PHP Cookie and Session

Fajiang Yu, fjyu@whu.edu.cn

School of Computer, Wuhan University

2017.4

Agenda

Agenda

Introduction

HTTP is a stateless protocol, and the Internet is a stateless development environment.

cookie

- A cookie is simply a file, containing a series of variable-value pairs and linked to a domain.
- When a client requests a particular domain, the values in the cookie file are read and imported into the server environment, where a developer can read, modify and use them for different purposes.
- A cookie is a convenient way to carry forward data from one client visit to the next.

Introduction

session

- Another common approach is to use a session to store connection-specific data;
- This session data is preserved on the server for the duration of the visit, and is destroyed on its conclusion.
- Sessions work by associating every session with a session ID (a unique identifier for the session) that is automatically generated by PHP.
- This session ID is stored in two places: on the client using a temporary cookie, and on the server in a flat file or a database.
- By using the session ID to put a name to every request received, a developer can identify which client initiated which request, and track and maintain client-specific information in session variables (variable-value pairs which remain alive for the duration of the session and which can store textual or numeric information).

Agenda

the first session

```
<?php
   // initialize a session
   session_start();
   // increment a session counter
   $_SESSION['counter']++;
   // print value
   echo "You have viewed this page "
        . $_SESSION['counter'] . " times";
 ?>
```

the first session

- request the script above through your browser a few times.
- the counter increases by 1 on each subsequent page load.
- If you open up two browser windows and request the same page in each one, PHP will maintain and increment individual session counters for each browser instance.
- The session ID is used to identify which client made which request, and recreate the prior saved environment for each individual session.
- This also means that if you visit one (or more) other Web sites during the same session and then return to the script above without shutting down your browser in the interim, your previous session will be retrieved and recreated for you.

the first session

- Every session in PHP begins with a call to the `session_start()` function. This function checks to see whether a session already exists, and either restores it (if it does) or creates a new one (if it doesn't).
- Session variables can then be registered by adding keys and values to the special `$_SESSION` superglobal array, and can be accessed at any time during the session using standard array notation.
- The first time a session is created, this key will have the value 0.

session id

```
<?php
   // initialize a session
   session_start();
   // print session ID
   echo "I'm tracking you with session ID " . session_id();
 ?>
```

remember me

```
<?php
   // initialize a session
   session_start();
 ?>

<html>
<head></head>
<body>
<?php
   if (!isset($_SESSION['name'])
       && !isset($_POST['name'])) {
      // if no data, print the form
 ?>
      <form action="<?php echo $_SERVER['PHP_SELF']?>"
         method="post">
         <input type="text" name="name">
       <input type="submit" name="submit" value="Enter your name">
      </form>
```

remember me

```
<?php
   }
   else if (!isset($_SESSION['name']) && isset($_POST['name'])) {
     // if a session does not exist but the form has been submitted
     // check to see if the form has all required values
     // create a new session
     if (!empty($_POST['name'])) {
        $_SESSION['name'] = $_POST['name'];
        $_SESSION['start'] = time();
        echo "Welcome, " . $_POST['name'] . "
              . A new session has been activated for you.
              Click <a href=" . $_SERVER['PHP_SELF'] . ">here</a>
              to refresh the page.";
     }
     else {
        echo "ERROR: Please enter your name!";
     }
   }
```

remember me

```
    else if (isset($_SESSION['name'])) {
        // if a previous session exists
        // calculate elapsed time since session start and now
        echo "Welcome back, " . $_SESSION['name'] . "
              . This session was activated "
              . round((time() - $_SESSION['start']))
              . " second(s) ago.
             Click <a href=" . $_SERVER['PHP_SELF'] . ">here</a>
             to refresh the page.";
    }
 ?>

</body>
</html>
```

remember me

- It's important to note that the call to `session_start()` must appear first, before any output is generated by the script.
- This is because the PHP session handler internally uses in-memory cookies to store session data, and the cookie creation headers must be transmitted to the client browser before any output.

session destroy

- When the user shuts down the client browser and destroys the session, the `$_SESSION` array will be flushed of all session variables.
- You can also explicitly destroy a session – for example, when a user logs out – by calling the `session_destroy()` function

```
<?php
   // initialize a session
   session_start();
   // then destroy it
   session_destroy();
 ?>
```

super global

$_SESSION is a superglobal, so you can use it inside and outside functions without needing to declare it as global first.

```
<?php
   // initialize a session
   session_start();
   // this function checks the value of a session variable
   // and returns true or false
   function isAdmin() {
      if ($_SESSION['name'] == 'admin') {
         return true;
      }
      else {
         return false;
      }
   }
```

super global

```
  // set a value for $_SESSION['name']
  $_SESSION['name'] = "guessme";
  // call a function which uses a session variable
  // returns false here
  echo isAdmin()."<br />";

  // set a new value for $_SESSION['name']
  $_SESSION['name'] = "admin";
  // call a function which uses a session variable
  // returns true here
  echo isAdmin()."<br />";
?>
```

Agenda

Introduction

- Since cookies are used to record information about your activities on a particular domain, they can only be read by the domain that created them
- A single domain cannot set more than twenty cookies, and each cookie is limited to a maximum size of 4 KB

Introduction

A cookie usually possesses six attributes

- name: the name of the cookie
- value: the value of the cookie
- expires: the date and time at which the cookie expires
- path: the top-level directory on the domain from which cookie data can be accessed
- domain: the domain for which the cookie is valid
- secure: a Boolean flag indicating whether the cookie should be transmitted only over a secure HTTP connection

Introduction

- Since cookies are stored on the user's hard drive, the developer have very little control over them.
- If a user decides to turn off cookie support in his or her browser, your cookies will simply not be saved.
- Therefore, avoid writing code that depends heavily on cookies; and have a backup plan ready in case cookie data cannot be retrieved from the client.

meeting old friend

PHP offers a single function for cookie manipulation: `setcookie()`

```
<?php
   if (!isset($_COOKIE['visited'])) {
    // if a cookie does not exist
    // set it
    setcookie("visited", "1", mktime()+86400, "/")
       or die("Could not set cookie");
    echo "This is your first visit here today.";
   }
   else {
    // if a cookie already exists
    echo "Nice to see you again, old friend!";
   }
?>
```

form and cookie

```
<?php
   if (!isset($_POST['email'])) {
      // if form has not been submitted
      // display form
      // if cookie already exists,
      // pre-fill form field with cookie value
 ?>

<html>
<head></head>
<body>
<form action="<?php echo $_SERVER['PHP_SELF']?>" method="post">
   Enter your email address:
   <input type="text" name="email"
          value="<?php echo $_COOKIE['email']; ?>">
   <input type="submit" name="submit">
```

form and cookie

```
<?php
    // also calculate the time since the last submission
    if ($_COOKIE['lastsave']) {
        $days = round((time() - $_COOKIE['lastsave']));
        echo "<br /> $days second(s) since last submission";
    }
 ?>
</form>
</body>
</html>
```

form and cookie

```
<?php
   }
   else {
      // if form has been submitted, set cookies with form value
      // and timestamp both cookies expire after 30 days
      if (!empty($_POST['email'])) {
         setcookie("email", $_POST['email'],
                   mktime()+(86400*30), "/");
         setcookie("lastsave", time(), mktime()+(86400*30), "/");
         echo "Your email address has been recorded.";
      }
      else {
         echo "ERROR: Please enter your email address!";
      }
   }
 ?>
</body>
</html>
```

remove cookie

```
<?php
   // delete cookie
   setcookie("lastsave", NULL, mktime() - 3600, "/");
 ?>
</body>
</html>
```

Authentication and Access Control

a simple user authentication system

- where certain pages can only be viewed by users who successfully log in to the system
- Users who have not been authenticated with a valid password are denied access to these “special” pages

Authentication and Access Control

The list of valid usernames and passwords is stored in a MySQL database

```
<?php
if (isset($_POST['name']) || isset($_POST['pass'])) {
   // form submitted
   // check for required values
   if (empty($_POST['name'])) {
      die ("ERROR: Please enter username!");
   }
   if (empty($_POST['pass'])) {
      die ("ERROR: Please enter password!");
   }

   // set server access variables
   $host = "127.0.0.1";
   $user = "root";
   $pass = "john.2005";
   $db = "testdb";
```

Authentication and Access Control

```
// open connection
$connection = mysqli_connect($host, $user, $pass)
   or die ("Unable to connect!");
// select database
mysqli_select_db($connection, $db)
   or die ("Unable to select database!");
// create query
$query = "SELECT * FROM users WHERE name = '"
   . $_POST['name'] . "' AND pass = '" . $_POST['pass'] . "'";
// execute query
$result = mysqli_query($connection, $query)
   or die ("Error in query: $query. "
   . mysqli_error($connection));
// see if any rows were returned
if (mysqli_num_rows($result) == 1) {
   // if a row was returned
   // authentication was successful
   // create session and set cookie with username
```

Authentication and Access Control

```
        session_start();
        $_SESSION['auth'] = 1;
        setcookie("username", $_POST['name'], time()+(84600*30));
        echo "Access granted!";
        echo "<br /><a href='userauthtest.php'>userauth test</a>";
    }
    else {
        // no result
        // authentication failed
        echo "ERROR: Incorrect username or password!";
    }

    // free result set memory
    mysqli_free_result($result);
    // close connection
    mysqli_close($connection);
}
```

Authentication and Access Control

```
else {
   // no submission
   // display login form
 ?>
<html>
<head></head>

<body>
<center>
<form method="post"
      action="<?php echo $_SERVER['PHP_SELF']; ?>">
   Username <input type="text" name="name"
             value="<?php echo $_COOKIE['username']; ?>">
   <p />
   Password <input type="password" name="pass">
   <p />
   <input type="submit" name="submit" value="Log In">
</form>
```

Authentication and Access Control

```
</center>
</body>
</html>

<?php
}
 ?>
```

Authentication and Access Control

- A security check must also be carried out on each of the restricted pages.
- Without this check, any user could bypass the login screen and simply type in the exact URL to each page to view it.

```
<?php
 // start session
 session_start();
 if (!$_SESSION['auth'] == 1) {
    // check if authentication was performed
    // else die with error
    die ("ERROR: Unauthorized access!");
 }
 else {
 ?>
```

Authentication and Access Control

```
<html>
<head></head>
<body>
   This is a secure page.
   You can only see this if $_SESSION['auth'] = 1
</body>
</html>

<?php
 }
 ?>
```

Thank You!
Any Questions?