**第11章-数据埋点**

**学习目标**

项目上线之后我们想知道今天有多少客户访问那些保险产品？或者看这些产品的用户性别如何？地域分布情况如何？那如何在系统中体现呢？通过今天的学习我们采用数据埋点的方式来进行处理。来看下今天的学习目标

* 了解什么是数据埋点，数据埋点能做什么
* 掌握数据埋点的gateway实现方案并能阐述流程
* 掌握数据埋点采集的数据存储方式及实现步骤

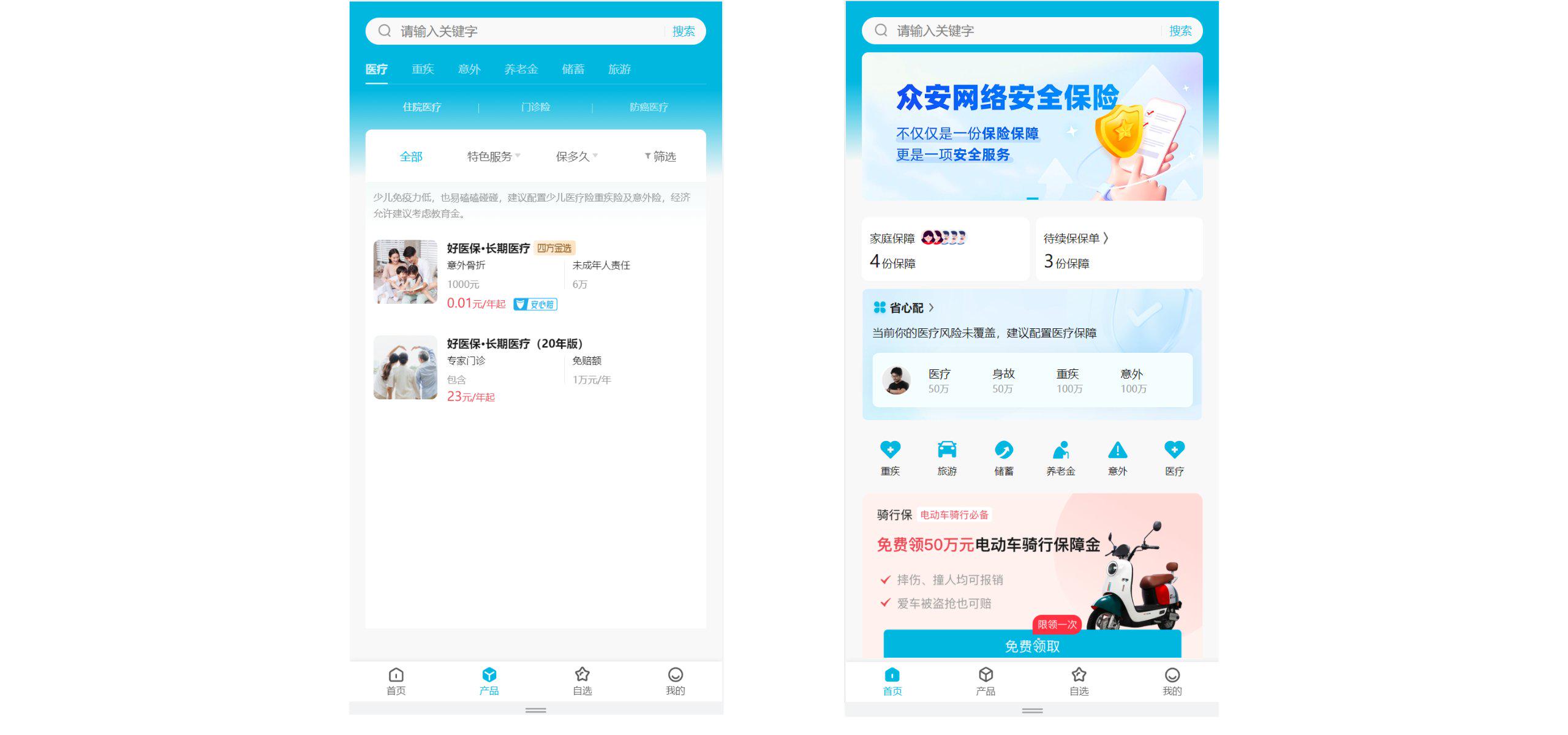
**1、埋点概念**

**1.1、埋点**

埋点 是数据采集领域的术语，指的是**针对特定用户行为或事件进行捕获、处理和发送的相关技术及其实施过程**。在此过程中收集所需信息，以跟踪用户的使用情况，最后分析数据作为后续迭代产品或者运营工作的数据支撑。

埋点就是**采集数据**的。

埋点也是为了满足快捷、高效、丰富的数据应用而做的用户行为过程及结果记录，是一种常用的**数据采集的方法**，采集的数据可以分析网站/APP的使用情况，用户行为习惯等，是建立用户画像、用户行为路径等数据产品的基础。



**为什么要 埋点 ？**

* **提高渠道转化**：通过跟踪用户的操作路径，找到用户流失的节点，比如支付转化率，分析出用户在哪个环节流失率最大，找到问题并给予优化。
* **精准客户运营：**按照一定需求对用户打标签或分组，实现精准营销、智能推荐(千人千面)等。比如根据(电商)用户浏览行为、收藏行为、加购行为、 购买行为，可用按商品等维度进行分组，推荐不同价格的商品给不同分组的用户
* **完善客户画像：**基本属性（性别、年龄、地区等），行为属性
* **数据分析：**埋点作为原料放在数据仓库中。提供渠道转化、个性推荐等；
* **改善产品：**通过用户行为分析产品是否有问题，例如用户有没有因为设计按钮过多导致用户行为无效等问题，以此发现功能设计缺陷等

**1.2、埋点设计流程**

数据埋点是数据治理流程中重要的一环，是多部门协作共同完成的工作，我们将数据埋点流程梳理如下图所示：



**领域划分**：

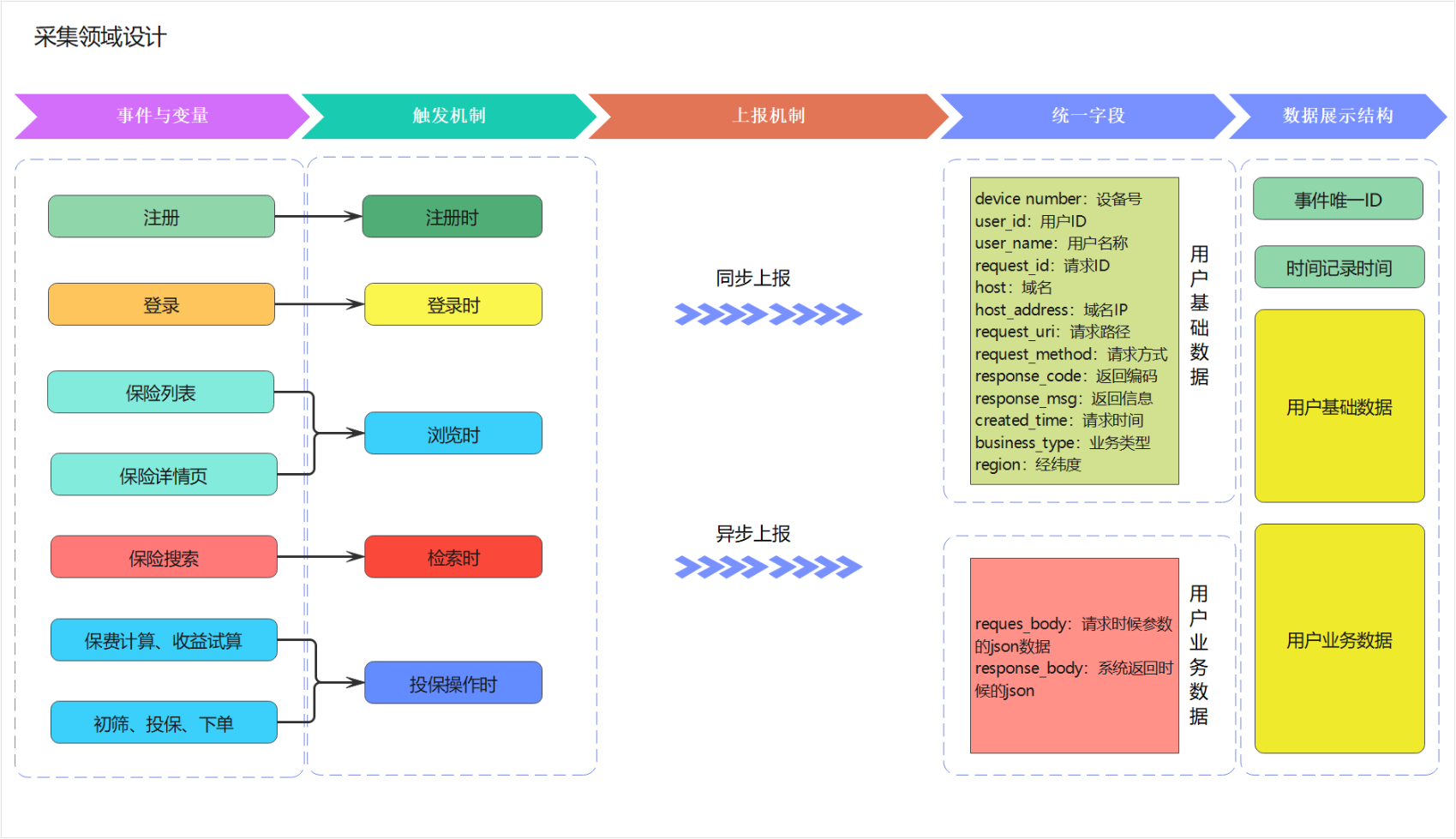
* 产品领域：产品经理对产品业务需求进行分析，架构师会评估需求，从数据指标、存储、计算引擎进行规划
* 采集领域：架构师给出埋点设计及方案实现
* 分析领域：大数据架构拿到采集的数据进行数据清洗

**最终产出：**

* **PV(page view)**：即页面浏览量，用户每次对页面访问均被记录计数；
* **UV(unique visitor)**：即独立访客，访问您网站的一台电脑客户端为一个访客，00:00-24:00内相同的客户端只被计算一次；
* **转化率**：只在一个统计周期内，完成转化行为的次数占总数的比率；
* **活跃度**：主要衡量产品的粘性，用户的稳定性以及核心用户的规模，观察产品在线的周期性变化，如日活、月活；
* **日活 (Daily Active Users, DAU)：**日活全称为每日活跃用户数量，指的是在一天之内至少使用过一次APP或访问过某个网站的独立用户数量。这个指标帮助了解产品每天的活跃用户基数，反映了产品日常的用户参与度和使用频率。
* **留存率**：在统计周期（周/月）内，每日活跃用户数在第N日仍启动该App的用户数占比的平均值。其中N通常取2、3、7、14、30，分别对应次日留存率、三日留存率、周留存率、半月留存率和月留存率。

**1.3、采集模型设计**

对于我们后端开发来说；如何设计一个采集模型对埋点的实现至关重要；接下来分析一下采集模型。



* **事件与变量**：事件一般指*产品中的功能*或者用户的操作，而变量是指描述事件的属性或者关键指标。确认事件与变量可以通过模型进行逐步拆解，理清用户生命周期和行为路径，抽象出每一个步骤的关键指标
* **触发时机**：不同的触发时机代表不同的计算口径，因此触发时机是影响数据准确的重要因素。以用户付款为例，是以用户点击付款界面作为触发条件，还是以付款成功作为触发条件进行埋点呢？二者口径不同，数据肯定会有一定差异，因此明确事件触发条件非常重要
* **上报机制**：明确数据上报机制，是实时上报还是异步上报
* **统一字段**：用两层数据表结构，第一层存放用户基信息，第二层存放用户行为信息。
* **数据结构：**统一的结构，对于数据分析师的分析有利

**2、网关埋点**

**2.1、埋点方案**

在实际的开发过程中常用的埋点方式有下列几种方案：前端页面+Nginx、业务代码埋点【硬编码】、AOP切面埋点【动态代理】、统一网关埋点【数据流】；

1、**前端页面+Nginx**：在前端页面、app界面中使用js等异步通信后端采集数据；到Nginx然后由Nginx记录不同的数据格式记录存储到一个存储空间（磁盘）再利用kafka从该存储空间读取并发送数据到大数据等环境

2、**业务代码埋点【硬编码】**：后端开发过程中Controller请求后端返回数据的过程中接入服务化的接口调用SDK里面相应的数据接口发送数据

* 优点：灵活性强，使用者可以比较方便的自定义属性、事件，传递各种所需的数据到服务端
* 缺点：侵入性强，人力成本高，每一个埋点都需要技术人员手动的添加代码；更新成本较大，每一次更新埋点方案，可能都需要改代码

3、**AOP切面埋点【动态代理】**：后端开发过程中对controller进行基于切面的信息采集

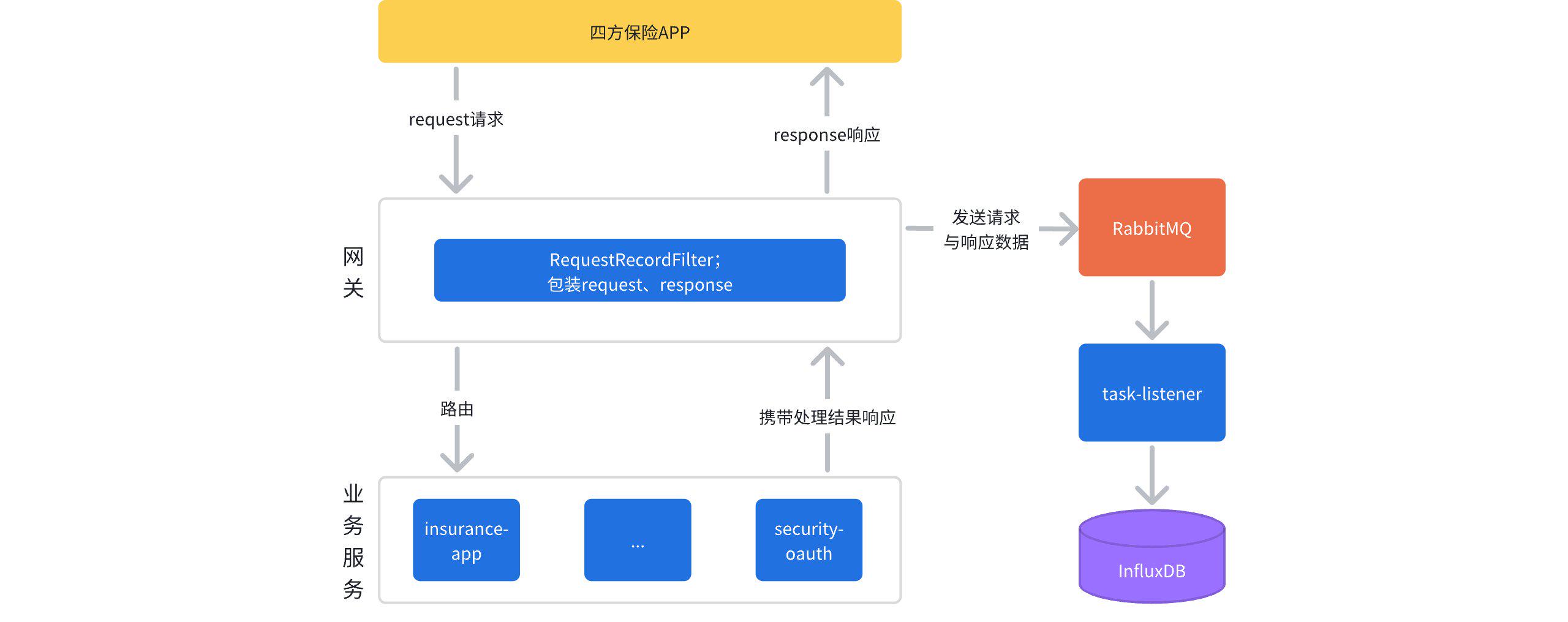
* 优点：灵活性强，使用者可以比较方便的自定义属性、事件，传递各种所需的数据到服务端，对业务层代码侵入性低
* 缺点：AOP本身是没有缺点的，但是从目前springcloud大行其道的时代，业务层controller已经不是对外的最外层结构

4、**统一网关埋点【数据流】**：网关作为数据流的统一入口和出口的最外层非常适合做数据埋点，只需要定义GlobalFilter然后包装即可

* 优点：灵活性强，使用者可以比较方便的自定义属性、事件，传递各种所需的数据到服务端，对业务层代码侵入性低，无需代理
* 缺点：对于非请求到后端的用户数据无法采集收取。

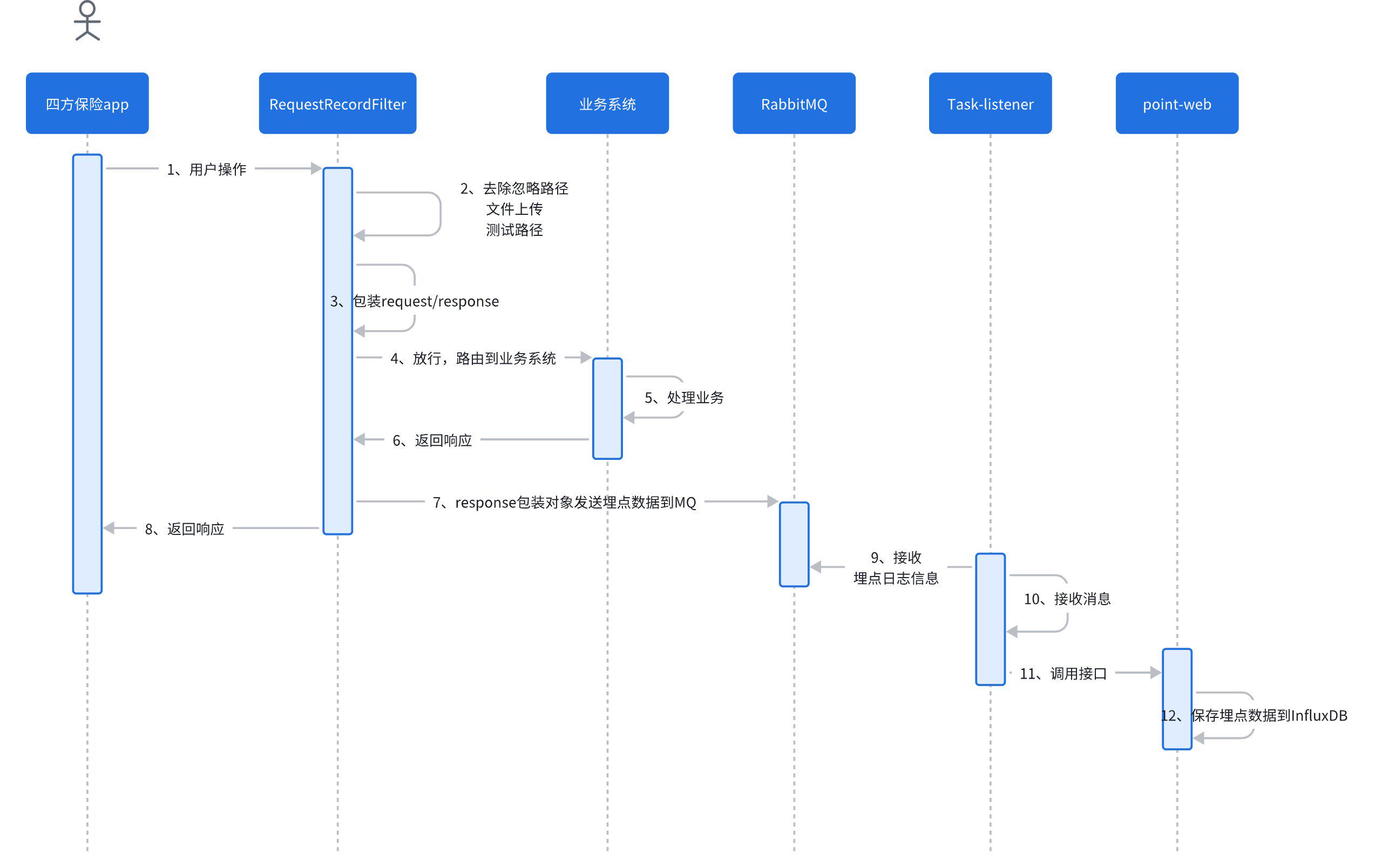
**2.2、网关埋点**

在《四方保险》中采用**统一网关埋点【数据流】**的方式，我们简单看下整个的架构设计，如下图所示：



* 首先，所有的前端的所有请求都需要经过gateway网关GlobalFilter过滤处理，这里处理2部分信息请求数据、返回数据
* 然后，通过decorator装饰器处理，请求时：获得请求参数数据，返回时：构建返回数据结构。
* 最后，CacheServerHttpResponseDecorator通过spring cloud-stream发送延迟消息存储influxDB中。

**2.3、埋点时序图**



整个的处理过程分为2个核心的内容：

**采集数据：**

* RequestRecordFilter：请求记录过滤器，用来接用户发送的业务请求，其中会去除忽略路径，测试路径，文件上传路径
* CacheServerHttpRequestDecorator：继承ServerHttpRequestDecorator装饰器，从Flux中拿到请求体，并且添加到header中
* 设置了CacheServerHttpResponseDecorator：继承ServerHttpResponseDecorator装饰器，从Flux中拿到请求body体，发送消息到rabbitMQ中

**异步数据存储**：

* RabbitMQ 接收埋点数据并远程调用接口保存数据到InfluxDB

**3、采集数据**

选定的是网关埋点；那么可以利用Spring Cloud Gateway网关进行数据的采集；上述章节也是按照过滤器的事项方案进行分析的。

采集数据则是在过滤器中将请求的信息、处理完后的结果等记录起来，存储到数据库中。

**3.1、数据库表设计**

按照分析，在网关中设置过滤器来记录每个用户源源不断的请求数据，分析不同时段用户的使用情况；这样按照时间顺序来进行分析处理的；可以使用InfluxDB存储数据（InfluxDB 1.x版本与关系型数据库类似），存储到InfluxDB中的数据要包含用户基本信息、业务操作等；具体的表结构如下：



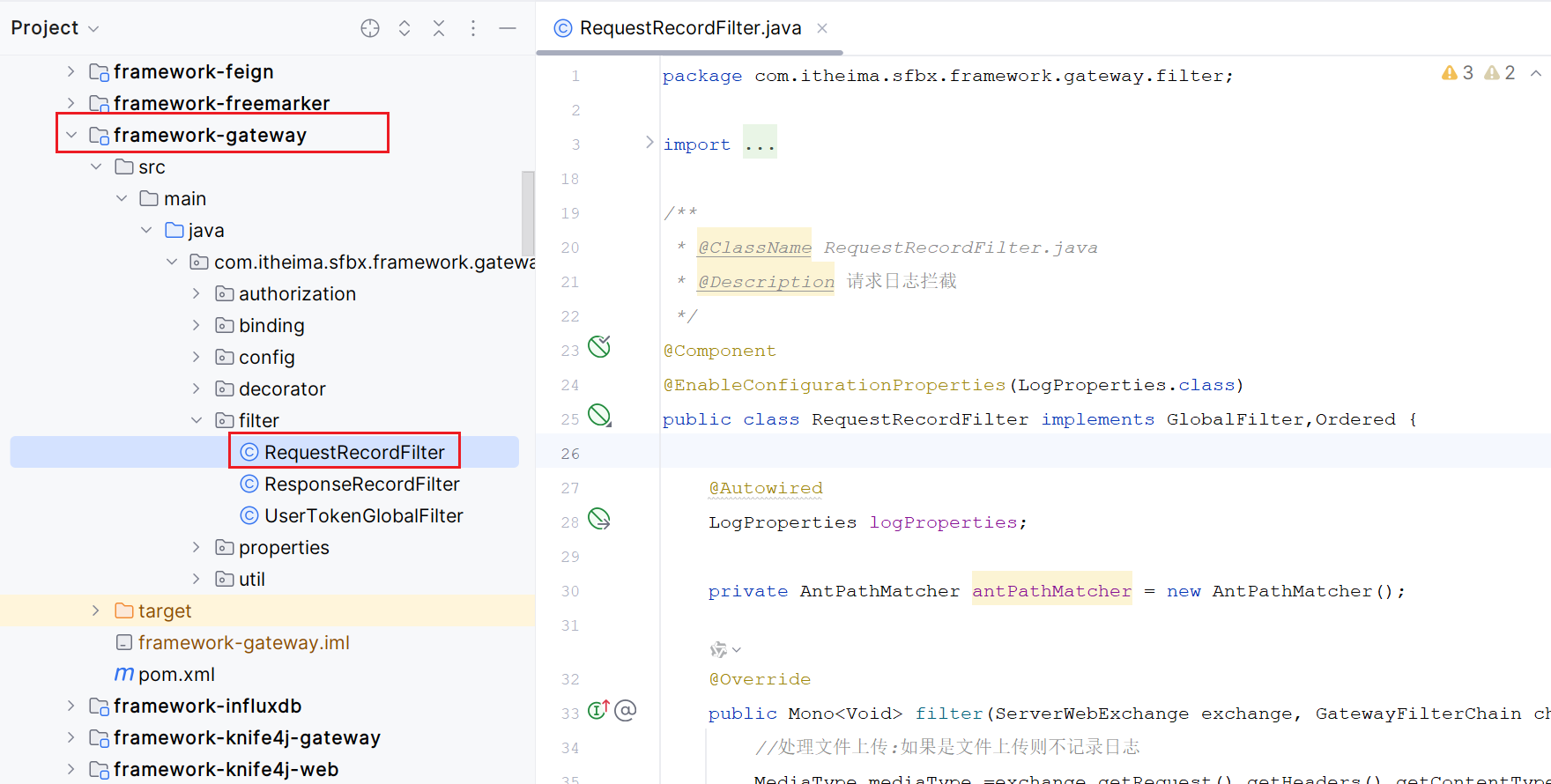
**点击图片可查看完整电子表格**

**3.2、请求数据处理**

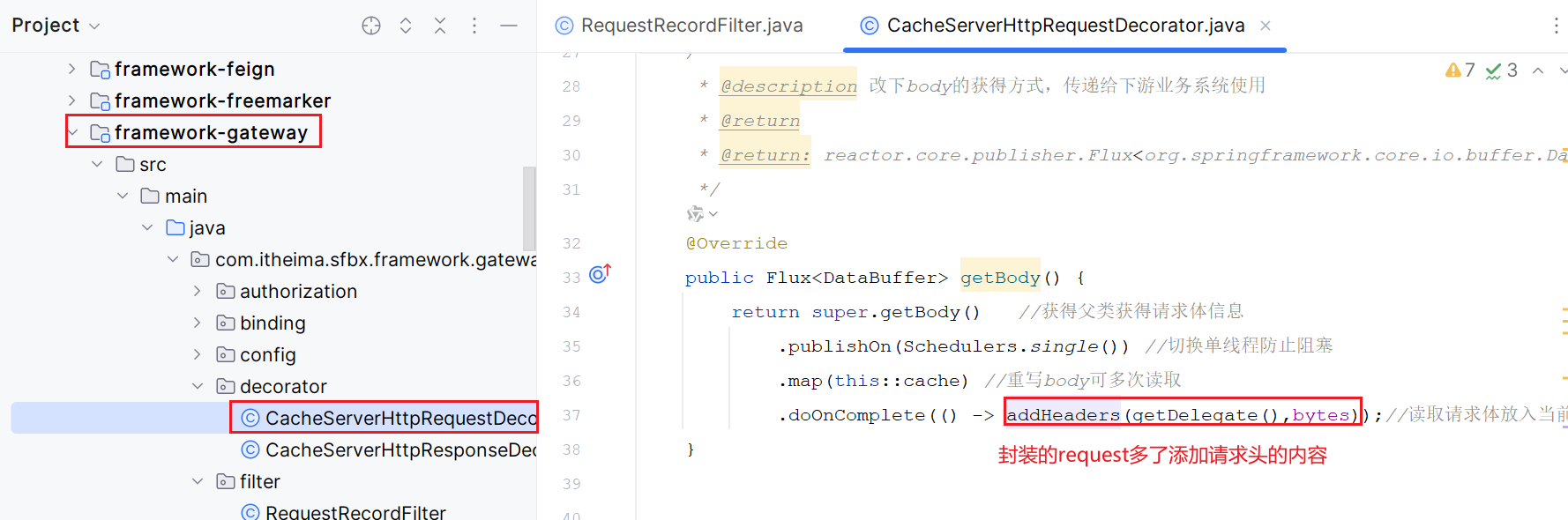
主要处理逻辑：

* 请求记录过滤器，用来接用户发送的业务请求，其中会校验文件上传路径，去除忽略路径
* 替换原有请求对象Request并设置参数；使用CacheServerHttpRequestDecorator继承ServerHttpRequestDecorator装饰器，从Flux中拿到请求体，并且添加到header中

跟进 framework-gateway 中的全局过滤器如下：

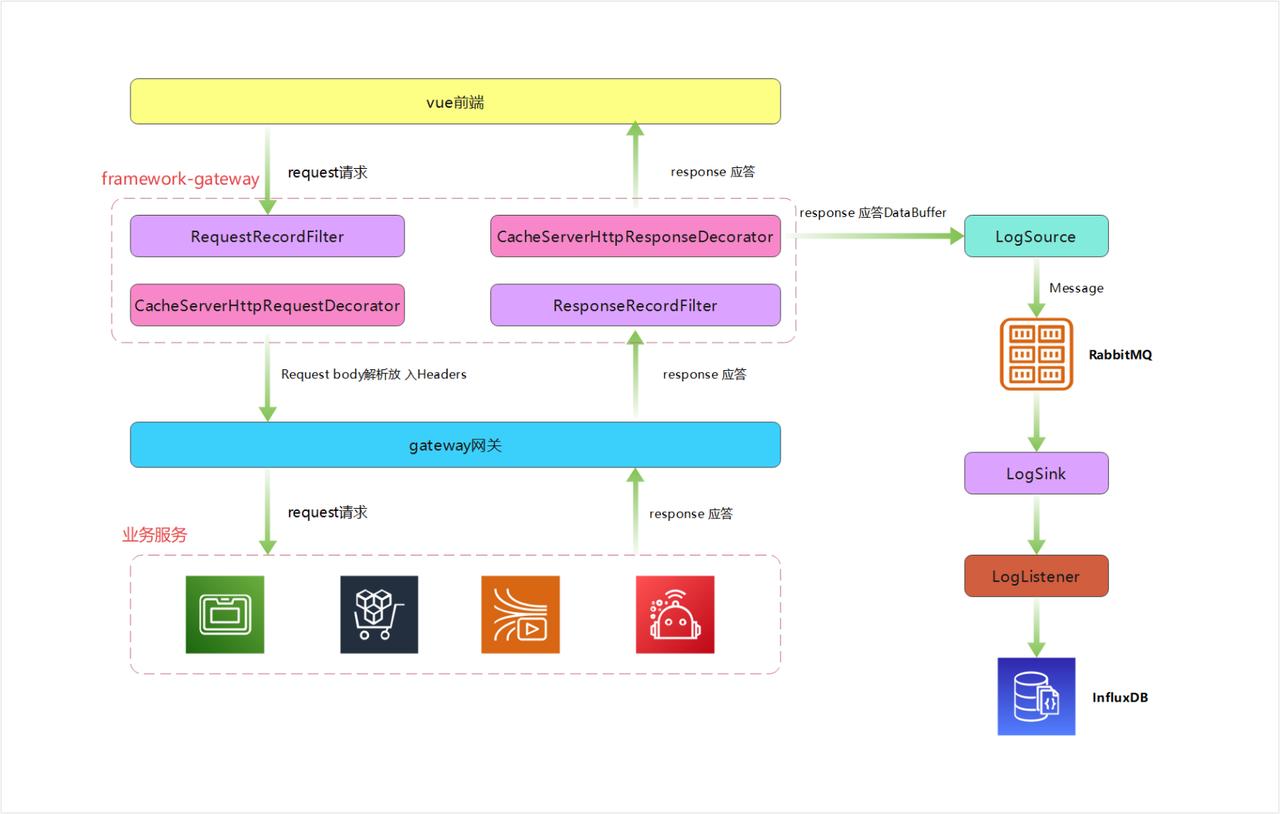


其中；重新封装的request对象中多封装了请求头，记录了当前请求的请求体内容；这样在处理完业务之后能够从响应对象中获取请求信息并发送到MQ。



**3.3、返回数据处理（采集）**

**原有的返回数据处理逻辑：**



* ResponseRecordFilter：响应记录过滤器，用来处理业务系统返回数据，其中会去除忽略路径，测试路径，文件上传路径
* CacheServerHttpResponseDecorator：继承ServerHttpResponseDecorator装饰器，从Flux中拿到请求body体，调用Logsource发送消息到rabbit中

代码如下：

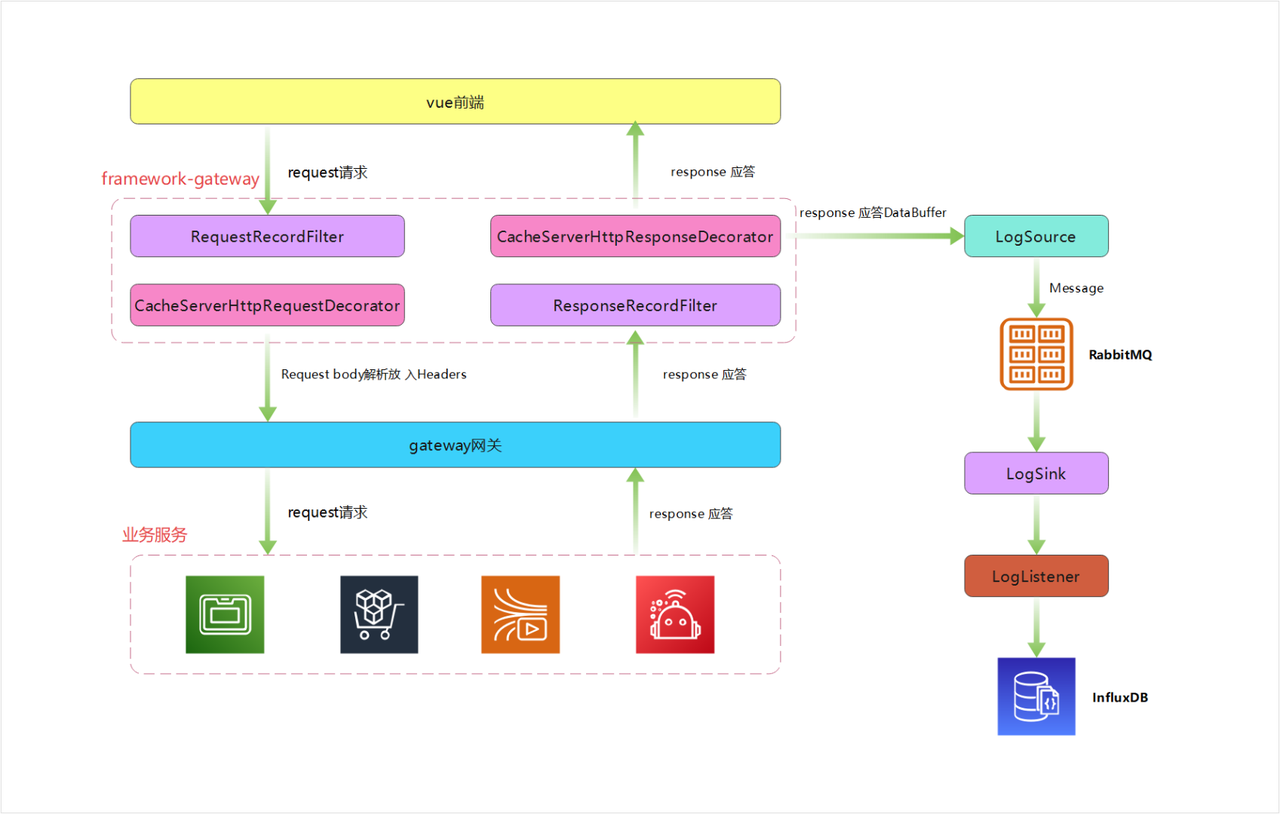


**3.4、优化返回数据处理**

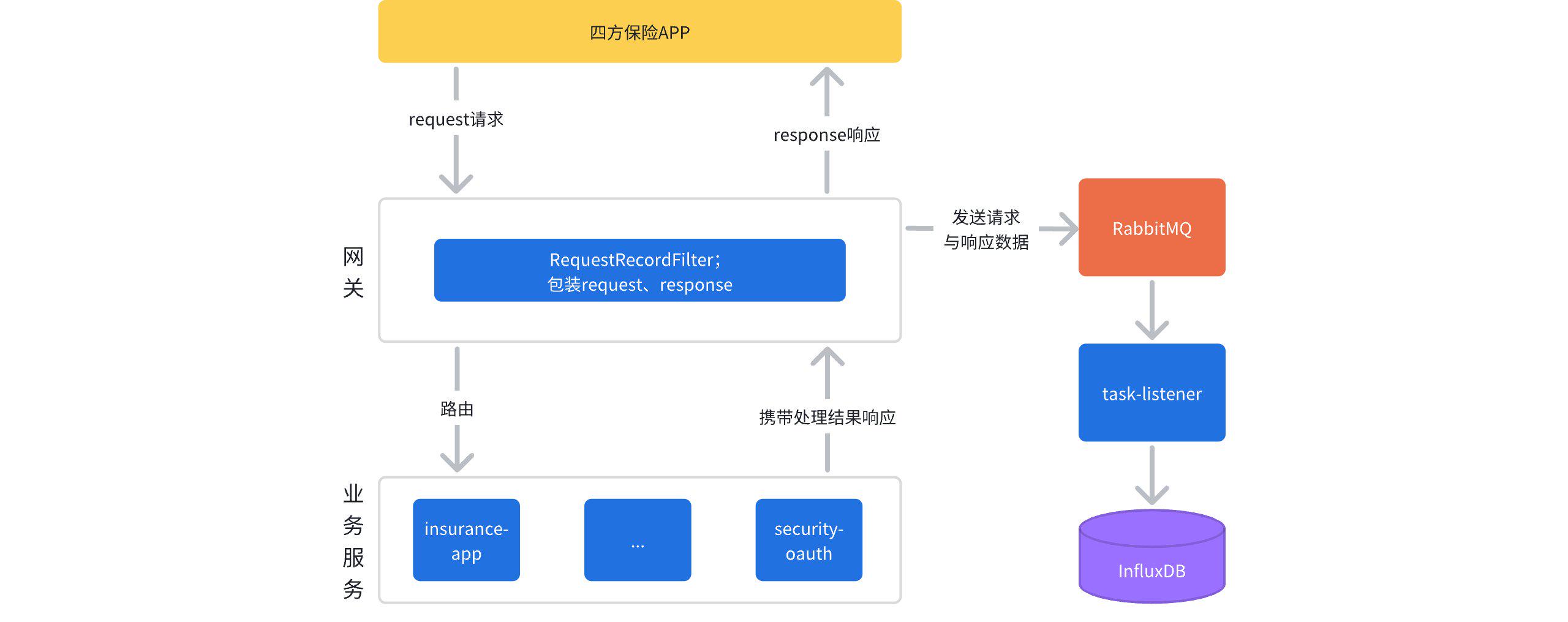
**1）分析**

上述又增加了一个全局过滤器；里面只是对响应对象进行重新封装，前面的判断都还是一样存在的？是否多此一举？能否优化？

这是目前的处理逻辑：



而我们期望的是：



问题：

* 目前写了两个过滤器，两个过滤器的目的是分别对请求、响应对象进行重新封装
* 两个过滤器的大部分代码相同；冗余了，增加了维护的成本

解决：

* 将两个过滤器合二为一；将 ResponseRecordFilter 的内容整合到 RequestRecordFilter 中。

**2）优化**

修改 sfbx-cloud\sfbx-framework\framework-gateway\src\main\java\com\itheima\sfbx\framework\gateway\filter\RequestRecordFilter.java 为如下：

|  |
| --- |
| Java @Component @EnableConfigurationProperties(LogProperties.class) public class RequestRecordFilter implements GlobalFilter,Ordered {   @Autowired  LogProperties logProperties;   @Autowired  private LogSource logSource;   @Autowired  private SnowflakeIdWorker snowflakeIdWorker;   @Value("${spring.application.name}")  private String applicationName;   @Value("${server.port}")  private String port;   private AntPathMatcher antPathMatcher = new AntPathMatcher();   @Override  public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  *//处理文件上传:如果是文件上传则不记录日志* MediaType mediaType =exchange.getRequest().getHeaders().getContentType();  boolean flag = RequestHelper.*isUploadFile*(mediaType);  *//忽略路径处理:获得请求路径然后与logProperties的路进行匹配，匹配上则不记录日志* String path = exchange.getRequest().getURI().getPath();  List<String> ignoreTestUrl = logProperties.getIgnoreUrl();  for (String testUrl : ignoreTestUrl) {  if (antPathMatcher.match(testUrl, path)){  flag = true;  break;  }  }  *//无需记录日志：直接放过请求* if (flag){  return chain.filter(exchange);  }  *//需记录日志:对ServerHttpRequest进行二次封装，解决requestBody只能读取一次的问题* CacheServerHttpRequestDecorator decorator = new CacheServerHttpRequestDecorator(exchange.getRequest());  *//需记录日志:对ServerHttpResponse进行二次封装* CacheServerHttpResponseDecorator serverHttpResponseDecorator =  new CacheServerHttpResponseDecorator(  exchange,  logSource,  snowflakeIdWorker.nextId(),  applicationName+":"+port);  *//把当前的请求体进行改变，用于传递新放入的body* return chain.filter(exchange.mutate().request(decorator).response(serverHttpResponseDecorator).build());  }   @Override  public int getOrder() {  return Ordered.*HIGHEST\_PRECEDENCE*;  }  } |

删除 ResponseRecordFilter 。

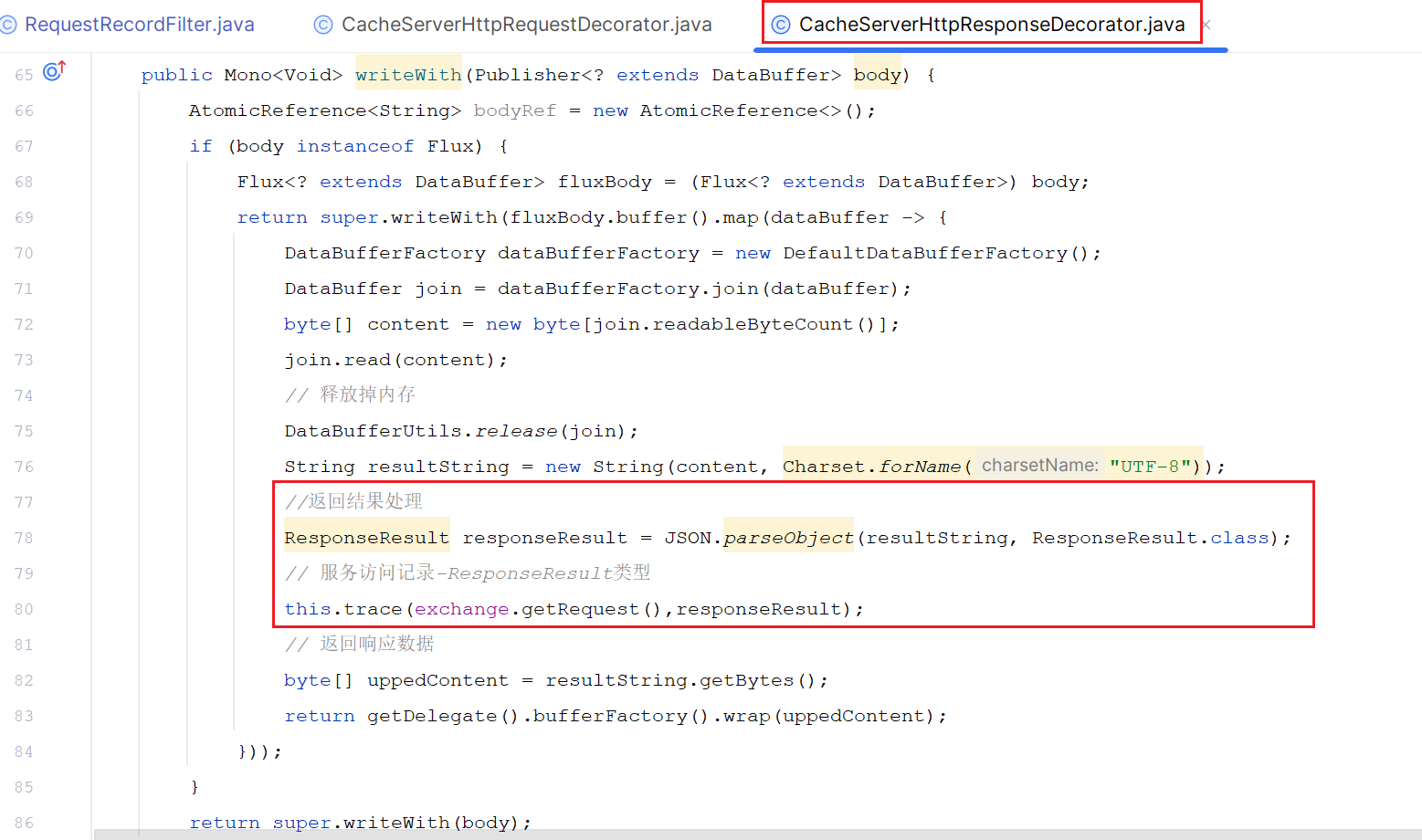
**3）测试**

重启系统；在 CacheServerHttpResponseDecorator 中打断点； 然后在app端中任何访问，查看是否能进入断点；如能说明重新封装的response是生效的。优化成功！

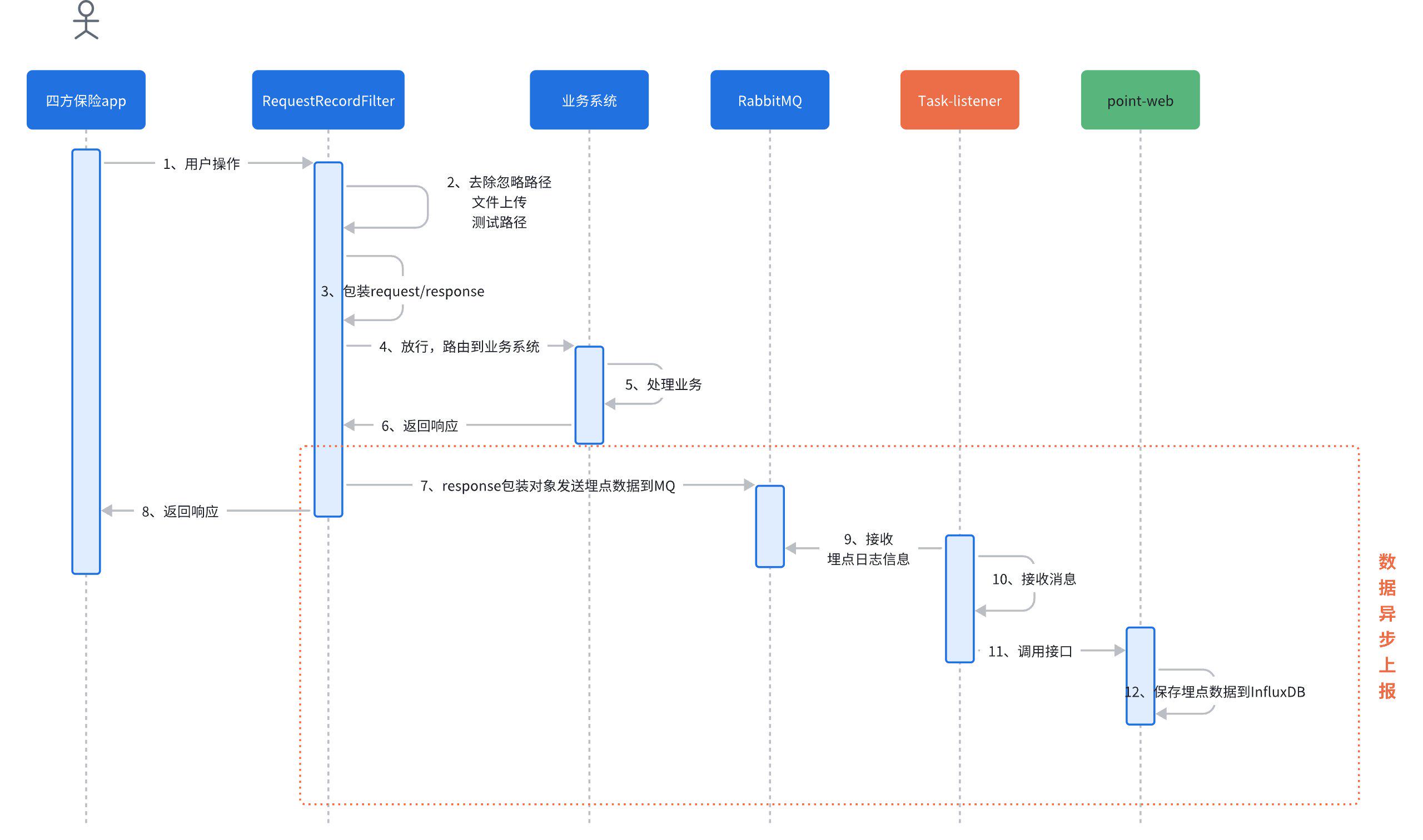
**4、数据异步上报**

**4.1、功能分析**

在上述采集数据的过程中；我们发现响应Response对象被包装为 CacheServerHttpResponseDecorator；它里面有对响应的数据进行处理的方法：



就是上述的两行代码；将采集到的数据发送到MQ，然后再接收MQ消息后存储到InfluxDB的。

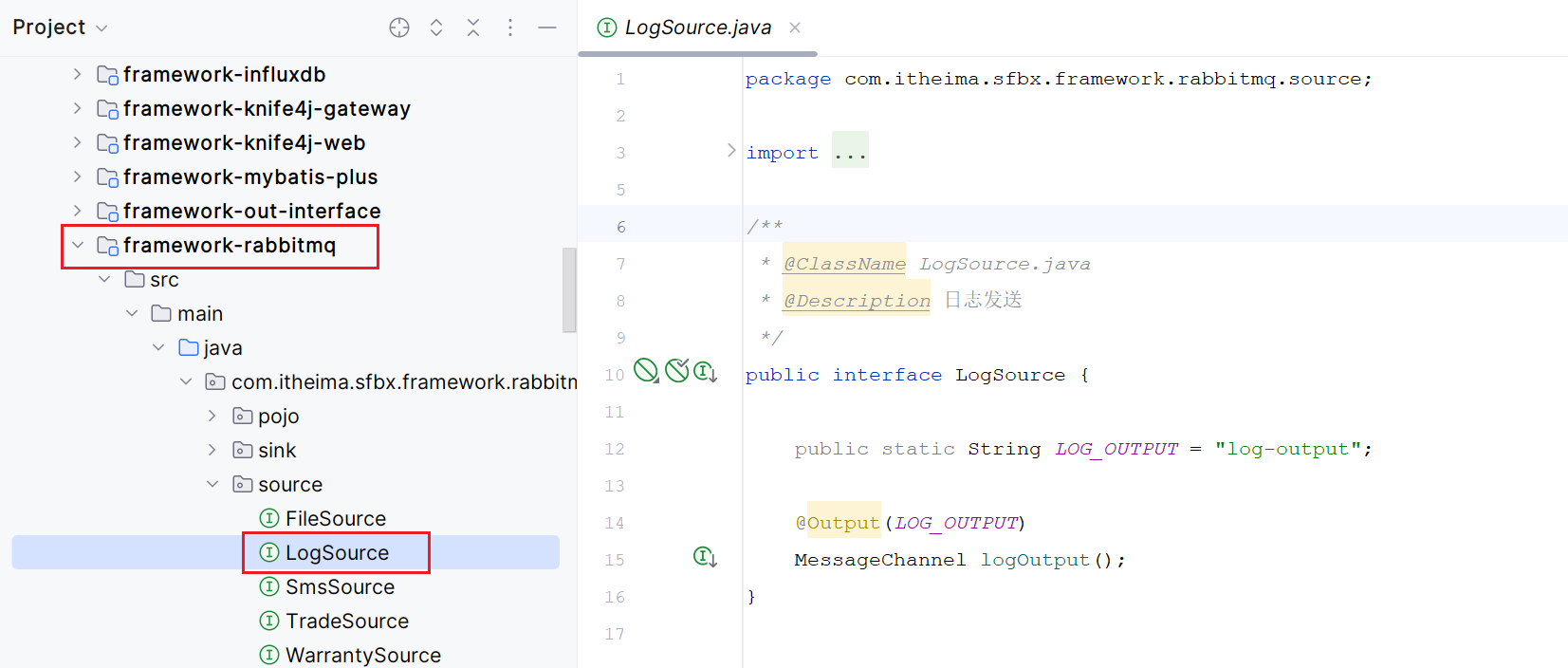


**4.2、功能实现**

**4.2.1、消息发送**

消息的发送，通过分析和上图我们能看出来是在 网关gateway-app 中进行的；那么网关中要能有SpringCloud Stream的Source即可发送。实现步骤如下：

1）在 framework-rabbitmq 模块中定义**LogSource** **类，**使用@EnableBinding让 LogSource生效

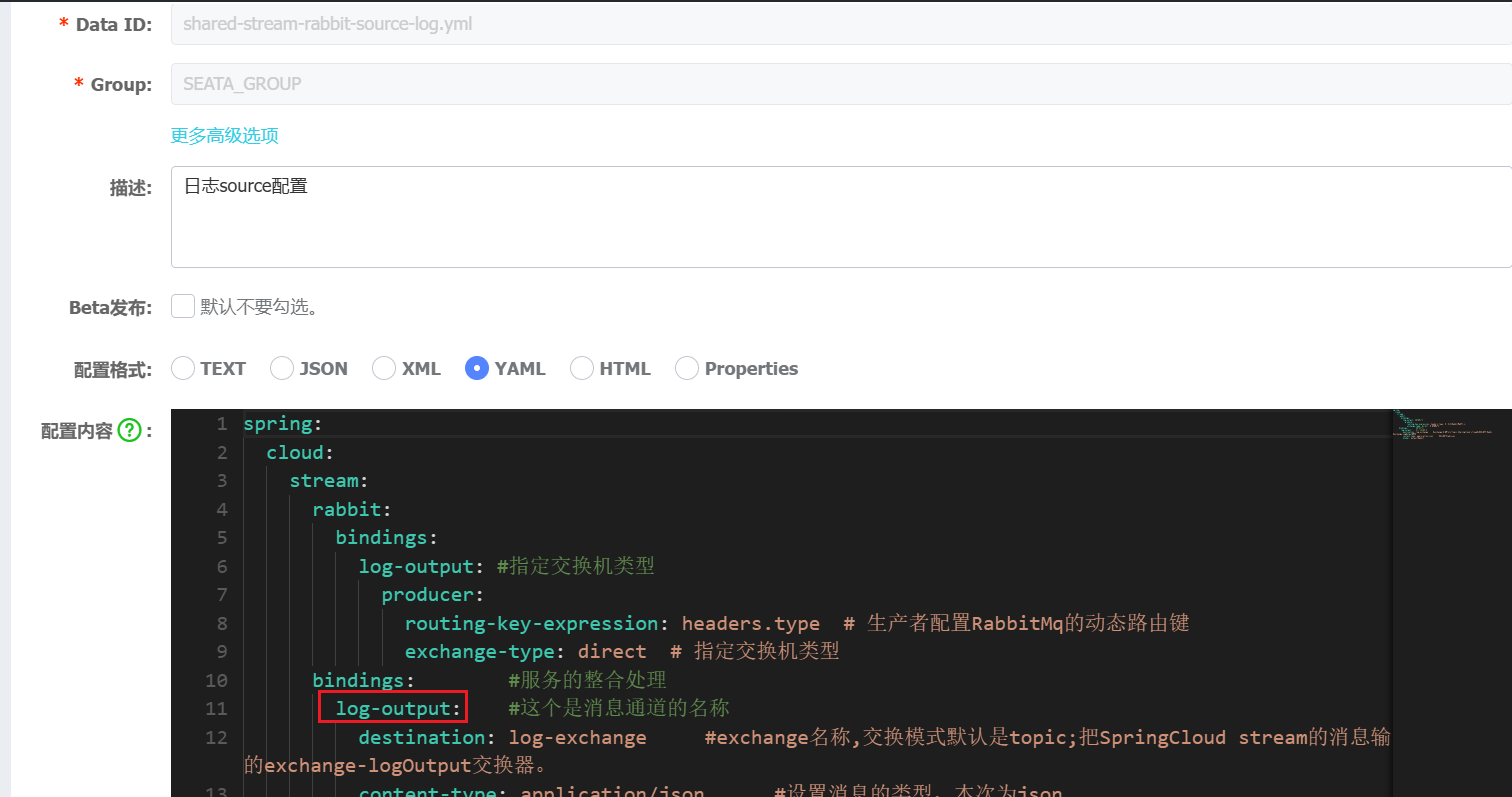




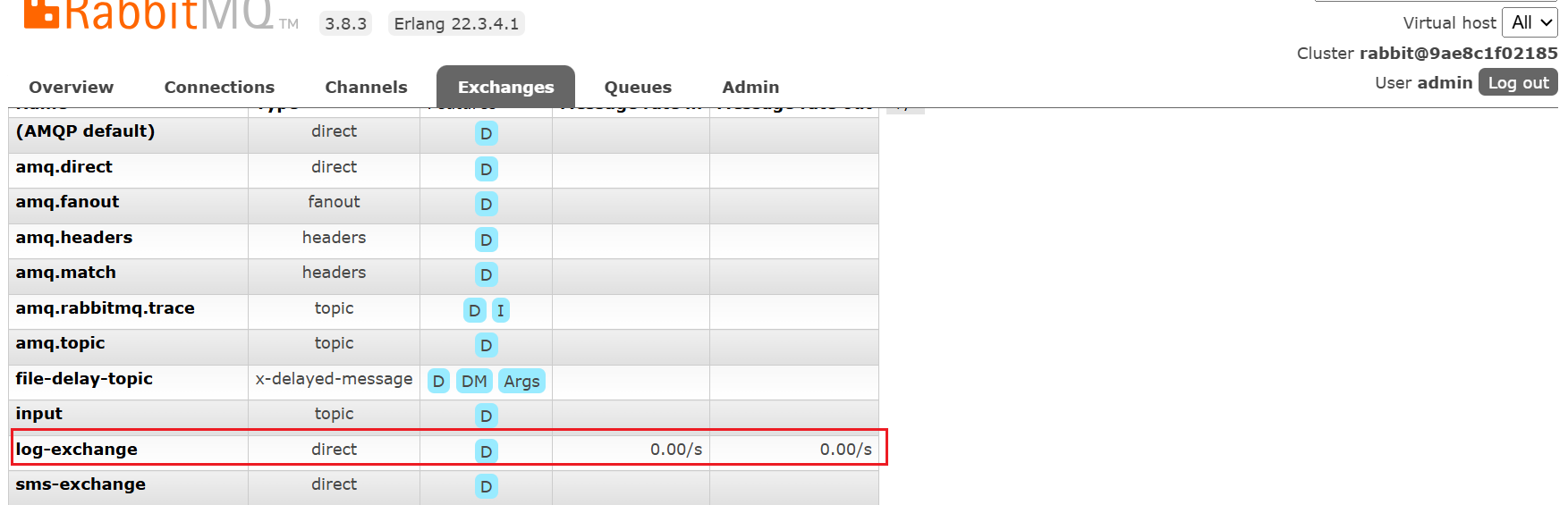
|  |
| --- |
| 因为 gateway-app 中引入了 framework-gateway；所以 framework-gateway中绑定类在 gateway-app中生效。 |

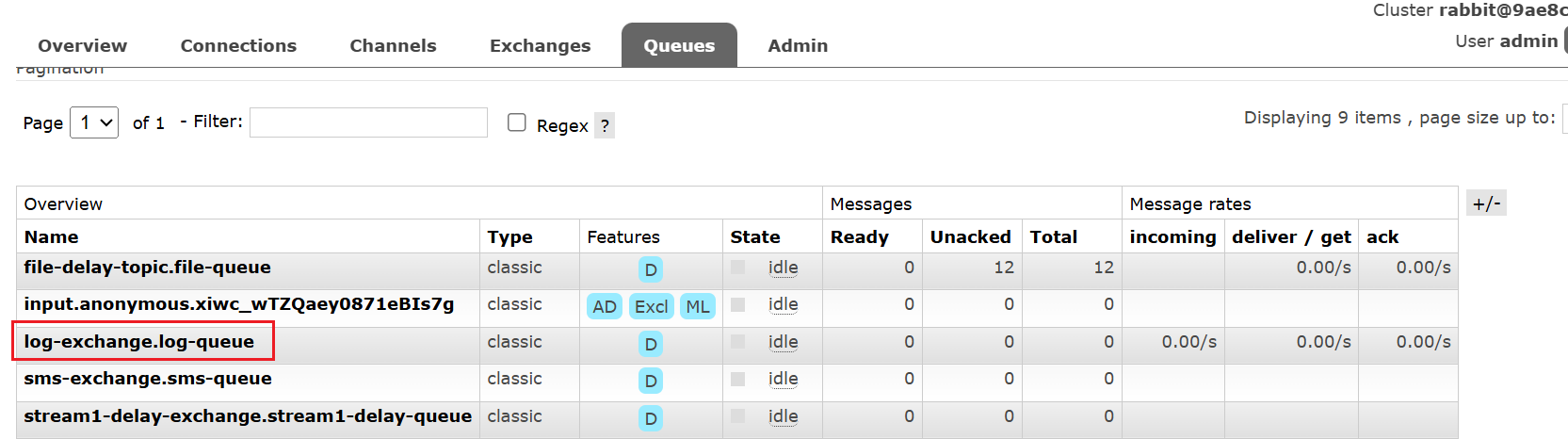
2）在nacos中的 shared-stream-rabbit-source-log.yml中定义SpringCloud Stream信息





项目启动后可以打开RabbitMQ的管理界面http://192.168.12.129:15672查看：



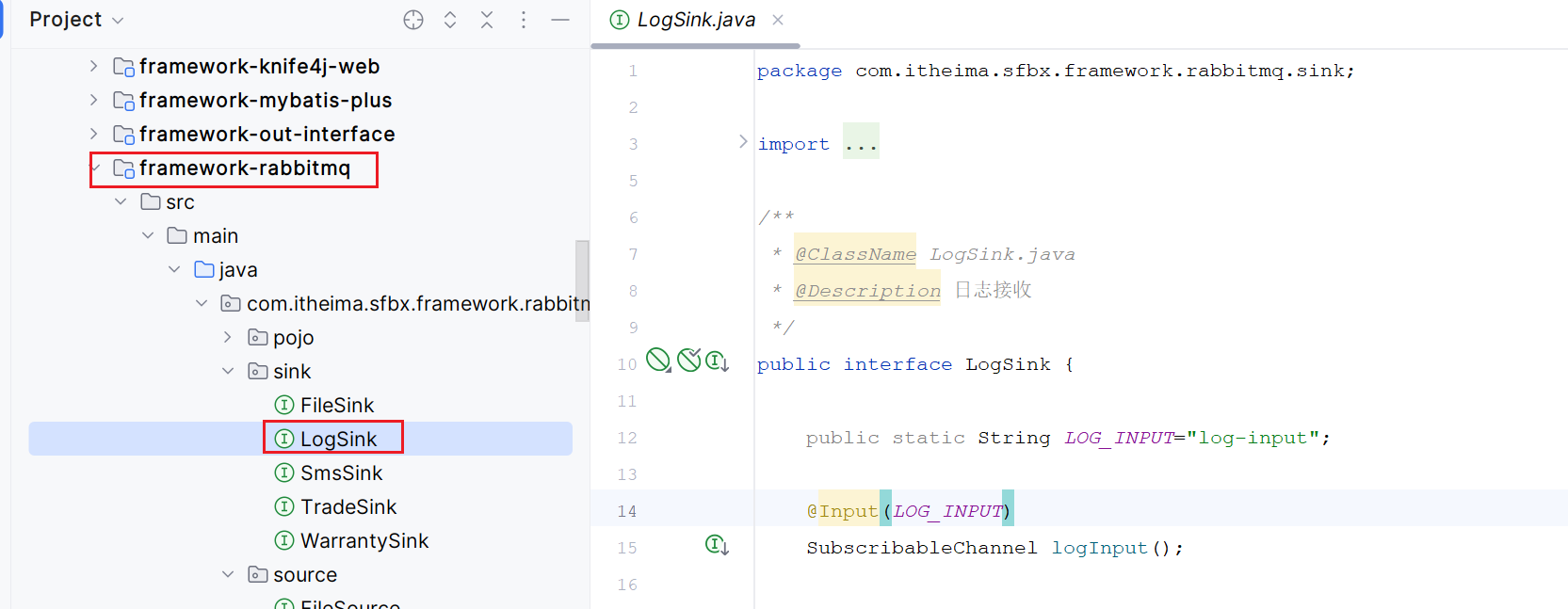


**4.2.2、消息接收**

而作为消息接收；在四方保险中有独立的模块接收来自MQ的消息，这就是 task-listener 在这里要接收MQ的消息的话；实现步骤有如下三步：

1）定义LogSink类信息，并使用@EnableBinding让FileSink生效

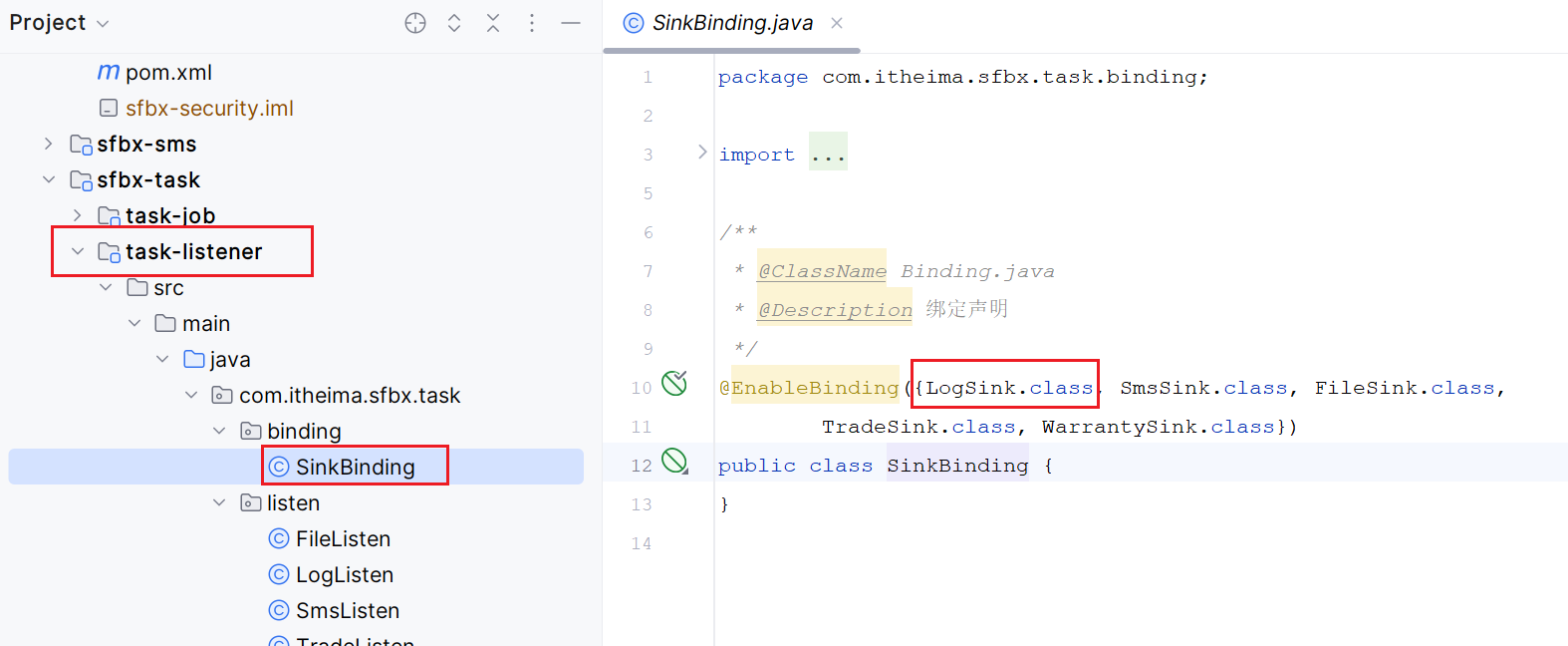
首先，在framework-rabbitmq模块中定义**LogSink类：**



然后，在task-listener模块的pom文件中导入framework-rabbitmq模块坐标：

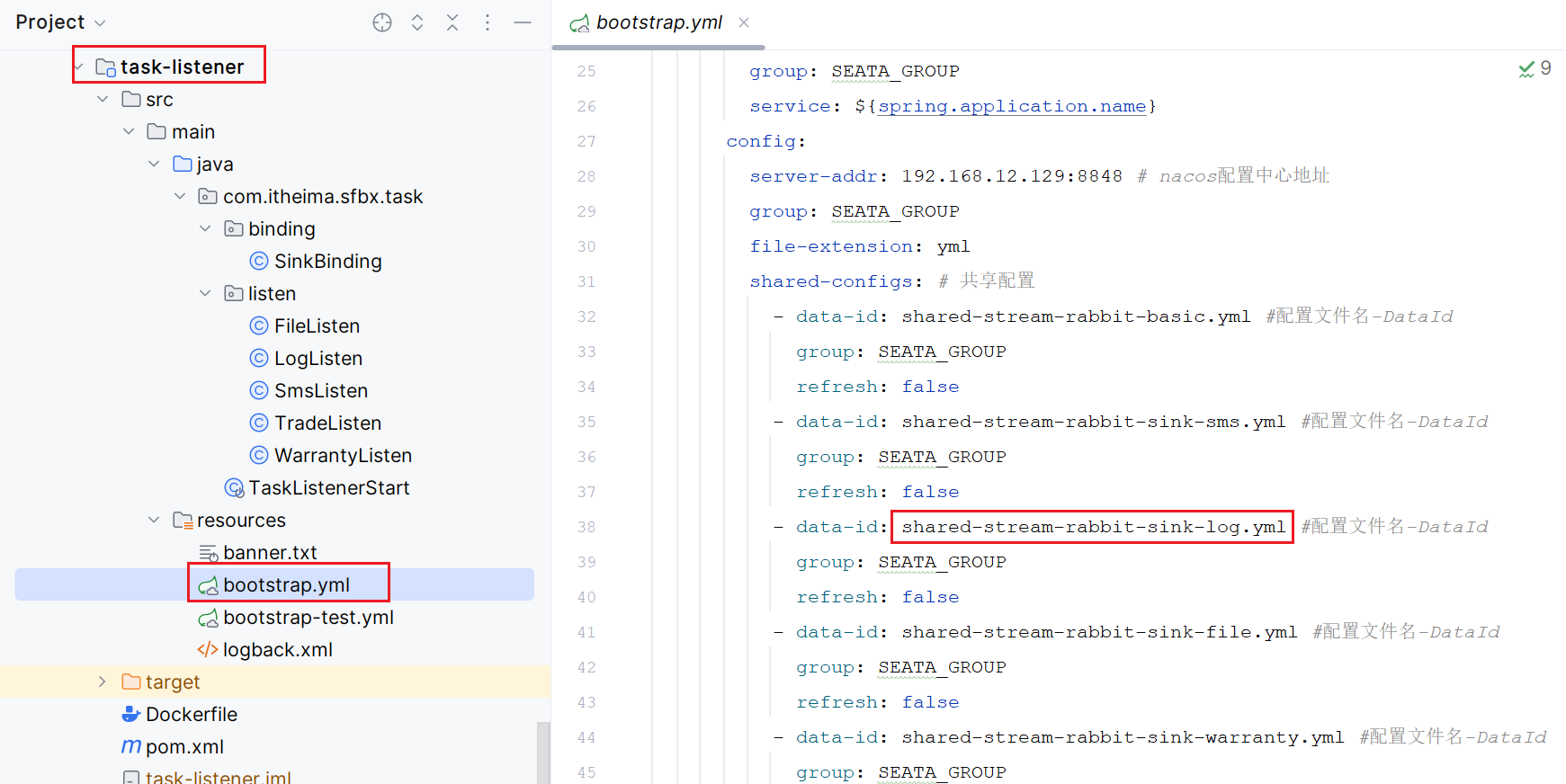
|  |
| --- |
| XML <dependency>  <groupId>com.itheima.bolee</groupId>  <artifactId>framework-rabbitmq</artifactId> </dependency> |

然后，在task-listener模块定义SinkBinding类，生效 LogSink

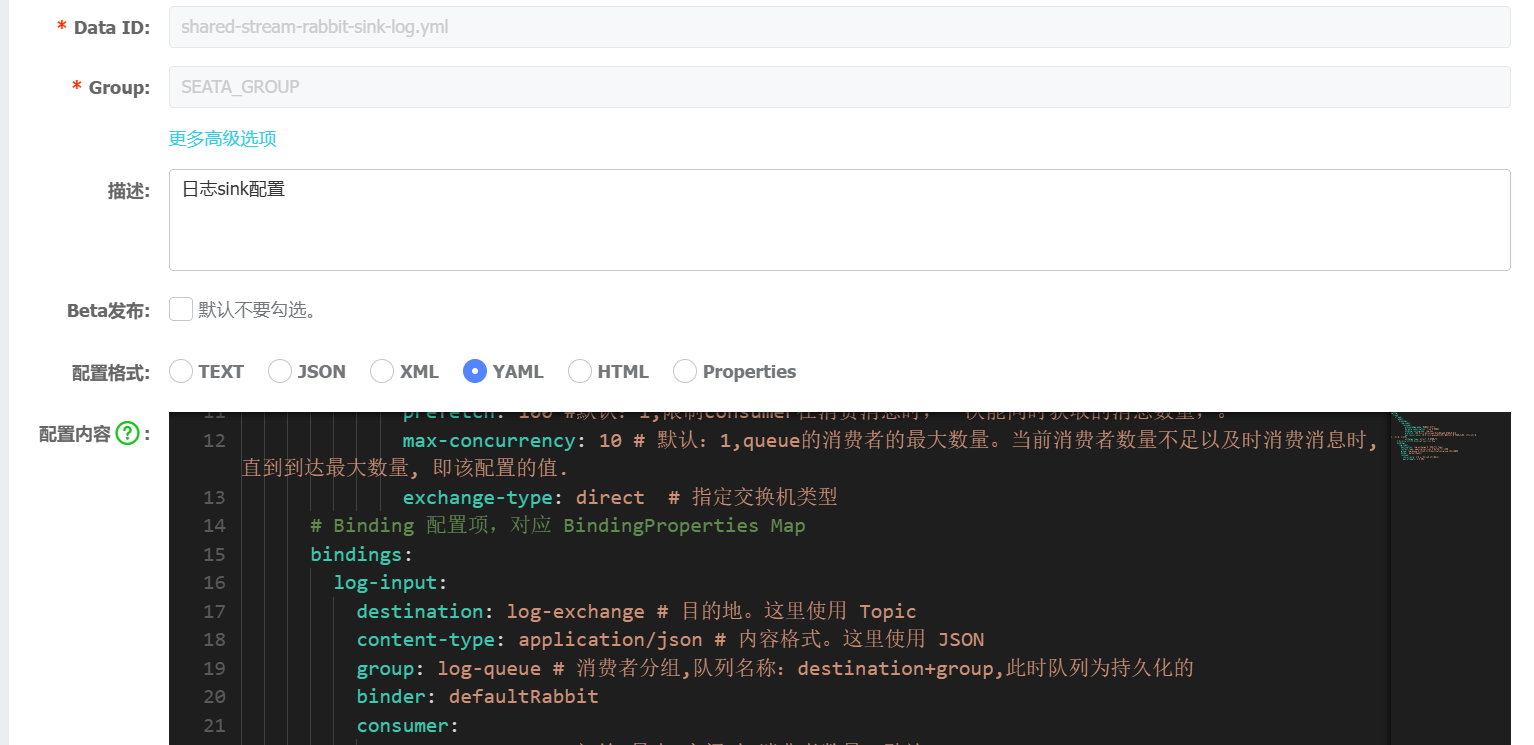


2）在nacos中 配置 shared-stream-rabbit-sink-log.yml 中定义SpringCloud Stream信息

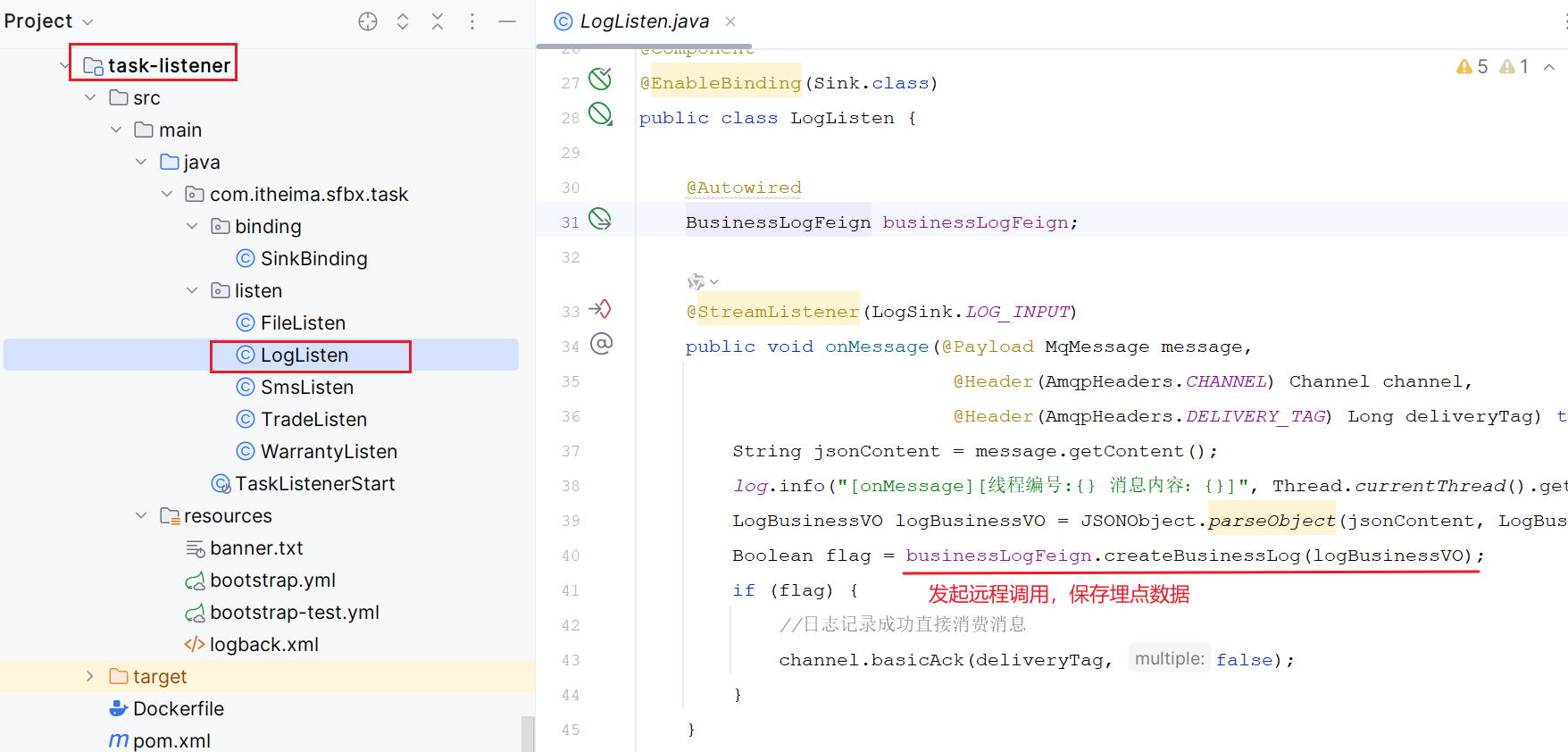
在task-listener模块的bootstrap.yml中引入关于SpringCloud Stream配置



在nacos中 shared-stream-rabbit-sink-log.yml 配置：



3）编写消息监听器



**5、面试题**

**1）埋点在哪里？怎么设计的？**

前端/app：广告区域、详情（左上角、中部）、列表；——》全部是由前端的js处理的；放置访问数据、日志到对应的路径下；再由消息组件读取并进行转换、清洗到大数据存储组件（hadoop）

后端网关gateway；我们的埋点是用户的所有行为、操作都会被采集；在网关中通过过滤器实现的。

**2）你们系统PV多少？UV呢？DAU呢？**

首先这个问题是偏运维的，如果不好回或者想不起来，完全可以答不清楚，开发完就到下一个项目组了所以不知道。

* **PV(page view)**：即页面浏览量，用户每次对页面访问均被记录计数；

我们系统上线一段时间，不过听运维说，PV并不高，每秒页面浏览量最高大概在100以内

* **UV(unique visitor)**：即独立访客，访问您网站的一台电脑客户端为一个访客，00:00-24:00内相同的客户端只被计算一次；

系统推广不多，有一次线上维护过所以查看了一下用户量大概在3万，日UV的话大概 300

* **日活 (Daily Active Users, DAU)：**日活全称为每日活跃用户数量，指的是在一天之内至少使用过一次APP或访问过某个网站的独立用户数量。这个指标帮助了解产品每天的活跃用户基数，反映了产品日常的用户参与度和使用频率。

与 日UV一样意思，要说具体数量也是300

**3）了解前端埋点吗？怎么做**

前端埋点就是在界面中用户点击的地方如果也需要记录这次点击；那么就算一个埋点。

比如：手机的左上角用户比较喜欢点击，所以看起来是白色的空白块；但是有一个隐形的html元素在，点击就会发送一个 ajax 异步请求到nginx或者到后端记起来；再统计后发现规律，可能给这个区域投放什么类型的广告等

基本上所有的网站、app都会存在前端埋点；也就是能看见图文的区域或者没有图文的区块都会根据产品经理要求设置埋点。

怎么做？每个埋点都对应一个ajax请求，点击了就发送到nginx；nginx数据存入到一个日志文件；再由kafka定时读取清洗发送到大数据团队给定的接口。大数据团队分析、再清洗、统计之后存入到mysql，我们后端再从mysql中读出数据展现在我们的管理后台。领导们看到这些统计信息再对网站、app的运营做出决策。

**4）安全登录问题**

在用户登录系统的时候；前端输入了账号和明文密码到后端；如果这时候被人拦截了；那么密码就被人知道了；怎么办？不够安全，如何解决？

* 在用户注册的时候；对他的密码；加密保存（md5）
* 登录的时候；用户在前端输入的密码在传输到后端的时候；也使用md5加密
* 登录校验的时候；就是使用加密的字符串与数据库中加密的密码字符串进行对比；一致则登录成功生成token

**5）生成一个分布式ID的方案**

* 雪花算法（工作id、序列id，+ 时间戳（纳秒） + 随机字符串）
* UUID（非分布式）
* Redis 可以使用自增key作为id

**6）简历职责**

**简历职责-候选**：实现并开发项目中app端用户行为数据的采集及统计系统的PV、UV、DAU/WAU/MAU，留存率、转化率

* 从注册用户转化为购买保险
* 从浏览保险产品到支付