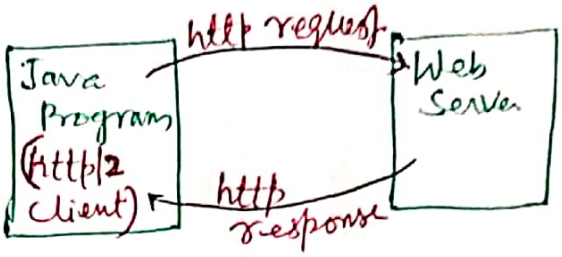
30-05-2020

Network Programming using Java (java.net package)

Any body wants to get any information from Internet network, he should send http request to the web server and get http response from the Web Servers.



In Java 1.8 version —
There is a class for getting connection through http server —
used from Java version 1.1 (1997) — legacy class. (Supports http/1.1 only)

HttpURLConnection

For Java 1.9 version — There is a new class API came which is supporting http/2.0 (2015) protocol.

→ http/1.1 supports only text but not binary data.
supports only one request at a time.

→ http/2.0 supports text and binary data with more than one requests at a time.

→ http/1.0 supports — synchronous mode communication. (Blocking mode)
(we do not wait for the response every time)

Problems with traditional HttpURLConnection class —

1. It is very difficult to use.
2. It supports only HTTP/1.1 protocol but not HTTP/2.0 (2015) where —
   (a) we can send only one request at a time per TCP connection, which creates network traffic problems and performance problems.
   (b) It supports only Text-data but not binary data
3. It works only in synchronous (blocking) mode, which creates performance problems.

Because of these problems, slowly developers started using 3rd party HttpClients like — Apache Httpclient and Google Http client etc.

JDK 9 Engineers addresses these issues and introduced a brand new HTTP/2 client in Java 9.

Advantages of Java 9 HTTP/2 client —

1. It is lightweight and very easy to use.
2. It supports both HTTP/1.1 and HTTP/2.0
3. It supports both text data and Binary data (Streams). We can send multiple requests at a time.
4. It can work in both Blocking and Nonblocking mode (Synchronous communication and Asynchronous communication)
5. It provides better performance in comparison to traditional HttpURLConnection (from JDK1.1 version)

Important Components of Java 9 HTTP/2 client —

Incubator module ((not fully developed ... but in process of development.))

① Module: java.incubator.httpClient

② Package: jdk.incubator.http

Three classes — { HttpClient, HttpRequest, HttpResponse }

— By using HttpClient — we can send a request to the web server.

— By using HttpRequest — we can perform the task of sending request to the web server

— By using HttpResponse — we can get response coming from the web server.

Name of Module — jdk.incubator.httpclient

Name of Package — jdk.incubator.http

Name of Classes — HttpClient, HttpRequest, HttpResponse.

Note:- Incubator module is by default not available to our java application. Hence compulsory we should read explicitly by using requires directive —

```
module demoModule
{
    requires jdk.incubator.httpclient;
}
```

Steps to send Http Request and procss HttpResponse from java application.

① Create HttpClient object

```
HttpClient client = HttpClient.newHttpClient();
```

② Create HttpRequest object

```
String url = "http://www.redbus.in"; //any URL
HttpRequest req = HttpRequest.newBuilder(new URI(url)).GET().build();
```

③ By using Httpclient object, send request and get HttpResponse.

```
client.send();
client.sendAsync();
```

④ Once the response is ready, process the HttpResponse.

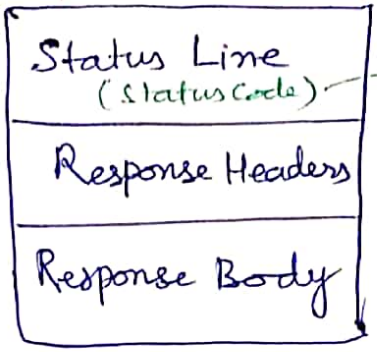
```
HttpResponse resp = client.send(req, HttpResponse.BodyHandlers.
                    asString());
```

or Sending to a file

```
HttpResponse resp = client.send(req, HttpResponse.BodyHandler.asFile(
                    Paths.get("abc.html")));
```

HttpResponse contains :- three parts

Structure of HttpResponse



1xx ⇒ informational Response
2xx ⇒ Successful Response
3xx ⇒ Re-direction Response
4xx ⇒ Client-error
5xx ⇒ Server-error

Three methods :- Process the HttpResponse using the three methods :-

```
resp.statusCode();
resp.body();
resp.headers();
```

Let us take an small example to summarize :-

```
HttpClient client = HttpClient.newHttpClient();
String url = "http://www.amazon.in";
HttpRequest req = HttpRequest.newBuilder(new URI(url)).GET().
                  build();
HttpResponse resp = client.send(req, HttpResponse.BodyHandler.
                    asString());
System.out.println("StatusCode:" + resp.statusCode());
System.out.println("Body:" + resp.body());
System.out.println("Response Code:" +
HttpHeaders headers = resp.headers();
Map<String, List<String>> map = headers.map();
map.forEach((k,v) -> System.out.println(k+"→"+v));
```

//Lamda Expression

These are the steps in java 1.9 to send http request to server and get Response from the server

That was very difficult in earlier versions using HttpURLConnection class.

Now, the total theory we have written in above pages can be converted to a single executable program as follows:—

```
package packA;
import jdk.incubator.http.HttpClient;
import jdk.incubator.http.HttpRequest;
import jdk.incubator.http.HttpResponse;
import jdk.incubator.http.HttpHeaders;
import java.net.URI;
import java.util.Map;
import java.util.List;
public class TestHttp {
    public static void main(String[] args) throws Exception{
        HttpClient client = HttpClient.newHttpClient();
        String url = "https://www.redbus.in/info/aboutus";
        HttpRequest req = HttpRequest.newBuilder(new URI(url)).
                          GET().build();
        HttpResponse resp = client.send(req, HttpResponse.BodyHandler.
                                        asString());
        System.out.println("Status code :"+ resp.statusCode());
//      System.out.println("Response Body :"+ resp.body());
        HttpHeaders header = resp.headers();
        Map<String, List<String>> map = header.map();
//      System.out.println("Response Headers:");
//      map.forEach((k,v)-> System.out.println("\t"+k+" :"+v));
    }
}
```

✡ First uncomment the ResponseHeaders then uncomment Response Body & see the results.

Compile and Run Commands : - In src folder →

Compile — javac --module-source-path src -d out -m demoModule

Run — java --module-path out -m demoModule/packA.TestHttp

Directory structure

D:\Java\src> tree /f

```
D:.
└── demoModule
    │   module-info.java
    │
    └── packA
            TestHttp.java
```

Compiling from D:\Java (directory only)

Now we want to write http response body to file abc.html.
For that purpose the response can be sended to a file as :-

```
HttpResponse resp = client.send(req, HttpResponse.BodyHandler.
                    asFile(Paths.get("abc.html")));
```

Paths is a class present in java.nio.file package and hence we should write import as

```
import java.nio.file.Paths;
```

In this case, abc.html file will be created in the current working directory which contains total response body. In next class we will modify current program to do the same.
