**威海市房屋交易与产权管理信息平台**接口协议

版本V1.0

威海时空云图信息技术有限公司

二0一九年三月

目录

[一、需求方案及流程解析 5](#_Toc2644)

[1.1资金存入 5](#_Toc24031)

[1.2资金划拨 5](#_Toc14971)

[1.3申请退款 6](#_Toc23205)

[1.4账户注销 6](#_Toc25883)

[1.5利息读取 7](#_Toc7692)

[1.6交易冲正 7](#_Toc24242)

[二、基础设施规划 7](#_Toc15620)

[2.1自建基础设施规划 8](#_Toc22831)

[2.1.1 网络设备规划 8](#_Toc22397)

[2.1.2 服务器规划 8](#_Toc15252)

[2.1.3 存储 9](#_Toc11256)

[2.1.4硬件扩容策略 9](#_Toc6084)

[三、技术选型 10](#_Toc9383)

[3.1环境 10](#_Toc14231)

[3.1.1JDK版本 10](#_Toc4782)

[3.1.2 项目管理工具 10](#_Toc17317)

[3.2、主框架 11](#_Toc12348)

[3.2.1微服务框架 11](#_Toc22897)

[3.2.2 安全管理 11](#_Toc2197)

[3.3、持久层 12](#_Toc21460)

[3.3.1关系型数据计算 12](#_Toc7493)

[3.3.2相关型数据计算 13](#_Toc24538)

[3.4、前端 13](#_Toc6674)

[3.4.1Web框架 13](#_Toc2416)

[3.5、数据高速缓存存储框架 14](#_Toc13680)

[3.5.1 J2Cache 14](#_Toc1683)

[2.5.2 Spring Redis 14](#_Toc2422)

[四、基础架构 15](#_Toc31964)

[4.1架构 15](#_Toc20907)

[4.2注册中心 15](#_Toc7335)

[4.3应用层 15](#_Toc7213)

[应用层高可用方案 16](#_Toc1566)

[4.4数据层 16](#_Toc12529)

[4.4.1数据层高可用 16](#_Toc20465)

[4.4.2数据库主备方案 17](#_Toc18487)

[4.5接口数据的设计 18](#_Toc1157)

[五、网络安全方案 19](#_Toc12644)

[5.1网络安全规划 19](#_Toc20958)

[5.1.1核心数据库采用物理隔离策略 20](#_Toc21981)

[5.1.2应用系统中间件服务器采取综合安全策略： 20](#_Toc18831)

[5.1.3内部局域网采取信息安全策略： 20](#_Toc8093)

[5.2接口安全机制 20](#_Toc14708)

[六、运维方案 21](#_Toc18844)

[6.1 巡检 21](#_Toc11017)

[6.1.1 巡检周期 21](#_Toc4160)

[6.1.2 巡检内容 21](#_Toc18067)

[6.2 监控 21](#_Toc10854)

[6.2.1 基础设施监控 21](#_Toc120)

[6.2.2 应用层监控 23](#_Toc30675)

[6.2.3 业务层监控 24](#_Toc27259)

[6.3 备份 26](#_Toc30730)

[6.3.1 网络备份 26](#_Toc4024)

[6.3.2 应用备份 26](#_Toc29904)

[七、接口定义 26](#_Toc14735)

[7.1 存入资金银行回执信息 26](#_Toc26096)

[7.1.1接口方向 26](#_Toc11491)

[7.1.2接口描述 27](#_Toc32760)

[7.1.3接口地址 27](#_Toc10841)

[7.1.4接口请求 27](#_Toc11785)

[7.1.5接口响应 27](#_Toc4469)

[7.2资金划拨转款后回执消息 28](#_Toc30443)

[7.2.1接口方向 28](#_Toc17869)

[7.2.2接口描述 28](#_Toc4102)

[7.2.3接口地址 28](#_Toc11949)

[7.2.4接口请求 28](#_Toc13558)

[7.2.5接口响应 28](#_Toc11474)

[7.3申请退款后回执消息 29](#_Toc20680)

[7.3.1接口方向 29](#_Toc32566)

[7.3.2接口描述 29](#_Toc11820)

[7.3.3接口地址 29](#_Toc5886)

[7.3.4接口请求 29](#_Toc19992)

[7.3.5接口响应 30](#_Toc7174)

[7.4账户注销后回执消息 30](#_Toc16568)

[7.4.1接口方向 30](#_Toc11937)

[7.4.2接口描述 30](#_Toc31385)

[7.4.3接口地址 30](#_Toc6279)

[7.3.4接口请求 30](#_Toc12401)

[7.4.5接口响应 31](#_Toc28166)

[7.5账户利息数据更新 31](#_Toc22010)

[7.5.1接口方向 31](#_Toc27124)

[7.5.2接口描述 31](#_Toc175)

[7.5.3接口地址 32](#_Toc25077)

[7.5.4接口请求 32](#_Toc24070)

[7.5.5接口响应 32](#_Toc8285)

[7.6交易冲正数据记录 32](#_Toc30076)

[7.6.1接口方向 33](#_Toc28772)

[7.6.2接口描述 33](#_Toc25809)

[7.6.3接口地址 33](#_Toc7248)

[7.6.4接口请求 33](#_Toc10992)

[7.6.5接口响应 33](#_Toc17769)

# 

# 一、需求方案及流程解析

## 1.1资金存入

分为两种存款方式：①开发商通过网银存入银行子账户②通过银行柜台存入子账户。

流程图：

发起子账户申请

打印交款通知单

建立户室和子账户的关联

监管系统

开发企业存入申请

接口服务

回执

存入银行子账户①

银行服务

流程描述：

开发企业存入申请-》打印交款通知单-》存入银行子账户-》建立户室和子账户的关联-》银行回执信息-》监管中心确认归档

流程节点①解析：

1. 银行存入消息回执，子账户的存入结果调用接口服务进行自动回执
2. 银行提供查询接口，监管系统根据子账户名称可查询到存入状态和明细信息。

应用场景业务逻辑：合同网签销售经理转件-》对应子账户判断钱数是否和通知单一致-》提示异常信息。

## 1.2资金划拨

流程图：

打印取款通知单

监管中心审核

银行确认并转款

监管系统

接口服务

开发企业取款申请

确认信息

回执

流程描述：

开发企业取款申请-》打印取款通知单-》监管中心审核-》银行现场确认-》转款后回执消息给平台

## 1.3申请退款

流程图：

打印退款通知单

监管中心审核

银行确认并退款

监管系统

接口服务

开发企业退款申请

子账户关联

确认信息

回执

解除

流程描述：

开发企业退款申请-》打印退款通知单-》监管中心审核-》银行现场确认-》银行退款-》房屋和子账户解除关联-》银行回执消息给平台

## 1.4账户注销

流程图：

开发企业注销申请

监管系统

监管中心审核

打印注销通知单

注销

监管账户

银行确认

确认信息

接口服务

释放

回执

子账户

流程描述：

开发企业申请注销-》打印注销通知单-》监管中心审核-》银行确认-》释放账户下的子账户-》回执信息给平台-》注销监管账户

备注：如果子账户存在余额则转入监管账户

## 1.5利息读取

流程图：

推送

银行系统

接口服务

监管系统

监管账户

更新

流程描述：

银行推送利息数据给平台-》平台对应账户更新利息数据

## 1.6交易冲正

流程图：

银行系统

接口服务

监管系统

冲正记录

推送

添加

流程描述：

银行定期推送冲正记录-》平台信息存入

备注：当退款时需要判断当前子账户余额，如果不足则划拨其他子账户的钱来完成退款；当申请资金划拨时需要扣除手续费，优先扣除利息，如果金额不足就提醒首先存入手续费，当前扣除和存入都操作总账户余额。

# 二、基础设施规划

## 2.1自建基础设施规划

### 2.1.1 网络设备规划

**负载均衡器**

RADWARE应用负载均衡设备,型号:为ODS-504,有，4个可选的千兆位电端口,1G主内存,500M处理能力(最大可通过License升级为4G)

**防火墙**

CISCO ASA5520防火墙

并发连接：280000

网络吞吐：450

安全过滤：225MB

网络端口：4个千兆以太网接口+1个快速

用户数限：无用户数限制用户

VPN支持：支持

**交换机**

Quidway S3952P-EI

传输速率：10Mbps/100Mbps/1000Mbps

网络标准：IEEE 802.1Q、IEEE 802.1D

端口数量：48

接口介质：10/100Base-T、1000Base-X

传输模式：全双工/半双工自适应

背板带宽：32Gbps

### 2.1.2 服务器规划

**数据库服务器**

IBM System x3850 M2, 4个处理器，每处理器为6核，共计24核。内存大小16G。SAS硬盘，硬盘大小587 GB。4U 机架，集成双千兆以太网接口，两块千兆的光纤网卡。

**WEB服务器**

IBM System x3850 M2, 4个处理器，每处理器为6核，共计24核。内存大于8G。SAS硬盘，硬盘大小587 GB。4U 机架，集成双千兆以太网接口，两块千兆的光纤网卡。

**管理终端**

IBM System x3560，1个Intel Xeon E5450处理器，内存大小2G，2U机架。

### 2.1.3 存储

**光纤存储柜**

光纤存储柜（EVA4100）

**光纤交换机**

光纤交换机（ 4/32B SAN Switch）

### 2.1.4硬件扩容策略

当网站发展到一定阶段，随着用户量不断扩大，现有的网络资源和服务器资源不能满足用户需要的时候，就需要对平台进行服务器和网络的扩容。以下是两种平台扩容的方式：

**增加服务器**

对于web的并发处理有瓶颈时，新增的web服务器，把新增的web服务器填加到Web服务器集群中，以增加WEB的并发处理能力。

对于数据库有处理压力时，可以增加数据库服务器，增加数据库服务器加入数据库的集群中。

**增加存储**

对于存储容量不能满足业务需要时，可以考虑在磁盘柜中新增加硬盘，甚至考虑新增磁盘柜。

**升级服务器**

可以升级服务器的内存、硬盘，甚至考虑用新的性能更高的服务器来替换。

**网络扩容**

1) 申请更大的网络带宽

2) 引入CDN

3) 升级内网交换机。

# 三、技术选型

## 3.1环境

### 3.1.1JDK版本

根据业务需求，项目不是单一应用服务，具有大数据计算等需求，结合稳定性，性能等因素考虑选择Java JDK 8版本，Java8版本优势

* Java7官方已经停止技术支持
* 支持大数据开发的相应业务需求。
* 在很多情况下，Java8 都能提升应用性能，而无需任何改变或性能调优。
* Lambda 表达式、 Streams API 以及现有类的新方法都是提高生产力的重要工具。
* Java8 新推出的 Optional 类型在处理 null 值时，能减少 NullPointerExceptions 的可能性，给开发者极大的灵活度。

### 3.1.2 项目管理工具

为什么要选择项目管理工具？构建是程序员每天要做的工作，而且相当长的时间花在了这上面，而项目管理工具使这系列的工作完全自动化。 我们一直在寻找避免重复的方法，设计的重复，文档的重复，编码的重复，构建的重复等，我们需要一个项目管理工具具有跨平台，最大的消除了构建的重复的能力。所以经过调研，选择Maven作为项目管理工具

Maven的优势：

* maven不仅是构建工具，它还是依赖管理工具和项目管理工具，提供了中央仓库，能够帮我们自动下载构件。
* 为了解决的依赖的增多，版本不一致，版本冲突，依赖臃肿等问题，它通过一个坐标系统来精确地定位每一个构件（artifact）。
* 还能帮助我们分散在各个角落的项目信息，包括项目描述，开发者列表，版本控制系统，许可证，缺陷管理系统地址。
* .maven还为全世界的java开发者提供了一个免费的中央仓库，在其中几乎可以找到任何的流行开源软件。通过衍生工具(Nexus),我们还能对其进行快速搜索
* maven对于目录结构有要求，约定优于配置，用户在项目间切换就省去了学习成本。

## 3.2、主框架

### 3.2.1微服务框架

**Spring Boot的优势：**

* 编码简单：Spring由于其繁琐的配置，一度被人认为“配置地狱”，各种XML、Annotation配置，让人眼花缭乱，而且如果出错了也很难找出原因。
* 配置，部署简单
* 监控简单：采用了spring-boot-start-actuator之后，直接以REST的方式，获取进程的运行期性能参数。当然这些metrics有些是有敏感数据的，spring-boot-start-actuator为此提供了一些Basic Authentication认证的方案，这些方案在实际应用过程中也是不足的。Spring Boot更多的是采用Java Config的方式，对Spring进行配置。

**Spring Boot相关技术融合：**

整体采用SEDA，也就是Stage-EDA。可以看到，整体是以处理顺序进行展示的，响应过程类似。在处理过程中，主要会有前置过滤，核心功能处理，后置过滤几大部分。

过滤器都是可插拔式的，并且可以根据实际场景进行扩展开发。每个过滤器都是Stage，比如ClientInstance合法性检查、调用鉴权、解密、限流等等。

一个请求Stage与Stage的转换，实现上是切换不同的线程池，并以EDA的方式驱动。

对于业务逻辑的开发者而言，只需要关心CORE部分的业务逻辑实现，其他的非功能都由框架进行统一实现。

### 3.2.2 安全管理

Shiro 是一个用 Java 语言实现的框架，通过一个简单易用的 API 提供身份验证和授权。使用 Shiro，您就能够为您的应用程序提供安全性而又无需从头编写所有代码。

这个解决方案提供了一个高度安全的集中式的身份验证基础设施，通过它的 API 或诸如 RADIUS 或 LDAP 这样的标准身份验证协议可很容易地与各种应用程序或 IT 基础设施组件集成。它支持广泛的软硬件或基于手机的身份验证令牌，在选择具体的身份验证方法方面提供了最大限度的灵活性。

由于 Shiro 提供具有诸多不同数据源的身份验证，以及 Enterprise Session Management，所以是实现单点登录（SSO）的理想之选 — 大型企业内的一个理想特性，因为在大型企业内，用户需要在一天内经常登录到并使用不同系统。这些数据源包括 JDBC、LDAP、 Kerberos 和 Microsoft® Active Directory® Directory Services (AD DS)。

Shiro 的 Session 对象允许无需 HttpSession 即可使用一个用户会话。通过使用一个通用的 Session 对象，即便该代码没有在一个 Web 应用程序中运行，仍可以使用相同的代码。没有对应用服务器或 Web 应用服务器会话管理的依赖，您甚至可以在命令行环境中使用 Shiro。换言之，使用 Shiro 的 API 编写的代码让您可以构建连接到 LDAP 服务器的命令行应用程序并且与 web 应用程序内用来访问 LDAP 服务器的代码相同。

## 3.3、持久层

### 3.3.1关系型数据计算

Apache MyBatis

MyBatis 包含一个非常强大的查询缓存特性,它可以非常方便地配置和定制。MyBatis 3 中的缓存实现的很多改进都已经实现了,使得它更加强大而且易于配置。

默认情况下是没有开启缓存的,除了局部的 session 缓存,可以增强变现而且处理循环 依赖也是必须的。要开启二级缓存,你需要在你的 SQL 映射文件中添加一行:  <cache/>

字面上看就是这样。这个简单语句的效果如下:

* 映射语句文件中的所有 select 语句将会被缓存。
* 映射语句文件中的所有 insert,update 和 delete 语句会刷新缓存。
* 缓存会使用 Least Recently Used(LRU,最近最少使用的)算法来收回。
* 根据时间表(比如 no Flush Interval,没有刷新间隔), 缓存不会以任何时间顺序来刷新。
* 缓存会存储列表集合或对象(无论查询方法返回什么)的 1024 个引用。
* 缓存会被视为是 read/write(可读/可写)的缓存,意味着对象检索不是共享的,而且可以安全地被调用者修改,而不干扰其他调用者或线程所做的潜在修改。

所有的这些属性都可以通过缓存元素的属性来修改。

比如: <cache  eviction="FIFO"  flushInterval="120000"  size="512"  readOnly="true"/>

这个更高级的配置创建了一个 FIFO 缓存,并每隔 120秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会 导致冲突。

可用的收回策略有, 默认的是 LRU:

* LRU – 最近最少使用的:移除最长时间不被使用的对象。
* FIFO – 先进先出:按对象进入缓存的顺序来移除它们。
* SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
* WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

flushInterval(刷新间隔)可以被设置为任意的正整数,而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置,也就是没有刷新间隔,缓存仅仅调用语句时刷新。

size(引用数目)可以被设置为任意正整数,要记住你缓存的对象数目和你运行环境的可用内存资源数目，默认值是1024。

readOnly(只读)属性可以被设置为 true 或 false。只读的缓存会给所有调用者返回缓 存对象的相同实例，因此这些对象不能被修改，这提供了很重要的性能优势。可读写的缓存 会返回缓存对象的拷贝(通过序列化) ，这会慢一些,但是安全,因此默认是 false。

Mybatis优势：

* MyBatis可以进行更为细致的SQL优化，可以减少查询字段。
* MyBatis容易掌握，而Hibernate门槛较高。

### 3.3.2相关型数据计算

Druid 1.1

Druid是一个高效的数据查询系统，主要解决的是对于大量的基于时序的数据进行聚合查询。数据可以实时摄入，进入到Druid后立即可查，同时数据是几乎是不可变。通常是基于时序的事实事件，事实发生后进入Druid，外部系统就可以对该事实进行查询。

小米，阿里等互联网大厂在时序数据处理的架构中也应用到此系统。

## 3.4、前端

### 3.4.1Web框架

thymeleaf+ace

1、Thymeleaf 是 Web 和独立环境的现代服务器端 Java 模板引擎，能够处理HTML，XML，JavaScript，CSS 甚至纯文本。

2、Thymeleaf 的主要目标是提供一种优雅和高度可维护的创建模板的方式。为了实现这一点，它建立在自然模板的概念上，将其逻辑注入到模板文件中，不会影响模板被用作设计原型。这改善了设计的沟通，弥补了设计和开发团队之间的差距。

3、Thymeleaf 也从一开始就设计了Web标准 - 特别是 HTML5 - 允许您创建完全验证的模板，Spring Boot 官方推荐使用  thymeleaf 而不是 JSP。

4、Spring Boot 中使用 Thymeleaf  模板引擎时非常简单，因为 Spring Boot 已经提供了默认的配置，比如解析的文件前缀，文件后缀，文件编码，缓存等等，程序员需要的只是写 html 中的内容即可。

Ace是一款轻量且功能丰富的管理模板，干净且易于使用。

当前版本具有以下功能：

1. 4种不同的皮肤
2. 基于Bootstrap 3的响应式设计（特别是3.1.1）
3. 使用Javascript和CSS构建器来构建您自己的最小文件
4. 在页面帮助中，您可以轻松选取任何元素并查看其工作原理
5. 自定义元素和插件

## 3.5、数据高速缓存存储框架

### 3.5.1 J2Cache

J2Cache 是 OSChina 目前正在使用的两级缓存框架（要求至少 Java 8）。第一级缓存使用内存(同时支持 Ehcache 2.x、Ehcache 3.x 和 Caffeine)，第二级缓存使用 Redis(推荐)/Memcached 。 由于大量的缓存读取会导致 L2 的网络成为整个系统的瓶颈，因此 L1 的目标是降低对 L2 的读取次数。 该缓存框架主要用于集群环境中。单机也可使用，用于避免应用重启导致的缓存冷启动后对后端业务的冲击。

### 2.5.2 Spring Redis

redis是一个key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

Redis 是一个高性能的key-value数据库。 redis的出现，很大程度补偿了memcached这类key/value存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了Java，C/C++，C#，PHP，JavaScript，Perl，Object-C，Python，Ruby，Erlang等客户端，使用很方便。

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

# 四、基础架构

## 4.1架构



## 4.2注册中心

管理员对合法企业、银行进行注册，分别分配appKey、appSecret，用于接口访问时的权限验证。

## 4.3应用层

Tomcat是一个开放源代码、运行servlet和JSP Web应用软件的基于Java的Web应用软件容器。Tomcat Server是根据servlet和JSP规范进行执行的，因此我们就可以说Tomcat Server也实行了Apache-Jakarta规范且比绝大多数商业应用软件服务器要好。

Tomcat是Java Servlet 2.2和JavaServer Pages 1.1技术的标准实现，是基于Apache许可证下开发的自由软件。Tomcat是完全重写的Servlet API 2.2和JSP 1.1兼容的Servlet/JSP容器。Tomcat使用了JServ的一些代码，特别是Apache服务适配器。随着Catalina Servlet引擎的出现，Tomcat第四版号的性能得到提升，使得它成为一个值得考虑的Servlet/JSP容器，因此目前许多WEB服务器都是采用Tomcat。

这次选型的时候，考虑到，容器的稳定性，和功能性，选择tomcat8的版本，8版本为现在最稳定的tomcat版本，本可以支持现在主流的大数据分析平台的要求。

### 应用层高可用方案

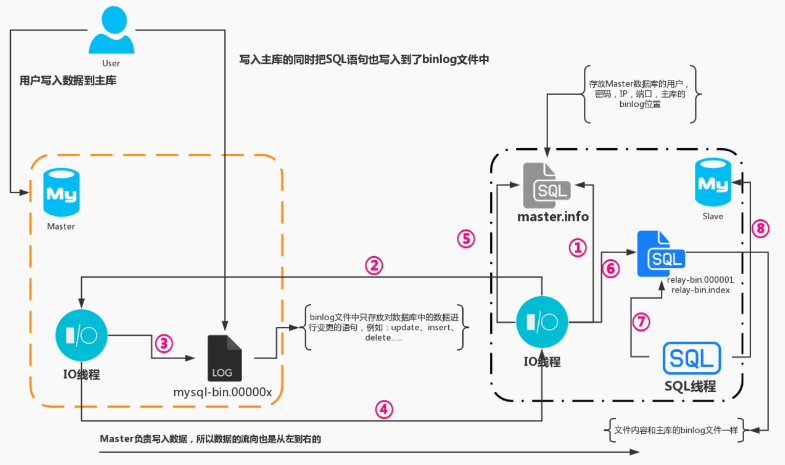
高可用的方案采用的是，冗余tomcat，redis共享session的方案，同一个应用会启动多个tomcat，

前端通过nginx的哈希规则路由的后端tomcat，这时会出现一个问题，当一个tomcat宕机时，访问在这台服务器的用户请求怎么才能被路由到其他正常的服务器。Nginx分发请求的时候会有一个健康检查，当健康检查3次没有存活时，流量会被分发到其他的Tomcat中，那么这时，就需要redis来保存tomcat会话的session，来做到用户无感知。

## 4.4数据层

### 4.4.1数据层高可用

主从复制



MySQL主从复制之前我们需要先启动Master数据库然后再启动Salve数据库，然后在Salve数据库中执行start slave;，执行完成之后，流程就如下了：

Salve的IO线程会读取mastr.info文件中配置好的主库信息，比如说存放的有：Master数据库的用户名、密码、端口、还有Master的binlog索引位置；

拿到信息之后就带着信息去链接Master的主库IO线程

当主库的IO线程先检查SLave传过来的配置信息是否正确，如果正确，就拿着Slave传过来的binlog索引位置和Master库的binlog文件中最后一个索引位置进行对比，如果一致就陷入等待状态，等待Master的binlog索引位置更新；

如果不一致就把Slave传过来的binlog索引位置往后的所有SQL语句包括最后一条SQL语句的索引位置发送个给Slave的IO线程；

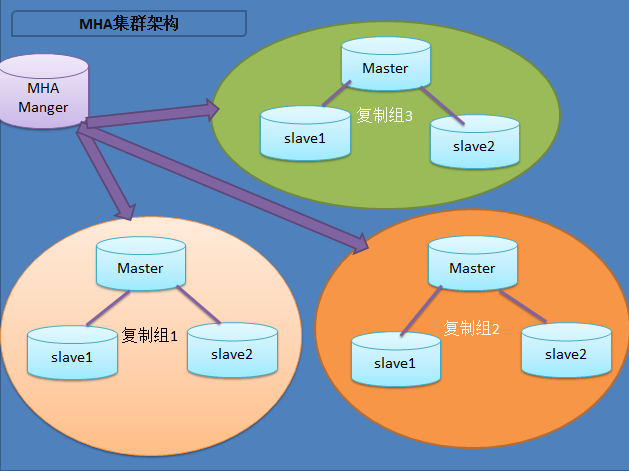
Slave的IO线程拿到信息之后，先把Master传过来的binlog索引在Slave的master.info文件中进行更新；

然后再把Master传过来的SQL语句写入到relay文件中，然后继续循环执行第二个步骤；

Slave的SQL线程会一直持续的观察relay日志文件中是否有改动，如果没有就继续监听；

如果发现relay中有变动，那么就获取变动的内容转换为SQL语句，并且把SQL语句在Salve的数据库中进行执行

### 4.4.2数据库主备方案



MHA（Master High Availability）目前在MySQL高可用方面是一个相对成熟的解决方案，它由日本DeNA公司youshimaton（现就职于Facebook公司）开发，是一套优秀的作为MySQL高可用性环境下故障切换和主从提升的高可用软件。在MySQL故障切换过程中，MHA能做到在0~30秒之内自动完成数据库的故障切换操作，并且在进行故障切换的过程中，MHA能在最大程度上保证数据的一致性，以达到真正意义上的高可用。

该软件由两部分组成：MHA Manager（管理节点）和MHA Node（数据节点）。MHA Manager可以单独部署在一台独立的机器上管理多个master-slave集群，也可以部署在一台slave节点上。MHA Node运行在每台MySQL服务器上，MHA Manager会定时探测集群中的master节点，当master出现故障时，它可以自动将最新数据的slave提升为新的master，然后将所有其他的slave重新指向新的master。整个故障转移过程对应用程序完全透明。

在MHA自动故障切换过程中，MHA试图从宕机的主服务器上保存二进制日志，最大程度的保证数据的不丢失，但这并不总是可行的。例如，如果主服务器硬件故障或无法通过ssh访问，MHA没法保存二进制日志，只进行故障转移而丢失了最新的数据。使用MySQL 5.5的半同步复制，可以大大降低数据丢失的风险。MHA可以与半同步复制结合起来。如果只有一个slave已经收到了最新的二进制日志，MHA可以将最新的二进制日志应用于其他所有的slave服务器上，因此可以保证所有节点的数据一致性。

## 4.5接口数据的设计

接口的数据一般都采用JSON格式进行传输，JSON的值有六种数据类型：

* Number：整数或浮点数
* String：字符串
* Boolean：true 或 false
* Array：数组包含在方括号[]中
* Object：对象包含在大括号{}中
* null：空类型

所以，传输的数据类型不能超过这六种数据类型。如果有日期类型，则用毫秒数表示日期。

服务器返回的数据结构，一般为：

{

"status": "001",

"msg": "系统错误",

"data": {

"obj": null,

"totalPages": 0

}

}

status: 返回码，000表示成功，非000表示各种不同的错误

000 ： 成功

001 ：系统错误

003 ： 未登录

004 ： 没有权限

005 ： 未查询到数据

msg: 描述信息，成功时为"success"，错误时则是错误信息

data: 成功时返回的数据，类型为对象或数组

totalPages 总页数

obj 返回的数据，数据类型限定为对象或数组，当请求需要的数据为单个对象时则传回对象，当请求需要的数据是列表时，则为某个对象的数组。

# 五、网络安全方案

## **5.1网络安全规划**

网络安全是整个系统安全运行的基础，是保证系统安全运行的关键。网络系统的安全需求包括以下几个方面：

Ø 网络边界安全需求

Ø 入侵监测与实时监控需求

Ø 安全事件的响应和处理需求分析

这些需求在各个应用系统上的不同组合就要求把网络分成不同的安全层次。

我们针对企业网络层的安全策略采用硬件保护与软件保护，静态防护与动态防护相结合，由外向内多级防护的总体策略。

根据安全需求和应用系统的目的，整个网络可划分为六个不同的安全层次。具体是：

Ø 核心层：核心数据库；

Ø 安全层：应用信息系统中间件服务器等应用；

Ø 基本安全层：内部局域网用户；

Ø 可信任层：公司本部与营业部网络访问接口；

Ø 危险层：Internet。

信息系统各安全域中的安全需求和安全级别不同，网络层的安全主要是在各安全区域间建立有效的安全控制措施，使网间的访问具有可控性。具体的安全策略如下：

### 5.1.1核心数据库采用物理隔离策略

应用系统采用分层架构方式，客户端只需要访问中间件服务器即可进行日常业务处理，从物理上不能直接访问数据库服务器，保障了核心层数据的高度安全。

### 5.1.2应用系统中间件服务器采取综合安全策略：

应用系统中间件的安全隐患主要来自局域网内部，为了保障应用系统中间件服务的安全，在局域网中可通过划分虚拟子网对各安全区域、用户和安全域间实施安全隔离，提供子网间的访问控制能力。同时，中间件服务器本身可以通过配置相应的安全策略，限定经过授权的工作站、用户方能访问系统服务，保障了中间件服务器的安全性；

### 5.1.3内部局域网采取信息安全策略：

公司本部及营业部内部局域网处于基本安全层的网络，主要是对于安全防护能力较弱的终端用户在使用，因此考虑的重点在于两个方面，一个是客户端的病毒防护，另一个是防止内部敏感信息的对外泄露。因此，通过选用网络杀毒软件达到内部局域网的病毒防护，同时，使用专用网络安全设备（如硬件防火墙）建立起有效的安全防护，通过访问控制ACL等安全策略的配置，有效地控制内部终端用户和外部网络的信息交换，实现内部局域网的信息安全。

## **5.2接口安全机制**

接口采用RESTful架构，RESTFul最重要的一个设计原则就是，客户端与服务器的交互在请求之间是无状态的，也就是说，当涉及到用户状态时，每次请求都要带上身份验证信息。实现上，大部分都采用token的认证方式，一般流程是：

1、用户用密码登录成功后，服务器返回token给客户端；

2、客户端将token保存在本地，发起后续的相关请求时，将token发回给服务器；

3、服务器检查token的有效性，有效则返回数据，若无效，分两种情况：

token错误，这时需要用户重新登录，获取正确的token

token过期，这时客户端需要再发起一次认证请求，获取新的token

然而，此种验证方式存在一个安全性问题：当登录接口被劫持时，黑客就获取到了用户密码和token，后续则可以对该用户做任何事情了。用户只有修改密码才能夺回控制权。

如何优化呢？我们目前的做法是给每个接口都添加签名。通过注册中心给客户端分配一个密钥，每次请求接口时，将密钥和所有参数组合成源串，根据签名算法生成签名值，发送请求时将签名一起发送给服务器验证。这样，黑客不知道密钥，不知道签名算法，就算拦截到登录接口，后续请求也无法成功操作。

# 六、运维方案

## 6.1 巡检

### 6.1.1 巡检周期

1. 设备巡检，不定期巡检
2. 网络巡检，日巡检
3. 服务运行情况巡检，日巡检

### 6.1.2 巡检内容

1、设备巡检：查看机房环境，室温，湿度，机柜情况，服务器和交换机指示灯情况

2、网络巡检，巡检各个网络设备CPU，内存使用情况，有无报警信息，端口配置是否正常，电源风扇是否正常。

3、服务运行情况巡检，基础设施：包括服务器CPU，内存，磁盘和流量使用情况，

4、应用服务巡检：包括redis集群运行情况，数据库运行情况，Tomcat容器运行情况

## 6.2 监控

### 6.2.1 基础设施监控

基础设施监控采用Zabbix

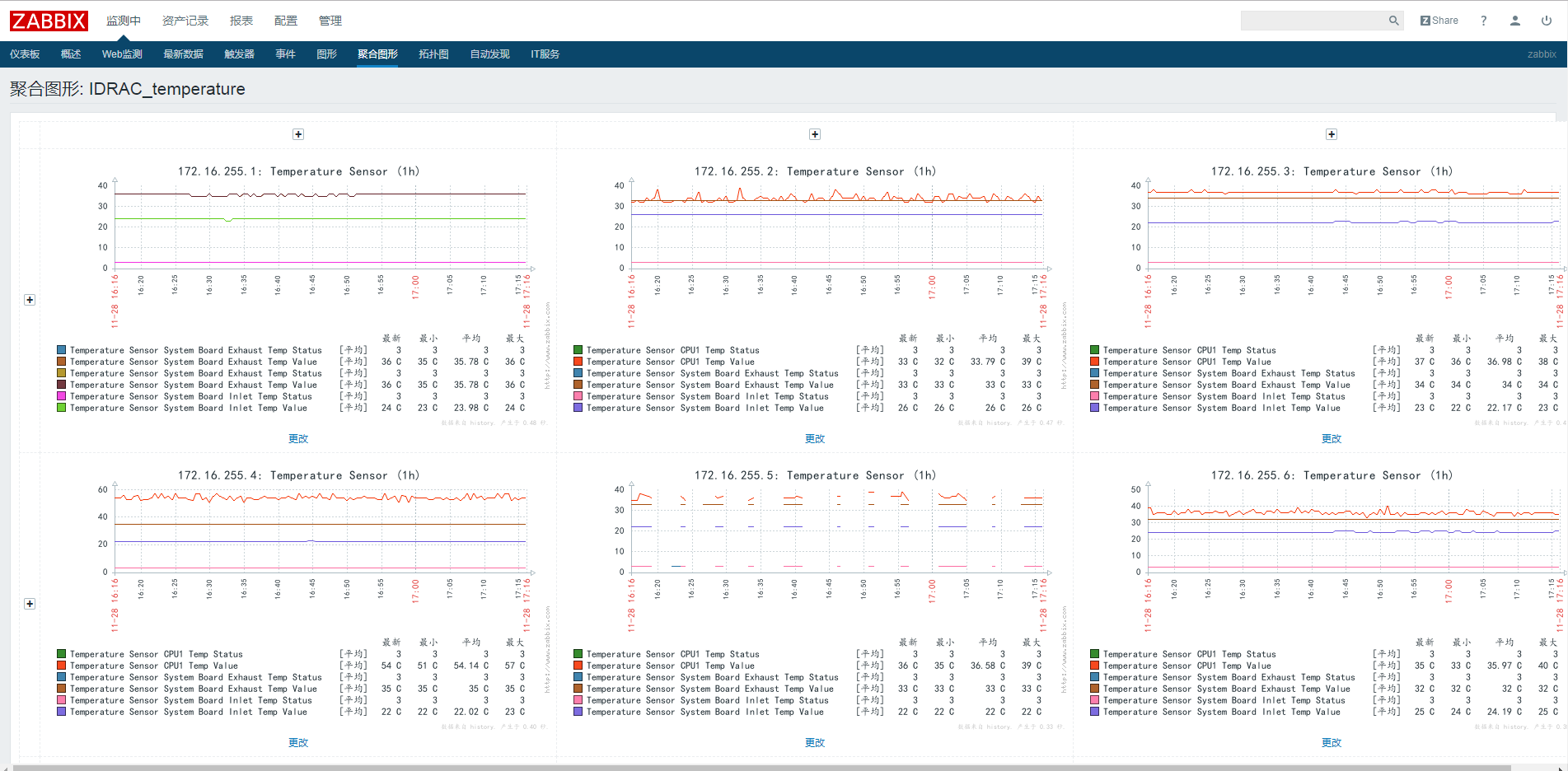
zabbix是一个基于WEB界面的提供分布式系统监视以及网络监视功能的企业级的开源解决方案。

zabbix能监视各种网络参数，保证服务器系统的安全运营；并提供灵活的通知机制以让系统管理员快速定位/解决存在的各种问题。

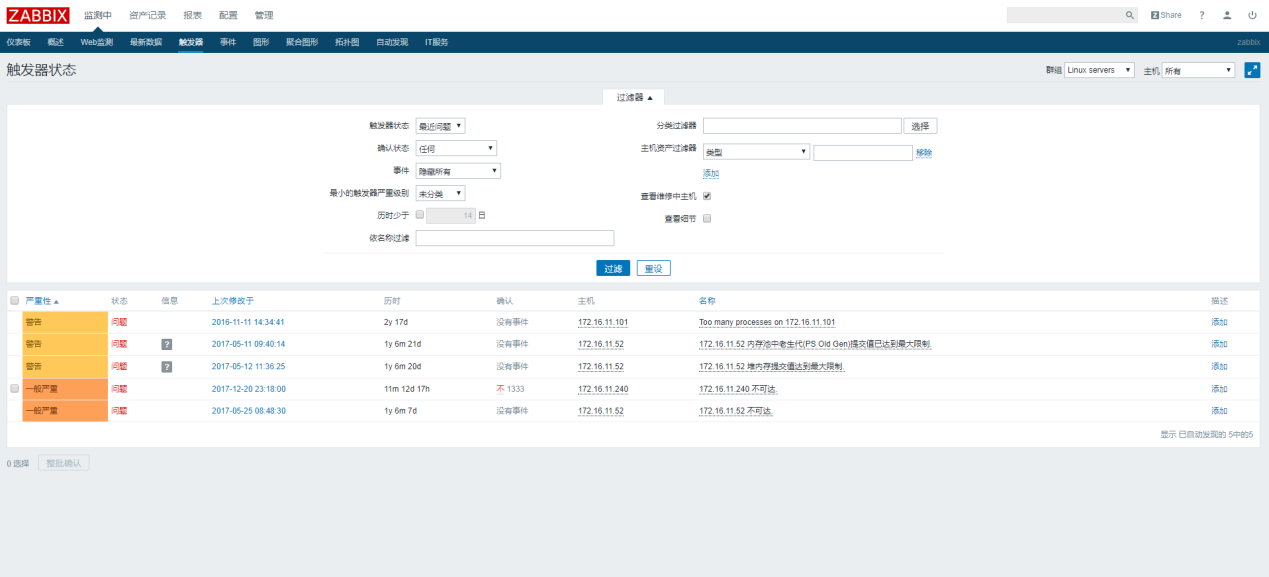
zabbix由2部分构成，zabbix server与可选组件zabbix agent。

zabbix server可以通过SNMP，zabbix agent，ping，端口监视等方法提供对远程服务器/网络状态的监视，数据收集等功能，它可以运行在Linux，Solaris，HP-UX，AIX，Free BSD，Open BSD，OS X等平台上。

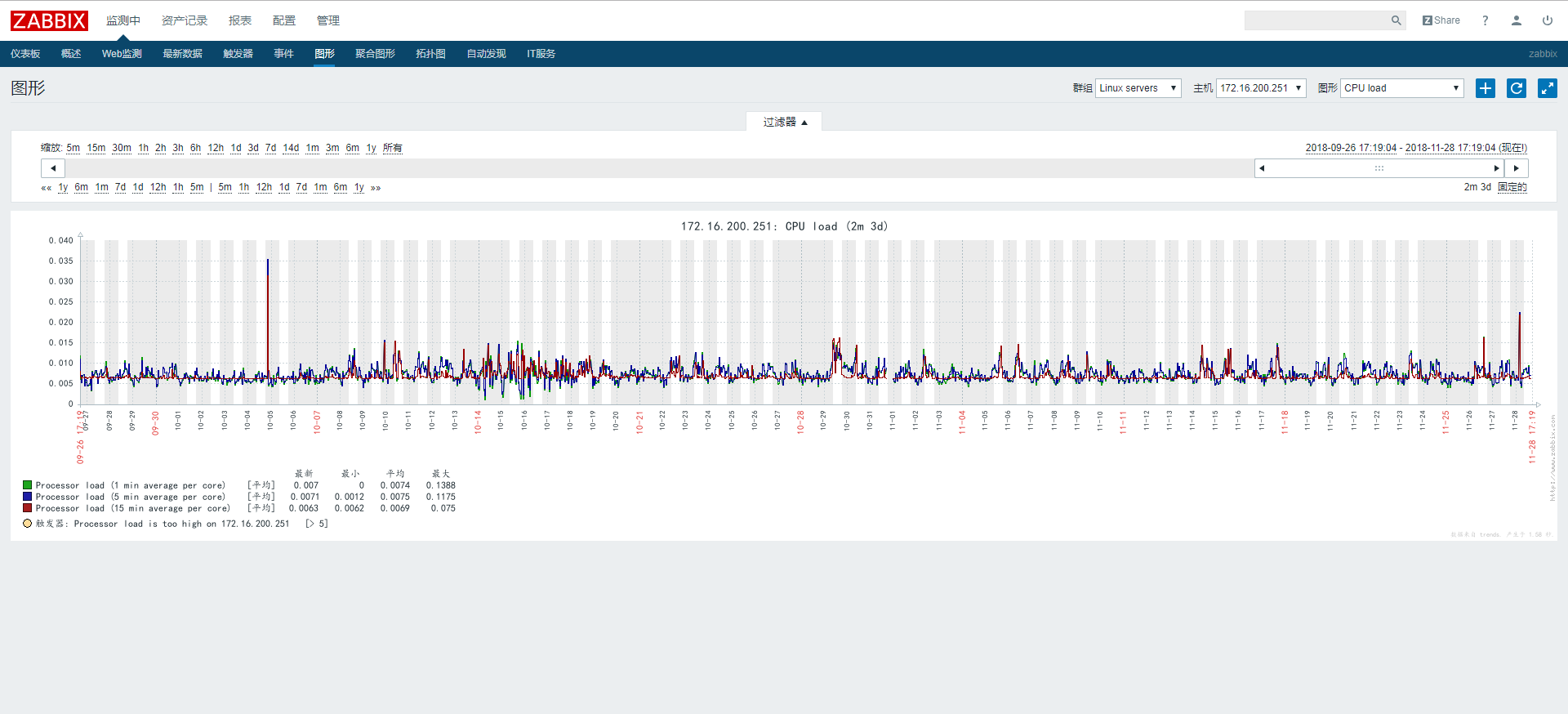
**聚合图形展示：**



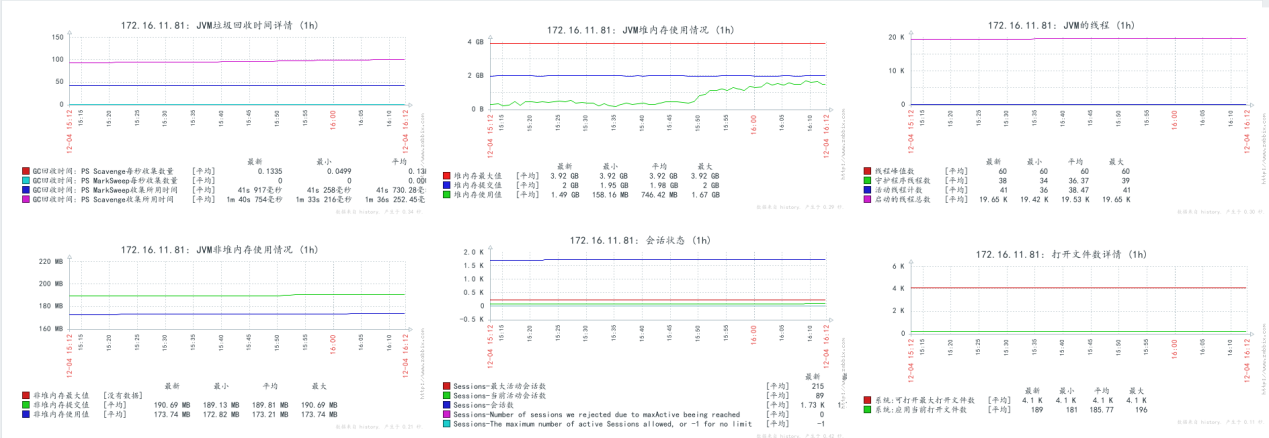
**触发器配置：**



**详细数据图形展示：**



### 6.2.2 应用层监控



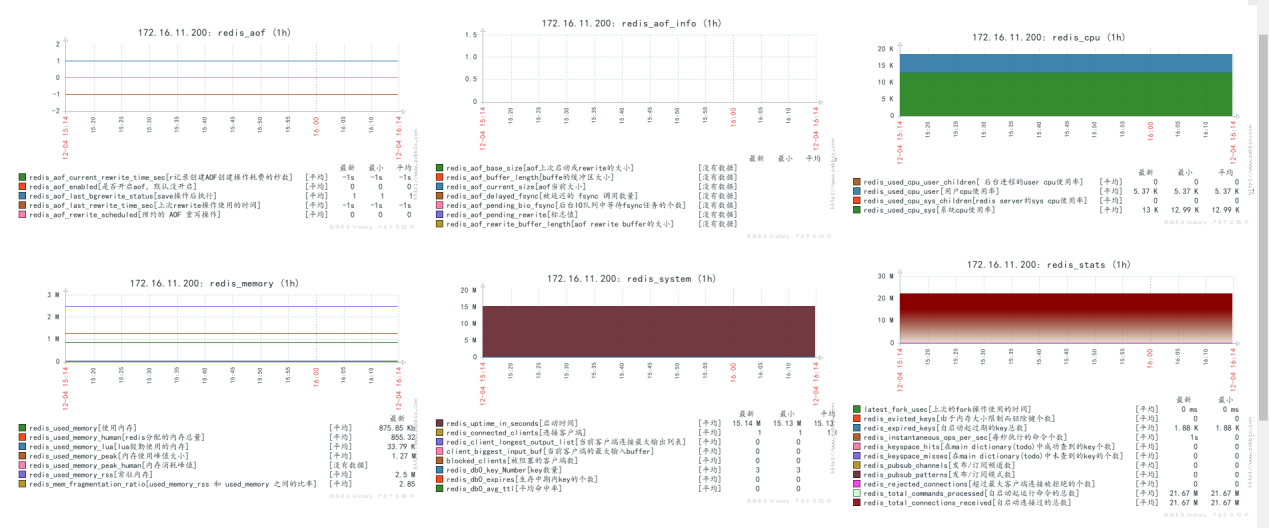
Tomcat整体运行情况监控。包括：

JVM内存使用情况

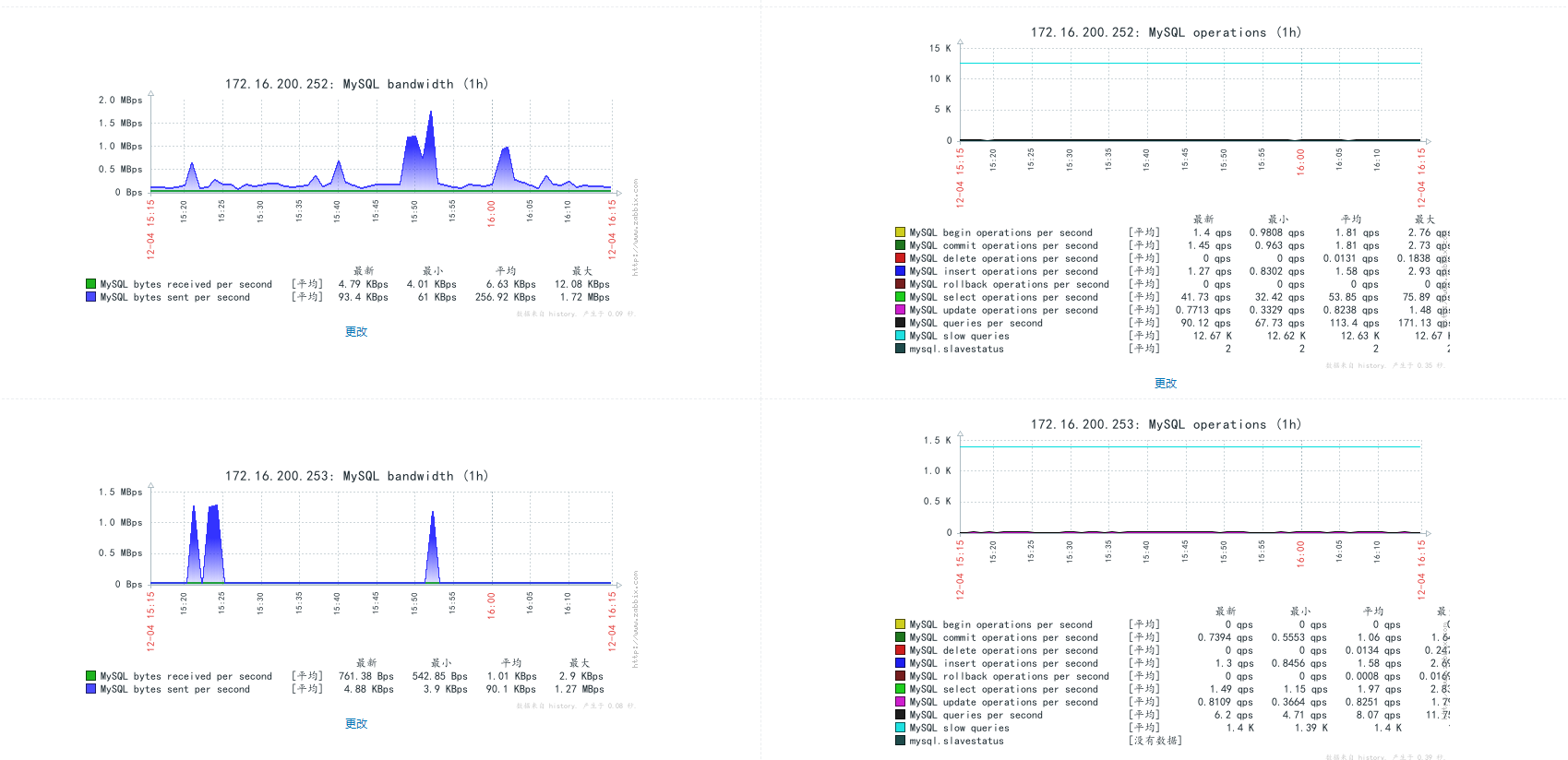
JVM线程

会话状态

打开文件状态



redis监控：



Oracle监控

针对Oracle情况，状态，流量等监控

### 6.2.3 业务层监控

基于ELK架构的日志分析

一般我们需要进行日志分析场景：直接在日志文件中 grep、awk 就可以获得自己想要的信息。但在规模较大的场景中，此方法效率低下，面临问题包括日志量太大如何归档、文本搜索太慢怎么办、如何多维度查询。需要集中化的日志管理，所有服务器上的日志收集汇总。常见解决思路是建立集中式日志收集系统，将所有节点上的日志统一收集，管理，访问。

一般大型系统是一个分布式部署的架构，不同的服务模块部署在不同的服务器上，问题出现时，大部分情况需要根据问题暴露的关键信息，定位到具体的服务器和服务模块，构建一套集中式日志系统，可以提高定位问题的效率。

一个完整的集中式日志系统，需要包含以下几个主要特点：

收集－能够采集多种来源的日志数据

传输－能够稳定的把日志数据传输到中央系统

存储－如何存储日志数据

分析－可以支持 UI 分析

警告－能够提供错误报告，监控机制

ELK提供了一整套解决方案，并且都是开源软件，之间互相配合使用，完美衔接，高效的满足了很多场合的应用。目前主流的一种日志系统。

ELK简介：

ELK是三个开源软件的缩写，分别表示：Elasticsearch , Logstash, Kibana , 它们都是开源软件。新增了一个FileBeat，它是一个轻量级的日志收集处理工具(Agent)，Filebeat占用资源少，适合于在各个服务器上搜集日志后传输给Logstash，官方也推荐此工具。

Elasticsearch是个开源分布式搜索引擎，提供搜集、分析、存储数据三大功能。它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。

Logstash 主要是用来日志的搜集、分析、过滤日志的工具，支持大量的数据获取方式。一般工作方式为c/s架构，client端安装在需要收集日志的主机上，server端负责将收到的各节点日志进行过滤、修改等操作在一并发往elasticsearch上去。

Kibana 也是一个开源和免费的工具，Kibana可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助汇总、分析和搜索重要数据日志。

Filebeat隶属于Beats。目前Beats包含四种工具：

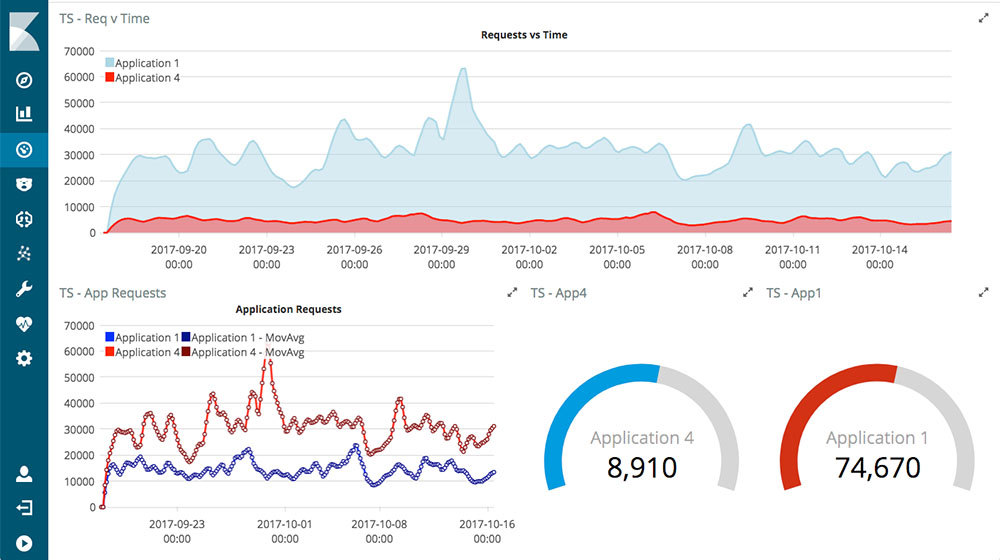
Packetbeat（搜集网络流量数据）

Topbeat（搜集系统、进程和文件系统级别的 CPU 和内存使用情况等数据）

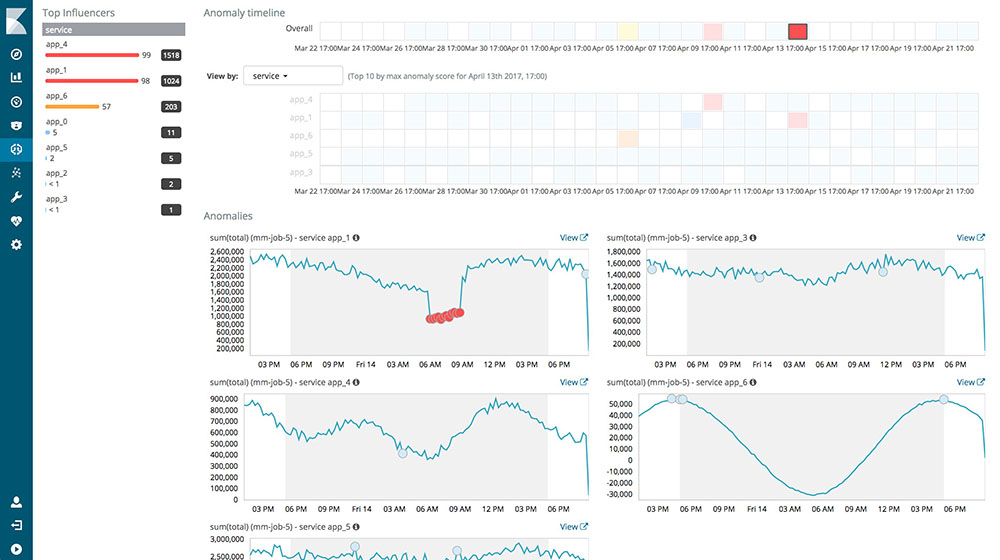
Filebeat（搜集文件数据）

Winlogbeat（搜集 Windows 事件日志数据）

* 强大的图表分析



机器学习发现潜在危险



## 6.3 备份

### 6.3.1 网络备份

网络备份，主要备份网络设备配置文件，周期为每周至少两次。

### 6.3.2 应用备份

应用才用多地备份，生产环境备份保存一周，每天通过定时任务，备份到公司内部环境，备份生产压缩包通过VPN传到公司内部环境，生成压缩包需进行MD5加密，在公司内部环境进行校验，校验通过完成备份。

备份流程：

应用备份保留一个月

数据库备份保留永久

# 七、接口定义

## 7.1 存入资金银行回执信息

### 7.1.1接口方向

银行->平台

### 7.1.2接口描述

由银行端调用，用于记录资金存入相关信息

### 7.1.3接口地址

待定

### 7.1.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| caseNo | body | 字符串 | 案件编号 |
| zzh | body | 浮点数 | 存入的子账户 |
| deposit | body | 浮点数 | 存入金额 |
| state | body | 字符串 | 存入结果状态 |

### 7.1.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {}

}

}

## 7.2 存入资金查询接口

### 7.2.1接口方向

平台->银行

### 7.2.2接口描述

由银行端调用，用于查询资金存入相关信息

### 7.2.3接口地址

待定

### 7.2.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| caseNo | body | 字符串 | 案件编号 |

### 7.2.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {

caseNo:2019014578963,

accountNo:6335897222322,

accountName:张三，

deposit：1450000

}

}

}

## 7.3资金划拨转款后回执消息

### 7.3.1接口方向

平台->外部

### 7.3.2接口描述

由银行端调用，用于开发企业取款申请成功后发送给平台的相关信息

### 7.3.3接口地址

待定

### 7.3.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| accountNo |  |  | 账户号 |
| zzh | body |  | 子账户号（多个用逗号隔开） |
| hbje | body |  | 划拨金额 |
| result | body |  | 转账结果 |

### 7.3.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {}

}

}

## 7.4 资金划拨查询接口

### 7.4.1接口方向

平台->银行

### 7.4.2接口描述

由银行端调用，用于查询资金划拨相关信息

### 7.4.3接口地址

待定

### 7.4.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| caseNo | body | 字符串 | 案件编号 |
| qtype | body | 字符串 | 查询类型（0-存入，1-划拨，2-退款，3-注销） |

### 7.4.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {

caseNo:2019014578963,

accountNo:6335897222322,

accountName:张三，

hbje：1450000

}

}

}

## 7.5申请退款后回执消息

### 7.5.1接口方向

银行-》平台

### 7.5.2接口描述

由银行端调用，用于开发企业退款申请成功后发送给平台的相关信息

### 7.5.3接口地址

待定

### 7.5.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| caseNo | body | 字符串 | 案件编号 |
| refund | body | 字符串 | 退款金额 |
| status | body | 字符串 | 退款状态 |

### 7.5.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {}

}

}

## 7.6 申请退款查询接口

### 7.6.1接口方向

平台->银行

### 7.6.2接口描述

由银行端调用，用于查询申请退款相关信息

### 7.6.3接口地址

待定

### 7.6.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| caseNo | body | 字符串 | 案件编号 |
| qtype | body | 字符串 | 查询类型（0-存入，1-划拨，2-退款，3-注销） |

### 7.6.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {

caseNo:2019014578963,

accountNo:6335897222322,

accountName:张三，

tkje：1450000，

reason:正常原因

}

}

}

## 7.7账户注销后回执消息

### 7.7.1接口方向

银行->平台

### 7.7.2接口描述

由银行端调用，用于开发企业账户注销申请成功后发送给平台的相关信息

### 7.7.3接口地址

待定

### 7.7.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| caseNo | body | 字符串 | 案件编号 |
| state | body | 字符串 | 注销状态 |

### 7.7.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {}

}

}

## 7.8 账户注销查询接口

### 7.8.1接口方向

平台->银行

### 7.8.2接口描述

由银行端调用，用于查询账户相关信息

### 7.8.3接口地址

待定

### 7.8.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| caseNo | body | 字符串 | 案件编号 |
| qtype | body | 字符串 | 查询类型（0-存入，1-划拨，2-退款，3-注销） |

### 7.8.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {

caseNo:2019014578963,

accountNo:6335897222322,

accountName:张三，

reason:正常注销

}

}

}

## 7.9账户利息数据更新

### 7.9.1接口方向

银行->平台

### 7.9.2接口描述

由银行端调用，用于开发企业账户利息数据的更新

### 7.9.3接口地址

待定

### 7.9.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| accountNo | body |  | 监管账号 |
| interest | body |  | 利息金额 |

### 7.9.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {}

}

}

## 7.10交易冲正数据记录

### 7.10.1接口方向

银行->平台

### 7.10.2接口描述

由银行端调用，用于银行定期推送冲正记录到平台中。

### 7.10.3接口地址

待定

### 7.10.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| accountNo | body | 字符串 | 监管账号 |
| yje | body | 浮点型 | 原账户金额 |
| remainder | body | 浮点型 | 账户余额 |
| detail | body | 字符串 | 冲正详情 |
| czyy | body | 字符串 | 冲正原因 |

### 7.10.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {}

}

}

## 7.11手续费充值记录

### 7.11.1接口方向

银行->平台

### 7.11.2接口描述

由银行端调用，用于银行推送手续费充值记录到平台中。

### 7.11.3接口地址

待定

### 7.11.4接口请求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点名称** | **父节点名称** | **类型** | **说明** |
| appKey | body | 字符串 | 分配给接口调用者的唯一标识 |
| sign | body | 字符串 | 签名，根据appSecret对所有传递的参数进行md5加密后的字符串 |
| timestamp | body | long | 当前时间戳(毫秒数) |
| accountNo | body | 字符串 | 监管账号 |
| czje | body | 浮点型 | 充值金额 |
| bz | body | 字符串 | 备注 |

### 7.11.5接口响应

{

"status": "000",

"msg": "成功",

"data": {

"obj": {}

}

}