



下载APP



## 22 | 调用链追踪：集成 Sleuth 和 Zipkin，实现链路打标

2022-02-02 姚秋辰

《Spring Cloud 微服务项目实战》

课程介绍 >



讲述：姚秋辰

时长 21:50 大小 20.00M



你好，我是姚秋辰。

在上节课中，我们讲了链路追踪技术在线上故障排查中的重要作用，以及 Sleuth 是如何通过“打标记”来实现链路追踪的。

今天，我们就通过实战来落地一套完整的调用链追踪方案。在实战阶段，我会为你详细讲解 Sleuth 在调用链中传递标记信息的原理。为了进一步提高线上故障排查效率，我们还会搭建 Zipkin 组件作为链路追踪的数据可视化工具，并通过一条高可用的数据传输通道借助 RabbitMQ 将日志信息从应用服务器传递到 Zipkin 服务器。当你学完这节课，你就可以掌握“调用链追踪”方案搭建的全过程了。



链路打标是整个调用链追踪方案的基础功能，所以我们就先从这里开始，在实战项目中集成 Sleuth，实现日志打标动作。

## 集成 Sleuth 实现链路打标

我们的微服务模块在运行过程中会输出各种各样的日志信息，为了能在日志中打印出特殊的标记，我们需要将 Sleuth 的打标功能集成到各个微服务模块中。

Sleuth 提供了一种无感知的集成方案，只需要添加一个依赖项，再做一些本地启动参数配置就可以开启打标功能了，整个过程不需要做任何的代码改动。

所以第一步，我们需要将 Sleuth 的依赖项添加到模板服务、优惠计算服务和用户服务的 pom.xml 文件中。具体代码如下。

[复制代码](#)

```
1 <!-- Sleuth依赖项 -->
2 <dependency>
3     <groupId>org.springframework.cloud</groupId>
4     <artifactId>spring-cloud-starter-sleuth</artifactId>
5 </dependency>
```

第二步，我们打开微服务模块的 application.yml 配置文件，在配置文件中添加采样率和每秒采样记录条数。


[复制代码](#)

```
1 spring:
2   sleuth:
3     sampler:
4       # 采样率的概率，100%采样
5       probability: 1.0
6       # 每秒采样数字最高为100
7       rate: 1000
```

你可以从代码中看到，我在配置文件里设置了一个 **probability**，它应该是一个 0 到 1 的浮点数，用来表示**采样率**。我这里设置的 probability 是 1，就表示对请求进行 100% 采样。如果我们把 probability 设置成小于 1 的数，就说明有的请求不会被采样。如果一个请求未被采样，那么它将不会被调用链追踪系统 Track 起来。

你还会在代码中看到 **rate 参数**，它代表**每秒最多可以对多少个 Request 进行采样**。这有点像一个“限流”参数，如果超过这个阈值，服务请求仍然会被正常处理，但调用链信息不会被采样。

到这里，我们的 Sleuth 集成工作就已经搞定了。这时你只要启动项目，顺手调用几个 API，就能在控制台的日志信息里看到 Sleuth 默认打印出来的 Trace ID 和 Span ID。比如我这里调用了 Customer 服务的优惠券查询接口，在日志中，你可以看到两串随机生成的数字和字母混合的 ID，其中排在前面的那个 ID 就是 Trace ID，而后面则是 Span ID。

 复制代码

```
1 DEBUG [coupon-customer-serv,69e433d6432522e4,936d8af942b703d2] 81584
2 --- [io-20002-exec-1] c.g.c.customer.feign.TemplateService:xxxx
```

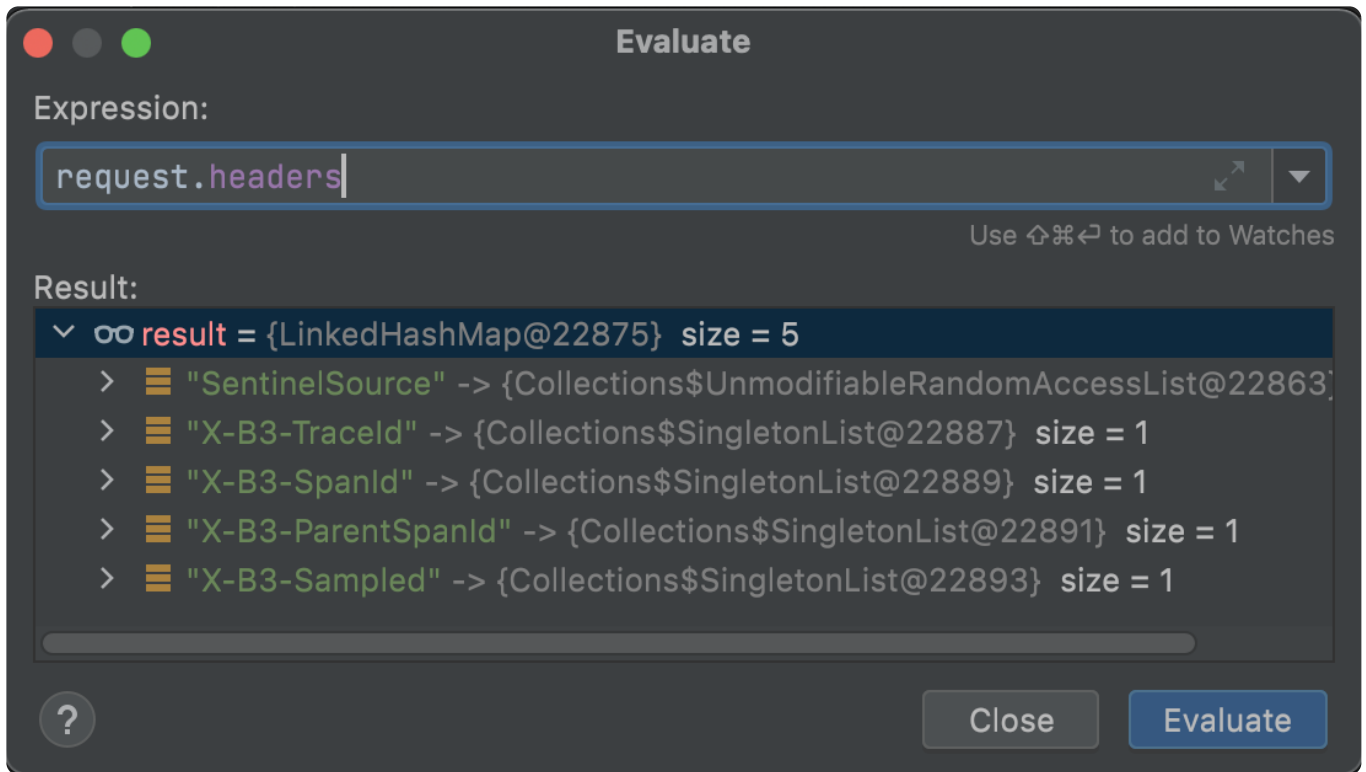
接下来问题来了，在跨服务的调用链中，你知道 Sleuth 是如何将这些标记从一个微服务传递给下一个微服务的吗？接下来我们就去看看 Sleuth 具体动了哪些手脚吧。

## Sleuth 如何在调用链中传递标记

以 Customer 微服务为例，在我们访问 findCoupon 接口查询优惠券的时候，用户微服务通过 OpenFeign 组件向 Template 微服务发起了一次查询请求。

Sleuth 为了将 Trace ID 和 Customer 服务的 Span ID 传递给 Template 微服务，它在 OpenFeign 的环节动了一个手脚。Sleuth 通过 **TracingFeignClient 类**，将一系列 Tag 标记塞进了 OpenFeign 构造的服务请求的 Header 结构中。

我在 TracingFeignClient 的类中打了一个 Debug 断点，将 Request 的 Header 信息打印了出来：



在这个 Header 结构中，我们可以看到有几个以 X-B3 开头的特殊标记，这个 X-B3 就是 Sleuth 的特殊接头暗号。其中 X-B3-TraceId 就是全局唯一的链路追踪 ID，而 X-B3-SpanId 和 X-B3-ParentSpanId 分别是当前请求的单元 ID 和父级单元 ID，最后的 X-B3-Sampled 则表示当前链路是否是一个已被采样的链路。通过 Header 里的这些信息，下游服务就完整地得到了上游服务的情报。

以上是 Sleuth 对 OpenFeign 动的手脚。为了应对调用链中可能出现的各种不同组件，Sleuth 内部构造了各式各样的适配器，用来在不同组件中使用同样的接头暗号 “X-B3-\*”，这样就可以传递链路追踪的信息。如果你对这部分的源码感兴趣，你可以深入研究 spring-cloud-sleuth-instrumentation 和 spring-cloud-sleuth-brave 两个依赖包的源代码，了解更加详细的实现过程。

搞定了链路打标之后，我们怎样才能通过 Trace ID 来查询链路信息呢？这时就要找 Zipkin 来帮忙了。

## 使用 Zipkin 收集并查看链路数据

Zipkin 是一个分布式的 Tracing 系统，它可以用来收集时序化的链路打标数据。通过 Zipkin 内置的 UI 界面，我们可以根据 Trace ID 搜索出一次调用链所经过的所有访问单元，并获取每个单元在当前服务调用中所花费的时间。

为了搭建一条高可用的链路信息传递通道，我将使用 RabbitMQ 作为中转站，让各个应用服务器将服务调用链信息传递给 RabbitMQ，而 Zipkin 服务器则通过监听 RabbitMQ 的队列来获取调用链数据。相比于让微服务通过 Web 接口直连 Zipkin，**使用消息队列可以大幅提高信息的送达率和传递效率。**

我画了一张图来帮你理解 Zipkin 和微服务之间是如何通信的，你可以参考一下。



下面我来带你手动搭建 Zipkin 服务器。

## 搭建 Zipkin 服务器

首先，我们要下载一个 Zipkin 的可执行 jar 包，这里我推荐你使用 2.23.9 版本的 Zipkin 组件。你可以通过访问 [maven 的中央仓库](#) 下载 zipkin-server-2.23.9-exec.jar 文件，我已经将版本参数添加到了地址中，不过你可以将地址超链接复制出来，通过修改 URL 中的版本参数来下载指定版本。

搭建 Zipkin 有两种方式，一种是直接下载 Jar 包，这是官方推荐的标准集成方式；另一种是通过引入 Zipkin 依赖项的方式，在本地搭建一个 Spring Boot 版的 Zipkin 服务器。如果你需要对 Zipkin 做定制化开发，那么可以采取后一种方式。

接下来，我们需要在本地启动 Zipkin 服务器。我们打开命令行，在下载下来的 jar 包所在目录执行以下命令，就可以启动 Zipkin 服务器了。

复制代码

```
1 java -jar zipkin-server-2.23.9-exec.jar --zipkin.collector.rabbitmq.addresses=
```



要注意的是，我在命令行中设置了 `zipkin.collector.rabbitmq.addresses` 参数，所以 Zipkin 在启动阶段将尝试连接 RabbitMQ，你需要**确保 RabbitMQ 始终处于启动状态**。Zipkin 已经为我们内置了 RabbitMQ 的默认连接属性，如果没有特殊指定，那么 Zipkin 会使用 `guest` 默认用户登录 RabbitMQ。如果你想要切换用户、指定默认监听队列或者设置连接参数，那么可以在命令行中添加以下参数进行配置。

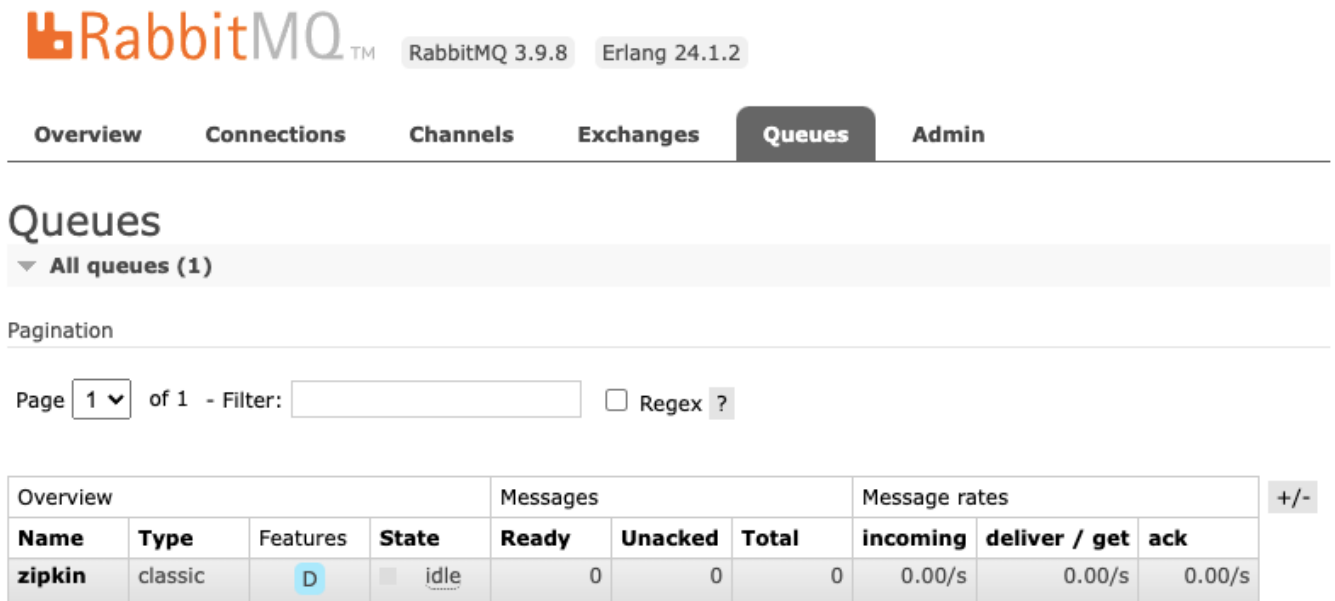
zipkin.collector.rabbitmq.password	RabbitMQ密码，默认guest
zipkin.collector.rabbitmq.username	RabbitMQ用户名，默认guest
zipkin.collector.rabbitmq.connection-timeout	RabbitMQ建立连接的超时时间
zipkin.collector.rabbitmq.queue	监听的消息队列，默认为zipkin
zipkin.collector.rabbitmq.use-ssl	是否使用SSL建立连接

启动成功后，你可以在命令行看到 Zipkin 的特色 Logo，以及一行 Serving HTTP 的运行日志。

[illegible]

最后，我们只需要验证消息监听队列是否已就位就可以了。我们使用 guest 账号登录 RabbitMQ，并切换到 “Queues” 面板，如果 Zipkin 和 RabbitMQ 的对接一切正常，那

么你会在 Queues 面板下看到一个名为 zipkin 的队列，如下图所示。



RabbitMQ 3.9.8 Erlang 24.1.2

Overview Connections Channels Exchanges **Queues** Admin

## Queues

▼ All queues (1)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
zipkin	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	

到这里，我们就完成了 Zipkin 服务器的创建。接下来，你还需要将应用程序生成的链路数据发送给 Zipkin 服务器。

## 传送链路数据到 Zipkin

我在方案中使用 RabbitMQ 作为中转站来传递链路调用数据，因此应用程序并不需要直连 Zipkin，而是需要接入到 RabbitMQ，并将链路数据发布到 RabbitMQ 中的“zipkin”队列中就可以了。


首先，我们需要在每个微服务模块的 pom.xml 中添加 Zipkin 适配插件和 Stream 的依赖。其中，Stream 是 Spring Cloud 中专门用来对接消息中间件的组件，我会在下个章节为你详细讲解它。

复制代码

```
1 <dependency>
2     <groupId>org.springframework.cloud</groupId>
3     <artifactId>spring-cloud-sleuth-zipkin</artifactId>
4 </dependency>
5 <!-- 提前剧透Stream -->
6 <dependency>
7     <groupId>org.springframework.cloud</groupId>
8     <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
9 </dependency>
```

接下来，我们需要将 Zipkin 的配置信息添加到每个微服务模块的 application.yml 文件中。

在配置项中，我通过 zipkin.sender.type 属性指定了传输类型为 RabbitMQ，除了 RabbitMQ 以外，Zipkin 适配器还支持 ActiveMQ、Kafka 和直连的方式，我推荐你**使用 Kafka 和 RabbitMQ 来保证消息投递的可靠性和高并发性**。我还通过 spring.zipkin.rabbitmq 属性声明了消息组件的连接地址和消息投递的队列名称。

 复制代码

```
1 spring:
2   zipkin:
3     sender:
4       type: rabbit
5     rabbitmq:
6       addresses: 127.0.0.1:5672
7       queue: zipkin
```

有一点你需要注意，**在应用中指定的队列名称，一定要同 Zipkin 服务器所指定的队列名称保持一致**，否则 Zipkin 无法消费链路追踪数据。

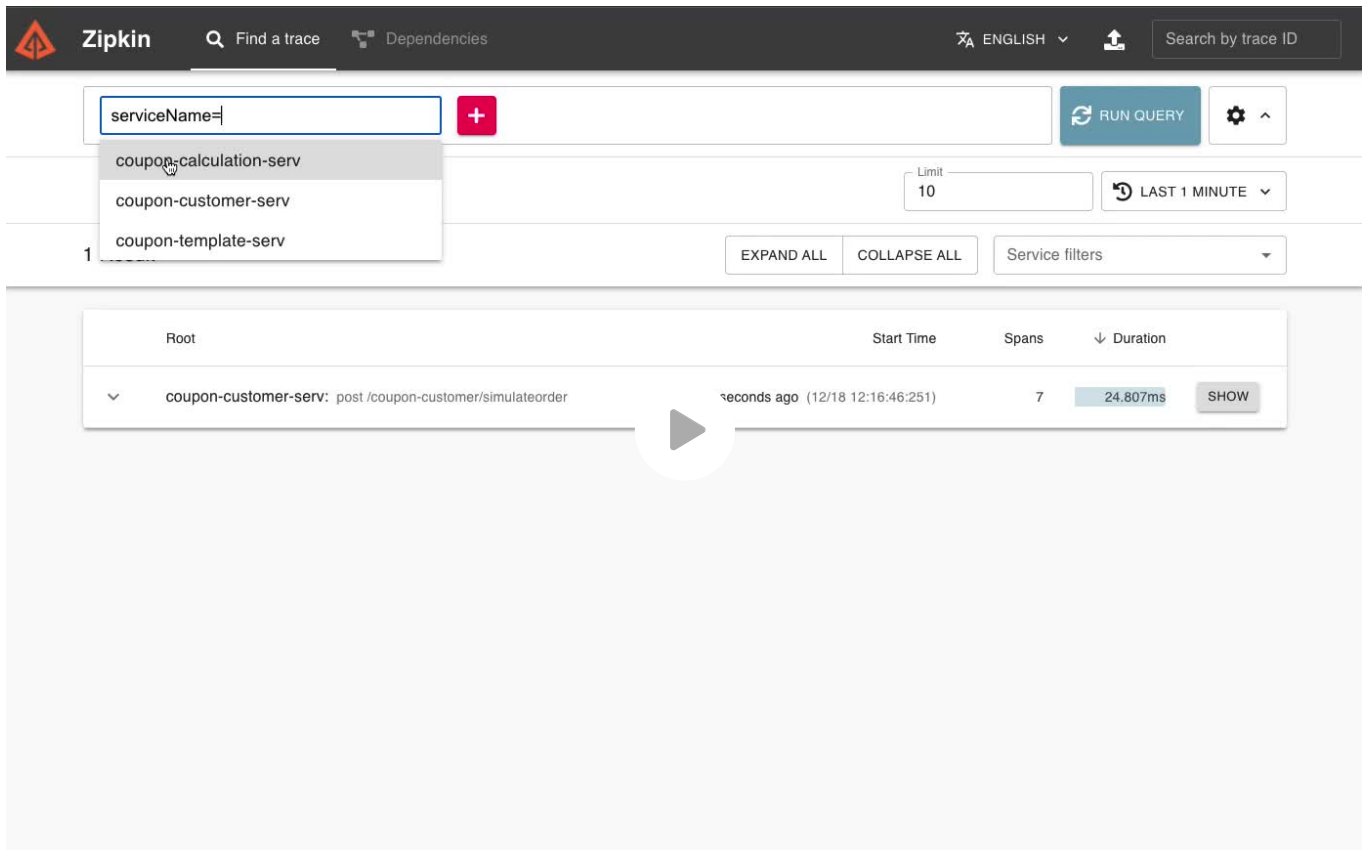
到这里，我们就完成了一套完整的链路追踪系统的搭建，是不是很简单呢？接下来，就可以把你的应用启动起来，通过 Postman 发起几个跨服务的调用了。我来带你去 Zipkin 上看一下可视化的链路追踪数据长啥样。

## 查看链路追踪信息

你可以在浏览器中打开 localhost:9411 进到 Zipkin 的首页，在首页中你可以通过各种搜索条件的组合，从服务、时间等不同维度查询调用链数据。

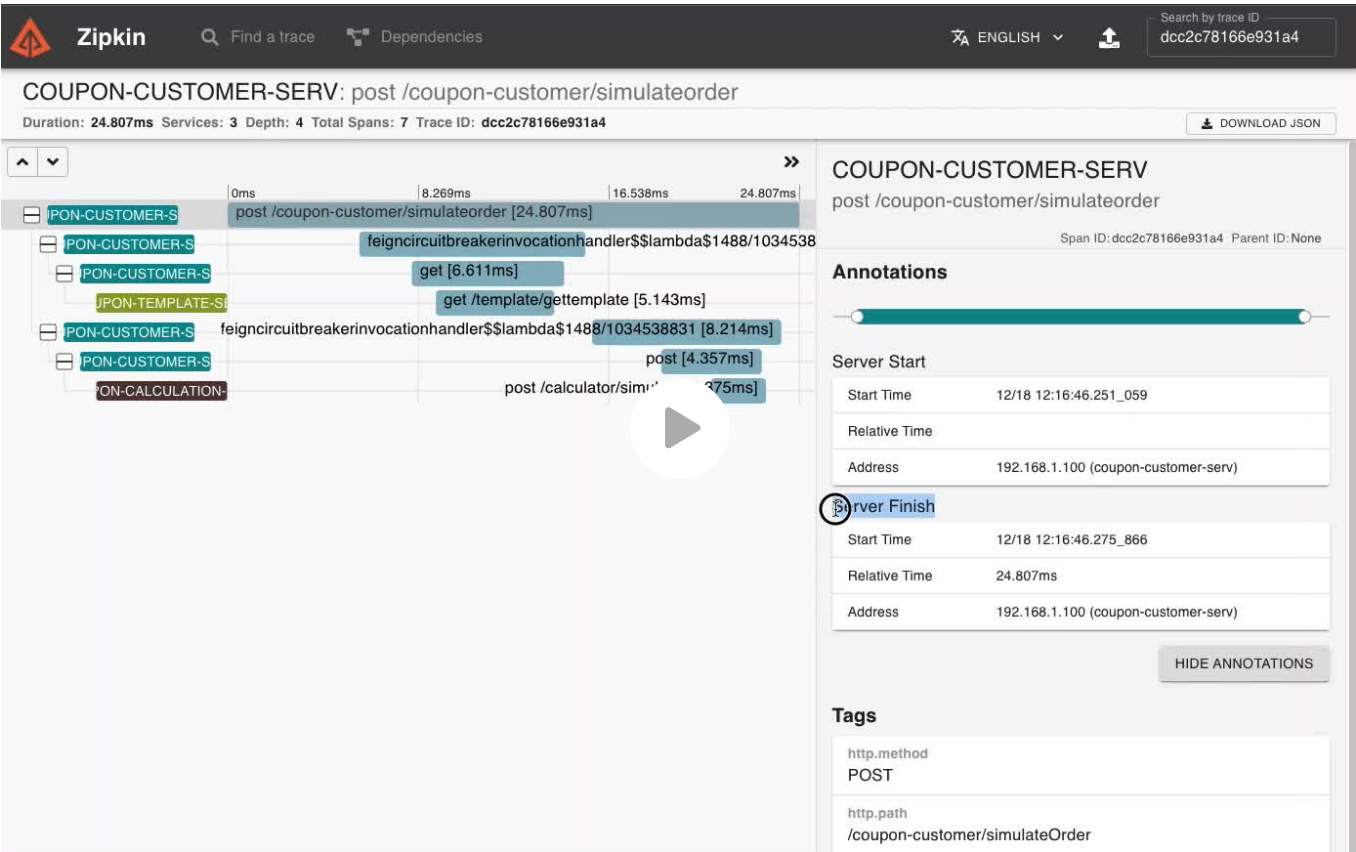
我在本地调用了 Customer 服务的订单价格试算接口，而 Customer 服务又相继调用了 Template 服务和 Calculation 服务，现在我就用一段小 video 来演示如何在 Zipkin 上查询调用链数据。



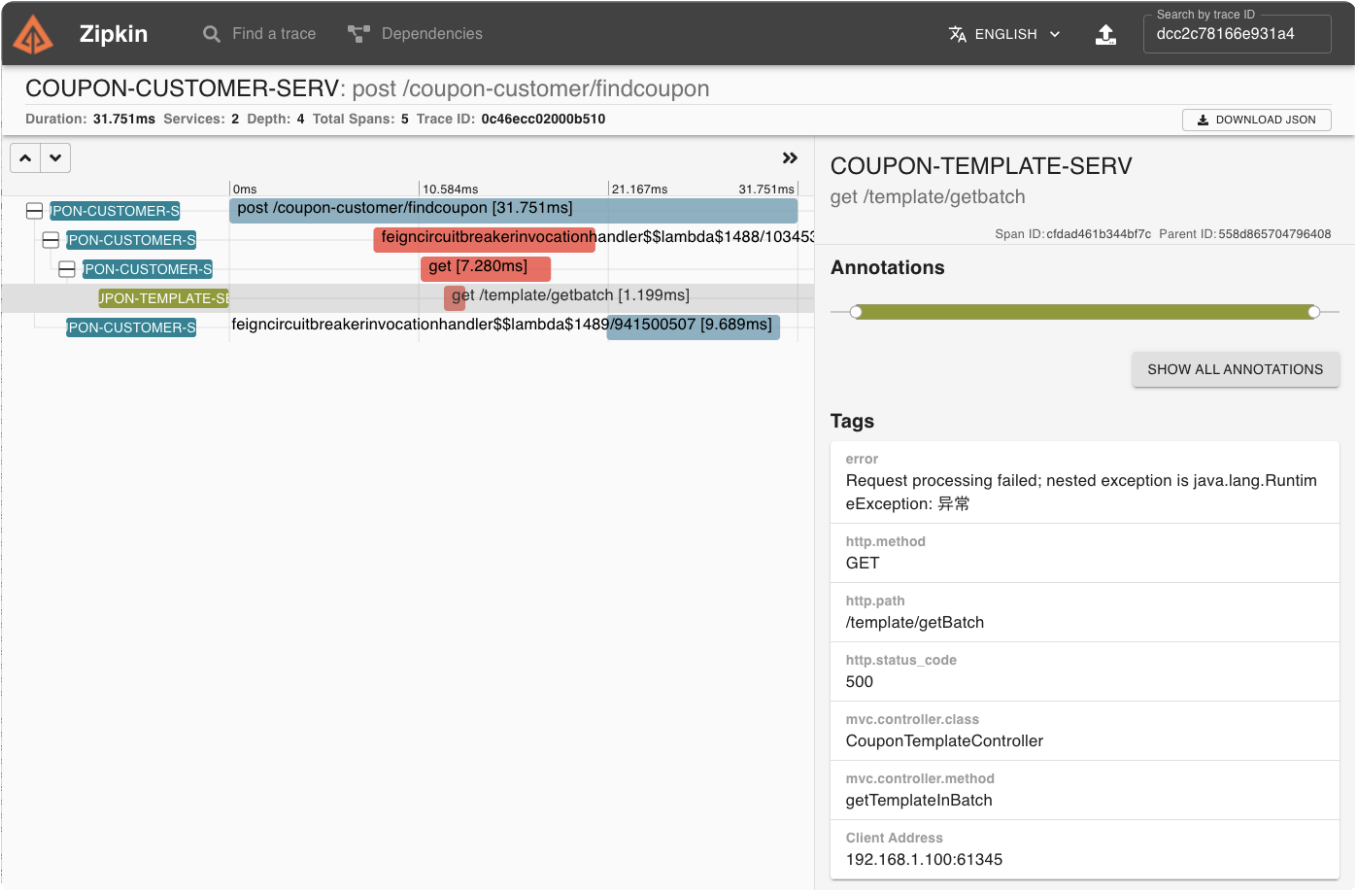


在这段 video 中，我可以选择想要搜索的时间范围，还可以搜索包含特定服务的调用链。而在搜索结果中，当前调用链都访问了哪些微服务，你可以一目了然。

如果你知道了某个调用链的全局唯一 Trace ID，那么你也可以通过这个 Trace ID 把一整条调用链路查出来。我又录了一段 video 来演示这个过程。在链路详情页面中，**所有 Span 都以时间序列的先后顺序进行排布**，你可以从链路中清晰地看到每个 Span 的开始、结束时间，以及处理用时。

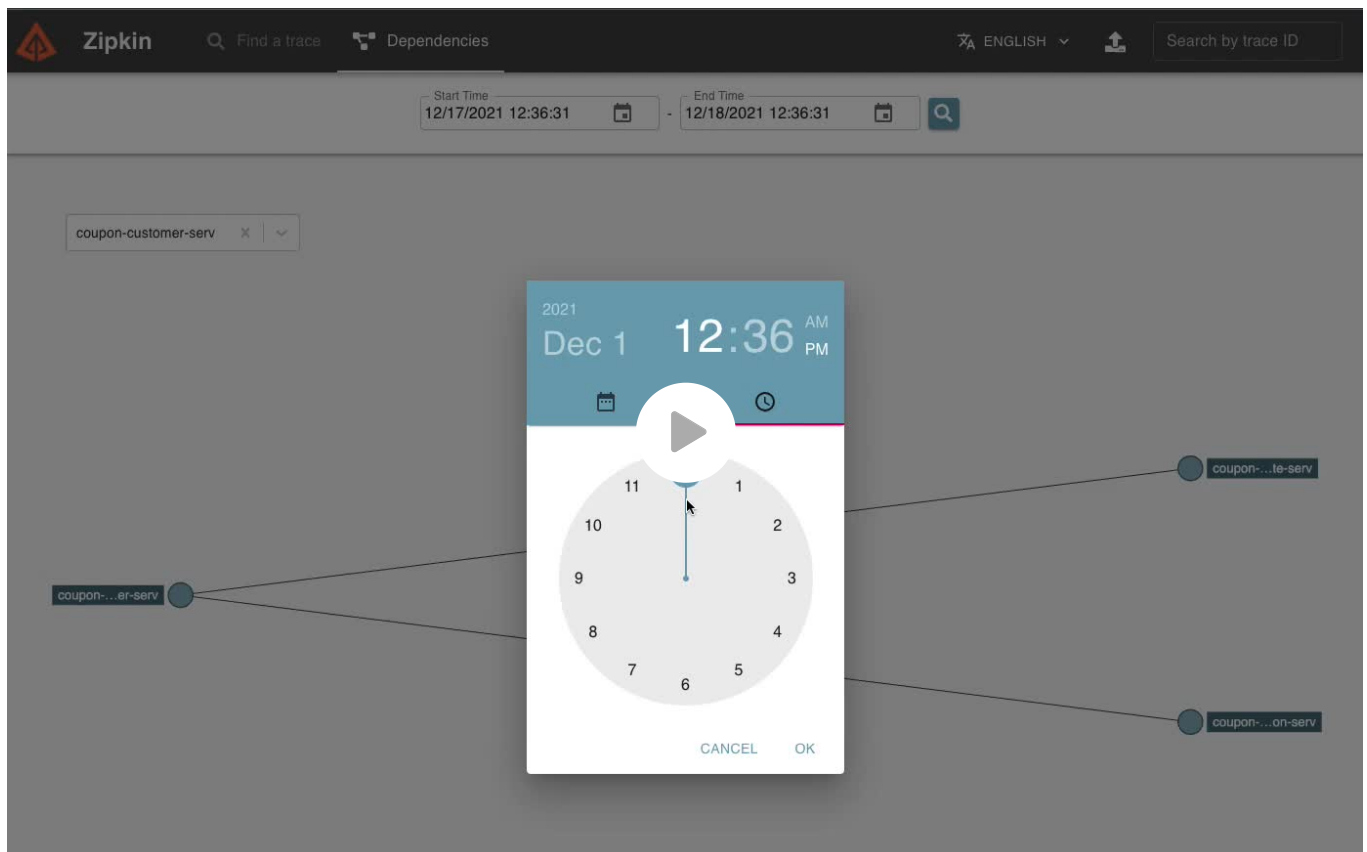


如果某个调用链出现了运行期异常，那么你可以从调用链中轻松看出异常发生在哪个阶段。比如下图中的调用链在 OpenFeign 调用 Template 服务的时候抛出了 RuntimeException，相关 Span 在页面上已被标红，如果你点击 Span 详情，就可以看到具体的 Error 异常提示信息。



除此之外，Zipkin 还有一个很花哨的依赖报表功能，它会以图形化的方式展示某段时间内微服务之间的相互调用情况，如果两个微服务之间有调用关系，Zipkin 就会用一条实线将两者关联起来，而实线上流动的小圆点则表示调用量的多少，圆点越多则表示这条链路的流量越多。而且，小圆点还会有红蓝两种颜色，其中红色表示调用失败，蓝色表示调用成功。

我录了一小段 video，你可以感受一下依赖报表功能是怎么用的。



到这里，相信你已经对调用链追踪系统的搭建和使用十分了解了，现在让我们来回顾一下这节课的重点内容吧。

## 总结

今天我们通过集成 Sleuth 和 Zipkin，搭建了一套完整的链路追踪系统。**链路追踪的核心是“标记”**，也就是 Sleuth 在链路中打上的 Trace ID 等标记，我推荐你从 Sleuth 的源码入手，了解一下 Sleuth 是如何为每个不同的组件编写适配器，完成打标和标记传递的。

Zipkin 在默认情况下将链路数据保存在内存中，默认最多保存 50000 个 Span 数据，所以这种保存数据的方式是不能应用在生产环境中的。

Zipkin 天然支持通过 Cassandra、ElasticSearch 和 MySQL 这三种方式保存数据，如果你想要将内存方式切换为其它数据源，则需要在启动命令中添加数据源的连接信息，相关启动参数可以在 [Sleuth 的配置文件](#) 中找到。在这个链接中，你可以在 zipkin.storage 节点下找到每个数据源的参数列表，通过 zipkin.storage.type 字段你可以指定 Zipkin 的数据源。

在 Spring Boot 2.0 之后，Zipkin 的官方社区就不再推荐我们通过自定义的方式搭建 Zipkin Server 端了。除非有很特殊的定制需求，否则我还是推荐你使用 zipkin 的可执行

jar 包，并通过标准的启动参数来搭建 Zipkin 服务器。

## 思考题

根据 [Sleuth 配置文件](#) 中的参数定义，你能通过传入启动参数的方式，对 Zipkin 做一个改造，并使用 MySQL 作为数据源吗？欢迎你在评论区把自己的改造过程分享出来。

好啦，这节课就结束啦。欢迎你把这节课分享给更多对 Spring Cloud 感兴趣的朋友。我是姚秋辰，我们下节课再见！

分享给需要的人，Ta 购买本课程，你将得 20 元

 生成海报并分享

 赞 0  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 21 | Sleuth 体系架构：为什么微服务架构需要链路追踪？

下一篇 23 | 调用链追踪：如何通过 ELK 实现日志检索？

## 精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。