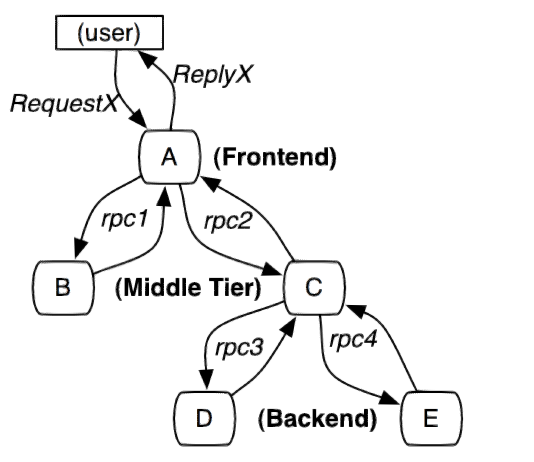
TRACING

先介绍一个概念：分布式跟踪，或分布式追踪。

对于分布式服务，每一个请求过来后，会经过多个业务系统并留下足迹，并产生对各种Cache或DB的访问，但是这些分散的数据对于问题排查，或是流程优化都帮助有限。对于这么一个跨进程/跨线程的场景，汇总收集并分析海量日志就显得尤为重要。要能做到追踪每个请求的完整调用链路，收集调用链路上每个服务的性能数据，计算性能数据和比对性能指标（SLA），甚至在更远的未来能够再反馈到服务治理中，那么这就是分布式跟踪的目标了。

在业界，twitter 的 zipkin 和淘宝的鹰眼就是类似的系统，它们都起源于 Google Dapper 论文。



这里主要介绍zipkin

Zipkin 是一款开源的分布式实时数据追踪系统（Distributed Tracking System），基于 Google Dapper 的论文设计而来，由 Twitter 公司开发贡献。其主要功能是聚集来自各个异构系统的实时监控数据，用来追踪微服务架构下的系统延时问题。

https://github.com/openzipkin/zipkin

http://bigbully.github.io/Dapper-translation/

三个具体的设计目标：

1.低消耗：跟踪系统对在线服务的影响应该做到足够小。在一些高度优化过的服务，即使一点点损耗也会很容易察觉到，而且有可能迫使在线服务的部署团队不得不将跟踪系统关停。

2.应用级的透明：对于应用的程序员来说，是不需要知道有跟踪系统这回事的。如果一个跟踪系统想生效，就必须需要依赖应用的开发者主动配合，那么这个跟踪系统也太脆弱了，往往由于跟踪系统在应用中植入代码的bug或疏忽导致应用出问题，这样才是无法满足对跟踪系统“无所不在的部署”这个需求。面对当下想Google这样的快节奏的开发环境来说，尤其重要。

3.延展性：Google至少在未来几年的服务和集群的规模，监控系统都应该能完全把控住。

重要概念：

1、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。

2、Trace：一系列spans组成的一个树状结构，请求一个微服务系统的API接口，这个API接口，需要调用多个微服务，调用每个微服务都会产生一个新的Span，所有由这个请求产生的Span组成了这个Trace。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。

3、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 这些注解包括以下：

\* 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

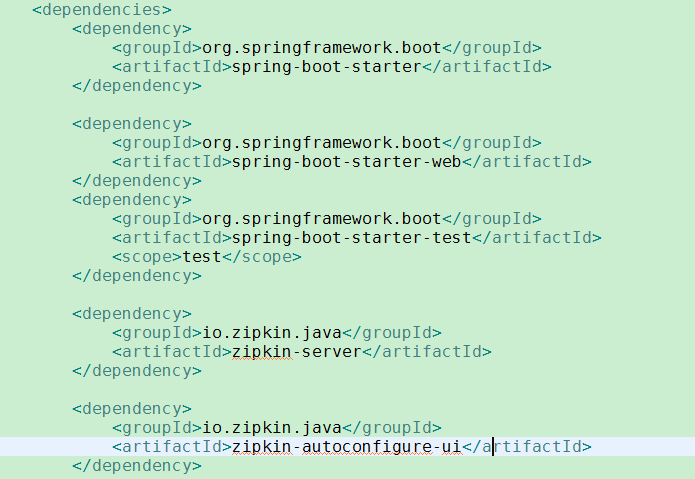
\* cr - Client Received -表明span的结束，客户端成功接收到服务端的回复，如果cr减去cs时间戳便可得到客户端从服务端获取回复的所有所需时间。

项目server-zipkin

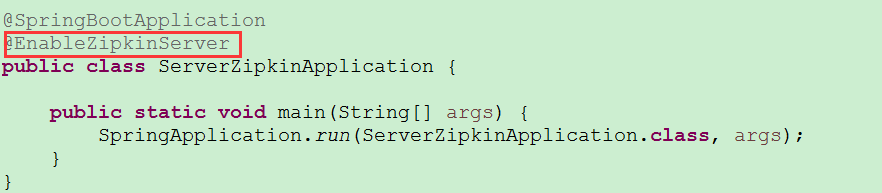
启动zipkin两种方式，第一种jar包的形式自行百度很简单。

第二种代码

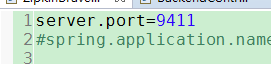
pom.xml



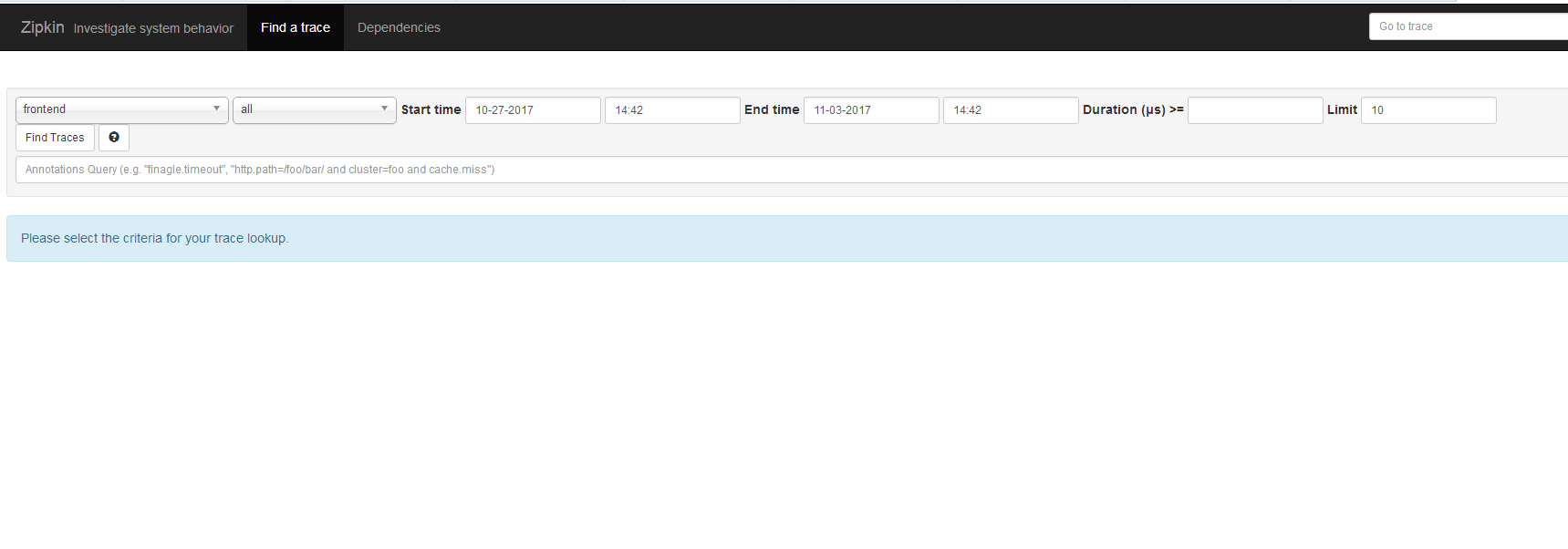
启动类添加注释



配置文件



启动访问9411就可以



二、Brave

2.1、简介

Brave 是用来装备 Java 程序的类库，提供了面向 Standard Servlet、Spring MVC、Http Client、JAX RS、Jersey、Resteasy 和 MySQL 等接口的装备能力，可以通过编写简单的配置和代码，让基于这些框架构建的应用可以向 Zipkin 报告数据。同时 Brave 也提供了非常简单且标准化的接口，在以上封装无法满足要求的时候可以方便扩展与定制。

2.2、初始化

Brave 的初始化就是要构建 Brave 类的实例，该库提供了 Builder 类用来完成这件事情。