## 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

## Spring cloud gateway跨域CORS

我们知道，传统的Ajax请求只能获取在同一个域名下的资源，但是HTML5规范中打破了这种限制，允许Ajax发起跨域的请求；（只是需要设置一下）

其实浏览器本身是可以发起跨域请求的，比如你可以链接一个另一个域名下的图片或者js，比如<img src=http://www.baidu.com/image/logo.png>，但是javascript脚本是不能获取这些另一个域名下的资源内容的；

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing），它允许浏览器向跨域的另一台服务器发出XMLHttpRequest请求，从而克服了AJAX只能访问同域名下的资源的限制；

这种CORS使用了一个额外的HTTP响应头来赋予当前user-agent（浏览器）获得跨域资源的权限，这里的跨域也就是Cross-Origin的概念，这里的权限就是访问另一个域名下的资源权限；

CORS是现在HTML5标准中的一部分，在大部分现代浏览器中都有所支持，可能在某些老版本的浏览器不支持CORS，如果要兼容一些老的浏览器版本，则需要采用JSONP进行跨域请求；

## 同源与非同源的定义（跨域和不跨域）

如果 **访问协议、端口（**如果指定了端口的话**）、host**都相同，则称之为同源（不跨域），否则为非同源（跨域）；

比如源链接： <http://store.company.com/dir/page.html>

| **URL** | **是否同源** | **原因** |
| --- | --- | --- |
| <http://store.company.com/dir2/other.html> | 是 |  |
| <http://store.company.com/dir/inner/another.html> | 是 |  |
| <https://store.company.com/secure.html> | 否 | 协议不同 |
| <http://store.company.com:81/dir/etc.html> | 否 | 端口不同 |
| <http://news.company.com/dir/other.html> | 否 | host不同 |

Spring Cloud Gateway解决跨域问题，只需要配置如下代码即可：

*/\*\*  
 \* 配置网关跨域cors请求支持  
 \*/*@Configuration  
public class CorsConfig {  
 @Bean  
 public CorsWebFilter corsFilter() {  
 CorsConfiguration config = new CorsConfiguration();  
 config.addAllowedMethod("\*");  
 config.addAllowedOrigin("\*");  
 config.addAllowedHeader("\*");  
 UrlBasedCorsConfigurationSource source = new

UrlBasedCorsConfigurationSource(new PathPatternParser());  
 source.registerCorsConfiguration("/\*\*", config);  
 return new CorsWebFilter(source);  
 }  
}

测试一下！

## Skywalking分布式链路跟踪

### What is skywalking？

分布式链路跟踪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）架构而设计；

也就是说Skywalking是用于微服务的“跟踪" ；

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折？

如何跟踪整个业务流程的调用处理顺序？

Skywalking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；

Skywalking是国人采用Java开发的，现在已经是apache下的一个等级项目；

### 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；

官网：<http://skywalking.apache.org/>

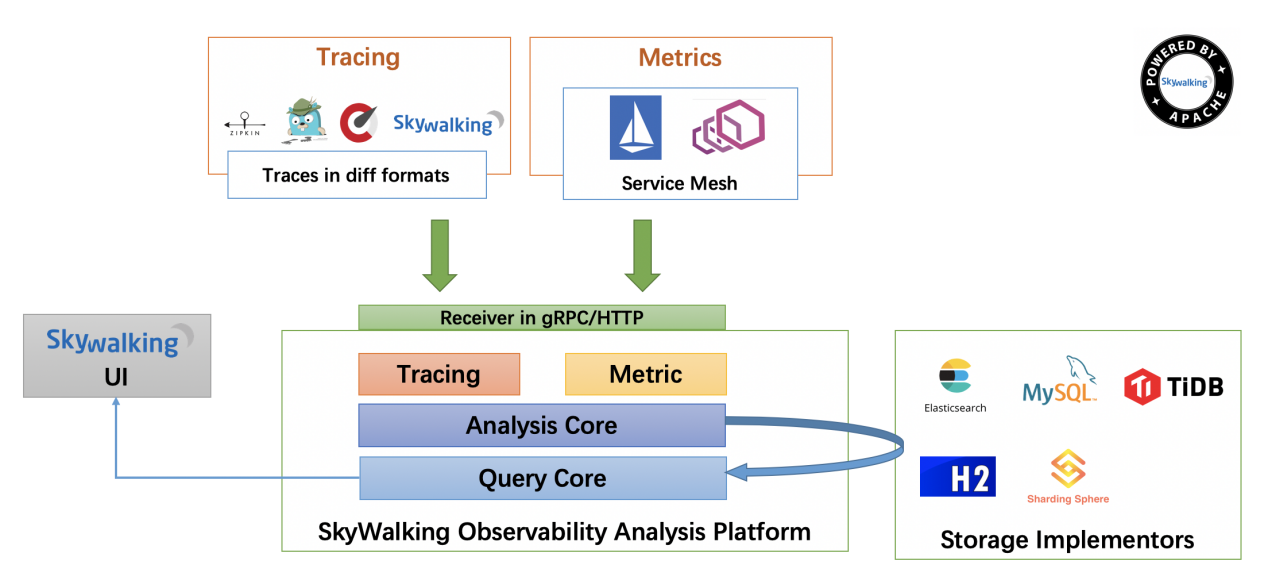
下载：<http://skywalking.apache.org/downloads/>

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

使用公司：（国内非常多）

<https://github.com/apache/skywalking/blob/master/docs/powered-by.md>

### Skywalking整体架构



整个架构分成四部分：

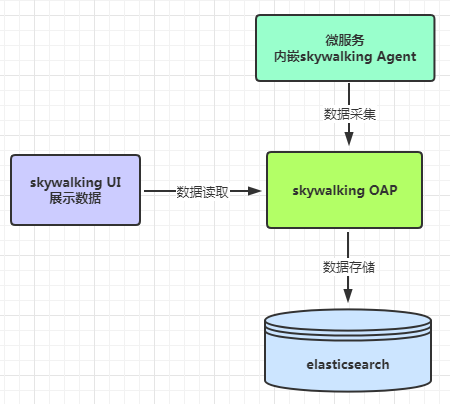
1、上部分Agent ：负责从应用中，收集链路信息，发送给 SkyWalking OAP 服务器；

2、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；

3、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；

4、左部分SkyWalking UI：负责提供控台，查看链路等等；

### SkyWalking 环境搭建部署



1，下载 SkyWalking 软件包；

2，搭建一个 SkyWalking OAP 和SkyWalking UI服务；

3，启动一个Spring Boot应用，并配置SkyWalking Agent；

数据存储暂时先使用它默认的H2数据库存储，后续我们再使用其他存储；

**1、下载 SkyWalking 软件包**

对于 SkyWalking 的软件包，有两种方式获取：

手动编译

官方包

一般情况下，我们建议使用官方包，手动编译也可以；

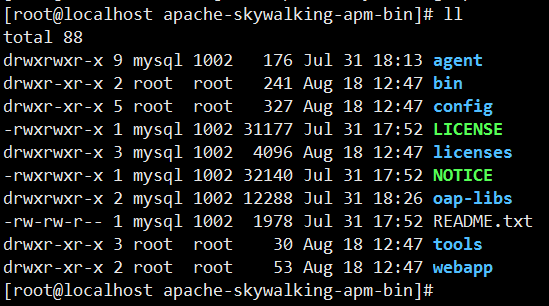
从这里下载：<http://skywalking.apache.org/downloads/>

**2、SkyWalking OAP 搭建**

解压：tar -zxvf apache-skywalking-apm-8.1.0.tar.gz

解压后即完成了安装，不需要做其他操作；

切换：cd apache-skywalking-apm-bin



目录说明：

agent #SkyWalking Agent

bin #执行脚本

config #SkyWalking OAP Server 配置文件

LICENSE

licenses

NOTICE

oap-libs #SkyWalking OAP Server

README.txt

tools

webapp #SkyWalking UI

**3、启动 SkyWalking OAP 服务**

切换到bin目录：./startup.sh

启动后会启动两个服务，一个是skywalking-oap-server，一个是skywalking-web-ui；

查看安装目录下的 ./logs 下的日志文件，检查两个服务的日志文件是否启动成功；

skywalking-oap-server服务启动后会占用：11800 和 12800 两个端口；

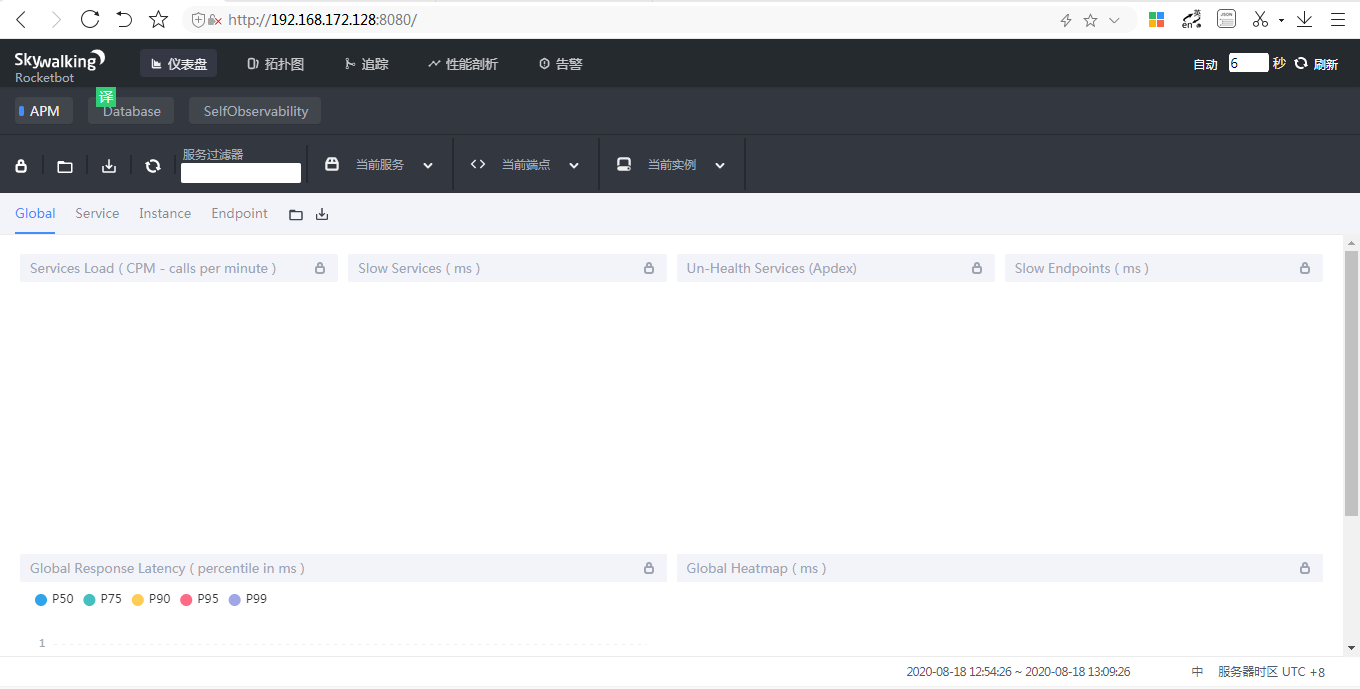
skywalking-web-ui服务会占用 8080 端口；

如果想要修改SkyWalking UI服务的参数，可以编辑webapp/webapp.yml 配置文件，比如：

server.port：SkyWalking UI服务端口，默认是8080；

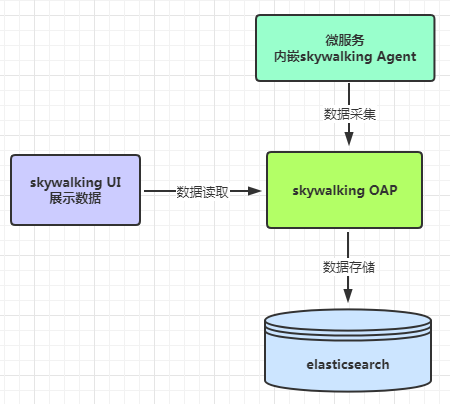
collector.ribbon.listOfServers：SkyWalking OAP服务地址数组，SkyWalking UI界面的数据是通过请求SkyWalking OAP服务来获得；

1. 访问SkyWalking UI界面：<http://192.168.172.128:8080/>



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；

## SkyWalking Agent跟踪微服务



案例一：

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；

#!/bin/sh

# SkyWalking Agent配置

export SW\_AGENT\_NAME=11-springboot #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:/usr/local/apache-skywalking-apm-bin/agent/skywalking-agent.jar

java **$JAVA\_AGENT** -jar 11-springboot-1.0.0.jar #jar启动

在启动程序前加一个-javaagent 参数即可完成对程序的跟踪；

案例二：

在tomcat中部署war包配置SkyWalking Agent来跟踪微服务；

修改/usr/local/apache-tomcat-9.0.31/bin/catalina.sh 文件，在顶部第一行加上：

CATALINA\_OPTS="$CATALINA\_OPTS -javaagent:/usr/local/apache-skywalking-apm-bin/agent/skywalking-agent.jar";

export CATALINA\_OPTS;

如果tomcat端口与skywalking ui端口冲突的话，修改一下tomcat端口；

测试，访问一下项目，然后进入 SkyWalking UI 界面查看跟踪情况，由于上传数据是异步的，访问完项目后，可能需要等几秒才能看到跟踪数据；

## 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，我们可以看到 Spring Boot 应用服务为 "11-springboot"，就是我们在环境变量 SW\_AGENT\_NAME 中所定义的；

**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；

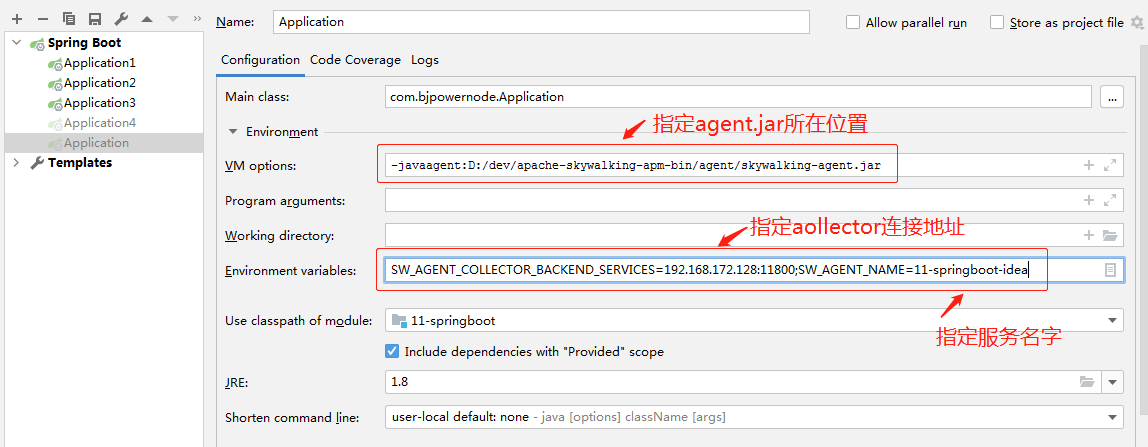
这里我们可以看到 Spring Boot 应用的服务为 {agent\_name}-pid:{pid}@{hostname}，由 Agent 自动生成；

**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；

我们可以看到 Spring Boot 应用的一个端点，为API接口 /index；

## 在IDEA中使用Skywalking

在运行的程序配置jvm参数和环境变量参数，如下图所示：



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

SW\_AGENT\_COLLECTOR\_BACKEND\_SERVICES=127.0.0.1:11800;SW\_AGENT\_NAME=sharding

## 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；

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、网络钩子(Webhook}：当警告触发时，哪些服务终端需要被通知；

3、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；

比如service\_resp\_time\_rule规则：

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 minutes.

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；

只有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；

**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请查阅：

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；

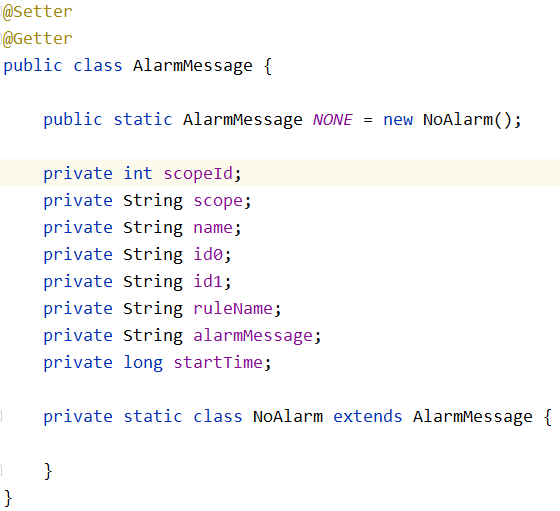
4、id1. 未使用；

5、ruleName. 您在 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；

AlarmMessage类的代码：



提示：根据内容，建议学员自己操作一遍，折腾得越多，技术提升得越多，纸上得来终觉浅，绝知此事要躬行（避免眼高手低，战略上藐视，战术上重视）；