Tema 5

APIs y Servicios web

Texto

APIs y Servicios web

5.1

Introducción

APIs web vs. Servicios web

- Servicio web: un componente remoto al que se puede acceder mediante **protocolos web estándar** y desde **cualquier plataforma** (cualquier lenguaje y S.O.)
  - Normalmente el servicio web ofrece un API
- No todos los APIs son servicios web
  - Por ejemplo, el API de Google Maps que usamos en prácticas no se puede considerar un servicio web, ya que solo se puede usar desde Javascript
    - Sin embargo sí hay una versión de servicios web de Google Maps (http://code.google.com/intl/es-ES/apis/maps/documentation/webservices/)
  - Los proveedores de servicios suelen ofrecer librerías en distintos lenguajes que simplifican el acceso al servicio.
    - Hay implementaciones similares del API de Google Maps para otros lenguajes (ahora mismo solo para Flash)

Tipos de servicios web

- **SOAP** o "clásicos"
  - El énfasis se pone en la interoperabilidad transparente al desarrollador (el desarrollador hace llamadas y recibe la respuesta en su lenguaje de programación sin importar en qué lenguaje está desarrollado el servicio)
  - La interoperabilidad se consigue a partir de una serie de tecnologías y herramientas que hacen "transparente" la comunicación
  - Esta "transparencia" supone una carga computacional considerable
  - No se suelen usar desde Javascript (Firefox era el único que tenía soporte nativo pero se eliminó en la versión 3)
- **REST**:
  - El énfasis se pone en hacer servicios "ligeros"
  - La interoperabilidad se consigue a base de modelar las llamadas al API como peticiones HTTP y recibir la respuesta en un formato estándar (por ejemplo JSON o XML).
  - Por tanto, cualquier lenguaje que permita hacer peticiones HTTP y procesar JSON o XML servirá.

Restricciones de Javascript

- En principio Javascript debería servir para acceder a servicios REST
- El problema es que por la política de seguridad "del **mismo origen**" no podemos hacer peticiones AJAX a *hosts* distintos del que proviene el documento
- Alternativas técnicas con Javascript
  - CORS
  - JSONP
  - IFRAMEs
  - Proxy en nuestro servidor

5.2

Servicios REST

Texto

APIs y Servicios web

Servicios web REST

- Como se ha dicho, un servicio REST es accesible a través de HTTP

- En aplicaciones web clásicas, la URL suele representar la operación a ejecutar, y los parámetros HTTP sus parámetros de entrada
  - `http://www.hotel.com/reservarHab?idHotel=5&noches=2&entrada=10/10/10`

- **REST** (REpresentational State Transfer): es un "estilo/filosofía/arquitectura" para servicios web, no una tecnología
  - En REST se asume el uso de HTTP para hacer las peticiones, al igual que en aplicaciones web clásicas, pero la forma de expresar la URL, los parámetros, etc, cambia.

Principios básicos de REST

- El recurso afectado por una operación se representa con la URL (pero en la URL no aparece la operación)
  - Si los recursos siguen una jerarquía esta se suele reflejar en la URL
    - `http://www.hotel.com/hoteles/5/habitaciones/3` (habitación 3 del hotel 5)
    - `http://localhost:8080/eventos/1` (el evento con id "1")
- La operación se expresa con el método HTTP
  - En REST la diferencia entre GET y POST no es meramente el sitio donde se coloquen los parámetros en la petición, sino que representan operaciones diferentes
  - GET es una "lectura". Es obtener el valor de un recurso
    - GET `http://localhost:8080/eventos/1` devolvería los datos del evento "1"
  - POST es una "inserción". Es crear un nuevo recurso
    - POST `http://localhost:8080/eventos/` insertaría un nuevo evento, que habría que enviar en el cuerpo de la petición

Principios básicos de REST (II)

- Ya tenemos "POST:creación" y "GET:lectura". Para tener una aplicación **CRUD** (Create/Read/Update/Delete) solo nos faltan dos operaciones
  - Aunque pueda parecer limitado, prácticamente TODAS las operaciones en una aplicación web encajan en el esquema CRUD (salvo quizás la de login)
- En HTTP existen dos métodos además de GET/POST que no se solían usar por motivos de seguridad
  - PUT: lo podemos asimilar a "Update"
  - DELETE: evidentemente es el "Delete" que nos falta
    - DELETE `http://localhost:8080/eventos/1` borrar el evento "1"

Principios básicos de REST (III)

- El intercambio de datos con el servidor se debe hacer en formatos estándar
  - XML/JSON
- Las aplicaciones no "guardan estado"
  - Lo que quiere decir por ejemplo, que si una petición requiere autentificación, habrá que enviar las credenciales del usuario con cada petición (no se puede hacer login y a partir de ahí "estar logueado")

Servicios REST “públicos”

- Muchos sitios (Google, Twitter, Flickr, Delicious, Facebook...) ofrecen APIs de tipo REST
  - Aunque casi ninguno sigue al pie de la letra los principios REST
  - En realidad son APIs a los que se pueden hacer peticiones GET, POST, PUT y DELETE (en algunos solo GET/POST) y aceptan/devuelven datos en JSON/XML

- Ejemplo: Twitter (http://dev.twitter.com)
  - Obtener en JSON los últimos 20 tweets enviados a Twitter
    - GET http://api.twitter.com/1/statuses/public_timeline.json
  - Obtener info sobre un usuario en JSON (incluido su último tweet)
    - GET http://api.twitter.com/1/users/show.json?screen_name=***usuario***

Texto

APIs y Servicios web

5.3

Servicios web desde Javascript

Opción 1: CORS

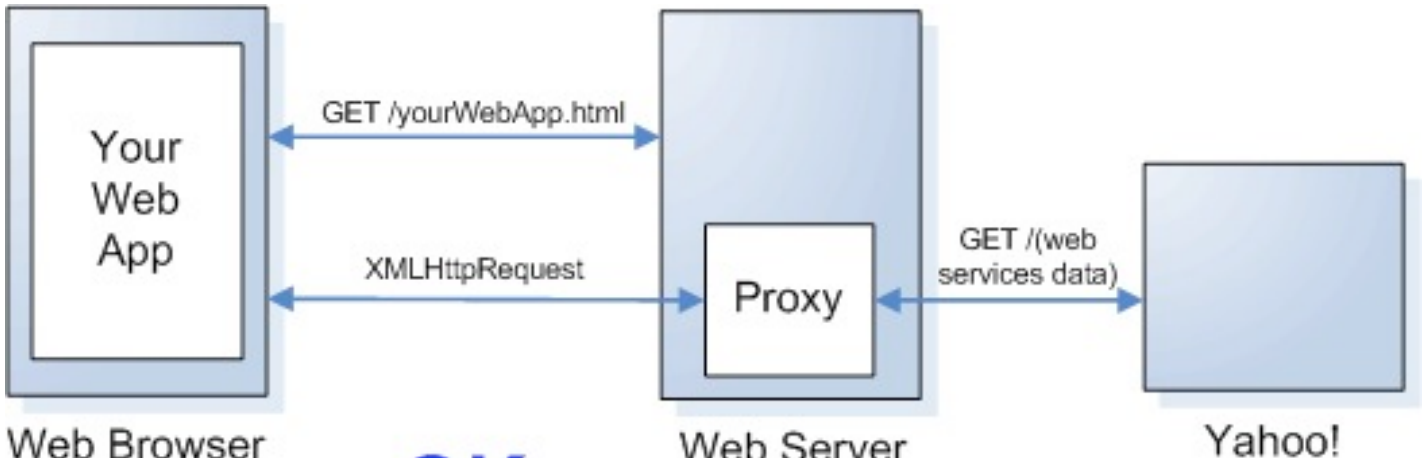
- Es el mecanismo estándar para hacer llamadas AJAX "cross-domain"
- Lo vimos en el tema de AJAX: Si el **servidor** al que le haces la petición la permite (**enviando la cabecera Access-Control-Allow-Origin**), el navegador también dejará que se haga

```
HTTP/1.1 200 OK
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
```

- Por desgracia muy pocos sitios lo ofrecen en la actualidad
  - Ejemplos: bit.yl, twitpic.com
  - Es de esperar que en un futuro su uso se extienda (...*o no*)

Opción 2: Usar un proxy en el servidor

- Podemos colocar en nuestro servidor un programa que simplemente "retransmita" la petición al *host* al que queremos llegar
  - Como las peticiones desde el lado del servidor no están restringidas, no habrá problema
  - Ejemplo: http://developer.yahoo.com/javascript/howto-proxy.html°



Opción 3: Usar el tag <script>

- Las restricciones de seguridad no se aplican a la etiqueta <script>. Con ella podemos cargar (y ejecutar!!) código Javascript de cualquier origen
- Podríamos cargar en un punto del documento la respuesta del servidor en formato JSON, por ejemplo:

```
<script
   src="http://search.twitter.com/search.json?q=javascript">
</script>
```

  - Pero esto no sirve de mucho. Simplemente insertaría en ese punto del documento el objeto en formato JSON, pero no "haría nada" con esa información

JSONP

- Si consiguiéramos ejecutar una función nuestra que recibiera como parámetro el JSON que envía el servidor todo estaría resuelto
- En los servicios que admiten **JSONP**, debemos pasar un parámetro (normalmente se llama "callback") que especifica el nombre de la función a llamar.

```
http://search.twitter.com/search.json?q=javascript
&callback=miFuncion
```

- El servidor nos devolverá un Javascript del estilo:

```
miFuncion(JSON_RESULTADO_DE_LA_PETICION)
```

- Es decir, se ejecutará la función "miFuncion" recibiendo como argumento el JSON. En "miFuncion" procesaríamos los datos (los filtraríamos y mostraríamos en la página, por ejemplo)

Ejecutando JSONP “a petición”

- Según los ejemplos anteriores, parece que la llamada al servicio se tenga que hacer cuando se carga la página, pero también se puede hacer en respuesta a un evento
- Una etiqueta <script> creada dinámicamente se ejecuta en el momento en que se inserta en el documento

```
<body>
<script>
  function llamarServicio() {
      var s = document.createElement("script")
      s.src = "http://search.twitter.com/search.json?q=javascript
&callback=miFuncion"
      document.body.appendChild(s);
  }
  function miFuncion(json) { alert(JSON.stringify(json))}
</script>
<input type="button" onclick="llamarServicio()" value="JSONP">
</body>
```

JSONP en jQuery

- JSONP está integrado en jQuery, si a la URL de la petición "ajax" le añadimos un parámetro "callback=?" usará automáticamente JSONP
  - Recordemos que JSONP **no es AJAX**, pero así se uniformiza la forma de trabajar

```
$.getJSON(
   'http://search.twitter.com/search.json?callback=?',
   //parámetros de la llamada AJAX.
   {q: "javascript"},
    //callback AJAX
    twitter_callback
)
```