7. Tracking Session Data

Contents

[1. Introduction 1](#_Toc10662)

[2. Hidden Form Fields 1](#_Toc15355)

[3. URL Rewriting 1](#_Toc19325)

[4. Handling Cookies in Servlets 1](#_Toc18151)

[5. Demo: Handling Cookies in Servlets 2](#_Toc7822)

[6. Handling Sessions in Servlets 2](#_Toc8918)

[7. Demo: Handling Sessions in Servlets 2](#_Toc31088)

[8. Summary 2](#_Toc31680)

# 1. Introduction

=>slides: Pg. 1

In this module, we shall understand the importance of tracking the session data, and also we shall understand all the different methodologies to track the session data in Servlets.

=>slides: Pg. 2

We shall start with understanding the usage of hidden fields and how to use them in Servlets. Then we shall learn what is meant by URL rewriting, and how to use in Servlets for tracking the session data. Then we shall understand the most efficient methods for tracking the session data with the support of cookies and sessions. I will explain in detail about all the above methods, the pros and cons with proper demos in this module.

=>slides: Pg. 3

But before we start understanding the various methods to track the session data, we need to first understand why we need to track the session data.

=>slides: Pg. 4

So let us understand what is the requirement of tracking the session data. HTTP is a stateless protocol, which means that whenever a client sends a request to the web server, once the web server processes that instruction and provides the response to the client, the response may be a valid content requested by the user,

=>slides: Pg. 5

or it may be invalid.

=>slides: Pg. 6

But once the response is given to the client, then the server will not maintain any information about the client. Every time a client sends a request to the web server, a new connection will always be established with the server. Privacy advocates may consider this as a feature. But when it comes to the web application development, this creates a whole lot of problems because we won't be developing the entire web application in a single Servlet.

=>slides: Pg. 7

For example, in most of the web applications, we can observe that we will have a login page where we provide the username and password. And once the user credentials are validated, then we will be redirected to the default page of the website. And then whenever we navigate within the website, we can observe the user details

=>slides: Pg. 8

that we have provided at the login page that we will be using across the pages present within the website.

=>slides: Pg. 9

We usually observe this type of requirement in web applications such as an email application, shopping carts, chat applications, forums, etc. But I informed you earlier that HTTP protocol is a stateless protocol, and automatically it can't recognize that client.

=>slides: Pg. 10

So for that reason, we use a concept called as tracking session data in Servlets. We have various methods for tracking the session data--hidden fields, URL rewriting, cookies, and sessions. In the next clip, we shall understand how to use hidden fields to track the session data.

# Hidden Form Fields

=>slides: Pg. 11

In the previous clip, we got an idea why we need to track the session data. Now let us understand how to track the session data using hidden form fields. Hidden form field is an HTML form element which is used to store the value similar to the textbox element. But as the name suggests, hidden form field, this element can store the value, but it will not be visible on the page. But we can use a Servlet or any other server-side code to extract the value from the hidden form field.

=>slides: Pg. 12

Syntax, in order to define a hidden form field, will be input type="hidden" name="FormElementName" Value="Value to be maintained across the pages". In order to track the session data using an HTML hidden form field, we need to understand a simple process.

=>slides: Pg. 13

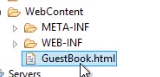
Let us assume we have an HTML page with the name Index. html where we accept some user input. And when the user clicks on Login, then the data should be submitted to the login Servlet page. Within the login Servlet page, the form element's data will be collected, request this process, and the Servlet page will generate the response, which will be given to the client as an output. But if the value of what we have accepted in the Index. html page, if it is required in the second Servlet page, for example, inbox Servlet, then it is mandatory to define an HTML hidden form field at the login Servlet such that the value can be accessed from the inbox Servlet. And the same process will continue if that value needs to be accessed from other Servlet pages like compose.

=>slides: Pg. 14

We need to remember one important point, that is, when we use HTML hidden form field, it is mandatory that in every Servlet page, we need to define an HTML hidden form field to maintain the value. This approach can be used if we have to submit the data in every Servlet page. Now let us understand how to use HTML hidden fields for tracking the session data using the demo.

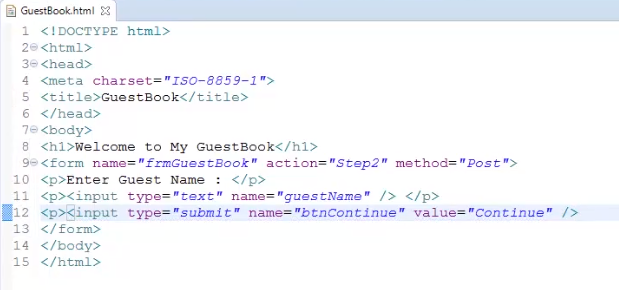
=>slides: Pg. 15

For this demo, I will be creating an HTML page with the name GuestBook where I like to collect the guest name and them submit that data to the Servlet, Step2. And in Servlet, Step2, I would like to collect the email Id. And once I've collected the email Id, again, we shall submit the data to the Preview Servlet, where I want to show the information that I provided in the GuestBook page and the Servlet, Step2. For tracking this information, we shall use the HTML hidden form fields. Now let us get started.



I already added an HTML page with the name GuestBook. html

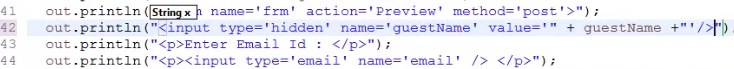


and created two Servlet pages--Step2 and the Preview. Now let me open the GuestBook. html page. 

First, let me update the title of the page. So let me type in GuestBook. Within the body tag, let me add a header to display Welcome to My GuestBook. We need a form for collecting the guest names. So let me type in form name="frmGuestBook" action="Step2" method="Post", and let me close the form tag. Within the form tag, let me add the code for collecting the guest name and submit the data. So let me type in paragraph Enter Guest Name : paragraph, input type="text" name="guestName", paragraph, input type="submit" name="btnContinue" value="Continue". Let us save the file.

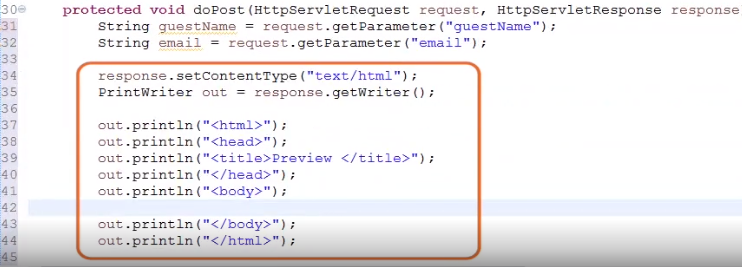


Now let me open the Step2 Servlet page. Within the doPost method, let me first collect the value from the GuestBook. html page. String guestName = request. getParameter("guestName"). Now let me set the response type as HTML. So I type in response. setContentType("text/html"). We need the PrintWriter object to display the information. So let me add some code to create an object for the PrintWriter class and a couple of out statements to generate a basic HTML page. Within the body tag, let me display a welcome message, out. println("Welcome" + guestName. Then I want to collect the guest email Id, so I require a form and an input element for accepting the email Id. So let me add the code. Now before we continue further, we need to recollect that if I submit the data from this Servlet, the value of what we have accepted from the GuestBook will not be available for the Preview Servlet.

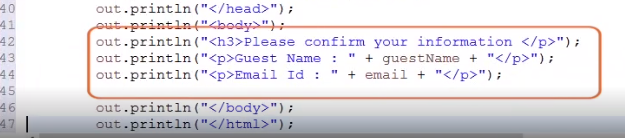


So we need to have an HTML hidden field for passing the guest name to the Servlet Preview. So within the form tag, let me type in out. println("input type='hidden') name='guestName' value= "+ guestName. Let us save this Servlet page.

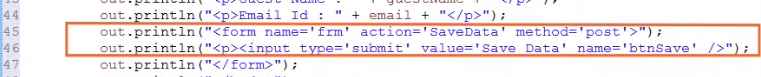
Now let me open that Preview Servlet page.

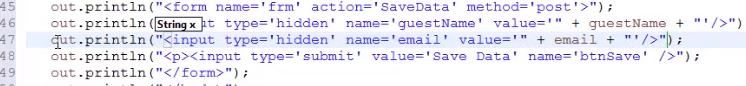


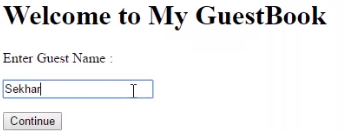
First, let me collect the values. So I type in String guestName = request. getParameter("guestName"), String email = request. getParameter("email"). Now let me add some code to set the content type. Create an object for the PrintWriter class and a couple of out statements to generate that response content.



Within the body tag, let me add some out statements to display the guest name and the email Id of the guest. Now assume that we need to submit this data from the Servlet to Save Data Servlet for storing the information permanently. For that, we require a form tag.

So let me add some code to generate the form tag and also to generate a Submit button. Now if we leave this Servlet like this, then within that same data Servlet page, we will not be able to access the guest name and the email Id.

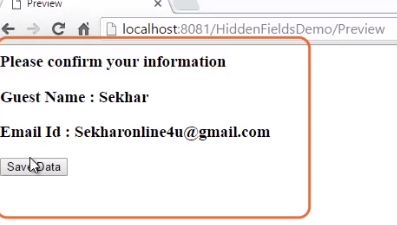
So within the form tag, we need to again create the HTML hidden fields for storing the guest name and the email Id. So let me type in out. println("input type='hidden' name='guestName' value='" + guestName"'), out. println("input type='hidden' name='email' value='" + email"'. Let me save that Servlet page. I'm not creating the SaveData Servlet page for this demo. The intention of using the form tag with action SaveData is to mention that if we need to carry the value across the Servlet pages, then it is mandatory to define the HTML hidden form fields to maintain the data in all the Servlet pages.



Now let me flip to GuestBook. html page and execute the page. Let me type in the guest name as Sekhar and click on Continue. We can observe the welcome message.



Now let me type in the email Id. For example, let me type in sekharonline4u@gmail. com. And then let me click on Preview button.



We can observe the values that have been provided in the GuestBook. html and the Step2 Servlet are still accessible.

=>slides: Pg. 16

Now let us understand the pros and cons of using HTML hidden form fields. The advantage we have is hidden fields are supported by all browsers. We don't have to depend on any browser capabilities. That is, irrespective of web browser settings of the client system, the application will work. But what the problem in using HTML hidden form fields is it is complex to implement. It is mandatory to submit the data to each Servlet page. That is, in every Servlet page, the HTML hidden field has to be defined. Else, the value will not be available for the next Servlet page. In the next clip, we shall understand the other alternate method--URL rewriting for tracking the session data.

# URL Rewriting

=>slides: Pg. 17

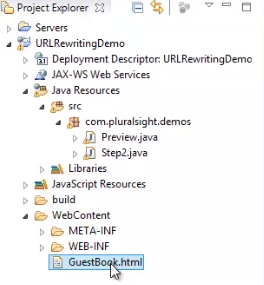
In the last clip, we have understood how to use hidden fields for tracking the session data. Now let us understand what is URL rewriting, and then we shall understand how to use URL rewriting for tracking the session data. So let us start with understanding what is URL rewriting.

=>slides: Pg. 18

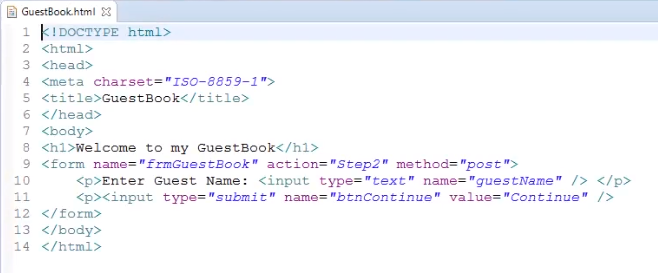
URL rewriting allows the user to append additional information which is required across the Servlet pages at the end of the URL where the value appended at the URL can be considered as a query string data, and it can be accessed from the Servlet page.

=>slides: Pg. 19

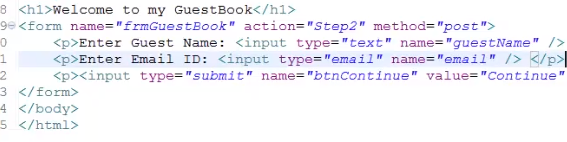
One important point that we have to remember is the data to be used across the Servlet's pages should always be appended with a URL by rewriting the URL dynamically with the values to be passed to the other Servlet pages. Now let us understand how to use URL rewriting for tracking the session data. We shall use the GuestBook example for understanding this concept.



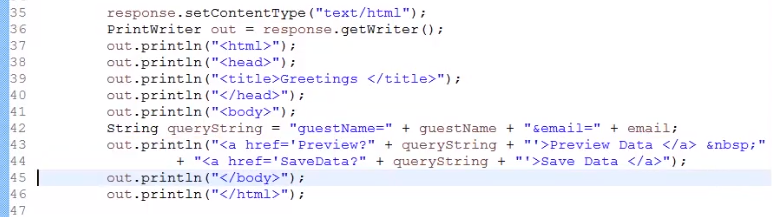
I have created a new project and added the GuestBook. html file and the two Servlets--Step2 and the Preview. There won't be much change in the code of GuestBook file.



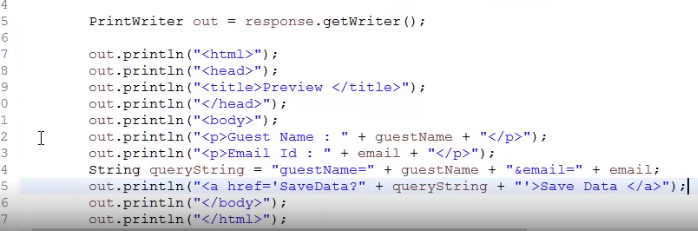
So I've taken the liberty to copy the code from the GuestBook. html file from hidden fields application and pasted it within the GuestBook. html file of our URL rewriting application. We can observe there is a form element to accept the guest name and a Submit button for submitting the data to the Step2 Servlet. Additionally, I would like to collect the email Id from the user. So let me add the input element for accepting the email Id.



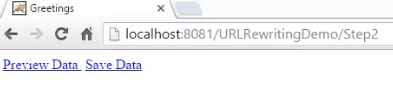
Let me type in paragraph, Enter Email ID:, input type ="email" name="email", and let me close that paragraph.



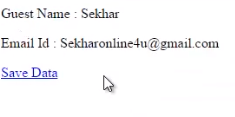
Now let me open that Step2 Servlet. First, let me add the code to collect the values of the guestName and the email Id. Now let me add the code to set the content type of the response and create an object for the PrintWriter class and create a couple of out statements to generate the response content. Now let me first prepare the query string to be passed to the service. So let me type in queryString = "guestName=" + guestName + "&email=" + email. Now let me add the out statement to generate the hyperlinks for our Preview Data and Save Data. We get two hyperlinks, one to navigate for Preview Servlet and the other to navigate for SaveData Servlet. We can observe that the URL has been dynamically updated with the values to be passed to the Servlet page. Let me save the file and open the Preview Servlet page.



First, let me collect the values of the guest name and the email Id. So let me type in String guestName = request. getParameter("guestName"), String email = request. getParameter("email"). Now let me add the code to set the response content type and create a PrintWriter object and a couple of out statements to generate the response content. Within the body tag, let me add the code to display the values of the guest name and the email Id. Now let me generate the query string and add an out statement to generate the hyperlink by rewriting the URL. Let me save the Servlet page. I won't be creating the SaveData page for this demo. But the reason I added the hyperlink in this Servlet page is to remind us that whenever we need that data across the Servlet pages, it is mandatory to rewrite the URL dynamically with the required data.



Let me open the GuestBook. html page, and let me execute the page. Now let me provide the guest name as Sekhar and the email Id as sekharonline4u@gmail. com. And then let me click on Continue button. Now we can observe this Step2 where we can see the two hyperlinks, one for the Preview data and the other to save the data.



Let me click on the Preview data link. We can observe that the values of what we have provided within the GuestBook. html page are accessible from the Preview page.

And also we can observe the URL, which has been rewritten.

=>slides: Pg. 20

Now let us understand the pros and cons of URL rewriting. The advantage of the URL rewriting concept is like HTML form fields, even URL rewriting is supported by all the web browsers. It doesn't depend on web browser capabilities. That is, if the web browser is enabled or disabled with cookies, the website will function properly. I will explain in detail what is meant by cookies in the next clip. But as of now, just remember the URL rewriting concept will work even when browsers don't support cookies. And what does the limitations of URL rewriting mean? URL rewriting concept works only with the hyperlinks. Using URL rewriting we can submit only text information. Whenever we have to pass the data across multiple Servlets, then implementing URL rewriting may be complex because all the URLs have to be updated with the data. In the next clip, let us understand one of the efficient methods to track the session data with the support of cookies.

# Handling Cookies in Servlets

=>slides: Pg. 21

In the last two clips, we have understood how to track the session data using HTML hidden fields and the URL rewriting method. Both of these methods are a bit complex to implement, especially when the data has to be accessed across many Servlet pages.

=>slides: Pg. 22

Now let us understand one of the efficient ways to track the session data with the support of cookies. Cookies are used to store the server-side information at the client system such that the values can be used across the Servlet pages. That is, once we set the cookies in one Servlet page, then that value can be used from other Servlet pages present within the application. We don't have to pass the cookie information explicitly to every Servlet page.

=>slides: Pg. 23

Cookies can be categorized into two types based on the expire parameter, session cookies or temporary cookies, and the other one is persistent cookies or permanent cookies. Whenever we use a session cookie, then the cookie information may be maintained within the memory allocated at the web browser, and the cookie data will be cleared implicitly when the user closes the browser. Persistent cookies maintain the cookie data within a physical file at the client system until the expiration parameter of the cookie.

=>slides: Pg. 24

Now let us understand how the cookies work. Whenever a client sends a request to the Servlet page for the first time, then the Servlet page processes the request

=>slides: Pg. 25

and creates the cookie which will be passed through the client using the response headers along with the Servlet HTTP response. Then the cookie will be stored at the client's browser or at the physical file presented at the client system.

=>slides: Pg. 26

And for a further request, the cookie which has been stored at the client will be passed to the server using the request headers, which will be used by the Servlet page for tracking the user session data.

=>slides: Pg. 27

To set the cookie, we need to first create an object for the cookie class, which exists within the package javax. servlet. http. Syntax will be Cookie cookieObject = new Cookie("key", "value"). When we create a cookie, then we can additionally set many different attributes for the cookies, such as a comment, path, domain, qualifiers, a maximum age, and a version number, etc. Whenever a cookie is created, by default, it will be considered as a session cookie or temporary cookie. And in order to set the expiration period for the cookie information, we can use setMaxAge(seconds) while defining the cookie. For example, to set the expiration period for the cookie, we can use the following syntax--cookie. setMaxAge(3000). Once the cookie is set with the expiration period, then it will be considered as a persistent cookie. And this cookie will expire after 3000 seconds. One important point we need to remember--if the value for the cookie is set to a negative value, for example, -1, then that cookie is not stored persistently and will be deleted when the web browser exits. If 0 is passed as the value, then the cookie will expire immediately. And if you provide a value greater than 0, then only it will be considered as a persistent cookie. Once we've created the object and set the attributes for the cookie, then we need to add the cookie object to the HttpServletResponse headers. To do, we need to use response. addCookie(cookieObject). A Servlet can retrieve the information about the cookie through the request methods, request. getCookies, which returns an array of cookie objects.

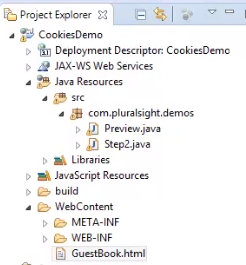
=>slides: Pg. 28

Additionally, we have a list of useful methods which we can use while manipulating cookies in Servlet. setDomain(String pattern)--this method sets the domain to which cookie applies, for example, Pluralsight. com. getDomain--this method gets the domain to which cookie applies. setMaxAge(int expiry)--this method sets how much time in seconds should elapse before the cookie expires. getMaxAge--this method returns the maximum age of the cookie specified in seconds. The default value will be -1 indicating the cookie will persist until the browser is shut down. getName--this method returns the name of the cookie. The name cannot be changed after the creation of the cookie. setValue(String newValue)--this method will set a new value associated with the cookie. getValue--this method gets the value associated with the cookie. setPath(String uri)--this method sets the path to which the cookie applies. If we don't specify a path, then the cookie will be available for all the URLs in the same directory as the current page, as well as the subdirectories. getPath--this method gets the path to which the cookie applies. setSecure(boolean flag)--this method sets the Boolean value indicating whether the cookie should be sent over secure connections, that is, SSL. setComment(String purpose)--this method specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user. getComment--this method returns the comment describing the purpose of the cookie or null if the cookie is defined with no comments. Now let us understand how to use cookies for tracking the session data in Servlets using the GuestBook demo.

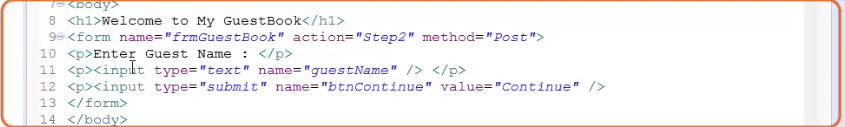
# Demo: Handling Cookies in Servlets

=>slides: Pg. 29

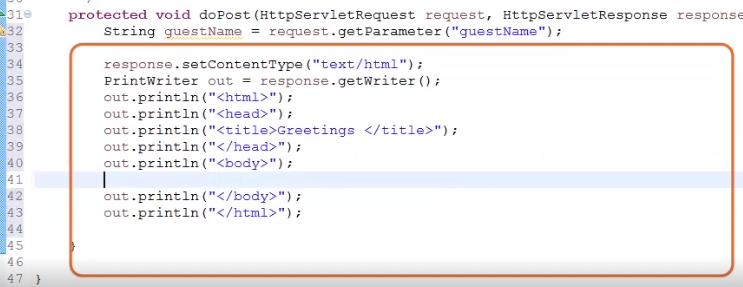
For this demo, we shall extend the same GuestBook application similar to the one that we have observed for understanding the HTML hidden form fields. That is, I will be creating an HTML page with the name GuestBook where I will collect the guest name and then submit that data to the Servlet Step2. Within that Step2 Servlet, I would like to collect the email Id. And once I've collected the email Id, then we shall submit the data to the Preview Servlet where we shall display the values collected from the GuestBook page and the Step2 Servlet page. And to understand once we set the cookies, we can access the cookie information from other Servlet pages, we shall add another Servlet page with the name SaveData where we shall collect the values and display the values with the support of cookies.



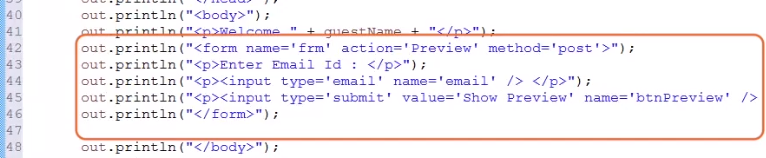
I have already created a new Servlet application and added GuestBook. html file and two Servlets--Step2 and Preview. I have taken the liberty to copy the code of GuestBook. html file, our hidden field Servlet application,



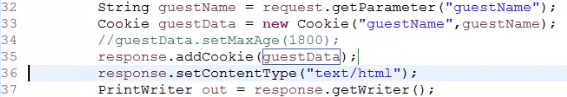
and pasted that code within this GuestBook. html file. We have a simple form with a textbox for accepting the guest name and then a Submit button to submit the data to Step2 Servlet. There is no change within the code, so let me open the Step2 Servlet page.

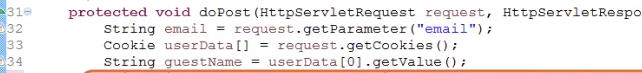


And first let me add the code to collect the guest name. Now let me add the code to set the response content type and object for the PrintWriter class and a couple of out statements to generate the response content. Within the body tag, let me write a welcome statement.

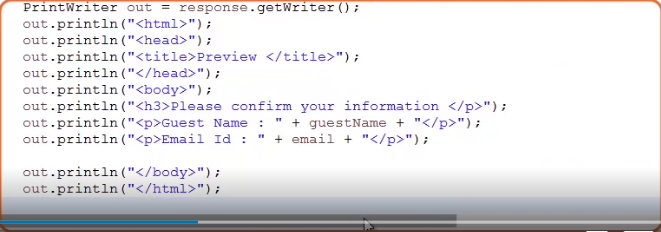


So let me type in out. println("Welcome" + guestName). I would like to collect the email Id of the guest, so let me add a couple of out statements to generate the form tag, an input element to collect the email Id, and a Submit button to submit the data. Now whenever we submit the data from the Step2 page to the Preview Servlet, then the data that we have provided in the GuestBook will not be accessible from the Preview Servlet, so we need to track the data. And for this purpose, we are now interested in using one of the most efficient methods for tracking the session data, that is, cookies. Now let me write the code to set the cookie with the guest name.



After the statement where I have collected the value of the guest name, let me type in Cookie guestData = new Cookie("guestName", guestName). If you want to make this cookie information as persistent, then we can set the max age for the guest name by typing guestData. setMaxAge(some seconds), for example, 1800. But for this demo, we don't require a persistent cookie, so let me comment this line. Once the cookie object is created, now we need to add that cookie to the response, so let me type in response. addCookie(guestData). Since we have not set the setMaxAge for the guestName cookie, it will be considered as a session cookie. Let me save the file, and let me open the Preview Servlet page. 

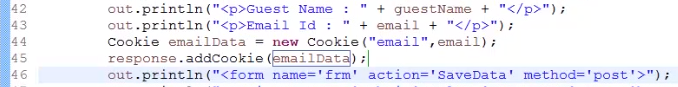
First, let me collect the value of the email Id, so let me type in String email = request. getParameter("email"). Now let us collect the values from the cookies where we have stored the information about the guest name. First, let me collect all the cookies with the support of request. getCookies method, which will return a cookie array object. So let me type in Cookie userData array = request. getCookies. Since we have only one cookie set to collect the values of guest name, I can type it in as String guestName = userData(0). getValue.



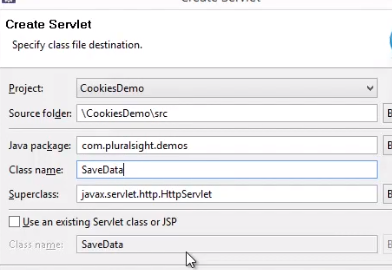
Now let me add the code to set the response content type and object for the PrintWriter class and a couple of out statements to generate the response content.



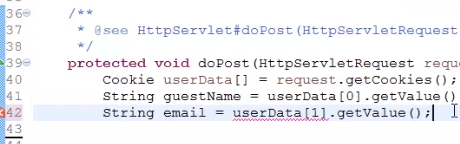
And now let me add the code for generating the form tag and Submit button to submit the data to the SaveData Servlet. From this page, if we wanted to submit the data to the SaveData Servlet, then the value that we have provided for the email should also be set with the cookies.



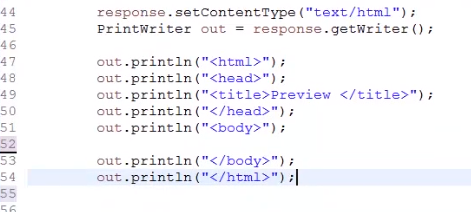
So let me add the code to set the cookie data for email. Let me type in Cookie emailData = new Cookie("email", email), response. addCookie(emailData). One important point I wanted to mention here is the browser is expected to support 20 cookies for each web server, 300 cookies total, and may limit the cookie size to 4 KB each. Let me save the file.



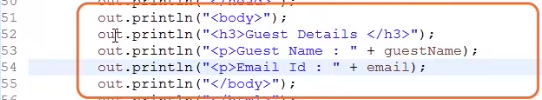
And now let me add another Servlet page with the name SaveData. Right-click on the application and click on New, Servlet, and let me provide the package name as com. Pluralsight. demos and the class name as SaveData, and let the superclass be HttpServlet itself, and let me click on Finish button to create the Servlet page. Now, first let us collect the guest name and the email Id from the cookies.



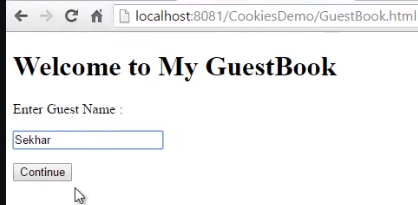
So let me type in Cookie userData array = request. getCookies, String guestName = userData(0). getValue, String email = userData(1). getValue.

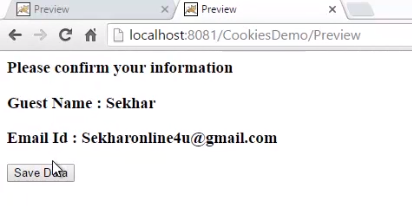


Now let me add the code to set the response content type, an object for the PrintWriter class, and then a couple of out statements to generate the response content.

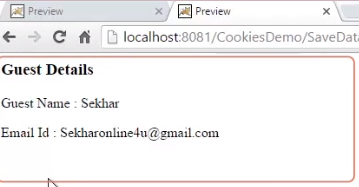


Within the body tag, let me add the code to display the guest name and the email Id. Save the Servlet page. If you are aware of interacting with the database using Java, then feel free to store this information within the table presented in that database.



Now let me flip to GuestBook. html page and execute the page. Let me provide the value for the guest name, for example, Sekhar, and let me click on Continue. We can observe a welcome message. And now let me type in the value for email Id, Sekharonline4u@gmail. com, and click on the Preview button. 

We can observe the guest details, that is, the guest name and the email Id. Now let me click on the Save Data button.



We can observe the values of the guest name provided at the GuestBook. html page and the email Id we provided at the Step2 Servlet.

=>slides: Pg. 30

Now let us understand the pros and cons of using cookies. The advantage of cookies will be it is very easy to implement. Since the cookie information is maintained at the client system, performance will also be very fast. We can store huge information for a longer period of time using the cookies by setting the cookie. setMaxAge method with a value. And when it comes to the limitations of cookies, cookies are browser dependent. That is, if we disable the cookies at the browser, then the web application which uses the cookies will not function properly. And also the cookies which have been sent by one web browser, other web browsers can't read that value. Since the cookie information is stored at the client system, it is not advisable to store any secure data within the cookie. Cookies can store only String information. In the next clip, we shall understand another alternative and one of the most favorite methods used by the Java developers for tracking the session data, that is, sessions.

# Handling Sessions in Servlets

=>slides: Pg. 31

We have discussed various methods to track the session data. Now let us understand how to track the user data with the support of sessions. HTTP sessions are considered as one of the best methods to track the session data. Sessions are used to maintain the user information at the server side temporarily such that the values can be accessed across the Servlets present within the web applications. Servlet provides HTTP session interface, which provides a way to identify a user across more than one page request or visit to a website and to store information about that user.

=>slides: Pg. 32

We can create an HTTP session object by calling the public method getSession of HttpServletRequest as HttpSession session = request. getSession. And one important point that we need to remember is that we need to call request. getSession before we send any content to the client.

=>slides: Pg. 33

Now let us understand how the session API concept works. Whenever a client sends a request for the first time, then the web container will create a unique identifier, which is also called as a SessionID for the client at the server and allocates some memory for maintaining the user information at the server. =>slides: Pg. 34

And once the processing is completed, then along with the HTTP response, the server will return the session Id for the client as an HTTP cookie. Once that client receives the response along with the session Id,

=>slides: Pg. 35

the next time if the client sends the request to the server, then always an HttpRequest will be submitted to the server along with the SessionID of the client. The web container uses the SessionID of the client, submits it to the server, and accesses the data present within the memory block, and processes the request, and provides the response based on the SessionID.

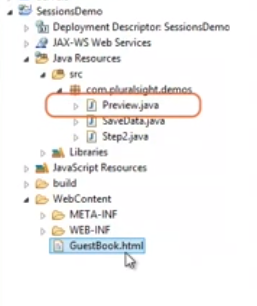
=>slides: Pg. 36

HTTP session object provides various important methods to track the session data. getAttribute(String name)--this method returns the object bound with the specified name in the session and returns null if no object is bound under that name. getAttributeNames--this method returns an enumeration of string objects containing the names of all the objects bound to this session. getCreationTime--this method returns the time when the session object was created, and the value will be measured in milliseconds. getId--this method returns a string containing the unique identifier assigned to this session. getLastAccessedTime--this method returns the last accessed time of the session in the format of milliseconds. getMaxInactiveInterval--this method returns the maximum time interval in seconds that the Servlet container will keep the session open between the client accesses. invalidate--this method invalidates the session and unbinds any objects bound to it. isNew--this method returns a Boolean value true if that client does not know about the session. Else, it returns false. removeAttribute(String name)--this method removes the object bound with the specified name from the session. setAttribute(String name, object value)--this method binds an object to this session using the name specified. setMaxInactiveInterval(int interval)--this method specifies the time in seconds between the client request before the Servlet container will invalidate the session. In the next clip, we shall understand how to track the session data using HTTP session with a proper demo.

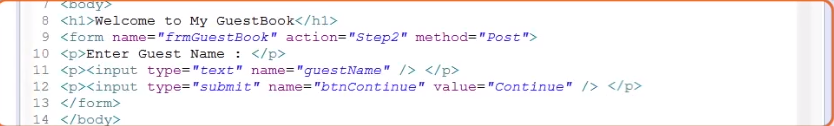
# Demo: Handling Sessions in Servlets

=>slides: Pg. 37

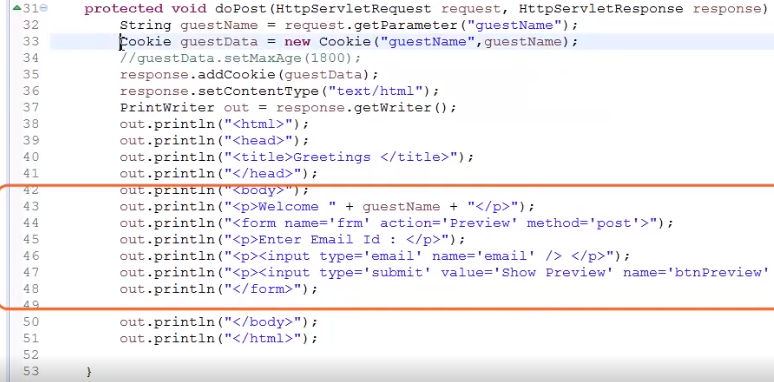
To understand how to track the session data using HTTP session, let us use the same GuestBook application.



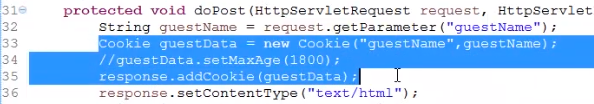
I already created the Servlet application and added the GuestBook. html and also added for this Step2, SaveData, and Preview. And also I have taken the liberty to update the GuestBook. html, Step2, SaveData, and Preview Servlet pages with the code present in the previous demos of cookies. Let me open the GuestBook. html.



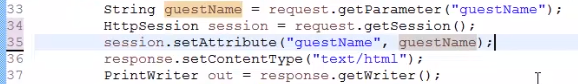
We can observe the code for generating a simple form with an input element textbox for collecting the guest name and a Submit button to submit the data to the Step2 Servlet.



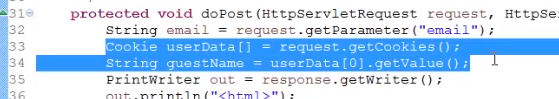
Now, let me open the code of Step2 Servlet page. We can observe the code for displaying the guest name and the form element to collect the value of the email Id and a Submit button to submit the data to the Preview Servlet.



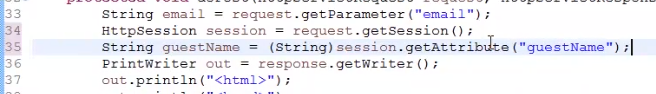
Let me remove the code which is used to set the cookies. Now for this demo, instead of cookies, we would like to use sessions.



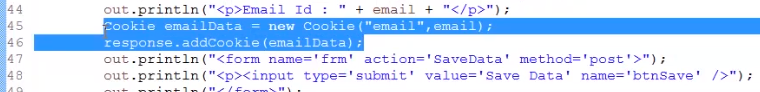
So let me type in HttpSession session = request. getSession, session. setAttribute("guestName", guestName). Now let me open the Preview Servlet page. Once again, let me remove the code written for cookies.



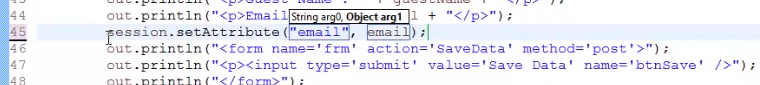
Now let me type in the code to extract the value from the session.



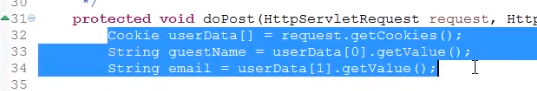
HttpSession session = request. getSession, String guestName = (String)session. getAttribute("guestName"). Since the getAttribute method returns an object, I have typecast it to string. Since we require the value of email Id in the next Servlet,



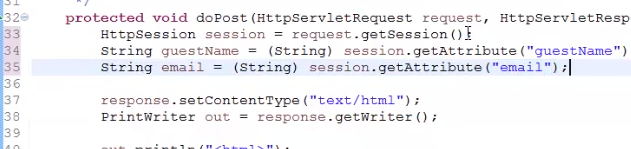
let me remove the code for cookie and use the session to set those values.



To do, let me type in session. setAttribute("email", email). Now let me open the SaveData Servlet.



First, let me remove the code of cookies, and let me type in the code to extract the values from the session.



HttpSession session = request. getSession, String guestName = (String) session. getAttribute("guestName"), String email = (String) session. getAttribute("email"). Let me save this Servlet and open GuestBook. html and execute that page. Let me provide the value for the guest name, for example, Sekhar. And then let me click on Continue. We can observe a welcome message. And now let me type in the value for email Id, Sekharonline4u@gmail. com and click on Preview button. We can observe the guest details, that is, the guest name and the email Id. Now let me click on Save Data button. We can observe the values of guest name provided at the GuestBook. html page and the email Id we provided at the Step2 Servlet.

=>slides: Pg. 38

Now let us understand what the pros and cons are of sessions. The advantage of the sessions is it is very easy to handle the data across the pages present within the website. Sessions can store not only string-type of data, it can also store objects as the value. Since session information is stored within the server side, the data will always be secure. When we observe the limitations, as the data is stored at the server side, it is not advisable to store the data for longer periods of time. And also it is not advisable to store huge data within the session. Like cookies, if the web browser has blocked the cookies, then the site will not function properly.

# Demo: Session Login and Logout

=>slides: Pg. 39

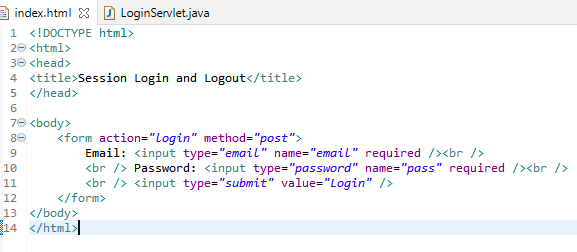
There are various ways to maintain session but here I will use ****HttpSession**** class.

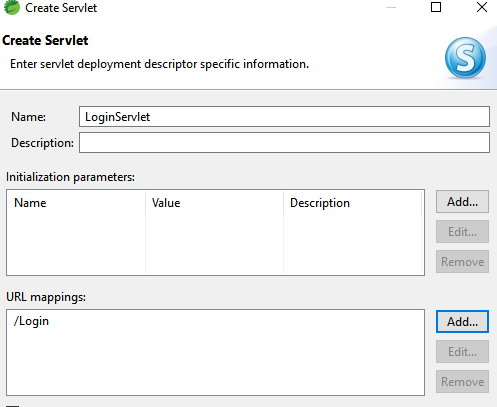
In this example we have a ****index.html**** page where a login form is displayed. When user enters login details and submits the form the request is sent to ****LoginServlet****. If the details are correct then user is redirected to ****HomeServlet****otherwise redirected to ****index.html****.

****LogoutServlet**** invalidates the sessions to logout the user and redirect to ****index.html****.

If anyone tries to access Home page directly without doing login then he/she will be redirected to ****index.html.****

In this example I have hard coded email and password but you can use database to compare login details. For doing successful login the email should be ****abc@gmail.com**** and password should be ****abc****.





@WebServlet("/login")

**public** **class** LoginServlet **extends** HttpServlet {@WebServlet("/logout")



/\*\*

\* Servlet implementation class HomeServlet

\*/

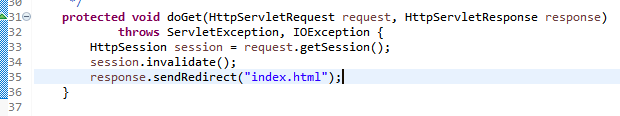
@WebServlet("/home")

**public** **class** HomeServlet **extends** HttpServlet {



@WebServlet("/logout")

**public** **class** LogoutServlet **extends** HttpServlet {



# Summary

=>slides: Pg. 40

In this module, we have understood one of the most important features in Servlets, that is, tracking session data with the support of HTML hidden form fields, URL rewriting, cookies, and sessions with proper demos. We can use any of the above methods to track the session data, and Servlets also support the use of the combination of more than one approach within a single Servlet page. That is, we can use sessions for maintaining the objects and secure data for a short period of time and cookies to maintain string information for a longer period of time within the single Servlet. In the next module, we shall understand how to upload a file using the new Servlet API.

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