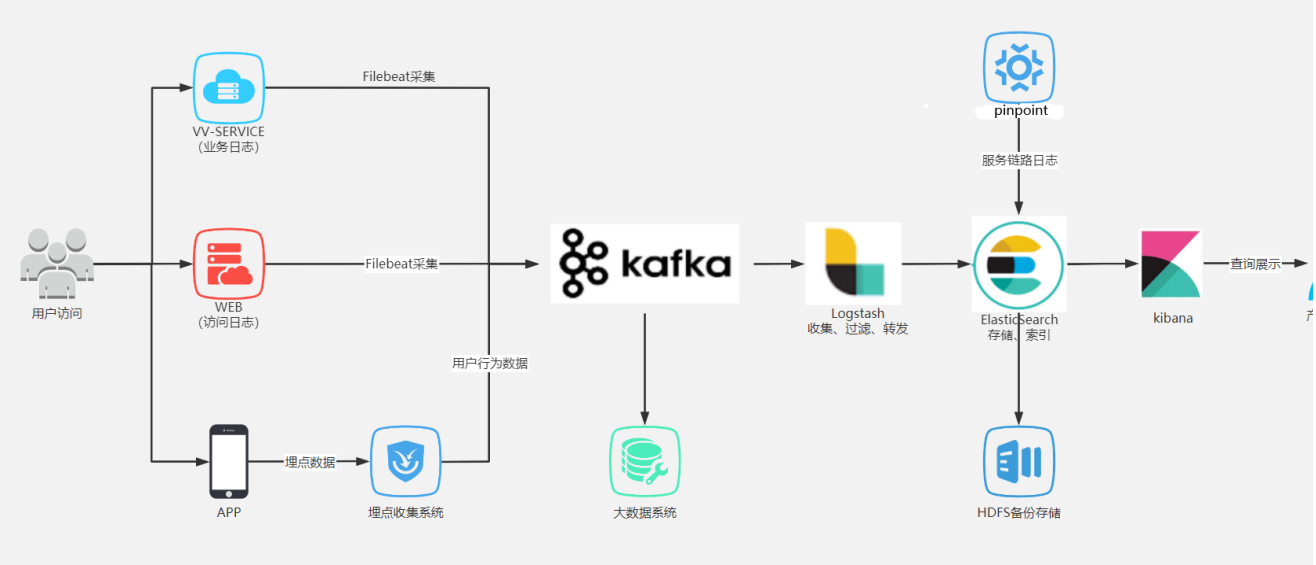
大型互联网公司后端日志收集系统架构



Filebeat是一个日志文件托运工具，在你的服务器上安装客户端后，filebeat会监控日志目录或者指定的日志文件，追踪读取这些文件（追踪文件的变化，不停的读）。

Kafka是一种高吞吐量的分布式发布订阅消息系统，它可以处理消费者规模的网站中的所有动作流数据。

Logstash是一根具备实时数据传输能力的管道，负责将数据信息从管道的输入端传输到管道的输出端；与此同时这根管道还可以让你根据自己的需求在中间加上滤网，Logstash提供里很多功能强大的滤网以满足你的各种应用场景。

ElasticSearch它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。

Kibana是ElasticSearch的用户界面。

在实际应用场景下，为了满足大数据实时检索的场景，利用Filebeat去监控日志文件，将Kafka作为Filebeat的输出端，Kafka实时接收到Filebeat后以Logstash作为输出端输出，到Logstash的数据也许还不是我们想要的格式化或者特定业务的数据，这时可以通过Logstash的一些过滤插件对数据进行过滤最后达到想要的数据格式以ElasticSearch作为输出端输出，数据到ElasticSearch就可以进行丰富的分布式检索了。Kibana可以将ElasticSearch里的数据很好的展示给用户使用。

安装配置ELK日志收集系统

ELK即Elasticsearch、Logstash、Kibana,组合起来可以搭建线上日志系统，本文主要讲解使用ELK来收集SpringBoot应用产生的日志。

- Elasticsearch:用于存储收集到的日志信息；
- Logstash:用于收集日志，SpringBoot应用整合了Logstash以后会把日志发送给Logstash,Logstash再把日志转发给Elasticsearch；
- Kibana:通过Web端的可视化界面来查看日志。

使用Docker Compose 搭建ELK环境，docker compose文件内容如下：

```
version: '3'
services:
 elasticsearch:
 image: elasticsearch:6.4.0
 container_name: elasticsearch
 environment:
 - "cluster.name=elasticsearch" #设置集群名称为elasticsearch
 - "discovery.type=single-node" #以单一节点模式启动
 - "ES_JAVA_OPTS=-Xms512m -Xmx512m" #设置使用jvm内存大小，稍微配置大点，不然有可能启动不成功
 volumes:
 - /mydata/elasticsearch/plugins:/usr/share/elasticsearch/plugins #插件文件挂载
 - /mydata/elasticsearch/data:/usr/share/elasticsearch/data #数据文件挂载
 ports:
 - 9200:9200
 - 9300:9300
 kibana:
 image: kibana:6.4.0
 container_name: kibana
 links: #同一个compose文件管理的服务可以直接用服务名访问，如果要给服务取别名则可以用links实现，如下面的es就是elasticsearch服务的别名
```

```
  - elasticsearch:es #可以用es这个域名访问elasticsearch服务
  depends_on:
  - elasticsearch #kibana在elasticsearch启动之后再启动
  environment:
  - "elasticsearch.hosts=http://es:9200" #设置访问elasticsearch的地址
  ports:
  - 5601:5601
  logstash:
  image: logstash:6.4.0
  container_name: logstash
  volumes:
  - /mydata/logstash/logstash-springboot.conf:/usr/share/logstash/pipeline/logstash.conf #挂载logstash的配置文件，docker对单个文件的挂载需要先在宿主机建好对应文件才能挂载成功
  depends_on:
  - elasticsearch #kibana在elasticsearch启动之后再启动
  links:
  - elasticsearch:es #可以用es这个域名访问elasticsearch服务
  ports:
  - 4560:4560
```

logstash的配置文件logstash-springboot.conf内容如下：

```
input { #logstash直接收集日志的tcp端口
 tcp {
 mode => "server"
 host => "0.0.0.0"
 port => 4560· codec => json_lines
 }
}
output { #logstash将收集到的日志发送到es里存储
 elasticsearch {
  hosts => "es:9200"
  index => "springboot-logstash-%{+YYYY.MM.dd}"
 }
}
```

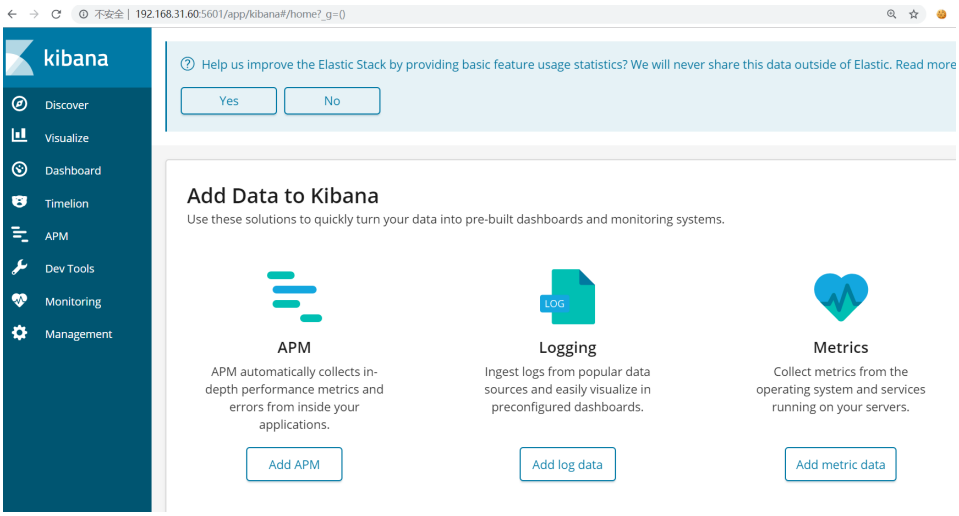
启动ELK，Elasticsearch启动可能需要好几分钟，要耐心等待

```
docker-compose up -d
```

在logstash中安装json_lines插件

```
# 进入logstash容器
docker exec -it logstash /bin/bash
# 进入bin目录
cd /bin/
# 安装插件,有点慢，耐心等下
logstash-plugin install logstash-codec-json_lines
# 退出容器
exit
# 重启logstash服务
 docker restart logstash
 关闭防火墙并在kibana中查看
 systemctl stop firewalld
```

访问kibana地址：http://192.168.50.60:5601



SpringBoot应用集成Logstash

在springboot应用的pom.xml中添加logstash-logback-encoder依赖

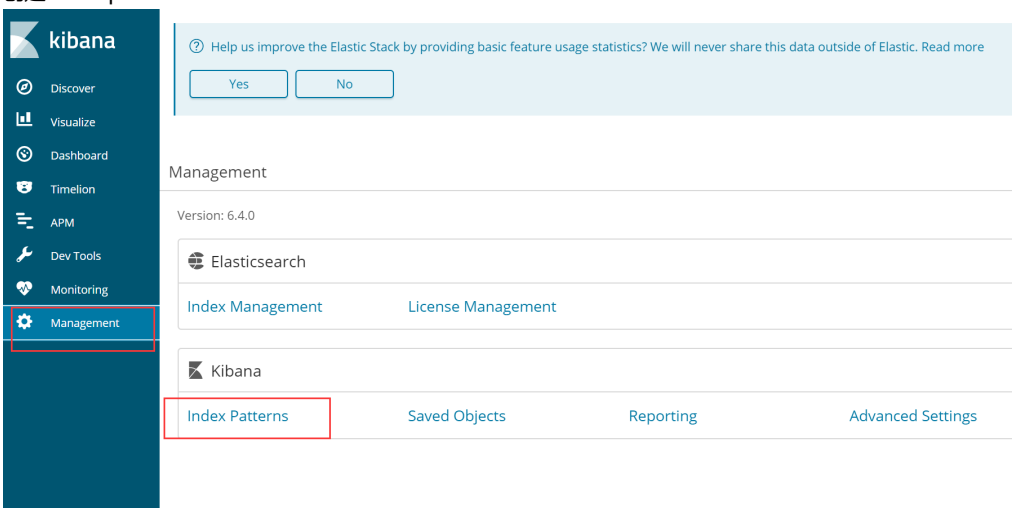
```
<!--集成logstash-->
<dependency>
 <groupId>net.logstash.logback</groupId>
 <artifactId>logstash-logback-encoder</artifactId>
 <version>5.3</version>
</dependency>
```

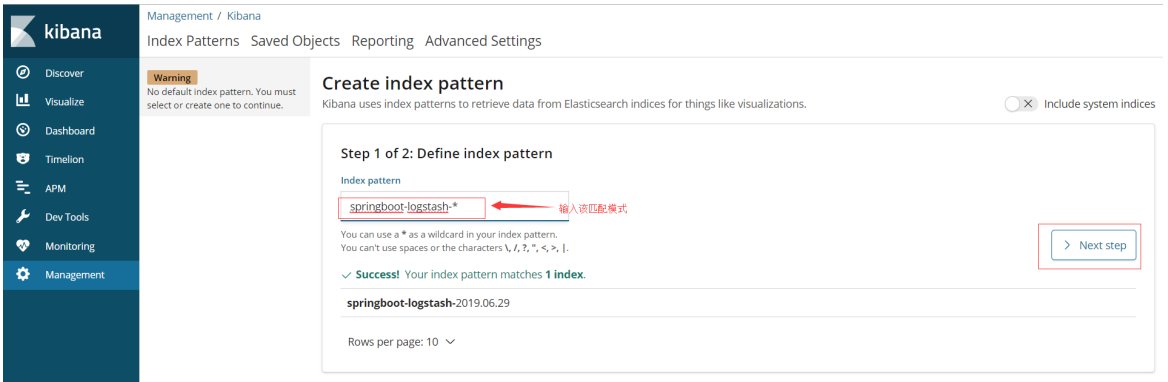
为springboot应用添加配置文件logback-spring.xml让logback的日志输出到logstash，注意appender节点下的destination需要改成你自己的logstash服务地址，logback-spring.xml内容如下：

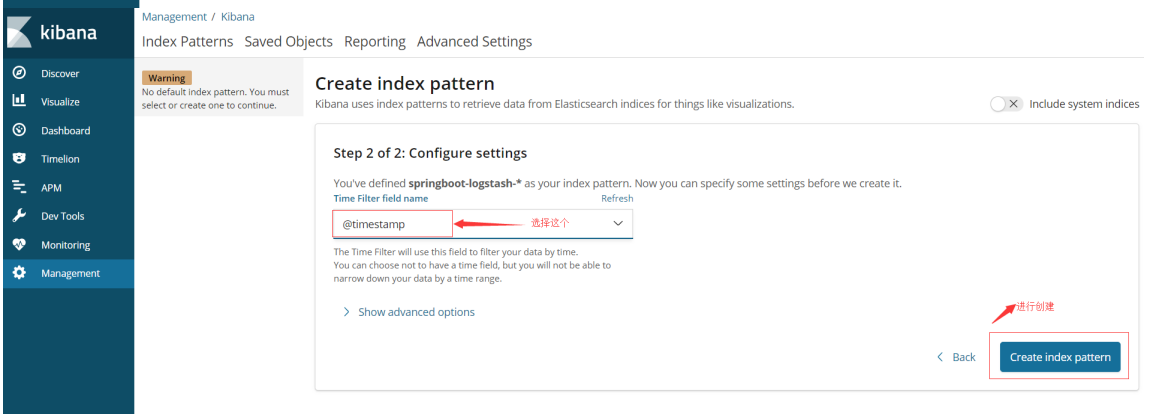
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration>
<configuration>
 <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
 <include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
 <!--应用名称-->
 <property name="APP_NAME" value="tulingmall-order"/>
 <!--日志文件保存路径-->
 <property name="LOG_FILE_PATH" value="${LOG_FILE:-${LOG_PATH:-${LOG_TEMP:-${java.io.tmpdir:-/tmp}}}/logs}"/>
 <contextName>${APP_NAME}</contextName>
 <!--每天记录日志到文件appender-->
 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
 <fileNamePattern>${LOG_FILE_PATH}/${APP_NAME}-%d{yyyy-MM-dd}.log</fileNamePattern>
 <maxHistory>30</maxHistory>
 </rollingPolicy>
 <encoder>
 <pattern>${FILE_LOG_PATTERN}</pattern>
 </encoder>
 </appender>
 <!--输出到logstash的appender-->
 <appender name="LOGSTASH" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
 <!--可以访问的logstash日志收集端口-->
 <destination>172.17.0.1:4560</destination>
 <encoder charset="UTF-8" class="net.logstash.logback.encoder.LogstashEncoder"/>
 </appender>
 <root level="INFO">
 <appender-ref ref="CONSOLE"/>
 <appender-ref ref="FILE"/>
 <appender-ref ref="LOGSTASH"/>
 </root>
</configuration>
```

运行Springboot应用，在kibana中查看日志信息

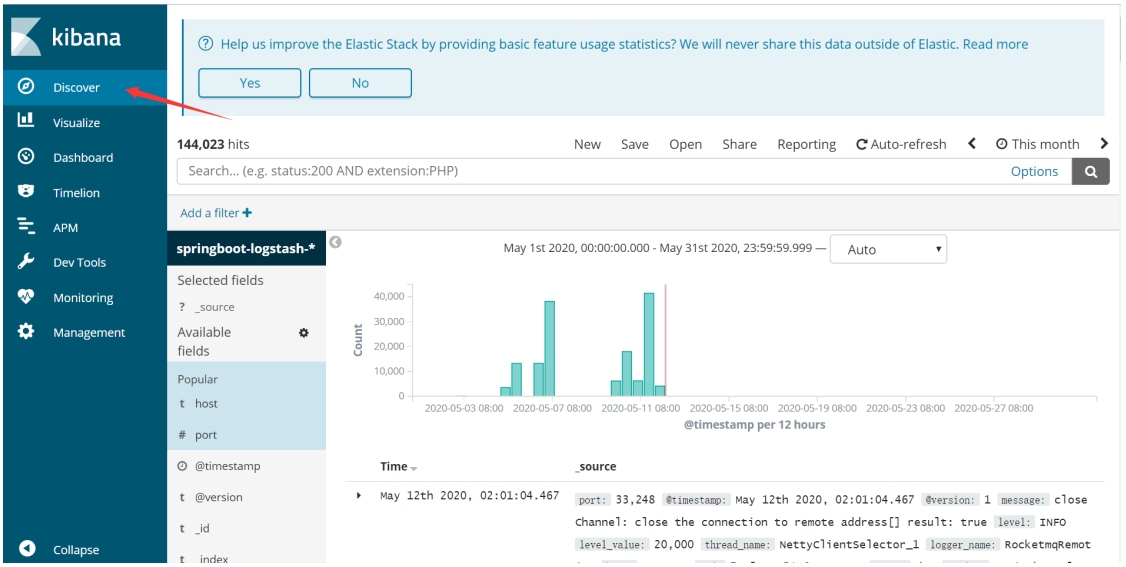
创建index pattern







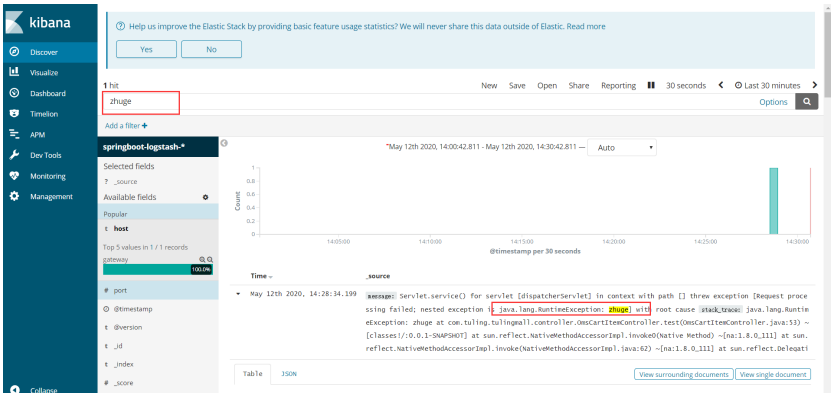
查看收集的日志



调用接口进行测试，制造一个异常并查看

```
@RequestMapping(value = "/test", method = RequestMethod.GET)
@ResponseBody
public CommonResult<List<OmsCartItem>> test(@RequestHeader("memberId") Long memberId)
    throw new RuntimeException("zhuge");
}
```

调用该接口并查看日志



搭建了ELK日志收集系统之后，我们如果要查看SpringBoot应用的日志信息，就不需要查看日志文件了，直接在Kibana中查看即可。

SpringBoot与Logstash中间插入Kafka

按上面的架构如果web应用比较多，都同时往logstash直接发送日志，logstash可能扛不住压力，如果还有一些其他系统也需要收集日志，比如大数据系统，那上面这种架构可能就不是很合适了，所以，我们一般会在web应用和logstash之间加入一层kafka中转。

在docker compose文件里加入如下内容，并启动kafka：

```
kafka:
  image: wurstmeister/kafka
  ports:
  - "9092:9092"
  environment:
  KAFKA_ADVERTISED_HOST_NAME: 192.168.50.60 #宿主机ip
  KAFKA_CREATE_TOPICS: "test:1:1" #主题:分区:副本数
  KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
  volumes:
   - /var/run/docker.sock:/var/run/docker.sock
```

logstash的配置文件logstash-springboot.conf需要对应修改，加入kafka相关配置，如下：

```
input {
# tcp {
# mode => "server"
```

```
# host => "0.0.0.0"
# port => 4560
# codec => json_lines
# }

 kafka {
  id => "kafka_input_id" #随便写
  bootstrap_servers => "192.168.65.60:9092"
  topics => ["test"]
  auto_offset_reset => "latest"
  }
 }
 output {
  elasticsearch {
  hosts => "es:9200"
  index => "springboot-logstash-%{+YYYY.MM.dd}"
  }
 }
```

web应用那边需要新加入一个依赖：

```
<dependency>
  <groupId>com.github.danielwegener</groupId>
  <artifactId>logback-kafka-appender</artifactId>
  <version>0.2.0-RC2</version>
</dependency>
```

web应用下的logback-spring.xml文件内容需要修改，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration>
<configuration>
  <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
  <include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
  <!--应用名称-->
  <property name="APP_NAME" value="tulingmall-order"/>
  <!--日志文件保存路径-->
  <property name="LOG_FILE_PATH" value="${LOG_FILE:-${LOG_PATH:-${LOG_TEMP:-${java.io.tmpdir:-/tmp}}}/logs}"/>
  <contextName>${APP_NAME}</contextName>
  <!--每天记录日志到文件appender-->
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
  <fileNamePattern>${LOG_FILE_PATH}/${APP_NAME}-%d{yyyy-MM-dd}.log</fileNamePattern>
  <maxHistory>30</maxHistory>
  </rollingPolicy>
  <encoder>
  <pattern>${FILE_LOG_PATTERN}</pattern>
  </encoder>
  </appender>

  <!--输出到kafka的appender-->
  <appender name="KAFKA" class="com.github.danielwegener.logback.kafka.KafkaAppender">
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
  <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
  </encoder>
  <topic>test</topic>
  <!-- 不关心key的分区策略 -->
  <keyingStrategy class="com.github.danielwegener.logback.kafka.keying.NoKeyKeyingStrategy" />
  <!-- 异步发送消息，不会阻塞应用程序 -->
  <deliveryStrategy class="com.github.danielwegener.logback.kafka.delivery.AsynchronousDeliveryStrategy" />
  <producerConfig>bootstrap.servers=192.168.50.60:9092</producerConfig>
  <!-- 不需要等待kafka的ack确认，对于日志数据一般都选这种，可能会丢一点日志，但是关系不大 -->
  <producerConfig>acks=0</producerConfig>
  <!-- 如果kafka挂了，日志会发送到console上 -->
```

```
<appender-ref ref="CONSOLE"/>
</appender>

<root level="INFO">
<appender-ref ref="CONSOLE"/>
<appender-ref ref="FILE"/>
<appender-ref ref="KAFKA"/>
</root>
</configuration>
```

分布式调用链追踪系统Pinpoint

Pinpoint介绍

Pinpoint是一款全链路分析工具，提供了**无侵入式**的调用链监控、方法执行详情查看、应用状态信息监控等功能。基于GoogleDapper论文进行的实现，与另一款开源的全链路分析工具Zipkin类似，但相比Zipkin提供了无侵入式、代码维度的监控等更多的特性。 Pinpoint支持的功能比较丰富，可以支持如下几种功能：

- 服务拓扑图：对整个系统中应用的调用关系进行了可视化的展示，单击某个服务节点，可以显示该节点的详细信息，比如当前节点状态、请求数量等。
- 实时活跃线程图：监控应用内活跃线程的执行情况，对应用的线程执行性能可以有比较直观的了解。
- 请求响应散点图：以时间维度进行请求计数和响应时间的展示，拖过拖动图表可以选择对应的请求查看执行的详细情况。
- 请求调用栈查看：对分布式环境中每个请求提供了代码维度的可见性，可以在页面中查看请求针对到代码维度的执行详情，帮助查找请求的瓶颈和故障原因。
- 应用状态、机器状态检查：通过这个功能可以查看相关应用程序的其他的一些详细信息，比如CPU使用情况，内存状态、垃圾收集状态，TPS和JVM信息等参数。

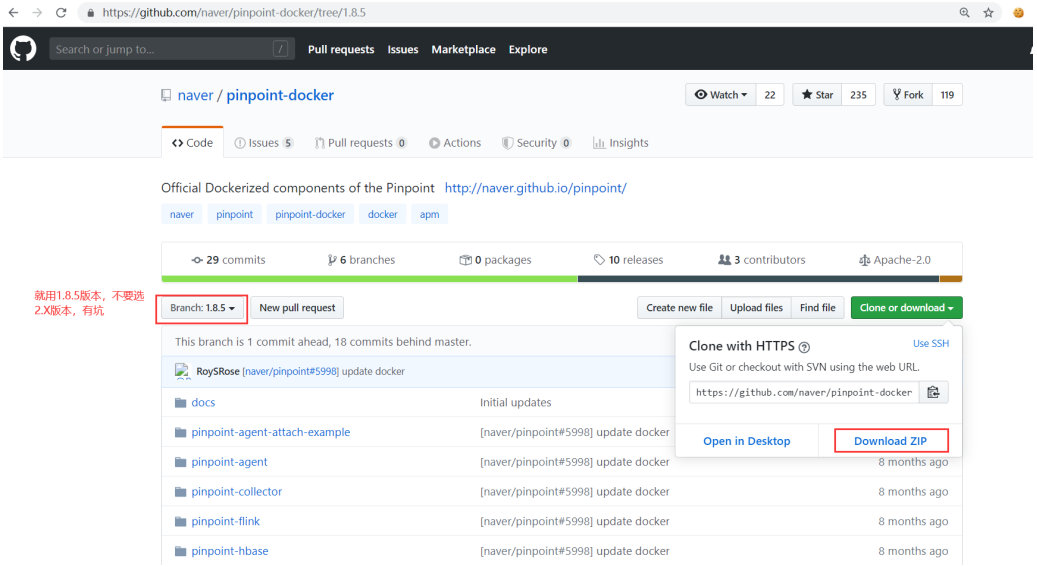
Pinpoint架构

Pinpoint 主要由 3 个组件外加 Hbase 数据库组成，三个组件分别为：Agent、Collector 和 Web UI。

- Agent组件：用于收集应用端监控数据，无侵入式，只需要在启动命令中加入部分参数即可。
- Collector组件：数据收集模块，接收Agent发送过来的监控数据，并存储到HBase。
- WebUI：监控展示模块，展示系统调用关系、调用详情、应用状态等，并支持报警等功能。

安装Pinpoint服务端

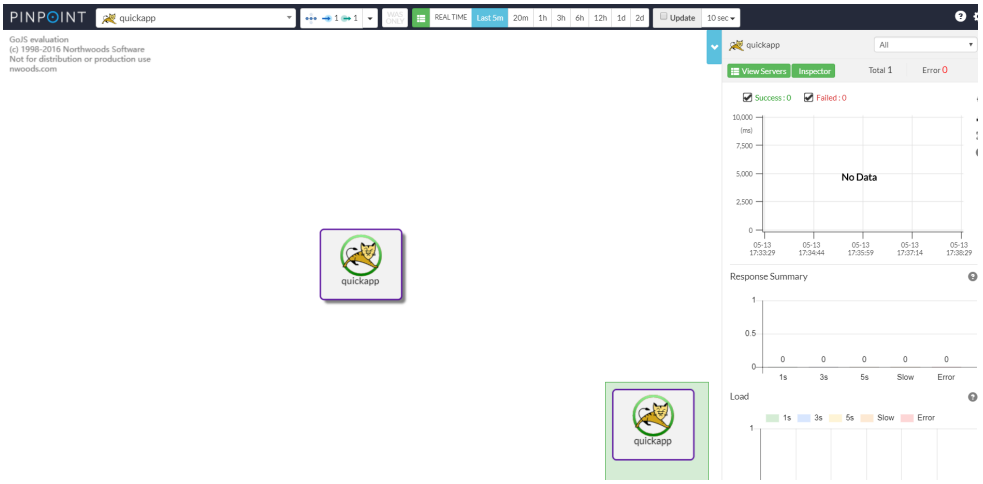
直接下载1.8.5版本的压缩包(比用git下快一点)：https://github.com/naver/pinpoint-docker/tree/1.8.5



新建一个目录pinpoint-docker，将压缩包内容解压到该目录，在该目录启动pinpoint：

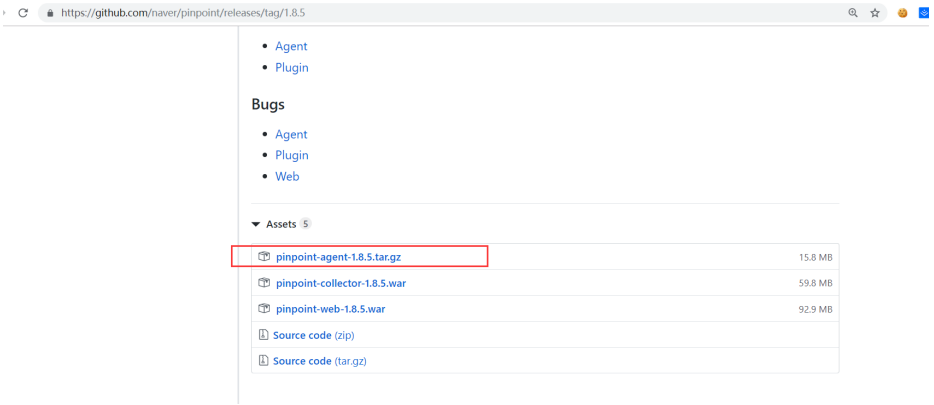
```
docker-compose pull && docker-compose up -d
```

启动之后访问：http://192.168.50.61:8079 ，ip是宿主机ip，页面如下：



安装Pinpoint客户端agent收集组件

进入链接：https://github.com/naver/pinpoint/releases/tag/1.8.5，下载pinpoint-agent-1.8.5.tar.gz



进入电商项目docker的部署目录docker-mall，将agent包放入并解压到pinpoint-agent目录，进入pinpoint-agent目录，修改配置文件pinpoint.config，一般情况只需要修改配置项 profiler.collector.ip=127.0.0.1 为你自己的Collector组件的IP。

在需要接入pinpoint的web应用程序启动vm参数里加上如下内容：

```
-javaagent:${pinpointPath}/pinpoint-bootstrap-1.8.5.jar #pinpoint-agent目录的jar包
-Dpinpoint.applicationName=order-service #在pinpoint上显示的名字
-Dpinpoint.agentId=order-service #id,可以和applicationName相同，也可以不同
```

将所有微服务的Dockerfile文件修改成如下所示，以tulingmall-order微服务为例：

```
# 基于哪个镜像
From java:8
# 复制文件到容器
ADD tulingmall-order-0.0.1-SNAPSHOT.jar /app.jar
# 配置容器启动后执行的命令
ENTRYPOINT ["java","-jar","-javaagent:/app/pinpoint-agent/pinpoint-bootstrap-1.8.5.jar","-Dpinpoint.agentId=order-service","-Dpinpoint.applicationName=order-service","/app.jar"]
```

然后修改docker-compose-app.yml文件，将宿主机的pinpoint-agent目录映射到微服务的容器里去，让微服务启动时能找到pinpoint-bootstrap-1.8.5.jar，内容如下：

```
version: '3'
services:
 tulingmall-authcenter:
 image: mall/tulingmall-authcenter:0.0.1
 build: ./tulingmall-authcenter
 container_name: tulingmall-authcenter
 ports:
 - 9999:9999
 volumes:
 - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
 - ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
 external_links:
 - nacos:nacos #可以用nacos这个域名访问nacos服务
 - mysql:db #可以用db这个域名访问mysql服务
```

```
tulingmall-gateway:
image: mall/tulingmall-gateway:0.0.1
build: ./tulingmall-gateway
container_name: tulingmall-gateway
ports:
- 8888:8888
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
- ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
depends_on:
- tulingmall-authcenter #gateway在authcenter启动之后再启动
external_links:
- nacos:nacos
tulingmall-member:
image: mall/tulingmall-member:0.0.1
build: ./tulingmall-member
container_name: tulingmall-member
ports:
- 8877:8877
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
- ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
external_links:
- nacos:nacos
- mysql:db #可以用db这个域名访问mysql服务
- mongo
- redis
- rabbitmq
tulingmall-product:
image: mall/tulingmall-product:0.0.1
build: ./tulingmall-product
container_name: tulingmall-product
ports:
- 8866:8866
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
- ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
external_links:
- nacos:nacos
- mysql:db #可以用db这个域名访问mysql服务
- redis
- zookeeper
tulingmall-order:
image: mall/tulingmall-order:0.0.1
build: ./tulingmall-order
container_name: tulingmall-order
ports:
- 8844:8844
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
- ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
external_links:
- nacos:nacos
- mysql:db #可以用db这个域名访问mysql服务
- redis
- rabbitmq
- rockermq
```

重新构建所有微服务镜像并启动容器：

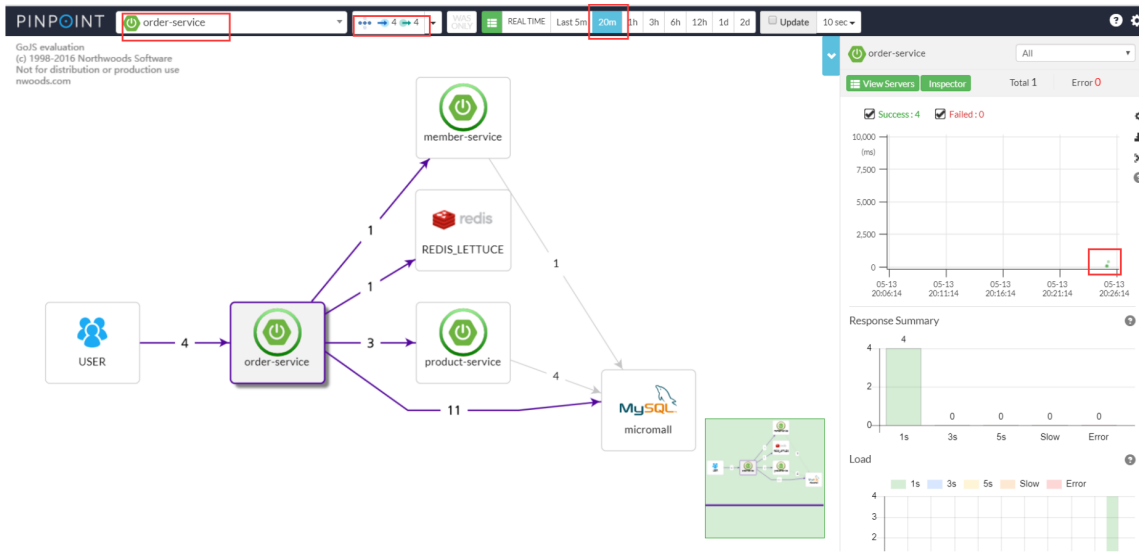
```
docker-compose -f docker-compose-app.yml up -d --build
```

这样web应用所有的调用链路请求都会被发送到pinpoint服务端。

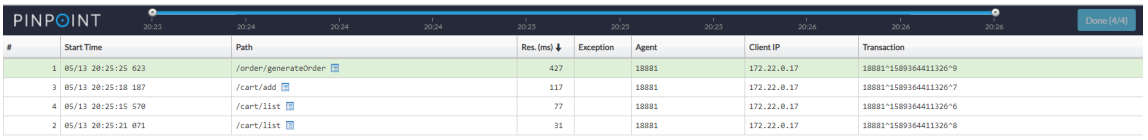
访问下微服务的几个api，访问接口参数参看视频

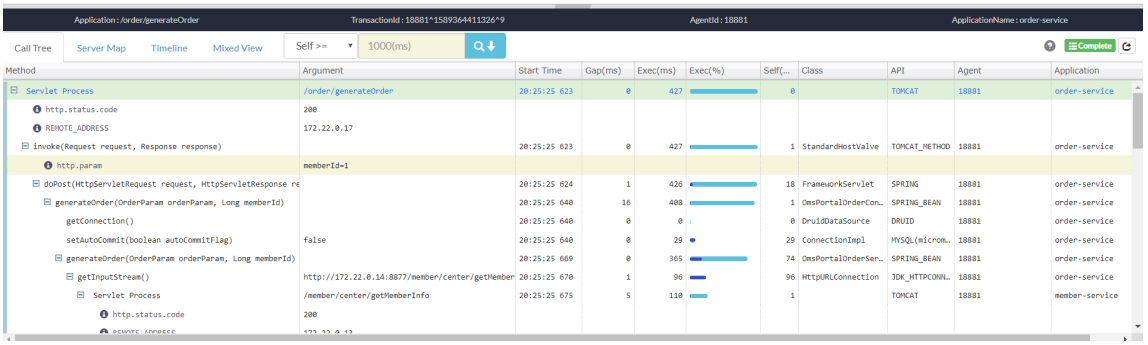
```
1、通过网关访问登录接口获取token，post方式：
http://192.168.50.60:8888/sso/login?username=test&password=test
2、通过网关访问添加购物车接口，post方式：
http://192.168.50.60:8888/cart/add?memberId=1&nickName=windir
3、通过网关访问查询购物车接口，get方式：
http://192.168.50.60:8888/cart/list?memberId=1
4、通过网关访问创建订单接口，post方式：
http://192.168.50.60:8888/order/generateOrder?memberId=1
```

然后再看下pinpoint的数据，如下图，选择order-service，会展示出订单服务应用调用的上下游所有链路：



选中右边垂直图里的小绿点，可以展示具体的请求调用链路，如下：





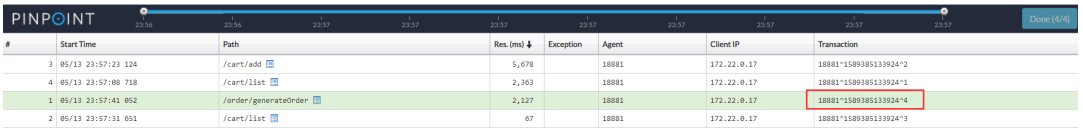
实现整个调用链路日志串联查询

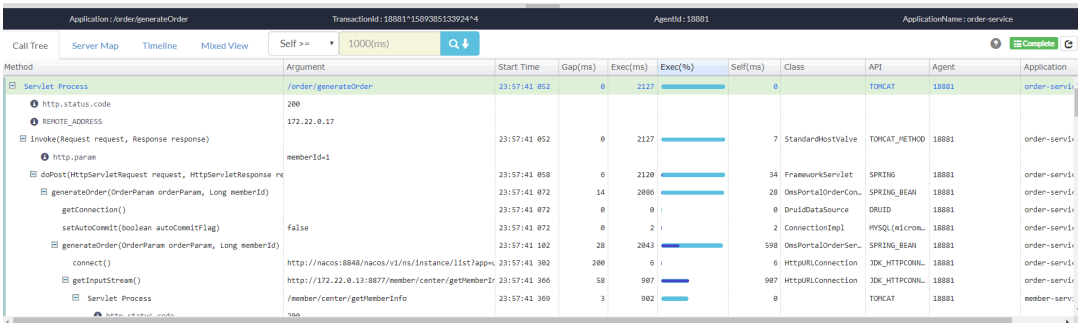
进入pinpoint-agent目录，修改配置文件pinpoint.config里的配置profiler.logback.logging.transactioninfo的值为true，重启微服务容器，这样pinpoint agent会把调用请求唯一标识transaction id写进logback的日志输出里，如果有跨微服务的请求调用，这个transaction id会从第一个请求调用开始生成，然后随着请求的调用，传递到其他的微服务里，我们只需要修改下微服务的logback配置文件logback-spring.xml，**加入PtxId，这个就是transaction id**

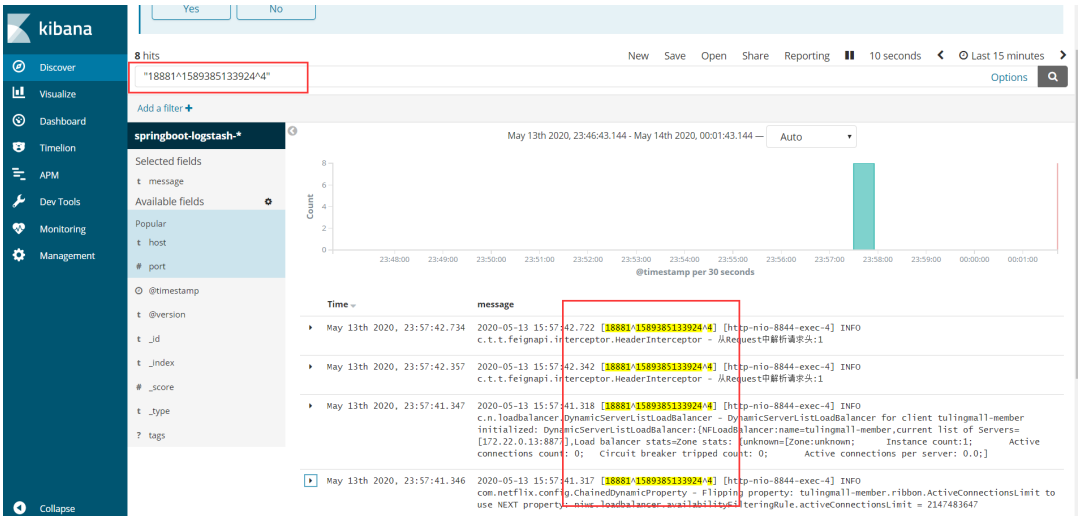
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration>
<configuration>
 <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
 <include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
 <!--应用名称-->
 <property name="APP_NAME" value="tulingmall-order"/>
 <!--日志文件保存路径-->
```

```
<property name="LOG_FILE_PATH" value="${LOG_FILE:-${LOG_PATH:-${LOG_TEMP:-${java.io.tmpdir:-/tmp}}}/logs}"/>
<contextName>${APP_NAME}</contextName>
<!--每天记录日志到文件appender-->
<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
<fileNamePattern>${LOG_FILE_PATH}/${APP_NAME}-%d{yyyy-MM-dd}.log</fileNamePattern>
<maxHistory>30</maxHistory>
</rollingPolicy>
<encoder>
<pattern>${FILE_LOG_PATTERN}</pattern>
</encoder>
</appender>


<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
<encoder>
<Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%X{PtxId}] [%thread] %-5level %logger{50} - %msg%n</Pattern>
<charset>UTF-8</charset>
</encoder>
</appender>


<!--输出到logstash的appender-->
<appender name="LOGSTASH" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
<!--可以访问的logstash日志收集端口-->
<destination>172.17.0.1:4560</destination>
<encoder>
<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%X{PtxId}] [%thread] %-5level %logger{50} - %msg%n</pattern>
<charset>UTF-8</charset>
</encoder>
</appender>
<root level="INFO">
<appender-ref ref="CONSOLE"/>
<appender-ref ref="FILE"/>
<appender-ref ref="LOGSTASH"/>
</root>
</configuration>
```

这样这个transaction id就会发往ELK日志搜集系统，如果我们在pinpoint里发现有调用较慢或报错的请求，我们可以把请求对应的transaction id复制出来，在kibana里搜索，搜索出来的所有日志就这个请求跨微服务系统调用的所有日志，这样就能非常方便的定位问题了。







文档：04-电商大数据日志收集系统实战.note

链接：http://note.youdao.com/noteshare?id=332a1c98e5ecb96a24c111c8aa7d762f&sub=80BB33F4629548149C6F8E2CAAF1DF79