tutorial gSoap

par Rodrigo Pons (home)

Date de publication : 08/08/2009

Dernière mise à jour : 05/07/2010

Les propongo aquí un pequeño tutorial para crear rápidamente un cliente y\o un servidor gSoap stand-alone en C/C ++.
Después de una presentación corta, detallo allí cómo hacer en Windows y en Linux.

volver a la página inicial

I - Preliminares

I-A - Introducción

SOAP es un protocolo de comunicación. Determina el formato de las tramas que van a ser enviadas del cliente al servidor (y viceversa) con el fin de que cada uno pueda reconstituir los datos enviados.

Contrariamente a lo que es escrito a menudo, gSoap no es una librería. En realidad, es un compilador: toma un fichero (.h o .wsdl) y crea los ficheros fuentes (.c/.cpp y .h) que implemientan al servidor y\o el cliente. La parte emergente de gSoap esta compuesta de 2 ejecutables: soapcpp2.exe y wsdl2h.exe.

Hay 2 modos de utilizar gSoap:

1 A partir de un cabecero c/c++ (fichero .h).
2 A partir de un fichero wsdl.

La ventaja del 1. es que es más sencillo: no es necesario conocer a la norma wsdl, y para un desarrollador c/c++ es más "intuitivo" de escribir un cabecero c/c++ que un wsdl.
La ventaja del 2. es que existen unos herramientas para verificar la validez del wsdl, crear un xsd correspondiente, etc. Como el wsdl es xml, es también más fácil para un desarrollador que no gusta/conoce el c/c++.

En ambos casos, el resultado es lo mismo: un conjunto de ficheros .h y .c/.cpp que, una vez compilados, dan un servidor y\o un cliente SOAP.
El tipo de cliente/servidor (stand alone o "agregado" a un servidor existente, versión, etc.) generado depende de los parámetros de compilación pasados a los compiladores wsdl2h y soapcpp2.

En este tutorial, trataré unicamente del servidor de tipo "stand alone", y hablare de la problemática de los proxies.

I-B - Instalación

En primer lugar usted debe descargar gSoap. Este tutoriel es escrito y testado con la versión 2.7.14. gSoap se puede descargar en **la página de descarga de gSoap en sourceforge** y escoja la versión que usted desea. Copia le en la carpeta su elección, y es todo.

I-C - A propósito de soapcpp2.exe

Soapcpp2 es el ejecutable que genera los ficheros fuentes de un cliente y\o de un servidor SOAP a partir de un cabecero (fichero .h).
Ejemplo de utilización:

```
soapcpp2.exe mi_cliente.h
```

Lista principales parámetros de ejecución

- **-C** genera sólo el código para el cliente
- **-S** genera sólo el código para el servidor
- **-L** no genera los ficheros xxxLib (estos ficheros sirven para crear a nuestro cliente/camarero en forma de librería)
- **-c** genera código C (si no, genera código C++)

- **-d path** especifica el lugar donde son generados los ficheros. Si este parámetro no es especificado, los ficheros son generados en la carpeta corriente..

I-D - A propósito de wsdl2h.exe

WSDL es un protocolo que permite definir un web service (la norma wsdl es descrita **aquí**). Escribir un fichero wsdl corresponde a definir precisamente lo que va a hacer este web service.
WSDL es una forma particular de XML.

Wsdl2h es el ejecutable que genera el cabecero de un cliente y\o de un servidor SOAP a partir de un fichero wsdl.
Ejemplo de utilización:

```
wsdl2h.exe mi_cs.wsdl
```

Lista principales parámetros de ejecución

- **-c** genera código C (si no, genera código C++)
- **-l path** va a buscar el(los) fichero(s) wsdl en path
- **-o file** especifica el nombre del cabecero generado
- **-s** no genera el código necesario para la utilización de la stl

II - Tutorial Windows

Para entender cómo funciona gSoap, vamos a crear a un servidor SOAP que resuelve una operación simple, así como su cliente. Esta operación simple, que llamaré *op1*, toma 2 parámetros enteros a y b, y efectúa la operación siguiente: **r = 2a + b**. r es el resultado, es decir el valor que será devuelta al cliente por el servidor.

II-A - A partir de un cabecero .h

Comenzaremos por escribir el cabecero, que define unicamente la firma de nuestra operación. Lo llamaremos mi_cs.h (**mi_c**liente **s**ervidor):

```
// mi_cs.h
int op1( int a, int b, int * r );
```

Atención, las funciones gSoap pueden devolver sólo un entero, que es un código de error. Entonces, el resultado debe ser recuperado por un parámetro pasado en forma de puntero o de referencia. Aquí, el resultado es el tercer parámetro (r).

Luego, vamos a generar el cliente y el servidor. Para hacerlo, aconsejo proceder como sigue:
1. Copiar el ejecutable (soapcpp2.exe) en la misma carpeta que vuestro cabecero.
2. Crear un fichero batch (por ejemplo: mi_cs.bat), en el mismo lugar, y escribir en este fichero el comando que genera el cliente y/o el servidor.
He aquí el batch más simple que le propongo:

```
soapcpp2.exe -L mi_cs.h
pause
```

El -L es para que gSoap no genera los ficheros para utilizar a nuestro cliente/servidor como una biblioteca. Esto no es necesario, pero evita la generación de ficheros que no utilizaremos aquí.
La línea "pausa" es importante: sirve para que la ventana cmd no se cierre inmediatamente para de que podimos verificar que la generación se ha pasado bien, o de ver, en caso de problema, los mensajes de errores.

Si la generación se ha pasado bien, los ficheros siguientes han sido creados:
- soapC.cpp

- soapClient.cpp
- soapServer.cpp
- soapH.h
- soapStub.h

Estos ficheros deberán ser compilados con el resto de su código (salvo soapClient.cpp para el servidor, y soapServer.cpp para el cliente).
Usted también deberá añadir los ficheros stdsoap2.h y stdsoap2.cpp a su proyecto. Estos 2 ficheros se encuentran en la raíz de la carpeta gSoap que usted descargó. Le aconsejo copiar estos 2 ficheros en la carpeta repertorio de su proyecto.

II-A-1 - El servidor

Un servidor gSoap es un web servicie. Entonces se trata de un bucle que espera tramas en un puerto dado. Y cuando una trama es recibida, lo lee y mira lo que puede hacer con ella. Entonces, hay 2 partes en un servidor:
1. El bucle principal
2. La definición de las operaciones

El bucle principal se parecerá a esto:

```
SOAP_SOCKET s; // slave socket
struct soap soap; /* soap structure */
for ( ; ; )
{
 s = soap_accept( &soap ); // Reception de una petición
 std::cout << "conexión con el client valida: slave socket = " << s << std::endl;
 if ( !soap_valid_socket( s ) )
 {
  std::cout << "error: problema de socket" << std::endl << "pulse una tecla para quitar";
  return 1;
 }

 // ejecución de la petición. soap_serve() va a llamar la función correspondiente a la petición.
 soap_serve( &soap );

 // finalización
 soap_end( &soap );
}
```

Luego, las operaciones serán de simple funciones escritas en C ++. Por ejemplo, nuestra op1() será escrita como sigue:

```
int op1( struct soap *soap, int a, int b, int * r )
{
 std::cout << "op1 llamada con: a=" << a << ", b=" << b << std::endl;
 *r = 2 * a + b;
 return SOAP_OK;
}
```

II-A-2 - El cliente

Un cliente SOAP es un programa que envía peticiones a un servidor del que conoce la dirección y el puerto de acceso.
En este tutorial, el cliente es, como el servidor, mínimo y sabe hacer sólo una operación (el que llamo op1).
El cliente pues es muy simple. La unica línea representativa de código es la siguiente:

```
soap_call_op1( &soap, server, "", a, b, &result );
```

soap_call_op1 es una función generada por gSoap a partir de nuestro cabecero. Va a recuperar los parámetros, a crear una o varias tramas correspondientes según el protocolo SOAP, y a enviarlas.

Este cliente se utiliza así
- Abrir una ventana cmd.

- Ir en el repertorio donde es generado el ejecutable del cliente.
- Entre el comando siguiente: cliente *a b* (donde *a* y *b* son enteros).

II-A-3 - Las fuentes

Código fuente del cliente/servidor (con los ficheros de proyecto y soluciones para visual 2008 express ed.):

Código fuente

Esta carpeta contiene el cabecero a partir del cual gSoap va a generar el cliente y el servidor, así como el pequeño script (batch) que lanza esta generación utilizando el ejecutable soapcpp2.
Este ejemplo es mínimo y crea un cliente/servidor que funciona en local.

II-B - A partir de un fichero WSDL

Comenzaremos por escribir el fichero wsdl a partir del cual todo el resto va a ser generado.
La norma WSDL es bastante compleja, porque permite definir totalmente un web service, asi que no voy a restrasarme en eso, pero voy a suministrarles un ejemplo mínimo. Para profundizar, ustedes pueden ir sobre el sitio de W3C donde la norma es descrita exhaustivamente (**aquí**).

II-B-1 - El fichero WSDL

II-B-1-a - Definiciones

En primer lugar, hay que precisar definiciones que van a servir para la generación del cliente/servidor. Siempre utilizo lo mismo, es decir:

```
<definitions name="mi_cs"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:mi_cs"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Solo tienen que reemplazar "mi_cs" por el nombre de su propio cliente/servidor.

II-B-1-b - Tipos

Luego vienen los tipos que usted necesitará. Por ejemplo, si utiliza una estructura compleja para enviar o recibir datos, hay que declararla aquí. En el ejemplo que doy aquí, no utilizo ningún tipo particular, entonces no declaré nada en esta parte. Para más precisión, le reenvío a **la definición de la norma**.

II-B-1-c - Mensajes

Luego vienen las definiciones de los mensajes. Un mensaje definido por una petición y una respuesta. En la petición, hay que especificar los parámetros (eventualmente con tipos definidos en la sección precedente). Ídem para la respuesta.
Por ejemplo, para nuestra operación op1, que toma dos enteros en entrada y que devuelve un entero, dará a esto:

```
<message name="op1Request">
 <part name="a" type="xsd:int"/>
```

```
 <part name="b" type="xsd:int"/>
</message>

<message name="op1Response">
 <part name="r" type="xsd:int"/>
</message>
```

II-B-1-d - Port types

Después vienen las definiciones de los "port types". Se trata de declaración de operaciones virtuales. Es, en cierto modo, una "predeclaración" de las operaciones. Para nuestra operación op1, tendremos esto:

```
<portType name="mi_csPortType">
 <operation name="op1">
  <documentation>2a+b</documentation>
  <input message="tns:op1Request"/>
  <output message="tns:op1Response"/>
 </operation>
</portType>
```

Nota: la baliza <documentation> es opcional.

II-B-1-e - Bindings

Ahora vienen los bindings. Los bindings hacen el lazo entra los port types y las operaciones. Para op1, tendremos:

```
<binding name="mi_cs" type="tns:mi_csPortType">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="op1">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:mi_cs" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:mi_cs" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
  </output>
 </operation>
</binding>
```

II-B-1-f - Services

Y por fin, definimos el service, en el cual especificamos, en particular, la dirección del servidor.

```
<service name="mi_cs">
 <documentation>test of gSoap</documentation>
 <port name="mi_cs" binding="tns:mi_cs">
  <SOAP:address location="http://127.0.0.1"/>
 </port>
</service>
```

II-B-2 - El servidor

Un servidor gSoap es un web servicie. Entonces se trata de un bucle que espera tramas en un puerto dado. Y cuando una trama es recibida, lo lee y mira lo que puede hacer con ella. Entonces, hay 2 partes en un servidor:
1. El bucle principal
2. La definición de las operaciones
El bucle principal se parecerá a esto:

```
SOAP_SOCKET s; // slave socket
struct soap soap; /* soap structure */
for ( ; ; )
{
 s = soap_accept( &soap ); // Reception de una petición
 std::cout << "conexión con el client valida: slave socket = " << s << std::endl;
 if ( !soap_valid_socket( s ) )
 {
  std::cout << "error: problema de socket" << std::endl << "pulse una tecla para quitar";
  return 1;
 }

 // ejecución de la petición. soap_serve() va a llamar la función correspondiente a la petición.
 soap_serve( &soap );

 // finalización
 soap_end( &soap );
}
```

Luego, las operaciones serán de simple funciones escritas en C ++. Por ejemplo, nuestra op1() será escrita como sigue:

```
int op1( struct soap *soap, int a, int b, int * r )
{
 std::cout << "op1 llamada con: a=" << a << ", b=" << b << std::endl;
 *r = 2 * a + b;
 return SOAP_OK;
}
```

II-B-3 - El cliente

Un cliente SOAP es un programa que envía peticiones a un servidor del que conoce la dirección y el puerto de acceso. En este tutorial, el cliente es, como el servidor, mínimo y sabe hacer sólo una operación (el que llamo op1).
El cliente pues es muy simple. La unica línea representativa de código es la siguiente:

```
soap_call_op1( &soap, server, "", a, b, result );
```

soap_call_op1 es una función generada por gSoap a partir de nuestro cabecero. Va a recuperar los parámetros, a crear una o varias tramas correspondientes según el protocolo SOAP, y a enviarlas.

Este cliente se utiliza así
- Abrir una ventana cmd.
- Ir en el repertorio donde es generado el ejecutable del cliente.
- Entre el comando siguiente: cliente *a b* (donde *a* y *b* son enteros).

II-B-4 - Las fuentes

Codifica fuente del cliente/servidor (con ficheros proyecto y solución para visual 2008 express ed.): Source **Código fuente**
Esta carpeta contiene los ficheros fuentes del cliente y del servidor, así como el script (batch) que lanza la generación del código. Esta generación se hace en 2 pasos: 1. generación del cabecero a partir del fichero wsdl. 2. generación del cliente y del servidor a partir de esto cabecero. Este ejemplo es mínimo y creado un cliente/servidor que funciona en local.

II-C - Notas a propósito de las pruebas

Para probar a su cliente y\o servidor, aconsejo pasar por un software externo. En efecto, si nos contentamos de probar a nuestro propio cliente con nuestro propio servidor, y vice-versa, no somos asegurados de ninguna manera

que nuestro servidor funcionará con otro cliente SOAP y respectivamente, que nuestro cliente funcionará con otros servidores SOAP. Par eso, suelo utilizar soapUI, que es gratuito y que usted podrá encontrar **aquí**.

III - Notas para Linux

GSoap funciona exactamente del mismo modo con Windows y Linux.
Entonces, puede leer el tutorial Windows y recuperar sus fuentes, esto será igual, menos los pequeños scripts (batch) de generación que habrá que traducir en su shell preferido.

IV - Enlaces y contacto

Página de descargo de gSoap en sourceforge
Documentación oficial de gSoap
Norma WSDL

Me pueden contactar por mail aquí: pons *punto* rodrigue *at* gmail *punto* com.