

# Java分布式跟踪系统Zipkin（一）： 初识Zipkin

原创 v墨竹v 最后发布于2017-12-05 16:51:31 阅读数 7942 ☆ 收藏

所有博文均在个人独立博客<http://blog.mozhu.org>首发，欢迎访问！

在2010年，谷歌发表了其内部使用的分布式跟踪系统Dapper的论文，讲述了Dapper在谷歌内部两年的演变和设计、运维经验。Twitter也根据该论文开发了自布式跟踪系统Zipkin，并将其开源。

论文地址：<http://static.googleusercontent.com/media/research.google.com/zh-CN/archive/papers/dapper-2010-1.pdf>

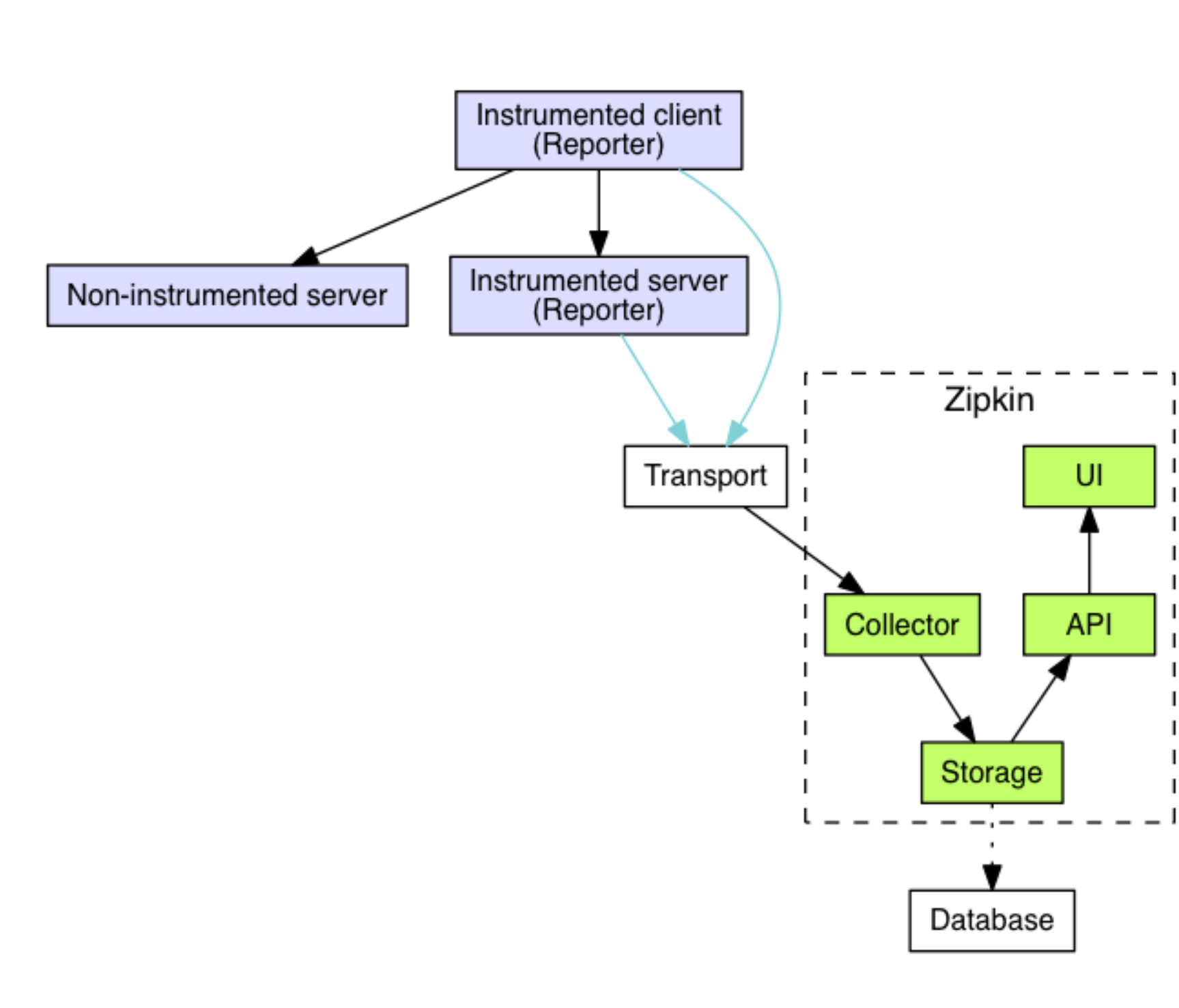
译文地址：<http://bigbully.github.io/Dapper-translation/>

分布式跟踪系统还有其他比较成熟的实现，例如：Naver的Pinpoint、Apache的HTrace、阿里的鹰眼Tracing、京东的Hydra、新浪的Watchman，美团点评的skywalking等。

本系列博文，主要以Zipkin为主，介绍Zipkin的基本使用，原理，以及部分核心源代码的分析，当前Zipkin版本为2.2.1

## 概述

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。其主要功能来自各个异构系统的实时监控数据。



如上图所示，各业务系统在彼此调用时，将特定的跟踪消息传递至zipkin，zipkin在收集到跟踪信息后将其聚合处理、存储、展示等，用户可通过web UI方便络延迟、调用链路、系统依赖等等。

Zipkin主要包括四个模块

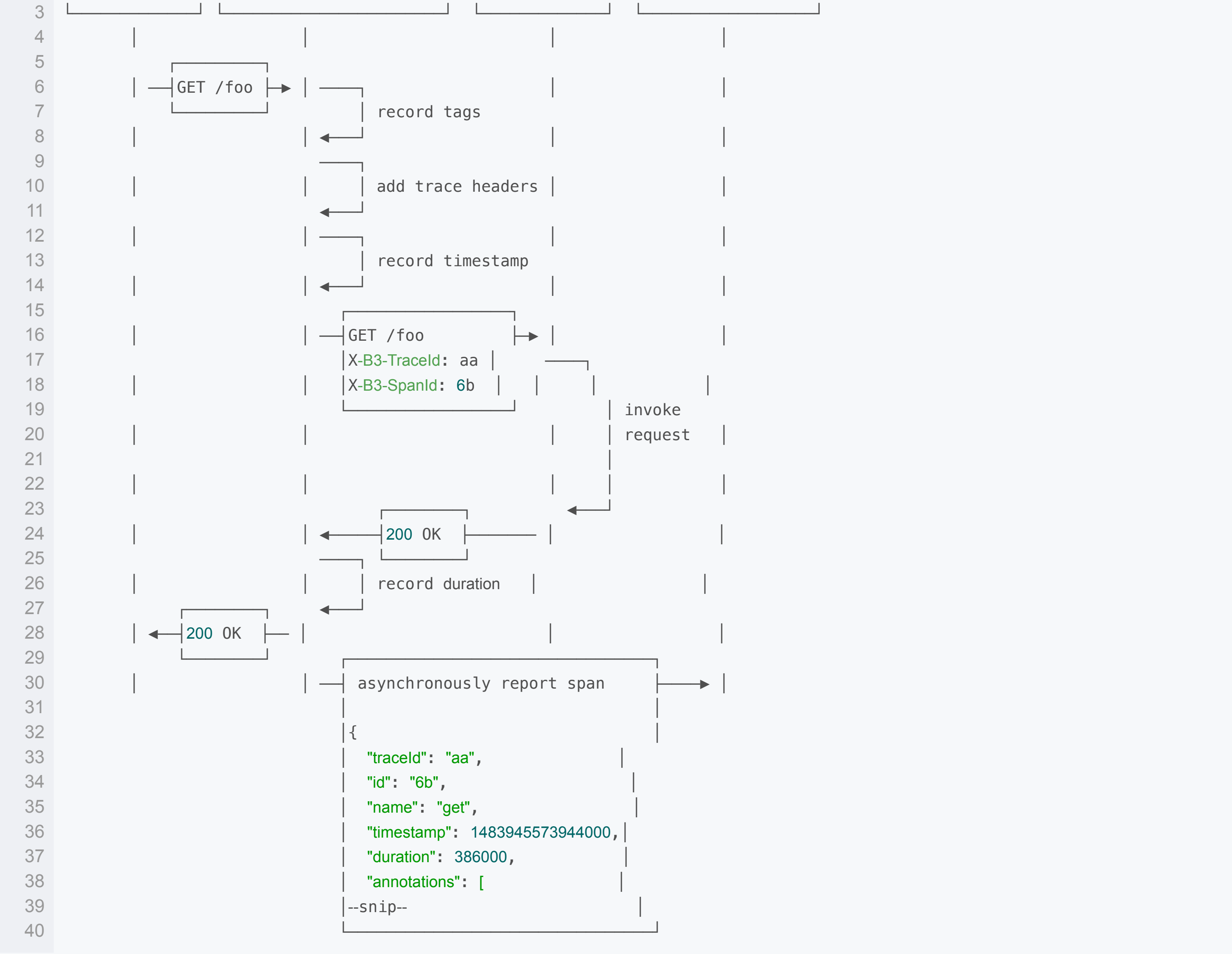
- **Collector** 接收或收集各应用传输的数据
- **Storage** 存储接受或收集过来的数据，当前支持Memory，MySQL，Cassandra，ElasticSearch等，默认存储在内存中。
- **API（Query）** 负责查询Storage中存储的数据，提供简单的JSON API获取数据，主要提供给web UI使用
- **Web** 提供简单的web界面

Instrumented Client 和Instrumented Server，是指分布式架构中使用了Trace工具的两个应用，Client会调用Server提供的服务，两者都会向Zipkin上报Trace信息。在Client 和 Server通过Transport上报Trace信息后，由Zipkin的Collector模块接收，并由Storage模块将数据存储在对应的存储介质中，然后Zipkin提供A界面查询Trace跟踪信息。

Non-Instrumented Server，指的是未使用Trace工具的Server，显然它不会上报Trace信息。

## 流程图





由上图可以看出，应用的代码（User Code）发起Http Get请求（请求路径/foo），经过Trace框架（Trace Instrumentation）拦截，并依次经过如下步骤，记录信息到Zipkin中：

1. 记录tags信息
2. 将当前调用链的Trace信息记录到Http Headers中
3. 记录当前调用的时间戳（timestamp）
4. 发送http请求，并携带Trace相关的Header，如X-B3-TraceId:aa，X-B3-SpanId:6b
5. 调用结束后，记录当次调用所花的时间（duration）
6. 将步骤1-5，汇总成一个Span（最小的Trace单元），异步上报该Span信息给Zipkin Collector

## Zipkin的几个基本概念

**Span**：基本工作单元，一次链路调用（可以是RPC，DB等没有特定的限制）创建一个span，通过一个64位ID标识它，span通过还有其他的数据，例如描述时间戳，key-value对的（Annotation）tag信息，parent-id等，其中parent-id 可以表示span调用链路来源，通俗的理解span就是一次请求信息

**Trace**：类似于树结构的Span集合，表示一条调用链路，存在唯一标识，即TraceId

**Annotation**：注解，用来记录请求特定事件相关信息（例如时间），通常包含四个注解信息

- **cs** - Client Start，表示客户端发起请求
- **sr** - Server Receive，表示服务端收到请求
- **ss** - Server Send，表示服务端完成处理，并将结果发送给客户端
- **cr** - Client Received，表示客户端获取到服务端返回信息

**BinaryAnnotation**：提供一些额外信息，一般以key-value对出现

## 安装

本系列博文使用的Zipkin版本为2.2.1，所需JDK为1.8

下载最新的Zipkin的jar包，并运行

```
1 wget -O zipkin.jar 'https://search.maven.org/remote_content?g=io.zipkin.java&a=zipkin-server&v=LATEST&c=exec'
2 java -jar zipkin.jar
```

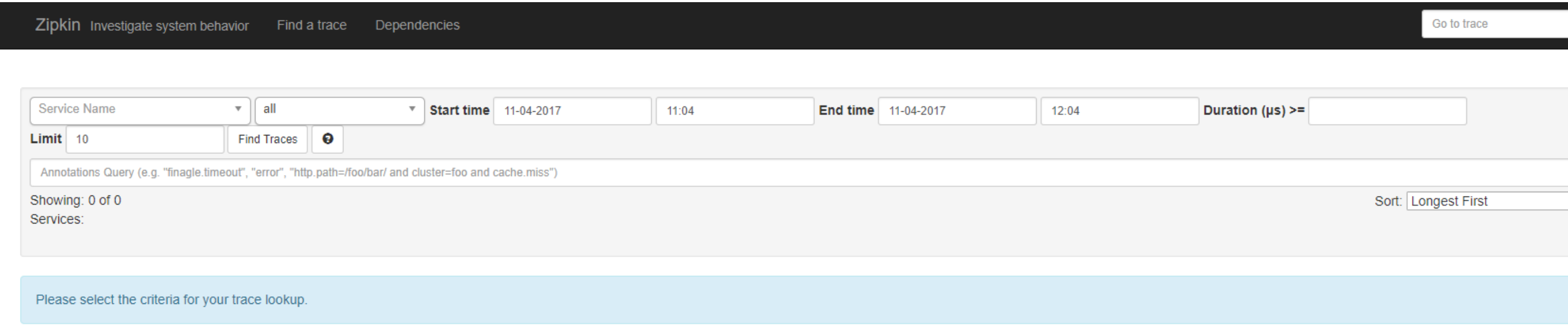
还可以使用docker， 具体操作请参考：

<https://github.com/openzipkin/docker-zipkin>

启动成功后浏览器访问

<http://localhost:9411/>

打开Zipkin的Web UI界面



下面用一个简单的Web应用来演示如何向Zipkin上报追踪数据

代码地址：<https://gitee.com/mozhu/zipkin-learning>

在Chapter1/servlet25中，演示了如何在传统的Servlet项目中使用Brave框架，向Zipkin上传Trace数据

分别运行

```
1 mvn jetty:run -Pbackend
```

```
1 mvn jetty:run -Pfrontend
```

则会启动两个端口为8081和9000的服务，Frontend会发送请求到Backend，Backend返回当前时间

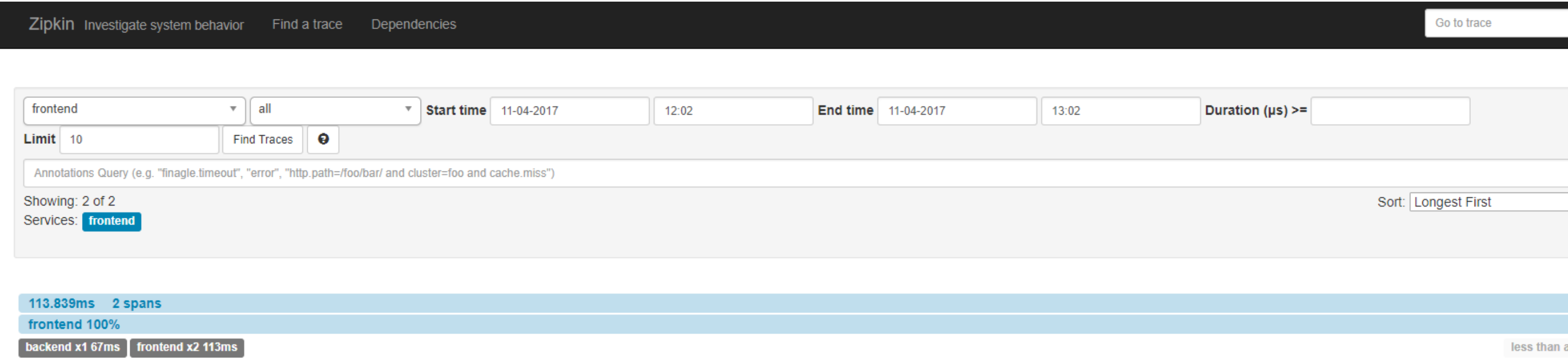
Frontend: <http://localhost:8081/>

Backend: <http://localhost:9000/api>

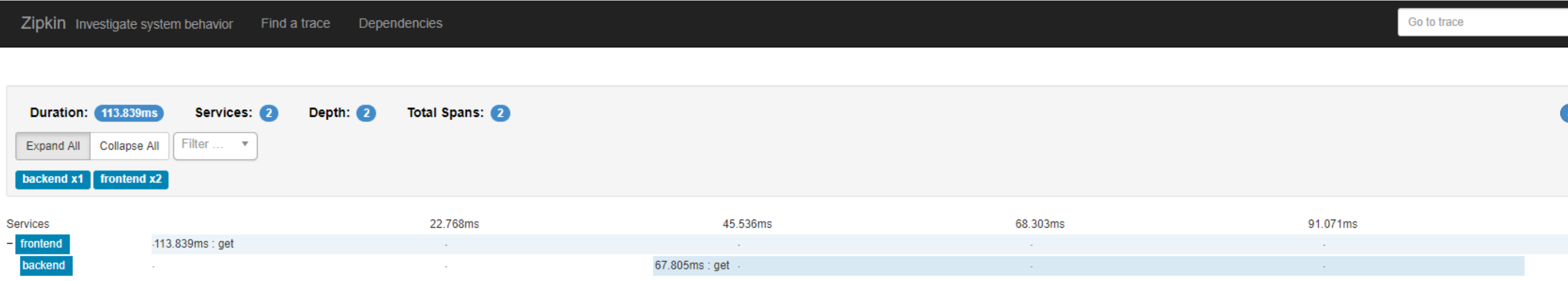
浏览器访问 <http://localhost:8081/> 会显示当前时间

Fri Nov 03 18:43:00 GMT+08:00 2017

打开Zipkin Web UI界面，点击 Find Traces，显示如下界面：

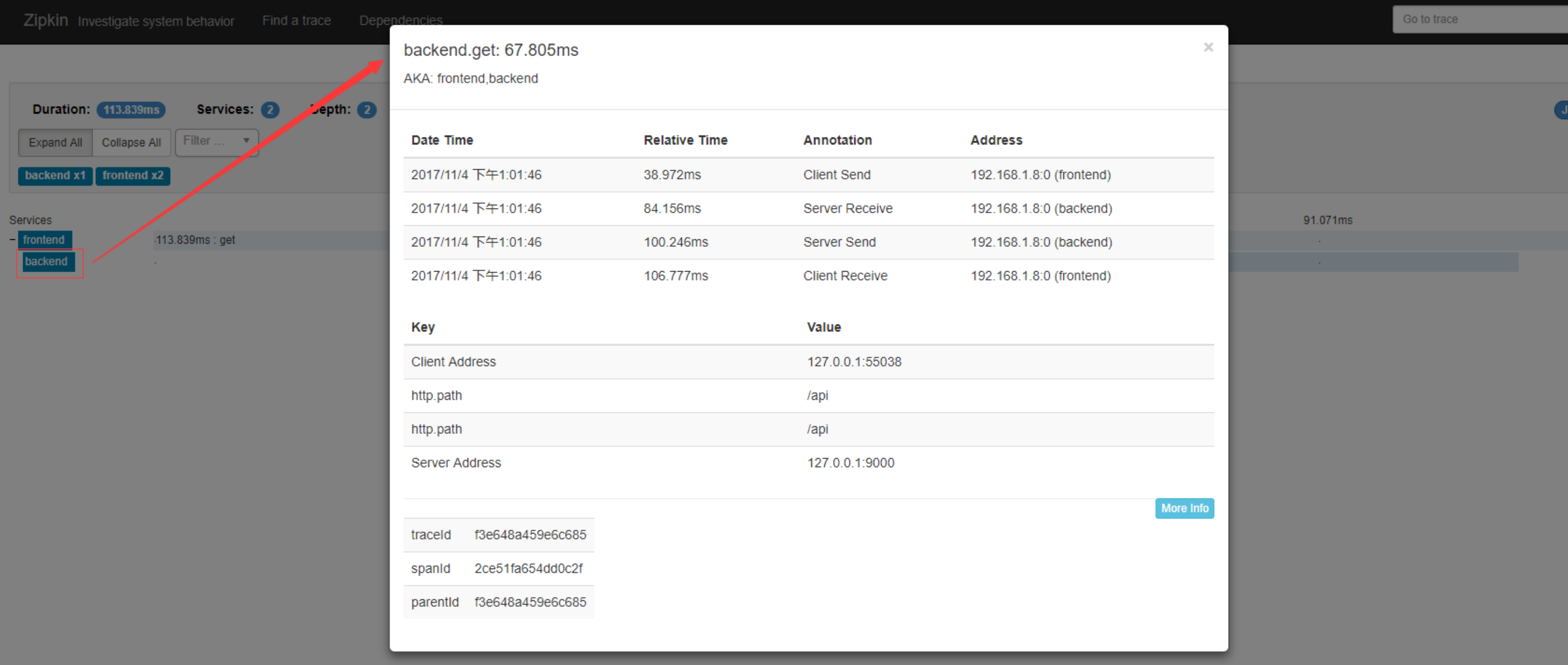
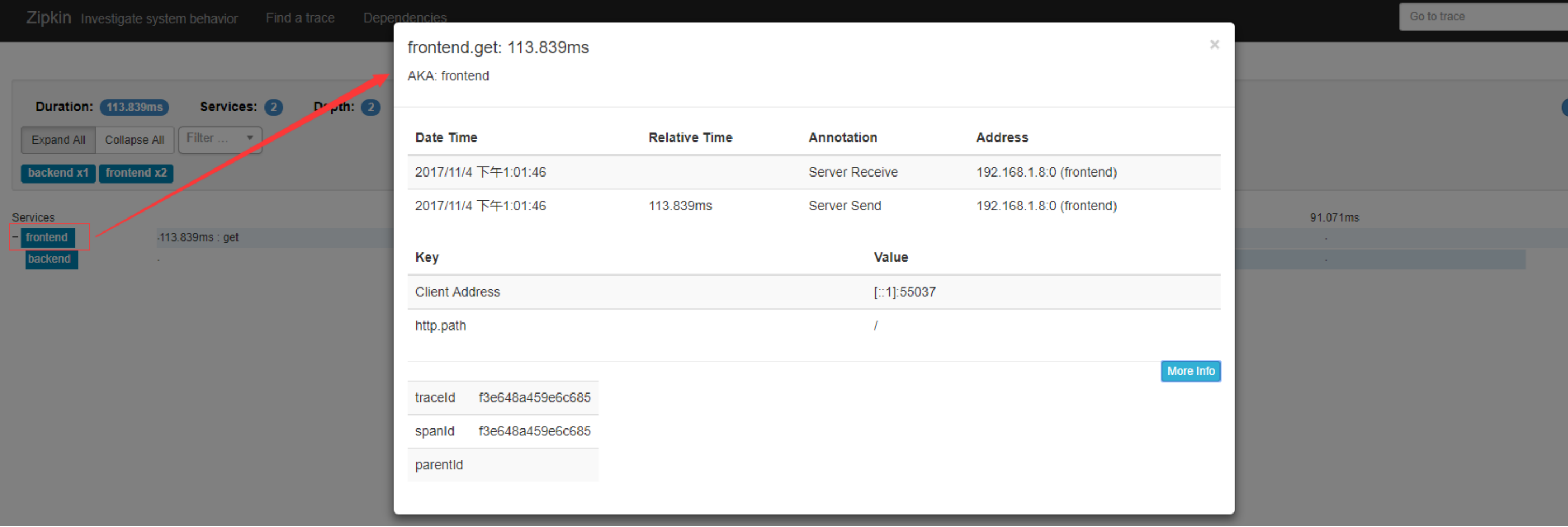


继续点击，查看详情，界面如下：



可以看到Frontend调用Backend的跟踪链信息，Frontend整个过程耗时113.839ms，其中调用Backend服务耗时67.805ms

点击左侧跟踪栈的frontend和backend，分别打开每条跟踪栈的详细信息



点击页面右上角的JSON，可以看到该Trace的所有数据

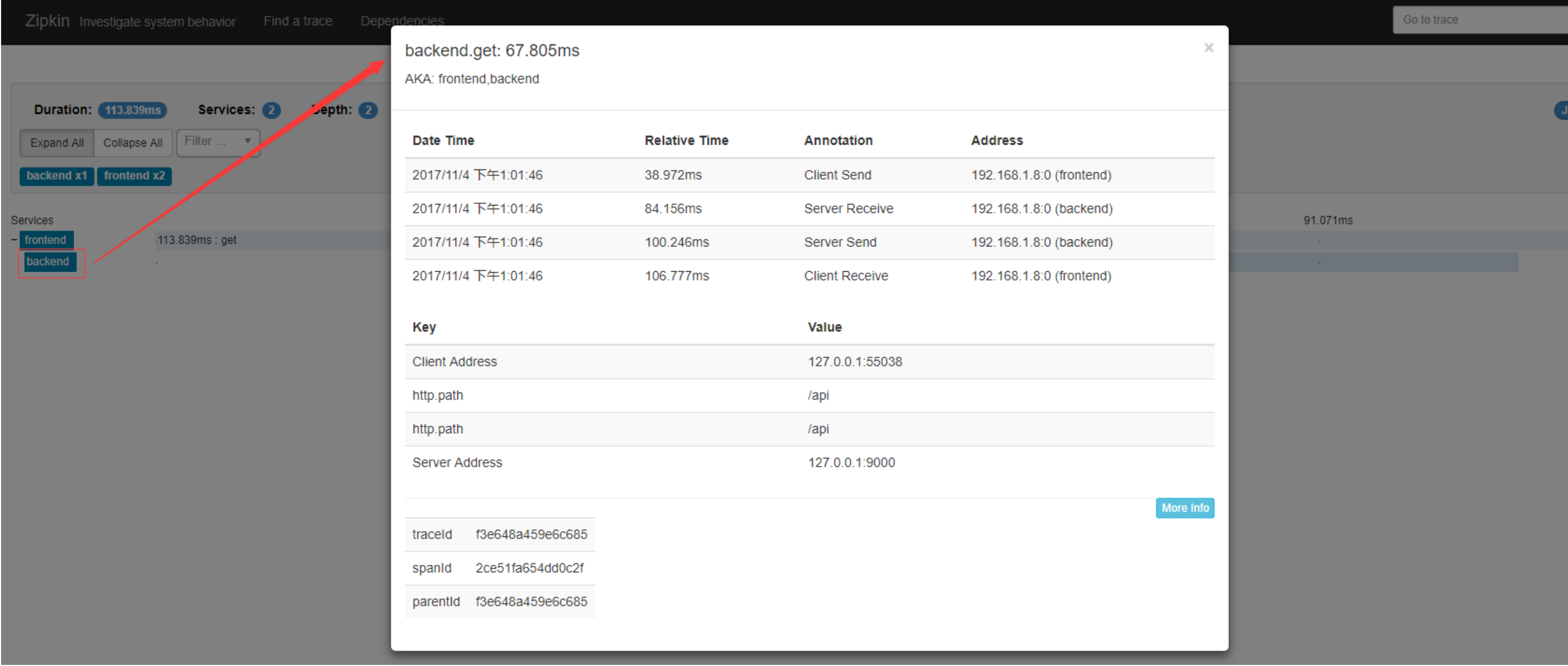
```
1 [
2   {
3     "traceId": "f3e648a459e6c685",
4     "id": "f3e648a459e6c685",
5     "name": "get",
6     "timestamp": 1509771706395235,
7     "duration": 113839,
8     "annotations": [
```



```
9      {
10        "timestamp": 1509771706395235,
11        "value": "sr",
12        "endpoint": {
13          "serviceName": "frontend",
14          "ipv4": "192.168.1.8"
15        }
16      },
17      {
18        "timestamp": 1509771706509074,
19        "value": "ss",
20        "endpoint": {
21          "serviceName": "frontend",
22          "ipv4": "192.168.1.8"
23        }
24      }
25    ],
26    "binaryAnnotations": [
27      {
28        "key": "ca",
29        "value": true,
30        "endpoint": {
31          "serviceName": "",
32          "ipv6": "::1",
33          "port": 55037
34        }
35      },
36      {
37        "key": "http.path",
38        "value": "/",
39        "endpoint": {
40          "serviceName": "frontend",
41          "ipv4": "192.168.1.8"
42        }
43      }
44    ]
45  },
46  {
47    "traceId": "f3e648a459e6c685",
48    "id": "2ce51fa654dd0c2f",
49    "name": "get",
50    "parentId": "f3e648a459e6c685",
51    "timestamp": 1509771706434207,
52    "duration": 67805,
53    "annotations": [
54      {
55        "timestamp": 1509771706434207,
56        "value": "cs",
57        "endpoint": {
58          "serviceName": "frontend",
59          "ipv4": "192.168.1.8"
60        }
61      },
62      {
63        "timestamp": 1509771706479391,
64        "value": "sr",
65        "endpoint": {
66          "serviceName": "backend",
67          "ipv4": "192.168.1.8"
68        }
69      },
70      {
71        "timestamp": 1509771706495481,
72        "value": "ss",
73        "endpoint": {
74          "serviceName": "backend",
75          "ipv4": "192.168.1.8"
76        }
77      },
78      {
79        "timestamp": 1509771706502012,
```

```
80     "value": "cr",
81     "endpoint": {
82       "serviceName": "frontend",
83       "ipv4": "192.168.1.8"
84     }
85   },
86 ],
87   "binaryAnnotations": [
88     {
89       "key": "ca",
90       "value": true,
91       "endpoint": {
92         "serviceName": "",
93         "ipv4": "127.0.0.1",
94         "port": 55038
95       }
96     },
97     {
98       "key": "http.path",
99       "value": "/api",
100       "endpoint": {
101         "serviceName": "frontend",
102         "ipv4": "192.168.1.8"
103       }
104     },
105     {
106       "key": "http.path",
107       "value": "/api",
108       "endpoint": {
109         "serviceName": "backend",
110         "ipv4": "192.168.1.8"
111       }
112     },
113     {
114       "key": "sa",
115       "value": true,
116       "endpoint": {
117         "serviceName": "",
118         "ipv4": "127.0.0.1",
119         "port": 9000
120       }
121     }
122   ]
123 }
124 ]
```

点击Dependencies页面，可以看到下图， frontend和backend的依赖关系图



在复杂的调用链路中假设存在一条调用链路响应缓慢，如何定位其中延迟高的服务呢？  
在使用分布式跟踪系统之前，我们一般只能依次分析调用链路上各个系统中的日志文件，  
而在使用了Zipkin提供的WebUI界面后，我们很容易搜索出一个调用链路中延迟高的服务

后面博文中会详细介绍Zipkin的用法原理，以及和我们现有的系统框架整合。

👍 点赞 1

☆ 收藏

🔗 分享

...



v墨竹v

发布了72 篇原创文章 · 获赞 91 · 访问量 65万+


他的留言板

### 比较完整的mes系统的介绍下载

### mes系统为什么自己开发




想对作者说点什么



赫丙

11个月前

我运行了上面博文的代码，发现用localhost:8080访问，为什么front和back的cmd窗口会分别打印两条日志？




yyxiaoqiang

1年前

作者你好，看到你后面的文章好像有吧代码上传到github 但是没找到，请问有链接吗？谢谢了

查看回复(1)



q3126287

1年前

zipkin启动了，但是我登录ip:9411却页面没有响应

查看回复(1)

Zipkin安装与初识

阅读数 5012

Zipkin Zipkin是一个分布式跟踪系统。它有助于收集用于解决微服务架构中的延迟问题... [博文](#) | 来自: [小小码农](#)

初识zipkin

阅读数 91

简介: Zipkin是一款开源的分布式实时数据追踪系统（Distributed Tracking System）... [博文](#) | 来自: [m0\\_3806...](#)