欺诈检测系统设计文档

作者：黄寅飞
版本：1.0
发布日期：2025-5-11

1. 简介

随着电子商务与金融交易的普及，欺诈行为呈现高发态势。本系统旨在通过实时数据分析与机器学习技术，构建低延迟、高精度的欺诈检测解决方案，覆盖交易风险识别、用户行为分析、预警通知等功能。

本系统有如下设计目标：

- 功能性目标：实现交易数据的实时采集、处理、规则判定与报警，欺诈检测响应时间≤100ms。
- 非功能性目标：
  - 高可用性：系统可用性≥99.99%，支持 Kubernetes 集群部署与自动扩缩容。

2. 系统架构设计

系统采用微服务架构，分为以下核心模块：

1. 数据采集：通过阿里云消息队列 MNS 处理实时交易数据流。

2. 规则引擎：基于 Spring Boot，对实时消息进行处理，基于规则引擎进行欺诈行为判定，并生成实时报警。

3. 数据存储：使用 MySQL 数据库进行持久化存储。

4. 预警通知：基于 Vue.js 3，通过前端界面向用户作预警通知，由用户实施进一步冻结账户或人工审核的流程。

5. 容器云：使用 Azure AKS 容器云及 Azure ACR 镜像仓库，实现云原生部署、故障自愈、弹性伸缩能力。

6. 日志服务：使用阿里云 SLS 日志服务，对 Log、Metric、Trace 进行一站式采集加工分析。

7. 测试框架：基于 Junit，对后端模块的数据、业务逻辑、通信进行测试验证。

8. 测试工具：用于模拟金融交易实时数据流的辅助工具。

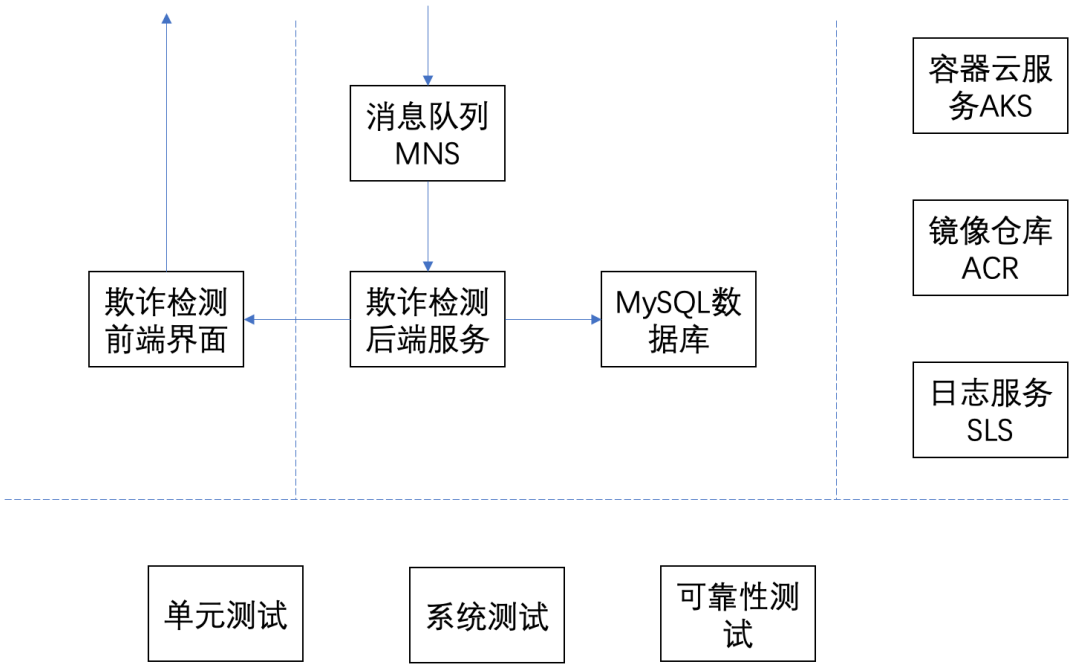

图 1 系统架构设计图

技术选型：

模块	技术栈
消息队列	阿里云轻量消息队列 MNS
日志服务	阿里云日志服务 SLS
容器云	Azure Kubernetes Service
镜像仓库	Azure Container Registries
数据库	MySQL
后端框架	Spring Boot
前端框架	Vue.js 3

3. 模块详细设计

3.1 欺诈检测规则引擎

设置如下检测规则：

- 规则一：金额异常：(当前交易金额 - 用户历史月均交易额) / 用户历史交易标准差 > 3
- 规则二：时空矛盾：当前交易 GPS 位置与上一次交易地点的直线距离 / 两次交易时间差 > 300

3.2 消息队列通知服务

当消息队列收到入站金融交易消息时，调用此接口通知规则引擎进行欺诈检测。

服务接口：/api/v1/notify

3.3 违规预警通知服务

前端预警通知页面定期轮询调用此接口，获得最新的实时预警通知信息。

服务接口：/api/v1/alert

3.4 欺诈检测前端服务

预警通知按照时间先后顺序进行可视化展示，如下图。

编号	账号	业务	金额	触发规则	报警内容
8	A123456789	W	100	2	Alert: location rule broken!
7	A123456789	W	1000	1	Alert: amount rule broken!
6	A123456789	W	100	2	Alert: location rule broken!
5	A123456789	W	100	2	Alert: location rule broken!
4	A123456789	W	100	2	Alert: location rule broken!
3	A123456789	W	100	2	Alert: location rule broken!

图 2 欺诈检测预警界面

4. 数据设计

4.1 金融交易记录

表 1：交易记录表结构

列名	#	数据类型	非空	自增	键
id	1	int	[v]	[v]	PRI
account	2	char(10)	[ ]	[ ]	
side	3	char(1)	[ ]	[ ]	
amount	4	decimal(16,2)	[ ]	[ ]	
avg_amount	5	decimal(16,2)	[ ]	[ ]	
std	6	decimal(16,2)	[ ]	[ ]	
gps_x	7	float	[ ]	[ ]	
gps_y	8	float	[ ]	[ ]	
last_gps_x	9	float	[ ]	[ ]	
last_gps_y	10	float	[ ]	[ ]	
last_time	11	datetime	[ ]	[ ]	
trans_time	12	datetime	[ ]	[ ]	

4.2 预警通知

表 2：预警通知表结构

列名	#	数据类型	非空	自增	键
id	1	int	[v]	[v]	PRI
account	2	char(10)	[ ]	[ ]	
side	3	char(1)	[ ]	[ ]	
amount	4	decimal(16,2)	[ ]	[ ]	
rule	5	int	[ ]	[ ]	
content	6	varchar(255)	[ ]	[ ]	

5. 测试计划

5.1 单元测试

编号	测试内容	测试结果
1	交易数据验证	通过
2	消息队列验证	通过
3	数据加工验证	通过
4	规则一未触发	通过
5	规则一触发	通过
6	规则二未触发	通过
7	规则二触发	通过
8	规则引擎验证	通过

5.2 功能测试

测试工具：tool/send.py

依次输入命令：

python send.py 0

python send.py 1

python send.py 2

python send.py 3

其中消息 2 会触发规则一金额异常，消息 3 会触发规则二时空矛盾。

使用前端界面可收到对应的报警通知。

5.3 可靠性测试

部署到 AKS 容器云后，查看容器实例健康状态，命令如下：

kubectl get pods

```
NAME                                READY   STATUS    RESTARTS   AGE
fd-deployment-5664484d9c-d8lxv   1/1     Running   0          44m
fd-deployment-5664484d9c-wzmj7   1/1     Running   0          44m
```

将其中一个实例杀掉，命令如下：

kubectl delete pod fd-deployment-5664484d9c-d8lxv

再次查看容器实例健康状态，可看到瞬间又启动一个新的 POD。

```
NAME                                READY   STATUS    RESTARTS   AGE
fd-deployment-5664484d9c-dkrhs   1/1     Running   0          3s
fd-deployment-5664484d9c-wzmj7   1/1     Running   0          45m
```

6. 部署与维护

6.1 后端代码编译执行

git clone https://github.com/hifi2046/fault-detect.git

maven clean

maven compile

maven package

java -jar target/fault_detect-0.0.1-SNAPSHOT.jar

6.2 前端代码编译执行

git clone https://github.com/hifi2046/fault-detect-fe.git

npm install

npm run dev

6.3 发布容器云

docker build -t fault-detect:1.0 .

docker tag fault-detect:1.0 hifiwork.azurecr.io/fault-detect:1.0

docker push hifiwork.azurecr.io/fault-detect:1.0

```
kubectl apply -f deployment.yml

kubectl apply -f service.yml
```