# 配列を含む IMS トランザクション用の J2C アプリケーションの作成

# 目次

# 配列を含む IMS トランザクション用の J2C アプリケーションの作成

このチュートリアルでは、「J2C Java™ Bean」ウィザードを使用して、配列が収められた入出力データを持つ IMS™ トランザクションの処理を行う、シンプルな Web アプリケーションを作成する方法を説明します。

## 学習目標

このチュートリアルでは以下のことを説明します。

- 「J2C Bean」ウィザードを使用して、IMS トランザクションを実行する J2C Java Bean を作成する。
- その Bean 用の Java メソッドを作成し、IMS トランザクションを実行する。
- アプリケーションをテストするためのテスト用プロキシー Java クラスを作成する。

## 所要時間

30 分

**関連情報**

PDF バージョンを表示する

入出力配列のサンプル

# 配列を含む IMS トランザクション用の J2C アプリケーションの作成: はじめに

このチュートリアルでは、「J2C Java Bean」ウィザードを使用して、配列が収められた入出力データを持つ IMS トランザクションの処理を行う、シンプルな Web アプリケーションを作成する方法を説明します。

このチュートリアルには、オプションでインストール可能なコンポーネントが必要な場合があります。システム要件を参照して、適切なオプション・コンポーネントがインストールされていることを確認してください。

このチュートリアルは幾つかの練習に分かれています。チュートリアルを正しく機能させるためには、練習を順序どおりに実行する必要があります。このチュートリアルでは、「J2C Java Bean」ウィザードを使用して、IMS でトランザクションを実行する Java Bean を作成する方法を学習します。練習では、次の作業を行います。

- 「J2C Bean」ウィザードを使用して、IMS トランザクションを実行する J2C Java Bean を作成する。
- その Bean 用の Java メソッド `runInOut.java` を作成し、IMS トランザクションを実行する。
- テスト・プロキシー Java クラス `TestInOutProxy.class` を作成し、IMS トランザクションの入力メッセージのビルド、IMSトランザクションを実行する J2C Java Bean メソッドの起動、さらに IMS トランザクションから戻される出力データの表示を行う。

  **注:** `TestInOutProxy.java` Java クラスは、英語地域用に作成されているため、その他の地域では、コードを変更する必要がある場合があります。

### 所要時間

このチュートリアルを終了するには、約 30 分かかります。このチュートリアルに関連した他の概念を検討する場合、完了するのにさらに長い時間がかかります。

### スキル・レベル

熟練

### 対象読者

このチュートリアルは、特にエンタープライズ情報システム (EIS) および IMS に詳しいユーザーを対象にしています。

### システム要件

このチュートリアルを完了するには、以下のツールとコンポーネントがインストール済みである必要があります。

- IBM® WebSphere® Application Server バージョン 6.1
- J2EE コネクター (J2C) ツール
- **IMS 環境**に関する情報: このチュートリアルでは、あなたのアプリケーションは、IMS のアプリケーション・プログラムと対話します。 IMS 接続のホスト名およびポート番号や、トランザクションを実行する IMS データ・ストアの名前などの情報を取得する必要があります。この情報については、IMS システム管理者に連絡してください。また、このサンプルを実行する場合は、IMS でいくつかのセットアップ作業を実行する必要があります。
- **COBOL ファイル `InEqualsOut.cbl`** のコピー: このファイルは、製品のインストール・ディレクトリー: <installdir>\IBM\SDP70Shared\plugins\com.ibm.j2c.cheatsheet.content_7.0.0\samples\IMS\InOutArray にあります。このファイルをローカルに保管する場合は、../resources/inequalsout.dita からコードをコピーすることができます。
- **クリーンなワークスペース**

このチュートリアルを使用するには、アプリケーション・サーバーのインストールと構成が行なわれていることが必要です。サーバー・ランタイム環境が使用可能であるかどうかを検証するには、「**ウィンドウ**」→「**設定**」をクリックして、「**サーバー**」を展開し、「**インストール済みランタイム**」をクリックします。このペインを使用して、インストール済みサーバー・ランタイムの定義を追加、除去、または編集することができます。また、新規サーバーのサポートをダウンロードしてインストールすることもできます。

### 前提条件

このチュートリアルの内容すべてを実行するためには、次の知識が必要です。

- J2EE および Java プログラミング
- 基本 IMS Transaction Manager (IMS TM) の概念

## 演習 1.1: リソース・アダプターの選択

この演習では、リソース・アダプターを選択して、IMS サーバーに接続するように構成するための、詳細なステップについて説明します。

このチュートリアルでは、COBOL データ構造を使用して、IMS トランザクションの入力メッセージおよび出力メッセージについて説明します。入力メッセージと出力メッセージは同一であり、顧客情報要素の配

列を含み、その後に、機能コードが収められた単一のフィールドが続きます。配列には最大 8 つの要素を含めることができますが、このチュートリアルでは、 IMS アプリケーション・プログラムへの入力は 3 つの要素だけであり、 IMS アプリケーション・プログラムから戻される要素は 4 つだけです。

このチュートリアルで使用する IMS トランザクションは、IMS インストール検査プログラムではありません。制御ステートメントの情報に基づいて IMS への呼び出しを発行する IMS アプリケーション・プログラムである、DFSDDLT0 が使用されます。このチュートリアル用の DFSDDLT0 制御ステートメントを、以下に示します。ただし、このチュートリアルを実行する場合には、ご使用の環境を DFSDDLT0 に合わせて構成し、また、必要な JCL を提供する必要があります。このチュートリアルでは、SKS2 を DFSDDLT0 アプリケーションのトランザクション・コードとして使用します。

**DFSDDLT0 制御ステートメント**

```
S11 1 1 1 1    TP     1
L        GU
E      OK
E  Z0088 DATA  SKS2 03CN001Cathy Tang          CN002Haley Fung         X
                CN003Steve Kuo           123456
WTO IC4JINOU: Single segment received from JITOC
L        GN
E      QD
WTO IC4JINOU: End of input segments from JITOC
L        ISRT  JITOC53
L  Z0113 DATA  TRNCD04CN001Cathy T.            CN002Haley F.           X
                CN003Steve K.            CN004Kevin F.            65432X
               1
E      OK
WTO IC4JINOU: Single segment inserted - 3 elements !!!!!!!!!!!!!
L        GU
```

**IMS サーバーへの接続**

1. J2EE アイコン (  ) がワークスペースの右上部タブに表示されない場合は、J2EE パースペクティブに切り替える必要があります。メニュー・バーから、「**ウィンドウ**」>「**パースペクティブを開く**」>「**その他**」と選択します。「パースペクティブの選択」ウィンドウが開きます。
2. 「**J2EE**」を選択します。
3. 「**OK**」をクリックします。J2EE パースペクティブが開きます。
4. J2EE パースペクティブで、「**ファイル**」>「**新規**」>「**その他**」と選択します。
5. 「新規」ページで、「**J2C**」>「**J2C Java Bean**」と選択します。「**次へ**」をクリックします。
6. 「リソース・アダプターの選択」ページで、J2C 1.5 IMS リソース・アダプターを選択します。このチュートリアルでは、「**IMS Connector for Java (IBM: 9.1.0.2.3)**」を選択します。「**次へ**」をクリックします。
7. 「接続プロパティー」ページで、「**管理接続**」を選択解除し、「**非管理接続**」を選択します。 (このチュートリアルでは、非管理対象の接続を使用して、IMS に直接アクセスします。) デフォルトの接続クラス名 `com.ibm.connector2.ims.ico.IMSManagedConnectionFactory` を受け入れます。ブランク・フィールドに、必要なすべての接続情報を入力してください。必須フィールドは以下のとおりです。これらはアスタリスク (*) で表されています。
   a. **TCP/IP 接続の場合**:
      1) **ホスト名**: (必須) IMS 接続の IP アドレスまたはホスト名。
      2) **ポート番号**: (必須) ターゲット IMS 接続で使用するポートの番号。
   b. **ローカル・オプション接続の場合**:
      1) **IMS Connect 名**: (必須) ターゲット IMS 接続の名前。

c. **両方の接続用**:

1) **Data Store 名**: (必須) ターゲット IMS データ・ストアの名前。

8. これらの接続情報は、担当の IMS システム管理者から得ることができます。必要な接続情報を指定したら、「**次へ**」をクリックしてください。

## 演習 1.2: Web プロジェクトおよび Java インターフェースと実装のセットアップ

演習 1.2 では、Web プロジェクトおよび Java インターフェースと実装の作成を段階的に学習します。

開始する前に、演習 1.1 を完了しておく必要があります。この演習では、次のことを行います。

- J2C Java Bean を作成する
- 動的 Web プロジェクトを作成する

1. ワークベンチを使用して行うすべての作業は、プロジェクトに関連付ける必要があります。プロジェクトでは、プロジェクトのタイプに基づいた機能により最適化された、作業ファイルとディレクトリーの系統的なビューが提供されます。ワークベンチでは、すべてのファイルがプロジェクト内になければならないため、J2C Java Bean を作成する前に、ファイルを入れるプロジェクトを作成しておく必要があります。「新規 J2C Java Bean」ページで、「**プロジェクト名**」フィールドに、値 `InOutArray` を入力します。
2. 「**プロジェクト名**」フィールドの横にある「**新規**」ボタンをクリックします。
3. 次に、動的 Web プロジェクトを作成します。「新規ソース・プロジェクトの作成」ページで、「**Web プロジェクト**」を選択し、「**次へ**」をクリックします。
4. 「新規動的 Web プロジェクト」ページで、次の値が選択されていることを確認します。
   a. **プロジェクト名**: InOutArray
   b. **プロジェクト・コンテンツ**: デフォルトの使用
   c. **ターゲット・ランタイム**: WebSphere Application Server v6.1
   d. **構成**: デフォルトを受け入れる
   e. **EAR にプロジェクトを追加**: チェック
   f. **EAR プロジェクト名**: InOutArrayEAR
5. 「**次へ**」をクリックします。
6. 「プロジェクト・ファセット」ページで、デフォルト設定を受け入れます。
7. 「**次へ**」をクリックします。
8. 「Web モジュール」ページで、デフォルト設定を受け入れます。
9. 「**終了**」をクリックします。
10. 動的 Web パースペクティブに切り替えるかどうかを尋ねるダイアログ・ボックスが表示されます。「**はい**」をクリックします。
11. 「J2C Java Bean 出力プロパティー」ページで、以下を実行します。
    a. 「**プロジェクト名**」フィールドで、「**参照**」をクリックして「`InOutArray`」プロジェクトを選択します。「**OK**」をクリックします。
    b. 「**パッケージ名**」フィールドに `sample.ims` と入力します。
    c. 「**インターフェース名**」フィールドに `InOut` と入力します。
    d. 「**実装名**」フィールドに `InOutImpl` と入力します。
    e. 「**Ant スクリプトとしてセッションを保管**」を未チェックのままにします。

12. 「**終了**」をクリックします。

## 演習 1.3: Java メソッドの作成

演習 1.3 では、Java メソッドの作成を学習します。

開始する前に、演習 1.2 を完了しておく必要があります。この演習では、次のことを行います。

- Java メソッドを作成する
- COBOL と Java の間の入出力データ・マッピングを作成する

1. **最初に Java メソッドを作成します。** 「プロジェクト・エクスプローラー」ビューで、「動的 Web プロジェクト」の **InOutArray** プロジェクトを展開します。
2. 「Java ソース」にある **InOutImpl.java** を右クリックし、「**ソース> J2C Java Bean へのメソッドの追加**」を選択します。
3. 「Java メソッド」ページで、「**追加**」をクリックします。
4. 「**Java メソッド名**」フィールドに、このメソッドの名前として `runInOut` と入力します。
5. 「**次へ**」をクリックします。
6. **COBOL と Java の間の入力データ・マッピングを作成します。** 「Java メソッド」ページの「**入力タイプ**」フィールドの横にある「**新規**」をクリックします。
7. 「データ・インポート」ページで、「**マッピングの選択**」フィールドが **COBOL から Java** になっていることを確認します。「**COBOL ファイル**」フィールドの横にある「**参照**」をクリックします。
8. 参照して InEqualsOut.cbl ファイルのロケーションを探します (このファイルは、製品のインストール・フォルダー内の <installdir>¥IBM¥SDP70Shared¥plugins¥com.ibm.j2c.cheatsheet.content_7.0.0¥Samples¥IMS¥InOutArray にあります)。
9. 「**開く**」をクリックします。
10. 「**次へ**」をクリックします。
11. 「COBOL インポーター」ページで、「**詳細を表示**」をクリックします。
    a. 次のオプションを選択してください。

*表 1. COBOL インポーターのパラメーター設定値*

| パラメーター | 値 |
|---|---|
| プラットフォーム | z/OS |
| コード・ページ | IBM-037 |
| 浮動小数点形式 | IBM 16 進数 |
| 外部 10 進符号 | EBCDIC |
| エンディアン名 | Big |
| リモート整数エンディアン名 | Big |
| 引用符名 | DOUBLE |
| TRUNC 名 | STD |
| Nsymbol 名 | DBCS |

    b. 「**照会**」をクリックして、データをロードします。
    c. データ構造のリストが表示されます。「**データ構造**」フィールドで **IN-OUT-MSG** を選択します。
    d. 「**次へ**」をクリックします。

12. 「保管するプロパティー」ページで、次の手順を実行します。
    a. 「**スタイルの生成**」で「**デフォルト**」を選択します。
    b. 「**プロジェクト名**」の横にある「**参照**」をクリックして、Web プロジェクト **InOutArray** を選択します。
    c. 「**パッケージ名**」フィールドに、sample.ims.data と入力します。
    d. 「**クラス名**」フィールドで、デフォルトの **INOUTMSG** を受け入れます。「**終了**」をクリックします。
    e. 「**Ant スクリプトとしてセッションを保管**」を未チェックのままにします。
13. 「Java メソッド」ページで、「**出力に入力タイプを使用**」を選択します。
14. 「**終了**」をクリックします。
15. 「**終了**」をクリックして、メソッドの定義を終了します。

## 演習 1.4: アプリケーションをテストするための Java テスト・クラスの作成

演習 1.4 では、アプリケーションのテストを行うための Java テスト・クラスの作成を学習します。

開始する前に、演習 1.3 を完了しておく必要があります。この演習では、次のことを行います。

- Java テスト・クラスを作成する。
- 以下に示したコードを使用してクラスを編集する。
- テスト・クラスを実行してアプリケーションをテストする。

1. **最初に Java テスト・クラスを作成します。 InOutArray** プロジェクトを展開し、「**Java リソース**」を展開してから、**sample.ims** パッケージを選択します。
2. 右マウス・ボタンをクリックして、「**新規**」を選択します。 クラス・オプションを選択して、新規 Java クラスを作成します。
3. 「Java クラス名」に、TestInOutProxy と入力します。 TestInOutProxy クラスは、例示用としてのみ提供されている点に注意してください。ご使用の IMS マシンの仕様に合わせて、トランザクション・コードを変更する必要がある場合があります。トランザクション・コードについては、IMS 管理者にお尋ねください。コード内にあるステートメント `input.setWs__trcd("SKS7");` を見つけて、変更を行うことができます。
4. 「**ソース・フォルダー**」フィールドに **InOutArray/JavaSource** が含まれ、「**パッケージ名**」フィールドに **sample.ims.data** が含まれていることを確認します。
5. 「**終了**」をクリックします。
6. 「**TestInOutProxy**」をダブルクリックして、Java エディターでファイルを開きます。
7. 以下に示したコードをすべてコピーし、**TestInOutProxy.java** クラスに貼り付けます。エディターで既存コードを置換してください。

   **注:** TestInOutProxy.java Java クラスは、英語地域用に作成されているため、その他の地域では、コードを変更する必要がある場合があります。

```
/*
 * Created on 4-Oct-2004
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package sample.ims;
```

```
import sample.ims.data.*;
import com.ibm.connector2.ims.ico.IMSDFSMessageException;

/**
 * @author ivyho
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class TestInOutProxy
{
 public static void main (String[] args)
 {
  try
  {
   // ----------------------------------------------------
   // Create the formatHandler, then create the input
   // message bean from the formatHandler.
   // ----------------------------------------------------
   INOUTMSG input = new INOUTMSG();

   int sz = input.getSize();
   System.out.println("¥nInitial size of input message is: " + sz);

   // ----------------------------------------------------
   // Don't set the length (LL) field yet... wait until
   // input message has been adjusted to reflect only
   // the number of array elements actually sent.
   // ----------------------------------------------------
   input.setWs__zz((short) 0);
   input.setWs__trcd("SKS7 ");

   // ----------------------------------------------------
   // Construct an array and populate it with the elements
   // to be sent to the IMS application program.  In this
   // case three elements are sent.
   // ----------------------------------------------------
   Inoutmsg_ws__customer[] customers = new Inoutmsg_ws__customer[3];

   Inoutmsg_ws__customer aCustomer1 = new Inoutmsg_ws__customer();
   aCustomer1.setWs__cust__name("Cathy Tang");
   aCustomer1.setWs__cust__number("CN001");
   customers[0] = aCustomer1;

   Inoutmsg_ws__customer aCustomer2 = new Inoutmsg_ws__customer();
   aCustomer2.setWs__cust__name("Haley Fung");
   aCustomer2.setWs__cust__number("CN002");
   customers[1] = aCustomer2;

   Inoutmsg_ws__customer aCustomer3 = new Inoutmsg_ws__customer();
   aCustomer3.setWs__cust__name("Steve Kuo");
   aCustomer3.setWs__cust__number("CN003");
   customers[2] = aCustomer3;

   // ----------------------------------------------------
   // Set the array on the input message.
   // ----------------------------------------------------
   input.setWs__customer(customers);
   input.setIndx((short) 3);

   System.out.println("¥nInitial value of INDX is: " + input.getIndx());

   // ----------------------------------------------------
   // Reallocate the buffer to the actual size
   // ----------------------------------------------------
   byte[] bytes = input.getBytes();
   int size = input.getSize();
```

```
byte[] newBytes = new byte[size];
System.arraycopy(bytes, 0, newBytes, 0, size);

// ------------------------------------------------
// Set the bytes back into the format handler and set
// the length field of the input message, now that
// we know the actual size.
// ------------------------------------------------
input.setBytes(newBytes);
input.setWs__ll((short) size);
System.out.println("¥nAdjusted size of input message is: " + size);
System.out.println("¥nAdjusted size of INDX is: " + input.getIndx());

// ------------------------------------------------
// Set fields that follow the array after the input
// message has been adjusted.
// ------------------------------------------------
input.setWs__func__code("123456");

InOutImpl proxy = new InOutImpl();

INOUTMSG output = new sample.ims.data.INOUTMSG();
output = proxy.runInOut(input);

short outndx = output.getIndx();
System.out.println("¥nOutput value of INDX is: " + outndx);

Inoutmsg_ws__customer outArray[] = output.getWs__customer();

for (int i = 0; i < outndx; i++)
{
 System.out.println(
  "¥n"
   + outArray[i].getWs__cust__name()
   + outArray[i].getWs__cust__number());
}
}
catch (Exception e)
{
 if (e instanceof IMSDFSMessageException)
 {
  System.out.println(
   "¥nIMS returned message: "
    + ((IMSDFSMessageException) e).getDFSMessage());
 }
 else
 {
  System.out.println(
   "¥nIMS Connector exception is: " + e);
 }
}
}
}
```

8. **Ctrl+S** キーを押して、変更を保管します。
9. **次に、アプリケーションをテストします。** **InOutArray** プロジェクトと **sample.ims** パッケージを展開します。
10. **TestInOutProxy.java** クラスを右クリックし、「**実行**」を展開し、「**実行**」>「**Java アプリケーション**」と選択します。
11. コンソールに以下のような出力が表示されれば、テストは成功です。

```
Initial size of input message is: 217
Initial value of INDX is: 3
Adjusted size of input message is: 92
Adjusted size of INDX is: 3
Output value of INDX is: 4
Cathy T.            CN001
Haley F.            CN002
Steve K.            CN003
Kevin F.            CN004
```

おつかれさまでした。これで、入出力配列のチュートリアルは完了です。

## 配列を含む IMS トランザクション用の J2C アプリケーションの作成

このチュートリアルでは、配列を含む IMS トランザクション用の J2C アプリケーションを生成するための、詳細なステップを説明しました。

このチュートリアルでは、「J2C Java Bean」ウィザードを使用して、配列を含む入出力データを持つ IMS トランザクションの処理を行う、シンプルな Web アプリケーションを作成する方法を学習しました。

### 演習での学習事項

演習では、次の作業を行いました。

- 「J2C Bean」ウィザードを使用して、IMS トランザクションを実行する J2C Java Bean を作成する。
- その Bean 用の Java メソッド `runInOut.java` を作成し、IMS トランザクションを実行する。
- テスト・プロキシー Java クラス `TestInOutProxy.java` を作成し、IMS トランザクションの入力メッセージのビルド、IMSトランザクションを実行する J2C Java Bean メソッドの起動、さらに IMS トランザクションから戻される出力データの表示を行う。

  **注:** `TestInOutProxy.java` Java クラスは、英語圏用に作成されているため、その他の地域では、コードを変更する必要がある場合があります。