《ServiceComb + Zipkin : 源码解读》

目录

[《ServiceComb + Zipkin : 源码解读》 1](#_Toc535600322)

[1. Zipkin 是什么 1](#_Toc535600323)

[2. ServiceComb java-chassis中如何构建Zipkin框架 1](#_Toc535600324)

[3. ServiceComb Java chassis 的handler-tracing-zipkin模块解读 1](#_Toc535600325)

[ServiceComb Java chassis 的 handler-tracing-zipkin模块源码解读 1](#_Toc535600326)

### 1. Zipkin 是什么

#### 1.1 Zipkin简介

Zipkin是一种分布式调用追踪系统，通过获取时序信息来追踪服务架构中的调用链路，帮助分析服务架构中的时延问题。Zipkin基于Dapper论文实现，由Twitter公司开源。

#### Zipkin架构

##### 1.2.1 Zipkin服务调用链追踪 流程图

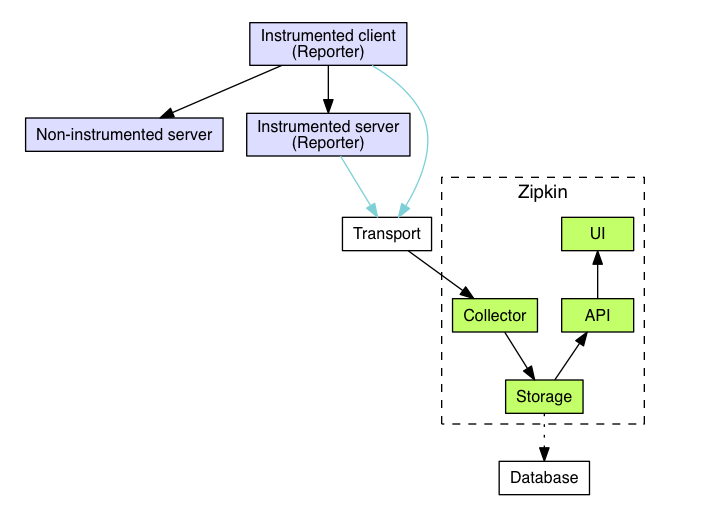


图1 Zipkin 服务调用链追踪 流程图

图1中,Instrumented Client 和 Instrumented Server 是分布式架构中被追踪的两个服务，Client会调用Server提供的服务，形成调用链。追踪器通过transport向Zipkin上报调用链上的Trace信息。zipkin的collector负责收集Trace信息，并由Storage模块将收集到的数据存储在对应的存储介质中，并提供检索Trace信息的API给UI界面使用，UI提供给用户查询Trace追踪时序信息和调用关系的界面。Non-instrumented Server是没有被追踪的Server，不会上报Trace信息。

##### 1.2.2 Zipkin Server模块 及 zipkin组件

Zipkin Server主要包括4个模块：Collector收集器、Storage存储、API、UI界面。Zipkin Collecter 收集到Trace信息，会将Trace信息存储在zipkin storage中，API会查询和索引存储的trace信息，提供给web UI使用。

Zipkin有两个重要组件：追踪器 和 reporter。

追踪器：收集服务中调用的耗时信息并将其上报，不同的编程语言有各自的追踪器实现，java语言的追踪器实现是github上的‘openzipkin/brave’；

reporter组件：用于发送追踪器记录的Trace信息到Zipkin 收集器，主要实现了HTTP，Kafka和Scribe三种传输方式。

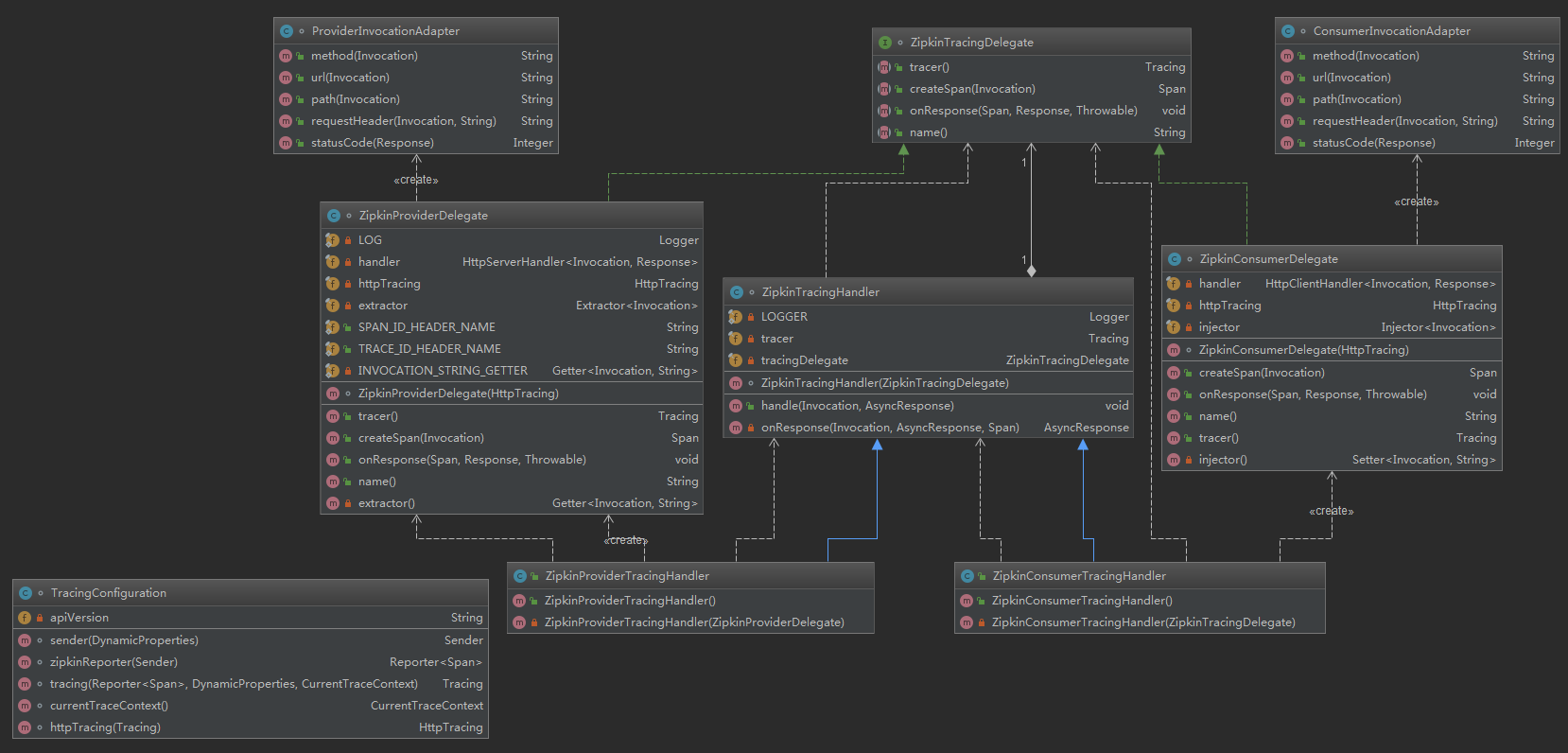
### 2. ServiceComb java-chassis中如何构建Zipkin框架

ServiceComb 集成了zipkin 提供自动调用链追踪能力。ServiceComb 通过handler处理链来实现分布式调用链追踪，使用handler-tracing-zipkin 模块支持zipkin，主要应用于追踪基于ServiceComb java chassis 的微服务。ServiceComb java-chassis主要实现了zipkin服务的客户端，向zipkin服务发送追踪信息。Java chassis使用zipkin的brave组件去追踪java chassis中的服务，记录调用链中的耗时信息，并使用zipkin中的reporter组件发送Trace信息到zipkin 收集器中,如此一来，只需要在后台运行zipkin分布式追踪服务，即可在web 上查看服务调用链的追踪信息。

ServiceComb项目如何使用zipkin实施调用链追踪，可以参见 <http://servicecomb.apache.org/docs/tracing-with-servicecomb/>

### 3. ServiceComb Java chassis 的handler-tracing-zipkin模块解读

Handler-tracing-zipkin 模块的UML 类图如下：



TrcaingConfiguration ：初始化配置httpTracing，并注册为bean对象，载入IOC容器中。

ZipkinTracingDelegate 接口：定义了tracer(),createSpan(Invocation),onResponse(Span,Response,Throwable),name()等方法，用于创建http请求追踪器并命名，创建Span记录追踪数据，响应生成完整的Span。

ZipkinConsumerDelegate ：实现了ZipkinTracingDelegate接口的所有方法，并增加量自己的injector()方法用于发送Tracer内容到http请求中。ZipkinConsumerDelegate依赖ConsumerInvocationAdapter（消费端适配器类）创建消费端调用链追踪处理器。

ZipkinProviderDelegate：实现了ZipkinTracingDelegate接口的所有方法，并增加量自己的extractor()方法用于获取Tracer内容从http请求中。ZipkinProviderDelegate 依赖ProviderInvocationAdapter（服务端适配器类）创建服务端调用链追踪处理器

ZipkinTracingHandler：成员变量包含ZipkinTracingDelegate，是聚合关系；同时依赖ZipkinTracingDelegate类进行初始化。该类实例化ZipkinTracingDelegate对象，用于创建追踪器跟踪http请求，并打印跟踪信息到日志中。

ZipkinConsumerTracingHanler：继承父类ZipkinTracingHandler，依赖ZipkinConsumerDelegate 和 ZipkinTracingHandler，用于注入实例化的httpTracing对象，是handler-tracing-zipkin模块消费端的入口。

ZipkinProviderTracingHandler：继承父类ZipkinTracingHandler，依赖ZipkinProviderDelegate 和 ZipkinTracingHandler，用于注入实例化的httpTracing对象，是handler-tracing-zipkin模块服务端的入口。

### ServiceComb Java chassis 的 handler-tracing-zipkin模块源码解读

#### 4．1 Reporter，httpTracing的初始化

#### 4．1.1 Reporter类初始化，并加载到IOC容器中

#### C:\Users\ZHOUZH~1\AppData\Local\Temp\1\WeChat Files\03f4184ea5077cfcb510630d0f7425e.png

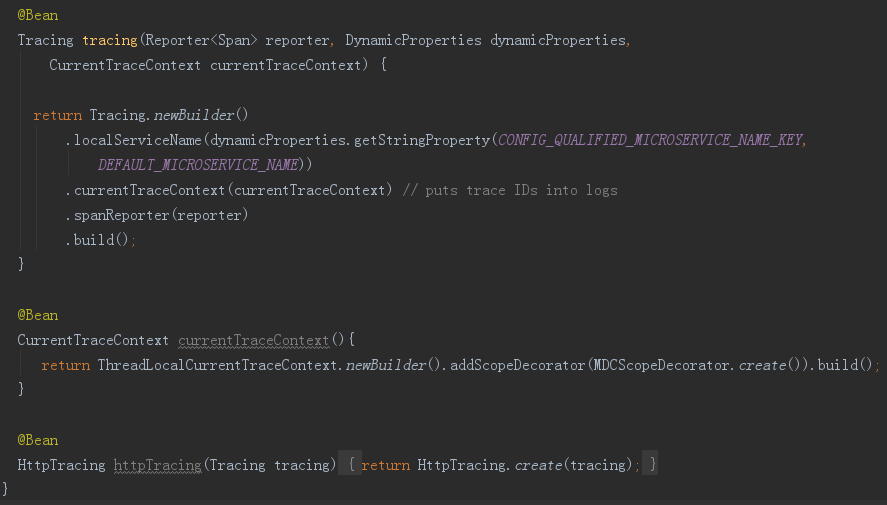
Reporter组件控制发送span到zipkin server。

AsyncReporter: reporter的实现类，使用了两个重要类SpanBytesEncoder和 Sender，初始化reporter。

SpanBytesEncoder ：span编码器，将Span编码为二进制，程序中默认使用json.v2编码。

Sender：依赖io.zipkin.reporter2:zipkin-sender-okhttp3初始化，将编码后的Span二进制数据发送给zipkin。

#### 4.1.2 httpTracing 类初始化，并加载到IOC容器中



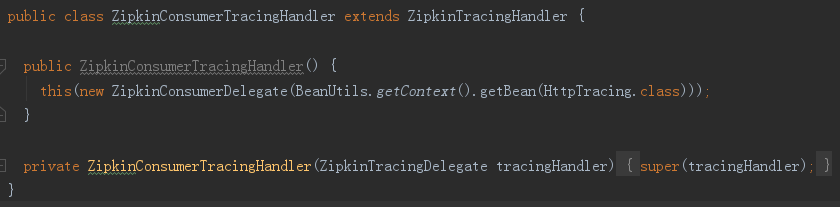
httpTracing类包含对追踪器的应用，可用于创建Http请求的追踪器

Tracing.newBuilder：用于初始化tracing，localServiceName方法绑定服务名字，currentTraceContext 方法绑定当前追踪内容，spanReporter方法绑定reporter组件。

ThreadLocalCurrentTraceContext.newBuilder()：初始化currentTraceContext，并提取当前追踪的traceId放到currentTraceContext中

HttpTracing.create(Tracing)：初始化httpTracing

#### 4.2 获取HttpTracing 实例对象，handler-tracing-zipkin模块消费端追踪入口



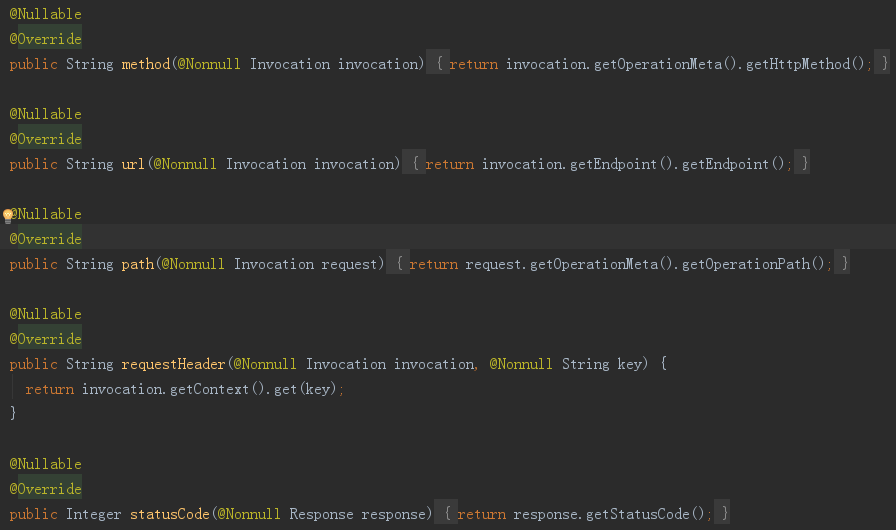
BeanUtils.getContext().getBean(HttpTracing.class) ：从IOC容器中获取httpTracing对象

new ZipkinConsumerDelegate(BeanUtils.getContext().getBean(HttpTracing.class)) ：使用httpTracing 实例化 ZipkinConsumerDelegate对象

ZipkinConsumerTracingHandler(ZipkinTracingDelegate tracingHandler) { super(tracingHandler);} 使用ZipkinConsumerDelegate实例化父类ZipkinTrcacingHandler对象

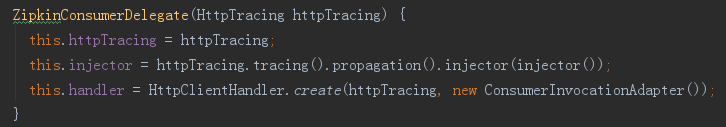
#### 4.3 消费端追踪：使用httpTracing创建处理器，追踪http调用链请求，接收服务端响应，上传Span信息到zipkin

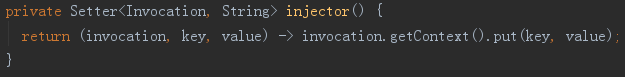
##### 4.3.1 配置HTTP请求的消费端追踪适配器



ConsumerInvocationAdapter 消费端适配器可以从http request中获取各种信息，如method, url , path , header，statuscode。

##### 4.3.2 httpTracing创建handler消费端处理器

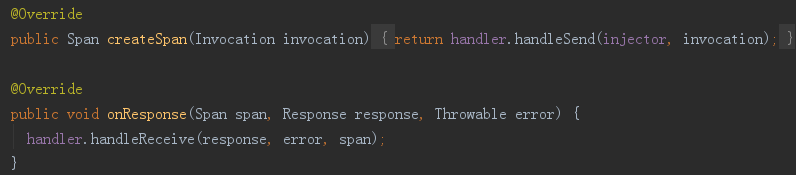




this.handler = HttpClientHandler.*create*(httpTracing, new ConsumerInvocationAdapter())：创建适配于当前http请求的 handler追踪处理器

httpTracing.tracing().propagation().injector(injector()): 初始化inject，用于发送tracer信息到http请求中。

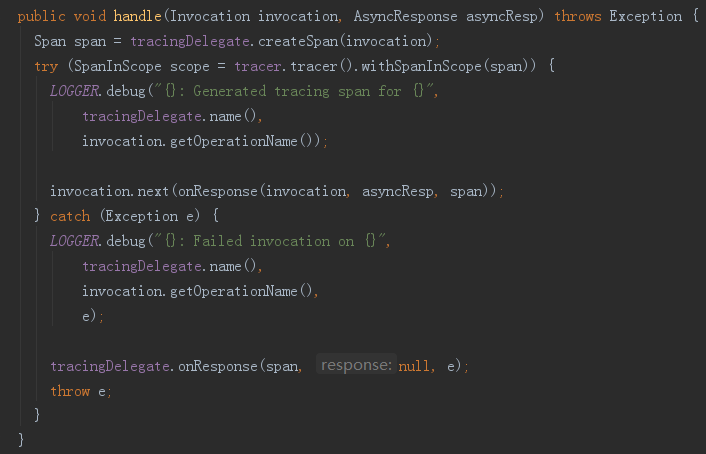
##### 4.3.3 消费端handler处理器创建Span，完成并上报Span



handler.handleSend(injector, Invocation) : 使用injector发送tracer信息到http request中，创建Span，Span的kind类型设置为CLIENT,Span的name设置为request的method（GET或者POST）将当前请求的路径（http.path）写入Span, 调用span.start()设置开始时间。

handler.hanldeReceive(Response, Throwable, Span) : 客户端收到服务端响应后，将http状态码(http.statusCode)写入Span中，如果出错，则将错误写入Span中，最后调用span.finish()，触发reporter组件将Span数据上报到Zipkin。

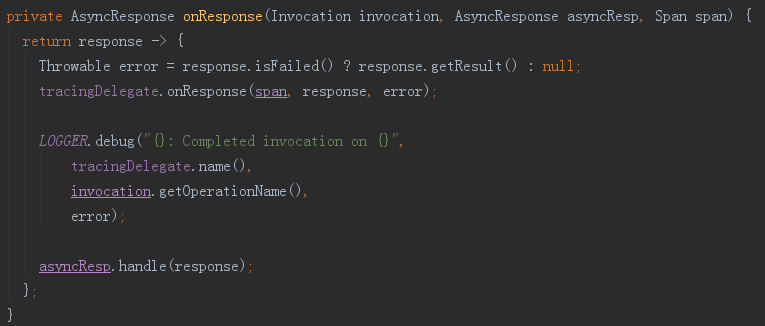
##### 4.3.4 追踪消费端http请求及日志输出



TracingDelegate.createSpan(invocation)： 调用handler.handleSend方法，创建Span。

SpanInSope ：tracer.tracer().withSpanInScope(Span) 创建SpanInSope对象，以供在当前线程内可获取currentTraceContext(当前追踪内容)。

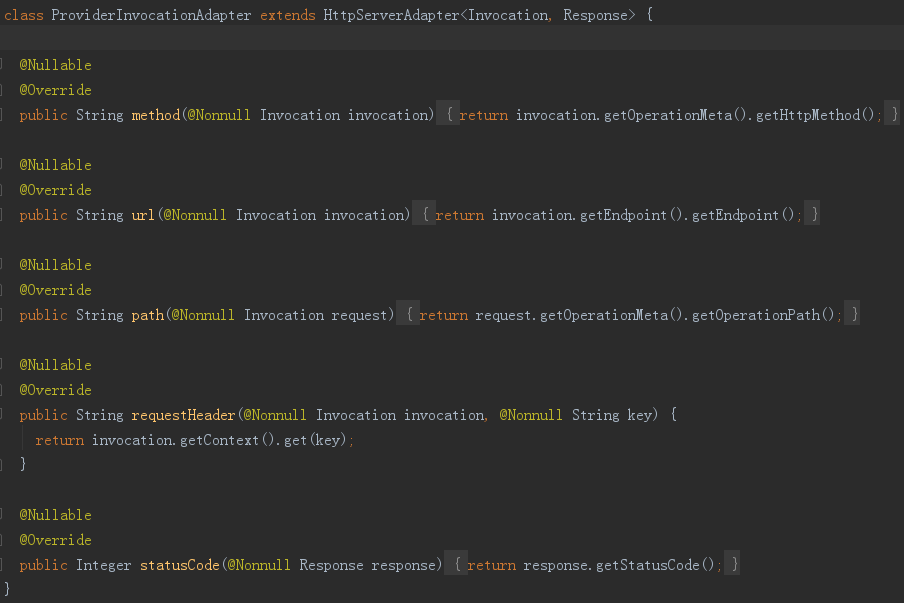
LOGGER.debug：记录当前调用操作的追踪情况，tracingDelegate.name()获取追踪器名字，invocation.getOperationName()获取调用操作名称。



onResponse(Invocation,AsyncResponse,Span): 获取调用链响应，触发handler.handleReceive方法，上报Span,并记录日志

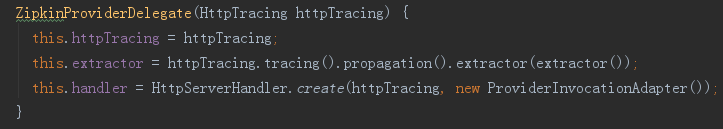
#### 4.4 服务端追踪：使用httpTracing创建处理器，获取http调用链的tracing信息，创建Span，发送服务端响应，上报服务端span到zipkin。

##### 4.4.1 配置服务端http适配器



ProviderInvocationAdapter 消费端适配器可以从http request中获取各种信息，如method, url , path , header，statusCode。

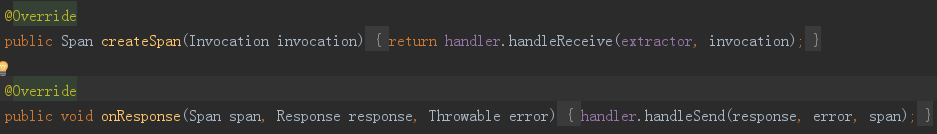
##### 4.4.2 httpTracing创建handler服务端处理器



HttpClientHandler.*create*(httpTracing, new ProviderInvocationAdapter())：创建适配于当前http处理链的 handler追踪处理器，extractor用于获取http request中的tracer内容。

httpTracing.tracing().propagation().extractor(extractor()): 初始化extractor，用于获取request中的追踪内容

##### 4.4.3 服务端handler处理器创建Span，完成并上报Span



handler.handleReceive(extractor, Invocation) : 服务端接收请求，使用extractor从request中获取tracer 内容，创建Span，将Span的kind类型设置为SERVER，Span的name设置为request的method（GET或者POST），将当前请求的路径（http.path）写入Span, 调用span.start()设置开始时间。

handler.hanldeSend(Response, Throwable, Span) : 服务端发送响应，将http状态码写入Span的Tag中，如果出错，则将错误写入Tag中，最后调用span.finish()，触发reporter的report方法将Span信息上报到Zipkin。

##### 4.4.4 追踪服务端http请求及日志输出

此处代码实现和4.3.4 追踪消费端http请求及日志输出代码一致，参见4.3.4。