

Микросервисы: Eureka и client-side Load Balancing

В этой статье рассмотрим пример с двумя микросервисами. Обнаруживать друг друга они будут с помощью Eureka. Кроме того, рассмотрим, как запускать микросервисы в нескольких экземплярах и балансировать нагрузку на микросервис (со стороны клиента).

1 Пример с микросервисами

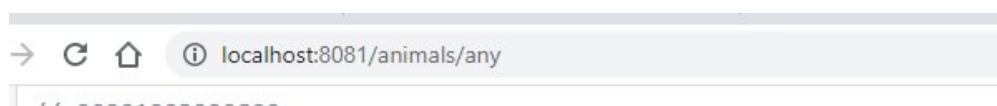
Итак, пусть у нас имеется два микросервиса (два Spring Boot приложения, предоставляющих REST API). Один — для внешнего пользователя, второй — для того, чтобы к нему обращался первый.

Микросервис Zoo

Сюда внешний пользователь приходит, чтобы посмотреть животных. Он заходит на адрес:

```
1. localhost:8081/animals/any
```

и видит там одно случайно выбранное животное. Например, *dog*:



```
// 20201223220809
// http://localhost:8081/animals/any

{
  "name": "dog"
}
```

Вот контроллер для отображения:

```
1.  @RestController
2.  public class ZooController {
3.      @Autowired
4.      private RandomAnimalClient randomAnimalClient;
5.
6.      @GetMapping("/animals/any")
7.      ResponseEntity<Animal> seeAnyAnimal() {
8.          return randomAnimalClient.random();
9.      }
10. }
```

RandomAnimalClient внедрен потому, что животные на самом деле берутся из второго микросервиса Random Animal.

RandomAnimalClient как раз и делает запрос к нему.

Микросервис Random Animal — выдает случайное животное

Второй микросервис — тоже отдельное Spring Boot приложение.

Вот его контроллер:

```
1.  @RestController
2.  public class RandomAnimalController {
3.
4.      private final AnimalDao animalDao;
5.
6.      public NameController(AnimalDao animalDao) {
7.          this.animalDao=animalDao;
8.      }
9.
10.     @GetMapping("/random")
```

```

11.     public Animal randomAnimal() {
12.         return animalDao.random();
13.     }
14. }

```

Этот микросервис запущен в двух экземплярах на портах 8082 и 8083. То есть случайное животное можно получить по любому из адресов:

```

1.  localhost:8082/random
2.  localhost:8083/random

```

AnimalDao, внедренный в *RandomAnimalController*, берет случайное животное из списка:

```

1.  @Component
2.  public class AnimalDao {
3.      private List<Animal> list = Arrays.asList(new
Animal("cat"), new Animal("dog"), new
Animal("fox"));
4.
5.
6.      public Animal random() {
7.          Random rand = new Random();
8.          return
list.get(rand.nextInt(list.size()));
9.      }
10. }

```

② Решение без Spring Cloud и возникающие проблемы

Если решать задачу без использования замечательных возможностей Spring Cloud, то со временем возникают проблемы:

1. Придется вести учет *url* и портов. У нас задача простая — два микросервиса на трех портах, и это не сложно. Но если микросервисов много, и адреса динамически меняются (запускаются новые экземпляры микросервисов на новых портах, какие-то экземпляры падают)? Хотелось бы, чтобы при запуске и отключении очередного экземпляра микросервиса другие

микросервисы были автоматически информированы о появлении и пропаже, и всё продолжало бы работать без исправления кода.

2. Надо как-то выбирать, на какой из запущенных экземпляров микросервиса обратиться. Мы запускаем два экземпляра `Random Animal`. И первый микросервис должен долбить не один и тот же экземпляр, а выбирать их (примерно) по очереди.

Есть еще проблемы, но о них в следующих статьях, а пока про первые две.

1. Первая проблема решается с помощью сервера Eureka. Мы запускаем отдельное приложение Eureka, которое ведет учет микросервисов и их адресов. Eureka по умолчанию запускается на порту 8761 — клиенты-микросервисы знают номер и уведомляют о себе при старте. Также сама Eureka периодически проверяет, жив ли клиент-микросервис. Чтобы сделать приложение клиентом сервера Eureka, мы добавляем в него Maven-зависимость, и всё. После этого он регистрируется в Eureka при запуске автоматически.
2. Нагрузка балансируется автоматически, если для обращения к экземплярам использовать не просто *RestTemplate*, а *@LoadBalanced RestTemplate*. (Можно еще использовать *WebClient* — он поддерживает реактивность — но о нем не в этой статье).

3 Решение со Spring Cloud

Итак, сначала о сервере Eureka, которая ведет реестр микросервисов.

Обнаружение сервисов с помощью Eureka Server

Чтобы создать приложение Eureka Server, в POM-файл нужно

добавить зависимость:

```
1. <dependency>
2.   <groupId>org.springframework.cloud</groupId>
3.   <artifactId>spring-cloud-starter-netflix-
   eureka-server</artifactId>
4. </dependency>
```

А главный класс нужно аннотировать `@EnableEurekaServer`:

```
1. @SpringBootApplication
2. @EnableEurekaServer
3. public class EurekaApplication {
4.
5.     public static void main(String[] args) {
6.
7.         SpringApplication.run(EurekaApplication.class,
8.                               args);
9.     }
```

Eureka Server готов. Осталось добавить настройки.

В `application.properties` пропишем:

```
1. eureka.client.registerWithEureka=false
2. eureka.client.fetchRegistry=false
3. spring.application.name=eureka-server
4. spring.cloud.loadbalancer.ribbon.enabled=false
```

Теоретически сервер может выступать и клиентом. Первые две настройки отменяют эту возможность: не дают серверу регистрировать самого себя в качестве клиента.

Третья настройка задает имя микросервиса, а четвертая отменяет использование Ribbon в качестве балансировщика нагрузки по умолчанию. Эти настройки будут в всех микросервисах. Но о балансировке ниже.

Пока займемся регистрацией микросервисов в качестве клиентов Eureka.

Сами микросервисы — Eureka Clients

Чтобы сделать микросервисы Zoo и Random Animal клиентами Эврики, в них необходимо добавить зависимости:

```
1. <dependency>
2.     <groupId>org.springframework.cloud</groupId>
3.     <artifactId>spring-cloud-starter-netflix-
   eureka-client</artifactId>
4. </dependency>
```

Для каждого микросервиса необходимо задать имя, порт и отключить балансировщик нагрузки Ribbon (потому что будем использовать другой):

```
1. spring.application.name=zoo
2. spring.cloud.loadbalancer.ribbon.enabled=false
3. server.port=8081
```

i

По заданному имени микросервиса они смогут обращаться друг к другу

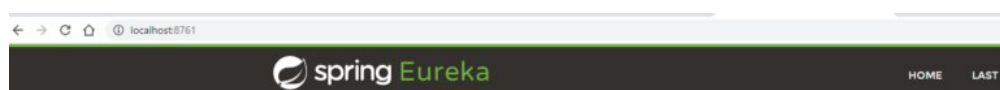
Все вместе — запуск сервера Eureka и его клиентов

Далее нужно запустить сначала сервер Eureka, а затем клиенты-микросервисы.

Eureka по умолчанию запускается на порту 8761, и если открыть

```
1. http://localhost:8761/
```

то увидим список запущенных микросервисов с их именами:



System Status			
Environment	N/A	Current time	
Data center	N/A	Uptime	
		Lease expiration enabled	
		Renews threshold	
		Renews (last min)	

DS Replicas	
localhost	

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
RANDOM-ANIMAL	n/a (2)	(2)	UP (2) - DESKTOP-UHTR2G1.random-animal.8082, DESKTOP-UHTR2G1.random-animal.8083
ZOO	n/a (1)	(1)	UP (1) - DESKTOP-UHTR2G1.zoo.8081

General Info	
Name	Value

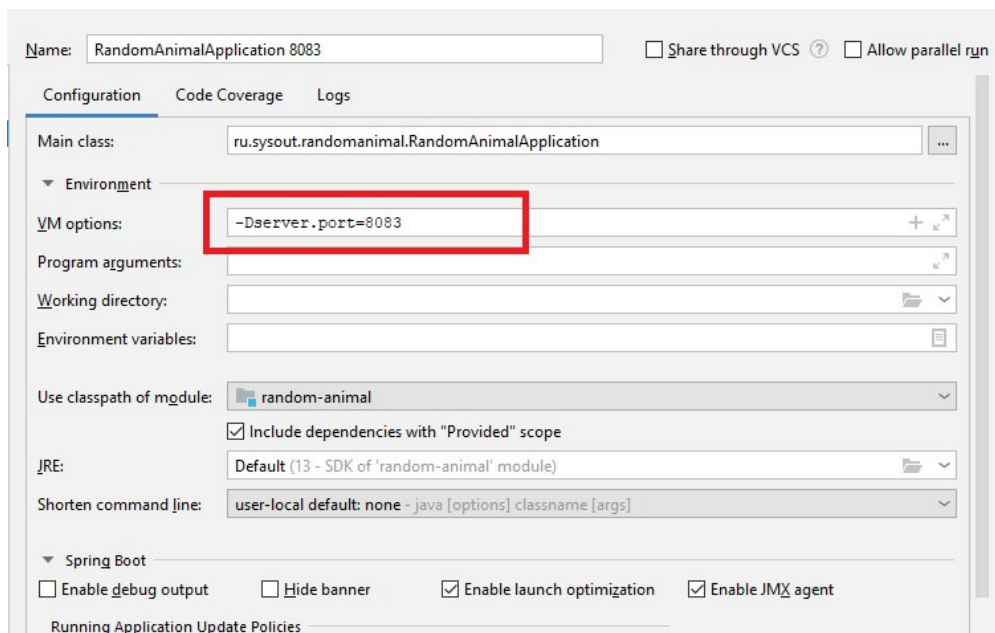
Eureka показывает своих клиентов zoo и random-animal (2 экземпляра)

Random-animal запущен дважды — на портах 8082 и 8083.

Как запустить экземпляр на другом порту в IDEA

Чтобы указать порт для второго экземпляра *Random-animal*, в IntelliJ IDEA в Run->Edit Configurations в поле VM Options прописываем:

```
1. -Dserver.port=8083
```



Запуск на конкретном порту

Эта настройка перезаписывает порт, заданный в *application.properties*.

Балансировка нагрузки с помощью `@Loadbalanced RestTemplate`

Для обращения одного микросервиса к другому нам понадобятся имена, заданные в настройках (*spring.application.name*).

Итак, из Zoo к Random Animal мы обращались с помощью *RandomAnimalClient*, вот его код:

```
1.  @Component
2.  public class RandomAnimalClient {
3.      private static final Logger LOGGER =
4.          LoggerFactory
5.              .getLogger(RandomAnimalClient.class);
6.
7.      private final RestTemplate restTemplate;
8.
9.      RandomAnimalClient(RestTemplate restTemplate) {
10.         this.restTemplate = restTemplate;
11.     }
12.
13.     //spring.application.name=random-animal есть в
14.     //настройках Random Animal
15.     public ResponseEntity<Animal> random() {
16.         LOGGER.debug("Sending request for animal
17.             {}");
18.         return
19.             restTemplate.getForEntity("http://random-
20.                 animal/random",
21.                 Animal.class);
22.     }
23. }
```

Но *RestTemplate* тут не простой, а сбалансированный. Именно поэтому обращение

```
1.  http://random-animal/random
```

идет на соседний микросервис с прописанным в настройках именем *random-animal*, а не в интернет.

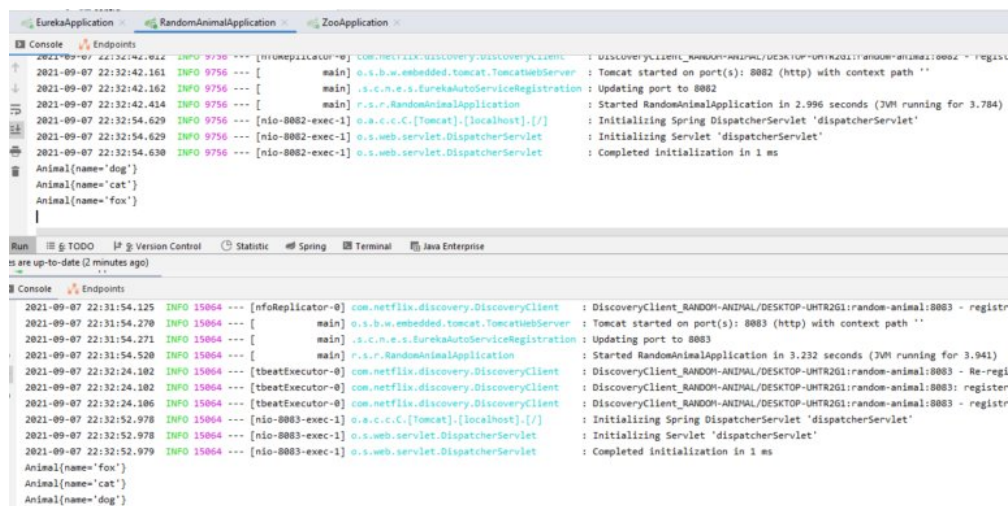
Чтобы сделать *RestTemplate* сбалансированным, просто

аннотируем его `@LoadBalanced`:

```
1. @Configuration
2. public class RestTemplateConfig {
3.
4.     @Bean
5.     @LoadBalanced
6.     RestTemplate restTemplate() {
7.         return new RestTemplate();
8.     }
9. }
```

Для использования `@LoadBalanced RestTemplate` в POM никакой зависимости добавлять не нужно.

Давайте добавим в *RandomAnimalController* вывод животного в консоль. Снова всё запустим и будем обновлять главную страницу. В консолях двух запущенных экземпляров *Random Animal* будет по очереди выводиться животное — то в одной консоли, то в другой. Видно, что балансировка нагрузки происходит:



```
EurekaApplication RandomAnimalApplication ZooApplication
Console Endpoints
2021-09-07 22:32:42.162 INFO 9756 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8082 (http) with context path ''
2021-09-07 22:32:42.162 INFO 9756 --- [main] o.s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8082
2021-09-07 22:32:42.414 INFO 9756 --- [main] r.s.r.RandomAnimalApplication : Started RandomAnimalApplication in 2.996 seconds (JVM running for 3.784)
2021-09-07 22:32:54.629 INFO 9756 --- [nio-8082-exec-1] o.s.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-09-07 22:32:54.629 INFO 9756 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-09-07 22:32:54.630 INFO 9756 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
Animal(name='dog')
Animal(name='cat')
Animal(name='fox')
|

Run TODO Version Control Statistic Spring Terminal Java Enterprise
is are up-to-date (2 minutes ago)
Console Endpoints
2021-09-07 22:31:54.125 INFO 15064 --- [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_RANDOM-ANIMAL/DESKTOP-UHTR261:random-animal:8083 - registr
2021-09-07 22:31:54.270 INFO 15064 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8083 (http) with context path ''
2021-09-07 22:31:54.271 INFO 15064 --- [main] o.s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8083
2021-09-07 22:31:54.520 INFO 15064 --- [main] r.s.r.RandomAnimalApplication : Started RandomAnimalApplication in 3.232 seconds (JVM running for 3.941)
2021-09-07 22:32:24.182 INFO 15064 --- [tbeatExecutor-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_RANDOM-ANIMAL/DESKTOP-UHTR261:random-animal:8083 - Re-regi
2021-09-07 22:32:24.182 INFO 15064 --- [tbeatExecutor-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_RANDOM-ANIMAL/DESKTOP-UHTR261:random-animal:8083 - registr
2021-09-07 22:32:24.186 INFO 15064 --- [tbeatExecutor-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_RANDOM-ANIMAL/DESKTOP-UHTR261:random-animal:8083 - Re-regi
2021-09-07 22:32:52.978 INFO 15064 --- [nio-8083-exec-1] o.s.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-09-07 22:32:52.978 INFO 15064 --- [nio-8083-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-09-07 22:32:52.979 INFO 15064 --- [nio-8083-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
Animal(name='fox')
Animal(name='cat')
Animal(name='dog')
```

Консоли экземпляров *Random Animal*

Рассмотренная балансировка называется *client-side load balancing*, потому что именно клиент (тот, кто обращается к REST API) решает, к какому именно экземпляру сервиса обратиться. И делает он это с помощью `@Loadbalanced RestTemplate`.

DiscoveryClient — альтернатива @Loadbalanced RestTemplate

Кстати, код с *@Loadbalanced RestTemplate* можно переписать на более понятный. Можно взять обычный *RestTemplate*, внедрить *DiscoveryClient* в *RandomAnimalClient* и переписать метод *random()* так:

```
1.  @Component
2.  public class RandomAnimalClient {
3.      private static final Logger LOGGER = LoggerFactory
4.          .getLogger(RandomAnimalClient.class);
5.
6.      private final RestTemplate restTemplate;
7.      private final DiscoveryClient discoveryClient;
8.
9.      RandomAnimalClient(RestTemplate restTemplate,
10.         DiscoveryClient discoveryClient) {
11.          this.restTemplate = restTemplate;
12.          this.discoveryClient = discoveryClient;
13.      }
14.
15.      public ResponseEntity<Animal> random() {
16.
17.          ServiceInstance instance =
18.              discoveryClient.getInstances("random-animal")
19.                  .stream().findAny()
20.                  .orElseThrow(() -> new
21.                      IllegalStateException("Random-animal service unavailable"));
22.
23.          UriComponentsBuilder uriComponentsBuilder =
24.              UriComponentsBuilder
25.                  .fromHttpUrl(instance.getUri().toString()
26.                      + "/random");
27.
28.          return
29.              restTemplate.getForEntity(uriComponentsBuilder.toUriString(),
30.                  Animal.class);
31.      }
32.  }
```

DiscoveryClient получает список экземпляров микросервиса *random-animal* по его имени. Мы выбираем случайный экземпляр и обращаемся к нему.

@Loadbalanced RestTemplate делает примерно то же самое, но алгоритм выбора экземпляра лучше, так что рекомендуется использовать его.

С 2015 по умолчанию в Spring Cloud был включен балансировщик Ribbon от Netflix. Но сейчас есть новый Spring Cloud Load balancer, именно поэтому в настройках мы отключаем старый, как уже было показано выше:

```
1. spring.cloud.loadbalancer.ribbon.enabled=false
```

4 Итоги

Исходный код доступен на [\[ссылка\]](#). В [\[ссылка\]](#) рассмотрим API Gateway — балансировку на стороне сервера.

Есть также о [\[ссылка\]](#) и [\[ссылка\]](#).

sysout / 23.12.2020 / Spring, Spring Cloud

Микросервисы: Eureka и client-side Load Balancing: 8 комментариев

Сергей

20.06.2021 в 13:04

Самый адекватный ресурс по Spring Framework «по-русски»!
Спасибо большое автору за проделанную работу

Ответить

Д

06.09.2021 в 05:48

Здравствуйте.

«В консолях двух запущенных экземпляров Random Animal будет по очереди выводиться животное – то в одной консоли, то в другой. Видно, что балансировка нагрузки происходит:», а ниже 2 одинаковых скриншота консоли. Видно, что 8082 порт двух на скриншотах. Это скорее всего не корректно. Меня просто сильно озадачил одинаковый вывод, даже если смотреть по времени.

Ответить

sysout 👤

06.09.2021 в 09:51

Да, конечно, это ошибка. Там должны быть разные консоли на 8081 и 8082, спасибо.

Ответить

sysout 👤

07.09.2021 в 20:40

точнее, на портах 8082 и 8083. Скриншот обновлен.

Ответить

Д

09.09.2021 в 03:59

Спасибо большое за статьи и за то что исправляете ошибки. Поправьте и тут тогда пж.

Этот микросервис запущен в двух экземплярах на портах 8082 и 8083. То есть случайное животное можно получить по любому из адресов:

localhost:8081/random

localhost:8082/random

Ответить

sysout 🧑

09.09.2021 в 07:01

ок, исправлено.

Ответить

М

06.01.2022 в 05:15

Спасибо за проделанную работу!

Ответить

Сергей

30.03.2022 в 21:08

Большое спасибо за ресурс! Не останавливайтесь! 😊

Ответить

Добавить комментарий

Ваш адрес email не будет опубликован. Обязательные поля помечены *

КОММЕНТАРИЙ *

ИМЯ *

EMAIL *

САЙТ

ОТПРАВИТЬ КОММЕНТАРИЙ

НАЗАД

Составной ключ с @ClassId

ДАЛЕЕ

Spring Cloud API Gateway

Прошу прощения: на комментарии временно не отвечаю.

СВЕЖИЕ КОММЕНТАРИИ

- sysout к записи [Отношение OneToMany в Hibernate и Spring](#)
- Сергей Акопов к записи [Отношение OneToMany в Hibernate и Spring](#)
- Дмитрий к записи [N+1 проблема в Hibernate](#)
- Дмитрий к записи [Spring Custom Login Form](#)
- Руслан к записи [Отказоустойчивость микросервисов: шаблон Circuit Breaker](#)
- Николай к записи [Spring Custom Login Form](#)
- Денис к записи [@Primary, @Qualifier и внедрение списка](#)
- Михаил к записи [Работа с IoC-контейнером в Spring](#)
- Арсен к записи [Как работает Flush в Hibernate](#)
- Сергей к записи [Spring Data JDBC: CrudRepository и Query methods](#)
- Yustas к записи [Пример приложения с JWT-токеном](#)
- sysout к записи [Пример приложения с JWT-токеном](#)
- Михаил к записи [Пример приложения с JWT-токеном](#)
- Андрей к записи [Преобразование Entity в DTO с помощью ModelMapper](#)

[Core Java](#)

[Spring](#)



