Reference links

<https://digma.ai/restclient-vs-webclient-vs-resttemplate/>

Create rest template from restTemplate builder bean which is automatically configured by spr boot

@Configuration

public **class** RestTemplateConfig **{**

@Bean

//Here RestTemplatedBuilder bean will be created by spr boot as part of auto config

public RestTemplate restTemplate**(**RestTemplateBuilder builder**)** **{**

**return** builder.build**()**;

**}**

**}**

In controller to return ResponseEntity use below

ResponseEntity.status(HttpStatus.OK).body(“);

## RestTemplate

Provides blocking calls

|  |  |
| --- | --- |
| @Configuration  public **class** RestTemplateConfig **{**  @Bean  public RestTemplate restTemplate**(**RestTemplateBuilder builder**)** **{**  **return** builder.build**()**;  **}**  **}** | @GetMapping**(**"/echo/{message}"**)**  **public** Echo echo**(**@PathVariable String message**)** **{**  **return** restTemplate.getForObject**(**"http://localhost:8080/echo/" + message, Echo.class**)**;  **}** |

RestTemplate provides several groups of overloaded methods based on the Spring template pattern:

* XXXForObject (most straightforward)
* XXXForEntity (represent more information)
* exchange (more generalized with extra flexibility)
* execute (most generalized with full control)

Like any library, RestTemplate has its pros and cons. knowing about them will help us to make the best of it and also allow us to compare it with two other libraries in the following:

**Pros:**

* Switchable underlying HTTP client library
* Supports declarative HTTP interface
* Highly configurable
* Usable in older versions of Spring Framework

**Cons:**

* Not suitable for **non-blocking** environments (for example, WebFlux)
* Having **multiple overloads** of a method makes this library hard to use
* The classic **Spring template pattern is old-fashioned**

## WebClient

For non-blocking & async calls use webclient & we can use this for blocking calls also by calling .block() method

RestTemplate is a simple and flexible library to call HTTP services, but it was **synchronous** and **blocking,** and it was the **main reason** that The Spring non-blocking stack (**WebFlux**) introduced a new and modern and fully non-block and Asynchronous HTTP client library with functional and fluent API called WebClient.

WebClient was introduced and designed specifically for the **WebFlux stack.**Unlike RestTemplate, it was not based on an old-fashioned template-based API and followed the modern **functional** and **fluent style API**.

The WebClient API is much clearer than the RestTemplate, and they are equivalent to HTTP methods.

|  |  |
| --- | --- |
| @Configuration  **public** **class** WebClientConfig **{**  @Bean  **public** WebClient webClient**(**WebClient.Builder builder**)** **{**  **return** builder  .baseUrl**(**"http://localhost:8080"**)**  .build**()**;  **}**  **}** | @GetMapping**(**"/echo/{message}"**)**  **public** Mono**<**Echo**>** echo**(**@PathVariable String message**)** **{**  **return** webClient  .get**()**  .uri**(**"/echo/" + message**)**  .retrieve**()**  .bodyToMono**(**Echo.class**)**;  **}** |

As I said before, WebClient was designed for **WebFlux**, and the API and threading model differs from the RestTemplate. WebClient was introduced in **Spring Framework 5**, although it was designed for Spring **WebFlux** but heavily used in the blocking stack (**Web MVC**) by developers by calling the block() operation from WebClient:

@GetMapping**(**"/echo/{message}"**)**

**public** Echo echo**(**@PathVariable String message**)** **{**

**return** webClient

.get**()**

.uri**(**"/echo/" + message**)**

.retrieve**()**

.bodyToMono**(**Echo.class**)**

.block**()**;

Pros

*With this u can get both blocking and non-blocking behaviour*

In [**this article**](https://itnext.io/using-webclient-synchronously-in-spring-web-mvc-stack-8c5057eae008), I describe in detail that, technically, there is not any problem with using WebClient in the Web MVC stack and calling the block() operation:

*When we call the block() method on the WebClient return type, it will****block the calling thread from the WebMVC thread pool****, and as a result, the WebClient can continue to call external services or APIs asynchronous and non-blocking.*

*Cons:-*

In this way, you can have all the benefits of the WebClient API and its infrastructure (like non-blocking, performance and streaming support, and more),

but the cost is to add **extra library dependency** to your Web MVC project (WebFlux and Project Reactor)

since resttemplate is deprecated, if u are still using the webclient for blocking calls then its waste to add all these above non-blocking jars Hence

if ur requirement is blocking API and if u don’t want rest template then prefer RESTClient

## RestClient

RestClient was introduced in **Spring Framework 6.1** and has the same infrastructure and abstraction as RestTemplate, which means it is blocking, but it has a fluent API similar to WebClient

If you don’t want to have the Spring WebFlux dependency in your Spring MVC project, RestClient is a good choice, and keep the codes in your project unmixed.

Migrate from RestTemplate to RestClient

The Spring Framework team recommends using RestClient for the new Spring MVC project and also provides guidelines to **migrate** from RestTemlate to RestClient.

This is also purely blocking same like rest template

|  |  |
| --- | --- |
| @Bean  **public** RestClient restClient**()** **{**  **return** RestClient.builder**()**  .baseUrl**(**"http://localhost:8080"**)**  .build**()**;  **}** | @GetMapping**(**"/echo/{message}"**)**  **public** Echo echo**(**@PathVariable String message**)** **{**  **return** restClient  .get**()**  .uri**(**"/echo/" + message**)**  .retrieve**()**  .body**(**Echo.class**)**;  **}** |

As you can see in this example, the RestClient API is almost identical to the WebClient API, except we don’t need to have the **WebFlux** library in our dependency and also call the block() method!

Read [**this part of Spring Framework reference documentation**](https://docs.spring.io/spring-framework/reference/integration/rest-clients.html#_migrating_from_resttemplate_to_restclient) to learn more about migrating from RestTemplate to RestClient.

# Creating RESTful apis

|  |  |
| --- | --- |
| to upload a file | @RestController  public class FileUploadController {  @PostMapping("/upload")  public String uploadFile(@RequestParam("file") MultipartFile file) {  // You can process the file here  String filename = file.getOriginalFilename();  // Save file, handle file logic, etc.  return "File uploaded successfully: " + filename;  }  } |

Annotations

|  |  |
| --- | --- |
| @RequestParam  its like  @QueryParam in apache cxf | <http://localhost:8080/openai/query?search=is%20god%20exists>  @GetMapping("/query")  public String hitChatGPTLLM(@RequestParam("search") String *request*) {  System.*out*.println("received query is " + *request*);  *ChatClient*.*CallResponseSpec* call = chatClient.prompt(*request*).call();  return call.content();  } |
|  |  |