Rep\_name治理：商家名称包含很多黑名单字符，导致不能生成midcode，现在在用户新增代理时，新增用户名校验逻辑，同时存量rep\_name黑名单字符刷为空格

1. 用户新增代理时，新增校验逻辑
2. Gavin任务，**批量查询**，**限流**，过滤出包含黑名单字符的rep\_name刷新用户名

新增风控审核流程：治理审核和风控审核后，代理才算创建成功，并且要显示审核的进度

1. 用户创建代理后，会在rep表中创建一条记录，触发**DTS**，发送一条消息到MQ, 开启审核
2. 消费者中同时进行治理和风控的送审，即发送到对应的**MQ**中
3. 风控和治理审核没有先后顺序，审核结果夜通过MQ返回，自己写2个消费者去读取治理、风控的审核结果，修改rep\_status字段，显示在页面。
4. 因为可能同时修改DB的rep\_status字段，所以使用了redis的**分布式锁**

新增业务ID: 原有的rep\_id是**全局的递增id**, 现在要对外展示，为了业务安全，新设计了一个全局id,

1. 借助原有的musql号段模式递增id生成，和时间戳进行混合，保证全局递增，全局唯一
2. 新id生成加入创建代理的逻辑，同时gavin任务刷新存量数据。

消息盒子优化：店铺下统计快过期欧代、已经过期欧代数、被驳回欧代数、绑定商品数，随着业务增长，DB的查询量接近5KQPS

1. 优化：考虑到数据实时性要求不高，考虑使用**hive**表来进行优化

MySQL --定时-->Hive –-> Airbus接口--> olaf服务调用

商品绑定代理异常打标签：商品绑定的代理异常会影响商品销售，需要展示商品的异常信息

1. 根据商品绑定代理的各种异常口径使用离线hive表对商品打标签
2. 构建es供业务方快速查询。

异常细化：OlafBusinessException, OlafRPCException异常细化，方便monitor统计,**进行告警**

1. OlafBusinessException根据业务逻辑细分
2. OlafRPCException根据不同的rpc细分

慢SQL优化：一个慢查询耗时300多ms，经过优化降低到60多ms

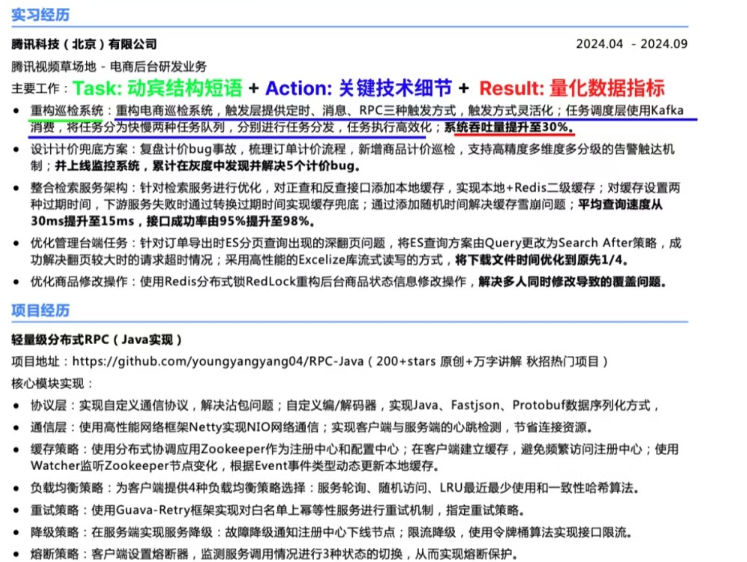
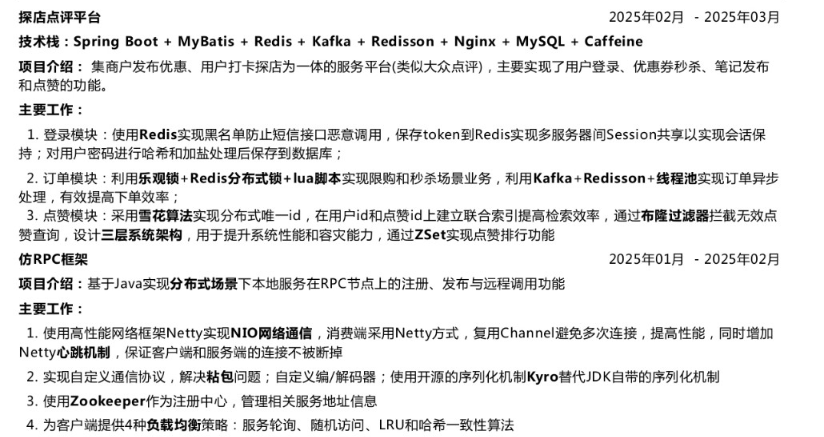
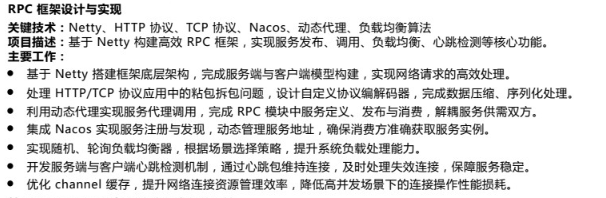
1. monitor查询到对应的SQL
2. explain查询**执行计划**，发现三个子查询，其中一个子查询rows = 24w多，扫表数据太多
3. From 里面有个子查询，然后Inner Join 里面还有一个子查询，结合业务逻辑，在From中的子查询中增加过滤条件，经过调试 子查询rows = 8w多

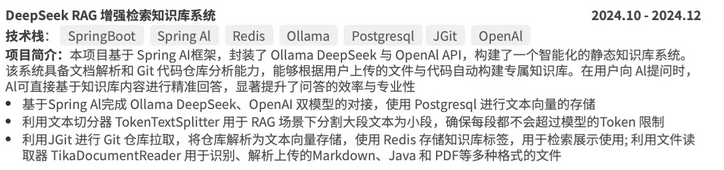
限流异常优化：从RPC异常告警中发现**限流**问题，对项目中的RPC调用处使用**ratelimiter**进行限流。

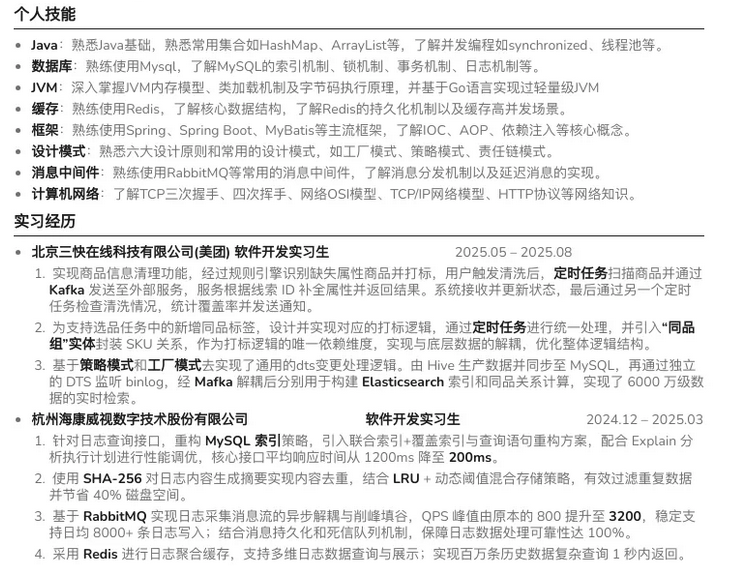
1. 调用rpc的地方其实是定时任务，查询国家的区域信息放入jvm缓存，所以瞬间的调用rpc的qps很高
2. 单机限流其实很低，5qps, 但是线上有90台机子，瞬间QPS达到阈值 100\*4
3. 使用中间件ratelimiter进行限流，单机qps限制为1，定时任务可以慢慢刷
4. Monitor观察限流QPS正常 和 rpc接口调用时间正常, 5min内刷完

RPC:

1. 使用Netty的NIO进行网络传输，简化开发，提升效率
2. 借助Netty的IdleStateHandler的写空闲，发送心跳包，实现心跳检测，读空闲关闭长连接
3. 自定义消息协议和编解码器，解决TCP的粘包问题







轻量级分布式RPC（Java实现）：仿Dubbo实现其核心功能

主要工作：

* 协议层：实现自定义通信协议、编/解码器，解决TCP沾包问题；实现kyro、Hessian、Protostuff高效序列化。
* 通信层：使用Netty简化NIO网络通信开发，并实现客户端与服务端的心跳检测，维持长连接，及时处理异常断开连接。
* 服务缓存策略：使用分布式协调应用Zookeeper作为服务注册中心；采用Channel复用技术实现并发请求，提升高并发场景性能。
* 负载均衡策略：为客户端提供4种负载均衡策略选择：随机、轮询、LRU最近最少使用和一致性哈希算法。
* 服务层：集成Spring通过注解进行注册服务与消费，运用动态代理技术屏蔽远程调用细节。