4. Handling HTTP Requests and Responses

Contents

[1. Introduction 1](#_Toc6308)

[2. Reading HTTP Headers 1](#_Toc1519)

[3. Demo: Reading HTTP Headers 1](#_Toc16774)

[4. Setting HTTP Response Headers 2](#_Toc24333)

[5. Demo: Setting HTTP Response Headers 2](#_Toc17798)

[6. Setting HTTP Status Code 2](#_Toc1311)

[7. Demo: Setting HTTP Status Code 2](#_Toc7349)

[8. Summary 2](#_Toc1153)

# 1. Introduction

=>slides: Pg. 1-2

In this module, we shall understand the importance of HTTP request headers, HTTP response headers, and HTTP status code. We shall also observe how to use them practically within a Servlet application. Now let us get started.

# Reading HTTP Headers

=>slides: Pg. 3

Whenever we pass the data from the client to the server, then many additional information will be passed from the client to the server, which is set to be HTTP request headers, and this information will not be visible directly for the users, but this information helps a lot for the server to perform the task, and also having an idea on this information helps developers in developing the application more efficiently. HttpServletRequest object provides various methods to read HTTP header information.

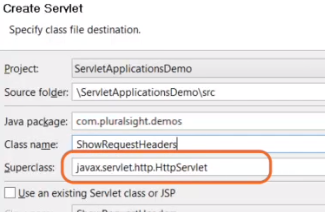
=>slides: Pg. 4

Now let us understand some of the important methods which help in reading HTTP headers. getHeader--this method accepts a string as an argument and returns the specified request header as a string. If the user doesn't provide a valid header name, then this method will return a null value. Usually, each header name appears only once in the request, but there is a possibility that a header can appear multiple times also with each occurrence having a separate value. Then this method returns the value of the first header in the request. If the user wants to get all the values, then we can use getHeader's method, which will return an enumeration of the values of all the occurrences of the header. getHeaderNames--this method returns all the header names present within the request as an enumeration. If the request doesn't have any headers, then this method will return an empty enumeration. getAuthType--this method returns the name of the authentication schema used to protect the Servlet. If the Servlet is not authenticated, then this method will return a null value. getRemoteUser--this method is used to return the name of the user making the request if the user has been authenticated. Else, this method will return a null value. getRemoteAddr--this method is used to return the IP address of the client or the last proxy address that sent a request. getContentLength--this method returns the value of the content length header as an int. And if the content length is not known, then this method will return -1. getContentType--this method returns the value of the content type header as a string. If the type is not known, then this method will return a null value. getDateHeader--this method accepts a string and returns the value of the specified request header as a long value that represents a date object where the date is written as a number of milliseconds since January 1, 1970, GMT. If the specified name is not a valid header, then this method will return -1. getIntHeader--this method also accepts a string and returns the value of the specified header as an integer. If the specified name is not a valid header, then this method will return -1. And if the specified string name as a header value cannot be converted to an integer, then this method throws a number format exception. In the next clip, let us understand two important concepts. First, let us observe how to display all the HTTP request headers using the code. And then we shall also understand how to use the HTTP request header information while working with Servlet application.

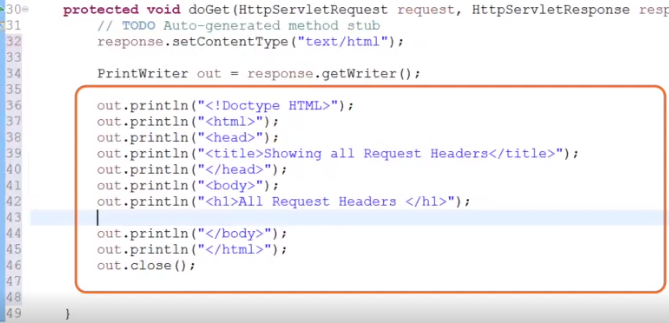
# Demo: Reading HTTP Headers

=>slides: Pg. 5

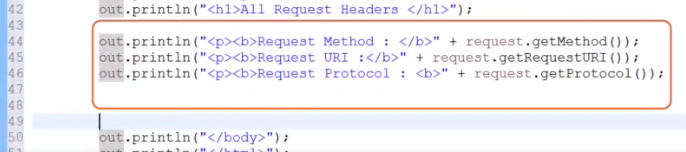
Now, let us create a Servlet page to read all the HTTP headers and display on the page. So let me first add a Servlet page.



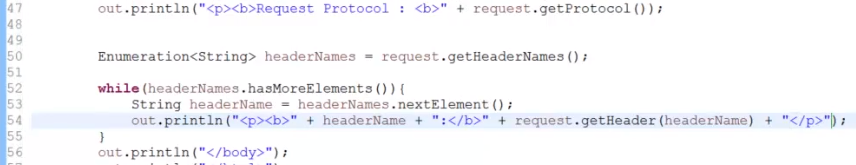
To do, I will right-click on the application and click on New, Servlet. Let us provide some meaningful package name, for example, com. dxc. demos. And then let me provide a class name, for example, ShowRequestHeaders. And let the superclass be HttpServlet itself. And click on the Finish button to create the Servlet page.



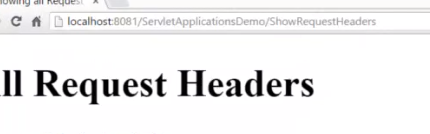
First, let me set the content type for the page. So let me type in response. setContentType of text/html. Let me define the PrintWriter object. PrintWriter out = response. getWriter. Now let me add some out statements to generate the basic structure of our HTML page. I have added the tags to generate the HTML, head, title, body, and a header for the page.



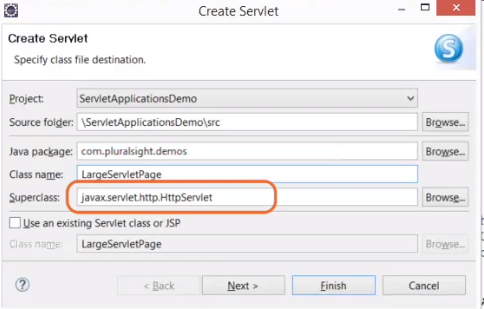
Now let me add some more out statements to get the HTTP method Request URI and the protocol. In order to read all the request header information, HttpServletRequest object provides us a method called getHeaderNames, which returns an enumeration of String.



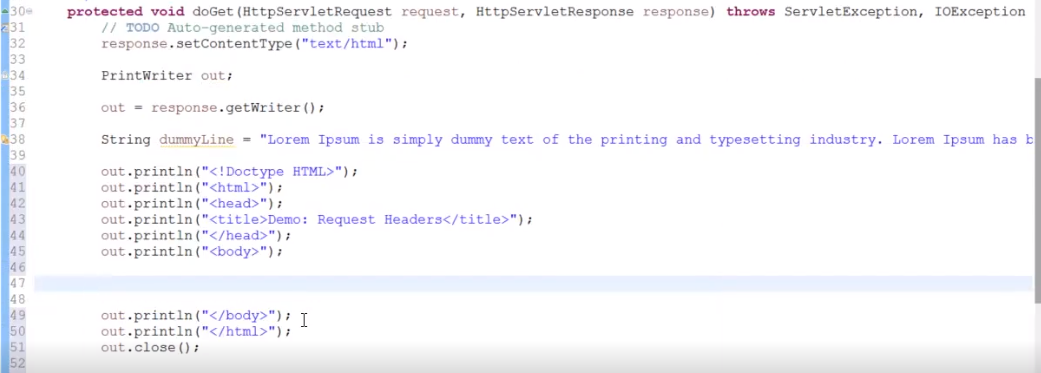
So let me type in Enumeration of String, headerNames = request. getHeaderNames. Once we have the enumeration, we can iterate the enumeration in the standard manner using hasMoreElements. So let me type in a while loop, while of headerNames. hasMoreElements within the loop. Now let us get the header name. So let me type in String headerName = headerNames. nextElement. And now let us display the header name and its value. To do, let me type in out. println, paragraph, bold, + headerName + we close the bold tag. And I want it to display the header value. So let me type in + request. getHeader of headerName. By the time the while loop completes its iteration, all the request headers will be displayed along with their values.



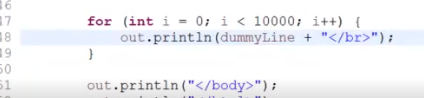
Now let me execute the Servlet page. We can observe the request method URL protocol, and also all the headers passed from the client to the server. Once we have an idea on the request headers, let us understand how to use these request headers for writing an efficient Servlet program. To do, let us understand how to send compressed pages to the client for efficient performance.



Let me add a Servlet page to display a huge information to the client. Let me right-click on application and click on New, Servlet. Let me provide the package name as com. dxc. demos, and the class name as LargeServletPage, and let the superclass be HttpServlet. And let me click on Finish button.



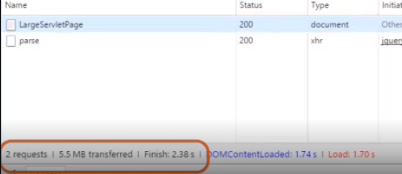
Within the doGet method, let me write a simple code to generate a large page. First, let me set the content type as HTML. So let me type in response. setContentType of text/html. And then let me define an object for the PrintWriter. So let me type in PrintWriter out, out = response. getWriter. Let me define a string variable to hold a paragraph. String dummyLine = some string. Let me add some out statements to generate the basic HTML structure.



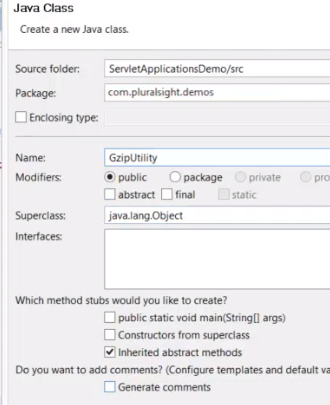
Within the body tag, let me add a for loop to display the dummyLine for 10, 000 times. So let me type in for (int i = 0; i is less than 10000; i++) out. println(dummyLine), and let me add a break tag. Now let me execute this Servlet page.



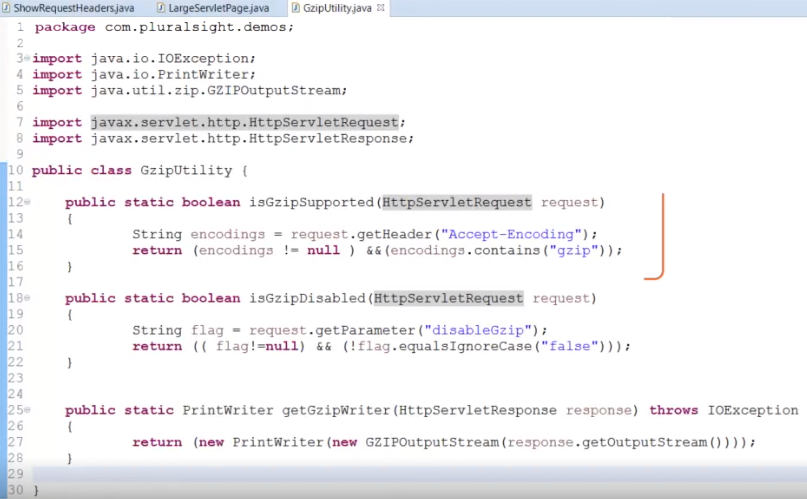
Let me open the Developer Tools, and let me click on the Network and refresh the page again.



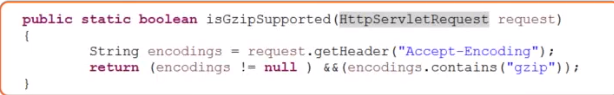
We can observe the amount of data transferred, load time, etc. Now let us improve the efficiency of this page with the support of HTTP request headers.



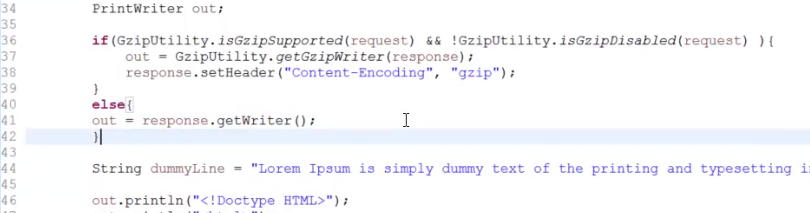
Now let me first add a class file to our application. So let me right-click on the application and click on New, Class, and let me provide the name of the package as com. dxc. demos, and the class name as GzipUtility, and click on Finish button.

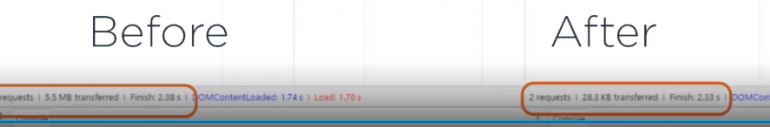


Now let me add some couple of static methods.



The first method is to verify if the Gzip encoding is supported by the browser or not. So in order to verify, I have written a method with the name isGzipSupported where I'm reading the HTTP request header information Accept-Encoding, and this method returns true if Gzip encoding is supported. Else, it returns false. The second method is to verify if Gzip encoding is enabled or disabled. To verify, I'm using an HTTP request header variable, disableGzip. And this method returns a Boolean value. The third method, getGzipWriter, is used to encode the regular HttpServletResponse and returns the PrintWriter object. Now once the utility class has been generated, now let us update the long Servlet code. If the Gzip encodings are supported, then we should encode the response using Gzip. Else, we should use the regular response.



To do, let me add an if condition. If(GZipUtility. isGzipSupported(request) &&! GzipUtility. isGzipDisabled(request)), then out = GzipUtility. getGzipWriter (response). Else, out = response. getWriter. Let me execute the long Servlet page, and let me open the Network and refresh the page. 

We can observe the size of the response content downloaded to the client is a lot less when compared with the size of the response content downloaded using the previous version of this code.

# Setting HTTP Response Headers

=>slides: Pg. 6

In the previous clip, we have observed the importance of HTTP request headers. Now let us understand about the response headers also. Whenever the web server response to an HTTP request, the response not only consists of the information, which is displayed on the browser, additionally, it will also send some information to the browser, which is called as the response headers. Response headers typically consist of a status length. Some response headers are \_\_\_\_\_ in the document. Now let us understand some important methods to set the HTTP response headers.

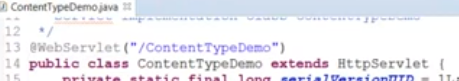
=>slides: Pg. 7

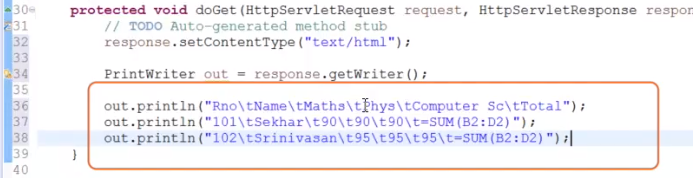
These methods are available with HttpServletResponse objects. setHeader--this is the most general way to set the response headers. This method takes two string arguments, the header name and the header values, and sets the response header with the given name and value. Headers of all types can be set with this method. setIntHeader--this method accepts two arguments, the header name and the integer value, and it can be used to set the response header, which requests an integer as its value. If the header has been set already, then the new value overrides the previous value. setDateHeader--this method can be used to set the response header, which requests the value as a timestamp. And this method accepts the date value as a long that represents the number of milliseconds since January 1, 1970, GMT. setContentType--this method accepts a string value, which represents the MIME type and is used to set the content type header of the response being sent to the client. And this method is used by the majority of Servlets, and we are also using this method from the beginning of our course. containsHeader--this method accepts a string value and provides a way to verify if a header exists or not. And this method returns true if the header exists. Else, this method returns false. setContentLength--this method is used to set the length of the content body in the response in HttpServlet. And this method sets the HTTP content length header. setLocale--this method sets the locale of the response if the response has been committed. This method can also be used to set the response character encoding for the locale if the character encoding has not been set explicitly using the setContentType method. In the next clip, let us understand how to use the HTTP response header information using a Servlet.

# Demo: Setting HTTP Response Headers

=>slides: Pg. 5

Now let us understand the advantage of setting up HTTP response variable.

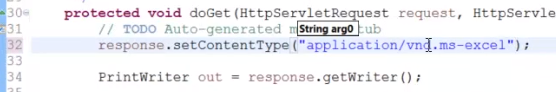


I already added a Servlet page. Now within the doGet method, as usual, let me first set the content type. 

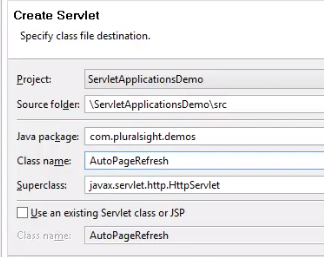
So let me type in response. setContentType("text/html"). And then let me create an object for the PrintWriter class. PrintWriter out = response. getWriter. Now let me add a couple of out statements to display the details of some students. Now let us execute the Servlet page.



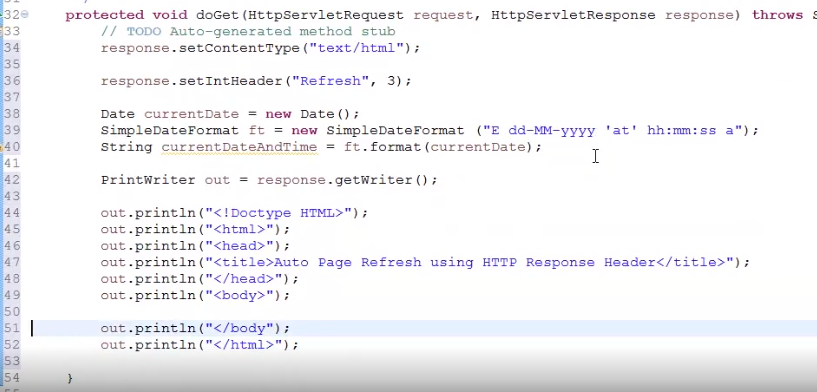
We can observe the output in the browser. Let me flip to Eclipse again



and update the content type. Let me replace text/html to application/vnd. ms-excel. Now I have set the response. setContentType as an Excel. Let us execute the Servlet page again. We can observe the output is displayed in an Excel sheet. As we have already seen, setContentType method working in previous examples also, let us observe it in one more example to understand the importance of HTTP response headers. With the support of set in the header, I'll set a refresh header. So let me add another Servlet page.



Let me right-click on the application again and click on New, Servlet. And let me provide the name of the package as com. dxc. demos and the class name as AutoPageRefresh. And click on Finish.



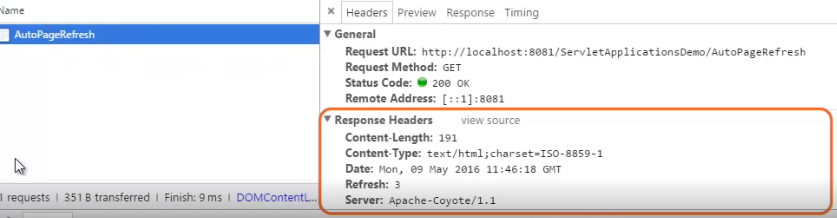
First, let me set the content type. So I type in response. setContentType("text/html"). I want the Servlet page to be reloaded every 3 seconds. So we need to set the HTTP response header refresh. To do, let me type in response. setIntHeader("Refresh", 3). To understand if the page is getting refreshed or not, let me define a variable to hold the current date and time. Now let me add some code. I have created an object for the date and SimpleDateFormat with the format day of the week, day, month, year, hours, minutes, and seconds, along with the am or pm marker. And then define a string variable currentDateAndTime to hold that formatted date and time value. Now let me create an object for the PrintWriter class. PrintWriter out = response. getWriter. Now let me add a couple of out statements to generate the basic HTML page.



Now within the body tag, let me add another out statement, out. println, paragraph, Page Last Refreshed at currentDateAndTime. Now let me execute the Servlet page.



We can observe the page is getting automatically reloaded for every 3 seconds. Now let me open the Developer Tools to verify if the response header refresh has been set or not.



We can observe the response header refresh has been set with the value 3. As we have understood the HTTP request and HTTP response headers, now in the next clip, we shall understand the importance of HTTP status code and how to set HTTP status code.

# Setting HTTP Status Code

=>slides: Pg. 9

HTTP status codes are the standard response codes given by the web server on the internet. These codes help to identify the cause of the problem whenever the page or the other resources do not load properly. There are five types of status codes available. The first one is 100 series, that is, information series, which provides the status codes such as 100 for Continue, which is used to inform the client that the initial part of the request has been received and has not been rejected by the server. 101 for Switching Protocol, which is used to inform that the server is waiting to comply with the claims request using the update message header field. The server will switch protocol to those defined by the responses upgrade header field. The protocol should be switched only when it is advantageous to do so. For example, switching to a newer version of HTTP is always advantageous over older versions. Second is 200 series, that is, success series, which indicates that the client's request was successfully received, understood, and accepted. 200 series provides various status codes such as 200 for OK. This is the standard response for successful HTTP request. The actual response will depend on the request method used, GET or POST, that is, for a GET request, the response will contain an entity corresponding to the requested resource, and for the POST request, the response will contain the result of the action. 201 - Created, which means that the request has been fulfilled and resulted in a new resource being created. 202 - Accepted, which means that the request has been accepted for processing, but the processing has not been completed. 203 - Non-Authoritative Information, which means that the server has successfully processed the request, but the returning information may be from another source. This status code is not present in HTTP 1. 0. It is only available from HTTP 1. 1. 204 - No Content, which means that the server successfully processed the request but is not returning any content. 205 - Reset Content, which means that the server has successfully processed the request, but it is not returning any content. But unlike 204 response, this requests that the requestor reset the document view. 206 - Partial Content, which means the server is delivering only part of the resource due to a range header set by the client. Third is the 300 series, which is called Redirection series, which indicates that further action needs to be taken by the user agent in order to fulfil the request. 300 series consists itself various status codes such as 300 - Multiple Choices. It represents a link list. A user can select a link and go to that location, a maximum of five addresses allowed. 301 - Moved Permanently, which means that the requested resource has been assigned a new permanent URL, and any further references to this resource should use the returned URL. 302 - Found, which means that the requested page has moved temporarily to a new URL. 303 - See Other, which means that the response to the request can be found under a different URL and should be retreived using the GET method on that resource. 305 - Use Proxy, which means that the requested resource must be accessed through the proxy given by the location field. 306 - Unused. It was used in a previous version of the specification. To specify the subsequent request should use the specific proxy. And these days, this is no longer used. 307 - Temporary Redirect. It is used to specify that the requested page has moved temporarily to a new URL. The fourth one is 400 series, which is caused because of client error. This series consists of various status codes such as 400 - Bad Request, which means that the server did not understand the request. 401 - Unauthorized, which means that the requested page needs username and password. 402 - Payment Required. This code is reserved for the future use. The intention of this code might be used as a part of some form of digital cache or micropayment scheme, but that has not happened. 403 - Forbidden. This means that the server has understood the request of the client, but the server is refusing to provide that response. 404 - File Not Found, which means that the server has not found anything which matches the requested URL, and also this code is commonly used whenever the server does not wish to reveal exactly why the request has been refused or when no other response is applicable. 405 - Method Not Allowed, which means that the method specified in the request line is not allowed for the resource identified by the request URL, for example, using a GET method on a form which requests data to be presented using the POST or PUT methods. 406 - Not Acceptable. This is to specify that the server can only generate a response that is not accepted by the client. 407 - Proxy Authentication Required. This is similar to 401, Unauthorized, but the difference is this code indicates that the client must first authenticate itself with a proxy. 408 - Request Timeout, which means that the client did not produce a request within the time that the server was prepared to wait. And the last one, the fifth, is 500 series, which is caused because of the server error. This series consists of the following status codes. 500 - Internal Server Error. This is to specify that the request was not completed as the server met unexpected exception. 501 - Not implemented. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource. 502 - Bad Gateway. This is to specify that the request was not completed. The server received an invalid response from the upstream server. 503 - Service Unavailable. This will be the code we will see when the server is temporarily overloading or down. 504 - Gateway Timeout. This code is returned when the server while acting as a gateway or proxy did not receive a timely response from the upstream server specified by the URL, for example, HTTP, FTP, LDAP, or some other auxiliary server, for example DNS. We have many other status codes other than these codes. I have mentioned some of the status codes, which are very commonly used.

=>slides: Pg. 10

Whenever the client sends a request, and if the server successfully received, understood, and accepted, web server will return the status code in the 200 series along with the response.

=>slides: Pg. 11

And whenever the client error occurs, the web server will return the status code in the 400 series.

=>slides: Pg. 12

And whenever the client sends a request to the server and while processing that request, if any exceptions are raised at the server site, then the web server will return the status code in the 500 series. So, having an idea on this HTTP status code will always be an advantage while writing the Servlet application. And in order to set the HTTP status code using the Servlet program,

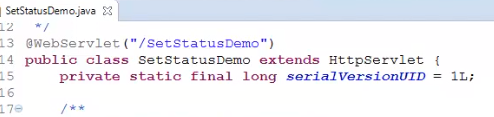
=>slides: Pg. 13

the following methods can be used. setStatus(int sc)--this method accepts the status code as the parameter, sets the status using that code. sendError(int sc, String errmsg)--this method accepts two arguments. The first one is the status code to be set, and the second argument is the error message to be displayed. And this method is used to send an error response to the client by using a specified status code. sendRedirect(String URL)--this method accepts the URL as a parameter and sends a temporary redirect response to the client based on the URL. This method makes the client browser create a new request to get the resource. The client can see the new URL in the browser. sendRedirect method accepts \_\_\_\_\_ URL, so it can call for resources inside or outside the server. In the next clip, let us understand how to use the above methods to set the HTTP status codes with demos.

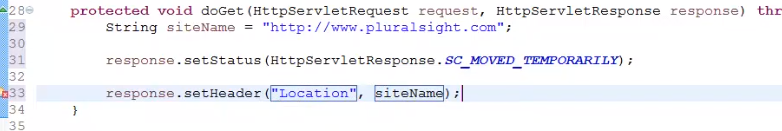
# Demo: Setting HTTP Status Code

=>slides: Pg. 14

Now let us understand how to use setStatus, sendRedirect, and sendError methods to set HTTP status codes.



I have already added a Servlet page with the name SetStatusDemo under com. dxc. demos package. In this demo, we shall understand how to set the status code 302 that has moved temporarily. So let me first define our variable to hold the site name where we want to redirect the user.



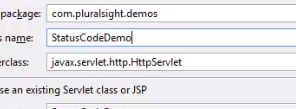
So let me type in String siteName = "http://www. dxc. com". HttpServlet response has provided mnemonic constant values as static fields to refer the HTTP status codes. So let me type in response. setStatus of, in order to use the static fields, we can either suppress the warnings of static access, or we can refer the members using that class name. So let me type HttpServletResponse. and select SC\_MOVED\_TEMPORARILY. In order to navigate to the specified site, let me set the location header, so let me type in response. setHeader("Location", "siteName"). Now, let me execute that Servlet page. We can observe the user has been navigated to dxc website. Whenever, we have a requirement such that we need to set the status code and also we need to redirect the user to another page. Then we can use sendRedirect method.



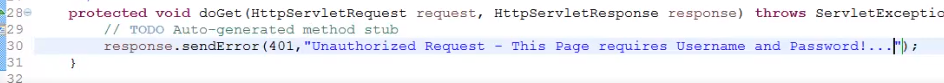
So let me flip to Eclipse again, and then let me comment the response. setStatus method and the response. setHeader method.



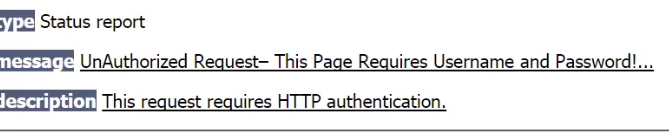
Now let me type in response. sendRedirect(siteName). Once again, let me execute the page. We can observe the user is getting redirected to the dxc website. Now let us understand with one more demo to set the HTTP status code using sendError.



Let me flip to Eclipse and add a Servlet page. So let me right-click on the application and click on New, Servlet, and provide the package name as com. dxc. demos, and the class name as StatusCodeDemo. And let the superclass be HttpServlet. And let me click on the Finish button.



Let us assume that this Servlet page should not be accessed without any authentication. If accessed directly, I would like to specify that this page requests authentication. So let me type in response. sendError(401, "Unauthorized Request - This Page requires Username and Password! "). Now let me execute this Servlet page.



We can observe the error status code has been displayed. And this helps in allowing the authorized users only to access the resource.

# Summary

=>slides: Pg. 15

In this module, we have understood what is the importance of HTTP request headers, HTTP response headers, and HTTP status code. And also we have understood how to use them effectively within our Servlet application. In the next module, we shall understand one of the most important features of Servlet programming-- intercepting HTTP requests with filters.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*