Servlets
Query Parameters, Sessions

Devdatta Kulkarni

Passing parameters to Servlets

Passing Parameters to Servlet

- Query Parameters
- Request Headers
- Request Body

Query Parameters

- Parameters that are passed with the resource URL as a query string
  - Query String: Starts with ‘?’
  - Parameters separated by: &

- Example: query-parameters

Parsing different parts of the resource

- getRequestURL
  - Returns the entire URL
- getRequestURI
  - Returns the context root
- getServletPath
  - Returns the url-pattern

- Example:
  - Use ‘query-parameters’

Request Headers

- We can set arbitrary headers
- Convention is to start the header with “X-”
  - http://stackoverflow.com/questions/3561381/custom-http-headers-naming-conventions

Request Body

- Parameters can be sent in as request body

- Example:
  - Use ‘query-parameters’ doPost
  - Use RESTClient Firefox add-on

When to use which type of parameter?

- Query parameters
  - When the parameters are concerned with the resource itself
    - Search queries that will provide subset of resources in response
- Request Headers
  - When the parameters are concerned with the request/response interaction
    - E.g.: Cookies
- Request Body
  - When the parameters are related to the resource's content
    - E.g.: Form submission

Sessions

What is a Session

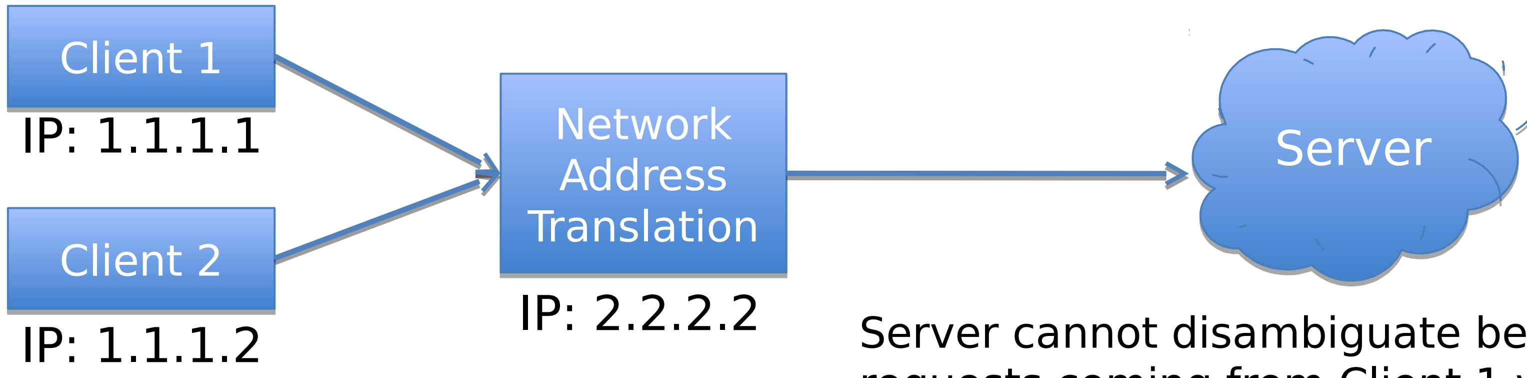
- Application-defined set of logical actions that have some application-specific semantic meaning
  - Example 1: Shopping cart
    - A session is actions between user adding something to a cart till the payment is confirmed on the payment screen
  - Example 2: Email systems
    - A session is actions between a user logging in and user logging out
      - All the activity during that time belongs to that session

Why are sessions needed?

- So that the server will know that *consecutive requests* are part of the *same user-level action*
  - Why is that needed?
    - To support high-level user actions such as doing online purchases
    - To customize and personalize displayed information
      - Recommendations on Netflix

Approaches to track requests?

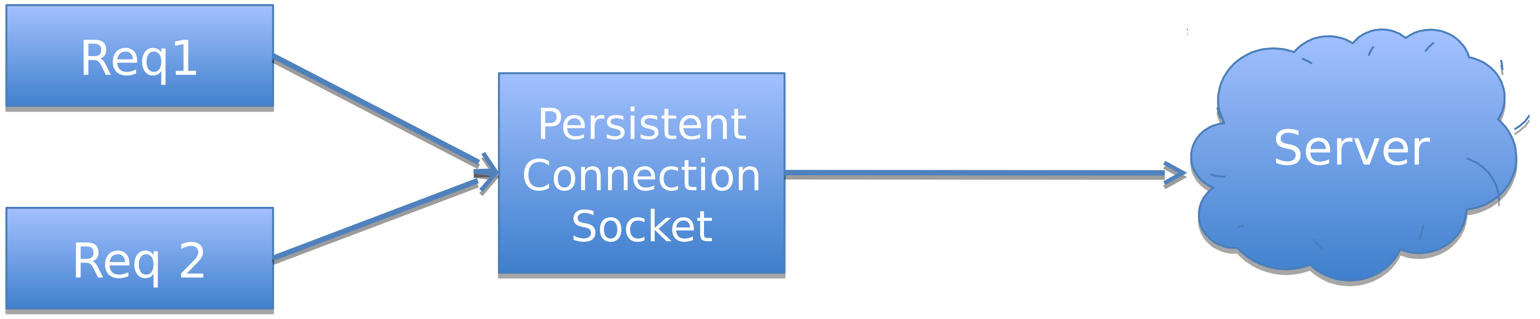
- Server needs something unique to identify a client
  - Could use Client IP + Browser combination
    - Problematic due to things such as Network Address Translation



Server cannot disambiguate between requests coming from Client 1 vs Client 2

Approaches to track requests?

- What about requests coming on a particular ``persistent connection’’



Could be used; but having a persistent connection is not a necessary condition

Persistent Connections vs. Sessions

- Persistency is the property of the transport mechanism
  - A persistent connection is essentially a ``long-lived'' Socket
- A Session is the property of a group of HTTP requests
  - A request is either part of a Session or it is not
- Requests belonging to a Session *may or may not* use the same persistent connection

Approaches to track requests

- Use a *unique session_id* to identify a client
  - Server generates and sends it to the Client
  - Client then sends the *session_id* in each subsequent request
  - Server considers all the requests that include the *session_id* as belonging to one session

Session

- How to send *sessionid* to the client?
  - What mechanisms does the server have to send back data to the Client?
    - Approach 1: Response Headers
    - Approach 2: Resource URL itself
- Approach 1: By embedding it in a Response Header (Cookies)
  - Example
    - Use: session-example
- Approach 2: By embedding it within resource URL itself
  - URL Rewriting

Session Cookie

- Approach 1: By embedding it in a Response Header (Cookies)

- What is a Cookie?
  - Small piece of information sent by the server in a response header to the client for tracking purposes

Session Cookie

- Server uses *Set-Cookie* response header to send the *sessionid* to the client

- Client uses *Cookie* request header to send the *sessionid* in subsequent requests
  - RFC 2109
    - https://www.ietf.org/rfc/rfc2109.txt

Session Cookie

- Cookie attributes
  - Cookie name
  - Maximum age
  - Secure flag
    - The client will send the cookie only if the server is operating with a secure protocol like https
  - Http-only flag
    - The cookie will be accessible only to browsers and not to technologies such as JavaScript

Session Cookie

- Cookie name, cookie value:
  - ASCII character set
    - http://stackoverflow.com/questions/1969232/allowed-characters-in-cookies
  - Maximum age
    - Specified in seconds
    - Negative value means that the cookie won’t be persistently stored

Session Cookie

- Cookie attributes
  - Domain name
    - Cookie set by parent domain is available to all its sub-domains
      - Example:
        - Cookie set by cs.utexas.edu will be sent when requesting the page for www.cs.utexas.edu
        - But it won’t be sent for utexas.edu
  - Path
    - Resource path

Session: Server side

- How to send session_id to the Client
  - By embedding it within resource URL itself
    - URL Rewriting
  - By embedding it in a Response Header as a ``Cookie’’
    - Use Set-Cookie Response header
      - RFC 2109
        - » https://www.ietf.org/rfc/rfc2109.txt
    - What is a ``Cookie’’?
      - Small piece of information used for tracking purposes

Session: Client side

- How to send session_id to the Server
  - URL_Rewriting method used
    - Nothing to do; the session_id is available in the URL itself
  - Set-Cookie response header used
    - Send the session_id within the ``Cookie’’ request header

Examples

- Cookie storage in Firefox
  - Show how to access cookies.sqlite
    - http://stackoverflow.com/questions/7610896/how-to-use-sqlite-to-read-data-from-the-firefox-cookies-file

- SQLite Database Browser
  - http://sourceforge.net/projects/sqlitebrowser/

Checking Cookies in Firefox

- Firefox version 35.0.1
  - Preferences -> Privacy -> remove individual cookies -> localhost
- Find the cookies file (on MacOS)
  - Finder -> CMD+Shift+G
    - ~/Library/Application Support/Firefox/Profiles
    - cookies.sqlite
    - sqlitebrowser

Cookie handling on User Agent

- Latest Http state management RFC
  - http://tools.ietf.org/html/rfc6265
- Cookie header handling algorithm (Section 5.4)
  - Cookies with longer paths are listed before cookies with shorter paths.
  - Among cookies that have equal-length path fields, cookies with earlier creation-times are listed before cookies with later creation-times.
  - Update the last-access-time of all cookies to the current date and time.

URL Rewriting

URL Rewriting

- Server will generate a unique session ID
- Within application code, we ``rewrite’’ URL by ``encoding’’ it with the generated session ID
  - Encoding step appends the session ID to the URL
    - JSESSIONID=<sessionID>
- When the encoded URL is clicked the request is considered to be part of that session

URL Rewriting

- Example:
  - Use url-rewriting:
  - Key steps:
    - HttpSession session = request.getSession(true)**;**
    - String encodedURL = response.encodeURL(url);
  - request.getSession():
    - Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session
  - response.encodeURL():
    - Encodes the specified URL by including the session ID in it

URL Rewriting: Issues

- Unsafe
  - Session id visible within the URLs
- All the URLs that need to be considered as part of a particular session workflow need to be encoded
- Can we use it to control access to different url patterns?
  - Yes;
    - String url = request.getRequestURL().toString(); is specific to a URL
- URL Rewriting is useful when browser does not support cookies

- How to expire the session when it is tracked using URL rewriting?
  - One approach: remove the session id from all the URLs

Cookies: Security Issues

- Copy and Paste Mistake
  - User sends the link with session ID embedded in it to others
    - Show using url-rewriting
  - How to avoid
    - Disable embedding sessionIDs in URLs
      - So no URL rewriting method for session tracking
- Session Fixation
  - Attacker sends the links to the victims with a session ID embedded in it
  - When users click on it, the attacker gets the control

Cookies: Security Issues

- Cross-Site Scripting and Session Hijacking
  - Attacker injects attack JavaScript into vulnerable site
  - Attack JavaScript copies the session ID cookie using ‘document.cookie’ property
    - How to prevent?
      - Set ``HttpOnly’’ attribute of the cookie
- Insecure Cookies
  - Man-in-the-middle attack
    - Attacker observing network traffic between Client and Server
    - How to prevent?
      - Use the ``Secure’’ attribute
        - » Indicates that the cookie should only be transferred over HTTPS
          - Cookie is transmitted as encrypted
      - Drawback:
        - » Site needs to be behind HTTPS

Readings

- Chapters 1, 2, 3, 5 of ``Java for Web Applications’’ book