中山大學
SUN YAT-SEN UNIVERSITY

# Lecture 23
# Cookies and Sessions

**SE-805 Web 2.0 Programming (supported by Google)**

http://my.ss.sysu.edu.cn/courses/web2.0/

**School of Software, Sun Yat-sen University**

# Outline

- **Cookies**
- Sessions

# Stateful Browser/Server Interaction

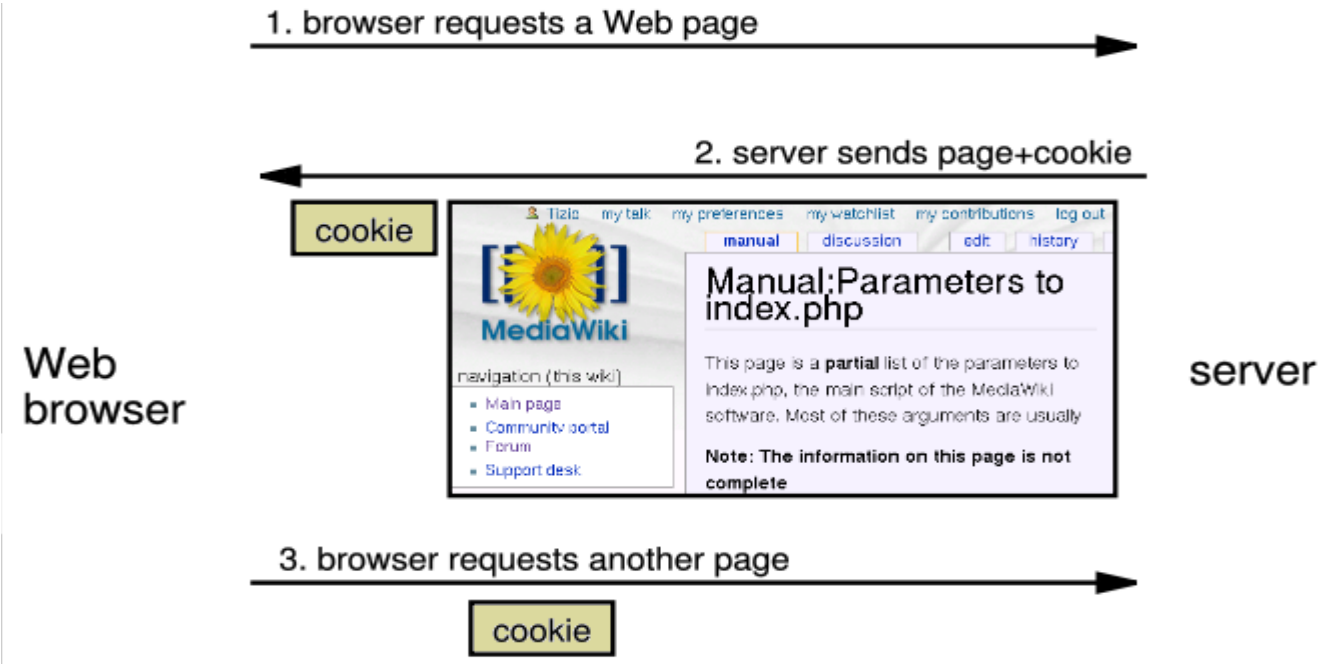
*Sites like amazon.com seem to "know who I am." How do they do this? How does a client uniquely identify itself to a server, and how does the server provide specific content to each client?*


- HTTP is a **stateless** protocol; it simply allows a browser to request a single document from a web server
- In these slides, we'll learn about pieces of data called **cookies** used to work around this problem, which are used as the basis of higher-level **sessions** between clients and servers

# What is a Cookie?

- cookie: a small amount of information sent by a server to a browser, and then sent back by the browser on future page requests
- Cookies have many uses:
  - Authentication
  - User tracking
  - Maintaining user preferences, shopping carts, etc.
- A cookie's data consists of a single name/value pair, sent in the header of the client's HTTP GET or POST request

# How Cookies are Sent




- When the browser requests a page, the server may send back a cookie(s) with it
- If your server has previously sent any cookies to the browser, the browser will send them back on subsequent requests
- Alternate model: client-side JS code can set/get cookies

# Myths about Cookies

- Myths:
  - Cookies are like worms/viruses and can erase data from the user's hard disk.
  - Cookies are a form of spyware and can steal your personal information.
  - Cookies generate popups and spam.
  - Cookies are only used for advertising.
- Facts:
  - Cookies are only data, not program code.
  - Cookies cannot erase or read information from the user's computer.
  - Cookies are usually anonymous (do not contain personal information).
  - Cookies CAN be used to track your viewing habits on a particular site.

# How Long does a Cookie Exist?

- **Session cookie** : the default type; a temporary cookie that is stored only in the browser's memory
  - When the browser is closed, temporary cookies will be erased
  - Can not be used for tracking long-term information
  - Safer, because no programs other than the browser can access them
- **Persistent cookie** : one that is stored in a file on the browser's computer
  - Can track long-term information
  - Potentially less secure, because users (or programs they run) can open cookie files, see/change the cookie values, etc.

# Where are the Cookies on My Computer

- **IE**: *HomeDirectory*\Cookies
  - e.g. C:\Documents and Settings\administrator\Cookies
  - Each is stored as a .txt file similar to the site's domain name
- **Firefox**: %APPDATA%\Mozilla\Firefox*???*.default\cookies.txt (cookies.sqlite)
  - View cookies in Firefox preferences: Privacy, Show Cookies...

# Cookie in JavaScript

```JS
document.cookie = "username=smith;password=12345";
```

- JS has a global document.cookie field (a string)
- You can manually set/get cookie data from this field (sep. by ;), and it will be saved in the browser
- You can't remove a cookie, but instead, you can make it expire, and why?

# Setting Cookie in PHP

```php
setcookie("name", "value");
```

```php
setcookie("username", "martay");
setcookie("favoritecolor", "blue");
```

- `setcookie` causes your script to send a cookie to the user's browser
- `setcookie` must be called before any output statements (HTML blocks, print, or echo)
- You can set multiple cookies (20-50) per user, each up to 3-4K bytes

# Retrieving Information from a Cookie

```php
$variable = $_COOKIE["name"];    # retrieve value of the cookie
```

```php
if (isset($_COOKIE["username"])) {
  $username = $_COOKIE["username"];
  print("Welcome back, $username.\n");
} else {
  print("Never heard of you.\n");
}
print("All cookies received:\n");
print_r($_COOKIE);
```

- Any cookies sent by client are stored in `$_COOKIES` associative array
- Use isset function to see whether a given cookie name exists

# Setting a Persistent Cookie in PHP

```
setcookie("name", "value", timeout);
```
PHP

```
$expireTime = time() + 60*60*24*7;    # 1 week from now
setcookie("CouponNumber", "389752", $expireTime);
setcookie("CouponValue", "100.00", $expireTime);
```
PHP

- To set a persistent cookie, pass a third parameter for its timeout in seconds
- time function returns the current time in seconds
  - date function can convert a time in seconds to a readable date

# Removing a Persistent Cookie

```php
setcookie("name", "", time() - 1);
```

```php
setcookie("CouponNumber", "", time() - 1);
```

- If the server wants to remove a persistent cookie, it should set it again, passing a timeout that is prior to the present time

# Outline

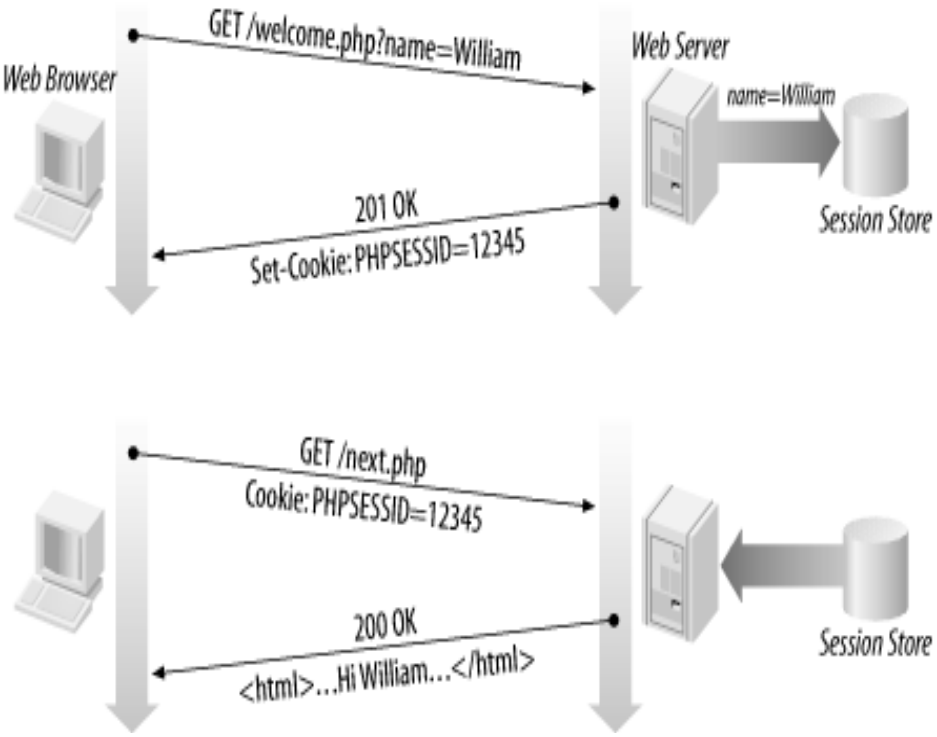
- Cookies
- **Sessions**

# What is a Session?

- **Session**: an abstract concept to represent a series of HTTP requests and responses between a specific Web browser and server
  - HTTP doesn't support the notion of a session, but PHP does
- Sessions vs. cookies:
  - A cookie is data stored on the client
  - A session's data is stored on the server (only 1 session per client)
- Sessions are often built on top of cookies:
  - The only data the client stores is a cookie holding a unique **session ID**
  - On each page request, the client sends its session ID cookie, and the server uses this to find and retrieve the client's session data

# How Sessions are Established

- Client's browser makes an initial request to the server
- Server notes client's IP address/browser, stores some local session data, and sends a session ID back to client
- Client sends that same session ID back to server on future requests




- Server uses session ID to retrieve the data for the client's session later, like a ticket given at a coat-check room

# Session in PHP: session_start

```php
session_start();
```

- **session_start** signifies your script wants a session with the user
  - Must be called at the top of your script, before any HTML output is produced
- When you call **session_start**:
  - If the server hasn't seen this user before, a new session is created
  - Otherwise, existing session data is loaded into **$_SESSION** associative array
  - You can store data in $_SESSION and retrieve it on future pages
- Complete list of PHP session functions

# Accessing Session Data

```
$_SESSION["name"] = value;          # store session data
$variable = $_SESSION["name"];      # read session data
if (isset($_SESSION["name"])) {   # check for session data
```
PHP

```
if (isset($_SESSION["points"])) {
  $points = $_SESSION["points"];
  print("You've earned $points points.\n");
} else {
  $_SESSION["points"] = 0;   # default
}
```
PHP

- The **`$_SESSION`** associative array reads/stores all session data
- Use `isset` function to see whether a given value is in the session

# Where is Session Data Stored?

- On the client, the session ID is stored as a cookie with the name **PHPSESSID**
- On the server, session data are stored as temporary files such as `/tmp/sess_fcc17f071...`


- You can find out (or change) the folder where session data is saved using the session_save_path function
- For very large applications, session data can be stored into a SQL database (or other destination) instead using the session_set_save_handler function

# Browsers Don't Support Cookies

```php
session_start();    # same as usual

# Generate a URL to link to one of our site's pages
# (you probably won't ever need to do this)
$orderUrl = "/order.php?PHPSESSID=" . session_id();
```

- If a client's browser doesn't support cookies, it can still send a session ID as a query string parameter named **PHPSESSID**
  - This is done automatically; `session_start` detects whether the browser supports cookies and chooses the right method
- If necessary (such as to build a URL for a link on the page), the server can find out the client's session ID by calling the session_id function

# Session Timeout

- Because HTTP is stateless, it is hard for the server to know when a user has finished a session
- Ideally, user explicitly logs out, but many users don't
- Client deletes session cookies when browser closes
- Server automatically cleans up old sessions after a period of time
  - Old session data consumes resources and may present a security risk
  - Adjustable in PHP server settings or with session_cache_expire function
  - You can explicitly delete a session by calling session_destroy

# Summary

- Cookies
  - Statelful BS interactions
  - How cookies are sent and stored
  - How long they exist
- Sessions
  - How sessions are established and stored
  - Session timeout
  - Using sessions in PHP

# Exercises

- Write a simple **user aware** to-do list application as a web page.
  - A  element wraps all html elements
  - A form for adding new to-do items
  - A list of all to-do items
  - Buttons of "select all", "deselect all", and "remove
  - When the "add" button is clicked the new to-do item will be inserted to the bottom of the list
  - Make the to-do list retain all items across web sessions
    - Using raw cookie, persisting all to-do items in a cookie, (be careful, the data should be encrypted)
    - Using session provided by PHP

# Further Readings

- Introduction of Cookie
  http://en.wikipedia.org/wiki/HTTP_cookie
- Introduction of Session
  http://en.wikipedia.org/wiki/Session_(computer_science)
- PHP Cookies
  http://php.net/manual/en/features.cookies.php
- W3Schools JavaScript Cookies
  http://www.w3schools.com/JS/js_cookies.asp
- PHP Sessions
  http://www.php.net/manual/en/book.session.php
- PHP Sessions tutorial
  http://www.tizag.com/phpT/phpsessions.php

# Thank you!