COMSM0085

Overview of Software Tools

Software Tools: Part 2

(COMS10012 / COMSM0085)

This week (Week 6):

- HTTP
- HTML

Coming up:

- CSS (Week 7)
- JS (Week 8)

HTTP

HyperText Transfer Protocol

Immediate questions:

- What is ‘hypertext’?
- What is a ‘transfer protocol’?

HTTP: The Protocol

Developed by Tim Berners-Lee at CERN, 1989.

Published as a set of *RFC* specifications.

Your exercise this week includes reading parts of RFC 7230!

Protocols

A protocol is a plan for how cooperating components of a system should interact.

Simple example:

Client: Give me block #200

Server: Here you go: *0A 2F EE* …

Protocols: Status Communications

Responses are not just the data requested. For example:

Client: Give me block #45000

Server: Sorry, I couldn’t find that block.

or

Client: Gimme block #200

Server: Sorry, I don’t know what you’re asking.

or

Client: Give me block #200

Server: I think you meant block #201, which is *0A 2F EE* …

HTTP Status Communications

HTTP transfers hypertext (of course). This is the *data* the protocol is concerned with. Like the previous toy example, HTTP is also a client-server protocol with request-response semantics.

But HTTP also transfers *metadata* about the status of communicating parties. There are two key mechanisms for this:

- request and response *headers*
- status codes

Important to understand: the metadata is an important part of the HTTP *protocol*. However, *metadata* is separate from the *data*. A hypertext document does not *have to* arrive via HTTP.

HTTP message structure

From the RFC:

```
HTTP-message   = start-line
                 *( header-field CRLF )
                 CRLF
                 [ message-body ]
```

A `start-line` can be a `request-line` (for a request) or a `status-line` (for a response).

Headers are optional, and internally are `field-name : field-value` .

HTTP Requests

Per RFC 7230, the format for a HTTP request-line is:

```
method SP request-target SP HTTP-version CRLF
```

- `SP` = space
- `CRLF` = carriage return

The `HTTP-version` rarely changes (though it can!).

Key elements to understand are `method` and `request-target` .

HTTP Methods

- `GET` : retrieve a copy of the target resource
- `POST` : submit payload data to a target resource
- `HEAD` : retrieve metadata for corresponding GET
- `PUT` : replace target resource with payload
- `DELETE` : delete the target resource

In practice, many servers do not implement or will ignore `DELETE` or `PUT` requests in favour of custom semantics using `POST` requests.

HTTP request-target

Simply, the resource you are targeting with your request.

Relates to the `path` and `query` components of a URI (Uniform Resource Identifier).

- `path` : e.g., `/` , or `/files/index.html` or `/user/george/`
- `query` : e.g., `?name=welcome&action=view`
  - formed of a series of *parameters* (`name` , `action`) with values (`welcome` , `view`)

The same resource at a `path` might respond differently to different `query` strings.

HTTP Status Line

Now for the *response*. The format of a status line is:

` status-line = HTTP-version SP status-code SP reason-phrase CRLF `

- `HTTP-version` we covered this already in the request
- `status code` 3-digit code with specific meaning
- `reason-phrase` description to explain status code

HTTP Status Codes

Can have very specific meanings, but are grouped by first digit with semantic meaning:

- `1xx` information (e.g., `100 Continue`).
- `2xx` success (e.g., `200 OK`)
- `3xx` redirect (e.g., `301 Moved Permanently`)
- `4xx` client error (e.g., `403 Forbidden`)
- `5xx` server error (e.g., `500 Internal Server Error`)

Example HTTP Exchange

Request

```
GET /index.html HTTP/1.1
Host: www.bristol.ac.uk
Connection: close
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 1009
```

```
<!DOCTYPE html>
<html lang="en">
...
```

Content-Type

An important *header* in the response is the `Content-Type`.

This is still *metadata*. Tells the client *what type of data* the response body will contain.

Very important for *browsers*, as clients that interpret response bodies for humans.

If we changed `Content-Type` of our response from `text/html` to `text/plain`, what would happen?

HTML

Finally, hypertext. The concept is *interactive text*. At its core, think 'text with hyperlinks'.

HTML is HyperText Markup Language. A language for 'marking up' text to make it interactive (and structured).

A HTML document reader (like a browser) has to interpret the markup and present the result to the user.

Key elements

- **Tags** indicate meaningful document components.
- Nesting of tags and text within tags organises document structure.
- Tags can have **attributes** that affect interpretation of their semantics.

Important HTML tags

Everything is nested within `<html>` .

`<head>` vs `<body>`

Note: this is within the HTML document itself! `<head>` has no relation to HTTP headers.

- `<head>` contains `<title>` and `<meta>` tags, to describe `<body>`
- `<body>` contains the 'visible' portion of the document.
  - Most document components should be placed within the body.
  - Common contents include `<p>` , `<div>` , `<span>` ...

HTML Document Example

```
<!DOCTYPE html>
<html lang="en">
 <head>
```

```
    <meta charset="utf-8" />
    <title>A web page</title>
  </head>
  <body>
     <h1>An example web-page</h1>
     <p>A paragraph of text here, perhaps with <a href='./another_page/'>a link</
  </body>
</html>
```

HTML Presentation

There are common visual defaults for GUI browsers (e.g., blue underline).

But the semantic structure of a document is (meant to be) *separate* from its presentation.

- Consider how a browser should read a HTML document to a blind user.

In a browser, the presentation of elements in HTML documents is governed by *stylesheets*.

Next week, we'll be discussing how style is applied via *Cascading Style Sheets* (CSS).

Exercises this week

1. Carrying out simple HTTP client-server interactions
2. Studying how URIs are constructed
3. Launching 'real' web servers to serve files
4. Writing a simple HTML document
5. Using a templating engine to have a server *generate* HTML documents

See you on Friday.