WebServices with BEA WebLogic

WebServices in a Commercial Product
(Short Version)

Exercise 8

Pizzas(short)

WebServices with BEA WebLogic

WebServices in a Commercial Product (Short Version)

Introduction

BEA WebLogic is BEA's flagship application server product. It is a fully-featured product and provides support for the whole J2EE specification as well as extra facilities for clustering, transaction handling, etc.

WebLogic also supports SOAP and WSDL.

In this exercise you will take a brief look at WebLogic's WebServices support features and also see how SOAP makes it possible for various technologies to interoperate.

Setting Up

There are a number of housekeeping tasks you need to do before you can get started.

Installing the Software

In this session, you will require the following:

- MS SOAP Toolkit 3.0
- Microsoft Windows Script 5.6
- BEA WebLogic Server 8.1

Each of the above are distributed as windows installers.

Install MS SOAP Toolkit

Installing this toolkit is another straightforward task using the installer supplied on your CD in **Resources\soapsdk.exe**.

Install MS Windows Script

If you are using Windows 2000 or above, this facility is part of the standard Operating System, otherwise simply double-click the supplied installer **Resources\scr56en.exe**.

Install BEA Weblogic 8.1

Simply double-click the supplied installer **Resources\WebLogic 8.1\server811_win32.exe**. You should accept all the default settings offered to you by the installer. In particular, this exercise assumes that WebLogic is installed under the directory **C:\bea**.

Pizzas!

WebLogic exposes a large proportion of its infrastructure for the WebServices developer to use. In particular, WebLogic exposes Enterprise Java Beans[†] as WebServices, thus allowing the WebServices developer to utilise the full power of the J2EE environment.

In this section, you will create a Stateless Session EJB (a close analogue to a Servlet) and expose it as a WebService.

Populate Development Directories

Open a Command Prompt window and then execute the following command to set up a new directory for you to 'play' in (in this sheet the DOS command prompt is shown as >, *you should not type this directly*. This command assumes that your CD-ROM is accessible via drive Z:):

```
> xcopy /e/i "Z:\Exercises\8 Pizzasshort\framework" C:\Pizzasshort
```

This directory contains some empty files for you to edit, alongside a number of 'boilerplate' files that you should examine briefly but that should not be changed.

You should do the majority of your work within the **C:\Pizzasshort** directory.

Retrieve a Weblogic Domain

A Weblogic domain is a set of configuration files and code components collected together into a directory and described by an XML configuration file.

A pre-configured, pre-populated domain has been prepared for you.

Execute the following command:

```
> xcopy /e/i "Z:\Exercises\8 Pizzasshort\user_projects.zip" C:\
```

Once you have copied that file, you will need to extract it (use WinZip, included on your CD in the resources directory if not already installed). You should ensure that it is extracted to a directory hierarchy rooted at **C:\bea**. You should end up with the following heirarchy:

C:\bea\user_projects
    domains

[†] An Enterprise Java Bean may be thought of as a "more evolved servlet" that lives within an environment that provides higher functionality support services than those provided to a simple servlet.

pizzadomain
…

Examine the EJB Components (*optional*)

In this session, you will be exposing a Java-based Enterprise Java Bean (EJB) as a Web Service. This component has been supplied for you but you may wish to take a quick look at the various parts that comprise the bean.

An EJB consists of two separate Java interfaces and a single Java class file.

- Home interface
  Provides lifecycle support of the bean (creation, removal, etc.).
- Remote interface
  Provides a means of advertising the business logic methods implemented by the Enterprise Java Bean.
- Bean class
  The implementation class of the EJB.

You may briefly examine these various files.

The Home Interface

Examine the file **C:\pizzasshort\pizzaEJB\PizzaHome.java**:

```
package pizzas.pizzaEJB;

public interface PizzaHome
  extends javax.ejb.EJBHome
  {
  Pizza create()
    throws javax.ejb.CreateException, java.rmi.RemoteException;
  }
```

The Remote Interface

Examine the file **C:\pizzasshort\pizzaEJB\Pizza.java**:

```
package pizzas.pizzaEJB;

import java.rmi.RemoteException;

public interface Pizza
  extends javax.ejb.EJBObject
  {
  public float getStandardPizza ()
    throws RemoteException;
  public float getCustomPizza (String [] toppings)
    throws RemoteException;
  }
```

The Bean Component Class

Examine the file **C:\pizzasshort\pizzaEJB\PizzaBean.java**:

```
package pizzas.pizzaEJB;

public class PizzaBean
  implements javax.ejb.SessionBean
  {
  private final float
    STANDARD_COST = 7.50f,
    TOPPING_COST = 0.5f;

  public void ejbActivate () {  }
  public void ejbRemove () { }
  public void ejbPassivate () { }
  public void setSessionContext (javax.ejb.SessionContext ctx) { }

  public void ejbCreate ()
    throws javax.ejb.CreateException
    { }

  public float getStandardPizza ()
    {
    return (STANDARD_COST);
    }

  public float getCustomPizza (String [] toppings)
    {
    return (getStandardPizza () +
            (toppings.length * TOPPING_COST));
    }
  }
```

The XML Deployment Descriptors

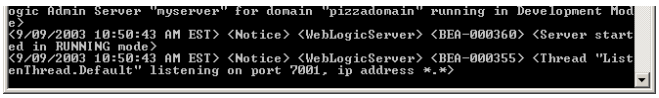
You may also briefly examine the associated XML files: ejb-jar.xml and weblogic-ejb-jar.xml (they are simply text files that can be opened with any editor). Together, these files tell WebLogic which Java class files comprise the bean, how session/transaction handling should be performed on behalf of the bean, etc.

Start WebLogic

To start WebLogic running (and thus, to deploy the Pizza EJB), open up a new Command Prompt and execute the following command sequence:

```
> cd /d C:\bea\user_projects\domains\pizzadomain
> .\startweblogic.cmd
```

You will know that Weblogic has started correctly when you see the "…listening on port 7001…" message, as shown below:

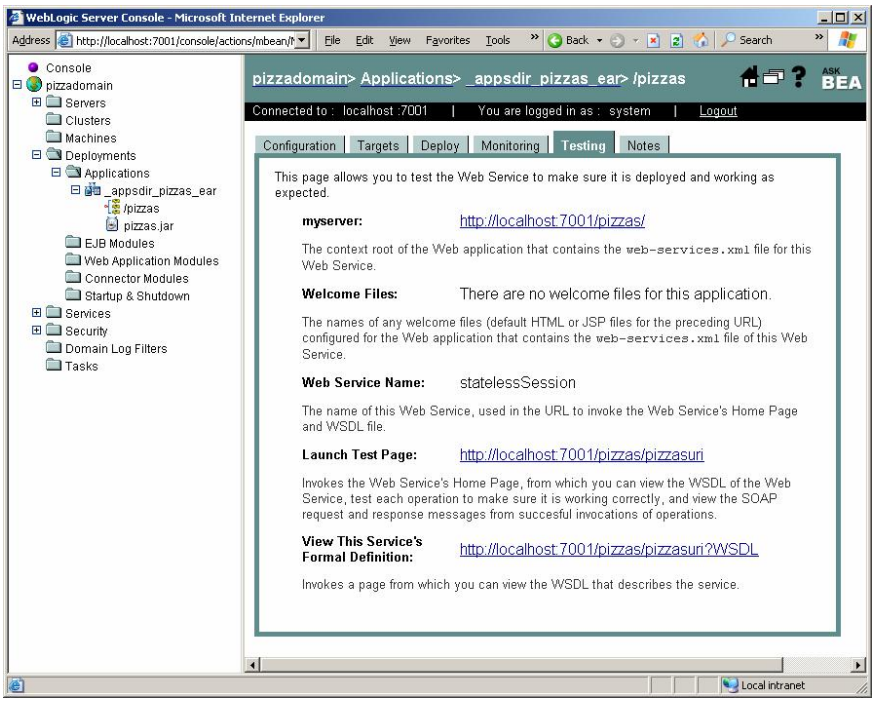


Verify the Deployment with the Console

WebLogic has a web-based console that allows one to examine deployments and the overall operation of the system.

Point your browser to the URL **http://localhost:7001/console**. You will be asked to provide a username and password (use *system* and *weblogic* respectively) and you will see the console. You can navigate around the system using the tree on the left of the window.

The following picture shows what you should see if the Pizzas EJB has been successfully deployed:

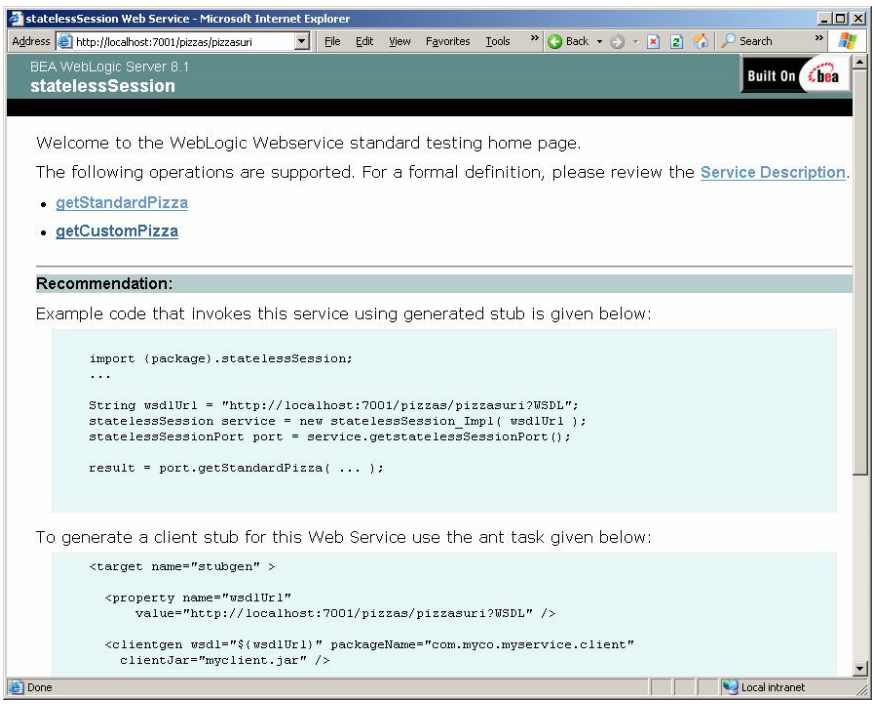


Navigate through the left-hand tree to pizzadomain→Deployments→Applications→ _appsdir_pizzas_ear→/pizzas and look at the various tabs, including the Testing tab, as shown above.

The Testing tab (shown in the above screenshot) lets you view and work with several aspects of the Web Service that is created from the underlying Pizza EJB.

Examine the Web Service Deployment

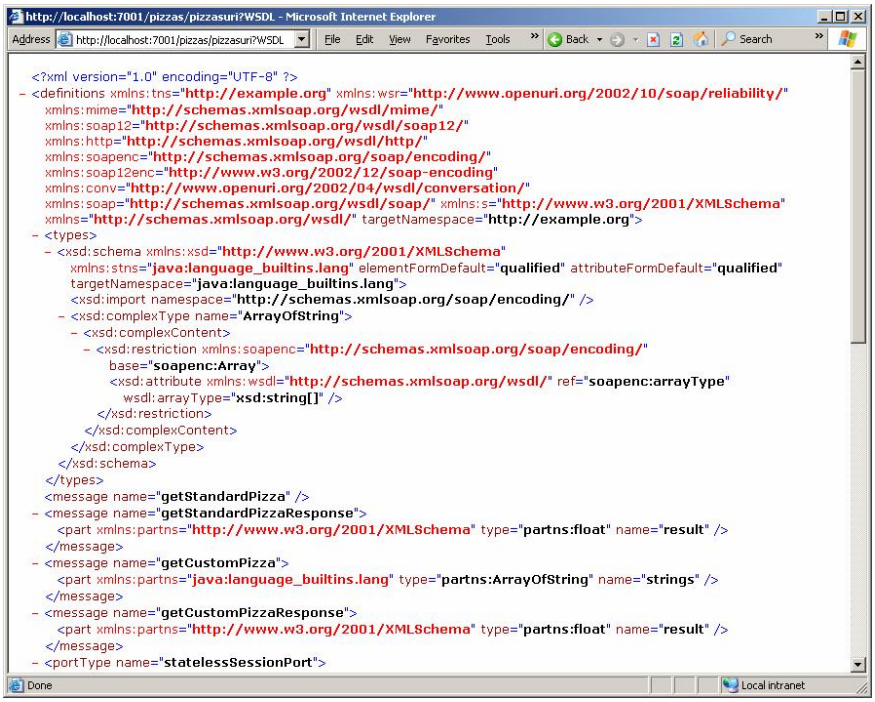
Click on the **http://localhost:7001/pizzas/pizzasuri** link. You will see the following:



This page provides a number of handy recommendations for developers writing code that uses this Web Service.

Examine the WSDL File

You should examine the Web Services Description Language file that you can retrieve by clicking on the “Service Description” link (if you use Netscape, It may not display in the browser window so either save it to disk first, or use the appropriate “View Source” command to examine it once your browser has stopped downloading). You should see something like the following:

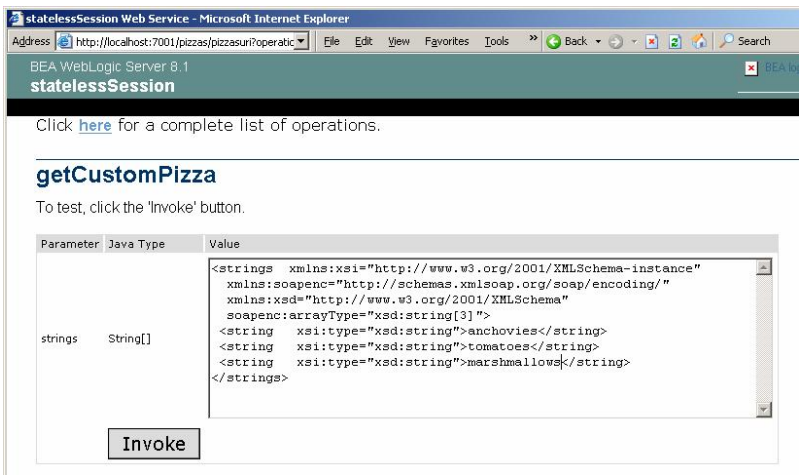


As you examine this file, you should see that it describes everything about the Web Service to make it possible for a “self-configuring” client to come along and interact with the service.

Test a Web Service Method

To make sure that all is working properly, return to the ‘statelessSession’ page and click on the ‘getCustomPizza’ link shown at the top of the page.

You will be presented with a form that makes it possible to pass data to the method to test its operation. Edit the XML request parameter XML so that it passes in a couple of pizza toppings:



To execute the method, press the Invoke button. You will be greeted with a new page echoing the request message data and showing the response message, as well as showing the result of the execution:

This is a useful “quick and dirty” test facility.

Create the Java Client

A bit of Java Development…

Create the following in the file **C:\Pizzasshort\jClient\Client.java,** following the pattern shown to you earlier (in the Weblogic console ‘Testing’ page):

```
package pizzas.jClient;

import java.io.PrintWriter;

public class Client
```

```
{
private static final PrintWriter
  out = new PrintWriter (System.out, true);
public static void main (String[] args)
  throws Exception
  {

  String wsdlUrl = "http://localhost:7001/pizzas/pizzasuri?WSDL";
  StatelessSession
    service = new StatelessSession_Impl( wsdlUrl );
  StatelessSessionPort port = service.getstatelessSessionPort();

  out.println ("Cost of a standard pizza: $" +
    port.getStandardPizza ());

  final String []
    toppings = { "mushrooms", "pepperoni" };
  out.print ("Cost of a pizza with");
  for (int i = 0; i < toppings.length; i ++)
    out.print (" " + toppings [i]);
  out.println (": $" + port.getCustomPizza (toppings));
  }
}
```

Compile and Execute the Java Client

BEA WebLogic 8.1 provides the open source community's Ant tool to let you control and script the development process. This tool is configured via the file build.xml. You should examine this file.

Using Ant is easy. It is necessary to ensure that Ant is executable via the command line and the Java CLASSPATH is configured correctly (this only needs to be done once when a Command Prompt window is initially opened, but it needs to be done each time a new Command Prompt window is opened).

In a new Command Prompt window type the following sequence of commands to build and execute your Java client application:

```
> cd /d C:\Pizzasshort
> C:\bea\user_projects\domains\pizzadomain\setEnv.cmd
> ant –verbose
> ant run
```

You should see the application run and return the price of various pizza combinations (obtained from the pre-built and pre-deployed Pizza EJB).

A Cross-Language Demonstration

Since WebLogic WebServices allow SOAP-based access, any SOAP-aware client can make use of a WebLogic Webservice. This is an *important* feature. The typical J2EE systems allow for Web, Java and CORBA clients but have not had (and still do not have) a convincing way of working with Microsoft COM-based technologies. This has long been a stumbling block for large enterprises. WebServices hold a lot of promise for the many enterprises that already have a heavy investment in Microsoft COM-based technologies.

This section will show how Visual Basic Scripting combined with WebService technology can be used to access a Java-based EJB.

Establish a Working Directory

In a new Command Prompt window execute the commands:

```
> cd /d C:\pizzas\vClient
```

Create the Visual Basic Script

Create the following in the file **pizzas.vbs**:

```
SET soapclient = WScript.CreateObject ("MSSOAP.SoapClient")
CALL soapclient.mssoapinit
  ("http://localhost:7001/pizzas/pizzasuri?WSDL")
WScript.Echo "The cost of a standard Pizza is: $" &
  soapclient.getStandardPizza()
DIM toppings(1)            ' actually *2* elements in the array
toppings(0) = "marshmallows"
toppings(1) = "anchovies"
WScript.Echo "The cost of a custom Pizza is: $" &
  soapclient.getCustomPizza(toppings)
```

Be careful of line breaks:indented lines should be typed directly on the previous line; there should only be seven lines of code in the source file.

Execute the Script

Execute the following simple command:

```
> cscript /nologo pizzas.vbs
```

You will see the now-familiar cost of various pizzas displayed for you.

Do not underestimate the power implied in this simple exercise: you have created an enterprise-grade application in Java and invoked it from a simple script written in VBScript. This is a good example of cross-language interoperability. The solution is capable of working through firewalls and is fairly simple to get going and to administer. WebServices really hold a lot of promise for the future.