Spring cloud ribbon with eureka – Client side load balancer example

Dans cette partie Spring Cloud, apprenez à utiliser l' équilibrage de charge côté client à l'aide de Netflix Ribbon dans les projets Spring Boot/Cloud. Apprenez à créer des applications basées sur des microservices qui utilisent le ribbon comme équilibreur de charge côté client et eureka comme service de registre. Découvrez comment nous pouvons ajouter dynamiquement de nouvelles instances de microservices sous l'équilibreur de charge.

1. Équilibrage de charge côté serveur traditionnel

L'équilibrage de charge côté serveur est impliqué dans les applications monolithiques où nous avons un nombre limité d'instances d'application derrière l'équilibreur de charge. Nous déployons nos fichiers war/ear dans plusieurs instances de serveur qui sont essentiellement un pool de serveurs ayant la même application déployée et nous plaçons un équilibreur de charge devant celui-ci.

L'équilibreur de charge a une adresse IP et un DNS publics. Le client fait une demande en utilisant cette adresse IP/DNS publique. L'équilibreur de charge décide vers quel serveur d'applications interne la demande sera transmise. Il utilise principalement un algorithme de session circulaire ou persistante. Nous l'appelons l'équilibrage de charge côté serveur.

1.1. Problèmes dans l'architecture des microservices

La plupart du temps, l'équilibrage de charge côté serveur est un effort manuel et nous devons ajouter/supprimer manuellement des instances dans l'équilibreur de charge pour fonctionner. Donc, idéalement, nous perdons

l'évolutivité à la demande d'aujourd'hui pour découvrir et configurer automatiquement lorsque de nouvelles instances seront supprimées.

Un autre problème est d'avoir une politique de basculement pour fournir au client une expérience transparente. Enfin, nous avons besoin d'un serveur distinct pour héberger l'instance d'équilibrage de charge, ce qui a un impact sur les coûts et la maintenance.

2. Équilibrage de charge côté client

Pour surmonter les problèmes de l'équilibrage de charge traditionnel, l'équilibrage de charge côté client est entré en scène. Ils résident dans l'application en tant que composant intégré et sont regroupés avec l'application, nous n'avons donc pas à les déployer sur des serveurs séparés.

Voyons maintenant la situation dans son ensemble. Dans l'architecture de microservices, nous devrons développer de nombreux microservices et chaque microservice peut avoir plusieurs instances dans l'écosystème. Pour surmonter cette complexité, nous avons déjà une solution populaire pour utiliser le modèle de découverte de service . Dans les applications de démarrage de spring, nous avons quelques options dans l'espace de découverte de services telles que eureka, consoul, zookeeper, etc.

Désormais, si un microservice souhaite communiquer avec un autre microservice, il recherche généralement le registre de services à l'aide du client de découverte et le serveur Eureka renvoie toutes les instances de ce microservice cible au service appelant. Ensuite, il est de la responsabilité du service appelant de choisir à quelle instance envoyer la requête.

Ici, l'équilibrage de charge côté client entre en scène et gère automatiquement les complexités de cette situation et délègue à l'instance appropriée de manière équilibrée. Notez que nous pouvons spécifier l'algorithme d'équilibrage de charge à utiliser.

3. ribbon Netflix – Équilibreur de charge côté client

Le ribbon Netflix de la famille Spring Cloud fournit une telle installation pour configurer l'équilibrage de charge côté client avec le composant de registre de service. Spring boot a un très bon moyen de configurer l'équilibreur de charge côté client ribbon avec un minimum d'effort. Il offre les fonctionnalités suivantes

1. L'équilibrage de charge
2. Tolérance aux pannes
3. Prise en charge de plusieurs protocoles (HTTP, TCP, UDP) dans un modèle asynchrone et réactif
4. Mise en cache et traitement par lots

Pour obtenir des binaires de ribbon, accédez à maven central . Voici un exemple pour ajouter une dépendance dans Maven :

```
<dependency>
    <groupId>com.netflix.ribbon</groupId>
    <artifactId>ribbon</artifactId>
    <version>2.2.2</version>
</dependency>
```

4. Exemple de ribbon Netflix

4.1. Pile technologique

- Java, Eclipse, Maven comme environnement de développement
- Spring-boot et Cloud comme framework d'application
- Eureka en tant que serveur de registre de service
- ribbon en tant qu'équilibreur de charge côté client

Nous allons créer les composants suivants et voir comment l'ensemble de l'écosystème se coordonne dans un environnement distribué.

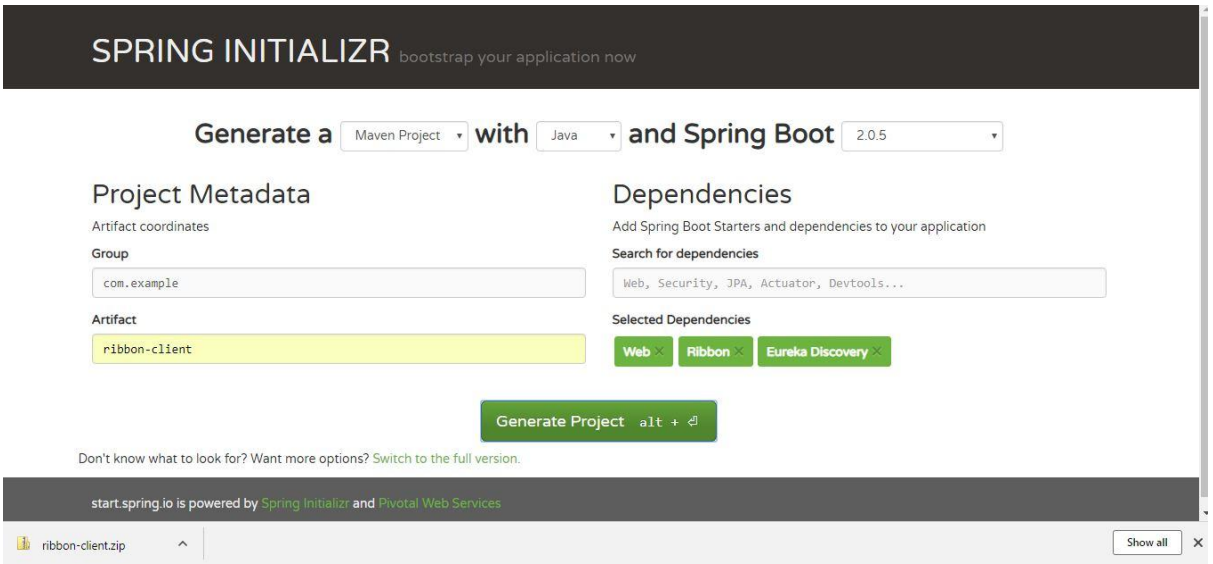
- Deux microservices utilisant Spring Boot. L'un doit en invoquer un autre selon les besoins de l'entreprise

- Serveur de registre de service Eureka
- ribbon dans le microservice appelant pour appeler l'autre service de manière équilibrée AVEC la découverte de service
- Appel du service de manière équilibrée en charge SANS découverte de service

4.2. Créer un microservice backend

Nous allons créer un microservice simple à l'aide de Spring Boot et exposer un point de terminaison REST simple. Créez un projet de démarrage de spring simple nommé `ribbon-server` avec spring-boot-web et la dépendance du client de découverte de service pour l'héberger sur un serveur Web et exposez un REST Controller à tester.

Pour ce faire, nous devons aller sur https://start.spring.io/ et donner les coordonnées maven et sélectionner les dépendances. Téléchargez le fichier zip contenant le projet squelette



Projet généré à partir de Spring Initializer

4.2.1. Créer un REST endpoint

Écrivez un contrôleur de REST et exposez un point de terminaison de REST comme ci-dessous.

4.2.2 Activer discovery client

Enregistrez ce service sur eureka pour ce faire, nous devons ajouter @EnableDiscoveryClient dans la classe d'application. Nous devons également ajouter les entrées ci-dessous dans le fichier de propriétés de l'application.

après le code java voilà la conf :

```
spring.application.name=server
server.port = 9090

eureka.client.serviceUrl.defaultZone=
http://${registry.host:localhost}:${registry.port:8761}/eureka/
eureka.client.healthcheck.enabled= true
eureka.instance.leaseRenewalIntervalInSeconds= 1
eureka.instance.leaseExpirationDurationInSeconds= 2
```

4.3. Serveur de registre du service Eureka

Créez le serveur de découverte de services. C'est aussi très facile. Nous avons juste besoin de créer un projet de démarrage de spring comme ci-dessus avec Eureka Server en tant que dépendance et de faire les configurations ci-dessous.

4.3.1. Configuration du serveur Eureka

Une fois que le service Spring Boot est prêt et importé dans Eclipse, ajoutez l' annotation @EnableEurekaServer dans la classe d'application Spring Boot et ajoutez également la configuration ci-dessous dans le fichier de propriétés de l'application.

après le code java voilà la conf :

```
spring.application.name= ${springboot.app.name:eureka-serviceregistry}
server.port = ${server-port:8761}
eureka.instance.hostname= ${springboot.app.name:eureka-serviceregistry}
eureka.client.registerWithEureka= false
eureka.client.fetchRegistry= false
eureka.client.serviceUrl.defaultZone:
http://${registry.host:localhost}:${server.port}/eureka/
```

4.4 Créer un autre microservice

Suivez la section précédente pour créer un autre service nommé `ribbon-client` avec une dépendance supplémentaire `spring-cloud-starter-netflix-ribbon`.

4.4.1. Configuration du ribbon

Dans la classe d'application, ajoutez deux annotations @RibbonClient et @EnableDiscoveryClient pour activer le ribbon et le client Eureka pour le registre de service.

Dans le `application.properties`, nous devons effectuer les configurations ci-dessous. Ici server.ribbon.listOfServers est désactivé, nous pouvons l'activer pour ajouter manuellement un serveur à cet équilibreur de charge. Nous allons vérifier cela dans la section des tests. D'autres propriétés sont explicites.

```
spring.application.name=client
server.port=8888

eureka.client.serviceUrl.defaultZone=
http://${registry.host:localhost}:${registry.port:8761}/eureka/
eureka.client.healthcheck.enabled= true
eureka.instance.leaseRenewalIntervalInSeconds= 1
eureka.instance.leaseExpirationDurationInSeconds= 2

server.ribbon.eureka.enabled=true
```

```
#server.ribbon.listOfServers=localhost:9090,localhost:9091,localhost:909
2
server.ribbon.ServerListRefreshInterval=1000
#logging.level.root=TRACE
```

Nous devons maintenant créer une classe de configuration supplémentaire pour le ribbon pour mentionner l' algorithme d'équilibrage de charge et le contrôle de santé . Nous allons maintenant utiliser les valeurs par défaut fournies par Ribbon, mais dans cette classe, nous pouvons très bien les remplacer et ajouter la nôtre logique personnalisée.

5. Testez l'application

5.1. Composants de démarrage

Exécutez la commande build use finale `mvn clean install` et vérifiez si la génération est réussie. S'il y a une erreur, vous devez la corriger pour continuer. Une fois que nous aurons réussi à construire tous les projets Maven, nous commencerons les services un par un.

Eureka d'abord, puis le micro service back-end et enfin le micro service frontend.

Pour démarrer chaque microservice, nous utiliserons `'java -jar -Dserver.port=XXXX target/YYYYY.jar'` command.

5.2. Déployer plusieurs instances de microservice backend

Pour ce faire, nous devons utiliser un port différent pour cela, pour démarrer le service dans un port spécifique, nous devons passer le port de cette manière.

`java -jar -Dserver.port=XXXX target/YYYYY.jar`. Nous allons créer 3 instances de ce service dans les ports 9090, 9091 et 9092.

5.3. Vérifier le serveur eureka

Allez maintenant `http://localhost:8761/` dans le navigateur et vérifiez que le serveur eureka est en cours d'exécution avec tous les microservices enregistrés avec le nombre d'instances souhaité.

5.4. Vérifiez si l'équilibrage de charge côté client fonctionne

Dans le microservice frontal, nous appelons le microservice backend à l'aide de RESTTemplate . Le modèle de REST est activé en tant qu'équilibreur de charge côté client à l'aide de l' annotation @LoadBalanced .

Accédez maintenant au navigateur et ouvrez le point de terminaison de REST du microservice client `http://localhost:8888/client/frontend` et voyez que la réponse provient de l'une des instances backend.

Pour comprendre, ce serveur principal renvoie son port en cours d'exécution et nous l'affichons également dans la réponse du microservice client. Essayez d'actualiser cette URL plusieurs fois et notez que le port du serveur principal ne cesse de changer, ce qui signifie que l'équilibrage de charge côté client fonctionne. Essayez maintenant d'ajouter plus d'instances de serveur principal et vérifiez qu'elles sont également enregistrées sur le serveur eureka et éventuellement prises en compte dans le ribbon, car une fois qu'elles seront enregistrées dans eureka et le ribbon, le ribbon enverra également une demande aux nouvelles instances.

5.5. Test avec des backends de code dur sans découverte de service

Accédez au `application.properties` fichier de microservice frontal et activez-le.

```
server.ribbon.listOfServers=localhost:9090,localhost:9091,localhost:9092
server.ribbon.eureka.enabled=false
```

Testez maintenant l'URL du client. Vous obtiendrez une réponse des instances enregistrées uniquement. Désormais, si vous démarrez une nouvelle instance de microservice backend dans un port différent, le ribbon n'enverra pas de

demande à la nouvelle instance tant que nous ne l'aurons pas enregistré manuellement dans le ribbon.

Si vous avez des difficultés à tester cela, je suggérerai également de supprimer toutes les configurations liées à eureka de toutes les applications et d'arrêter également le serveur eureka. J'espère que vous ne rencontrerez aucune difficulté à tester cela également.

6. Résumé

Nous avons donc vu avec quelle facilité nous pouvons utiliser Ribbon avec Eureka dans le développement de microservices de démarrage de spring. Donc, la prochaine fois, si vous avez besoin de ce type d'exigences, vous pouvez utiliser cette approche.