CS 240: Spark Routes Transcript

[00:00:01] So in the previous video, we talked about endpoints that your server needs to implement and the fact that spark calls endpoints routes.

[00:00:10] And so I want to say a little bit more about spark routes and how they're defined.

[00:00:17] So as we saw in the previous examples, the spark class has some methods that you can call to define your routes.

[00:00:28] And so you can call spark dot get spark dot post, spark dot put, or spark dot delete.*Start visual description. The professor demonstrates how to define routes in Spark by calling methods like spark. Get, spark.post, spark.put, and spark.delete. End visual description.*

[00:00:33] Now, of course, these, these method names map to the different http methods or method types that that we're going to use.

[00:00:42] And these are the most commonly used methods in http.

[00:00:44] There are others, but these are the most commonly used ones.

[00:00:49] Get requests are typically used to retrieve information from the server.

[00:00:53] And so a client would send a get request if they're asking for something.

[00:00:57] Post requests are oftentimes used to create things.

[00:01:00] So if you're writing an endpoint that creates an object of some kind, uh a game or a user or, or something like that, you probably want to use a post request for that put requests are oftentimes used to update objects that already exist.

Start visual description. The professor explains the different HTTP methods and their uses, such as GET for retrieving information, POST for creating objects, PUT for updating objects, and DELETE for deleting objects. End visual description.

[00:01:14] So if you want to update a user's uh record or if you want to update a game, then those kinds of requests would probably use put as the method.

[00:01:22] And then of course, we have the delete method for deleting objects.

[00:01:26] OK? And so, you'll see that the syntax for all of these different method types is, is identical.

[00:01:32] The first parameter you pass in is the URL.

[00:01:36] Now all these end points use the same URL, which is just a slash but it could be a longer URL.

[00:01:42] And you would think that putting the same URL on different end points would be confusing, but it's actually not confusing to spark because they look at the HTTP method as well. So when a request comes in from a client there, what spark does is it looks for the first route in the routes that you've provided that matches the incoming request and by matches, I mean it matches the http method type and the URL in the request also matches the, the routes URL.

[00:02:12] And so the first parameter is the URL.

[00:02:14] And the second parameter as we've already seen is the code that implements the http handler.

Start visual description. The professor shows how to add a new route to the hello BYU example, demonstrating the syntax for defining a DELETE route. End visual description.

[00:02:21] So if we wanted to go back to a previous example and add a new route to it, we could um go back to our hello BYU example and I'm going to add a new route. So, the first route is a get route and it just returns to BYU. I'm going to add another route here.

[00:02:42] We're going to make it a delete route and maybe we'll, and maybe we'll return delete BYU.

[00:03:05] I guess that's not very nice.

[00:03:06] We don't want to delete BYU, but um because it is a delete end point, we're going to go ahead and delete BYU.

[00:03:12] Now, the URL S are different for these two end points.

[00:03:15] So let's um let's run this little server in this case, um That that example is running on port 8080.

[00:03:29] So when we call it with curl, we're going to run a curl. First one is going to be a get.

[00:03:46] So we get a BYU.

[00:03:49] But if we change the HTTP method to delete and we put goodbye in the URL, we're going to get back delete BYU.

[00:04:04] So it's really a pretty simple idea.

[00:04:07] They just look in the request and find the route that matches what came in.

[00:04:11] Now, interestingly, we could actually have the same URL on both of those end points if we wanted to.

[00:04:23] But in this case, it wouldn't matter because they're using different Http methods.

[00:04:26] And so spark would still be able to tell the difference between get and delete requests.

[00:04:31] So now if I call the Get Endpoint, it still works. But if I call the delete endpoint with hello, that will also work.

[00:04:49] OK? So that's just a little bit more about how to create uh routes or end points with spark.

[00:04:55] All right. The next concept related to Ralph's is the notion of named parameters.

[00:05:00] So when a client calls a, a web API N point, they put the URL, of course in the http request.

Start visual description. The professor discusses the concept of named parameters in URLs and how to use them to make endpoints more dynamic. End visual description.

[00:05:10] And a lot of times that URL is, is fixed, I mean, like slash hello is always slash hello.

[00:05:15] It's never anything else.

[00:05:17] But a lot of times it's convenient to put parameters in the URL parameters that change they're dynamic. And so we can't assume that they're fixed.

[00:05:28] And so in this endpoint example, um the, the first part of the URL is slash hello, but the second part is a slash and then a name.

[00:05:38] And so the notion is that this, this endpoint will return hello and then it will return also the name that you passed in the URL.

[00:05:47] And so in this case, we're passing the name parameter in through the second component of the URL path.

[00:05:53] And so you can see here that when um we register this endpoint, we register the URL as usual, but we put a put colon name and that's a syntax for uh specifying a parameter that's dynamic.

[00:06:06] And so all that means is that when a request that that starts with slash hello comes into the server, uh spark will just will take the second component of the URL and put it in the request object that gets passed into the handler.

[00:06:21] So it, it, the request contains the parameters, a set of parameters and it's really just a map.

[00:06:28] Um So we have the names of the, the parameters and then their values.

[00:06:32] And so all you have to do inside your handler is go to the paras map inside the request and grab the value of the name parameter.

[00:06:40] And so that, that allows you to make your end points a little more dynamic so that the, the caller can pass in information um through the URL.

[00:06:48] And so if we come back out to our code examples, let's go ahead and actually modify it to do to have a name.

[00:07:08] So in this case will go to our route here, I put col name on it.

[00:07:17] And then rather saying BYU here, we're going to say hello, request dot paras and pass in name.

[00:07:34] I guess we should put an exclamation point on there just to be enthusiastic about it.

[00:07:41] OK? So now we rerun the server.

[00:07:49] We should be able to pass the parameters.

[00:07:58] Hello Frankenstein. Let's see if we get the right thing back.

[00:08:02] All right, there we go.

[00:08:04] So that shows you how to, to use name parameters um in your end points. Now, in general, http requests consist of a URL path headers and a request body.

[00:08:17] It's actually possible to pass parameters or information into an endpoint in any of those three places, the URL, the headers or the body of the request.

Start visual description. The professor explains the different ways to pass parameters into an endpoint, including through the URL, headers, and request body. End visual description.

[00:08:27] So what we just did showed you how to pass the parameters through the URL.

[00:08:32] All right, let's go back to the slides.

[00:08:41] Spark also supports what's called um wildcard parameters.

[00:08:46] So the previous parameters we talked about are named parameters.

[00:08:49] Every parameter has, has a name.

[00:08:51] In this case, these are called splat parameters.

[00:08:54] Now by splat they're talking about the asterisk character.

[00:08:56] So, so the asterisk character is used as a wild card.

[00:08:59] And what that means is that the asterisk will match any, any value that the caller puts into that position in the URL path.

[00:09:06] So really what this says is that there's two fixed components to this path, say and two and then there's two spots where the caller can pass in whatever they want.

Start visual description. The professor introduces wildcard parameters (splat parameters) and demonstrates how to use them in Spark routes. End visual description.

[00:09:16] And in this case, these parameters are not named.

[00:09:19] And so what spark will do is it'll take all the parameters that match the splat, and they put those parameters in a, in an array inside the request. So, there's a, there's a property inside the request called splat and that splat property is really just an array of all the parameters that match these spots in the URL in the incoming call.

[00:09:41] So you can use those, I think name parameters are usually a little more um useful, but there are cases where the splats are, are useful as well.

[00:09:54] Now, in general, as we've seen the request and response parameters that get passed to the http handler.

[00:10:04] Um These are objects that contain, you know, lots of useful stuff and I want us to talk a little bit more about what's inside of these objects that you can take advantage of.

[00:10:14] Now again, the request has all the information about the incoming http request.

Start visual description. The professor elaborates on the request and response objects in Spark, highlighting useful methods like body, header, and status for handling HTTP requests and responses. End visual description.

[00:10:19] The response has uh methods that you can use to build up an HTTP response and send it back to the caller.

[00:10:26] But um it turns out that there's a lot of useful methods on both of these classes that you want to be aware of.

[00:10:34] Um For example, on the request class, there's a method named body.

[00:10:38] So if the incoming request has a has a body on it, you'll want to access the contents of that body. And so you can, you can just call the body method on the request and it'll give you back a, a string that contains the contents of the request body.

[00:10:51] You can also call the header method which will give you back a, a map of strings, two sets of strings. So basically the keys in this map are the names of the headers and the values are um each header can actually have zero or more values.

[00:11:10] And so for each um header, there's a set of string values.

[00:11:15] Um And so those are probably the most useful things on the request other than getting the URL itself, which you can do also, you can get the URL out of it.

[00:11:25] Now, on the response, um they have a method called body as well and this is a method you can use to put the data that you want to return in the response body inside the response.

[00:11:36] And so you just call the body method on the response, pass it a string and then that string populates the re response body.

[00:11:42] You can also call the status method on the response, which is where you can return the status code or specify the status code that you want to return.

[00:11:49] So you can return, you know, 200 in most cases, but in failure cases, it'll be a 400 something or a 500 something.

[00:11:57] In this case, it's a 404.

[00:11:58] So that's where you can return um a status code. You can also return headers through this response as well. There's a header method on the response also.

[00:12:07] And so all of those uh methods on those classes give you access to all the information that you need.

[00:12:14] So we could uh go back to our code example and maybe implement it a little bit differently.

[00:12:23] So let's go to our um route here that currently just returns hello and let's um put it in a some curly braces here. So we can add more lines of code.

[00:12:44] So what I might do here instead of just returning um the string because by default in a http handler, whatever value you return from the handler, what spark will do is it will take the string that you return and it will put that in the response body.

[00:13:02] So whatever you return becomes a response body, they'll assume um that it, it succeeded.

[00:13:09] So they'll return a 200 status code and so forth.

[00:13:14] Now, if your handler throws an exception, then they'll return a failure of some sort.

[00:13:20] But by default, just return the response body and, and the right things happen.

[00:13:24] So, so spark does a lot of uh stuff for you.

[00:13:27] But let's say I wanted more control over what I'm returning to the client.

[00:13:31] Let let's not use the return in that way, let's use the response.

[00:13:36] So in this case, I could say response dot um status.

[00:13:42] So let's uh specify 200 ourselves.

[00:13:46] And of course, in failure cases, you'd want to specify something other than 200 a 401 or a 404 or something.

[00:13:53] Um What else? We're going to return the response body? So in this case, that's how we return the response body.

[00:14:17] Let, let's say we want to put some headers on this as well.

[00:14:21] Maybe um we want to return the type of data that's in the response body.

[00:14:28] You might not have noticed this. But um if I if I call our Hello endpoint, what you'll see here is that the server returns hello Frankenstein.

[00:14:42] But it also says that the type of this, this data that we returned is html.

[00:14:48] Now, technically, that's not what I'm returning.

[00:14:50] I'm really returning plain text, but by default Spark will return text slash html.

[00:14:56] So what I really want to do is when I return some data from an endpoint, I want to specify the, the real type of the data.

[00:15:03] So in this case, I could say response dot type.

[00:15:08] In this case, I'll say text slash plane and I could also specify other headers as well.

[00:15:19] Um Let's just make up a header. Let's have CS 240 be a header and we'll give it the value.

[00:15:26] Awesome.

[00:15:31] Now it turns out we still need to return a value from this, this handler method. So, the convention with Spark is that if you fill the body in with the body method up here, then you would just return the value that you already specified. So, I'm going to return response dot body in this case.

[00:15:51] So now if we rerun the server, OK, it still works. We still get hello Frankenstein back. We still get a 200.

[00:16:07] But you'll see now that it has the right content type. It's got text plan.

[00:16:11] It's also got the CS 240 header that we added.

[00:16:14] And so if you need more control over the http responses, there's a lot you can do with that response object.

[00:16:23] All right.

[00:16:30] All right. That's the same thing I just typed in.

[00:16:33] So I, I think that's um pretty much what you need to know about creating routes in, in, in spark.