CS1116/CS5018

Web Development 2

Dr Derek Bridge

School of Computer Science & Information Technology

University College Cork

HTTP requests

A browser sends an HTTP request to a server…

- when the user clicks on a hyperlink
- when the user enters a URL into the Location box
- when the browser submits a form to the server

But a client-side script can also send a HTTP request

`XMLHttpRequest` objects

Summary of what your client-side script needs to do in order to send an HTTP request:

1. Create the request object
2. Register a function that will handle the response (assuming asynchronous response handling)
3. Specify the URL and the HTTP method (e.g. GET or POST)
4. Optionally, specify any special headers that are to be sent
5. Send the request

`XMLHttpRequest` example

1. Create the request object:
```
let request = new XMLHttpRequest();
```
2. Register a function that will handle the response (assuming asynchronous response handling):
```
request.addEventListener('readystatechange', handle_response, false);
```
3. Specify the URL and the HTTP method (e.g. GET or POST):
```
request.open("GET", url, true);
```
4. Optionally, specify any special headers that are to be sent, e.g.:
```
request.setRequestHeader("User-Agent", "XMLHttpRequest");
request.setRequestHeader("Accept-Language", "en");
```
5. Send the request:
```
request.send(null);
```

The response

- The same object that was used for the request is also used for the response (in our case `request`)
- You can use the following properties and methods:
  - `request.status`: the status code sent back by the server
    Q: What number are you hoping for?
  - `request.getResponseHeader("…")`: to access the specified response header
  - `request.getAllResponseHeaders()`: to access all response headers as an array
  - `request.responseText`: to access the body of the server's response as a string
  - `request.responseXML`: to access the body of the server's response as XML (inc. HTML)

Typical function to handle the response

```
function handle_response() {
    // Check that the response has fully arrived
    if ( request.readyState === 4 ) {
        // Check the request was successful
        if ( request.status === 200 ) {
            // do whatever you want to do with
            // request.responseText or request.responseXML
        }
    }
}
```

Ajax

Ajax...

- ...stands for *Asychronous JavaScript and XML*
- It's **not** a programming language
- It's a way of using JavaScript (especially `XMLHttpRequests`), the DOM, HTML and CSS

What's the idea...

- Classically, if the content of a page changes, the whole page is fetched again from the server
- Think of all the activities involved — they will take considerable time
- The idea in Ajax is that client-side JavaScript will:
  - send an asynchronous `XMLHttpRequest` to fetch from the server just the content that has changed
  - use the response to update relevant parts of the page
- Smaller amounts of data are fetched — this takes less time

Uses of Ajax

- Auto-suggestions in Google search box
- Status updates in Twitter
- Ajax is usually behind so-called **single-page apps**
  - A single HTML web page uses Ajax to rewrite the current page, rather than loading new pages

Ajax example 1

- Suppose we modify the asteroids program to keep a score:

  ```
  let score = 0;
  ```

  which goes up by 1 for every 33ms that the player survives:

  ```
  function draw() {
      score = score + 1;
      …
  }
  ```

- When the game is over, we want to send the score to the server where a program called `store_score.py` will put the score into a database:

  ```
  let request;

  function stop() {
      clearInterval(interval_id);
      window.removeEventListener('keydown', activate);
      window.removeEventListener('keyup', deactivate);
      let url = 'store_score.py?score=' + score;
      request = new XMLHttpRequest();
      request.addEventListener('readystatechange', handle_response, false);
      request.open('GET', url, true);
      request.send(null);
  }

  function handle_response() {
      // Check that the response has fully arrived
      if ( request.readyState === 4 ) {
          // Check the request was successful
          if ( request.status === 200 ) {
              if ( request.responseText.trim() === 'success' ) {
                  // score was successfully stored in database
              } else  {
                  // score was not successfully stored in database
              }
          }
      }
  }
  ```

store_score.py

```
#!/usr/local/bin/python3

from cgitb import enable
enable()

from cgi import FieldStorage
from html import escape
import pymysql as db

print('Content-Type: text/plain')
print()

form_data = FieldStorage()
score = escape(form_data.getfirst('score', '').strip())
try:
    connection = db.connect('localhost', 'userid', 'password', 'database_name')
    cursor = connection.cursor(db.cursors.DictCursor)
    cursor.execute("""INSERT …""", (score))
    connection.commit()
    print('success')
    cursor.close()
    connection.close()
except db.Error:
    print('problem')
```

Ajax example 2

- We'll modify `register.py` from the lecture on user authentication
- Alongside the *Name* text field, we'll add a hyperlink: *Check name is available*
- If this link is clicked, some client-side JavaScript (in `check_name_available.js`) will
  - send a `XMLHttpRequest` to the server, asking it to run a Python program (`check_name_available.py`)
  - the response will be just a short string
  - when the response is received, the JavaScript will change the text alongside the *Name* text field to either *Name available* or *Name not available*
- Note how we are not reloading the whole page; we fetch only a small amount of data and update a small part of the page

`register.py` (modified)

Identical except for:

```
print("""
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="utf-8" />
            <title>Web Dev 2</title>
            <script src="check_name_available.js" type="module"></script>
        </head>
        <body>
            <form action="register.py" method="post">
                <label for="username">User name: </label>
                <input type="text" name="username" id="username" value="%s" />
                <span id="checker"></span>
                <label for="password1">Password: </label>
                <input type="password" name="password1" id="password1" />
                <label for="passwords2">Re-enter password: </label>
                <input type="password" name="password2" id="password2" />
                <input type="submit" value="Register" />
            </form>
            %s
        </body>
    </html>""" % (username, result))
```

The Python program: `check_name_available.py`

```
#!/usr/local/bin/python3

from cgitb import enable
enable()

from cgi import FieldStorage
from html import escape
import pymysql as db

print('Content-Type: text/plain')
print()

form_data = FieldStorage()
username = escape(form_data.getfirst('username', '').strip())
try:
    connection = db.connect('localhost', 'userid', 'password', 'database_name')
    cursor = connection.cursor(db.cursors.DictCursor)
    cursor.execute("""SELECT * FROM users
                      WHERE username = %s""", (username))
    if cursor.rowcount > 0:
        print('in_use')
    else:
        print('available')
    cursor.close()
    connection.close()
except db.Error:
    print('problem')
```

The client-side JavaScript: `check_name_available.js`

```javascript
let username;
let checker;
let request;

document.addEventListener('DOMContentLoaded', init, false);

function init() {
    username = document.querySelector('#username');
    checker = document.querySelector('#checker');
    username.addEventListener('keypress', set_link, false);
    checker.addEventListener('click', check_name_available, false);
}

function set_link() {
    checker.innerHTML = '<a href="#">Check name is available</a>';
}

function check_name_available() {
    let url = 'check_name_available.py?username=' + username.value;
    request = new XMLHttpRequest();
    request.addEventListener('readystatechange', handle_response, false);
    request.open('GET', url, true);
    request.send(null);
}

function handle_response() {
    // Check that the response has fully arrived
    if ( request.readyState === 4 ) {
        // Check the request was successful
        if ( request.status === 200 ) {
            if ( request.responseText.trim() === 'available' ) {
                checker.innerHTML = 'Name available';
                } else if ( request.responseText.trim() === 'in_use' ) {
                checker.innerHTML = 'Name not available';
            }
        }
    }
}
```

All that glitters is not gold: problems with Ajax

- The user typically does not know when requests are being made
- When the content changes, the URL doesn't: you cannot easily bookmark different 'versions' of the page
- Similarly, the Back button and History list of your browser may not function as some users would expect them to
- By fetching small amounts of content, Ajax is supposed to be faster
  Q: But why might it, in fact, slow things down?