HTTP Primer

Protocol

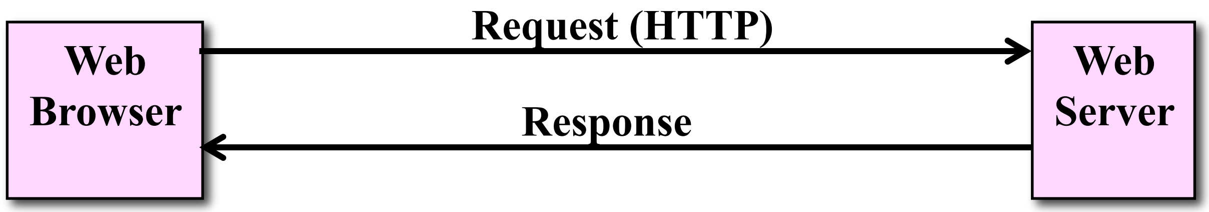
Request Headers

Response Headers

HTTP Basics

HTTP is the protocol of the web

- An *application layer* **protocol** typically built on TCP/IP
- **Synchronous** (what does this mean?)
- **Stateless** (what does this mean?)
- **Connection[-oriented]** (… stay tuned)



URLs: Uniform (Universal) Resource Locators

- Address of an *object* or *resource* (not limited to just a file)
  - `http://HOST[:port][abs_path [?query]]`
- Unsafe characters are encoded (; , ? : @ = &), and the space and tab characters too (space can be replaced by "+" also)
- Encoding takes the form of "%" plus the 2 digit hex value of the ASCII code for that character. (e.g. %20 is space)

HTTP Requests

HTTP supports various "methods"

- GET: Makes a request on a resource
- POST: Used to pass input to the server
    - GET encodes on the URL, POST puts it in the body of the message
    - POST handles binary payloads, makes them invisible, and can support input requests of an arbitrary length (w/ Content-Length)
- Others: OPTIONS, PUT, DELETE, CONNECT, TRACE, PATCH
    - We'll cover these later with REST

Example HTTP 1.1 Request

```
GET /search?keywords=servlets+jsp HTTP/1.1
Accept: image/gif, image/jpg, */*
Accept-Encoding: gzip, deflate, sdch
Connection: Keep-Alive
Cookie: userID=id456578
Host: www.somebookstore.com
Referer: http://www.somebookstore.com/findbooks.html
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11) AppleWebKit/
 537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36
```

Common HTTP 1.1 Request Headers

Accept

- Indicates MIME types browser can handle
- Can send different content to different clients. For example, PNG files have good compression characteristics but are not widely supported in browsers. A server could check to see if PNG is supported, sending <IMG SRC="picture.png" ...> if it is supported, and <IMG SRC="picture.gif" ...> if not.

Accept-Encoding

- Indicates encodings (e.g., gzip or compress) browser can handle

Authorization

- User identification for password-protected pages.
- We will spend a fair amount of time on this later.

Common HTTP 1.1 Request Headers

Connection

- In HTTP 1.0, keep-alive means browser can handle persistent connection. In HTTP 1.1, persistent connection is the default.
- Persistent connections mean that the server can reuse the same socket over again for requests very close together from the same client (e.g., the images associated with a page, or cells within a framed page).
  - Note that this is not the same thing as stateful

Cookie

- Gives cookies previously sent to client.

Host

- Indicates host given in original URL
- This is a required header in HTTP 1.1.

If-Modified-Since

- Client wants page only if it has been changed after specified date

Common HTTP 1.1 Request Headers

Referer

- URL of referring Web page
- Useful for tracking traffic; logged by many servers
- Can also be used to let users set preferences and then return to the page they came from
- Can be easily spoofed, so don't let this header be your sole means of deciding (for example) how much to pay sites that show your banner ads.

User-Agent

- String identifying the browser making the request
- Best used for identifying category of client
  - Web browser vs. cell phone, etc.
- For Web applications, use other headers if possible
- Again, can be easily spoofed.

HTTP Status Codes

Example HTTP 1.1 Response

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE ...>
<HTML>
...
</HTML>
```

Changing the status code lets you perform a number of tasks not otherwise possible

- Forward client to another page
- Indicate a missing resource
- Instruct browser to use cached copy

Set status before sending document

HTTP Status Code Categories

1xx	Informational	Rarely used
2xx	Successful	Most common is 200 for "OK"
3xx	Redirection	Server indicates that the client should automatically redirect to another URL
4xx	Client Error	Improper request (syntax or other reason). 404 means "Resource not found"
5xx	Server Error	A valid request was received but the server could not fulfill it.

These codes may not make sense now, but you will need them later as you will programatically set the values of several of these header fields.

Common HTTP 1.1 Status Codes

200 (OK)

- Everything is fine; document follows.

301 (Moved Permanently)

- Requested document permanently moved (indicated in Location header).
- Browsers go to new location automatically.

302 (Found) [Updated in 1.1 to use 303 & 307]

- Requested document temporarily moved elsewhere (indicated in Location header). Browsers go to new location automatically.

400 (Bad Syntax)

403 (Permission Denied)

404 (Not Found)

405 (Method Not Allowed)

501 (Not Implemented)

- Could be a future implementation

503 (Internal Server Error)

- A problem prevented the server from fulfilling the client’s request.

Common HTTP 1.1 Response Headers

Cache-Control (1.1) and Pragma (1.0)

- A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.

Content-Encoding

- The way document is encoded. Browser reverses this encoding before handling document.

Content-Length

- The number of bytes in the response.

Set-Cookie

- The cookies that browser should remember.

Common HTTP 1.1 Response Headers

Content-Type

- The MIME type of the document being returned.

Expires

- The time at which document should be considered out-of-date and thus should no longer be cached.

Last-Modified

- The time document was last changed.

Refresh

- The number of seconds until browser should reload page. Can also include URL to connect to.

WWW-Authenticate

- The authorization type and realm needed in Authorization header.

HTTP Summary

1. HTTP is a synchronous, stateless protocol between a client and server. It is often connectionless.
2. While it is the *lingua franca* of the Web, realize it is often used as a transport proxy for other applications
   - Usually allowed through firewalls
3. HTTP != HTML. They have nothing to do with each other
4. You (the web application programmer) are responsible, in whatever language or framework, for processing the request header and payload
   - But, the language/framework can make it easier on you
5. The service programmer is responsible for assembling a response, including the response header and payload
   - Again, this is where your tools can help you out