IBM Software Group

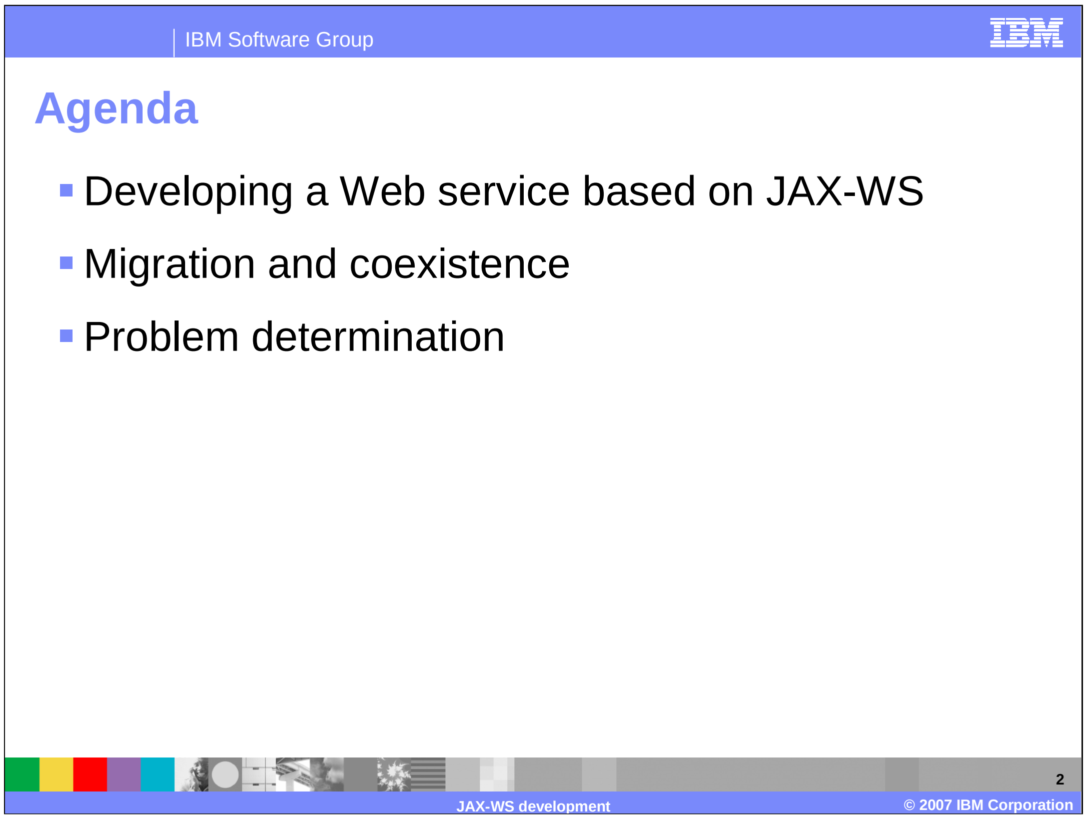
# IBM® WebSphere® Application Server V6.1 Feature Pack for Web services

## *JAX-WS development*

@business on demand.

Updated July 23, 2007

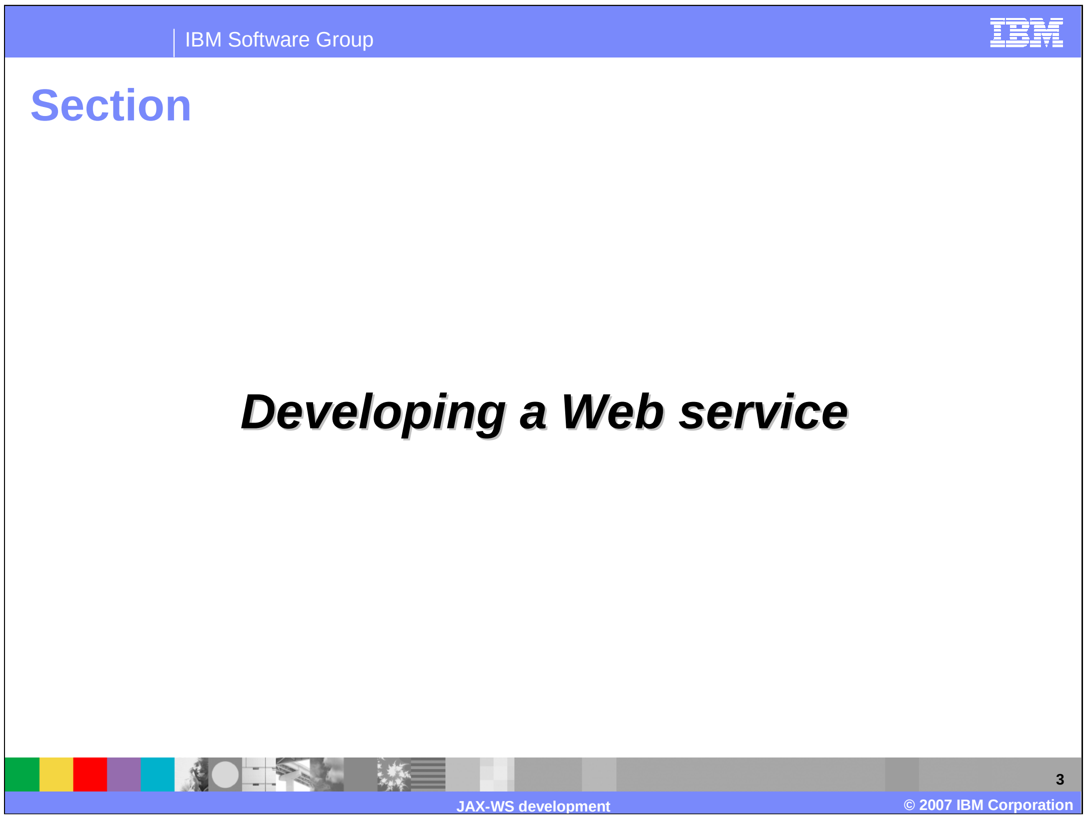
This presentation will provide and understanding of the changes in the development model for JAX-WS.



## Agenda

- Developing a Web service based on JAX-WS
- Migration and coexistence
- Problem determination

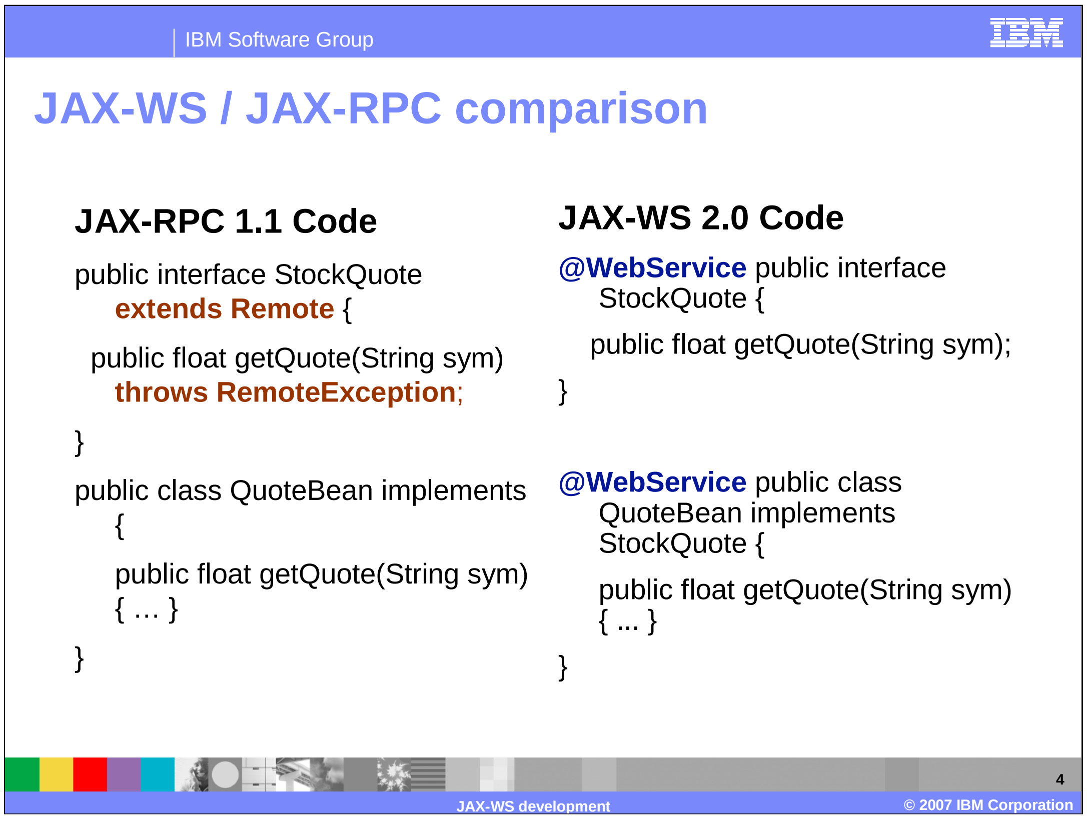
This presentation will begin with information about how to develop a Web service based on JAX-WS, and additional information on migration and coexistence of applications. Next will be a section on problem determination.



## Section

***Developing a Web service***

The next section discusses developing a Web service.



## JAX-WS / JAX-RPC comparison

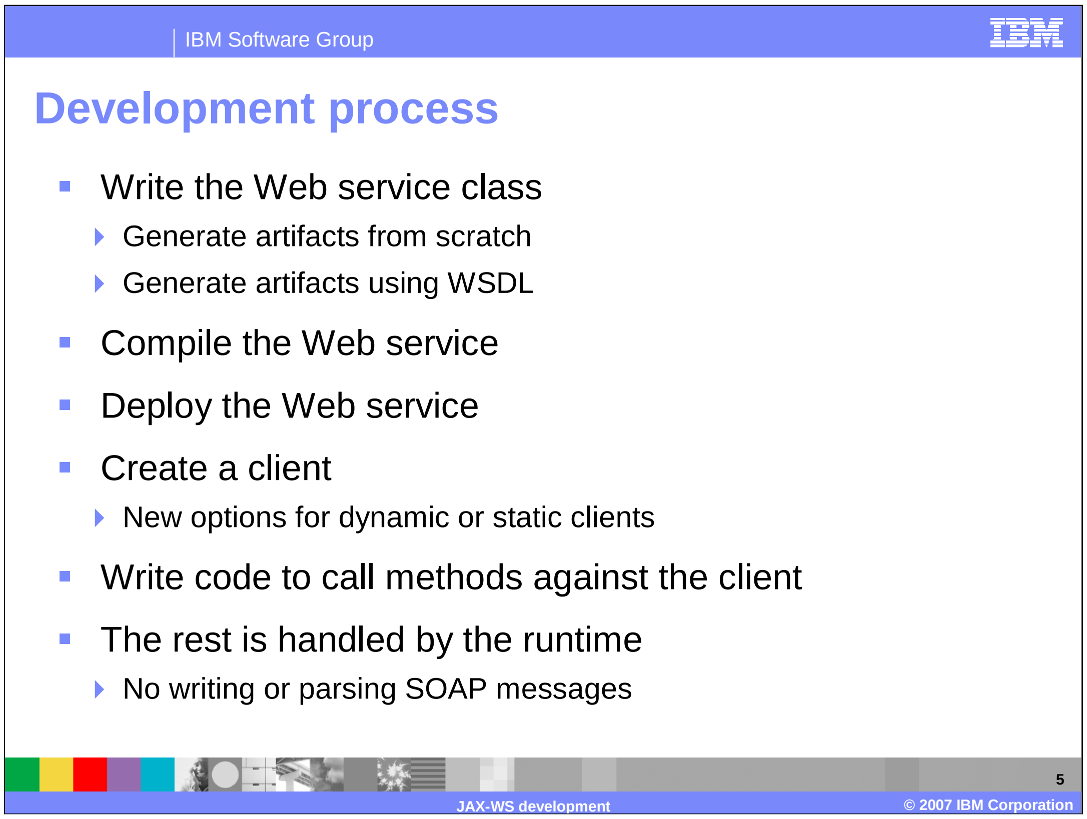
### JAX-RPC 1.1 Code

```
public interface StockQuote
    extends Remote {
  public float getQuote(String sym)
    throws RemoteException;
}
public class QuoteBean implements
    {
    public float getQuote(String sym)
    { … }
}
```

### JAX-WS 2.0 Code

```
@WebService public interface
    StockQuote {
   public float getQuote(String sym);
}

@WebService public class
    QuoteBean implements
    StockQuote {
    public float getQuote(String sym)
    { ... }
}
```

This shows an example of the JAX-WS programming model shown side by side with JAX-RPC. Notice the removal of the remote interface; this is consistent with what is coming in Java™ EE 5 in the EJB support. Add to that the Web service annotation to denote that this will be exposed as a Web service. Similar to the rest of Java EE 5, the service endpoint interface is not required to be specified.



IBM Software Group

## Development process

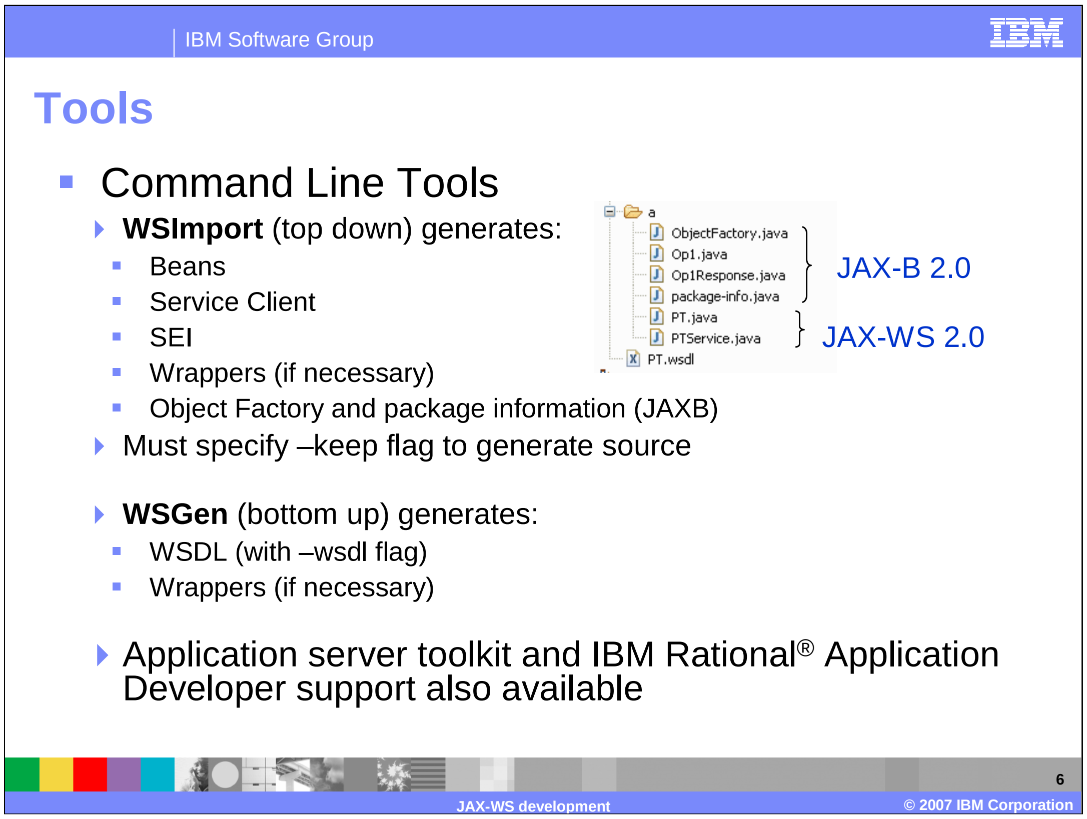
- Write the Web service class
  - Generate artifacts from scratch
  - Generate artifacts using WSDL
- Compile the Web service
- Deploy the Web service
- Create a client
  - New options for dynamic or static clients
- Write code to call methods against the client
- The rest is handled by the runtime
  - No writing or parsing SOAP messages

5

JAX-WS development

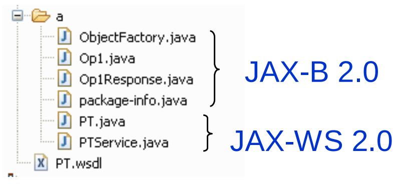
© 2007 IBM Corporation

The development process for creating a Web service has not changed much with the introduction of JAX-WS. Begin by creating a Web service class, either from scratch, or from a WSDL document using WSImport. Next, compile the Web service. Then deploy the Web service to a feature pack for Web services enabled WebSphere Application Server V6.1 server. A proxy that can be used by clients can be created using WSImport, client code can be written to call methods against the proxy. The rest is handled by the runtime, neither the client code nor the provider code is required to write or parse SOAP messages.

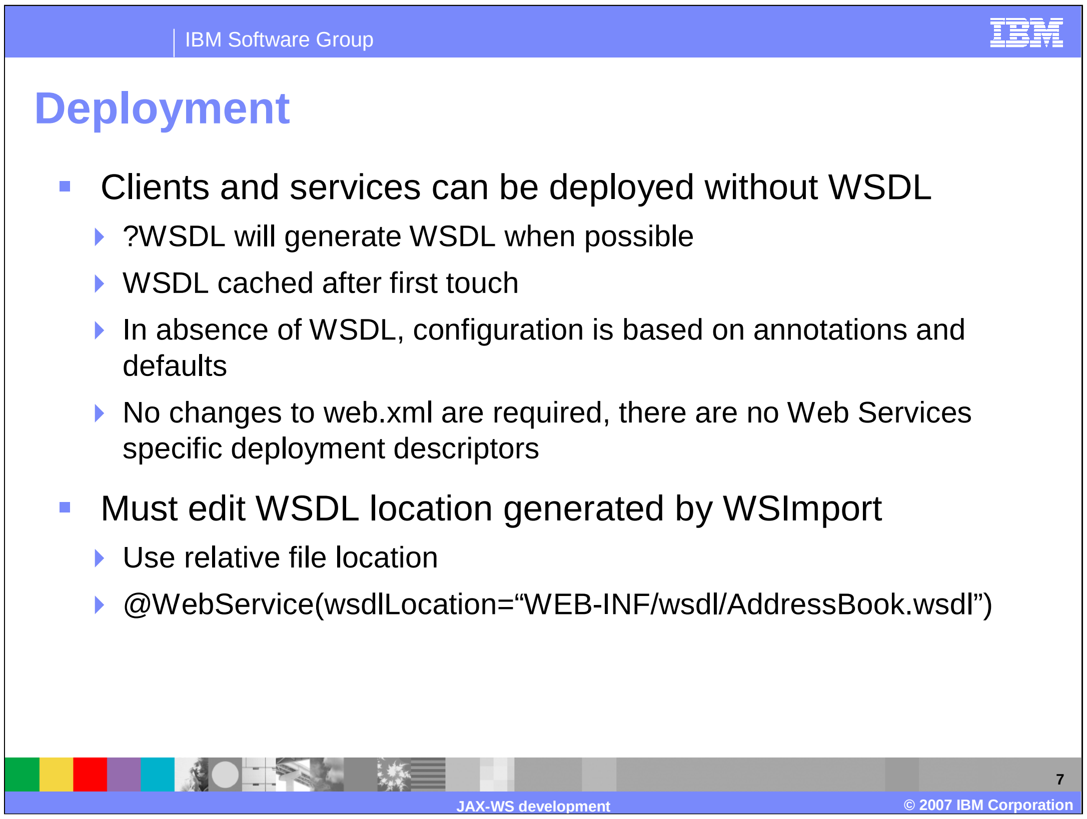


## Tools

- Command Line Tools
  - **WSImport** (top down) generates:
    - Beans
    - Service Client
    - SEI
    - Wrappers (if necessary)
    - Object Factory and package information (JAXB)
  - Must specify –keep flag to generate source
  - **WSGen** (bottom up) generates:
    - WSDL (with –wsdl flag)
    - Wrappers (if necessary)
  - Application server toolkit and IBM Rational® Application Developer support also available



There are two main command line tools for working with JAX-WS to develop Web services. WSImport is a top down development tool that will create the necessary beans, service client, service endpoint interface, and wrappers from a provided WSDL file. To generate source materials specify the –keep flag when running the WSImport command. The WSGen command will create a WSDL document and wrappers when needed from Java code with the proper Web service annotations. There is also support for developing JAX-WS Web services in the Application Server Toolkit and the IBM Rational Application Developer.


IBM Software Group

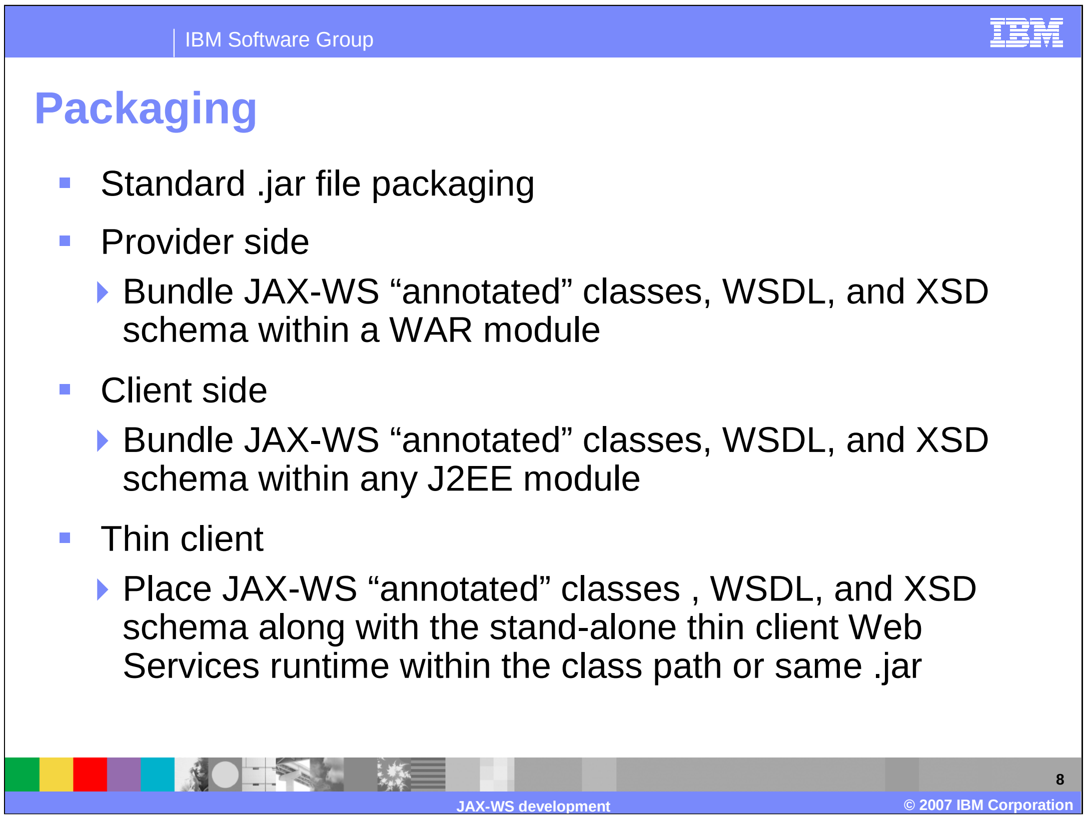
## Deployment

- Clients and services can be deployed without WSDL
  - ?WSDL will generate WSDL when possible
  - WSDL cached after first touch
  - In absence of WSDL, configuration is based on annotations and defaults
  - No changes to web.xml are required, there are no Web Services specific deployment descriptors
- Must edit WSDL location generated by WSImport
  - Use relative file location
  - @WebService(wsdlLocation="WEB-INF/wsdl/AddressBook.wsdl")

7

JAX-WS development © 2007 IBM Corporation

Web services and clients can be deployed with a WSDL document based on the JAX-WS specification. The ?WSDL command will generate a WSDL document when possible. When a Web service is first accessed, a WSDL will be created and cached for future use so that a WSDL does not need to be generated for each subsequent call. If a WSDL is not available, the configuration for the Web Service is based on default values and information stored in the service's annotations. It is important to edit the files generated by WSImport to use a relative file location for the WSDL location annotation.


IBM Software Group

## Packaging

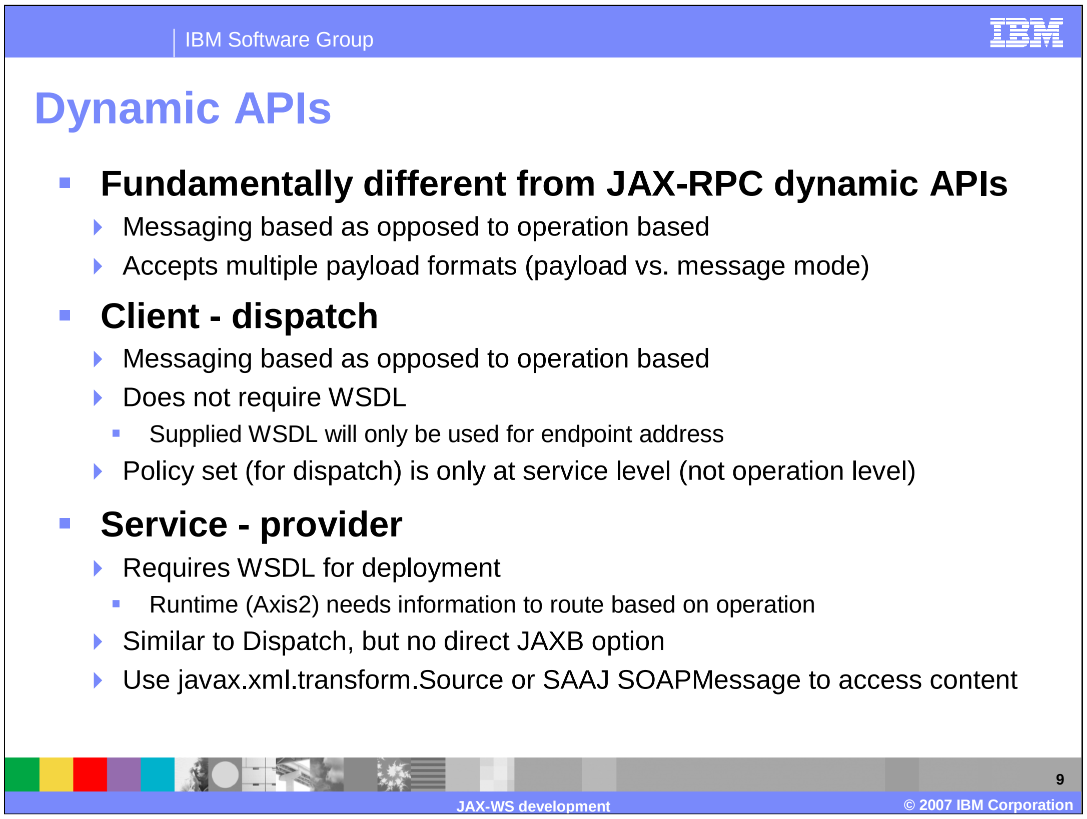
- Standard .jar file packaging
- Provider side
  - Bundle JAX-WS “annotated” classes, WSDL, and XSD schema within a WAR module
- Client side
  - Bundle JAX-WS “annotated” classes, WSDL, and XSD schema within any J2EE module
- Thin client
  - Place JAX-WS “annotated” classes , WSDL, and XSD schema along with the stand-alone thin client Web Services runtime within the class path or same .jar

8

JAX-WS development

© 2007 IBM Corporation

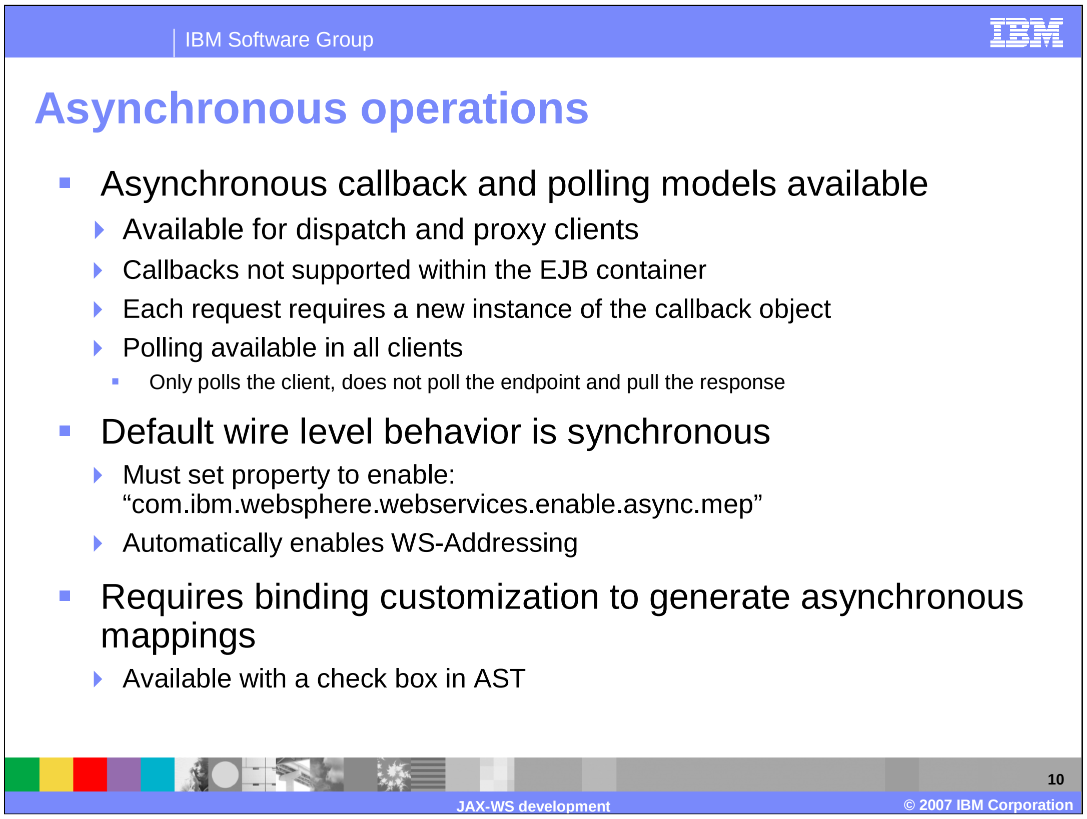
JAX-WS Web services are packaged as simple jar files. On the provider side, the JAX-WS annotated classes are bundled with the WSDL and XSD schema into a WAR module. On the client side, the JAX-WS annotated classes are bundled with the WSDL and XSD schema within a J2EE module. For the thin client, the stand-alone think client Web services runtime must be included.



## Dynamic APIs

- **Fundamentally different from JAX-RPC dynamic APIs**
  - Messaging based as opposed to operation based
  - Accepts multiple payload formats (payload vs. message mode)
- **Client - dispatch**
  - Messaging based as opposed to operation based
  - Does not require WSDL
    - Supplied WSDL will only be used for endpoint address
  - Policy set (for dispatch) is only at service level (not operation level)
- **Service - provider**
  - Requires WSDL for deployment
    - Runtime (Axis2) needs information to route based on operation
  - Similar to Dispatch, but no direct JAXB option
  - Use javax.xml.transform.Source or SAAJ SOAPMessage to access content

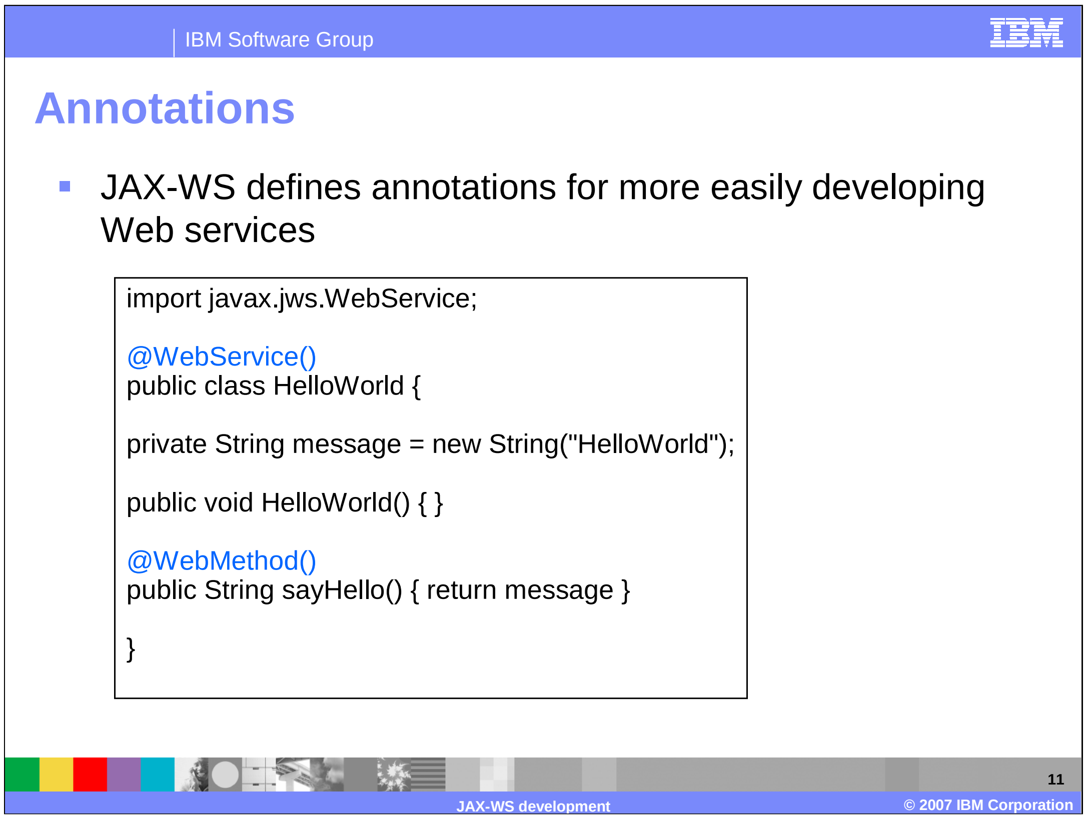
The dynamic APIs provided with JAX-WS are fundamentally different from the JAX-RPC dynamic APIs. The JAX-WS dynamic APIs are message based rather than operation based, and accepts multiple payload formats, a payload and message mode. On the client side, there is no requirement for a WSDL, if a WSDL is supplied it will only be used for the endpoint address. Policy sets can only be applied at the service level and not at a more fine-grained operation level. The provider side does need a WSDL for deployment, as the runtime will use the information to route requests based on the operations that are called.



## Asynchronous operations

- Asynchronous callback and polling models available
  - Available for dispatch and proxy clients
  - Callbacks not supported within the EJB container
  - Each request requires a new instance of the callback object
  - Polling available in all clients
    - Only polls the client, does not poll the endpoint and pull the response
- Default wire level behavior is synchronous
  - Must set property to enable: “com.ibm.websphere.webservices.enable.async.mep”
  - Automatically enables WS-Addressing
- Requires binding customization to generate asynchronous mappings
  - Available with a check box in AST

There are also asynchronous callback and polling models available for the dispatch and proxy based clients. Callbacks are not supported within the EJB container at this time. By default, the wire level behavior is synchronous; the property specified above must be enabled to turn on asynchronous behavior on the wire. This will also enable WS-Addressing by default, as this is required for asynchronous behavior. With the application server toolkit, binding customizations to generate asynchronous mappings can be enabled with a check box.

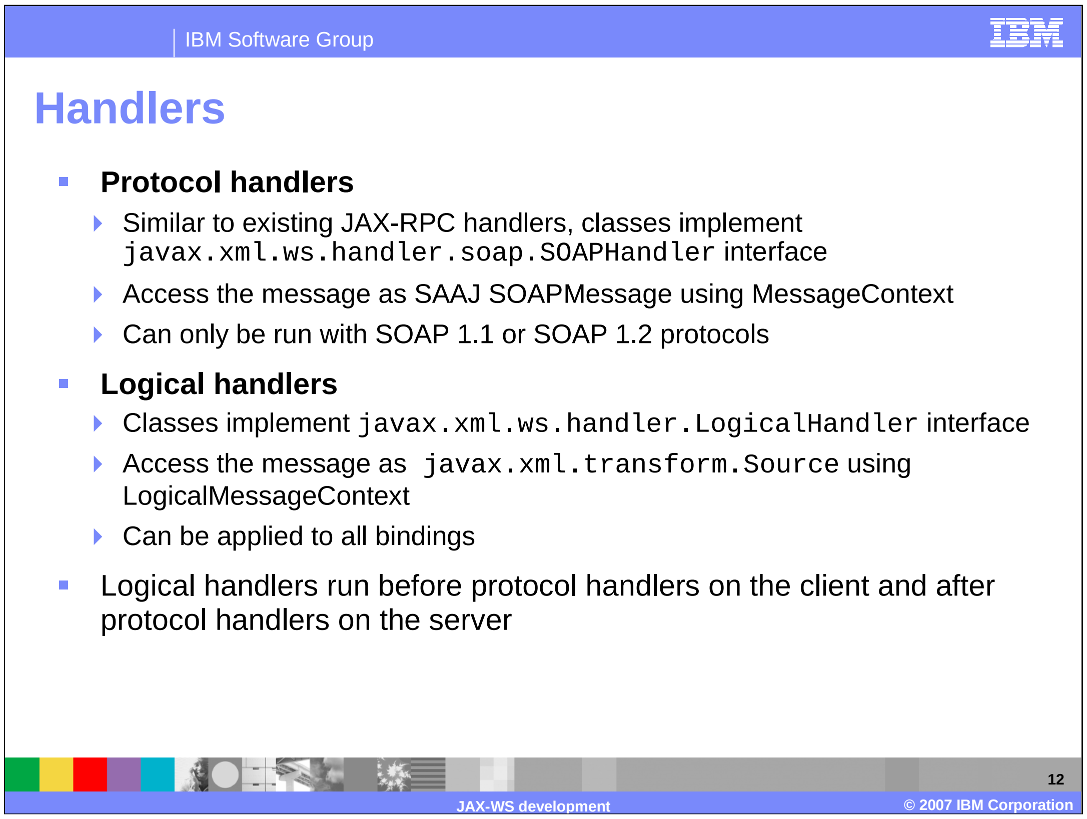


## Annotations

- JAX-WS defines annotations for more easily developing Web services

```
import javax.jws.WebService;

@WebService()
public class HelloWorld {

private String message = new String("HelloWorld");

public void HelloWorld() { }

@WebMethod()
public String sayHello() { return message }

}
```

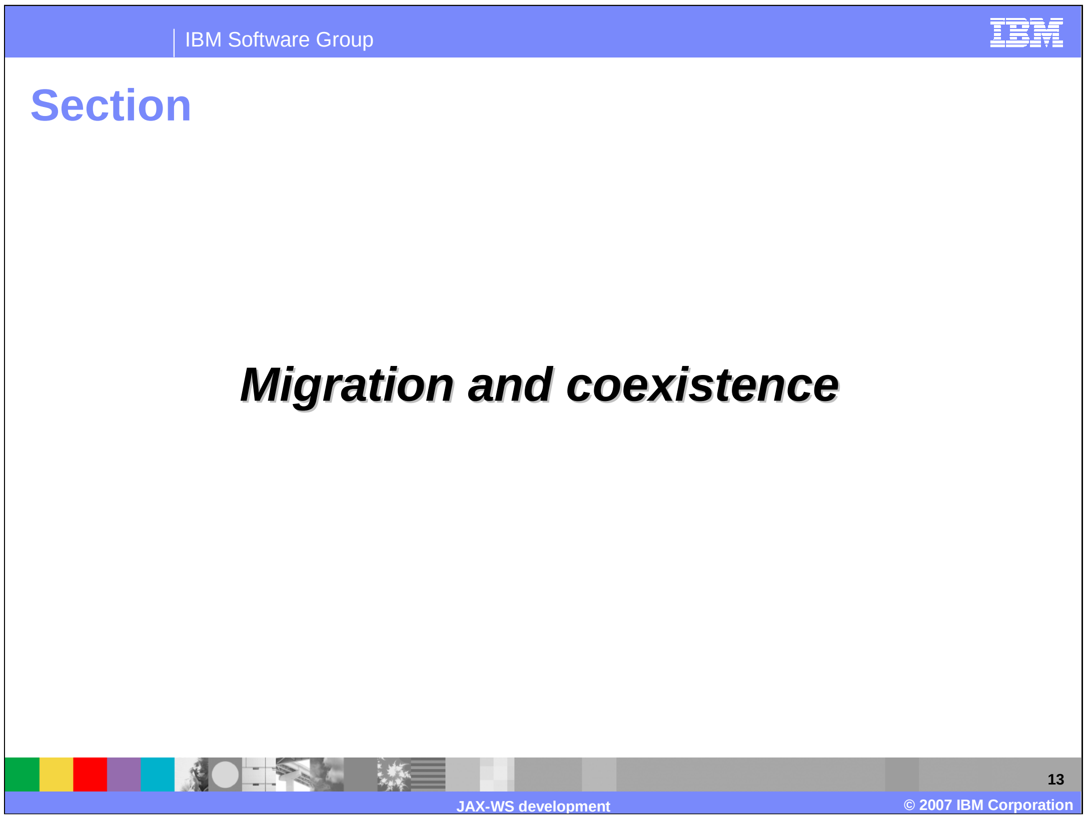
Some of the annotations used come from JSR-181 rather than the JAX-WS specification. JAX-WS defines its own set of annotations to cover the portions that are not addressed in JSR-181. When combined they allow a developer to specify configuration data within the source files. This can simplify the development process.



## Handlers

- **Protocol handlers**
  - Similar to existing JAX-RPC handlers, classes implement `javax.xml.ws.handler.soap.SOAPHandler` interface
  - Access the message as SAAJ SOAPMessage using MessageContext
  - Can only be run with SOAP 1.1 or SOAP 1.2 protocols
- **Logical handlers**
  - Classes implement `javax.xml.ws.handler.LogicalHandler` interface
  - Access the message as `javax.xml.transform.Source` using LogicalMessageContext
  - Can be applied to all bindings
- Logical handlers run before protocol handlers on the client and after protocol handlers on the server

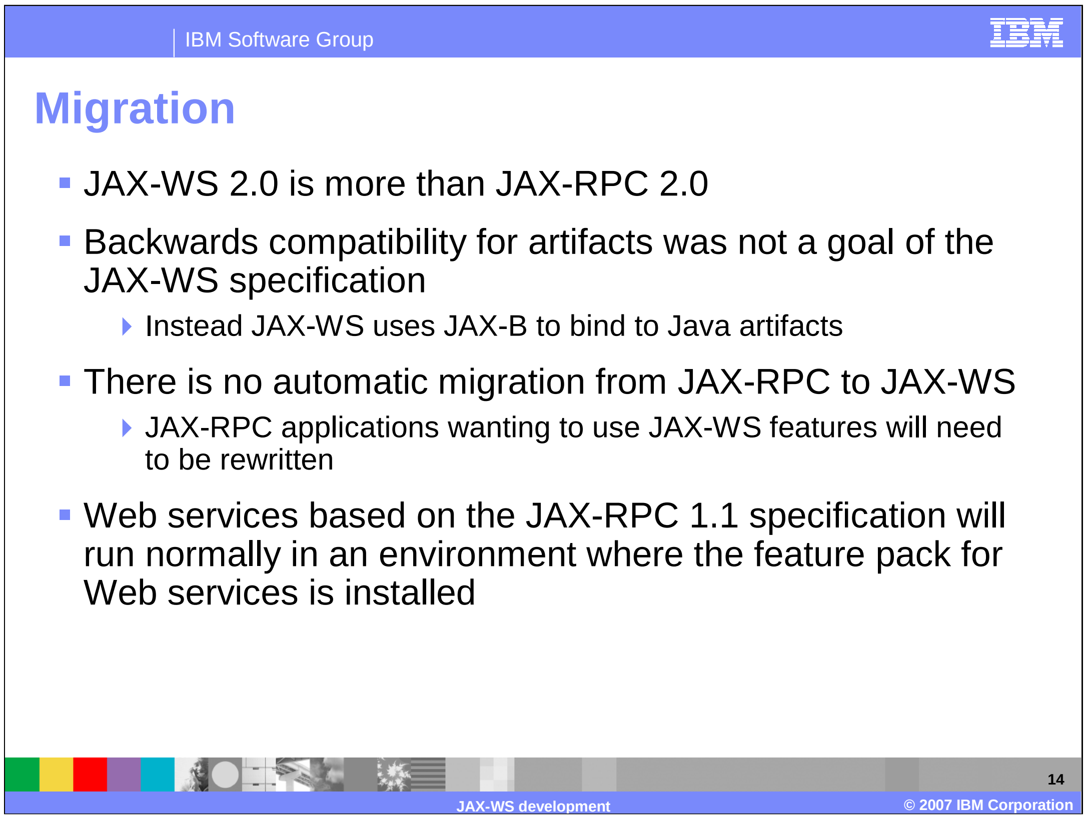
The JAX-WS specification details support for handlers is similar to JAX-RPC. The Feature Pack for Web services supports two types of handlers for JAX-WS, protocol handlers and logical handlers. The protocol handler is similar to JAX-RPC handlers, and accesses the message as a SAAJ base SOAPMessage with the MessageContext. Logical handlers use a LogicalMessageContext to access the message instead, and can be applied to all binding types. Logical handlers are designed to run before protocol handlers on the clients and after protocol handlers on the provider side.



## Section

***Migration and coexistence***

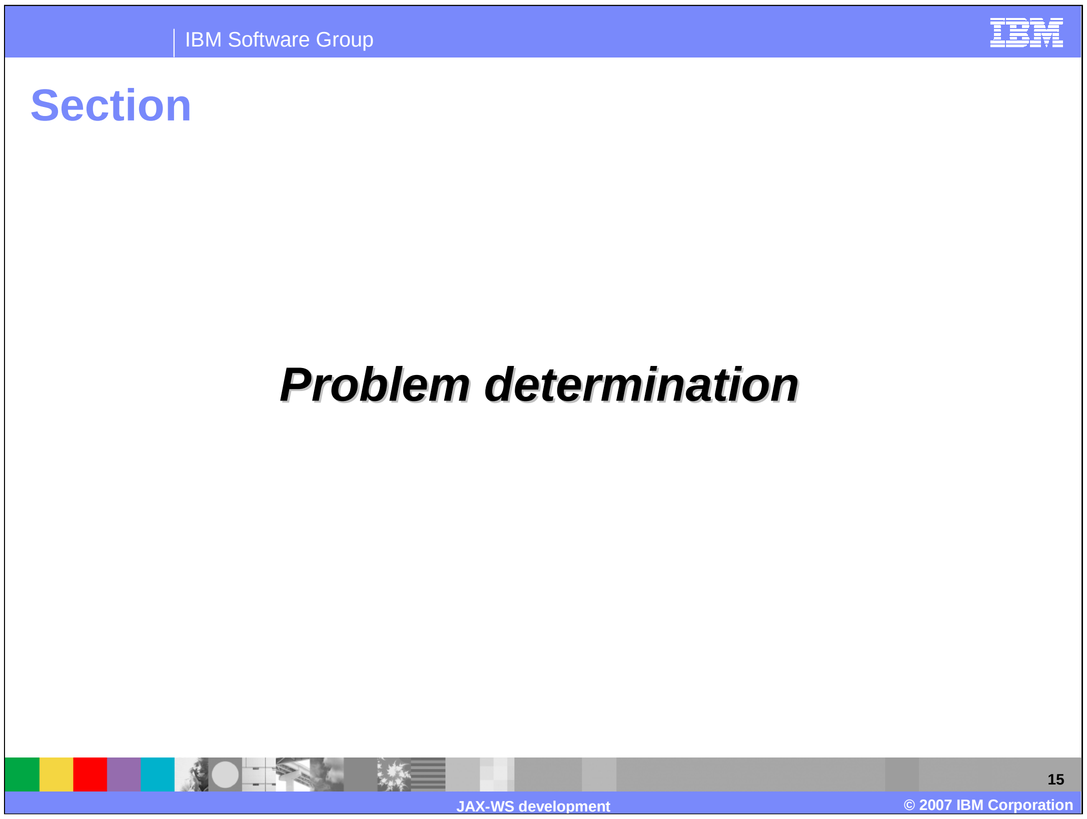
The next section explains migration and coexistence of JAX-RPC and JAX-WS Web services.



## Migration

- JAX-WS 2.0 is more than JAX-RPC 2.0
- Backwards compatibility for artifacts was not a goal of the JAX-WS specification
  - Instead JAX-WS uses JAX-B to bind to Java artifacts
- There is no automatic migration from JAX-RPC to JAX-WS
  - JAX-RPC applications wanting to use JAX-WS features will need to be rewritten
- Web services based on the JAX-RPC 1.1 specification will run normally in an environment where the feature pack for Web services is installed

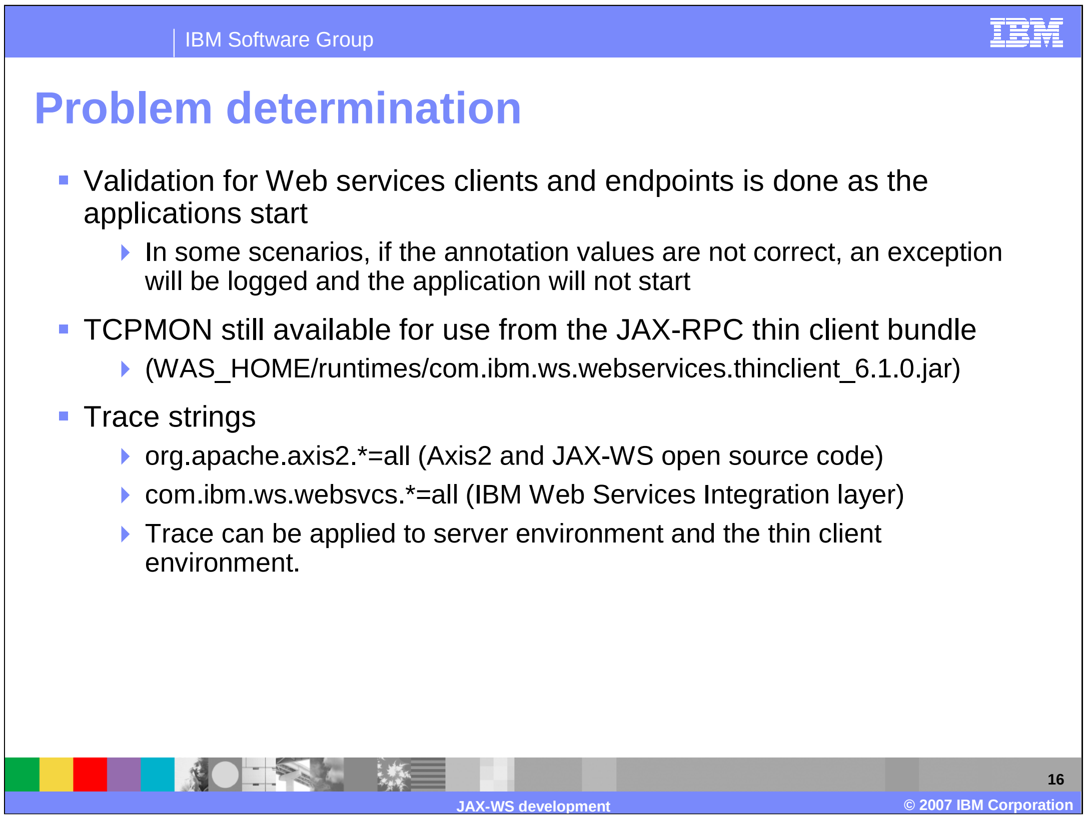
The JAX-WS specification was not designed with backwards compatibility of artifacts as a goal. Due to this, there is no automatic or simple migration from JAX-RPC to JAX-WS. JAX-RPC applications that need to use these new technologies will have to be rewritten based on the JAX-WS specification. In an environment with the feature pack for Web services installed, JAX-RPC applications will still function normally, but cannot take advantage of the new JAX-WS based features.



## Section

### *Problem determination*

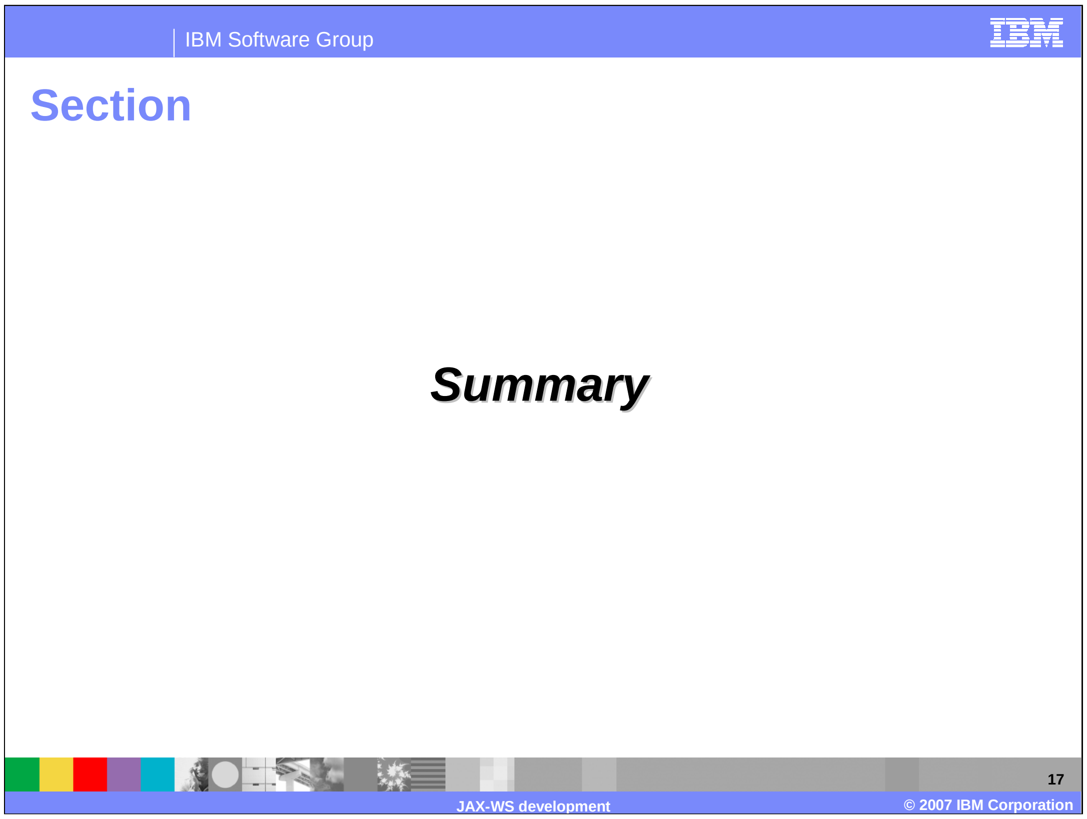
The next section discusses problem determination of a JAX-WS application.



## Problem determination

- Validation for Web services clients and endpoints is done as the applications start
  - In some scenarios, if the annotation values are not correct, an exception will be logged and the application will not start
- TCPMON still available for use from the JAX-RPC thin client bundle
  - (WAS_HOME/runtimes/com.ibm.ws.webservices.thinclient_6.1.0.jar)
- Trace strings
  - org.apache.axis2.*=all (Axis2 and JAX-WS open source code)
  - com.ibm.ws.websvcs.*=all (IBM Web Services Integration layer)
  - Trace can be applied to server environment and the thin client environment.

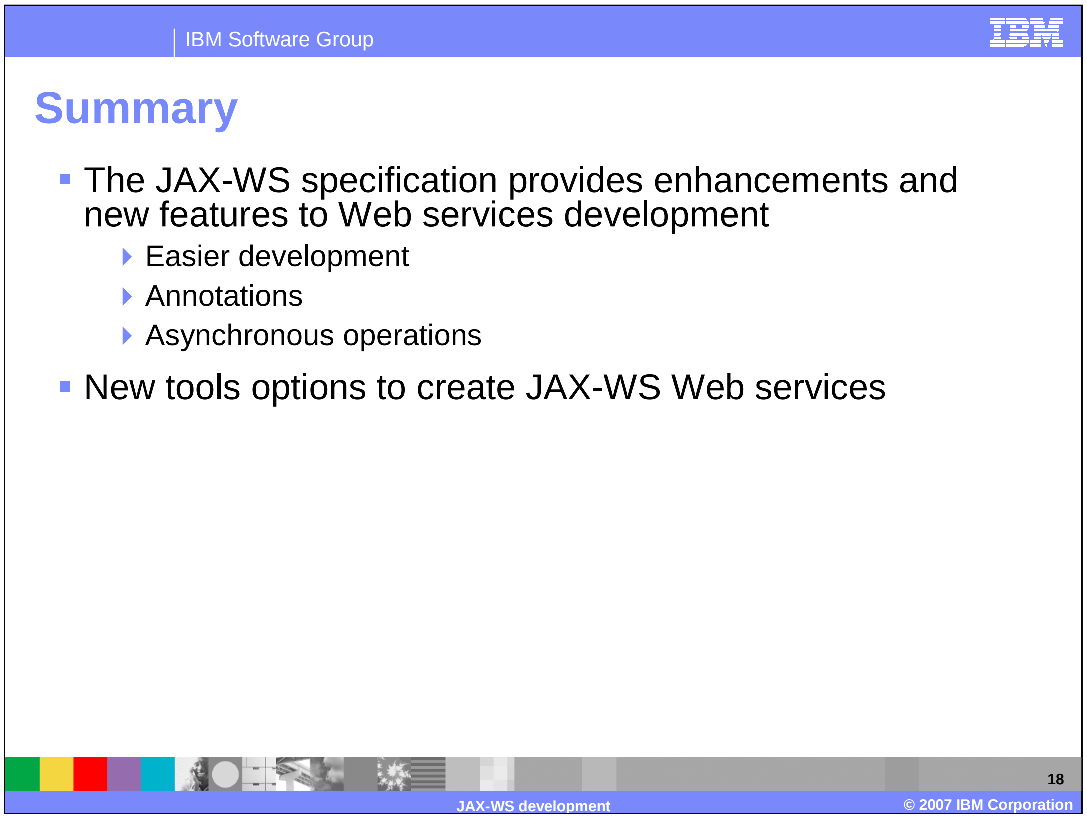
Certain considerations should be taken when diagnosing problems related to JAX-WS based Web services. When a client or service written with JAX-WS annotations is started, those annotations will be validated by the runtime. If there are problems with the form or content of the annotations the application may not start properly and errors may be logged. A TCP monitoring tool can be used with JAX-WS services to view the contents of the messages being sent, similar to JAX-RPC. Specific trace strings that can be used to gather detailed information about JAX-WS problems are shown above.



## Section

**_Summary_**

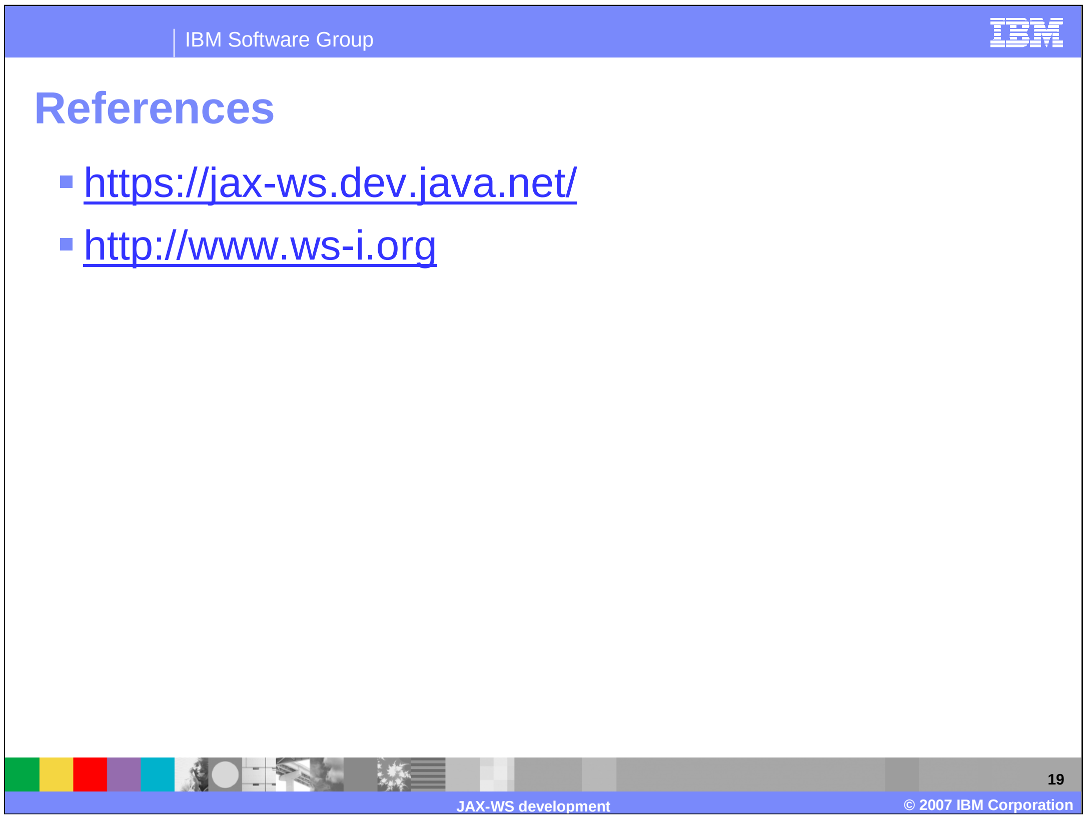
Following is the summary of the presentation.



## Summary

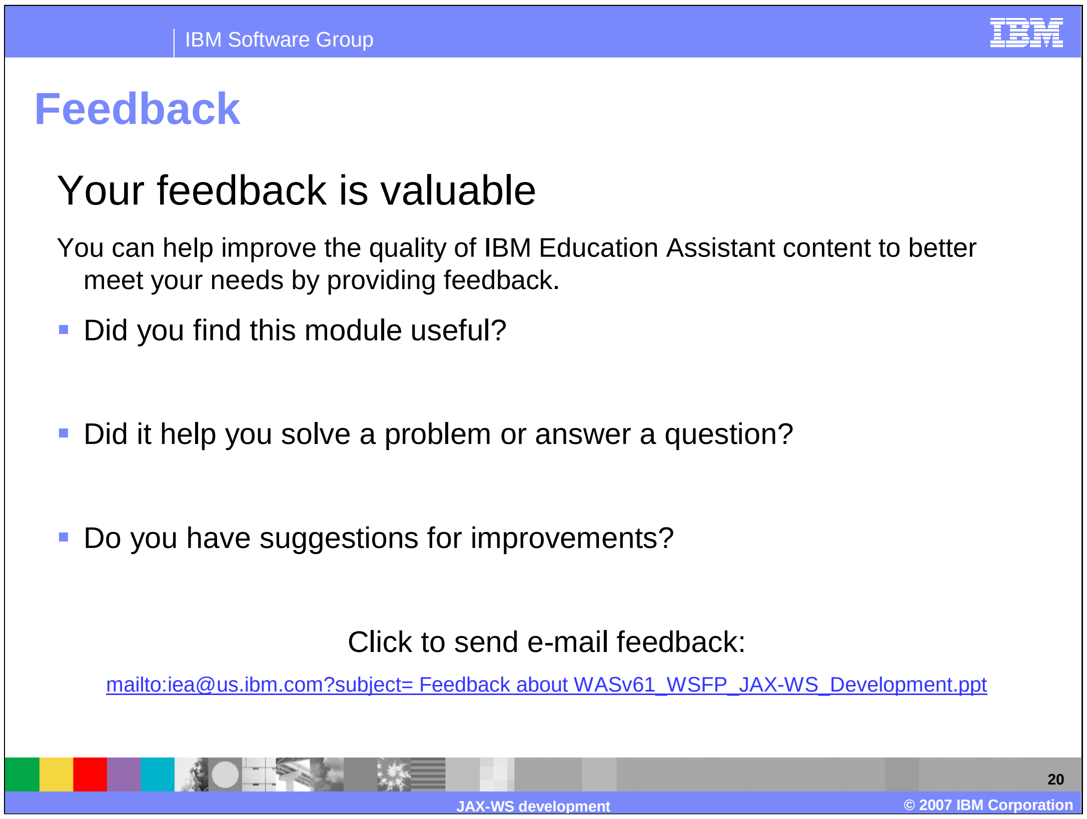
- The JAX-WS specification provides enhancements and new features to Web services development
  - Easier development
  - Annotations
  - Asynchronous operations
- New tools options to create JAX-WS Web services

The new JAX-WS specification provides key new features for developing Web services, such as an annotation based programming model and support for asynchronous operations.  The Feature Pack for Web services also provides a number of new tools options for creating JAX-WS based Web services.



## References

- https://jax-ws.dev.java.net/
- http://www.ws-i.org



## Feedback

### Your feedback is valuable

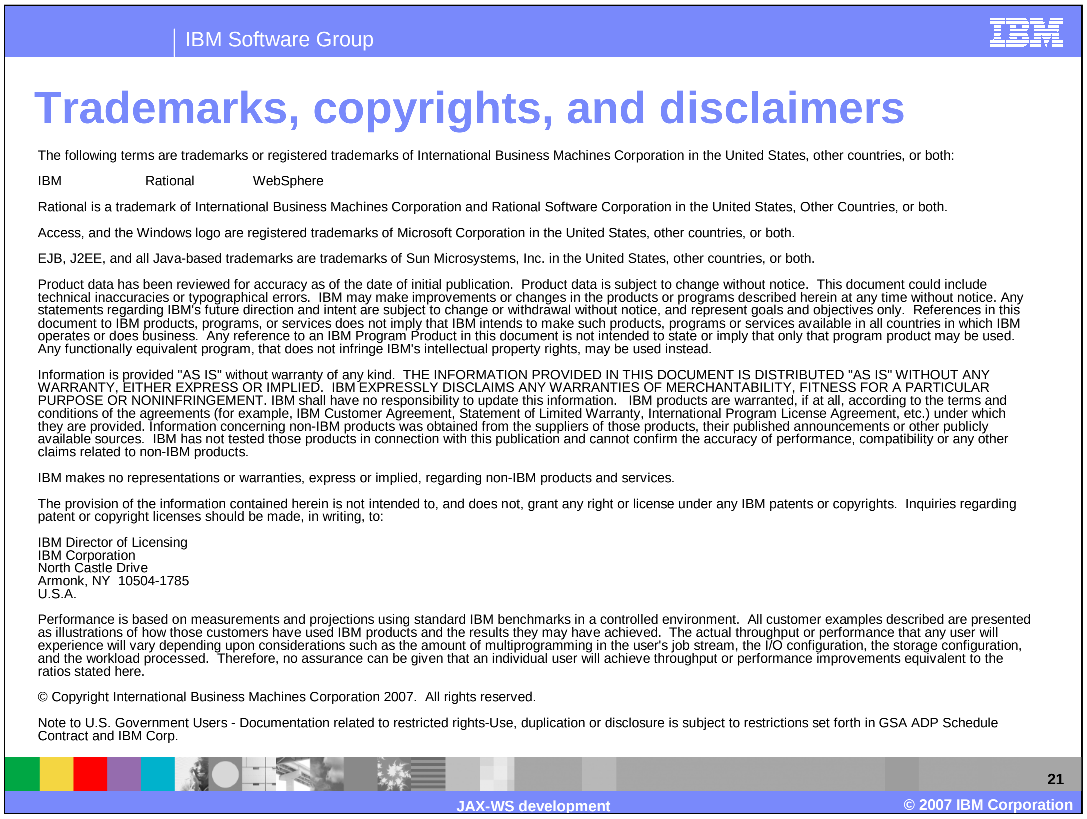
You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject= Feedback about WASv61_WSFP_JAX-WS_Development.ppt

You can help improve the quality of IBM Education Assistant content by providing feedback.



## Trademarks, copyrights, and disclaimers