**基于java的日志监控分析工具的设计与实现**

摘要

随着全球信息网络的发展与信息科学技术的兴起，全球无论是生活、经济、文化都被紧密连接。所有方便人们更好更快更便捷的生活方式与工具的产生，其思想以及概念都能够迅速通过先进的信息技术和网络传播为广大人民所熟知和接受。在经济国际化趋势的推动下，各行各业的操作模式、产业结构甚至于全部社会的管理方式体系都潜移默化的发生着革命性的改变。近年来，互联网信息科学技术渐渐被应用到企业/个人的销售管理工作中，为大中小型企业乃至个人提供强有力的方法支持、劳动力节省，预示着不可小觑的发展前景。

日志是软件的运行轨迹，我们能够通通过分析日志数据来窥探软件工作的状态、行为。日志信息往往记录在日志文件中，当软件出现异常信息时，文件中的异常日志信息对于外界来说是无感知的，需要维护人员主动查看日志文件从而去排查信息，但是，当维护人员发现异常的时候，代表软件的故障已经发生，而且已经导致功能失效。这个过程是被动的，低效的。由此，日志监控成为了提升软件稳定性和可用性的重要保障。

本篇论文首先阐明了本次设计的背景、课题现状和目的。之后对设计的需求从用例、性能、选型等多个角度进行了分析。本设计包括了三个大模块：监控模块，收集模块和查询模块。这三个模块分别用于监控应用服务器的日志文件并报警或上报、收集各个监控客户端上报的日志信息并存储和报警、提供web客户端供工具使用人员来查询日志数据和配置处理规则。之后，对以上三个模块从整体和局部上进行设计。在实现以上三个模块过程中，抽离出了大量的相似逻辑，从而抽象出了用于数据路由的Router组件和远程数据传输通信的Remote组件，在本章中阐明了组件的设计结构和设计细节，并提炼出了各个模块和组件的实现中的核心部分。最终，对这次设计过程做个人总结。

**关键字：** 日志监控，日志收集，模块化组件

# Java-based Design and Implementation of Log monitoring and analysis system

Author:Leihao Jin

Tutor:Peng Zhang

**Abstract**

Log is the track of the software, we can analyze the log of the software to know the state and behavior when the software is running. Log information often recorded in the log file, when the software is unusual, exception log information in the file to outsiders is not visible, so person in charge needs to take a view of the log file to troubleshooting information, however, when person in charge found abnormal, on behalf of the software has failed and has caused the failure. This process is passive and inefficient. As a result, log monitoring has become an important guarantee to improve the stability and availability of software.

This thesis illustrates the background, research status and purpose of this design first. Then analyze the demand from the use case, performance, selection and other aspects. This design consists of three modules: log monitoring module, log collection module and log query, statistical module. The three modules respectively for monitoring log files of the application servers and alarm or report, collecting log information of each client reported and storage and alarm, provide web client for the user to query log statistics information and configure the log processing rules. After that, to design of the above three modules from the overall and local. In achieving the above three modules, I draw out the large number of similar logic, and to abstract the router component of data routing and remote component of data transmission, in this chapter I illustrates the component-based design structure and design details, and extracted the realization of each module block and component of the core part. Finally, a personal summary of the entire design process is carried out.

**Keywords:** Log Monitor，Log Collection，Modularization

目 录

[目 录 I](#_Toc30491)

[第1章　绪论 1](#_Toc21554)

[1.1 课题背景 1](#_Toc20106)

[1.2 研究现状 2](#_Toc12889)

[1.3 目的及意义 3](#_Toc14170)

[第2章　系统分析 4](#_Toc23607)

[2.1 需求分析 4](#_Toc17004)

[2.2 用例分析 4](#_Toc26)

[2.3 性能分析 6](#_Toc27164)

[2.4 数据库选型 7](#_Toc10021)

[第3章　系统设计 9](#_Toc4688)

[3.1 模块设计 9](#_Toc28588)

[3.2 架构设计 9](#_Toc10732)

[3.3 数据库设计 10](#_Toc14178)

[3.3.1 elasticSearch索引设计 10](#_Toc205)

[3.3.2 mysql数据表设计 12](#_Toc7128)

[3.4 公共组件设计 13](#_Toc16062)

[3.4.1 Remote组件 13](#_Toc11303)

[3.4.2 Router组件 16](#_Toc2826)

[3.5 功能模块设计 17](#_Toc15547)

[3.5.1 日志监控客户端 17](#_Toc12686)

[3.5.2 日志收集服务器 20](#_Toc5778)

[3.5.3 日志的筛选、警报配置和日志的查询客户端 22](#_Toc14719)

[第4章　程序实现 24](#_Toc17783)

[4.1 Router组件实现 24](#_Toc17219)

[4.1.1 主要类关系 24](#_Toc9463)

[4.1.2 主要类关系 25](#_Toc28160)

[4.1.3 Sink实现 28](#_Toc2600)

[4.1.4 Channel实现 29](#_Toc19161)

[4.2 Remote组件实现 31](#_Toc6264)

[4.2.1 工作流程 31](#_Toc21239)

[4.2.2 tcp数据拆包 33](#_Toc538)

[4.2.3 消息的序列化/反序列化 33](#_Toc16063)

[4.3 日志的筛选、警报配置和日志的查询客户端实现 36](#_Toc14549)

[4.3.1 筛选、警报配置界面实现 36](#_Toc20465)

[4.3.2 查询界面实现 38](#_Toc31289)

[致 谢 39](#_Toc28030)

[参考文献 40](#_Toc14981)

第1章　绪论

1.1 课题背景

日志是程序的运行轨迹，它伴随着程序的运行而产生。日志记录下来了程序的所有执行过程，并按照预订的规格展现出来[1]。我们可以利用日志排查程序故障的原因，可以利用日志分析系统性能的瓶颈，可以利用日志统计用户的行为方式并对业务进行改进。

     传统软件行业对维护的实时性要求不高，当软件出现故障时，工程师上门对软件记录的日志进行分析，并对问题进行排查。对于互联网行业，对于程序的可用性需求较高，基本是利益相关。一分钟的系统故障可能会导致十万或者百万数量级的用户无法正常使用服务。因此，实时的对日志进行监控并根据监控信息及时排查故障来保证系统的可用性已经成为互联网企业必须具备的能力[2]。

     互联网企业系统的用户数量往往非常庞大，用户的网页浏览日志中隐性地包含用户的行为习惯，在每天记录的日志信息中，用户的行为信息成为了潜在的具有巨大价值的信息[3]。用户的行为信息能体现出用户的需求走向从而对产品进行新需求迭代，能够反映某功能是否符合用户期望从而对产品功能改进，能够反映系统使用者的爱好偏向从而向其推荐个性化的产品。因此，日志中隐含的用户行为信息包含着巨大的价值，对这些信息提取、分析，反过来改善产品的用户体验和确定下一步的发展方向，从而形成行为驱动开发的良性循环[4]。

     综上，对日志监控并分析能够及时得到软件系统的运行状态的反馈，做到及时响应，同时，对日志信息的分析能够得到用户行为信息，做到产品的纠正、改进、持续迭代[5]。日志监控分析系统对于软件系统的可用性、易用性、功能性都提供很好的支持[6]。

1.2 研究现状

目前，主流的日志监控分析工具或系统大致分为三类：

1. 基于SaaS服务的平台级的日志监控分析解决方案。市面上有很多此类服务平台，例如Splunk,Loggly等。这种服务平台集成了很多日志监控工具，分析模型，数据可视化工具等等，能够省去开发和维护的成本。不过，由于服务在第三方，所以日志信息需要上传到远程服务器中。这就导致了一下问题：

1）上报的日志信息可能涉及信息安全，需要做好信息屏蔽与过滤[7]。

2）当我们的应用负载高，单位时间产生的日志信息量巨大时（互联网企业中的主服务日志往往都是每天nT级别），上报过程会遇到瓶颈[8]。

3）另外，使用第三方服务，则导致对日志的监控、分析的方式，只能局限在服务平台提供的工具和服务中，不能根据实际业务进行定制。

除了以上问题，此类平台的服务是付费的，企业是否有必要通过付费购买其服务来省去自己开发维护的成本也是此类方案需要考虑的问题[9]。

1. 基于多个开源框架共同构建的日志监控分析系统。当企业通过第三方的日志服务来监控分析日志或第三方的服务在性能、功能上无法满足其需求，同时又想快速搭建一套日志监控分析系统时，企业往往根据业务场景，选择多个开源框架来搭建系统。因为开源框架的种类繁多，所以此类的解决方案也数不胜数。现在使用率比较高的解决方案有以下几种：

1）logstash+elasticsearch+kibana。这个方案集成了集群日志收集，日志存储及搜索和日志数据统计、可视化展示。此方案较为轻量，集成起来相对较容易，能够快速搭建[10]。

2）Flume+hdfs+kafka+storm。这个方案相比于上一种，拥有用更高性能，同时支持日志流的实时分发、计算[11,12,13]，当然需要的开发成本和维护成本也更高

3）日志文件监控工具。部署在与应用相同的服务器上的进程，同步监控日志文件，并及时报警。这个方案很轻量，适合对单台应用的监控，但不能解决应用服务集群日志的汇总和统计。

1.3 目的及意义

现在，我实习所在公司内部还没有推广自主研发的日志监控分析平台，所以在部分部门，还没有对日志进行有效的监控和分析。这导致了一下几个问题：

1. 因为没有实时监控线上应用的日志，导致出现系统异常无感知，无法迅速修复问题，从而可能引发严重的线上问题;

2. 日志数据含有大量信息，对其分析能够获取到很多有参考意义的数据：系统性能、业务使用量、用户行为等等。不对其加以利用，则浪费了这些资源，尤其对于流量巨大的互联网公司，更是巨大的损失。

由于以上问题，我决定设计并实现一套日志监控分析工具，服务于部门内部项目。这套工具能够实时监控项目产生的日志，当出现异常时能及时提醒负责人修复问题。同时，能够统计分析日志中的数据，用于改善产品。

第2章　系统分析

2.1 需求分析

根据需求，日志监控工具需要被设计成两种形态：单点监控和上报监控。

单点监控：将监控工具部署到应用服务器上，对日志文件实时监控，当监控到的日志信息需要报警时，直接向报警目标报警，不需要上报日志信息。

上报监控：是在应用服务器上部署监控客户端，实时监控日志文件，同时将日志信息上报到服务器端，在服务器端对所有应用的日志信息分析报警。服务器端的筛选、警报的规则可以由用户自由配置。同时能够从服务器端查询到上报的日志信息，并对日志分析。

2.2 用例分析

1. 单点监控用例

图2-1是单点监控的用例分析图。

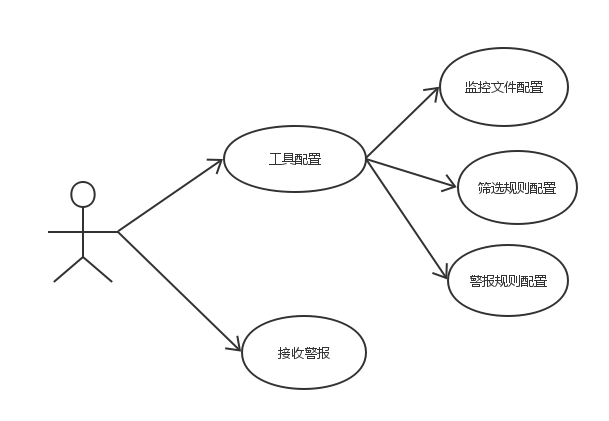


图2-1 单点监控用例图

1. 上报监控用例

图2-2是上报监控的用例分析图。

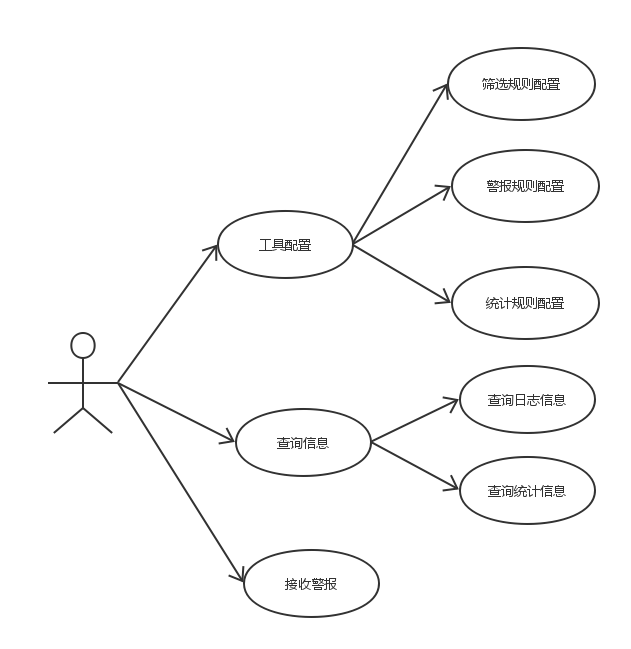


图2-2 上报监控用例图

2.3 性能分析

1. 可靠性

日志信息的实时监控对于用户量较大的业务系统是必要且重要的，如果监控失效，可能会导致异常的日志信息没有被发现，从而导致严重问题，可靠性对于日志监控工具来说，必须得到保证[14]。如果工具出现异常导致功能失效，需要尽可能使工具能够在非人为情况下重新恢复功能。

1. 易用性

工具的使用方式复杂，则会导致使用人员的学习成本和使用成本提高，不利于工具在组织内部推广。因此，友好、简单的工具使用方式很有必要，尽量能够通过简单的配置就能使用。

1. 时效性

异常发生的时间越短，解决异常的动作越迅速，则造成的损失就越小。因此，当异常日志信息出现时，需要及时的通过报警的方式来提醒维护人员，降低损失。工具的时效性是保证该工具意义的重要一点。

2.4 数据库选型

在整个日志监控管理工具中，需要持久化的数据并不复杂，分为两类：一类是工具的目标数据，即日志信息；一类是辅助工具运作的业务数据，包括对日志信息的筛选规则信息、报警规则信息，对日志信息查询规则信息和使用工具的用户信息等等。

对于日志信息的存储需要满足一下两点：

1. 支持高并发、高吞吐量。每台日志收集服务器会同时与数台日志监控客户端连接，并且每台日子监控客户端都会实时的上传日志信息到服务器。日志信息往往是低质量的文本信息，包含着一些荣誉的信息，每个应用每天会产生大量的日志，少则几十MB，多则若干GB，因此日志信息的持久化对存储的吞吐量要求较高。

2. 集成数据的存储与搜索。日志信息上报后，用户会通过日志查询客户端来对日志信息查询，因此日志信息的搜索必不可少。如果使用结构化数据库，其对数据的搜索没有支持，并且，若果通过条件查询的方式来实现数据搜索的话，结构化的数据存储方式在搜索性能上得不到保障。如果使用第三方的搜索引擎，则需要另外寻找一个非结构化数据库，并在搜索引擎与数据库之间进行二次开发，这样的话成本太高。因此，能够同时集成数据的存储和搜索的存储方案是此项目需要的。

根据以上的需求，我选择了elasticSearch作为日志信息的存储方案。原因如下：

1. elasticSearch支持多线程读写，在一定数量范围内，写入线程越多，其写入效率越高，满足了高并发的需求。
2. elasticSearch的默认写入速度限制是20mb/s，通过参数和存储优化，可以使写入速度提升至100-200mb/s甚至更高[15]，完全满足日志信息对存储的高吞吐量的需求。
3. elasticSearch既是一个搜索引擎，又支持数据的文档化存储，满足需求。对于业务数据，使用关系型数据库就可以满足需求，因此使用了免费的mysql。

第3章　系统设计

3.1 模块设计

整个项目分为三个模块：

1. 日志监控客户端。它能够对单点机器的应用进行日志监控。对于应用来说，感知不到监控客户端的存在，因此对于应用是无侵入、无感知的。客户端能够根据配置文件来筛选日志信息，并且能够根据配置信息来对需要报警的日志信息进行上报或者报警。
2. 日志收集服务器端。它是日志监控客户端的上报目的地，可以同时采集多台客户端上报的日志数据，并把日志信息持久化。同时，根据各个应用的配置信息来筛选收集的报警日志信息，对筛选出来的日志信息使用配置的报警方式进行报警。
3. 日志的筛选、警报配置客户端和日志的查询客户端。它提供给应用负责人员使用，日志收集后的筛选和警报的规则信息通过该客户端来配置。配置后，日志收集服务器端会及时更新新的规则来处理日志信息。它也提供日志信息的查询功能。

3.2 架构设计

整个日志监控分析工具的架构设计如图3-1所示。

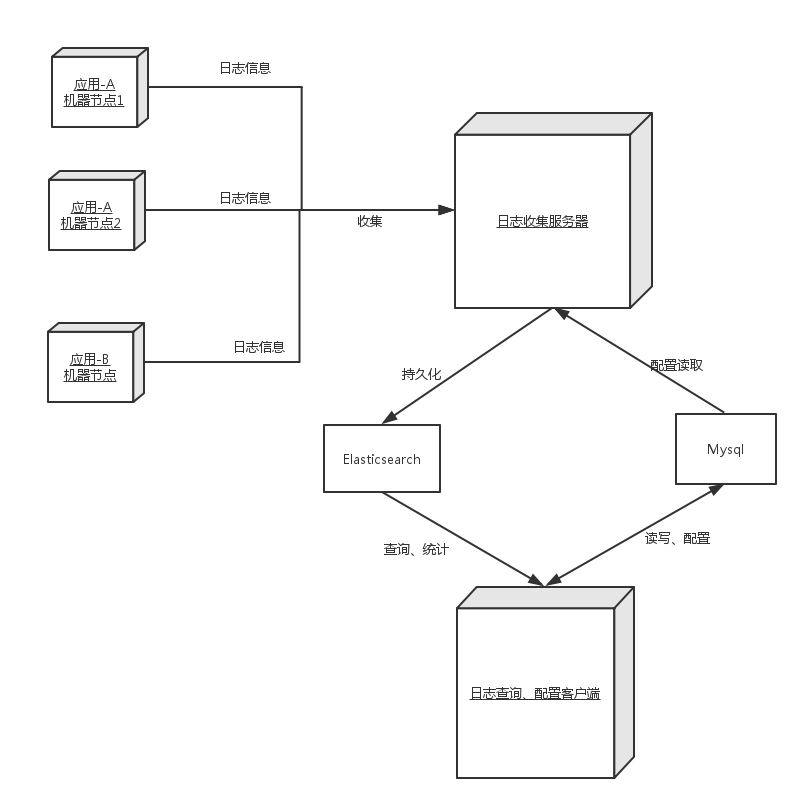


图3-1 架构图

3.3 数据库设计

### 3.3.1 elasticSearch索引设计

在elasticSearch中，构建一个用于存储日志信息的索引，命名为log\_index，以下是log\_index索引的mapping结构：

"mappings": {

"logMsg": {

"\_ttl": {

"enabled": false

},

"properties": {

"date": {

"format": "strict\_date\_optional\_time||epoch\_millis",

"type": "date"

},

"clientName": {

"type": "string"

},

"source": {

"type": "string"

},

"projectName": {

"type": "string"

},

"info": {

"type": "string"

}

}

}

}

log\_index索引的字段设计见表3-1。

表3-1 log\_index索引字段表

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 含义 |
| date | date | 日志信息的上报时间 |
| clientName | String | 客户端编号，格式为“应用名称-应用机器序列号”，例：hawkeye-1 |
| source | String | 日志信息来源的ip地址和端口号，格式为“ip地址：端口号”，例：10.58.47.74:8927 |
| projectName | String | 日志信息的应用名称 |
| Info | String | 日志信息 |

### 3.3.2 mysql数据表设计

1. t\_alert\_config表字段设计见表3-2。

表3-2 t\_alert\_config字段表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| id | Int | 主键 |
| project\_id | Int | 上报日志应用主键 |
| config | String | 上报日志应用的筛选、警报配置（具体配置规则在3.5.2.3节） |

2. t\_project表字段设计见表3-3。

表3-3 t\_project字段表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| id | Int | 主键 |
| name | String | 应用名称 |
| group\_id | Int | 应用所属小组的主键 |
| department\_id | Int | 应用所属部门的主键 |

3. t\_user表字段设计见表3-4。

表3-4 t\_user字段表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| uid | int | 主键 |
| group\_id | int | 用户所属小组的主键 |
| create\_date | date | 用户创建日期 |
| account | string | 登录账号 |
| password | string | 登录密码 |

4. t\_department表字段设计见表3-5。

表3-5 t\_department字段表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| id | int | 主键 |
| name | string | 部门名称 |

5. t\_group表字段设计见表3-6。

表3-6 t\_group字段表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| id | int | 主键 |
| department\_id | int | 小组所属部门主键 |
| name | string | 小组名称 |

3.4 公共组件设计

### 3.4.1 Remote组件

1. 功能描述

Remote组件封装了远程主机间的通信和信息交互，主要功能有：

1）提供了基于Netty框架的java NIO远程通信服务，通过异步的方式，来提升消息接收方的消息处理速度和消息接收性能。

2）具备自定义的远程数据交互协议，并通过二进制流对数据进行编码和解码，从而实现数据的序列化和反序列。

3）被通信方可以对通信方的连接请求进行过滤，通过通信方的ip、端口等信息，判断连接请求是否可以接受。

4）保持主机间的长连接，并具有超时检测机制。

2. 交互流程

1）客户端向服务器发送连接请求。

2）服务端收到连接请求，将客户端的ip和端口与白名单中的ip、端口进行比较，存在于白名单，则建立连接；否则，拒绝连接。

3）客户端向服务器端发送数据。

4）客户端每间隔一段时间向服务器端发送心跳包，服务器端收到心跳包后返回应答消息。如果客户端长时间没有收到应答消息，或服务器端长时间没有收到心跳消息，则判断连接断开。

3. 通信协议

协议消息分为两部分：消息头和消息体。 协议消息结构定义见表3-7。

表3-7 消息结构表

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 含义 |
| header | Header（java bean） | 136 | 消息头 |
| body | Object（java object） | 不定 | 消息体 |

消息头定义见表3-8。

表3-8 消息头结构表

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 含义 |
| code | int | 32 | 消息校验码 |
| length | int | 32 | 消息的总长度 |
| messageId | long | 64 | 消息的唯一id |
| type | byte | 8 | 消息类别 |

1. 通信协议编码规则

1）构建ByteBuf实体。

2）获取remoteInfo的校验码，以long型数据写入buf中。

3）获取唯一的消息id，以long型数据写入buf中。

4）获取消息类别对应的类别id，以int型数据写入buf中。消息类别设计见表3-9。

表3-9 消息类别表

|  |  |  |
| --- | --- | --- |
| 名称 | 标识 | 含义 |
| LOGIN\_RESP | 4 | 客户端连接请求消息 |
| LOGIN\_REQ | 3 | 服务器端接受连接请求  响应消息 |
| HEARTBEAT\_REQ | 5 | 心跳包消息 |
| HEARTBEAT\_RESP | 6 | 心跳响应消息 |
| MESSAGE\_REQ | 7 | 消息传送消息 |
| MESSAGE\_RESP | 8 | 消息接受响应消息 |

5）计算整个消息的总长度，以int型数据写入buf中。

5. 通信协议解码规则

1）构建NettyMessage实体。

2）读取64位长度数据，与remoteInfo消息校验码比对，如果符 合则判定为正确消息体。

3）读取64位长度数据，写入NettyMessage实体的messageId属性。

4）读取8位长度数据，写入NettyMessage实体的type属性。

5）读取32位长度数据，计算出消息体长度=总长度-消息头长度。

6）读取消息体长度数据，写入NettyMessage实体的body属性。

### 3.4.2 Router组件

1. 功能描述

Router是我封装的一种数据路由组件。它实现了数据的接收、数据的缓存和数据向外分发。无论是日志监控端对日志信息的实时采集，还是日志收集服务端的接收远程客户端上报的日志信息的过程，都做了相同的动作，即从日志信息源接受信息，然后将日志信息发送到目的端。日志监控端的信息源为日志文件中新增的日志文本，目的端为远程的日志收集服务端；日志收集端的信息源为远程客户端上报的日志信息，目的端则有两个：负责持久化日志信息的数据库以及负责过滤和警报的组件。在接收和发送之间，则构建了缓冲管道，因此，即使发送数据的速率低于接收数据的速率，也不会影响整体的性能。

 将以上共同的逻辑和结构抽离出来，形成了Router组件，因此节约了开发成本，也使代码易于维护。无论是在远程主机之间，还是在同一个主机中，都可以使用Router来根据业务需要进行自由组合，实现数据的远程/本机路由。

1. 结构设计

一个Router节点分为三部分：

1）Source：数据来源端。

2）Channel：数据缓冲管道。

3）Sink：数据发送端。

图3-2是Router节点结构设计图。

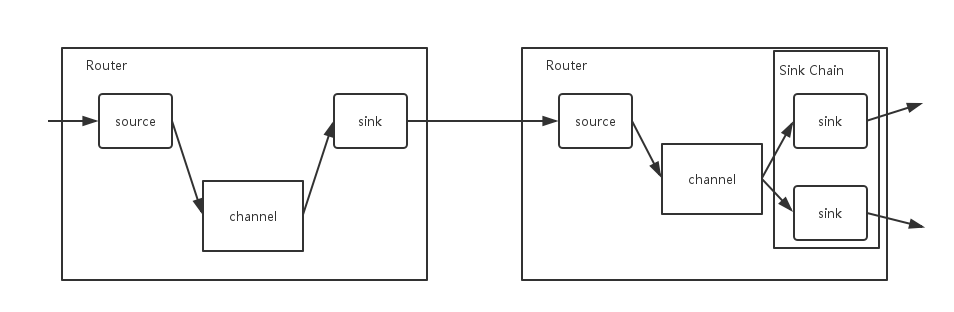


图3-2 Router节点结构设计图

1. 数据流向

1）在source中，数据从目标源流入，并将数据放入缓冲管道channel中。目标源可以是一个文件、一个缓冲区或者是另一个Router节点。

2）数据从source放入channel中，channel的容量根据实际情况来设定。缓冲在channel中的数据等待sink的提取，如果channel为空，则循环等待；如果channel非空，则将数据流入sink。

3）sink不断从channel中提取数据，并将数据发送到目的端。目的端可以是一个文件、一个缓冲区、一个网络邮箱或者是另一个Router节点。sink也可以是一个sinkChain链条，包含多个sink，从channel中获取的数据会分别发送给链条中的每个sink。

3.5 功能模块设计

### 3.5.1 日志监控客户端

1. 功能描述

日志监控客户端是部署在目标应用程序所在主机中的客户端，用来实时监控目标日志文件，动态获取日志信息并根据配置来筛选出目标日志信息并对其执行不同的操作：发送警报邮件或者上传到日志收集服务器。

2. 结构设计

日志监控客户端的结构设计如图3-3所示。

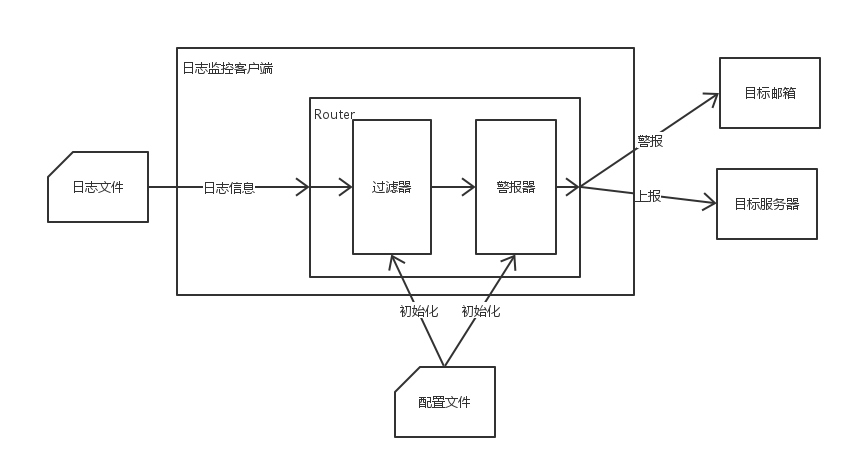


图3-3 日志监控客户端结构图

3. 客户端配置规则

日志监控客户端的运行方式通过客户端外部的配置文件来制定，配置文件放置于客户端同一目录下，使用json语法来描述配置项。以下为示例配置：

{

  "file\_path": "/hawkeye/client/logs/app.log",

  "chain": [

    {

      "type": "level",

      "value": "error"

    },{

      "type": "keyword",

      "value": "second"

    }

  ],

  "alert": {

    "type": "email",

    "value": "[303548901@qq.com](mailto:303548901@qq.com)"

  }

}

配置包含三部分：

1）file\_path：监控的日志文件路径名。

2）chain：日志信息的过滤器链。

过滤器链配置是一个数组，数组中的每个项为过滤器的配置项，type为过滤器类别，value为类别对应的配置参数。过滤器配置选项见表3-10。

表3-10 过滤器配置选项表

|  |  |  |
| --- | --- | --- |
| 类别 | 参数 | 说明 |
| level（日志级别过滤器） | 日志级别 | 筛选出高于参数日志级别的日志信息，日志级别由低到高分别为：trace、debug、info、warn、error、fatal |
| regex（正则匹配过滤器） | 正则表达式 | 筛选出与正则表达式匹配的日志信息 |
| keyword（关键词过滤器） | 关键词 | 筛选出包涵关键词的日志信息 |
| blank（空白过滤器） | 无 | 全部通过，不做任何筛选 |

2）alert：日志信息报警器。

报警器的配置决定了筛选出来的日志信息的去处，有两种可选择的去处：一种是直接通过通讯手段来进行报警，即客户端对目标日志文件单点监控，不通过日志收集服务器；第二种是上报到日志收集服务器，由服务器来做后续的处理。配置项中，type为报警器类别，value为类别对应的配置参数。报警器配置选项见表3-11。

表3-11 报警器配置选项表

|  |  |  |
| --- | --- | --- |
| 类别 | 参数 | 说明 |
| email（邮件报警器） | 邮箱地址 | 以邮件的方式，将筛选出来的日志信息发送到目标邮箱 |
| upload（上报报警器） | 日志收集服务器ip:端口号 | 将筛选出来的日志信息上传到日志收集服务器 |

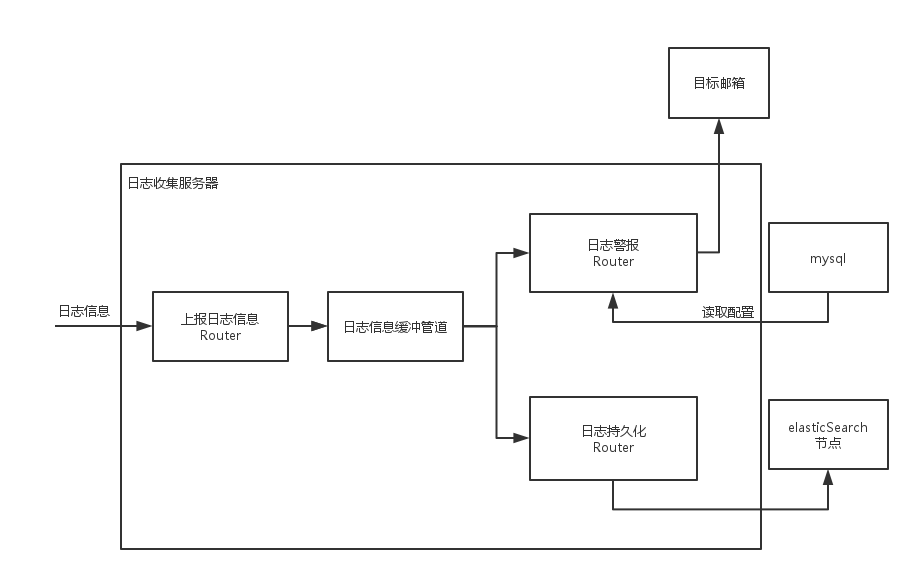
### 3.5.2 日志收集服务器

1. 功能描述

日志收集服务器接收从各个日志监控客户端传送上来的日志信息，将日志信息持久化，同时，对日志信息进行筛选，如果出现目标日志信息，则发送警报。

2. 结构设计

整个日志收集服务器大致分三部分：接收客户端上传数据的部分，持久化数据的部分，根据配置对数据筛选、警报的部分。这三部分的主要逻辑具有共通的部分，即获取数据并分发数据。因此对这部分的逻辑做一层抽象，产生了Router组件。日志收集服务器的结构设计如图3-4。

图3-4 日志收集服务器结构设计图

3. 过滤、警报配置规则

日志收集服务器对日志信息的筛选和警报方式由对应的配置决定。每个的项目会有一个或多个配置，服务器的筛选器和警报器根据配置初始化。以下为示例配置：

{

  "name": "异常提醒",

  "filter": [

    {

      "type": "level",

      "value": "error"

    },

    {

      "type": "keyword",

      "value": "serious exception"

    }

  ],

  "alert": {

    "type": "email",

    "value": "[123@qq.com](mailto:123@qq.com)",

    "template": "线上出现严重异常!请立即排查!"

  }

}

配置分为三部分：

1. name：配置的标注名称。由于同一个应用会对应一个或多个配置，所以使用这个值来做区分标注。
2. filter：过滤器链配置。具体配置选项同表3-10。
3. alert：警报器配置。具体配置选项同表3-11。

### 3.5.3 日志的筛选、警报配置和日志的查询客户端

1. 功能描述

日志的筛选、警报配置和日志的查询客户端是提供给用户用来对应用的日志信息筛选规则和警报规则进行配置的web客户端，同时可以查询已经上报的日志信息数据，是日志收集客户端的窗口。

2. 结构设计

日志的筛选、报警配置和日志查询客户端的结构设计如图3-1。

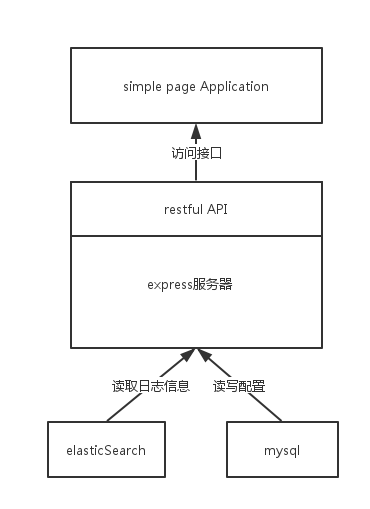


图3-1 日志的筛选、警报配置和日志查询客户端结构图

3. 技术选型

考虑到开发成本和开发效率，对于服务器选用了nodejs的express框架来搭建，对外暴露REST风格的API来操作elasticSearch和mysql中的数据，用于对数据的增、删、改、查。

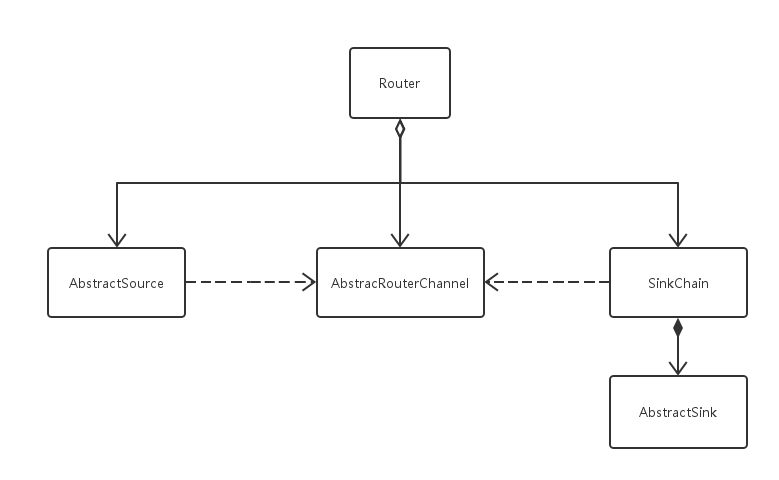
同样考虑到前端页面的开发成本和效率，使用了AngularJS框架来搭建前端应用。由于AngularJS中集成了依赖注入、路由、MVC等特性，非常适合本客户端的需求。前端与后端服务器交互都是通过restful API，简化了前后端的交互方式，实现了前后端分离。

第4章　程序实现

4.1 Router组件实现

### 4.1.1 主要类关系

Router组件的主要类的类图如图4-1所示。

图4-1 Router组件类图

Router组件的核心类是Router类，整个组件都是有Router类整合到一起并发挥总用，它是面向外部的门店类，所有与Router组件的交互都通过Router类。Source是Router节点的数据接收端，AbstractSource是Source的抽象类，继承了Thread类，封装了Source中监听数据、获取数据、把数据放入缓冲管道的主要逻辑，并提供给业务相关的Source来根据业务进行扩展。AbstractRouterChannel是Channel的抽象类，它是一个容器，用来缓冲由Source传来的数据并传给Sink，它内部封装了操作容器的方法，同时封装了默认的Queue，业务相关的Channel可以通过扩展AbstractRouterChannel抽象类来使用不同的Queue实现，从而满足不同的业务场景。AbstractSink是Sink的抽象类，继承了Thread类，封装了Sink中监听缓冲管道中数据、发送数据到目的端的逻辑，提供给业务相关的Sink实现类来实现业务相关扩展。SinkChain类整合了Router节点中的所有Sink，将所有Sink构建成一条Sink链，实时监听缓冲管道，把数据分别发送给每一个Sink实例。

### 4.1.2 主要类关系

AbstractSource中的抽象方法source（）由不同的实现类的具体实现，从而实现不同的业务逻辑。其他线程等公共逻辑则封装在AbstractSource类中。Source的工作流程图见图4-2。

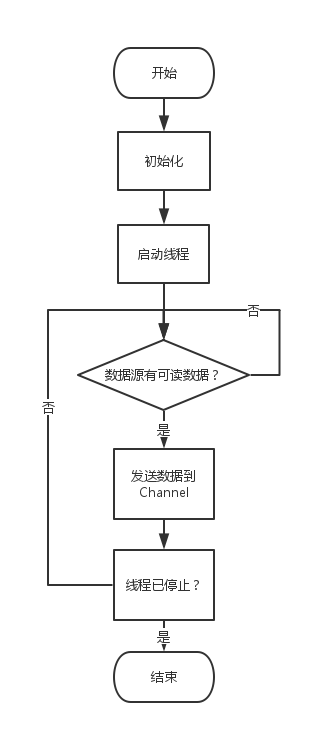


图4-2 Source工作流程图

1. 在程序启动时，首先根据配置来初始化Source成配置对应的Source实例。

2. 启动Source线程。

3. 监听目的数据源是否有可读数据（监听的数据源类型和具体的监听逻辑在AbstractSource类中的source方法中实现）。如果数据源没有数据，重复步骤3；如果数据源有数据，将数据发送到Channel。

4. 判断线程是否停止。如果是，结束source的监听；如果否，重复步骤3。

AbstractSource的具体实现类通过继承AbstractSource并实现source（）抽象方法来适应不同的业务逻辑。表4-1是根据设计需要而实现的AbstractSource的实现类。

表4-1 AbstractSource实现类表

|  |  |  |  |
| --- | --- | --- | --- |
| 类名 | 构造函数 | 参数说明 | 功能 |
| ChannelSource | ChannelSource(AbstractRouterChannel<T> channel, AbstractRouterChannel<T> sourceChannel) | sourceChannel：目标数据源。  channel：数据存储缓冲管道。 | 从目标数据管道中获取数据。 |
| NettySource | NettySource(final AbstractRouterChannel<RemoteInfo> channel, int serverPort) | serverPort：监听的目标端口。  channel：数据存储缓冲管道。 | 监听目标端口，获取上传到这个端口中的数据。 |
| FileSource | FileSource(String fileName, AbstractRouterChannel channel) | filename：目标文件。  channel：数据存储缓冲管道。 | 从目标文件中实时获取新增数据。 |

### 4.1.3 Sink实现

AbstarctSink类中的sink()抽象方法由具体的业务相关sink类来实现，其他的共同逻辑由AbstractSink类封装，sink的工作流程图见图4-3。

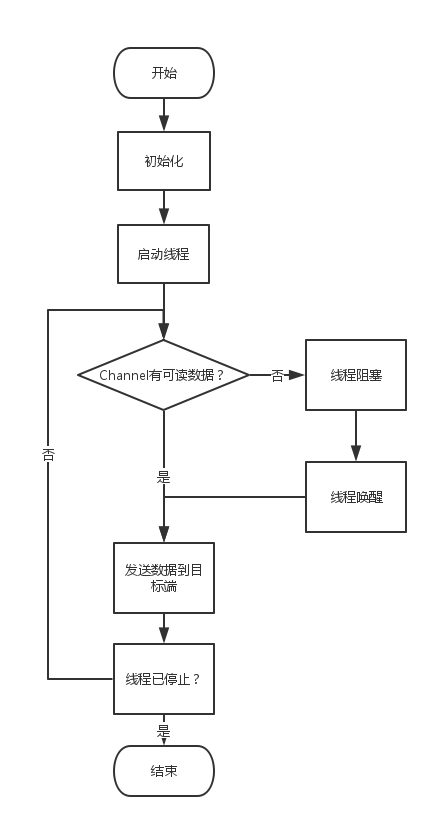


图4-3 Sink工作流程图

1. 在程序启动时，首先根据配置来初始化sink成配置对应的sink实例。
2. 启动sink线程。
3. 监听Channel中是否有可读数据。如果没有，线程被阻塞，直至数据到达时线程被唤醒。
4. 将读取到的数据发送到目的端（具体的目的端类型和发送方式在AbstractSink类的sink方法中实现）。
5. 判断线程是否停止。如果是，结束sink的监听；如果否，重复步骤3。

AbstractSink的具体实现类通过继承AbstractSink并实现source（）抽象方法来适应不同的业务逻辑。表4-2是根据设计需要而实现的AbstractSink的实现类。

表4-2 AbstractSink实现类表

|  |  |  |  |
| --- | --- | --- | --- |
| 类名 | 构造函数 | 参数说明 | 功能 |
| ChannelSink | ChannelSink(AbstractRouterChannel sinkChannel) | sinkChannel：目标数据管道。 | 将数据送入目标数据管道。 |
| EmailSink | public EmailSink(String emailAddress) | emialAddress：目标邮箱地址。 | 将数据发送至目标邮箱。 |
| ESSink | ESSink(String index, String type) | index：目标 | 从目标文件中实时获取新增数据。 |

### 4.1.4 Channel实现

AbstractRouterChannel类是Channel的基本实现类，其中包含了两个主要类对象：FilterChain和RouterQueue。FilterChain是过滤器链，用来筛选出目标数据，筛选后的数据存入RouterQueue。RouterQueue是阻塞队列，当队列为空时，从队列中取数据的线程会被阻塞直至队列中加入了新的数据。Channel流入数据的处理过程如图4-4所示。

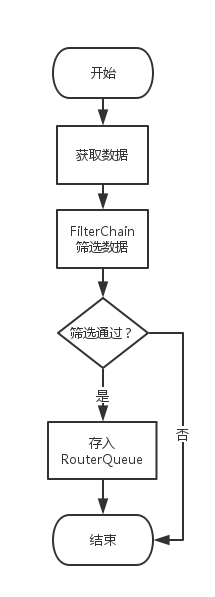


图4-4 Channel数据流入流程图

1. channel从外界获取数据。

2. 使用FIlterChain对数据进行筛选。如果筛选通过，把数据存入RouterQueue。

Channel流出数据过程如图4-5所示。

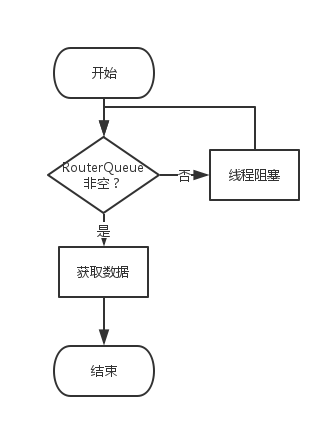


图4-5 Channel数据流出流程图

判断RouterQueue中是否非空，如果非空，从queue中读取数据；如果为空，读取线程阻塞，直至queue中有新数据，重复第一步。

4.2 Remote组件实现

### 4.2.1 工作流程

Remote组件的工作流程如图4-6所示。

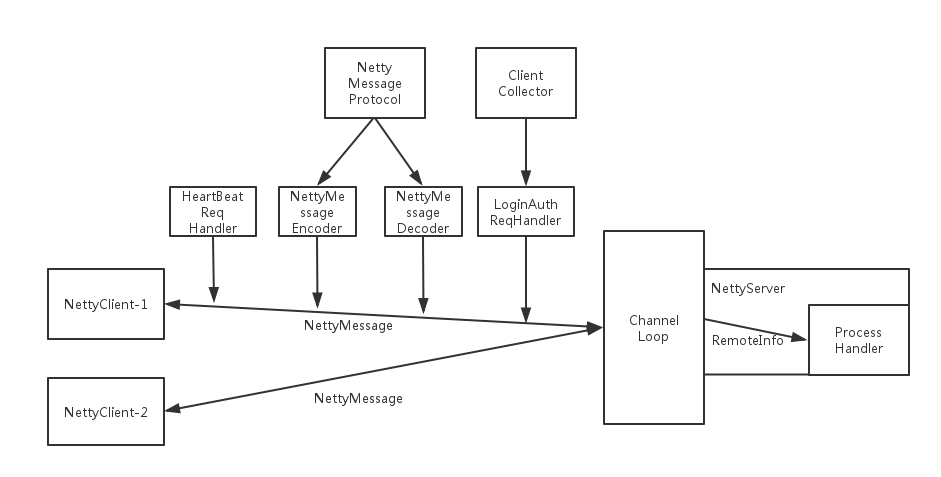


图4-6 Remote组件交互图

Remote组件的核心类为NettyClient类和NettyServer类。NettyClient类是提供给客户端使用的用来与服务器连接并传输数据的类，NettyServer类是提供给服务器端用来与客户端建立连接并接收数据的类。在每一个NettyClient与NettyServer之间都会建立一条管道，管道中传输着NettyMessage类的实例，并且在传输过程中，会有多个 Handler对数据进行处理。

Remote组件的具体工作流程：

1. NettyServer启动，开始监听目标端口。

2. NettyClient启动，向目标ip地址和端口发送tcp连接请求。

3. 连接请求经过LoginAuthReqHandler，LoginAuthReqHandler根据白名单来判断此连接请求是否合法。如果合法，建立tcp连接，并将NettyClient的ip地址、端口等信息存入ClientCollector中；否则，拒绝连接，不再向下进行。

4. 建立连接后，NettyClient的HeartBeatReqHandler开启，每隔5秒向NettyServer发送一个类型为HEARTBEAT\_REQ的NettyMessage（消息类型设计在3.4.1.3节）。业务数据也通过NettyClient向NettyServer发送。

5. 从NettyClient发送出的数据，都会经过NettyMessageEncoder来进行序列化，然后经过NettyMessageDecoder来进行反序列化，到达NettyServer。

6. NettyServer收到数据后，首先判断NettyMessage的消息类型：如果是HEART\_REQ数据，则向对方发送回一个HEART\_RESP来通知对方收到心跳包；如果是MESSAGE\_REQ数据，则传递到ProcessHandler的handler()方法中对数据进行下一步的业务处理，并返回一条MESSGE\_RESP数据通知NettyClient成功接收到此条数据。

7. NettyClient收到回应信息，判断NettyMessage的消息类型：如果是HEART\_RESP数据，则代表连接正常，若长时间未收到HEART\_RESP数据，代表服务器宕机或者连接断开，此时重新连接NettyServer，直至成功；如果是MESSAGE\_RESP数据，代表数据成功到达NettyServer，若长时间未收到MESSAGE\_RESP数据，重新发送此条NettyMessage。

### 4.2.2 tcp数据拆包

在NettyClient和NettyServer之间传输的NettyMessage是非定长的数据，所以，无法通过固定的长度值来对tcp数据包进行拆包。由于本设计中制定的通信协议的消息头定长，因此可以通过获取消息头中的消息长度字段的数据从而根据消息长度来对tcp数据包进行拆分。

在Netty中提供了一个工具类LengthFieldBasedFrameDecoder，这个类封装好了获取长度字段并根据消息长度拆分数据包的逻辑，因此在拆包模块使用这个工具类。通过以下语句来构建LengthFieldBasedFrameDecoder的实例：

new LengthFieldBasedFrameDecoder(1024 \* 1024, 4, 4, -8, 0);

以上5个入参分别代表最大数据包长度为1024\*1024、长度字段的偏移量为4bit、长度字段的数据长度为4bit、向前补偿8bit的数据、将前0bit的数据从数据包中删掉。

### 4.2.3 消息的序列化/反序列化

协议数据的远程传输需要把数据序列化成二进制数据流，然后通过TCP链接发送到远端，然后将二进制数据流反序列化成原数据形式，序列化前和序列化后的数据形式是一致的。对java的序列化/反序列化有两种形式：一是把java对象实例转化成json串，然后把json串转化成二进制流；二是把java对象的内存实例直接转化成二进制流。相比于第一种的json串形式，第二种的效率更高，同时转化出来的二进制形式的内存实例数据在反序列化时能够得到更多的类信息和内存状态信息，更利于恢复成序列化前的状态，所以Remote中对数据的序列化/反序列化使用第二种形式。

鉴于以上，对NettyMessage的序列化/反序列化使用了Kryo框架来辅助实现，分别实现了序列化工具KryoEncoder和反序列化工具KryoDecoder。

KryoEncoder实现如下：

public class KryoEncoder {

//消息长度字段的占位符，由于序列化前不知道消息体的总长度，所以使用占位符填充ByteBuf中消息长度字段的位置，在序列化之后，将数据的长度写到此处。

    private static final byte[] LENGTH\_PLACEHOLDER = new byte[4];

    Kryo kryo;//Kryo实例

    public KryoEncoder() {

        kryo = new Kryo();

    }

    protected void encode(Object obj, ByteBuf out) {

        Output output = null;

        try {

            int lengthPos = out.writerIndex();//长度字段的游标位置

            out.writeBytes(LENGTH\_PLACEHOLDER);//设置长度字段占位符

            ByteBufOutputStream bos = new ByteBufOutputStream(out);

            output = new Output(bos);

            kryo.writeClassAndObject(output, obj);//使用Kryo对目标java对象序列化

            out.writeBytes(output.toBytes());//把序列化后的二进制流写入Output

            int bodyLength = out.writerIndex() - lengthPos - 4;//计算消息长度

            out.setInt(lengthPos, bodyLength);//把计算出的消息长度数据写入长度字段的位置

        } catch (Exception e) {

        } finally {

            output.close();

        }

    }

}

KryoDecoder实现如下：

public class KryoDecoder {

    private Kryo kryo;

    public KryoDecoder() {

        kryo = new Kryo();

    }

    protected Object decode(ByteBuf in) throws Exception {

        Input input = null;

        try {

            int objectSize = in.readInt();//读取消息数据长度

            ByteBuf objBuf = in.slice(in.readerIndex(), objectSize);//根据数据长度，获取数据的二进制流

            byte[] objArray = new byte[objectSize];

            objBuf.readBytes(objArray);

            input = new Input(objArray);

            Object obj = kryo.readClassAndObject(input);//使用Kryo将二进制流反序列化成java对象实例

            return obj;

        } catch (Exception e) {

            throw e;

        } finally {

            input.close();

        }

    }

}

4.3 日志的筛选、警报配置和日志的查询客户端实现

### 4.3.1 筛选、警报配置界面实现

1. 添加配置界面

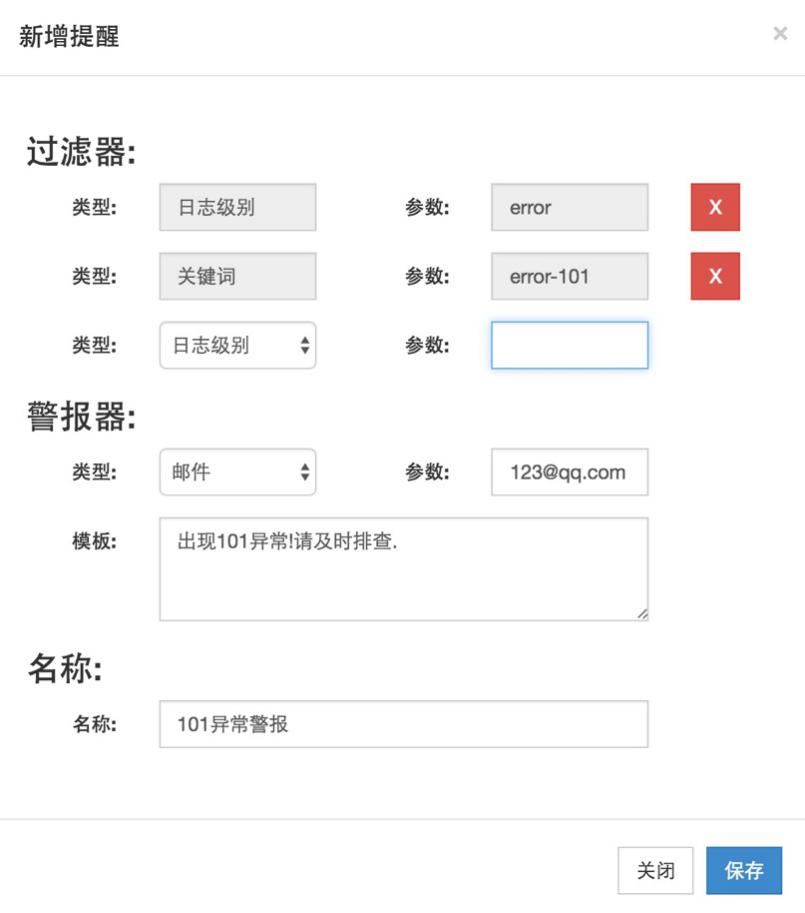


图4-7 添加配置项界面

添加配置界面如图4-7所示。在添加配置界面，用户根据需要添加一个或多个过滤器，可自行选择过滤器的类型和填写过滤器的参数。设置好过滤器类型和参数后，按回车键确认输入，配置好的类型和参数会变成不可编辑状态，并且后面出现删除按钮，同时界面中出现一条新的类型和参数配置框，供再次添加过滤器。用户可以格局需要选择一个警报器的类型，并填写警报器的参数和警报的模板。每个配置项可以对其命名，用来与其他配置项加以区分。

2. 查看配置界面



图4-8 查看配置项界面

查看配置界面如图4-8所示。进入客户端的查询模块，用户可以查看已经存在的配置项列表。最上方显示的是此配置项的名称。名称下方是过滤器和警报器的配置信息。如果想对此配置项做修改、删除操作，则可以点击下方的修改、移除按钮来实现。

### 4.3.2 查询界面实现

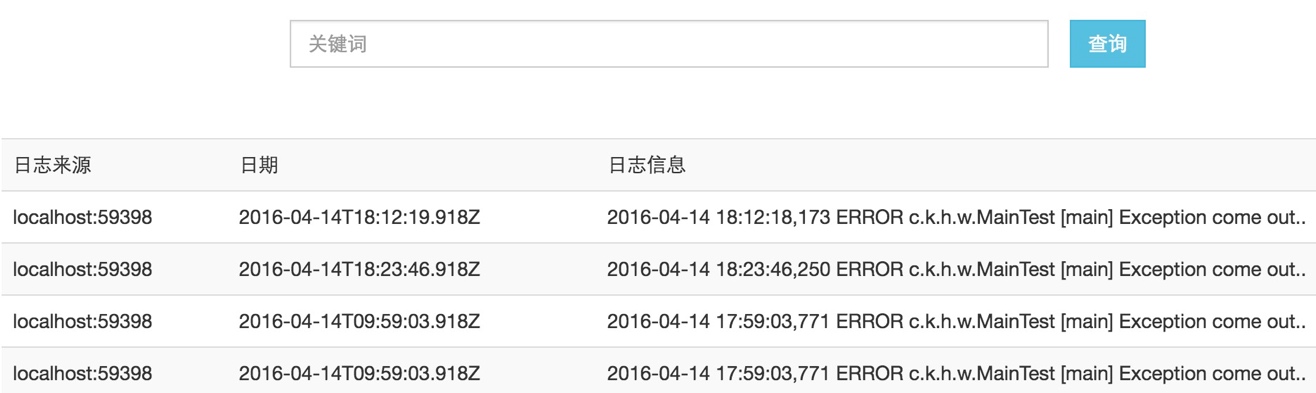


图4-9 日志查询界面

查询界面如图4-9所示。进入查询界面，用户可以通过输入要搜索的关键词对应用上报的日志数据搜索。关键词可以是日志来源的ip和端口、日期或者日志信息中的关键词。

致 谢

创造的过程是快乐的。程序员的梦想总是想要创造能够想到的一切，最终改变世界，我也是这样。从确定题目到分析需求，再到具体的设计和实践的过程，我很投入，也很开心！其实现在已经有很多日志监控的开源解决方案，它们有强大的社区，有完备的文档，有面对大量问题场景的具体解决方案，但我依然觉得自己来实现一个日志监控工具是有意义的。放下所有现成的解决方案，自己来思考该如何去优雅地实现它。在设计的过程中，不断地借鉴现有方案与自己的方案来对比，来重新思考之前的设计思路。对于他人来说这是一个重复造轮子的过程，但对于我来说一个去了解成熟框架中设计思想，并将它加入到我自己的设计中的过程，在这个过程中学习了很多在仅仅使用成熟框架中了解不到的东西，受益匪浅。

感谢xxxx….

**参考文献**

1. 孙赵平. 基于关联规则的web日志挖掘应用研究[D] . 安徽：安徽大学，2010.
2. 夏洁. 论互联网产品开发中的用户体验及可用性研究[D] . 浙江：浙江大学，2008.
3. 郭俊霞，高城，许南山，卢罡. 基于网页浏览日志的用户行为分析[J] . [计算机科学](http://ss.zhizhen.com/s?sw=JournalTitle%28%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%29&size=15&isort=0&x=0_920), 2014
4. ,3: 110-115.
5. 廖宇雷. 初探行为驱动开发[J] . 程序员,2008,10:76-78.
6. 周将铭. 以用户为中心的心产品开发方法研究与应用[D].北京：清华大学，2013.
7. 赵刚. 基于Web日志的网站可用性及用户行为可视化分析方法研究[D] . 山西：山西大学, 2007.
8. 麻光景. SaaS平台数据安全问题的研究[D] . 西安：西安电子科技大学，2014.
9. 申利民，张旭晖. 面向SAAS模式的日志架构设计与实现[J]. 计算机应用与软件,2011,12:57-59.
10. 悠虎.选择SaaS应用需要考虑的12大问题[DB/OL]. http://soft.chinabyte.com/11/12363011.shtml,2012-06-22/2016-05-10.
11. 摆渡者.ELK(ElasticSearch, Logstash, Kibana)搭建实时日志分析平台[DB/OL]. http://my.oschina.net/itblog/blog/547250,2015-12-18/2016-05-10.ju.alex.基于FLume的美团日志收集系统（一）架构与设计[DB/OL]. http://tech.meituan.com/mt-log-system-arch.html, 2013-12-09/2016-05-10.
12. dengjie. Kafka实战-实时日志统计流程[DB/OL]. http://www.cnblogs.com/smartloli/p/4581501.html, 2015-06-16/2016-05-10.
13. 郭俊.Kafka设计解析-Kafka背景及架构介绍[DB/OL]. http://www.jasongj.com/2015/03/10/KafkaColumn1/,2015-03-10/2016-05-11.
14. zhenjing.大规模分布式系统问题集及解决方案[DB/OL].：[2011-07-30]. http://www.cnblogs.com/zhenjing/archive/2011/07/30/distribute\_system.html
15. Rafal Kuc, Marek Rogozinski. ElasticSearch:可扩展的开源弹性搜索解决方案[M]. 北京：电子工业出版社，2015:145-182.