Revision of Server-Side Languages Material

SET09103 Advanced Web Technologies

School of Computing
Napier University, Edinburgh, UK
Module Leader: Uta Priss

2008

Outline

This is HTML:

```
<html>
<head><title>My page</title></head>
<body>
<h3>Big headline</h3>
Click <a href="http://somewhere.co/index.html">here</a>.
</body>
</html>
```

Dynamic Content

Some HTML documents provide **dynamic** content. This content

- is generated by a computer program;
- can retrieve information from a database;
- can respond to a specific user request (e.g. a web form);
- is converted into an HTML page;
- which is retrieved via HTTP by the user's browser.

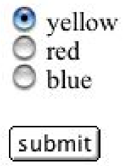
Advantages of Dynamic Content

Dynamic content

- is more flexible than static content (e.g. on-line newspapers);
- can respond to specific user requests (e.g. e-commerce);
- can collect user information (e.g. on-line surveys, guest-books);
- can provide an interface to a database (e.g. search engines);
- can facilitate basic user interaction (e.g. on-line shopping).

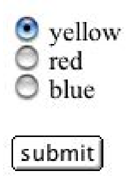
HTML content can be dynamically generated in response to a form which a user has filled in.

For example, a user could be asked to check one out of three check boxes and then click a submit button:



HTML content can be dynamically generated in response to a form which a user has filled in.

For example, a user could be asked to check one out of three check boxes and then click a submit button:



After submitting the button the user could then be presented with:

Hello! Your choice was yellow

Implementing this example

Two HTML pages need to be created for this example:

- An HTML page which contains the web form (i.e., the radio buttons and the submit button).
- A dynamically generated HTML page, which uses the colour which the user had selected.

The second page is generated by a script. The script stores the parameters from the web form as variables, which can then be used within the HTML:

```
<p>Hello! Your choice was
<font color=$formparameter>$formparameter</font><p>
```

Format of the first line of an HTTP request:

```
METHOD space Request-URI space HTTP-Version \n\n
```

Methods are GET, HEAD, OPTIONS, POST, DELETE, etc.

The Request-URI is usually the path and name of the document (eg. “/” or “/somedir/index.html”).

The HTTP-Version is currently either 1.0 or 1.1.

A typical HTTP request with GET

The HTTP request is normally generated by the user's browser. A typical request with "GET" is

```
GET /cgi-bin/somefile.cgi HTTP/1.0
CONNECTION: Keep-Alive
USER-AGENT: Mozilla/4.0 (compatible; MSIE 5.22)
PRAGMA: no-cache
HOST: www.dcs.napier.ac.uk
ACCEPT: image/gif, image/x-xbitmap, image/jpeg, */*
QUERY_STRING: name=Mary&comments=Hello
```

A typical HTTP request with POST

```
POST /cgi-bin/somefile.cgi HTTP/1.0
REFERER: http://napier.ac.uk/cgi-bin/someform.html
CONNECTION: Keep-Alive
USER-AGENT: Mozilla/4.0 (compatible; MSIE 5.22)
HOST: www.dcs.napier.ac.uk
ACCEPT: image/gif, image/x-xbitmap, image/jpeg, */*
CONTENT-TYPE: application/x-www-form-urlencoded
CONTENT-LENGTH: 42
name=Mary&comments=Hello
```

A typical answer from a webserver

```
HTTP/1.1 200 OK
Date: Thu, 18 Nov 2004 15:41:04 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Accept-Ranges: bytes
X-Powered-By: Open SoCks 1.2.0000
Last modified: Thu, 18 Nov 2004 15:41:04 GMT
Connection: close
Content-Type: text/html; charset=ISO-8859-1
<!DOCTYPE HTML PUBLIC ‘‘-//W3C//DTD HTML 4.01 Transitional/
‘‘http://www.w3.org/TR/html4/loose.dtd’’>
<html>

...
```

Environment Variables

- Environment variables are a means for server-side web languages to exchange information between the server and the client.
- They control the information which is disclosed by either server or client or which refers to a specific HTTP request.
- Not all environment variables are always available.

Client-Side Information

Environment variables which contain client-side information:

HTTP_USER_AGENT	type and version of the client's browser client
HTTP_ACCEPT	accepted MIME types
REMOTE_ADDR	IP address of the client

Sever-Side Information

Environment variables which contain server-side information:

SERVER_SOFTWARE	software used for webserver (e.g. Apache)
SERVER_NAME	name of server
SERVER_PROTOCOL	protocol used by server (e.g. HTTP/1.1)

Request-Specific Information

Environment variables which contain information about a specific HTTP request:

SCRIPT_NAME	the URL of the script
REQUEST_METHOD	usually either GET or POST
QUERY_STRING	form parameters as part of URL (only for GET)
CONTENT_LENGTH	(only for POST)
HTTP_COOKIE	content of a cookie
REMOTE_USER	username - if authentication is used
HTTP_REFERER	URL of previous page

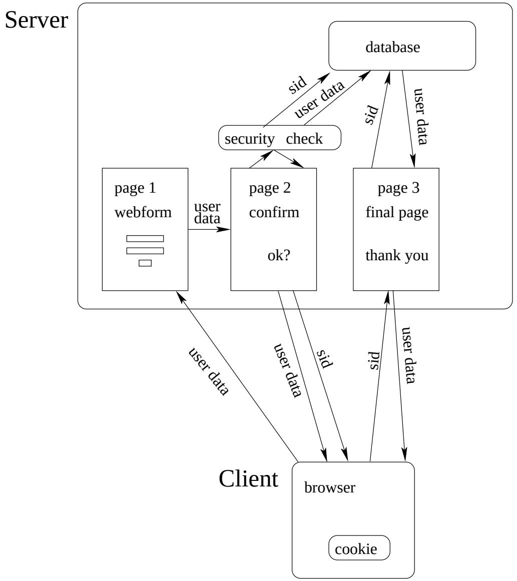
Can environment variables be trusted?

Can environment variables be trusted?

- The value of environment variables cannot be trusted.
- For example: clients can lie about which type of browser they use.
- HTTP_REFERER should never be used for security because it can be modified by a client.
- For highly secure applications, the content of cookies should be encrypted because otherwise it can be modified by a client.

A challenge for server-side web languages

Session management poses a challenge for server-side web languages because, in contrast to stand-alone applications, server-side applications transfer data between servers and clients. A server-side application does not have full control over the data at the client's computer. A client can delete, modify or add to any data. Therefore client data cannot be trusted.


Server
database
sid
user data
sid
user data
security check
page 1
webform
user
data
page 2
confirm
ok?
page 3
final page
thank you
user data
user data
sid
sid
user data
Client
browser
cookie

Using session IDs

- Sessions are created to minimise the risk of data corruption.
- All user data is stored on the server and associated with a session ID. Data retrieval is based on the session ID.
- A security check must be performed on any data, which a user enters, before the data is stored on the server.
- It is not necessary to perform security checks on data that is retrieved from the server. This is in contrast to CGI applications without sessions: in that case a security check must be performed on user data every time the data is used!

Session IDs

Session IDs are passed from page to page either

- by using hidden HTML form text or
- by using the query string or
- by using cookies (on the client computer).

The session IDs should be generated by the server and should be long and complicated enough to deter users from manual tampering with the IDs.

Storing user data

The user data, which is associated with a session ID can be stored on the server

- in a file
- in a database.

This approach reduces the number of security checks that must be performed on user data.

Storing user data

The user data, which is associated with a session ID can be stored on the server

- in a file
- in a database.

This approach reduces the number of security checks that must be performed on user data.

Further measures can be taken for applications which require high security:

- the session can be password protected;
- the cookie data can be encrypted;
- the whole session can be encrypted using SSL.

Cookies

Cookies are transmitted between client and server via the HTTP_COOKIE environment variable.

This means that any cookie data needs to be **written** before the HTML header is sent.

Cookie data can be **read** at any time.

Workflow for cookies

- First: read any existing cookie.
- Second: compile cookie data and send it.
- Third: Print the HTML header.

Note:
it is counterintuitive to first read the cookie and then write it!