1. Distributed Tracing简介

分布式跟踪是一种监视和配置应用程序的技术，特别是那些使用微服务体系结构构建的应用程序。它也称为分布式请求跟踪。开发人员使用分布式跟踪来调试和优化代码。

分布式跟踪提供了一个我们可以看到“特定请求发生了什么”的地方。这一点很重要，因为微服务涉及到各种各样的组件。

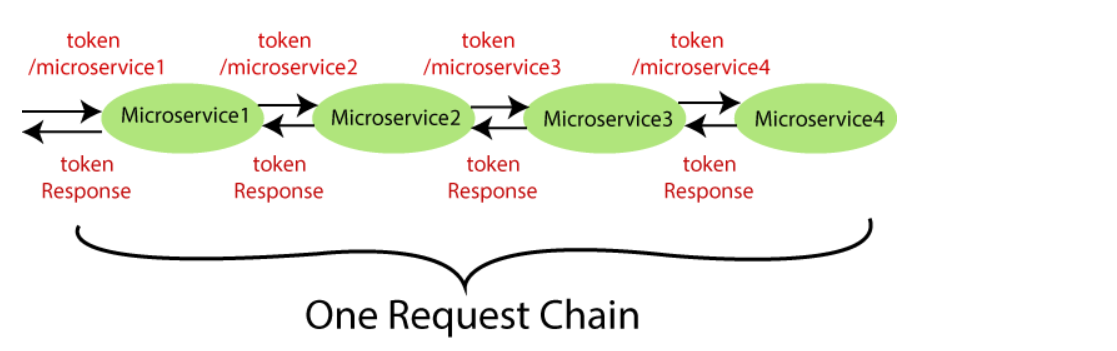
2. Spring Cloud Sleuth & Zipkin

Spring Cloud Sleuth是一个Spring云库，它提供了通过在适当的HTTP请求头文件上添加trace和span Ids来跟踪后续微服务进程的能力。Sleuth库基于MDC(Mapped Diagnostic Context)概念，我们可以轻松地提取值，放入上下文，并将它们显示在日志中。

Zipkin是一个开源的、基于java的分布式跟踪系统。它有一个管理控制台，提供了发送、接收、存储和可视化后续服务的跟踪细节的机制。

在Zipkin服务器的帮助下，我们可以将所有组件的所有日志放到MQ (RabbitMQ)中。我们将日志发送到Zipkin服务器，在那里合并日志。这样做之后，我们可以监视不同的请求。我们还可以发现一个特定的请求发生了什么。

3. 使用Spring Cloud Sleuth实现Distributed Tracing

⑴在这一步中，我们将为所有微服务添加Spring Cloud Sleuth。它向所有请求添加唯一的Id。它用于生成并将trace Id、span Id附加到日志中，以便工具(Zipkin)可以使用这些Id。

⑵Spring Cloud Sleuth组件：

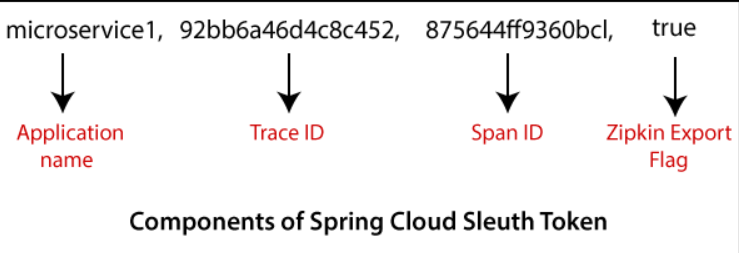
①Application name：在properties文件中定义的应用程序的名称

②Trace Id：Sleuth添加的Trace Id。对于给定的请求，它在所有服务中保持不变。

③Span Id：Sleuth添加的Span Id。它在一个工作单元中保持不变，但对于给定请求，针对不同服务而有所不同。

④Zipkin Export Flag：它表示一个布尔值。它可以是true，也可以是false。

下图显示了Spring Cloud Sleuth token：

⑶项目中使用Spring Cloud Sleuth

①在项目中引入依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-sleuth</artifactId>

</dependency>

②Sampler

分布式跟踪可能有非常大量的数据，因此采样在分布式跟踪中很重要。Spring Cloud Sleuth提供了Sampler策略。在Sampler的帮助下，我们可以实现提供对算法的控制的采样算法。默认情况下，如果span（相关性：是单个操作）已经处于活动状态，我们将获得一个连续执行跟踪的过程。

但是新创建的spans始终标记为不可导出。如果所有应用程序都与Sampler一起运行，则可以在日志中看到trace（由span组成的端到端延迟图），而不是在任何远程位置。 默认情况下，Spring Cloud Sleuth将所有范围设置为不可导出。

当我们将span数据导出到Zipkin或Spring Cloud Stream时，Spring Cloud Sleuth提供了AlwaysSampler类，该类将所有内容导出到Zipkin。它还提供了一个PercentageBasedSampler类，该类对span的固定部分进行采样。

接下来我们需要在每个微服务中添加Sampler的Bean实例，例如：

@Bean

public Sampler defaultSampler(){

return Sampler.ALWAYS\_SAMPLE;

}

在微服务1和2中添加日志信息。

在浏览器地址栏使用Zuul API Gateway请求微服务2，如http://localhost:8765/service2/service2，运行结果如下：

微服务2控制台信息：

微服务1控制台信息：

Zuul ZPI Gateway控制台输出：

⑷项目中使用Zipkin

我们已经在微服务1，微服务2和Zuul API Gateway Server中安装了Spring Cloud Sleuth依赖项。我们已经看到为每个请求分配了唯一的Id。 我们使用这些Id在这些多个服务的日志中跟踪请求。

但是，我们在跟踪方面面临着挑战。如果要跟踪请求，则必须检查单个应用程序的日志。该问题的解决方案称为集中式日志。

我们需要集中所有微服务的所有日志。我们可以搜索Spring Cloud Sleuth分配的Id。在集中的地方，我们将能够搜索并找出特定请求的情况。

集中式日志的解决方案：ELK Stack，Kibana或Zipkin。接下来，我们将通过使用Zipkin实现日志的跟踪。

①RabbitMQ简介

RabbitMQ是广泛部署的开源消息代理软件，它实现了高级消息队列协议(AQMP)。它是轻量级的，易于在云中部署。它支持多个消息传递协议。它可以部署在分布式环境中，以满足高规模和高可用性的需求。它以AMQP标准为模型。RabbitMQ是用Erlang编程语言编写的。它是在开放电信平台(OTP)框架上开发的，用于集群和故障转移。

AMQP定义的内容：在何处发送消息(路由)、如何到达(送货)和进去的必须出来(忠诚)。

消息代理(Message broker)：消息代理位于计算机和分布式计算系统之间。不是直接将消息传递给接收方，而是首先将消息发送给消息代理(RabbitMQ)。message broker在优化队列中对消息进行排序，并在机器准备处理消息时将它们传递给接收机器。

消息代理做的事：将消息发布者和使用者解耦、存储消息、路由消息、消息的监视和管理、在生产者和消费者之间转换消息格式。

②安装Zipkin

在官方下载Docker安装包或者jar，下载jar，启动如下：

SET RABBIT\_URI=amqp://localhost

java -jar zipkin-server-2.19.3-exec.jar

③项目中使用zipkin

i. 首先在项目中添加zipkin依赖(两个微服务和Zuul API Gateway项目中)

添加amqp与RabbitMQ通信

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

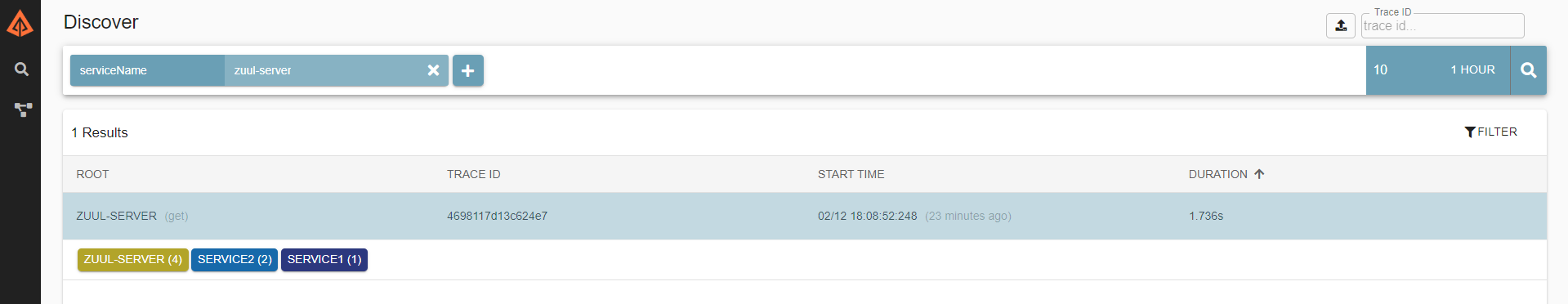
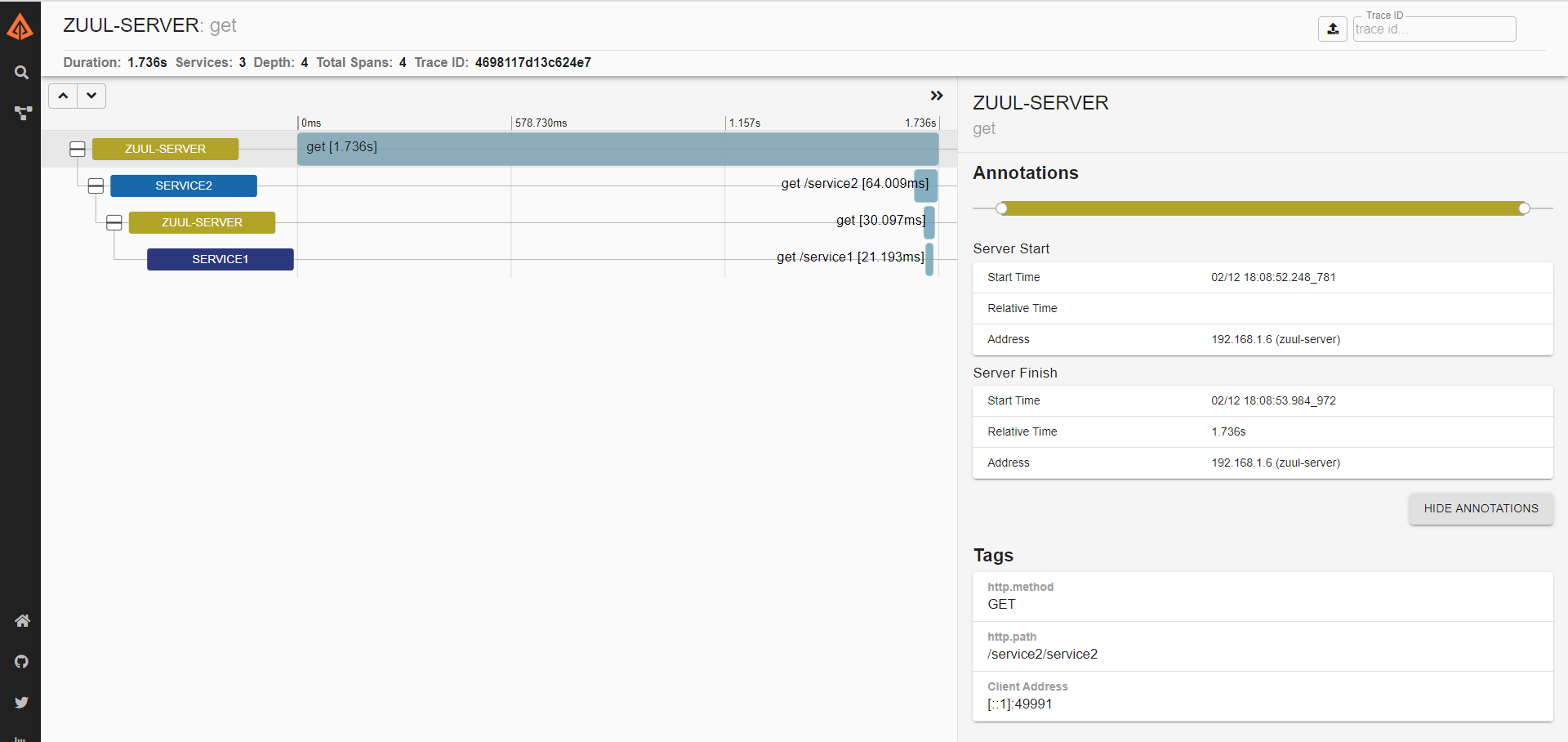
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-sleuth-zipkin</artifactId>

</dependency>

④启动微服务和Zipkin，顺序为Rureka、Config、Zipnkin、微服务和Zuul。

在浏览器输入请求通过Zuul API Gateway，如http://localhost:8765/service2/service2。在Zipkin中查询，相关截图如下：