Cliente HTTP

Programación Web

1014-2918 Darvy Betances

Prof. Carlos Camacho

Índice

Introducción

El tema a tratar para este reporte es la creación de un cliente HTTP, pero antes de entender que es el “Cliente” como tal debemos saber cómo funciona y que es “HTTP”, durante este reporte estaremos trabajando con los términos antes mencionados, profundizando un poco en el tema claro está, igualmente, presentaremos el ejercicio propuesto por el maestro de la asignatura, en donde se aplicarán los conceptos que se estén trabajando en el momento y concluiremos presentando la importancia y los casos de usos de los clientes HTTP. Igualmente para lograr este proyecto se hizo uso del controlador de versiones Git y el sistema de construcción de software Gradle.

Una breve introducción para entender lo que es un cliente HTTP sería “un programa o aplicación que se utiliza para enviar solicitudes a un servidor web y recibir respuestas. El cliente HTTP es el que inicia la comunicación y hace la petición de información al servidor web.” HTTP significa Hypertext Transfer Protocol y ya podemos entender el porqué de la definición anterior. El navegador trabaja juntamente con lo que son los servidores HTTP ya que es donde se almacenan las paginas Web, por ello se le llama la relación “Cliente - Servidor” y por lo tanto de ahí deriva el nombre Cliente HTTP.

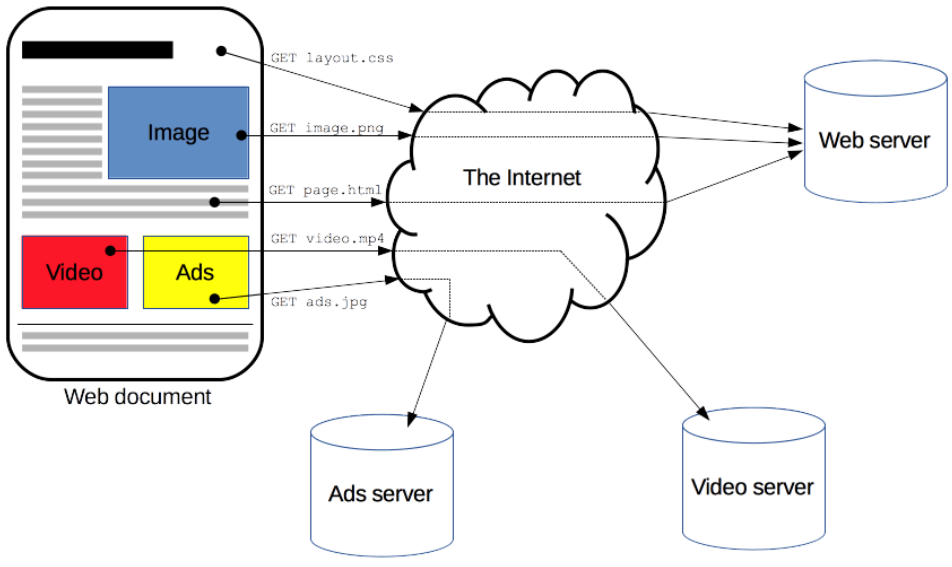
Por otro lado. Gradle lo utilizamos para automatizar tareas de desarrollo de software, como compilación, pruebas, empaquetado, distribución, entre otras cosas que hace que la vida de los desarrolladores sea un poco más fácil y que el trabajo sea más eficiente. Por ello en este reporte se hace uso de esta tecnología.

Por último, es importante resaltar el uso del controlador de versiones git que fue utilizado para gestionar el proyecto en donde pusimos en práctica lo que es el cliente HTTP. Git es una herramienta esencial para cualquier desarrollador o equipo de desarrolladores que quiera mantener un registro de los cambios en su código y colaborar de manera efectiva, aunque en este caso no se colaboró, pero fue bastante útil para controlar los cambios y avance del proyecto.

Desarrollo

De una manera más profunda, el protocolo de transferencia de Hipertexto es un protocolo de la capa de aplicación utilizado para transmitir información en la Web, pero hay algo que tenemos que entender bien y es que HTTP no es una sola tecnología ni es algo que se maneja por si solo, hay muchos otros protocolos que trabajan en conjunto con este para permitir la correcta comunicación en la web y en la red en general, algunos de estos pueden ser:

- TCP/IP: HTTP se basa en TCP/IP para establecer conexiones y transmitir información.
- DNS: Es el protocolo de nombres de dominio que se utiliza para traducir los sitios web en direcciones IP.
- SSL/TLS: Es un protocolo de seguridad que se utiliza para encriptar la información que se transmite a través de HTTP, garantizando la privacidad y la integridad de la información.



(Imagen 1: https://developer.mozilla.org/es/docs/Web/HTTP/Overview)

Ahora bien, la relación entre HTTP y el cliente como se mencionó en la introducción está denotada por lo que son los navegadores web como

Google Chrome, Mozilla Firefox, Safari, etc. Y se puede ver en la imagen 1 un pequeño esquema de cómo funciona, clientes y servidores se comunican intercambiando mensajes individuales, los mensajes que el cliente envía (en el navegador web) se llaman *peticiones* y los mensajes enviados por el servidor son *respuestas*. Por lo tanto, tenemos dos agentes dentro la “ecuación”, el agente usuario y el agente servidor

Agente Usuario: Es cualquier herramienta que actúe en nombre de un cliente, función desempeñada por los navegadores.

Agente Servidor: Es el cual "sirve" los datos que ha pedido el cliente, este puede reconocerse como una entidad conceptual aunque se forma de varios elementos

Relación con el problema resuelto:

Como complemento del reporte se tuvo la tarea de implementar un Cliente HTTP haciendo uso de JAVA y algunas bibliotecas como JSOUP y httpComponents, estaremos interpretando las peticiones del protocolo HTTP y HTML de forma directa, para ello estaremos creando un programa por consola que pida por la entrada estándar una URL valida.

Para es necesario tener en cuenta que el cliente HTTP envía una petición HTTP al servidor, y esta a su vez incluye información bastante útil que se manifiesta mediante la URL deseada, los encabezados de la solicitud y, opcionalmente, un cuerpo de solicitud. Estos tres parámetros son vitales para poder interpretar la información deseada. Por ejemplo, si queremos detectar el tipo de archivo al cual estamos haciendo una petición pues solo hay que obtener el header que contenga esa información y parsearla para imprimirla.

Recordando:

El header contiene información adicional sobre la solicitud o respuesta, incluyendo el tipo de contenido, la codificación de caracteres, la longitud del cuerpo, la fecha y hora, y otros metadatos que describen el recurso solicitado o la respuesta generada.

El servidor recibe la petición y, si es válida, envía una respuesta HTTP, que incluye un código de estado que indica si la solicitud fue exitosa o no, encabezados de respuesta y, opcionalmente, un cuerpo de respuesta. El cliente HTTP luego interpreta la respuesta y procesa la información que ha recibido. Este es el mecanismo básico de peticiones y respuestas en un cliente HTTP.

```
try(CloseableHttpClient httpClient = HttpClients.createDefault()){
    HttpGet solicitud = new HttpGet(URL);
    HttpResponse respuesta = httpClient.execute(solicitud);

    //Verificando que tipo de archivo es la URL

    Header[] headerTipo = respuesta.getHeaders( name: "Content-Type");

    if (!headerTipo[0].getValue().contains("text/html")){
        System.out.println();
        System.out.println("----------------------------------------");
        System.out.println("      El Archivo es " + headerTipo[0].getValue() + "    ");
        System.out.println("----------------------------------------");
    }
    else{
        System.out.println();
        System.out.println("----------------------------------------");
        System.out.println("-           El Archivo es HTML        -");
        System.out.println("----------------------------------------");
```

Podemos ver en ese ejemplo como hacemos uso de los headers en el proyecto para determinar si el recurso es HTML o no, pero antes fijémonos en como creamos una instancia de un cliente HTTP y luego creamos la instancia de un response para obtener la solicitud que sería la URL mediante un GET.

- **GET** es un método utilizado para obtener un recurso específico del servidor. Tiene información adicional sobre la solicitud, como los parámetros de búsqueda, se envía en la URL.
- **POST** es un método utilizado para enviar información al servidor, como un formulario en línea. La información se envía en el cuerpo de la solicitud y es invisible para el usuario.

Esos métodos en el protocolo HTTP y se utilizan para solicitar información como se realizó en el proyecto.

Por cierto, la creación del proyecto se pudo haber realizado completamente en JSOUP, pero por motivos de conocer las dos tecnologías decidí hacer uso de las dos, HttpComponents para obtener y recursos y

hacer conexiones y JSOUP para acceder a elementos de los recursos obtenidos con anterioridad.

Extraer información y Cuerpos de Mensajes

Algo interesante que vimos también en la práctica es el tema de solicitar información para luego parsearla y poder aprovechar para tener datos útiles. Este es uno de los grandes poderes de un cliente HTTP, poder acceder a los recursos de un documento de hipertexto como HTML, y poder conocer datos importantes, como la cantidad de elementos de cierto tipo. Para hacer esto se hizo uso de la función getEntity de HttpComponents que es una manera de acceder a la información de la respuesta HTTP, para luego procesarla o utilizarla para cualquier otro propósito. Por ejemplo, si se realiza una petición HTTP y se obtiene una respuesta HTML:

```
String documento = EntityUtils.toString(respuesta.getEntity());
//Para algunos de los siguientes puntos voy a utilizar Jsoup para aprovechar su formateador HTML
Document doc = Jsoup.parse(documento);

//1. Indicar la cantidad de lineas del recurso retornado

System.out.println("  > Cant. Lineas: " + cantLineas(documento));
System.out.println();

//2. Indicar la cantidad de párrafos (p) que contiene el documento HTML

Elements p = doc.select( cssQuery: "p");
System.out.println("  > Cant. Parrafos: " + p.size());
System.out.println();

//3. Indicar la cantidad de imágenes (img) dentro de los párrafos que contiene el archivo HTML

Elements imgParrafos = p.select( query: "img");
System.out.println("  > Cant. Imagenes dentro de párrafos: " + imgParrafos.size());
System.out.println();
```

Esta información también podemos filtrarla mediante las mismas propiedades de JSOUP.

Pero así como recibimos información también podemos enviarla y es que podemos crear nuestros propios headers y enviar esa respuesta al servidor, información a la que podremos acceder después haciendo las peticiones que hicimos anteriormente.

Para esta parte recordar los métodos GET y POST y el funcionamiento de petición y respuestas del protocolo HTTP

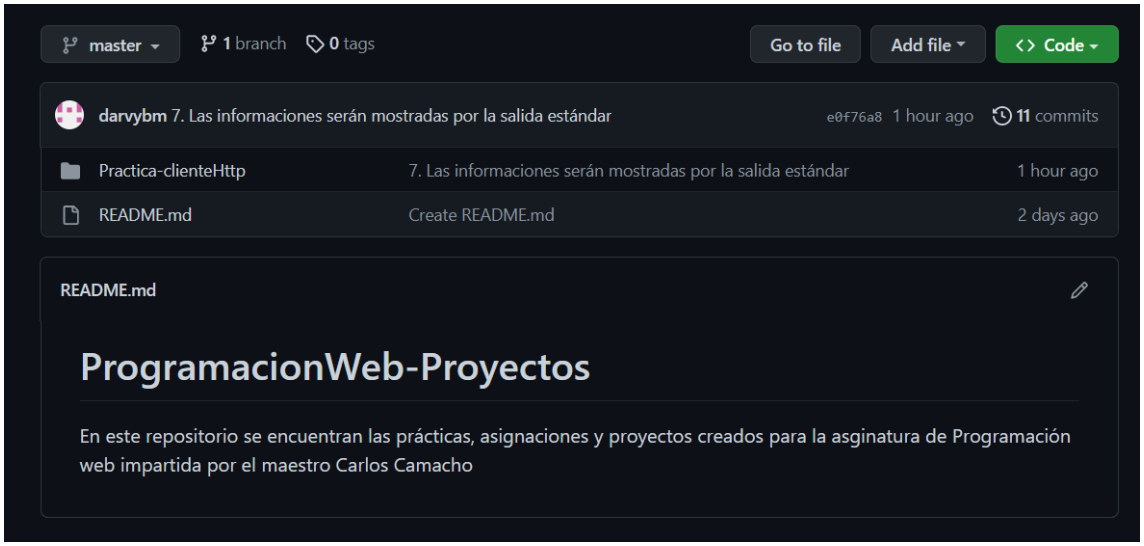
```java
System.out.println("  > Forms - obteniendo respuesta a la petición realizada que es el header y el cuerpo del mensaje (con statusCode)");
for (int i = 0; i < forms.size(); i++) {
    Element form = forms.get(i);
    if (form.attr( attributeKey: "method").equalsIgnoreCase( anotherString: "POST")){
        System.out.println("    > Form: " + String.format("%0" + 3 + "d", i));
        //Obteniendo la url del form action
        HttpPost httpPost = new HttpPost(form.absUrl( attributeKey: "action"));

        //Creando el header mediante la función setHeader()
        httpPost.addHeader( name: "matricula-id",  value: "1014-2918");

        //Creando la petición mediante la función setEntity()
        ArrayList<BasicNameValuePair> parametros = new ArrayList<BasicNameValuePair>();
        parametros.add(new BasicNameValuePair( name: "asignatura",  value: "practica1"));
        httpPost.setEntity(new UrlEncodedFormEntity(parametros));
        CloseableHttpResponse response = httpClient.execute(httpPost);
        System.out.println("        - Status code: " + response.getStatusLine().getStatusCode());
    }
}
```

Se puede observar aquí como se crea una instancia de tipo POST (para enviar información mediante el cuerpo del mensaje) y agregamos la información a ese post, siendo esta un Header y un mensaje, los cuales enviamos al servidor mediante la misma instancie de un cliente HTTP que se creó anteriormente.

Es algo bastante intuitivo en verdad y bastante sencillo de comprender y el proceso fue aún más sencillo al hacer uso de Gradle para importar las librerías y ejecutar el proyecto mediante la tarea Run.

```
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
    implementation 'org.apache.httpcomponents:httpclient:4.5'
    implementation 'org.jsoup:jsoup:1.15.3'
}
```

Igualmente, git fue muy útil al permitirme gestionar el proyecto de manera que pueda visualizar el avance de este y controlar las versiones.



Conclusión

En conclusión. El cliente HTTP es importante porque es la parte de una aplicación que se encarga de enviar peticiones a un servidor web y recibir las respuestas. Permite a las aplicaciones acceder y obtener información de recursos en línea, como páginas web, imágenes y otros tipos de contenido. Además, permite la interacción con los servidores, lo que facilita la creación de aplicaciones web dinámicas y la interacción en tiempo real con los usuarios. Mediante esta tecnología podemos obtener mucha información ya sea a través de los headers, como de la URL o cuerpo del mensaje. En resumen, el cliente HTTP es fundamental para la implementación de aplicaciones y servicios en la web.

El tema trabajado puede ser aplicado en diferentes áreas donde se necesiten peticiones web o invocar recursos como podría ser:

1. Navegación web: los navegadores web son clientes HTTP que envían solicitudes a servidores web para descargar y mostrar páginas web.
2. Aplicaciones móviles: las aplicaciones móviles a menudo usan clientes HTTP para comunicarse con servidores web y acceder a servicios en la nube o a datos en línea.
3. Desarrollo de software: los desarrolladores de software utilizan clientes HTTP para probar y depurar servicios web y aplicaciones.

En resumen, los clientes HTTP son una parte esencial de muchas aplicaciones y tecnologías en línea y son esenciales para la comunicación eficiente y confiable con servidores web.

En mi experiencia personal, trabajar con un cliente HTTP fue un poquito difícil al principio porque había muchos conceptos nuevos que entender. pero luego de poner en práctica todo lo que estaba leyendo logré completar con éxito el proyecto y a su vez comprender el tema de una buena forma, además, una vez que comprendí los conceptos básicos de la petición HTTP y la respuesta del servidor, fue mucho más fácil trabajar con el cliente. Además, trabajar con un cliente HTTP me permitió tener una mejor comprensión de cómo funcionan las aplicaciones web y cómo se comunican los diferentes componentes a través de la red. Fue una experiencia enriquecedora y estoy emocionado de seguir explorando más sobre el desarrollo de aplicaciones web.

Bibliografía

- *Browser, Navegador, Explorador o Cliente HTTP. (n.d.). Edu4java.com. Retrieved February 7, 2023, from http://www.edu4java.com/es/web/web2.html*

- *Caules, C. Á. (2015, April 1). ¿Qué es Gradle? Arquitectura Java. https://www.arquitecturajava.com/que-es-gradle/*

- *Cómo utilizar métodos get y post en una app de ingresos y gastos. (2022, August 12). KeepCoding Tech School. https://keepcoding.io/blog/metodos-get-y-post-en-app-ingresos-y-gastos/*

- *Fernández, Y. (2019, October 30). Qué es Github y qué es lo que le ofrece a los desarrolladores. Xataka.com; Xataka. https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores*

- *Generalidades del protocolo HTTP. (n.d.). Mozilla.org. Retrieved February 7, 2023, from https://developer.mozilla.org/es/docs/Web/HTTP/Overview*

- *Hedley, J. (n.d.). jsoup: Java HTML parser, built for HTML editing, cleaning, scraping, and XSS safety. Jsoup.org. Retrieved February 7, 2023, from https://jsoup.org/*

- *HTTP. (n.d.). Concepto. Retrieved February 7, 2023, from https://concepto.de/http/*

- *HttpClient overview. (n.d.). Apache.org. Retrieved February 7, 2023, from https://hc.apache.org/httpcomponents-client-5.2.x/*

- *Wikipedia contributors. (n.d.). Cliente-servidor. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Cliente-servidor&oldid=149100803*