**1. RestTemplate class**

Accessing a third-party REST service inside a Spring application revolves around the use of the Spring [**RestTemplate**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html) class. The RestTemplate class is designed on the same principles as the many other Spring \*Template classes (e.g., JdbcTemplate, JmsTemplate ), providing a simplified approach with default behaviors for performing complex tasks.

Given that the RestTemplate class is **designed to call REST services**, It should come as no surprise that its main methods are closely tied to REST’s underpinnings, which are the HTTP protocol’s methods: **HEAD**, **GET**, **POST**, **PUT**, **DELETE**, and **OPTIONS**.

Read More: [RestTemplate Configuration with HttpClient](https://howtodoinjava.com/spring/spring-restful/resttemplate-httpclient-java-config/)

Spring docs recommend to use the non-blocking, reactive WebClient which offers efficient support for both sync, async and streaming scenarios. RestTemplate will be deprecated in a future version.

**2. Spring RestTemplate – HTTP GET Method Example**

Supported methods are:

* getForObject(url, classType) – retrieve a representation by doing a GET on the URL. The response (if any) is unmarshalled to given class type and returned.
* getForEntity(url, responseType) – retrieve a representation as *ResponseEntity* by doing a GET on the URL.
* exchange(requestEntity, responseType) – execute the specified request and return the response as *ResponseEntity*.
* execute(url, httpMethod, requestCallback, responseExtractor) – execute the *httpMethod* to the given URI template, preparing the request with the *RequestCallback*, and reading the response with a *ResponseExtractor*.

**2.1. XML Response**

**REST API Code**

Spring REST API for HTTP GET method.

|  |
| --- |
| @RequestMapping(value = "/employees", produces = MediaType.APPLICATION\_XML\_VALUE, method = RequestMethod.GET)  public String getAllEmployeesXML(Model model)  {      model.addAttribute("employees", getEmployeesCollection());      return "xmlTemplate";  } |

**REST Client Code**

Spring REST client using RestTemplate to access HTTP GET api requests.

|  |
| --- |
| private static void getEmployees()  {      final String uri = "<http://localhost:8080/springrestexample/employees.xml>";        RestTemplate restTemplate = new RestTemplate();      String result = restTemplate.getForObject(uri, String.class);        System.out.println(result);  } |

**2.2. JSON Response**

**REST API Code**

|  |
| --- |
| @RequestMapping(value = "/employees", produces = MediaType.APPLICATION\_JSON\_VALUE,  method = RequestMethod.GET)  public String getAllEmployeesJSON(Model model)  {      model.addAttribute("employees", getEmployeesCollection());      return "jsonTemplate";  } |

**REST Client Code**

|  |
| --- |
| private static void getEmployees()  {      final String uri = "<http://localhost:8080/springrestexample/employees.json>";        RestTemplate restTemplate = new RestTemplate();      String result = restTemplate.getForObject(uri, String.class);        System.out.println(result);  } |

**2.3. Custom HTTP Headers with RestTemplate**

**REST API Code**

|  |
| --- |
| @RequestMapping(value = "/employees", produces = MediaType.APPLICATION\_JSON\_VALUE,  method = RequestMethod.GET)  public String getAllEmployeesJSON(Model model)  {      model.addAttribute("employees", getEmployeesCollection());      return "jsonTemplate";  } |

**REST Client Code**

|  |
| --- |
| private static void getEmployees()  {      final String uri = "<http://localhost:8080/springrestexample/employees>";        RestTemplate restTemplate = new RestTemplate();        HttpHeaders headers = new HttpHeaders();      headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));      HttpEntity<String> entity = new HttpEntity<String>("parameters", headers);        ResponseEntity<String> result = restTemplate.exchange(uri, HttpMethod.GET, entity, String.class);        System.out.println(result);  } |

**2.4. Get response as object**

**REST API Code**

|  |
| --- |
| @RequestMapping(value = "/employees", produces = MediaType.APPLICATION\_XML\_VALUE, method = RequestMethod.GET)  public String getAllEmployeesXML(Model model)  {      model.addAttribute("employees", getEmployeesCollection());      return "xmlTemplate";  } |

**REST Client Code**

|  |
| --- |
| private static void getEmployees()  {      final String uri = "<http://localhost:8080/springrestexample/employees>";      RestTemplate restTemplate = new RestTemplate();        EmployeeListVO result = restTemplate.getForObject(uri, EmployeeListVO.class);        System.out.println(result);  } |

**2.5. URL Parameters**

**REST API Code**

|  |
| --- |
| @RequestMapping(value = "/employees/{id}")  public ResponseEntity<EmployeeVO> getEmployeeById (@PathVariable("id") int id)  {      if (id <= 3) {          EmployeeVO employee = new EmployeeVO(1,"Lokesh","Gupta","howtodoinjava@gmail.com");          return new ResponseEntity<EmployeeVO>(employee, HttpStatus.OK);      }      return new ResponseEntity(HttpStatus.NOT\_FOUND);  } |

**REST Client Code**

|  |
| --- |
| private static void getEmployeeById()  {      final String uri = "<http://localhost:8080/springrestexample/employees/>{id}";        Map<String, String> params = new HashMap<String, String>();      params.put("id", "1");        RestTemplate restTemplate = new RestTemplate();      EmployeeVO result = restTemplate.getForObject(uri, EmployeeVO.class, params);        System.out.println(result);  } |

**3. Spring RestTemplate – HTTP POST Method Example**

Supported methods are:

* postForObject(url, request, classType) – POSTs the given object to the URL, and returns the representation found in the response as given class type.
* postForEntity(url, request, responseType) – POSTs the given object to the URL, and returns the response as *ResponseEntity*.
* postForLocation(url, request, responseType) – POSTs the given object to the URL, and returns returns the value of the Location header.
* exchange(url, requestEntity, responseType)
* execute(url, httpMethod, requestCallback, responseExtractor)

**REST API Code**

Spring REST API for HTTP POST method.

|  |
| --- |
| @RequestMapping(value = "/employees", method = RequestMethod.POST)  public ResponseEntity<String> createEmployee(@RequestBody EmployeeVO employee)  {      System.out.println(employee);      return new ResponseEntity(HttpStatus.CREATED);  } |

**REST Client Code**

Spring REST client using RestTemplate to access HTTP POST api requests.

|  |
| --- |
| private static void createEmployee()  {      final String uri = "<http://localhost:8080/springrestexample/employees>";        EmployeeVO newEmployee = new EmployeeVO(-1, "Adam", "Gilly", "test@email.com");        RestTemplate restTemplate = new RestTemplate();      EmployeeVO result = restTemplate.postForObject( uri, newEmployee, EmployeeVO.class);        System.out.println(result);  } |

**4. Spring RestTemplate – HTTP PUT Method Example**

Supported methods are:

* put(url, request) – PUTs the given request object to URL.

**REST API Code**

Spring REST API for HTTP PUT method.

|  |
| --- |
| @RequestMapping(value = "/employees/{id}", method = RequestMethod.PUT)  public ResponseEntity<EmployeeVO> updateEmployee(@PathVariable("id") int id, @RequestBody EmployeeVO employee)  {      System.out.println(id);      System.out.println(employee);      return new ResponseEntity<EmployeeVO>(employee, HttpStatus.OK);  } |

**REST Client Code**

Spring REST client using RestTemplate to access HTTP PUT api requests.

|  |
| --- |
| private static void updateEmployee()  {      final String uri = "<http://localhost:8080/springrestexample/employees/>{id}";        Map<String, String> params = new HashMap<String, String>();      params.put("id", "2");        EmployeeVO updatedEmployee = new EmployeeVO(2, "New Name", "Gilly", "test@email.com");        RestTemplate restTemplate = new RestTemplate();      restTemplate.put ( uri, updatedEmployee, params);  } |

**5. Spring RestTemplate – HTTP DELETE Method Example**

Supported methods are:

* delete(url) – deletes the resource at the specified URL.

**REST API Code**

Spring REST API for HTTP DELETE method.

|  |
| --- |
| @RequestMapping(value = "/employees/{id}", method = RequestMethod.DELETE)  public ResponseEntity<String> updateEmployee(@PathVariable("id") int id)  {      System.out.println(id);      return new ResponseEntity(HttpStatus.OK);  } |

**REST Client Code**

Spring REST client using RestTemplate to access HTTP DELETE api requests.

|  |
| --- |
| private static void deleteEmployee()  {      final String uri = "<http://localhost:8080/springrestexample/employees/>{id}";        Map<String, String> params = new HashMap<String, String>();      params.put("id", "2");        RestTemplate restTemplate = new RestTemplate();      restTemplate.delete ( uri,  params );  } |

Feel free to copy and modify above **Spring RestTemplate** examples for building the **Spring REST client** in your MVC application

# Spring RestTemplateBuilder Example

Learn to use RestTemplateBuilder to create or build [RestTemplate](https://howtodoinjava.com/spring/spring-restful/resttemplate-httpclient-java-config/) class which can be used to send HTTP requests.

## 1) Default RestTemplateBuilder

To inject RestTemplateBuilder, pass it as constructor argument in your service class.

|  |
| --- |
| MyService.java |
| @Service  public class MyService {        private final RestTemplate restTemplate;        public MyService(RestTemplateBuilder restTemplateBuilder) {          this.restTemplate = restTemplateBuilder.build();      }        public Details someRestCall(String name) {          return this.restTemplate.getForObject("/{name}/details", Details.class, name);      }  } |

## 2) Custom RestTemplateBuilder

To create custom RestTemplateBuilder, create @Bean of type RestTemplateBuilder in Spring context.

#### RestTemplateBuilder Bean

|  |
| --- |
| RestTemplateConfig.java |
| @Configuration  public class RestTemplateConfig {        @Bean      @DependsOn(value = {"customRestTemplateBuilder"})      public RestTemplateBuilder restTemplateBuilder()      {          return new RestTemplateBuilder(customRestTemplateCustomizer());      }        @Bean      public CustomRestTemplateCustomizer customRestTemplateCustomizer()      {          return new CustomRestTemplateCustomizer();      }  } |

#### RestTemplateCustomizer Bean

Create custom RestTemplateCustomizer as per your requirements.

|  |
| --- |
| CustomRestTemplateCustomizer.java |
| public class CustomRestTemplateCustomizer implements RestTemplateCustomizer  {      @Override      public void customize(RestTemplate restTemplate)      {          restTemplate.setRequestFactory( new BufferingClientHttpRequestFactory(clientHttpRequestFactory()) );          restTemplate.setMessageConverters(Collections.singletonList(mappingJacksonHttpMessageConverter()));          restTemplate.setInterceptors(Collections.singletonList(new RequestResponseLoggingInterceptor()));      }  } |

#### Inject custom RestTemplateBuilder

And inject this custom RestTemplateBuilder to your service class as given in first section i.e. similar to default RestTemplateBuilder.

|  |
| --- |
| MyService.java |
| @Service  public class MyService {        private final RestTemplate restTemplate;        public MyService(RestTemplateBuilder restTemplateBuilder) {          this.restTemplate = restTemplateBuilder.build();      }  } |