UT8. Programación de servicios Web

DAW - Desarrollo Web Entorno Servidor
Fernando Galindo

Contenidos

1. Introducción

Los servicios Web son una herramienta que nos permiten compartir información entre las aplicaciones

Un servicio web es un sistema software diseñado para soportar la interacción máquina-a-máquina, a través de una red, de forma interoperable. Cuenta con una interfaz descrita en un formato procesable por un equipo informático (específicamente en WSDL), a través de la que es posible interactuar con el mismo mediante el intercambio de mensajes SOAP, típicamente transmitidos usando serialización XML sobre HTTP conjuntamente con otros estándares web.

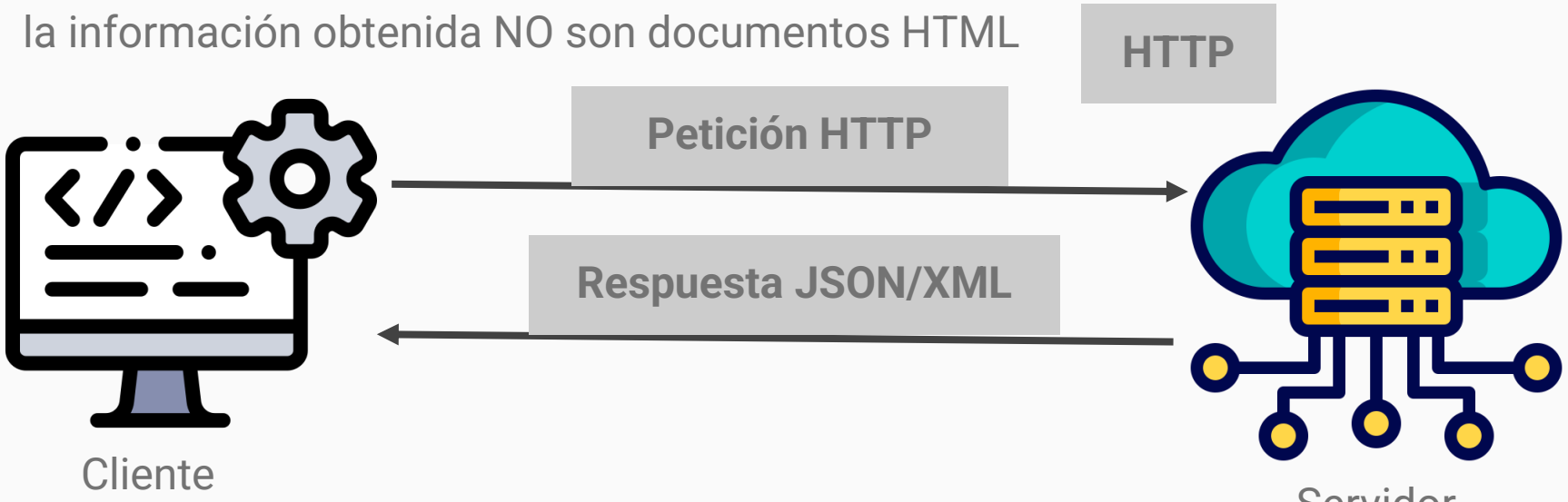
W3C

1. Introducción

Internet dispone de numerosos protocolos para compartir información

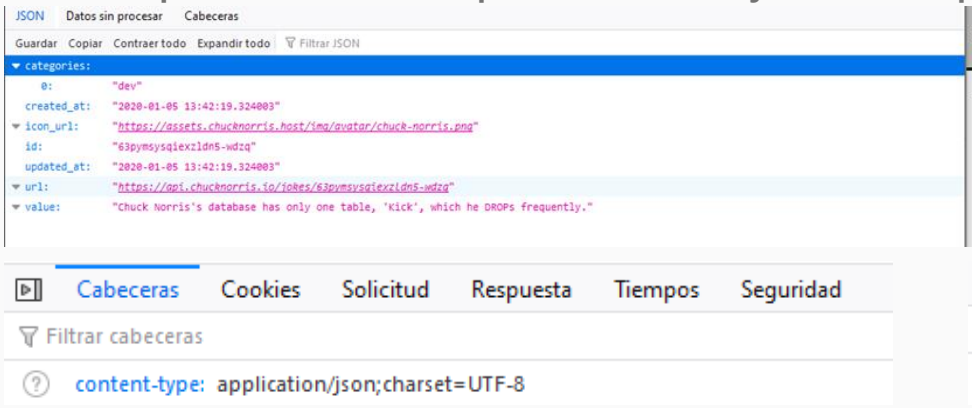
HTTP, FTP, SMTP (e-mail), POP3, Telnet

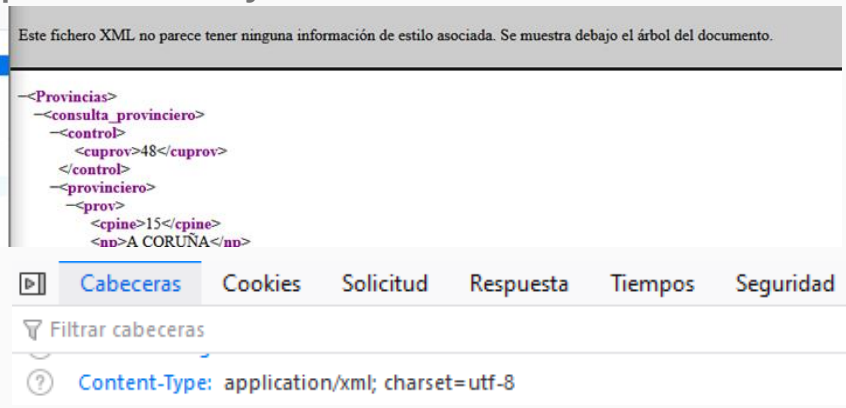
Los servicios Web utilizan el protocolo HTTP para transmitir información, pero la información obtenida NO son documentos HTML



1. Introducción

Las cabeceras de los mensajes de respuesta cobran una mayor importancia, dado que indican el tipo de mensaje con el que se trabaja





Podemos indicar en nuestros mensajes la cabecera con la función header

```
header("Content-Type: application/xml; charset=UTF-8"); // Para mensajes XML

header("Content-Type: application/json; charset=UTF-8"); // Para mensajes json
```

1. Introducción

Los navegadores soportan los mensajes recibidos XML y JSON, siendo capaces de formatear la salida

Disponemos de otras herramientas para intercambiar información sin necesidad de usar el navegador

Una de las herramientas más populares es cURL, es un intérprete en línea de comandos para la transferencia de archivos en internet soportando múltiples protocolos

```
curl [options] [URL...]
```

```
C:\Users\Fernando>curl https://api.chucknorris.io/jokes/random?category=history
{"categories":["history"],"created_at":"2020-01-05 13:42:19.576875","icon_url":"https://assets.chucknorris.host/img/avat
ar/chuck-norris.png","id":"rqcvwdgqq6amwony3nngba","updated_at":"2020-01-05 13:42:19.576875","url":"https://api.chucknor
ris.io/jokes/rqcvwdgqq6amwony3nngba","value":"In the Words of Julius Caesar, \"Veni, Vidi, Vici, Chuck Norris\". Transla
tion: I came, I saw, and I was roundhouse-kicked inthe face by Chuck Norris."}
```

2. URLs y Métodos

Un servicio Web se encuentra dentro de una URL, el servicio que indica esta URL es único para dicha página

Por ejemplo, esta página tiene dos servicios

GET https://api.chucknorris.io/jokes/random

GET https://api.chucknorris.io/jokes/categories

Algunos servicios permiten el paso por parámetros

GET https://api.chucknorris.io/jokes/random?category=history

2. URLs y Métodos

Los servicios Web soportan los métodos GET, PUT, DELETE & POST, de tal forma que indica la operación a realizar

Esta operación es parte de la **cabecera enviada por el cliente** al servicio

- GET método que transmite la información del servicio al cliente, es la operación por defecto
- PUT edita o modifica el recurso identificado por el cliente en la URL
- DELETE elimina un recurso identificado por el cliente en la URL
- POST es similar a PUT

2. URLs y Métodos

Resulta evidente que los navegadores lanzan peticiones GET y los servicios web responden, pero cómo podemos enviar otro tipo de peticiones sin necesidad de usar navegadores

Petición por defecto GET: `curl http://servicio_web.php`

Podemos pasar parámetros por GET: `curl http://servicio_web.php?parametro=valor`

Modificar el tipo de mensaje con la opción -X GET, POST, PUT o DELETE

```
curl -X GET http://servicio_web.php
```

2. URLs y Métodos

Los parámetros de POST lo indicamos con la opción -d, en este caso pasamos 2 parámetros

```
curl -X POST -d "para1=valor1&para2=valor2" http://servicio.php
```

Podemos modificar las cabeceras de la petición de ENVÍO, opción -H

```
curl -X POST -d "para1=valor1" -H "Content-Type: application/x-www-form-urlencoded" http://servicio.php
```

En el caso anterior, estamos indicando que el tipo de información es parte de un formulario POST

Podemos enviar otro tipo de información ficheros JSON, modificando el método de envío (PUT) ajustando la información que enviamos con **-d** y el **content type**

```
curl -X PUT -d '{"campo":"valor"}' -H "Content-Type: application/json" http://servicio.php
```

2. URLs y Métodos

Si utilizamos curl desde la consola de Windows debemos escapar las comillas para que no alteren el formato del fichero JSON

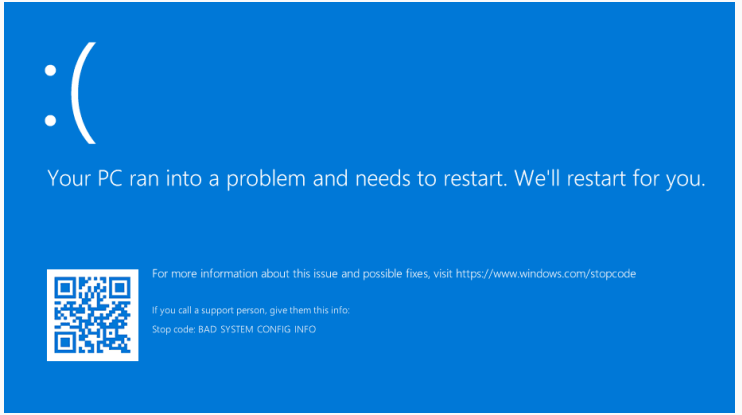
```
curl -X PUT -d "{\"parametro\":\"miValor\"}" http://servicio.php
```

La información JSON puede estar en un fichero y no necesitamos pasarla por comando

```
curl -X PUT -d @fichero.json http://servicio.php
```

Recordar que debemos indicar el Content-Type

```
-H "Content-Type: application/json"
```



2. URLs y Métodos

Existen muchas más opciones y variantes para poder realizar peticiones con CURL

`-o fichero` guarda el resultado de la petición en un fichero

`-d fichero` utiliza el fichero como parámetros a enviar al servicio web

`-u "usuario:password"` credenciales usadas para poder acceder al servicio

`-v` verbose, muestra más información

2. URLs y Métodos

Veamos un ejemplo y utilizaremos los comandos cURL con el siguiente servicio

https://jsonplaceholder.typicode.com/

2. URLs y Métodos

El siguiente punto es cómo podemos construir nuestras aplicaciones para poder definir servicios web

Recordamos que la información de las cabeceras es dónde se define toda la información (metadatos) relevante de nuestra página

```
header("Access-Control-Allow-Origin: *");

header("Content-Type: application/json; charset=UTF-8");

header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE");

header("Access-Control-Allow-Headers: Content-Type, Access-Control-
Allow-Headers, Authorization, X-Requested-With");

//Para CORS header("Access-Control-Allow-Headers: Content-Type,
Access-Control-Request-Method, X-Requested-With, X-API-KEY, Origin,
Accept");
```

2. URLs y Métodos

Desde PHP podemos obtener el método de petición de la página (GET, POST, PUT & DELETE) a través de la variable

```
$method = $_SERVER['REQUEST_METHOD'];

switch ($method) {
    case 'POST':
        echo "Tratamos POST";
        break;
    case 'PUT':
        echo "Tratamos PUT";
        break;
    case 'DELETE':
        echo "Tratamos DELETE";
        break;
    default: //Suponemos que por defecto es una petición GET
        echo "Opción por defecto GET";
        break;
}
```

2. URLs y Métodos

Los parámetros pasados al servicio web pasadas por los métodos GET y POST se acceden de la misma forma que los formularios

`$_GET['variable']` y `$_POST['variable']`

Resulta conveniente utilizar la función `$empty($_GET['variable']`para comprobar que el parámetro está definido para poder utilizarlo

2. URLs y Métodos

En el caso de que los métodos sean diferentes a GET y POST, debemos procesar la entrada, una forma posible sería

```
$entrada=file_get_contents('php://input');
```

En este momento la variable `$entrada` son los datos que han introducido para realizar la llamada y deberíamos procesarlo de la forma adecuada dependiendo del tipo de dato, array(implode), xml, JSON…

```
$array_json=json_decode($entrada,true);
```

Con el parámetro true, convertimos la entrada en un array asociativo y podemos acceder a cada uno de los campos como un array

Opción 2 - El resultado lo tenemos en `$put_cadena`

```
$put_fichero = fopen('php://input', 'r');
$put_cadena = '';
while($datos = fread($put_fichero, 1024))
    $put_cadena .= $datos;
fclose($put_fichero);
```

Posteriormente la cadena tendría que tratarse en función del tipo de dato

3. Servicios SOAP

SOAP (Simple Object Access Protocol) es un protocolo para el intercambio de información mediante mensajes XML, siendo otro de los protocolos utilizados para los servicios Web

SOAP trabaja con una arquitectura cliente servidor, donde el cliente manda una petición (mensaje XML) y el servidor responde con otra petición (XML)

Petición

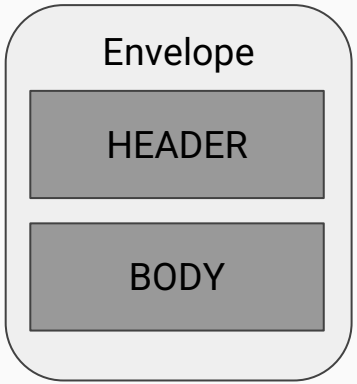
```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body>
  <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

Respuesta

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body>
  <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

Fuente: https://www.w3schools.com/xml/xml_soap.asp

En vista a los ejemplos anteriores, podemos apreciar que los mensajes tienen una estructura particular



Envelope: Es el nodo raíz del documento XML y tiene 2 hijos (Header y Body)
Header: Es un campo opcional, contiene información relacionada con el mensaje a procesar
Body: Es un campo obligatorio, contiene la información que debe ser manejada por el receptor
Fault: Es un hijo de Body, que contiene información sobre los errores

XSD Definición mensaje SOAP:
https://www.w3.org/2003/05/soap-envelope/

3. Servicios SOAP

Los servicios Web normalmente están asociados a un **WSDL**

WSDL (*Web Service Description Language*) corresponde a la definición de un servicio Web los funciones admitidas, tipos de mensajes, etc.

Los WSDL están disponibles a través de la URL del servicio añadiendo el parámetro **GET ?wsdl**

http://servidor/servicioSOAP**?wsdl**

La estructura de un documento WSDL es la siguiente:

- **types**. Definiciones de los tipos de datos que se usan en el servicio
- **message**. Conjuntos de datos, como la lista de parámetros que recibe una función o los valores que devuelve
- **portType**. Cada portType son las funciones que implementa el servicio web.
- **binding**. Define cómo va a transmitirse la información de cada portType.
- **service**. Contiene una lista de elementos de tipo port. Cada port indica dónde (en qué URL) se puede acceder al servicio web.

XSD Definición WSDL: https://schemas.xmlsoap.org/wsdl/

3. Servicios SOAP

Veamos algunos servicios Web SOAP:

https://www.w3schools.com/xml/tempconvert.asmx

http://www.dneonline.com/calculator.asmx

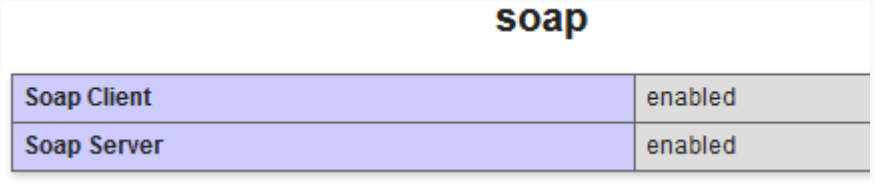
http://ovc.catastro.meh.es/ovcservweb/OVCSWLocalizacionRC/OVCCallejero.asmx

https://pagosonline.redsys.es/entornosPruebas.html

Una vez que conocemos WSDL para definir servicios SOAP, vamos a definir nuestros propios servicios en PHP

Deberemos habilitar la extensión SOAP de php.ini (función `phpinfo()`)

```
extension=soap
```



soap

Soap Client	enabled
Soap Server	enabled

Recomendable modificar el parámetro de 1 a 0, para deshabilitar la caché de los documentos WSDL y que los cambios que hagamos se reflejen automáticamente

```
soap.wsdl_cache_enabled=0
```

3. Servicios SOAP

La extensión entre otros nos habilita dos roles para actuar como:

- CLIENTE, nos conectamos contra un servicio para lanzar peticiones
- SERVIDOR, definimos un servicio al que se pueden conectar clientes

La utilización de un cliente es relativamente sencilla

```
$url          = "https://www.w3schools.com/xml/tempconvert.asmx";

$cliente      = new SoapClient($url);
```

El objeto cliente dispone de diversos métodos, y ver los métodos que dispone el servicio web

```
var_dump($cliente->__getFunctions());
```

Veamos un ejemplo concreto de una aplicación

```
<?php

$url =
"https://www.w3schools.com/xml/tempconvert.asmx?wsdl";

$cliente = new SoapClient($url);

var_dump($cliente->__getFunctions());

$parametro=array("Celsius"=>"50");

var_dump($cliente->CelsiusToFahrenheit($parametro));

?>
```

Identificar los parámetros de la aplicación y su estructura es lo más importante a la hora de realizar las llamadas

El resultado será un tipo de dato que deberemos manejar y tratar de forma adecuada, en este caso un objeto

3. Servicios SOAP

La definición de un servidor SOAP consta de 2 partes:

- Definición de una clase con las funciones que procesa nuestro servicio SOAP
- Definición del servicio SOAP que atenderá las peticiones de los clientes

Parte 1 - Calculadora.php

Definimos una clase con nuestras funciones, parámetros y valores que devuelve

```php
<?php
class Calculadora{
    public function suma($a, $b){
        return $a + $b;
    }
    public function resta($a, $b){
        return $a - $b;
    }
    public function multiplica($a, $b){
        return $a * $b;
    }
    public function divide($a, $b){
        return $a / $b;
    }
}
```

Parte 2 - ws_soap.php

Importamos la clase donde se han definido las funciones, creamos un servidor SOAP al que indicamos la clase anterior

```php
<?php
require_once ('Calculadora.php');
$server = new SoapServer(null, array('uri' => ''));
$server->setClass('Calculadora');
$server->handle();
```

3. Servicios SOAP

Como no tenemos creado aún un WSDL para nuestro servicio no lo podemos usar para crear nuestro cliente

Disponemos de un constructor en el que no necesitamos definir un WSDL sino la localización del servicio y dónde se encuentra

Las llamadas a las funciones y manejo de resultados siguen la misma dinámica que en los casos anteriores

```php
<?php
$cliente = new SoapClient(null,
    array('location' => "http://dominio/serverSOAP.php",
      'uri' => "http://dominio/serverSOAP.php"));
echo $cliente->suma(5,3);
```

La creación del documento WSDL necesitamos documentar nuestra clase similar a como se hace en Java con Javadocs

https://www.phpdoc.org/

https://manual.phpdoc.org/HTMLSmartyConverter/PHP/

Usaremos las estructuras básicas para documentar los parámetros de entrada y retorno de nuestras clases

```php
/**
 * Suma dos números y devuelve el resultado
 * @param float $a
 * @param float $b
 * @return float
 */
public function suma($a, $b){
    return $a + $b;
}
```

Los tipos que podemos usar a la hora de realizar llamadas o devolver operaciones están limitados a los siguientes:

- integer
- float
- string
- boolean
- array

Podemos añadir un comentario a nuestra clase indicando el autor

```
/** Clase Calculadora *
* Desarrollo Web en Entorno Servidor
* Tema 6: Servicios web
* Generación automática del documento WSDL
* @author Fernando Galindo */
```

Una vez tenemos documentada nuestra clase, nos apoyaremos en el siguiente script que generará el WSDL a partir de la clase

https://code.google.com/archive/p/wsdldocument/

El programa necesita alguna pequeña adaptación para que funcione correctamente en las nuevas versiones de PHP

```php
<?php
header("Content-Type: application/xml; charset=UTF-8");
require_once("Calculadora.php");
require_once("WSDLDocument.php");
$wsdl = new WSDLDocument( "Calculadora", "http://dominio/servicio.php",
"http://dominio/" );
echo $wsdl->saveXml();
```

Lo único que queda para completar nuestro servicio SOAP es asegurar que manejamos correctamente el parámetro GET para mostrar el WSDL

3. Servicios SOAP

```php
<?php
if(isset($_GET["wsdl"])){
    //Generamos el WSLD cada vez que llamamos a la aplicación
    header("Content-Type: application/xml; charset=UTF-8");
    require_once("Calculadora.php"); // Ruta a WSDLDocument
    require_once("WSDLDocument.php");
    $wsdl = new WSDLDocument( "Calculadora", "http://localhost/server.php",
"http://localhost/" );
    echo $wsdl->saveXml();
}else{
    //El código de nuestro servicio SOAP
    require_once ('Calculadora.php');
    $server = new SoapServer(null, array('uri' => ''));
    $server->setClass('Calculadora');
    $server->handle();
}
```

3. Servicios SOAP

```php
<?php
if(isset($_GET["wsdl"])){
    //Asumimos que tenemos un fichero con nuestro WSDL almacenado
    header("Content-Type: application/xml; charset=UTF-8")
    $xml=simplexml_load_file("definicionServicio.wsdl");
    echo $xml->saveXML()."\n";
}else{
    //El código de nuestro servicio SOAP
    require_once ('Calculadora.php');
    $server = new SoapServer(null, array('uri' => ''));
    $server->setClass('Calculadora');
    $server->handle();
}
```