The Lifecycle of a Request

00:00: Welcome to the sixth module of Course Two: The lifecycle of a request. A quick note is that this module is definitely code heavy so go ahead and start the project in your favorite ID if you want to follow along. We are going to basically debug through the entire lifecycle of an HTTP request so we are going to explore exactly how the mapping handler of the request gets resolved. We are going to have a look at the marshalling and unmarshalling processes. And we're also going to have a very quick look at how to catch exceptions in the dispatch of servlet. Alright, so let's get started.

00:38: Okay. The very first thing we need to do here is we need to define the HTTP request that we are going to debug through in the API. So notice that we are going with slash privileges here, so that is the request that we are going to be sending to the API and that is basically the full request that we are going to be debugging through this entire module. Also, notice that we are asking for JSON data, we are defining this accept header right here and so we are asking for a JSON representation of our data. And that is because the API does support both XML and JSON, so we want to basically make a decision early on. Okay, let's switch to Eclipse, let's start the API and let's start debugging through this request. Alright, so let's now start our API application and then let's start debugging... And there we go.

01:31: We now have the full application started so we are going to switch back to the browser and send the data request. But before we do that, we of course want to have a breakpoint ready so that data request actually hits that breakpoint and we can start debugging. And this is really where things start becoming interesting. We are going to set a breakpoint in the dispatcher servlet, that is the central place where the breakpoint needs to go so that we can catch this entire HTTP request from the very beginning, or almost from the very beginning because there is some processing going on in the HTTP servlet. So first of all, you can see how it leaves, retrieves the sources for this file without us having to do anything. So just opening the dispatcher servlet was enough. So let's maximize this class and let's find the do dispatch method and this is where we are going to set our breakpoint. And there we go, we now have a breakpoint ready so we are going to very quickly switch to the browser and send that request.

02:38: Okay, we are now sending the request and of course as soon as we do that, the breakpoint that we just configured in Eclipse will be hit. And there we go, we can immediately see that the request is processing, but of course it's not actually getting a return because it's blocked in that breakpoint. So let's switch back to Eclipse and let's start debugging. Okay, so we are now finally debugging through this HTTP request, so what we'll do here is I will step through this code and we will basically go over the entire lifecycle of this request. And the very first thing that we care about is a little bit lower right here, this is going to be the resolution process of the handler that's actually going to handle this HTTP request. We will basically iterate through all of the available handlers until we get a match for the handler that is actually able to process our request.

03:31: So let's step into this get handler and let's see exactly what this does. And this I was just explaining, this is simply iterating through all the handler mappings and is going to basically stop when the very first handler is actually matched. So the first handler that is going to be not null, this is going to be the handler that gets match and this is what the request is going to be processed with. So of course the first question here is what handler mappings do we actually have and we can easily figure that out by looking at this variable and looking at the handler mappings that we do have. And you can see now that the first one is the request mapping then we'll have the handler mapping that looks at the bin name and matches that to the URL and then we have three other handler mapping objects that we don't really care about and that is mostly because this will almost never match and they're also very, very low in the priority order.

04:29: So if we look into these handler mappings, we'll notice that they have a order. So this is the second one of course this would be order zero, so this is the most important one and these other three have a very high order, specifically so that they run way after the standard mappings that we actually care about and that actually have a chance of mapping to our request. Okay, so let's now go forward and let's see how this first handler, which is of course the request mapping one, let's see exactly how this gets resolved, because of course, this is the one that we actually expect to be resolved. We don't expect this request to be handled by any of the other mappings, we expect this one to be the one that gets resolved and that handles the request.

05:18: So let's step through, so this is the get handler implementation but we will go even deeper, we will go into the get handler internal implementation because ultimately that is where the resolution happens. So let's now step into this implementation and there we go. This is the actual resolution process where we are going to see the lookup path and then we are going to see the exact lookup of the method handler. So let's first see what the lookup path actually is and as expected that is slash privileges because that is what we were requesting from the client side, some very minor logging, acquiring a lock to the mapping registry and before we go further, let's actually have a look inside the mapping registry.

06:06: This is where all of these mappings are held and so, this is the central thing to understand when you're talking about resolution of HTTP requests to controller methods. This is basically the driver of that resolution process. We really dug through this code to get to the mapping registry. So, let's have a look at that. So we can see here a lot of maps. We can see a course look-up map, this won't have anything in it because we are not defining anything for course, this is empty. And we'll see a bunch of others, but the one we care about is this mapping look-up. So if we have a look at the mapping look-up, we will basically see the entire list of controller methods and their individual mappings. This is exactly the mapping process in its entirety. This is basically where the resolution happens.

07:00: So this is why this is so important to understand because this can save you a lot of time. Whenever you have an invalid mapping, whenever you're sending a request but it's not getting mapped and it's not hitting the right controller, whenever any mapping problem happens, this is the place to look. So basically understanding this will be super useful and again, this is where this map is held. So now, let's look at the look-up method here and of course, now that we know the look-up path and we also know where the mappings are held, it's pretty obvious how this look-up works. It's basically looking into that map and matching the key. So, not very complicated once you understand where to look.

07:45: And of course, we now have a handler method. We are now seeing that the handler method is this public privileges controller find all. So just as expected, the privileges controller find all is the one that is going to handle slash privileges. Okay, so now with the handler method resolved, let's step our way back to the dispatcher servlet. Now instead of actually stepping through the code, let's actually go back to the dispatcher servlet here and let's set a breakpoint right after the handler resolution process, because we can do that, and now we can simply run the code until it hits this other breakpoint. And the reason I'm doing that is, if you don't care about the mapping process, then you can simply have a breakpoint ready to go here and you're going to only hit that once the handler has been resolved. And, there we go. We have hit this breakpoint here and so we have our mapped handler.

08:46: Okay, so now that we have the handler, what's the next step in the lifecycle of this request? Well, the next step is going to be actually invoking the handler, hitting the privileges controller find all operation and then continuing on to process the request. So let's step through and get to the execution of the handler. And there we go. This is the next major place where we could install our breakpoint if we don't care about the handler mapping process and we just care about the actual execution. So we could have a breakpoint here. Let's now actually switch to our privileges controller and let's have a breakpoint in that find all operation. So we have this breakpoint here and we can now just run the execution and wait until this gets hit. So we can see here that the find all operation executes as expected and, of course, the very last step in the lifecycle is going to be dealing with the return of this method. This method will return a list of privileges and so, what we care about now is, how do these privileges get written to the response?

10:00: Remember that from the client side, we were asking for JSON data. We used the accept header and we basically requested a JSON representation of our data. So what that means is that now springer needs to figure out exactly how to get that JSON representation and how to write that to the response. And for that, we will go forward to the final, highly relevant class that's involved in this whole process which is going to be the request response body method processor. And, as you can immediately see in the Java doc of this implementation, this will basically deal with the request body annotation and it will basically marshal the response types that we are getting from our internal implementations back to the client. So, back to JSON or back to XML or back to whatever media type the client requested. In this case of course, is JSON.

10:53: So let's put a breakpoint in here as well, and let's resume the execution. And the main entry point that we are going to use to install our breakpoint is going to be a method called write with message converters and this is actually a method that is in the base class so, we're going to have to step into the base class. And now, we're going to find this write with message converters entry point. And the name of this method should be pretty self-explanatory. This is going to be the process where our internal representation gets written to the response using the framework HTTP message converters.

11:29: Okay, so let's resume the execution and let's start stepping through this code. And, here we are. We resumed the execution and we reached this write with message converters method and now we can step through this implementation and we can see exactly what the process looks like. So, the first bit of meta-information here is going to be the return value class and you can see the Array List because in our case, we are returning an Array List of privileges. So the return value class is the Array List. The implementation then tries to determine the generic type information in our response, and you can see that it does get it right. We have here a list of privileges, so it's able to determine that generic information correctly. And then looking at the request, it's trying to determine what the request media types were. So it's basically trying to figure out exactly what the client asked for. And of course, as expected, this is application JSON.

12:29: Now, the producible media types here are not what the client asked for but what the API can actually produce, so the producible media types should be a lot more than just application JSON. Let's have a look. We have here application XML, we can produce text XML, we can produce a lot of variations of XML, and we can produce of course application JSON and some variations as well. So we can see that our API is actually able to produce a lot more than just application JSON, but in our case we care about application JSON.

13:07: The next bit of logic is trying to match what the requested media type was with the producible media types, and it's basically trying to determine if we have a match or not. Because what's important here is are we able to actually produce what the client asks for. So if the client asks for JSON, and our API does support JSON, we are good. But of course, for example, if the client was asking for HELD, which is a media type that we do not support, then there would be no match between our producible media types and what the client requested. In which case, we would have this HTTP media type not acceptable exception. But of course, in our case, we are going to have a match and we are going to move forward.

13:52: There we go. This is the match, the requested type is application JSON, the producible type is application JSON, so we have a match, and we are moving forward. There is a very quick sorting process here in case we have multiple matches, which is not the case here. The next bit of logic is going to select the exact media type based on the previous information, so we are going to have a media type of JSON. Let's have a quick look and let's make sure that is the case.

14:25: And now the next bit of code is where the actual marshalling happens. We are going to iterate through all of our message converters. We can see here these four that basically goes through all of the message converters that we are defining. And the one that actually is able to support this media type that's the one that is going to get used. So even though this process seems a little bit complicated, the actual process is pretty simple. It's basically iterating through these message converters. And the very first one that actually can handle the media type that's going to be used and that is going to marshal our list of privileges to the response.

15:05: So let's very quickly step our way through this process, and let's get to that exact point. We iterated through a lot of message converters until we found this one. So let's see exactly what this message converter is and we can immediately see that we are talking about the Jackson 2 message converter, which is of course to be expected because this is the only converter we have that is able to write JSON data. So of course, this would be the hit, and we can also see exactly how this check is done.

15:44: So the message converter has this 'can write' operation. So the process basically checks can we write JSON, and all of the previous message converters that we checked could not write JSON because they simply did not have JSON support. This is the first one that has JSON support, and so of course this one is the one that matches and it's the one that's going to get used to write out the body.

16:08: So let's step to that exact line where the body is going to get written and then let's step forward. This is the exact point where our return value, our list of privileges, is going to get written to the body of the HTTP response. And that is pretty much it. That is basically the end of this lifecycle process. There is just a little bit of exception handling I want to go over back in the dispatcher servlet, but this is basically where the lifecycle finishes. And this is where, if of course everything goes smoothly here, this is where the last step of this lifecycle runs.

16:46: So let's wrap this up by going back to the dispatcher servlet, and let's have a very quick look at the exception handling. Okay, so we are now back in the dispatcher servlet, on the actual invocation of the handler, and we can see here a few very good places to have breakpoints in case something fails in this whole process. So it's a good idea to have a breakpoint here in this catch, and then it's another good idea to have breakpoints here and here. And this is basically so that if something does go wrong and if you don't yet have the logging super well configured and you're not seeing what exactly went wrong, this is basically where to catch that exception and where to figure out what the details of that exception were. So basically it's a very good place to figure that stuff out.

17:38: Alright, this wraps up our deep dive into the lifecycle of an HTTP request through the dispatcher servlet.