



Introducción a los Servicios Web en Java



Siguiente Parte >>

(/articulo/introduccion_a_los_servicios_web_en_java_190/2)




Seguro que habrás oído el bombardeo publicitario, y probablemente estarás aturdido con todos los acrónimos. Por eso ¿qué son los Servicios Web y Cómo podemos utilizarlos?. Este tutorial intentará desmitificar los Servicios Web y mostrará, paso a paso, cómo construirlos, desplegarlos, usarlos y encontrarlos.

Los Servicios Web básicos no son muy difíciles de crear. Para probar este punto, te mostraremos, en esta primera página, como construir un Servicio Web en unos 30 minutos. En las siguientes páginas profundizaremos algo más en los Servicios web y explicaremos los siguientes tópicos en más detalle:

- La mensajería SOAP.
- Las definiciones WSDL y su relación con el código.
- Publicación de Servicios en un directorio UDDI.
- Exponer aplicaciones legales como Servicios Web.
- Tópicos avanzados, como la seguridad.





Hosting para aplicaciones JAVA

JVM privada desde 6,90€/Mes

Hosting específico para aplicaciones
JAVA. Con garantía de reembolso 30
días para probarlo

anw.es

ABRIR

En esta primera página, empezaremos con una definición programática de los Servicios Web, luego mostraremos una simple clase Java que llama y ejecuta un Servicio Web. Hemos creado nuestros ejemplos usando un conjunto de herramientas gratuitas de **Systinet** (los detalles de cómo acceder y descarga este software están en la sección **Instalar el Software**). No tienes porqué utilizar estos productos para entender los ejemplos, pero te lo recomendamos. Los conceptos presentados y el código que hemos creado se puede aplicar de forma general y es relativamente independiente de las herramientas utilizadas. Asumimos algún conocimiento de XML pero ninguno de Servicios Web.

Creemos que J2EE es la arquitectura más madura para la implementación de la lógica de negocios, y nuestro objetivo es presentar los Servicios Web como una extensión natural del modelo de componentes existente en J2EE, proporcionando un protocolo basado en XML como estándar industrial. Esto le da a los sistemas existentes basados en J2EE mucho más alcance que antes y hace de J2EE una mucho mejor opción para la implementación de la lógica de negocio principal dentro de los entornos típicamente heterogéneos de sistemas de información corporativa.

■



El Servicio Web - una Definición Programática

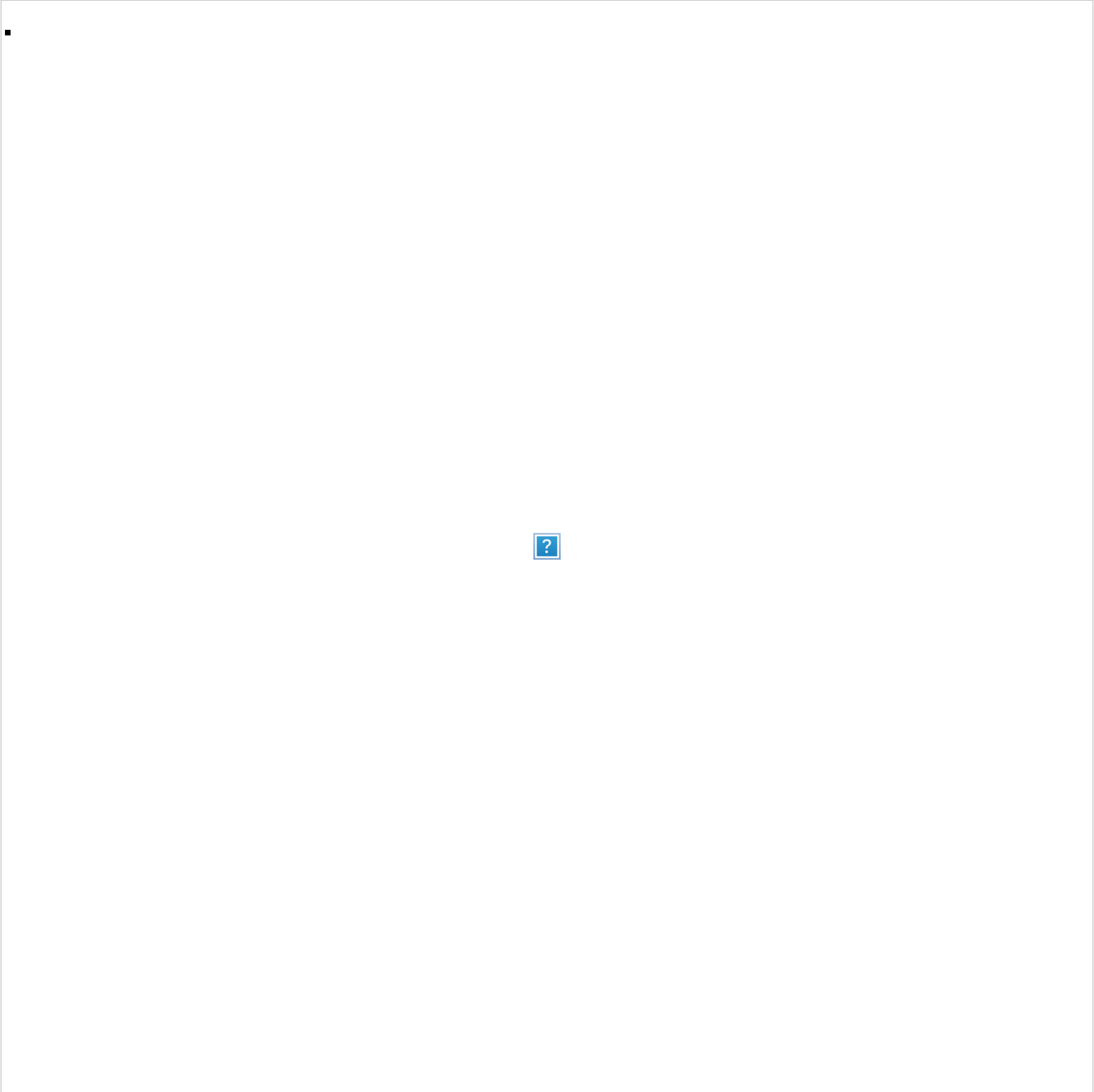
Un Servicio Web es un componente software con las siguientes características:

- Es accesible a través del interface **SOAP** (<https://www.w3.org/TR/SOAP/>) (Simple Object Access Protocol).
- Su interface se describe en un documento **WSDL** (<https://www.w3.org/TR/wsdl>) (Web Service Description Language).

SOAP es un protocolo de mensajería XML extensible que forma la base de los Servicios Web. SOAP proporciona un mecanismo simple y consistente que permite a una aplicación enviar mensajes XML a otra aplicación. Un mensaje SOAP es una transmisión de una-vía desde un emisor SOAP a un receptor SOAP, y cualquier aplicación puede participar en este intercambio como emisor o receptor. Los mensajes SOAP se pueden combinar para soportar muchos comportamientos de comunicación, incluyendo, solicitud/respuesta, respuesta solicitada, mensajería asíncrona de una-vía, o incluso notificación. SOAP es un

protocolo de alto nivel que sólo define la estructura del mensaje y unas pocas reglas para su procesamiento. Es completamente independiente del protocolo de transporte subyacente, por eso los mensajes SOAP se pueden intercambiar sobre HTTP, JMS o protocolos de transporte de e-mail. Actualmente el protocolo HTTP es el más utilizado para los mensajes HTTP.

WSDL es un documento XML que contiene un conjunto de definiciones que describen un Servicio Web. Proporciona toda la información necesaria para acceder y utilizar un Servicio Web. Un documento WSDL describe qué hace el Servicio Web, cómo se comunica, y dónde reside. Usamos el documento WSDL en el momento del despliegue para crear nuestros interfaces de servicio. Algunas implementaciones SOAP, incluyendo WASP de Systinet, también usan WSDL durante la ejecución para soportar comunicaciones dinámicas.



Instalar el Software

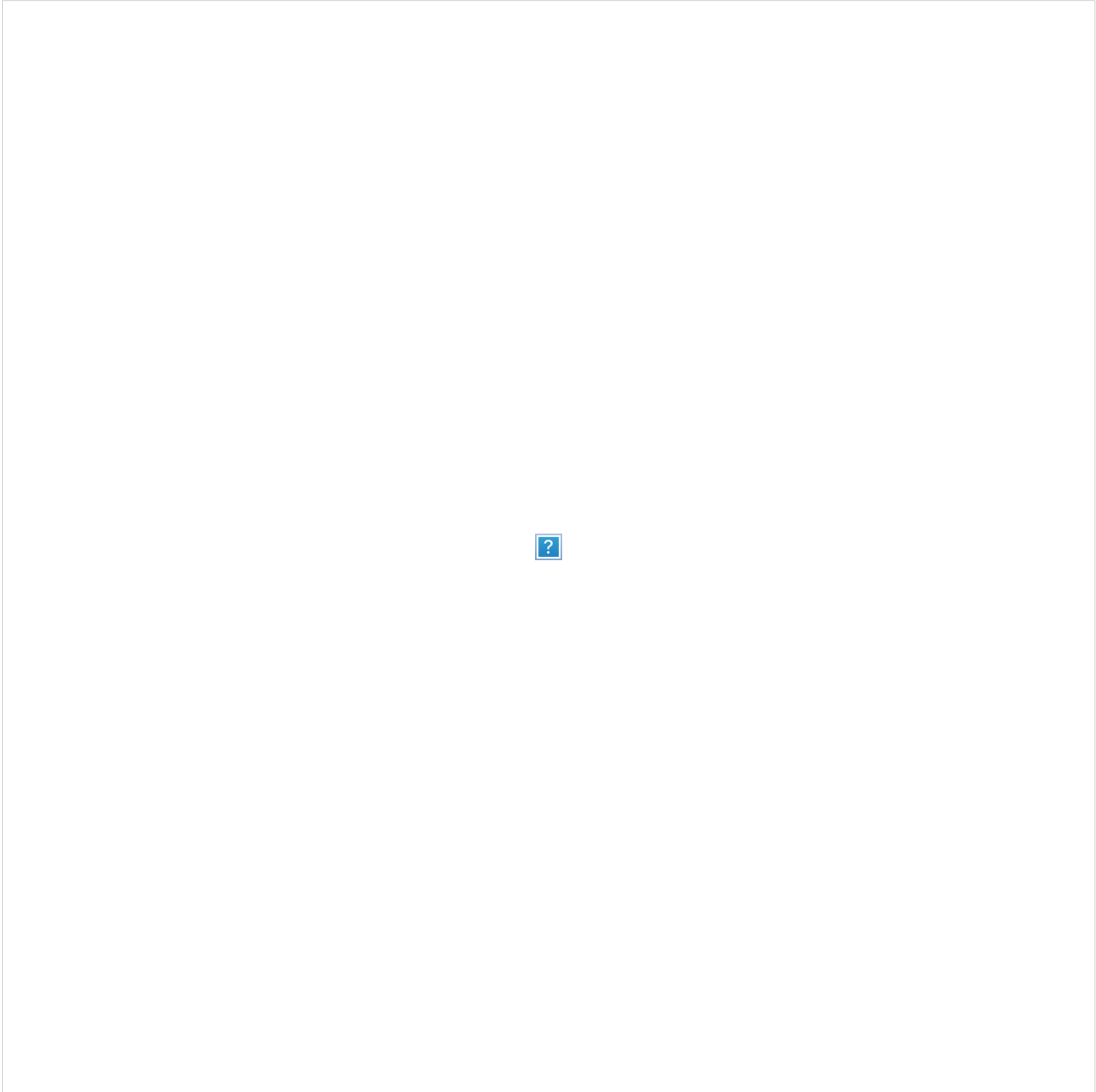
REQUERIMIENTOS: *Asumimos que tienes instalada una versión del SDK superior a la 1.3,*

y que dispones de un navegador HTTP estándar. La variable de entorno `JAVA_HOME` debería apuntar al directorio de instalación del SDK.

Si quieres probar los ejemplos, del tutorial, necesitarás descargar **WASP Advanced** (http://www.systinet.com/products/wasp_advanced/index.html) de Systinet. Descomprime el paquete descargado en tu disco duro (preferiblemente en `C :`) y ejecuta el script `install` del subdirectorio `bin` de la instalación de WASP Advanced.

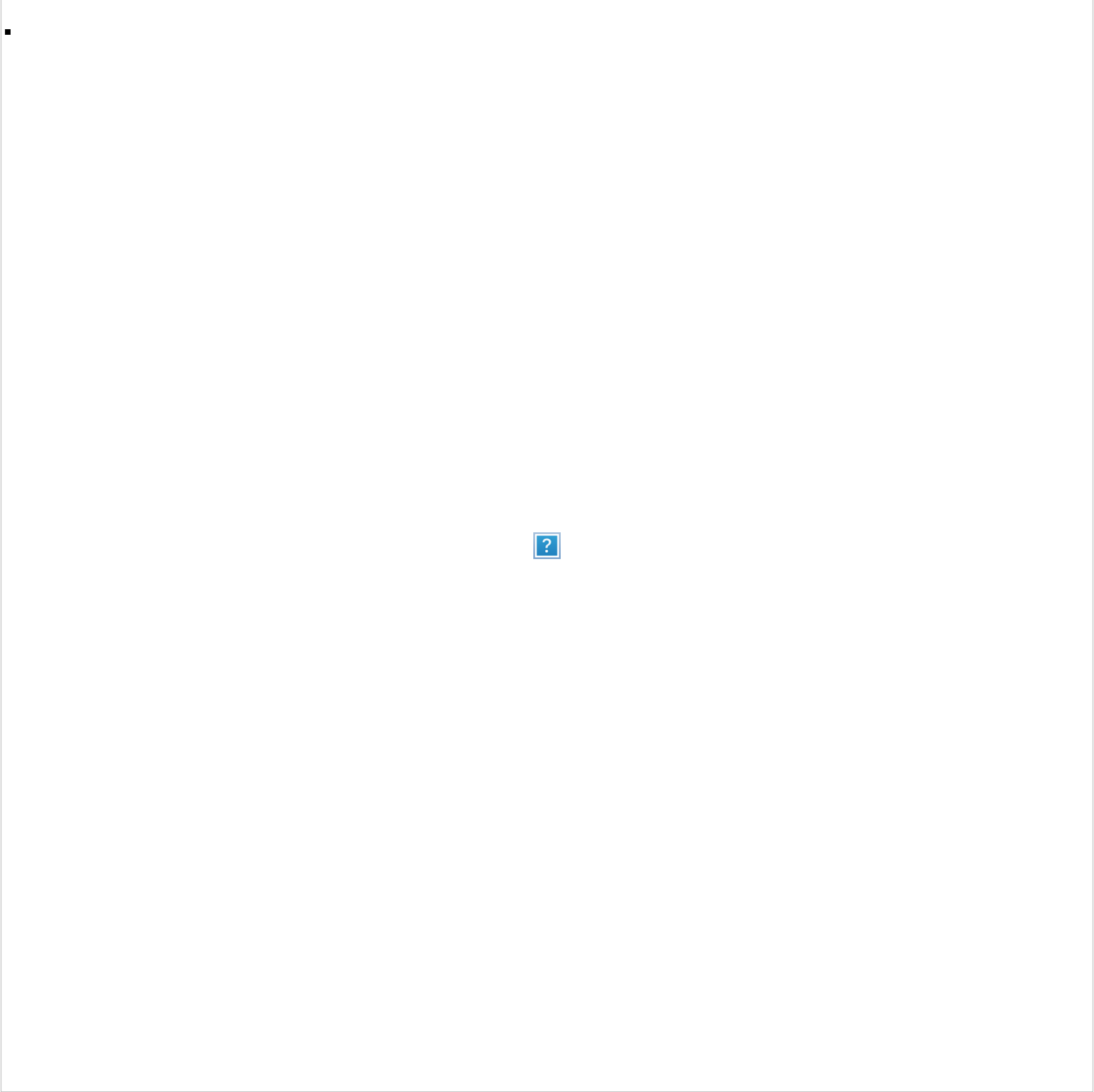
En nuestros ejemplos asumimos que has desempaquetado WASP en el directorio `c : \wasp-advanced`.

También necesitarás descargar el



código fuente (/cursos_descargas/servic_web/classes/1.zip) y descomprimirlo en el directorio `c : \wasp_demo`.

Si eliges nombres diferentes de directorios, por favor, actualiza el script `env.bat` de la forma apropiada (cambia las variables de entorno `WASP_HOME` y `WASP_DEMO` para que apunten respectivamente al directorio de instalación de WASP y al directorio de la demo).



Implementar un Sencillo Servicio Web

Seguiremos estos pasos para crear nuestro sencillo Servicio Web:

- **Crear la lógica del negocio del Servicio Web.** Primero necesitamos escribir una clase Java que implemente la lógica de negocio del servicio Web. En este caso, nuestra lógica de negocio será un simple clase Java que simula una servicio de citas de stocks.
- **Desplegar la clase Java en el servidor SOAP.** Luego necesitamos convertir la clase Java en un Servicio Web. Mostraremos como desplegar la clase Java en un servidor SOAP usando la heramienta de despliegue de WASP.
- **Generar las clases de acceso para los clientes.** Una aplicación cliente usa un

objeto proxy para acceder al Servicio Web. En el momento de la solicitud, el proxy acepta una llamada a un método Java desde la aplicación y la traduce en un mensaje XML. En el momento de la respuesta, el proxy recibe el mensaje de respuesta SOAP, lo traduce en objetos Java y devuelve el resultado a la aplicación cliente.

- **Desarrollar la aplicación cliente.** La aplicación cliente trata el proxy como un objeto Java estándar que facilita la comunicación con un Servicio Web.

Empecemos con una simple clase Java que implementa la función de búsqueda de un servidor de Stocks:

```
/*
 * StockQuoteService.java
 *
 * Created on Sat 13th Oct 2001, 15:25
 */

package com.systinet.demos.stock;

/**
 * Simple stock quote service
 * @author zdenek
 * @version 1.0
 */
public class StockQuoteService {

    public double getQuote(String symbol) {
        if(symbol!=null && symbol.equalsIgnoreCase("SUNW"))
            return 10;
        if(symbol!=null && symbol.equalsIgnoreCase("MSFT"))
            return 50;
        if(symbol!=null && symbol.equalsIgnoreCase("BEAS"))
            return 11;
        return 0;
    }

    public java.util.LinkedList getAvailableStocks() {
        java.util.LinkedList list = new java.util.LinkedList();
        list.add("SUNW");
        list.add("MSFT");
        list.add("BEAS");
        return list;
    }

}
```

Nuestro ejemplo es el tan manido sistema servidor de stocks (ya hemos visto muchos de estos, los desarrolladores deberían pagar un impuesto por su utilización), pero ilustra lo fácil que se puede crear y desplegar un servicio Web. En nuestro ejemplo, vamos a recuperar los precios de tres stocks (BEAS, MSFT, y SUNW).

La forma más fácil de convertir nuestra clase en un servicio Web es compilar nuestras clases y luego usar la herramienta de despliegue para desplegarlas en el entorno de ejecución de Servicio Web.

Primero arrancamos el entorno de Servicios Web con el script `startserver.bat`. Luego compilamos `StockQuoteService.java` y desplegamos la clase compiladas al servidor SOAP usando el script `deploy.bat`.

Luego nos aseguramos de que todo funciona correctamente abriendo la consola de administración introduciendo está dirección en nuestro navegador

`http://localhost:6060/admin/console`. Pulsamos sobre el botón **Refresh** para ver una lista de todos los paquetes desplegados en el servidor. Deberíamos ver el paquete `StockService` con un `StockQuoteService` desplegado en el servidor. Observa que el entorno de ejecución de Servicios Web ha generado automáticamente el fichero WSDL y lo publica en la dirección **`http://localhost:6060/StockQuoteService/`**.

Aquí tenemos el fichero WSDL generado automáticamente:

```
<?xml version='1.0'?>
<wsdl:definitions name='com.systinet.demos.stock.StockQuoteService'
targetNamespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stock/'
  xmlns:mime='http://schemas.xmlsoap.org/wsdl/mime/'
  xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:ns0='http://idoox.com/containers'
  xmlns:http='http://schemas.xmlsoap.org/wsdl/http/'
  xmlns:tns='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stock/'
  xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'>
  <wsdl:message name='StockQuoteService_getQuote_Request'>
    <wsdl:part name='p0' type='xsd:string'/>
  </wsdl:message>
  <wsdl:message name='StockQuoteService_getQuote_Response'>
    <wsdl:part name='response' type='xsd:double'/>
  </wsdl:message>
  <wsdl:message name='StockQuoteService_getAvailableStocks_Request'/>
  <wsdl:message name='StockQuoteService_getAvailableStocks_Response'>
    <wsdl:part name='response' type='ns0:LinkedList'/>
  </wsdl:message>
  <wsdl:portType name='StockQuoteService'>
    <wsdl:operation name='getAvailableStocks'>
      <wsdl:input name='getAvailableStocks'
        message='tns:StockQuoteService_getAvailableStocks_Request'/>
      <wsdl:output name='getAvailableStocks'
        message='tns:StockQuoteService_getAvailableStocks_Response'/>
    </wsdl:operation>
    <wsdl:operation name='getQuote' parameterOrder='p0'>
      <wsdl:input name='getQuote'
        message='tns:StockQuoteService_getQuote_Request'/>
      <wsdl:output name='getQuote'
```



```

        message='tns:StockQuoteService_getQuote_Response' />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name='StockQuoteService' type='tns:StockQuoteService'>
    <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='rpc
' />
    <wsdl:operation name='getAvailableStocks'>
        <soap:operation soapAction='' style='rpc' />
        <wsdl:input name='getAvailableStocks'>
            <soap:body use='encoded' encodingStyle='http://schemas.xmlsoap.or
g/soap/encoding/'
            namespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demo
s/stock/' />
        </wsdl:input>
        <wsdl:output name='getAvailableStocks'>
            <soap:body use='encoded' encodingStyle='http://schemas.xmlsoap.or
g/soap/encoding/'
            namespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demo
s/stock/' />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name='getQuote'>
        <soap:operation soapAction='' style='rpc' />
        <wsdl:input name='getQuote'>
            <soap:body use='encoded' encodingStyle='http://schemas.xmlsoap.or
g/soap/encoding/'
            namespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demo
s/stock/' />
        </wsdl:input>
        <wsdl:output name='getQuote'>
            <soap:body use='encoded' encodingStyle='http://schemas.xmlsoap.or
g/soap/encoding/'
            namespace='http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demo
s/stock/' />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name='JavaService'>
    <wsdl:port name='StockQuoteService' binding='tns:StockQuoteService'>
        <soap:address location='http://localhost:6061/StockQuoteService/' />
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

El fichero WSDL contiene una descripción total del Servicio Web desplegado. Básicamente hay tres partes en un fichero WSDL:

- La parte **QUE**, consta de elementos `types`, `message`, y `portType`, define el mensaje y los tipos de datos intercambiados entre el cliente y el servidor. Un `message` es el elemento de comunicación básico de SOAP. Un mensaje puede constar de una o más partes, cada parte representa un parámetro con tipo. Hay dos mensajes (entrada y salida) por cada método de nuestra clase Java del servicio de Stocks. Como no usamos ningún tipo complejo o compuesto, no hay definiciones de tipos compuestos en este WSDL (no te preocupes, veremos muchos de ellos en otros ejemplos). Todos

los mensajes se agrupan en `operations` en una entidad llamada `portType`. Un `portType` representa el interface -- un conjunto concreto de operaciones soportadas por el Servicio Web. Un Servicio Web puede tener varios interfaces representados por distintos `portTypes`. Mira el `portType`, `StockQuoteService` en el fichero WSDL de ejemplo. Incluye dos operaciones: `getAvailableStocks` y `getQuote`. Para invocar al método `getQuote`, el cliente envía un mensaje `StockQuote_getQuote_Request`. (Encontraras la definición más abajo en el fichero). Observa que el mensaje `StockQuote_getQuote_Request` consta de una `part` (el parámetro de entrada) llamado `p0`, que está definido como un tipo `string` del esquema XML (`xsd:string`). Se supone que el Servicio Web responderá con el mensaje `StockQuote_getQuote_Response`, que contiene una `part` (el valor de retorno) llamado `response`, que es del tipo `double` del esquema XML (`xsd:double`).

- La parte **COMO**, consta de elementos `binding`, describe los detalles de la implementación técnica de nuestro Servicio Web. Un `binding` une un `portType` a un protocolo de comunicación (en este caso, SOAP sobre HTTP). Como estamos usando SOAP, usamos varios elementos de extensibilidad SOAP para WSDL para definir las especificidades de nuestra unión SOAP. (Observa que muchos de los elementos de esta sección usan el prefijo del espacio de nombres `soap:`. Estos elementos son extensiones SOAP para WSDL. El atributo `soapAction` del elemento `soap:operation` es un atributo específico-HTTP que se puede utilizar para especificar el propósito de un mensaje SOAP. Puede contener un parámetro de enrutamiento del mensaje o un valor que ayude al entorno de ejecución SOAP a determinar qué aplicación o método se debería ejecutar. El valor especificado por este atributo también se debe especificar en el atributo `SOAPAction:` de la cabecera HTTP del mensaje de solicitud SOAP. En nuestro caso este atributo no contiene ningún valor. Una unión SOAP requiere que especifiquemos el estilo de comunicación usado para cada operación en el `portType`. SOAP soporta dos estilos de comunicación: `RPC` y `Document`. El estilo `RPC` soporta formación y deformación automática de mensajes, permitiendo a los desarrolladores expresar una solicitud como una llamada a método con un conjunto de parámetros, que devuelve una respuesta que contiene un valor de retorno. El estilo `Document` no soporta formación automática de mensajes. Asume que los contenidos del mensaje SOAP son datos XML bien-formateados. Una unión SOAP también requiere que especifiquemos como se expresan los mensajes en XML. Podemos usar valores `literal` o tipos de datos `encoded`. El atributo `use='literal'` indica que el entorno de ejecución SOAP debería enviar el XML como se le ha proporcionado. El atributo `use='encoded'` indica que el entorno de ejecución SOAP debería serializar los datos por nosotros usando un estilo de codificación particular. Un estilo de codificación define un conjunto de reglas para expresar tipo de un lenguaje de programación en XML. En este caso hemos usado el estilo de codificación definido en la sección 5 de la especificación SOAP. Aunque también se puede utilizar otros estilos de codificación.
- Finalmente, la parte **DONDE**, consta del elemento `service`, colocando juntos el

porttype, el binding, y la localización real (una URI) del Servicio Web. Chequea el elemento `service` al final del documento WSDL.

Como puedes ver, un fichero WSDL describe completamente un Servicio Web. Dando un fichero WSDL, tenemos toda la información necesaria para crear una aplicación cliente que pueda acceder a nuestro Servicio Web de Stocks.

■



Implementar un Cliente Java de un Servicio Web

Un cliente se `bind`(une) a un Servicio Web remoto usando un componente proxy Java, Cuando se utiliza WASP de Systinet, este proxy se genera en tiempo de ejecución desde el fichero WSDL. Necesitamos un interface Java que pueda mantener una referencia a este objeto creado dinámicamente. Podemos crear el interface nosotros mismos, o podemos usar el `WSDLCompiler` de WASP para que genere uno por nosotros. La creación del interface es sencilla porque el único requerimiento es que los métodos del interface deben ser un subconjunto de los métodos de la clase Java de la lógica de negocio del Servicio

Web. Echemos una mirada al código de abajo. Primero, el cliente crea un objeto `WebServiceLookup`. Este objeto se utiliza entonces para crear el proxy del Servicio Web invocando el método `lookup`. Este método requiere dos parámetros: una referencia al fichero WSDL y la clase del interface Java que referenciará el ejemplar proxy. El método `lookup` devuelve el proxy que se utiliza para invocar el Servicio Web.

```

/**
 * Stock Client
 *
 * @created July 17, 2001
 * @author zdenek
 */

package com.systinet.demos.stock;

import org.idoox.wasp.Context;
import org.idoox.webservice.client.WebServiceLookup;

public class StockClient {

    /**
     * Web service client main method.
     * Finds the web service and
     * @param args not used.
     */
    public static void main( String[] args ) throws Exception {

        // lookup service
        WebServiceLookup lookup = (WebServiceLookup)Context.getInstance(Context.WEB
SERVICE_LOOKUP);
        // bind to StockQuoteService
        StockQuoteServiceProxy quoteService = (StockQuoteServiceProxy)lookup.lookup
(
    "http://localhost:6060/StockQuoteService/",
    StockQuoteServiceProxy.class
);

        // use StockQuoteService
        System.out.println("Getting available stocks");
        System.out.println("-----");
        java.util.LinkedList list = quoteService.getAvailableStocks();
        java.util.Iterator iter = list.iterator();
        while(iter.hasNext()) {
            System.out.println(iter.next());
        }
        System.out.println("");

        System.out.println("Getting SUNW quote");
        System.out.println("-----");
        System.out.println("SUNW "+quoteService.getQuote("SUNW"));
        System.out.println("");

    }

}

```

Ejecutamos el script `runJavaclient.bat`. Este script ejecutará el `WSDLCompiler` para generar el interface Java, luego lo compilará y ejecutará la aplicación cliente. Deberías ver la salida de los métodos `getAvailableStocks` y `getQuote` en la consola.

■



Desplegar y Ejecutar el Servicio Web JavaScript

Nota:

Por favor, observa que el Cliente del Servicio Web JavaScript necesita Microsoft Internet Explorer 6.0 o Microsoft Internet Explorer 5.0 con



Microsoft XML Parser 3.0 SP2

(<https://download.microsoft.com/download/xml/Install/3.0/WIN98Me/EN-US/msxml3.exe>).

Podemos generar un cliente JavaScript basado en navegador usando el script `runJScriptClient.bat`. Este script abrirá un navegador IE con una página HTML. Entonces podremos invocar todos los métodos del Servicio Web desde esta página.

■



Mensajes SOAP de un Vistazo

Ahora podemos usar la consola de administración de WASP para ver los mensajes SOAP que se han intercambiado entre el cliente y el servidor. Primero debemos abrir la consola introduciendo esta dirección **http://localhost:6060/admin/console** en nuestro navegador. Luego pulsamos sobre el botón **Refresh** para ver todos los paquetes desplegados. Deberíamos ver como nuestro servicio `StockQuoteService` está desplegado en el servidor. Activamos la depuración de todos los mensajes SOAP pulsando el enlace "enable" (cerca de la etiqueta "Debug is OFF:" en la sección `StockQuoteService` de la consola de administración). Luego volvemos a ejecutar el script del cliente del Servicio Web `runJavaclient.bat` y pulsamos sobre el enlace `show SOAP conversation` en la consola de administración. Esto debería abrir una ventana de navegador que muestra dos parejas de mensajes SOAP de entrada y salida:

```
==== INPUT ==== http://localhost:6060/StockQuoteService/ ==== 11/7/01 3:45 PM =
<?xml version="1.0" encoding="UTF-8"?>
```



```
<ns0:Envelope xmlns:ns0="http://schemas.xmlsoap.org/soap/envelope/">
  <ns0:Body
    ns0:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <ns0:getAvailableStocks
      xmlns:ns0="http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stock/" />
    </ns0:Body>
  </ns0:Envelope>
```

==== CLOSE =====

==== OUTPUT ==== http://localhost:6060/StockQuoteService/ =====

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:Envelope xmlns:ns0="http://schemas.xmlsoap.org/soap/envelope/">
  <ns0:Body
    ns0:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <ns0:getAvailableStocksResponse
      xmlns:ns0="http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stock/">
      <response xsi:type="ns1:LinkedList" xmlns:ns1="http://idoox.com/containers">
        <item xsi:type="xsd:string">SUNW</item>
        <item xsi:type="xsd:string">MSFT</item>
        <item xsi:type="xsd:string">BEAS</item>
      </response>
    </ns0:getAvailableStocksResponse>
  </ns0:Body>
</ns0:Envelope>
```

==== CLOSE =====

==== INPUT ==== http://localhost:6060/StockQuoteService/ ==== 11/7/01 3:45 PM =

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:Envelope xmlns:ns0="http://schemas.xmlsoap.org/soap/envelope/">
  <ns0:Body
    ns0:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <ns0:getQuote
      xmlns:ns0="http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stock/">
      <p0 xsi:type="xsd:string">SUNW</p0>
    </ns0:getQuote>
  </ns0:Body>
</ns0:Envelope>
```

==== CLOSE =====

==== OUTPUT ==== http://localhost:6060/StockQuoteService/ =====

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:Envelope xmlns:ns0="http://schemas.xmlsoap.org/soap/envelope/">
  <ns0:Body
    ns0:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<ns0:getQuoteResponse
  xmlns:ns0="http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stock/">
  <response xsi:type="xsd:double">10.0</response>
</ns0:getQuoteResponse>
</ns0:Body>
</ns0:Envelope>
==== CLOSE =====
```

Los mensajes SOAP siguen la siguiente estructura básica:

```
<ENVELOPE attrs>
  <HEADER attrs>
    <directives/>
  </HEADER>
  <BODY attrs>
    <payload/>
  </BODY>
  <FAULT attrs>
    <errors/>
  </FAULT>
</ENVELOPE>
```

El contenido del mensaje está encerrado en el ENVELOPE. En este caso sencillo, nuestros mensajes SOAP, sólo contienen una sección BODY. También puede haber otras dos secciones, llamadas HEADER y FAULT. La sección HEADER normalmente se usa para la propagación de distinta información de contexto (por ejemplo, contexto de transacción, credenciales de seguridad, etc.). Si ocurre un error, la sección FAULT debería portar información sobre la naturaleza del fallo. La sección BODY contiene la información importante (en nuestro ejemplo, el valor del stock y datos relacionados). Generalmente, SOAP no obliga a ninguna regla para la sección BODY. Ya hemos mencionado dos posibles estilos para la sección BODY: Document y RPC. El estilo Document no tiene requerimientos de formateo rígidos más allá de las reglas estándar del XML, mientras que el estilo RPC define reglas para marcar la llamada a método con todos sus parámetros. La especificación SOAP recomienda pero no obliga al estilo codificado, el marco de trabajo básico para expresar valores tipados en un mensaje SOAP. El estilo de codificación SOAP está basado en los tipos de datos definidos en el "La Parte 2 de la Recomendación del Esquema XML" que incluye tipos primitivos como `int`, `float`, `double` o `string`. La codificación SOAP también define reglas para construir tipos complejos (por ejemplo arrays, estructuras, etc.) sobre estos tipos primitivos. En nuestro caso (estamos usando el estilo RPC con codificación SOAP), la sección BODY del mensaje de entrada contiene el nombre del método invocado con los parámetros codificados:

```
<ns0:getQuote xmlns:ns0="http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stock/">
  <p0 xsi:type="xsd:string">SUNW</p0>
</ns0:getQuote>
```

El mensaje de salida contiene el resultado de la llamada al método:

```
<ns0:getQuoteResponse
  xmlns:ns0="http://idoox.com/wasp/tools/java2wsdl/output/com/systinet/demos/stoc
k/">
  <response xsi:type="xsd:double">10.0</response>
</ns0:getQuoteResponse>
```

■



Limpieza

Al final debemos eliminar nuestro sencillo servicio del servidor ejecutando el script `undeploy.bat`.



COMPARTE ESTE ARTÍCULO

 **ENVIAR A UN AMIGO**

 **COMPARTIR EN FACEBOOK**

 **COMPARTIR EN TWITTER**

 **COMPARTIR EN GOOGLE +**

ARTÍCULO ANTERIOR

Tutorial básico de MySQL
(/articulo/tutorial_basico_de_mysql_189)

SIGUIENTE ARTÍCULO

IIS y el método Server.Execute
(/articulo/iis_y_el_metodo_server_execute_191)

Hosting para aplicaciones JAVA - JVM privada desde 6,90€/Mes

Hosting específico para aplicaciones JAVA. Con garantía de reembolso 30 días para probarlo anw.es

 **Anuncio**

ABRIR

¡SÉ EL PRIMERO EN COMENTAR!

Conéctate (/login) o **Regístrate (/registro)** para dejar tu comentario.

Secciones

Artículos (/articulos)
Tutoriales y código fuente (/codigos)
Foros (/foros)
Eventos (/eventos)
Empleo (/empleo)

Lenguajes Destacados

PHP (/php)
Java (/java)
ASP (/asp)
Bases de datos (/bases-de-datos)
C (/c)

Información

Datos Legales (/datos_legales)
Política de privacidad (/politica_de_privacidad)
Publicidad (/publicidad)

Contacto

Contacte con nosotros (/contacto)

Publicidad (/publicidad)



(<https://www.facebook.com/programacionencastellano>)



(<https://twitter.com/noprogramacion>)

Diseño web y desarrollo web (<https://colorvivo.com>). Un proyecto de los hermanos Carrero (<https://carrero.es>).

Alojado en cloud privado StackScale (<https://www.stackscale.es/>)

Más internet: Password (<https://password.es>) | Favicon (<https://getfavicon.com>) | Crear un Avatar (<https://face.co>)

Copyright © 1998-2019 Programación en Castellano. Todos los derechos reservados