

## 第九章 Spring Cloud 调用链跟踪

一样的在线教育，不一样的教学品质



# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置

## 小节导学

### 问题

- 跨服务调用发生异常，如何快速定位问题
- 跨服务调用有性能瓶颈，如何快速定位瓶颈位置

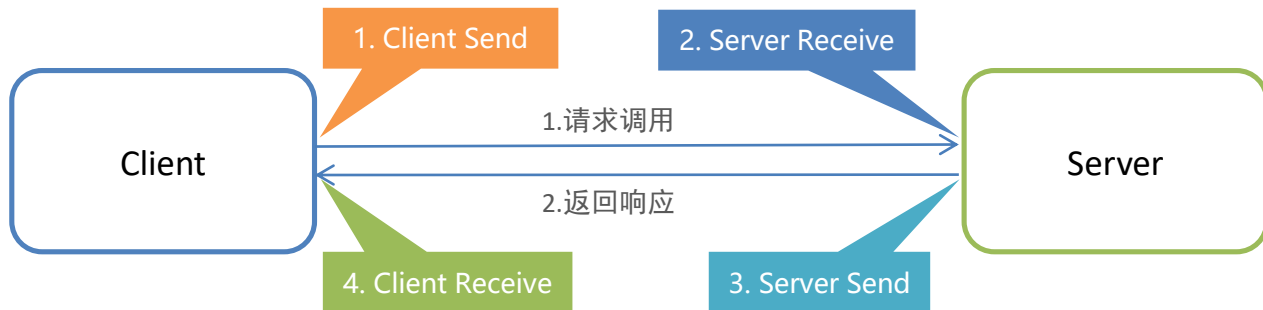
调用链跟踪可以帮助我们解决跨服务调用的这2种问题，那么解决的思路是怎样的呢？

本节我们就分析一下调用链跟踪的基本原理。



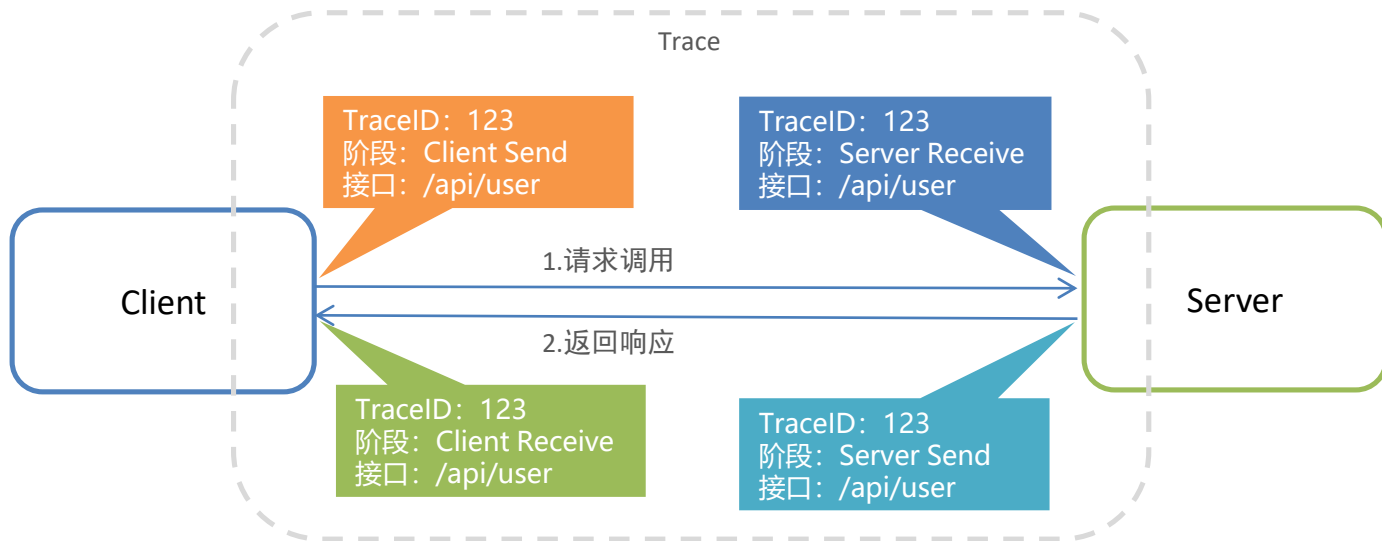
## 服务调用关键点

1. Client 发起调用 (Client Send)
2. Server 接收请求 (Server Receive)
3. Server 发送响应 (Server Send)
4. Client 接收响应 (Client Receive)



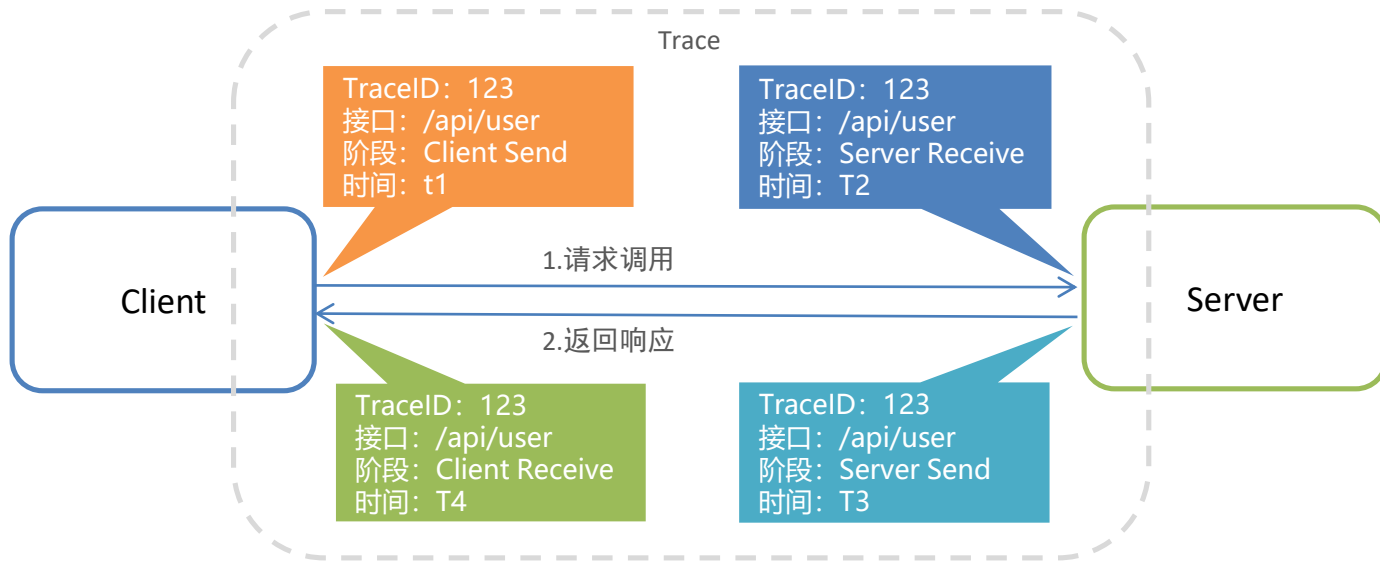
## 解决异常问题

一次完整的请求包含这4个关键点，我们只需要**跟踪每个关键点**，就可以知道调用是否正常。  
少了哪个阶段的跟踪记录，就说明这个阶段异常了。



## 解决性能问题

记录每个阶段的**时间戳**，就可以计算任意阶段的耗时。





## 总结

### 重难点

1. 服务调用的4个关键点
2. 调用链跟踪解决异常问题的思路
3. 调用链跟踪解决性能问题的思路

# 总结

## 服务整合 sleuth

实践 Service 中如何整合 sleuth 来采集调用信息





# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置

## 小节导学

SpringCloud Sleuth 是 SpringCloud 的分布式调用链解决方案，Sleuth 对用户透明，服务调用的交互信息都能被自动采集，可以通过日志文件获取服务调用的链路数据，也可以将数据发送给远程服务统一收集分析。

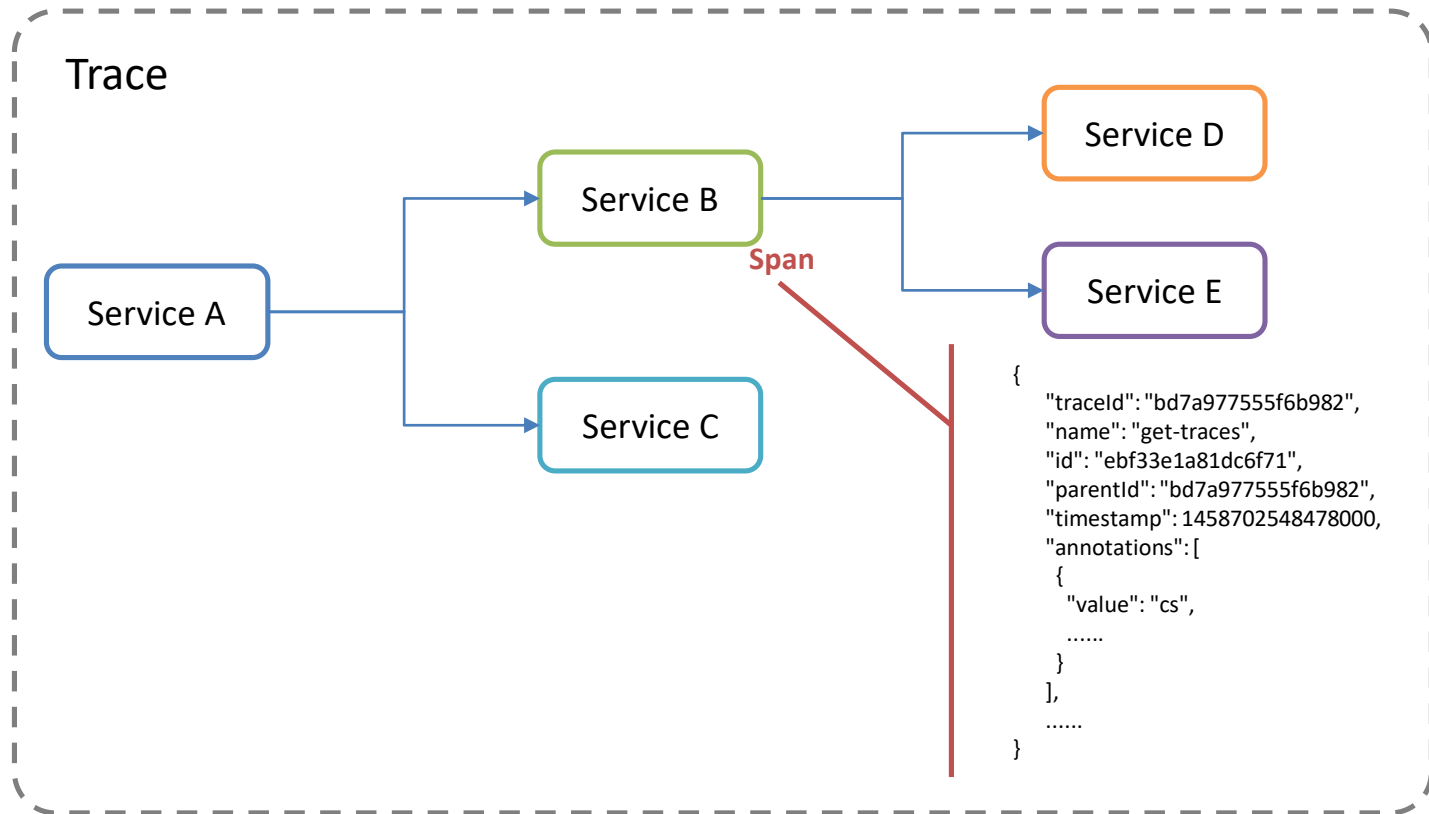
本节我们就实践一下 Sleuth 与服务的整合。

- Sleuth 的术语
- 服务整合 Sleuth 实践

# 1. Sleuth 的术语

名称	含义	
Span	基本工作单元，64位的ID标识，发送一次请求就是一个新的Span。	
Trace	一次完整的调用请求，包含一组 Span。	
Annotation	描述事件的实时状态。	
	cs (client send)	客户端发起，表示一个span开始。
	sr (server received)	服务端解决请求开始处理。
	ss (server send)	服务端返回响应数据。
	cr (client received)	客户端收到响应数据。

# 1. Sleuth 的术语



## ■ 2. 服务整合 Sleuth 实践

### 步骤

#### 1. 添加依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
    <scope>provided</scope>
</dependency>
```

## ■ 2. 服务整合 Sleuth 实践

### 步骤

#### 2. 接口

```
@Slf4j
@RestController
public class TestController {
    @GetMapping("/hi")
    public String hi() {
        log.info("hi");
        return "hi";
    }
}
```

## ■ 2. 服务整合 Sleuth 实践

### 步骤

#### 3. 测试

访问 `http://localhost:8080/hi`, 输出日志:

```
... INFO [sleuth-consumer, faae8eda1df0a3c8, faae8eda1df0a3c8, false] ... : hi
```

服务名称

TraceID

SpanID

是否上报



## 总结

### 重难点

1. Sleuth 重要术语
2. 服务整合 Sleuth 的方法
3. Sleuth 日志结构



# 总结

## Sleuth 如何支持各种服务调用方式?

1. RestTemplate
2. Feign
3. 多线程



# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置

# ■ Sleuth 对 Feign, RestTemplate, 多线程 的支持

## 小节导学

RestTemplate 与 Feign 都是服务调用的常用方式，还有多线程调用也是可能的。

使用他们调用远程服务时，Sleuth 可以正常跟踪吗？

本节就实践一下这3种方式整合 Sleuth 的效果。

我们会创建2个服务：sleuth-provider、sleuth-consumer，都集成 Sleuth，sleuth-consumer 中分别使用 RestTemplate 和 Feign、多线程的方式调用 sleuth-provider，查看调用效果。

- RestTemplate 实践
- Feign 实践
- 多线程实践

# ■ 1. RestTemplate 实践

## 步骤

### 1. 创建 RestTemplate Bean

```
@Bean public RestTemplate restTemplate() {  
    return new RestTemplate();  
}
```

### 2. RestTemplate 调用

```
@Autowired  
RestTemplate restTemplate;  
  
@GetMapping("/helloByRestTemplate")  
public String helloByRestTemplate() {  
    String result =  
        restTemplate.getForObject(url, String.class);  
    log.info("resttemplate result: {}", result);  
    return result;  
}
```

# ■ 1. RestTemplate 实践

## 验证

访问 Consumer 的接口 “/helloByRestTemplate”

consumer 与 provider 都应输出 sleuth 结构的日志

# ■ 1. RestTemplate 实践

## 验证

访问 Consumer 的接口 “/helloByRestTemplate”

consumer 与 provider 都应输出 sleuth 结构的日志

## 结论

Sleuth 可以**正常跟踪** RestTemplate。



## ■ 2. Feign 实践

### 步骤

#### 1. 创建 FeignClient 接口

```
@FeignClient(name = "sleuth-provider"
            ,url = "localhost:8081")

public interface HelloService {

    @RequestMapping("/hello")
    String sayHello();

}
```

#### 2. sleuth-consumer Feign调用

```
@Autowired

private HelloService helloService;

@GetMapping("/helloByFeign")

public String helloByFeign(String name){

    String result = helloService.sayHello();

    log.info("Feign result: {}",result);

    return result;

}
```

## ■ 2. Feign 实践

### 验证

访问 Consumer 的接口 “/helloByFeign”

consumer 与 provider 都应输出 sleuth 结构的日志



## ■ 2. Feign 实践

### 验证

访问 Consumer 的接口 “/helloByFeign”

consumer 与 provider 都应输出 sleuth 结构的日志

### 结论

Sleuth 可以正常跟踪 Feign。



## ■ 3. 多线程实践

### 步骤

#### 1. 定义线程服务

```
@Autowired  
private ExecutorService executorService = Executors.newFixedThreadPool(1);
```

#### 2. 开启线程

```
@GetMapping("/helloByThread")  
public String helloByNewThread() throws ExecutionException, InterruptedException {  
    Future future = executorService.submit() -> {  
        log.info("in thread");  
        return helloService.sayHello();  
    };  
    .....  
}
```

## ■ 3. 多线程实践

### 验证

访问 Consumer 的接口 `"/helloByThread"`

consumer 与 provider 都应输出 sleuth 结构的日志

## ■ 3. 多线程实践

### 验证

访问 Consumer 的接口 `"/helloByThread"`

consumer 与 provider 都应输出 sleuth 结构的日志

### 结论

Sleuth 可以不能跟踪 Thread 线程。



## ■ 3. 多线程实践

### 解决方法

使用 Sleuth 提供的可跟踪的线程服务：

```
@Autowired
BeanFactory beanFactory;

@Bean
public ExecutorService executorService() {
    ExecutorService executorService = Executors.newFixedThreadPool(2);
    return new TraceableExecutorService(this.beanFactory, executorService);
}
```



## 总结

### 重难点

1. RestTemplate、Feign 正常使用
2. 多线程需使用 Sleuth 提供的多线程实现



## 总结

### 重难点

1. RestTemplate、Feign 正常使用
2. 多线程需使用 Sleuth 提供的多线程实现

### 下节

如何让 Sleuth 帮我们传递信息？



# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置



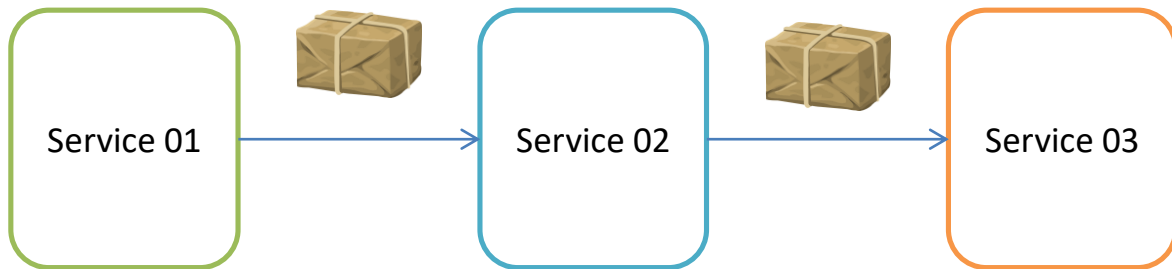
## 小节导学

Sleuth 对服务的调用链进行跟踪，在服务间传递跟踪数据，例如 Span 信息。

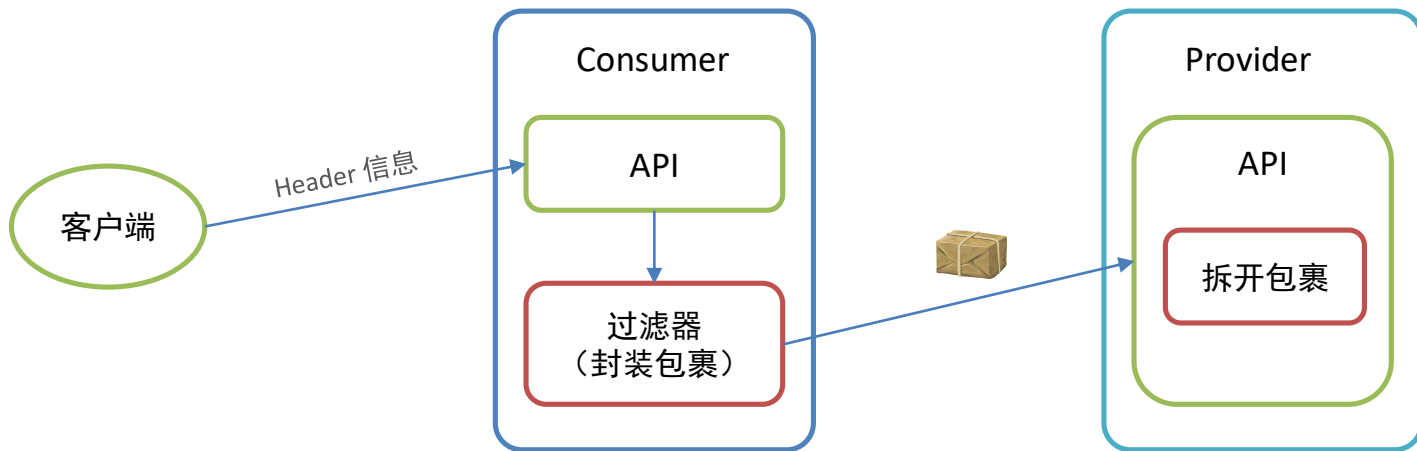
那么我们可不可以传递一些自定义的信息呢？

通过 **tracefilter**（过滤拦截 Span 信息）和 **baggage**（包裹，key/value 对）可以实现。

本节我们在 consumer 中添加一个 filter，把 header 中的一个信息作为 baggage 传递到 provider。



## 整体流程



## 步骤

### 1. Consumer 创建 filter

```
@Component
public class MyTraceFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
        throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        ExtraFieldPropagation.set("BaggageId", httpRequest.getHeader("BaggageId"));
        filterChain.doFilter(request, response);
    }
}
```

## 步骤

### 2. Provider 拆开包裹

```
@GetMapping("/baggage")  
public String sayHello(String name) {  
    return "hello, " +  
        ExtraFieldPropagation.get("BaggageId");  
}
```

### 3. Provider、Consumer 设置属性

```
spring:  
  application:  
    name: ...  
  sleuth:  
    baggage-keys:  
      - BaggageId
```



## 总结

### 重难点

1. Sleuth 自定义传递数据的方法

( `ExtraFieldPropagation` )



## 总结

### 重难点

1. Sleuth 自定义传递数据的方法

( `ExtraFieldPropagation` )

### 下节

调用链可视化分析 ZipKin



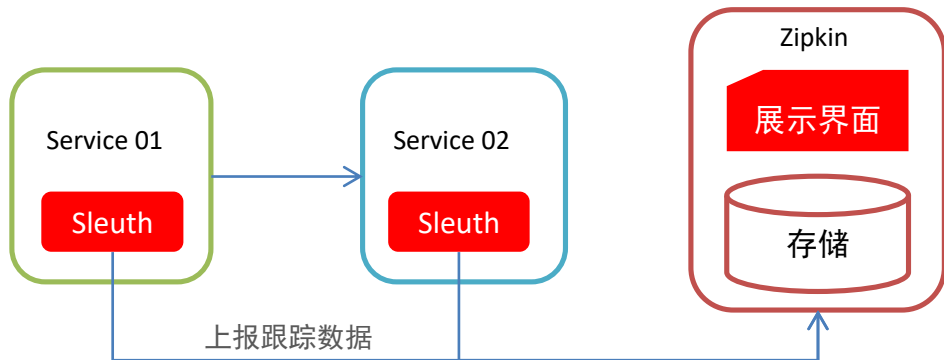
# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置

## 小节导学

前面我们实践 Sleuth 时都是看的日志信息，因为 Sleuth 是做调用链信息采集的，如何方便的查看这些调用信息呢？可以使用 **Zipkin**，**Twitter 开源的分布式跟踪系统**，主要用来收集系统的时序数据，从而跟踪系统的调用问题，类似数据库 + 管理界面。

- Zipkin 搭建
- Sleuth 整合 Zipkin 实践







# 1. Zipkin 搭建

## 步骤

### 1. 编译运行

# 下载源码

```
> git clone https://github.com/openzipkin/zipkin
```

```
> cd zipkin
```

# 编译

```
> ./mvnw -DskipTests --also-make -pl zipkin-server clean install
```

# 运行

```
java -jar ./zipkin-server/target/zipkin-server-*exec.jar
```



# 1. Zipkin 搭建

## 步骤

### 1. 编译运行

访问 `http://localhost:9411/`

Zipkin 查找 已保存 依赖 Try Lens UI  搜索

服务名

Span名称

时间

all

Span Name

1小时

根据Annotation查询

持续时间 (μs) >=

数量

排序

For example: http.path=/foo/bar/ and cluster=foo and cache.miss

Ex: 100ms or 5s

10

耗时降序

查找

?

请选择查找的条件。

## ■ 2. Sleuth 整合 Zipkin 实践

### 步骤

#### 1. provider 与 consumer 中添加依赖

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

去掉之前的 `spring-cloud-starter-sleuth`

## ■ 2. Sleuth 整合 Zipkin 实践

### 步骤

#### 2. provider 与 consumer 中添加 zipkin 配置

```
spring:
  application:
    name: ...
  sleuth:
    sampler:
      probability: 1.0
  zipkin:
    base-url: http://localhost:9411
```

## 2. Sleuth 整合 Zipkin 实践

### 步骤

#### 3. 测试

访问 `http://localhost:8080/hi`

服务名: all | Span名称: all | 时间: 1小时

根据Annotation查询: For example: http.path=/foo/bar/ and cluster=foo and cache.miss

持续时间 (μs) >=: Ex: 100ms or 5s | 数量: 10 | 排序: 耗时降序

查找 ?

Showing: 1 of 1

Services: all

JSON

209.326ms 3 spans

sleuth-consumer x2 209.326ms | sleuth-provider x1 48.453ms

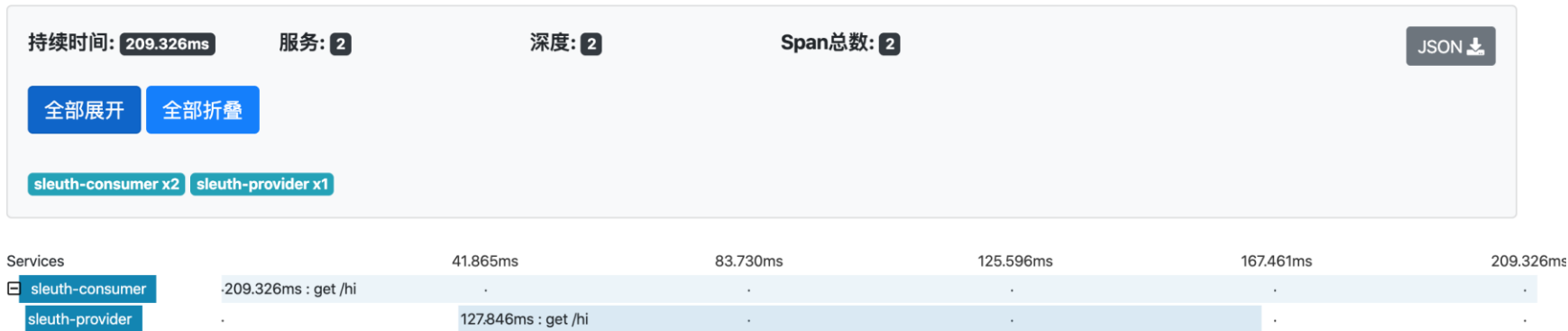
3 minutes ago

## 2. Sleuth 整合 Zipkin 实践

### 步骤

#### 3. 测试

访问 `http://localhost:8080/hi`

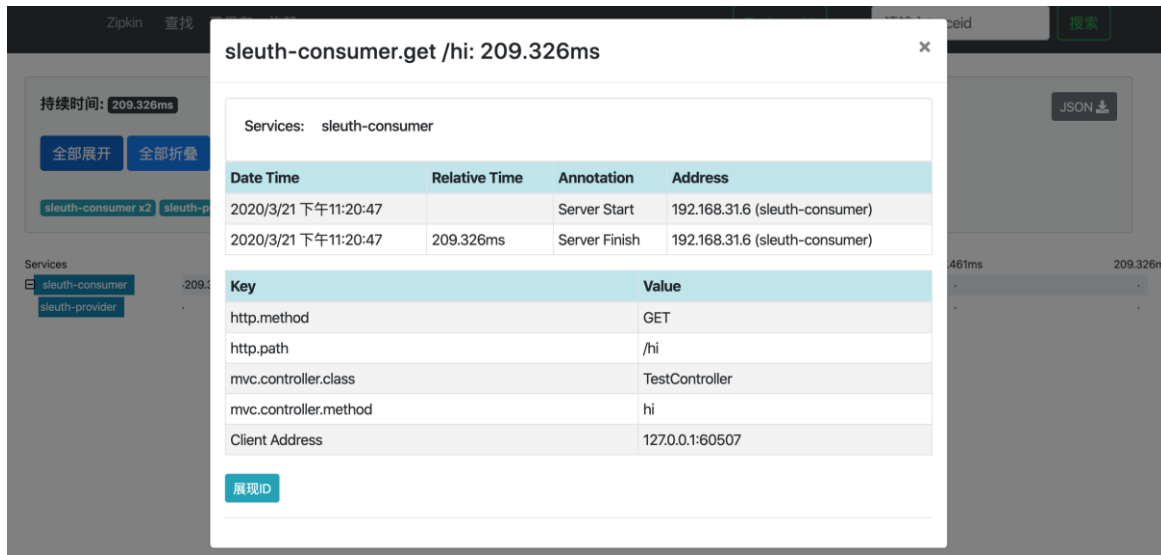


## ■ 2. Sleuth 整合 Zipkin 实践

### 步骤

#### 3. 测试

访问 `http://localhost:8080/hi`



The screenshot displays the Zipkin web interface. A modal window titled "sleuth-consumer.get /hi: 209.326ms" is open, showing the details of a specific trace. The modal contains a table with the following data:

Date Time	Relative Time	Annotation	Address
2020/3/21 下午11:20:47		Server Start	192.168.31.6 (sleuth-consumer)
2020/3/21 下午11:20:47	209.326ms	Server Finish	192.168.31.6 (sleuth-consumer)

Key	Value
http.method	GET
http.path	/hi
mvc.controller.class	TestController
mvc.controller.method	hi
Client Address	127.0.0.1:60507

Below the tables, there is a button labeled "展现ID".



## 总结

### 重难点

1. Sleuth 整合 Zipkin (添加zipkin依赖、配置zipkin地址)





## 总结

### 重难点

1. Sleuth 整合 Zipkin (添加zipkin依赖、配置zipkin地址)

### 下节

使用 Elasticsearch 持久化跟踪数据



# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置

## 小节导学

Zipkin 支持的数据存储方式:

1. In-Memory (用于测试)
2. Cassandra (熟悉度低)
3. Elasticsearch (熟悉度高, 分布式, 高性能, 适合)
4. MySQL (熟悉度高, 性能不高)

官方文档(<https://github.com/openzipkin/zipkin>)已经说明: 使用 Elasticsearch 时, 需要一个 “spark job” 组件配合使用, 用于分析依赖关系。

本节我们的目标是以下几点:

- Elasticsearch 搭建
- Zipkin 整合 Elasticsearch
- Zipkin 依赖关系图

# ■ 1. Elasticsearch 搭建

## 步骤

### 1. 下载

官网 <https://www.elastic.co>

官方说明需要使用 elasticsearch 5+

目前最高版本是 7。

elasticsearch 7 需要 JDK11

如果使用的是 JDK8，需要下载 elasticsearch 6

### 2. 启动

注意：需要使用普通用户启动，不能使用 root

# 解压

```
> tar xzf elasticsearch-xxx.tar.gz
```

# 启动

```
> bin/elasticsearch
```



# 1. Elasticsearch 搭建

## 步骤

### 3. 测试

访问 `http://IP:9200`

```
{
  "name": "vk9nTuR",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "eS0pPcfARWGlDg69bxEww",
  "version": {
    "number": "6.8.2",
    "build_flavor": "default",
    "build_type": "tar",
    "build_hash": "b506955",
    "build_date": "2019-07-24T15:24:41.545295Z",
    "build_snapshot": false,
    "lucene_version": "7.7.0",
    "minimum_wire_compatibility_version": "5.6.0",
    "minimum_index_compatibility_version": "5.0.0"
  },
  "tagline": "You Know, for Search"
}
```

## ■ 2. Zipkin 整合 Elasticsearch

### 步骤

#### 1. 启动 Zipkin 连接 Elasticsearch

- STORAGE\_TYPE - 指定存储类型
- ES\_HOSTS - 指定 Elasticsearch 的地址，多个地址用 “,” 分隔

```
> STORAGE_TYPE=elasticsearch ES_HOSTS=es.me:9200 java -jar zipkin-server-2.12.9-exec.jar
```

## 2. Zipkin 整合 Elasticsearch

### 步骤

#### 2. 测试

访问接口 `http://localhost:8080/hi`, Zipkin 界面中可以正常得到数据

服务名

Span名称

sleuth-consumer

all

根据Annotation查询

For example: http.path=/foo/bar/ and cluster=foo and cache.miss

查找

?

Showing: 1 of 1

Services: sleuth-consumer

103.447ms 3 spans

sleuth-consumer 100%

sleuth-consumer x2 103.447ms sleuth-provider x1 31.456ms

## ■ 2. Zipkin 整合 Elasticsearch

### 步骤

#### 2. 测试

重启 Zipkin，仍然可以显示数据，说明数据存储成功。

查看 Elasticsearch 的 Index 列表，可以看到 Zipkin：

```
> curl -X GET "es.me:9200/_cat/indices"
```

```
yellow open zipkin:span-2020-03-22 KrfdimqYTC23x5ocXlpReg 5 1 3 0 20.5kb 20.5kb
```



### 3. Zipkin 依赖关系图

## 问题

使用 Elasticsearch 之后，依赖关系为空

[Zipkin](#) [查找](#) [已保存](#) [依赖](#) [Try Lens UI](#)  [搜索](#)

开始时间

结束时间

[依赖分析](#)

## ■ 3. Zipkin 依赖关系图

### 解决方案 - 安装 zipkin-dependencies

#### 1. 下载 zipkin-dependencies

`https://search.maven.org/remote_content?g=io.zipkin.dependencies&a=zipkin-dependencies&v=LATEST`

或

```
> curl -sSL https://zipkin.io/quickstart.sh | bash -s io.zipkin.dependencies:zipkin-dependencies:LATEST zipkin-dependencies.jar
```

#### 2. 启动

```
> STORAGE_TYPE=elasticsearch ES_HOSTS=localhost:9200 java -jar zipkin-dependencies-2.3.2.jar
```

## 3. Zipkin 依赖关系图

### 解决方案 - 安装 zipkin-dependencies

#### 3. 测试



注意:

zipkin-dependencies 是**单次运行**，不是启动后就一直在后台分析，每次启动分析一次  
分析的粒度为**天**，如不指定日期，默认为当天，指定日期运行:

```
> STORAGE_TYPE=elasticsearch ES_HOSTS=localhost:9200 java -jar zipkin-dependencies-x.jar 2020-06-01
```



## 总结

### 重难点

1. Zipkin 整合 Elasticsearch (启动是指定ES存储方式、地址)
2. 整合 Elasticsearch 后依赖关系图的分析方法 (zipkin-dependencies)



## 总结

### 重难点

1. Zipkin 整合 Elasticsearch (启动是指定ES存储方式、地址)
2. 整合 Elasticsearch 后依赖关系图的分析方法 (zipkin-dependencies)

### 下节

**SkyWalking** 方便易用



# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置

## 小节导学

SkyWalking 是一个**监测分析平台**，也是一个**APM** (Application Performance monitor) 系统。  
提供分布式跟踪、数据分析聚合、可视化展示等功能。

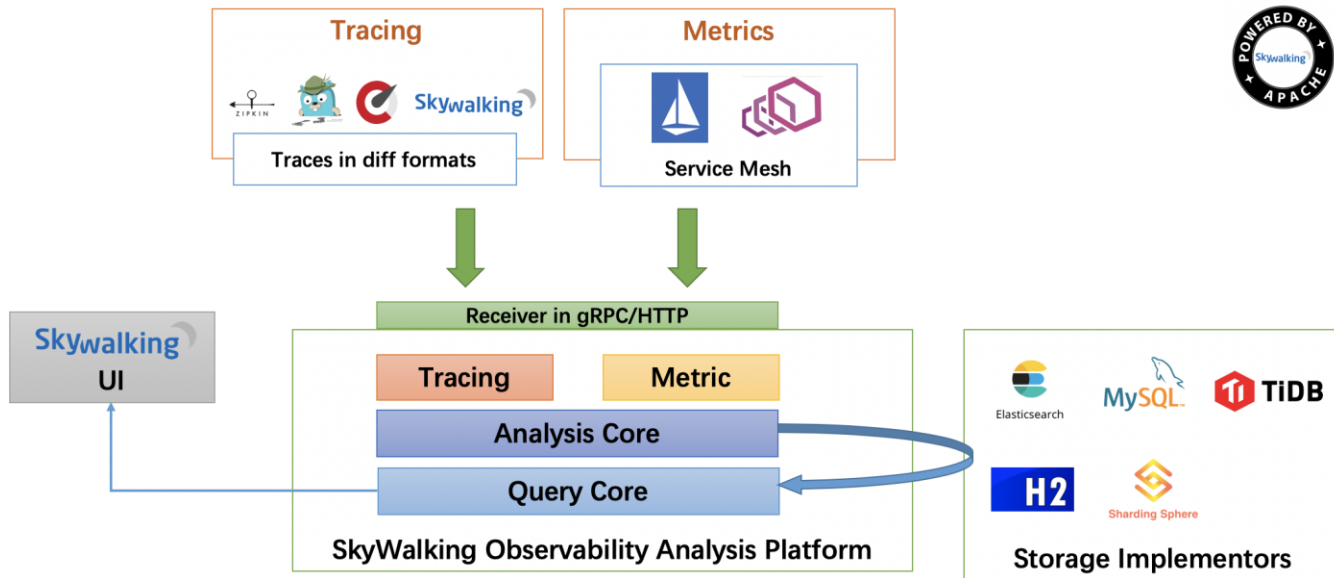
**支持多语言**，例如 Java, .Net Core, PHP, NodeJS, Golang, LUA

相对于老牌的 Sleuth+Zipkin, SkyWalking **功能增强**，**性能更优**，而且是**国产**开源，被**阿里推荐**。

- SkyWalking 结构分析
- SkyWalking 服务端环境搭建

# 1. SkyWalking 结构分析

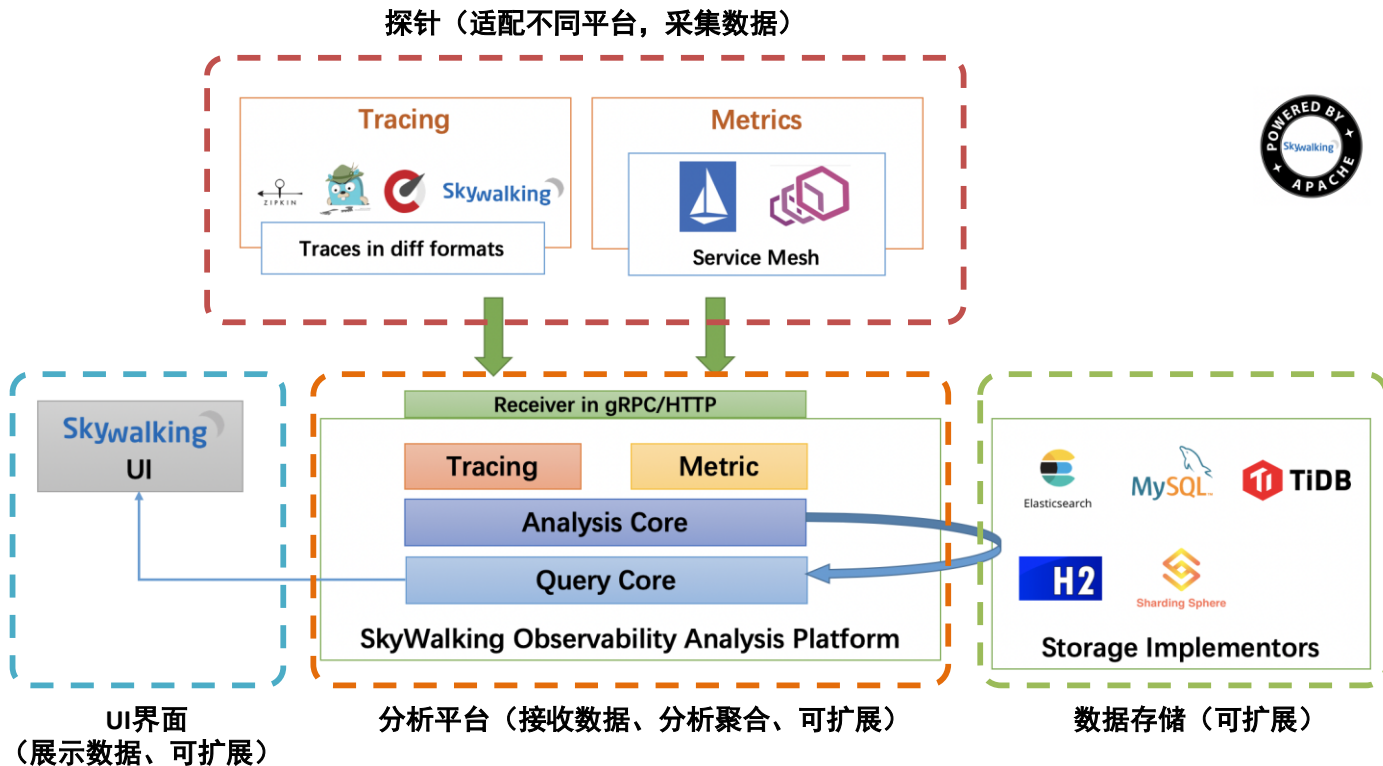
## 构成





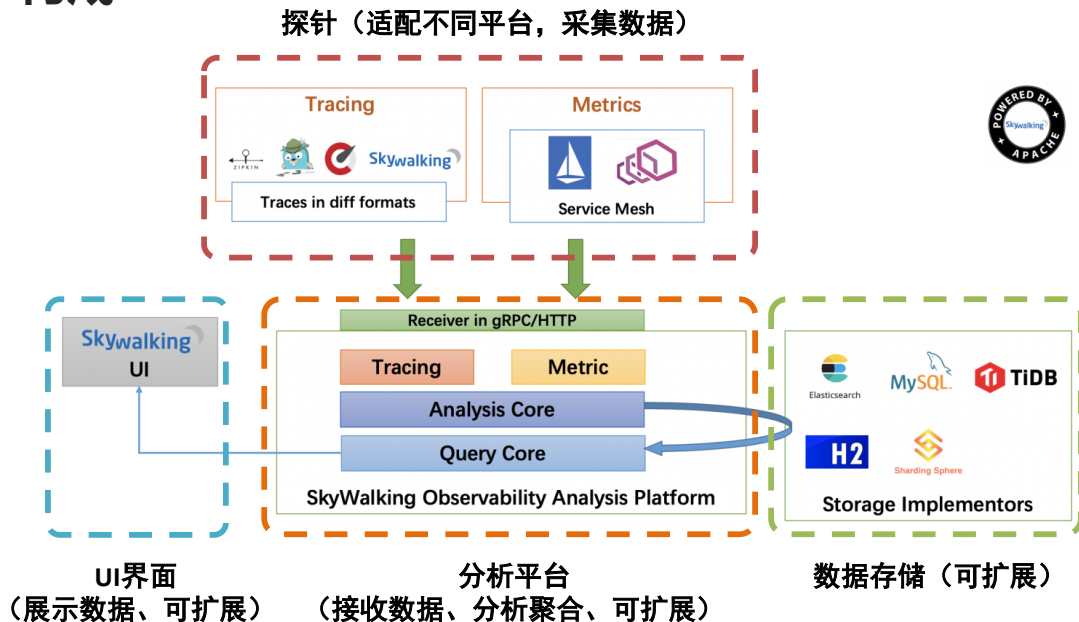
# 1. SkyWalking 结构分析

## 构成



# 1. SkyWalking 结构分析

## 构成



## 强大

- 一站式全流程
- 多语言自动探针
- 灵活扩展

## 易用

- 只需3步（启动平台、配置存储、添加探针）

## ■ 2. SkyWalking 服务端环境搭建

### 步骤

#### 1. 安装 Elasticsearch

#### 2. 下载解压 SkyWalking

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

```
> tar zxf apache-skywalking-apm-6.5.0.tar.gz
```

Nov. 15th, 2019	6.5.0	Source code	<a href="#">[src]</a> <a href="#">[asc]</a> <a href="#">[sha512]</a>
		Binary Distribution (Windows)	<a href="#">[zip]</a> <a href="#">[asc]</a> <a href="#">[sha512]</a>
		Binary Distribution (Linux)	<a href="#">[tar]</a> <a href="#">[asc]</a> <a href="#">[sha512]</a>
		Documentation	<a href="#">Documentation</a>

## 2. SkyWalking 服务端环境搭建

### 步骤

#### 3. 配置

config/application.yml

storage.h2 默认是打开的，需要注释掉

storage.Elasticsearch 默认是注释的，需要打开  
并把 clusterNodes 地址改为自己的 es 地址

webapp/webapp.yml

UI 默认端口 8080，会与自己的应用冲突

可以根据自己的端口安排进行相应修改。

```
storage:
  # elasticsearch:
  #   namespace: ${SW_NAMESPACE:""}
  #   clusterNodes: ${SW_STORAGE_ES_CLUSTER_NODES:localhost:9200}
  #   protocol: ${SW_STORAGE_ES_HTTP_PROTOCOL:"http"}
  #   trustStorePath: ${SW_SW_STORAGE_ES_SSL_JKS_PATH:"../es_keystore.jks"}
  #   trustStorePass: ${SW_SW_STORAGE_ES_SSL_JKS_PASS:""}
  #   user: ${SW_ES_USER:""}
  #   password: ${SW_ES_PASSWORD:""}
  #   indexShardsNumber: ${SW_STORAGE_ES_INDEX_SHARDS_NUMBER:2}
  #   indexReplicasNumber: ${SW_STORAGE_ES_INDEX_REPLICAS_NUMBER:0}
  #   # Those data TTL settings will override the same settings in core module.
  #   recordDataTTL: ${SW_STORAGE_ES_RECORD_DATA_TTL:7} # Unit is day
  #   otherMetricsDataTTL: ${SW_STORAGE_ES_OTHER_METRIC_DATA_TTL:45} # Unit is day
  #   monthMetricsDataTTL: ${SW_STORAGE_ES_MONTH_METRIC_DATA_TTL:18} # Unit is month
  #   # Batch process setting, refer to https://www.elastic.co/guide/en/elasticsearch
  #   bulkActions: ${SW_STORAGE_ES_BULK_ACTIONS:1000} # Execute the bulk every 1000
  #   flushInterval: ${SW_STORAGE_ES_FLUSH_INTERVAL:10} # flush the bulk every 10 s
  #   concurrentRequests: ${SW_STORAGE_ES_CONCURRENT_REQUESTS:2} # the number of co
  #   resultWindowMaxSize: ${SW_STORAGE_ES_QUERY_MAX_WINDOW_SIZE:10000}
  #   metadataQueryMaxSize: ${SW_STORAGE_ES_QUERY_MAX_SIZE:5000}
  #   segmentQueryMaxSize: ${SW_STORAGE_ES_QUERY_SEGMENT_SIZE:200}

  h2:
    driver: ${SW_STORAGE_H2_DRIVER:org.h2.jdbcx.JdbcDataSource}
    url: ${SW_STORAGE_H2_URL:jdbc:h2:mem:skywalking-oap-db}
    user: ${SW_STORAGE_H2_USER:sa}
    metadataQueryMaxSize: ${SW_STORAGE_H2_QUERY_MAX_SIZE:5000}
```

## 2. SkyWalking 服务端环境搭建

### 步骤

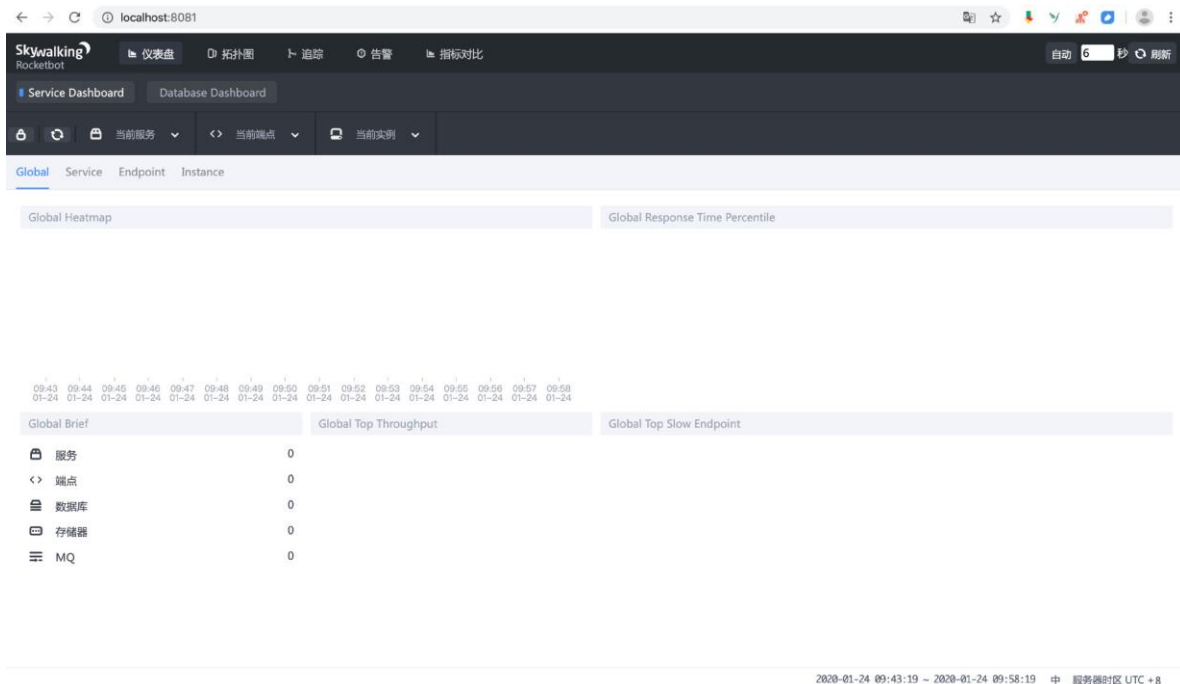
#### 4. 启动 SkyWalking

```
> bin/startup.sh
```

访问: <http://localhost:9001/>

启动日志位置:

logs/webapp.log





## 总结

### 重难点

1. SkyWalking 的结构（探针、分析平台、存储、UI展示）
2. SkyWalking 服务端环境搭建流程（改存储配置）

## 总结

### 重难点

1. SkyWalking 的结构（探针、分析平台、存储、UI展示）
2. SkyWalking 服务端环境搭建流程（改存储配置）

### 下节

服务整合 SkyWalking



# 目录 Contents

- ◆ 调用链跟踪原理
- ◆ 服务整合 Sleuth
- ◆ Sleuth 对 Feign, RestTemplate, 多线程 的支持
- ◆ Sleuth tracefilter 用法
- ◆ Sleuth 整合 Zipkin
- ◆ Zipkin 数据持久化
- ◆ SkyWalking 服务端配置
- ◆ SkyWalking 客户端配置



## 小节导学

SkyWalking 需要各个服务集成 Agent，用来自动采集数据。

Agent 如何与服务集成？

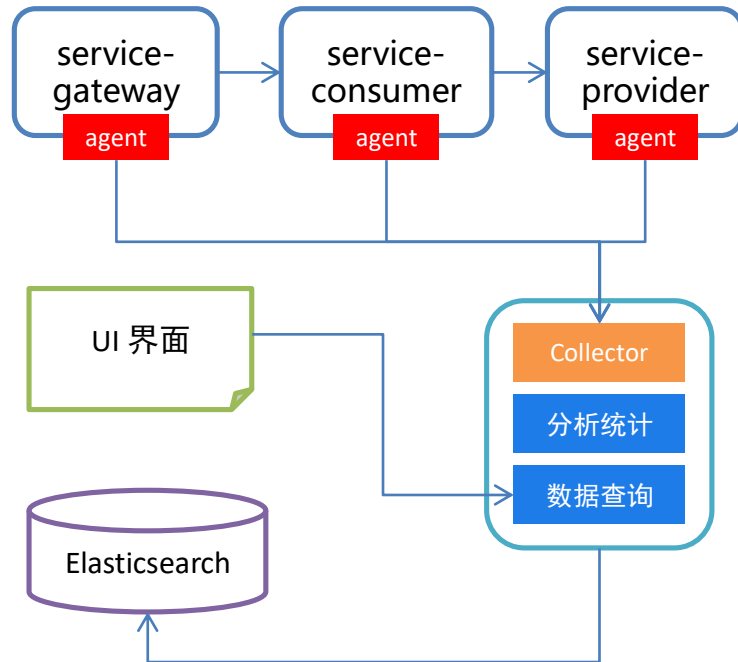
Agent 如何对接 Collector？

本节我们通过实践来解决这些问题。

创建3个测试服务：

1. service-provider
2. service-consumer
3. service-gateway

每个服务都集成 Agent，对接 Collector，最后体验效果。



## 服务集成 Agent

启动项目 jar 包时添加3个参数:

```
# 指定 skywalking-agent.jar 所在位置
-javaagent:apache-skywalking-apm-bin/agent/skywalking-agent.jar
# 对应当前服务的名称
-Dskywalking.agent.service_name=service-provider
# 对接 SkyWalking 服务端地址
-Dskywalking.collector.backend_service=localhost:11800
```

## 步骤

### 启动服务

#### # service-provider

```
$ java -javaagent:apache-skywalking-apm-bin/agent/skywalking-agent.jar  
-Dskywalking.agent.service_name=service-provider  
-Dskywalking.collector.backend_service=localhost:11800 -jar target/service-provider.jar
```

#### # service-consumer

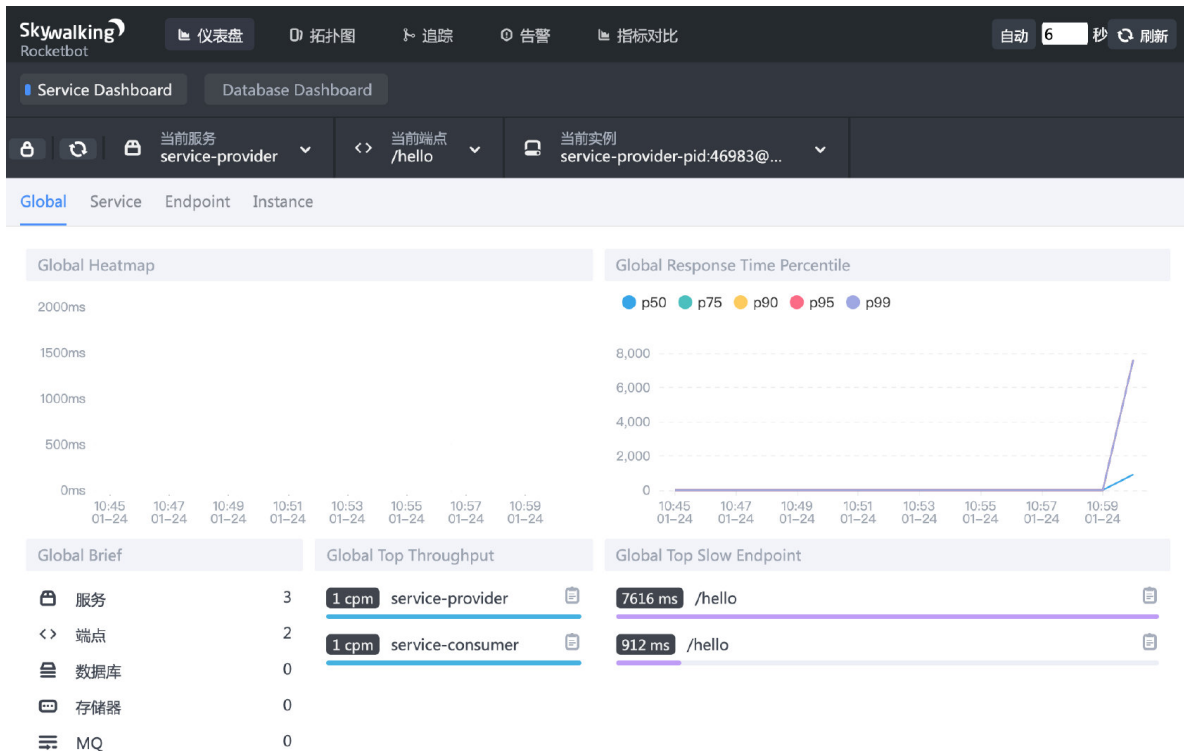
```
$ java -javaagent:apache-skywalking-apm-bin/agent/skywalking-agent.jar  
-Dskywalking.agent.service_name=service-consumer  
-Dskywalking.collector.backend_service=localhost:11800 -jar target/service-consumer.jar
```

#### # gateway-demo

```
$ java -javaagent:-javaagent:apache-skywalking-apm-bin/agent/skywalking-agent.jar  
-Dskywalking.agent.service_name=gateway-demo  
-Dskywalking.collector.backend_service=localhost:11800 -jar target/service-gateway.jar
```

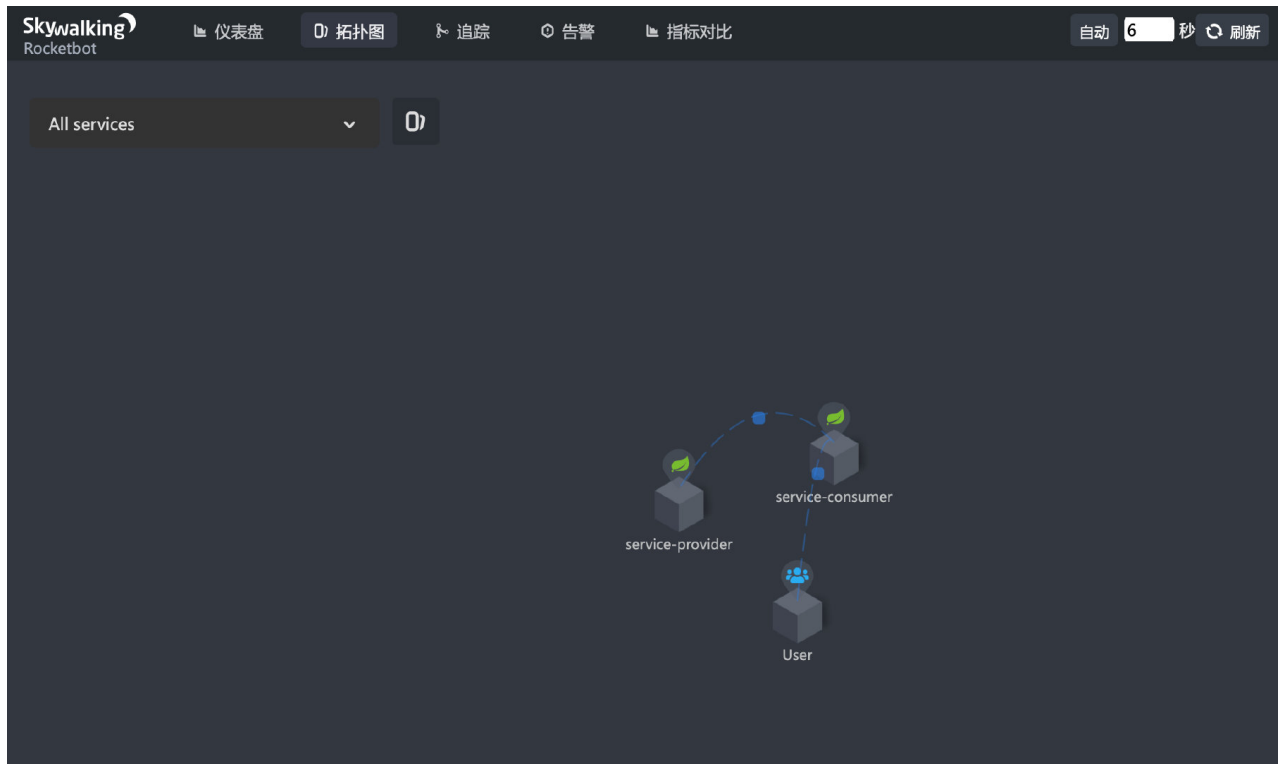
# SkyWalking 客户端配置

## 效果

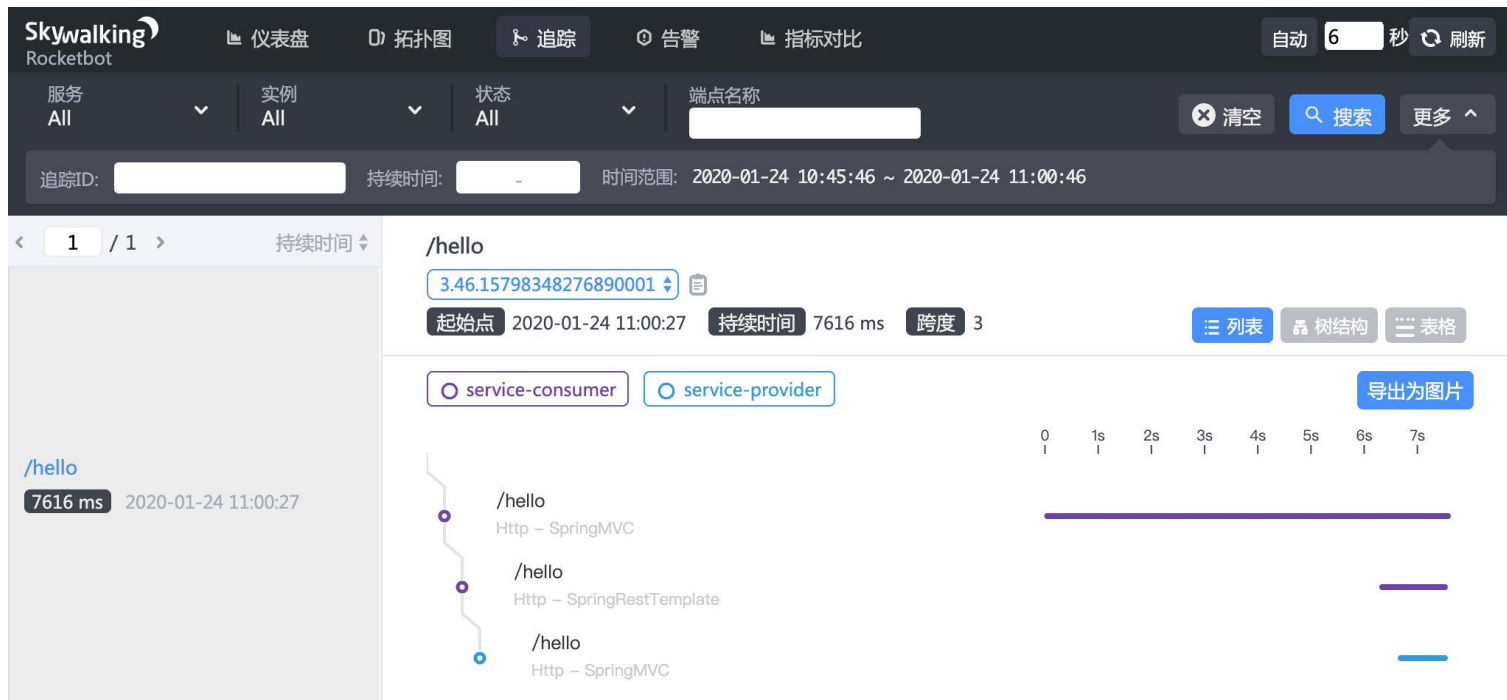


# SkyWalking 客户端配置

## 效果



## 效果





## 总结

### 重难点

#### 1. 服务集成 SkyWalking 的流程

(指定 agent jar 的位置、服务名称、SkyWalking 后端地址)

## 总结

### 重难点

#### 1. 服务集成 SkyWalking 的流程

(指定 agent jar 的位置、服务名称、SkyWalking 后端地址)

### 下节

本章内容总结





一样的在线教育，不一样的教学品质