Web technologies

HTTP cookies and PHP session

David Jelenc

April 2017

Index

Index

States in HTTP

- HTTP protocol does not maintain state

States in HTTP

- HTTP protocol does not maintain state
- Server processes every HTTP request independent of any previous requests

States in HTTP

- HTTP protocol does not maintain state
- Server processes every HTTP request independent of any previous requests
- We can simulate the connection state with the use of HTTP cookies or PHP session

HTTP cookies

- An HTTP coookie is a *name*/*value* pair

HTTP cookies

- An HTTP coookie is a *name*/*value* pair
- It is stored on the **client**

HTTP cookies

- An HTTP coookie is a *name*/*value* pair
- It is stored on the **client**
- Once the server sets the cookie and sends it to the client, the client *knows* that it has to send the cookie to the server in all subsequent HTTP requests.

HTTP cookies

- An HTTP coookie is a *name*/*value* pair
- It is stored on the **client**
- Once the server sets the cookie and sends it to the client, the client *knows* that it has to send the cookie to the server in all subsequent HTTP requests.
- Cookies thus carry data from client to the server and allow the server to maintain the *connection state.*

How cookies work

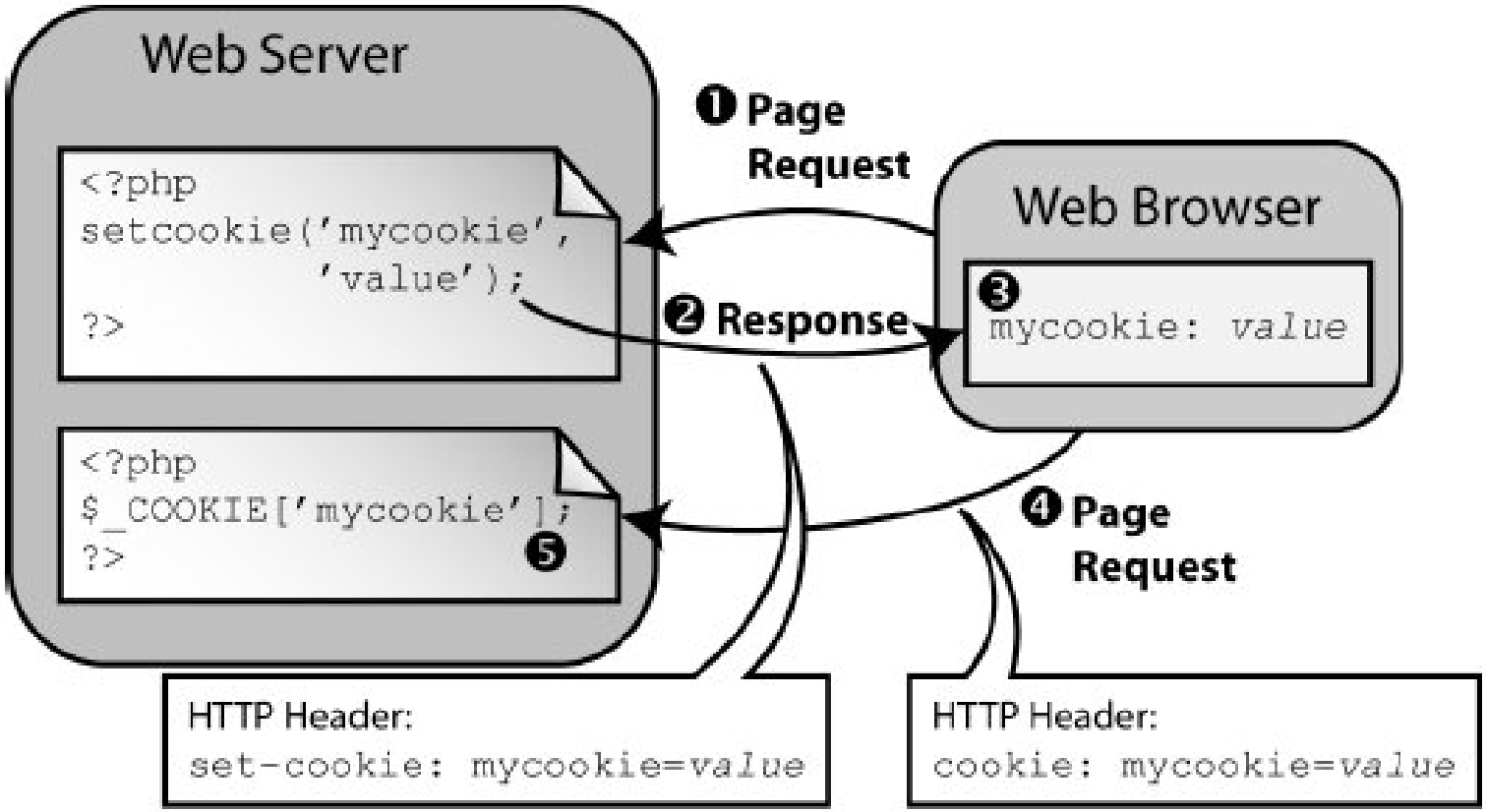

Figure: Using HTTP cookies in PHP

HTTP cookies

- Invoking `setcookie('visited', 1, time() + 5)` sets the cookie with name `visited` to value `1`. The cookie is valid for 5 seconds.

HTTP cookies

- Invoking `setcookie('visited', 1, time() + 5)` sets the cookie with name `visited` to value `1`. The cookie is valid for 5 seconds.
- **This means:** If the client is to send another HTTP request to the server within next 5 seconds, it *should* add a cookie to the request.

HTTP cookies

- Invoking `setcookie('visited', 1, time() + 5)` sets the cookie with name `visited` to value `1`. The cookie is valid for 5 seconds.
- **This means:** If the client is to send another HTTP request to the server within next 5 seconds, it *should* add a cookie to the request.
- Find out more: `http://php.net/setcookie`

PHP session

HTTP cookies have drawbacks

PHP session

HTTP cookies have drawbacks

- The client can disable/ignore them

PHP session

HTTP cookies have drawbacks

- The client can disable/ignore them
- The client can **change their names and/or values**

PHP session

HTTP cookies have drawbacks

- The client can disable/ignore them
- The client can **change their names and/or values**
- Cookies are not meant to store large amounts of data

PHP session

HTTP cookies have drawbacks

- The client can disable/ignore them
- The client can **change their names and/or values**
- Cookies are not meant to store large amounts of data
- HTML5 Local storage
  `http://diveintohtml5.info/storage.html`

PHP session

HTTP cookies have drawbacks

- The client can disable/ignore them
- The client can **change their names and/or values**
- Cookies are not meant to store large amounts of data
- HTML5 Local storage
  `http://diveintohtml5.info/storage.html`

It is often better to use **PHP session.**

PHP session

- With PHP session, the server assigns each client **a unique number PHPSESSID or SID**

PHP session

- With PHP session, the server assigns each client **a unique number PHPSESSID or SID**
- The data that is written to the session is stored on the server (not on the client)

PHP session

- With PHP session, the server assigns each client **a unique number PHPSESSID or SID**
- The data that is written to the session is stored on the server (not on the client)
- Once the SID has been sent to the client, the client *knows* that it has to send the SID to the server with each subsequent HTTP request. The client can send its SID either:

PHP session

- With PHP session, the server assigns each client **a unique number PHPSESSID or SID**
- The data that is written to the session is stored on the server (not on the client)
- Once the SID has been sent to the client, the client *knows* that it has to send the SID to the server with each subsequent HTTP request. The client can send its SID either:
  - with **a cookie**

PHP session

- With PHP session, the server assigns each client **a unique number PHPSESSID or SID**
- The data that is written to the session is stored on the server (not on the client)
- Once the SID has been sent to the client, the client *knows* that it has to send the SID to the server with each subsequent HTTP request. The client can send its SID either:
  - with **a cookie**
  - or with a **request parameter (GET or POST) with name SID**

PHP session

- With PHP session, the server assigns each client **a unique number PHPSESSID or SID**
- The data that is written to the session is stored on the server (not on the client)
- Once the SID has been sent to the client, the client *knows* that it has to send the SID to the server with each subsequent HTTP request. The client can send its SID either:
  - with **a cookie**
  - or with a **request parameter (GET or POST) with name SID**
- The server then associates each client with its session data based on presented SID.

PHP session: Usage

- Session is created and restored by invoking `session_start()`

PHP session: Usage

- Session is created and restored by invoking `session_start()`
- All session data is accessible via the superglobal associative array `$_SESSION`

PHP session: Usage

- Session is created and restored by invoking `session_start()`
- All session data is accessible via the superglobal associative array `$_SESSION`
- Elements of this field can be freely modified

PHP session: Usage

- Session is created and restored by invoking `session_start()`
- All session data is accessible via the superglobal associative array `$_SESSION`
- Elements of this field can be freely modified
- To delete a value stored under some `key`, simply call `unset($_SESSION["key"])`

PHP session: Usage

- Session is created and restored by invoking `session_start()`
- All session data is accessible via the superglobal associative array `$_SESSION`
- Elements of this field can be freely modified
- To delete a value stored under some `key`, simply call `unset($_SESSION["key"])`
- To destroy all data in the session, invoke `session_destroy()`

PHP session: Usage

- Session is created and restored by invoking `session_start()`
- All session data is accessible via the superglobal associative array `$_SESSION`
- Elements of this field can be freely modified
- To delete a value stored under some `key`, simply call `unset($_SESSION["key"])`
- To destroy all data in the session, invoke `session_destroy()`
- By default, the session expires after some period of inactivity or once the browser has been closed

PHP session: A word of warning

- The session identifier is a **security sensitive piece of information**

PHP session: A word of warning

- The session identifier is a **security sensitive piece of information**
- Log-ins systems are often implemented with the help of session. If an attacker finds out the SID, it can log-in into such system without knowing the victim's username or password.

PHP session: A word of warning

- The session identifier is a **security sensitive piece of information**
- Log-ins systems are often implemented with the help of session. If an attacker finds out the SID, it can log-in into such system without knowing the victim's username or password.
- Therefore we omit sending the data over request parameters (it is now disabled by default in many PHP/Apache set-ups)

PHP session: A word of warning

- The session identifier is a **security sensitive piece of information**
- Log-ins systems are often implemented with the help of session. If an attacker finds out the SID, it can log-in into such system without knowing the victim's username or password.
- Therefore we omit sending the data over request parameters (it is now disabled by default in many PHP/Apache set-ups)
- A typical attack: *session hijacking* with the help of a cross-side scripting attack (XSS)

PHP session: A word of warning

- The session identifier is a **security sensitive piece of information**
- Log-ins systems are often implemented with the help of session. If an attacker finds out the SID, it can log-in into such system without knowing the victim's username or password.
- Therefore we omit sending the data over request parameters (it is now disabled by default in many PHP/Apache set-ups)
- A typical attack: *session hijacking* with the help of a cross-side scripting attack (XSS)
  - JavaScript can be used to access the cookies

PHP session: A word of warning

- The session identifier is a **security sensitive piece of information**
- Log-ins systems are often implemented with the help of session. If an attacker finds out the SID, it can log-in into such system without knowing the victim's username or password.
- Therefore we omit sending the data over request parameters (it is now disabled by default in many PHP/Apache set-ups)
- A typical attack: *session hijacking* with the help of a cross-side scripting attack (XSS)
  - JavaScript can be used to access the cookies
  - This can be disabled, but not all browsers support this (`http://php.net/setcookie`, `httponly` flag)