Additional REST Service Techniques

1. Consuming REST services
2. Microservice architecture
3. Circuit breakers

1. Consuming REST Services

- Overview
- How to use `RestClient`
- Example
- Other ways to consume a REST service
- Consuming a REST service from HTML

Overview

- Spring Web provides 2 ways to consume REST services:
  - `RestClient` – fluent coding style, recommended
  - `RestTemplate` – traditional approach, not recommended

- In either case, you must include the following dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

pom.xml

- We'll concentrate on using `RestClient`
  - See following slides…

How to Use `RestClient`

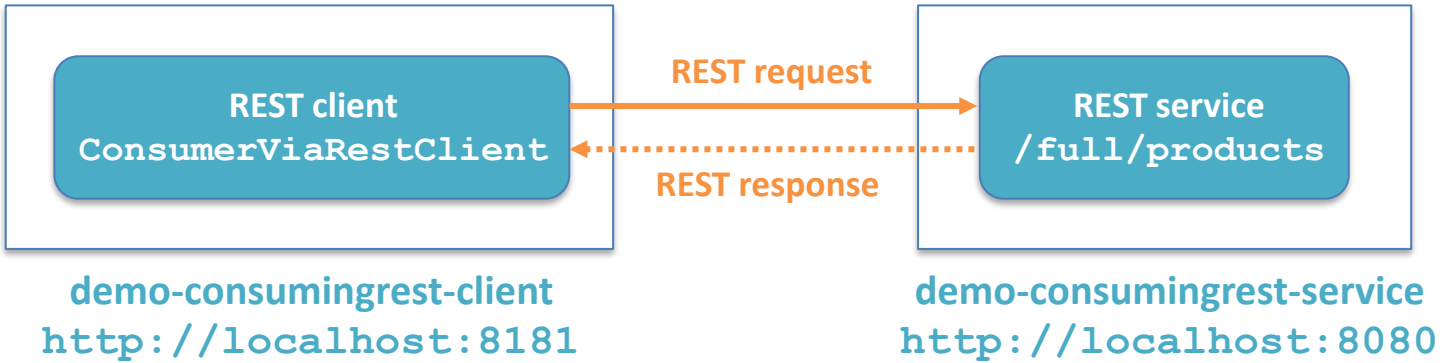
- Here's an example of how to use `RestClient` for a GET API:

```
RestClient client = RestClient.create();
String someUri = "http://localhost:8080/full/products/1";

ResponseEntity<Product> response = client
                                        .get()
                                        .uri(someUri))
                                        .retrieve()
                                        .toEntity(Product.class);

System.out.printf("\nGot product 1, response details:");
System.out.printf("  Status code: %d\n", response.getStatusCodeValue());
System.out.printf("  Response body: %s\n", response.getBody());
```

- There are similar methods for other types of requests
  - `post()`
  - `put()`
  - `delete()`

Example

- Let's see a complete example of using `RestClient`:



- See the code here:
  - Project: `demo-consumingrest-client`
  - Class: `ConsumerViaRestClient.java`

Other Ways to Consume a REST Service

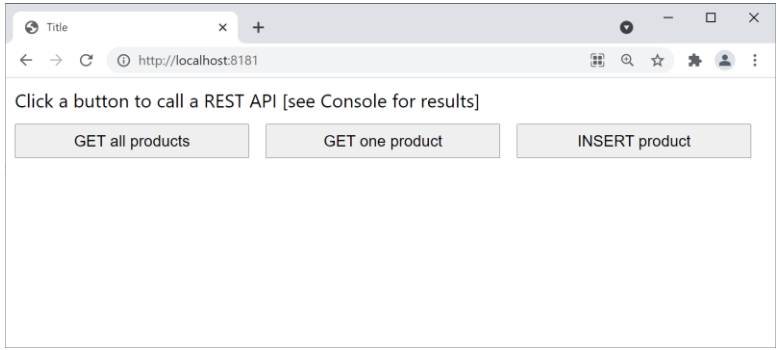
- Via `RestTemplate`
  - This is the original way to consume a REST service
  - Not as elegant as `RestClient`, but widely used in the past
  - See `ConsumerViaRestTemplate.java`

- Via `WebClient`
  - Can be synchronous or asynchronous
  - Well suited to calling reactive REST services (WebFlux)
  - See `ConsumerViaWebClient.java` and `pom.xml`

Consuming a REST Service from HTML

- We've also implemented a simple HTML page to show how to consume a REST service from a web UI
  - Project: `demo-consumingrest-client`
  - Folder: `src/main/resources/static`

- Open a browser and browse to `http://localhost:8181`



2. Microservice Architecture

- What are microservices
- Microservice application example
- Implementing the catalog service
- Implementing the client service

What are Microservices?

- According to Wiki:

Microservices is a specialisation of an implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable software systems.

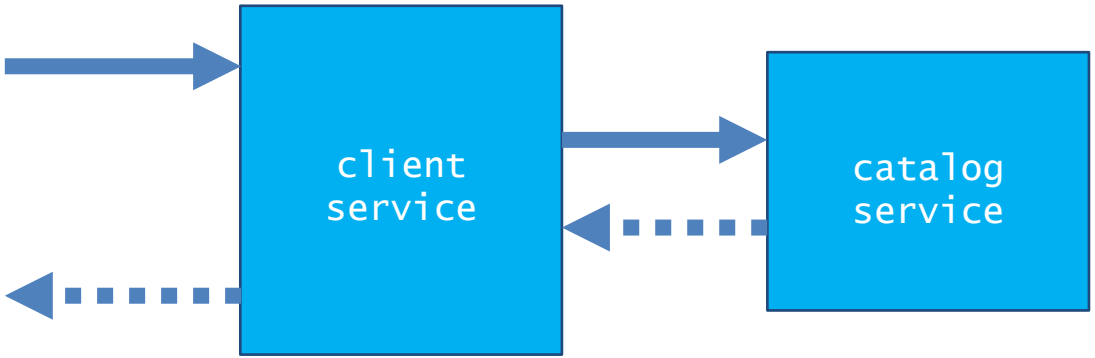
Services in a **microservice architecture (MSA)** are processes that communicate with each other over a network in order to fulfil a goal. These services use technology-agnostic protocols.

The microservices approach is a first realisation of SOA that followed the introduction of DevOps and is becoming more popular for building continuously deployed systems.

Microservice Application Example

- Let's see a complete (simple) example of how to create a microservice application

- There are two Spring Boot applications in the demo:
  - `demo-msa-clientservice`
  - `demo-msa-catalogservice`



Implementing the Catalog Service (1 of 2)

- The "catalog" service is a Spring Boot application with a REST service that returns catalog info
  - See `demo-msa-catalogservice`
  - The `server.port` property is `8081`

- Take a look at the endpoints in `CatalogController`:
  - `/catalog`
  - `/catalog/{index}`

Implementing the Catalog Service (2 of 2)

- Run the catalog app and ping the following URLs…

http://localhost:8081/catalog


```
[
    "Bugatti Divo",
    "Lear Jet",
    "Socks from M&S"
]
```

http://localhost:8081/catalog/0

```
Bugatti Divo
```

Implementing the Client Service (1 of 3)

- The "client" service is another Spring Boot application with a REST service
  - See `demo-msa-clientservice`
  - The `server.port` property is `8080`

- Take a look at the endpoint in `ClientController`:
  - `/client/{index}`

Implementing the Client Service (2 of 3)

- The "client" service invokes the "catalog" service

HTTP request

```
@RestController
public class ClientController {

    @GetMapping("/client/{index}")
    public String getItem(@PathVariable int index){
        String result = getData(index);
        return String.format("[%s] Item %d %s", LocalTime.now(), index, result);
    }

    public String getData(int index) {
        URI catalogUri = URI.create("http://localhost:8081/catalog/" + index);
        RestClient client = RestClient.create();
        return client
                .get()
                .uri(catalogUri)
                .retrieve()
                .body(String.class);
    }
}
```

Catalog service

Implementing the Client Service (3 of 3)

- Run the client app and ping the following URL…
  - `http://localhost:8080/client/0`

3. Circuit Breakers

- Overview
- Circuit breakers in Spring Cloud
- Spring Cloud circuit breaker dependency
- Spring Cloud circuit breaker example
- Seeing a circuit breaker in action

Overview

- In a microservice application, services call other services
  - E.g. ServiceA calls ServiceB, ServiceB calls ServiceC, etc.

- If any service is down, you get a ripple effect of failures
  - E.g. if ServiceC is down…
  - Then ServiceB will fail (because it depends on ServiceC)
  - Then ServiceA will fail (because it depends on ServiceB), etc.

- To avoid the ripple effect of failures, use a **circuit breaker**
  - Specify a fallback method that can be called, if a service fails

Circuit Breakers in Spring Cloud

- Spring Cloud provides a circuit breaker API
  - Via the `CircuitBreakerFactory` class

- `CircuitBreakerFactory` is an abstraction over various circuit breaker implementations, including:
  - Resilience4J (we'll use this)
  - Netflix Hystrix
  - Sentinel
  - Spring Retry

Spring Cloud Circuit Breaker Dependency

- To use the Resilience4J circuit breaker implementation, add the following dependency to the pom file in your (client) project :

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
    <version>3.1.0</version>
</dependency>
```

pom.xml (client project)

- Once you've added this dependency, Spring Boot autoconfig will automatically create a Resilience4J bean
  - This bean is exposed via `CircuitBreakerFactory`
  - See next slide for an example of how to use a circuit breaker…

Spring Cloud Circuit Breaker Example

HTTP request

```
@RestController
public class ClientWithFallbackController {

   @Autowired
   private CircuitBreakerFactory factory;

    @GetMapping("/clientWithFallback/{index}")
    public String getItem(@PathVariable int index){

        CircuitBreaker circuitBreaker = factory.create("circuitbreaker");

        String result = circuitBreaker.run(
                ()  -> getData(index),
                err -> getFallbackData(index));

        return String.format("[%s] Item %d %s", LocalTime.now(), index, result);
    }

    public String getData(int index) { … }

    public String getFallbackData(int index) {return "FALLBACK-DATA-" + index;}
}
```

Catalog service

Seeing a Circuit Breaker in Action

- To see the effect of the circuit breaker, follow these steps:
  - Stop the catalog service
  - Then ping the following client endpoints…

http://localhost:8080/client/0

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Jan 18 14:02:46 GMT 2024
There was an unexpected error (type=Internal Server Error, status=500).

http://localhost:8080/clientWithFallback/0

[14:03:23.402472500] Item 0 FALLBACK-DATA-0

Summary

- Consuming REST services
- Microservice architecture
- Circuit breakers