基础服务技术架构

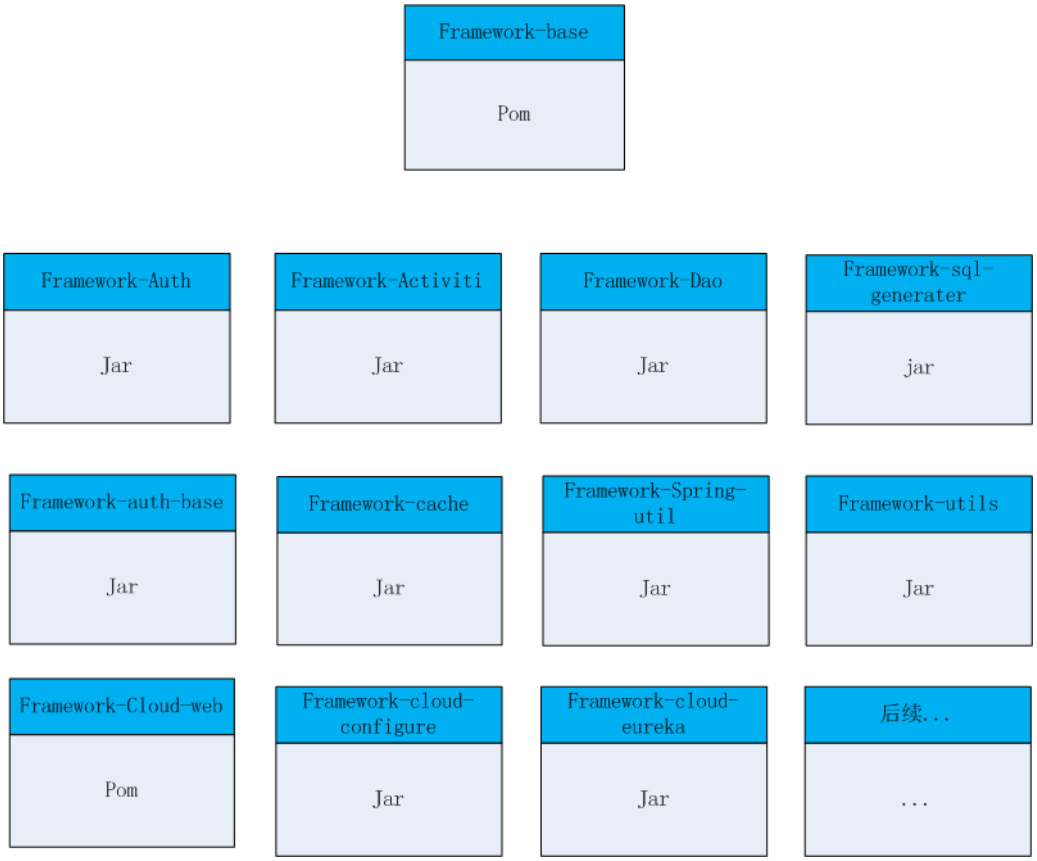
by lyq

前言

本项目是为后面项目快速开发而实现的一套开发套件，后端采取 maven 工程应用相关模块组件，前端通过 vue-admin 框架实现功能模板，实现即插即用，方便快速构建工程。

1 maven 开发套件

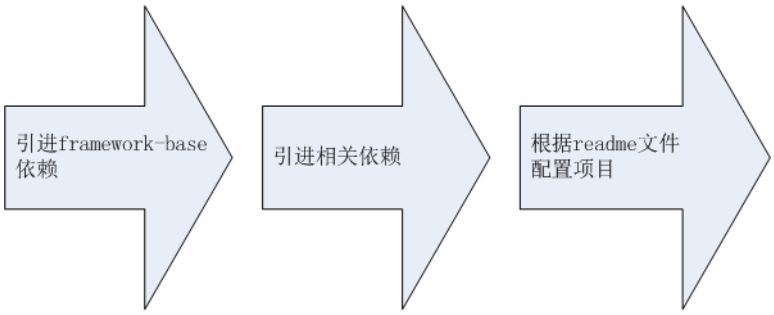
目前第一版计划构建模块有 springcloud、权限、日志；当前实现组件如下图：



后面持续完善相关功能。

2 构建项目

构建项目步骤如下：



开发平台所开发 springboot 或者 springcloud 项目，需要先引入相关依赖包，springcloud 应用可之间引用 framework-cloud-web 工程，使构建工程尽量简单

例如：我们需要在 springcloud 开发权限相关的功能，我们需要做的三步工作：

```
   <!---步骤一：-->
<parent>
        <groupId>com.ddd.framework.concise</groupId>
        <artifactId>concise</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <artifactId>auth-test</artifactId>
    <name>auth-test</name>
    <properties>
<!--指定执行类-->
        <mainClass>com.ddd.AuthMainApp</mainClass>
    </properties>
  <dependencies>
    <!--步骤二 -->
    <dependency>
        <groupId>com.ddd.framework.concise</groupId>
            <artifactId>framework-cloud-web</artifactId>
            <type>pom</type>
     </dependency>
    <dependency>
            <groupId>com.ddd.framework.concise</groupId>
            <artifactId>framework-auth-base</artifactId>
        </dependency>
  </dependencies>

 <build>
     <plugins>
    <!--指定执行类控件-->
          <plugin>
              <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-maven-plugin</artifactId>
          </plugin>
     </plugins>
  </build>
```

需要注意到是启动主项目工程需要指定main方法的执行类：

<!--步骤三 -->

按照framework-auth-base项目下的README.md配置项目。

3 权限管理

权限管理实现采用shiro+redis+rbac设计。

.3.1 数据表原型

权限数据库表设计采用RBAC权限原型设计，表结构如下：

3.1.1 RBAC 的组成

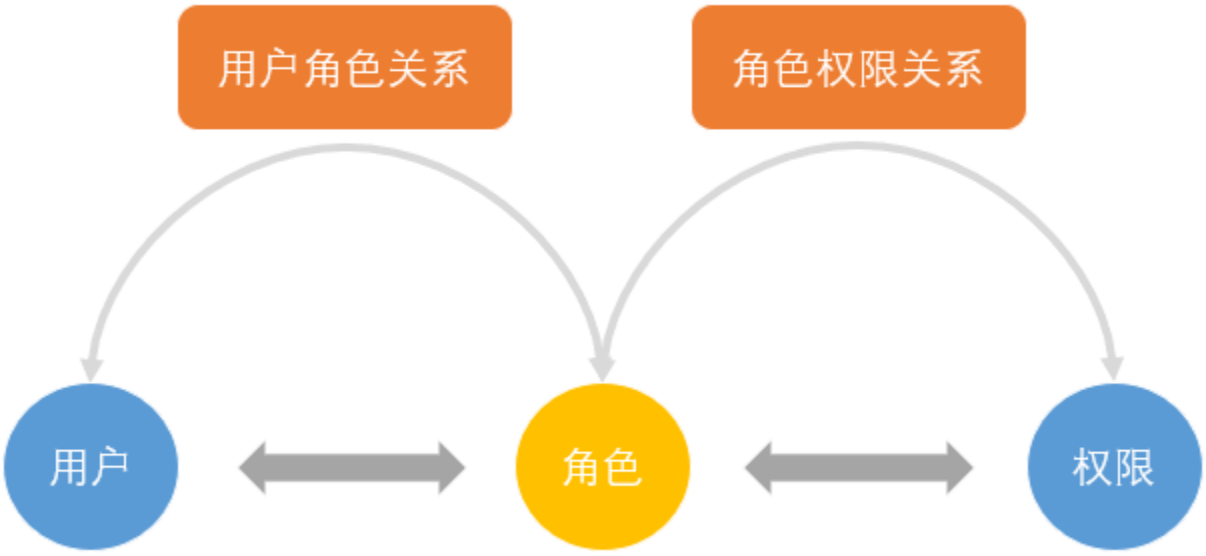
在 RBAC 模型里面，有 3 个基础组成部分，分别是：用户、角色和权限。

RBAC 通过定义角色的权限，并对用户授予某个角色从而来控制用户的权限，实现了用户和权限的逻辑分离（区别于 ACL 模型），极大地方便了权限的管理

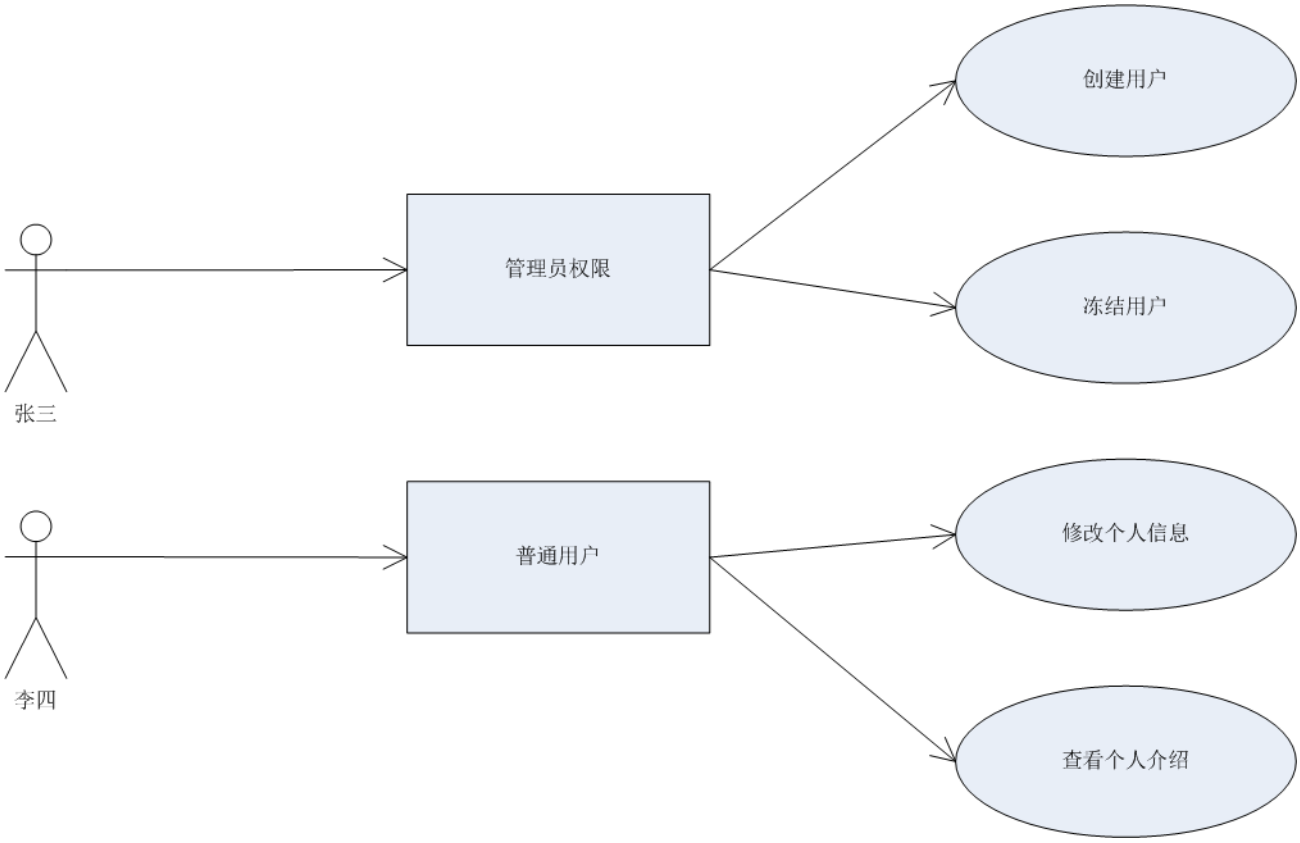
下面在讲解之前，先介绍一些名词：

- User（用户）：每个用户都有唯一的 UID 识别，并被授予不同的角色
- Role（角色）：不同角色具有不同的权限
- Permission（权限）：访问权限
- 用户-角色映射：用户和角色之间的映射关系
- 角色-权限映射：角色和权限之间的映射

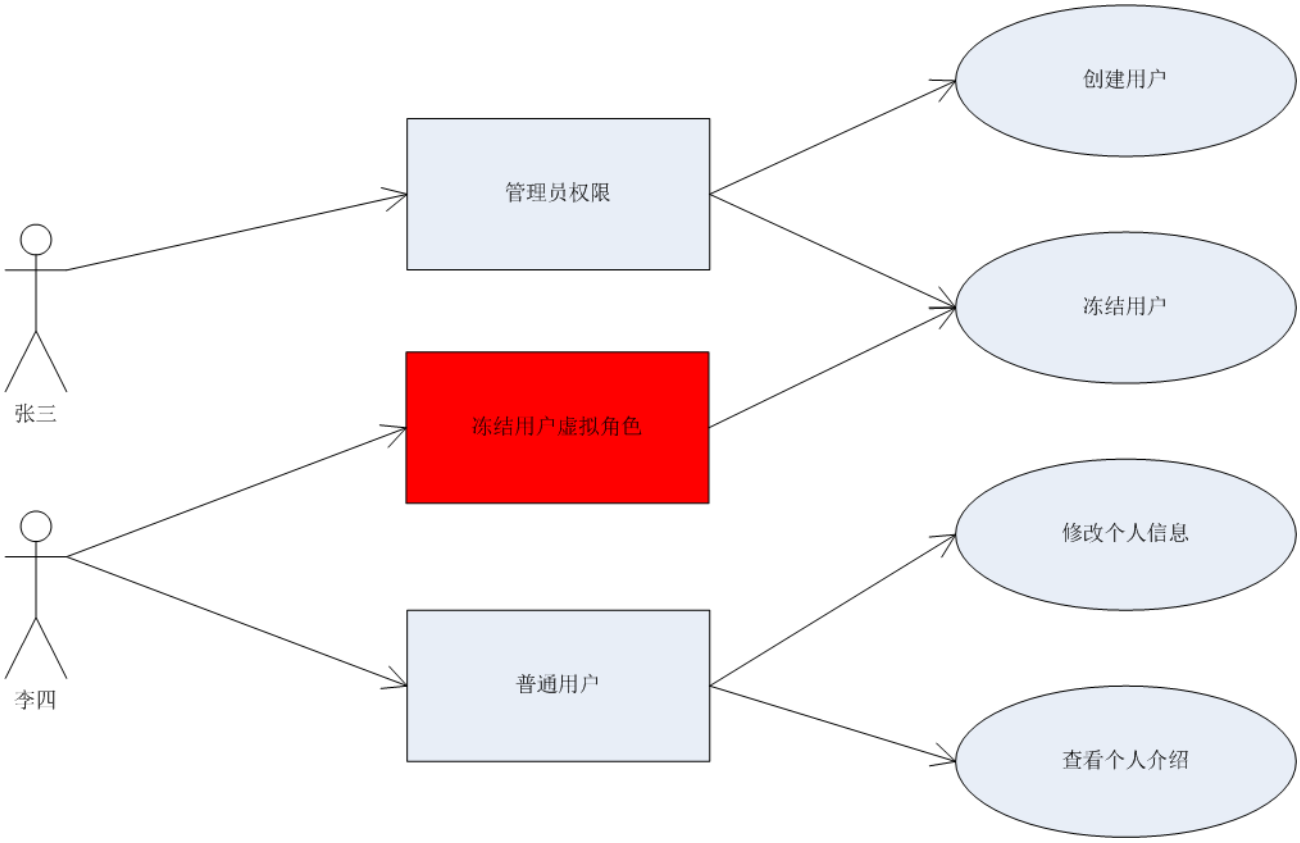
它们之间的关系如下图所示：



例如下图，管理员和普通用户被授予不同的权限，普通用户只能去修改和查看个人信息，而不能创建创建用户和冻结用户，而管理员由于被授 予所有权限，所以可以做所有操作。



如果单独给李四授权，冻结用户，本权限组件方案是给用户建立一个虚拟角色建立桥梁，这个虚拟角色不一定展示给用户看，如下图：



3.1.2RBAC 支持的安全原则

RBAC 支持三个著名的安全原则：最小权限原则、责任分离原则和数据抽象原则

- 最小权限原则：RBAC 可以将角色配置成其完成任务所需的最小权限集合
- 责任分离原则：可以通过调用相互独立互斥的角色来共同完成敏感的任务，例如要求一个计账员和财务管理员共同参与统一过账操作
- 数据抽象原则：可以通过权限的抽象来体现，例如财务操作用借款、存款等抽象权限，而不是使用典型的读、写、执行权限

3.1.3RBAC 的优缺点

（1）优点：

- 简化了用户和权限的关系
- 易扩展、易维护

（2）缺点：

- RBAC 模型没有提供操作顺序的控制机制，这一缺陷使得 RBAC 模型很难适应哪些对操作次序有严格要求的系统

.3.2　shiro 权限管理

3.2.1 shiro 简介

Apache Shiro 是 Java 的一个安全框架。Shiro 可以非常容易的开发出足够好的应用，其不仅可以用在 JavaSE 环境，也可以用在 JavaEE 环境。Shiro 可以帮助我们完成：认证、授权、加密、会话管理、与 Web 集成、缓存等；

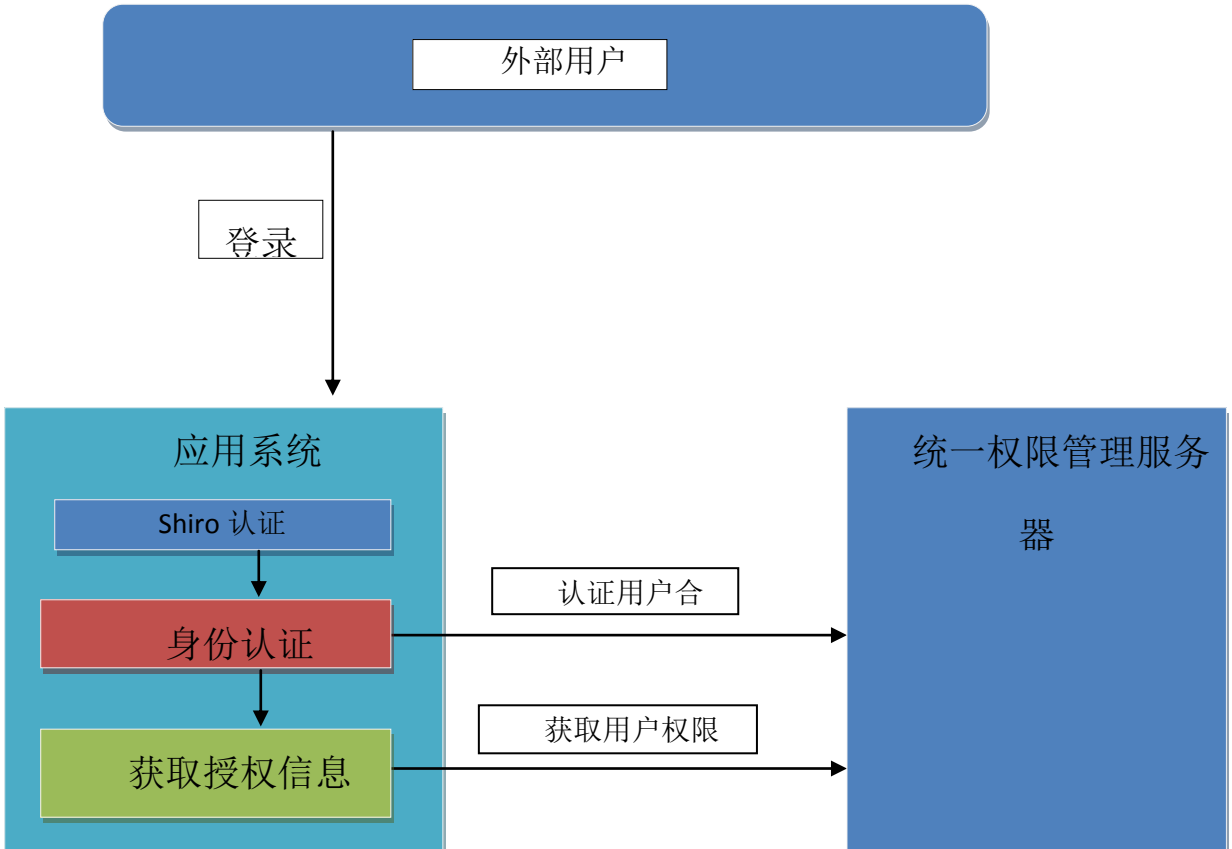
shiro 相关配置

3.2.2 项目 shiro 配置

本权限组件通过 properties 灵活配置项目需求，默认配置如下，*（以下配置关于跳转的配置都在该权限组件实现。）*

```
#登出路径
shiro.logout=/login/logout
#登录验证url，最终验证成功会跳转到shiro.successUrl，
shiro.loginUrl=/login/login
#登录成功后跳转到页面
shiro.successUrl=/index/index
#无权限访问时，跳转的路径
shiro.unauthorizedUrl=/login/notauth
#session过期时间 ，单位秒，不写为2天
shiro.session.timeout=604800
#zookeeper地址
shiro.zookeeper.url=localhost
#web domain
shiro.web.domain=192.168.30.88
#用户密码AES加密键
shiro.encryptKey=ddd
shiro.zookeeper.zkEnabled = false
   shiro.testEnabled = true
```

3.2.3 shiro 登录流程



我们需要给 Shiro 注入用户、权限信息，每次访问系统 url 的时候，会去验证两点：

1、用户是否有登录系统，如果没有返回登录界面。

2、如果已登录，判断用户是否有权限访问该 url 接口权限，跳转到配置好的相应页面，提示没有权限访问。

url 权限拦截器是通过 com.framework.security.ShiroFilterFactoryBean 实现的，相关配置如下（*该配置不需要做任何改动，只需了解即可*）：

```
<bean id="shiroFilter" class="com.framework.security.ShiroFilterFactoryBean">
      <!-- 调用我们配置的权限管理器 -->
       <!-- 设定登出时，重定向到url，此处为login/login -->
      <property name="securityManager" ref="securityManager" />
      <property name="redirectUrl" value ="${shiro.redirectUrl}" />
      <property name="loginUrl" value="${shiro.loginUrl}" />
      <property name="successUrl" value="${shiro.successUrl}" />
      <property name="unauthorizedUrl" value="${shiro.unauthorizedUrl}" />
      <property name="filterChainDefinitions">
          <value>
               ${shiro.redirectUrl} = anon
               ${shiro.loginUrl} = authc
               /swagger-ui.html = anon
               /webjars/** = anon
               /swagger-resources/** = anon
               /v2/api-docs/** = anon
               /css/** = anon
               /js/** = anon
               /img/** = anon
               ${shiro.logout} = logout
               ${shiro.successUrl} = authc
               /login/info = authc
               /** = url
          </value>
      </property>
       <property name="filterClassNames">
```

```
            <map>
    <!--指定filterChainDefinitions 中/**=url统一都会通过UriAuthorizationFilter 拦截-->
        <entry key="url" value="com.framework.security.filter.UriAuthorizationFilter"/>
    <!--指定filterChainDefinitions 中authc统一都会通过FormAuthenticationFilter拦截-->
<entry key="authc" value="com.framework.security.filter.FormAuthenticationFilter"/>
<!--指定filterChainDefinitions 中${shiro.logout} = logout统一都会通过LogoutFilter拦截-->

            entry key="logout" value="org.apache.shiro.web.filter.authc.LogoutFilter" />
            </map>
          </property>
       </bean>
```

登录验证实现

登录及鉴权实现都在 CustomAuthorizingRealm 实现。
登录验证：doGetAuthenticationInfo
鉴权：doGetAuthorizationInfo。

3.2.4 密码安全

本组件考虑到用户密码信息安全性，在数据库保存用户密码采用 aes 加盐的方式保存，验证的时候通过类 AESHashedMatcher 的方法 doCredentialsMatch。

3.2.5 session 缓存

采用redis保存用户session信息，用户信息在redis采用二进制字节保存，更多细节请参考代码 com.framework.security.RedisCacheManager。

加速访问
如果权限需要每次验证角色权限不去访问数据库，都经过缓存获取角色权限，则需要在配置文件设置 shiro.testEnabled = false，如果该配置设置成true，则每次登录都会从数据库查询用户相关角色权限。

3.2.6 修改角色权限

内存中角色权限可以通过 RoleManager 去实现修改，且必须通过搞改类里面的方法去实现修改。
权限放在应用内存，不做 redis 单独存储，如果应用满足两个条件需要开启 zookeeper 配置，即。
1、从数据库之装载一次权限，再次登录从缓存里面拿权限。
2、应用做了集群，如负载均衡。
该类修改权限方法通过 zookeeper 通知各个应用的修改权限信息。
zoookeeper 配置如下：

```
#zookeeper地址
shiro.zookeeper.url=localhost
shiro.zookeeper.zkEnabled = true
```

3.2.7 rbac 数据库交互

相关数据库的操作，该权限组件通过实现 com.framework.security.service 里面的三个接口完成,即：
SecurityPermissionService
SecurityRoleService
SecurityUserService

3.2.8 自动化建表

rbac 数据库相关的表通过引用 framework-sql-generator 实现自动化建表。

3.2.9 使用说明

请参考 framework-auth-base 下最新的 README.md。

4 execl 导入导出

JXL：支持比较低版本的 excel，比如 Excel 95 ,97 ,2000，2003
由于 Excel 版本比较低，导致最大行有限制，无法导出 65535 以上量级的数据
对于内存，和时间的花费也比 POI 基于内存+磁盘的方式高。

技术说明
1. 读取 Excel 公式（可以读取 Excel 97 以后的公式）
2. 生成 Excel 数据表（格式为 Excel 97）
3. 支持字体、数字、日期的格式化
4. 支持单元格的阴影操作，以及颜色操作
5. 修改已经存在的数据表
6. 是最基础的 excel api
7. 小文件读取效率比较高

通过产品设计生成产出两样文件：
1、execl 模板；
2、json 配置文件，这里不保存数据库，尽量不关联数据库。

5 日志分类管理实现

用户查询操作日志，此处不做说明，请参照信使就有架构。

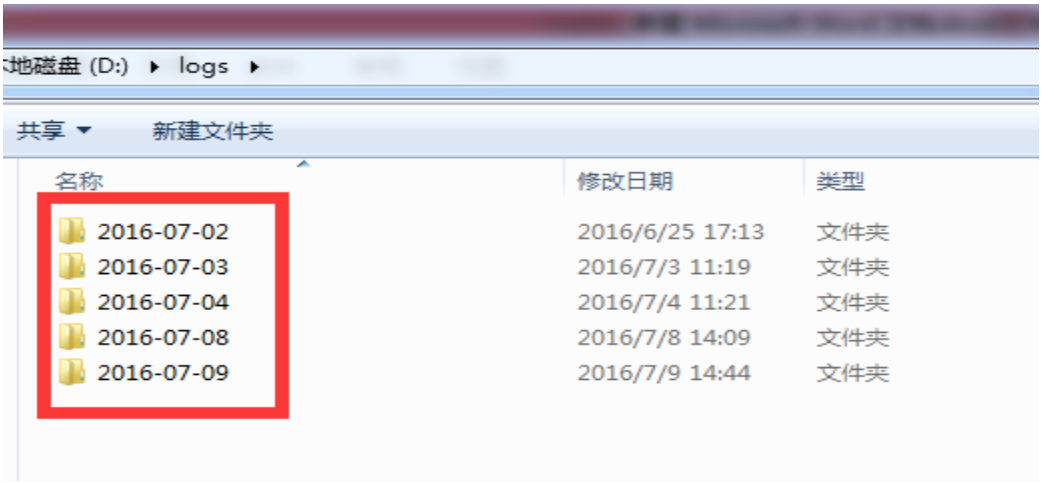
.5.1 日志级别

现设置如下的日志级别，请按照项目的不同情况可以进行变更输出。

级别	概要	说明	输出目的地
FATAL	致命错误	那些涉及的程序的异常终止。应立即输出到控制台等	控制台，文件
ERROR	错误	意外的其他运行时错误。应该立即输出到控制台等	控制台，文件
WARN	警告	使用该 API，这成为一个废元件，使用不当的 API，如靠近错误的事件。问题，但不被所述执行时所产生的异常没有它也不同一些预期的正常	控制台，文件
INFO	信息	运行时一些值得注意的事件（例如开始和结束）。留言内容应短暂停留	控制台，文件
DEBUG	调试信息	关于该系统的操作状态的详细信息	文件
TRACE	跟踪信息	比起调试信息，更详细的信息	文件

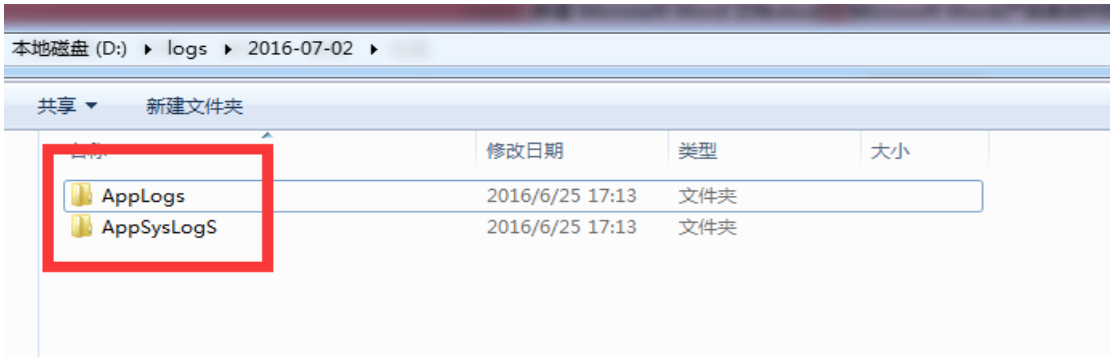
.5.2 应用层面日志

应用层面日志默认分为两小类，分别是应用操作层面、应用系统层面；应用层面日志，按天展示每日日志创建以当天日期命名的独立文件夹进行管理，如下图（1）；



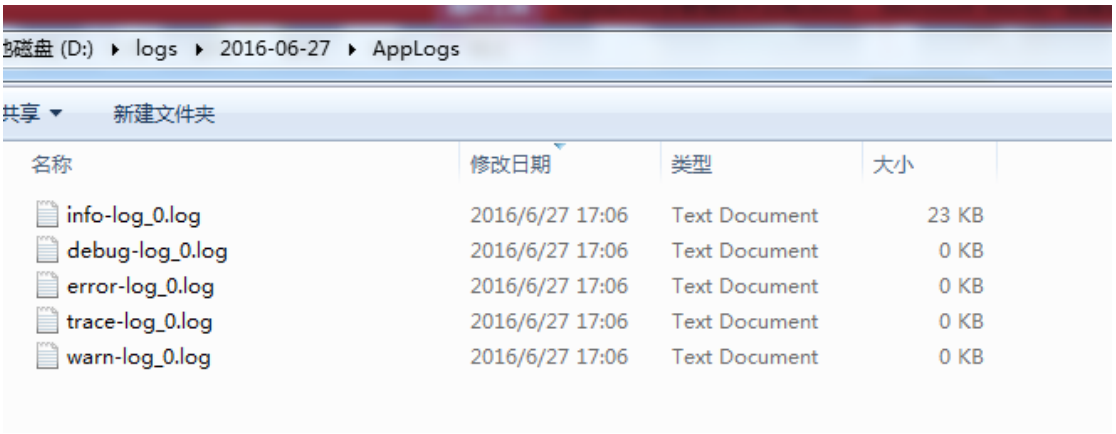
图（1）

每日日期命名创建的文件夹默认创建两个子文件夹（AppLogs、AppSysLogs），分别用来管理应用操作层面的日志（AppLogs）跟应用系统层面的日志（AppSysLogs），如下图（2）



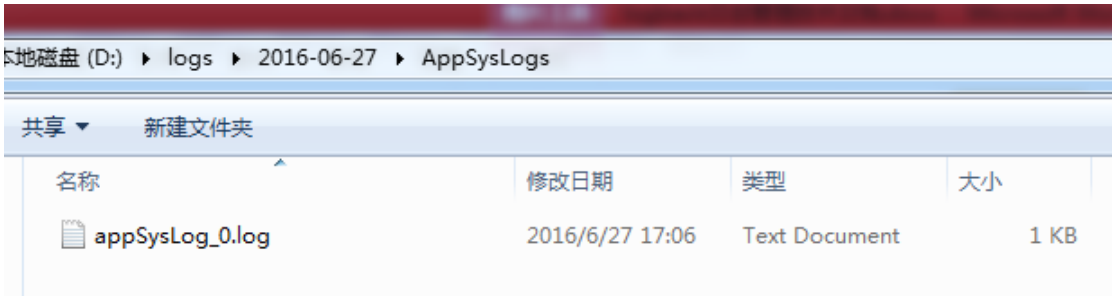
图（2）

进入到应用操作层面日志文件夹（AppLogs）看到如下图（3）所示的相关日志输出展示文件



图（3）

进入到应用系统层面的日志文件夹（AppSysLogs）可以看到如下图（4）所示的日志输出展示文件



图（4）

支持配置按生成文件的大小，版本递增自动分解文件，方便用户快速打开跟踪查看日志信息。

.5.3 DB 存储数据

数据存储基本样表

如下：

logging_event（日志基本信息记录表）

列名	描述
timestmp	时间戳
formatted_message	格式化信息
logger_name	
level_string	级别
thread_name	线程名
reference_flag	
arg0	
arg1	
arg2	
arg3	
caller_filename	文件名
caller_class	线程类
caller_method	方法
caller_line	行号
event_id	事件 ID

logging_event_exception（异常日志记录关联表）

列名	描述
event_id	时间 ID
trace_line	行号

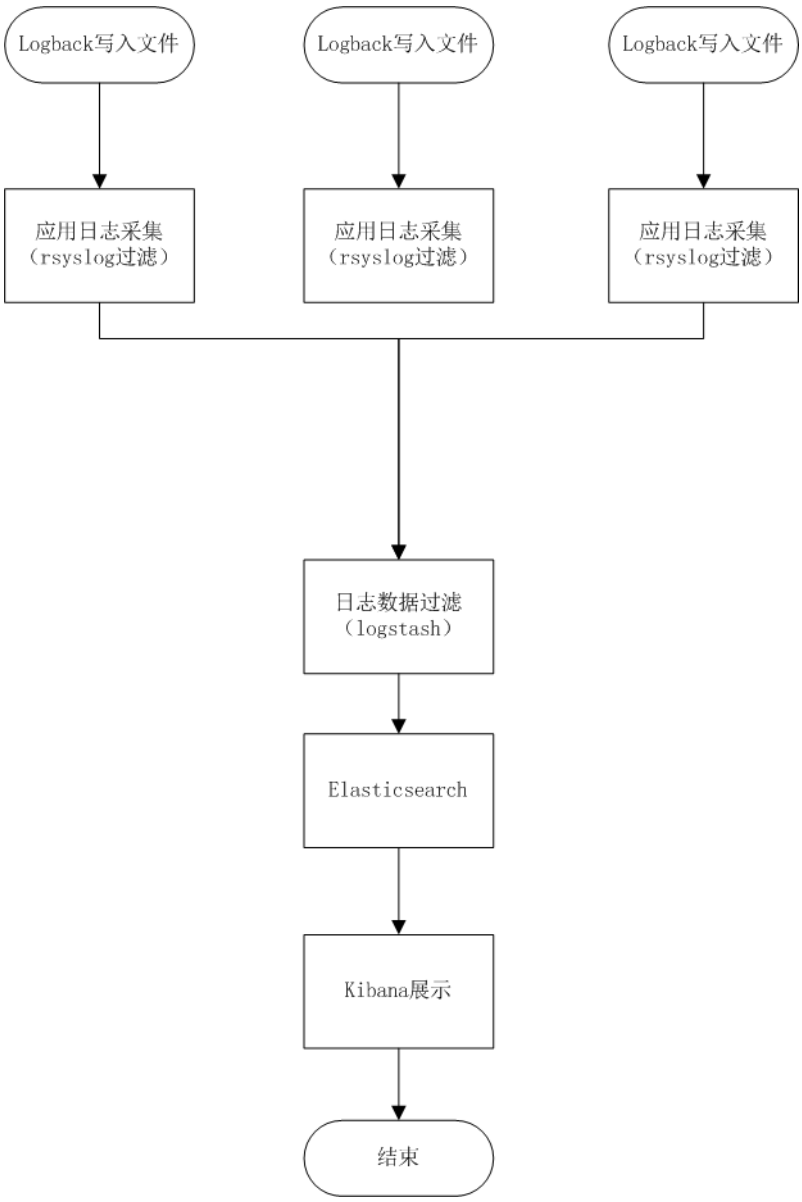
logging_event_property（日志生成对象）

列名	描述
event_id	事件 ID
mapped_key	对象映射主机键
mapped_value	对象映射主机键值

.5.4 日志采集流程

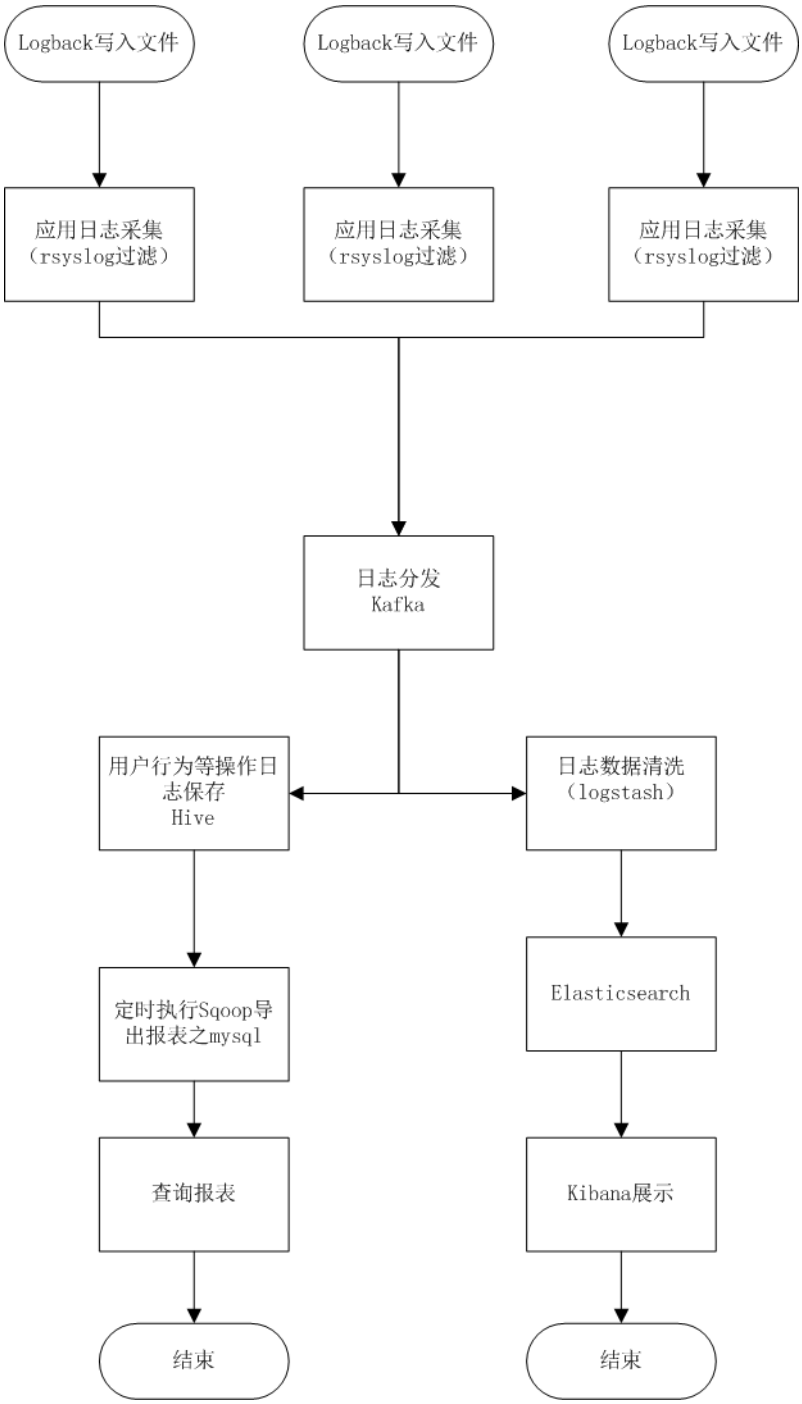
1.0 版本

此版本日志采集用于运维，流程如下:



2.0 版本

此版本日志除了采集回来运维，同时还需要满足大数据分析需求，流程如下：



流程步骤解读：1、Client 程序调用：用户从客户端访问请求调用程序模块功能

2、获取开始时间：客户程序调用时，开始记录程序响应时间

3、接口执行方法：客户端调用执行的接口响应的方法

4、获取接口结束时间接口方法请求参数，返回参数，响应时间等：接口调用结束完成时间记录、持续时长记录、返回参数记录

5、异常日志：如果接口调用发生异常，直接捕获异常日志，封装成日志对象，如果不是异常日志，则进行下一步

6、判断注解：没有注解，则不记录日志，有注解则根据写入方法级别的注解或者类(class)级别的注解来记录日志，如果两种注解都有，则方法级别的注解优先级高于类级别的注解，否则以类上的注解信息来记录日志

7、封装成日志对象：将本次请求获取到的相关参数进行封装成日志对象，然后转化成 json 字符串存入本地文件

.5.5 Aop 实现程序片段

5.5.1 日志拦截记录程序片段

```
/**
 * AOP 拦截日志记录。
 * @author cheng.zhu
 */
@Component
@Aspect
public class DddSoapApect {
 private static Logger log = Logger.getLogger(DddSoapApect.class);
 @Around("execution (* com.ddd..*(..))")
 public Object aroundMethod(ProceedingJoinPoint pjd) throws Throwable {
        /**
          *日志相关信息收集及文件存储逻辑实现
        */
      }
 }
```

5.5.2 自定义注解

注解类名	LogWrite	
字段	注释	是否必填
logDescription()	日志描述	否

代码：

```
package com.ddd.logs.annotations;
import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Target({ElementType.METHOD,ElementType.TYPE})//目标是方法跟类
@Retention(RetentionPolicy.RUNTIME)//注解会在class中存在，运行时可通过反射获取
@Documented//文档生成时，该注解将被包含在javadoc中，可去掉
public @interface LogWrite {
 /**
 *@param 日志描述
 */
 String     logDescription() default "";

}
```

5.5.3 方法上使用注解

```
/**
        * @Description: 打开activeMq演示页面
        * @Author:
        * @date下午14:55:00
```

```
     * @return
    */
@LogWrite(logDescription ="打开activeMq演示页面")
    @RequestMapping(value="/activeMqDisplay")
    public ModelAndView    displayRabbitMqPage(){
     ModelAndView modelView = new ModelAndView();
modelView.setViewName("activemq/activeMqDisplay");
    return modelView;
    }
```

5.5.4 类上使用注解

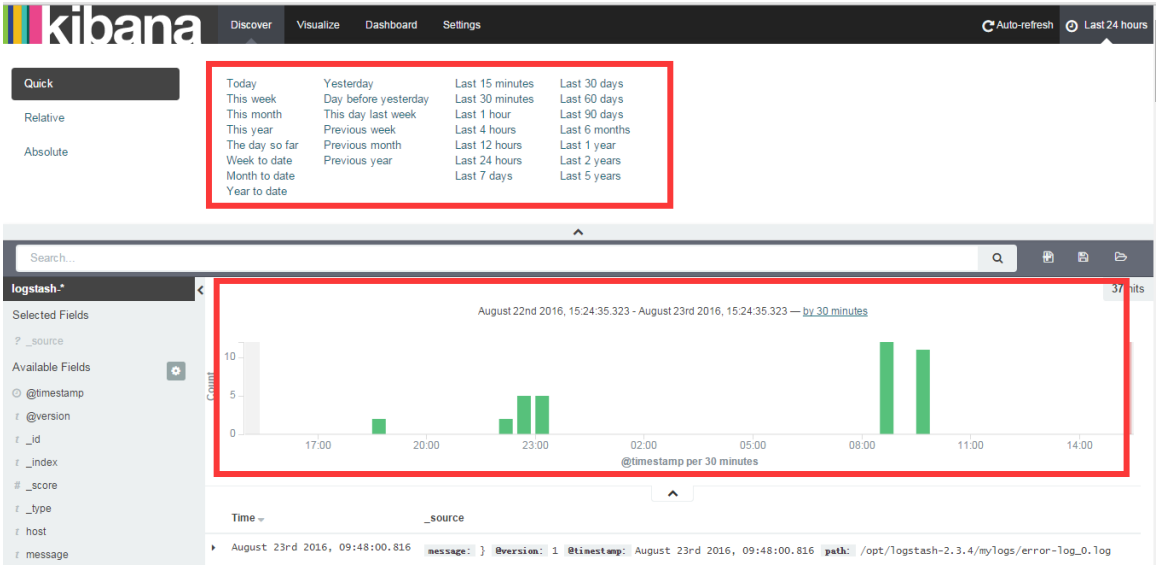
```
@Controller
@RequestMapping("/system/user")
@LogWrite(logDescription="用户管理模块控制类")
public class UserContorller {
……
……
}
```

5.5.5 日志过滤

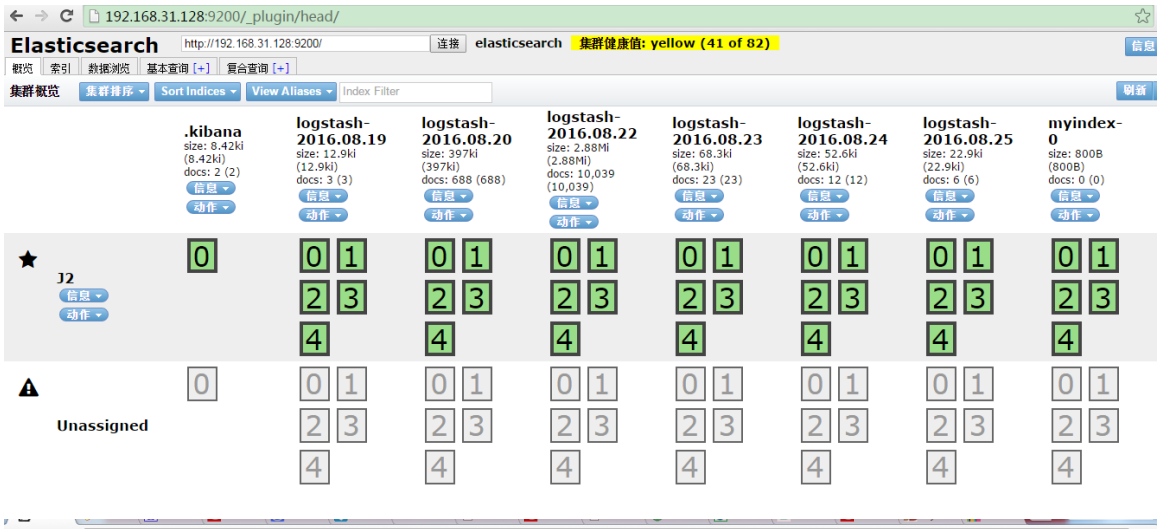
默认对全工程异常日志进行记录，对于无注解的定义不做日志记录，有注解才进行日志记录。

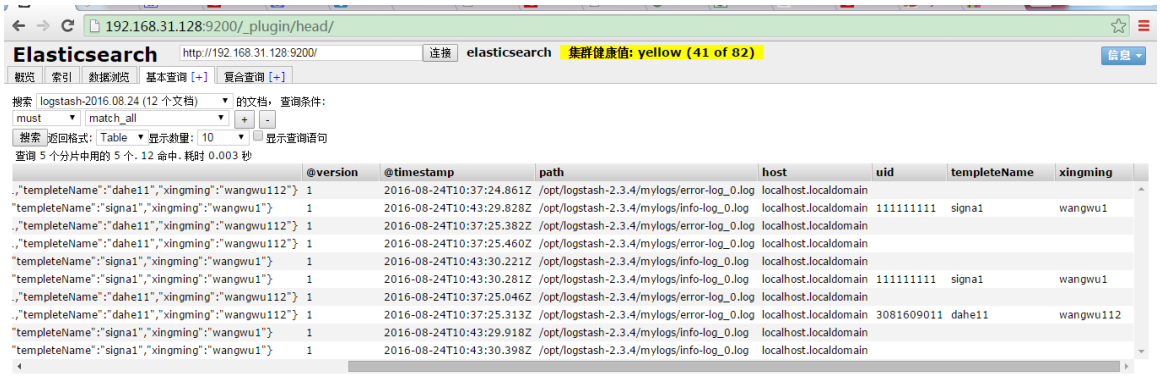
5.5.6 日志展示及检索

（1）日志展示采用 kibana 来展示，如下示例通过 kibana 可以选择不同的时间区间维度来展示日志写入的具体情况



（2）通过 elasticSearch 来进行检索，如下图示例





	@version	@timestamp	path	host	uid	templeteName	xingming
.,"templeteName":"dahe11","xingming":"wangwu112"}	1	2016-08-24T10:37:24.861Z	/opt/logstash-2.3.4/mylogs/error-log_0.log	localhost.localdomain			
"templeteName":"signa1","xingming":"wangwu1"}	1	2016-08-24T10:43:29.828Z	/opt/logstash-2.3.4/mylogs/info-log_0.log	localhost.localdomain	111111111	signa1	wangwu1
.,"templeteName":"dahe11","xingming":"wangwu112"}	1	2016-08-24T10:37:25.382Z	/opt/logstash-2.3.4/mylogs/error-log_0.log	localhost.localdomain			
.,"templeteName":"dahe11","xingming":"wangwu112"}	1	2016-08-24T10:37:25.460Z	/opt/logstash-2.3.4/mylogs/error-log_0.log	localhost.localdomain			
"templeteName":"signa1","xingming":"wangwu1"}	1	2016-08-24T10:43:30.221Z	/opt/logstash-2.3.4/mylogs/info-log_0.log	localhost.localdomain			
"templeteName":"signa1","xingming":"wangwu1"}	1	2016-08-24T10:43:30.281Z	/opt/logstash-2.3.4/mylogs/info-log_0.log	localhost.localdomain	111111111	signa1	wangwu1
.,"templeteName":"dahe11","xingming":"wangwu112"}	1	2016-08-24T10:37:25.046Z	/opt/logstash-2.3.4/mylogs/error-log_0.log	localhost.localdomain			
.,"templeteName":"dahe11","xingming":"wangwu112"}	1	2016-08-24T10:37:25.313Z	/opt/logstash-2.3.4/mylogs/error-log_0.log	localhost.localdomain	3081609011	dahe11	wangwu112
"templeteName":"signa1","xingming":"wangwu1"}	1	2016-08-24T10:43:29.918Z	/opt/logstash-2.3.4/mylogs/info-log_0.log	localhost.localdomain			
"templeteName":"signa1","xingming":"wangwu1"}	1	2016-08-24T10:43:30.398Z	/opt/logstash-2.3.4/mylogs/info-log_0.log	localhost.localdomain			