Designing Asynchronous Interfaces

The distributed nature of web-based computing introduces unpredictable and sometimes very long latencies, which means it may take an operation a long time to complete. If a process executing over the network involves human interaction at the back end, an operation can take on the order of days. If all interactions over the web were synchronous, clients with pending operations could consume resources on their host systems for unacceptably long periods of time.

BEA Workshop for WebLogic Platform provides tools that make it easy for you to build *asynchronous* web services and controls that don't require clients to block execution while waiting for results. WebLogic for Workshop provides multiple approaches for returning results to your web services' and controls' clients; you can choose the one that best suits each situation.

The topics below describe how to build asynchronous interfaces for your web services and controls.

Current Release Information:

- What's New in 9.2
- Upgrading from 8.1

Useful Links:

- Tutorials
- Tips and Tricks

Other Resources:

- Online Docs
- Dev2Dev
- Discussion Forums
- Development Blogs

Topics Included in This Section

Using Events and Callbacks to Enable Long-Running Operations

Describes when and how to build asynchronous web services and controls by using events, callbacks and buffering

Designing Conversational Web Services

Discusses how to maintain state for an asynchronous web services using conversations.

Related Topics

Web Services

Custom Controls

Using Events and Callbacks to Enable Long-Running Operations

BEA Workshop for WebLogic Platform provides tools that make it easy for you to build asynchronous web services and custom controls. This section introduces the basic concepts of asynchronous computing with Workshop for WebLogic.

Note that WebLogic Platform 9.2 supports three different technologies related to asynchronous computing. These technologies are described below.

(1) Beehive-based Control Events

Control events are part of the Beehive control framework. Control events provide a simplified, flexible model for controls and control clients to trigger and handle events. They can be used to expose different types of events, including local events (from sources inside the same application) and those that may come from an external source such as a web service callback.

(2) Web Service Callbacks

WebLogic Server provides a simplified development model for building web service callbacks. When using callbacks, both the client and the target service must be set up to understand the callback mechanism.

The Workshop for WebLogic service control provides support for clients wishing to communicate with a service that exposes callbacks. With this control, callback from a target service are exposed as Beehive control events to the client of the service control.

(3) WebLogic Server Asynchronous Request-Response:

WebLogic Server asynchronous request-response provides a way for web service clients to asynchronously invoke any web service, whether or not the target web service is designed for asynchronous invocation. Asynchronous request-response gives the client a way to wait for a response from the web service without having to stops its other processes. Asynchronous request-response requires that the client be running on WebLogic Server.

The topics below describe only (1) Beehive-based control events and (2) web service callbacks. For information on WebLogic Server Asynchronous Request-Response see Invoking a Web Service Using Asynchronous Request-Response in the WebLogic Server documentation.

Topics Included in This Section

Web Service Callbacks

Introduces callbacks for web services, and explains when and how to implement them.

Introduction to Control Events

Introduces control events supported by the Beehive controls framework.

Using Buffering to with Web Service Callbacks and Control Events

Explains how buffering methods can help your web services and controls handle high-volume loads.

Related Topics

Weblogic Server documentation: Invoking a Web Service Using Asynchronous Request-Response

Web Service Callbacks

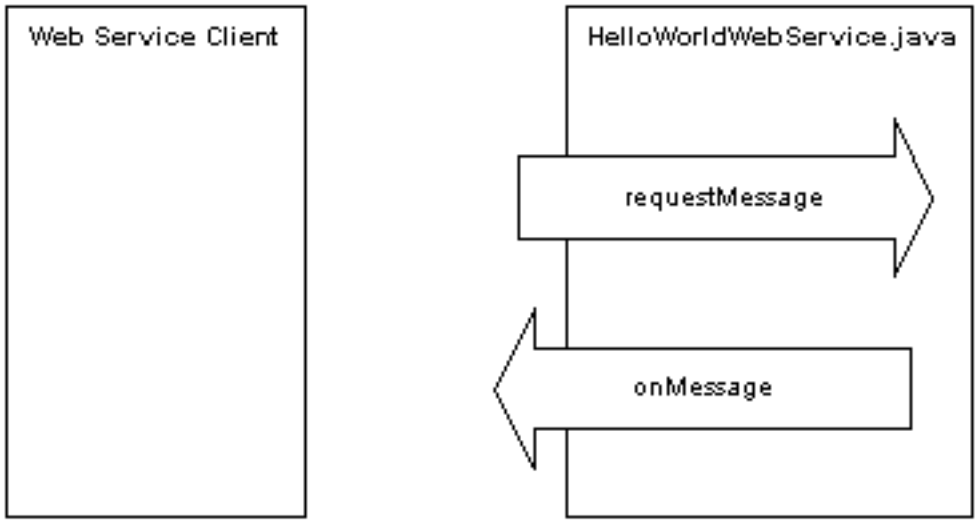
Web services typically use HTTP to provide communication between a client and a server (where the web service resides). HTTP is a *request-response protocol* where each operation consists of a request-response pair: (1) a request message sent from the client to a server followed by (2) a response message returned from the server to the client. The server must always send a response for the operation to complete successfully. Such requests are called *synchronous* because during the request the client is synchronized with the server; the client cannot continue processing until the server responds or the request times out (the client may *time out* if a response is not received within a specific period of time).

Some of the operations the web service performs may be *long-running*. If an operation involves human interaction such as approval by a loan officer of a bank, the operation could take days to complete. It would be a poor design if individual request-response cycles were allowed to span days; such requests would unnecessarily engage resources on both the client and server hosts.

With Workshop for WebLogic and WebLogic Server, you can design your web service to be *asynchronous*, which means that the service disengages from the client after the initial request and *calls back* the client when the final response is ready. This allows the client to continue performing other work while the application completes the requested operation. It also keeps each request-response interaction between the client and application as short as possible. Think of phoning a friend to request some information. If your friend doesn't have the answer right away, you wouldn't sit on hold for hours at a time. Instead you would ask the friend to call you back when he gets the information.

An asynchronous web service provides a method that accept requests from clients that *begin* an operation but do not wait for the operation to complete. The method typically returns immediately, supplying the response portion of the initial request-response interaction but *not* supplying the actual, substantial result of the requested operation. The service also provides a callback method, which sends the results of the long-running operation to the client when the results are finally ready.

The following diagram shows a web service with two methods: an initiating method and a callback method. The client (typically a web service control) invokes requestMessage to initiate the web service's main process. When the main process is complete (that is, when the response is constructed) the onMessage callback method is invoked that sends the response back to the client.



Adding Callback Methods to a Web Service

To add a callback method to a web service, right-click anywhere in the web service source code and select **Insert > Callback**. The default callback interface containing one callback method will be added. By default the callback method is named newMethod(). In the example below, the default method name has been changed to sendMessage().

```
@WebService
public class HelloWorldWebService {


        @Callback
        private CallbackSvc callback;


        @WebMethod
        public void requestMessage() {
        }


        @CallbackService
        public interface CallbackSvc extends CallbackInterface {
                @WebMethod
                public void onMessage();
        }
}
```

Multiple callback methods can be added to the callback interface.

Parameters added to the callback method represent data sent to the client. For example, if you want to send a String response back to the client, add a String parameter to the callback method as follows.

```
        @CallbackService
        public interface CallbackSvc extends CallbackInterface {
                @WebMethod
```

```
            public void sendMessage(String message);
        }
```

To send a response back to the client, you must (1) declare the callback interface and then (2) invoke the appropriate callback method based on that declaration.

(1) To declare the callback interface, use the @Callback annotation. (The Insert > Callback command will automatically supply this declaration.)

```
    @Callback
    private CallbackSvc callback;
```

(2) Then invoke the appropriate callback method:

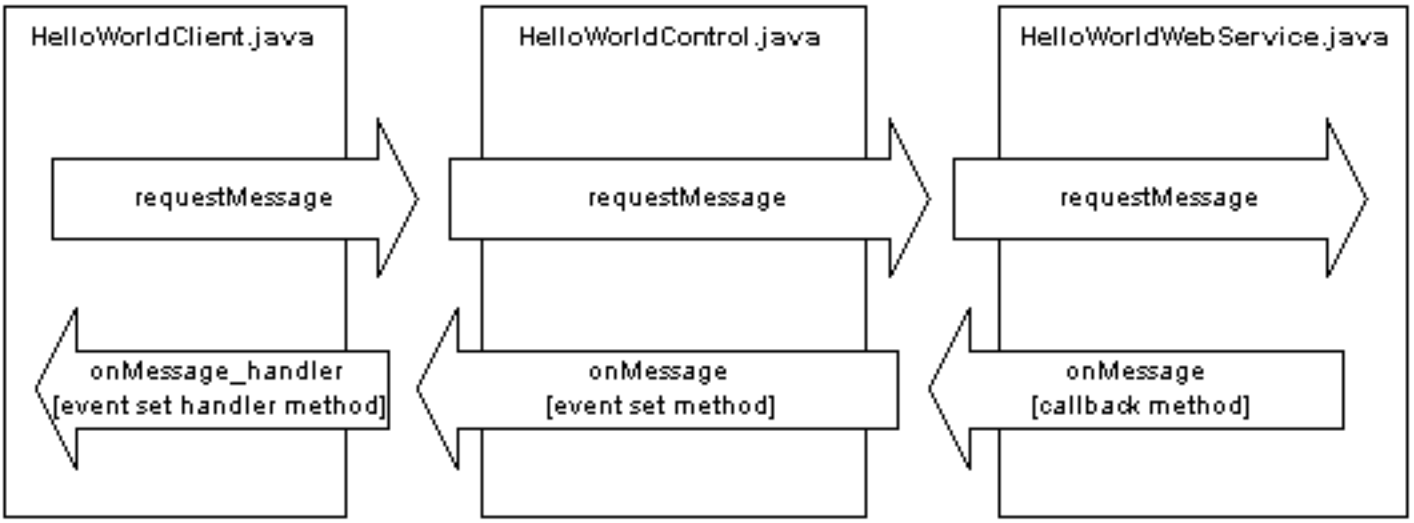
```
@WebService
public class HelloMessage {


        @Callback
        private CallbackSvc callback;


        @WebMethod
        public void initiateRequest() {
            callback.sendMessage("This is your callback message.");
        }


        @CallbackService
        public interface CallbackSvc extends CallbackInterface {
                @WebMethod
                public void sendMessage(String message);
        }
}
```

Typically, a web service communicates with its client through a service control.

The diagram below shows the relationship between a web service client, a service control, and a web service.

The client (on the far left) begins the interaction by requesting a message, a request that is passed on to the web service through the web service control.

When the web service has completed the response, it is passed back to the original client. This process has three stages: (1) The web service invokes its callback method that sends the response to the control. (2) The control event set method listens for the response and passes it to the client. (3) Finally the client handler method listens for the control event set method and receives the event.



For more information about the relationship between a web service and a web service control, see Handling Web Service Callback Messages.

For more information about the relationship between a client and a web service control, see Creating and Using a Service Control.

For more information about testing Web Service callback methods see Testing Web Service Callback Methods.

Related Topics

Building Web Services with Workshop for WebLogic

Creating Conversational Web Services

Creating Buffered Web Services

Sample: Asynchronous Hello World

Control Events

Beehive-based controls allow for the definition of events that can asynchronously send messages to a control client.

For more information about defining events in a control, see Control Event Set Definition.

For more information on setting up a client to handle events sent from a control see Handling Control Events.

Related Topics

Apache Beehive documentation: @EventSet

Apache Beehive documentation: @EventHandler

Apache Beehive documentation: @Client

Using Buffering with Web Service Callbacks and Control Events

For an application receiving high-volume traffic, you may want to add buffers to the application's methods and callbacks. When clients call a buffered method, the call is stored in a buffer and the client does not have to wait for the application to handle the call. When an application sends a callback, the message is stored in a buffer and the application does not have to wait until the client processes the callback.

Buffers can be added to web service methods and callbacks and to control methods, provided the control is used within a web service.

Web Service Buffers

Web service buffers are implemented using the weblogic.jws.MessageBuffer annotation.

For more information on adding buffers to web services see Creating Buffered Web Services in the WebLogic Server documentation.

Control Buffers

Control buffers are implemented using the com.bea.control.annotations.MessageBuffer annotation.

For more information on adding buffers to controls see com.bea.control.annotations.MessageBuffer.

Related Topics

WebLogic Server documentation: JWS Annotation Reference

com.bea.control.annotations.MessageBuffer

Designing Conversational Web Services

A single web service may communicate with multiple clients at the same time, and it may communicate with each client multiple times. In order for the web service or control to track data for the client during asynchronous communication, it must have a way to remember which data belongs to which client and to keep track of where each client is in the process of operations. In BEA WebLogic Platform, you use conversations to uniquely identify a given communication between a client and your application and to maintain state between operations.

The following topics will explain how to design and develop conversational web services:

WebLogic Server documentation: Creating Conversational Web Services

Tutorial: Creating a Web Service with Timer Control

Tutorial-based Samples: TimerTutorial

Related Topics

Asynchrony and Long-Running Operations