## 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 に準拠しているので同じく 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 環境ヘインストールし、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 環境ヘインストールする 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 にディプロイする 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。

#### > <u>アプリケーションサーバー側 ソフトウェア</u>

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 ヘディプロイ
- 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 ヘディプロイ
- 12. テスト用クライアントアプリケーションを JBOSS EAP 6.4 にディプロイ
- 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 限があるパスヘコピーします。

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 startrdodaemon コマンドを実行します。

コマンド例)

\$COBDIR/remotedev/startrdodaemon

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

- 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
- 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 ヘディプロイ

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 \$JBOSS\_HOME/standalone/deployments ヘコピーします。

例:cp \$COBDIR/javaee/javaee6/jbossEAP64/mfcobol-xa.rar \$JBOSS\_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

ディプロイ > 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ィプロイおよび構成 > EJB とリソース アダプターのディプロイ > EJB のディプロイ - 概要 > JBoss へのディプロイ

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 スアダプターもディプロイされています



#### 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
  - [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。

「「「新規プロジェクト	_		×
ウイザードを選択			<
Micro Focus リモート COBOL フロシェクトを1FR以しまう			
ウィザード( <u>W</u> ):			
フィルタ入力			
> > > JAXB > > JPA			^
<ul> <li>         ・ Maven         ・         ・         ・</li></ul>			
COBOL コピーファイル プロジェクト のののとプロジェクト			
御 リモート COBOL コピーファイル・プロジェクト     御 リモート COBOL プロジェクト			
> > Micro Focus Unit lest			
<ul> <li>&gt; 🧀 Web</li> <li>&gt; プラグイン開発</li> </ul>			~
(P)         (P) </td <td>)</td> <td>キャンセ</td> <td>JL</td>	)	キャンセ	JL

- 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。

[10] リモート COBOL プロジェクトの新規作成	_		×
<b>リモート coBol プロジェクト</b> ワークスペースまたは外部にリモート COBOL プロジェクトを作成			
ブロジェクト名: NOXA リモート設定			
接続名:	~接続	の新規作	成
JE-Fa	~	▲ Brow	se
リモートの場所はリモートマシンのプロジェクトパスに設定しなければいけません。			



- 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。

🔤 New Connection — 🗆 🗙				
Remote Micro Focus D Define connection inform	<b>evHub (RSE 経由) System Connectic</b> nation			
Parent profile:	tok-vmwin10cu			~
Host name:	10.18.11.121			~
Connection name :	10.18.11.121			
Description :				
✓ Verify host name <u>Configure proxy settings</u>				

- 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

Browse For Folder	×
Select a folder	
My Home	
> <sup>t</sup> My Home > <sup>t</sup> Root	

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。

call リモート COBOL プロジェクトの新規作成	-	_		×
リモート cobol プロジェクト			1	-
ワークスペースまたは外部にリモート COBOL プロジェクトを作成				\$
プロジェクト名: NOXA				
リモート設定				
接続名: 10.18.11.120	$\sim$	接続の	新規作	成
リモートク /home/yamasiro/work/NOXA		~	Brow	se
リモートの場所はリモートマシンのプロジェクト パスに設定しなければいけません。				

- 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。
- ② 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。
- [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。

アルタ入力       COBOL       () マ () マ         > リソース       Micro Focus       () レドボス         ビルドパス       () レド環域       () レド環域         > COBOL       () レド環域         > DUF環境       () レド環域         > リング       () レビット         シーグット設定       () レビット         アンドリボイント       () 32 ビット         () Validation       () 64 ビット	📶 プロパティ: NOXA		— 🗆 X	
<ul> <li>&gt; リソース</li> <li>Micro Focus じ以下 (元 ・ じ以下 場成</li> <li>&gt; (COBOL イベント じル 「環境         ・ 」 リンク         <ul> <li>▲ カガス: New Configuration.bin</li> <li>&gt; メロンド リルマック</li> <li>※ 「丁・ジェクトリアボックト」</li> <li>※ 実行時構成</li> <li>Project Facets</li> <li>&gt; Task Repository Task Rags</li> <li>&gt; Validation</li> </ul> </li> </ul>	ንብሥንእታ	COBOL	<	
WikiText ジレダ-	<ul> <li>&gt; リソ−ス</li> <li>Micro Focus ビルドパス</li> <li>ビルド爆成</li> <li>&gt; COBOL イベント</li> <li>ビルド環境</li> <li>&gt; リンク</li> <li>&gt; プロジェクト設定</li> <li>&gt; 実行時構成</li> <li>Project Facets</li> <li>&gt; Task Repository Task Tags</li> <li>&gt; Validation</li> <li>WikiText</li> <li>ビルダー</li> <li>プロジェクト参照</li> </ul>	New Configuration [使用中]         出力パス:       New Configuration.bin         エントリポイント:       ターヴット設定         ダーヴット設定       ブラットフォームデーゲーパー         ダーヴット設定       ブラットフォームデーゲーパー         ダーヴット設定       ブラットフォームデーゲーパー         ダーヴット設定       ブラットフォームデーゲーパー         マーヴット設定       〇32 ビット         ・ プロジェクトの COBOL の設定の上書き       〇32 ビット	✓ 構成の管理	

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。

ターゲット設定 ターゲットの種類 すべて ネイティブライブラリ ファイル 〜	ブラットフォームターゲット ○ 32 ビット   ● 64 ビット
<ul> <li>プロジェクトの COBOL の設定の上書き</li> <li>☑ 構成の固有な設定を可能にする(C)</li> </ul>	

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。

警告レベル:	回復可能なエ	5-を含める(レベル E)	~	^
最大エラー数:	100			
追加指令: P(cobsql) COBSQLTYPE=(	DRACLE8 END-C ENDP			^ \

指定值:「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 のコモン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。

値 インポート	_	
<b>ファイル・システム</b> ローカル・ファイル・システムからリソースをインポートします。		
次のディレクトリーから(Y): C:¥work	~	参照(R)
>   work		

 ⑤ Windows から Linux ヘ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。

א-עעב 🖳	🖹 問題 🧔 タスク 🔲 プロパティー 🏗 Table Results 🥡 Filter Defin 🧏 コードカバレ 📴 Micro Focu 🖓 E
	🗙 🔆 📄 🛃 🔛 🚽 🚍 🗸 📑
Micro Focus ビルド	S WITHXA
pre.build.c	fg.New_Configuration:
cobol.compi	le.cfg.New_Configuration:
[cobol]	
[cobol]	Compiling (64-bit) withxa.cbl from project 'WITHXA' on connection '10.18.11.12
[cobol]	
[cobol]	* Cobsql Integrated Preprocessor
[cobol]	* CSQL-I-018: Oracle プリコンパイラトランスレータを起動しま�
[cobol]	★ CSQL−I−020: Oracle プリコンパイラの出力を処理中。
[cobol]	★ CSQL−I−001: COBSQL:チェッカへの引き渡しを完了しました。
[cobol]	Compilation complete with no errors
cobol.link.	cfg.New Configuration:
[cobollink]	Linking (64-bit) withxa.so
[cobollink]	Cobol プログラム "SQLBEX" が未定義。動的にロードされるものとみなす
[cobollink]	Cobol プログラム "SQLADR" が未定義。動的にロードされるものとみなす
·	Tick

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。

```
    ✓ I WITHXA [10.18.11.120:/root/work/withxa
    ✓ I COBOL J □ J ⊃ J_A
    > Ø withxa.cbl
    ✓ > New Configuration.bin
    ✓ withxa.idy
    ✓ withxa.o
    ✓ withxa.o
    ✓ withxa.o. 1.tlog
    ✓ withxa.so
```

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-r	rhel73-64 N	New_Configuration	on.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のでプログラム中には 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
  - 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。

📶 Java インターフェイスの新規作成ウィザード	—		×
Java インターフェイスの新規作成 このページで Java インターフェイスを新規作成します		e-(	S
Java インターフェイス名: WITHXAS マッピング: ④ デフォルト 〇 無し			
マップするプログラム: WITHXA/withxa.cbl		参	照

- 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。

KAGE SECTION:		WITHXA オペレーション - インターフェ	イスフィール	5:
前	PICTURE	名前	方向	型
L-EMP-NAME	X(10)	L_EMP_NAME_io	入出力	String
L-EMP-NUMBER	S9(4) comp-5	L_EMP_NUMBER_io	入出力	short
L-COMMIT-OR-ROLLBACK	Х	L_COMMIT_OR_ROLLBACK	入出力	String
L-LOG-MSG	X(80)	► L_LOG_MSG_io	入出力	String

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。

<b>ER</b> 71-	-ルド プロパティ			×
名前:	L_EMP_NAME_io			
型:	String	~	OCCURS:	0
方向: - マッピ	●入力 ●出力 ○入出力 ング			
L-EN	IP-NAME		編集	
	ОК		キャンセル	,



- 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)] > [Directry 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。

📖 新規 Directory Server 接	結			×
接続の新規作成 既存の Micro Focus Directo	ny Server への接続を新規作成します		ļ	-
名前: サーバアドレス (IPv4/ホスト名): サーバーボート:	tok-rhel73-64 10.18.11.120 86	]		
?	終了	( <u>F</u> )	キャンセ	λ

#### 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。

編集	MFES (MSS)	MSSDEMO	停止 開始	1 3 リスナー 詳細	- / 10
追加					

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: ESDEMO64 動作モード: ③ 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。

サーノ	(一追加	I (Page 2 of 3):
サーノ	(一名:	ESDEMO64
サーノ	(ータイ	プ:
۲	MFES	Micro Focus Enterprise Server An enterprise server that provides an execution environment for COBOL
0	MFES (MSS)	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。



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(	プロ	コセス 1	☑自	動調	起動					
	リス	+-	ブロセスID	1	コントロールチャネルアドレス					
編集	2	追加	] .	top	top:*:*					
	and the second diversity of	June 1	10	÷.,						
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					ステータスログ					
	編	集	Veb Services a	and	tcp:*:*	停止	12/17/15-10:52:04	ок		
	編	集 \	Veb		tcp:*:*	停止	12/17/15-10:52:04	ок		

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 TCP 🗸	エンドポイントプロトコル	TCP 🗸
エンドポイントアドレス **	エンドポイントアドレス	*:9006

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。

▲    ► Server ESDEMO64 [停止]
<b>サーバー…</b> リスナー (2) サービス (3) ハンドラ (4) パッケージ (1)
<mark>プロパティ</mark> 構成 診断 過去の統計
──般 XAリソース (0) MSS MQ スクリプト アクセス権 セキュリティ
名前: ESDEMO64 ×



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。
	XA リソース名として任意の名前を指定します。
名前	Oracleの場合は Oracle_XA 固定です。
	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。
モジュール	【Oracle 使用時の例】
	動的登録 /home/tarot/xa/ESORAXA64_D.so を入力します。
	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】
OPEN	Oracle_XA+SesTm=100+SqlNet=tok-par+Acc=P/scott/tiger を入力します。 OPEN文字列:
又子列	Oracle_XA+SesTm=100+SqlNet=tok-par+Acc=P/scott/tiger
	※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。



▲ Server ESDEMO64 [開始 ✔]					
<mark>サーバー</mark> リスナー (2) サービス (3) ハンドラ (4) パッケージ (1)					
<mark>プロパティ</mark> 構成 診断 過去の統計					
-般 XAリソース(0) MSS MQ スクリプト アクセス権 セキュリティ					
名前: ESDEMO64 開始オプション:					
共有メモリページ数: 512 サービス実行プロセス: 2					
共有メモリクッション: 32 要求ライセンス: 10					
ローカルコンソールを表示: □ 動的デバッグを許可: ☑ Start on System Start: □ 64-Bit Working Mode: ☑					
以前のログを削除: 🗌 コンソールログサイズ (K): 🛛					

(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
  - 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[スタータス]が「開始」に変わります。

編集 MFES ESDEMO84 開始 1 tcp:10.18.11.118*:254 (ym-rhel71) ✓ 2 リスナー 詳細
---

- 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
  - 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 <mark>ESコンソール</mark> CSコンソール
画面	● Show last 10 lines of 35 total entries
Entry	Event Show Entire Log
26	151222 16181773 11074 ESDEMO64 CASTS0002I ES TSC Service Process initialization complete 18:18:17
27	151222 16181773 11081 ESDEMO64 CASCS5100I Communications Process instance 01 is ready to accept requests 16:18:17
28	151222 16181869 CASCD1071I Administration SEP created for Server ESDEM064, 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 ESDEMO84 CASXO0020I ORCL XA interface loaded. Name(Oracle_XA), Registration Mode(Dynamic) 18:18:19
33	151222 16181986 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

#### 3.11 コンパイルした COBOL アプリケーションを Enterprise Server ヘディプロイ

- 1) 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「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。

<ul> <li>✓ ● WITHXA [10.18.11.1</li> <li>✓ ● COBOL プログラム</li> <li>&gt; ● withxa.cbl</li> <li>✓ ● Java インターフェイ</li> </ul>	18:/home/yoshihiro/v	vork/wpJBoss/WITHXAprj]	
New Conf	新規作成(N)		>
V Wew_com			
withxa 💥	削除(D)		
iii withxa	プロパティ(P)		
📄 withxa	70/()·1(P)		
📄 withxa	ディプロイ		

- 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。

Call Enterprise Server を選択						
ディプロイ先の En	terprise Serve	er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ィプロイする」を選択します。
- 「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。

# 

💷 マッピング プロパティ					×
一般 ディプロイメントサーバー	アプリケーションファイル	EJB 生成			
レガシーアプリケーションをディプロ	イ済みか、またはサーバー	にディプロイする必要があるかを指定して	ください。		
○レガシーアプリケーションは既にディプロイ済み ディプロイされたアプリケーションのパス:					
◉ レガシーアプリケーションをディプロイする					
アプリケーションファイル:					
/root/work/withxa/New_Configuration.bin/withxa.idy /root/work/withxa/New_Configuration.bin/withxa.so ファイル削除					
				AN LOUISING	- 1

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。

ſ	ער אינ	ング プロパティ			
	一般	ディプロイメントサ	ーパー アプ	リケーションファイル	EJB 生成
	アプリケ	ーション サーバー	JEE 6 $\lor$	JBoss EAP 6.4	$\sim$

- 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。
- 同様に下記の J2EE クラスパスを設定します。
  - > \$JBOSS\_HOME/modules/system/layers/base/javax/ejb/api/main/jboss-ejbapi\_3.1\_spec-1.0.2.Final-redhat-3.jar
  - > \$JBOSS\_HOME/modules/system/layers/base/javax/servlet/api/main/jboss-servletapi\_3.0\_spec-1.0.2.Final-redhat-2.jar
  - \$JBOSS\_HOME/modules/system/layers/base/javax/resource/api/main/jbossconnector-api\_1.6\_spec-1.0.1.Final-redhat-3.jar
- 2) 作成した Java サービスを Enterprise Server ヘディプロイする
  - 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 [ディプロイ] 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 して正常にディプロイできたことを確認することができます。

# Focus

					Server ESDEMO64	[開始 🖌]			
サーバー	גען	ナー (2)	サービス	(4)	ハンドラ (4) パッケ・	-ジ(2)			
サービス語	表示フィノ	レタ ネー1	ムペース	:	אל <mark>ו</mark> אל	ノーション	<i>r</i>		
1 - 4 of 4	displayab	le namespa	ices from a	a to	tal 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 top:10.18.11.118^:82480* (ym-rhel71)			Available	ок
	ES	ES 編集	MF ES	1	CP 1         Web Services and J2EE           tcp:10.18.11.118*:9006         (ym-rhel71)			Available	ок
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	ок
削除	withxa	l 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	ок

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

- 1) ディプロイした 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。

VITHXA [10.18	11.118:/home/yoshihiro/work/wpJBoss/WITHXAprj]	_
🗸 🖉 COBOL 🖓	グラム	
> 🖻 withxa.d	:bl	HO
🗸 🕞 Java インター	דילא	P!
> 😂 WITH	新担作成(NI)	>
✓	7/1/3/CTF/2/(TN)	
🔲 with: 🔉	<b>(</b> 削除(D)	
📄 with:	プロパティ(P)	
📄 with:	≓, 10 /	
📄 with:	71701	
> 🗁 repos	検査	
📄 cobsqlin	開く	
📄 mccerro	クライアント生成	

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

#### JCA による JBOSS EAP 連携の設定と開発



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

例:

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

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
  - 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。
  - [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。

名前(N): 新規構成	
💦 一般 🐚 ソース 🔲 共通(C) 🦭 デバッグシンボル	
COBOL プログラムの起動を待機しながら Enterprise Server 上でデバッグセッションを限 <ul> <li>▼ COBOL プロジェクト(P)</li> </ul>	見始します。
NOXA	参照
✓ Enterprise Server	
接続: サーバー:	参照

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。

🏘 デバッグ 😂	워 Servers
✓ ★ 新規構成 ★ 新規構成 ★ COB	t [COBOL Enterprise Server] OL デバッガ: (アタッチ待機)

#### 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) ディプロイした 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

Perform the test by entering values:

/ 🚼 Test Client for WITHXAS.W 🗙 🔪

← → C 🗋 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。

WITHXA_L_EMP_NAME_io:	HOGAN
WITHXA_L_EMP_NUMBER_io:	7876
WITHXA_L_COMMIT_OR_ROLLBACK_io :	С
	Go!

- 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

# 

-	
奪 デバッグ 2	3 👯 Servers 🙀 🛋 🕾 🎄
<ul> <li>✓ </li> <li>※ 新規</li> <li>✓ </li> <li>※ </li> <li>✓ </li> <li>✓ </li> </ul>	構成 [COBOL Enterprise Server] OBOL デバッガ: (一時停止) @ COBOL スレッド:4426 (一時停止)
ŤW	withxa : BEGIN-PGM (行: 21)
withxa.cb	
- -	
🖻 wit	hxa.cbl
	• • * A · 1 · 📴 · • • · · · · 2 · · · • • · · · 3 · · · • • · · · 4 · · · • • · · · 5 · · · • • · · · 6 · · · • • · · · 7
180	PROCEDURE DIVISION USING L-EMP-NAME L-EMP-NUMBER
190	L-COMMIT-OR-ROLLBACK L-LOG-MSG.
200	DEDIN-PUH.
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	EXII.
34	MOVE EMPENAME TO IMPECNAME.
35	

- 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。

<ul> <li>なデバッグ 23</li></ul>	<ul> <li>(x)-変数 23 ● ブレークポイント</li> <li>名前</li> <li>● EMP-NAME-ARR</li> <li>● R-EMP-NAME</li> <li>● SALARY</li> <li>● R-SALARY</li> </ul>		値 JAMES JAMES +00950.00
noxa.cbl 🕴 😂 NOXAS	JAMES 16進: 4444522222 AID5300000		E 701-517 X
noxa.cbl      set-retval.      Move eng-name-arr to r-emp-name.      Move stalary to r-salary.     If comm-IND = -1     Move stalary to r-salary.     If comm-IND = -1     Move stalary to r-commission     Else     Move commission to p-commission	567.·I8	* I I V	● R· ● R· ● R· ● Procedure Di ● BEGIN-P( ● OBTAIN- ● LOGON ● SET-RET

- 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:	HOGAN
WITHXA_L_EMP_NUMBER_io:	7876
WITHXA_L_COMMIT_OR_ROLLBACK_io:	С
	Go!

#### Result:

Variable	Value
Result	ENAME CHANGED FROM ADAMS TO HOGAN

<u>Back</u>

 ⑦ 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>	SQL> select * from emp where EMPNO=7876;								
	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM		
D	)EPTNO								
	7876	HOGAN	CLERK	7788	23-MAY-87	1100			
	20							E	
SQL>								~	

- 2) ディプロイした 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

<u>Back</u>

Perform the test by entering values:

WITHXA_L_EMP_NAME_io:	FLAIR
WITHXA_L_EMP_NUMBER_io:	7876
WITHXA_L_COMMIT_OR_ROLLBACK_io :	R
	Go!



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。



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。

<b>m</b> デバッ	ガ実行時エラー	×
8	153 添字が指定範囲外になっている: (/root/WITHXA/withxa.cbl 内, 50 行) デバッガエラーコード: 153	
		OK

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。

J 😵 JBoss Web/7.5.7.Final-red: 🗙
← → C 🗋 10.18.11.118:8080/WITHXAS/WITHXASServlet
org-jboss-cjp-chrent-cbBinvocatronmanarch-dornace(cbBinvocatronmanarch-javarroo) org-jboss-cjp-chrent-cbBinvocatronmanarch-dornace(cbBinvocatronmanarch-javarroo) com_sun_prox/sProxyl1.WITHXA(Unknown Source) com_mypackage-WITHXAS-WITHXASServlet.performTask(Unknown Source) com_mypackage-WITHXAS.WITHXASServlet.obPost(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.exec(CobolInteraction.java:291) com.mypackage.WITHXAS.WITHXASBean.WITHXA(Unknown Source) sun.reflect.NativeMethodAccessorImpl.invoke(Native Method) sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57) sun.reflect.MativeMethodAccessorImpl.invoke(VelegatingMethodAccessorImpl.java:43) java.lang.reflect.Method.invoke(Method.java:606)

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> sel	SQL> select * from emp where EMPNO=7876;								
EMF	NO ENAME	JOB	MGR	HIREDATE	SAL	COMM			
DEPI	'NO								
78	76 HOGAN	CLERK	7788	23-MAY-87	1100				
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。