## 一站式微服务解决方案Spring Cloud Alibaba

主讲：Cat 老师

北京动力节点教育科技有限公司  
2020 • 北京

动力节点•版权所有•禁止传播

## 快速回顾

1、Spring家族开源项目梳理

2、Spring Cloud下的开源项目梳理

3、Spring Cloud Alibaba下的开源及商业项目梳理

4、微服务的基础模型：服务消费者-注册中心-服务提供者

5、What is Nacos？

6、Nacos的运行环境部署（Java写的，springboot项目）

7、Nacos的后台web管控台

8、Nacos作为注册中心注册服务

9、Nacos作为注册中心发现/订阅服务

10、服务消费者负载均衡调用服务提供者（ribbon）restTemple、feign（openfeign）

11、Nacos宕机时服务消费者缓存注册中心信息

12、Nacos作为配置中心存储项目各种配置

13、Nacos作为配置中心支持自动配置刷新（不需要重启应用）

14、Nacos配置中心DataId+Group+Properties/yaml+配置内容（比较灵活）

15、Nacos配置中心多环境配置（profile）

即${spring.application.name}-${profile}.${file-extension:properties}

16、Nacos服务配置数据模型（命名空间、Group、Data Id）

17、Nacos数据持久化（mysql）

18、Ncaos集群部署（nginx）

19、主要调用方式：restTemplate、feign、ribbon（spring cloud）

20、流量控制Sentinel（流控、降级、热点、系统、授权 规则）

21、Sentinel Darshboard通信原理（与微服务通信）

22、Sentinel 对应用保护的三种方式

23、Sentinel整合RestTemplate流控熔断

24、Sentinel整合Feign流控熔断

25、Sentinel规则持久化（默认、pull模式、push模式）

26、Spring Cloud Gateway网关（核心概念、如何工作、路由、谓词11个、过滤器31个）

27、Spring Cloud Gateway自定义谓词

28、Spring Cloud Gateway谓词不匹配404处理

29、Spring Cloud Gateway自定义路由过滤器

30、Spring Cloud Gateway全局过滤器（默认根据条件注解生效，不需要单独配置）

31、Spring Cloud Gateway集成ribbon负载均衡

32、Spring Cloud Gateway集成Sentinel

33、Spring Cloud Gateway集成Sentinel规则持久化（文件、nacos）

34、Spring Cloud Gateway内部流程源码分析

35、Spring cloud gateway跨域CORS

36、Skywalking分布式链路跟踪

37、Skywalking主要功能特性和整体架构

38、SkyWalking 环境搭建部署

39、SkyWalking Agent跟踪微服务

40、IDEA中使用Skywalking Agent跟踪运行的程序

41、Skywalking告警和回调通知

## Skywalking持久化到elasticsearch

Skywalking跟踪数据默认是存放在内嵌式数据库H2中的，重启skywalking，跟踪数据就丢失了，我们可以把跟踪数据持久化到mysql或者elasticsearch中，上一次课我们介绍了如何将跟踪数据持久化到mysql，此次我们介绍一些如何将跟踪数据持久化到elasticsearch中；

需要解压另一个专门针对elasticsearch版本的Skywalking：

1、tar -zxvf apache-skywalking-apm-es7-8.1.0.tar.gz

2、cd apache-skywalking-apm-es7-8.1.0

3、修改application.yml配置文件：

storage:

selector: ${SW\_STORAGE:elasticsearch7}

4、启动elasticsearch7

./elasticsearch -d 后台运行 （es不能用root启动）

注：Elasticsearch的jvm内存不能配置太小，至少512m，小了会出现错误：

status line [HTTP/1.1 429 Too Many Requests]

5、启动elasticsearch-head插件：npm run start 便于查看elasticsearch数据；  
6、启动skywalking

./startup.sh

启动时会向elasticsearch中创建大量的index索引用于持久化数据，每天会产生一个新的索引文件；

7、启动应用程序，查看跟踪数据是否已经持久化到elasticsearch的索引中；

8、然后重启skywalking，验证跟踪数据会不会丢失；

## 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；

-javaagent:D:/dev/apache-skywalking-apm-bin/agent/skywalking-agent.jar

SW\_AGENT\_COLLECTOR\_BACKEND\_SERVICES=192.168.172.128:11800;SW\_AGENT\_NAME=11-springboot-idea

## 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；

1. 添加依赖；

*<!-- SkyWalking 工具类 -->*<dependency>  
 <groupId>org.apache.skywalking</groupId>  
 <artifactId>apm-toolkit-trace</artifactId>  
 <version>8.1.0</version>  
</dependency>

2、在业务代码中使用如下方法：

TraceContext.*putCorrelation*("myKey", "myValue");  
Optional<String> op = TraceContext.*getCorrelation*("myKey");  
*log*.info("myValue = {} ", op.get());  
  
String traceId = TraceContext.*traceId*();  
*log*.info("traceId = {} ", traceId);

这样就可以拿到跟踪的traceId，便于根据traceId去ui界面搜索整个跟踪链路；

另外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；

## Skywalking集成日志框架

1、添加如下依赖；

<!-- apm-toolkit-logback-1.x -->

<dependency>

<groupId>org.apache.skywalking</groupId>

<artifactId>apm-toolkit-logback-1.x</artifactId>

<version>8.1.0</version>

</dependency>

2、添加logback-spring.xml文件（参见git代码）；

3、在代码中记录日志：

*log*.info("/goods --> getAllGoods --> ......" );

4、访问controller测试，查看日志输出，就可以在日志中看到跟踪的traceId；

## Skywalking ui页面功能

1

仪表盘：查看被监控服务的运行状态；

拓扑图：以拓扑图的方式展现服务之间的关系，并以此为入口查看相关信息；

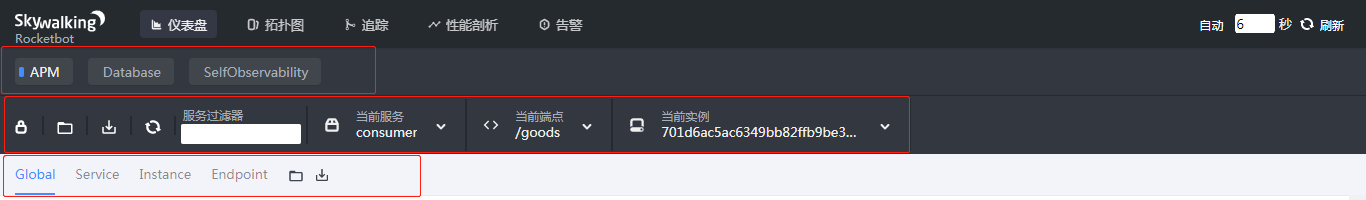
追踪：以接口列表的方式展现，追踪接口内部调用过程；

性能剖析：对端点进行采样分析，并可查看堆栈信息；

告警：触发告警的告警列表，包括服务失败率，请求超时等；

自动刷新：刷新当前页面数据内容；

### 控制栏

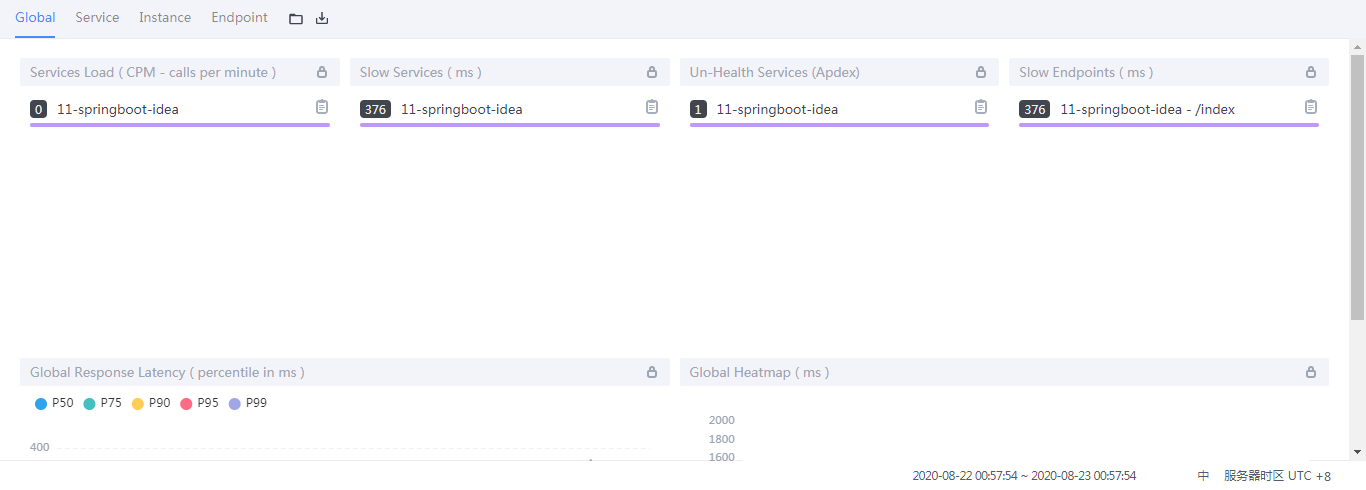


第一栏：不同内容主题的监控面板，应用性能管理/数据库/容器等；

第二栏：操作，包括 编辑/导出当前数据/倒入展示数据/不同服务端点筛选展示；

第三栏：不同纬度展示，全局/服务/实例/端点；

### 展示栏



**Global全局维度**

第一栏：Global、Service、Instance、Endpoint不同展示面板；

Services load：服务每分钟请求数；

Slow Services：慢响应服务，单位ms；

Un-Health services(Apdex): Apdex性能指标，1为满分；

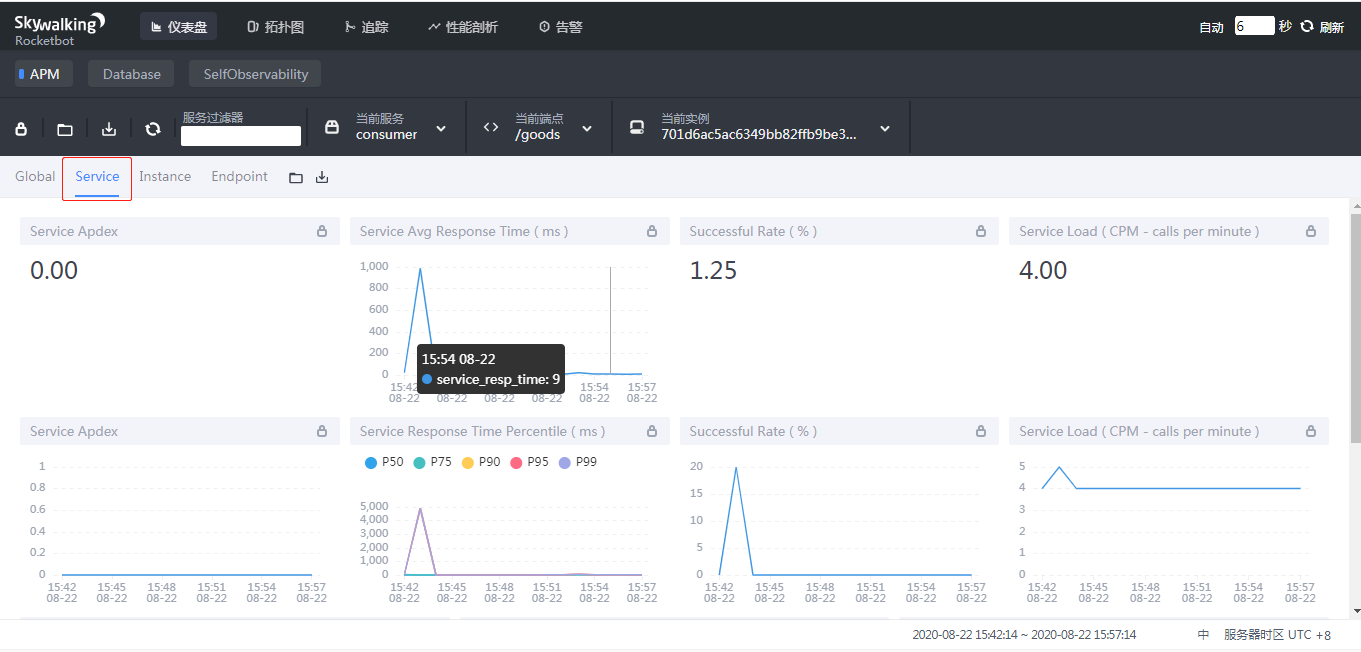
Slow Endpoint：慢响应端点，单位ms；

Global Response Latency：百分比响应延时，不同百分比的延时时间，单位ms；

Global Heatmap：服务响应时间热力分布图，根据时间段内不同响应时间的数量显示颜色深度；

底部栏：展示数据的时间区间，点击可以调整；

### Service服务维度



Service Apdex（数字）:当前服务的评分；

Service Apdex（折线图）：不同时间的Apdex评分；

Service Avg Response Times：平均响应延时，单位ms；

Global Response Time Percentile：百分比响应延时；

Successful Rate（数字）：请求成功率；

Successful Rate（折线图）：不同时间的请求成功率；

Servce Load（数字）：每分钟请求数；

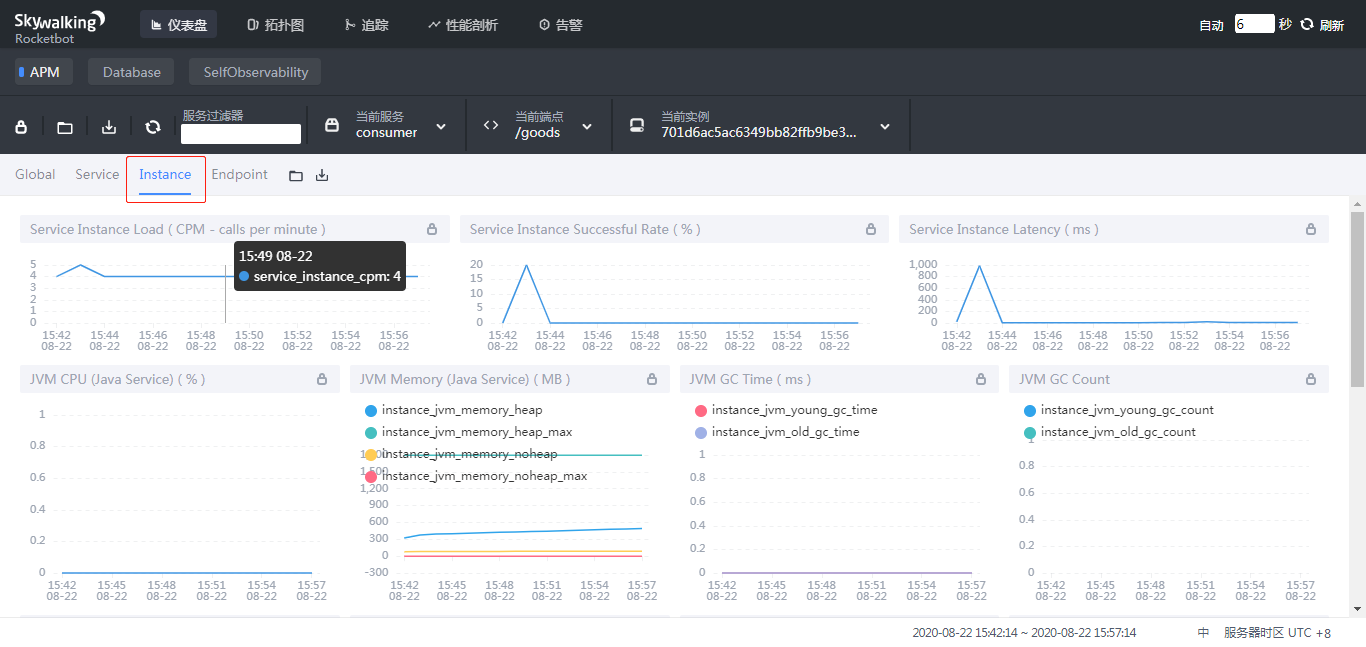
Servce Load（折线图）：不同时间的每分钟请求数；

Servce Instances Load：每个服务实例的每分钟请求数；

Show Service Instance：每个服务实例的最大延时；

Service Instance Successful Rate：每个服务实例的请求成功率；

### Instance实例维度



Service Instance Load：当前实例的每分钟请求数；

Service Instance Successful Rate：当前实例的请求成功率；

Service Instance Latency：当前实例的响应延时；

JVM CPU：jvm占用CPU的百分比；

JVM Memory：JVM内存占用大小，单位m；

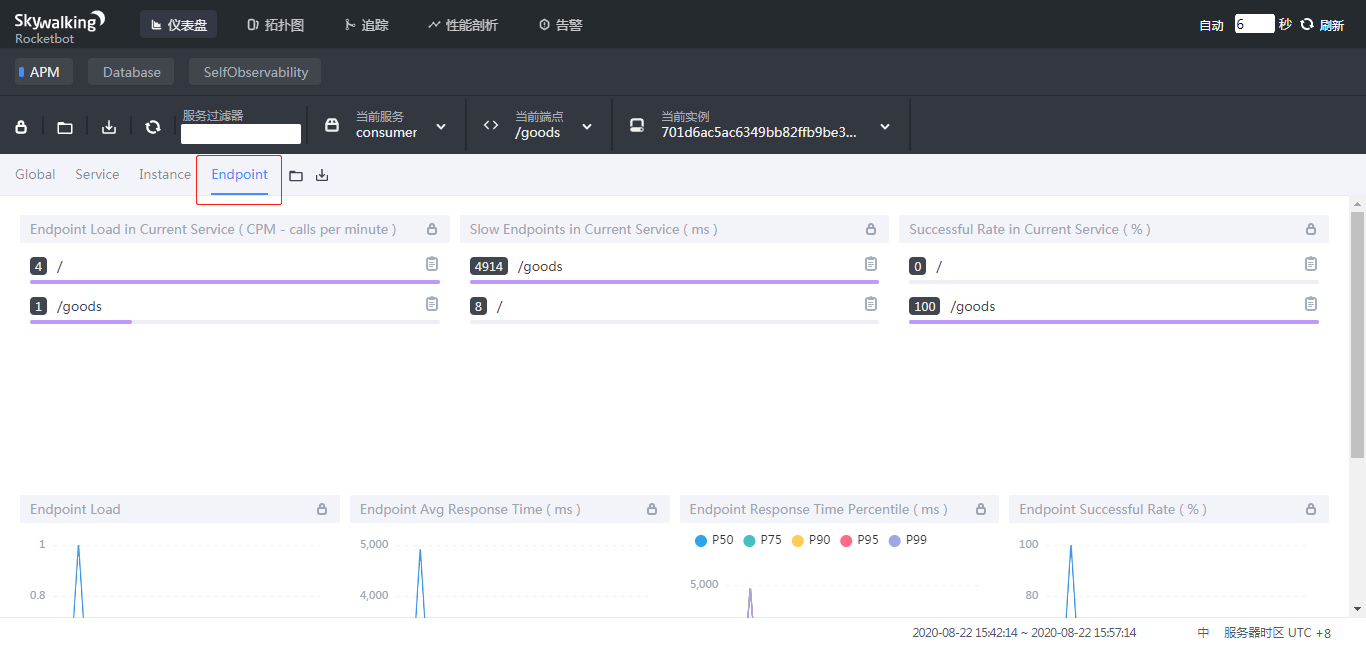
JVM GC Time：JVM垃圾回收时间，包含YGC和OGC；

JVM GC Count：JVM垃圾回收次数，包含YGC和OGC；

JVM Thread Count：JVM线程数；

还有几个是.NET的，类似于JVM虚拟机，暂时不做说明；

### Endpoint端点（API）维度



Endpoint Load in Current Service：每个端点的每分钟请求数；

Slow Endpoints in Current Service：每个端点的最慢请求时间，单位ms；

Successful Rate in Current Service：每个端点的请求成功率；

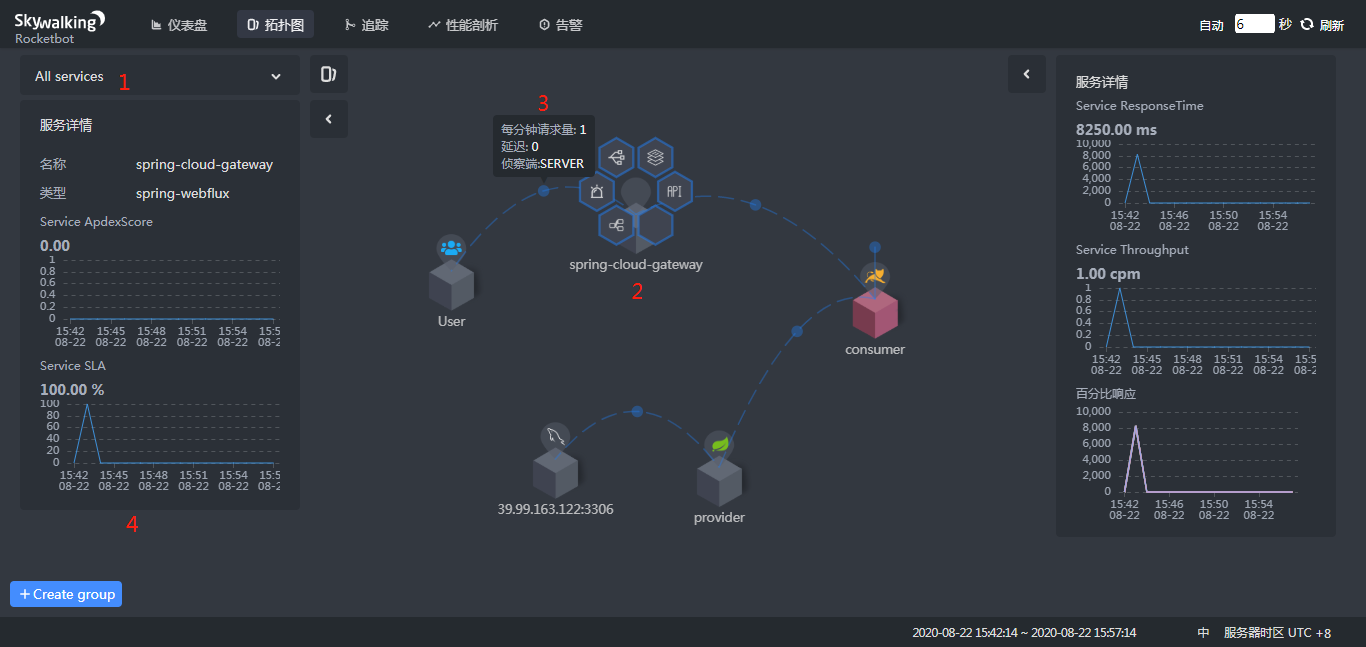
Endpoint Load：当前端点每个时间段的请求数据；

Endpoint Avg Response Time：当前端点每个时间段的请求行响应时间；

Endpoint Response Time Percentile：当前端点每个时间段的响应时间占比；

Endpoint Successful Rate：当前端点每个时间段的请求成功率；

### 拓扑图



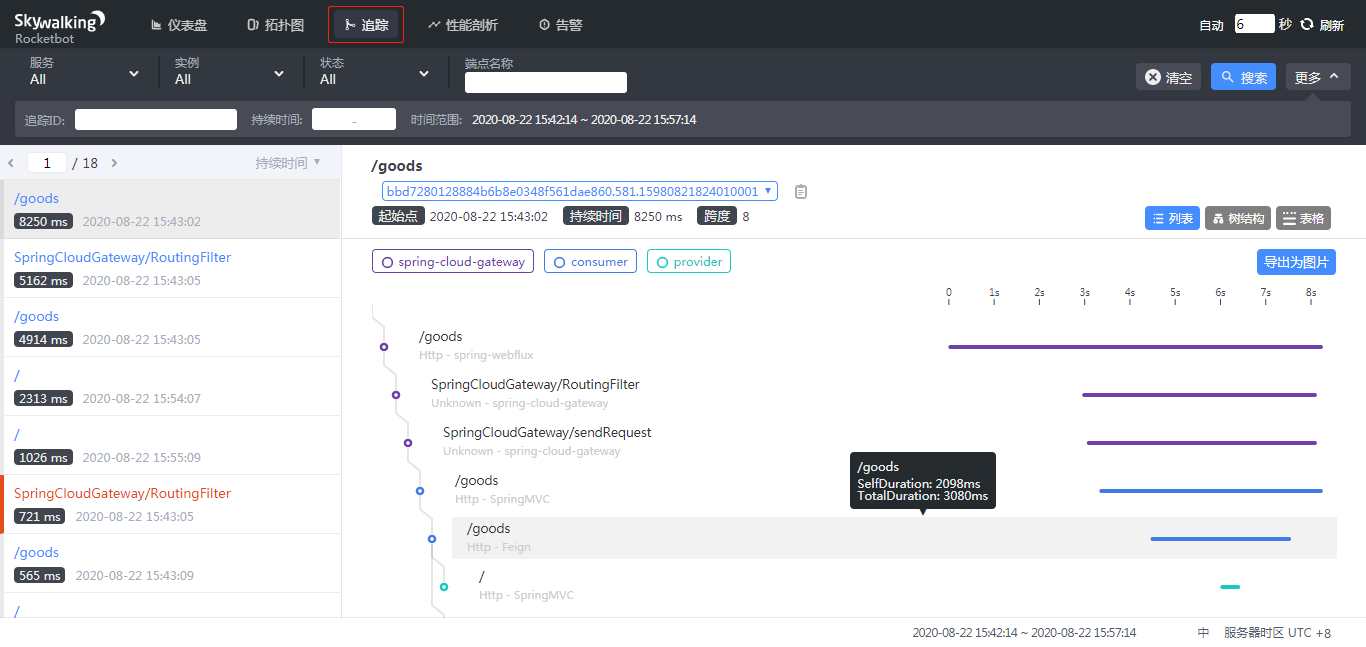
1：选择不同的服务关联拓扑；

2：查看单个服务相关内容；

3：服务间连接情况；

4：分组展示服务拓扑；

### 追踪



左侧：api接口列表，红色-异常请求，蓝色-正常请求；

右侧：api追踪列表，api请求连接各端点的先后顺序和时间；

### 性能剖析



服务：需要分析的服务；

端点：链路监控中端点的名称，可以在链路追踪中查看端点名称；

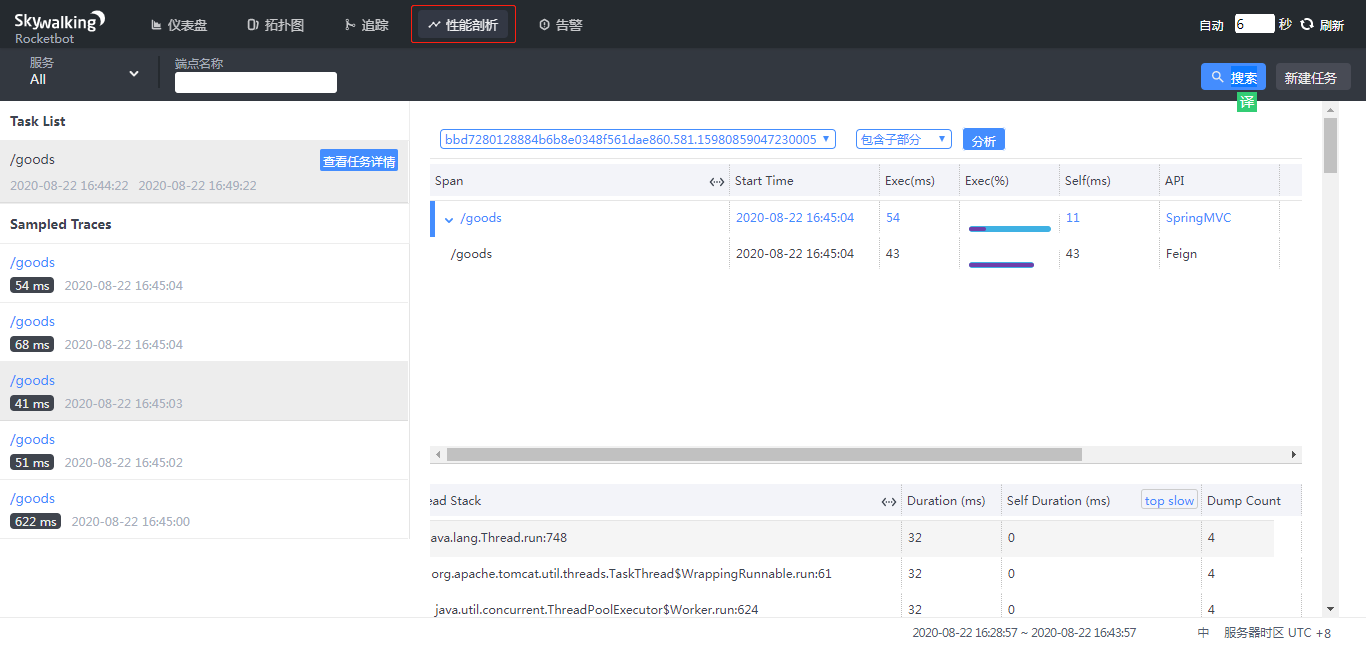
监控时间：采集数据的开始时间；

监控持续时间：监控采集多长时间；

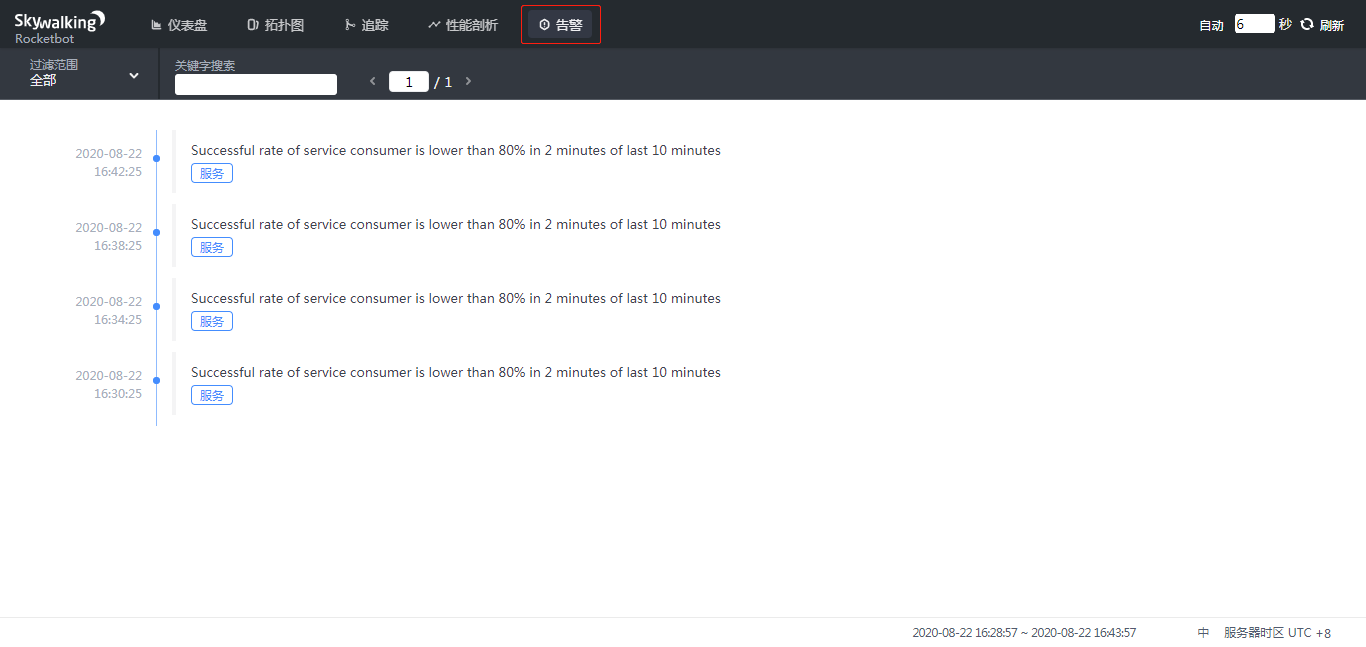
起始监控时间：多少秒后进行采集；

监控间隔：多少秒采集一次；

最大采集数：最大采集多少样本；



### 告警



不同维度告警列表，可分为服务、端点和实例；

## Skywalking集群

生产中不搭集群也是可以的，因为这个只是调用链路跟踪，skywalking oap跟踪服务如果宕机了，完全不会影响正常业务；

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（也可以把nacos集群）

（2）至少一个ElasticSearch（也可以把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）

具体步骤：

1. 解压两份apache-skywalking-apm-es7-8.1.0.tar.gz

2、修改配置application.yml文件

/config/application.yml

cluster:

selector: ${SW\_CLUSTER:nacos}

nacos:

serviceName: ${SW\_SERVICE\_NAME:"SkyWalking\_OAP\_Cluster"}

hostPort: ${SW\_CLUSTER\_NACOS\_HOST\_PORT:localhost:8848}

第一台

restHost: ${SW\_CORE\_REST\_HOST:0.0.0.0}

restPort: ${SW\_CORE\_REST\_PORT:12801}

gRPCPort: ${SW\_CORE\_GRPC\_PORT:11801}

gRPCHost: ${SW\_CORE\_GRPC\_HOST:0.0.0.0}

第二台

restHost: ${SW\_CORE\_REST\_HOST:0.0.0.0}

restPort: ${SW\_CORE\_REST\_PORT:12802}

gRPCPort: ${SW\_CORE\_GRPC\_PORT:11802}

gRPCHost: ${SW\_CORE\_GRPC\_HOST:0.0.0.0}

使用elasticsearch作为storage

storage:

selector: ${SW\_STORAGE:elasticsearch7}

storage:

elasticsearch7:

nameSpace: ${SW\_NAMESPACE:""}

clusterNodes: ${SW\_STORAGE\_ES\_CLUSTER\_NODES:localhost:9200}

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

listOfServers: 127.0.0.1:12801,127.0.0.1:12802

4、启动测试

启动一个skywalking-webapp.jar使用脚本webappService.sh；

启动两个OAPServerStartUp使用脚本oapService.sh；

5、启动应用程序进行测试，如果是jar包启动：

java -javaagent:xxxxx/agent/skywalking-agent.jar

-Dskywalking.agent.service\_name=gateway

-Dskywalking.collector.backend\_service=192.168.172.128:11801,192.168.172.128:11802

-jar 11-springboot-idea-1.0.0.jar

如果是idea启动应用程序：（注意配置两个连接地址）

SW\_AGENT\_COLLECTOR\_BACKEND\_SERVICES=192.168.172.128:11801,192.168.172.128:11802;SW\_AGENT\_NAME=11-springboot-idea