为 CICS COBOL 副本构建 J2C 应用程序：多个可能的输出

目录

为包含多个可能输出的 CICS 事务创建 J2C 应用程序

本教程描述如何使用 J2C Java™ Bean 向导构建简单的 Web 应用程序，以便处理可以处理多个可能输出的事务。

学习目标

本教程使您能够完成下列任务:

- 使用 J2C Java bean 向导来创建一个 J2C 应用程序，该 J2C 应用程序使用外部调用接口（ECI）来连接 CICS 事务。根据客户的类别（首选客户、一般客户和不良客户）不同，程序返回的有关此客户的输出信息也不同。创建 JSP 以便在 WebSphere Application Server 上部署应用程序。
- 创建可接受客户号的 Java 方法。
- 创建 JSP 以便在 WebSphere Application Server 上部署应用程序。

所需时间

30 分钟

相关信息

查看 PDF 版本

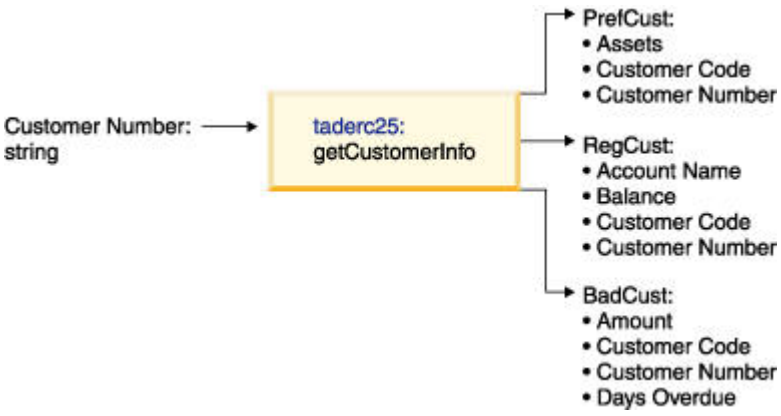
“多个可能的输出”样本

简介: 为包含多个可能输出的 CICS 事务创建 J2C 应用程序

本教程描述如何使用 J2C Java Bean 向导来构建简单 Web 应用程序，以便处理带有多个可能输出的 CICS® 事务。

本教程可能需要一些可选可安装组件。要确保您已安装了适当的可选组件，请参阅“系统要求”列表。

本教程将指导您完成一些详细步骤，以便生成一个 J2C 应用程序，该 J2C 应用程序使用外部调用接口（ECI）与 CICS 事务进行交互。此服务是从 CICS COBOL 函数 getCustomerInfo 构建的，该函数接受客户号。根据客户的类别（首选客户、一般客户和不良客户）不同，程序返回的有关此客户的输出信息也不同。



本教程分为一些练习，必须按顺序完成这些练习，教程才有意义。本教程教您如何使用 J2C Java bean 向导来连接至 CICS ECI 服务器。在完成这些练习的过程中，您将执行下列操作:

- 通过 J2C Java bean 向导来创建使用外部调用接口（ECI）与 CICS 事务进行连接的 J2C 应用程序。
- 创建 Java 方法 *getCustomerInfo*，该方法接受客户号。根据客户的类别（首选客户、一般客户和不良客户）不同，程序返回的有关此客户的输出信息也不同。
- 创建 Java 类 TestECIMPO.java 以测试应用程序。

所需时间

完成本教程大约需要 30 分钟。如果研究其他与本教程相关的概念，则完成本教程可能需要更长时间。

技能级别

熟练

用户

本教程面向熟悉企业信息系统（EIS），特别是熟悉 CICS ECI 的用户。

系统要求

为了完成本教程，需要安装下列工具和组件:

- IBM® WebSphere® Application Server V6.1
- J2EE 连接器（J2C）工具
- **与 CICS ECI 服务器的连接**: 在本教程中，您的应用程序与服务器上的 CICS 程序进行交互。具体地说，您需要在机器上设置 CICS 事务网关以便访问该服务器。如果想要在 CICS 服务器机器上运行 CICS，则还需要对该服务器进行一些设置。这些步骤在此处未作讨论。
- **COBOL 文件 taderc25.cbl 的一个副本**。可以在产品安装目录中找到此文件: <installdir>\IBM\SDP70Shared\plugins\com.ibm.j2c.cheatsheet.content_7.0.0\Samples\CICS\taderc25。如果想将它存储到本地，可从以下链接复制其代码: 第 3 页的『taderc25.cbl』
- **一个干净的工作空间**。

要使用本教程，必须安装并配置了应用程序服务器。要验证是否提供了服务器运行时环境，请单击**窗口 → 首选项**，展开**服务器**，然后单击**已安装的运行时**。可以使用此窗格来添加、除去或编辑已安装的服务器运行时定义。还可以下载并安装对新服务器的支持。

必备知识

为了能够彻底完成本教程，您应该熟悉下列内容:

- J2EE 和 Java 编程
- COBOL 编程语言
- CICS ECI 服务器技术

课程 1.1: 选择资源适配器

本课程将指导您完成一些详细的步骤来选择并配置资源适配器以连接至 CICS ECI 服务器。

连接至 CICS ECI 服务器

1. 如果工作空间的右上角选项卡中未出现 J2EE 图标  ，则您需要切换至 J2EE 透视图。从菜单栏中，选择**窗口 > 打开透视图 > 其他**。将打开“选择透视图”窗口。
2. 选择 **J2EE**。
3. 单击**确定**。将打开 J2EE 透视图。
4. 在 J2EE 透视图中，选择**文件 > 新建 > 其他**。
5. 在“新建”页面中，选择 **J2C > J2C Java Bean**。单击**下一步**。
6. 在“资源适配器”页面中，在**查看依据**下面，选择 **JCA 版本**。展开 **1.5** 并选择 **ECIResourceAdapter (IBM:6.0.2)**。单击**下一步**。
7. 在“连接属性”页面中，选择**非受管连接**。（在本教程中，将使用非受管连接来直接访问 CICS 服务器，因此您不需要提供 JNDI 名称。）接受缺省的**连接类名** `com.ibm.connector2.cics.ECIManagedConnectionFactory`。在空白字段中，提供连接信息。用星号（*）指示的必填字段有:
   - **服务器名称**: （不是必需的）CICS 事务网关服务器的名称。
   - **连接 URL***: （必需）CICS ECI 服务器的服务器地址。
   - **端口号**: （不是必需的）用来与 CICS 事务网关通信的端口号。缺省端口是 2006。
   - **用户名**: （必需）用于建立连接的用户名。
   - **密码**（必需）用于建立连接的密码。
   - :

   可从 CICS 服务器系统管理员获取连接信息。
8. 单击**下一步**。

taderc25.cbl

以下是 taderc25.cbl 中的代码:

taderc25.cbl

```
     identification division.
program-id. TADERC25.
       environment division.
       data division.
       working-storage section.
       01 tmp pic a(40).
       01  ICOMMAREA.
              02  ICustNo    PIC X(5).
              02  Ifiller    PIC X(11).
       01  GENCUST.
              02  GCUSTCODE PIC X(4).
              02  GFILLER PIC X(40).
       01  PREFCUST.
              02  PCUSTCODE PIC X(4).
              02  PCUSTNO    PIC X(5).
              02  ASSETS  PIC S9(6)V99.
       01  REGCUST.
              02  RCUSTCODE PIC X(4).
              02  RCUSTNO    PIC X(5).
              02  ACCOUNTNAME PIC A(10).
              02  BALANCE PIC S9(6)V99.
       01  BADCUST.
              02  BCUSTCODE PIC X(4).
              02  BCUSTNO    PIC X(5).
              02  DAYSOVERDUE PIC X(4).
              02  AMOUNT PIC S9(6)V99.
        LINKAGE SECTION.
```

```
       01 DFHCOMMAREA.
             02 inputfield pic x(50).
       procedure division.
       start-para.
               move DFHCOMMAREA to ICOMMAREA.
               IF ICustNo EQUAL '12345'
                     move 'PREC' to PCUSTCODE
                     move ICustNo to PCUSTNO
                     move 43456.33 to ASSETS
                     move PREFCUST TO DFHCOMMAREA
               ELSE IF ICustNo EQUAL '34567'
                               move 'REGC' to RCUSTCODE
                               move ICustNo to RCUSTNO
                               move 'SAVINGS' TO ACCOUNTNAME
                               move 11456.33 to BALANCE
                               move REGCUST TO DFHCOMMAREA
                           ELSE
                               move 'BADC' to BCUSTCODE
                               move ICustNo to BCUSTNO
                               move '132' to DAYSOVERDUE
                               move -8965.33 to AMOUNT
                               move BADCUST TO DFHCOMMAREA
        *          END-IF.
                   END-IF.
                   EXEC CICS RETURN
       END-EXEC.
```

课程 1.2: 设置 Web 项目以及 Java 接口和实现

本教程指导您完成设置 Web 项目以及 Java 接口和实现。

在开始之前，必须先完成课程 1.1。在本课程中，您将完成下列任务:

- 创建 J2C Java bean
- 创建动态 Web 项目

1. 在工作台中完成的所有工作都必须与一个项目相关联。项目为工作文件和目录提供了一个有组织的视图，并根据项目的类型对视图功能进行了优化。在工作台中，所有文件都必须位于一个项目中，因此在创建 J2C Java bean 之前，首先需要创建用于存储此 bean 的项目。
2. 在“新建 J2C Java Bean”页面的**项目名称**字段中，输入值 `Taderc25Sample`。
3. 单击**项目名称**字段旁边的**新建**来创建新项目。
4. 在“创建新的源项目”页面中，选择 **Web 项目**，并单击**下一步**。
5. 在“新建动态 Web 项目”页面中，单击**显示高级**。
6. 确保选择了下列值:
   a. **项目名称**: Taderc25Sample
   b. **项目内容**: 接受缺省值
   c. **目标运行时**: WebSphere Application Server V6.1
   d. **配置**: 接受缺省值
   e. **将项目添加至 EAR**: 已选中
   f. **EAR 项目名称**: Taderc25SampleEAR
7. 单击**完成**。
8. 可能会出现一个对话框询问是否要切换到动态 Web 透视图。单击**是**。
9. 在“J2C Java Bean 输出属性”页面上:
   a. 在**包名**字段中，单击**浏览**并选择 `Taderc25Sample` 项目。单击**确定**。

b. 在**包名**字段中输入 sample.ims。

c. 在**接口名称**字段中输入 CustomerInfoMO。

d. 在**实现名称**字段中输入 CustomerInfoMOImpl。

e. 不要选中**将会话另存为 Ant 脚本**。

10. 单击**完成**。

课程 1.3: 创建 Java 方法

课程 1.3 将指导您完成创建 Java 方法。

在开始之前，必须完成“课程 1.2: 设置 Web 项目以及 Java 接口和实现”。在本课程中，您将完成下列任务:

- 创建 Java 方法
- 在 COBOL 与 Java 之间创建输入和输出数据映射

1. **首先将创建 Java 方法**: 现在将创建一个 Java 方法，该 Java 方法将使用 COBOL 导入器在 COBOL 源与 Java 方法中的数据之间映射数据类型。
2. 通过单击**窗口 > 显示视图 > 片段**来打开“片段”视图。在“片段”视图中，单击 **J2C**。
3. 右键单击**将 Java 方法添加至 J2C Java bean** 并选择**插入**。
4. 在“新建 Java 方法”页面中，单击**添加**。
5. 在 **Java 方法名称**字段中，输入 getCustomerInfo 作为操作的名称。单击**下一步**。
6. **接下来，将创建输入参数数据映射**: 在此步骤中，将导入 第 3 页的『taderc25.cbl』（COBOL）文件，需要此文件来创建应用程序。taderc25.cbl 文件位于 <installdir>\IBM\SDP70Shared\plugins\com.ibm.j2c.cheatsheet.content_7.0.0\Samples\CICS\taderc25 中，其中 <installdir> 是此产品的安装目录。此 COBOL 文件包含在 CICS 服务器上运行的程序。它包含将通过通信区（COMMAREA）传递给 CICS 服务器的结构的定义。此结构表示从 CICS 应用程序返回的客户记录。必须将文件从文件系统导入工作台之后才能够处理该文件。在“Java 方法”页面的**指定输入/输出类型**字段中，单击**新建**。
7. 在“数据导入”页面中，确保**选择映射**字段是 **COBOL_TO_JAVA**。单击“COBOL 文件”字段旁边的**浏览**。
8. 在文件系统中找到 taderc25.cbl 文件，并单击**打开**。
9. 单击**下一步**。
10. 在“COBOL 导入器”页面中，选择一种**通信数据结构**:
    a. 选择 **Win32** 作为**平台名称**。
    b. 选择 **ISO-8859-1** 作为**代码页**。
    c. 单击**查询**。
    d. 对**数据结构**选择 **ICOMMAREA**。
11. 单击**下一步**。
12. 在“保存属性”页面中:
    a. 对**生成样式**选择**缺省值**。
    b. 单击**浏览**来选择 Web 项目 **Taderc25Sample**。
    c. 在**包名**字段中，输入 sample.cics.data。
    d. 在**类名**字段中，缺省值为 **ICOMMAREA**，将此缺省值替换为 InputComm。
13. 单击**完成**。
14. **接下来将为输出参数创建多个可能输出**: 在“Java 方法”页面的**指定输入/输出类型**字段中，单击“输出类型”区域旁边的**新建**。

15. 在“数据导入”页面中，确保“选择映射”字段是 **COBOL_MPO_TO_JAVA**。
16. 单击多个可能输出区域旁边的**新建**。
17. 单击“COBOL 文件名”字段旁边的**浏览**，并找到 **taderc25.cbl** 文件所在的位置。单击**打开**。
18. 单击**下一步**。
19. 在“COBOL 导入器”页面中，选择一种**通信数据结构**:
    a. 选择 **Win32** 作为**平台名称**。
    b. 选择 **ISO-8859-1** 作为**代码页**。
    c. 单击**查询**。
    d. 对于**数据结构**，选择 **PREFCUST**、**REGCUST** 和 **BADCUST**。
20. 单击**完成**。在“指定数据导入配置属性”页面上，您将看到列示了这三种数据类型。
21. 单击**下一步**。
22. **接下来将指定保存属性**: 在“保存属性”页面中，您将看到为每个客户类型记录设置的缺省值。确保**项目名称**字段中的内容为 **Taderc25Sample**。单击**浏览**并选择 Web 项目 **Taderc25Sample**。
    a. 在“指定保存属性”页面中，突出显示 **COBOL MPO 至 Java 保存属性**。
       - 在**包名**字段中输入 `sample.cics.data`。
       - 在**类名**字段中输入 `OutputComm`。
       - 可以选择**覆盖现有类**。
    b. 展开 **COBOL MPO 至 Java 保存属性**。应出现三个数据绑定元素。
    c. 突出显示 **COBOL 至 Java 保存属性（对于 taderc25.cbl 文件中的“PREFCUST”）**
       - 对于**生成样式**，选择**缺省值**。
       - 在**包名**字段中输入 `sample.cics.data`。
       - 在**类名**字段中输入 `PrefCust`。
       - 可以选择**覆盖现有类**。
    d. 突出显示 **COBOL 至 Java 保存属性（对于 taderc25.cbl 文件中的“REGCUST”）**。
       -
       - 在**包名**字段中输入 `sample.cics.data`。
       - 在**类名**字段中输入 `RegCust`。
       - 可以选择**覆盖现有类**。
    e. 突出显示 **COBOL 至 Java 保存属性（对于 taderc25.cbl 文件中的“BADCUST”）**。
       - 在**包名**字段中输入 `sample.cics.data`。
       - 在**类名**字段中输入 `BadCust`。
       - 可以选择**覆盖现有类**。
23. 单击**完成**。展开 *OutputComm*，您将看到它在**输出类型**字段中包含 PrefCust、RegCust 和 BadCust。
24. 在“Java 方法”页面中，单击**完成**。
25. 在“Java 方法”页面中:
    a. 在 **functionName** 字段中输入 `TADERC25`（COBOL 程序标识）。
    b. 选择**显示高级**。
    c. 在 **interactionVerb** 字段中选择 **SYNC_SEND_RECEIVE(1)**。
    d. 在 **replyLength** 字段中输入 `-1`。
26. 单击**完成**。

27. **现在将添加识别模式标记以生成 Java 输出数据映射文件**: 因为返回的输出可以是这些数据类型中的任何一种数据类型，所以与某种数据类型相匹配的唯一方法就是在数据流中预定义一些模式。匹配方法将检查识别模式。

```
@type-descriptor.external-decimal-td
@type-descriptor.float-td
@type-descriptor.initial-value
@type-descriptor.integer-td
@type-descriptor.level88
@type-descriptor.packed-decimal-td
@type-descriptor.recognition-desc
@type-descriptor.restriction
@type-descriptor.simple-instance-td
```

a. 要为 PrefCust 添加识别模式:
   - 在 Java 编辑器中打开 PrefCust.java 文件。
   - 浏览到 **getPcustcode()** 方法。完成此任务的最佳方法是打开“大纲”视图，然后向下滚动，直到找到所需要的方法为止。
   - 在方法注释区域，添加标记 `@type-descriptor.recognition-desc pattern="PREC"`；或者通过按 Alt+/ 使用内容辅助并向下浏览列表找到此标记，然后输入`"PREC"`作为模式。
   - 保存更改，将会重新生成代码 PrefCust.java。
   - 浏览到匹配方法以确保进行了更改:

```
/**
  * @generated

  */
 public boolean match(Object obj) {
  if (obj == null)
   return (false);
  if (obj.getClass().isArray()) {
   byte[] currBytes = buffer_;
   try {
    byte[] objByteArray = (byte[]) obj;
    buffer_ = objByteArray;
    if (!("PREC".equals(getPcustcode().toString())))
     return (false);
   } catch (ClassCastException exc) {
    return (false);
   } finally {
    buffer_ = currBytes;
   }
  } else
   return (false);
  return (true);
 }
```

b. 要为 RegCust 添加识别模式:
   - 在 Java 编辑器中打开 RegCust.java 文件。
   - 浏览到 **getRcustcode()** 方法。完成此任务的最佳方法是打开“大纲”视图，然后向下滚动，直到找到所需要的方法为止。

- 在方法注释区域，添加标记 @type-descriptor.recognition-desc pattern="REGC"；或者通过按 Alt+/ 使用内容辅助并向下浏览列表找到此标记，然后输入"REGC"作为模式。
- 保存更改，将会重新生成代码 RegCust.java。
- 浏览到匹配方法以确保进行了更改:

```
/**
 * @generated

 */
public boolean match(Object obj) {
 if (obj == null)
  return (false);
 if (obj.getClass().isArray()) {
  byte[] currBytes = buffer_;
  try {
   byte[] objByteArray = (byte[]) obj;
   buffer_ = objByteArray;
   if (!("REGC".equals(getRcustcode().toString())))
    return (false);
  } catch (ClassCastException exc) {
   return (false);
  } finally {
   buffer_ = currBytes;
  }
 } else
  return (false);
 return (true);
}
```

c. 要为 BadCust 添加识别模式:

- 在 Java 编辑器中打开 BadCust.java 文件。
- 浏览到 **getBcustcode()** 方法。完成此任务的最佳方法是打开“大纲”视图，然后向下滚动，直到找到所需要的方法为止。
- 在方法注释区域，添加标记 @type-descriptor.recognition-desc pattern="BADC"；或者通过按 Alt+/ 使用内容辅助并向下浏览列表找到此标记，然后输入"BADC"作为模式。
- 保存更改，将会重新生成代码 BadCust.java。
- 浏览到匹配方法以确保进行了更改:

```
/**
 * @generated

 */
public boolean match(Object obj) {
 if (obj == null)
  return (false);
 if (obj.getClass().isArray()) {
  byte[] currBytes = buffer_;
  try {
   byte[] objByteArray = (byte[]) obj;
   buffer_ = objByteArray;
   if (!("BADC".equals(getBcustcode().toString())))
    return (false);
  } catch (ClassCastException exc) {
   return (false);
  } finally {
   buffer_ = currBytes;
  }
 } else
  return (false);
 return (true);
}
```

课程 1.4: 部署应用程序

课程 1.4 将指导您创建 Java 类以测试应用程序。

在开始之前，必须先完成课程 1.3。在本课程中，您将完成下列任务:

- 创建 Java 类以测试应用程序。
- 运行测试类。

1. **首先您将创建 TestECIMPO 文件**: 展开 **CustomerProj** 项目，然后展开 **Java 资源**部分并选择 **sample.cics** 包。
2. 右键单击并选择**新建**。选择  类选项以创建新的 Java 类。
3. 在 **Java 类名**字段中，输入 TestECIMPO。
4. 在 Java 编辑器中打开 **TestECIMPO**。
5. 使用下列代码替换编辑器中的所有代码:

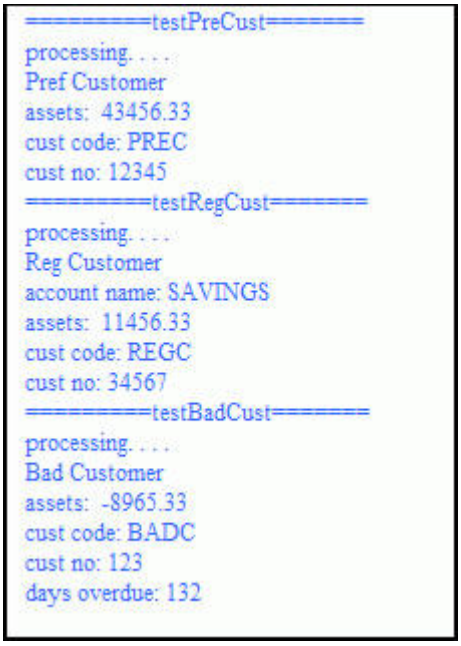
   注: 已经为英语语言环境创建了 Java 类 TestECIMPO.java，您可能需要对其他语言环境的代码进行修改。

```
/*****************************************************************
  * Licensed Materials - Property of IBM
 *
  * com.ibm.j2c.cheatsheet.content
 *
  * Copyright IBM Corporation 2004. All Rights Reserved.
*
  * Note to U.S. Government Users Restricted Rights:
Use, duplication or disclosure restricted by GSA ADP  Schedule Contract with IBM Corp.
 *****************************************************************/
package sample.cics;

import sample.cics.data.*;
public class TestECIMPO
{



 public static void process(InputComm input)
 {

  System.out.println("processing....");
try {
  //CustomerInfoMOImpl proxy = new CustomerInfoMOImpl();
  CustomerInfoMOImpl proxy = new CustomerInfoMOImpl();
       OutputComm output = proxy.getCustomerInfo (input);

  BadCust badCust = output.getBadCust();
  PrefCust prefCust = output.getPrefCust();
  RegCust regCust = output.getRegCust();

  if (regCust != null)
  {
   System.out.println("Reg Customer");
   System.out.println("account name:  " + regCust.getAccountname());
   System.out.println("balance:  " + regCust.getBalance());
   System.out.println("cust code:  " + regCust.getRcustcode());
   System.out.println("cust no:  " + regCust.getRcustno());
  }
  else if (prefCust != null)
  {
   System.out.println("Pref Customer");
   System.out.println("assets:  " + prefCust.getAssets());
   System.out.println("cust code:  " + prefCust.getPcustcode());
```

```
  System.out.println("cust no:  " + prefCust.getPcustno());
 }
 else if (badCust != null)
 {
  System.out.println("Bad Customer");
  System.out.println("amount:  " + badCust.getAmount());
  System.out.println("cust code:  " + badCust.getBcustcode());
  System.out.println("cust no:  " + badCust.getBcustno());
  System.out.println("days overdue:  " + badCust.getDaysoverdue());
 }
 else
  System.out.println("No match");
}
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}

}

public static void testPrefCust()
{
   System.out.println("===========testPreCust==============");
try {
 InputComm input = new InputComm();
 String prefC = "12345";
 input.setICustNo (prefC);
 process(input);
}
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}


}

public static void testRegCust()
{
   System.out.println("===========testRegCust==============");
try {
 InputComm input = new InputComm();
 String regC = "34567";
 input.setICustNo (regC);
 process(input);
}
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}

}

public static void testBadCust()
{

 System.out.println("===========testBadCust==============");
try {

 InputComm input = new InputComm();
 String badC = "123";
 input.setICustNo (badC);
 process(input);
```

```
}
catch (Exception exc)
{
 System.out.println (exc);
 exc.printStackTrace();
}
}


public static void main (String[] args)
{
 testPrefCust();
 testRegCust();
 testBadCust();

 }
}
```

6. **接下来，将测试此应用程序**
7. 右键单击 **TestECIMPO.java** 并选择**运行方式 > Java 应用程序**。
8. 控制台中应显示以下输出:



祝贺您！您已完成 CICS Taderc25 教程。

总结: 为包含多个可能输出的 CICS 事务创建 J2C 应用程序

本教程已教您如何使用 J2C Java Bean 向导构建简单的 Web 应用程序，以便处理带有多个可能输出的 CICS 事务。

已学习的课程

在本教程中，您已经学习了如何完成下列任务:

- 通过 J2C Java bean 向导来创建使用外部调用接口（ECI）与 CICS 事务进行连接的 J2C 应用程序。
- 创建 Java 方法 **getCustomerInfo**，该方法接受客户号。根据客户的类别（首选客户、一般客户和不良客户）不同，程序返回的有关此客户的输出信息也不同。

- 创建测试 Java 类来测试应用程序。
-