日志分析系统设计文档

版本：V0.1

修订记录

时间	修订人	版本	修订内容
2015-11-17	顺炽国	V0.1	初稿
			1.

目录

1. 概要

1.1. 背景

对现网应用程序进行日志采集，传送到日志分析系统进行处理分析，将结果存储到统计系统中，以报表的方式展现给开发、运维及管理人员查看。对分析结果进行监控，可以达到对系统健康度、故障率等实时告警，提升系统异常处理效率。

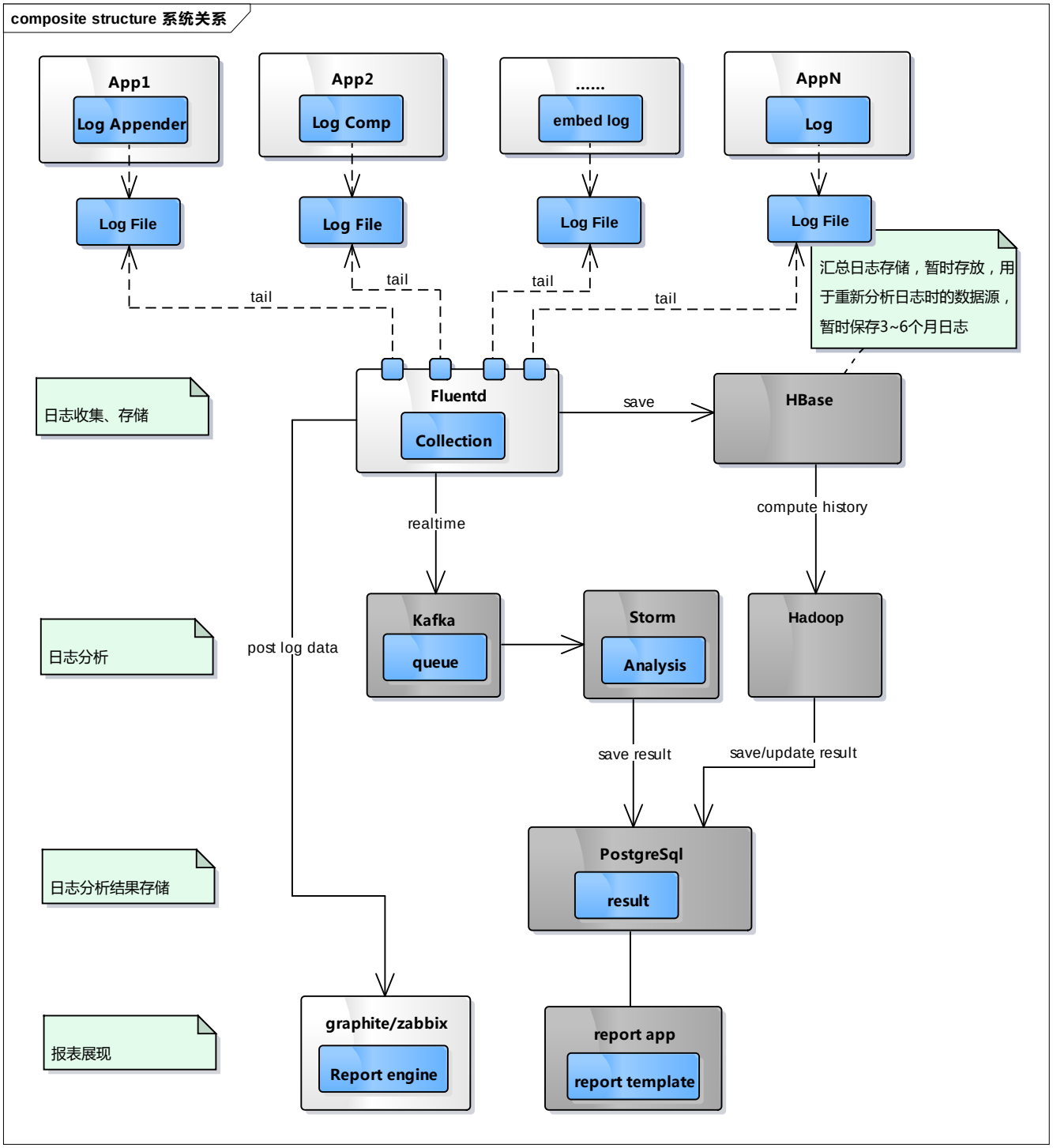
1.2. 目标

- 需要支持多种应用程序的日志数据格式采集；
- 约定日志记录格式
- 日志集中化存储
- 支持实时日志分析、统计
- 需要考虑后续业务功能扩展，如：日志格式，日志分析逻辑
- 需要支持分布式部署方案

1.3. 名词约定

2. 系统框架

2.1. 系统关系图



说明：

1. 系统分为五个部分，分别为：日志采集、日志收集存储、日志分析、分析结果存储及报

表展现

2. 日志采集第一期支持 4 种源输入，分别为桥接模式（借助第三方插件）、HTTP 接入（需要修改业务代码）、tail 文件监控输入、Tcp 接入（需要修改业务代码或借助第三方插件）
3. 日志存储暂时保存 3 个月内数据，超过时限的数据自动删除
4. 日志分析部分可以视进度情况进行删减，初期工作量较大的情况下可减化为只进行最简单的分析计算，优先实现质量指标数据计算
5. 日志分析结果存储到指定的结果数据中（Mongodb、redis 或 postgreSql）

采用开源框架
日志采集框架：fluentd(暂定)
原始日志存储：HBase
实时日志统计：Elasticsearch + grafana
日志分析框架：第一期自己实现，后续替换为 kafka 和 strom
分析结果存储：postgresql

第一期仅实现日志采集、转发及日志报表输出，日志存储、分析作为保留接口后续实现。

2.2.服务部署

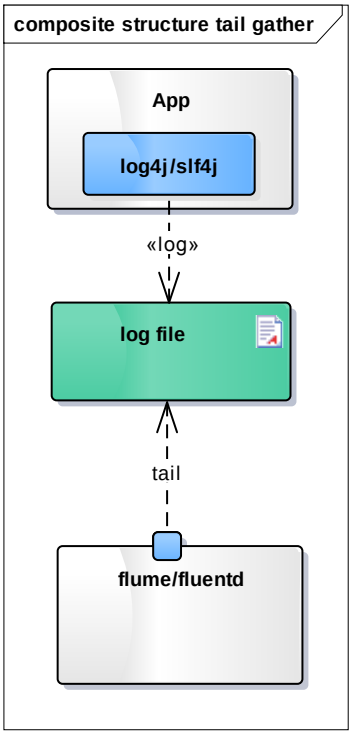
2.2.1. 部署架构图

2.2.2. 风险及规避方法

3. 系统方案

3.1. 日志接口约定

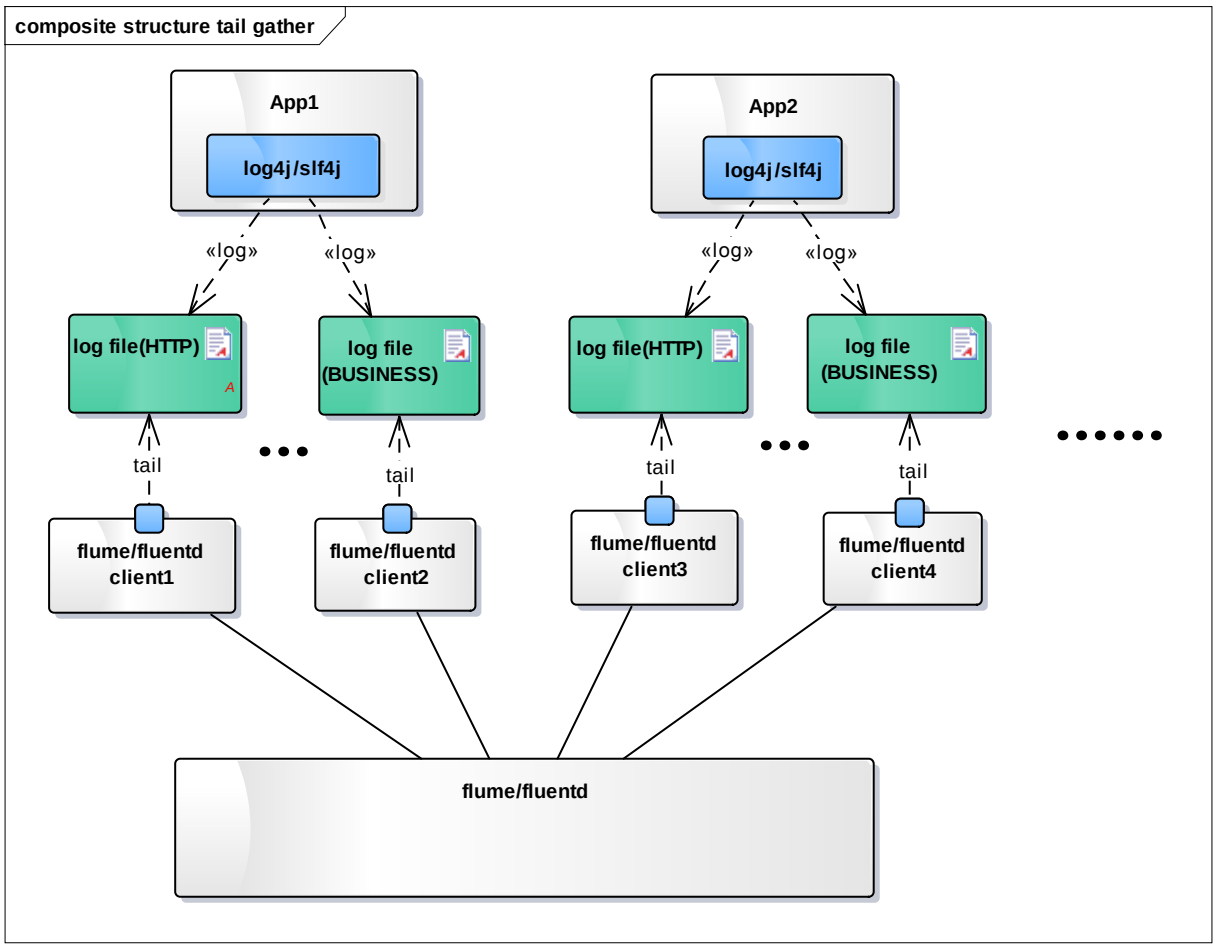
3.1.1. 文件方式(tail)



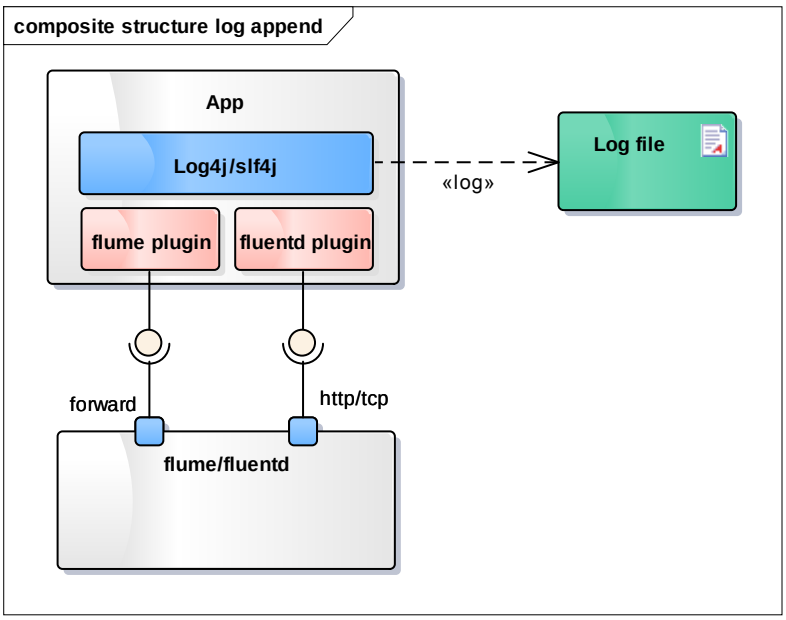
应用程序将需要导入的日志写到指定目录，日志采集程序配置一个采集源（source），使用 tail 命令作为输入源进行日志信息采集。

此方式程序可以各自使用自己的日志记录类库，只要遵循 3.1 约定的日志格式生成日志文件即可，相对改动较小。只是需要验证在日志文件发生变更（文件切换、删除、重建）等情况时，是否能自动跟踪日志变化。

如果在实际环境中使用，一个应用会生成多个需要采集的日志，那么情况会变成如下情况：



3.1.2. 桥接方式

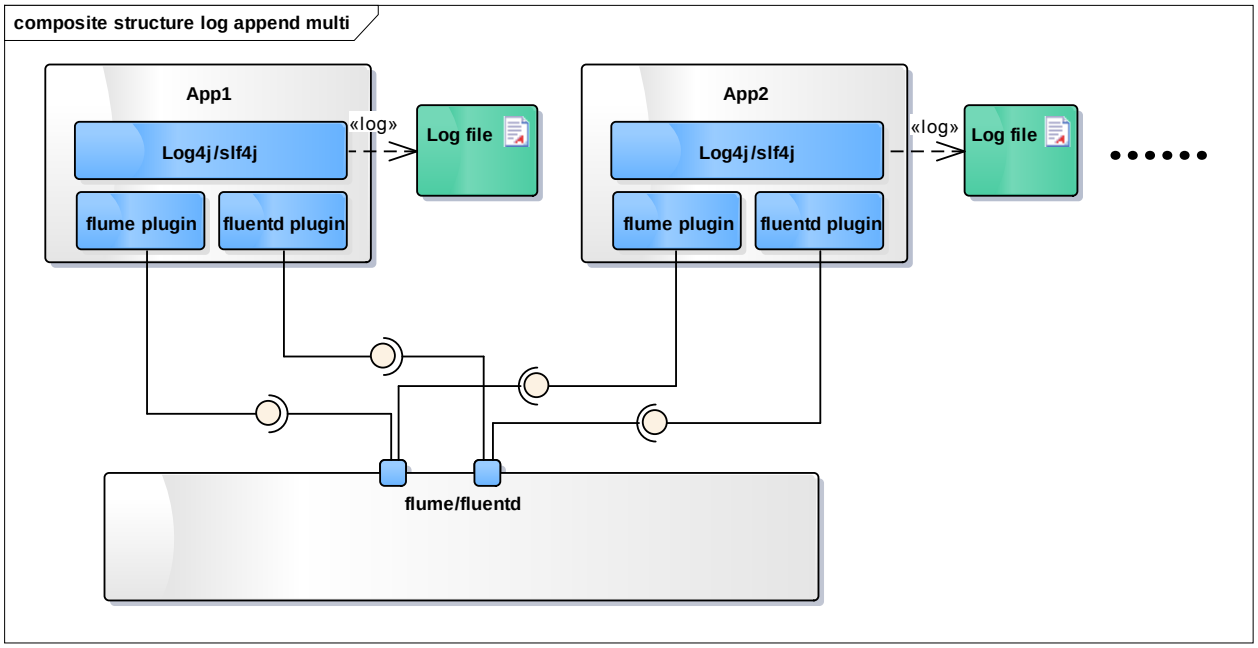


应用在使用 log4j、slf4j 之类的日志类库时，可以使用第三方的插件将日志内容桥接到日志采集程序中。

此类方法相对第一种方式来说，需要引入额外的桥接类库，且受限于使用类库的第三方插件支持，同时实现 3.1 中约定的日志格式记录。优点是不用修改代码即可实现日志的同步采集，不受限于生成的日志文件。

如果有多个应用存在，程序关系如下图：

方案 1（集中式采集）：



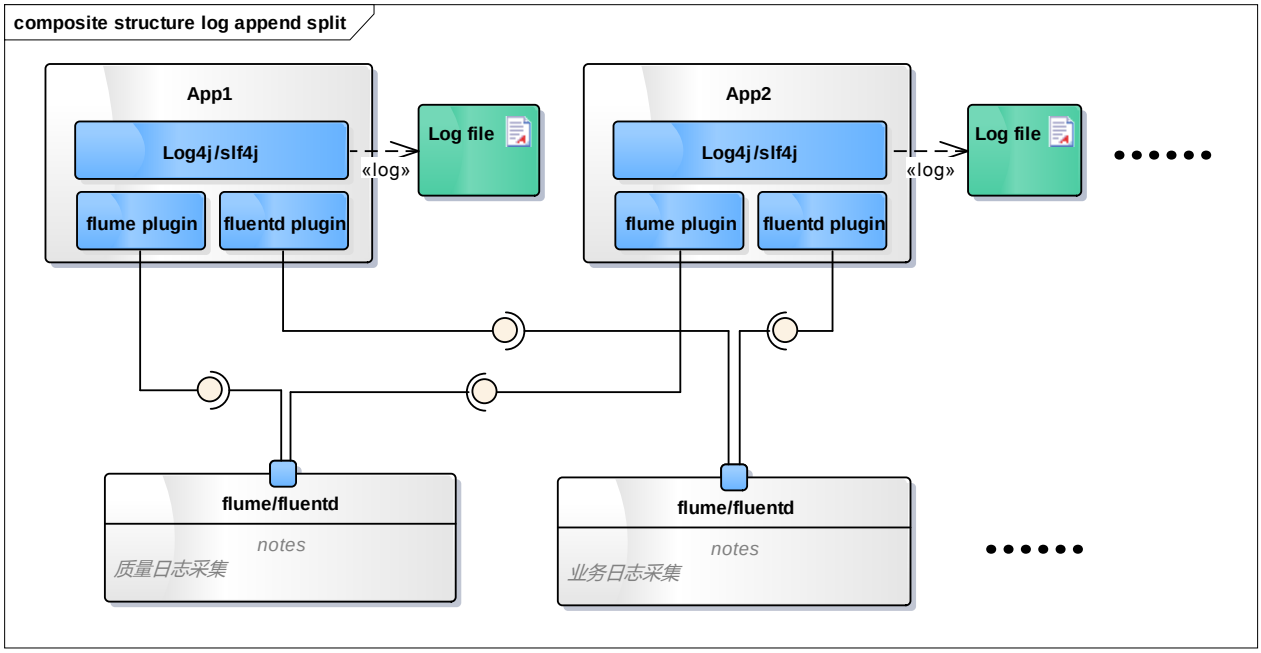
说明：

每台服务器部署一个日志采集服务，统一采集所有应用的日志类别信息，各个应用与日志采集服务通讯。

优点：部署简单、配置少

缺点：如果采集程序异常退出，则所有日志无法采集

方案 2（按类别采集）



说明：

每台服务所有应用的日志类别信息，各个应用与日志采集服务通讯。

优点：部署简单、配置少

缺点：如果采集程序异常退出，则所有日志无法采集

3.1.3. 代码嵌入方式

应用程序自己实现日志采集程序的通知接口（HTTP、TCP 等），主动将日志通知到日志采集程序中。

此种方法控制能力最强，可以自由定义日志的采集推送方式，缺点是实现工作量较大，需要自己实现各类采集通知接口。

方案对比：

功能	tail 方式	桥接方式	嵌入方式
程序修改	少	适中	较多
原始日志文件	依赖	不依赖	不依赖
对程序性能影响	无	由插件情况决定	由开发情况决定
多应用接口复杂性	复杂	少	少

建议使用方案为：

桥接方式（如果可用） > tail 方式 > 嵌入方式

3.2. 日志规范

3.2.1. 日志格式约定

日志文件每一条完整记录为一行，内容不允许换行，如果日志内容有换行的字符，需要去除此类不可见字符，格式如下：

Log line 1

Log line 2

……

Log line N

日志格式：

公共日志标记，保存了所有日志的通用信息，其定义如下：

REQUEST TIME| APPNAME| LOG TYPE| CLIENT IP:PORT| SERVER IP:PORT| ELAPSED TIME| COMMUNICATION TYPE| REQUEST DATA LENGTH | RESPONSE DATA LENGTH| DYNAMIC PARAMS

参数说明

序号	参数名称	参数说明
1	REQUEST TIME	请求访问时间，使用 UNIX 时间戳格式，精确到毫秒
2	APP NAME	日志输入 APP 的名称
3	LOG TYPE	日志类型，目前约定为：**ACCESS**
4	CLIENT IP:PORT	客户端请求 IP 地址和端口，格式为:**IP:PORT**
5	SERVER IP:PORT	应用服务所在的 IP 地址和端口，格式为:**IP:PORT**
6	ELAPSED TIME	请求消耗时间，单位为毫秒，表示整个请求进入到处理结束所消耗的时间
7	COMMUNICATION TYPE	通讯类型，目前约定为:**HTTP、THRIFT** 两类，后续视情况增加
8	REQUEST DATA LENGTH	请求数据大小，除去 HTTP 头或 TCP 协议头的实际内容长度,单位 byte
9	RESPONSE DATA LENGTH	响应数据大小，除去 HTTP 头或 TCP 协议头的实际内容长度,单位 byte
10	DYNAMIC PARAMS	动态定义参数，由 APP NAME、LOG TYPE 和 COMMUNICATION TYPE 两个参数而定，在质量项目中，仅考虑 LOG TYPE 和 COMMUNICATION TYPE 参数，详细定义请见：3.2.2，3.2.3 的定义

约定：

- 字段间使用字符“|”分隔
- 如果某个日志字段没有内容，可以将此字段内容设置为空，分隔符不能省略
- 动态参数由各个类别的日志自行定义，定义的字段分隔符可以自行约定，默认使用“|”符号分隔，由分析程序根据日志类别自动处理
- 日志内容中如果有“|”字符，暂时先替换成空白字符

3.2.2. HTTP 日志

当日志类型中 COMMUNICATION TYPE 值为 **HTTP** 时，格式如下：

METHOD| URL| STATUS CODE| QUERY ENTITY| CALL METHOD| BUSINESS CODE| RETURN CONTENT

参数说明

序号	参数名称	参数说明
1	METHOD	HTTP 请求方法，如：GET、POST、PUT 等
2	URL	请求的 URL 地址，不包含查询参数、域名或 IP 端口信息
3	STATUS CODE	请求响应状态码
4	QUERY ENTITY	请求主体，用于识别查询请求，由业务约定
5	CALL METHOD	请求执行的方法，请求执行的具体操作名称，由业务约定
6	BUSSINESS CODE	业务状态码，业务端返回的业务状态码
7	RETURN CONTENT	返回内容，用于定义请求问题

例如，下面是一个查询用户信息的接口调用日志：

1448245749586|account api|ACCESS|202.12.33.11:32210|192.168.30.10:8281|112|HTTP|0|12311|GET|/user/1|200|100001|findUserById|100|{userId:100001, username:"张三", Sex:1,……}

日志格式说明：

1. 对于某些没有业务代码的应用，默认返回两种状态，成功（0）和失败（1），也可自行约定状态码
2. 返回内容默认情况只返回 API 调用接口的返回内容（建议只返回前 50 个字符，后续字符使用省略号[……]代替），对于其它内容（二进制数据、HTML、超大文本等）则不填写，直接留空
3. 日志仅记录业务调用请求（如：API、服务接口、网页等）对于其它请求（图片、文件等）则不记录
4. 对于 CALL METHOD 字段，如果是 RESTful 风格接口，且没有直接对应的处理方法，那么此处可以填写 METHOD+URL 的名称，否则统一填写实际执行的操作方法名称。
5. 日志文件统一命名为 quality_access.log，每天结束生成前一天日志的备份文件，格式为:quality_access_yyyyMMdd.log。
   如：假设当天的时间为 2015-11-25，那么当天的日志文件名为:quality_access.log，当到了 2015-11-26 日，前一天的日志文件自动备份成:quality_access_20151125.log，同时创建:quality_access.log 存储 2015-11-26 的日志文件，以此类推。

~~3.2.3. Thrift~~ 日志

~~当日志类型中 COMMUNICATION TYPE 值为 THIRFT 时，有如下格式：~~

~~*METHOD|STATUS CODE|QUERY ENTITY|CALL METHOD|BUSINESS CODE|RETURN CONTENT*~~

~~参数说明~~

~~序号~~	~~参数名称~~	~~参数说明~~
~~1~~	~~METHOD~~	~~Thrift 请求调用的方法名称~~
~~2~~	~~QUERY ENTITY~~	~~请求主体，用于识别查询请求，由业务约定~~
~~3~~	~~CALL METHOD~~	~~请求执行的方法，请求执行的具体操作名称，由业务约定~~
~~4~~	~~BUSSINESS CODE~~	~~业务状态码，业务端返回的业务状态码~~
~~5~~	~~RETURN CONTENT~~	~~返回内容，用于定义请求问题~~

3.3. 日志存储

集中采集到的日志，根据日志定义格式，合并存储到数据库中，可选存储方案为：HBase、MongoDB 及 Redis，考虑到后续日志数量剧增的情况，建议使用 HBase 来进行数据存储，存储时间限制在 3~6 个月内，方便进行历史日志数据的再次分析。

超过半年的日志数据则直接删除掉。

在第一个版本可以暂时存储在 Mongo 或 Redis 中，存储格式以 AppName 和日志类别分类存储。

日志存储字段在数据库中存储分别使用动态列的方式来存储，其中如下列是必须存在，自定义日志内容则由业务约定，使用动态列存储

日志存储固定字段如下：

序号	字段名称	字段说明
1	APP NAME	日志输入 APP 的名称
2	LOG TYPE	日志类型
3	REQUEST TIME	请求访问时间，使用 UNIX 时间戳格式，精确到毫秒
4	CLIENT IP:PORT	客户端请求 IP 地址和端口，格式为:**IP:PORT**
5	SERVER IP:PORT	应用服务所在的 IP 地址和端口，格式为:**IP:PORT**
6	ELAPSED TIME	请求消耗时间，单位为毫秒，表示整个请求进入到处理结束所消耗的时间
7	COMMUNICATION TYPE	通讯类型，目前约定为:**HTTP、THRIFT** 两类，后续视情况增加
8	REQUEST DATA LENGTH	请求数据大小，除去 HTTP 头或 TCP 协议头的实际内容长度
9	RESPONSE DATA LENGTH	响应数据大小，除去 HTTP 头或 TCP 协议头的实际内容长度

质量日志存储字段

序号	字段名称	字段说明
1	METHOD	HTTP 请求方法，如：GET、POST、PUT 等
2	URL	请求的 URL 地址，不包含查询参数、域名或 IP 端口信息
3	STATUS CODE	请求响应状态码
4	QUERY ENTITY	请求主体，用于识别查询请求，由业务约定
5	CALL METHOD	请求执行的方法，请求执行的具体操作名称，由业务约定
6	BUSSINESS CODE	业务状态码，业务端返回的业务状态码
7	RETURN CONTENT	返回内容，用于定义请求问题

3.4. 日志分析

3.4.1. 实时日志分析

将日志收集服务采集到的日志推送到 Kafka 消息队列中进行缓存，消息根据日志消息中的日志类别、应用名称、通讯方式等分成多个类别送入不同的队列，交由不同的分析程序获取队列数据进行分析计算，得出实时结果并通知存储到数据库中。

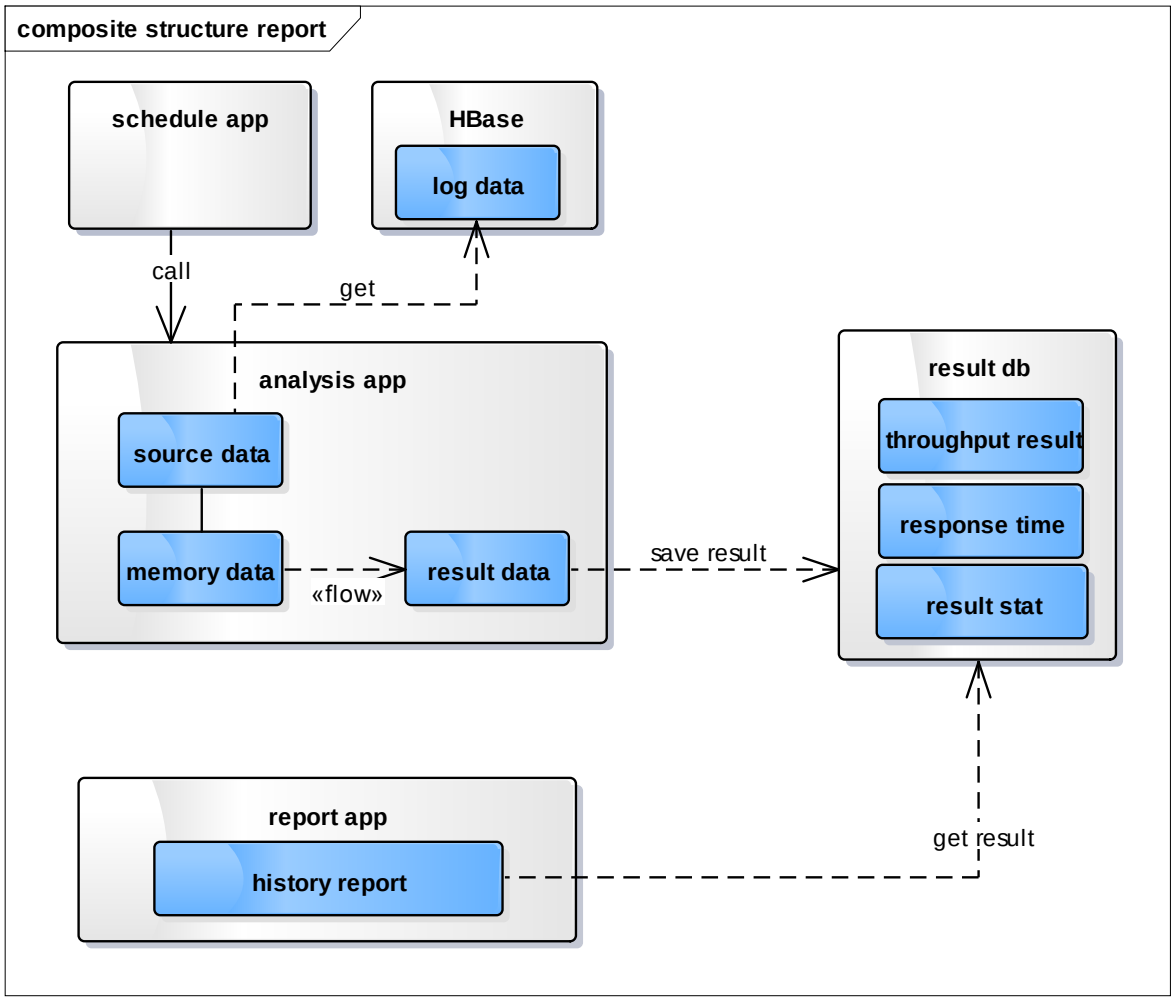
本期实现考虑到人力，可以减化此处功能，使用自行开发的应用获取日志消息，优先实现性能指标的数据分析和计算。

3.4.2. 历史日志分析

在指标发生变更，需要对以前的数据进行更新的时候，此时需要从存储的日志数据中获取原始数据，根据最新的分析程序逻辑重新计算分析结果，此时数据来源是存储在分布式存储库中的日志源数据，为了不影响实时运算结果，可交由 Hadoop 分布式运算并将结果更新

到结果数据库中。

3.5. 报表展现



说明：

1. 因为数据统计结果需要隔一定间隔才会将统计结果存储到结果数据库中，因此，对于一些相对统计时间较长的统计指标，需要实时显示统计结果，此时可以与报表程序做一个对接，将内存中的统计数据实时推送到报表系统中展现
2. 对于统计周期较长的指标，可以定期将中间结果存储到结果数据库中，减少程序异常退出时统计数据的丢失机率。
3. 报表中可以同时展现实时结果及历史结果数据

3.5.1. 分析结果存储与通知

最终日志结果

1. 系统吞吐量

字段名称	字段说明

APP NAME	日志输入 APP 的名称
METHOD	请求方法
URL	请求地址
RECORD TIME	记录时间，精确到分钟，格式如：yyyyMMddHHmm
THROUGHPUT	吞吐量（请求数）

每秒钟生成一条记录。

2. 响应时间

字段名称	字段说明
APP NAME	日志输入 APP 的名称
METHOD	请求方法
URL	请求地址
RECORD TIME	记录时间，暂定为小时，格式如：yyyyMMddHH，即：每小时生成一次
AVERAGE TIME	平均响应时间
MAX TIME	最大响应时间
MIN TIME	最小响应时间

注：在未到采集时间点的时间段，数据实时统计，可以与报表系统对接显示实时结果（刷新时间可配置）。

3. 请求信息

字段名称	字段说明
APP NAME	日志输入 APP 的名称
METHOD	请求方法
URL	请求地址
RECORD TIME	记录时间，暂定为小时，格式如：yyyyMMddHH，即：每小时生成一记录
STATUS CODE	响应状态码
REQUEST NUM	请求总数

注：在未到采集时间点的时间段，数据实时统计，可以与报表系统对接显示实时结果（刷新时间可配置）。

3.5.2. 分析指标

1. 系统吞吐量

统计间隔以分钟为单位，细化到每个应用下的每个接口，同时需要统计针对每个应用的吞吐量指标。

2. 响应时间

统计应用中每个接口请求的响应平均时间，按小时生成统计结果，单位为毫秒。

3. 最小响应时间

记录每个接口中最小的响应时间，单位为毫秒，按小时生成统计结果。查询多个时间分别显示各个小时结果并生成汇总结果。

4. **最大响应时间**

记录每个接口中的最大响应时间，单位为毫秒，按小时生成统计结果。查询多个时间分别显示各个小时结果并生成汇总结果。

5. **请求成功次数**

记录每个接口中的成功次数及占比。按小时生成统计结果，查询多天可以分别显示多天结果并生成汇总结果。

6. **请求失败次数**

记录每个接口中的失败次数及占比。按小时生成统计结果，查询多天可以分别显示多天结果并生成汇总结果。

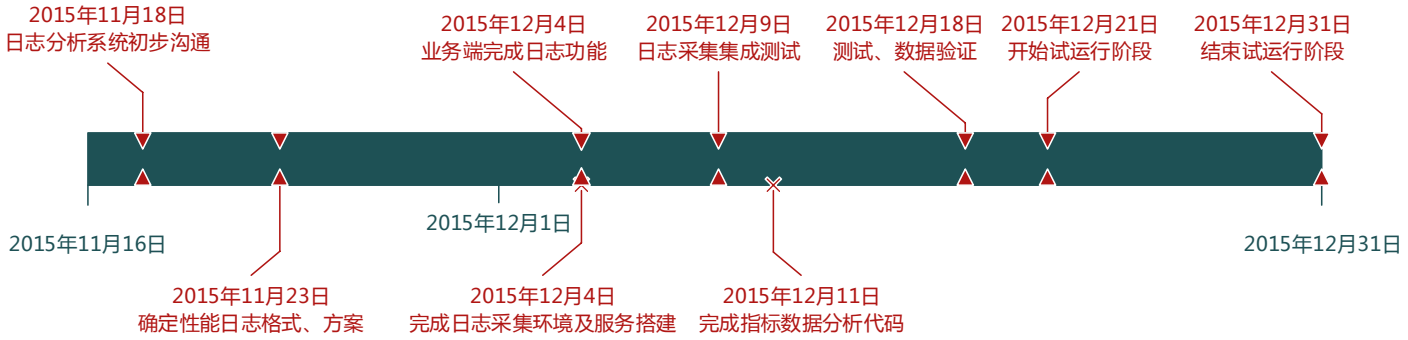
6.1.1. 分析结果报表

6.1.2. 告警处理

7. 日志分析涉及范围

涉及所有的服务项目，对于后端管理项目暂时不采集。

8. 预计实施计划



9. 任务列表

阶段	任务内容	负责人	预计完成时间	备注
设计	制定日志分析方案	顺炽国	2015-11-20	
	调研 fluentd 框架	顺炽国	2015-11-25	
日志采集	实现日志采集客户端功能(fluentd 采集代理)	顺炽国	2015-11-27	
	实现日志采集服务端功能(fluentd 服务)	顺炽国	2015-12-02	
	完成日志数据存储、查询功能(HBase)	顺炽国	2015-12-04	
	~~日志输出：bxr-api~~	~~陆宝勤~~	~~2015-11-25~~	
	~~日志输出：bi_data_api,account-api~~	~~陆宝勤~~	~~2015-11-26~~	
	~~日志输出：reader-api~~	~~陆宝勤~~	~~2015-11-27~~	
	~~日志输出：bxr_app_web,bxr_app_admin~~	~~陆宝勤~~	~~2015-12-01~~	
	~~日志采集、存储集成测试~~	~~宝勤、炽国~~	~~2015-12-09~~	
日志分析	日志分析：数据读取存储	顺炽国	2015-12-04	
	日志分析：任务调度、配置接口	顺炽国	2015-12-08	
	日志分析：吞吐量	顺炽国	2015-12-09	
	日志分析：响应时间	顺炽国	2015-12-10	
	日志分析：业务状态码统计	顺炽国	2015-12-11	
	分析结果数据集成测试	顺炽国	2015-12-18	
	分析结果数据验证	顺炽国，Roy	2015-12-21	
实时监控、报表	报表框架对比	杨成	2015-11-27	
	实时报表输出	杨成	2015-12-04	
	实时报表集成测试	杨成	2015-12-11	
试运行	试运行上线准备	杨成	2015-12-21	
	试运行		2015-12-31	

开发计划每周一次迭代，每次迭代目标如下：

第一周(11-23~11-30):

- 调研 fluentd 框架

- 日志采集客户端功能（fluentd 采集代理）
- 应用端采集功能实现：bxr-api、bi_data_api、account-api、reader-api

第二周(11-30~12-7):
- 日志采集服务端功能
- 日志数据存储、查询
- 应用端采集功能实现：bxr_app_web、bxr_app_admin
- 日志分析：数据读取存储
- 实时报表输出

第三周(12-7~12-14):
- 日志采集、存储测试
- 日志分析：任务调度、配置接口
- 日志分析：吞吐量
- 日志分析：响应时间
- 日志分析：业务状态码统计
- 实时报表集成测试

第三周(12-14~12-21):
- 分析结果数据集成测试
- 分析结果数据验证
- 试运行上线准备

10. 风险评估

11. 性能指标