通过之前的课程，我们已经能够通过使用它们搭建起一个基础的微服务架构系统来实现我们的业务需求了。但是，随着业务的发展，我们的系统规模也会变得越来越大，各微服务间的调用关系也变得越来越错综复杂。通常一个由客户端发起的请求在后端系统中会经过多个不同的微服务调用来协同产生最后的请求结果，在复杂的微服务架构系统中，几乎每一个前端请求都会形成一条复杂的分布式服务调用链路，在每条链路中任何一个依赖服务出现延迟过高或错误的时候都有可能引起请求最后的失败。这时候对于每个请求全链路调用的跟踪就变得越来越重要，通过实现对请求调用的跟踪可以帮助我们快速的发现错误根源以及监控分析每条请求链路上的性能瓶颈等好处。

针对上面所述的分布式服务跟踪问题，Spring Cloud Sleuth提供了一套完整的解决方案。在本章中，我们将详细介绍如何使用Spring Cloud Sleuth来为我们的微服务架构增加分布式服务跟踪的能力。

实战练习

在介绍各种概念与原理之前，我们先通过实现一个简单的示例，对存在服务调用的应用增加一些sleuth的配置实现基本的服务跟踪功能，以此来对Spring Cloud Sleuth有一个初步的了解，随后再逐步展开介绍实现过程中的各个细节部分。

准备工作

在引入Sleuth之前，我们先按照之前章节学习的内容来做一些准备工作，构建一些基础的设施和应用：

- 服务注册中心：eureka-server，这里不做赘述，直接使用之前构建的工程。
- 微服务应用：trace-a，实现一个REST接口/trace-a，调用该接口后将触发对trace-b应用的调用。具体实现如下：
    - 创建一个基础的Spring Boot应用，在pom.xml中增加下面依赖：

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-ribbon</artifactId>
    </dependency>
</dependencies>
```

- 创建应用主类，并实现/trace-a接口，并使用RestTemplate调用trace-b应用的接口。具体如下：

```
@RestController
@EnableDiscoveryClient
@SpringBootApplication
public class TraceaApplication {
    private final Logger logger = Logger.getLogger(getClass());

    @Bean
    @LoadBalanced
    RestTemplate restTemplate() {
        return new RestTemplate();
    }

    @RequestMapping(value = "/trace-a", method = RequestMethod.GET)
    public String trace() {
        logger.info("===call trace-a===");
        return restTemplate().getForEntity("http://trace-b/trace-b",
String.class).getBody();
    }

    public static void main(String[] args) {
        SpringApplication.run(TraceaApplication.class, args);
    }
}
```

- application.properties中将eureka.client.serviceUrl.defaultZone参数指向eureka-server的地址，具体如下：

```
spring.application.name=trace-a
server.port=2411
eureka.client.serviceUrl.defaultZone=http://localhost:1001/eureka/
```

- 微服务应用：trace-b，实现一个REST接口/trace-b，供trace-a调用。具体实现如下：
  - 创建一个基础的Spring Boot应用，pom.xml中的依赖与trace-a相同
  - 创建应用主类，并实现/trace-b接口，具体实现如下：

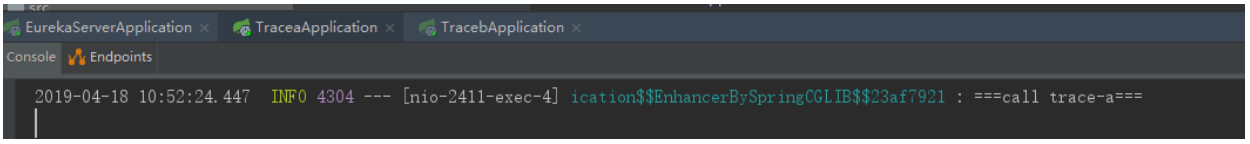
```
@RestController
@EnableDiscoveryClient
@SpringBootApplication
public class TracebApplication {
    private final Logger logger = LoggerFactory.getLogger(getClass());

    @RequestMapping(value = "/trace-b", method = RequestMethod.GET)
```

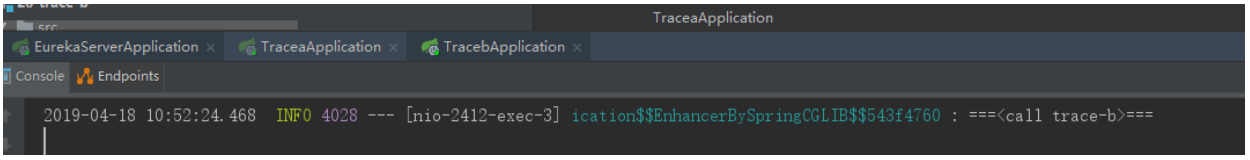
```
    public String trace() {
        logger.info("===<call trace-b>===");
        return "Trace";
    }

    public static void main(String[] args) {
        SpringApplication.run(TracebApplication.class, args);
    }

}
```

- application.properties中将eureka.client.serviceUrl.defaultZone参数指向eureka-server的地址，另外还需要设置不同的应用名和端口号，具体如下：

```
spring.application.name=trace-b
server.port=2412
eureka.client.serviceUrl.defaultZone=http://localhost:1001/eureka/
```

在实现了上面内容之后，我们可以将eureka-server、trace-a、trace-b三个应用都启动起来，并通过postman或curl等工具来对trace-a的接口发送请求http://localhost:2411/trace-a，我们可以得到返回值Trace，同时还能在它们的控制台中分别获得下面的输出：



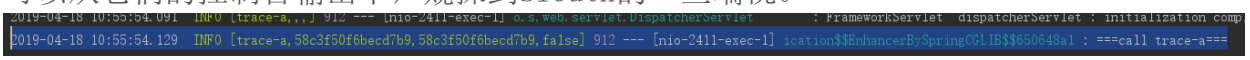


实现跟踪

在完成了准备工作之后，接下来我们开始进行本章的主题内容，为上面的trace-a和trace-b来添加服务跟踪功能。通过Spring Cloud Sleuth的封装，我们为应用增加服务跟踪能力的操作非常简单，只需要在trace-a和trace-b的pom.xml依赖管理中增加spring-cloud-starter-sleuth依赖即可，具体如下：

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

到这里，实际上我们已经为trace-a和trace-b实现服务跟踪做好了基础的准备，重启trace-a和trace-b后，再对trace-a的接口发送请求http://localhost:2411/trace-a。此时，我们可以从它们的控制台输出中，窥探到sleuth的一些端倪。



2019-04-18 10:55:54.129 INFO [trace-a,58c3f50f6becd7b9,58c3f50f6becd7b9,false] 912 --- [nio-2411-exec-1] ication$$EnhancerBySpringCGLIB$$650648a1 : ===call

trace-a===

2019-04-18 10:55:54.958 INFO [trace-b,58c3f50f6becd7b9,554e00fb95fb314b,false] 14464 --- [nio-2412-exec-1] ication$$EnhancerBySpringCGLIB$$959616e0 : ===<call trace-b>===

从上面的控制台输出内容中，我们可以看到多了一些形如[trace-1,58c3f50f6becd7b9,58c3f50f6becd7b9,false]的日志信息，而这些元素正是实现分布式服务跟踪的重要组成部分，它们每个值的含义如下：

- 第一个值：trace-a，它记录了应用的名称，也就是application.properties中spring.application.name参数配置的属性。
- 第二个值：58c3f50f6becd7b9，Spring Cloud Sleuth生成的一个ID，称为Trace ID，它用来标识一条请求链路。一条请求链路中包含一个Trace ID，多个Span ID。
- 第三个值：58c3f50f6becd7b9，Spring Cloud Sleuth生成的另外一个ID，称为Span ID，它表示一个基本的工作单元，比如：发送一个HTTP请求。s
- 第四个值：false，表示是否要将该信息输出到Zipkin等服务中来收集和展示。

上面四个值中的Trace ID和Span ID是Spring Cloud Sleuth实现分布式服务跟踪的核心。在一次服务请求链路的调用过程中，会保持并传递同一个Trace ID，从而将整个分布于不同微服务进程中的请求跟踪信息串联起来，以上面输出内容为例，trace-a和trace-b同属于一个前端服务请求来源，所以他们的Trace ID是相同的，处于同一条请求链路中。