Micro Focus Visual COBOL チュートリアル

JCA による Oracle WebLogic 連携

アプリケーション開発 編

1. 目的

Micro Focus Visual COBOL に付属する COBOL 専用のアプリケーションサーバ「Enterprise Server」は、Native コードにコンパイルした COBOL アプリケーションを EJB 経由で呼び出し可能なサービス、SOAP に準拠した Web Service、RESTful な Web Service として公開することを可能にします。前者の機能を使って公開する場合、Java EE アプリケーションサーバ上の Java EE クライアントは EJB を呼び出すと JCA の仕様に則ったリクエストが Enterprise Server に渡され COBOL アプリケーションが処理をして結果を返します。更にこの Enterprise Server は、XA に準拠したリソースマネージャとして動作するよう設計されたデータベース等に対してトランザクションや接続等をコンテナ管理する機能も備えています。

Micro Focus Visual COBOL の UNIX/Linux 版をご購入いただきますと、UNIX/Linux 環境へインストールする Development Hub 並びに Windows 環境へインストールする Eclipse 版のライセンスが付与されます。これにより、Windows 上の Eclipse で操作しつつも、実際には Linux/UNIX 上のソースを編集し、コンパイル命令を発行するとこれらの環境上にモジュールを生成するとリモート開発機能をご利用いただけます。更に本機能は生成されたモジュールを Windows 側でデバッグ実行することも可能としています。

Pro*COBOL を使った Oracle 連携アプリケーションの開発を考えた場合、通常ではプリコンパイルした後にコンパイルするというステップを踏む必要がありますが、Visual COBOL にはコンパイル時に内部的に Pro*COBOL を利用してワンステップでコンパイルする COBSQL という技術も提供しております。この技術を利用すれば、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 Eclipse 上で直接メンテナンスすることが可能です。

本書は上述した各技術を使い Oracle Database 連携の COBOL アプリケーションを Java EE アプリケーションの一部として開発するようすをチュートリアル形式で紹介いたします。

2. 前提

本チュートリアルを実施するには事前に以下に列挙した項目を満たす必要があります。

- Linux 環境に Visual COBOL 2.3J for x64/x86 Linux に付属する Visual COBOL Development Hub がインストール済であること[1] [2]
- Windows 環境に Visual COBOL 2.3J for x64/x86 Linux に付属する Visual COBOL for Eclipse がインストール済であること[3]
- Linux 環境と Windows 環境がネットワーク通信可能であること
- Linux 環境に Oracle Weblogic Server がインストール済且つドメインが構成済みであること[4]
- Linux 環境に Oracle Database Client もしくは Server がインストール済であること[5]
- Linux 環境より Oracle Database Client を通じて利用可能な Oracle Database Server が存在すること
- Linux 環境より接続される Oracle Database には Oracle が提供する Scot のサンプルスキーマが存在すること

3. チュートリアル手順の概要

1. Oracle Database にテストデータをストア
2. XA Switch モジュールの準備
3. Oracle 連携の COBOL のアプリケーションを開発
4. Enterprise Server へ生成したアプリケーションをディプロイ
5. Java EE クライアントアプリケーションを WebLogic Server にディプロイ
6. Java EE クライアントアプリケーションより COBOL アプリケーションをテスト呼び出し
7. Enterprise Server 配下で稼働する COBOL アプリケーションを Eclipse IDE で動的デバッグ

---

[1] 本書では Visual COBOL 2.3J for x64/x86 Linux(HotFix 1 適用版) を RedHat Enterprise Linux 7.1 にインストールした環境を例にとり紹介しています。OS によって多少出力内容や利用する OS のコマンド等に差はありますが、チュートリアル中で取り上げる技法の多くは他の Linux/UNIX OS でも流用できます。

[2] チュートリアル中で示す Linux 上での作業は全て ja_JP.UTF-8 ロケール配下で処理しています。

[3] 本書では Visual COBOL 2.3J for Eclipse(HotFix 1 適用版) を Windows 10 Enterprise 32 bit にインストールした環境を例に紹介しています。OS によっては多少画面イメージが異なることもあります。

[4] 本書では Oracle Weblogic Server 12c(12.2.1) をインストールした環境を利用しています。

[5] 本書では 64bit 版 Oracle Database 12c(12.1.0.2.0) をインストールした環境を利用しています。

4. チュートリアル手順

4.1 Oracle Database にテストデータをストア

Linux Server 上での作業

1) SQL*Plus で Oracle Database に接続

2) 検証で利用するテーブルを作成

ここでは、Oracle が提供するサンプルスキーマに含まれる SCOTT.EMP テーブルを EMP_WK にコピーして準備しています：

```
SQL> select user from dual;

USER
------------------------------
SCOTT

SQL> create table emp_wk as select * from emp;

Table created.

SQL>
```

Scott ユーザでログオンしていることを確認

SCOTT.EMP テーブルを SCOTT.EMP_WK にコピー

3) テーブルの格納値を確認

```
SQL> select EMPNO, ENAME from EMP_WK where EMPNO between 7300 and 7700;

     EMPNO ENAME
---------- ----------
      7369 SMITH
      7499 ALLEN
      7521 WARD
      7566 JONES
      7654 MARTIN
      7698 BLAKE

6 rows selected.

SQL>
```

4.2 XA Switch モジュールの準備

Linux Server 上での作業

1) Linux サーバへログイン

2) Visual COBOL の環境設定スクリプトを実行

Visual COBOL Development Hub のインストールディレクトリが /opt/mf/VC23HF1 の場合の出力例：

```
$ . /opt/mf/VC23HF1/bin/cobsetenv
COBDIR set to /opt/mf/VC23HF1
$
```

<インストールディレクトリ>/bin/cobsetnenv にあるスクリプトを実行します

3) Switch Module のソースファイル及びビルドスクリプトを作業フォルダにコピー

```
$ cp $COBDIR/src/enterpriseserver/xa/* ./
$
```

4) Switch Module をビルド

```
$ ./build ora
building 64-bit switch module...

* Cobsql Integrated Preprocessor
* CSQL-I-018: Oracle プリコンパイラトランスレータを起動しま▒
* CSQL-I-020: Oracle プリコンパイラの出力を処理中。
* CSQL-I-001: COBSQL：チェッカへの引き渡しを完了しました。

* Cobsql Integrated Preprocessor
* CSQL-I-018: Oracle プリコンパイラトランスレータを起動しま▒
* CSQL-I-020: Oracle プリコンパイラの出力を処理中。
* CSQL-I-001: COBSQL：チェッカへの引き渡しを完了しました。

If you intend to execute JES-initiated transactions under Enterprise Server,
then ESORAXA64.so needs to reside in a directory included in your
$LD_LIBRARY_PATH setting, such as $COBDIR/lib.

If you do not do so, then such transactions will not be able to communicate
with the database server.
$
```

5) モジュールが生成されたことを確認[6]

```
$ ls
ESDB2XA.CBL   ESORAXA.CBL   ESORAXA64_D.so  build    ctf64.cfg  xaws.cpy
ESODBCXA.CBL  ESORAXA64.so  ESPGSQLXA.CBL   ctf.cfg  xapd.cpy
$$
```

ビルドされた Switch Module

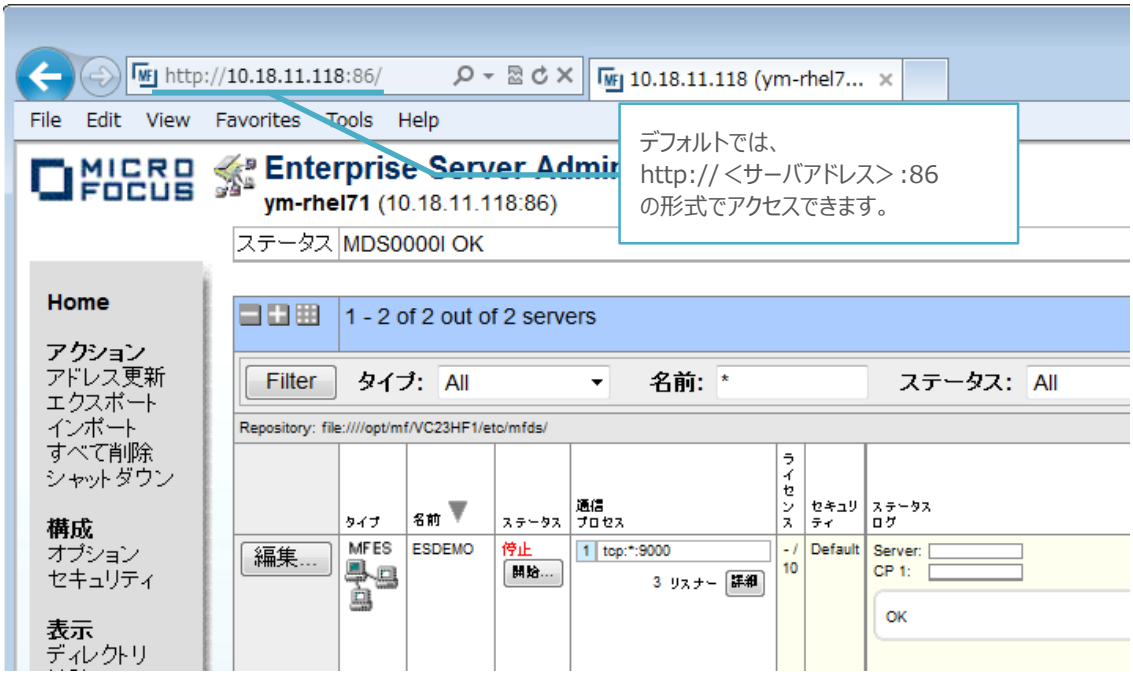
[6] ESORAXA64.so は Enterprise Server へスイッチモジュールを静的に登録するスイッチモジュールです。ESORAXA64_D.so は動的に登録します。本検証ではこのうちの ESORAXA64_D.so を利用します。

6) Micro Focus Directory Server が起動されていない場合は、Micro Focus Directory Server を起動

```
$ su
パスワード:
# mfds &
[1] 5425
#
```

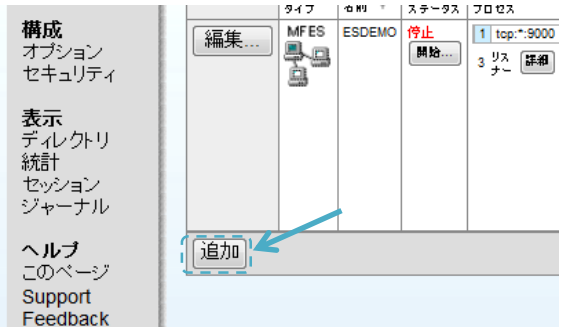
root に切り替え、mfds コマンドをバックグラウンドで実行

7) ブラウザを起動して Enterprise Server Administration 画面を表示

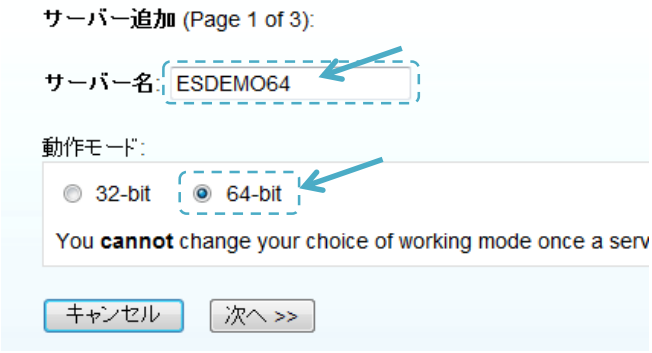


8) 64 bit 版の Enterprise Server を追加

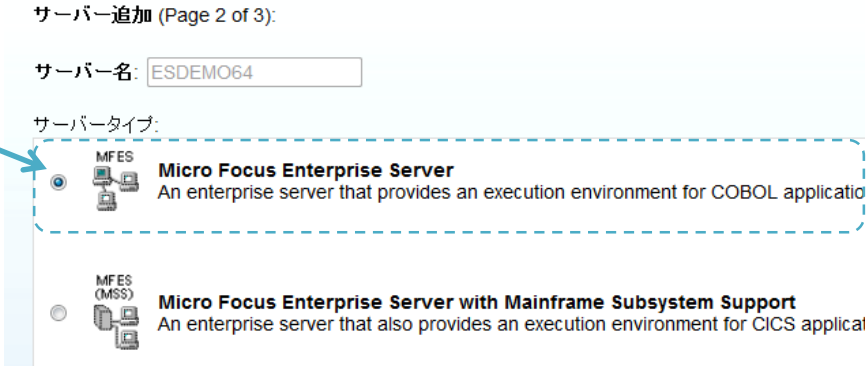
① Enterprise Server Administration 画面にて [追加] ボタンを押下



② [サーバ名] 欄には任意のサーバ名を指定し、[動作モード] 欄は [64-bit] を指定し、[次へ] ボタンを押下

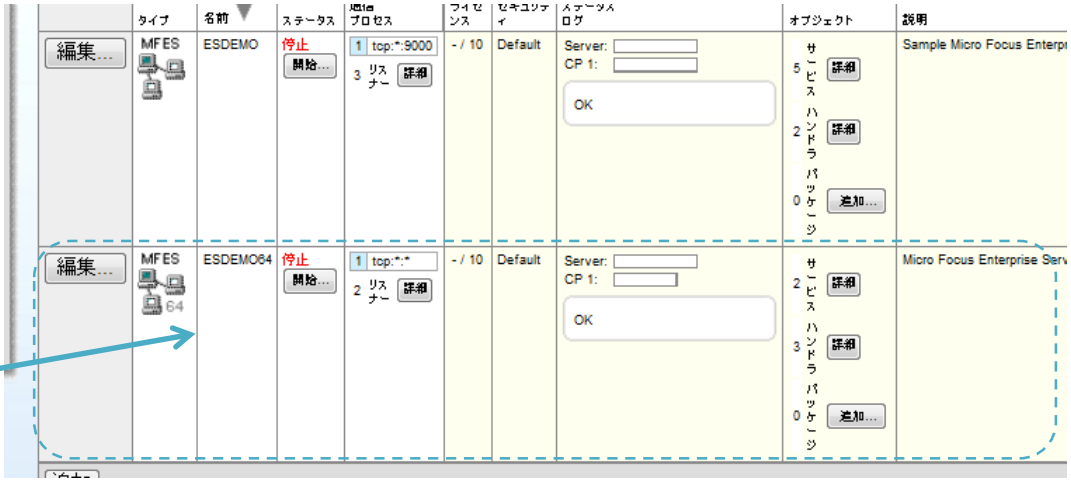


③ [サーバータイプ] 欄では [Micro Focus Enterprise Server] を選択し [次へ] ボタンを押下



④ [追加] ボタンを押下

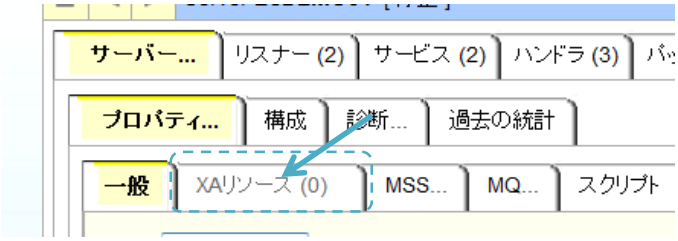
トップ画面に戻り、追加した Enterprise Server がリストに追加されていることを確認できます：



9) Enterprise Server に XA Resource Manager 情報を定義

① 追加した 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 XA リソースの情報を指定し、[OK] ボタンを押下

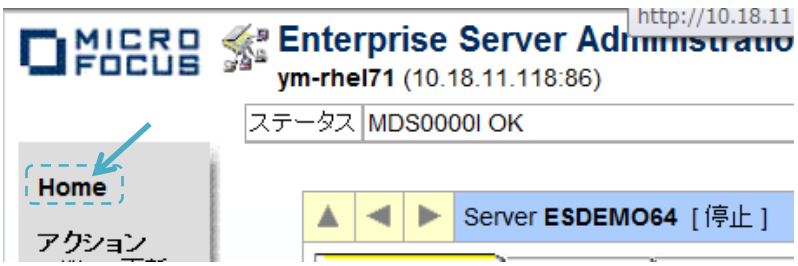
指定後の画面[7]：

ID	名前	モジュール	Open文字列	Close 文字列
ORCL	ORACLE12C	/home/yoshihiro/work/wpJBoss/switchmod/ESORAXA64_D.so	Oracle_XA+SesTm +SqlNet=ORCL+Acc=P/scott/tiger+LogDir=/tmp/oraxalog	

本欄における設定値：

パラメータ名	設定内容
ID	Enterprise Server 内でユニークな XA リソース ID を指定
名前	xa_switch_t 構造の NAME フィールドと同じ値を指定
モジュール	上の手順でビルドしたスイッチモジュールを指定
Open 文字列	
SesTm	セッションアイドル時間の上限値（秒単位で指定）
SqlNet	接続対象の DB Server への Oracle Net データベース・リンク名
Acc	Oracle Database への認証情報。「P/」をプリフィックスとして付加。
LogDir	XA ライブラリのエラー及びトレース情報を格納するログディレクトリ
DbgFL	トレースレベル

⑤ [Home] をクリックし、トップ画面に戻る



[7] ここでの表示内容は本書執筆環境に合わせた設定値となります。このチュートリアルをお試しになる場合は、実際の環境に合わせた設定値に適宜置き換えてご利用ください。また各フィールドの詳細等については Oracle のマニュアル等を参照してください。

10) 動的デバッグの有効化

① 追加した Enterprise Server の行にある [編集] ボタンを押下

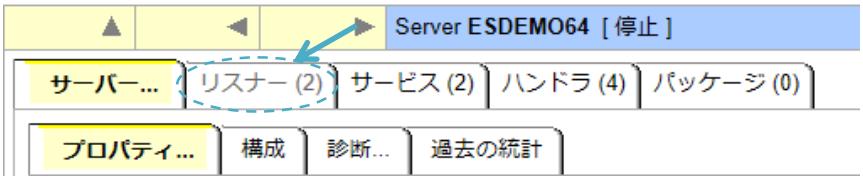
② [動的デバッグを許可] をチェック[8]

③ [OK] ボタンを押下

11) リスナー Port の静的割り当て

① 追加した Enterprise Server の行にある [編集] ボタンを押下

② [リスナー] タブをクリック

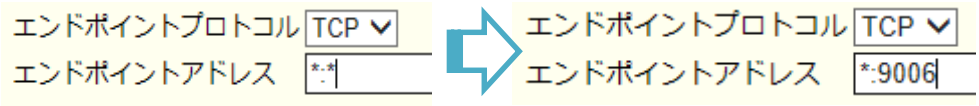


③ 「Web Services and J2EE」列中の [編集] ボタンを押下

[8] 動的デバッグを有効にしなくても Switch Module の登録は可能です。しかし、後のデバッグの際にこの設定が必要となるため、予め本設定を有効にしておきます。

④ エンドポイントアドレスを下図のように「*:*」から「*:＜任意のポート番号＞」へ変更

本例では、ポート番号 9006 に割り当てています：

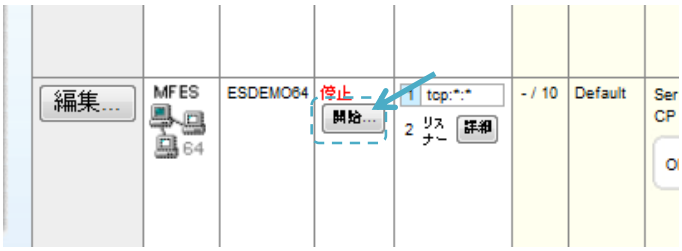


⑤ [OK] ボタンを押下

⑥ 画面左上の Home をクリック

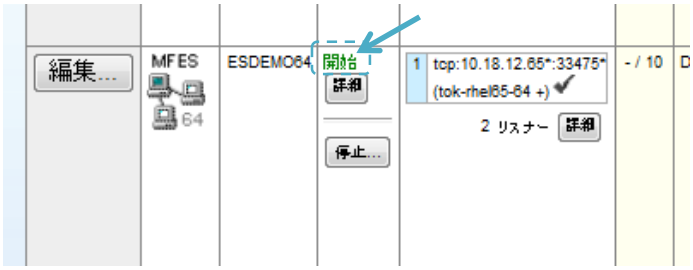
12) Enterprise Server の起動

① 追加した Enterprise Server の行中の [開始] ボタンを押下



② [OK] ボタンを押下

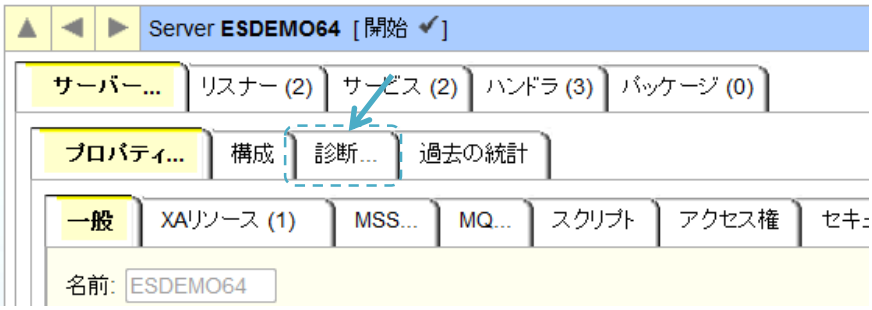
正常に起動されると、ステータス欄の表示が [**開始**] にアップデートされます：



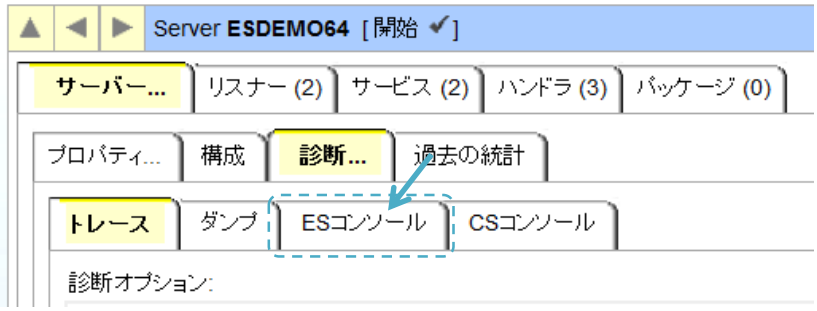
13) XA Switch Module が正常にロードされていることをログより確認

① 対象の Enterprise Server の行中の [編集] ボタンを押下

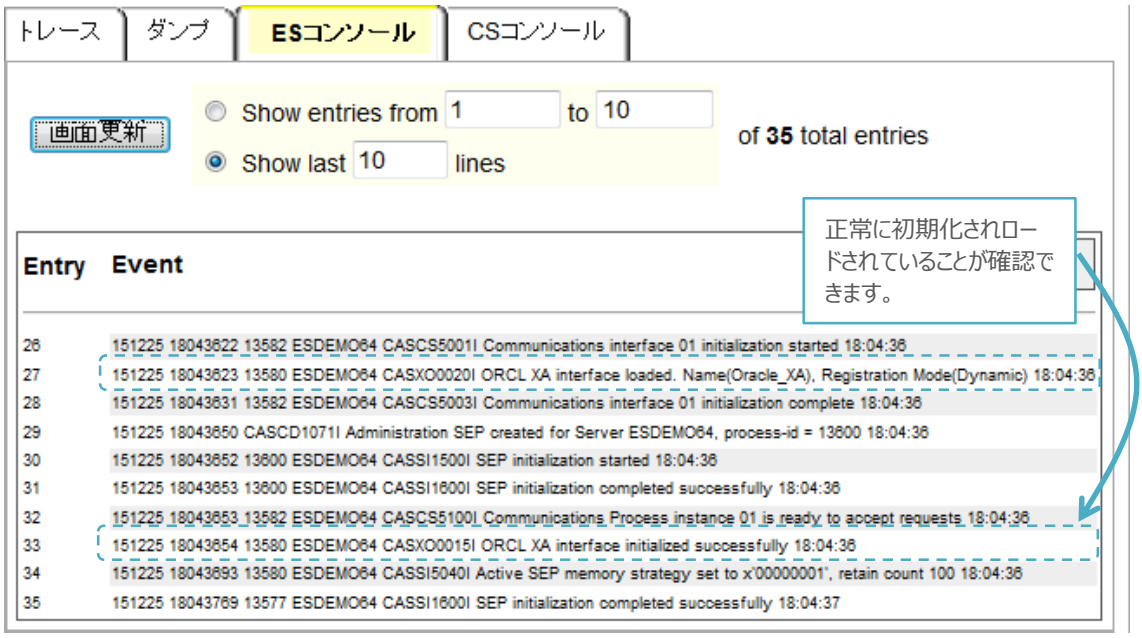
② [診断] をクリック



③ [ES コンソール] をクリック



④ 「CAS**XO**・・・」のエントリを確認



4.3 Oracle 連携の COBOL のアプリケーションを開発

Linux Server 上での作業

1) Linux サーバへログイン

2) リモート開発用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用意

コマンド例：

```
$ mkdir -p $HOME/test/tutorial/rmtprj
$
```

3) root ユーザへ切り替え

4) Visual COBOL の環境設定スクリプトを実行

```
# . ＜Visual COBOL のインストールディレクトリ＞/bin/cobsetenv
COBDIR set to /opt/mf/VC23HF1
#
```

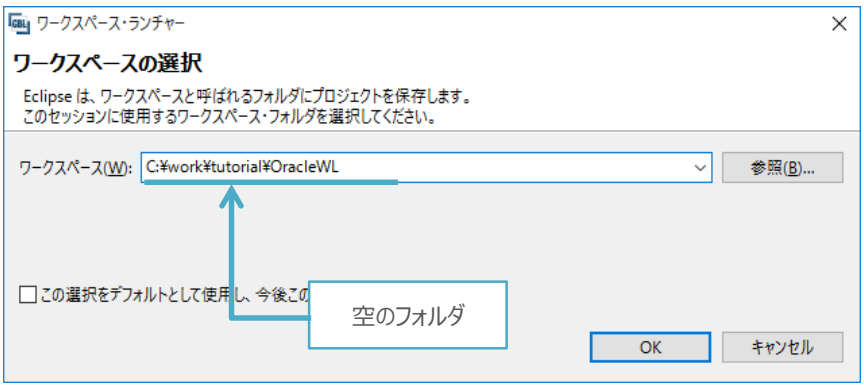
5) COBOL リモート開発用のデーモンを起動

```
# $COBDIR/remotedev/startrdodaemon
Checking Java Version
Correct Java Version installed, proceeding
Starting RSE daemon...
Daemon running on: ym-rhel71, port: 4075
                                                    :
```

Windows 端末上での作業

6) Visual COBOL for Eclipse を起動

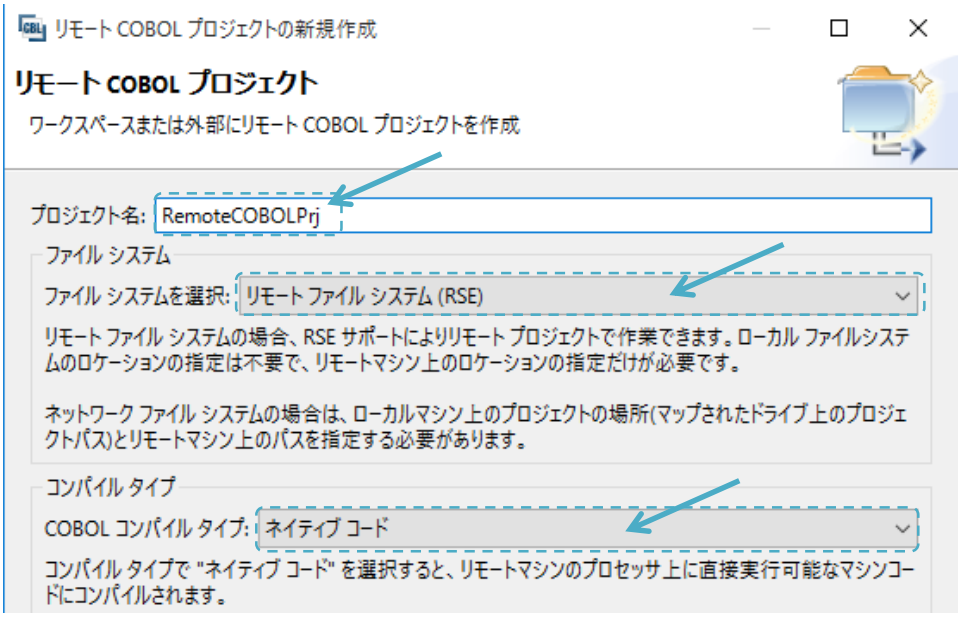
7) ワークスペース・ランチャーにて任意の空フォルダをワークスペースとして指定



8) COBOL リモートプロジェクトを作成

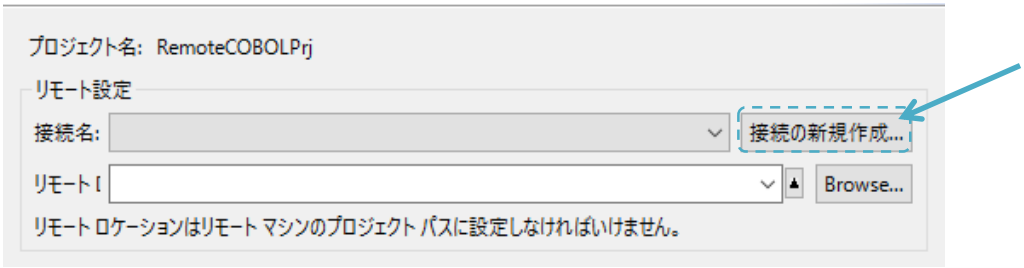
① [ファイル]メニュー ＞ [新規] ＞ [COBOL リモートプロジェクト] を選択

② [プロジェクト名]欄は任意の名前を指定、[ファイルシステムを選択] 欄は [リモートファイルシステム(RSE)] が選択されていることを確認、[COBOL コンパイルタイプ] は [ネイティブコード] が選択されていることを確認し [次へ] ボタンを押下

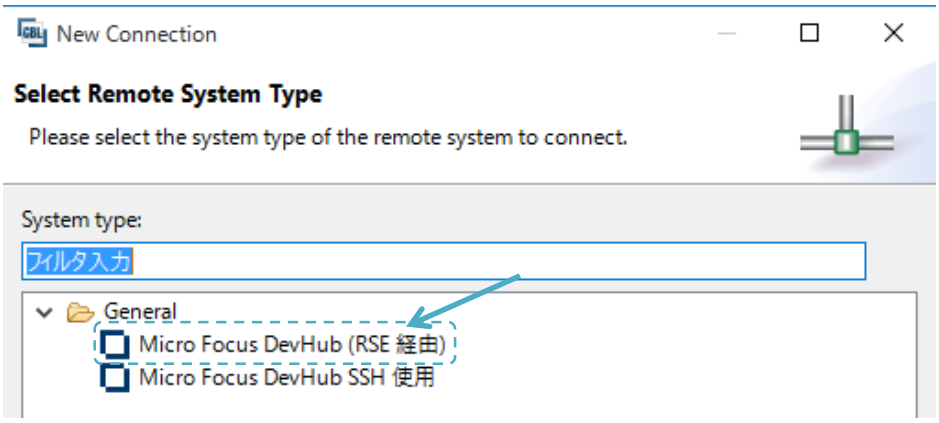


③ [プロジェクトテンプレートを選択] 画面ではデフォルトの [Micro Focus テンプレート] を選択したまま [次へ] ボタンを押下

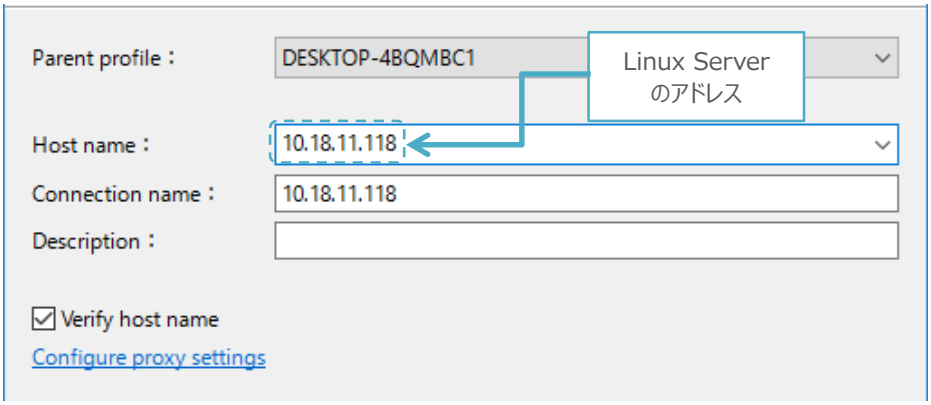
④ [接続の新規作成] ボタンを押下



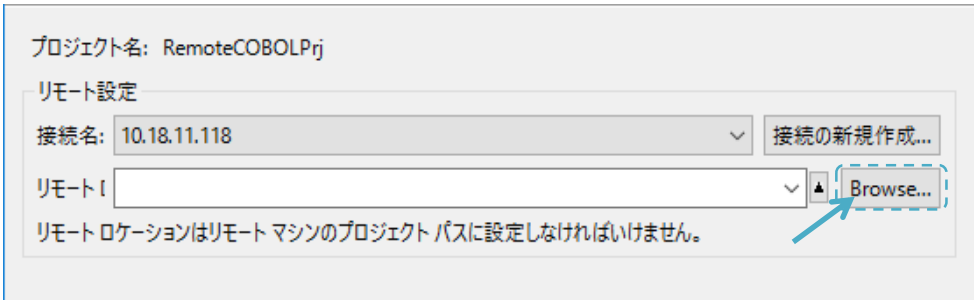
⑤ [Micro Focus DevHub(RSE 経由)] を選択し [次へ] ボタンを押下



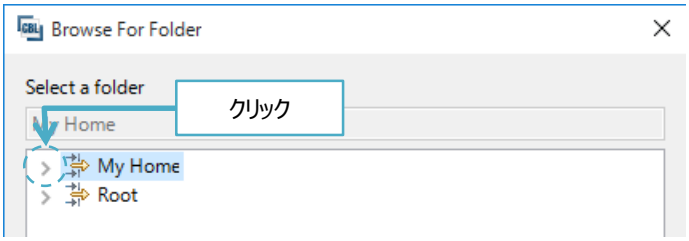
⑥ [Host name] 欄に Linux Server のアドレスもしくはホスト名を入力し [終了] ボタンを押下



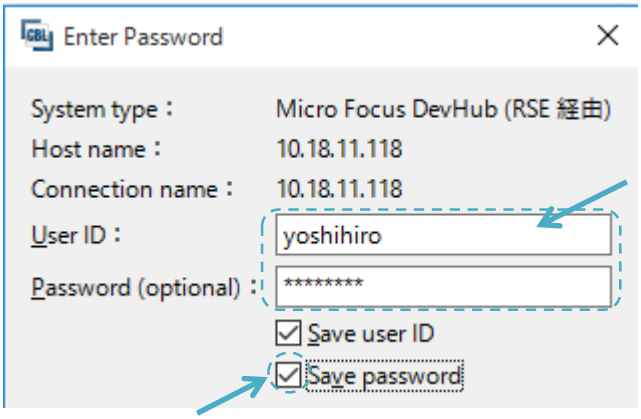
⑦ [Browse] ボタンを押下



⑧ ツリーを展開



⑨ 1）でログインした一般ユーザのログイン情報を指定し、[Save password] にもチェックを入れ、[OK] ボタンを押下



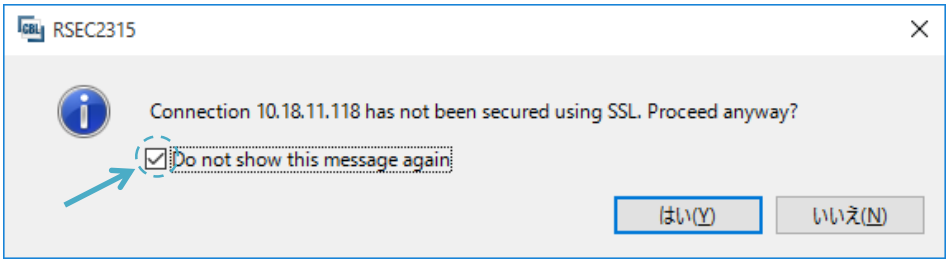
⑩

⑪ 警告が返される場合は確認の上 [はい] を選択

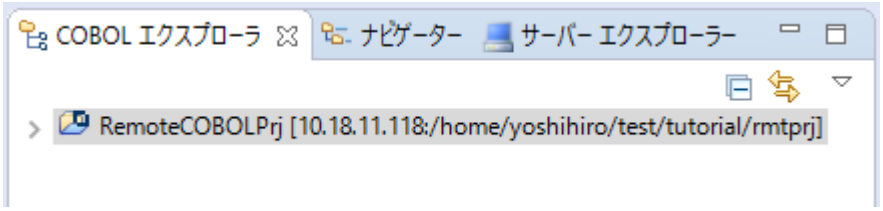
⑫ 2）で作成したディレクトリをツリーにて選択し [OK] ボタンを押下

⑬ [終了] ボタンを押下

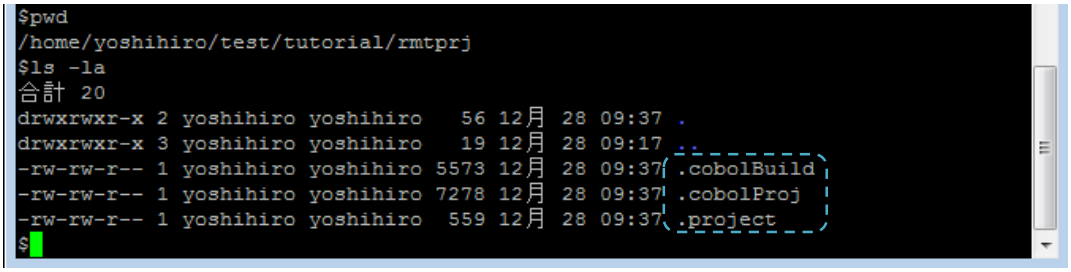
下図のようなポップアップが返される場合は、確認し [Do not show this message again] にチェックを入れ [はい] を選択：



Eclipse の COBOL エクスプローラにて Linux Server 上のプロジェクトディレクトリをポイントしているプロジェクトが生成されていることを確認できます：



実際、Linux Server に接続したターミナルで確認するとプロジェクトディレクトリにプロジェクトファイル等が生成されていることが確認できます：

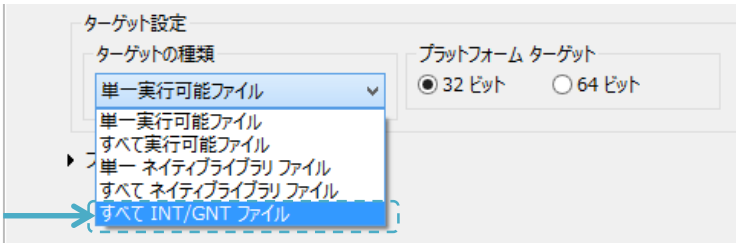


9) モジュールのターゲット及びコンパイラ指令を指定

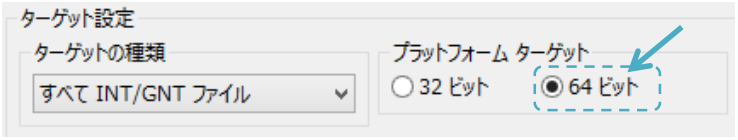
① COBOL エクスプローラにてプロジェクトを右クリックし、[プロパティ] を選択

② 左ペイン中のツリーにて [Micro Focus] > [ビルド構成] > [COBOL] へとナビゲート

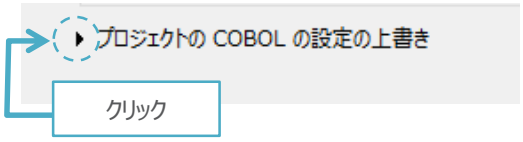
③ [ターゲットの種類] 欄にて [すべて INT/GNT ファイル] を選択



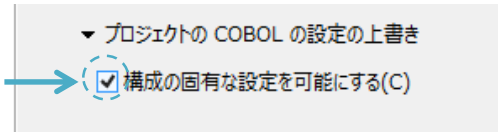
④ [プラットフォームターゲット] 欄にて [64 ビット] を選択



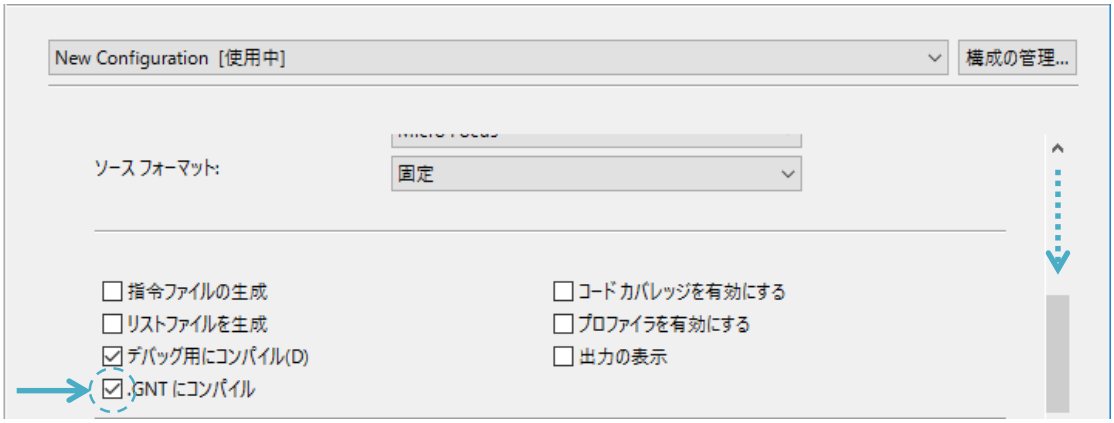
⑤ [プロジェクトの COBOL の設定の上書き] を展開



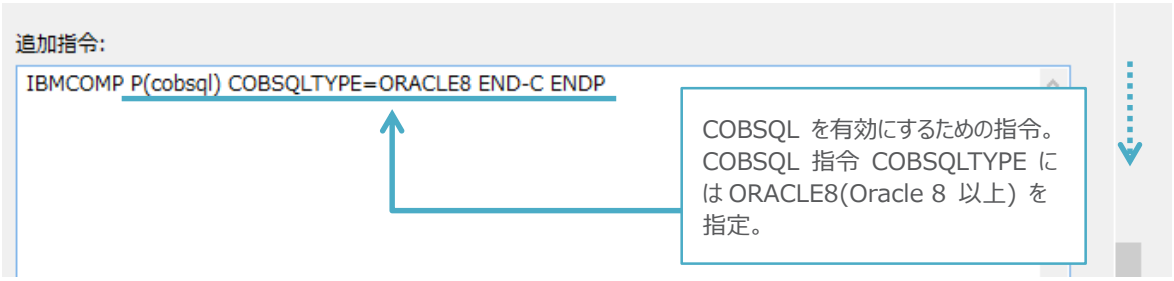
⑥ [構成の固有な設定を可能にする] をチェック



⑦ 画面を少し下へスクロールし [.GNT にコンパイル] をチェック



⑧ 更に画面を下へスクロールし COBSQL を有効にするためのコンパイラ指令を [追加指令] 欄に追加



⑨ [OK] ボタンを押下

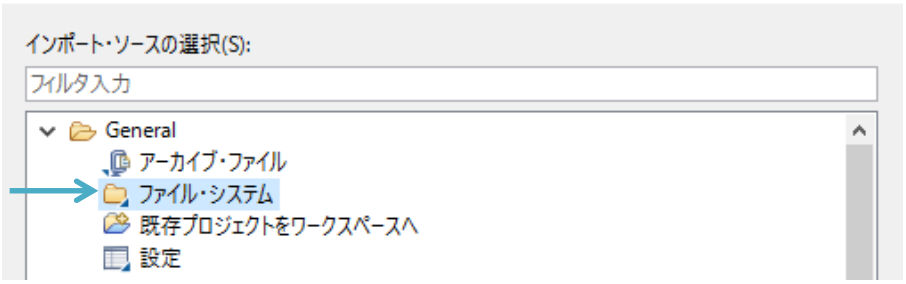
10) プログラムソースをプロジェクトにインポート

① COBOL エクスプローラにてプロジェクトを右クリックし

[インポート] > [インポート]

を選択

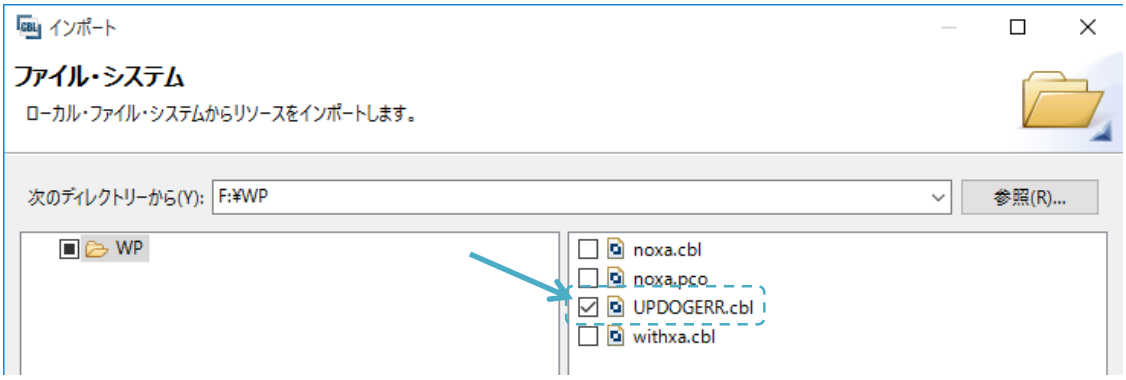
② [General] > [ファイルシステム] を選択し [次へ] ボタンを押下



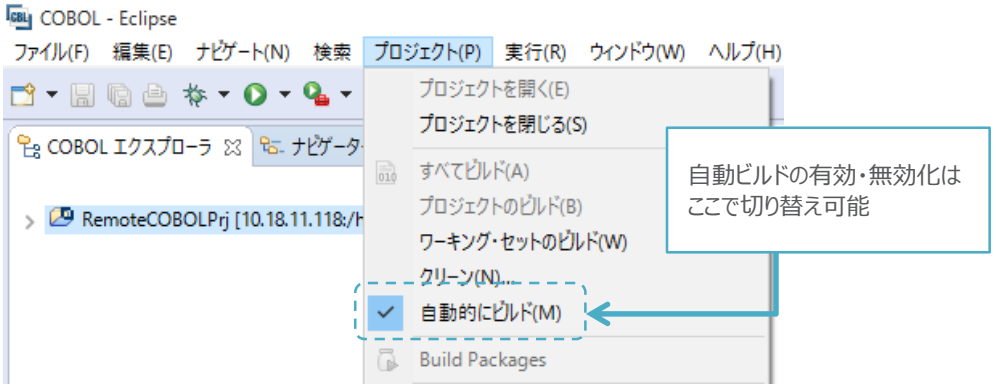
③ [参照] ボタンを押下し、エクスプローラでプログラムソースが格納されたフォルダへナビゲートし [OK] ボタンを押下[9]

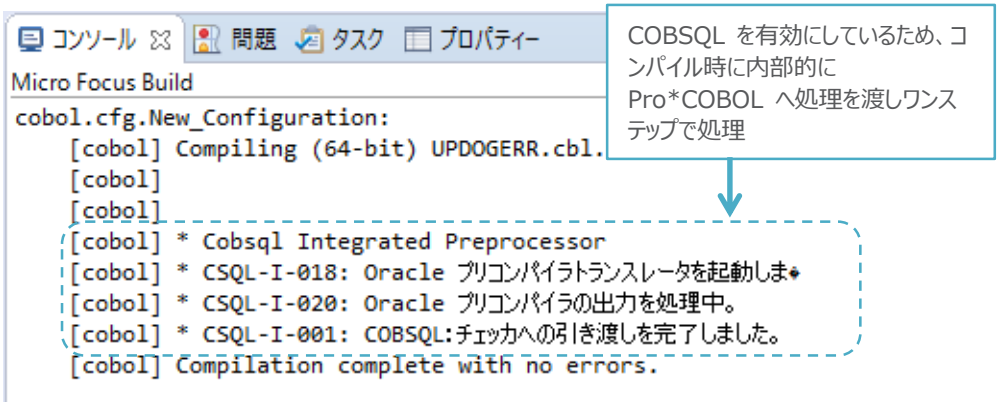
[9] チュートリアル中で利用するプログラムは本書とともに公開しています。

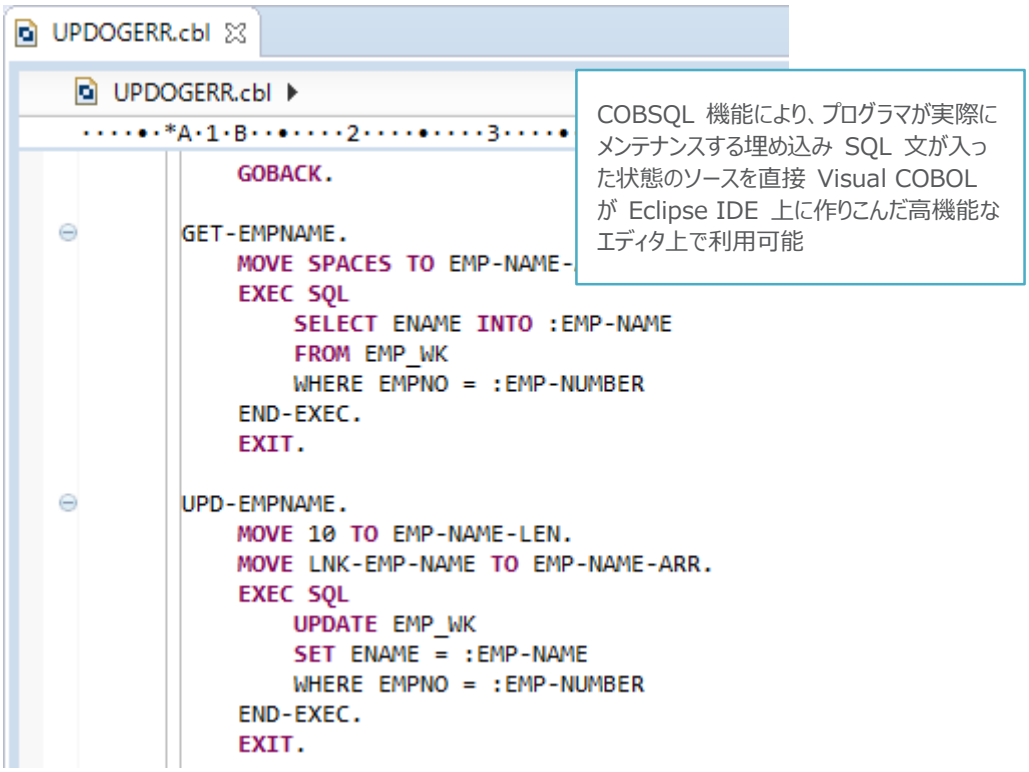
④ チュートリアルで利用するプログラム UPDOGERR.cbl にチェックを入れ、[終了] ボタンを押下



自動ビルドが有効になっているため、インポートされた時点で自動的にコンパイルされ動的ロードモジュール Linux 環境に生成されます：






UPDOGERR.cbl
UPDOGERR.cbl
GOBACK.
GET-EMPNAME.
MOVE SPACES TO EMP-NAME-
EXEC SQL
SELECT ENAME INTO :EMP-NAME
FROM EMP_WK
WHERE EMPNO = :EMP-NUMBER
END-EXEC.
EXIT.
UPD-EMPNAME.
MOVE 10 TO EMP-NAME-LEN.
MOVE LNK-EMP-NAME TO EMP-NAME-ARR.
EXEC SQL
UPDATE EMP_WK
SET ENAME = :EMP-NAME
WHERE EMPNO = :EMP-NUMBER
END-EXEC.
EXIT.
COBSQL 機能により、プログラマが実際にメンテナンスする埋め込み SQL 文が入った状態のソースを直接 Visual COBOL が Eclipse IDE 上に作りこんだ高機能なエディタ上で利用可能


RemoteCOBOLPrj
COBOL プログラム
UPDOGERR.cbl
New_Configuration.bin
UPDOGERR.gnt
UPDOGERR.gnt.1.tlog
UPDOGERR.idy
$pwd
/home/yoshihiro/test/tutorial/rmtprj
$ls -l New_Configuration.bin/
合計 72
-rw-r--r-- 1 yoshihiro yoshihiro 8055 12月 28 09:53 UPDOGERR.gnt
-rw-r--r-- 1 yoshihiro yoshihiro 299 12月 28 09:53 UPDOGERR.gnt.1.tlog
-rw-r--r-- 1 yoshihiro yoshihiro 58368 12月 28 09:53 UPDOGERR.idy
$
生成された動的ロードモジュール
Windows 上の Eclipse でモジュールが生成されたことを確認できますが、実態は Linux Server 上あります

11) プログラムの処理内容を確認

本サンプルプログラムのポイントを下記に列挙します：

> Database への接続、切断に関するロジックがない
> COMMIT や ROLLBACK などの DCL(データ制御言語) がない
> 受け取ったパラメータに基づき UPDATE 文を実行
> UPDATE 文の前後で更新する/されたレコードの内容をログ排出
> LNK-ERR-FLG に「Y」が渡された場合は意図的に実行時エラーを発生させる

プログラム抜粋：

```
LINKAGE SECTION.
01 LNK-EMP-NUMBER     PIC S9(4) COMP.
01 LNK-EMP-NAME       PIC X(10).
01 LNK-ERR-FLG        PIC X(1).
PROCEDURE DIVISION
        USING LNK-EMP-NUMBER LNK-EMP-NAME LNK-ERR-FLG.
MAIN-PARA.
    MOVE LNK-EMP-NUMBER TO EMP-NUMBER.
    PERFORM GET-EMPNAME.
    DISPLAY "EMPNAME BEFORE UPDATE: " EMP-NAME-ARR
        UPON CONSOLE.
    PERFORM UPD-EMPNAME.
    PERFORM GET-EMPNAME.
    DISPLAY "EMPNAME AFTER UPDATE: " EMP-NAME-ARR
        UPON CONSOLE.
    PERFORM ERR-HANDLING.
    MOVE EMP-NAME-ARR TO LNK-EMP-NAME.
    GOBACK.

GET-EMPNAME.
    MOVE SPACES TO EMP-NAME-ARR.
    EXEC SQL
        SELECT ENAME INTO :EMP-NAME
        FROM EMP_WK
        WHERE EMPNO = :EMP-NUMBER
    END-EXEC.
    EXIT.

UPD-EMPNAME.
    MOVE 10 TO EMP-NAME-LEN.
    MOVE LNK-EMP-NAME TO EMP-NAME-ARR.
    EXEC SQL
        UPDATE EMP_WK
        SET ENAME = :EMP-NAME
        WHERE EMPNO = :EMP-NUMBER
    END-EXEC.
    EXIT.

ERR-HANDLING.
    IF LNK-ERR-FLG = 'Y' THEN
      SET IDX TO 11
      MOVE SPACE TO TABLE-ITEM(IDX)
    END-IF.
    EXIT.
```

4.4 Enterprise Server へ生成したアプリケーションをディプロイ

Windows 端末上での作業

1) Windows 上で Linux 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

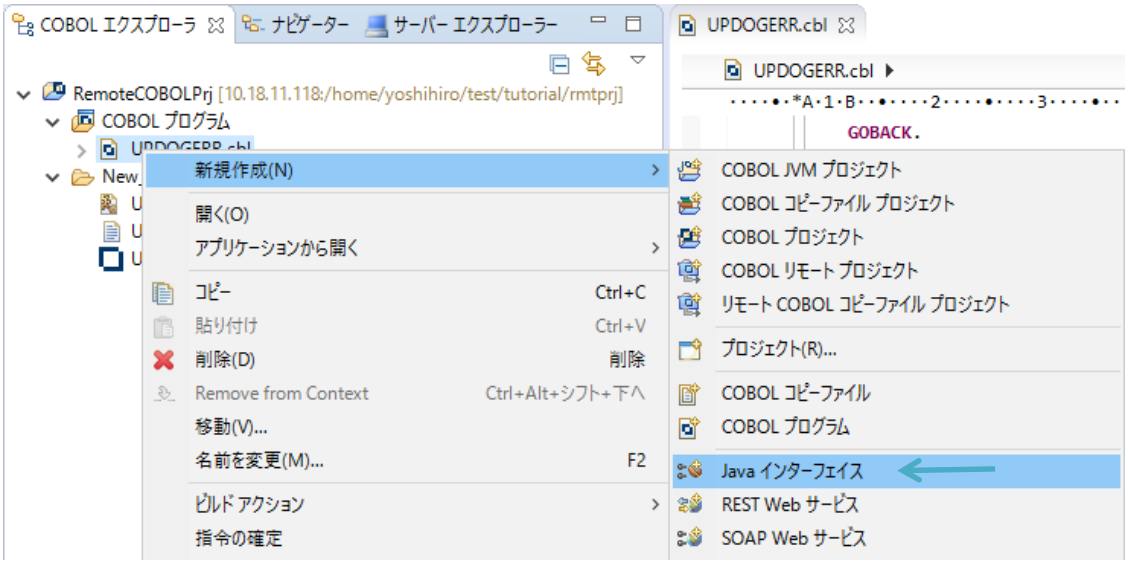
mrpi://<Linux サーバの IP アドレス>:0

の形式で Linux サーバの Directory Server エントリを追加

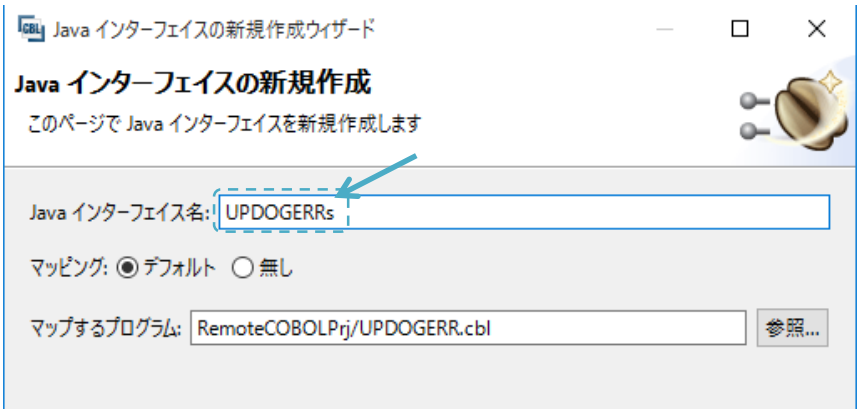
2) Java インターフェイスをプロジェクトに追加

① COBOL エクスプローラにてインポートしたプログラム UPDOGERR.cbl を右クリックし

[新規作成] > [Java インターフェイス] を選択



② [Java インターフェイス] 欄に任意の名前を入力

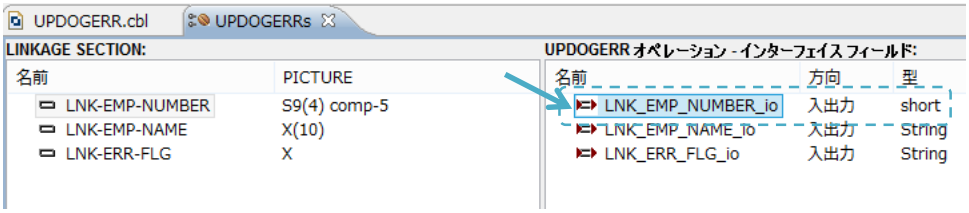


③ [終了] ボタンを押下

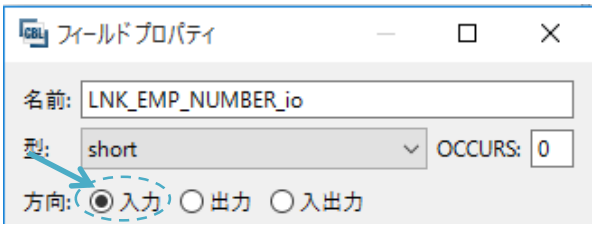
3) COBOL – Java のパラメータ変換マッピングを編集

デフォルトでは全てのパラメータが入出力となっていますが、ここでは、LNK-EMP-NUMBER 及び LNK-ERR-FLG を入力パラメータに変更してみます。

① [LNK_EMP_NUMBER] をダブルクリック

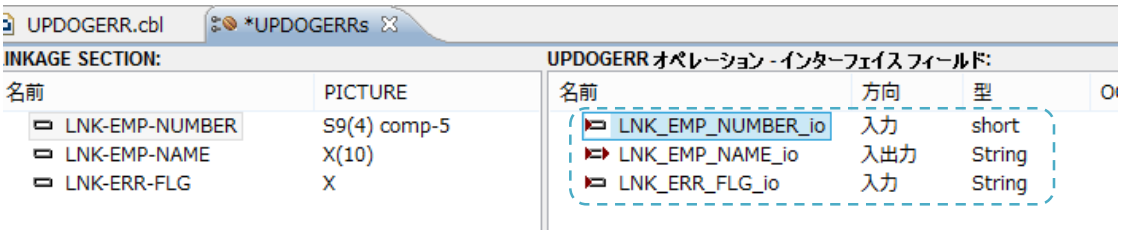


② [方向] を [入力] に変更し [OK] ボタンを押下

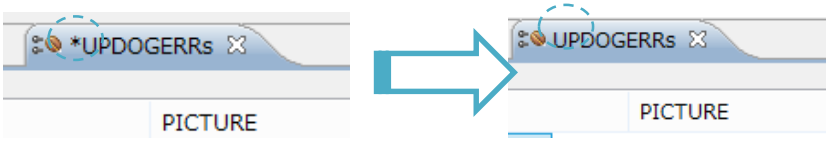


③ 同じ要領で LNK-ERR-FLG も入力パラメータへ変更

編集後の画面：

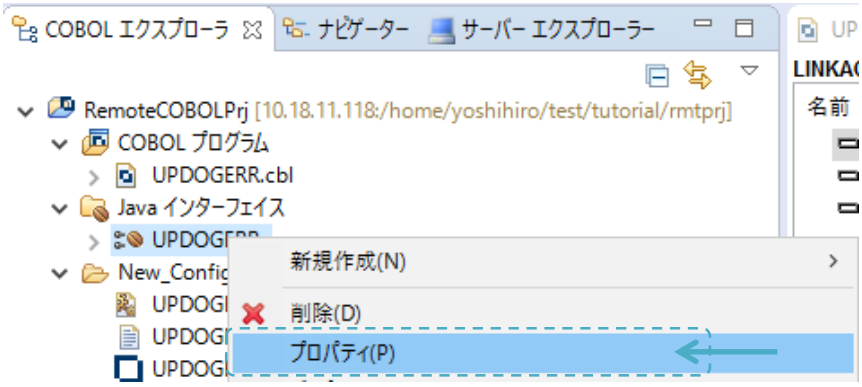


④ Ctrl + S を打鍵し、変更を保存

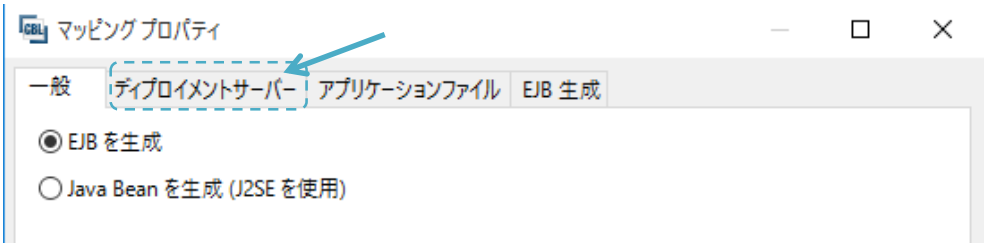


4) ディプロイする COBOL サービスの各種情報を設定

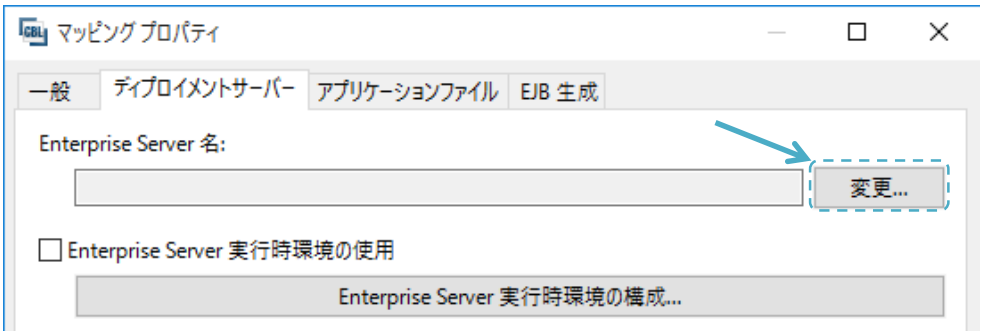
① COBOL エクスプローラにて追加した Java インターフェイスを右クリックし [プロパティ] を選択



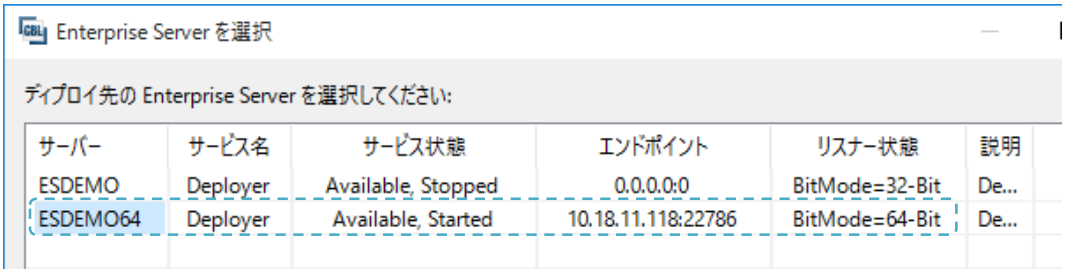
② [ディプロイメントサーバー] タブを選択



③ [Enterprise Server 名] 欄にて [変更] ボタンを押下

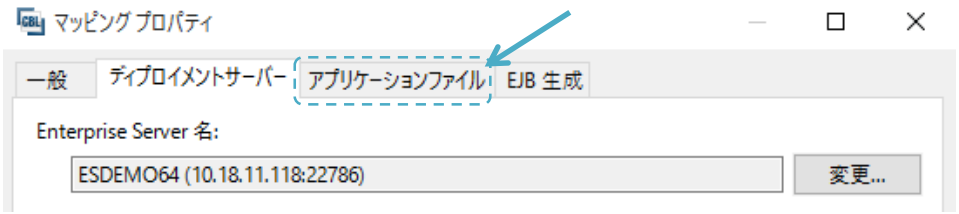


④ Linux Server 上で稼働する XA Switch Module をディプロイした Enterprise Server を選択し [OK] ボタンを押下

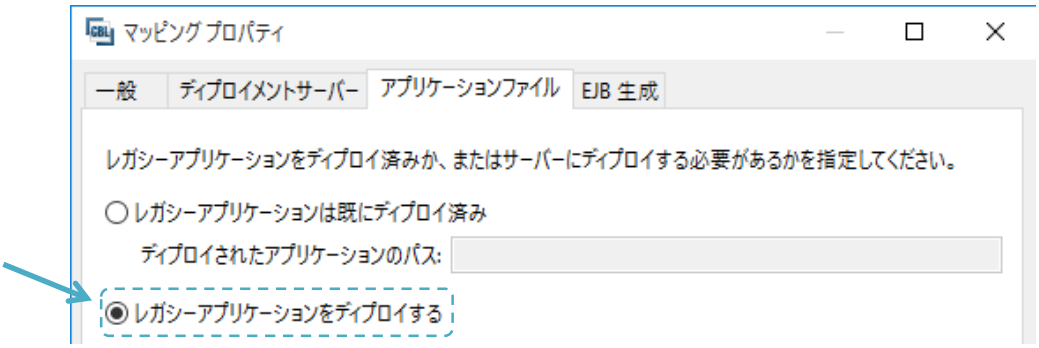


⑤ [トランザクション管理] 欄では [コンテナ管理] を選択

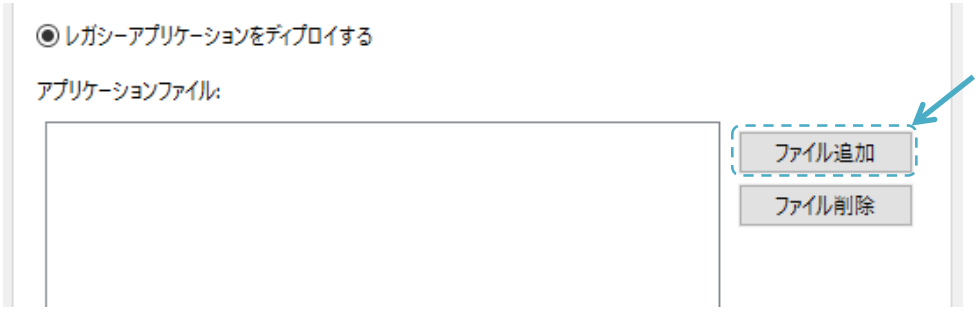
⑥ [アプリケーションファイル] タブを選択



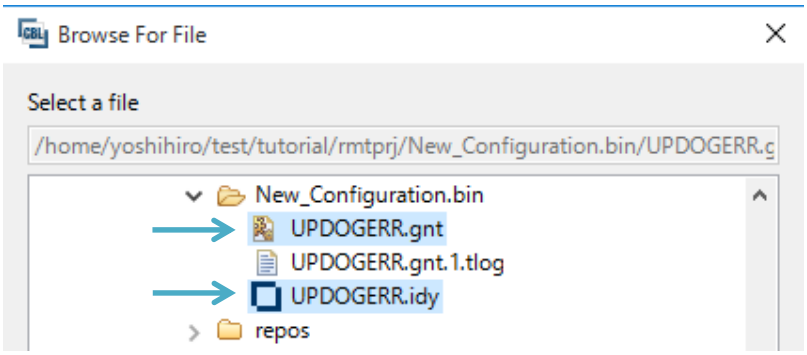
⑦ [レガシーアプリケーションをディプロイする] を選択



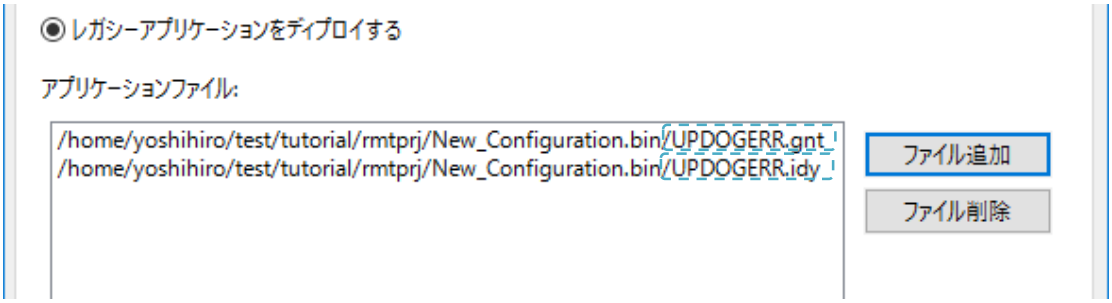
⑧ [ファイル追加] ボタンを押下



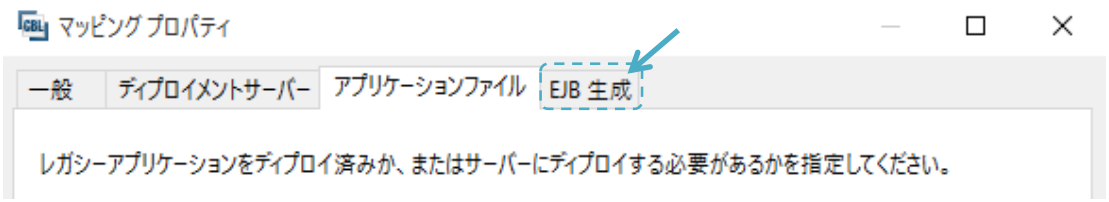
⑨ プロジェクトフォルダ以下に生成された動的ロードモジュール及びデバッグ情報ファイルの 2 ファイルを Ctrl を打鍵しながら選択し、[OK] ボタンを押下



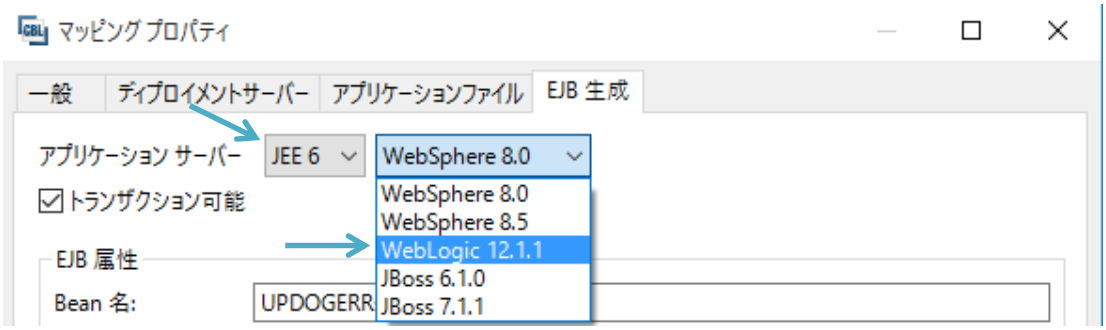
ボタン押下後の画面イメージ：



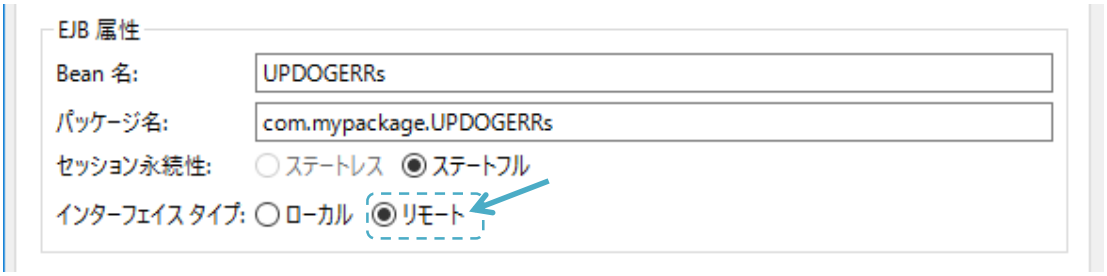
⑩ [EJB 生成] タブをクリック



⑪ [アプリケーションサーバー] 欄にて [JEE6] 及び [WebLogic 12.1.1] を選択



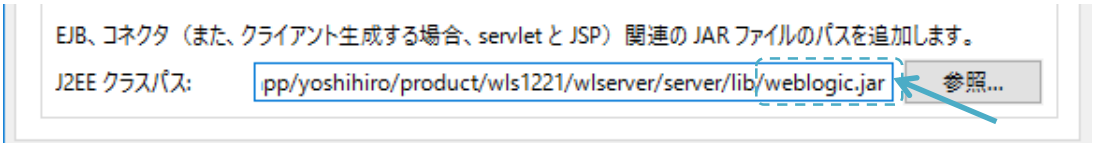
⑫ [インターフェイスタイプ] 欄にて [リモート] を選択



⑬ [J2EE クラスパス] 欄中の [参照] ボタンを押下

⑭ ポップアップされるツリー画面にて WebLogic のインストールディレクトリに格納されている weblogic.jar を選択し [OK] ボタンを押下

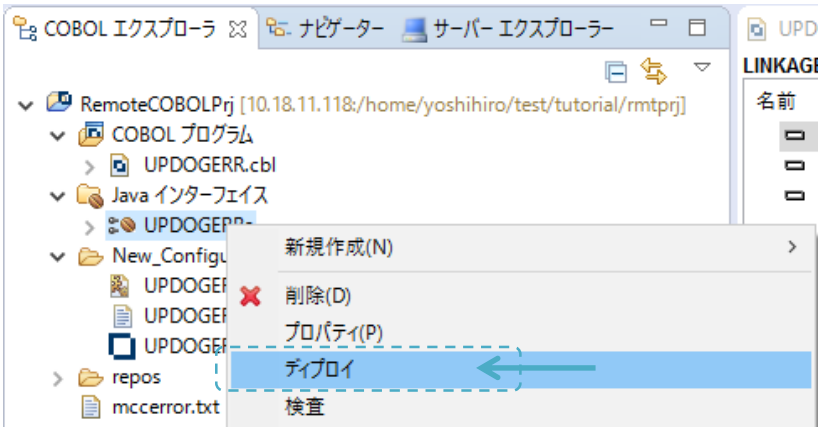
選択後の画面イメージ：



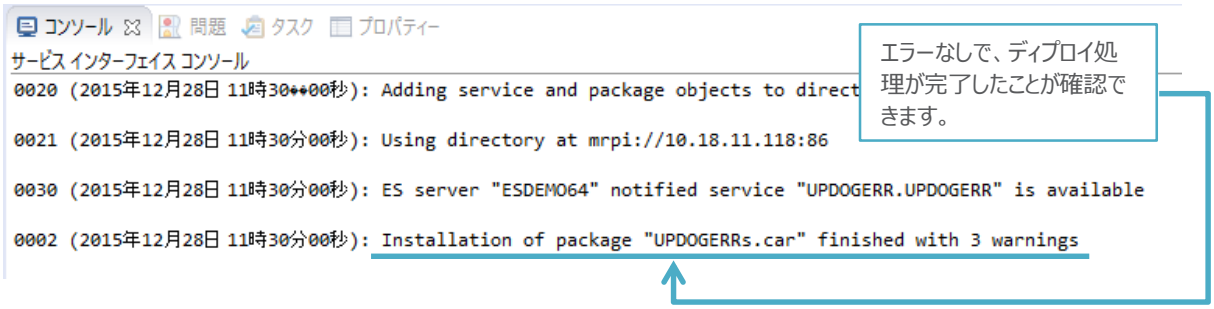
⑮ [OK] ボタンを押下し、設定画面を閉じる

5) 作成した COBOL サービスを Enterprise Server へディプロイ

COBOL エクスプローラにて作成したサービスを右クリックし [ディプロイ] を選択

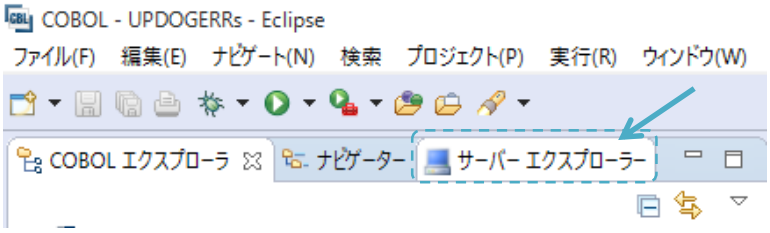


処理の経過はコンソールビューにて確認ができます：

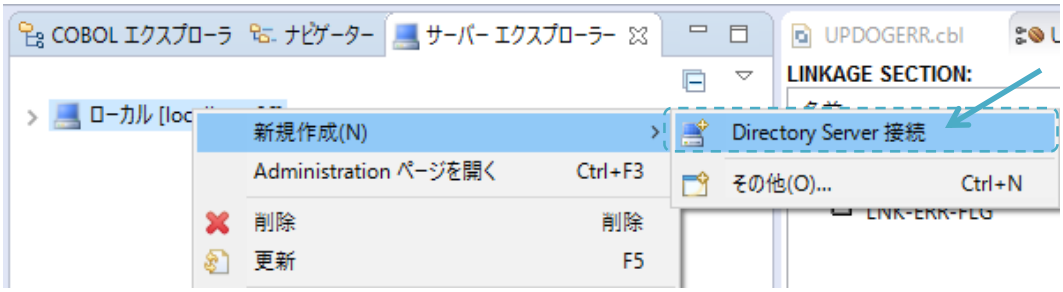


6) Enterprise Server Administration Console 画面よりディプロイが正常に処理されたことを確認

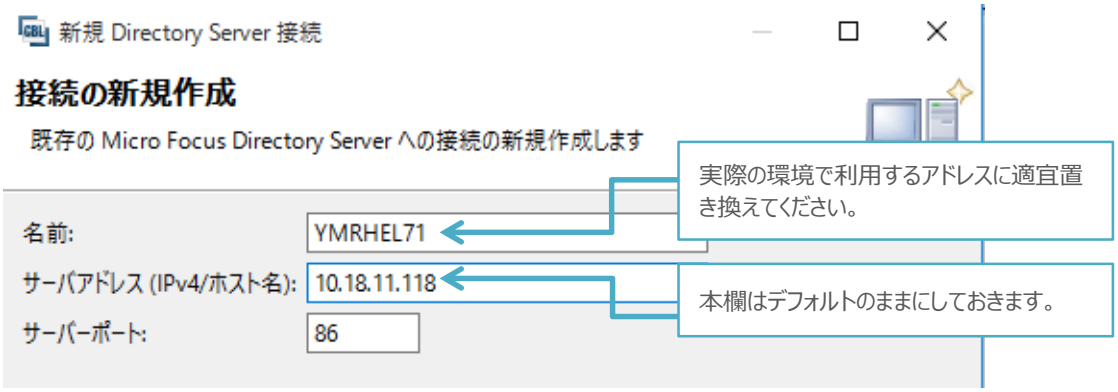
① [サーバーエクスプローラー] タブをクリック



② [ローカル] を右クリックし [新規] > [Directory Server 接続] を選択

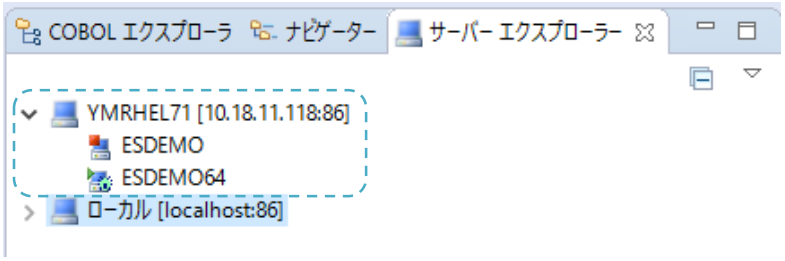


③ [名前] 欄には Linux Server を識別する任意の名称を、[サーバアドレス] 欄には Linux Server のアドレスを入力

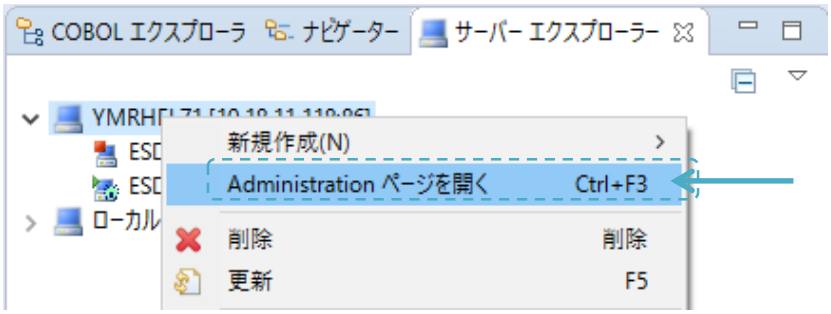


④ [終了] ボタンを押下

[終了] ボタン押下後のイメージ：

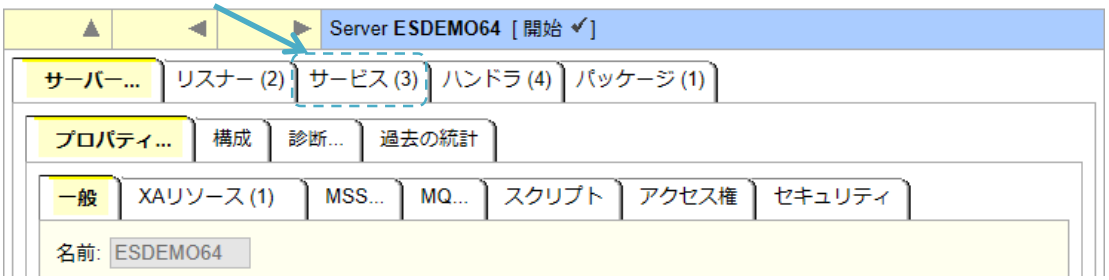


⑤ 追加したサーバを右クリックして [Administration ページを開く] を選択

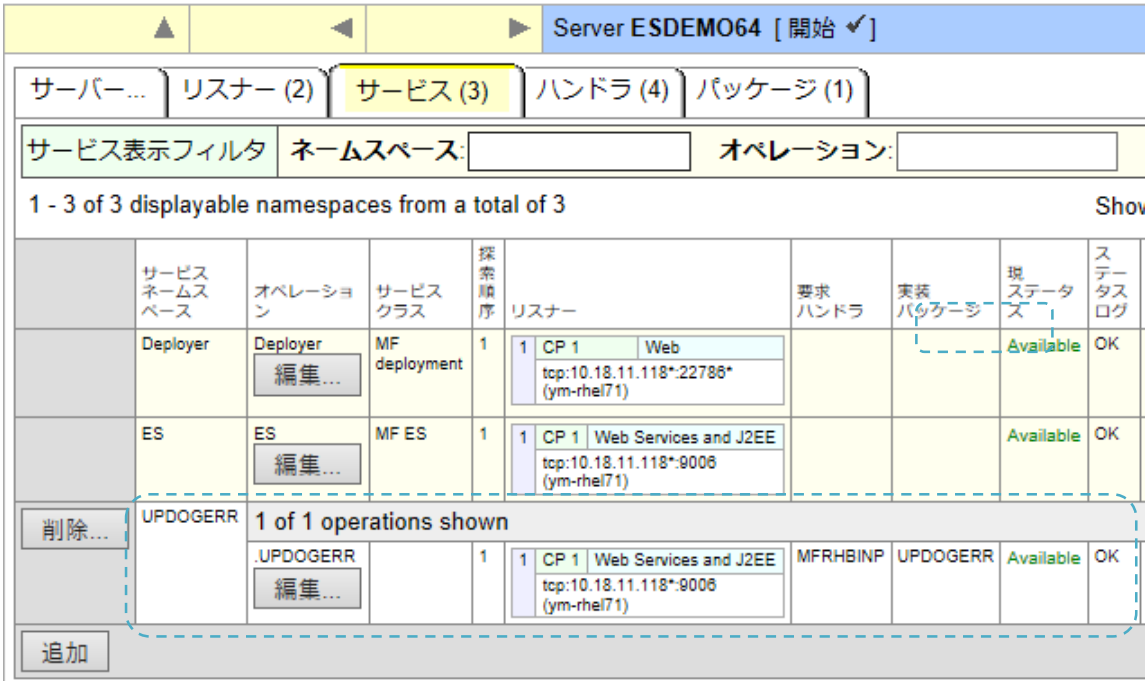


⑥ Enterprise Server Administration 画面にて対象の Enterprise Server の行中の [編集] ボタンを押下

⑦ [サービス] タブをクリック



追加したサービスがディプロイされ、正常ステータスで稼働中であることが確認できます：

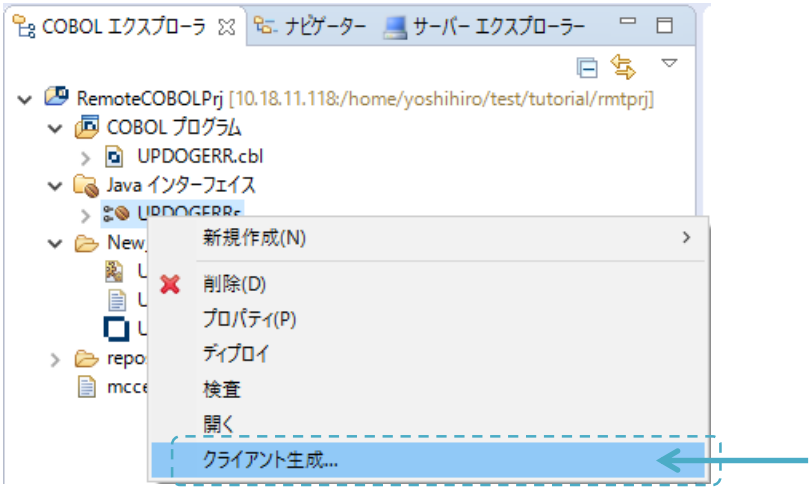


4.5 Java EE クライアントアプリケーションを WebLogic Server にディプロイ

Windows 端末上での作業

1) 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

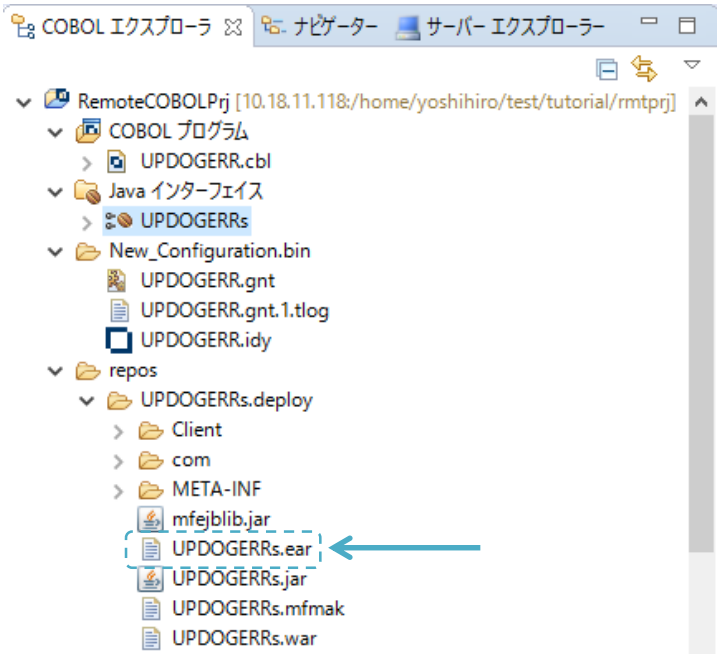
COBOL エクスプローラにて Java インターフェイスを右クリックして [クライアント生成] を選択



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：



Linux Server 上での作業

2) Linux Server にログイン

3) Visual COBOL の環境設定スクリプト $COBDIR/bin/cobsetenv を実行

4) $DOMAIN_HOME/lib に
$COBDIR/javaee/javaee6/oracleweblogic1211/mfconnector.jar 及び
Log4j 1.2.12 以降の jar[10]
を追加

5) 対象の WebLogic サーバを起動[11]

① startWebLogic.sh を実行し管理サーバを起動

出力例：

```
$ $DOMAIN_HOME/bin/startWebLogic.sh
                                                    :
<2015/12/28 15 時 27 分 56 秒 JST> <Notice> <Server> <BEA-002613> <チャネル"Default[1]"は、現在 10.18.11.118: 7001 でプロトコル iiop, t3, ldap, snmp, http をリスニングしています。>
<2015/12/28 15 時 27 分 56 秒 JST> <Notice> <Server> <BEA-002613> <チャネル"Default[2]"は、現在 0:0:0:0:0:0:0:1%lo: 7001 でプロトコル iiop, t3, ldap, snmp, http をリスニングしています。>
<2015/12/28 15 時 27 分 56 秒 JST> <Notice> <WebLogicServer> <BEA-000360> <サーバーが RUNNING モードで起動しました。>
<2015/12/28 15 時 27 分 56 秒 JST> <Notice> <WebLogicServer> <BEA-000365> <サーバー状態が RUNNING に変化しました。>
```

② startNodeManager.sh を実行し Node Manager を起動

出力例：

```
$ $DOMAIN_HOME/bin/startNodeManager.sh
                                                    :
<2015/12/28 15 時 33 分 43 秒 JST> <INFO> <base_domain> <AppSvr01> <サーバー'AppSvr01'の状態が FORCE_SHUTTING_DOWN から SHUTDOWN へ調整されました>
<2015/12/28 15 時 33 分 44 秒 JST> <INFO> <ポート 5556、ホスト localhost/127.0.0.1 でセキュア・ソケット・リスナーが開始しました>
```

[10] mfconnector.jar は Log4j 1.2.12 以降で追加された機能を利用します。しかし、WebLogic にビルドインで組み込まれる Log4j は 1.2.8 となるため、本手順のようにして依存を解決させます。本例では 1.2.17 jar「log4j-1.2.17.jar」を利用しました。

[11] 本書で示すのはあくまでも起動方法の一例です。詳細については Oracle 社のマニュアル等を参照してください。

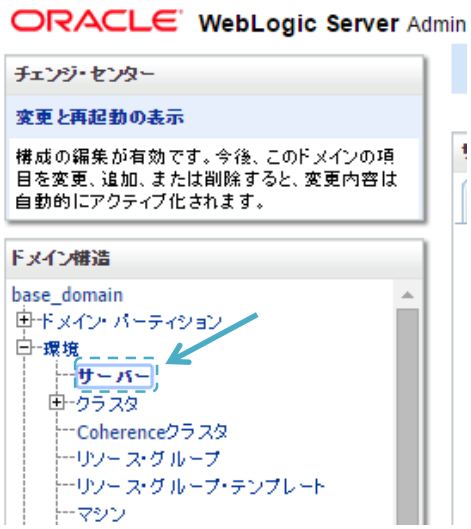
③ WebLogic の管理コンソールへログイン

デフォルトから変更していない場合は、web ブラウザで

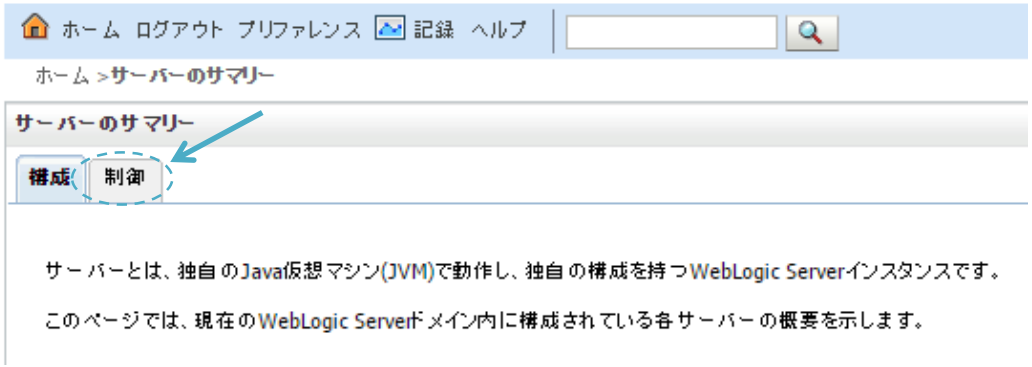
http://＜Linux サーバのアドレス＞:7001

へアクセスします。

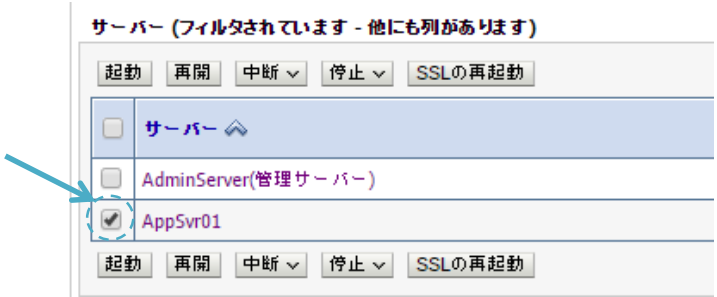
④ [base_domain] > [環境] > [サーバー] へとナビゲート



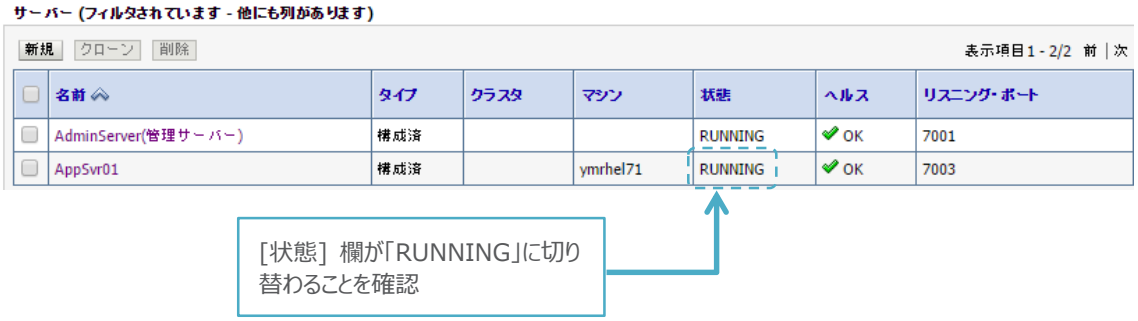
⑤ [制御] タブをクリック



⑥ 対象のサーバにチェックを入れ [起動] ボタンを押下(本例では AppSvr01 を起動します)



⑦ 再度 [base_domain] > [環境] > [サーバー] へとナビゲートし、サーバの状態を確認



6) 別のターミナルにて Linux Server へログイン

7) Visual COBOL の環境設定スクリプト $COBDIR/bin/cobsetenv を実行

8) リソースアダプタに埋め込まれたポート番号をデフォルトの「9003」から 4.2 11) で指定した「9006」へ変更

① root 権限を持ったユーザへ切り替え

② $COBDIR/javaee へ移動

```
# cd $COBDIR/javaee
#
```

③ COBOL Resource Adapter utility を使ってポイントするポート番号を変更

```
#bash ravaluesupdater.sh
Your available application servers are:
ibmwebsphere7
jboss5
oracleweblogic10
ibmwebsphere8
ibmwebsphere85
jboss6
jboss7
oracleweblogic1211
Please enter the application server you would like to update: oracleweblogic1211
Your available resource adapters are:
mfcobol-localtx.rar
mfcobol-notx.rar
mfcobol-xa.rar
Please enter the resource adapter you would like to update: mfcobol-xa.rar
ServerHost is currently set to: localhost (Default: localhost)
Would you like to change the value of ServerHost?  (y/n/reset to default x to exit)
n
ServerPort is currently set to: 9003 (Default: 9003)
Would you like to change the value of ServerPort?  (y/n/reset to default x to exit)
y
Please enter the new value:
9006
Trace is currently set to: false (Default: false)
Would you like to change the value of Trace?  (y/n/reset to default x to exit)
x
Any changes have already been saved. Are you sure you want to exit? Please enter y to confirm:
y
#
```

WebLogic 12c 向けの mfcobol-xa.rar を指定します。

ポート番号を 9006 へ変更します。

④ 一般ユーザへ戻る

9) Enterprise Server のリソースアダプタを WebLogic へディプロイ

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

```
$ export WL_USER=weblogic
$ export WL_PASSWD=weblogic123
$ export NAME=mfcobol-xa
$ export FULL_PATH_TO_THE_RAR_FILE=$COBDIR/javaee/javaee6/oracleweblogic1211/mfcobol-xa.rar
$ export TARGETSVR=AppSvr01
$
```

実際の環境で構成したパスワードに適宜置き換えてください。

ディプロイするリソースアダプタ

ディプロイ先のサーバ(本例では 5) で起動した AppSvr01 を指定しています。)

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

出力例：

```
$ java -classpath $WL_HOME/server/lib/weblogic.jar weblogic.Deployer -username $WL_USER -passwo
rd $WL_PASSWD -name $NAME -deploy $FULL_PATH_TO_THE_RAR_FILE -targets $TARGETSVR
weblogic.Deployer がオプション -username weblogic -name mfcobol-xa -deploy /opt/mf/VC23HF1/javae
e/javaee6/oracleweblogic1211/mfcobol-xa.rar -targets AppSvr01 を指定して呼び出されました
<2015/12/28 16 時 08 分 40 秒 JST> <Info> <J2EE Deployment SPI> <BEA-260121> <アプリケーション mfc
obol-xa [アーカイブ: /opt/mf/VC23HF1/javaee/javaee6/oracleweblogic1211/mfcobol-xa.rar]の deploy
操作を AppSvr01 に初期化しています。>
タスク 0 が開始されました: [Deployer:149026]AppSvr01 上のアプリケーション mfcobol-xa をデプロイ。
タスク 0 完了: [Deployer:149026]AppSvr01 上のアプリケーション mfcobol-xa をデプロイ。
ターゲットの状態: サーバー AppSvr01 で deploy 完了

$
```

10) リソースアダプタがディプロイされたことを確認

① WebLogic Server Administration Console へログイン

② [デプロイメント] をクリック

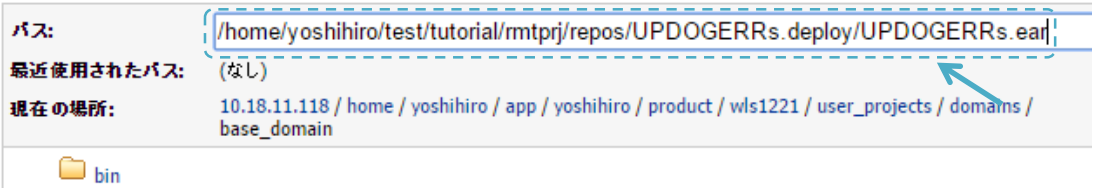
③ [デプロイメント] 表にてディプロイしたファイルがリソースアダプタとして認識されていることを確認

11) 1) で生成したスタブクライアントアプリケーションのディプロイ

① WebLogic Server Administration Console のトップ画面にて再度 [デプロイメント] をクリック

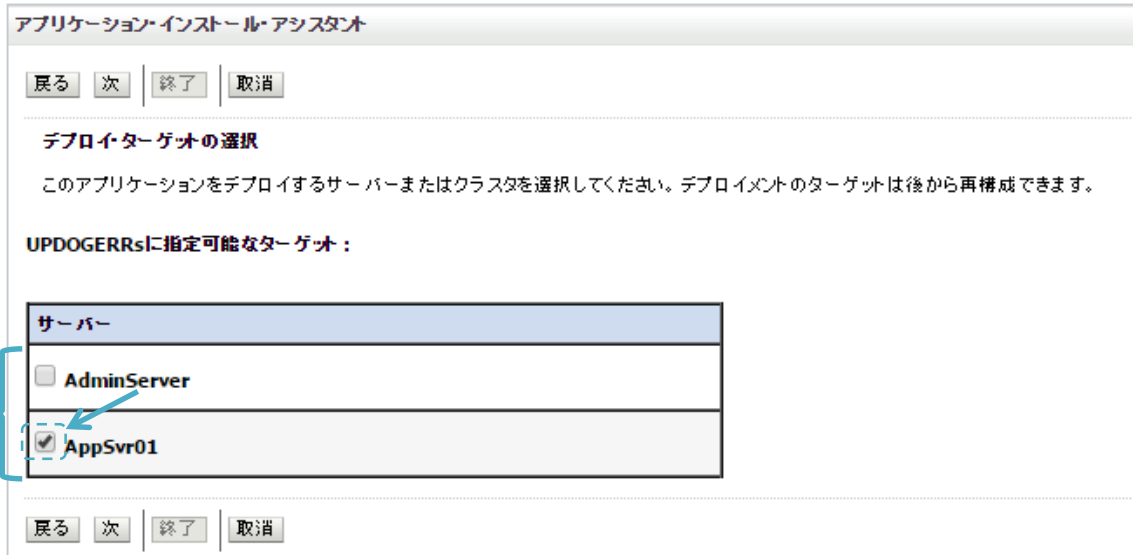
② [インストール] ボタンを押下

③ [パス] 欄に１）で生成した .ear にアーカイブされたアプリケーションを指定し [次] ボタンを押下

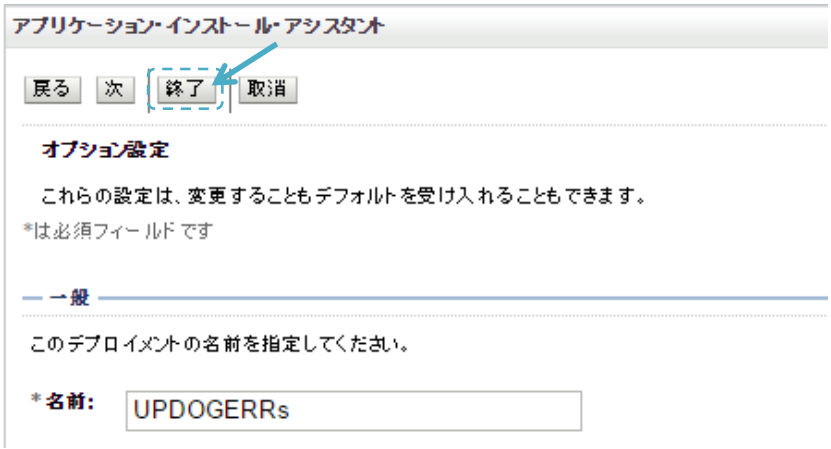


④ [このデプロイメントをアプリケーションとしてインストールする] を選択し [次] ボタンを押下

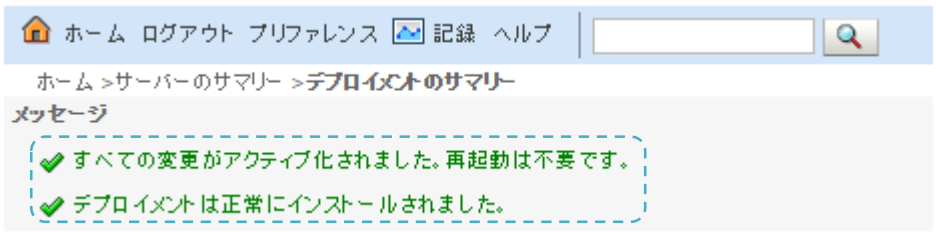
⑤ [デプロイ・ターゲットの選択] 画面ではディプロイ先のサーバにチェックを入れ、[次] ボタンを押下



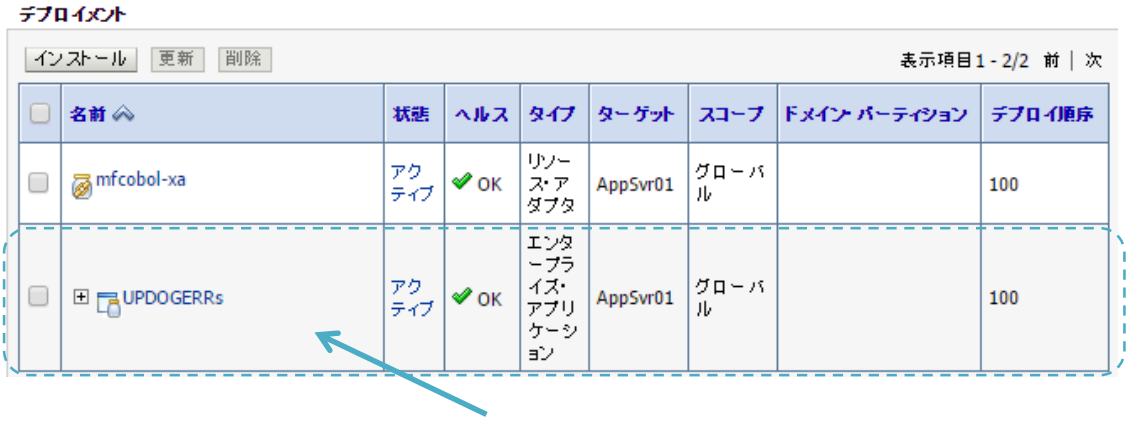
⑥ オプション設定の画面ではデフォルトの状態のまま [終了] ボタンを押下



インストールが正常に処理されると、その旨のメッセージが [メッセージ] 欄に出力されます：



デプロイメントの表からもインストールされたことを確認できます：



4.6 Java EE クライアントアプリケーションより COBOL アプリケーションをテスト呼び出し

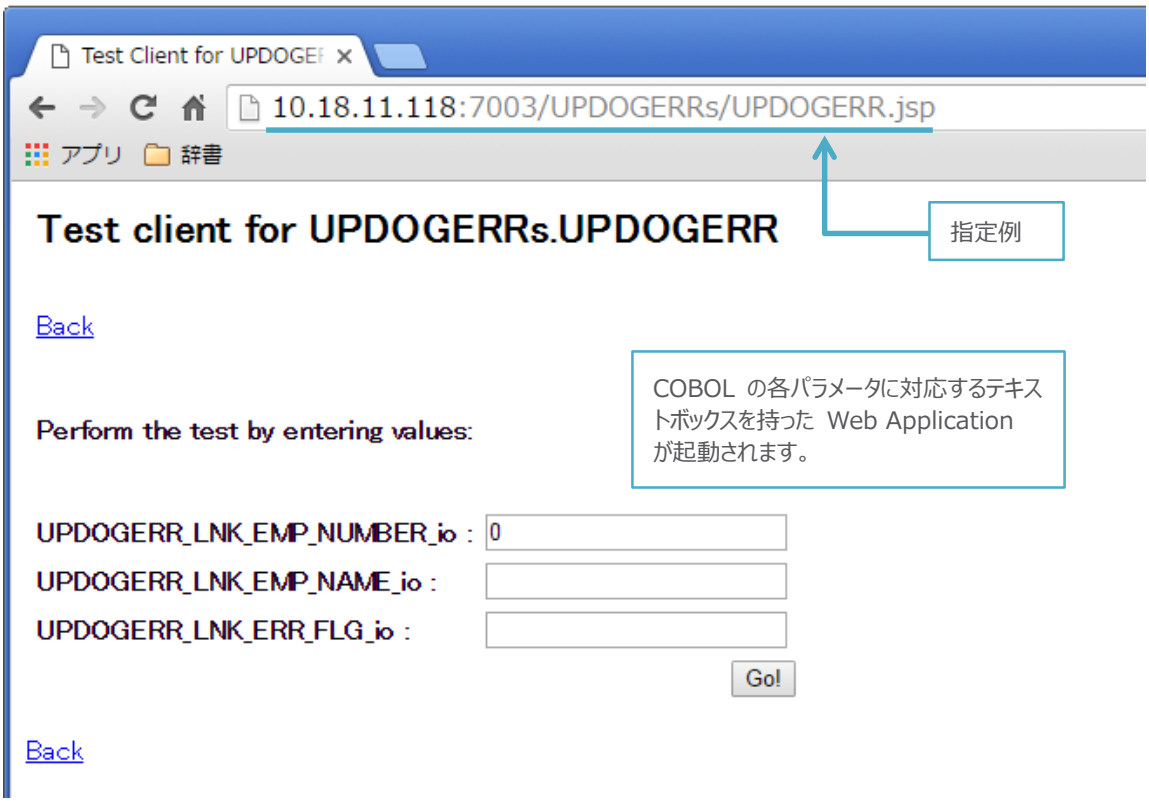
Windows 端末上での作業

1) Java EE クライアントアプリケーションの入口画面を表示

① ブラウザを起動

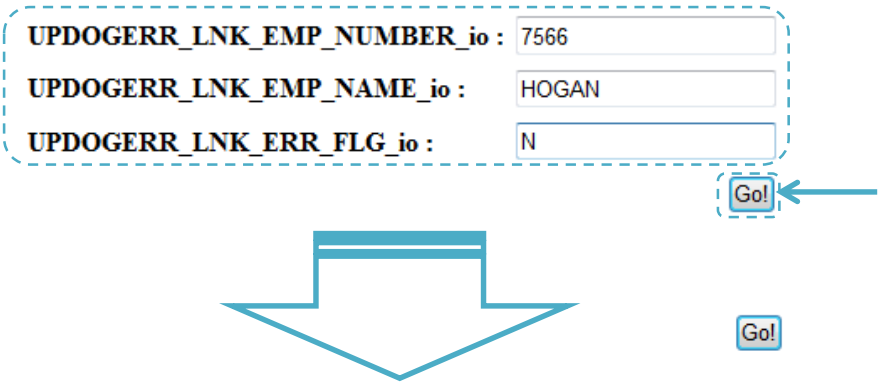
② **<Linux Server のアドレス>:<WebLogic のポート番号>/<デプロイメント名>/<オペレーション名>.jsp**
の形式でアプリケーションのメインページのアドレスをアドレスバーに入力し、Enter を打鍵

呼出し後の画面イメージ：



2) COBOL 内でエラーを発生させないパターンでアプリケーションを実行

[UPDOGERR_LNK_EMP_NUMBER_io] 欄には「**7566**」を
[UPDOGERR_LNK_EMP_NAME_io] 欄には**任意の名前**を
[UPDOGERR_LNK_ERR_FLG_io] 欄には「**N**」を
を入力し [GO] ボタンを押下



Result:

Variable	Value
Result	HOGAN

Back

UPDATE 文の実行 →
SELECT 文で変更したレコードを取得 →
取得したデータを LINKAGE パラメータに MOVE

というフローを経た後に Java EE クライアント側に戻された値がこのように表示されます。

Linux Server 上での作業

3) COBOL アプリケーションが操作したレコードを SQL*Plus で確認

```
SQL> SELECT ENAME FROM EMP_WK WHERE EMPNO=7566;

ENAME
----------
HOGAN

SQL>
```

COBOL のトランザクションが確定され、このように別セッションから見ても更新が反映されていることを確認できます。

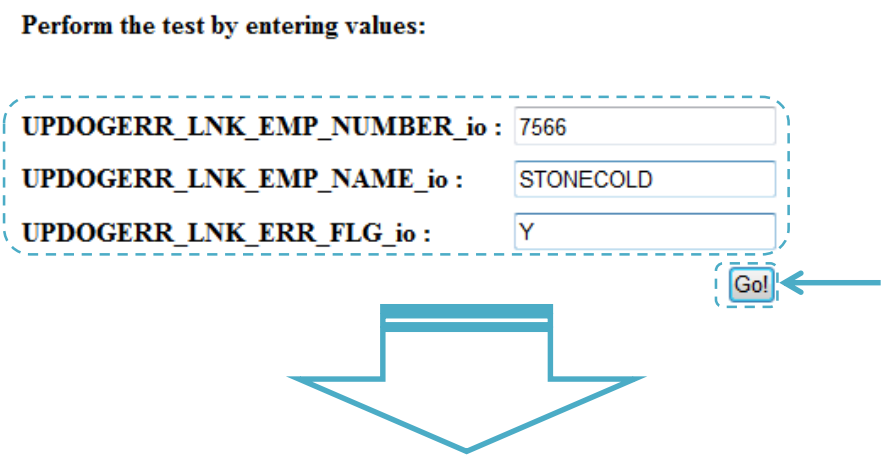
4) COBOL 内でエラーを発生させないパターンでアプリケーションを実行

[UPDOGERR_LNK_EMP_NUMBER_io] 欄には「**7566**」を

[UPDOGERR_LNK_EMP_NAME_io] 欄には**2）の入力と異なる名前**を

[UPDOGERR_LNK_ERR_FLG_io] 欄には「**Y**」を

を入力し [GO] ボタンを押下




```
javax.ejb.EJBException: UPDOGERR threw a resource Exception
        at com.mypackage.UPDOGERRs.UPDOGERRsBean.UPDOGERR(Unknown Source)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:497)
        at com.bea.core.repackaged.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:310)
        at com.bea.core.repackaged.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:182)
        at com.bea.core.repackaged.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:149)
        at com.oracle.pitchfork.intercept.MethodInvocationInvocationContext.proceed(MethodInvocationInvocationContext.java:101)
        at com.oracle.pitchfork.intercept.JeeInterceptorInterceptor.invoke(JeeInterceptorInterceptor.java:101)
        at com.bea.core.repackaged.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:171)
        at com.oracle.pitchfork.intercept.MethodInvocationInvocationContext.proceed(MethodInvocationInvocationContext.java:101)
        at org.jboss.weld.ejb.AbstractEJBRequestScopeActivationInterceptor.aroundInvoke(AbstractEJBRequestScopeActivationInterceptor.java:73)
        at org.jboss.weld.ejb.SessionBeanInterceptor.aroundInvoke(SessionBeanInterceptor.java:52)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:497)
        at com.oracle.pitchfork.intercept.JeeInterceptorInterceptor.invoke(JeeInterceptorInterceptor.java:94)
        at com.bea.core.repackaged.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:171)
        at com.bea.core.repackaged.springframework.aop.support.DelegatingIntroductionInterceptor.doProceed(DelegatingIntroductionInterceptor.java:131)
        at com.bea.core.repackaged.springframework.aop.support.DelegatingIntroductionInterceptor.invoke(DelegatingIntroductionInterceptor.java:119)
        at com.bea.core.repackaged.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:171)
        at com.bea.core.repackaged.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:215)
        at com.sun.proxy.$Proxy174.UPDOGERR(Unknown Source)
```

<中略>

```
        at weblogic.work.ExecuteThread.run(ExecuteThread.java:346)
Caused by: javax.resource.spi.EISSystemException: CobolException Recoverable:
目的コード エラー: ファイル '/opt/mf/VC23HF1/deploy/UPDOGERRs.JAXIalrP/UPDOGERR.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 (/home/yoshihiro/test/tutorial/rmtprj/UPDOGERR.cbl 内, 57 行) executing UPDOGERR.UPDOGERR
        at com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:291)
        at com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:176)
        ... 62 more
```

[UPDOGERR_LNK_ERR_FLG_io] 欄に「Y」を入力したため、意図的に実行時エラーを発生させるロジックへ遷移したことがわかります。COBOL で実行時エラーが発生した場合はこのように例外が Java EE クライアント側に返ってきます。

Linux Server 上での作業

5) COBOL アプリケーションが操作したレコードを SQL*Plus で確認

```
SQL> SELECT ENAME FROM EMP_WK WHERE EMPNO=7566;

ENAME
----------
HOGAN

SQL>
```

COBOL アプリケーション内でエラーが発生したため、トランザクションが取り消され、前の手順で更新した値のままになっています。

Windows 端末上での作業

6) Enterprise Server の Console.log を確認し、トランザクションが取り消される前に取得したレコードの値を確認

① Enterprise Server Administration Console にて対象の Enterprise Server の行中の [編集] ボタンを押下

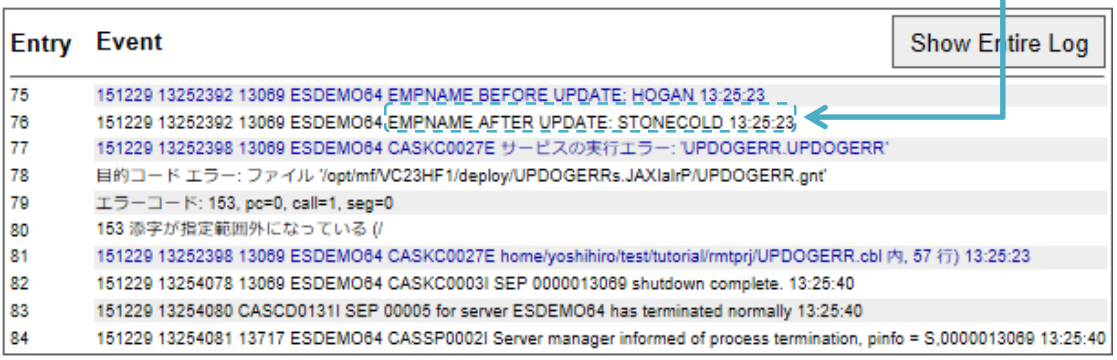
② [診断] をクリック

③ [ES コンソール] をクリック

④ ４）の作業をしていた際のログを確認

実行時エラー発生前に発行した SELECT 文では、更新した値を取得していることが確認できます。これにより、実行時エラー発生後にトランザクションが ROLLBACK されたことが改めて裏付けられます。

出力例：



4.7 Enterprise Server 配下で稼働する COBOL アプリケーションを Eclipse IDE で動的デバッグ

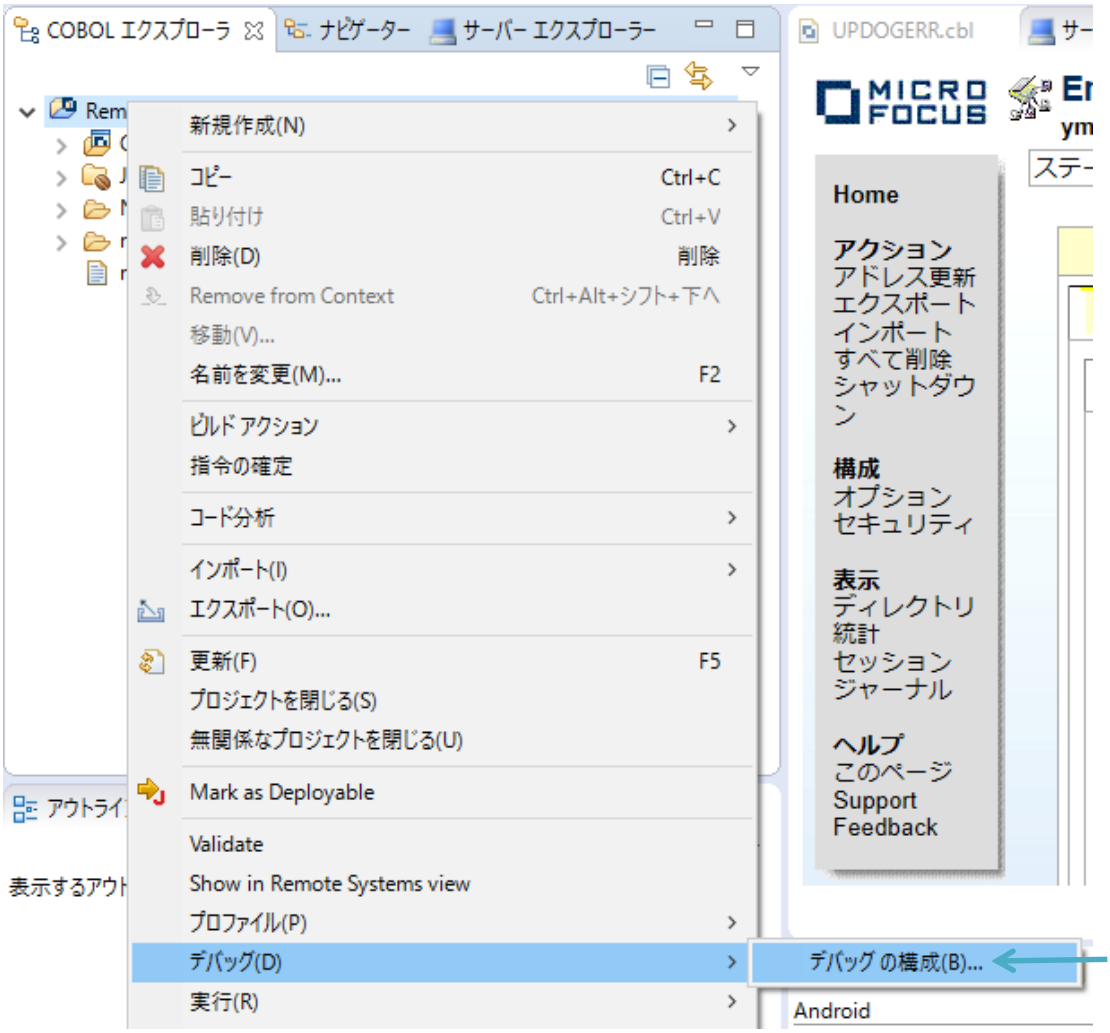
Windows 端末上での作業

1) Enterprise Server デバッグを起動

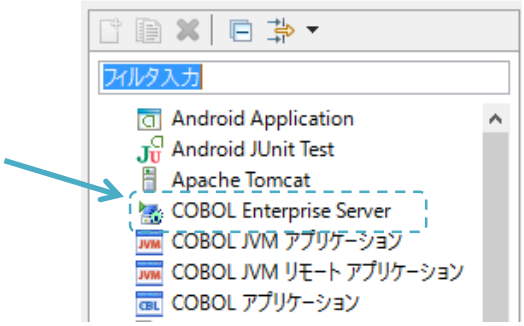
① Eclipse IDE の COBOL エクスプローラにてプロジェクトを右クリックし、

[デバッグ] > [デバッグの構成]

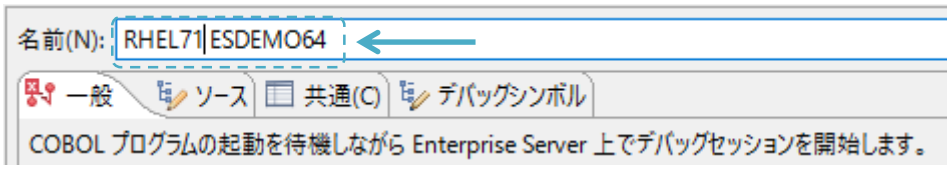
を選択



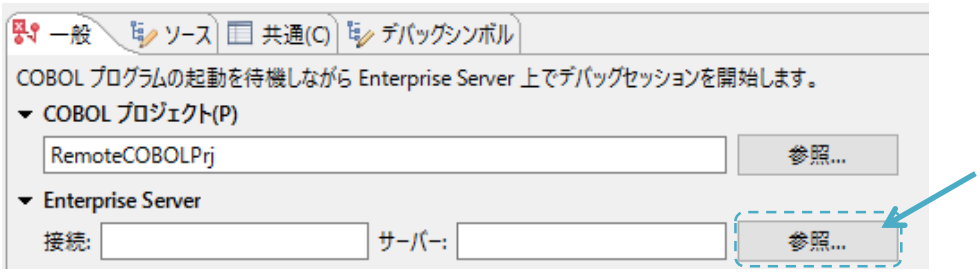
② [COBOL Enterprise Server] をダブルクリック



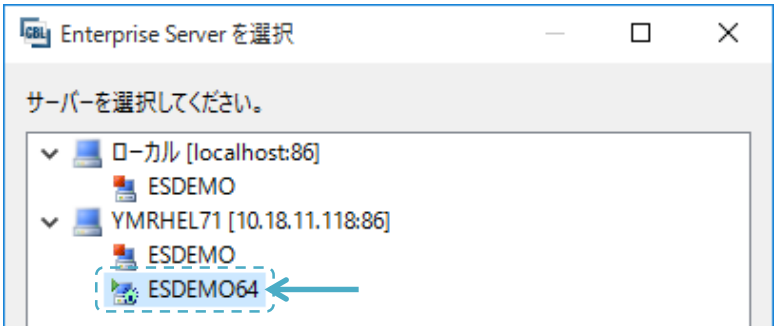
③ [名前] 欄にデバッグ構成を識別するための任意の名前を入力



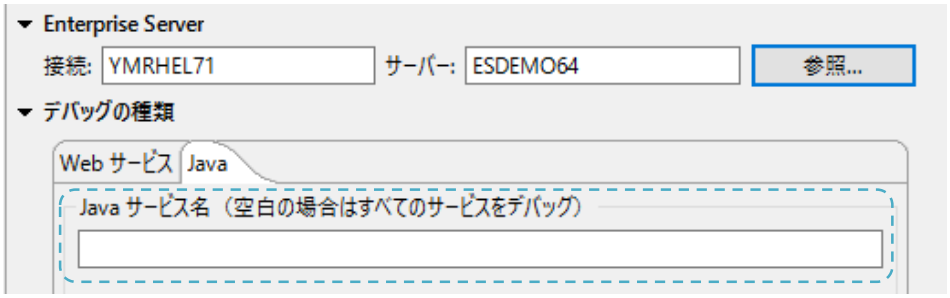
④ [Enterprise Server] 欄の [参照] ボタンを押下



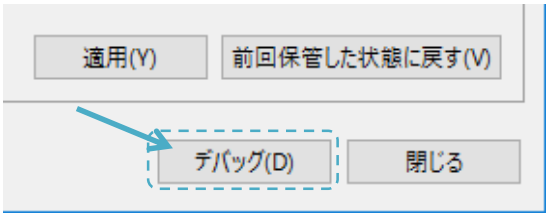
⑤ COBOL アプリケーションをディプロイした Enterprise Server を選択し [OK] ボタンを押下



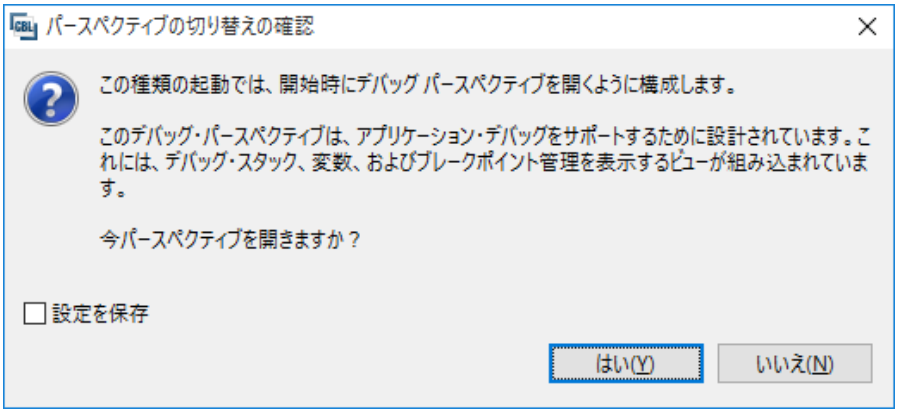
⑥ [デバッグの種類] 欄にて [Java] タブを選択し、全てのサービスがデバッグ対象になっていることを確認



⑦ [デバッグ] ボタンを押下



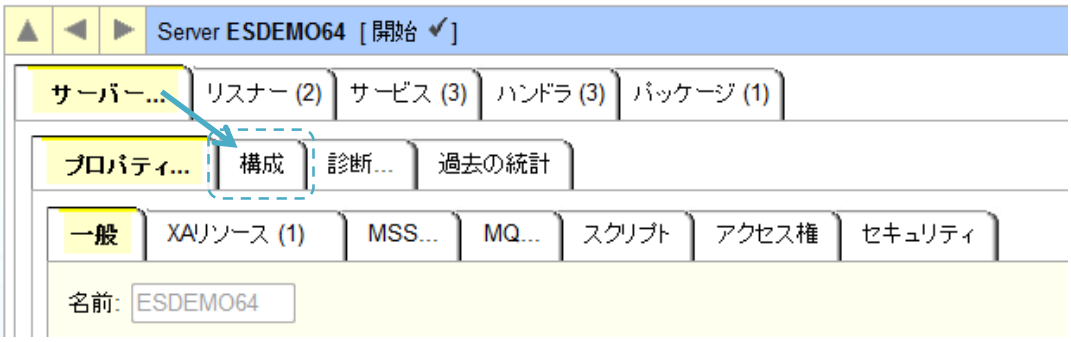
⑧ [パースペクティブの切り替えの確認] ウィンドウがポップアップされたら [はい] ボタンを押下



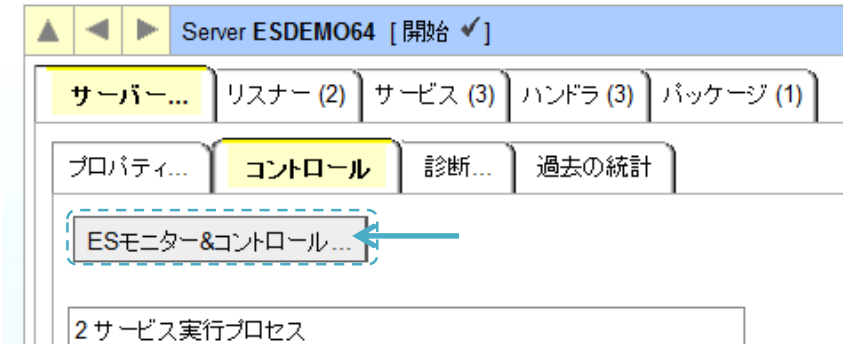
2) 動的デバッグが正しく待機状態になっていることを Enterprise Server Administration 画面より確認

① Enterprise Server Administration Console 画面にて対象の Enterprise Server の行の [編集] ボタンを押下

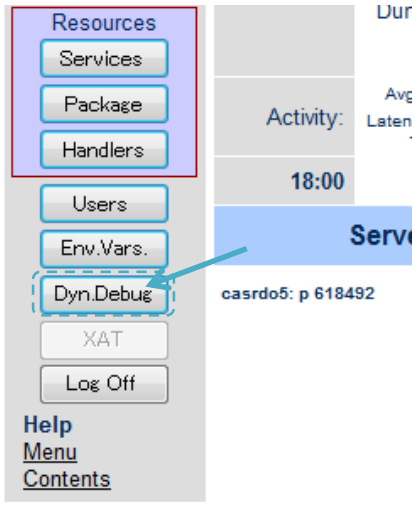
② [構成] タブをクリック



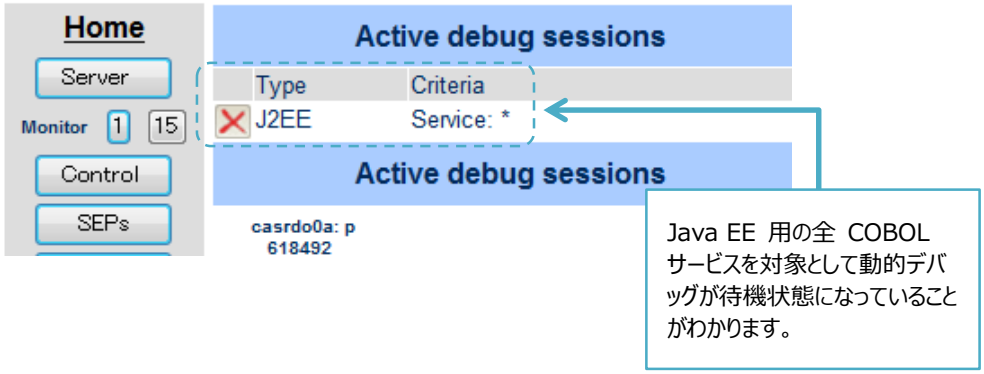
③ [ES モニター & コントロール] ボタンを押下



④ 左ペイン下方の［Dyn.Debug］ボタンを押下



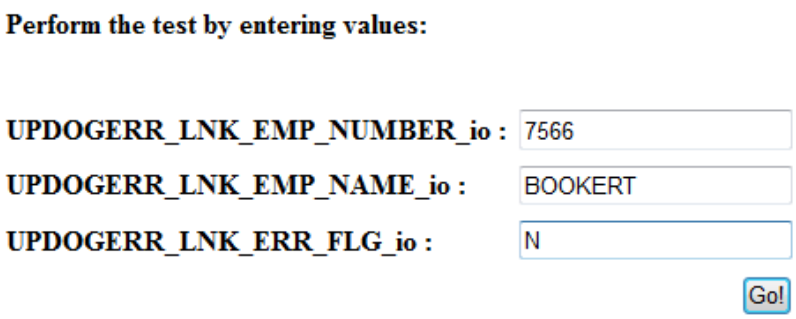
ボタン押下後の画面イメージ：



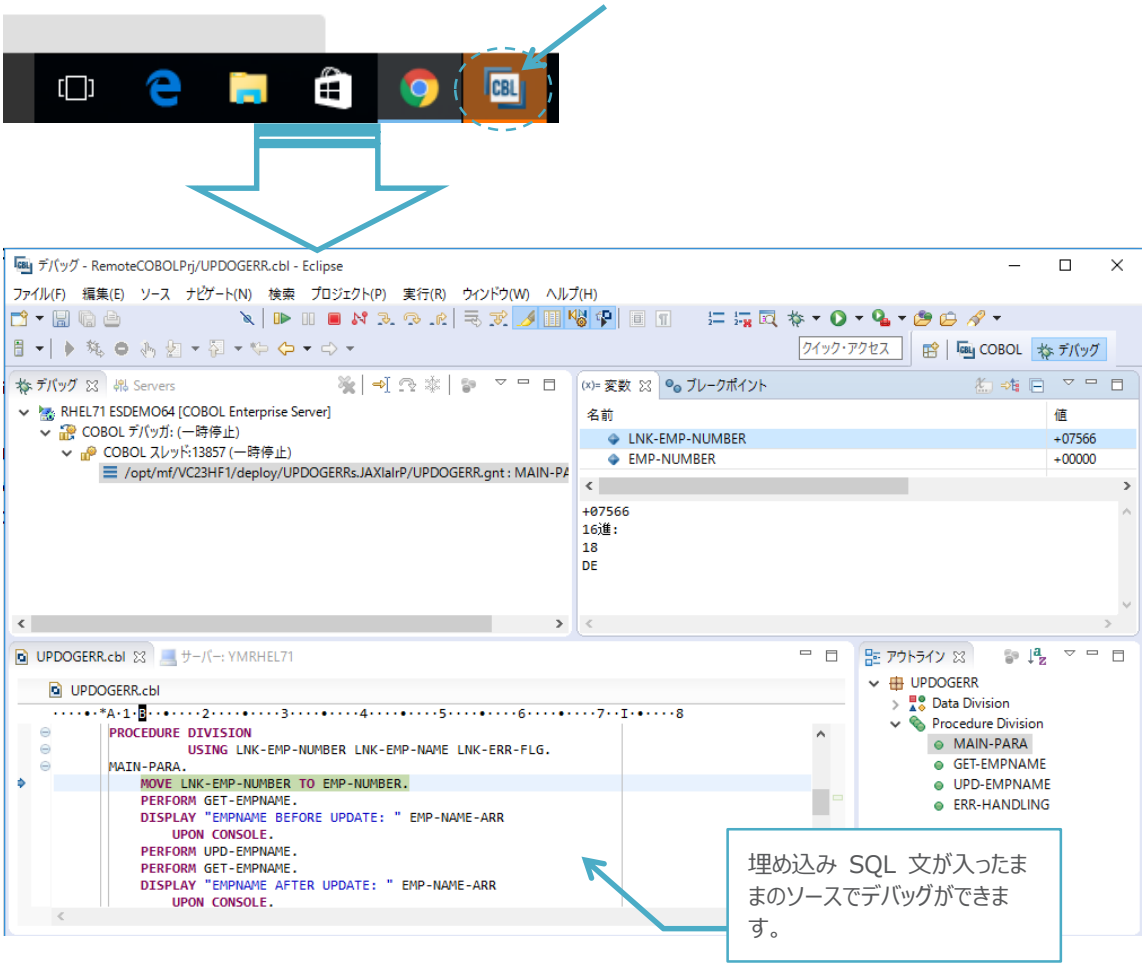
3) アプリケーションをデバッグ実行

① 前項の要領でアプリケーションのトップページにて何らかの値を入力の上［GO］ボタンを押下し、COBOL へのリクエスト処理を開始

入力例：

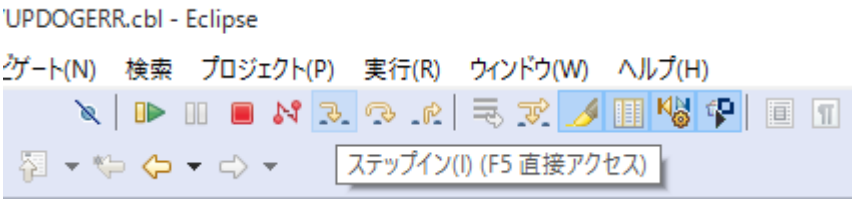


COBOL に処理が渡った時点で、Eclipse のデバッガが処理を引き込みます：



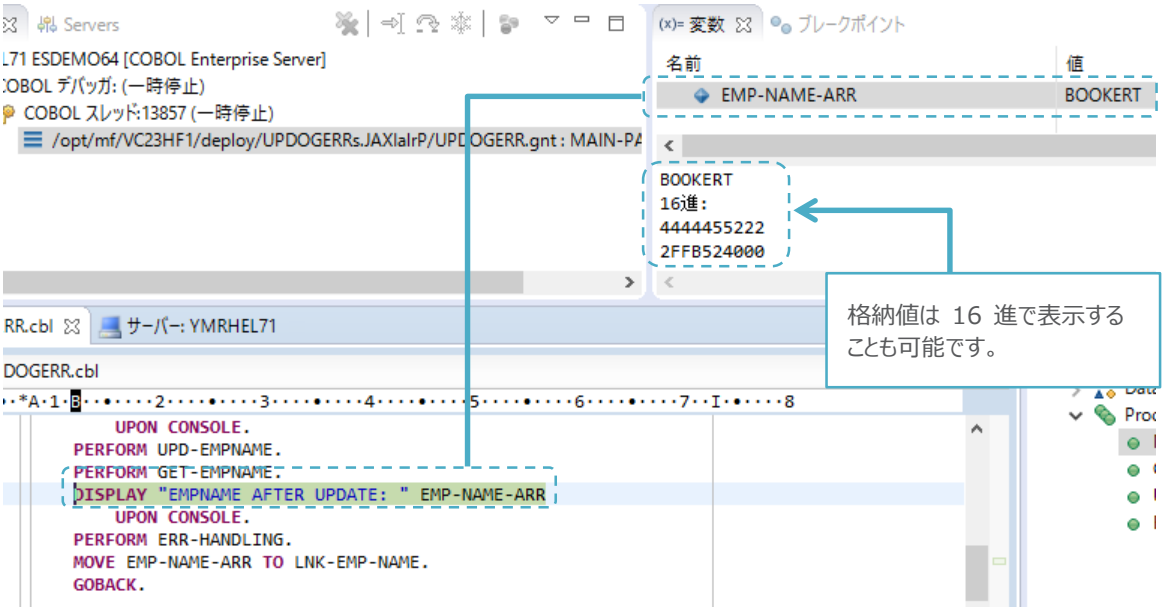
② デバッグ実行

上のツールバー中のステップインやステップオーバーのアイコンをクリックして一行ずつ処理を進められます[12]：



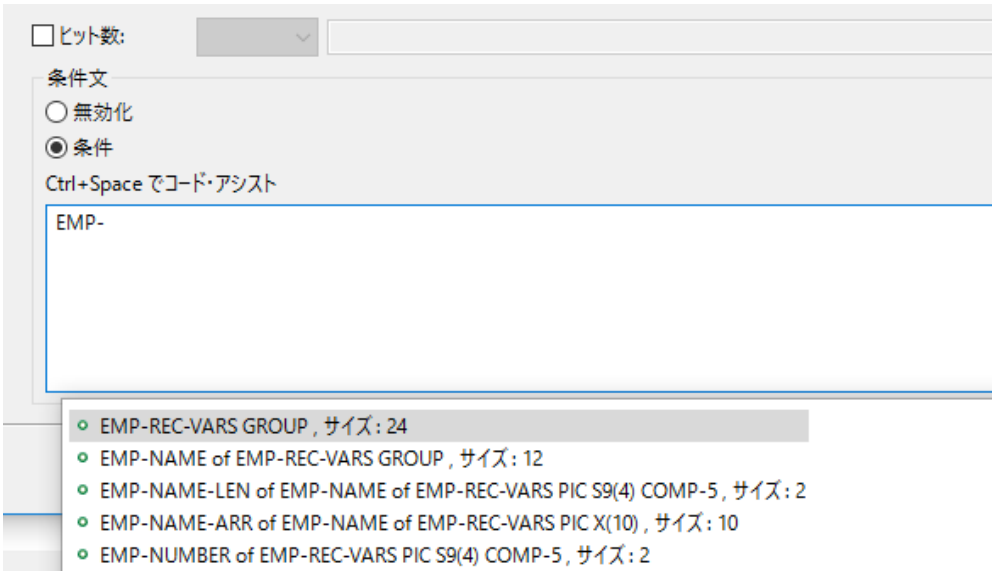
[12] ステップインを選択すると PERFORM 文や CALL 文で呼んでいる先までデバッガを進めます。一方、ステップオーバーを選択すると PERFORM 文や CALL 文の先までデバッガを進めずワンステップとして処理をします。

[変数] ビューにて実行中の COBOL 文が参照する変数の格納値を確認できます：

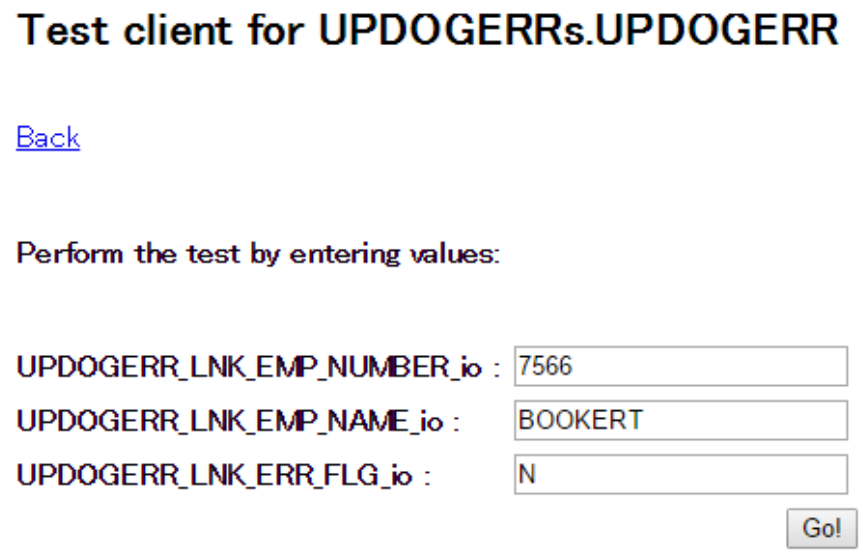


任意の箇所にブレークポイントを指定することも可能です：

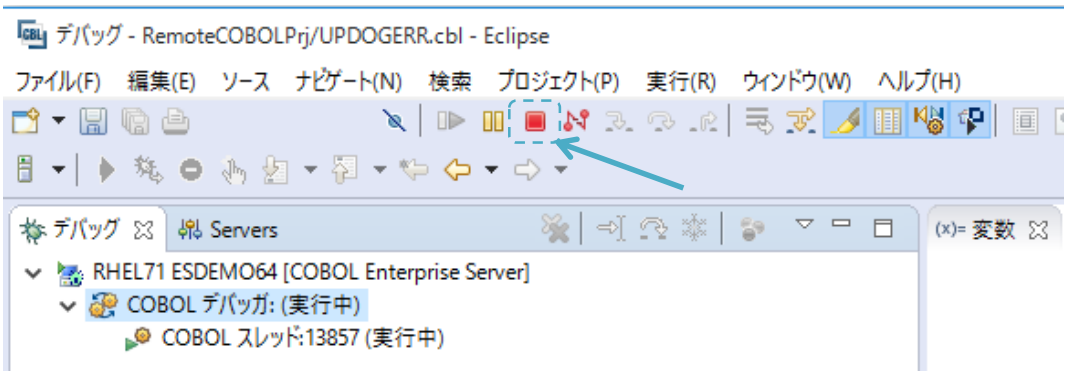
設定したブレークポイントにブレーク条件を付与することも可能です：



処理を最後まで進めるとこれまでと同様に Java EE クライアント側に値が返り、その後の処理が実行され結果をブラウザで確認することができます：



③ デバッグ機能が確認できたら Eclipse のツールバーにて［終了］アイコンをクリックし、デバッガを停止



以上でチュートリアルを終了します。