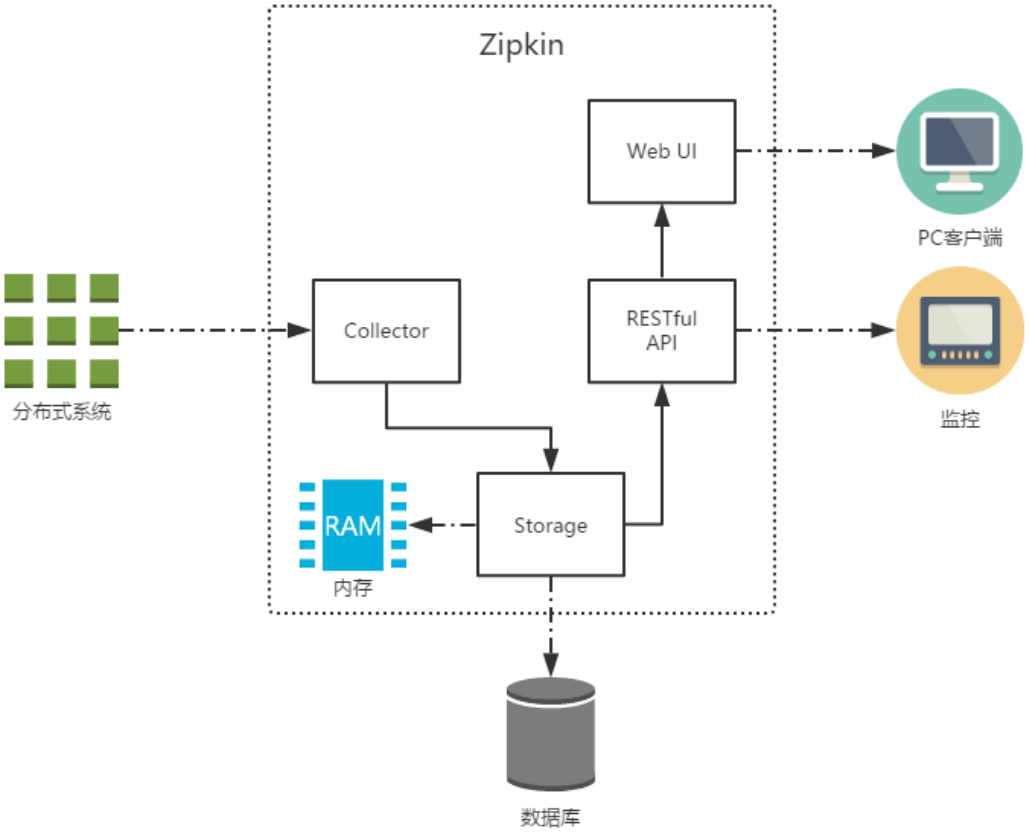
我们虽然已经能够利用ELK平台提供的收集、存储、搜索等强大功能，对跟踪信息的管理和使用已经变得非常便利。但是，在ELK平台中的数据分析维度缺少对请求链路中各阶段时间延迟的关注，很多时候我们追溯请求链路的一个原因是为了找出整个调用链路中出现延迟过高的瓶颈源，亦或是为了实现对分布式系统做延迟监控等与时间消耗相关的需求，这时候类似ELK这样的日志分析系统就显得有些乏力了。对于这样的问题，我们就可以引入Zipkin来得以轻松解决。

Zipkin简介

Zipkin是Twitter的一个开源项目，它基于Google Dapper实现。我们可以使用它来收集各个服务器上请求链路的跟踪数据，并通过它提供的REST API接口来辅助我们查询跟踪数据以实现对分布式系统的监控程序，从而及时地发现系统中出现的延迟升高问题并找出系统性能瓶颈的根源。除了面向开发的API接口之外，它也提供了方便的UI组件来帮助我们直观的搜索跟踪信息和分析请求链路明细，比如：可以查询某段时间内各用户请求的处理时间等。



上图展示了Zipkin的基础架构，它主要有4个核心组件构成：

- Collector：收集器组件，它主要用于处理从外部系统发送过来的跟踪信息，将这些信息转换为Zipkin内部处理的Span格式，以支持后续的存储、分析、展示等功能。
- Storage：存储组件，它主要对处理收集器接收到的跟踪信息，默认会将这些信息存储在内存中，我们也可以修改此存储策略，通过使用其他存储组件将跟踪信息存储到数据库中。

- RESTful API：API组件，它主要用来提供外部访问接口。比如给客户端展示跟踪信息，或是外接系统访问以实现监控等。
- Web UI：UI组件，基于API组件实现的上层应用。通过UI组件用户可以方便而有直观地查询和分析跟踪信息。

HTTP收集

在Spring Cloud Sleuth中对Zipkin的整合进行了自动化配置的封装，所以我们可以很轻松的引入和使用它，下面我们来详细介绍一下Sleuth与Zipkin的基础整合过程。主要分为两步：

第一步：搭建Zipkin Server

- 创建一个基础的Spring Boot应用，命名为zipkin-server，并在pom.xml中引入Zipkin Server的相关依赖，具体如下：

```
<dependencies>
  <dependency>
        <groupId>io.zipkin.java</groupId>
        <artifactId>zipkin-server</artifactId>
  </dependency>
  <dependency>
        <groupId>io.zipkin.java</groupId>
        <artifactId>zipkin-autoconfigure-ui</artifactId>
  </dependency>
</dependencies>
```

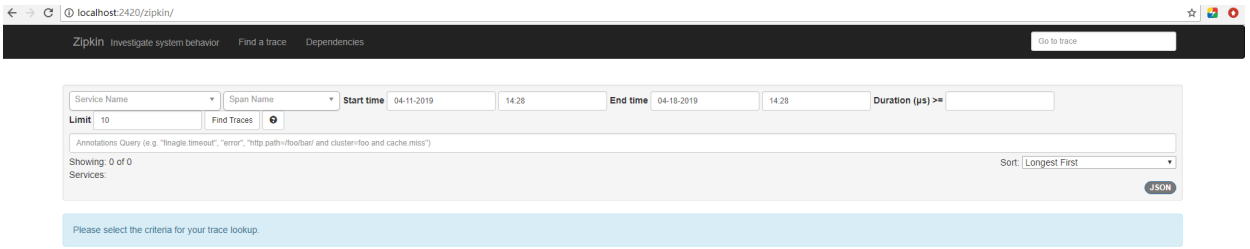
- 创建应用主类ZipkinApplication，使用@EnableZipkinServer注解来启动Zipkin Server，具体如下：

```
@EnableZipkinServer
@SpringBootApplication
public class ZipkinServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ZipkinServerApplication.class, args);
    }
}
```

- 在application.properties中做一些简单配置，比如：设置服务端口号为2420（客户端整合时候，自动化配置会连接9411端口，所以在服务端设置了端口为9411的话，客户端可以省去这个配置）。

```
spring.application.name=zipkin-server
server.port=2420
```

创建完上述工程之后，我们将其启动起来，并访问http://localhost:2420/，我们可以看到如下图所示的Zipkin管理页面：



第二步：为应用引入和配置Zipkin服务

在完成了Zipkin Server的搭建之后，我们还需要对应用做一些配置，以实现将跟踪信息输出到Zipkin Server。我们以之前实现的trace-a和trace-b为例，对它们做以下改造内容：

- 在trace-a和trace-b的pom.xml中引入spring-cloud-sleuth-zipkin依赖，具体如下所示。

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

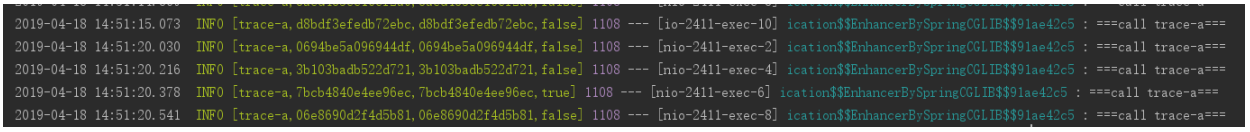
- 在trace-a和trace-b的application.properties中增加Zipkin Server的配置信息，具体如下所示（如果在zip-server应用中，我们将其端口设置为9411，并且均在本地调试的话，该参数也可以不配置，因为默认值就是http://localhost:9411）。

```
spring.zipkin.base-url=http://localhost:2420
```

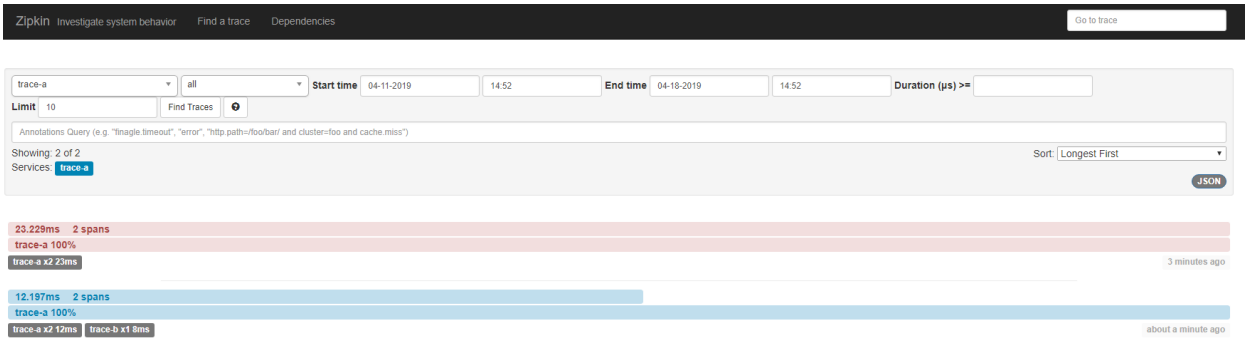
测试与分析

到这里我们已经完成了接入Zipkin Server的所有基本工作，我们可以继续将eureka-server、trace-a和trace-b启动起来，然后我们做一些测试实验，以对它的运行机制有一些初步的理解。

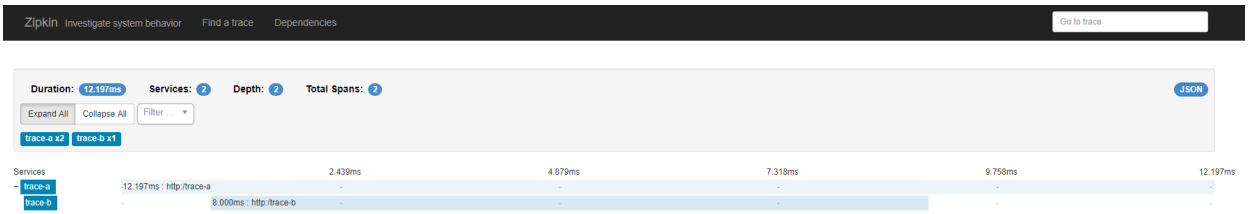
我们先来向trace-1的接口发送几个请求：http://localhost:9101/trace-1，当我们在日志中出现跟踪信息的最后一个值为true的时候，说明该跟踪信息会输出给Zipkin Server：



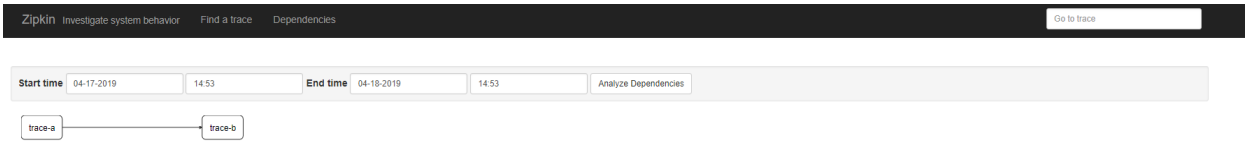
此时我们可以去Zipkin Server的管理页面中选择合适的查询条件后，点击Find Traces，就可以查询出刚才在日志中出现的跟踪信息了（也可以根据日志中的Trace ID，在页面的右上角输入框中来搜索），具体如下页面所示：



点击下方trace-a端点的跟踪信息，我们还可以得到Sleuth收集到的跟踪到详细信息，其中包括了我们关注的请求时间消耗等。



点击导航栏中的Dependencies菜单，我们还可以查看Zipkin Server根据跟踪信息分析生成的系统请求链路依赖关系图：



消息中间件收集

Spring Cloud Sleuth在整合Zipkin时，不仅实现了以HTTP的方式收集跟踪信息，还实现了通过消息中间件来对跟踪信息进行异步收集的封装。通过结合Spring Cloud Stream，我们可以非常轻松的让应用客户端将跟踪信息输出到消息中间件上，同时Zipkin服务端从消息中间件上异步地消费这些跟踪信息。

接下来，我们基于之前实现的trace-a和trace-b应用以及zipkin-server服务端做一些改造，以实现通过消息中间件来收集跟踪信息。改造的内容非常简单，只需要我们做项目依赖和配置文件做一些调整就能马上实现，下面我们分别对客户端和服务端的改造内容做详细说明：

第一步：修改客户端trace-a和trace-b

- 为了让trace-a和trace-b在产生跟踪信息之后，能够将抽样记录输出到消息中间件中，我们除了需要之前引入的spring-cloud-starter-sleuth依赖之外，还需要引入zipkin对Spring Cloud Stream的扩展依赖spring-cloud-sleuth-stream以及基于Spring Cloud Stream实现的消息中间件绑定器依赖，以使用RabbitMQ为例，我们可以加入如下依赖：

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

注意：pom.xml中需要删除用http方式实现收集服务跟踪的依赖：

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

- 在application.properties配置中去掉HTTP方式实现时使用的spring.zipkin.base-url参数，并根据实际部署情况，增加消息中间件的相关配置，如下是RabbitMQ的配置信息：

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=springcloud
spring.rabbitmq.password=123456
```

第二步：新建zipkin-server-stream服务端

因为采用消息中间件形式跟踪服务，这与zipkin-server工程不同，所以新建zipkin-server-stream服务从消息中间件中获取跟踪信息，我们只需要在pom.xml中引入针对消息中间件收集封装的服务端依赖spring-cloud-sleuth-zipkin-stream，同时为了支持具体使用的消息中间件，我们还需要引入针对消息中间件的绑定器实现，如果使用RabbitMQ，在依赖中增加如下内容：

```
<dependencies>
    <dependency>
        <groupId>io.zipkin.java</groupId>
        <artifactId>zipkin-autoconfigure-ui</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-sleuth-zipkin-stream</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
    </dependency>
</dependencies>
```

其中，spring-cloud-sleuth-zipkin-stream依赖是实现从消息中间件收集跟踪信息的核心封装，其中包含了用于整合消息中间件的核心依赖、zipkin服务端的核心依赖、以及一些其他通常会被使用的依赖（比如：用于扩展数据存储的依赖、用于支持测试的依赖等）。需要注意的是这个包里并没有引入zipkin的前端依赖zipkin-autoconfigure-ui，为了方便使用，我们在这里也引用了它。

- **创建应用主类**

注意加入@EnableZipkinStreamServer注解，这与zipkin-server工程注解不同。

```
@EnableZipkinStreamServer
@SpringBootApplication
public class ZipkinServerStreamApplication {
    public static void main(String[] args) {
        SpringApplication.run(ZipkinServerStreamApplication.class, args);
    }
}
```

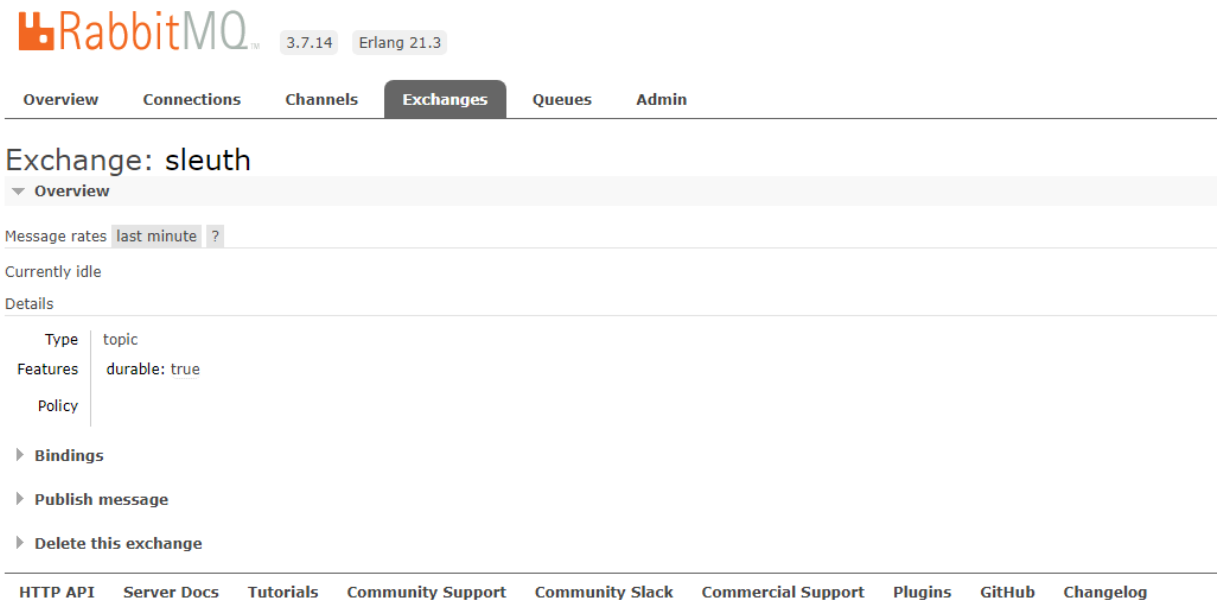
- **application.properties配置文件**

```
spring.application.name=zipkin-server-stream
server.port=2430

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=springcloud
spring.rabbitmq.password=123456
```

测试与分析

在完成了上述改造内容之后，我们继续将`eureka-server`、`zipkin-server-stream`、`trace-a`和`trace-b`都启动起来，同时确保RabbitMQ也处于运行状态。此时，我们可以在RabbitMQ的控制页面中看到一个名为`sleuth`的交换器，它就是zipkin的消息中间件收集器实现使用的默认主题。



最后，我们使用之前的验证方法，通过向`trace-a`的接口发送几个请求：`http://localhost:2411/trace-a`，当有被抽样收集的跟踪信息时（调试时我们可以设置`AlwaysSampler`抽样机制来让每个跟踪信息都被收集），我们可以在RabbitMQ的控制页面中发现有消息被发送到了`sleuth`交换器中，同时我们再到zipkin服务端的Web页面中也能够搜索到相应的跟踪信息，那么我们使用消息中间件来收集跟踪信息的任务到这里就完成了。