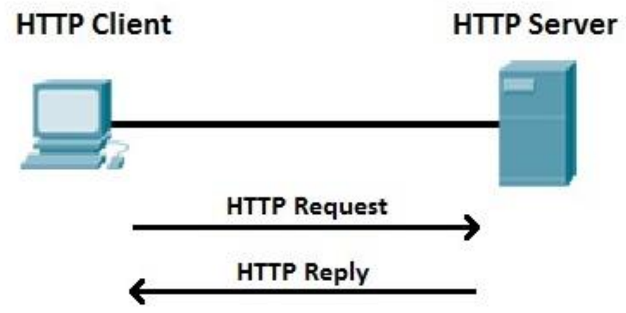
Unit-4

HTTP:

- Hypertext Transfer Protocol (HTTP) is a method for **encoding and transporting information between a client (such as a web browser) and a web server**. HTTP is the primary protocol for transmission of information across the Internet.
- It Establish communication between Client and Server.
- The primary or most commonly-used HTTP methods are **POST, GET, PUT, PATCH(partial modification), and DELETE**. These methods correspond to create, read, update, and delete (or CRUD) operations, respectively.
- HTTP is the set of rules for transferring files -- such as text, images, sound, video and other multimedia files -- over the web. As soon as a user opens their web browser

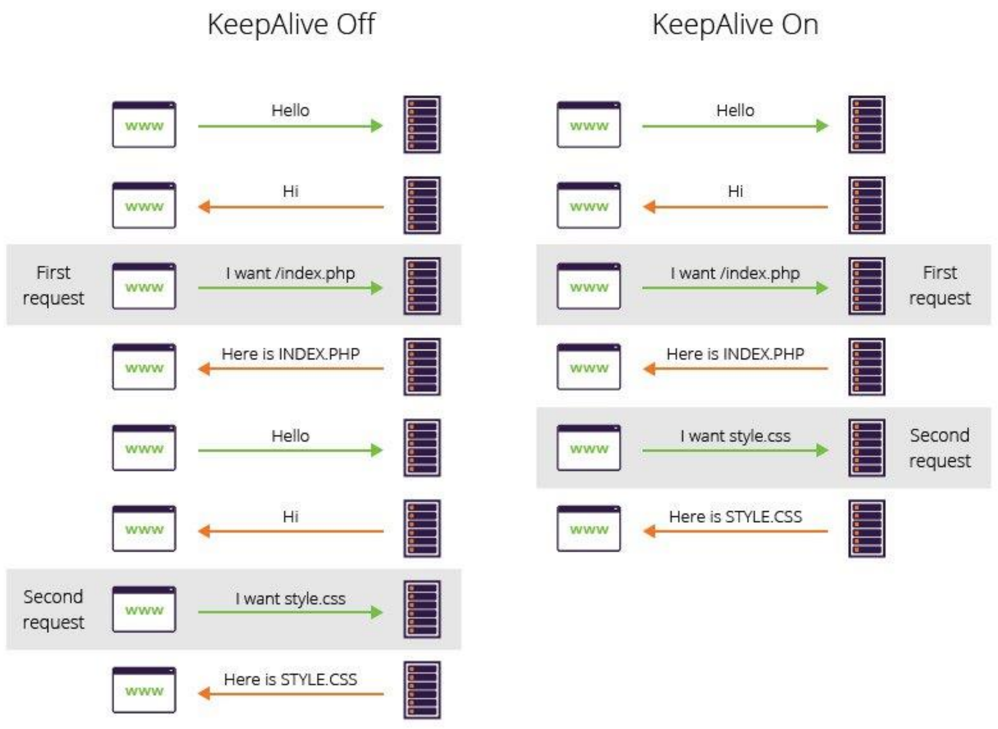


HTTP Keep alive Protocol:

- Creating multiple connections may reduce the loading time. It also utilizes many resources on the server.
- We can eventually overcome this issue and transfer all those files through a single connection by enabling the Keep-Alive, ***which avoids the need to repeatedly open and close a new connection.***
- ***If it is not enabled***, the process could take considerably longer to display the web page.
- Enabling the keep-alive header allows you to serve all web page resources over a single connection. ***Keep-alive also reduces both CPU and memory usage on your server.***

Benefits of Keep-Alive

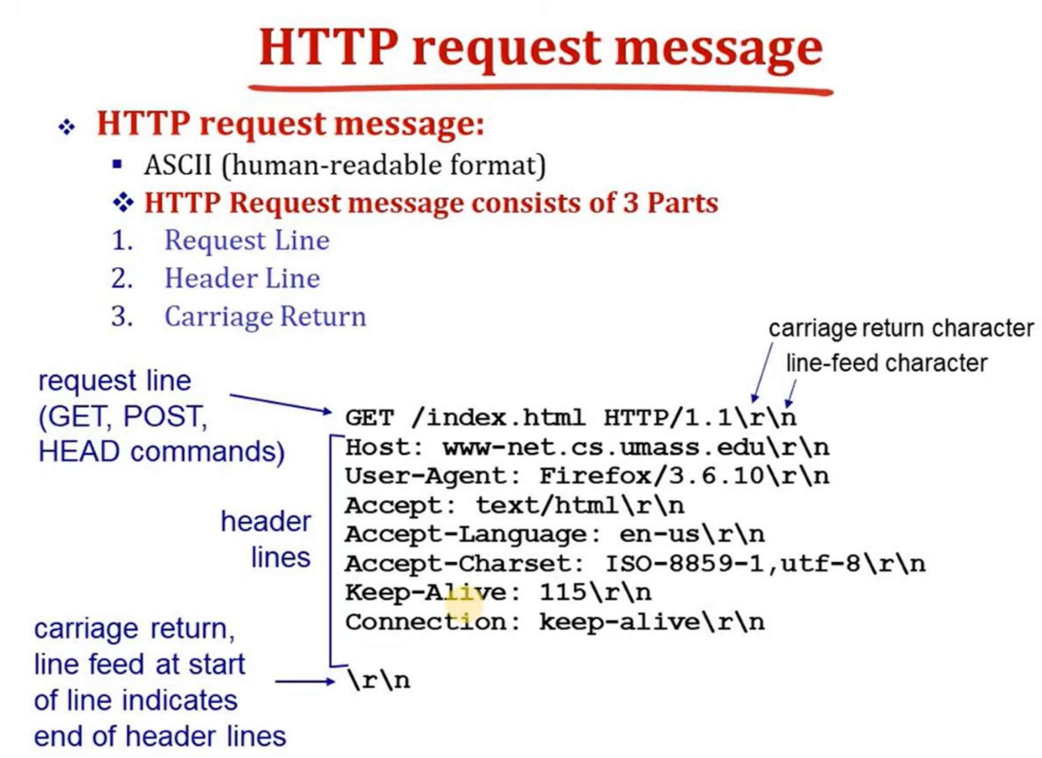
1. **Reduced CPU Usage**
2. **Web page speed:**



Java HTTP Keep Alive Properties:

1. **KeepAlive**
   Use *"KeepAlive On"* to enable it.
   To disable, just use *"KeepAlive Off"*.
2. Set **http.maxConnections** to the number of sockets you're willing to hold open at one time. The default is 5

HTTP Method:



HTTP request message

- **HTTP request message:**
  - ASCII (human-readable format)
  - **HTTP Request message consists of 3 Parts**
    1. Request Line
    2. Header Line
    3. Carriage Return

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html\r\n
Accept-Language: en-us\r\n
Accept-Charset: ISO-8859-1,utf-8\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return character
line-feed character
request line (GET, POST, HEAD commands)
header lines
carriage return, line feed at start of line indicates end of header lines

Method in Request Lines

- GET:get the data from a resource
- PUT:update data at a resource
- POST:to create data at a resource
- DELETE: to delete data at a resource
- HEAD: to partially update data at a resource

Request Lines has 3 fields

- Method Field
- Url Field
- HTTP Version Field

The method field can ta on several different values, including GET, POST, HEAD, PUT and DELETE

In the figures, the browser is requesting the object /index.html and version is self-explanatory; in the example, the browser implements version HTTP/1.1

COOKIES:

- Many websites use small strings of text known as cookies to store persistent client-side state between connections.
- Cookies are passed from server to client and back again in the HTTP headers of requests and responses.
- Cookies can be used by a server to indicate session IDs, shopping cart contents, login credentials, user preferences, and more.

To set a cookie in a browser, the server includes a Set-Cookie header line in the HTTP header. For example, this HTTP header sets the cookie "cart" to the value "ATVPD- KIKX0DER":

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: cart=ATVPDKIKX0DER
```

Servers can set more than one cookie. For example, a request I just made to Amazon fed my browser five cookies:

```
Set-Cookie:skin=noskin
Set-Cookie:ubid-main=176-5578236-9590213
Set-Cookie:session-token=Zg6afPNqbaMv2WmYFOv57zCU1O6Ktr
Set-Cookie:session-id-time=2082787201l
Set-Cookie:session-id=187-4969589-3049309
```

There are two types of Cookies:

Persistence: It is valid for multiple session. It is not removed each time where user closes the browser. It is removed only if user logout or signout.

Non Persistence: It is valid for single session only. It removed each time when user closes the browser.

CookieManager:

- Java 5 includes an abstract **java.net.CookieHandler** class that defines an API for **storing** and **retrieving** cookies.
- **CookieManager** adds the cookies to **CookieStore** for every HTTP Response and retrieves cookies from the **CookieStore** for every HTTP request
- To create the default *CookieHandler* and set it as the system-wide default:

    -
      ```
      CookieManager cm = new CookieManager();
      CookieHandler.setDefault(cm);
      ```

Method in Cookie Manager

- ***setCookiePolicy***

  public void setCookiePolicy(CookiePolicy cookiePolicy)

  To set the cookie policy of this cookie manager.

  A instance of CookieManager will have cookie policy ACCEPT_ORIGINAL_SERVER by default. Users always can call this method to set another cookie policy.

  Parameters:
  cookiePolicy - the cookie policy. Can be null, which has no effects on current cookie policy.

- ***CookiePolicy.ACCEPT_ORIGINAL_SERVER*** – Accepts only cookies from the original server. (the default implementation)
- ***CookiePolicy.ACCEPT_ALL*** – accept all cookies
- ***CookiePolicy.ACCEPT_NONE*** – reject all cookies

- ***getCookieStore***

  public CookieStore getCookieStore()

  To retrieve current cookie store.

  Returns:

  the cookie store currently used by cookie manager.

Lab: Write a program to handle HTTP cookies in Java using the CookieManager and HttpCookie classes also retrieve and display cookie information from a specified URL.

```java
import java.net.*;
import java.util.*;
public class App
    {
      private final static String URL_STRING = " https://hamrobazaar.com/";
    public static void main(String[] args) throws Exception {
      CookieManager cookieManager = new CookieManager();
      CookieHandler.setDefault(cookieManager);
      URL url = new URL(URL_STRING);
      URLConnection connection = url.openConnection();
      connection.getContent();
      CookieStore cookieStore = cookieManager.getCookieStore();
      List<HttpCookie> cookieList = cookieStore.getCookies();
      // iterate HttpCookie object
      for (HttpCookie cookie : cookieList)
      {
          // gets domain set for the cookie
          System.out.println("Domain: " + cookie.getDomain());
          // gets max age of the cookie
          System.out.println("max age: " + cookie.getMaxAge());
          // gets name cookie
          System.out.println("name of cookie: " + cookie.getName());
          // gets path of the server
          System.out.println("server path: " + cookie.getPath());
          // gets boolean if cookie is being sent with secure protocol
          System.out.println("is cookie secure: " +
cookie.getSecure());
          // gets the value of the cookie
          System.out.println("value of cookie: " + cookie.getValue());
          // gets the version of the protocol with which the given cookie is
related.
          System.out.println("value of cookie version: " + cookie.getVersion());

      }

    }
}
```

Cookie Store:

A **CookieStore** is an interface in Java that is a storage area for cookies. It is used to store and retrieve cookies.

The **CookieManager** adds the cookies to the **CookieStore** for every incoming HTTP response by calling **CookieStore.add()** and retrieves the cookies from the CookieStore for every outgoing HTTP request by calling **CookieStore.get().**

Method	Description
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

Lab: Write a java program to manage HTTP cookies using the **CookieStore** and **HttpCookie** classes also add, retrieve, and remove cookies from a **CookieStore**.

Example:

```
import java.io.*;
import java.net.*;

public class App {
    private final static String URL_STRING =
"http://www.samriddhicollege.edu.np";

    public static void main(String[] args) throws IOException {
        // CookieManager and CookieStore
        CookieManager cookieManager = new CookieManager();
        CookieStore cookieStore = cookieManager.getCookieStore();
        // Creating cookies and URI
        HttpCookie cookieA = new HttpCookie("First", "1");
        HttpCookie cookieB = new HttpCookie("Second", "2");
        // Setting additional cookie attributes
        cookieA.setMaxAge(3600); // 1 hour
        cookieB.setSecure(true); // Secure cookie
        URI uri = URI.create(URL_STRING);

        // Method 1 - add(URI uri, HttpCookie cookie)
        cookieStore.add(uri, cookieA);
        cookieStore.add(null, cookieB);
        System.out.println("Cookies successfully added\n");

        // Method 2 - get(URI uri)
        var cookiesWithURI = cookieStore.get(uri);
        System.out.println("Cookies associated with URI in CookieStore: " +
cookiesWithURI + "\n");

        // Method 3 - getCookies()
        var cookieList = cookieStore.getCookies();
        System.out.println("Cookies in CookieStore: " + cookieList + "\n");

        // Method 4 - getURIs()
        var uriList = cookieStore.getURIs();
        System.out.println("URIs in CookieStore: " + uriList + "\n");

        // Method 5 - remove(URI uri, HttpCookie cookie)
        System.out.println("Removal of Cookie: " + cookieStore.remove(uri,
cookieA));
        System.out.println("Remaining Cookies: " + cookieList + "\n");

        // Method 6 - removeAll()
        System.out.println("Removal all Cookies: " + cookieStore.removeAll());
        System.out.println("Empty CookieStore: " + cookieList);
    }
}
```

Output:

```
Cookies successfully added

Cookies associated with URI in CookieStore: [First="1"]

Cookies in CookieStore: [First="1", Second="2"]

URIs in CookieStore: [http://www.samriddhicollege.edu.np]

Removal of Cookie: true

Remaining Cookies: [Second="2"]

Removal of all Cookies: true

Empty CookieStore: []
```