Java Application Development with Tomcat

by Richard Monson-Haefel

Course Overview

Course Overview

[Autogenerated] Hi, everyone. My name is Richard Monsoon Hateful. Welcome to my course. Java application development with Tomcat. I am software engineer With more than 25 years of experience developing Java and Java enterprise applications, this course is a quick introduction to developing Java Web applications for Tomcat using the surly A P I and Java server pages. Some of the major topics that will cover include an or view of the HDP 1.1 protocol and how HDP is processed by Tomcat, the servant, a p I, including coverage of filters and listeners, Java server pages including to Java, standard tag library packaging, Java Web application in the war files and hot deploying them on time Cat. By the end of this course, you'll have a solid foundation to get started developing Java Web applications and deploying them on Tomcat. From here. Continue learning of Java Web development and tomcat by taking the course time cat for Java development, which is designed to be used with this course to teach you how to run, configure, troubleshoot and secure tomcat in development. I hope youll drawing me on this journey to learn about service and Js peas with the Java application development with Tomcat course at plural site

Installing Tomcat and the Demos

Installation and Initial Configuration of Tomcat

[Autogenerated] Hi. This is Richard Manson hateful. Walk up to the module installation and initial configuration of time Cat. In this module, you'll be downloading installing a running Apache Tomcat for the first time. Tomcatters cross platform, meaning it's configured and runs the same way on Windows, Mac OS and Lennox. Installation is slightly different from one operating system to the next, which is covered in this module. I'll be using Mackel s throughout the most of this course, but I'll switch operating system if there's a significant difference. What follows are installation instructions for each operating system, followed by directions for downloading and setting up the course examples.

Install Tomcat on Windows 10

[Autogenerated] in this clip, we're installing Tomcat on Windows 10. We'll start out by downloading the Tomcat installation file. Then we'll unzip it, rename it and move it to the sea dry. Next, we'll set up environment variables, including Java Home, Catalina, Home and Path. Then we'll start the time cat server and test it using your favorite browser. The Tomcat Web server is developed by the Apache Software Foundation, so that's the best place to find the downloads. Tomcat started out as a servant. Reference of a billet ation provided by Sun Microsystems in early 1998 has been managed by the patchy foundation says 2002 when it was released in version 4.1. This course is concerned with Tomcat Night, which was released at the start of 2018. If you open the side bar menu on the left, it may already be open. You'll find the download link for Tomcat nine Under downloads, click on Tomcat nine link and navigate to the download page. Scroll down until you see the various download options. Choose the 64 bit Windows 10 download. Once the Tomcat nine Zip file has been downloaded, open the file Explorer and extract the archive file. The directory with the same name as the Zip file should appear to make things easier. Let's rename that directory to simply Tomcat. We want to install time cat somewhere other than a download directory for this course, I suggest that you placed the directory directly under the C drive. Like thus in production, you may want to place it somewhere else. We're going to work with the command prompt in order to set up the directory rights and start and stop the Web server. The command prime can be found by typing command prompt in the windows search box like this.

You may want to say this to your tool bar or pin it to your start because we're going to be using this a lot. You might prefer to use power shell, which is fine, but the commands use may need to be adjusted as you follow along. Once you're in the command prop window, just type C h d I R for change directory C colon backslash to go to the C drive. Let's take a look inside the C drive to see what we have here. You can see in addition to the normal directories, the Tomcat directory. We want to change directory souls that we are in the Tomcat directory. To run Tomcat, you need to make sure that you have at least Java eight installed on your system. The course has been tested on both Driving eight and job 11. So either is fine. If you're unsure if Jonah is installed or which version you have, you can check pretty quickly. Just type Java dash version like this. In this case, I have DR 11 installed. If you don't have Java installed or you have a version earlier than Java eight, then you can download the version you want from Oracle or open J T K. Either will work fine. If you're installing the J. D. K. We want the JD cane that the J. R. E then make sure that you set your drive a homeopath to point at the J. D. K installation. You can see if the Java home is set by typing Set Java home at the command prompt. It should look something like this. In most cases, the J T. K will be installed in your C program files Java directory. If Java underscore. Home doesn't show anything or you get an air. Then you can configure the Java Underscore home in your system settings while we configure a couple of other environment variables. Let's set a couple of environment variables to enable Time Cat to run at Windows Search Box Type in Control panel and open it with a dialog box, Open select system and security and then system. Next, choose advanced system settings on the left navigation pane. When the dialogue appears, click on the environment Variables button at the bottom right of the dialogue. If you're driving underscore Home is not pointing to Java eight or 11. Then press the edit button to change it. If it's missing, click on the new button as you can see my job underscore Home is already set in addition to Java. Underscore home. You want to set what it's called the Catalina Underscore Home as you will learn a little later in the course. Catalina is the part of Tom _____ that runs servants and jazz peas. We set the environment variable Catalina Underscore, Home to be the top level of the Tomcat installation directory. In this case, the Tomcat directory just copied to your C drive. In my case, it looks as follows. We want to add Catalina, underscore Home and Java Underscore. Home to your path. Find the path environment variable and click on it. Now click at it toe. Add Catalina, underscore home click on the new button and enter percent Catalina Underscore home percent. My path already includes job home, so I won't need to add that. But if you're does not have that as well. When you're done, press OK and they're okay again and choose clothes with the J Decay installed and the Java underscore home and Catalina and score home and the path variable set. You're ready to run. Tomcat returned to the command line and change directory so that you're in a BIN directory. To run Tomcat, you need to execute the startup that bat file type and start up and

hit Enter. If all went well, you should see Tomcat start up and I'll put a bunch of log messages. The Tomcat server is now running in the command window. Open your favorite browser and go to local host Colon 8080 If you see a Web page like this one shown here. That means you've successfully installed and started Tomcat on Windows 10. Now let's shut down, Tomcat! Return of the command line and type. Shut down! That's it You've successfully installed, started and shut down Tomcat! Congratulations.

Install Tomcat on Linux

[Autogenerated] in this clip, we're going to install Tomcat on Lennox. We'll start out by downloading the Tomcat insulation file. Then we'll on archive it and rename it next. We'll set up the environment variables, including Java, Home, Catalina, Home and Path. Then we'll start the time cat server, and finally, we'll test it using your favorite browser. The time Cat Web server is developed by the Apache Software Foundation, so that's the best place to find a download. Go to tanaka dot Apache dot or GE Tomcat started out as a survey reference implementation provided by Sun Microsystems as early as 1998. It has been managed by Apaches since 2002 when it released version 4.1. This course is concerned with Tomcat nine, which was released at the start of 2008. Team. If you open the sidebar, many on the left and may already be open. You will find the Donald Link for Tomcat. Nine. Under downloads, click on the Tomcat nine linked to navigate to the download page. Scroll down until you see the various download options. Since this is for Lennix, select the tar dot GZ file and copy the link to your clipboard. We're going to use the W get utilities. Do the actual download now open the command terminal, which you'll find by typing terminal into the search field. The first thing we want to do is download Tomcat. It's usually recommended they put Tomcat under the O. P. T. Directory Option Directory Soul will create a Tomcat directory there changed to the O. P. T directory. Next, we'll use the W get program to download the Tomcat distribution, simply type w get space and then paste in the U. R L you copied earlier. If you run the L s command, you can see the archive file with the Tomcat archive download. It's timeto unzip or on archive it. To do that, we'll use the Tar Command with the Ex's E and F options. These options tell the tar utility to UN archive the file named in the listing. If you run a directory listing, you should see both the archive and a new directory that looks something like the one shown on the screen. We don't need the archive file any longer, so go ahead and delete it so that we can keep the op directory as clean as possible. The directory name is pretty long, so let's rename it to Tomcat to make things a little easier moving for next, you're going to set a couple of environment burials for Catalina, Homeopath and Java home. First, you need to make sure that you have at least Java eight installed on your system. The course has been tested

both job eight and 11 so either is fine. If you're unsure if job was installed or which version you have, you can check pretty quickly. Just tight job. A dash version. I have DR 11 installed If you don't have Java installed or you have version earlier than job, eh? You can go to either Oracle website and download eight or 11. Or you can go toe open J T K and download the free open source versions. The primary difference between oracles, J T K and Open J K is licensing and support, so either will work just fine. Go to your home directory and let's add it. The Bash profile file. I happen to use the sublime editor, but you can use whatever you want. If you need to add Dr a home environment, variable do so mine's already there. You will also need to set what is called the Catalina Hole. As you will learn a little later in the chorus, Catalina is the part of Time Cat that runs the servants and jazz peas. We set the environment variable Catalina home to be the top level of the Tomcat insulation directory. In this case, the Tomcat directory you just created in your act Directory type in export, Catalina, underscore Home equals slash opt slash tomcat. Finally, you want open both your java home and your Tomcat home. Pass to your path, variable. To do this, enter the following export path equals dollar sign path Colon 1000 Catalina Home Colon dollar sign Java home colon. Now that you have your path set up, you can save the dot bash, profile and go to the command prompt. In order for the new environment variables toe work, you can either close and reopen the Command Council. Or you can reload the terminal by typing source, got bash profile, navigate into the Tomcat directory and then start Tomcat by executing the following command Been startup s age. If all went well, you should see time cat start up and out Put a bunch of log messages. The town can't. Server is now running in the command window. Open your favorite Web browser and go to local host 80 80. If you see a page like the one shown here, that means you have successfully installed and started Tomcat on Lennox. Now let's shut down Time. Cat returned to the command line and type been shut down. That s H. That's it. You have successfully installed, started and shut down Tomcat. Congratulations.

Install Tomcat on macOS

[Autogenerated] in this clip won't stall Tomcat on Mac OS. We will start all by downloading and Tomcat installation file. Then we'll un archive it and rename it Next. We'll set up an environment variable, including you Have a Home, Catalina Home and Path. Then we'll start the Tomcat server and test it using your favorite browser. The Tomcat Web server is developed by the Apache Software Foundation, so that's the best place to go. To find a download, go to tomcat dot Apache dot org's Tomcats started out as the servant reference implementation provided by Sun Microsystems as early as 1998. It has been managed by Apaches since 2002 when it was released in version 4.1. This course is concerned with Tomcat nine, which was released at the start of 2008.

Team. If you open the sign bar menu on the left and may already be open, you will see the download link for Tomcat nine Under downloads. Click on the Tomcat nine Link and navigate to the download page. Scroll down until you see the various download options. Since this is for Mac OS, Select, the Zip or the tire version whichever you're most familiar with and downloaded. Now go to the finder and unzip or open the archive file. The name of the directory is pretty long. So to make things easier in the future, let's change it to just Tomcat. You can stall time cat anywhere, but for this tutorial, we're going to put it right under your home directory. In my case, the Home Directory is plural. Site copy of the entire directory to use her home directory like this. Next, you want to open a command line Terminal window. You can find the terminal window under utilities in the application directory. Or you can find it via the Mac OS search. Open up a terminal window to your home directory. If you do a list, you can see the time cat installed directory. Now you see H Mind Command to tell Mac OS that the files ending in S H under Tomcat bin directory are executed ble. Next, you're going to set a couple of environment variables. First, you need to make sure that you have at least Java eight installed on your system. The course has been tested. Both drive eight and 11. So either is fine. If you're unsure if Java is installed or which version you have, you can check pretty quickly. Just open a terminal window and type Java dash version like this. In this case, I have driving 11 installed. Anything from driving a through driving 11 should work fine. If you don't have Java installed or you have a version earlier than Job eight, you can go to either the Oracle website and download eight or 11. Or you can go to open today to Kay and download the free and open source versions. The primary difference between oracles, J T K and Open J K is licensing and support, so either will work fine. We need to set a couple of war environment variables for Tomcat using your favorite tax tattered er, I haven't used text mate. Open a hidden file named dot bash profile. This should be directly on your home directory. Next, you want to set what is called Catalina Home as you will learn a little later in the course. Catalina's the part of Time Cat that runs servants and jazz peas. We set the environment variable Catalina home to be the top level of the Tomcat installation directory in this case, the Tomcat directory you just copied and to use her home. In my case, it looks less follows. Quote users. Plural site. Tomcat. Finally, you want upend both your job, a home and your Catalina home environment Variables to your path environment variable. To do so, Enter the following Export path equals dollar sign path Colon dollar sign, Catalina Home Colon Dollar Sign Job A home Now that you have the rights and your past set, you can save and close the dot bash profile file. Then reload the terminal so that the dot bash profile is loaded again and your settings are fixed. You contest that your environment variables are correct by issuing echo commands like this. Echo Dollar Sign, Catalina Home. Good. Now you can start Tomcat by navigating into the Apache Tomcat directory CD Tomcat and then issuing this startup command as follows Been slashed. Startup dot s h This executes the time

Can't start up script. If all went well, you should be able to see Tomcat Start up and I'll put a bunch of log messages. Tomcat server is now up and running in the command window. Open your favorite Web browser and go to a local host. 80 80. If you see a Web page like mine showing here, that means you've successfully installed Tomcat on Mac OS. Now let's shut down time Cat returned to the command line and type been slash shutdown dot s age. And that's it you have successfully installed, started and shut down Tomcat. Congratulations.

Loading the Examples

[Autogenerated] you need the file The instructions in this video to see the examples Working on your own installation of Tomcat. I'm gonna be working mostly in the bash. Ella Mackel s. The commands I'm using are usually the same on Windows 10 and Lennox. So when I talk about the Tomcat directory, I'm talking about the home Slash Tomcat, director of Mac OS, the C colon slash Tomcat directory on Windows 10 and the Opt Tomcat directory on Lennox. No matter which operating system you're on, you should be able to follow along without a problem. If there is a significant difference between operating systems, I'll show you how it's done on each one. Examples for this course are located on the landing page of the course with any plural site course. Just select the exercise tabs and then click on the download button. Do that. Now Your download directory. You should see a foul name. Java application development. Tomcat got zip unzip that file and opened director with the same name. Within that directory, they'll be nine subdirectories. The directories, labelled 01 through 08 contained the slides for this course. The example. Source Code directory contains all the example, Source code and deployment descriptors. When it's time to look at the source code, I'll direct you to the correct file in the examples by popping up a banner like this one. It lists the name of the file and a directory path so that you can open it up in your own tech center and take a look at it. You should also see a maven palm file. I use this while developing the course it has. The Tomcat may even plug in, which automatically compiles the source code and packages it into a Web. Applications that run in Tomcat you can use may have been here if you like, but you don't have to. I've already packaged the demos for you, which leads me to the last item under the Examples Directory Zip file named Plural site. Tomcat War files dot zip contains the compiled and ready to run Web applications used as examples. In this course, we're going to deploy the contents of that zip into your Tomcat installation so that all the examples are ready to the demo. Copy the plural site Tomcat war files dot zip file and then navigate to where you installed Tomcat from this point forward when navigating to Tomcat installation directory, I'll use the Mac OS file finder, but everything I do within the Tomcat directory will be the same regardless of what operating system you're using. If there's a

significant difference, I'll show you how to do the exercise for each operating system, but that generally won't be needed under the Tomcat directory Paste. The Plural site. Tomcat War Files That's it. File into a directory named Webb APS. The Web Ex directory is where you put your Web applications when you want to run them on Tomcat. In addition to the ZIP file you just put in there, you'll see several directories, manager, whole Spanish, your examples. Docks and root. These come with Tomcat. They are the default Web applications that provide a landing page with Tomcat documentation examples, which is really handy when you're developing but would be removed for production. Unzip the plural site Tomcat War files that zip file, and you should see a director with the same name. Within that directory are several files that end with that war. These are the execute all Web applications. I'll be demo ing throughout the course copy all the war files and paste them so that they are directly under the Web APS directory. Next, go ahead and delete the plural site. Time. Cat War files that zip and the corresponding directory. You won't need them moving forward. Your Web APS directory should now look like this as a final step will test one of the demos to make sure they're all working. Go to the command line and change directories and tell your inside the Tomcat directory to start Tomcat. Type in the startup command as a reminder. Here is the start of command being executed on MCA less Windows 10 and Lennox right? The command down. If it's hard to remember, because from this point forward, I'll be starting and shutting down Tomcat. Using the Mac OS command line following along with Windows or linen should be easy. When Tom Cat is running open a Web browser and enter the following You Earl local host Colon 80 80 ford slash my underscore Serval it ford slash high. You see a Web page come up that has a plane tax, which says, Hello, world. If you do, then you're ready for the rest of the course. If you don't, then go back and follow the steps again until you get it working. You may have made a mistake when doing the insulation or when deploying. Example. Boris. Take your time. I'll wait. At this point, you have successfully installed Tomcat and download the examples and deployed the demos. This was an important first step and ensures that the examples makes sense and that the demos work properly. You might want to put the examples directory somewhere other than your download directory. I'll be referencing the contents of it frequently throughout the course. Now that everything is ready, it's time to learn more about a C, P and the Tomcat architecture.

HTTP and the Tomcat Architecture

HTTP 1.1

[Autogenerated] Hi, I'm Richard Johnson. Hateful. Welcome to the module HDP and the Tomcat Architecture. Er, we'll start our covering the, http 1.1 protocol in detail. Then we'll move on to the architecture of the Tomcat Web server, including Coyote, Catalina and Jasper. Http is the application network protocol used by your browser and other clients to communicate with Web Server? Remember, the sole purpose of Tomcat or any Web server is to handle http requests and generate http responses. When your browser requests a Web page, it first creates a T c P i p network connection to the Web server. Then it sends an HTP requests over the TCP I p connection to the Web server. The Web server, then res. The HDP request finds a resource or does some processing and returns in age to be response over TCP. Once that is done, the TCP socket is disconnected. Http uses plain text or ut f A and has a specific format for both requests and responses. If you were to look at a A C P request, it has three basic parts the Internet Web address and requested method meta data in the form of headers and, depending on the type of request, a body part in order to discuss the http request a response in more detail. We're going to use a program called Curl Curl is a powerful utility that allows you to send HTP requests at the command line. It's been in use for over two decades, and it's found on Windows, Lennox and Macal as operating systems. To test Tomcat with curl, open a command line prompt and start the Tomcat server you installed. Assuming you've seen Tomcats start up in your command window, execute the following command curl dash V for verbose and then local host 80 80 slash Examples slash Make sure you get the last splash in there. What you see here is the last few lines of the body of the HDP request scroll of command window all the way up to the top to see the H W quest. In response headers the first lines of output Tell us that curl connected to the local __ support 80 80 using TCP i p. It then sends the HDP request. The first line of the request is the method type the path and the version of http used There are several method tights, but get is the most commonly used. The next most commonly used http method is post. Then there's put in delete and a few others. You used the get method to request a resource in this case, a Web page and the post method to send data to a server. When you fill out a Web form and hit, submit your browser uses the post method to send the data the next few lines. Air headers the headers convey vary, but at the very least, you should see host, user agent and accept headers. Hosts identifies the host to which the message was sent. Notice that is just host, not the path The path is in the first line. The user agent identifies type of program that is sending the request. In this case, we're using the curl application, the scent request. Have you been using a browser? Chrome, for example. This line would identify the client as a chrome browser and might look something like this. If you used an iPhone with safari browser, it might look like this. The identity of the user

agent is captured in logs, which can help with troubleshooting and also help with a Web server to modify its response. For example, if a mobile device sends a request, the Web server can be set up to return an HTML page formatted for a mobile device rather than one for a desktop Web browser. The except Header tells the Web server it can return any kind of data, be it plain text, html images or some other format. There are many possible except types, typically a browser except a large variety of data types. A request for Microsoft Edge might look like this, while the same request from Safari on Mac might look like this once the Web browser has found the resource, it will return an http response. The first line of an HDP response identifies the version of 82 p used and the response code. In this case, the response quote is 200 or okay, which means that the request was executed successfully. There are five categories of response colds. In the most cases, you want to see 200 or something between 202 199 because this indicates that the request was successfully processed. You don't want to see coz between 405 199 for instance, the dreaded 404 Because these are eric codes after the first line comes the headers, and these will vary according to the value of the response code. The content type, the Web server use and even the operating system, the most important. Besides, the response code is probably going to be the content type, which in this case, is HT Mile tax encoded with UT f A. Have you been accessing rest values that returns? Jason Formatted Data? It might look like this. If you are accessing a PdF, it might look like this. The content types are based on the media types, a K mime type standardized by the Internet Engineering Task Force. You confined all the media types defined by the i E. T f. At this, you are all. Finally, there's the HT two Peabody, which is all the content that comes in after the headers. In this case, it's an HTML Web page. Http is the foundation of both the Web and Web service applications, and understanding it fully will help you be a better developer. But there are other courses at plural site that cover the protocols in detail. The pro sight course that covers http 1.1 is HDB Fundamentals By Scott Allen Although we won't be discussing HTV two, which is an advanced topic, there is a course that covers that as well.

Tomcat Architecture

[Autogenerated] in this video and explain how an HTP requests flows through three components of Tomcat coyote, Catalina and Jasper. When aged P Client, for example, a Web browser sense an HTP requests it first establishes on T. C P I. P. Connection to Tomcat. The process of listening for connection requests and then establishing TCP Epi connections. Network connections is handled by a major Tomcat component called Coyote. When Coyote Connector receives a mashing tea spy connection, it examines the protocol in domain and passes the request to another component called Catalina. Catalina is a serval it engine. It's the part of Time Cat that manages and execute

serval. It's after Coyote hands often HDP request to Catalina. Catalina parses the HDP and looks at the path and resource requested and fetches a circle. It fromthe Web APS directory. It then delegates the HTP requests, encapsulated in a P I to the serval it. The A. C P Sir Villa type provides an A P I for accessing information about the HTP. Requests such as the method get post etcetera the path, any query or other parameters sent with the request, headers and, if appropriate, the body of the request The component, written by a developer, uses this information about the HTP requests as well as business logic and resource is like a database to process the request. Then Catalina takes the information return by the serval. It converts it into an H. D p response and returns an HP response to Coyote, which forwards it onto the Web client. That's the process. When it's a serval, it when it's the Jsp, it's slightly different. Let's assume that the Web app is a jazz peop age instead of a circle it. If the request is for A J S P. Paige, then another component on compiler called Jasper comes into play. Jasper's designed to convert Jsp pages into servility bite coat. The Jsp pages sent a Jasper, which converts it into a serval. It it generates a customs service lit class, an extension of HTV serval it for each GSP page based on the JSP Markup E L and tag libraries. After generating the JSP serve lit, this new Servilia is sent to Catalina and behaves just like a Searle. It written by a developer. It's important to remember that most cases a single instance of a serval it or JSP will handle all the requests from all clients instead of in Stan. She ating a new instance with each request the same Servilia instances used again and again. In addition, multiple clients could be accessing the same instance at the same time. Because of this, you have to treat serval, it's and jazz Pisa stateless, expecting a single instance to handle many concurrent threads. All this is configurable. You enforce time, can't to use an instance with each request or use a single instance but only handle requests one at a time. That said, using a single stateless instance is what we recommend every serval it has what's called a life cycle when the first call for a specific servility is received. Can Ilia in Stan? She hates the serval it and then calls. It's a nit method passing in an object called the serval Config, once a servant is initialized, is ready to handle requests. At some point when the survey has thrown an exception, redeployed or otherwise shut down, Catalina calls its destroy method. Thean it met that gives a servant time to set up resource is in variables before its first call. The destroy method allows it to do cleanup like closing Resource is before us discarded. Okay, we covered a lot of ground. This module. Hopefully the discussion of http and the structure of its request and responses help you to understand the full time cat architecture. Understanding Tomcat architecture will help you understand subsequent modules and make it easier for you to put the pieces together as they're covered. And the next module will get into some cold as you learn some basics about developing surveillance. If you're interested in learning more about running time, Cat consider watching the companion to this course Tomcat for Java development. While

this course focuses on programming servants and Js peas, the Tomcat for Java development course focuses on running, configuring, troubleshooting and securing Tomcat in the development environment.

Developing Servlets

The Servlet Class

[Autogenerated] Hi, I'm Richard Matson. Hey, Phil. Welcome to the developing surveillance module in this module. We're gonna talk about the serval it programming model. We'll start with the class hierarchy, which will help you understand how htp requests get routed to the correct service methods on your servility. Then we'll talk more about those service methods. Next, I'll talk about the primary interface to htp requests. A response is the age to be servant requests and the HB Servant response and the primary interface the server container, the serval context. And finally, the interface to the user session, the HDP session. There are lots of examples and demos in this module, so I'm excited to get down to coding the servant a p I One point all was released in 1996. The original version of the surly AP. I was limited to a proprietary project, code named Jeeves. The first time I worked with service was in the fall of 1997 when Sun Microsystem released the Java Web serval IT Development kit with a servant, a P I to 19970.1. In 1999 the Java Web service development kit was donated to the Apache Software Foundation and renamed Tomcat. This was a complete rewrite By James Duncan Davidson. Since then, the service platform has evolved significantly as an independent frame or and as part of the job E e and now Jakarta E. But the basics are still the same. The primary purpose of the survey, a P i n Jay's Peas, is the process HTP requests for dynamically generated content. In 2019 Oracle donated the entire job e platform to the Eclipse Foundation, including the serval It AP I, which was renamed Jakarta. Eeee and Jakarta surveil its respectively. So when you hear the term serval it, Java, sir, let a P I or Jakarta serval it. They're all talking about the same thing. Before we get to the cold, Let's examine the super types of the circle it from 20,000 feet. The generic servility and the HDP serve it and how these types are extended to create custom serval. It's and the next video developer First service, which will then expand adding more capabilities. The servant FBI in general is designed to take a single stream of data and return the stream of data. It's a very simple and elegant programming model. The generic service is an abstract based class of all surveillance. The generic service class, which is general in scope, is rarely extended directly by developers. Instead, it's

specialized to meet specific needs. In the case of the service at a P I. That's always been processing HTP requests and responses. The HDP service specializes to generic servant so he can handle HTP requests and responses they used to be serving. It, in turn, is extended by developers to create custom service and is also the foundation for service generated from JSP pages.

Developing a Simple Servlet

[Autogenerated] in this video, we're going to develop a simple Servet which were growing complexity in subsequent videos. I'm going to start from scratch using only a text editor. The actual cold for the servant could be founding examples you downloaded specifically the example in the directory serval. It's examples as you learn the last video a serval. It is a job. A classic stuns the HP servant class. We import the age to be servant type from the job back servility HDP package Our class Hello World serval. It needs only extend the HDP service abstract type and override the appropriate methods. Toe work. Let's take a look at the Asian Be Servant class here. You can see that the HP serve It extends the generic serval it hp Cyril. It pulverized a generic service service method, which is called whenever an input stream is delegated to the serval. It. The service methods simply cast the request and response objects into the HDP counterparts, age to be servant, request an HB servant response and then calls the HP Cyril its own service method. The HDP service method does one simple thing. It looks at the original HTP requests and determines what HCP method is being called. It can access the HCV method type using a serval in AP I specifically using the HP Servant request Get method Call that returns a string that identifies the htp method, which will be get post. Put it delete etcetera. Remember that the method of the HP request is found in the first line of the request. Catalina Tomcat Circle Container extracts the method from this line. If it's an HP get request, the logic will invoke the do get method. The H p service default Do get methods, simply returns an error code indicating that the HP get method has not supported the do post method as well as other methods. Do the same thing in order to process HDP. Get requests are custom class. Hello World Searle. It has to override the do get method. Let's do that now. First, I'll copy the method signature of the do get method, and I'll paste that into our hello world servility. The do get method will take care of reading the HP get requests and sending back in HDP response. In order for this to compile, we're going to add imports. The HB servant request and the HP servant response and certainly exception and Io exception. Right now, the servant will not do anything we need to modify it, so that will return some kind of utf eight taxed. To do that, we use HDP service response there. Spice Object has a method that provides an IO print writer, and we can use that print writer to output the text. Hello World. The last thing we need to do has told Tomcat the final HDP path to the serval It we use an

at Web servility notation. To do that, we'll set it to the path, slash high and then import the annotations. So is a code will compile the devil for the Hello World Service was deployed when you copied the example. War files to the Tomcat Web APS directory earlier in the course. Assuming that time Cat is writing if a stock go ahead and get it started. Open your favorite Web browser and sent an age to be request to the URL associated with the hello world. Serval it in this case, http local holes 80 80. My serval. It's high, the my serval It part of the path is the war name. In this case, Hello. World Demo was packaged in a war foul called my serval. It You can see it in web, perhaps directory here by default. The name of the war file is used as the first part of the path. The last part of path is the slash high. We declared in the Web service an imitation. Now, son, the request, it returns. Hello, world, Which is the response we hold for Excellent. Now we're gonna take what we've learned and expand a little bit by obtaining more information from the HDP service request.

Working with HTTP Parameters

[Autogenerated] in this video, we're going to build on our work with the simple servility, adding ability to read and use query parameters. This is possible because the age to be servant request contains the parse contents of the HTP requests sent by the browser. They used to be Request Path, Queer Parameters and Other Header cookies and attachments, and so on can be obtained using the HP request methods. All this data comes directly from the HDP request received by Tomcat. In addition, the HTP requests object contains information specific to the runtime state of Catalina and the Web application that is running, which it obtains from the Tomcat and Catalina engine. For the demo, we're going to improve the hello World, serve it so that it's a bit smarter by working with the HDP request parameters. Specifically, we want a more personalized greeting than Hello World. To do this, we're going to use the code from the first example, but change the class name to surveil it. Pai Rams for parameters will also change the path from high to Pai rams. Before you make any more changes, though, let's see the finished demo in action then, with the understanding what we want to happen. We can call the rest of our serval it assuming that Tomcat servers still running. If it's not, please started. Now we can demonstrate this Web application in action. Open a new tab in your favorite Web browser and type in the following you. Earl Local host 80 80 My serval It Paramus. Question Mark Name equals Richard or whatever name you want to use. The question Mark marks the start of the query string parameter. The name Equal Richard sets a parameter of name with the value of Richard. Now execute the request. You should now see the browser return. Hello, Richard. Now let's take a closer look at the code. In the HDP request we set from the browser, we specified a query string specifically the key

name followed by the Value Richard. To get that query string we need on Lee, call the method get parameter on the age to be servant request with the name of the parameter, which, according to your all his name, the value returned. We put in the name variable. If there's no parameters with that name, the method returns No. So let's account for that now. Let's change the tax we write to hello and then contaminate the value of the name variable. That gets us to a simple greeting that we saw in the demo. Let's try it with a different name. Try Bob. Great. Now you know how to use the H P Service request to get query parameters from the Earl. Let's keep going with the next video.

Working with HTTP Headers

[Autogenerated] there's much more we can do with the HDP. Servant requests an age to be servant response. For example, we can read standard and custom Hatters from the request and change or add headers to the response. Like the HCV servant requests. The HP servant response is a rapper for http. It provides a number of methods that confused too right data to the response, such as the status code, the content type headers and body content. The http servant response provides all the basic HDP response data that we need by the false, so we don't need to create it from scratch. But if we need to add a header or a response code, we can do that, among other things, using the age to be servant response. To make things more interesting, let's get one of the headers from the request and use that to further personalize our greeting and set a header in the response as an example will read the except language header from the HDP request. The except language hunter informs the Web server what language the browser prefers. Http uses a B, c P. 47 i e. T f language codes, a browser may have more than one language preference and a language code may be followed by a country code here. The language called his fr for France, which was followed by the Country Code for France, as opposed to Canada or Belgium. English E n has a country code of us, while German is D e with no country code. In this case, we can use that information to customize the greeting of our response. Instead of always returning the English Hello will make it able to respond in other languages, specifically French and German. Before we start quoting, Let's run the demo to see what's supposed to happen. By default, we will assume that the browser has a prefer language of English, so I would expect in English greeting Open your favor browser and enter in local host 80 80. My serval It headers. Question Mark name equals Richard. We use the Web address for the My Servant Web application again, but this time we're changing the final path from Paramus Toe headers hit. Enter, There we go. No surprises here. Now let's write the code ourselves. We're going to start with the serval. It cold from the previous video and expand it. First, I'll change the name to serve a litte headers and the path to

slash headers like this. Next, we'll get the except language header from the age to be servant requests The age to be several requests provides a convenient method for obtaining the except language header called Get Local, which returns a Javal Util local object like this. Let's assume the first language code in the list is the correct language for this exercise. We don't care what the country code is. So will modify our get locale to select the 1st 2 characters on Lee. So if it's France should be f R. If it's German of B D E. Now, depending on the preferred language, we want to respond with a greedy int in the correct language. This calls for a case statement that tests the language code and sets a greedy into the correct value. As an example, I'm supporting three languages French, German and English. English is the default. Now we can change the HDP response body to use the correct greedy int. The value of the greeting is set in the case statement. So if the preferred language is French it will be bond. You're at the preferred languages. German. It'll be guten tag for anything else will make the English Hello the default. Next we'll change it from hello to use the greeting. Finally, let's tell the browser the language we're responding with by using a response header called Content Language. We can do this using the set header method on the age to be servant response. The set header method takes two parameters. The name of the header and the valuable contain. This method can be used for adding or modifying any header to the HDP response. Here we're setting the header named content language to master preferred language of the browser. So if the preferred languages and German, we set the content language to D E for German. When we executed a demo earlier in this video, the preferred language of my Web browser was English, so it returned the default English response. Hello. Now we will change the preferred language of the browser to French and we should see the page reload as Barnes, you're Richard instead of Hello, Richard, If you are using Google Chrome or Microsoft Edge, you can change the language code as follows. Click on the more hamburger button on the upper right hand corner of the browser, then on settings. Type the word language in the search bar and open and click on the add language button. Now, assuming you already have English set at French, then move that to the top of your preferred language list. The process is basically the same for fire fax, except you go into the preferences rather than the settings For safari. You have to change the language of your entire machine, which is asking a bit much for a demo. Now reload the browser. As you can see, the servant detected the preferred language used by the browser in this case, French and return the correct green Bonn jur. Feel free to try this with German or other languages if you like.

Working with the Servlet Context

[Autogenerated] the serval contacts is the interface between your servility and the servant container, which in this case is Catalina. The serval context can tell you a lot about service configuration and can also be used to store values or attributes that the servant can use in a thread safe manner. I'll demonstrate this by storing a counter as an attributes and update it with every call to the serval. It you could obtain a reference to the servant context from the HB servant request passed into the do get method like this will also create a temporary variable that will use later toe output. The count the context object allows you to store any value you want and was called on attributes we're gonna be storing the counter were using in an attribute named Counter. We're not sure if it's been set yet, so we'll ask for it before doing anything. Once you have a reference to the servant contacts, you can create or get access to account. We're not sure that the counter attribute has been set yet, so we'll ask for a generic type rather than trying to cast it. If it's no casting, it will generate an exception. Since servants are multi threaded, many client requests may be executing the do get method. At the same time, we want to use a thread safe object. The Java Concurrency Library provides what's called an atomic imager, which is threads safe. So we'll use that if the attributes already stored in the servant context, then we want to incriminate it, which is actually easy to do in a thread safe manner using atomic imager. Then we'll place the current value of the atomic manager into that temporary variable we declared earlier. If, however, the counter has not been set, then the reference will be no. Then we want to create a counter for the service like this. So if the colander already exists, we've simply increased its value by one. If it didn't already exist, then this is the first time to do get method is called on a service since it was initialized. So we set one up with the counter update. We can send the count back to the browser like this. Okay, let's test the demo and see what happens. Open a Web browser. I'll start with Google Chrome toe local host, 80 80. My underscore serval it context. Underscore Count. You should see serval account equals one because we're executing this. Circle it for the first time. Now, if you have another browser installed, open that if you alternate back and forth between the browsers, you'll see that they're both using the same count. In my case, I'm using Google Chrome now, but I'll open up Microsoft Edge as well, using the same girl. Load the page in a second browser. As you can see, it's too now let's try fire, fax. It's three, just as we expected, the question might be asking right now is why not use an instant variable or static variable on the serval it itself. The reason you don't want to do this is simple, Although Tomcat tends to use the same instance, the process all requests, it might be configured to use multiple instances now or in the future. If that's the case, then each instance will have been loaded with a separate class lawyer. So not only will the instance variable would be unique to each instance, but so too, with the static variable. Another possibility is that the servant container ran into issues with the original instance and replaced it with a new one. It's considered a best practice to avoid using instance or

static variables. Ana Searle it for this reason. In addition to setting and getting attributes, the server context provides a lot of information about the serval. It itself from the version number two servants specific resource is.

Working with the HttpSession

[Autogenerated] while the Serval Contacts represents a single serval it deployment. The HDP session object provides access to variables and information associate. With the communications between a specific client and a Web application. Assessing an object might be used to store a cart Every time the user makes a page request, they're using a different serval it, but the same session object. So adding something to the cart from any page results in the item being safe for all the pages in the Web application to create a session with a client, for example, your Web brother Tomcat will need to use cookies, hidden form files or what is called U R L rewriting Without one of these methods to track the user's activity, every request from the client will be considered as a new client requests in a new session. We'll use cookies in our example of shoulder differences between sessions and one browser and obsession and another rouser. We get the session object from the age to be servant request, just like we did from the servant context. From that point forward, we do exactly the same thing we did with the servant contacts. Only this time we'll set and get the attributes from the session object again. The circle is already deployed, so you can access it from your browser by loading the following you Earl local host. 80 80 My underscore serval it session underscore Comb reloaded and you can see the count increase if you do the same thing with another browser. In my case, Microsoft Edge Tomcat will consider that to be a different client and therefore a different session. The same is true if I switch to Firefox, alternate between the browsers reading the page and you see that they're counts are different. There are essentially two ways to end the user session. The first is to call the invalidate method on the session object. You'll probably want to do this if the user logs off of the Web application. The second way is via timeout period time. Cat itself has a default time out of 30 minutes. If the user has no interaction with the Web application for more than 30 minutes, the session will automatically timeout and any attributes stored in the session object will be discarded. You can change that time in Tomcat configuration if you need to have a different time out for your Web application than the default time won't set. Like Tomcat, you can do so in the Web XML or by an imitation condition to setting and getting attributes and setting a time out. The HB session object. Provide some information about when the session started the last time it was used and its current setting for time out. You can also get a reference to the servant context, which were covered in the last video. In this module, you learn about the servant programming model, which is essential

to understanding how to develop jazz P pages as well. It's configuring Web applications and Tomcat itself, so everything we've card thus far will come back in the future modules. You learned about the inheritance graph of servants that practically all servants are subtypes of age. To be servility, you learn how to use do get method with the http request and the HP response to get information about the request and determine the HP response content and headers. And you learned about storing attributes in both the servant contacts in the session object and how the scope of these attributes differ. The serval context of specific to a deployed service. Instance, while the session object is specific to a user's conversation with Web application, you learned that Tomcat normally uses a single stateless instance for each service type to process concurrent requests. Next, you're going to learn about JSP and how it builds on the Cyrillic programming model to provide a powerful template ing engine for generating Web applications.

Developing JSPs and JSTL

JSP Expressions

[Autogenerated] in this module, you're gonna learn how to use the JSP template ing language to create dynamic Web pages. You learn how JSP pages air compiled into service and get a peek at the generated source code. You'll learn about JSP expressions, jazz piece, cripple its and declarations. And finally, you learn about using Java beans and the jazz TL Tag library. When I was creating my first Java Web applications in mid 1997 the jazz piece specifications was not yet ready for production. As a result, I had to write all of my HTML manually using the age to be servant response output stream. You could imagine how unruly right in Java code I'll put every HTML tag can get. The serval it programming model was great, but generating HTML was laborious. When Java server pages was introduced in 1998 everything became much simpler. You could seamlessly mix HTML with Java code, taking advantage of the powerful Cyrillic programming model as well, a simple html Marco. Since then, Java Sir pages has continued to evolve for 22 years and is today a part of Jakarta Eee platform and officially named Jakarta Server pages. Although the JSP moniker is probably more familiar. JSP is a type of template ing language, meaning it provides markup for generating dynamic Web pages. Jsp allows you to mix familiar HTML tags with specialty tags and Java code. It's really pretty elegant in many ways. For example, here is A. J. S P. Paige. It does the same thing as the simple servant you created in the first video. The last module, you know, says just plain html up to the point where we get the name

from the query parameters. At that point, our JSP page breaks into what's called a jsp expression tag, taking the form of a lesser than simple followed by percent and then equals. This is followed by the Java cold, which must have a result in this case, the value of the name query parameter delimited by percent greater than symbol. After that, it's just plain html. Again, let's see how this JSP page work, says a demo. Using your favorite browser tested Jazz P page with the following you Earl local host 80 80 My underscore jazz p Just be simple. That jazz p question mark name equals Richard. As you can see the JSP paid you just looked at, which is simpler and richer than hand coding. A sir little put. HTML works perfectly fine. You can think of a jazz P page as the body of a do get method, anything that you can do it a serval it you can do in a jazz P. Paige. He probably knows at the get parameter method is called in a request object. Jazz be pages automatically have access to objects in the surly AP I, which are called JSP. Implicit objects. The full list of implicit objects available to any G S. P. Paige are as follows The requests and the response objects give you access to the http servant request and the HCP servant response. The out object is a direct reference to the A c P sir Responses output stream in the form of a print writer. In addition, there explicit objects called config application session page Context page, an exception which corresponded the same objects in a circle. It a p. I remember when we talked about how Jsp pages were compiled into serval. It's by the component called Jasper. Under the covers. Jasper's actually a built in serval it like the default serval it, But its purpose is to compile GSP pages into Searle a code. So while Jasper's conceptually a top level component in Tomcat and practices a predefined Servet called the Jazz P Service, which has one sole purpose of reading JSP pages and compile them into service by code. Normally, when Jasper converts JSP page into a serve licked source code, the source code is then compound into bite code and stored on the server for use later, so that E S P pages don't need to be re compiled every time you use the generators. Source code for the JSP service discarded in the process. But we can tell time cat to reserve it. As an example, I'm going to show you the Java source code generated for our first jazz P example. This is the servants source code that Jasper generated. As you can see, there's a lot of code here. The generated Java class just takes the content I put in the jazz P. Paige and spits it out, using the HCV servant response output stream. Simple right? You write the Jazz P code, and Jasper does all the work of creating the corresponding serval it. In addition to using the JSP expression tag, you can also use the JSP expression language. The expression language is really very simple and limited. It provides Boolean an arithmetic operations and dot notation for working with Java types and has its own set of implicit objects that overlap with the JSP. Implicit objects expressing language, also called e l, is delimited by curly brackets and prefaced with the dollar sign like this using the e l Implicit Object program, we can accomplish the same thing with this here. L is treating the key name as a member of the parameter object A dictionary of age to

be servant request parameters. It does the same thing is http several across get parameter that we used in the second video The last module. Here is a list of all the e l implicit objects available within the e l statement. They're sort of matt types that could be accessed using the dot notation shown in the last example. They can also be access using map index seem

JSP Scriptlets, Declarations, and JavaBeans

[Autogenerated] the jazz P expression language when used in combination with jazz piece Cripple. Its A script lit could include multiple lines of Java quote instead of a single line. As it's the case with the GSP expressions, this could be useful if our code is more complex. This is a jazz P Paige into which we're going to put a script lit. The script lit starts with a less than character, followed by a percent sign and ends with a percent side, followed by a greater than character. To make things easier, let's take the code from the serval. It underscore headers. Example. Fromthe Last module. You can take the contents of the do get method and copied almost verbatim into the jazz piece. Cripple it like this. All we need to do is import the Java you till the cow class in what is called a jazz P directive like this. Another change we have to make is the substitute, the jazz p implicit out reference for the use of the response. Get writer. If we don't, then the output won't be inside the H one element. It'll just print in plain tax somewhere outside the HTML now open a browser and assuming time Cat is running navigate to the following you Earl Local host. My underscore jazz P jazz piece Cripple its that jazz P question mark Name equals Richard. Now I've set my browser to the German language to differentiate the results from previous examples. Let's see what happens. Guten talk, Richard. That's exactly what we wanted. As you can see, this code allows us to do more interesting things, but take notice. It still looks pretty messy. In fact, as you add more more scripless, the situation just gets worse. The situation can be addressed to some degree by using what is called jazz P declarations using JSP declarations the less than percent exclamation point we can use to complete Java methods and call those methods using jazz P expressions. It's a little better, but in practice it would be better if we could just separate the Java code from the JSP entirely. That's where javabeans comes into play. Job of beans are one of the oldest standards in the Java platform. Created 1996 by Sun Microsystems. The javabeans a P I is really a collection of idioms and conventions that Java classes can adopt in order to become a Java bean, you could turn any simple Java class into a job of being with only a few tweaks. A Java being doesn't implement a specific AP, I necessarily. It should adhere to standard idioms and conventions. It's good to be aware of these conventions. So is that your beings work seamlessly with JSP. Here are the basic rules a Java being must have a no or constructor if it has other constructors are no constructor at all. Javabeans stores at state, in the

form of properties and properties, are access via getter and set her methods. A job of being can have other methods, but you need get her and set her methods. Access properties properties must be discoverable using reflection. The beans properties must be serialize herbal job being support a specific event, a p I. Where it could be both a publisher and a listener. But it doesn't have to implement that. A p I Jazz P was especially designed to work with Java beans components. The idea is that you can put your Java code into Java class instead of inside the jazz. P. Paige and the Page Condell, a gate the work to the bean or use the being to maintain state. I've created a Java being called Hello Helper. Here is the class definition. It has one method. Get greeting, which is not a property but utility method. We can modify the JSP page to use the Hello helper class. First, I have to import the Hello help her class I created so that I can reference it in a jazz P. Paige. For that, I use a tag that's less than at page. This is called a directive. Next, I'll get a reference to the Helo Helper instance and start in a variable by using the jazz P use being tagged like this, the jazz be used. Being tag will automatically and Stan, she ate a new hello helper. If one has not already been created or if one has been created, it will reuse that you could set the scope of the instance reuse at one of four levels specific to the jazz P. Paige. The requests So Influences is in Stansky with each requests and discarded with the corresponding reply the session. So that same instance is used for all communications between a specific client session or the Web application level, so that the same instance is used in all requests in all sessions that access the Jazz. P. Paige. Since our Hello helpers stateless, I'm gonna use the application sculpt to avoid having to create and destroy the instance more than once in the life cycle Web application. That is, from the time it's deployed until the time it's un deployed in Tomcat or Tom Gotta shut down. With that. I can simply replace the Jazz P Declaration with the jazz P expression that refers to the helper variable and invokes the get greedy method on the helper class. To see this in action, run this example. Local host 80 80 My underscore Jazz P Jazz P Being jazz P name equals Richard and there you go still in German. The nice thing about all these options is that they can be used as needed. We can build on JSP expressions using different constructs, including jazz piece cripple, its Jensby decorations and references to Java beans. Next, we'll learn about hag libraries

JSP Standard Tag Libraries

[Autogenerated] the JSP standard tag library, or JST. El for short is a suite of JSP extensions that provides tags and functions that could be used when developing JSP pages, while Jakarta Eee application servers are required support JST l playing servant containers like Tomcat or not. Still JST l's in such common use that it really needs to be covered in this course on Tomcat condition. Tomcat is frequently used as the servant container for full Jakarta EEE application servers, where

support for jazz TL is required. There are six libraries defined by J. S. T. L that you can use in your Js peace, including Core, which provides variable assignments, control and flow functionality formatting, which allows for date and number four manning as well. Internationalization Functions, which provides a number of helpful functions for working with string values in numbers and so on. XML for reading Ex Mel documents and SQL for executing sequel queries against the database. For this video, we're gonna focus on core formatting and function JST l libraries. The XML and SQL libraries are no longer recommended for use in anything but prototyping, so those will not be covered. We'll start with a J S T L Corps Library. In order to use a jazz tell library you have to import it into your jazz P. Paige. This is done with a jazz P Tag live directive, which starts with an angle bracket, followed by the standard Jazz P percent sign and then an at symbol to identify it. As a directive, we use a directive in the last video to import the Hello helper class. This director's a little different because it imports tag libraries. The your eyes a unique identify WR like a package name that identifies the tag a library and use every tag library should have one. We're gonna sign this. You are I an alias, the prefix of C. The prefix allows it to fully qualify the tags. With a library imported, we can start to use course specific tags. Here's a list of court tags. As you can see, there are flow tags such as, if otherwise, choose looping tags like four each. Four tokens win. There are directives like import redirect, remove and some other useful tags like set u R l an implicit object references out in Paramus. Of these, we're going to look at the if choose and sat jazz steel core library tags. We'll start out with a tag SAT, which is a variable declaration for this. We need a C prefix to identify the tag line over his jazz T L corps, followed by the reserved words set. Let's create a variable called language code, as we have in previous examples using the VAR attributes and then set that equal to a preferred language code of the browser by studying the value attributes equal to the following jazz P expression. This expression gets the except language header, splits it into an array with each language code as an element and then uses the first element, which should be the preferred language. It's messy and not very robust, but its use is temporary until we cover Jazztel functions in a just peace cripple it declaration and javabeans examples. We use a Java switch statement to select the greeting. We can use a J S T o cor tag. Choose to accomplish the same thing. We start out with the choose tag prefaced with the core library identifier. Within the truth tag are one or more wind tags, which define a test that must evaluate true. In order for the text in the body of the tag to be used in this case, we're checking to see if the language called Variable declared in Set take contains a value of f R for French or D E for German. If there is no match, then it goes to the fault. Otherwise, tag, which sets the language to English, choose statements, terminate execution as soon as a matches found, so a break is not needed. So this use of the truce tag, other than using different syntax, is basically the same thing you've been doing in all the examples with the Java switch statement.

Finally, we want to greet the person by name, so we'll add a JSP expression to get the name from the query string in the URL. You've used this before. Prior examples. We closed the age to mill with end tags for H one body and html. Now let's run the example. All you need to do is enter the following you, Earl local host 80 80 My underscore J S t l jazz till core dot jsp question mark name equals Richard. Now execute. If your browser said the English will say hello, Richard orbits at the French or German. It will show those greedy INTs. You can play around with language settings to see different, greedy INTs. Next, we'll see example of the J S tail functions and use. Unlike the jazz, tell Core library. The function library provides functions rather than tags that you can use inside the JSP expressions. Thes could be grouped into functions that test to see if values contains strings, string modifiers and a couple of other things. In the next example, we use two of these functions starts with an upper case. Starting with the last example, we add another directive to import the jazz Tell function library like this. No, it's the prefixes set the FN, which is what most people use with a J S T L function library. Important we can put it to use. We could start by replacing GSB expression with a simple request for the entire except language hunter, which you'll remember will look something like this. We just used the e l implicit object header passing in the name of that header we want to access. So now we have an entire heading in our language code variable. We contest the 1st 2 characters of the string to determine the preferred language of the browser. Using starts with jazz still function. The starts with function works just like the javelins String starts with method. In this case, it takes a full string value of the except language header and test the 1st 2 characters for codes fr d e. If neither is found, the otherwise value is used. I'll wrap up this example by converting the greeting into upper case using the two upper case jst l function like this. You contest this by entering the following You are all local host 80 80 My underscore J S T l Jazztel functions that jazz p question Mark name equals Richard. And there you go, the greeting fouled by the name in upper case again. You can play around with language preferences. If you want the jazz, tell formatting library allows you to convert tax from one format to another and from one language to another, I'm gonna add the date and time to the output so the final results will look like this. We're going to use the jazz tell formatting tag named format date to make this happen? There are several other four Manning tags has shown here. The first thing we need to do is import the J S T L formatting library, just like we did for the corn function libraries. Next, we'll get a reference to the Java you till date object using the JSP used being tag. We don't need to import the job you till date because it will be used in the e l expression, and it's fully qualified already. That gives us a variable with a reference to today's date. Next, we'll format it using the J S T l format date tag. The attributes tell us that we're formatting the value of the date object in the now variable. The pattern attribute requires a time date pattern. The pattern's use Here are the same patterns used in the Java that tax that simple date

format class in this case, day of the week, month, day of the month and time. Finally, we take the formatted string and place it into a new variable called date underscore time. When we all put the greeting, we will follow it with a date time stamp by simply using the date time variable in a JSP expression like this. I'm gonna go ahead and take the upper case out and add a little bit of Burbage. Now, go ahead and run the demo by pointing your browser at the following u r l local Holst 80 80 My underscore J s t l jazz till four batting dot jsp Question mark name equals Richard And there you go. Hello, Richard. With the time stamp works Perfect And this one G s p. Paige used three G s, t l libraries, core functions and formatting. There are other third party tag libraries and you could even create your own. Although that's outside the scope of this course. One more thing if you change the preferred language, not only does the greeting change, but the date and time value does as well here. I just changed the preferred language to French and now I'll reload it. See, it's in French. The date and time I'll do German next and there it is in German. Pretty cool right In this module, you learn that jazz P is a dynamic template ing language that uses implicit jazz P objects, jazz P expressions with their own implicit expression objects James Piece cripple its declarations and Java beans. In most cases, you're gonna want to use Java beans in orderto add Java functionality to A. J. S P. Paige. But you will always have access to scripless and declarations in addition to those. And of course, you'll be using the JSP expressions throughout. You also learn about the Java standard tag library, specifically the core library, the Functions library in a formatting library.

Developing Filters and Listeners

Servlet Filters

[Autogenerated] Welcome to the module. Developing filters and listeners. Service and jazz peas are essentially a combination of presentation and controller models in that they control the presentation of the Web pages and control the interaction between clients and the business logic. Servant filters and listeners are neither controllers nor presentation components. They are utilities for enhancing the performance of servants and jazz P pages and observing and monitoring their execution. Servant filters are essentially htp requests in response interceptors, which can add different qualities of service to those requests and responses. Servant listeners react a life cycle events and changes in state of a serval context. H P session and HP requests listeners air used to

monitor and log behavior. Filters are used to change behavior. The Servant Filter, a P I. Is a standard part of Java Web applications, and it's supported by all servility engines. While you can reuse the cold from filters. An instance of that code while we created for each Web app and won't impact other Web APS deployed on Tomcat, we will talk about this Maurine the modules on deployment and configuration. But a Web application is a combination of one or more surveillance Jazz, P's Web pages, images and filters, all packaged into a container called a war. File filters all follow the same programming model. They are custom classes created by the developer that extend the HT to be filter type. They implement a single method. The do filter method, which is similar to the servants. Do get method with the age to be Cyril. It request in HB servant response parameters but also a filter chain parameter. The do filter method is divided into two parts. The cold that executes before listserv lit or JSP is above and runs before the call to the chain do filter, and the cold that executes after the service or JSP is called below or after the chain do filter. The idea is that the filter allows you to read and modify requests before they get to your serval it, and also to modify responses right after your service is done. But before the responses are returned to the client, filters could be used for whatever you want, but are commonly used for authentication, data compression, blogging, token izing, encryption and validation. Filters can be chained in a Siri's. You might have one filter doing logging, another handling security rules and 1/3 unzipping or zipping data. When can delete a hands off an age to ___ request to a serval it The filters are called first in the order they were declare. The code designed for pre processing the HTP requests handles HB requests on the way to the serval it. Then, after the service, it's done the filters air called again to pulse process. The HP cross in the reverse order, looking at the coat again. All the logic and a filter that needs to be execute before a service is called. It's placed in the do filter method just before the call to chain do filter all the logic that comes after the filter is called is the code following the chain Do filter call. Let's do a really simple implementation. You can add content to the HB servant response both before and after the service is invoked by writing to the output stream both before and after the call. Here's an example. Now imagine. I set up three different filters in the chain. Call them filter one filter to and filter three. So the output felter one before the serval. It is called would write filter one B before, before and the same filter would write. Filter one, eh? Where A is for after we can see how this would be executed. Using a sequence diagram the B stands for before the Circle It and A for after the serval it first Catalina calls Do filter on Filter one, which obtains a reference the print writer and outputs the tax filter. One B filter one then calls the chain do filter method, which passes the call to the filter too. Filter to outputs the tax filter to be. And then the sequence continues until the do get method is called on the serval. It after the service is done processing the do get method. The filters air called again, executing in reverse Filter three is called first,

which outputs filter three a and returns after it returns. The calls past a filter to which outputs filter to a and so on until all the filters have executed and controls passed back to Catalina to illustrate, I've created an example that will execute exactly a show him above. Assuming time Cat is running. Point your browser to HTV local host, 80 80 my filters, you should see the following output. Let's review first Catalina calls the Do filter on Filter one, which obtains a reference to the print writer and outputs the tax to filter one. Be filter one, then cause a chain do filter method, which passes the call to the filter too filter to outputs the text to be. And then the sequence continues until the do get method is called in a serval. It after the server is done processing, it's do get method. The filters air called again executing Reverse Filter three is called first, which outputs filter three a and returns after returns. The calls past a filter to which I'll puts filter to a and after filter to it passes the filter one which I'll puts Filter One, eh?

Application Event Listeners

[Autogenerated] There are many types of application listeners, which can be divided up into two categories. Lifecycle listeners and state change listeners. Lifecycle listeners respond to changes in creation, use and destruction of serve like contexts. HTP Sessions and HP requests state change. Listeners respond to changes in attributes and other objects. Associate with the serval it session or request. Let's take a look at a single example to see how listeners work in combination with events generated by Catalina in the A, C. P and Tomcat architecture module. I explain the basic life cycle. Serval it. Let's review that quickly. When the first call for a serval it is received Catalina and Stan. She hates the serval it and then calls. It's a nit method. Once the service in Stan she ate it is ready to handle requests from that point forward, every request to do, get or do post or any other HDP method is handled by the same instance, so a single instance can have many threads running through it at the same time. This is why you must not rely on instance variables in a serval it because it certainly is not thread safe. At some point when the seller has thrown. An exception is redeployed or otherwise shut down. Catalina calls its destroy method Thean. It method gives the servant time to set up a resource is and variables before its first call. The destroy method allows it to do cleanup like closing Resource is before it's discarded. Here is an implementation of the serval it context listener used in this demo. Looking at the sequence diagram again, we can add the servant Context listener, which is associated with the serval it but is invoked by Catalina. The context initialized method is called immediately after the serval innit method is called and before the first service requests do get or do Post is called the context destroyed. Method is called immediately after the serval it destroy method is called. But before the service is actually D referenced and garbage collected, remember that the servant destroy method might be called in

the event of an exception or if a servant is updated or destroyed, a redeployed or removed. Go ahead and start Tomcat and open a Web browser. Enter in the following Earl local host 80 80. My underscore Listener hit. Enter the message you see here says that the serval it works, which is good, but we're interested in seeing the messages logged by our servant context listener. To see that message navigate to the Tomcat Installation Directory in the Logs directory. There, you'll see a number of logs open the log named local host dot than today's day Got lock. There may be a lot of logs in here already, but you're interested in the log written by the listener, which happens to be this first log message. This is the message that was logged by our listener in the initialized context method. Now, to see the context destroy message, close the tax editor and go into the Web ABS directory. And there you'll see all the war files and their directories for the demos removed. The war file Name my underscore listener dot war. Don't delete the directory with the same name. Once you have removed the war file, put it a safe place so you find it later after the war files removed, washed to sea When time cat deletes the my Underscore listener directory. This means that the serval it has been destroyed and is no longer in service. Now, if you open up the local host log as you did before and go to the bottom of log. There, you'll see a new log message that was sent by the context destroyed method in the listener. This message was logged by our listeners. Destroy Context Method If you are interested in learning more about log files and other directories under time Cat installation Fuller, please consider watching the sister course to this one titled Tomcat for Java Development. It provides a lot of detail about logging, configuration and other Tomcat specific operations has explained. There are almost a dozen types of listeners. The listeners all have their own scope and purpose or event type. You can use listeners the log events that happen in your application, but they're not the same as filters. They cannot impact the processing of any messages or the servant itself. They simply react to events. In this module, you learn about developing servant filters and listeners. Filters are extremely useful for dealing with incoming HTP requests before they reach the serval, it or outgoing age to be responses from serval. It's before they go to the client. You can use them for a variety of purposes, including authentication, data compression, logging, token izing encryption and validation filters intercept and could modify http messages. Listeners, on the other hand, do not modify age to be messages or change the way servant responds to request. Instead, they listen for certain life cycle events and ST changes to surveil its context. HCP sessions and they should be requests. Use filters to help in handling request use listeners to observe or monitor serval it's sessions and requests.

Deploying Web Applications

The WAR File

[Autogenerated] Welcome to the module. Deploying Web applications In the next couple videos, you're gonna learn how Web applications air packaged into war files and that war files air really jar files that have a very specific directory structure. You learn how to create war files using the Java jar utility, and finally, you learn how to hot deploy war files into Tomcat. A Web application, a K Web app, is packaged and was called a Web archive or war file. A War file is simply a zip file with a specified directory structure. It contains your service jazz, peas, filters, listeners and static content such as HTML Web pages, images, CSS and JavaScript. A war file has an extension of DOT war, and the process of creating a war file is called packaging or archiving. This requires the use of a tool of some kind. Open a file browser and navigate your Tomcat server directory, also known as Catalina Home. If you expand the Web directory, you'll see a lot of jar files with names like My Underscores Serval. It's Not War and my underscore jazz P dot war, as well as directories that have the same name as the war files. But without the dot war extension, when you deploy a weather application, you will normally just drop a war file directly into the Web. APS directory and Tomcat will automatically UN archive the war file, leaving a matching directory. This is called a hot deployment. Let's take a look at the Hello Web application. Every Web application has a root directory and, at the very least, a subdirectory called meta i n F. In most cases, you could ignore the manifest on M F file containing the metal iron F. The manifest dot M F file can be useful in some advanced scenarios, but for 99% of Web applications, you'll never touch it in time. Cat, The Meta Enough directory is also used to store what is called the Context Out XML file. The context dot XML is a file used by Tomcat to map a reference to a resource like Ajay BBC data source to a physical resource such as a database. For example, this context XML file declares a J D B C data source and how its driver connects to a specific database. In this case, my sequel, a War file, also requires the presence of a subdirectory called Web I Enough. The Web are enough. Can, in the simplest cases, be empty, but it's more likely to contain a Web XML, file libraries and possibly some compiled servant classes. When you deploy a serval it in a war file, the class files for the servant have to go under the Web. Enough classes directory nested in subdirectories that match the classes package Name the JSP, CSS, HTML. And image files, on the other hand, could be placed directly under the route. Developers configure whether applications using either an ex Mel document named Web XML or servility annotations the Web XML is stored directly under the Web I Enough. Directory of the War file annotations and Web X Malcolm used together. However,

if there are annotations that overlap with configuration of the Web XML, the Web XML takes precedence. It's important to understand that the Web XML foul can be used to configure multiple surveillance and jazz P pages, all packaged into the same war file, and that you can only have one Web XML file for each war file. So a war file contains a Web application, which may consist of one or more surveillance and jazz P pages, all of which are dependent on the same Web XML configuration file and, to a lesser extent, Servilia annotations.

Package and Deploy a WAR

[Autogenerated] in this video, we're gonna manually create a simple war file using the command line tool that ships with the J. D. K called jar and then deploy it to Tomcat using your file bras or go to the directory Examples. Source. Deployment Underscore examples. There are three files in that directory. The Hello Jsp, The Hello, serval it that Java and the Web XML file. We're going to use the 1st 2 files. Hello Jazz P and Hello, servant that Java to create a Web application. Let's take a look at these two Web components. Open up the file named Hello, that jazz ___ in your favorite text editor. This should look familiar as it uses a jazz P tag to obtain a query. String crammed her with a key name. It should output Hello, followed by the name followed by from hello dot jsp. Next open the hello serval it dot java in attacks outer here, you can see it's simply outputs Hello, followed by the name followed by from Hello Serval it not Draba. Before we can package these Web components into a war foul, we need to compile the Hello Serval it that Java class and to do that. We need the servant. A p I classes imported into the surveillance. Lucky for us, the entire AP I ships with Tomcat, go to the Tomcat directory, also known as Catalina Home and open the Lid directory. This is where Tomcat stores all of its libraries. Make a copy of the serval it dash a p i dot jar file and put it under the deployment. Underscore examples directory under the examples. Now go back to the command line, navigate into the employment. Underscore examples directory. If you do, you should be able to do a listing that would look like this. Now let's compile a serval it execute the following command. Java Sea. Hello, Serval it that Java dash class path. Serval it dash a p i dot jar when you have successfully compiled a hello servant that Java file into a class foul, you're ready to move it into the right directory. First will create the Web i N F directory, which is intended to store Java classes. Next will create the class of subdirectory. The rest of the directory structure on the classes has to match the package name of the hello world dot Java class. If we take a peek at the file again, you can see that the package name is calm. That plural site dot tomcat, we need to replicate that directory structure under the Web. I enough directory below the classes now create those directories. First is calm and under that plural, say, and finally Tomcat. Now copy the hello world that class file into that directory. Next, we need to package the

Web application into a war file. We do that using the jar command, assuming your command line is still in the deployment. Underscore examples directory. You can run the jar command as follows. Jar C F v my underscore weather app dot war Hello, that jsp Web Dash I Enough. The jar is the utility that will create the war file. The C F ve tells the jar file to create an archive using the foul name. My underscore Web app War. After naming the war file, we list which files and directories want included in the war. In this case, we only need hello dot jsp and everything under the Web dash i n f directory. I've intentionally left out the web dot xml file because for this Web application. It's simple, and you don't need it. Since it's not listed in the jar command, it won't be included in the war file. If you do a listing again, you'll see a new file. My underscore Web app dot war. Now using your file browser, make a copy of that war file and paste it into the time cap Web APS directory and watch what happens if Tom Cat is running. Your file was created as shown here. You should see after a few seconds not only my underscore Web app, that war file, but also matching my underscore Web directory. If you open that directory, you'll see that the helo that jazz p file the Web I enough file and all the classes To run this, you need to open a Web browser and enter the following you. Earl Local host. 80 80 My underscore Web app. Question Mark name equals Richard, and you should see the results. Hello, Richard from Hello, Serval it That class Perfect. That's the serval It next. Let's try the jsp page Change the u R L adding hello dot jsp in the path and you should see the HTML message. Hello, Richard from Hello JSP. As you can see, both the new servility and the original JSP both worked. As expected, You have successfully deployed a war file that contains multiple Web components, namely a jazz P. Paige and a serve lit. The two Web components are part of the same Web app and are packaged in the same war file. Now we can configure them both using the Web XML file, which is covered in the next module. You learn in this module that a Web applications packaged into a war file, which is really a type of jar file, that the war file has a specific directory structure that includes the meta Enough directory, which has generated automatically, and the Web I enough directory where you put your Web, XML and Java class files. You also learn that jazz Ph. Two mil images, JavaScript and CSS files can be included directly under the root of the war. Finally, you learn how to use the Java jar utility to create war packages and how the hot deploy war files and Tomcat

Configuring Web Applications

Servlet and JSP Mapping

[Autogenerated] welcome to configuring Web applications. The final module of this course. In this module, you'll learn how to use configuration files and annotations to compliment code that you write for your surveillance jazz, peas, filters and listeners. We'll start by doing some simple map ings of service and jazz peas to ur else in the Web XML file. Then we'll implement some context parameters. Finally, it will configure some filters to use with our Web components. The simplest jazz P pages do not require a Web XML file at all. The JSP examples that you have worked with thus far haven't used them. But in production, it's likely you'll want to use them at the very leaves to declare you are all past your Web components. We're gonna start with the demo code from the last module. Open the Web XML file Under deployment. Examples. The simplest Web XML file just declares an ex Mel schema and version and looks like this within the Web. At route we put any configuration require for Sir Let's and Jazz P pages. It's important to understand that a Web XML file can be used to configure multiple surveillance and jazz P pages all packaged into the same war file and that you can only have one Web XML file for each war file. We won't be talking in detail about serval. It annotations in this course because they can only be used with service. And we want configurable service and jazz P pages. For that you use the web. XML, a servant of mapping, is created in two parts. First, there is the service element, which assigns a logical name to the serval it in a war file. For example, let's give a logical name. Hi. Serval it to the com plural site that Tomcat Hello, servant class. You declared earlier the circle it name value doesn't have to match the servants class name. Once you have a circle it name sat, you can add a second part which maps the servant identified by that name to a path. This element is named the serval it mapping and it assigns a path to the logical name of the serval. It declared in the previous element. The logical name we use is high. Serval it so I'll like that to the servant name element. Next, I'll set the path at which the servant could be located. Remember that the path is appended to the war. Name, for this example will set the path to four slash aloha. Don't leave out the Ford Slash before the path name or the Web App won't deploy. Let's compile a repackage my underscore Web at war and deploy it with the new path mapping. This will be an excellent review of the last module. First saved the Web XML and then move it into the Web. Dash i n F. Directory. Assuming the class file you compiled in the last module is still under the Web, I enough classes directory. We can skip compiling that and just repackage the war file with the Web XML included. Open a command line and navigate to the examples source deployment Example Directory. Next, execute the jar Command as you did before creating a war file named My Underscore Web App. That war. If you look closely at the output, you'll see that in addition to everything else, the Web XML file is now included in the war file. It was not included in the last module. Make a copy of the

Web, underscore Web APS war file and paste it into the Tomcat Web APS directory overriding the one that is already there if Tom Cat is running, will automatically detect that the war file has changed and update the Web app. Now try the girl that worked before it no longer works. It's been overridden by the Web that XML file the U. R L now requires the addition of Aloha to the path. Add that to the earl and try again. You should see the message Hello, Richard. From the helo Serval it That class file servant mapping is can also be used for jazz P pages. Right now, if you change the Earl replacing Aloha with L o J S P, you'll see the jazz p page again. But fault Tomcat maps any JSP pages in the root of the war file which in this case, is hello Jsp to the name of the war file, followed by the JSP page name. We start out with the same element to assign identifier to the JSP as we did with the serval it. But instead of the serval it class element, we use the jazz P file element and we enter the actual directory path relative to the Web applications route. Hello, That jazz P is listed directly under the route. So all we need to do is list the name as slash. Hello, J s P. Make sure you include the slash in there or you won't be able to use the jazz P page or the serval it with a new logical name assigned to the JSP page. We can add the second part of the mapping, which is like the one we did for the serval It serval it mapping servant name will call it high Jsp And the next element is called neural pattern. And for that, we'll set it to slash all that. To access a servant, you would use this. You are all It probably seems a bit weird that we're using elements serval it mapping and servant name to configure the path for a jazz P. Paige. But I think this was done for two reasons. First, it reminds us that at runtime every jsp pages actually compiled by Jasper into a serval it second, it makes for fewer XML elements that we have to remember. Save the Web XML and go ahead and repackage the web app as you've done before. and added to the Web's directory. Wait 10 seconds or so for time, Cat the update and then enter the girl you've been using thus far. But replace hello dot jazz p with Ola. Well, hopefully it worked. You should see the message from the Jazz P. Paige. Take special note that you no longer have to identify the Jazz P page with the file name before removed to the next video. I have to warn you that time has extremely fussy about the structure of the Web XML file. For example, if you don't put a four slash in front of the Jazz P file name, the Web application will not compile. Also, if you do not specify all your service elements before the Cyrillic map ings, it'll fail. Make sure you foul the structure exactly.

Context Parameter

[Autogenerated] frequently. An application requires access to parameters that you do not want to. Hard code examples include Web service and points directory, pass resource lynx and other parameters. Before serval, it's it was accomplished using static members, property files and ex

Mel documents with SERVERLESS Angie as peas, you can provide access to shared parameters easily using context parameters, which are made available to all service and jazz peas in the same war. File To demonstrate, go to the deployment, underscore example folder again and open the Web XML file. You just added in the last video toe. Add context parameters used the context hyphen parameter element. And this is important. Place it above the surge alert elements. Remember that time Cat is really fussy about the Web XML structure. In this case, I'm telling the application that I want a parameter named greeting with a value of howdy available to all Web components in the war file toe access those in the Jsp page, you can use the implicit object and knit. Paramus, open up the hello jazz P Paige and you can replace the hello text in the Circle it with the JSP expression like this at runtime the Jsp servant will access the value associate with a contact parameter named Greeting. We could make a similar change to the helo Searle. It's not Java class. In this case. We access the context parameter from the context. Object first will import the Cyrillic context tight. Next will make a call to get a reference to the serval context. Then we can use the context to get the parameter we want with the parameter. We can replace the Hello Tax, and I'll put it with the greeting as you have done before. Compiled the Hello sir. Let that Java file and move the class under the Web I Enough classes directory within its package. Overriding the existing file. Next used the jar utility to package the Web application with the updates to both the Web XML and the Hello Servant Class. Now copy the My Underscore Web at War, file to the Time Cap Webs Directory and overwrite the file that's already there. Then wait a bit for time. Can't to reload the Web application, open your Web browser and tasked the Jsp page first with the following you Earl Local host 80 80. My eye on the score Web app Ola question mark name equals Richard. The results should be as shown on the screen. See how hello has been replaced with howdy Now change the path they used Aloha and subtle Ola in order to execute the serval it. And there you go Both of Jsp and the serval it have access to the same context parameter.

Filters

[Autogenerated] as you learn in the filters and listeners, module filters can be used to intercept and act on. Incoming and outgoing http messages. Let's review how filters fit into the overall architecture filters process. Incoming HTP requests and outgoing HP responses. If more than one filter is used, the filters will execute. In Siri's first of filters process. The incoming HDP request reading, modifying and adding headers and any dated to the request body when more than one filters declared they executed a Siri's, according to several rules. Here is a Web XML that declares Filters at the top are the server and serval it mapping declarations. And just below that are the filter and filter mapping declarations. Let's take a closer look at the filter declarations. The filter

element has to sub elements the filter name in the filter class, similar to the serval it map ings. The filter names are identifiers associated with fully qualified filter classes, so each filter identifier mass to exactly one filter class. The question you may be asking is why a sign, a logical identify rhe to a fully qualified class name when you could just use the fully qualified class name the filter elements also lie. You declare initialization parameters. For example, if we change these filter element so that there's two filters, filter P and filter M weaken, declare them so that each references the same class filter one but has different parameters. We can differentiate them by citing the different logical identifiers. Okay, let's go back to the filter elements we had before. Given these filters and their elements, let's take a look at the filter map things. The filter's execute according to a few rules you are all patterns execute in the order listed. So if the filters air declared in this order, the HP request is processed in the order of filter one filter to filter three. Then, after service done processing the filters execute reverse order filter three. Then filter to and last filter one servant li and execute in the order listed just like Earl Patterns on the HTP requests. It's filter one filter to and filter three on HB Rie Spies will be filtered. Three filter to and then filter one. If a single mapping has both A you are a pattern and a serval it name the U. R L pattern is used. You are. All patterns take precedence over servant names, So if you have a mix of several it mapping is where some are. You are all patterns and others are serval. It names all the U R L patterns will execute before the serval it names in this case, filter one in filter three Have you are all patterns so those will execute before filter to which has a serval it name. So filter one executes, then filter three than filter to on the HP Request and onion HB response. It reverses filter to then filter three and then filter one. The Web XML includes a servant which you've already seen execute the three filters in order the 1st 2 filters used you are oh mapping. So those will execute first which happens to match the order in which they're declared the third filter Uses a serval it name so ill execute Last enter in the following you Earl local host 80 80 My underscore filters Okay, that worked is expected. Let's take another look at the filter map ings According to the serval, it mapping the third filter will only execute If the component is a serval. It name the serval it. Remember that war file may have multiple surveillance and jazz peas, but there can only be a single Web XML for war file. So determining which filters execute depend on the surveillance or jazz P Paige you're accessing. Although we did not test it at the time, the war we used for the demo in the Filters and Listeners module actually has both. A servant, Anna Jasper Page. You've only seen the circle it execute, but if you look at the Web XML it, you can see that there's also a jazz P. Paige. If we call the Jsp Page, we should see filter one and filter to execute, but not filter three, because it's specific to the serval it name, not a you're all which is required for jazz P pages. If we change the euro using her browser, adding simple that jazz p at the end. Then we're targeting a jazz P. Paige, and so the serval it enter local host 80 80. My underscore filters simple dot jsp. As you can see,

the third filter did not execute because it was Matt to the servant and not the jazz P. Paige. Well, that wraps up the last module configuring Web applications. You should now have a pretty good understanding of how to configure service and jazz peas, that multiple surveillance and Js peas could be configured using the same Web XML and that you can configure the Earl at which service and Js pease air accessed using the serval it map ings and which filters air run using the filter map ings. Congratulations. You've reached the end of this course Java application development with Tomcat. I hope you learned a lot and enjoyed it. The purpose of this course was it introduced you to the development of Web applications that is service and jazz pees on Tomcat. This is a broad subject, so you build to use this course as a springboard for more complex topics. This course also complements another course I've authored named Tomcat for Java Development. While this course focuses on development of Web applications, the Tomcat for Java development course focuses on running, configuring, troubleshooting and securing time cat for Web application development. If you want to improve your mastery of Tomcat, I recommend taking that course like this one. It's less than two hours of video, so it's short and packed with useful information. Thank you for watching and good luck

Course author

Richard Monson-Haefel

Richard has more than 24 years of experience as a professional software developer and architect. He has written five books on enterprise Java including EJB, JMS, web services, and software...

Course info

Level	Intermediate
Rating	★★★★★
My rating	★★★★★
Duration	2h 0m
Released	10 Apr 2020

Share course