**Argus**

**2016-5-23**

修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 版本 | 描述 | 作者 | 备注 |
| 2016/5/25 | V1.0 | 初稿 | zhang |  |
| 2016/6/2 | V1.1 | 进一步细化，完善实现方案 | zhang | deadline:6.7 |

目录

[1 背景 5](#_Toc473996960)

[2 现状和挑战 5](#_Toc473996961)

[3 目标 6](#_Toc473996962)

[4 Argus总体架构 7](#_Toc473996963)

[4.1 Argus架构示意图 7](#_Toc473996964)

[4.1.1 系统概要说明 7](#_Toc473996965)

[4.2 监控核心 8](#_Toc473996966)

[5 Argus功能概要 9](#_Toc473996967)

[5.1 监控信息接入系统 9](#_Toc473996968)

[5.1.1 功能描述 9](#_Toc473996969)

[5.2 监控系统 9](#_Toc473996970)

[5.2.1 被监控服务 10](#_Toc473996971)

[5.2.2 监控项和监控策略 10](#_Toc473996972)

[5.3 报警系统 11](#_Toc473996973)

[5.3.1 报警策略 12](#_Toc473996974)

[5.3.2 基本报警策略 13](#_Toc473996975)

[5.3.3 报警屏蔽 13](#_Toc473996976)

[5.4 监控配置系统 14](#_Toc473996977)

[5.5 后期扩展 14](#_Toc473996978)

[5.5.1 监控报警 14](#_Toc473996979)

[5.5.2 应用级信息统计 14](#_Toc473996980)

[6 Argus实现方案 15](#_Toc473996981)

[6.1 日志接入层 15](#_Toc473996982)

[6.1.1 日志采集服务 15](#_Toc473996983)

[6.1.2 监控准入校验 19](#_Toc473996984)

[6.1.3 日志过滤 19](#_Toc473996985)

[6.1.4 日志规范化 19](#_Toc473996986)

[6.1.5 日志流入队 20](#_Toc473996987)

[6.2 监控系统 21](#_Toc473996988)

[6.2.1 日志流抽取 22](#_Toc473996989)

[6.2.2 监控配置加载(规则引擎) 22](#_Toc473996990)

[6.2.3 监控项过滤 22](#_Toc473996991)

[6.2.4 监控规则应用 23](#_Toc473996992)

[6.2.5 监控策略类型 24](#_Toc473996993)

[6.3 报警系统 27](#_Toc473996994)

[6.4 报警系统实现 27](#_Toc473996995)

[6.5 监控配置系统 27](#_Toc473996996)

[6.6 存储 27](#_Toc473996997)

[7 实现方案 27](#_Toc473996998)

# 背景

目前app后端服务缺乏有效的统一监控系统，运维部门目前对机器、磁盘、网络有商用的监控，但是在业务方面缺少实时的、监控策略灵活配置的监控系统 。对于业务的一些异常，无法实时获知，对业务的稳定性和可用性带来影响。

# 现状和挑战

目前日志的分析都使用hadoop来做，其过程简单来讲，即将准备处理的数据上传至hdfs，将其按照大小进行数据分片(Map/Reduce)，通过层层worker的处理，最终输出结果。对于日志监控这种实时性要求较高的场景来讲，有以下不足：

1. 实时性较差，数据要进入hdfs，再Map/Reduce，响应不会太积极；
2. 监控一般只关心少量数据，不必持久化存储所有日志，而将全部日志持久化入hdfs，有点得不偿失。

当前app后端是使用elk的全套服务来做日志的归集和索引，虽然可以使用相关脚本来进行一些关键词的统计，但是无法做到配置灵活、实时的产生报警，可扩展性差，无法支持自定义报警推送，以及一些中间件运行时状态的监控。

**面临的挑战主要是：**

1、各种应用系统的日志格式以及方式不一，难以通过统一的一个日志模型来进行抽象；

2、每天的日志量比较大，如果快速、准确的进行日志过滤是个挑战；

3、目前缺少一种监控报警模型来进行系统层面的概括，同时缺少灵活可配置的监控、报警策略；

Storm是一个免费且开源的分布式实时计算系统，它使得海量数据的可靠处理变得很简单，同时storm可以实时的完成和Hadoop批处理一样的功能。

Storm有各种各样的使用场景，如实时分析，在线机器学习，连续计算，DRPC，ETL等，Storm性能很高，基准的benchmark显示，其单节点在每秒钟可以处理高达100W个tople。[[refer](http://storm.apache.org/)]

拟采用基于agent的数据采集、storm的实时流式处理监控数据源（如日志，运行时参数）配合各种灵活的监控、报警策略来处理，有以下优点：

1. 对现有应用改动小，部署agent，往统一的入口push日志流即可；
2. Storm相比hadoop更轻量级，实时性和性能均很高，有效满足业务场景需要；
3. 监控、报警策略灵活可配，无论在业务功能上以及系统性能上都有较高的伸缩性；
4. 数据实时流式处理，只对关心的数据进行持久化，不必对全量数据存储，有效节省存储资源，同时满足高性能处理。

除此之外，可以对团队在实时计算、流式处理、机器学习等方面做一些技术储备。

# 目标

设计开发一个业务监控报警系统，实现如下目标：

1. 对线上日志实时分析处理，触发监控项，应用监控策略和预设的报警方式，由值班人接收报警信息。【一期】
2. 支持外部接口调用，由调用方push报警信息和报警方式，应用系统自定义的监控策略，触发报警。【二期】
3. 支持探测机机制，由监控探测机进行端口、服务的可用性进行存活探测，触发报警。【二期】
4. 支持一些外部工具的监控，如redis、jvm等运行时状态的监控。【三期】

# Argus总体架构

## Argus架构示意图

图1 argus系统总体结构

### 系统概要说明

1. **日志****接入系统：**

由于进行的是线上日志的实时分析，所以数据量会很大，如果直接进入监控系统，会给系统带来冲击，所以数据buffer作为数据进入监控系统前的缓冲，同时允许一定的数据丢弃策略，如将非com.creditease.service\*的包产生的日志丢弃掉。

1. **监控系统**

该子系统是整个系统的核心功能所在，根据预定的监控项，对于采集到的日志，进行实时分析，对于触发的监控项，应用监控策略，决定是否生成报警。对于生成的报警信息一方面进行持久化，另一方面推送至报警系统，给预定的值班人发送报警信息。

1. **报警系统**

接收监控系统的报警信息，应用预定的监控策略，向值班人发送报警，支持多种报警方式。有重试机制。

1. **监控配置系统**

该系统作为整个监控系统的MIS，提供可视化的界面进行监控项、监控策略、预定报警方式、值班人、以及系统配置功能，同时提供报警信息查询以及统计分析功能。

1. **存储**

对报警信息进行持久化，用于查询、统计分析。注意，如果进行了报警合并，如某监控项产生了10次报警，报警系统将10次报警合并为一条报警短信，那么仍然存储10条报警信息。

## 监控核心



# Argus功能概要

## 监控信息接入系统

### 功能描述

日志接入层采用缓存队列来存储采集节点推送来的日志数据，队列使用redis或者mq来进行缓存，使用持久化策略，在该缓存之前，提供统一的日志接入服务，各服务器的日志采集模块统一将日志push到该接入服务，由该接入服务分发到缓存队列。

该服务的功能如下：

1. 进行日志的初步过滤、清洗，如过滤掉无意义的文本，支持规则配置的过滤方法。
2. 进行日志的标准化处理，按照系统预定义的日志实体模型进行规范化。
3. 进行日志统一模型实体信息的分发，如分发到相应的处理队列，由下游系统进行处理。同时监控队列的状态，如果队列full，则发出报警并进行接入服务降级，如采取包丢弃策略和拒绝服务，或者对下游处理系统资源进行调度，以保护监控系统防止洪泛。

图3 日志接入层

## 监控系统

监控系统作为整个系统的核心部分，是基本功能的体现，为提供可扩展、灵活配置的服务，监控系统处理的监控模型定义如下：

### 被监控服务

目标系统即是本系统的监控对象，相关信息采集服务即运行在该系统上，通过服务和所在的机器ip（域名）来唯一标识。每当新接入一个被监控系统，该系统即在本系统中注册，生成唯一标识，该标识需要和信息采集服务一致。以上由监控配置信息来统一管理。

### 监控项和监控策略

**监控项**：定义要监控什么，由什么来触发报警。如，根据预设定的关键词，日志中出现“放款失败”的关键词，则认为业务系统发生了一个业务异常，定义为异常单元，而“放款失败“即是一个监控项。

每个监控项在配置时，必须要和某个被监控服务对应。二者的实体关系如下：



图 监控项和被监控服务实体关系

每个监控项唯一的属于一个被监控服务。每个被监控服务可以有多个监控项。

**监控策略**：对于直接命中监控项的异常单元并不直接生成监控异常，而是通过一定的策略去生成。如“交易失败”的监控项触发了异常单元，该异常单元在10s内发生了三次，则生成监控异常。这个“*10s内发生3次*”的策略即为监控策略。这样做的目的是，对于偶尔小概率发生的一些业务异常，在监控层面有更加灵活、细分的策略去进行定义。只有监控项触发的异常单元满足预定的监控策略才会生成监控异常。监控项、异常单元、监控策略、监控异常的模型间逻辑关系如下：



监控项具有以下基本属性：

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | | **属性定义** | **说明** |
| 监控项名称 | | 对监控项的名字定义 | 如“交易失败” |
| 监控项 | | 触发异常单元的关键词定义 | 如“交易失败”，”loan [(pay)(repay)]+error ” |
| 监控策略 | 异常次数 | 监控项触发产生异常单元的统计周期，单位秒 | 如3次 |
| 监控周期 | 在监控周期内触发几次异常生成监控异常 | 如10秒，1分钟，2天等 |
| 优先级 | | 该监控项触发的监控报警的优先级 | 如高、中、低，对于高的，不加合并直接发送出去，对于中、低可以等一段时间如果有相同报警合并发送出去 |
| 报警接收人 | | 报警对象，姓名 |  |
| 报警方式 | | 短信、邮件、微信.. |  |

**基本监控策略：**

|  |  |  |
| --- | --- | --- |
| **策略名** | **策略规则** | **说明** |
| 周期性策略 | (S0\_cnt>5)/10s | 10s内大于5次 |
| 复合型策略 | (S1\_cnt>5 &S2\_cnt>10)/10s | 在10s内监控策略S1出现5次并且S2出现10次 |
| (S1\_cnt>5 ||S2\_cnt>10)/10s | 在10s内监控策略S1出现5次或者S2出现10次 |
| 其他 | 待定 |  |

## 报警系统

监控策略决定了哪些监控项可以生成报警单元，而对应的有灵活的监控策略决定哪些异常单元可以生成监控异常，而对于这些异常，不应该不加处理的直接生成报警推送给相关人。如在一些极端情况下，如一些业务依赖的服务器down机，会导致监控异常洪泛，如果异常不加处理，会导致报警系统压力巨大，从而导致短信、邮件系统压力飙升，带来雪崩效应。为避免这种情况，应该有完善灵活的报警策略。报警系统的相关模型定义如下：

### 报警策略

报警策略是和监控项对应的，它决定什么情况下才将监控异常的报警发出，如5.2中的“交易失败”的监控异常，可以定义为在10s内发生3次才发短信，也可以定义为1分钟内最多报警2次，1天内最多报警100次，每报警2次之后，隔900s再重新提醒期间总共发生的报警次数。虽然报警信息得到了合并，但是在数据存储上，仍然记录每一条报警。如900s内发生了300次报警，虽然只发了一条短信，但是后台仍然记录300条报警记录。

类似的，报警系统和监控系统的模型图如下：



图 监控&报警模型

结合监控系统，整体的模型如下所示：



图 监控&报警整体模型

### 基本报警策略

根据报警策略的定义以及要解决的问题，现给出几个基本的报警策略：

|  |  |  |
| --- | --- | --- |
| **策略名** | **说明** | **备注** |
| 每次发生即立刻警 | N=1 |  |
| 每x秒有n次报警 | n/xs |  |
| 连续x次产生报警 | N=x |  |
| 超过最大报警次数n次后x秒重新报警 | N|x |  |

### 报警屏蔽

支持报警屏蔽，关闭可预期的报警发送动作，屏蔽策略如下：

1. 时间维度。包括一个连续的时间段，如连续1小时不再执行报警发送动作；一个固定的时间段，如每天的0点到6点不再报警；节假日不报警；周末不报警等等。
2. 屏蔽粒度。如某个监控项不产生报警；特定的应用服务不报警。不支持全部关闭，粒度最大到应用级别。
3. 报警对象，屏蔽特定的报警接收人。

## 监控配置系统

大体功能如下：

1. 监控项、监控策略配置。
2. 报警、报警策略配置，含报警接收人，报警方式配置。
3. 系统参数配置，队列、storm运行状态监控。队列列表维护、加载，运行时接入新的被监控应用。
4. 报警Web页面查看（短网址跳转，附在报警短信和邮件中，查看报警详情）

## 后期扩展

### 监控报警

1. 自定义报警push，提供rpc、http等接口，供相关系统自定义push报警信息。
2. 探测机系统，提供相关服务的可用性探测（存活探测），或支持自定义语义结构的请求-应答服务。
3. 中间件、第三方应用的健康状况监控，如jvm堆、线程池数量的监控。
4. 和商业以及开源的监控系统整合，提供扩展服务，进行系统级的监控，如cpu、硬盘、内存、进程等监控。

### 应用级信息统计

1. 系统的qps、tps统计，pv、uv的实时统计；
2. 访问路径、响应时间统计；
3. 监控特定用户访问痕迹，请求追踪等；
4. 其他涉及日志数据的相关业务应用场景。

# Argus实现方案

## 日志接入层

日志流进入接入层后，不能直接进入监控系统，需要做预处理，该层的处理流程如下：



图 接入层处理流程

### 日志采集服务

日志采集服务基于已有的logstash，由其agent采集日志推送至统一的日志接入服务。Logstash在push日志时，需要携带argus提供的准入认证信息，即在header中携带argus系统分配给该系统的唯一的ARGUS\_ID，argus会在日志接入时校验该id是否和系统中配置的host一致，一致则进行日志包的过滤处理，否则即丢弃。

Logstash的push脚本如下定义[[参考官网Document](https://www.elastic.co/guide/en/logstash/current/plugins-outputs-http.html)]：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类别** | **Key** | | **value** | **备注** |
| out | format | | json | 以json方式push |
| http\_method | | post | 使用post方式 |
| url | | http://xx.xx.xx | 日志push的统一入口 |
| headers | **ARGUS\_SERVICE\_ID** | 一个16位的随机字符串，只包含数字和字母，全部大写 | 唯一标识接入系统，该id在argus白名单中并且和HOST一一对应才允许接入，否则丢弃，由argus系统分配。如app后台有4台机器，则这4台机器用同一个ARGUS\_ID，而不同的机器使用HOST来区分 |
| **HOST** | 日志服务器的主机名 | 标识服务器地址，用于报警查看定位到所在服务器，和ARGUS\_ID共同标识某应用系统的不同节点 |
| filter | 规则待定，为减少接入的配置成本，在该处只做一些基本的过滤，如MDC中只保留operateId，其余全部删掉，其余的更加细化的规则放在接入层处理，以尽可能的减少日志穿透到后端的流量。 | | | |
| 扩展待定 | -- | | -- | -- |

现有的线上日志格式为：

|  |
| --- |
| 2016-07-01 18:19:25.391 | catalina-exec-434 | INFO | c.c.y.c.u.RegexUtil:RegexUtil.java(55) | 2016-07-01 18:19:27.066 | 10.130.80.231 | 125.36.110.184, 125.36.110.184 | 10.131.16.36 | X | ff458f9a-1650-4f2d-90c0-07c3d525b73b | 869161028294080 | | d2e91b5f056b24683d2e0337dbc8afe1 | d2e91b5f056b24683d2e0337dbc8afe1 | X | 6.0 | X | 4C9EF1676B564240DF6AA684F968E85A | 3.0.0 | 1080X1920 | c13aa3fa-2a89-48bc-82e6-2e1743ce4742 | 39.90738138,117.15843576 | X | 姓名滕广娟替换为\*广娟  2016-07-01 18:19:25.391 | catalina-exec-434 | INFO | c.c.y.c.a.DataSourceAnnotationAspect:DataSourceAnnotationAspect.java(46) | 2016-07-01 18:19:27.066 | 10.130.80.231 | 125.36.110.184, 125.36.110.184 | 10.131.16.36 | X | ff458f9a-1650-4f2d-90c0-07c3d525b73b | 869161028294080 | | d2e91b5f056b24683d2e0337dbc8afe1 | d2e91b5f056b24683d2e0337dbc8afe1 | X | 6.0 | X | 4C9EF1676B564240DF6AA684F968E85A | 3.0.0 | 1080X1920 | c13aa3fa-2a89-48bc-82e6-2e1743ce4742 | 39.90738138,117.15843576 | X | 数据源自动织入-getP2puserOrderAmountByPassportId设定数据源为dataSourceKey1 |

配置示例：

input { file {path => "/app/apache-tomcat-6.0.30/apache-tomcat-6.0.30/logs/catalina.out"

start\_position => beginning

ignore\_older => 0

}}

output { http {

format => "json"

http\_method => "post"

headers=>["ARGUS\_SERVICE\_ID","1001GFH0001NJHYT",”HOST”,”%host%”]

url =>"http://10.106.146.103:8080/medicine/log/rvc"

}

备选方案：rsyslog，rsyslog的性能比较高，fsg即使用rsyslog做日志的集中化，每天几百TB的日志归集，性能比较高，实时性很强，稳定性良好，对线上应用服务器资源侵占极少。

### 监控准入校验

argus提供基础的统一http接入服务，多个http服务节点使用nginx做lb，提供统一的接入地址。由logstash往该地址push日志信息。由nginx转发至接入服务，其中ARGUS\_ID的校验可以考虑在nginx层进行拦截处理，减少后端系统流量处理的压力。

除了ARGUS\_SERVICE\_ID的校验，被接入服务器的IP要有白名单规则，因为ARGUS\_SERVICE\_ID是明文传输的，如果某服务截获该ARGUS\_ID，则可以伪造身份接入，因此此处还有IP白名单过滤，如果被接入服务器IP不在白名单，直接拒绝响应。

### 日志过滤

后端系统在获取日志实体后，进行清洗、过滤，过滤规则可以配置，如过滤debug的日志，或者过滤mybatis的日志等，规则的细则可以不断完善，尽量减少日志流入后端系统的流量。具体规则需要结合具体日志来确定。

|  |  |  |
| --- | --- | --- |
| **规则名称** | **规则** | **备注** |
| 待定 |  |  |

### 日志规范化

经过以上流程的处理，当前的日志可以进入后面的流程进行监控，但是此时日志仍然是文本，含有各种信息，如日志级别，时间戳，调用类、包名，具体的信息详情等，需要由一个统一的模型来描述存储。因此，结合实际日志的结构，该模型定义如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key** | **Value** | **Required** | **Default** | **备注** |
| ArgusId | ARGUS\_ID | Y | N/A | 该类字段为基础的信息 |
| HostName | 机器名，如test56.creditease.cn | Y | N/A |
| SystemName | 系统名，如system后台服务 | N | “” |
| TimeStamp | 日志中记录的时间戳 | Y | current |
| detail | 日志详情 | Y |  |
| LogId | 日志操作id(operateId) | N | “” | 这些为适应更多的系统，如不存在logid的日志，或者非j2ee体系下的应用解决方案，如python,nginx日志，access\_log等 |
| SystemTime | 当前argus系统生成的时间戳 | Y | current |
| **PathName** | 日志由谁触发的，如java中的cls name,nginx中的uri | Y | “” |
| **method** | 方法名，或者http的get、post等 | N |  |
| attach | userId | N | N/A | 一些附带的用于扩展的属性，  如用户id，用户的ip等，用于异常原因的摘要，便于定位问题 |
| business |

由于不同的接入系统日志打印的规则都不同，如log id，所以需要有一个灵活能适应各种情况下的解析规则。如果能解析到则生成，解析不到则为空。【提供扩展点】

### 日志流入队

日志标准化之后，下一步即push到日志队列中，供后续job来进行分析、处理。此处拟每个接入系统以(HOST)为维度，每个HOST使用一个队列，redis可以使用一台，也可以使用多台来进行分片。以ARGUS\_ID来做consistent hashing来决定进入哪个分片。

在日志进入分片之前，由接入层获取监控配置系统的所有redis服务器配置，初始化至zookeeper节点，由监控系统的spout来获取队列列表，当日志入队后，要将队列及时注册至对应redis节点下，通知spout对该队列抽取日志流。

图 日志入队策略

此处使用多个redis节点做数据分片，当redis节点down掉后，进行即时的resh进行动态调整。由于监控系统并非一个强业务系统，允许一定的日志丢弃策略，暂不考虑HA，此处的HA作为后续架构优化的一个点。

## 监控系统

日志经过接入层的清洗、处理、标准化之后即进入监控系统，监控系统的整体工作流程如下：



图 监控系统工作流程

### 日志流抽取

日志流的抽取由多个worker实例的spout来完成，而由于日志流的队列分布于各个缓存服务器的分片中，因此此处采用多个spout来遍历所有分片的所有队列来获取。此处有哪些分片，分片中有哪些队列，由Zookeeper来集中维护，spout初始时从zookeeper获取分片列表以及分片下的队列列表，本地cache，并在zookeeper上注册一个listener，如果节点没有变化，则后续spout每次从本地cache中获取队列列表抽取日志发射出去，当注册节点有变化（接入层注册、摘掉和MIS系统维护触发），会主动去获取最新的队列列表抽取日志流，如下面示意图所示：

图[3] 日志流抽取

### 监控配置加载(规则引擎)

监控系统定义了众多的监控项和监控策略，这些策略是结构化存储的，监控的最底层最基础的功能都是围绕这些配置展开，因此，需要在进行日志流的监控处理前将这些配置进行预处理，生成运行时规则去供系统分析运用。

【拟建立规则引擎专门处理，做到可扩展、通用、简单，高内聚，接口化】

### 监控项过滤

该部分的逻辑相对比较固定，只是监控项是变化的，即根据日志包的信息获取ARGUS\_ID，HOST等信息获取对应的配置监控项，进行监控项的查找或者正则匹配，生成异常单元，然后组装异常单元的body报文，发射出去。

异常单元的报文格式如下：

|  |  |  |
| --- | --- | --- |
| **Key** | **Value** | **备注** |
| sn | 日志包的唯一序列号 | argusId(4位)+时间戳(8位)+8位随机数字 |
| ArgusId | argusId |  |
| logid | 触发异常单元的日志logid | 如没有则为空 |
| host | 服务器主机地址 | hostname or ip |
| ItemId | 监控项id |  |
| ItemName | 监控项名称 | 【暂时删掉，异常单元一旦生成，该项作用不大了】 |
| TimeStamp | 日志中解析到的timeStamp | 若无logid，后续追查日志时可以通过host和时间戳定位位置 |
| detail | 异常日志的日志体 | 防止过长影响网络传输效率，考虑压缩，而且此处只显示日志摘要，如包路径方法名，尽可能精简，用于附在报警邮件或者短信中 |

注意该异常单元还要经过监控规则和报警规则的双重过滤才决定是否生成报警信息。

### 监控规则应用

监控规则决定着监控异常的总量，如果不加控制的将每个监控项生成报警，则会导致报警系统巨大的压力，监控系统承担着系统90%左右的运算量，因此，报警系统所占据的整个系统的运算资源上应更为轻量，将更多资源摊派至监控系统，所以监控规则的应用异常关键。

**系统级异常合并**

该规则为基础的通用规则，属于**系统规则**，系统规则会自动应用到所有的被监控系统，如某应用系统的某监控项在10秒内发生1000多次报警，这种情况下，每个报警的详情是什么其实已经不重要，重要的是产生了大量报警的事实，因此，将该报警合并后发送更为合理，这样下游的报警系统原本需要处理1000次报警信息，现在只需要发送1条，大大减轻了资源占有量。

该规则为整个监控系统监控异常数目的上限**阈值**，属于系统级保护性策略。

**应用级异常合并**

由应用在配置监控策略时提供的自定义异常合并策略，该策略的阈值要低于系统级异常合并策略。

以上的合并对象均为每个HOST（暂不以应用ARGUS\_ID为合并对象，因为跨多个机器报警，不容易追查问题）。生成的监控异常对象进入MQ，由报警系统处理生成报警信息推送。

### 监控策略类型

目前根据相关业务场景，设计了6种监控策略，这6种策略基本可以涵盖90%的业务以及系统监控场景。

#### 节点监控策略

该种策略适合此类场景，即，当被监控源出现了期望的一些节点关键词后，即引发报警单元的生成，如，在系统日志监控过程中，如果出现了“交易失败”关键词，即是一个报警，也可以为“**交易失败** *且* **银行卡限额** *或* **交易失败** *且* **支付密码错误**” 等方式，支持正则匹配。

该种监控策略处可以用来报警之外，也可以用来统计，如统计某些操作或者请求的次数，结合一些时间窗口定义，可以输出一些频次的值，如统计登录请求的pv，或者交易请求的qps等。

#### 节点阈值监控

该种策略适合此类场景，即，如某类监控策略在某个时间窗口内发生的次数同配置的阈值相比，达到某个条件即报警；如，某渠道用户的注册量半小时内小于100w即报警；再如，某用户5天之内购买行为小于10次即报警，等等。

该种策略需要和策略6.2.5.1来结合，是建立在该基础之上的。

#### 节点阈值比例监控

该种策略适合此类场景，即，某类监控类型发生的次数占据全部类型中发生的比例和预设的阈值相比，达到某个条件即报警；如，用户注册都会产生发送短信行为，如果在10分钟内，系统产生了1000次发送短信请求，而只有900次发送成功，那么发送成功比例为90%，如果低于95%即报警。再如，在12小时内，系统共处理了1000比交易请求，而只有900比返回成功，那么支付成功率为90%，如果低于95%则产生报警。

#### 定点数据阈值监控

该种策略适合此类场景，即对某些监控数据来源中的特定数据进行监控，如在日志监控中，系统响应前端请求时会返回数据，假如返回JSON的形式，如：

|  |
| --- |
| {  "status": "OK",  "errorCode": "000",  "data": [  {  "productName": "理财产品V1",  "productId": "1",  "curPeriod": "42",  **"productDetailId": "e5a05b6d03ee45589f5704ba7210e3dd"**,  "allocateAmt": "200000",  "soldAmt": "10000",  **"minMobileAmt": "10000",**  "minIncAmt": "10000",  "productCycle": "12",  "closurePeriod": "12",  "preIncomeRate": "10",  "manaFee": "0",  "exitFee": "2",  "normalExitRatio": "0",  },  {  "productName": "理财产品V2",  "productId": "7",  "curPeriod": "5",  "productDetailId": "df07060a8f9c445e8cddbc0ebf8d198e",  "allocateAmt": "3000000",  "soldAmt": "1943000",  "maxRenewDaysOrMonths": "24",  "isNew": "0",  "exitNextOpt": "0",  "raiseInterestFlag": 0,  "raiseInterestRate": 0  } |

如上图中的JSON数据，如果“**e5a05b6d03ee45589f5704ba7210e3dd**”对应的**productDetailId**的JSON对象中的**minMobileAmt** 的值大于1000即产生报警。

系统可以自动对不同类型的数据自动解析，这样在配置时，配置一个锚点数据，配置描述该锚点数据代表的是哪个key的值，如**productDetailId，**系统通过这两个数据定位到数据对象，即JSON的Object，然后获取该Object中的minMobileAmt的值。无论何种数据格式，应用系统前端都是根据一定的规则去解析，存在于日志中的数据也遵循这种规则，这样监控系统根据这种规则去解析。对于常见的JSON以及KEY-Value的表示方式系统可以支持，未来有其他的格式，此处需要扩展解析的策略。

#### 标准异常监控

该种监控策略旨在对日志源中的一些标准的异常进行监控，这样用户可以无需输入异常，只需要选择即可，如NullpointerException,ClassCastException等等。可以维护异常的列表以及根据不同的语言类型来区分。

#### 标准APM监控

该种监控策略旨在对一些如JVM，数据库的运行时参数，redis实例状态等进行监控。如可以配置中间件周期性打印GC log，log中出现了Full GC时即报警。或者JMX周期性打印出当前JVM实例各参数状态，监控日志中的某些参数，获取参数值通过和阈值对比报警。

如redis，可以周期性的获取info指令的输出打印到日志中，对日志中某些参数值进行监控。

|  |
| --- |
| uptime\_in\_seconds:145830  uptime\_in\_days:1  lru\_clock:947459  used\_cpu\_sys:0.02  used\_cpu\_user:0.02  used\_cpu\_sys\_children:0.00  used\_cpu\_user\_children:0.00  connected\_clients:1  connected\_slaves:0  client\_longest\_output\_list:0  client\_biggest\_input\_buf:0 |

该种方式可以灵活的应用到各种系统工具和应用系统中。

## 报警系统

由于系统的整个瓶颈在监控系统，也就是监控项和监控规则的处理和应用，因此，报警系统的负载压力在正常情况下要比监控系统小很多，但是也要考虑接入系统在大规模生成监控异常时的情况，此时会有大量的监控异常涌入报警系统，所以该系统要方便的横向扩展，最佳的策略是能实时动态的横向扩展资源【架构优化的一个点】

1. 报警策略

参见基本[报警策略](#_基本报警策略)

【其他策略待完善，考虑是否取消报警策略，由监控策略来替代，或者二者合并？】

## 报警系统实现

监控系统和报警系统之间通过一个分布式双端队列进行解耦，每个监控系统的监控策略对应一个队列，监控的报警单元进入队列，报警系统维护一个线程池，报警系统实时获取队列列表，每当有新的队列产生即动态发起一个处理线程去处理报警信息。如果队列长时间空闲，则线程终结，以节省资源。需要分布式锁去做同步。

## 监控配置系统

监控配置系统，作为整个平台的管理、配置系统，以及一些参数监控，监控整个平台的运行状态。

## 存储

拟使用Mysql，后续根据实际的存储格式和功能需求，决定是否使用nosql，（其实对于监控报警的信息格式比较适合nosql），使用mq和报警&监控系统交互，做数据的持久化操作。只做监控策略生成的监控报警和报警系统生成的报警信息的持久化，并不做不相关日志的存储（已有kibana）。

# 实现方案