Tema 7

Optimización de rendimiento

Optimización

- La optimización de software es el proceso de modificación de un software para hacer que algún aspecto del mismo funcione de manera más eficiente y/o utilizar menos recursos (mayor rendimiento).

- En general, un programa puede ser optimizado para que se ejecute más rápidamente, o sea capaz de operar con menos memoria u otros recursos, o consuman menos energía.

- Debemos centrar nuestros esfuerzos en mejorar las partes que más tiempo se utilicen y/o las partes más lentas de nuestro sistema (cuello de botella).

Optimización de aplicaciones web

- Cuando estamos creando aplicaciones web existen varios aspectos que podemos mejorar:
  - La eficiencia y el consumo de recursos del servidor o servidores
  - El tiempo de carga de las páginas para el usuario
  - Mejoras en la persistencia, generalmente en el SGBD y en la forma en que la aplicación interactúa con él.
  - Reducir el tráfico generado
  - Otros

Ejecución en paralelo

- Si tenemos varias tareas que no dependen unas de otras, podemos ejecutarlas en paralelo para que terminen más rápidamente.

- Necesitaremos el modulo de node “async”

```
$ npm install async
```

- Para lanzar varias tareas en paralelo utilizamos el método parallel

```
async.parallel([
    function(){ ... },
    function(){ ... }
], callback);
```

Métodos asíncronos

- Node se ejecuta en un sólo hilo pero posee una API de llamadas asíncronas para operaciones de I/O.

- Para no bloquear nuestra aplicación con este tipo de operaciones(acceso a ficheros,base de datos …) debemos usar siempre su versión asíncrona. Estos métodos poseen un callback que se ejecutará cuando termine la llamada.

```
// Asíncrono
fs.readFile('file.txt', function(err, buffer) {
  var content = buffer.toString();
});

// Síncrono
var content = fs.readFileSync('file.txt').toString();
```

Caching

- Si estamos accediendo continuamente a un contenido que no cambia con mucha frecuencia, crear una cache mejorará el rendimiento de nuestra aplicación.

- Si existe un valor almacenado en la caché, el servidor lo devuelve, en caso contrario se accede a base de datos o donde se encuentre la información.

- Existen varios módulos disponibles en npm para esta finalidad. Uno de los más sencillos es node-cache

```
$ npm install memory-cache
```

```
cache.put('houdini', 'disapear', 100) // Time in ms
console.log('Houdini will now ' + cache.get('houdini'));//Houdini will now disappear

setTimeout(function() {
  console.log('Houdini is ' + cache.get('houdini'));//Houdini is null
}, 200);
```

Compresión gzip

- Comprimir la respuesta que enviamos al cliente, hará que la reciba más rápidamente, siempre que su navegador sea compatible.
- También reduciremos el tráfico generado.

```
var compress = require('compression');

app.use(compress());
```

Renderizar en el navegador

- Se trata de emplear frameworks que trabajan en el navegador del cliente: AngularJS,Ember,Meteor…
- Con estos frameworks MVC/MVVM (del cliente) el servidor sólo debe devolver los datos en formato JSON y es el navegador el encargado de renderizar las vistas.
- Al seguir este paradigma generamos menos tráfico ya que no enviamos todas las etiquetas HTML, sólo los datos. Además el servidor responde más rápidamente por no tener que renderizar.

https://angularjs.org/

Concatenar y compactar (Minify)

- Conseguiremos mejorar aún más el rendimiento si hacemos minify de nuestros scripts javascript y de nuestros estilos CSS.

- Este proceso consiste en quitar todos los caracteres que hacen legible el código(Espacios, saltos de línea y comentarios).Se logra reducir bastante el tamaño de los ficheros y por tanto, se reduce el tráfico y se envía más rápido.

- También podemos juntar todos nuestros scripts en un sólo fichero para reducir el número de peticiones.

- Para que esta tarea sea muy cómoda para el desarrollador se aconseja usar grunt o algún software similar.

Content Delivery Network (CDN)

- Es un conjunto de servidores que contienen copias de una misma serie de contenidos (imágenes, vídeos, documentos, …) y que están ubicados en puntos diversos de una red para poder servir sus contenidos de manera más eficiente.

- En cualquier caso, el objetivo básico que se persigue es siempre el mismo: hacer más eficiente y fiable la distribución de esos contenidos, mediante la eliminación de cuellos de botella y la cercanía al usuario.

- la CDNs suelen ser gestionadas por terceros, lo que permite a las empresas desentenderse de su creación y mantenimiento.

Optimización MongoDB

- Crea indices para las consultas que más se utilizan. Buscar en un índice es mucho más rápido que buscar en una colección. Puedes crear indices simples y compuestos.

```
db.posts.ensureIndex( { timestamps : 1 } )
```

- Con este indice optimizamos la consulta que obtiene los posts ordenados por fecha.

```
db.posts.find().sort( { timestamp : -1 } )
```

Optimización MongoDB

- Utiliza el método limit y las proyecciones para obtener únicamente los datos que se necesitan en cada consulta y así ahorrar recursos.

```
db.posts.find().sort( { timestamp : -1 } ).limit(10)
```

```
db.posts.find( {}, { timestamp : 1 , title : 1 , author : 1 , abstract :
1} ).sort( { timestamp : -1 } )
```

```
Post.find().limit(10).exclude('comments').exec(function(err, posts) {
  //send posts to client
});
```

Más consideraciones

- Cuidado con las librerías que empleamos ya que pueden contener partes bloqueantes o cuellos de botella y afectar negativamente al rendimiento.

- Uso de módulos binarios en lugar de módulos javascript cuando sea posible. Un ejemplo son los módulos de cifrado, usando la versión binaria conseguiremos mayor velocidad en cada respuesta.

- Emplear sólo los frameworks estrictamente necesarios para no sobrecargar el servidor. En ocasiones existen versiones reducidas del framework con los aspectos que necesitamos.

- Es posible optimizar las imágenes para reducir su peso, generaremos menos tráfico, las páginas se cargaran antes y los usuarios consumirán menos datos.