

---

# Micro Focus Visual COBOL チュートリアル

---

## JCA による JBOSS EAP 連携の設定と開発

### 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ー「Enterprise Server」は、ネイティブにコンパイルした COBOL のビジネスロジックを EJB として再利用し、J2EE クライアントから呼び出す機能および REST API を利用し Web サービスとして呼び出す機能を提供しています。EJB コンポーネントとして呼び出しを行う場合、Java アプリケーションサーバー上の J2EE クライアントは JCA の仕様にもとづいたリクエストを Enterprise Server に渡し、COBOL のビジネスロジックが処理をして結果を返します。また、Enterprise Server は、XA に準拠しているのと同じく XA に準拠しているデータベースや他のシステムと協調してトランザクション処理を行うことができます。

Micro Focus Visual COBOL の UNIX/Linux 版には、UNIX/Linux 環境へインストールし、リモート接続を可能にする Development Hub および開発クライアントとして Windows 環境へインストールする Eclipse 版のライセンスが提供されます。これにより、Windows 上の Eclipse で開発作業を行い、Linux/UNIX 上のソースコードを直接編集し、コンパイルするリモート開発機能が利用できます。

通常、Pro\*COBOL を使った Oracle 連携アプリケーションの開発を行う場合、ソースコードをプリコンパイルし、生成された COBOL のソースコードを COBOL コンパイラでコンパイルするという 2 つのステップが必要ですが、Visual COBOL は Pro\*COBOL を利用しプリコンパイルからコンパイルまでワンステップで行う COBSQL という技術を提供しています。

このドキュメントでは JBOSS EAP と Enterprise Server を JCA による連携を行い、Enterprise Server にデプロイする COBOL アプリケーションは、Oracle データベースを利用してトランザクション連携する方法を説明します。

### 2. 前提条件

本チュートリアルは、下記の環境を前提に作成されています。サポートしているプラットフォームであれば他の Linux/UNIX でも利用可能です。

#### > アプリケーションサーバー側 ソフトウェア

OS	Red Hat Enterprise Linux Server 7.3 (64bit)
COBOL 開発環境製品	Micro Focus Visual COBOL 4.0J Development Hub (PU5 適用版)
AP サーバー製品	Red Hat JBoss EAP 6.4.21
Oracle DBMS 製品	Oracle 12c Release 2 クライアント (12.2.0.1.0) (Oracle Pro* COBOL 含む)
Oracle JDK	Java SE Development Kit 8, Update 171

> 開発クライアント ソフトウェア

OS	Windows Server 2016 Standard Edition (64bit)
COBOL 開発環境製品	Micro Focus Visual COBOL 4.0J for Eclipse (PU5 適用版)

> データベースサーバー ソフトウェア

OS	Windows Server 2016 Standard Edition (64bit)
Oracle DBMS 製品	Oracle 12c Release 2 (12.2.0.1.0)

※事前に Oracle 提供の Scot のサンプルスキーマとデータが設定済み

### 3. チュートリアル手順の概要

1. COBOL 環境変数の設定
2. Oracle 関連の設定
3. Oracle 用 XA スイッチモジュールの作成
4. リモート開発用デーモンの起動
5. Micro Focus Directory Server の起動
6. リソースアダプターの編集
7. リソースアダプターを JBoss EAP 6.4 にデプロイ
8. JBoss EAP 6.4 の設定と起動
9. Windows クライアントでの開発作業
10. 64bit Enterprise Server のインスタンス作成と起動
11. コンパイルした COBOL アプリケーションを Enterprise Server にデプロイ
12. テスト用クライアントアプリケーションを JBOSS EAP 6.4 にデプロイ
13. テスト用アプリケーションを経由して COBOL アプリケーションを呼び出し
14. 全てのインスタンスの停止

### 3.1 COBOL 環境変数の設定

インストールした製品を COBOL 実行環境に設定するため環境変数を設定します。製品ディレクトリの bin ディレクトリに cobsetenv が用意されていますので、これを一般ユーザーで実行します。

コマンド例) ./opt/mf/VC40PU5/bin/cobsetenv

実行すると環境変数 COBDIR にインストールした製品パス設定されます。

### 3.2 Oracle 関連の設定

#### 1) Oracle 用オプションファイルの作成

Oracle のライブラリをリンクするためにオプションファイルを生成します。この作業を行うためには Oracle 関連の環境変数が適切に設定されている必要があります。

##### ① set\_cobopt\_oracle の実行

```
$ $COBDIR/src/oracle/set_cobopt_oracle
Set COBOPT to /home/oracle/cobopt.ora before starting the RDO daemon.
Ensure that you specify both the main entry point name and the
output name when linking your user application.
From the command-line, you can do this by passing
entry_point -o output_name to cob.
$
```

### 3.3 Oracle 用 XA スイッチモジュールの作成

トランザクション処理を伴うデータベース I/O を Enterprise Server 経由で行うには XA スイッチモジュール経由でデータベースと接続することになります。このチュートリアルでは Oracle を使用するので Oracle 用の XA スイッチモジュールを root ユーザーで作成します。

#### 1) XA リソースのコピー

ビルドを行うため、インストールディレクトリ配下の \$COBDIR/src/enterpriseserver/xa をディレクトリごと書き込み権限があるパスへコピーします。

コピー元パス例 : \$COBDIR/src/enterpriseserver/xa

コピー先パス例 : /home/tarot/xa

\$ cp \$COBDIR/src/enterpriseserver/xa/\* /home/tarot/xa

#### 2) XA スイッチモジュールのビルド準備

生成する環境の設定を行います。

##### ① COBOL 作業モードの設定

接続するデータベースのビット数に合わせた数値を指定します。XA スイッチモジュールはこの設定値に沿って生成されます。cobmode コマンドまたは環境変数 COBMODE を使用して設定します。

64 ビット設定例) export COBMODE=64

### 3) XA スイッチモジュールのビルド実行

- ① 書き込み権限のあるコピー先パスへ移動します。

コマンド例) `cd /home/tarot/xa`

- ② Oracle を使用する場合は下記コマンドを実行し、XA スイッチモジュールを生成します。

コマンド) `./build ora`

ビット数ごとに静的と動的登録用の 2 ファイルが生成されます。

ESORAXA.so	32-bit	static
ESORAXA64.so	64-bit	static
ESORAXA_D.so	32-bit	dynamic
ESORAXA64_D.so	64-bit	dynamic

## 3.4 リモート開発用デーモンの起動

root ユーザーで `startdodaemon` コマンドを実行します。

コマンド例)

`$COBDIR/remotedev/startdodaemon`

Starting RSE daemon...

## 3.5 Micro Focus Directory Server の起動

root ユーザーで `mfds` (Micro Focus Directory Server) コマンドを実行します。使用する環境によって、明示的に 32 bit 環境用に `mfds32` コマンド、64 bit 環境用に `mfds64` コマンドを実行することもできます。

コマンド例) `mfds &`

上記 `&` を付加すると、前項の環境変数を基にバックグラウンドで `mfds` のプロセスが起動されます。

## 3.6 リソースアダプターの編集

root ユーザーのまま作業を行います。

### 1) COBOL Resource Adapter utility の実行

`$COBDIR/javaee` へ移動し、`ravaluesupdater.sh` (COBOL Resource Adapter Utility)を実行します。

例 : `bash ravaluesupdater.sh`

- ① どのアプリケーションサーバーを使用しているのかの問いには“jboss64EAP”をタイプします。

Please enter the application server you would like to update: `jboss64EAP`

- ② どのリソースアダプターを編集するのかの問いには“mfcobol-xa.rar”をタイプします。

Please enter the resource adapter you would like to update: `mfcobol-xa.rar`

- ③ サーバーホスト名を変更するかどうかの問いには“n”をタイプします。

- ④ サーバーポートの変更をするかの問いには“y”を入力し、“9006”を指定します。

- ⑤ トレースを取得するかの問いにはデフォルトである“x”を指定します。
- ⑥ 全ての変更を保存するかどうかの問いには“y”を指定して終了します。

### 3.7 リソースアダプターを JBoss EAP 6.4 ヘッドプロイ

- 1) COBOL Resource Adapter utility で編集した「mfcobol-xa.rar」をコピー

\$ \$COBDIR/javaee/javaee6/jbossEAP64 へ移動し、「mfcobol-xa.rar」ファイルを  
\$JBASS\_HOME/standalone/deployments へコピーします。

例 : cp \$COBDIR/javaee/javaee6/jbossEAP64/mfcobol-xa.rar \$JBASS\_HOME/standalone/deployments

### 3.8 JBoss EAP 6.4 の設定と起動

- 1) JBoss の設定ファイルを XA 用のリソースアダプター向けに編集

一般 ユーザーでログインし、JBoss の Standalone サーバー向け設定ファイルをエディタ等で開いて編集します。編集内容は Visual COBOL のマニュアルを参照し、「mfcobol-xa.rar」をリソースアダプターに追加します。

マニュアル参照箇所 : <https://www.microfocus.co.jp/manuals/VC40/vc40indx.html>

デプロイ > 構成および管理 > モダナイズされたアプリケーションのデプロイおよび構成 > EJB とリソース アダプターのデプロイ > EJB のデプロイ - 概要 > JBoss へのデプロイ

- 2) JBoss の起動

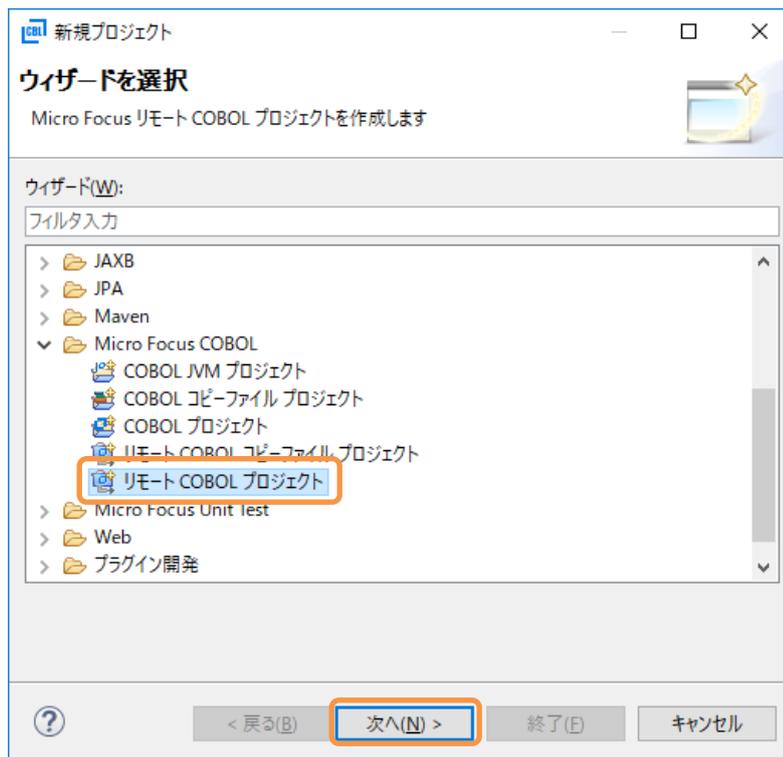
下記のコマンドを実行し JBoss EAP 6.4 を起動します。例にあるようなメッセージが表示されていれば正しく起動されリソースアダプターもデプロイされています

```
$ $JBASS_HOME/bin/standalone.sh -b 0.0.0.0 -bmanagement=0.0.0.0
=====
JBoss Bootstrap Environment
JBASS_HOME: /opt/EAP-6.4.0
JAVA: java

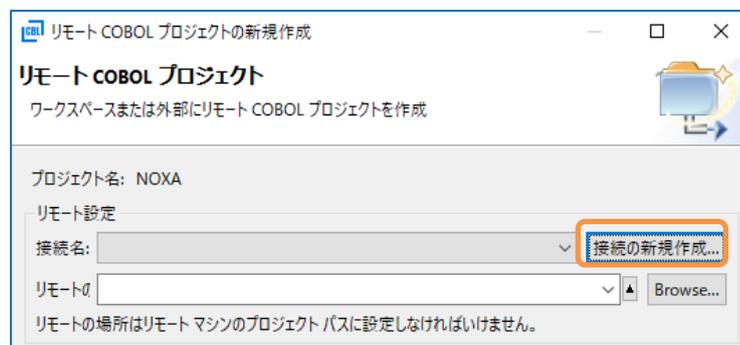
2018-08-16 10:04:41,118 INFO INFO [org.jboss.as.connector.deployment] (MSC service
thread 1-4) JBAS010401: JCA ConnectionFactory [java:/eis/MFCobol_v1.5] をバインドしまし
た。
2018-08-16 10:04:41,823 INFO [org.jboss.as.server] (ServerService Thread Pool -- 36)
JBAS015859: "mfcobol-xa.rar" (runtime-name : "mfcobol-xa.rar") をデプロイしました。
途中省略
2018-08-16 10:04:41,971 INFO [org.jboss.as] (Controller Boot Thread) JBAS015874:
JBoss EAP 6.4.21.GA (AS 7.5.21.Final-redhat-1) は 9879ms で開始しました - サービス 231 個
のうち 193 個を開始しました (59 のサービスはレイジー、パッシブ、またはオンデマンドです)。
```

### 3.9 Windows クライアントでの開発作業

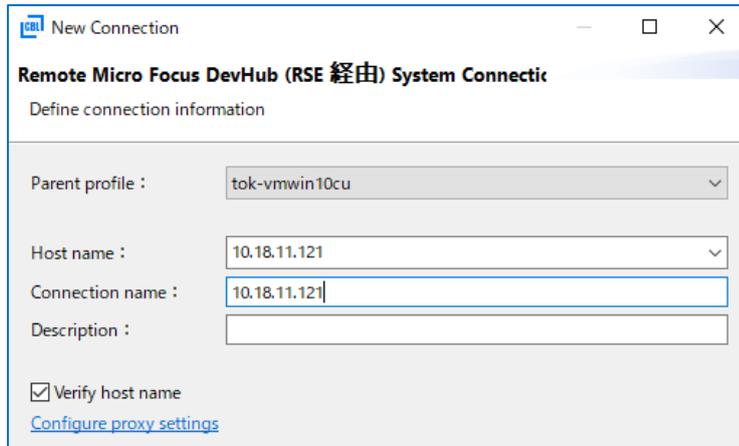
- 1) Windows 上で Linux サーバーで稼働する Enterprise Server を操作するための環境を設定
  - ① <Visual COBOL のインストールフォルダ>¥bin¥mf-client.dat をテキストエディタで開きます。
  - ② [directories] 欄に mrpi://<Linux サーバーの IP アドレス> :0 の形式で Linux サーバーの Directory Server エントリを追加します。
  
- 2) Visual COBOL for Eclipse を起動し、リモート COBOL プロジェクトを作成
  - ① [ファイル] メニュー > [新規] > [プロジェクト] > [Miro Focus COBOL] > [リモート COBOL プロジェクト] を選択します。



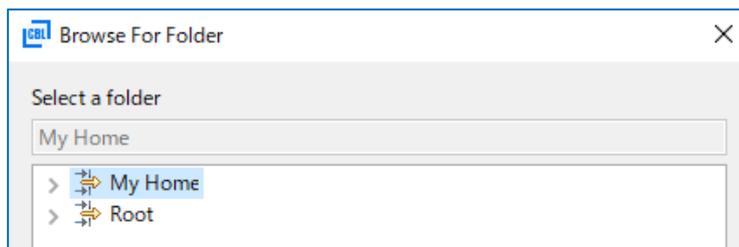
- ② プロジェクト名 "WITHXA" を指定し、[次へ] ボタンをクリックします。
- ③ プロジェクトテンプレートを選択する画面では「Micro Focus テンプレート[64 ビット]」を選択し [次へ] ボタンをクリックします。
- ④ [接続の新規作成] ボタンをクリックします。



- 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 Linux サーバーの IP アドレスを入力し、[終了] ボタンをクリックします。

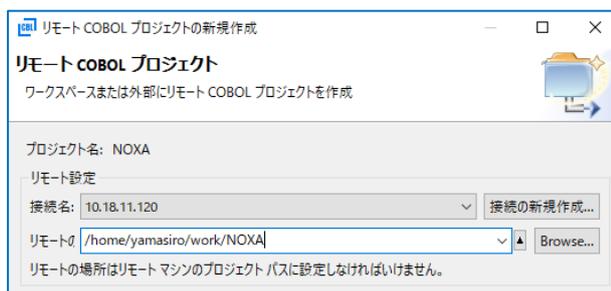


- ⑦ [接続の新規作成...]ボタンの下にある[Browse...] ボタンを押下
- ⑧ 「My Home」の左にある展開アイコンをクリック



ユーザー認証に関するポップアップが出たら Linux サーバーのユーザーの認証情報を入力し、[Save Password] にチェックを入れ、[OK] ボタンをクリックします。

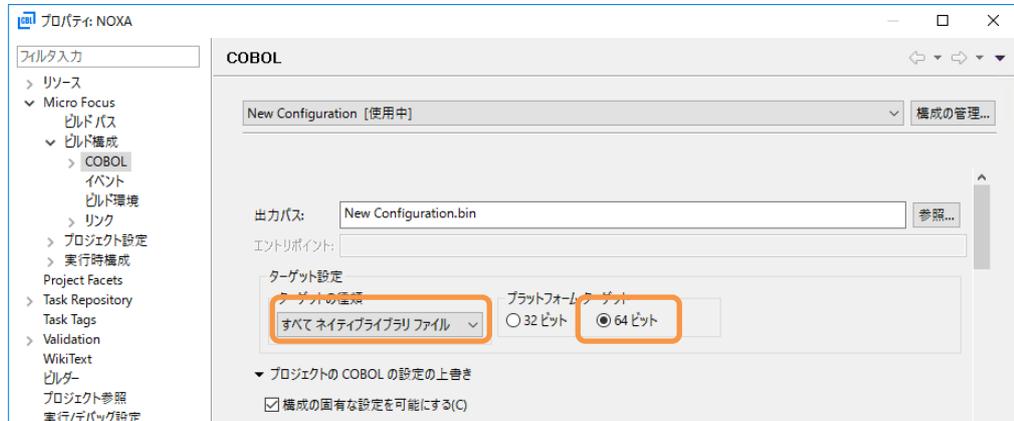
- ⑨ Linux サーバー上でリモート開発のプロジェクトディレクトリとして利用するディレクトリをツリーで指定し、[OK] ボタンをクリックします。
- ⑩ [終了] ボタンをクリックします。



- ⑪ Connection 接続先名 has not been secured using SSL. Proceed anyway? のポップアップに対しては「Do not show this message again」にチェックを入れ [はい] ボタンをクリックします。
- ⑫ Linux サーバー上に Eclipse の COBOL プロジェクトが生成されます。また、同時にリモート接続先のフォルダにもプロジェクトファイルが作成されます。

3) 64 bit の Oracle 連携アプリケーションを生成するようプロジェクトを構成

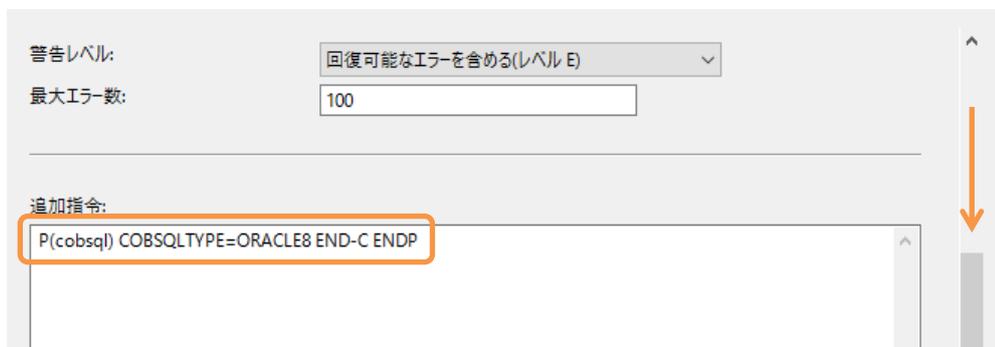
- ① COBOL エクスプローラにて作成したプロジェクトを右クリックし、[プロパティ] を選択します。
- ② [Micro Focus] > [ビルド構成] > [COBOL] を展開します。
- ③ [ターゲットの種類] 欄は「すべてネイティブライブラリファイル」に[プラットフォームターゲット] 欄は [64 ビット] へ変更します。



- ④ [プロジェクトの COBOL の設定の上書き] を展開し、[構成の固有な設定を可能にする] をチェックします。



- ⑤ 画面下へスクロールし、[追加指令] 欄に COBSQL の指令を指定し、[OK]ボタンをクリックします。

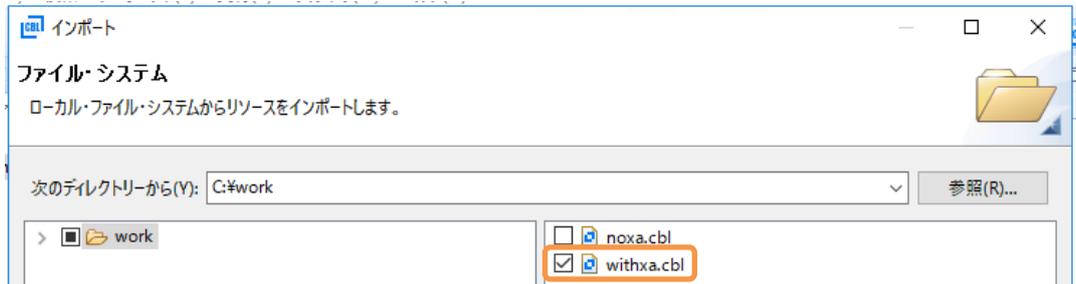


指定値 : 「P(cobsql) COBSQLTYPE=ORACLE8 END-C ENDP」

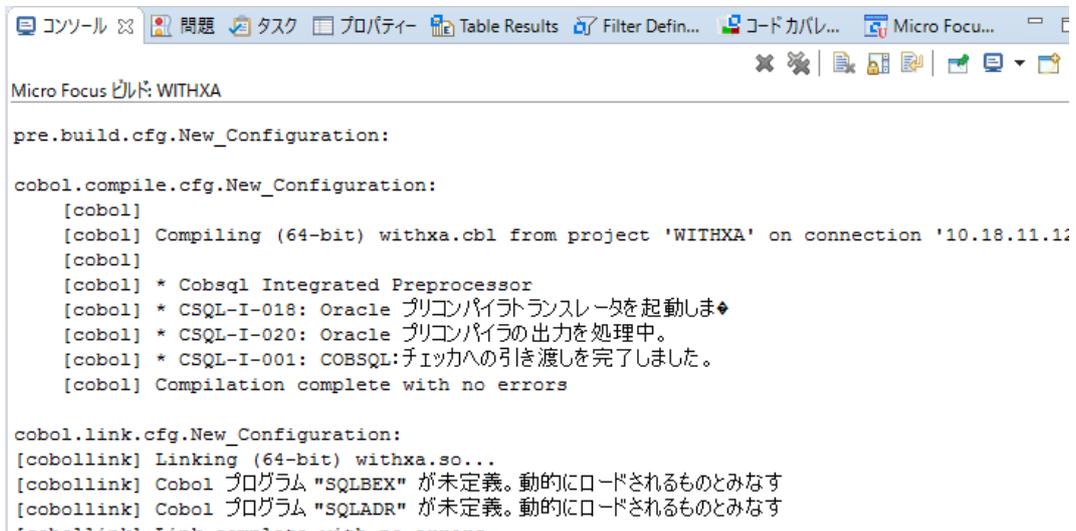
- 4) Oracle 上のデータを照会/更新する埋め込み SQL 文の入った COBOL プログラムをプロジェクトに追加

- 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]を選択します。

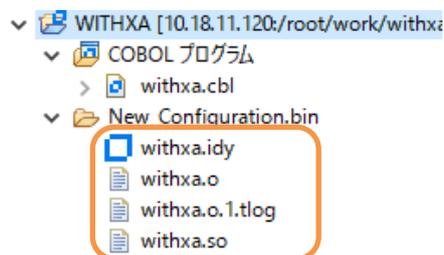
- ② インポート用のダイアログが表示されるので[一般] > [ファイル・システム] を指定し、[次へ]ボタンをクリックします。
- ③ [参照]ボタンを押して Windows のコマンドダイアログからソースコードが保存されているフォルダまで移動します。
- ④ 対象のソースコード「withxa.cbl」にチェックを入れて[終了]ボタンをクリックします。



- ⑤ Windows から Linux へプログラムソースがダイレクトに転送され、Linux 上の Pro\*COBOL プリコンパイラ及び Visual COBOL コンパイラを使ってビルド処理されます。



- ⑥ COBOL エクスプローラビューにて、Linux サーバー環境に呼び出し可能な共有オブジェクトが生成されていることを確認できます。



- ⑦ 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。

```

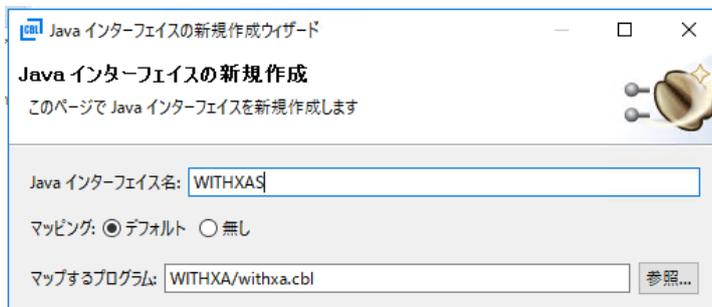
[root@tok-rhel73-64 New_Configuration.bin]# ls
withxa.idy  withxa.o  withxa.o.1.tlog  withxa.so
[root@tok-rhel73-64 New_Configuration.bin]#
  
```

5) インポートしたプログラムの概要

- Oracle Database への接続は XA スイッチモジュールを経由しているためプログラム中には CONNECT 文等の接続関連の命令は記述しません。
- 更新前の EMP.ENAME を取得後、LINKAGE 経由で受け取った値に基づき、EMP テーブルを更新します。
- LINKAGE パラメータ L-COMMIT-OR-ROLLBACK に「R」が渡されると意図的に添え字範囲外の実行時エラーを発生させます。

6) 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「withxa.cbl」を右クリックし、コンテキストメニューから[新規作成] > [Java インターフェイス]を選択します。
- ② Java インターフェイス名には“WITHXAS”を入力し、[終了]ボタンをクリックするとデフォルトのインターフェイスマッピングが生成されます。

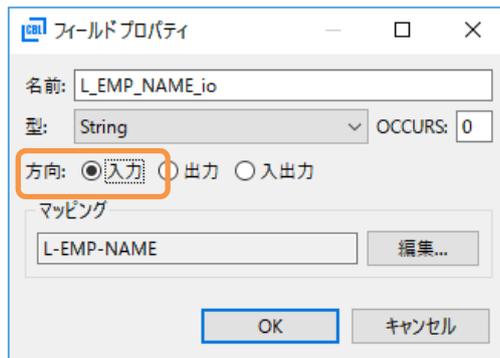


7) 生成されたインターフェイスマッピングを編集

- ① 「WITHXA オペレーション - インターフェイスフィールド:」を編集します。

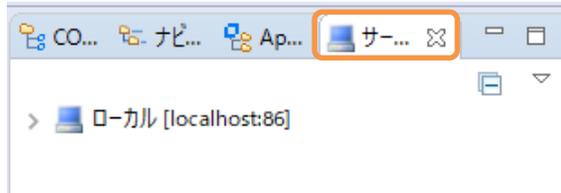


- ② L\_EMP\_NAME\_io をクリックして[方向]をに“入力”に変更し、[OK] ボタンをクリックします。

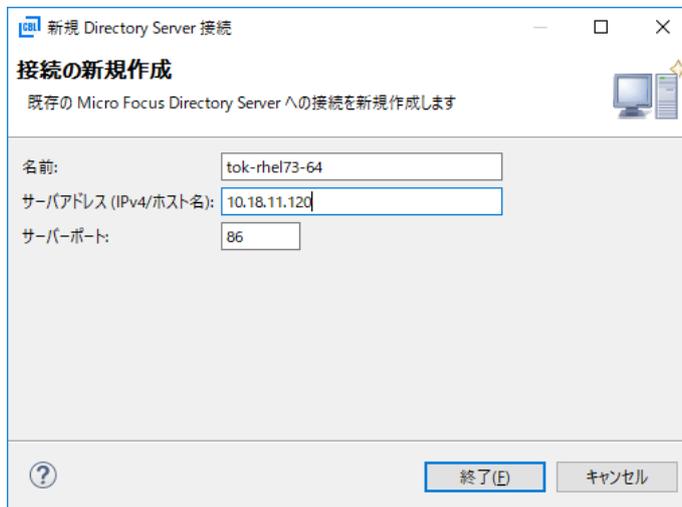


- ③ 同様の作業を「L\_EMP\_NUMBER\_io」、「L\_COMMIT\_OR\_ROLLBACK\_io」に行います。
  - ④ 「L\_LOG\_MSG\_io」は“出力”に変更します。
  - ⑤ CTRL+S キーを押して設定を保存します。
- 8) Linux サーバー上で稼動する Micro Focus Directory Server をサーバーエクスプローラビューへ追加

- ① Eclipse 上でサーバーエクスプローラビューを表示します。



- ② ローカル[localhost:86]をクリックし、右ボタンクリックからコンテキストメニューを表示し、[新規作成(N)] > [Directory Server 接続]を選択します。
- ③ Linux サーバーのアドレスを指定し、[終了] ボタンをクリックします。



### 3.10 64 ビット Enterprise Server のインスタンス作成と起動

- 1) インスタンスの作成

Enterprise Server Administration Console 画面からインスタンスを作成します。

- ① 管理画面左下の [追加] ボタンをクリックします。



- ② 追加画面へ遷移しますので、[サーバー名] に “ESDEMO64” を入力し、動作モードに「64 ビット」を選択して [次へ] ボタンをクリックします。

サーバー追加 (Page 1 of 3):

サーバー名:

動作モード:

32-bit  64-bit

You cannot change your choice of working mode onc

- ③ 次の画面では [サーバータイプ] 欄にて「Micro Focus Enterprise Server」を選択し、[次へ] ボタンをクリックします。

サーバー追加 (Page 2 of 3):

サーバー名:

サーバータイプ:

- Micro Focus Enterprise Server**  
An enterprise server that provides an execution environment for COBOL
- Micro Focus Enterprise Server with Mainframe Subsystem Support**  
An enterprise server that also provides an execution environment for CIC

- ④ 一覧に作成した「ESDEMO64」インスタンスが表示されますので、[編集] ボタンをクリックします。

<input type="button" value="編集..."/>	64	ESDEMO64	停止 <input type="button" value="開始..."/>	1 tcp.* 2 リスナー
--------------------------------------	----	----------	--	-------------------

- ⑤ [リスナー] タブを選択し、「Web Services and J2EE」列中の [編集] ボタンをクリックします。

通信プロセス 1  自動起動

リスナー	プロセスID	コントロールチャネルアドレス
<input type="button" value="編集..."/> 2 <input type="button" value="追加"/>	-	tcp.*

名前	アドレス	ステータス	前回のステータス変更	ステータスログ
<input type="button" value="編集..."/> Web Services and J2EE	tcp.*	停止	12/17/15-10:52:04	OK
<input type="button" value="編集..."/> Web	tcp.*	停止	12/17/15-10:52:04	OK

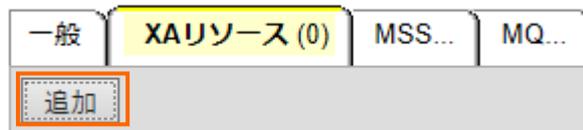
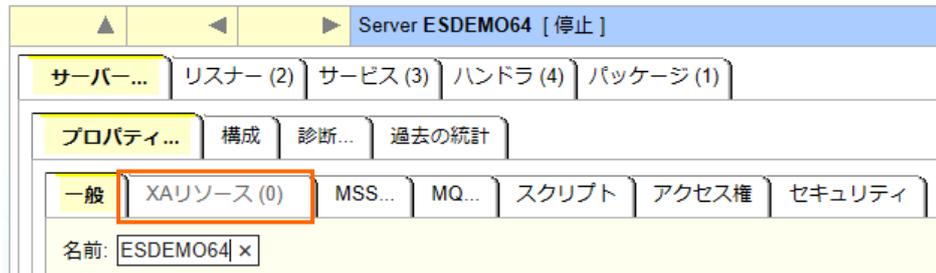
- ⑥ エンドポイントアドレスを下図のように「\*:\*」から「\*:9006」へ変更します。

エンドポイントプロトコル       エンドポイントプロトコル

エンドポイントアドレス       エンドポイントアドレス

- ⑦ [OK]ボタンを押して、画面左上の Home リンクをクリックします。

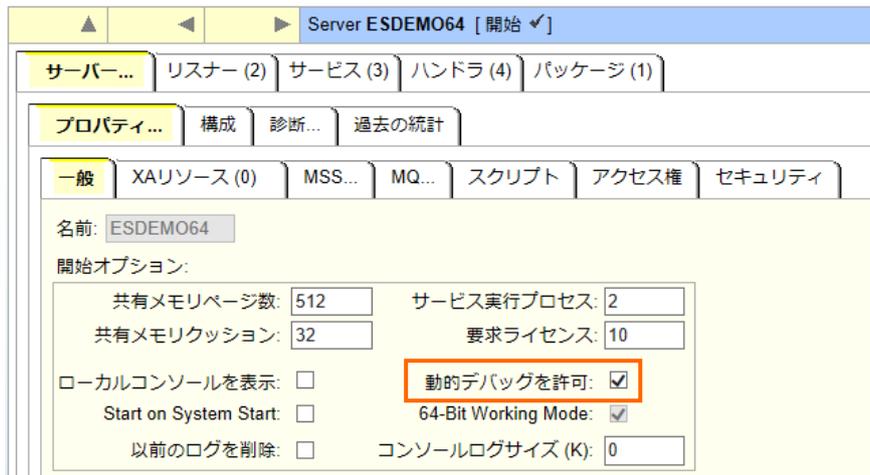
- ⑧ 前項で作成した ESDEMO64 に XA スイッチモジュールを定義します。[編集]ボタンをクリックしてから[サーバー] > [プロパティ] > [XA リソース] タブを表示して、左下の [追加] ボタンをクリックします。



- ⑨ 必要項目を入力後 [追加] ボタンをクリックします。

項目名	説明
ID	ORACLE のインスタンス名を指定します。ここでは ORCL を指定します。
名前	XA リソース名として任意の名前を指定します。 Oracle の場合は Oracle_XA 固定です。
モジュール	前項で作成した XA スイッチモジュールのパスとファイル名を指定します。 【Oracle 使用時の例】 動的登録 /home/tarot/xa/ESORAXA64_D.so を入力します。
OPEN 文字列	対象データベースのオープン文字列を指定します。  【Oracle 使用時の例】 Oracle_XA+SesTm=100+SqlNet=tok-par+Acc=P/scott/tiger を入力します。 OPEN文字列: <code>Oracle_XA+SesTm=100+SqlNet=tok-par+Acc=P/scott/tiger</code>  ※詳しい Oracle への接続文字列は Oracle ドキュメントを参照するか Oracle 管理者に確認してください。
有効	有効、無効切り替えチェックを指定します。ここではオンを指定します。

- ⑩ 次に一般タブの「動的デバッグを許可」を有効にして[適用]ボタンをクリックします。



- ① 画面左上の [Home] をクリックして一覧画面に戻ります。



## 2) Enterprise Server を起動

- ① トップページから Enterprise Server Administration コンソール画面にて「ESDEMO64」の「開始」ボタンをクリックします。
- ② 確認画面が表示されるので続けて「OK」ボタンをクリックします。
- ③ Enterprise Server が起動すると下記のように「スタート」が「開始」に変わります。



## 3) ログを見て XA スイッチモジュールが正しく構成されたことを確認

- ① Enterprise Server Administration コンソール画面にて「編集」ボタン > 「診断」タブ > 「ES コンソール」に移動します。
- ② コンソールログ内に XA 関連のメッセージを確認します。

トレース ダンプ **ESコンソール** CSコンソール

画面更新  Show entries from 1 to 10 of 35 total entries  
 Show last 10 lines

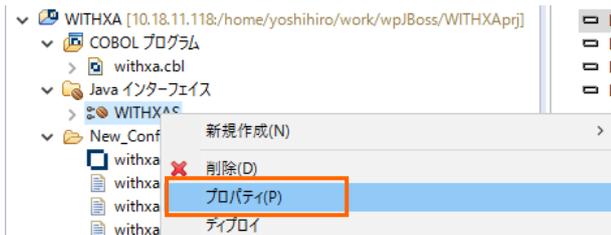
Entry	Event
26	151222 16181773 11074 ESDEMO64 CASTS0002I ES TSC Service Process initialization complete 16:18:17
27	151222 16181773 11081 ESDEMO64 CASC5100I Communications Process instance 01 is ready to accept requests 16:18:17
28	151222 16181869 CASCD1071I Administration SEP created for Server ESDEMO64, process-id = 11097 16:18:18
29	151222 16181871 11097 ESDEMO64 CASSI1500I SEP initialization started 16:18:18
30	151222 16181872 11097 ESDEMO64 CASSI1600I SEP initialization completed successfully 16:18:18
31	151222 16181969 11075 ESDEMO64 CASSI1600I SEP initialization completed successfully 16:18:19
32	151222 16181981 11075 ESDEMO64 CASXO0020I ORCL XA interface loaded. Name(Oracle_XA), Registration Mode(Dynamic) 16:18:19
33	151222 16181988 11075 ESDEMO64 CASXO0015I ORCL XA interface initialized successfully 16:18:19
34	151222 16181993 11075 ESDEMO64 CASSI5040I Active SEP memory strategy set to x'00000001', retain count 100 16:18:19
35	151222 16182071 11078 ESDEMO64 CASSI1600I SEP initialization completed successfully 16:18:20

Show Entire Log

### 3.11 コンパイルした COBOL アプリケーションを Enterprise Server ヘッドプロイ

#### 1) Enterprise Server へのデプロイ情報を指定

- ① COBOL エクスプローラにて追加した Java インターフェイス「WITHXAS」を右クリックし、コンテキストメニューから [プロパティ] を選択します。



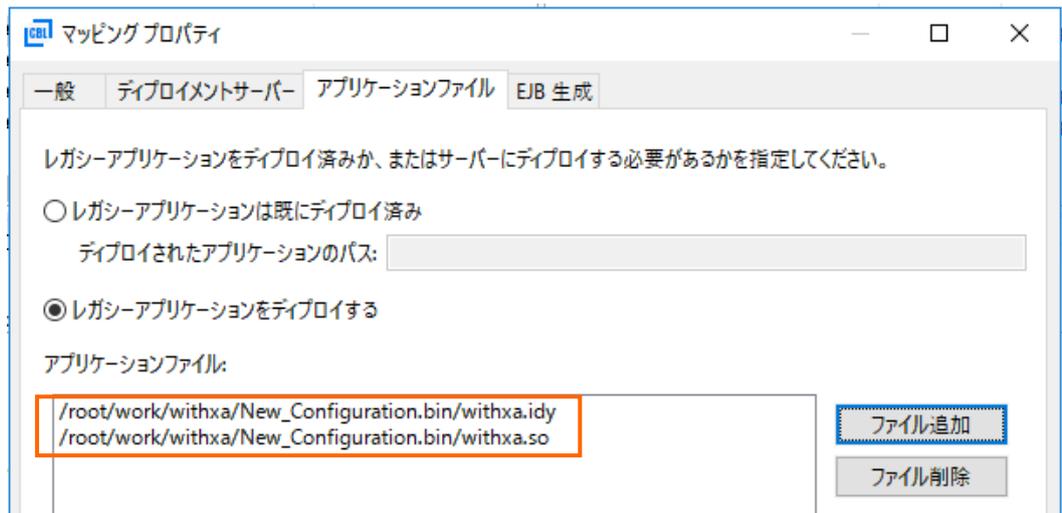
- ② [デプロイメントサーバー] タブを選択し、[変更]ボタンをクリックします。
- ③ Linux サーバーで稼動する「ESDEMO64」を選択し [OK] ボタンをクリックします。

Enterprise Server を選択

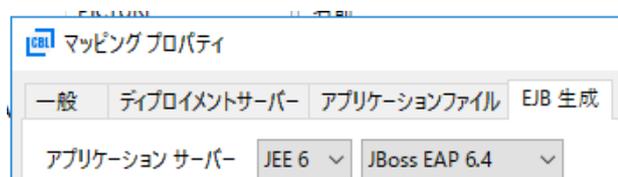
デプロイ先の Enterprise Server を選択してください:

サーバー	サービス名	サービス状態	エンドポイント	リスナー状態
ESDEMO	Deployer	Available, Sto...	0.0.0.0:0	BitMode=32-Bit
ESDEMO64	Deployer	Available, Sta...	10.18.11.118:62460	BitMode=64-Bit

- ④ [トランザクション管理] フィールドにて「コンテナ管理」を選択します。
- ⑤ 次に[アプリケーションファイル] タブを選択し、「レガシーアプリケーションをデプロイする」を選択します。
- ⑥ [ファイル追加]ボタンを押して、プロジェクトディレクトリ配下の「New\_Configuration.bin」に生成された withxa.so 及び withxa.idy を選択し、[OK] ボタンをクリックします。
- ⑦ Linux サーバーで稼動する「ESDEMO64」を選択し [OK] ボタンをクリックします。



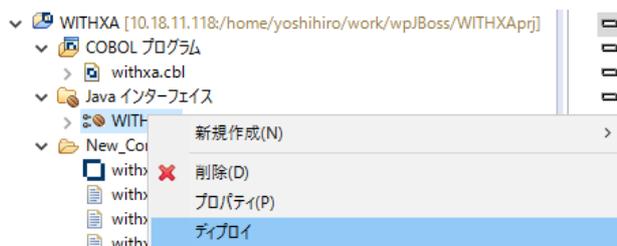
- ⑧ 次に[EJB 生成] タブを選択し、[アプリケーションサーバー]フィールドにて「J2EE6」、「JBoss EAP 6.4」を選択します。



- ⑨ [トランザクション可能]のチェックを外して無効に設定します。
- ⑩ 「J2EE と J2EE の属性」欄にて[Java コンパイラ]フィールドに使用している JDK のパスを入力します。
- ⑪ 同様に下記の J2EE クラスパスを設定します。
- > \$JBASS\_HOME/modules/system/layers/base/javax/ejb/api/main/jboss-ejb-api\_3.1\_spec-1.0.2.Final-redhat-3.jar
  - > \$JBASS\_HOME/modules/system/layers/base/javax/servlet/api/main/jboss-servlet-api\_3.0\_spec-1.0.2.Final-redhat-2.jar
  - > \$JBASS\_HOME/modules/system/layers/base/javax/resource/api/main/jboss-connector-api\_1.6\_spec-1.0.1.Final-redhat-3.jar

## 2) 作成した Java サービスを Enterprise Server ヘッドアップロードする

- ① COBOL エクスプローラにて作成した Java インターフェイス「WITHXAS」を右クリックし、コンテキストメニューから [デプロイ] を選択します。



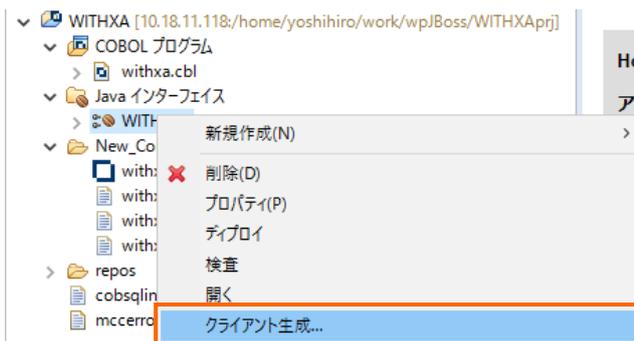
- ② Enterprise Server Administration コンソール画面より[Home] > [編集] ボタン > [サービス] タブと移動して正常にデプロイできたことを確認することができます。

Server ESDEMO64 [開始 ✓]									
サーバー...		リスナー (2)		サービス (4)		ハンドラ (4)		パッケージ (2)	
サービス表示フィルタ		ネームスペース:			オペレーション:				
1 - 4 of 4 displayable namespaces from a total of 4									
サービス ネーム スペース	オペレーショ ン	サービス クラス	探索 順序	リスナー	要求 ハンドラ	実装 パッ ケージ	現 ステータ ス	ス テータ ス ログ	
Deployer	Deployer 編集...	MF deployment	1	1 CP 1 Web tcp:10.18.11.118*:62460* (ym-rhel71)			Available	OK	
ES	ES 編集...	MF ES	1	1 CP 1 Web Services and J2EE tcp:10.18.11.118*:9006 (ym-rhel71)			Available	OK	
削除...	noxa	1 of 1 operations shown							
	SELECTEMP 編集...		1	1 CP 1 Web Services and J2EE tcp:10.18.11.118*:9006 (ym-rhel71)	MFRHBINP	noxa	Available	OK	
削除...	withxa	1 of 1 operations shown							
	WITHXA 編集...		1	1 CP 1 Web Services and J2EE tcp:10.18.11.118*:9006 (ym-rhel71)	MFRHBINP	withxa	Available	OK	

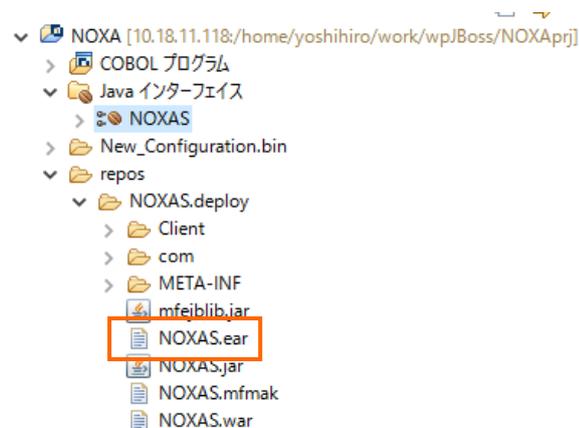
### 3.12 テスト用クライアントアプリケーションを JBoss EAP にデプロイ

1) デプロイした Java サービスをテストするための J2EE アプリケーションを生成する

- ① 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] を選択します。



- ② 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 配下に拡張子.ear 形式にアーカイブされた J2EE アプリケーションが生成されます。



2) 生成された J2EE アプリケーションを JBoss EAP 6.4 にデプロイ

- ① Telnet で Visual COBOL 作業用ディレクトリに生成された "WITHXAS.ear" を

\$JBOSS\_HOME/standalone/deployments 配下にコピーする。

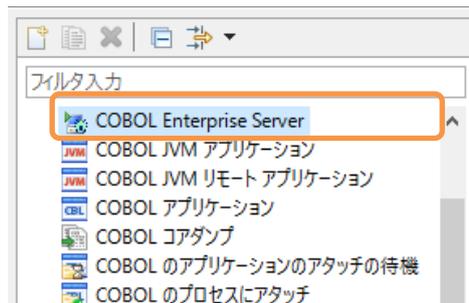
例：

cp WITHXAS.ear \$JBOSS\_HOME/standalone/deployments

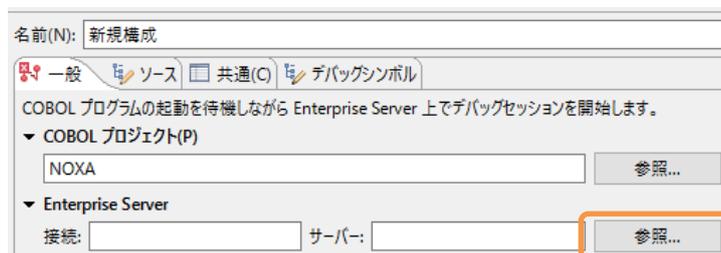
- ② 正常にデプロイされたことを JBoss を起動した Linux のターミナルから確認ができます。

### 3) Enterprise Server をデバッグ待機する

- ① COBOL エクスプローラにてプロジェクトを右クリックし、コンテキストメニューから[デバッグ] > [デバッグの構成]を選択します。
- ② [COBOL Enterprise Server]をダブルクリックします。



- ③ [一般]タブの Enterprise Server フィールドの[参照]ボタンをクリックします。



- ④ Linux サーバー上で稼働する ESDEMO64 を選択し、[OK] ボタンをクリックします。



- ⑤ [Java] タブをクリックし、Java サービス名が空白（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します。
- ⑥ [デバッグ]ボタンをクリックし、[パースペクティブの切り替えの確認] のプロンプトに対しては [はい] を選択します。
- ⑦ 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が確認できます。



### 3.13 テスト用アプリケーションを経由して COBOL アプリケーションを呼び出し

1) デployした J2EE アプリケーションをデバッグ実行する（成功パターン）

- ① ブラウザを起動し、Jboss 実行中のサーバーの IP アドレスを入力し、アプリケーションを起動します。

例 : 10.18.11.118:8080/WITHXAS/WITHXA.jsp



## Test client for WITHXAS.WITHXA

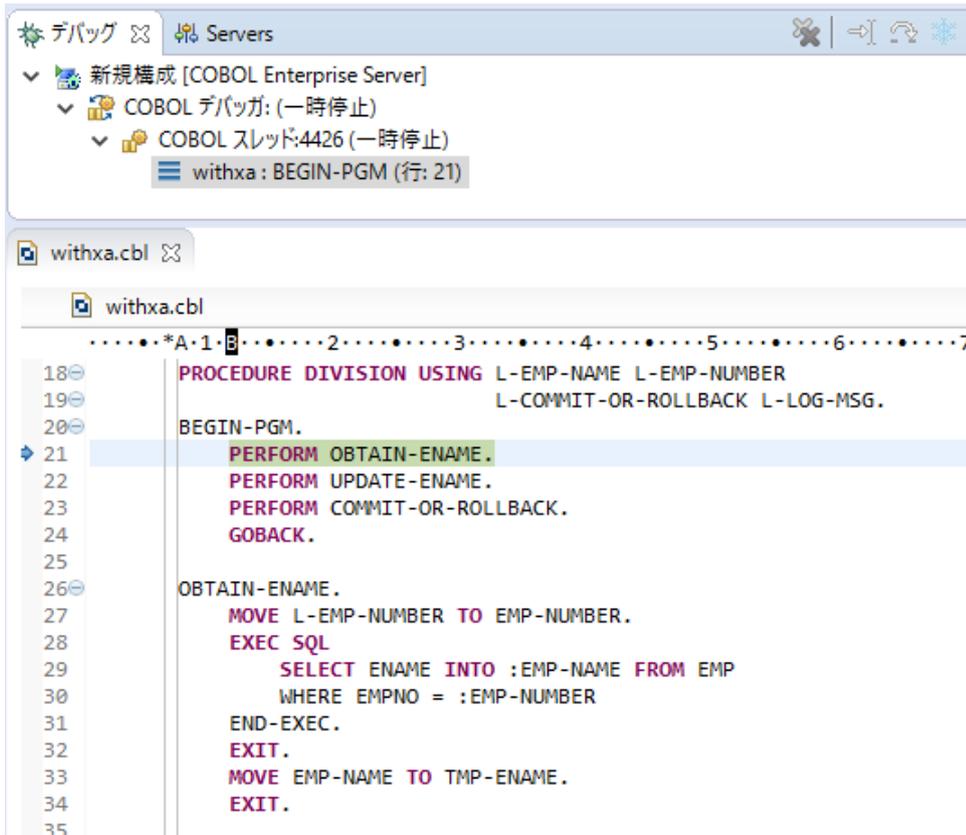
[Back](#)

- ② 3つのパラメータを入力し、[Go!]ボタンをクリックして、アプリケーションを実行します。

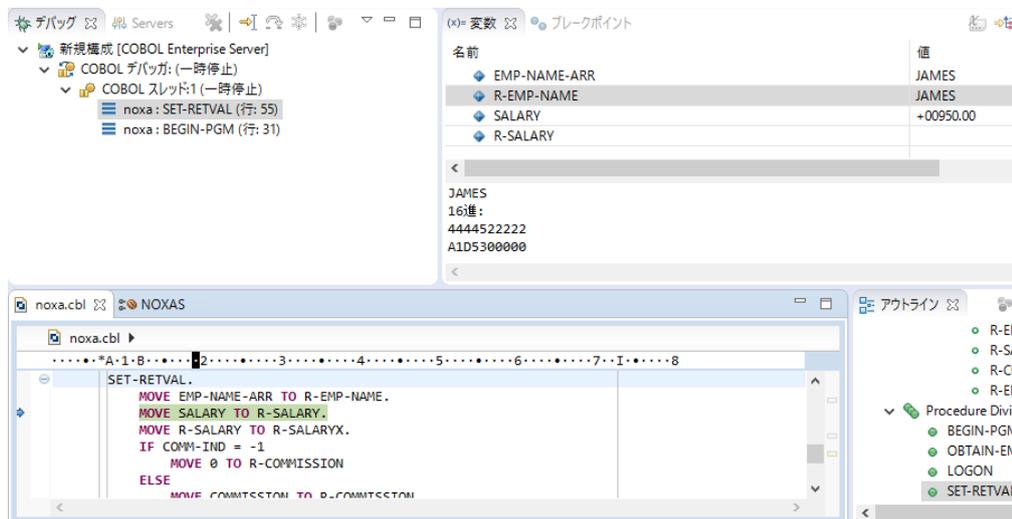
Perform the test by entering values:

WITHXA_L_EMP_NAME_io :	<input type="text" value="HOGAN"/>
WITHXA_L_EMP_NUMBER_io :	<input type="text" value="7876"/>
WITHXA_L_COMMIT_OR_ROLLBACK_io :	<input type="text" value="C"/>

- ③ 処理が Enterprise Server も渡り、Eclipse のデバッガが起動します。
- ④ Enterprise Server に deployした COBOL プログラムの最初の行を実行する前で処理が一時停止していることが確認できます



- ⑤ 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 文が入ったプリコンパイル前のソースでデバッグができます。
- ⑥ 変数ビューでは、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。



- ⑦ 本プログラムはトランザクションマネージャー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、正常に SQL 文を実行しています。
- ⑧ 処理を最後まで進めると Java 側に処理が戻り、COBOL から返された値を戻します。

Perform the test by entering values:

WITHXA\_L\_EMP\_NAME\_io :

WITHXA\_L\_EMP\_NUMBER\_io :

WITHXA\_L\_COMMIT\_OR\_ROLLBACK\_io :

Result:

Variable	Value
Result	ENAME CHANGED FROM ADAMS TO HOGAN

[Back](#)

- ⑨ アプリケーションが処理したレコードを SQL\*Plus で確認します。トランザクションが COMMIT されたので値が更新されていることが確認できます

```
SQL> select * from emp where EMPNO=7876;

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7876	HOGAN	CLERK	7788	23-MAY-87	1100	

```
SQL>
```

- 2) デプロイした J2EE アプリケーションをデバッグ実行する（実行時エラーで終了するパターン）

- ① ブラウザを起動し、Jboss 実行中のサーバーの IP アドレスを入力し、アプリケーションを起動します。
- ② 3つのパラメータを入力し、[Go!]ボタンをクリックして、アプリケーションを実行します。

## Test client for WITHXAS.WITHXA

[Back](#)

Perform the test by entering values:

WITHXA\_L\_EMP\_NAME\_io :

WITHXA\_L\_EMP\_NUMBER\_io :

WITHXA\_L\_COMMIT\_OR\_ROLLBACK\_io :

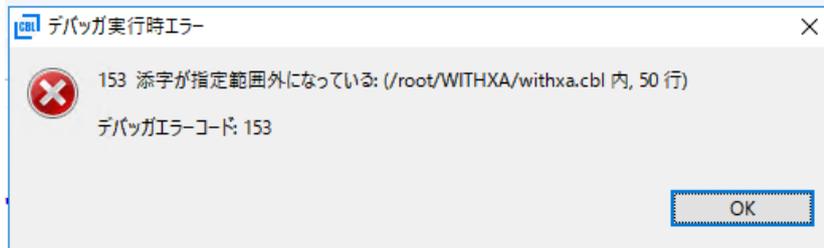
- ③ 今回は L-COMMIT-OR-ROLLBACK に「R」を格納したため、UPDATE 文実行後の下図の IF 文は真と評価されます。

```

withxa.cbl  サーバー: tok-rhel73-64  noxa.cbl  WITHXAS
withxa.cbl
-----*A-1-3-----2-----3-----4-----5-----6-----
        WHERE EMPNO = :EMP-NUMBER
        END-EXEC .
        EXIT .
        STRING "ENAME CHANGED FROM " TMP-ENAME "TO " EMP-NAME
        INTO L-LOG-MSG .
        EXIT .

COMMIT-OR-ROLLBACK .
        IF L-COMMIT-OR-ROLLBACK = 'R'
            SET IDX TO 11
            MOVE SPACE TO TABLE-ITEM (IDX)
        END-IF .
        EXIT .
    
```

- ④ IF 文中の MOVE 文を実行すると添え字範囲外の実行時エラーが発生します。



- ⑤ 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。

```

JBoss Web/7.5.7.Final-redi X
10.18.11.118:8080/WITHXAS/WITHXASServlet
org.jboss.ejb.client.EJBInvocationHandler.doInvoke(EJBInvocationHandler.java:100)
org.jboss.ejb.client.EJBInvocationHandler.invoke(EJBInvocationHandler.java:146)
com.sun.proxy.$Proxy11.WITHXA(Unknown Source)
com.mypackage.WITHXAS.WITHXASServlet.performTask(Unknown Source)
com.mypackage.WITHXAS.WITHXASServlet.doPost(Unknown Source)
javax.servlet.http.HttpServlet.service(HttpServlet.java:754)
javax.servlet.http.HttpServlet.service(HttpServlet.java:847)

JBWEB000071: root cause
javax.resource.spi.EISSystemException: CobolException Recoverable:
目的コード エラー: ファイル 'withxa'
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/work/wpJBoss/WITHXAprj/withxa.cbl 内, 50 行) executing withxa.WITHXA
com.microfocus.cobol.connector.cci.CobolInteraction.exec(CobolInteraction.java:291)
com.microfocus.cobol.connector.cci.CobolInteraction.execute(CobolInteraction.java:176)
com.mypackage.WITHXAS.WITHXASBean.WITHXA(Unknown Source)
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.lang.reflect.Method.invoke(Method.java:806)
    
```

- ⑥ アプリケーションが処理したレコードを SQL\*Plus で確認します。トランザクションがロールバックされたので Update 文による値の更新は取り消されています。

```
SQL> select * from emp where EMPNO=7876;

  EMPNO ENAME      JOB              MGR HIREDATE          SAL          COMM
-----
DEPTNO
-----
   7876 HOGAN          CLERK            7788 23-MAY-87          1100
      20

SQL>
```

### 3.14 インスタンスの停止

#### 1) Enterpiser Server の停止

- ① Enterprise Server Administration Console 画面から「ESDEMO64」を停止します。

## WHAT'S NEXT

- 本チュートリアルで学習した技術の詳細については製品マニュアルをご参照ください。