

- 1、业务功能完善
- 2、增加gateway网关
  - gateway有什么用？
  - 引入Gateway的重要步骤
- 3、配置信息迁移到Nacos
  - 为什么要将配置信息迁移到Nacos？
  - 配置迁移过程
- 4、阶段总结
  - 1、构建特色用户体系
  - 2、典型微服务架构
  - 3、高并发场景常见优化方案

## 五、用户管理微服务架构整合

阶段目标：完善用户管理相关服务。整合完整微服务体系。

# 1、业务功能完善

---

主要完善三个功能：

- 1、登录页面短信推送功能。 - 整合容联云短信登录平台。
- 2、用户登录功能 - 手机号登录，首次登录自动创建用户。
- 3、用户登录状态缓存功能 - 登录完成后，生成token，保存到前端Cookie。通过token可查询用户登录状态。

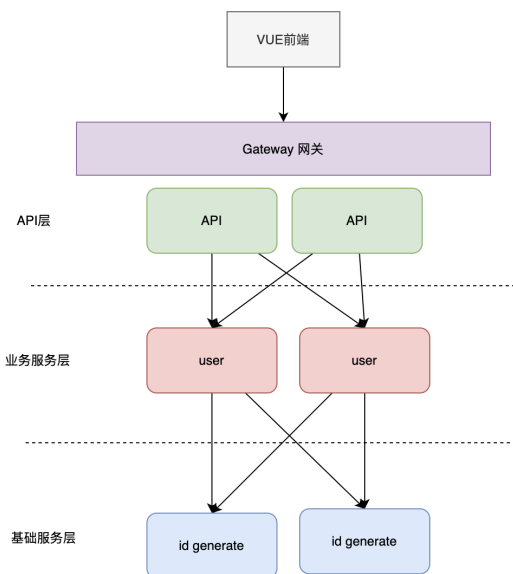
# 2、增加gateway网关

---

## gateway有什么用？

---

现状总结：前端对接live-api，这是一对一的访问，无法水平扩展。live-api对接后面的dubbo服务，这是可以水平扩展的。



gateway的作用：

- 1、作为live-api之前的统一门户，让下游的微服务具备负载均衡，水平扩展的能力。
- 2、gateway可以与微服务注册中心链接，实现微服务无感知动态扩容
- 3、对于无法访问的服务，可以做到自动熔断，不需要人工参与
- 4、对后端请求统一进行权限验证。关键请求需要登录后才能访问。

## 引入Gateway的重要步骤

### 1、核心Maven依赖

```
<spring-cloud-starter-gateway.version>4.0.6</spring-cloud-starter-gateway.version>
<spring-cloud-starter-loadbalancer.version>4.0.3</spring-cloud-starter-loadbalancer.version>

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
    <version>${spring-cloud-starter-gateway.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-loadbalancer</artifactId>
    <version>${spring-cloud-starter-loadbalancer.version}</version>
</dependency>
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bootstrap</artifactId>
    <version>${spring-cloud-bootstrap.version}</version>
</dependency>
```

## 2、配置路由转发规则

```
spring:
  application:
    name: tl-live-gateway
  cloud:
    nacos:
      discovery:
        server-addr: nacos.tllive.com:8848
        namespace: tl-live
    gateway:
      discovery:
        locator:
          enabled: true
      routes:
        - id: tl-live-api
          uri: lb://tl-live-api
          predicates:
            - Path=/user/**
        # 测试路由规则
        - id: gateway-test
          uri: lb://tl-live-user-provider
          predicates:
            - Path=/test/**
```

## 3、实现GlobalFilter接口，定制自定义过滤规则(可选)

自定义网关过滤规则：

- 1、定义请求白名单，白名单请求不做限制，直接通过
- 2、非白名单请求，通过token判断是否有用户登录
- 3、没有登录，则拒绝请求
- 4、有登录信息，则将UserID通过header传给下游服务。

# 3、配置信息迁移到Nacos

## 为什么要将配置信息迁移到Nacos?

所有应用的配置信息集中管理，实现配置的动态化、实时化、可视化、可控化。提高整个系统的可靠性、可维护性和可扩展性。

## 配置迁移过程

- 1、核心Maven依赖

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bootstrap</artifactId>
  <version>${spring-cloud-bootstrap.version}</version>
</dependency>
```

## 2、增加bootstrap.yml文件，指定配置中心的配置文件

spring-cloud-starter-alibaba-nacos-config 找配置文件的方式：

Spring Cloud Alibaba Nacos Config 目前提供了三种配置能力从 Nacos 拉取相关的配置。

A：通过 `spring.cloud.nacos.config.shared-configs[n].data-id` 支持多个共享 Data Id 的配置

B：通过 `spring.cloud.nacos.config.extension-configs[n].data-id` 的方式支持多个扩展 Data Id 的配置

C：通过内部相关规则(应用名、应用名+ Profile)自动生成相关的 Data Id 配置

当三种方式共同使用时，他们的一个优先级关系是： $A < B < C$

## 3、基础的迁移思想：

公用组件抽取成统一配置。每个应用只保留自己独有的业务配置

- MySQL和Redis抽取成通用的tl-live-common-db。
- 未来MQ也可以抽取成通用配置。

tl-live-user-provider配置示例

```
spring:
  profiles:
    active: dev
  application:
    name: tl-live-user-provider
  cloud:
    nacos:
      discovery:
        server-addr: nacos.tllive.com:8848
        namespace: tl-live
      config:
        import-check:
          enabled: false
        file-extension: yaml
        server-addr: nacos.tllive.com:8848
        namespace: tl-live
        shared-configs:
          - data-id: tl-live-common-dubbo.yaml
            refresh: true
          - data-id: tl-live-common-db.yaml
            refresh: true
```

## 4、阶段总结

---

### 1、构建特色用户体系

---

如何支持微信登录、微博登录等等其他登录方式？

### 2、典型微服务架构

---

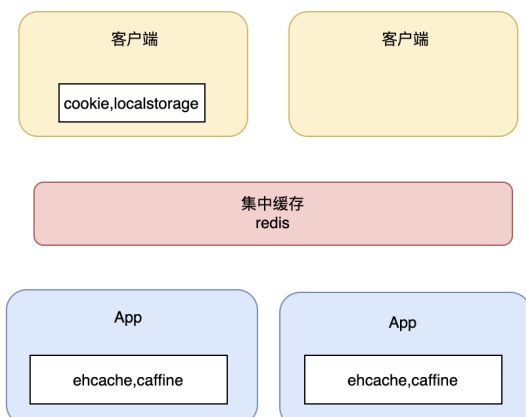
Spring Cloud Alibaba 仓库：<https://github.com/alibaba/spring-cloud-alibaba/>

微服务架构的核心：各司其职。

### 3、高并发场景常见优化方案

---

多级缓存方案



分库分表方案

