

Tema 6

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);
```

Ejecución en paralelo

```
function runInParallel() {
  async.parallel([
    getUserProfile,
    getRecentActivity,
    getNotifications
  ], function(err, results) {
    //Este callback se ejecutará cuando todo termine
  });
}
```

Métodos asíncronos

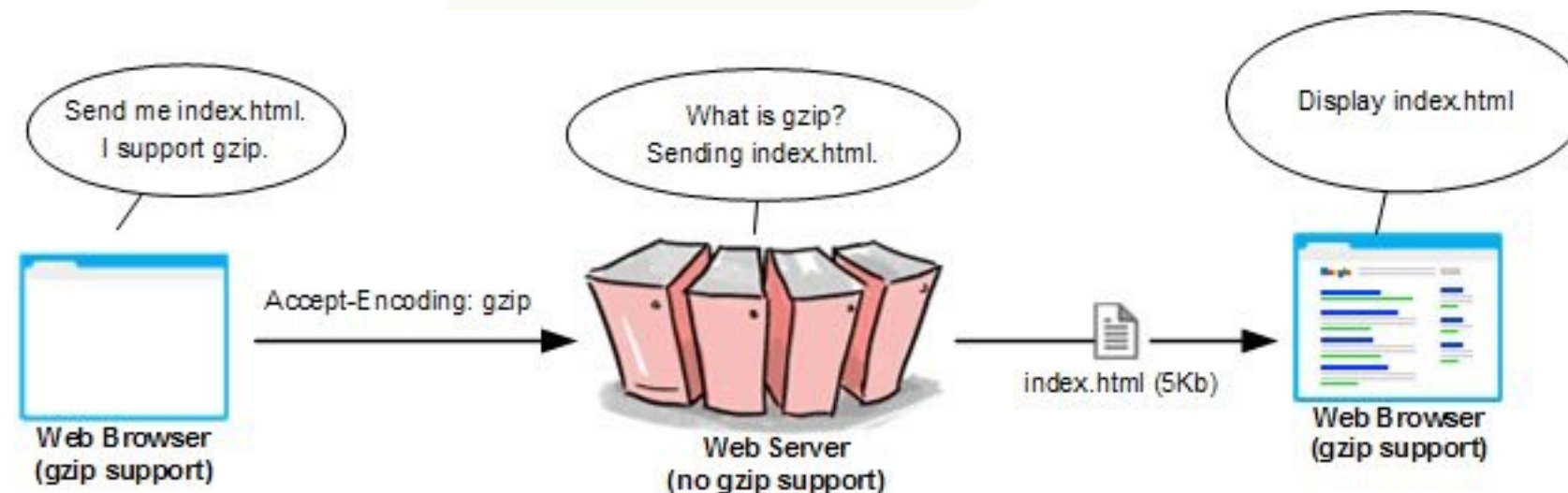
- Node se ejecuta en un sólo hilo pero posee una API de llamadas asíncronas para operaciones de I/O.
- Para no bloquear nuestra aplicación con este tipo de operaciones (acceso a ficheros, base de datos ...) debemos usar siempre su versión asíncrona. Estos métodos poseen un callback que se ejecutará cuando termine la llamada.

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

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

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.



Compresión gzip

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

app.use(compression()); //usar compresión

//En una aplicación en Express podemos crear un
middleware para servir contenido estático:
app.use(express.static(path.join(__dirname, 'public')));
```


Renderizar en el navegador

- Se trata de emplear frameworks que trabajan en el navegador del cliente: **Angular, 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.



Optimización MongoDB

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

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

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

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

Optimización MongoDB

- Y por encima de todo, utiliza el método limit y las proyecciones para obtener únicamente los datos que se necesitan en cada consulta y así ahorrar recursos.
- Si por ejemplo tengo un blog, y quiero mostraren mi primera página los últimos 10... diferentes soluciones combinables:

```
//cargamos todos los posts
db.posts.find();
//cargamos los últimos 10
db.posts.find().sort( { timestamp : -1 } ).limit(10)
//solo los campos que necesitamos (timestamp,title, author, abstract)
db.posts.find( {}, { timestamp : 1 , title : 1 , author : 1 , abstract :
1} ).sort( { timestamp : -1 } ).limit(10)
//y si no necesitamos los documentos anidados, los excluimos.
Posts.find().limit(10).exclude( 'comments' ).exec(function(err, posts) {
    //send posts to client
});
```

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', 'disappear', 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);
```

Caching

- ¿Y si por ejemplo no publicamos artículos muy a menudo? ¿O nuestra lista de artículos más vendidos es bastante “estática”...
- Podemos guardarnos ese contenido en cache con Redis.

```
var redis = require('redis'),
    client = redis.createClient(null, null, { detect_buffers: true }),
    router = express.Router();

router.route('/latestPosts').get(function(req,res){
  client.get('posts', function (err, posts) {
    if (posts) {
      return res.render('posts', { posts: JSON.parse(posts) });
    }

    Post.getLatest(function(err, posts) {
      if (err) {
        throw err;
      }

      client.set('posts', JSON.stringify(posts));
      res.render('posts', { posts: posts });
    });
  });
});
```

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.

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.



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.



¿Qué es Grunt?

- Grunt en una palabra es **automatización**. Grunt permite automatizar tareas repetitivas de forma que el proceso de compilación y testeo sea rápido y automático.
- Es una herramienta muy utilizada en el desarrollo de aplicaciones web, ya que existen muchas tareas predefinidas y repetitivas que un desarrollador debe llevar a cabo para lograr optimizar el software y convertirlo en una pieza de calidad que siga los estándares de la web con el menor esfuerzo posible.

¿Qué es Grunt?

- **Grunt** puede ejecutar casi cualquier tarea que puedas definir de manera programada, incluso gran parte de las tareas comunes conforman ahora la comunidad de plugins de grunt.
- Entre ellos podrás encontrar minificadores y compresores de código, convertidores de un lenguaje a otro (como SASS a CSS), ejecución de pruebas, validadores, rutinas de conexión a servidores, instalación de dependencias especiales, y mucho más.
- Sin duda lo más común (o incluso lo menos esperado) ya está desarrollado por algún miembro de la comunidad; sin embargo si ninguno cumple con una necesidad particular que puedas tener siempre puedes desarrollar tu propia tarea y hasta la puedes publicar en npm para que los demás hagan uso de ella

¿Quién usa Grunt?



- Grunt es usado por importantes empresas que utilizan JavaScript, como pueden ser:
 - **jQuery:** jQuery, jQuery UI, QUnit
 - **Twitter:** Tweetdeck, Typeahead
 - **Mozilla:** Firefox Accounts projects
 - **Adobe:** Brackets, CSS FilterLab
 - **Wordpress:** WordPress Build Process, bbPress, BuddyPress
 - **Microsoft:** Visual Studio 2015 tiene soporte nativo para Grunt

Instalación

- Para instalar Grunt, de forma global en nuestros proyectos, lo haremos con npm como estamos acostumbrados:

```
$ npm install -g grunt-cli
```

- Realmente esto no instala el ejecutor de tareas como tal (ya que éste se instala como un módulo más al incluirlo como una dependencia en el `package.json` del proyecto), sino una interfaz que permite ejecutar la versión correcta de Grunt dependiendo del proyecto en el que esté.

Instalación

- Cada vez que grunt se lanza se comprueba si está instalado en el proyecto usando el sistema de node de `require()` es por esto por lo que se puede ejecutar grunt desde cualquier subcarpeta del proyecto.
- Si se encuentra localmente Grunt, CLI (Command Line Interface) carga la instalación de la librería de Grunt y añade la configuración al Gruntfile, y ejecuta la tarea que se ha intentado ejecutar.

Instalación

- Una vez instalado grunt-cli para integrarlo en nuestro proyecto necesitamos crear dos ficheros:
 - **package.json**: Este es el fichero utilizado habitualmente por pm para guardar información de proyectos con módulos npm. Los plugins de Grunt deberán ir en `devDependencies` como veremos más adelante.
 - **Gruntfile**: Este fichero podrá ser Gruntfile.js o Gruntfile.coffe según el lenguaje en el que se escriba y se usa para configurar o definir tareas y cargar plugins de Grunt

Instalación

- **Package.json:**

- Está en el directorio raíz, en el mismo lugar que Gruntfile, debe ser confirmado si el proyecto utiliza control de versiones y contiene una lista de dependencias que se instalarán con el comando `npm install`
- La mayoría de plantillas automáticamente crean un fichero `package.json`
- El generador automático de Express genera ese mismo fichero también.

Instalación

- **Package.json:**

- Un ejemplo de package.json con grunt sería:

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  }
}
```


Instalación

- La forma más sencilla de añadir plugins de Grunt es a través de npm:

```
$ npm install <module> --save-dev
```

- Este comando no sólo instalará el módulo sino que lo guardará en nuestra lista de dependencias, en el apartado de dependencias para el desarrollo (devDependencies)
- Estos son algunos ejemplos:

```
npm install grunt --save-dev  
npm install grunt-contrib-jshint --save-dev
```

Gruntfile

- El fichero Gruntfile es un fichero JavaScript o CoffeeScript que está contenido en el directorio raíz del proyecto.
- Un fichero Gruntfile está compuesto de estas partes:
 - La función contenedora
 - Proyecto y configuraciones de las tareas
 - Carga de plugins y tareas
 - Tareas por defecto

Gruntfile

- Un ejemplo de Gruntfile sería:

```
//Función contenedora
module.exports = function(grunt) {

    // Configuración del proyecto
    grunt.initConfig({
        pkg: grunt.file.readJSON('package.json'),
        uglify: {
            options: {
                banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
            },
            build: {
                src: 'src/<%= pkg.name %>.js',
                dest: 'build/<%= pkg.name %>.min.js'
            }
        }
    });

    // Carga el plugin "uglify".
    grunt.loadNpmTasks('grunt-contrib-uglify');

    // Tareas predeterminadas.
    grunt.registerTask('default', ['uglify']);

};
```

- Ahora veremos cada una de las partes por separado

Gruntfile

- **La función contenedora:**
 - Es únicamente una función que se exporta como módulo y que debe contener todo lo relacionado con Grunt en su interior.
 - En el caso anterior se correspondería con esto:

```
module.exports = function(grunt) {  
    // Todo lo relacionado con Grunt  
};
```

Gruntfile

- **Proyecto y configuración de tareas:**

- La mayoría de las tareas Grunt utilizan los datos de configuración de definidos como un objeto que se obtiene con el método `grunt.initConfig`.
- En este ejemplo `grunt.file.readJSON('package.json')` importa la información del JSON almacenada en `package.json` al fichero de configuración de Grunt.
- La notación `<% %>` hace referencia a propiedades de la configuración como rutas de directorios o lista de ficheros.
- Se puede almacenar cualquier dato dentro del objeto de configuración siempre y cuando no cause conflictos con los datos de las tareas, en ese caso serán ignorados.

Gruntfile

- **Proyecto y configuración de tareas:**

- Como la mayoría de tareas, la tarea uglify del plugin grunt-contrib-uglify espera que su configuración esté especificada en una propiedad con el mismo nombre.
- Dentro de el se especifica la opción banner y un nombre y destino para build:

```
grunt.initConfig({
  pkg: grunt.file.readJSON('package.json'),
  uglify: {
    options: {
      banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
    },
    build: {
      src: 'src/<%= pkg.name %>.js',
      dest: 'build/<%= pkg.name %>.min.js'
    }
  }
});
```

Gruntfile

- **Carga de plugins:**

- Las tareas más comunes como concatenación o minificación están disponibles como plugins de grunt.
- Al igual que un plugin está especificado en `package.json` como una dependencia y se instala con `pm install`, debe estar habilitado en el Gruntfile con este comando:

```
grunt.loadNpmTasks( 'grunt-contrib-uglify' );
```

Gruntfile

- **Tareas por defecto:**

- Se pueden configurar una o varias tareas que se ejecutarán por defecto cuando se ejecute el comando grunt sin especificar una tarea concreta:
- Si se ejecuta `grunt <plugin>` únicamente se ejecutará el plugin indicador, si se ejecuta `grunt` o `grunt default` se ejecutará uglify en este caso:

```
grunt.registerTask('default', ['uglify']);
```


contrib-clean

- Se encarga de limpiar directorios y archivos.
- Se instala con: `npm install grunt-contrib-clean --save-dev`
- Se carga en Gruntfile con: `grunt.loadNpmTasks('grunt-contrib-clean');`
- Ejemplos:

```
clean: ["path/to/dir/one", "path/to/dir/two"]
clean: {
  build: ["path/to/dir/one", "path/to/dir/two"],
  release: ["path/to/another/dir/one", "path/to/another/dir/two"]
}
clean: {
  build: {
    src: ["path/to/dir/one", "path/to/dir/two"]
  }
}
```

contrib-uglify

- Elimina espacios innecesarios reduciendo el tamaño de los archivos.
- Se instala con: `npm install grunt-contrib-uglify --save-dev`
- Se carga en Gruntfile con: `grunt.loadNpmTasks('grunt-contrib-uglify');`
- Contiene multiples opciones de compresión.
- Ejemplo:

```
grunt.initConfig({
  uglify: {
    my_target: {
      files: {
        'dest/output.min.js': ['src/input1.js', 'src/input2.js']
      }
    }
  }
});
```

contrib-uglify

- También nos sirve para por ejemplo empaquetar las librerías externas que utilicemos, siendo de gran utilidad para obligar a nuestro cliente a cargar muchos menos archivos js en caso de utilizar Angular (por ejemplo)

```
// Combina todos los archivos en src/
grunt.initConfig({
  uglify: {
    all_src : {
      options : {
        sourceMap : true,
        sourceMapName : 'sourceMap.map'
      },
      src : 'src/**/*.js',
      dest : 'composite.all.min.js'
    }
  }
});
```

contrib-copy

- Copia archivos y carpetas
- Se instala con: `npm install grunt-contrib-copy --save-dev`
- Se carga en Gruntfile con: `grunt.loadNpmTasks('grunt-contrib-copy');`
- Ejemplo:

```
copy: {
  main: {
    files: [
      {expand: true, src: ['path/*'], dest: 'dest/', filter: 'isFile'},
      {expand: true, src: ['path/**'], dest: 'dest/'},
      {expand: true, cwd: 'path/', src: ['**'], dest: 'dest/'},
      {expand: true, flatten: true, src: ['path/**'], dest: 'dest/', filter: 'isFile'},
    ],
  },
},
```

contrib-less

- Compila los archivos LESS a CSS
- Se instala con: `npm install grunt-contrib-less --save-dev`
- Se carga en Gruntfile con: `grunt.loadNpmTasks('grunt-contrib-less');`
- Ejemplo:

```
less: {
  development: {
    options: { paths: ["assets/css"] },
    files: { "path/to/result.css": "path/to/source.less" }
  },
  production: {
    options: {
      paths: ["assets/css"],
      plugins: [
        new require('less-plugin-autoprefixer')({browsers: ["last 2 versions"]}),
        new require('less-plugin-clean-css')(cleanCssOptions)
      ],
      modifyVars: { imgPath: '"http://mycdn.com/path/to/images"', bgColor: 'red' }
    },
    files: { "path/to/result.css": "path/to/source.less" }
  }
}
```

contrib-coffe

- Compila los archivos CoffeeScript a JavaScript
- Se instala con: `npm install grunt-contrib-coffee --save-dev`
- Se carga en Gruntfile con: `grunt.loadNpmTasks('grunt-contrib-coffee');`
- Ejemplo:

```
coffee: {
  compile: {
    files: {
      'path/to/result.js': 'path/to/source.coffee',
      'path/to/another.js': ['path/to/sources/*.coffee', 'path/to/more/*.coffee']
    }
  },
  ...
}
```