Web Performance Optimization

La velocidad es un elemento diferencial; el rendimiento es una nueva oportunidad.

08/05/2011

Keep It Simple Lab
Javier Casares

javier.casares@kisslab.com
http://webperformanceoptimization.es/
http://keepitsimplelab.com/

CONTENIDO

¿QUÉ ES WEB PERFORMANCE OPTIMIZATION?

La primera vez que oí hablar a alguien de **WEB PERFORMANCE OPTIMIZATION (WPO)** u *Optimización del Rendimiento de Web* fue a Steve Souders, trabajador de Google (y anteriormente en Yahoo!, conocido allí como Yahoo! Superstar) que se centra sobre todo en el rendimiento web y código abierto.

En esos primeros artículos venía a resumir que **CUANTO MÁS RÁPIDO VA UN SITIO, MEJOR**. Aunque existen una serie de puntos de inicio cuando hablamos del WPO:

- **Rapidez por defecto**: muchas aplicaciones que se construyen para CMS, lenguajes de programación, "la nube", bibliotecas de JavaScript, navegadores, servidores… ya están pensadas para ir rápido.
- **Maquetación del navegador**: con el fin de hacer que las páginas web más rápido los desarrolladores necesitan la capacidad de encontrar qué partes son más lentas. Esto requiere revisar el tiempo que tarda en cargar y ejecutarse el JavaScript, los CSS, la maquetación de los elementos, la gestión del DOM…
- **Consolidación**: las herramientas de rendimiento de la web, servicios y similares no han llevado un único camino, sino que cada uno ha puesto sus esfuerzos de forma separada. Eso va a cambiar y pronto veremos herramientas que combinan la depuración de JavaScript, el perfil de JavaScript, DOM, el uso de la red… todo en una sola herramienta. Las métricas de rendimiento se gestionarán desde un único panel en lugar de tener que visitar múltiples servicios separados. La consolidación también va a ocurrir a nivel de empresa, donde las empresas más pequeñas relacionados con el rendimiento son adquiridos por las grandes empresas de consultoría y servicios.
- **TCP y HTTP**: Los protocolos por los que funciona Internet deben ser optimizados, y SPDY es una propuesta. Tenemos que tratar de conseguir más apoyo para el *pipelining*. Cualquier mejora en la red llegará a todos los sitios y usuarios.
- **Estándar**: hay que establecer un estándar sobre las formas de medir, los datos, las pruebas… La Web Timing Spec es un primer ejemplo a tener presente.
- **Organizaciones en la industria**: dentro del mundillo de la WPO veremos nacer y crecer organizaciones profesionales, formación, certificaciones, organismos de normalización… Un ejemplo podría ser que los editores web compartan información acerca de los anuncios de publicidad lentos.
- **Los datos**: hacer seguimiento de los resultados y encontrar nuevas oportunidades de rendimiento requiere un gran análisis de datos. Es probable que comiencen a verse repositorios públicos de datos relacionados con el rendimiento.
- **Verde**: los estudios realizados que cuantifican cómo mejorar el funcionamiento web confirman la reducción del consumo de energía y por ello la contaminación que generan los centros de datos.
- **Rendimiento móvil**: es como un nuevo punto de partida, se necesita recopilar todo tipo de información hasta encontrar los principales problemas, las causas y encontrar soluciones y crear herramientas para así poder ofrecer información sobre todo esto.

- **La velocidad como elemento diferenciador**: muchas de las decisiones que se tomarán sobre Internet se basarán en el rendimiento. Cuando alguien adquiera un dispositivo, elija un proveedor, se revise un sitio web, la lealtad de los usuarios será un factor importante a la hora de hacer mediciones.

¿QUÉ TIENE EN CUENTA EL WEB PERFORMANCE OPTIMIZATION?

Lo primero y principal: **ES 100% TECNOLOGÍA**. La optimización se basa en las mejoras referentes a la conectividad de redes, la optimización de los servidores web y la mejora de los diferentes elementos que tienen los propios sitios web, desde el HTML hasta el JavaScript, pasando por los CSS o la cantidad de peticiones a servidores DNS.

WEB PERFORMANCE OPTIMIZATION Y LAS MEJORAS DE CONVERSIÓN

Los datos lo dicen todo:

- **Amazon**: 0,1 segundos de retraso implican una pérdida del 1% de los ingresos.
- **AOL**: hizo una revisión del número de páginas vistas en muchos sitios web y concluyó que aquellos que funcionan rápido tienen unas 7-8 páginas vistas por usuario y que las lentas tan sólo 3-4 páginas vistas por usuario.
- **Bing**: 1 segundo de retraso implica una caída del 2,8% de los ingresos; 2 segundos de retraso implican una bajada del 4,3% de los ingresos por usuario.
- **Facebook**: 0,5 segundos más lento provoca una caída de tráfico del 3%; 1 segundo provoca una caída del 6%.
- **Google**: 0,4 segundos de retraso causan una caída del 0,59% de las búsquedas por usuario; 0,5 segundos más en cargar implica un 25% menos de búsquedas.
- **Google Maps**: redujo un 30% el tamaño de sus ficheros y el número de peticiones aumentó un 30%.
- **Hotmail**: 6 segundos de retraso en la carga implica 40 millones de anuncios menos al mes, lo que supone 6 millones de dólares menos al año.
- **Mozilla**: hizo su página de descargas 2,2 segundos más rápida y hubo un crecimiento de descargas de un 15,4%.
- **Netflix**: activó el sistema gzip en sus servidores consiguiendo un aumento de entre el 13% y 25% de velocidad de carga y reducción de un 50% del volumen de tráfico.
- **Shopzilla**: consiguió reducer el tiempo de carga de las páginas de 7 segundos a 2 segundos y la conversión se incrementó entre un 7% y un 12%, además de aumentar un 25% las páginas vistas del sitio y pudiendo reducir la cantidad de servidores a la mitad.
- **Yahoo!**: 0,4 segundos de retraso causan una caída entre el 5% y el 9% del tráfico.

De forma general ya nos encontramos con:

- El 47% de los usuarios esperan que una página cargue en menos de 2 segundos.
- El 14% cambia de tienda online si la página tarda en cargar.
- El 40% de los usuarios abandona una página que tarda más de 3 segundos en cargar.
- El 64% de los compradores que no están satisfechos cambia de sitio para su próxima compra.
- El 52% de los compradores afirman que un sitio que carga rápido los fideliza.

¿CÓMO SE PUEDE ORGANIZAR EL WEB PERFORMANCE OPTIMIZACION?

Aunque las mejoras de WPO se pueden organizar de muchas maneras, podríamos reducirlas a los siguientes grandes grupos:

- Conectividad
- Contenidos
- Cookies
- CSS
- Imágenes
- JavaScript
- Móvil
- Servidor

HERRAMIENTAS QUE AYUDAN AL WEB PERFORMANCE OPTIMIZATION

Los grandes sitios como Yahoo! o Google han lanzado distintas herramientas para revisar la velocidad y rendimiento de los sitios web, ya sean desde sus plataformas web como con aplicaciones externas que funcionan en algunos navegadores.

- Apache JMeter permite hacer pruebas Web, SOAP, bases de datos...
- boomerang es un código JavaScript que te permite medir el rendimiento de las páginas enviando la información al servidor para analizar los resultados.
- Fidder2 es un Web Debugging Proxy que permite monitorizar todo el tráfico contra Internet.
- Google mod_pagespeed: es un módulo para Apache 2 que lleva la configuración por defecto de muchas opciones (compresión, agrupar CSS y JS...).
- Google Page Speed: es una herramienta que se puede instalar en Mozilla Firefox junto al añadido Firebug y también en Google Chrome y permite revisar una serie de elementos de WPO.
- Httperf es una herramienta de Hewlett-Packard que permite hacer peticiones de estrés a los servidores y ver cómo se comportan.

- IBM Page Detailer es una herramienta que permite ver cómo se manda la información de un sitio a los navegadores.
- Loads.In permite analizar el tiempo de carga de una página desde diferentes puntos del mundo y comparar los resultados.
- OctaGate SiteTimer permite monitorizar el tiempo de descarga de cada uno de los elementos de una página.
- Pylot realiza pruebas de carga HTTP de forma que se puede calcular el estrés.
- Site-Perf permite calcular el tiempo de carga de una página de manera realista dando información de todo tipo.
- Web Page Analyzer permite calcular el tamaño de la página, tiempo de descarga y ofrece recomendaciones.
- WebLOAD permite hacer pruebas de estrés a aplicaciones JavaScript de forma que se puede mejorar su rendimiento.
- Yahoo! YSlow: es una herramienta que se puede instalar en Mozilla Firefox junto al añadido Firebug que permite revisar una serie de elementos WPO, también para Google Chrome.

NAVIGATIONTIMING, EL ESTÁNDAR DEL W3C

El W3C (organismo que vela por los estándares de Internet) creó hace un tiempo un grupo de trabajo para crear un estándar en lo que a tiempos de carga se refiere.

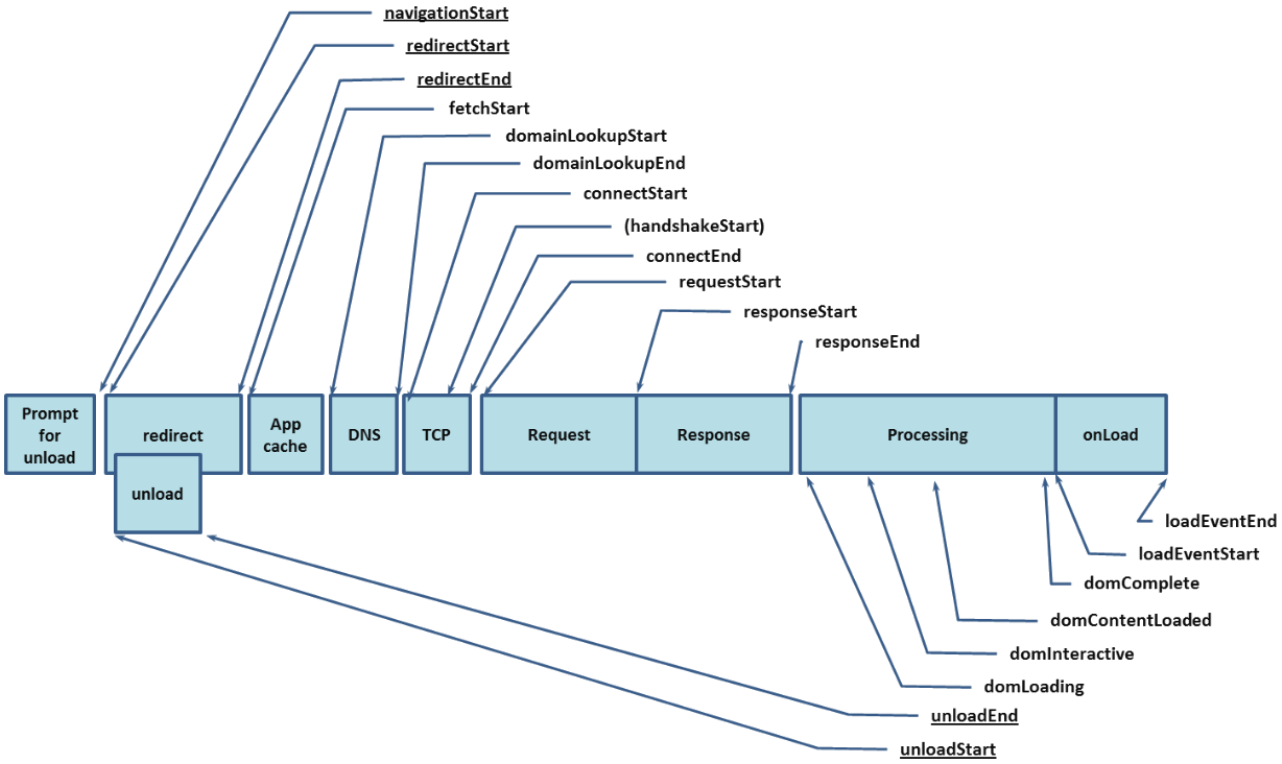
Aunque la propuesta no es definitiva, ya existe una primera versión del estándar que permite ver algunos datos de la propuesta. Ahora mismo funciona en Internet Explorer 9 y en Google Chrome 11. Puedes acceder a esta página para ponerlo a prueba.

Este estándar tiene una serie de atributos con los que podremos tener ciertos datos:

- *navigationStart*:
  El valor corresponde al tiempo inmediatamente después que el navegador hace un "unload" del documento anterior. En caso de que no haya documento anterior, devolverá lo mismo que *fetchStart*. Si existen redirecciones anteriores o no hay origen, el valor será 0.
- *unloadEventStart*:
  Si el documento anterior y el actual tienen el mismo origen se devolverá el valor que haya justo antes de ejecutar el "unload". Si no hay documento previo o es una redirección, será 0.
- *unloadEventEnd*:
  Si el documento anterior y el actual tienen el mismo origen, se devolverá el valor que haya justo después de ejecutar el "unload". Si no hay documento previo, o el "unload" no se lleva a cabo correctamente se devolverá 0.
- *redirectStart*:
  Si hay redirecciones y son del mismo origen, se devolverá el momento de inicio en el que se inicia la redirección. Sino estará a 0.

- *redirectEnd*:
  Si hay redirecciones y son del mismo origen, se devolverá el momento en el que se reciba el último byte de la última redirección. Sino estará a 0.
- *fetchStart*:
  Si el recurso que se va a mostrar se llama mediante una consulta GET, se devolverá el valor anterior de que el navegador compruebe si existe algún tipo de caché del recurso. En cualquier otro caso devolverá el valor en el que el navegador comience a recuperar el recurso.
- *domainLookupStart*:
  Se devolverá el valor inmediatamente anterior a que el navegador haga la petición DNS del documento. En el caso de conexiones persistentes o que esté cacheado, devolverá el mismo valor que *fetchStart*.
- *domainLookupEnd*:
  Se devolverá el valor inmediatamente después a que el navegador haya realizado la petición DNS y haya sido contestada. En el caso de conexiones persistentes o que esté cacheado, devolverá el mismo valor que *fetchStart*.
- *connectStart*:
  Devuelve el momento inmediatamente anterior a que el navegador haga la petición para recuperar el documento. En el caso de que existan cachés o conexiones persistentes se devolverá el mismo valor de *domainLookupEnd*.
- *connectEnd*:
  Devuelve el momento inmediatamente después a que el navegador acabe la petición para recuperar el documento actual. En el caso de que existan cachés o conexiones persistentes se devolverá el mismo valor de *domainLookupEnd*. En el caso de que haya un error en la conexión, se devolverá los valores de la nueva conexión. Este valor también incluirá la conexión en el protocolo SSL.
- *secureConnectionStart*:
  (Opcional). Devuelve el tiempo inmediatamente anterior a que el navegador comience a procesar la petición de protocolo seguro. Si no se utiliza, devolverá 0.
- *requestStart*:
  Devuelve el momento inmediatamente anterior a que se solicite el documento actual. Es aplicado antes de comprobar las cachés. Si hay un error en la conexión, se devuelve el momento en el que se inicia la nueva petición.
- *responseStart*:
  Devuelve el tiempo inmediatamente anterior a que el navegador reciba el primer byte de información desde el servidor o desde la caché.
- *responseEnd*:
  Devuelve el tiempo inmediatamente posterior a que el navegador reciba el último byte del documento o se cierre la conexión.
- *domLoading*:
  El momento en el que el navegador comienza el “loading” de la página.
- *domInteractive*
  El momento en el que el navegador permite la “interactive” de la página.
- *domContentLoadedEventStart*:
  Devuelve el instante en el que el navegador establece el evento *DOMContentLoaded*.

- *domContentLoadedEventEnd*:
  Devuelve el instante en el que el navegador finaliza el evento *DOMContentLoaded*.
- *domComplete*:
  El momento en el que el navegador marca el "complete" de la página.
- *loadEventStart*:
  Devuelve el tiempo en el que el documento se dispara, quedando a 0 mientras esto no ocurra.
- *loadEventEnd*:
  Devuelve el tiempo en el que el documento está completo, quedando a 0 mientras esto no ocurra o sea erróneo.



REDUCIR PETICIONES HTTP

El objetivo es que al realizar menos peticiones HTTP el tiempo de descarga se reduce bastante. Hay que tener presente que en IPv4 se consume en muchos casos más tráfico en la petición de la información que en la transferencia de la información en sí.

COMBINAR VARIOS CSS O JS EN UNO

Categoría: Conectividad, CSS, JavaScript

Normalmente los JavaScript y los CSS se encuentran en puntos de la página como la cabecera *<head>* que bloquean la descarga de otros elementos. Esto significa que cuando la página se empieza a descargar, la descarga en paralelo de estos elementos hace que se puedan bloquear otros.

Para que funcione todo más rápido, es mejor tener un único fichero *.JS* o *.CSS* de un tamaño mayor que no varios pequeños. Uno de los mayores costes de conectividad es la propia conexión y desconexión de las descargas, por lo que si lo reducimos, mejor que mejor.

Una de las formas que hay para reducir estos ficheros y combinarlos en uno es hacer uso de HttpCombiner[1]. La idea que combinar todos los scripts en un bloque, y los CSS en otro. De esta forma tendríamos algo tal que:

```
<script type="text/javascript"
src="HttpCombiner.ashx?s=scripts&t=text/javascript&v=1"></script>
```

Algo parecido ocurriría con las hojas de estilo:

```
<link type="text/css" rel="stylesheet"
href="HttpCombiner.ashx?s=css&t=text/css&v=1"></link>
```

Para ello usaremos un fichero de configuración (*web.config*) que será similar a este:

```
<appSettings>
  <add key="scripts" value="js/jquery.js, js/jDate.js, js/jQuery.Core.js,
js/jQuery.Delegate.js, js/jQuery.Validation.js"/>
  <add key="css" value="css/Theme.css, css/Common.css, css/grid.css"/>
</appSettings>
```

[1] http://webperformanceoptimization.es/httpcombiner/httpcombiner.zip

COMBINAR IMÁGENES O ICONOS EN CSS SPRITES

Categoría: Conectividad, CSS, Imágenes

En muchas ocasiones los sitios web tienen pequeños iconos que se van repitiendo a lo largo de las distintas páginas. Esta cantidad de iconos puede producir el efecto de un exceso de peticiones y, por ello, un número elevado de conexiones inútiles.

Como la tecnología lo permite, una buena solución es la de integrar varias imágenes en una imagen única. Esta imagen, que suele ser un PNG8, contiene una cantidad de imágenes pequeñas suficiente para no tener un tamaño excesivo y con una única imagen poder gestionar todos los elementos desde el CSS.

Este sistema, conocido como CSS Sprites permite que carguemos una única URL (que se usará en distintas zonas de la página) de forma rápida y con una única petición HTTP.

Para gestionar las imágenes, podemos hacer uso de códigos similares a los siguientes:

```
#nav li a {background-image:url('imagen.png')}
#nav li a.item1 {background-position:0px 0px;}
#nav li a:hover.item1 {background-position:0px -72px;}
#nav li a.item2 {background-position:0px -143px;}
#nav li a:hover.item2 {background-position:0px -215px;}
```

INCLUIR IMÁGENES EN HTML O CSS CON EL MÉTODO “DATA:”

Categoría: Conectividad, Contenidos, CSS, Imágenes

Siguiendo el mismo sistema que los CSS Sprites, podemos plantearnos que también son algo inefectivos en aquellos casos en los que estos pequeños iconos se utilicen de forma muy concreta. En esos momentos casi se puede considerar que incluir la imagen directamente en el código fuente del HTML sería más rápido.

El caso del data: URL scheme viene a resolver esta situación, y es que en 1998 ya se planteó este estándar que no cumplen todos los navegadores, por lo que deberemos probar en cada caso.

La idea es incluir en el propio código el elemento en sí, codificado en Base64 que hará que no tengamos que realizar peticiones externas sino que va incorporado en el propio código fuente, ya sea del HTML como del CSS.

El funcionamiento de este elemento sería algo tal que así:

```
data:[<mediatype>][;base64],<data>
```

El primero de los elementos *<mediatype>* corresponde con el MIME Type del contenido a mostrar (por ejemplo un *image/png*). Al final, en codificación Base64 irá el contenido. Quedaría algo parecido a esto::

```
<img src=" iVw0SUhE...bP7WlFSCC" alt="logo">
```

Lo interesante es que se puede utilizar dentro de los CSS, por lo que las imágenes pasarían a formar parte de un CSS de un tamaño mayor y reduciendo la cantidad de peticiones.

USO DE GET EN PETICIONES AJAX

Categoría: Conectividad

El funcionamiento del HTTP es bastante complejo y permite trabajar de muchas maneras en la comunicación entre el cliente-servidor.

El funcionamiento del sistema POST se divide en dos partes: la primera en la que se mandan los encabezados y, una vez se haya llevado a cabo esto, se manda la información. Además, algunos navegadores sólo permiten un tamaño limitado de información que puede ser menor a los 2 Kilobytes.

Por esta razón se recomienda el uso del sistema GET, ya que con una única petición se envía y recibe toda la información. Además, según el RFC2616, el uso de GET está semánticamente diseñado para la recuperación de información, que es lo que habitualmente se suele querer con el uso de AJAX.

USO DE JSON EN PETICIONES AJAX

Categoría: Conectividad

Las peticiones en AJAX pueden enviar y recibir cualquier tipo de información, pero debido a la forma de trabajar de los distintos motores JavaScript por parte de los navegadores, la mejor forma de enviar y recibir información es mediante JSON.

Lo interesante de JSON (*JavaScript Object Notation*) es que es mucho más sencillo a la hora de analizar que XML, y mucho más sencillo de utilizar cuando hay un intercambio de datos fluido entre el cliente y servidor.

PARALELIZAR PETICIONES HTTP

Cuando hablamos de sitios web no podemos olvidar cómo funciona el protocolo que permite el intercambio de información, que principalmente es el HTTP. Los navegadores tienen ciertos límites a la hora de descargar y enviar información, y es que no podemos pretender descargar 100 cosas en paralelo porque simplemente la carga de los sitios se haría interminable.

Es por esto que los navegadores paralelizan entre 2 y 5 conexiones simultáneas para de esa manera poder ir descargando información poco a poco y que se vaya mostrando por pantalla.

El mayor problema de esto es tener muchas peticiones pequeñas o tener peticiones que bloquean otras, ya que no todas se pueden tratar de la misma manera.

MEJOR UNA GRANDE QUE MUCHAS PEQUEÑAS

Categoría: Conectividad

Las peticiones HTTP suelen consumir algunos bytes en cada una de sus peticiones que son "inútiles" en cuanto a que se utilizan para la comunicación de las plataformas pero no para mostrar el resultado final del sitio web. Teniendo en cuenta esto, en la mayoría de casos, se da pie a que es mejor hacer una petición grande que muchas pequeñas para reducir el coste de estas comunicaciones.

Es por esto que es muy recomendable la combinación de CSS y JS o de las imágenes en CSS Sprites.

INCLUIR LOS CSS EN LA PARTE SUPERIOR

Categoría: Conectividad, CSS

Como la mayoría de sitios hoy en día utilizan hojas de estilo, su uso implica que el navegador ha de saber exactamente dónde ha de ir la información antes de mostrarla por pantalla para un uso óptimo. Esto significa que cuanto antes tenga dicha información, antes podrá aparecer.

El hecho de incorporar los CSS en la parte del <head> del HTML implica que se debería cargar esta información antes de mostrar el resto de la página en el navegador, algo que aumentará la velocidad a la hora de renderizar los elementos.

INCLUIR LOS JS EN LA PARTE INFERIOR

Categoría: Conectividad, JavaScript

Los JavaScript tienen una peculiaridad y es que es muy probable que bloqueen la carga del sitio debido a que no suelen permitir más de 2 peticiones en paralelo, dependiendo del navegador usado.

Esto también puede llegar a hacer bloquear todo el sitio si por alguna razón el JavaScript no puede procesarse en un tiempo razonable, y en tal caso puede bloquear imágenes, CSS u otros contenidos.

Por esta razón se recomienda que, en la medida de lo necesario, los ficheros de JavaScript se sitúen al final de la página, para que se puedan cargar todos los otros elementos antes, y al final estos scripts.

USO DEL ATRIBUTO «DEFER» PARA SCRIPTS QUE SIN «DOCUMENT.WRITE»

Categoría: Conectividad, JavaScript

Una de las peculiaridades que tiene la carga de JavaScript desde ficheros externos es que se le puede indicar que sean “aplazados” mediante el atributo defer.

```
<script src=”fichero.js” defer></script>
```

Gracias a esto, aquellos ficheros de funciones o que no necesiten “pintar por pantalla” pueden cargarse posteriormente si el sistema así lo decide y no bloquear el resto de peticiones.

USO DE «NON-BLOCKING SCRIPTS»

Categoría: Conectividad, JavaScript

Gracias a la aparición de nuevas versiones de navegadores y, sobre todo, de nuevos motores de proceso de este lenguaje, el trabajo del AJAX (JavaScript asíncrono) esté permitiendo que los scripts se carguen de una forma que no bloquean el resto de peticiones.

Además, se está planteando añadir el atributo “async” en la carga de los scripts para que no bloquee en absoluto las peticiones del navegador y no tenga porqué cargarse en el orden establecido.

Normalmente, en el caso de usar el “async”, el sistema funcionaría como si cargásemos el script tras un “onload” de la página.

```
<script src=”fichero.js” async></script>
```

Otra de las formas sencillas de cargar archivos de forma que no queden bloqueados es a través del uso de JavaScript y elementos del DOM. Para ello podemos usar un sistema bastante sencillo:

```
var js = document.createElement('script');
js.src = 'fichero.js';
var head = document.getElementsByTagName('head')[0];
head.appendChild(js);
```

En este caso la carga del JavaScript se hará desde la propia cabecera del sitio y no se incluirá en el pie, como se suele recomendar.

Aunque, otra opción más avanzada sería la de descargar el JavaScript, y, en el momento en el que está descargado el fichero, hacer que el navegador lo cargue y en este momento, si se ha de ejecutar se podría hacer sin necesidad de esperar.

```
var js = document.createElement("script");
js.preload = true;
js.src = "fichero.js";                          //aquí se descarga
js.onpreload = function() {
  document.body.appendChild(script);            //aquí se ejecuta
};
```

De la misma forma, aunque no sea con scripts, Firefox tiene limitaciones con los ficheros CSS, de forma que podríamos cargarlos mediante un sistema similar.

```
var h = document.getElementsByTagName('head')[0];
var link = document.createElement('link');
link.href = 'fichero.css';
link.type = 'text/css';
link.rel = 'stylesheet';
h.appendChild(link);
```

A parte de estos métodos, podemos encontrar dos formas más para cargar archivos en JavaScript que tampoco bloquean la carga. La primera de ellas es utilizando la función *setTimeout()*.

```
setTimeout(function() {
  var script = document.createElement("script");
  script.type = "text/javascript";
  script.src = "archivo.js";
```

```
  document.getElementsByTagName("head")[0].appendChild(script);
}, 0);
```

Al cargar el setTimeout() con un tiempo de 0 lo que hacemos es cargarlo en cuando se lea el código de una forma asíncrona.

La otra forma es la carga cuando la ventana se haya acabado de cargar, es decir, cuando se haya acabado de renderizar el código.

```
window.onload = function() {
  var script = document.createElement("script");
  script.type = "text/javascript";
  script.src = "archivo.js";
  document.getElementsByTagName("head")[0].appendChild(script);
};
```

EVITAR ENLACES A CONTENIDOS «404 NOT FOUND»

Categoría: Conectividad, Contenidos

Otro de los grandes problemas que provoca una ralentización a la hora de descargar contenidos de cualquier tipo es hacer peticiones a archivos que no existen, ya sean imágenes, JavaScript, CSS...

Por norma general, si se hace una petición a un elemento inexistente, el navegador parará todas las peticiones hasta que se procese correctamente el error.

EXTERNALIZAR CONTENIDOS CSS Y JS

Los navegadores de hoy en día permiten que el código fuente de una página no tenga que estar completamente incluido sino que se puede externalizar. Es por esto que, al igual que las imágenes, los ficheros JavaScript u Hojas de Estilo se pueden solicitar desde ficheros externos.

FICHEROS .CSS Y .JS CACHEABLES

Categoría: CSS, JavaScript

Una de las ventajas de externalizar los ficheros JS o CSS es que se pueden cachear de forma que, mediante las reglas que he comentado antes, podemos informar a los navegadores si han de descargarse nuevas versiones o no de los ficheros.

Además, hay que recordar que este tipo de archivos son textuales, por lo que se pueden cachear y enviar comprimidos con el sistema Gzip si el servidor web lo permite.

También es interesante recordar que estos ficheros en principio no están programados y no tienen porqué necesitar de cookies, así que se podrían poner en un dominio para estáticos sin ningún problema.

MEJOR USAR EL TAG <LINK> EN VEZ DE @IMPORT PARA LOS CSS

Categoría: CSS

El uso de @import es, como quien dice, arcaico y que hoy en día ya no debería utilizarse. El uso hace un tiempo era básicamente porque como los navegadores más antiguos no eran capaces de leerlo, se podían crear pequeños sistemas alternativos para que unos navegadores leyesen unos y otros, otros. Estamos hablando de navegadores como Netscape 4 o Internet Explorer 3.

Con el paso del tiempo eso ya no es necesario, por lo que este sistema no tiene necesidad de utilizarse.

MINIMIZAR CSS Y JS

Aunque parezca una tontería, en muchas ocasiones por descuido de los programadores o por la simple evolución de un sitio se vuelven a incorporar partes de código repetidas y se genera mucho código sucio con comentarios que no son útiles.

Como lo ideal es enviar un único código comprimido y bien controlado, podemos usar algún sistema que genere y comprima los códigos JavaScript y CSS que podamos usar en cada una de las páginas.

REDUCIR EL TAMAÑO DE CSS Y JS

Categoría: CSS, JavaScript

Hay muchas formas de reducir el código JavaScript y CSS, aunque básicamente la idea es eliminar cualquier comentario, espacio o carácter que no sea útil en ningún sentido, de forma que el tamaño del archivo se reduzca lo máximo posible. En caso de haber varios ficheros, la idea es unirlos en uno único para que no haya un exceso de peticiones contra el servidor.

Para hacer esto podemos usar herramientas como JSMin, JSMin-PHP, YUI Compressor, Packer, JavaScript Compressor, Minify, CSSMin o CSS Compressor. Cada una de estas herramientas tiene sus cosas buenas y malas, por lo que cada uno decidirá la mejor en su caso.

OFUSCAR EL CÓDIGO

Categoría: CSS, JavaScript

Además, existen otras opciones como las de ofuscación del código. Gracias a esto, aparte de reducirlo al máximo, reduce todos los nombres de funciones y similares a otras más sencillas y cortas, por lo que se ahorra código.

Hay que tener en cuenta que la reducción de código es algo relativamente sencillo, y que la ofuscación es más compleja y puede generar más errores. Eso sí, las medias dicen que la reducción reduce cerca de un 21% los datos, y la ofuscación un 25%, pero por esas cifras casi es mejor no arriesgarse y mantener simplemente la reducción de código.

GENERAR UN ARCHIVO ÚNICO Y CACHEARLO

Categoría: CSS, JavaScript

Una vez generado estos ficheros lo mejor es cachearlos y de esta forma mantenerlo creados y con una devolución muy rápida por parte del servidor web.

Con la mayoría de los sistemas de reducción de código ya conseguimos esta unificación de todos los ficheros en uno único. Hay que pensar que puede ser bastante razonable crear un fichero que, en principio, tenga un tamaño elevado pero que sea tratado con fechas y cacheos de forma que no se tenga que descargar en toda la sesión y de cobertura a la práctica totalidad del funcionamiento del sitio web.

CÓDIGOS CORTOS DE COLOR

Categoría: CSS

Un detalle sobre los CSS es que los navegadores permiten una serie de colores que no hacen falta que estén en hexadecimal, sino que se pueden poner en texto plano. Estos 16 colores son los siguientes:

- white: #fff
- yellow: #ff0
- red: #ff0
- fuchsia: #f0f
- silver: #c0c0c0
- gray: #808080
- olive: #808000
- purple: #800080
- maroon: #800000
- aqua: #f00
- lime: #0f0
- teal: #008080
- green: #008000
- blue: #00f
- navy: #000080
- black: #000

Aunque esto puede hacer que sea mucho más sencillo, sí que hay unos de estos colores que, escritos en forma textual son más cortos que su versión hexadecimal, por lo que, de forma óptima, podríamos usar los siguientes colores:

- red
- silver
- gray
- olive
- purple
- maroon
- teal
- green
- navy

REDUCIR EL TAMAÑO DE LAS COOKIES

Las *cookies* (galletas) son unos pequeños fragmentos de texto que se pueden guardar en el navegador del usuario y que permiten a los sitios web añadir y modificar información para ser recuperada durante la sesión o más adelante.

Aunque las cookies se guardan en texto plano y no suelen ocupar mucho tamaño, cada vez que se hace una petición, el navegador envía la información de las cookies que coincida con el dominio solicitante.

ELIMINAR COOKIES QUE NO SEAN NECESARIAS

Categoría: Cookies

Como cada vez que se hace una petición se han de enviar todas las cookies de nuevo, está claro que lo mejor es utilizar cuanta menos información en las cookies sea posible. Es por esto que si las utilizamos, debemos cuidar la cantidad de información que guardamos.

En muchos casos es probable que según vaya pasando el tiempo el usuario interactúe cada vez más con nuestro sitio y de ahí que vayamos guardando más información en las cookies, por lo que es muy razonable tener un sistema que, cada cierto tiempo, vaya eliminando aquella información que no sea absolutamente necesaria para trabajar.

REDUCIR EL TAMAÑO DE LAS COOKIES AL MÍNIMO POSIBLE

Categoría: Cookies

Aunque en muchas ocasiones no podremos eliminar información de las cookies, sí que podemos hacer que estas ocupen menos de lo que lo hacen. Para ello, usaremos las cookies como si fueran un identificador de sesión.

El hecho de que la información se tenga que enviar y recibir cada vez puede generar una ralentización del sitio, por lo que no es nada descartable el poder guardar simplemente un identificador y que la información quede almacenada en el propio servidor web, en la base de datos...

De esta forma conseguiremos que la cookie simplemente sea un número o una pequeña combinación de letras y números que no signifiquen nada (y así también aumentar la privacidad), de forma que sólo se envíen unos pocos bytes en cada petición.

APLICAR LAS COOKIES AL NIVEL DE DOMINIO-SUBDOMINIO NECESARIO

Categoría: Cookies

Por cómo funcionan las cookies hay que tener muy en cuenta cómo se utilizan en cuanto a sus limitaciones. Es por eso que normalmente cuando se utilizan subdominios, el tratamiento de la información es muy distinto.

Si bien es cierto que algunas cookies, como podría ser un identificador, puede ser útil en cualquier subdominio, lo más probable es que cada uno de ellos tenga algunas peculiaridades que hacen que no sea necesario su uso.

Muchos de los navegadores por defecto no limitan el uso de las cookies a los subdominios, por lo que la información, y la velocidad de intercambio de datos, puede ralentizar el sistema si no se trabaja correctamente con ello.

APLICAR UNA FECHA DE ELIMINACIÓN NI MUY LEJANA NI MUY TEMPRANA

Categoría: Cookies

Las cookies tienen un detalle peculiar y es la fecha de su caducidad. Por defecto las cookies son “de sesión”, que significa que cuando el usuario cierra el navegador por completo (o en su defecto, apaga el ordenador) esa cookie desaparece ya que el sistema se encargará de eliminarla. Esto puede ser muy interesante en caso de que cada visita sólo almacene cierta información que en la próxima no sea necesaria.

En caso de indicar fecha para almacenar información durante cierto tiempo, es conveniente limitarla ya que tanto el usuario puede notar un descenso del rendimiento de su máquina como el rendimiento del sitio al tener que hacer uso de cookies que ya son completamente inútiles.

¿Cuál es el tiempo adecuado para una cookie? Eso dependerá de cada proyecto, pero hay cifras mínimas como una semana, medias como 4 semanas, o máximas como 24 semanas. Aun así, esto dependerá de las necesidades del sitio y de mantener esa información en la memoria del navegador.

EVITAR REDIRECCIONES

Las redirecciones son un tipo de código del HTTP que permite pasar de una página a otra manteniendo determinada información.

Cuando en SEO se habla de redirecciones siempre se hace mención a dos de ellas, la 301 y la 302, pero por norma general se les da un uso erróneo. Si leemos el RFC2616 podremos ver el correcto uso de estos códigos:

- Código *301 Moved Permanently* (Mudado permanentemente): significa eso que dice, que la URL anterior ha de dejarse de usar y hay que usar la nueva.
- Código *302 Found* (Encontrado): muchos lo usan como una redirección temporal, pero realmente es el mismo significado que un código 200 pero con la peculiaridad de que se hace una redirección. Este código suele generar contenidos duplicados.
- Código *307 Temporary Redirect* (Redirigido temporalmente): la dirección URL que existe es correcta, pero en estos momentos se redirige a otra que incluye contenido relacionado.

REDIRECCIONES ACOMPAÑADAS DE "EXPIRES" O "CACHE-CONTROL"

Categoría: Conectividad, Contenidos, Servidor

Por norma general cuando se hace una redirección no se le indica la duración de la misma. Esto significa que tanto los robots como los usuarios, cada vez que visiten la página "antigua" han de hacer la petición porque no se le ha indicado el fin de esta.

En principio el código 301 no debería necesitar de este sistema de indicación de caducidad o caché, pero es recomendable indicarlo ya que no deben ser indefinidas, sino que los 301 hay que eliminarlos pasado un tiempo prudencial (entre 6 meses y un año). Una vez pasado este tiempo esa redirección se debería convertir en un código *404 Not Found*.

Un ejemplo en PHP de una redirección correcta podría ser esta:

```
<?php
Header("Location: http://nuevadireccion.com/", true, 301);
Header("Expires: Thu, 01 Dec 2011 12:00:00 GMT");
?>
```

AUTOMATIZAR LA BARRA "/" AL FINAL DE URL

Categoría: Conectividad, Servidor

Gracias a sistemas como el Apache Mod_rewrite tenemos la posibilidad de crear URLs amigables para usuarios y máquinas, pero en muchas ocasiones no se controla correctamente si las URL finalizan con una "/" barra final o no. Hay que tener en cuenta que llevarla o no es totalmente distinta, ya que la URI deja de ser única y genera contenidos duplicados.

Es por esto que existen distintos métodos para que, en muchos casos, se añada o elimine de forma automática dejando al sistema solucionar esta situación. La principal es el uso del Apache Mod_dir con su directiva DirectorySlash, gracias a la cual podremos configurar si queremos la corrección automática o no.

USO DEL META-REFRESH

Categoría: Conectividad

De forma histórica los navegadores son capaces de gestionar una meta-etiqueta que permite redirigir la carga de la página a otro. Para ello se utiliza un código similar al siguiente:

```
<meta http-equiv="refresh" content="3;url=http://www.google.com/">
```

El uso de este sistema, dependiendo del navegador que se utilice, necesita validar de forma condicional e incondicional algunos elementos, por lo que el tiempo de la redirección aumenta.

Además, en el caso de usar proxy, se pueden incluir entradas *Pragma: no-cache* a los encabezados. Aun así, encabezados como el *If-Modifies-Since* pueden mantenerse y seguir devolviendo códigos 304.

COMPRIMIR LOS CONTENIDOS POSIBLES

Una forma de hacer que los sitios web vayan más rápidos es hacer que la velocidad de descarga sea menor. Para eso muchos servidores y navegadores permiten el uso de la compresión Gzip/Deflate que comprime los contenidos antes de enviarlos y los descomprime al ser recibidos.

Aunque esto pueda parecer una contradicción, ya que supone cierto tiempo el comprimir y descomprimir los elementos, si le sumamos los sistemas de caché el tiempo de realizar esta acción es menor que el tiempo que tarda en enviarse la información sin comprimir.

Además, hay que tener en cuenta que la compresión principalmente se ha de aplicar a los contenidos textuales (que son los que tienen más posibilidades de comprimirse) por lo que la carga de las páginas es mucho más rápida.

USAR EL PROTOCOLO «HTTP/1.1»

Categoría: Servidor

En la actualidad existen dos versiones del protocolo HTTP activos, la versión 1.0 y la 1.1. Cada una de ellas tiene sus particularidades, pero a día de hoy lo mejor sería aplicar la versión HTTP/1.1 (definido en el RFC 2616) a todos los servidores web que podamos, ya que permite ciertas funcionalidades que son muy interesantes, como es la del uso de la compresión de los datos.

Una de las cosas que permite precisamente el HTTP/1.1 es la activación de elementos como el "gzip" o el "deflate". De esta forma los servidores web permitirán saber si la relación cliente-servidor

ACTIVAR EL DEFLATE EN APACHE

Categoría: Servidor

El servidor Apache por defecto no incorpora la compresión de los archivos, por lo que es necesario comprobar que esté disponible el Mod_Deflate. Una vez esté comprobado, tan sólo hay que revisar la configuración para que cada uno de los dominios del sistema tenga la compresión activada.

Como antes comentaba, hay determinados archivos que es mejor no comprimir, como son la mayoría de imágenes o ficheros binarios, ya que suelen ir comprimidos en su gran mayoría.

Una configuración básica de este módulo es la siguiente:

```
DeflateCompressionLevel 9
<FilesMatch "\.(js|css|php|html)$">
  SetOutputFilter DEFLATE
  BrowserMatch ^Mozilla/4  gzip-only-text/html
  BrowserMatch ^Mozilla/4\.0[678] no-gzip
  BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
</FilesMatch>
```

De esta forma, automáticamente, todos los ficheros partirán con la compresión activada.

DOMINIOS SIN COOKIES

Las cookies son unos pequeños fragmentos de texto que los servidores envían a los navegadores para que el usuario guarde cierta información que, después, puede ser utilizada en las páginas web. Y quiero remarcar eso de "las páginas web" porque las cookies pueden ser accesibles desde HTML o programación, pero no tiene ningún sentido que sean utilizadas por otros archivos como pueden ser las imágenes.

Cada vez que se hace la solicitud de un archivo (ya sea una página o imagen) se envía toda la información de las cookies, por lo que, en el caso de los estáticos, sale a cuenta reducir esa información de las cabeceras que no se utiliza para nada.

DOMINIOS SIN COOKIES PARA ESTÁTICOS

Categoría: Conectividad, Cookies, CSS, Imágenes, JavaScript, Servidor

Por norma general, y como he comentado en la parte del CDN o del Domain Sharding, es interesante que los contenidos estáticos estén en un dominio distinto al que se usa para la programación o el sitio web navegable por el usuario.

Teniendo en cuenta esto, se plantea como un tema interesante que los contenidos estáticos (o sea, los dominios para estáticos) no tengan cookies, ya que no van a ser utilizadas y vamos a reducir el ancho de banda de cada una de las peticiones.

Para conseguir esto necesitamos un dominio (es mejor no usarlo con un subdominio o similar) y que el servidor web no acepte cookies. Para ello, por ejemplo en Apache podemos usar una codificación tal que:

```
<FilesMatch "\.(ico|gif|jpg|png|flv|pdf|mp3|js|css|xml)$">
  Header set Cache-Control "max-age=2592000"
  Header always unset Set-Cookie
  Header unset ETag
  FileETag None
</FilesMatch>
```

Para eliminarlo, esta vez en Internet Information Server 6 hay que seguir los pasos:

En el sitio web pulsar botón de propiedades y entrar en propiedades.
Entrar en la pestaña HTTP Headers / Cabeceras HTTP
Añadir una entrada nueva: Etag de nombre y vacío el contenido.

USAR ETAGS

Aunque cachear contenidos es un buen sistema para que los navegadores no hagan peticiones de más, otro sistema muy interesante y proactivo es el de indicar la última vez que se modificó el documento. Gracias a esta información tanto el servidor como los navegadores podrán definir si el usuario debe descargar una nueva versión del contenido o no.

Este sistema es totalmente compatible y complementario con las cachés, por lo que su uso, sobre todo en contenidos dinámicos, se vuelve casi imprescindible.

Hay que diferenciar entre el sistema "Last-Modified" y el "ETag". El primero de ellos manda la fecha de última modificación y el segundo manda un identificador único del contenido y la actualización del mismo.

Gracias al uso de este sistema los servidores web y los navegadores se comunicarán de tal manera que el servidor podrá devolver un Código de Estado 304, gracias al cual se avisa al navegador que el contenido no ha cambiado y que, por lo tanto, no hay ninguna necesidad de tener que volverlo a descargar y de esta forma ahorrar tiempo y ancho de banda.

DEVOLVER LA CABECERA ETAG

Categoría: Contenidos, Servidor

La mejor forma de comunicación entre los navegadores y el servidor Apache en el caso de saber si un contenido se ha modificado o no es el uso del ETag. Este sistema se basa en los inode, tamaño y fecha de actualización, de forma que se convierte en una forma más completa que la simple fecha de actualización.

Para activarlo tan sólo habría que configurar algo similar a lo siguiente:

```
FileETag All
```

En el caso de IIS, a partir de la versión 7 el sistema ya viene activado por defecto, aunque sólo funciona con contenidos estáticos. En versiones anteriores es bastante complejo de configurar aunque existe documentación oficial de Microsoft.

CABECERA ETAG EN CDN O DOMAIN SHARDING

Categoría: Contenidos, Servidor

Aunque activar los ETag puede ser muy interesante, si tenemos un sistema distribuido en formato Domain Sharding, por ejemplo y ya no digamos en un CDN, podemos tener ciertos problemas ya que una imagen podría tener distintos ETag dependiendo del servidor (host) al que se esté llamando.

El ETag permite diferentes opciones. Por defecto usamos All para que lo configure automáticamente, aunque permite tres parámetros: INode MTime Size.

En el caso de redes en CDN o con Domain Sharding sería interesante usar la configuración:

```
FileEtag MTime Size
```

CONTROL DE CACHÉ Y ALMACENAJE ESTÁTICO

Sin duda alguna una de las formas de reducir la cantidad de tráfico no-útil en un sitio es utilizando sistemas de caché. Gracias a esto, y de varias formas, podremos reducir la cantidad de peticiones contra el servidor si el usuario utiliza navegadores mínimamente nuevos.

Además, el hecho de separar los sitios/contenidos estáticos de los dinámicos también nos permitirá hacer muchas mejoras en cuanto al uso de tecnologías adecuadas para cada uno de los casos.

DOMAIN SHARDING

Categoría: Conectividad, CSS, Imágenes, JavaScript, Servidor

Por norma general no podemos invertir mucho dinero en proyectos con un CDN (del que luego hablaré) por lo que si queremos jugar con la paralelización de peticiones podemos utilizar el llamado Domain Sharding. Este sistema básicamente lo que permite es montar un sistema de subdominios para realizar varias peticiones aleatorias y en paralelo al mismo servidor estático.

Básicamente se trata de tener un servidor de archivos estáticos (imágenes, JavaScript, hojas de estilo...) y que, de una forma más o menos ordenada y mediante subdominios, podamos hacer muchas peticiones.

La forma normal sería hacer estas peticiones:

- www.dominio.ext/estilos.css
- www.dominio.ext/scripts.js
- www.dominio.ext/imagen1.png
- www.dominio.ext/imagen2.png
- www.dominio.ext/imagen3.png
- www.dominio.ext/imagen4.png
- www.dominio.ext/imagen5.png
- www.dominio.ext/imagen6.png
- www.dominio.ext/imagen7.png

Haciendo esto, encontramos una situación que, haciendo las peticiones en este orden, la imagen1.png no se cargaría hasta que el JavScript se haya acabado de cargar, y como los navegadores suelen venir configurados para hacer 3-4 peticiones en paralelo, la imagen 4 no se empezaría a cargar hasta que no se hubieran cargado los ficheros anteriores.

Si creamos distintos subdominios que apunten al mismo lugar, tendríamos algo así:

- www.dominio.ext/estilos.css
- www.dominio.ext/scripts.js
- ww1.dominio.ext/imagen1.png
- ww1.dominio.ext/imagen2.png
- ww1.dominio.ext/imagen3.png
- ww2.dominio.ext/imagen4.png
- ww2.dominio.ext/imagen5.png
- ww2.dominio.ext/imagen6.png
- ww1.dominio.ext/imagen7.png

De esta forma, se mejorará sensiblemente la velocidad de carga de los distintos ficheros. Hay que tener en cuenta también que no es bueno crear más de 3-4 subdominios distintos, y que cada uno de ellos se debe llamar al menos 3-4 ocasiones para que sea realmente óptima la situación.

NOTA: En estos casos se genera una probabilidad muy elevada de generar contenidos duplicados, ya que los www, ww1 y ww2 acaban llamando a los mismos ficheros, por lo que es recomendable que, en caso de querer dejar indexar estos contenidos, sólo se deje hacerlo con el www pero no con el ww1 o ww2.

DISTRIBUCIÓN GEOGRÁFICA DE CONTENIDOS ESTÁTICOS (CDN)

Categoría: Conectividad, CSS, Imágenes, JavaScript, Servidor

Otra de las formas de mejorar la velocidad de conexión es la suma del Domain Sharding con la geolocalización de contenidos, que es lo que se consigue utilizando un CDN. Básicamente es lo mismo que el Domain Sharding, pero, aparte de tenerlo en una máquina en un país concreto, el sistema se replica automáticamente en centros de datos repartidos por el mundo. De esta forma y automáticamente, tú siempre enlazas a la misma dirección (estatico.dominio.ext) y el sistema hace una redirección al contenido que ha de descargarse.

Este sistema es muy recomendable sobre todo en sitios muy internacionales, ya que el tráfico trasatlántico está muy saturado y provoca lentitud en las comunicaciones, por lo que si tenemos los contenidos en Europa y Estados Unidos haremos que en esas dos zonas la descarga sea más rápida.

El uso de CDN no está recomendado en sitios pequeños ya que puede ser una inversión cara que no tiene mucho sentido, pero sí que lo tiene en sitios con cierto tamaño o que tenga una proyección internacional.

Hay muchos sitios grandes de Internet que utilizan sus propios CDN, como por ejemplo Google y su gstatic.com, Yahoo! con su yimg.com o Youtube con su ytimg.com.

Entre los servicios más habituales tenemos el Amazon CloudFront, Akamai o Windows Azure CDN.

CABECERAS CON CONTROL DE CACHÉ Y EXPIRACIÓN

Categoría: CSS, Imágenes, JavaScript, Servidor

Cachear contenidos web es bastante sencillo siempre y cuando se traten de páginas en HTML, ya que tenemos algunas metaetiquetas que permiten indicar la fecha de caducidad de la página; pero cuando se trata de otros contenidos, la única forma de indicar cuándo caducan dichos contenidos es a través del propio servidor web.

La función básica de la caché es que un usuario no se vuelva a descargar muchas veces un mismo contenido que no cambia. Para esto debemos indicarle al servidor los contenidos (normalmente las extensiones) de los ficheros y el tiempo que queremos que estén cacheados.

Para ello podemos usar dos sistemas: el de avisar la cantidad de tiempo que el archivo ha de mantenerse en caché (normalmente en segundos) o la fecha concreta hasta la que hay que almacenar dicha información.

En el caso de Apache lo habitual es indicar un tiempo desde la fecha de acceso o de modificación, de forma que podremos incluir algo de este estilo en el fichero de configuración general o en el particular del dominio. Existen varias formas de indicarlo, gracias al Mod_Expires.

```
<FilesMatch "\.(ico|gif|jpg|png|flv|pdf|mp3|js|css|xml)$">
  Header set Cache-Control "max-age=2592000"
</FilesMatch>
```

En el caso de querer activarlo en IIS, podemos hacer lo siguiente:

1. *Entrar en las propiedades del dominio.*
2. *Pulsar en la pestaña de Cabeceras HTTP*
3. *Seleccionar la opción de Caducidad de los contenidos.*
4. *Seleccionar el tipo de caducidad (inmediata, después de o hasta) y completar con la información correspondiente.*

Hay que tener en cuenta que si cambiamos frecuentemente los archivos (como puede pasar con los CSS o los JS) es interesante plantearse el versionado de los ficheros, porque una vez un usuario lo ha leído, aunque lo sobrescribamos el sistema no dejará recuperar la nueva versión y es muy probable que el sitio se vea con errores o deformaciones de diseño.

CACHÉ EN CONTENIDOS DINÁMICOS

Categoría: Contenidos, Servidor

Aunque en el caso anterior os comentaba principalmente el cachear los contenidos el máximo tiempo posible, puede darse la necesidad de querer cachear los contenidos dinámicos. Por ejemplo, si tenemos un sitio que sabemos que va a recibir un pico elevado de tráfico, podemos limitar la actualización de las páginas a los usuarios durante unos pocos minutos, a sabiendas que puede darse el caso de que van a recargar las páginas muchas veces. Otro caso es el de un fichero que redirija a otro lugar, caso también que, habitualmente si depende de un parámetro, mandará siempre a ese otro lugar, podemos cachear la información.

El funcionamiento es muy similar al código anterior del Apache, y un ejemplo en PHP podría ser el siguiente:

```
// segundos*minutos*horas*días
$caducidad = 60*60*24*7;
header("Pragma: public");
header("Cache-Control: maxage=".$expires);
header('Expires : '.gmdate('D, d M Y H:i:s', time()+$expires).' GMT');
```

Gracias a este sistema que se puede replicar a otros lenguajes de programación, estamos informando de la cantidad máxima de segundos por los que se ha de cachear, además de la fecha de caducidad del contenido.

USAR “CACHE CONTROL: PUBLIC” PARA CACHEAR CONEXIONES SEGURAS “HTTPS”

Categoría: Contenidos, Servidor

Un tipo concreto de peticiones que por defecto no se cachean nunca son las conexiones seguras (https://...) ya que, como su nombre indica, son seguras y por lo tanto no deben almacenarse en ningún lugar. Pero aunque eso es así, en muchas ocasiones nos encontramos con que una web segura tiene una parte pública que puede cachearse sin problema. Un ejemplo muy claro sería la web de un banco que por defecto sea segura, pero tiene una zona navegable pública (para todos) y otra zona bajo usuario y contraseña. La primera zona, la pública, podría cachearse sin problema ya que no implica ninguna brecha de seguridad para el usuario tenerla en su propia máquina o al servidor ofrecer una copia prealmacenada.

El funcionamiento de este sistema es igual que el de los anteriores, pero hay que indicar de una forma expresa el código “public” para que pueda servirse de esta manera.

```
<FilesMatch "\.(ico|pdf|flv|jpg|jpeg|png|gif|js|css|swf)$">
Header set Cache-Control "public"
Header set Expires "Thu, 15 Apr 2010 20:00:00 GMT"
</FilesMatch>
```

EN APACHE USAR MOD_CACHE, MOD_DISK_CACHE, MOD_MEM_CACHE, MOD_FILE_CACHE Y HTCACHECLEAN

Categoría: Servidor

En el caso de utilizar servidores Apache para servidor contenidos existen distintos módulos muy útiles a la hora de prestar cachés, sobre todo del propio servidor que ayudan a la entrega más rápida de los contenidos.

- Mod_Cache: implementa el RFC 2616 que permite cachear los contenidos en local o a través de un proxy.
- Mod_Disk_Cache: permite que la caché se almacene en el disco local.
- Mod_Mem_Cache: permite que la caché se almacene en la memoria de una mejor forma.
- Mod_File_Cache: mejora del sistema de acceso a la caché de disco.
- Htcacheclean: es un programa que permite optimizar la caché limpiándola y manteniéndola.

PRECARGA DE ELEMENTOS

Una forma de hacer que un sitio web vaya más rápido (de cara al usuario) es precargando elementos, que no deja de ser otra cosa que avanzarse a la navegación del usuario.

Esta técnica puede ser muy útil sobre todo si sabemos que el usuario tiene que navegar de una cierta manera y podemos predecir sus pasos.

ANTICIPAR Y CACHEAR ELEMENTOS

Categoría: Conectividad, Contenidos

En muchas ocasiones podemos saber que si el usuario hace alguna acción en concreto significa que va a navegar hacia otra página.

Un ejemplo sencillo podría ser el que se realiza cuando se pone un elemento en el “carrito de la compra”. En este caso sabemos que, en breve, el usuario pulsará en el carrito por lo que podemos precargar elementos que aparecerán en esa pantalla, como pueden ser imágenes o scripts.

Otro ejemplo, que usan buscadores como Yahoo!, es el del cajetín de búsqueda. Cuando el usuario entra en la página principal puede ir a otros sitios, pero si pulsa en el cajetín significa que va a buscar, y en el tiempo que realiza la consulta y pulsa en “buscar”, se pueden precargar las imágenes y scripts de las páginas de resultados.

REDUCIR LAS PETICIONES DNS

Cada vez que necesitamos saber la dirección IP de un dominio o subdominio tenemos que realizar una consulta a los servidores DNS, lo que significa que el usuario pierde cierto tiempo en estas peticiones.

MINIMIZAR LAS PETICIONES EXTERNAS

Categoría: Conectividad

Si queremos tener menos peticiones, claro está, hay que intentar que en tu sitio web haya la menor cantidad de peticiones a dominios distintos. Esto significa que si podemos hacer peticiones sólo a 1 o 2 dominios conseguiremos una combinación óptima en la resolución de DNS.

Si tenemos en cuenta que tenemos un dominio para los contenidos dinámicos y otro con contenidos estáticos (incluso jugando con Domain Sharding) llegaremos a esa combinación.

Para saber si es rentable incluir otros dominios a la página sólo hemos de ver que al menos se hagan 2 llamadas. Esto hace que códigos como Google Analytics o Google Adsense ralenticen la carga de sitios. Es por esto que, en estos casos (siempre que sea posible) se haga una versión de caché de aquellos elementos que son repetitivos pero sólo se llaman una vez y que se carguen desde nuestro dominio estático.

USO DE SCRIPS ASÍNCRONOS

Categoría: Conectividad, JavaScript

Otra opción para que la carga del sitio se haga más rápida es que, aquellos scripts que se llamen desde dominios distintos se carguen de forma asíncrona. Gracias a esto conseguiremos que el tiempo de respuesta de las DNS no altere la carga de dichos scripts.

Aun así, este sistema podría hacer que, si el usuario cambia o cierra la página muy rápido, estas peticiones nunca se lleven a cabo y los scripts u otros elementos no se lleguen a ejecutar nunca.

REDUCIR EL USO DE CNAME

Categoría: Conectividad

Cuando hacemos una petición a los servidores DNS lo que buscamos en la mayoría de ocasiones es la dirección IP a la que corresponde dicho dominio o subdominio. Esto hace que podamos relacionar ese dominio o subdominio en concreto con su IP y almacenarla en la caché de DNS.

Pero en muchas ocasiones se configuran datos en las DNS como si se tratase de entradas CNAME, que no dejan de ser "alias" de otros resultados de las mismas. Esto significa que si yo hago una petición de un subdominio y me devuelve un CNAME, tendré que realizar otra petición al servidor DNS para resolver la dirección IP de ese alias.

Esto implica que el proceso vaya más lento, por lo que se recomienda reducir el uso de CNAME a aquellas entradas de direcciones IP que no podamos controlar, como podrían ser servicios que nos dan terceros.

CONFIGURACIÓN DE LOS DNS

Categoría: Conectividad

Aunque por norma general las DNS vienen configuradas por defecto, en un proyecto nuevo e importante nunca está de más configurar de forma personalizada los servidores DNS.

- **TTL**: El TTL es el tiempo de vida (caché) de las DNS. Teniendo en cuenta que las DNS no es algo que se actualice con frecuencia, hay que pensar en poner unos valores medianamente razonables y altos. Como mínimo hay que poner 60 minutos y no está de más plantearse incluso indicar 1 día.
- **Dominios "exóticos"**: Hay que tener cuidado con algunos dominios (por ejemplo los .LY) ya que se encuentran "lejos" de los usuarios finales. Estos dominios a nivel global suelen tener un TTL muy elevado (incluso 2 días) y puede haber problemas cuando el servicio falla. Además, la dependencia de los países es demasiado elevada. Los .LY tienen 2 servidores en Libia, 2 servidores en USA y 1 en Holanda, lo que hace tener mucha dependencia.
- **Distribución geográfica**: Una forma de mejorar las peticiones DNS es hacer una especie de CDN de DNS, y distribuirlas geográficamente.
- **DNS Públicas y Privadas**: En las entradas DNS se pueden incluir IPs internas y externas. Esto puede hacer que haya muchas resoluciones incorrectas. Lo mejor es separar las IPs públicas de las corporativas o privadas en distintos dominios.
- **Desactivar la Recursividad**: Por norma general los servidores DNS de un dominio concreto no son servidores DNS públicos, por lo que no es necesario tener activada la recursividad.

DNS PREFETCHING

Categoría: Conectividad

Una de las novedades que van presentando los distintos navegadores es el DNS prefetching. Este sistema básicamente lo que permite es anticipar las peticiones de los dominios resolviendo las DNS antes que se cargue la página o el elemento en cuestión en páginas siguientes o en elementos de la propia página.

En estos momentos los navegadores que lo permiten son Google Chrome 7+, Mozilla Firefox 3.5+, Apple Safari 5+. Internet Explorer no se ha pronunciado sobre ello y Opera parece estar trabajando en ello.

Si queremos hacer un uso manual de este sistema podemos activar la precarga desde el propio encabezado de la página, añadiendo una entrada <link>.

```
<link rel="dns-prefetch" href="http://www.example.com/">
```

Otra opción es la de activarlo o desactivarlo a lo largo de la página. Para ello podemos usar, en el lugar del cuerpo que queramos, los siguientes elementos:

```
<meta http-equiv="x-dns-prefetch-control" content="off">

<meta http-equiv="x-dns-prefetch-control" content="on">
```

OPTIMIZACIÓN DE LAS IMÁGENES

Aunque Internet se basa principalmente en el hipertexto, desde sus inicios que las imágenes han tenido un papel muy relevante frente a otros elementos multimedia.

Aunque por lo general hoy en día hay 3 formatos de imágenes que son soportados por la red (JPEG, GIF y PNG) cada programa y sistema genera las imágenes a su manera, algunas de ellas, aunque la imagen se vea perfectamente, no es la óptima.

CANTIDAD DE COLORES

Categoría: Imágenes

Cada formato de imagen tiene su forma de guardar la paleta de colores. Algunas guardan todos los colores y otras sólo los colores en uso.

Por lo general, el formato que suele fallar más a la hora de reducir la cantidad de colores de la paleta, y por tanto ocupa más el archivo de la imagen, es el GIF. Para reducir la paleta podemos usar algún software como ImageMagick o Smush.it.

REDUCIR EL PESO DE LAS IMÁGENES

Categoría: Imágenes

Aunque uno de los objetivos a la hora de gestionar imágenes sea que se vean de la mejor forma posible, está claro que el tamaño es el principal reto. Es por esto que hay ciertas formas de mejorar el tamaño de las imágenes sin perder calidad en las mismas.

- Intentar convertir los GIF en PNG si reducen su tamaño; por lo general hacer esto suele reducir el tamaño de los archivos cerca de un 20%, por ejemplo con ImageMagick.
- Optimizar los GIF animados; una de las herramientas que permite hacerlo es Gifsicle.
- Optimizar los PNG; puede llegarse a reducir sobre un 15% con herramientas como Pngcrush.
- Optimizar los JPG; puede reducirse el tamaño cerca de un 10% con herramientas como jpegtran.

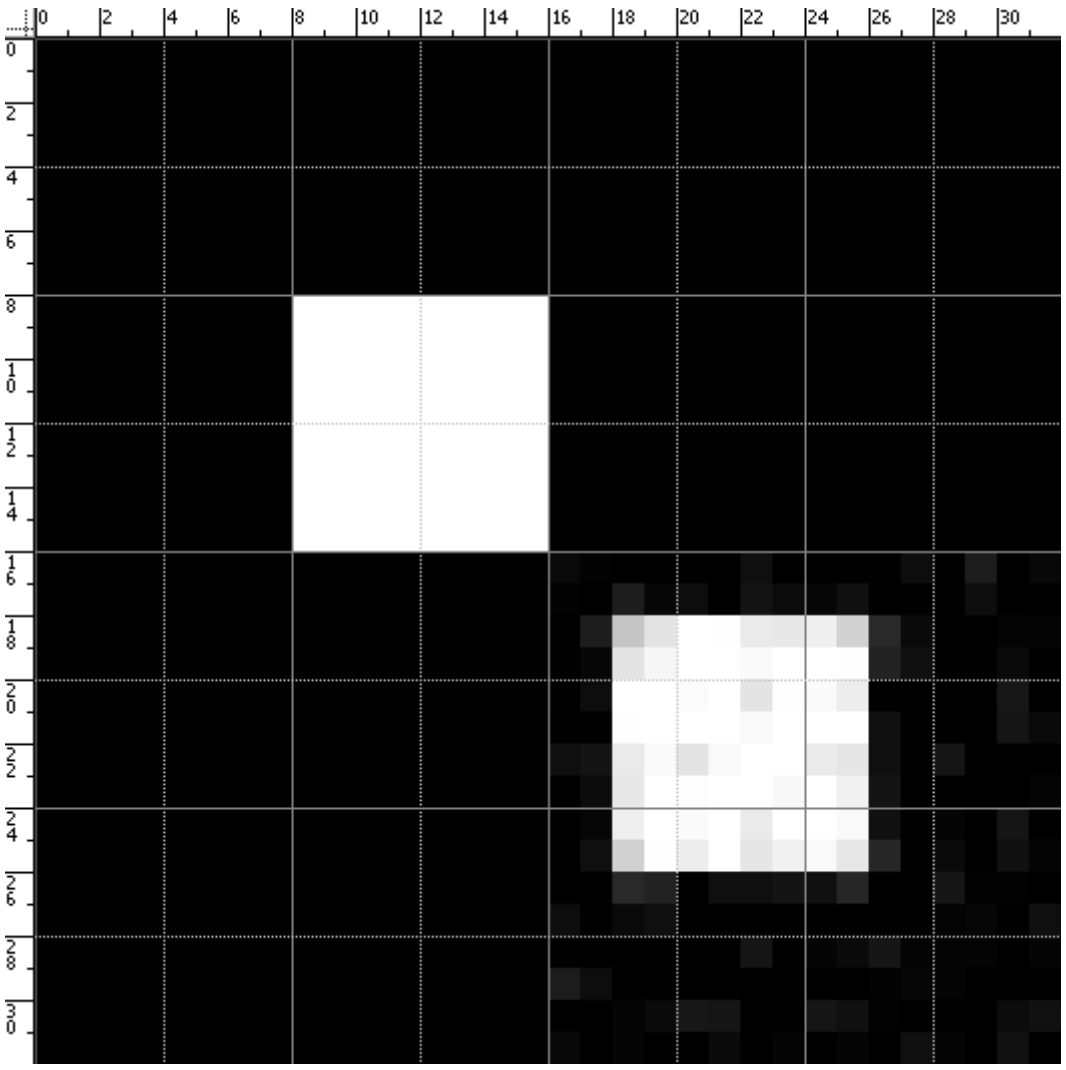
CODIFICACIÓN DE LOS JPEG

Categoría: Imágenes

Los archivos JPEG (*Joint Photographic Experts Group*), que más que un formato de archive de imagen, es un sistema de compresión, permiten varias formas de compresión.

Tras varios análisis sobre la mejor forma de comprimir, los resultados dicen que, si el archivo ocupa menos de 10 kilobytes, la mejor forma de comprimir es usando el algoritmo “baseline” o “estándar”. En cambio, en aquellos archivos que ocupen más de 10 kilobytes, la mejor forma de compresión es la de “progresive”.

Un detalle del sistema de compresión de los archivos JPEG es que se basa en los bloques de 8x8 de forma que se puede comprimir en base a esta regla. En la imagen, el primer recuadro está alineado en el bloque de 8x8 pero el segundo bloque no lo está.



Otro detalle importante de la calidad delos JPEG es que hay ciertos límites. Por ejemplo, comprimir una imagen al 95% o al 100% es inapreciable, pero el tamaño del fichero varía muchísimo. De la misma forma, comprimir a un 50% puede provocar una pérdida excesiva de calidad y, sin embargo, comprimir a un 51% puede llegar a ocupar menos tamaño el fichero y una menor pérdida.

CODIFICACIÓN DE LOS PNG

Categoría: Imágenes

El formato de imágenes PNG (*Portable Network Graphics*) se creó como sustitución del sistema GIF que tiene licencia propietaria y, además, se mejoró ya que se conocían muchos problemas de los GIF y JPEG.

Este tipo de imágenes tiene 5 posibilidades de almacenamiento: *Grayscale* (escala de grises), *Truecolor* (color verdadero), *Indexed-color* (color indexado), *Grayscale with alpha* (escala de grises con canal alfa) y *Truecolor with alpha* (color verdadero con canal alfa). Además, el *Indexed-color* permite almacenar información como *bit transparency* (cada píxel puede ser totalmente transparente o totalmente opaco) o *palette transparency* (cada píxel puede ser semitransparente).

Uno de los mayores problemas es que la mayoría de programas no acaba de permitir elegir entre todos estos formatos y opciones, por lo que siempre será recomendable el uso de herramientas como OptiPNG o pngcrush que convertirán la imagen a la mejor opción posible. En este caso, lo mejor será guardar la imagen en escala de grises si es el caso, o en color verdadero si es en color, ya que estas herramientas automáticamente comprueban si se puede reducir a color indexado.

Otra de las formas de reducir el tamaño de los ficheros en PNG sin perder una cantidad excesiva de calidad es utilizando la *posterización* (por ejemplo, a un 40%). Aunque este sistema implique una pérdida de calidad, puede reducir sobre un 25% el tamaño final del fichero.

ESCALADO DE IMÁGENES

Categoría: Imágenes

En el HTML, cuando se habla del elemento <img> encontramos que los atributos de ancho y alto son obligatorios, ya sea por el propio HTML o mediante CSS. Esto ayuda a la hora de renderizar la página ya que se sabe cuál es el tamaño de la imagen, como interactuará con el resto de elementos. En caso de no indicarlo, el navegador tendrá que esperarse a descargar por completo la imagen, analizarla y sacar el tamaño que ocupa para poder situarla en la página.

Incluso esto, en muchas ocasiones el tamaño real de las imágenes no coincide con el que se ha indicado en la página. Esto implica que, una vez se ha descargado la imagen, el navegador tenga que

dedicar cierto tiempo a recalcular el tamaño de la imagen y generar la imagen que verá el usuario, lo cual implica un tiempo bastante elevado.

En estos casos es siempre mejor disponer de la imagen en los 2 tamaños, el original y uno que se haga generado y cacheado anteriormente que será el que se muestre por pantalla.

OPTIMIZAR LOS CSS SPRITES

Categoría: Imágenes

Cuando generamos ficheros de CSS Sprites hay que tener ciertas consideraciones para saber cuál es la forma óptima de organizar los distintos elementos que incluirá esta "imagen de imágenes".

Para comenzar, los análisis que se han hecho dicen que es mejor disponer las imágenes en horizontal que en vertical, de forma que la imagen resultante siempre debe medir más de largo que de alto.

Otro detalle a tener en cuenta, y principalmente porque las imágenes que componen este fichero suelen ser iconos, hay que intentar combinar los colores para que el resultado quede en GIF 256 o PNG 8, que, en definitiva viene a resumirse en dejar la imagen resultante en 256 colores u 8 bits.

Y, como no, una de las cosas por las que se utiliza este sistema es para reducir el tamaño de las peticiones, por lo que, como llamaremos a los distintos iconos del interior mediante CSS, hay que intentar no dejar espacios entre las imágenes y aprovechar al máximo todos los píxeles.

USAR UN «FAVICON.ICO» PEQUEÑO Y CACHEABLE

Categoría: Imágenes

Los “favicon” son esos pequeños iconos que aparecen al lado de las direcciones URL de los navegadores más modernos y que, en un principio, se usaban como acompañamiento en los listados de favoritos.

Estos iconos, por norma general, son llamados de forma automática por los navegadores, y si no se les indica su existencia en una dirección concreta, se buscan automáticamente como el fichero “favicon.ico” en la carpeta raíz del dominio.

Como este fichero no suele cambiar, se recomienda que exista (o en su defecto tener un fichero que ocupe 0 bytes para evitar generar errores Código 404) y que esté cacheado por mucho tiempo (incluso de forma indefinida).

Además, debido a su uso, es muy recomendable que el tamaño del mismo no sobrepase 1 kilobyte.

ELEMENTOS DE IMÁGENES SIN CONTENIDO

Categoría: Imágenes

En algunos casos los sitios web pueden tener elementos HTML de imagen sin contenido, algo similar a <img src=””> o incluso creado con JavaScript.

Aunque las nuevas versiones de los navegadores ya mejoran el tratamiento de estos casos, que básicamente bloquean la conexión, algunos como Internet Explorer intentan traerse todos los elementos del directorio actual, Safari o Google Chrome intentan cargar la propia página como imagen... esto puede provocar una cantidad de tráfico sin sentido en peticiones HTTP además de provocar problemas en la experiencia del usuario, como la destrucción de información, pudiendo generar problemas con cookies.

HERRAMIENTAS PARA OPTIMIZAR IMÁGENES

Categoría: Imágenes

Herramientas en línea:

- Online Image Optimizer <*http://tools.dynamicdrive.com/imageoptimizer/*>
- PunyPNG <*http://www.punypng.com/*>
- SiteReportCard Image Optimization <*http://sitereportcard.com/imagereducer.php*>
- Smush.it <*http://www.smushit.com/*>

Herramientas de escritorio:

- ImageOptim <*http://imageoptim.pornel.net/*> - Mac OSX
- JPG & PNG Stripper <*http://www.steelbytes.com/?mid=30*> - Windows
- OptiPNG <*http://optipng.sourceforge.net/*>
- PNGGauntlet <*http://benhollis.net/software/pnggauntlet/*> - Windows
- PNGOUT <*http://www.advsys.net/ken/util/pngout.htm*> - Windows
- Radical Image Optimization Tool <*http://luci.criosweb.ro/riot/*> - Windows
- Shrink O'Matic <*http://toki-woki.net/p/Shrink-O-Matic/*> - Adobe AIR
- TweakPNG <*http://entropymine.com/jason/tweakpng/*> - Windows

OPTIMIZACIÓN DE CONTENIDOS MULTIMEDIA

Además de las imágenes, los sitios web pueden incorporar otros elementos multimedia que cada día van creciendo. Estos elementos multimedia suelen ser ficheros externos que hay que cargar, habitualmente con los elementos <object> o <embed> y que pueden suponer una sobrecarga para las páginas.

REDUCCIÓN DE FICHEROS SWF (ADOBE FLASH)

Categoría: Contenidos

Aunque los ficheros de Adobe Flash no son concretamente imágenes, sí que es cierto que como elementos multimedia se deben un respeto, ya que existen en muchos sitios web. Los ficheros SWF están comprimidos de por sí gracias al sistema *deflate* propio de Java, por lo que al final, lo que recibimos ya está reducido, pero se puede reducir aún más gracias a 7-Zip, que comprime mejor. Para ello usaremos apparat. Es recomendable mirar en profundidad su configuración.

```
reducer -i test\test.swf
[i] Apparat -- http://apparat.googlecode.com/
[i] Launching tool: Reducer
[i] Waiting for 7z ...
[i] Compression ratio: 18.224573%
[i] Total bytes: 310
[i] Completed in 547ms.
```

OPTIMIZACIÓN DE FUENTES CSS @FONT-FACE

Categoría: Conectividad, Contenidos

Una de las novedades de CSS3 es que permite a los navegadores cargar un fichero de fuente y así mostrar la página con los tipos de letra que el sitio web decide y no depender de las fuentes del sistema. Estas fuentes se cargan a través de las hojas de estilo y, como los ficheros de fuente no suelen ser muy reducidos en tamaño, pueden perjudicar el rendimiento de carga.

Hay que tener en cuenta que, aunque por defecto el código del @font-face se carga en la cabecera de la página, hasta que esta no acaba de pintarse en el navegador, no se ejecuta el cambio del tipo de letra.

Teniendo en cuenta esto, y que gracias a JavaScript podemos modificar los estilos, podemos mejorar un código tal que así:

```
<style type="text/css" media="screen">
  @font-face {
        font-family: 'WebFont';
        src: url('http://www.example.com/webfont.eot');
        src: local('☺'), url('http://www.example.com/webfont.woff')
format('woff'), url('http://www.example.com/webfont.ttf')
format('truetype'), url('http://www.example.com/webfont.svg#webfont')
format('svg');
        font-weight: normal;
        font-style: normal;
  }
  h1 {font: 60px/68px 'WebFont', Arial, sans-serif; letter-spacing: 0;}
  p {font: 18px/27px 'WebFont', Arial, sans-serif;}
</style>
```

Por un código en dos partes, donde dejaremos en la cabecera algunos estilos:

```
<style type="text/css" media="screen">
  h1 {font: 60px/68px 'WebFont', Arial, sans-serif;letter-spacing: 0;}
  p {font: 18px/27px 'WebFont', Arial, sans-serif;}
</style>
```

Y al final de la página, cargaremos de forma retardada las fuentes, eso sí, cacheadas en un servidor estático para que no se tengan que descargar cada página.

```
<script type="text/javascript">
  var fuente1 = "@font-face {" +
        "       font-family: 'WebFont';" +
        "       src: url('http://s.example.com/webfont.eot');" +
        "       src: local('☺'),
url('http://s.example.com/webfont.woff') format('woff'),
url('http://s.example.com/webfont.ttf') format('truetype'),
url('http://s.example.com/webfont.svg#webfont') format('svg');" +
        "       font-weight: normal; font-style: normal;" +
  "}";
```

```
(function() {
        var estilo1 = document.styleSheets[0];
        if ("function" === typeof(estilo1.insertRule)) {
                estilo1.insertRule(fuente1, 0);
        }       else if ("string" === typeof(estilo1.cssText) ) {
                estilo1.cssText = fuente1;
        }
})();
</script>
```

Además, hay que tener en cuenta que, en algunos casos, Internet Explorer tiene ciertos problemas con la carga de las fuentes, sobre todo si el sitio incluye varios JavaScript, ya que dependiendo si se ponen antes de la hoja de estilos puede bloquear la carga. En el caso óptimo esto no ocurre lo que permite mejorar el rendimiento en cualquier situación.

DEVOLVER CÓDIGO PARCIALMENTE

Sin duda una de las particularidades de Internet es que todos los elementos se pueden dividir en varios y de esta forma poder transmitir múltiples paquetes en momentos distintos.

Gracias a esto podemos también dividir una página en varios fragmentos y así procesarla por partes, haciendo que la carga de otros elementos (como imágenes o scripts) se haga también por bloques.

FUNCIÓN "FLUSH()"

Categoría: Conectividad

La mayoría de lenguajes de programación tienen una función "flush()" que permite limpiar el buffer de salida de datos y enviarlo directamente al usuario en el momento en el que se decida.

Aunque no hay muchos sitios web que lo utilicen, es muy interesante planteárselo en momentos estratégicos como podría tras el </head> pudiendo enviar al usuario todas las cabeceras y comenzar a descargar los scripts o CSS que haya en las cabeceras.

Por norma general, se recomendaría, en una web básica, utilizar este sistema tras el encabezado de la página, la cabecera, el menú, el contenido y el pie, enviando de esta manera los distintos bloques y pudiendo renderizarlos el navegador poco a poco.

PROGRAMACIÓN EN LOS CSS

Aunque no es muy común hacerlo, los ficheros CSS permiten cierto código JavaScript en su interior para hacerlo algo más sencillo a la hora de desarrollar.

De todas formas, esta aplicación no es un estándar en todos los navegadores por lo que puede provocar algunas situaciones no deseadas cuando el sistema que ha de renderizar no sabe qué hacer.

PROBLEMAS CON INTERNET EXPLORER

Categoría: CSS, JavaScript

La mayoría de problemas en la programación dentro de CSS viene por versiones antiguas de Internet Explorer. Si bien es cierto que este navegador introdujo ciertas facilidades a la hora de realizar hojas de estilo semiprogramadas, es muy recomendable no utilizar estas técnicas.

Otra de las que no parece funcionar muy bien, también en este navegador, es el uso de filtros del estilo “AlphaImageLoader”, ya que hacen que el consumo de recursos aumente de forma exagerada.

PROGRAMACIÓN DE LOS JS

Sin duda cada vez más JavaScript se ha convertido en u lenguaje de programación en el cliente que permite mucha interactividad con los servidores y con los elementos DOM del sitio web.

Aunque JavaScript cada vez más está integrado con el lenguaje HTML, bien es cierto que existen muchas técnicas para mejorar la velocidad de procesamiento del código.

OPTIMIZACIÓN GENERAL PARA JAVASCRIPT

Categoría: JavaScript

JavaScript, como cualquier otro lenguaje de programación, tiene sus cosas buenas y malas, pero sobre todo, al ser un lenguaje que se puede interpretar y que se ejecuta en el navegador cliente, vale la pena aplicar una serie de reglas básicas.

- Evitar las variables globales es una idea bastante buena ya que el rendimiento de dichas variables es bastante bajo. En el caso de ejecutar varios códigos es probable que se sobrescriban dichas variables. Para ello podríamos tener elementos de este estilo que se protegen de accesos externos o la posibilidad de ser sobrescritos.

```
miNameSpace = function() {
  var current = null;
  function init() {...}
  function change() {...}
  return {
            init:init,
            set:change
  }
}();
```

- Otro detalle importante es el uso de un código lo más estricto posible, es decir, un código que se ajuste a cualquier motor que interprete JavaScript y sea fácilmente interpretable. Para verificar que estamos usando un código correcto podemos hacer uso de herramientas como JSLint.
- Es mejor no cambiar valores por defecto del DOM en los valores de los elementos (sobre todo en aquellos que hacen referencia a los estilos), sino modificarlos haciendo cambios en clases que se aplican a dichos elementos. Por ejemplo, podemos aplicar unos cambios tales que:

```
inputs.style.borderColor = '#f00';
inputs.style.borderStyle = 'solid';
inputs.style.borderWidth = '1px';
```

O podemos hacer un cambio más sencillo, tal que este:

```
inputs.className += ' error';
```

Y que esta clase "error" sea la que haga el cambio en el estilo, sin modificar directamente los elementos.

- Es mejor utilizar la anotación corta para crear objetos o arrays. De esta forma, un objeto tal que este:

```
var perro = new Object();
perro.color = 'negro';
perro.tamano = 'grande';
```

Podría crearse mucho más eficiente así:

```
var perro = {
  color: 'negro',
  tamano = 'grande'
};
```

Con un array pasaría lo mismo. Si tenemos algo así:

```
var series = new Array();
series[0] = 'Fringe';
series[1] = 'The Big Bang Theory';
```

Sería mucho más eficiente creándose así:

```
var series = [
  'Fringe',
  'The Big Bang Theory'
];
```

- Otra forma de optimizar código puede ser reduciendo los condicionales. De esta forma un condicional habitual tal que:

```
var direccion;
if(x > 10) {
  direccion = 1;
```

```
} else {
  direccion = -1;
}
```

Podría llegar a reducirse a un código tal que:

```
var direccion = (x > 100) ? 1 : -1;
```

- Otra acción sencilla que permite mejorar el rendimiento es el de optimizar los bucles. Un bucle habitual podría ser este:

```
var beatles = ['George','Ringo','Paul','John'];
for(var i=0;i< beatles.length;i++){
  Actuan(beatles[i]);
}
```

Tendría una opción mejorada tal que así, creando las variables sólo una vez:

```
var beatles = ['George','Ringo','Paul','John'];
for(var i=0, j= beatles.length;i<j;i++){
  Actuan(beatles[i]);
}
```

OPTIMIZACIÓN PARA JQUERY

Categoría: JavaScript

jQuery es una biblioteca de funciones preestablecidas de JavaScript que permite interactuar de forma más sencilla con los elementos DOM de una página, además de conseguir trabajar de una forma más sencilla gracias a las ampliaciones de código que existen.

Existen algunas reglas que permiten trabajar de forma más rápida y mejorada con jQuery:

- La forma más rápida de acceder a un elemento del DOM es hacerlo mediante un selector ID en vez de hacerlo con un descendiente suyo. Esto es debido a la existencia y uso de la función *getElementById()*. Esto significa que es mucho más óptimo hacer uso de algo así:

```
var boton1 = $('#boton1');
```

que no hacer uso de algo así:

```
var boton1 = $('#botones .boton1');
```

- La siguiente forma de acceder rápidamente a un elemento es hacerlo a través de un "tag" ya que se aprovecha el uso de la función *getElementsByTagName()*.
- Otro detalle a tener en cuenta es el uso de variables para almacenar información de un elemento y no aplicar cambios directamente sobre él. De esta forma tendríamos una función optimizada tal que así:

```
var $activar = $('#light input.on');
$activar.css('border', '3px dashed yellow');
$activar.css('background-color', 'orange');
$activar.fadeIn('slow');
```

Sobre unas no optimizadas que podrían ser así:

```
$('#light input.on).css('border', '3px dashed yellow');
$('#light input.on).css('background-color', 'orange');
$('#light input.on).fadeIn('slow');
```

Este sistema podría mejorar incluso un poco más haciendo uso de encadenamiento:

```
var $activar = $('#light input.on');
$activar.css('border', '3px dashed yellow').css('background-color',
'orange').fadeIn('slow');
```

- Otra forma de acceder rápidamente a subelementos es utilizar la función *find()*.

```
var $activar = $('#light'), $activo = $activar.find('input.on'), $inactivo =
$activar.find('input.off');
```

- Los eventos son uno de los elementos que más se utilizan en jQuery y debemos aprovechar la delegación de los mismos para generar el llamado *Bubbling*. Por ejemplo podemos tener una función sencilla como:

```
$('#lista li).bind('click', function() {
  $(this).addClass('clicked');
});
```

Esto implica tener que trabajar enormemente con el DOM, algo que podríamos reducir utilizando una función algo más compleja aparentemente, pero más efectiva:

```
$('#lista).bind('click', function(e) {
  var pulsa = e. pulsa, $ pulsa = $( pulsa);
  if (pulsa.nodeName === 'LI') {
            $ pulsa.addClass('clicked');
  }
});
```

- Cargar las funciones tras la carga de la ventana y no del estado del documento. Por norma general las funciones se cargan tras el *$(document).ready* pero se pueden cargar de una forma más eficiente con *$(window).load*.

ELEMENTOS DOM

Los elementos DOM de una página web son prácticamente todos aquellos elementos con los que se puede interactuar de alguna manera. Básicamente se podría resumir en que son todos los tags, ID, clases, etc...

NÚMERO ELEVADO DE ELEMENTOS DOM

Categoría: Contenidos

El hecho de que los elementos DOM sean accesibles a través de lenguajes como JavaScript puede provocar que, con un número elevado de elementos, la página se pueda llegar a sobrecargar y hacer que el acceso a estos recursos se ralentice enormemente.

Es por esto que se recomienda intentar reducir el uso de elementos DOM al mínimo y mantenerlo por debajo de los 1.000, ya que a partir de esa cifra se considera que los navegadores comienzan a perder "facultades" si han de modificar alguna cosa.

Una forma de saber cuántos elementos DOM estamos utilizando es ejecutar en la consola de Firebug la siguiente consulta:

```
document.getElementsByTagName('*').length
```

REDUCIR EL NÚMERO DE <IFRAME>

Categoría: Contenidos

Los elementos que se incluyen en un <iframe> no dejan de ser páginas dentro de otras páginas, pero con la gran diferencia de que están anidadas.

El uso de iFrames provoca que el número de elementos del DOM casi se duplique, por lo que el navegador ha de gestionar el doble de elementos, y encima anidados, de forma que gestionar o hacer algún cambio en la página puede provocar un colapso.

USAR TABLAS CON ANCHO FIJO

Categoría: Contenidos

Aunque hoy en día cada vez se utilizan menos las tablas en HTML, una forma de hacer que el navegador procese mucho más rápido las tablas es dándole las medidas de la misma. Si dejamos que los contenidos de la tabla se ajusten a las celdas, el navegador no podrá precalcular ni renderizar la misma hasta que no acabe de cargarse.

Por norma general la mejor forma de adaptar el tamaño sería haciéndolo mediante CSS y teniendo en cuenta estos tres elementos:

- Establecer el elemento CSS *table-layout* a *fixed* en la tabla.
- Definir de forma explícita los *col* para cada columna.
- Establecer el atributo *width* para cada columna.

CERRAR LOS ELEMENTOS HTML

Categoría: Contenidos

A diferencia del XML, el HTML permite el cierre de elementos (tags HTML) de forma implícita, lo que significa que algunos elementos como el P, LI, IMG... no tienen porqué tener el bloque que los cierra.

Este hecho hace que sea el navegador el que tenga que calcular cuándo comienza y acaba un bloque de elementos, lo que hace que el renderizado sea más elevado en tiempo y coste.

TAMAÑO DE LOS CONTENIDOS

Cada vez más Internet se encuentra en muchos tipos de dispositivos, y aunque sigue siendo frecuente el uso de ordenadores de mesa o portátiles, la tecnología móvil sigue creciendo de una forma exponencial, casi igualando al resto.

Está claro que no es lo mismo la potencia que puede tener un ordenador de mesa, que una videoconsola que un teléfono móvil, por lo que debemos pensar en las limitaciones de estos terminales.

DEVOLVER CONTENIDOS HTML DE MENOS DE 25KB

Categoría: Contenidos, Móvil

Muchos de los dispositivos móviles que se utilizan hoy en día tienen una caché muy limitada y que puede rondar tan sólo los 25,6 kilobytes, como es el caso de los primeros iPhone.

Debido a estas limitaciones en nuevos dispositivos se plantea como una opción interesante que cualquiera de los elementos de una página no supere esta cifra (desde el propio HTML a los JavaScript o CSS, pasando, claro está, por los elementos multimedia).

Para los JavaScript y CSS se recomienda no sobrepasar el tamaño de 1 megabyte. Si bien es cierto que es un tamaño muy elevado, cuando se llega a esta cifra comienza a haber problemas de almacenamiento.

SITIOS WEB MÓVILES

Sin duda los teléfonos móviles y las tabletas son el futuro de las máquinas y la navegación en Internet. Está claro que seguirá habiendo una gran mayoría de usuarios que navegará desde los terminales tradicionales, pero cada vez más va creciendo el uso de los sitios móviles.

Es por esto que, de la misma forma que se puede optimizar un sitio para lor ordenadores de toda la vida, podemos hacer determinadas mejoras a la hora de trabajar con dispositivos móviles.

ADAPTARSE A LA PANTALLA DEL TERMINAL

Categoría: CSS, Imágenes, Móvil

Hay que tener en cuenta que los terminales móviles tienen un tamaño de pantalla limitado. Estamos acostumbrados a las pantallas con altas resoluciones y a diseñar sitios para un ancho mínimo de 1024 píxeles, pero... ¿qué ocurre cuando tenemos pantallas de 480 píxeles?

Teniendo en cuenta esto, y gracias a los CSS podemos hacer que el diseño de las imágenes, textos y demás se adapte al terminal.

Para empezar podríamos elegir el tamaño de la imagen de la cabecera de la página en base al tamaño de pantalla:

```
/* pantalla mayor de 480px */
@media only screen and (min-device-width: 481px) {
  #header { background-image: url('header-full.png'); }
}
/* pantalla menor de 480px */
@media only screen and (max-device-width: 480px) {
  #header { background-image: url('header-small.png'); }
}
```

Esta es una opción aunque no es la única, ya que los últimos terminales llevan la posibilidad de distinguir la resolución de pantalla, por lo que otra posibilidad, en vez del tamaño, sería la calidad. Para ello usaremos los dpi (*dots per inch*) o puntos por pulgada.

```
/* alta calidad */
@media only screen and (min-resolution: 300dpi),
  only screen and (-webkit-min-device-pixel-ratio: 1.5),
  only screen and (min--moz-device-pixel-ratio: 1.5) {
  #header { background-image: url('header-300dpi.png'); }
}
/* baja calidad */
@media only screen and (max-resolution: 299dpi),
  only screen and (-webkit-max-device-pixel-ratio: 1.5),
  only screen and (max--moz-device-pixel-ratio: 1.5) {
  #header { background-image: url('header-72dpi.png'); }
}
```

ADAPTARSE A LA VELOCIDAD DE CONEXIÓN

Categoría: Conectividad, CSS, Imágenes, JavaScript, Móvil

Los navegadores del sistema operativo Android disponen desde la versión 2.2 de unas funciones en JavaScript con las que se puede detectar el tipo de conectividad que tiene el dispositivo en ese momento, de forma que podemos saber qué cantidad de ancho de banda disponemos. La función es la *navigator.connection*.

```
navigator = {
  connection: {
            "type": "4",
            "UNKNOWN": "0",
            "ETHERNET": "1",
            "WIFI": "2",
            "CELL_2G": "3",
            "CELL_3G": "4"
  }
};
```

Gracias a esta función tenemos disponible una serie de posibilidades si mezclamos un poco de JavaScript y un poco de CSS.

```
var connection, connectionSpeed, htmlNode, htmlClass;
connection = navigator.connection || {'type':'0'};
switch(connection.type) {
  case connection.CELL_3G:
            connectionSpeed = 'mediumbandwidth';
            break;
  case connection.CELL_2G:
            connectionSpeed = 'lowbandwidth';
            break;
  default:
            connectionSpeed = 'highbandwidth';
}
htmlNode = document.body.parentNode;
htmlClass = htmlNode.getAttribute('class') || '';
htmlNode.setAttribute('class', htmlClass + ' ' + connectionSpeed);
```

Gracias a esto conseguimos asignar la clase base al sitio web, que será medio, bajo o alto, según el tipo de conexión disponible. Ahora sólo queda darle forma gracias a los CSS.

```
.highbandwidth .logo   { background-image:url('logo-high.jpg'); }
.mediumbandwidth .logo { background-image:url('logo-medium.jpg'); }
.lowbandwidth .logo    { background-image:url('logo-low.jpg'); }
```

REDUCCIÓN DE PETICIONES HTTP

Categoría: Conectividad, Móvil

Aunque, en cualquier caso, siempre hay que aplicar las técnicas de reducción de peticiones HTTP generales, no está de más hacer hincapié en algunos detalles para los teléfonos móviles de última generación.

Hay que tener en cuenta que, ya de por sí, las cabeceras HTTP en los terminales móviles pueden ser un 25% más amplias que en los navegadores de escritorios, al tener que enviar por cada petición la lista más detallada de elementos MIME que soportan, entre otras cosas.

- Usar CSS3 para aquellas opciones en las que se necesitaban imágenes y similares (como los bordes redondeados). Los terminales como Android o Safari soportan *border-radius*, *text-shadow*, *background linear/radial gradients* o *box-reflect*.

- Usar en los CSS y HTML codificación en base64 (método *data:* en base64 para imágenes, principalmente).
- Evitar redirecciones, como por ejemplo la del acceso a una página que cambia la URL a una versión móvil.
- Cachear la mayor cantidad de elementos posibles, incluido el AJAX. Incluso, en los casos en los que sea posible, usar el sistema de cacheo de HTML5 (*cache-manifest* y *client-side database*).
- Intentar evitar el uso de cookies y pasar al uso de LocalStorage.
- Externalizar los CSS y JS a ficheros externos de forma que se puedan cachear con mucha más facilidad y sólo sea necesaria una petición al visitar el sitio por primera vez.
- Hay sistemas como iOS 2.2 que disponen de una lista de emoticonos[2] integrados en el sistema, y que mediante códigos HTML se pueden mostrar en cualquier página.

REDUCIR EL USO DE JAVASCRIPT

Categoría: JavaScript, Móvil

Sin duda que los terminales de última generación lleven navegadores en los que se puede usar la última especificación de JavaScript es un gran paso, además de estar adaptado al uso de CSS3. Gracias a esto algunas funcionalidades que se convertían en pesadas se pueden realizar de formas mucho más sencillas.

Para las transiciones, anteriormente se usaba una serie de *timeout* para controlar los tiempos. Ahora, los navegadores basados en WebKit, Firefox u Opera disponen de funciones tal que:

- W: onwebkittransitionend / F: ontransitionend / O: onotransitionend
- W: onwebkitanimationstart / F: onanimationstart / O: onoanimationstart
- W: onwebkitanimationiteration / F: onanimationiteration / O: onoanimationiteration
- W: onwebkitanimationend / F: onanimationend / O: onoanimationend

Otros detalles a tener en cuenta son los formularios. HML5 permite introducir unos campos que ya incorporan su propia validación de forma interna, como el campo de tipo *email* por lo que ya no será necesario hacer uso de JavaScript para controlarlos.

Gracias a la base de datos *indexedDB* que se ha incorporado en iOS 4.2 y Android 2.2 es posible almacenar mucha información con un fácil acceso.

De la misma forma, el sistema de GeoLocalización es muy útil si se necesita, pero en vez de solicitarlo por cada página vista es mejor almacenarlo en la sesión y usarlo como base al menos durante un espacio de tiempo en el que no pueda variar en exceso.

[2] http://webperformanceoptimization.es/emoticonos-ios/

MEJORANDO EL RENDIMIENTO DE ELEMENTOS COMUNES

En muchas ocasiones nos encontramos códigos bastante genéricos que se usan en muchos sitios web, como puede ser el de Google Analytics. Aunque en este caso Google se preocupa por darnos la mejor solución... ¿es realmente la óptima?

CÓDIGO DE GOOGLE ANALYTICS

Categoría: Conectividad, Contenidos

Uno de los códigos que muchos sitios web tienen es el código de Google Analytics. Aunque Google se preocupa por darnos las mejores opciones siempre se puede mejorar, aunque suponga cierto trabajo por nuestra parte, que será de agradecer para los usuarios.

Básicamente hay 3 cosas que podemos hacer para mejorar el rendimiento del código:

- Colocar el código en el pie de página.
- Cachear en el servidor local el fichero JavaScript.
- Eliminar el cálculo de HTTP Seguro.

Por normal general Google, con su código asíncrono, nos pide que coloquemos el código justo antes del </head>. Esto puede generar un bloqueo en la paralelización de peticiones, por lo que es mejor incluirlo en el pie de página, justo antes del </body>.

Si miramos el código original podremos ver que siempre se hace una petición al archivo http://www.google-analytics.com/ga.js y que este se cachea durante tan sólo 1 hora. En este caso, a sabiendas de que el código de Google Analytics no cambia con tanta frecuencia, lo mejor es descargarlo (si puede ser de forma automática con un Cron cada 6 horas, por ejemplo, mejor) y tenerlo en el propio servidor estático.

Para acabar, la mayoría de sitios web suelen estar en protocolo HTTP y no en el HTTP Seguro. Google Analytics está pensado para ser un código estándar y eso implica que ha de calcular, por cada impresión, si el sitio está en seguro o no.

Con todo esto, podríamos generar un código para Google Analytics mucho más sencillo y que puede llegar a funcionar hasta un 15% más rápido que el que nos ofrece la propia Google.

```
<script type="text/javascript">
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-XXXXXXXXX-1']);
  _gaq.push(['_trackPageview']);
  (function() {
            var ga = document.createElement('script');
            ga.type = 'text/javascript';
            ga.async = true;
            ga.src = 'http://www.example.com/ga.js';
            var s = document.getElementsByTagName('script')[0];
            s.parentNode.insertBefore(ga, s);
  })();
</script>
</body>
```

Este sería el código por defecto simplemente eliminando la primera petición DNS y el protocolo seguro... pero ¿se puede mejorar el código propio JavaScript? La respuesta es sí, siempre y cuando usemos el código "general".

```
<script type="text/javascript">
var _gaq = [['_setAccount', 'UA-XXXXX-X'], ['_trackPageview'],
['_trackPageLoadTime']];
setTimeout((function(d, t) {
  var g = d.createElement(t), s = d.getElementsByTagName(t)[0];
  g.async = 1;
  g.src = '//www.example.com/ga.js';
  s.parentNode.insertBefore(g,s);
} , 0);
</script>
```

Este nuevo código mejora la carga de las variables, añade el seguimiento de tiempos de carga del propio Google Analytics y es compatible con el protocolo que sea. Además, se carga con un *setTimeout()* para no forzar el renderizado.

GOOGLE ANALYTICS PARA MEDIR LA VELOCIDAD DE CARGA DE UNA PÁGINA

Categoría: Contenidos

Hace un momento os comentaba que como punto diferencial el código de Google Analytics debería estar al final de la página, algo que nos permite hacer un pequeño *hack* para medir el tiempo de carga de una página y enviarlo a Google Analytics y así poder jugar con los filtros avanzados y saber qué páginas van más rápido y cuáles más lentas, y así poder tomar decisiones.

Eso sí, una cosa a tener en cuenta es que esto no deberíamos tenerlo siempre, sino usarlo con usuarios de prueba, administradores y usuarios aleatorios.

El primer paso es, justo después del <head> incluir un código tal que así:

```
<head>
  <script type="text/javascript">
        var tiempoini = new Date();
  </script>
```

En este momento pondremos un contador de tiempo en marcha. Ahora, justo en el código de Google Analytics haremos ciertos cambios para calcular cuánto ha tardado en cargar el código de la página y enviarlo al sistema de analítica.

Para ello haremos algunos cambios tal que así:

```
<script type="text/javascript">
  window.onload=function() {
        var tiempofin = new Date();

        var tiempocarga = tiempofin.getTime() - tiempoini.getTime();
        if(tiempocarga <1000)
              textocarga = "MuyRapido";
        else if (tiempocarga <2000)
              textocarga = "Rapido";
        else if (tiempocarga <3000)
              textocarga = "Normal";
        else if (tiempocarga <5000)
              textocarga = "Regular";
        else if (tiempocarga <10000)
              textocarga = "Lento";
        else
              textocarga = "MuyLento";
        var direccion = document.location.pathname;
        if( document.location.search)
              direccion += document.location.search;
        try {
              var _gaq = _gaq || [];
              _gaq.push(['_setAccount', 'UA-XXXXXXXXX-1']);
              _gaq.push(['_trackEvent','TiempoCarga (ms)',
textocarga + ' PaginaCarga', direccion, tiempocarga]);
              _gaq.push(['_trackPageview']);
              (function() {
                    var ga = document.createElement('script');
ga.type = 'text/javascript'; ga.async = true;
                    ga.src = 'http://www.example.com/ga.js';
```

```
                                    var s =
document.getElementsByTagName('script')[0];
                                    s.parentNode.insertBefore(ga, s);
                            })();
                    } catch(err){}
  }
</script>
```

Gracias a esto conseguiremos ver en los eventos de Google Analytics los textos que indicarán el tiempo de carga de las distintas páginas agrupadas según la velocidad y así poder tomar decisiones en caso de ser necesario.

CÓDIGO DE GOOGLE ADSENSE

Categoría: Conectividad, Contenidos

Al igual que el código de Google Analytics, sin duda el código de Google AdSense es otro de los más utilizados en la red, y aunque internamente se han hecho cambios para mejorar su rendimiento, la forma de cargar tiene una situación más comprometida debido a que se utiliza la función *document.write* que impide su carga asíncrona.

La única forma, por ahora, que tenemos, es la de mejorar la carga del fichero JavaScript reduciendo la petición DNS en primera instancia y cacheándolo con más frecuencia. Por defecto Google lo almacena durante 1 día. Nosotros también podemos hacerlo, descargándolo de forma automática con un Cron cada 6 horas y tenerlo en nuestro propio servidor estático.

```
<script type="text/javascript"><!--
  google_ad_client = "ca-pub-xxxxxxxxxx";
  google_ad_slot = "xxxxxxxxxx";
  google_ad_width = 728;
  google_ad_height = 90;
//--></script>
<script type="text/javascript"
src="http://www.example.com/show_ads.js"></script>
```

TIEMPOS DE INACTIVIDAD DEL SITIO WEB

Categoría: Conectividad, Contenidos

Uno de los problemas más habituales que nos podemos encontrar a la hora de actualizar un sitio es qué hacer en el tiempo en el que la web está caída debido a la actualización. En muchos casos podemos hacer una actualización transparente pero en otros casos no.

En estos casos deberíamos aplicar el código HTTP 503 Service Temporarily Unavailable que evitará que los usuarios puedan acceder a contenidos indebidos, o que los motores de búsqueda comiencen a indexar el mensaje de error como páginas correctas.

Este código deberá ir acompañado siempre de la fecha siguiente en la que el sitio debería comenzar a ser rastreado de nuevo. Por ejemplo tendríamos un código en PHP tal que así:

```
header('HTTP/1.1 503 Service Temporarily Unavailable');
header('Retry-After: Sat, 8 Oct 2011 18:27:00 GMT');
```