[1.1. 项目目的 2](#_Toc437273844)

[1.2. 核心功能分析 2](#_Toc1055325025)

[1.3. 技术难点分析 2](#_Toc1392823365)

[1.3.1. 日志发送采用哪种方式? 2](#_Toc1113109451)

[1.3.2. 已有Java程序如何接入本系统 3](#_Toc1202993787)

[1.3.3. 如何保证日志消息有序 3](#_Toc597719711)

[1.3.4. 如何实现实时日志浏览 3](#_Toc935272252)

[1.3.5. 如何备份及下载日志文件 4](#_Toc357265799)

[1.3.6. 如何降低耦合 4](#_Toc608274146)

[1.3.7. 如何监控系统运行状况 5](#_Toc1199205478)

[1.3.8. 如何处理宕机 5](#_Toc1086555539)

[1.4. 整体项目架构 6](#_Toc962948402)

[1.5. 技术选型 7](#_Toc1746876977)

[1.5.1. 开发环境 7](#_Toc757363655)

[1.5.2. 主要使用框架及用处 7](#_Toc756481718)

[1.6. 项目规范 8](#_Toc1077265784)

[1.6.1. 减少系统部署复杂度 8](#_Toc851623767)

[1.6.2. 高度自定义 8](#_Toc1651611542)

[1.7. 项目结构——maven 8](#_Toc2046444155)

[1.8. 数据库设计 8](#_Toc1643739059)

[1.8.1. 时序数据库 8](#_Toc1169732394)

[1.8.2. h2数据库 8](#_Toc1297232490)

## 项目目的

开发一款基于Java的日志采集系统，要求有日志存储、日志下载、实时日志浏览等功能

## 核心功能分析

* 日志生产——Java程序主动发送日志到指定位置
* 日志采集——系统通过某种手段收集程序发送的日志
* 日志备份——日志需要定时备份到文件
* 实时日志——需要有查看实时日志的功能
* 服务监控——查看服务运行情况

## 技术难点分析

### 日志发送采用哪种方式?

* 思路一：使用socket通信
  + 优点
    - 实现简单
  + 缺点
    - socket服务端横向扩展困难
    - 日志产生方和日志采集方耦合度过高
* 思路二：引入消息中间件作为通信桥梁，使用队列模式
  + 优点
    - 日志产生方和日志采集方解耦
    - 日志采集方可以横向扩展
  + 缺点
    - 采集的日志无法保证有序
    - 备份的日志文件依旧散乱

#### 解决方案

采用思路二

### 已有Java程序如何接入本系统

#### 解决方案

* 能够改动程序
* 系统提供基于slf4j的工具类，只需要引入工具，修改项目配置文件即可接入系统
* **前提：Java程序使用slf4j作为日志框架**
* 不能或不愿改动程序
* 系统提供基于文件监控的工具，只需要引入工具，修改工具配置即可接入本系统
* **前提：Java程序将日志输出到文件**
* **建议使用第一种方案**

### 如何保证日志消息有序

由于网络为不可靠资源，无法保证到达目的地顺序和发送顺序一致 且日志采集方采用横向扩展方案，代表同一段时间内的消息会被分散到多个机器，无法使用内存排序

#### 解决方案

引入时序数据库，日志采集方将采集到的日志存储到时序数据库，由时序数据库负责排序；日志采集方间隔一段时间再从数据库获取数据；以此保证在一定时间段内的日志消息有序。

### 如何实现实时日志浏览

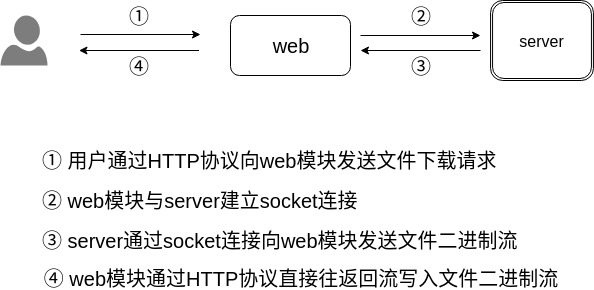
目前日志采用的是消息中间件发送，使用queue模式；一旦日志消息被消费，则不能再次利用 且web模块采用B/S架构，常规B/S架构为半双工通信，无法实现S端推送消息到B端

#### 解决方案

* 日志采集方消费日志后，再次将日志消息以topic模式发送给消息中间件
* web模块前端采用WebSocket与后端小链接
* web模块订阅消息中间件，一旦接收到日志消息，则通过WebSocket广播到前端

### 如何备份及下载日志文件

#### 解决方案

* 日志采集方定期从时序数据库抓取数据备份到本地服务器
* web模块通过socket与日志采集方通信，获取文件列表及文本流
* 

### 如何降低耦合

#### 耦合来源

* 对于消息中间件的依赖——如果实现中大量使用ActiveMQ独有代码，将会导致无法替换消息中间件
* 对于时序数据库的依赖

#### 解决方案

* 消息中间件依赖

采用工厂模式，定义消息中间件发送、接收等操作等接口规范；只要提供实现了该规范的依赖文件，即可实现消息中间件替换

* 时序数据库依赖

时序数据库遵循jdbc标准，只要代码中不涉及某个时序数据库独有的操作，即可轻松替换时序数据库

### 如何监控系统运行状况

#### 解决方案

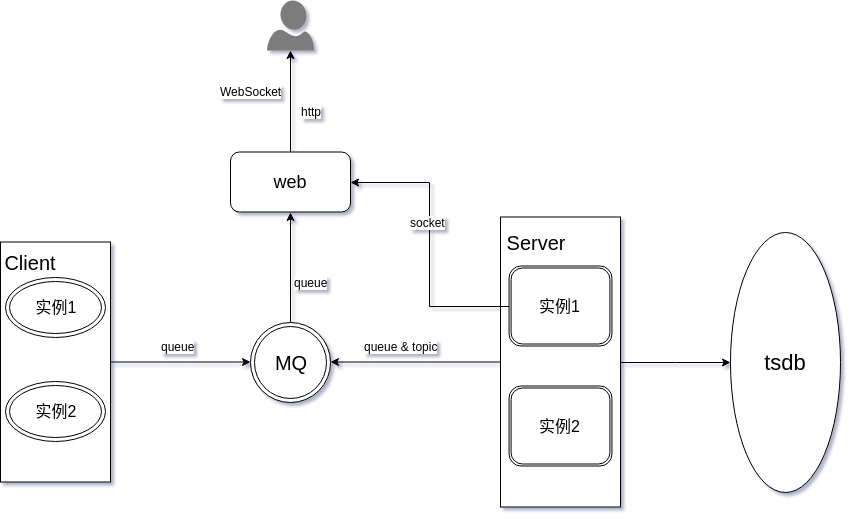
日志采集模块定时发送心跳包到消息中间件，web模块通过订阅消息查看日志采集模块状态

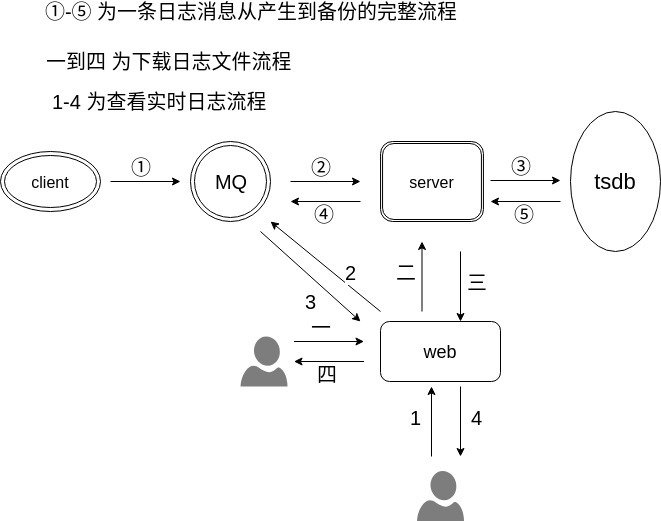
### 如何处理宕机

#### 解决方案

* 消息中间件宕机
* 消息中间件宕机将导致日志无法发送，此时日志产生方将会把日志缓存到本地，等待消息中间件恢复后重新发送；对其余模块影响不大，等待恢复即可
* 日志采集方宕机
* 日志采集方采用分布式架构，少数机器宕机对系统影响不大；同时web模块可监控到日志采集模块宕机，及时恢复即可
* web模块宕机
* web模块为非必需模块，宕机不会影响系统核心功能；建议采用额外手段监控web模块状态

## 整体项目架构





## 技术选型

### 开发环境

* deepin 15.9——操作系统
* Java1.8——程序运行环境
* idea2018——开发工具
* ActiveMq 15.8——依赖服务
* influxdb——依赖服务
* git——版本控制工具

### 主要使用框架及用处

* Spring 5.2 基础架构，所有核心模块基于Spring架构
* mybatis ORM，日志持久化
* Jetty 用于搭建嵌入式B/S架构
* Vert.x 用于实现socket通信
* slf4j+logback 日志库
* h2 嵌入式数据库
* junit 单元测试
* quartz 任务调度框架
* maven 依赖管理器
* thymeleaf 模板引擎，用于搭建B/S架构
* elementUI 前端UI框架
* Vue 前端数据绑定框架

## 项目规范

### 减少系统部署复杂度

* web模块不采用传统的war包方式，使用Jetty嵌入式开发，减少对外部Servlet容器的依赖，简化部署流程

### 高度自定义

* 所有模块禁止使用硬编码方式，重要配置项要求可在打包后正常配置

## 项目结构——maven

root  
|--client 为Java程序提供的日志发送工具  
|--server 日志采集模块  
|--web web模块  
|--mq-api 消息中间件规范模块  
|--activeMq 消息中间件规范的activeMq实现  
|--quartz 定时任务模块  
|--server 日志采集模块  
|--model 相关数据模块

## 数据库设计

### 时序数据库

#### log表

存储临时存储收集的日志信息

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| ip | tag | 日志生产者ip |
| tag | tag | 用来区分不同服务 |
| msg | field | 日志主体数据 |
| time | timestamp | 日志生产时间 |

### h2数据库

#### backup表

日志本机备份，用于Server模块

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| id | int | 主键 |
| path | varchar(255) | 日志文件所在绝对路径 |
| created | datetime | 日志文件创建时间 |
| start | datetime | 包含日志开始时间 |
| end | datetime | 包含日志结束时间 |
| size | int | 文件包含日志条数 |
| length | int | 文件字节数 |

#### server表

服务监控，用于Web模块

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 说明 |
| ip | varchar(15) | Server模块服务地址 |
| run | long | 已运行时长 |
| status | bool | 状况，true为正常运行 |
| file\_port | int | 用于文件传输的端口 |
| last | datetime | 上次心跳包发送时间 |