Micro Focus Visual COBOL 2.3J for x64/x86 Linux

Red Hat JBoss Enterprise Application Platform 6.4

動作検証 検証結果報告書

2015 年 12 月 24 日

マイクロフォーカス株式会社

1. 検証概要、目的及びテスト方法

1.1 検証概要

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岐に渡る 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、Red Hat JBoss Enterprise Application Platform(以下 EAP) 6.4 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 2.3J の Enterprise Server が提供する Java EE Connector は、WebSphere, WebLogic, JBoss Application Server(以下 AS) の各種バージョンとの連携動作を検証した上で提供されています。しかし、この Enterprise Server は JCA 仕様準拠のコンテナとして設計されているため、この他にも JCA 仕様に準拠したすべてのアプリケーションサーバーとの連携も可能です。

Visual COBOL は JCA 1.6 仕様に準拠し EJB 3.1 をサポートするコンテナを装備する JBoss AS 7.1.1 向けのリソースアダプタを提供しています。Red Hat は JBoss EAP 6.4 に関して JCA 1.6 仕様をサポートすると公表しています[1]。また、JCA 仕様に準拠したリソースアダプタであれば同サーバへディプロイ可能である旨の記述を同文書より確認できます。つまり、理論的には Micro Focus が提供する JBoss AS 7.1.1 向けのリソースアダプタは JBoss EAP 6.4 に対してもディプロイが可能です。即ち、JBoss EAP 6.4 は Enterprise Information Systems として稼動する Enterprise Server と連携が可能です。

[1] JBoss Enterprise Application Platform 6.4 Administration and Configuration Guide
引用文：「JBoss EAP 6 provides full support for the Java EE Connector API (JCA) 1.6 specification.」
https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.4/html/Administration_and_Configuration_Guide/chap-Java_Connector_Architecture_JCA.html#About_the_Java_EE_Connector_API_JCA1
(リンク確認: 2015/12/09)

Visual COBOL に付属する Interface Mapping ToolKit は Enterprise Server へディプロイするサービス向けに EJB3.1 準拠のラッパーコードを自動生成させる機能も有しています。JBoss EAP 6.4 は EJB 3.1 仕様に従って開発されたアプリケーションを完全サポートします[2]。そこで、本検証ではこのラッパーコードを用いて JBoss EAP 6.4 と Enterprise Server の連携を確認しました。

本検証では、コンテナによるトランザクション管理が不要なアプリケーション向けのリソースアダプタを使った連携並びに XA によるコンテナトランザクション管理機能を利用するアプリケーション向けのリソースアダプタが正常に JBoss EAP 6.4 と連携動作することを検証しました。Visual COBOL は、Oracle Database、IBM DB2、Microsoft SQL Server、IBM WebSphere MQ のリソースマネージャとの連携に加え ODBC 経由での接続[3]も動作保証しています。本検証ではこのうち Oracle Database に対するリソースマネージャアクセス機能を使ってコンテナ管理によるトランザクション管理機能が正常に動作することも確認しています。

Visual COBOL の Linux/UNIX 版ライセンスは Linux/UNIX 環境にインストールして利用する Development Hub に加えて Windows 環境にインストールして利用する Eclipse 版のライセンスもセットになって提供されています。利用者はこの Eclipse 版に装備されたリモート開発機能を使って Windows 側で操作しつつも Linux/UNIX をターゲットとしたアプリケーション開発に取り組むことができます。本検証ではこのリモート開発機能を用いて Windows 上でアプリケーションの開発、サービス定義・ディプロイ、デバッグ実行の全てを操作しています。

---

[2] JBoss Enterprise Application Platform 6.4 Development Guide
引用文：「JBoss EAP 6 has full support for applications built using the Enterprise JavaBeans 3.1 specification.」
https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.4/html/Development_Guide/chap-Enterprise_JavaBeans.html#Overview_of_Enterprise_JavaBeans
(リンク確認: 2015/12/09)

[3] Oracle Database、DB2、Microsoft SQL Server に関しては two-phase commit 機能をサポートしていますが、ODBC 経由で接続する場合は、one-phase commit のみのサポートとなります。

2. 検証環境

- **Application 開発サーバ**

  ソフトウェア

  - Red Hat Enterprise Linux 7.1(VM のゲスト OS として稼動)
  - Red Hat JBoss Enterprise Application Platform 6.4
  - Oracle Database 12c Release 1 (12.1.0.2.0)
  - Java SE Development Kit 7, Update 79
  - Micro Focus Visual COBOL 2.3J Development Hub(Hot Fix 1 適用版)

  ハードウェア

  機種：　Dell Latitude E6530
  CPU：　Intel Core i5-3360M 2.80GHz
  Memory：　3.00 Gbyte memory(ゲスト OS に割り当てたサイズ)

- **Application 開発クライアント**

  ソフトウェア

  - Windows 10 Enterprise 32bit (VM のゲスト OS として稼働)
  - Micro Focus Visual COBOL 2.3J for Eclipse(Hot Fix 1 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。詳細な手順については付録に記載します。

3.1 Oracle 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

Oracle は Pro*COBOL のサンプルプログラムとしてサンプルスキーマ scott における emp 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 SAMPLE1.pco を提供しています。このプログラムは照会するキーは ACCEPT 文より受け取り、結果を DISPLAY 文で出力していますが、本検証ではこれらをそれぞれ入力、出力パラメータに変更して利用しました。

(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

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 Oracle 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

入力パラメータとして指定された emp テーブルのキーに対するレコードを、指定された値で UPDATE する簡単な COBOL サブルーチンです。更に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。ここでは XA によるコンテナ管理トランザクションを実現させるため、接続や TCS 関連の命令は埋め込まれていません。

(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あり）

(3) 使用した Enterprise Server

3.1 で検証用に追加した Enterprise Server を利用しました。

本検証においては、Oracle 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用意し、これを Enterprise Server に XA リソースとして追加登録・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、続けて自動生成可能な Web モジュールクライアントを使用しました。実行後、Oracle 上の該当する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. 検証結果

以下の「付録」で示す通り、上記の一連の作業が問題なく実行できることが検証できました。

以上

付録 1. Oracle 照会プログラムのディプロイと、EJB 経由の JCA 呼び出し

■ Linux サーバ

1) 一般ユーザでログインし、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/VC23
$
```

2) Oracle のライブラリをリンクインするためのオプションファイルを生成[4]

```
$ $COBDIR/src/oracle/set_cobopt_oracle
Set COBOPT to /home/yoshihiro/work/wpJBoss/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) root 権限を持ったユーザに切り替え

4) Visual COBOL の環境設定スクリプトを実行

5) 2) で生成した cobopt.ora を環境変数 COBOPT に指定

```
# export COBOPT=`pwd`/cobopt.ora
#
```

6) 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
                          :
```

7) 別セッションにて JBoss の実行権限を持つユーザでログイン

8) Visual COBOL のマニュアルに従い、standalone.xml を構成

① <archive-validation> タグを修正

[4] Pro*COBOL によりプリコンパイル展開された COBOL プログラムを実行形式のモジュールにビルドするには Oracle が提供するサンプル make ファイル demo_procob.mk で処理されるように Oracle のライブラリをリンクリンする必要があります。
$COBDIR/src/oracle/set_cobopt_oracle スクリプトを実行するとカレントディレクトリの cobopt.ora にこのリンクインオプションを書き出します。

② <subsystem> タグにてリソースアダプタ mfcobol-notx.rar に関連する情報を指定[5]

9) リソースアダプタ中に埋め込まれている Enterprise Server のポートを 9003 から 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: jboss7
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-notx.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
#
```

JBoss 7.1.1 向けの mfcobol-notx.rar を指定します。

ポート番号を 9006 へ変更します。

[5] マニュアルでは JBoss Application Server 7.1 中の standalone.xml に合わせて xmlns 属性は「urn:jboss:domain:resource-adapters:**1.0**」と表示していますが、JBoss EAP 6.4 では「urn:jboss:domain:resource-adapters:**1.1**」となります。本ファイルを構成する際は JBoss EAP のデフォルト構成に合わせて「1.1」のままにしておきます。

④　一般ユーザに戻る

10) リソースアダプタ mfcobol-notx.rar を JBoss EAP 6.4 へディプロイ

```
$ cp $COBDIR/javaee/javaee6/jboss7/mfcobol-notx.rar $JBOSS_HOME/standalone/d
eployments
$
```

11) JBoss を起動

```
$ $JBOSS_HOME/bin/standalone.sh -b 0.0.0.0
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /opt/JBoss/jboss-eap-6.4

  JAVA: java

  JAVA_OPTS:  -server -verbose:gc -Xloggc:"/opt/JBoss/jboss-eap-6.4/standalone/log/gc.log
" -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogF
iles=5 -XX:GCLogFileSize=3M -XX:-TraceClassUnloading -Xms1303m -Xmx1303m -XX:MaxPermSize=
256m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djav
a.awt.headless=true -Djboss.modules.policy-permissions=true

=========================================================================

[Om13:20:49,814 INFO  [org.jboss.modules] (main) JBoss Modules version 1.3.6.Final-redhat
-1
[Om[Om13:20:50,377 INFO  [org.jboss.msc] (main) JBoss MSC version 1.1.5.Final-redhat-1
[Om[Om13:20:50,603 INFO  [org.jboss.as] (MSC service thread 1-2) JBAS015899: JBoss EAP 6.
4.0.GA (AS 7.5.0.Final-redhat-21) が起動しています。
                                                    :
[Om[Om13:06:42,285 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-2) JBAS01
5876: "mfcobol-notx.rar" (runtime-name: "mfcobol-notx.rar") のデプロイメントを開始しまし
た。
[Om[Om13:06:42,541 INFO  [org.jboss.as.remoting] (MSC service thre
7.0.0.1:9999 をリッスン
[Om[Om13:06:42,545 INFO  [org.jboss.as.remoting] (MSC service thre
7.0.0.1:4447 をリッスン
[Om[Om13:06:42,545 INFO  [org.jboss.as.connector.subsystems.datasources] (MSC service thr
ead 1-1) JBAS010400: データソース[java:jboss/datasources/ExampleDS] をバインドしました。
[Om[Om13:06:43,319 INFO  [org.jboss.as.connector.deployment] (MSC service thread 1-1) JBA
S010406: 接続ファクトリ java:/eis/MFCobol_v1.5 を登録しました。
[Om[Om13:06:43,430 INFO  [org.jboss.as.connector.deployers.RaXmlDeployer] (MSC service th
read 1-1) IJ020002: Deployed: file:/opt/JBoss/jboss-eap-6.4/standalone/tmp/vfs/temp/tempc
33922fc63f06064/mfcobol-notx.rar-c67d6d065ae7bd04/contents/
[Om[Om13:06:43,437 INFO  [org.jboss.as.connector.deployment] (MSC service thread 1-1) JBA
S010401: JCA ConnectionFactory [java:/eis/MFCobol_v1.5] をバインドしました。
[Om[Om13:06:43,557 INFO  [org.jboss.as.server] (ServerService Thread Pool -- 28) JBAS0158
59: "mfcobol-notx.rar" (runtime-name : "mfcobol-notx.rar") をデプロイしました。
```

リソースアダプタがディプロイされたことが確認できます。

```
Om[Om13:06:43,608 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015961: http://127.0.
0.1:9990/management 上でリッスンする HTTP 管理インターフェース
[Om[Om13:06:43,609 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015951: 管理コンソー
ルは http://127.0.0.1:9990 をリッスンしています。
[Om[Om13:06:43,609 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss EAP 6.
4.0.GA (AS 7.5.0.Final-redhat-21) は 6889ms で開始しました – サービス 230 個のうち 192
個を開始しました (59 のサービスはパッシブ、またはオンデマンドです)。
```

正しく起動できたこと
が確認できます。

■ Windows クライアントマシン

12) 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

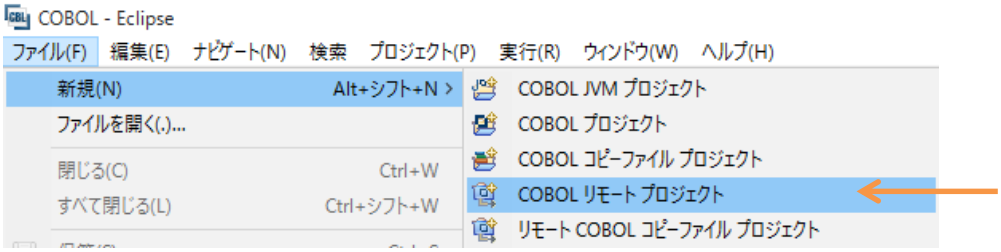
① ＜Visual COBOL のインストールフォルダ＞¥bin¥mf-client.dat をテキストエディタで開く

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

13) Visual COBOL for Eclipse を起動

14) COBOL リモートプロジェクトを作成

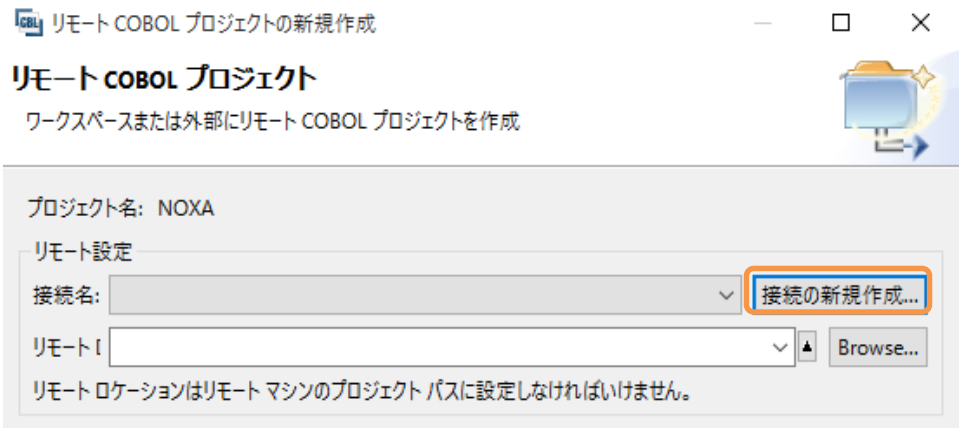
① [ファイル] メニュー > [新規] > [COBOL リモートプロジェクト] を選択



③ プロジェクト名を指定し、[次へ] ボタンを押下

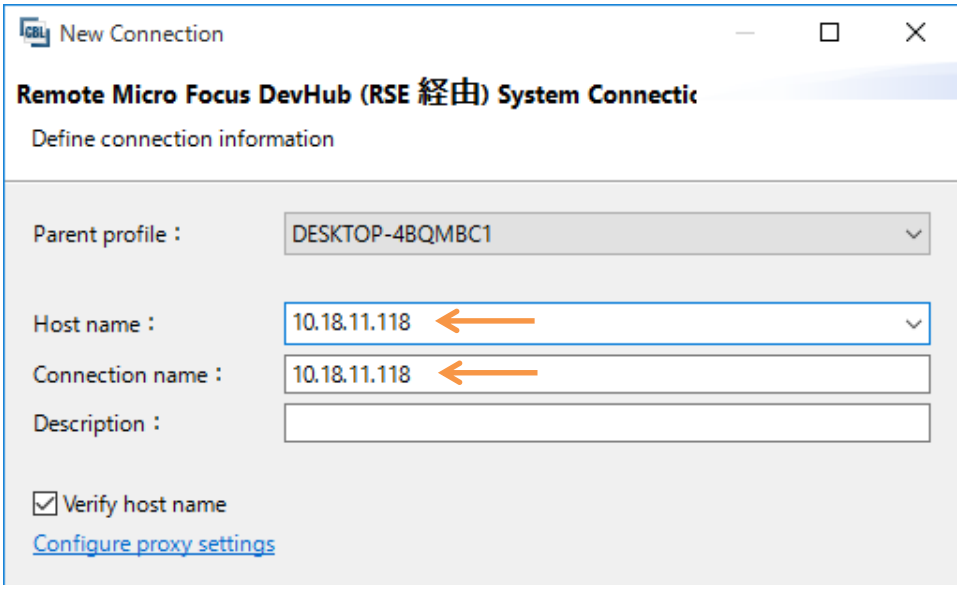
④ プロジェクトテンプレートを選択する画面では [Micro Focus テンプレート] を選択し [次へ] ボタンを押下

⑤ [接続の新規作成] ボタンを押下



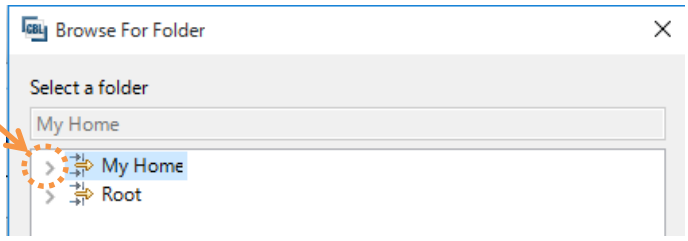
⑥ [Micro Focus DevHub(RSE 経由)] を選択の上 [次へ] ボタンを押下

⑦ [Host name] 欄に Linux サーバの IP アドレスを入力し、[終了] ボタンを押下



⑧ [Browse] ボタンを押下

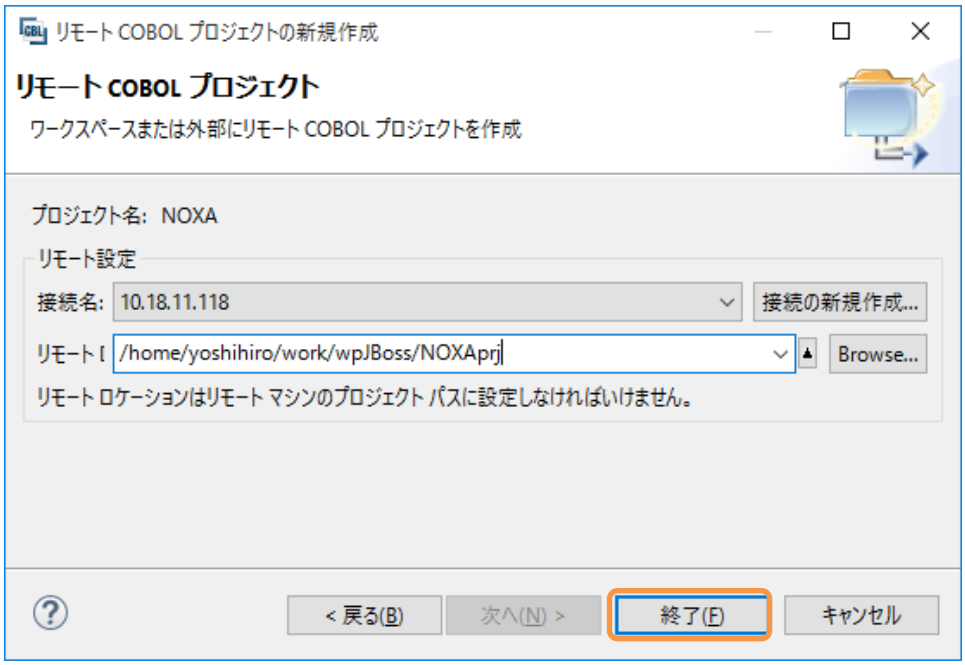
⑩ 「My Home」の左にある展開アイコンをクリック



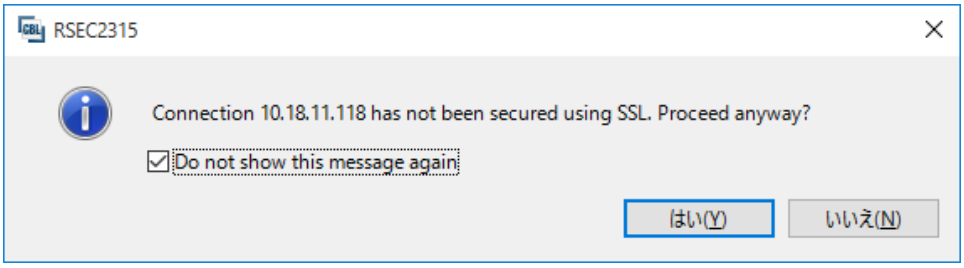
⑪ ユーザ認証に関するポップアップが出たら Linux サーバのユーザの認証情報を入力し、[Save Password] にチェックを入れ、[OK] ボタンを押下

⑫ Linux 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

⑬ [終了] ボタンを押下

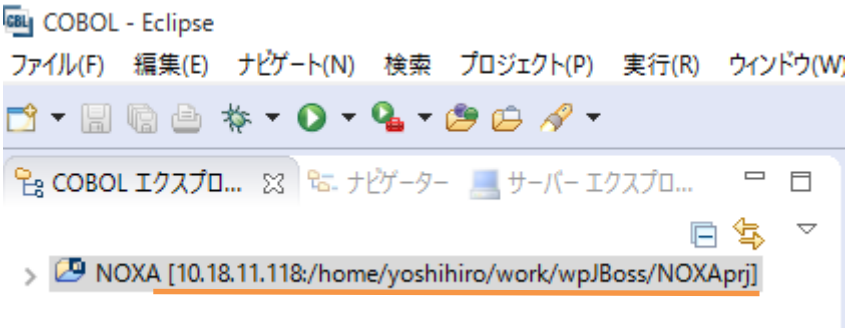


⑭ 下図のようなポップアップに対しては「Do not show this message again」にチェックを入れ［はい］ボタンを押下

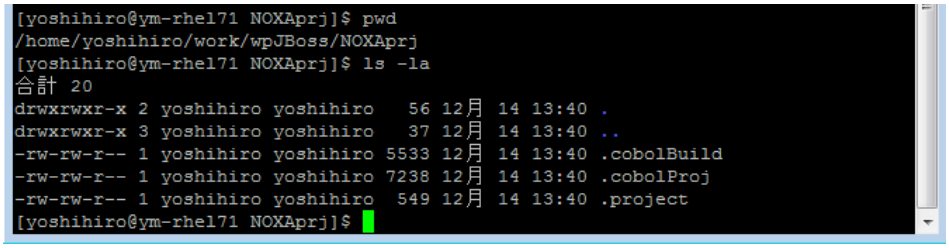


Linux サーバ上に Eclipse の COBOL プロジェクトが生成されます。

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:



Linux サーバ上に生成されたプロジェクトファイル

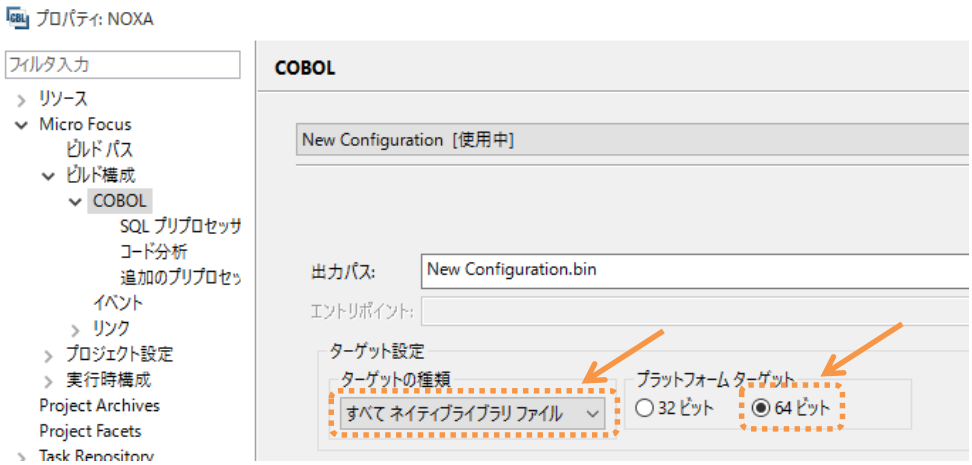


15) 64 bit の Oracle 連携アプリケーションを生成するようプロジェクトを構成

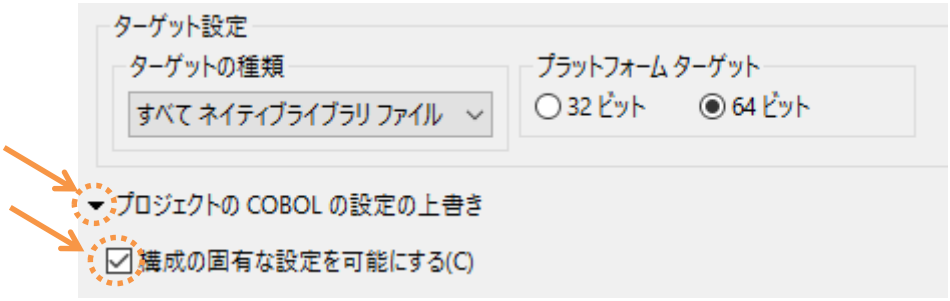
① COBOL エクスプローラにて作成したプロジェクトを右クリックし、[プロパティ] を選択

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

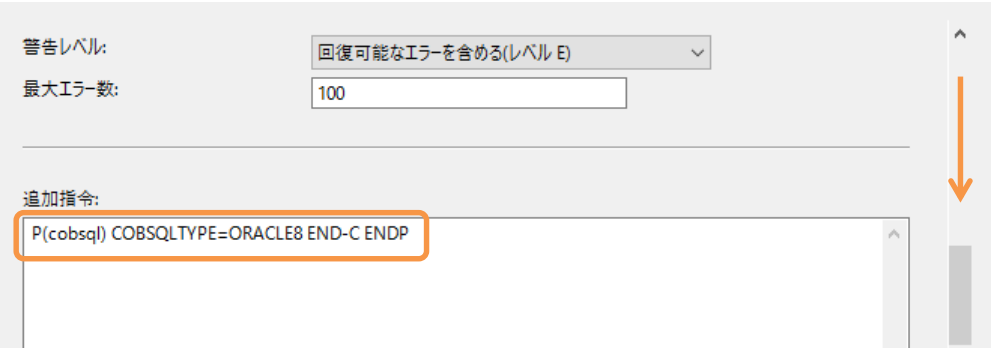
③ [ターゲットの種類] 欄は「すべてネイティブライブラリファイル」に[プラットフォームターゲット] 欄は [64 ビット] へ変更



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



⑤ 画面下へスクロールし、[追加指令] 欄に COBSQL の指令を指定[6]



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

⑥ [OK] ボタンを押下

16) Oracle 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[7]

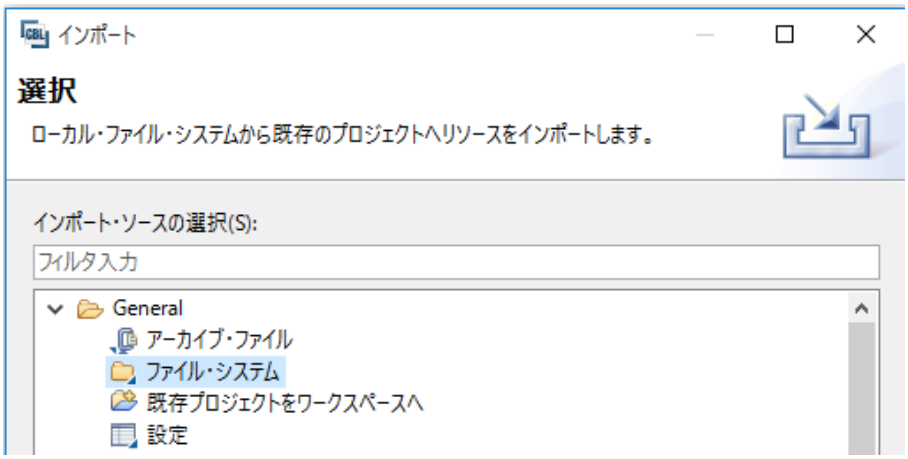
① COBOL エクスプローラにてプロジェクトを右クリックし、
[インポート] > [インポート]
を選択

② [General] > [ファイル・システム] を指定し、[次へ] ボタンを押下

---

[6] Micro Focus が提供する COBSQL を使うと Oracle Pro*COBOL をはじめとする precompiler をコンパイル命令が発行された際に内部的に実行させます。従来の precompiler を使った開発では precompiler によりプリコンパイル展開されたソースが COBOL ソースとなるため、IDE 等ではプリコンパイル展開された実際にプログラミングされたソースと異なる資産を扱う必要がありました。一方、COBSQL を活用しますと開発者は Eclipse や Visual Studio IDE 上で直接プリコンパイル前のソースを使ってメンテナンスが可能となります。

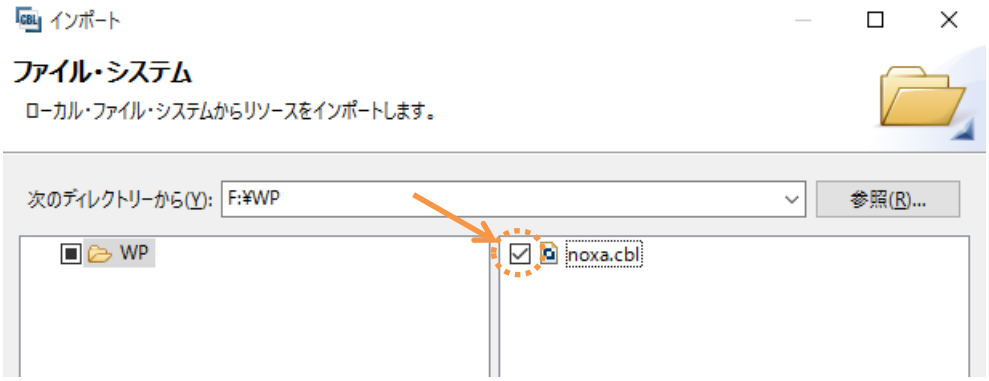
[7] 本検証で利用した実際のソースは本文書とともに公開しています。



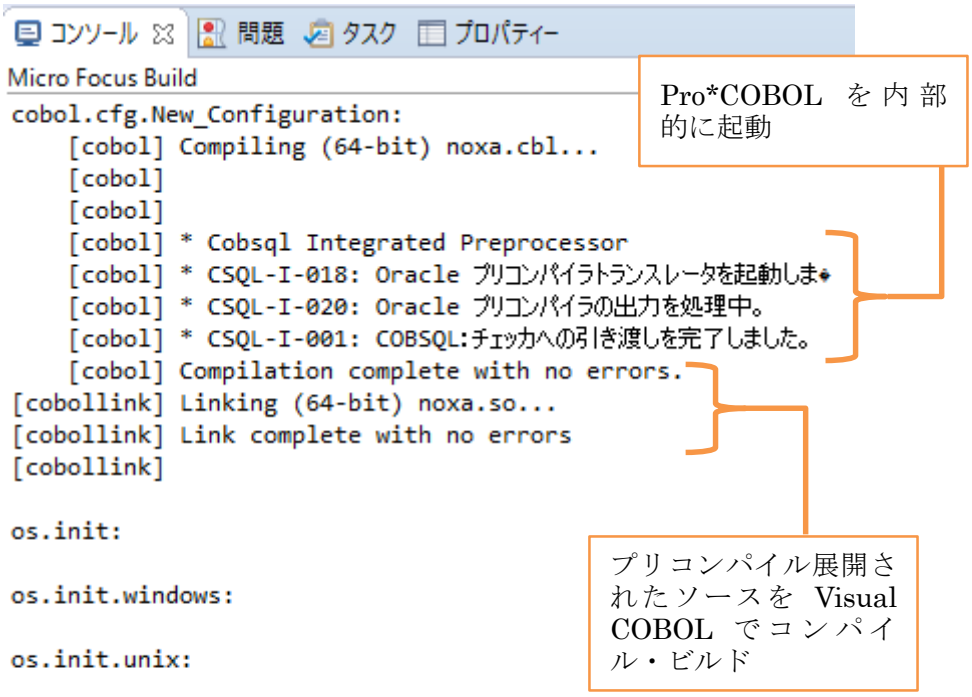
③ [参照] ボタンを押下

④ エクスプローラにて Windows 上でプログラムが格納されているフォルダまでナビゲート

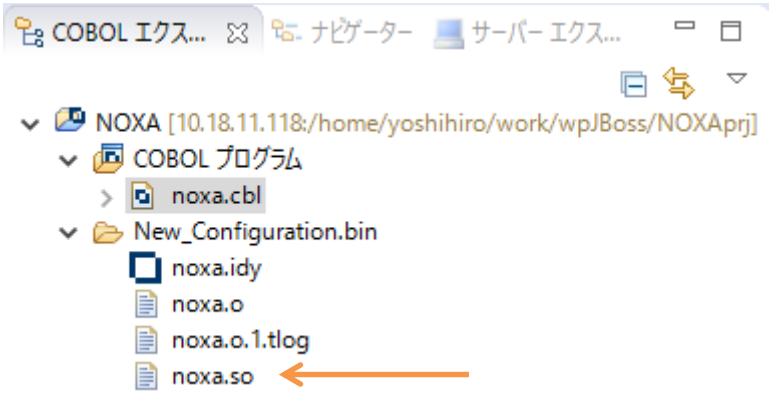
⑤ 対象のプログラム「noxa.cbl」にチェックを入れ [終了] ボタンを押下



Windows から Linux へプログラムソースがコピー転送され、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：



COBOL エクスプローラビューにて、Linux サーバ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：

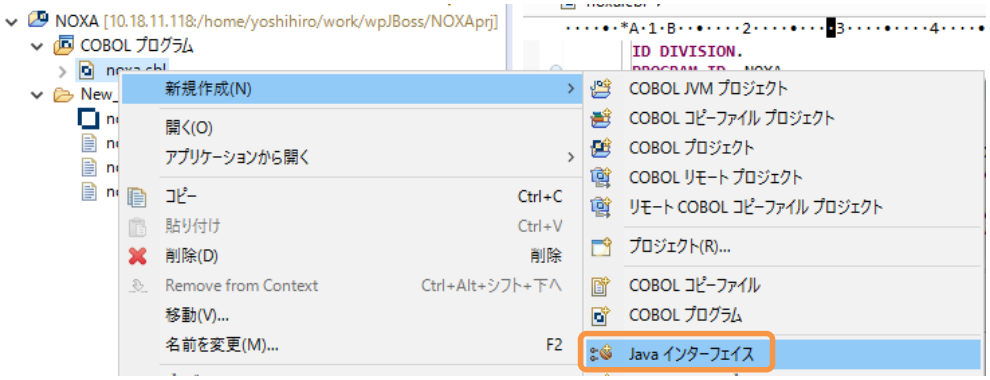


端末エミュレータでも実際に生成されていることを確認できます：

```
[yoshihiro@ym-rhel71 NOXAprj]$ pwd
/home/yoshihiro/work/wpJBoss/NOXAprj
[yoshihiro@ym-rhel71 NOXAprj]$ ls New_Configuration.bin/
noxa.idy  noxa.o  noxa.o.1.tlog  noxa.so
[yoshihiro@ym-rhel71 NOXAprj]$
```

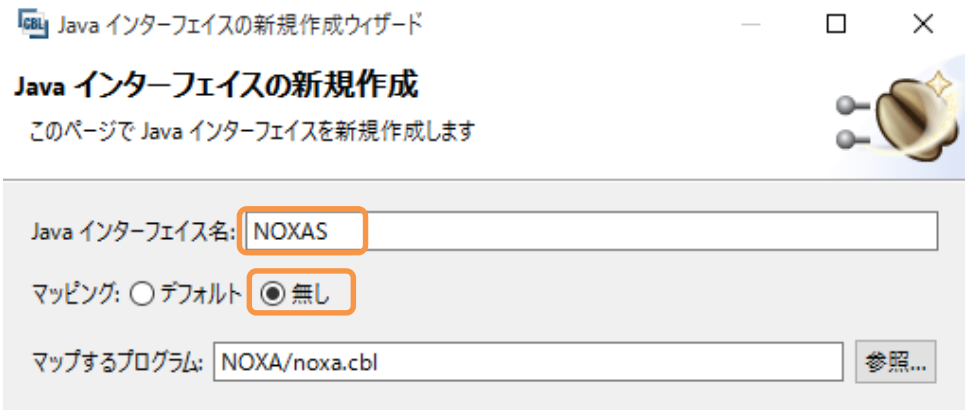
17) 検証用アプリケーションの Java インターフェースのプロファイルを作成

① COBOL エクスプローラにてプログラムを右クリックし
[新規作成] > [Java インターフェイス]
を選択



② Java インターフェイス名には「NOXAS」を指定

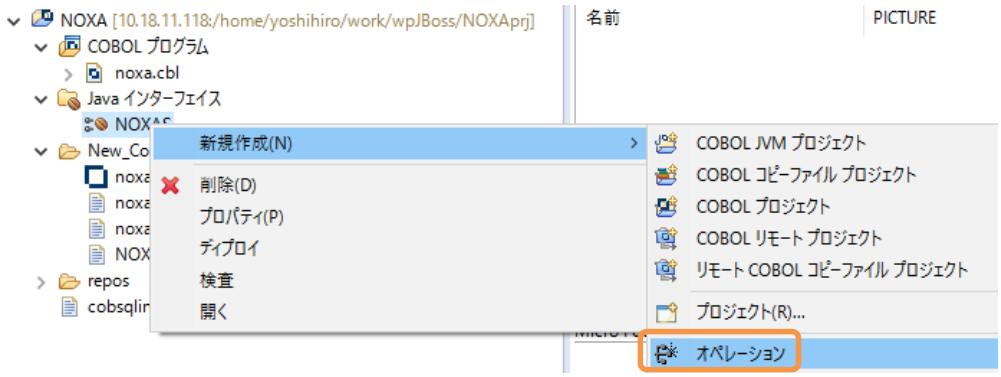
③ [マッピング] 欄では「無し」を選択



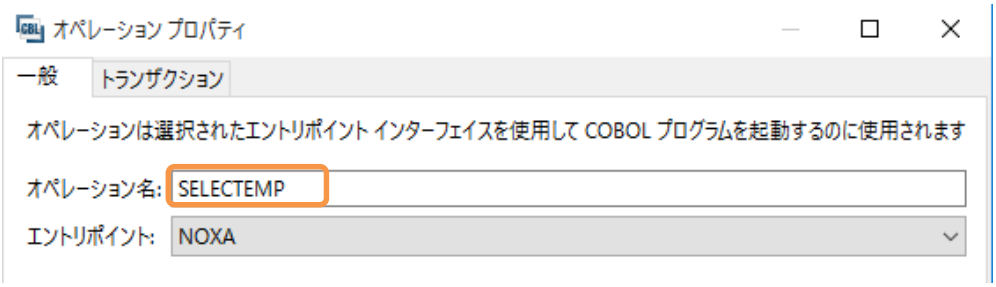
④ [終了] ボタンを押下

18) COBOL – Java 変換マッピングを生成

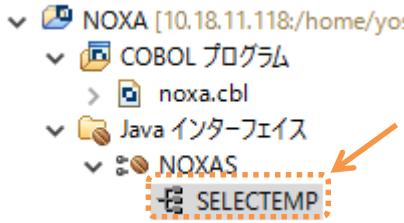
① COBOL エクスプローラにて生成された Java インターフェースプロファイルを右クリックし、
[新規作成] > [オペレーション]
を選択



② [オペレーション名] 欄にオペレーションを識別可能な値を指定し、[OK] ボタンを押下



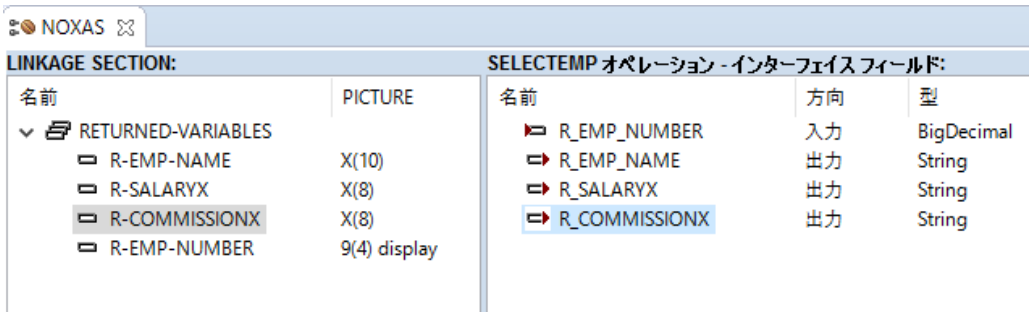
③ COBOL エクスプローラにて生成されたオペレーションをダブルクリック



④ 対象のフィールドを SELECTEMP オペレーションインターフェースフィールドへドラッグアンドドロップ

⑤ 入力・出力を変更

最終的には下図のようなインターフェースマッピングとなるよう構成しました：



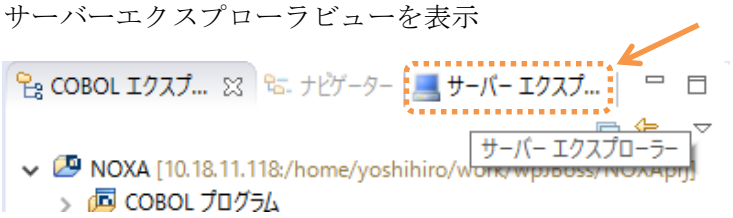
COBOL のデータ型と Java のデータ型のインターフェースマッピングを設定しています。例えば、PIC 9(4) DISPLAY として定義されている E-EMP-NUMBER は BigDecimal にマッピングしています。そのため、Java 側ではゾーン十進型を意識せず、Java の BigDecimal クラスを利用することが可能です。

ここでは、R-EMP-NUMBER を入力パラメータとして受け取り、R-EMP-NAME、R-SALARYX、R-COMMISSIONX を出力パラメータとして返すよう設定しています。

⑥ Ctrl + S を打鍵し、設定内容を保存

19) Linux サーバ上で稼動する Directory Server をサーバーエクスプローラビューへ追加

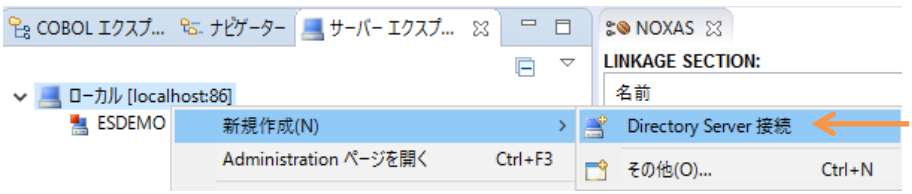
① サーバーエクスプローラビューを表示



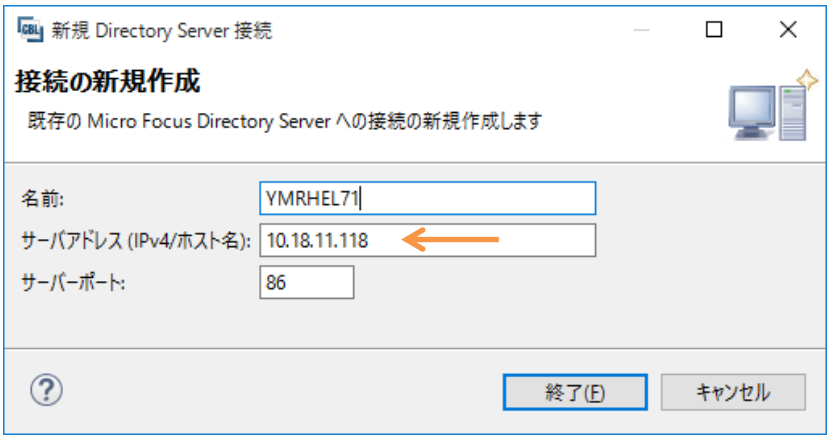
② ローカルを右クリックし

[新規作成] > [Directory Server 接続]

をクリック

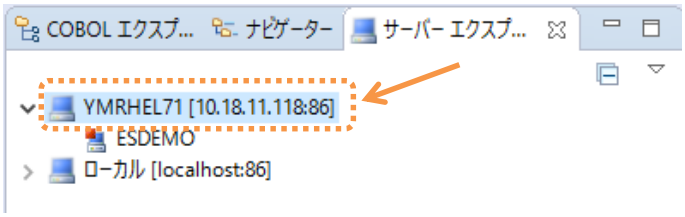


③ Linux サーバのアドレスを指定し、[終了] ボタンを押下

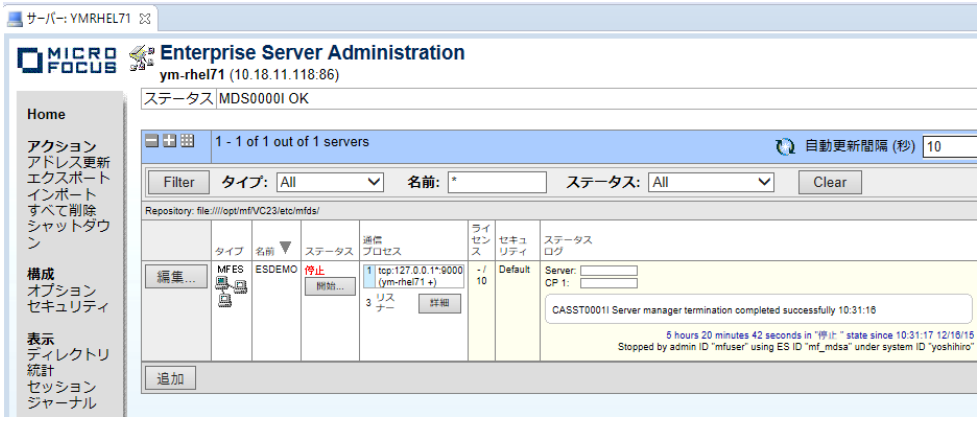


20) 64 bit 用の Enterprise Server を追加

① サーバーエクスプローラーにて追加した Linux サーバをダブルクリック



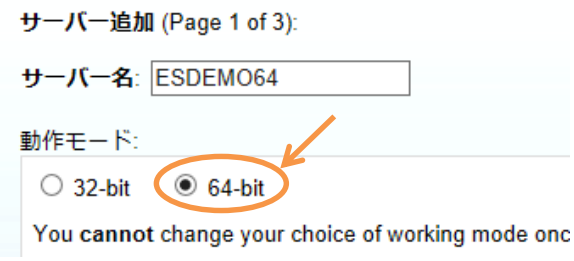
Enterprise Server Administration Console 画面が開きます：



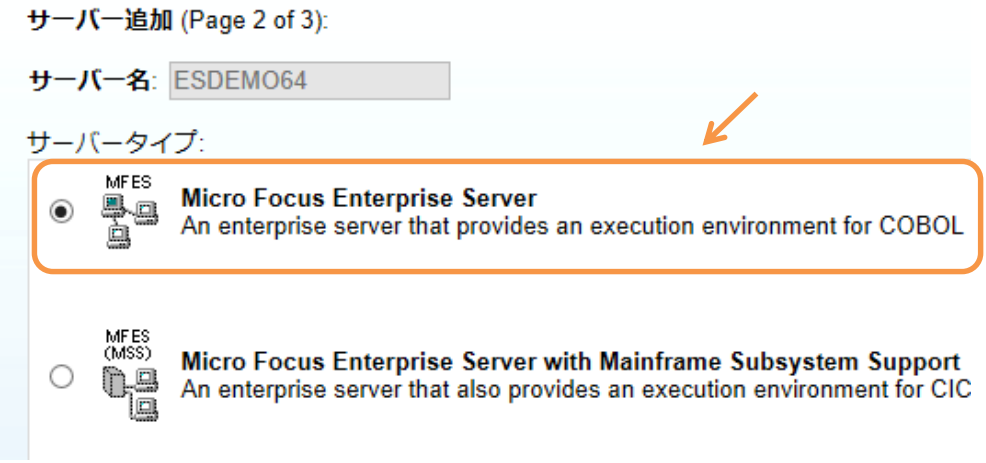
② Enterprise Server Administration Console 画面にて [追加] ボタンを押下

③ [サーバー名] 欄にて「ESDEMO64」を指定

④ [動作モード] 欄にて「64-bit」を選択し [次へ] ボタンを押下

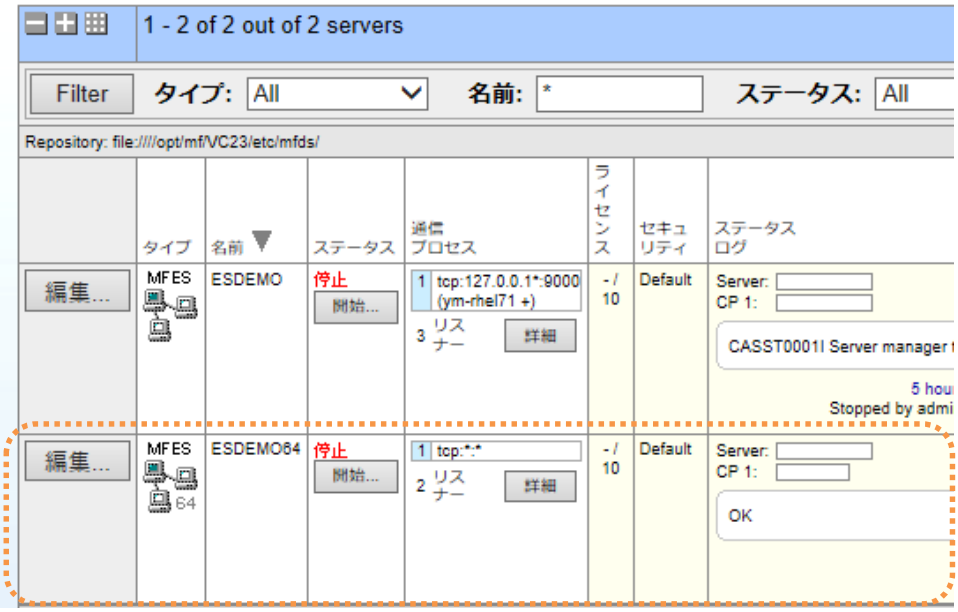


⑤ [サーバータイプ] 欄にて「Micro Focus Enterprise Server」を選択し、[次へ] ボタンを押下



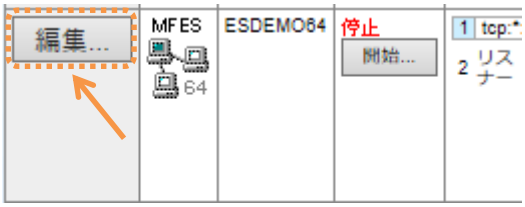
⑥ 設定内容を確認し、[追加] ボタンを押下

追加した Enterprise Server がコンソール上に表示されます：

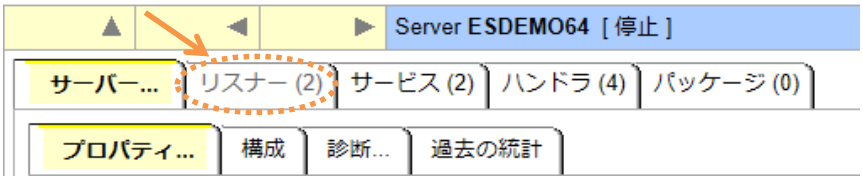


21) Enterprise Server が利用するポート番号を指定

① Enterprise Server Administration Console 上で追加した Enterprise Server 「ESDEMO64」行中の［編集］ボタンを押下

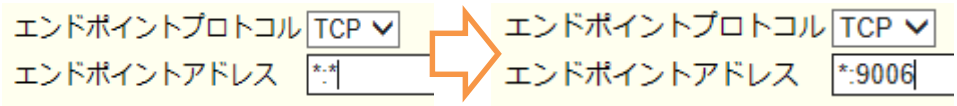


② ［リスナー］タブを選択



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

④ エンドポイントアドレスを下図のように「*:*」から「*:9006」へ変更

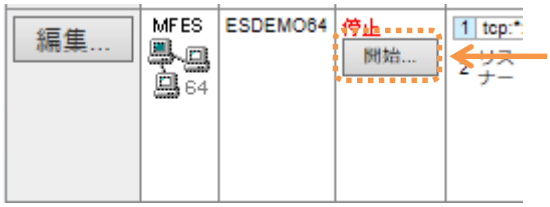


⑤ ［OK］ボタンを押下

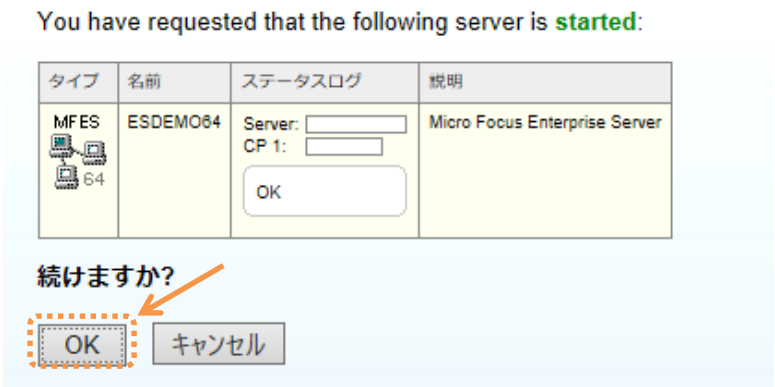
⑥ 画面左上の Home をクリック

22) Enterprise Server を起動

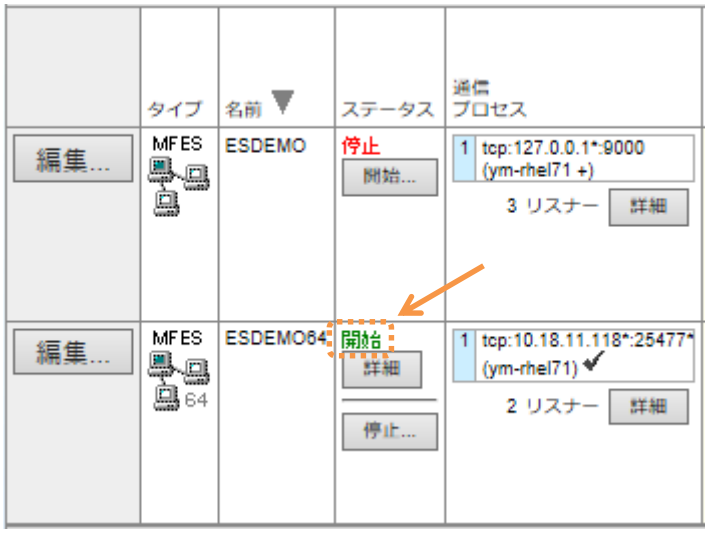
Enterprise Server Administration Console 上で追加した Enterprise Server 「ESDEMO64」行中の［開始］ボタンを押下



下図のような確認画面にて［OK］ボタンを押下



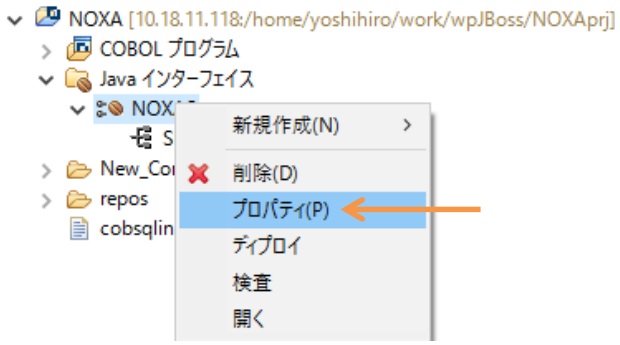
Enterprise Server が正常に起動されると、ステータス下図のように切り替わります：



23) 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

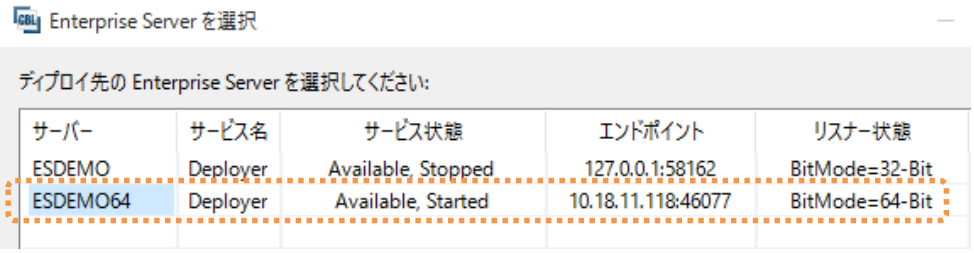
② 追加した Java インターフェイスを右クリックし［プロパティ］を選択



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 Linux サーバ上で稼動する ESDEMO64 を選択し［OK］ボタンを押下

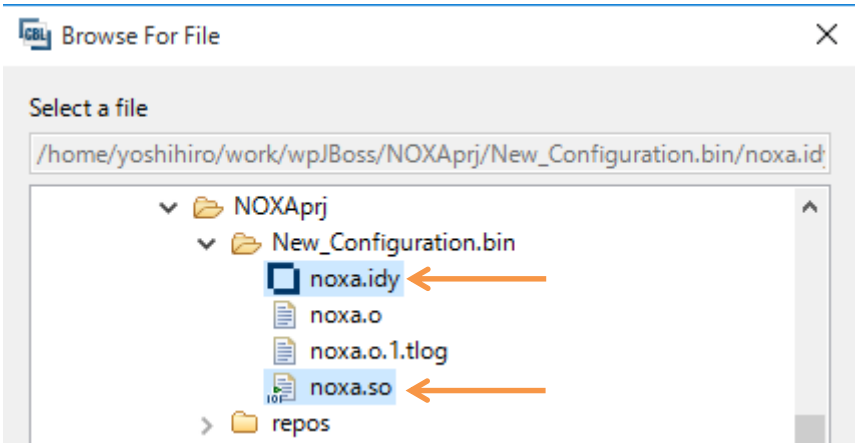


⑥ [アプリケーションファイル] タブを選択

⑦ [レガシーアプリケーションをディプロイする] を選択

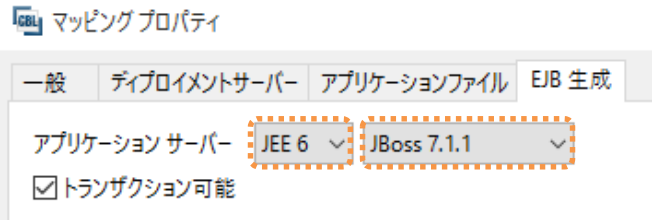
⑧ [ファイル追加] ボタンを押下

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呼び出し可能な共有オブジェクト noxa.so 及びデバッグ情報ファイル noxa.idy を選択し［OK］ボタンを押下

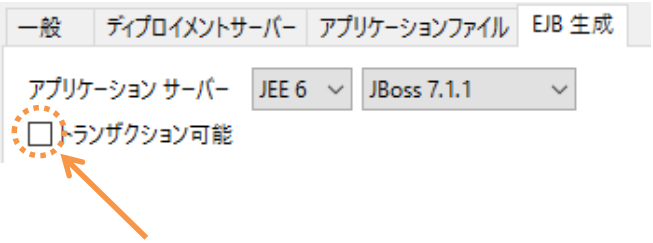


⑩ ［EJB 生成］タブを選択

⑪ ［アプリケーションサーバー］欄にて JEE 6、JBoss 7.1.1 を指定



⑫ ［トランザクション可能］のチェックを外す



⑬ [J2EE クラスパス] 欄にて [参照] ボタンを押下し下記ファイルを追加

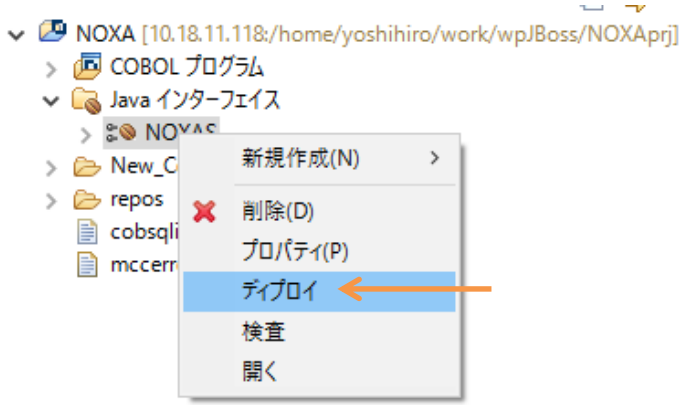
- $JBOSS_HOME/modules/system/layers/base/javax/ejb/api/main/jboss-ejb-api_3.1_spec-1.0.2.Final-redhat-3.jar
- $JBOSS_HOME/modules/system/layers/base/javax/servlet/api/main/jboss-servlet-api_3.0_spec-1.0.2.Final-redhat-2.jar
- $JBOSS_HOME/modules/system/layers/base/javax/resource/api/main/jboss-connector-api_1.6_spec-1.0.1.Final-redhat-3.jar

追加後のイメージ：

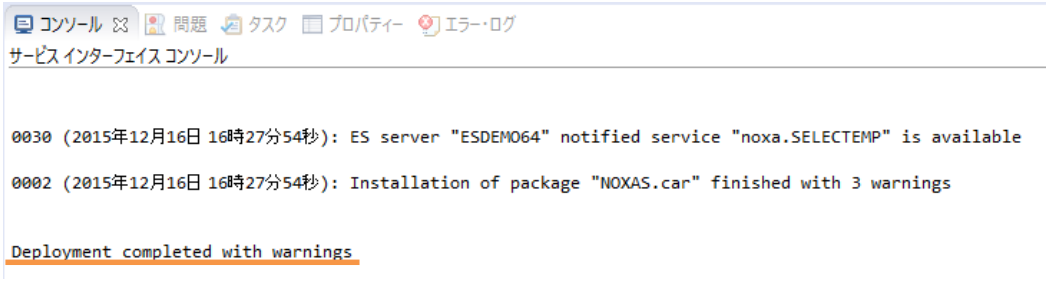
⑭ [OK] ボタンを押下し、設定画面を閉じる

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

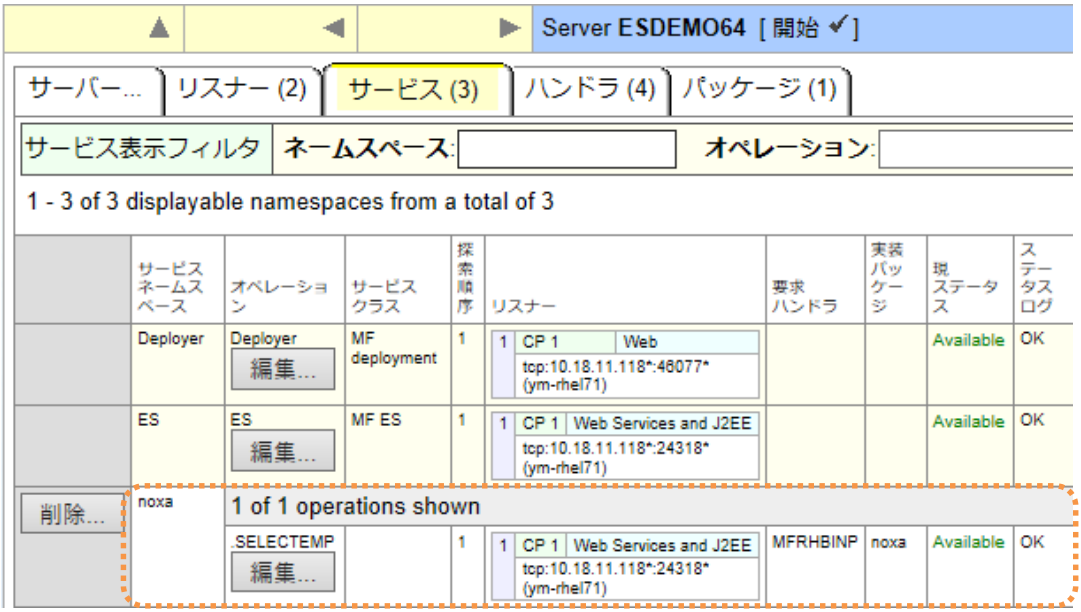
COBOL エクスプローラにて用意した Java インターフェイスを右クリックし、[ディプロイ] を選択



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：

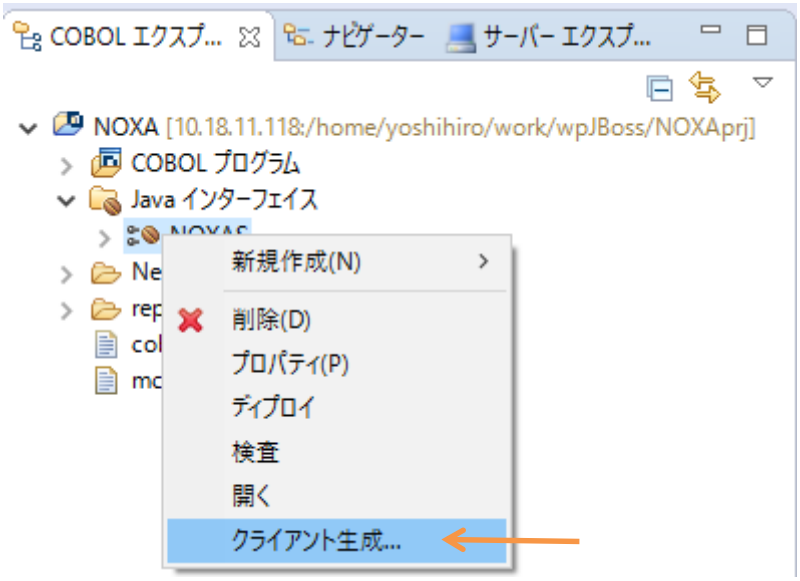


同様に Enterprise Server Administration Console 画面においても
[詳細] ボタン > [サービス] タブ
へとナビゲートして表示される画面においても確認ができます：



25) 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

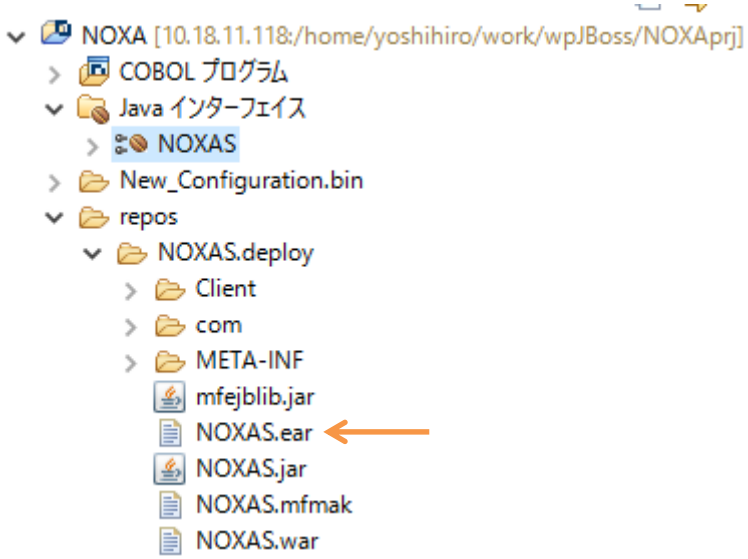
COBOL エクスプローラにて Java インターフェイスを右クリックして［クライアント生成］を選択



正常に処理されると

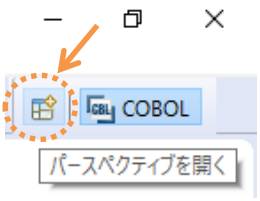
＜プロジェクトディレクトリ＞/repos/＜サービス名＞.deploy

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：

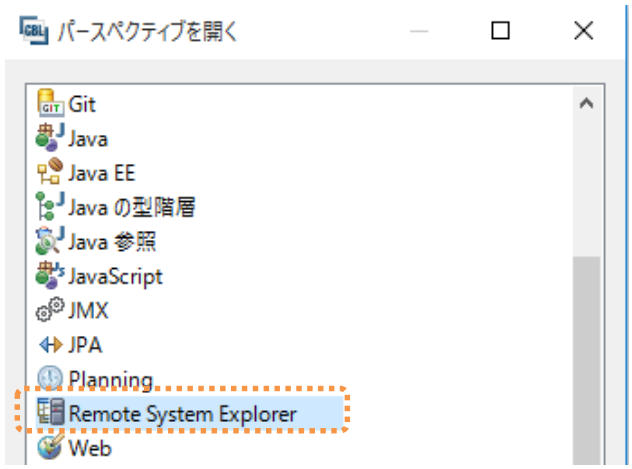


26) 生成された Java EE アプリケーションを JBoss EAP 6.4 へディプロイ

① 画面右上の［パースペクティブを開く］アイコンをクリック



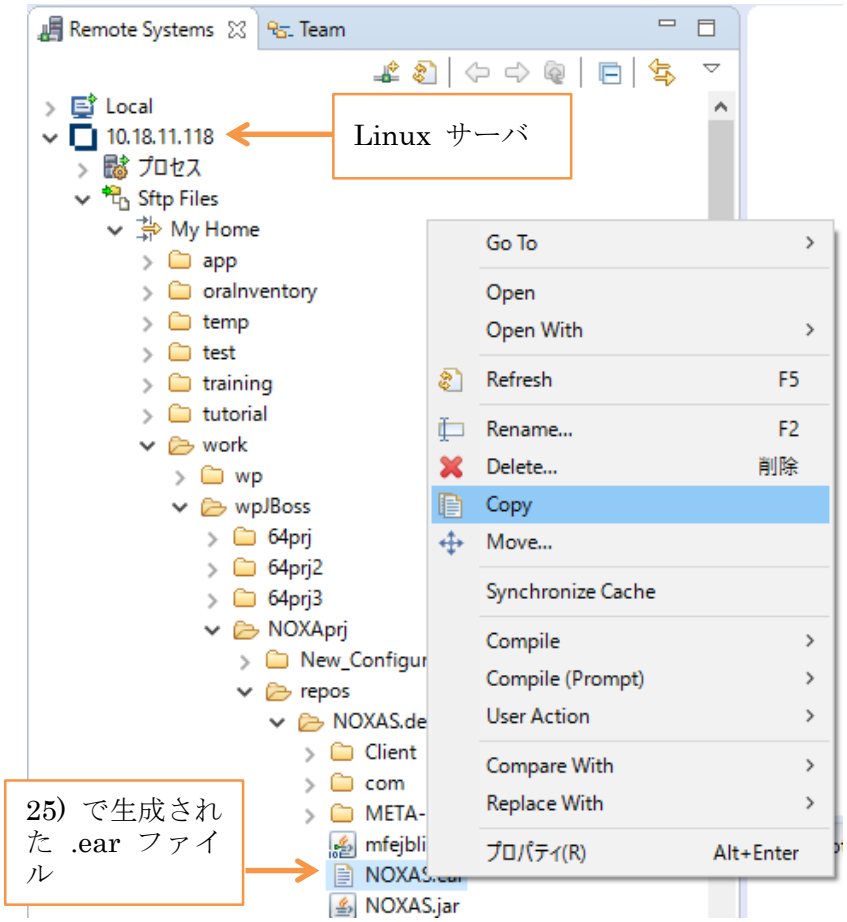
② ［Remote System Explorer］を選択し［OK］ボタンを押下



③ [Remote System] ビューにて

＜Linux サーバ＞ → Sftp Files

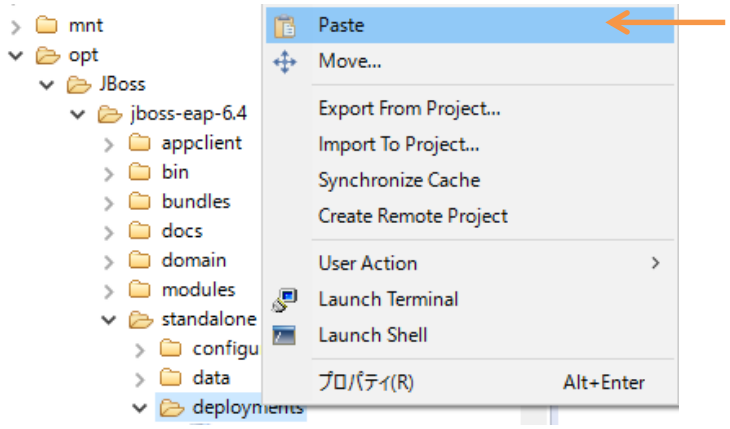
配下にある生成された .ear を右クリックし、[Copy] を選択



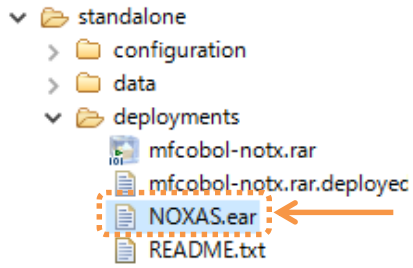
④ 同ビューにて

$JBOSS_HOME/standalone/deplyoments

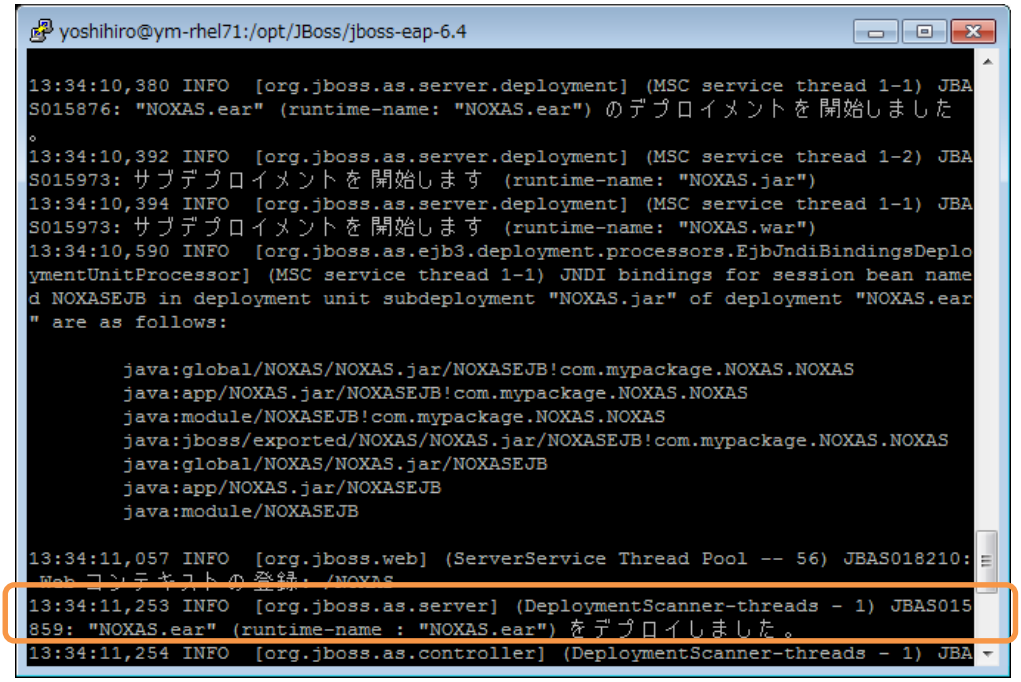
ディレクトリを右クリックし、[Paste] を選択



$JBOSS_HOME/standalone/deplyoments ディレクトリに .ear が追加されます：

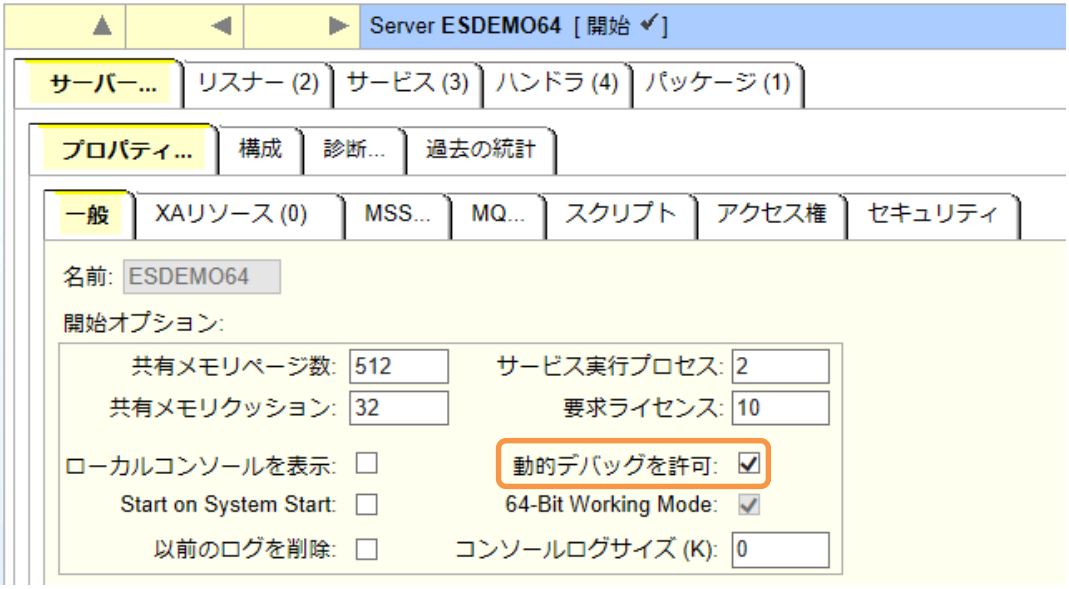


正常にディプロイされたことを JBoss を起動した Linux のターミナルからも確認ができます：



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

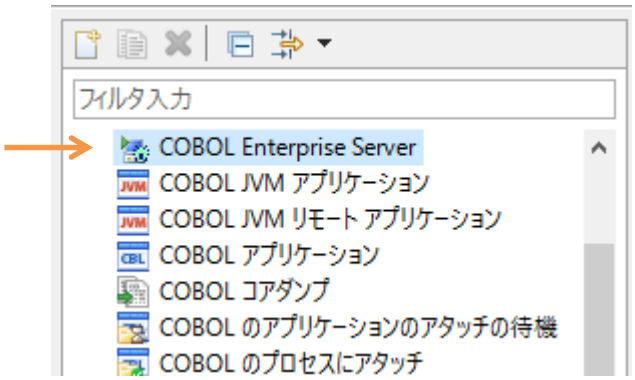
Enterprise Server Administration 画面を開き、ESDEMO64 の行における [編集] ボタンを押下し、確認[8]



28) Enterprise Server デバッグを起動

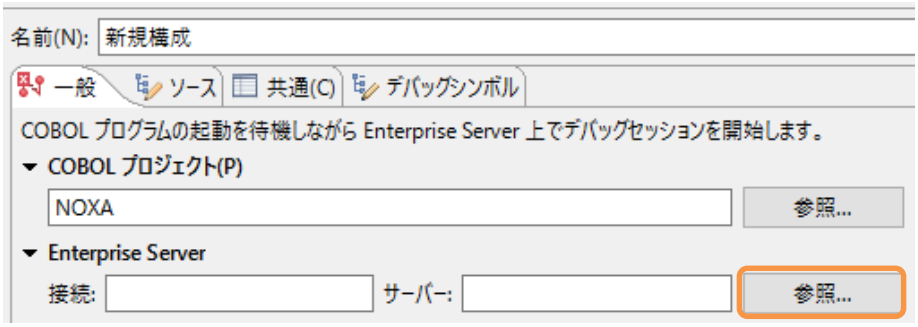
① COBOL エクスプローラにてプロジェクトを右クリックし
[デバッグ] > [デバッグの構成]
を選択

② [COBOL Enterprise Server] をダブルクリック



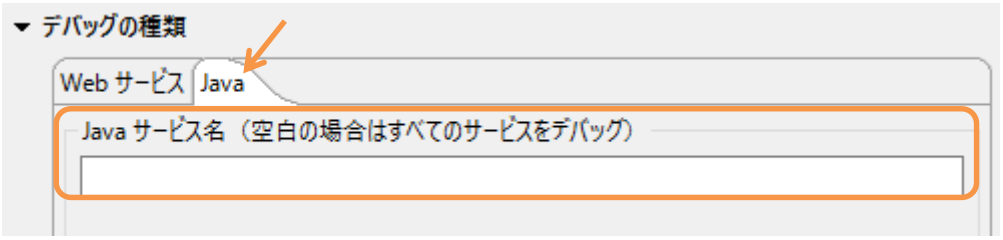
[8] 本オプションはデフォルトではチェックが入っていません。チェックが入っていない場合は、Enterprise Server を停止し、オプションをチェックの上、起動します。

③ [Enterprise Server] 欄の［参照］ボタンを押下



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

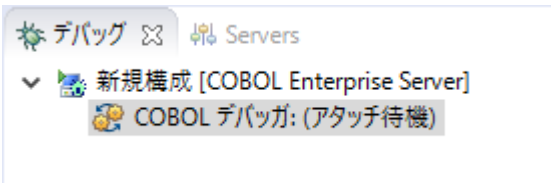
⑤ [Java] タブをクリックし、全てのサービスがデバッグ対象となっていることを確認



⑥ [デバッグ] ボタンを押下

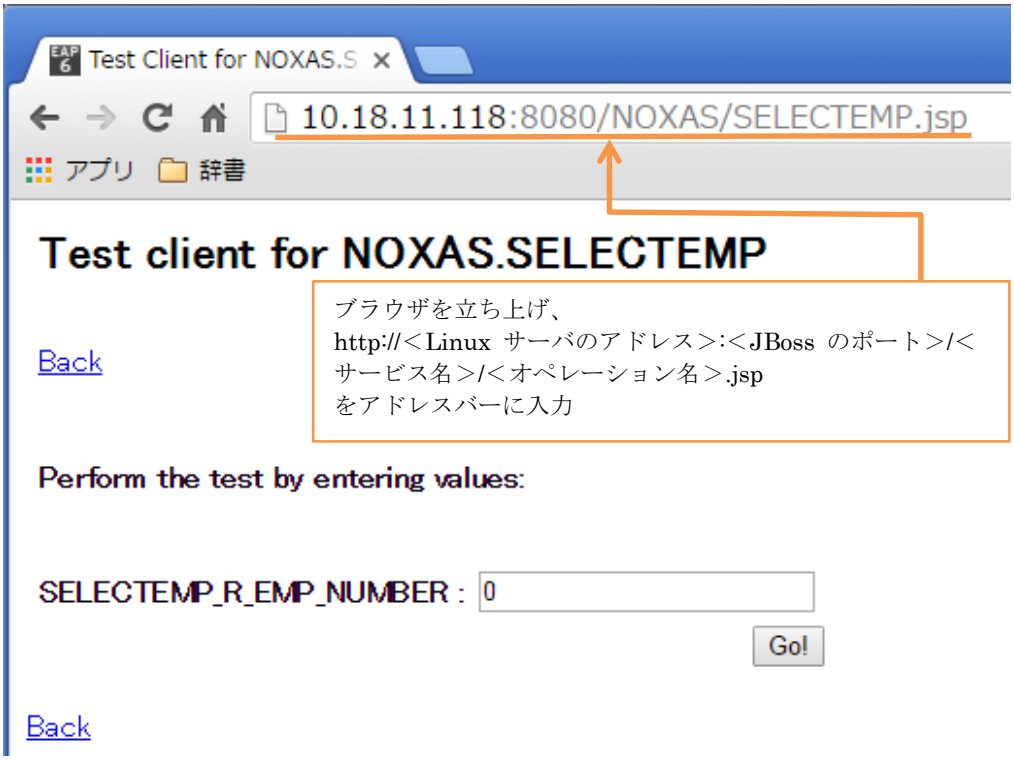
⑦ [パースペクティブの切り替えの確認] のプロンプトに対しては［はい］を選択

⑧ 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：

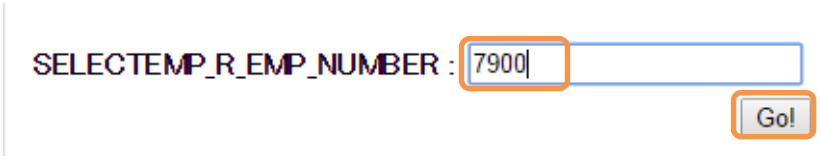


29) ディプロイしたアプリケーションをデバッグ実行

① JBoss 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

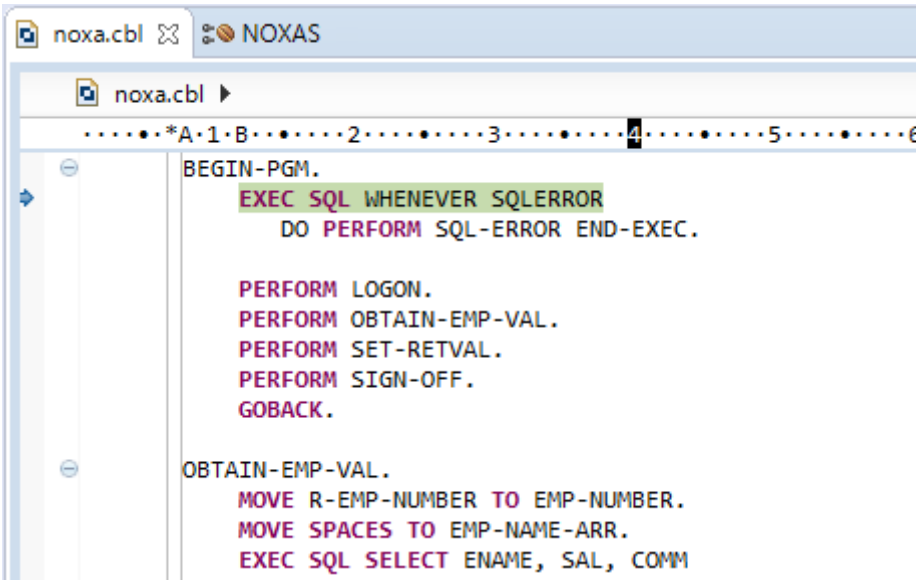


② [SELECTEMP_R_EMP_NUMBER] 欄に [7900] を入力し [Go!] ボタンを押下



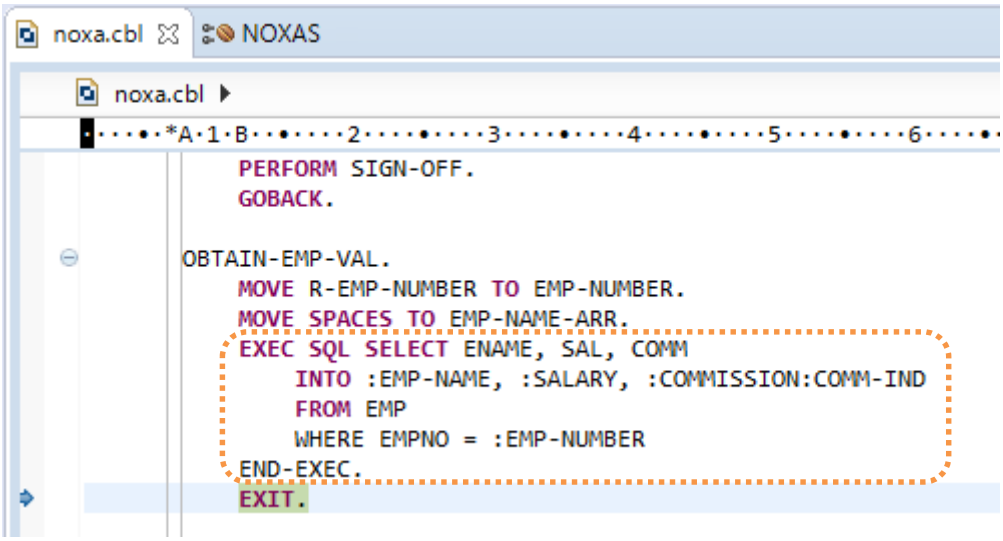
処理が Enterprise Server に渡り、Eclipse のデバッガがその処理を引き込みます：

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一時停止していることが確認できます：

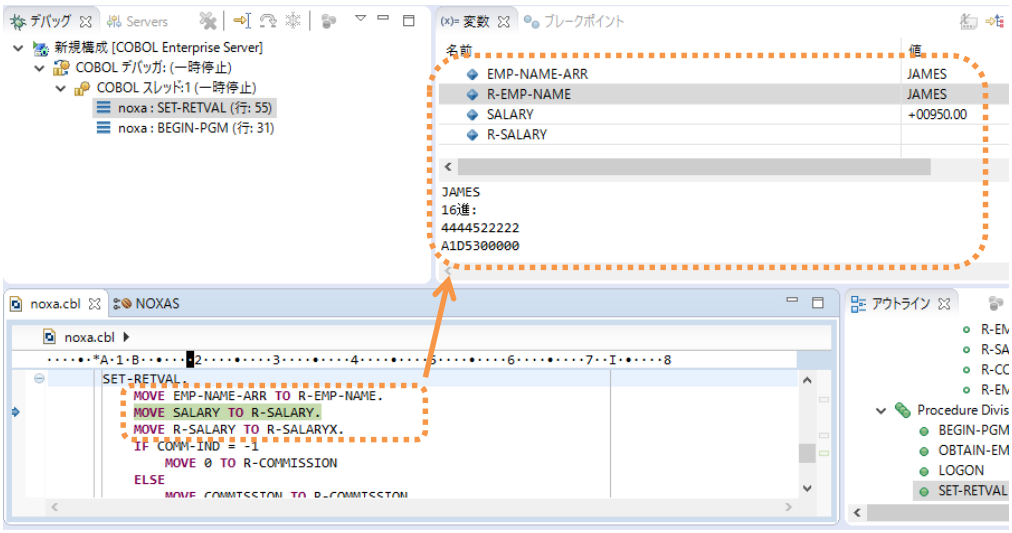


③ ディプロイしたアプリケーションをデバッグ実行

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プリコンパイル前のソースでデバッグができます：

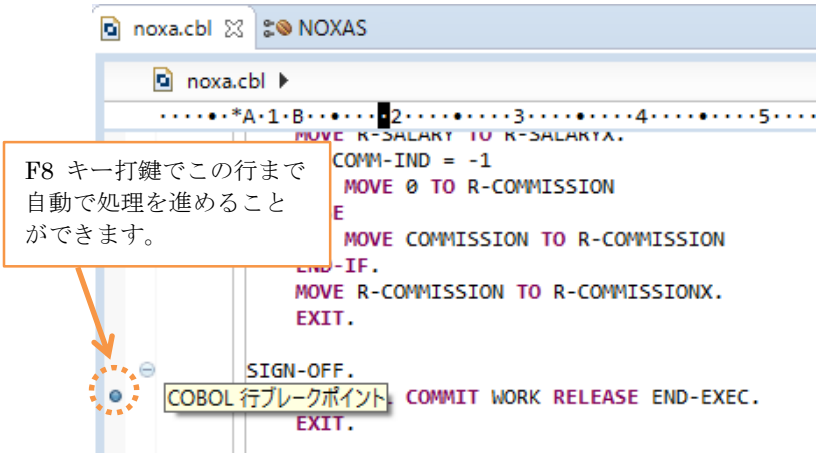


変数ビューで現在のステップで参照している変数の中身を確認できます：



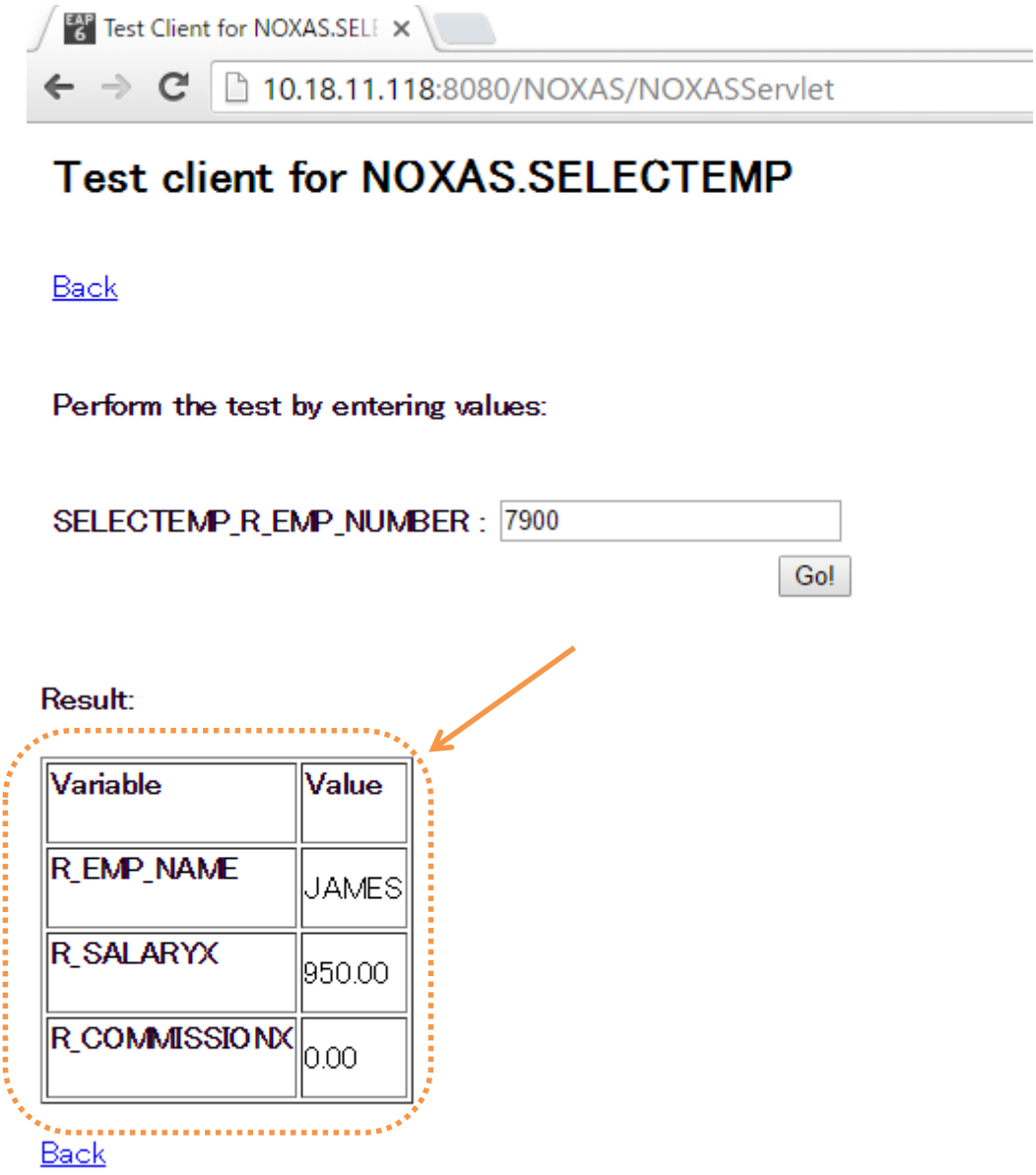
データの中身は 16 進表示にすることもできます：

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：

⑤ 処理を最後まで進めると、COBOL が Oracle より取得したデータが表示され、Java – COBOL – Oracle が正常に連携できていることを確認できます：



30) JBoss EAP、Enterprise Server、デバッグ、リモート開発デーモンを停止

付録２に進む前に JBoss サーバ、Enterprise Server、デバッグ実行、リモート開発用のデーモンそれぞれを停止します。

付録 2. Oracle 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

■ Linux サーバ

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

2) Oracle の XA スイッチモジュールを作成

① 作業用のディレクトリを用意

② Visual COBOL にビルドインされるスイッチモジュールのビルド用リソースを①で用意したディレクトリへコピー

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

③ ②でコピーしたリソースに含まれるビルド用スクリプトを使って Oracle 用のスイッチモジュールを生成

```
$ ./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す。
* 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す。
* 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.
$
```

下記の 2 ファイルが生成されます[9]：

```
$ ls *.so
ESORAXA64.so  ESORAXA64_D.so
$
```

[9] 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。

3) リモート開発用のデーモンを起動

① root 権限を持ったユーザへ切り替え

② リモート開発用デーモンの起動スクリプトを実行

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

4) JBoss の設定ファイルを XA 用のリソースアダプタ向けに編集

① 別セッションにて Linux サーバへ一般ユーザでログイン

② Visual COBOL のマニュアルに従い、standalone.xml 中の subsystem タグを編集

付録 1 では、mfcobol-notx.rar を利用する前提で編集していますが、ここでは、mfcobol-xa.rar を利用する場合の内容に編集します。

5) リソースアダプタ mfcobol-xa.rar を ESDEMO64 にポイントするよう編集

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: jboss7
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
```

JBoss 7.1.1 向けの mfcobol-xa.rar を指定します。

```
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
#
```

ポート 番号を 9006 へ変更します。

④ 一般ユーザに戻る

6) リソースアダプタ mfcobol-xa.rar を JBoss EAP 6.4 へディプロイ

① 付録 1 で使用した mfcobol-notx.rar を JBoss EAP 6.4 のディプロイメントディレクトリより削除

```
$ rm $JBOSS_HOME/standalone/deployments/mfcobol-notx.rar
$
```

② 編集した mfcobol-xa.rar を JBoss EAP 6.4 へディプロイ

```
$ cp $COBDIR/javaee/javaee6/jboss7/mfcobol-xa.rar $JBOSS_HOME/standalone
/deployments
$
```

7) JBoss を起動

```
$ $JBOSS_HOME/bin/standalone.sh -b 0.0.0.0
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /opt/JBoss/jboss-eap-6.4

  JAVA: java

                                        :

17:24:15,390 INFO  [org.jboss.as.connector.deployment] (MSC service thread 1
-1) JBAS010406: 接続ファクトリ java:/eis/MFCobol_v1.5 を登録しました。
17:24:15,394 WARN  [org.jboss.as.connector.deployers.RaXmlDeployer] (MSC ser
vice thread 1-1) IJ020016: Missing <recovery> element. XA recovery disabled
for: java:/eis/MFCobol_v1.5
17:24:15,400 INFO  [org.jboss.as.connector.deployers.RaXmlDeployer] (MSC ser
vice thread 1-1) IJ020002: Deployed: file:/opt/JBoss/jboss-eap-6.4/standalon
e/tmp/vfs/temp/tempe1e8aeb8ebc77711/mfcobol-xa.rar-b093e3385cb7ce10/contents
/
```

```
17:24:15,406 INFO  [org.jboss.as.connector.deployment] (MSC service thread 1
-1) JBAS010401: JCA ConnectionFactory [java:/eis/MFCobol_v1.5] をバインドし
ました。
17:24:15,522 INFO  [org.jboss.as.server] (ServerService Thread Pool -- 28) J
BAS015859: "mfcobol-xa.rar" (runtime-name : "mfcobol-xa.rar") をデプロイしま
した。
17:24:15,599 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015961: http:
//127.0.0.1:9990/management 上でリッスンする HTTP 管理インターフェース
17:24:15,600 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015951: 管理
コンソールは http://127.0.0.1:9990 をリッスンしています。
17:24:15,600 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss
 EAP 6.4.0.GA (AS 7.5.0.Final-redhat-21) は 7305ms で開始しました - サービ
ス 230 個のうち 192 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)。
```

リソースアダプタがデプロイされたことが確認できます。

JBoss EAP 6.4 が正しく起動できたことが確認できます。

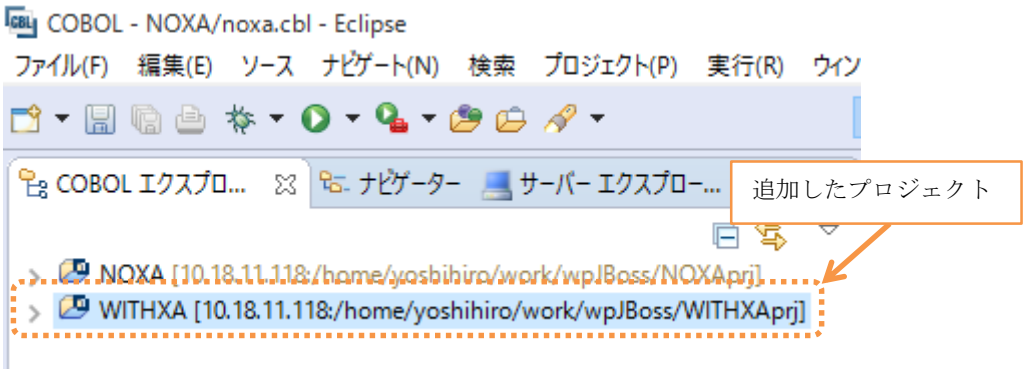
■ Windows クライアントマシン

8) Visual COBOL for Eclipse を起動

9) 付録 1 で使用した Eclipse ワークスペースを開く

10) 付録 2 用のリモートプロジェクトを作成

付録 1 の要領で付録 2 の検証用のリモートプロジェクトワークスペースに追加

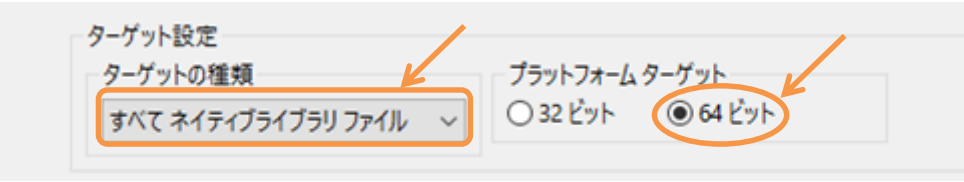


11) プロジェクト構成を指定

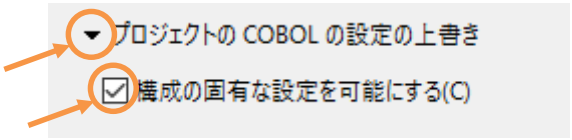
① プロジェクトを右クリックして [プロパティ] を選択

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

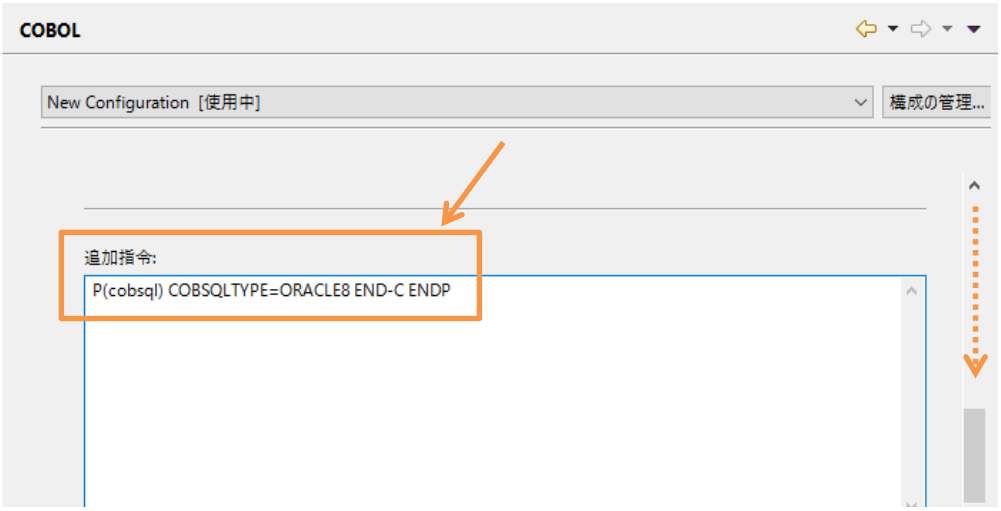
③ [ターゲットの種類] 欄にて「すべてネイティブライブラリファイル」を、[プラットフォームターゲット] 欄にて「64 ビット」を選択



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



⑤ 画面を下にスクロールし、[追加指令] 欄に COBSQL の指令を指定

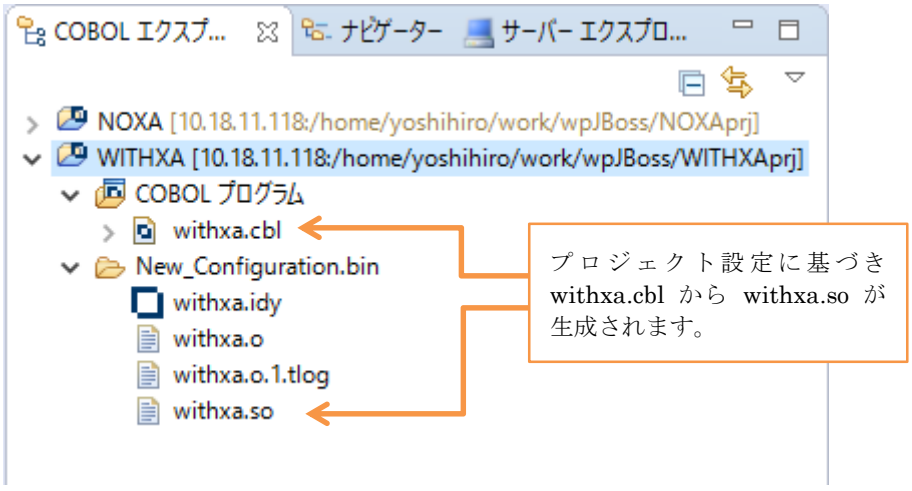


⑥ [OK] ボタンを押下

12) プログラムをプロジェクトへ追加

本文書に添付された withxa.cbl を付録 1 の要領でプロジェクトへ追加

追加後のイメージ：



追加されたプログラム：

noxa.cbl　NOXAS　withxa.cbl

withxa.cbl

```
13       LINKAGE SECTION.
14       01  L-EMP-NAME               PIC X(10).
15       01  L-EMP-NUMBER             PIC S9(4) COMP-5.
16       01  L-COMMIT-OR-ROLLBACK     PIC X.
17       01  L-LOG-MSG                PIC X(80).
18       PROCEDURE DIVISION USING L-EMP-NAME L-EMP-NUMBER
19                                L-COMMIT-OR-ROLLBACK L-LOG-MSG.
20       BEGIN-PGM.
21           PERFORM OBTAIN-ENAME.
22           PERFORM UPDATE-ENAME.
23           PERFORM COMMIT-OR-ROLLBACK.
24           GOBACK.
25
26       OBTAIN-ENAME.
27           MOVE L-EMP-NUMBER TO EMP-NUMBER.
28           EXEC SQL
29               SELECT ENAME INTO :EMP-NAME FROM EMP
30               WHERE EMPNO = :EMP-NUMBER
31           END-EXEC.
32           EXIT.
33           MOVE EMP-NAME TO TMP-ENAME.
34           EXIT.
35
36       UPDATE-ENAME.
37           MOVE L-EMP-NAME TO EMP-NAME.
38           EXEC SQL
39               UPDATE EMP SET ENAME = :EMP-NAME
40               WHERE EMPNO = :EMP-NUMBER
41           END-EXEC.
42           EXIT.
43           STRING "ENAME CHANGED FROM " TMP-ENAME "TO " EMP-NAME
44           INTO L-LOG-MSG.
45           EXIT.
46
47       COMMIT-OR-ROLLBACK.
48           IF L-COMMIT-OR-ROLLBACK = 'R'
49             SET IDX TO 11
50             MOVE SPACE TO TABLE-ITEM(IDX)
51           END-IF.
52           EXIT.
```

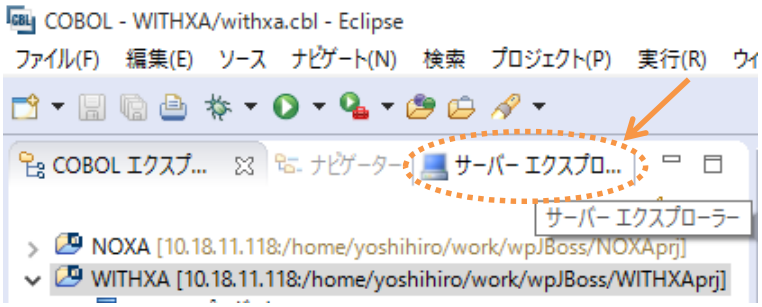
※1　Oracle Database へ XA スイッチモジュールを経由して接続するため、プログラム中には CONNECT 文等の接続関連の命令は記述しません。

※2　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。

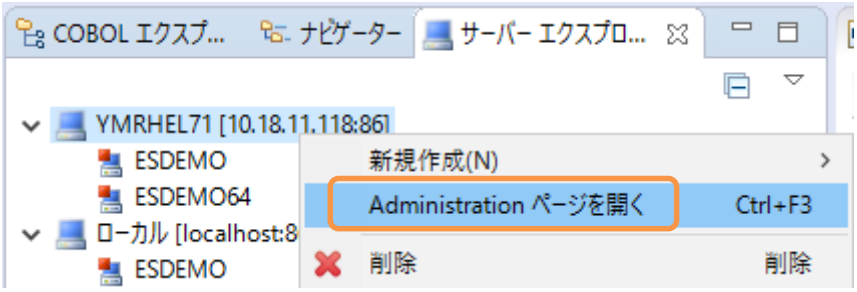
※3　LINKAGE パラメータ　L-COMMIT-OR-ROLLBACK に「R」が渡されると意図的に添え字範囲外の実行時エラーを発生させます。

13) Enterprise Server にスイッチモジュールを追加

① [サーバーエクスプローラー] を選択

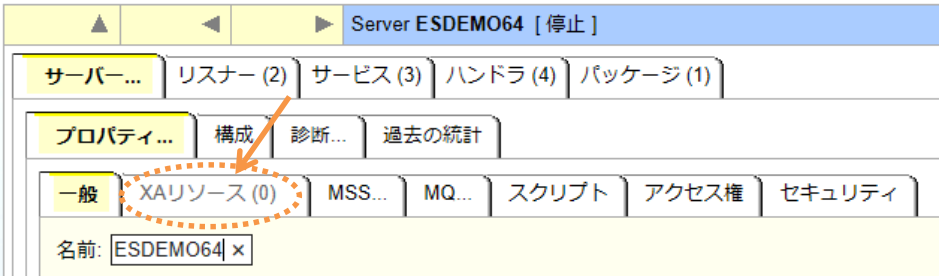


② Linux サーバを右クリックし、[Administration ページを開く] を選択

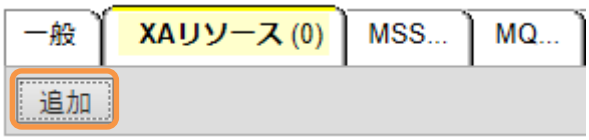


③ ESDEMO64 の列で [編集] ボタンを押下

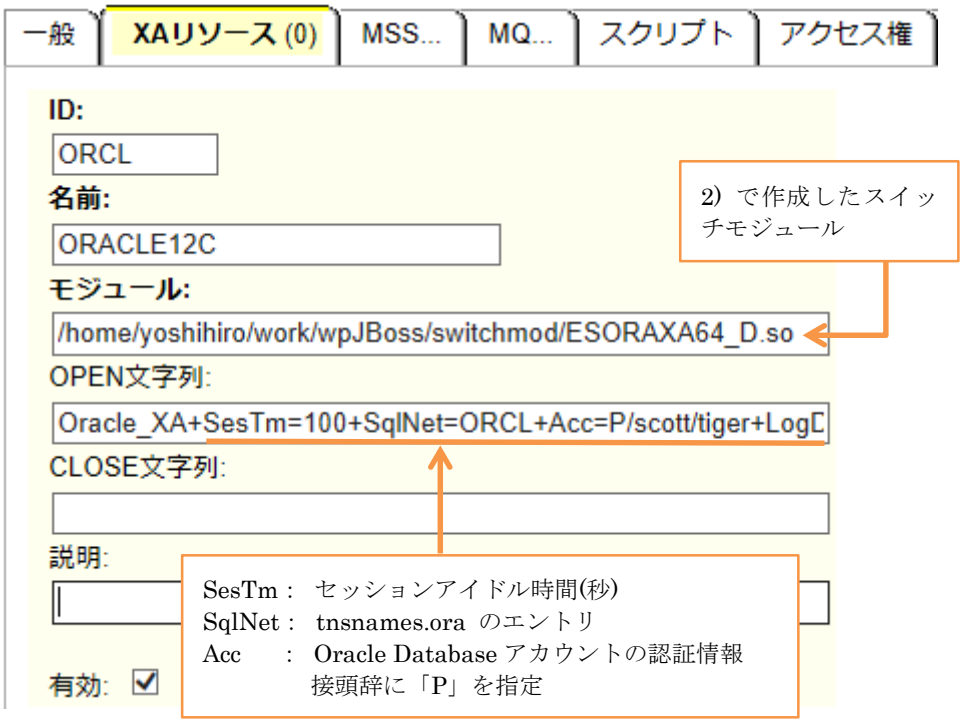
④ [XA リソース] タブをクリック



⑤ [追加] ボタンを押下

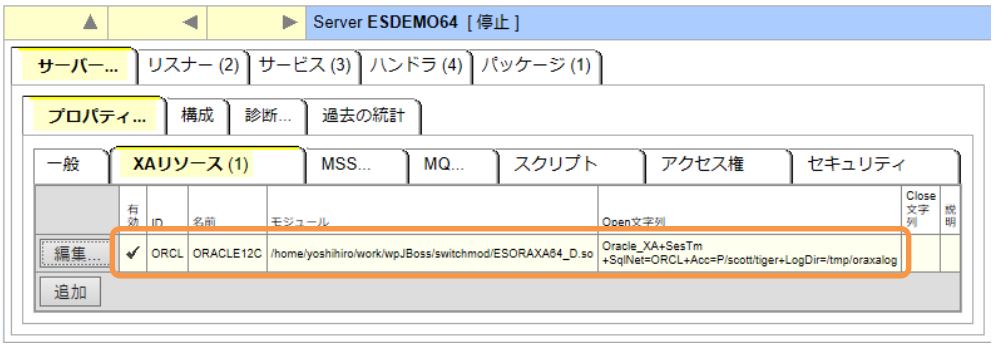


⑥ Open 文字列等 XA リソースに関する情報を指定[10]



⑦ [追加] ボタンを押下

⑧ XA リソースが追加されたことを確認できます：

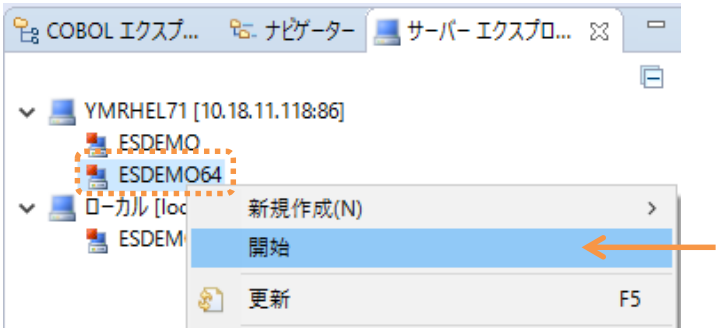


⑨ [Home] をクリック

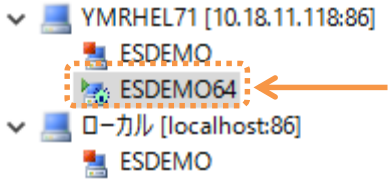
[10] [OPEN 文字列] 欄への設定値に関する詳細は Oracle のドキュメント等を参照してください。

14) Enterprise Server を起動

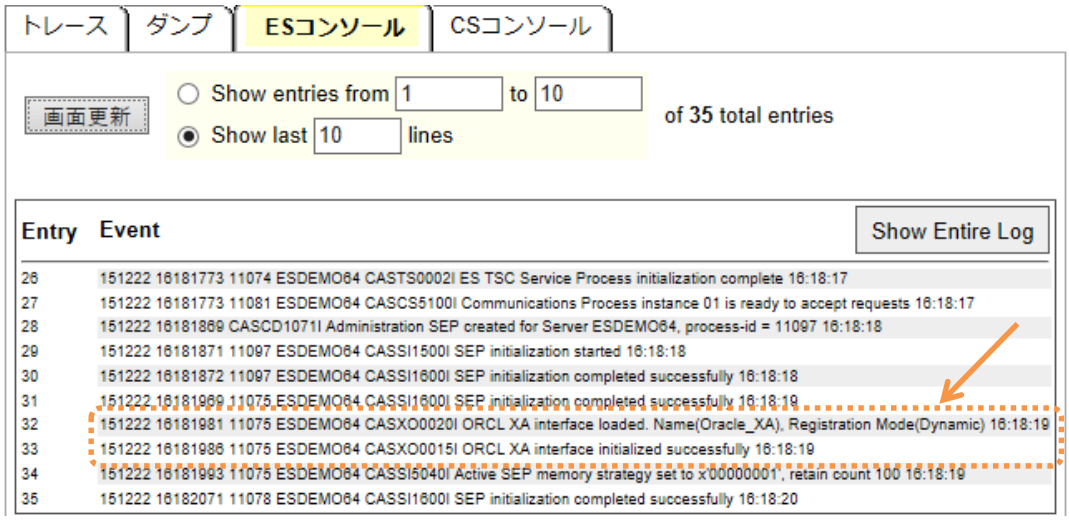
サーバーエクスプローラーにて［ESDEMO64］を右クリックし、［開始］を選択



正常に起動されると ESDEMO64 のアイコンが緑色に切り替わります：



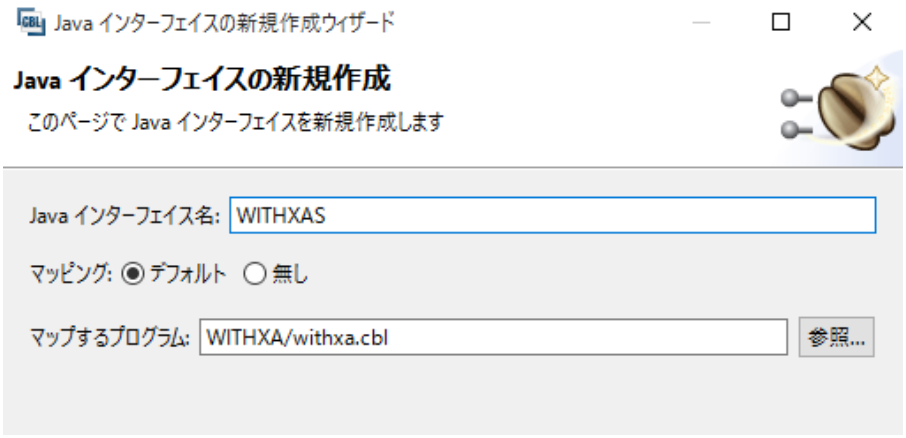
XA スイッチモジュールが正しく構成され、動作することを Enterprise Server のコンソールログより確認できます。下図は Enterprise Server Administration 画面にて、［編集］ボタン >［診断］タブ >［ES コンソール］
とナビゲートし表示したコンソール画面になります：



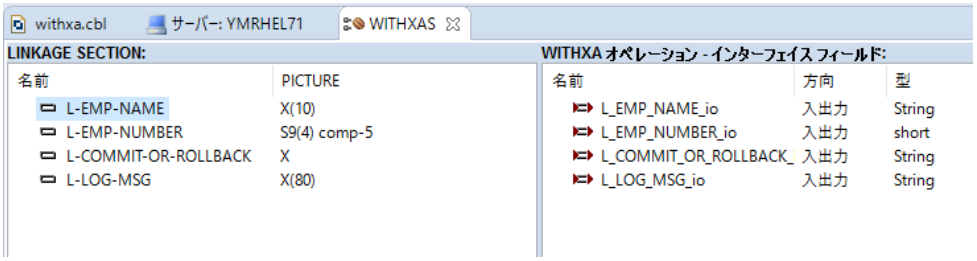
15) アプリケーションの COBOL – Java 変換マッピングを作成

① COBOL エクスプローラにて withxa.cbl を右クリックし
[新規作成] > [Java インターフェイス]
を選択

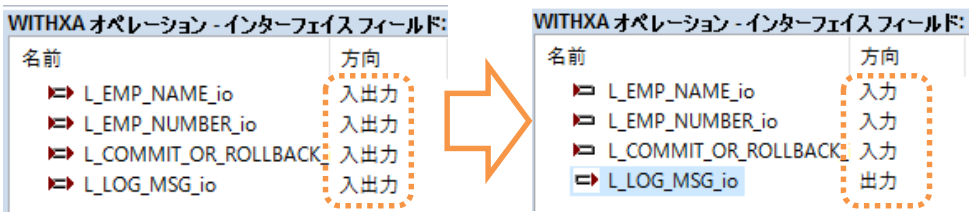
② [Java インターフェイス名] 欄を指定し、[終了] ボタンを押下



デフォルトのインターフェースマッピングが生成されます：



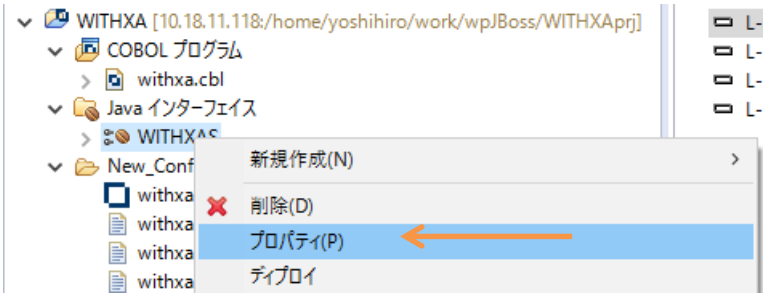
③ Java 側のインターフェースフィールドの方向を変更



④ CTRL + S を同時打鍵し、変更を保存

16) Enterprise Server へのディプロイ情報を指定

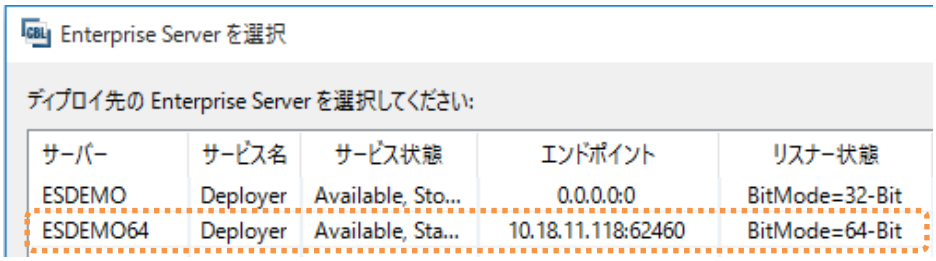
① COBOL エクスプローラにて追加した Java インターフェイスを右クリックし［プロパティ］を選択



② ［ディプロイメントサーバー］タブを選択

③ ［変更］ボタンを押下

④ Linux サーバで稼動する 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

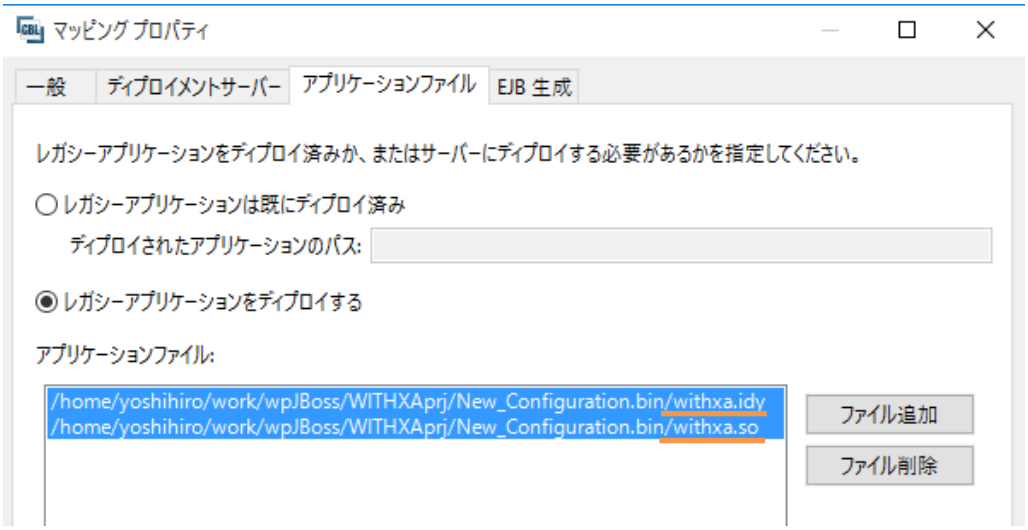
⑥ ［アプリケーションファイル］タブを選択

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 withxa.so 及び withxa.idy を選択し [OK] ボタンを押下

追加後の画面：



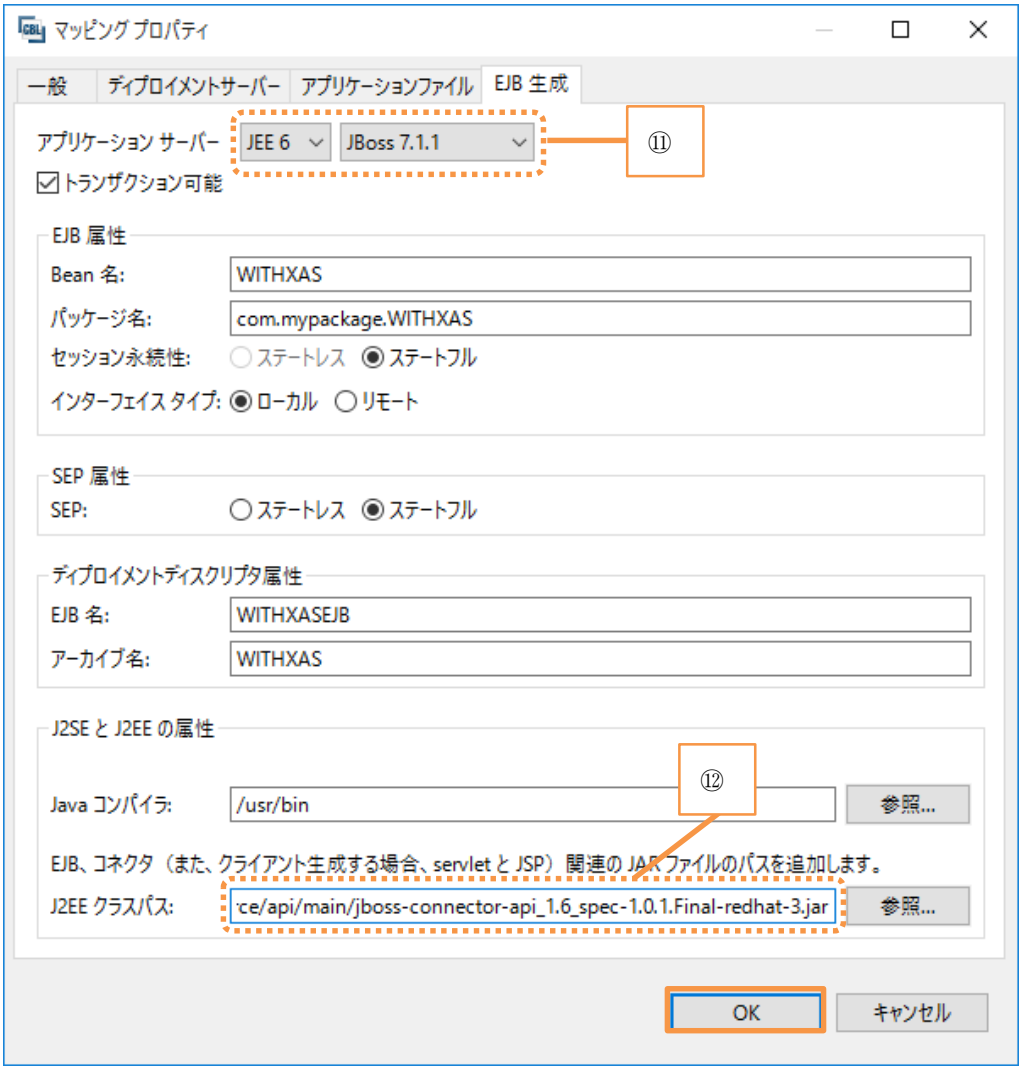
⑩ [EJB 生成] タブを選択

⑪ [アプリケーションサーバー] 欄にて JEE 6、JBoss7.1.1 を指定

⑫ [J2EE クラスパス] 欄にて [参照] ボタンを押下し下記ファイルを追加

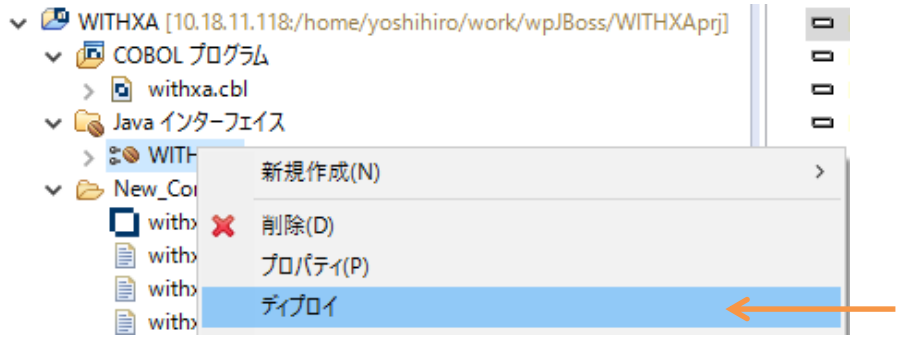
- $JBOSS_HOME/modules/system/layers/base/javax/ejb/api/main/jboss-ejb-api_3.1_spec-1.0.2.Final-redhat-3.jar
- $JBOSS_HOME/modules/system/layers/base/javax/servlet/api/main/jboss-servlet-api_3.0_spec-1.0.2.Final-redhat-2.jar
- $JBOSS_HOME/modules/system/layers/base/javax/resource/api/main/jboss-connector-api_1.6_spec-1.0.1.Final-redhat-3.jar

⑬ [OK] ボタンを押下し、設定画面を閉じる

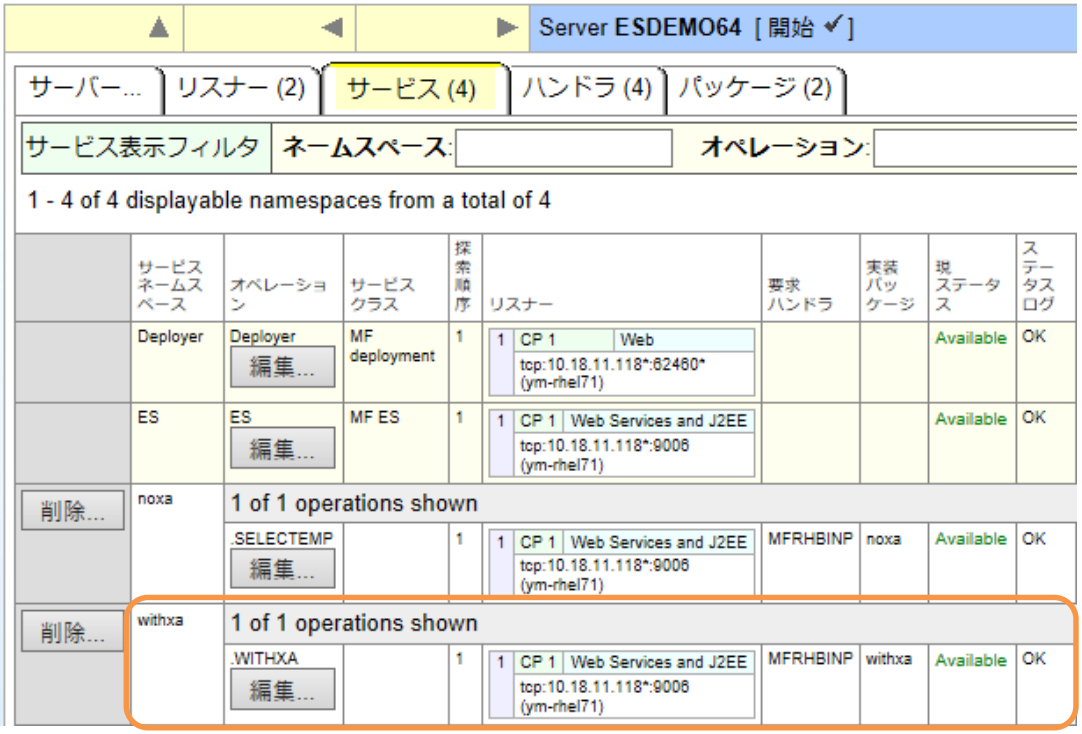


17) 作成した Java サービスを Enterprise Server へディプロイ

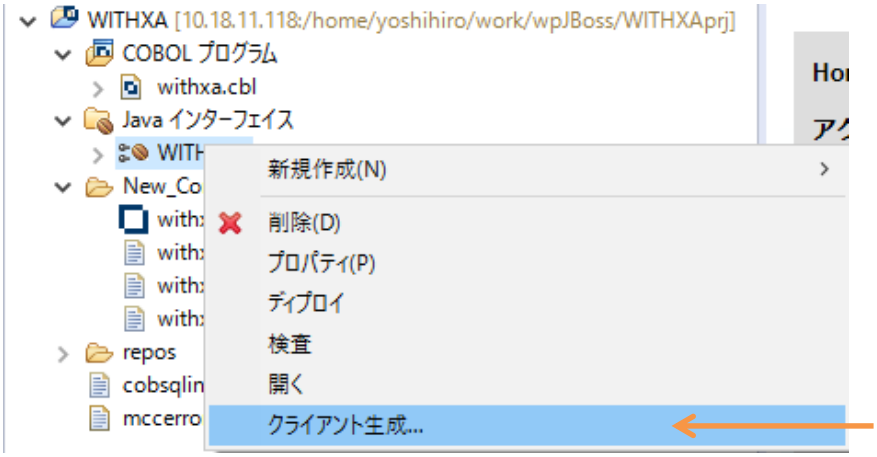
COBOL エクスプローラにて用意した Java インターフェイスを右クリックし、[ディプロイ] を選択



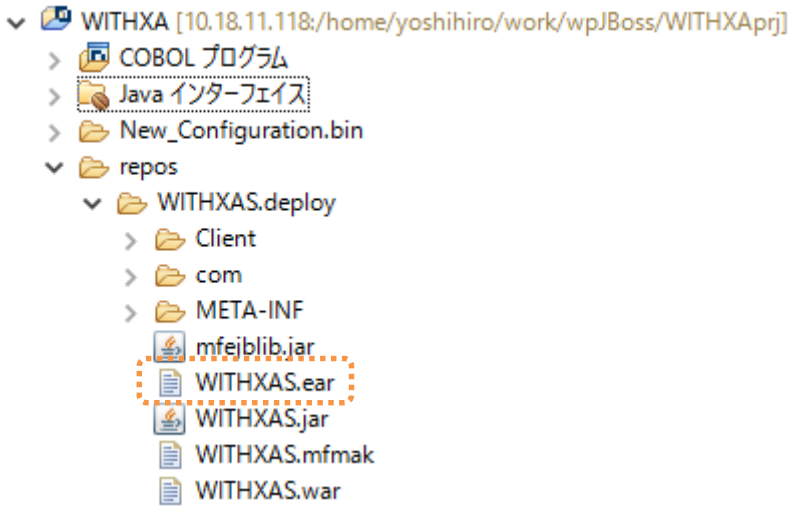
正常にディプロイできたことを Enterprise Server Administration 画面にて確認することができます（[Home] > [編集] ボタン > [サービス] タブ）：



18) ディプロイしたサービスをテスト呼び出しする JavaEE 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

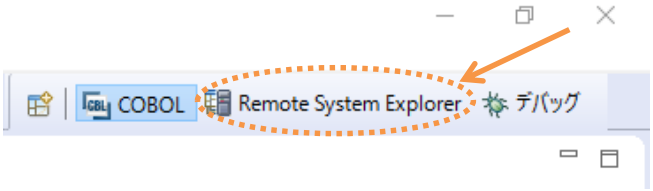


COBOL エクスプローラにて .ear にアーカイブされたアプリケーションが生成されていることを確認できます：

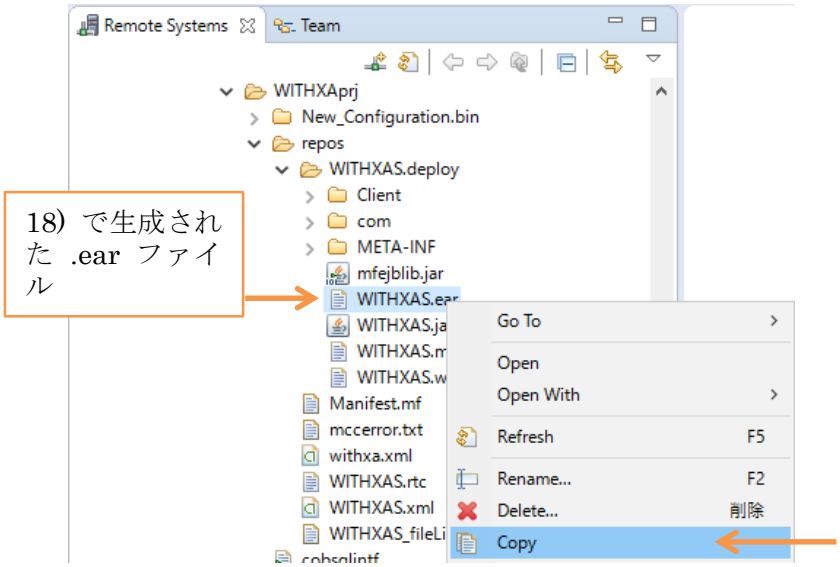


19) 生成された Java EE アプリケーションを JBoss EAP 6.4 へディプロイ

① [Remote System Explorer] パースペクティブへ切り替え



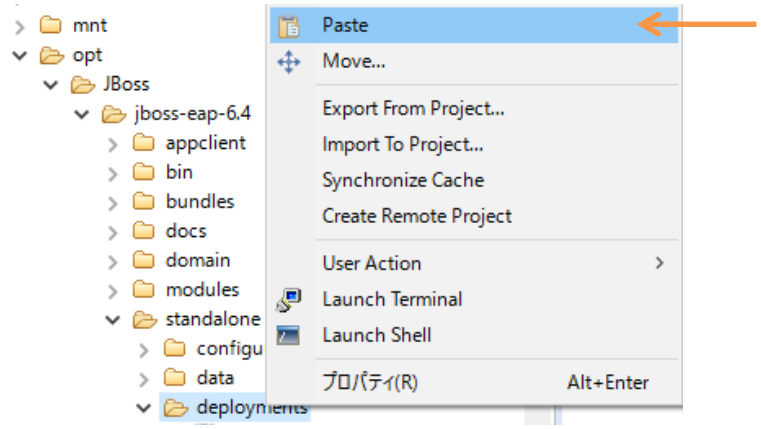
② [Remote System] ビューにて
＜Linux サーバ＞ → Sftp Files
配下にある生成された .ear を右クリックし、[Copy] を選択



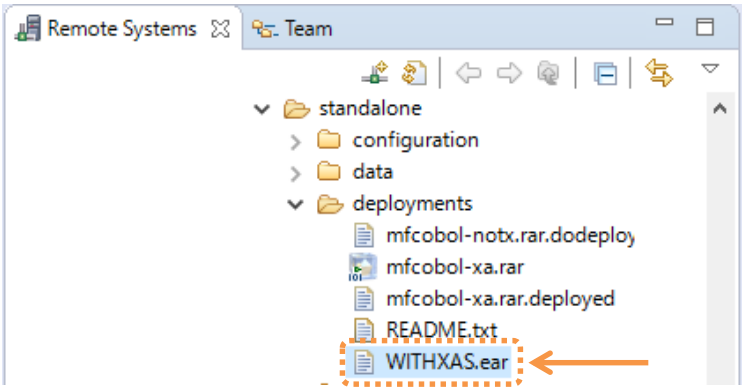
③ 同ビューにて

$JBOSS_HOME/standalone/deplyoments

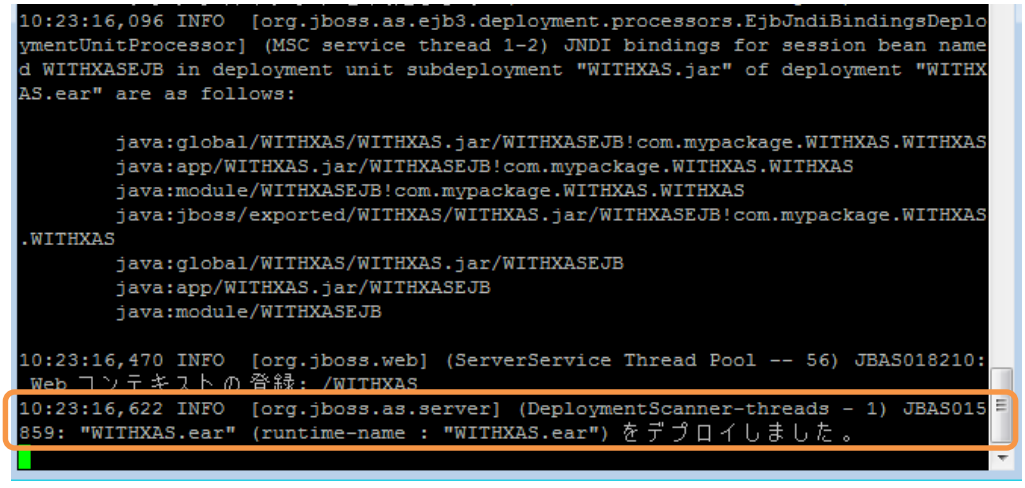
ディレクトリを右クリックし、[Paste] を選択



$JBOSS_HOME/standalone/deplyoments ディレクトリに .ear が追加されます：



正常にディプロイされたことを JBoss を起動した Linux のターミナルからも確認ができます：



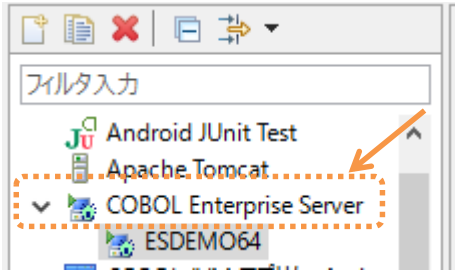
20) Enterprise Server デバッグを起動

① [COBOL] パースペクティブに戻る

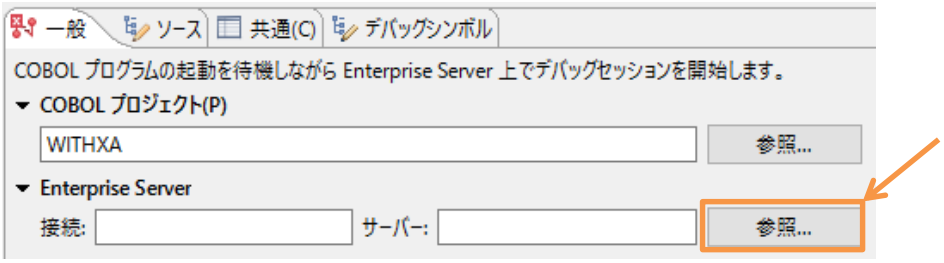
② COBOL エクスプローラにてプロジェクトを右クリックから
[デバッグ] > [デバッグの構成]
を選択

③ COBOL Enterprise Server をダブルクリック

④ 付録 1 で作成したデバッグ構成をダブルクリック

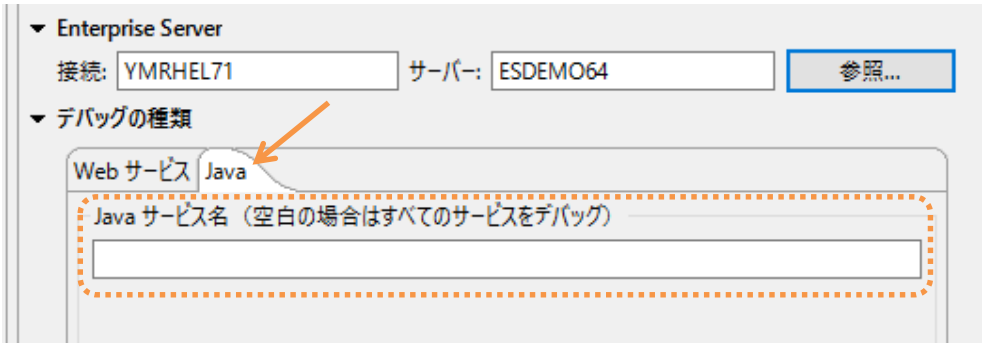


⑤ [Enterprise Server] 欄にて [参照] ボタンを押下



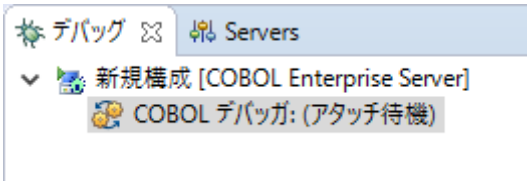
⑥ ESDEMO64 を選択

⑦ [Java] タブをクリックし、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



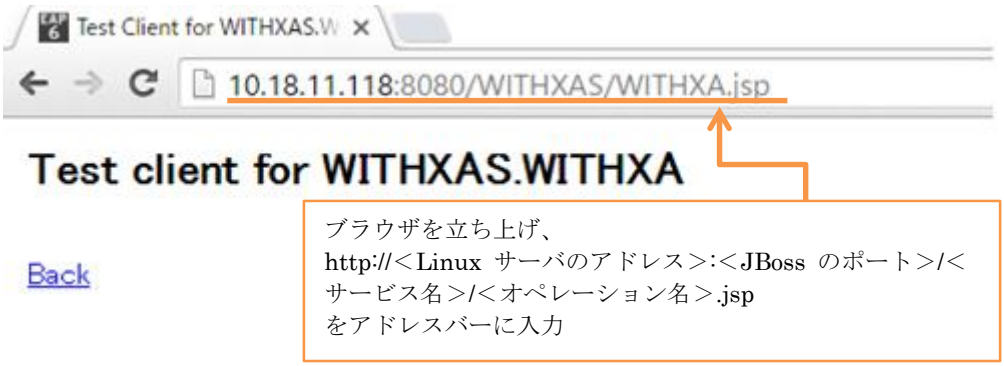
⑧ [デバッグ] ボタンを押下

⑨ [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：

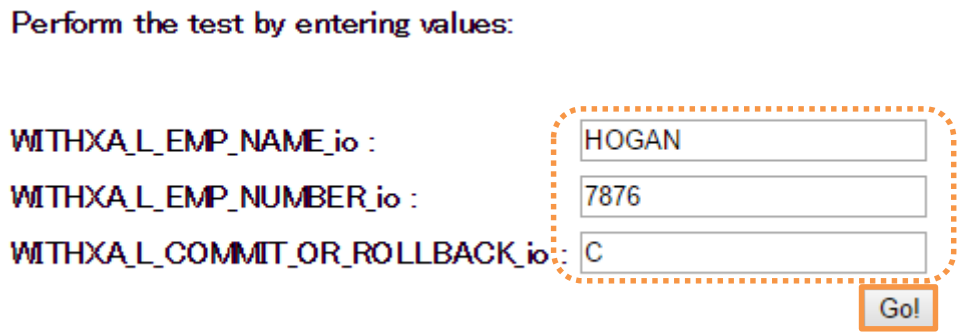


21) ディプロイしたアプリケーションをデバッグ実行（実行時エラーなく終了するパターン）

① アプリケーションを起動

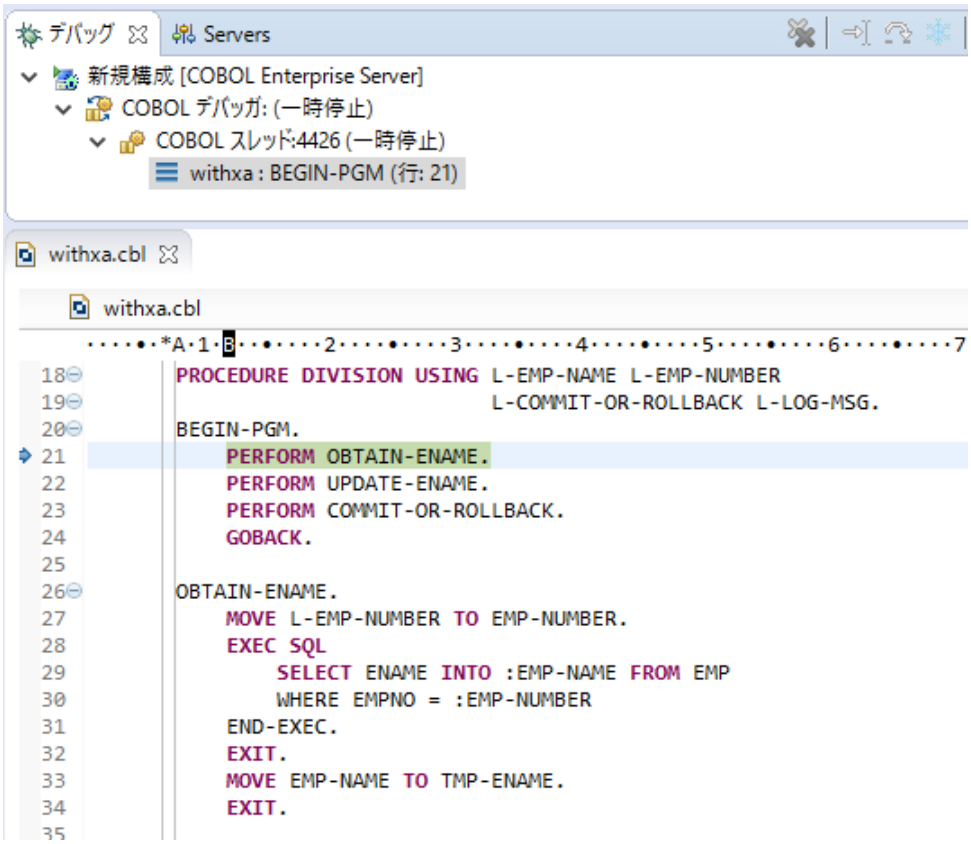


② アプリケーションを実行
[WITHXA_L_EMP_NAME_io] 欄に [HOGAN] を
[WITHXA_L_EMP_NUMBER_io] 欄に [7876] を
[WITHXA_L_COMMIT_OR_ROLLBACK_io] 欄に [C] を
入力し [Go!] ボタンを押下



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：

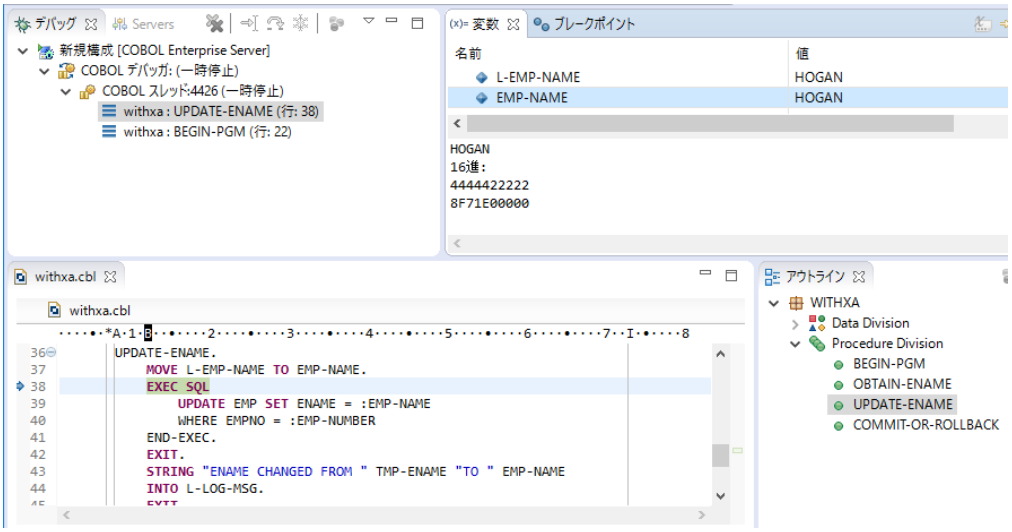
Enterprise Server にディプロイした COBOL プログラムの最初の行を実行する前で処理が一時停止していることが確認できます：



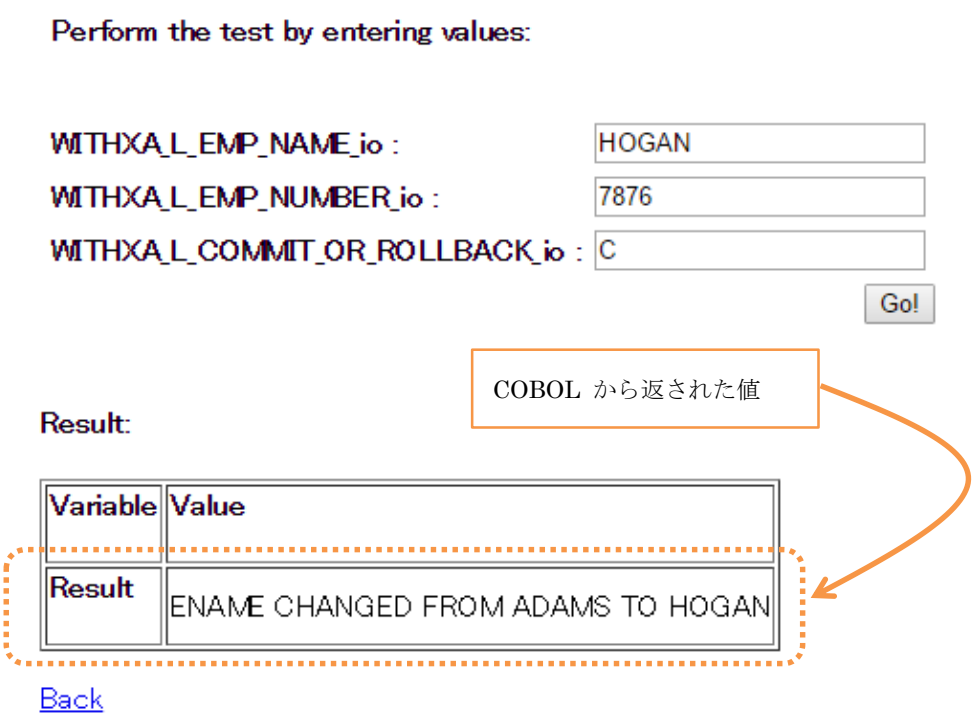
付録 1 の要領でデバッグ実行を進めます。

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、正常に SQL 文を実行しています。

デバッグ実行イメージ：



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 Web のリスポンスとして戻します：



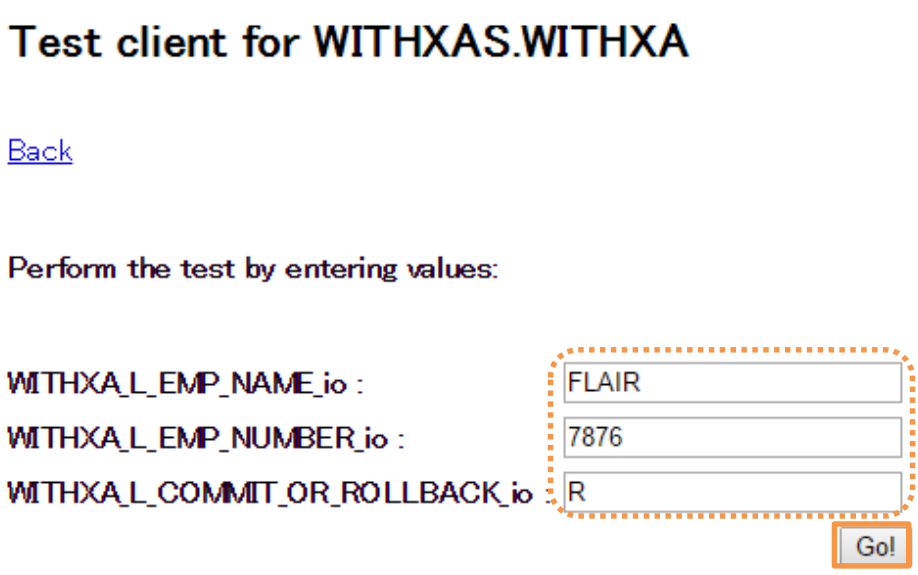
22) アプリケーションが処理したレコードを SQL*Plus で確認

23) ディプロイしたアプリケーションをデバッグ実行（実行時エラーで終了するパターン）

① アプリケーションを起動

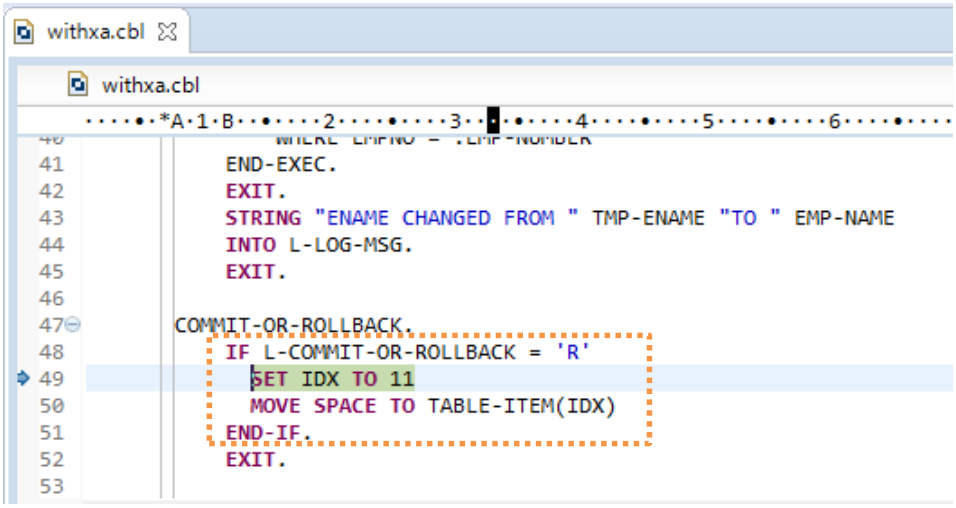
② アプリケーションを実行

[WITHXA_L_EMP_NAME_io] 欄に [FLAIR] を
[WITHXA_L_EMP_NUMBER_io] 欄に [7876] を
[WITHXA_L_COMMIT_OR_ROLLBACK_io] 欄に [R] を
入力し [Go!] ボタンを押下

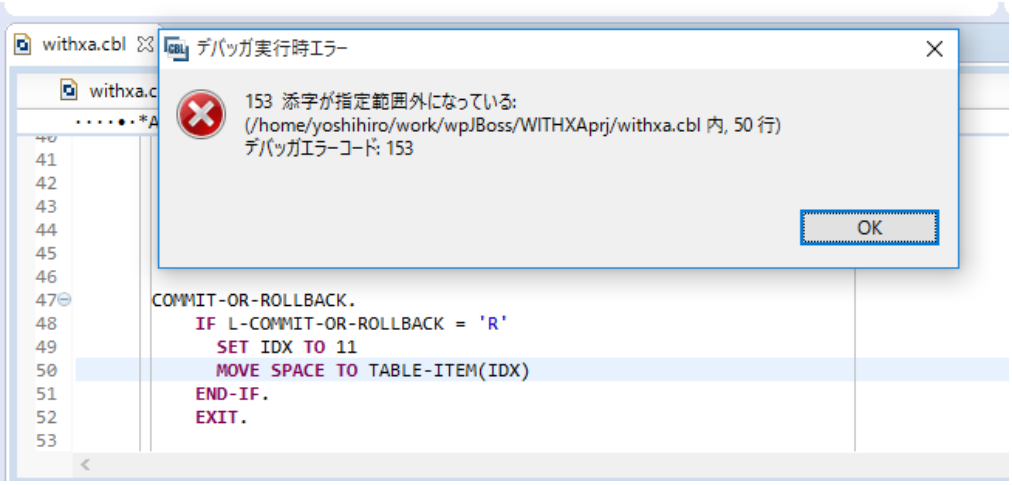


これまでの要領でデバッグ実行を進めます。

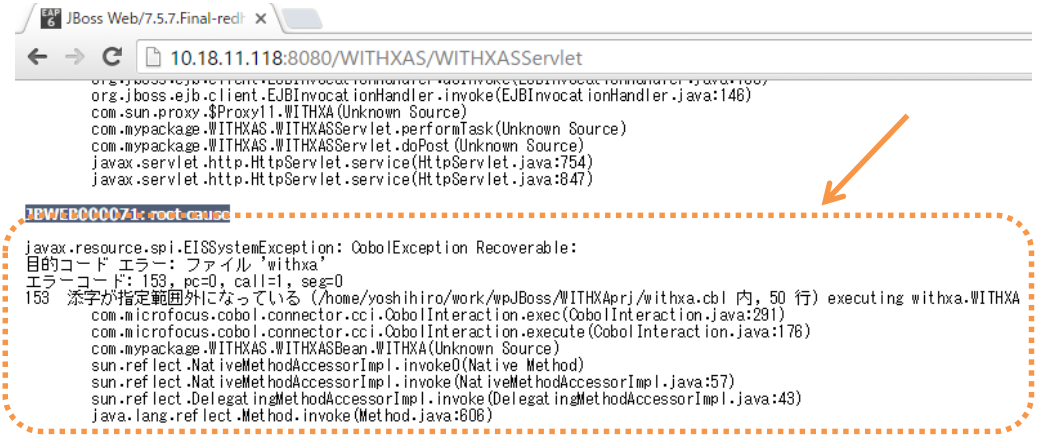
今回は L-COMMIT-OR-ROLLBACK に「R」を格納したため、UPDATE 文実行後の下図の IF 文は真と評価されます：



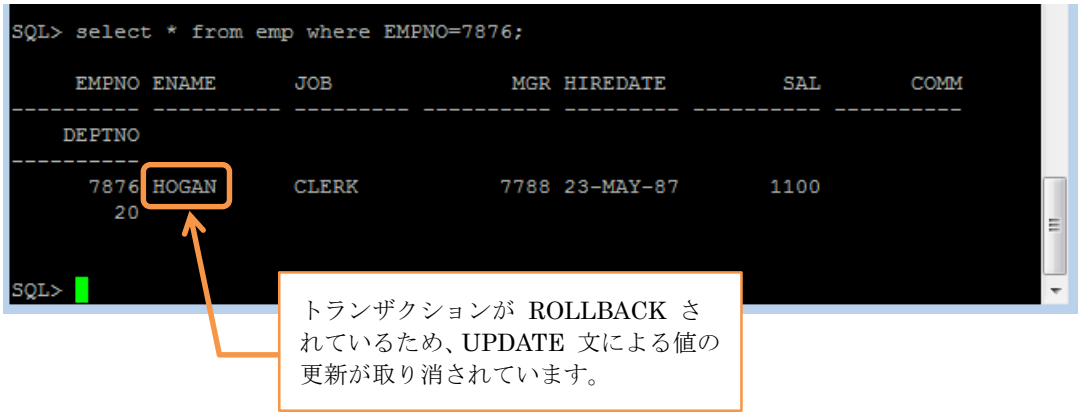
IF 文中の MOVE 文を実行すると添え字範囲外の実行時エラーが発生します：



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：



24) アプリケーションが処理したレコードを SQL*Plus で確認



25) JBoss EAP、Enterprise Server、デバッグ、リモート開発デーモンを停止

以上