

Kafka+ELK

概述

ELK (Elasticsearch、Logstash、Kibana) + Kafka来搭建一个日志系统。

什么是ELK?

Elasticsearch是一个基于Lucene、分布式、通过Restful方式进行交互的近实时搜索平台框架。像类似百度、谷歌这种大数据全文搜索引擎的场景都可以使用Elasticsearch作为底层支持框架，可见Elasticsearch提供的搜索能力确实强大,市面上很多时候我们简称Elasticsearch为es。

当然，Elasticsearch 不仅仅是 Lucene，并且也不仅仅只是一个全文搜索引擎。它可以被下面这样准确地形容：

- 一个分布式的实时文档存储，每个字段可以被索引与搜索；
- 一个分布式实时分析搜索引擎；
- 能胜任上百个服务节点的扩展，并支持 PB 级别的结构化或者非结构化数据。

Logstash是ELK的中央数据流引擎，用于从不同目标（文件/数据存储/MQ）收集的不同格式数据，经过过滤后支持输出到不同目的地（文件/MQ/redis/Elasticsearch/Kafka等）。

Kibana可以将Elasticsearch的数据通过友好的页面展示出来，提供实时分析的功能。

为什么用ELK?

1. 以前不用ELK的做法

一般单体结构的项目使用log4j来把日志写到log文件中。

微服务之后，项目有了高可用的要求，进行了分布式部署web。如果我们还是用log4j这样的方式来记录log的话，那么有多少个分布式机器，就有多少个日志记录，这个时候查找log起来非常麻烦，不方便定位bug。后来，直接将log写到数据库中去，这样做，虽然解决了查找异常信息便利性的问题了，但存在两个缺陷：

1. log记录一多，表不够用，必须分库分表
2. 使用数据库必须考虑到数据库的异常，如果数据库异常，log就会出现丢失了。那么为了解决log丢失的问题，那么还得先将log写在本地，然后等db连通了后，再将log同步到db。

2. 现在ELK的做法

ELK方案，可以解决以上问题。

首先是，使用Elasticsearch来存储日志信息，对一般系统来说可以理解为可以存储无限条数据，因为Elasticsearch有良好的扩展性，然后是有一个Logstash，可以把理解为数据接口，为Elasticsearch对接外面过来的log数据，它对接的渠道，有Kafka、log、redis等等，最后还有一个部分就是kibana，它主要用来做数据展现，log那么多数据都存放在Elasticsearch中，需要可视化展示，这个kibana就是为了让我们的看log数据的，但还有一个更重要的功能是，可以编辑N种图表形式，什么柱状图，折线图等等，来对log数据进行直观的展现。

业务流程：

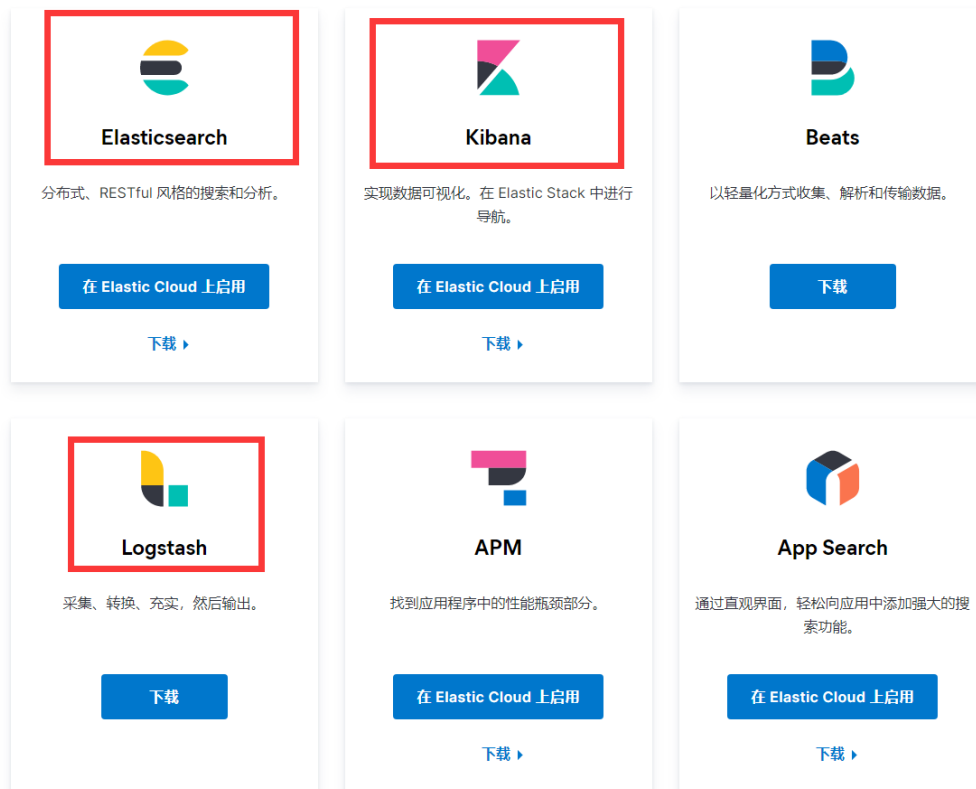
1. 使用Spring aop进行日志收集

2. 通过Kafka将日志发送给Logstash
3. Logstash再将日志写入Elasticsearch，这样Elasticsearch就有了日志数据了。
4. 使用Kibana将存放在Elasticsearch中的日志数据显示出来，实时的数据图表分析。




ELK搭建

1. 下载ElasticSearch+Logstash+Kibana

官网地址<https://www.elastic.co/downloads>



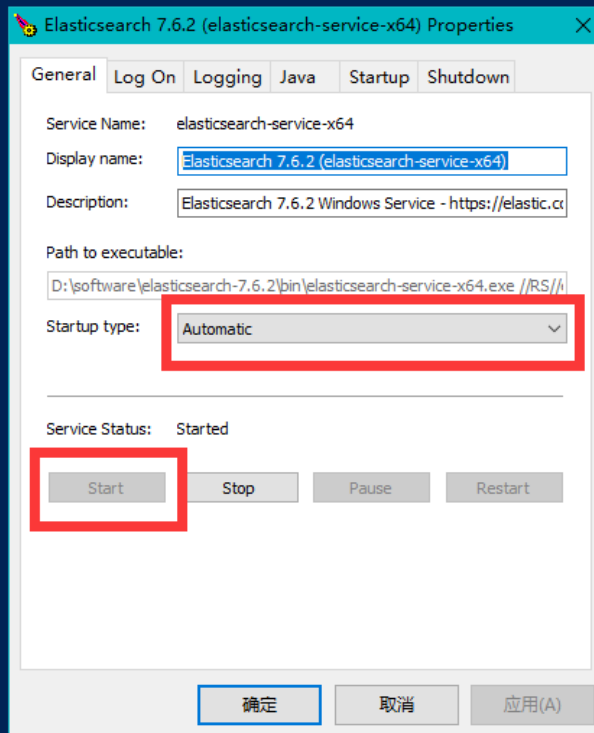
下载后解压

	elasticsearch-7.6.2-windows-x86_64.zip	2020/4/27 9:25	压缩(zipped)文件...	286,268 KB
	kibana-7.6.2-windows-x86_64.zip	2020/4/27 9:27	压缩(zipped)文件...	290,699 KB
	logstash-7.6.2.zip	2020/4/27 9:24	压缩(zipped)文件...	175,362 KB

2. 启动 Elasticsearch

- 打开elasticsearch-7.6.2\bin，cmd运行elasticsearch-service.bat install
- 运行 elasticsearch-service.bat manager 管理配置ES，点击Start启动服务

```
S D:\software\elasticsearch-7.6.2\bin> ./elasticsearch-service.bat manager
```



- 这里可以设置automatic开机自启。
- 输入网址 <http://localhost:9200/>，可以看到如下信息

```
< > ↻ 🏠 📖 | ☆ localhost:9200

{
  "name" : "DESKTOP-T5Q6LSJ",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "tGYFisntRFSGKv8h-ZJbPA",
  "version" : {
    "number" : "7.6.2",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "ef48eb35cf30adf4db14086e8aabd07ef6fb113f",
    "build_date" : "2020-03-26T06:34:37.794943Z",
    "build_snapshot" : false,
    "lucene_version" : "8.4.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

其中会有版本号等信息。

3. Logstash

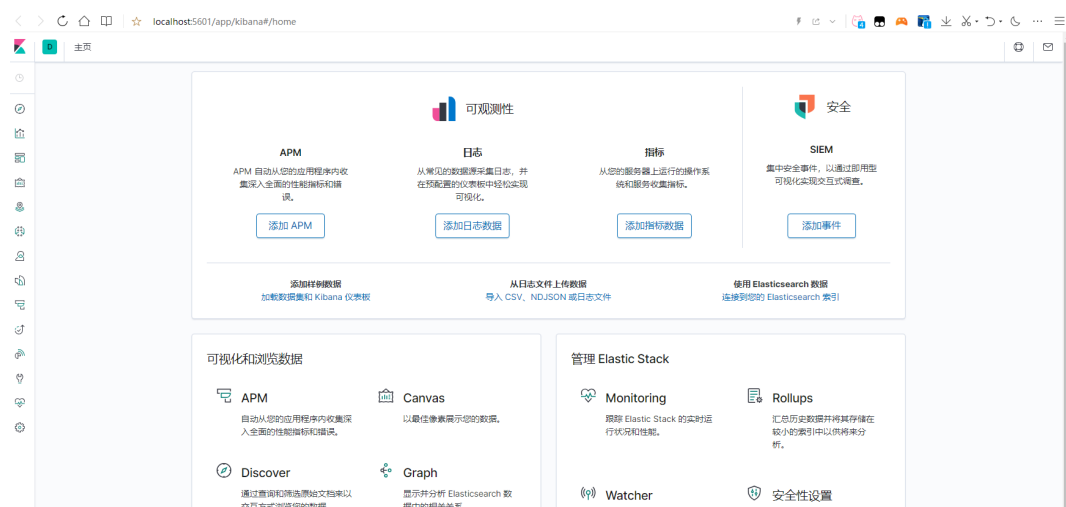
Logstash是一个导入导出数据的工具，使用时直接编辑配置文件，输入指令即可。下面会结合实例介绍具体用法。

4. 启动Kibana

- 进入kibana-7.6.2-windows-x86_64\bin，双击运行kibana.bat

```
C:\Windows\system32\cmd.exe
log [02:08:24.815] [info] [status] [plugin:oss_telemetry@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.819] [info] [status] [plugin:file_upload@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.823] [info] [status] [plugin:data@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.830] [info] [status] [plugin:lens@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.843] [info] [status] [plugin:snapshot_restore@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.850] [info] [status] [plugin:input_control_vis@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.852] [info] [status] [plugin:kibana_react@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.853] [info] [status] [plugin:management@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.855] [info] [status] [plugin:navigation@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.857] [info] [status] [plugin:region_map@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.863] [info] [status] [plugin:telemetry@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.885] [info] [status] [plugin:timelion@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.899] [info] [status] [plugin:ui_metric@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.900] [info] [status] [plugin:markdown_vis@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.901] [info] [status] [plugin:metric_vis@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.993] [info] [status] [plugin:vega@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.995] [info] [status] [plugin>tagcloud@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:24.996] [info] [status] [plugin:table_vis@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:26.507] [warning] [reporting] 正在为 xpack.reporting.encryptionKey 生成随机密钥。要防止待处理报告在重新启动时失败,请在 kibana.yml 中设置 xpack.reporting.encryptionKey
log [02:08:26.513] [info] [status] [plugin:reporting@7.6.2] Status changed from uninitialized to green - Ready
log [02:08:26.532] [info] [listening] Server running at http://localhost:5601
log [02:08:26.636] [info] [server] [Kibana] [http] http server running at http://localhost:5601
Could not get dynamic index pattern because indices "apm-*" don't exist
Could not get dynamic index pattern because indices "apm-*" don't exist
Could not get dynamic index pattern because indices "apm-*" don't exist
```

- 可以访问 <http://localhost:5601> 查看是否启动成功

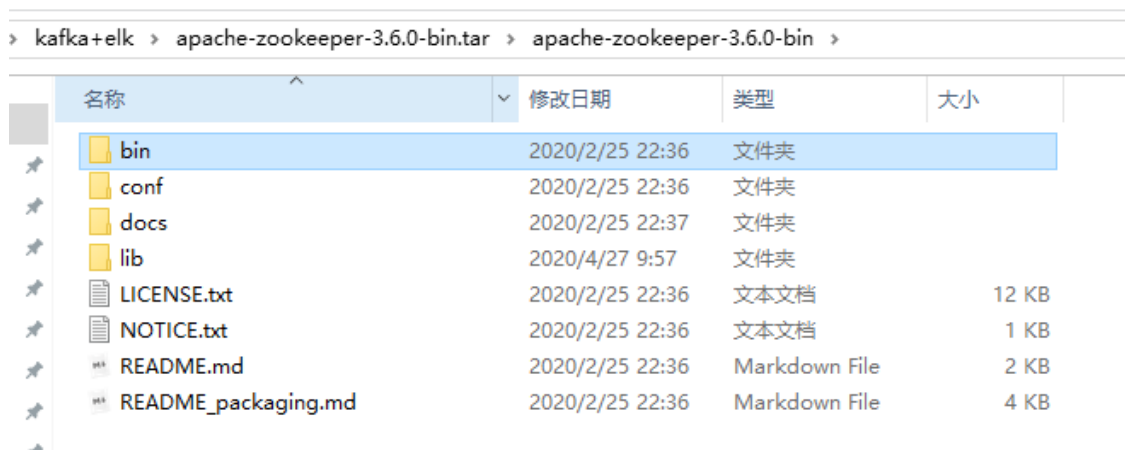


Kafka搭建

1. 安装ZooKeeper

因为Kafka依赖于ZooKeeper, 所以先安装Zookeeper。

- 下载ZooKeeper, <https://downloads.apache.org/zookeeper/zookeeper-3.6.0/apache-zookeeper-3.6.0-bin.tar.gz>
- 解压缩



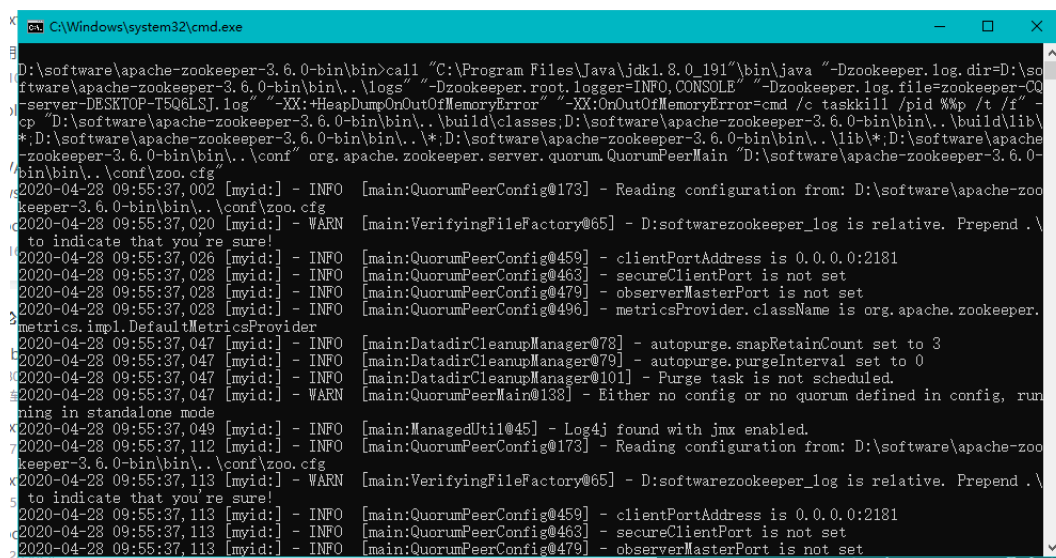
- 修改配置文件

\apache-zookeeper-3.6.0\conf下, 重命名zoo_sample.cfg为zoo.cfg

如：修改端口号，log文件地址

```
1 # The number of milliseconds of each tick
2 tickTime=2000
3 # The number of ticks that the initial
4 # synchronization phase can take
5 initLimit=10
6 # The number of ticks that can pass between
7 # sending a request and getting an acknowledgement
8 syncLimit=5
9 # the directory where the snapshot is stored.
10 # do not use /tmp for storage, /tmp here is just
11 # example sakes.
12 dataDir=D:\software\zookeeper_log
13 # the port at which the clients will connect
14 clientPort=2181
15 # the maximum number of client connections.
16 # increase this if you need to handle more clients
17 #maxClientCnxns=60
18 #
19 # Be sure to read the maintenance section of the
20 # administrator guide before turning on autopurge.
21 #
22 # http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_ma
```

- 运行zookeeper，打开 zookeeper-3.6.0\bin 目录，运行 zkServer.cmd



可以输入jps查看是否启动

```
C:\Users\CQ>jps
1232 AuthBootstrap
18752 ConsoleProducer
1156 Kafka
20852
10840 QuorumPeerMain
1736 Logstash
21976 Launcher
7832 BasicSystemBootstrap
21580 GatewayBootstrap
5004 Jps
556 RemoteMavenServer

C:\Users\CQ>
```

2. 安装Kafka

- 下载地址<http://kafka.apache.org/downloads.html>
- 修改配置文件，进入kafka_2.12-2.5.0\config，修改server.properties


```

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

```

KafkaProducerConfig

配置Kafka的连接信息，配置了ip、端口号，注意添加注解@Configuration和@EnableKafka

```

/**
 * pmp
 *
 * @author : CQ
 * @date : 2020-04-28 09:13
 **/
@Configuration
@EnableKafka
public class KafkaProducerConfig {

    public Map<String, Object> producerConfigs() {
        Map<String, Object> props = new HashMap<>();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "127.0.0.1:9092");
        // 如果请求失败，生产者会自动重试，我们指定是0次，如果启用重试，则会有重复消息的可能性
        props.put(ProducerConfig.RETRIES_CONFIG, 0);
        /**
         * Server完成 producer request 前需要确认的数量。 acks=0时，producer不会等待确
         认，直接添加到socket等待发送；
         * acks=1时，等待leader写到local log就行； acks=all或acks=-1时，等待isr中所有
         副本确认（注意：确认都是 broker
         * 接收到消息放入内存就直接返回确认，不是需要等待数据写入磁盘后才返回确认，这也是
         kafka快的原因）
         */
        // props.put("acks", "all");

        /**
         * Producer可以将发往同一个Partition的数据做成一个Produce
         * Request发送请求，即Batch批处理，以减少请求次数，该值即为每次批处理的大小。
         * 另外每个Request请求包含多个Batch，每个Batch对应一个Partition，且一个Request
         发送的目的Broker均为这些partition的leader副本。
         * 若将该值设为0，则不会进行批处理
         */
        props.put(ProducerConfig.BATCH_SIZE_CONFIG, 4096);
        /**
         * 默认缓冲可立即发送，即缓冲空间还没有满，但是，如果你想减少请求的数量，可以设置
         linger.ms大于0。
         * 这将指示生产者发送请求之前等待一段时间，希望更多的消息填补到未满的批中。这类似于TCP
         的算法，例如上面的代码段，
         * 可能100条消息在一个请求发送，因为我们设置了linger(逗留)时间为1毫秒，然后，如果我
         们没有填满缓冲区，
         * 这个设置将增加1毫秒的延迟请求以等待更多的消息。 需要注意的是，在高负载下，相近的时间
         一般也会组成批，即使是

```

```

        * linger.ms=0。在不处于高负载的情况下，如果设置比0大，以少量的延迟代价换取更少的，
        更有效的请求。
        */
        props.put(ProducerConfig.LINGER_MS_CONFIG, 1);
    /**
     * 控制生产者可用的缓存总量，如果消息发送速度比其传输到服务器的快，将会耗尽这个缓存空
    间。
     * 当缓存空间耗尽，其他发送调用将被阻塞，阻塞时间的阈值通过max.block.ms设定， 之后它
    将抛出一个TimeoutException。
     */
        props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 40960);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        return props;
    }

    public ProducerFactory<String, String> producerFactory() {
        return new DefaultKafkaProducerFactory<>(producerConfigs());
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<String, String>(producerFactory());
    }
}

```

创建AOP注解

```

/**
 * @author CQ
 */
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface SystemLog {
}

```

对带有注解的方法进行处理

处理参数，发起信息

注意：kafkaTemplate.send("test-log",message.toJSONString())

将数据发送给Kafka， test-log是topic名

```

/**
 * pmp
 *
 * @author : CQ
 * @date : 2020-04-28 09:16
 */
@Aspect
@Component
public class LogInterceptor implements Ordered{

```



```

@Autowired
private kafkaTemplate kafkaTemplate;

@Around("@annotation(systemLog)")
public Object Log(ProceedingJoinPoint joinPoint, SystemLog systemLog){

    Object result = null;
    try {
        if (joinPoint == null) {
            return null;
        }
        JSONObject message = new JSONObject();
        HttpServletRequest request = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getRequest();

        Date now = new Date();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd
HH:mm:ss"); //可以方便地修改日期格式
        message.put("time", dateFormat.format(now)); //时间

        message.put("ip", getIpAddr(request));
        message.put("requestURL", request.getRequestURL().toString());
        message.put("params", request.getQueryString()); //参数

        result = joinPoint.proceed();
        message.put("return", result); //返回结果

        //message.put("requestMethod",
joinPoint.getSignature().getName()); //方法
        //message.put("class", joinPoint.getTarget().getClass().getName());

        kafkaTemplate.send("test-log", message.toJSONString());
        System.out.println("message:" + message.toJSONString());

    } catch (Throwable throwable) {
        throwable.getMessage();
    }
    return result;
}

/**
 * 如果不需要ip地址，这段可以省略
 */
public String getIpAddr(HttpServletRequest request) {
    String ipAddress = null;
    ipAddress = request.getHeader("x-forwarded-for");
    if (ipAddress == null || ipAddress.length() == 0
        || "unknown".equalsIgnoreCase(ipAddress)) {
        ipAddress = request.getHeader("Proxy-Client-IP");
    }
    if (ipAddress == null || ipAddress.length() == 0
        || "unknown".equalsIgnoreCase(ipAddress)) {
        ipAddress = request.getHeader("WL-Proxy-Client-IP");
    }
    if (ipAddress == null || ipAddress.length() == 0
        || "unknown".equalsIgnoreCase(ipAddress)) {

```

```

        ipAddress = request.getRemoteAddr();
    }
    // 对于通过多个代理的情况，第一个IP为客户端真实IP,多个IP按照','分割
    // "****.***.***.***".length()
    if (ipAddress != null && ipAddress.length() > 15) {
        // = 15
        if (ipAddress.indexOf(",") > 0) {
            ipAddress = ipAddress.substring(0, ipAddress.indexOf(","));
        }
    }
    //或者这样也行,对于通过多个代理的情况，第一个IP为客户端真实IP,多个IP按照','分割
    //return ipAddress!=null&&!"".equals(ipAddress)?ipAddress.split(",")
    [0]:null;
    return ipAddress;
}

@Override
public int getOrder() {
    return 0;
}
}

```

controller中使用注解，访问对应url时便会触发kafka发送

```

/**
 * 用户服务提供
 *
 * @author cq
 */
@RestController
@RequestMapping("/api")
public class UserController {
    @SystemLog
    @GetMapping("no")
    @PreAuthorize("hasAnyAuthority('no')")
    public String hello(){
        return "no";
    }

    @SystemLog
    @GetMapping("current")
    public Principal user(Principal principal) {
        return principal;
    }

    @SystemLog
    @GetMapping("yes")
    @PreAuthorize("hasAnyAuthority('yes')")
    public String query() {
        return "拥有yes权限";
    }
}

```

```
message:{"requestURL":"http://192.168.6.129:56030/api/yes","ip":"0:0:0:0:0:0:1","time":"2020/04/30 10:58:35","params":{"access_t
```

使用命令 `bin/kafka-topics.sh --zookeeper 127.0.0.1:2181 --topic test-log --describe`，可以查看分区等信息

```
S D:\software\kafka_2.12-2.5.0\bin\windows> .\kafka-topics.bat --zookeeper 127.0.0.1:2181 --topic test-log --describe
Topic: test-log PartitionCount: 1      ReplicationFactor: 1      Configs:
        Topic: test-log Partition: 0    Leader: 0      Replicas: 0      Isr: 0
S D:\software\kafka_2.12-2.5.0\bin\windows>
```

接下来就是用Logstash把数据从Kafka put 到Elasticsearch

在logstash下的config文件夹内新建一个.conf文件

需要在conf文件内加上解析JSON的filter

```
# Sample Logstash configuration for creating a simple
# Beats -> Logstash -> Elasticsearch pipeline.

input {
  kafka {
    bootstrap_servers => "127.0.0.1:9092"
    topics => ["test-log"]
  }
}
filter {
  json{
    source => "message"
  }
}
output {
  elasticsearch {
    hosts => "127.0.0.1:9200"
    action => "index"
    index => "test-log--%{+YYYY.MM.dd}"
    codec => "json"
  }
}
```

在logstash的bin目录下执行：

```
./logstash -f ../config/logstash-test.conf --config.reload.automatic
```

成功的话，数据就到elasticsearch内了，验证：

http://localhost:9200/_cat/indices?v

localhost:9200/_cat/indices?v									
health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
green	open	.kibana_task_manager_1	tkzBHpPdSTu15nMEfpwRbA	1	0	2	1	10.4kb	10.4kb
yellow	open	test-log--2020.04.28	9FLI9B1YKw42LypQ-48Yg	1	1	26	0	90kb	90kb
green	open	ilm-history-1-000001	PzsFxo2pSnioOfqkN48zQ	1	0	18	0	25.4kb	25.4kb
green	open	.apm-agent-configuration	ZVeS3iu2SgCpMhZaaHRLA	1	0	0	0	283b	283b
yellow	open	logstash-2020.04.27-000001	Gf4kp_aQqi00aaC6i0PJhw	1	1	0	0	283b	283b
green	open	.kibana_1	nYz_VYH7R3KWT3U3emplAg	1	0	18	1	31.9kb	31.9kb

操作Kibana: