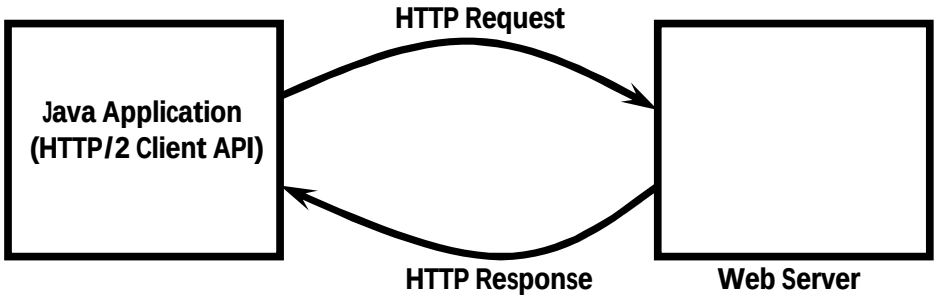
Java 9 HTTP/2 Client

What is the purpose of HTTP/2 Client:

HTTP/2 Client is one of the most exciting features, for which developers are waiting for long time. By using this new HTTP/2 Client, from Java application, we can send HTTP Request and we can process HTTP Response.



Prior to Java 9, we are using HttpURLConnection class to send HTTP Request and to Process HTTP Response. It is the legacy class which was introduced as the part of JDK 1.1 (1997).There are several problems with this HttpURLConnection class.

Problems with Traditional HttpURLConnection class:

1. It is very difficult to use.
2. It supports only HTTP/1.1 protocol but not HTTP/2(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 Blocking Mode (Synchronous Mode), which creates performance problems.

Because of these problems, slowly developers started using 3rd party Http Clients like Apache Http client 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.
3. It supports both Text data and Binary Data (Streams)
4. It can work in both Blocking and Non-Blocking Modes (Synchronous Communication and Asynchronous Communication)
5. It provides better performance and Scalability when compared with traditional HttpURLConnection.

etc...

Important Components of Java 9 HTTP/2 Client:

In Java 9, HTTP/2 Client provided as incubator module.

Module: jdk.incubator.httpclient
Package: jdk.incubator.http

Mainly 3 important classes are available:

1. HttpClient
2. HttpRequest
3. HttpResponse

Note:

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

```
1)  module demoModule
2)  {
3)    requires jdk.incubator.httpclient;
4)  }
```

Steps to send Http Request and process Http Response from Java Application:

1. Create HttpClient Object
2. Create HttpRequest object
3. Send HttpRequest by using HttpClient and Get the HttpResponse
4. Process HttpResponse

1. Creation of HttpClient object:

We can use HttpClient object to send HttpRequest to the web server. We can create HttpClient object by using factory method: newHttpClient()

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

2. Creation of HttpRequest object:

We can create HttpRequest object as follows:

```
String url="http://www.durgasoft.com";
HttpRequest req=HttpRequest.newBuilder(new URI(url)).GET().build();
```

Note:

newBuilder() method returns Builder object.
GET() method sets the request method of this builder to GET.
build() method builds and returns a HttpRequest.

public static HttpRequest.Builder newBuilder(URI uri)
public static HttpRequest.Builder GET()
public abstract HttpRequest build()

3.Send HttpRequest by using HttpClient and Get the HttpResponse:

HttpClient contains the following methods:
1. send() to send synchronous request(blocking mode)
2. sendAsync() to send Asynchronous Request(Non Blocking Mode)

Eg:

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

Note:

BodyHandler is a functional interface present inside HttpResponse. It can be used to handle body of HttpResponse.

4. Process HttpResponse:

HttpResponse contains the status code, response headers and body.

Status Line
Response Headers
Response Body

Structure of HTTP Response

HttpResponse class contains the following methods retrieve data from the response

1. statusCode()
Returns status code of the response
It may be (1XX,2XX,3XX,4XX,5XX)

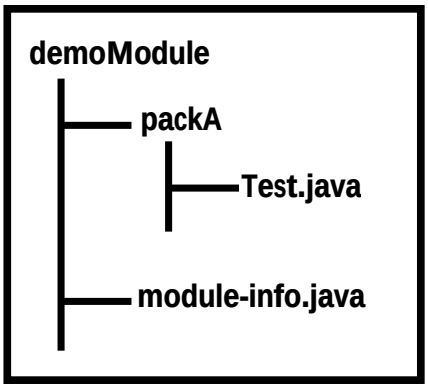
2. body()
Returns body of the response

3. headers()
Returns header information of the response

Eg:

```
System.out.println("Status Code:"+resp.statusCode());
System.out.println("Body:"+resp.body());
System.out.println("Response Headers Info");
HttpHeaders header=resp.headers();
Map<String,List<String>> map=header.map();
map.forEach((k,v)->System.out.println("\t"+k+":"+v));
```

Demo Program to send GET Request in Blocking Mode:



module-info.java:

```
1)  module demoModule
2)  {
3)     requires jdk.incubator.httpclient;
4)  }
```

Test.java:

```
1)  package packA;
2)  import jdk.incubator.http.HttpClient;
3)  import jdk.incubator.http.HttpRequest;
4)  import jdk.incubator.http.HttpResponse;
5)  import jdk.incubator.http.HttpHeaders;
6)  import java.net.URI;
7)  import java.util.Map;
8)  import java.util.List;
9)  public class Test
10) {
11)    public static void main(String[] args) throws Exception
12)    {
13)       String url="https://www.redbus.in/info/aboutus";
14)       sendGetSyncRequest(url);
15)    }
16)    public static void sendGetSyncRequest(String url)   throws Exception
17)    {
18)       HttpClient client=HttpClient.newHttpClient();
19)       HttpRequest req=HttpRequest.newBuilder(new  URI(url)).GET().build();
```

```
20)       HttpResponse resp=client.send(req,HttpResponse.BodyHandler.asString());
21)       processResponse(resp);
22)    }
23)    public static void processResponse(HttpResponse resp)
24)    {
25)       System.out.println("Status Code:"+resp.statusCode());
26)       System.out.println("Response Body:"+resp.body());
27)       HttpHeaders header=resp.headers();
28)       Map<String,List<String>> map=header.map();
29)       System.out.println("Response Headers");
30)       map.forEach((k,v)->System.out.println("\t"+k+":"+v));
31)    }
32) }
```

Note:

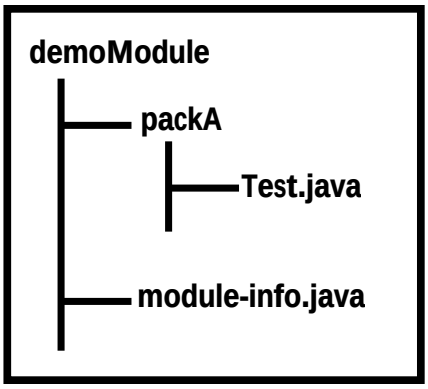
Writing Http Response body to file abc.html:

```
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.

Demo Program:



module-info.java:

```
1)  module demoModule
2)  {
3)     requires jdk.incubator.httpclient;
4)  }
```

Test.java:

```
1)  package packA;
2)  import jdk.incubator.http.HttpClient;
3)  import jdk.incubator.http.HttpRequest;
4)  import jdk.incubator.http.HttpResponse;
5)  import jdk.incubator.http.HttpHeaders;
6)  import java.net.URI;
7)  import java.util.Map;
8)  import java.util.List;
9)  import java.nio.file.Paths;
10) public class Test
11) {
12)    public static void main(String[] args) throws Exception
13)    {
14)       String url="https://www.redbus.in/info/aboutus";
15)       sendGetSyncRequest(url);
16)    }
17)    public static void sendGetSyncRequest(String url) throws Exception
18)    {
19)       HttpClient client=HttpClient.newHttpClient();
20)       HttpRequest req=HttpRequest.newBuilder(new  URI(url)).GET().build();
21)       HttpResponse resp=client.send(req,HttpResponse.BodyHandler.asFile(Paths.get("abc.
      html")));
22)       processResponse(resp);
23)    }
24)    public static void processResponse(HttpResponse resp)
25)    {
26)       System.out.println("Status Code:"+resp.statusCode());
27)       //System.out.println("Response Body:"+resp.body());
28)       HttpHeaders header=resp.headers();
29)       Map<String,List<String>> map=header.map();
30)       System.out.println("Response Headers");
31)       map.forEach((k,v)->System.out.println("\t"+k+":"+v));
32)    }
33) }
```

abc.html will be created in the current working directory. Open that file to see body of response.

Asynchronous Communication:

In Blocking Mode (Synchronous Mode), Once we send Http Request, we should wait until getting response. It creates performance problems.

But in Non-Blocking Mode (Asynchronous Mode), we are not required to wait until getting the response. We can continue our execution and later point of time we can use that HttpResponse once it is ready, so that performance of the system will be improved.

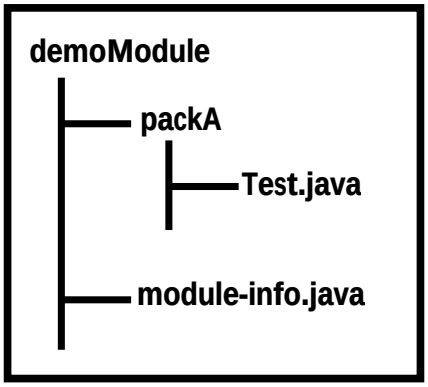
HttpClient class contains sendAync() method to send asynchronous request.

```
CompletableFuture<HttpResponse<String>> cf =
client.sendAsync(req,HttpResponse.BodyHandler.asString());
```

CompletableFuture Object can be used to hold HttpResponse in aynchronous communication. This class present in java.util.concurrent package.This class contains isDone() method to check whether processing completed or not.

```
public boolean isDone()
```

Demo Program For Asynchronous Communication:



module-info.java:

```
1)  module demoModule
2)  {
3)     requires jdk.incubator.httpclient;
4)  }
```

Test.java:

```
1)  package packA;
2)  import jdk.incubator.http.HttpClient;
3)  import jdk.incubator.http.HttpRequest;
4)  import jdk.incubator.http.HttpResponse;
5)  import jdk.incubator.http.HttpHeaders;
6)  import java.net.URI;
7)  import java.util.Map;
8)  import java.util.List;
9)  import java.util.concurrent.CompletableFuture;
10) public class Test
11) {
12)    public static void main(String[] args) throws Exception
13)    {
14)       String url="https://www.redbus.in/info/aboutus";
15)       sendGetAsyncRequest(url);
```

```
16)    }
17)    public static void sendGetAsyncRequest(String url) throws Exception
18)    {
19)       HttpClient client=HttpClient.newHttpClient();
20)       HttpRequest req=HttpRequest.newBuilder(new  URI(url)).GET().build();
21)       System.out.println("Sending Asynchronous Request...");
22)       CompletableFuture<HttpResponse<String>> cf = client.sendAsync(req,HttpResponse.B
      odyHandler.asString());
23)       int count=0;
24)       while(!cf.isDone())
25)       {
26)          System.out.println("Processing not done and doing other activity:"+ ++count);
27)       }
28)       processResponse(cf.get());
29)    }
30)    public static void processResponse(HttpResponse resp)
31)    {
32)       System.out.println("Status Code:"+resp.statusCode());
33)       //System.out.println("Response Body:"+resp.body());
34)       HttpHeaders header=resp.headers();
35)       Map<String,List<String>> map=header.map();
36)       System.out.println("Response Headers");
37)       map.forEach((k,v)->System.out.println("\t"+k+":"+v));
38)    }
39) }
```