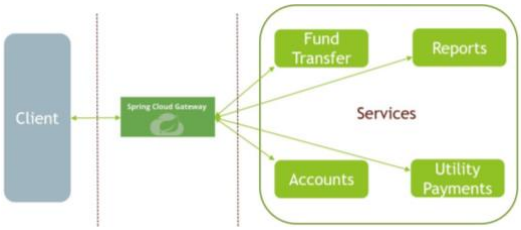
Gateway Server Einführung

Microservices



Beschreibung

Spring Cloud Gateway Server (Edge Server) mit Eureka einrichten und konfigurieren.

Inhaltsverzeichnis

1 Gateway Server

1.1 Einleitung

https://cloud.spring.io/spring-cloud-gateway/reference/html

1.2 Spring Cloud Gateway

Spring Cloud Gateway ist ein Open-Source-Gateway für das Erstellen von API-Gateways in verteilten Microservices-Architekturen. Es ist Teil des Spring Cloud-Projekts und bietet eine flexible Lösung zur Steuerung des Datenverkehrs zwischen den externen Clients und den internen Services. Durch den Einsatz des Gateways kommuniziert ein externer Client nie direkt mit einem internen Service.

Spring Cloud Gateway ist eng mit anderen Spring-Cloud-Projekten integriert, wie z.B. mit dem Eureka-Server für das Service Discovery und die Spring Security mit ihren Sicherheitsfunktionen.

Ein Gateway ermöglicht es, den Datenverkehr zwischen Client-Anfragen und verschiedenen Backend-Diensten zu steuern. Es kann als Einstiegspunkt für deine Microservices-Anwendungen dienen und bietet Funktionen wie Routing, Lastverteilung, Protokollierung, Authentifizierung und mehr.

Spring Cloud Gateway kann so konfiguriert werden, dass es HTTP-Anfragen basierend auf verschiedenen Kriterien, wie dem Pfad oder den Header-Informationen, an verschiedene interne Dienste weiterleitet. Zudem bietet das Gateway Filters, die verwendet werden können, um Anfragen und Antworten zu manipulieren. Dies unterstützt Dinge wie die Authentifizierung und Autorisierung, Request/Response-Transformation und das Hinzufügen von benutzerdefinierten HTTP-Headern.

Spring Cloud Gateway verwendet intern Spring WebFlux, das auf dem Project Reactor aufbaut, um reaktive und nicht-blockierende Verarbeitungsfunktionen bereitzustellen.

Fachbegriffe

- **Route**
  Der Grundbaustein des Gateways. Es wird durch eine ID, einen Ziel-URI, eine Sammlung von Predicates und eine Sammlung von Filtern definiert. Eine Route wird abgeglichen, wenn das Aggregatprädikat wahr ist.
- **Predicate**
  Dabei handelt es sich um ein Java 8-Funktionsprädikat, mit dem alles aus der HTTP-Anfrage abgeglichen werden kann, beispielsweise Header oder Parameter.
- **Filter**
  Dabei handelt es sich um Instanzen von Gateway-Filtern, die mit einer bestimmten Factory erstellt wurden. Mit Filtern können Anfragen und Antworten vor oder nach dem Senden nachgelagerter Anfragen geändert werden.

2 Beispielprojekt aufsetzen

2.1 Einleitung

Dependencies

- Eureka Client
- Spring Cloud Gateway / Webflux

Eureka Client

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  <version>4.0.2</version>
</dependency>
```

Spring Cloud Gateway

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>

<properties>
  <java.version>17</java.version>
  <spring-cloud.version>2022.0.4</spring-cloud.version>
</properties>
```

2.2 Gateway-Handler-Mapping

Die Gateway-Handler-Mapping Komponente ist für die Zuordnung eingehender Anforderungen zum entsprechenden Handler basierend auf den Routing-Regeln verantwortlich. In Spring Cloud Gateway kann der Handler ein Backend-Dienst, ein Filter oder ein Endpunkt sein.

Die Implementierung der Gateway-Handler-Mapping kann auch in der Application-Konfiguration (yml) erfolgen. Hier ist ein Beispiel für die Definition eines Handler-Mappings in der Application-Konfiguration:

application.properties bzw. application.yml

```
# Server config
server:
  port: 8080

# Eureka Client
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      default-zone: http://localhost:8761/eureka/

# Spring Cloud
spring:
  application:
    name: app-gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
          lower-case-service-id: true
      routes:
        - id: service-01
          uri: http://localhost:8081/api
          predicates:
            - Path=/service-01/**
```

Hier ist ein Beispiel für die Definition einer Handler-Zuordnung für einen Backend-Dienst:

```
@Bean
public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
    return builder.routes()
        .route("service-01", r -> r.path("/service-01/**")
            .uri("http://localhost:8081/api"))
        .build();
}
```

3 Aufgaben

3.1 Aufgabe 1

Implementieren Sie das Gateway gemäss der Anleitung im Kapitel 1.

Implementieren Sie folgende Routes:

```
routes:
  - id: service-01
    uri: http://localhost:8081/api
    predicates:
      - Path=/api/first/**
  - id: service-02
    uri: http://localhost:8082/api
    predicates:
      - Path=/api/second/**
```

3.2 Aufgabe 2

Starten Sie die folgenden Apps der Reihe nach:

- Eureka Server starten
- Eureka Server im Browser prüfen (http://localhost:8761)
- Service 01 starten
- Registrierung des Service 01 im Eureka Server im Browser prüfen
- Api des Service 01 prüfen (http://localhost:8081/api)
- App-Gateway starten
- Registrierung des App-Gateways im Eureka Server im Browser prüfen

Nun können Sie die externen Request prüfen, indem Sie direkt das Gateway angeben:

- http://localhost:8080/api/first/api
- http://localhost:8080/api/second/api