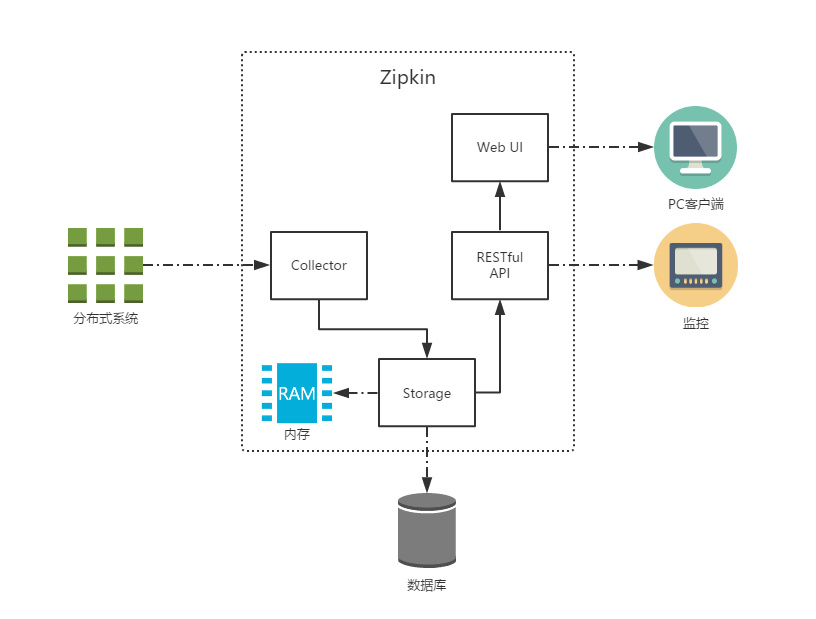
OpenZipkin的使用与源码分析

Zipkin是Twitter的开源项目，基于Google Dapper的实现，可以使用它来收集各个服务器上请求链路的跟踪数据，并通过它提供的REST API接口来辅助查询跟踪数据以实现对分布式系统的监控数据，从而及时地发现系统中出现的延迟升高问题并找出系统性能瓶颈的根源。Zipkin还提供了UI组件来直观的搜索跟踪信息和分析请求链路明细，比如可以查询某段时间内各用户请求的处理时间等。

# Zipkin的基础架构

Zipkin核心组件构成如下图：



1. Collector，收集器组件，主要用于处理从外部系统发送过来的跟踪信息，将这些信息转换为Zipkin内部处理的Span格式，以支持后续的存储、分析和展示等功能
2. Storage，存储组件，主要用于对处理收集器接收到的跟踪信息，默认会将这些信息存储在内存用，可以通过修改存储策略选择其他存储组件
3. RESFul API，API组件，主要用来提供外部访问接口，比如给客户端展示跟踪信息，或者外部系统访问以实现监控等
4. Web UI，UI组件，基于API实现的上层应用。通过UI组件用户可以方便而又直观地查询和分析跟踪信息

## **基本概念**

Zipkin以Trace结构表示对一次请求的追踪，可以把每个Trace拆分成若干有依赖关系的Span。在微服务架构中，一次用户请求可能会由后台若干个服务负责处理，那么每个处理请求的服务可以理解为一个Span(包括API 、缓存、数据库等服务)。处理请求的服务可能继续请求其他的服务，从而形成Spark树形结构，以体现服务间的调用关系。

ZipKin的Span模型是参考Dapper中的Span模型设计，一次RPC调用关联一个SpanId，在Span中主要包含三个数据部分：

* 基础数据，用于跟踪树中节点关联和界面展示，如下表所示：

|  |  |
| --- | --- |
| 字段 | 描述 |
| tracerId | 追踪ID |
| parentId | 父服务调用 |
| spanId | 调用的ID号 |
| name | 调用名 |
| timestamp | 调用开始时间 |
| duration | 调用总耗时 |

其中parentId为null的Span将成为跟踪树的根节点，也是调用链的起点，示例如下：

*{*

*"traceId": "bd7a977555f6b982",*

*"name": "query",*

*"id": "be2d01e33cc78d97",*

*"parentId": "ebf33e1a81dc6f71",*

*"timestamp": 1458702548786000,*

*"duration": 13000*

*}*

* Annotation数据，用来记录关键事件，包括以下四种：

|  |  |  |
| --- | --- | --- |
| 缩写 | 全称 | 说明 |
| CS | Client send | 客户端/消费者发起请求 |
| CR | Client receive | 客户端/消费者接收到应答 |
| SR | Server Receive | 服务端/生产者接收到请求 |
| SS | Server Send | 服务端/生产者发送应答 |

在Span模型中，Annotation是一个列表，每个关键事件包括：

|  |  |
| --- | --- |
| 字段 | 描述 |
| value | 以上四种关键事件的一种 |
| timestamp | 发生的时间 |
| endpoint | 用于记录发生的机器ip和服务名称 |

示例如下：

{

*"annotations": [{*

*"endpoint": {*

*"serviceName": "zipkin-query",*

*"ipv4": "192.168.1.2",*

*"port": 9411*

*},*

*"timestamp": 1458702548786000,*

*"value": "cs"*

*},*

*{*

*"endpoint": {*

*"serviceName": "zipkin-query",*

*"ipv4": "192.168.1.2",*

*"port": 9411*

*},*

*"timestamp": 1458702548799000,*

*"value": "cr"*

}

]

}

* BinaryAnnotation数据，跟踪树中不仅展示调用链时间信息，还需要绑定一些业务数据，这部分数据写入BinaryAnnotation中，结构和Annotation数据相同，示例如下：

*"binaryAnnotations": [*

*{*

*"key": "jdbc.query",*

*"value": "select distinct.....",*

*"endpoint": {*

*"serviceName": "zipkin-query",*

*"ipv4": "192.168.1.2",*

*"port": 9411*

*}*

*},*

*{*

*"key": "sa",*

*"value": true,*

*"endpoint": {*

*"serviceName": "spanstore-jdbc",*

*"ipv4": "127.0.0.1",*

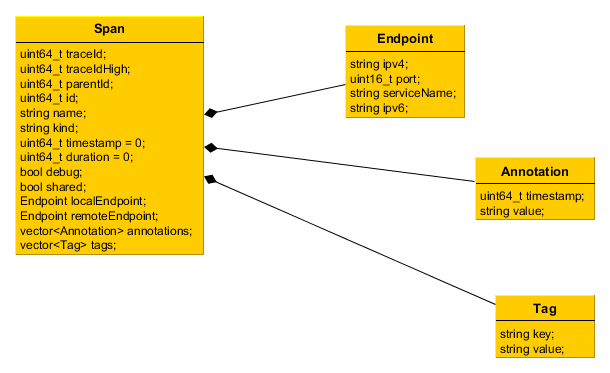
*"port": 3306*

*}*

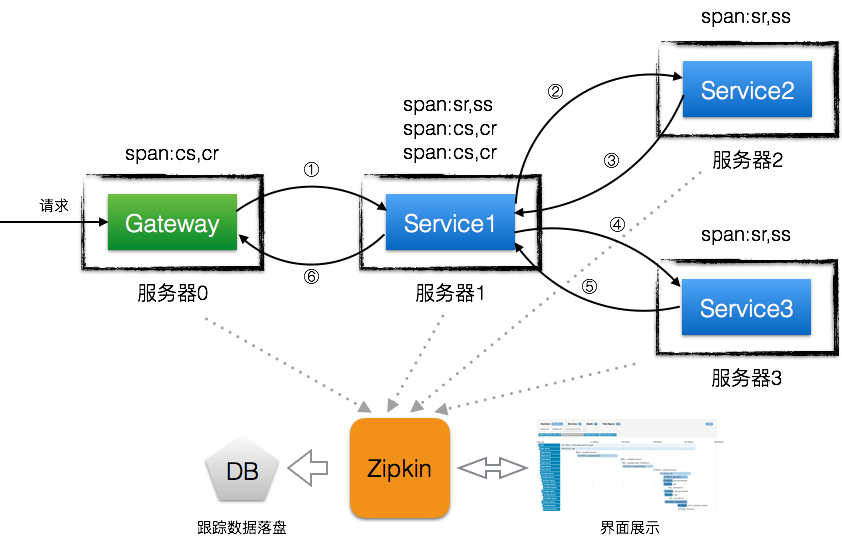
*}*

*]*

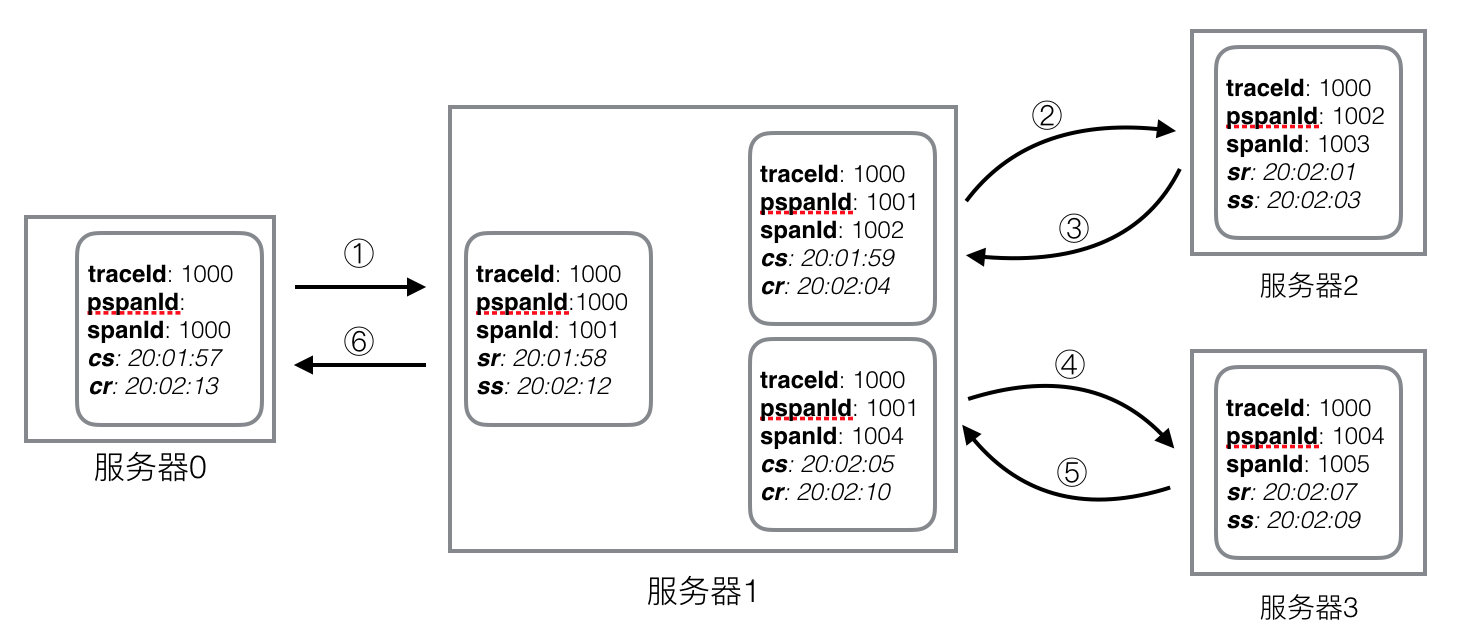
Span的核心数据结构如下：



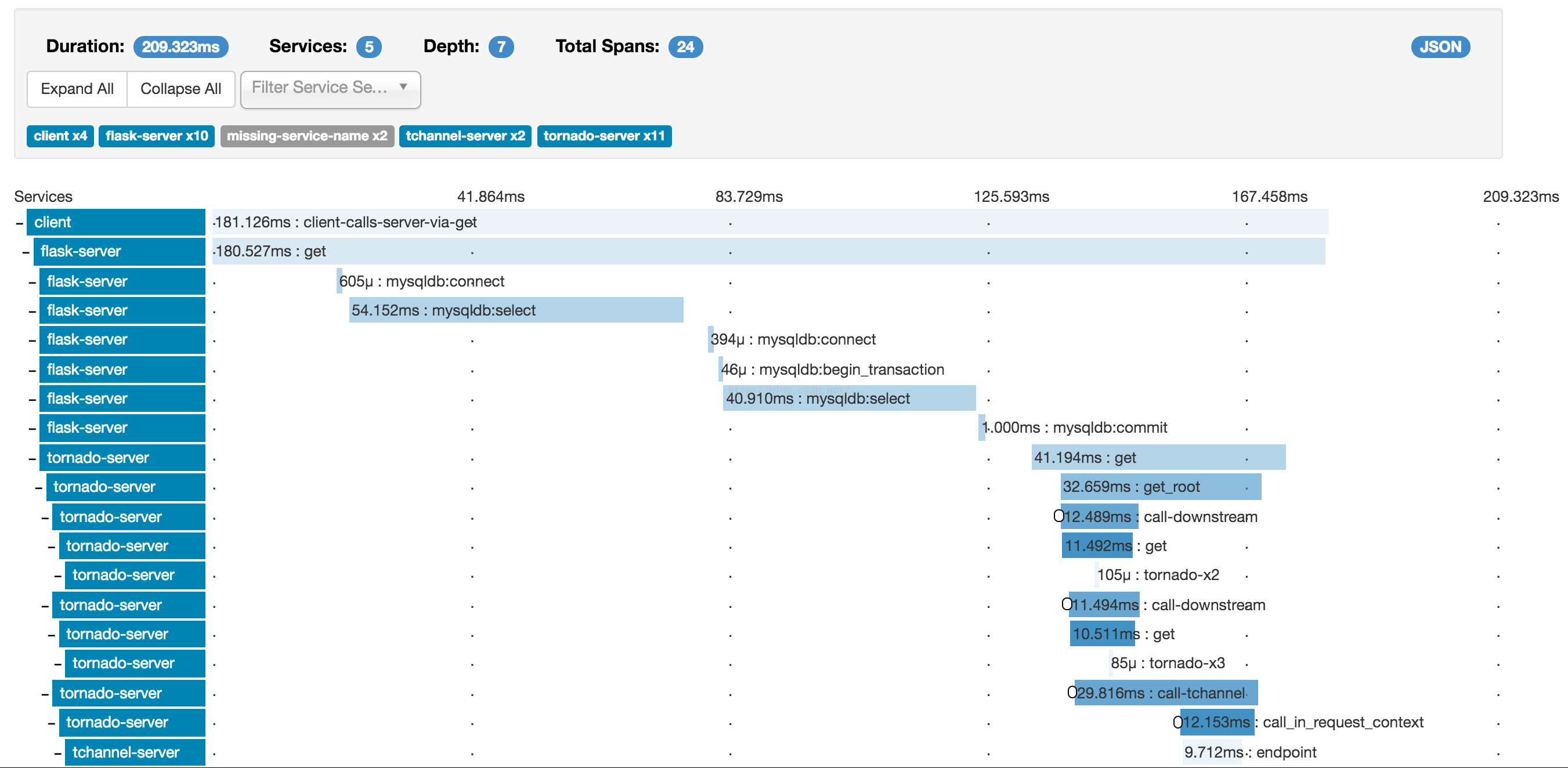
以上是Span的内部结构，Zip服务端将收集到的具有相同traceId和spanId的Span组装正Span追踪树，如下图调用关系：



形成的Span调用关系图如下：

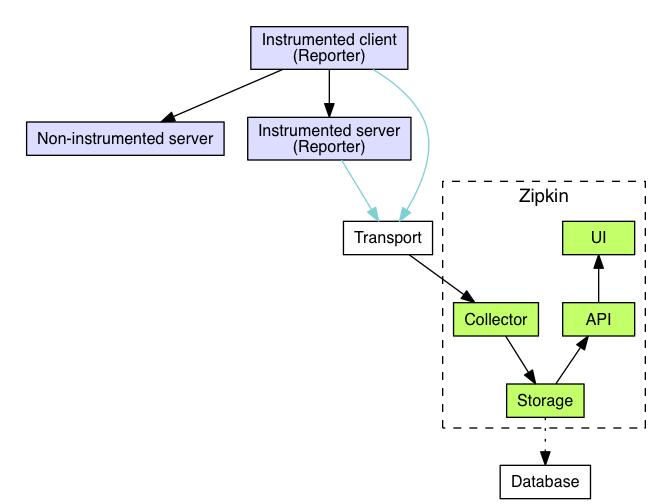


上图没有显示Span所有细节，比如Name和binaryAnnotation等。下图是通过Zipkin展示的跟踪树界面：



## **执行流程**

在分布式系统中Zipkin Tracers记录其所进行的操作，将其封装成Span，然后发送给Zipkin Server中的Collector，Reporter负责操作数据的发送。Ziplin Server接收到数据后将其持久化存储，并提供UI来显示数据，其执行流程如下图所示：



分布式跟踪系统数据流分为以下步骤：

1. 数据采集

通过Instrumentation来实现数据的采集，RPC框架有不同的实现，对于Java提供Brave库，提供了面向Standard Servlet/Http Client/Jersey等接口的Instrument的能力，编写简单的配置和代码，就可以让基于这些框架的应用向Zipkin报告数据。

2）数据的发送

Reporter负责将数据发送给Zipkin Collector，发送通过Transport来实现，目前主要包括Http/Kafka和Scribe三种主要的实现。

3）数据的持久化存储

Zipkin Collector接收到数据后，将数据持久化存储，目前支持Cassandra、MySQL及ElasticSearch等原生模式。

4）数据查询及Web UI

数据存储后，Zipkin提供JSON API来进行查询，目前主要是Web UI来使用，并将数据展示给用户。

# **Zipkin的安装和使用**

在Spring Cloud Sleuth中对Zipkin的整合进行了自动化配置的封装，主要为分为两步搭建Zipkin Server和为应用引入和配置Zipkin服务。

## **搭建Zipkin Server**

1）添加依赖

创建一个基础的Spring Boot应用，命名为Zipkin-Server，在pom.xml中引入Zipkin Server的相关依赖，具体如下：

*<parent>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-parent</artifactId>*

*<version>1.5.10.RELEASE</version>*

*<relativePath/>*

*</parent>*

*<dependencies>*

*<dependency>*

*<groupId>io.zipkin.java</groupId>*

*<artifactId>zipkin-server</artifactId>*

*</dependency>*

*<dependency>*

*<groupId>io.zipkin.java</groupId>*

*<artifactId>zipkin-autoconfigure-ui</artifactId>*

*</dependency>*

*</dependencies>*

*<dependencyManagement>*

*<dependencies>*

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-dependencies</artifactId>*

*<version>Dalston.SR5</version>*

*<type>pom</type>*

*<scope>import</scope>*

*</dependency>*

*</dependencies>*

*</dependencyManagement>*

2）创建应用主类

创建应用主类Zipkin App，使用@EnableZipServer注解来启动Zipkin Server，具体如下：

*@EnableZipkinServer*

*@SpringBootApplication*

*public class ZipkinApplication {*

*public static void main(String[] args) {*

*SpringApplication.run(ZipkinApplication.class, args);*

*}*

*}*

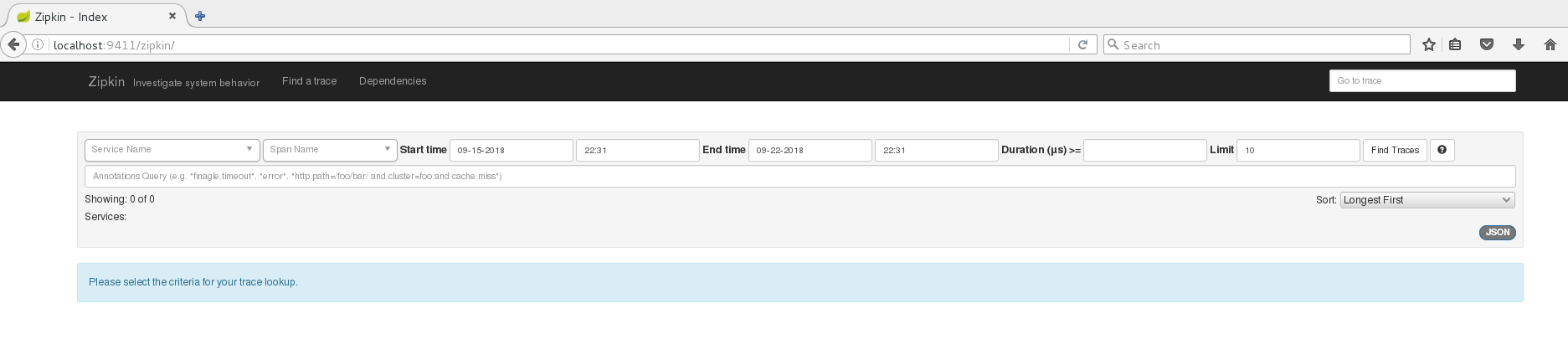
3）配置主类

在application.properties做一些简单的配置，比如设置服务端口：9411

*spring.application.name=zipkin-server*

*server.port=9411*

启动后，访问http://localhost:9411/，可以看到Zipkin管理页面



在Zipkin Web UI中包含两个主要功能：

1. Find a trace，查询相关的Trace
2. Dependencies，查看每个组件之间的调用依赖关系图

此时在Web UI中看不到任何Trace，这里还没有通过任何的Zipkin客户端产生相关的跟踪数据，随后可以使用Zipkin的Java客户端Brave来收集并推送跟踪数据，并将Brave集成到Spring Boot应用程序中。

## **应用引入和配置Zipkin服务**

完成Zipkin Server搭建之后，需要对应用做一些配置，以实现将跟踪信息输出到Zipkin Server中。这里定义3个微服务，让它们形成调用链关系：

* foo服务，通过REST API方式来调用bar服务
* bar服务，通过REST API方式调用hello服务
* hello服务，输出一个hello字符串

在程序中使用Spring自带的RestTemplate来实现REST API调用，首先在Spring Context中创建RestTemplate的Spring Bean，然后将其注入到相应的Controller中即可。

1. 定义zipkin Common模块，实现类：

*@Configuration*

*@Import({RestTemplate.class, BraveClientHttpRequestInterceptor.class, ServletHandlerInterceptor.class})*

*public class ZipkinConfiguration extends WebMvcConfigurerAdapter {*

*@Autowired*

*private ZipkinProperties zipkinProperties;*

*@Bean*

*public Sender sender(){*

*return OkHttpSender.create(zipkinProperties.getEndpoint());*

*}*

*@Bean*

*public Reporter<Span> reporter() {*

*return AsyncReporter.builder(sender()).build();*

*}*

*@Bean*

*public Brave brave() {*

*return new Brave.Builder(zipkinProperties.getService()).reporter(*

*reporter()).build();*

*}*

*@Bean*

*public SpanNameProvider spanNameProvider() {*

*return new SpanNameProvider() {*

*@Override*

*public String spanName(HttpRequest httpRequest) {*

*return String.format(*

*"%s %s" ,*

*httpRequest.getHttpMethod(),*

*httpRequest.getUri().getPath()*

*);*

*}*

*};*

*}*

*@Autowired*

*private RestTemplate restTemplate;*

*@Autowired*

*private BraveClientHttpRequestInterceptor clientInterceptor;*

*@Autowired*

*private ServletHandlerInterceptor serverInterceptor;*

*@PostConstruct*

*public void init() {*

*List<ClientHttpRequestInterceptor> interceptors = restTemplate.getInterceptors();*

*interceptors.add(clientInterceptor);*

*restTemplate.setInterceptors(interceptors);*

*}*

*@Override*

*public void addInterceptors(InterceptorRegistry registry) {*

*registry.addInterceptor(serverInterceptor);*

*}*

*}*

定义ZipkinConfiguration类

*@Component*

*@ConfigurationProperties("zipkin")*

*@Getter*

*@Setter*

*public class ZipkinProperties {*

*private String endpoint;*

*private String service;*

*}*

1. 定义Foo服务，创建Spring Boot应用程序，代码如下：

*@SpringBootApplication(scanBasePackages = "com.fys.zipkin")*

*public class FooApplication {*

*public static void main(String[] args) {*

*SpringApplication.run(FooApplication.class);*

*}*

*@Bean*

*public RestTemplate restTemplate() {*

*return new RestTemplate();*

*}*

*}*

定义FooController

*@RestController*

*public class FooController {*

*@Autowired*

*private RestTemplate restTemplate;*

*@GetMapping("/foo")*

*public String foo() {*

*try {*

*Thread.sleep(new Random().nextInt(1000));*

*} catch (InterruptedException e) {*

*e.printStackTrace();*

*}*

*//调用bar服务*

*return restTemplate.getForObject("http://localhost:8082/bar",String.class);*

*}*

*}*

1. 定义Bar服务

*@SpringBootApplication(scanBasePackages = "com.fys.zipkin")*

*public class BarApplication {*

*public static void main(String[] args) {*

*SpringApplication.run(BarApplication.class);*

*}*

*@Bean*

*public RestTemplate restTemplate() {*

*return new RestTemplate();*

*}*

*}*

定义BarController

*@RestController*

*public class BarController {*

*@Autowired*

*private RestTemplate restTemplate;*

*@GetMapping("/bar")*

*public String bar() {*

*try {*

*Thread.sleep(new Random().nextInt(1000));*

*} catch (InterruptedException e) {*

*e.printStackTrace();*

*} //调用Hello服务*

*return restTemplate.getForObject("http://localhost:8083/hello",String.class);*

*}*

*}*

1. 定义Hello服务

*@SpringBootApplication(scanBasePackages = "com.fys.zipkin")*

*public class HelloApplication {*

*public static void main(String[] args) {*

*SpringApplication.run(HelloApplication.class);*

*}*

*@Bean*

*public RestTemplate restTemplate() {*

*return new RestTemplate();*

*}*

*}*

创建HelloController

*@RestController*

*public class HelloController {*

*@Autowired*

*RestTemplate restTemplate;*

*@GetMapping("/hello")*

*public String hello() {*

*try {*

*Thread.sleep(new Random().nextInt(1000));*

*} catch (InterruptedException e) {*

*e.printStackTrace();*

*}*

*return "hello";*

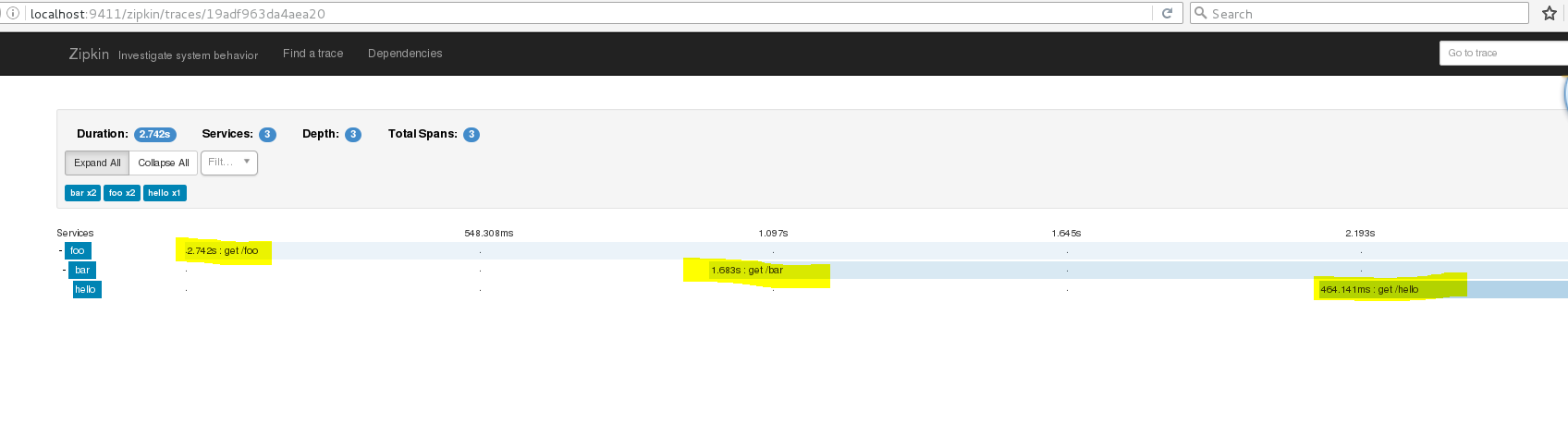
*}*

*}*

以上服务以来zipkin-common模块

## **Zipkin服务测试**

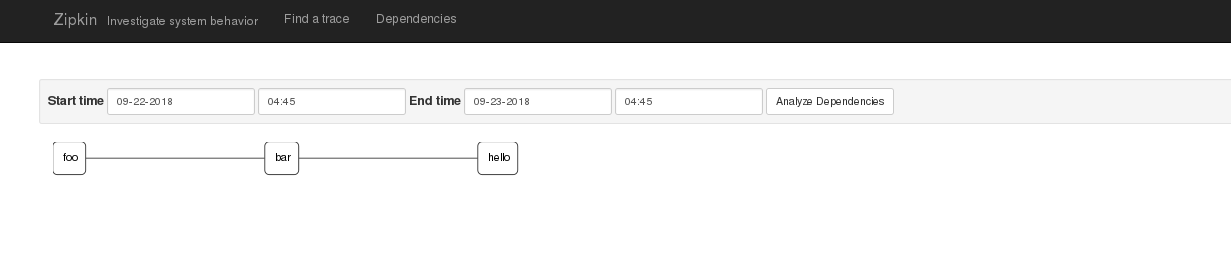
分别启动Foo/Bar/Hello服务，在客户端调用http://localhost:8081/foo，查看Zipkin Server信息如下：



可以看到调用属顺序如下：

*foo -> bar -> hello*

其耗费时间如上图所示。



调用关系的JSON如下：



https://www.cnblogs.com/chenweida/p/9025522.html

https://blog.csdn.net/gaowenhui2008/article/details/70231239

http://www.tangrui.net/2016/zipkin-vs-pinpoint.html