支付中心设计文档

作者:夏孟兵

2017-04-13

目录

一、 简介.....3
1. 编写目的.....3
2. 文档范围.....3
3. 定义.....3
二、 设计思想.....3
三、 体系和描述.....3
1. 关键质量需求.....4
2. 约束条件.....4
3. 整体说明.....5
四、 功能划分.....7
关键系统功能划分.....7
五、 开发策略.....7
1. 概述.....7
2. 系统层次划分.....8
3. 主要系统结构和代码结构.....8
4. 接口设计.....9
六、 运行视图.....9
1. 概述.....9
2. 关键运行视图.....9
七、 数据视图.....13
1. 概述.....13
八、 部署方式.....14
1. 概述.....14
2. 部署图.....15
九、 其他说明.....15

一、简介

1. 编写目的

本文档全面与系统地表述了支付中心的构架,并从不同角度描述本系统的各个主要方面,以满足系统的相关涉众对本系统的不同关注焦点和需求。

本文档的预期阅读人员为项目经理、开发人员、测试人员和其他有关的工作人员。

2. 文档范围

本文档适合于支付中心的总体架构。

1. 设计思想
2. 体系描述
3. 系统和模块化分
4. 系统模块功能描述
5. 主要模块接口设计

3. 定义

二、设计思想

为了降低系统耦合度，增加系统内聚性，在需求发生更改时能在较短的时间内对系统做出修改，并重新投入使用，把支付中心做为一个独立的系统,同时系统采用分层体系风格，严格按照一定的规则来进行接口设计，并以之为根据进行详细设计。系统内分为数据层、业务逻辑层、表示层。本设计的业务需求来源为产品经理,主要依据产品经理提出的支付中心展开技术方面的设计。

三、体系和描述

整个系统顶层架构采用分层的风格，整个系统的体系结构非常清晰，易于详细设计、编码、维护以及适应需求变更。通过划分系统,分层，定义出层与层之间接口，使得在更加规范的同时拥有更为丰富的接口描述，使得业务直接解耦,同时使层与层之间的耦合度降低,增强了支付中心的系统化和模块化和可扩展性以及可维护性。同时，分层也有益于任务的分配，通过明确清晰的接口，降低集成的难度，提高效率。

1. 关键质量需求

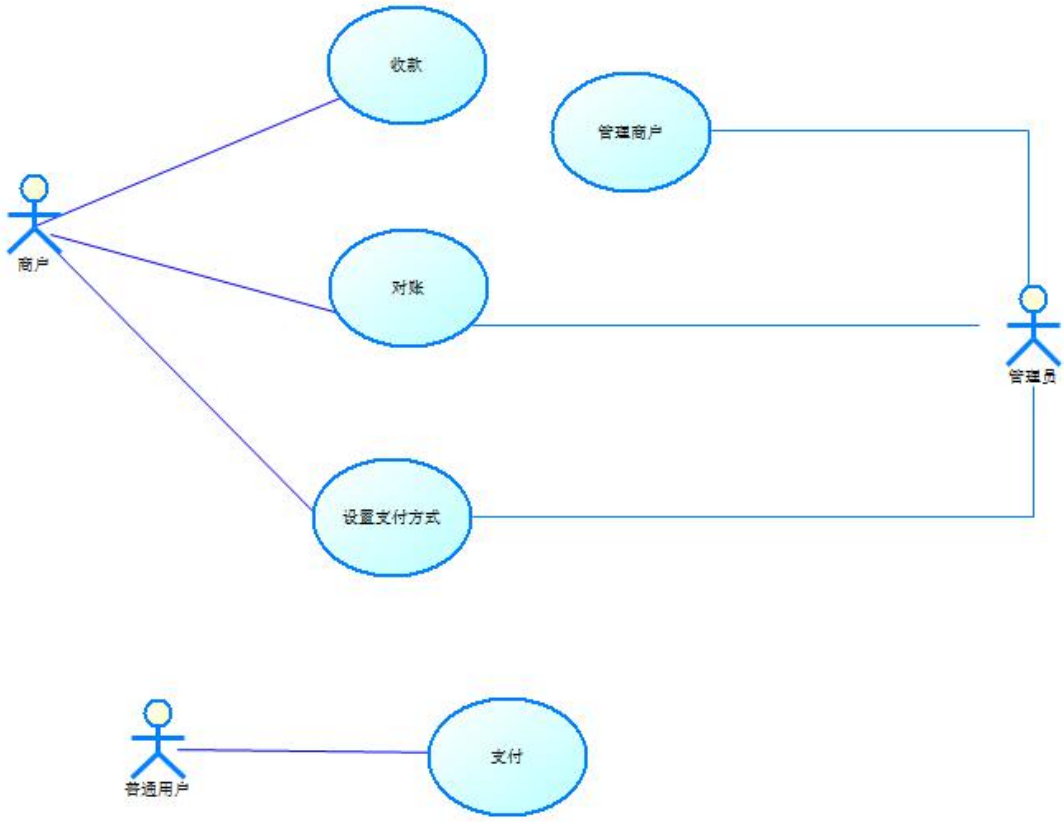
由于支付中心针对互联网,对外开放,使用频度高,系统性能要求高,要求有较为可靠的安全性能。

- 查询速度：不超过 3 秒(后台统计类除外)；
- 其它交互功能反应速度：不超过 3 秒(后台和统计类除外)；
- 系统必须能够保证每天 24 小时不间断运行，可用率为 99%
- 合理的设计系统的结构以保证较高的可维护性，系统的模块应该可替换
- 系统应当正确处理发生的异常或者错误，并返回错误信息

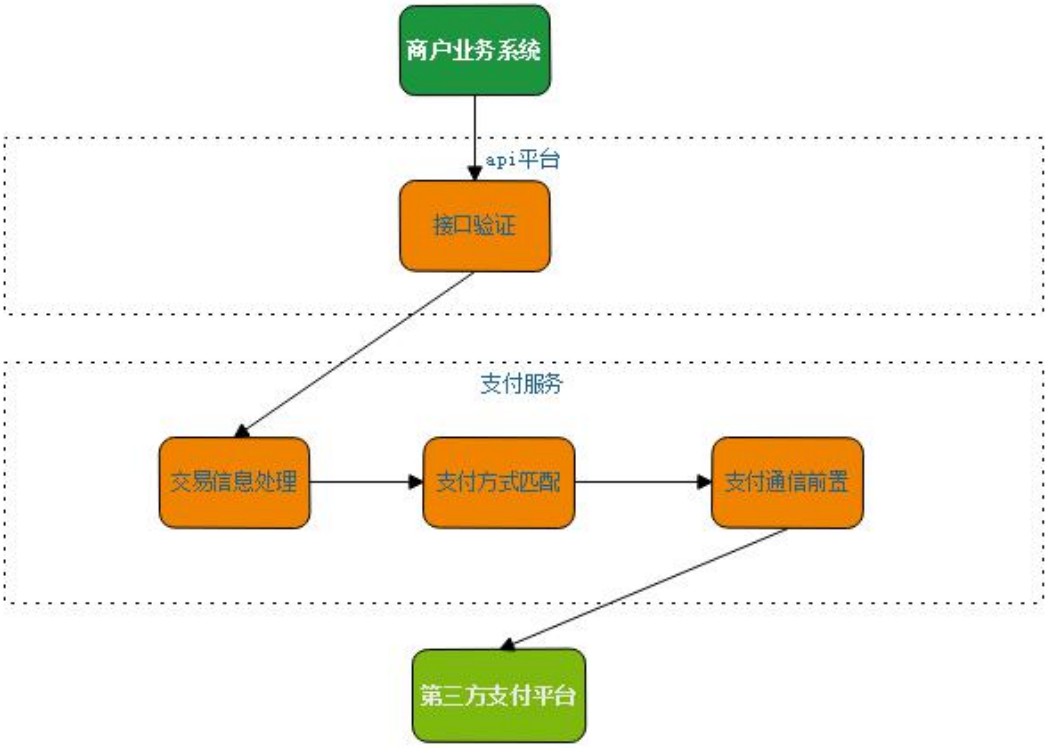
2. 约束条件

- 运行在 LINUX 系统中
- 采用开源框架和软件
- 采用 java 开发
- 采用普通 pc 服务器
- 采用 BS 架构

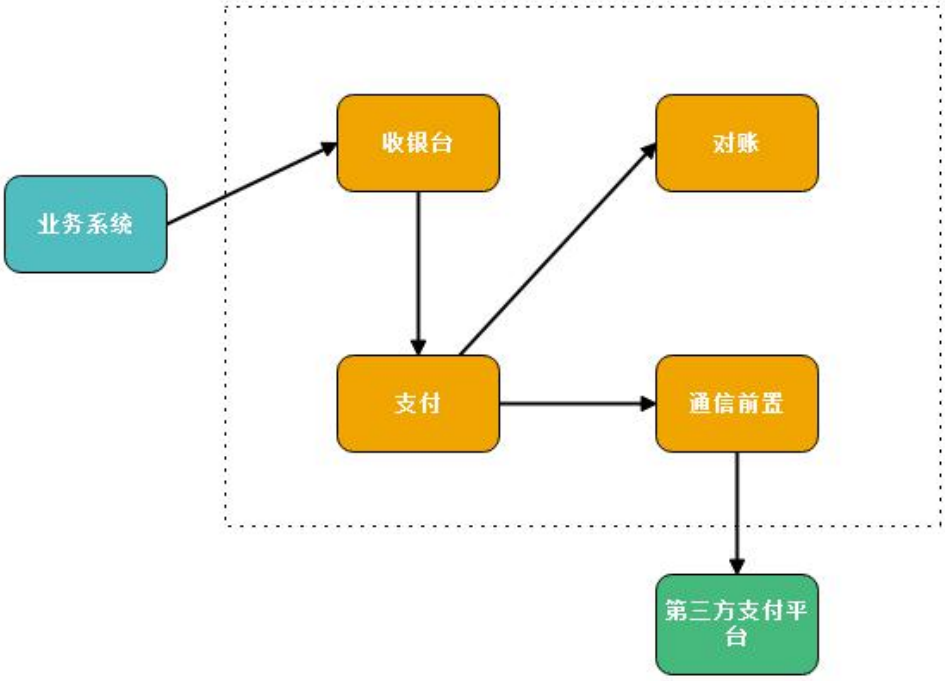
3. 整体说明



接口处理:


商户业务系统
api平台
接口验证
支付服务
交易信息处理
支付方式匹配
支付通信前置
第三方支付平台

资金处理:


业务系统
收银台
对账
支付
通信前置
第三方支付平台

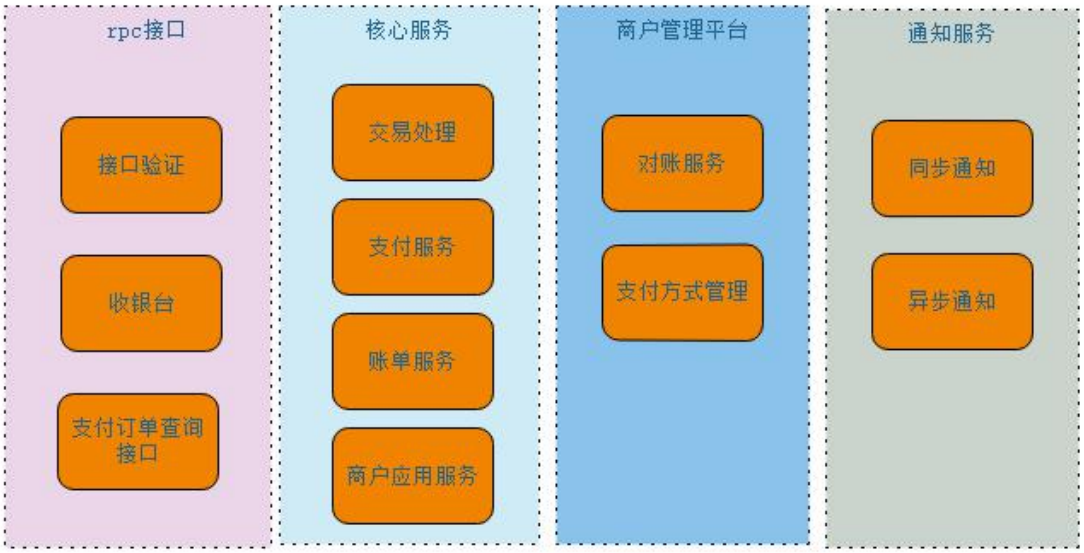
四、功能划分

关键系统功能划分

为了使支付中心功能更加细化、模块化,系统化,提高系统的可扩展和稳定性。本系统分为三大子系统,各个子系统内采用分层模式开发,再通过集成各个系统组成完整系统.

具体系统划分为对外接口系统,管理平台,定时任务为主的三大系统进行开发,而在此三大系统基础上,又细分成了商户签约、商户支付设置,对账,支付处理处理等模块。如商户基础数据采用现有商家中心数据,注册和登录,修改密码手机邮件等基础信息操作全部通过商家中心完成.

系统和功能模块示意图如下所示：



五、开发策略

1. 概述

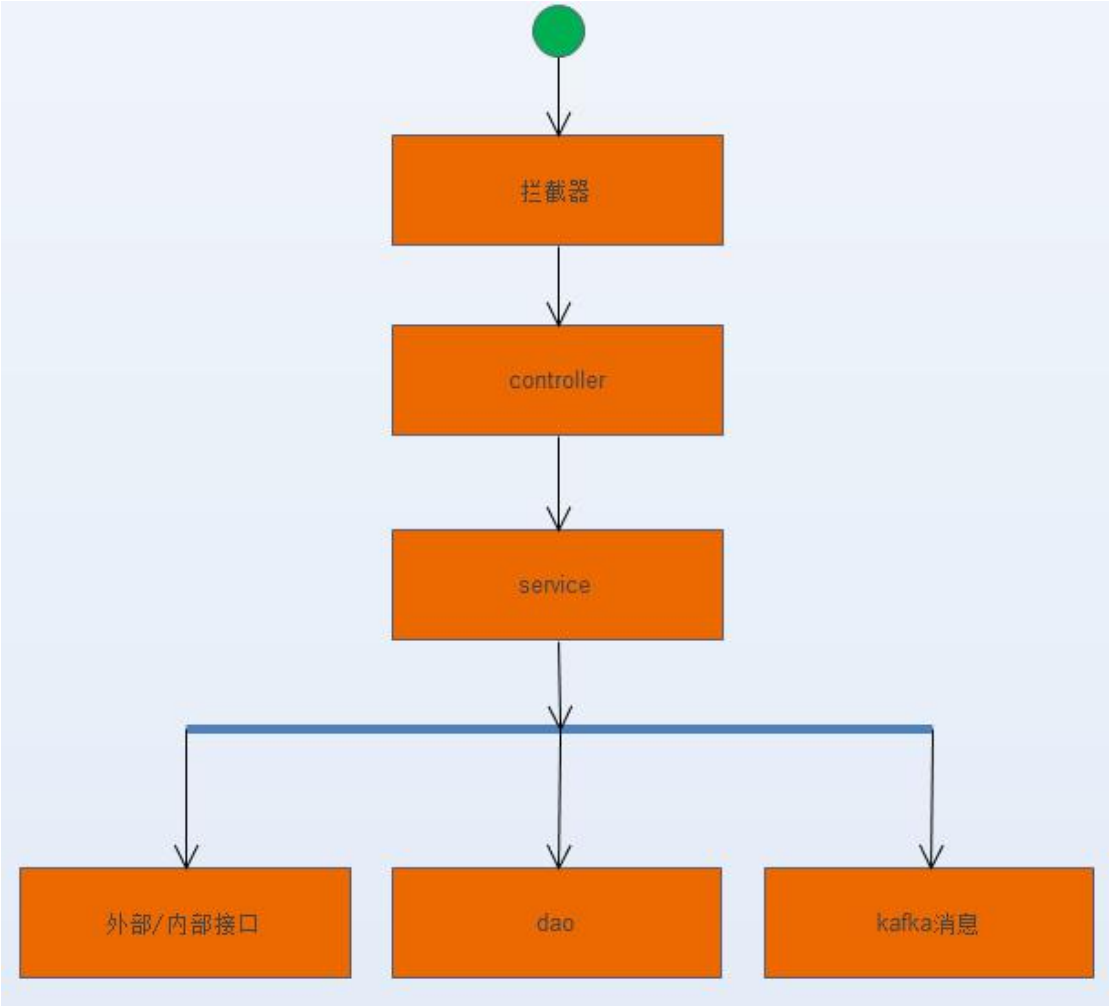
开发语言使用 java,采用基础框架 spring mybatis 缓存使用 redis 系统建异步通知采用消息中间件 kafka 系统间的同步接口采用遵循 restful 风格的 http 协议.

系统采用 maven 构建,运行在 jdk1.8 以上环境,开发工具建议 eclipse 或 myeclipse

2. 系统层次划分

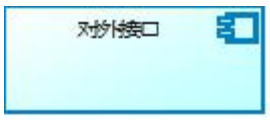
支付中心各个系统除定时任务外均为 web 项目,故将系统内部划分为通用的三层架构,分别为表示层 业务层 数据层.

其中表示层使用 spring mvc 实现,业务层使用 spring 实现事物,数据层数据库操作用 mybatis 框架,操作 redis 使用 jedis 实现.



3. 主要系统结构和代码结构

- 关键的系统代码结构



- 消息中间件
  系统对外交互的异步通讯方式采用 kafka 消息中间件形式,如支付结果,退款,支付关闭等

4. 接口设计

基本原则:系统对外提供的接口全部为同步接口,同时使用遵循 restful 风格的 http 协议

接口安全限制:因为 http 协议接口通过外网形式传输,接口的安全性要求高,采用支持多种加密方式加密数据,具体支持:MD5 签名及 AES RSA,同时对接口做幂等

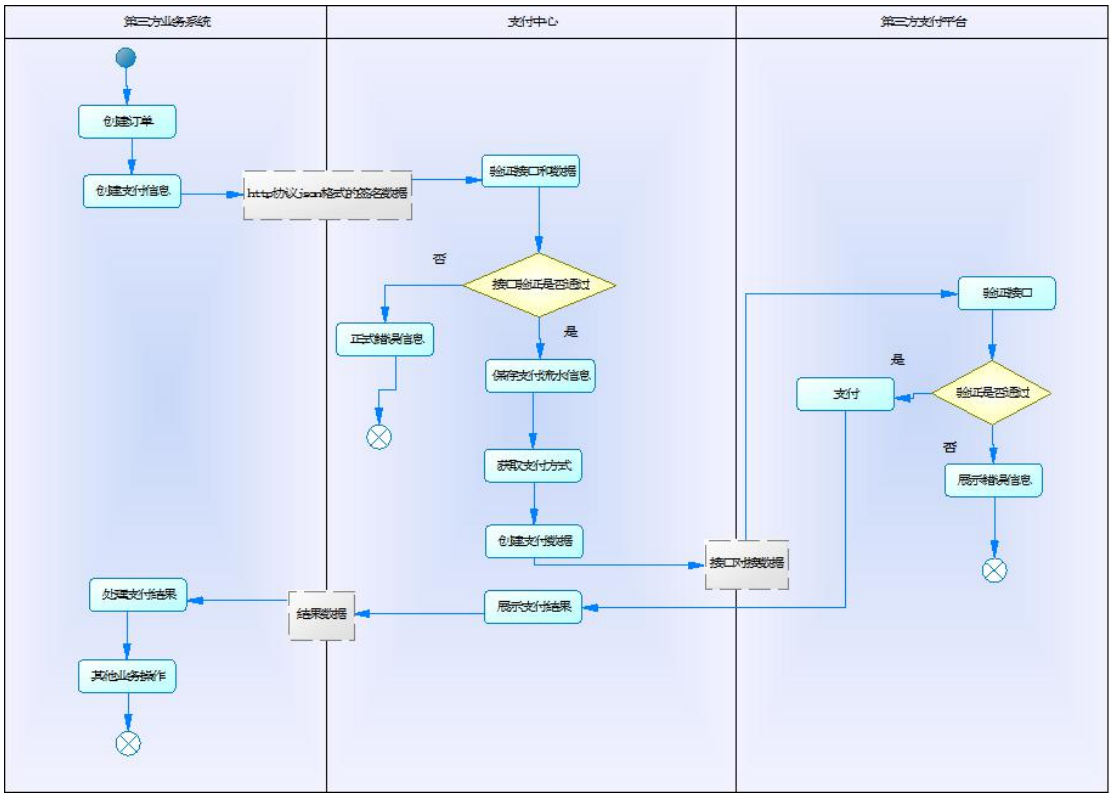
六、运行视图

1. 概述

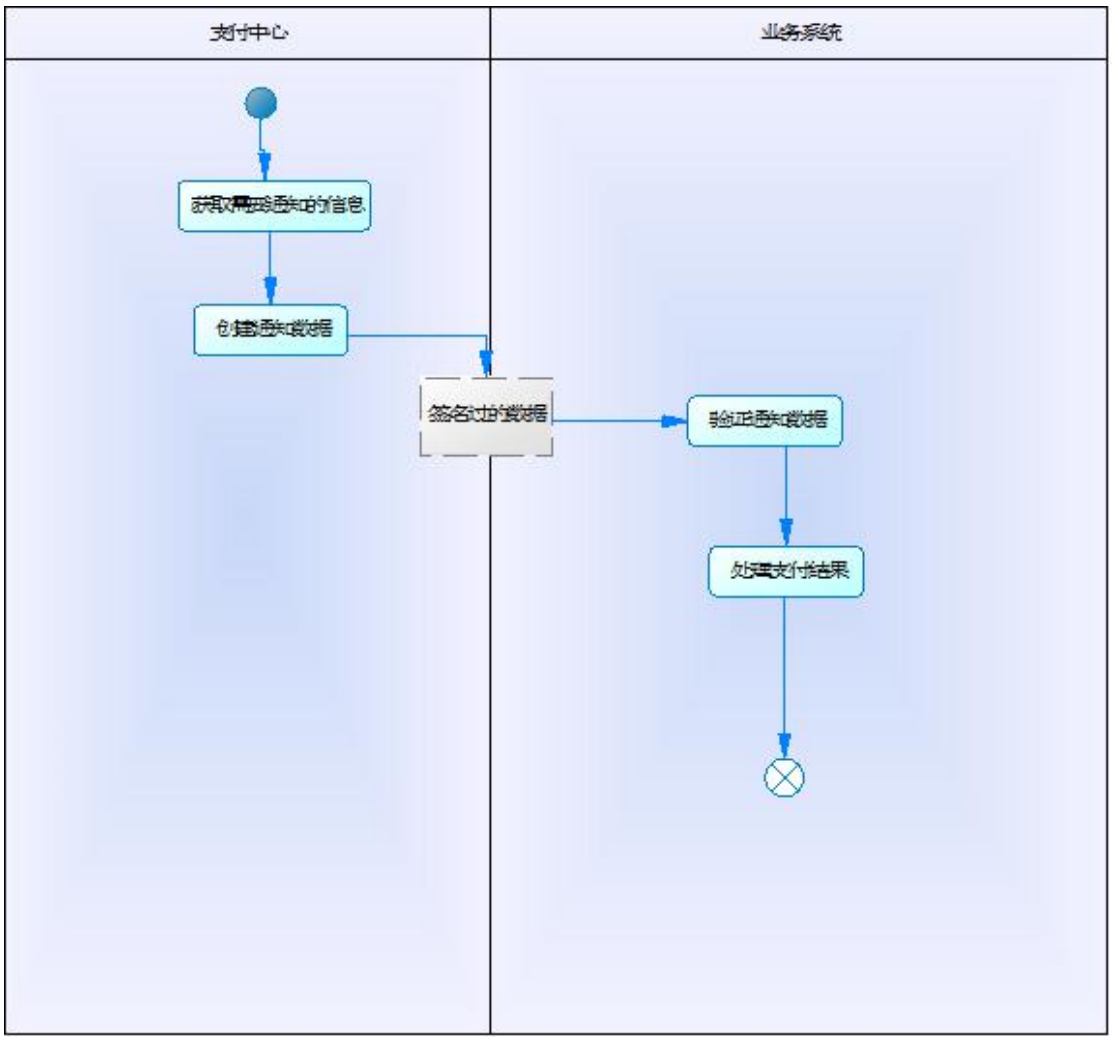
本部分只描述支付中心关键部分的运行视图。

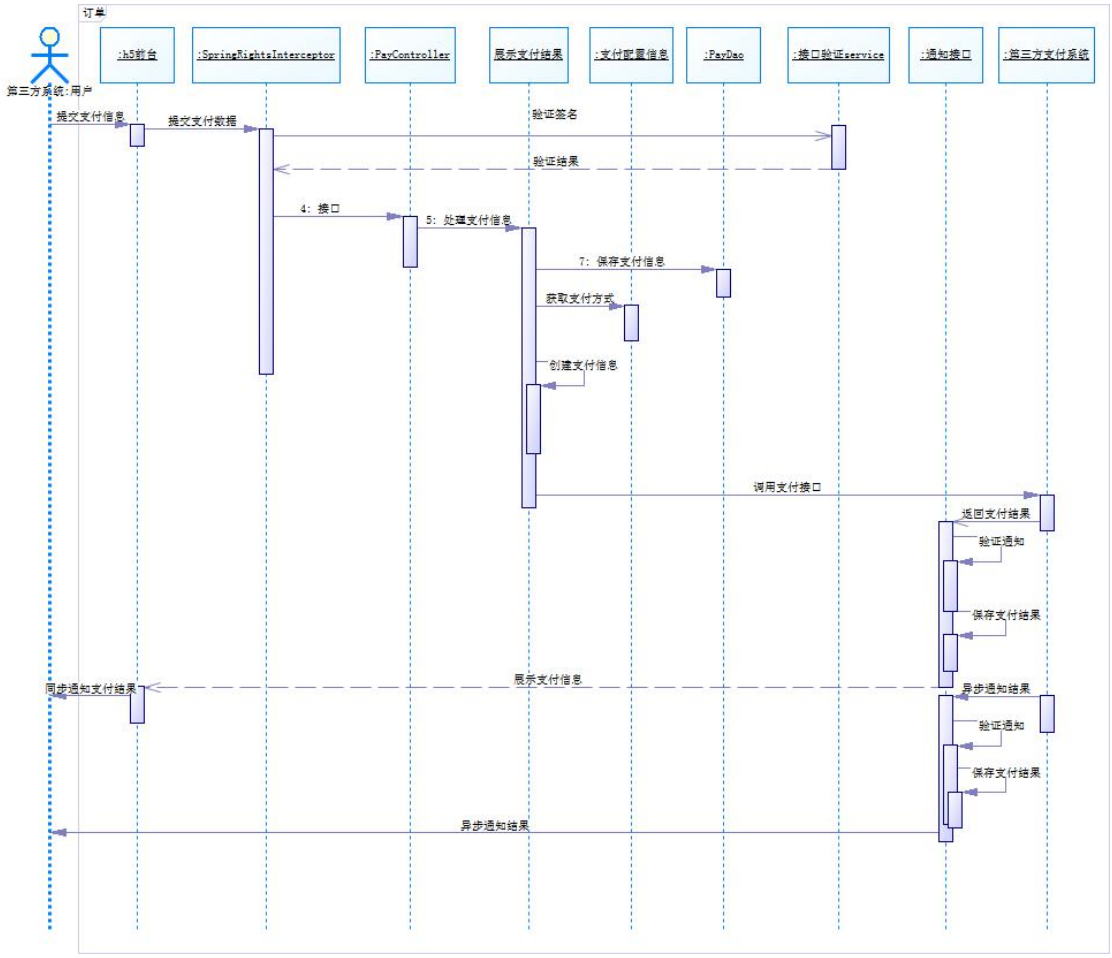
2. 关键运行视图

- 支付接口：

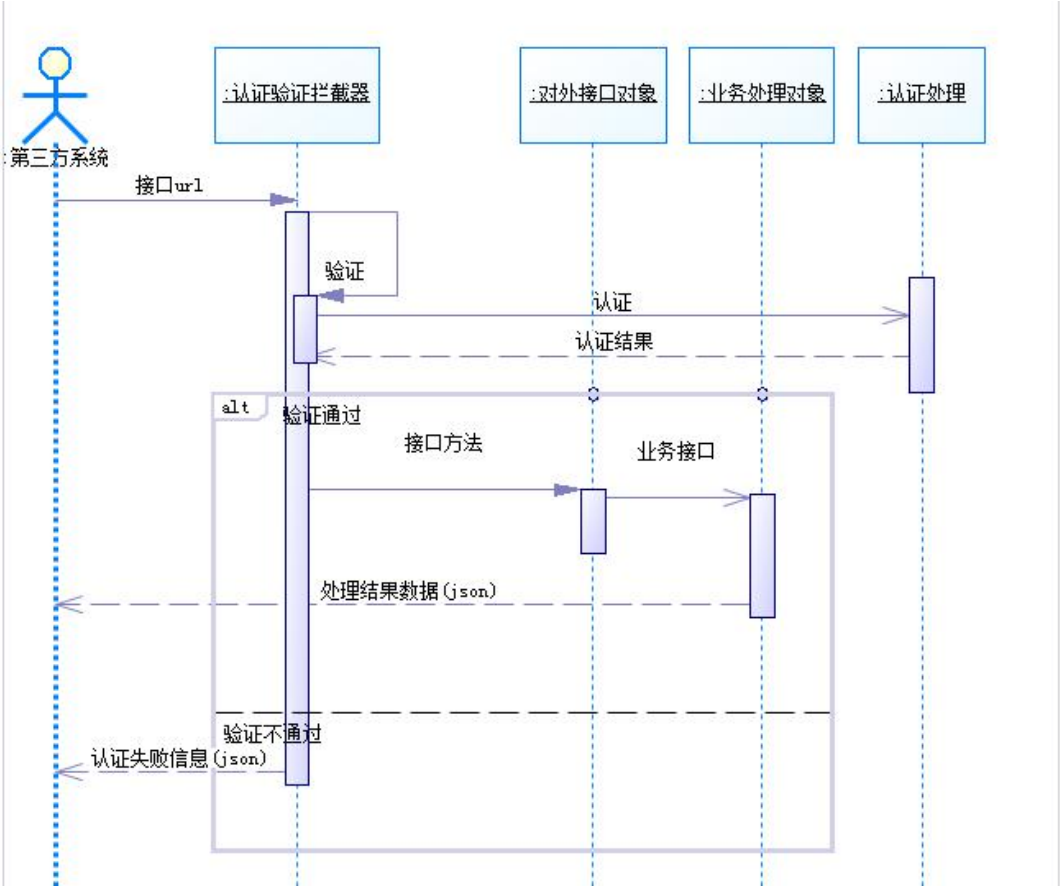

第三方业务系统
支付中心
第三方支付平台
创建订单
创建支付信息
http协议json格式的签名数据
验证接口和数据
否
接口验证是否通过
是
正式错误信息
保存支付流水信息
获取支付方式
创建支付数据
接口对接数据
验证接口
是
支付
验证是否通过
否
展示错误信息
处理支付结果
结果数据
展示支付结果
其他业务操作

- 通知服务：


支付中心
业务系统
签名过的数据
验证通知数据


订单
第三方系统:用户
:h5前台
:SpringRightsInterceptor
:PayController
展示支付结果
:支付配置信息
:PayDao
:接口验证service
:通知接口
:第三方支付系统
提交支付信息
提交支付数据
验证签名
验证结果
4：接口
5：处理支付信息
7：保存支付信息
获取支付方式
创建支付信息
调用支付接口
返回支付结果
验证通知
保存支付结果
展示支付信息
同步通知支付结果
异步通知结果
验证通知
保存支付结果
异步通知结果

- 同步 rest 验证：



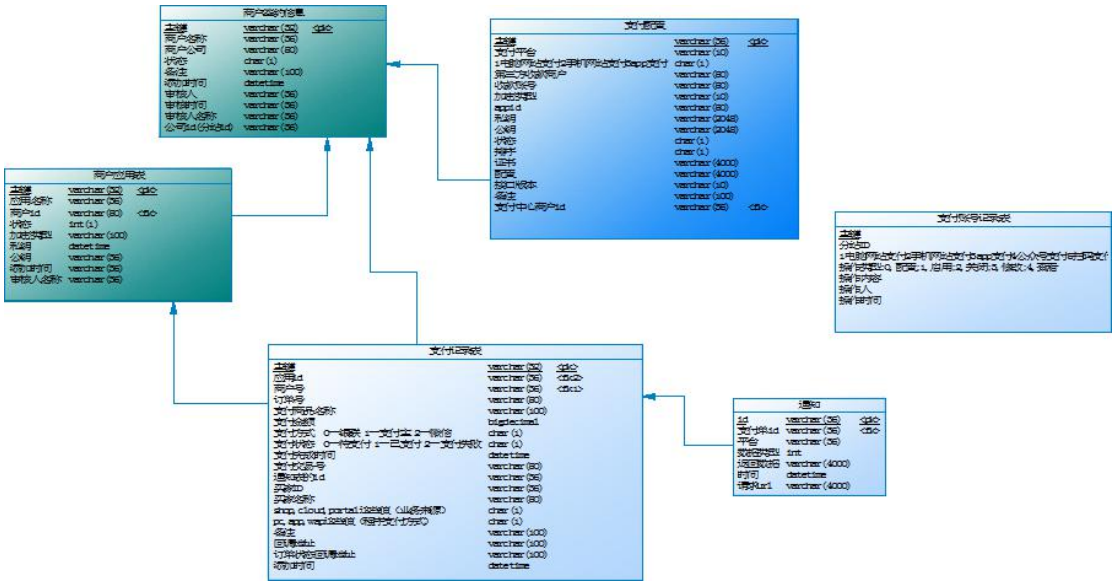
七、数据视图

1. 概述

数据库采用 mysql,缓存使用 redis

需要持久化的结构化数据统一使用 mysql 来存储,无需持久化数据或需要缓存或加快速度或减轻数据库压力的数据全部放入 redis;

数据库模型:



八、部署方式

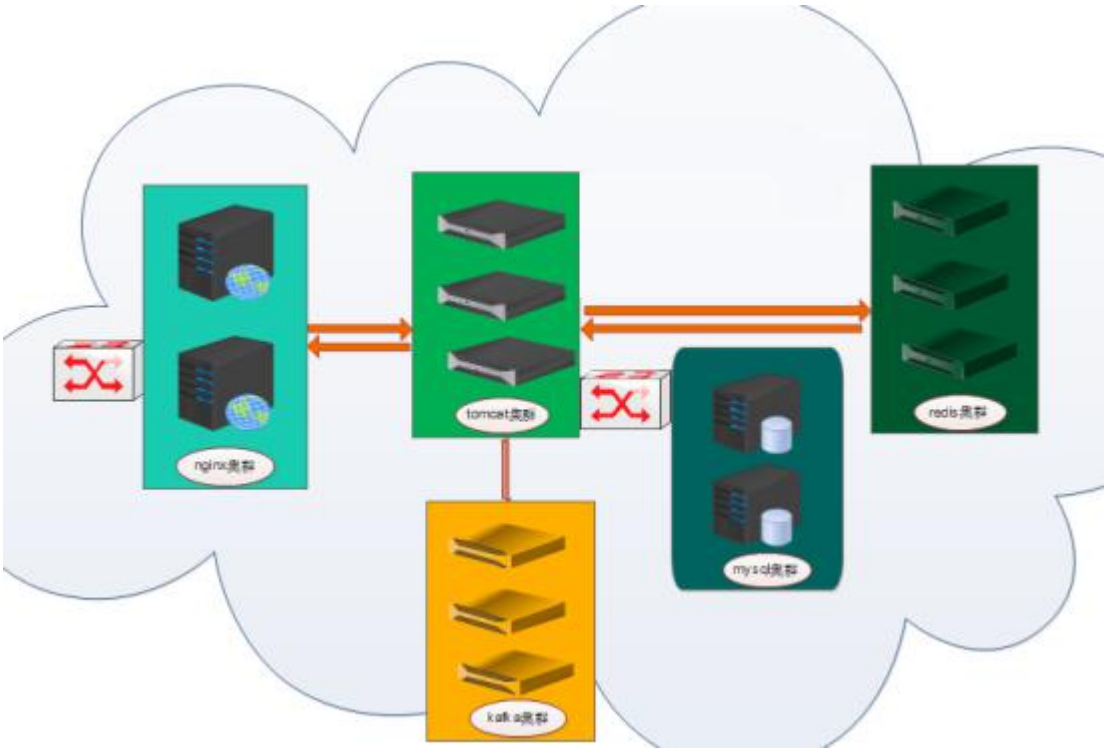
1. 概述

由于本方案采用 BS 结构设计，客户端只需要安装基本的操作系统和浏览器就可以使用本系统. 同时考虑访问量和高可用,服务器全部采用集群方式提供服务,避免单点故障,本系统的网络结构主要由数据库服务器集群、tomcat 服务器集群、nginx 服务器 redis 服务器集群,kafka 消息中间件集群等组成。

其中服务器软件版本说明:

- 操作系统 centos7.0 以上版本
- Nginx 采用 1.9 以上版本
- JDK 采用 1.8 以上版本
- Tomcat 采用 8.5 以上版本
- 数据库 mysql 建议采用 5.7 以上版本
- Redis 建议采用 3.2 以上版本
- Kafka 采用 0.9.0.1 以上版本

2. 部署图



九、其他说明

支付中心已有老系统在运行,考虑到新中心可以全新设计不受老系统限制同时使老系统数据兼容有效,对老系统进行数据迁移;迁移部分:支付信息具体规则细化中