主讲老师： Fox

skywalking版本： v9.1.0

下节课会讲微服务的安全Spring Security Oauth2，需要掌握Spring Security，没用过的同学可以先学习基础课程https://vip.tulingxueyuan.cn/detail/p_63203123e4b00a4f37406646/6?product_id=p_63203123e4b00a4f37406646

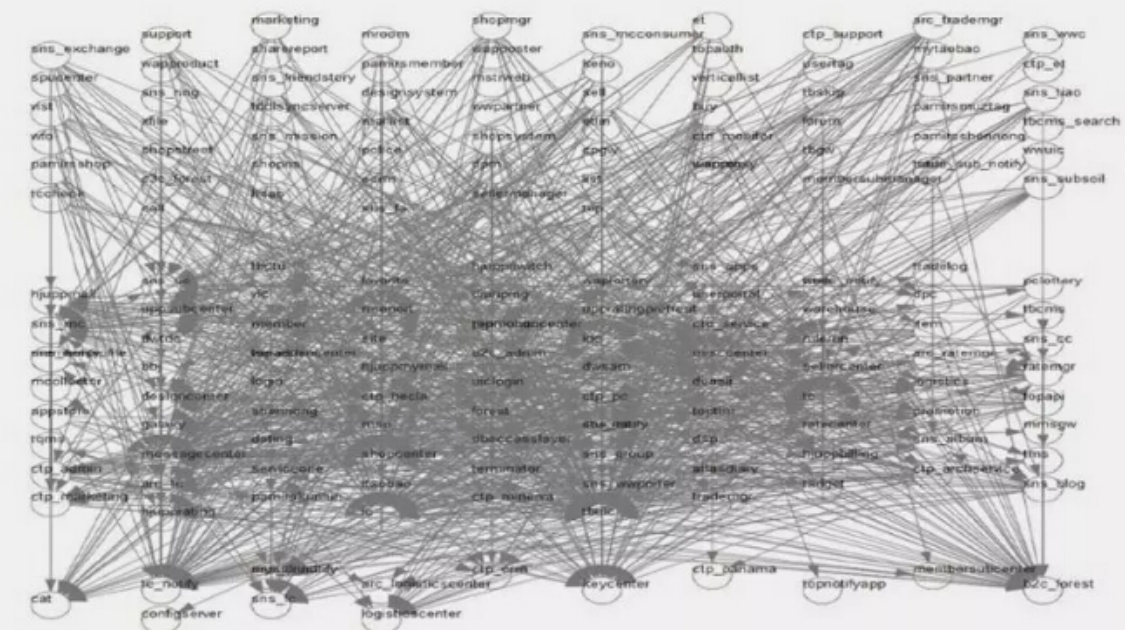
自动化监控系统Prometheus&Grafana实战：
https://vip.tulingxueyuan.cn/detail/v_60f96e69e4b0e6c3a312c726/3?from=p_6006cac4e4b00ff4ed156218&type=8&parent_pro_id=p_6006d8c8e4b00ff4ed1569b2

APM-性能监控项目班：
https://vip.tulingxueyuan.cn/detail/p_602e574ae4b035d3cdb8f8fe/6

```
1 文档：19 微服务链路追踪组件Skywalking实战....
2 链接：http://note.youdao.com/noteshare?id=f46cb9250e146defacf3c707ba847246&sub=09D7
EA6FCDDC4098BC718A42636B11AF
```

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：

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



全链路追踪：对请求源头到底层服务的调用链路中间的所有环节进行监控。

OpenTracing语义规范：
https://github.com/opentracing/specification/blob/master/specification.md

链路追踪组件选型

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。

基本原理

类别	Zipkin	Pinpoint	SkyWalking	CAT
实现方式	拦截请求，发送（HTTP，mq）数据至zipkin服务	java探针，字节码增强	java探针，字节码增强	代码埋点（拦截器，注解，过滤器等）

接入

类别	Zipkin	Pinpoint	SkyWalking	CAT
接入方式	基于linkerd或者sleuth方式，引入配置即可	javaagent字节码	javaagent字节码	代码侵入
agent到collector的协议	http,MQ	thrift	gRPC	http/tcp
OpenTracing	Ã	×	Ã	×

分析

类别	Zipkin	Pinpoint	SkyWalking	CAT
颗粒度	接口级	方法级	方法级	代码级
全局调用统计	×	Ã	Ã	Ã
traceid查询	Ã	×	Ã	×
报警	×	Ã	Ã	Ã
JVM监控	×	×	Ã	Ã

探针性能对比

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. 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。SkyWalking 是观察性分析平台和应用性能管理系统，提供分布式追踪、服务网格遥测分析、度量聚合和可视化一体化解决方案。

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/v9.1.0/readme/

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

版本： v9.1.0

采集数据——》传输数据——》存储数据——》分析数据——》监控报警

1.1 Skywalking主要功能特性

1、多种监控手段，可以通过语言探针和service mesh获得监控的数据；

2、支持多种语言自动探针，包括 Java，.NET Core 和 Node.JS；

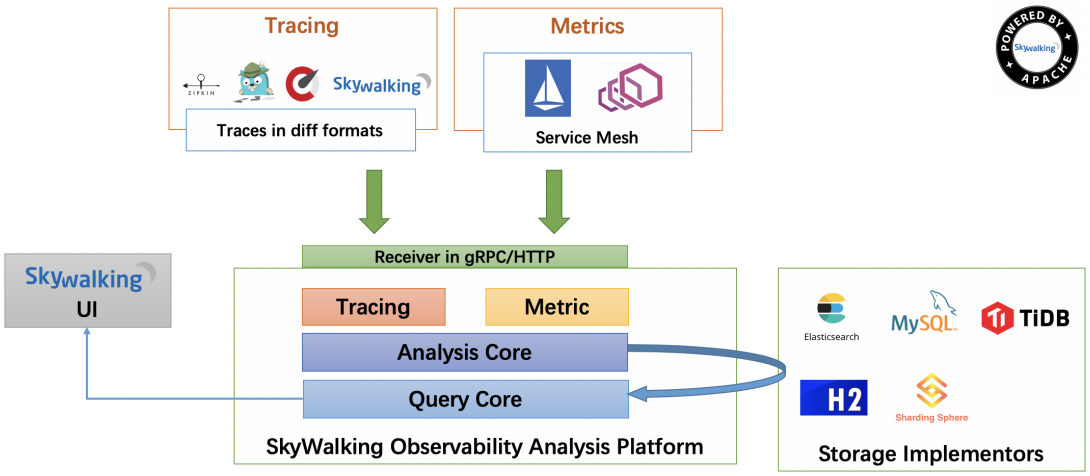
3、轻量高效，无需大数据平台和大量的服务器资源；

4、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；

1.2 Skywalking整体架构



整个架构分成四部分：

- **上部分Agent**：负责从应用中，收集链路信息，发送给 SkyWalking OAP 服务器；

- **下部分 SkyWalking OAP**：负责接收Agent发送的Tracing数据信息，然后进行分析(Analysis Core)，存储到外部存储器(Storage)，最终提供查询(Query)功能；
- **右部分Storage**：Tracing数据存储，目前支持ES、MySQL、Sharding Sphere、TiDB、H2多种存储器，目前采用较多的是ES，主要考虑是SkyWalking开发团队自己的生产环境采用ES为主；
- **左部分SkyWalking UI**：负责提供控制台，查看链路等等；

SkyWalking支持三种探针：

- Agent – 基于ByteBuddy字节码增强技术实现，通过jvm的agent参数加载，并在程序启动时拦截指定的方法来收集数据。
- SDK – 程序中显式调用SkyWalking提供的SDK来收集数据，对应用有侵入。
- Service Mesh – 通过Service mesh的网络代理来收集数据。

后端（Backend）

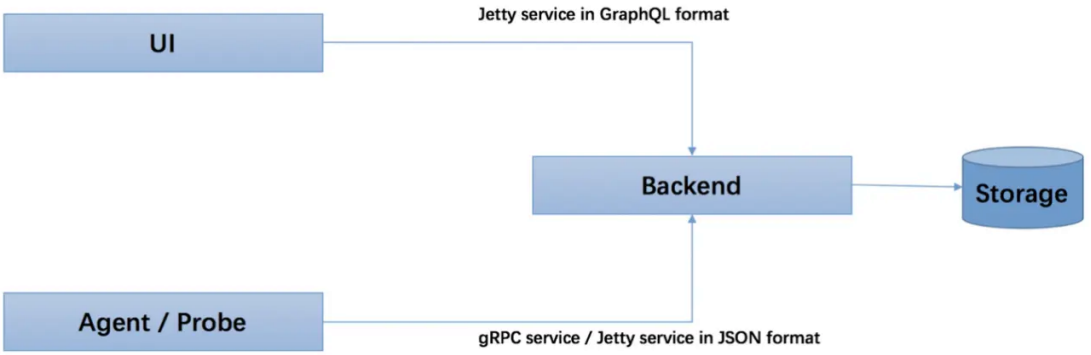
接受探针发送过来的数据，进行度量分析，调用链分析和存储。后端主要分为两部分：

- OAP（Observability Analysis Platform）- 进行度量分析和调用链分析的后端平台，并支持将数据存储到各种数据库中，如：ElasticSearch，MySQL，InfluxDB等。
- OAL（Observability Analysis Language）- 用来进行度量分析的DSL，类似于SQL，用于查询度量分析结果和警报。

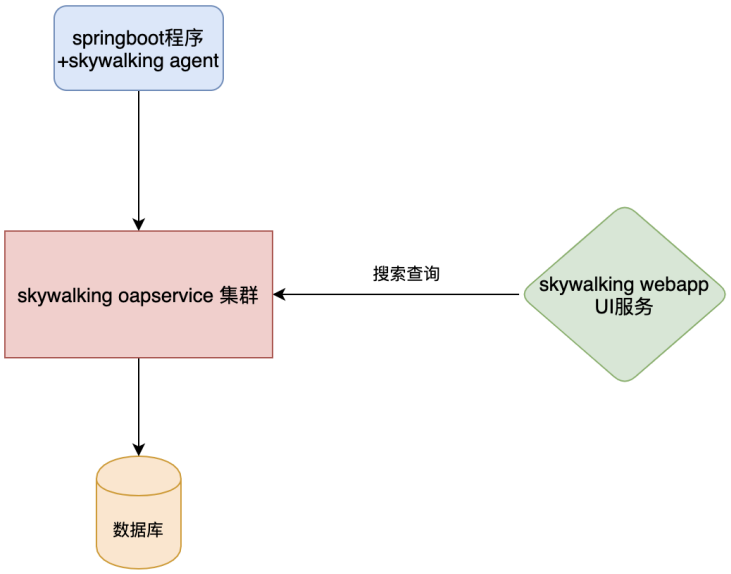
界面(UI)

- RocketBot UI – SkyWalking 7.0.0 的默认web UI
- CLI – 命令行界面

这三个模块的交互流程：



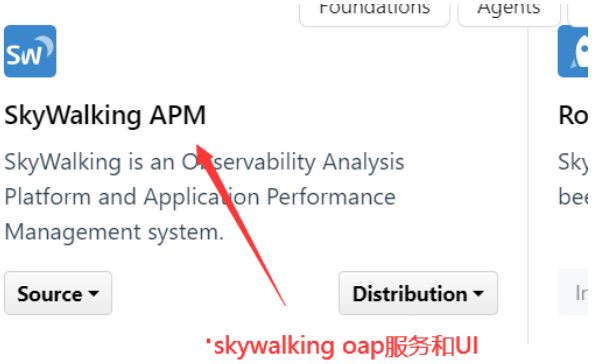
1.3 SkyWalking 环境搭建部署

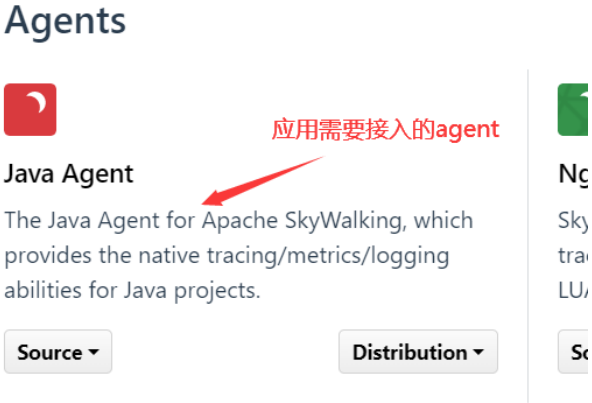


- skywalking agent和业务系统绑定在一起，负责收集各种监控数据
- Skywalking oapservice是负责处理监控数据的，比如接受skywalking agent的监控数据，并存储在数据库中;接受skywalking webapp的前端请求，从数据库查询数据，并返回数据给前端。Skywalking oapservice通常以集群的形式存在。
- skywalking webapp，前端界面，用于展示数据。
- 用于存储监控数据的数据库，比如mysql、elasticsearch等。

下载 SkyWalking

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





- SkyWalking APM: v9.1.0

```
wget https://archive.apache.org/dist/skywalking/9.1.0/apache-skywalking-apm-9.1.0.tar.gz
```

- Java Agent: v8.11.0

```
wget https://archive.apache.org/dist/skywalking/java-agent/8.11.0/apache-skywalking-java-agent-8.11.0.tgz
```

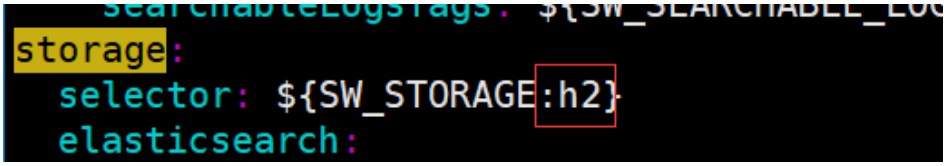
目录结构

- webapp: UI 前端（web 监控页面）的 jar 包和配置文件；
- oap-libs: 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；

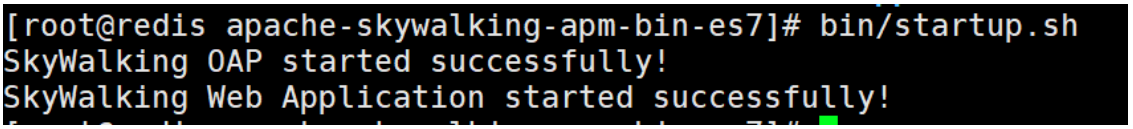
搭建SkyWalking OAP 服务

1）先使用默认的H2数据库存储,不用修改配置

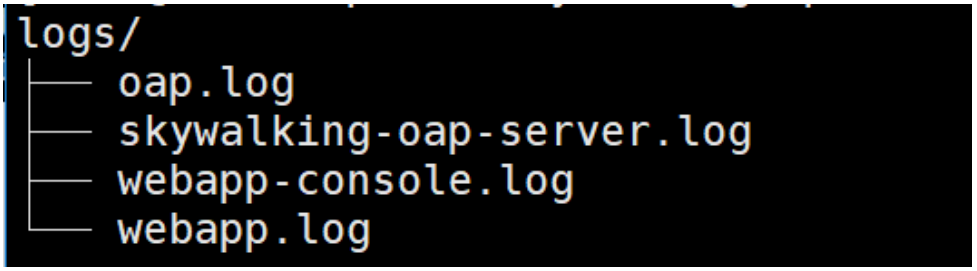
config/application.yml



2）启动脚本bin/startup.sh



日志信息存储在logs目录

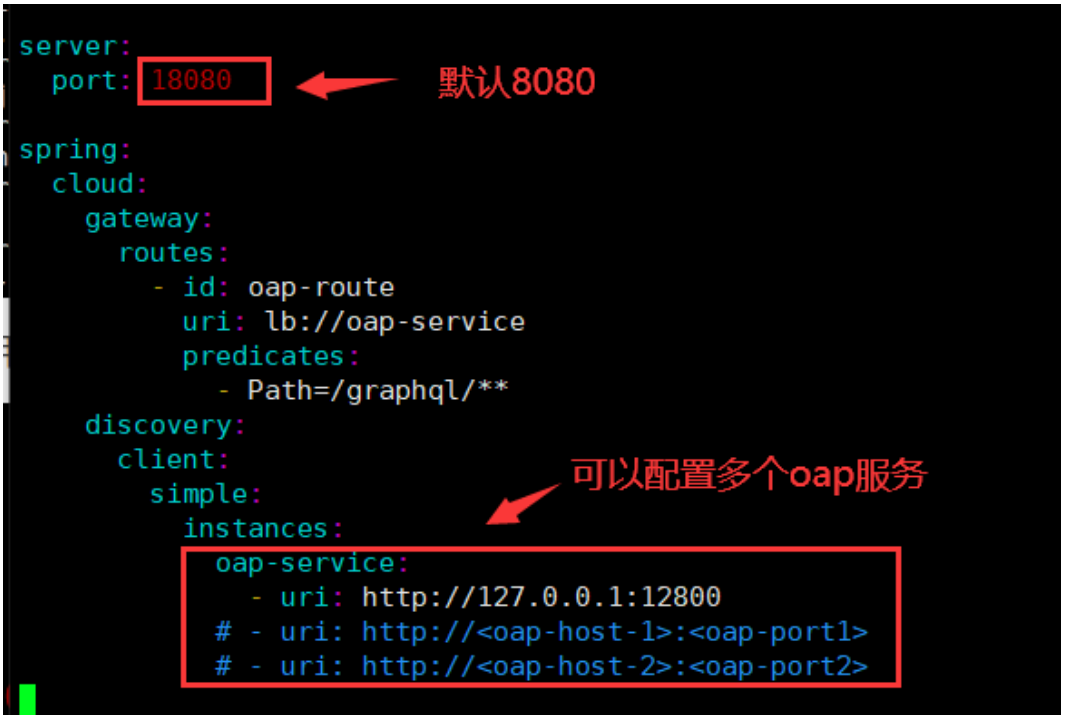


启动成功后会启动两个服务，一个是skywalking-oap-server，一个是skywalking-web-ui

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

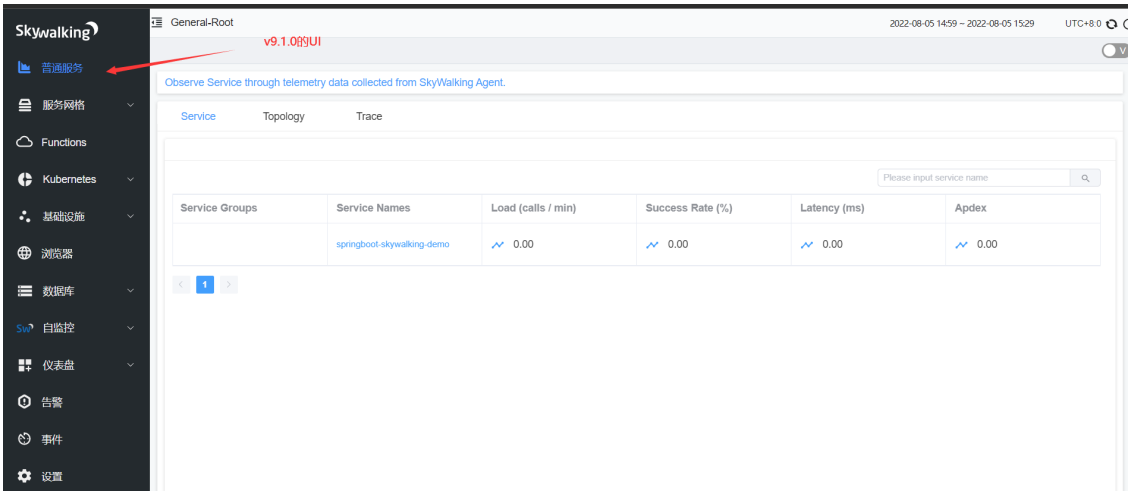
```
[root@redis apache-skywalking-apm-bin-es7]# tail -f logs/skywalking-oap-server.log
2021-02-21 15:02:00,655 - org.apache.skywalking.oap.server.library.server.grpc.GRPC
Server - 140 [main] INFO  [] - Bind handler JVMMetricReportServiceHandler into gRPC
 server 0.0.0.0:11800
2021-02-21 15:02:00,661 - 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：SkyWalking UI服务端口，默认是8080；
- spring.cloud.discovery.client.simple.instances.oap-service：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://192.168.65.206:8080/



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；

2. SkyWalking快速开始

2.1 SkyWalking Agent追踪微服务

2.1.1 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来追踪微服务；

startup.sh脚本：

```
#!/bin/sh
# SkyWalking Agent配置
export SW_AGENT_NAME=springboot-skywalking-demo #Agent名字,一般使用`spring.application.name`
export SW_AGENT_COLLECTOR_BACKEND_SERVICES=127.0.0.1:11800 #配置 Collector 地址。
export SW_AGENT_SPAN_LIMIT=2000 #配置链路的最大Span数量，默认为 300。
export JAVA_AGENT=-javaagent:/root/skywalking-agent/skywalking-agent.jar
java $JAVA_AGENT -jar springboot-skywalking-demo-0.0.1-SNAPSHOT.jar #jar启动
```

等同于

```
java -javaagent:/root/skywalking-agent/skywalking-agent.jar
-DSW_AGENT_COLLECTOR_BACKEND_SERVICES=127.0.0.1:11800
-DSW_AGENT_NAME=springboot-skywalking-demo -jar springboot-skywalking-demo-0.0.1-SNAPSHOT.jar
```

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

```
# The service name in UI
agent.service_name=${SW_AGENT_NAME:Your_ApplicationName}
# Backend service addresses.
collector.backend_service=${SW_AGENT_COLLECTOR_BACKEND_SERVICES:127.0.0.1:11800}
```

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

```
-javaagent:/root/skywalking-agent/skywalking-agent.jar
-Dskywalking.agent.service_name=springboot-skywalking-demo
-Dskywalking.collector.backend_service=127.0.0.1:11800
```

测试： http://192.168.65.206:8000/user/list

2.1.2 在IDEA中使用Skywalking

在运行的程序配置jvm参数

```
-javaagent:D:\apache\apache-skywalking-java-agent-8.11.0\skywalking-agent\skywalking-agent.jar
-DSW_AGENT_NAME=springboot-skywalking-demo
-DSW_AGENT_COLLECTOR_BACKEND_SERVICES=192.168.65.206:11800
```

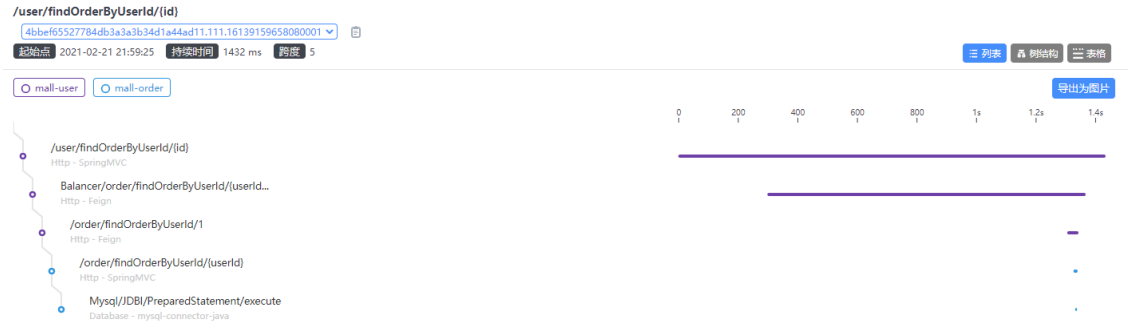
测试：http://localhost:8000/user/list

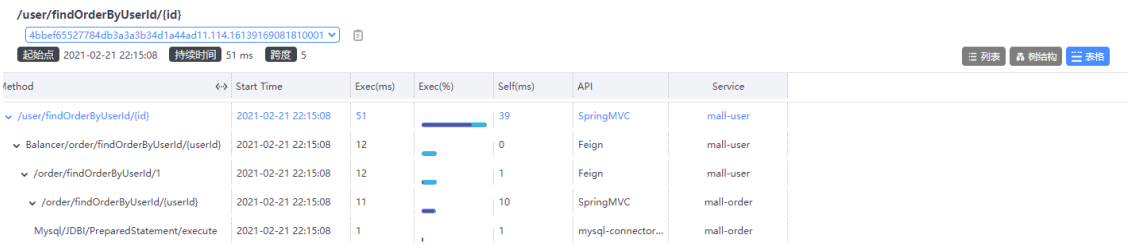
2.1.3 Skywalking跨多个微服务追踪

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

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

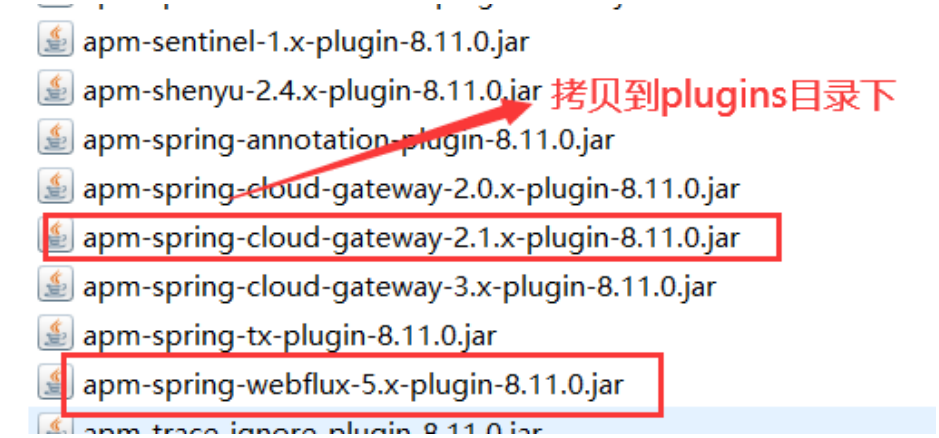
测试：http://localhost:8888/user/findOrderByUserId/1



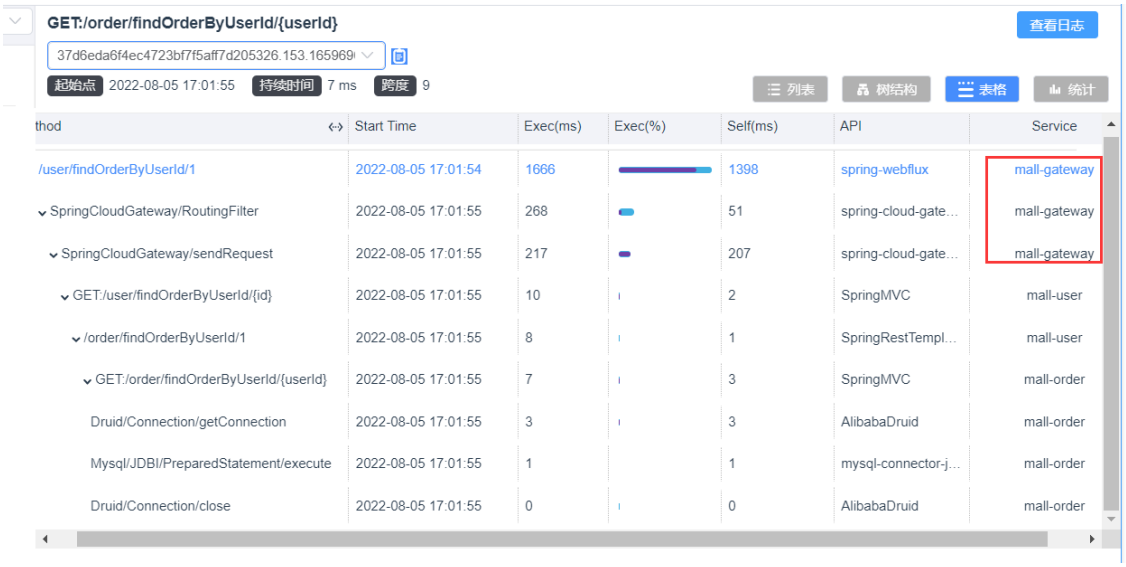


注意：此处存在bug，追踪链路不显示gateway

解决方案：拷贝agent/optional-plugins目录下的gateway插件和webflux插件到agent/plugins目录



查看调用链路



2.2 Skywalking集成日志框架

https://skywalking.apache.org/docs/skywalking-java/latest/en/setup/service-agent/java-agent/application-toolkit-logback-1.x/

引入依赖

```
<!-- apm-toolkit-logback-1.x -->
<dependency>
    <groupId>org.apache.skywalking</groupId>
    <artifactId>apm-toolkit-logback-1.x</artifactId>
    <version>8.11.0</version>
</dependency>
```

微服务添加logback-spring.xml文件，并配置 %tid 占位符

logback-spring.xml
1.33KB

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
```

```
3
4   <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
5   <!-- 日志的格式化 -->
6   <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
7   <layout class="org.apache.skywalking.apm.toolkit.log.logback.v1.x.TraceIdPatternL
ogbackLayout">
8   <Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%tid] [%thread] %-5level %logger{36} -
%msg%n</Pattern>
9   </layout>
10   </encoder>
11   </appender>
12
13   <!-- 设置 Appender -->
14   <root level="INFO">
15   <appender-ref ref="console"/>
16   </root>
17
18  </configuration>
```

测试http://localhost:8888/user/findOrderByUserId/1，查看调用日志

```
022-08-05 17:28:41.809 [TID:2fa2a77d1b244697aafb8a158ba5ad81.244.16596917217780005] [http-nio-8040
022-08-05 17:28:43.670 [TID:2fa2a77d1b244697aafb8a158ba5ad81.244.16596917236650009] [http-nio-8040
022-08-05 17:28:45.198 [TID:2fa2a77d1b244697aafb8a158ba5ad81.244.16596917251940013] [http-nio-8040
022-08-05 17:28:46.012 [TID:2fa2a77d1b244697aafb8a158ba5ad81.244.16596917260070017] [http-nio-8040
022-08-05 17:28:46.933 [TID:2fa2a77d1b244697aafb8a158ba5ad81.244.16596917269290021] [http-nio-8040
```

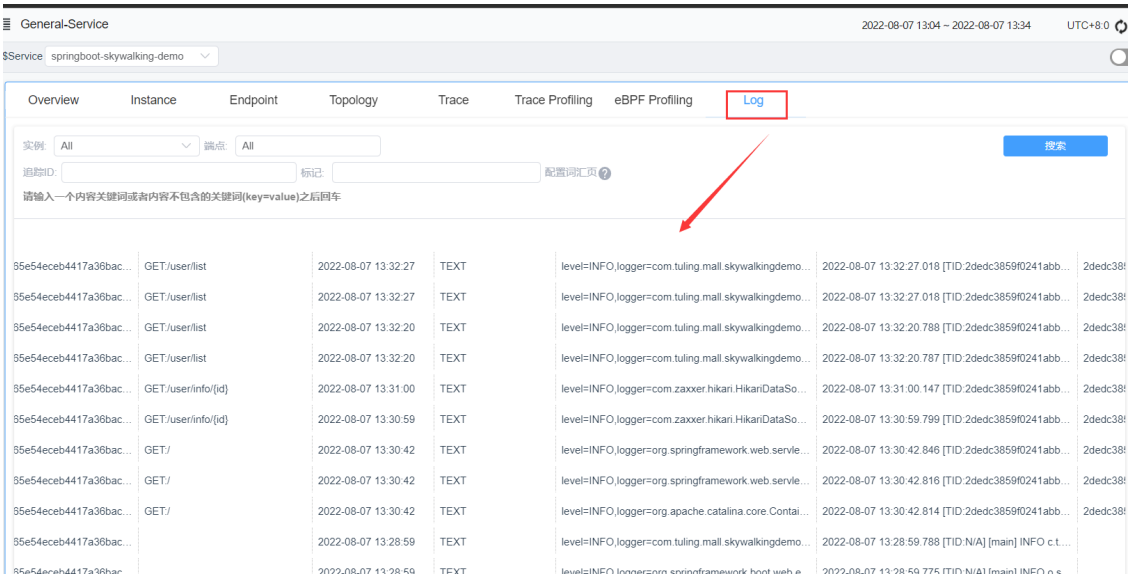
Skywalking通过grpc上报日志 （需要v8.4.0以上）

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   <!-- https://skywalking.apache.org/docs/skywalking-java/latest/en/setup/service-a
gent/java-agent/application-toolkit-logback-1.x/ -->
2  <!-- 通过grpc上报日志到skywalking oap-->
3  <appender name="grpc-log" class="org.apache.skywalking.apm.toolkit.log.logback.v1.
x.log.GRPCLogClientAppender">
4   <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
5   <layout class="org.apache.skywalking.apm.toolkit.log.logback.v1.x.TraceIdPatternL
ogbackLayout">
6   <Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%tid] [%thread] %-5level %logger{36} -
%msg%n</Pattern>
7   </layout>
8   </encoder>
9  </appender>
```

Skywalking UI效果



2.3 Skywalking告警通知

skywalking告警的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三部分:

- 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；
- 网络钩子(Webhook}：当警告触发时，哪些服务终端需要被通知；
- gRPC钩子：远程gRPC方法的主机和端口，告警触发后调用；

为了方便，skywalking发行版中提供了默认的alarm-setting.yml文件，包括一些规则，每个规则有英文注释，可以根据注释得知每个规则的作用：

- 在最近10分钟的3分钟内服务平均响应时间超过1000ms
- 最近10分钟内，服务成功率在2分钟内低于80%
- 服务实例的响应时间在过去10分钟的2分钟内超过1000ms
- 数据库访问{name}的响应时间在过去10分钟的2分钟内超过1000ms

只要我们的服务请求符合alarm-setting.yml文件中的某一条规则就会触发告警。

比如service_resp_time_rule规则：

该规则表示服务{name}的响应时间在最近10分钟的3分钟内超过1000ms

```
# Sample alarm rules.
rules:
  # Rule unique name, must be ended with `_rule`.
  service_resp_time_rule:
    metrics-name: service_resp_time
    op: ">"
    threshold: 1000
    period: 10
    count: 3
    silence-period: 5
    message: Response time of service {name} is more than 1000ms in 3 minutes of last 10 min
utes.
  service_sla_rule:
```

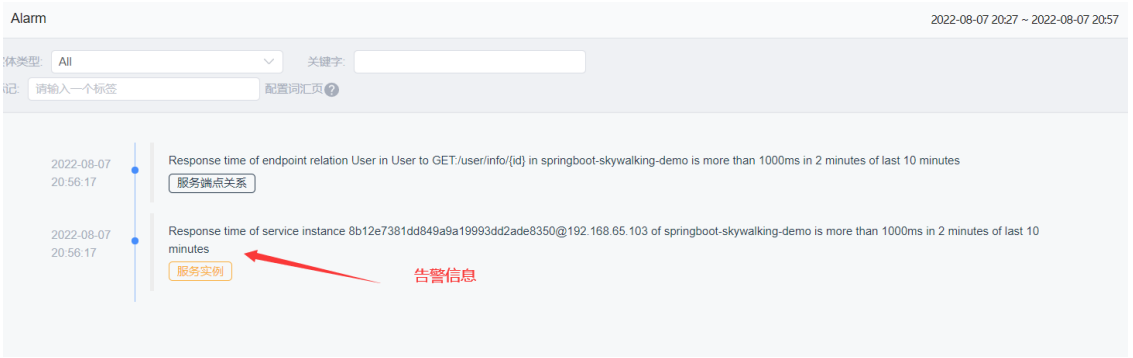
- metrics-name：度量名称，也是OAL脚本中的度量名。默认配置中可以用于告警的度量有：服务，实例，端点，服务关系，实例关系，端点关系。它只支持long,double和int类型。
- op：操作符。

- threshold：阈值。
- period：多久告警规则需要被检查一下。这是一个时间窗口，与后端部署环境时间相匹配。
- count：在一个周期窗口中，如果按op计算超过阈值的次数达到count，则发送告警
- silence-period：在时间N中触发报警后，在N -> N + silence-period这段时间内不告警。
- message：该规则触发时，发送的通知消息。

测试：编写接口，模拟慢查询

```
@RequestMapping("/info/{id}")
public User info(@PathVariable("id") Integer id){

 try {
 Thread.sleep(2000);
 } catch (InterruptedException e) {
 e.printStackTrace();
 }

  return userService.getById(id);
}
```

访问接口，过段时间会在skywalking控制界面出现了告警信息



实现回调接口

```
@RequestMapping("/notify")
public String notify(@RequestBody Object obj){
 //TODO 告警信息，给技术负责人发短信，钉钉消息，邮件，微信通知等
 System.err.println(obj.toString());
 return "notify successfully";
}
```

在config/alarm-settings.yml中配置回调接口，并重启skywalking服务

```
webhooks:
#   - http://127.0.0.1/notify/
#   - http://127.0.0.1/go-wechat/
  - http://192.168.65.103:8000/notify/
```

测试访问：http://localhost:8000/user/info/1，满足告警规则后，控制台输出告警信息

```
[{scopeId=2, scope=SERVICE_INSTANCE, name=98e1839a6fdf48b0aedb0ecabb8ea5f7@192.168.233.1 of springboot-skywalkin
[{scopeId=1, scope=SERVICE, name=springboot-skywalking-demo, id0=c3ByaW5nYm9vdC1za3l3YWxraW5nLWRlbW8=.1, id1=, r
[{scopeId=2, scope=SERVICE_INSTANCE, name=98e1839a6fdf48b0aedb0ecabb8ea5f7@192.168.233.1 of springboot-skywalkin
```

参考：

https://github.com/apache/skywalking/blob/master/docs/en/setup/backend/backend-alarm.md

对接钉钉：

```
dingtalkHooks:
  textTemplate: |-
    {
      "msgtype": "text",
      "text": {
        "content": "Apache SkyWalking Alarm: \n %s."
      }
    }
  webhooks:
    - url: https://oapi.dingtalk.com/robot/send?access_token=dummy_token
      secret: dummysecret
```

Webhook回调通知

SkyWalking告警Webhook回调要求接收方是一个Web容器（比如tomcat服务），告警的消息会通过HTTP请求进行发送, 请求方法为POST, Content-Type为application/json, JSON格式基于List<org.apache.skywalking.oap.server.core.alarm.AlarmMessage>的集合对象数据, 集合中的每个AlarmMessage包含以下信息：

1. scopeId. 所有可用的Scope，参考：org.apache.skywalking.oap.server.core.source.DefaultScopeDefine；
2. name. 目标 Scope 的实体名称；
3. id0. Scope 实体的 ID；
4. id1. 未使用；
5. ruleName. 在 alarm-settings.yml 中配置的规则名；
6. alarmMessage. 报警消息内容；
7. startTime. 告警时间, 位于当前时间与 UTC 1970/1/1 之间；

```
[{
  scopeId = 2,
  scope = SERVICE_INSTANCE,
  name = 98e1839 a6fdf48b0aedb0ecabb8ea5f7 @192 .168 .233 .1 of springboot - skywalking - demo,
  id0 = c3ByaW5nYm9vdC1za3l3YWxraW5nLWRlbW8 = .1 _OThlMTgzOWE2ZmRmNDhiMGFlZGIwZWNhYmI4ZWE1ZjdAMTkyLjE2OC4yMzMuMQ == ,
  id1 = ,
  ruleName = service_instance_resp_time_rule,
```

```
  alarmMessage = Response time of service instance 98e1839 a6fdf48b0aedb0ecabb8ea5f7 @192 .168 .233 .1 of springboot - skywalking - demo is more than 1000 ms in 2 minutes of last 10 minutes,
  startTime = 1613913565462
 }, {
  scopeId = 6,
  scope = ENDPOINT_RELATION,
  name = User in User to / user / info / {
  id
  } in springboot - skywalking - demo,
  id0 = VXNlcg == .0 _VXNlcg == ,
  id1 = c3ByaW5nYm9vdC1za3l3YWxraW5nLWRlbW8 = .1 _L3VzZXIvaW5mby97aWR9,
  ruleName = endpoint_relation_resp_time_rule,
  alarmMessage = Response time of endpoint relation User in User to / user / info / {
  id
  } in springboot - skywalking - demo is more than 1000 ms in 2 minutes of last 10 minutes,
  startTime = 1613913565462
 }]
```

2.4 Skywalking持久化追踪数据

2.4.1 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}
  elasticsearch:
```

默认h2,改为mysql

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}
    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}
```

修改mysql配置

```
storage:
  #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
  selector: ${SW_STORAGE:mysql}
  #使用mysql作为持久化存储的仓库
  mysql:
  properties:
  #数据库连接地址 创建swtest数据库
  jdbcUrl: ${SW_JDBC_URL:"jdbc:mysql://1ocalhost:3306/swtest"}
  #用户名
   dataSource.user: ${SW_DATA_SOURCE_USER:root}
   #密码
```

```
12    dataSource.password: ${SW_DATA_SOURCE_PASSWORD:root}
```

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-libs目录下

```
mvel2-2.4.8.Final.jar
mysql-connector-java-8.0.17.jar
nacos-api-1.3.1.jar
```

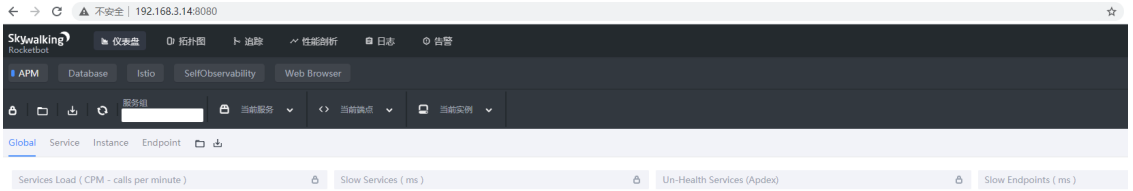
4. 启动Skywalking

```
[root@redis oap-libs]# 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_resp_time	instance_jvm_memory_heap_max	service_instance_relation_server_resp
browser_app_page_first_pack_percentile	database_access_sla	instance_jvm_memory_noheap	service_instance_relation_server_side
browser_app_page_fmp_avg	endpoint_avg	instance_jvm_memory_noheap_max	service_instance_resp_time
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_resp_time	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_resp_time
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_resp_time

说明启动成功了，打开配置对应的地址http://192.168.65.206:8080/，可以看到skywalking的web界面。



测试：重启skywalking，验证追踪数据会不会丢失

2.4.2 基于elasticsearch持久化

1.准备好elasticsearch环境（参考ES专题）

启动elasticsearch服务

```
bin/elasticsearch -d
```

```
[es@es-node2 ~]$ jps
22900 OAPServerStartUp
22710 Logstash
23430 Jps
15208 skywalking-webapp.jar
29549 Elasticsearch
```

2.修改config/application.yml配置文件，指定存储使用ES，修改elasticsearch的连接配置

```
storage:
  selector: ${SW_STORAGE:elasticsearch}
  elasticsearch:
    namespace: ${SW_NAMESPACE:"mall-"}
    clusterNodes: ${SW_STORAGE_ES_CLUSTER_NODES:localhost:9200}
    protocol: ${SW_STORAGE_ES_HTTP_PROTOCOL:"http"}
    connectTimeout: ${SW_STORAGE_ES_CONNECT_TIMEOUT:3000}
    socketTimeout: ${SW_STORAGE_ES_SOCKET_TIMEOUT:30000}
    responseTimeout: ${SW_STORAGE_ES_RESPONSE_TIMEOUT:15000}
    numHttpClientThread: ${SW_STORAGE_ES_NUM_HTTP_CLIENT_THREAD:0}
    user: ${SW_ES_USER:"elastic"}
    password: ${SW_ES_PASSWORD:"123456"}
    trustStorePath: ${SW_STORAGE_ES_SSL_JKS_PATH:""}
    trustStorePass: ${SW_STORAGE_ES_SSL_JKS_PASS:""}
    secretsManagementFile: ${SW_ES_SECRETS_MANAGEMENT_FILE:""} # Secrets management file in the pro
```

指定ES存储数据

指定命名空间，用于生成索引的前缀mall-

ES配置

3. 启动Skywalking服务

```
[root@redis ~]# cd /usr/local/soft/apache-skywalking-apm-bin-es7/
[root@redis apache-skywalking-apm-bin-es7]# ls
agent  bin  config  LICENSE  licenses  logs  NOTICE  oap-libs  README.txt
[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
10105 OAPServerStartUp
10137 Jps
10122 skywalking-webapp.jar
```

启动时会向elasticsearch中创建大量的index索引用于持久化数据

启动应用程序，查看追踪数据是否已经持久化到elasticsearch的索引中，然后重启skywalking，验证追踪数据会不会丢失

2.5 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

```
<!-- SkyWalking 工具类 -->
<dependency>
    <groupId>org.apache.skywalking</groupId>
    <artifactId>apm-toolkit-trace</artifactId>
    <version>8.11.0</version>
</dependency>
```

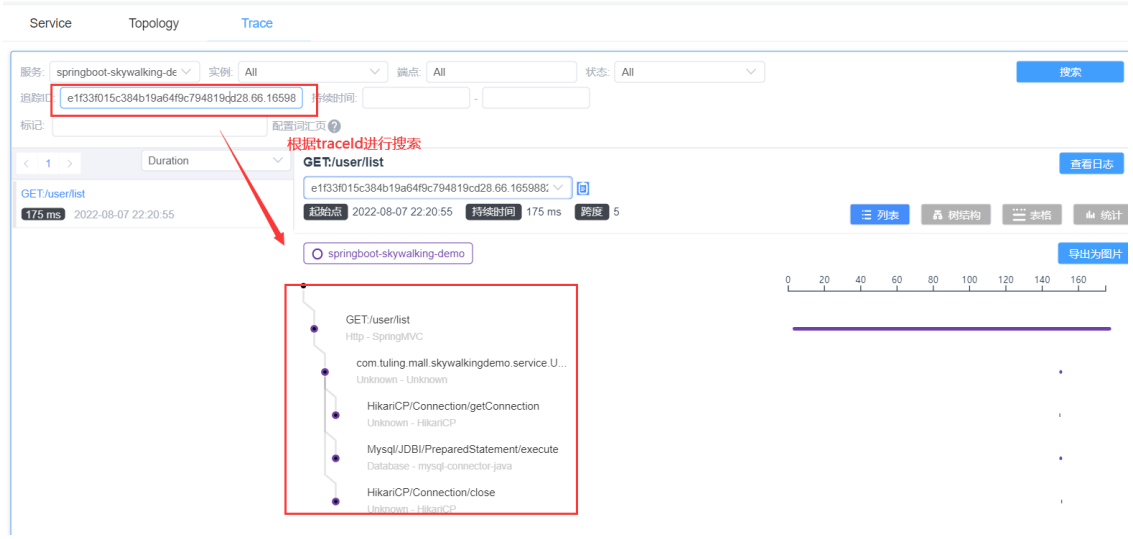
在业务方法中可以TraceContext获取到traceId

```
@RequestMapping("/list")
public List<User> list(){

 //TraceContext可以绑定key-value
 TraceContext.putCorrelation("name", "fox");
 Optional<String> op = TraceContext.getCorrelation("name");
 log.info("name = {} ", op.get());
 //获取追踪的traceId
 String traceId = TraceContext.traceId();
  log.info("traceId = {} ", traceId);

  return userService.list();
}
```

测试 http://localhost:8000/user/list

```
ler.UserController        : name = fox
ler.UserController        : traceId = 283766d19e734048b10294d664a9510f.49.16140588953660001
```

在Skywalking UI中查询tranceId



2.5.1 @Trace将方法加入追踪链路

如果一个业务方法想在ui界面的追踪链路上显示出来，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();
    }
```

UserService中加入@Trace

测试：

2.5.2 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。

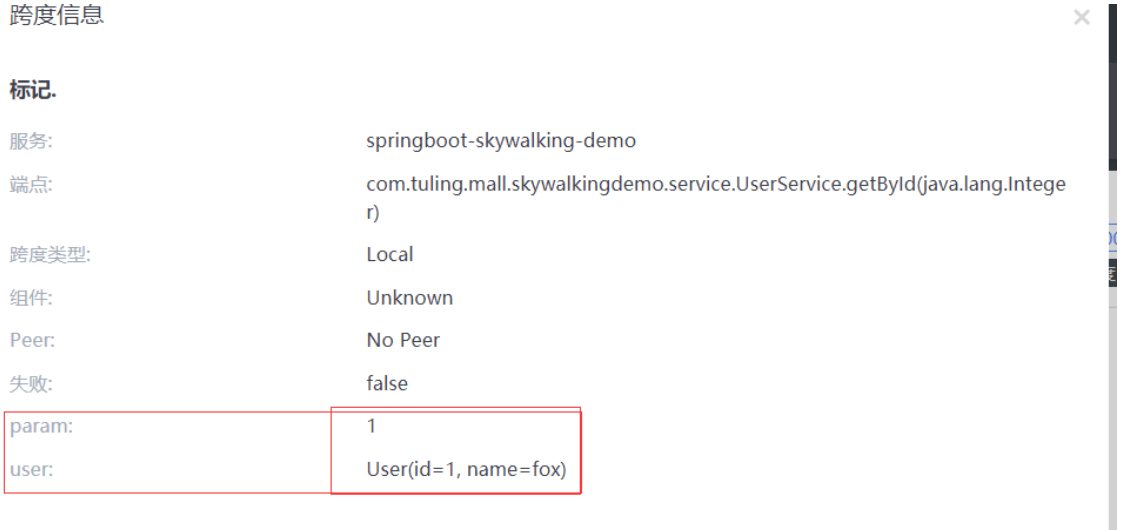
```
@Trace
@Tag(key = "list", value = "returnedObj")
```

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

@Trace
@Tags({@Tag(key = "param", value = "arg[0]"),
 @Tag(key = "user", value = "returnedObj")})
public User getById(Integer id){
 return userMapper.getById(id);
}

```

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)]



2.6 Skywalking集群部署(oap服务高可用)

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，就能进行追踪。

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（也可以是es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

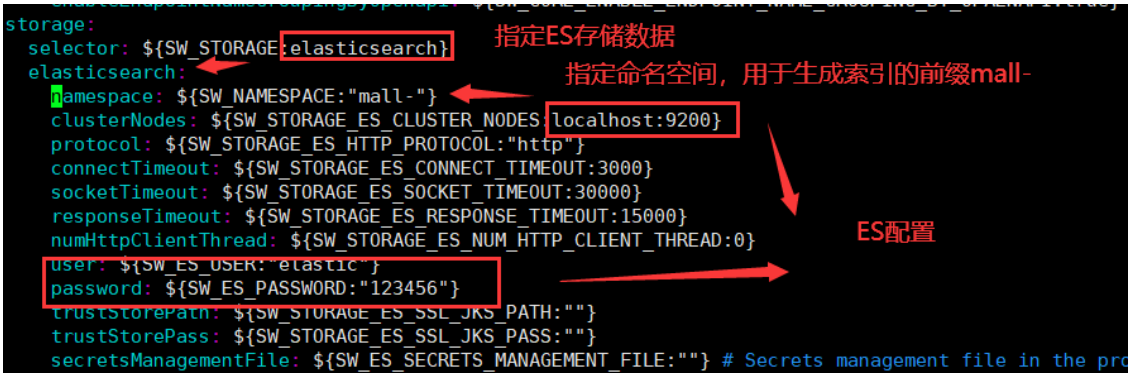
修改nacos配置

```
  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}
```

修改存储策略，使用elasticsearch作为storage



2. 配置ui服务webapp.yml文件的oap-service，写多个oap服务地址

3.启动微服务测试

指定微服务的jvm参数

```
-Dskywalking.collector.backend_service=ip1:11800,ip2:11800
```