Programación en WWW usando CGI

Jesús Vegas

Departamento de Informática
Universidad de Valladolid
España
jvegas@infor.uva.es

Esquema

- Introducción
- La Interacción entre el *Browser* y el Servidor
- Formularios HTML
- Un Formulario de Ejemplo
- Leyendo los Datos CGI
- Instalando un *Script* CGI
- Consideraciones de Seguridad
- Referencias Bibliográficas
- Ejemplos

Introducción I

El objetivo de este curso es explicar el modo en el que se puede utilizar el CGI para interactuar con un documento HTML transmitido según el protocolo HTTP, a través de la WWW.

- Los navegadores presentan los documentos HTML, habitualmente estáticos.
- Con formularios, los docs HTML se vuelven interactivos, pueden recoger información.
- Esta información puede ser tratada en el servidor Web por los *scripts* CGI.

Introducción II: WWW

- Word Wide Web, WWW es un sistema hipermedia interactivo desarrollado sobre Internet.
- Hipermedia: juntar texto, imágenes, audio y vídeo dentro de un mismo envoltorio llamado documento
- Crear un sistema de difusión del conocimiento.
- WWW se asienta sobre el protocolo HTTP (Hyper Text Transfer Protocol) y sobre el lenguaje de definición de documentos hipermedia HTML (HyperText Markup Language)

Introducción III: HTTP

- HTTP es el protocolo que permite que CGI pueda funcionar, y determina qué se puede o no mandar y recibir via CGI.
- URI, *Universal Resource Identifier*, URL, URN.
- Arquitectura Cliente–Servidor.
- Protocolo sin conexión y sin estado.
- Tipos de datos MIME (*Multipart Internet Mail Extension*).
- Transacción HTTP compuesta por una cabecera, y opcionalmente, por una línea en blanco seguida de los datos.

Interacción Cliente–Servidor

- El cliente solicita el documento `doc1.html` al servidor intercambiando la siguiente transacción HTTP:

```
GET /doc1.html HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/gif
User-Agent:  Lynx/2.2  libwww/2.14
From: jvegas@infor.uva.es
/* una linea en blanco */
```

- El servidor responde mandando la siguiente transacción HTTP:

```
HTTP/1.0  200 OK
Date: Friday, 03-Jul-98 12:00:00 GMT
Server: Apache/1.1.1
Content-type: text/html
Content-length: 230
/* una linea en blanco */
<HTML><HEAD><TITLE> ........ </HTML>
```

EJECUTANDO *Scripts* I

- Un URL puede identificar un fichero que contiene un programa, en lugar de un documento HTML.
- De modo que este fichero debe ser ejecutado cuando es solicitado por el cliente, en lugar de devolver su contenido.
- La diferencia entre un fichero que contiene un documento HTML y un programa está en el modo y/o el lugar en que está almacenado en el servidor.
- Estos programas llamados *scripts* CGI pueden estar escritos en cualquier lenguaje que se pueda ejecutar sobre la máquina que mantiene el servidor Web (C, TCL o Perl).

EJECUTANDO *Scripts* II

- Como ejemplo podemos ver un *script* CGI (`cgi-bin/hola`) escrito en el lenguaje de comandos del *shell* cuyo resultado será un documento HTML con un mensaje de bienvenida:

```
#!/usr/bin/ksh
echo "Content-type: text/html"
echo ""
echo "<HTML>"
echo "<HEAD><TITLE>hola</TITLE></HEAD>"
echo "<BODY>hola.</BODY></HTML>"
echo "</HTML>"
```

- La petición se realiza desde el siguiente documento HTML:

```
<HTML> <HEAD> <TITLE> Script CGI </TITLE> </HEAD>
<BODY>
<P>Pulsa
<A HREF="http://descartes.dcs.fi.uva.es/cgi-bin/hola">aquí</A>
para recibir un saludo.</P>
</BODY>
</HTML>
```

Ejecutando *Scripts* III

- Se puede dar un paso más en este proceso cuando el *script* CGI reciba datos del cliente, los gestione, y devuelva una respuesta adecuada al cliente.
- Esto es lo que necesitamos cuando queremos hacer accesible al mundo WWW una BD clásica (los *scripts* CGI como interfaz entre WWW y la BD).
- doc HTML + formulario → CGI + BD → doc HTML respuesta.
- Para ello, necesitaremos presentar un formulario al cliente, y una vez rellenado, enviar la información recogida al servidor especificado en el formulario.
- Los formularios en HTML se definen utilizando un conjunto de etiquetas HTML específico que vamos a presentar a continuación.

Formularios I

- Un formulario es una interfaz gráfica de usuario, GUI, que permite la inclusión de botones de diferentes tipos, areas de texto, listas de selección, etc.
- Cada elemento tiene un par de atributos asociado: `NAME` y `VALUE`
- Cuando el usuario presiona un botón indicando que el formulario debe ser enviado, *submitted*, la información en el formulario es mandada al servidor para ser procesada por un *script* CGI.
- Codificación URL:

```
nombre=Pepe&apellidos=P%e9rez+Pardo
```

Formularios II

```
<FORM ACTION="http://www.sitio/el_script"
      METHOD="POST">
</FORM>
```

- `ACTION` es el URL del *script* CGI que debe tratar los datos recopilados por el formulario.
- `METHOD` determina el método a utilizar para acceder al URL determinado en el campo `ACTION`.
  - `METHOD="GET"` El *script* CGI recibe la información en `QUERY_STRING` (restricciones en su longitud).
  - `METHOD="POST"` El *script* CGI recibe la información del formulario por su entrada estandar, se especifican los *bytes* en `CONTENT_LENGTH`.
- `ENCTYPE` es `application/x-www-form-urlencoded`.

Elementos de Formulario I

- Dentro de un formulario podremos recoger información utilizando para ello los distintos elementos disponibles.
- La forma más usual es permitiendo que el usuario introduzca texto, aunque también le podemos elegir una opción entre una lista, pulsando uno o más botones, etc.
- Todo ello se puede realizar combinando tres etiquetas HTML: `INPUT`, `TEXTAREA` y `SELECT`.

Elementos de Formulario II

```
<INPUT TYPE="tipo" NAME="nombre" VALUE="valor"
       SIZE="tamaño" MAXLENGTH="max_longitud" CHECKED>
```

- `TYPE`: el tipo de entrada que se va a utilizar: `text`, `password`, `checkbox`, `radio`, `submit`, `reset`, `hidden`.
- `NAME`: nombre simbólico al que se le va asociar el valor.
- `VALUE`: (significado dependiente del elemento).
- `SIZE`: tamaño físico del campo de entrada en elementos texto o `password` (por defecto es 20).
- `MAXLENGTH`: número máximo de caracteres que se permiten en un elemento de tipo texto o `password` (ilimitado por defecto).
- `CHECKED`: activación por defecto de botones.

Elementos de Formulario III

```
<TEXTAREA NAME="nombre"
          ROWS="filas" COLS="columnas">
texto por defecto
</TEXTAREA>
```

- Se presenta una caja donde el usuario puede introducir varias líneas de texto.
- La caja de texto tendrá barras de desplazamiento y unas dimensiones de `ROWS`$\times$`COLS`.
- El texto por defecto (optativo) aparecerá entre las etiquetas de apertura y cierre del area de texto.

Elementos de Formulario IV

```
<SELECT NAME="nombre" SIZE="num" MULTIPLE>
<OPTION VALUE="valor_asociado_A" SELECTED>Opción A
<OPTION VALUE="valor_asociado_B" >Opción B
<OPTION VALUE="valor_asociado_C" >Opción C
</SELECT>
```

- `SIZE`: número de opciones visibles. Si es 1, el elemento selección tendrá la apariencia de un menú desplegable.
- `MULTIPLE`: permite seleccionar varias opciones.
- `VALUE`: que indica el valor que se asociará al elemento de selección en caso de que esa opción sea la elegida. Este atributo puede obviarse, en cuyo caso el valor devuelto será la descripción de la opción misma.
- `SELECTED`: la opción seleccionada por defecto.

Formularios: 3.2 v 4

- El atributo `accesskey` provee el acceso directo a los campos de formulario desde el teclado.
- Con el atributo `disabled` se puede deshabilitar un elemento del formulario.
- Se pueden declarar campos de sólo lectura con el atributo `readonly`.
- Las etiquetas de los botones de tipo `radio` y `checkbox` se vuelven sensibles.
- Validación de los campos en el cliente: atributo `onchange-INPUT` y lenguajes como *javascript*.
- Los botones pueden ahora estar etiquetados con figuras o texto formateado.

Entrada CGI I

- Leer la cadena de pares nombre=valor. GET de la variable QUERY_STRING y POST de la entrada estándar CONTENT_LENGTH *bytes*.
- Decodificar URL.
- Variables de entorno:
- AUTH_TYPE el método de autentificación utilizado para validar al usuario, si el servidor soporta la autentificación de usuarios.
- CONTENT_LENGTH la longitud de la información mandada por el cliente.
- CONTENT_TYPE el tipo de información asociada a las consultas.
- GATEWAY_INTERFACE la revisión de la especificación CGI que soporta el servidor, con el formato CGI/revisión.

ENTRADA CGI II

- PATH_INFO la información asociada al *path* virtual, tal cual es dada por el cliente, y que es decodificada por el servidor.
- PATH_TRANSLATED el *path* físico obtenido a partir del PATH_INFO.
- QUERY_STRING la información que sigue a la '?' en el URL.
- REMOTE_ADDR la dirección IP del *host* remoto que realiza la petición.
- REMOTE_IDENT el nombre de usuario remoto, si el servidor HTTP soporta la identificación RFC 931.
- REMOTE_USER el nombre de usuario autentificado si el *script* CGI está protegido y el servidor soporta autentificación de usuarios.
- REQUEST_METHOD el método mediante el cual se hizo la petición.

ENTRADA CGI III

- `SCRIPT_NAME` el *path* virtual del programa que se esta ejecutando.
- `SERVER_NAME` el nombre de *host* del servidor, alias DNS, o dirección IP.
- `SERVER_PORT` el número de puerto al que fue mandada la petición.
- `SERVER_PROTOCOL` el nombre y revisión del protocolo utilizado para esta petición.
- `SERVER_SOFTWARE` el nombre y revisión del software del servidor que responde a la petición.

Salida CGI

- `GET` y `POST` tienen en común que su salida estándar está conectada con el cliente que hizo la petición.
- El resultado de la ejecución de un *script* CGI será recibido por el cliente que realizó la petición como un documento HTML

Instalando un CGI

- Todo depende del servidor Web: consultar al *webmaster* por el servidor, el camino donde instalar el *script*, cómo llamarlo, los permisos que debe tener, y si el *script* necesita escribir sobre algún fichero, qué permisos debe tener ese fichero.
- Caso de estudio: servidor Apache con tres ficheros de configuración: `httpd.conf`, `srm.conf` y `access.conf`, normalmente en `/usr/local/apache` o similar.
- Directivas en `srm.conf`:

```
ScriptAlias  /cgi-bin/    /usr/local/apache/cgi-bin/
AddHandler   cgi-script   .cgi
```

Seguridad en CGI I

- Los *scripts* CGI son, sin querer, una puerta por donde pueden entrar los problemas dando acceso al *shell* de las máquinas donde está el servidor Web, permitiendo el acceso a información restringida, o posibilitando que el sistema sufra sabotajes del exterior.
- Hay que intentar evitar que nuestros *scripts* se queden colgados por cualquier razón, y como respuesta a ese fallo se deje acceso al *shell* desde el que se estaban ejecutando, abriendo así las puertas a un acceso no controlado.
  - No confiar en nada.
  - Cuidado con los nombres de ficheros.
  - Cuidado al invocar programas.

Seguridad en CGI II

- La idea principal es que nunca se debe confiar en nada, especialmente en los datos de entrada de los formularios. Las siguientes ideas son falsas:
  - Si se ha puesto una lista de selección, la entrada será una de las opciones posibles.
  - Si la longitud máxima de un campo de entrada está fijada, el *browser* mandará como mucho esos caracteres.
  - Los campos de la variable `QUERY_STRING` se corresponderán con los de la página.
  - El contenido de la variable `QUERY_STRING` será alguno de los válidos que pueden ser transmitidos por HTTP.

SEGURIDAD EN CGI III

- Problablemente los nombres de los ficheros codificados en los programas CGI son seguros, pero los nombres obtenidos de los formularios, `PATH_INFO` por ejemplo, son sospechosos y hay que comprobarlos.
- Lista de nombres válidos que eviten considerar, por ejemplo, como inválido cualquier nombre que comience por '/'.
- Nunca ha de permitirse que comiencen por '`.`' ni por '`;`' (lista de caractéres válidos).
- Si la máscara `UMASK` por defecto de muchos servidores Web es `0`, cualquier fichero creado por un *script* CGI podrá ser escrito por cualquiera.
- La máscara `UMASK` debe ser puesta a `022` o a `077`.

Seguridad en CGI IV

- Muchos *scripts* CGI invocan a otros programas, ya sean creados por el usuario o del sistema, lo que supone el principal agujero de seguridad en CGI.
- El problema básico es que por ejemplo un *script* CGI llame a `grep` con una expresión regular de cualquiera de las maneras siguientes (en Perl o C):

  ```
  system("grep $exp database");
  sprintf(comando,"grep %s database",exp);system(comando);
  ```

  tiene varios problemas, sobre todo si consideramos que `exp` puede valer "`root /etc/passwd; rm`", con lo que lee el fichero de *passwords* y además inutiliza el sistema.

Referencias Bibliográficas

- *The World Wide Web Consortium*: `http://www.w3c.org/`
- *CGI: The Common Gateway Interface*: `http://hoohoo.ncsa.uiuc.edu/cgi/`
- *CGI: The Common Gateway Interface for Server-side Processing*: `http://Stars.com/Authoring/CGI/`
- *Apache Proyect*: `http://www.apache.org/`
- *Browser Netscape*: `http://home.netscape.com`
- *Java Technology Home Page*: `http://java.sun.com/`
- *The Perl Institute*: `http://www.perl.org/`