

ch05-服务链路追踪

公众号：锋哥聊编程

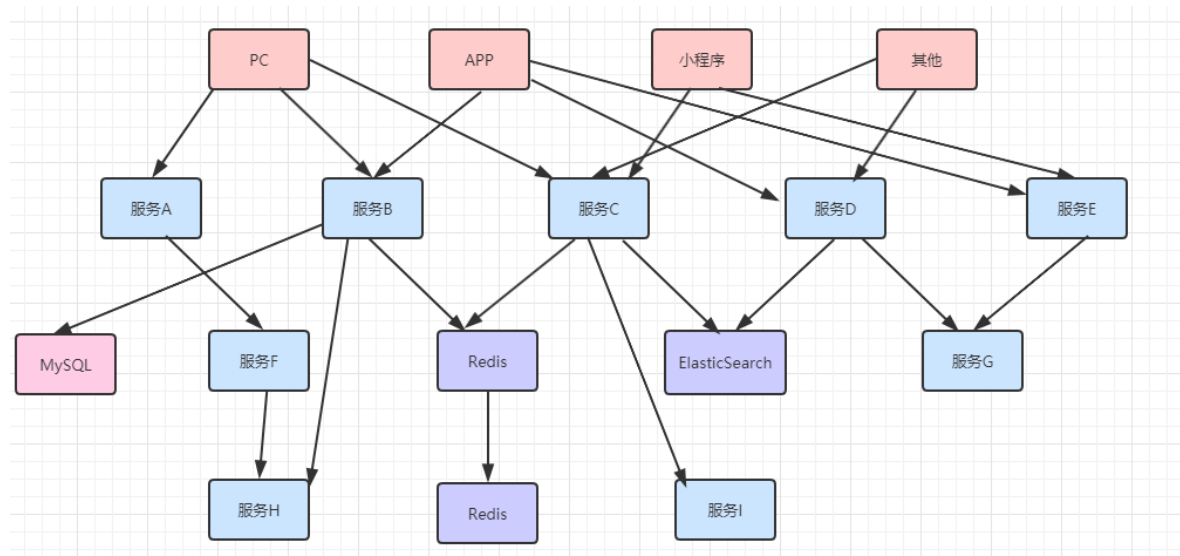
主讲：小锋



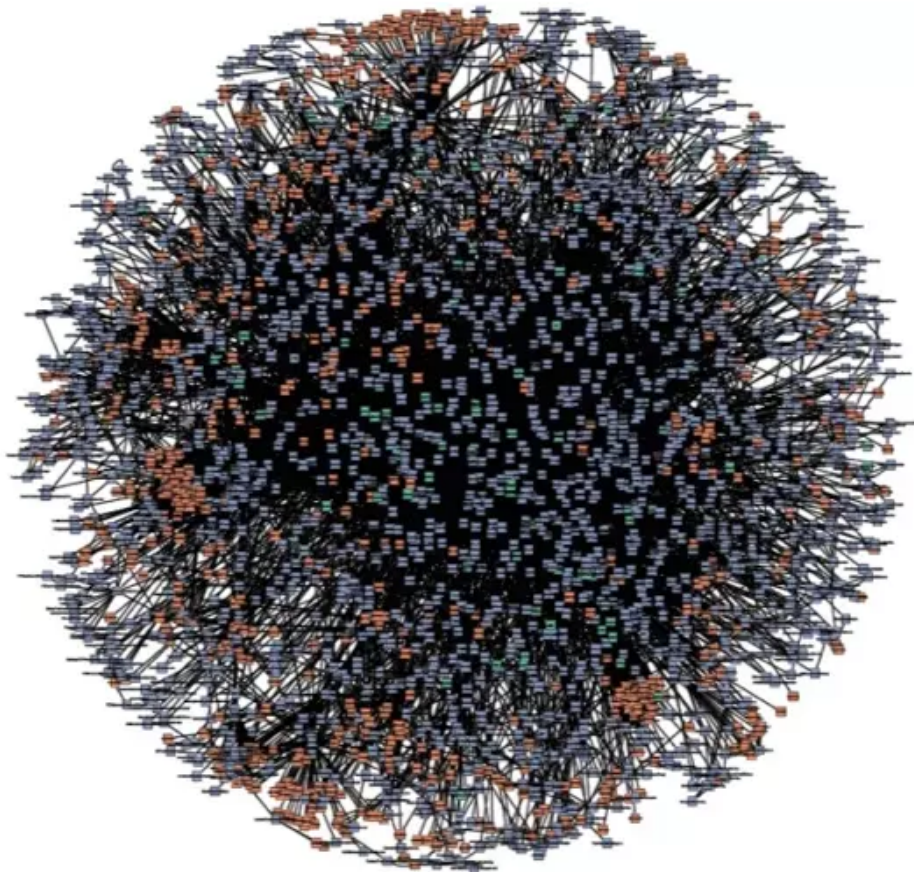
1、什么是分布式链路追踪

为什么需要分布式链路追踪

随着系统设计变得日趋复杂，越来越多的组件开始走向分布式化，如微服务、分布式数据库、分布式缓存等，使得后台服务构成了一种复杂的分布式网络。往往前端的一个请求需要经过多个微服务、跨越多个数据中心才能最终获取到结果，如下图



并且随着业务的不断扩张，服务之间互相调用会越来越复杂，这个庞大的分布式系统调用网络可能会变的如下图所示：



那随之而来的就是我们将会面临的诸多困扰：

- 问题定位：当某一个服务节点出现问题导致整个调用失败，无法快速清晰地定位问题服务。
- 性能分析：服务存在相互依赖调用的关系，当某一个服务接口耗时过长，会导致整个接口调用变的很慢，我们无法明确每一个接口的耗时。
- 服务拓扑图：随着需求迭代，系统之间调用关系变化频繁，靠人工很难梳理清楚系统之间的调用关系。
- 服务告警：当服务出现问题，我们无法做到由系统自动通知相关人员。

为了解决这些问题，分布式链路追踪应运而生。它会将一次分布式请求还原成调用链路，将一次分布式请求的调用情况集中展示，比如各个服务节点上的耗时、请求具体到达哪台机器上、每个服务节点的请求状态、生成服务调用拓扑图等等。也就是说我们要设计并开发一些分布式追踪系统来帮助我们解决这些问题

什么是APM系统

APM (Application Performance Management) 即应用性能管理系统，是对企业系统即时监控以实现对应应用程序性能管理和故障管理的系统化的解决方案。应用性能管理，主要指对企业的业务应用进行监测、优化，提高企业应用的可靠性和质量，保证用户得到良好的服务，降低IT总拥有成本。

APM系统是可以帮助理解系统行为、用于分析性能问题的工具，以便发生故障的时候，能够快速定位和解决问题。

主流的APM产品

PinPoint

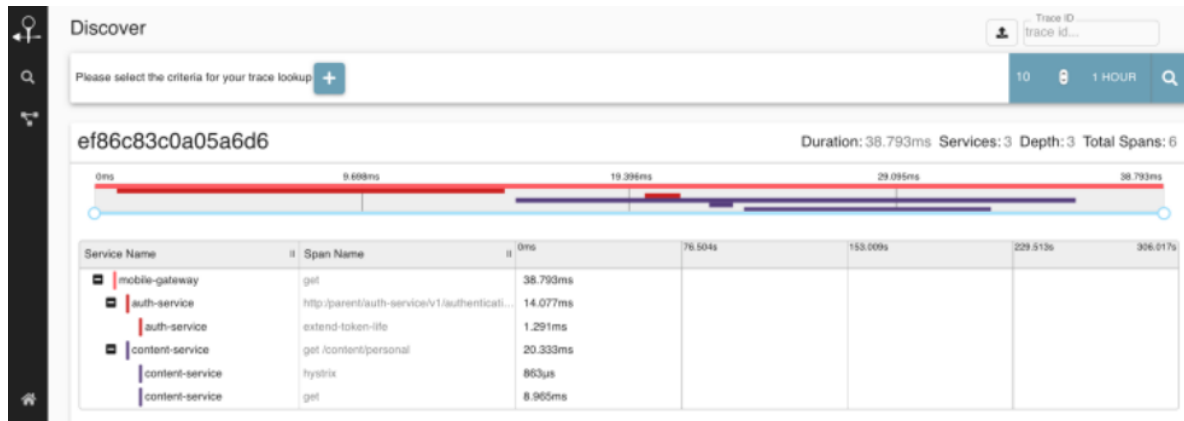
Pinpoint是由一个韩国团队实现并开源，针对Java编写的大规模分布式系统设计，通过JavaAgent的机制做字节代码植入，实现加入traceid和获取性能数据的目的，对应用代码零侵入。

官方网站: <https://github.com/naver/pinpoint>

Zipkin

Zipkin是由Twitter开源，是分布式链路调用监控系统，聚合各业务系统调用延迟数据，达到链路调用监控跟踪。Zipkin基于Google的Dapper论文实现，主要完成数据的收集、存储、搜索与界面展示。

官方网站: <https://zipkin.io/>



CAT

CAT是由大众点评开源的项目，基于Java开发的实时应用监控平台，包括实时应用监控，业务监控，可以提供十几张报表展示。

官方网站: <https://github.com/dianping/cat>

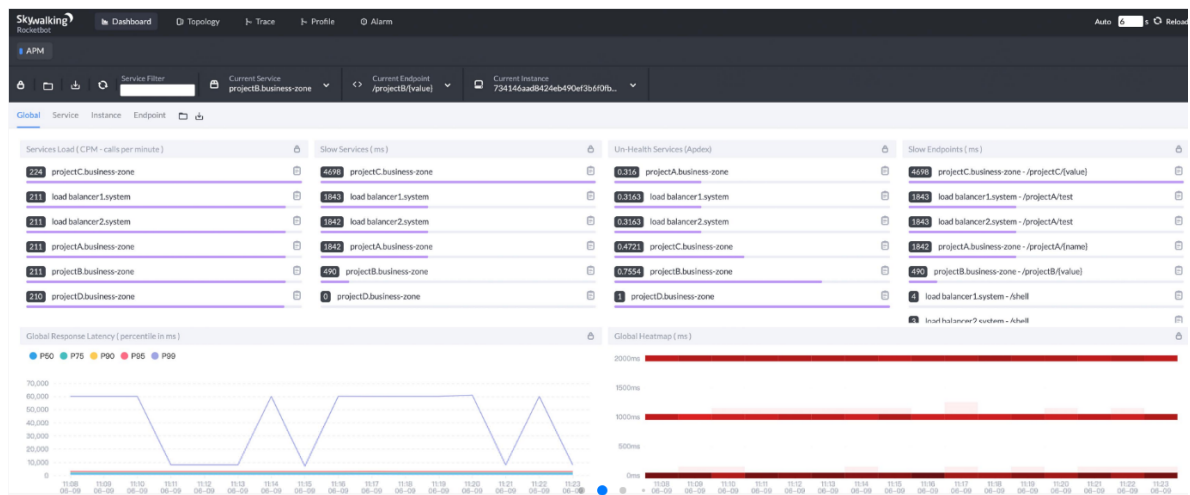
Spring Cloud Sleuth

SpringCloud 提供的分布式系统中链路追踪解决方案。很可惜的是阿里系并没有链路追踪相关的开源项目，我们可以采用**Spring Cloud Sleuth+Zipkin**来做链路追踪的解决方案。

SkyWalking

SkyWalking是Apache基金会下面的一个开源APM项目，为微服务架构和云原生架构系统设计。它通过探针自动收集所需的指标，并进行分布式追踪。通过这些调用链路以及指标，Skywalking APM会感知应用间关系和服务间关系，并进行相应的指标统计。Skywalking支持链路追踪和监控应用组件基本涵盖主流框架和容器，如国产RPC Dubbo和motan等，国际化的spring boot，spring cloud。

官方网站: <http://skywalking.apache.org/>



2、分布式链路追踪（一）：Spring Cloud Sleuth

Spring Cloud Sleuth简介

Spring Cloud Sleuth实现了一种分布式的服务链路跟踪解决方案，通过使用Sleuth可以让我们快速定位某个服务的问题。简单来说，Sleuth相当于调用链监控工具的客户端，集成在各个微服务上，负责产生调用链监控数据。

注意：Spring Cloud Sleuth只负责产生监控数据，通过日志的方式展示出来，并没有提供可视化的UI界面。

学习Sleuth之前必须了解它的几个概念：

Span：基本的工作单元，相当于链表中的一个节点，通过一个唯一ID标记它的开始、具体过程和结束。我们可以通过其中存储的开始和结束的时间戳来统计服务调用的耗时。除此之外还可以获取事件的名称、请求信息等。

Trace：一系列的Span串联形成的一个树状结构，当请求到达系统的入口时就会创建一个唯一ID (traceId)，唯一标识一条链路。这个traceId始终在服务之间传递，直到请求的返回，那么就可以使用这个traceId将整个请求串联起来，形成一条完整的链路。

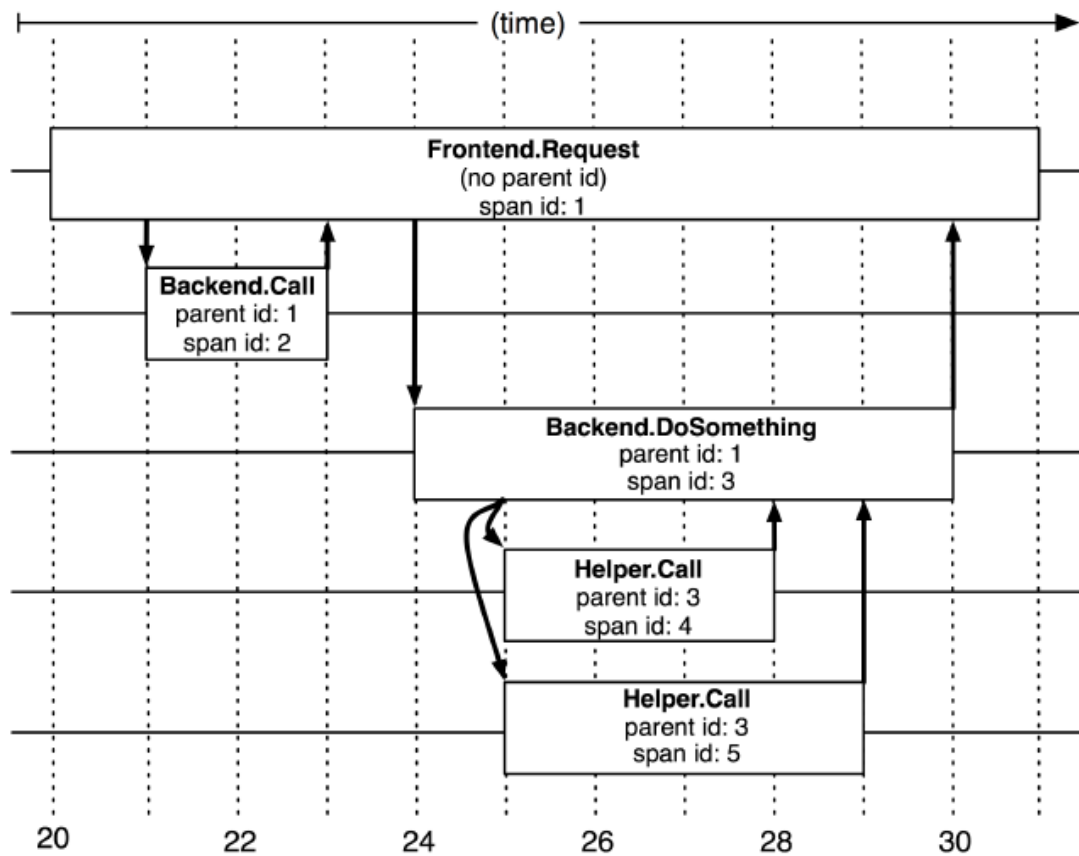
Annotation：一些核心注解用来标注微服务调用之间的事件，重要的几个注解如下：

cs(Client Send)：客户端发出请求，开始一个请求的生命周期

sr (Server Received)：服务端接受请求并处理； $sr - cs = \text{网络延迟} = \text{服务调用的时间}$

ss (Server Send)：服务端处理完毕准备发送到客户端； $ss - sr = \text{服务器上的请求处理时间}$

cr (Client Reveived)：客户端接受到服务端的响应，请求结束； $cr - sr = \text{请求的总时间}$



微服务整合Spring Cloud Sleuth

在mafeng-user和mafeng-order完成以下操作

导入依赖

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-starter-sleuth</artifactId>  
</dependency>
```

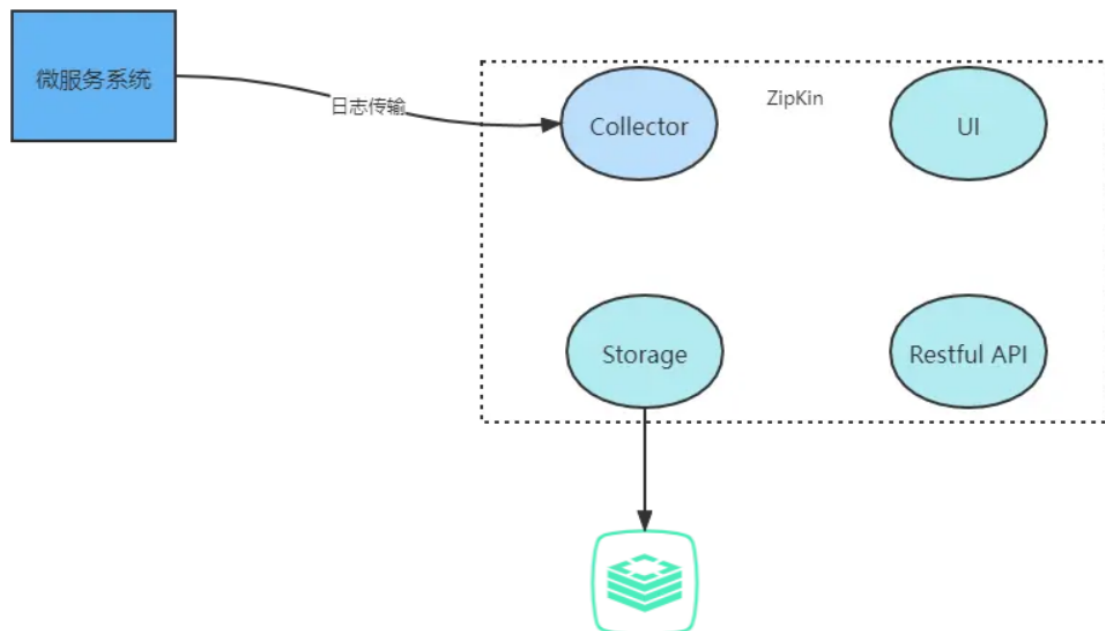
调整日志级别

bootstrap.yml

```
logging:  
  level:  
    org.springframework.cloud.sleuth: debug
```

启动测试

访问mafeng-order远程调用mafeng-user查看日志



Zipkin共分为4个核心的组件，如下：

Collector：收集器组件，它主要用于处理从外部系统发送过来的跟踪信息，将这些信息转换为Zipkin内部处理的 Span 格式，以支持后续的存储、分析、展示等功能。

Storage：存储组件，它主要对处理收集器接收到的跟踪信息，默认会将这些信息存储在内存中，我们也可以修改此存储策略，通过使用其他存储组件将跟踪信息存储到数据库中

RESTful API：API 组件，它主要用来提供外部访问接口。比如给客户端展示跟踪信息，或是外接系统访问以实现监控等。

UI：基于API组件实现的上层应用。通过UI组件用户可以方便而有直观地查询和分析跟踪信息

zipkin分为服务端和客户端，服务端主要用来收集跟踪数据并且展示，客户端主要功能是发送给服务端，微服务的应用也就是客户端，这样一旦发生调用，就会触发监听器将sleuth日志数据传输给服务端。

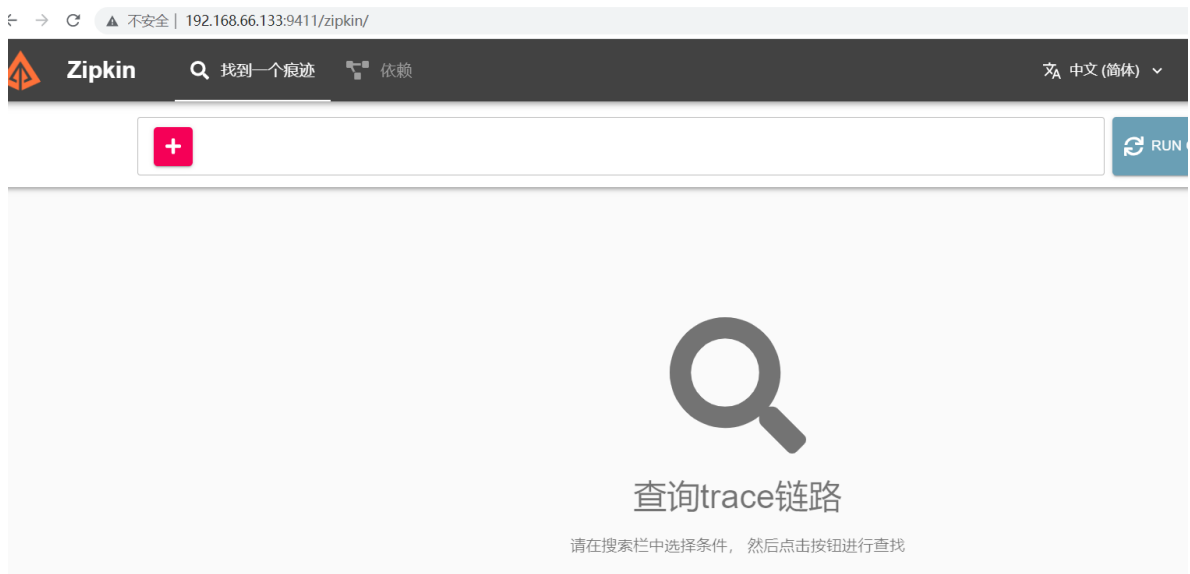
安装Zipkin

- docker安装zipkin-server

```
# 拉到镜像
docker pull openzipkin/zipkin

# 创建容器
docker run -d -p 9411:9411 --name=zipkin openzipkin/zipkin
```

- 启动访问: <http://192.168.66.133:9411>



微服务整合Zipkin

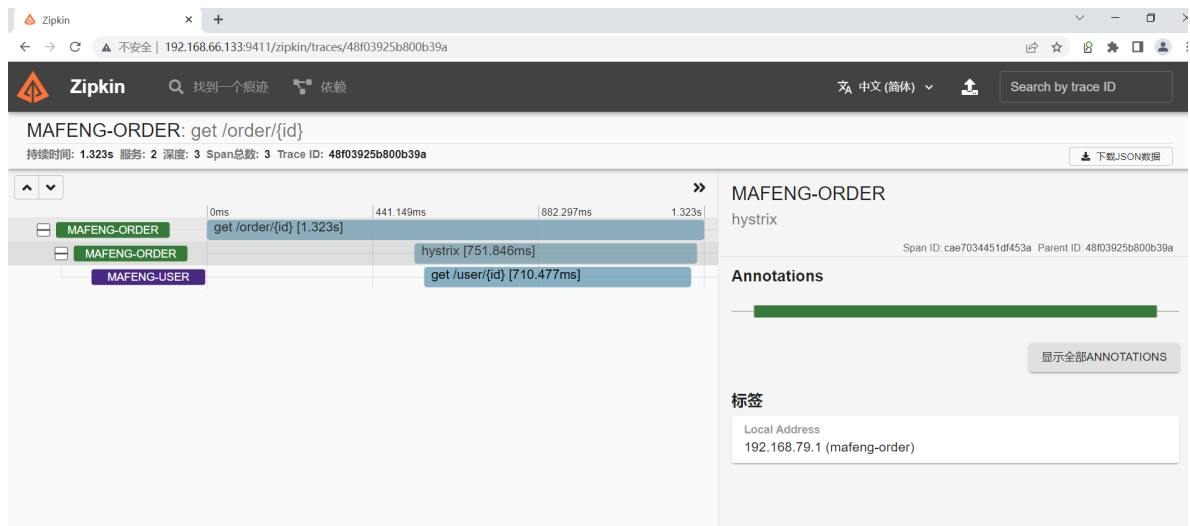
导入依赖

```
<!-- 配置zipkin启动器 -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
```

配置

```
spring:
  zipkin:
    base-url: http://192.168.66.133:9411
  sleuth:
    sampler:
      probability: 1.0 # 日志采样率
```

测试





Zipkin日志持久化

zipkin的信息默认是存储在内存中，服务端一旦重启信息将会丢失，但是zipkin提供了可插拔式的存储。

zipkin支持以下四种存储方式：

- 内存：服务重启将会失效，不推荐
- MySQL：数据量越大性能较低
- Elasticsearch：主流的解决方案，推荐使用
- Cassandra：技术太牛批，用的人少，自己选择，不过官方推荐

我们以Elasticsearch为例。

搭建Elasticsearch服务器

```
docker run --name elasticsearch -d -e ES_JAVA_OPTS="-Xms512m -Xmx512m" -e "discovery.type=single-node" -p 9200:9200 -p 9300:9300 elasticsearch:7.7.0
```

访问：<http://192.168.66.133:9200>


```
{
  "name": "476dc35cd408",
  "cluster_name": "docker-cluster",
  "cluster_uuid": "290XgrDWTMKS-FNufwRenQ",
  "version": {
    "number": "7.7.0",
    "build_flavor": "default",
    "build_type": "docker",
    "build_hash": "81ale9eda8e6183f5237786246f6dced26a10eaf",
    "build_date": "2020-05-12T02:01:37.602180Z",
    "build_snapshot": false,
    "lucene_version": "8.5.1",
    "minimum_wire_compatibility_version": "6.8.0",
    "minimum_index_compatibility_version": "6.0.0-beta1"
  },
  "tagline": "You Know, for Search"
}
```















搭建Kibana客户端

```
docker run -d -p 5601:5601 --name=kibana --link elasticsearch -e
"ELASTICSEARCH_URL=http://192.168.66.133:9200" kibana:7.7.0
```

访问: <http://192.168.66.133:5601>

← → ↻ ⚠ 不安全 | 192.168.66.133:5601/app/kibana#/management/elasticsearch/index_management/indices

 Management / Index Management



Elasticsearch

- [Index Management](#)
- Index Lifecycle Policies
- Transforms
- Rollup Jobs
- Snapshot and Restore
- License Management
- Remote Clusters
- 8.0 Upgrade Assistant

Kibana

- Index Patterns
- Alerts and Actions
- Spaces
- Saved Objects
- Reporting
- Advanced Settings

Index Management

[Indices](#) [Index Templates](#)

Update your Elasticsearch indices individually or in bulk.

No indices to show

Zipkin整合Elasticsearch

删除之前的Zipkin容器，重新创建

```
docker run --name zipkin -d -p 9411:9411 -e STORAGE_TYPE=elasticsearch -e ES_HOSTS=192.168.66.133:9200 openzipkin/zipkin
```

访问接口，发现Elasticsearch中自动创建索引库

Name	Health	Status	Primaries	Replicas	Docs count	Storage size
zipkin-span-2022-11-21	yellow	open	5	1	4	16.3kb

查看数据

```
GET zipkin-span-2022-11-21/_search
{
  "query": {
    "match_all": {}
  }
}
```

```
{
  "hits": {
    "total": {
      "value": 4,
      "relation": "eq"
    },
    "max_score": 1.0,
    "hits": [
      {
        "_index": "zipkin-span-2022-11-21",
        "_type": "doc",
        "_id": "a8474c09f1256849-23c9fa133f34e52353a059564967788b",
        "_score": 1.0,
        "_source": {
          "traceId": "a8474c09f1256849",
          "duration": 681133,
          "localEndpoint": {
            "serviceName": "mafeng-order",
            "ipv4": "192.168.79.1"
          },
          "timestamp_millis": 1669043712322,
          "kind": "CLIENT",
          "name": "get",
          "id": "3b68116173895215",
          "parentId": "3cf3df9c3dec489c",
          "timestamp": 1669043712322009,
          "tags": {
            "http.method": "GET",
            "http.path": "/user/4"
          }
        }
      }
    ]
  }
}
```

4、分布式链路追踪（三）：SkyWalking

SkyWalking简介

Skywalking

2015年由个人吴晟（华为开发者）主导开源，作者是华为开发云监控产品经理，主导监控产品的规划、技术路线及相关研发工作，也是OpenTracing分布式追踪标准组织成员，该项目 2017年加入Apache孵化器，是一个分布式系统的应用程序性能监控工具（APM），专为微服务、云原生架构和基于容器（Docker、K8s、Mesos）架构而设计。

官方站点：<http://skywalking.apache.org/>

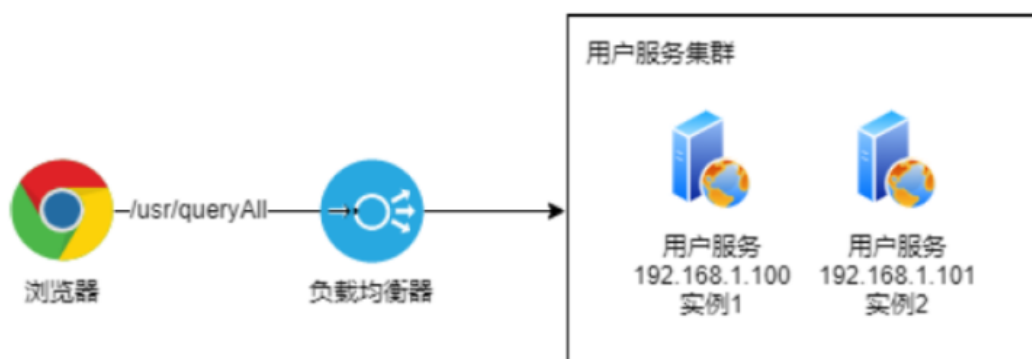
GitHub项目地址：<https://github.com/apache/skywalking>

其核心功能要点如下：

- **指标分析**：服务，实例，端点指标分析
- **问题分析**：在运行时分析代码，找到问题的根本原因
- **服务拓扑**：提供服务的拓扑图分析
- **依赖分析**：服务实例和端点依赖性分析
- **服务检测**：检测慢速的服务和端点
- **性能优化**：根据服务监控的结果提供性能优化的思路
- **链路追踪**：分布式跟踪和上下文传播
- **数据库监控**：数据库访问指标监控统计，检测慢速数据库访问语句（包括SQL语句）
- **服务告警**：服务告警功能

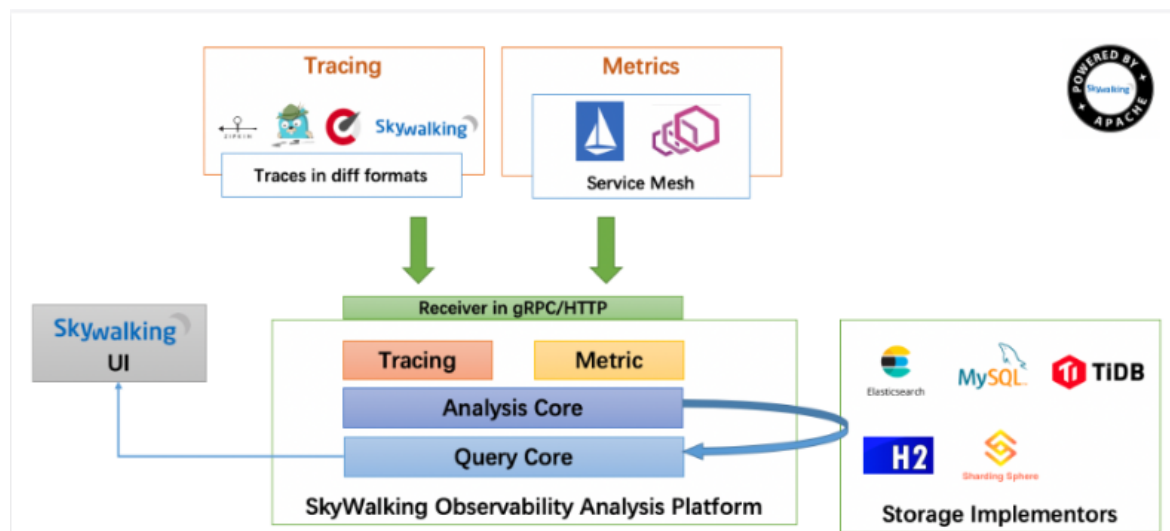
名词解释：

- 服务（service）：业务资源应用系统
- 端点（endpoint）：应用系统对外暴露的功能接口
- 实例（instance）：物理机



SkyWalking架构设计

SkyWalking的整体架构设计如下图所示：



skyWalking整体可分为：客户端，服务端

客户端：agent组件

基于探针技术采集服务相关信息（包括跟踪数据和统计数据），然后将采集到的数据上报给skywalking的数据收集器

服务端：又分为OAP，Storage，WebUI

OAP：observability analysis platform可观测性分析平台，负责接收客户端上报的数据，对数据进行分析，聚合，计算后将数据进行存储，并且还会提供一些查询API进行数据的查询，这个模块其实就是我们所说的链路追踪系统的Collector收集器

Storage：skyWalking的存储介质，默认是采用H2，同时支持许多其他的存储介质，比如：ElasticSearch，mysql等

WebUI：提供一些图形化界面展示对应的跟踪数据，指标数据等等

搭建SkyWalking服务端

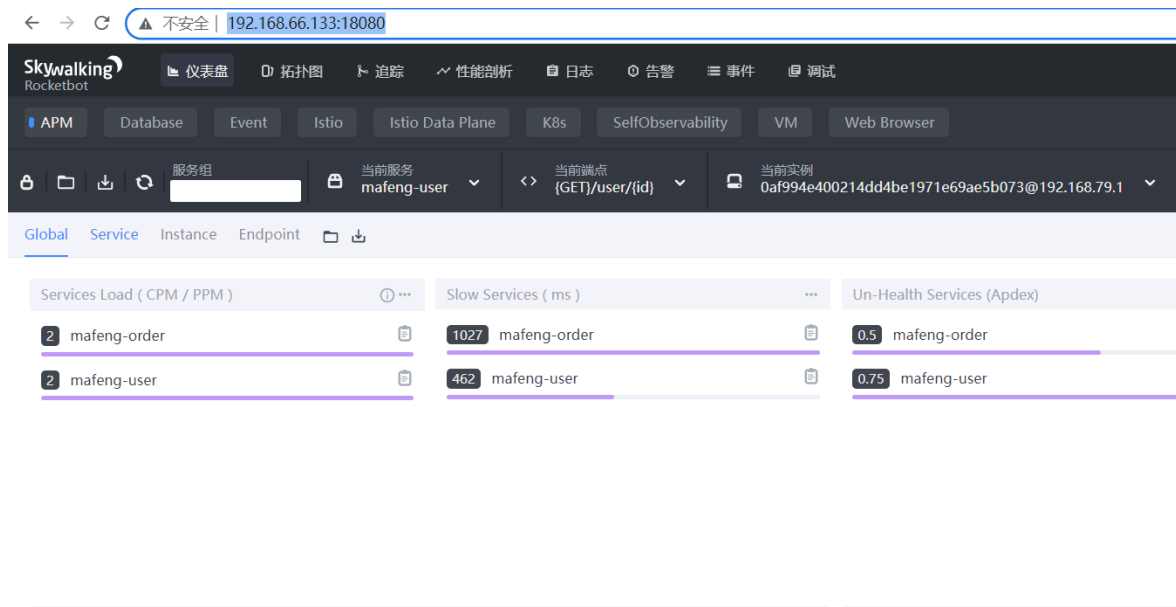
SkyWalking-OAP

```
docker run --name oap --restart always -d \
-e TZ=Asia/Shanghai \
-p 12800:12800 \
-p 11800:11800 \
-e SW_STORAGE=elasticsearch7 \
-e SW_STORAGE_ES_CLUSTER_NODES=192.168.66.133:9200 \
--privileged=true \
apache/skywalking-oap-server:8.7.0-es7
```

SkyWalking-UI

```
docker run --name oap-ui --restart always -d -p 18080:8080 -e
SW_OAP_ADDRESS=http://192.168.66.133:12800 apache/skywalking-ui
```

访问：<http://192.168.66.133:18080/>



微服务整合SkyWalking

探针在下载好的 tar 包中，我们直接把 tar 包的 agent 目录解压出来即可。

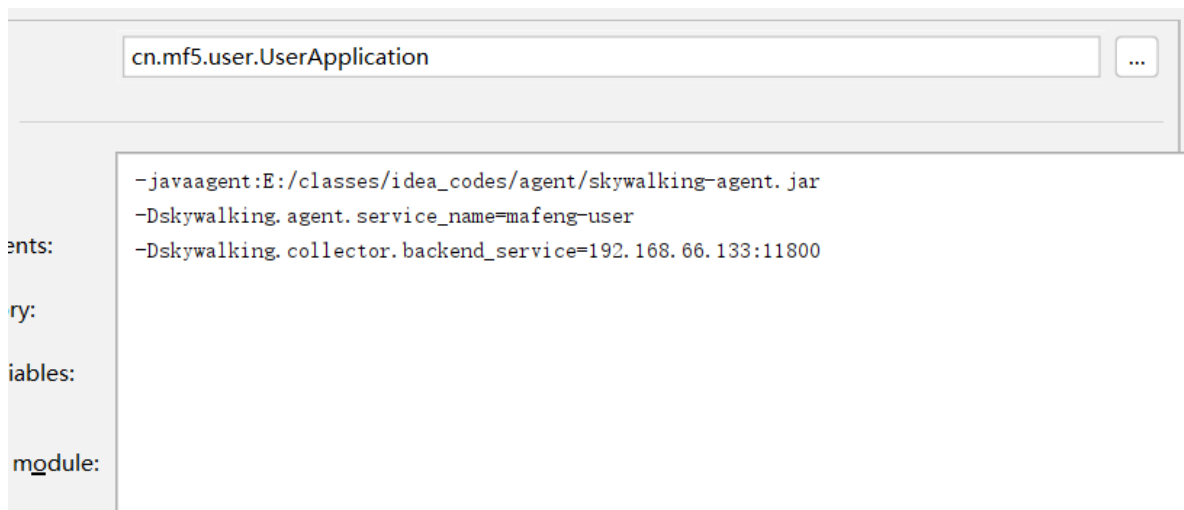
电脑 > 本地磁盘 (E:) > classes > idea_codes > agent

名称	修改日期
activations	2022/11/22 0:10
bootstrap-plugins	2022/11/22 0:10
config	2022/11/22 0:01
logs	2022/11/22 0:10
optional-plugins	2022/11/22 0:10
optional-reporter-plugins	2022/11/22 0:01
plugins	2022/11/22 0:10
skywalking-agent.jar	2021/7/30 20:35

skywalking-agent.jar 就是探针的包。

skywalking是无侵入性的，在开发阶段，我们不用改代码，只需要在部署启动时加入一些参数即可。

现在为了测试，我们在开发阶段，直接在idea工具中配置即可，在启动项右键，选择 Edit Configuration



在展开的输入框中，输入下面的配置：

User微服务的VM配置：

```
-javaagent:E:/classes/idea_codes/agent/skywalking-agent.jar
-Dskywalking.agent.service_name=mafeng-user
-Dskywalking.collector.backend_service=192.168.66.133:11800
```

Order微服务的VM配置：

```
-javaagent:E:/classes/idea_codes/agent/skywalking-agent.jar
-Dskywalking.agent.service_name=mafeng-order
-Dskywalking.collector.backend_service=192.168.66.133:11800
```

注意：

- `-javaagent:E:/classes/idea_codes/skywalking-agent.jar`：配置的是skywalking-agent.jar这个包的位置，要修改成你自己存放的目录
- `-Dskywalking.agent.service_name=bhd-eureka`：是当前项目的名称，需要改成当前微服务的名称
- `-Dskywalking.collector.backend_service=192.168.66.133:11800`：是skywalking的OPA服务地址，采用的是GRPC通信，因此端口是11800，不是8080

加入了配置之后，正常启动即可。

访问测试

