服务链路追踪

极客时间

扫码试看/订阅

《玩转 Spring 全家桶》

通过 Dapper 理解链路治理

我们在关注什么？

- 系统中都有哪些服务
- 服务之间的依赖关系是什么样的
- 一个常见请求具体的执行路径是什么样的
- 请求每个环节的执行是否正常与耗时情况 服务究竟慢在哪
- …… 采样 日志

Google Dapper 的一些术语

一篇paper

google官方论文库

- Span - 基本的工作单元
- Trace - 由一组 Span 构成的树形结构

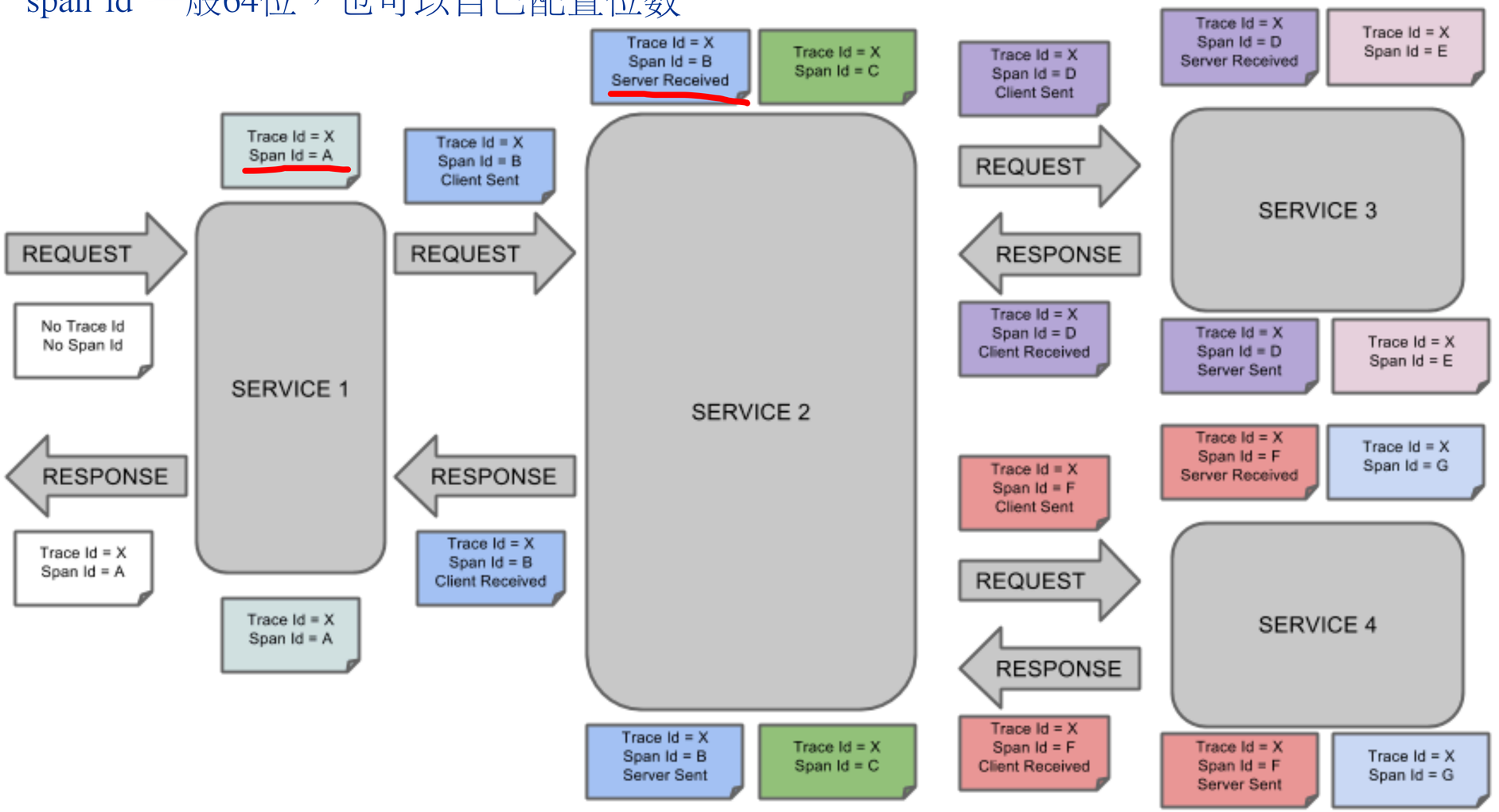
一次业务操作，一个系统的业务ID不会变

- Annotation - 用于及时记录事件
  - cs - Client Sent
  - sr - Server Received
  - ss - Server Sent
  - cr - Client Received

ms,mr消息发送和接收

Google Dapper

span id 一般64位，也可以自己配置位数



通过 Spring Cloud Sleuth 实现链路追踪

Spring Cloud 提供的服务治理功能

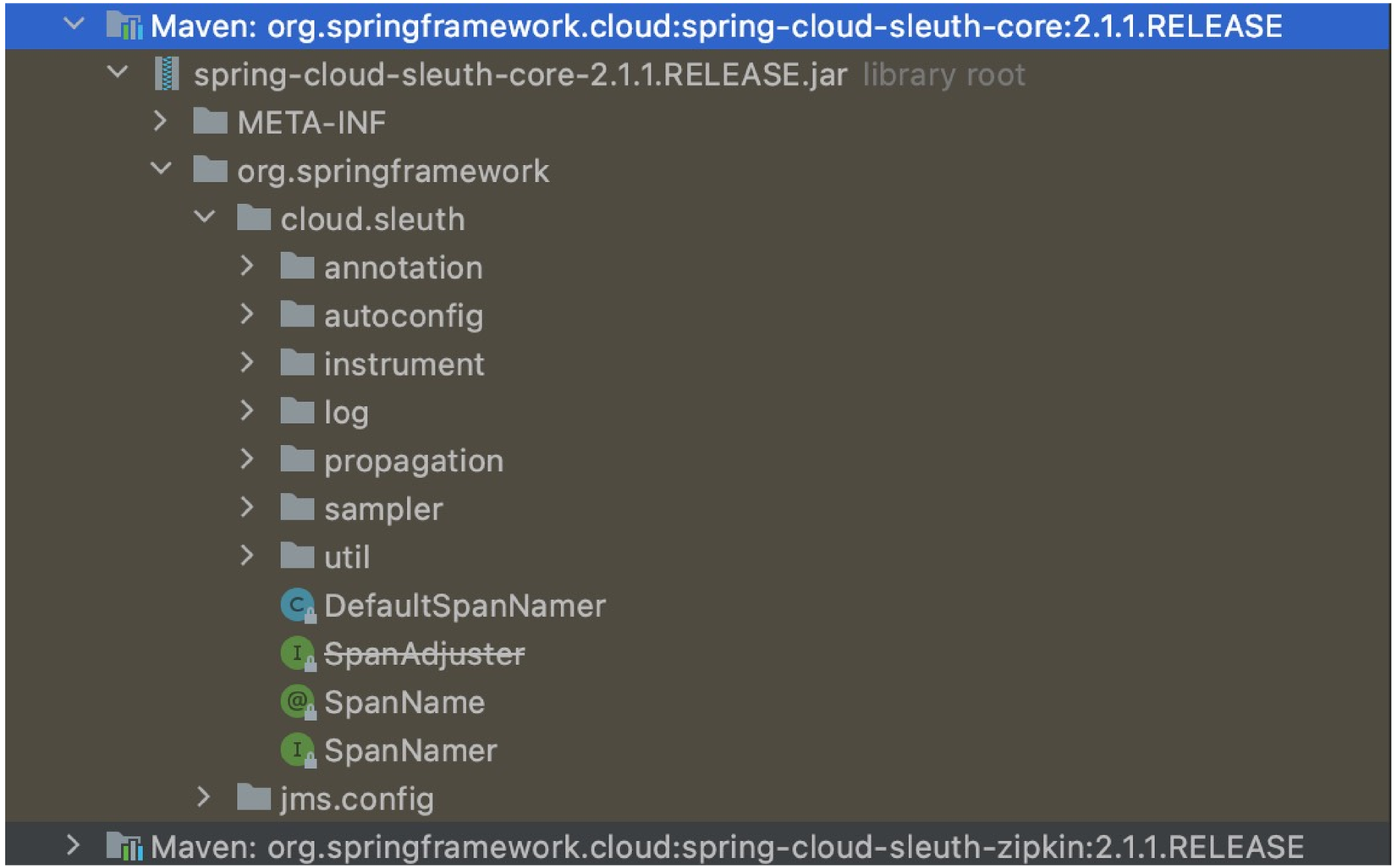
依赖

- Spring Cloud Sleuth - spring-cloud-starter-sleuth
- Spring Cloud Sleuth with Zipkin - spring-cloud-starter-zipkin

日志输出　　输出到open zipkin

- `[appname,traceId,spanId,exportable]`

埋的点是不是输出到外部系统里，比如open zipkin


Maven: org.springframework.cloud:spring-cloud-sleuth-core:2.1.1.RELEASE
spring-cloud-sleuth-core-2.1.1.RELEASE.jar library root
META-INF
org.springframework
cloud.sleuth
annotation
autoconfig
instrument
log
propagation
sampler
util
DefaultSpanNamer
SpanAdjuster
SpanName
SpanNamer
jms.config
Maven: org.springframework.cloud:spring-cloud-sleuth-zipkin:2.1.1.RELEASE

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

已经依赖了：

spring-cloud-sleuth-sipkin

spring-cloud-starter-sleuth

Spring-cloud-sleuth使用的是brave来做的相关处理

brave是open zipkin推荐的java客户端之一

Spring Cloud 提供的服务治理功能

配置

- spring.zipkin.base-url=http://localhost:9411/
- spring.zipkin.discovery-client-enabled=false
- spring.zipkin.sender.type=web | rabbit | kafka
- spring.zipkin.compression.enabled=false
- spring.sleuth.sampler.probability=0.1

也可以服务发现找zipkin

向zipkin埋点方式：

1. http请求，发web请求
2. 通过mq方式

sleuth采样比例，如果是1的话就是所有都发送

通过 Docker 启动 Zipkin

官方指引

- https://hub.docker.com/r/openzipkin/zipkin
- https://github.com/openzipkin/docker-zipkin

获取镜像

- `docker pull openzipkin/zipkin`　没有配置就是使用内存数据库

运行 Zipkin 镜像

- `docker run --name zipkin -d -p 9411:9411 openzipkin/zipkin`

“Talk is cheap, show me the code.”

Chapter 16 / sleuth-waiter-service sleuth-customer-service

如何追踪消息链路

用 Spring Cloud Sleuth 追踪消息

通过消息埋点

依赖

- Spring Cloud Sleuth with Zipkin - spring-cloud-starter-zipkin
  - 如需通过 MQ 埋点，需增加 RabbitMQ 或 Kafka 依赖

配置

- 如使用 HTTP 埋点，则与追踪 HTTP 服务完全一致
- `spring.zipkin.sender.type=rabbit`

配置rabbitmq

- `spring.zipkin.rabbitmq.queue=zipkin`
- `spring.rabbitmq.*`

让 Zipkin 能通过 RabbitMQ 接收消息

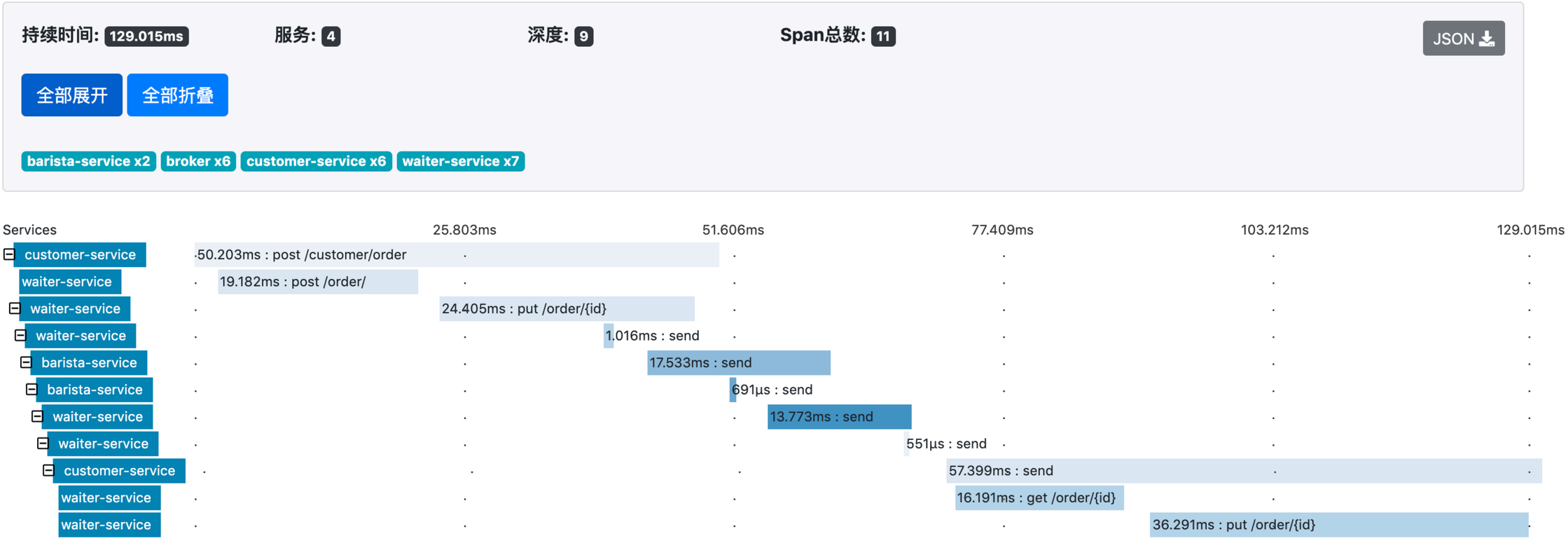
环境变量

- RABBIT_ADDRESSES=<RabbitMQ地址>
- RABBIT_USER / RABBIT_PASSWORD 用户名和密码
- https://github.com/apache/incubator-zipkin/tree/master/zipkin-collector/rabbitmq

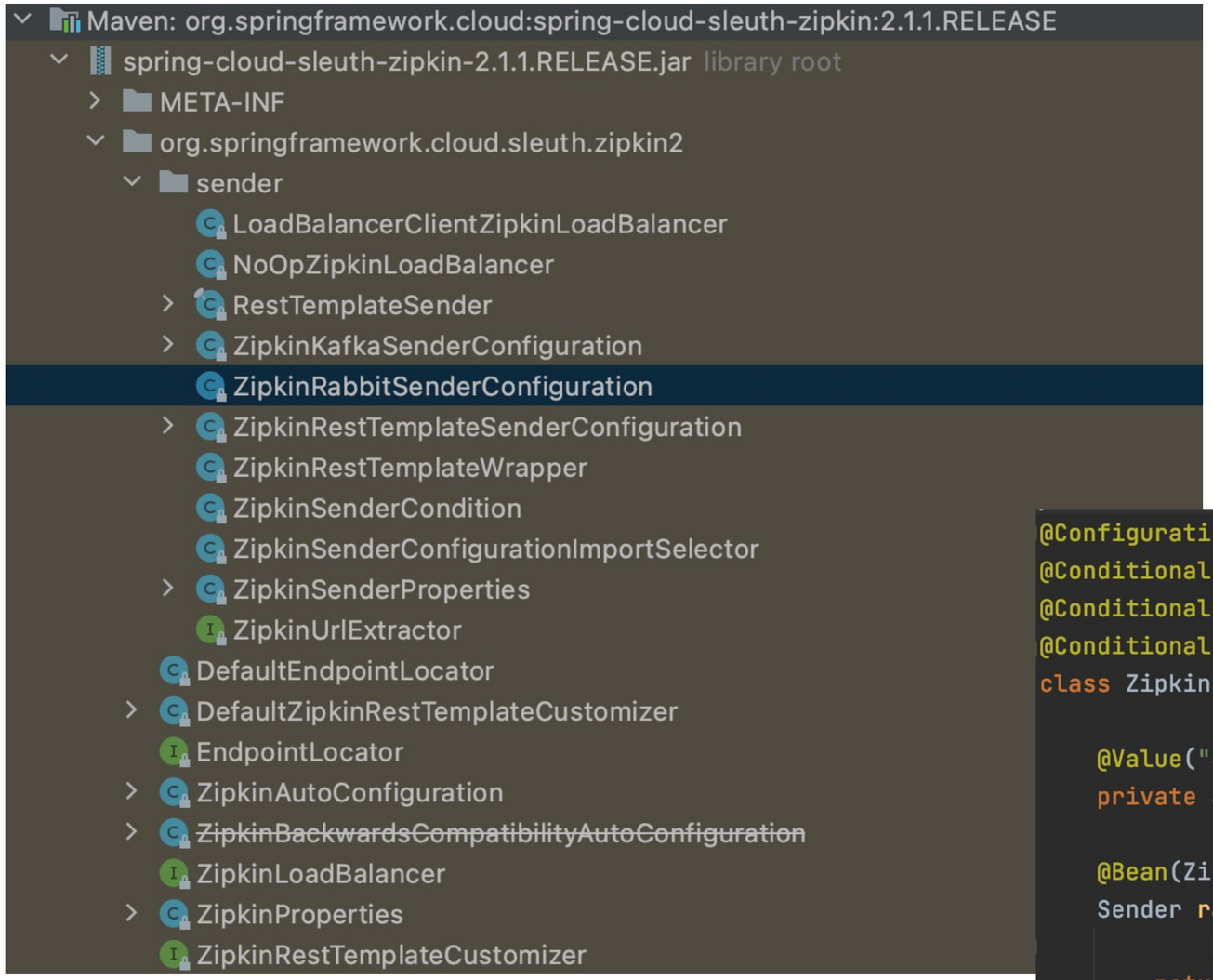
运行 Zipkin 镜像

- ```
  docker run --name rabbit-zipkin -d -p 9411:9411
   --link rabbitmq -e RABBIT_ADDRESSES=rabbitmq:5672
   -e RABBIT_USER=spring -e RABBIT_PASSWORD=spring openzipkin/
  zipkin
  ```
  ```

完整的应用链路


持续时间: 129.015ms
服务: 4
深度: 9
Span总数: 11
JSON
全部展开
全部折叠
barista-service x2
broker x6
customer-service x6
waiter-service x7
Services
25.803ms
51.606ms
77.409ms
103.212ms
129.015ms
customer-service
50.203ms : post /customer/order
waiter-service
19.182ms : post /order/
waiter-service
24.405ms : put /order/{id}
waiter-service
1.016ms : send
barista-service
17.533ms : send
barista-service
691µs : send
waiter-service
13.773ms : send
waiter-service
551µs : send
customer-service
57.399ms : send
waiter-service
16.191ms : get /order/{id}
waiter-service
36.291ms : put /order/{id}

如何控制使用Web还是rabbit：



现在版本都用brave做zipkin支持，
不再需要用stream server方式处理了

```
@Configuration
@ConditionalOnBean(CachingConnectionFactory.class)
@ConditionalOnMissingBean(name = ZipkinAutoConfiguration.SENDER_BEAN_NAME)
@Conditional(ZipkinSenderCondition.class)
class ZipkinRabbitSenderConfiguration {

    @Value("${spring.zipkin.rabbitmq.queue:zipkin}")
    private String queue;

    @Bean(ZipkinAutoConfiguration.SENDER_BEAN_NAME)
    Sender rabbitSender(CachingConnectionFactory connectionFactory,
            RabbitProperties config) {
        return RabbitMQSender.newBuilder()
                .connectionFactory(connectionFactory.getRabbitConnectionFactory())
                .queue(this.queue).addresses(config.determineAddresses()).build();
    }

}
```

```java
class ZipkinSenderCondition extends SpringBootCondition {

    @Override
    public ConditionOutcome getMatchOutcome(ConditionContext context,
            AnnotatedTypeMetadata md) {
        String sourceClass = "";
        if (md instanceof ClassMetadata) {
            sourceClass = ((ClassMetadata) md).getClassName();
        }
        ConditionMessage.Builder message = ConditionMessage.forCondition( condition: "ZipkinSender",
                sourceClass);
        String property = context.getEnvironment()
                .getProperty("spring.zipkin.sender.type");
        if (StringUtils.isEmpty(property)) {
            return ConditionOutcome.match(message.because( reason: "automatic sender type"));
        }
        String senderType = getType(((AnnotationMetadata) md).getClassName());
        if (property.equalsIgnoreCase(senderType)) {
            return ConditionOutcome.match(message.because( reason: property + " sender type"));
        }
        return ConditionOutcome.noMatch(message.because( reason: property + " sender type"));
    }

}
```

满足条件才会创建rabbit sender

```java
public class ZipkinAutoConfiguration {

    Zipkin reporter bean name. Name of the bean matters for supporting multiple tracing systems.
    public static final String REPORTER_BEAN_NAME = "zipkinReporter";

    Zipkin sender bean name. Name of the bean matters for supporting multiple tracing systems.
    public static final String SENDER_BEAN_NAME = "zipkinSender";

    @Bean(REPORTER_BEAN_NAME)
    @ConditionalOnMissingBean(name = REPORTER_BEAN_NAME)
    public Reporter<Span> reporter(ReporterMetrics reporterMetrics,
            ZipkinProperties zipkin, @Qualifier(SENDER_BEAN_NAME) Sender sender) {
        // historical constraint. Note: AsyncReporter supports memory bounds
        return AsyncReporter.builder(sender).queuedMaxSpans(1000)
                .messageTimeout(zipkin.getMessageTimeout(), TimeUnit.SECONDS)
                .metrics(reporterMetrics).build(zipkin.getEncoder());
    }

    @Bean
    @ConditionalOnMissingBean
    public ZipkinRestTemplateCustomizer zipkinRestTemplateCustomizer(
            ZipkinProperties zipkinProperties) {
        return new DefaultZipkinRestTemplateCustomizer(zipkinProperties);
    }
```

“Talk is cheap, show me the code.”

Chapter 16 / mq-zipkin-barista-service

除了链路还要追踪什么

服务治理关心什么才好

我们已经看过了

- 简单服务之间的依赖关系
- 一个请求的同步、异步链路

我们还需要关注

- 很多……很多……

“一个企业实施的用以保障事情正确完成的流程，即遵循最佳实践，体系架构原则，治理条例，法律和其他决定因素。SOA治理是指用于管理SOA的采用和实现的流程。”

– Anne Thomas Manes
Wikipedia - SOA governance

服务治理关心什么才好

宏观上

- 架构设计是否合理
- 哪些链路算是关键链路
- 链路的容量水位趋势
- 对系统变更的管理与审计

微观上

- 一个系统都依赖了什么
- 一个系统都有哪些配置
- 一个系统的主观与客观质量

……

SpringBucks 实战项目进度小结

本章小结

Spring Cloud 的服务治理功能

- 借鉴自 Google Dapper
- Spring Cloud Sleuth
  - Zipkin
    - Web
    - RabbitMQ
- 我们应该关心更多

SpringBucks 进度小结

waiter-service / customer-service

- 增加基于 Web 向 Zipkin 埋点功能

barista-service

- 增加基于 MQ 向 Zipkin 埋点功能

最终的成品

- 通过 Docker 运行整个 SpringBucks

“Talk is cheap, show me the code.”

*Chapter 16 / final-**

极客时间

扫码试看/订阅

《玩转 Spring 全家桶》