AJAX / Escalabilidad de aplicaciones web

Jesús Arias Fisteus

Computación Web (2024/25)

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Parte I

Peticiones HTTP asíncronas en JavaScript

AJAX

El término **AJAX**[1] se refiere a la capacidad de un *script* que se ejecuta en el lado del cliente en el contexto de una página Web para **cargar datos** desde el servidor o **enviar datos** al servidor **sin necesidad de recargar la página completa**.

[1] Originalmente, acrónimo de *Asynchronous JavaScript and XML*.

Peticiones síncronas y asíncronas

El intervalo de tiempo desde que el *script* envía una petición HTTP hasta que recibe su respuesta es impredecible (p.e., podría haber congestión en la red o el servidor, etc.).

Una **petición síncrona** bloquearía el hilo de ejecución hasta la llegada de la respuesta, impidiendo mientras tanto la actualización de la página o la ejecución de sus *scripts*.

Una **petición asíncrona** no bloquea el hilo de ejecución, y la página continúa reaccionando normalmente mientras no llega la respuesta.

Peticiones asíncronas

Es **necesario**, por tanto, que los *scripts* **envíen sus peticiones asíncronamente**.

Con peticiones asíncronas, se procesará la respuesta cuando esta llegue, mediante el modelo de **programación asíncrona** de JavaScript (basado en el objeto **`Promise`**) o mediante **funciones de callback**.

Las APIs fetch y XmlHttpRequest

Los programas JavaScript pueden enviar peticiones asíncronas mediante dos APIs alternativas: **fetch** y **XmlHttpRequest**.

El API **`fetch`** proporciona una solución más moderna y sencilla, y se basa en el objeto **`Promise`**.

El API **`XmlHttpRequest`** es más antigua, compleja y se basa en funciones de *callback*.

JSON

JSON (JavaScript Object Notation) es actualmente el formato más habitual para representar los datos intercambiados entre cliente y servidor.

```
{
    "name": "mary",
    "email": "mary@example.com",
    "roles": ["player", "moderator"],
    "level": 12,
    "location": {
        "city": "Leganés",
        "region": "Comunidad de Madrid",
        "country": "Spain"
    }
}
```

Ejemplo con fetch (GET)

```
1 fetch("/last_msgs")
2     .then((response) => response.json())
3     .then((json) => console.log(json)
4     .catch((err) => console.error("error:", err));
```

Ejemplo con fetch (POST)

```
1 fetch("/post_msg", {
2     method: "POST",
3     headers: {
4         "Content-Type": "application/json",
5     },
6     body: JSON.stringify(message)
7 }).then((response) => response.json())
8     .then((json) => console.log(json)
9     .catch((err) => console.error("error:", err));
```

Ejemplo con JQuery (GET)

```
1 $.getJSON("/last_msgs").done(function(messages) {
2     console.log(messages);
3 }).fail(function(jqxhr, textStatus, error) {
4     var err = textStatus + ", " + error;
5     console.log("Request Failed: " + err);
6 });
```

Ejemplo con JQuery (POST)

```
1  const message = new Message(...);
2  $.ajax({
3      type: "post",
4      url: "/post_msg",
5      data: JSON.stringify(message),
6      contentType: "application/json",
7      success: function (confirmation) {
8          console.log(confirmation);
9      },
10     error(function(jqxhr, textStatus, error) {
11         var err = textStatus + ", " + error;
12         console.log("Request Failed: " + err);
13     }
14 );
```

Política *same origin*

Por motivos de seguridad, **los navegadores aplican restricciones a las peticiones** enviadas desde el origen de la página a recursos en otros orígenes (otros dominios).

En general, un *script* en el contexto de una página de un origen dado puede enviar peticiones asíncronas, con algunas restricciones, a otros orígenes, pero **no puede leer sus respuestas**.

Sin esta restricción, sería fácil que un script ejecutándose en una pestaña robase datos privados del usuario de aplicaciones abiertas en otras pestañas.

CORS

Una aplicación podría necesitar, para funcionar correctamente, deshabilitar esta protección, si se considera que esto es seguro, mediante **CORS (Cross-Origin Resource Sharing)**.

Añadir la cabecera siguiente en la respuesta HTTP a una petición desde otro origen, sea cual sea, permite al script leer esta respuesta:

```
Access-Control-Allow-Origin: *
```

CORS

Añadir la cabecera siguiente en la respuesta HTTP permite al script leer esta respuesta siempre y cuando este se esté ejecutando en el contexto de una página con origen en `https://www.example.com`:

```
Access-Control-Allow-Origin: https://www.example.com
```

Referencias

- Fetch API en MDN.
- Cross-Origin Resource Sharing (CORS) en MDN.

Parte II

Escalabilidad en aplicaciones web

Escalabilidad en aplicaciones Web

El término **escalabilidad** se refiere a la capacidad de la infraestructura de servidor de una aplicación web de dar servicio a una carga creciente (tasa de peticiones, número de usuarios, etc.), o capacidad de ser ampliable para ello.

Uso de memoria caché

El uso de **memoria caché** reduce la carga del servidor Web y del gestor de bases de datos.

No es necesario volver a construir páginas HTML que han sido construidas recientemente sobre datos que no han cambiado.

No es necesario volver a solicitar datos que se han consultado recientemente a la base de datos si no han cambiado.

Uso de memoria caché

- Uso de los mecanismos de control de caché de HTTP 1.1 y versiones posteriores.
- Uso de caché de páginas delante del servidor Web (*proxy* inverso), como NGINX[2].
- Uso de sistemas distribuidos de almacenamiento clave-valor en RAM, como Memcached o Redis:
  - Almacenamiento de fragmentos de HTML ya construidos.
  - Almacenamiento de objetos obtenidos de la base de datos.

[2] `https://www.nginx.com/resources/glossary/caching/`

Escalar incrementando recursos

Cuando el uso de memoria caché no es suficiente, se pueden **incrementar los recursos de cómputo**.

Dos formas de escalar incrementando recursos

La **escalabilidad vertical** consiste en mejorar los equipos que ejercen de servidor Web o gestor de base de datos (más y/o mejores CPUs, más RAM, más disco, etc.).

La **escalabilidad horizontal** consiste en replicar equipos (añadir más servidores Web o más servidores de bases de datos), sin necesariamente mejorar la capacidad de cada servidor.

Escalado horizontal del servidor Web

Para escalar horizontalmente el servidor Web, **se despliegan servidores Web en varias máquinas** y se reparten las peticiones entre ellos:

- Balanceo de carga por DNS: un mismo nombre de dominio con varias IPs.
- Balanceo de carga con servidores de *front-end*.

Es necesario **gestionar sesiones de forma distribuida**.

Escalado horizontal de la base de datos

En general, resulta **más complejo replicar el gestor de bases de datos** que los servidores Web.

Cuando el número de lecturas en la base de datos es considerablemente superior al de escrituras es factible replicar el gestor de bases de datos en **un servidor maestro y varios servidores esclavos**.

Los esclavos se utilizarían principalmente para lecturas, y el maestro realizaría las escrituras y propagaría los cambios a los esclavos.

Partición

En ocasiones es posible **dividir tareas y/o datos en particiones** para escalar horizontalmente.

Partición de tareas

Particionar tareas consiste en que cada servidor Web se encargue de un grupo de tareas concreto.

Por ejemplo, una tienda Web podría ejecutar las tareas de gestión de ventas en un servidor (carro de la compra, tramitación de pedidos, etc.) y las tareas de gestión de clientes en otro (creación de cuentas, configuración de cuentas, autenticación, etc.).

Partición de datos

Particionar datos consiste en que cada gestor de bases de datos guarde un subconjunto de los datos, procurando que cada tarea se pueda llevar a cabo, en la medida de lo posible, sobre un único gestor de bases de datos.

Partición de datos

Una posible estrategia consiste en **particionar datos por temática**.

Por ejemplo: clientes en un gestor de bases de datos, pedidos en otro, registros contables en otro, etc.

Partición de datos

También se pueden particionar datos atendiendo a **otros criterios**.

Por ejemplo, se podrían dividir los datos por países o regiones, de forma que cada gestor de bases de datos guardase todos los datos de clientes, pedidos, registros contables, etc. de un país o región concretos.

Uso de bases de datos no relacionales

Es posible también utilizar **bases de datos no relacionales**[3] para escalar el gestor de bases de datos más eficientemente.

Las bases de datos no relacionales relajan las restricciones ACID para distribuir la base de datos de forma más eficiente.

[3] Conocidas habitualmente como bases de datos NoSQL.

Bases de datos no relacionales

Las bases de datos **clave-valor** almacenan cada valor asociado a una clave.

Son más eficientes en el acceso a valores por clave.

Ejemplos: Memcached[4], Redis[5], Amazon DynamoDB[6], Oracle Berkeley DB[7].

[4] https://memcached.org/

[5] https://redis.io/

[6] https://aws.amazon.com/es/dynamodb/

[7] https://www.oracle.com/database/technologies/related/berkeleydb.html

Bases de datos relacionales

Las bases de datos **orientadas a columnas** almacenan de forma contigua las columnas de datos en vez de las filas.

Son más eficientes en cómputos sobre una o varias columnas que las basadas en registros (filas).

Ejemplos: Apache Cassandra[8], Apache HBase[9], Google Bigtable[10].

[8] `https://cassandra.apache.org/`
[9] `https://hbase.apache.org/`
[10] `https://cloud.google.com/bigtable`

Bases de datos no relacionales

Las bases de datos **basadas en documentos** almacenan los datos de forma semiestructurada y jerárquica, sin un esquema prefijado.

Son más eficientes para almacenar datos que no siguen un esquema (tablas/columnas) prefijado.

Ejemplos: MongoDB[11], CouchDB[12], Google Firebase Realtime Database[13].

[11] `https://www.mongodb.org/`
[12] `https://couchdb.apache.org/`
[13] `https://firebase.google.com/docs/database`

Bases de datos no relacionales

Las bases de datos **basadas en grafos** almacenan los datos con estructura de grafo (vértices y aristas).

Son más eficientes para almacenar y procesar datos altamente conectados en forma de grafos como, por ejemplo, datos de la Web semántica (RDF).

Ejemplos: Neo4J [14], Openlink Virtuoso[15].

[14] `https://www.neo4j.org/`

[15] `https://virtuoso.openlinksw.com/`

Referencias

- Martin L. Abbott; Michael T. Fisher. “Scalability Rules: Principles for Scaling Web Sites, 2nd Edition”. Addison-Wesley Professional (2016).
  - Acceso en línea en O’Reilly
- Chander Dhall. “Scalability Patterns: Best Practices for Designing High Volume Websites”. Apress (2018).
  - Capítulos 1, 2 y 3.
  - Acceso en línea en O’Reilly