# 一站式微服务架构Spring Cloud

主讲：Cat 老师

北京动力节点教育科技有限公司  
2019 • 北京

动力节点•版权所有•禁止传播

# 快速回顾

1. 分布式与微服务架构的理论梳理；
2. 什么是Spring Cloud？
3. Spring Cloud的整体架构 （与Dubbo比较）
4. 服务消费者Controller直连调用服务提供者Controller（http协议的restful）
5. Spring Cloud的注册中心Eureka
6. Spring Cloud Eureka与Zookeeper比较
7. Spring Cloud Eureka高可用集群
8. Spring Cloud Eureka自我保护机制
9. Spring Cloud Ribbon负载均衡
10. Spring Cloud Feign声明式服务调用（与Dubbo接口层比较）
11. Spring Cloud Hystrix 服务熔断降级、服务限流
12. Spring Cloud Hystrix DashBoard仪表盘监控
13. Spring Cloud Hystrix Turbine聚合监控
14. Spring Cloud Zuul网关（路由、过滤、异常、降级）
15. Spring Cloud Config配置中心（用途、使用、加解密）
16. Spring Cloud Config 配置中心（自动刷新、高可用、安全认证）

# 本次课主题大纲

1. Spring Cloud Sleuth分布式链路跟踪
2. Spring Cloud Sleuth整合Zipkin
3. Zipkin Server数据持久化

# Spring Cloud Sleuth分布式链路跟踪

## 2.1 分布式链路跟踪概述

前面我们接触过几种微服务的监控方式，比如Spring Boot Actuator监控微服务，Spring Boot Admin监控微服务，Hystrix Dashboard监控Hystrix服务，Hystrix Turbine聚合多个Hystrix服务的监控信息等，接下来我们要讨论的是微服务的“跟踪" ；

对于一个大型的几十个、几百个微服务构成的微服务架构系统，通常会遇到下面一些问题，比如：

如何串联整个调用链路，快速定位问题？

如何理清各个微服务之间的依赖关系？

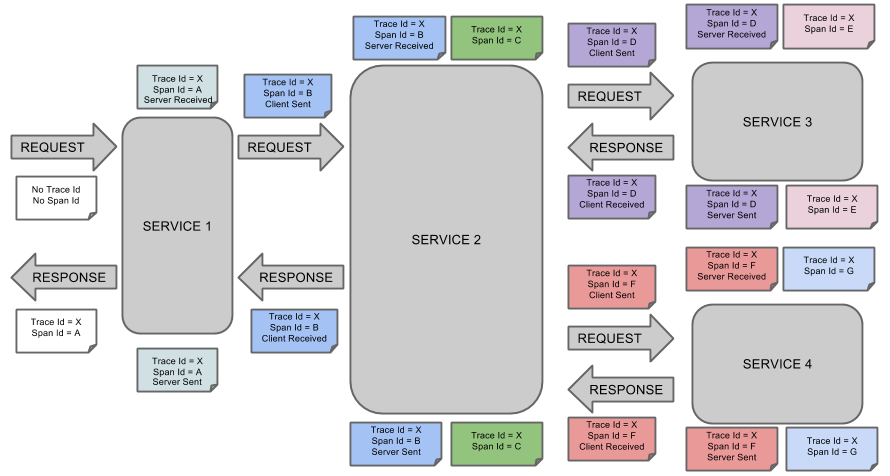
如何进行各个微服务接口的性能分折？

如何跟踪整个业务流程的调用处理顺序？

Spring Cloud Sleuth [sluːθ]为 spring Cloud提供了分布式跟踪的解决方案，它大量借用了Google Dapper、Twitter Zipkin和 Apache HTrace的设计，帮我们解决像上面提到的问题；

Spring Cloud Sleuth可以追踪10种类型的组件：async、Hystrix，messaging，websocket，rxjava，scheduling，web（Spring MVC Controller，Servlet），webclient（Spring RestTemplate）、Feign、Zuul；

我们通过一张图来了解一个简单的微服务的调用链路：



图片出自：<https://cloud.spring.io/spring-cloud-static/Greenwich.SR3/single/spring-cloud.html#_spring_cloud_sleuth>

这里面涉及到Spring Cloud Sleuth的一些术语

**span（跨度）**：基本工作单元。span用一个64位的id唯一标识。除ID外，span还包含其他数据，例如描述、时间戳事件、键值对的注解（标签）， spanID、span父 ID等。span被启动和停止时，记录了时间信息。初始化 span被称为"rootspan"，该 span的 id和 trace的 ID相等；

**trace（跟踪）**：一组共享"rootspan"的 span组成的树状结构称为 traceo trace也用一个64位的 ID唯一标识， trace中的所有 span都共享该 trace的 ID；

**annotation（标注）**： annotation用来记录事件的存在，其中，核心annotation用来定义请求的开始和结束。

**CS（Client sent客户端发送）**：客户端发起一个请求，该annotation描述了span的开始；

**SR（server Received服务器端接收）**：服务器端获得请求并准备处理它。如果用 SR减去 CS时间戳，就能得到网络延迟；

**SS（server sent服务器端发送）**：该annotation表明完成请求处理（当响应发回客户端时）。如果用 SS减去 SR时间戳，就能得到服务器端处理请求所需的时间；

**CR（Client Received客户端接收）**： span结束的标识。客户端成功接收到服务器端的响应。如果 CR减去 CS时间戳，就能得到从客户端发送请求到服务器响应的所需的时间；

## 2.2 Spring Cloud Sleuth

Spring Cloud Sleuth对于分布式链路的跟踪仅仅是生成一些数据，这些数据不便于人类阅读，所以我们一般把这种跟踪数据上传给Zipkin Server，由Zipkin通过UI页面统一进行数据的展示；

## 2.3 整合Zipkin实现分布式链路跟踪

### 认识一下Zipkin

Zipkin是Twitter开源的分布式实时数据跟踪系统（Distributed Tracking System），基于Google Dapper的论文设计而成，Google开源了 Dapper链路追踪组件，并在2010年发表了论文《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure》，这篇文章是业内实现链路追踪的标杆和理论基础，具有非常大的参考价值。

Zipkin它的主要功能是收集系统的时序数据，从而追踪微服务架构的系统延时等问题，从而达到链路调用监控跟踪作用，另外Zipkin还提供了一个非常友好的UI界面，来帮助分析追踪数据，Zipkin官网地址：<http://zipkin.io>

分布式跟踪系统有一些成熟的开源产品，比如：韩国Naver的Pinpoint，Apache的HTrace，阿里的鹰眼EagleEye，京东的Hydra等，这些产品我们也把他们叫做APM（应用性能管理）工具；

### 搭建Zipkin Server

1. 创建一个SpringBoot项目，用于搭建Zipkin Server服务端；
2. 添加依赖：

*<!-- zipkin-autoconfigure-ui -->*<**dependency**>  
 <**groupId**>io.zipkin.java</**groupId**>  
 <**artifactId**>zipkin-autoconfigure-ui</**artifactId**>  
 <**version**>2.12.3</**version**>  
</**dependency**>  
  
*<!-- zipkin-server -->*<**dependency**>  
 <**groupId**>io.zipkin.java</**groupId**>  
 <**artifactId**>zipkin-server</**artifactId**>  
 <**version**>2.12.3</**version**>  
</**dependency**>

注意：这个里面有版本的兼容性问题，当前我们的spring cloud G SR3版本使用2.12.3便可以正常使用，如果使用zipkin-server的最新的版本，可能会启动失败；

2、配置文件：

*#zipkin启动报错无法访问的解决方法*

**management.metrics.web.server.autoTimeRequests**=**false**

3、在启动类上加入注解:@EnableZipkinServer：

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

4、然后启动zipkin server服务，访问<http://localhost:9410> （默认启动了Undertow服务的9410端口）



能看到上面这个页面，zipkin server服务搭建OK；

服务名：就是微服务配置文件中的application name；

Span名称：跨度；

时间段 ：现在查询的时间段；

根据Annotation查询：根据标注查询，用于自定义查询条件；

持续时间：一次调用链的持续时间；

数量：一页数量；

排序：排序规则；

目前我们还查询不到数据，我们需要把微服务和sleuth整合，并把sleuth记录的数据上传到zipkin server，此时我们才能在页面上看到数据；

### Sleuth微服务整合Zipkin

将我们的微服务的跟踪数据上传到Zipkin Server中；

注意，每个微服务都需要和zipkin整合，这样便于把我们的各个微服务的调用链路信息上传到Zipkin Server中；

在各个微服务里面进行操作：

1、添加依赖

*<!-- spring-cloud-starter-sleuth -->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-sleuth</**artifactId**>  
</**dependency**>

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

2、配置文件：

*#指定Zipkin server地址***spring.zipkin.base-url**=**http://localhost:9410**

*#发送跟踪数据到zipkin的类型web（http）***spring.zipkin.sender.type**=*web**#request采样的数量 默认是0.1 也即是10%，即采样10%的请求数据；  
#因为在分布式系统中，数据量可能会非常大，因此采样非常重要我们示例数据少最好配置为1全采样，100%的采集会稍微影响一点性能***spring.sleuth.sampler.probability**=**1.0**

## Zipkin Server 数据持久化

前面我们已经把分布式链路调用信息上传到 zipkin server 上，通过zipkin server 的ui界面我们能看到调用链路信息，但是这些上传了的跟踪信息没有持久化保存，当zipkin重启后分布式链路数据就全部清空了，因为zipkin server 默认数据是存储在内存中的，所以为了后续一直都能查看调用链路信息，最好是将这些信息持久化保存；

### 使用Elastic Search 做数据持久化

我们还没有正式学习过Elastic Search课程，我们先试用一下Elastic Search 来做链路跟踪数据的持久化，这个数据的持久化还可以用数据库、ELK等来做；

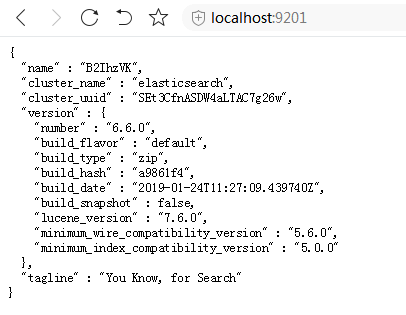
Elastic Search是一个分布式可扩展的实时搜索和分析引擎,一个建立在全文搜索引擎 Apache Lucene基础上的搜索引擎，我们暂时不想象那么复杂，我们可以把它当作一个数据库来对待，就是用来存储数据的；

GitHub：<https://github.com/elastic/elasticsearch>

下载：<https://www.elastic.co/cn/downloads/elasticsearch>

解压下载后的压缩包即完成安装，切换到bin目录，使用双击elasticsearch.bat脚本启动；

启动后访问：<http://localhost:9201，如果能返回json信息则表示安装启动OK；>



zipkin 与 Elastic Search整合

在zipkin Server中添加依赖:

1、添加依赖

<dependency>  
 <groupId>io.zipkin.java</groupId>  
 <artifactId>zipkin-autoconfigure-storage-elasticsearch-http</artifactId>  
 <version>2.8.4</version>  
</dependency>

配置文件：

**zipkin.storage.type**=**elasticsearch  
zipkin.storage.elasticsearch.cluster**=**elasticsearch  
zipkin.storage.elasticsearch.hosts**=**http://localhost:9201  
zipkin.storage.elasticsearch.index**=**zipkin**

至此 zipkin server上的跟踪数据便存储在了Elasticsearch中，当zipkin server 重启或宕机，历史数据依然不会丢失；