11

Web Service Error Handling

Objectives

After completing this lesson, you should be able to do the following:

- Describe how SOAP web services convey errors
- Describe how REST web services convey errors
- Return SOAP faults
- Return HTTP error status codes
- Map thrown exceptions to HTTP status codes
- Handle errors with SOAP clients & Handle errors with Jersey clients

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 1: Introduction to Web Services
- Lesson 2: Creating XML Documents
- Lesson 3: Processing XML with JAXB
- Lesson 4: SOAP Web Services Overview
- Lesson 5: Creating JAX-WS Clients

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 6: Exploring REST Services
- Lesson 7: Creating REST Clients
- Lesson 8: Bottom Up JAX Web Services
- Lesson 9: Top Down JAX Web Services
- Lesson 10: Implementing JAX RS Web Services

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 11: Web Service Error Handling

You are here!

- Lesson 12: Java EE Security and Securing JAX WS

Web Service Errors

Web Services can experience errors in two places:

- On the server
  - In your web service an exception is thrown. How you convey that to a client depends on the type of web service (SOAP or REST)
- On the client
  - Clients receive the errors produced by a web service.
  - Clients experience error without there being any error produced by a server (networking problems, for example).

SOAP Faults

The equivalent of an exception in a SOAP web service is called a "fault."

- Application faults must be listed in the service's WSDL.
  - Similar to methods declaring they throw a checked exception
  - Known as modeled exceptions
  - Mapped to custom classes
- Faults can occur without being in the WSDL.
  - Similar to runtime exceptions, things that shouldn't happen
  - Known as unmodeled exceptions
  - Uses `javax.xml.ws.soap.SOAPFaultException`

Client-Caused Fault: Example

HTTP status: `500 Internal Server Error`

Response Body:

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
  <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
    <faultcode>S:Client</faultcode>
    <faultstring>
     Cannot find dispatch method for {http://ou/}greeting
    </faultstring>
  </S:Fault>
</S:Body>
</S:Envelope>
```

Fault

The `<S:Fault>` element contains:

- `<faultcode>` - Contains a value of: `VersionMismatch`, `MustUnderstand`, `Client`, or `Server`
- `<faultstring>` - A human readable explanation
- `<faultactor>` - A URI that specifies who caused the fault if the message travels through multiple processing nodes. Not always present.
- `<detail>` - Details about why the `<S:Body>` could not be processed. Common place for a stack trace. Not always present.

Bottom-Up SOAP Faults

Any web method may indicate that it throws a checked exception (included application-specific subclasses).

```
public void faultyOne() throws Exception {
    if (true) {
        throw new Exception("my message");
    }
}
```

Bottom-Up `fault` Element

```
<portType name="Faulty">
<operation name="faultyOne">
<input  wsam:Action="http://ou/Faulty/faultyOneRequest"  me
   ssage="tns:faultyOne"/>
<output  wsam:Action="http://ou/Faulty/faultyOneResponse"
   message="tns:faultyOneResponse"/>
<fault  message="tns:Exception"  name="Exception"  wsam:Acti
   on="http://ou/Faulty/faultyOne/Fault/Exception"/>
</operation>
</portType>
```

Bottom-Up Exception Message Type

```
<xs:complexType name="Exception">
<xs:sequence>
  <xs:element name="message" type="xs:string"
    minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

```
<message name="Exception">
<part name="fault" element="tns:Exception"/>
</message>
```

Top-Down Faults: Generated Exception

For each `<fault>` type included in an operation an exception is generated.

```
@WebFault(name = "Exception",
          targetNamespace = "http://ou/")
public class Exception_Exception
    extends java.lang.Exception {
    public Exception_Exception(String message,
        ou.Exception faultInfo) {}
    public Exception_Exception(String message,
        ou.Exception faultInfo, Throwable cause) {}
    public ou.Exception getFaultInfo() {}
}
```

Throwing Top-Down Generated Fault Exceptions

- The operations will already have a throws clause.
- Create the fault info (`<detail>`) instance.
- Create and throw the wrapper exception.

```
public void faultyOne() throws Exception_Exception
{
 ou.Exception oue = new ou.Exception();
 oue.setMessage("detail message");
 throw new Exception_Exception("faultinfo message",
                                            oue);
}
```

Wraps the fault info.

Top-Down Fault Response

```xml
<S:Fault  xmlns:ns4="http://www.w3.org/2003/05/soap-
  envelope">
    <faultcode>S:Server</faultcode>
    <faultstring>faultinfo message</faultstring>
    <detail>
      <ns2:Exception xmlns:ns2="http://ou/">
        <message>detail message</message>
      </ns2:Exception>
    </detail>
</S:Fault>
```

WebApplicationException

`WebApplicationException` is a runtime exception that can be thrown from any HTTP method to produce an HTTP 4XX or 5XX status response.

```
throw new WebApplicationException()
throw new WebApplicationException(int)
throw new WebApplicationException(Response.Status)
```

Custom `WebApplicationException`

Custom `WebApplicationException` subclasses can be created to produce specific HTTP status codes and response bodies.

```
public class TooManyPlayersException extends
                                    WebApplicationException {
  public TooManyPlayersException() {
    super(
      Response.status(Response.Status.BAD_REQUEST)
        .entity("Try less players")
        .type(MediaType.TEXT_PLAIN)
        .build());
  }
}
```

Jersey Client:

When not getting a `ClientResponse` a `UniformInterfaceException` is thrown.

```
try {
  Client c = Client.create();
  WebResource resource =
  c.resource("http://localhost:7001/app/resources/root");
  String s = resource
    .accept(MediaType.TEXT_PLAIN_TYPE)
    .get(String.class) ;
} catch (UniformInterfaceException ex) {
  ClientResponse response = ex.getResponse();
  int status = response.getResponseStatus();
  // …
}
```

Quiz

In a SOAP message, a <S:Fault> element will always contain:

a. <faultcode>

b. <faultstring>

c. <faultactor>

d. <detail>

Resources

Topic	Website
Handling Exceptions Using SOAP Faults	http://docs.oracle.com/cd/E24329_01/web.1211/e24965/faults.htm
Jersey Client API – Receiving a Response	http://jersey.java.net/nonav/documentation/latest/client-api.html#d4e665

Summary

In this lesson, you should have learned how to:

- Describe how SOAP web services convey errors
- Describe how REST web services convey errors
- Return SOAP faults
- Return HTTP error status codes
- Map thrown exceptions to HTTP status codes
- Handle errors with SOAP clients & Handle errors with Jersey clients

Practice 11 : Overview

This practice covers the following topics:

- JAX-WS Basic Error Handling
- JAX-RS Error Handling