Micro Focus Visual COBOL 2.2J

Sybase ASE 16.0 / WebLogic Server 12c

動作検証 検証結果報告書

2014 年 9 月 29 日

マイクロフォーカス株式会社

Copyright © 2014 Micro Focus. All Rights Reserved.
記載の会社名、製品名は、各社の商標または登録商標です

1. 検証概要、目的及びテスト方法

1.1 検証概要

Micro Focus Visual COBOL には COBOL 専用の Application Server として機能する Enterprise Server が付属します。この Enterprise Server が提供する Java EE Connector 機能は、JCA 仕様準拠のコンテナとして多くの Java EE 準拠アプリケーションサーバや、XA 仕様に準拠したデータベースリソースマネージャとの動作検証が行われています。
本報告書は、SAP Sybase ASE 16.0 のリソースマネージャを使用し、Oracle Weblogic Server 12c との Java EE Connector の接続性を検証し、報告するものです。

1.2 目的及びテスト方法

Micro Focus Visual COBOL の Enterprise Server が提供する Java EE Connector は、現在 Oracle Database、DB2、Microsoft SQL Server、WebSphere MQ のリソースマネージャとの連携が動作保証されています。しかし Enterprise Server は、JCA 仕様準拠のコンテナとして、設計上は XA 仕様に準拠したすべてのリソースマネージャとの連携が可能です。
SAP Sybase ASE 16.0 は、XA 仕様に準拠したリソースマネージャをサポートしており、TX Series、TUXEDO、Encina といった標準的なトランザクションマネージャとの連携が動作保証されています。従って、理論的には Enterprise Server のトランザクションマネージャとも連携するはずです。今回、以下のテストプログラムを実行することによって、上記のことを実際に検証しました。検証は全て SJIS(ja_JP.PCK) ロケール配下で実施しています。

(1) Sybase 照会プログラムのディプロイと、EJB 経由の JCA 呼び出し
(2) Sybase 更新プログラムのディプロイと、EJB 経由の JCA 呼び出しにおけるコンテナ管理トランザクション

本検証では全ての項目を Windows 環境からリモートで操作して確認しました。開発には、Visual COBOL に付属する統合化プリプロセッサ COBSQL を利用しました。本機能を利用せずプリコンパイル、コンパイルという手順を採ると場合、Visual COBOL に渡されるのはプログラマが実際にメンテナンスする埋め込み SQL プログラムではなく、埋め込み SQL 文がプリコンパイルによりライブラリ Call 等に展開されたソースとなります。本機能を利用すれば、Visual COBOL にプリコンパイル前のソースを直接渡せるため、実

際にプログラマがメンテナンスする埋め込み SQL 文が入ったソースを使って IDE 上で編集、デバッグをすることが可能です。

2. 検証環境

- **Application 開発サーバ**

  ソフトウェア

  - Oracle Solaris 11.2
  - SAP Sybase Adaptive Server Enterprise 16.0(64 bit 版)
  - SAP Sybase Adaptive Server Enterprise SDK 16.0(32 bit 版)
  - Oracle Weblogic Server 12c(12.1.3)
  - Java SE Development Kit 7, Update 67
  - Micro Focus Visual COBOL 2.2J Update 1 Development Hub(Hot Fix 4 適用版)

  ハードウェア

  機種：　Fujitsu SPARC-Enterprise-T5220
  CPU：　UltraSPARC T2
  Memory：　15.86 Gbyte memory

- **Application 開発クライアント**

  ソフトウェア

  - Windows 8.1 Pro(VM のゲスト OS として稼働)
  - Micro Focus Visual COBOL 2.2J Update 1 for Eclipse(Hot Fix 4 適用版)

  ハードウェア

  機種：　Dell OPTIPLEX7010
  CPU：　Intel Core 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

以下に実施したテストの概要を述べます。それぞれ 32 bit 及び 64 bit について検証しています。詳細な手順については付録に記載します。

3.1 Sybase 照会プログラムのディプロイと、EJB 経由の JCA 呼び出し

(1) 使用した COBOL ロジック

Sybase の pub2 データベースのテーブルから指定されたキーのレコードを SELECT しその内容を返す簡単な COBOL サブルーチンです。インターフェースマッピングはそれぞれの特性に合わせて編集した上で Enterprise Server へディプロイしました。使用したテーブルについては備考を参照してください。

(2) 使用したリソースアダプタ

mfcobol-notx.rar（トランザクションなし）

(3) 使用した Enterprise Server

32 bit の検証では Visual COBOL にビルドインされた ESDEMO を使用しました。64 bit の検証では 64 bit 用の Enterprise Server を検証用に追加し、それを利用しました。

(4) 使用した Java EE クライアント

Visual COBOL の Interface Mapping Toolkit がディプロイ時に自動生成する EJB と、自動生成される Web モジュールクライアントを使用しました。

3.2 Sybase 更新プログラムのディプロイと、EJB 経由の JCA 呼び出しにおけるコンテナ管理トランザクション

(1) 使用した COBOL ロジック

Sybase の pub2 データベースのテーブルから指定されたキーのレコードを、指定された値で UPDATE する簡単な COBOL サブルーチンです。入力されたパラメータに応じて意図的にアプリケーション例外を発生させるロジックを別途埋め込んでいます。このサブルーチンにインターフェースマッピングはデフォルトの状態のままで Enterprise Server へディプロイしています。

(2) 使用したリソースアダプタ

mfcobol-xa.rar（XA トランザクションのサポート）

(3) 使用した Enterprise Server

32 bit の検証では Visual COBOL にビルドインされた ESDEMO を使用しました。64 bit の検証では 64 bit 用の Enterprise Server を検証用に追加し、それを利用しました。

本検証においては、Sybase が提供する XA Switch を利用する XA トランザクションスイッチモジュールを 32 bit、64 bit 用にそれぞれ用意しました。これらを Enterprise Server に XA リソースとして追加登録し、使用しました。

(4) 使用した Java EE クライアント

Visual COBOL の Interface Mapping Toolkit がディプロイ時に自動生成する EJB と、自動生成される Web モジュールクライアントを使用しました。実行後、Sybase isql コマンドから、該当する Sybase テーブルのレコードへの更新が、予期された通りに COMMIT/ROLLBACK されているかを確認しています。

4. 結果

前章で上述した検証の結果は下表のとおりすべてのパターンについて正常に実行されることを確認しました。結果の詳細については付録に記載します。

試験パターン			結果
プログラムの特性	トランザクション管理	32bit /64 bit	
Sybase よりデータ照会	アプリケーション管理	32 bit	○
Sybase よりデータ照会	アプリケーション管理	64 bit	○
Sybase のデータを更新	コンテナ管理	32 bit	○
Sybase のデータを更新	コンテナ管理	64 bit	○

5. テスト結果及び考察

Oracle Weblogic Server 12c より、Sybase 上のデータをする操作する Micro Focus Visual COBOL 2.2J で開発した COBOL アプリケーションを Java EE Connector 接続にて問題なく利用できることが検証できました。

以上

付録 1. Sybase 照会プログラムのディプロイと、EJB 経由の JCA 呼び出し(32bit 編)

■ Solaris サーバ

1) 一般ユーザでログインし、Visual COBOL 及び Sybase 32 bit SDK の環境設定スクリプトを実行

2) Sybase のライブラリをリンクした共有ライブラリを作成

① Visual COBOL の動作モードを 32 bit に指定

```
$ COBMODE=32;export COBMODE
$ cobmode
Effective Default Working Mode: 32 bit
$
```

② ダミーの COBOL プログラムを用意

```
$ cat dummy.cbl
       PROCEDURE DIVISION.
            GOBACK.
$
```

③ Sybase のライブラリをリンクした共有ライブラリを作成[1]

```
$ cob -ze "" dummy.cbl -L$SYBASE/$SYBASE_OCS/lib -lsybcobct_r -lsybct_r
-sybtcl_r -lsybcs_r -lsybcomn_r -lsybintl_r -lsybunic  -lsocket -lnsl
-ldl -lpthread -lthread -lm -o SYBINIT32.so
$
```

3) Native Threads 用の Sybase プリコンパイラのシンボリックリンクを作成[2]

```
$ ln -s $SYBASE/$SYBASE_OCS/bin/cobpre_r ./cobpre
$
```

[1] $SYBASE/$SYBASE_OCS/sample/esqlcob に用意されているサンプルを SYBPLATFORM=nthread_sun_svr4 でビルドする場合にピックアップされるライブラリを指定しています。

[2] cobpre_r は SYBPLATFORM=nthread_sun_svr4 でサンプルをビルドする際に利用されるプリコンパイラです。

4) Visual COBOL にビルドインされた 32 bit Enterprise Server ESDEMO を起動

```
$ casstart
....
CASCD0167I ES Daemon successfully auto-started 15:10:50
CASCD0050I ES "ESDEMO" initiation is starting 15:10:50
$
```

5) Root ユーザに切り替え
6) Visual COBOL 及び Sybase 32 bit SDK の環境設定スクリプトを実行
7) Enterprise Server や ディプロイしたサービスの情報等を管理する MF Directory Server をバックグラウンドで起動

```
# mfds &
[1] 18236
#
```

8) COBOL リモート開発用のデーモンを起動
   ① 3) で生成したシンボリックリンクを PATH の先頭に追加[3]

```
# PATH=`pwd`:$PATH;export $PATH
#
```

   ② COBOL リモート開発用のデーモンを起動

```
# $COBDIR/remotedev/startrdodaemon
Checking Java Version
Correct Java Version installed, proceeding
Starting RSE daemon...
Daemon running on: tok-putter, port: 4075
```

[3] 生成したシンボリックリンクを cobpre として Sybase SDK に格納されているバイナリ cobpre よりも優先してピックアップさせます。

■ Windows クライアントマシン

9) Windows 上で Solaris サーバで稼働する Enterprise Server を操作するための環境を設定

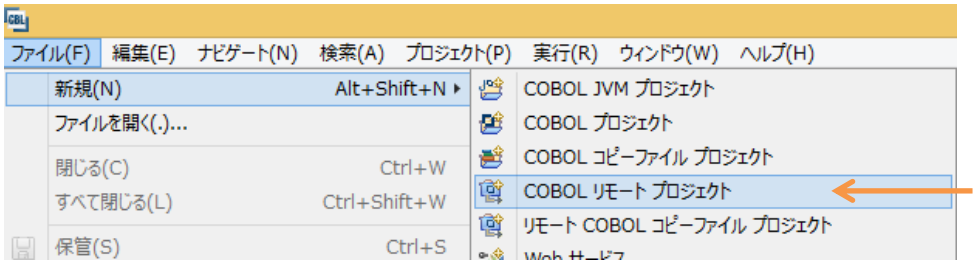
① ＜Visual COBOL のインストールフォルダ＞¥bin¥mf-client.dat をテキストエディタで開く

② [directories] 欄に
mrpi://＜Solaris サーバの IP アドレス＞:0
の形式で Solaris サーバの Directory Server エントリを追加

10) Visual COBOL for Eclipse を起動

11) COBOL リモートプロジェクトを作成

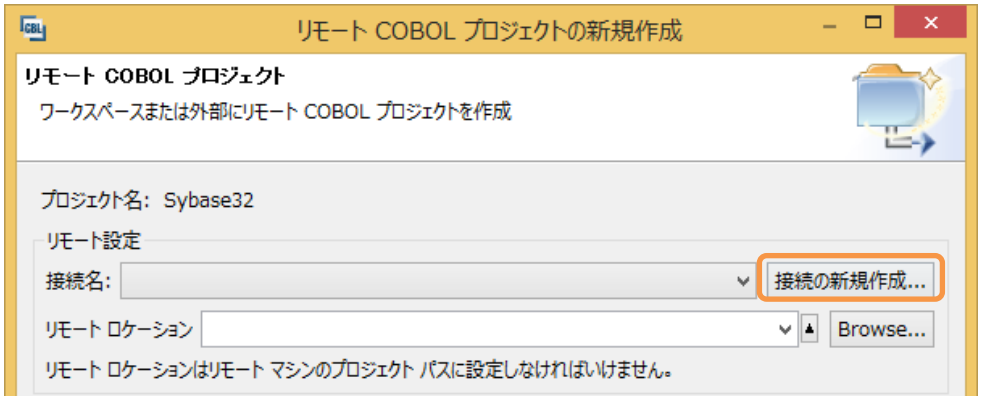
① [ファイル] メニュー > [新規] > [COBOL リモートプロジェクト] を選択



② プロジェクト名を指定し、[次へ] ボタンを押下

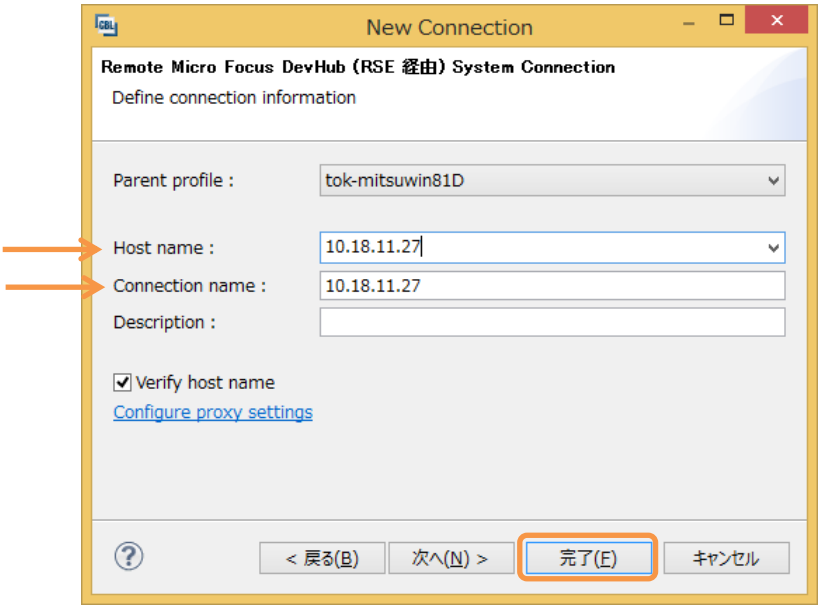
③ プロジェクトテンプレートを選択する画面では [Micro Focus テンプレート] を選択し [次へ] ボタンを押下

④ [接続の新規作成] ボタンを押下



⑤ [Micro Focus DevHub(RSE 経由)] を選択の上 [次へ] ボタンを押下

⑥ [Host name] 欄に Solaris サーバの IP アドレスを入力し、[完了] ボタンを押下

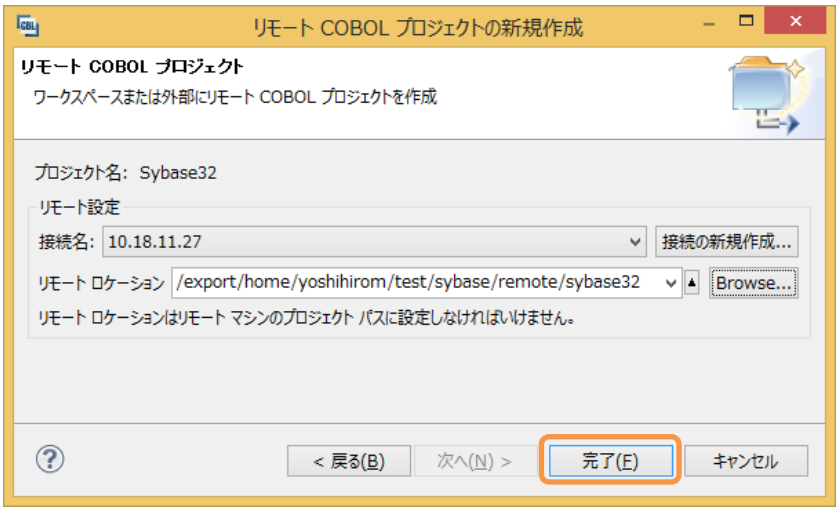


⑦ [Browse] ボタンを押下

⑧ ユーザ認証に関するポップアップが出たら Solaris サーバのユーザの認証情報を入力し、[Save Password] にチェックを入れ、[OK] ボタンを押下

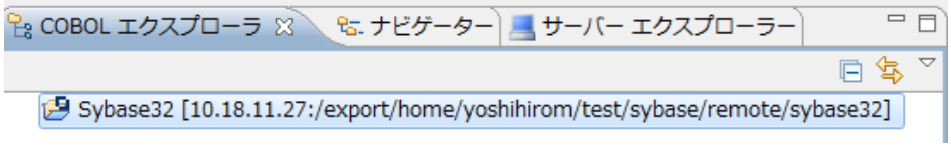
⑨ Solaris サーバ上でリモート開発のプロジェクトディレクトリとして利用するディレクトリをツリーで指定し、[OK] ボタンを押下

⑩ [完了] ボタンを押下

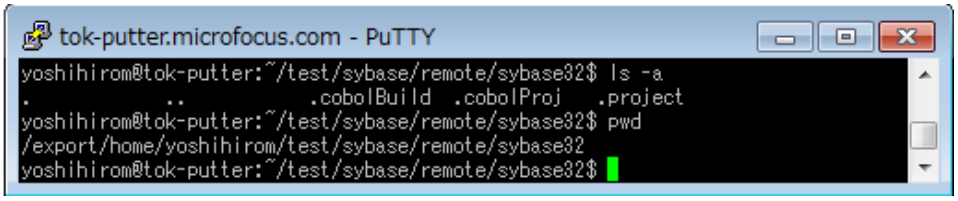


Solaris サーバ上に Eclipse の COBOL プロジェクトが生成されます：

[COBOL エクスプローラに表示される COBOL リモートプロジェクト]



[Solaris サーバ上に生成されたプロジェクトファイル]



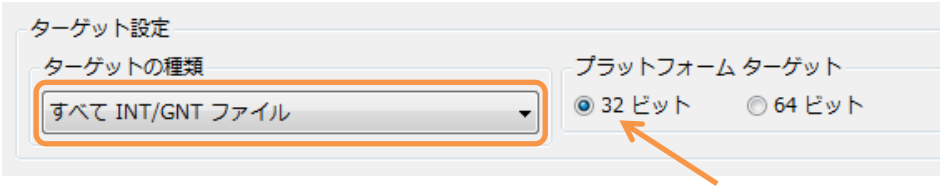
12) COBOL リモートプロジェクトを 32 bit の動的ロードモジュールを開発用に設定

① COBOL エクスプローラにて作成したプロジェクトを右クリックし、[プロパティー] を選択

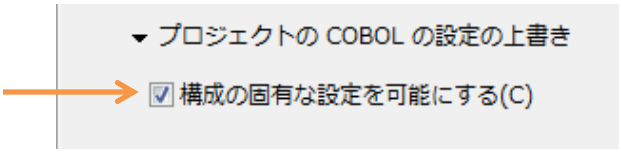
② [Micro Focus] > [ビルド構成] > [COBOL] へとナビゲート

③ [ターゲットの種類] 欄を [すべて INT/GNT ファイル] に変更

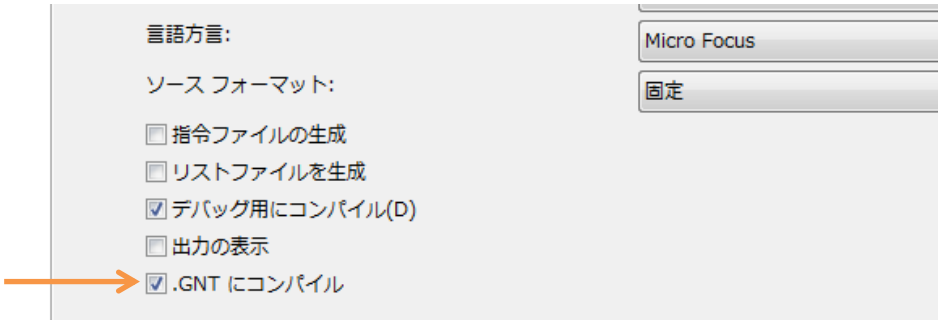
④ [プラットフォームターゲット] 欄が [32 ビット] となっていることを確認



⑤ [プロジェクトの COBOL 設定の上書き] を展開し、[構成の固有な設定を可能にする] にチェック



⑥ [.GNT にコンパイル] をチェック



⑦ [OK] ボタンを押下

13) Sybase 上のデータを照会する埋め込み SQL 文の入った COBOL プログラムをプロジェクトに追加[4]

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] > [COBOL プログラム]
を選択

② [新規ファイル名] 欄では [Sel.cbl] を指定

③ [完了] ボタンを押下

[4] 本検証で利用した実際のソースは本文書とともに公開しています。

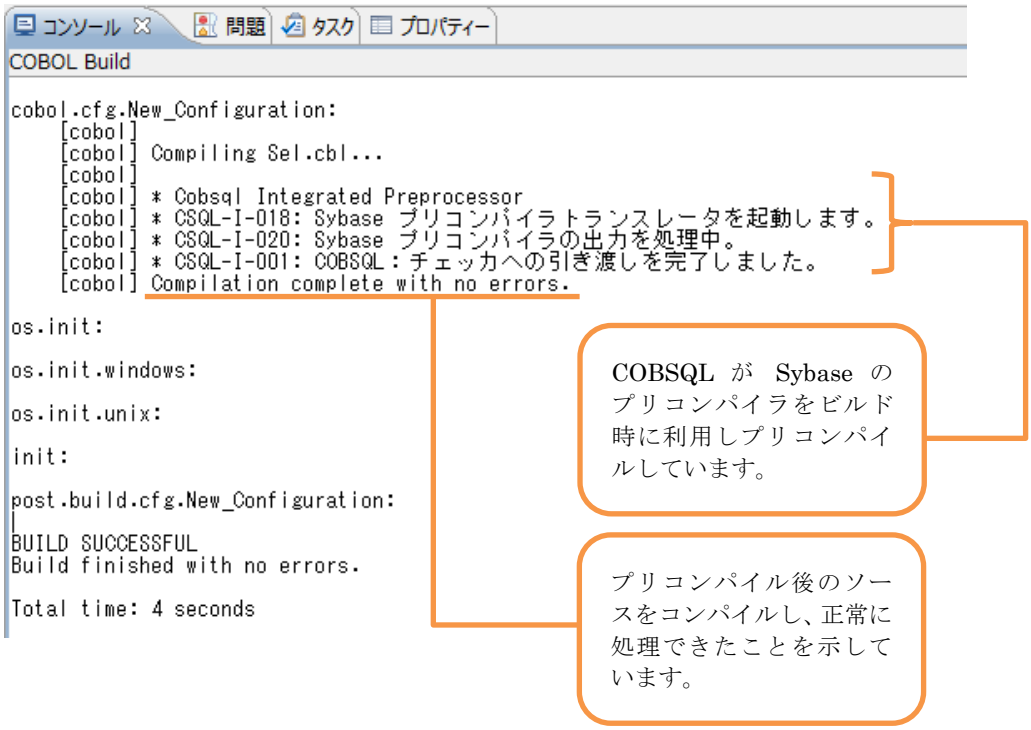
④ ソースビューに以下プログラムをコーディング[5]

```
      $SET INITCALL(SYBINIT32) p(cobsql) COBSQLTYPE=SYBASE END-C ENDP
       IDENTIFICATION DIVISION.
       PROGRAM-ID. Sel.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
           EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01 STAFF-ID           PIC S9(9) COMP-5.
       01 STAFF-NAME         PIC X(10).
           EXEC SQL END DECLARE SECTION END-EXEC.
           EXEC SQL INCLUDE SQLCA END-EXEC.
       LINKAGE SECTION.
       01 LK-STAFF-ID        PIC S9(09) COMP-5.
       01 LK-STAFF-NAME      PIC X(10).
       PROCEDURE DIVISION USING LK-STAFF-ID LK-STAFF-NAME.
       1.
           DISPLAY "CONNECT STEP"  UPON CONSOLE.
           EXEC SQL
                CONNECT "sa" IDENTIFIED BY "sapass"
                USING "TOKPUTTER"
           END-EXEC.
           DISPLAY "CONNECT SQLCODE:" SQLCODE  UPON CONSOLE.
           IF SQLCODE NOT = 0
              DISPLAY "MSG:" SQLERRMC UPON CONSOLE
              EXIT PROGRAM
           END-IF
           EXEC SQL USE pubs2 END-EXEC.
           MOVE LK-STAFF-ID TO STAFF-ID.
           EXEC SQL SELECT NAME
               INTO :STAFF-NAME
               FROM STAFF
               WHERE ID=:STAFF-ID
           END-EXEC.
           DISPLAY "SELECT SQLCODE:" SQLCODE  UPON CONSOLE.
           DISPLAY "NAME:" STAFF-NAME  UPON CONSOLE.
           MOVE STAFF-NAME TO LK-STAFF-NAME.
            EXEC SQL RELEASE END-EXEC.
           EXIT PROGRAM.
```

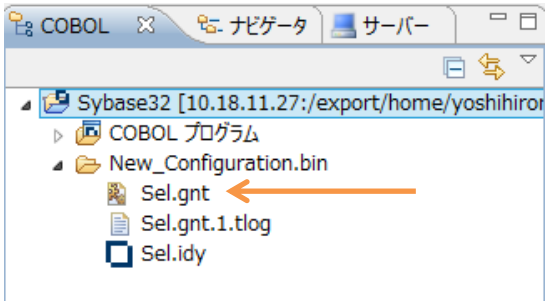
[5] 本検証では、プログラムを動的ロードモジュール形式にコンパイルします。この形式のモジュールには Sybase のライブラリをリンクインできません。そこで、これらのライブラリがリンクインされた共有ライブラリを INITCALL コンパイラ指令でプログラム実行前に呼び出し、ライブラリへの依存を解決させます。

⑤ Ctrl + S を打鍵し、ソースを保存

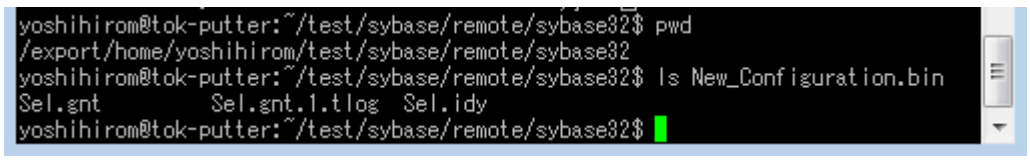
自動ビルドが有効なため、ビルド処理が走ります。正常にビルドされた旨のメッセージをコンソールビューにて確認することができます：



COBOL エクスプローラビューにて、Solaris サーバ環境に動的ロードモジュールが生成されていることを確認できます：



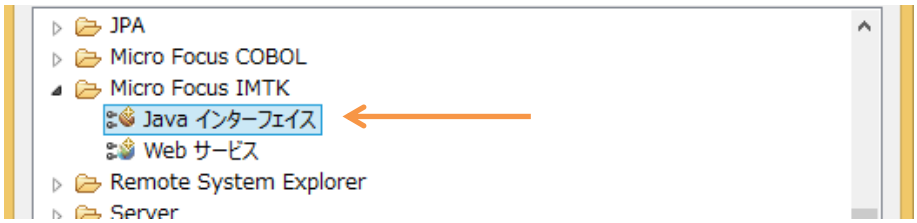
端末エミュレータでも実際に生成されていることを確認できます：



14) 検証用アプリケーションの COBOL – Java 変換マッピングを作成

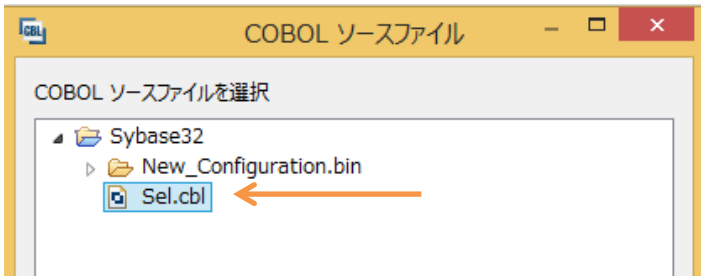
① COBOL エクスプローラにてプロジェクトを右クリックし
[新規] > [その他]
を選択

② [Micro Focus IMTK] > [Java インターフェイス] へとナビゲートし [次へ] ボタンを押下



③ Java インターフェイス名には「SelS」を指定し、[参照] ボタンを押下

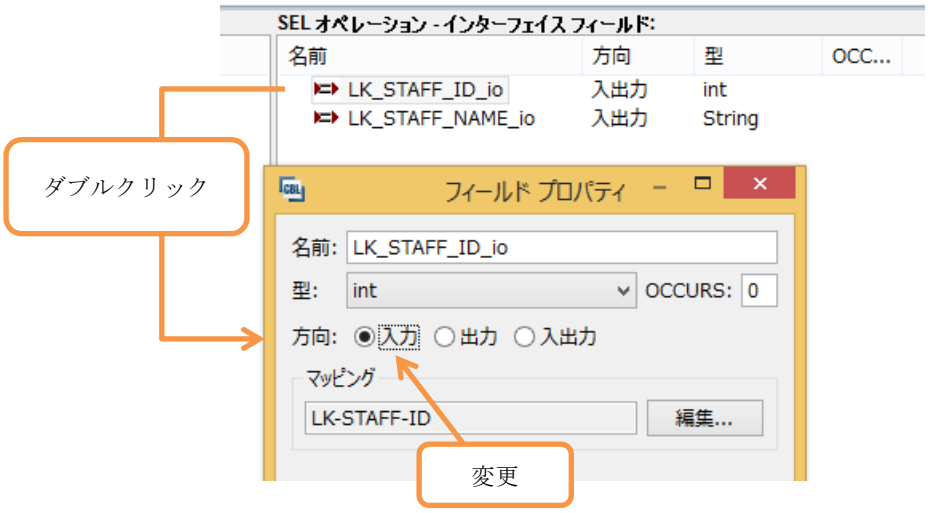
④ 追加したプログラムを選択



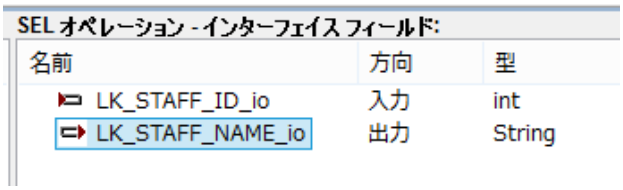
⑤ [完了] ボタンを押下

⑥ 入出力をアプリケーションの特性に合わせて、キー値として利用する STAFF ID を入力に、照会データの STAFF NAME を出力に設定

各項目をダブルクリックし、[方向] 欄を変更



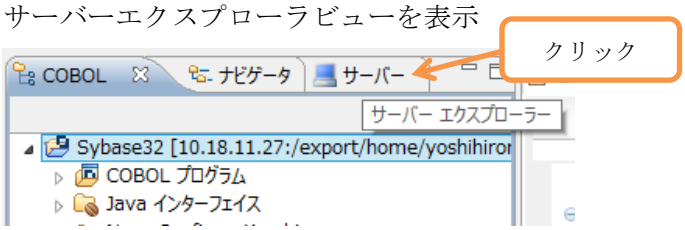
変更後の画面：



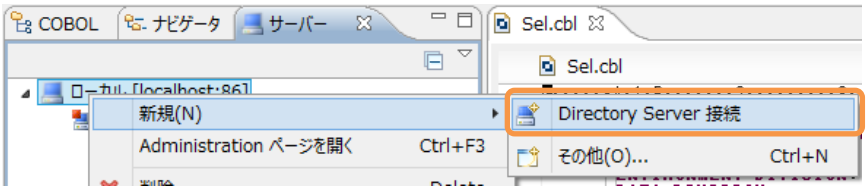
⑦ Ctrl + S を打鍵し、マッピングを保存

15) Solaris サーバの Directory サーバーをサーバーエクスプローラビューへ追加

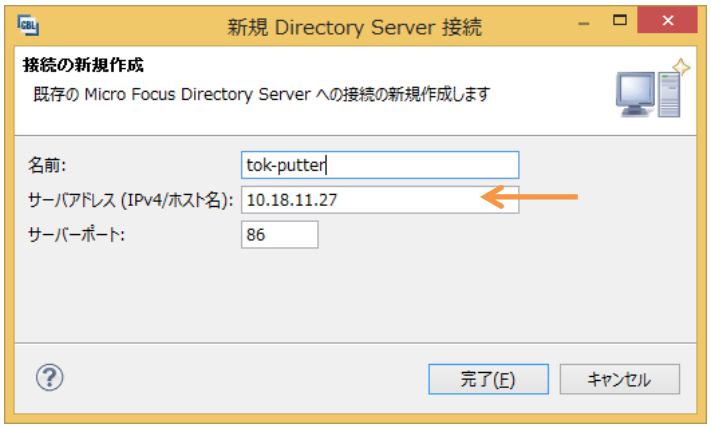
① サーバーエクスプローラビューを表示



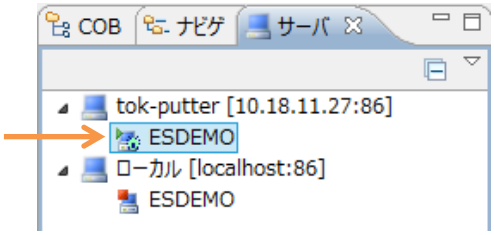
② ローカルを右クリックから [新規] > [Directory Server 接続] をクリック



③ Solaris サーバのアドレスを指定し、[完了] ボタンを押下



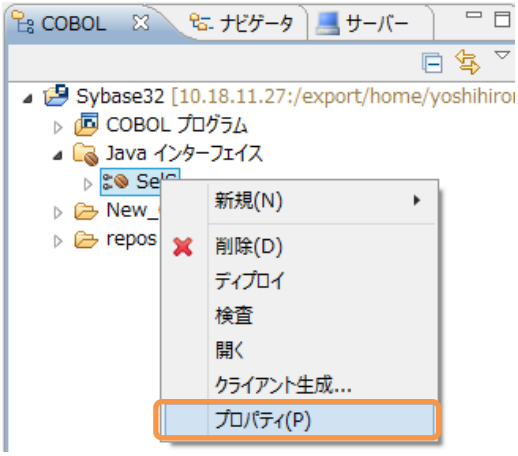
Server エクスプローラに Solaris サーバの情報が表示されます。4) にて ESDEMO を起動済のため、その旨を示すアイコンを確認できます：



16) Enterprise Server へのディプロイ情報を指定

① COBOL エクスプローラへ戻る

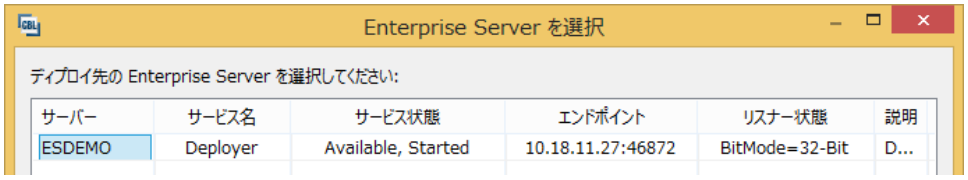
② 追加した Java インターフェイスを右クリックから［プロパティ］を選択



③ ［ディプロイメントサーバー］タブを選択

④ ［変更］ボタンを押下

⑤ Solaris サーバの ESDEMO を選択し［OK］ボタンを押下



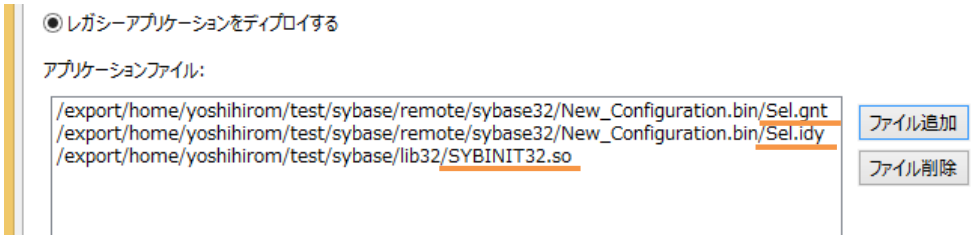
⑥ ［アプリケーションファイル］タブを選択

⑦ ［レガシーアプリケーションをディプロイする］を選択

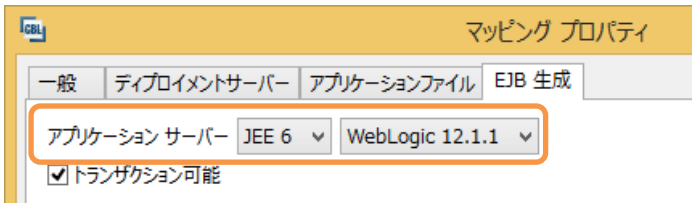
⑧ ［ファイル追加］ボタンを押下

⑨ プロジェクトディレクトリ配下の New_Configuration.bin ディレクトリに生成された Sel.gnt 及び Sel.idy を選択し［OK］ボタンを押下

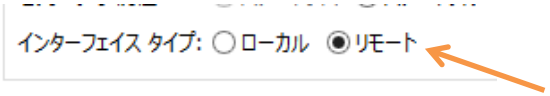
⑩ 再度［ファイル追加］ボタンを押下し、2）で作成した SYBINIT32.so を追加
追加後の画面：



⑪ ［EJB 生成］タブを選択
⑫ ［アプリケーションサーバー］欄にて JEE 6、WebLogic 12.1.1 を指定



⑬ ［インターフェイスタイプ］欄にて［リモート］を指定[6]



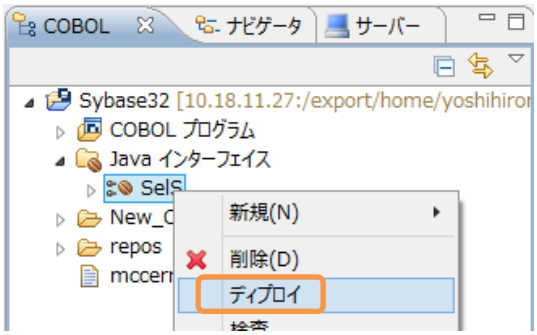
⑭ ［Java コンパイラ］欄にて利用する javac が格納されたディレクトリを指定
⑮ ［J2EE クラスパス］欄にて［参照］ボタンを押下し WebLogic のインストールディレクトリに格納されている weblogic.jar を選択

⑯ ［OK］ボタンを押下し、設定画面を閉じる

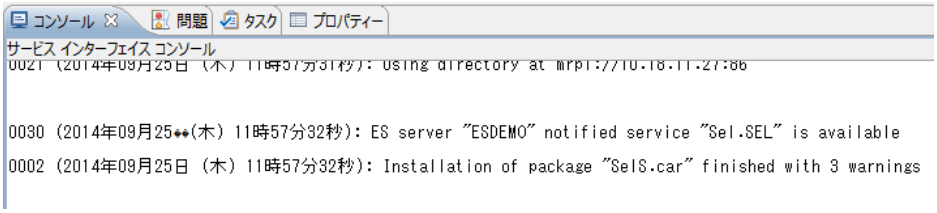
[6] Visual COBOL は EJB 3.0 のクライアントアプリケーションを生成します。この仕様では local interface を global JNDI 名で利用することを標準化しておらず、WebLogic ではデフォルトの状態では利用できないことが報告されているため当該オプションを変更します。

17) 作成した Java サービスを Enterprise Server へディプロイ

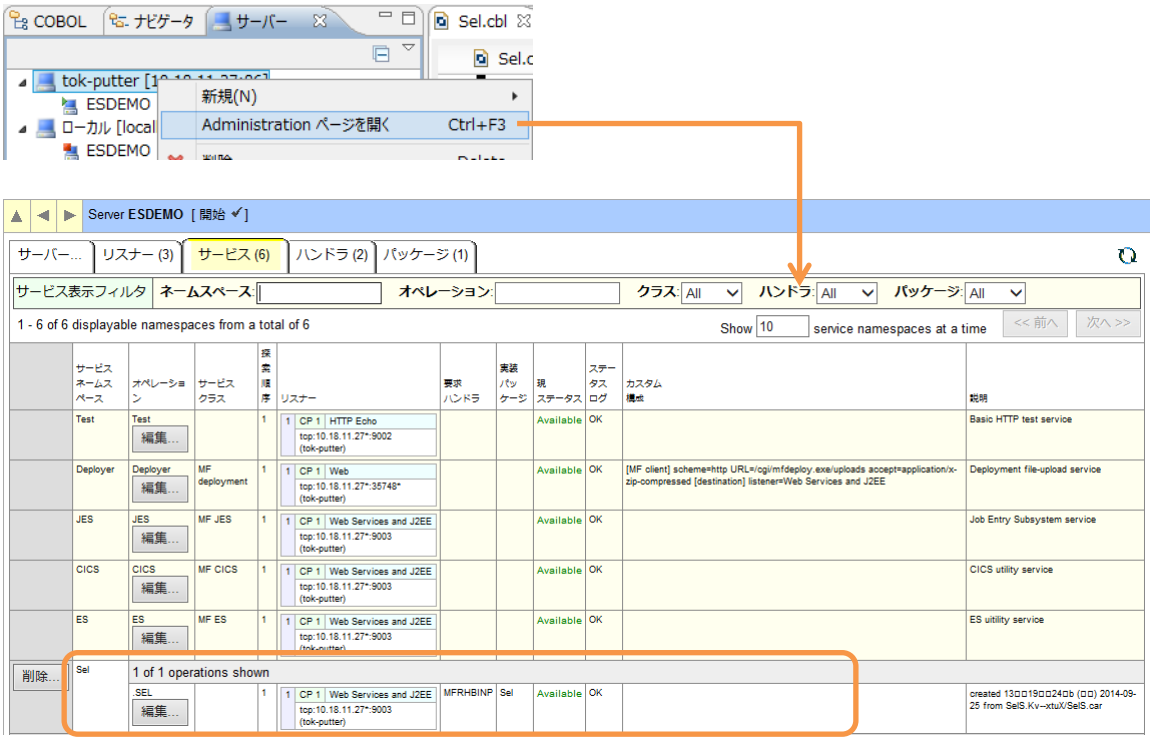
COBOL エクスプローラにて用意した Java インターフェイスを右クリックし、[ディプロイ] を選択



正常にディプロイできたことをコンソールビューにて確認することができます：

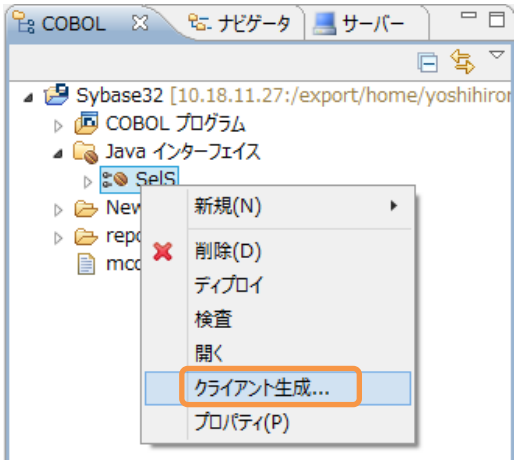


同様にサーバーエクスプローラにて Solaris サーバを右クリックし [Administration ページを開く] で表示される Administration 画面からも確認できます：



18) ディプロイしたサービスをテスト呼び出しするスタブクライアントアプリケーションを作成

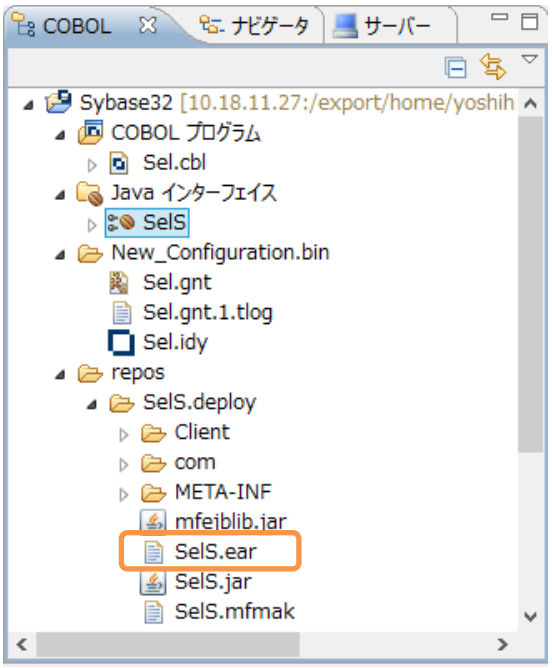
COBOL エクスプローラにて Java インターフェイスを右クリックして［クライアント生成］を選択



正常に処理されると

＜プロジェクトディレクトリ＞/repos/＜サービス名＞.deploy

配下に .ear にアーカイブされた Java EE アプリケーションが生成されます：



■ Solaris サーバ

19) WebLogic の環境設定スクリプト setWLSEnv.sh 編集し、$COBDIR/javaee/javaee6/oracleweblogic1211/mfconnector.jar を CLASSPATH に追加するよう改造して実行

20) WebLogic サーバを起動

21) Visual COBOL に付属されるトランザクションサポートのない WebLogic 12c 向けのリソースアダプタを WebLogic へ追加

① WebLogic が提供するディプロイメントツール weblogic.Deployer を利用するための環境変数を設定

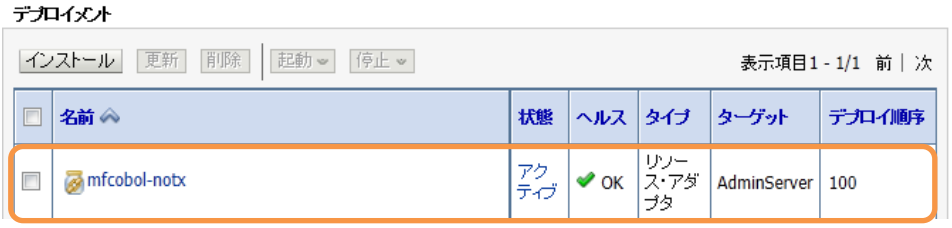
```
$ WL_USER=weblogic;export WL_USER
$ WL_PASSWD=P@ssw0rd;export WL_PASSWD
$ NAME=mfcobol-notx;export NAME
$ FULL_PATH_TO_THE_RAR_FILE=$COBDIR/javaee/javaee6/oracleweblogic121
1/mfcobol-notx.rar;export FULL_PATH_TO_THE_RAR_FILE
$
```

ディプロイする
リソースアダプタ

② weblogic.Deployer を使ってリソースアダプタを WebLogic へディプロイ

```
$ java -classpath $WL_HOME/server/lib/weblogic.jar weblogic.Deployer
 -username $WL_USER -password $WL_PASSWD -name $NAME -deploy $FULL_P
ATH_TO_THE_RAR_FILE
weblogic.Deployer がオプション -username weblogic -name mfcobol-notx
-deploy /opt/mf/ED22U1HF4/javaee/javaee6/oracleweblogic1211/mfcobol-
notx.rar を指定して呼び出されました
<2014/09/25 14 時 01 分 07 秒 JST> <Info> <J2EE Deployment SPI> <BEA-26
0121> <Initiating deploy operation for application, mfcobol-notx [ar
chive: /opt/mf/ED22U1HF4/javaee/javaee6/oracleweblogic1211/mfcobol-n
otx.rar], to configured targets.>
タスク 0 が開始されました: [Deployer:149026]デプロイ application mfcob
ol-notx on AdminServer.
タスク 0 完了: [Deployer:149026]デプロイ application mfcobol-notx on
AdminServer.
ターゲットの状態: サーバー AdminServer で deploy 完了

$
```

正常にディプロイできたことを WebLogic Server Administration Console より確認できます：



22) 18) で生成したスタブクライアントを WebLogic へインストール

① WebLogic Server Administration Console へログイン

② 左ペインより対象のドメイン配下の［デプロイメント］をクリック

③ ［インストール］ボタンを押下

④ ［パス］欄に生成された .ear が格納されたディレクトリを指定し Enter を打鍵

⑤ 生成された SelS.ear を選択し、［次へ］ボタンを押下

⑥ ［このデプロイメントをアプリケーションとしてインストールする］が選択されていることを確認して、［次へ］ボタンを押下

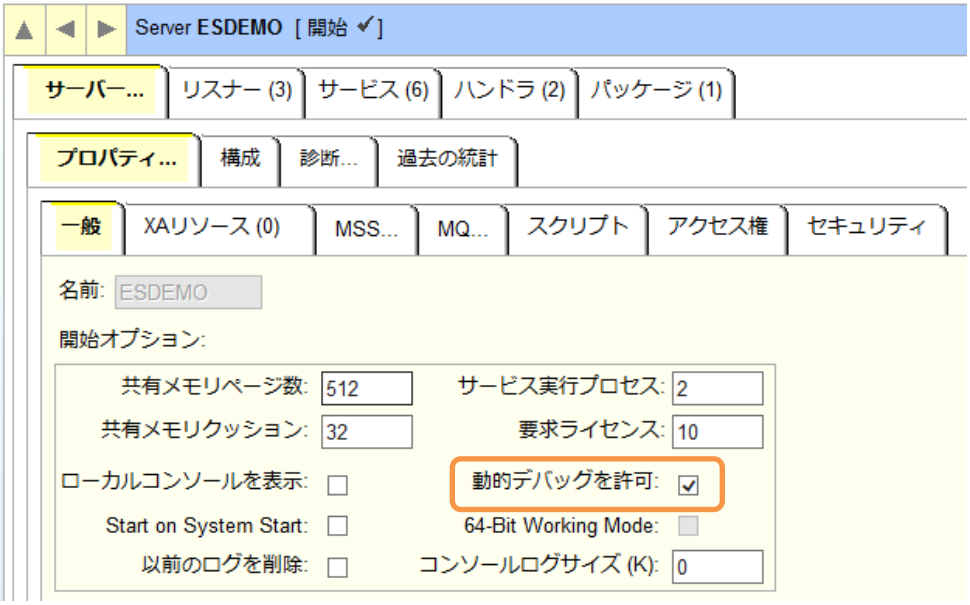
⑦ ［終了］ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます：

■ Windows クライアントマシン

23) Enterprise Server が動的デバッグ有効で起動されていることを確認

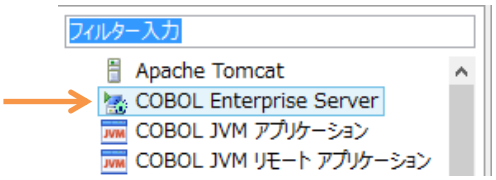
Enterprise Server Administration 画面を開き、ESDEMO の行における［編集］ボタンを押下し、確認[7]



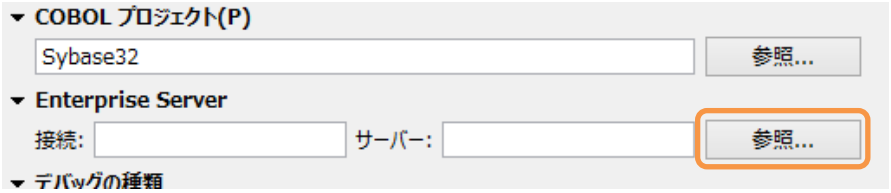
24) Enterprise Server デバッグを起動

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

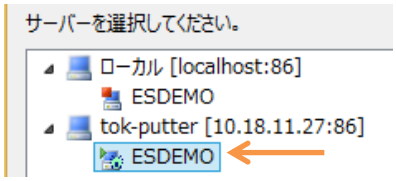


③ [Enterprise Server] 欄の［参照］ボタンを押下

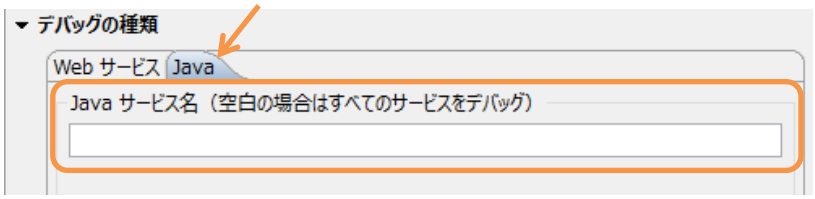


[7] 本オプションはデフォルトではチェックが入っていません。チェックが入っていない場合は、Enterprise Server を停止し、オプションをチェックの上起動します。

④ Solaris サーバ上で稼働する ESDEMO を選択し、[OK] ボタンを押下

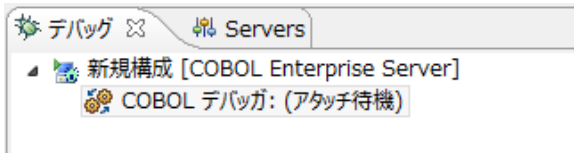


⑤ [Java] タブをクリックし、全てのサービスがデバッグ対象となっていることを確認



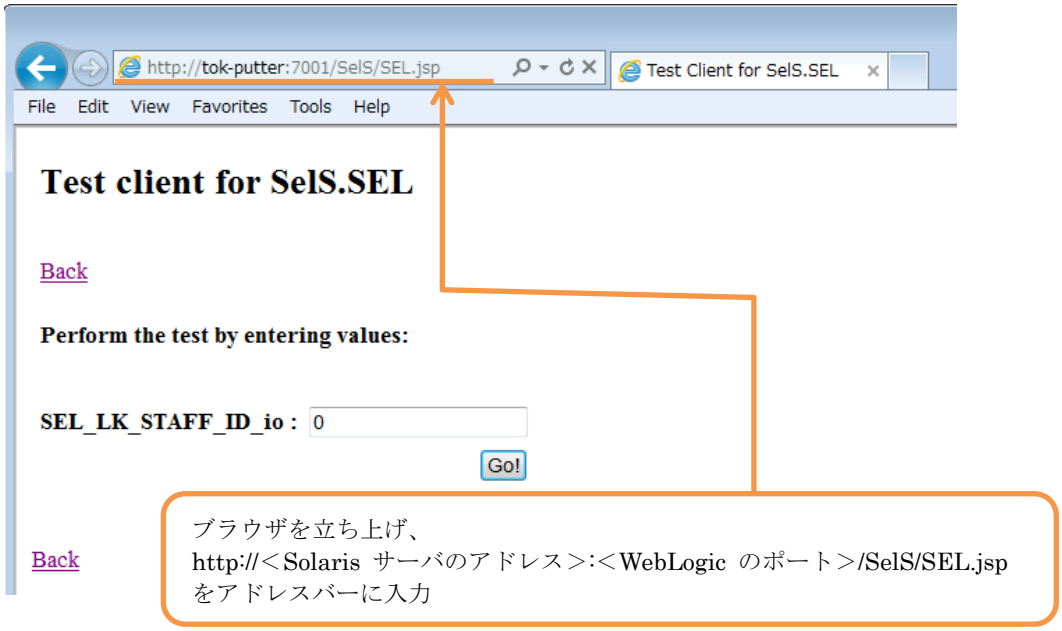
⑥ [デバッグ] ボタンを押下

⑦ [パースペクティブの切り替えの確認] 画面にて［はい］を選択
デバッグパースペクティブにてアタッチ待機状態になっていることを確認できます：

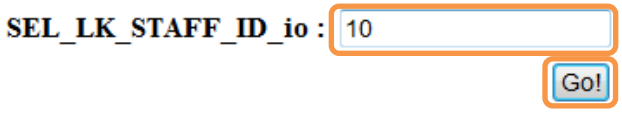


25) ディプロイしたアプリケーションをデバッグ実行

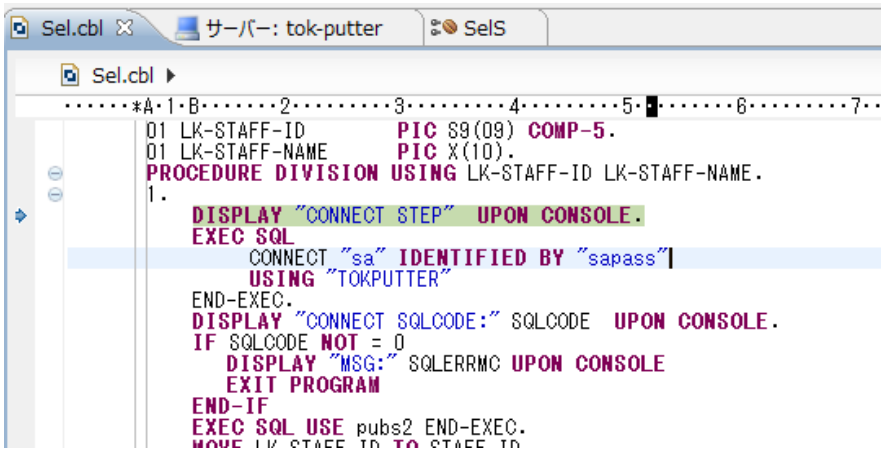
① WebLogic にディプロイしたスタブクライアントアプリケーションを起動



② [SEL_LK_STAFF_ID_io] 欄に [10] を入力し [Go!] ボタンを押下



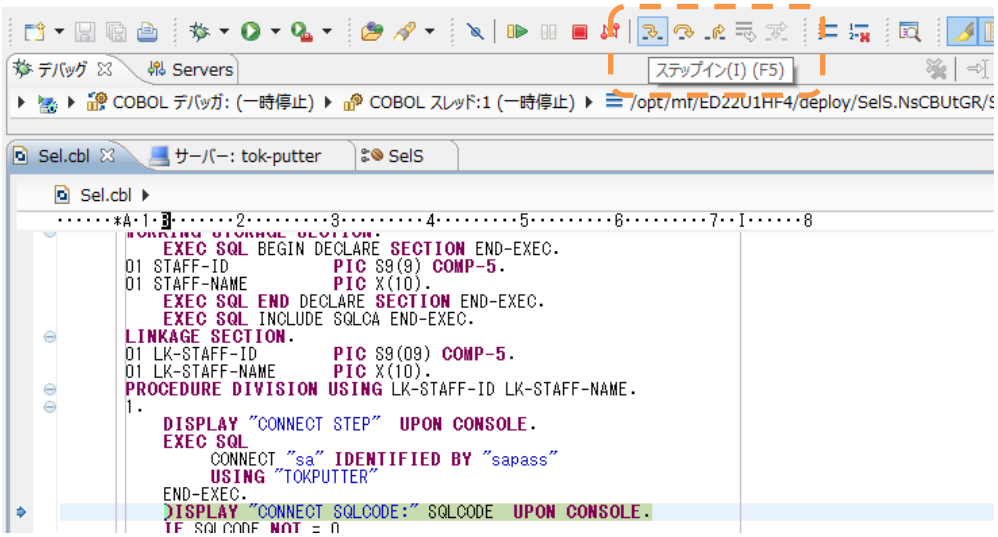
処理が Enterprise Server に渡り、Eclipse のデバッガがその処理を引き込みます。Enterprise Server にディプロイした COBOL プログラムの最初の行を実行する前で処理が停止しているのが確認できます：



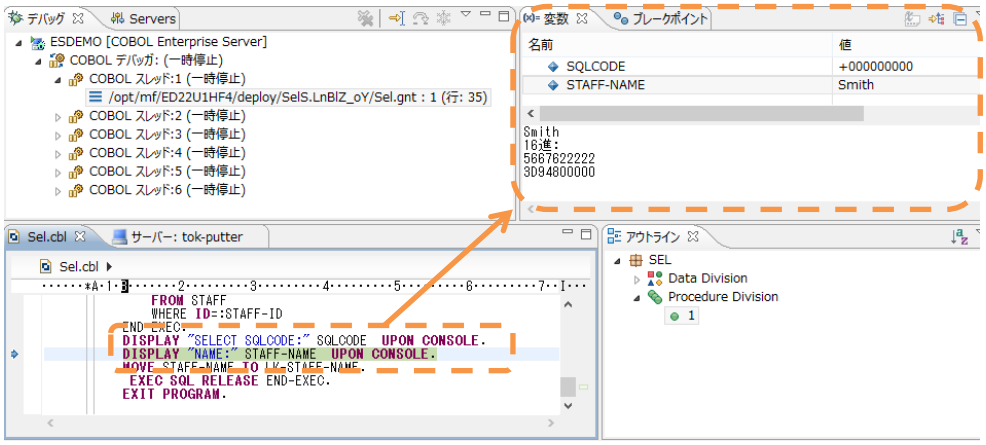
26) ディプロイしたアプリケーションをデバッグ実行

① デバッグ実行

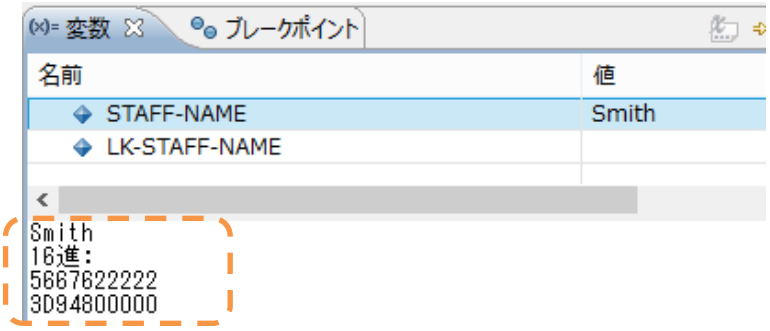
F5 キー打鍵で 1 ステップずつ処理を進めることができます。尚、このプログラムは COBSQL を利用してコンパイルしているため、プリコンパイル後のソースではなく埋め込み SQL 文が入った実際にプログラマがメンテナンスするソースでデバッグができます：



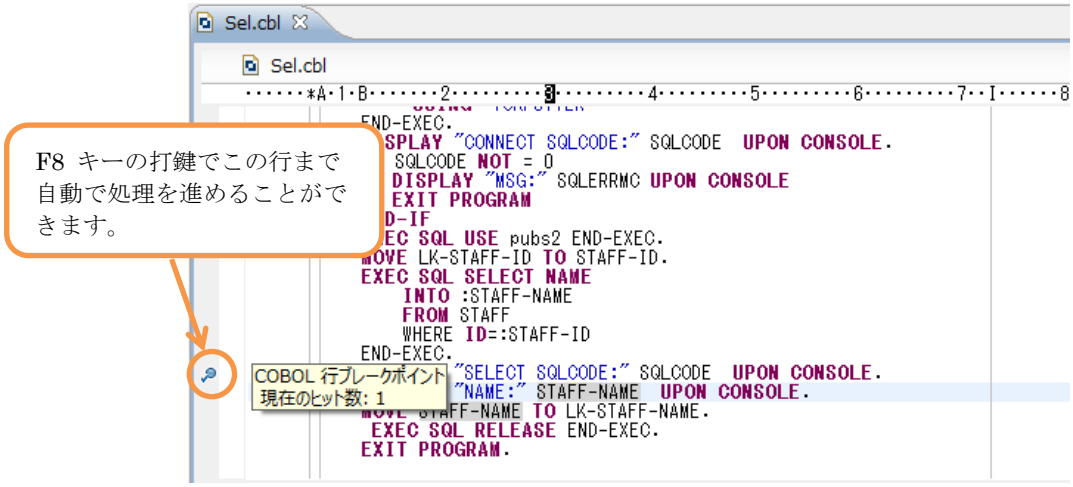
変数ビューで現在のステップで参照している変数の中身を確認できます：



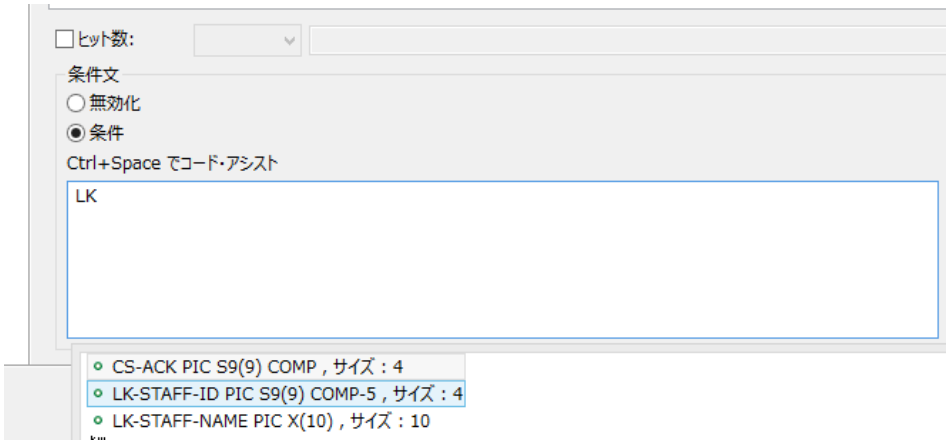
データの中身は 16 進表示にすることもできます：



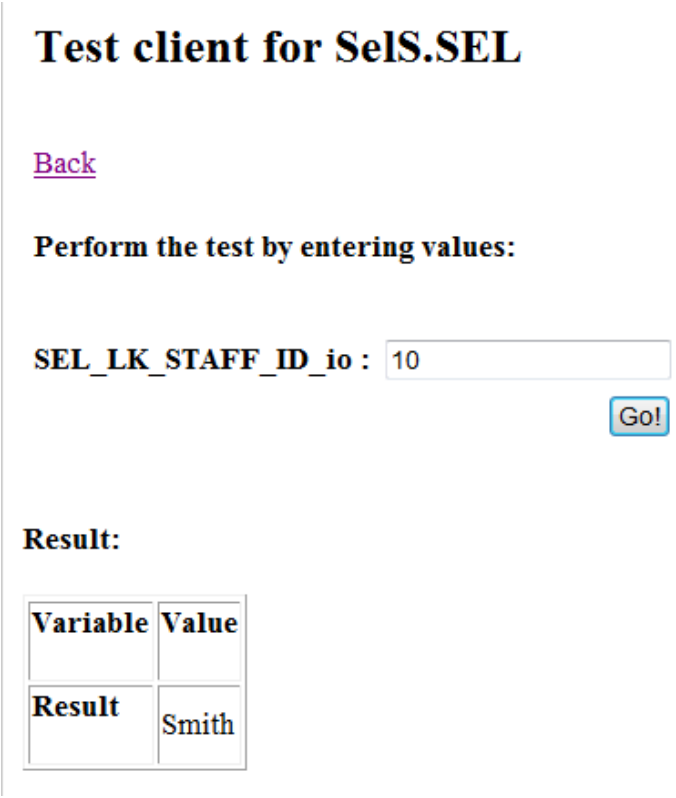
ソース中にブレークポイントを指定し、任意の位置まで処理を自動実行させることも可能です。更に、このブレークポイントに停止条件を指定することもできます：



ブレークポイントプロパティ画面：



処理を最後まで進めると、COBOL で Sybase より取得したデータが表示され、Java - COBOL – Sybase が正常に連携できていることを確認できます：



■ Solaris サーバ

27) Enterprise Server を停止

```
$ casstop
CASST0005I Shutdown of ES ESDEMO starting 15:20:30
CASSI8003I Enterprise Server "ESDEMO" termination completed 15:20:30
Return code:    0
$
```

28) COBOL リモート開発用のデーモンを開始したセッションにて COBOL リモート開発用のデーモンを 停止

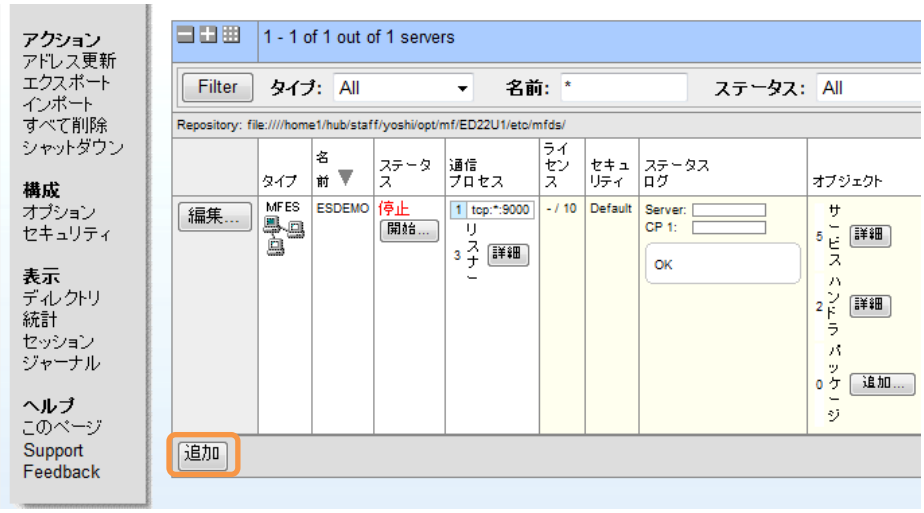
```
# $COBDIR/remotedev/stoprdodaemon
Process 19203 located:

     UID   PID  PPID   C      STIME TTY         TIME CMD

Do you wish to continue and kill it? (y/n): y
kill: 19203: no such process
Kill signal sent to process 19203
#
```

付録 2. Sybase 照会プログラムのディプロイと、EJB 経由の JCA 呼び出し(64bit 編)

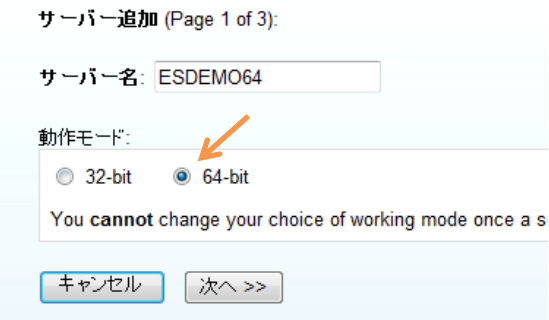
■ Windows クライアントマシン

1) 64 bit 版 Enterprise Server を追加

① Enterprise Server Administration ページのトップ画面にて［追加］ボタンを押下



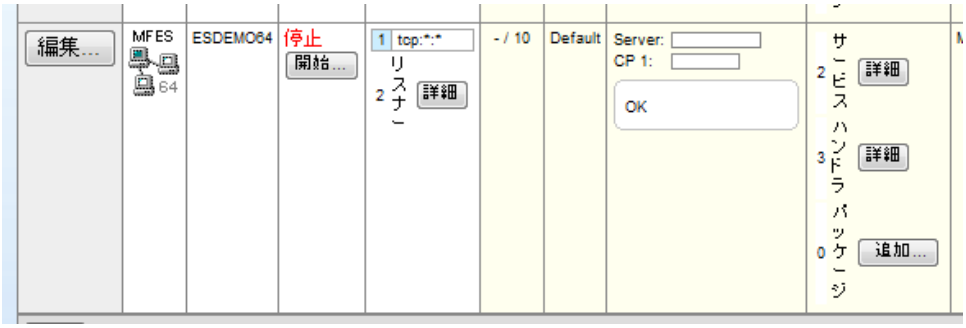
② サーバ名を指定し、動作モードを［64-bit］に選択し、［次へ］ボタンを押下



③ サーバタイプは［Micro Focus Enterprise Server］を選択し［次へ］ボタンを押下

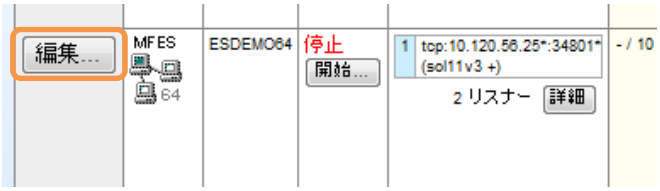
④ オプションの設定欄はデフォルトのまま［追加］ボタンを押下

トップ画面に戻り、追加されたことが確認できます：

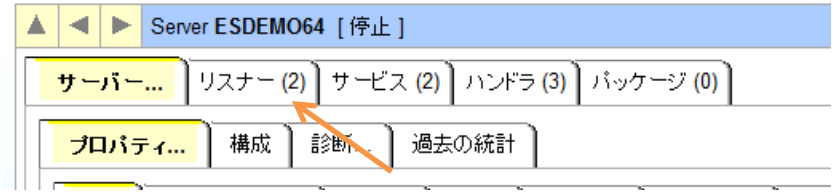


2) Enterprise Server の Listener に固定のポートを割り当て

① Enterprise Server Administration 画面にて追加した Enterprise Server の行にある［編集］ボタンを押下



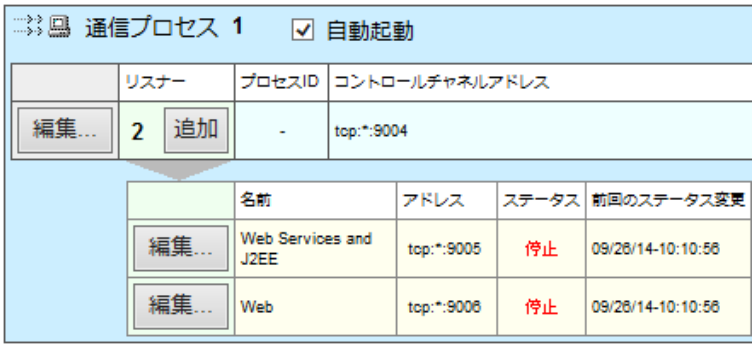
② ［リスナー］タブを選択



③ それぞれのリスナーの［編集］ボタンを押下し、下表のように設定

リスナー名	ポート番号
プロセス	9004
Web Services and J2EE	9005
Web	9006

設定後の画面：



■ Solaris サーバ

3) 一般ユーザでログインし、Visual COBOL 及び Sybase 64 bit SDK の環境設定スクリプトを実行[8]

4) Sybase のライブラリをリンクした共有ライブラリを作成

① Visual COBOL の動作モードを 64 bit に指定

```
$ COBMODE=64;export COBMODE
$ cobmode
Effective Default Working Mode: 64 bit ←
$
```

② 付録１で作成したダミーの COBOL プログラムをコピー

③ Sybase のライブラリをリンクした共有ライブラリを作成[9]

```
$ cob -ze "" dummy.cbl -L$SYBASE/$SYBASE_OCS/lib -lsybcobct_r64
-lsybct_r64 -lsybtcl_r64 -lsybcs_r64 -lsybcomn_r64 -lsybintl_r64
-lsybunic64  -lsocket -lnsl -ldl -lpthread -lthread -lm -o SYBINIT64.so
$
```

[8] SYBASE の環境設定用スクリプトは LD_LIBRARY_PATH_64 も設定します。Solaris では LD_LIBRARY_PATH と LD_LIBRARY_PATH_64 の両者が存在する場合、64 bit リンクには LD_LIBRARY_PATH_64 が使用されます。そのため、この環境変数を維持する場合は、$COBDIR/lib を LD_LIBRARY_PATH_64 に追加する必要があります。

[9] $SYBASE/$SYBASE_OCS/sample/esqlcob に用意されているサンプルを SYBPLATFORM=nthread_sun_svr464 でビルドする場合にピックアップされるライブラリを指定しています。

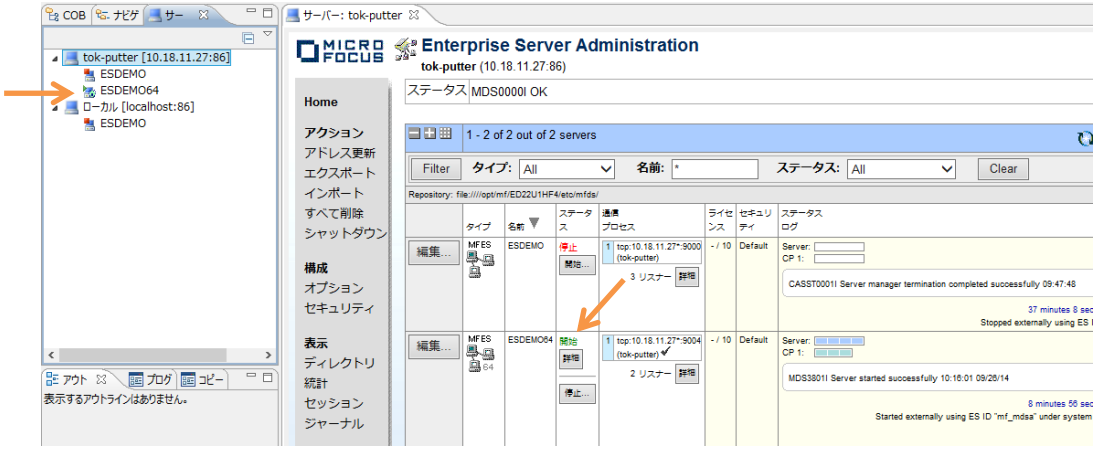
5) Native Threads 用の Sybase プリコンパイラのシンボリックリンクを作成[10]

```
$ ln -s $SYBASE/$SYBASE_OCS/bin/cobpre_r64 ./cobpre
$
```

6) 1) で作成した 64 bit Enterprise Server ESDEMO64 を起動

```
$ casstart -rESDEMO64
....
CASCD0167I ES Daemon successfully auto-started 10:16:00
CASCD0050I ES "ESDEMO64" initiation is starting 10:16:00
$
```

サーバーエクスプローラビュー及び Administration 画面で起動できたことを確認できます：



7) Root ユーザに切り替え

8) 環境変数を設定の上、COBOL リモート開発用のデーモンを起動

① Visual COBOL 及び Sybase 64 bit SDK の環境設定スクリプトを実行

② 5) で生成したシンボリックリンクを PATH に追加

③ COBOL リモート開発用のデーモンを起動

```
# $COBDIR/remotedev/startrdodaemon
Checking Java Version
Correct Java Version installed, proceeding
Starting RSE daemon...
Daemon running on: tok-putter, port: 4075
```

[10] cobpre_r64 は SYBPLATFORM=nthread_sun_svr464 でサンプルをビルドする際に利用されるプリコンパイラです。

■ Windows クライアントマシン

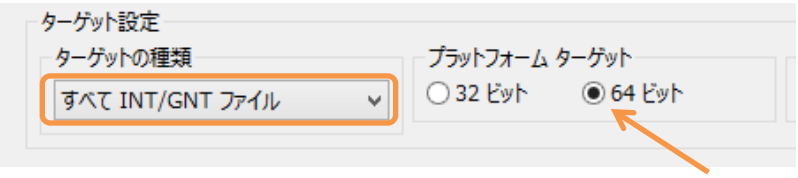
9) Visual COBOL for Eclipse を起動

10) COBOL リモートプロジェクトを作成

① [ファイル] メニュー > [新規] > [COBOL リモートプロジェクト] を選択
② プロジェクト名を指定し、[次へ] ボタンを押下
③ プロジェクトテンプレートを選択する画面では [Micro Focus テンプレート] を選択し [次へ] ボタンを押下
④ [接続の新規作成] ボタンを押下
⑤ [Micro Focus DevHub(RSE 経由)] を選択の上 [次へ] ボタンを押下
⑥ [Host name] 欄に Solaris サーバの IP アドレスを入力し、[完了] ボタンを押下
⑦ [Browse] ボタンを押下
⑧ ユーザ認証に関するポップアップが出たら Solaris サーバのユーザの認証情報を入力し、[Save Password] にチェックを入れ、[OK] ボタンを押下
⑨ Solaris サーバ上でリモート開発のプロジェクトディレクトリとして利用するプロジェクトをツリーで指定し、[OK] ボタンを押下
⑩ [完了] ボタンを押下

11) COBOL リモートプロジェクトを 64 bit の動的ロードモジュールを開発用に設定

① COBOL エクスプローラにて作成した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] 欄を [すべて INT/GNT ファイル] に変更
④ [プラットフォームターゲット] 欄を [64 ビット] に指定

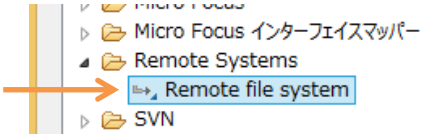


⑤ [プロジェクトの COBOL 設定の上書き] を展開し、[構成の固有な設定を可能にする] にチェック
⑥ [.GNT にコンパイル] をチェック
⑦ [OK] ボタンを押下

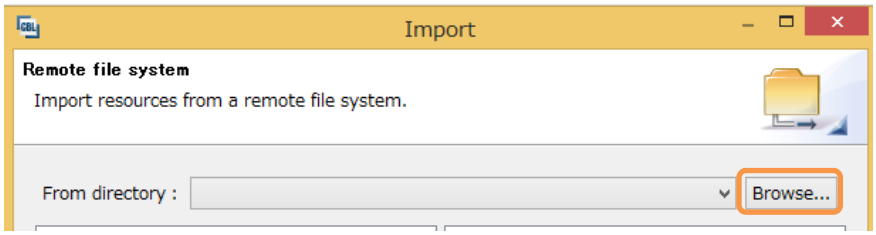
12) Sybase 上のデータを照会する埋め込み SQL 文の入った COBOL プログラムをプロジェクトに追加（付録１の検証と同じプログラムを利用します。）

① COBOL エクスプローラ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]
を選択

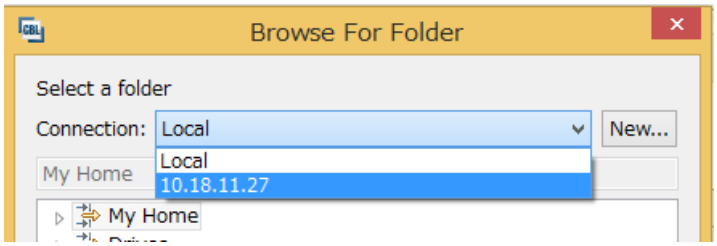
② [Remote Systems] > [Remote file system] へとナビゲートし、[次へ] ボタンを押下



③ [Browse] ボタンを押下

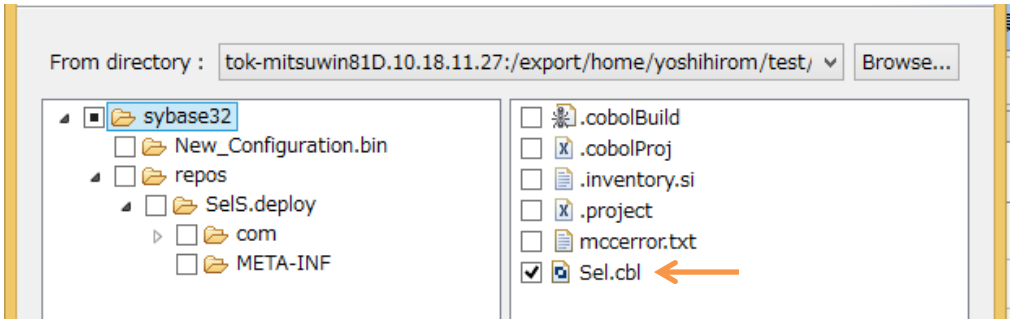


④ [Connection] 欄で Solaris サーバのアドレスを選択



⑤ 付録１で使用した COBOL リモートプロジェクトのディレクトリへナビゲートし [OK] ボタンを押下

⑥ 付録１で使用した Sel.cbl のみにチェックを入れ、[完了] ボタンを押下



13) コンパイラ指令を 4) で作成した利用する共有ライブラリに合わせて変更
下記のようにソースの１行目を編集：

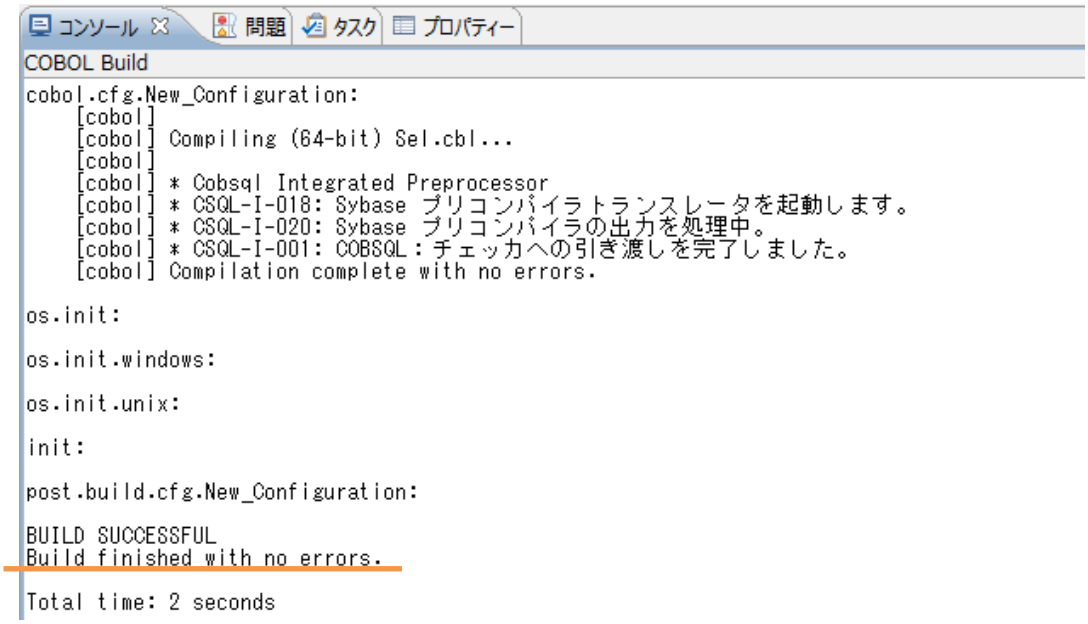
[編集前]

```
$SET INITCALL(SYBINIT32) p(cobsql) COBSQLTYPE=SYBASE END-C ENDP
 IDENTIFICATION DIVISION.
 PROGRAM-ID. Sel.
```

[編集後]

```
$SET INITCALL(SYBINIT64) p(cobsql) COBSQLTYPE=SYBASE END-C ENDP
 IDENTIFICATION DIVISION.
 PROGRAM-ID. Sel.
```

変更後、ソースを保存するとビルドされ、変更が反映された .gnt が生成されます：



14) 検証用アプリケーションの COBOL – Java 変換マッピングを作成

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] へとナビゲートし [次へ] ボタンを押下

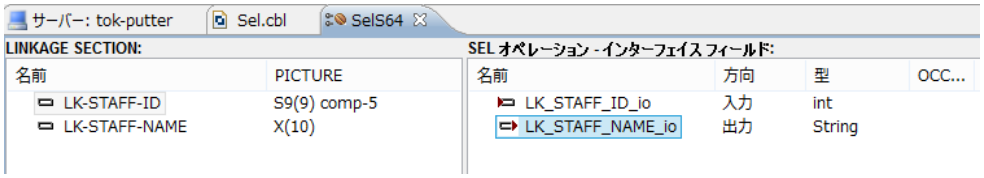
③ 付録１で使用したものとは異なる Java インターフェイス名「SelS64」を指定し、[参照] ボタンを押下

④ 追加したプログラムを選択

⑤ [完了] ボタンを押下

⑥ 入出力をアプリケーションの特性に合わせて、キー値として利用する STAFF ID を入力に、照会データの STAFF NAME を出力に設定

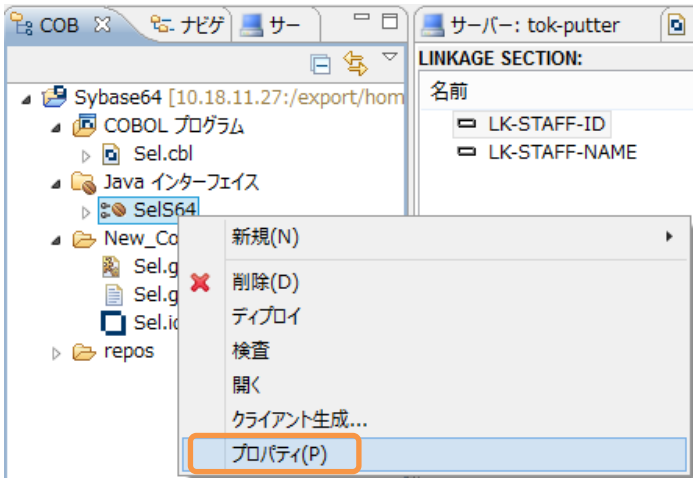
変更後の画面：



⑦ Ctrl + S を打鍵し、マッピングを保存

15) Enterprise Server へのディプロイ情報を指定

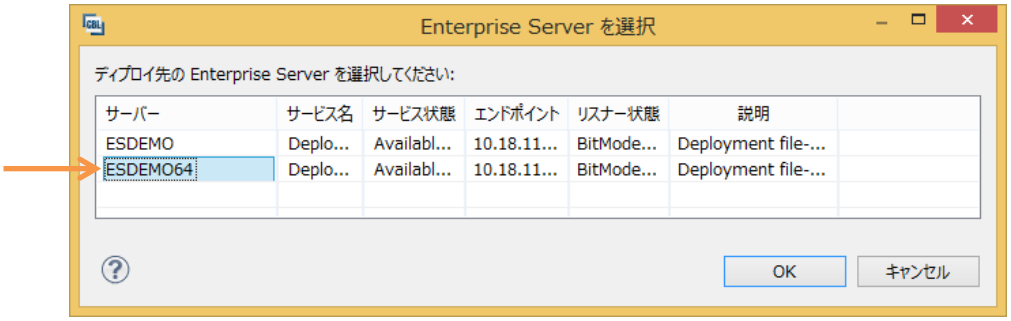
① COBOL エクスプローラにて追加した Java インターフェイスを右クリックから［プロパティ］を選択



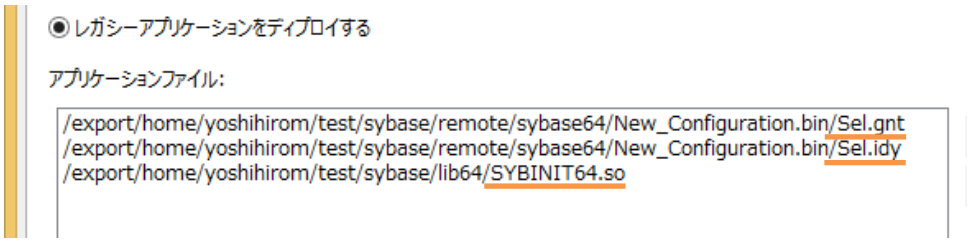
② ［ディプロイメントサーバー］タブを選択

③ ［変更］ボタンを押下

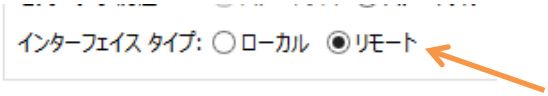
④ Solaris サーバで稼働する 1) で追加した Enterprise Server ESDEMO64 を選択し［OK］ボタンを押下



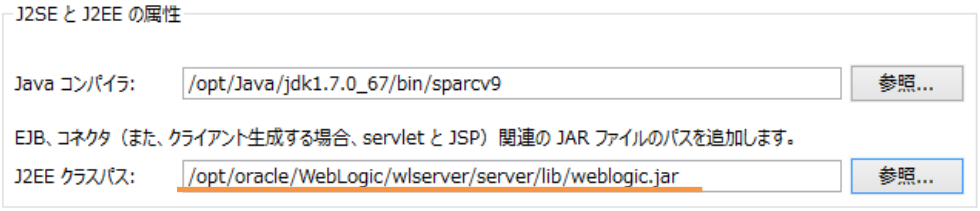
⑤ [アプリケーションファイル] タブを選択
⑥ [レガシーアプリケーションをディプロイする] を選択
⑦ [ファイル追加] ボタンを押下
⑧ プロジェクトディレクトリ配下の New_Configuration.bin ディレクトリに生成された Sel.gnt 及び Sel.idy を選択し [OK] ボタンを押下
⑨ 再度 [ファイル追加] ボタンを押下し、4) で作成した SYBINIT64.so を追加
追加後の画面：



⑩ [EJB 生成] タブを選択
⑪ [アプリケーションサーバー] 欄にて JEE 6、WebLogic 12.1.1 を指定
⑫ [インターフェイスタイプ] 欄にて [リモート] を指定



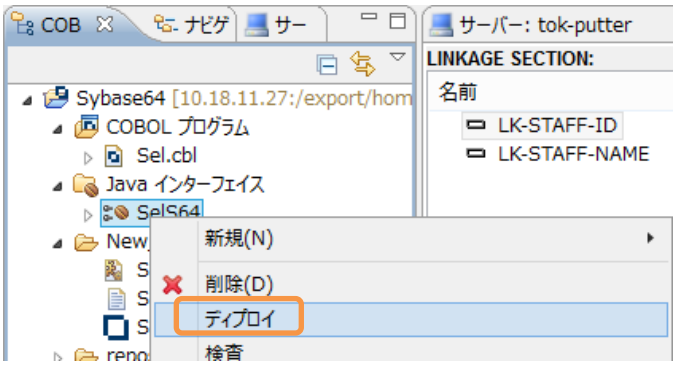
⑬ [Java コンパイラ] 欄にて利用する javac が格納されたディレクトリを指定
⑭ [J2EE クラスパス] 欄にて [参照] ボタンを押下し WebLogic のインストールディレクトリに格納されている weblogic.jar を選択



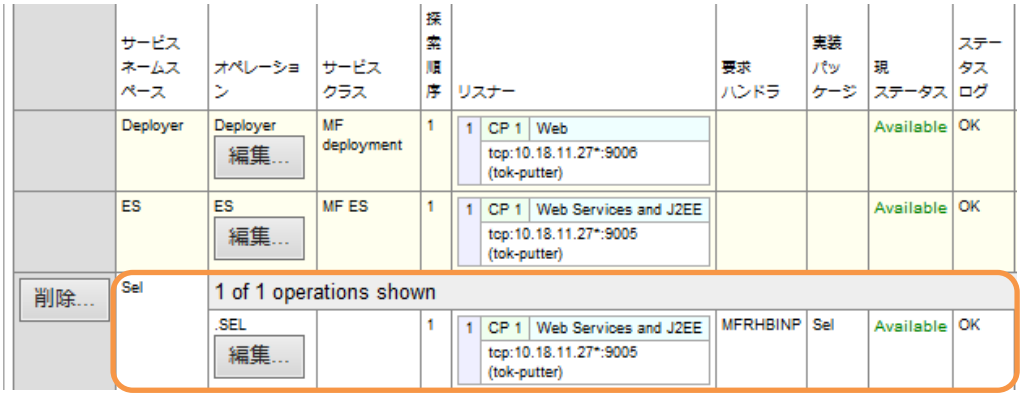
⑮ [OK] ボタンを押下し、設定画面を閉じる

16) 作成した Java サービスを Enterprise Server へディプロイ

COBOL エクスプローラにて用意した Java インターフェイスを右クリックし、[ディプロイ] を選択

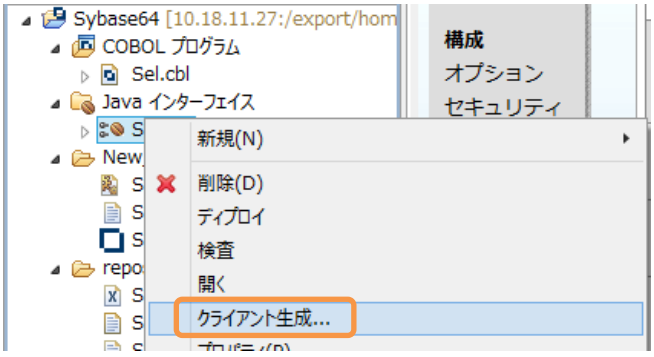


正常にディプロイできたことを Enterprise Server Administration 画面等から確認することができます：

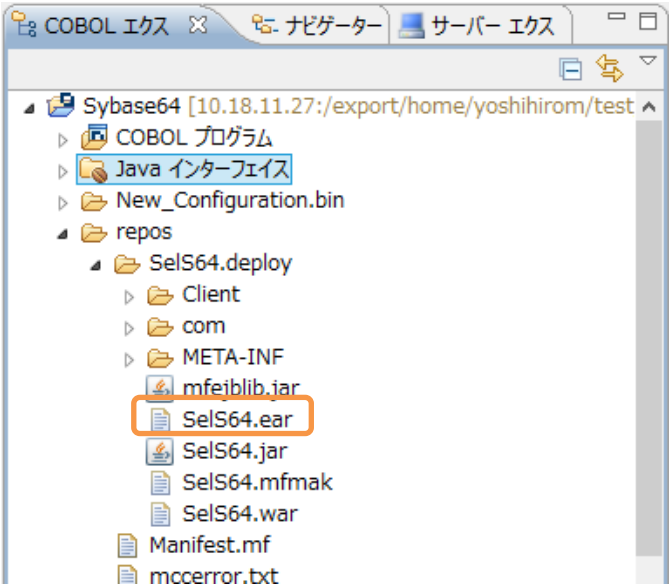


17) ディプロイしたサービスをテスト呼び出しするスタブクライアントアプリケーションを作成

COBOL エクスプローラにて Java インターフェイスを右クリックして［クライアント生成］を選択

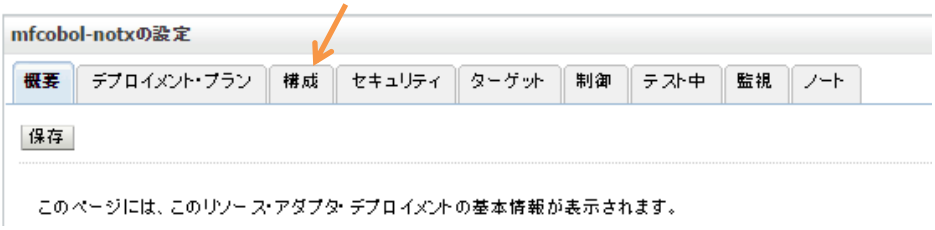


正常に処理されると
＜プロジェクトディレクトリ＞/repos/＜サービス名＞.deploy
配下に .ear にアーカイブされた Java EE アプリケーションが生成されます：



18) 追加した Enterprise Server ESDEMO64 へポイントするようリソースアダプタのプロパティを編集

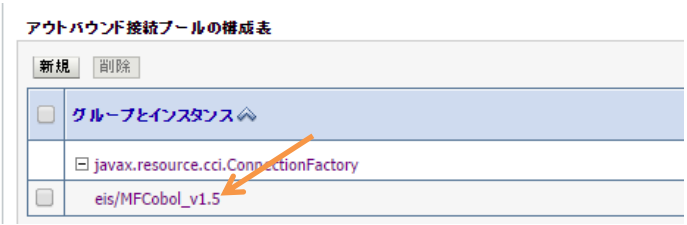
① WebLogic Server Administration Console へログイン
② 左ペインより対象のドメイン配下の［デプロイメント］をクリック
③ 追加したリソースアダプタをクリック
④ ［構成］タブをクリック



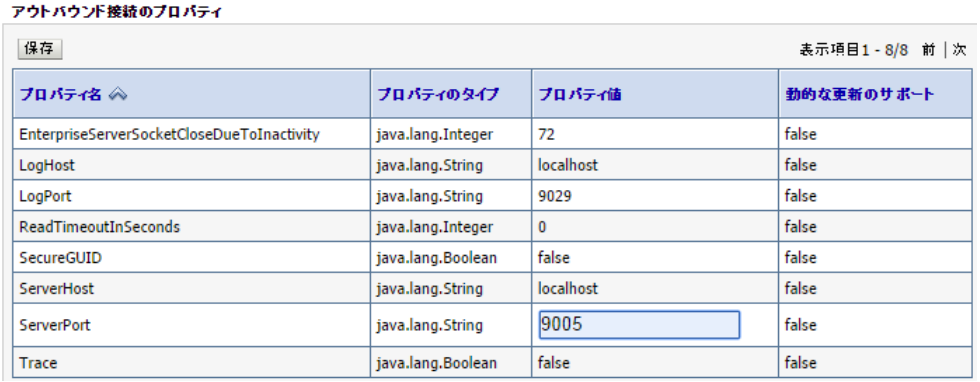
⑤ ［アウトバウンド接続プール］をクリック

⑥ javax.resource.cci.ConnectionFactory を展開

⑦ eis/MFCobol_v1.5 をクリック

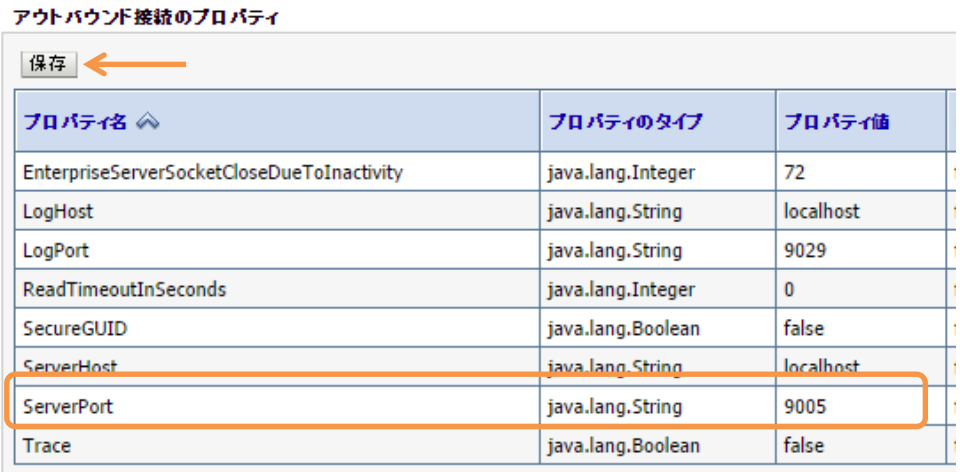


⑧ ServerPort 欄をクリックし、デフォルトの 9003 から上で指定した 9005 へポート番号を変更



プロパティ名	プロパティのタイプ	プロパティ値	動的な更新のサポート
EnterpriseServerSocketCloseDueToInactivity	java.lang.Integer	72	false
LogHost	java.lang.String	localhost	false
LogPort	java.lang.String	9029	false
ReadTimeoutInSeconds	java.lang.Integer	0	false
SecureGUID	java.lang.Boolean	false	false
ServerHost	java.lang.String	localhost	false
ServerPort	java.lang.String	9005	false
Trace	java.lang.Boolean	false	false

⑨ [保存] ボタンを押下



プロパティ名	プロパティのタイプ	プロパティ値
EnterpriseServerSocketCloseDueToInactivity	java.lang.Integer	72
LogHost	java.lang.String	localhost
LogPort	java.lang.String	9029
ReadTimeoutInSeconds	java.lang.Integer	0
SecureGUID	java.lang.Boolean	false
ServerHost	java.lang.String	localhost
ServerPort	java.lang.String	9005
Trace	java.lang.Boolean	false

⑩ デプロイメント・プランの保存確認画面では［OK］ボタンを押下

29) 17) で生成したスタブクライアントを WebLogic へインストール

① WebLogic Server Administration Console へログイン

② 左ペインより対象のドメイン配下の［デプロイメント］をクリック

③ ［インストール］ボタンを押下

④ ［パス］欄に生成された .ear が格納されたディレクトリを指定し Enter を打鍵

⑤ 生成された SelS64.ear を選択し、［次へ］ボタンを押下

⑥ ［このデプロイメントをアプリケーションとしてインストールする］が選択されていることを確認して、［次へ］ボタンを押下

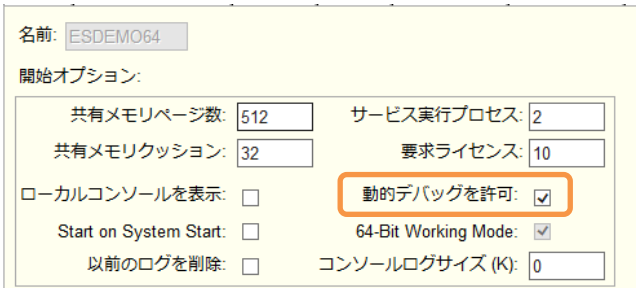
⑦ ［終了］ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます：

30) Enterprise Server が動的デバッグ有効で起動されていることを確認

Enterprise Server Administration 画面を開き、ESDEMO64 の行における［編集］ボタンを押下し、確認[11]

[11]本オプションはデフォルトではチェックが入っていません。チェックが入っていない場合は、Enterprise Server を停止し、オプションをチェックの上起動します。



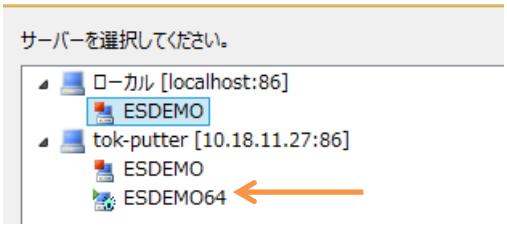
31) Enterprise Server デバッグを起動

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

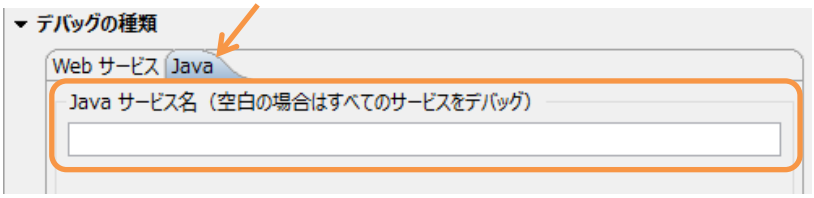
② [COBOL Enterprise Server] をダブルクリック

③ [Enterprise Server] 欄の [参照] ボタンを押下

④ Solaris サーバ上で稼働する ESDEMO64 を選択し、[OK] ボタンを押下

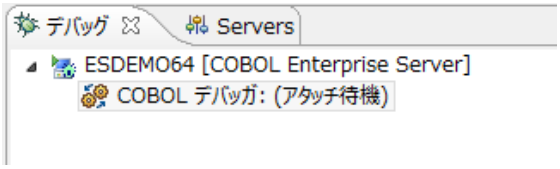


⑤ [Java] タブをクリックし、全てのサービスがデバッグ対象となっていることを確認



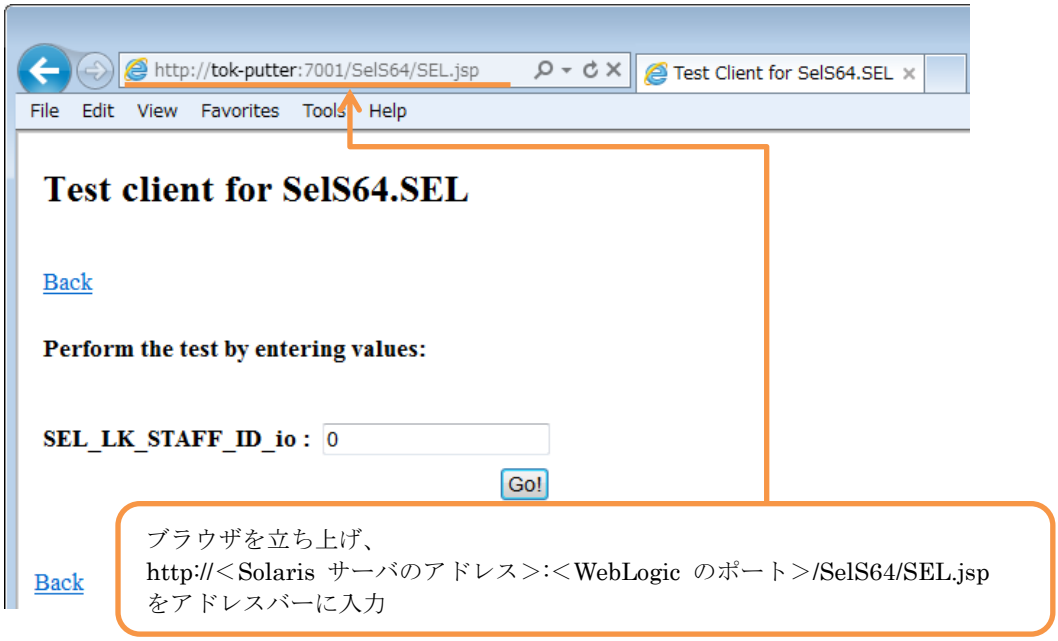
⑥ [デバッグ] ボタンを押下

⑦ [パースペクティブの切り替えの確認] 画面にて [はい] を選択
デバッグパースペクティブにてアタッチ待機状態になっていることを確認できます：

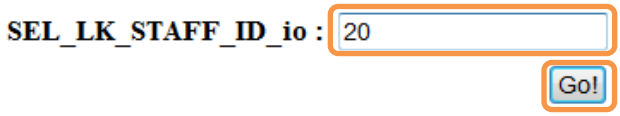


32) ディプロイしたアプリケーションをデバッグ実行

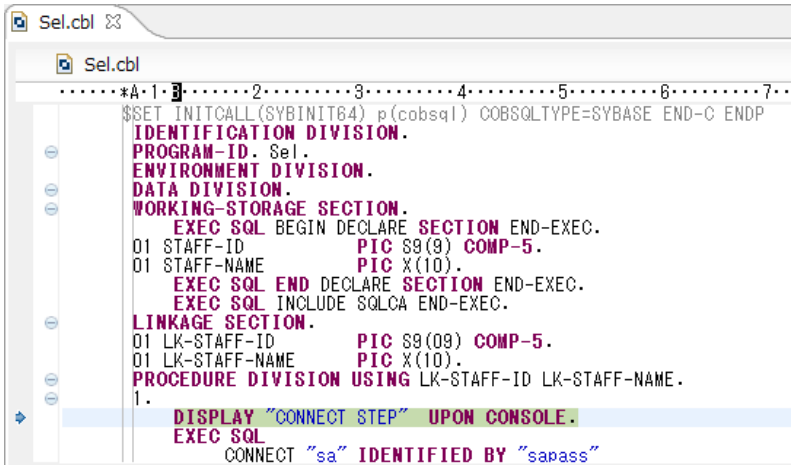
① WebLogic にディプロイしたスタブクライアントアプリケーションを起動



② [SEL_LK_STAFF_ID_io] 欄に [20] を入力し [Go!] ボタンを押下

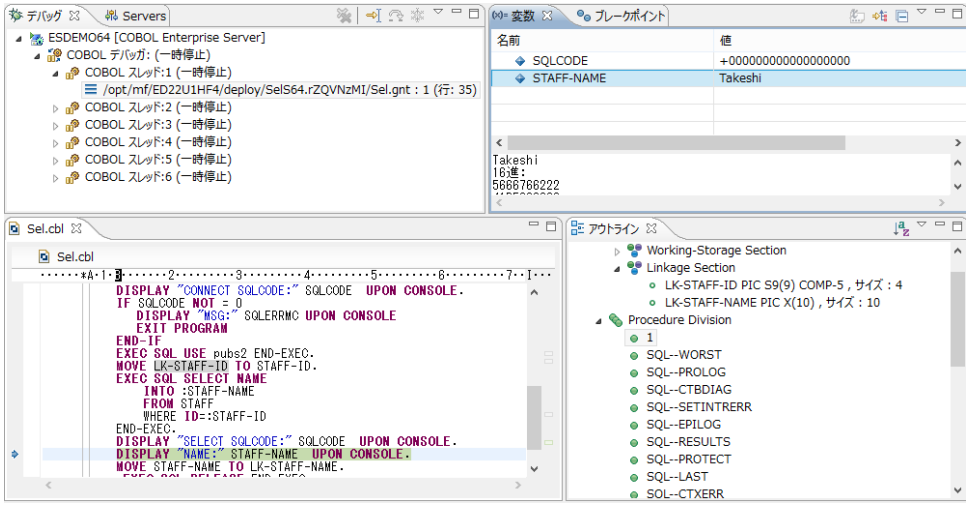


処理が Enterprise Server に渡り、Eclipse のデバッガがその処理を引き込みます。Enterprise Server にディプロイした COBOL プログラムの最初の行を実行する前で処理が停止しているのが確認できます：

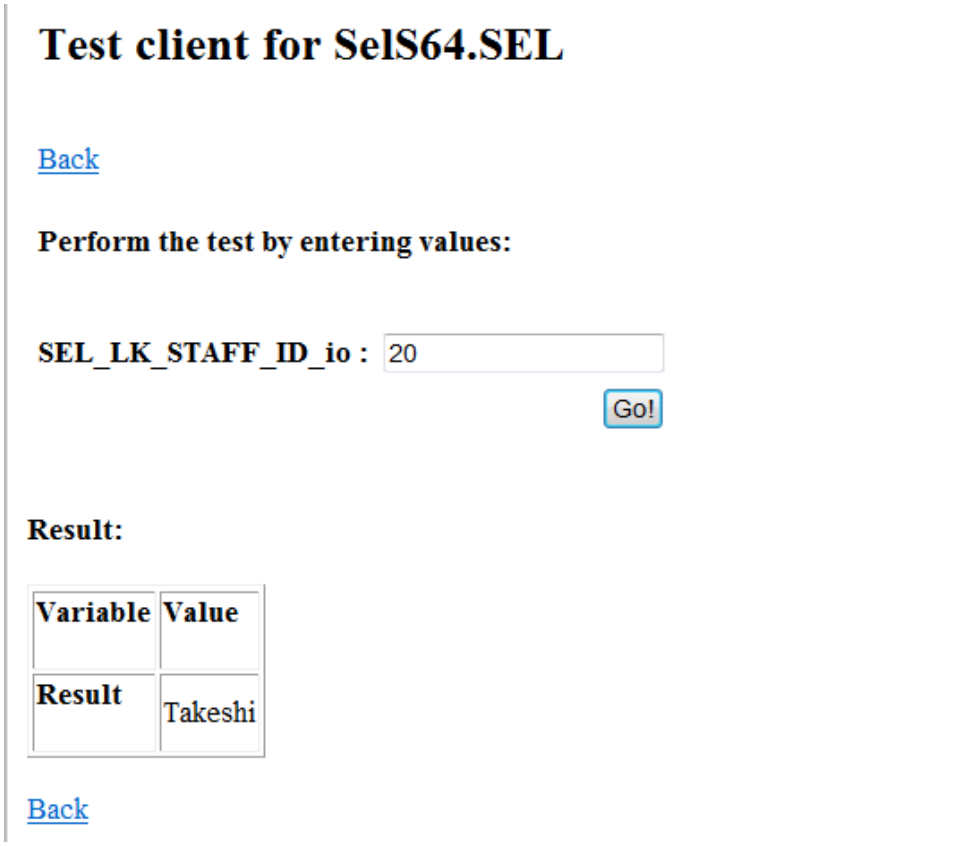


③ デバッグ実行

付録 1 と同じ要領で、Visual COBOL が Eclipse に作りこんだデバッガを使って、埋め込み SQL 文が入ったままの状態のソースでデバッグ実行できます：



処理を最後まで進めると、COBOL で Sybase より取得したデータが表示され、Java - COBOL – Sybase が正常に連携できていることを確認できます：



■ Solaris サーバ

33) Enterprise Server を停止

```
$ casstop -rESDEMO64
CASST0005I Shutdown of ES ESDEMO64 starting 15:35:06
CASSI8003I Enterprise Server "ESDEMO64" termination completed 15:35:06
Return code:    0
$
```

34) COBOL リモート開発用のデーモンを開始したセッションにて COBOL リモート開発用のデーモンを 停止

```
# $COBDIR/remotedev/stoprdodaemon
Process 19495 located:

     UID   PID  PPID   C      STIME TTY         TIME CMD

Do you wish to continue and kill it? (y/n): y
kill: 19495: no such process
Kill signal sent to process 19495
#
```

付録 3. Sybase 更新プログラムのディプロイと、EJB 経由の JCA 呼び出しにおけるコンテナ管理のトランザクション(32bit 編)

■ Solaris サーバ

1) 一般ユーザでログインし、Visual COBOL 及び Sybase 32 bit SDK の環境設定スクリプトを実行

2) Sybase の XA スイッチモジュールを作成

① Sybase 用のスイッチモジュールソースは Visual COBOL に同梱されていないため、以下のソースを持つプログラムを用意

```
$ cat ESSYBASEXA.cbl
       *>
       *> Micro Focus Server Express XA switch module for Sybase.
       *>
       *> (C) Copyright 2005-2014 Micro Focus (IP) Limited
       *> All Rights Reserved.
       *>

       IDENTIFICATION DIVISION.
       PROGRAM-ID. ESSYBASEXA.
       ENVIRONMENT DIVISION.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 wk-test          pic x(01).

       01 SybasePtr        USAGE PROCEDURE-POINTER.
       01 SybaseXaPtr      USAGE PROCEDURE-POINTER.
       01 SybaseXaPtr-pointer USAGE POINTER
                           REDEFINES SybaseXaPtr.
       01 MFocusCasPtr  USAGE PROCEDURE-POINTER.
       LINKAGE SECTION.
       01  XA-SWITCH     PIC X(128).

       PROCEDURE DIVISION.

          INITIALIZE wk-test.
          *> Load Enterprise Server module
          SET MFocusCasPtr TO ENTRY "casaxlib.so"

          *> Attempt to load a pointer to the Sybase XA switch struc
ture
          SET SybaseXaPtr TO ENTRY "sybase_TXS_xa_switch".
          SET ADDRESS OF XA-SWITCH TO SybaseXaPtr-pointer.

          GOBACK RETURNING SybaseXaPtr.
$
```

② Visual COBOL の動作モードを 32 bit に指定

```
$ COBMODE=32;export COBMODE
$ cobmode
Effective Default Working Mode: 32 bit ←
$
```

③ 用意したスイッチモジュールを Visual COBOL でビルド[12]

```
$ cob -z,sys,nounload ESSYBASEXA.cbl -to ESSYBASEXA.so -e "" -L$SYBA
SE/$SYBASE_OCS/lib -lsybcobct_r -lsybblk_r -lsybct_r -sybtcl_r -lsyb
cs_r -lsybcomn_r -lsybintl_r -lsybunic -lsybxadtm -lsocket -lnsl -ld
l -lpthread -lthread -lm
$
```

32 bit 用のスイッチモジュールが生成されます：

```
$ cobfile ESSYBASEXA.so
ESSYBASEXA.so:  ELF 32-bit MSB dynamic lib SPARC Version 1, dynamica
lly linked, not stripped, no debugging information available
$
```

3) XA 構成ファイルを SYBASE 32 bit SDK のディレクトリに準備

$SYBASE/$SYBASE_OCS/xa_config に下記のファイルを準備：

```
$ cat $SYBASE/$SYBASE_OCS/config/xa_config
; Comment line as first line of file REQUIRED!

[all]
        logfile="xalog32.txt"
        traceflags="all"
        properties="CS_LOGIN_TIMEOUT"="60"

[xa]
        lrm=connection32
        server=TOKPUTTER
$
```

[12] 付録 1 にて共有ライブラリを作成した際にリンクしたライブラリに加えて、$SYBASE/$SYBASE_OCS/libsybxadtm.so 及び libsybblk_r.so を追加でリンクしています。

4) DTM_TM_ROLE を持った SYBASE のユーザを作成

① ISQL を使って sa ユーザ SYBASE にログイン

```
$ isql -Usa -Psapass -STOKPUTTER
1>
```

② XA 接続で利用するユーザを追加

```
1> use master
2> go
1> sp_addlogin xauser, password
2> go
パスワードが変更されました。
アカウントはロックされませんでした。
新しいログインが作成されました。
(return status = 0)
1>
```

③ 追加したユーザが pubs2 データベースで利用できるように設定

```
1> use pubs2
2> go
1> sp_adduser xauser
2> go
新しいユーザが追加されました。
(return status = 0)
1> grant all to xauser
2> go
1>
```

④ 追加したユーザのデフォルト DB を pubs2 に指定

```
1> sp_modifylogin xauser, "defdb", pubs2
2> go
デフォルトデータベースは変更されました。
(return status = 0)
1>
```

⑤ 追加したユーザに検証で利用するテーブルを操作するための権限を付与

```
1> grant all on STAFF to xauser
2> go
1>
```

⑥ 追加したユーザに dtm_tm_role 権限を付与

```
1> sp_role "grant", dtm_tm_role, xauser
2> go
権限が更新されました。
(return status = 0)
1>
```

⑦ 「enable DTM」構成パラメータを有効化

```
1> sp_configure "enable DTM", 1
2> go
 Parameter Name                  Default     Memory Used
                        ----- 途中省略 -----

(1 row affected)
設定オプションが変更されました。このオプションは静的なので、変更を反映するために
Adaptive Server をリブートしてください。
'enable DTM' の値を変更しても、Adaptive Server
で使用するメモリの量は増加しません。
(return status = 0)
1>2>
```

5) SYBASE Adaptive Server をリブート

⑧ SYBASE Adaptive Server を停止

```
1> shutdown
2> go
サーバが要求により SHUTDOWN しました。
ASE はこのプロセスを終了しています。
CT-LIBRARY error:
        ct_results(): ネットワークパケットレイヤ: 内部 Net Library
エラー: Net-Library 操作は切断により終了しました。
$
```

⑨ SYBASE Adaptive Server を起動

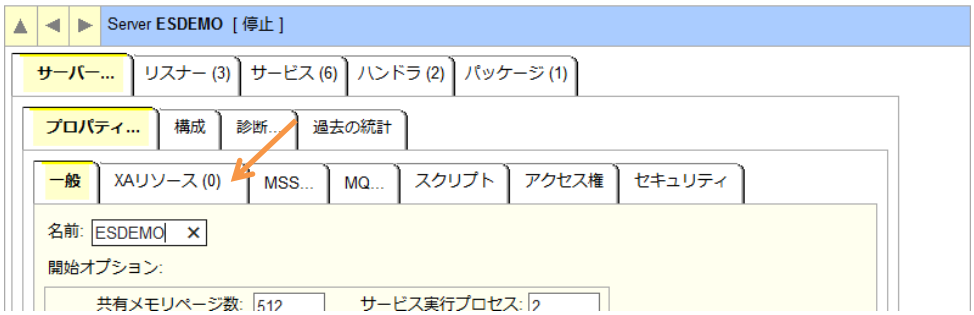
SYBASE Adaptive Server 起動スクリプトを実行

6) Enterprise Server にスイッチモジュールを追加

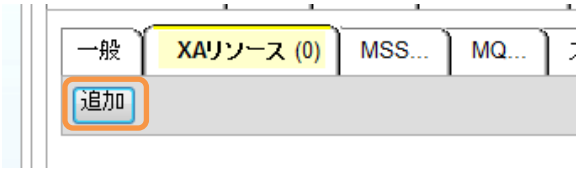
① Enterprise Administration 画面を開く

② ESDEMO の列で［編集］ボタンを押下

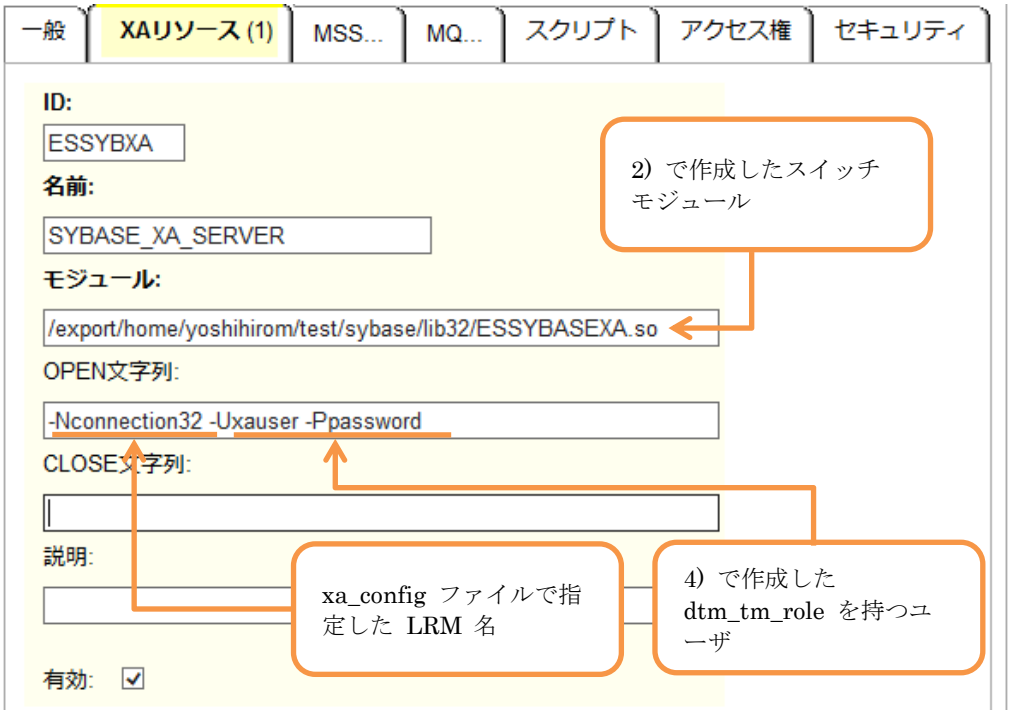
③ ［XA リソース］タブをクリック



④ [追加] ボタンを押下

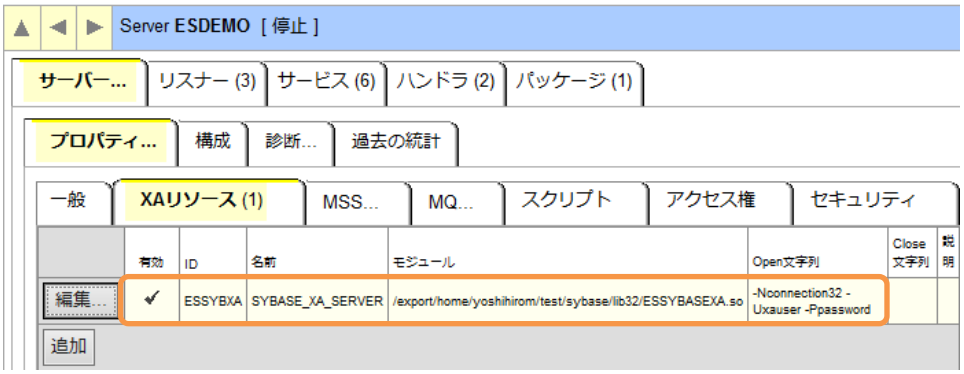


⑤ Open 文字列等 XA リソースに関する情報を指定[13]



⑥ [追加] ボタンを押下

XA リソースが追加されたことを確認できます：

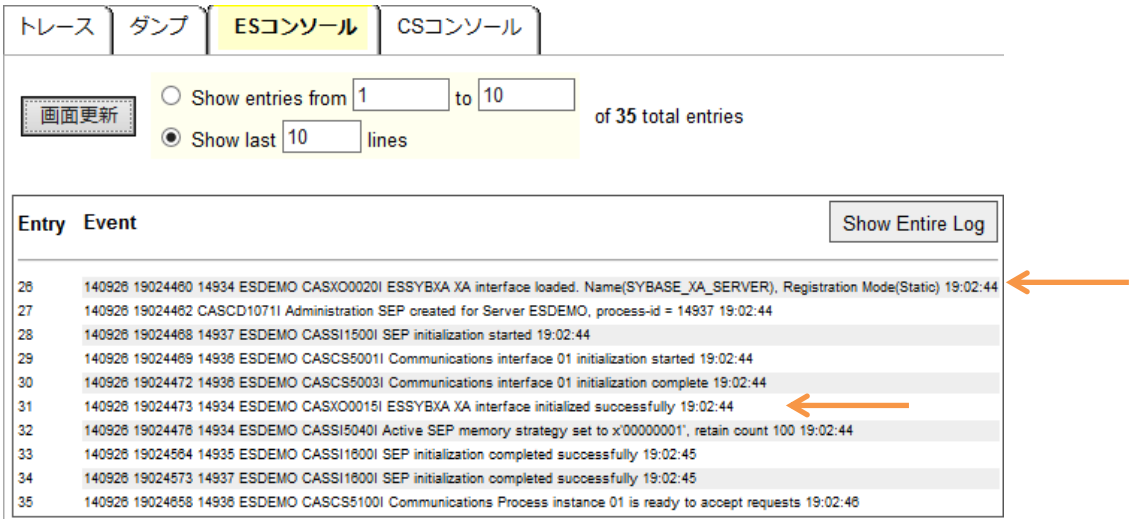


[13] Open 文字列の詳細は Sybase のマニュアルを参照願います。

7) Enterprise Server を起動

```
$ casstart -rESDEMO
....
CASCD0167I ES Daemon successfully auto-started 18:14:09
CASCD0050I ES "ESDEMO" initiation is starting 18:14:09
$
```

XA スイッチモジュールが正しく構成され、動作することを Enterprise Server のコンソールログより確認できます。下図は Enterprise Server Administration 画面にて、[編集] ボタン > [診断] タブ > [ES コンソール]
とナビゲートし確認したコンソール画面になります：



8) 付録 1 の要領で COBOL リモート開発用のデーモンを起動

■ Windows クライアントマシン

9) Visual COBOL for Eclipse を起動

10) 付録 1 で使用したリモートプロジェクトを開く

11) Sybase 上のデータに対して DML 文を発行する埋め込み SQL 文の入った COBOL プログラムをプロジェクトに追加[14]

⑩ 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] > [COBOL プログラム]
を選択

⑪ [新規ファイル名] 欄では [UPP.cbl] を指定

⑫ [完了] ボタンを押下

[14] 本検証で利用した実際のソースは本文書とともに公開しています。

⑬　ソースビューに以下プログラムをコーディング[15]

```
$SET p(cobsql) COBSQLTYPE=SYBASE END-C ENDP
 IDENTIFICATION DIVISION.
 PROGRAM-ID. UPP.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01 STAFF-ID PIC S9(9) COMP-5.
 01 STAFF-NAME PIC X(10).
     EXEC SQL END DECLARE SECTION END-EXEC.
     EXEC SQL INCLUDE SQLCA END-EXEC.
 01 TABLE-ITEM PIC X OCCURS 10 TIMES INDEXED BY IDX.
 LINKAGE SECTION.
  01 LK-STAFF-ID PIC S9(9) COMP-5.
  01 LK-STAFF-NAME PIC X(10).
  01 LK-Commit-Or-Rolback  PIC X.
 PROCEDURE DIVISION USING LK-STAFF-ID LK-STAFF-NAME
         LK-Commit-Or-Rolback.
 1.
     DISPLAY "Update TEST"  UPON CONSOLE.
     EXEC SQL SET CONNECTION "connection32" END-EXEC.
     DISPLAY "CONNECT SQLCODE:" SQLCODE  UPON CONSOLE.
     IF SQLCODE NOT = 0
        DISPLAY "MSG:" SQLERRMC UPON CONSOLE
        EXIT PROGRAM
     END-IF
     EXEC SQL USE pubs2 END-EXEC.
     DISPLAY "USE SQLCODE:" SQLCODE  UPON CONSOLE.
     IF SQLCODE NOT = 0
        DISPLAY "MSG:" SQLERRMC UPON CONSOLE
        EXIT PROGRAM
     END-IF
     MOVE LK-STAFF-ID TO STAFF-ID.
     MOVE LK-STAFF-NAME TO STAFF-NAME.
     EXEC SQL UPDATE STAFF SET NAME=:STAFF-NAME
              WHERE ID=:STAFF-ID
     END-EXEC.
      DISPLAY "UPDATE = " SQLCODE UPON CONSOLE.
     IF SQLCODE NOT = 0
        DISPLAY "MSG:" SQLERRMC UPON CONSOLE
        EXIT PROGRAM
     END-IF

     IF LK-Commit-Or-Rolback = 'R'
       SET IDX TO 11
       MOVE SPACE TO TABLE-ITEM(IDX)
     END-IF.
     EXIT PROGRAM.
```

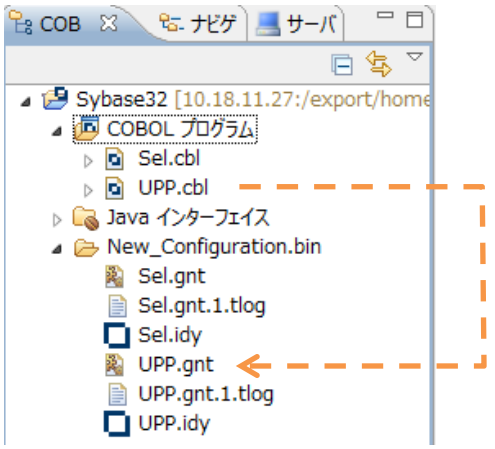
XA を介して Syabase にアクセスするため、付録１で指定した INITCALL(SYBINIT32) の指定は不要です。

プログラムから CONNECTION を確立するための処理は不要です。ここでは LRM を指定します。

[15] 本プログラムは第１パラメータで渡されたキー値に対するレコード中の NAME 列を第２パラメータで渡された値で更新します。第３パラメータで「R」が渡された場合は意図的に添え字範囲例外を発生させます。これにより上で処理した更新のトランザクションは ROLLBACK されます。このような例外が発生せず正常に処理できた場合は Enterprise Server は COMMIT を発行し、トランザクションを確定させます。

⑭ Ctrl + S を打鍵し、ソースを保存

自動ビルドが有効なため、ビルド処理が走ります。UPP.cbl をコンパイルし、動的ロードモジュールが生成されたことを COBOL エクスプローラより確認できます：



12) アプリケーションの COBOL – Java 変換マッピングを作成

① COBOL エクスプローラにてプロジェクトを右クリックし

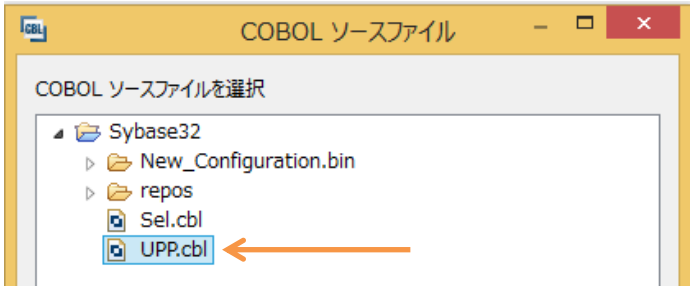
[新規] > [その他]

を選択

② [Micro Focus IMTK] > [Java インターフェイス] へとナビゲートし [次へ] ボタンを押下

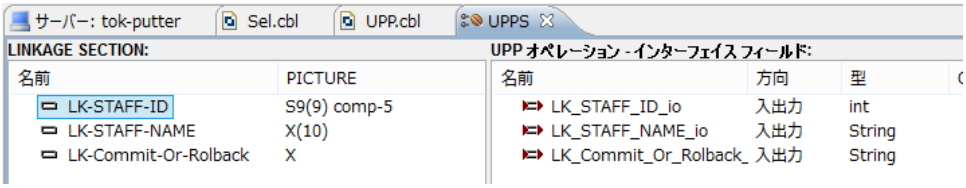
③ Java インターフェイス名「UPPS」を指定し、[参照] ボタンを押下

④ 追加したプログラムを選択



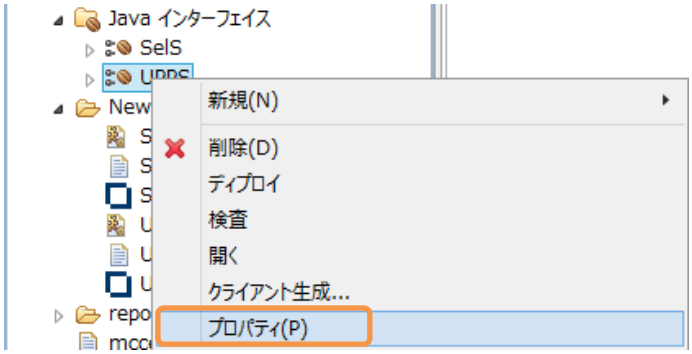
⑤ [完了] ボタンを押下

ここでは、デフォルトのマッピングをそのまま利用します：



13) Enterprise Server へのディプロイ情報を指定

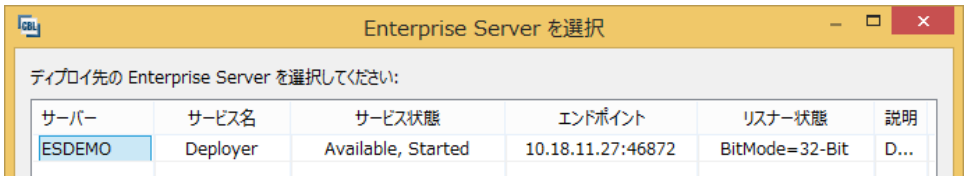
① COBOL エクスプローラにて追加した Java インターフェイスを右クリックから［プロパティ］を選択



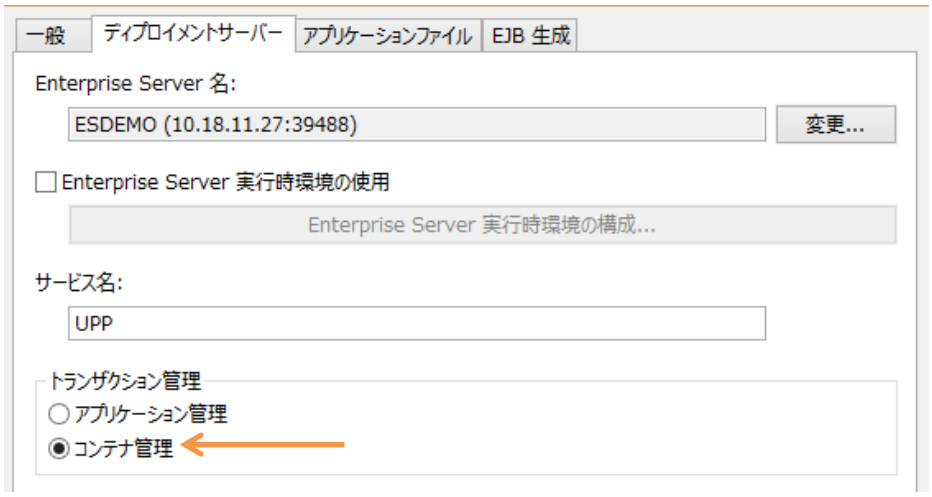
② ［ディプロイメントサーバー］タブを選択

③ ［変更］ボタンを押下

④ Solaris サーバの ESDEMO を選択し［OK］ボタンを押下



⑤ ［トランザクション管理］欄にて［コンテナ管理］を選択



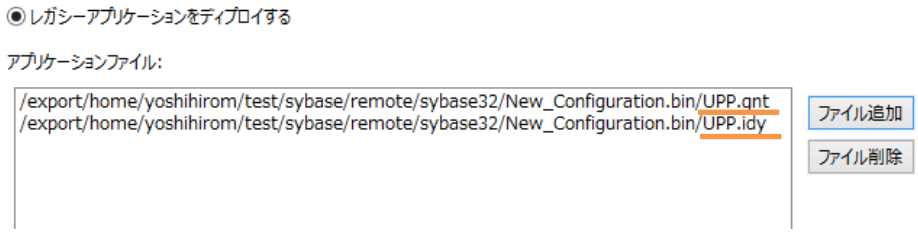
⑥ ［アプリケーションファイル］タブを選択

⑦ ［レガシーアプリケーションをディプロイする］を選択

⑧ ［ファイル追加］ボタンを押下

⑨ プロジェクトディレクトリ配下の New_Configuration.bin ディレクトリに生成された UPP.gnt 及び UPP.idy を選択し [OK] ボタンを押下

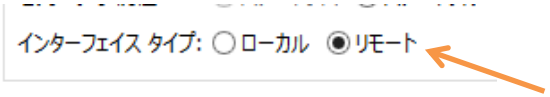
追加後の画面：



⑩ [EJB 生成] タブを選択

⑪ [アプリケーションサーバー] 欄にて JEE 6、WebLogic 12.1.1 を指定

⑫ [インターフェイスタイプ] 欄にて [リモート] を指定



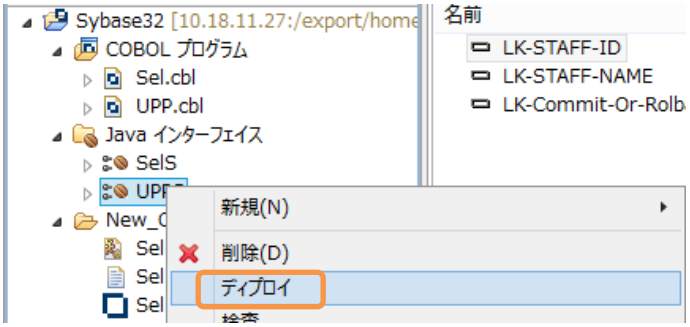
⑬ [Java コンパイラ] 欄にて利用する javac が格納されたディレクトリを指定

⑭ [J2EE クラスパス] 欄にて [参照] ボタンを押下し WebLogic のインストールディレクトリに格納されている weblogic.jar を選択

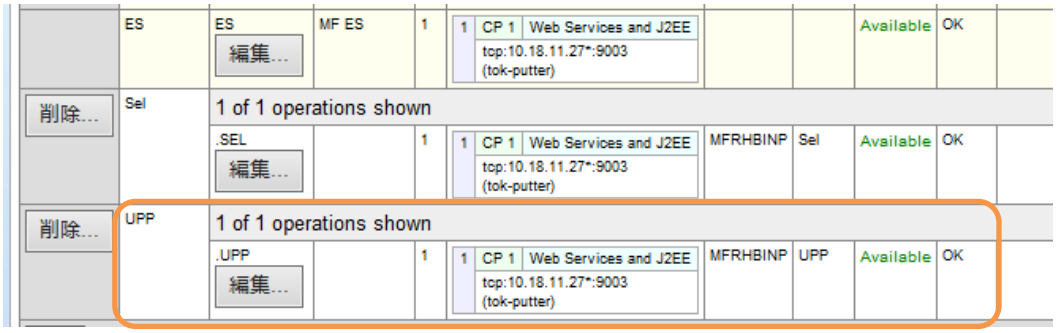
⑮ [OK] ボタンを押下し、設定画面を閉じる

14) 作成した Java サービスを Enterprise Server へディプロイ

COBOL エクスプローラにて用意した Java インターフェイスを右クリックし、[ディプロイ] を選択

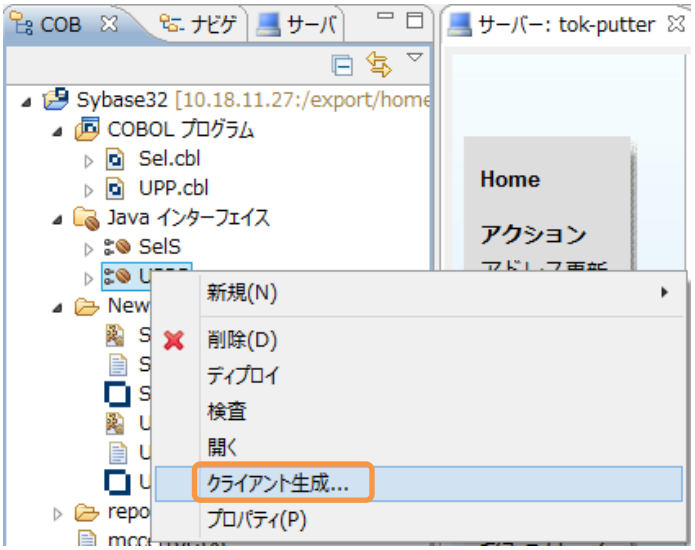


正常にディプロイできたことを Enterprise Server Administration 画面にて確認することができます：

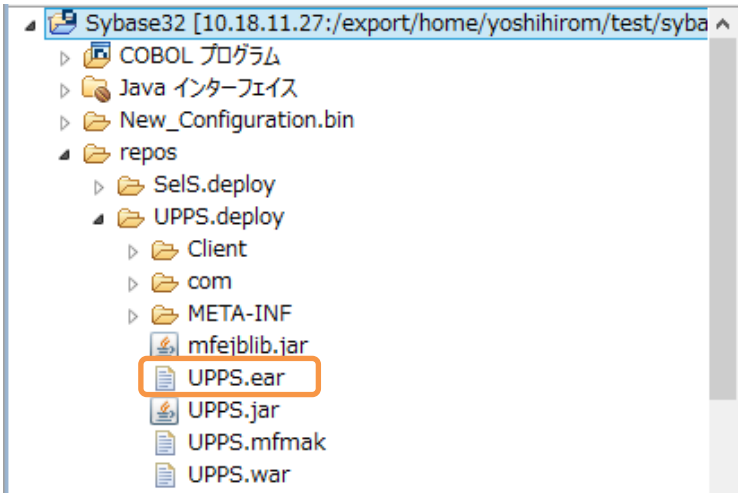


15) ディプロイしたサービスをテスト呼び出しするスタブクライアントアプリケーションを作成

COBOL エクスプローラにて Java インターフェイスを右クリックして［クライアント生成］を選択



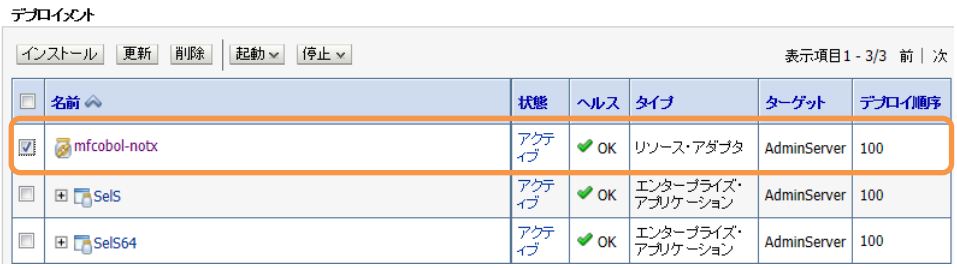
COBOL エクスプローラにて .ear にアーカイブされたアプリケーションが生成されていることを確認できます：



■ Solaris サーバ

16) 付録 1 で WebLogic にディプロイしたリソースアダプタを削除

⑮ WebLogic Administration Console 画面を開き、ディプロイ済みのリソースアダプタを選択



⑯ [削除] ボタンを押下

17) Visual COBOL に付属される XA トランザクションをサポートする WebLogic 12c 向けのリソースアダプタを WebLogic へ追加

① WebLogic が提供するディプロイメントツール weblogic.Deployer を利用するための環境変数を設定

```
$ WL_USER=weblogic;export WL_USER
$ WL_PASSWD=P@sswOrd;export WL_PASSWD
$ NAME=mfcobol-notx;export NAME
$ FULL_PATH_TO_THE_RAR_FILE=$COBDIR/javaee/javaee6/oracleweblogic121
1/mfcobol-xa.rar;export FULL_PATH_TO_THE_RAR_FILE
$
```

ディプロイする
リソースアダプタ

② weblogic.Deployer を使ってリソースアダプタを WebLogic へディプロイ

```
$ java -classpath $WL_HOME/server/lib/weblogic.jar weblogic.Deployer
 -username $WL_USER -password $WL_PASSWD -name $NAME -deploy $FULL_P
ATH_TO_THE_RAR_FILE
weblogic.Deployer がオプション -username weblogic -name mfcobol-xa -d
eploy /opt/mf/ED22U1HF4/javaee/javaee6/oracleweblogic1211/mfcobol-x
a.rar を指定して呼び出されました
<2014/09/26 21 時 25 分 09 秒 JST> <Info> <J2EE Deployment SPI> <BEA-26
0121> <Initiating deploy operation for application, mfcobol-xa [arch
ive: /opt/mf/ED22U1HF4/javaee/javaee6/oracleweblogic1211/mfcobol-xa.
rar], to configured targets.>
タスク 9 が開始されました: [Deployer:149026]デプロイ application mfcob
ol-xa on AdminServer.
タスク 9 完了: [Deployer:149026]デプロイ application mfcobol-xa on Ad
minServer.
ターゲットの状態: サーバー AdminServer で deploy 完了

$
```

正常にディプロイできたことを WebLogic Server Administration Console より確認できます：

18) 15) で生成したスタブクライアントを WebLogic へインストール

① WebLogic Server Administration Console へログイン

② 左ペインより対象のドメイン配下の［デプロイメント］をクリック

③ ［インストール］ボタンを押下

④ ［パス］欄に生成された .ear が格納されたディレクトリを指定し Enter を打鍵

⑤ 生成された UPPS.ear を選択し、［次へ］ボタンを押下

⑥ ［このデプロイメントをアプリケーションとしてインストールする］が選択されていることを確認して、［次へ］ボタンを押下

⑦ ［終了］ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます：

■ Windows クライアントマシン

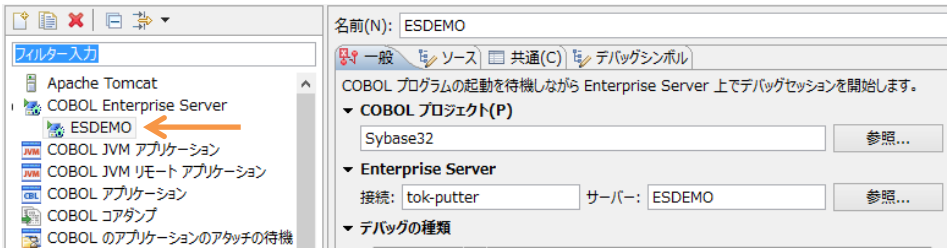
19) Enterprise Server が動的デバッグ有効で起動されていることを確認

Enterprise Server Administration 画面を開き、ESDEMO の行における［編集］ボタンを押下し、確認

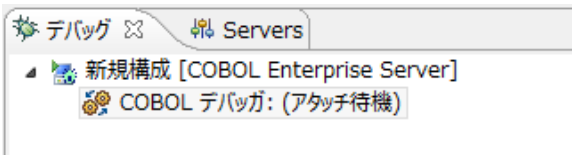
20) Enterprise Server デバッグを起動

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 付録 1 で作成したデバッグ構成をダブルクリック

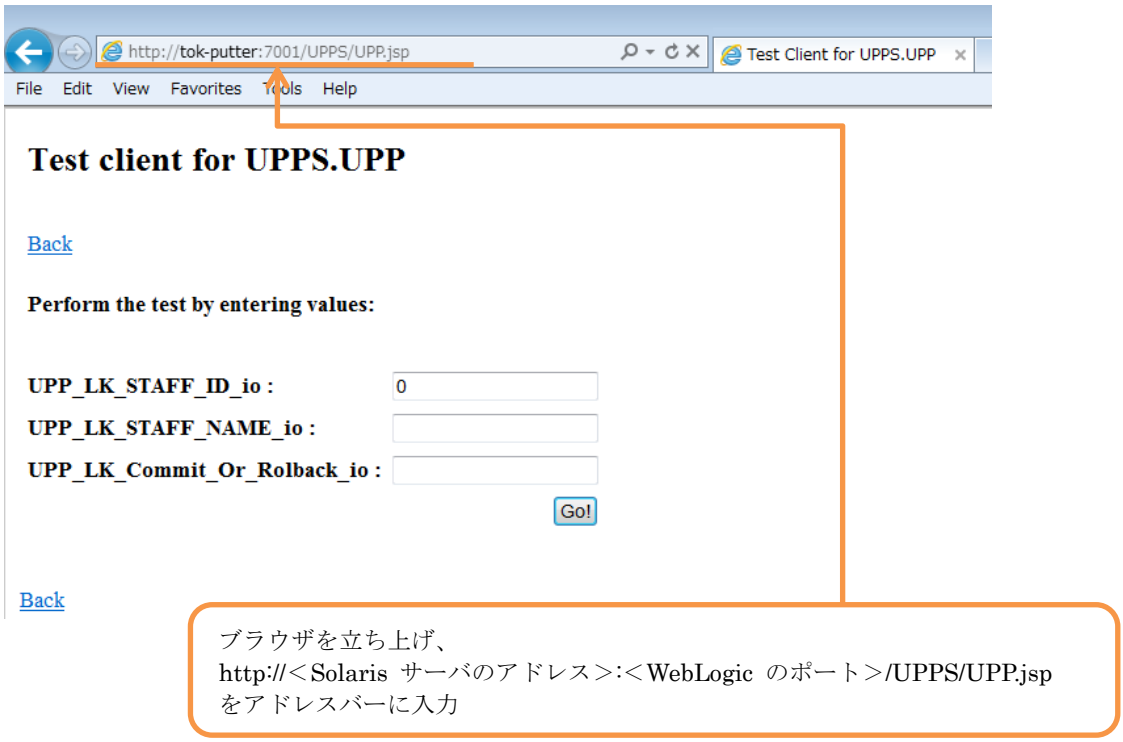


③ [パースペクティブの切り替えの確認] 画面にて［はい］を選択
デバッグパースペクティブにてアタッチ待機状態になっていることを確認できます：

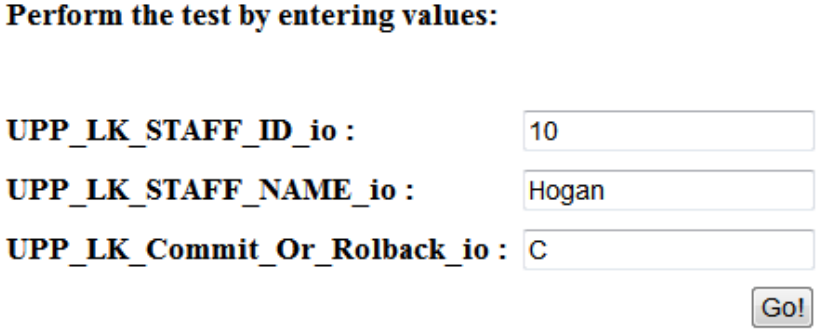


21) ディプロイしたアプリケーションをデバッグ実行

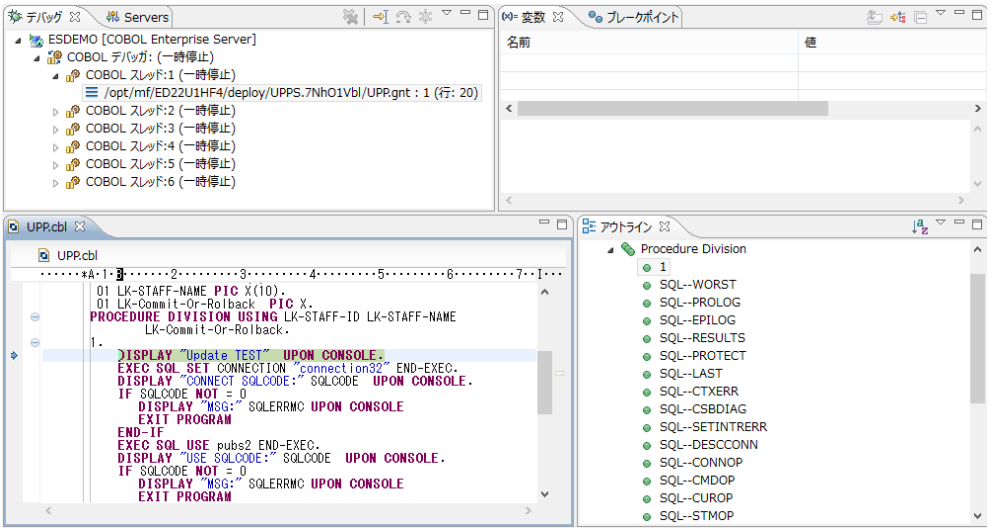
① WebLogic にディプロイしたスタブクライアントアプリケーションを起動



② [UPP_LK_STAFF_ID_io] 欄に [**10**] を
[UPP_LK_STAFF_NAME_io] 欄に [**Hogan**] を
[UPP_LK_Commit_Or_Rollback_io] 欄に [**C**] を
入力し [Go!] ボタンを押下



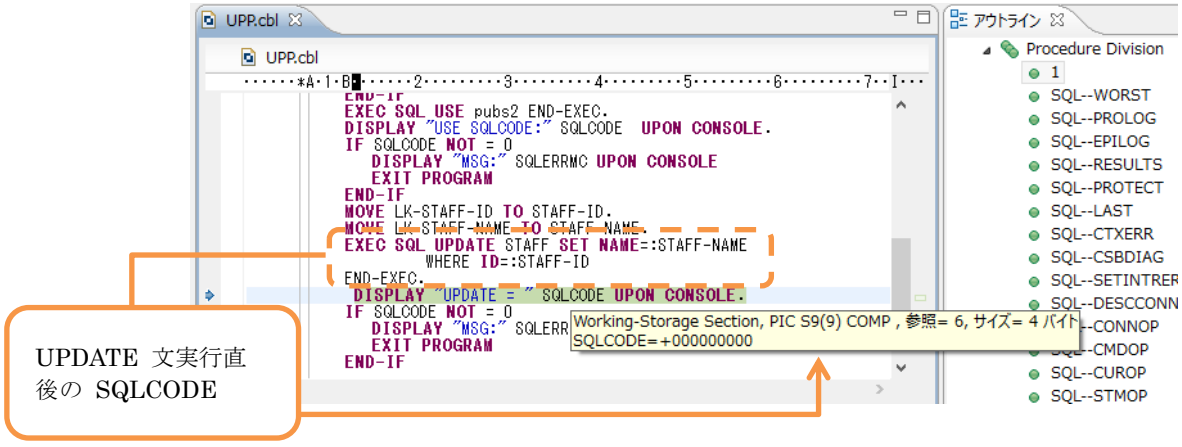
処理が Enterprise Server に渡り、Eclipse のデバッガがその処理を引き込みます。Enterprise Server にディプロイした COBOL プログラムの最初の行を実行する前で処理が停止しているのが確認できます：



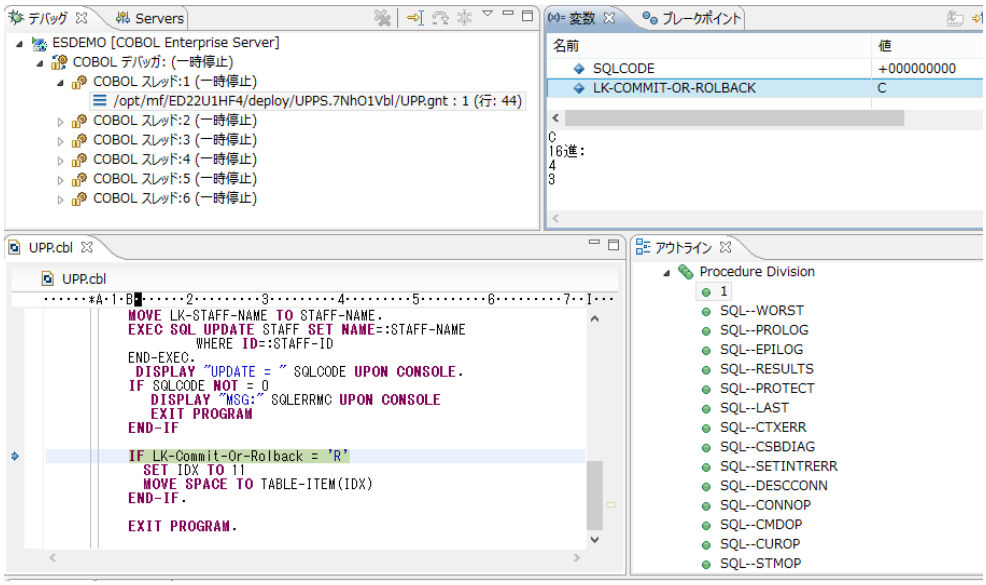
③ デバッグ実行

これまでの検証と同様に F5 キー打鍵でプリコンパイル前の埋め込み SQL 文が入った状態のソースを 1 ステップずつ処理を進めることができます。

本プログラムはトランザクションマネージャが確立した接続を利用するため、プログラムから CONNECT 文は発行していませんが、下図のように正常に SQL 文を実行できます：



デバッグ実行イメージ：



COBOL 側の処理を最後まで進めると、Java 側に処理が戻り Web の画面が切り替わります：

Test client for UPPS.UPP

Back

Perform the test by entering values:

UPP_LK_STAFF_ID_io : 10

UPP_LK_STAFF_NAME_io : Hogan

UPP_LK_Commit_Or_Rolback_io : C

Go!

Result:

Variable	Value
LK_STAFF_ID_io	10
LK_STAFF_NAME_io	Hogan
LK_Commit_Or_Rolback_io	C

Back

■ Solaris サーバ

22) アプリケーションがレコードを正しく更新していることを確認

```
$ isql -Usa -Psapass -STOKPUTTER
1> use pubs2
2> go
1> select * from STAFF
2> go
 ID          NAME       DEPT        JOB         YEARS   SALARY
        COMM
 ----------- ---------- ----------- ---------- ------- -------------
        -------------
          10 Hogan              333 Mgr               7      18357.50
                0.00
          20 Takeshi            333 Sales             8      18171.25
              612.45
          30 Marenhi             38 Mgr               5      17506.75
                0.00

(3 rows affected)
1>
```

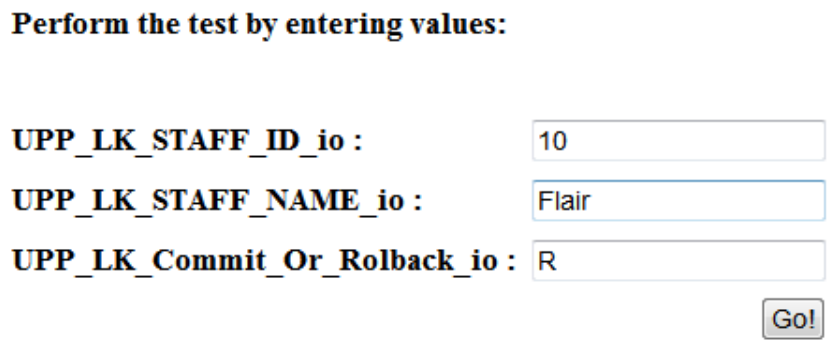
付録 1 の検証にて照会した際は「Smith」が格納されていましたが、「Hogan」に更新されています。

■ Windows クライアントマシン

23) トランザクションが取り消される条件でパラメータを渡し、ディプロイしたアプリケーションをデバッグ実行

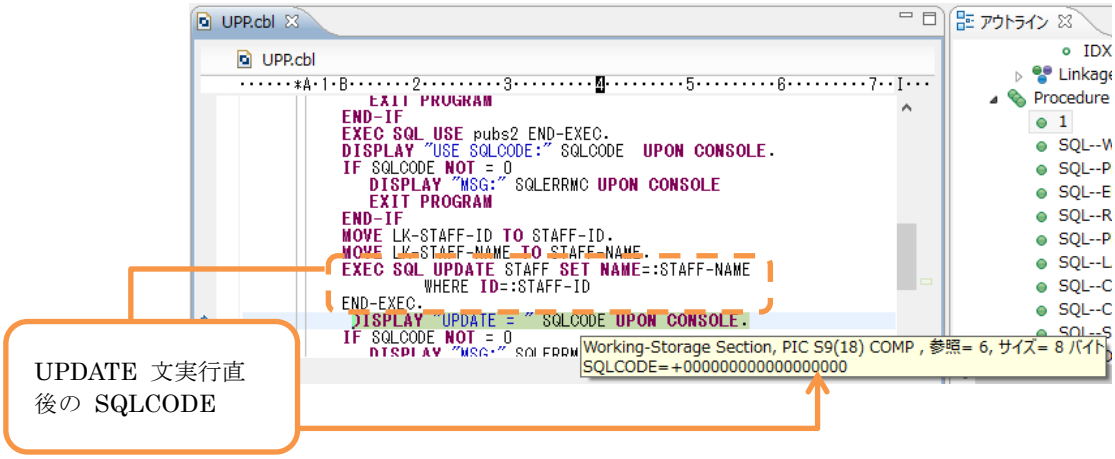
① WebLogic にディプロイしたスタブクライアントアプリケーションを起動

② [UPP_LK_STAFF_ID_io] 欄に [**10**] を
[UPP_LK_STAFF_NAME_io] 欄に [**Flair**] を
[UPP_LK_Commit_Or_Rollback_io] 欄に [**R**] を
入力し [Go!] ボタンを押下



③ デバッグ実行

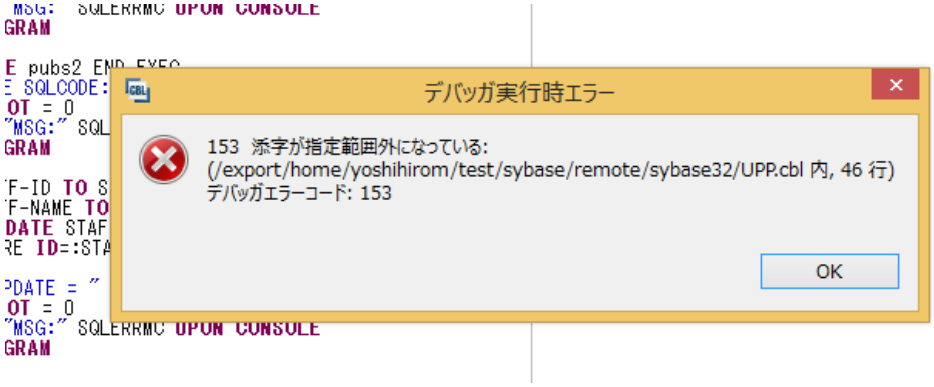
Update 文実行後の SQLCODE を見ると 0 が返ってきており、Update 文は Sybase 上で正常に実行できたことがわかります：



ステップをそのまま進めていきますと第３パラメータに「R」を指定したため、10 回の繰り返し項目の添え字に 11 を格納するロジックへ入ります：

```
    EXEC SQL INCLUDE SQLCA END-EXEC.
01 TABLE-ITEM PIC X OCCURS 10 TIMES INDEXED BY IDX.
LINKAGE SECTION.
PROCEDURE DIVISION USING LK-STAFF-ID LK-STAFF-NAME
        LK-Commit-Or-Rolback.
1.
    DISPLAY "Update TEST"  UPON CONSOLE.
    EXEC SQL SET CONNECTION "connection64" END-EXEC.
    DISPLAY "CONNECT SQLCODE:" SQLCODE  UPON CONSOLE.
    IF SQLCODE NOT = 0
       DISPLAY "MSG:" SQLERRMC UPON CONSOLE
       EXIT PROGRAM
    END-IF
    EXEC SQL USE pubs2 END-EXEC.
    DISPLAY "USE SQLCODE:" SQLCODE  UPON CONSOLE.
    IF SQLCODE NOT = 0
       DISPLAY "MSG:" SQLERRMC UPON CONSOLE
       EXIT PROGRAM
    END-IF
    MOVE LK-STAFF-ID TO STAFF-ID.
    MOVE LK-STAFF-NAME TO STAFF-NAME.
    EXEC SQL UPDATE STAFF SET NAME=:STAFF-NAME
            WHERE ID=:STAFF-ID
    END-EXEC.
     DISPLAY "UPDATE = " SQLCODE UPON CONSOLE.
    IF SQLCODE NOT = 0
       DISPLAY "MSG:" SQLERRMC UPON CONSOLE
       EXIT PROGRAM
    END-IF

    IF LK-Commit-Or-Rolback = 'R'
      SET IDX TO 11
      MOVE SPACE TO TABLE-ITEM(IDX)
    END-IF.
```

繰り返しの範囲を超えた添え字を指定して MOVE 文を実行すると実行時エラーが発生します：



デバッガ側で処理を止めずに処理を進める場合、 Java 側へも COBOL の処理で例外が発生したことが伝播されブラウザにもエラーが発生した旨が表示されます：

Error 500--Internal Server Error

```
javax.ejb.EJBException: UPP threw a resource Exception
        at com.mypackage.UPPS.UPPSBean.UPP(Unknown Source)
        at com.mypackage.UPPS.UPPSEJB_imgfn_UPPSImpl.__WL_invoke(Unknown Source)
        at weblogic.ejb.container.internal.SessionRemoteMethodInvoker.invoke(SessionRemoteMethodInvoker.java:34)
        at com.mypackage.UPPS.UPPSEJB_imgfn_UPPSImpl.UPP(Unknown Source)
        at com.mypackage.UPPS.UPPSEJB_imgfn_UPPSImpl_CBV.UPP(Unknown Source)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at weblogic.ejb.container.internal.RemoteBusinessIntfProxy.invoke(RemoteBusinessIntfProxy.java:84)
        at com.sun.proxy.$Proxy86.UPP(Unknown Source)
        at com.mypackage.UPPS.UPPSServlet.performTask(Unknown Source)
        at com.mypackage.UPPS.UPPSServlet.doPost(Unknown Source)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:751)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:844)
        at weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:280)
        at weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:254)
        at weblogic.servlet.internal.StubSecurityHelper.invokeServlet(StubSecurityHelper.java:136)
        at weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:346)
        at weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:243)
        at weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.wrapRun(WebAppServletContext.java:3432)
        at weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:3402)
        at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:321)
        at weblogic.security.service.SecurityManager.runAs(SecurityManager.java:120)
        at weblogic.servlet.provider.WlsSubjectHandle.run(WlsSubjectHandle.java:57)
        at weblogic.servlet.internal.WebAppServletContext.doSecuredExecute(WebAppServletContext.java:2285)
        at weblogic.servlet.internal.WebAppServletContext.securedExecute(WebAppServletContext.java:2201)
        at weblogic.servlet.internal.WebAppServletContext.execute(WebAppServletContext.java:2179)
        at weblogic.servlet.internal.ServletRequestImpl.run(ServletRequestImpl.java:1572)
        at weblogic.servlet.provider.ContainerSupportProviderImpl$WlsRequestExecutor.run(ContainerSupportProviderImpl.java:255)
        at weblogic.work.ExecuteThread.execute(ExecuteThread.java:311)
        at weblogic.work.ExecuteThread.run(ExecuteThread.java:263)
Caused by: javax.resource.spi.EISSystemException: CobolException Recoverable:
目的コード エラー: ファイル '/opt/mf/ED22U1HF4/deploy/UPPS.7NhO1Vbl/UPP.gnt'
エラーコード: 153, pc=0, call=1, seg=0
153  添字が指定範囲外になっている (/export/home/yoshihirom/test/sybase/remote/sybase32/UPP.cbl 内, 46 行) executing UPP.UPP
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:291)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:
```

■ Solaris サーバ

24) 「Flair」へ更新する UPDATE 文のトランザクションが取り消されていることを確認

```
$ isql -Usa -Psapass -STOKPUTTER
1> use pubs2
2> go
1> select * from STAFF
2> go
 ID          NAME       DEPT        JOB        YEARS   SALARY
        COMM
 ----------- ---------- ----------- ---------- ------- -------------
        ------------
          10 Hogan              333 Mgr              7       18357.50
                0.00
          20 Takeshi            333 Sales            8       18171.25
              612.45
          30 Marenhi             38 Mgr              5       17506.75
                0.00

(3 rows a
1>
```

22) で確認した「Hogan」がそのまま残っています。

25) Enterprise Server を停止

26) COBOL リモート開発用のデーモンを開始したセッションにて COBOL リモート開発用のデーモンを 停止

付録 4. Sybase 更新プログラムのディプロイと、EJB 経由の JCA 呼び出しにおけるコンテナ管理のトランザクション(64bit 編)

■ Solaris サーバ

1) 一般ユーザでログインし、Visual COBOL 及び Sybase 64 bit SDK の環境設定スクリプトを実行

2) Sybase の XA スイッチモジュールを作成

① 付録３で用意したスイッチモジュール用のソース ESSYBASEXA.cbl を作業ディレクトリへコピー

② Visual COBOL の動作モードを 64 bit に指定

```
$ COBMODE=64;export COBMODE
$ cobmode
Effective Default Working Mode: 64 bit
$
```

③ ①でコピーしたスイッチモジュールを Visual COBOL でビルド[16]

```
$ cob -z,sys,nounload ESSYBASEXA.cbl -to ESSYBASEXA.so -e "" -L$SYBA
SE/$SYBASE_OCS/lib -lsybcobct_r64 -lsybblk_r64 -lsybct_r64 -lsybtcl_
r64 -lsybcs_r64 -lsybcomn_r64 -lsybintl_r64 -lsybunic64 -lsybxadtm64
 -lsocket -lnsl -ldl -lpthread -lthread -lm
$
```

64 bit 用のスイッチモジュールが生成されます：

```
$ cobfile ESSYBASEXA.so
ESSYBASEXA.so:  ELF 64-bit MSB dynamic lib SPARCV9 Version 1, dynami
cally linked, not stripped, no debugging information available
$
```

[16] 付録２にて共有ライブラリを作成した際にリンクしたライブラリに加えて、$SYBASE/$SYBASE_OCS/libsybxadtm64.so 及び libsybblk_r64.so を追加でリンクしています。

3) XA 構成ファイルを SYBASE 64 bit SDK のディレクトリに準備

$SYBASE/$SYBASE_OCS/xa_config に下記のファイルを準備：

```
$ cat $SYBASE/$SYBASE_OCS/config/xa_config
; Comment line as first line of file REQUIRED!

[all]
        logfile="xalog64.txt"
        traceflags="all"
        properties="CS_LOGIN_TIMEOUT"="60"

[xa]
        lrm=connection64
        server=TOKPUTTER
$
```

4) Enterprise Server にスイッチモジュールを追加

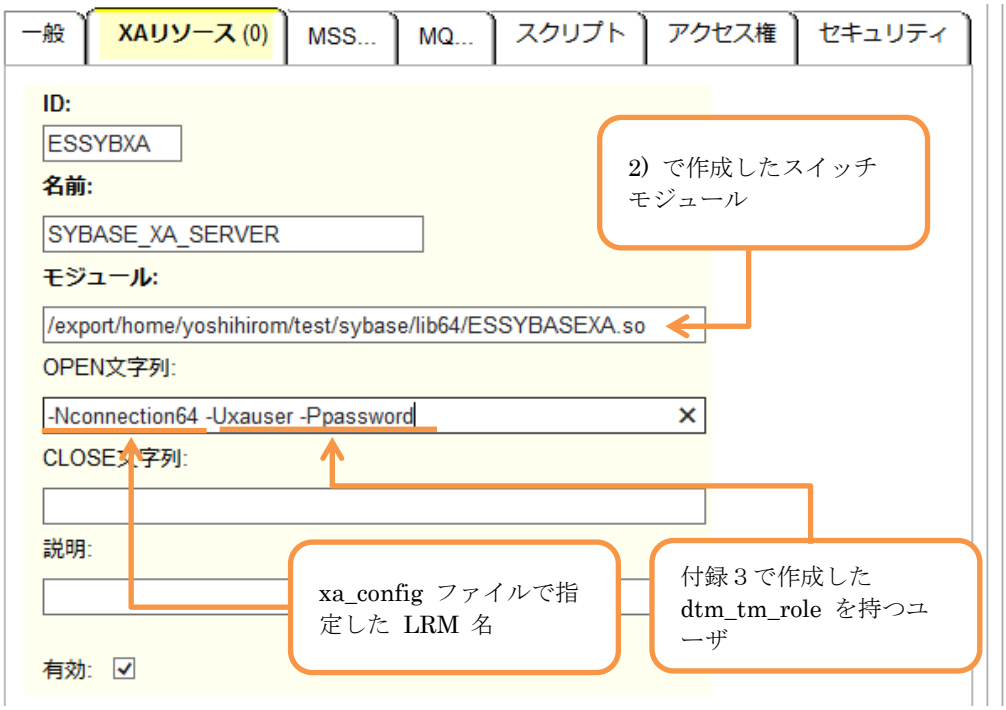
① Enterprise Administration 画面を開く

② ESDEMO64 の列で［編集］ボタンを押下

③ ［XA リソース］タブをクリック

④ ［追加］ボタンを押下

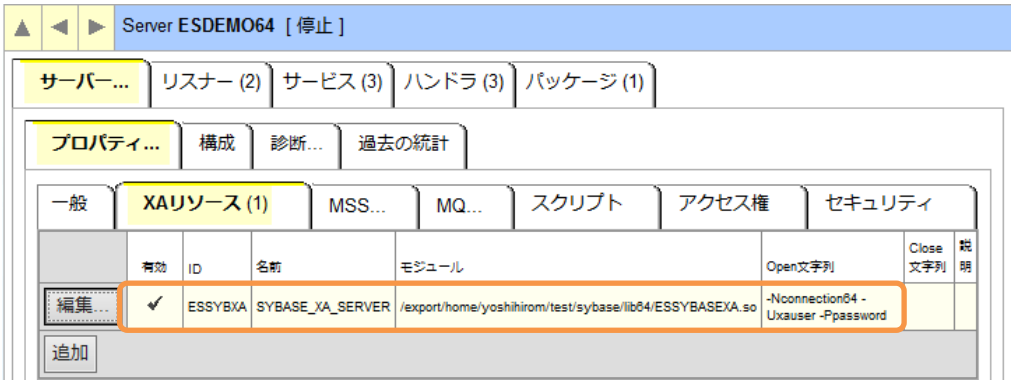
⑤ Open 文字列等 XA リソースに関する情報を指定[17]



[17] Open 文字列の詳細は Sybase のマニュアルを参照願います。

⑥ [追加] ボタンを押下

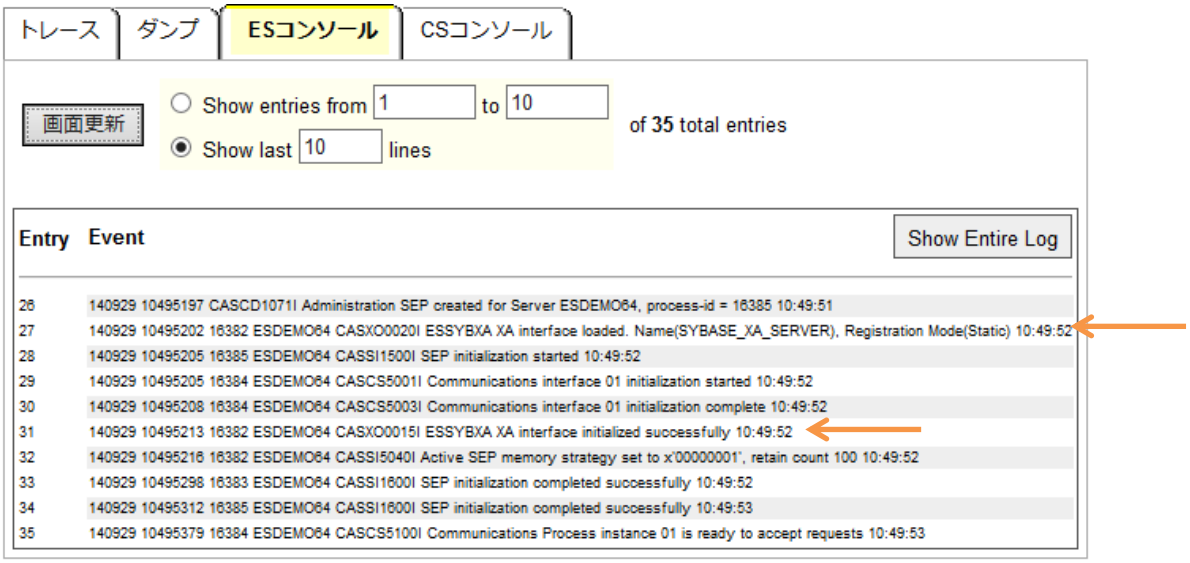
XA リソースが追加されたことを確認できます：



5) Enterprise Server を起動

```
$ casstart -rESDEMO64
....
CASCD0167I ES Daemon successfully auto-started 10:49:51
CASCD0050I ES "ESDEMO64" initiation is starting 10:49:51
$
```

XA スイッチモジュールが正しく構成され、動作することを Enterprise Server のコンソールログより確認できます。下図は Enterprise Server Administration 画面にて、[編集] ボタン > [診断] タブ > [ES コンソール]
とナビゲートし確認したコンソール画面になります：



6) 付録 2 の要領で COBOL リモート開発用のデーモンを起動

■ Windows クライアントマシン

7) Visual COBOL for Eclipse を起動

8) 付録２で使用したリモートプロジェクトを開く

9) Sybase 上のデータを更新する埋め込み SQL 文の入った COBOL プログラムをプロジェクトに追加（付録３の検証と同じプログラムを利用します。）

① COBOL エクスプローラ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]
を選択

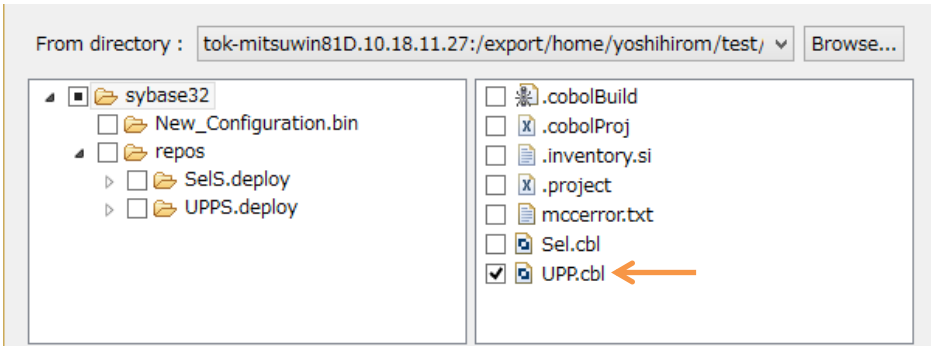
② [Remote Systems] > [Remote file system] へとナビゲートし、[次へ] ボタンを押下

③ [Browse] ボタンを押下

④ [Connection] 欄で Solaris サーバのアドレスを選択

⑤ 付録１で使用した COBOL リモートプロジェクトのディレクトリへナビゲートし [OK] ボタンを押下

⑥ 付録３で使用した UPP.cbl のみにチェックを入れ、[完了] ボタンを押下



10) プログラムが SET CONNECTION 文を発行する先の Logical Resource Manager 名を 3) で構成した名前に変更
UPP.cbl のソースを下記のように変更し保存：

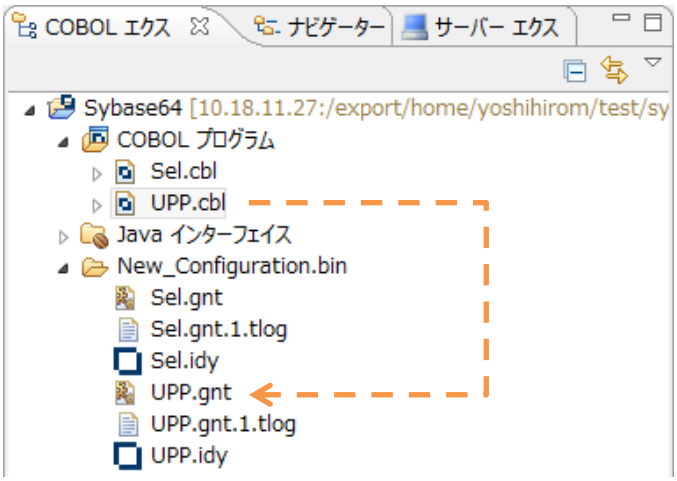
[編集前]

```
           :
       EXEC SQL SET CONNECTION "connection32" END-EXEC.
           :
```

[編集後]

```
           :
       EXEC SQL SET CONNECTION "connection64" END-EXEC.
           :
```

保存をすると、自動ビルドが有効なため、下図のように動的ロードモジュールが生成されます：



11) アプリケーションの COBOL – Java 変換マッピングを作成

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

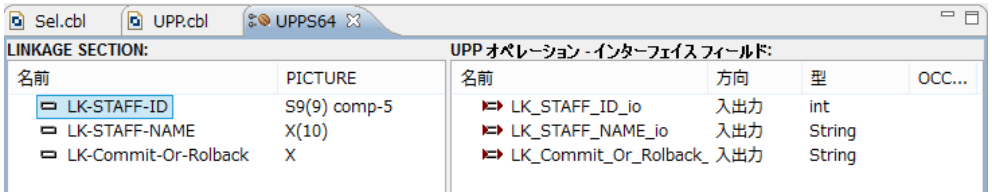
② [Micro Focus IMTK] > [Java インターフェイス] へとナビゲートし [次へ] ボタンを押下

③ Java インターフェイス名「UPPS64」を指定し、[参照] ボタンを押下

④ 追加したプログラムを選択

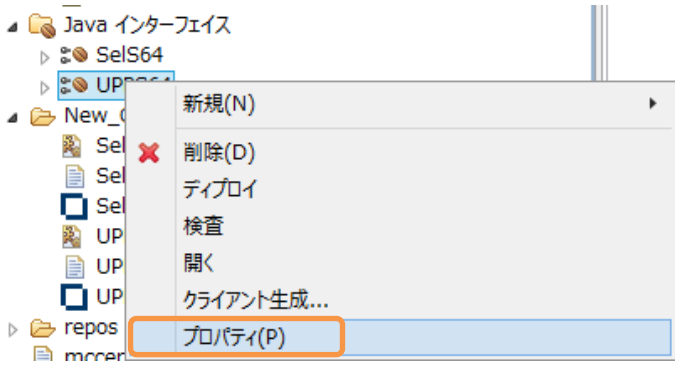
⑤ [完了] ボタンを押下

ここでは、デフォルトのマッピングをそのまま利用します：



12) Enterprise Server へのディプロイ情報を指定

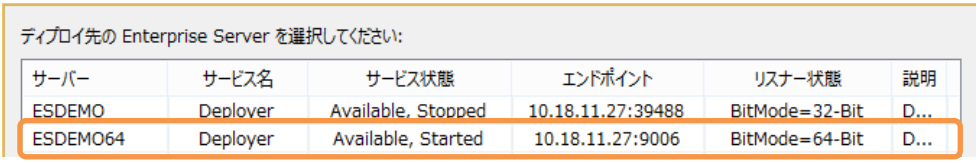
① COBOL エクスプローラにて追加した Java インターフェイスを右クリックから［プロパティ］を選択



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

④ Solaris サーバの ESDEMO64 を選択し［OK］ボタンを押下



⑤ ［トランザクション管理］欄にて［コンテナ管理］を選択

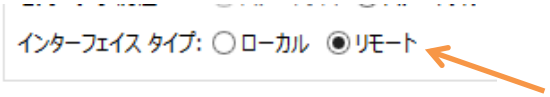
⑥ ［アプリケーションファイル］タブを選択

⑦ ［レガシーアプリケーションをディプロイする］を選択

⑧ ［ファイル追加］ボタンを押下

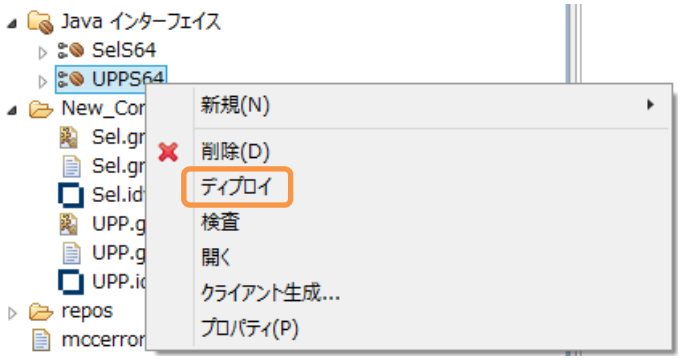
⑨ プロジェクトディレクトリ配下の New_Configuration.bin ディレクトリに生成された UPP.gnt 及び UPP.idy を選択し［OK］ボタンを押下

⑩ [EJB 生成] タブを選択
⑪ [アプリケーションサーバー] 欄にて JEE 6、WebLogic 12.1.1 を指定
⑫ [インターフェイスタイプ] 欄にて [リモート] を指定

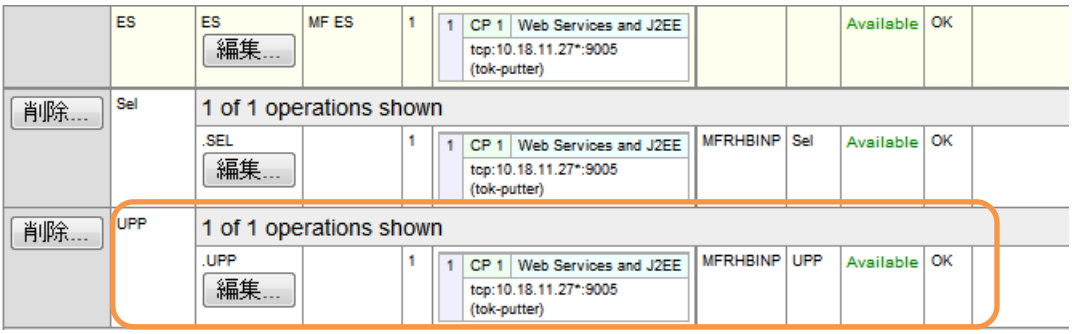


⑬ [Java コンパイラ] 欄にて利用する javac が格納されたディレクトリを指定
⑭ [J2EE クラスパス] 欄にて [参照] ボタンを押下し WebLogic のインストールディレクトリに格納されている weblogic.jar を選択
⑮ [OK] ボタンを押下し、設定画面を閉じる

13) 作成した Java サービスを Enterprise Server へディプロイ
COBOL エクスプローラにて用意した Java インターフェイスを右クリックし、[ディプロイ] を選択

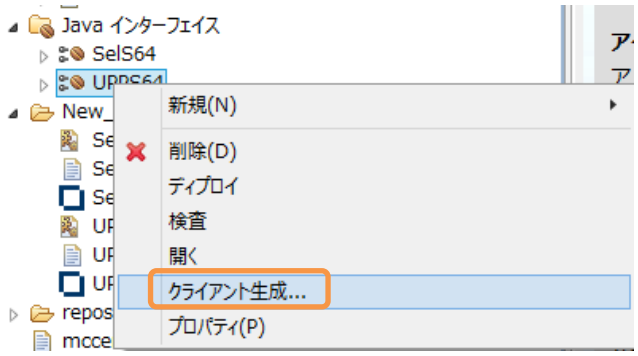


正常にディプロイできたことを Enterprise Server Administration 画面にて確認することができます：

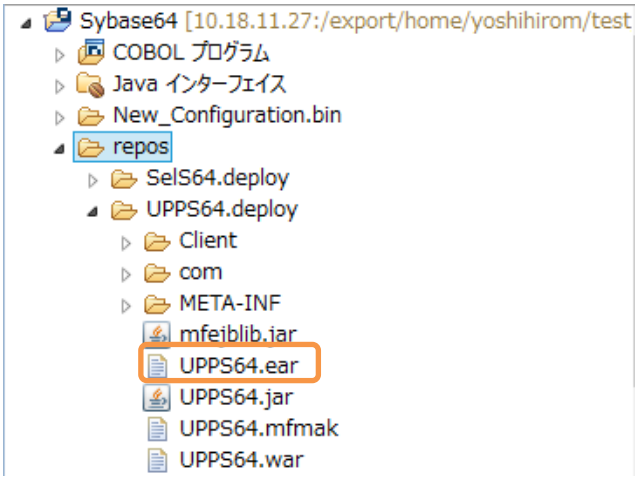


14) ディプロイしたサービスをテスト呼び出しするスタブクライアントアプリケーションを作成

COBOL エクスプローラにて Java インターフェイスを右クリックして［クライアント生成］を選択



COBOL エクスプローラにて .ear にアーカイブされたアプリケーションが生成されていることを確認できます：

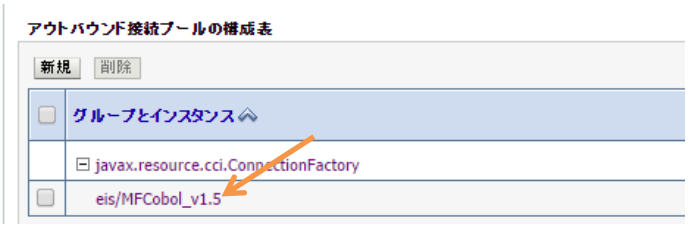


■ Solaris サーバ

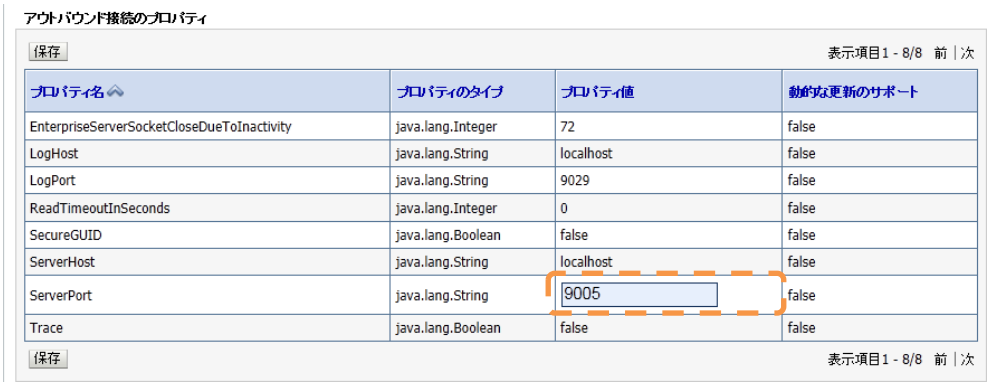
15) 付録 3 で追加したリソースアダプタが 64bit の Enterprise Server へポイントするよう構成を編集

① WebLogic Server Administration Console へログイン
② 左ペインより対象のドメイン配下の［デプロイメント］をクリック
③ 追加したリソースアダプタ mfcobol-xa をクリック
④ ［構成］タブをクリック
⑤ ［アウトバウンド接続プール］をクリック
⑥ javax.resource.cci.ConnectionFactory を展開

⑦ eis/MFCobol_v1.5 をクリック



⑧ ServerPort 欄をクリックし、デフォルトの 9003 から上で指定した 9005 へポート番号を変更



⑨ [保存] ボタンを押下

⑩ デプロイメント・プランの保存確認画面では [OK] ボタンを押下

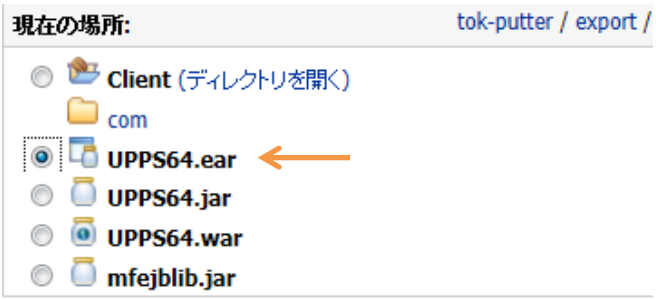
16) 13) で生成したスタブクライアントを WebLogic へインストール

① WebLogic Server Administration Console の左ペインより対象のドメイン配下の [デプロイメント] をクリック

② [インストール] ボタンを押下

③ [パス] 欄に生成された .ear が格納されたディレクトリを指定し Enter を打鍵

④ 生成された UPPS64.ear を選択し、[次へ] ボタンを押下



⑤ [このデプロイメントをアプリケーションとしてインストールする] が選択されていることを確認して、[次へ] ボタンを押下

⑥ [終了] ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます：

■ Windows クライアントマシン

17) Enterprise Server が動的デバッグ有効で起動されていることを確認

Enterprise Server Administration 画面を開き、ESDEMO64 の行における [編集] ボタンを押下し、確認

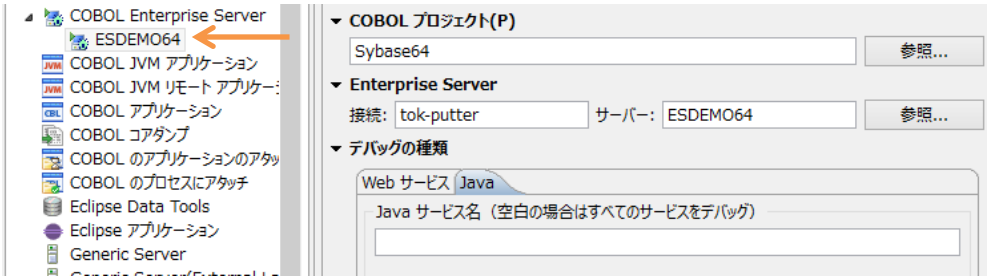
18) Enterprise Server デバッグを起動

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]

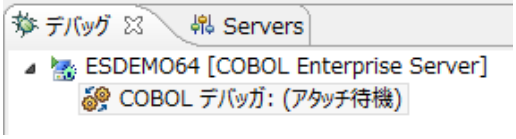
を選択

② 付録 2 で作成したデバッグ構成をダブルクリック



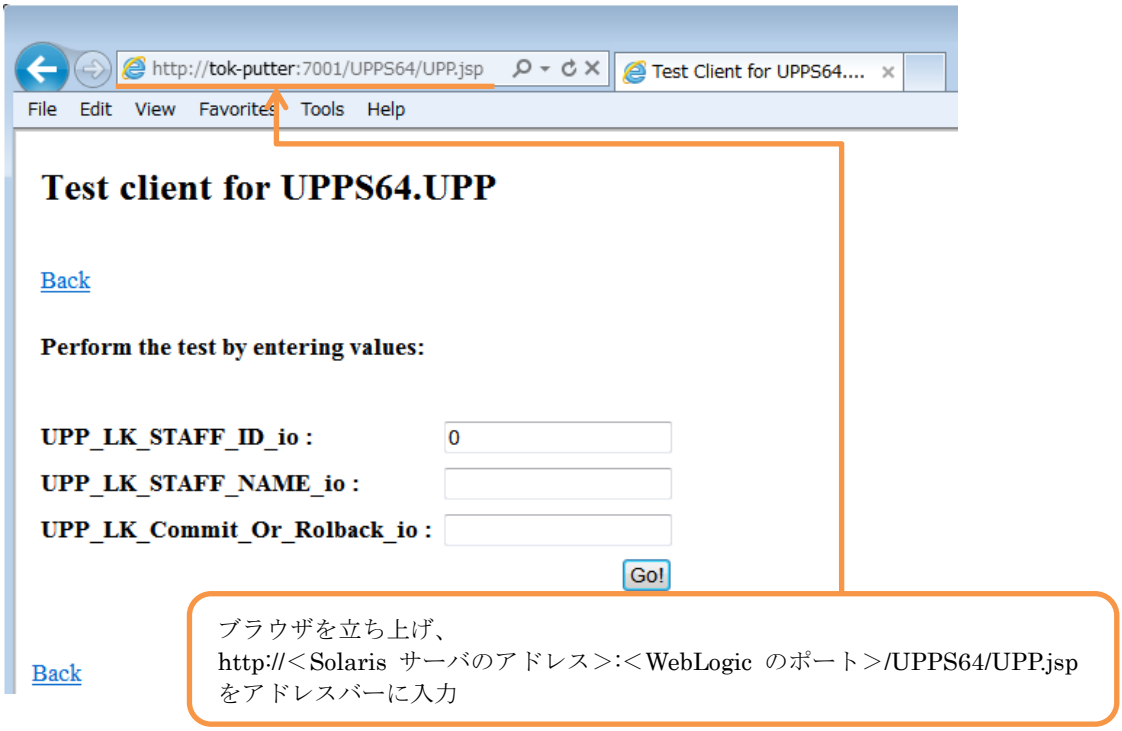
③ [パースペクティブの切り替えの確認] 画面にて [はい] を選択

デバッグパースペクティブにてアタッチ待機状態になっていることを確認できます：

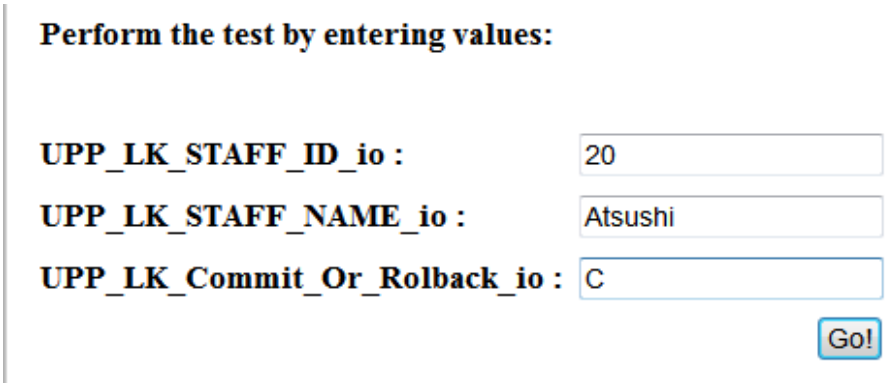


19) ディプロイしたアプリケーションをデバッグ実行しデータを更新

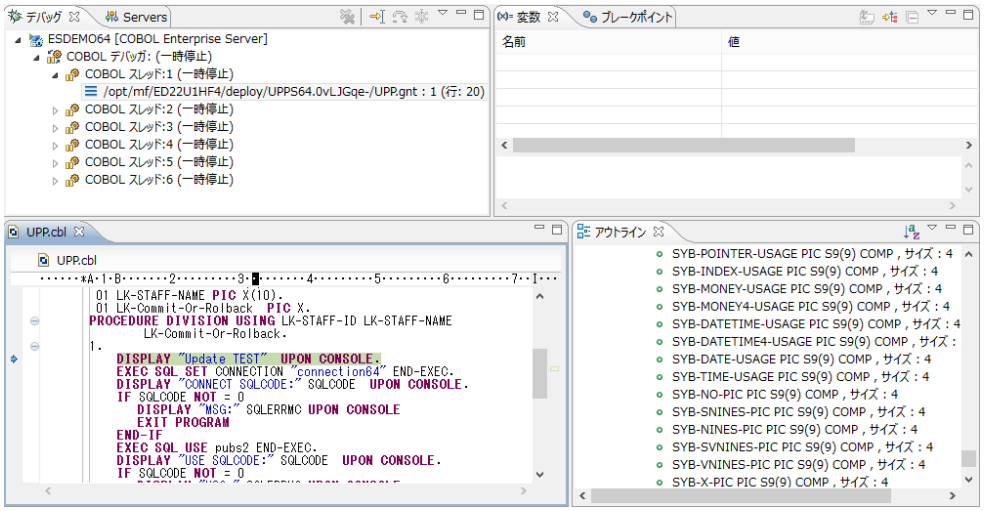
① WebLogic にディプロイしたスタブクライアントアプリケーションを起動



② [UPP_LK_STAFF_ID_io] 欄に [**20**] を
[UPP_LK_STAFF_NAME_io] 欄に [**Atsushi**] を
[UPP_LK_Commit_Or_Rollback_io] 欄に [**C**] を
入力し [Go!] ボタンを押下



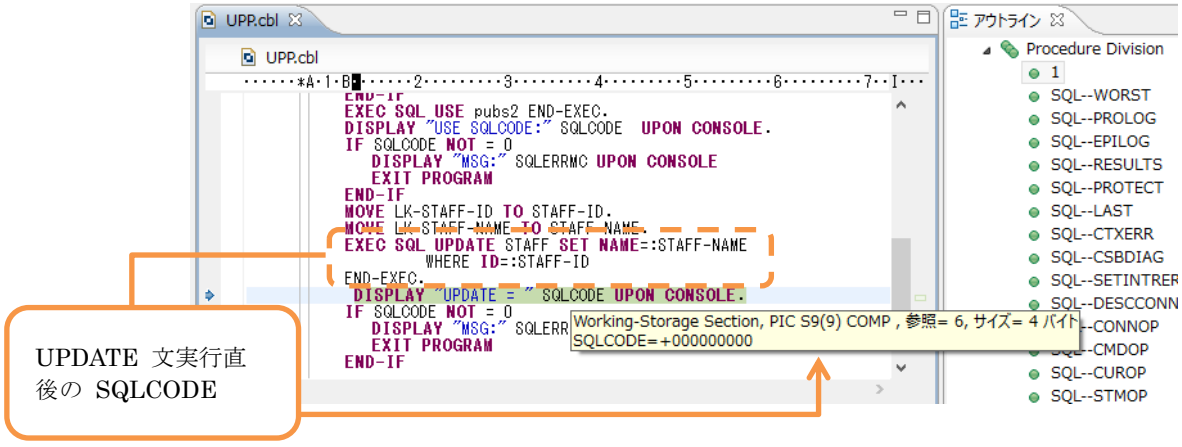
処理が Enterprise Server に渡り、Eclipse のデバッガがその処理を引き込みます。Enterprise Server にディプロイした COBOL プログラムの最初の行を実行する前で処理が停止しているのが確認できます：



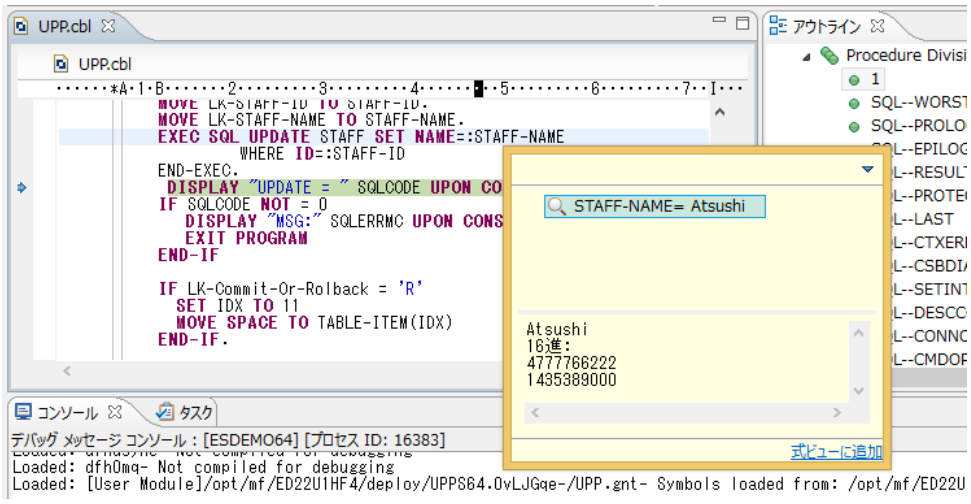
③ デバッグ実行

これまでの検証と同様に F5 キー打鍵でプリコンパイル前の埋め込み SQL 文が入った状態のソースを 1 ステップずつ処理を進めることができます。

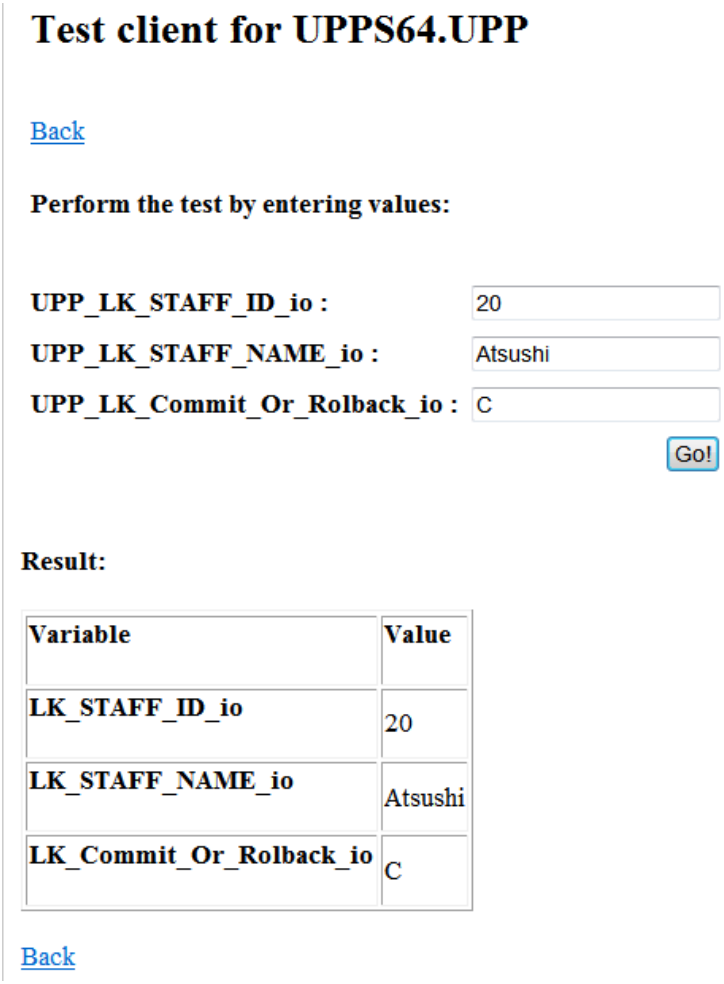
本プログラムはトランザクションマネージャが確立した接続を利用するため、プログラムから CONNECT 文は発行していませんが、下図のように正常に SQL 文を実行できます：



デバッグ実行イメージ：



COBOL 側の処理を最後まで進めると、Java 側に処理が戻り Web の画面が切り替わります：



■ Solaris サーバ

20) アプリケーションがレコードを正しく更新していることを確認

```
$ isql -Usa -Psapass -STOKPUTTER
1> use pubs2
2> go
1> select * from STAFF
2> go
 ID          NAME       DEPT        JOB         YEARS   SALARY
        COMM
 ----------- ---------- ----------- ---------- ------- -------------
        -------------
          10 Hogan              333 Mgr               7       18357.50
                0.00
          20 Atsushi            333 Sales             8       18171.25
              612.45
          30 Marenhi             38 Mgr               5       17506.75
                0.00

(3 rows affected)
1>
```

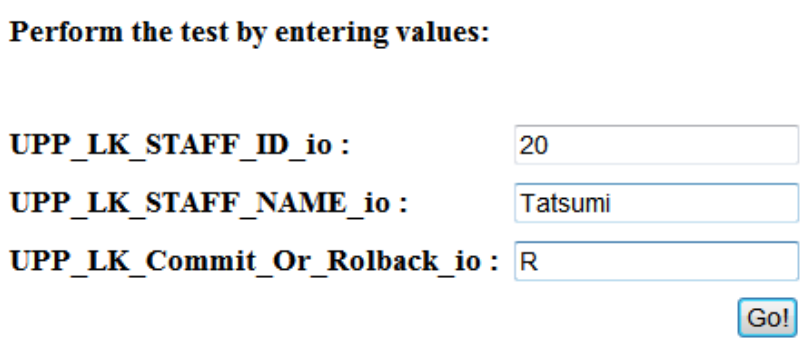
付録２の検証にて照会した際は「Takeshi」が格納されていましたが、「Atsushi」に更新されています。

■ Windows クライアントマシン

21) トランザクションが取り消される条件でパラメータを渡し、ディプロイしたアプリケーションをデバッグ実行

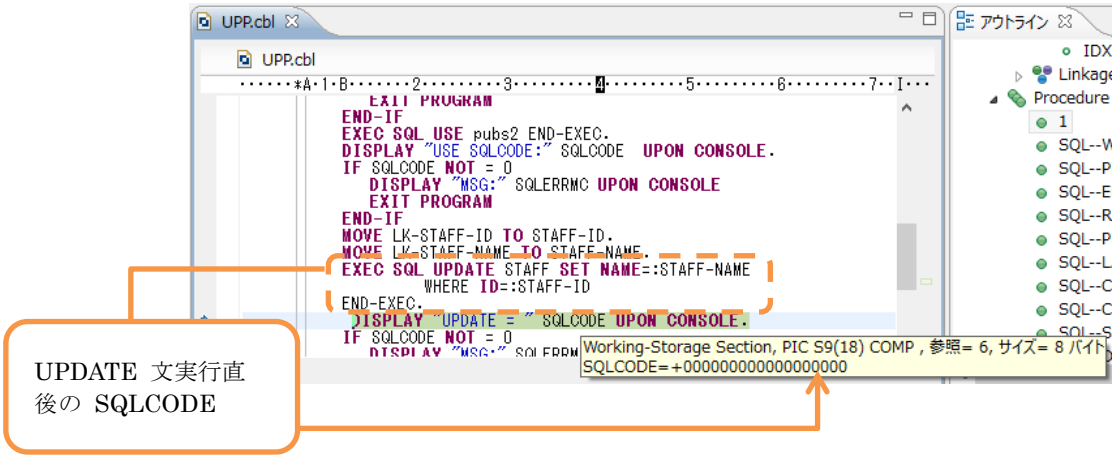
④ WebLogic にディプロイしたスタブクライアントアプリケーションを起動

⑤ [UPP_LK_STAFF_ID_io] 欄に [**20**] を
[UPP_LK_STAFF_NAME_io] 欄に [**Tatsumi**] を
[UPP_LK_Commit_Or_Rollback_io] 欄に [**R**] を
入力し [Go!] ボタンを押下



⑥ デバッグ実行

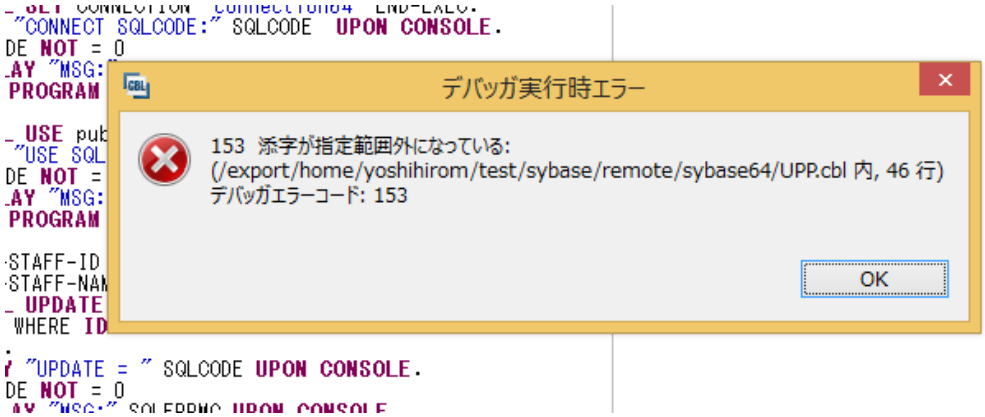
Update 文実行後の SQLCODE を見ると 0 が返ってきており、Update 文は Sybase 上で正常に実行できたことがわかります：



ステップをそのまま進めていきますと第３パラメータに「R」を指定したため、10 回の繰り返し項目の添え字に 11 を格納するロジックへ入ります：

```
    EXEC SQL INCLUDE SQLCA END-EXEC.
01 TABLE-ITEM PIC X OCCURS 10 TIMES INDEXED BY IDX.
LINKAGE SECTION.
PROCEDURE DIVISION USING LK-STAFF-ID LK-STAFF-NAME
        LK-Commit-Or-Rolback.
1.
    DISPLAY "Update TEST"  UPON CONSOLE.
    EXEC SQL SET CONNECTION "connection64" END-EXEC.
    DISPLAY "CONNECT SQLCODE:" SQLCODE  UPON CONSOLE.
    IF SQLCODE NOT = 0
       DISPLAY "MSG:" SQLERRMC UPON CONSOLE
       EXIT PROGRAM
    END-IF
    EXEC SQL USE pubs2 END-EXEC.
    DISPLAY "USE SQLCODE:" SQLCODE  UPON CONSOLE.
    IF SQLCODE NOT = 0
       DISPLAY "MSG:" SQLERRMC UPON CONSOLE
       EXIT PROGRAM
    END-IF
    MOVE LK-STAFF-ID TO STAFF-ID.
    MOVE LK-STAFF-NAME TO STAFF-NAME.
    EXEC SQL UPDATE STAFF SET NAME=:STAFF-NAME
            WHERE ID=:STAFF-ID
    END-EXEC.
     DISPLAY "UPDATE = " SQLCODE UPON CONSOLE.
    IF SQLCODE NOT = 0
       DISPLAY "MSG:" SQLERRMC UPON CONSOLE
       EXIT PROGRAM
    END-IF

    IF LK-Commit-Or-Rolback = 'R'
      SET IDX TO 11
      MOVE SPACE TO TABLE-ITEM(IDX)
    END-IF.
```

繰り返しの範囲を超えた添え字を指定して MOVE 文を実行すると実行時エラーが発生します：



デバッガ側で処理を止めずに処理を進める場合、 Java 側へも COBOL の処理で例外が発生したことが伝播されブラウザにもエラーが発生した旨が表示されます：

Error 500--Internal Server Error

```
javax.ejb.EJBException: UPP threw a resource Exception
        at com.mypackage.UPPS64.UPPS64Bean.UPP(Unknown Source)
        at com.mypackage.UPPS64.UPPS64EJB_fdig5p_UPPS64Impl.__WL_invoke(Unknown Source)
        at weblogic.ejb.container.internal.SessionRemoteMethodInvoker.invoke(SessionRemoteMethodInvoker.java:34)
        at com.mypackage.UPPS64.UPPS64EJB_fdig5p_UPPS64Impl.UPP(Unknown Source)
        at com.mypackage.UPPS64.UPPS64EJB_fdig5p_UPPS64Impl_CBV.UPP(Unknown Source)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at weblogic.ejb.container.internal.RemoteBusinessIntfProxy.invoke(RemoteBusinessIntfProxy.java:84)
        at com.sun.proxy.$Proxy86.UPP(Unknown Source)
        at com.mypackage.UPPS64.UPPS64Servlet.performTask(Unknown Source)
        at com.mypackage.UPPS64.UPPS64Servlet.doPost(Unknown Source)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:751)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:844)
        at weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:280)
        at weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:254)
        at weblogic.servlet.internal.StubSecurityHelper.invokeServlet(StubSecurityHelper.java:136)
        at weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:346)
        at weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:243)
        at weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.wrapRun(WebAppServletContext.java:3432)
        at weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:3402)
        at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:321)
        at weblogic.security.service.SecurityManager.runAs(SecurityManager.java:120)
        at weblogic.servlet.provider.WlsSubjectHandle.run(WlsSubjectHandle.java:57)
        at weblogic.servlet.internal.WebAppServletContext.doSecuredExecute(WebAppServletContext.java:2285)
        at weblogic.servlet.internal.WebAppServletContext.securedExecute(WebAppServletContext.java:2201)
        at weblogic.servlet.internal.WebAppServletContext.execute(WebAppServletContext.java:2179)
        at weblogic.servlet.internal.ServletRequestImpl.run(ServletRequestImpl.java:1572)
        at weblogic.servlet.provider.ContainerSupportProviderImpl$WlsRequestExecutor.run(ContainerSupportProviderImpl.java:255)
        at weblogic.work.ExecuteThread.execute(ExecuteThread.java:311)
        at weblogic.work.ExecuteThread.run(ExecuteThread.java:263)
Caused by: javax.resource.spi.EISSystemException: CobolException Recoverable:
目的コード エラー: ファイル '/opt/mf/ED22U1HF4/deploy/UPPS64.OvLJGqe-/UPP.gnt'
エラーコード: 153, pc=0, call=1, seg=0
153  添字が指定範囲外になっている (/export/home/yoshihirom/test/sybase/remote/sybase64/UPP.cbl 内, 46 行) executing UPP.UPP
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:291)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:
```

■ Solaris サーバ

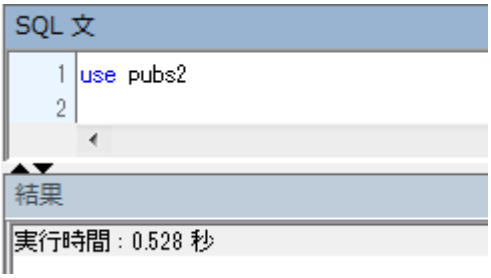
22) 「Tatsumi」へ更新する UPDATE 文のトランザクションが取り消されていることを確認

```
$ isql -Usa -Psapass -STOKPUTTER
1> use pubs2
2> go
1> select * from STAFF
2> go
 ID          NAME       DEPT        JOB         YEARS   SALARY
         COMM
 ----------- ---------- ----------- ----------- ------- --------------
         --------------
          10 Hogan              333 Mgr               7       18357.50
                0.00
          20 Atsushi            333 Sales             8       18171.25
              612.45
          30 Marenhi             38 Mgr               5       17506.75
                0.00

(3 rows a
1>
```

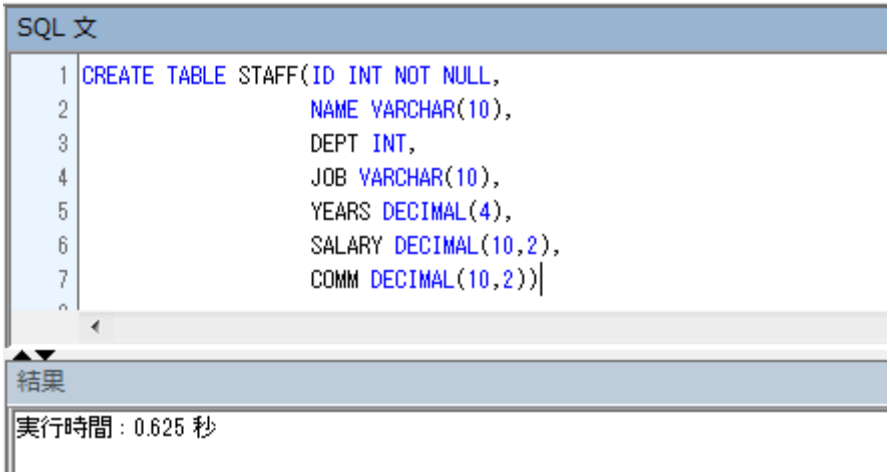
20) で確認した「Atsushi」がそのまま残っています。

備考. Sybase 上で使用するテーブル staff の Data Definition 及びデータ

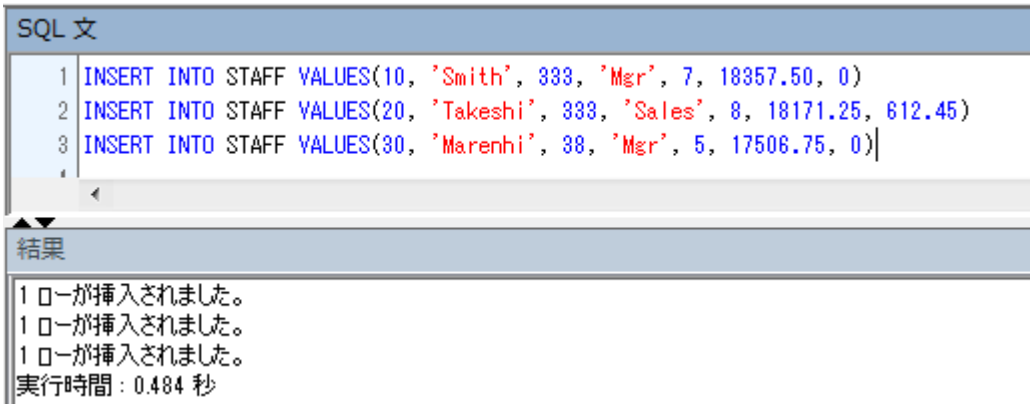
1) Interactive SQL を起動し、Database Server へ接続
2) 使用するデータベースを pubs2 に切り替え



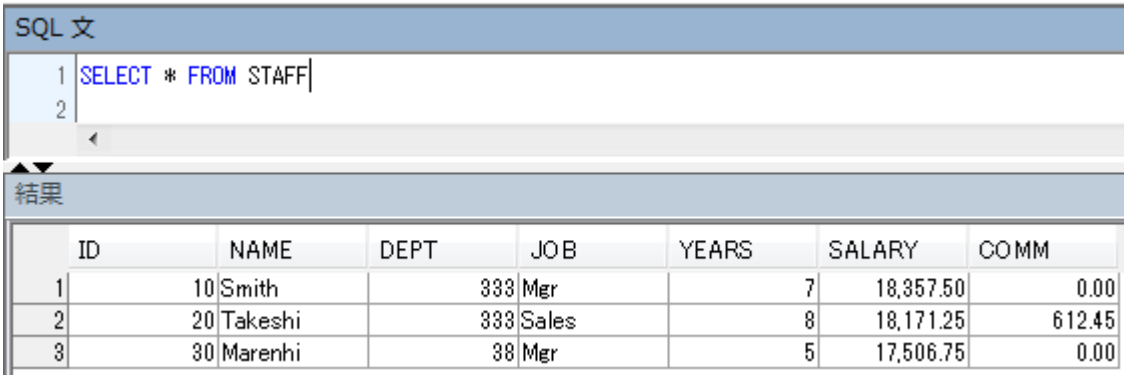
3) 検証で利用するテーブルを作成



4) レコードを３件追加



5) 用意したデータを確認



	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
1	10	Smith	333	Mgr	7	18,357.50	0.00
2	20	Takeshi	333	Sales	8	18,171.25	612.45
3	30	Marenhi	38	Mgr	5	17,506.75	0.00

以上