数据流动态分析平台

软件架构文档

版本 <0.3>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2022 年 11 月 6 日 | 0.1 | 完成简介、质量属性设计 | 胡昊源 |
| 2022 年 11 月 7 日 | 0.2 | 完成 2、3、4、5 节 | 佘智龙、李梦瑶 |
| 2022 年 11 月 8 日 | 1.0 | 完成 6、7 节，整合文档 | Naomi、胡昊源 |
| 2022年11月17日 | 1.1 | 修改第6、7节 | Naomi |
| 2022年11月17日 | 1.2 | 完成低代码平台综述 | Naomi |
| 2022年11月17日 | 1.3 | 修改进程视图和部署视图 | 佘智龙 |
| 2022年11月18日 | 1.4 | 完成依据架构评审的修改 | 李梦瑶 |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

2.1 管理数据源数据功能 4

2.2 管理数据分析服务功能 4

3. 逻辑视图 5

3.1 概述 5

3.2 在构架方面具有重要意义的设计包 6

数据源处理 6

数据流处理 6

4. 进程视图 7

5. 部署视图 8

6. 实现视图 9

7. 技术视图 10

8. 质量属性的设计 10

8.1 系统性能 10

8.2 系统可扩展性 10

8.3 系统可靠性 11

8.4 系统易用性 11

8.5 系统可移植性 11

软件架构文档

# 简介

## 目的

本文档将从架构方面对数据流动态分析平台进行综合概述，其中会使用多种不同的架构视图来描述平台的各个方面。它用于记录并表述已对系统的架构方面作出的重要决策。本文档主要面向开发人员、产品经理、运维人员以及未来的潜在用户。本节为软件架构简介，文档将会对平台的各个架构做出介绍，读者可结合实际问题需求，从文档中找到对应视图进行深入阅读。

## 项目基础

本项目依托于IST实验室正在研发中的低代码平台进行开发，该平台是一款支持用户低代码开发数据模型、业务服务、服务编排、用户界面的平台，旨在为用户提供快速定义并上线产品的功能，用户研发人员只需进行低代码开发，就可以完整地构建业务场景所需的数据模型、业务服务、服务流程以及用户界面。该平台为BtoC模式，以提升用户和用户的使用体验为核心，针对前述问题和需求提出相应解决方案，并且针对整个系统的不同受众，提供特点服务。下面来简要描述一下该平台在技术实现和功能方面具的特点：

### **可动态加载的业务数据模型**

采用数据元模型动态加载，支持开发者根据业务需要灵活构建数据模型。开发者可以根据需要定义数据模型的属性名、属性类型、属性能否为空等信息，也可以定义数据模型的关联属性，在开发者上传数据模型之后，系统会在对应的数据库中自动生成数据库表。

平台提供可视化数据模型自定义界面，开发者可以简单直观地定义数据模型。

### **多维度可视化的业务数据配置能力**

提供业务数据的可视化操作界面，支持业务数据的按需配置与多维度查询。在研发人员发布数据模型之后，业务人员可以按照业务需求对数据按需进行配置，如定义一款新产品的产品名、存款周期、年化收益率等基本属性，又如更新一款存款产品的剩余额度。

此外，平台还提供面向数据的多维度查询功能。业务人员可以通过直接查询某项数据的属性获得目标数据，也可以通过对数据的关联数据进行查询，获得目标数据。平台的这种多维度查询设计，能够支持各种复杂的业务场景，适配可能出现的多种业务场景。

### **面向核心业务自定义的功能模型服务化能力**

当开发一款产品时，往往需要从新定义服务框架，需要完成大量和业务逻辑无关的代码开发。低代码平台为了能够减少开发者的开发压力，节约开发时间，设计了一款轻量级业务服务自研框架。开发者只需要定义业务服务的输入和输出，实现平台所暴露的服务接口，完成业务的核心逻辑，将代码以jar包的形式上传，平台就会自动解析业务服务，帮助开发者通过低代码开发的形式灵活拓展业务。

### **随调随启的业务服务执行机制**

平台采用事件驱动的设计理念，当开发者上传的业务服务被平台解析后，不会立刻部署并运行。当服务被调用时，会产生服务实例，服务实例被部署并执行，运行开发者设定的服务逻辑，根据服务输入产生服务输出，而当服务调用完成，服务实例即会结束运行。采用这种随调随启的执行机制，可以大大提高资源的利用效率。

此外，平台还提供业务服务本地化与容器化两种执行环境。本地化执行环境启用更加快速，适合轻量级服务，而容器化环境会将服务打包镜像，在服务器上部署运行，适合较重量级服务，可以支持不同需求的业务场景。

### **支持复杂控制逻辑的服务编排与执行引擎**

平台采用基于可视化增强的服务编排DSL，帮助开发者可视化编排服务流程。平台的可视化服务编排DSL有节点和线条两种组件。节点分为服务节点和控制节点，控制节点又可以分为并行节点、循环节点以及分支选择节点，使平台的服务编排DSL可以支持多种复杂逻辑，适配不同需求的业务场景。线条则用来连接节点，表示节点在服务流程中的先后关系。开发者可以拖拽组件，为节点绑定服务，编排业务流程。

### **所见即所得的界面开发方法**

界面组件动态渲染，支持开发者通过拖拽组件的方式绘制用户界面。平台支持多种可视化组件，如按钮、输入框、文字等。开发者可以拖拽组件、绑定服务，快速绘制用户界面，直观便捷。

用户界面的组件能够集成用户动态输入、业务服务以及数据服务等多种数据源，打通业务与数据链路。并且开发者能够根据业务需求定义用户界面的跳转逻辑，使用户界面的跳转逻辑准确贴合已定义的业务流程。

## 参考资料

[1] 蔡鸿明.沈备军，任锐，互联网时代的软件工程[M].上海交通大学出版社,2021.11.

[2] 沈备军，陈昊鹏，陈雨亭，软件工程原理[M].高等教育出版社,2013.2.

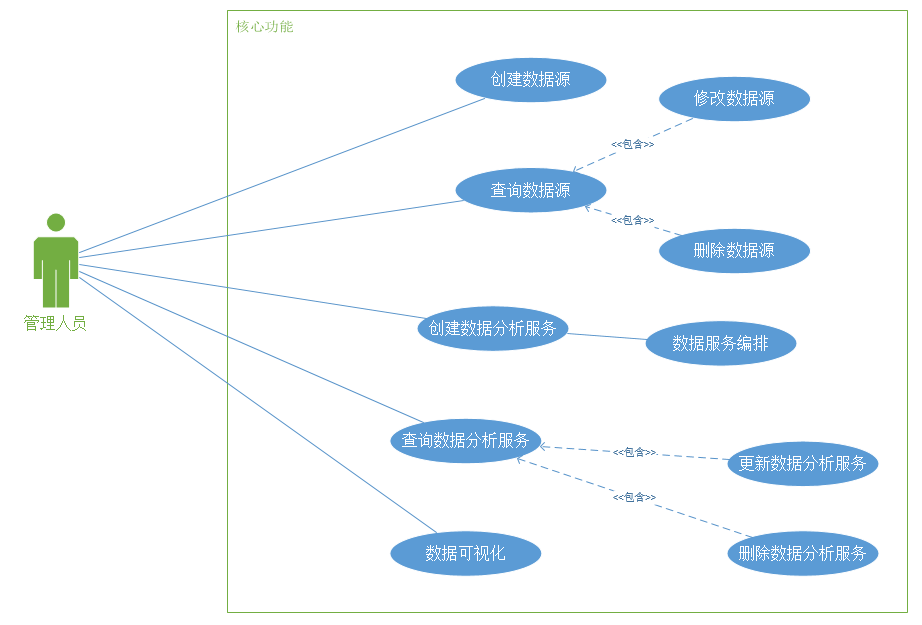
[3] 胡昊源，Naomi，佘智龙，李梦瑶，数据流动态分析平台软件需求规约.上海交通大学软件学院课程文档,2022.11.

# 用例视图

管理人员通过管理数据源功能对数据流输入进行管理，包括创建数据源和查询数据源。根据提供的数据源接口，配置输入数据流的数据类型、名字、id等信息，创建数据源。同时，可以查询所创建的数据源，并对数据源进行修改和删除。

管理人员通过管理数据分析功能对数据流处理分析服务进行管理，包括创建数据分析服务和查询数据分析服务。根据提供的数据分析服务接口，配置数据处理的相关参数，创建对应的数据分析服务。同时，可以查询所创建的数据分析服务，并对数据分析服务进行修改和删除。

管理人员通过数据可视化功能对数据分析的结果进行可视化展示，可查看数据分析结果的各类视图。



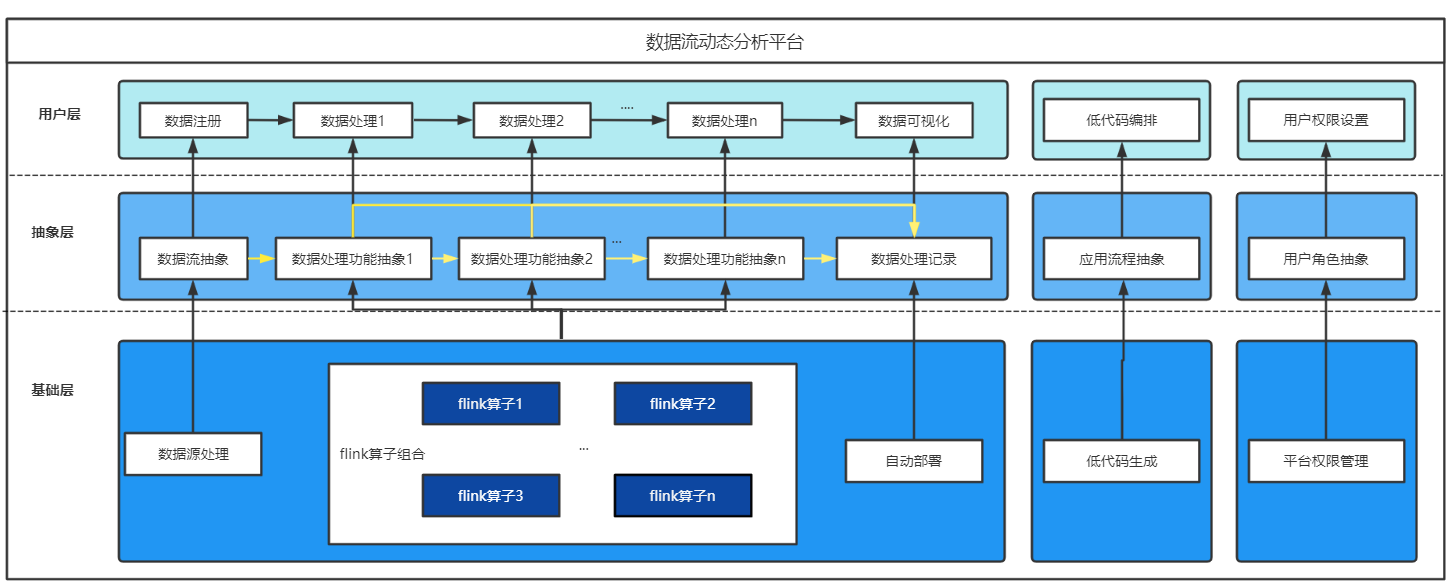
**图1 核心用例视图**

# 逻辑视图

## 概述

如图所示，本数据流动态分析平台采用分层结构和管道过滤器风格结合的方式，共分为基础层、抽象层和用户层三层。

基础层提供底层的数据源接入处理、基于 Flink 的算子 API 组合、自动部署、低代码生成和平台权限管理功能。抽象层分别对基础层中的底层模块进行抽象，形成数据流抽象、多种类型数据处理功能、数据处理记录、应用流程抽象、用户角色抽象这些接口，向上功能给用户层。用户层利用这些接口形成数据流注册、数据处理编排、数据可视化这一管道流程，同时还提供了低代码编排和用户权限设置两大功能。



**图2 数据处理平台逻辑视图**

## 在构架方面具有重要意义的设计包

## 数据源处理

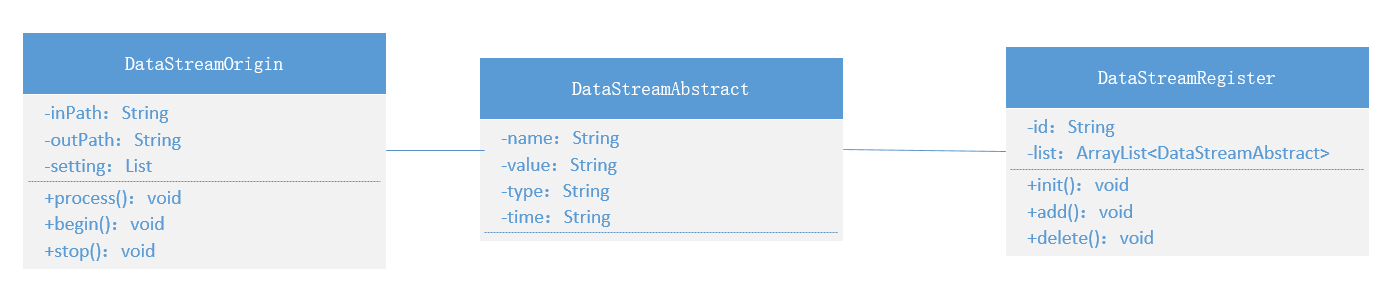
数据源主要对数据流进行抽象，包括接受数据流，对数据流的格式进行抽象，用户注册数据流。

重要的类：

DataStreamOrigin：接受数据流，由于项目没有实时数据流环境，需要模拟数据流，所以该类主要完成将日志类数据转换为实时数据流的功能；

DataStreamAbstract：定义数据流的格式，包括数据属性、id等，作为用户定义所需的数据流输入的依据；

DataStreamRegister：用户根据界面表单，填写数据流中数据的名称，需要处理的数据等信息。



**图3 数据源处理设计包**

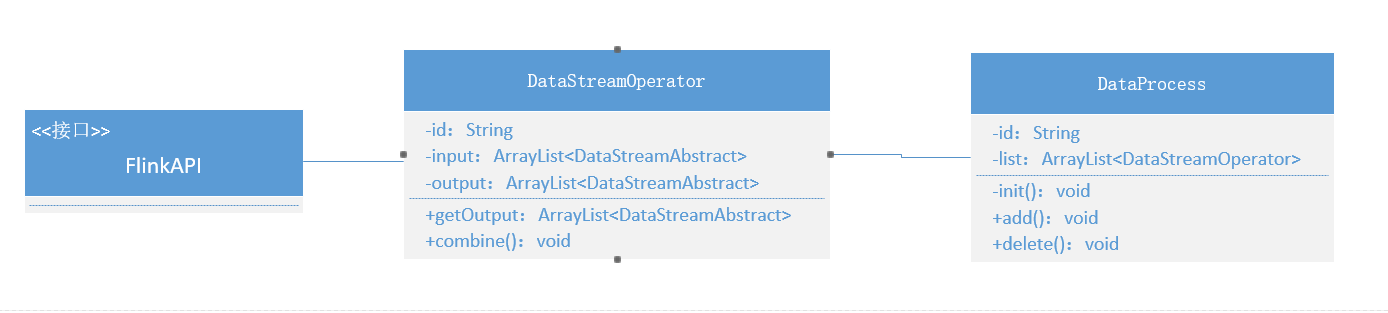
## 数据流处理

数据流处理主要对数据流进行map、reduce、统计分析等各类处理操作。

重要的类：

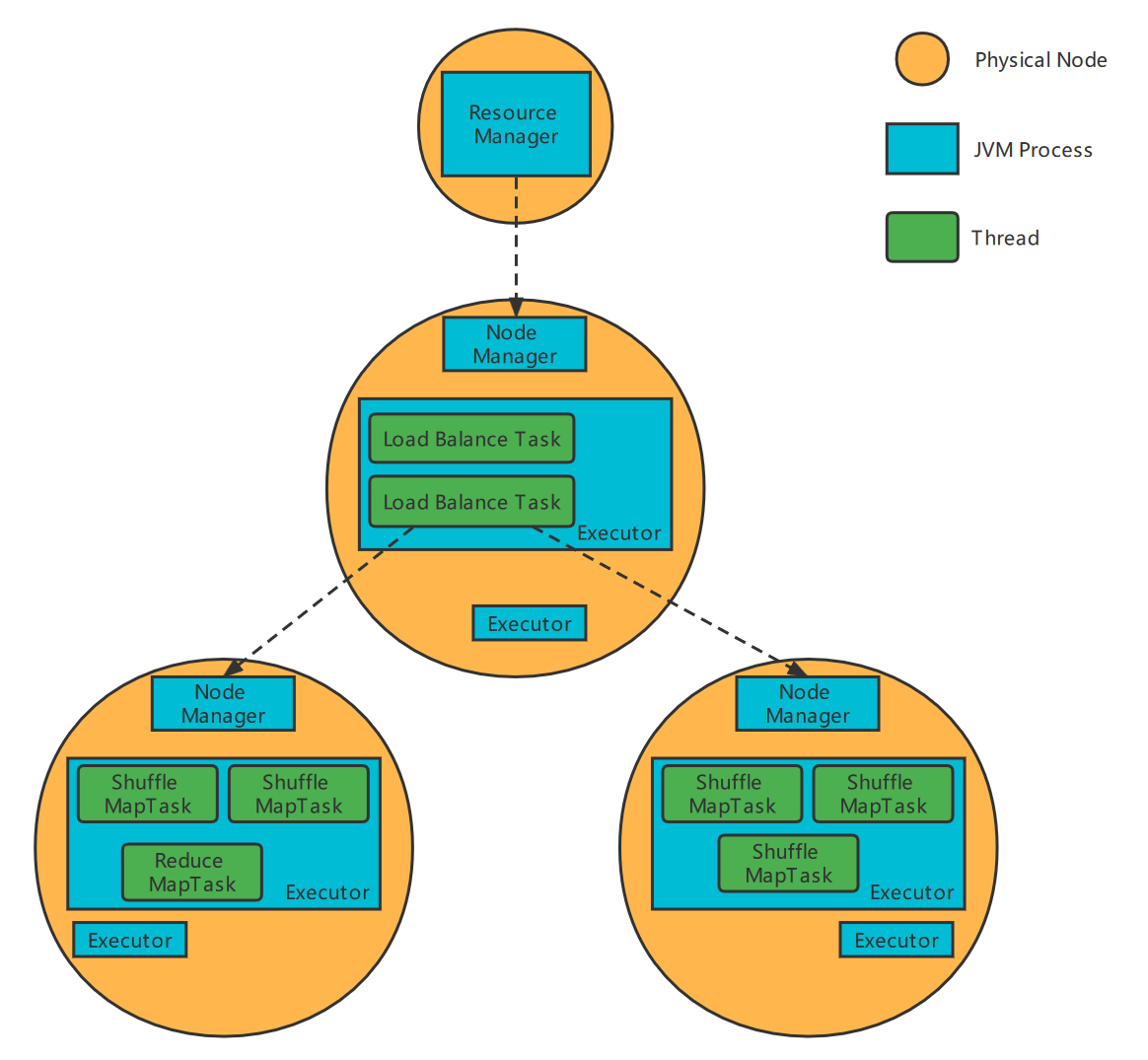
FlinkAPI： flink框架提供的各种数据流处理API；

DataStreamOperator：基于flink DataStream API，抽象出数据流处理接口，封装底层具体实现，便于用户按需求调用编排所需的数据处理流程；

DataProcess：用户按照数据处理的需求，选择相应的数据处理功能接口，填写相应的数据处理信息。

**图4 数据流处理设计包**

# 进程视图



**图5 数据流动态处理平台进程视图**

1） 每个节点上可以运行一个或多个Executor服务；

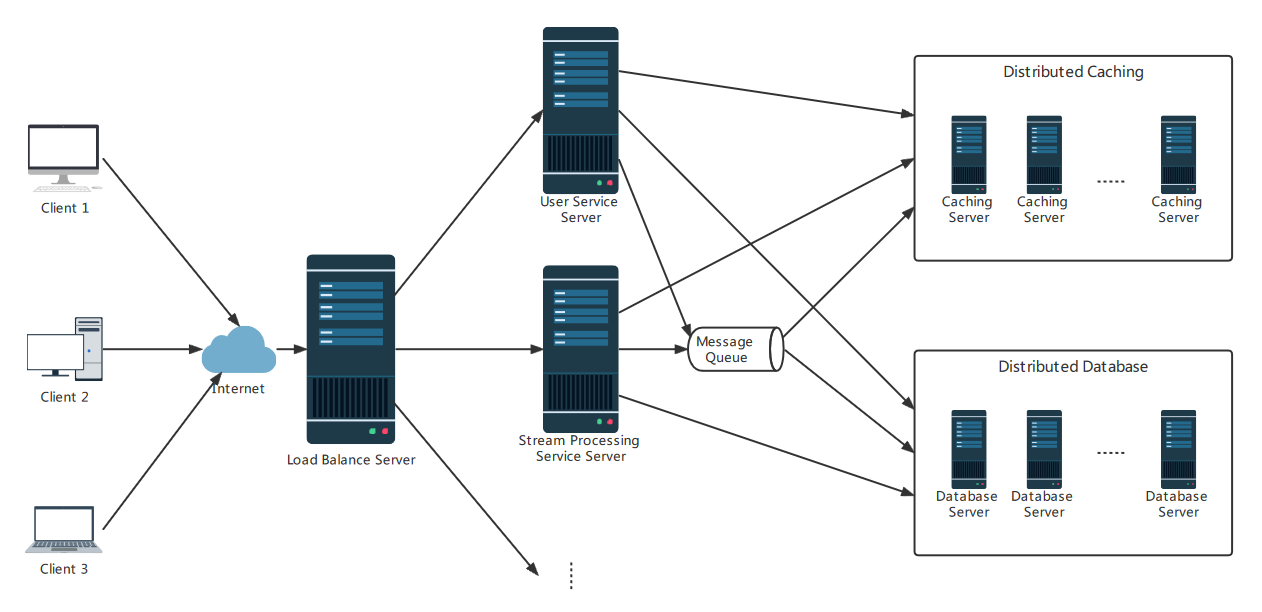
2） 每个Executor配有一定数量的slot，表示该Executor中可以同时运行多少个ShuffleMapTask或者ReduceTask；

3） 每个Executor单独运行在一个JVM进程中，每个Task则是运行在Executor中的一个线程；

4） 同一个Executor内部的Task可共享内存，比如通过函数SparkContext#broadcast广播的文件或者数据结构只会在每个Executor中加载一次，而不会像MapReduce那样，每个Task加载一次；

5） Executor一旦启动后，将一直运行，且它的资源可以一直被Task复用，直到Spark程序运行完成后才释放退出。

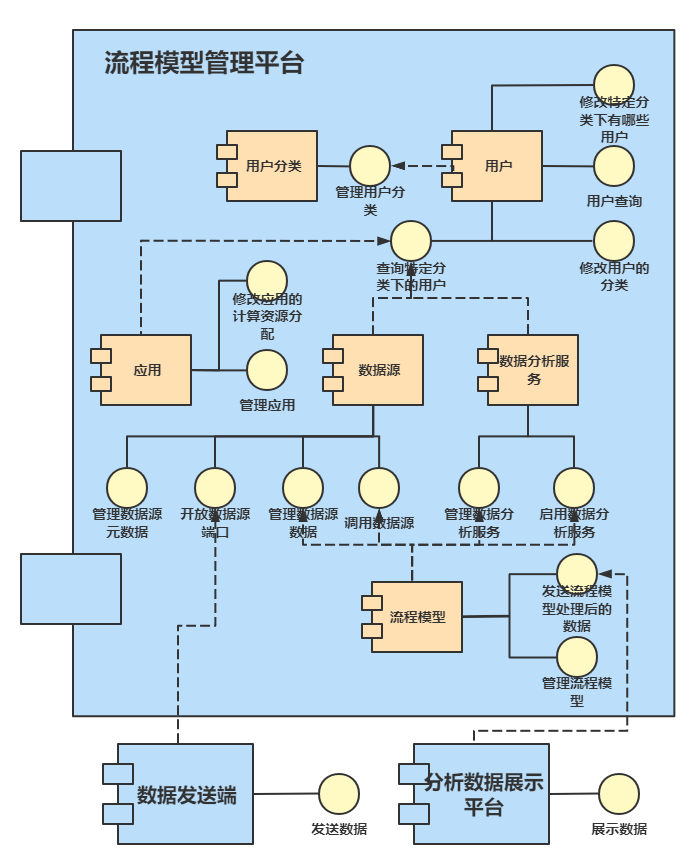
# 部署视图



**图6 数据流动态处理平台部署视图**

数据流动态分析平台的部署视图如上所示，是 C/S 架构。对于 Client 和 Server、Server和Server之间来说，他们之间均通过 WLAN 网络进行连接。其中，进程视图中的Resource Manager对应部署视图的 Load Balance Server，其余节点则对应Application Server。

# 实现视图

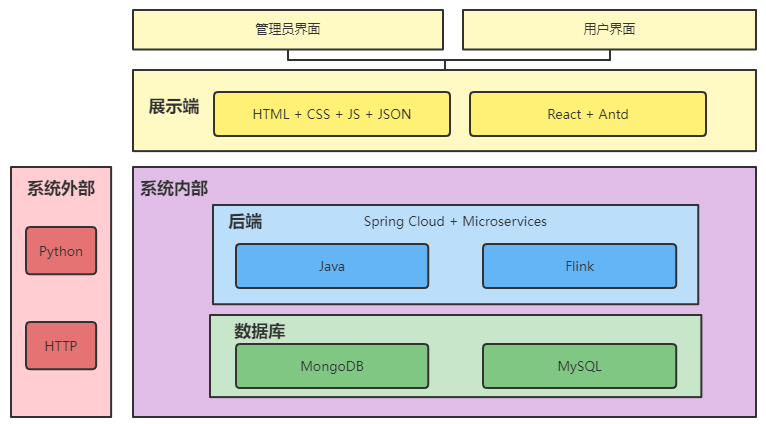


**图7 数据流动态处理平台实现视图**

我们选择使用构件图来作为实现视图。其中将平台分为三个主要的构件：一个是负责主要工作的流程模型管理平台、还有一个是负责为平台发送数据的数据发送端、以及展示最终处理完成的数据的分析数据展示平台。

流程模型管理平台中，又有用户分类、用户、应用、数据源、数据分析服务和流程模型6个分构件。用户分类提供给管理用户修改用户的分类数据的接口，而用户则主要提供给其他构件查询用户分类的接口。查询特定分类下的用户主要被运用于其他构件的用户权限检查。数据源需要为数据发送端提供端口来让其发送数据，同时也需要具备在流程模型中调用数据源的功能。数据分析服务也需要能够被流程模型调用。流程模型构件在创建好流程模型后，需要将处理好的数据发送到分析数据展示平台。

# 技术视图



**图8 数据流动态处理平台技术视图**

1. 应用前端：HTML、CSS、JS、JSON、React、Antd

2）应用后端：Java、SpringBoot

3）计算与管理： Flink

5）数据存储：MySql、Mongodb

# 质量属性的设计

## 系统性能

由于系统采用微服务架构，微服务之间的数据互通，在一定程度上产生了性能损耗。为此，系统采用了这样的设计，来优化性能：

**负载均衡**

系统通过一个负载均衡进程，实现服务器之间合理的任务分配，避免因负载造成的性能损失，在一定程度上提高了系统性能。

**消息队列**

对于需要批量处理的数据，系统设计了消息队列进行缓存。同时对于那些可被立即处理的非数据流分析请求，系统提供直接访问存储服务器接口，提高系统性能。

## 系统可扩展性

由于系统采用微服务架构，易于进行扩展于调整。且对于微服务来说，增加服务吞吐量，往往只需要增加节点数量，因此系统具有较高的可扩展性。

## 系统可靠性

对于微服务架构来说，同一个服务同时会有多个节点运行。在一个节点发生崩溃的情况下，系统业务仍然能够正常运转，因此系统就有较高的可靠性。

## 系统易用性

数据流动态处理平台在设计阶段考虑了低代码技术，具有低代码编排流程功能，旨在为用户提供良好的界面和易用的体验，因此系统具有较高的易用性。

## 系统可移植性

鉴于平台技术栈，系统前端采用 React 、JavaScript、Html进行开发，后端采用 SpringBoot 进行开发，在设计时主要考虑网页端，并没有考虑到移动端等其他平台，因此系统可移植性较差。