**[学习微服务-第5天] ServiceComb+Zipkin源码解读**

**SeviceComb + Zipkin 简介**

ServiceComb 是Apache的微服务顶级项目，在微服务框架中，微服务之间通过网络进行通信，我们必须处理所有与网络相关的问题，例如延迟，超时和分区。随着部署的微服务越来越多，我们需要系统监控微服务网络延迟和请求流。  
上篇文章我们介绍了如何使用ServiceComb与Zipkin进行协同定位微服务应用的异常的微服务和具体异常函数。

本篇将介绍ServiceComb如何通过自身handler处理链机制支持Zipkin 微服务级别和函数级别的调用链追踪的。

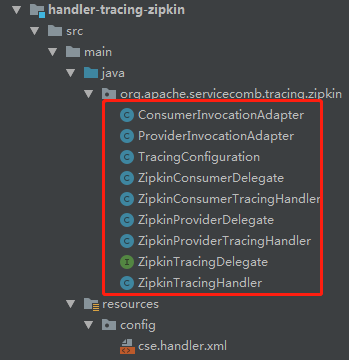
**ServiceComb 如何支持zipkin**

ServiceComb 提供了处理链机制，消费端和服务端的调用链请求都会经过该处理链，通过扩展handler处理链接口，可以实现负载均衡、熔断容错、流量控制等能力。同样，调用链追踪能力也是通过扩展该接口实现的。

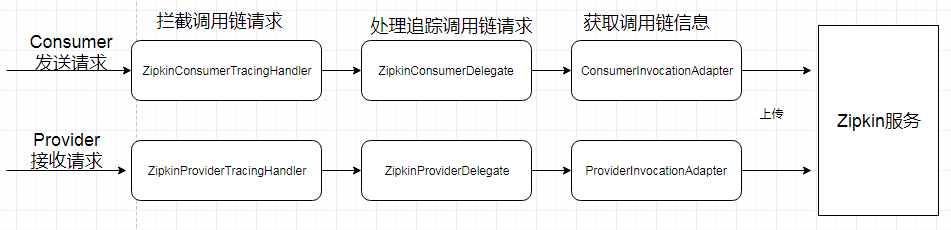
关于ServiceComb处理链参考

https://docs.servicecomb.io/java-chassis/zh\_CN/references-handlers/intruduction.html

ServiceComb 扩展handler处理链接口，编写了handler-tracing-zipkin 模块。Handler-tracing-zipkin 模块在java-chassis/handler处理链下，模块内容如下。



handler-tracing-zipkin模块实现追踪调用链记录数据和上传追踪数据到zipkin服务，即可支持Zipkin。



如果读者对于使用ServiceComb对于Zipkin支持的能力还不熟悉，可参考ServiceComb官网相关文档

1.http://servicecomb.apache.org/docs/tracing-with-servicecomb/

2.https://servicecomb.apache.org/cn/docs/customized-tracing-with-servicecomb/

**handler-tracing-zipkin模块源码解读**

每一次接口调用请求都会触发handler链的处理，而在这个handler链当中，ServiceComb专门为Zipkin编写了handler类ZipkinConsumerTracingHandler

和ZipkinProviderTracingHandle进行适配。下面我们来看下这两个类。

**ZipkinConsumerTracingHandler 和ZipkinProviderTracingHandler**

查看这两个类源码可知，都继承自ZipkinTracingHandler  
，都只有两个构造器。

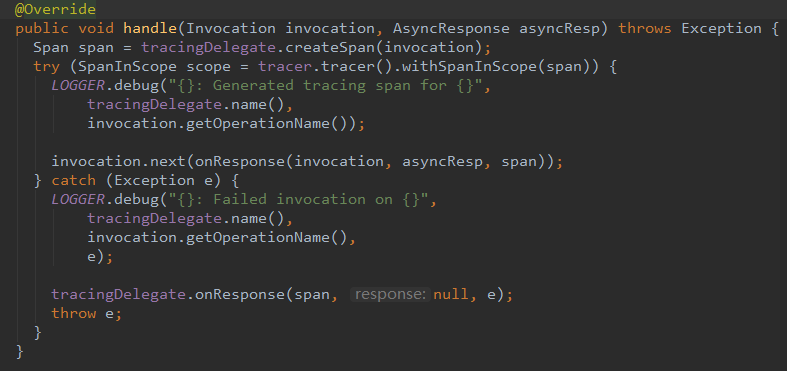
区别在于分别向父类构造器传递了不同的ZipkinTracingDelegate实现。

ZipkinTracingDelegate实现分别为ZipkinConsumerDelegate和ZipkinProviderDelegate。

这两个代理类分别封装了对应的Zipkin请求消费和请求生产操作。下面重点看下ZipkinTracingHandler的源码实现。

**ZipkinTracingHandler**

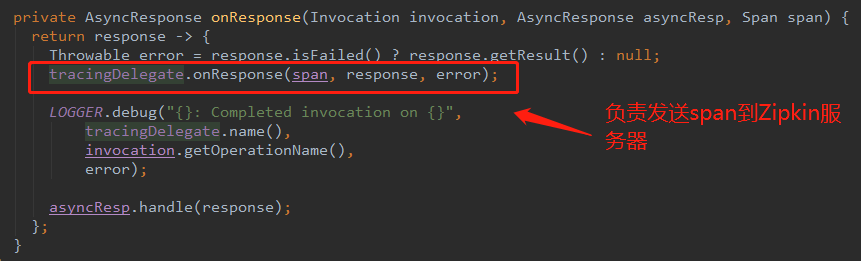
handler类最重要的方法是handler方法，该方法接收一个Invocation对象和AsyncResponse对象（全是ServiceComb内置对象）。Invocation对象包含当前调用相关信息（包括HttpServletRequest对象）。下面我们看下这个方法做了什么事情。



**handler方法执行步骤：**

一、调用了tracingDelegate.createSpan(invocation)方法创建了一个span。tracingDelegate是一个ZipkinTracingDelegate对象。

二、调用当前对象的onResponse方法封装成一个AsyncResponse对象。该对象是是一个函数式对象，在如下图的函数体中可看到最终调用了tracingDelegate.onResponse(span, response, error)上传span到Zipkin服务器。



三、调用invocation.next()方法将生成的AsyncResponse对象传递给下一个handler处理。

四、如果在以上操作中发生异常，将调用tracingDelegate.onResponse(span, null, e)方法发送带有异常信息的span到Zipkin服务器。

从以上分析我们可以看到tracingDelegate十分重要，下面我们接着看这个ZipkinTracingDelegate接口到底做了什么。

**ZipkinTracingDelegate接口**

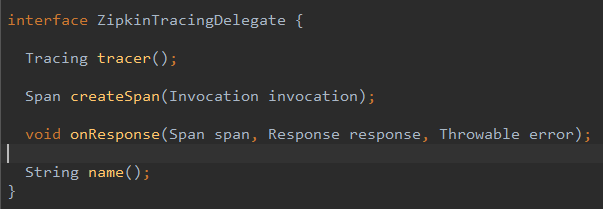
如下图，该接口定义了4个方法。

1.tracer() ,获取对应的追踪器

2.createSpan(), 根据Invocation对象携带的信息创建对应的span

3.onResponse()，将span对象和对应的响应信息和异常信息上传到Zipkin服务器

4.name() , 区分消费操作和生产操作

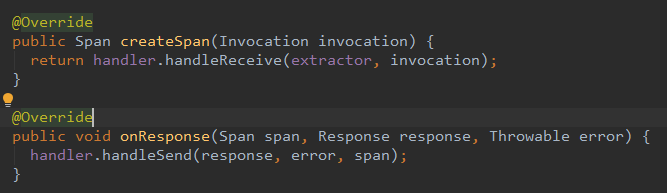


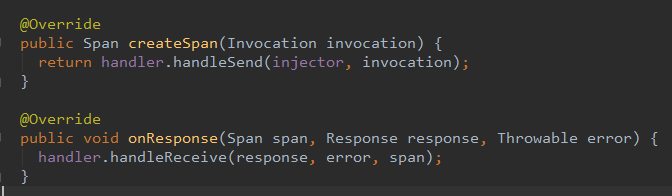
该接口有两个实现类ZipkinConsumerDelegate和ZipkinProviderDelegate，分别对应请求消费操作和请求生产操作。

下面我们可以看到两个实现的区别。

**ZipkinConsumerDelegate和ZipkinProviderDelegate**

下面我们先上两张图片仔细对比一下，第一张是ZipkinConsumerDelegate类，第二张是ZipkinProviderDelegate类。





我们会发现它们都是用handler变量来进行相应的操作，注意这里的handler变量在两个类分别是不一样的类型，ZipkinConsumerDelegate的handler变量是HttpClientHandler对象，而ZipkinProviderDelegate的hanler变量是HttpServerHandler对象。HttpClientHandler和HttpServerHandler类都是final修饰的类，不可继承。前面我们看到创建span和发送span都是由这两个类来负责，那么我们来看下这两个对象怎么生成的。

**仔细观察可发现**

ZipkinConsumerDelegate构造器部分代码：

this.handler = HttpClientHandler.create(httpTracing, new ConsumerInvocationAdapter());

ZipkinProviderDelegate构造器部分代码 ：

this.handler = HttpServerHandler.create(httpTracing, new ProviderInvocationAdapter());

从上面两段构造器代码中可发现HttpClientHandler和HttpServerHandler在创建对象时都分别传入ConsumerInvocationAdapter对象和ProviderInvocationAdapter对象，这两个对象分别继承了Zipkin的HttpClientAdapter和HttpServerAdapter抽象类，提供了属于ServiceComb本身的客户端信息和服务端信息。而Zipkin可以在不改动代码的情况下获取到这些定制信息并用于调用链追踪。

至此，ServiceComb如何一步一步去实现zipkin分布式调用链追踪，已经解读完毕。