Lecture 3: The HTTP Protocol and HTML Language

(02-05-2020)

(Reading: Lecture Notes)

Lecture Outline

1. Introduction
2. The HTTP protocol
3. The HTML language
4. Web client/server interactions
5. CGI interfaces and format of server replies

1. Introduction

(1) HTTP (hypertext transfer protocol) is an application layer protocol for distributed, collaborative, hypermedia information systems

(2) History

a. First version of HTTP: HTTP/0.9, 1990. Primitive and experimental

b. HTTP/1.0, 1996. Much improved. But failed to consider the effects of hierarchical caching, proxies, the need of consistent connections (virtual hosts).

c. HTTP/1.1, 1999.

2. The HTTP protocol

(1) Official HTTP 1.1 site and specification:

a. Official HTTP URI:

http://www.w3.org/Protocols

b. Official HTTP 1.1 specification (in BNF):

http://www.w3.org/Protocols/rfc2616/rfc2616.txt

The section numbers referred in this lecture are those in the above official spec.

c. There is also an HTML version of rfc2616:

http://www.w3.org/Protocols/rfc2616/rfc2616.html

(2) HTTP URI (universal resource identifier), URL (universal resource locator), and URN (universal resource name)

a. Syntax of http URL:

```
http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

b. Semantics

(3) HTTP messages

a. Syntax

```
HTTP-message = Request | Response ; HTTP/1.1 messages
```

b. Both types of messages use the generic message format of RFC 822 for transferring entities (the payload of the message). Both types of message consist of a start-line, zero or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and possibly a message-body.

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
start-line      = Request-Line | Status-Line
```

c. Message headers

(a) HTTP message headers contain control information that is being used between HTTP servers and clients (browsers).

(b) Types of headers: *general header, request header, response header*, and *entity header*.

(c) A message may contain all or some of the four types of headers. The order of message headers is not important (although it is a good practice to have general header first and entity header last.

(d) Format of message headers: Each header field consists of a name followed by a colon (":") and the field value. Field names are case-insensitive.

```
message-header = field-name ":" [ field-value ]
field-name     = token
field-value    = *( field-content | LWS )
field-content  = <the OCTETs making up the field-value
                 and consisting of either *TEXT or combinations
                 of token, separators, and quoted-string>
```

(e) Syntax of general-header

```
general-header = Cache-Control            ; Section 14.9
               | Connection               ; Section 14.10
               | Date                     ; Section 14.18
               | Pragma                   ; Section 14.32
```

```
                    | Trailer                 ; Section 14.40
                    | Transfer-Encoding       ; Section 14.41
                    | Upgrade                 ; Section 14.42
                    | Via                     ; Section 14.45
                    | Warning                 ; Section 14.46
```

d. Message body

(a) The message-body (if any) of an HTTP message is used to carry the entity-body associated with the request or response. The message-body differs from the entity-body only when a transfer-coding has been applied, as indicated by the Transfer-Encoding header field.

(b) Syntax:

```
message-body = entity-body
             | <entity-body encoded as per Transfer-Encoding>
```

(4) Request messages

a. A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use. Request messages have the following format:

method	*URL*	*HTTP version*	*headers*	*message body*
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/1.1		

Figure 4.15 HTTP *request* message

b. Syntax of request messages:

```
Request       = Request-Line              ; Section 5.1
                *(( general-header        ; Section 4.5
                 | request-header         ; Section 5.3
                 | entity-header ) CRLF)  ; Section 7.1
                CRLF
                [ message-body ]          ; Section 4.
```

c. Request-line.

(a) The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

(b) Syntax:

```
Request-Line   = Method SP Request-URI SP HTTP-Version CRLF
```

d. Request headers

(a) The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation.

(b) Syntax:

```
request-header = Accept                   ; Section 14.1
               | Accept-Charset           ; Section 14.2
               | Accept-Encoding          ; Section 14.3
               | Accept-Language          ; Section 14.4
               | Authorization            ; Section 14.8
               | Expect                   ; Section 14.20
               | From                     ; Section 14.22
               | Host                     ; Section 14.23
               | If-Match                 ; Section 14.24
               | If-Modified-Since        ; Section 14.25
               | If-None-Match            ; Section 14.26
               | If-Range                 ; Section 14.27
               | If-Unmodified-Since      ; Section 14.28
               | Max-Forwards             ; Section 14.31
               | Proxy-Authorization      ; Section 14.34
               | Range                    ; Section 14.35
               | Referer                  ; Section 14.36
               | TE                       ; Section 14.39
               | User-Agent               ; Section 14.43
```

(5) HTTP methods

a. The Method token (in the request-line) indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.

b. Syntax:

```
Method         = "OPTIONS"                ; Section 9.2
               | "GET"                    ; Section 9.3
               | "HEAD"                   ; Section 9.4
               | "POST"                   ; Section 9.5
               | "PUT"                    ; Section 9.6
               | "DELETE"                 ; Section 9.7
               | "TRACE"                  ; Section 9.8
               | "CONNECT"                ; Section 9.9
               | extension-method
extension-method = token
```

(a) *OPTIONS*: it represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.
Responses to this method are not cacheable.

(b) *GET*: it means retrieval of whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

- Conditional GET: if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A conditional GET method requests that the entity be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.
- Partial GET: if the request message includes a Range header field. A partial GET requests that only part of the entity be transferred, The partial GET method is intended to reduce unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

The response to a GET request is cacheable if and only if it meets the requirements for HTTP caching.

(c) *HEAD*: Similar to GET. But it will not retrieve message body. The metainformation contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request.
The response to a HEAD request may be cacheable in the sense that the information contained in the response may be used to update a previously cached entity from that resource. It can be used by a proxy server or cache server to check freshness of a resource.

(d) *POST*: it requests that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

i. Annotation of existing resources;
ii. Posting a message to a discussion group or message board;
iii. Submitting an online application, an online electronic payment, or an online order request to a data-processing process;

iv. Extending a database through an append operation.

Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource

(e) *PUT*: specifies that the data supplied in the request should be stored with the given URI as identifier. If the Request-URI refers to an already existing resource, the enclosed entity should be considered as a modified version of the one residing on the origin server
Responses to this method are not cacheable.

(f) *DELETE*: requests the remote Web server to delete the resource identified by the specified URL. Potentially insecure and disabled on many Web server. Responses to this method are not cacheable.

(g) *TRACE*: the server will echo the request sent by a client. For debugging purpose. It is used to invoke a remote, application-layer loop-back of the request message. The final recipient of the request should reflect the message received back to the client as the entity-body of a 200 (OK) response.

(h) *CONNECT*: it is reserved for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunneling).

(6) Response messages

a. After receiving and interpreting a request message, a server responds with an HTTP response message. Response messages are of the format:

HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		

Figure 4.16 HTTP *reply* message

b. Syntax of response messages:

```
Response      = Status-Line               ; Section 6.1
                *(( general-header        ; Section 4.5
                 | response-header         ; Section 6.2
                 | entity-header ) CRLF)   ; Section 7.1
                CRLF
                [ message-body ]          ; Section 7.2
```

c. Status-line. The first line of a Response message is the Status-Line. Syntacticall it consists of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence. The numeric status code informs the client how well the request went.

(a) Code values in the range 200-299 indicate normal successful replies;

(b) Code values in the range 300-399 indicate that abnormal but non-severe problems occurred;

(c) Code values in the range 400-599 indicate that various severe abnormal problems occurred.

d. Reponse header fields:

```
response-header = Accept-Ranges           ; Section 14.5
                | Age                     ; Section 14.6
                | ETag                    ; Section 14.19
                | Location                ; Section 14.30
                | Proxy-Authenticate      ; Section 14.33
                | Retry-After             ; Section 14.37
                | Server                  ; Section 14.38
                | Vary                    ; Section 14.44
                | WWW-Authenticate        ; Section 14.47
```

3. The HTML language

(1) Official HTML site and specification

a. Official HTML URI:

```
http://www.w3.org/MarkUp
```

b. Official HTML 4.01 specification:

```
http://www.w3.org/TR/html401/
```

c. HTML tutorial:

```
http://www.w3.org/MarkUp/Guide
http://www.w3.org/MarkUp/Guide/Advanced.htm
http://www.w3schools.com/html/
```

The first two are formal instroduction and definitions of HTML. The third one includes more examples that are helpful to beginners.

(2) SGML and markup languages

a. SGML (standard generalized markup language) is a language for defining markup languages. HTML is one of the many markup languages

b. Each markup language defined in SGML is called an SGML application. Each application is characterized by:

(a) An SGML declaration that specifies the characters and delimiters that may appear in the application.

(b) A DTD (docoument type definition). This is the core of the application. It defines the syntax of markup constructs.

(c) A specification that describes the semantics to be ascribed to the markup. It also provides additional syntax restrictions that cannot be expressed with DTD.

(d) Document instances containing data (content) and markup. Each instance contains a reference to the DTD to be used to interpret it.

c. How to read HTML DTD

(a) DTD comments: comments in DTD are delimited by a pair of −− marks.

(b) Parameter entity definitions:

i. A parameter entity definition defines a kind of macros that may be referenced and expanded elsewhere in the DTD. These macros can only appear in the DTD, not in the defined markup language.

ii. A parameter entity definition begins with the keyward <!ENTITY % followed by the entity name, the quoted string the entity expands to, and finally a closing >

iii. Example:

```
<!ENTITY % fontstyle "TT | T | B | BIG | SMALL">
```

(c) Element declarations

i. Element declarations are the bulk of a DTD application. Each element declaration begins with the keyword <!ELEMENT and ends with a closing >

ii. Between the keyword and the closing bracket the following are specified:

The element name;

Whether the element tags are optional;

The element's content.

Example:

```
<!ELEMENT % UL - - (LI)+>
```

(3) Basics

a. It is *tag*ged language.

(a) A pair of tags (also called *delimiter*) specifies an *element*.

(b) Element has name and optional attributes

b. Structure of an HTML document. An HTML (v.4) document consists of three sections:

(a) A line specifying HTML version info;

(b) A declarative header section (tagged with the *HEAD* tag. Use of HEAD tag is optional;

(c) A body, which contains the document's actual content. The body part may be implemented as a *BODY* element or as a FRAMESET element. Use of BODY tag is optional.

Section 2 and 3 should be enclosed in a pair of HTML tags

c. HTML comments: between the tags $<!--$ and $-->$:

```
<!-- This is a sample HTML document -->
<!-- This document demonstrates the frame element,
     together with scripts -->
```

d. Document character set specification

```
Content-type: text/html; charset=ASCII
```

e. Basic HTML data types

(a) Text strings

(b) URIs

(c) Colors

(d) Content types (MIME types)

(e) Character encoding

(f) Dates and times

(g) Linktypes

(4) Text and lists

a. Text

(a) Structured text: phrase elements EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ABBR, and ACRONYM

(b) Quotations

(c) Lines and paragraphs

· Paragraph: the P element

· Line breaks: the BR element

· Preformatted text: the PRE element

b. Lists

(a) Unordered lists: the UL element

(b) Ordered lists: the OL element

(c) List items: the LI element

(5) Tables

(6) Links

(7) Objects, images, and applets

(8) Frames

(9) Forms

(10) Scripts

4. Web client/server interactions

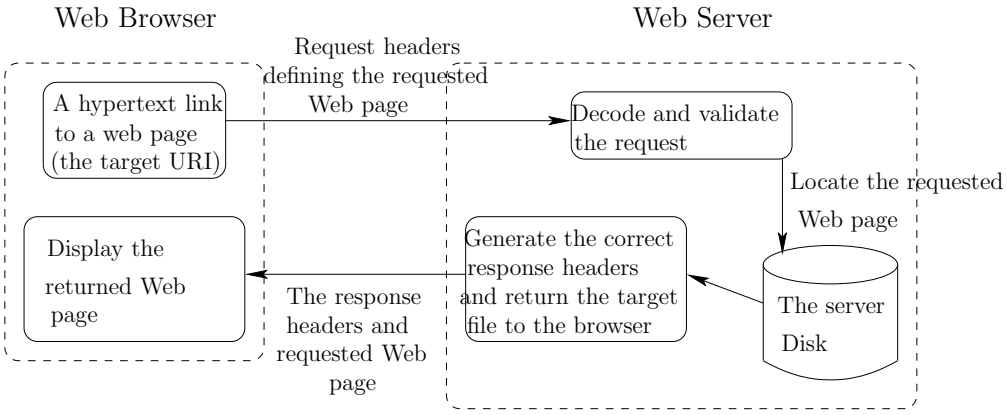

Figure 14: Basic Web client/server interactions

(1) Basic interactions: Fig.14

(2) Advanced interactions: Fig.15

(3) The decoding process

   a. Can be done through CGI scripts

   b. Can be done through other middle layer construct such as Java Servlet

(4) The process of generating reply pages

   a. Some reply pages are static and are fetched from files or databases directly.

   b. Other replies have to be constructed dynamically using CGI scripts, Servlets, or other server supporting programs.

5. CGI interfaces and format of server replies

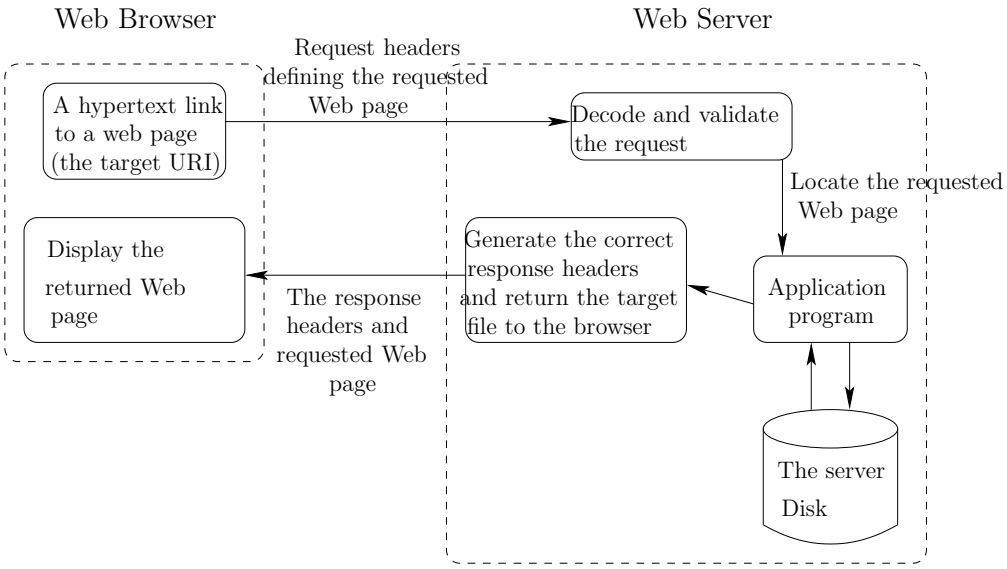

Figure 15: Advanced Web client/server interactions

(1) Server replies

a. The *Content-Type* header: indicates the type of data from server. Note: the Content-Type header must be separated from the next non-empty line by at least one empty-line.

b. If the *Content-Type* header contains value *text/html*, then what follows should be a valid HTML document that follows the HTML specification.

(2) CGI interfaces

a. CGI interfaces can be written in any appropriate languages (usually scripting languages) such as Perl, Javascript, or PHP.

b. An CGI interface script normally dynamically formats server replies. It accepts data returned from a information processing program and constructs reply HTML pages from the data.