Oracle® Tuxedo

Programming an Oracle Tuxedo Application Using Java

12*c* Release 1 (12.1.1)

June 2012

ORACLE®

Contents

CHAPTER 1

Introduction to Oracle Tuxedo Java Programming

This topic includes the following sections:

- Overview
- Programing Guidelines

Overview

- Developing ATMI Java Server

  You can develop the ATMI Java server using pure Java language. The server implemented with Java language functions the same as other Tuxedo servers. You can call the services advertised by the Java-implemented server using ATMI interfaces from client/ Tuxedo server, and similarly, you can call the services advertised by the Tuxedo server using TJATMI interfaces from the java-implemented server.

  You can use the TJATMI interface, JATMI TypedBuffers, POLO java object, and other java technology to implement your services.

- Developing Oracle Tuxedo Java Clients

  It is same as development of existing non-Java clients. You can call java-implemented services from any type of clients, such as native clients, /WS clients, and Jolt clients.

Programing Guidelines

The following conventions should be complied when programing Oracle Tuxedo applications using Java:

- The ATMI Java server classes that implement the services should inherit from the TuxedoJavaServer class.
- The method to be advertised as a service should take the `TPSVCINFO` interface as its only input argument.
- You need to provide the default constructor for the ATMI Java server.
- You need to implement the `tpsvrinit()` method which is called when server starts up or reloads and put the class scope initialization in this method.
- You need to implement the `tpsvrdone()` method which is called when the server is shut down or reloaded and put the class scope cleanup actions in this method.
- The `tpreturn()` in Java server does not immediately disrupt the service method execution, which is different from how it behaves in the existing Tuxedo system. Unlike in the existing Tuxedo system, the Tuxedo system automatically transfers the control flow to caller function when a `tpreturn()` is called, in ATMI Java server, you must explicitly implement the control flow after calling `tpreturn()`. It is better to use `tpreturn()` to return the service data and status to client. If you throw the exception in service, the ATMI Java server will return the TPFAIL to client.

CHAPTER 2

Programming Environment

This topic includes the following sections:

- Updating the UBB Configuration File

Updating the UBB Configuration File

You need to configure the path of where the Tuxedo Java server finds the configuration file for the Java-implemented services in CLOPT.

As the ATMI Java server is a multithread server, you also need to specify the limit of dispatching threads. For more information about the multithread server configuration, see Defining the Server Dispatch Threads.

Listing 2-1 shows an example of UBB configuration for ATMI Java Server.

Listing 2-1 UBB Configuration for ATMI Java Server

```
*SERVERS

TMJAVASVR SRVGRP=TJSVRGRP SRVID=3

CLOPT="-- -c /home/oracle/app/javaserver/TJSconfig.xml"

MINDISPATCHTHREADS=2 MAXDISPATCHTHREADS=3
```

See Also

- *Setting Up an Oracle Tuxedo Application*
- `UBBCONFIG(5)` in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

CHAPTER 3

ATMI Java Server User Interfaces

This topic includes the following sections:

- TuxedoJavaServer
- Oracle Tuxedo Java Context
- TJATMI Primitives for Tuxedo Java Applications
- TypedBuffers for Tuxedo Java Applications
- Get/Set Service Information

TuxedoJavaServer

`TuxedoJavaServer` is an abstract class, which should be inherited by all the user-defined classes that implement the services.

Table 3-1 TuxedoJavaServer Interfaces

Function	Description
`tpsvrinit`	An abstract method, which should be implemented by child class to do some initialization works
`tpsvrdone`	An abstract method, which should be implemented by child class to do some cleanup works
`getTuxAppContext`	Use to retrieve the current attached Tuxedo application Java context.

Oracle Tuxedo Java Context

To access the TJATMI primitives provided by Oracle Tuxedo Java Support, you need to get a `TuxAppContext` object that implements all the TJATMI primitives.

Because the service class inherits from `TuxedoJavaServer`, you can call `getTuxAppContext()` in the service to get the context object.

TJATMI Primitives for Tuxedo Java Applications

TJATMI is a set of primitives that provides communication between clients and servers, such as calling the services, starting and ending transactions, getting the connection to DataSource, logging, and etc.

Table 3-2 TJATMI Primitives

Name	Operation
`tpcall`	Use for synchronous invocation of an Oracle Tuxedo service during request/response communication.
`tpreturn`	Use to set the return value and data in Tuxedo Java Server.
`tpbegin`	Use to begin a transaction.
`tpcommit`	Use to commit the current transaction
`tpabort`	Use to abort the current transaction
`tpgetlev`	Use to check if a transaction is in progress
`getConnection`	Use to get a connection to the configured DataSource
`userlog`	Use to print the user log in Tuxedo user log file

Note: The service continues running after `tpreturn` ends execution. It is recommended put `tpreturn()` as the last executive statement in the service.

TypedBuffers for Tuxedo Java Applications

ATMI Java server reuses the Oracle WebLogic Tuxedo Connector TypedBuffers that corresponds to Oracle Tuxedo typed buffers. Messages are passed to servers in typed buffers. The ATMI Java server provides the following buffer types in Table 3-3:

Table 3-3 TypedBuffers

Buffer Type	Description
`TypedString`	Buffer type used when the data is an array of characters that terminates with the null character. Oracle Tuxedo equivalent: `STRING`.
`TypedCArray`	Buffer type used when the data is an undefined array of characters (byte array), any of which can be null. Oracle Tuxedo equivalent: `CARRAY`.
`TypedFML`	Buffer type used when the data is self-defined. Each data field carries its own identifier, an occurrence number, and possibly a length indicator. Oracle Tuxedo equivalent: `FML`.
`TypedFML32`	Buffer type similar to TypeFML but allows for larger character fields, more fields, and larger overall buffers. Oracle Tuxedo equivalent: `FML32`.
`TypedXML`	Buffer type used when data is an XML based message. Oracle Tuxedo equivalent: XML for Tuxedo Release 7.1 and higher.
`TypedView`	Buffer type used when the application uses a Java structure to define the buffer structure using a view description file. Oracle Tuxedo equivalent: `VIEW`
`TypedView32`	Buffer type similar to View but allows for larger character fields, more fields, and larger overall buffers. Oracle Tuxedo equivalent: `VIEW32`.
`TypedMBString`	Buffer type used when the data is a wide array of characters to support multibyte characters. Oracle Tuxedo equivalent: `MBSTRING`.

Get/Set Service Information

Use the `TPSVCINFO` class to get/set service information sent by the Oracle Tuxedo client.

Table 3-4 Getter Functions

Function	Description
`getServiceData`	Use to return the service data sent from the Oracle Tuxedo Client.
`getServiceFlags`	Use to return the service flags sent from the Oracle Tuxedo Client.
`getServiceName`	Use to return the service name that was called.

Table 3-5 Setter Functions

Function	Description
`setServiceData`	Use to set the service data sent from the Oracle Tuxedo Client.
`setServiceFlags`	Use to set the service flags sent from the Oracle Tuxedo Client.
`setServiceName`	Use to set the service name that was called.

Use `TuxATMIReply` to get the reply data and meta-data from a service invocation.

Table 3-6 Getter Functions for Reply

Function	Description
`getReplyBuffer`	Return the (possibly null) typed buffer returned from a service
`gettpurcode`	Return the `tpurcode` returned from a service

Exception

You need to catch the exception thrown by JATMI primitives in the service, such as `tpcall()`. There are two types of exceptions that JATMI can throw:

- `TuxATMITPException`: Exception thrown that represents a JATMI failure.
- `TuxATMITPReplyException`: Exception thrown that represents a JATMI failure when user data is associated with the exception thrown.

You can throw the exception within the service, but it is not recommended throw an exception to client.

CHAPTER 4

Implementing Services in Oracle Tuxedo Java Server

This topic includes the following sections:

- Typical Procedures
- Example: Implementing Java Service without Transaction
- Example: Implementing Java Service with Transaction

Typical Procedures

Typical steps of implementing the services in Oracle Tuxedo Java server are as follows.

1. Define a class that inherits from `TuxedoJavaServer`
2. Provide a default constructor
3. Implement the `tpsvrinit()` and `tpsvrdone()` method
4. Implement the service method which should use `TPSVCINFO` as its only argument parameter, as follows:
   a. Get the `TuxAppContext` object using `getTuxAppContext()` method
   b. Get the client request data using `TPSVCINFO.getServiceData()` method from `TPSVCINFO` class
   c. If you have configured a DataSource, get a connection to the DataSource using `TuxAppContext.getConnection()` method

d. Do the business logic, such as call some other services using `TuxAppContext.tpcall()`, manipulate the database, etc.

e. Allocate a new TypedBuffer and put a reply data in the TypedBuffer

f. Call `TuxAppContext.tpreturn()` to return the reply data to client

Example: Implementing Java Service without Transaction

Following is a simple example that implements the `TOUPPER` service. It includes three steps:

1. Defining Java Classes: Listing 4-1
2. Creating Java Server Configuration File: Listing 4-2
3. Updating UBB Configuration File: Listing 4-3

Defining Java Classes

Listing 4-1 Java Class Definition

```
import weblogic.wtc.jatmi.TypedBuffer;
import weblogic.wtc.jatmi.TypedString;
import com.oracle.tuxedo.tjatmi.*;
/* MyTuxedoServer is user defined class */
public MyTuxedoServer extends TuxedoJavaServer{
    public MyTuxedoServer ()
    {
        return;
    }
    public int tpsvrinit() throws TuxException
    {
        System.out.println("In MyTuxedoServer.tpsvrinit()");
        return 0;
    }
    public void tpsvrdone()
    {
        System.out.println("In MyTuxedoServer.tpsvrdone()");
           return;
    }
public void TOUPPER (TPSVCINFO rqst) {
        TypedBuffer svcData;
        TuxAppContext   myAppCtxt = getTuxAppContext();

svcData = rqst.getServiceData();
        TypedString TbStr = (TypedString)svcData;
System.out.println("svcData:" + TbStr.toString());

String newstr = TbStr.toString();
newstr = newstr.toUpperCase();
TypedString replyTbString = new TypedString(newstr);
myAppCtxt.tpreturn(TPSUCCESS, 0, replyTbString, 0);
    }
}
```

Creating Java Server Configuration File

Listing 4-2 shows an example that exports `MyTuxedoServer.TOUPPER()` method as service name: `TOUPPER`.

Listing 4-2 Java Server Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<TJSconfig>
<ClassPaths>
<ClassPath>>/home/oracle/app/javaserver/MyTuxedoServer.jar</ClassPath>
</ClassPaths>
<TuxedoServerClass>MyTuxedoServer</TuxedoServerClass>
</TJSconfig>
```

Updating UBB Configuration File

Listing 4-3 UBB Config File Configuration

```
*GROUPS
TJSVRGRP  LMID=simple GRPNO=2
*SERVERS
TMJAVASVR SRVGRP= TJSVRGRP  SRVID=4CLOPT="-- -c TJSconfig.xml"
       MINDISPATCHTHREADS=2 MAXDISPATCHTHREADS=2
```

Example: Implementing Java Service with Transaction

Following is a simple example that implements the `TOUPPER` service. It includes three steps:

1. Defining Java Classes: Listing 4-4
2. Creating Java Server Configuration File: Listing 4-5
3. Updating UBB Configuration File: Listing 4-6

Defining Java Classes

Listing 4-4 Class Definition

```
import weblogic.wtc.jatmi.TypedBuffer;
import weblogic.wtc.jatmi.TypedString;
import com.oracle.tuxedo.tjatmi.*;
import java.sql.SQLException;
/* MyTuxedoTransactionServer is user defined class */
public MyTuxedoTransactionServer extends TuxedoJavaServer{
    public MyTuxedoTransactionServer ()
    {
        return;
    }
    public int tpsvrinit() throws TuxException
    {
        System.out.println("In MyTuxedoTransactionServer.tpsvrinit()");
        return 0;
    }

    public void tpsvrdone()
    {
        System.out.println("In MyTuxedoServer.tpsvrdone()");
        return;
    }
public void writeDB_SVCTRN_COMMIT(TPSVCINFO rqst)  {
        TuxAppContext   myAppCtxt;
        TypedBuffer     rplyBuf = null;
        String          strType = "STRING";
        String          ulogMsg;
        TypedString     rqstMsg;
        Connection      connDB = null;
        Statement       stmtDB = null;
        String          stmtSQL;
        int             trnLvl, trnStrtInSVC;
        int             trnRtn;
        int             rc = TPSUCCESS;


        rqstMsg = (TypedString)rqst.getServiceData();
        myAppCtxt = getTuxAppContext();
        myAppCtxt.userlog("JAVA-INFO: Request Message Is \"" +
rqstMsg.toString() + "\"");
        rplyBuf = new TypedString("This Is a Simple Transaction Test from
Tuxedo Java Service");
        long trnFlags = 0;
        try {
            trnStrtInSVC = 0;
            trnLvl = myAppCtxt.tpgetlev();
            if (0 == trnLvl) {
                long trnTime = 6000;
                myAppCtxt.userlog("JAVA-INFO: Start a transaction...");
                trnRtn = myAppCtxt.tpbegin(trnTime, trnFlags);
                myAppCtxt.userlog("JAVA-INFO: tpbegin return " + trnRtn);
                trnStrtInSVC = 1;
            }
            connDB = myAppCtxt.getConnection();
            if (null != connDB) {
                myAppCtxt.userlog("JAVA-INFO: Get connection: (" +
                                  connDB.toString() + ").");
            }
            stmtDB = connDB.createStatement();
            if (null != stmtDB) {
                myAppCtxt.userlog("JAVA-INFO: Create statement: (" +
                                  stmtDB.toString() + ").");
            }
            stmtSQL = "INSERT INTO TUXJ_TRAN_TEST VALUES ('" +
                      rqstMsg.toString() + "')";
           myAppCtxt.userlog("JAVA-INFO: Start to execute sql (" + stmtSQL
+ ")...");
            stmtDB.execute(stmtSQL);
           myAppCtxt.userlog("JAVA-INFO: End to execute sql (" + stmtSQL +
").");
            if (1 == trnStrtInSVC) {
                    myAppCtxt.userlog("JAVA-INFO: tpcommit current
transaction...");
                    trnRtn = myAppCtxt.tpcommit(trnFlags);
                 myAppCtxt.userlog("JAVA-INFO: tpcommit return " + trnRtn);
                    trnStrtInSVC = 0;
                    if (-1 == trnRtn ) {
                      rc = TPFAIL;
    }
            }
        } catch (TuxATMIRMException e) {
           String errMsg = "ERROR: TuxATMIRMException: (" + e.getMessage()
+ ").";
            System.out.println(errMsg);
            myAppCtxt.userlog("JAVA-ERROR: " + errMsg);
    rc = TPFAIL;
        } catch (TuxATMITPException e) {
           String errMsg = "ERROR: TuxATMITPException: (" + e.getMessage()
+ ").";
            System.out.println(errMsg);
            myAppCtxt.userlog("JAVA-ERROR: " + errMsg);
           rc = TPFAIL;
        } catch (SQLException e) {
          String errMsg = "ERROR: SQLException: (" + e.getMessage() + ").";
            System.out.println(errMsg);
            myAppCtxt.userlog("JAVA-ERROR: " + errMsg);
           rc = TPFAIL;
        } catch (Exception e) {
            String errMsg = "ERROR: Exception: (" + e.getMessage() + ").";
            myAppCtxt.userlog("JAVA-ERROR: " + errMsg);
           rc = TPFAIL;
        } catch (Throwable e) {
            String errMsg = "ERROR: Throwable: (" + e.getMessage() + ").";
            myAppCtxt.userlog("JAVA-ERROR: " + errMsg);
            rc = TPFAIL;
        } finally {
            if (null != stmtDB)
                stmtDB.close();
        }
        myAppCtxt.tpreturn(rc, 0, rplyBuf, 0);
    }
}
```

Creating Java Server Configuration File

Listing 4-5 Java Server Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<TJSconfig>
     <ClassPaths>

<ClassPath>/home/oracle/app/oracle/product/11.2.0/dbhome_2/ucp/lib/ucp.jar
</ClassPath>

<ClassPath>/home/oracle/app/oracle/product/11.2.0/dbhome_2/jdbc/lib/ojdbc6
.jar</ClassPath>
     </ClassPaths>


        <DataSources>
           <DataSource name="oracle">

<DriverClass>oracle.jdbc.xa.client.OracleXADataSource</DriverClass>
            <JdbcDriverParams>

<ConnectionUrl>jdbc:oracle:thin:@//10.182.54.144:1521/javaorcl</Connection
Url>
            </JdbcDriverParams>
          </DataSource>
        </DataSources>
       <TuxedoServerClasses>
   <TuxedoServerClass name=" MyTuxedoTransactionServer">
          </TuxedoServerClass>
       </TuxedoServerClasses>
</TJSconfig>
```

Updating UBB Configuration File

Listing 4-6 UBB Conf File Configuration

```
*GROUPS
ORASVRGRP  LMID=simple GRPNO=1
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/zhiyhan/passw0rd+SesTm=120+MaxCur=5+Lo
gDir=.+SqlNet=javaorcl"
TMSNAME=TMSORA TMSCOUNT=2
*SERVERS
TMJAVASVR SRVGRP=ORASVRGRP SRVID=3
       CLOPT="-- -c TJSconfig.xml"
       MINDISPATCHTHREADS=2 MAXDISPATCHTHREADS=4
```

CHAPTER 5

Java Server Configuration File Schema

Listing 5-1 shows the XML schema of the Java server configuration file:

Listing 5-1 Java Server Configuration Schema File

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="TJSconfig">
    <xs:complexType>
      <xs:all>
        <xs:element minOccurs="0" maxOccurs="1" ref="ClassPaths"/>
        <xs:element minOccurs="0" maxOccurs="1" ref="DataSources"/>
        <xs:element minOccurs="0" maxOccurs="1" ref="Resources"/>
       <xs:element minOccurs="1" maxOccurs="1" ref="TuxedoServerClasses"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="ClassPaths">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="ClassPath"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ClassPath" type="xs:string"/>
  <xs:element name="DataSources">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" ref="DataSource"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DataSource">
    <xs:complexType>
      <xs:all>
        <xs:element minOccurs="1" maxOccurs="1" ref="DriverClass"/>
        <xs:element minOccurs="1" maxOccurs="1" ref="JdbcDriverParams"/>
      </xs:all>
      <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="DriverClass" type="xs:string"/>
  <xs:element name="JdbcDriverParams">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element minOccurs="1" maxOccurs="1" ref="ConnectionUrl"/>

      </xs:sequence>

    </xs:complexType>

  </xs:element>

  <xs:element name="ConnectionUrl" type="xs:string"/>

  <xs:element name="Resources">

    <xs:complexType>

      <xs:sequence>

        <xs:element minOccurs="0" maxOccurs="unbounded"
ref="FieldTable16Classes"/>

        <xs:element minOccurs="0" maxOccurs="unbounded"
ref="FieldTable32Classes"/>

        <xs:element minOccurs="0" maxOccurs="unbounded"
ref="ViewFile16Classes"/>

        <xs:element minOccurs="0" maxOccurs="unbounded"
ref="ViewFile32Classes"/>

      </xs:sequence>

    </xs:complexType>

  </xs:element>

  <xs:element name="FieldTable16Classes" type="xs:string"/>

  <xs:element name="FieldTable32Classes" type="xs:string"/>

  <xs:element name="ViewFile16Classes" type="xs:string"/>

  <xs:element name="ViewFile32Classes" type="xs:string"/>

  <xs:element name="TuxedoServerClasses">

    <xs:complexType>

      <xs:sequence>

        <xs:element minOccurs="1" maxOccurs="unbounded"
ref="TuxedoServerClass"/>

      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="TuxedoServerClass">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Services"/>
      </xs:sequence>
      <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Services">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Service"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Service">
    <xs:complexType mixed="true">
      <xs:attribute name="name" use="required" type="xs:string"/>
      <xs:attribute name="target" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```