实操 | 基于 SOFATracer + Zipkin 实现分布式链路跟踪

原创：雪连 金融级分布式架构 8月31日

> SOFA 中间件是蚂蚁金服自主研发的金融级分布式中间件，包含了构建金融级云原生架构所需的各个组件，包括微服务研发框架，RPC 框架，服务注册中心，分布式定时任务，限流/熔断框架，动态配置推送，分布式链路追踪，Metrics 监控度量，分布式高可用消息队列，分布式事务框架，分布式数据库代理层等组件，是在金融场景里锤炼出来的最佳实践。
>
> SOFATracer 是一个用于分布式系统调用跟踪的组件，通过统一的 TraceId 将调用链路中的各种网络调用情况以日志的方式记录下来，以达到透视化网络调用的目的，这些链路数据可用于故障的快速发现，服务治理等。
>
> SOFATracer 的 Github 的地址是：
> https://github.com/alipay/sofa-tracer

前言

SOFATracer 是一个用于分布式系统调用跟踪的组件，通过统一的 TraceId 将调用链路中的各种网络调用情况以日志的方式记录下来，以达到透视化网络调用的目的，这些链路数据可用于故障的快速发现，服务治理等。

为了解决在实施大规模微服务架构时的链路跟踪问题，SOFATracer 基于 OpenTracing 规范并扩展其能力，包括基于 Disruptor 高性能无锁循环队列的异步落地磁盘的日志打印能力，自定义日志格式，日志自清除和滚动能力，基于 SLF4J MDC 的扩展能力，统一的配置能力等。

同时 SOFATracer 也对接了开源生态，可以选择将 SOFATracer 的链路数据对接到 Zipkin 等开源产品做展示。

本文主要介绍 SOFATracer 在 SOFABoot 应用中的实践，通过一个应用利用 HttpClient 调用另一个应用 Spring MVC 提供的 RESTful 服务，然后再这个 Spring MVC 应用中通过指定 SOFARPC 服务端地址的方式调用 SOFARPC 提供的服务，最后在这个 SOFARPC 服务中访问 DB，以此来完成基于 SOFATracer + Zipkin 实现分布式链路跟踪。

一、背景

在当下的技术架构实施中，统一采用面向服务的分布式架构，通过服务来支撑起一个个应用，而部署在应用中的各种服务通常都是用复杂大规模分布式集群来实现的。同时，这些应用又构建在不同的软件模块上，这些软件模块，有可能是由不同的团队开发，可能使用不同的编程语言来实现、有可能部署了几千台服务器。

因此，就需要一些可以帮助理解各个应用的线上调用行为，并可以分析远程调用性能的组件。为了能够分析应用的线上调用行为以及调用性能，蚂蚁金服基于 OpenTracing 规范提供了分布式链路跟踪 SOFATracer 的解决方案。

二、基本原理

SOFATracer 通过全局唯一的 TraceId 作为一次链路调用的唯一标识，在特定的一次链路调用中，这个 TraceId 会一直保持不变并传递给被调用的系统。有了 TraceId 我们能够唯一的标识和确定这次调用链路，但是却无法标识出这个链路的调用层次是如何的？那么如何解决这个问题呢，那就是 SpanId，SpanId 标示的一次链路调用所在的层次，SpanId 的不同即标示这次链路调用处在不同的层次上。

为了让大家在下文更好的理解链路调用的过程，先给大家介绍下我们 TraceId 和 SpanId 的生成原则。

2.1 TraceId

SOFATracer 通过 TraceId 来将一个请求在各个服务器上的调用日志串联起来，TraceId 一般由接收请求经过的第一个服务器产生，产生规则是： 服务器 IP + 产生 ID 时候的时间 + 自增序列 + 当前进程号。比如：

```
0ad1348f1403169275002100356696
```

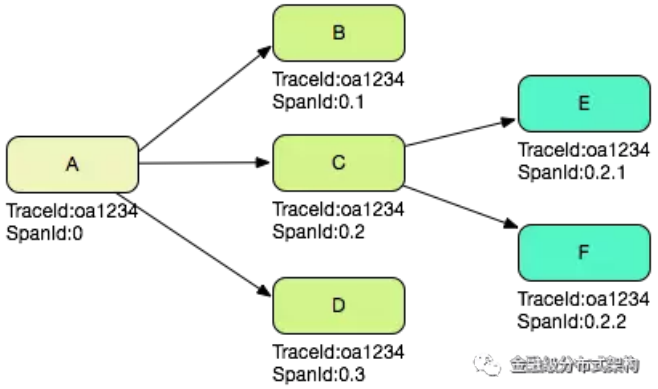
前 8 位 `0ad1348f` 即产生 `TraceId` 的机器的 IP，这是一个十六进制的数字，每两位代表 IP 中的一段，我们把这个数字，按每两位转成 10 进制即可得到常见的 IP 地址表示方式 `10.209.52.143`，大家也可以根据这个规律来查找到请求经过的第一个服务器。 后面的 13 位 `1403169275002` 是产生 TraceId 的时间。 之后的 4 位 `1003` 是一个自增的序列，从 1000 涨到 9000，到达 9000 后回到 1000 再开始往上涨。 最后的 5 位 `56696` 是当前的进程 ID，为了防止单机多进程出现 TraceId 冲突的情况，所以在 TraceId 末尾添加了当前的进程 ID。

IP 地址（8 位）	TraceId 的生成时间（13 位）	自增序列（4 位）	进程 ID（可变）
0ad1348f	1403169275002	1003	56696

2.2 SpanId

SOFATracer 中的 SpanId 代表本次调用在整个调用链路树中的层次位置，假设一个 Web 系统 A 接收了一次用户请求，那么在这个系统的 SOFATracer MVC 日志中，记录下的 SpanId 是 0，代表是整个调用的根节点，如果 A 系统处理这次请求，需要通过 RPC 依次调用 B，C，D 三个系统，那么在 A 系统的 SOFATracer RPC 客户端日志中，SpanId 分别是 0.1，0.2 和 0.3，在 B，C，D 三个系统的 SOFATracer RPC 服务端日志中，SpanId 也分别是 0.1，0.2 和 0.3；如果 C 系统在处理请求的时候又调用了 E，F 两个系统，那么 C 系统中对应的 SOFATracer RPC 客户端日志是 0.2.1 和 0.2.2，E，F 两个系统对应的 SOFATracer RPC 服务端日志也是 0.2.1 和 0.2.2。根据上面的描述，我们可以知道，如果把一次调用中所有的 SpanId 收集起来，可以组成一棵完整的链路树。

我们假设一次分布式调用中产生的 TraceId 是 0a1234（实际不会这么短），那么根据上文 SpanId 的产生过程，有下图：



SOFATracer 通过一个全局唯一的标识 `TraceId` 来唯一的确定一条调用链路，并通过 `SpanId` 来确定被调用的系统在调用链路中的层次位置。

三、使用 SOFATracer

SOFATracer 基于 OpenTracing 规范提供了的标准实现，这个标准实现提供的一些基础和核心的链路跟踪能力，如通用的配置能力、异步落地磁盘的日志打印能力、基于 SLF4J MDC 的扩展能力等等，如果期望三方开源组件或者二方组件也能够提供 SOFATracer 的链路数据，那么就需要基于 SOFATracer 的能力提供相应的组件扩展插件，这个 SOFATracer 的扩展插件主要是通过利用各个组件的 Filter 或者类似 Interceptor 的机制，通过代码埋点的方式构造出各个组件期望获取到关键数据来完善和透明各个组件的使用方式。

下面将从实践的角度，向大家介绍如何使用 SOFATracer 构造一个调用链路，本文构造的调用链路是一个如下的调用层次关系：

DEMO 实践的 GitHub 仓库地址(https://github.com/guanchao-yang/sofa-tracer-demo)，基于 SOFATracer 的版本为：`2.2.0-SNAPSHOT`

3.1 本地搭建一个 Zipkin Server

Zipkin Server 的 GitHub 仓库 (https://github.com/openzipkin/zipkin/tree/master/zipkin-server)。我们可以使用最新 RELEASE 的 Zipkin 在本地搭建一个 Server，目前最新的版本为 2.11.3(https://dl.bintray.com/openzipkin/maven/io/zipkin/java/zipkin-server/2.11.3/)，我们下载这个 `zipkin-server-2.11.3-exec.jar` ，然后在本地直接采用如下的方式 `java -jar zipkin-server-2.11.3-exec.jar` ，注意：Zipkin Server 启动时 JDK 至少应该是 JDK8 。

启动后，有类似如下的日志，说明 Zipkin Server 启动成功：

```
2018-08-29 11:59:50.402  INFO 30967 --- [           main] z.s.ZipkinServer
2018-08-29 11:59:50.413  INFO 30967 --- [           main] z.s.ZipkinServer
2018-08-29 11:59:50.541  INFO 30967 --- [           main] ConfigServletWebServerApplicationCo
2018-08-29 11:59:56.850  INFO 30967 --- [           main] o.xnio
2018-08-29 11:59:56.896  INFO 30967 --- [           main] o.x.nio
2018-08-29 11:59:57.096  INFO 30967 --- [           main] o.s.b.w.e.u.UndertowServletWebServe
2018-08-29 11:59:57.102  INFO 30967 --- [           main] z.s.ZipkinServer
```

在本地访问 http://localhost:9411 可以看到如下的界面展示：

3.2 构建一个 SOFARPC 服务

基于一个 SOFABoot 应用构造一个 SOFARPC 服务，关键的几个步骤为：

- 引入 SOFABoot 依赖

```
<parent>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>sofaboot-dependencies</artifactId>
        <version>2.4.7</version>
</parent>
```

- 引入 SOFARPC 依赖、SOFATracer 依赖和 Zipkin 依赖

```
<dependency>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>rpc-sofa-boot-starter</artifactId>
</dependency>

<dependency>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>tracer-sofa-boot-starter</artifactId>
</dependency>

<dependency>
        <groupId>io.zipkin.zipkin2</groupId>
        <artifactId>zipkin</artifactId>
</dependency>

<dependency>
        <groupId>io.zipkin.reporter2</groupId>
        <artifactId>zipkin-reporter</artifactId>
</dependency>
```

- 构造并发布一个 RPC 服务

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:sofa="http://sofastack.io/schema/sofaboot"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframe
            http://sofastack.io/schema/sofaboot http://sofastack.io/schema/sofaboot.xsd"
       default-autowire="byName">

    <bean id="rpcCallServiceImpl" class="com.alipay.sofa.tracer.demo.rpc.impl.RpcCallServiceI
    <sofa:service ref="rpcCallServiceImpl" interface="com.alipay.sofa.tracer.demo.facade.RpcC
        <sofa:binding.bolt/>
    </sofa:service>

</beans>
```

发布的 RPC 服务是 `com.alipay.sofa.tracer.demo.facade.RpcCallService`。

- 配置应用名、日志输出目录和搭建的 Zipkin Server(https://github.com/openzipkin/zipkin/tree/master/zipkin-server) 的地址

在当前应用的配置文件中，配置应用名称、期望的日志输出目录和搭建的 Zipkin Server 的地址，如：

```
# Application Name
spring.application.name=RPCServer
# logging path
logging.path=./logs
# zipkin server url
com.alipay.sofa.tracer.zipkin.baseUrl=http://localhost:9411
# server.port changed to avoid conflict
server.port=8081
```

这里需要解释的参数是 `server.port` 的配置，之所以增加这个端口的配置是为了避免 DEMO 中有多个 Web 应用启动而造成的 `8080` 端口的冲突问题。

- 启动 SOFABoot 应用：在此先不启动应用，待下一步的 H2DB 服务的配置完成后再启动应用

3.3 构建一个访问 H2DB 的服务

在上一个构建的 RPC 服务的应用 `RPCServer` 中，构建一个访问 H2DB 的服务，关键的几个步骤为：

- 引入 SOFABoot 依赖

由于此应用和上一个构建的 `RPCServer` 应用是同一个，所以只需要按照前一个步骤添加依赖即可。

- 引入 H2DB 客户端、druid 和 SOFATracer 的数据源插件依赖

```
<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofa-tracer-datasource-plugin</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.193</version>
</dependency>

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.0.12</version>
</dependency>
```

- 配置 H2DB 数据源

在应用的 XML 配置文件中，可以按照如下的方式配置数据源：

```
<bean id="smartDataSource" class="com.alipay.sofa.tracer.plugins.datasource.SmartDataSource"
    <property name="delegate" ref="simpleDataSource"/>
    <!-- applicationName -->
    <property name="appName" value="${spring.application.name}"/>
    <!-- db -->
    <property name="database" value="h2DataSource"/>
    <!-- type MYSQL, ORACLE-->
    <property name="dbType" value="MYSQL"/>
    <!-- tracer delegate -->
    <property name="clientTracer" ref="clientTracer"/>
</bean>
<!-- dataSource pool -->
<bean id="simpleDataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init"
    <property name="driverClassName" value="org.h2.Driver"/>
    <property name="url" value="jdbc:h2:~/test"/>
    <property name="username" value="sofa"/>
    <property name="password" value="123456"/>
</bean>
<bean id="clientTracer" class="com.alipay.sofa.tracer.plugins.datasource.tracer.DataSourceCli
```


- 全局配置文件 `application.properties` 中，增加对 H2DB 相关的关键配置

```
# h2 web consloe path
spring.h2.console.path=/h2-console
# open h2 web consloe default false
spring.h2.console.enabled=true
# h2 web console
spring.h2.console.settings.web-allow-others=true

spring.datasource.username=sofa
spring.datasource.password=123456
spring.datasource.url=jdbc:h2:~/test
spring.datasource.driver-class-name=org.h2.Driver
```

- 在 RPC 的服务实现中，注入此 H2DB 的数据源实现，并执行简单逻辑，类似如下：

```
@Autowired
private SmartDataSource smartDataSource;

@Override
```

```
public String helloTracer(String param) throws Exception{
    Connection cn = smartDataSource.getConnection();
    Statement st = cn.createStatement();
    st.execute("DROP TABLE IF EXISTS TEST;\n"
            + "CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));");
    return "Param is " + param + " and call success!";
}
```

- 启动 SOFABoot 应用

H2DB 的参考资料参见这里(http://www.h2database.com/html/quickstart.html)

3.4 构建一个 **RESTful** 服务

使用 SOFABoot 基于 Spring MVC 构造一个提供 RESTful 服务的应用，并同时在这个 RESTful 服务中发起对上文应用的 RPC 调用，关键的几个步骤为：

- 引入 SOFABoot 依赖

```
<parent>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>sofaboot-dependencies</artifactId>
        <version>2.4.7</version>
</parent>
```

- 引入 Spring MVC、SOFARPC、SOFATracer 和 Zipkin 依赖

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>rpc-sofa-boot-starter</artifactId>
</dependency>

<dependency>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>tracer-sofa-boot-starter</artifactId>
</dependency>

<dependency>
        <groupId>io.zipkin.zipkin2</groupId>
        <artifactId>zipkin</artifactId>
</dependency>

<dependency>
        <groupId>io.zipkin.reporter2</groupId>
        <artifactId>zipkin-reporter</artifactId>
</dependency>
```

- 构造一个 RESTful 服务并在这个服务中发起一次 RPC 调用

```
@RestController
public class SampleRestController {

    @Autowired
    private RpcCallService rpcCallService;

    private static final String TEMPLATE = "Hello, %s!";

    private final AtomicLong    counter  = new AtomicLong();

    /***
```

```
     * http://localhost:8080/springmvc
     * @param name name
     * @return map
     */
    @RequestMapping("/springmvc")
    public Map<String, Object> springmvc(@RequestParam(value = "name", defaultValue = "SOFATr
        Map<String, Object> resultMap = new HashMap<String, Object>();
        resultMap.put("success", true);
        resultMap.put("id", counter.incrementAndGet());
        resultMap.put("content", String.format(TEMPLATE, name));
        resultMap.put("rpc", rpcCallService.helloTracer(name));
        return resultMap;
    }
}
```

构造的 RESTful 服务为 `http://${host}:8080/springmvc` （host 为此应用部署的 IP 地址），同时在这个服务中发起一次 RPC 调用，被调用的这个 RPC 服务是 `com.alipay.sofa.tracer.demo.facade.RpcCallService` ,调用的方式为 `direct` 方式：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:sofa="http://sofastack.io/schema/sofaboot"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframe
            http://sofastack.io/schema/sofaboot http://sofastack.io/schema/sofaboot.xsd"
       default-autowire="byName">
    <sofa:reference id="rpcCallService" interface="com.alipay.sofa.tracer.demo.facade.RpcCall
        <sofa:binding.bolt>
            <sofa:route target-url="bolt://127.0.0.1:12200"/>
        </sofa:binding.bolt>
    </sofa:reference>
</beans>
```

- 配置应用名、日志输出目录和搭建的 Zipkin Server(https://github.com/openzipkin/zipkin/tree/master/zipkin-server) 的地址

在当前应用的配置文件中，配置应用名称、期望的日志输出目录和搭建的 Zipkin Server 的地址，如：

```
# Application Name
spring.application.name=SpringMVC
# logging path
logging.path=./logs
# zipkin server url
com.alipay.sofa.tracer.zipkin.baseUrl=http://localhost:9411
```

- 启动 SOFABoot 应用

3.5 构建一个 HttpClient 客户端

基于一个 SOFABoot 应用构造一个 HttpClient 客户端并发起对上文的 RESTful 服务的调用，关键的几个步骤为：

- 引入 SOFABoot 依赖

```
<parent>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>sofaboot-dependencies</artifactId>
        <version>2.4.7</version>
</parent>
```

- 引入 HttpClient、SOFATracer 依赖、SOFATracer 的 HttpClient 插件和 Zipkin 依赖

```
<dependency>
        <groupId>org.apache.httpcomponents</groupId>
```

```
        <!-- 版本 4.5.X 系列 -->
        <artifactId>httpclient</artifactId>
</dependency>

<dependency>
        <groupId>com.alipay.sofa</groupId>
        <artifactId>tracer-sofa-boot-starter</artifactId>
</dependency>

<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>sofa-tracer-httpclient-plugin</artifactId>
</dependency>

<dependency>
        <groupId>io.zipkin.zipkin2</groupId>
        <artifactId>zipkin</artifactId>
</dependency>

<dependency>
        <groupId>io.zipkin.reporter2</groupId>
        <artifactId>zipkin-reporter</artifactId>
</dependency>
```

- 构造 HttpClient 发起一次对上文的 RESTful 服务的调用

构造 HttpClient 的过程不详细展开，具体可以参见 DEMO 中的构造方式(https://github.com/guanchao-yang/sofa-tracer-demo/blob/master/httpclient-demo/src/main/java/com/alipay/sofa/tracer/demo/httpclient/client/HttpClientInstance.java) 供参考。其中需要重点强调的是，为了能够使得 HttpClient 这个第三方开源组件能够支持 SOFATracer 的链路调用，SOFATracer 提供了 HttpClient 的插件扩展即 `sofa-tracer-httpclient-plugin`，为了保证 SOFATracer 的 HttpCliet 插件扩展正确的埋点，我们需要使用 `HttpClientBuilder` 去构造 HttpClient 的实例，并需要显示的调用 `SofaTracerHttpClientBuilder.clientBuilder(httpClientBuilder)` 来构造出一个经过 SOFATracer 埋点的 `HttpClientBuilder`，关键代码示例：

```
HttpClientBuilder httpClientBuilder = HttpClientBuilder.create();//SOFATracer
SofaTracerHttpClientBuilder.clientBuilder(httpClientBuilder);
CloseableHttpClient httpClient = httpClientBuilder.setConnectionManager(connManager).disableA
                .build();
```

通过 `SofaTracerHttpClientBuilder` 构造的 HttpClient 在发起对上文的 RESTful 服务调用的时候，就会埋点 SOFATracer 的链路的数据。

- 配置应用名、日志输出目录和搭建的 Zipkin Server(https://github.com/openzipkin/zipkin/tree/master/zipkin-server) 的地址

在当前应用的配置文件中，配置应用名称、期望的日志输出目录和搭建的 Zipkin Server 的地址，如：

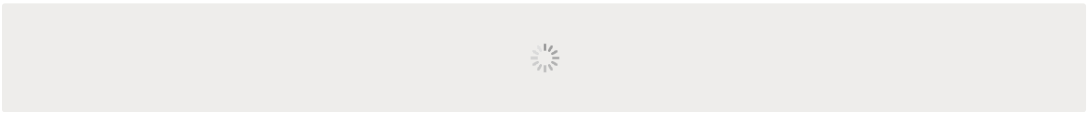
```
# Application Name
spring.application.name=HttpClient
# logging path
logging.path=./logs
# zipkin server url
com.alipay.sofa.tracer.zipkin.baseUrl=http://localhost:9411
```

- 启动 SOFABoot 应用，发起对上文的 RESTful 服务的调用

```
HttpClientInstance httpClientInstance = new HttpClientInstance(10 * 1000);
String httpGetUrl = "http://localhost:8080/springmvc";
String responseStr = httpClientInstance.executeGet(httpGetUrl);
logger.info("Response is {}", responseStr);
```

四、分析调用结果

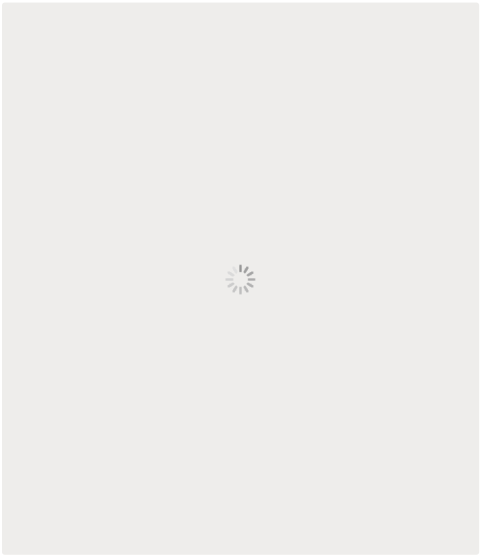
先启动 `RPCServer` 应用提供 RPC 服务，然后启动 `SpringMVC` 提供 RESTful 服务，在前两个应用均成功启动后再启动应用 `HttpClient` ，之后由 HttpClient 发起的调用会构造一个如下的调用链路：



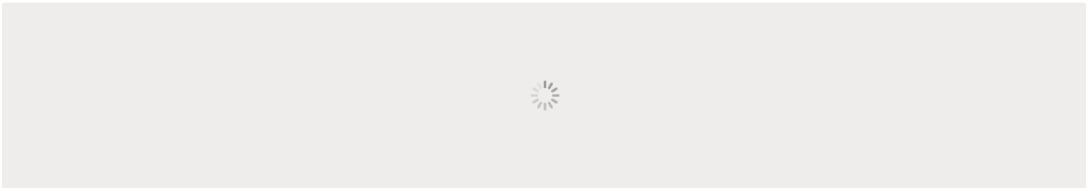
由于，这三个应用我们构造在一个工程中并且其中 RPCServer 和 DB 的访问在一个应用中，并同时配置日志输出目录为：当前工程根目录下的 `logs` 。在启动完成 `HttpCient` 并发起了一次 HttpClient 的调用后，我们可以看到在控制台中有类似如下的成功日志：

```
2018-08-25 19:28:23.346  INFO 7813 --- [           main] c.a.s.t.d.h.HttpClientDemoApplicatio
2018-08-25 19:28:24.650  INFO 7813 --- [           main] c.a.s.t.d.h.HttpClientDemoApplicatio
```

4.1 tracelog 分析



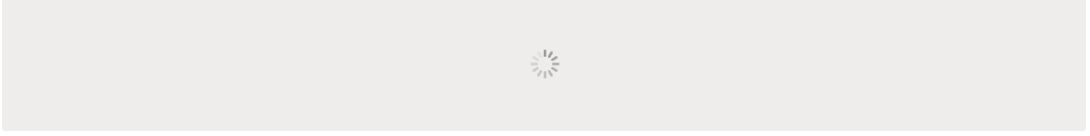
完成一次链路调用后，SOFATracer 会异步的将链路调用数据写到磁盘中，这些日志和链路调用层次的对应关系如下图展示：



具体的日志打印内容不再示例，通过 SOFATracer 打印的 `*-digest.log` 摘要日志，我们可以得到如上的链路调用图。其中，`-stat.log` 是统计日志，默认的输出时间间隔为 60 秒；`shadow` 目录是存放压测日志的目录，主要是避免在生产环境将压测日志和正常的流量的请求日志混淆。

4.2 Zipkin 展示分析

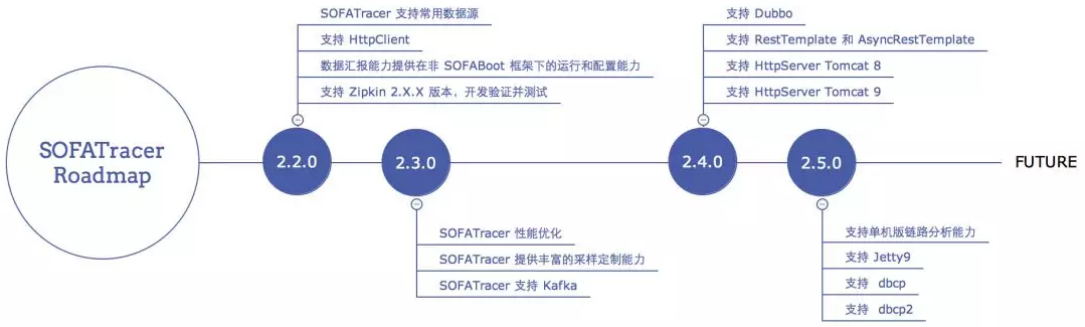
在开始的步骤中，本文演示了在本地搭建 Zipkin Server ，并在项目中配置 SOFATracer 的链路数据的远程上报到 Zipkin Server 的上报地址 `com.alipay.sofa.tracer.zipkin.baseUrl=http://localhost:9411` ，那么我们可以打开 Zipkin Server 的访问地址 http://localhost:9411/zipkin/ 并在导航栏的 `Service Name` 中选择要展示的调用链路，最后可以在 Zipkin Server 中得到类似如下的链路调用图：



五、总结

本文主要介绍 SOFATracer 在 SOFABoot 应用中的实践，通过三个应用并访问 DB 来构建一个调用链路，即 `HttpClient` -> `SpringMVC` -> `RPCServer` -> `H2DB` 来演示 SOFATracer 的链路跟踪能力，每次链路调用 SOFATracer 均会有相应的调用地址异步的落地到磁盘中，同时在配置了 Zipkin 的远程上报能力后，也可以将链路数据远程上报到 Zipkin Server 做链路展示。

未来我们会支持越来越多的开源组件，详见我们的下图的 **ROADMAP** ，也非常欢迎大家积极参与共建。



附录：相关文档链接

- SOFATracer :https://github.com/alipay/sofa-tracer
- 本文 DEMO：https://github.com/guanchao-yang/sofa-tracer-demo
- Zipkin：https://github.com/openzipkin/zipkin/tree/master/zipkin-server
- H2DB 快速开始：http://www.h2database.com/html/quickstart.html
- Disruptor：https://github.com/LMAX-Exchange/disruptor
- OpenTracing 规范：http://opentracing.io/documentation/pages/spec.html
- Zipkin Server 可执行 Fat Jar 下载地址：https://dl.bintray.com/openzipkin/maven/io/zipkin/java/zipkin-server/2.11.3/