Desenvolvimento de Aplicações Web baseadas em SOA e AJAX

Francisco M. Couto

DI–FCUL–LO–2010

DOI:10455/3334

(http://hdl.handle.net/10455/3334)

21 de Fevereiro de 2012

Desenvolvimento de Aplicações Web baseadas em SOA e AJAX

Francisco M. Couto

21 de Fevereiro de 2012

Conteúdo

1

Introdução

Este manual tem como objectivo apoiar o desenvolvimento de aplicações web baseados na arquitectura SOA e no uso de técnicas AJAX.

Este documento introduz os conceitos básicos explicados através de exemplos, que embora tenham sido testados na infra-estrutura informática do Departamento de Informática da FCUL podem ser facilmente adaptados a qualquer outra infra-estrutura que possua a mesma tecnologia.

O código aqui apresentado está disponível em `http://dx.doi.org/10455/3334` em ficheiro anexo protegido por uma senha que deverá requisitar ao autor deste documento.

1.1 Pré-requisitos

Este documento pressupõe que o leitor já tem conhecimentos básicos de sistemas de informação, assim como de algumas tecnologias web básicas. Caso não as possua recomenda-se a leitura prévia dos seguintes manuais:

- Sistemas de Informação: `http://dx.doi.org/10455/3167`

- HTML: `http://www.w3schools.com/html/`

- HTML: http://www.w3schools.com/htmldom/
- XML: http://www.w3schools.com/xml/
- RSS: http://www.w3schools.com/rss/
- XHTML: http://www.w3schools.com/xhtml/
- JavaScript: http://www.w3schools.com/js/

1.2 SOA

SOA (Service Oriented Architecture[1] é um estilo de arquitectura de software que modela as várias componentes do sistema pelas funcionalidades que implementam em forma de serviços (Web Services). A implementação de aplicações complexas envolve a interligação entre as várias componentes através da invocação destes serviços. Web Services[2] podem-se considerar como API[3] que permite esta interligação entre aplicações de software heterogéneas através do protocolo HTTP[4] usado na Web. As mensagens trocadas pelos Web Services usam o formato XML[5], e podem seguir diferentes standards, tais como SOAP[6], REST[7], ou JSON[8].

[1] http://en.wikipedia.org/wiki/Service-oriented_architecture
[2] http://en.wikipedia.org/wiki/Web_services
[3] http://en.wikipedia.org/wiki/Api
[4] http://en.wikipedia.org/wiki/HTTP
[5] http://en.wikipedia.org/wiki/XML
[6] http://en.wikipedia.org/wiki/SOAP
[7] http://en.wikipedia.org/wiki/REST
[8] http://en.wikipedia.org/wiki/JSON

Através da definição das interfaces dos Web Services, que pode por exemplo ser descrita em WADL[9], podem-se criar facilmente aplicações Mashup[10], que combinam as funcionalidades e informação de diversos serviços disponíveis, disponibilizando assim um serviço mais preciso e completo.

1.3 AJAX

AJAX[11] (asynchronous JavaScript and XML) agrega um conjunto de técnicas de desenvolvimento web aplicadas ao Front-end[12]. A grande vantagem do AJAX é comunicação com os Web services de uma forma assíncrona através do objecto XMLHttpRequest[13]. Isto permite aumentar o dinamismo e interactividade das interfaces com o utilizador.

1.4 Outros Manuais

Este documento apresenta apenas uma introdução aos conceitos, existem muitas mais funcionalidades que aqui não são abordadas e que devem ser consultados em:

- `http://www.w3schools.com/webservices/`

- `http://www.w3schools.com/xpath/`

- `http://www.w3schools.com/ajax/`

[9] `http://en.wikipedia.org/wiki/WADL`
[10] `http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)`
[11] `http://en.wikipedia.org/wiki/Ajax_(programming)`
[12] `http://en.wikipedia.org/wiki/Ajax_(programming)`
[13] `http://en.wikipedia.org/wiki/XMLHttpRequest`

1.5 Próximos Capítulos

Os capítulos seguintes irão apresentar através de exemplos a forma como construir uma aplicação web baseado na tecnologia aqui referida. O capítulo 2 irá apresentar exemplos de clientes de Web Services. O capítulo 3 irá apresentar um exemplo de um web service RESTful. O capítulo 4 irá apresentar exemplos de Mashups. O capítulo 5 irá apresentar exemplos de AJAX.

2

Web Services Clients

Este capítulo tem por objectivo dar uma introdução dos conceitos básicos para a utilização de web services. Recorre a exemplos baseados nas API dos seguintes web services:

- Google Maps: `http://www.google.com/apis/maps/`
- Flickr: `http://www.flickr.com/services/api/`
- YQL: `http://developer.yahoo.com/yql/`

2.1 Google Maps

O Google Maps é um serviço da Google que oferece tecnologia de mapeamento potente e de fácil utilização, bem como informações de negócios locais - incluindo localizações de empresas, informações de contacto e direcções de condução (ver a descrição do serviço para informação adicional).

A Google Maps API permite-lhe inserir mapas em páginas web, e fornece uma vasta lista de funções utilitárias para manipulação dos mapas.

Começe por usar o seguinte código para criar uma página web que apresenta um mapa centrado em Lisboa:

HTML

```
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=
    true"></script>
<script type="text/javascript">
  function initialize() {
    var latlng = new google.maps.LatLng(38.71284,-9.135475);
    var myOptions = {
      zoom: 8,
      center: latlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var map = new google.maps.Map(document.getElementById("map_canvas"),
        myOptions);
  }

</script>
</head>
<body onload="initialize()">
  <div id="map_canvas" style="width:100%; height:100%"></div>
</body>
</html>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/google_maps/`

O significado deste código está disponível em: `http://code.google.com/apis/maps/documentation/v3/introduction.html`.

Para melhor entender que opções estão disponíveis veja e implemente os exemplos em: `http://code.google.com/apis/maps/documentation/v3/examples/index.html`

2.2 Flickr

O Flickr é um sítio web de partilha de fotos (ver a descrição do serviço para informação adicional).

Os Flickr Services disponibilizam uma série de funcionalidades para gestão das contas e incluem também métodos para aceder, actualizar, e inserir imagens (fotográficas) e informação referente a estas.

Para testar poderá usar nas chamadas aos web services a seguinte chave (relativa ao user=difcul com password=fculdi):

HTML

```
fd3fabe4e9d94ca725df1de71b8d285b
```

Abaixo tem um exemplo em PHP de acesso a um web service REST para verificar o título de uma determinada foto:

PHP

```
<?php
# build the API URL to call
$params = array(
                'api_key'=> 'fd3fabe4e9d94ca725df1de71b8d285b',
                'method'=> 'flickr.photos.getInfo',
                'photo_id'=> '251875545',
                'format'=> 'php_serial',
                );
$encoded_params = array();
foreach ($params as $k => $v){
  $encoded_params[] = urlencode($k).'='.urlencode($v);
}

# call the API and decode the response
$url = "http://api.flickr.com/services/rest/?".implode('&', $encoded_params);
```

```
$rsp = file_get_contents($url);
$rsp_obj = unserialize($rsp);

# display the photo title (or an error if it failed)
if ($rsp_obj['stat'] == 'ok'){
  $photo_title = $rsp_obj['photo']['title']['_content'];
  echo "Title is $photo_title!";
 }else{
  echo "Call failed!";
}
?>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/flickr/`

O Flickr disponibiliza vários formatos para os pedidos e respectivas respostas

Formato dos pedidos:

- REST: `http://www.flickr.com/services/api/request.rest.html`
- XML-RPC: `http://www.flickr.com/services/api/request.xmlrpc.html`
- SOAP: `http://www.flickr.com/services/api/request.soap.html`

Formato das respostas:

- REST: `http://www.flickr.com/services/api/response.rest.html`
- XML-RPC: `http://www.flickr.com/services/api/response.xmlrpc.html`

- SOAP: `http://www.flickr.com/services/api/response.soap.html`

- JSON: `http://www.flickr.com/services/api/response.json.html`

Ou seja, existem várias formas de invocar o mesmo método.

2.3 YQL

O YQL (*Yahoo! Query Language*) é uma linguagem tipo SQL que permite interrogar, filtrar e juntar dados de vários web services.

Por exemplo, abaixo tem um exemplo em PHP de acesso ao serviço de musicas do Yahoo! (`http://music.yahoo.com/`), que devolve a lista de albuns mais populares actualmente.

PHP

```
<?php
  // execute query
  // get list of 15 most popular music releases
  // retrieve result as SimpleXML object
$xml = simplexml_load_file('http://query.yahooapis.com/v1/public/yql?q=SELECT␣*␣
    FROM␣music.release.popular');

// iterate over query result set
echo '<h2>Popular␣Music</h2>';
$results = $xml->results;
foreach ($results->Release as $r) {
  echo '<p>';
  echo '<a␣href="' . $r['url'] . '">' . $r['title'] .
    '</a>␣(' . $r['releaseYear'] . ')␣-␣';
  echo '<a␣href="' . $r->Artist['url'] . '">' . $r->Artist['name'] .
    '</a>␣<br/>';
```

```
  echo 'Current chart position: ' . $r->ItemInfo->ChartPosition['this'] .
    ' / Last chart position: ' . $r->ItemInfo->ChartPosition['last'];
  echo '</p>';
}
?>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/yql/music.php`

Outro exemplo também em PHP de acesso ao serviço de eventos do Yahoo! (`http://events.yahoo.com/`) é apresentado abaixo, que devolve a lista de conferências em Berlin.

PHP

```
<?php
  $BASE_URL = "https://query.yahooapis.com/v1/public/yql";
  $location = 'berlin';
  $query = 'conference';
  $events="";

  // Form YQL query and build URI to YQL Web service
  $yql_query = "select * from upcoming.events where location='$location' and
      search_text='$query'";
  $yql_query_url = $BASE_URL . "?q=" . urlencode($yql_query) . "&format=json";

  // Make call with cURL
  $session = curl_init($yql_query_url);
  curl_setopt($session, CURLOPT_RETURNTRANSFER,true);
  $json = curl_exec($session);
  // Convert JSON to PHP object
  $phpObj = json_decode($json);

  // Confirm that results were returned before parsing
  if(!is_null($phpObj->query->results)){
```

```
    // Parse results and extract data to display
    foreach($phpObj->query->results->event as $event){
      $events .= "<div><h2>" . $event->name . "</h2><p>";
      $events .= html_entity_decode(wordwrap($event->description, 80, "<br/>"));
      $events .="</p><br/>$event->venue_name<br/>$event->venue_address<br
          />";
      $events .="$event->venue_city, $event->venue_state_name";
      $events .="<p><a href=$event->ticket_url>Buy Tickets</a></p></div>";
    }
  }
  // No results were returned
  if(empty($events)){
    $events = "Sorry, no events matching $query in $location";
  }
  // Display results and unset the global array $_GET
  echo $events;
?>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/yql/events.php`

Exemplos de outras funcionalidades pode encontrar em `http://developer.yahoo.com/yql/guide/yql-code-examples.html` ou em `http://www.ibm.com/developerworks/opensource/library/x-yqlphp1/index.html`.

3

Web Services

Este capítulo tem por objectivo dar uma introdução dos conceitos básicos para a criação de um web service RESTful em PHP.

Iremos apresentar um web service que permite guardar e devolver mensagens em formato XML ou JSON.

3.1 Web Service

O script começa por utilizar a classe XML Serializer[1] do PHP, para criar ficheiros XML automáticamente.

PHP

```
<?php


require_once 'XML/Serializer.php';
require_once 'XML/Util.php';
```

De seguida é criada a classe RestRequest onde é guarda a informação de um pedido:

PHP

```
class RestRequest
```

[1] `http://pear.php.net/package/XML_Serializer/`

```
{
        private $request_vars;
        private $data;
        private $http_accept;
        private $method;

        public function __construct()
        {
                $this->request_vars  = array();
                $this->data    = '';
                $this->http_accept  = (strpos($_SERVER['HTTP_ACCEPT'], 'json'))
                        ? 'json' : 'xml';
                $this->method   = 'get';
        }

        public function setData($data)
        {
                $this->data = $data;
        }

        public function setMethod($method)
        {
                $this->method = $method;
        }

        public function setRequestVars($request_vars)
        {
                $this->request_vars = $request_vars;
        }

        public function getData()
        {
                return $this->data;
```

```
        }

        public function getMethod()
        {
                return $this->method;
        }

        public function getHttpAccept()
        {
                return $this->http_accept;
        }

        public function getRequestVars()
        {
                return $this->request_vars;
        }
}
```

De seguida é criada a classe RestUtils para processar um pedido:

PHP

```
class RestUtils
{
        public static function processRequest()
        {
          // get our verb
          $request_method = strtolower($_SERVER['REQUEST_METHOD']);
          $return_obj= new RestRequest();
          // we'll store our data here
          $data= array();

          switch ($request_method)
```

```
  {
    // gets are easy...
  case 'get':
    $data = $_GET;
    break;
    // so are posts
  case 'post':
    $data = $_POST;
    break;
    // here's the tricky bit...
  case 'put':
    // basically, we read a string from PHP's special input location,
    // and then parse it out into an array via parse_str... per the PHP docs:
    // Parses str as if it were the query string passed via a URL and sets
    // variables in the current scope.
    parse_str(file_get_contents('php://input'), $put_vars);
    $data = $put_vars;
    break;
  }

// store the method
$return_obj->setMethod($request_method);

// set the raw data, so we can access it if needed (there may be
// other pieces to your requests)
$return_obj->setRequestVars($data);

if(isset($data['data']))
  {
    // translate the JSON to an Object for use however you want
    $return_obj->setData($data['data']);
  }
return $return_obj;
```

```
    }

    public static function sendResponse($status = 200, $body = '', $content_type =
        'text/html')
    {
        $status_header = 'HTTP/1.1 ' . $status . ' ' . RestUtils::
            getStatusCodeMessage($status);
        // set the status
        header($status_header);
        // set the content type
        header('Content-type: ' . $content_type);

        // pages with body are easy
        if($body != '')
        {
            // send the body
            echo $body;
            exit;
        }
        // we need to create the body if none is passed
        else
        {
            // create some body messages
            $message = '';

            // this is purely optional, but makes the pages a little nicer to
                read
            // for your users. Since you won't likely send a lot of different
                status codes,
            // this also shouldn't be too ponderous to maintain
            switch($status)
            {
                case 401:
```

```
                    $message = 'You must be authorized to view
                        this page.';
                    break;
                case 404:
                    $message = 'The requested URL ' .
                        $_SERVER['REQUEST_URI'] . ' was not
                        found.';
                    break;
                case 500:
                    $message = 'The server encountered an error
                        processing your request.';
                    break;
                case 501:
                    $message = 'The requested method is not
                        implemented.';
                    break;
            }

            // servers don't always have a signature turned on (this is an
                apache directive "ServerSignature On")
            $signature = ($_SERVER['SERVER_SIGNATURE'] == '') ?
                $_SERVER['SERVER_SOFTWARE'] . ' Server at ' .
                $_SERVER['SERVER_NAME'] . ' Port ' . $_SERVER['
                SERVER_PORT'] : $_SERVER['SERVER_SIGNATURE'];

            // this should be templatized in a real-world solution
            $body = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD
                HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.
                dtd">
<html>
<head>
<meta http-equiv="
    Content-Type" content="text/html; charset=iso-8859-1">
```

```
                                                            <title>' . $status . ' ' .
    RestUtils::getStatusCodeMessage($status) . '</title>
                                                    </head>
                                                    <body>
                                                            <h1>' . RestUtils::
    getStatusCodeMessage($status) . '</h1>
                                                            <p>' . $message . '</p
    >
                                                            <hr />
                                                            <address>' . $signature
     . '</address>
                                                    </body>
                                            </html>';

                        echo $body;
                        exit;
                }
        }



        public static function getStatusCodeMessage($status)
        {
                // these could be stored in a .ini file and loaded
                // via parse_ini_file()... however, this will suffice
                // for an example
                $codes = Array(
                    100 => 'Continue',
                    101 => 'Switching Protocols',
                    200 => 'OK',
                    201 => 'Created',
                    202 => 'Accepted',
                    203 => 'Non-Authoritative Information',
                    204 => 'No Content',
```

```
205 => 'Reset Content',
206 => 'Partial Content',
300 => 'Multiple Choices',
301 => 'Moved Permanently',
302 => 'Found',
303 => 'See Other',
304 => 'Not Modified',
305 => 'Use Proxy',
306 => '(Unused)',
307 => 'Temporary Redirect',
400 => 'Bad Request',
401 => 'Unauthorized',
402 => 'Payment Required',
403 => 'Forbidden',
404 => 'Not Found',
405 => 'Method Not Allowed',
406 => 'Not Acceptable',
407 => 'Proxy Authentication Required',
408 => 'Request Timeout',
409 => 'Conflict',
410 => 'Gone',
411 => 'Length Required',
412 => 'Precondition Failed',
413 => 'Request Entity Too Large',
414 => 'Request-URI Too Long',
415 => 'Unsupported Media Type',
416 => 'Requested Range Not Satisfiable',
417 => 'Expectation Failed',
500 => 'Internal Server Error',
501 => 'Not Implemented',
502 => 'Bad Gateway',
503 => 'Service Unavailable',
504 => 'Gateway Timeout',
```

```
                505 => 'HTTP_Version_Not_Supported'
            );

            return (isset($codes[$status])) ? $codes[$status] : '';
        }
}
```

De seguida são utilizadas as classes anteriores para implementar o web service:

PHP

```
$data = RestUtils::processRequest();

$filename = '/tmp/messages.txt';

switch($data->getMethod())
  {
  case 'get':
    $messages = explode('#',file_get_contents($filename, true));

    if($data->getHttpAccept() == 'json')
      {
        RestUtils::sendResponse(200, json_encode($messages), 'application/json');
      }
    else if ($data->getHttpAccept() == 'xml')
      {
        // using the XML_SERIALIZER Pear Package
        $options = array
          (
           'indent' => '    ',
           'addDecl' => false,
           'rootName' => 'messages',
           XML_SERIALIZER_OPTION_RETURN_RESULT => true
```

```
            );
        $serializer = new XML_Serializer($options);

        RestUtils::sendResponse(200, $serializer->serialize($messages), 'application/
            xml');
      }

    break;
    // new user create
  case 'post':
    //echo $data->getData();

    file_put_contents($filename, $data->getData().'#', FILE_APPEND);

    // just send the new ID as the body
    // RestUtils::sendResponse(201, '');
    break;
  }

?>
```

3.2 Web Service Client

Para utilizar o web service anterior pode criar o seguinte ficheiro HTML:

HTML

```
<html>
<form name="input" action="index.php" method="post">
Message:
<input type="text" name="data" />
<input type="submit" value="Submit" />
</form>
```

```
<form name="input" action="index.php" method="get">
<input type="submit" value="Get Messages" />
</form>

</html>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/rest_web_service/client.html`

Mais informação disponível em: `http://www.gen-x-design.com/archives/create-a-rest-api-with-php/`.

4

Mashup

Este capítulo tem por objectivo dar uma introdução das técnicas básicas de construção de mashups.

4.1 XPath

Esta secção tem por objectivo mostrar os conceitos básicos para extracção de informação de documentos XML usando queries XPath (XML Path Language) e criação de documentos com recurso à API DOM (Document Object Model).

Iremos apresentar um exemplo para extrair mensagens disponíveis na public timeline do Twitter em formato XML, e identificar, em cada mensagem, vídeos no YouTube que tenham sido publicados pelo mesmo autor.

4.1.1 Resultado em HTML

Começamos por disponibilizar o mashup em HTML através do código que se indica de seguida.

O script começa por recolher a public timeline do Twitter e converter o conteúdo num objecto XML do ambiente PHP

```
<?php
        $url = 'http://twitter.com/statuses/public_timeline.xml';
        $xml = simplexml_load_file($url);
```

De seguida filtra-se essa timeline com recurso a uma query XPATH

PHP

```
$status = $xml->xpath('//statuses//status');
```

Percorre-se cada post encontrado e imprime-se o texto do post e respectivo autor

PHP

```
foreach ($status as $s) {
$username=$s->user->screen_name;

echo '<h3>Twitter Post</h3> <p>"<i>'.$s->text.'</i>"';

echo ' by '.$s->user->name.'</p>';
```

De seguida recolhe-se o YouTube feed do autor do post do Twitter (presume-se que os nomes de utilizador coincidem)

PHP

```
$feedURL = 'http://gdata.youtube.com/feeds/base/users/'.$username.'/uploads';

// read feed into SimpleXML object
$sxml = simplexml_load_file($feedURL);
Verifica-se se existe algum vídeo publicado pelo autor

if ($sxml) {
if($sxml->entry){
echo '<p>'.$sxml->title.':</p>';
```

```
} else {
echo '<p>No videos found for '.$username.'</p>';
}
```

Finalmente imprime-se a informação dos vídeos

PHP

```
// iterate over entries in feed
foreach ($sxml->entry as $entry) {
echo $entry->content;
}
}
}
?>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/mashup_xpath/simple.php`

4.1.2 Resultado em XML

Iremos agora disponibilizar o mashup em XML através do código que se indica de seguida.

Começa-se por criar um documento DOM, com o root element posts

PHP

```
<?php
$doc = new DOMDocument();
$doc->formatOutput = true;

$posts = $doc->createElement( "posts" );
$doc->appendChild( $posts );
```

De seguida explora-se o public timeline do Twitter como feito anteriormente

PHP

```
$url = 'http://twitter.com/statuses/public_timeline.xml';

$xml = simplexml_load_file($url);
$status = $xml->xpath('//statuses//status');

foreach ($status as $s) {
$username=$s->user->screen_name;
```

Criam-se os elementos text e author por cada post

PHP

```
 $post = $doc->createElement( "post" );
$posts->appendChild( $post );

$text= $doc->createElement( "text" );
$text->appendChild($doc->createTextNode($s->text));
$post->appendChild( $text );

$author = $doc->createElement( "author" );
$author->appendChild($doc->createTextNode( $username ));
$post->appendChild( $author )
```

Identificam-se os vídeos tal como anteriormente e cria-se o elemento videos (o simbolo @ evita warnings quando o url não existe)

PHP

```
$feedURL = 'http://gdata.youtube.com/feeds/base/users/'.$username.'/uploads';

// read feed into SimpleXML object
$sxml = @simplexml_load_file($feedURL);
```

```
if ($sxml) {
if($sxml->entry){
$videos = $doc->createElement( "videos" );
$post->appendChild( $videos );
}
```

Cria-se um elemento vídeo por cada vídeo encontrado e no fim imprime-se o XML relativo ao documento DOM

PHP

```
// iterate over entries in feed
foreach ($sxml->entry as $entry) {
$video= $doc->createElement( "video" );
$video->appendChild($doc->createTextNode($entry->link['href']));
$videos->appendChild( $video );
}
}
}
echo $doc->saveXML();
?>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/mashup_xpath/dom.php`

4.2 cURL

Esta secção tem por objectivo a introdução dos conceitos básicos para a recolha de informação de feeds, e respectivo armazenamento local e publicação na web através da biblioteca cURL (client URL) do PHP.

A ideia será recolher automaticamente fotos do Public Feed do Flickr, e de seguida criar um TinyURL para cada foto, que será depois colocado numa base de dados local

4.2.1 MySQL

Para processar os pedidos à base de dados deverá criar o ficheiro mysql.php com o código que se indica de seguida.

A primeira parte do código deverá conter a definição dos parâmetros necessários para fazer uma ligação à base de dados (deverá alterar de acordo com os dados do seu grupo)

PHP

```
<?php
# Configuration Settings
$dbhost = 'appserver.di.fc.ul.pt';
$dbuser = 'awxxx';
$dbpass = 'awxxx';
$dbname = 'awxxx';
```

De seguida deverá executar a ligação à base de dados:

PHP

```
// connect to database
$conn = mysql_connect($dbhost, $dbuser, $dbpass)
        or die ('Error connecting to mysql');
mysql_select_db($dbname);
```

O código deverá ter uma função para criar a tabela onde serão armazenados os links

PHP

```
function db_init_db () {
$query = "DROP TABLE IF EXISTS flickr_link";
mysql_query($query) or die('Error, insert query failed'.$query);

$query = "CREATE TABLE flickr_link (link VARCHAR(255) PRIMARY KEY);";
```

```
mysql_query($query) or die('Error,_insert_query_failed'.$query);
}
```

O código deverá ter uma função para verificar se um dado link já está armazenado

PHP

```
function db_check_link ($link) {
$query = "SELECT_*_FROM_flickr_link_WHERE_link='$link'";
$result= mysql_query($query) or die('Error,_insert_query_failed'.$query);
return mysql_fetch_row($result);
}
```

O código deverá ter uma função para inserir um dado link

PHP

```
function db_insert_link ($link) {
$query = "INSERT_INTO_flickr_link(link)_VALUES_('$link');";
mysql_query($query) or die('Error,_insert_query_failed'.$query);
}
```

O código deverá conter instruções para criar a tabela, que deverão ser executados uma vez e de seguida postos como comentário

PHP

```
# init database, just to execute once
// db_init_db();
// mysql_close($conn);
```

4.2.2 Oracle

Se quiser usar Oracle indica-se de seguida o código equivalente ao do MySQL mas para Oracle.

PHP

```
<?php
# Configuration Settings
$dbhost = '//luna.di.fc.ul.pt/difcul';
$dbuser = 'awxxx';
$dbpass = 'awxxx';
$dbname = 'awxxx';

// connect to database
$conn = oci_connect($dbuser, $dbpass, $dbhost) or die ($conn.'Error connecting');

function db_init_db () {
  global $conn;
  $query = "DROP TABLE flickr_link";
  $stid = oci_parse($conn, $query);
  oci_execute($stid, OCI_DEFAULT);

  $query = "CREATE TABLE flickr_link (link VARCHAR(255) PRIMARY KEY)";
  $stid = oci_parse($conn, $query);
  oci_execute($stid, OCI_DEFAULT);
}

function db_check_link ($link) {
  global $conn;
  $query = "SELECT * FROM flickr_link WHERE link='$link'";
  $stid = oci_parse($conn, $query);
  $r = oci_execute($stid, OCI_DEFAULT);
  return oci_fetch_row($stid);
}

function db_insert_link ($link) {
  global $conn;
  $query = "INSERT INTO flickr_link(link) VALUES ('$link')";
```

```
  $stid = oci_parse($conn, $query);
  $r = oci_execute($stid, OCI_DEFAULT);
}

# init database, just to execute once
// db_init_db();
// oracle_close($conn);

?>
```

4.2.3 Tiny URL

Para criar um tinyURL a partir de um link deverá criar o ficheiro tinyurl.php com o código que se indica de seguida.

PHP

```
<?php
# Configuration Settings
$tinyurl_url = 'http://tinyurl.com/api-create.php?url=';

function get_tinyurl ($link) {
global $tinyurl_url;

// Get tinyURL for this link
$curl = curl_init($tinyurl_url . $link);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
$tiny = curl_exec($curl);
curl_close($curl);

// Check for error
if(strpos($tiny, "Error") != FALSE){
echo "WARNING: A TinyUrl could not be created\n";
```

```
$tiny = $link;
}

return $tiny;
}
?>
```

O código executa o pedido ao web service para de seguida verificar se existiu um erro.

4.2.4 Flickr

Para recolher as fotos do public feed do Flickr deverá criar o ficheiro flickr.php com o código que se indica de seguida.

O código começa por definir o url do feed e uma função que devolve um array com os títulos, datas e link de cada foto publicada no feed.

PHP

```
<?php
# Configuration Settings
$flickr_feed = "http://api.flickr.com/services/feeds/photos_public.gne?id=&tags=&lang
    =en-us&format=rss_200";

function retrieve_photos () {
global $flickr_feed;

$result = array();
```

De seguida o código recolhe o feed

PHP

```
$curl = curl_init($flickr_feed);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
```

```
$xml = curl_exec($curl);
curl_close($curl);
```

Cria um documento DOM (Document Object Model)

PHP

```
$doc = new DOMDocument();
if($doc->loadXML($xml) === false){
echo "ERROR: Could not parse XML\n";
}
```

De seguida o código verifica o formato do feed por forma a extrair a lista das entradas, ou seja as fotos

PHP

```
$atom = false;

// Check if it is atom format
if($doc->firstChild->nodeName == "feed"){
$atom = true;
$entries = array();
$root = $doc->firstChild;
foreach($root->childNodes as $child){
if($child->nodeName == "entry"){
$entries[] = $child;
}
}
}
// Check if it is rss format
elseif($doc->firstChild->nodeName == "rss"){
$entries = $doc->getElementsByTagName("item");
}
// format unknown
else{
```

```
echo "ERROR: Unknown feed format\n";
exit;
}
```

Para cada entrada (foto) extrai o título, a data e o link

PHP

```
// Go through entries
foreach ($entries as $item) {
$title = $item->getElementsByTagName("title")->item(0)->textContent;
$date = $item->getElementsByTagName("date.Taken")->item(0)->textContent;
$link = $item->getElementsByTagName("link");
```

Recolhe apenas o url da informação contida no link

PHP

```
<?// Get link for atom
if($atom){
$links = $link;
$link = false;
foreach($links as $elem){
if($elem->getAttribute("rel") == "alternate"){
$link = $elem->getAttribute("href");
break;
}
}

if($link === false){
echo "ERROR: Item '$title' missing appropriate link element.\n";
continue;
}
}
else{
$link = $link->item(0)->textContent;
```

```
}
```

Armazena os dados de cada foto no array e devolve esse array

PHP

```
$result[] = array('title' => $title, 'date' => $date, 'link' => $link);
}
return $result;
}
?>
```

4.2.5 Execução

Finalmente deverá criar o ficheiro que interliga as componentes anteriores com o código que se indica de seguida.

Primeiro define-se que o output será texto e não html e inclui-se os ficheiros criados anteriormente

PHP

```
<?php

header('Content-Type: text');

require_once ('flickr.php');
require_once ('tinyurl.php');

// just one of the following lines can be enabled
    require_once ('mysql.php');
// require_once ('oracle.php');
```

Defini-se o número máximo de horas que uma foto poderá ter para ser publicada

PHP

```
// hours ago
$timeout = 1;
```

Recolhe-se as fotos do Flickr

PHP

```
$photos = retrieve_photos();


foreach ($photos as $photo) {
$title = $photo['title'];
$date = $photo['date'];
$link = $photo['link'];
```

E finalmente cria-se o tinyURL e armazena-se o link na base de dados.

PHP

```
 if(db_check_link($link) === false){
        echo "\nPreparing $link\n";
        $tiny = get_tinyurl($link);
        // Add to database
        db_insert_link($tiny);
        echo "Link '$tiny' for photo '$title' on '$date' was inserted in database\n";
    }
?>
```

Poderá ver o resultado em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/mashup_curl/`

4.3 Outras Técnicas

Para além das técnicas aqui apresentadas, existem muitas outras que podem ser exploradas para criar mashups, tais como:

- XQuery: `http://www.w3schools.com/xquery/`

- Enterprise Mashup Markup Language (EMML): `http://en.wikipedia.org/wiki/EMML`

5

AJAX

Este capítulo tem por objectivo introduzir os conceitos básicos de programação de páginas web interactivas com actualização assíncrona do seu conteúdo usando tecnologia AJAX. Os exemplos aqui apresentados foram retirados do tutorial apresentado no capítulo 2 do livro *Ajax Design Patterns*[1].

5.1 Preparação

Comece por criar as pastas: public_html/ajax/display, public_html/ajax/remoting, public_html/ajax/dynamic.

Depois execute os seguintes comandos:

shell

```
mkdir ajax
cd ajax
mkdir display remoting dynamic
```

Coloque o seguinte código HTML nos ficheiros display/index.html, remoting/index.html, e dynamic/index.html

HTML

```
<html>
```

[1] `http://oreilly.com/catalog/9780596101800/`, `http://ajaxify.com/tutorial/`

```
<head>
<title>AjaxPatterns.org − Tutorial</title>
<script type="text/javascript" src="tutorial.js"></script>
</head>

<body>

<h1>AjaxPatterns Tutorial</h1>

<div id="sandbox">
</div>

</body>

</html>
```

5.2 Manipulação do conteúdo usando o DOM

A utilização do DOM permite que o conteúdo de página Web seja definido por javascript.

Entre na pasta ajax/display e coloque no ficheiro tutorial.js o seguinte código javascript:

JavaScript

```
window.onload = function() {
 var greeting = document.createElement("span");
 greeting.style.backgroundColor = "yellow";
 greeting.innerHTML = "Hello World!";
 document.getElementById("sandbox").appendChild(greeting);
 }
```

Use o seu browser para ver o resultado, que deverá ser semelhante ao obtido em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/ajax/display/`

Se vizualizar a *source* irá verificar que o código JavaScript inseriu o seguinte código HTML no conteúdo do documento.

HTML

```
<div id="sandbox">
 <span style="background-color:yellow;">Hello World!</span>
 </div>
```

Para evitar repetir a função document.getElementById() várias vezes:

JavaScript

```
window.onload = function() {
 ...
 greeting.innerHTML = "Hello World!";
 $("sandbox").appendChild(greeting); }
function $(id) { return document.getElementById(id); }
```

Para adicionar um link:

JavaScript

```
window.onload = function() {


...


$("sandbox").appendChild(greeting);
$("sandbox").appendChild(document.createElement("hr"));
var link = document.createElement("a");
link.href = "http://ajaxpatterns.org";
link.appendChild(document.createTextNode("More Ajax Stuff ..."));
$("sandbox").appendChild(link);
 } function $(id) { return document.getElementById(id); }
```

Use de novo o seu browser para ver o resultado

5.3 Comunicação com o servidor

Uma página Web pode fazer um pedido a um web service e apresentar o resultado na própria página.

Entre na pasta ajax/remoting e coloque no ficheiro message.html o seguinte código HTML:

HTML

```
<h2>This is message.html</h2>
 and there's <strong>some markup</strong> here<br/>
 as well as an <a href="http://ajaxpatterns.org">AjaxPatterns link</a>.
```

E coloque no ficheiro tutorial.js o seguinte código:

JavaScript

```
function createXMLHttpRequest() { // The XMLHttpRequest Call creation is browser-
    specific
 try { return new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) {}
 try { return new ActiveXObject("Microsoft.XMLHTTP"); } catch (e) {}
 try { return new XMLHttpRequest(); } catch(e) {}
 alert("XMLHttpRequest not supported");
 return null;
 }

function $(id) { return document.getElementById(id); }

window.onload = function() {
 var xhr = createXMLHttpRequest();
 xhr.onreadystatechange = function() {
 if (xhr.readyState==4) { // Request is finished
```

```
if (xhr.status==200) {
$("sandbox").innerHTML = "Retrieved from server ...<hr/>";
$("sandbox").innerHTML += xhr.responseText;
} else {
alert("Message returned, but with error status.");
}
}
}
xhr.open("GET", "message.html", true);
xhr.send(null); // no headers only body
}
```

Use o seu browser para ver o resultado, que deverá ser semelhante ao obtido em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/ajax/remoting/`

O XMLHttpRequest Call é composto por três partes:

- onreadystatechange: definição da função a executar durante os vários passos da chamada
- open(request method, url, asynchronous?): ligação ao servidor
- send: envio do corpo da mensagem

5.4 Comportamento dinâmico

Entre na pasta ajax/dynamic e coloque no ficheiro tutorial.js o seguinte código:

JavaScript

```
window.onload = function() {
```

```
$("sandbox").innerHTML = "Click␣Here!<br/>";

$("sandbox").onclick = function(ev) {
ev = ev || window.event;
 $("sandbox").innerHTML =
"<p>␣Clicked␣at␣" + new Date() + "␣Event:␣" + ev + ".</p><hr/>";

for (property in ev) {
var message = "Property␣" + property + ":␣" + ev[property] + "<br/>";
$("sandbox").innerHTML += message;
} }; } function $(id) { return document.getElementById(id); }
```

Use o seu browser para ver o resultado, que deverá ser semelhante ao obtido em: `http://appserver.di.fc.ul.pt/manual_soa_ajax/ajax/dynamic/`

Verifique que cada vez que faz um click as propriedades são alteradas sem que haja um refresh total da página.

5.5 AJAXificação de uma Aplicação

Verifique como a interface para a mesma aplicação vai evoluindo nos seguintes exemplos:

Ajaxagram - A aplicação convencional: `http://ajaxify.com/tutorial/ajaxagram/`

Agora sem refresh de toda a página: `http://ajaxify.com/tutorial/ajaxagram/ajaxified/`

Agora muito mais dinâmico: `http://ajaxify.com/tutorial/ajaxagram/ajaxified/richer/`

O ultimo exemplo usa três características essenciais do AJAX:

- Live Search

- Progress Indicator

- One-Second Spotlight

Para mais exemplos consulte `http://www.w3schools.com/ajax/`, ou a biblioteca de ferramentas AJAX da Yahoo YUI[2]

[2] `http://developer.yahoo.com/yui/`