

32 Zipkin 实践

更新时间：2019-07-30 09:30:32



“天才免不了有障碍，因为障碍会创造天才。”

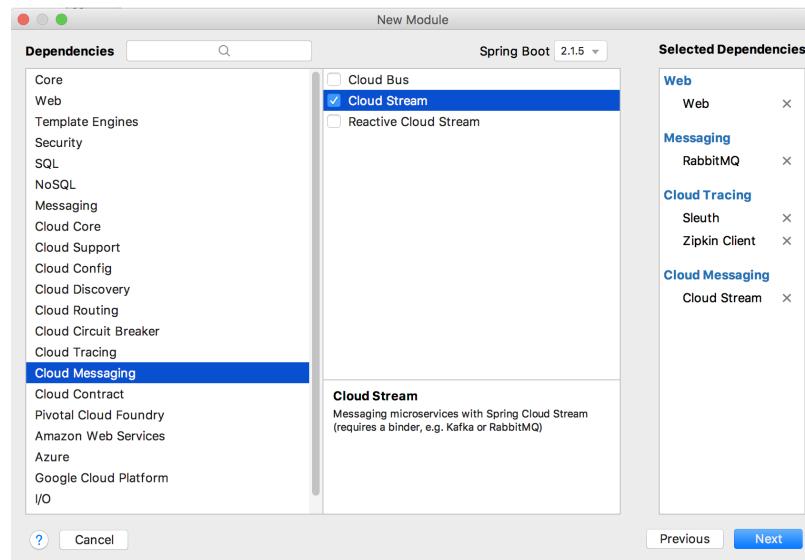
——罗曼·罗兰

Zipkin 实践

上篇文章带领大家了解了 Zipkin 的基本概念，以及 Zipkin 中环境的搭建，现在是万事俱备只欠东风了，本文就来和大家聊一下这搭好的环境要如何使用。

创建 provider

要演示链路追踪，我们需要提前准备好两个微服务，两个服务之间互相调用，然后我们来观察链路追踪情况。因此需要首先创建一个名为 zipkin 的 maven 父工程，然后在 zipkin 项目中创建一个名为 provider 的 module，创建时分别添加 Web、Sleuth、RabbitMQ、Spring Cloud Stream 以及 Zipkin 依赖，如下：



工程创建完成后，pom.xml 文件核心依赖如下：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
<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>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
</dependency>
```

创建完成后，我们首先在 application.properties 中添加 stream、zipkin 以及 rabbitmq 相关的配置：

```
spring.application.name=provider
spring.sleuth.web.client.enabled=true
spring.sleuth.sampler.probability=1
spring.zipkin.base-url=http://localhost:9411
spring.zipkin.enabled=true
spring.zipkin.sender.type=rabbit
spring.rabbitmq.addresses=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
server.port=8080
```

这些配置有一些是大家面熟的，也有一些可能是第一次接触到的，主要是 sleuth 和 zipkin 的配置可能第一次接触，我这里就主要来介绍一下这两个相关的配置的含义吧：

- spring.sleuth.web.client.enabled 表示开启链路追踪；
- spring.sleuth.sampler.probability 表示追踪信息导出到 zipkin 的比例，这里默认是 0.1，即 10% 的追踪信息导出到 zipkin，我们这里将之配置为 1；
- spring.zipkin.base-url 表示指定 zipkin server 的地址；
- spring.zipkin.base-url 表示开启 zipkin；
- spring.zipkin.sender.type 表示设置追踪信息的发送类型。

配置完成后，我们再添加一个 HelloController，提供一个测试接口，如下：

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello(String name) {
        return "hello " + name + " !";
    }
}
```

接下来我们就可以启动 provider 了。provider 启动成功之后，先放着，我们再来看 consumer 的创建。

创建 consumer

consumer 的创建和 provider 的步骤基本一致，需要添加的依赖以及 application.properties 中的配置都是一样的（除了项目启动端口不一致），因此这个步骤我就不再赘述，当 application.properties 配置完成后，我们在 consumer 中首先配置一个 RestTemplate 的 Bean，如下：

```
@SpringBootApplication
public class ConsumerApplication {

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

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

注意，这里的 RestTemplate 的实例我没有开启负载均衡功能，所以这里主要是给大家展示链路追踪的用法，没有引入服务注册中心，因此也没有引入负载均衡的注解。最后再添加一个 Controller 去消费 provider 中提供的接口，如下：

```
@RestController
public class UseHelloController {
    @Autowired
    RestTemplate restTemplate;
    @GetMapping("/sayhello")
    public void hello() {
        String s = restTemplate.getForObject("http://localhost:8080/hello?name={1}", String.class, "javaboy");
        System.out.println(s);
    }
}
```

这段代码也很简单，配置完成之后，我们再来启动 consumer 工程。consumer 启动成功之后，我们先尝试在浏览器中发送请求调用 /sayhello 接口：<http://localhost:8081/sayhello>。

查看链路追踪

当 consumer 中的请求发送完成之后，接下来我们刷新 zipkin 的调用页面，发现已经有了一条调用记录，如下：



The screenshot shows the Zipkin search interface. At the top, there are dropdowns for '服务名' (Service Name) set to 'all', 'Span名称' (Span Name) set to 'all', '远程服务名' (Remote Service Name) set to 'all', and a '时间' (Time) dropdown set to '1小时' (1 hour). Below these are search filters: '根据Annotation查询' (Search by Annotation), '持续时间 (μs) >=' (Duration (μs) >=), '数量' (Count), and '排序' (Sort). A search bar with placeholder 'For example: http.path=/foo/bar/ and cluster=foo and cache.miss' and a '搜索' (Search) button are also present. The results table shows '171.197ms 3 spans' and 'provider x 3 171.197ms'. The table has a header row with columns 'provider' and 'span'. The first row shows 'provider' and '171.197ms : get /sayhello'. The second row shows 'provider' and '34.239ms : get /hello'. The third row shows 'provider' and '68.479ms : get /hello'. The fourth row shows 'provider' and '102.718ms : get /hello'. The fifth row shows 'provider' and '136.958ms : get /hello'. The sixth row shows 'provider' and '171.197ms : get /hello'. The bottom right of the table area says 'about a minute ago'.

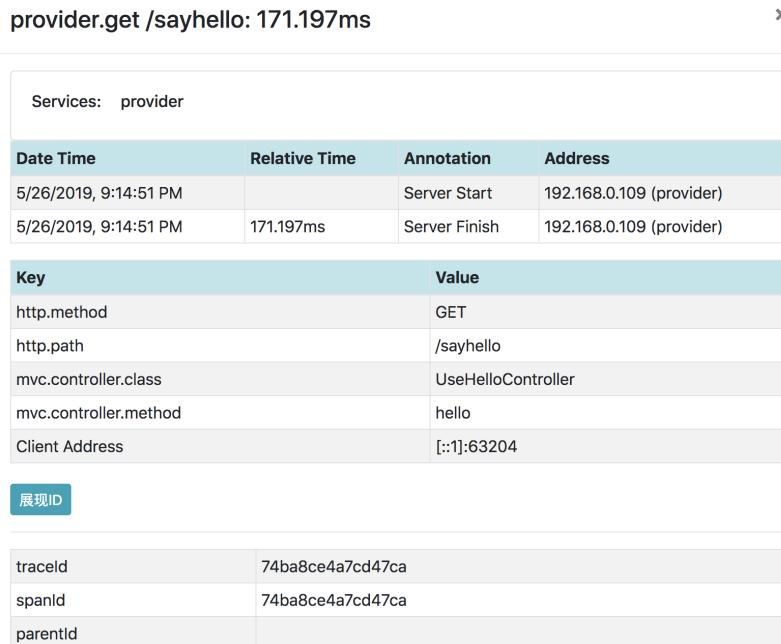
如果服务比较多，可以使用该页面提供的搜索功能进行搜索，就能快速定位到自己需要的服务。

点开这条调用记录，如下：



The screenshot shows the Zipkin trace details page for the trace 'provider.get /sayhello: 171.197ms'. The top bar shows the trace ID 'provider x 3 171.197ms'. Below the bar, there are buttons for '全部展开' (Expand All) and '全部折叠' (Collapse All). The main area shows a timeline of events: 'provider' at 171.197ms, 'sayhello' at 34.239ms, 'hello' at 68.479ms, 'hello' at 102.718ms, 'hello' at 136.958ms, and 'provider' at 171.197ms. Each event is accompanied by its duration and the method it represents.

可以看到整个调用链以及请求分别在 `/sayhello` 和 `/hello` 接口上所花费的时间。点击某一个接口，还可以看到具体的数据：



The screenshot shows the detailed trace page for the trace 'provider.get /sayhello: 171.197ms'. The top bar shows the trace ID 'provider.get /sayhello: 171.197ms'. The main content is divided into several sections: 'Services: provider', a table of service metrics, a table of annotations, and a table of client details. The 'Services: provider' section shows the provider service. The 'Annotations' table shows the following data:

Date	Time	Relative Time	Annotation	Address
5/26/2019, 9:14:51 PM			Server Start	192.168.0.109 (provider)
5/26/2019, 9:14:51 PM		171.197ms	Server Finish	192.168.0.109 (provider)

The 'Annotations' table shows the following data:

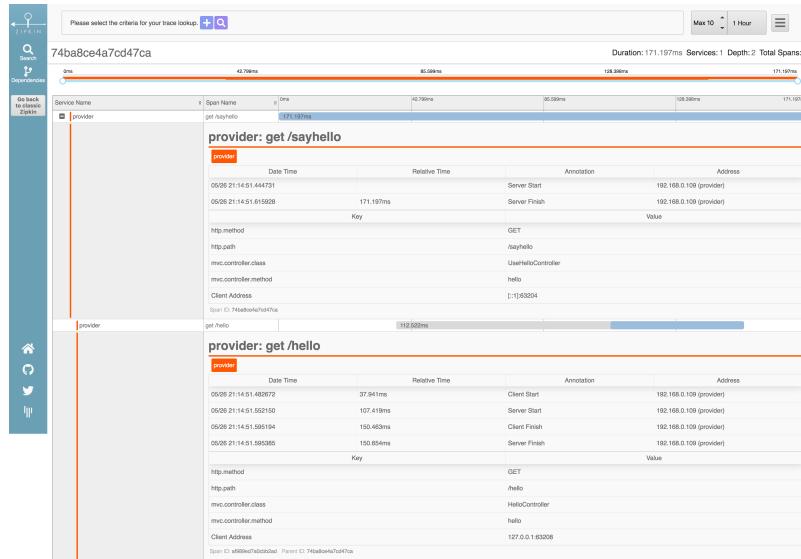
Key	Value
http.method	GET
http.path	/sayhello
mvc.controller.class	UseHelloController
mvc.controller.method	hello
Client Address	[:1]:63204

At the bottom, there is a '展现ID' (Show ID) button and a table of trace identifiers:

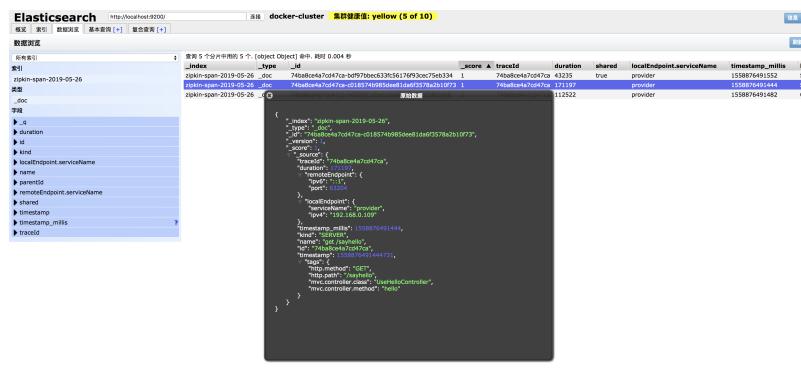
traceld	74ba8ce4a7cd47ca
spanld	74ba8ce4a7cd47ca
parentld	

这里可以看到每一个步骤的详细信息，包括请求方法、对应的 method 、相关的 Controller 以及客户端的地址等。由于这里是 `/sayhello` 接口，因此没有 parentld，下一个请求开始就有 parentld 了。

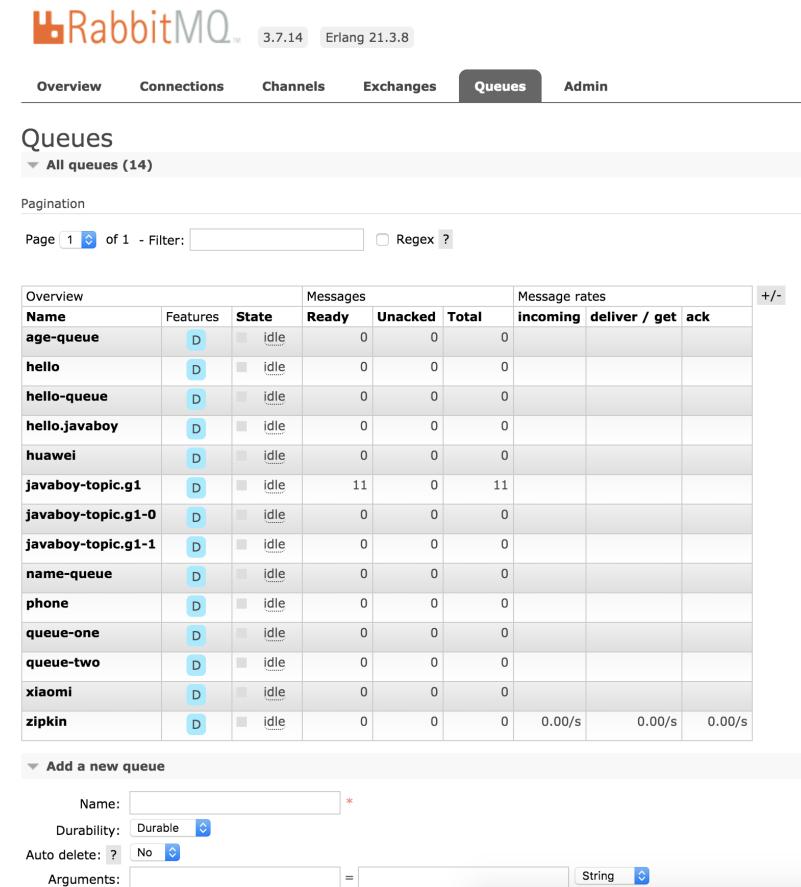
也可以点击右上角的 Try Lens UI 按钮，换一个 UI 风格：



然后我们打开 Elasticsearch-head，可以看到数据已经存储到 Elasticsearch-head 上了，如下：



最后我们再打开 RabbitMQ 的管理面板，也可以看到有一个名为 zipkin 的队列，如下：



小结

经过上面的步骤之后，一个分布式的服务链路追踪系统就算完成了，我们平时只需要通过 zipkin 的 WebUI 界面就能快速查看每一个请求的状况，包括在每一个微服务上花费的时间，就能快速定位出性能瓶颈。

[31 Zipkin 入门介绍](#)

[33 Spring Boot Admin 介绍](#)

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示



目前暂无任何讨论