Module M2105
Web dynamique
Partie 4 : Services WEB

IUT de Béziers, dépt. R&T © 2014-2021
http://www.borelly.net/
Christophe.BORELLY@umontpellier.fr

Généralités

- « Un service web est un programme permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. »

Sigles

- **XML-RPC** (XML - Remote Procedure Call)
- SOAP (Simple Object Access Protocol)
- WSDL (Web Service Description Language)
- UDDI (Universal Description Discovery and Integration)
- RDF (Resource Description Framework)
- **REST** (Representational state transfer)
- **JSON** (JavaScript Object Notation)
- **AJAX** (Asynchronous JavaScript And XML)
- **FETCH** (Nouvelle API sur les navigateurs)
- ...

XML-RPC

- Transport des messages en HTTP
- Types de données :
  - <**boolean**> : valeurs 0 ou 1
  - <i4> ou <**int**> : entier signé sur 4 octets
  - <**double**> : réel double précision
  - <**string**>
  - <**dateTime.iso8601**> : date et heure
  - <**base64**> : binaire encodé en base64

Type structure

```
<struct>
 <member>
  <name>age</name>
  <value><int>20</int></value>
 </member>
 <member>
  <name>nom</name>
  <value><string>DOE</string></value>
 </member>
</struct>
```

Type Array

```
<array>
 <data>
  <value><i4>12</i4></value>
  <value>
    <string>bla bla</string>
  </value>
  <value>
    <boolean>0</boolean>
  </value>
  <value><i4>-10</i4></value>
 </data>
</array>
```

Requête XML-RPC

```
POST /xml/rpc.php HTTP/1.0
User-Agent: Firefox/36.0
Host: www.borelly.net
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>hello</methodName>
  <params>
   <param>
    <value><string>John DOE</string></value>
   </param>
  </params>
</methodCall>
```

Réponse XML-RPC

```
HTTP/1.1 200 OK
Content-Length: 158
Content-Type: text/xml
Date: Tue, 24 Mar 2015 15:20:28 GMT
Server: R&T SIOUX Server 1.1.0

<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
   <value><string>Hello John DOE</string></value>
  </param>
 </params>
</methodResponse>
```

Serveur XMLRPC en PHP

```
...
$xmlrpc_server=xmlrpc_server_create();
xmlrpc_server_register_method($xmlrpc_server,'moyenne','calcul');
header('Content-Type: text/xml; charset=utf-8');
$method=null;
$r=xmlrpc_decode_request($request_xml,$method);
if ($method!='moyenne') {
  print xmlrpc_encode(array('faultCode'=>42,
                            'faultString'=>'Unknown method !'));
}
else {
  $userData=array();
  $options=array('encoding'=>'utf-8');
  print xmlrpc_server_call_method($xmlrpc_server,$request_xml,
                                  $userData,$options);
}
```

Client XMLRPC en PHP

```php
$params=array('John DOE',array(10.5,15.4,16.3));
$options=array('encoding'=>'utf-8');
$request=xmlrpc_encode_request('moyenne',$params,$options);
printf("XMLRPC-REQUEST:\n%s\n",$request);
$serverURL='http://127.0.0.1/M2105/xmlrpc-server-moy.php';
$context=stream_context_create(array('http'=>array(
  'method'=>'POST',
  'header'=>"Content-Type: text/xml\r\nUser-Agent: PHPRPC/1.0\r\n",
  'content'=>$request
)));
$file=file_get_contents($serverURL,false,$context);
printf("XMLRPC-RESPONSE:\n%s\n",$file);
$response=xmlrpc_decode($file);
if (is_array($response)&&xmlrpc_is_fault($response)) {
  fprintf(STDERR,"XMLRPC(%d) : %s\n",$response['faultCode']
                                    ,$response['faultString']);
} else {
  printf("Moyenne de [%s] : %s\n",$response['user'],$response['moy']);
}
```

REST

- Les services REST exposent leur fonctionnalités avec des URI par l'intermédiaire du protocole HTTP (GET, POST ; PUT, DELETE, …)
- Service sans états (une requête doit contenir toutes les informations nécessaires pour obtenir le résultat)
- Service qui peut être mis en cache (comme les documents WEB)

URI REST

- Format d'URI :
  - http://ServiceName/ResourceType/ResourceID
- Préférable de ne pas utiliser des paramètres pour éviter une trop grande complexité :
  - ~~http://MyService/Persons/1?format=json&encoding=UTF8~~
  - http://MyService/Persons/1/json/UTF8

AJAX

- Asynchronous JavaScript And XML
- Requête HTTP Asynchrone permettant d'envoyer ou recevoir des données pour « dynamiser » le document
- Utilise l'objet **XMLHttpRequest**
  - Créé en 1998 par Microsoft comme un composant ActiveX et ajouté au JavaScript des navigateurs (ECMAScript - European Computer Manufacturers Association) à partir de 2002

AJAX et JSON

- Maintenant, on utilise plutôt le format **JSON** (JavaScript Object Notation) pour recevoir des données à la place de XML.
- Un objet JavaScript est défini avec des **{}** et contient des propriétés et des valeurs :

```
var obj={prenom:"John",nom:"DOE"};
obj.age=42;
obj.infos=[5.2,10,"10h45"];
//Tableau
```

JSON

- JavaScript Object Notation (RFC 4627 et 7159)
- Utilisé généralement avec AJAX
  - Asynchronous Javascript And XML
- Contient des paires nom/valeur ou bien des listes ordonnées de valeurs
- Permet de représenter des objets ({}), des tableaux ([]) ou des valeurs génériques (string, number, object, array, true, false, null)
- MIME : application/json

Format JSON

- JSON = ws value ws
- ws (whitespace) = SPACE, TAB, CR, LF
- value = null, false, true, number, string,
  array, object
- array = **[** (value (**,** value)*)? **]**
- object = **{** (member (**,** member)*)? **}**
- member = string **:** value

Exemple XML - JSON

```
<list>
 <item>
  <id>1234</id>
  <nom>toto</nom>
 </item>
 <item>
  <id>4567</id>
  <nom>titi</nom>
 </item>
</list>
```

```
{ "list" : [
             {"id":1234,"nom":"toto"},
             {"id":4567,"nom":"titi"}  ] }
```

AJAX en JS

```
var request=new XMLHttpRequest();
request.onreadystatechange=function() {
 if (request.readyState===4 && request.status===200) {
  var type=request.getResponseHeader("Content-Type");
  if (type.match(/application\/json/)) {
    var jsonObj=JSON.parse(request.responseText);
    ...
  }
 }
}
request.open("GET","http://example.com/ajax.php");
request.send();
```

AJAX avec JQuery

- Plusieurs méthodes:

`.load( url [, data ] [, complete ] )`

  - Charge les données de l'URL et les copies dans la partie HTML des éléments sélectionnés.
  - https://api.jquery.com/load/

`jQuery.get( url [, data ] [, success ] [, dataType ] )`

  - Télécharge les données de l'URL en HTTP GET
  - dataType : xml, json, script, text, html
  - https://api.jquery.com/jQuery.get

`jQuery.getJSON( url [, data ] [, success ] )`

  - https://api.jquery.com/jquery.getjson/

`jQuery.ajax( url [, settings ] )`

  - https://api.jquery.com/jQuery.ajax/

AJAX avec jQuery (1/4)

```
$("#m1").load("http://srv.fr/ajax1.php");

$("#m2").load("http://srv.fr/ajax2.php",
    function(responseTxt,statusTxt,xhr) {
      if(statusTxt==="success") {
        $(this).html(responseTxt);
      }
      if(statusTxt==="error") {
        $(this).html("Load error!");
      }
    });
```

AJAX avec jQuery (2/4)

```
$.get("http://srv.fr/ajax1.php")
  .done(function(data) {$("#m1").html(data);})
  .fail(function() {
     $("#m1").html("GET ERROR !!!");
  });

$.get("http://srv.fr/ajax2.php",null,null,"html")
  .done(function(data) {$("#m2").html(data);})
  .fail(function() {
     $("#m2").html("GET ERROR !!!");
  });
```

AJAX avec jQuery (3/4)

```
$.getjson("http://srv.fr/ajax1.php")
   .done(function(jsonObj) {
      $("#m1").html(JSON.stringify(jsonObj));
   })
   .fail(function() {
      $("#m1").html("GETJSON ERROR !!!");
   });

$.post("http://srv.fr/ajax2.php",{id:12})
   .done(function(data) {$("#m2").html(data);})
   .fail(function() {
      $("#m2").html("POST ERROR !!!");
   });
```

AJAX avec jQuery (4/4)

```
var jsonQuery={id:18};
$.ajax({
  url : "http://srv.fr/ajax.php",
  data : jsonQuery,
  method : "POST",
  dataType : "json"
  })
  .done(function(jsonObj) {...})
  .fail(function() {...});
```

API Fetch

- Fetch est une API des navigateurs pour charger des textes, images, données structurées, de façon asynchrone pour mettre à jour une page HTML.
- Fetch est construit sur l'objet **Promise** ce qui simplifie beaucoup le code, surtout si on l'emploi en conjonction avec `async/await`.

Exemple Fetch

```
fetch("http://srv.fr/ajax.php")
    .then(response => response.json())
    .then(jsonObj => {...JSON.stringify(jsonObj)...})
    .catch(error => {...});
```

- Version asynchrone :

```
async function getValue() {
  var r=null;
  await fetch("http://srv.fr/ajax1.php")
    .then(response => response.json())
    .then(jsonObj => r=jsonObj.xxx)
    .catch(error => {...});
  return r;
}
```

Javascript - XML (1)

```
var parser=new DOMParser();
var xmlText="<bookstore><book><title
id='18'>XML</title></book><book><title
id='45'>Javascript</title></book><book><title
id='323'>JSON</title></book></bookstore>";
var xmlDoc=parser.parseFromString(xmlText,"text/xml");

var x1=xmlDoc.getElementById("323")
          .childNodes[0].nodeValue; // JSON
x1=xmlDoc.getElementsByTagName("title")[1]
          .childNodes[0].nodeValue; // Javascript
x1=xmlDoc.getElementsByTagName("title")[1]
          .attributes[0].value; // 45
x1=xmlDoc.getElementsByTagName("title")[1]
          .getAttribute("id"); // 45
```

Javascript - XML (2)

https://developer.mozilla.org/fr/docs/Web/API/Document/evaluate

```
document.evaluate(xpathExpression,contextNode,
namespaceResolver,resultType,result);
```

- ANY_TYPE
- NUMBER_TYPE, STRING_TYPE, BOOLEAN_TYPE
- UNORDERED_NODE_ITERATOR_TYPE, ORDERED_NODE_ITERATOR_TYPE
- UNORDERED_NODE_SNAPSHOT_TYPE, ORDERED_NODE_SNAPSHOT_TYPE
- ANY_UNORDERED_NODE_TYPE, FIRST_ORDERED_NODE_TYPE

```
// Syntaxe XPath
var nb=xmlDoc.evaluate('count(//book)',xmlDoc,null,
                       XPathResult.ANY_TYPE,null);
console.log("Nombre de livres: "+nb.numberValue);
```

Javascript - XML (3)

```
var t=xmlDoc.evaluate('//title/@id',xmlDoc,null,
                      XPathResult.ANY_TYPE,null);
try {
  var thisNode=t.iterateNext();
  while (thisNode) {
    console.log("ID: "+thisNode.textContent);
    thisNode=t.iterateNext();
  }
} catch (e) {console.log('Erreur : ' +e);}
////////////////////////////////////////////////
var t2=xmlDoc.evaluate('//title',xmlDoc,null,
                       XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,null);
for (var i=0;i<t2.snapshotLength;i++){
  console.log("Titre: "+t2.snapshotItem(i).textContent);
}
////////////////////////////////////////////////
var t3=xmlDoc.evaluate('//title',xmlDoc,null,
                       XPathResult.FIRST_ORDERED_NODE_TYPE,null);
console.log("Titre: "+t3.singleNodeValue.textContent);
```

Cross-Origin Resource Sharing (CORS)

- https://developer.mozilla.org/fr/docs/Web/HTTP/CORS
- Partage de ressources entre origines multiples
  - Ajouter des en-têtes HTTP afin d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.
  - XMLHttpRequest et l'API Fetch respectent la règle d'origine unique (sauf si des en-têtes CORS sont utilisés par la cible : `Access-Control-Allow-Origin`).

PHP - JSON

- Depuis PHP 5.2.0

```php
header('Access-Control-Allow-Origin: *'); // CORS
header('Content-Type: application/json;
                      charset=UTF-8');
$data=...; // Array, Object, ...
echo json_encode($data);
exit(0);


$obj=json_decode($strJSON); // Object

$tab=json_decode($strJSON,true); // Array
```

Références

- http://fr.wikipedia.org/wiki/Service_web
- http://fr.wikipedia.org/wiki/Representational_State_Transfer
- http://en.wikipedia.org/wiki/JSON