4、项目实施

4.1 基本思路

从访问源头开始，按链路逐个写入日志，使用不同的手段，实现用户请求的各个点的日志收集

4.2 前端请求

4.2.1 概述

目前项目多采用动静分离方式，静态页由nginx处理。那么nginx上的请求日志如何收集处理呢?

第一可以采用输出到log文件，filebeat采集，送入kafka。第二可以采用lua脚本方式，直接输出到kafka，本章节采用第二种方式，完成静态文件部分的请求日志追踪。

4.2.2 openresty

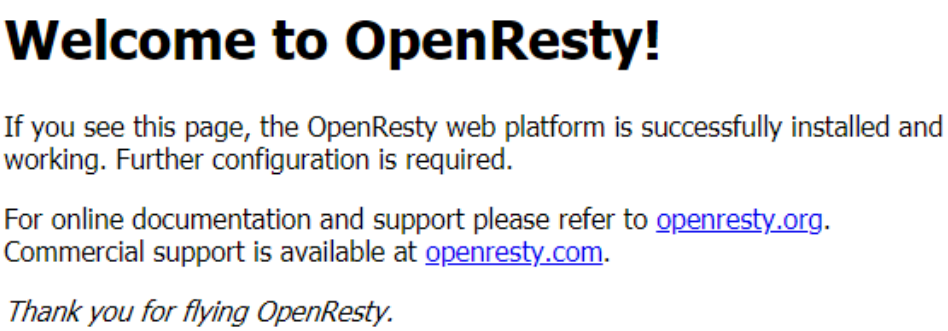
1）官网

http://openresty.org/cn/

2）下载地址

https://openresty.org/download/openresty-1.15.8.2-win64.zip

3）解压，启动与测试



4.2.3 lua-kafka

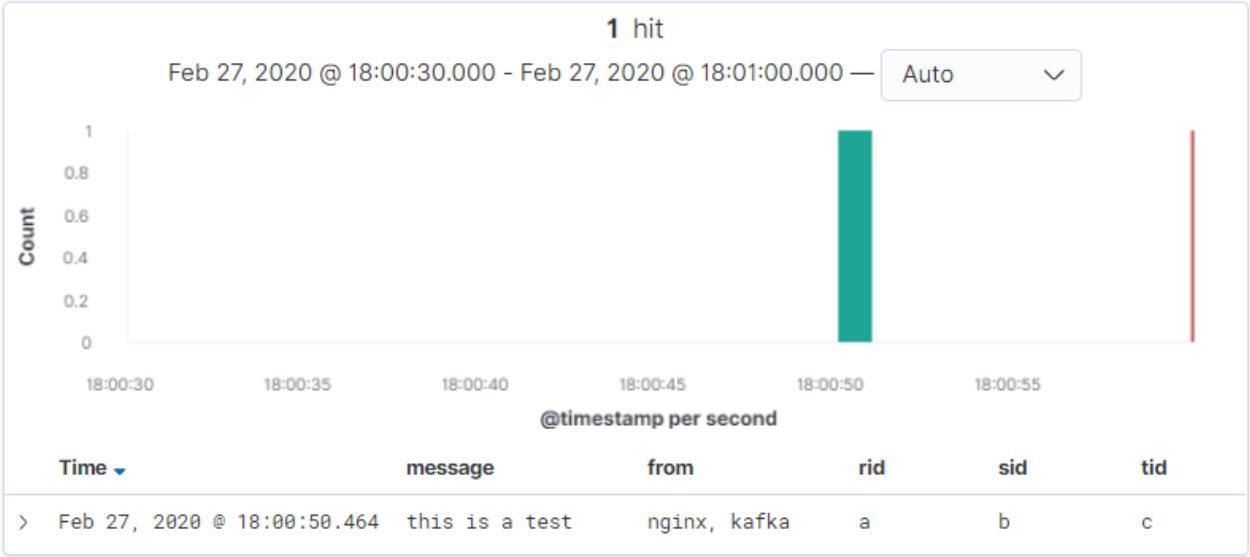
1）下载：https://github.com/doujiang24/lua-resty-kafka/archive/master.zip

2）解压，将resty目录解压到openresty的安装目录/lualib下

3）修改配置nginx配置文件

```
    location / {
        root   html;
        index  index.html index.htm;
        log_by_lua '

            local cjson = require "cjson"
            local producer = require "resty.kafka.producer"

            local broker_list = {
                { host = "39.98.133.153", port = 9103 },
            }

            local log_json = {}
            log_json["message"]="this is a test"
            log_json["from"]="nginx"
            log_json["rid"]="a"
            log_json["sid"]="b"
            log_json["tid"]="c"

            local message = cjson.encode(log_json);

            local bp = producer:new(broker_list, { producer_type = "async" })
            -- 发送日志消息,send第二个参数key,用于kafka路由控制:
            -- key为nill(空)时，一段时间向同一partition写入数据
            -- 指定key，按照key的hash写入到对应的partition
            local ok, err = bp:send("demo", nil, message)

            if not ok then
                ngx.log(ngx.ERR, "kafka send err:", err)
                return
            end
        ';
    }
```

4）测试log



Q&A：

问：为什么会有，nginx，kafka两个from?

答：logstash定义的field追加from导致，查看logstash配置及日志debug信息：

```
  kafka {
    bootstrap_servers => ["39.98.133.153:9103"]
    group_id => "logstash"
    topics => ["demo"]
    consumer_threads => 1
    decorate_events => true
    add_field => {"from" => "kafka"}
    codec => "json"
  }
```

```
{
      "@version" => "1",
           "rid" => "a",
       "message" => "this is a test",
          "from" => [
        [0] "nginx",
        [1] "kafka"
    ],
    "@timestamp" => 2020-02-27T10:00:50.464Z,
           "tid" => "c",
           "sid" => "b"
}
```

4.2.4 rid的生成

1）rid生成策略：前端的nginx生成请求rid，向下传递到整个请求环节，一直到请求结束。

2）生成方式：使用lua生成随机字符串即可

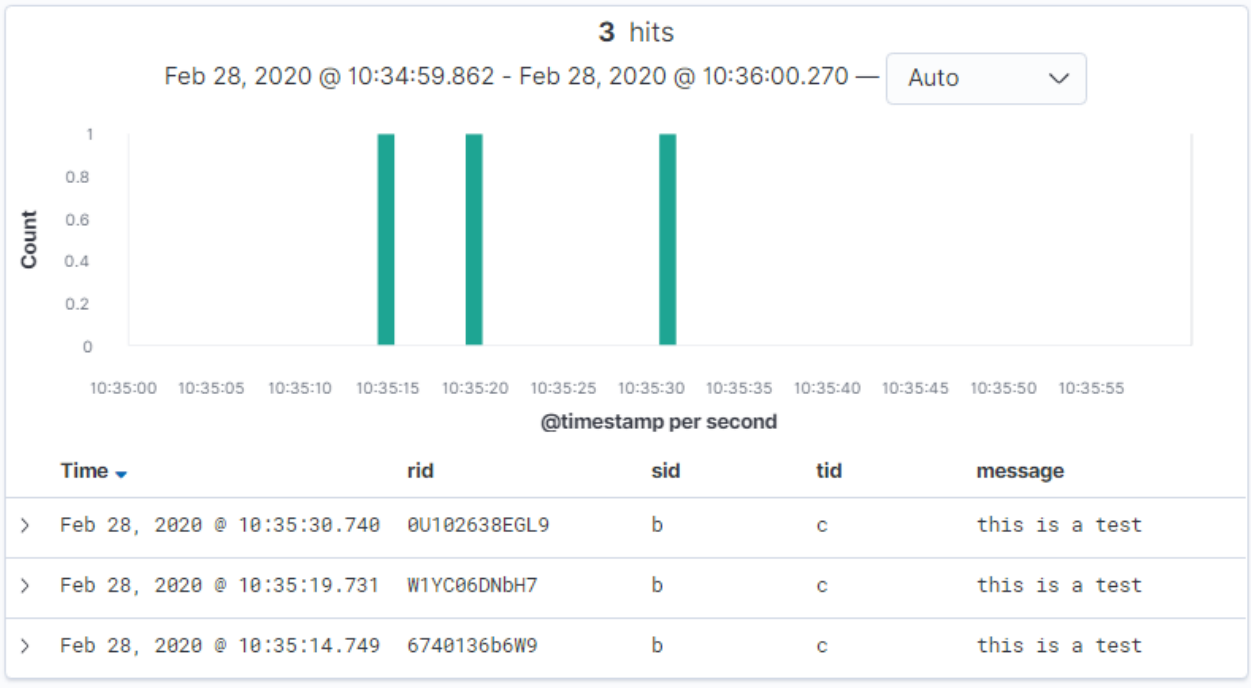
3）定义一个lua函数，lualib/resty/kafka/tools.lua

```
tools={}

function tools.getRandomStr(len)
  local rankStr = ""
  local randNum = 0
  for i=0,len do
    if math.random(1,3)==1 then
      randNum=string.char(math.random(0,26)+65)
    elseif math.random(1,3)==2 then
      randNum=string.char(math.random(0.26)+97)
    else
      randNum=math.random(0,10)
    end
    rankStr=rankStr..randNum
  end
  return rankStr
end

return tools
```

4）修改nginx.conf，随机长度可以根据实际业务调整，这里使用10

```
local kafkatools = require "resty.kafka.tools"

log_json["rid"]=kafkatools.getRandomStr(10)
```

5）重启ningx -s reload , 请求80端口，刷新几次，查看kibana是不是生成了不同的rid



4.2.5 tid的生成

1）tid，也就是终端的id，可以区分不同设备。常见于手机端和pc端多终端的场景分析中

2）策略：获取head头里的user-agent即可，浏览器debug信息如下：

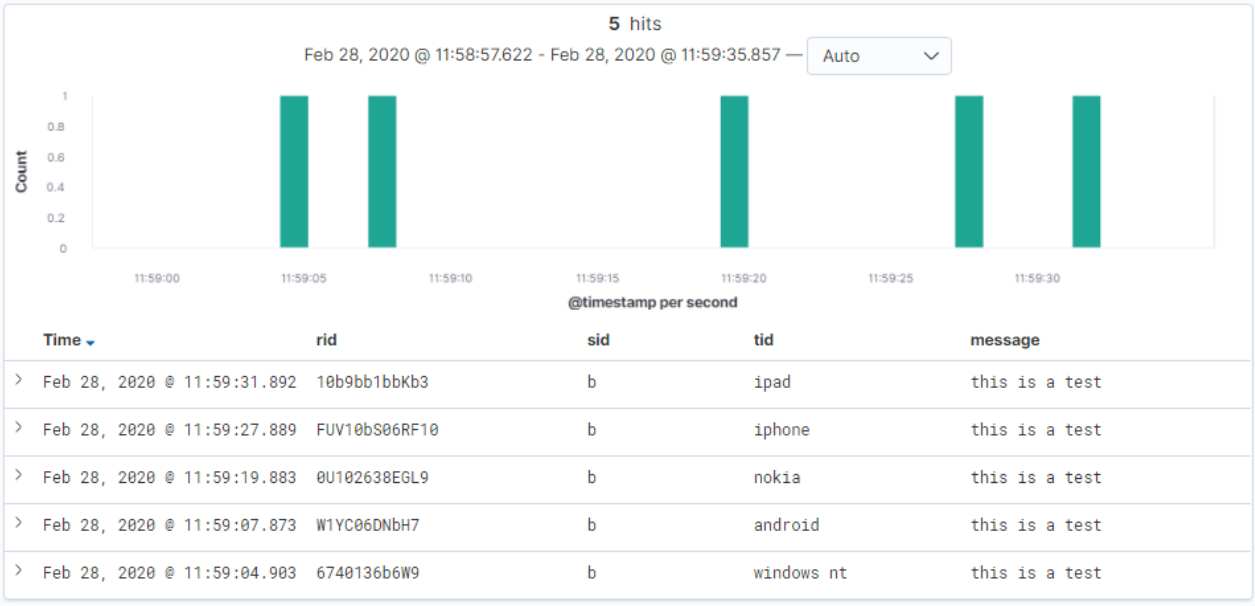
3）lua函数

```
function tools.getDevice()
  local headers=ngx.req.get_headers()
  local userAgent=headers["User-Agent"]
    local mobile = {
        "iphone", "android", "touch", "ipad", "symbian", "htc", "palmos",
"blackberry", "opera mini", "windows ce", "nokia", "fennec","macintosh",
        "hiptop", "kindle", "mot", "webos", "samsung", "sonyericsson", "wap",
"avantgo", "eudoraweb", "minimo", "netfront", "teleca","windows nt"
    }
    userAgent = string.lower(userAgent)

    for i, v in ipairs(mobile) do
        if string.match(userAgent, v) then
            return v
        end
    end
    return userAgent
end
```

4）修改tid，调用函数

```
log_json["tid"]=kafkatools.getDevice()
```

5）通过chrome调试模式进行验证



Time	rid	sid	tid	message
Feb 28, 2020 @ 11:59:31.892	10b9bb1bbKb3	b	ipad	this is a test
Feb 28, 2020 @ 11:59:27.889	FUV10bS06RF10	b	iphone	this is a test
Feb 28, 2020 @ 11:59:19.883	0U102638EGL9	b	nokia	this is a test
Feb 28, 2020 @ 11:59:07.873	W1YC06DNbH7	b	android	this is a test
Feb 28, 2020 @ 11:59:04.903	6740136b6W9	b	windows nt	this is a test

4.2.6 ip与url

1）使用lua脚本获取真实的ip，传递到下游。因为代理服务器作为请求的第一道关卡，可以首先获取到客户端的ip，而url可以从req中获取

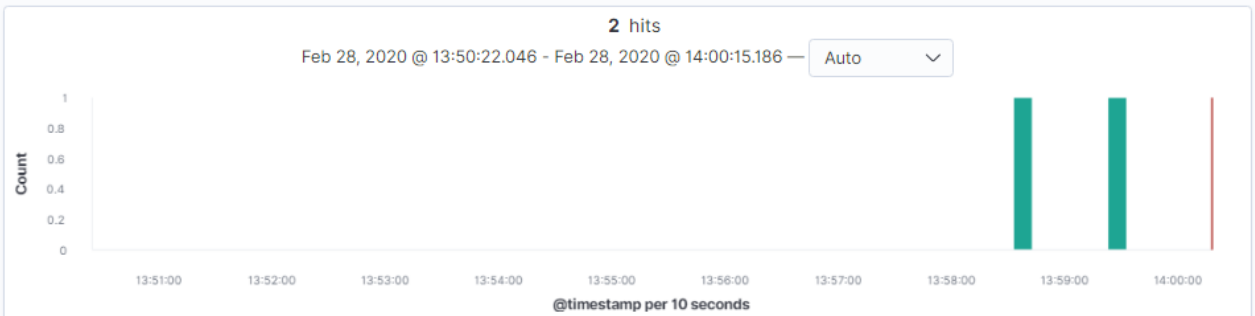
2）修改tools，添加函数

```
function tools.getClientIp()
    local headers=ngx.req.get_headers()
    local ip=headers["X-REAL-IP"] or headers["X_FORWARDED_FOR"] or
ngx.var.remote_addr or "0.0.0.0"
    return ip
end
```

3）修改nginx配置，加上ip信息

```
log_json["ip"]=kafkatools.getClientIp()
log_json["message"]="nginx:"..(ngx.var.uri)
```

4）kibana验证



Time	rid	sid	tid	ip	message
Feb 28, 2020 @ 13:59:27.958	0U102638EGL9	b	windows nt	127.0.0.1	nginx:/login.html
Feb 28, 2020 @ 13:58:32.928	W1YC06DNbH7	b	iphone	127.0.0.1	nginx:/index.html

还少一个sid即当前登录用户，这个不是nginx能解决的，我们下文再讲。

4.3 微服务层

4.3.1 概述

后台服务同样需要生成上面的rid，tid，sid和基本的ip与url，这些在java端被封装在了HttpServletRequest中。通过request对象，获取相应的信息后，借助上一章的框架集成kafka，可以将访问信息送入日志平台。

4.3.2 代理转发

1）修改nignx配置文件，将静态文件剥离，api发往后台gateway，注意headers的设置和下放

```
#添加location，将 /api 请求转发给后台
location ^~ /api/ {
  proxy_pass http://127.0.0.1:8002;
}
```

2）在web中新增一个ApiController， 新增 /test 请求

```
@RequestMapping("/api")
public class ApiController {
    @GetMapping("/test")
    public Object test(){
        return "this is a test";
    }
}
```

3）测试nginx入口是否正常转发到后台

4.3.3 filter第一关

请求进入后台后，第一道关卡可以通过filter或者interceptor拿到相关信息，记录其调用日志。

1）在utils项目下增加filter过滤器

```
package com.itheima.logdemo.utils;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
```

```
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;

@Configuration
@Order(1)
@WebFilter(filterName = "logFilter", urlPatterns = "/*")
public class LogFilter implements Filter {
    //知识点：被哪个app引用，当前from的日志记录就是当前app的名字
    @Value("${spring.application.name}")
    String appName;
    //知识点：slf4j的好处，utils被其他项目引用时不会给对方的日志产生干扰
    private Logger logger = LoggerFactory.getLogger("kafka");

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws IOException,
ServletException {
        logger.info(new LogBean("rid","sid","tid",appName,"I am
filter").toString());
        chain.doFilter(request, response);

    }

}
```

2）给web的启动类添加扫描，将filter扫入spring

```
@ComponentScan("com.itheima.logdemo")
```

3）启动web测试filter日志是否进入kibana

4.3.4 id的生成

上面日志可以正常进入采集通道，但是各个id采用的是字符串，怎么取得真实的场景数据呢？本节讨论id的生成

1）utils下定义一个CommonUtils，与前台一样，rid使用随机字符串，定义一个工具函数

```
public static String getRandomStr(int len){
    char[]
chars="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".toCharAr
ray();
    char[] str = new char[len];
    for (int i = 0; i < len; i++) {
        str[i] = chars[RandomUtils.nextInt(0,61)];
    }
```

```
    return new String(str);
}

//测试:
public static void main(String[] args) {
  for (int i = 0; i < 5; i++) {
    System.out.println(getRandomStr(10));
  }
}
```

2）tid，同样的方式，取header

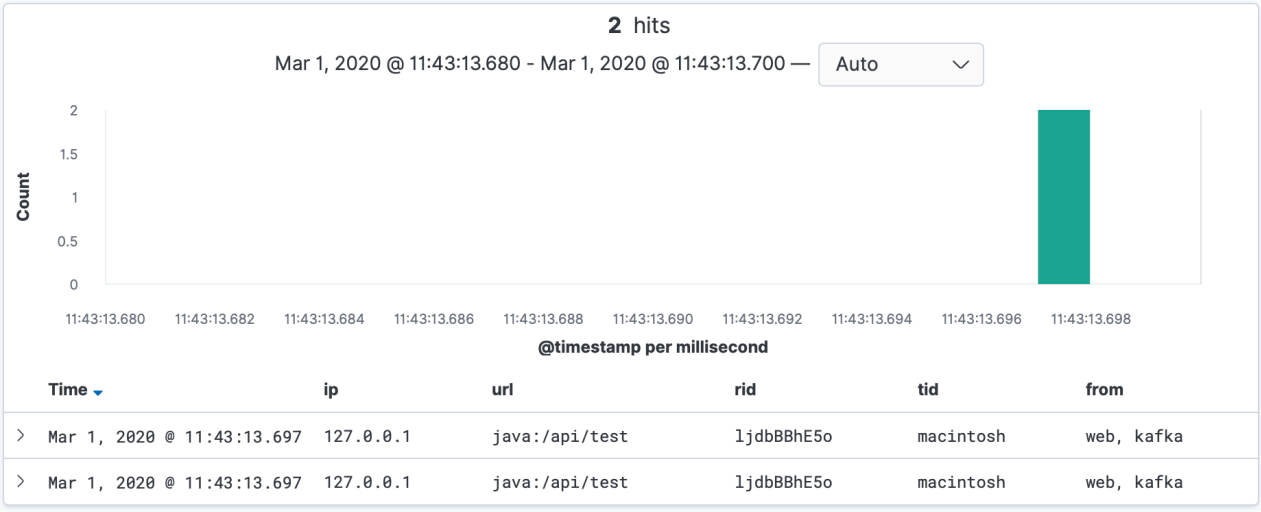
```
public static String getDevice(String userAgent){
    String[] terminals = {
            "iphone", "android", "touch", "ipad", "symbian", "htc", "palmos",
"blackberry", "opera mini", "windows ce", "nokia", "fennec","macintosh",
            "hiptop", "kindle", "mot", "webos", "samsung", "sonyericsson",
"wap", "avantgo", "eudoraweb", "minimo", "netfront", "teleca","windows nt"
    };
    userAgent = userAgent.toLowerCase();
    for (String terminal : terminals) {
        if (userAgent.indexOf(terminal) != -1){
            return terminal;
        }
    }
    return userAgent;

}
```

3）ip与url

```
//LogBean中新增ip和url属性，添加get，set方法

//CommonUtils添加获取ip的方法
   public static String getIpAddress(HttpServletRequest request) {
        String ip = request.getHeader("x-forwarded-for");
        if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
            ip = request.getHeader("Proxy-Client-IP");
        }
        if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
            ip = request.getHeader("WL-Proxy-Client-IP");
        }
        if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
            ip = request.getHeader("HTTP_CLIENT_IP");
        }
        if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
            ip = request.getHeader("HTTP_X_FORWARDED_FOR");
        }
```

```
        if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
            ip = request.getRemoteAddr();
        }
        return ip;

    }

//修改filter的构造，url可以从request取到

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws IOException,
ServletException {
        HttpServletRequest httpServletRequest = (HttpServletRequest) request;
        LogBean logBean = new LogBean(CommonUtils.getRandomStr(10), "sid",
                CommonUtils.getDevice(httpServletRequest.getHeader("User-
Agent")),
                appName, "I am filter");
        logBean.setIp(CommonUtils.getIpAddress(httpServletRequest));
        logBean.setUrl("java:" + httpServletRequest.getRequestURI());
        logger.info(logBean.toString());
        chain.doFilter(request, response);

    }
```

4）进入kibana测试采集情况



4.3.5 参数传递

以上步骤，filter中生成了相关的id信息，如何传递到下面的方法中呢？答案就是threadlocal

1）修改LogBean的构造函数，生成的时候，将自己放入当前threadlocal，filter不需要改动

```
public final static ThreadLocal<LogBean> logBeanThreadLocal = new ThreadLocal<>
();

private String rid,sid,tid,from,message;

public LogBean(String rid, String sid, String tid, String from, String message)
{
    this.rid = rid;
    this.sid = sid;
    this.tid = tid;
    this.from = from;
    this.message = message;
    logBeanThreadLocal.set(this);
}
```

2）新建ApiService

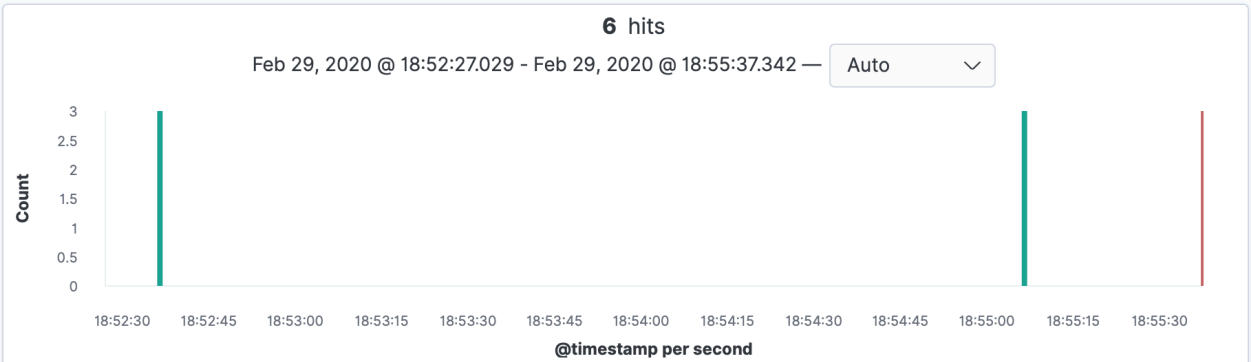
```
@Service
public class ApiService {
    private final static Logger logger = LoggerFactory.getLogger("kafka");

    public void test(){
        LogBean logBean = LogBean.logBeanThreadLocal.get();

        logBean.setMessage("I am service");

        logger.info(logBean.toString());

    }
}
```

3）修改ApiController

```
@RestController
@RequestMapping("/api")
public class ApiController {
    private final static Logger logger = LoggerFactory.getLogger("kafka");

    @Autowired ApiService apiService;

    @GetMapping("/test")
    public Object test(){

        LogBean logBean = LogBean.logBeanThreadLocal.get();

        logBean.setMessage("I am Controller");

        logger.info(logBean.toString());
```

```
        apiService.test();

        return "this is a test" ;
    }

}
```

4）请求/api/test，查看kibana确认日志采集情况，可见同一请求的rid和tid相同，成功传递。而两次不同请求的rid不同，终端相同，tid相同。



Time	sid	rid	tid	from	message
Feb 29, 2020 @ 18:55:06.797		P1SxL0PfwW	macintosh	web, kafka	I am service
Feb 29, 2020 @ 18:55:06.797		P1SxL0PfwW	macintosh	web, kafka	I am Controller
Feb 29, 2020 @ 18:55:06.796		P1SxL0PfwW	macintosh	web, kafka	I am filter
Feb 29, 2020 @ 18:52:36.505		zZsexHWe5E	macintosh	web, kafka	I am service
Feb 29, 2020 @ 18:52:36.505		zZsexHWe5E	macintosh	web, kafka	I am Controller
Feb 29, 2020 @ 18:52:36.505		zZsexHWe5E	macintosh	web, kafka	I am filter

4.3.6 切面

以上方式，每个方法调用都需要手写，那有没有办法将注意力放在业务上，而不必硬编码日志的打印呢?

1）utils引入相关坐标

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

2）定义一个注解用于方法标示

```java
package com.itheima.logdemo.utils;

import java.lang.annotation.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface LogInfo {
    String value() default "";
}
```

3）定义切面

```java
package com.itheima.logdemo.utils;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.aspectj.lang.reflect.MethodSignature;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

/**
 * 日志切面
 */
@Aspect
@Component
public class LogAspect {

    private final static Logger logger = LoggerFactory.getLogger("kafka");


    @Pointcut("@annotation(com.itheima.logdemo.utils.LogInfo)")
    public void log() {}


    /**
     * 环绕通知
     */
    @Around(value = "log()")
    public Object arround(ProceedingJoinPoint pjp) {
        try {
            MethodSignature signature = (MethodSignature) pjp.getSignature();
            String className = pjp.getTarget().getClass().getSimpleName();
            String methodName = signature.getName();
```

```
            LogBean logBean = LogBean.logBeanThreadLocal.get();
            logBean.setMessage("before "+className+"."+methodName);
            logger.info(logBean.toString());

            //方法执行
            Object o =  pjp.proceed();

            logBean.setMessage("after "+className+"."+methodName);
            logger.info(logBean.toString());

            return o;
        } catch (Throwable e) {
            e.printStackTrace();
            return null;
        }
    }



}
```

4）对service打注解，controller不要加，再次访问

```
@LogInfo
public void test(){
    LogBean logBean = LogBean.logBeanThreadLocal.get();

    logBean.setMessage("I am service");
    try {
        Thread.sleep(500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    logger.info(logBean.toString());

}
```

5）进kibana验证信息采集情况

Time	from	rid	tid	message
Feb 29, 2020 @ 20:00:53.693	web, kafka	CYGAqs64uv	macintosh	after ApiService.test
Feb 29, 2020 @ 20:00:53.693	web, kafka	CYGAqs64uv	macintosh	I am service
Feb 29, 2020 @ 20:00:53.370	web, kafka	CYGAqs64uv	macintosh	before ApiService.test
Feb 29, 2020 @ 20:00:53.370	web, kafka	CYGAqs64uv	macintosh	I am Controller
Feb 29, 2020 @ 20:00:53.370	web, kafka	CYGAqs64uv	macintosh	I am filter
Feb 29, 2020 @ 19:57:36.501	web, kafka	PiQkMUjUyG	macintosh	after ApiService.test
Feb 29, 2020 @ 19:57:36.492	web, kafka	PiQkMUjUyG	macintosh	I am service
Feb 29, 2020 @ 19:57:35.987	web, kafka	PiQkMUjUyG	macintosh	I am Controller
Feb 29, 2020 @ 19:57:35.987	web, kafka	PiQkMUjUyG	macintosh	before ApiService.test
Feb 29, 2020 @ 19:57:35.986	web, kafka	PiQkMUjUyG	macintosh	I am filter

4.3.7 乱序问题

注意上面的结果，信息全了，但是为什么是乱序的呢?

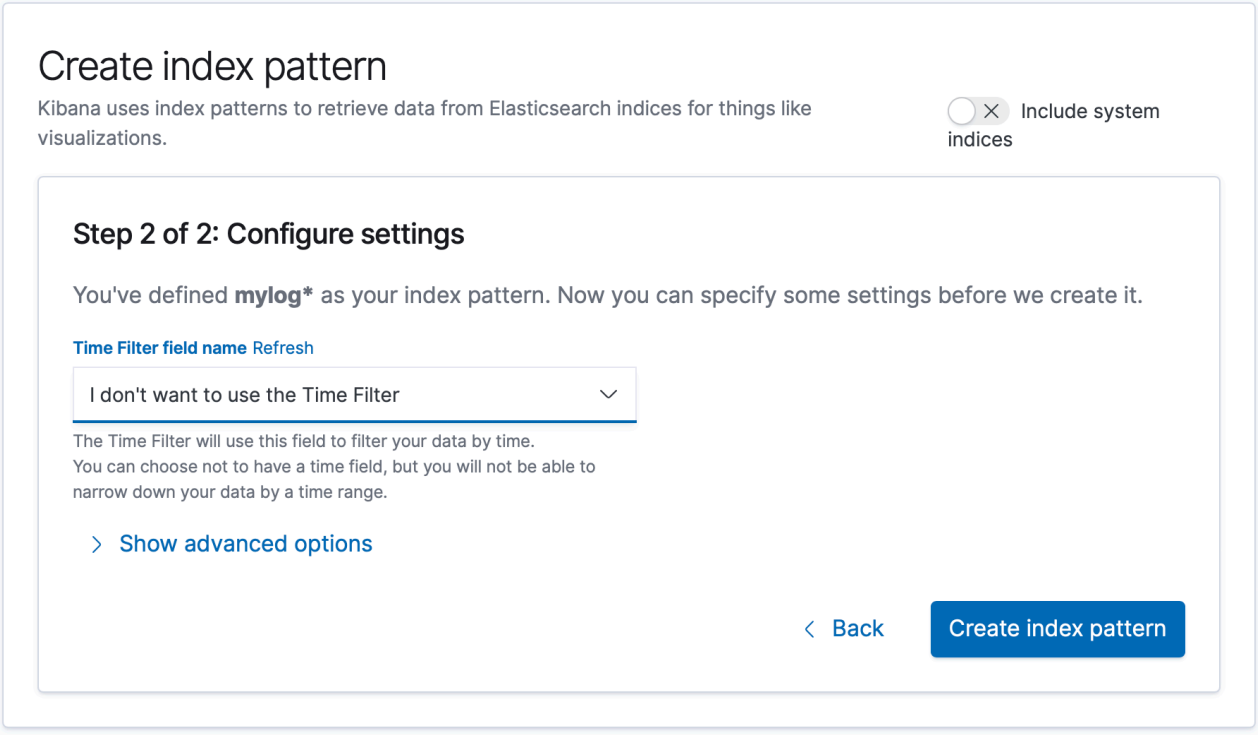
原因：来自timestamp的值是logstash取出kafka的时间，因为消息通过kafka异步传送后，这个时间不再能精确反映日志诞生的时间，也就无法保证顺序性，如何解决呢?

1）修改LogBean，再需要打印，也就是调toString时，将当前时间戳放进去

```
package com.itheima.logdemo.utils;

import com.alibaba.fastjson.JSON;

public class LogBean {
    public final static ThreadLocal<LogBean> logBeanThreadLocal = new
ThreadLocal<>();

    private String rid, sid, tid, from, message;
    private long time;

    public LogBean(String rid, String sid, String tid, String from, String
message) {
        this.rid = rid;
        this.sid = sid;
        this.tid = tid;
        this.from = from;
```

```
        this.message = message;
        logBeanThreadLocal.set(this);
    }

    //getters and setters ....

    public long getTime() {
        return time;
    }

    public void setTime(long time) {
        this.time = time;
    }

    @Override
    public String toString() {
        this.time = System.currentTimeMillis();
        return JSON.toJSONString(this);
    }
}
```

2）进入kibana重新创建一个索引，不再使用timestamp



3）kibana展示时，选time，并倒序

5 hits

rid	message	from	tid	time
CYGAqs64uv	after ApiService.test	web, kafka	macintosh	1,582,977,653,591
CYGAqs64uv	I am service	web, kafka	macintosh	1,582,977,653,590
CYGAqs64uv	before ApiService.test	web, kafka	macintosh	1,582,977,653,083
CYGAqs64uv	I am Controller	web, kafka	macintosh	1,582,977,653,077
CYGAqs64uv	I am filter	web, kafka	macintosh	1,582,977,652,266

可以看到，message严格按照时间顺序打印

4.4 跨服务调用

问题：threadlocal解决了log信息在本应用内的传递，但是项目中存在跨服务的调用，例如web调user微服务，那在web模块中生成的相关id如何传递给user呢？本章我们讨论这个问题

4.4.1 默认RestTemplate

1）修改web模块的App，添加restTemplate

```
@LoadBalanced
@Bean
RestTemplate getRestTemplate() {
    return new RestTemplate();
}
```

2）修改service方法改为远程调用

```
package com.itheima.logdemo.web;

import com.itheima.logdemo.utils.LogBean;
import com.itheima.logdemo.utils.LogInfo;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;


@Service
public class ApiService {
    private final static Logger logger = LoggerFactory.getLogger("kafka");

    @Autowired
    RestTemplate restTemplate;

    @LogInfo
```

```
    public void test(){
        LogBean logBean = LogBean.logBeanThreadLocal.get();

        logBean.setMessage("I am service");
        logger.info(logBean.toString());

        logBean.setMessage("before call user");
        logger.info(logBean.toString());

        String res =
restTemplate.getForObject("http://user/info",String.class);
        logBean.setMessage("after call user, res="+res);
        logger.info(logBean.toString());

    }
}
```

3）user模块新增UserController，加一个info方法

```
package com.itheima.logdemo.user;

import com.itheima.logdemo.utils.LogBean;
import com.itheima.logdemo.utils.LogInfo;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    private static final Logger logger = LoggerFactory.getLogger("kafka");

    @LogInfo
    @RequestMapping("/info")
    public String info(){
        LogBean logBean = LogBean.logBeanThreadLocal.get();
        logBean.setMessage("I am user.controller");
        logger.info(logBean.toString());
        return "zhangsan";
    }
}
```

4）不要忘记user模块上App的扫包路径

```
@ComponentScan("com.itheima.logdemo")
```

5）访问web的api/test请求，查看kibana验证

98 hits

time	rid	tid	ip	from	message
1583144341361	L6MUHsp2AS	macintosh	0:0:0:0:0:0:0:1	web, kafka	after ApiService.test
1583144341361	L6MUHsp2AS	macintosh	0:0:0:0:0:0:0:1	web, kafka	after call user, res=zhangsan
1583144341331	ikBZwA6Y5B	java/1.8.0_181	192.168.0.105	user, kafka	after UserController.info
1583144341330	ikBZwA6Y5B	java/1.8.0_181	192.168.0.105	user, kafka	I am user.controller
1583144341309	ikBZwA6Y5B	java/1.8.0_181	192.168.0.105	user, kafka	before UserController.info
1583144341053	ikBZwA6Y5B	java/1.8.0_181	192.168.0.105	user, kafka	I am filter
1583144335537	L6MUHsp2AS	macintosh	0:0:0:0:0:0:0:1	web, kafka	before call user
1583144335537	L6MUHsp2AS	macintosh	0:0:0:0:0:0:0:1	web, kafka	I am service
1583144335532	L6MUHsp2AS	macintosh	0:0:0:0:0:0:0:1	web, kafka	before ApiService.test
1583144335525	L6MUHsp2AS	macintosh	0:0:0:0:0:0:0:1	web, kafka	I am Controller
1583144335274	L6MUHsp2AS	macintosh	0:0:0:0:0:0:0:1	web, kafka	I am filter

注意：user的rid和tid，以及ip来源，是由filter重新生成的。我们希望的值应该是与web同一个才对，这就涉及到远程服务调用时，链路信息的传递问题。

4.4.2 自定义RestTemplate

问题：上一个微服务生成的链路，到下一个微服务调用时，中断了。怎么追踪呢?

1）自定义MyRestTemplate，继承默认的RestTemplate，并覆盖execute方法

```
@Override
protected <T> T doExecute(URI url, HttpMethod method, RequestCallback
requestCallback, ResponseExtractor<T> responseExtractor) throws
RestClientException {
    Assert.notNull(url, "URI is required");
    Assert.notNull(method, "HttpMethod is required");
    ClientHttpResponse response = null;

    Object var14;
    try {
        ClientHttpRequest request = this.createRequest(url, method);
        if (requestCallback != null) {
            requestCallback.doWithRequest(request);
        }

        //重点在这里!
        LogBean logBean = LogBean.logBeanThreadLocal.get();
```

```
        HttpHeaders httpHeaders = request.getHeaders();
        httpHeaders.add("rid",logBean.getRid());
        httpHeaders.add("sid",logBean.getSid());
        httpHeaders.add("tid",logBean.getTid());
        httpHeaders.add("ip",logBean.getIp());

        response = request.execute();
        this.handleResponse(url, method, response);
        var14 = responseExtractor != null ?
responseExtractor.extractData(response) : null;
    } catch (IOException var12) {
        String resource = url.toString();
        String query = url.getRawQuery();
        resource = query != null ? resource.substring(0, resource.indexOf(63))
: resource;
        throw new ResourceAccessException("I/O error on " + method.name() + "
request for \"" + resource + "\": " + var12.getMessage(), var12);
    } finally {
        if (response != null) {
            response.close();
        }

    }

    return (T) var14;
}
```

2）修改web的App启动类，获取自定义的RestTemplate

```
    @LoadBalanced
    @Bean
    RestTemplate getRestTemplate() {
//        return new RestTemplate();
        return new MyRestTemplate();
    }
```

3）改造Filter，优先取header的值

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
                     FilterChain chain) throws IOException, ServletException {
    HttpServletRequest httpServletRequest = (HttpServletRequest) request;

    String rid =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("rid"),CommonUtils.getR
andomStr(10));
```

```
        String sid =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("sid"),CommonUtils.getR
andomStr(10));
        String tid =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("tid"),CommonUtils.getD
evice(httpServletRequest.getHeader("User-Agent")));
        String ip =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("ip"),CommonUtils.getIp
Address(httpServletRequest));

        LogBean logBean = new LogBean(rid,sid,tid,appName, "I am filter");
        logBean.setIp(ip);
        logBean.setUrl("java:" + httpServletRequest.getRequestURI());

        logger.info(logBean.toString());
        chain.doFilter(request, response);

    }
```

4）再次请求进kibana验证链路传递情况

120 hits

time	rid	tid	ip	from	message	url
1583146443395	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	web, kafka	after ApiService.test	java:/api/test
1583146443394	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	web, kafka	after call user, res=zhangsan	java:/api/test
1583146443351	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	user, kafka	after UserController.info	java:/info
1583146443350	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	user, kafka	I am user.controller	java:/info
1583146443342	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	user, kafka	before UserController.info	java:/info
1583146443034	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	user, kafka	I am filter	java:/info
1583146437337	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	web, kafka	I am service	java:/api/test
1583146437337	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	web, kafka	before call user	java:/api/test
1583146437330	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	web, kafka	before ApiService.test	java:/api/test
1583146437325	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	web, kafka	I am Controller	java:/api/test
1583146436991	MrnoqgwOig	macintosh	0:0:0:0:0:0:0:1	web, kafka	I am filter	java:/api/test

结论：rid等链路信息得以传递

4.5 sid的生成

截止到目前为止，整个链路已基本完成，但是不要忘记，还有一种特殊场景：用户登陆，这就涉及到我们的sid

4.5.1 登陆业务

1）修改user模块，新增一个用户名密码校验微服务，模拟登陆校验

```
@LogInfo
@RequestMapping("/check")
public boolean check(@RequestParam("username") String username ,
@RequestParam("password") String password){

    //check username and password

    LogBean logBean = LogBean.logBeanThreadLocal.get();
    logBean.setMessage("correct user");
    logger.info(logBean.toString());

    return true;
}
```

2）在web中新增login请求，模拟登陆操作，由web调user微服务

```
@LogInfo
@RequestMapping("/login")
public String login(HttpServletRequest request, HttpServletResponse response){

    String username = request.getParameter("username");
    String password = request.getParameter("password");

    boolean ok = apiService.login(username,password);

    if (ok){
        Cookie cookie = new Cookie("sid",username);
        //设置path，让所有请求均可以获取
        cookie.setPath("/");
        response.addCookie(cookie);
        return "success";
    }else {
        return "error";
    }

}
```

```
@LogInfo
public boolean login(String username, String password) {
    return restTemplate.getForObject("http://user/check?
username="+username+"&password="+password,boolean.class);
}
```

4.5.2 后台sid生成

1）修改filter，sid从cookie获取

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
                     FilterChain chain) throws IOException, ServletException {
    HttpServletRequest httpServletRequest = (HttpServletRequest) request;

    String cookieVal = null;
    Cookie[] cookies = httpServletRequest.getCookies();
    if (cookies != null){
        for (Cookie cookie : cookies) {
            if ("sid".equals(cookie.getName())){
                cookieVal = cookie.getValue();
                break;
            }
        }
    }

    String rid =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("rid"),CommonUtils.getR
andomStr(10));
    String sid =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("sid"),cookieVal);
    String tid =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("tid"),CommonUtils.getD
evice(httpServletRequest.getHeader("User-Agent")));
    String ip =
StringUtils.defaultIfBlank(httpServletRequest.getHeader("ip"),CommonUtils.getIp
Address(httpServletRequest));

    LogBean logBean = new LogBean(rid,sid,tid,appName, "I am filter");
    logBean.setIp(ip);
    logBean.setUrl("java:" + httpServletRequest.getRequestURI());

    logger.info(logBean.toString());
    chain.doFilter(request, response);

}
```

2）调用登陆接口： http://localhost/api/login?username=zhangsan&password=123

3）登陆成功后，再次访问：http://localhost/api/test

2）验证kibana里的sid

189 hits

time	rid	sid	tid	ip	from	url
1583203456789	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	web, kafka	java:/api/test
1583203456789	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	web, kafka	java:/api/test
1583203456787	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	user, kafka	java:/info
1583203456787	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	user, kafka	java:/info
1583203456786	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	user, kafka	java:/info
1583203456786	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	user, kafka	java:/info
1583203456783	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	web, kafka	java:/api/test
1583203456783	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	web, kafka	java:/api/test
1583203456782	HGuNkOQj8n	zhangsan	macintosh	0:0:0:0:0:0:0:1	web, kafka	java:/api/test

4.5.3 前端nginx的sid

通过上节的cookie设置和filter获取，后端的链路中sid可以正常生成，那么前端nginx和lua部分如何生成呢?

1）修改lua脚本

```
        log_by_lua '
            -- 引入lua所有api
            local cjson = require "cjson"
            local producer = require "resty.kafka.producer"
            local kafkatools = require "resty.kafka.tools"
            -- 定义kafka broker地址，ip需要和kafka的host.name配置一致
            local broker_list = {
                { host = "39.98.133.153", port = 9103 },
            }
            -- 定义json便于日志数据整理收集
            local log_json = {}
            log_json["message"]="nginx:"..(ngx.var.uri)
            log_json["from"]="nginx"
            log_json["rid"]=kafkatools.getRandomStr(10)

            -- 取cookie里的sid
            log_json["sid"]=ngx.var.cookie_sid
            -- 拼接时间参数
            ngx.update_time()
            log_json["time"]=ngx.now()

            log_json["tid"]=kafkatools.getDevice()
            log_json["ip"]=kafkatools.getClientIp()
            -- 转换json为字符串
            local message = cjson.encode(log_json);
            -- 定义kafka异步生产者
            local bp = producer:new(broker_list, { producer_type = "async" })
```

```
            -- 发送日志消息,send第二个参数key,用于kafka路由控制:
            -- key为nill(空)时，一段时间向同一partition写入数据
            -- 指定key，按照key的hash写入到对应的partition
            local ok, err = bp:send("demo2", nil, message)

            if not ok then
                ngx.log(ngx.ERR, "kafka send err:", err)
                return
            end
        ';
```

2）重新请求登陆，注意，必须是经过nginx代理的地址：http://localhost/api/login?username=zhangsan&password=123 如果直接请求微服务端口上的rest，生成的cookie会因为不同域，而取不到响应的值。

3）重新请求首页index.html

4）进入kibana查看采集情况

235 hits

time	rid	sid	tid	ip	from	url
1583209361.422	FUV10bS06RF10	zhangsan	macintosh	127.0.0.1	nginx, kafka	/index.html
1583209361.152	0U102638EGL9	zhangsan	macintosh	127.0.0.1	nginx, kafka	/index.html
1583209360.619	6740136b6W9	zhangsan	macintosh	127.0.0.1	nginx, kafka	/index.html
1583209360.851	W1YC06DNbH7	zhangsan	macintosh	127.0.0.1	nginx, kafka	/index.html

4.5.4 整体测试

1）清除所有cookie

2）nginx中新建登录页login.html

```
<form action="/api/login">
    <input name="username"/>
    <input type="password"/>
    <button type="submit">submit</button>
</form>
```

3）请求 index.html验证sid为空

4）请求login.html

5）登陆，再次请求index.html验证sid

6）请求后台接口 /api/test ，验证完整的请求链路

4.6 总结

1. 前端链路收集：lua+kafka
2. 微服务filter，第一道关卡
3. threadlocal服务内上下文传递
4. 切面和注解配合完成日志的自动打印
5. 乱序问题
6. 跨服务的传递
7. 用户登陆的sid

附：

kafka可以配置多个topic，为每个微服务提供特定的topic，有助于提升性能