Introduction to Web Engineering

SWE 642, Fall 2008
Nick Duan

What is web engineering?

- The concepts, tools and techniques for creating web-based applications
- Areas of Interests include
  - The Internet environment
    - Client/Server computing based on standard Internet protocols (TCP/IP and HTTP)
  - Client-side engineering
    - Web browser, application-specific web client (e.g. RSS Reader, Google Desktop, etc.)
  - Server-side engineering
    - Web servers, web application servers, web application components, web server APIs and extensions
The Internet Environment

- The Internet environment consists of
  - Basic infrastructure and infrastructure services
    - Telecom companies, ISPs
    - DNS (mapping domain names to IP addresses), domain name registry services
  - Standard protocols (ensure everyone speaks the same language)
    - TCP/IP and other network management standards (routing/switching, error checking, etc.)

The Web Environment

The WWW is considered as an application layer built on top of the Internet environment
Web-based Client/Server Computing

- Client/server computing using HTTP
- Message exchange for http://www.gmu.edu

http://www.gmu.edu

1. Look up for www.gmu.edu
2. Return IP of www.gmu.edu
3. Request for index.html page
4. Response with index.html page

HTTP Standard

- Define by RFC 2616; HTTP version 1.1
- Two message types: Request and Response
- Each web conversation starts with a web client sending a request message to a web server
- Statelessness: After the response message is received, the underlying TCP/IP connection will be closed
  - Enable a web server to serve a large number of client requests simultaneously
  - How to make an HTTP conversation stateful?
- A TCP/IP session: a single pair of HTTP request and response messages
HTTP Message Format

- HTTP message = Request | Response
- Generic message = start-line
  *(message-header CRLF)
  CRLF
  [message-body]
- Start-line = Request-Line | Status-Line
- message-header = field-name ":" [ field-value ]
- message-body = entity-body | <entity-body encoded as per Transfer-Encoding>

HTTP Request Message

- Request = Request-Line
  *( (general-header
  | request-header
  | entity-header ) CRLF ) ;
  CRLF
  [ message-body ]
- Request-Line = Method SP Request-URI SP
  HTTP-Version CRLF
- GET  http://www.w3c.org/index.html HTTP/1.1
HTTP Generic Headers

- general-header = Cache-Control
  - Connection
  - Date
  - Pragma
  - Trailer
  - Transfer-Encoding
  - …

HTTP Request Methods

- Method = "OPTIONS"
  - "GET"
  - "HEAD"
  - "POST"
  - "PUT"
  - "DELETE"
  - "TRACE"
  - "CONNECT"
  - extension-method

- extension-method = token
HTTP Request Headers

- request-header = Accept
  - Accept-Charset
  - Accept-Encoding
  - Accept-Language
  - Authorization
  - From
  - Host
  - If-Match
  - If-Modified-Since
  - If-None-Match
  - If-Range
  - If-Unmodified-Since
  - Max-Forwards
  - Proxy-Authorization
  - User-Agent
  - ...

HTTP Response Message

- Response = Status-Line
  - *(( general-header
    | response-header
    | entity-header ) CRLF)
  - CRLF
  - [ message-body ]

- Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
Status Code Categorization

- **1xx: Informational**
  - Request received, continuing process

- **2xx: Success**
  - The action was successfully received, understood, and accepted

- **3xx: Redirection**
  - Further action must be taken in order to complete the request

- **4xx: Client Error**
  - The request contains bad syntax or cannot be fulfilled

- **5xx: Server Error**
  - The server failed to fulfill an apparently valid request

Sample Status Code

- "100": Continue
- "200": OK
- "202": Accepted
- "300": Multiple Choices
- "305": Use Proxy
- "307": Temporary Redirect
- "400": Bad Request
- "401": Unauthorized
- "403": Forbidden
- "404": Not Found
- "405": Method Not Allowed
- "500": Internal Server Error
- "501": Not Implemented
- "503": Service Unavailable
- "504": Gateway Time-out
HTTP Response Headers
- response-header = Accept-Ranges
  - Age
  - ETag
  - Location
  - Proxy-Authenticate
  - Retry-After
  - Server
  - Vary
  - WWW-Authenticate

Entity Headers
- entity-header = Allow
  - Content-Encoding
  - Content-Language
  - Content-Length
  - Content-Location
  - Content-MD5
  - Content-Range
  - Content-Type
  - Expires
  - Last-Modified
  - extension-header
- extension-header = message-header
Sample Entity Header

- Content-type: text/html
  Set-Cookie: foo=bar; path=/; expires Mon, 09-Dec-2002 13:46:00 GMT
- Content-type: text/html
  Cookie: foo=bar
- Which cookie (Set-Cookie, Cookie) header is used by server, and which is used by client?

Message Example

- Request message sent via Telnet
  ise ~% telnet www.apache.org 80
  Trying 140.211.11.130...
  Connected to www.apache.org.
  Escape character is '^[].
  GET / index.html HTTP 1.1
Response Message

HTTP/1.1 200 OK
Date: Sun, 27 Jan 2008 04:46:56 GMT
Server: Apache/2.2.8 (Unix)
ETag: "7ad47b-42e9-4446c07de2b80"
Accept-Ranges: bytes
Content-Length: 17129
Cache-Control: max-age=86400
Expires: Mon, 28 Jan 2008 04:46:56 GMT
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

…
</html>
Connection closed by foreign host.

Web Client/Server Computing

Client Environment

- Browser Techniques
  - HTML, DOM, CSS, JavaScript, VBScript, etc
- ActiveX controls
- Plugins
- Java Applets
- Browser API Libs

Server Environment

- CGI/FastCGI
- Java Web Component
- ColdFusion Apps
- HTTP Server
- Server APIs & Extensions
- Media server
- Backend Apps

Client Platforms and Server Platforms
There was CGI in the beginning

- CGI is designed to support dynamic HTML forms by HTTP server via OS commands/scripts
- Two essential CGI environmental variables are mapped to HTML Form parameters
  - REQUEST_METHOD (method)
  - QUERY_STRING (input parameters)
- Some CGI variables are defined directly as HTTP headers, e.g.
  - CONTENT_LENGTH
  - CONTENT_TYPE
  - REMOTE_HOST
  - HTTP_USER_AGENT

Summary

- HTTP, as the standard Web protocol, is based on TCP/IP and stateless by nature
- Only two types of HTTP messages: Request and Response
- Each HTTP message starts with a Start-line or Status-code line, followed by one or more message headers, followed by a blank line, and then the message body
- Types of headers can be extended to satisfy user-specific or application specific needs
- HTML Form was designed originally to support CGI programming
Quiz

- Which message header specifies the length of the message body (in bytes) in the message?
- What would be the response message look like if the server encounters an exception?
- What would be the response message look like if the server redirect the request to another URL?
- How would you design a customized message header (entity header) to support passing user parameters back and forth between client and the server?
- How do fields in an HTML Form map to an HTTP message?