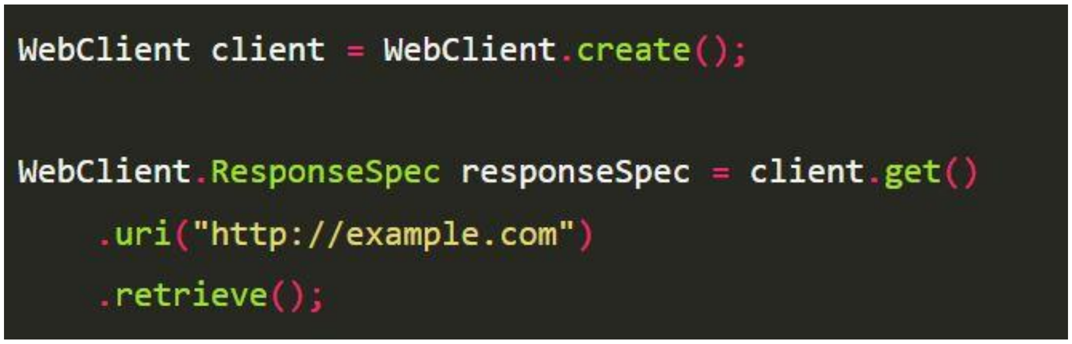
webclinet

In Spring 5, Spring gained a reactive web framework: Spring WebFlux. This is designed to co-exist alongside the existing Spring Web MVC APIs, but to add support for non-blocking designs. Using WebFlux, you can build asynchronous web applications, using reactive streams and functional APIs to better support concurrency and scaling.

As part of this, Spring 5 introduced the new `WebClient` API, replacing the existing `RestTemplate` client. Using `WebClient` you can make synchronous or asynchronous HTTP requests with a functional fluent API that can integrate directly into your existing Spring configuration and the WebFlux reactive framework.

To get started, you'll first need to add some dependencies to your project, if you don't have them already. If you're using Spring Boot you can use spring-boot-starter-webflux, or alternatively you can install spring-webflux and reactor-netty directly.



```
WebClient client = WebClient.create();

WebClient.ResponseSpec responseSpec = client.get()
    .uri("http://example.com")
    .retrieve();
```

This is everything required to send a request, but it's important to note that no request has actually been sent at this point! As a reactive API, the request is not actually sent until something attempts to read or wait for the response.

How to Handle an HTTP Response with WebClient

Once we've made a request, we usually want to read the contents of the response.

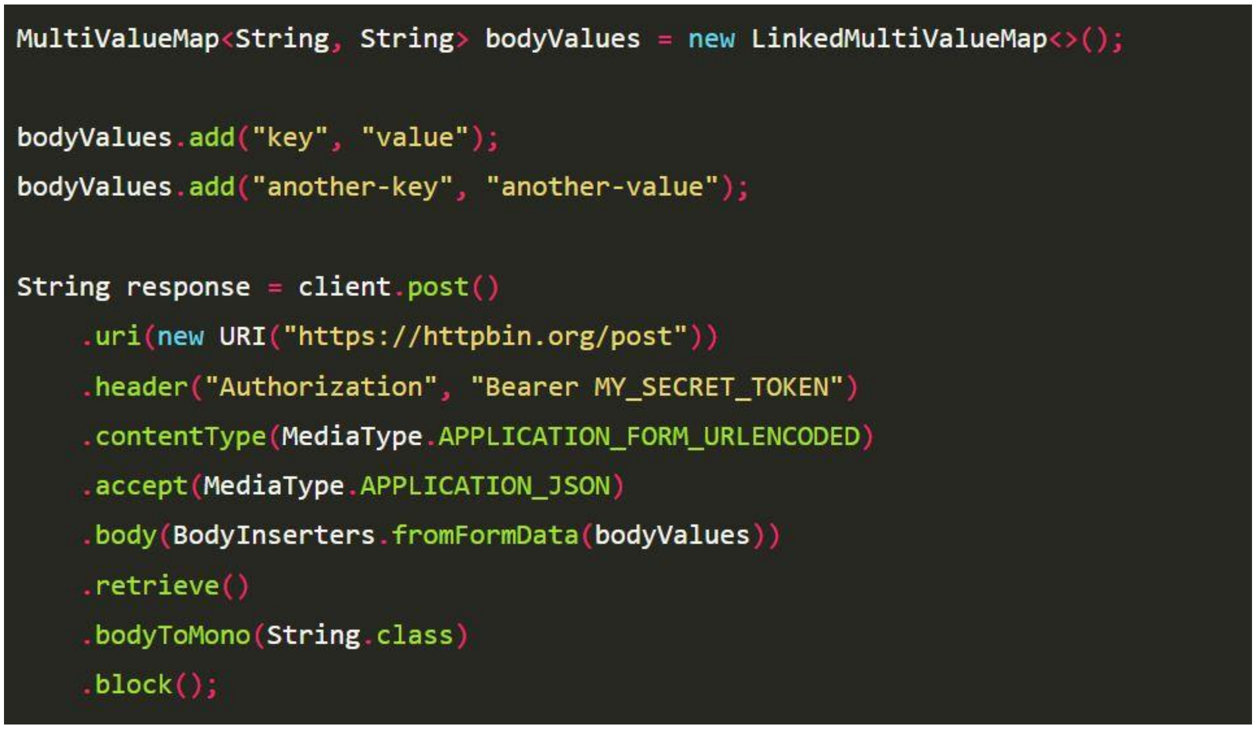
In the above example, we called `.retrieve()` to get a `ResponseSpec` for a request. This is an asynchronous operation, which doesn't block or wait for the request itself, which means that on the following line the request is still pending, and so we can't yet access any of the response details.

Reading the Body

To read the response body, we need to get a Mono (i.e: an async future value) for the contents of the response. We then need to unwrap that somehow, to trigger the request and get the response body content itself, once it’s available.

Note that we’re not checking the status here ourselves. When we use .retrieve(), the client automatically checks the status code for us, providing a sensible default by throwing an error for any 4xx or 5xx responses. We’ll talk about custom status checks & error handling later on too.

How to Send a Complex POST Request with WebClient



```
MultiValueMap<String, String> bodyValues = new LinkedMultiValueMap<>();

bodyValues.add("key", "value");
bodyValues.add("another-key", "another-value");

String response = client.post()
    .uri(new URI("https://httpbin.org/post"))
    .header("Authorization", "Bearer MY_SECRET_TOKEN")
    .contentType(MediaType.APPLICATION_FORM_URLENCODED)
    .accept(MediaType.APPLICATION_JSON)
    .body(BodyInserters.fromFormData(bodyValues))
    .retrieve()
    .bodyToMono(String.class)
    .block();
```

- Call the retrieve() or exchange() method. The retrieve() method directly performs the HTTP request and retrieve the response body.
  The exchange() method returns ClientResponse having the response status and headers.
- Else, use the method exchange() which will return the ClientResponse which has all the response elements such as status, headers and response body as well.

- Please note that bodyToMono() and bodyToFlux() methods always expect a response body of a given class type. If the response status code is 4xx (client error) or 5xx (Server error) i.e. there is no response body then these methods throw WebClientException.

retrieve() vs exchange()

While using the Spring WebClient, we have the option to use retrieve() or exchange() method. Let's look at the difference between these 2 methods to understand their use-cases.

- The retrieve() method should be preferred in case we are interested in response body.
- exchange method provides more control and details like status, headers and response body, etc.
- retrieve() method provides automatic error signal (e.g. 4xx and 5xx).
- No automatic error signal is available for exchange() method and we need to check status code and handle it.

Resources:

https://www.baeldung.com/spring-5-webclient

https://howtodoinjava.com/spring-webflux/webclient-get-post-example/

https://reflectoring.io/spring-webclient/

https://www.javadevjournal.com/spring/spring-webclient/