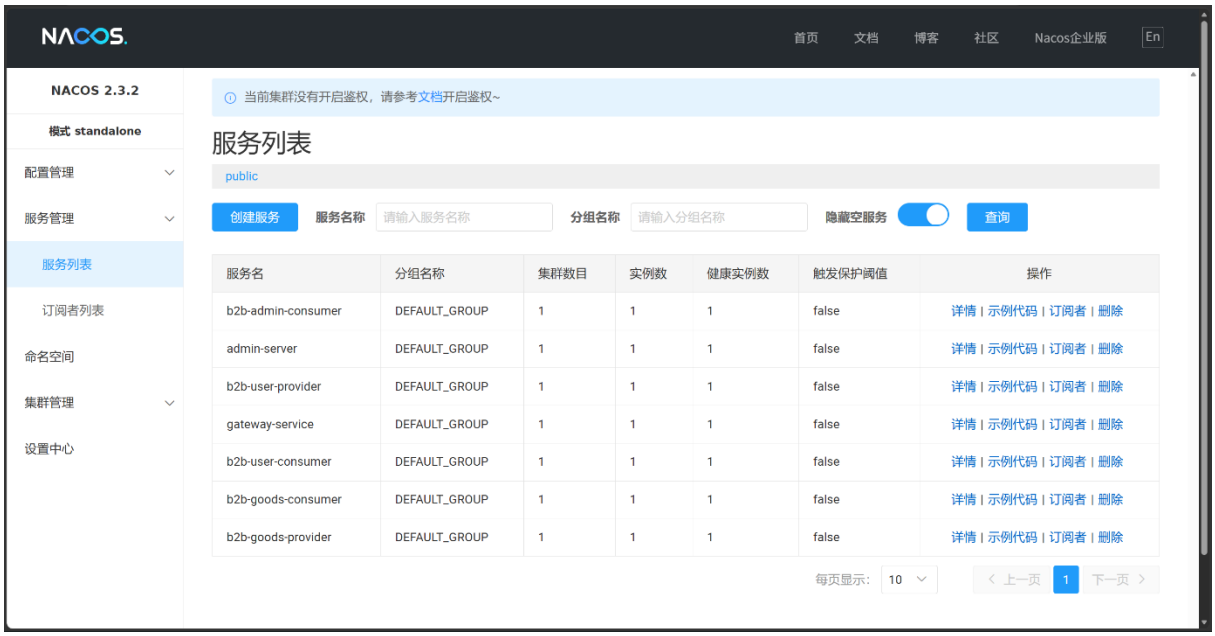
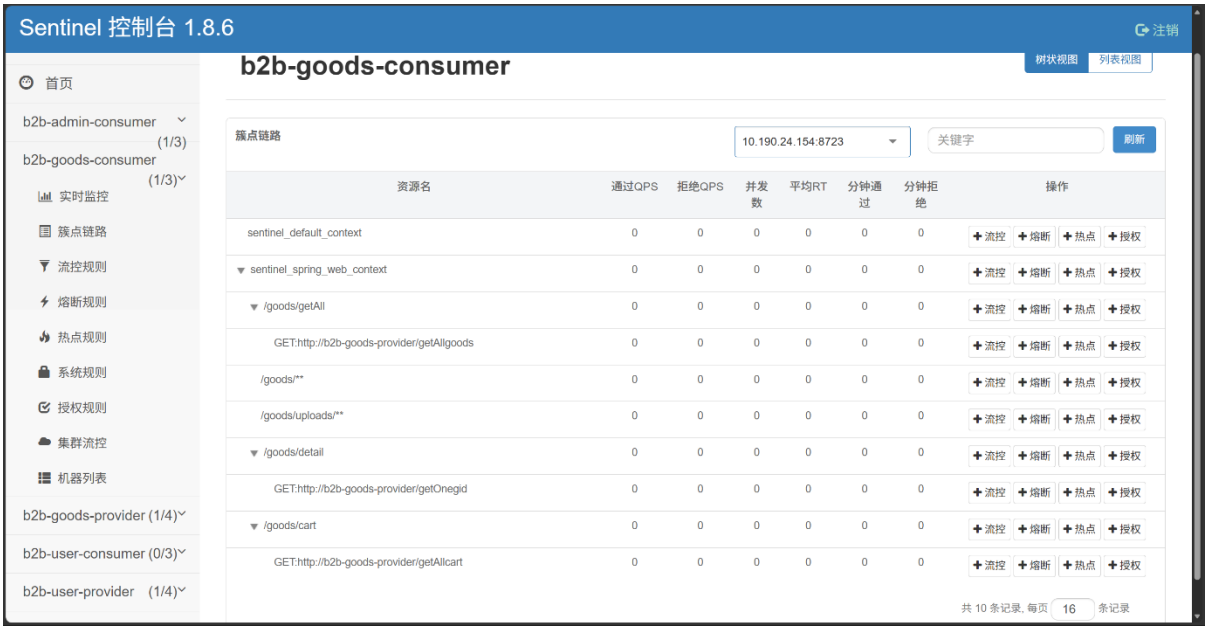
附件 1：部分代码和效果截图

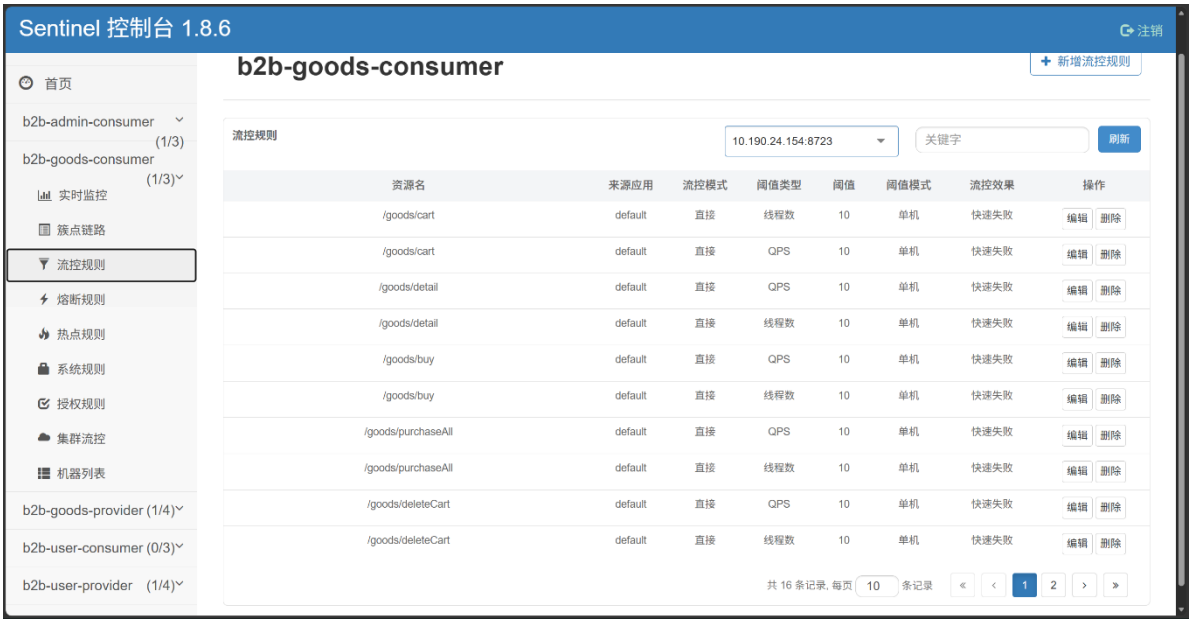
▼图1： Nacos 服务列表页



▼图2： Sentinel 簇点链路



▼图3： Sentinel 流控规则

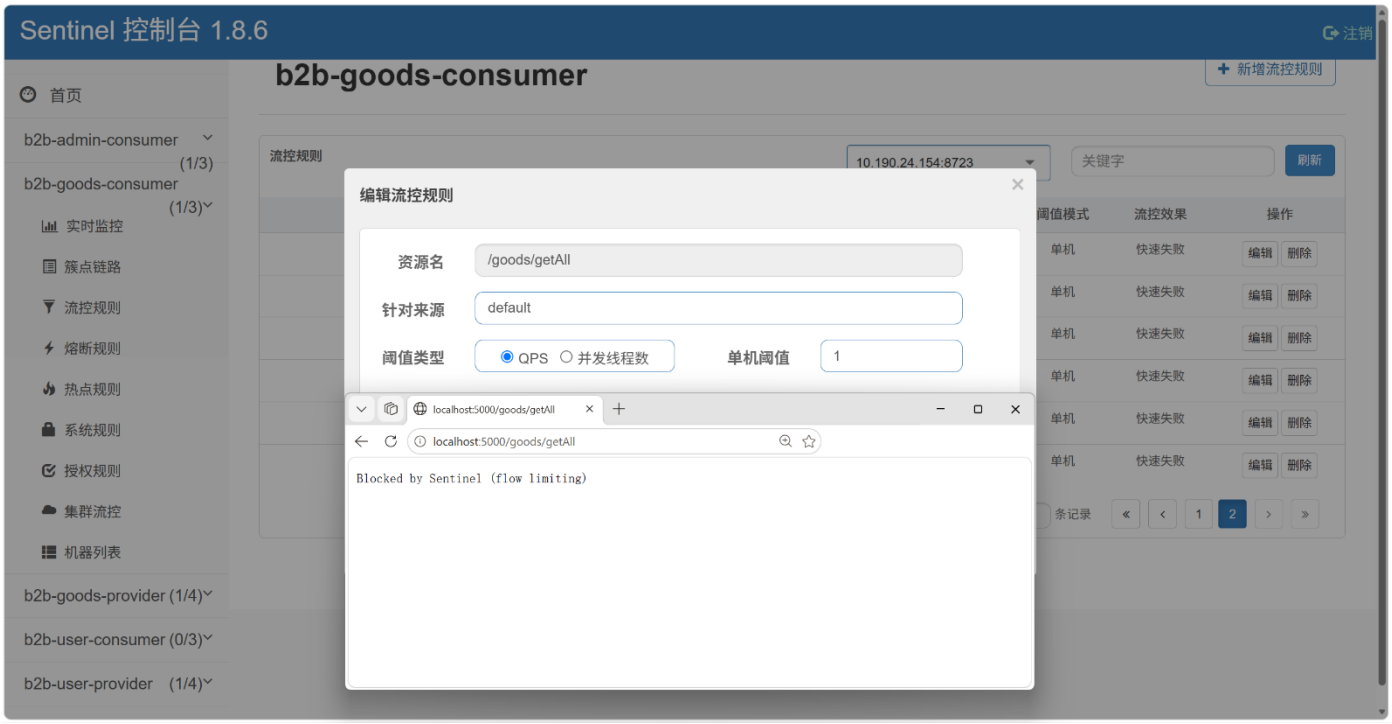


▼图4： Sentinel 硬编码规则（规则加载器抽取到公共类中，以 User Consumer 为例）

```java
public final class SentinelRuleLoader {

    private SentinelRuleLoader() {
    }

    public static void loadDefaultRules(String... resources) {
        List<FlowRule> flowRules = new ArrayList<>();
        List<DegradeRule> degradeRules = new ArrayList<>();

        for (String resource : resources) {
            flowRules.add(createFlowRule(resource, RuleConstant.FLOW_GRADE_QPS));
            flowRules.add(createFlowRule(resource, RuleConstant.FLOW_GRADE_THREAD));
            degradeRules.add(createDegradeRule(resource));
        }

        FlowRuleManager.loadRules(flowRules);
        DegradeRuleManager.loadRules(degradeRules);
    }

    private static FlowRule createFlowRule(String resource, int grade) {
        FlowRule rule = new FlowRule();
        rule.setResource(resource);
        rule.setGrade(grade);
        rule.setCount(10);
        rule.setLimitApp("default");
        return rule;
    }

    private static DegradeRule createDegradeRule(String resource) {
        DegradeRule rule = new DegradeRule();
        rule.setResource(resource);
        rule.setGrade(RuleConstant.DEGRADE_GRADE_EXCEPTION_RATIO);
        rule.setCount(0.2);
        rule.setTimeWindow(10);
        rule.setMinRequestAmount(5);
        rule.setStatIntervalMs(1000);
        return rule;
    }
}
```

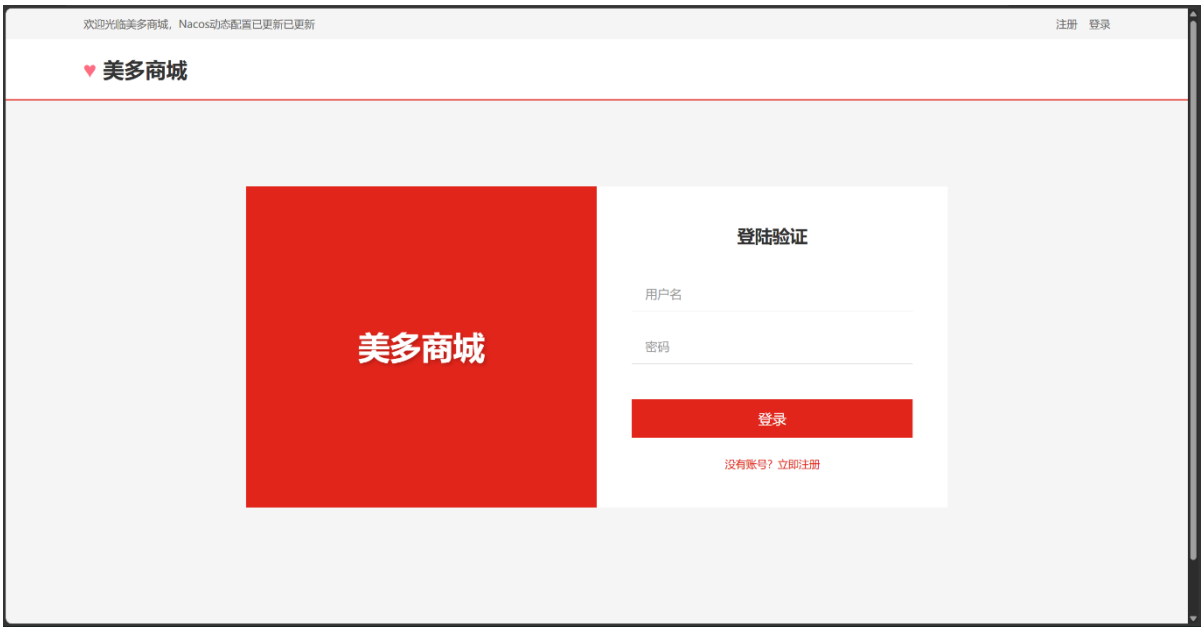
```java
@Configuration
public class SentinelRuleConfig {

    @EventListener(ApplicationReadyEvent.class)
    public void initRules() {
        SentinelRuleLoader.loadDefaultRules(
                "/user/toregister",
                "/user/tologin",
                "/user/register",
                "/user/login"
        );
    }
}
```

▼图5： Sentinel 限流阻断效果

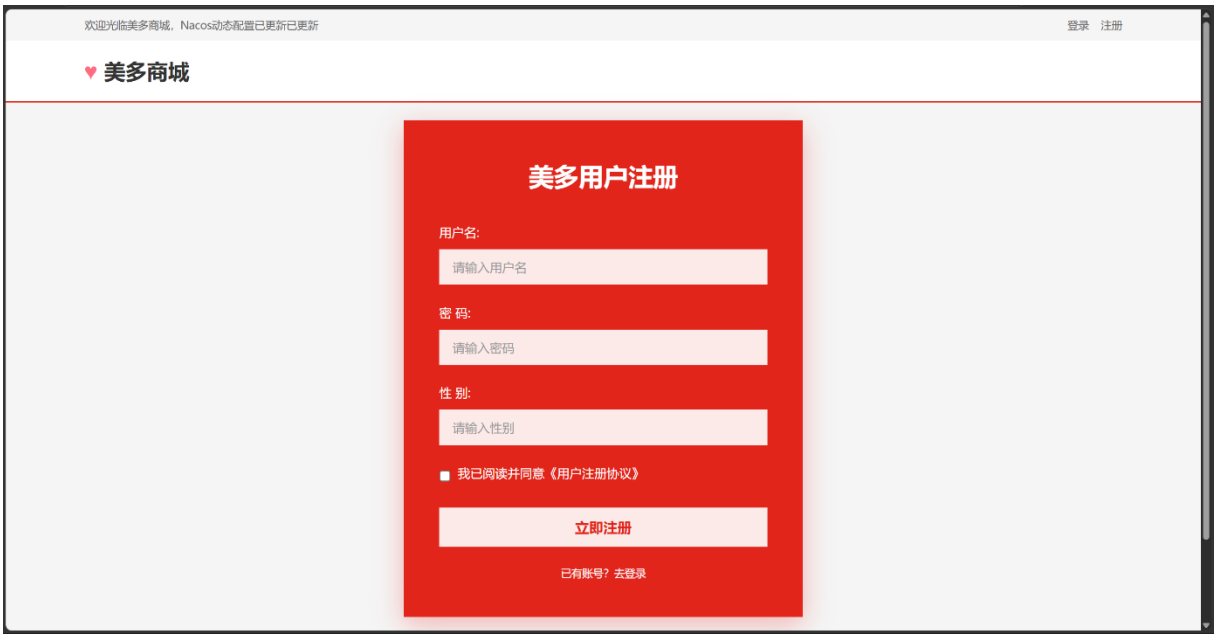


附件 1：部分代码和效果截图

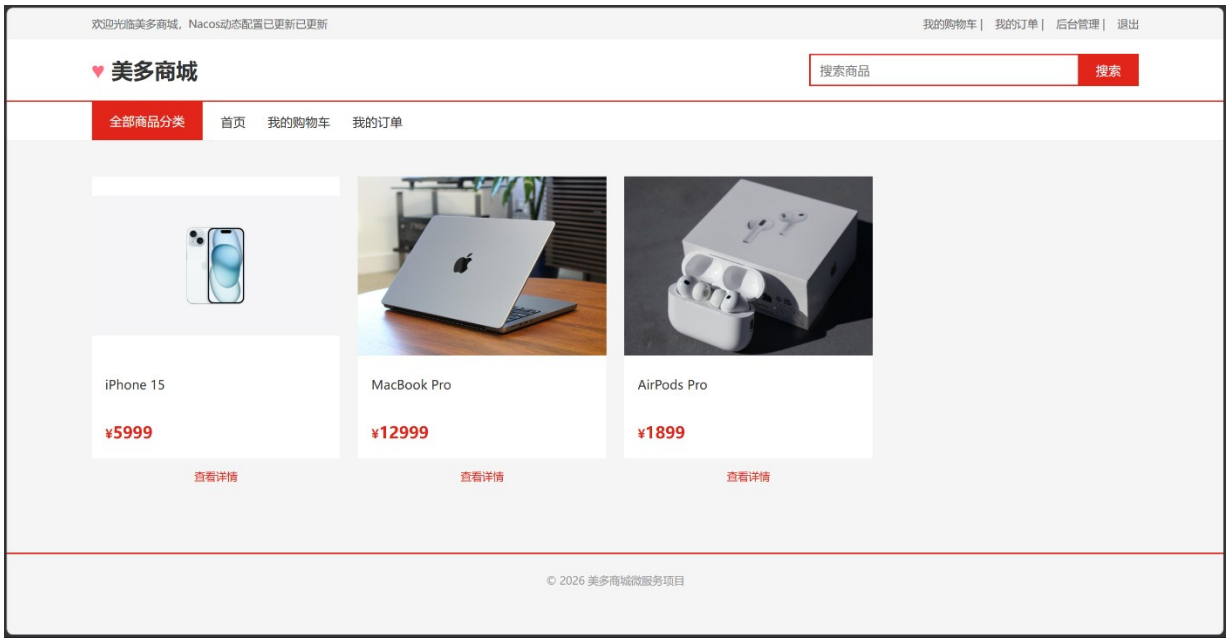
▼图6： 前端登录页



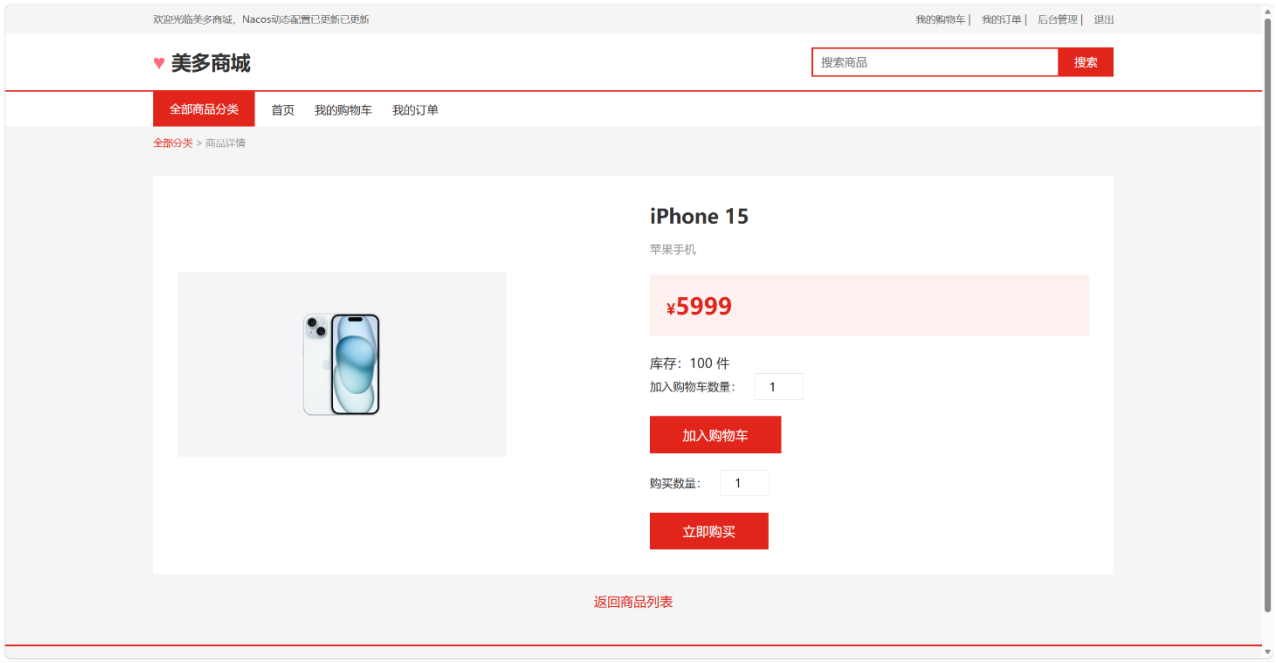
▼图7： 前端注册页



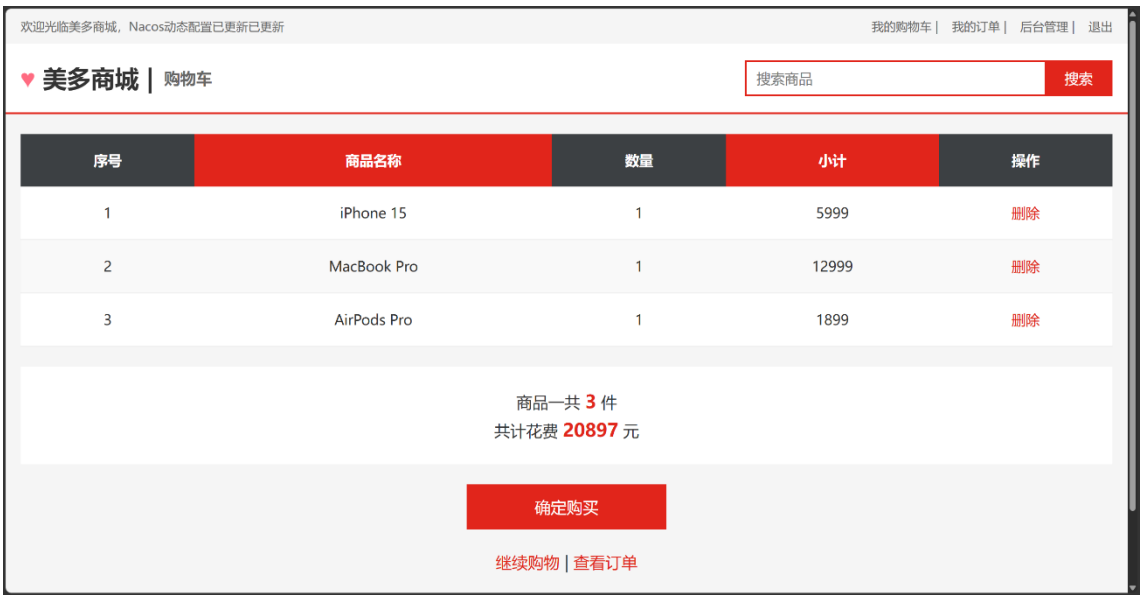
▼图8： 前端商品列表页



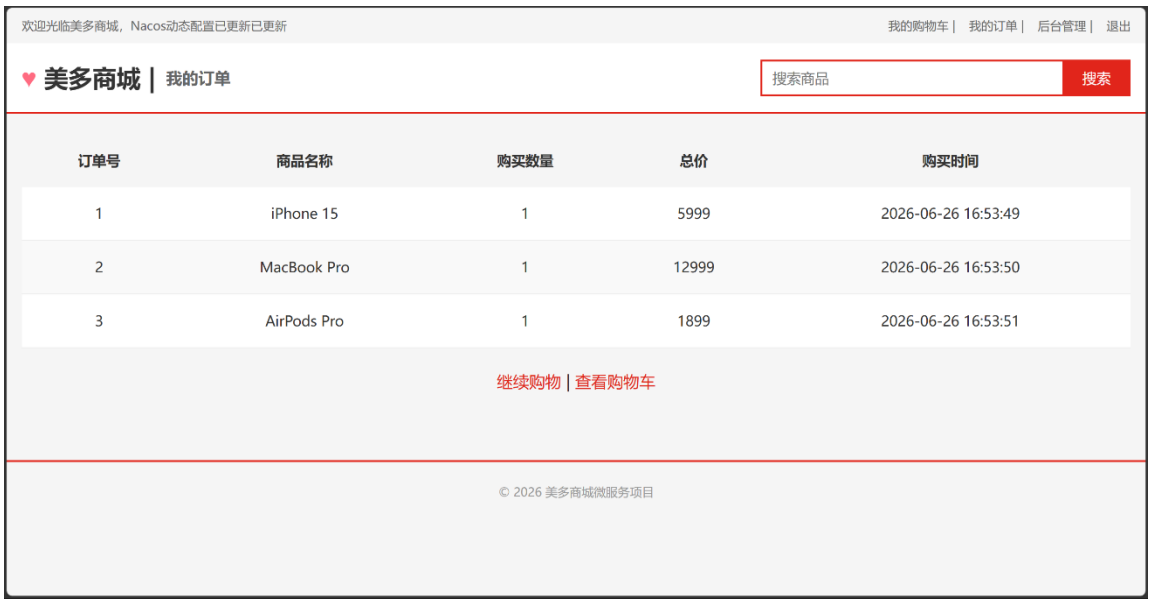
▼图9： 前端商品详情页



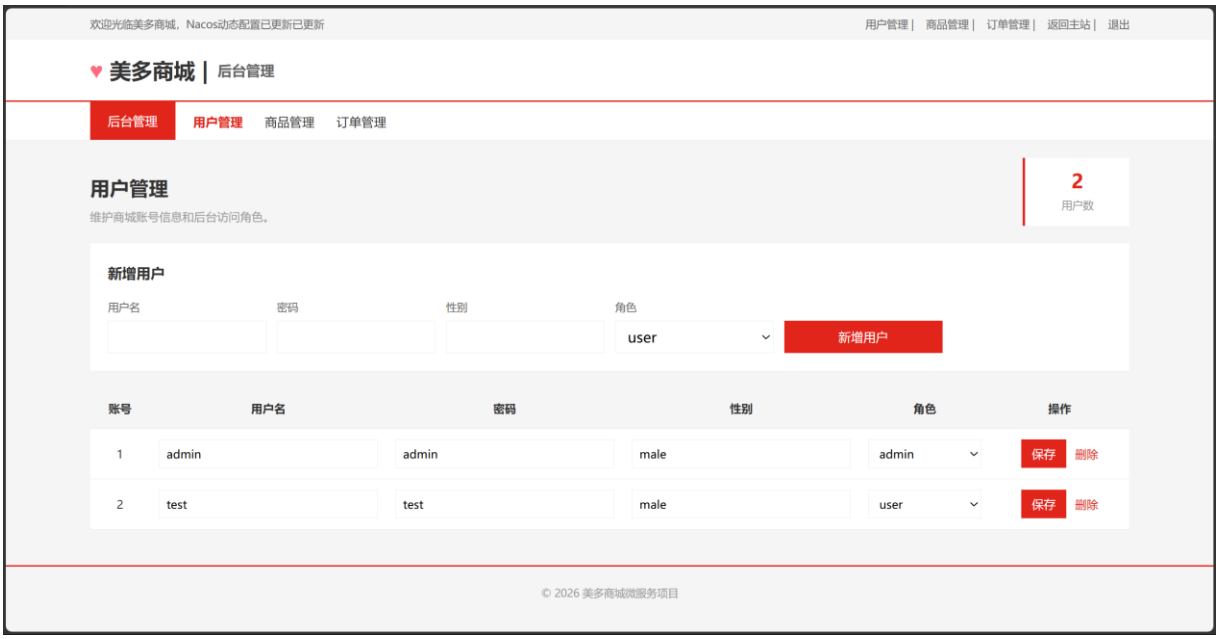
▼图10： 购物车页



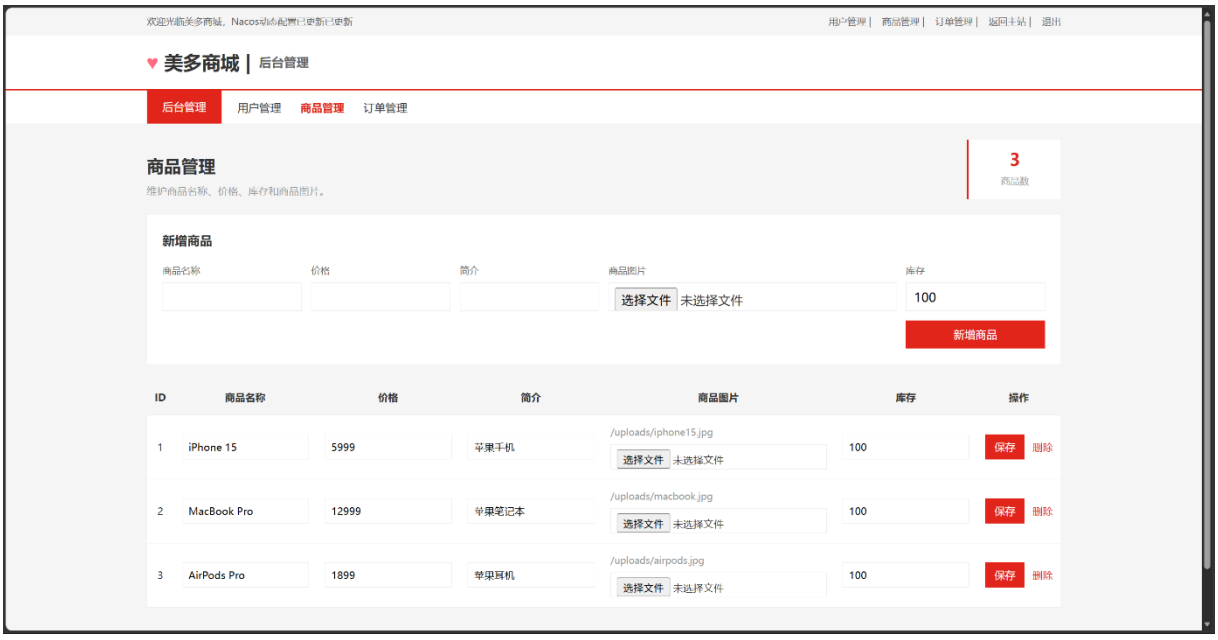
▼图11： 订单页



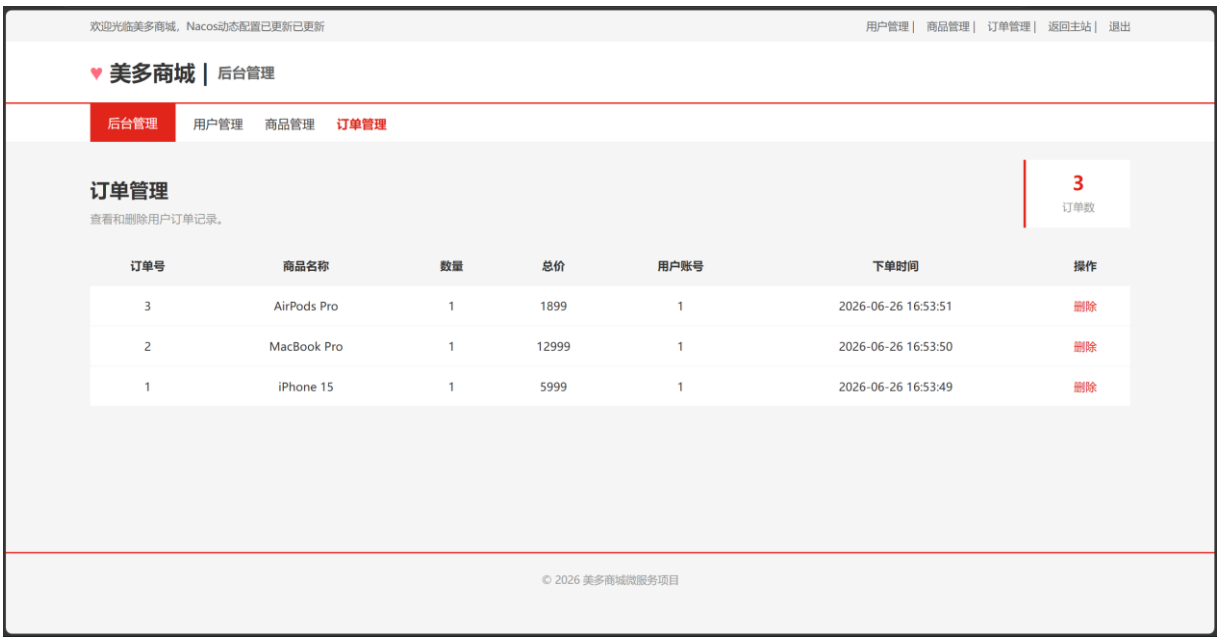
▼图12：　后台管理用户页



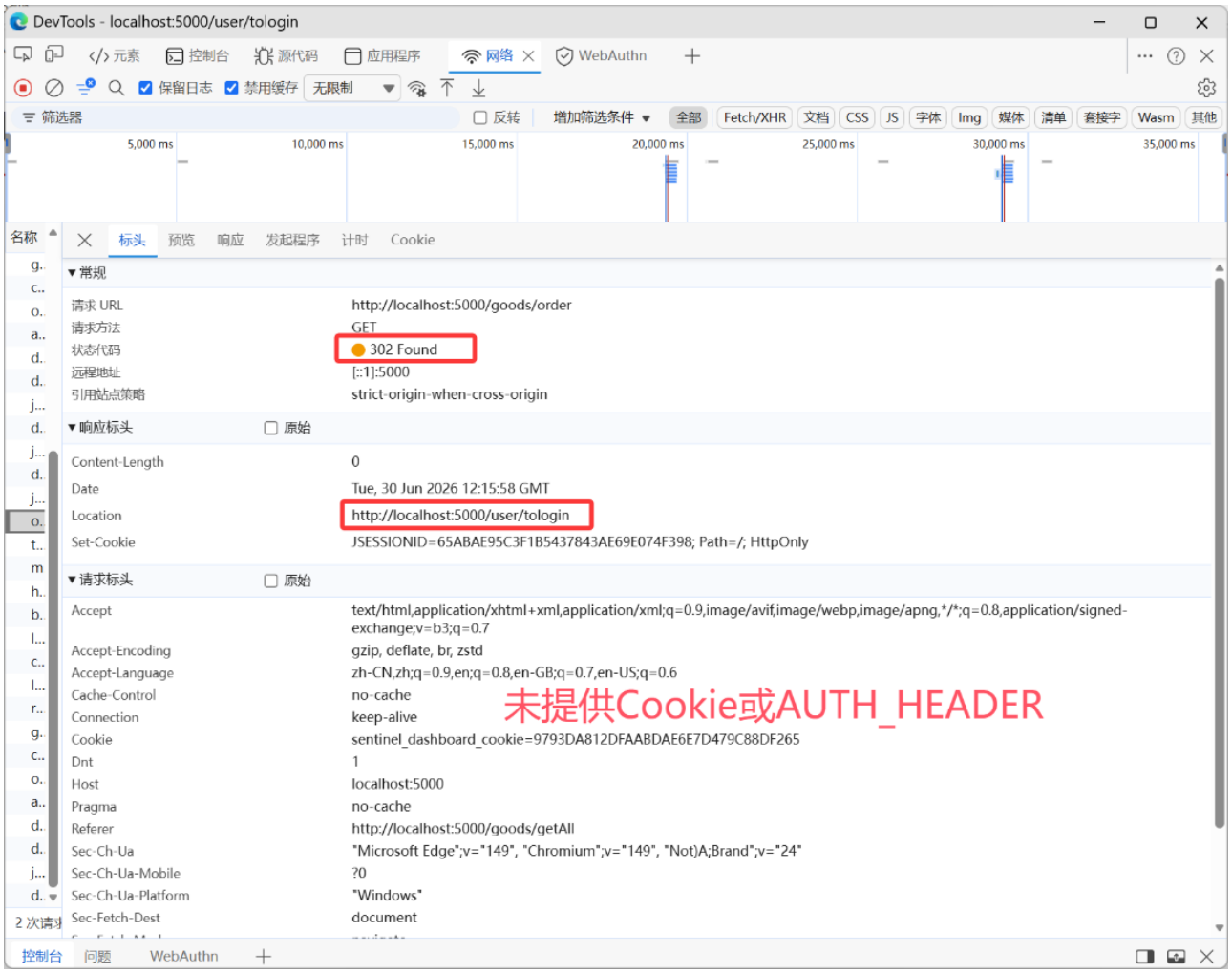
▼图13：　后台管理商品页



▼图14：　后台管理订单页



▼图15：　未登录用户添加商品到购物车时将跳转到登录页

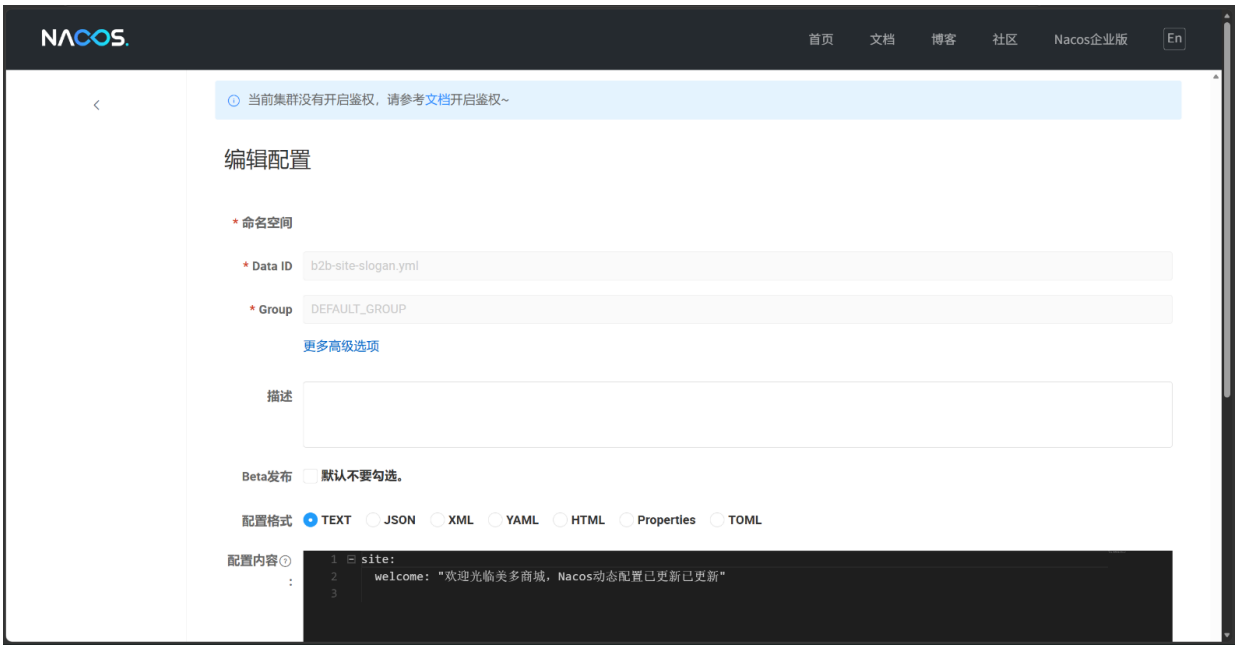


▼图16：　拦截未登录用户并跳转到登录页的关键代码

```java
// 验证用户登录状态拦截器
@Component
public class SysInterceptor implements HandlerInterceptor {

    private static final Logger log = LoggerFactory.getLogger(SysInterceptor.class);

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
                             Object handler) throws Exception {
        Optional<User> user = AuthTokenUtil.resolveUser(request);
        if (user.isPresent()) {
            AuthTokenUtil.attachRequestUser(request, user.get());
            log.info("我证明用户已通过JWT认证: {}", user.get().getUname());
            return true;
        }

        if (AuthCookieUtil.findAuthToken(request).isPresent()) {
            AuthCookieUtil.expireAuthToken(response);
        }
        log.info("我证明用户没有登录");
        String redirectURL = LoginUrlUtil.resolveLoginUrl(request);
        log.info("重定向到: {}", redirectURL);
        response.sendRedirect(redirectURL);
        return false;
    }
}
```

▼图17：　　Nacos 配置中心

▼图18：　　Nacos 配置中心详情页

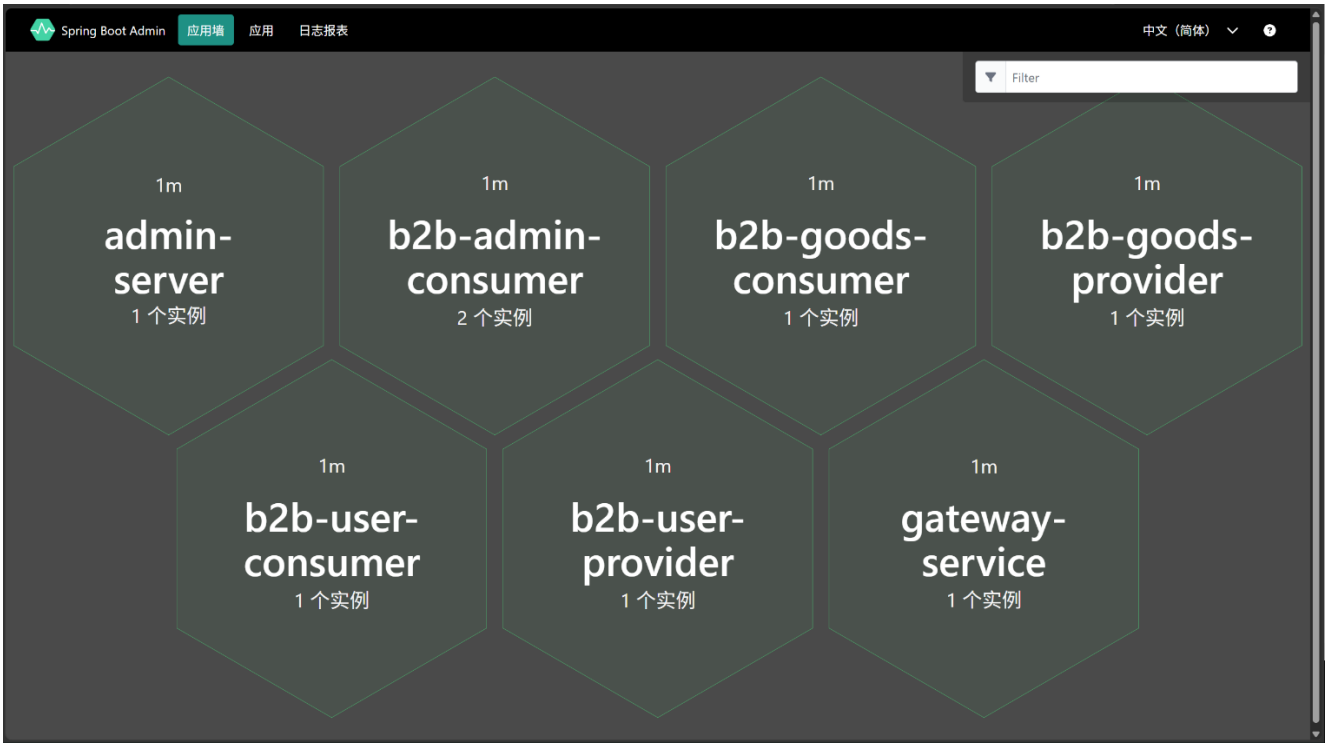


▼图19：　　Nacos 动态配置更新示例

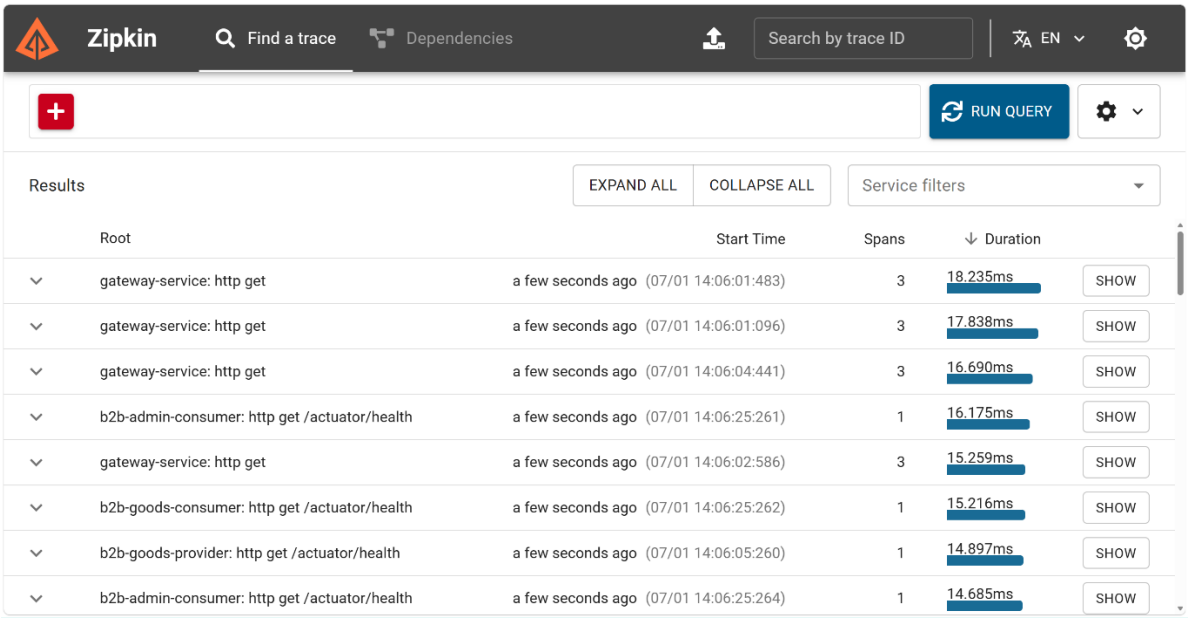
▼图20：　Nacos 配置中心自动创建配置文件代码

```
try {
    // 创建配置服务
    Properties properties = new Properties();
    properties.put("serverAddr", serverAddr);
    ConfigService configService = NacosFactory.createConfigService(properties);

    // 检查配置是否已存在
    String existingConfig = null;
    try {
        existingConfig = configService.getConfig(dataId, group, 5000);
    } catch (NacosException e) {
        log.debug("配置不存在，将创建新配置 b2b-site-slogan");
    }

    if (existingConfig != null && !existingConfig.isEmpty()) {
        log.info("配置已存在:\n{}", existingConfig);
    } else {
        // 发布配置
        boolean isPublishOk = configService.publishConfig(dataId, group, content);
        if (isPublishOk) {
            log.info("配置发布成功! ");
            log.info("配置内容:\n{}", content);
        } else {
            log.error("配置发布失败");
        }
    }

    // 验证配置
    String config = configService.getConfig(dataId, group, 5000);
    log.info("读取配置成功，内容:\n{}", config);

} catch (NacosException e) {
    log.error("Nacos 配置初始化失败: {}", e.getMessage(), e);
}
```

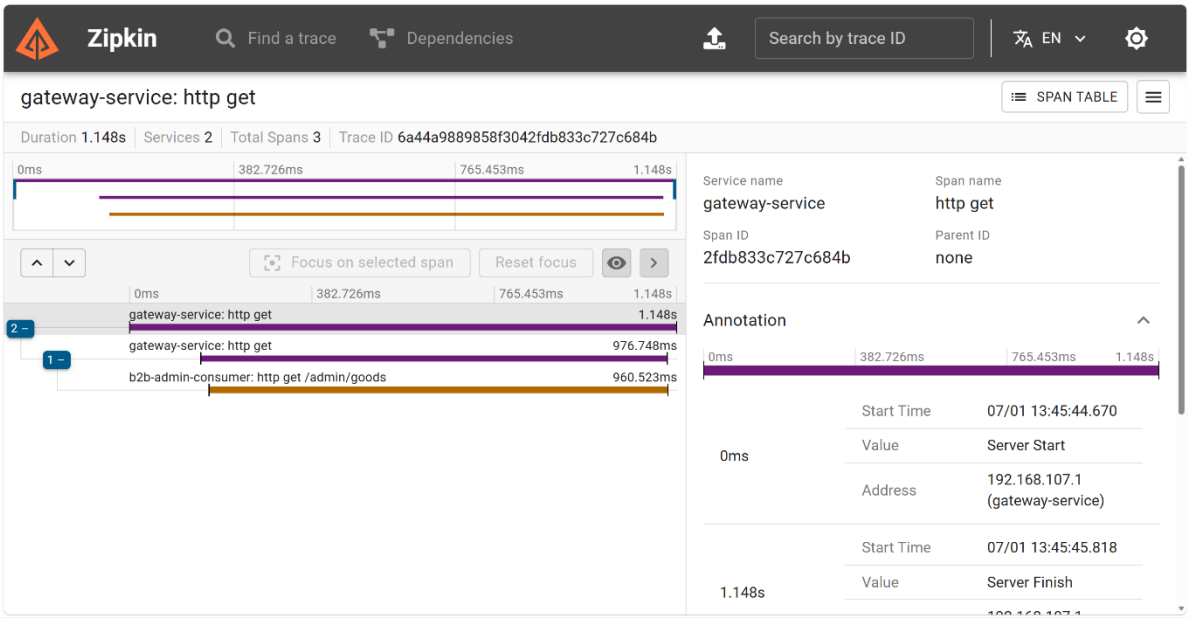
▼图21：　Spring Boot Admin 服务监控



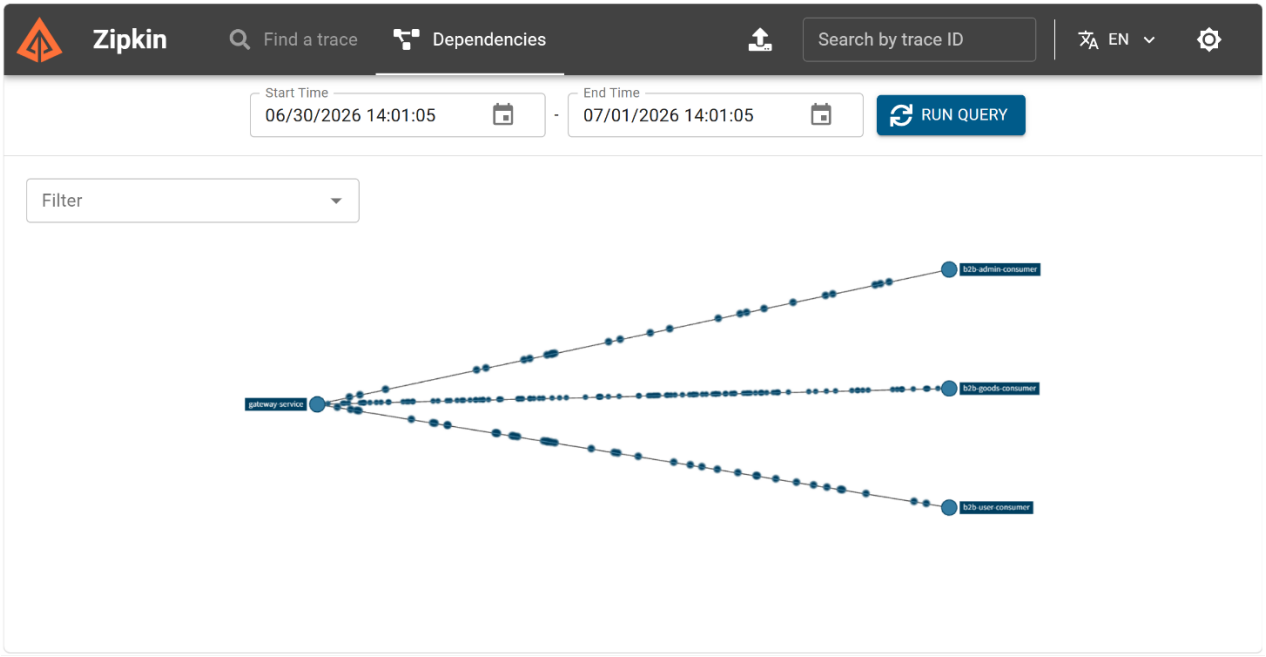
▼图22：　Zipkin 服务列表



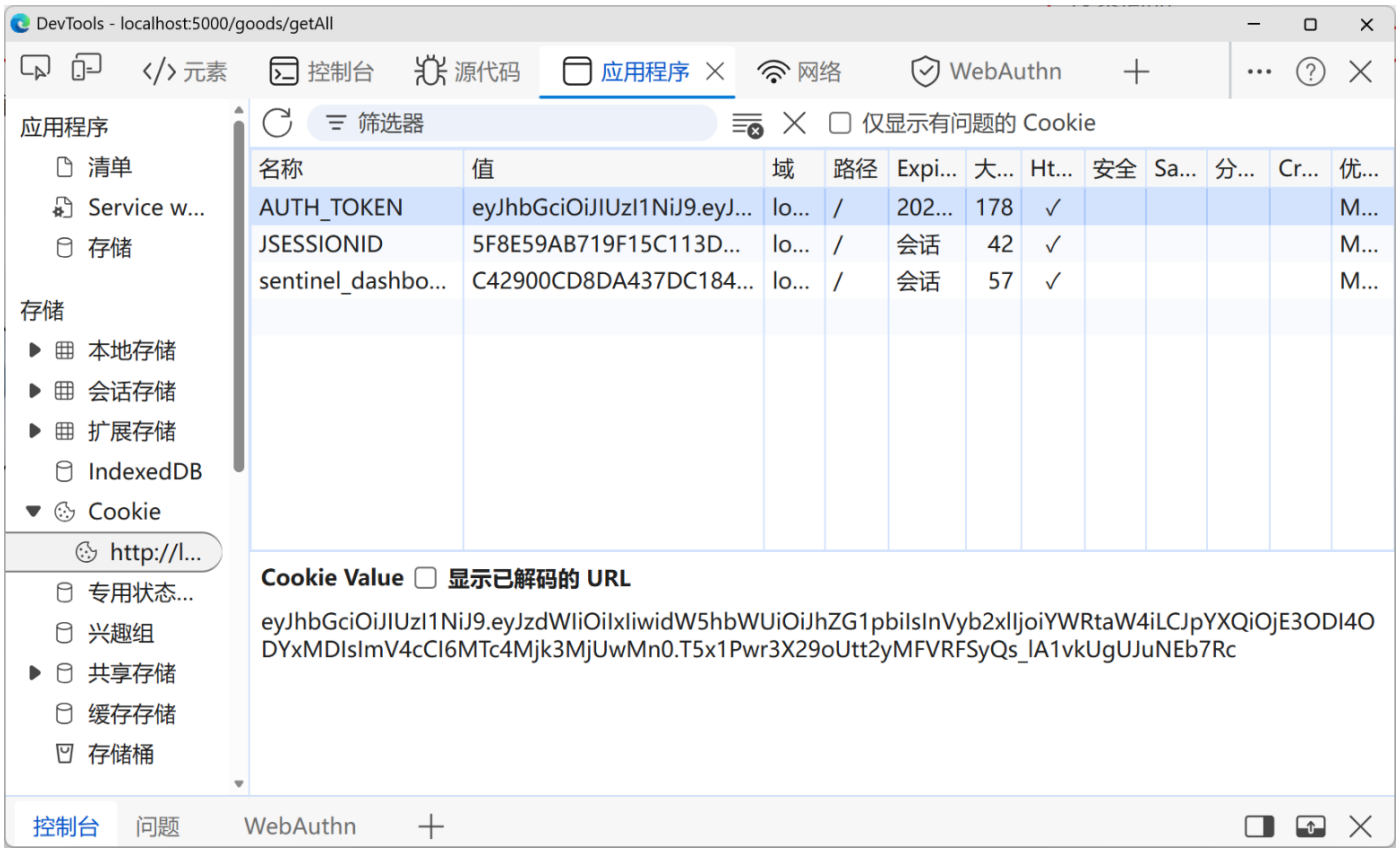
▼图23：　Zipkin 链路追踪



▼图24：　Zipkin 依赖追踪



▼图25：　　JWT 生成效果图



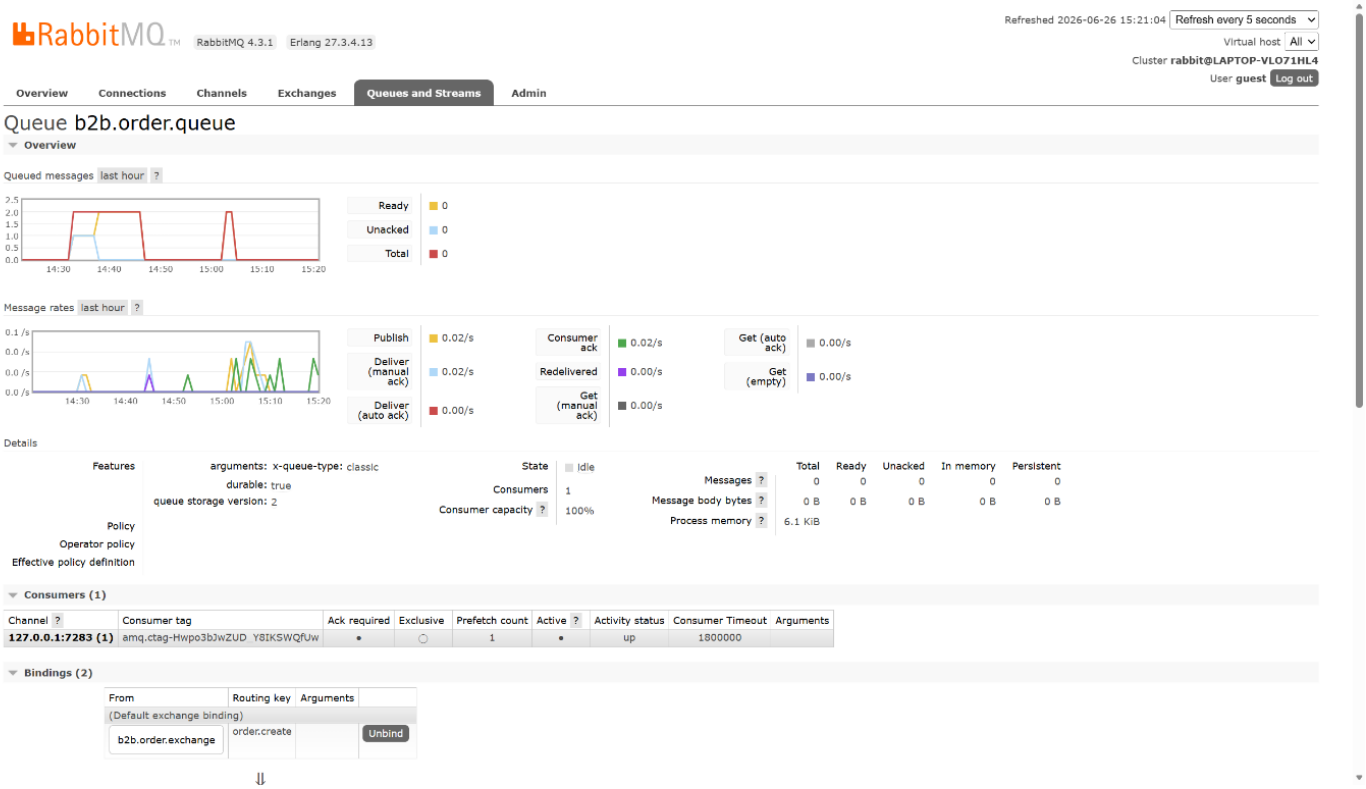
▼图26：　　JWT 实现关键代码

```java
public final class JwtUtil {
    // 全局共享密钥
    private static final SecretKey KEY = Keys.hmacShaKeyFor("spring-cloud-demo-jwt-secret-key".getBytes(StandardCharsets.UTF_8));

    private JwtUtil() {
    }

    public static String generateToken(User user) {
        Date now = new Date();
        String urole = user.getUrole() == null || user.getUrole().isBlank() ? AuthConstants.ROLE_USER : user.getUrole().trim();
        return Jwts.builder()
                .subject(String.valueOf(user.getUaccount()))
                .claim("uname", user.getUname())
                .claim("urole", urole)
                .issuedAt(now)
                .expiration(new Date(now.getTime() + AuthConstants.TOKEN_MAX_AGE_MILLIS))
                .signWith(KEY)
                .compact();
    }

    public static JwtEntity parseToken(String token) {
        Claims claims = Jwts.parser()
                .verifyWith(KEY)
                .build()
                .parseSignedClaims(token)
                .getPayload();
        JwtEntity jwtEntity = new JwtEntity();
        jwtEntity.setSubject(claims.getSubject());
        try {
            jwtEntity.setUaccount(Integer.parseInt(claims.getSubject()));
        } catch (Exception e) {
            jwtEntity.setUaccount(0);
        }
        jwtEntity.setUname(claims.get("uname", String.class));
        jwtEntity.setUrole(claims.get("urole", String.class));
        jwtEntity.setIssuedAt(claims.getIssuedAt());
        jwtEntity.setExpiration(claims.getExpiration());
        return jwtEntity;
    }
}
```

▼图27：　在 Gateway 中读取 JWT 并注入认证相关请求头

```java
// 验证并解析JWT Token
JwtEntity jwtEntity = JwtUtil.parseToken(cookie.getValue());

String uid = jwtEntity.getSubject();
String uname = jwtEntity.getUname();
String role = jwtEntity.getUrole();

// 将用户信息注入Header，传递给下游服务
ServerHttpRequest.Builder requestBuilder = sanitizedRequest.mutate()
        .header(AuthConstants.HEADER_AUTH_UID, uid)
        .header(AuthConstants.HEADER_AUTH_NAME, uname);

if (role != null && !role.isBlank()) {
    requestBuilder.header(AuthConstants.HEADER_AUTH_ROLE, role);
}
```

▼图28：　Rabbit MQ 消息队列页面



▼图29：　goods-provider（左）和 goods-consumer（右）的 Rabbit MQ 配置

```yaml
cloud:
  stream:
    bindings:
      order-create-in-0:
        destination: b2b.order.exchange
        group: b2b.order.queue
        consumer:
          max-attempts: 1
    rabbit:
      bindings:
        order-create-in-0:
          consumer:
            binding-routing-key: order.create
            queue-name-group-only: true
            requeue-rejected: true
            prefetch: 1
rabbitmq:
  host: localhost
  port: 5672
  username: guest
  password: guest
  virtual-host: /
```

```yaml
cloud:
  stream:
    bindings:
      order-create-out-0:
        destination: b2b.order.exchange
    rabbit:
      bindings:
        order-create-out-0:
          producer:
            routing-key-expression: '''order.create'''
rabbitmq:
  host: localhost
  port: 5672
  username: guest
  password: guest
  virtual-host: /
```

▼图30：　Rabbit MQ 消息生产者代码

```
private final StreamBridge streamBridge;

public OrderProducer(StreamBridge streamBridge) {
    this.streamBridge = streamBridge;
}

public void sendOrderMessage(OrderMessage message) {
    log.info("发送订单消息: {}", message);
    boolean sent = streamBridge.send("order-create-out-0", message);
    if (!sent) {
        throw new IllegalStateException("订单消息发送失败");
    }
    log.info("订单消息发送成功");
}
```

▼图31：　Rabbit MQ 消息消费者代码（包括使用事务确保不会出现半成功状态）

```
@Override
@Transactional(rollbackFor = Exception.class)
public void accept(OrderMessage msg) {
    log.info("收到订单消息: {}", msg);
    try {
        String messageId = msg.getMessageId();
        if (messageId == null || messageId.isBlank()) {
            messageId = UUID.randomUUID().toString();
            log.warn("订单消息缺少 messageId，将按非幂等消息处理: {}", msg);
        }

        int inserted = shoppingDao.insertConsumeLog(messageId);
        if (inserted == 0) {
            log.info("订单消息已处理，跳过重复消费: messageId={}", messageId);
            return;
        }

        int affected = goodsDao.updateStock(msg.getGid(), msg.getNumber());

        if (affected == 0) {
            log.warn("库存不足: gid={}, number={}", msg.getGid(), msg.getNumber());
            shoppingDao.deleteConsumeLog(messageId);
            return;
        }

        Userorder order = new Userorder();
        order.setGoodsname(msg.getGoodsname());
        order.setNumber(msg.getNumber());
        order.setPrice(msg.getPrice());
        order.setUid(msg.getUid());
        order.setTime(new java.util.Date());
        int orderInserted = shoppingDao.insertOrder(order);
        if (orderInserted == 0) {
            throw new IllegalStateException("订单写入失败");
        }

        Integer cartId = msg.getCartId();
        if (cartId != null && cartId > 0) {
            int deleted = shoppingDao.deleteCart(cartId, msg.getUid());
            if (deleted == 0) {
                throw new IllegalStateException("购物车条目不存在或已删除: cartId=" + cartId + ", uid=" + msg.getUid());
            }
        }

        log.info("订单处理成功: messageId={}, gid={}, uid={}", messageId, msg.getGid(), msg.getUid());
    } catch (Exception e) {
        log.error("订单处理失败", e);
        throw new IllegalStateException("订单处理失败", e);
    }
}
}
```