**史上最简单的SpringCloud教程 | 第九篇: 服务链路追踪(Spring Cloud Sleuth)**

2017年04月13日 21:03:29

阅读数：139393

转载请标明出处：   
<http://blog.csdn.net/forezp/article/details/70162074>   
本文出自[方志朋的博客](http://blog.csdn.net/forezp)

* 最新版本：
* [史上最简单的SpringCloud教程 | 第九篇: 服务链路追踪(Spring Cloud Sleuth)(Finchley版本)](http://blog.csdn.net/forezp/article/details/81041078)

这篇文章主要讲述服务追踪组件zipkin，Spring Cloud Sleuth集成了zipkin组件。

**一、简介**

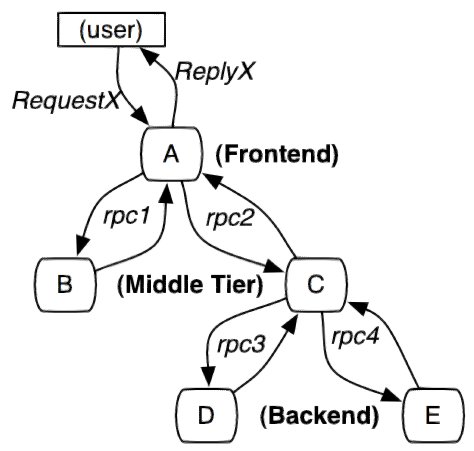
Add sleuth to the classpath of a Spring Boot application (see below for Maven and Gradle examples), and you will see the correlation data being collected in logs, as long as you are logging requests.

—— [摘自官网](https://github.com/spring-cloud/spring-cloud-sleuth)

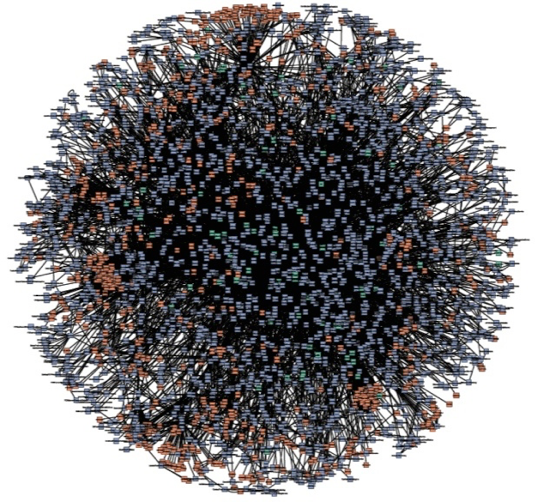
Spring Cloud Sleuth 主要功能就是在分布式系统中提供追踪解决方案，并且兼容支持了 zipkin，你只需要在pom文件中引入相应的依赖即可。

**二、服务追踪分析**

微服务架构上通过业务来划分服务的，通过REST调用，对外暴露的一个接口，可能需要很多个服务协同才能完成这个接口功能，如果链路上任何一个服务出现问题或者网络超时，都会形成导致接口调用失败。随着业务的不断扩张，服务之间互相调用会越来越复杂。



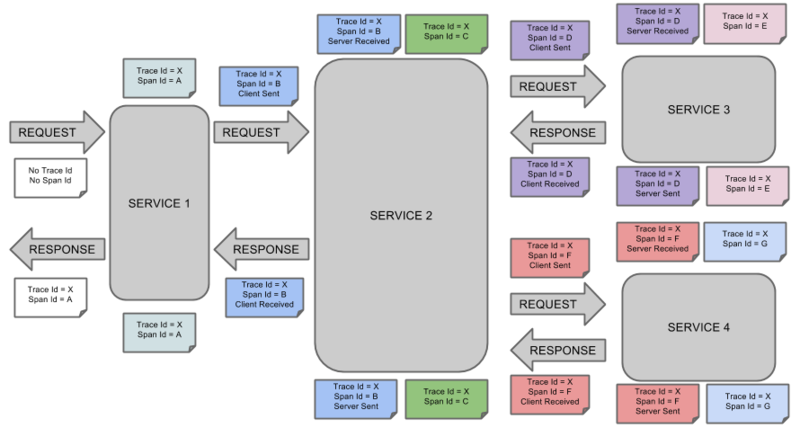
随着服务的越来越多，对调用链的分析会越来越复杂。它们之间的调用关系也许如下：



**三、术语**

* Span：基本工作单元，例如，在一个新建的span中发送一个RPC等同于发送一个回应请求给RPC，span通过一个64位ID唯一标识，trace以另一个64位ID表示，span还有其他数据信息，比如摘要、时间戳事件、关键值注释(tags)、span的ID、以及进度ID(通常是IP地址)   
  span在不断的启动和停止，同时记录了时间信息，当你创建了一个span，你必须在未来的某个时刻停止它。
* Trace：一系列spans组成的一个树状结构，例如，如果你正在跑一个分布式大数据工程，你可能需要创建一个trace。
* Annotation：用来及时记录一个事件的存在，一些核心annotations用来定义一个请求的开始和结束   
  + cs - Client Sent -客户端发起一个请求，这个annotion描述了这个span的开始
  + sr - Server Received -服务端获得请求并准备开始处理它，如果将其sr减去cs时间戳便可得到网络延迟
  + ss - Server Sent -注解表明请求处理的完成(当请求返回客户端)，如果ss减去sr时间戳便可得到服务端需要的处理请求时间
  + cr - Client Received -表明span的结束，客户端成功接收到服务端的回复，如果cr减去cs时间戳便可得到客户端从服务端获取回复的所有所需时间   
    将Span和Trace在一个系统中使用Zipkin注解的过程图形化：

将Span和Trace在一个系统中使用Zipkin注解的过程图形化：



**四、构建工程**

基本知识讲解完毕，下面我们来实战，本文的案例主要有三个工程组成:一个server-zipkin,它的主要作用使用ZipkinServer 的功能，收集调用数据，并展示；一个service-hi,对外暴露hi接口；一个service-miya,对外暴露miya接口；这两个service可以相互调用；并且只有调用了，server-zipkin才会收集数据的，这就是为什么叫服务追踪了。

**4.1 构建server-zipkin**

建一个spring-boot工程取名为server-zipkin，在其pom引入依赖：

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.zipkin.java</groupId>

<artifactId>zipkin-server</artifactId>

</dependency>

<dependency>

<groupId>io.zipkin.java</groupId>

<artifactId>zipkin-autoconfigure-ui</artifactId>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Camden.SR6</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40

在其程序入口类, 加上注解@EnableZipkinServer，开启ZipkinServer的功能：

@SpringBootApplication

@EnableZipkinServer

public class ServerZipkinApplication {

public static void main(String[] args) {

SpringApplication.run(ServerZipkinApplication.class, args);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

在配置文件application.yml指定服务端口为：

server.port=9411

* 1

**4.2 创建service-hi**

在其pom引入起步依赖spring-cloud-starter-zipkin，代码如下：

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--compile('org.springframework.cloud:spring-cloud-starter-zipkin')-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zipkin</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33

在其配置文件application.yml指定zipkin server的地址，头通过配置“spring.zipkin.base-url”指定：

server.port=8988

spring.zipkin.base-url=http://localhost:9411

spring.application.name=service-hi

* 1
* 2
* 3
* 4
* 5

通过引入spring-cloud-starter-zipkin依赖和设置spring.zipkin.base-url就可以了。

对外暴露接口：

@SpringBootApplication

@RestController

public class ServiceHiApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceHiApplication.class, args);

}

private static final Logger LOG = Logger.getLogger(ServiceHiApplication.class.getName());

@Autowired

private RestTemplate restTemplate;

@Bean

public RestTemplate getRestTemplate(){

return new RestTemplate();

}

@RequestMapping("/hi")

public String callHome(){

LOG.log(Level.INFO, "calling trace service-hi ");

return restTemplate.getForObject("http://localhost:8989/miya", String.class);

}

@RequestMapping("/info")

public String info(){

LOG.log(Level.INFO, "calling trace service-hi ");

return "i'm service-hi";

}

@Bean

public AlwaysSampler defaultSampler(){

return new AlwaysSampler();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40

**4.3 创建service-miya**

创建过程痛service-hi，引入相同的依赖，配置下spring.zipkin.base-url。

对外暴露接口：

@SpringBootApplication

@RestController

public class ServiceMiyaApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceMiyaApplication.class, args);

}

private static final Logger LOG = Logger.getLogger(ServiceMiyaApplication.class.getName());

@RequestMapping("/hi")

public String home(){

LOG.log(Level.INFO, "hi is being called");

return "hi i'm miya!";

}

@RequestMapping("/miya")

public String info(){

LOG.log(Level.INFO, "info is being called");

return restTemplate.getForObject("http://localhost:8988/info",String.class);

}

@Autowired

private RestTemplate restTemplate;

@Bean

public RestTemplate getRestTemplate(){

return new RestTemplate();

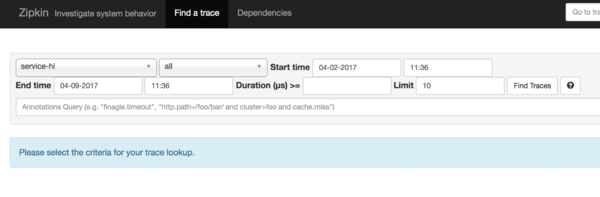
}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33

**4.4 启动工程，演示追踪**

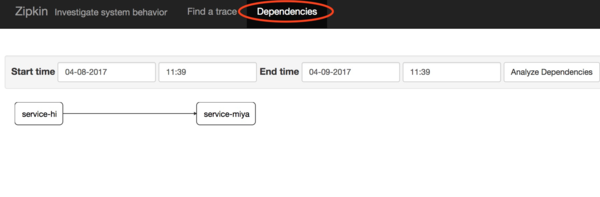
依次启动上面的三个工程，打开浏览器访问：<http://localhost:9411/>，会出现以下界面：



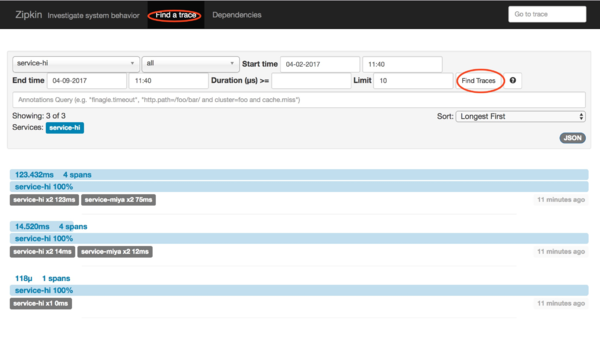
访问：<http://localhost:8989/miya>，浏览器出现：

i’m service-hi

再打开<http://localhost:9411/>的界面，点击Dependencies,可以发现服务的依赖关系：



点击find traces,可以看到具体服务相互调用的数据：



本文源码下载：   
<https://github.com/forezp/SpringCloudLearning/tree/master/chapter9>

**五、参考资料**

[spring-cloud-sleuth](https://github.com/spring-cloud/spring-cloud-sleuth)

[利用Zipkin对Spring Cloud应用进行服务追踪分析](https://yq.aliyun.com/articles/60165)

[Spring Cloud Sleuth使用简介](http://blog.csdn.net/u010257992/article/details/52474639)

**优秀文章推荐：**

* [史上最简单的 SpringCloud 教程 | 终章](http://blog.csdn.net/forezp/article/details/70148833)
* [史上最简单的 SpringCloud 教程 | 第一篇: 服务的注册与发现（Eureka）](http://blog.csdn.net/forezp/article/details/69696915)
* [史上最简单的SpringCloud教程 | 第七篇: 高可用的分布式配置中心(Spring Cloud Config)](http://blog.csdn.net/forezp/article/details/70037513)

版权声明：本文为博主原创文章，欢迎转载，转载请注明作者、原文超链接 ，博主地址：http://blog.csdn.net/forezp。