

Comprender las especificaciones de los servicios web, Parte 1: SOAP

Nicholas Chase (ibmquestions@nicholaschase.com)

05-08-2011

Freelance writer

Backstop Media

El énfasis actual en las arquitecturas orientadas a servicios (SOA) ha puesto el foco en los servicios web, pero es fácil perderse en toda la información disponible. Esta serie trae la verdadera historia de las principales especificaciones de los servicios web, y comienza con el Protocolo simple de acceso a objetos (SOAP), hasta llegar al Lenguaje de ejecución de procesos de negocios WS (WS-BPEL). Este tutorial explica los conceptos básicos de los servicios web y de SOAP, y explica cómo crear un servidor y un cliente SOAP.

[Ver más contenido de esta serie](#)

Antes de comenzar

Esta serie de tutoriales relata la creación de un sistema de flujo de trabajo basado en servicios web en un pequeño diario ficticio, el Daily Moon. Es para desarrolladores que quieren conocer mejor los conceptos que subyacen en los servicios web, para que puedan crear sus aplicaciones de una manera más eficiente. Los usuarios no-técnicos también encontrarán cosas valiosas en estos tutoriales, puesto que explican los conceptos antes de entrar en la programación.

Usted tendrá que contar con conocimientos básicos de programación y, si desea seguir adelante con los ejemplos de programación, también algo de Java. Hablaremos sobre XML, por necesidad, pero no ahondaremos en el tema, y abarcaremos los conceptos que sean necesarios.

Acerca de esta serie

Esta serie de tutoriales enseña los conceptos básicos de los servicios web, aprovechando el pequeño diario ficticio Daily Moon, ya que el personal usa servicios web para crear un sistema de flujo de trabajo para incrementar la productividad en estos tiempos agitados.

Esta primera parte comienza de una forma sencilla, explica los conceptos básicos inherentes a los servicios web y muestra cómo usar SOAP, la especificación que subyace en casi todo lo que vendrá a continuación, conectando al departamento de Clasificados con el Content Management System (Sistema de gestión de contenidos).

Las entregas futuras de esta serie se basarán en los conceptos básicos:

- La parte dos va un paso más adelante, y explica cómo usar Web Services Description Language (lenguaje de descripción de servicios web, WSDL, por su sigla en inglés), para definir los mensajes creados, tal como se espera de los servicios web, lo que permitirá que el equipo cree servicios y clientes que se conectan con ellos, de una forma más sencilla.
- La parte tres encuentra al equipo con una cantidad de servicios instalados y el deseo de ubicarlos con facilidad. Como respuesta a ello, Universal Description, Discovery and Integration (Descripción, Descubrimiento e Integración Universal, UDDI), proporciona un registro de servicios disponibles, que permite realizar búsquedas como una forma de promocionar sus propios servicios a otros.
- Las partes cuatro y cinco, WS-Security (Seguridad-WS) y WS-Policy (Política-WS), relatan cómo proteger los servicios del diario y los cambios que deberán hacer los equipos para poder acceder a estos servicios recientemente protegidos.
- Interoperabilidad es la palabra clave de la parte seis, ya que hay varias implementaciones distintas a las que se debe acceder desde un único sistema. La parte seis abarca los requisitos y pruebas que implican la certificación WS-I.
- Para terminar, la parte siete muestra cómo usar el Business Process Execution Language (Lenguaje de ejecución de procesos de negocios para servicios web, WS-BPEL), para crear aplicaciones complejas a partir de servicios individuales.

Ahora veamos con un poco más de detalle lo que abarca este tutorial.

Acerca de este tutorial

Este tutorial le presenta el concepto de servicios web, y el equipo de Clasificados de nuestro diario ficticio Daily Moon. Usted acompañará al equipo, a medida que se integra con un sistema de servicios web existente, y será testigo de la creación de un servicio. El foco estará puesto en el Service Object Access Protocol (Protocolo de acceso a objetos de servicios, SOAP).

Durante el curso de este tutorial, usted aprenderá lo siguiente:

- Los conceptos básicos de los servicios web
- Los fundamentos de XML
- La estructura y el propósito de un mensaje SOAP
- Cómo instalar un servidor de aplicaciones en el que pueda ejecutar sus aplicaciones de servicios web.
- Cómo instalar una implementación de servicios web en un servidor de aplicaciones
- Cómo crear un mensaje SOAP de manera programática.
- Cómo crear un cliente para un servicio web basado en SOAP mediante el uso de Java y Apache Axis2
- Cómo crear un servicio web Java y Apache Axis2 basado en SOAP

En este tutorial, el Departamento de Clasificados se integrará con el sistema de gestión de contenidos, y creará un cliente. También dará un vistazo al proceso de creación de uno de los servicios con el que interactúa el departamento. Los ejemplos de programación se muestran en Java con el uso del proyecto Apache Axis2, pero los conceptos valen virtualmente para cualquier otro lenguaje y entorno.

Requisitos previos

Para poder avanzar con el código de este tutorial, deberá disponer del siguiente software:

- Un servidor de aplicaciones Apache Geronimo u otro. Usted creará diversos servicios web a lo largo de este tutorial, y necesitará una aplicación a donde poder ejecutarlos. Como, por supuesto, los servicios web son interoperables, no tiene importancia cuál use. En este tutorial, demostraremos la instalación y el uso de Apache Geronimo, que es también la base de IBM® WebSphere® Community Edition. También puede usar otros servidores de aplicaciones, tales como WebSphere Application Server. Puede descargar Apache Geronimo del sitio [Apache Geronimo Downloads](#).
- Apache Axis2 u otra implementación SOAP. Puede crear mensajes SOAP a mano, y los puede interpretar a mano, pero es mucho más sencillo tener una implementación al alcance de la mano. Usará Apache Axis2, que contiene implementaciones de diversas APIs relacionadas con SOAP, que le harán la vida mucho más sencilla. Puede descargar Apache Axis2 en: [Apache.org](#). Este tutorial usa la versión 0.94, pero también debería funcionar con versiones más actuales.
- Java™ 2 Standard Edition versión 1.4 o superior. Ambas herramientas están basadas en Java, al igual que los servicios y clientes que creará en este tutorial. Puede descargar J2SE SDK desde [aquí](#).
- También necesitará un navegador Web y un editor de texto, pero estoy seguro de que ya los tiene. Si lo desea, también puede usar un IDE, como Eclipse, pero como el foco está puesto en las tecnologías y no en las herramientas, simplemente usaré un editor de texto y la línea de comandos para editar archivos y compilarlos.

¿Qué son los servicios web?

Comencemos con una idea general de lo que son los servicios web, y por qué son importantes para el desarrollo de software.

Después de todo ¿cuál es el problema?

Si usted no se hubiese cansado de escuchar todo tipo de información acerca de la arquitectura orientada a los servicios (SOA) y los servicios web, no estaría aquí, entonces la pregunta es ¿por qué tanto problema con esto? La respuesta es que se trata de algo importante porque es un cambio de paradigma en la forma en que las aplicaciones se comunican entre ellas. Las SOAs existen desde hace muchísimo tiempo. Al principio se trataba mayormente de aplicaciones middleware en las que un único tipo de middleware posee, como mínimo, los dos extremos del cable. Por otra parte, los servicios web, están compuestos por un grupo de estándares que tratan de posibilitar que diversos sistemas puedan comunicarse, sin la necesidad de un middleware en particular, de lenguaje de programación, o incluso de un sistema operativo. Veamos la progresión desde donde comenzamos hasta llegar a ahora.

Aplicaciones tradicionales

Al principio, fueron las computadoras. Y fue algo bueno. Las computadoras hacían tareas aparentemente milagrosas, automatizaban muchas cosas que la gente hacía a mano, desde cálculos complejos, para pasar a las finanzas, y a muchas otras tareas.

Pero las aplicaciones tradicionales son “silos”. La aplicación de recursos humanos no podía hablar con la de finanzas que, a su vez, no podía hablar con la aplicación de distribución. Todas estas aplicaciones tenían su propio hogar, en sus propias computadoras y, si bien eran útiles, no era una buena forma de compartir datos entre ellas. Uno tenía la opción de escribir procesos batch (por lotes) para pasar los datos de un sistema al otro, pero eso no era una sustitución de la integración en tiempo real.

Computación distribuida

El paso siguiente en nuestra cadena de evolución es la computación distribuida. La computación distribuida permitía que diferentes aplicaciones se comunicaran entre sí, aun estando en computadoras distintas. Las tecnologías tipo CORBA, NTS, y Enterprise Java Beans (EJB), proporcionaron un sistema que incluía un registro de estilos, para que las aplicaciones pudiesen encontrar componentes con los que querían interactuar, y luego llamarlos como si estuviesen ubicados en la máquina local.

Estos sistemas era compatibles mediante middleware, o más específicamente, mediante middleware orientado a mensajes, que proporcionaba los dos requisitos. Ahora se pueden crear las aplicaciones de manera tal que pueden acceder a los recursos de otros sistemas, incluso si se encuentran en ubicaciones geográficas diferentes.

Pero seguía habiendo un problema. Si bien las aplicaciones se podían comunicar a cualquier parte dentro del sistema, éste seguía siendo un sistema cerrado. Como mínimo su aplicación de cliente debía usar la misma tecnología de la aplicación del servidor. Asimismo, por regla general, los sistemas no estaban diseñados para que se accediera a ellos por fuera de la organización individual que los había creado.

servicios web

El siguiente, y casi inevitable vínculo en esta cadena evolutiva son los servicios web. Basados en XML, y, en la mayoría de los casos, HTTP, los “servicios web” todavía significas muchas cosas para mucha gente, pero en este caso, hablaremos de los servicios web como el intercambio entre sistemas de mensajes basados en SOAP.

Estos mensajes se componen de XML, que es un estándar abierto basado en texto, accesible por cualquier persona desde cualquier aplicación (cualquier aplicación que no esté diseñada para aceptarlo). Esto amplía el mundo de su aplicación para que cualquier persona pueda acceder a ella en su red. (Si eso le enciende alarmas de seguridad, está bien, aprenderá cómo resolverlo en la parte cuatro de esta serie.)

El servicio web basado en SOAP implica el envío de un mensaje XML, tal como aparece en el Listado 1.

Listado 1. Servicio web basado en SOAP

```
<SOAPenv:Envelope
  xmlns:SOAPenv="http://schemas.xmlsoap.org/SOAP/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAPenv:Body>
    <req:getNumberOfArticles xmlns:req="http://daily-moon.com/CMS/">
      <req:category>classifieds</req:category>
    </req:getNumberOfArticles>
  </SOAPenv:Body>
</SOAPenv:Envelope>
```

Estos mensajes van de un sistema a otro, por lo general, vía HTTP. El sistema receptor interpreta el mensaje, hace lo que se supone que debe hacer, y devuelve una respuesta con la forma de otro mensaje SOAP.

Es un sistema simple, y como tal, hay muchos aspectos de la computación a nivel de empresa que no los contempla. Afortunadamente, se han tomado en cuenta muchos de estos aspectos, y tienen sus propias

especificaciones para determinar la manera en que debe producirse esta transacción para incorporar muchos de los aspectos de seguridad y otros del middleware tradicional orientado a mensajes.

Otros tipos de servicios web

Sería negligente si no mencionara que SOAP no es la única forma de hacer servicios web. Existen otras formas basadas en XML para enviar mensajes entre sistemas, algunas de las cuales son adecuadas para un entorno de empresa, y otras que no lo son. Por ejemplo, Amazon fue una de las primeras empresas basadas en Web en ofrecer a su público el acceso de servicios web a su sistema. Amazon incluye un servicio basado en SOAP, pero también proporciona un servicio basado en Representational State Transfer (Transferencia de Estado Representacional, REST).

REST es un tipo de servicio web en el que el usuario simplemente accede a la URL, y la respuesta es un auténtico documento XML, como el que aparece en el Listado 2.

Listado 2. Respuesta REST

```
<currentArticles>
  <category>classifieds</category>
  <subcategory>forsale</subcategory>
  <article id="888204">
    <articleHeadline></articleHeadline>
    <articleText>30 ft ladder, only used once. Willing to let
      go for half it's worth. Has slight dent near the middle.
      Harder than a human head. $150 OBO.</articleText>
  </article>
  <article id="888242">
    <articleHeadline></articleHeadline>
    <articleText>Vintage 1963 T-Bird. Less than 300 miles.
      Driven by my daughter until I took it away. Serious inquires only.
      555-3264 after 7 PM.</articleText>
  </article>
</currentArticles>
```

No hay un formato en particular para estos mensajes. Simplemente se trata de lo que sea el dato.

Otro tipo de servicios web implica el uso de un estándar, tal como XML-RPC. En este caso, los comandos se envían a un sistema a través de XML, como aparece en el Listado 3.

Listado 3. XML-RPC

```
<?xml version="1.0"?>
<methodCall>
  <methodName>CMS.getNumberOfArticles</methodName>
  <params>
    <param>
      <value><string>classifieds</string></value>
    </param>
    <param>
      <value><string>forsale</string></value>
    </param>
  </params>
</methodCall>
```

La respuesta viene con un formato similar.

Como usted aprenderá a usar SOAP, puede pensar que REST y XML-RPC son mucho más sencillos que un sistema basado en SOAP. Y tiene razón. De alguna manera lo son. Sin embargo, no hablamos de una aplicación sencilla para visualizar el estado del tiempo en su sitio Web. Aquí se trata de aplicaciones a nivel de empresa, y estas aplicaciones necesitan atributos a nivel de empresa, como por ejemplo seguridad, interoperatividad, etc. Estas capacidades están cubiertas por especificaciones adicionales que nacieron en torno a los servicios web basados en SOAP, lo que, a la larga, hace de SOAP una mejor opción para las aplicaciones a nivel de empresa.

Veamos algunas de estas especificaciones.

Especificaciones básicas de los servicios web

Por lo general, las especificaciones de los servicios web entran en dos categorías: especificaciones básicas de servicios web y especificaciones ampliadas de servicios web. Las especificaciones básicas son:

- **SOAP:** El fundamento de todos los servicios web basados en SOAP, la especificación SOAP detalla el formato de los mensajes. También detalla la forma en que las aplicaciones deben tratar determinados aspectos del mensaje, tales como los elementos del “encabezado”, lo que le permitirá crear aplicaciones en las que un mensaje pasa entre múltiples intermediarios antes de llegar a su destino final. Este tutorial abarcará la especificación SOAP.
- **WSDL:** Web Services Description Language (Lenguaje de descripción de los servicios web) es una especificación que detalla una forma estándar de describir un servicio web basado en SOAP, que incluye la forma que deberán tomar los mensajes, y a dónde deben ser enviados. También detalla la respuesta a ese mensaje. Combinado con las herramientas adecuadas, WSDL le permite crear de manera programática, una llamada a un servicio web sin saber en realidad lo que busca el servicio web; la aplicación puede extraer esos detalles del archivo WSDL y proporcionarle las interfaces programáticas para que las use. Nos ocuparemos de WSDL en la parte dos de esta serie.
- **UDDI:** Universal Description, Discovery and Integration (Descripción, Descubrimiento e Integración Universales) es un estándar que ha sufrido algunos cambios desde su concepción inicial. La idea era proporcionarle a las empresas una forma de registrar sus servicios en un registro global, y consultar ese registro global para buscar servicios que quisieran utilizar. Sin embargo, como es comprensible que muchas empresas sean algo renuentes a abrir sus sistemas a desconocidos, no se materializó este objetivo. No obstante, UDDI se afianzó como un registro interno de servicios e información de servicios; la parte tres de esta serie da detalles de su uso.

Esos son los fundamentos. También existen literalmente docenas de estándares ampliados para hacer que los servicios basados en SOAP sean más útiles.

Especificaciones ampliadas de los servicios web

De las docenas de especificaciones WS-* que andan dando vueltas, varias se destacan como particularmente útiles para la empresa. Y son:

- **WS-Security** (Seguridad para servicios web): Esta especificación maneja el encriptado y las firmas digitales, lo que le permitirá crear una aplicación para que los mensajes no pueden ser ‘espiados’, y donde es imposible el no rechazo. La parte cuatro de esta serie se ocupa de WS-Security.

- **WS-Policy** (Política de los servicios web): Esta especificación es una ampliación de WS-Security, la que le permitirá detallar de una manera más específica cómo y quiénes pueden usar un servicio web. La parte cinco de esta serie se ocupa de WS-Policy.
- **WS-I**: Aunque se supone que los servicios web están diseñados para tener interoperabilidad, en realidad hay bastante flexibilidad en las especificaciones de que las interpretaciones entre las distintas implementaciones pueden causar problemas. WS-I proporciona un conjunto de estándares y prácticas para prevenir ese tipo de problemas, como así también, pruebas estandarizadas para verificar problemas. WS-I es el objeto de la parte seis de esta serie.
- **WS-BPEL** (Lenguaje de ejecución de procesos de negocios para servicios web): un servicio único está bien, pero en la mayoría de los casos no se trata de una aplicación. Como mínimo, la computación a nivel de empresa necesita que usted cree servicios múltiples dentro de un sistema general, y WS-BPEL le proporciona el medio para especificar interacciones tales como el procesamiento bifurcado y coincidente, necesario para crear esos sistemas. La parte siete de esta serie se ocupa de WS-BPEL.

Otras especificaciones, que tienen un papel importante en los servicios web, no están incluidas en esta serie e incluyen **WS-ReliableMessaging**, que le permite estar seguro de que se ha recibido una, y solo una, copia de un mensaje, y que ha sido recibida definitivamente; **WSRF**, el Web Services Resource Framework (Marco de trabajo de recursos de servicios web), le permite usar estados en un entorno que, básicamente, no guarda estados precedentes; y **Web Services Distributed Management (WSDM)** (Gestión de servicios web distribuidos), que trata el problema de la gestión y el uso de los servicios web. Encontrará más información acerca de estas especificaciones y otras en [Temas relacionados](#) (Recursos) de este tutorial.

Lo que vamos a lograr

En este tutorial, usted seguirá al departamento Clasificados del diario Daily Moon a medida que investigan cómo integrar sus propios sistemas con la interfaz basada en servicios web que usa el sistema de gestión de contenidos. Usted creará una aplicación que crea un mensaje basado en SOAP, lo envía al servicio, y recibe una respuesta. Una vez que lo haya hecho, verá cómo crear un servicio que responda solicitudes y envíe una respuesta propia. Esto lo hará mediante la creación de aplicaciones Java, con un servidor de aplicaciones y sin él.

Configuración

Ahora que ya comprende los principios básicos que esto implica, comencemos con la creación efectiva de una aplicación. El primer paso es instalar el software.

Configuración de Apache Geronimo

El primer software que necesitará es un servidor de aplicaciones Web. ¿Por qué necesita un servidor de aplicaciones Web? Bueno, porque sin él, le va ser muy difícil dar servicios web. Un servidor de aplicaciones Web atiende solicitudes, las traduce a algo que el servicio pueda entender, y luego hace el proceso que sea necesario.

Para este proceso usted puede usar virtualmente cualquier servidor Web, pero en la mayoría de los casos instalará software que facilite ese proceso, y eso suele requerir un servidor de aplicaciones determinado de algún tipo. Específicamente, este tutorial supone que usted usará un servidor de aplicaciones Java. (En realidad, supone que será un servidor de aplicaciones J2EE.)

Cuando se trata de servidores J2EE, usted tiene muchas opciones y, en este caso, usará Apache Geronimo. Geronimo es el servidor de aplicaciones de código abierto que forma la base de IBM WebSphere Application Server Community Edition. Geronimo es pequeño, de fácil instalación y de fácil manejo. También funciona bien con otros proyectos Apache, como por ejemplo Axis2, que usted instalará después.

Descargue el software (ver [Requisitos Previos](#)) y extraiga los archivos en un directorio de destino. Verá que los archivos extraídos tienen su propio directorio, de manera que sólo tendrá que descomprimirlos y moverlos a donde quiera. Se acepta cualquier directorio, pero evite aquellos que contengan un espacio en su nombre, como “Archivos de Programa”, “Documents and Settings”, o sus descendientes. Por ejemplo, la instalación de prueba para el tutorial usa `e:\geronimo-1.0`.

Listo. Ya instaló Geronimo. Fácil ¿no?

Para iniciar el servidor, abra una ventana de petición de comandos y ejecute los siguientes comandos:

```
cd <GERONIMO_HOME>
```

```
java -jar server.jar
```

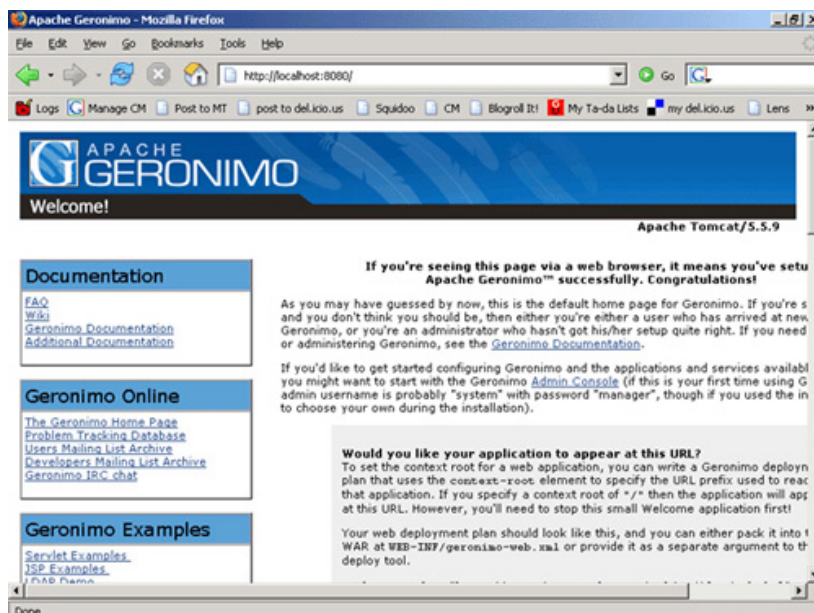
Para asegurarse de que el servidor funciona, abra un navegador y apúntelo a la URL que aparece en el Listado 4:

Listado 4. URL del servidor

`http://localhost:8080`

Deberá ver una página similar a la de la Figura 1.

Figura 1. Servidor en funcionamiento



Ahora instalemos el software de los servicios web.

Instalación de Apache Axis2

Es completamente posible dar servicios web desde un servidor HTTP común. Sin embargo, no es aconsejable. Hay mucho que hacer para procesar mensajes SOAP, y no hay motivos para que usted reinvente la rueda. Desde hace ya varios años, el proyecto Apache Axis2 simplifica esta tarea con la creación de un entorno donde, crear y procesar servicios web es muy fácil. El software incluye aplicaciones que lo ayudarán a crear un servicio web a partir de un objeto común, crear un objeto Java a partir de un servicio web, y a procesar ambos. El grupo Apache designó una nueva versión de Axis, Axis2, que toma todo el trabajo hecho en Axis y lo eleva una categoría mediante el cambio de arquitectura para permitirle un mayor grado de elasticidad. Esto es importante, porque continuamente aparecen especificaciones para servicios web. La creación de Axis2 permite una integración más sencilla con proyectos tales como WSS4J, la implementación de WS-Security de Apache. Como más adelante usaremos estos proyectos, instale Axis2 ahora (ver [Requisitos previos](#) de información sobre la descarga).

Asegúrese de descargar tanto la Binary Distribution (Distribución binaria) como la War Distribution (Distribución War). La primera lo ayudará con la creación de los clientes, y la segunda con la creación de los servicios.

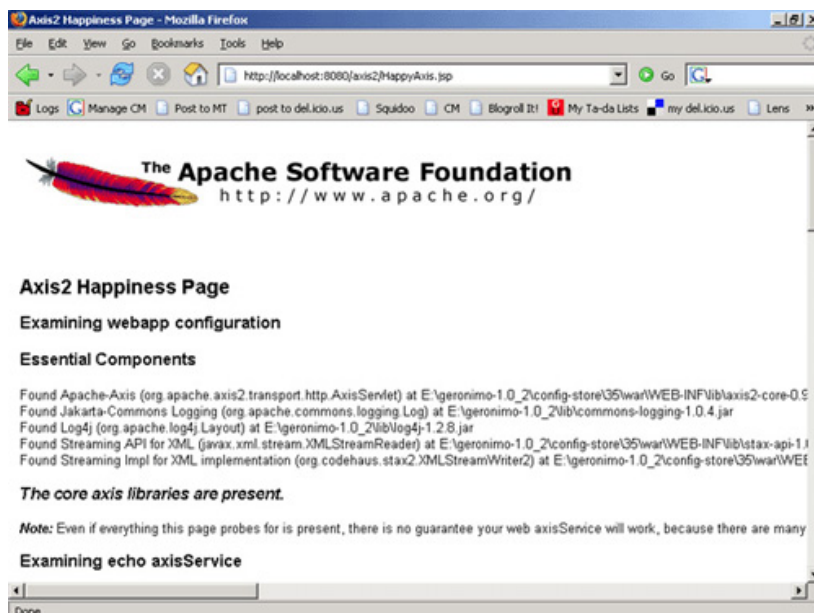
Para instalar Axis2 en el servidor Web, copie el archivo axis2.war en el directorio de implementación de Geronimo. (Para que aparezca el directorio, debe asegurarse de haber iniciado Geronimo por lo menos una vez.) Geronimo detecta su presencia inmediatamente, por lo que no tendrá que implementar nada.

Verificación del eje a la instalación

Para estar seguro de que todo se ha instalado correctamente, apunte su navegador a <http://localhost:8080/axis2/index.jsp> y haga clic en Validate. (Validar).

Deberá ver una página como la que se muestra en la Figura 2.

Figura 2. Axis feliz



Asegúrese de que Axis esté feliz antes de continuar con el resto del tutorial.

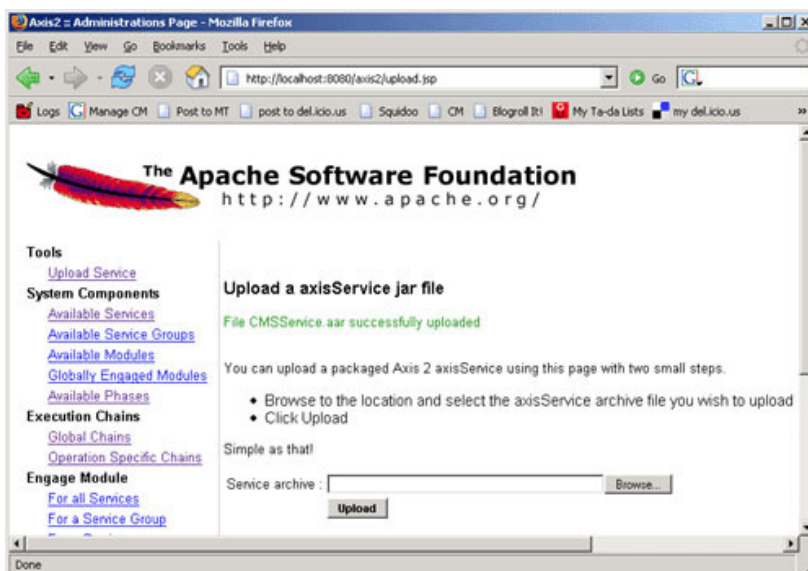
Instalación de un servicio de muestra

La primera aplicación que escribirá es un cliente, por lo que necesitará un servicio al que éste pueda acceder. Afortunadamente, la distribución Axis2 trae varios. En principio, instalará el servicio MyService de la siguiente forma:

1. Auténtíquese en la aplicación Axis2 apuntando su navegador a `http://localhost:8080/axis2/Login.jsp` y después inicie la sesión. El nombre de usuario y la contraseña predeterminados son `admin` y `Axis2`, respectivamente.
2. Haga clic en **Upload service>Browse** (Cargar servicio>Navegar).
3. Navegue hacia el archivo MyService.aar. Podrá encontrarlo en el directorio `samples/userguide` de la distribución estándar de Axis2. Haga clic en **OK**.
4. Haga clic en **Upload** (Cargar).

Deberá ver una notificación de que el servicio ha sido incorporado (ver Figura 3). Axis2 incluye, de manera predeterminada, “hot deployment”, por lo que no tendrá que hacer nada para activarlo.

Figura 3. Se ha cargado MyService.aar



Ahora veamos lo que usted va a crear.

Entender SOAP

Ahora que ya instaló el software, puede comenzar a ver el servicio web propiamente dicho. Ahora que ya instaló el software, puede comenzar a ver el servicio web propiamente dicho. Como mencioné en [Other kinds of Web services](#) (Otros tipos de servicios web), usted dispone de diversos formatos para elegir. En esta serie, usará SOAP.

Un breve comentario acerca de XML

Todos estos mensajes que van y vienen están basados en Extensible Markup Language (Lenguaje de marcado extensible) o XML. Si no está familiarizado con XML, investigue un poco al respecto antes de

entrar en profundidad en los temas de los servicios web. Sin embargo, aquí aparecen los fundamentos que debe conocer antes de seguir adelante con este tutorial.

XML es un lenguaje de “marcado ampliable”, lo que significa que proporciona una forma de suministrar información adicional sobre el contenido. Esta información tiene la forma de “tags” (“etiquetas”), las que denotan “elements” (“elementos”). Por ejemplo, tome en cuenta este documento XML sencillo que aparece en el Listado 5:

Listado 5. Archivo XML que muestra los fundamentos

```
<article articleId="88271" categoryId="classifieds" subcategoryId="forsale">
  <articleHeadline>Fun, fun, fun</articleHeadline>
  <articleText>Vintage 1963 T-Bird. Less than 300 miles.
Driven by my daughter until I took it away. Serious
inquires only. 555-3264 after 7 PM.</articleText>
</article>
```

Observe un par de cosas acerca de este texto. Ante todo, es texto. Eso lo hace legible por casi cualquiera, o por casi cualquier cosa. Segundo, las etiquetas están indicadas con > y <, con una etiqueta “open” (de apertura) que contiene un nombre, y atributos posibles, como la ID del artículo, y una etiqueta de cierre con una barra (/). Los elementos deben estar auto-contenidos y anidados correctamente. En otras palabras, usted no podría tener un documento XML similar al del Listado 6.

Listado 6. Muestra no válida de un documento XML

```
<article articleId="88271" categoryId="classifieds" subcategoryId="forsale">
  <articleHeadline>Fun, fun, fun
  <articleText></articleHeadline>Vintage 1963 T-Bird.
Less than 300 miles. Driven by my daughter until I
took it away. Serious inquires only. 555-3264 after
7 PM.</articleText>
</article>
```

XML también proporciona una forma de separar el contenido en diferentes “espacios de nombres”, para que pueda ser tratado de otra manera por una aplicación. Por ejemplo, un mensaje SOAP puede parecerse al siguiente, del Listado 7.

Listado 7. Ejemplo de mensaje SOAP

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
  </env:Header>
  <env:Body>
    <cms:getNumberOfArticles xmlns:cms="http://www.daily-moon.com/cms">
      <cms:category>classifieds</cms:category>
      <cms:subcategory>forsale</cms:subcategory>
    </cms:getNumberOfArticles>
  </env:Body>
</env:Envelope>
```

No se preocupe por la estructura del mensaje, pero observe que hay dos “prefijos” distintos, cada uno de los cuales corresponde a un espacio de nombres en particular. En este caso, diferenciamos el “sobre” SOAP de la carga útil real.

Reitero, hay mucho para aprender acerca de XML, pero esos son los fundamentos que debe conocer para este tutorial.

El sobre SOAP

La unidad básica de un mensaje de servicio web es el sobre SOAP. Se trata de un documento XML que incluye toda la información necesaria para procesar el mensaje (ver Listado 8).

Listado 8. Ejemplo de mensaje SOAP

```
<?xml version='1.0' ?> <env:Envelope
  xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header> </env:Header> <env:Body>
</env:Body> </env:Envelope>
```

En este caso, usted tiene un Envelope (Sobre) sencillo, con el espacio de nombres especificado como SOAP versión 1.2. Incluye los dos sub-elementos, un Header (Encabezado) y un Body (Cuerpo). Veamos qué hacen cada uno de estos elementos.

El encabezado SOAP

El Header (Encabezado) de un mensaje SOAP tiene la finalidad de proporcionar información acerca del mensaje en sí mismo, contrariamente a la información destinada a la aplicación. Por ejemplo, el Header (Encabezado) puede incluir información de ruteo, tal como lo hace en este ejemplo que aparece en el Listado 9.

Listado 9. Información de ruteo en el Header (Encabezado)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
    <wsa:ReplyTo xmlns:wsa=
      "http://schemas.xmlsoap.org/ws/2004/08/addressing">
      <wsa:Address>
http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
      </wsa:Address>

    </wsa:ReplyTo>
    <wsa:From>
      <wsa:Address>
http://localhost:8080/axis2/services/MyService</wsa:Address>
    </wsa:From>
    <wsa:MessageID>ECE5B3F187F29D28BC11433905662036</wsa:MessageID>
  </env:Header>

  <env:Body>
</env:Body>
</env:Envelope>
```

En este caso usted ve un elemento de WS-Addressing, que incluye información acerca del destino del mensaje y del destino de las respuestas. El Header (Encabezado) y puede incluir todo tipo de información sobre el mensaje en sí mismo. En realidad, la especificación SOAP insume mucho tiempo en elementos que pueden ir en el Header (Encabezado), y cómo deben ser tratados por los "intermediarios SOAP". En otras palabras, la especificación SOAP no infiere de manera alguna que el mensaje irá directamente de un

punto a otro, desde el cliente al servidor. Permite la idea de que un mensaje SOAP puede ser procesado en efecto por diversos intermediarios, en su ruta al destino final, y la especificación es muy clara en cuanto a la forma en que deberán tratar esos intermediarios la información del **Header** (Encabezado). No obstante, esa discusión está más allá del alcance de este tutorial. Por lo que, por ahora, simplemente comprenda que el **Header** (Encabezado) le puede proporcionar una funcionalidad mucho mayor si la necesitara.

Ahora veamos la carga útil.

The SOAP body (El cuerpo SOAP)

Cuando usted envía un mensaje SOAP, lo hace con un motivo en mente. Usted trata de decirle al destinatario que haga algo, o trata de impartir información al servidor. Esta información se llama "carga útil". La carga útil va en el **Body** del **Envelope**. También tiene su propio espacio de nombres, en este caso el que corresponde al sistema de gestión de contenidos. En este caso, la elección del espacio de nombres, es completamente arbitraria. Sólo tiene que ser diferente del espacio de nombres de SOAP (ver Listado 10).

Listado 10. Ejemplo de una carga útil

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    <cms:addArticle xmlns:cms="http://www.daily-moon.com/cms">

      <cms:category>classifieds</category>
      <cms:subcategory>forsale</cms:subcategory>
      <cms:articleHeadline></cms:articleHeadline>
      <cms:articleText>Vintage 1963 T-Bird. Less than 300 miles.
Driven by my daughter until I took it away. Serious inquires only.
555-3264 after 7 PM.</cms:articleText>

    </cms:addArticle>
  </env:Body>
</env:Envelope>
```

En este caso, usted tiene una carga útil simple, que incluye instrucciones para agregar un artículo al sistema de gestión de contenidos para el Daily Moon.

La selección de la estructura de la carga útil incluye el estilo y la codificación.

Estilo y codificación

Este tema lo verá con mayor profundidad en la parte dos, que habla del Web Services Description Language (Lenguaje de descripción de los servicios web, WSDL), pero cuando cree su aplicación, necesitará decidir la estructura de la carga útil que enviará una y otra vez. A tal fin, tomemos unos instantes para hablar de la programación del estilo y de la codificación.

Dicho de manera sencilla, predominan dos estilos de programación diferentes para los mensajes SOAP. El primero es el estilo RPC, que se basa en el concepto de usar mensajes SOAP para crear Remote Procedure Calls (Llamadas a procedimientos remotos). En este estilo, la idea es que usted enviará un comando al servidor, como por ejemplo “agregar un artículo”, e incorporará los parámetros para ese comando, como

por ejemplo el artículo que se agregará y la categoría a la que deberá ser incorporado, como elementos secundarios del método en general, según el Listado 11. Listado 11.

Listado 11. Estilo RPC

```
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
```

La alternativa al estilo RPC implica que usted tenga solamente sus datos como contenido del cuerpo SOAP, e incluye la información relativa al procedimiento o función a la que ésta pertenece en el ruteo del mensaje por parte del servidor de aplicaciones (ver Listado 12).

Listado 12. Datos como contenido en el cuerpo SOAP

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
</env:Header>
  <env:Body>
    <cms:addArticle xmlns:cms="http://www.daily-moon.com/cms">
      <cms:category>classifieds</category>

      <cms:subcategory>forsale</cms:subcategory>
      <cms:articleHeadline></cms:articleHeadline>
      <cms:articleText>Vintage 1963 T-Bird. Less than 300
miles. Driven by my daughter until I took it away.
Serious inquires only. 555-3264 after 7 PM.</cms:articleText>
    </cms:addArticle>

  </env:Body>
</env:Envelope>
```

Una variante del estilo RPC es RPC/encoded (RPC/codificado), a diferencia de RPC/literal, como podemos ver arriba. En aquel caso, el mensaje incluye información de tipo, tal como aparece en el Listado 13.

Listado 13. Ejemplo de RPC/codificado

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
</env:Header>
  <env:Body>
    <cms:addArticle xmlns:cms="http://www.daily-moon.com/cms">

      <cms:category xsi:type="xsd:string">classifieds</category>
      <cms:subcategory xsi:type="xsd:string">forsale
</cms:subcategory>
      <cms:articleHeadline xsi:type="xsd:string" />

      <cms:articleText xsi:type="xsd:string">Vintage 1963
T-Bird. Less than 300 miles. Driven by my daughter until
I took it away. Serious inquires only. 555-3264 after 7
PM.</cms:articleText>
    </cms:addArticle>
  </env:Body>
</env:Envelope>
```

Al segundo estilo se lo conoce como estilo document/literal, que implica únicamente la incorporación de los datos apropiados en el mensaje, como el que se muestra en el Listado 14.

Listado 14. Ejemplo de estilo document/literal

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
  </env:Header>
  <env:Body>
    <category>classifieds</category>

    <subcategory>forsale</subcategory>
    <articleHeadline></articleHeadline>
    <articleText>Vintage 1963 T-Bird. Less than 300 miles.
    Driven by my daughter until I took it away. Serious
    inquires only. 555-3264 after 7 PM.</articleText>
  </env:Body>
</env:Envelope>
```

En este caso, el mensaje propiamente dicho no incluye información acerca del proceso del destino a donde deberán ser enviados los datos; eso lo maneja el software de ruteo. Por ejemplo, todos los llamados a una URL o punto final en particular deben apuntar a una operación en particular. Usted también podría usar el estilo documento/codificado técnicamente, pero nadie lo hace, así que, por el momento, olvídelo.

Cada uno de estos estilos implica concesiones mutuas y, reitero, la parte dos de esta serie los abarca con más detalle. Sin embargo, es importante saber que hay un tercer estilo, "document wrapped," que no está especificado formalmente en ninguna parte, pero que se ha hecho popular por diversos motivos de interoperabilidad. En este caso, el contenido de la carga útil está ajustado en un único documento, pero ese elemento no puede llevar el mismo nombre de la aplicación o procedimiento al que pertenecen los datos. Para el ojo humano, estos mensajes son virtualmente idénticos a los mensajes RPC/literales.

Patrones de intercambio de mensajes

Cuando se trata del envío de mensajes, usted cuenta con una amplia variedad de opciones, que van desde el envío de una solicitud y la espera de una respuesta, al envío de una solicitud y olvidarse de ella, hasta el envío de una solicitud y hacerla pasar a través de intermediarios múltiples antes de que llegue a su destino final. No obstante, al final, sólo tiene dos opciones:

- **Solicitud/Respuesta:** Según el patrón Solicitud/Respuesta, usted envía una solicitud con forma de mensaje SOAP y simplemente espera la llegada de una respuesta. La solicitud puede ser sincrónica o asincrónica.
- **Mensajería unidireccional:** Este caso, también conocido como método "dispare y olvídense", implica el envío de la solicitud para después olvidarse directamente de ella. Esto lo puede usar en situaciones en las que simplemente imparte información, o en alguna otra en la que, en realidad, no le importa lo que el destinatario pueda decir de ella.

Ahora, observe la falta de los términos "cliente" y "servidor". Esto se explica porque estos patrones de intercambio de mensajes pueden usarse básicamente para crear cualquier cantidad de alternativas diferentes, tal cual las mencionadas un poco más arriba. Por ejemplo, usted puede crear una situación en la cual envía una solicitud, y cuenta con que el destinatario se ocupará de ella, enviando un mensaje en algún momento futuro cuando esté listo lo que se supone que debe hacer. Para hacerlo, usted usará una combinación de mensajes unidireccionales, por lo que no tiene sentido hablar del "cliente" y del "servidor", porque cada mensaje tiene un remitente y un destinatario, y los así llamados cliente y servidor, cambian de lugar.

Creación de un cliente SOAP

Bien, ya vio la teoría, ahora llegó el momento de crear las aplicaciones reales. Comencemos por el cliente.

La forma antigua

Cuando aparecieron por primera vez las APIs Java para el trabajo con mensajes SOAP, eran muy específicas en cuanto a su razón de ser. Eran, a las claras, para la creación del mensaje SOAP. Se requería que usted creara el mensaje, el `Envelope` (Sobre), el `Header` (Encabezado), el `Body` (Cuerpo), etc. Por ejemplo, puede crear un cliente según el “estilo antiguo” para acceder a la función `echo` (eco) del servicio `MyService` que instaló antes (ver Listado 15).

Nota: Para compilar y ejecutar este cliente, necesitará una implementación SAAJ, como el software original de Axis. Puede descargar Axis desde <http://ws.apache.org/axis/>. Supuestamente, Axis2 0.95 también viene con esta implementación, pero no ha sido probada para este tutorial.

Listado 15. Cliente SOAP en la forma antigua

```
import javax.xml.SOAP.*;
import javax.xml.transform.*;
import java.io.InputStream;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;

public class SendSOAP {

    public static void main(String args[]) {

        try {

            MessageFactory messageFactory = MessageFactory.newInstance();
            SOAPMessage message = messageFactory.createMessage();

            //Create objects for the message parts
            SOAPPart SOAPPart = message.getSOAPPart();
            SOAPEnvelope envelope = SOAPPart.getEnvelope();
            SOAPBody body = envelope.getBody();

            SOAPElement bodyElement =
                body.addChildElement(envelope.createName("echo",
                    "req", "http://localhost:8080/axis2/services/MyService/"));
            bodyElement.addChildElement("category")
                .addTextNode("classifieds");
            message.saveChanges();

            SOAPPart SOAPpartbefore = message.getSOAPPart();
            SOAPEnvelope reqenv = SOAPpartbefore.getEnvelope();

            System.out.println("REQUEST:");
            System.out.println(reqenv.toString());

            //Now create the connection
            SOAPConnectionFactory SOAPConnFactory =
                SOAPConnectionFactory.newInstance();
            SOAPConnection connection =
                SOAPConnFactory.createConnection();

            SOAPMessage reply = connection.call(message,
                "http://localhost:8080/axis2/services/MyService");
```



```

    SOAPPart SOAPpart = reply.getSOAPPart();
    SOAPEnvelope replyenv = SOAPpart.getEnvelope();

    System.out.println("\nRESPONSE:");
    System.out.println(replyenv.toString());

    connection.close();

} catch (Exception e){
    System.out.println(e.getMessage());
}
}
}

```

Note que usted crea directamente el `SOAPEnvelope`, `SOAPBody`, etc. Usted puede agregar elementos, tales como el elemento `echo` (eco) y el elemento `category` (categoría) en ese cuerpo. A partir de allí, usted crea la conexión, hace la llamada, y podrá volver a abrirse camino a través de la estructura del mensaje SOAP para llegar al contenido. En caso de ejecutar este cliente, verá una respuesta similar a la que aparece en el Listado 16.

Listado 16. El cliente hasta aquí

REQUEST:

```

<SOAPEnv:Envelope xmlns:SOAPEnv=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAPEnv:Body>
    <req:echo xmlns:req=
"http://localhost:8080/axis2/services/MyService/">
      <req:category>classifieds</req:category>
    </req:echo>
  </SOAPEnv:Body>
</SOAPEnv:Envelope>

```

RESPONSE:

```

<SOAPEnv:Envelope xmlns:SOAPEnv=
"http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa=
"http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <SOAPEnv:Header>
    <wsa:ReplyTo>
      <wsa:Address>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:From>
      <wsa:Address>
        http://localhost:8080/axis2/services/MyService</wsa:Address>
      </wsa:From>
    <wsa:MessageID>ECE5B3F187F29D28BC11433905662036</wsa:MessageID>
  </SOAPEnv:Header>
  <SOAPEnv:Body>
    <req:echo xmlns:req=
"http://localhost:8080/axis2/services/MyService/">
      <req:category>classifieds</req:category>
    </req:echo>
  </SOAPEnv:Body>
</SOAPEnv:Envelope>

```

Todo lo que, en realidad hace el servicio `echo` es volver a solicitar con la solicitud que ha recibido, lo que es una buena oportunidad para ver la diferencia entre el proceso del estilo antiguo y el estilo nuevo. Hablemos de esa diferencia.

La forma nueva

En la actualidad, hay un movimiento creciente para esconder la complejidad del trabajo del programador con los mensajes de servicios web basados en XML. Han surgido todo tipo de iniciativas, la mayoría de las cuales tienden a hacer la programación de los servicios web lo más parecido a programar cualquier otra arquitectura.

En Axis2, esto significa más que eso. Axis2 presenta una forma totalmente nueva de trabajo con el XML que representa al mensaje SOAP, aunque en la superficie sea muy similar al uso de Document Object Model (Modelo de Objetos de Documentos, DOM). El modelo de objeto AXIs, o AXIOM, hace muchos cambios, pero por el momento sólo mencionaré que se concentra en el conjunto de información del mensaje, que es la información genuina contenida en los elementos y atributos, más que en la versión en serie de las etiquetas que vemos normalmente. Lo que es más importante, sin embargo, es que Axis2 se ocupa del mensaje SOAP por nosotros, lo que nos permite concentrarnos simplemente en la creación de la carga útil o, en el caso de los servicios reales, en analizar la carga útil y crear una respuesta. Veamos cómo funciona.

Crear la solicitud

Para comenzar con la creación del cliente, asegúrese de que todos los archivos *.jar del directorio Axis2 lib —que sería la distribución Estándar, no la War— estén en su CLASSPATH, y cree una clase nueva llamada `ClassifiedClient`. Cree la carga útil como se muestra en el Listado 17.

Listado 17. Carga útil

```
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.apache.axis2.om.OMElement;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import java.io.StringWriter;
import org.apache.axis2.om.OMAbstractFactory;
import org.apache.axis2.SOAP.SOAPFactory;
import org.apache.axis2.om.OMFactory;
import org.apache.axis2.om.OMNamespace;

public class ClassifiedClient {

    public static OMElement getEchoOMElement() {
        SOAPFactory fac = OMAbstractFactory.getSOAP12Factory();
        OMNamespace omNs = fac.createOMNamespace(
            "http://daily-moon.com/cms", "cms");
        OMElement method = fac.createOMElement("echo", omNs);
        OMElement value = fac.createOMElement("category", omNs);
        value.addChild(fac.createText(value, "classifieds"));
        method.addChild(value);

        return method;
    }

    public static void main(String[] args) {
        try {
            OMElement payload = ClassifiedClient.getEchoOMElement();
        } catch (Exception e) { //(XMLStreamException e) {
            System.out.println(e.toString());
        }
    }
}
```

Primero, usted crea un generador y un espacio de nombres, y los usa para crear elementos. En este caso, usted creará los mismos elementos que creó en el ejemplo anterior, ya que usará otra vez este cliente para acceder a la función `echo`. (Más adelante, lo cambiará para acceder al servicio real.)

Luego, creará la solicitud.

Crear la solicitud

El paso siguiente es crear la solicitud real. Aquí volverá a ver la marcha del tiempo. En lugar de enviar simplemente la solicitud a una URL, usted configura una "referencia de extremo", tal como aparece en el Listado 18.

Listado 18. Referencia de extremo en la solicitud

```
...
public class ClassifiedClient {
    private static EndpointReference targetEPR = new
        EndpointReference(
            "http://localhost:8080/axis2/services/MyService");

    public static OMElement getEchoOMElement() {
    ...
    }

    public static void main(String[] args) {
        try {
            OMElement payload = ClassifiedClient.getEchoOMElement();
            Options options = new Options();
            options.setTo(targetEPR);
            options.setTransportInProtocol(Constants.TRANSPORT_HTTP);

        } catch (Exception e) { //(XMLStreamException e) {
            System.out.println(e.toString());
        }
    }
}
```

El comentario completo acerca de WS-Addressing está fuera del alcance de este tutorial, pero basta decir que una referencia de extremo incluye la URL hacia donde se orienta el mensaje pero que, opcionalmente, puede incluir otra información, tal como una respuesta-a la dirección, y otras propiedades de los recursos.

Primero, usted crea las Opciones para la solicitud, las que le permitirán configurar la EPR (referencia de extremo) para la solicitud, como así también otra información, como por ejemplo, el transporte que piensa usar. Una vez que tiene todo eso listo, ya puede enviar la solicitud.

Enviar la solicitud

Una vez que ha configurado todo, es hora de enviar la solicitud. Desde Axis, hasta la versión 0.94, la forma preferible de enviar un mensaje es a través de la clase `ServiceClient` que aparece en el Listado 19.

Listado 19. Envío de la solicitud

```
...
public static void main(String[] args) {
    try {
        OMElement payload = ClassifiedClient.getEchoOMElement();
        Options options = new Options();
        options.setTo(targetEPR);
        options.setTransportInProtocol(Constants.TRANSPORT_HTTP);

        ServiceClient sender = new ServiceClient();
        sender.setOptions(options);
        OMElement result = sender.sendReceive(payload);

    } catch (Exception e) { //(XMLStreamException e) {
        System.out.println(e.toString());
    }
}
```

Usted crea el objeto `ServiceClient` y configura las `Options` (Opciones) que creó anteriormente para él. A partir de allí, ya podrá enviar el mensaje. Y como quiere obtener una respuesta, usará el método `sendReceive()` que es un mensaje de entrada/salida. Verá la mensajería unidireccional en el capítulo [The one-way service](#) (Servicio unidireccional) de este tutorial. El método `sendReceive()` remite la respuesta real, tal como la recibió el cliente.

Enviar el resultado

En realidad, `sendReceive()` no remite toda la respuesta, sino únicamente la carga útil. Si usted envía el resultado a la línea de comandos, podrá verlo claramente en el Listado 20.

Listado 20. Envío de `sendReceive`

```
...
sender.setOptions(options);
OMElement result = sender.sendReceive(payload);

System.out.println(result.toString());

} catch (Exception e) { //(XMLStreamException e) {
    System.out.println(e.toString());
}
}
```

La ejecución de este cliente le da la respuesta que aparece en el Listado 21.

Listado 21. Respuesta de `sendReceive`

```
<cms:echo xmlns:cms="http://daily-moon.com/cms"><cms:category>classifieds</cms:category></cms:echo>
```

Por supuesto, podrá hacer muchas cosas con estos datos, una vez que los recibe. Por ahora, vamos a crear el verdadero servicio `getNumberOfArticles()`.

Creación de un servicio SOAP

Si el proceso de creación de un servicio web le resulta bastante sencillo, tiene razón. Y la creación de un servicio, hasta cierto punto, es igualmente sencilla.

Proceso general

El proceso general de la creación de un servicio web Axis2 implica los siguientes pasos:

1. Crear el manifiesto de servicios
2. Crear la clase
3. Empaquetarlos en un archivo de almacenamiento Axis
4. Cargar el archivo de almacenamiento Axis en la aplicación Web Axis2
5. De ser necesario, reiniciar el servidor

Eso es todo. Comencemos con el manifiesto de servicios.

Crear el manifiesto

El manifiesto de servicios le dice a la aplicación Axis2 (y por extensión, al servidor de aplicaciones) qué solicitudes corresponden a qué clases. Por ejemplo, usted puede especificar dos funciones de servicio, como se muestra en el Listado 22.

Listado 22. Especificación de dos funciones de servicio en el manifiesto

```
<service name="CMSService">
  <description>
    This is a sample web service for the newspaper's
    Content Managment System.
  </description>

  <parameter name="ServiceClass" locked="false"
    >CMSService</parameter>

  <operation name="getNumberOfArticles">
    <messageReceiver class=
      "org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
  </operation>
  <operation name="addArticle">
    <messageReceiver class=
      "org.apache.axis2.receivers.RawXMLINOnlyMessageReceiver"/>
  </operation>
</service>
```

First, you define the overall service, giving it a name and a description and specifying the class to actually serve the request. Next, you define the actual operations. Notice that this example specifies two different types of `messageReceiver`. El primero, `RawXMLINOutMessageReceiver`, es para el servicio solicitud/respuesta tradicional. El segundo, `RawXMLINOnlyMessageReceiver` es para mensajes unidireccionales. El nombre de la operación corresponde al elemento raíz de la carga útil, como así también el método a ejecutar.

Guarde este archivo como `services.xml`.

Ahora creemos la aplicación real.

Crear la aplicación

Comience con la creación de una clase que imite la función `echo` (eco) que vimos antes, remitiendo una copia de la carga útil original, que aparece en el Listado 23.

Listado 23. Clase CMSService

```
import org.apache.axis2.om.OMElement;
import javax.xml.stream.XMLStreamException;

public class CMSService {

    public OMElement getNumberOfArticles(OMElement element)
        throws XMLStreamException {

        element.build();
        element.detach();

        return element;
    }
}
```

Para compilar esta aplicación, asegúrese de que todos los archivos *.jar de <axis2_home>/lib estén en su CLASSPATH.

La aplicación es bastante sencilla, con una sola función que corresponde a la operación `getNumbereOfArticles`. Esta función, y cualquier otra función que tenga como destino una operación, toma un único `OMElement`, que representa la carga útil. Aquí, usted primero usa el método `build()` para asegurarse de que se hayan recibido todos los datos —AXIOM usa un método pull de acceso a los datos— y luego desvincula el elemento de su árbol actual para poder remitirlo.

Si se cree valiente, no dude en [Implementar el servicio](#) y en [Acceder al servicio](#) para acceder al servicio y ver los resultados. Deberá ver algo similar a lo que se muestra en el Listado 24.

Listado 24. Respuesta de la Clase CMSService

```
<cms:getNumberOfArticles><cms:category>classifieds</cms:category></cms:
getNumberOfArticles>
```

Ahora veamos cómo manejar los datos en la realidad.

Extraer la carga útil

La extracción de información de la carga útil es una forma de manipulación del elemento carga útil recibido usando técnicas muy similares a DOM (ver Listado 25).

Listado 25. Extracción de información de la carga útil

```
...
import javax.xml.stream.XMLStreamException;

public class CMSService {
    public OMElement getNumberOfArticles(OMElement element)
        throws XMLStreamException {
        element.build();
        element.detach();

        String rootName = element.getLocalName();
        OMElement categoryElement = element.getFirstElement();
        String categoryElementName = categoryElement.getLocalName();
        String categoryValue = childElement.getText();

        return element;
    }
}
```

Recuerde que, la raíz de una carga útil es el elemento recibido por la función `getNumberOfArticles`. En este caso, usted extrae el nombre de ese elemento, y pasa al primer elemento de su elemento secundario (contrariamente a su elemento secundario, que puede ser un nodo de texto de espacio en blanco) y extrae su nombre y valor. Tome nota de que usted usa el método `getText()` para extraer lo que es esencialmente el valor de un elemento secundario de nodo de texto de la categoría elemento. ¡Definitivamente un atajo muy útil!

Crear y remitir la respuesta

Para terminar, usted usará los datos que extrajo de la carga útil de la solicitud para crear una respuesta. En este caso, lo alimentará a una segunda función que, en una aplicación real, haría otro trabajo (ver Listado 26).

Listado 26. Creación de una respuesta

```
...
import javax.xml.stream.XMLStreamException;

public class CMSService {
    public OMElement getNumberOfArticles(OMElement element)
        throws XMLStreamException {
        element.build();
        element.detach();

        String rootName = element.getLocalName();
        OMElement childElement = element.getFirstElement();
        String childName = childElement.getLocalName();
        String categoryValue = childElement.getText();

        SOAPFactory factory = OMAbstractFactory.getSOAP12Factory();
        OMNamespace namespace = factory.createOMNamespace(
            "http://daily-moon.com/cms/", "resp");
        OMElement resultElem = factory.createOMElement(
            "numberOfArticles", namespace);

        String actualValue =
            (articleCount(categoryValue)).toString();
        resultElem.setText(actualValue);

        return resultElem;
    }
}
```

```
private Integer articleCount(String catId){  
  
    //Perform some function such as searching the CMS  
    //database, and return the actual value. For our  
    //purposes, you'll hardcode it.  
    return new Integer(42);  
  
}
```

Primero, cree el generador que usará para crear todos los otros objetos, y después cree el espacio de nombres que agregará a la carga útil de la respuesta. A continuación, cree los elementos de resultado reales, en este caso un elemento llamado `numberOfArticles`.

El contenido del elemento `numberOfArticles` será un número que devolverá la función `articleCount()` que, en este caso, es completamente arbitrario. En una aplicación real, usted hará todo lo que sea necesario para obtener este dato. Una vez obtenido, lo configurará como contenido del elemento `numberOfArticles` y simplemente remitirá ese elemento.

Lo único que falta ahora, es implementar el servicio.

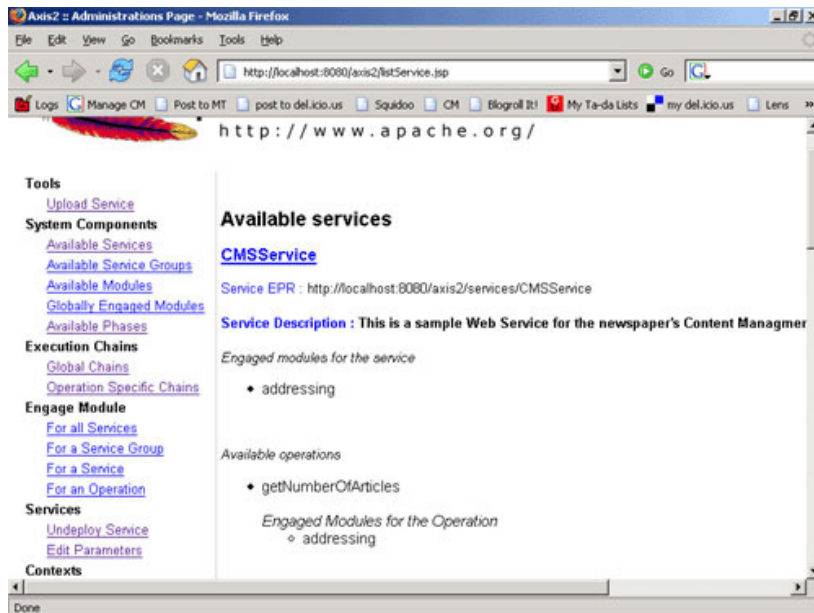
Implementar el servicio

Para poder implementar el servicio, deberá crear un archivo de almacenamiento Axis. Este archivo es como un archivo `*.jar` o `*.war`, ya que se trata de un simple archivo zip con una extensión de archivo especial, en este caso `.aar`. Siga estos pasos para crear este archivo:

1. Incorpore todos los archivos del directorio `<AXIS2_HOME>/lib` a su `CLASSPATH` y compile el archivo `CMSService.java`.
2. Cree un directorio nuevo llamado `META-INF` en el mismo directorio del archivo `CMSService.class`.
3. Desde el directorio que contiene el archivo `CMSService.class`, emita este comando: `<code type="section" width="100"> jar cvf CMSService.aar ./* </code>` Deberá ver como resultado, algo parecido a: `<code type="section" width="100"> added manifest adding: CMSService.class(in = 513) (out= 330)(deflated 35%) adding: CMSService.java(in = 328) (out= 182)(deflated 44%) ignoring entry META-INF/ adding: META-INF/services.xml(in = 391) (out= 229)(deflated 41%) </code>`
4. Incorpore el servicio al servidor siguiendo los pasos descritos en [Instalación de un servicio de muestra](#). (Si ve errores de servlet en la interfaz Web, asegúrese de haber iniciado sesión en la aplicación Axis2. Si su sesión venció, la aplicación no se lo avisará, pero probablemente le muestre un error.)
5. En caso de ser necesario, reinicie Geronimo. (Es probable que no deba hacerlo después de haber incorporado el servicio, pero probablemente deba hacerlo después de realizar cambios.)

Si hace clic en el vínculo View services (Ver servicios), debe ver algo similar a lo que se muestra en la Figura 4.

Figura 4. Servicios disponibles



Acceder al servicio

Ahora que creó el servicio, llegó la hora de acceder a él a través del cliente. Haga los siguientes cambios en el archivo ClassifiedClient.java que creó anteriormente (ver Listado 27).

Listado 27. Modificación de ClassifiedClient

```
...
public class ClassifiedClient {
    private static EndpointReference targetEPR =
        new EndpointReference(
            "http://localhost:8080/axis2/services/CMSService");

    public static OMElement getEchoOMElement() {
        SOAPFactory fac = OMAbstractFactory.getSOAP12Factory();
        OMNamespace omNs = fac.createOMNamespace(
            "http://daily-moon.com/cms", "cms");
        OMElement method = fac.createOMElement("getNumberOfArticles", omNs);
        OMElement value = fac.createOMElement("category", omNs);
        value.addChild(fac.createText(value, "classifieds"));
        method.addChild(value);

        return method;
    }

    public static void main(String[] args) {
        try {
            OMElement payload = ClassifiedClient.getEchoOMElement();
            Options options = new Options();
            options.setTo(targetEPR);
            options.setTransportInProtocol(Constants.TRANSPORT_HTTP);

            ServiceClient sender = new ServiceClient();
            sender.setOptions(options);
            OMElement result = sender.sendReceive(payload);

            String response = result.getText();
            System.out.println("There are "+response+" classifieds at the moment.");

        } catch (Exception e) { //(XMLStreamException e) {
```

```
System.out.println(e.toString());
    }
}
}
```

Cuando compile y ejecute esta aplicación, deberá ver la respuesta que aparece en el Listado 28.

Listado 28. Respuesta de ClassifiedClient

En este momento hay 42
clasificados.

El servicio unidireccional

Antes seguir adelante, veamos las diferencias que implica trabajar con un servicio unidireccional, en lugar de hacerlo con uno de solicitud/respuesta.

El servicio

La creación de un servicio unidireccional es bastante sencilla. El proceso es exactamente el mismo que para la creación de un servicio solicitud/respuesta, con la excepción de que no remite nada. Por ejemplo, puede crear la operación `addArticle` en la clase `CMSService`, como se muestra en la Figura 29.

Listado 29. Operación addArticle en CMSServiceclass

```
...
...
private Integer articleCount(String catId){
...
}

public void addArticle(OMElement element)
    throws XMLStreamException{
    element.build();
    System.out.println(element);
}
}
```

En el archivo `services.xml`, usted especificó la operación `addArticle` como una operación “entrante únicamente”, por lo que no se esperará remitir nada, pero sólo para que pueda ver que, en realidad sucede algo, envíe la carga útil recibida a la línea de comandos. La verá en la ventana de Geronimo.

En una aplicación real, este método extraería la información de la carga útil para incorporarla a determinado tipo de base de datos o repositorio.

El cliente

El cliente para este servicio también es similar al que usted usaría para un servicio solicitud/respuesta (ver Listado 30).

Listado 30. Creación del cliente

```
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.apache.axis2.om.OMElement;
import org.apache.axis2.SOAP.SOAPFactory;
```

```

import org.apache.axis2.om.OMAbstractFactory;
import org.apache.axis2.om.OMNamespace;

public class AddArticleClient {
    private static EndpointReference targetEPR =
        new EndpointReference(
            "http://localhost:8080/axis2/services/CMSService");

    private static OMElement getOMElement(){

        SOAPFactory fac = OMAbstractFactory.getSOAP12Factory();
        OMNamespace omNs = fac.createOMNamespace(
            "http://daily-moon.com", "cms");
        OMElement method = fac.createOMEElement("addArticle", omNs);

        OMElement category = fac.createOMEElement("category", omNs);
        category.setText("classifieds");

        OMElement subcategory =
            fac.createOMEElement("subcategory", omNs);
        category.setText("wantads");

        OMElement adtext = fac.createOMEElement("article", omNs);
        adtext.setText("Do you have good head for numbers"+
            " and a great deal of patience? Do you like"+
            " to sit for hours sorting objects by their"+
            " size? If so, then you could be the"+
            " next goober counter in the world famous"+
            " Murphy Brothers peanut factory. "+
            " Willingness to dress up as our mascot"+
            " helpful, but not required.");

        method.addChild(category);
        method.addChild(subcategory);
        method.addChild(adtext);

        return method;
    }

    public static void main(String[] args) {
        try {
            OMElement payload = AddArticleClient.getOMElement();
            ServiceClient serviceClient = new ServiceClient();

            Options options = new Options();
            serviceClient.setOptions(options);
            options.setTo(targetEPR);

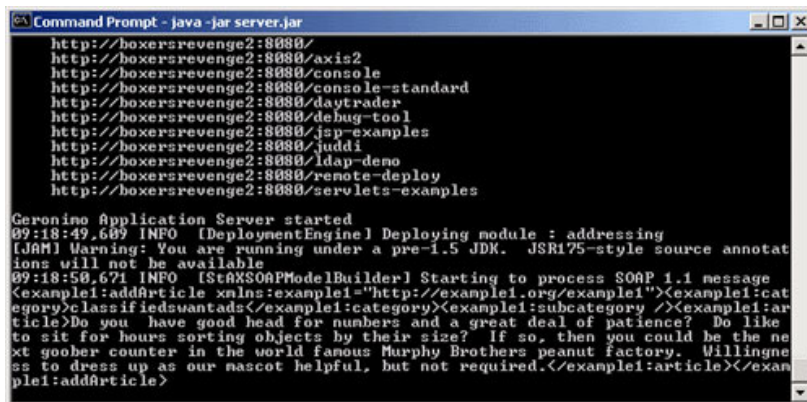
            serviceClient.fireAndForget(payload);

        } catch (AxisFault axisFault) {
            axisFault.printStackTrace();
        }
    }
}

```

Aunque la carga útil es diferente, por lo que puede ver en el método `getOMElement()`, el único cambio verdadero, en lo que hace a la programación, es el uso del método `fireAndForget()` en lugar del método `sendReceive()`. Este método no devuelve una respuesta.

Si ejecuta este cliente, verá un envío en la ventana de Geronimo, similar al que aparece en la Figura 5.

Figura 5. Envío de la línea de comandos


```

Command Prompt - java -jar server.jar
http://boxersrevenge2:8080/
http://boxersrevenge2:8080/axis2
http://boxersrevenge2:8080/console
http://boxersrevenge2:8080/console-standard
http://boxersrevenge2:8080/daytrader
http://boxersrevenge2:8080/debug-tool
http://boxersrevenge2:8080/jsp-examples
http://boxersrevenge2:8080/juddi
http://boxersrevenge2:8080/ldap-demo
http://boxersrevenge2:8080/remote-deploy
http://boxersrevenge2:8080/servlets-examples

Geronimo Application Server started
09:18:49.609 INFO [DeploymentEngine] Deploying module : addressing
[JAM] Warning: You are running under a pre-1.5 JDK. JSR175-style source annotations will not be available
09:18:50.671 INFO [StackSOAPModelBuilder] Starting to process SOAP 1.1 message
<example1:addArticle xmlns:example1="http://example1.org/example1"><example1:category>classifieds</example1:category><example1:subcategory></example1:subcategory><example1:article>Do you have good head for numbers and a great deal of patience? Do like to sit for hours sorting objects by their size? If so, then you could be the next goober counter in the world famous Murphy Brothers peanut factory. Willingness to dress up as our mascot helpful, but not required.</example1:article></example1:addArticle>

```

Acceso al servicio vía GET

Antes de SOAP 1.2, la única forma de acceder a un servicio web basado en SOAP con el uso de HTTP, era mediante el uso de una solicitud **POST**. Usted tenía que crear un cliente que creara una solicitud **POST** con el mensaje SOAP como contenido de esa solicitud. Sin embargo, SOAP 1.2, define la forma de acceder a un servicio web basado en SOAP con el uso de una solicitud **GET**.

GET contra POST

Antes de seguir adelante, es importante comprender la diferencia entre las solicitudes **GET** y **POST** en HTTP. Aunque muchos programadores actúan como si estas dos solicitudes fuesen intercambiables, existe en realidad un motivo para cada una de ellas. **GET**, donde toda la información acerca del recurso que usted solicita está contenida en la URL, por lo general como parámetros, está destinada al uso exclusivo de solicitudes idempotentes. Esas son solicitudes para las que no hay “efectos secundarios”. En otras palabras, usted podrá llamar a esa solicitud una docena de veces, cien veces, mil veces; y no debería cambiar nada. Por ejemplo, una solicitud Web para la temperatura actual en Albuquerque es idempotente. Una solicitud Web que inserta un comentario en la base de datos de un blog no lo es.

El motivo de esto es que las solicitudes **GET** pueden ser incorporadas a los marcadores de un usuario, y accedidas sin advertencias. También se las puede actualizar sin advertencia. Por otra parte, una solicitud **POST** incluye su información en el cuerpo de la solicitud, y por lo tanto es difícil repetir por accidente.

En lo que respecta a SOAP, esto significa que usted debe ser capaz de usar **GET** para solicitudes SOAP que simplemente recuperan información sin hacer cambios. También debería usar **POST** para cualquier operación que sí haga cambios.

Acceso al servicio

En Axis2, puede crear una solicitud **GET** y el servidor la traducirá a un mensaje SOAP y remitirá como resultado la carga útil. Por ejemplo, apunte su navegador a la ubicación que aparece en el Listado 31.

Listado 31. Acceso a los servicios

```
http://localhost:8080/axis2/services/CMSService/getNumberOfArticles?category=classifieds
```

Desde la versión 0.94 usted verá una respuesta como la que aparece en el Listado 32.

Listado 32. Respuesta de la carga útil SOAP

```
<resp:numberOfArticles>42</resp:numberOfArticles>
```

Sin embargo, esto no es muy preciso. Según la Recomendación SOAP 1.2, usted debería ver la respuesta SOAP completa. Esto probablemente cambie con las versiones futuras de Axis2.

Manejo de adjuntos

Otra variante del mensaje SOAP común es el adjunto. Los adjuntos se han ido metiendo en la piel de la gente durante años, pero dado que ahora son requeridos por determinadas especificaciones ampliadas, hay que aprender a manejarlos.

Datos binarios y XML

Aunque XML es un formato basado en texto, no podemos ignorar que existe un mundo binario. Como tal, llegará el momento en que debamos pasar información binaria hacia o desde un servicio web.

Usted podrá manejar esta situación de una de estas dos formas: La primera opción es incluir el dato binario dentro del documento. Un ejemplo de esto sucede cuando guardamos un documento de Microsoft Word como archivo XML. Si tenemos imágenes incrustadas en ese documento, Word las incrusta en el documento XML como dato binario, codificado en Base64. La segunda opción es referenciar el dato para que la aplicación que procesa el documento la pueda encontrar. Un ejemplo muy significativo de esto es un navegador Web y cómo maneja las imágenes referenciadas desde un archivo HTML. El archivo XHTML incluye un elemento de `img`, (o, si usted es adecuadamente avanzado, un elemento de `object` y ese elemento incluye un atributo `src` que contiene la URL que apunta a ese dato. La aplicación puede entonces cargar el dato desde esa ubicación y usarlo apropiadamente.

Lo mismo vale para los documentos SOAP. Digamos que, si usted fuera a enviar una imagen a un servicio basado en SOAP, tiene dos opciones. Puede incrustar ese dato binario en la carga útil, o puede encontrar una forma de referenciarlo. Históricamente, se lo ha referenciado, debido a los problemas de ancho de banda.

Paquetes Optimizados XML binario

Como puede ver, XML, ya es más detallado de lo que quisieran los oponentes del binario. Y como tal, usa mucho más ancho de banda. Entonces, cuando usted considera el hecho de que el método preferido para incorporar datos binarios a un documento de texto XML —codificándolo como Base64— tiene la tendencia a incrementar su tamaño por un factor de dos o más, usted tiene un verdadero problema.

En realidad, hubo tanto revuelo en los últimos años, por la falta de algún tipo real de soporte para los datos binarios, que fue casi una rebelión, y W3C se ocupó del problema. Como resultado, aparecieron los Paquetes optimizados XML binario (XOP). Este protocolo proporciona una forma de referenciar de manera fiable los datos externos desde dentro de un documento XML. Por ejemplo, la especificación SOAP con Adjuntos decía que el dato binario podía enviarse como parte de un documento MIME multiparte, donde el dato XML componía la primera parte, y el dato binario incorporado, como partes adicionales. El problema de esto es

que, aunque su programa pueda desconocer la existencia del dato, el documento no. Tampoco permite la optimización selectiva del documento, o el procesamiento retroactivo de un documento existente que incluya datos binarios.

XOP mejora esta situación mediante el suministro de un mecanismo por el cual se puede extraer selectivamente la información que debe ser optimizada, la incorpora a un mensaje MIME multiparte que también incluye nuestro mensaje SOAP, y hace una referencia directa a él. Veamos un ejemplo.

Supongamos, por ejemplo que, en lugar de incorporar un artículo nuevo como elemento de texto, el personal quisiera incorporarlo como documento binario, es decir, desde un programa que procesa texto. Sería terriblemente complicado incluir ese contenido en el cuerpo del mensaje, como se muestra en el Listado 33:

Listado 33. Incorporación de un documento binario

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
  </env:Header>
  <env:Body>
    <cms:addArticle xmlns:cms="http://www.daily-moon.com/cms">

      <cms:category>classifieds</category>
      <cms:subcategory>forsale
    </cms:subcategory>
      <cms:articleHeadline><cms:articleHeadline>
      <cms:articleText>wvvetwenptiubnweoirntuwopeirt4m[456n09ew7nv
sa0tv043u6y304o5mu60ew9rebtm45bm4-69nw-0er9utnv3094nb-26204
95u6-49kv6-m34956h-wb09emjb-0n67u-340v,=qw-enr7w8b64b03278-
ANDLOTSMOREBASE64ENCODEDDATAHERE</cms:articleText>

    </cms:addArticle>
  </env:Body>
</env:Envelope>
```

XOP especifica que, en cambio, usted extraiga el dato, y lo reemplace con un elemento `Include` (Incluir), que referencie su nueva ubicación, como en este ejemplo que aparece en el Listado 34.

Listado 34. Uso de XOP

```
MIME-Version: 1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
type="application/xop+xml";
start="<soapmsg.xml@daily-moon.com>";
  start-info="text/xml"
  Content-Description: An XML document with binary data in it

--MIME_boundary
Content-Type: application/xop+xml;
charset=UTF-8;
type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID: <soapmsg.xml@daily-moon.com>

<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
  <env:Header>
  </env:Header>
  <env:Body>
    <cms:addArticle xmlns:cms="http://www.daily-moon.com/cms">
      <cms:category>classifieds</category>
```

```

    <cms:subcategory>forsale
  </cms:subcategory>
  <cms:articleHeadline><cms:articleHeadline>
    <cms:articleText><xop:Include
      xmlns:xop='http://www.w3.org/2004/08/xop/include'
      href='cid:http://daily-moon.com/tbird.doc'
    /></cms:articleText>
  </cms:addArticle>
</env:Body>
</env:Envelope>

--MIME_boundary
Content-Type: application/vnd.oasis.openoffice
Content-Transfer-Encoding: binary
Content-ID: <http://daily-moon.com/tbird.doc>

// binary octets for the word processing file

--MIME_boundary--

```

Observe que la ubicación especificada en el elemento `Include` (Incluir), coincide con la `Content-ID` (ID de contenido), menos el protocolo de `cid`. Ahora es éste el mensaje que se envía, en lugar de un mensaje SOAP de texto común.

SOAP, datos binarios, y Axis2

El proceso de utilización de XOP en documentos SOAP se denomina MTOM (para Mecanismo de optimización de transmisión de mensajes SOAP). Axis2 proporciona soporte para esta forma de trabajo con datos SOAP, pero usted debe asegurarse de configurar la aplicación de manera apropiada.

Específicamente, tendrá que habilitarla en el archivo `axis2.xml` dentro del archivo `axis2.war` (ver Listado 35).

Listado 35. Uso de XOP con Axis2

```

<axisconfig name="AxisJava2.0">
  <!-- ===== -->
  <!-- Parameters -->
  <!-- ===== -->
  <parameter name="hotdeployment" locked="false">true</parameter>

  <parameter name="hotupdate" locked="false">true</parameter>
  <parameter name="enableMTOM" locked="false">true</parameter>

  <!-- Uncomment this to enable REST support -->
  <!-- <parameter name="enableREST" locked="false">true</parameter>-->

  <parameter name="userName" locked="false">admin</parameter>
  <parameter name="password" locked="false">axis2</parameter>
  ...

```

De ser necesario, puede extraer el archivo `axis2.war`, hacer este cambio, y volver a comprimirlo en un `.war`.
reemplazar la aplicación Axis2, visite la consola Geronimo, usando la URL que aparece en el Listado 36.

Listado 36. Consola Geronimo

```
http://localhost:8080/console
```

Inicie sesión como system/manager y haga clic en **Application>Web App WARs**, luego desinstale y reinstale la aplicación Axis2. (Recuerde que tendrá que recargar sus servicios web después de haber realizado este paso.)

El trabajo programático con MTOM está fuera del alcance de este tutorial, pero puede obtener más información sobre este tema en la sección [Temas relacionados](#) (Recursos). Sólo tenga presente que, en versiones de Axis2 anteriores a la 0.95, las cosas podrían no funcionar como se espera, lo que incluye una implementación de SOAP con Adjuntos API para Java (SAAJ) que funcione.

Resumen

En el mundo actual, donde la interoperabilidad entre sistemas es tan importante, los servicios web pueden ser el fundamento de su Arquitectura orientada a los servicios. Y SOAP es el fundamento de los servicios web a nivel de empresa. Este tutorial le mostró los conceptos básicos de los servicios web, y explicó tanto los conceptos como la programación necesaria para comprender y trabajar con SOAP en sus propias aplicaciones. Al mismo tiempo, usted aprendió lo siguiente:

- Los importantes conceptos que rodean a los servicios web
- Cómo instalar y usar el servidor de aplicaciones Geronimo
- Cómo instalar y usar el servidor de aplicaciones Web Axis2
- Cómo crear un cliente para acceder a un servicio SOAP
- Cómo crear un servicio SOAP
- Otros problemas en torno a los servicios SOAP, tales como las solicitudes [GET](#) y los adjuntos

En la Parte 2 de esta serie, Web Services Description Language (Lenguaje de descripción de servicios web), usted aprenderá cómo usar WSDL para automatizar muchos de los pasos que ejecutó aquí, como así también a proporcionar los medios para que otros puedan usar los servicios que usted crea de una manera más sencilla.

Descargas

Descripción	Nombre	tamaño
Source code	ws-understand-web-services1.zip	12KB

Sobre el autor

Nicholas Chase

Nicholas Chase participó en el desarrollo de sitios Web para empresas como Lucent Technologies, Sun Microsystems, Oracle y Tampa Bay Buccaneers. Nick fue profesor de física en escuela secundaria, gerente de instalaciones para tratamiento de desechos con bajo nivel de radioactividad, editor de una revista online de ciencia ficción, ingeniero multimedios, instructor de Oracle y Director de Tecnología en una empresa de comunicaciones interactivas. Es autor de varios libros, entre ellos XML Primer Plus (Sams).

© Copyright IBM Corporation 2011

(www.ibm.com/legal/copytrade.shtml)

Marcas

(www.ibm.com/developerworks/ssa/ibm/trademarks/)