16.Spring Cloud Sleuth

16.1 简介

在这种大规模的分布式系统中，一个完整的系统是由很多种不同的服务来共同支撑的。不同的系统可能分布在上千台服务器上，横跨多个数据中心。一旦系统出问题，此时问题的定位就比较麻烦。

分布式链路追踪：

在微服务环境下，一次客户端请求，可能会引起数十次、上百次服务端服务之间的调用。一旦请求出问题了，我们需要考虑很多东西：

- 如何快速定位问题？
- 如果快速确定此次客户端调用，都涉及到哪些服务？
- 到底是哪一个服务出问题了？

要解决这些问题，就涉及到分布式链路追踪。

分布式链路追踪系统主要用来跟踪服务调用记录的，一般来说，一个分布式链路追踪系统，有三个部分：

- 数据收集
- 数据存储
- 数据展示

Spring Cloud Sleuth 是 Spring Cloud 提供的一套分布式链路追踪系统。

trace：从请求到达系统开始，到给请求做出响应，这样一个过程成为 trace
span：每次调用服务时，埋入的一个调用记录，成为 span
annotation：相当于 span 的语法，描述 span 所处的状态。

16.2 简单应用

首先创建一个项目，引入 Spring Cloud Sleuth

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

接下来创建一个 HelloController ，打印日志测试：

```
@RestController
public class HelloController {
    private static final Log log = LogFactory.getLog(HelloController.class);
    @GetMapping("/hello")
    public String hello() {
        log.info("hello spring cloud sleuth");
        return "hello spring cloud sleuth";
    }
}
```

可以给当前服务配置一个名字，这个名字在输出的日志中会体现出来：

```
spring.application.name=javaboy-sleuth
```

启动应用，请求 /hello 接口，结果如下：

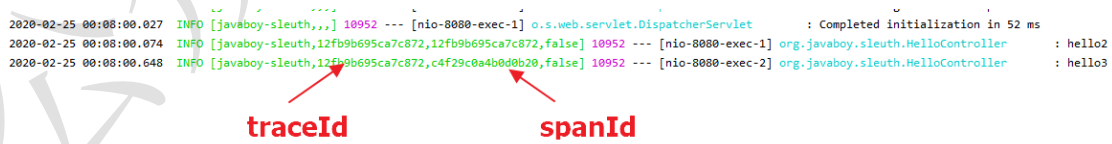
```
2020-02-25 00:00:54.049  INFO [javaboy-
sleuth,ee118b5d61416401,ee118b5d61416401,false] 2632 --- [nio-8080-exec-1]
org.javaboy.sleuth.HelloController       : hello spring cloud sleuth
```

这个就是 Spring Cloud Sleuth 的输出。

再定义两个接口，在 hello2 中调用 hello3，形成调用链：

```
@GetMapping("/hello2")
public String hello2() throws InterruptedException {
    log.info("hello2");
    Thread.sleep(500);
    return restTemplate.getForObject("http://localhost:8080/hello3",
String.class);
}
@GetMapping("/hello3")
public String hello3() throws InterruptedException {
    log.info("hello3");
    Thread.sleep(500);
    return "hello 3";
}
```

此时，访问 hello2，会先调用 hello3，拿到返回结果，会给 hello2。



一个 trace 由多个 span 组成，一个trace 相当于就是一个调用链，而一个 span 则是这个链中的每一次调用过程。

Spring Cloud Sleuth 中也可以收集到异步任务中的信息。

开启异步任务：

```
@SpringBootApplication
@EnableAsync
public class SleuthApplication {
```

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

    @Bean
    RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
...
```

创建一个 HelloService ，提供一个异步任务方法：

```java
@Service
public class HelloService {
    private static final Log log = LogFactory.getLog(HelloController.class);
    @Async
    public String backgroundFun() {
        log.info("backgroundFun");
        return "backgroundFun";
    }
}
```

再在 HelloController 中调用该异步方法：

```
@GetMapping("/hello4")
public String hello4() {
    log.info("hello4");
    return helloService.backgroundFun();
}
```

启动项目进行测试，发现 Sleuth 也打印出日志了，在异步任务中，异步任务是单独的 spanid。

```
2020-02-25 00:32:25.390  INFO [javaboy-sleuth,7a7726b908f60b50,7a7726b908f60b50,false] 12292 --- [nio-8080-exec-1] org.javaboy.sleuth.HelloController       : hello4
2020-02-25 00:32:25.437  INFO [javaboy-sleuth,7a7726b908f60b50,f84d64e3eaf17fd6,false] 12292 --- [        task-1] org.javaboy.sleuth.HelloController       : backgroundFun
```

Spring Cloud Sleuth 也可以手机定时任务的信息。

首先开启定时任务支持：

```
@SpringBootApplication
@EnableAsync
@EnableScheduling
public class SleuthApplication {

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

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

然后在 HelloSerivce 中，添加定时任务，去调用 background 方法。

```
@Scheduled(cron = "0/10 * * * * ?")
public void sche1() {
    log.info("start:");
    backgroundFun();
    log.info("end:");
}
```

然后访问 hello4 接口进行测试。

在定时任务中，每一次定时任务都会产生一个新的 Trace，并且在调用过程中，SpanId 都是一致的，这个和普通的调用不一样。

17.Zipkin

Zipkin 本身是一个由 Twitter 公司开源的分布式追踪系统。

Zipkin 分为 server 端和 client 端，server 用来展示数据，client 用来收集+上报数据。

17.1 准备工作

Zipkin 要先把数据存储起来，这里我们使用 Elasticsearch 来存储，所以，首先安装 es 和 es-head。

es 安装命令：

```
docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node" elasticsearch:7.1.0
```

可视化工具有三种安装方式：

1. 直接下载软件安装
2. 通过 Docker 安装
3. 安装 Chrome/Firefox 插件（公众号后后台回复 es-head ，获取 Chrome 插件的离线包）

这里采用第 3 种方式。

RabbitMQ 安装
Zipkin 安装：

```
docker run -d -p 9411:9411 --name zipkin -e ES_HOSTS=192.168.91.128 -e
STORAGE_TYPE=elasticsearch -e ES_HTTP_LOGGING=BASIC -e
RABBIT_URI=amqp://guest:guest@192.168.91.128:5672 openzipkin/zipkin
```

- ES_HOSTS：es 的地址
- STORAGE_TYPE：数据存储方式
- RABBIT_URI：要连接的 Rabbit 的地址

17.2 实践

首先来创建一个 Zipkin 项目，添加 web、sleuth、zipkin、rabbitmq、stream：


17-1.png

项目创建好之后，配置 zipkin 和 rabbitmq：

```
spring.application.name=zipkin01
# 开启链路追踪
```

```
spring.sleuth.web.client.enabled=true
# 配置采样比例，默认为 0.1
spring.sleuth.sampler.probability=1
# zipkin 地址
spring.zipkin.base-url=http://192.168.91.128:9411
# 开启 zipkin
spring.zipkin.enabled=true
# 追踪消息的发送类型
spring.zipkin.sender.type=rabbit

spring.rabbitmq.host=192.168.91.128
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

接下来提供一个测试的 HelloController：

```
@RestController
public class HelloController {
    private static final Logger logger =
LoggerFactory.getLogger(HelloController.class);

    @GetMapping("/hello")
    public String hello(String name) {
        logger.info("zipkin01-hello");
        return "hello " + name + " !";
    }
}
```

然后再创建一个 zipkin02，和 zipkin01 的配置基本一致。