# Micro Focus Visual COBOL 2.2J Sybase ASE 16.0 / WebLogic Server 12c 動作検証 検証結果報告書

# 2014年9月29日

マイクロフォーカス株式会社

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

### 1. 検証概要、目的及びテスト方法

#### 1.1 検証概要

Micro Focus Visual COBOL には COBOL 専用の Application Server として機能する Enterprise Server が付属します。この Enterprise Server が提供する Java EE Connector 機能は、JCA 仕様準拠のコンテナとして多くの Java EE 準拠アプリケーシ ョンサーバや、XA 仕様に準拠したデータベースリソースマネージャとの動作検証が行わ れています。

本報告書は、SAP Sybase ASE 16.0 のリソースマネージャを使用し、Oracle Weblogic Server 12c との Java EE Connector の接続性を検証し、報告するものです。

#### 1.2 目的及びテスト方法

Micro Focus Visual COBOL の Enterprise Server が提供する Java EE Connector は、 現在 Oracle Database、DB2、Microsoft SQL Server、WebSphere MQ のリソースマネ ージャとの連携が動作保証されています。しかし Enterprise Server は、JCA 仕様準拠の コンテナとして、設計上は XA 仕様に準拠したすべてのリソースマネージャとの連携が可 能です。

SAP Sybase ASE 16.0 は、XA 仕様に準拠したリソースマネージャをサポートしており、 TX Series、TUXEDO、Encina といった標準的なトランザクションマネージャとの連携 が動作保証されています。従って、理論的には Enterprise Server のトランザクションマ ネージャとも連携するはずです。今回、以下のテストプログラムを実行することによって、 上記のことを実際に検証しました。検証は全て SJIS(ja\_JP.PCK) ロケール配下で実施し ています。

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

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

### 2. 検証環境

▶ Application 開発サーバ

ソフトウェア

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

#### ハードウェア

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

#### ▶ Application 開発クライアント

ソフトウェア

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

#### ハードウェア

機種:	Dell OPTIPLEX7010
CPU :	Intel Core i7-3770 3.40GHz
Memory :	3.00 Gbyte memory(ゲスト OS に割り当てたサイズ)

# 3. テスト内容

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

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

- 使用した COBOL ロジック Sybase の pub2 データベースのテーブルから指定されたキーのレコードを SELECT しその内容を返す簡単な COBOL サブルーチンです。インターフェースマ ッピングはそれぞれの特性に合わせて編集した上で Enterprise Server ヘディプロイ しました。使用したテーブルについては備考を参照してください。
- (2) 使用したリソースアダプタmfcobol-notx.rar(トランザクションなし)
- (3) 使用した Enterprise Server
   32 bit の検証では Visual COBOL にビルドインされた ESDEMO を使用しました。
   64 bit の検証では 64 bit 用の Enterprise Server を検証用に追加し、それを利用しました。
- (4) 使用した Java EE クライアント
   Visual COBOL の Interface Mapping Toolkit がディプロイ時に自動生成する EJB
   と、自動生成される Web モジュールクライアントを使用しました。
- 3.2 Sybase 更新プログラムのディプロイと、EJB 経由の JCA 呼び出しにおける コンテナ管理トランザクション
- (1)使用した COBOL ロジック Sybase の pub2 データベースのテーブルから指定されたキーのレコードを、指定された値で UPDATE する簡単な COBOL サブルーチンです。入力されたパラメータに応じて意図的にアプリケーション例外を発生させるロジックを別途埋め込んでいます。このサブルーチンにインターフェースマッピングはデフォルトの状態のままで Enterprise Server ヘディプロイしています。
- (2) 使用したリソースアダプタmfcobol-xa.rar (XA トランザクションのサポート)

(3) 使用した Enterprise Server

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

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

(4) 使用した Java EE クライアント

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

#### 4. 結果

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

	試験パターン		結果
プログラムの特性	トランザクション管理	32bit /64 bit	
Sybase よりデータ照会	アプリケーション管理	32 bit	0
Sybase よりデータ照会	アプリケーション管理	64 bit	0
Sybase のデータを更新	コンテナ管理	32 bit	0
Sybase のデータを更新	コンテナ管理	64 bit	0

## 5. テスト結果及び考察

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

以上

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

- Solaris サーバ
- 一般ユーザでログインし、Visual COBOL 及び Sybase 32 bit SDK の環境設定スク リプトを実行
- 2) Sybase のライブラリをリンクした共有ライブラリを作成
  - ① Visual COBOL の動作モードを 32 bit に指定

\$ COBMODE=32;export COBMODE
\$ cobmode
Effective Default Working Mode: 32 bit <----\$</pre>

② ダミーの COBOL プログラムを用意

③ Sybase のライブラリをリンクした共有ライブラリを作成1

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

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

\$ In -s \$SYBASE/\$SYBASE\_OCS/bin/cobpre\_r ./cobpre
\$

<sup>&</sup>lt;sup>1</sup> \$SYBASE/\$SYBASE\_OCS/sample/esqlcob に用意されているサンプルを

SYBPLATFORM=nthread\_sun\_svr4 でビルドする場合にピックアップされるライブラ リを指定しています。

<sup>&</sup>lt;sup>2</sup> cobpre\_r は SYBPLATFORM=nthread\_sun\_svr4 でサンプルをビルドする際に利用 されるプリコンパイラです。

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

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

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

# mfds & [1] 18236 #

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

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

② COBOL リモート開発用のデーモンを起動

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

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

- Windows クライアントマシン
- Windows 上で Solaris サーバで稼働する Enterprise Server を操作するための環境 を設定
  - <Visual COBOL のインストールフォルダ>¥bin¥mf-client.dat をテキス トエディタで開く
  - ② [directories] 欄に
     mrpi://<Solaris サーバの IP アドレス>:0
     の形式で Solaris サーバの Directory Server エントリを追加
- 10) Visual COBOL for Eclipse を起動
- 11) COBOL リモートプロジェクトを作成
  - ① [ファイル] メニュー > [新規] > [COBOL リモートプロジェクト] を選択

GBL									
ファイ	ル(F) 編集(E)	ナビゲート(N)	検索(A)	プロジェク	가(P)	実行(R)	ウィンドウ(W)	へルプ(H)	
	新規(N)		Alt+Sh	ift+N ►	鬯	COBOL JV	/M プロジェクト		
	ファイルを開く(.).				2	COBOLプ	ロジェクト		
	閉じる(C)		C	trl+W	2	COBOL I	ピーファイル プロジ	ジェクト	
	すべて閉じる(L)		Ctrl+Sh	ift+W	ê	COBOL U	モート プロジェクト		←
					िं	リモート CO	BOL コピーファイ	′ル プロジェクト	
	保管(S)		C	trl+S	e 🚸	Wab #_P	7		

- ② プロジェクト名を指定し、[次へ] ボタンを押下
- ③ プロジェクトテンプレートを選択する画面では [Micro Focus テンプレート] を選択し [次へ] ボタンを押下
- ④ [接続の新規作成] ボタンを押下

GBL	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</b> ワークスペースまたは外部にリモ	ト ート COBOL プロジェクトを作成		1		\$
プロジェクト名: Sybase32 リモート設定 接続名: リモートロケーション		✓ 接続	の新規 Bro	見作成 owse	
リモート ロケーションはリモート	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 Solaris サーバの IP アドレスを入力し、[完了] ボタンを 押下

<u>au</u>	New Connection	- 🗆 🗙		
Remote Micro Focus Devi Define connection informa	Hub (RSE 経由) System Connection ation			
Parent profile :	tok-mitsuwin81D	~		
 Host name :	10.18.11.27	¥		
Connection name :	10.18.11.27			
Description :				
✓ Verify host name				
Configure proxy settings				
? < 戻	さ(B) 次へ(N) > 完了(E)	キャンセル		

- ⑦ [Browse] ボタンを押下
- ⑧ ユーザ認証に関するポップアップが出たら Solaris サーバのユーザの認証情報を入力し、[Save Password] にチェックを入れ、[OK] ボタンを押下
- ⑨ Solaris サーバ上でリモート開発のプロジェクトディレクトリとして利用するディレクトリをツリーで指定し、[OK] ボタンを押下
- ⑩ [完了] ボタンを押下

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 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: Sybase32 リモート設定 接続名: 10.18.11.27 v リモート ロケーション /export/home/yoshihirom/test/sybase/remote/sybase32 リモート ロケーションはリモート 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。	接続の新規作成 マ 🔺 Browse
(N) >         完了(E)	キャンセル

Solaris サーバ上に Eclipse の COBOL プロジェクトが生成されます: [COBOL エクスプローラに表示される COBOL リモートプロジェクト] <sup>Cobol エクスプローラ 2 <sup>Cobol</sup> サーバーエクスプローラー <sup>Cobol</sup> </sup>

[Solaris サーバ上に生成されたプロジェクトファイル]

ĺ	B tok-putter.microfocus.com - PuTTY	- • ×
	yoshihirom@tok-putter:~/test/sybase/remote/sybase32\$ Is -a	A
	cobolBuild .cobolProj .project	
	yoshihirom@tok-putter:~/test/sybase/remote/sybase32\$ pwd	
	/export/home/yoshihirom/test/sybase/remote/sybase32	
	yoshihirom@tok-putter:~/test/sybase/remote/sybase32\$	· · ·

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

- COBOL エクスプローラにて作成したプロジェクトを右クリックし、[プロパ ティー]を選択
- ② [Micro Focus] > [ビルド構成] > [COBOL] へとナビゲート
- ③ [ターゲットの種類] 欄を [すべて INT/GNT ファイル] に変更
- ④ [プラットフォームターゲット] 欄が [32 ビット] となっていることを確認

ターゲット設定	
ターゲットの種類	プラットフォーム ターゲット
すべて INT/GNT ファイル	● 32 ビット ◎ 64 ビット

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



⑥ [.GNT にコンパイル] をチェック

言語方言:	Micro Focus
ソース フォーマット:	固定
□ 指令ファイルの生成	
回 リストファイルを生成	
デバッグ用にコンパイル(D)	
🔄 出力の表示	
<ul> <li>□ リスドファイルを主成</li> <li>② デバッグ用にコンパイル(D)</li> <li>□ 出力の表示</li> <li>✓ .GNT にコンパイル</li> </ul>	

- ⑦ [OK] ボタンを押下
- 13) Sybase 上のデータを照会する埋め込み SQL 文の入った COBOL プログラムをプ ロジェクトに追加4
  - COBOL エクスプローラにてプロジェクトを右クリックし、
     [新規] > [COBOL プログラム]
     を選択
  - [新規ファイル名] 欄では [Sel.cbl] を指定
  - ③ [完了] ボタンを押下

<sup>4</sup> 本検証で利用した実際のソースは本文書とともに公開しています。

④ ソースビューに以下プログラムをコーディング5

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

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

⑤ Ctrl + S を打鍵し、ソースを保存

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

💷 コンソール 🛛 🔝 問題 🖉 タスク 🔲 プロ	パティー
COBOL Build	
cobol.cfg.New_Configuration: [cobol] [cobol] Compiling Sel.cbl [cobol] [cobol] * Cobsql Integrated Prep [cobol] * CSQL-I-018: Sybase プリ [cobol] * CSQL-I-020: Sybase プリ [cobol] * CSQL-I-001: COBSQL:チ [cobol] * CSQL-I-001: COBSQL:チ	rocessor Jコンパイラトランスレータを起動します。 Jコンパイラの出力を処理中。 ェッカへの51ぎ渡しを完了しました。 h no errors.
os.init:	
os.init.windows:	COBSQL が Sybase の
os.init.unix:	プリコンパイラをビルド
init:	時に利用しアリコンハイ
post.build.cfg.New_Configuration:	
BUILD SUCCESSFUL Build finished with no errors.	プリコンパイル後のソー
Total time: 4 seconds	スをコンパイルし、正常に
	処理できたことを示して います。

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



端末エミュレータでも実際に生成されていることを確認できます:

yoshihirom@tok-putter:~/test/sybase/remote/sybase32\$ pwd	
/export/home/yoshihirom/test/sybase/remote/sybase32	
yoshihirom@tok-putter:~/test/sybase/remote/sybase32\$ Is New_Configuration.	.bin 🗏
Sel.gnt Sel.gnt.1.tlog Sel.idy	
yoshihirom@tok-putter:~/test/sybase/remote/sybase32\$	-

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

- COBOL エクスプローラにてプロジェクトを右クリックし [新規] > [その他] を選択
- ② [Micro Focus IMTK] > [Java インターフェイス] へとナビゲートし [次へ] ボタンを押下



- ③ Java インターフェイス名には「SelS」を指定し、[参照] ボタンを押下
- ④ 追加したプログラムを選択

ICBL	COBOL ソースファイル	-	x
COBOL Y-27	マイルを選択		
▲	232 v_Configuration.bin cbl <		

- ⑤ [完了] ボタンを押下
- ⑥ 入出力をアプリケーションの特性に合わせて、キー値として利用する
   STAFF ID を入力に、照会データの STAFF NAME を出力に設定

	SEL オペレーション - インターフェ	SEL オペレーション - インターフェイス フィールド:					
	名前	方向	型	OCC			
	► LK_STAFF_ID_io ► LK_STAFF_NAME_io	入出力 入出力	int String				
ダブルクリック	「四 フィールド	プロパティ -					
	名前: LK_STAFF_ID_io 型: int 方向: ①入力 〇出力 〇 マッピング LK-STAFF-ID 変更	✓ OCO 入出力	CURS: 0 編集				

各項目をダブルクリックし、[方向] 欄を変更

変更後の画面:

SEL オペレーション - インターフェイス フィールド:									
名前	方向	型							
LK_STAFF_ID_io	አታ	int							
LK_STAFF_NAME_io	出力	String							

- ⑦ Ctrl+S を打鍵し、マッピングを保存
- 15) Solaris サーバの Directory サーバーをサーバーエクスプローラビューへ追加

(1)	サーバーエクスプローラビューを表示
	COBOL ※ をナビゲータ ■サーバー ← E クリック
	サーバーエクスプローラー
	Sybase32 [10.18.11.27:/export/home/yoshihiror]
	▶ 📠 COBOL プログラム
	▷ G Java インターフェイス

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

をクリ	ック	7				
🔁 совоі	۹ ۱	を ナビゲータ 🖪 サーバー 🛛 🕅		) Sel	.cbl 🛙	
	<b>d</b> - 1	51 U. 1. 0.63		0	Sel.cbl	
	- 1111.	Hocalhost 861				
1		新規(N)	۱.	8	Directory Server	接続
		Administration ページを開く	Ctrl+F3	CĴ	その他(0)	Ctrl+N
	~	20184	Delete	_		

③ Solaris サーバのアドレスを指定し、[完了] ボタンを押下

Real 亲	f規 Directory Server 接続 - □ ×
<b>接続の新規作成</b> 既存の Micro Focus Directo	ry Server への接続の新規作成します
名前:	tok-putter
サーバアドレス (IPv4/ホスト名):	10.18.11.27
サーバーポート:	86
?	完了(E) キャンセル

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



16) Enterprise Server へのディプロイ情報を指定

- ① COBOL エクスプローラへ戻る
- ② 追加した Java インターフェイスを右クリックから [プロパティ] を選択

🔓 COBOL 🛛 🕅	ि	- ナビゲータ	א–ש 📃	- )		
				E	<b>\$</b>	$\bigtriangledown$
a 🛃 Sybase32	[10.	18.11.27:/	export/hoi	me/y	oshih	iro
🔋 👂 🔁 COBOL	- プロ	グラム				
a 词 Java 1	ンター	・フェイス				
⊳ <mark>≋⊚</mark> Se <sup>t</sup>		#C+B/NI)				
🛛 🕞 New_		新历纪(IN)		·		
> 🗁 repos	×	削除(D)				
		ディプロイ				
		検査				
		開く				
		クライアント	圭成			
		プロパティ(P	)			
	_					

- ③ [ディプロイメントサーバー] タブを選択
- ④ [変更] ボタンを押下
- ⑤ Solaris サーバの ESDEMO を選択し [OK] ボタンを押下

<b>I</b> GBL	-	×							
ディプロイ先の Enterprise Server を選択してください:									
サーバー	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 説								
ESDEMO	Deployer	Available, Started	Available, Started 10.18.11.27:46872 BitMode=32-Bit						

- 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
- ⑦ プロジェクトディレクトリ配下の New\_Configuration.bin ディレクトリに
   生成された Sel.gnt 及び Sel.idy を選択し [OK] ボタンを押下

● レガシーアプリケーションをディプロイする マプリケーションファイリ・	
/export/home/yoshihirom/test/sybase/remote/sybase32/New_Configuration.bin/Sel.gnt /export/home/yoshihirom/test/sybase/remote/sybase32/New_Configuration.bin/Sel.idy /export/home/yoshihirom/test/sybase/lib32/SYBINIT32.so	ファイル追加 ファイル削除

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

אין דער איז איז דער איז איז דער איז דער דער איז דער איז
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
アプリケーション サーバー JEE 6 v WebLogic 12.1.1 v
✓トランザクション可能

- [3] [インターフェイスタイプ] 欄にて [リモート] を指定6
   インターフェイスタイプ: ○ローカル ●リモート
- ⑭ [Java コンパイラ] 欄にて利用する javac が格納されたディレクトリを指定
- [J2EE クラスパス] 欄にて [参照] ボタンを押下し WebLogic のインスト ールディレクトリに格納されている weblogic.jar を選択

- J2SE と J2EE の属性	E	
Java コンパイラ:	/opt/Java/jdk1.7.0_67/bin/sparcv9	参照
EJB、コネクタ(また、ク	フライアント生成する場合、servlet と JSP)関連の JAR ファイルのパスを追加します。	
J2EE クラスパス:	/opt/oracle/WebLogic/wlserver/server/lib/weblogic.jar	参照

16 [OK] ボタンを押下し、設定画面を閉じる

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

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

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



正常にディプロイできたことをコンソールビューにて確認することができます:



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

<sup>2</sup> e COBOL <sup>1</sup> を ナビゲータ <u>サーバー ☆ <sup>-</sup> □</u> ◎ Sel.cbl ☆											
	ok-putte	er [] <del>0-40</del>	11.07.0	- 1			Sel.o	1			
	ESDE	MO	新規(N)				•				
a 🔳 🗖	  ーカル [	ocal	Adminis	stra	ition ページを開く	Ctrl+F	3 🗕				
	ESDE	мо									
			111121			B-1-1	-				
A 🔺 🕨	Server	ESDEMO	[開始 ◀]								
サーバー	גע (	ナー (3)	サービス	(6)	ハンドラ (2) パッケー	ジ (1)					Ø
サービス	表示フィノ	19 <del>7-</del> 1	ムスペース:		1~L	ーション				クラス: All 🗸 ハンドラ: All 🗸 パッケージ	R: All
1 - 6 of 6	displayab	le namespa	ices from a	tota	l of 6					Show 10 service namespaces at a	time << 前へ 次へ >>
	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 バンドラ	実装 パッ ケージ	現 ステータス	ステー タス ログ	カスダム 機成	説明
	Test	Test 編集…		1	1 CP 1 HTTP Echo top:10.18.11.27*:9002 (tok-putter)			Available	ок		Basic HTTP test service
	Deployer	Deployer 編集…	MF deployment	1	1 CP 1 Web top:10.18.11.27*:35748* (tok-putter)			Available	ок	[MF client] scheme=http URL=/cgi/mfdeploy.exe/uploads accept=application/x- zip-compressed [destination] listener=Web Services and J2EE	Deployment file-upload service
	JES	JES 編集…	MF JES	1	1 CP 1 Web Services and J2EE top:10.18.11.27*:9003 (tok-putter)			Available	ок		Job Entry Subsystem service
	CICS	CICS 編集…	MF CICS	1	1 CP 1 Web Services and J2EE top:10.18.11.27*:9003 (tok-putter)			Available	ок		CICS utility service
	ES	ES 編集…	MF ES	1	1 CP 1 Web Services and J2EE top:10.18.11.27*:9003 (tok-outler)			Available	ок		ES uitility service
削除	Sel	1 of 1 oper	ations show	wn							
		.SEL 編集		1	1 CP 1 Web Services and J2EE top:10.18.11.27*:9003 (tok-putter)	MFRHBINP	Sel	Available	ок		created 13001900240b (00) 2014-09- 25 from SelS.Kv-xtuX/SelS.car

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

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



正常に処理されると

<プロジェクトディレクトリ>/repos/<サービス名>.deploy

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



- Solaris サーバ
- 19) WebLogic の環境設定スクリプト setWLSEnv.sh 編集し、

\$COBDIR/javaee/javaee6/oracleweblogic1211/mfconnector.jar を CLASSPATH に 追加するよう改造して実行

- 20) WebLogic サーバを起動
- 21) Visual COBOL に付属されるトランザクションサポートのない WebLogic 12c 向け のリソースアダプタを WebLogic へ追加
  - ① WebLogic が提供するディプロイメントツール weblogic.Deployer を利用 するための環境変数を設定

- \$ WL\_USER=weblogic;export WL\_USER \$ WL PASSWD=P@sswOrd;export WL PASSWD \$ NAME=mfcobol-notx;export NAME \$ FULL\_PATH\_TO\_THE\_RAR\_FILE=\$COBDIR/javaee/javaee6/oracleweblogic121 1/mfcobol-notx.rar;export FULL\_PATH\_TO\_THE\_RAR\_FILE \$ ディプロイする リソースアダプタ
- ② weblogic.Deployer を使ってリソースアダプタを WebLogic ヘディプロイ

<pre>\$ java -classpath \$WL_HOME/server/lib/weblogic.jar weblogic.Deployer -username \$WI_USER -password \$WI_PASSWD -name \$NAME -deploy \$FUIL P</pre>
ATH_TO_THE_RAR_FILE
weblogic.Deployer がオプション -username weblogic -name mfcobol-notx
-deploy /opt/mf/ED22U1HF4/javaee/javaee6/oracleweblogic1211/mfcobol-
notx. rar を指定して呼び出されました
<pre>&lt;2014/09/25 14 時 01 分 07 秒 JST&gt; <info> <j2ee deployment="" spi=""> <bea-26< pre=""></bea-26<></j2ee></info></pre>
0121> <initiating [ar<="" application,="" deploy="" for="" mfcobol-notx="" operation="" td=""></initiating>
chive: /opt/mf/ED22U1HF4/javaee/javaee6/oracleweblogic1211/mfcobol-n
otx.rar], to configured targets.>
タスク0が開始されました: [Deployer:149026]デプロイ application mfcob
ol-notx on AdminServer.
タスク 0 完了: [Deployer:149026]デプロイ application mfcobol-notx on
AdminServer.
ターゲットの状態: サーバー AdminServer で deploy 完了
¢
ψ

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

#### デプロイメント

インストール     更新     削除     信止 ●     表示項目1 - 1/1     前   次						
	名前 🗠	状態	ヘルス	タイプ	ターゲット	デプロイ順序
	藧 mfcobol-notx	アク ティブ	🖋 ок	リソー ス・アダ プタ	AdminServer	100

- 22) 18) で生成したスタブクライアントを WebLogic ヘインストール
  - ① WebLogic Server Administration Console  $\sim$ ログイン
  - ② 左ペインより対象のドメイン配下の [デプロイメント] をクリック
  - ③ [インストール] ボタンを押下

デプロ	ዝንፖት
12	マトール 更新 削除 起動 ▼ 停止 ▼
	名前 🗠
	🔯 mfcobol-notx

- ④ [パス] 欄に生成された .ear が格納されたディレクトリを指定し Enter を 打鍵
- ⑤ 生成された SelS.ear を選択し、[次へ] ボタンを押下

パス:	/export/home/yoshihirom		
最近使用されたパス:	(なし)		
現在の場所:	tok-putter / export / home / yo		
<ul> <li>管 Client (ディレクトリを開く)</li> <li>com</li> </ul>			
<ul> <li>GelS.ear</li> <li>GelS.jar</li> </ul>			
<ul> <li>SelS.war</li> <li>Imfejblib.jar</li> </ul>			
戻る 次 終了 取消			

- ⑥ [このデプロイメントをアプリケーションとしてインストールする] が選択されていることを確認して、[次へ] ボタンを押下
- ⑦ [終了] ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます:

l	インストール     更新     前除     起動 ▼     停止 ▼     表示項目1 - 2/2     前   次						
		名前 🚕	状態	ヘルス	タイプ	ターゲット	デプロイ順序
		👼 mfcobol-notx	アクテ イブ	🖋 ок	リソース・アダプタ	AdminServer	100
		🗈 🔂 SelS	アクテ イブ	🛩 ок	エンタープライズ・ アプリケーション	AdminServer	100

■ Windows クライアントマシン

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

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



- 24) Enterprise Server デバッグを起動
  - COBOL エクスプローラにてプロジェクトを右クリックから [デバッグ] > [デバッグの構成] を選択
  - ② [COBOL Enterprise Server] をダブルクリック

フィルター入力	
🗄 Apache Tomcat	
IVM COBOL JVM アプリケーション	
🏧 COBOL JVM リモート アプリケーション	

③ [Enterprise Server] 欄の [参照] ボタンを押下

▼ COBOL プロジェクト(P)						
Sybase32	参照					
✓ Enterprise Server						
接続: サーバー:	参照					
▼ デバッグの種類						

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

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

サーバーを選択してください。	
▲ 📃 ローカル [localhost:86]	
ESDEMO	
🖌 a 📃 tok-putter [10.18.11.27:86	5]
ESDEMO	

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

・デハ	ぶっ グの種類
W	Veb サービス Java
ſ	Java サービス名(空白の場合はすべてのサービスをデバッグ)

- ⑥ [デバッグ] ボタンを押下
- ⑦ [パースペクティブの切り替えの確認] 画面にて [はい] を選択
   デバッグパースペクティブにてアタッチ待機状態になっていることを確認で
   きます:

参デバッグ 🛛 🦓 Servers
<ul> <li>新規構成 [COBOL Enterprise Server]</li> <li></li></ul>

- 25) ディプロイしたアプリケーションをデバッグ実行
  - ① WebLogic にディプロイしたスタブクライアントアプリケーションを起動



② [SEL\_LK\_STAFF\_ID\_io] 欄に [10] を入力し [Go!] ボタンを押下

SEL_LK_STAFF_ID_io :	10
	Go!

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



26) ディプロイしたアプリケーションをデバッグ実行

① デバッグ実行

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



変数ビューで現在のステップで参照している変数の中身を確認できます:

参デバッグ 🛛 👭 Servers	🍇 🚽 🕾 🕸 🍸 🗖 🚺 🚧 🕱	● ブレークボイント	💯 🕫 🖻 🏅
a 😹 ESDEMO [COBOL Enterprise Server]	名前		値
A ife COBOL デバッガ: (一時停止)		ODE	+000000000
▲ 🔊 COBOL スレッド:1 (一時停止)	♦ STAF	F-NAME	Smith
/opt/mf/ED22U1HF4/deploy/SelS.LnBl	Z_oY/Sel.gnt:1(行:35)		
▷ 🔐 COBOL スレット:2 (一時停止)	Smith		
	16進:		
	5667622222		
	303400000		
▶ ₩. CODOC XD3110 ( 1014 m)			/
Sel.chl X _ t-/(-: tok-putter		E 701512 8	a, <sup>×</sup>
		A the SEI	* 2
Sel.cbl >		Data Division	
FROM STAFF		A S Procedure Division	
WHERE ID=:STAFF-ID	^	• 1	
END EXEC. DISPLAY "SELECT SQLCODE :" SQL			
DISPLAY "NAME:" STAFF-NAME U	PON CONSOLE		
EXEC SOL RELEASE END-EXEC.			
EXIT PROGRAM.			
<	· · · · · · · · · · · · · · · · · · ·		

データの中身は 16 進表示にすることもできます: 🕪= 変数 🖾 🔪 🗣 ブレークポイント <u>Å</u>\_) ⇒ 名前 値 STAFF-NAME Smith LK-STAFF-NAME < Smith 16進: 1 I 5667622222 3D94800000 L I 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: 🖻 Sel.cbl 🛛 Sel.cbl F8 キーの打鍵でこの行まで 自動で処理を進めることがで きます。 ھ

ブレークポイントプロパティ画面:

ド・アシスト				
	-ド・アシスト	-ド・アシスト	-F•752F	-F•7927

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

	Test client for SelS.SEL
	Back
	Perform the test by entering values:
	SEL_LK_STAFF_ID_io: 10
	Go!
	Result:
	Variable Value
	Result Smith
■ Solaris <sup>+</sup>	トーバ
27) Enterpris	se Server を停止

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

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

発用のデーモンを 停止 \_\_\_\_\_ # \$COBDIR/remotedev/stoprdodaemon Process 19203 located: UID PID PPID C STIME TTY TIME CMD Do you wish to continue and kill it? (y/n): y kill: 19203: no such process Kill signal sent to process 19203 # \_\_\_\_\_

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

- Windows クライアントマシン
- 1) 64 bit 版 Enterprise Server を追加
  - Enterprise Server Administration ページのトップ画面にて [追加] ボタン を押下

<b>アクション</b> アドレス更新 エクスポート	■ ■ Ⅲ 1 - 1 of 1 out of 1 servers Filter タイプ: All ▼ 名前: * ステータス: All
インポート	
すべて削除	Repository: file:///home1/hub/staff/yoshi/opt/mf/ED22U1/etc/mfds/
シャットタワン <b>構成</b>	名 ステータ 通信 フィータン ファータス 3月 マン セキュ ステータス 3月 マン マン マン ステータス 3月 マン
オプション セキュリティ	編集 MFES ESDEMO 停止 1 tep:*5000 -/ 10 Default Server: CP 1: CP 1: 5 ビデー 5 ビデー 10 Server: CP 1: 5 UP 10 Server: CP
<b>表示</b> ディレクトリ 統計 セッション ジャーナル	
<b>ヘルプ</b> このページ Support Feedback	道加 通加

 ② サーバ名を指定し、動作モードを [64-bit] に選択し、[次へ] ボタンを押下 サーバー追加 (Page 1 of 3):

サーバー名: ESDEMO64
動作モード:
32-bit  64-bit
You cannot change your choice of working mode once a s
キャンセル 次へ >>

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



④ オプションの設定欄はデフォルトのまま [追加] ボタンを押下

編集 MFES ESDEMO64 停止 1 tcp:*:* -/10 Defau		
	It Server: CP 1: CP 1: CK	サ 2 ビ 詳細 ス 3 ド ラ パ 0 ケ 追加

トップ画面に戻り、追加されたことが確認できます:

- 2) Enterprise Server の Listener に固定のポートを割り当て
  - Enterprise Server Administration 画面にて追加した Enterprise Server の行にある [編集] ボタンを押下

編集	MFES	ESDEMO64	停止	1 tcp:10.120.56.25*:34801*	- / 10
			用26	2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2	

② [リスナー] タブを選択

🔺 < 🕨 Server ESDEMO64 [停止]					
<b>サーバー…</b> リスナー (2) サービス (2) ハンドラ (3) パッケージ (0)					
<mark>ブロパティ</mark> 構成 診断 過去の統計					

③ それぞれのリスナーの [編集] ボタンを押下し、下表のように設定

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

設定後の画面:

⇒ 通信プロセス 1 🔽 自動起動								
	リスナー	プロセスID	プロセスID コントロールチャネルアドレス					
編集	2 追加	- tcp:*:9004						
		名前		アドレス	ステータス	前回のステータス変更		
	編集	Web Service J2EE	es and	tcp:*:9005	停止	09/28/14-10:10:58		
	編集	Web		tcp:*:9006	停止	09/28/14-10:10:58		

- Solaris サーバ
- 一般ユーザでログインし、Visual COBOL 及び Sybase 64 bit SDK の環境設定スク リプトを実行<sup>8</sup>
- 4) Sybase のライブラリをリンクした共有ライブラリを作成
  - ① Visual COBOL の動作モードを 64 bit に指定

\$ COBMODE=64;export COBMODE
\$ cobmode
Effective Default Working Mode: 64 bit <----\$</pre>

- ② 付録1で作成したダミーの COBOL プログラムをコピー
- ③ Sybase のライブラリをリンクした共有ライブラリを作成9

\$ cob -ze "" dummy.cbl -L\$SYBASE/\$SYBASE\_OCS/lib -lsybcobct\_r64 -lsybct\_r64 -lsybtcl\_r64 -lsybcs\_r64 -lsybcomn\_r64 -lsybintl\_r64 -lsybunic64 -lsocket -lnsl -ldl -lpthread -lthread -lm -o SYBINIT64.so \$

 <sup>&</sup>lt;sup>8</sup> SYBASE の環境設定用スクリプトは LD\_LIBRARY\_PATH\_64 も設定します。Solaris では LD\_LIBRARY\_PATH と LD\_LIBRARY\_PATH\_64 の両者が存在する場合、64 bit リンクには LD\_LIBRARY\_PATH\_64 が使用されます。そのため、この環境変数を維持す る場合は、\$COBDIR/lib を LD\_LIBRARY\_PATH\_64 に追加する必要があります。
 <sup>9</sup> \$SYBASE/\$SYBASE\_OCS/sample/esqlcob に用意されているサンプルを SYBPLATFORM=nthread\_sun\_svr464 でビルドする場合にピックアップされるライブ ラリを指定しています。

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

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

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

```
$ casstart -rESDEM064
....
CASCD0167I ES Daemon successfully auto-started 10:16:00
CASCD0050I ES "ESDEM064" initiation is starting 10:16:00
$
```

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

\_\_\_\_\_

😫 сов 🗞 ታピቻ 📃 サー 🛛 🦳 🗆	□ 🗶 Ħ_/(-: tok-putter 🛙								
tok-putter [10.18.11.27:86]     SEDEMO		Se Enter tok-put	rprise ter (10.1	e Serv 8.11.27:8	er Ad	Iministration			
SEDEMO64 ローカル [localhost:86]	Home	ステータス	K MDS0	000I OK					
ESDEMO	アクション		1 - 2 of	2 out of 2	servers				n
	アドレス更新								
	エクスポート	Filter	タイフ	': All		∨ 名前:*			ステータス: All く Clear
	インポート	Repository: fi	le:///opt/mf	/ED22U1HF	4/etc/mfds	/			
	すべて削除				ステータ	通信	ライセ	セキュリ	ステータス
	シャットダウン		タイプ	名前『	~	プロセス	22	71	DØ
		編集	MFES	ESDEMO	停止	1 top:10.18.11.27*:9000 (tok-putter)	- / 10	Default	Server: CP 1:
	構成		à		FR.80	3 リスナー 詳細			
	オプション								CASS100011 Server manager termination completed successfully 03:47:46
	セキュリティ								37 minutes 8 sec
	<b></b>	(=#	MFES	ESDEMO64	Set:e	1 tcp:10.18.11.27*:9004	- / 10	Default	Server:
< >>	ディレクトリ	編集	79		詳細	(tok-putter)			CP 1:
E アウト ※ ■ ブログ ■ コピー) □□	2年日+		84 64			2 リスナー 詳細			MDS3801I Server started successfully 10:18:01 09/28/14
表示するアウトラインはありません。	セッション				停止				8 minutes 56 sec
	ジャーナル								Started externally using ES ID "mf_mdsa" under system
	2. 200								

- 7) Root ユーザに切り替え
- 8) 環境変数を設定の上、COBOL リモート開発用のデーモンを起動
  - ① Visual COBOL 及び Sybase 64 bit SDK の環境設定スクリプトを実行
  - ② 5) で生成したシンボリックリンクを PATH に追加
  - ③ COBOL リモート開発用のデーモンを起動

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

<sup>&</sup>lt;sup>10</sup> cobpre\_r64 は SYBPLATFORM=nthread\_sun\_svr464 でサンプルをビルドする際に 利用されるプリコンパイラです。

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

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

- COBOL エクスプローラにて作成したプロジェクトを右クリックし、[プロパ ティー]を選択
- ② [Micro Focus] > [ビルド構成] > [COBOL] へとナビゲート
- ③ [ターゲットの種類] 欄を [すべて INT/GNT ファイル] に変更
- ④ [プラットフォームターゲット] 欄を [64 ビット] に指定

ターゲット設定 ターゲットの種類		プラットフォーム	ターゲット
すべて INT/GNT ファイル	~	🔾 32 ビット	● 64 ビット

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

- 12) Sybase 上のデータを照会する埋め込み SQL 文の入った COBOL プログラムをプ ロジェクトに追加(付録1の検証と同じプログラムを利用します。)
  - COBOL エクスプローラにてプロジェクトを右クリックし、 [インポート] > [インポート]
     を選択
  - ② [Remote Systems] > [Remote file system] へとナビゲートし、[次へ] ボタン を押下



③ [Browse] ボタンを押下

leau .	Import	– 🗆 ×
Remote file system Import resources from a remot	e file system.	
From directory :		V Browse

④ [Connection] 欄で Solaris サーバのアドレスを選択

CBL	Browse For Folder	×
Select a fold	er	
Connection:	Local 🗸	New
My Home	Local	
⊳ ♣ My H	10.18.11.27 ome	

- ⑤ 付録1で使用した COBOL リモートプロジェクトのディレクトリヘナビゲ ートし [OK] ボタンを押下
- ⑥ 付録1で使用した Sel.cbl のみにチェックを入れ、[完了] ボタンを押下

▲       →       Sybases2       →       ⇒       .cobolBuild         →       New_Configuration.bin       □       X       .cobolProj         ▲       →       repos       □       inventory.si         ▲       →       SelS.deploy       □       X       .project         ▶       →       com       □       mccerror.txt         →       META-INF       ✓       Sel.cbl	

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

[編集前]

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

[編集後]

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

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

🖳 コンソール 🖾 🕂 問題 🖉 タスク 🔲 プロパティー
COBOL Build
cobol.cfg.New_Configuration: [cobol] [cobol] Compiling (64-bit) Sel.cbl [cobol] [cobol] * Cobsql Integrated Preprocessor [cobol] * CSQL-I-018: Sybase ブリコンバイラトランスレータを起動します。 [cobol] * CSQL-I-018: Sybase ブリコンバイラの出力を処理中。 [cobol] * CSQL-I-001: COBSQL:チェッカへの引ぎ渡しを完了しました。 [cobol] Compilation complete with no errors.
os.init:
os.init.windows:
os.init.unix:
init:
post.build.cfg.New_Configuration:
BUILD SUCCESSFUL Build finished with no errors.
Total time: 2 seconds

- 14) 検証用アプリケーションの COBOL Java 変換マッピングを作成
  - COBOL エクスプローラにてプロジェクトを右クリックし [新規] > [その他] を選択
  - ② [Micro Focus IMTK] > [Java インターフェイス] へとナビゲートし [次へ] ボタンを押下
  - ③ 付録1で使用したものとは異なる Java インターフェイス名「SelS64」を指 定し、[参照] ボタンを押下
  - ④ 追加したプログラムを選択
  - ⑤ [完了] ボタンを押下

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

変更後の画面:

📃 サーバー: tok-putter 🛛 🗈 🛚	Sel.cbl 🛛 🕬 SelS64 🛛				
LINKAGE SECTION:		SEL オペレーション - インターフェイス	フィールド:		
名前	PICTURE	名前	方向	型	OCC
LK-STAFF-ID LK-STAFF-NAME	S9(9) comp-5 X(10)	LK_STAFF_ID_io LK_STAFF_NAME_io	入力 出力	int String	

⑦ Ctrl+S を打鍵し、マッピングを保存

15) Enterprise Server へのディプロイ情報を指定

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



- ② [ディプロイメントサーバー] タブを選択
- ③ [変更] ボタンを押下
- ④ Solaris サーバで稼働する 1) で追加した Enterprise Server ESDEMO64
   を選択し [OK] ボタンを押下

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	説明	
ESDEMO	Deplo	Availabl	10.18.11	BitMode	Deployment file	
ESDEMO64	Deplo	Availabl	10.18.11	BitMode	Deployment file	

- ⑤ [アプリケーションファイル] タブを選択
- ⑥ [レガシーアプリケーションをディプロイする]を選択
- ⑦ [ファイル追加] ボタンを押下
- ③ プロジェクトディレクトリ配下の New\_Configuration.bin ディレクトリに
   生成された Sel.gnt 及び Sel.idy を選択し [OK] ボタンを押下
- ⑨ 再度 [ファイル追加] ボタンを押下し、4) で作成した SYBINIT64.so を追加
   追加後の画面:

◉ 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:
/export/home/yoshihirom/test/sybase/remote/sybase64/New_Configuration.bin/Sel.gnt /export/home/yoshihirom/test/sybase/remote/sybase64/New_Configuration.bin/Sel.idy /export/home/yoshihirom/test/sybase/lib64/SYBINIT64.so

- [EJB 生成] タブを選択
- ① [アプリケーションサーバー] 欄にて JEE 6、WebLogic 12.1.1 を指定
- ③ [Java コンパイラ] 欄にて利用する javac が格納されたディレクトリを指定
- ④ [J2EE クラスパス] 欄にて [参照] ボタンを押下し WebLogic のインスト ールディレクトリに格納されている weblogic.jar を選択

– J2SE と J2EE の属性	ŧ	
Java コンパイラ:	/opt/Java/jdk1.7.0_67/bin/sparcv9	参照
EJB、コネクタ(また、	クライアント生成する場合、servlet と JSP)関連の JAR ファイルのパスを追加します。	
J2EE クラスパス:	/opt/oracle/WebLogic/wlserver/server/lib/weblogic.jar	参照

⑤ [OK] ボタンを押下し、設定画面を閉じる
16) 作成した Java サービスを Enterprise Server ヘディプロイ

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



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

	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	IJZ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:10.18.11.27*:9006 (tok-putter)			Available	ок
	ES	ES 編集…	MF ES	1	CP 1         Web Services and J2EE           tcp:10.18.11.27*:9005         (tok-putter)			Available	ок
削除	Sel	1 of 1 oper	ations shov	vn					
		.SEL 編集		1	CP 1         Web Services and J2EE           tcp:10.18.11.27*:9005         (tok-putter)	MFRHBINP	Sel	Available	ок

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

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



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

😤 COBOL エクス 🛛 😤 ナビゲーター 📕 サーバー エクス 🛛 🖓 🗖
⊑ 🕏 ▽
🔺 🔁 Sybase64 [10.18.11.27:/export/home/yoshihirom/test 🔺
> 📠 COBOL プログラム
Java インターフェイス
Dev
🛛 🗁 repos
🔺 🗁 SelS64.deploy
👂 🗁 Client
> 🗁 com
META-INF
🛃 mfejblib.jar
SelS64.ear
🛃 SelS64.jar
SelS64.mfmak
SelS64.war
Manifest.mf
mccerror.txt

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

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

mfcobo	ol-notxの設定	4	/						
概要	デプロイメント・プラン	構成	セキュリティ	ターゲット	制御	テスト中	監視	ノート	
保存									
ະຫ	ページには、このリソース	・アダプタ	▶ デプロイメント	の基本情報が	「表示され	ヽます。			

⑤ [アウトバウンド接続プール] をクリック

mfcobo	il-notxの設定			1								
概要	デプロイメント	・・プラン	構成	7, 7 - 1, 1, 5	-1	ターゲット	制	ân	テスト中	監視	ノート	
→般	プロパティ	アウトバ	ウンド接	統プール	管	理オブジェクト		<u>9</u> -	クロード	インスト	ゥルメンテ	ーション

⑥ javax.resource.cci.ConnectionFactory を展開

アウト	アウトバウンド接続ブールの構成表						
新規	1 削除						
	グループとインスタンス 🚳						
$\rightarrow$	javax.resource.cci.ConnectionFactory						
±=:+	2186						

⑦ eis/MFCobol\_v1.5 をクリック

アウ	アウトバウンド接続ブールの構成表			
新規				
	グループとインスタンス 🏟			
	□ javax.resource.cci.ConpectionFactory			
	eis/MFCobol_v1.5			

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

アウトバウンド接続のプロパティ			
保存			表示項目 <b>1-8/8</b> 前 次
ブロパティ名 🚕	プロパティのタイプ	プロパティ値	動的な更新のサポート
EnterpriseServerSocketCloseDueToInactivity	java.lang.Integer	72	false
LogHost	java.lang.String	localhost	false
LogPort	java.lang.String	9029	false
ReadTimeoutInSeconds	java.lang.Integer	0	false
SecureGUID	java.lang.Boolean	false	false
ServerHost	java.lang.String	localhost	false
ServerPort	java.lang.String	9005	false
Trace	java.lang.Boolean	false	false

⑨ [保存] ボタンを押下

## アウトバウンド接続のプロパティ

保存			
プロパティ名 💫	プロパティのタイプ	プロパティ値	
EnterpriseServerSocketCloseDueToInactivity	java.lang.Integer	72	1
LogHost	java.lang.String	localhost	1
LogPort	java.lang.String	9029	1
ReadTimeoutInSeconds	java.lang.Integer	0	1
SecureGUID	java.lang.Boolean	false	1
ServerHost	java.lang.String	localhost	1
ServerPort	java.lang.String	9005	1
Trace	java.lang.Boolean	false	1

⑩ デプロイメント・プランの保存確認画面では [OK] ボタンを押下

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

- ① WebLogic Server Administration Console  $\sim$ ログイン
- ② 左ペインより対象のドメイン配下の [デプロイメント] をクリック
- ③ 【インストール】 ボタンを押下
- ④ [パス] 欄に生成された .ear が格納されたディレクトリを指定し Enter を 打鍵
- ⑤ 生成された SelS64.ear を選択し、[次へ] ボタンを押下



- ⑥ [このデプロイメントをアプリケーションとしてインストールする] が選択されていることを確認して、[次へ] ボタンを押下
- ⑦ [終了] ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます:

デプロ	FJIIAS/H							
イン	ストール 更新 削除 起動 (停止 、				表示項目1	-3/3 前 次		
	名前 📣	状態	~JUZ	タイプ	ターゲット	デプロイ順序		
	\overline Əmfcobol-notx	アクテ イブ	🖋 ок	リソース・アダプタ	AdminServer	100		
	🖭 📑 SelS	アク <del>テ</del> イブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100		
	€ CSelS64	アク <del>テ</del> ィブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100		

30) Enterprise Server が動的デバッグ有効で起動されていることを確認
 Enterprise Server Administration 画面を開き、ESDEMO64 の行における [編集]
 ボタンを押下し、確認<sup>11</sup>

<sup>&</sup>lt;sup>11</sup>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。

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:		ンソールログサイズ (K): 0

31) Enterprise Server デバッグを起動

- COBOL エクスプローラにてプロジェクトを右クリックから [デバッグ] > [デバッグの構成] を選択
- ② [COBOL Enterprise Server] をダブルクリック
- ③ [Enterprise Server] 欄の [参照] ボタンを押下
- ④ Solaris サーバ上で稼働する ESDEMO64 を選択し、[OK] ボタンを押下

サーバーを選択してください。
▲ 🔜 ローカル [localhost:86]
ESDEMO
Itok-putter [10.18.11.27:86]
ESDEMO
ESDEMO64

 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

• ī	デバッグの種類
	Web サービス Java
	Java サービス名(空白の場合はすべてのサービスをデバッグ)

- ⑥ [デバッグ] ボタンを押下
- ⑦ [パースペクティブの切り替えの確認] 画面にて [はい] を選択
   デバッグパースペクティブにてアタッチ待機状態になっていることを確認できます:

参デバッグ 🛱 🖓 Servers
<ul> <li>▲ SDEMO64 [COBOL Enterprise Server]</li> <li></li></ul>

32) ディプロイしたアプリケーションをデバッグ実行

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

<del>(</del> )	Ø + ♂ ★ Ø = ○ ★ Ø = ○ ★ Ø = ○ ★ Ø = ○ ↓
File Edit	: View Favorites Tools Help
Test	client for SelS64.SEL
Back	
Perfor	m the test by entering values:
SEL_L	K_STAFF_ID_io: 0
Back	ブラウザを立ち上げ、 http:// <solaris サーバのアドレス="">:<weblogic のポート="">/SelS64/SEL.js をアドレスバーに入力</weblogic></solaris>

② [SEL\_LK\_STAFF\_ID\_io] 欄に [20] を入力し [Go!] ボタンを押下

SEL_LK_STAFF_ID_io :	20
	Go!

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

	Sel.cbl 🛛	3
	Sel.	cbl
		*A·1·3······2·····3·····4·····5······6·····7··
	Θ	<pre>\$SET INITCALL(SYBINIT64) p(cobsql) COBSQLTYPE=SYBASE END-C ENDP IDENTIFICATION DIVISION. PROGRAM-ID. Sel. ENVIRONMENT DIVISION.</pre>
	Θ	DATA DIVISION.
	Θ	WORKING-STORAGE SECTION.         EXEC SQL BEGIN DECLARE SECTION END-EXEC.         D1 STAFF-ID       PIC 39(9) COMP-5.         D1 STAFF-NAME       PIC X(10).         EXEC SQL END DECLARE SECTION END-EXEC.         EXEC SQL INCLUDE SQLCA END-EXEC.
	Θ	LINKAGE SECTION. 01 LK-STAFF-ID PIC S9(09) COMP-5. 01 LK-STAFF-NAME PIC X(10).
	0	PROCEDURE DIVISION USING LK-STAFF-ID LK-STAFF-NAME.
۵		DISPLAY "CONNECT STEP" UPON CONSOLE.
		EXEC SQL CONNECT "sa" IDENTIFIED BY "sapass"

③ デバッグ実行

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

🏇 デバッグ 🕄 🔍 Servers 🍡 📲 🕾 🕸 🏱 🗖 🗍	(い- 変数 22 💁 ブレークポイント 🕼 💏 🖻 🎽 🗖
<ul> <li>▲ SDEMO64 [COBOL Enterprise Server]</li> <li>▲ COBOL デルッボ: (一時停止)</li> <li>▲ COBOL スルッド:1 (一時停止)</li> <li>■ COBOL スルッド:2 (一時停止)</li> <li>▶ @ COBOL スルッド:3 (一時停止)</li> <li>▶ @ COBOL スルッド:4 (一時停止)</li> <li>▶ @ COBOL スルッド:5 (一時停止)</li> <li>▶ @ COBOL スルッド:6 (一時停止)</li> </ul>	名前 値 ◆ SQLCODE +000000000000000000000000000000000000
ତ Sel.cbl ଅ ତ Sel.cbl	□     Image: Polystyle     Image: Polystyle       □     Image: Polystyle     Image: Polystyle       Image: Polystyle     Image: Polystyle </th
	CHRUGE Sector      LeSTAFF-ID PIC S9(9) COMP-5, サイズ: 4      LESTAFF-ID PIC S9(9) COMP-5, サイズ: 4      LESTAFF-INAME PIC X(10), サイズ: 10      Procedure Division      Il      SQL-WORST      SQL-PROLOG      SQL-TPILIOG
INIU 35147-HAME FROM STAFF WHERE ID::STAFF-ID END-EXEC. DISPLAY "SELECT SOLCODE:" SOLCODE UPON CONSOLE. DISPLAY "SELECT SOLCODE:" SOLCODE UPON CONSOLE. MOVE STAFF-NAME TO LK-STAFF-NAME.	

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

Test client for SelS64.SEL					
Back					
Perform	the test by	enteri	ng values	5:	
SEL_LK	_STAFF_D	D_io :	20		Go!
Result:					
Variable	Value				
Result	Takeshi				
Back					

■ Solaris サーバ

33) Enterprise Server を停止

```
$ casstop -rESDEM064
CASST0005I Shutdown of ES ESDEM064 starting 15:35:06
CASSI8003I Enterprise Server "ESDEM064" termination completed 15:35:06
Return code: 0
$
```

```
34) COBOL リモート開発用のデーモンを開始したセッションにて COBOL リモート開
発用のデーモンを 停止
# $COBDIR/remotedev/stoprdodaemon
Process 19495 located:
UID PID PPID C STIME TTY TIME CMD
Do you wish to continue and kill it? (y/n): y
kill: 19495: no such process
Kill signal sent to process 19495
#
```

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

- Solaris サーバ
- 一般ユーザでログインし、Visual COBOL 及び Sybase 32 bit SDK の環境設定スク リプトを実行
- 2) Sybase の XA スイッチモジュールを作成
  - ① Sybase 用のスイッチモジュールソースは Visual COBOL に同梱されてい ないため、以下のソースを持つプログラムを用意

\$ cat ESSYBASEXA.cbl \*> \*> Micro Focus Server Express XA switch module for Sybase. \*> \*> (C) Copyright 2005-2014 Micro Focus (IP) Limited \*> All Rights Reserved. \*> IDENTIFICATION DIVISION. PROGRAM-ID. ESSYBASEXA. ENVIRONMENT DIVISION. DATA DIVISION. WORKING-STORAGE SECTION. 01 wk-test pic x(01). 01 SybasePtr USAGE PROCEDURE-POINTER. USAGE PROCEDURE-POINTER. 01 SybaseXaPtr 01 SybaseXaPtr-pointer USAGE POINTER REDEFINES SybaseXaPtr. 01 MFocusCasPtr USAGE PROCEDURE-POINTER. LINKAGE SECTION. 01 XA-SWITCH PIC X(128). PROCEDURE DIVISION. INITIALIZE wk-test. \*> Load Enterprise Server module SET MFocusCasPtr TO ENTRY "casaxlib.so" \*> Attempt to load a pointer to the Sybase XA switch struc ture SET SybaseXaPtr TO ENTRY "sybase\_TXS\_xa\_switch". SET ADDRESS OF XA-SWITCH TO SybaseXaPtr-pointer. GOBACK RETURNING SybaseXaPtr. \$

```
② Visual COBOL の動作モードを 32 bit に指定
```

```
$ COBMODE=32;export COBMODE
$ cobmode
Effective Default Working Mode: 32 bit <------
$</pre>
```

③ 用意したスイッチモジュールを Visual COBOL でビルド<sup>12</sup>

\$ cob -z, sys, nounload ESSYBASEXA.cbl -to ESSYBASEXA.so -e "" -L\$SYBA
SE/\$SYBASE\_OCS/lib -lsybcobct\_r -lsyblk\_r -lsybct\_r -sybtcl\_r -lsyb
cs\_r -lsybcomn\_r -lsybintl\_r -lsybunic -lsybxadtm -lsocket -lnsl -ld
l -lpthread -lthread -lm
\$

32 bit 用のスイッチモジュールが生成されます:

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

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

\$SYBASE/\$SYBASE\_OCS/xa\_config に下記のファイルを準備:

 <sup>&</sup>lt;sup>12</sup> 付録1にて共有ライブラリを作成した際にリンクしたライブラリに加えて、
 \$SYBASE/\$SYBASE\_OCS/libsybxadtm.so 及び libsybblk\_r.so を追加でリンクしています。

4) DTM\_TM\_ROLE を持った SYBASE のユーザを作成

```
① ISQL を使って sa ユーザ SYBASE にログイン
```

- **\$** isql -Usa -Psapass -STOKPUTTER 1>
- ② XA 接続で利用するユーザを追加

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

③ 追加したユーザが pubs2 データベースで利用できるように設定

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

④ 追加したユーザのデフォルト DB を pubs2 に指定

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

⑤ 追加したユーザに検証で利用するテーブルを操作するための権限を付与

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

⑥ 追加したユーザに dtm\_tm\_role 権限を付与

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

- ⑦ 「enable DTM」構成パラメータを有効化 1> sp\_configure "enable DTM", 1 2> go Parameter Name Default Memory Used ----- 徐中省略 -----(1 row affected) 設定オプションが変更されました。このオプションは静的なので、変更を反 映するために Adaptive Server をリブートしてください。 'enable DTM' の値を変更しても、Adaptive Server で使用するメモリの量は増加しません。 (return status = 0)1>2> \_\_\_\_\_
- 5) SYBASE Adaptive Server をリブート
  - ⑧ SYBASE Adaptive Server を停止

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

- ⑨ SYBASE Adaptive Server を起動
   SYBASE Adaptive Server 起動スクリプトを実行
- 6) Enterprise Server にスイッチモジュールを追加
  - ① Enterprise Administration 画面を開く
  - ② ESDEMO の列で [編集] ボタンを押下
  - ③ [XA リソース] タブをクリック

🔺 🚽 🕨 Server ESDEMO [停止]	
<b>サーバー…</b> リスナー (3) サービス (6) ハンドラ (2) パッケージ (1)	
<b>プロパティ</b> 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: ESDEMO ×	
開始オブション:	

④ [追加] ボタンを押下

				-
一般)	XAリソース (0)	MSS	MQ ]	5
追加				

⑤ Open 文字列等 XA リソースに関する情報を指定<sup>13</sup>

-般 XAUV-ス(1) MSS MQ スクリプト アクセス権 セキュ!	ノティ
ID: ESSYBXA 名前: SYBASE_XA_SERVER 2) で作成したスイッチ モジュール	
モジュール: /export/home/yoshihirom/test/sybase/lib32/ESSYBASEXA.so く OPEN文字列: -Nconnection32 -Uxauser -Ppassword	
CLOSE 文字列: 説明: xa_config ファイルで指 はm tm role を持つ	2
定した LRM 名 一ザ 有効: ☑	

⑥ [追加] ボタンを押下

XA リソースが追加されたことを確認できます:

< > S	Server I	ESDEMO	)[停止	:]							
<b>サーバー…</b> リスナー (3) サービス (6) ハンドラ (2) パッケージ (1)											
プロパテ	<b>イ</b>	構成	診断.	) 過去	の統計						
一般	XAL	<b>リソース</b> (	1)	MSS	MQ	スクリプト	アクセス構	Ī	セキュリ	7	
	有効	ID	名前		ಕಲೆユール			Open 🕉	[字列]	Close 文字列	説 明
編集	*	ESSYBXA	SYBASE	_XA_SERVER	/export/home/yoshi	ihirom/test/sybase/lib32/E	SSYBASEXA.so	-Nconr Uxaus	ection32 - er -Ppassword		
追加											

<sup>&</sup>lt;sup>13</sup> Open 文字列の詳細は Sybase のマニュアルを参照願います。

7) Enterprise Server を起動

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

XA スイッチモジュールが正しく構成され、動作することを Enterprise Server のコ ンソールログより確認できます。下図は Enterprise Server Administration 画面にて、 [編集] ボタン > [診断] タブ > [ES コンソール] とナビゲートし確認したコンソール画面になります: ESコンソール CSコンソール トレース ダンプ O Show entries from 1 to 10 画面更新 of 35 total entries Show last 10 lines Entry Event Show Entire Log 140926 19024460 14934 ESDEMO CASXO0020I ESSYBXA XA interface loaded. Name(SYBASE\_XA\_SERVER), Registration Mode(Static) 19:02:44 27 140926 19024462 CASCD10711 Administration SEP created for Server ESDEMO, process-id = 14937 19:02:44 28 140926 19024468 14937 ESDEMO CASSI1500I SEP initialization started 19:02:44 29 140928 19024489 14938 ESDEMO CASCS5001I Communications interface 01 initialization started 19:02:44 30 140928 19024472 14938 ESDEMO CASCS5003I Communications interface 01 initialization complete 19:02:44 31 140926 19024473 14934 ESDEMO CASXO0015I ESSYBXA XA interface initialized successfully 19:02:44 32 140928 19024478 14934 ESDEMO CASSI5040I Active SEP memory strategy set to x'00000001', retain count 100 19:02:44 33 140926 19024564 14935 ESDEMO CASSI1600I SEP initialization completed successfully 19:02:45 34 140926 19024573 14937 ESDEMO CASSI1600I SEP initialization completed successfully 19:02:45 35 140926 19024658 14936 ESDEMO CASCS5100I Communications Process instance 01 is ready to accept requests 19:02:46

8) 付録1の要領で COBOL リモート開発用のデーモンを起動

■ Windows クライアントマシン

- 9) Visual COBOL for Eclipse を起動
- 10) 付録1で使用したリモートプロジェクトを開く
- Sybase 上のデータに対して DML 文を発行する埋め込み SQL 文の入った COBOL プログラムをプロジェクトに追加<sup>14</sup>
  - ① COBOL エクスプローラにてプロジェクトを右クリックし、
     [新規] > [COBOL プログラム]
     を選択
  - (1) [新規ファイル名] 欄では [UPP.cbl] を指定
  - 12 [完了] ボタンを押下

<sup>14</sup> 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 ソースビューに以下プログラムをコーディング<sup>15</sup>



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

 ④ Ctrl+S を打鍵し、ソースを保存 自動ビルドが有効なため、ビルド処理が走ります。UPP.cbl をコンパイルし、 動的ロードモジュールが生成されたことを COBOL エクスプローラより確 認できます:



12) アプリケーションの COBOL – Java 変換マッピングを作成

- COBOL エクスプローラにてプロジェクトを右クリックし [新規] > [その他] を選択
- ② [Micro Focus IMTK] > [Java インターフェイス] へとナビゲートし [次へ] ボタンを押下
- ③ Java インターフェイス名「UPPS」を指定し、[参照] ボタンを押下
- ④ 追加したプログラムを選択

I CBL	COBOL ソースファイル	-	×
COBOL ソースファイル	を選択		
a 🗁 Sybase32			
🛛 🕞 New_C	Configuration.bin		
🛛 🕞 repos			
Sel.cbl			
UPP.cb			

⑤ [完了] ボタンを押下

ここでは、デフォルトのマッピングをそのまま利用します:

	🚽 サーバー: tok-putter 👘 🚺 Sel	.cbl 🖻 UPP.cbl	80	UPPS 8			
LINKAGE SECTION: UPP オペレーション - インターフェイス フィールド:							
L	名前	PICTURE		名前	方向	型	(
	LK-STAFF-ID	S9(9) comp-5		LK_STAFF_ID_io	入出力	int	
L	LK-STAFF-NAME	X(10)		LK_STAFF_NAME_io	入出力	String	
	🗢 LK-Commit-Or-Rolback	X		► LK_Commit_Or_Rolback	2 入出力	String	

- 13) Enterprise Server へのディプロイ情報を指定
  - COBOL エクスプローラにて追加した Java インターフェイスを右クリッ クから [プロパティ] を選択



- ② [ディプロイメントサーバー] タブを選択
- ③ [変更] ボタンを押下
- ④ Solaris サーバの ESDEMO を選択し [OK] ボタンを押下

ICBL		Enterprise Se	rver を選択	-	×	
ディプロイ先の Enterprise Server を選択してください:						
サーバー	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	説明	
ESDEMO	Deployer	Available, Started	10.18.11.27:46872	BitMode=32-Bit	D	

⑤ [トランザクション管理] 欄にて [コンテナ管理] を選択

nterprise Server 名:		
ESDEMO (10.18.11	.27:39488)	変更
]Enterprise Server 🕽	行時環境の使用	
	Enterprise Server 実行時環境の構成	
ービス名:		
UPP		
トランザクション管理		

- 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

⑦ プロジェクトディレクトリ配下の New\_Configuration.bin ディレクトリに
 生成された UPP.gnt 及び UPP.idy を選択し [OK] ボタンを押下
 追加後の画面:

 ●レガシーアグ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:



- [EJB 生成] タブを選択
- ① [アプリケーションサーバー] 欄にて JEE 6、WebLogic 12.1.1 を指定
- ① [インターフェイスタイプ] 欄にて [リモート] を指定
   インターフェイスタイプ: 〇ローカル ④リモート
- ③ [Java コンパイラ] 欄にて利用する javac が格納されたディレクトリを指定
- ④ [J2EE クラスパス] 欄にて [参照] ボタンを押下し WebLogic のインスト
   ールディレクトリに格納されている weblogic.jar を選択
- ⑤ [OK] ボタンを押下し、設定画面を閉じる

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

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:

	ES	ES 編集…	MF ES	1	1 CP 1 Web Services and J2EE top:10.18.11.27*:9003 (tok-putter)			Available	ок	
削除	Sel	1 of 1 oper	ations show	vn						
		.SEL 編集		1	1 CP 1 Web Services and J2EE top:10.18.11.27*:9003 (tok-putter)	MFRHBINP	Sel	Available	ок	
削除	UPP	1 of 1 oper	ations shov	vn						
		.UPP 編集		1	CP 1         Web Services and J2EE           top:10.18.11.27*:9003 (tok-putter)	MFRHBINP	UPP	Available	ок	

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

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



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

Sybase32 [10.18.11.27:/export/home/yoshihirom/test/syba	^
> 📠 COBOL プログラム	
🔉 🕞 Java インターフェイス	
🖻 🗁 New_Configuration.bin	
🔺 🗁 repos	
> 🗁 SelS.deploy	
🛛 🗁 UPPS.deploy	
> 🗁 Client	
> 🗁 com	
META-INF	
🛃 mfejblib.jar	
UPPS.ear	
📓 UPPS.jar	
UPPS.mfmak	
UPPS.war	

- Solaris サーバ
- 16) 付録1で WebLogic にディプロイしたリソースアダプタを削除
  - 15 WebLogic Administration Console 画面を開き、ディプロイ済みのリソース アダプタを選択

インストール     更新     削除       起動 >     停止 >     表示項目1 - 3/3 前   次							
	名前 🊕	状態	ヘルス	タイプ	ターグット	デプロイ順序	
	👼 mfcobol-notx	アク <del>テ</del> イブ	🖋 ок	リソース・アダプタ	AdminServer	100	
	€ CSelS	アク <del>テ</del> イブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100	
	€ SelS64	アクテ イブ	🖋 ок	エンタープライズ・ アプリケ <i>ー</i> ション	AdminServer	100	

16 [削除] ボタンを押下

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

① WebLogic が提供するディプロイメントツール weblogic.Deployer を利用

するための環境変数を設定

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



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

デプロイメノト

イン	インストール         更新         前除         停止 ▼         表示項目1 - 3/3 前   次						
	名前 🗠	状態	ヘルス	タイプ	ターゲット	デプロイ順序	
	👼 mfcobol-xa	アクテ イブ	🖋 ок	リソース・アダプタ	AdminServer	100	
	€ CselS	アク <del>テ</del> イブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100	
	€ To SelS64	アク <del>テ</del> イブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100	

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

- ① WebLogic Server Administration Console  $\sim$ ログイン
- ② 左ペインより対象のドメイン配下の [デプロイメント] をクリック
- ③ 【インストール】 ボタンを押下
- ④ [パス] 欄に生成された .ear が格納されたディレクトリを指定し Enter を 打鍵
- ⑤ 生成された UPPS.ear を選択し、[次へ] ボタンを押下

現在の場所:	tok-putter / export / home / yoshihirom / test / sy
🔘 🎥 Client (ディレク	トリを開く)
🗀 com	
💿 🗔 UPPS.ear 🔶	
💿 🦲 UPPS.jar	
💿 🧕 UPPS.war	
💿 🦲 mfejblib.jar	

- ⑥ [このデプロイメントをアプリケーションとしてインストールする] が選択されていることを確認して、[次へ] ボタンを押下
- ⑦ [終了] ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます:

デプ	тјп-(у.)-							
イン	インストール 更新 削除 停止 ▼ 表示項目1-4/4 前   次							
	名前 🗠	状態	ヘルス	タイプ	ターゲット	デプロイ順序		
	藧 mfcobol-xa	アクテ イブ	🖋 ок	リソース・アダプタ	AdminServer	100		
	🗷 📑 SelS	アクテ イブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100		
	. ■ SelS64	アクテ イブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100		
	₩ 📑 UPPS	アク <del>テ</del> イブ	🖋 ок	エンタープライズ・ アプリケーション	AdminServer	100		

■ Windows クライアントマシン

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

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

- 20) Enterprise Server デバッグを起動
  - COBOL エクスプローラにてプロジェクトを右クリックから [デバッグ]>[デバッグの構成] を選択
  - ② 付録1で作成したデバッグ構成をダブルクリック

<ul> <li>□ □ □ □ □ □ □ □</li> <li>□ □ □</li> <li>□ □</li></ul>	名前(N): ESDEMO <u> ほんりーズ</u> = 共通(C) <u> ほん</u> デバッグシンボル	
Apache Tomcat COBOL Enterprise Server	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	ンを開始します。
SESDEMO COBOL JVM アプリケーション	Sybase32	参照
MM COBOL JVM リモート アノリケーション 電 COBOL アプリケーション いていたい でのBOL コアダンプ でのBOL のアプリケーションのアタッチの待機	◆ Enterprise Server 接続: tok-putter サーバー: ESDEMO ◆ デバッグの種類	参照

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



21) ディプロイしたアプリケーションをデバッグ実行

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

		0 44 0 1	
	ok-putter:///UPPS/UPP.jsp	ク・C X @ Test Clien	t for UPPS.UPP ×
File Edit View Fa	vorites 70ls Help		
Test client	for UPPS.UPP		]
Back			
Perform the test	by entering values:		
UPP_LK_STAFI	F_ID_io : 0		
UPP_LK_STAFI	F_NAME_io :		
UPP_LK_Comm	it Or Rolback io :		
	Gol		
Back			
	ブラウザを立ち上げ、 http:// <solaris サーバのアドレス=""> をアドレスバーに入力</solaris>	: <weblogic th="" のポー<=""><th>ート&gt;/UPPS/UPP</th></weblogic>	ート>/UPPS/UPP

② [UPP\_LK\_STAFF\_ID\_io] 欄に [10] を
 [UPP\_LK\_STAFF\_NAME\_io] 欄に [Hogan] を
 [UPP\_LK\_Commit\_Or\_Rollback\_io] 欄に [C] を
 入力し [Go!] ボタンを押下

Perform the test by entering values:

UPP_LK_STAFF_ID_io :	10
UPP_LK_STAFF_NAME_io :	Hogan
UPP_LK_Commit_Or_Rolback_io :	С
	Go!

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

🏇 デパッグ 🕄 🚜 Servers 🦓 🚽 🕾 🌞 🎽 🕾	' 🗆 😡= 変数 😒	● ブレークポイント	🗄 🏘 🕞 🍸 🗖 🗋
▲ 1% ESDEMO [COBOL Enterprise Server] ▲ 1% COBOL デバッガ:(一時厚止) ▲ 1% COBOL スレッデ:1(一時停止) ■ (nat/mf/ED2211HE4/dealav/UJPPS.7Nh01Vh1/UJPP.ant:11(行:2)	名前		値
	<		> ~
UPP.cbl 🛙		E アウトライン ※	
UPP.cbl  At 1- 2	· · · · · · · · ·	Procedure Division     1     SQL-WORST     SQL-PROLOG     SQLREJULTS     SQLRESULTS     SQLRESULTS     SQLLAST     SQLCTXERR     SQLCSBDIAG     SQLSETINTRERR	^
EXEC'SOL USE PUBSE PUBSE PUBSE DISPLAY "USE SOLCODE:" SOLCODE UPON CONSOLE. IF SOLCODE NOT = 0 DISPLAY "MSG:" SOLERRMC UPON CONSOLE EXIT PROGRAM	>	<ul> <li>SQLDESCCONN</li> <li>SQLCONNOP</li> <li>SQLCMDOP</li> <li>SQLCUROP</li> <li>SQLSTMOP</li> </ul>	~

③ デバッグ実行

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

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



ダー ナハック ン	2 🖓 Servers 🛛 🙀 🖓 🕾 🕸 🖓	□ (×)= 変数 🛛	∞ ブレークポイント	£.,
<ul> <li>▲ Sesse</li> <li>▲ Sesse</li></ul>	EMO [COBOL Enterprise Server] :OBOL デバッ方:(一時停止) © COBOL スレッド:1(一時停止) ■ /opt/mf/ED22U1HF4/deploy/UPPS.7NhO1Vbl/UPP.gnt:1(行:44 © COBOL スレッド:2(一時停止) © COBOL スレッド:3(一時停止) © COBOL スレッド:5(一時停止) © COBOL スレッド:6(一時停止)	名前 ◆ SQLCOI ◆ LK-COM く C 16進: 4 3 く	DE IMIT-OR-ROLBACK	値 +000000000 C
UPP.cbl 5 UPF.cbl 5	Image: State in the state	······	P7:547> ⊠     Procedure Division     1     SQLPROLOG     SQLPROLOG     SQLPROTECT     SQLPROTECT     SQLLAST     SQLCSBDIAG     SQLSETINTRERR     SQLSETINTRERR     SQLDESCCONN     SQLDESCCONN	

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

# Test client for UPPS.UPP

### Back

Perform the test by entering values:

UPP_LK_STAFF_ID_io :	10
UPP_LK_STAFF_NAME_io :	Hogan
UPP_LK_Commit_Or_Rolback_io :	С
	Go!

**Result:** 

Variable	Value
LK_STAFF_ID_io	10
LK_STAFF_NAME_io	Hogan
LK_Commit_Or_Rolback_io	С

Back

■ Solaris サーバ

22) アプリケーションがレコードを正しく更新していることを確認

\$ isql - 1> use p 2> go 1> selec 2> go	-Usa -Psapass - oubs2 ct * from STAFF	STOKPUTTER			
ID	NAME	DEPT	JOB	YEARS	SALARY
	10 Hogan	333	Mgr	7	18357.50
	20 Takeshi 612 45	333	Sales	8	18171.25
	30 Marenhi 0.00	38	Mgr	5	17506. 75
(3 rows 1>	affected)				
	付録1の検証に 「Smith」が格 たが、「Hogan」 ます。	こて照会した際は 納されていまし   に更新されてい			

■ Windows クライアントマシン

- 23) トランザクションが取り消される条件でパラメータを渡し、ディプロイしたアプリケ ーションをデバッグ実行
  - ① WebLogic にディプロイしたスタブクライアントアプリケーションを起動
  - ② [UPP\_LK\_STAFF\_ID\_io] 欄に [10] を
     [UPP\_LK\_STAFF\_NAME\_io] 欄に [Flair] を
     [UPP\_LK\_Commit\_Or\_Rollback\_io] 欄に [R] を
     入力し [Go!] ボタンを押下

Perform the test by entering values:

UPP_LK_STAFF_ID_io :	10
UPP_LK_STAFF_NAME_io:	Flair
UPP_LK_Commit_Or_Rolback_io :	R
	Go!

③ デバッグ実行

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

DIPR.cbl 🛛		- 0	E アウトライン 🛙
	-hl		• IDX
		-	🕟 😤 Linkage
	FIL PRINKAN	• • • • •	A S Procedure
	END-IF	^	0.1
	EXEC SQL USE pubs2 END-EXEC.		
	DISPLAY "USE SQLCODE:" SQLCODE UPON CONSOLE.		
	IF SQLCODE NOT = 0		SQLP
	FYIT DOCCOAN		SQLE
	FND-TF		SQLR
	MOVE LK-STAFF-ID TO STAFF-ID.		SOLP
	MOVE LK-STAFF-NAME_TO STAFE-NAME.		
	EXEC SQL UPDATE STAFF SET NAME=:STAFF-NAME		
	WHERE ID=:01AFF-ID		0 SQLC
	DISPLAY "UPDATE = " SOLCODE UPON CONSOLE.		SQLC
	IF SQLCODE NOT = 0		
	DISPLAY "MSG." SOLEDDA WORKING-Storage Section, PIC 59(18) C	JMP, ⊗	照= 0, 91ス= 8 八1 F
UPDATE 又実行直	SQLCODE=+000000000000000000000000000000000000		
後の SOLCODE	<b>个</b>		
BOD DOLLOUDE			

ステップをそのまま進めていきますと第3パラメータに「R」を指定したため、 10 回の繰り返し項目の添え字に 11 を格納するロジックへ入ります: EXEC SQL INCLUDE SQLCA END-EXEC. 01 TABLE-ITEM PIC X OCCURS 10 TIMES INDEXED BY IDX. LINKAGE SECTION. PROCEDURE DIVISION USING LK-STAFF-ID LK-STAFF-NAME LK-Commit-Or-Rolback. 1. DISPLAY "Update TEST" UPON CONSOLE. EXEC SQL SET CONNECTION "connection84" END-EXEC. DISPLAY "CONNECT SQLCODE:" SQLCODE UPON CONSOLE. IF SQLCODE NOT = 0 DISPLAY "MSG:" SQLERRMC UPON CONSOLE EXIT PROGRAM END-IF END-IF EXEC SQL USE pubs2 END-EXEC. DISPLAY "USE SQLCODE:" SQLCODE UPON CONSOLE. IF SQLCODE NOT = 0 DISPLAY "MSG:" SQLERRMC UPON CONSOLE EXIT PROGRAM END-IF NOVE LK-STAFF-ID TO STAFF-ID. Nove LK-Staff-Name to Staff-Name. Exec SQL update Staff Set Name=:Staff-Name WHERE ID=:STAFF-ID END-EXEC. DISPLAY "UPDATE = " SQLCODE UPON CONSOLE. IF SQLCODE NOT = 0 DISPLAY "MSG:" : SQLERRMC UPON CONSOLE EXIT PROGRAM



END-IF

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

#### Error 500--Internal Server Error



■ Solaris サーバ

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

	\$ isql -L 1> use pu 2> go 1> select 2> go	Jsa -Psapass -S ubs2 t * from STAFF	STOKPUTTER			
	ID	NAME COMM	DEPT	J0B	YEARS	SALARY
		10 Hogan 0.00	333	Mgr	7	18357. 50
		20 Takeshi 612.45	333	Sales	8	18171.25
		30 Marenhi 22) で確認した	38 「Hagan」がその	Mgr	5	17506. 75
_	(3 rows ; 1>	22) で確認した まま残っていま	「nogan」 かその す。	J		

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

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

- Solaris サーバ
- 1) 一般ユーザでログインし、Visual COBOL 及び Sybase 64 bit SDK の環境設定スク リプトを実行
- 2) Sybase の XA スイッチモジュールを作成
  - ① 付録3で用意したスイッチモジュール用のソース ESSYBASEXA.cbl を作 業ディレクトリヘコピー
  - ② Visual COBOL の動作モードを 64 bit に指定

<pre>\$ COBMODE=64;export COBMODE</pre>	
\$ cobmode	
Effective Default Working Mode: 64 bit	
S	

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

\$ cob -z, sys, nounload ESSYBASEXA, cbl -to ESSYBASEXA, so -e "" -L\$SYBA SE/\$SYBASE\_OCS/lib -lsybcobct\_r64 -lsybblk\_r64 -lsybct\_r64 -lsybtcl\_ r64 -lsybcs\_r64 -lsybcomn\_r64 -lsybintl\_r64 -lsybunic64 -lsybxadtm64 -lsocket -lnsl -ldl -lpthread -lthread -lm \$

64 bit 用のスイッチモジュールが生成されます:

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

<sup>16</sup> 付録2にて共有ライブラリを作成した際にリンクしたライブラリに加えて、 \$SYBASE/\$SYBASE\_OCS/libsybxadtm64.so 及び libsybblk\_r64.so を追加でリンクし ています。

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

\$SYBASE/\$SYBASE\_OCS/xa\_config に下記のファイルを準備:

```
$ cat $SYBASE/$SYBASE_OCS/config/xa_config
; Comment line as first line of file REQUIRED!
[all]
        logfile="xalog64.txt"
        traceflags="all"
        properties="CS_LOGIN_TIMEOUT"="60"
[xa]
        lrm=connection64
        server=TOKPUTTER
$
```

- 4) Enterprise Server にスイッチモジュールを追加
  - ① Enterprise Administration 画面を開く
  - ② ESDEMO64 の列で [編集] ボタンを押下
  - ③ [XA リソース] タブをクリック
  - ④ [追加] ボタンを押下
  - ⑤ Open 文字列等 XA リソースに関する情報を指定17

-般 XAUVース (0) MSS MQ	スクリプト アクセス権 セキュリティ
ID: ESSYBXA 名前: SYBASE_XA_SERVER	2) で作成したスイッチ モジュール
モジュール: /export/home/yoshihirom/test/sybase/lib64/ES OPEN文字列: -Nconnection64 -Uxauser -Ppassword	SSYBASEXA.so
CLOSE文字列:	
説明: xa_config ファ・ 定した LRM 名 有効: ☑	イルで指 イルで指 dtm_tm_role を持つユ ーザ

<sup>17</sup> Open 文字列の詳細は Sybase のマニュアルを参照願います。

⑥ [追加] ボタンを押下

XA リソースが追加されたことを確認できます:

	Server I	ESDEMO	64 [ 停	[止]								
<mark>サーバー…</mark> リスナー (2) サービス (3) ハンドラ (3) パッケージ (1)												
<b>プロパティ</b> 構成 診断 過去の統計												
ー般 XAUY-ス(1) MSS MQ スクリプト アクセス権 セキュリティ												
	有効	ID	名前		モジュ-				Open☆	字列	Close 文字列	説 明
編集	*	✓ ESSYBXA SYBASE_XA_SERVER /export/home/yoshihirom/test/sybase/lib64/ESSYBASEXA.so -Nconnection64 - Uxauser -Ppassword										
追加												

5) Enterprise Server を起動

\$	casstart	-rESDEMO64
 C/ C/ \$	ASCD0167I ASCD0050I	ES Daemon successfully auto-started 10:49:51 ES "ESDEMO64" initiation is starting 10:49:51

XA スイッチモジュールが正しく構成され、動作することを Enterprise Server のコ ンソールログより確認できます。下図は Enterprise Server Administration 画面にて、 [編集] ボタン > [診断] タブ > [ES コンソール]

とナビゲートし確認したコンソール画面になります	`:
-------------------------	----

	ス ダンプ ESコンソール CSコンソール O Show entries from 1 to 10 更新 O Show last 10 lines of 35 total entries
Entry	Event Show Entire Log
26	140929 10495197 CASCD10711 Administration SEP created for Server ESDEM064, process-id = 16385 10:49:51
27	140929 10495202 16382 ESDEMO64 CASXO0020I ESSYBXA XA interface loaded. Name(SYBASE_XA_SERVER), Registration Mode(Static) 10:49:52
28	140929 10495205 16385 ESDEM064 CASSI1500I SEP initialization started 10:49:52
29	140929 10495205 16384 ESDEMO64 CASCS50011 Communications interface 01 initialization started 10:49:52
30	140929 10495208 16384 ESDEMO64 CASCS5003I Communications interface 01 initialization complete 10:49:52
31	140929 10495213 16382 ESDEMO64 CASXO0015I ESSYBXA XA interface initialized successfully 10:49:52
32	140929 10495218 16382 ESDEMO64 CASSI5040I Active SEP memory strategy set to x'00000001', retain count 100 10:49:52
33	140929 10495298 16383 ESDEMO64 CASSI1600I SEP initialization completed successfully 10:49:52
34	140929 10495312 16385 ESDEMO64 CASSI1600I SEP initialization completed successfully 10:49:53

6) 付録2の要領で COBOL リモート開発用のデーモンを起動

- Windows クライアントマシン
- 7) Visual COBOL for Eclipse を起動
- 8) 付録2で使用したリモートプロジェクトを開く
- 9) Sybase 上のデータを更新する埋め込み SQL 文の入った COBOL プログラムをプ ロジェクトに追加(付録3の検証と同じプログラムを利用します。)
  - COBOL エクスプローラにてプロジェクトを右クリックし、 [インポート] > [インポート]
     を選択
  - ② [Remote Systems] > [Remote file system] へとナビゲートし、[次へ] ボタン を押下
  - ③ [Browse] ボタンを押下
  - ④ [Connection] 欄で Solaris サーバのアドレスを選択
  - ⑤ 付録1で使用した COBOL リモートプロジェクトのディレクトリヘナビゲ ートし [OK] ボタンを押下
  - ⑥ 付録3で使用した UPP.cbl のみにチェックを入れ、[完了] ボタンを押下

From directory : tok-mitsuwin81D.10.18.11.27	:/export/home/yoshihirom/test/ v Browse
<ul> <li>sybase32</li> <li>New_Configuration.bin</li> <li>repos</li> <li>SelS.deploy</li> <li>UPPS.deploy</li> </ul>	.cobolBuild   .cobolProj   .inventory.si   .project   mccerror.txt   Sel.cbl

10) プログラムが SET CONNECTION 文を発行する先の Logical Resource Manager 名を 3) で構成した名前に変更

UPP.cbl のソースを下記のように変更し保存:

[編集前]

EXEC SQL SET CONNECTION "connection32" END-EXEC

:

[編集後]

EXEC SQL SET CONNECTION "connection<mark>64</mark>" END-EXEC.

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



11) アプリケーションの COBOL – Java 変換マッピングを作成

- COBOL エクスプローラにてプロジェクトを右クリックし [新規] > [その他] を選択
- ② [Micro Focus IMTK] > [Java インターフェイス] へとナビゲートし [次へ] ボタンを押下
- ③ Java インターフェイス名「UPPS64」を指定し、[参照] ボタンを押下
- ④ 追加したプログラムを選択
- ⑤ [完了] ボタンを押下

ここでは、デフォルトのマッピングをそのまま利用します:

🖻 Sel.cbl 🚺 UPP.cbl 🕬	UPPS64 🛛				
LINKAGE SECTION: UPP オペレーション - インターフェイス フィールド:					
名前	PICTURE	名前	方向	型	OCC
LK-STAFF-ID LK-STAFF-NAME LK-Commit-Or-Rolback	S9(9) comp-5 X(10) X	➡ LK_STAFF_ID_io ➡ LK_STAFF_NAME_io ➡ LK_Commit_Or_Rolback_	入出力 入出力 入出力	int String String	

## 12) Enterprise Server へのディプロイ情報を指定

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

▲ 💦 Java インター ▷ 😒 SelS64	דילג
▷ Solution OP ▲ ▷ New_(	新規(N)
Sel X Sel Sel UP UP	削除(D) ディプロイ 検査 開く
repos	プロパティ(P)

- ② [ディプロイメントサーバー] タブを選択
- ③ [変更] ボタンを押下
- ④ Solaris サーバの ESDEMO64 を選択し [OK] ボタンを押下

ディプロイ先の Ente	rprise 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	説明
ESDEMO	Deplover	Available, Stopped	10.18.11.27:39488	BitMode=32-Bit	D
ESDEMO64	Deployer	Available, Started	10.18.11.27:9006	BitMode=64-Bit	D

⑤ [トランザクション管理] 欄にて [コンテナ管理] を選択

Enterprise Server 名: ESDEMO64 (10.1)	8.11.27:9006)	変更
Enterprise Server	実行時環境の使用	
	Enterprise Server 実行時環境の構成	
ナービス名:		
UPP		
- トランザクション管理		
○アプリケーション管理		

- 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
- ③ プロジェクトディレクトリ配下の New\_Configuration.bin ディレクトリに
   生成された UPP.gnt 及び UPP.idy を選択し [OK] ボタンを押下
- [EJB 生成] タブを選択
- ① [アプリケーションサーバー] 欄にて JEE 6、WebLogic 12.1.1 を指定
- ③ [Java コンパイラ] 欄にて利用する javac が格納されたディレクトリを指定
- ④ [J2EE クラスパス] 欄にて [参照] ボタンを押下し WebLogic のインスト
   ールディレクトリに格納されている weblogic.jar を選択
- ⑤ [OK] ボタンを押下し、設定画面を閉じる

13) 作成した Java サービスを Enterprise Server ヘディプロイ

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

▲ 🕞 Java インターフェ ▷ 📽 SelS64	117	
⊿ 🧁 New_Cor	新規(N)	•
🖹 Sel.gr 🗙	削除(D)	
Sel.id	รัสวับส	
🛐 UPP.g	検査	
DPP.g	開く	
	クライアント生成	
repos mccerror	プロパティ(P)	

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:

	ES	ES 編集…	MF ES	1	1 CP 1 Web Services and J2EE tcp:10.18.11.27*:9005 (tok-putter)			Available	ок		
削除	Sel	1 of 1 operation	ations sh	owr	1						
		.SEL [編集]		1	1 CP 1 Web Services and J2EE tcp:10.18.11.27*:9005 (tok-putter)	MFRHBINP	Sel	Available	ок		
削除	UPP	1 of 1 operation	ations sh	owr	1						
		.UPP 編集		1	1 CP 1 Web Services and J2EE tcp:10.18.11.27*:9005 (tok-putter)	MFRHBINP	UPP	Available	ок	J	

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

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

▲ 🔓 Java インター ▷ 😒 SelS64	-7117	3	ק. ק
▷ So Uplices ▲ ▷ New_	新規(N)	•	
🖹 Se 🔀	削除(D)		
Se Se	ディプロイ 検査		
	開く		
	クライアント生成		
repos	プロパティ(P)		_

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:

a 🤔 Sybase64 [10.18.11.27:/export/home/yoshihirom/test
▷ 📠 COBOL プログラム
🔉 🕞 Java インターフェイス
> 🗁 New_Configuration.bin
🔺 🗁 repos
SelS64.deploy
UPPS64.deploy
> 🗁 Client
> 🗁 com
META-INF
🔊 mfejblib.jar
UPPS64.ear
🕌 UPPS64.jar
UPPS64.mfmak
UPPS64.war

■ Solaris サーバ

15) 付録 3 で追加したリソースアダプタが 64bit の Enterprise Server ヘポイントする

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

⑦ eis/MFCobol\_v1.5 をクリック

アウ	バウンド接続ブールの構成表
新規	見前時
	グループとインスタンス 🏟
	□ javax.resource.cci.ConnectionFactory
	eis/MFCobol_v1.5

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

アウトバウンド接続のプロパティ					
保存 表示項目1-8/8 前 ) 2					
プロパティ名 🌣	プロパティのタイプ	プロパティ値	動的な更新のサポート		
EnterpriseServerSocketCloseDueToInactivity	java.lang.Integer	72	false		
LogHost	java.lang.String	localhost	false		
LogPort	java.lang.String	9029	false		
ReadTimeoutInSeconds	java.lang.Integer	0	false		
SecureGUID	java.lang.Boolean	false	false		
ServerHost	java.lang.String	localhost	false		
ServerPort	java.lang.String	9005	false		
Trace	java.lang.Boolean	false	false		
保存			表示項目1-8/8 前 次		

- ⑨ [保存] ボタンを押下
- ⑩ デプロイメント・プランの保存確認画面では [OK] ボタンを押下

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

- WebLogic Server Administration Console の左ペインより対象のドメイン 配下の [デプロイメント] をクリック
- ② [インストール] ボタンを押下
- ③ [パス] 欄に生成された .ear が格納されたディレクトリを指定し Enter を 打鍵
- ④ 生成された UPPS64.ear を選択し、[次へ] ボタンを押下

現在の場所:	tok-putter / export /
💿 🎥 Client (ディレクトリを開く)	
com	
🖲 📑 UPPS64.ear 🗧	
💿 🦲 UPPS64.jar	
💿 🧕 UPPS64.war	
💿 🦲 mfejblib.jar	

- ⑤ [このデプロイメントをアプリケーションとしてインストールする] が選択さ れていることを確認して、[次へ] ボタンを押下
- ⑥ [終了] ボタンを押下

同コンソール画面にて正常にインストールされたことを確認できます:

_	7747824						
	インストール     更新     削除     「停止 ▼     表示項目1 - 5/5     前   次						
		名前 🕎	状態	ヘルス	タイプ	ターゲット	デプロイ順序
		藧 mfcobol-xa	<i>ア</i> クティ ブ	🖋 ок	リソース・アダプタ	AdminServer	100
		🗈 🔽 SelS	管理		エンタープライズ・アプリ ケーション	AdminServer	100
		🗈 🔽 SelS64	<i>ア</i> クティ ブ	🖋 ок	エンタープライズ・アプリ ケーション	AdminServer	100
		E DUPPS	<i>ア</i> クティ ブ	🖋 ок	エンタープライズ・アプリ ケーション	AdminServer	100
		E CUPPS64	アクティ ブ	🖋 ок	エンタープライズ・アプリ ケーション	AdminServer	100

■ Windows クライアントマシン

**17**) Enterprise Server が動的デバッグ有効で起動されていることを確認 Enterprise Server Administration 画面を開き、ESDEMO64 の行における [編集]

ボタンを押下し、確認

- 18) Enterprise Server デバッグを起動
  - COBOL エクスプローラにてプロジェクトを右クリックから [デバッグ] > [デバッグの構成] を選択
  - ② 付録2で作成したデバッグ構成をダブルクリック

COBOL Enterprise Server     Server     Set ESDEMO64	▼ COBOL プロジェクト(P) Subase64	参昭
Im COBOL JVM アプリケーション	Sybuseon	≥26.00
IM COBOL JVM リモート アプリケー:	✓ Enterprise Server	
■ COBOL アプリケーション	接続: tok-putter サーバー: ESDEMO64	参照
🛐 COBOL コアダンプ	- ニバッグの種類	
🛜 COBOL のアプリケーションのアタッ	◆ 7八ックの裡類	
📆 COBOL のプロセスにアタッチ	Web サービス Java	)
😝 Eclipse Data Tools	Java サービス名(空白の場合はすべてのサービスをデバッグ)	
Eclipse アプリケーション		
🗄 Generic Server		
A compare company/submedule		

③ [パースペクティブの切り替えの確認] 画面にて [はい] を選択

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

参デバッグ 🛛 斗 Servers	
▲ 😹 ESDEMO64 [COBOL Enterprise Server]	

19) ディプロイしたアプリケーションをデバッグ実行しデータを更新

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

← → Ø http://tok-	putter:7001/UPPS64/UPP.jsp	×5 - Q	Test Client for UPPS64	×
File Edit View Favor	iter Tools Help			
Test client fo	or UPPS64.UPP			
Back				
Perform the test by	entering values:			
UPP_LK_STAFF_I	<b>D_io</b> : 0			
UPP_LK_STAFF_N	NAME_io :			
UPP_LK_Commit	Or_Rolback_io :			
			Go!	
ブ <u>Back</u> を	ラウザを立ち上げ、 tp:// <solaris サーバの<br="">アドレスバーに入力</solaris>	アドレス>	: <weblogic th="" のポー<=""><th>ト&gt;/UPPS</th></weblogic>	ト>/UPPS

② [UPP\_LK\_STAFF\_ID\_io] 欄に [20] を
 [UPP\_LK\_STAFF\_NAME\_io] 欄に [Atsushi] を
 [UPP\_LK\_Commit\_Or\_Rollback\_io] 欄に [C] を
 入力し [Go!] ボタンを押下
 Perform the test by entering values:

UPP_LK_STAFF_ID_io :	20
UPP_LK_STAFF_NAME_io :	Atsushi
UPP_LK_Commit_Or_Rolback_io :	С
	Go!

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

🏇 デバッグ 🛙 斗 🕾 Servers 🦓 🛋 🕾 🎄 🍸 🖓	()= 変数 💥	●◎ ブレークポイント	🗄 🏘 🖻 🗸 🗖
<ul> <li>              ESDEMO64 [COBOL Enterprise Server]</li></ul>	名前 《	섙	>
UPP.cbl     ∴         UPP.cbl         UPP.cbl         UVP.cbl         UVV         UVV         UVV	•	Pプトライン      SYB-POINTER-USAGE PIC     SYB-INDEX-USAGE PIC S     SYB-MONEY-USAGE PIC S     SYB-MONEY-USAGE PIC S     SYB-MONEY-USAGE PIC S     SYB-DATETIME-USAGE PIC     SYB-DATETIME-USAGE PIC S9(     SYB-DATETIME-USAGE PIC S9(     SYB-NO-PIC PIC S9(9)     SYB-NINES-PIC PIC S9(9)     SYB-SNINES-PIC	Image         Image           59(9) COMP, th/X:4            (9) COMP, th/X:4            9(9) COMP, th/X:4            59(9) COMP, th/X:4            0(9) COMP, th/X:4
	>	SYB-X-PIC PIC S9(9) COM     <	P, サイズ:4 ×

③ デバッグ実行

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

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



デバッグ実行イメージ:

● UPP.cbl       □       □       □       □       □       ● Procedure Division         ● UPP.cbl       □       □       □       ● Procedure Division       ● 1         ● WWE LK-STAFF-ID       □       ○ STAFF-NAME       ● SQLWORST       ● SQLWORST         ● MOVE LK-STAFF-ID       □       ○ STAFF-NAME       ● SQLWORST       ● SQLWORST         ● WHERE ID=:STAFF-ID       EXD-EXEC.       □       □       ■ SQLWORST         ● DISPLAY "UPDATE = "SQLCODE UPON CONS       EXIT PROGRAM       ■       □       □         END-EXEC.       □       IF SQLCODE NOT = 0       □       □       □       □       □         IF SQLCODE NOT = 0       ISPLAY "WSG: SQLERRMC UPON CONS       EXIT PROGRAM       ■       □					
Image: UPP.cbl       ● Procedure Division         Image: Weblic Staff - Name       ● 1         Image: We		UPP.cbl 🛛	- 8	רוליק 🗄	
Constant 1-82		UPP.cbl		A S Proce	dure Divisi
WOVE LK-SIAFF-IND IO SIAFF-INE.     SQL-WORST     SQL-WORST     SQL-WORST     SQL-PROLO     SQ		••••••*A•1•B•••••2•••••3•••••4•••••	j•••••7•• I•••	U 1	
MOVE LK-STAFF-NAME TO STAFF-NAME.     EXEC SQL UPDATE STAFF SET NAME::STAFF-NAME     WHERE ID::STAFF SET NAME::STAFF-NAME     WHERE ID::STAFF-INAME: STAFF-NAME     WHERE ID::STAFF-INAME: STAFF-NAME     UEPILOC     L-RESUL'     L-RESUL'     L-RESUL'     L-RESUL'     L-LAST     END-IF     IF LK-Commit-Or-RoIback = 'R'     SET IDX TO 11     MOVE SPACE TO TABLE-ITEM(IDX)     END-IF.     IF LK-Commit-Or-RoIback = 'R'     SET IDX TO 11     MOVE SPACE TO TABLE-ITEM(IDX)     END-IF.     IF LSDEMO64] [/IIIIZX ID: 16383]     III = STAFF-NAME - Atsushi     III = SDEMO64] [/IIIIZX ID: 16383]		MOVE LK-STAFF-ID TO STAFF-ID.	•		QLWORS1
EXEC SUL UPDATE STAFF SET NAME::STAFF-NAME   WHERE ID::STAFF-ID   WHERE ID::STAFF-ID   END-EXEC.   DISPLAY "UPDATE : "SOLCODE UPON CO   IF SOLCODE NOT := 0   DISPLAY "MSG:" SOLERRMC UPON CONS   EXIT PROGRAM   END-IF   IF LK-Commit-Or-Rolback : 'R'   SET IDX TO 11   MOVE SPACE TO TABLE-ITEM (IDX)   END-IF.   Atsushi   CCONNC   IF LK-Commit-Or-Rolback : 'R' SET IDX TO 11 MOVE SPACE TO TABLE-ITEM (IDX) END-IF. Atsushi I-CONNC I-M S  4720 F/Kvd xyt2-5 IDX-L: [ESDEMO64] [/IItZ ID: 16383] F/Kvd xyt2-5 IDX-L: [ESDEMO64] [/IItZ ID: 16383]		MOVE LK-STAFF-NAME TO STAFF-NAME.		SC 0	LPROLO
WHERE ID=:STAFF-ID       U=CFICOC         END-EXEC.       DISPLAY "UPDATE = "SQLCODE UPON CO         IF SQLCODE NOT = 0       DISPLAY "MSG:" SQLERRMC UPON CONS         EXIT PROGRAM       U=CRESUL         END-IF.       IF LK-Commit-Or-Rolback = 'R'         SET IDX TO 11       MOVE SPACE TO TABLE-ITEM(IDX)         END-IF.       Atsushi         IF LK-Commit-Or-Rolback = 'R'         SET IDX TO 11         MOVE SPACE TO TABLE-ITEM(IDX)         END-IF.         IF.         IF.         MOVE SPACE TO TABLE-ITEM(IDX)         END-IF.         IF.         IF. <td< td=""><td></td><td>EXEC SQL UPDATE STAFF SET NAME=:STAF</td><td>-F-NAME</td><td></td><td></td></td<>		EXEC SQL UPDATE STAFF SET NAME=:STAF	-F-NAME		
Constraints of the second secon		WHERE ID=:STAFF-ID		-	L-LFILOC
IF SQLCODE NOT = 0 DISPLAY "MSG:" SQLERRMC UPON CONS EXIT PROGRAM END-IF IF LK-Commit-Or-Rolback = 'R' SET IDX TO 11 MOVE SPACE TO TABLE-ITEM(IDX) END-IF.		ENU-EXEC.		•	LRESUL
LLAST LLAST LLAST LLAST LCTXERI LCSBDI/	17	IE SOLCODE NOT - 0	CTAFE NAME Atouchi		LPROTE
EXIT PROGRAM EXIT PROGRAM EXIT PROGRAM END-IF IF LK-Commit-Or-Rolback = 'R' SET IDX TO 11 MOVE SPACE TO TABLE-ITEM(IDX) END-IF.  Atsushi 16道: 4777786222 1435389000		DISPLAY "MSG:" SQLERRMC UPON CONS	C STAFF-NAME = ALSUSHI		LLAST
END-IF IF LK-Commit-Or-Rolback = 'R' SET IDX TO 11 MOVE SPACE TO TABLE-ITEM(IDX) END-IF. 4tsushi 16道: 4777766222 1435389000 マレーンションソール 窓 タスク デバッグ メッセージ コンソール: [ESDEMO64] [プロセス ID: 16383]		EXIT PROGRAM			L CTYER
IF LK-Commit-Or-Rolback = 'R' SET IDX TO 11 MOVE SPACE TO TABLE-ITEM(IDX) END-IF.		END-IF			LCTALK
IF LK-Commit-Or-Rolback = 'R' SET IDX TO 11 MOVE SPACE TO TABLE-ITEM(IDX) END-IF.					LCSBDI/
SET IDX TO 11 MOVE SPACE TO TABLE-ITEM(IDX) END-IF.         Atsushi 16道: 4777766222 1435389000         LDESCO LCONNC LCONNC LCMDOF           マンソール 窓 タスク デバッグ メッセージ コンソール: [ESDEMO64] [プロセス ID: 16383]         マーレンシール: 1000000000000000000000000000000000000		IF LK-Commit-Or-Rolback = 'R'			LSETIN1
MOVE SPACE TO TABLE-ITEM(IDX) END-IF. 41505000 41 (プロセス ID: 16383) ポピューにきたの		SET IDX TO 11			IDESCO
END-IF・     16進: 4777766222 1435389000     LCONNC LCMDOF       マンソール 窓 塗 タスク     マンソール : [ESDEM064] [プロセス ID: 16383]     マビューにきか		MOVE SPACE TO TABLE-ITEM(IDX)	Atsushi		CONNIC
4777766222     1435389000      マンソール 窓 塗 タスク      デバッグ メッセージ コンソール: [ESDEMO64] [プロセス ID: 16383]      マジュンソール: [ESDEMO64] [プロセス ID: 16383]      マジュンソール: [ESDEMO64] [プロセス ID: 16383]      マジュンソール: [ESDEMO64] [プロセス ID: 16383]      マジョンソール: [ESDEMO64] [プロセス ID: 16383]		END-IF.	16進:	· · · · · · · · · · · · · · · · · · ·	LCONNC
			4777766222		LCMDOF
		`	1 435389000		
■ コンソール ※		(		~	
デパッグ メッセージ コンソール: [ESDEMO64] [プロセス ID: 16383]		コンソール 🛛 💆 タスク	<	>	
	デバ	ッグ メッセージ コンソール : [ESDEM064] [プロセス ID: 16383]		-But - ball	
	100	aca: arnasyne not comprise for acouseing		式ビューに追加	

Loaded: User Module]/opt/mf/ED22U1HF4/deploy/UPPS64.0vLJGqe-/UPP.gnt- Symbols loaded from: /opt/mf/ED22U

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

## **Test client for UPPS64.UPP**

## Back

Perform the test by entering values:

UPP_LK_STAFF_ID_io :	20
UPP_LK_STAFF_NAME_io :	Atsushi
UPP_LK_Commit_Or_Rolback_io :	С
	Go!

## **Result:**

Variable	Value
LK_STAFF_ID_io	20
LK_STAFF_NAME_io	Atsushi
LK_Commit_Or_Rolback_io	с

Back

■ Solaris サーバ

20) アプリケーションがレコードを正しく更新していることを確認

> sele > go ID	CT * Trom SIAFF	DEPT	JOB	YEARS	SALARY
	 10 Hogan	333	Mgr	7	18357.50
	20 Atsushi	333	Sales	8	18171.25
	30 Marenhi 0.00	38	Mgr	5	17506. 75
(3 rows >	affected)				

■ Windows クライアントマシン

- 21) トランザクションが取り消される条件でパラメータを渡し、ディプロイしたアプリケ ーションをデバッグ実行
  - ④ WebLogic にディプロイしたスタブクライアントアプリケーションを起動
  - ⑤ [UPP\_LK\_STAFF\_ID\_io] 欄に [20] を
     [UPP\_LK\_STAFF\_NAME\_io] 欄に [Tatsumi] を
     [UPP\_LK\_Commit\_Or\_Rollback\_io] 欄に [R] を
     入力し [Go!] ボタンを押下

Perform the test by entering values:

UPP_LK_STAFF_ID_io :	20
UPP_LK_STAFF_NAME_io:	Tatsumi
UPP_LK_Commit_Or_Rolback_io :	R
	Go!

⑥ デバッグ実行

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

🖻 UPP.cbl 🛛		- 0	E 701-542 X
B LIDD			• IDX
			🕟 😤 Linkage
	KA-1-B2	• 1 • • •	A S Procedure
	END-IF	^	
	EXEC SQL USE pubs2 END-EXEC.		
	DISPLAY "USE SQLCODE:" SQLCODE UPON CONSOLE.		SQLW
	IF SQLCODE NOT = 0		SQLP
	DISPLAY MSG: SQLERRMC UPON CONSULE		SQLE
	END-TE		SQLR
	MOVE LK-STAFF-ID TO STAFF-ID.		SOI P
	MOVE LK-STAEF-MAME_TO_STAFE-NAME.		
	EXEC SQL UPDATE STAFF SET NAME=:STAFF-NAME		0 JQLL
	WHERE ID=:STAFF-ID		SQLC
	DISPLAY "UPDATE = " SOLCODE UPON CONSOLE.		● SQLC
	IF SQLCODE NOT = 0		
	NTSPLAY "NSG-" SOLEPPH Working-Storage Section, PIC S9(18) C	омь ' ≥	照= 6, サイス= 8 ハイト
UPDATE 文実行直	SQLCODE=+000000000000000000000000000000000000		
後の SOLCODE			
12 V SQLOUL			

ステップをそのまま進めていきますと第3パラメータに「R」を指定したため、 10 回の繰り返し項目の添え字に 11 を格納するロジックへ入ります: EXEC SQL INCLUDE SQLCA END-EXEC. 01 TABLE-ITEM PIC X OCCURS 10 TIMES INDEXED BY IDX. LINKAGE SECTION. PROCEDURE DIVISION USING LK-STAFF-ID LK-STAFF-NAME LK-Commit-Or-Rolback. 1. DISPLAY "Update TEST" UPON CONSOLE. EXEC SQL SET CONNECTION "connection84" END-EXEC. DISPLAY "CONNECT SQLCODE:" SQLCODE UPON CONSOLE. IF SQLCODE NOT = 0 DISPLAY "MSG:" : EXIT PROGRAM SQLERRMC UPON CONSOLE END-IF END-IF EXEC SQL USE pubs2 END-EXEC. DISPLAY "USE SQLCODE:" SQLCODE UPON CONSOLE. IF SQLCODE NOT = 0 DISPLAY "MSG:" SQLERRMC UPON CONSOLE EXIT PROGRAM END-IF NOVE LK-STAFF-ID TO STAFF-ID. Nove LK-Staff-Name to Staff-Name. Exec SQL update Staff Set Name=:Staff-Name WHERE ID=:STAFF-ID END-EXEC. DISPLAY "UPDATE = " SQLCODE UPON CONSOLE. IF SQLCODE NOT = 0 DISPLAY "MSG:" : SQLERRMC UPON CONSOLE EXIT PROGRAM END-IF I IF LK-Commit-Or-Rolback = 'R' SET IDX TO 11 MOVE SPACE TO TABLE-ITEM(IDX) END-IF. \_ \_ J

繰り返しの範囲を超えた添え字を指定して MOVE 文を実行すると実行時エ



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

## Error 500--Internal Server Error



■ Solaris サーバ

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

\$ isql	-Usa -Psapass -S	STOKPUTTER				
	pubsz					
∠> go L> aala						
I> sele	ect * Trom STAFF					
:> go ⊐⊓				VEADO		
ID		DEPT	JUD	TEARS	SALARI	
	10 Hogan	333	Mør	7	18357 50	
	0.00				10007100	
	20 Atsushi	333	Sales	8	18171, 25	
	612.45					
	30 Marenhi	38	Mgr	5	17506.75	
	0.00					
	20) で確認した	「Atsushi」がそ				
(3 rows	のまま残ってい	すす				
>		6 7 o	J			

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

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



3) 検証で利用するテーブルを作成

SQL	文 文								
1	CREATE TABLE STAFF(ID INT NOT NULL,								
2	NAME VARCHAR(10),								
3	DEPT INT,								
4	JOB VARCHAR(10),								
5	YEARS DECIMAL(4),								
6	SALARY DECIMAL(10,2),								
7	COMM DECIMAL(10,2))								
°	4								
結果									
実行問	実行時間: 0.625 秒								

4) レコードを3件追加

SQL 3	文 文				
1	INSERT	INTO	STAFF	VALUES(10,	'Smith', 333, 'Mgr', 7, 18357.50, 0)
2	INSERT	INTO	STAFF	VALUES(20,	'Takeshi', 333, 'Sales', 8, 18171.25, 612.45)
3	INSERT	INTO	STAFF	VALUES(30,	'Marenhi', 38, 'Mgr', 5, 17506.75, 0)
	4				
▲▼					
結果					
10-1	が挿入さ	れまし	ž.		
1 0-1	が挿入さ	れまし	ż.		
10-1	が挿入さ	れまし	ž.		
実行時	寺間:0.4	84 秒			

5) 用意したデータを確認

SQL	SQL 文									
1 SELECT * FROM STAFF										
結果										
	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM			
1	10	Smith	333	Mer	7	18,357.50	0.00			
2	20	Takeshi	333	Sales	8	18,171.25	612.45			
3	30	Marenhi	38	Mgr	5	17,506.75	0.00			

以上