4.1 The protocol: Keep-Alive

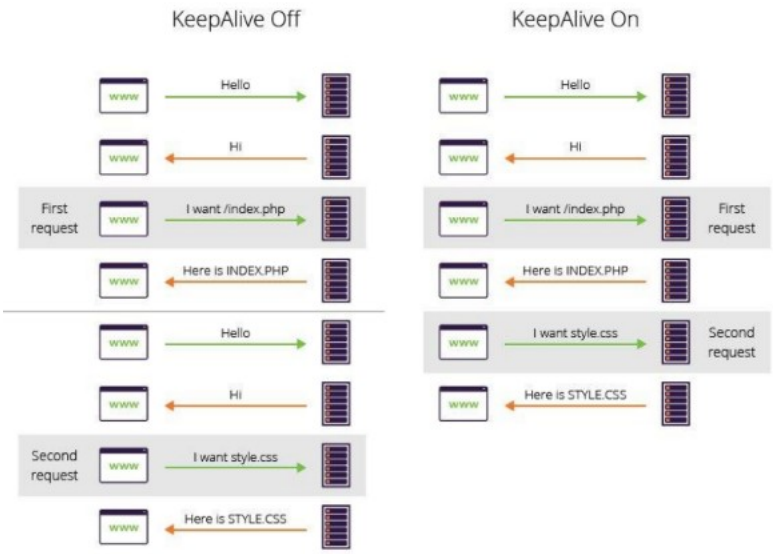
4.2 HTTP Methods

4.3 The Request Body

4.4 Cookies: CookieManager and CookiesStore

The protocol: Keep-Alive

The `Keep-Alive` protocol in HTTP is a way to reuse a single TCP connection to send and receive multiple HTTP requests/responses, instead of opening a new connection for every single request/response pair. This can improve the performance of web applications by reducing the overhead of establishing new connections.



In Java, you can control the `Keep-Alive` behavior using various libraries, such as the standard `java.net.HttpURLConnection` class or third-party libraries like Apache HttpClient. Here's how you can manage `Keep-Alive` connections using both approaches:

Standard java.net.HttpURLConnection:

- Although `HttpURLConnection` doesn't have a direct method to enable Keep-Alive, it utilizes this behavior by default in HTTP/1.1 connections.
- You can set the `Connection: keep-alive` header in the request to explicitly request Keep-Alive from the server (though typically not necessary).

- Remember to close the connection after you're done to release resources on the server-side.

2. Apache HttpClient:

- Apache HttpClient provides more granular control over Keep-Alive behavior.
- You can use `HttpClientBuilder` to configure the `SocketConfig` and set `SO_KEEPALIVE` to `true`. This enables Keep-Alive on the underlying socket connections.
- `RequestConfig` allows setting timeouts for connection establishment and data transfer while using Keep-Alive.

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class HttpKeepAlive {
    Run main | Debug main
    public static void main(String[] args) {
        try {
            URL url = new URL("http://www.google.com");
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");

            // Enable Keep-Alive
            connection.setRequestProperty("Connection", "keep-alive");

            int responseCode = connection.getResponseCode();
            System.out.println("Response Code : " + responseCode);

            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String inputLine;
            StringBuffer response = new StringBuffer();
                while ((inputLine = in.readLine()) != null) {
                    response.append(inputLine);
                }
                in.close();

                System.out.println(response.toString());

                // The connection will be kept alive and can be reused
            } catch (Exception e) {
                System.err.println(e);
            }
        }
}
```

//Program to demonstrate http header field
//Write a program to read URL showing the methods to print their content type, content length, content encoding, date of last modification, expiration data, and current date.

```java
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.*;


public class HttpHeaderDemo{
 Run main | Debug main
 public static void main(String[] args) throws Exception {
        try{
            URL u = new URL("http://www.javatpoint.com/java-networking");
            URLConnection uc = u.openConnection();
            if(uc.getContentEncoding() != null){
                System.out.println("Content-encoding:"+uc.getContentEncoding());
                }
                if(uc.getDate() != 0){
                    System.out.println("Date:"+ new Date(uc.getDate()));
                    }
                    if(uc.getLastModified() != 0){
                        System.out.println("Last modified:"+ new Date(uc.getLastModified()));
                        }
                    if(uc.getExpiration() != 0){
                        System.out.println("Expiration date:"+ new Date(uc.getExpiration()));
                    }
                      if(uc.getContentLength() != -1){
                          System.out.println("Content-length"+ uc.getContentLength());
                      }

                      }catch(MalformedURLException ex){
                      //System.err.println(ex);
                      System.err.println("is not a URL I understand.");
                      }


 }

}
```

Output

```
Date:Sun Jun 30 05:11:02 NPT 2024
Expiration date:Sun Jun 30 06:11:02 NPT 2024
Content-length167
```

Http Method

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Cookies: CookieManager and CookiesStore

A cookie is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

```
HttpCookie cookieA = new HttpCookie("NP", "Network Programming");
```

Types of Cookies:

1.Non-persistent cookie: It is valid for single session only. It is removed each time when user closes the browser.

2. Persistent cookie: It is valid for multiple session. It is not removed each time when user closes the browser. It is removed only if user logout or signout.

In Java, the `CookieManager` and `CookieStore` classes are part of the `java.net` package and are used for handling HTTP cookie

CookieManager:

- Acts as a middleman between the HTTP client and the cookie store.
- Handles receiving cookies from server responses and sending cookies in request headers.
- Uses a `CookieStore` to manage the actual storage of cookies.
- Can be set as the default manager for all HTTP connections.

Configuration:

- You can customize its behavior by providing:
  - A custom `CookieStore`: This allows you to control how cookies are stored (e.g., in memory, on disk).
  - A custom `CookiePolicy`: This lets you define rules for accepting or rejecting cookies based on specific criteria.

Method	Description
getCookieStore()	This method retrieves the current cookie store.
setCookiePolicy(CookiePolicy cookiePolicy)	This method is used to set the cookie policy of this cookie manager. E.g. ACCEPT_ORIGINAL_SERVER, ACCEPT_ALL, ACCEPT_NONE

CookieStore:

- Represents the storage mechanism for cookies.
- Provides methods for:
  - Adding new cookies.
  - Retrieving existing cookies for a specific URL.
  - Removing cookies.
- Implementations can store cookies in memory (in-memory store) or persist them (e.g., on disk).

Faculty: Gopal Maharjan (mahrzan.gopal@gmail.com)

Method	Action Performed
add(URI uri, HttpCookie cookie)	Adds one HTTP cookie to the store.
get(URI uri)	Retrieves cookies whose domain matches the URI.
getCookies()	Get all cookies in CookieStore which are not expired.
getURIs()	Get all URIs that identify cookies in CookieStore
remove(URI uri, HttpCookie cookie)	Removes a cookie from CookieStore
removeAll()	Removes all cookies in the CookieStore

In simpler terms:

- Imagine `CookieManager` as a receptionist at a hotel.
- The receptionist (CookieManager) receives cookie information (cookies) from the server (response header) and stores them in the hotel's storage (CookieStore).
- When a guest (HTTP client) makes a request, the receptionist checks the storage for relevant cookies and sends them along with the request (request header).
- You can decide where the hotel stores guest information (custom CookieStore) and have rules for who can stay (custom CookiePolicy).

Faculty: Gopal Maharjan (mahrzan.gopal@gmail.com)

```java
import java.net.*;
import java.util.List;

public class CookieExample {
    Run main | Debug main
    public static void main(String[] args) throws Exception {
        CookieManager cookieManager = new CookieManager();
        CookieStore cookieStore = cookieManager.getCookieStore();

        //creating cookies and URI

        HttpCookie cookieA = new HttpCookie("c1","ram");
        HttpCookie cookieB = new HttpCookie("c2","sita");
        HttpCookie cookieC = new HttpCookie("c3","hari");

        URI uri1 = URI.create("http://test1.com");
        URI uri2 = URI.create("http://test2.com");

        cookieStore.add(uri1, cookieA);
        cookieStore.add(uri2, cookieB);
        cookieStore.add(null, cookieC);
```

Faculty: Gopal Maharjan

```
        //Read stored all cookies

        List cookieList = cookieStore.getCookies();
        System.out.println("Cookie list in CookieStore: "+cookieList+"\n");

        //remove cookie of uri
        cookieStore.remove(uri2, cookieA);

        List remainingCookieList = cookieStore.getCookies();
        System.out.println("Remaining Cookies:"+ remainingCookieList+"\n");

        //remove all cookies
        cookieStore.removeAll();
        List EmptyCookieList = cookieStore.getCookies();
        System.out.println("Empty CookieStore:"+EmptyCookieList);


    }
}
```

Output

```
Cookie list in CookieStore: [c1="ram", c2="sita", c3="hari"]

Remaining Cookies:[c2="sita", c3="hari"]

Empty CookieStore:[]
```

Faculty: Gopal Maharjan (mahrzan.gopal@gmail.com)