Managing Data on the World Wide Web

Cookies and Tomcat
Overview

1. Cookies
2. Web Container
3. Tomcat
HTTP Cookies

• An HTTP is a small piece of data sent from a website and stored in the user's web browser while the user is browsing it.

• Every time the user loads the website, the browser sends the cookie back to the server to notify the user's previous activity.
HTTP Cookies

Client
(Firefox browser)

GET /index.html HTTP/1.1
Host: www.example.org
...

HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123;
   Expires=Wed;
   09 Jun 2021 10:18:14 GMT
...

Server
(www.example.org)

GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light;
       sessionToken=abc123
...

GET /index.html HTTP/1.1
Host: www.example.org
...
HTTP Cookies

• Usually an Http cookie is merely a name-value pair
  – a cookie also contains a number of attributes such as: version, a domain for which is valid, etc.

• Usages:
  – Provides mechanism for websites to remember stateful information (e.g., "shopping cart")
  – Authentication cookies: common method used by web servers to know whether the user is logged in or not
HTTP Cookies

• Types of cookies:
  – Session cookies, persistent cookies, third-party cookies, etc.

• Privacy issues:
  – A web page may contain images or other components stored on servers in other domains
  – Third-party components are a potentially powerful way to track Internet users

• Security issues:
  – Identity theft, DNS cache poisoning
Overview

1. Cookies
2. Web Container
3. Tomcat
Web Servers

- A **computer program** that:
  - Accepts HTTP requests from clients and serves HTTP responses
  - Usually a web content
    - e.g., HTML documents, images and etc.
- Sometimes the term does not refer to a program but rather to the **machine/system** that runs it
- Websites, Gaming, Data Storage, Emails, etc.
Common Web Server Features

- **Virtual hosting** to serve many Web sites using one IP address

- **Bandwidth throttling** to limit the speed of responses in order to not saturate the network and to be able to serve more clients

- **Server-side scripting** to generate dynamic Web pages, still keeping web server and website implementations separate from each other
A servlet is a small Java program that runs within a Web server.

Servlets receive and respond to requests from clients.
- Usually for creating dynamic content

The Servlet API defines the expected interactions of the web container and a servlet.
- I.e., a Servlet is a Java object that receives a request and generates a response based on that request.
- The API is defined in javax.servlet
A web container (AKA servlet container) is a web server component that interacts with the servlets. It is responsible for:

- Managing the lifecycle of servlets
  - Creates servlet instances
  - Loads and unloads servlets
  - Etc.
- Mapping a URL to a particular servlet
- Ensuring proper access rights to URLs
**Life Cycle of a Servlet**

- **Initialization**: the web container initializes the servlet instance by calling the `init()` method

- **Activation**: the servlet can service client requests
  - each request is serviced in its own separate thread
  - the web container calls the `service()` method per request
  - it determines the request type and dispatches it to the proper method

- **Destruction**: the web container calls the `destroy()` method to take the servlet out of service

- Servlets may provide an implementation for the above methods
@WebServlet("/MyPattern")
public class ServletLifeCycleExample extends HttpServlet {
    private int count;
    public void init(ServletConfig config) throws ServletException {
        count=100;
    }

    public void destroy() {
        ...
    }

    protected void service(HttpServletRequest request,
                           HttpServletResponse response) throws ServletException, IOException {
        // forwarding requests to do[Get,Post,Head,...]() methods
        super.service(request, response);
        response.getWriter().append("service() called\n");
        response.getWriter().append("Count = " + count++);
    }

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.getWriter().append("doGet() called\n");
    }
}
The Servlet API 3.0 introduces a new package called \texttt{javax.servlet.annotation} which provides annotation types.

The annotations can replace equivalent XML configuration in the web deployment descriptor file (web.xml) such as servlet declaration and servlet mapping.
Servlet Annotations - Types

- @WebServlet - used for declaring a servlet class and configuring mapping for it
  - the class still must extend from the HttpServlet class

- @WebInitParam - specifies an initialization parameter

- @MultipartConfig – enables handling multipart/form-data requests (e.g., for uploading file to server)

- And more:
  @HttpConstraint, @HttpMethodConstraint, @WebFilter, @ServletSecurity, @WebListener, @WebServlet
• Supports the **communication** between a servlet and its container
• Common usage:
  – get the MIME type of a file
  – dispatch requests
  – or write to a log file
• One context per "web application" per Java Virtual Machine
• More on next tutorial

```java
protected void doGet(...) {
    getServletContext().
        getMimeType("index.html");
}
```

*Defined in GenericServlet (the superclass of HttpServlet)*
A JSP compiler is a program that parses JSPs, and transforms them into executable Java Servlets. A program of this type is usually embedded into the application server (e.g., Tomcat).

Output:
Counting to three:
This number is 1.
This number is 2.
This number is 3.
OK.
JSP – Automatic Generation of Servlets

JSP Container
(a) Translation occurs at this point, if JSP has been changed or is new.
(b) If not, translation is skipped.
Overview

1. Cookies
2. Web Container
3. Tomcat
What is Tomcat?

- Apache Tomcat is an open-source web server
  - Developed by the Apache Software Foundation

- Provides a "pure Java" HTTP web server environment for Java code to run in
  - OS independent

- Implements the Servlet 3.1 and JSP 2.3 specifications
Tomcat Directory Structure

• $CATALINA_HOME – Root directory of the Tomcat installation
  
• /bin - Startup, shutdown, and other scripts  
  
• /conf - Configuration files.  
  – server.xml is the most important one 
  
• /logs - Default location of Log files  
  
• /webapps - This is where your webapps go
Tomcat Directory Structure

```
Tomcat-Base
  ├── conf
  │    └── server.xml
  │          └── Tomcat-users.xml
  │          └── myApp1
  │                  └── WEB-INF
  │                      └── web.xml
  │                      └── lib
  │                          └── classes
  │                                        └── JAR Files
  ├── logs
  │ └── myApp2
  │ └── myApp3
  └── work
      └── lib
          └── JAR files
```
Tomcat Directory Structure

- `myapp/`
  - contains HTML, CSS, GIF and other static content

- `myapp/WEB-INF/`
  - contains the *deployment descriptor* `web.xml`

- `myapp/WEB-INF/classes/`
  - contains servlet class files

- `myapp/WEB-INF/lib/`
  - contains extra jar files
Tomcat Manager

- Tomcat has a built in web app called “manager”
  - this app supports the management of webapps

- The supported features are:
  - Deploying a web application by posting a WAR file
  - Undeploy a deployed Web application
  - Start/Stop a Web application (available/unavailable)
  - Reload an existing web application (unpack new WARs)

- Notice that while “stop” makes an application unavailable, “undeploy” deletes the application directory and WAR file from webapps/
Deployment

- Deployment is the term used for the process of installing a web application into the Tomcat server.

- The Tomcat Manager is a web application that can be used interactively (via HTML GUI) to deploy and manage web applications.
  - Allows to deploy and undeploy applications on a running Tomcat server without restarting it.
• Apache Tomcat Maven Plugin project provides integration with Apache Maven
  – mvn tomcat7:deploy (should work of Tomcat 8 as well)

• Static deployment: the web application is setup before Tomcat is started
  – Copy your web application to $CATALINA_BASE/webapps then start Tomcat
  – The web application can be compressed (.WAR file), or non-compressed (exploded web application)
• Tomcat can be easily integrated into Eclipse

• All you need to do is download Apache Tomcat and create a new server in Eclipse

• Helpful YouTube guides:
  – Install and Configure Apache Tomcat Web Server in Eclipse IDE
  – Setting up, Download & Install Apache Tomcat & Eclipse
Web Archives

- A WAR (Web Archive) file is a JAR file that contains an entire Web-application directory
  - For example, to create a WAR file of myApp do:
    - `cd webapps/myApp`
    - `jar cvf myApp.war *`
- Upon startup, Tomcat will unpack all WAR files found under `$CATALINE_BASE/webapps/`
- The unpacked directory and context will be named as the WAR file name (without the .war extension)
  - The WAR will **NOT** be unpacked if webapps/ already contains the directory and the WAR is not newer
Running a Web Application in Tomcat

- Tomcat automatically identifies a directory
  \$CATALINA_BASE/webapps/myApp/ and associates it with the relative http://machine:port/myApp/
  - e.g., the file named index.html in myApp is mapped to by http://machine:port/myApp/index.html

- Subdirectories under myApp can also be used.
  - e.g., the file myApp/myImages/im.gif is accessed by the URL http://host:port/myApp/myImages/im.gif
What is a Deployment Descriptor?

• An XML file (web.xml) which holds the following:
  – mapping from URIs to application resources
  – initialization parameters
  – security constraints
  – registration of listeners and filters
• Application-specific configuration and declarations are written in the file `myApp/WEB-INF/web.xml`

• This file contains:
  – Servlet declarations, mappings and parameters
  – Default files for directory requests (e.g. `index.html`)
  – Error pages (sent in cases of HTTP errors)
  – Security constraints
  – Session time-out specification
  – Context (application) parameters
  – And more…
The element `<servlet>` declares a Servlet

The sub element `<init-param>` defines a parameter passed to the Servlet, *can be accessed by using:*
  
  ```java
  ServletConfig.getInitParameter(String paramName)
  ```

The element `<servlet-mapping>` maps a URL to a specific Servlet

The URL is relative to the application’s base URL *(http://host:port/myApp/)*
Publishing a Servlet - An Example

```xml
<web-app>
  <servlet>
    <servlet-name>MyServletName</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServletName</servlet-name>
    <url-pattern>/myPattern</url-pattern>
  </servlet-mapping>
</web-app>
```

http://localhost/myApp/MyPattern

The 2 blocks do not have to be adjacent within web.xml
• Use the `error-page` element to define the page sent in case of an HTTP error

• An error page element has two sub elements:
  – `error-code` -- the HTTP error status code
  – `location` -- the page that should be sent
Error Page Example

```html
<html>
<head><title>Not Found</title></head>
<body>
<h1 style="text-align:center; color:green">Sorry, no such file...</h1>
</body>
</html>
```

```web-app
<error-page>
  <error-code>404</error-code>
  <location>/my404.html</location>
</error-page>
</web-app>
```