**Feign Client in Spring Boot-2024**

# **Introduction**

[Feign](https://github.com/Netflix/feign) is a declarative web service client. It makes writing web service clients easier. To use Feign create an interface and annotate it. It has pluggable annotation support including Feign annotations and JAX-RS annotations.

**How does Feign work?**

Feign works by processing annotations into a templatized request. Arguments are applied to these templates in a straightforward fashion before output. Although Feign is limited to supporting text-based APIs, it dramatically simplifies system aspects such as replaying requests. Furthermore, Feign makes it easy to unit test your conversions knowing this.

Pom.xml

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**Main Application**

@SpringBootApplication

**@EnableFeignClients 🡸 Important and Required**

**public** **class** SpringFeignClientApp {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringFeignClientApp.**class**, args);

}

}

The following REST calls were implemented using Feign Client.

DELETE <http://localhost:8081/sample/api/empinfo>

PUT <http://localhost:8081/sample/api/sal>

POST <http://localhost:8081/sample/api/emp>

GET <http://localhost:8081/sample/api/id/123>

How to create Feign Client

@FeignClient(name = "empInfo", url = "http://localhost:8081/sample")

**public** **interface** EmpFeignClient {

**@GetMapping**(path = "/api/id/{id}")

String getEmpByIdAsString(@PathVariable **int** id);

**@GetMapping**(path = "/api/id/{id}")

Employee getEmpByIdAsObject(@PathVariable **int** id);

**@PostMapping**(path = "api/emp")

String createEmpInfo(@RequestBody Employee emp);

**@PutMapping**(path = "api/sal")

String updateEmpSal(@RequestBody Employee emp,

**@RequestHeader**(value = "key", required = **false**) String key,

**@RequestHeader**(value = "org", required = **false**) String org);

**@DeleteMapping**(path = "api/empinfo")

String deleteEmpInfo(@RequestBody Employee emp,

@RequestHeader(value = "key", required = **false**) String key,

@RequestHeader(value = "org", required = **false**) String org);

}

**Service Implementation Class**

@Service

**public** **class** EmpServiceImpl {

@Autowired

**private** EmpFeignClient empClient;

**public** **void** showEmpById(**int** id) {

String response = empClient.getEmpByIdAsString(id);

System.***out***.println("Emp By Id Response: "+response);

Employee emp = empClient.getEmpByIdAsObject(id);

System.***out***.println("Emp Resonse as Object: "+emp);

}

**public** **void** createEmpInfo(Employee emp) {

String response = empClient.createEmpInfo(emp);

System.***out***.println("Response : "+response);

}

**public** **void** updateEmpSalary(Employee emp) {

String orgHeader = "org";

String keyHeader = "someKey";

String response = empClient.updateEmpSal(emp,keyHeader, orgHeader);

System.***out***.println("Response : "+response);

}

**public** **void** deleteEmpInfo(Employee emp) {

String orgHeader = "org";

String keyHeader = "someKey";

String response = empClient.deleteEmpInfo(emp,keyHeader, orgHeader);

System.***out***.println("Response : "+response);

}

}

How to Pass dynamic BaseURI in FeignClient ?

Sometimes, it is required to pass the dynamic BaseURI in FeignClient. The code is given below.

@FeignClient(name = "anotherEmpClient",

url = "https://this-is-a-dummy-url-or-placeholder.com") 🡸 Dummy URL

**public** **interface** AnotherEmpFeignClient {

@GetMapping(path = "/api/id/{id}")

Employee getEmpByIdAsObject(**URI baseUri**, @PathVariable **int** id);

@PostMapping(path = "api/emp")

String createEmpInfo(**URI baseUri**, @RequestBody Employee emp);

}

Sample code mentioned in AutoRun class is given below.

**private** **void** showEmpObject() {

URI baseURI = URI.*create*("http://localhost:8081/sample");

Employee emp = feignClient.getEmpByIdAsObject(baseURI, 123);

System.***out***.println("Emp Resonse as Object: " + emp);

}

**public** **void** createEmpInfo() {

Employee emp = **new** Employee(12345, "Vidya", "Permanent", **new** String[] {"Admin"});

URI baseURI = URI.*create*("http://localhost:8081/sample");

String response = feignClient.createEmpInfo(baseURI, emp);

System.***out***.println("Response : "+response);

}

**FeignClient Fallback with Resilience4J - 2024**

To handle fallback in FeignClient, you have to use Resilience4J as circuit breaker along with Feign Client.

**pom.xml** relevant portion except feign details.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>

</dependency>

**application.properties**

server.port=8085

# The following property is important for FeignCLient CricuitBreaker

**spring.cloud.openfeign.circuitbreaker.enabled=true 🡨 Important**

**FeignClient class**

@FeignClient(name = "SomeFeignClientName",

url = "http://localhost:8081/sample",

**fallback = FeignClientFallBack.class**)

**public** **interface** **SampleFeignClient** {

@GetMapping("/v1/emp/{id}")

String getEmpById(@PathVariable("id") **int** id);

// For handling exception

@GetMapping("/v1/country/{countryCode}")

String getCountryDetailByCode(@PathVariable("countryCode") String countryCode);

}

**FallBack class**

@Component

**public** **class** FeignClientFallBack **implements** **SampleFeignClient** {

@Override

**public** String getEmpById(**int** id) {

**return** "No Response from server ...";

}

@Override

**public** String getCountryDetailByCode(String countryCode) {

**return** "Exception received from external service ...";

}

}

**AutoRun class**

@Component

**public** **class** AutoRun {

@Autowired

**private** SampleFeignClient client;

**private** **void** getEmpDetails() {

String response = client.getEmpById(124);

System.***out***.println("Feign Client Response: "+response);

}

**private** **void** getCountryDetails() {

String response = client.getCountryDetailByCode("a124");

System.***out***.println("Feign Client Response: "+response);

}

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("App started running ...");

// getEmpDetails();

getCountryDetails();

}

}

**How to Handle Exception in Feign Client – 2024**

**Step-1: Create a class by implementing feign.codec.ErrorDecoder**

**public** **class** CustomErrorDecoder **implements** ErrorDecoder {

@Override

**public** Exception decode(String methodKey, Response response) {

String requestUrl = response.request().url();

Response.Body responseBody = response.body();

HttpStatus responseStatus = HttpStatus.*valueOf*(response.status());

System.***out***.println("requestUrl---->" + requestUrl);

System.***out***.println("responseBody---->" + responseBody);

System.***out***.println("responseStatus---->" + responseStatus);

**switch** (response.status()) {

**case** 400:

// Handle 400 Bad Request

**return** **new** ApplicationBadRequestException("Bad Request");

**case** 404:

// Handle 404 Not Found

**return** **new** NotFoundException("Not Found");

**case** 500:

// Handle 500 Internal Server Error

**return** **new** InternalServerErrorException("Internal Server Error");

**default**:

**return** **new** Exception("Generic error");

}

}

}

**Step-2: Create a Feign Configuration Class.**

@Configuration

**public** **class** FeignClientConfiguration {

@Bean

**public** Feign.Builder feignBuilder() {

**return** Feign.*builder*().errorDecoder(**new** CustomErrorDecoder());

// Other Feign configurations can be chained here if needed

}

}

**Step-3: Create a Set of Exception classes like this.**

**public** **class** ApplicationBadRequestException **extends** RuntimeException {

**public** ApplicationBadRequestException(String message) {

**super**(message);

}

}

You can create other Exception classes like InternalServerErrorException, NotFoundException

**Step-4: Create a Global Exception Handler by annotating @ExceptionHandler or @RestExceptionHandler**.

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler({ ApplicationBadRequestException.**class** })

**public** ResponseEntity<String> handleBadRequestException(ApplicationBadRequestException exception,

WebRequest request) {

System.***out***.println("----------- Called ------------");

**return** **new** ResponseEntity<String>("Some unwanted application Error", HttpStatus.***INTERNAL\_SERVER\_ERROR***);

}

}

**Step-5: Create a Feign Client interface like this**.

@FeignClient(name = "AnotherFeignClientName", url = "http://localhost:8081/sample")

**public** **interface** SampleFeignClient {

// This method is used when external service is throwing exception

@GetMapping("/v1/country/{countryCode}")

String getCountryDetailByCode(@PathVariable("countryCode") String countryCode);

}

You can define a sample **controller** like this.

@RestController

**public** **class** TempController {

@Autowired

**private** SampleFeignClient anotherClient;

**private** **void** getMoreEmpDetails() {

String response = anotherClient.getCountryDetailByCode("a124");

System.***out***.println("Another Feign Client Response: "+response);

}

@GetMapping(path = "/hi")

**public** ResponseEntity<String> getSomeValue() {

getMoreEmpDetails();

**return** **new** ResponseEntity<String>("someValue", HttpStatus.***OK***);

}

}

**AutoRun class is given below**.

@Component

**public** **class** AutoRun {

@Autowired

**private** SampleFeignClient anotherClient;

**private** **void** getMoreEmpDetails() {

String response = "";

**try** {

response = anotherClient.getCountryDetailByCode("a124");

} **catch**(Exception ex) {

System.***err***.println("Handling exception--->"+ex.getMessage());

}

System.***out***.println("Another Feign Client Response: "+response);

}

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("Application started running ...");

getMoreEmpDetails();

}

}