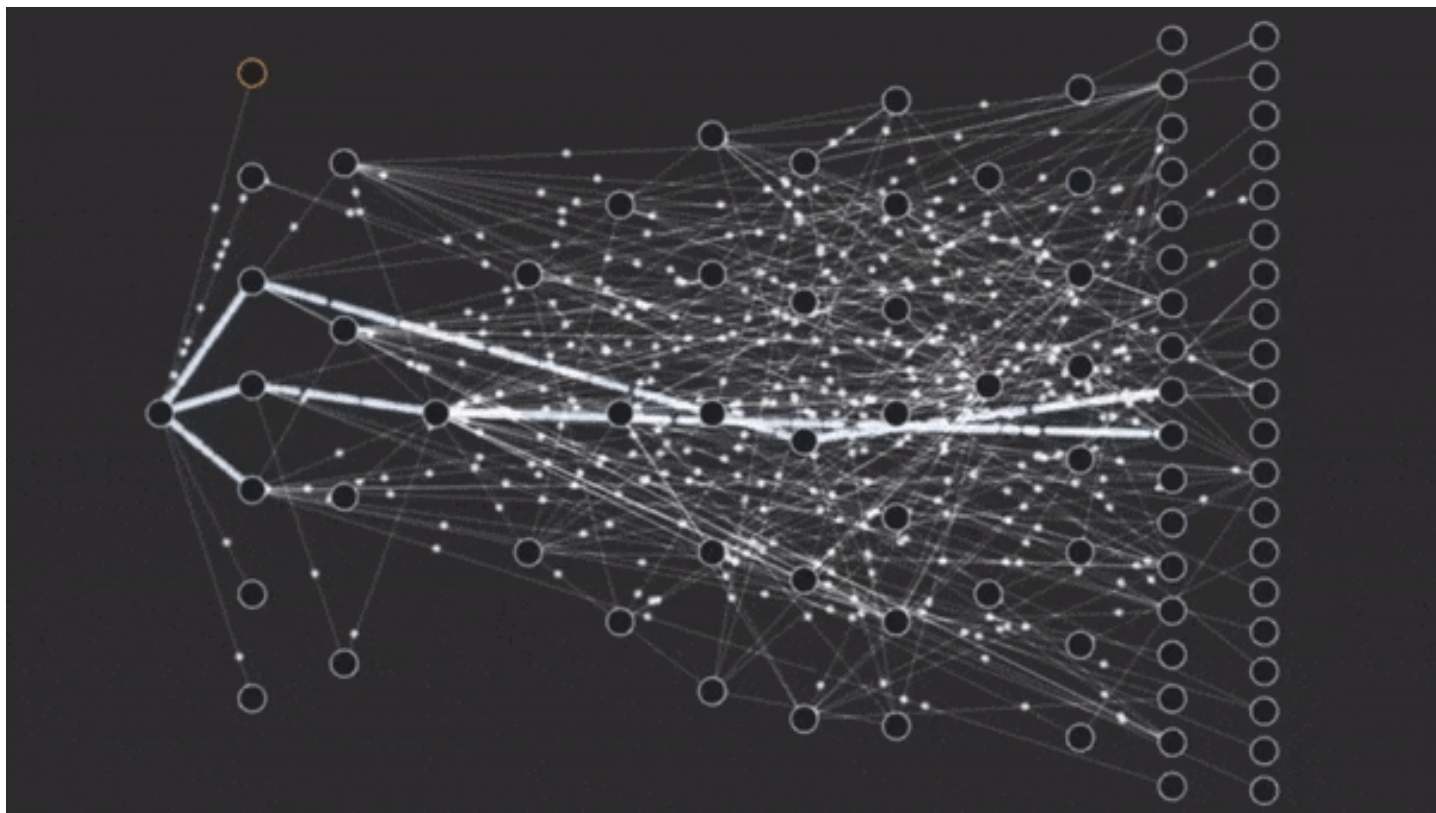


微服务链路追踪SkyWalking
链路追踪介绍
1、skywalking是什么
1.2 链路追踪框架对比
1.3 性能对比
1.4 Skywalking主要功能特性
2、SkyWalking 环境搭建部署
2.2 搭建SkyWalking OAP 服务
2.3 SkyWalking中三个概念
3、SkyWalking 接入微服务
3.1 linux环境—通过jar包方式接入
3.2 windos环境—在IDEA中使用Skywalking
3.3 Skywalking跨多个微服务跟踪
4、Skywalking持久化跟踪数据
4.1 基于mysql持久化:
5、自定义SkyWalking链路追踪
5.1 @Trace将方法加入追踪链路
5.2 加入@Tags或@Tag
6、Skywalking集成日志框架
Skywalking通过grpc上报日志 （需要v8.4.0+）
7、SkyWalking 告警功能

链路追踪介绍

对于一个大型的几十个、几百个微服务构成的微服务架构系统，通常会遇到下面一些问题，比如：

- 1. 如何串联整个调用链路，快速定位问题？
- 2. 如何理清各个微服务之间的依赖关系？
- 3. 如何进行各个微服务接口的性能分析？
- 4. 如何跟踪整个业务流程的调用处理顺序？



1、skywalking是什么

skywalking是一个国产开源框架，2015年由吴晟开源，2017年加入Apache孵化器。skywalking是分布式系统的应用程序性能监视工具，专为微服务、云原生架构和基于容器（Docker、K8s、Mesos）架构而设计。它是一款优秀的 APM（Application Performance Management）工具，包括了分布式追踪、性能指标分析、应用和服务依赖分析等。

官网：<http://skywalking.apache.org/>

下载：<http://skywalking.apache.org/downloads/>

Github：<https://github.com/apache/skywalking>

文档：<https://skywalking.apache.org/docs/main/v8.4.0/readme/>

中文文档：<https://skyapm.github.io/document-cn-translation-of-skywalking/>

1.2 链路追踪框架对比

1. Zipkin是Twitter开源的调用链分析工具，目前基于springcloud sleuth得到了广泛的使用，特点是轻量，使用部署简单。
2. Pinpoint是韩国人开源的基于字节码注入的调用链分析，以及应用监控分析工具。特点是支持多种插件，UI功能强大，接入端无代码侵入。
3. SkyWalking是本土开源的基于字节码注入的调用链分析，以及应用监控分析工具。特点是支持多种插件，UI功能较强，接入端无代码侵入。目前已加入Apache孵化器。
4. CAT是大众点评开源的基于编码和配置的调用链分析，应用监控分析，日志采集，监控报警等一系列的监控平台工具。

项目	Cat	Zipkin	Skywalking
调用链可视化	有	有	有
聚合报表	非常丰富	少	较丰富
服务依赖图	简单	简单	好
埋点方式	侵入式	侵入式	非侵入，字节码增强
VM监控指标	好	无	有
支持语言	java/.net	丰富	java/.net/Nodejs/php/go
存储机制	mysql (报表)、本地文件/HDFS (调用链)	内存、es、mysql等	H2、es
社区支持	主要在国内	国外主流	Apache支持
使用案例	美团、携程、陆金所	京东、阿里定制后不开源	华为、小米、当当、微众银行
APM	是	否	是
开发基础	eBay cal	Google Dapper	Google Dapper
是否支持webflux	否	是	是
Github stars(2019.12)	12.3K	12.2K	11.8K

1.3 性能对比

模拟了三种并发用户：500，750，1000。使用jmeter测试，每个线程发送30个请求，设置思考时间为10ms。使用的采样率为1，即100%，这边与生产可能有差别。pinpoint默认的采样率为20，即50%，通过设置agent的配置文件改为100%。zipkin默认也是1。组合起来，一共有12种。下面看下汇总表：

Id	APM	采样率	线程数	请求总数	平均请求时间 ms	最小请求时间 ms	最大请求时间 ms	90%Line	错误率 %	CPU	memory	Throughput /sec
1	none	1	500	15000	17	9	824	21	0	45%	50%	1385
2	Zipkin	1	500	15000	117	10	2101	263	0	56%	55%	990
3	Skywalking	1	500	15000	22	10	1026	23	0	50%	52%	1228
4	Pinpoint	1	500	15000	201	10	7236	746	0	48%	52%	774
5	none	1	750	22500	321	10	15107	991	0	56%	48%	956
6	Zipkin	1	750	22500	489	10	27614	1169	0	63%	55%	582
7	Skywalking	1	750	22500	396	10	16478	941	0	55%	50%	908
8	Pinpoint	1	750	22500	681	10	28138	1919	0	56%	48%	559
9	none	1	1000	30000	704	10	39772	1621	0	59%	53%	557
10	Zipkin	1	1000	30000	1021	10	36836	1978	0	63%	55%	533
11	Skywalking	1	1000	30000	824	10	25983	1758	0	62%	55%	667
12	Pinpoint	1	1000	30000	1148	10	40971	2648	0	60%	52%	514

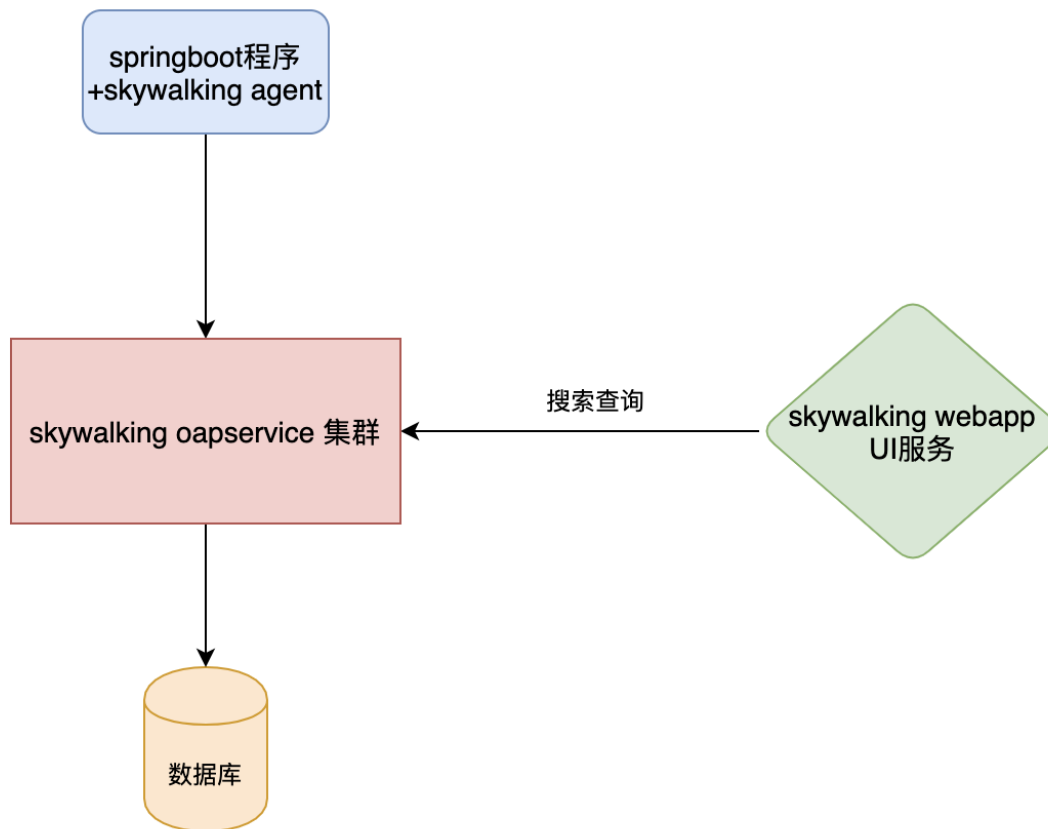
从上表可以看出，在三种链路监控组件中，**skywalking**的探针**对吞吐量的影响最小**，**zipkin**的吞吐量居中。**pinpoint**的探针**对吞吐量的影响较为明显**，在500并发用户时，测试服务的吞吐量从1385降低到774，影响很大。然后再看下CPU和memory的影响，在内部服务器进行的压测，对CPU和memory的影响都差不多在10%之内。

1.4 Skywalking主要功能特性

- 多种监控手段，可以通过语言探针和service mesh获得监控的数据；
- 支持多种语言自动探针，包括 Java，.NET Core 和 Node.JS；
- 轻量高效，无需大数据平台和大量的服务器资源；
- 模块化，UI、存储、集群管理都有多种机制可选；

- 5、支持告警;
- 6、优秀的可视化解决方案;

2、SkyWalking 环境搭建部署



- skywalking agent和业务系统绑定在一起，负责收集各种监控数据
- Skywalking oap service是负责处理监控数据的，比如接受skywalking agent的监控数据，并存储在数据库中;接受skywalking webapp的前端请求，从数据库查询数据，并返回数据给前端。Skywalking oap service通常以集群的形式存在。
- skywalking webapp，前端界面，用于展示数据。
- 用于存储监控数据的数据库，比如mysql、elasticsearch等。

2.1 下载 SkyWalking

下载: <http://skywalking.apache.org/downloads/>

Download the SkyWalking

Use the links below to download the Apache SkyWalking

Only source code releases are official Apache releases.

Download the latest versions

SkyWalking APM
SkyWalking is an Observability Analysis Platform and APM Performance Management system.

v8.4.0 for ElasticSearch 6 Feb. 4th, 2021	[tar] [asc] [sha512]
v8.4.0 for H2/MySQL/TiDB/InfluxDB/ElasticSearch 7 Feb. 4th, 2021	[tar] [asc] [sha512]
v8.3.0 for ElasticSearch 6 Dec. 3rd, 2020	[tar] [asc] [sha512]
v8.3.0 for H2/MySQL/TiDB/InfluxDB/ElasticSearch 7 Dec. 3rd, 2020	[tar] [asc] [sha512]
v8.2.0 for ElasticSearch 6 Oct. 27th, 2020	[tar] [asc] [sha512]
v8.2.0 for H2/MySQL/TiDB/InfluxDB/ElasticSearch 7 Oct. 27th, 2020	[tar] [asc] [sha512]

目录结构

- webapp: UI 前端 (web 监控页面) 的 jar 包和配置文件;
- oap-lib: 后台应用的 jar 包, 以及它的依赖 jar 包, 里边有一个 server-starter-*.jar 就是启动程序;
- config: 启动后台应用程序的配置文件, 是使用的各种配置
- bin: 各种启动脚本, 一般使用脚本 startup.* 来启动 **web 页面** 和对应的 **后台应用**;
 - oapService.*: 默认使用的后台程序的启动脚本; (使用的是默认模式启动, 还支持其他模式, 各模式区别见 启动模式)
 - oapServiceInit.*: 使用 init 模式启动; 在此模式下, OAP服务器启动以执行初始化工作, 然后退出
 - oapServiceNoInit.*: 使用 no init模式启动; 在此模式下, OAP服务器不进行初始化。
 - webappService.*: UI 前端的启动脚本;
 - startup.*: 组合脚本, 同时启动 oapService.*、webappService.* 脚本;
- agent:
 - skywalking-agent.jar: 代理服务 jar 包
 - config: 代理服务启动时使用的配置文件
 - plugins: 包含多个插件, 代理服务启动时会加载改目录下的所有插件 (实际是各种 jar 包)
 - optional-plugins: 可选插件, 当需要支持某种功能时, 比如 SpringCloud Gateway, 则需要把对应的 jar 包拷贝到 plugins 目录下;

2.2 搭建SkyWalking OAP 服务

启动脚本bin/startup.sh

```
[root@redis apache-skywalking-apm-bin-es7]# bin/startup.sh
SkyWalking OAP started successfully!
SkyWalking Web Application started successfully!
```

日志信息存储在logs目录

```
logs/
├── oap.log
├── skywalking-oap-server.log
├── webapp-console.log
└── webapp.log
```

启动成功后会启动两个服务, 一个是skywalking-oap-server, 一个是skywalking-web-ui : 8868

skywalking-oap-server服务启动后会暴露11800 和 12800 两个端口, 分别为**收集监控数据的端口11800**和**接受前端请求的端口12800**, 修改端口可以修改config/applicaiton.yml

```
[root@redis apache-skywalking-apm-bin-es7]# tail -f logs/skywalking-oap-server.log
2021-02-21 15:02:00,655 - org.apache.skywalking.oap.server.library.server.grpc.GRPC
Server - 140 [main] INFO [] - Bind handler JVMMetricReportServiceHandler into gRPC
server 0.0.0.0:11800
2021-02-21 15:02:00,661 - org.apache.skywalking.oap.server.library.module.Bootstrap
Flow - 46 [main] INFO [] - start the provider default in receiver-meter module.
2021-02-21 15:02:00,661 - org.apache.skywalking.oap.server.library.server.grpc.GRPC
Server - 140 [main] INFO [] - Bind handler MeterServiceHandler into gRPC server 0.
0.0.0:11800
2021-02-21 15:02:00,963 - org.apache.skywalking.oap.server.library.server.jetty.Jet
tyServer - 101 [main] INFO [] - start server, host: 0.0.0.0, port: 12800
2021-02-21 15:02:00,967 - org.eclipse.jetty.server.Server - 359 [main] INFO [] - j
etty-9.4.28.v20200408; built: 2020-04-08T17:49:39.557Z; git: ab228fde9e55e9164c738d
7fa121f8ac5acd51c9; jvm 1.8.0_181-b13
2021-02-21 15:02:01,050 - org.eclipse.jetty.server.handler.ContextHandler - 843 [ma
in] INFO [] - Started o.e.j.s.ServletContextHandler@37a3ec27{/,null,AVAILABLE}
2021-02-21 15:02:01,069 - org.eclipse.jetty.server.AbstractConnector - 331 [main] I
NFO [] - Started ServerConnector@31c2affc{HTTP/1.1, (http/1.1)}{0.0.0.0:12800}
2021-02-21 15:02:01,069 - org.eclipse.jetty.server.Server - 399 [main] INFO [] - S
tarted @25088ms
2021-02-21 15:02:01,070 - org.apache.skywalking.oap.server.core.storage.Persistence
Timer - 56 [main] INFO [] - persistence timer start
```

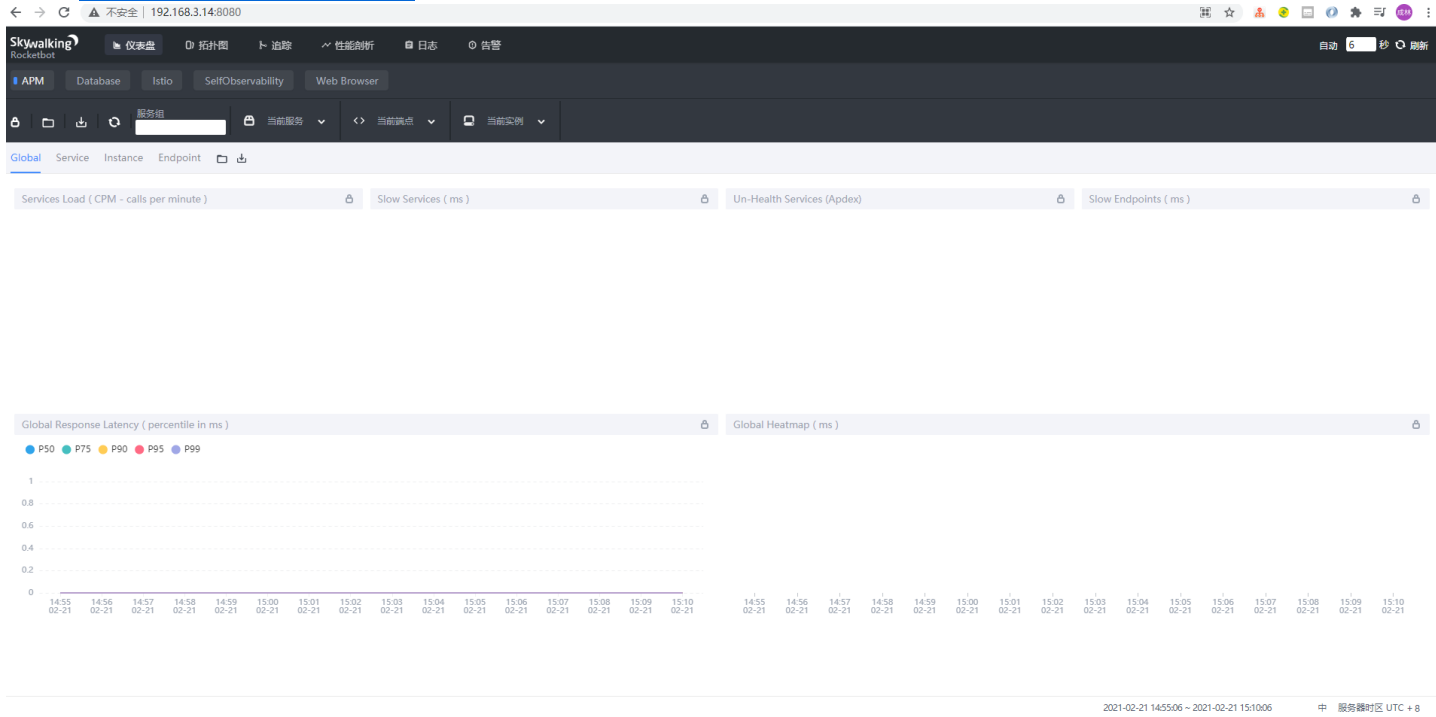
skywalking-web-ui服务会占用 8080 端口, 修改端口可以修改webapp/webapp.yml


```
server:
  port: 8080

collector:
  path: /graphql
  ribbon:
    ReadTimeout: 10000
    # Point to all backend's restHost:restPort, split by ,
    listOfServers: 127.0.0.1:12800
```

`server.port`: SkyWalking UI服务端口，默认是8080；
`collector.ribbon.listOfServers`: SkyWalking OAP服务地址数组，SkyWalking UI界面的数据是通过请求SkyWalking OAP服务来获得；

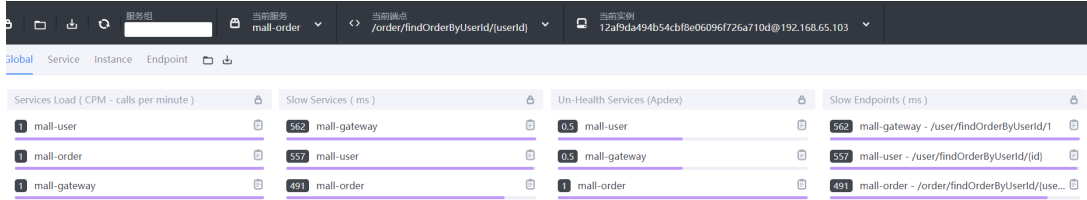
访问：<http://192.168.3.100:8080/>



页面的右下角可以中英文切换，可以切换选择要展示的时间区间的跟踪数据。

2.3 SkyWalking中三个概念

- 服务(Service)**：表示对请求提供相同行为的一系列或一组工作负载，在使用Agent时，可以定义服务的名字；
- 服务实例(Service Instance)**：上述的一组工作负载中的每一个工作负载称为一个实例，一个服务实例实际就是操作系统上的一个真实进程；
- 端点(Endpoint)**：对于特定服务所接收的请求路径, 如HTTP的URI路径和gRPC服务的类名 + 方法签名；



3、SkyWalking 接入微服务

3.1 linux环境—通过jar包方式接入

准备一个springboot程序，打成可执行jar包，写一个shell脚本，在启动项目的Shell脚本上，通过 `-javaagent` 参数进行配置SkyWalking Agent来跟踪微服务；
startup.sh脚本：

```

1 #!/bin/sh
2 # SkyWalking Agent配置
3 export SW_AGENT_NAME=springboot-skywalking-demo #Agent名字, 一般使用`spring.application.name`
4 export SW_AGENT_COLLECTOR_BACKEND_SERVICES=127.0.0.1:11800 #配置 Collector 地址。
5 export SW_AGENT_SPAN_LIMIT=2000 #配置链路的最大Span数量, 默认为 300。
6 export JAVA_AGENT=-javaagent:/usr/local/soft/apache-skywalking-apm-bin-es7/agent/skywalking-agent.jar

```

启动日志

```

[root@redis skywalking-demo-app]# sh startup.sh
DEBUG 2021-02-21 16:14:40:045 main AgentPackagePath : The beacon class location is jar:file:
/usr/local/soft/apache-skywalking-apm-bin-es7/agent/skywalking-agent.jar!/org/apache/skywalk
ing/apm/agent/core/boot/AgentPackagePath.class.
INFO 2021-02-21 16:14:40:047 main SnifferConfigInitializer : Config file found in /usr/local
/soft/apache-skywalking-apm-bin-es7/agent/config/agent.config.
agent配置文件

:: Spring Boot :: (v2.3.2.RELEASE)

2021-02-21 16:14:45.792 INFO 2336 --- [main] .m.s.SpringbootSkywalkingDemoApplic
ation : Starting SpringbootSkywalkingDemoApplication v0.0.1-SNAPSHOT on redis with PID 2336
(/usr/local/soft/skywalking-demo-app/springboot-skywalking-demo-0.0.1-SNAPSHOT.jar started b
y root in /usr/local/soft/skywalking-demo-app)

```

等同于

```

1 java -javaagent:/usr/local/soft/apache-skywalking-apm-bin-es7/agent/skywalking-agent.jar
2 -DSW_AGENT_COLLECTOR_BACKEND_SERVICES=127.0.0.1:11800

```

参数名对应agent/config/agent.config配置文件中的属性。

属性对应的源码: org.apache.skywalking.apm.agent.core.conf.Config.java

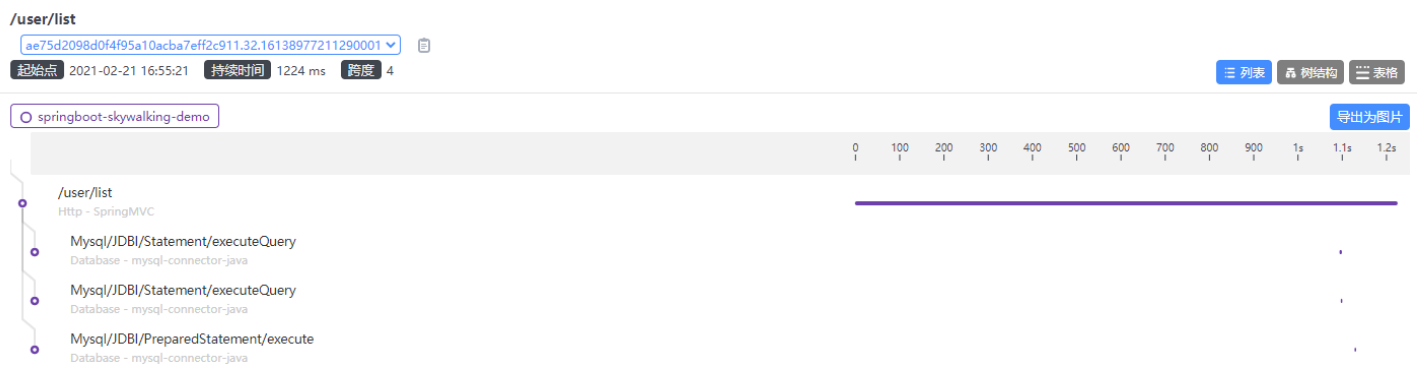
```

1 # The service name in UI
2 agent.service_name=${SW_AGENT_NAME:Your_ApplicationName}
3 # Backend service addresses.

```

我们也可以使用skywalking.+配置文件中的配置名作为系统配置项来进行覆盖。 javaagent参数配置方式优先级更高

测试: 访问你的微服务



3.2 windos环境—在IDEA中使用Skywalking

在运行的程序配置jvm参数, 如下图所示:

Name: ☐ Share ☒ Single instance

Configuration Code Coverage Logs

Main class:

Environment

VM options: 指定agent.jar所在位置

Program arguments: 指定服务名字

Working directory: 指定aollector连接地址

Environment variables:

Use classpath of module:

```

1 # skywalking-agent.jar的本地磁盘的路径
2 -javaagent:D:\apache\apache-skywalking-apm-es7-8.4.0\apache-skywalking-apm-bin-es7\agent\skywalking-agent.jar
3 # 在skywalking上显示的服务名
4 -DSW_AGENT_NAME=springboot-skywalking-demo
5 # skywalking的collector服务的IP及端口

```

-DSW_AGENT_COLLECTOR_BACKEND_SERVICES 可以指定远程地址，但是-javaagent必须绑定你本机物理路径的skywalking-agent.jar

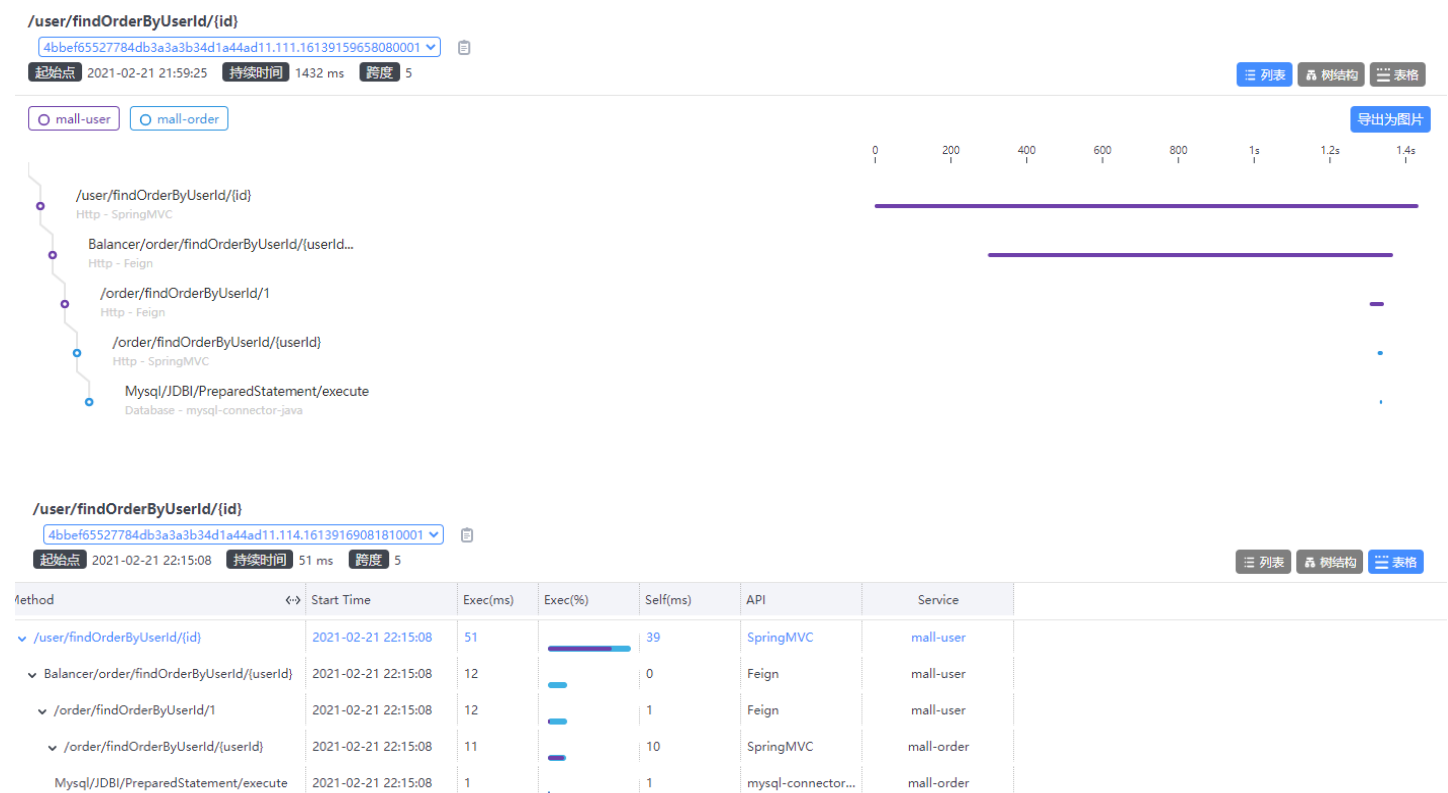
3.3 Skywalking跨多个微服务跟踪

Skywalking跨多个微服务跟踪，只需要每个微服务启动时添加javaagent参数即可。

测试：

启动微服务mall-gateway，mall-order，mall-user，配置skywalking的jvm参数

<http://localhost:8888/user/findOrderByUserId/1>






注意：此处存在bug，跟踪链路不显示gateway

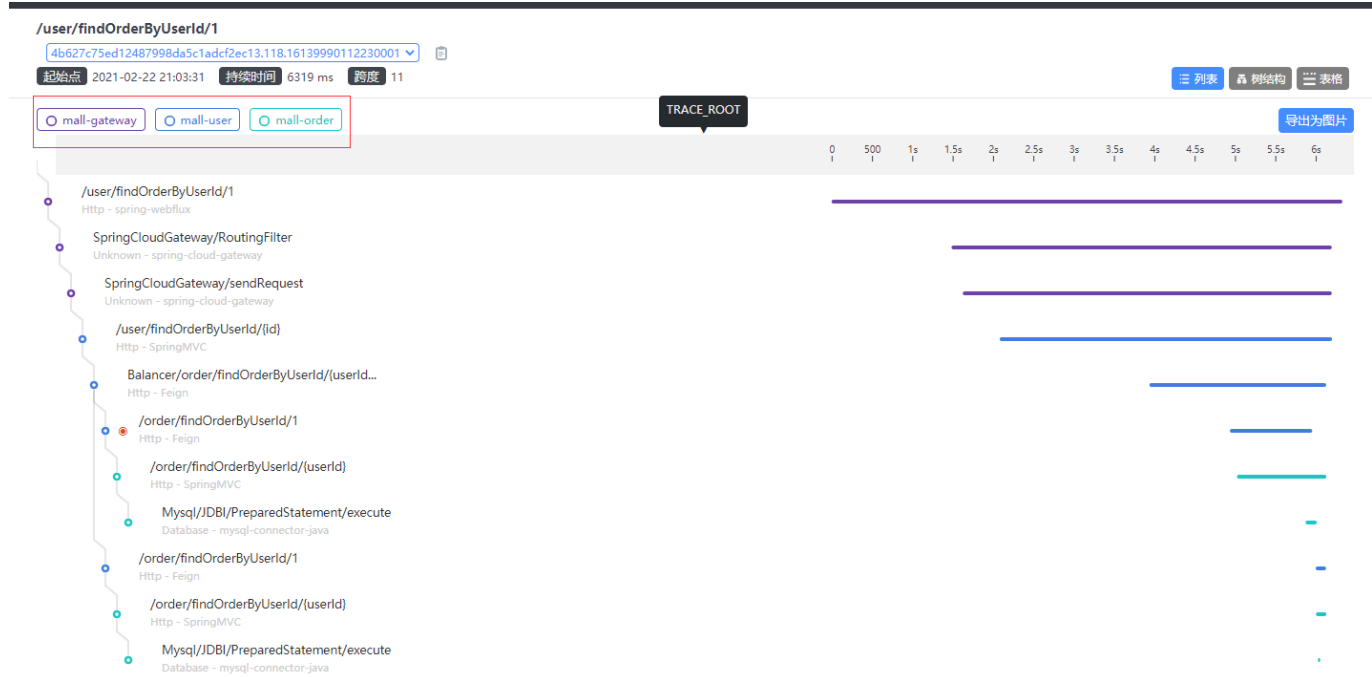
拷贝agent/optional-plugins目录下的gateway插件到agent/plugins目录

```
[root@redis apache-skywalking-apm-bin-es7]# ls
agent bin config LICENSE licenses logs NOTICE oap-libs README.txt tools webapp
[root@redis apache-skywalking-apm-bin-es7]# cd agent/
[root@redis agent]# ls
activations config optional-plugins plugins
bootstrap-plugins logs optional-reporter-plugins skywalking-agent.jar
[root@redis agent]# ls optional-plugins/
apm-customize-enhance-plugin-8.3.0.jar          apm-spring-cloud-gateway-2.1.x-plugin-8.3.0.jar
apm-gson-2.x-plugin-8.3.0.jar                  apm-spring-tx-plugin-8.3.0.jar
apm-kotlin-coroutine-plugin-8.3.0.jar          apm-spring-webflux-5.x-plugin-8.3.0.jar
apm-quartz-scheduler-2.x-plugin-8.3.0.jar      apm-trace-ignore-plugin-8.3.0.jar
apm-spring-annotation-plugin-8.3.0.jar        apm-zookeeper-3.4.x-plugin-8.3.0.jar
apm-spring-cloud-gateway-2.0.x-plugin-8.3.0.jar
[root@redis agent]# cp optional-plugins/apm-spring-cloud-gateway-2.1.x-plugin-8.3.0.jar plugins/
```

拷贝gateway插件到
plugins

DATA (D:) > apache > apache-skywalking-apm-es7-8.3.0 > apache-skywalking-apm-bin-es7 > agent > plugins

名称	修改日期	类型	大小
 apm-spring-cloud-feign-2.x-plugin-8...	2020/11/29 14:41	Executable Jar File	12 KB
 apm-spring-cloud-gateway-2.1.x-plu...	2020/11/29 15:12	Executable Jar File	43 KB
 apm-spring-concurrent-util-4.x-plugi...	2020/11/29 14:40	Executable Jar File	28 KB
 apm-spring-core-patch-8.3.0.jar	2020/11/29 14:42	Executable Jar File	28 KB



/user/findOrderByUserId/1							
4b627c75ed12487998da5c1adcf2ec13.118.16139990112230001							
起始点 2021-02-22 21:03:31 持续时间 6319 ms 跨度 11							
Method	Start Time	Exec(ms)	Exec(%)	Self(ms)	API	Service	
✓ /user/findOrderByUserId/1	2021-02-22 21:03:31	6319	<div></div>	1618	spring-webflux	mall-gateway	
✓ SpringCloudGateway/RouterFilter	2021-02-22 21:03:32	4701	<div></div>	140	spring-cloud-gat...	mall-gateway	
✓ SpringCloudGateway/sendRequest	2021-02-22 21:03:32	4561	<div></div>	454	spring-cloud-gat...	mall-gateway	
✓ /user/findOrderByUserId/{id}	2021-02-22 21:03:33	4107	<div></div>	1929	SpringMVC	mall-user	
✓ Balancer/order/findOrderByUserId/{...}	2021-02-22 21:03:35	2178	<div></div>	1055	Feign	mall-user	
✓ /order/findOrderByUserId/1	2021-02-22 21:03:36	1008	<div></div>	0	Feign	mall-user	
✓ /order/findOrderByUserId/{userId}	2021-02-22 21:03:36	1096	<div></div>	970	SpringMVC	mall-order	
Mysql/JDBI/PreparedStatement/ex...	2021-02-22 21:03:37	126	<div></div>	126	mysql-connector...	mall-order	
✓ /order/findOrderByUserId/1	2021-02-22 21:03:37	115	<div></div>	0	Feign	mall-user	
✓ /order/findOrderByUserId/{userId}	2021-02-22 21:03:37	116	<div></div>	99	SpringMVC	mall-order	
Mysql/JDBI/PreparedStatement/ex...	2021-02-22 21:03:37	17	<div></div>	17	mysql-connector...	mall-order	

4、Skywalking持久化跟踪数据

默认使用的H2数据库存储

config/application.yml

```
storage:
  selector: ${SW_STORAGE:h2}
  elasticsearch:
```

4.1 基于mysql持久化:

1. 修改config目录下的application.yml，使用mysql作为持久化存储的仓库

```
storage:
  selector: ${SW_STORAGE:mysql} ← 默认h2,改为mysql
  elasticsearch:
```

2. 修改mysql连接配置

```
mysql:
  properties:
    jdbcUrl: ${SW_JDBC_URL:"jdbc:mysql://localhost:3306/swtest"}
    dataSource.user: ${SW_DATA_SOURCE_USER:root}
    dataSource.password: ${SW_DATA_SOURCE_PASSWORD:root} ← 修改mysql配置
    dataSource.cachePrepStmts: ${SW_DATA_SOURCE_CACHE_PREP_STMTS:true}
    dataSource.prepStmtCacheSize: ${SW_DATA_SOURCE_PREP_STMT_CACHE_SQL_SIZE:250}
    dataSource.prepStmtCacheSqlLimit: ${SW_DATA_SOURCE_PREP_STMT_CACHE_SQL_LIMIT:2048}
    dataSource.useServerPrepStmts: ${SW_DATA_SOURCE_USE_SERVER_PREP_STMTS:true}
    metadataQueryMaxSize: ${SW_STORAGE_MYSQL_QUERY_MAX_SIZE:5000}
    maxSizeOfArrayColumn: ${SW_STORAGE_MAX_SIZE_OF_ARRAY_COLUMN:20}
    numOfSearchableValuesPerTag: ${SW_STORAGE_NUM_OF_SEARCHABLE_VALUES_PER_TAG:2}
```

```
1 storage:
2   #选择使用mysql 默认使用h2，不会持久化，重启skyWalking之前的数据会丢失
3   selector: ${SW_STORAGE:mysql}
4   #使用mysql作为持久化存储的仓库
5   mysql:
6     properties:
7       #数据库连接地址
8       jdbcUrl: ${SW_JDBC_URL:"jdbc:mysql://localhost:3306/swtest"}
9       #用户名
10      dataSource.user: ${SW_DATA_SOURCE_USER:root}
11      #密码
```

注意：需要添加mysql数据驱动包，因为在lib目录下是没有mysql数据驱动包的，所以修改完配置启动是会报错，启动失败的。

```
[root@redis apache-skywalking-apm-bin-es7]# tail -f logs/skywalking-oap-server.log
at org.apache.skywalking.oap.server.library.client.jdbc.hikaricp.JDBCHikariCPClient.connect(JDBCHikariCPClient.java:54) ~[library-client-8.3.0.jar:8.3.0]
at org.apache.skywalking.oap.server.storage.plugin.jdbc.mysql.MySQLStorageProvider.start(MySQLStorageProvider.java:152) ~[storage-jdbc-hikaricp-plugin-8.3.0.jar:8.3.0]
at org.apache.skywalking.oap.server.library.module.BootstrapFlow.start(BootstrapFlow.java:49) ~[library-module-8.3.0.jar:8.3.0]
at org.apache.skywalking.oap.server.library.module.ModuleManager.init(ModuleManager.java:62) ~[library-module-8.3.0.jar:8.3.0]
at org.apache.skywalking.oap.server.starter.OAPServerBootstrap.start(OAPServerBootstrap.java:43) [server-bootstrap-8.3.0.jar:8.3.0]
at org.apache.skywalking.oap.server.starter.OAPServerStartUp.main(OAPServerStartUp.java:27) [server-starter-es7-8.3.0.jar:8.3.0]
Caused by: java.sql.SQLException: No suitable driver
at java.sql.DriverManager.getDriver(DriverManager.java:315) ~[?:1.8.0_181]
at com.zaxxer.hikari.util.DriverDataSource.<init>(DriverDataSource.java:103) ~[HikariCP-3.1.0.jar:?]
... 10 more
```

3. 添加mysql数据驱动包到oap-lib目录下

```
mv 12-2.4.8.Final.jar
mysql-connector-java-8.0.17.jar
nacos-api-1.3.1.jar
```

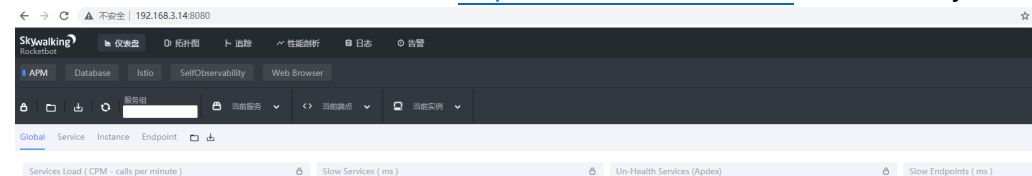
4. 启动Skywalking

```
[root@redis oap-lib]# cd ..
[root@redis apache-skywalking-apm-bin-es7]# bin/startup.sh
SkyWalking OAP started successfully!
SkyWalking Web Application started successfully!
[root@redis apache-skywalking-apm-bin-es7]# jps
736 QuorumPeerMain
2296 Jps
2252 OAPServerStartUp
2269 skywalking-webapp.jar
[root@redis apache-skywalking-apm-bin-es7]#
```

查看swtest数据库，可以看到生成了很多表。

alarm_record	browser_app_page_trans_avg	http_access_log	segment
all_heatmap	browser_app_page_ttfb_avg	instance_clr_available_completion_port_threads	service_apdex
all_percentile	browser_app_page_ttl_avg	instance_clr_available_worker_threads	service_cpm
browser_app_error_rate	browser_app_page_ttl_percentile	instance_clr_cpu	service_instance_cpm
browser_app_error_sum	browser_app_page_unknown_error_sum	instance_clr_gen0_collect_count	service_instance_relation_client_call
browser_app_page_ajax_error_sum	browser_app_pv	instance_clr_gen1_collect_count	service_instance_relation_client_cpm
browser_app_page_dns_avg	browser_app_single_version_error_rate	instance_clr_gen2_collect_count	service_instance_relation_client_perc
browser_app_page_dom_analysis_avg	browser_app_single_version_error_sum	instance_clr_heap_memory	service_instance_relation_client_resp
browser_app_page_dom_ready_avg	browser_app_single_version_pv	instance_clr_max_completion_port_threads	service_instance_relation_client_side
browser_app_page_dom_ready_percentile	browser_error_log	instance_clr_max_worker_threads	service_instance_relation_server_call
browser_app_page_error_rate	database_access_cpm	instance_jvm_cpu	service_instance_relation_server_cpm
browser_app_page_error_sum	database_access_percentile	instance_jvm_memory_heap	service_instance_relation_server_per
browser_app_page_first_pack_avg	database_access_res_time	instance_jvm_memory_heap_max	service_instance_relation_server_resq
browser_app_page_first_pack_percentile	database_access_sla	instance_jvm_memory_noheap	service_instance_relation_server_side
browser_app_page_fmp_percentile	endpoint_avg	instance_jvm_memory_noheap_max	service_instance_resptime
browser_app_page_fmp_percentile	endpoint_cpm	instance_jvm_old_gc_count	service_instance_sla
browser_app_page_fpt_avg	endpoint_percentile	instance_jvm_old_gc_time	service_percentile
browser_app_page_fpt_percentile	endpoint_relation_cpm	instance_jvm_thread_daemon_count	service_relation_client_call_sla
browser_app_page_js_error_sum	endpoint_relation_percentile	instance_jvm_thread_live_count	service_relation_client_cpm
browser_app_page_load_page_avg	endpoint_relation_resptime	instance_jvm_thread_peak_count	service_relation_client_percentile
browser_app_page_load_page_percentile	endpoint_relation_server_side	instance_jvm_young_gc_count	service_relation_client_resptime
browser_app_page_pv	endpoint_relation_sla	instance_jvm_young_gc_time	service_relation_client_side
browser_app_page_redirect_avg	endpoint_sla	instance_traffic	service_relation_server_call_sla
browser_app_page_res_avg	endpoint_traffic	network_address_alias	service_relation_server_cpm
browser_app_page_resource_error_sum	envoy_heap_memory_max_used	profile_task	service_relation_server_percentile
browser_app_page_ssl_avg	envoy_parent_connections_used	profile_task_log	service_relation_server_resptime

说明启动成功了，打开配置对应的地址<http://192.168.3.100:8080/>，可以看到skywalking的web界面。



测试：重启skywalking，验证跟踪数据会不会丢失

5、自定义SkyWalking链路追踪

如果我们希望对项目中的业务方法，实现链路追踪，方便我们排查问题，可以使用如下的代码引入依赖

```
1 <!-- SkyWalking 工具类 -->
2 <dependency>
3     <groupId>org.apache.skywalking</groupId>
4     <artifactId>apm-toolkit-trace</artifactId>
```

5.1 @Trace将方法加入追踪链路

如果一个业务方法想在ui界面的跟踪链路上显示出来，只需要在业务方法上加上@Trace注解即可

@Service

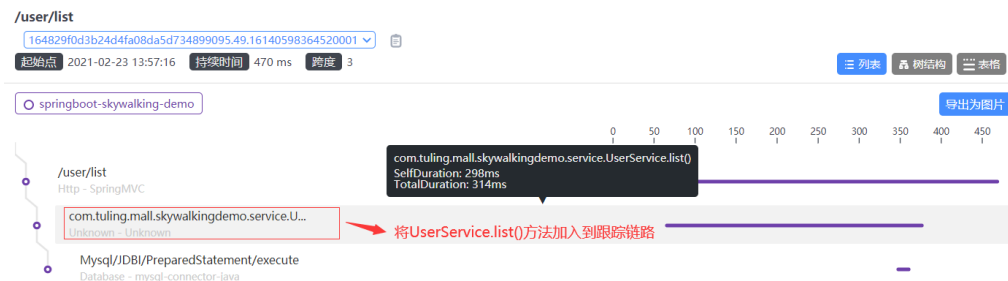
```
public class UserService {
    @Autowired
    private UserMapper userMapper;

    @Trace
    public List<User> list(){
        return userMapper.list();
    }
}
```

@Trace

UserService中加入@Trace

测试:



5.2 加入@Tags或@Tag

我们还可以为追踪链路增加其他额外的信息，比如记录参数和返回信息。实现方式：在方法上增加@Tag或者@Tags。

@Tag 注解中 key = 方法名； value = returnedObj 返回值 arg[0] 参数

```
1 @Trace
2 @Tag(key = "list", value = "returnedObj")
3 public List<User> list(){
4     return userMapper.list();
5 }
6
7 @Trace
8 @Tags({@Tag(key = "param", value = "arg[0]"),
9         @Tag(key = "user", value = "returnedObj")})
10 public User getById(Integer id){
11     return userMapper.getById(id);
12 }
```

跨度信息

标记.

服务:	springboot-skywalking-demo
端点:	com.tuling.mall.skywalkingdemo.service.UserService.list()
跨度类型:	Local
组件:	Unknown
Peer:	No Peer
失败:	false
list:	[User(id=1, name=fox), User(id=2, name=monkey)]

跨度信息

标记.

服务:	springboot-skywalking-demo
端点:	com.tuling.mall.skywalkingdemo.service.UserService.getById(java.lang.Integer)
跨度类型:	Local
组件:	Unknown
Peer:	No Peer
失败:	false
param:	1
user:	User(id=1, name=fox)

性能分析

skywalking的性能分析，在根据服务名称、端点名称，以及相应的规则建立了任务列表后，在调用了此任务列表的端点后。skywalking会自动记录，剖析当前端口，生成剖析结果，具体流程如图：

性能剖析

新建任务

1. 性能剖析，最开始这个菜单里是没有内容的
需要添加需要剖析的接口

2. 点击新建任务

Task List

./auth/token

2020-08-10 14:44:52

2020-08-10 14:45:12

查看详情

./auth/token

2020-08-10 12:04:42

2020-08-10 12:09:42

查看详情

Sampled Traces

./auth/token

112 ms

2020-08-10 14:45:14

./auth/token

109 ms

2020-08-10 14:45:14

./auth/token

109 ms

2020-08-10 14:45:14

./auth/token

109 ms

2020-08-10 14:45:13

./auth/token

109 ms

2020-08-10 14:45:13

Open

Start Time

Exec(ms)

Exec(N)

Self(ms)

API

Service

Operation

./auth/token

2020-08-10 14:45:14

172

190

Tomcat

testuaa1112

MySql/JDBC/PreparedStatement/execute

Letucue/GET/complete

2020-08-10 14:45:14

0

0

Letucue

testuaa1112

Letucue/GET/complete

Letucue/GET/complete

2020-08-10 14:45:14

0

0

Letucue

testuaa1112

Letucue/GET/complete

Letucue/GET/complete

2020-08-10 14:45:14

0

0

Letucue

testuaa1112

Letucue/GET/complete

Letucue/GET/complete

2020-08-10 14:45:14

0

0

Letucue

testuaa1112

Letucue/GET/complete

Thread Stack

Duration (ms)

Self Duration (ms)

Dump Count

java.lang.Thread.run:740

150

0

15

org.apache.tomcat.util.threads.TaskThread\$WrappingRunnable.run:1

150

0

15

java.util.concurrent.ThreadPoolExecutor\$Worker.run:24

150

0

15

org.apache.tomcat.util.net.SocketProcessorBase.run:49

150

0

15

org.apache.tomcat.util.net.NioEndpoint\$SocketProcessor.doRun:1589

150

0

15

org.apache.coyote.AbstractProtocol\$ConnectionHandler.process:888

150

0

15

org.apache.coyote.AbstractProtocol\$LightProcessors:95

150

0

15

org.apache.coyote.http11.Http11Processor.service:375

150

0

15

org.apache.catalina.connector.CoyoteAdapter.service:345

150

0

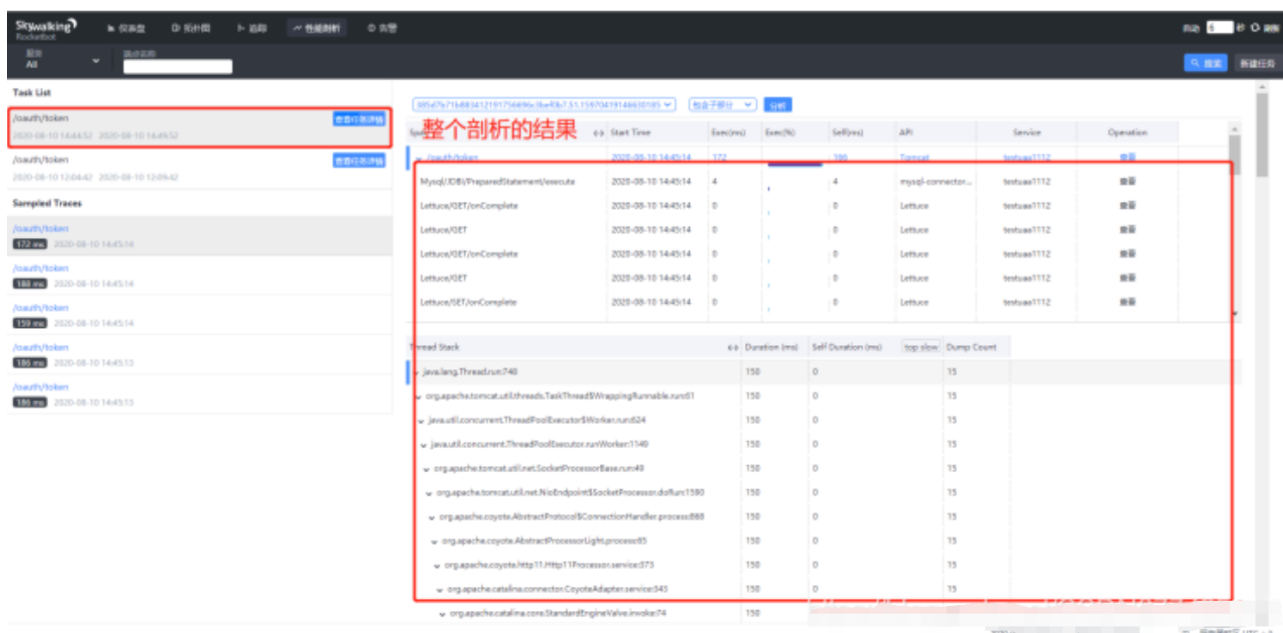
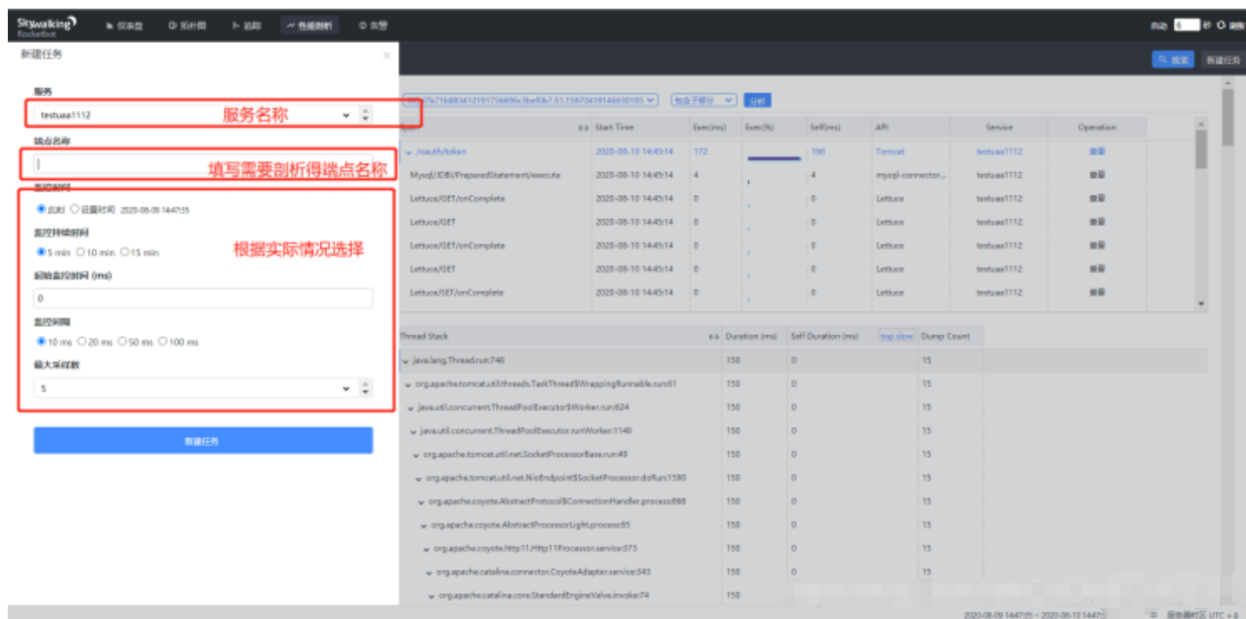
15

org.apache.catalina.core.StandardEngineValve.invoke:74

150

0

15



6、Skywalking集成日志框架

[logback官方配置](#)

[log4j官方配置](#)

[log4j2官方配置](#)

引入依赖

```
1 <!-- apm-toolkit-logback-1.x -->
2 <dependency>
3     <groupId>org.apache.skywalking</groupId>
4     <artifactId>apm-toolkit-logback-1.x</artifactId>
5     <version>8.5.0</version>
```

添加logback-spring.xml文件，并配置 %tid 占位符

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration>
3     <!-- 引入 Spring Boot 默认的 logback XML 配置文件 -->
4     <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
5
```



```

6
7 <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
8 <!-- 日志的格式化 -->
9 <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
10 <layout class="org.apache.skywalking.apm.toolkit.log.logback.v1.x.TraceIdPatternLogbackLayout">
11 <Pattern>${CONSOLE_LOG_PATTERN}</Pattern>
12 </layout>
13 </encoder>
14
15 </appender>
16
17 <!-- 设置 Appender -->
18 <root level="INFO">
19 <appender-ref ref="console"/>
20 </root>
21
22 </configuration>

```

测试

```

2021-02-23 15:40:59.863 INFO 6860 TID:4fc89b8d6add43d2a3dc76f023e6a329.49.16140660598470001 --- [nio-8000-
2021-02-23 15:40:59.864 INFO 6860 TID:4fc89b8d6add43d2a3dc76f023e6a329.49.16140660598470001 --- [nio-8000-
2021-02-23 15:40:59.873 INFO 6860 TID:4fc89b8d6add43d2a3dc76f023e6a329.49.16140660598470001 --- [nio-8000-
2021-02-23 15:40:59.928 INFO 6860 TID:4fc89b8d6add43d2a3dc76f023e6a329.49.16140660598470001 --- [nio-8000-
2021-02-23 15:40:59.930 INFO 6860 TID:4fc89b8d6add43d2a3dc76f023e6a329.49.16140660598470001 --- [nio-8000-
2021-02-23 15:40:59.954 INFO 6860 TID:4fc89b8d6add43d2a3dc76f023e6a329.49.16140660598470001 --- [nio-8000-
2021-02-23 15:41:00.147 INFO 6860 TID:4fc89b8d6add43d2a3dc76f023e6a329.49.16140660598470001 --- [nio-8000-

```

Skywalking通过grpc上报日志 (需要v8.4.0+)

gRPC报告程序可以将收集到的日志转发到SkyWalking OAP服务器上

logback-spring.xml中添加

```

1 <!-- v8.4.0提供 -->
2 <appender name="grpc-log" class="org.apache.skywalking.apm.toolkit.log.logback.v1.x.log.GRPCLogClientApp
3 <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
4 <layout class="org.apache.skywalking.apm.toolkit.log.logback.v1.x.mdc.TraceIdMDCPatternLogbackLa
5 <Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%X{tid}] [%thread] %-5level %logger{36} -%msg%n</Patte
6 </layout>
7 </encoder>
8 </appender>
9
10 <root level="info">
11 <appender-ref ref="grpc-log" />

```

```

<appender name="grpc-log" class="org.apache.skywalking.apm.toolkit.log.logback.v1.x.log
.GRPCLogClientAppender"></appender>

<!-- 设置 Appender -->
<root level="INFO">
<appender-ref ref="console"/>
<appender-ref ref="file"/>
<appender-ref ref="grpc-log" />
</root>

```

打开agent/config/agent.config配置文件，添加如下配置信息：

```
1 plugin.toolkit.log.grpc.reporter.server_host=${SW_GRPC_LOG_SERVER_HOST:192.168.3.100}
2 plugin.toolkit.log.grpc.reporter.server_port=${SW_GRPC_LOG_SERVER_PORT:11800}
3 plugin.toolkit.log.grpc.reporter.max_message_size=${SW_GRPC_LOG_MAX_MESSAGE_SIZE:10485760}
```

以上配置是默认配置信息,agent与oap在本地的可以不配

配置名	解释	默认值
plugin.toolkit.log.transmit_formatted	是否以格式化或未格式化的格式传输记录的数据	true
plugin.toolkit.log.grpc.reporter.server_host	指定要向其报告日志数据的grpc服务器的主机	127.0.0.1
plugin.toolkit.log.grpc.reporter.server_port	指定要向其报告日志数据的grpc服务器的端口	11800
plugin.toolkit.log.grpc.reporter.max_message_size	指定grpc客户端要报告的日志数据的最大大小	10485760
plugin.toolkit.log.grpc.reporter.upstream_timeout	客户端向上游发送数据时将超时多长时间。单位是秒	30

agent配置信息大全

Skywalking UI效果

skywalking

Rocketbot

仪表盘

拓扑图

追踪

性能剖析

日志

告警

日志类别

Service

▼

服务

All

▼

当前实例

All

▼

当前端点

All

▼

更多条件

搜索

清空

1

/ 3

追踪ID:

内容关键词:

内容不包含的关键词:

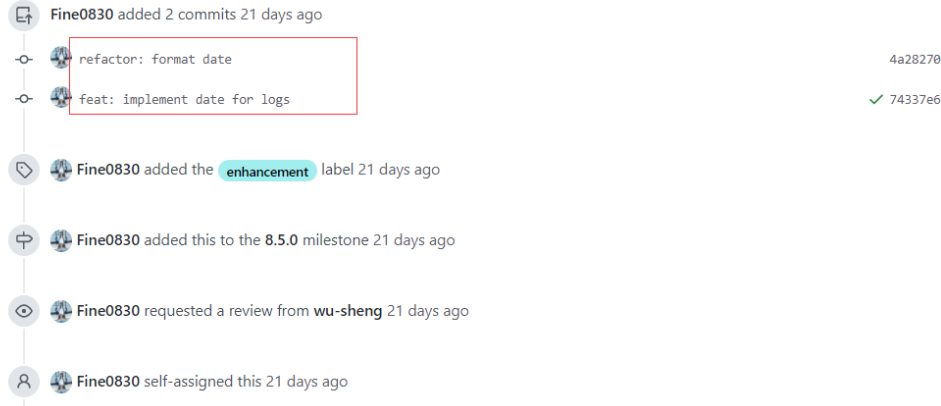
标记:

配置词汇总

当前服务	当前实例	时间	内容类型	标记	内容	追踪ID
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 15:12:05	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	traceld = a35abd7eb56d47b69a926d215dd6b67...	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 15:12:05	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	name = fox	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 15:17:08	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	name = fox	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 15:17:08	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	traceld = a35abd7eb56d47b69a926d215dd6b67...	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 14:59:00	TEXT	level=INFO,logger=com.zaxxer.hikari.HikariData...	HikariPool-1 - Start completed.	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 14:56:04	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	name = fox	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 14:56:04	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	traceld = a35abd7eb56d47b69a926d215dd6b67...	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 14:36:02	TEXT	level=INFO,logger=com.zaxxer.hikari.HikariData...	HikariPool-1 - Starting...	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 14:33:08	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	traceld = a35abd7eb56d47b69a926d215dd6b67...	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 14:33:07	TEXT	level=INFO,logger=com.tuling.mall.skywalkingd...	name = fox	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 13:26:00	TEXT	level=INFO,logger=org.springframework.web.se...	Completed initialization in 6 ms	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 13:25:04	TEXT	level=INFO,logger=org.springframework.web.se...	Initializing Servlet 'dispatcherServlet'	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 13:25:02	TEXT	level=INFO,logger=org.apache.catalina.core.Co...	Initializing Spring DispatcherServlet 'dispatcherS...	a35abd7eb56d47b69a926d215dd6b67...
springboot-skywalking-demo	ed96ea665fa7423ebbbe47a7...	1614-10-27 13:23:04	TEXT	level=INFO,logger=org.apache.tomcat.util.http...	A cookie header was received [161217935; use...	a35abd7eb56d47b69a926d215dd6b67...

此处日期格式存在问题

<https://github.com/apache/skywalking-rocketbot-ui/pull/428>



7、SkyWalking 告警功能

SkyWalking 告警功能是在6.x版本新增的，其核心由一组规则驱动，这些规则定义在`config/alarm-settings.yml`文件中。告警规则的定义分为两部分：

1. [告警规则](#)：它们定义了应该如何触发度量警报，应该考虑什么条件。
2. [Webhook \(网络钩子\)](#)：定义当警告触发时，哪些服务终端需要被告知

告警规则

SkyWalking 的发行版都会默认提供`config/alarm-settings.yml`文件，里面预先定义了一些常用的告警规则。如下：

1. 过去 3 分钟内服务平均响应时间超过 1 秒。
2. 过去 2 分钟服务成功率低于80%。
3. 过去 3 分钟内服务响应时间超过 1s 的百分比
4. 服务实例在过去 2 分钟内平均响应时间超过 1s，并且实例名称与正则表达式匹配。
5. 过去 2 分钟内端点平均响应时间超过 1 秒。
6. 过去 2 分钟内数据库访问平均响应时间超过 1 秒。
7. 过去 2 分钟内端点关系平均响应时间超过 1 秒。

这些预定义的告警规则，打开`config/alarm-settings.yml`文件即可看到

告警规则配置项的说明：

- **Rule name**：规则名称，也是在告警信息中显示的唯一名称。必须以`_rule`结尾，前缀可自定义
- **Metrics name**：度量名称，取值为oal脚本中的度量名，目前只支持long、double和int类型。详见[Official OAL script](#)
- **Include names**：该规则作用于哪些实体名称，比如服务名，终端名（可选，默认为全部）
- **Exclude names**：该规则作不用于哪些实体名称，比如服务名，终端名（可选，默认为空）
- **Threshold**：阈值
- **OP**：操作符，目前支持 >、<、=
- **Period**：多久告警规则需要被核实一下。这是一个时间窗口，与后端部署环境时间相匹配
- **Count**：在一个Period窗口中，如果values超过Threshold值（按op），达到Count值，需要发送警报
- **Silence period**：在时间N中触发报警后，在TN -> TN + period这个阶段不告警。默认情况下，它和Period一样，这意味着相同的告警（在同一个Metrics name拥有相同的Id）在同一个Period内只会触发一次
- **message**：告警消息

Webhook (网络钩子)

Webhook可以简单理解为是一种Web层面的回调机制，通常由一些事件触发，与代码中的事件回调类似，只不过是Web层面的。由于是Web层面的，所以当事件发生时，回调的不再是代码中的方法或函数，而是服务接口。例如，在告警这个场景，告警就是一个事件。当该事件发生时，SkyWalking就会主动去调用一个配置好的接口，该接口就是所谓的Webhook。SkyWalking的告警消息会通过 HTTP 请求进行发送，请求方法为 POST，Content-Type 为 application/json，其JSON 数据实基于`List<org.apache.skywalking.oap.server.core.alarm.AlarmMessage>`进行序列化的。JSON数据示例：

```
1  [{
2    "scopeId": 1,
3    "scope": "SERVICE",
4    "name": "serviceA",
5    "id0": "12",
6    "id1": "",
7    "ruleName": "service_resp_time_rule",
8    "alarmMessage": "alarmMessage xxxx",
9    "startTime": 1560524171000
10 }, {
11    "scopeId": 1,
```

```
12     "scope": "SERVICE",
13     "name": "serviceB",
14     "id0": "23",
15     "id1": "",
16     "ruleName": "service_resp_time_rule",
17     "alarmMessage": "alarmMessage yyy",
18     "startTime": 1560524171000
19 }]
```

字段说明：

- **scopeId**、**scope**：所有可用的 Scope 详见 `org.apache.skywalking.oap.server.core.source.DefaultScopeDefine`
- **name**：目标 Scope 的实体名称
- **id0**：Scope 实体的 ID
- **id1**：保留字段，目前暂未使用
- **ruleName**：告警规则名称
- **alarmMessage**：告警消息内容
- **startTime**：告警时间，格式为时间戳

邮件告警功能实践

根据以上两个小节介绍，可以得知：SkyWalking是不支持直接向邮箱、短信等服务发送告警信息的，SkyWalking只会在发生告警时将告警信息发送至配置好的Webhook接口。

但我们总不能人工盯着该接口的日志信息来得知服务是否发生了告警，因此我们需要在该接口里实现发送邮件或短信等功能，从而达到个性化的告警通知。

接下来开始动手实践，这里基于Spring Boot进行实现。首先是添加依赖：

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-mail</artifactId>
4 </dependency>
```

配置邮箱服务：

```
1 server:
2     port: 9134
3
4 #邮箱配置
5 spring:
6     mail:
7         host: smtp.163.com
8         #发送者邮箱账号
9         username: 你的邮箱@163.com
10        #发送者密钥
11        password: 你的邮箱服务密钥
12        default-encoding: utf-8
13        port: 465    #端口号465或587
14        protocol: smtp
15        properties:
16            mail:
17                debug:
```

```
18         false
19         smtp:
20         socketFactory:
21             class: javax.net.ssl.SSLSocketFactory
```

根据SkyWalking发送的JSON数据定义一个DTO，用于接口接收数据：

```
1 @Data
2 public class SwAlarmDTO {
3
4     private Integer scopeId;
5     private String scope;
6     private String name;
7     private Integer id0;
8     private Integer id1;
9     private String ruleName;
10    private String alarmMessage;
11    private Long startTime;
12 }
```

接着定义一个接口，实现接收SkyWalking的告警通知，并将数据发送至邮箱：

```
1
2 @Slf4j
3 @RestController
4 @RequiredArgsConstructor
5 @RequestMapping("/alarm")
6 public class SwAlarmController {
7
8     private final JavaMailSender sender;
9
10    @Value("${spring.mail.username}")
11    private String from;
12
13    /**
14     * 接收skywalking服务的告警通知并发送至邮箱
15     */
16    @PostMapping("/receive")
17    public void receive(@RequestBody List<SwAlarmDTO> alarmList) {
18        SimpleMailMessage message = new SimpleMailMessage();
19        // 发送者邮箱
20        message.setFrom(from);
21        // 接收者邮箱
22        message.setTo(from);
23        // 主题
24        message.setSubject("告警邮件");
25        String content = getContent(alarmList);
26        // 邮件内容
27        message.setText(content);
```

```

28     sender.send(message);
29     log.info("告警邮件已发送...");
30 }
31
32 private String getContent(List<SwAlarmDTO> alarmList) {
33     StringBuilder sb = new StringBuilder();
34     for (SwAlarmDTO dto : alarmList) {
35         sb.append("scopeId: ").append(dto.getScopeId())
36             .append("\nscope: ").append(dto.getScope())
37             .append("\n目标 Scope 的实体名称: ").append(dto.getName())
38             .append("\nScope 实体的 ID: ").append(dto.getId0())
39             .append("\nid1: ").append(dto.getId1())
40             .append("\n告警规则名称: ").append(dto.getRuleName())
41             .append("\n告警消息内容: ").append(dto.getAlarmMessage())
42             .append("\n告警时间: ").append(dto.getStartTime())
43             .append("\n\n-----\n\n");
44     }
45
46     return sb.toString();
47 }
48 }

```

最后将该接口配置到SkyWalking中，Webhook的配置位于config/alarm-settings.yml文件的末尾，格式为http://{ip}:{port}/{uri}。如下示例：

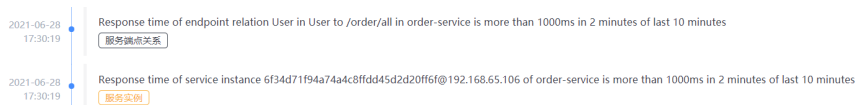
```

1 [root@ip-236-048 skywalking]# vim config/alarm-settings.yml
2 webhooks:
3   - http://127.0.0.1:9134/alarm/receive

```

测试告警功能

完成告警接口的开发及配置后，我们来进行一个简单的测试。这里有一条调用链路如下：



我在producer接口中增加了一行睡2秒的代码：

```

1 @Override
2 @Trace
3 @Tag(key="getAll", value="returnedObj")
4 public List<Order> all() throws InterruptedException {
5     TimeUnit.SECONDS.sleep(2);
6     return orderMapper.selectAll();
7 }

```


<http://localhost:8072/order/all>

执行完测试代码，等待约两分钟后，告警接口的控制台输出了一段日志信息：

```
2021-06-28 17:36:19.191 INFO 26660 --- [ctor-http-nio-3] c.t.controller.SwAlarmController : 告警邮件已发送...
scope: SERVICE_INSTANCE
目标 Scope 的实体名称: 6f34d71f94a74a4c8ffdd45d2d20ff6f@192.168.65.106 of order-service
Scope 实体的 ID: b3JkZXItc2VydmljZQ==.1_NmYzNGQ3MWY5NGE3NGE0YzhmZmRkNDVhMmQyMGZmNmZAMTKyLjE2OC42NS4xMDY=
id1:
告警规则名称: service_instance_resp_time_rule
告警消息内容: Response time of service instance 6f34d71f94a74a4c8ffdd45d2d20ff6f@192.168.65.106 of order-service is mo
告警时间: 1624872979183
标签: null
-----
```

7 Skywalking高可用

在大多数生产环境中，后端应用需要支持高吞吐量并且支持高可用来保证服务的稳定，所以你始终需要在生产环境进行集群管理。

Skywalking集群是将skywalking oap作为一个服务注册到nacos上，只要skywalking oap服务没有全部宕机，保证有一个skywalking oap在运行，就能进行跟踪。

搭建一个skywalking oap集群需要：

- (1) 至少一个Nacos（也可以是nacos集群）
- (2) 至少一个ElasticSearch/mysql（也可以是es/msql集群）
- (3) 至少2个skywalking oap服务；
- (4) 至少1个UI（UI也可以集群多个，用Nginx代理统一入口）

1.修改config/application.yml文件

使用nacos作为注册中心

```
cluster:
  selector: ${SW_CLUSTER:nacos} ← 使用nacos
```

修改nacos配置

```
nacos:
  serviceName: ${SW_SERVICE_NAME:"SkyWalking OAP Cluster"}
  hostPort: ${SW_CLUSTER_NACOS_HOST_PORT:192.168.3.10:8848}
  # Nacos Configuration namespace
  namespace: ${SW_CLUSTER_NACOS_NAMESPACE:"public"}
  # Nacos auth username
  username: ${SW_CLUSTER_NACOS_USERNAME:""}
  password: ${SW_CLUSTER_NACOS_PASSWORD:""}
  # Nacos auth accessKey
  accessKey: ${SW_CLUSTER_NACOS_ACCESSKEY:""}
  secretKey: ${SW_CLUSTER_NACOS_SECRETKEY:""}
```

可以选择性修改监听端口

```

core:
  selector: ${SW_CORE:default}
  default:
    # Mixed: Receive agent data, Level 1 aggregate, Level 2 aggregate
    # Receiver: Receive agent data, Level 1 aggregate
    # Aggregator: Level 2 aggregate
    role: ${SW_CORE_ROLE:Mixed} # Mixed/Receiver/Aggregator
    restHost: ${SW_CORE_REST_HOST:0.0.0.0}
    restPort: ${SW_CORE_REST_PORT:12800}
    restContextPath: ${SW_CORE_REST_CONTEXT_PATH:/}
    restMinThreads: ${SW_CORE_REST_JETTY_MIN_THREADS:1}
    restMaxThreads: ${SW_CORE_REST_JETTY_MAX_THREADS:200}
    restIdleTimeout: ${SW_CORE_REST_JETTY_IDLE_TIMEOUT:30000}
    restAcceptorPriorityDelta: ${SW_CORE_REST_JETTY_DELTA:0}
    restAcceptQueueSize: ${SW_CORE_REST_JETTY_QUEUE_SIZE:0}
    gRPCHost: ${SW_CORE_GRPC_HOST:0.0.0.0}
    gRPCPort: ${SW_CORE_GRPC_PORT:11800}
    maxConcurrentCallsPerConnection: ${SW_CORE_GRPC_MAX_CONCURRENT_CALL:0}
    maxMessageSize: ${SW_CORE_GRPC_MAX_MESSAGE_SIZE:0}
    gRPCThreadPoolQueueSize: ${SW_CORE_GRPC_POOL_QUEUE_SIZE:-1}
    gRPCThreadPoolSize: ${SW_CORE_GRPC_THREAD_POOL_SIZE:-1}

```

修改存储策略，使用elasticsearch7作为storage

```

storage:
  selector: ${SW_STORAGE:elasticsearch7}
  elasticsearch:

```

```

elasticsearch7:
  namespace: ${SW_NAMESPACE:""}
  clusterNodes: ${SW_STORAGE_ES_CLUSTER_NODES:192.168.3.100:9200}
  protocol: ${SW_STORAGE_ES_HTTP_PROTOCOL:"http"}
  trustStorePath: ${SW_STORAGE_ES_SSL_JKS_PATH:""}
  trustStorePass: ${SW_STORAGE_ES_SSL_JKS_PASS:""}
  dayStep: ${SW_STORAGE_DAY_STEP:1} # Represent the number of days in the one minute
  indexShardsNumber: ${SW_STORAGE_ES_INDEX_SHARDS_NUMBER:1} # Shard number of new indexes
  indexReplicasNumber: ${SW_STORAGE_ES_INDEX_REPLICAS_NUMBER:1} # Replicas number of new indexes
  # Super data set has been defined in the codes, such as trace segments.The follow

```

2. 配置ui服务webapp.yml文件的listOfServers，写两个地址

```

server:
  port: 8080

collector:
  path: /graphql
  ribbon:
    ReadTimeout: 10000
    # Point to all backend's restHost:restPort, split by ,
    listOfServers: 192.168.3.10:12800,192.168.3.12:12800

```

3.启动服务测试

启动Skywalking服务，指定springboot应用的jvm参数

```
1 -DSW_AGENT_COLLECTOR_BACKEND_SERVICES=192.168.3.10:11800,192.168.3.12:11800
```