30 Spring Cloud Sleuth 实践

更新时间：2019-07-29 10:32:41

> 自信和希望是青年的特权。
>
> ——大仲马

上一节课我们介绍了 Spring Cloud Sleuth 相关术语和工作原理，这节课我们将学习如何使用 Spring Cloud Sleuth 进行信息采集。

我们先来一个最简单的 Hello World。

快速入手

创建示例项目 spring-cloud-sleuth，按照以下步骤进行配置。

添加依赖

应用中增加 Sleuth 非常简单，只需在 pom.xml 增加以下的依赖：

```
<dependencies>
 <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>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

添加 `spring-boot-starter-web` 依赖包，是因为下面我们需要模拟测试 Web 请求。

创建服务

我们来创建一个 hello world 服务，打印一行日志来看看 Sleuth 都干了哪些事。

```
@RestController
public class SleuthController {
    private static final Log log = LogFactory.getLog(SampleController.class);
    @RequestMapping("/hello")
    public String hello() {
        log.info("Doing some work");
        return "hello world";
    }
}
```


一个很简单的服务，打印日志是因为 Sleuth 和日志已经做了深度融合，会将 Sleuth 收集的信息打印出来。

测试

上面工作准备完成之后，启动项目，在浏览器中访问地址：`http://localhost:8080/hello`调用上面创建好的服务，这时会在控制台看到这样一行日志：

```
2019-05-03 10:48:54.890  INFO [Spring Cloud Sleuth,4e088a46074173ee,4e088a46074173ee,false] 20124 --- [nio-8080-exec-2] com
.justdojava.sleuth.HelloController    : Doing some work
```

`[Spring Cloud Sleuth,4e088a46074173ee,4e088a46074173ee,false]`即为本次 Sleuth 输出的内容，日志的格式为：[application name, traceId, spanId, export]，上节我们已经做过解释，分别是应用名、traceId、spanId和是否对外输出。

这样最简单的一个 Sleuth 测试就完成了。

请求调用

上面只是一个最简单的调用示例，我们来看看如果在方法中调用另外一个方法，Sleuth 是如何记录数据的。

创建一个 hi() 方法，通过 restTemplate 去调用 hi2() 方法。代码如下：

```
@RequestMapping("/")
public String hi() throws InterruptedException {
    log.info("hi!");
    Thread.sleep(this.random.nextInt(1000));

    String s = this.restTemplate
            .getForObject("http://localhost:" + this.port + "/hi2", String.class);
    return "hi/" + s;
}

@RequestMapping("/hi2")
public String hi2() throws InterruptedException {
    log.info("hi2!");
    int millis = this.random.nextInt(1000);
    Thread.sleep(millis);
    this.tracer.currentSpan().tag("random-sleep-millis", String.valueOf(millis));
    return "hi2";
}
```

添加完成后重新启动项目，在浏览器中访问访问地址：`http://localhost:8080/`，查看控制台日志的打印信息。

```
2019-05-03 12:59:42.745  INFO [Spring Cloud Sleuth,063ad9837aebfe4a,063ad9837aebfe4a,true] 29536 --- [nio-8080-exec-5] com.
justdojava.sleuth.SampleController   : hi!
2019-05-03 12:59:43.216  INFO [Spring Cloud Sleuth,063ad9837aebfe4a,e5a424ae0f3007f2,true] 29536 --- [nio-8080-exec-6] com.
justdojava.sleuth.SampleController   : hi2!
```

因为涉及到两次调用，因此产生了两个 Span ,第一个 Span 和第二个 Span 的 ID 不同，从日志打印也可以看出两个 Span 有着同样的 traceId，表面它们属于同一个 Trace。

异步调用

我们再来模拟以下异步线程调用时，Sleuth 是如何记录 Span 信息的。下面进行演示：

首先需要在启动类添加注解 `@EnableAsync`，开启应用异步调用的功能。

```
@EnableAsync
public class SleuthApplication {
}
```

创建一个 SleuthService 类，代码如下：

```
public class SleuthService {
    private static final Log log = LogFactory.getLog(SleuthController.class);
    @Autowired
    private Tracer tracer;
    private Random random = new Random();
    @Async
    public void background() throws InterruptedException {
        log.info("background");
        int millis = this.random.nextInt(1000);
        Thread.sleep(millis);
        this.tracer.currentSpan().tag("background-sleep-millis", String.valueOf(millis));
    }
}
```

- `@Async`，添加此注解的方法会自动异步执行。接下来在 SleuthController 中添加调用此方法的入口。

```
@RequestMapping("/async")
public String async() throws InterruptedException {
    log.info("async");
    this.background.background();
    return "async";
}
```

添加完之后，重新启动项目，访问地址`http://localhost:8080/async`，查看控制台日志的打印信息。

```
2019-05-03 13:18:42.279  INFO [Spring Cloud Sleuth,36d6f60de86c3e6f,36d6f60de86c3e6f,true] 2064 --- [nio-8080-exec-1] com.j
ustdojava.sleuth.SleuthController   : async
2019-05-03 13:18:42.293  INFO [Spring Cloud Sleuth,36d6f60de86c3e6f,31c32c5f44694e80,true] 2064 --- [         task-1] com.j
ustdojava.sleuth.SleuthController   : background
```

从日志的打印情况来看和上面请求调用比较类型，同属于一个 TraceId，各自有各自的 SpanId。通过此示例可以表明 Sleuth 支持异步调用的信息收集。

定时任务

接下来我们测试 Sleuth 在定时任务 `@Scheduled` 中的信息收集情况。

首先在启动类上添加 `@EnableScheduling` 注解，开启应用的定时任务功能。

```
@EnableScheduling
public class SleuthApplication {
}
```

在 SleuthService 类中添加定时任务，定时任务中去调用 background() 方法。

```
@Scheduled(fixedDelay = 36000)
public void scheduledWork() throws InterruptedException {
    log.info("Start some work from the scheduled task");
    this.background();
    log.info("End work from scheduled task");
}
```


我们设置每 36 秒调用一次，添加完成后重新启动项目，查看控制台的日志输出信息。

```
2019-05-03 13:40:02.431  INFO [Spring Cloud Sleuth,48adb9bab82b50cb,48adb9bab82b50cb,true] 29084 --- [   scheduling-1] com.
justdojava.sleuth.SleuthController   : Start some work from the scheduled task
2019-05-03 13:40:02.431  INFO [Spring Cloud Sleuth,48adb9bab82b50cb,48adb9bab82b50cb,true] 29084 --- [   scheduling-1] com.
justdojava.sleuth.SleuthController   : background
2019-05-03 13:40:03.381  INFO [Spring Cloud Sleuth,48adb9bab82b50cb,48adb9bab82b50cb,true] 29084 --- [   scheduling-1] com.
justdojava.sleuth.SleuthController   : End work from scheduled task
2019-05-03 13:40:39.382  INFO [Spring Cloud Sleuth,51d22d621a0f96d1,51d22d621a0f96d1,true] 29084 --- [   scheduling-1] com.
justdojava.sleuth.SleuthController   : Start some work from the scheduled task
2019-05-03 13:40:39.383  INFO [Spring Cloud Sleuth,51d22d621a0f96d1,51d22d621a0f96d1,true] 29084 --- [   scheduling-1] com.
justdojava.sleuth.SleuthController   : background
2019-05-03 13:40:39.653  INFO [Spring Cloud Sleuth,51d22d621a0f96d1,51d22d621a0f96d1,true] 29084 --- [   scheduling-1] com.
justdojava.sleuth.SleuthController   : End work from scheduled task
...
```

通过日志分析可以看出，每次定时任务都会产生一个新的 Trace，并且调用过程中 SpanId 都是一致的。说明定时任务调用和页面调用是不同的，页面调用异步方法时会产生新的 Span ，而定时任务调用异步方法仍然使用的时同一个 Span。通过该实验也可以说明 Sleuth 完全支持定时任务信息收集。

总结

本节我们学习了 Spring Cloud Sleuth 在单体应用中如何收集数据信息，实践了 Sleuth 在 Web 调用、异步调用、定时任务中的实验方式。在真正的项目中，一般不会单独的使用 Spring Cloud Sleuth ，往往是结合 ZipKin 等图形界面软件一起使用。Zipkin 的介绍和使用在下一节会再给大家介绍。

参考链接：

https://github.com/spring-cloud/spring-cloud-sleuth

https://cloud.spring.io/spring-cloud-sleuth/spring-cloud-sleuth.html#_running_examples

29 分布式链路跟踪和 Spring Cloud Sleuth

31 Zipkin 入门介绍