Cómo funciona una aplicación Web

Escribir sobre cómo funciona una aplicación Web podría tomar varios libros. En este apartado y el siguiente vamos a ver una pequeña introducción sobre este tema; si desea profundizar más, puede ampliar material en Internet o en algún libro especializado.

Cuando se realizan aplicaciones Web con ASP.NET u otros lenguajes de desarrollo, tenemos la sensación de que todo funciona como si fuera magia negra, o que "algo no va bien", especialmente si el desarrollador viene del mundo de las aplicaciones de escritorio.

Para empezar, *¿por qué me hace falta un servidor?, ¿no puedo publicar mi aplicación en mi ordenador y que la gente acceda?, ¿cómo se comunica mi ordenador con él?...*

- Un servidor Web tiene asociada una dirección IP fija (normalmente nuestros equipos de trabajo tienen asociada una IP dinámica o interna, y usualmente, cada vez que nos conectamos obtenemos una distinta). Esto para nosotros es como si el servidor tuviera un número de teléfono al que llamar para dialogar con él.
- El servidor Web tiene abierto el puerto 80 y acepta peticiones de páginas a través de ese puerto. Normalmente, nuestros PC personales tienen cerrados a cal y canto todos los puertos (ya sea con un *firewall* de la red local, con el mismo *firewall* de Windows, o ambos) para evitar ataques de intrusos (ver figura 1.3).

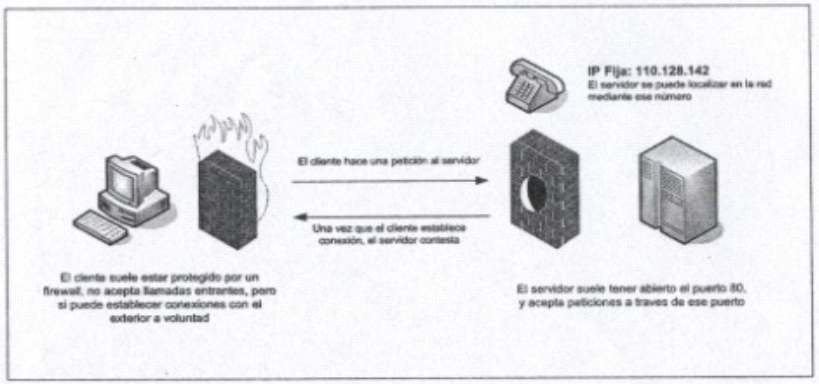

figura 1.3 Cómo puede establecer conexión un cliente con un servidor Web

nota

Aclaración: La dirección IP es el número con el que identificamos a un ordenador conectado a Internet (ej. 95.25.1.25), mientras que un nombre de dominio (ej.: http://www.dotnetmania.com) se corresponde con una dirección IP.

¡Ah! entiendo..., entonces yo abro una conexión con un servidor Web, le voy haciendo peticiones e intercambiando información y cuando termino cierro la conexión con él, ¿es así? **No**, y aquí es donde comienzan los problemas:

- Las aplicaciones Web funcionan sobre el protocolo HTTP (HTTPS para aplicaciones seguras).
- Este protocolo se basa en que el cliente hace una petición al servidor, éste le contesta, y aquí se acabó toda la relación entre las dos partes.
- El protocolo HTTP no guarda estado, esto es, cada vez que queramos pedirle una página al servidor es como si fuéramos un nuevo usuario accediendo al sistema (como si estuviéramos tratando con un persona que hubiera perdido la memoria y cada vez que habláramos con ella tuviéramos que decirle quién somos).

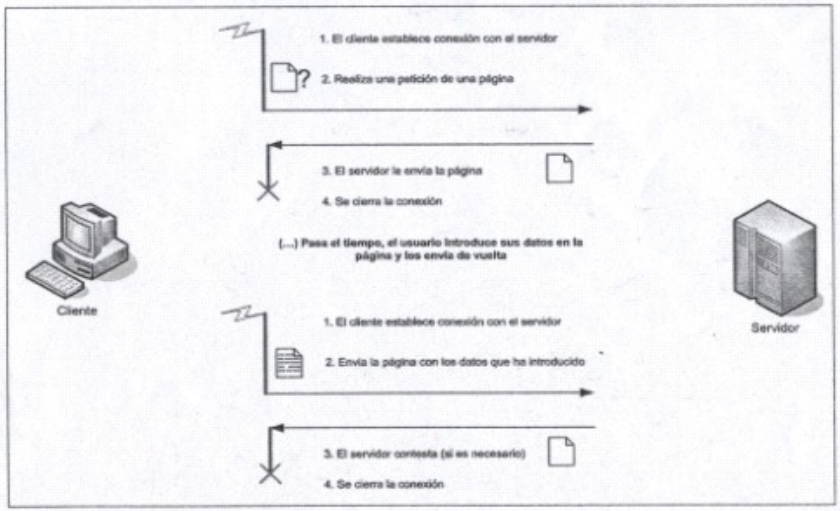

figura 1.4 El protocolo HTTP no mantiene la conexión ni datos de estado

¡Esto es un problema enorme! ¿Qué pasa si quiero hacer una aplicación Web para vender libros?, ¿cada vez que el usuario escoge un libro todo vuelve a empezar? Para el protocolo HTTP sí, pero afortunadamente, los *frameworks* de desarrollo Web tienen varias formas de mantener el estado de un cliente; esto lo veremos en el siguiente apartado.

Dónde se almacenan los datos/estado

HTTP es un protocolo sin estado, pero existen varios mecanismos tanto para mantenerlo como para guardar la información que nos haga falta. Vamos a ver, de forma resumida, lo que tenemos disponible.

- **Cookies:** Consiste en almacenar en el cliente un trocito de texto para después poder recuperarlo desde el servidor.

  - *Ejemplo*: ¿Se ha preguntado alguna vez cómo ciertos portales recuerdan quién es usted como por arte de magia? Una vez el usuario se autentifica, se guarda una *cookie* en el navegador del cliente. La próxima vez que acceda a dicha páginas, el servidor Web le pregunta al navegador si existe la *cookie* que identifica al usuario.
  - *Ventajas*: No carga al servidor (se almacena en cliente). Además, una *cookie* puede almacenarse durante un largo periodo de tiempo.
  - *Desventajas*: El tamaño es limitado, ya que solo almacenan texto (para guardar privacidad algunos sitios encriptan sus *cookies*), y algunos usuarios tiene configurados sus navegadores para no aceptar *cookies* por motivos de seguridad.

- **Campos ocultos:** En un formulario Web el usuario suele rellenar los datos y enviarlos al servidor para que los procese. Uno de los trucos que se empleaba en ASP 3.0 para mantener información de estado en una página Web era insertar campos ocultos en los formularios e introducir la información en ellos.

  - *Ejemplo*: Supongamos que en una página permitimos modificar información acerca de un cliente y no queremos mostrar el identificador interno del cliente al usuario; para ello almacenamos el identificador en un campo oculto, y cuando el usuario modifica los datos y los envía al servidor, obtenemos el identificador del campo oculto y podemos realizar la actualización en nuestra base de datos.

  - *Ventajas*: No carga al servidor, puesto que los datos se almacenan en la página, y los navegadores suelen permitir trabajar con campos ocultos.

  - *Desventajas*: Poco seguro, ya que se puede ver y modificar la lógica de nuestra aplicación editando el código fuente de la página. Además no debemos almacenar grandes cantidades de información ya que ésta viaja en cada *Round trip* que hagamos de la página al servidor y además la información se mantiene en una sola página.

- **Query string:** Consiste en añadir a la URL de nuestra página los parámetros que veamos convenientes para almacenar la información que necesitemos.

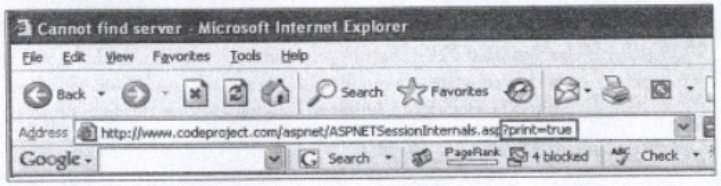

figura 1.5 Figura 1.5 Ejemplo de *query string*

  - *Ejemplo*: Podríamos usar un parámetro de *query string* para saber si nuestra página está en modo insertar/modificar o, tal como se muestra en la figura 1.5, si la página se debe visualizar respetando márgenes ya que está en modo "imprimir".
  - *Ventajas*: Fácil de usar y todos los navegadores aceptan estos parámetros.
  - *Desventajas*: Tamaño limitado, solo almacenan texto, y ofrecen muy poca seguridad, ya que se puede ver y modificar la lógica de nuestra aplicación, solamente cambiando un parámetro desde la barra de navegación (también hay aplicaciones Web que optan por encriptar los parámetros).

- **View state:** Permite almacenar información de estado y valores de controles en la propia página. Es una abstracción de ASPX muy cómoda de usar y que internamente se implementa como un campo oculto codificado. Permite enlazar automáticamente el contenido de los controles HTML con los controles de servidor.

  - *Ejemplo*: Podemos distinguir si nuestra página está en modo visualización de datos o edición (añadiríamos una entrada al *View state*).
  - *Ventajas*: Todos los controles ASP.NET la utilizan. Aunque solo puede almacenar texto, podemos serializar un objeto como XML y almacenarlo en el *View state*. La información se codifica, aunque esta codificación se pueda romper, no es algo tan claro como un campo oculto con texto en plano (si queremos añadir más seguridad podemos encriptar el *View state*), y funciona en granjas Web, ya que la información se almacena en cliente, al igual que pasa con los campos ocultos.
  - *Desventajas*: No debemos almacenar grandes cantidades de información en el *View state*, ya que ésta viaja en cada *round trip* que hagamos de la página al servidor, además la información se almacena en una sola página (no podemos compartirla con otras páginas).

- **Session state:** Almacena datos de un usuario en el servidor. Esta información estará activa mientras el usuario esté navegando por nuestro sitio, una vez que sale del mismo la sesión se destruye. ¿Cómo sabe el servidor que la petición viene de un usuario concreto? Bien usando una *cookie* que lo identifique o, si el navegador no admite *cookies* codificando la misma como parámetro de la URL.

  - *Ejemplo*: En una aplicación de comercio electrónico, podíamos almacenar el carrito de la compra del usuario en una variable de sesión.
  - *Ventajas*: Permite almacenar todo tipo de datos, la información almacenada en sesión se guarda en servidor, no viaja incrustada en las páginas, y se puede configurar para que funcione en una granja de servidores (utilizando un servidor de sesión).
  - *Desventajas*: Al almacenarse los datos en el servidor corremos el riesgo de saturarlo si hacemos un mal uso de estas variables o si el servidor tiene muchos usuarios simultáneos. También hay que tener en cuenta que los datos almacenados en esta área se destruyen una vez que el usuario abandona la sesión.

- **Application state:** Si bien las variables de sesión permiten almacenar información acerca de un usuario, las de aplicación permiten almacenar información común a todas las sesiones. Salvo casos muy justificados, debemos evitar usar este tipo de almacenamiento (en la mayoría de los casos es mejor usar objetos del tipo "cache").

  - *Ejemplo*: Nos podría servir para controlar cuantos usuarios están accediendo a nuestra aplicación Web o, por ejemplo, detectar si dos usuarios distintos se intentan conectar con un mismo *login* a la vez.
  - *Ventajas*: Permite gestionar información global a toda la aplicación Web.
  - *Desventajas*: La información se almacena en el servidor, no es buena opción si la aplicación se ejecuta en una granja de servidores, y si reiniciamos la aplicación se pierde toda la información.

- **Bases de datos:** La mayoría de mecanismos que hemos visto hasta ahora tenían caducidad: *x* días, duración de la sesión del usuario, o mientras la aplicación Web no se reiniciara. Cuando queremos almacenar información que debe persistir en el tiempo se utilizan las bases de datos

  - *Ejemplo*: Queremos guardar la dirección postal de nuestros clientes para que no tengan que volver a introducirla cada vez que realicen un nuevo pedido.
  - *Ventajas*: No se pierden los datos al reiniciar la aplicación, permite organizar y estructurar los datos de forma lógica y funciona con granjas de servidores.
  - *Desventajas*: La información se almacena en servidor, y si queremos almacenar información temporal, debemos proveer mecanismos para que ésta se elimine de la base de datos pasado un tiempo, por ejemplo, cuando el tiempo de sesión expire.

- **Ficheros:** Un fichero nos permite almacenar información y que ésta se mantenga en el tiempo, sin tener que tener instalado un motor de base de datos.

  - *Ejemplo*: Podemos almacenar los *settings* de una aplicación Web (cadena de conexión a base de datos, tiempo de vida de sesión…) en un fichero de configuración XML, lo que nos permitirá modificar dichos *settings* sin tener que tocar código de la aplicación.

- *Ventajas*: No es necesario configurar ningún motor de base de datos, la información se puede estructurar de forma lógica (sobre todo si usamos formato XML), y si se reinicia la aplicación los datos no se pierden.
- *Desventajas*: No es aconsejable usarlo para información que no va a ser de solo lectura (tendríamos que implementar control de concurrencia), y los datos se almacenan en el servidor.

- **Variable estáticas:** Es parecido al *Application state* que vimos en un punto anterior. Podemos crear variables estáticas en el fichero `global.asax` y que éstas puedan ser accedidas desde cualquier parte de la aplicación.

  - *Ejemplo*: Nos podría servir para controlar cuantos usuarios están usando nuestra aplicación Web o, por ejemplo detectar si dos usuarios distintos se intentan conectar con un mismo login a la vez.
  - *Ventajas*: Permite gestionar información global a toda la aplicación Web y es más eficiente que el *Application state*.
  - *Desventajas*: La información se almacena en servidor, y si reiniciamos la aplicación se pierde toda la información.

- **Profiles (perfiles):** Son similares a las variables de sesión pero con una gran diferencia: la información del usuario se puede mantener más allá de lo que dure la sesión.

  - *Ejemplo*: Podríamos dejar a un usuario que configurara la página de inicio a su gusto (por ejemplo, color de fondo, que elementos deberían aparecer en portada…), y guardar dicha información en un perfil; de esta forma la siguiente vez que entrara, la página automáticamente se adaptaría a sus preferencias.
  - *Ventajas*: No se pierden los datos al reiniciar la aplicación, soporta usuarios autentificados y anónimos.
  - *Desventajas*: La información se almacena en el servidor, si nuestro sitio tiene bastante tráfico y permitimos usar perfiles para usuarios anónimos el servidor se podría saturar.

nota

Para saber más: Este resumen está basado en un excelente artículo que se publicó en **dotNetManía**, octubre 2005: "Mecanismos para el mantenimiento del estado en ASP.NET".