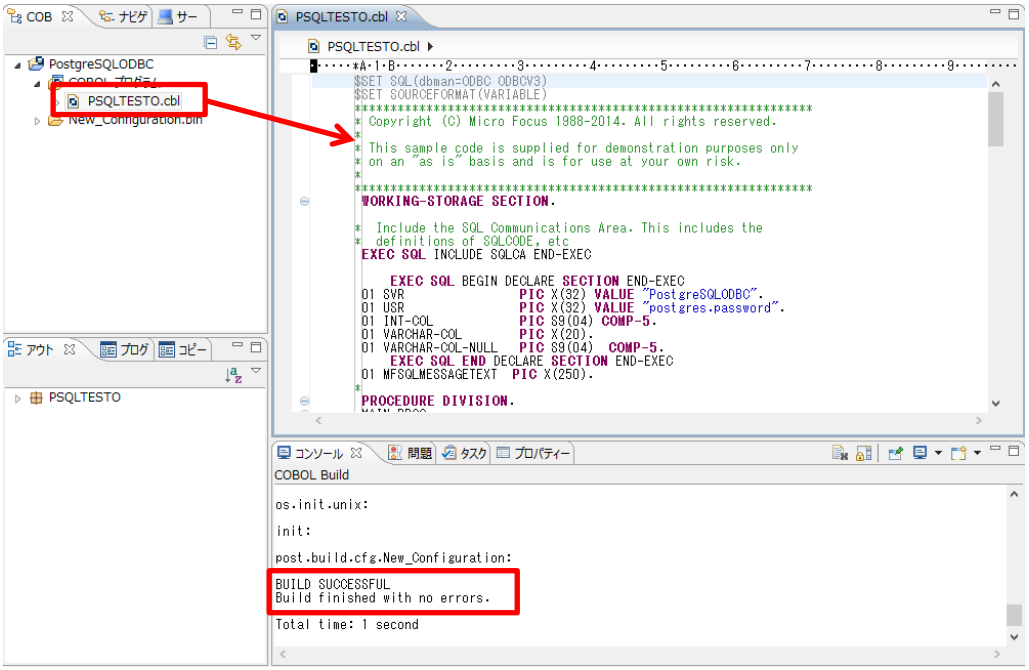
② [インポート] > [インポート] へとナビゲート

③ [一般] > [ファイルシステム] を選択し [次へ] ボタンを押下

④ [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エクスプローラをナビゲート

⑤ 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。



5) 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 エクスプローラにてプロジェクトを右クリックし、[プロパティー] を選択

② [Micro Focus] > [ビルド構成] > [COBOL] へとナビゲート

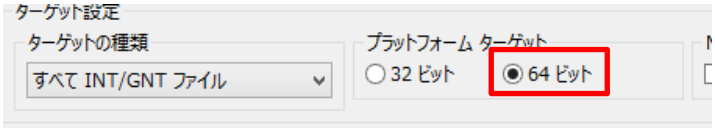
③ ターゲットの種類を「すべて INT/GNT ファイル」に設定

④ 「構成の固有な設定を可能にする」にチェック

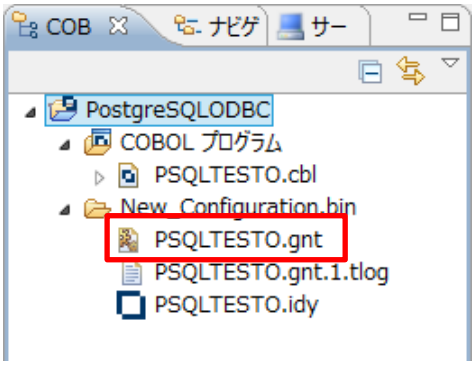
⑤ 「.GNT にコンパイル」にチェック

6) プラットフォームターゲットを 64bit に指定

5) の画面における「プラットフォームターゲット」欄にて「64 ビット」にチェック



7) 生成されたモジュールを確認

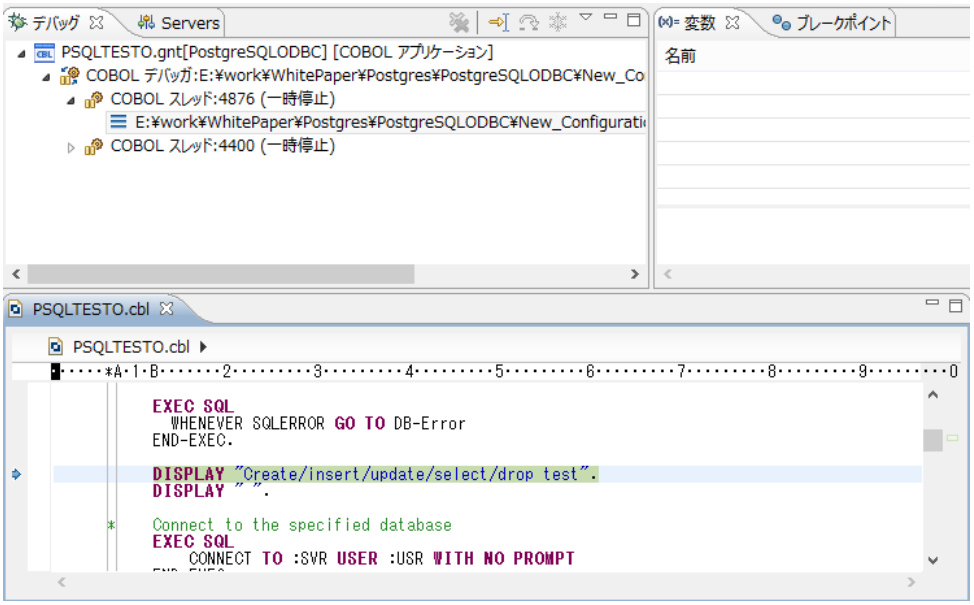


8) 生成された動的ロードモジュール PSQLTESTO.gnt をデバッグ実行

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

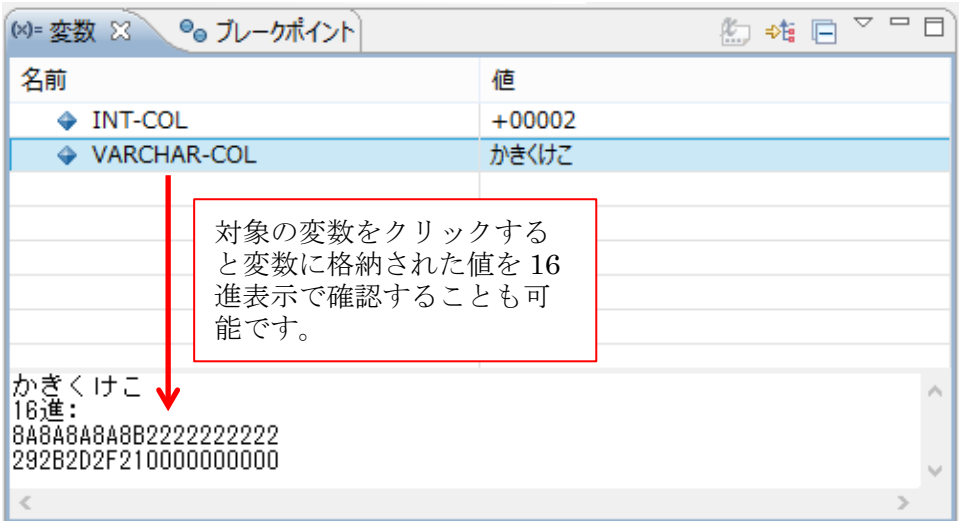
③ デバッグパースペクティブに切り替わり最初の COBOL 文の実行前で処理が止まっています。



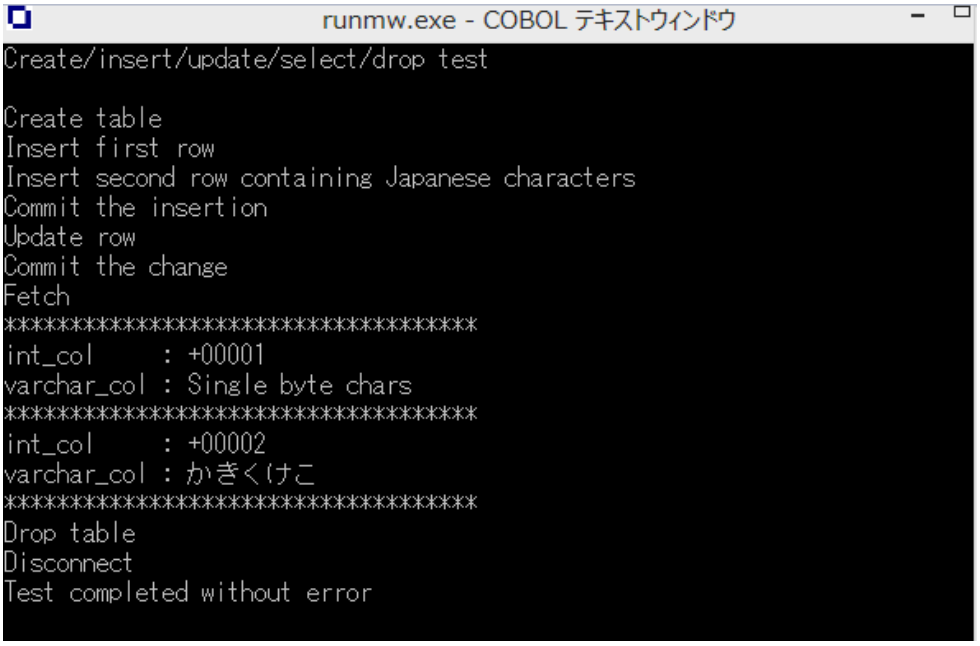
④ COBOL 用に作りこまれたデバッガの機能を駆使してデバッグ実行

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(CALL 文や PERFORM 文の先までステップを進めず、それらを１ステップをして実行)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：



⑥　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



付録 2. サンプルアプリケーションの実行 – JDBC 編

1) jndi.properties ファイルを作成

```
E:¥work¥jdbc_library¥JNDI>type jndi.properties
java.naming.factory.initial=org.osjava.sj.SimpleContextFactory
org.osjava.sj.root=E:¥¥work¥¥jdbc_library¥¥JNDI
E:¥work¥jdbc_library¥JNDI>
```

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

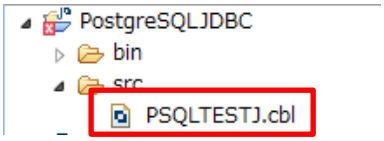
2) PostgreSQL を利用する Data Source 用の properties ファイルを作成

```
E:¥work¥jdbc_library¥JNDI>type pg.properties
type=javax.sql.DataSource
driver=org.postgresql.Driver
url=jdbc:postgresql:postgres
user=postgres
password=password

E:¥work¥jdbc_library¥JNDI>
```

3) COBOL JVM プロジェクトを作成

4) 付録 1 で利用したプログラムをプロジェクトにインポート
   ① COBOL エクスプローラにて src フォルダを右クリックし、
   ② [インポート] > [インポート] へとナビゲート
   ③ [一般] > [ファイルシステム] を選択し [次へ] ボタンを押下
   ④ [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エクスプローラをナビゲート
   ⑤ 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

5) COBOL エクスプローラにてファイルを右クリックし、[名前の変更] を選択して名称を変更



6) コンパイラオプションを JDBC 用に変更

編集前：

```
$SET SQL(dbman=ODBC ODBCV3)
$SET SOURCEFORMAT(VARIABLE)
                    :
```

編集後：

```
$SET SQL(dbman=JDBC)
$SET SOURCEFORMAT(VARIABLE)
                    :
```

7) CONNECT 文を JDBC 用に変更

編集前：

```
                    :
01 SVR                  PIC X(32) VALUE "PostgreSQLODBC".
01 USR                  PIC X(32) VALUE "postgres.password".
                    :
    EXEC SQL
        CONNECT TO :SVR USER :USR WITH NO PROMPT
    END-EXEC.
                    :
```

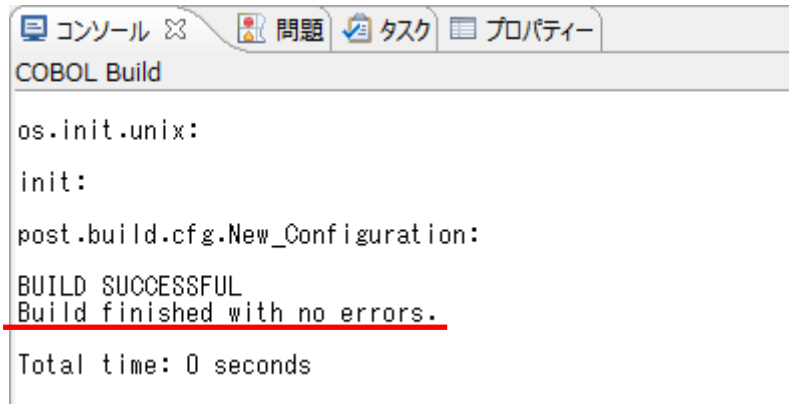
編集後：

```
                    :
       01 WK-DSN              PIC X(2) VALUE "pg".
                    :
           EXEC SQL
               CONNECT TO :WK-DSN WITH NO PROMPT
           END-EXEC.
                    :
```

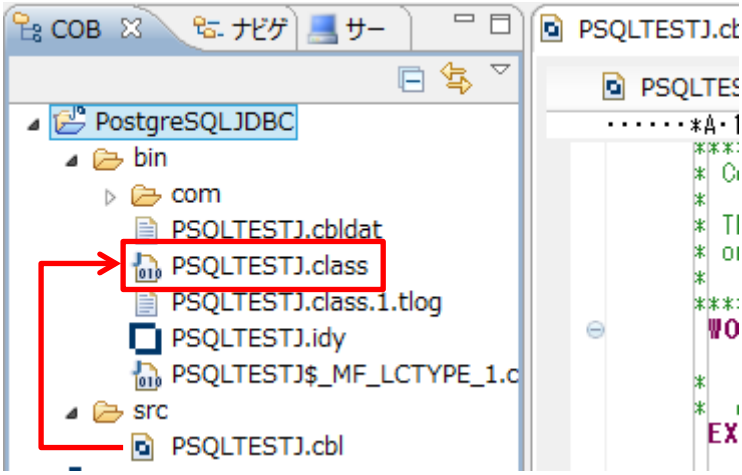
8) CLASSPATH に JDBC Driver 及び Simple JNDI のライブラリを追加

① COBOL エクスプローラにてプロジェクトを右クリックから［プロパティー］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、1) 及び 2) 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

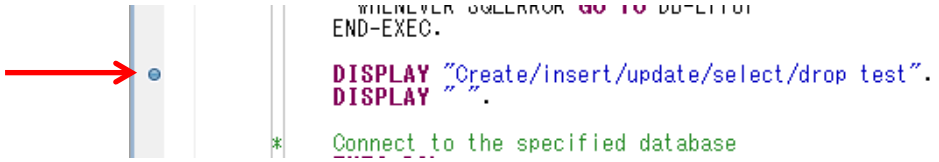


9) COBOL エクスプローラにて COBOL プログラムに対するクラスファイルが生成されていることを確認



10) 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

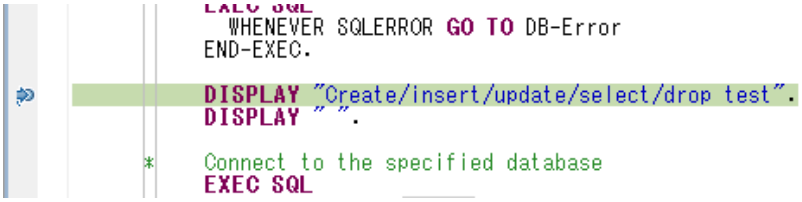
① 最初の COBOL 文にブレークポイントを指定



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

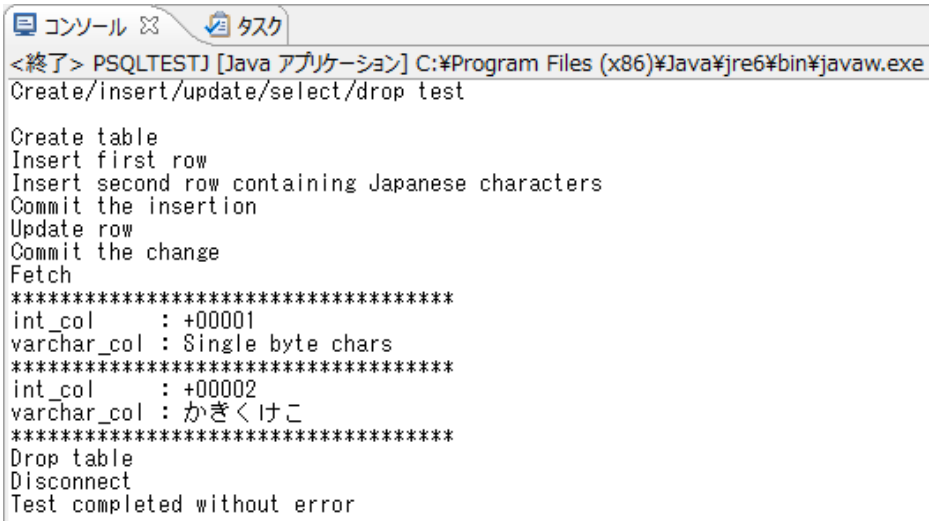
ブレークポイントを指定した文の実行前で停止しています：



④ 付録１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：

⑤ 正常に処理されたことをコンソールビューで確認



付録 3. サンプルアプリケーションの実行 – XA インターフェース編

1) 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「Enterprise Developer 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¥xa
配下にある以下のファイルを任意のフォルダへコピー

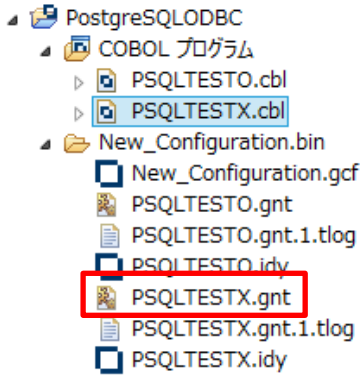
- build.bat
- esodbcxa.cbl
- xapd.cpy
- xaws.cpy

③ XA スイッチモジュールをビルド

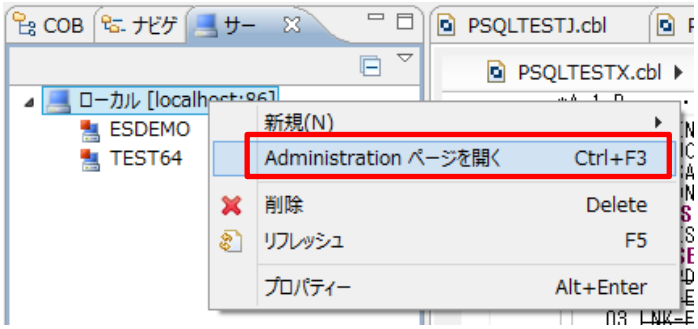
```
E:¥work¥xa>build odbc
Building 64-bit switch module...
Micro Focus COBOL - CBLLINK utility
Version 2.2.0.93 Copyright (C) Micro Focus 1984-2013. All rights reserve
d.

Micro Focus COBOL
Version 2.2.00151 Copyright (C) Micro Focus 1984-2013. All rights reserv
ed.
* 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:       14904     Literals:        1440
Microsoft (R) Incremental Linker Version 11.00.50727.1
Copyright (C) Microsoft Corporation.  All rights reserved.

ESODBCXA.obj
cbllds00000F5C.obj
   Creating library ESODBCXA.lib and object ESODBCXA.exp
Microsoft (R) Manifest Tool version 6.2.9200.16384
Copyright (c) Microsoft Corporation 2012.
All rights reserved.

Unable to locate the COBOL bin64 directory.

If you intend to execute JES-initiated transactions under Enterprise Ser
ver you must copy ESODBCXA.DLL to a directory on your PATH, such as your
 COBOL bin64 directory.

If you do not do so, then such transactions will not be able to communic
ate with the database server.

E:¥work¥xa>
```

2) 付録 1 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

3) 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



4) サーバエクスプローラにて対象の Directory Server を右クリックし [Administration ページを開く] を選択し Enterprise Server Administration ページを開く



5) Enterprise Server インスタンスに XA リソースを構成

① 利用する 64bit 版の Enterprise Server の行にある [編集] ボタンを押下

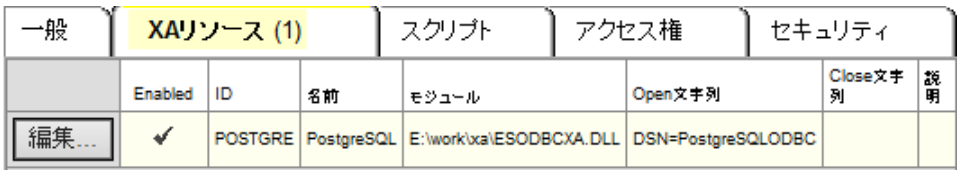
② [XA リソース] タブをクリック

③ [追加] ボタンを押下

④ 構成情報を入力

⑤ [OK] ボタンを押下

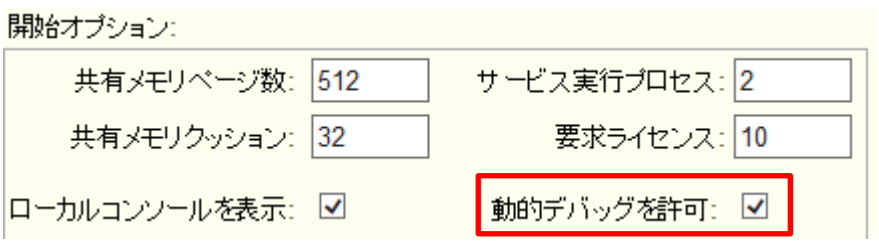
⑥ 正しく追加されていることを確認



6) 動的デバッグを受け付けるよう Enterprise Server を構成

① Enterprise Server Administration ページのトップより対象の Enterprise Server の行にある [編集] ボタンを押下

② [動的デバッグを許可] にチェック

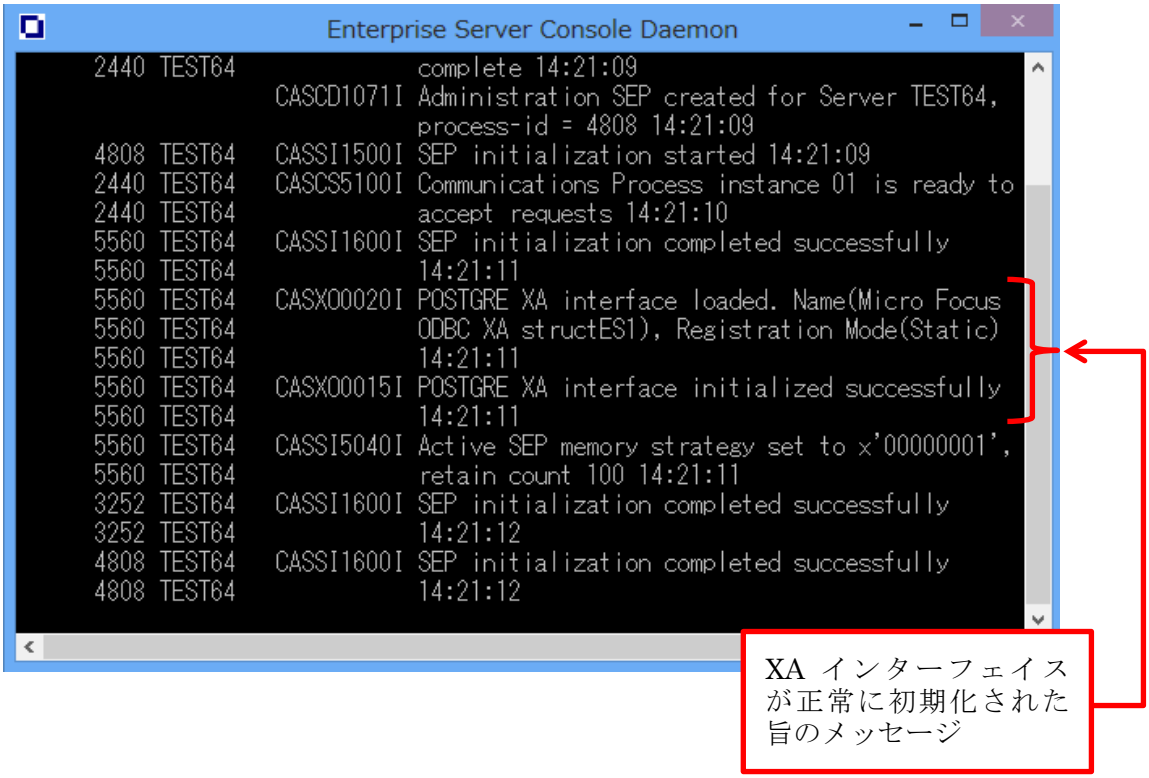


7) Enterprise Server を起動

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

```
E:¥work>casstart /rTEST64
CASCD0167I ES Daemon successfully auto-started 14:02:55
CASCD0050I ES "TEST64" initiation is starting 14:02:55

E:¥work >
```

正常に起動した場合の Enterprise Server Console Damon の画面：

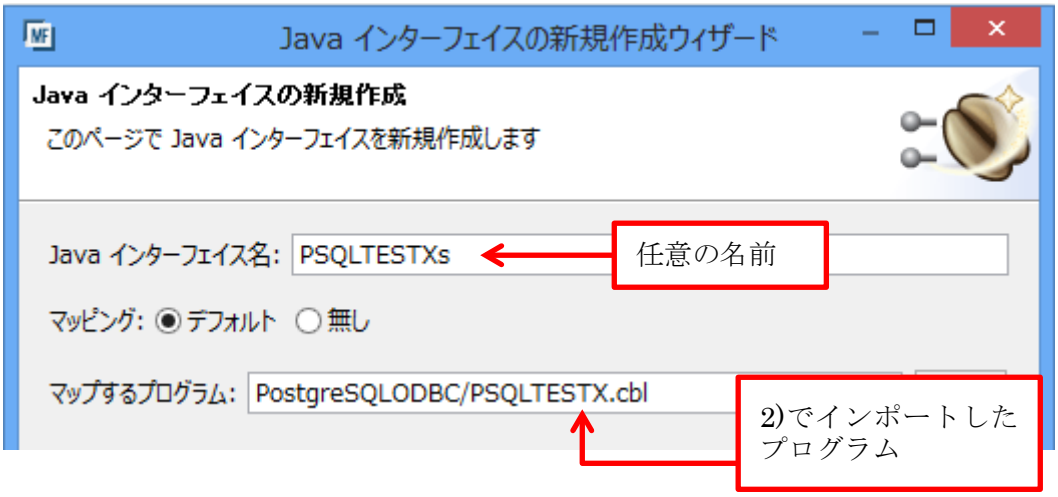


8) Java インターフェイスのプロファイル情報を追加

① COBOL エクスプローラにてプロジェクトを右クリックし、
[新規] > [その他]
を選択

② [Micro Focus IMTK] > [Java インターフェイス] を選択し、[次へ] ボタンを押下

③ Java インターフェイス名及びマップするプログラムを指定



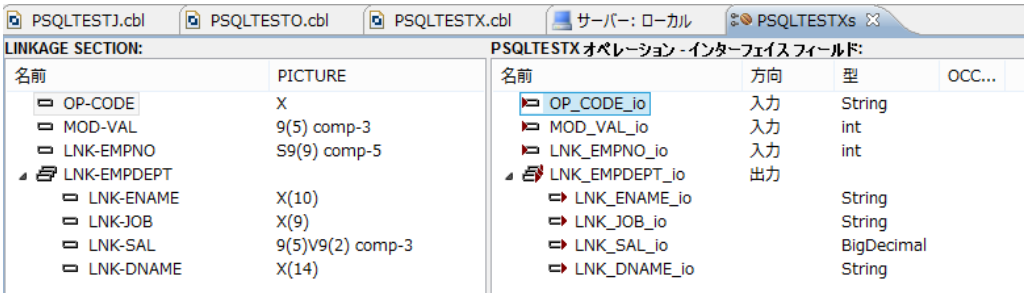
④ [完了] ボタンを押下

9) 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 エクスプローラにて
[＜対象のプロジェクト＞] > [Java インターフェイス] > [PSQLTESTXs] > [PSQLTESTX]
へとナビゲートし [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

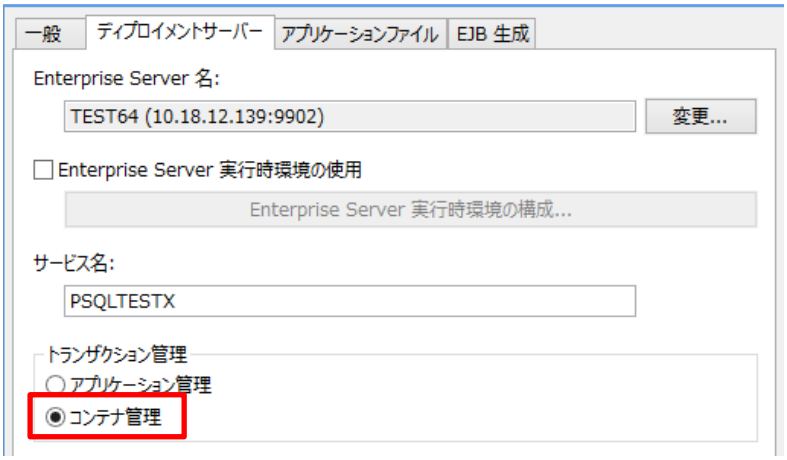
調整後の変換マッピング定義：



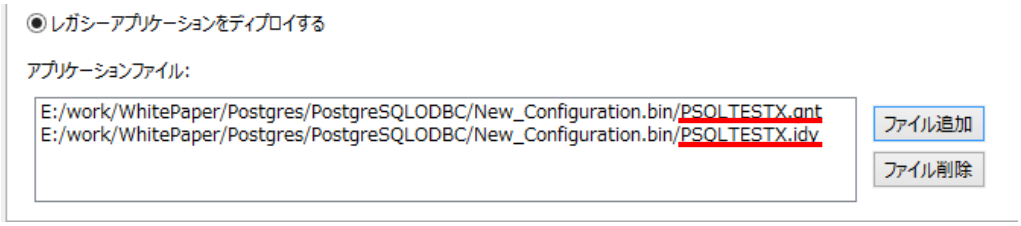
10) Java インターフェイスのプロファイル情報を構成

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 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



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



⑧ [EJB 生成] タブを選択
⑨ [アプリケーションサーバ] 欄にて
「JEE 6」、「JBoss 6.1.0」
を指定
⑩ [J2EE クラスパス] 欄にて
%JBOSS_HOME%¥client¥jbossall-client.jar
%JBOSS_HOME%¥client¥jboss-ejb-api_3.1_spec.jar
を指定
⑪ [OK] ボタンを押下

11) Java インターフェイスをディプロイ

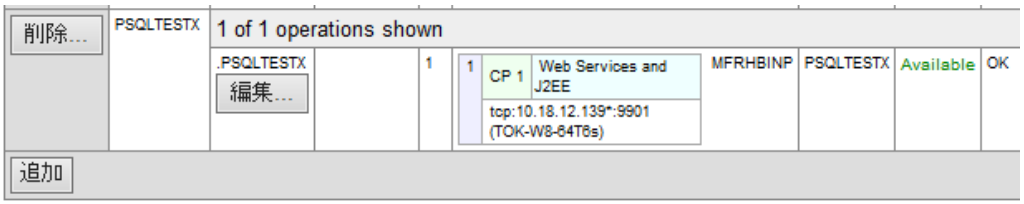
COBOL エクスプローラにて

[＜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 [ディプロイ] を選択

ディ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：

同様に Enterprise Server Administration からも確認できます：



12) 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¥javaee6¥jboss6

より

> mfcobol-xa.rar
> mfcobol-xa-ds.xml

を

%JBOSS_HOME%¥server¥default¥deploy

へコピー

13) JBoss Application Server を起動

```
E:¥jboss-6.1.0.Final¥bin>run -b localhost -c default
Calling E:¥jboss-6.1.0.Final¥bin¥run.conf.bat
=========================================================================
===

  JBoss Bootstrap Environment

  JBOSS_HOME: E:¥jboss-6.1.0.Final

                                        :

13:21:03,669 INFO  [org.jboss.bootstrap.impl.base.server.AbstractServer] JBo
ssAS
 [6.1.0.Final "Neo"] Started in 41s:819ms
```

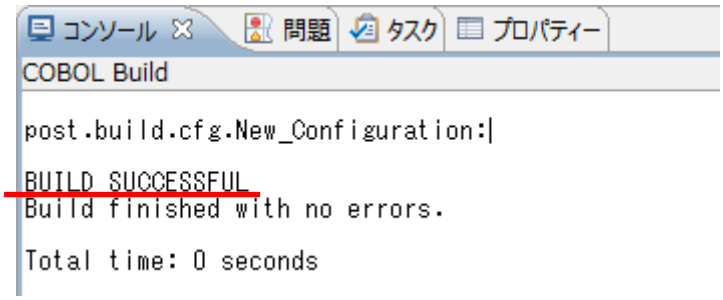
14) JBoss にディプロイするスタブクライアントアプリケーションを生成

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

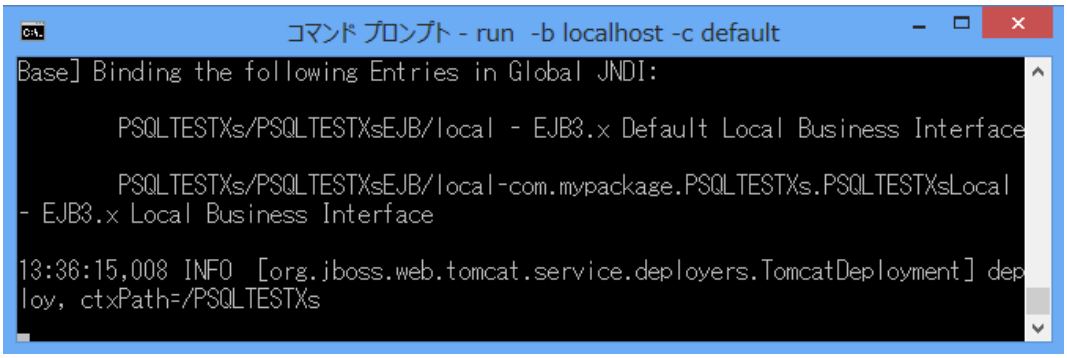
正常にビルドされますとコンソールビューにその旨のメッセージが出力されます：



15) スタブクライアントアプリケーションを JBoss にディプロイ

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 のディプロイフォルダ
%JBOSS_HOME%¥server¥default¥deploy
にコピー

② 13) で使用したプロンプト画面にて正しくディプロイされたことを確認



16) COBOL Enterprise Server デバッグを起動

① COBOL エクスプローラにてプロジェクトを右クリックから
[デバッグ] > [デバッグの構成] を選択

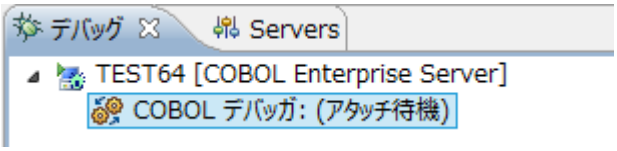
② [COBOL Enterprise Server] をダブルクリック

③ [Enterprise Server] 欄にて [参照] ボタンを押下し、利用中の Enterprise Server を選択

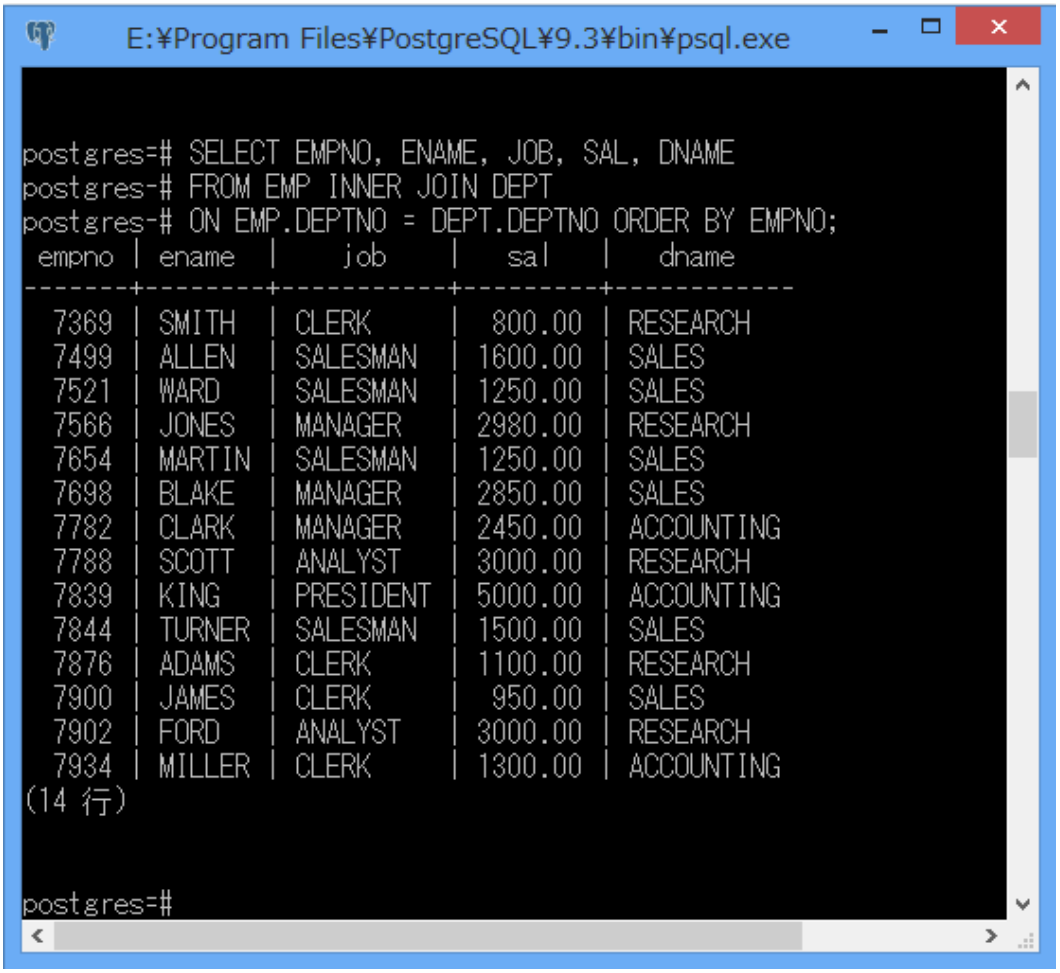
④ [デバッグ] ボタンを押下

⑤ [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：



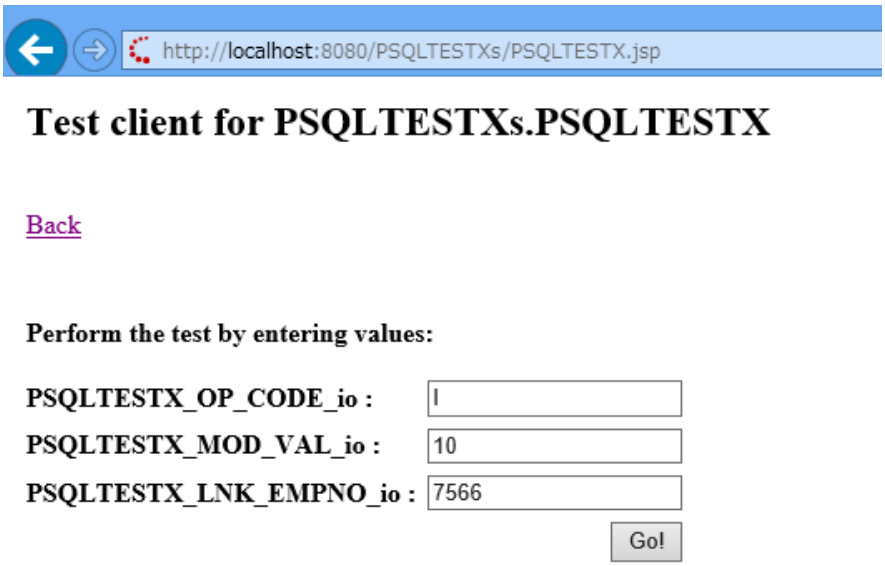
17) PostgreSQL のユーティリティ psql にて本検証で利用するデータを確認



18) スラ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の URL を入力
本検証で利用するアプリケーションの場合は
http://localhost:8080/PSQLTESTXs/PSQLTESTX.jsp

② テストデータを入力

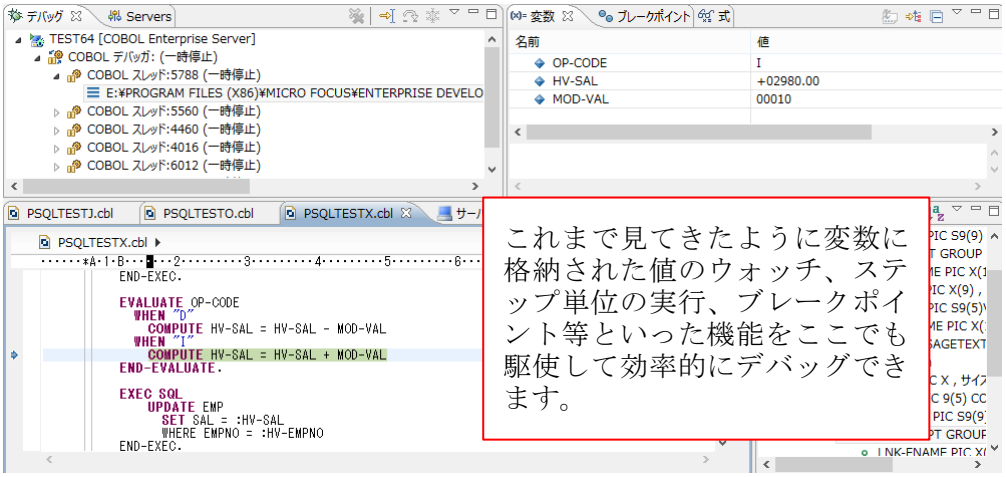


➔ 本例では、EMPNO = 7566 のレコードにおける SAL 列に 10 を足しこませます。

③ [Go!] ボタンを押下

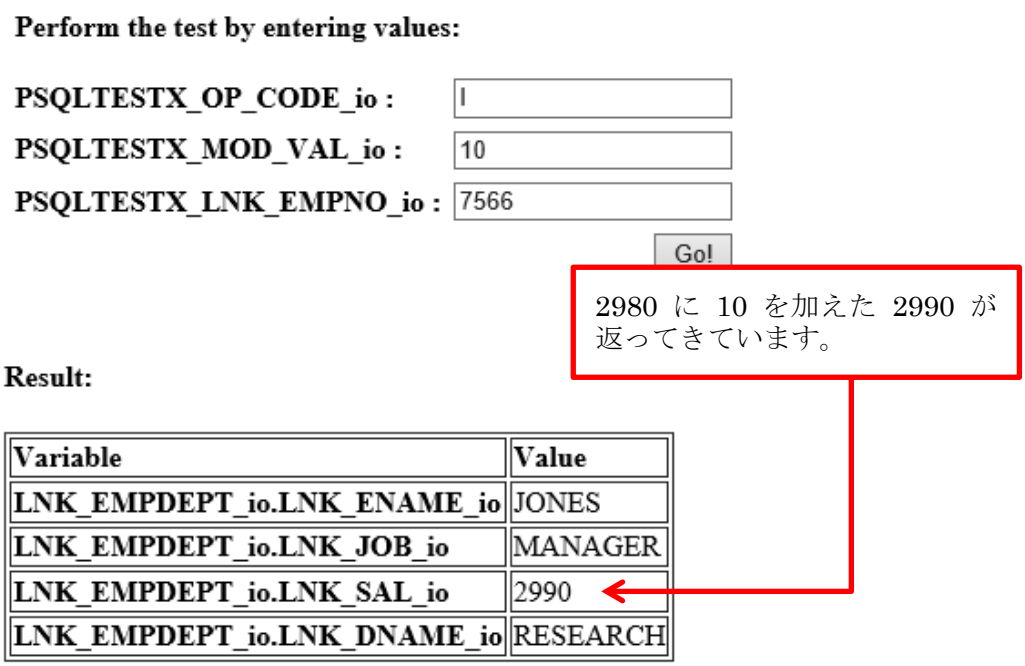
Eclipse のデバッガに処理が引き込まれます：

19) Eclipse 上でデバッグ実行

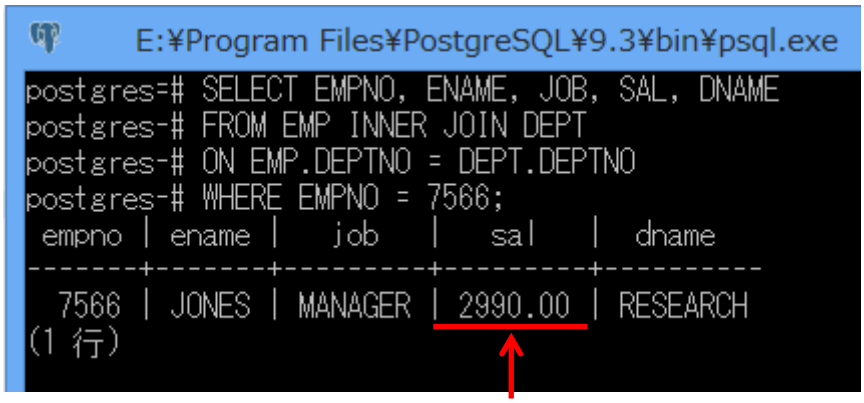


20) 実行結果の確認

ブラウザ上で JavaEE アプリケーション側に返ってきた結果を確認します：



21) psql にて変更が確定していることを確認



22) 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 へディプロイ、対応するスタブクライアントを JBoss にディプロイ

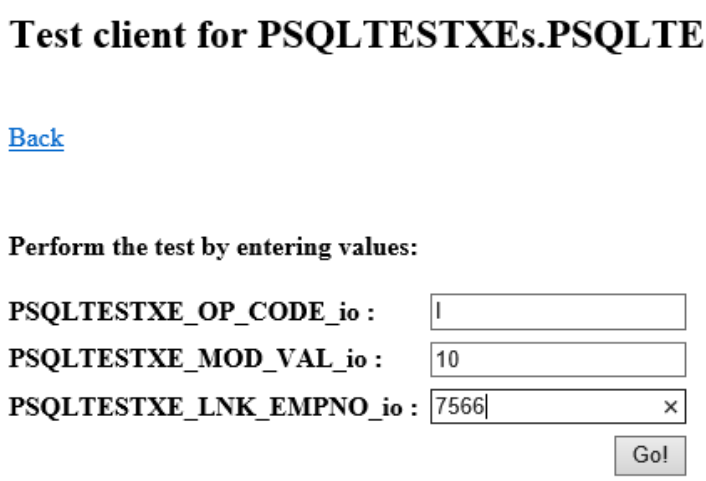
PSQLTESTXE.cbl の抜粋：

```
                                :
       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).
                                :
```

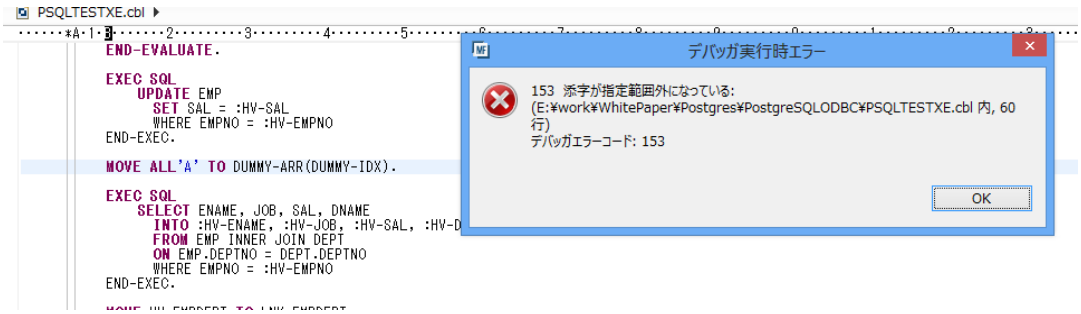
23) COBOL Enterprise Server デバッグを起動

24) 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

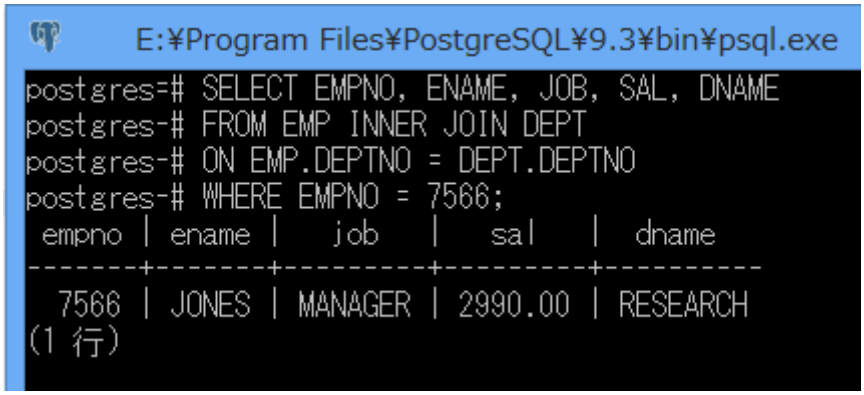
ここでは上と同じパラメータを指定します：



25) 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



26) Transaction Manager(Enterprise Server) にて UPDATE 文の変更が Rollback され、テーブル中のデータが実行前と変わっていないことを psql より確認



以上