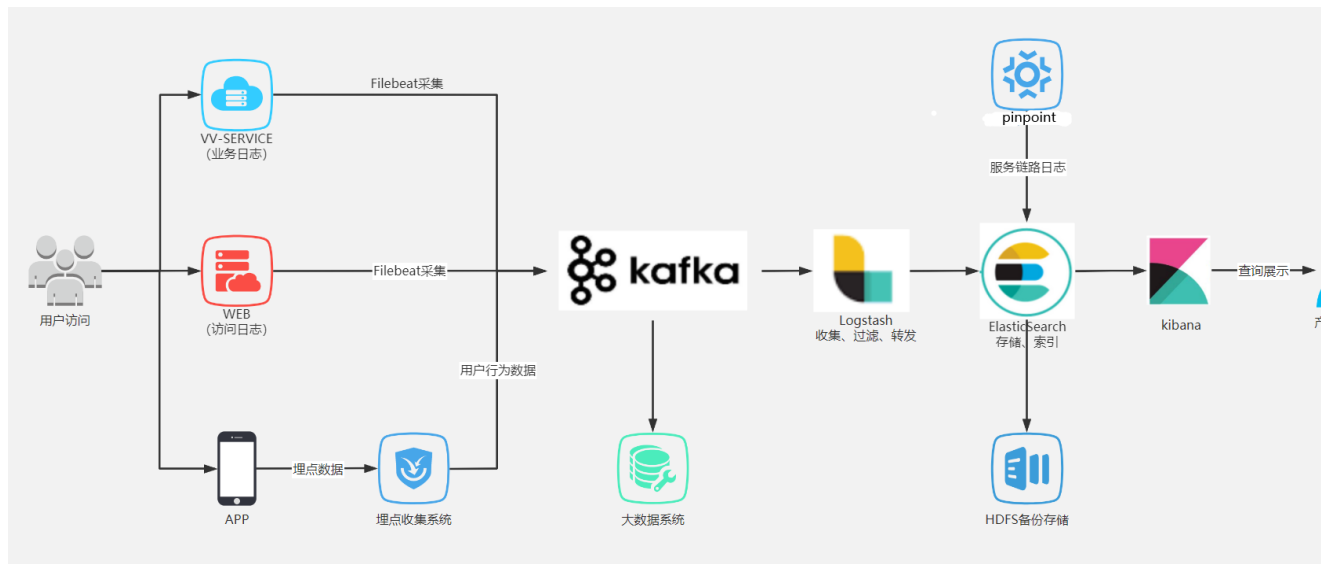


大型互联网公司后端日志收集系统架构



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：

```
1 version: '3'
2 services:
3   elasticsearch:
4     image: elasticsearch:6.4.0
5     container_name: elasticsearch
6     environment:
7       - "cluster.name=elasticsearch" #设置集群名称为elasticsearch
8       - "discovery.type=single-node" #以单一节点模式启动
9       - "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
10    volumes:
11      - /mydata/elasticsearch/plugins:/usr/share/elasticsearch/plugins #插件文件挂载
12      - /mydata/elasticsearch/data:/usr/share/elasticsearch/data #数据文件挂载
13    ports:
14      - 9200:9200
15      - 9300:9300
16    kibana:
17      image: kibana:6.4.0
18      container_name: kibana
19    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
```

```

20 - elasticsearch:es #可以用es这个域名访问elasticsearch服务
21 depends_on:
22 - elasticsearch #kibana在elasticsearch启动之后再启动
23 environment:
24 - "elasticsearch.hosts=http://es:9200" #设置访问elasticsearch的地址
25 ports:
26 - 5601:5601
27 logstash:
28 image: logstash:6.4.0
29 container_name: logstash
30 volumes:
31 - /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
32 depends_on:
33 - elasticsearch #kibana在elasticsearch启动之后再启动
34 links:
35 - elasticsearch:es #可以用es这个域名访问elasticsearch服务
36 ports:
37 - 4560:4560

```

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

```

1 input { #logstash直接收集日志的tcp端口
2   tcp {
3     mode => "server"
4     host => "0.0.0.0"
5     port => 4560 • codec => json_lines
6   }
7 }
8 output { #logstash将收集到的日志发送到es里存储
9   elasticsearch {
10    hosts => "es:9200"
11    index => "springboot-logstash-%{+YYYY.MM.dd}"
12  }
13 }

```

启动ELK，Elasticsearch启动可能需要好几分钟，要耐心等待

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

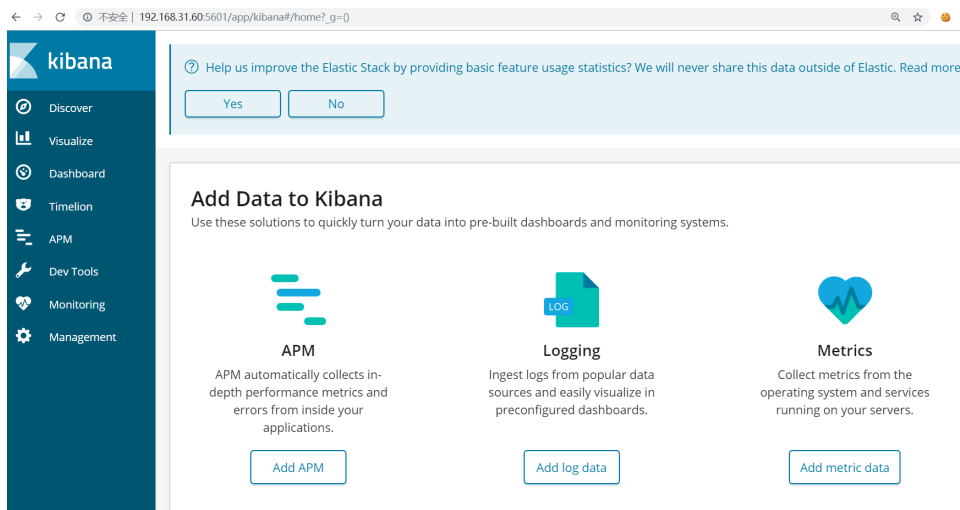
在logstash中安装json_lines插件

```

1 # 进入logstash容器
2 docker exec -it logstash /bin/bash
3 # 进入bin目录
4 cd /bin/
5 # 安装插件,有点慢,耐心等待
6 logstash-plugin install logstash-codec-json_lines
7 # 退出容器
8 exit
9 # 重启logstash服务
10 docker restart logstash
11 关闭防火墙并在kibana中查看
12 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

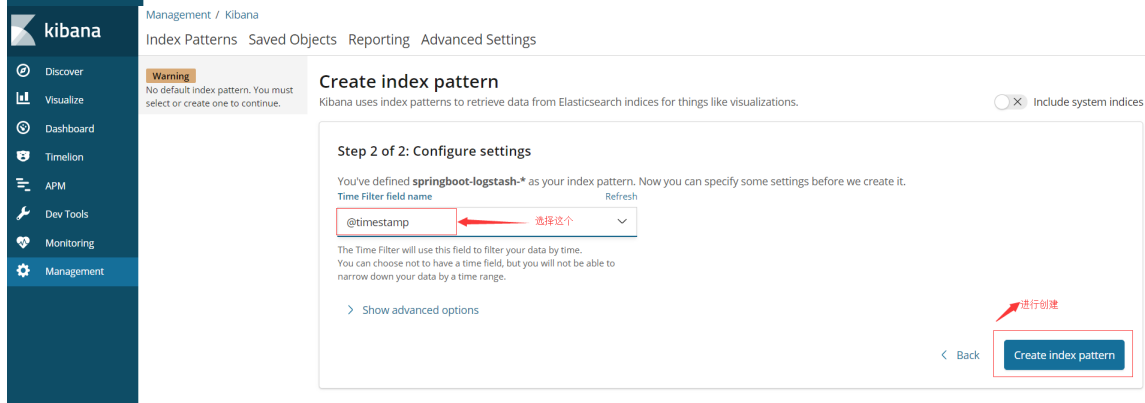
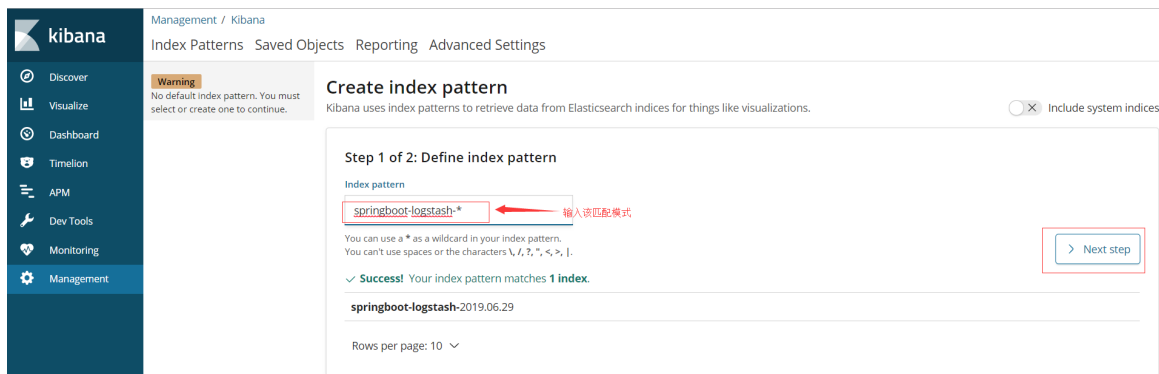
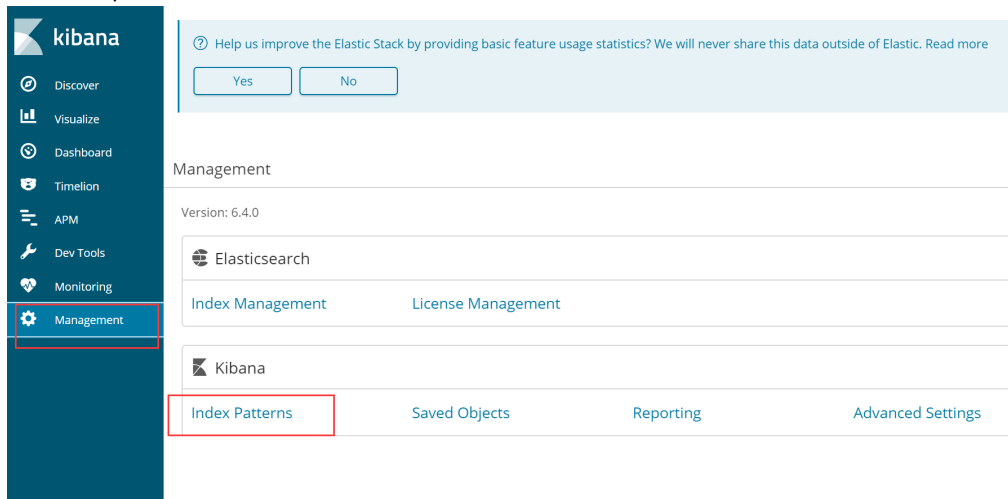
```
1 <!--集成logstash-->
2 <dependency>
3   <groupId>net.logstash.logback</groupId>
4   <artifactId>logstash-logback-encoder</artifactId>
5   <version>5.3</version>
6 </dependency>
```

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：

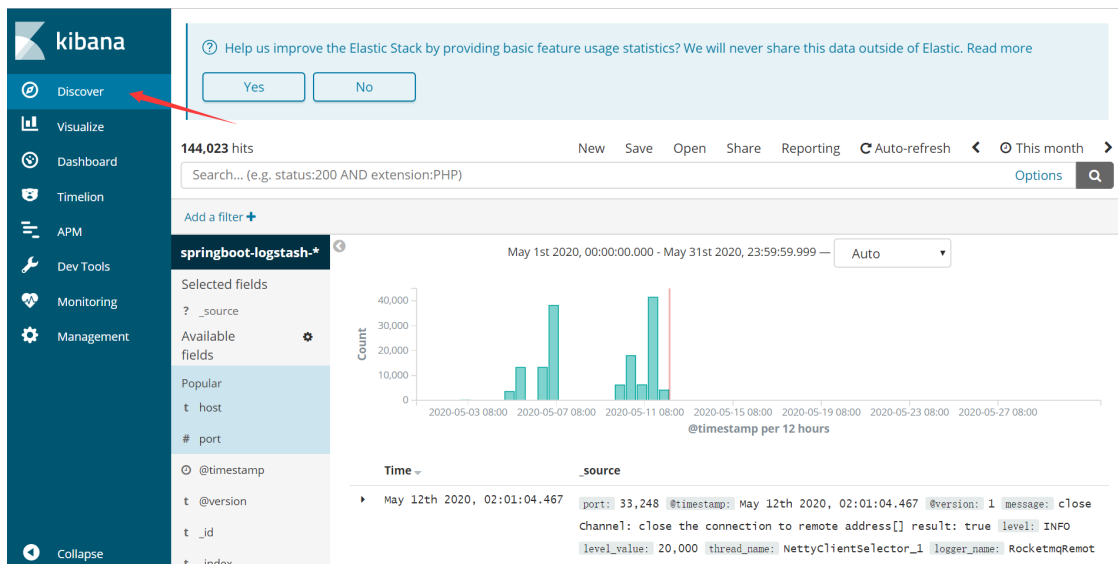
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE configuration>
3 <configuration>
4   <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
5   <include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
6   <!--应用名称-->
7   <property name="APP_NAME" value="tulingmall-order"/>
8   <!--日志文件保存路径-->
9   <property name="LOG_FILE_PATH" value="${LOG_FILE:-${LOG_PATH:-${LOG_TEMP:-${java.io.tmpdir:-/tmp}}}/logs}"/>
10  <contextName>${APP_NAME}</contextName>
11  <!--每天记录日志到文件appender-->
12  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
13    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
14      <fileNamePattern>${LOG_FILE_PATH}/${APP_NAME}-%d{yyyy-MM-dd}.log</fileNamePattern>
15      <maxHistory>30</maxHistory>
16    </rollingPolicy>
17    <encoder>
18      <pattern>${FILE_LOG_PATTERN}</pattern>
19    </encoder>
20  </appender>
21  <!--输出到logstash的appender-->
22  <appender name="LOGSTASH" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
23    <!--可以访问的logstash日志收集端口-->
24    <destination>172.17.0.1:4560</destination>
25    <encoder charset="UTF-8" class="net.logstash.logback.encoder.LogstashEncoder"/>
26  </appender>
27  <root level="INFO">
28    <appender-ref ref="CONSOLE"/>
29    <appender-ref ref="FILE"/>
30    <appender-ref ref="LOGSTASH"/>
31  </root>
32 </configuration>
```

运行Springboot应用，在kibana中查看日志信息

创建index pattern



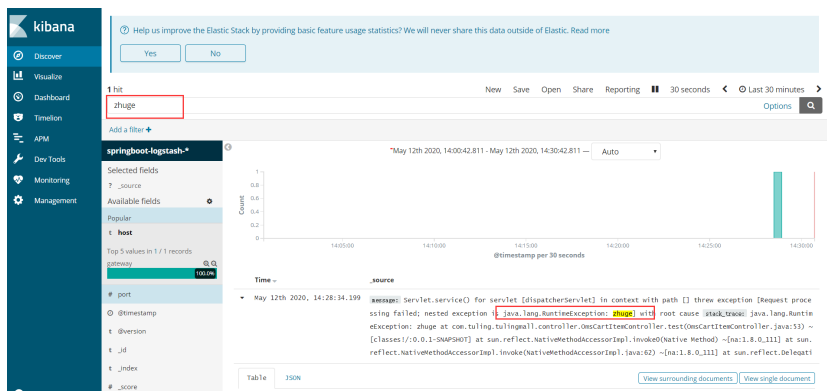
[查看收集的日志](#)



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：

```
1 kafka:
2   image: wurstmeister/kafka
3   ports:
4     - "9092:9092"
5   environment:
6     KAFKA_ADVERTISED_HOST_NAME: 192.168.50.60 #宿主机ip
7     KAFKA_CREATE_TOPICS: "test:1:1" #主题:分区:副本数
8     KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
9   volumes:
10    - /var/run/docker.sock:/var/run/docker.sock
```

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

```
1 input {
2   # tcp {
3   # mode => "server"
```

```

4 # host => "0.0.0.0"
5 # port => 4560
6 # codec => json_lines
7 # }
8
9 kafka {
10 id => "kafka_input_id" #随便写
11 bootstrap_servers => "192.168.65.60:9092"
12 topics => ["test"]
13 auto_offset_reset => "latest"
14 }
15 }
16 output {
17 elasticsearch {
18 hosts => "es:9200"
19 index => "springboot-logstash-%{+YYYY.MM.dd}"
20 }
21 }

```

web应用那边需要新加入一个依赖:

```

1 <dependency>
2 <groupId>com.github.danielwegener</groupId>
3 <artifactId>logback-kafka-appender</artifactId>
4 <version>0.2.0-RC2</version>
5 </dependency>

```

web应用下的logback-spring.xml文件内容需要修改, 如下:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE configuration>
3 <configuration>
4 <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
5 <include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
6 <!--应用名称-->
7 <property name="APP_NAME" value="tulingmall-order"/>
8 <!--日志文件保存路径-->
9 <property name="LOG_FILE_PATH" value="${LOG_FILE:-${LOG_PATH:-${LOG_TEMP:-${java.io.tmpdir:-/tmp}}}/logs}"/>
10 <contextName>${APP_NAME}</contextName>
11 <!--每天记录日志到文件appender-->
12 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
13 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
14 <fileNamePattern>${LOG_FILE_PATH}/${APP_NAME}-%d{yyyy-MM-dd}.log</fileNamePattern>
15 <maxHistory>30</maxHistory>
16 </rollingPolicy>
17 <encoder>
18 <pattern>${FILE_LOG_PATTERN}</pattern>
19 </encoder>
20 </appender>
21
22 <!--输出到kafka的appender-->
23 <appender name="KAFKA" class="com.github.danielwegener.logback.kafka.KafkaAppender">
24 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
25 <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
26 </encoder>
27 <topic>test</topic>
28 <!-- 不关心key的分区策略 -->
29 <keyingStrategy class="com.github.danielwegener.logback.kafka.keying.NoKeyKeyingStrategy" />
30 <!-- 异步发送消息, 不会阻塞应用程序 -->
31 <deliveryStrategy class="com.github.danielwegener.logback.kafka.delivery.AsynchronousDeliveryStrategy" />
32 <producerConfig>bootstrap.servers=192.168.50.60:9092</producerConfig>
33 <!-- 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 -->
34 <producerConfig>acks=0</producerConfig>
35 <!-- 如果kafka挂了, 日志会发送到console上 -->

```

```
36 <appender-ref ref="CONSOLE"/>
37 </appender>
38
39 <root level="INFO">
40 <appender-ref ref="CONSOLE"/>
41 <appender-ref ref="FILE"/>
42 <appender-ref ref="KAFKA"/>
43 </root>
44 </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。

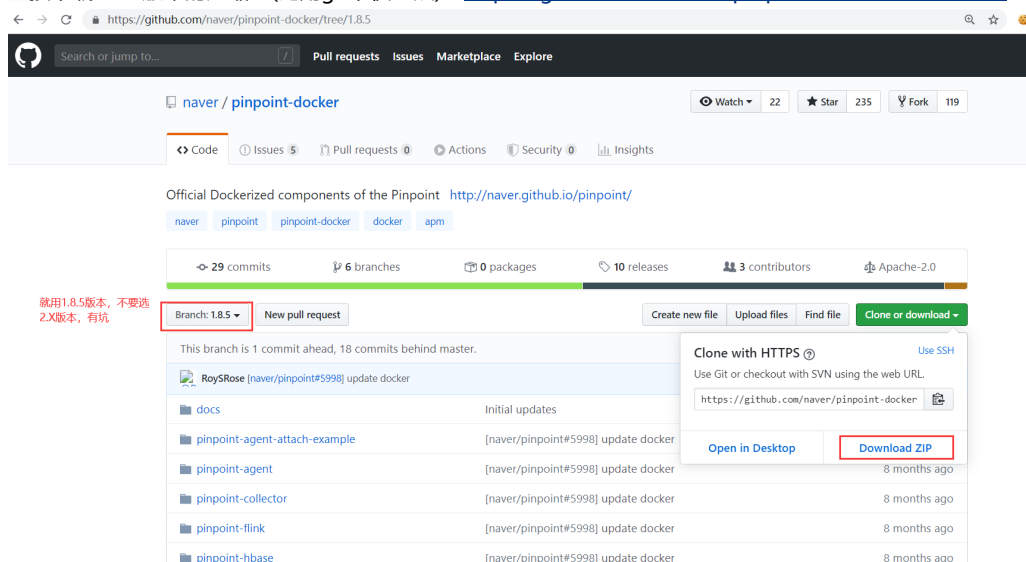
Pinpoint架构

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：

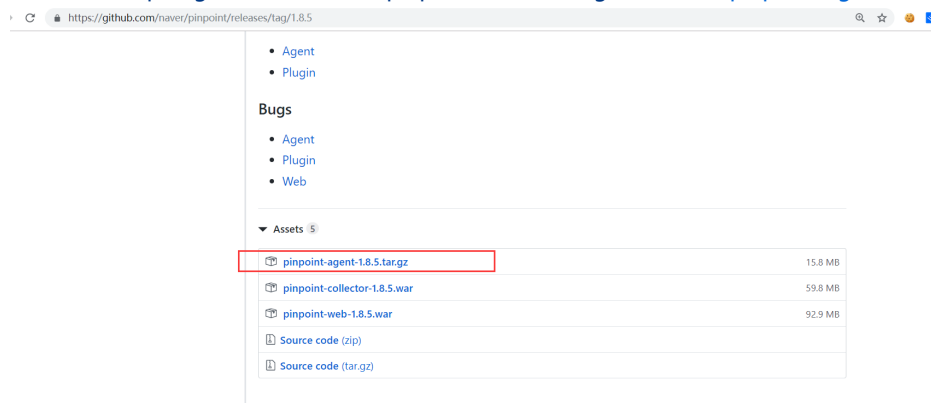
```
1 docker-compose pull && docker-compose up -d
```

启动之后访问：<http://192.168.50.61:8079>，ip是宿主主机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:

```
1 -javaagent:${pinpointPath}/pinpoint-bootstrap-1.8.5.jar #pinpoint-agent目录的jar包
2 -Dpinpoint.applicationName=order-service #在pinpoint上显示的名字
3 -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:

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

```
1 version: '3'
2 services:
3   tulingmall-authcenter:
4     image: mall/tulingmall-authcenter:0.0.1
5     build: ./tulingmall-authcenter
6     container_name: tulingmall-authcenter
7     ports:
8       - 9999:9999
9     volumes:
10      - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
11      - ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
12     external_links:
13       - nacos:nacos #可以用nacos这个域名访问nacos服务
14       - mysql:db #可以用db这个域名访问mysql服务
```



```

15  tulingmall-gateway:
16  image: mall/tulingmall-gateway:0.0.1
17  build: ./tulingmall-gateway
18  container_name: tulingmall-gateway
19  ports:
20  - 8888:8888
21  volumes:
22  - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
23  - ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
24  depends_on:
25  - tulingmall-authcenter #gateway在authcenter启动之后再启动
26  external_links:
27  - nacos:nacos
28  tulingmall-member:
29  image: mall/tulingmall-member:0.0.1
30  build: ./tulingmall-member
31  container_name: tulingmall-member
32  ports:
33  - 8877:8877
34  volumes:
35  - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
36  - ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
37  external_links:
38  - nacos:nacos
39  - mysql:db #可以用db这个域名访问mysql服务
40  - mongo
41  - redis
42  - rabbitmq
43  tulingmall-product:
44  image: mall/tulingmall-product:0.0.1
45  build: ./tulingmall-product
46  container_name: tulingmall-product
47  ports:
48  - 8866:8866
49  volumes:
50  - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
51  - ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
52  external_links:
53  - nacos:nacos
54  - mysql:db #可以用db这个域名访问mysql服务
55  - redis
56  - zookeeper
57  tulingmall-order:
58  image: mall/tulingmall-order:0.0.1
59  build: ./tulingmall-order
60  container_name: tulingmall-order
61  ports:
62  - 8844:8844
63  volumes:
64  - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
65  - ./pinpoint-agent:/app/pinpoint-agent #映射pinpoint的agent包
66  external_links:
67  - nacos:nacos
68  - mysql:db #可以用db这个域名访问mysql服务
69  - redis
70  - rabbitmq
71  - rockermq

```

重新构建所有微服务镜像并启动容器：

```
1 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。

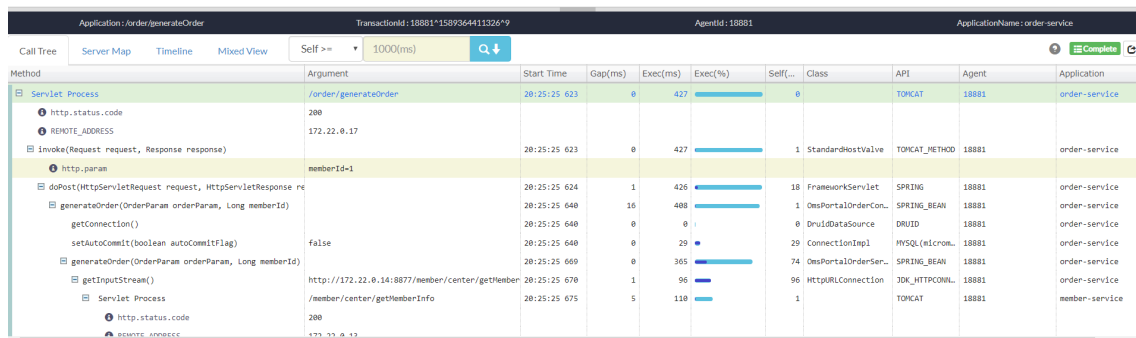
```

1 1、通过网关访问登录接口获取token，post方式：
2 http://192.168.50.60:8888/sso/login?username=test&password=test
3 2、通过网关访问添加购物车接口，post方式：
4 http://192.168.50.60:8888/cart/add?memberId=1&nickName=windir
5 3、通过网关访问查询购物车接口，get方式：
6 http://192.168.50.60:8888/cart/list?memberId=1
7 4、通过网关访问创建订单接口，post方式：
8 http://192.168.50.60:8888/order/generateOrder?memberId=1

```

The screenshot displays the Pinpoint application interface. At the top, the 'order-service' is selected, showing a success rate of 4 and a failure rate of 4. The service map on the left shows the 'order-service' as the central component, with incoming traffic from a 'USER' (4 requests) and outgoing traffic to 'member-service' (1 request), 'redis' (1 request), 'product-service' (3 requests), and 'MySQL' (11 requests). The 'MySQL' component is further detailed as 'micromall'. On the right, the 'order-service' details panel shows a 'Response Summary' bar chart with a peak at 4 requests and a 'Load' chart showing a peak at 4 requests. The 'Response Summary' chart also indicates a peak at 4 requests and a 'Load' chart showing a peak at 4 requests.

PINPOINT								Done (4/4)	
#	Start Time	Path	Res. (ms) ↓	Exception	Agent	Client IP	Transaction		
1	05/13 20:25:25 623	/order/generateOrder	427		18881	172.22.0.17	188811589164411326-9		
3	05/13 20:25:18 187	/cart/add	117		18881	172.22.0.17	188811589164411326-7		
4	05/13 20:25:15 570	/cart/list	77		18881	172.22.0.17	188811589164411326-6		
2	05/13 20:25:21 071	/cart/list	31		18881	172.22.0.17	188811589164411326-8		



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**

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

```

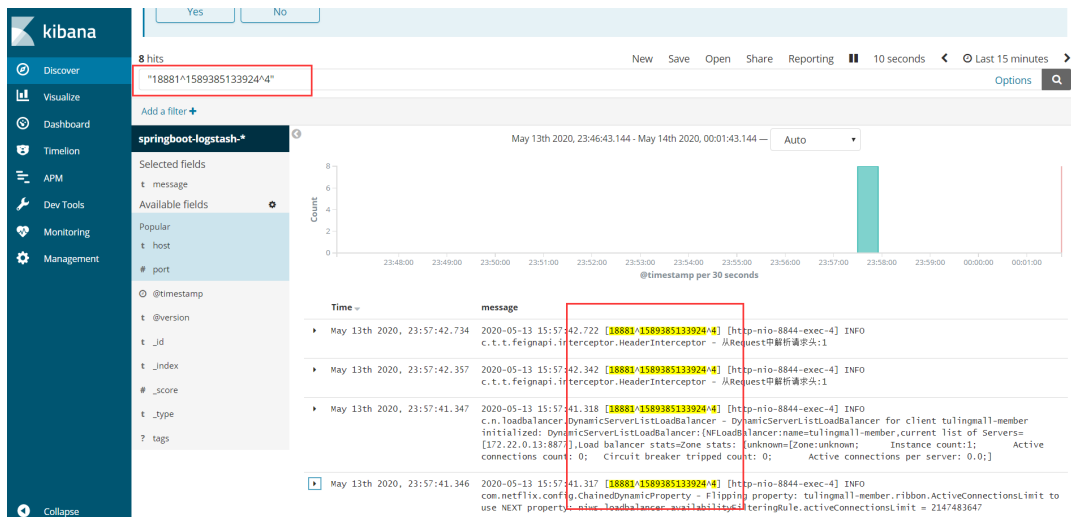
9 <property name="LOG_FILE_PATH" value="${LOG_FILE:-${LOG_PATH:-${LOG_TEMP:-${java.io.tmpdir:-/tmp}}}/logs}"/>
10 <contextName>${APP_NAME}</contextName>
11 <!--每天记录日志到文件appender-->
12 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
13 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
14 <fileNamePattern>${LOG_FILE_PATH}/${APP_NAME}-%d{yyyy-MM-dd}.log</fileNamePattern>
15 <maxHistory>30</maxHistory>
16 </rollingPolicy>
17 <encoder>
18 <pattern>${FILE_LOG_PATTERN}</pattern>
19 </encoder>
20 </appender>
21
22
23 <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
24 <encoder>
25 <Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%X{PtxId}] [%thread] %-5level %logger{50} - %msg%n</Pattern>
26 <charset>UTF-8</charset>
27 </encoder>
28 </appender>
29
30
31 <!--输出到logstash的appender-->
32 <appender name="LOGSTASH" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
33 <!--可以访问的logstash日志收集端口-->
34 <destination>172.17.0.1:4560</destination>
35 <encoder>
36 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%X{PtxId}] [%thread] %-5level %logger{50} - %msg%n</pattern>
37 <charset>UTF-8</charset>
38 </encoder>
39 </appender>
40 <root level="INFO">
41 <appender-ref ref="CONSOLE"/>
42 <appender-ref ref="FILE"/>
43 <appender-ref ref="LOGSTASH"/>
44 </root>
45 </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。

#	StartTime	Path	Res.(ms) ↓	Exception	Agent	Client IP	Transaction
3	05/13 23:57:33 124	/cart/add	5,678		18881	172.22.0.17	18881*1589385133924*2
4	05/13 23:57:40 718	/cart/list	1,363		18881	172.22.0.17	18881*1589385133924*1
1	05/13 23:57:41 852	/order/generateOrder	2,127		18881	172.22.0.17	18881*1589385133924*4
2	05/13 23:57:31 651	/cart/list	67		18881	172.22.0.17	18881*1589385133924*3

Application: /order/generateOrder		TransactionId: 18881*1589385133924*4		AgentId: 18881		ApplicationName: order-service				
Call Tree	Server Map	Timeline	Mixed View	Self >= 1000(ms)						
Method	Argument	Start Time	Gap(ms)	Exec(ms)	Exec(%)	Self(ms)	Class	API	Agent	Application
Servlet Process	/order/generateOrder	23:57:41 852	0	2127	<div></div>	0		TONCAT	18881	order-servi
http.status.code	200									
REMOTE_ADDRESS	172.22.0.17									
Invoke(Request request, Response response)		23:57:41 852	0	2127	<div></div>	7	StandardHostValve	TONCAT_METHOD	18881	order-servi
http.param	memberId=1									
doPost(HttpServletRequest request, HttpServletResponse re		23:57:41 858	6	2128	<div></div>	34	FrameworkServlet	SPRING	18881	order-servi
generateOrder(OrderParam orderParam, Long memberId)		23:57:41 872	14	2086	<div></div>	28	OrderPortalOrderCon.	SPRING_BEAN	18881	order-servi
getConnection()		23:57:41 872	0	0	<div></div>	0	DruidDataSource	DRUID	18881	order-servi
setAutoCommit(boolean autoCommitFlag)	false	23:57:41 872	0	2	<div></div>	2	ConnectionImpl	Mysql(micro	18881	order-servi
generateOrder(OrderParam orderParam, Long memberId)		23:57:41 182	28	2043	<div></div>	598	OrderPortalOrderSer	SPRING_BEAN	18881	order-servi
connect()	http://nacos:8848/nacos/v1/ns/instance/list?app=	23:57:41 382	280	6	<div></div>	6	HttpURLConnection	JOKE_HTTPCONN	18881	order-servi
getInputStream()	http://172.22.0.13:8877/member/center/getMemberIn	23:57:41 366	58	907	<div></div>	907	HttpURLConnection	JOKE_HTTPCONN	18881	order-servi
Servlet Process	/member/center/getMemberInfo	23:57:41 369	3	902	<div></div>	0		TONCAT	18881	member-servi
http.status.code	200									



文档：04-电商大数据日志收集系统实战.note

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

[id=332a1c98e5ecb96a24c111c8aa7d762f&sub=80BB33F4629548149C6F8E2CAAF1DF79](http://note.youdao.com/noteshare?id=332a1c98e5ecb96a24c111c8aa7d762f&sub=80BB33F4629548149C6F8E2CAAF1DF79)