|  |  |
| --- | --- |
| 日期： | 2019-05-06 |
| 密级： |  |
| 版本： | 1.0.0 |
| 编号： | YR-HD-19050601 |

天鹰智能判别与预警系统

概要设计说明书

2019年5月

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2019-05-06 | 1.0.0 | 初稿 | 海纳天鹰 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目 录

[1.引言 1](#_Toc8980634)

[1.1 编写目的 1](#_Toc8980635)

[1.2 背景 1](#_Toc8980636)

[1.3 术语定义 3](#_Toc8980637)

[1.4 参考资料 3](#_Toc8980638)

[2.系统总体设计 4](#_Toc8980639)

[2.1设计概述 4](#_Toc8980640)

[2.1.1 限制和约束 4](#_Toc8980641)

[2.1.2 设计原则和设计要求 5](#_Toc8980642)

[2.2 系统架构设计 5](#_Toc8980643)

[2.3 子系统定义 6](#_Toc8980644)

[2.3.1 系统主控子系统 6](#_Toc8980645)

[2.3.2 规则引擎子系统 6](#_Toc8980646)

[2.3.3 规则管理与控制子系统 6](#_Toc8980647)

[2.3.4 结果管理与控制子系统 6](#_Toc8980648)

[3.子系统设计 6](#_Toc8980649)

[3.1 系统主控子系统 6](#_Toc8980650)

[3.1.1 系统配置模块 7](#_Toc8980651)

[3.1.2 组件管理模块 7](#_Toc8980652)

[3.1.3 规则监视与载入模块 7](#_Toc8980653)

[3.1.4 引擎管控模块 8](#_Toc8980654)

[3.2 规则引擎子系统 8](#_Toc8980655)

[3.2.1 数据获取模块 8](#_Toc8980656)

[3.2.2 分析匹配模块 9](#_Toc8980657)

[3.2.3 组装流转模块 10](#_Toc8980658)

[3.3 规则管理与控制子系统 11](#_Toc8980659)

[3.3.1 规则增删改查模块 11](#_Toc8980660)

[3.3.3 规则控制模块 11](#_Toc8980661)

[3.3.2 规则编辑UI 11](#_Toc8980662)

[3.4 结果管理与控制子系统 11](#_Toc8980663)

[3.4.1 结果增删改查模块 11](#_Toc8980664)

[3.4.2 结果统计查询UI 11](#_Toc8980665)

[3.4.3 结果控制处理UI 11](#_Toc8980666)

[4.公共组件设计 11](#_Toc8980667)

[5.非功能性需求实现方案 12](#_Toc8980668)

[6.技术与数据结构设计（难点及解决方案） 12](#_Toc8980669)

[7.日志与异常处理设计 18](#_Toc8980670)

[8.数据库设计 18](#_Toc8980671)

[9.附录 18](#_Toc8980672)

# 1.引言

引言是对这份概要设计报告的概览，是为了帮助阅读者了解这份文档是如何编写的，并且应该如何阅读、理解和解释这份文档。

## 1.1 编写目的

本概要设计文档主要用于指导天鹰智能判别与预警系统的详细设计工作，为详细设计提供统一的参照标准，其中包括系统的总体架构、子系统及内外部接口设计、编程模型以及其他各种主要技术与问题的解决方案。在此文档被评审后，所有有关本系统的详细设计必须遵照此文档相关标准和约束来进行。另外，此文档也作为对详细设计文档进行评审所依照的标准之一。

在详细设计的过程中，如果发现需要添加新的概要设计标准或者约束来指导详细设计工作，必须对此文档进行更新和评审，以确保各模块详细设计的一致性和正确性。

本文档主要描述的是天鹰智能判别与预警系统的概要设计，其中包括定义系统总体架构、子系统及其内外部接口设计、相关的技术方案和设计标准、数据库结构和日志设计、异常处理，不会涉及系统业务逻辑实现的细节。

## 1.2 背景

环境监测预警、监视、示警的重要性

目前现状，当前做法，优缺点

天鹰智能判别与预警系统的思路、目标，技术路线，应用场景与效益

目标：核心为一套智能规则引擎系统，一套多领域专家知识库，一套智能算法库，一套特征模型库

环境监测预警体系：

说清污染源排放状况；

说清环境质量状况及变化趋势；

说清环境突发事故及潜在环境风险。

目标：由“先污染，后处理”向“源头防治、过程监管、末端监控”转变。

源头防治、统一监管，过程监管、全防全控，末端监控、防范风险，图文交互、智能决策。

事先，预测预判预警，防范检查准备

事中，监视告警控制，应急响应反馈

事后，评价评估示警，分析总结归纳；合规项、不合规项

螺旋式上升，逐步调整、适应、收敛

从数据角度看：数据的分类，数据的来源（何处取，如何取），不同类型数据的统计分析预测预判（怎么用）；

按数据的来源分类：

按数据的关注者、使用者分类：

数据的应用场景

从不同用户角度看：监管政府，排污企业，仪器厂家，检测检验机构，服务机构

从人、机、料、法、环、测等管理要素角度：

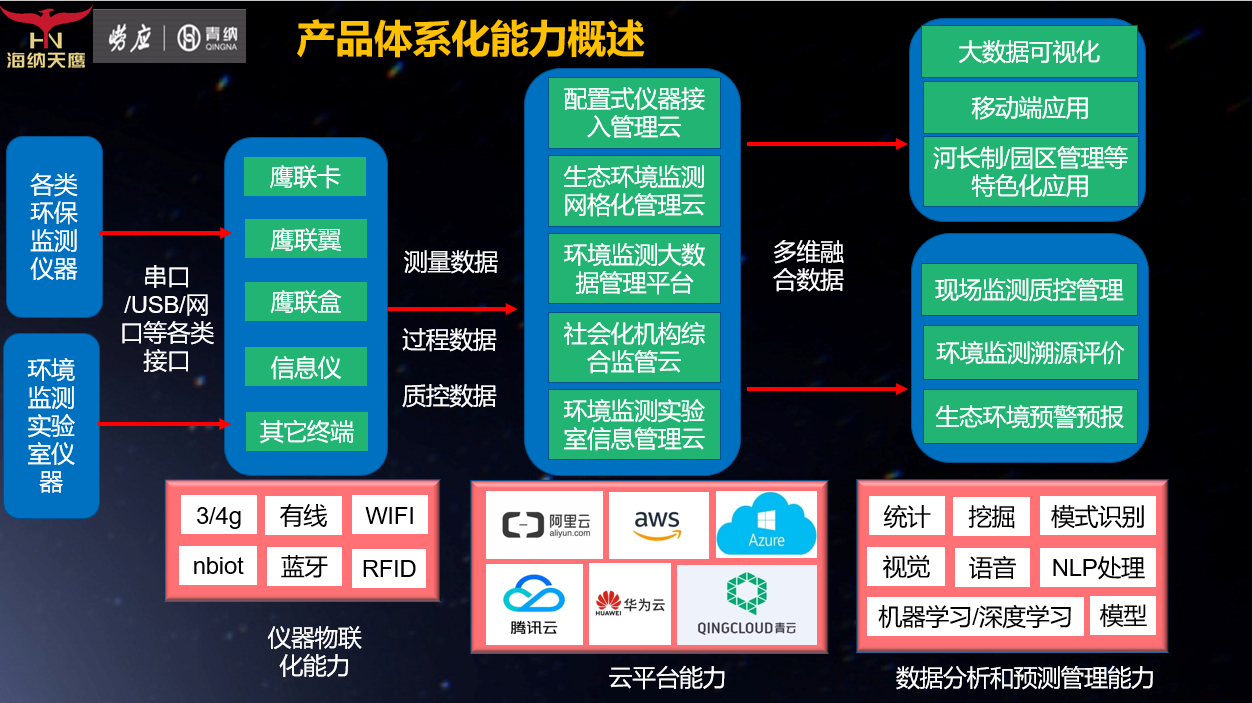
**智慧环保应用的快速发展也推动传统的便携环保仪器、在线式监测站、现场环保监测活动向更加便捷部署、快速移动、智能感知、一体化融合、综合平台运营的方式转化！**

**国家新一代人工智能发展规划、信息化发展战略、物联网新兴产业技术、创新驱动发展支撑手段为环保仪器装备的发展、融合、应用提供了大量新的赋能手段！**

**环境监测现场质控服务商：充分结合物联网和环境监测质控技术，实现对环保监测仪器的物联网升级，推进现场监测质控建设发展，助力传统实验室信息管理LIMS系统的质控升级，协助管理部门优化对环境监测的质控管理水平，完善对社会化机构的监管。**

**综合生态应用信息化服务商：充分应用物联网技术、大数据分析技术、人工智能技术和遥感技术的融合，提供以网格化生态管理系统、河长制系列管理系统、智慧环保系统、智慧水质监测系统、智慧城市系统为代表的综合生态应用信息化服务。**

**特色化仪器和应用供应商：突破光谱技术、卫星通讯技术、MEMS系统融合技术等，打造鹰联仪、新型光谱仪器、无人值守化微型综合监测系统的特色化仪器，并构建提供化工园区管理、多类型实验室综合管理、排污口监测与管理等特色化应用服务。**



就像互联网应用领域非常流行的概念云、网、端，端在这里就是我们的环保仪器设备，网就是物联网，云就是我们的云端，大数据，再加上智就是我们的智慧分析应用。智、云、网、端四者不是割裂的，是相互融合的整体。先是有各式仪器终端，然后端连接成网，网酝酿出云，云产生出智。

从相关标准规范、相关领域专家、客户用户，搜集、分析、整理，询问、捕捉、挖掘，形成零散的拟专家经验库；对库进行分门别类、归纳梳理，对每条经验或规则进行统一的自然语言规范性重组，形成分级分类、条理清晰、格式统一的自然语言规则库；该规则库可以不断积累扩展。

从各类应用系统、业务系统中抽取出，独立的数据监视、分析、告警服务。

微服务架构是一个分布式的应用技术架构，目的是有效的对应用进行拆分，实现敏捷开发、快速演化、便捷容错与弹性伸缩。区块链技术本质上就是分布式数据库，微服务架构与区块链技术的结合，并不能简单的看成是微服务与数据库的结合，而应该把区块链平台做为一个第三方应用进行交互，这也是微服务架构很好发挥作用的地方。

天鹰智能判别与预警系统是一个通用、灵活、独立的数据分析、判别推理和预警中心。系统以规则形式，描述待分析推理数据的获取方式、判别推理表达式、推理结果转换及流转方式。通过规则引擎运转规则，实现目标数据的分析推理和结果组装发送。

发生某事件或达到某临界条件或满足某判断条件

通用性、灵活性，

## 1.3 术语定义

|  |  |  |
| --- | --- | --- |
| **术语缩写** | **术语全称** | **中文翻译(供参考)** |
| CRUD | Create/Retrieve/Update/Delete | 增删改查(四种基本的数据操作) |
| BP | Business Process | 业务过程 |
| BO | Business Object | 业务对象 |
| VO | Value Object | 值对象 |
| MVC | Model/View/Controller | 模型/表示/控制模式 |
| DAO | Date Access Object | 数据访问对象 |
| DTO | Date Transfer Object | 数据传输对象 |

## 1.4 参考资料

列举编写软件产品概要设计报告时所用到的参考文献及资料，可能包括：

* 本项目的合同书；
* 上级机关有关本项目的批文；
* 本项目已经批准的计划任务书；
* 用户界面风格指导；
* 开发本项目时所要用到的标准；
* 系统规格需求说明；
* 使用实例文档；
* 属于本项目的其它已发表文件；
* 本软件系统概要设计报告中所引用的文件、资料：
* 相关软件系统概要设计报告：
* 等等。

为了方便读者查阅，所有参考资料应该按一定顺排列。如果可能，每份资料都应该给出：

* 标题名称；
* 作者或者合同签约者；
* 文件编号或者版本号；
* 发表日期或者签约日期；
* 出版单位或者资料来源。

# 2.系统总体设计

## 2.1设计概述

本节描述现有开发条件和需要实现的目标，说明进行概要设计时应该遵循的设计原则和必须采用的设计方法。

### 2.1.1 限制和约束

简要描述起到限制和约束作用的各种可能存在的条件，例如：

* 技术条件；
* 资金状况；
* 开发环境(包括：工具和平台)；
* 时间限制；
* 等等。

并且说明在上述条件下，应该实现的系统目标，

2.1.1.1 运行环境

1 软件环境

2 硬件环境

2.1.1.2 开发环境

1 服务器软件环境

2 服务器硬件环境

3 开发机器软件环境

4 开发机器硬件环境

### 2.1.2 设计原则和设计要求

描述对本软件系统进行概要设计的原则，通常可以考虑以下几方面的内容：

* 命名规则；
* 模块独立性原则：
* 边界设计原则；
* 数据库设计规则；
* 必须的安全措施；
* 安全性和保密原则；
* 系统灵活性要求；
* 系统易操作性要求；
* 系统可维护性要求；
* 等等。

## 2.2 系统架构设计

数据的抽取、转换、加载，包括查询、调取、传输、接收、转换、加载。

数据的分析、推理、判断、匹配、统计，形成综合结果。

预警、评价、预测，结果的存储、推送、展示。

事先预判预警

事中监视告警

事后评估示警



图2.1 系统总体架构

一个主控中心，N个核心规则引擎，两类组件，两个库，两类管控子系统。

分析目标数据源，规则管控人员，结果关注及管控人员。

1.系统启动

2.规则变化（增删改查），由规则管控人员发起

改：改变规则内容，包括选择规则手动运行

3.结果变化，由结果管控人员发起

## 2.3 子系统定义

### 2.3.1 系统主控子系统

主控子系统是系统的启动入口、核心控制中心，包括系统配置、组件管理、规则监视与载入、引擎管控四个模块，子系统的功能包括：

（1）读取系统配置文件，根据配置进行系统初始化，准备包括数据源链接、数据链路、消息系统链接等在内的各类资源和交互组件；

（2）从系统配置的规则库中加载所有规则；

（3）对每一规则生成一个规则引擎，纳入引擎管控，调度规则引擎的执行；

（4）监听规则的变化，根据变化情况管理规则（增删改）和对应的规则引擎（增删改）；

（5）监控和管理各规则引擎。

系统配置：系统初始启动入口。

引擎管控：引擎的增删改查、启停暂续，定时任务调度管理

### 2.3.2 规则引擎子系统

规则引擎（数据获取，分析匹配，组装流转）

包括数据获取、分析匹配、组装流转三个模块。

2.3.2.1 数据获取

2.3.2.2 分析匹配

2.3.2.3 组装流转

### 2.3.3 规则管理与控制子系统

规则管控（规则查询与编辑UI和API，规则控制）

规则控制：规则增删改查，对应规则引擎状态查询与启停暂续控制，选择多个规则立即单次运转。

### 2.3.4 结果管理与控制子系统

结果管控（存储，增删改查）

# 3.子系统设计

## 3.1 系统主控子系统

系统主控子系统结构如图3.1所示。



图3.1 系统主控子系统结构图

系统主控子系统触发运行：

1.系统启动

加载、启动所有run\_mode为auto的规则；

2.规则增删改查

增加规则->新建引擎，

删除规则->停止并清除引擎

改（如何知道已改了？）

查（查规则内容、查规则状态）

规则控制（启停暂续）

保持内存中规则与规则库中规则的一致性：规则管理与控制

内存中规则变化时，触发引擎管理与控制

### 3.1.1 系统配置模块

输入：系统配置文件

输出：

处理流程：读取application配置文件，连接数据源

### 3.1.2 组件管理模块

资源组件：获取分析数据的资源组件。关系型/非关系型数据库组件，stomp客户端组件，消息系统接收组件，http api接口访问组件，其它网络协议组件。

交互组件：SMS、stomp服务端组件、WeChat推送组件、eMail推送组件、消息系统发送组件。

### 3.1.3 规则监视与载入模块

规则验证

规则载入

规则状态查询

规则变化与监视，监视规则内容变化（增删改），

规则内容查询

规则控制

### 3.1.4 引擎管控模块

引擎增删，启停暂续

引擎状态，生命周期管理

周期定时任务调度管理

## 3.2 规则引擎子系统

引擎控制的粒度，整体控制还是更细粒度的过程控制

### 3.2.1 数据获取模块

支持从多种数据源获取/采集数据，支持push和pull两种数据获取模式；针对不同的数据源，提供多种数据获取方式。

支持以下数据获取方式：

* 调用外部HTTP API方式周期性主动获取数据；
* 连接外部websocket/stomp服务器，被动式接收数据
* 连接外部消息系统或协议服务器，被动式接收数据
* 直接从外部数据库或缓存，主动获取数据
* 提供API供外部调用，被动式接收数据
* 调用内部API或本地脚本，主动获取数据

先实现两种方式：

http/webservice api 调用，主动式获取数据。pull模式，周期性运行，获取本周期内的最新数据，交由分析匹配模块进行分析推理告警。

连接外部stomp服务器，被动式接收数据。push模式，收到数据时，即触发后续分析推理告警。

第二步实现两种方式：

本地local api或脚本调用，主动式

消息系统连接，被动式

最后实现有难度的方式：直接从数据库查询获取（主动式，探针驱动式），提供api供外部调用（被动式）

查询或过滤条件可以考虑使用Hibernate的Criteria

探针的作用就是通过调用应用或系统接口的方式采集监控数据并对应成指标返回给prometheus server。（探针不一定要和监控的应用部署在一台机器）

总的来说Prometheus数据采集流程就是，在Prometheus server中配置探针暴露的端口地址以及采集的间隔时间，Prometheus按配置的时间间隔通过http的方式去访问探针，这时探针通过调用接口的方式获取监控数据并对应指标返回给Prometheus server进行存储。（若探针在Prometheus配置的采集间隔时间内没有完成采集数据，这部分数据就会丢失）

获取的原始数据，要经过过滤器Filter处理，根据规则对不完整、错误、重复或满足/不满足要求的数据进行清洗、过滤。

过滤后的数据，进行一定的转换，进行某种预处理（如加上数据接收时刻、数据条目数，等等），变换为统一的数据格式

### 3.2.2 分析匹配模块

警情，警兆

info、notice、warning（警示，提醒注意，使警惕）、Crit、alert（警告）、emergency（紧急情况，需要立即处理）

info：表示有这样一些信息，想关注则关注，不想关注可以不看，等回头再查

notice：表示通知需要关注的事件发生了，提醒关注者多注意

warning：表示通过推理判断，后续可能会出现问题，警示关注者

crit：表示通过推理判断，已达临界点，如果不做处理改变，后续即将出现问题，严重警示关注者

alert：表示已经出现了问题，向关注者报警，请其处理

emergency：表示出现了紧急问题，向关注者严重告警，必须立即响应处理

### 3.2.3 组装流转模块

报警方式：DB，Cache，Debug，Stomp，SMS，Email，WeChat，RPC，LPC，Custom

分级、分类

定制报警通知模板

支持以下告警方式：

分组：将监控目标相同的告警进行分组。如发生停电，收到的应该是单一信息，信息中包含所有受影响宕机的机器，而不是针对每台宕机的机器都发送一条告警信息。

抑制：抑制是指当告警发出后，停止发送由此告警引发的其他告警的机制。如机器网络不可达，就不再发送因网络问题造成的其他告警。

静默：根据定义的规则过滤告警信息，匹配的告警信息不会发送。

延时：根据需求，延迟一定时间或指定时刻再发送告警信息，去重。

不希望频繁的收到重复的告警消息怎么办？Repeat interval

需要及时发送告警消息？Group interval

故障刚发生时，接连收到几个告警消息怎么办？Group wait

告警的对接：多样告警源，多元告警目标。大多数告警通过监控系统发出，有部分告警可以通过服务直接发出，所以我们希望支持多样的告警源。对于多样告警目标，不同公司可能用的办公软件都不一样，有的公司用微信进行通讯，有的公司通过钉钉进行接收消息，而客户希望我们把告警发到他们聊天工具中去。不同人员的需求也不同，运维人员习惯通过短信接收告警，大BOSS更喜欢用邮件接收告警，所以我们告警对接需要解决的第二个问题是多样的告警目标。

告警的收敛：大家是否遇到这样问题，收到一个告警然后开始排查问题，但是排查问题过程中告警消息不停地发送过来。处理故障就是精神高度紧张的时候，重复告警消息发过来，不仅对解决问题没有任何帮助，反而会增加运维人员压力。所以我们要收敛过多的告警消息。假设有一台服务器挂掉了，这台服务器首先会发送一个告警告诉你这台服务器挂掉了，这台服务器上面运行其他服务也被监控系统给监测到了，并且这些服务的告警消息也会发出来。但是实际上只有服务器挂掉这一条信息能帮助我们解决问题，所以我们告警的收敛解决关联告警过多的问题。关于运维期间不希望收到告警主要是因为运维通常会在大半夜进行，运维期间很有可能会产生一些告警。如果大BOSS接到报警马上打电话过来，这个压力可不小。所以这个时间段运维一般不希望把告警发出来，因此告警收敛需要解决的第三个问题是运维期间不希望收到告警。

告警的可用性：假如长时间没有收到告警消息，你们是会认为自己系统运行的很完美，还是会担心告警系统挂掉了。如果是告警系统挂掉了，不能及时把告警发出来，那么最后这个锅到底由谁来背。大家都不希望背锅，所以我们必须要实现告警系统的高可用。关于第二个隔离的故障域，有的监控系统和告警系统绑定在一起，如果监控系统挂掉，告警系统同样挂掉，所以我们希望监控系统和告警系统是分开部署的。

按分析推理规则需求，可能缓存前一时间窗口的原始数据或分析结果

Prometheus

Open-falcon

Zabbix

## 3.3 规则管理与控制子系统

### 3.3.1 规则增删改查模块

增删改规则内容

查规则内容

查规则运行状态（启动时间、运行时长、运转次数，未启动、运行中、已停止、已暂停）

查规则间依赖关系

### 3.3.3 规则控制模块

手动启动规则

停止规则

暂停/恢复规则

### 3.3.2 规则编辑UI

## 3.4 结果管理与控制子系统

### 3.4.1 结果增删改查模块

### 3.4.2 结果统计查询UI

### 3.4.3 结果控制处理UI

# 4.公共组件设计

WebSocket Stomp服务端

WebSocket Stomp客户端

Quartz任务管理调度

缓存组件

SMS组件

eMail组件

微信组件

MySQL数据库监控组件canal

文件监控组件（Windows/Java）

# 5.非功能性需求实现方案

安全性（安全管理）

并发性

可靠性（可靠性是与在规定的一段时间和条件下软件维持其性能水平的能力有关的一组属性。具体包括：• 成熟性：与有软件故障引起失效的频度有关的软件属性。 • 容错性：与在软件故障或违反指定接口的情况下维持规定的性能水平的能力有关的软件属性。如离线录入支持等。 • 易恢复性：与在是小发生后重建其性能水平并恢复直接受影响数据的能力，以及为达到此目的所需时间和努力有关的软件属性。如表单数据自动保存等。 这类非功能需求通常是全局的，他除了与系统运行环境、平台选择、代码质量相关之外，还会涉及部分技术性功能需求，特别是容错性、易恢复性的实现都需要一些具体的功能来支持。）

易使用性

可维护性

互操作性

健壮性

可重用性

可扩充性

# 6.技术与数据结构设计（难点及解决方案）

6.1 系统配置项设计

|  |  |  |  |
| --- | --- | --- | --- |
| 配置项名 | 配置项值类型 | 配置项说明 | 备注 |
| placement | 字符串 | 规则存放方式。file：文件式，db：数据库式 |  |
| uri | 字符串 | 规则存放地址。placement为file时，表示规则文件夹名，placement为db时，表示数据库表或API |  |
| rule\_monitoring | 布尔 | 表示是否监听规则的变化，如为true，规则变化时，相应规则引擎会重新载入运行 |  |
| type\_mappings | fetchers | fetcher、analyzer、transmitter的类型码与具体处理类的映射关系 |  |
| analyzers |
| transmitters |
| system\_resources | 列表 | 系统内部各资源组件的定义 |  |

6.2 规则配置项（Rule）设计

规则状态：启动时间，运行状态（未启动、运行中、已停止、已暂停）

规则控制：启动、停止、暂停/恢复

规则包含四部分：

1.规则总体部分

id、name、description、domain、category、priority

data\_fields：定义规则引擎内部使用的数据字典，key名，value类型，含义

run\_mode：自动（自动运行，自动获取数据）、纯手动（只能由人工启动）、触发式手动（当发生某事件或出现某结果时启动，该事件可以是另一个规则的推理结果，难点：如何建立两个规则间的数据关联）

frequency\_mode：单次，持续

cron\_entry：frequency\_mode为连续时，定义引擎运转周期

run\_trigger：启动触发器，run\_mode为触发式手动时，定义触发条件/触发器

stop\_trigger：停止触发器，frequency为持续时，定义停止条件/触发器

2.数据获取部分

先实现两种方式：

http/webservice api 调用，主动式

websocket/stomp连接，被动式

第二步实现两种方式：

本地local api或脚本调用，主动式

消息系统连接，被动式

最后实现有难度的方式：直接从数据库获取（主动式），提供api供外部调用（被动式）

data\_mappings：规则中使用的内部数据项和数据源各数据项间的映射关系。

data\_format：数据源获取到的数据的格式，json、xml、map、pojo、binary

data\_converter：内部统一使用json格式，若为其它格式，需将其它格式数据转换为json

3.分析匹配部分

4.组装流转部分

规则总体描述

|  |  |  |  |
| --- | --- | --- | --- |
| 配置项名 | 配置项值 | 配置项说明 | 备注 |
| id | 字符串 | 自动生成的唯一id |  |
| name | 字符串 | 唯一规则名，每个规则有自己唯一独立name |  |
| version | 版本号 |  |  |
| description | 字符串 | 规则详细描述 |  |
| run\_mode | 字符串 | 自动（自动运行，自动获取数据）、纯手动（只能由人工启动）、触发式手动（当发生某事件或出现某结果时启动，该事件可以是另一个规则的推理结果，难点：如何建立两个规则间的数据关联），只允许一次嵌套  例如：现场采样任务确定或下达时，对于仪器超出校准期限、人员进行报警  “现场采样任务确定或下达”为一个规则。该规则周期运行，其结果会触发“仪器超出校准期限”的规则运转。后一规则的数据获取查询条件包括现场采样任务的编号，即只对该任务相关的仪器进行超出校准期限的判别预警。  自动：auto，默认  手动：manual  触发式：triggered |  |
| frequency\_mode | 字符串 | 单次  连续（默认） |  |
| cron\_entry | CRON表达式 | frequency\_mode为连续时，定义引擎运转周期 |  |
| data\_fields | 字符串列表 | 数据结构/属性定义。用${field}引用属性值。预定义属性：count->数据条数，now->当前时间，用^{count}引用属性值。定义规则引擎内部使用的数据字典，key名，value类型，含义 | data\_fields:  - userId  - userName  - phoneNumber  - taskId  - taskName  - taskStartTime  - taskEndTime |
| run\_trigger |  | 启动触发器，run\_mode为触发式手动时，定义触发条件/触发器 | 如何表示嵌套规则或聚合规则 |
| stop\_trigger |  | 停止触发器，frequency为持续时，定义停止条件/触发器 |  |
| priority | 整数 | 报警优先级 |  |
| domain | 字符串 | 报警领域 |  |
| category | 字符串 | 报警领域下的类别 |  |

数据获取规则描述

|  |  |  |  |
| --- | --- | --- | --- |
| fetcher\_type | 字符串 | 数据获取方式。  先实现两种方式：  http/webservice api 调用，主动式  websocket/stomp连接，被动式  第二步实现两种方式：  本地local api或脚本调用，主动式  消息系统连接，被动式  最后实现有难度的方式：直接从数据库获取（主动式），提供api供外部调用（被动式） |  |
| data\_mappings | 列表 | 获取到的数据与系统内部数据项间的映射关系 |  |
| data\_format | 字符串 | 数据格式 |  |
| fetcher\_target | 字符串 | 数据获取的目的地，配合不同的获取方式，有不同的含义和处理。api方式下定义接口地址，rdb方式下定义数据库地址、库名、表名 |  |
| query\_condition | 字符串 | 查询条件 |  |
| filters | 字符串 | 过滤器定义。  对fetch到的原始数据OriginalData进行清洗、过滤（针对不完整、错误、重复数据），需要定义过滤条件。 |  |
| max\_size | 整数 | 单次获取的最大数据记录数。对于获取的数据量较大时，指定单次数据条数上限。 | 待确定 |
| [Options] |  | 不同数据获取方式具体实现类所需的特定配置项 |  |

分析匹配规则描述

|  |  |  |
| --- | --- | --- |
| analyzer\_type |  | 匹配触发类型。  any：只要有数据就触发  frequency：时间窗口内匹配次数超过次数则触发  其它… |
| expression |  | 分析匹配表达式 |
| time\_frame |  | 时间窗口 |
| [Options] |  | 不同分析匹配方式具体实现类所需的特定配置项 |

组装流转规则描述

|  |  |  |  |
| --- | --- | --- | --- |
| transmitter\_type | 字符串 | 报警发送方式，  sms：短信  db：存数据库  stomp：WebSocket推送  lpc：内部过程调用  debug：调试  email：电子邮件  wechat：微信  custom：自定义 |  |
| transmitter\_class |  | 自定义组装流转类名，transmitter\_type为custom时有效 |  |
| wrapper | 字符串 | 报警信息组装格式 |  |
| trigger\_type |  | 触发时机，  immediately：立即传输  timer：定时发送  delay：延时发送 |  |
| realert |  | 是否重复报警 |  |
| subscriptions |  | 报警信息传输对象 |  |
| [Options] |  | 不同组装流转方式具体实现类所需的特定配置项 |  |

6.3 事实数据（Facts）结构

6.4 推理结果数据（AnalysisResult）结构

6.5 引擎运转最终结果（FinalResult）结构

# 7.日志与异常处理设计

# 8.数据库设计

# 9.附录

9.1 ETL技术简介

ETL，是英文 Extract-Transform-Load 的缩写，用来描述将数据从来源端经过萃取（extract）、转置（transform）、加载（load）至目的端的过程。ETL一词较常用在[数据仓库](http://baike.baidu.com/view/19711.htm)，但其对象并不限于[数据仓库](http://baike.baidu.com/view/19711.htm)。

ETL负责将分布的、异构数据源中的数据如关系数据、平面数据文件等抽取到临时中间层后进行清洗、转换、集成，最后加载到数据仓库或数据集市中，成为联机分析处理、数据挖掘的基础。

ETL是数据仓库中的非常重要的一环。它是承前启后的必要的一步。相对于关系数据库，数据仓库技术没有严格的数学理论基础，它更面向实际工程应用。所以从工程应用的角度来考虑，按着物理数据模型的要求加载数据并对数据进行一些系列处理，处理过程与经验直接相关，同时这部分的工作直接关系数据仓库中数据的质量，从而影响到联机分析处理和数据挖掘的结果的质量。

数据仓库是一个独立的数据环境，需要通过抽取过程将数据从联机事务处理环境、外部数据源和脱机的数据存储介质导入到数据仓库中；在技术上，ETL主要涉及到关联、转换、增量、调度和监控等几个方面；数据仓库系统中数据不要求与联机事务处理系统中数据实时同步，所以ETL可以定时进行。但多个ETL的操作时间、顺序和成败对数据仓库中信息的有效性至关重要。

**ETL中的关键技术**

ETL过程中的主要环节就是数据抽取、数据转换和加工、数据装载。为了实现这些功能，各个ETL工具一般会进行一些功能上的扩充，例如工作流、调度引擎、规则引擎、脚本支持、统计信息等。

1.数据抽取

数据抽取是从数据源中抽取数据的过程。实际应用中，数据源较多采用的是关系数据库。从数据库中抽取数据一般有以下几种方式。

(1)全量抽取

全量抽取类似于数据迁移或数据复制，它将数据源中的表或视图的数据原封不动的从数据库中抽取出来，并转换成自己的ETL工具可以识别的格式。全量抽取比较简单。

(2)增量抽取

增量抽取只抽取自上次抽取以来数据库中要抽取的表中新增或修改的数据。在ETL使用过程中。增量抽取较全量抽取应用更广。如何捕获变化的数据是增量抽取的关键。对捕获方法一般有两点要求：准确性，能够将业务系统中的变化数据按一定的频率准确地捕获到；性能，不能对业务系统造成太大的压力，影响现有业务。目前增量数据抽取中常用的捕获变化数据的方法有：

a.触发器：在要抽取的表上建立需要的触发器，一般要建立插入、修改、删除三个触发器，每当源表中的数据发生变化，就被相应的触发器将变化的数据写入一个临时表，抽取线程从临时表中抽取数据，临时表中抽取过的数据被标记或删除。触发器方式的优点是数据抽取的性能较高，缺点是要求业务表建立触发器，对业务系统有一定的影响。

b.时间戳：它是一种基于快照比较的变化数据捕获方式，在源表上增加一个时间戳字段，系统中更新修改表数据的时候，同时修改时间戳字段的值。当进行数据抽取时，通过比较系统时间与时间戳字段的值来决定抽取哪些数据。有的数据库的时间戳支持自动更新，即表的其它字段的数据发生改变时，自动更新时间戳字段的值。有的数据库不支持时间戳的自动更新，这就要求业务系统在更新业务数据时，手工更新时间戳字段。同触发器方式一样，时间戳方式的性能也比较好，数据抽取相对清楚简单，但对业务系统也有很大的倾入性（加入额外的时间戳字段），特别是对不支持时间戳的自动更新的数据库，还要求业务系统进行额外的更新时间戳操作。另外，无法捕获对时间戳以前数据的delete和update操作,在数据准确性上受到了一定的限制。

c.全表比对：典型的全表比对的方式是采用MD5校验码。ETL工具事先为要抽取的表建立一个结构类似的MD5临时表，该临时表记录源表主键以及根据所有字段的数据计算出来的MD5校验码。每次进行数据抽取时，对源表和MD5临时表进行MD5校验码的比对，从而决定源表中的数据是新增、修改还是删除，同时更新MD5校验码。MD5方式的优点是对源系统的倾入性较小（仅需要建立一个MD5临时表），但缺点也是显而易见的，与触发器和时间戳方式中的主动通知不同，MD5方式是被动的进行全表数据的比对，性能较差。当表中没有主键或唯一列且含有重复记录时，MD5方式的准确性较差。

d.日志对比：通过分析数据库自身的日志来判断变化的数据。Oracle的改变数据捕获（CDC，Changed Data Capture）技术是这方面的代表。CDC 特性是在Oracle9i数据库中引入的。CDC能够帮助你识别从上次抽取之后发生变化的数据。利用CDC，在对源表进行insert、update或 delete等操作的同时就可以提取数据，并且变化的数据被保存在数据库的变化表中。这样就可以捕获发生变化的数据，然后利用数据库视图以一种可控的方式提供给目标系统。CDC体系结构基于发布者/订阅者模型。发布者捕捉变化数据并提供给订阅者。订阅者使用从发布者那里获得的变化数据。通常，CDC系统拥有一个发布者和多个订阅者。发布者首先需要识别捕获变化数据所需的源表。然后，它捕捉变化的数据并将其保存在特别创建的变化表中。它还使订阅者能够控制对变化数据的访问。订阅者需要清楚自己感兴趣的是哪些变化数据。一个订阅者可能不会对发布者发布的所有数据都感兴趣。订阅者需要创建一个订阅者视图来访问经发布者授权可以访问的变化数据。CDC分为同步模式和异步模式，同步模式实时的捕获变化数据并存储到变化表中，发布者与订阅都位于同一数据库中。异步模式则是基于Oracle的流复制技术。

ETL处理的数据源除了关系数据库外，还可能是文件，例如txt文件、excel文件、xml文件等。对文件数据的抽取一般是进行全量抽取，一次抽取前可保存文件的时间戳或计算文件的MD5校验码，下次抽取时进行比对，如果相同则可忽略本次抽取。

2.数据转换和加工

从数据源中抽取的数据不一定完全满足目的库的要求，例如数据格式的不一致、数据输入错误、数据不完整等等，因此有必要对抽取出的数据进行数据转换和加工。

数据的转换和加工可以在ETL引擎中进行，也可以在数据抽取过程中利用关系数据库的特性同时进行。

(1)ETL引擎中的数据转换和加工

ETL引擎中一般以组件化的方式实现数据转换。常用的数据转换组件有字段映射、数据过滤、数据清洗、数据替换、数据计算、数据验证、数据加解密、数据合并、数据拆分等。这些组件如同一条流水线上的一道道工序，它们是可插拔的，且可以任意组装，各组件之间通过数据总线共享数据。

有些ETL工具还提供了脚本支持，使得用户可以以一种编程的方式定制数据的转换和加工行为。

(2)在数据库中进行数据加工

关系数据库本身已经提供了强大的SQL、函数来支持数据的加工，如在SQL查询语句中添加where条件进行过滤，查询中重命名字段名与目的表进行映射，substr函数，case条件判断等等。下面是一个SQL查询的例子。

select ID as USERID, substr(TITLE, 1, 20) as TITLE, case when REMARK is null then ' ' else REMARK end as CONTENT from TB\_REMARK where ID > 100;

相比在ETL引擎中进行数据转换和加工，直接在SQL语句中进行转换和加工更加简单清晰，性能更高。对于SQL语句无法处理的可以交由ETL引擎处理。

3.数据装载

将转换和加工后的数据装载到目的库中通常是ETL过程的最后步骤。装载数据的最佳方法取决于所执行操作的类型以及需要装入多少数据。当目的库是关系数据库时，一般来说有两种装载方式：

(1)直接SQL语句进行insert、update、delete操作。

(2)采用批量装载方法，如bcp、bulk、关系数据库特有的批量装载工具或api。

大多数情况下会使用第一种方法，因为它们进行了日志记录并且是可恢复的。但是，批量装载操作易于使用，并且在装入大量数据时效率较高。使用哪种数据装载方法取决于业务系统的需要。