**第8章-投保处理**

**学习目标**

* 掌握并阐述用户投保的整个流程步骤
* 生成保险合同时我们为什么要做快照处理
* 对于下单未后续处理的保险合同如何处理

**1、生成合同**

**1.1、需求分析**

在app端；查看任何一个产品的时候，在产品详情页面中，点击 下一步 之后就是要生成合同；界面如下：



在点击完 下一步 进行生成合同之后会需要再次确认投保的信息如下（保单生成之后，可以在 我的——我的保单 中查看 ）：



而今天主要完成的就是 用户投保；也就是点击下一步做什么。保险行业属于专业性比较强的行业；它的业务、开发都有一定的流程，这些产品、项目经理会描述清楚。所以在今天的 用户投保 中；具体的流程一般需要结合需求、行业经验而分析、设计和开发。具体的；我们先来看它整个投保流程：

**1.1.1、投保流程**



整个流程如下，我们需要重点完成的为红色部分：

**保险投保路由【已完成】：**

* 四方保险项目启动时，InsureHandler投保处理接口的多种处理接口实现如：医疗、重疾、意外、养老等会被Spring托管放入IOC容器。
* 根据前端传递的标识符从IOC中获得具体的保处理接口的某个实现类去处理具体的业务

**保险合同生成：**

* 重新计算保费或理财收益，不能直接用前台传递金额，用以保证合同安全**【已完成】**
* 构建合同信息，形成合同快照信息包括如下：
* 用户信息：包含投保人和被保人的完整信息
* 保费信息：包含保费、缴费方式、缴费期限
* 保障信息：投保时所选方案对应的保障项信息，以及保障期限的起始
* 协议条款：保险附件相关信息

**保司承保：**

* 携带保险合同信息调用保司提供的承保接口

**合同订单生成：**

* 合同生成后，我们按合同需要支付的内容生成合同订单（快照），支付订单（快照）是支付的依据凭证

保存合同被保人**：**

* 将本次被投保的人保存到对应的合同被保人表中

**延迟合同取消：**

* 完成保险合同、合同订单的生成完成后，如果用户10分钟内未进行支付，需要取消合同和订单

**1.1.2、保单合同组成**

在app端中 我的 —— 我的保单 可以查看具体的保单信息，了解到保单合同的组成：



用户下单后需要形成对应的合同信息，具体的组成部分由4个部分组成：

* 用户信息：包含投保人和被保人的完整信息
* 保费信息：包含保费、缴费方式、缴费期限
* 保障信息：投保时所选方案对应的保障项信息，以及保障期限的起始
* 协议条款：保险附件相关信息

**1.1.3、快照处理**

保险产品会随着时间和保险行业情况，保司对这些产品有所调整；但是在用户投保后；用户投保的保险产品，不管过了多久；在生成合同之后 保险产品 内容应该是与当时查看的保持一致，不会改变的。

这样的话；就需要将当时查看保险产品信息做完整的信息冗余存储到保单合同中，这样不论保险产品如何改变也不影响用户签署下单的保险产品内容。而这种冗余存储就称为快照。

**具体开发思路是**：

用户投保时，我们会根据所选的保险产品信息内容，转换为对应的保险产品（及其对应的数据）JSON，作为保险合同表的一个属性进行存储，这样操作可以保存投保下单时保险产品的当时的具体信息且不会因保险产品的变化而引发保险合同的变化，达到合同快照的效果，从而使得保险合同具有投保凭证的效果。

**1.2、数据库表设计**

**1.2.1、概念模型**

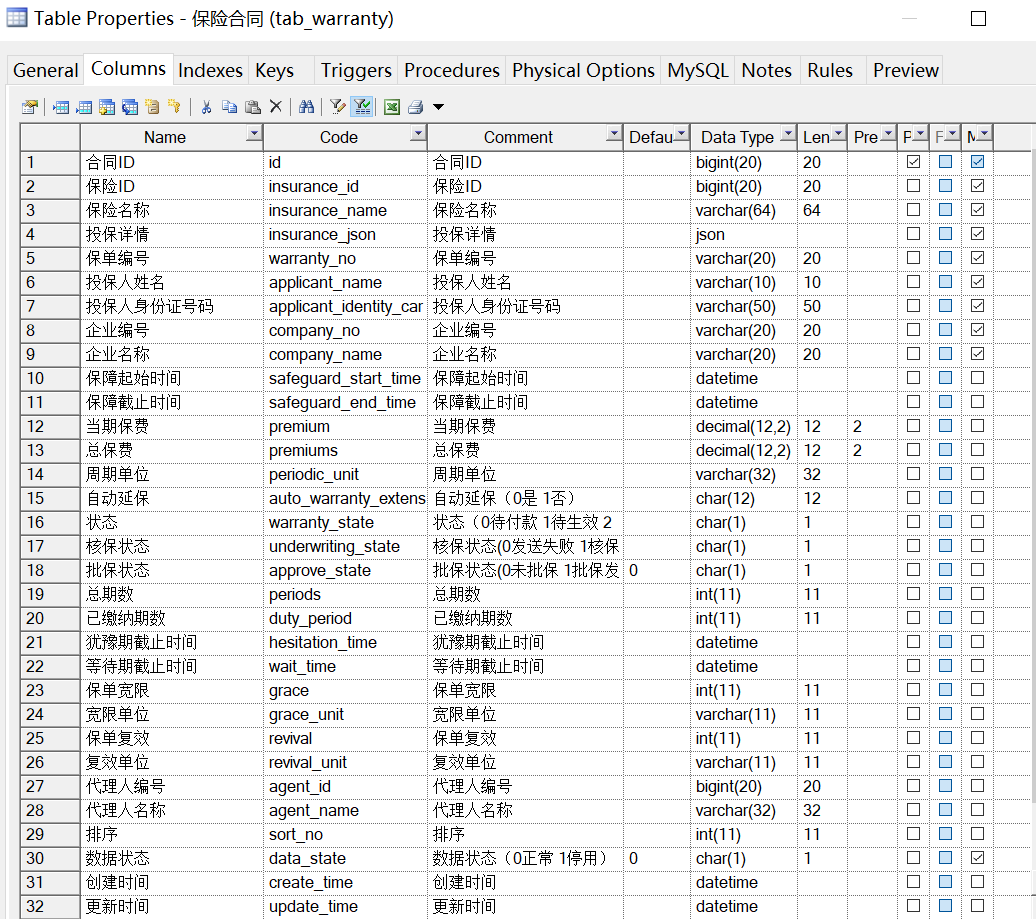
结合需求产品界面和行业经验；可以得出合同与保险产品等的概念模型如下：



* 一个保险产品可以有多个保险合同 ==》 一对多
* 一个保险合同可以有多个合同被保人 ==》 一对多
* 一个保险合同可以有多个合同订单 ==》一对多

**1.2.2、物理模型**

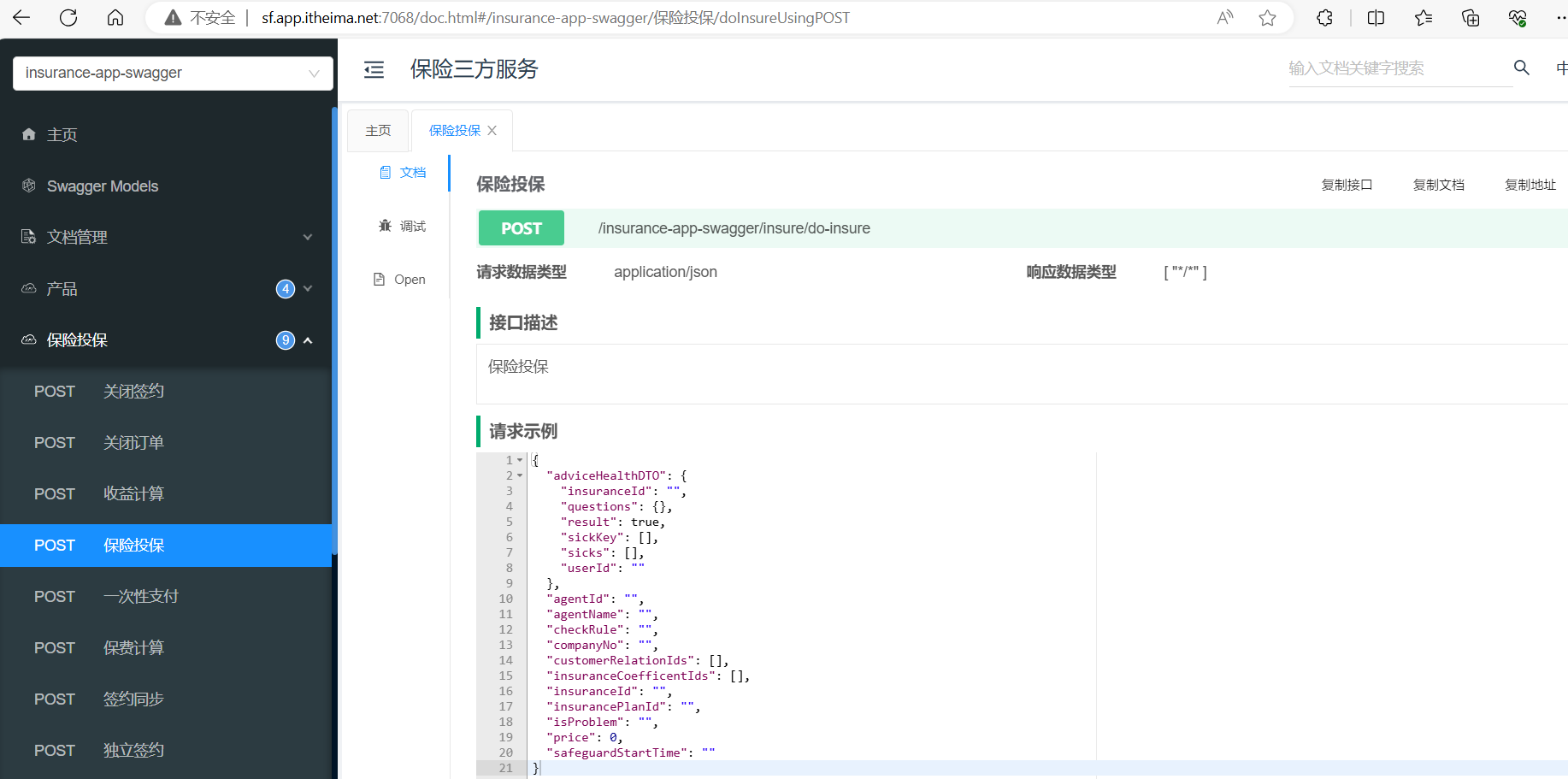
具体生产合同保存合同信息的 保险合同表 物理模型也就是数据库表如下：



**1.3、接口分析**

在swagger上接口地址：

http://sf.app.itheima.net:7068/doc.html#/insurance-app-swagger/%E4%BF%9D%E9%99%A9%E6%8A%95%E4%BF%9D/doInsureUsingPOST



具体在页面中点击下一步之后的跟踪信息接口：

|  |  |
| --- | --- |
| 请求项信息 | 说明 |
| **请求地址** | http://sf.app.itheima.net/api/insurance-app/insure/do-insure |
| **请求方式** | POST |
| **请求参数** | {  "checkRule": "3", #路由key  "companyNo": "100000001", # 企业编号  "insuranceId": "1717789017765937153", # 保险产品id  "insuranceCoefficentIds": [ # 产品系数id集合  "1724370485963145217",  "1724370485950562305",  "1724370485937979393",  "1724370485963145218",  "1724370485967339522",  "1724370485954756610",  "1724370485942173699"  ],  "insurancePlanId": "1726794967499673601", # 产品方案id  "customerRelationIds": [ # 被投保人id集合  "1225092536525656054"  ],  "price": 100 # 投保金额  } |
| **返回结果** | {  "code": 200,  "msg": "操作成功",  "data": {  "id": "1793998854249639939",  "createTime": "2024-05-24 21:33:52",  "updateTime": "2024-05-24 21:33:52",  "createBy": "1371500419615895666",  "updateBy": "1371500419615895666",  "dataState": "0",  "creator": null,  "insuranceId": "1717789017765937153",  "insuranceName": "全民保·稳想赢养老金",  "insuranceJson": " 保险产品对象json字符串 .... 这里省略"  "categoryNo": null,  "warrantyNo": "1793998854249639938",  "applicantName": "王伟",  "applicantIdentityCard": "342401198210019928",  "companyNo": "100000001",  "companyName": "保险集团",  "safeguardStartTime": "2024-05-24 21:33:52",  "safeguardEndTime": null,  "premium": 100,  "premiums": 500,  "autoWarrantyExtension": "0",  "warrantyState": "0",  "underwritingState": "3",  "approveState": null,  "periods": 5,  "periodicUnit": "YEAR",  "dutyPeriod": 0,  "sortNo": null,  "hesitationTime": null,  "waitTime": null,  "grace": 2,  "graceUnit": "MONTH",  "revival": 1,  "revivalUnit": "YEAR",  "checkedIds": null,  "customerRelationVOs": null,  "agentId": null,  "agentName": null,  "warrantyStates": null,  "warrantyOrderVOList": null,  "insuredVOS": null,  "warrantyNos": null  },  "operatorId": 1371500419615895666,  "operatorName": "cutomer@qq.com",  "operatorSex": "0"  } |

**1.4、功能实现说明**

跟进接口之后；找到对应的处理类，本质上处理的投保实现：

1、保存保险合同；

2、保存保险合同订单；

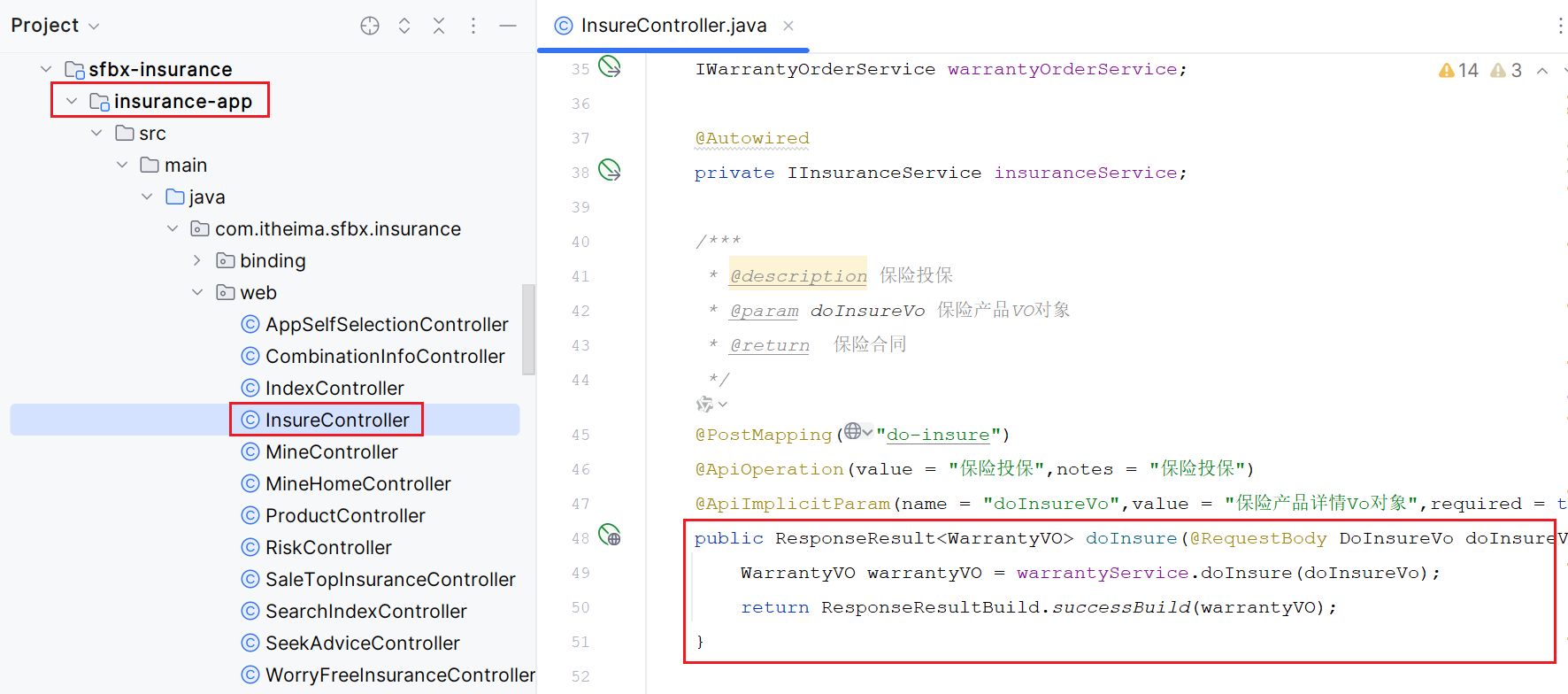
3、保存被投保人

4、发送延迟消息，处理超过10分钟未支付的订单，要取消合同

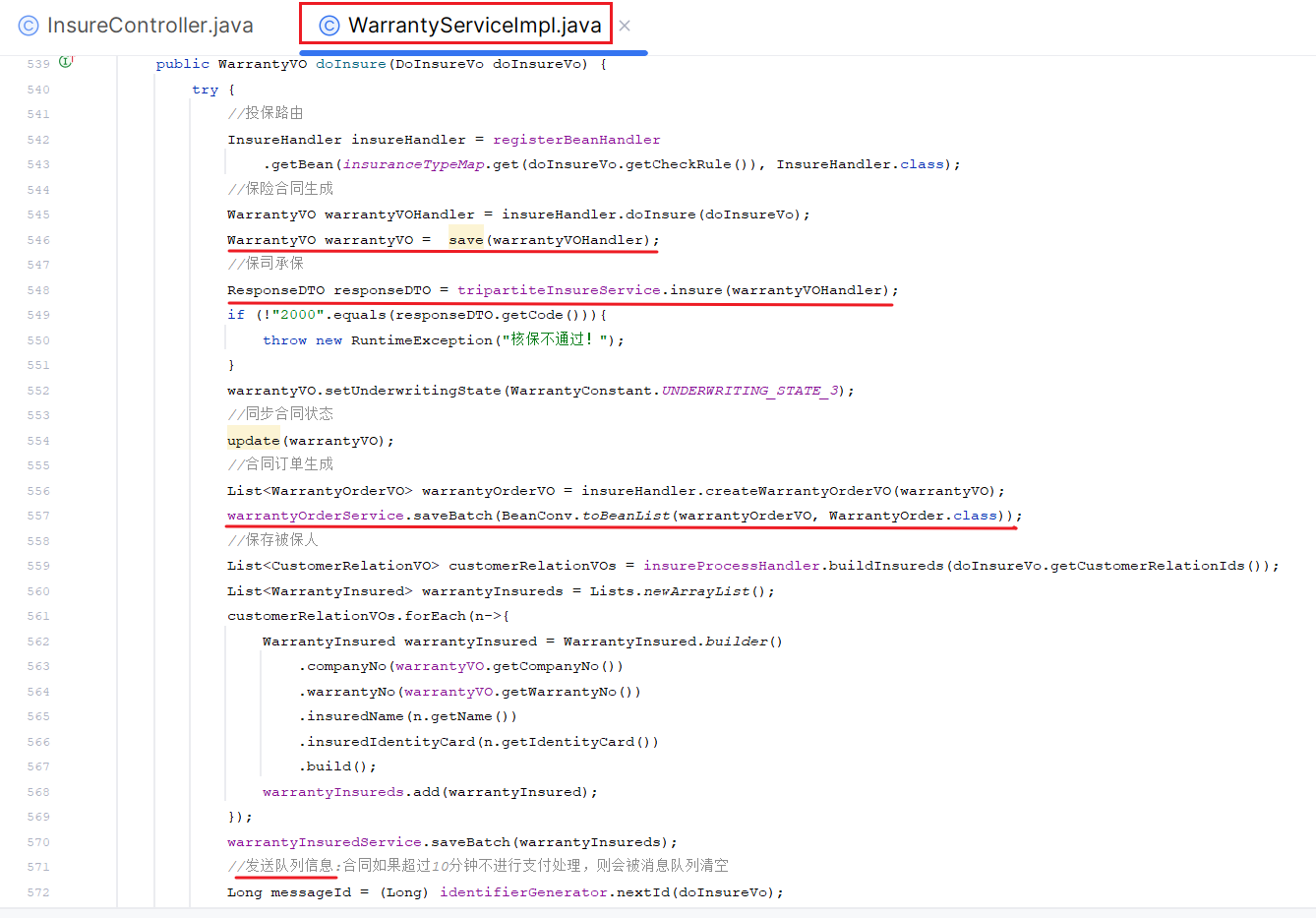
**1.4.1、用户投保**

根据接口信息；跟进代码如下：

处理器：



业务处理方法：



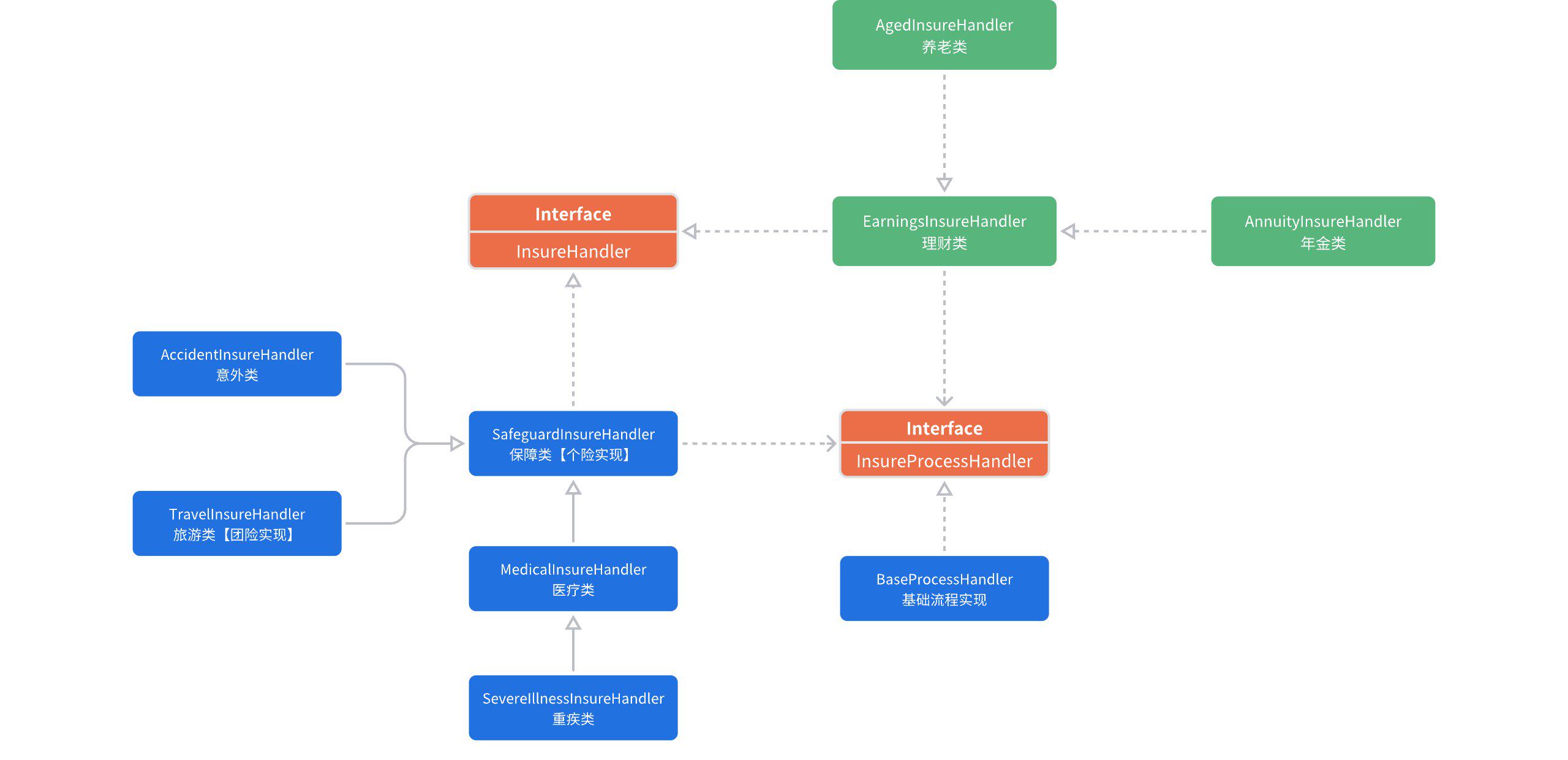
**1.4.2、保险合同生成**

在上述投保主体代码中；包含了合同生成、保司承保、合同订单生成、延迟取消合同。先分析查看合同生成

**保险合同生成：**

* 重新计算保费或理财收益，不能直接用前台传递金额，用以保证合同安全---可以复用之前保费计算和收益试算代码
* 构建合同信息，形成合同快照信息
* 用户信息：包含投保人和被保人的完整信息
* 保费信息：包含保费、缴费方式、缴费期限
* 保障信息：投保时所选方案对应的保障项信息，以及保障期限的起始
* 协议条款：保险附件相关信息

处理保险合同生成的处理器属于InsureHandler接口；它的实现类的doInsure方法来完成具体的投保处理流程，需要注意InsureHandler接口有多个实现类：



由于InsureHandler接口实现类太多，我们主要看理财类保险合同的生成，也就是看EarningsInsureHandler类的doInsure方法：



具体上述提到的对 犹豫期截止时间、等待期截止时间、保障截止时间 的处理如下：



* **犹豫期**： 指投保人在收到保险合同后的一个特定时间段（通常为10天至20天，具体时间以保险合同中的条款为准），如不同意保险合同内容，可将合同退还保险人并申请撤销的“冷静期”。在这个期间内，投保人如果决定退保，通常可以全额退还已交的保费。由当前时间+设置的保险产品犹豫期天数 叠加而来
* **等待期**： 也称观察期、免赔期是指保险合同在生效后的一个指定时间段内，即使发生保险事故，保险公司也不承担保险责任。由当前时间+设置的保险产品等待期天数 叠加而来
* **保障结束时间**：指保险合同约定的保险责任终止的时间点。由保障开始时间 + 保障周期 叠加而来

|  |
| --- |
| **思考问题**：  *1、在投保时生成保险合同，然后要到保司审核合同；如果保险公司的接口出了异常；当前这个合同应该怎么办？*  --- 定时重试 ——》XXL-JOB 或 自旋  2、用户投保时间（6.72s）过长  投保的时候处理了6个比较耗时的事情；所以时间比较长。——》优化  3、用户在投保时间过长情况下；会疯狂的刷新页面也就可能会出现生成多个保险合同的问题？该如何解决  ——》表单幂等处理 |

**1.5、性能优化**

**1.5.1、需求**

在点击 下一步 也就是用户投保的时候；由于要处理的业务流程比较多，处理效率比较差；需要对该处理进行优化。

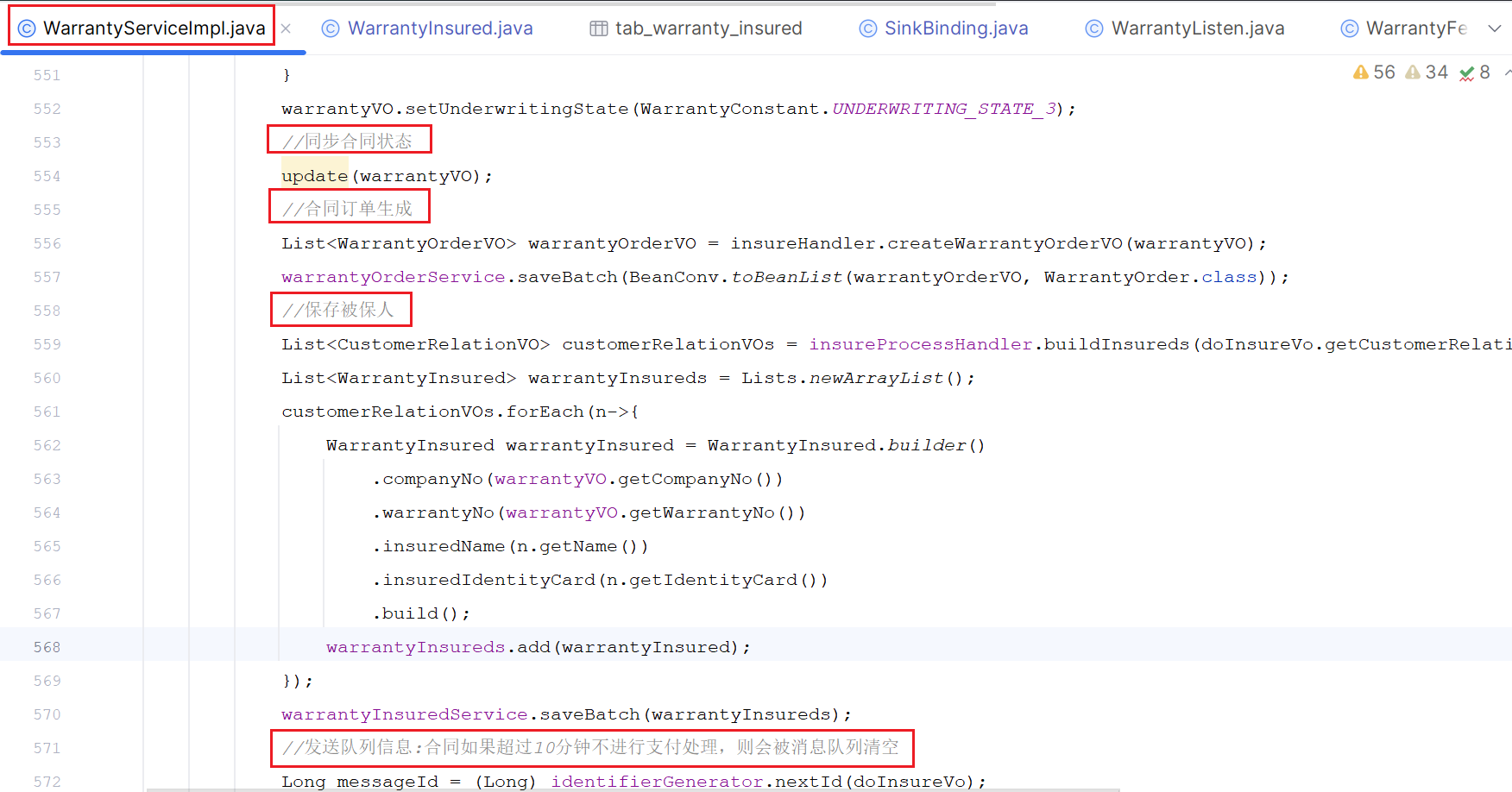
当前点击下一步的时候；它的总处理时长为：



也就是说个人开发机器上测试投保的话；花费了4.92秒（第一次访问时长会比较长，实际再次访问没有那么长，与人说时大概2-3秒合适）；如果到了服务器上机器性能好一些，但是也将比较耗时。需要优化。

**1.5.2、分析**

这是处理投保的代码：



我们分析代码发现；在过程中红色框住的是操作，也就是优化可以考虑的地方；

* 1、**更新合同状态**：目前更新整个对象，实际上只需要更新状态和更新时间即可；
* 2、合同订单生成：关键操作，需要与合同、订单等一起控制在一个事务中；暂不优化
* 3、保存被保人：与合同订单生成是一样的，需要控制在一个全局事务中；暂不优化
* 2、**发送未支付延迟消息**：这块可以使用线程池异步发送延迟消息，因为MQ这块是技术问题可以控制发送的可靠性；所以可以剥离出全局事务控制，利用异步线程处理

**1.5.3、优化（重要）**

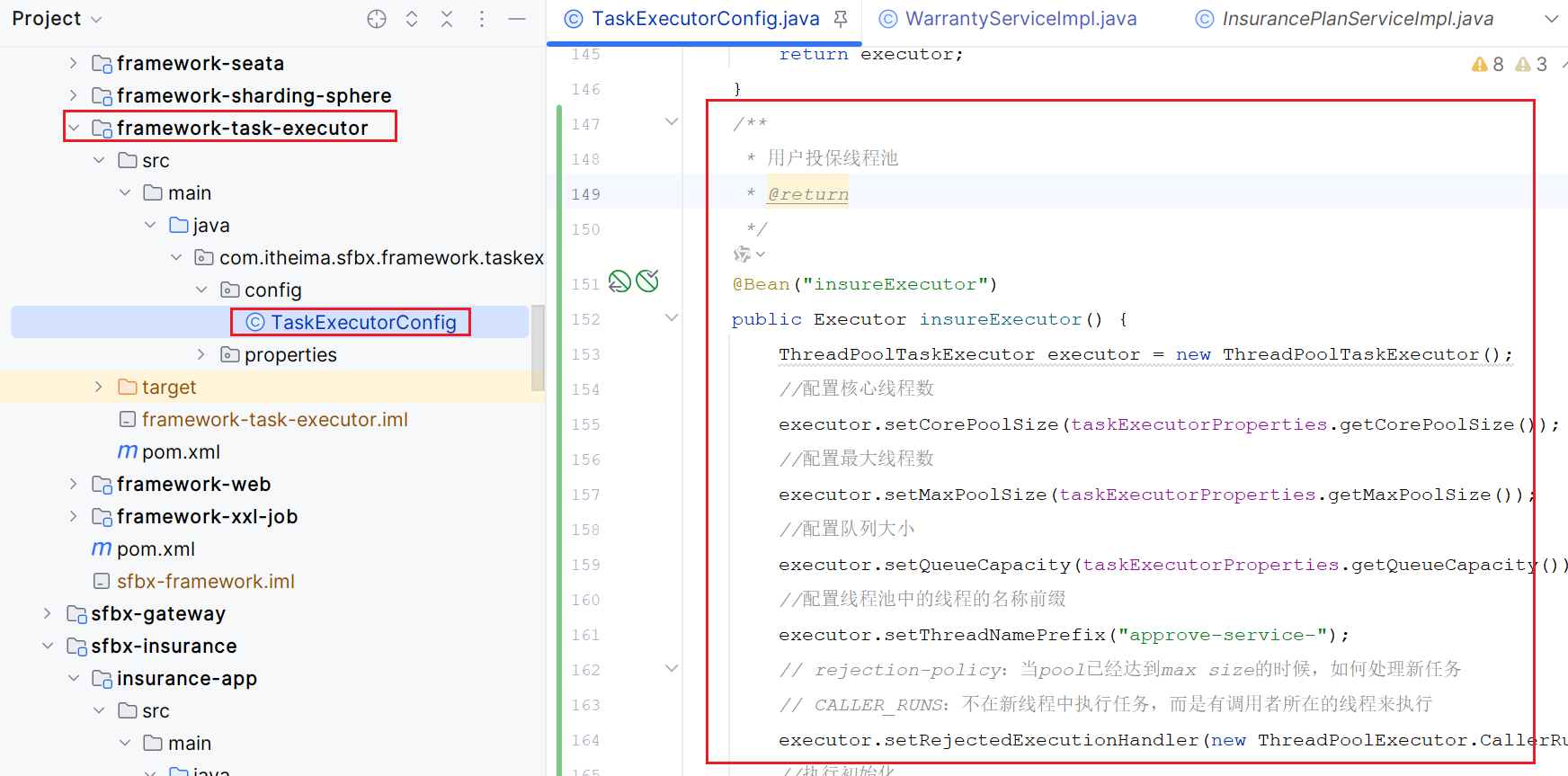
**1）优化更新状态**



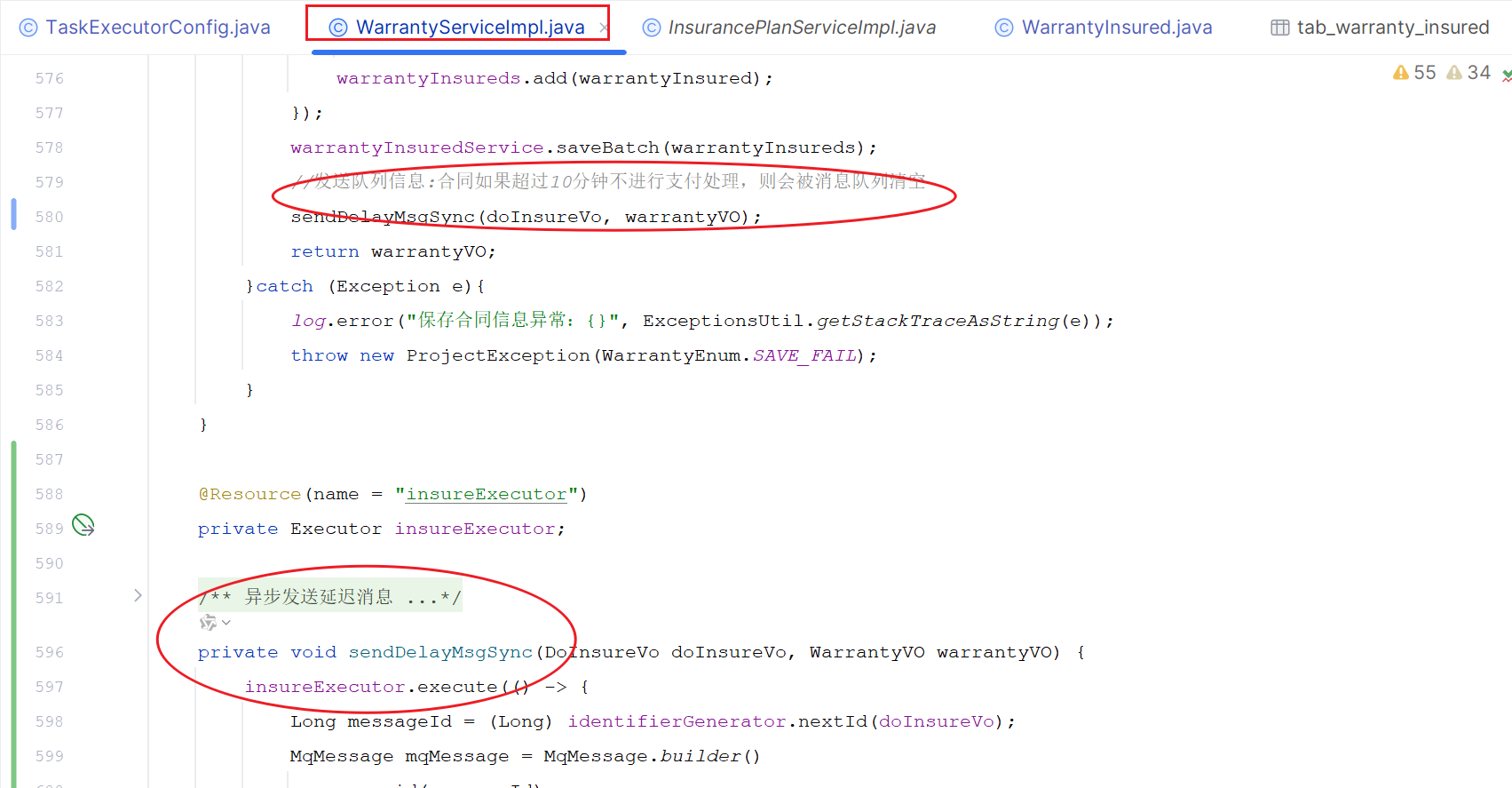
|  |
| --- |
| 此处的优化**启示**：  以后出去工作，大多情况下，都是对于原有操作的数据库优化操作；比如：  1、减少查询时关联过多的表  2、查询时索引的应用  3、减少不必要的操作（本次优化属于这方面） |

**2）异步发送**

配置线程池对象：



修改投保代码：

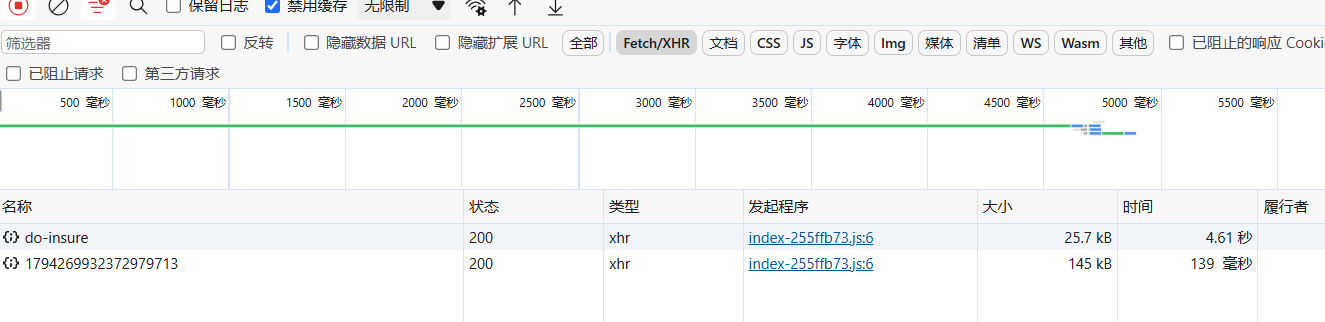


**1.5.4、测试**

启动之时不要忘记，需要启动保司的模拟服务器：

打开第一天 资料/代码/picc-simulate 并启动。

在优化后；再次进行投保，也就是点击下一步；其耗时情况如下：



从 4.92秒 优化到 4.61秒；因为我们优化的动作并不大，所以提升也比较有限；提升了：8% 。这是都是系统启动之后第一次访问并投保的时间对比；多测试几次效果会有不同。

**继续优化方向**：

* **合同订单** 可以异步处理；由于是本地数据库操作，所以假设操作失败的话，有补偿操作则不加入全局事务也是可以的
* **批量保存投保人** 可以异步处理；由于是本地数据库操作，所以假设操作失败的话，有补偿操作则不加入全局事务也是可以的
* 再有就是：参数预处理 时大量的查询，也是突破口

|  |
| --- |
| **面试问题**  **你使用过线程池吗？核心参数有哪些？哪些地方使用到了？**  有；ThreadPoolExecutor 创建；7个参数；核心线程数、最大线程数、空闲线程存活时间、空闲时间单位、工作队列、线程工厂、拒绝策略；  在用户投保的时候使用了；投保时间过长，大概需要2.6秒左右；分析之后对投保的流程进行梳理；  1、参数的预处理；原来的查询为了方便大量查询了不必要的关联数据，比如：给付方案、方案保障项等暂时不需要校验的数据  2、合同审批调用方式：原来直接通过httpclient访问，改造成使用okhttp并配置了线程池  3、合同审批后状态的修改；原来更新了一个完整的大对象，所有数据更新，实际只有状态的更变；优化只更新状态  4、合同订单、投保人的批量处理由主线程同步处理改为采用异步线程池处理  5、订单未支付的延迟消息发送由同步改为异步线程池发送  花费了半个月时间；我先梳理原来的投保流程；因为投保是保险的核心流程，所以花了一周才梳理完，然后再提出改造方案和组长也探讨过；没有问题就按照上述的方式去优化；再反复测试。从原来的2.6秒优化到900毫秒；提升 65% 的投保效率。  现在会答了吗？ |

**1.6、小结**

在这个章节中；主要梳理了用户投保的主流程：



在上述流程中；先主要分析、查看了生成合同的需求、数据库表结构和代码。

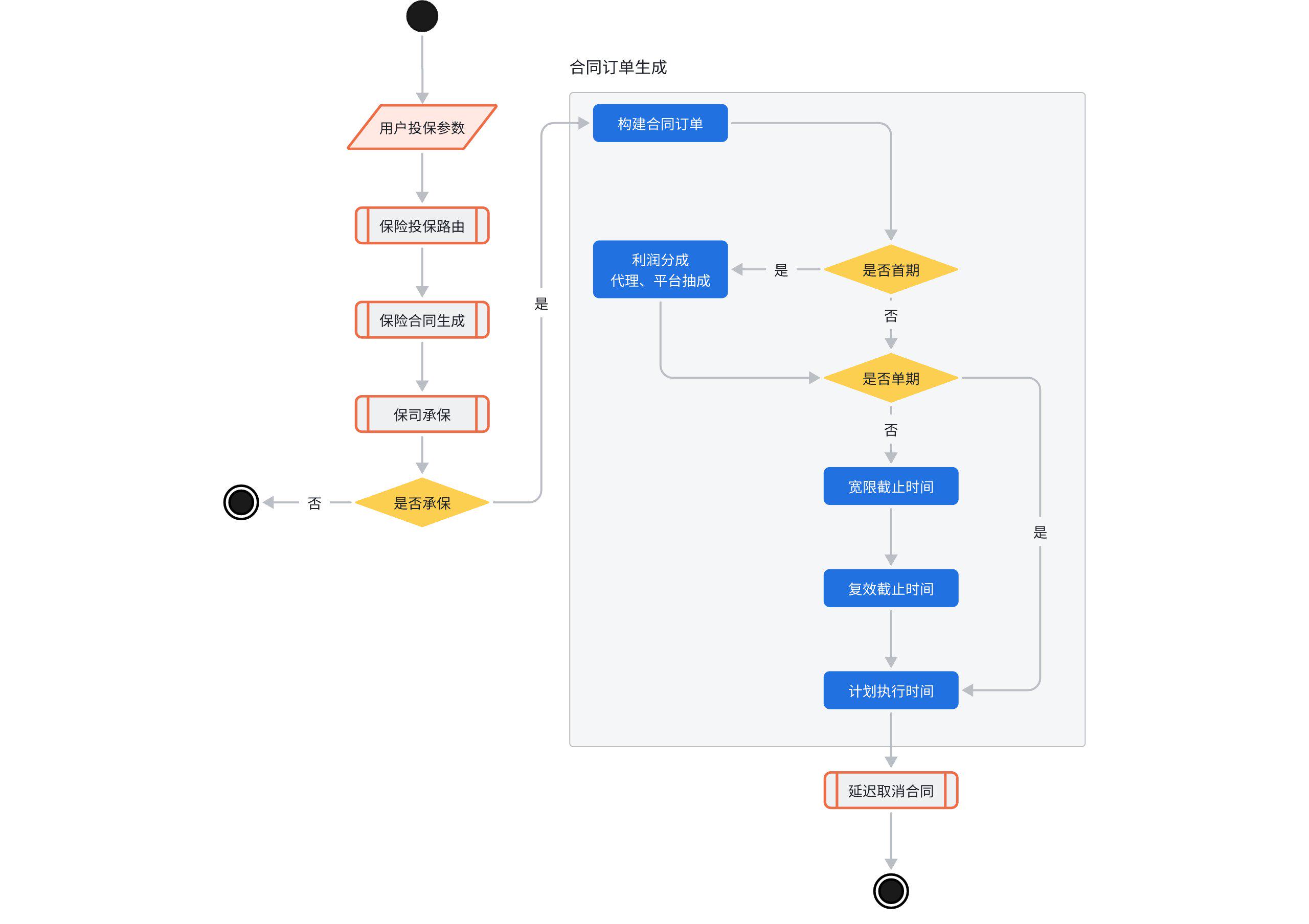
而上述流程中的 第2步 保司承保 需要商务与对应的保险公司谈判洽谈，并调用它接口实现保险合同的审核、承保，承保之后会返回合同的状态更新保存的合同状态。

|  |
| --- |
| **说明**：  保司承保；使用了外部项目进行模拟。大家在使用的时候时候需要将 “资料/代码/picc-simulate” 使用IDEA新的工程打开；然后启动 com.itheima.picc.simulate.PICCSimulateApp 即可。 |

**2、合同订单**

**2.1、需求分析**

按照上述的流程分析；合同生成后，我们需要按照合同的内容生成对应用于支付的合同订单，我们来看下整个合同订单的生成流程：

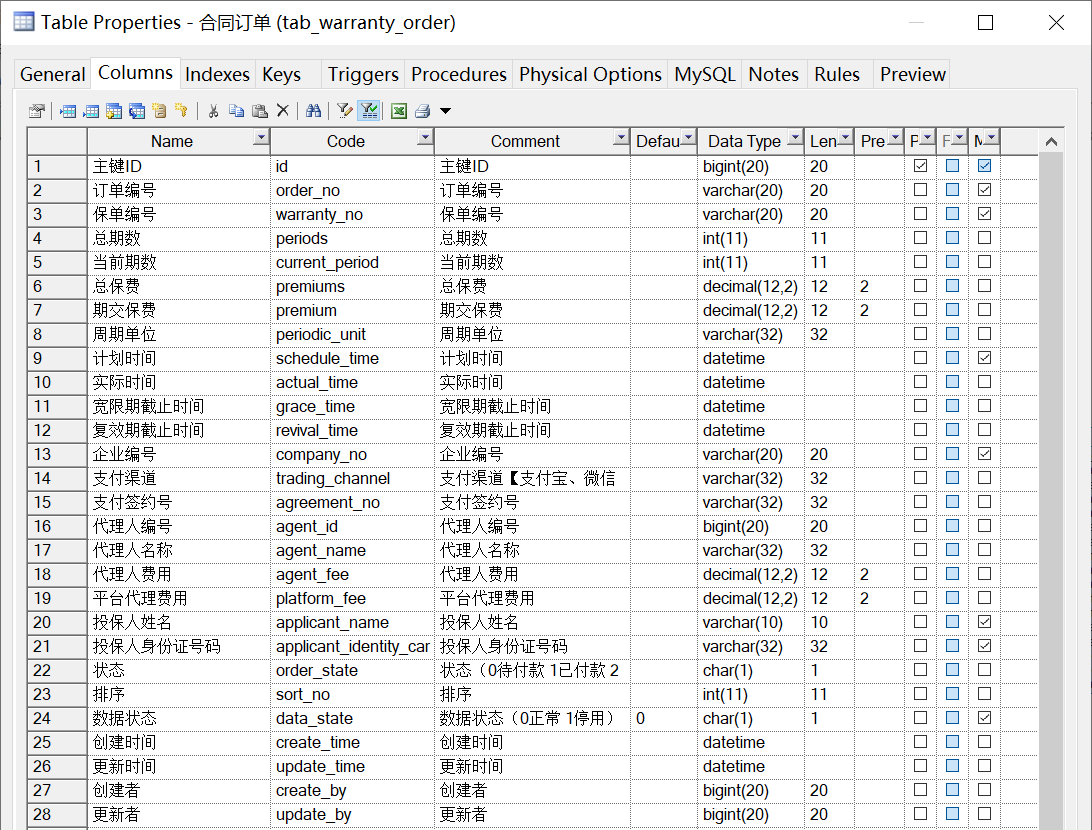


其中有几个比较关键的参数：

* **计划时间**：需要按照每个周期间隔分析出每期的**计划执行**时间
* **宽限期截止时间**：在分期缴纳保险费的保险合同中，在首次缴付保险费以后，如果投保人在各分期到期没有缴费，保险公司将给予投保人的一段时间。通常，这个宽限期是60天。在这段时间内，只要投保人缴纳了保险费，保险合同就会继续有效。每期计划时间按照款宽限间隔和宽限单位推算出
* **复效期截止时间**：在保险合同因未按时缴纳费用等原因失效后的时间内，投保人可以向保险公司提出恢复保险合同效力的最后期限**。**每期计划时间按照款复效间隔和复效单位推算出
* **利润分成**：**首期**会按照保险发布时候指定的分成比例进行分成，其它期则不再需要计算也没有

**2.2、数据库表设计**

具体的合同订单数据库表如下：



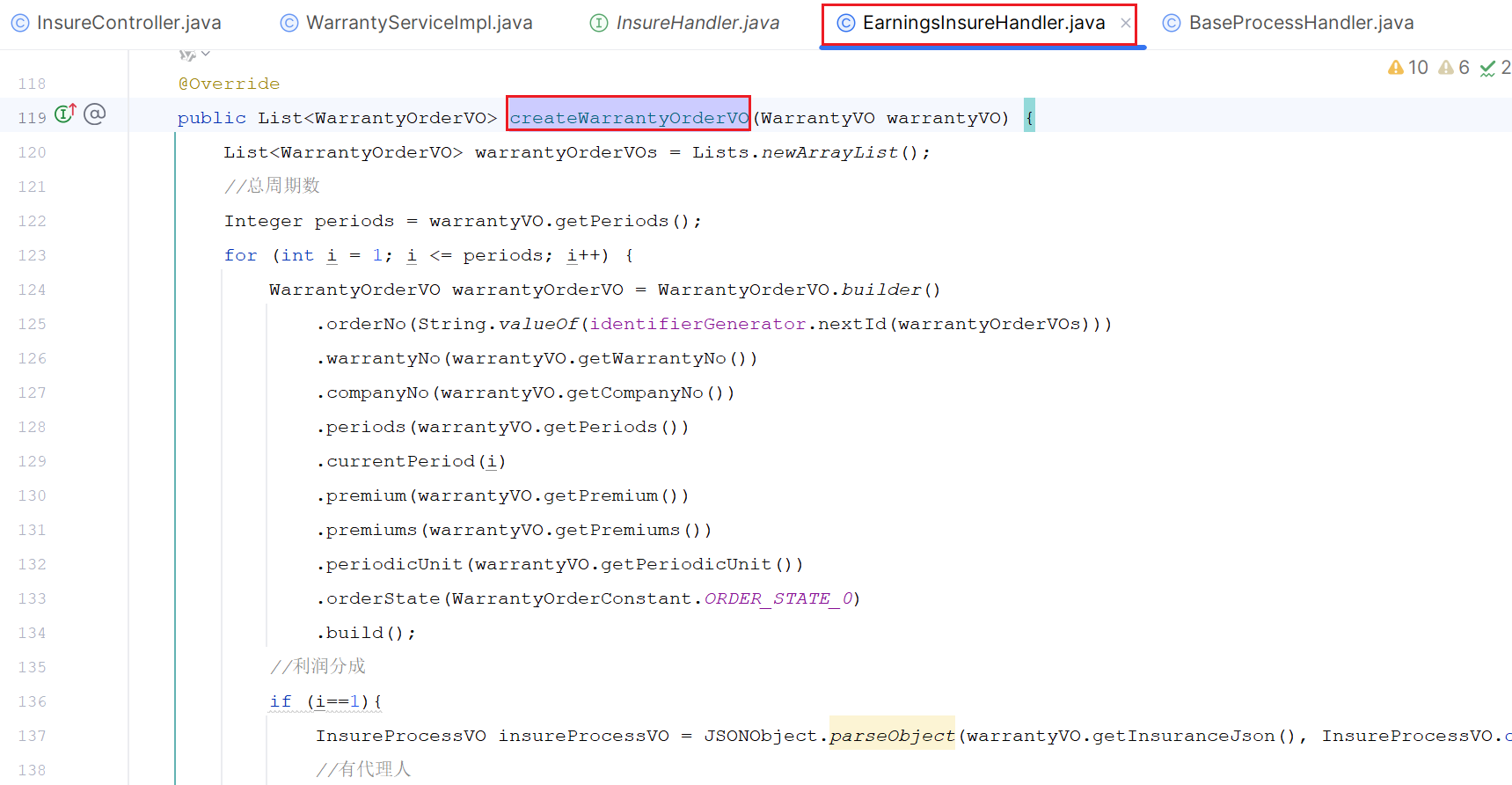
一个保险合同，可能对应有多个合同订单；因为如果是 按期缴纳 的话；每一期都是一个合同订单。所以保险合同与合同订单的关系是：一对多

**2.3、功能实现**

具体创建合同订单的代码实现如下：



再次跟进入到如下类方法中：com.itheima.sfbx.insurance.handler.impl.EarningsInsureHandler#createWarrantyOrderVO



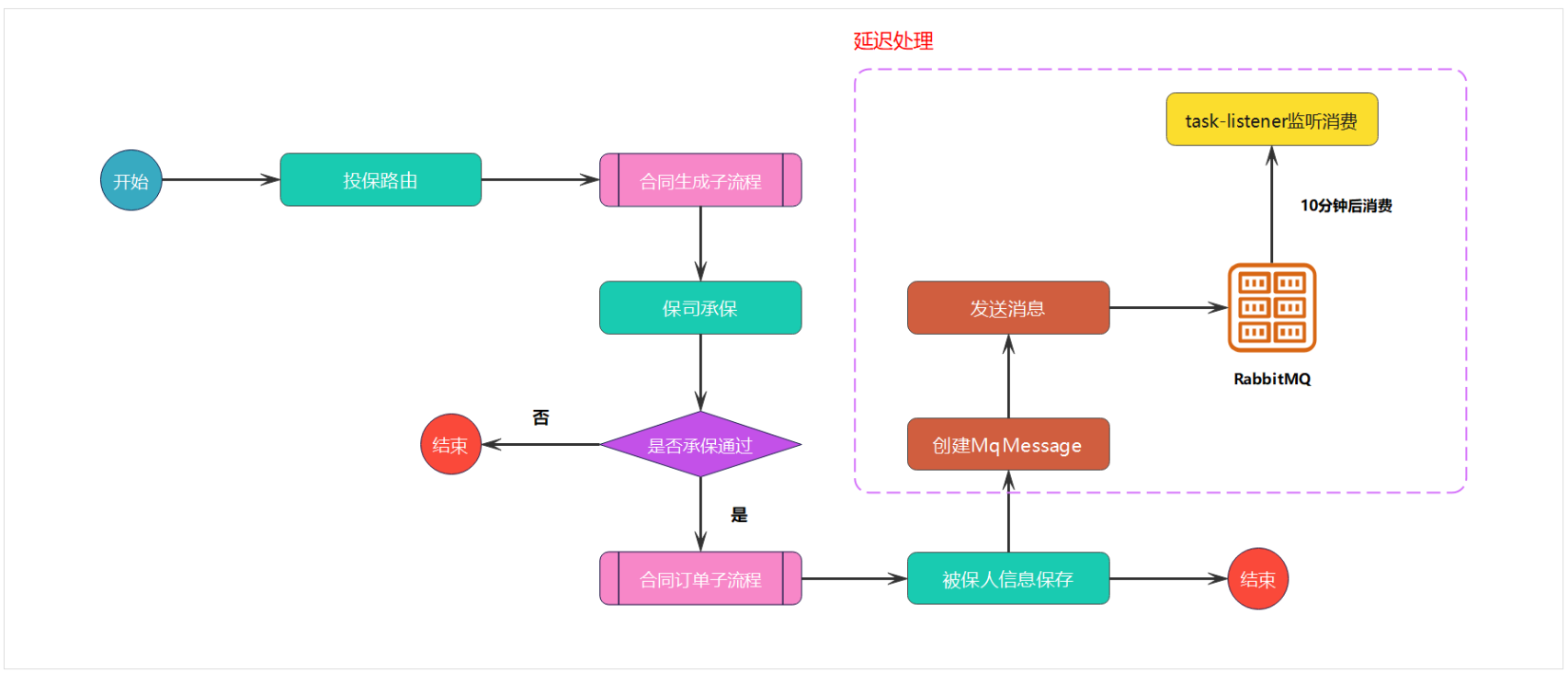
**2.4、小结**

一个保险合同，根据 投入方式 的不同会生成1或多个合同订单；而这些订单每个生成之后都是内容不可变的，这是作为支付的凭据。

**3、取消合同**

**3.1、需求分析**

完成保险合同、合同订单后，如果用户10分钟内未进行支付，需要取消合同和订单。具体的分析流程如下：

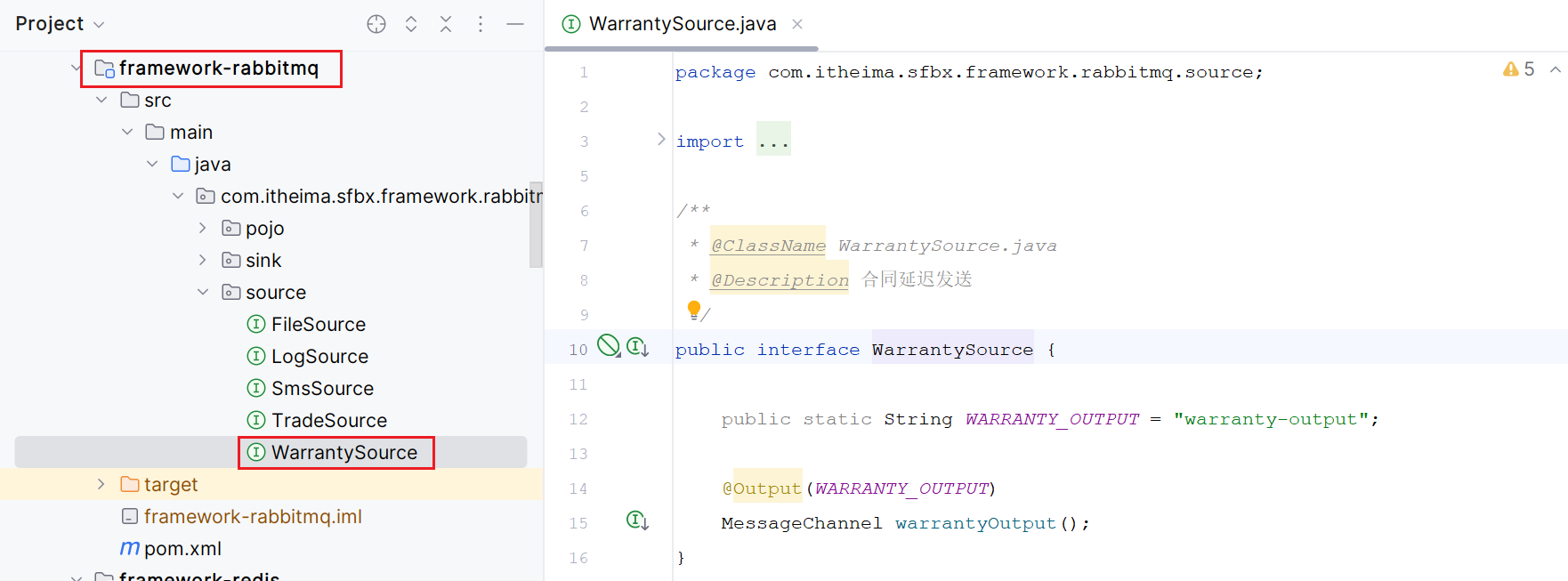


* 发送消息：insurance-app 发送带有合同编号的延迟消息
* 接收消息：task-listener 接收消息，调用 insurance-app 提供的接口，根据支付状态，如果未支付的话则取消合同

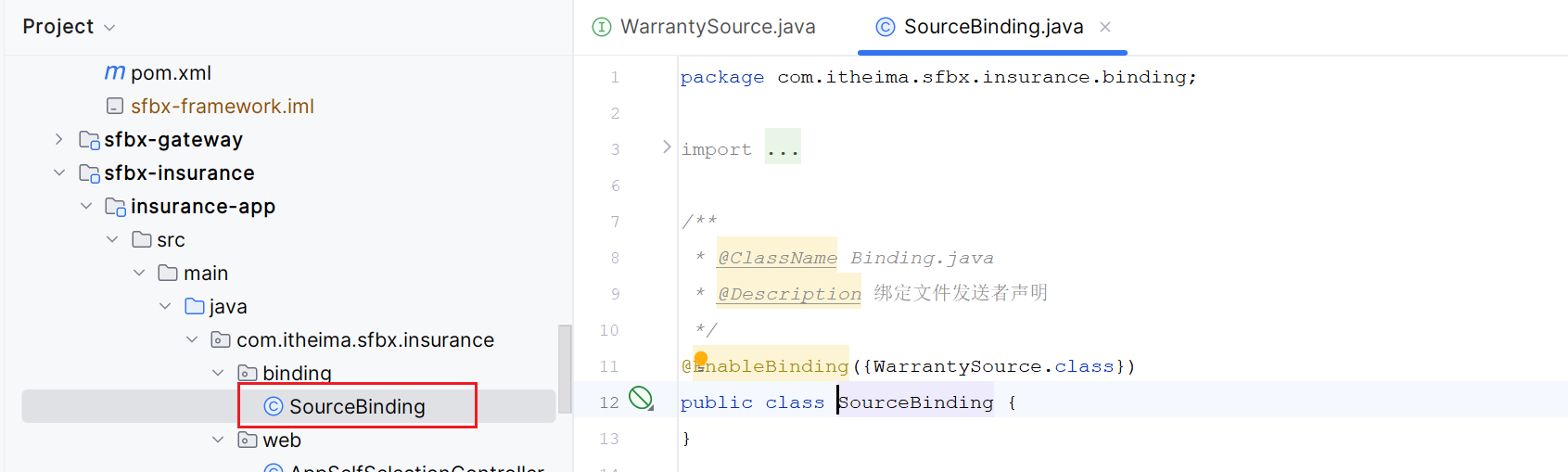
**3.2、发送延迟消息**

根据SpringCloud Stream中发送信息的用法；如下三步骤可实现发送延迟消息：

**1）编写Source**

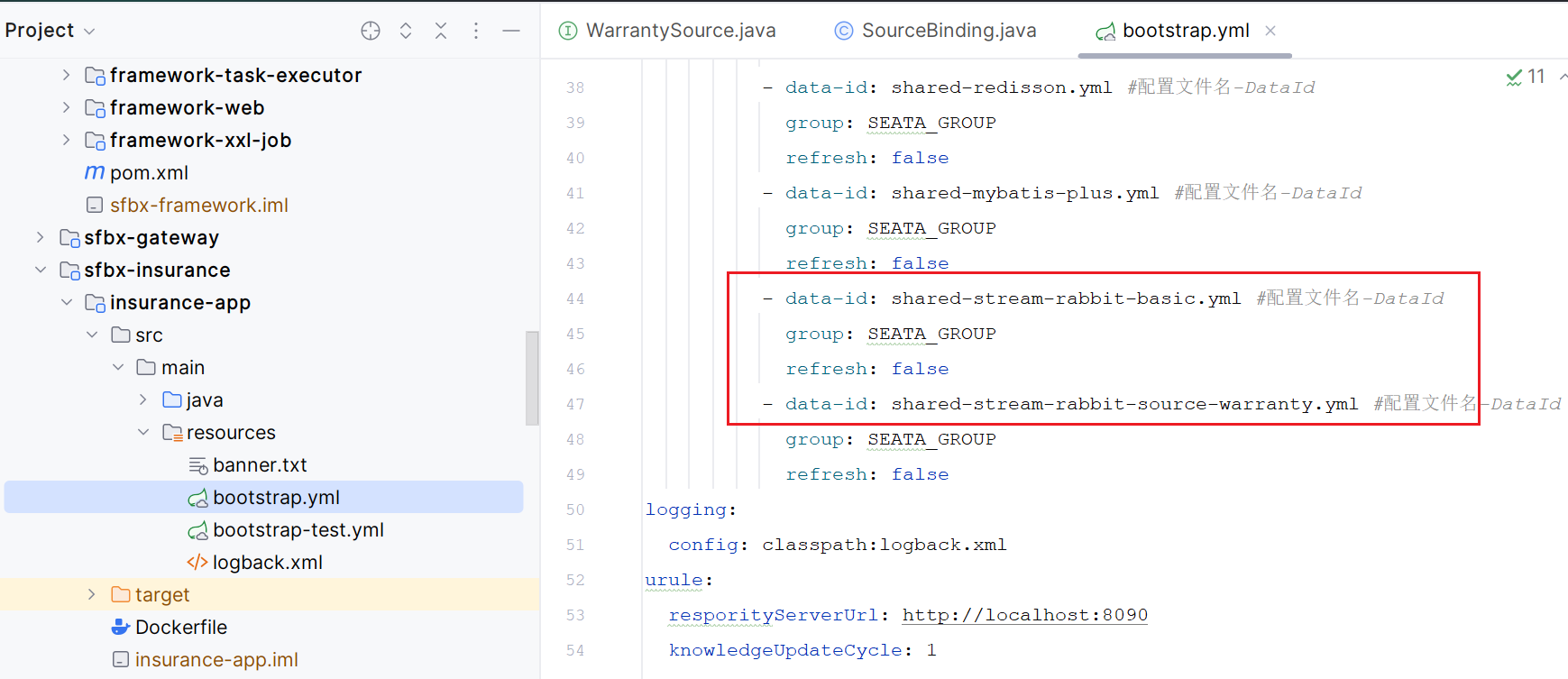


开启Source接口的绑定

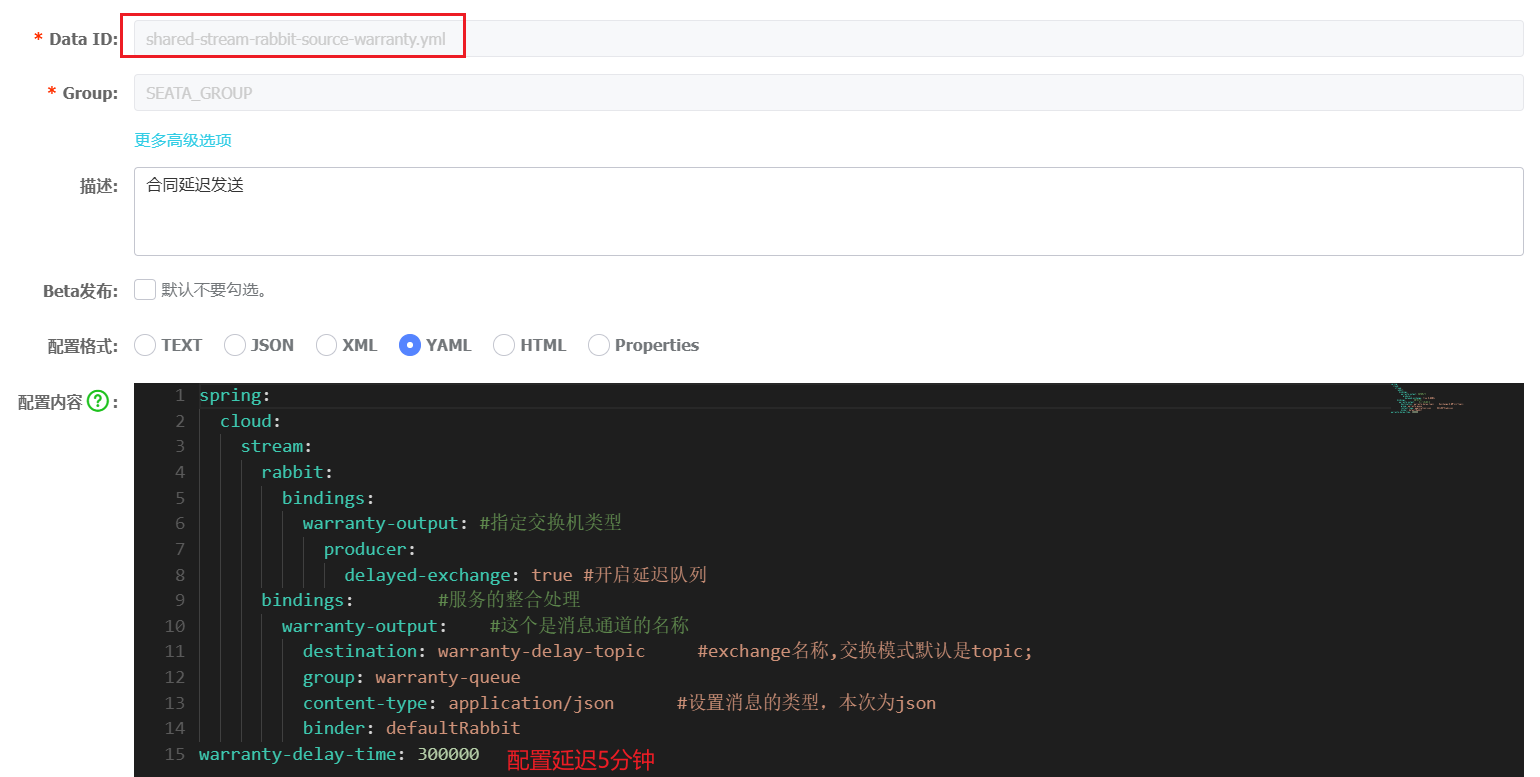


**2）配置绑定**

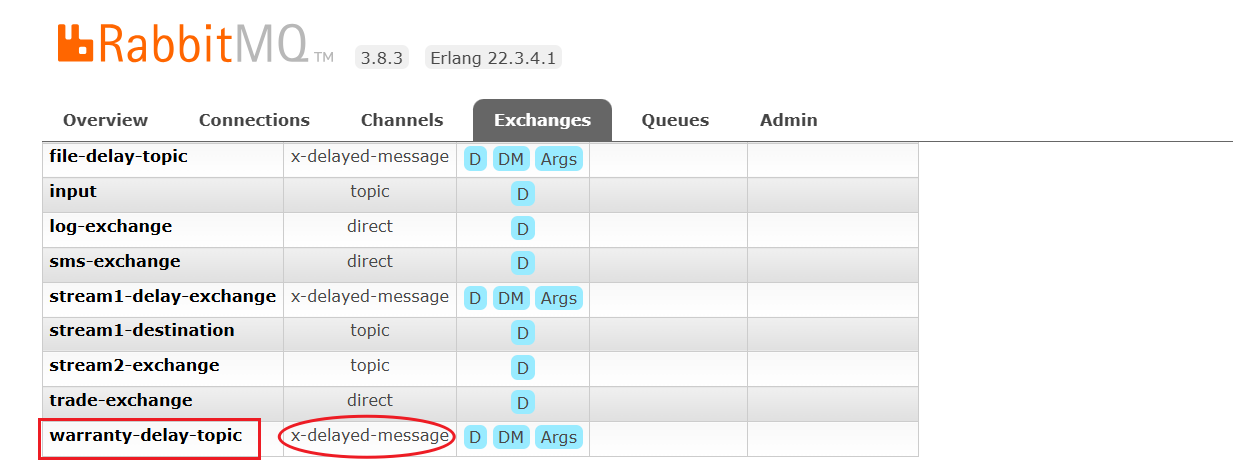
在insurance-app模块的bootstrap.yml中引入关于springcloud-stream配置

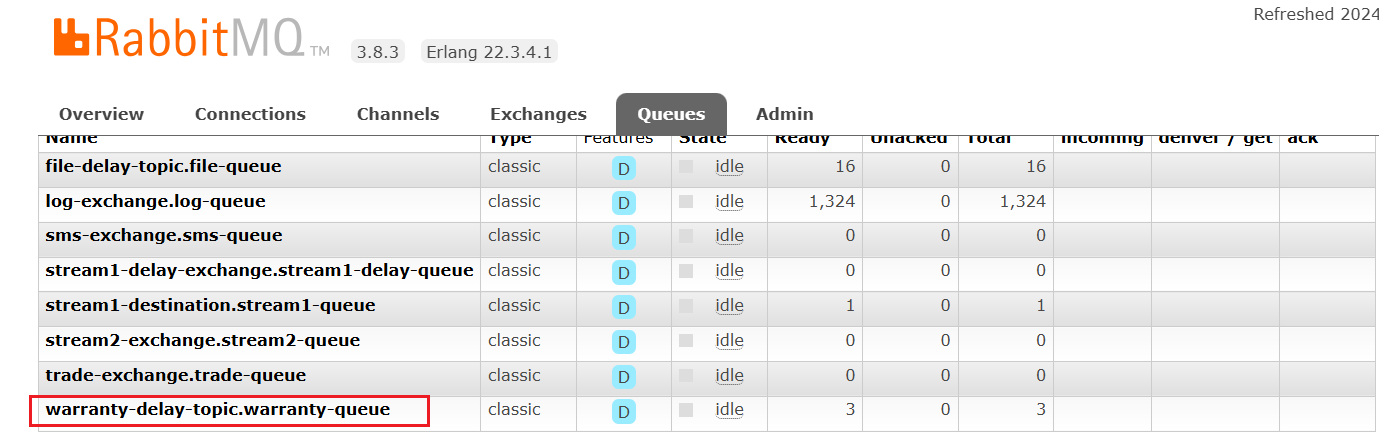


访问nacos查看对于发送源的配置：http://192.168.12.129:8848/nacos/#/configurationManagement?dataId=shared-stream-rabbit-source-warranty.yml&group=&appName=&pageSize=&pageNo=&namespace=



配置后访问rabbitMQ也能看到对应的交换机和队列：





**3）发送延迟消息**



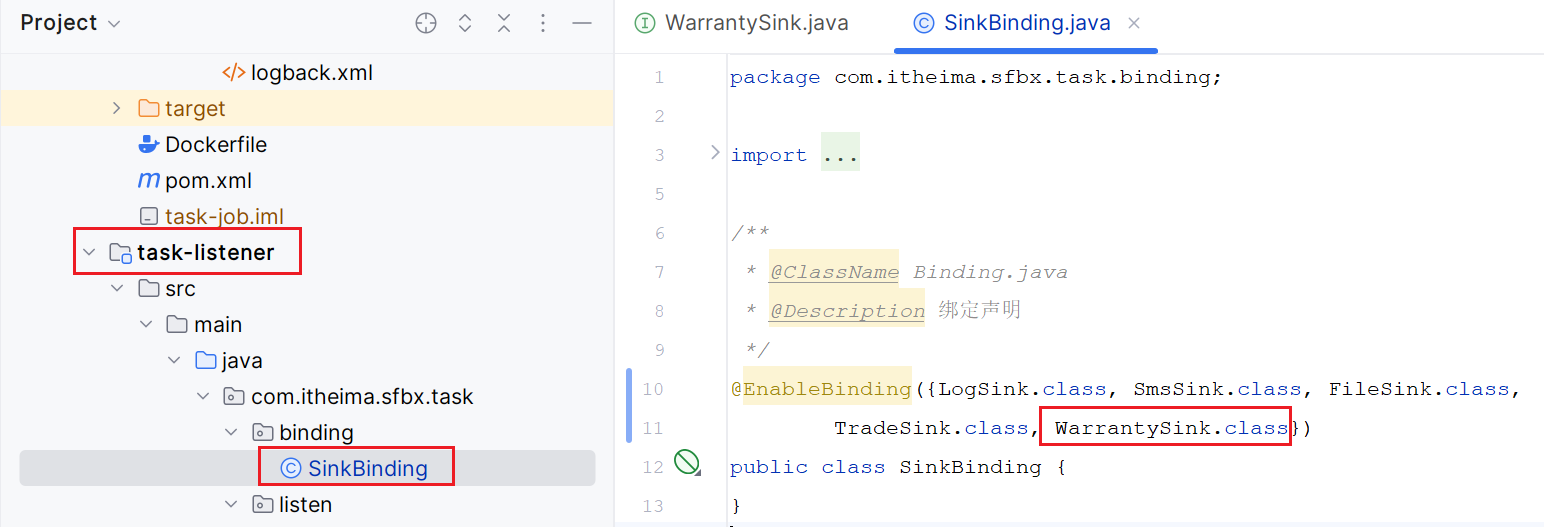
**3.3、接收消息-取消合同**

根据SpringCloud Stream中接受消息的用法；如下三步骤可实现接收消息：

**1）编写Sink类**

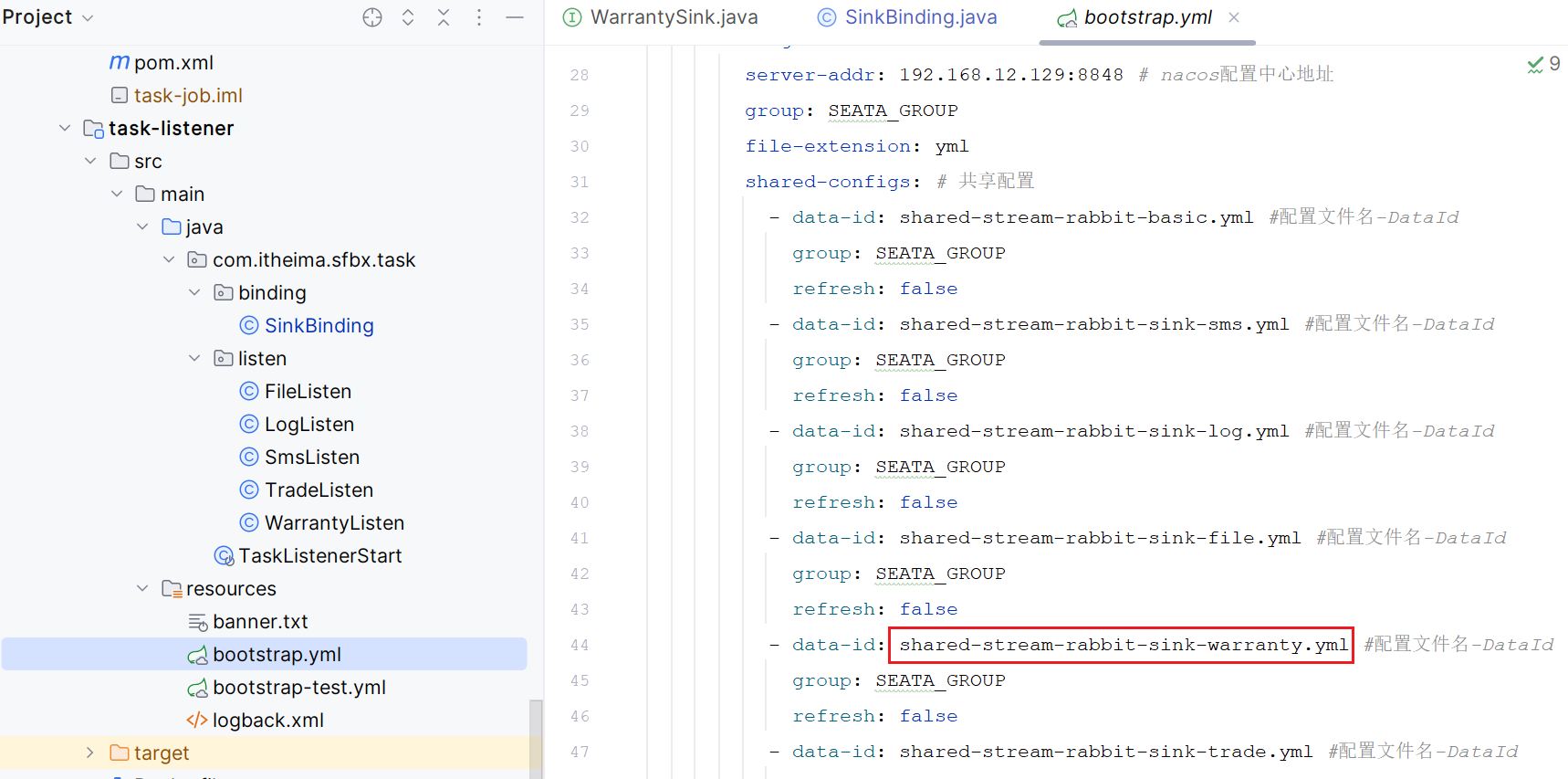


开启Sink接口的绑定



**2）配置绑定**

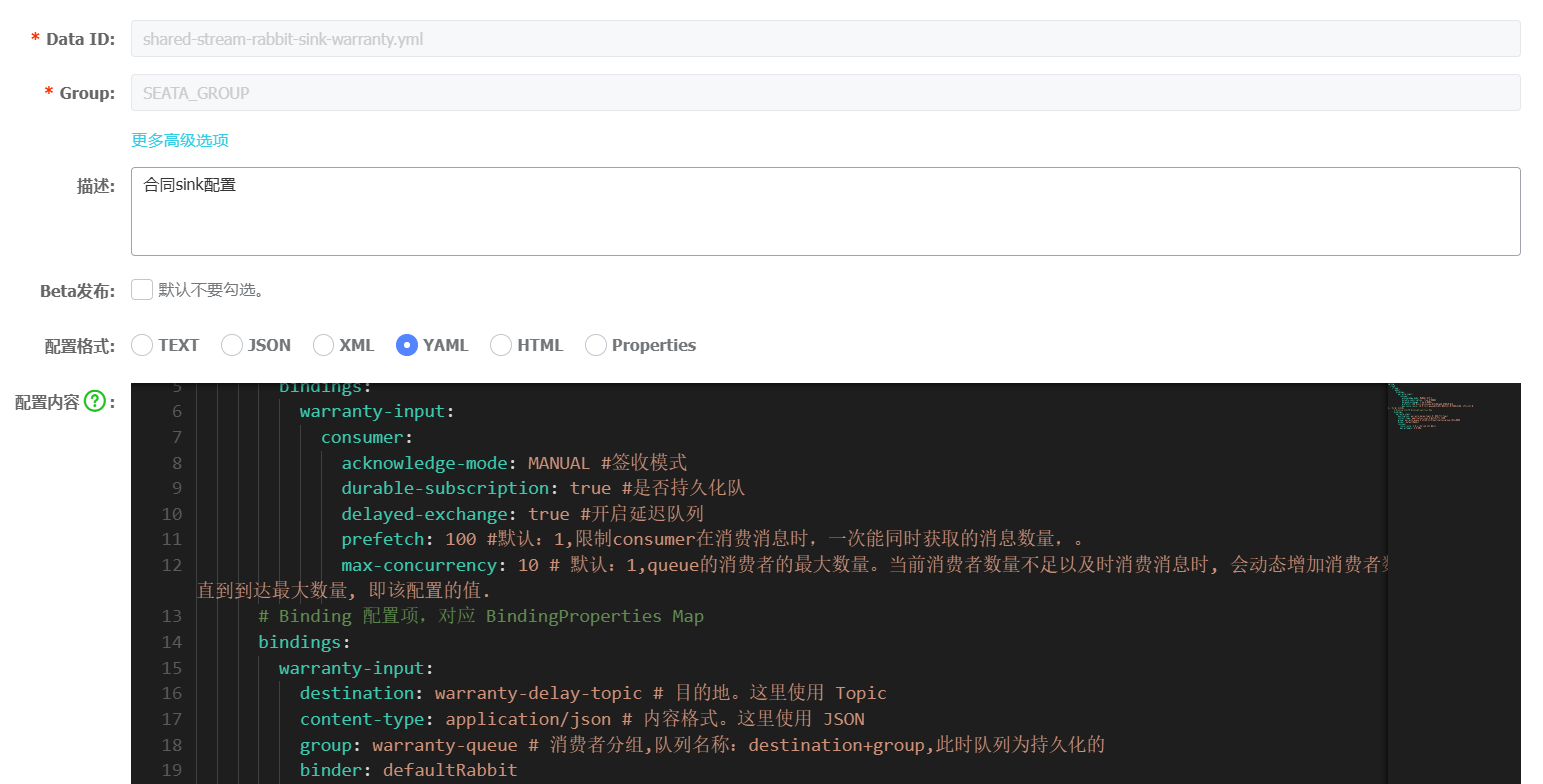
在 task-listener 中配置sink；需要在 bootstrap.yml 中引入关于springcloud-stream配置



在nacos的配置中心查看shared-stream-rabbit-sink-warranty.yml配置，

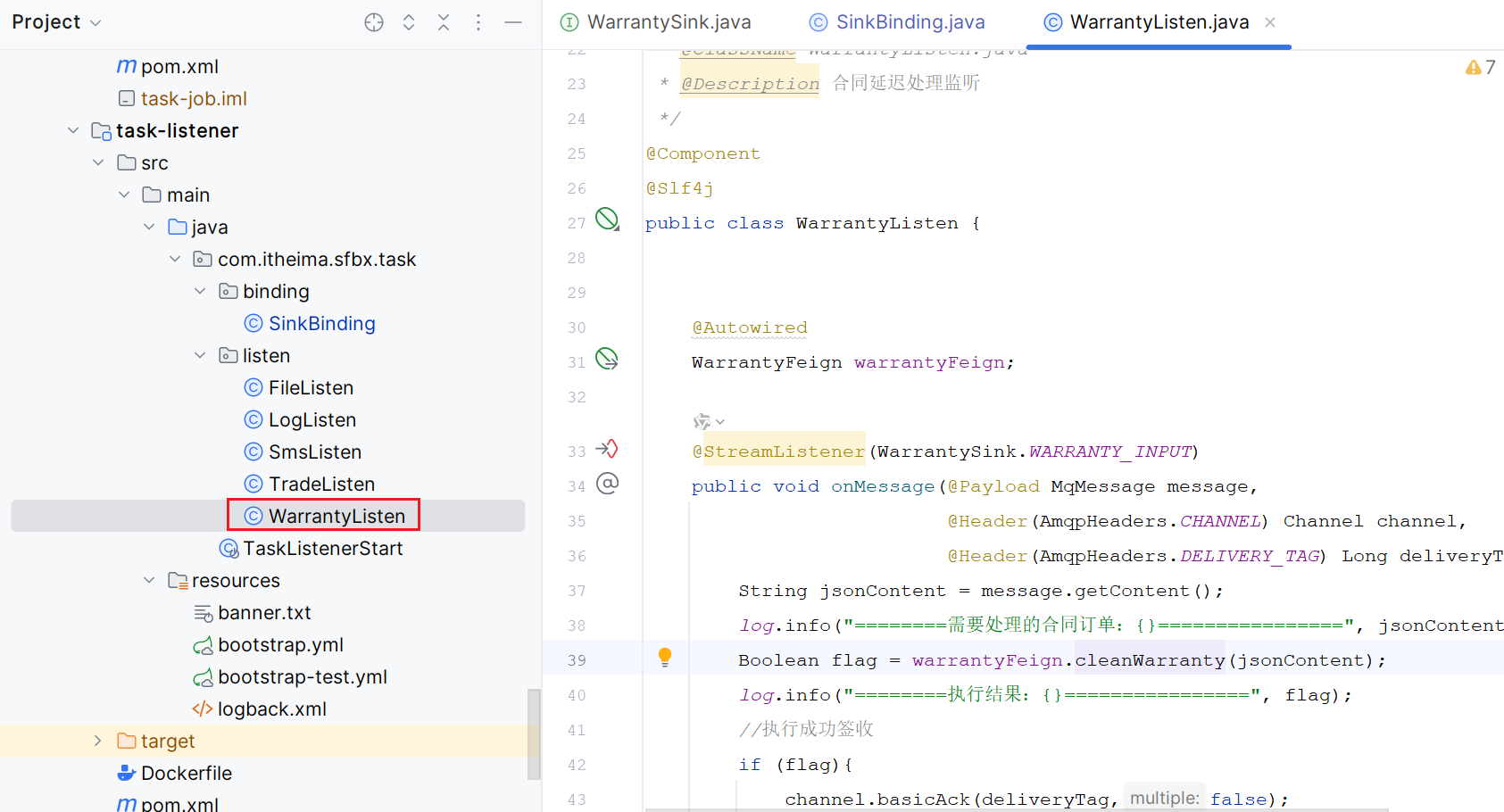
http://192.168.12.129:8848/nacos/#/configurationManagement?dataId=shared-stream-rabbit-sink-warranty.yml&group=&appName=&pageSize=&pageNo=&namespace=

其内容如下：



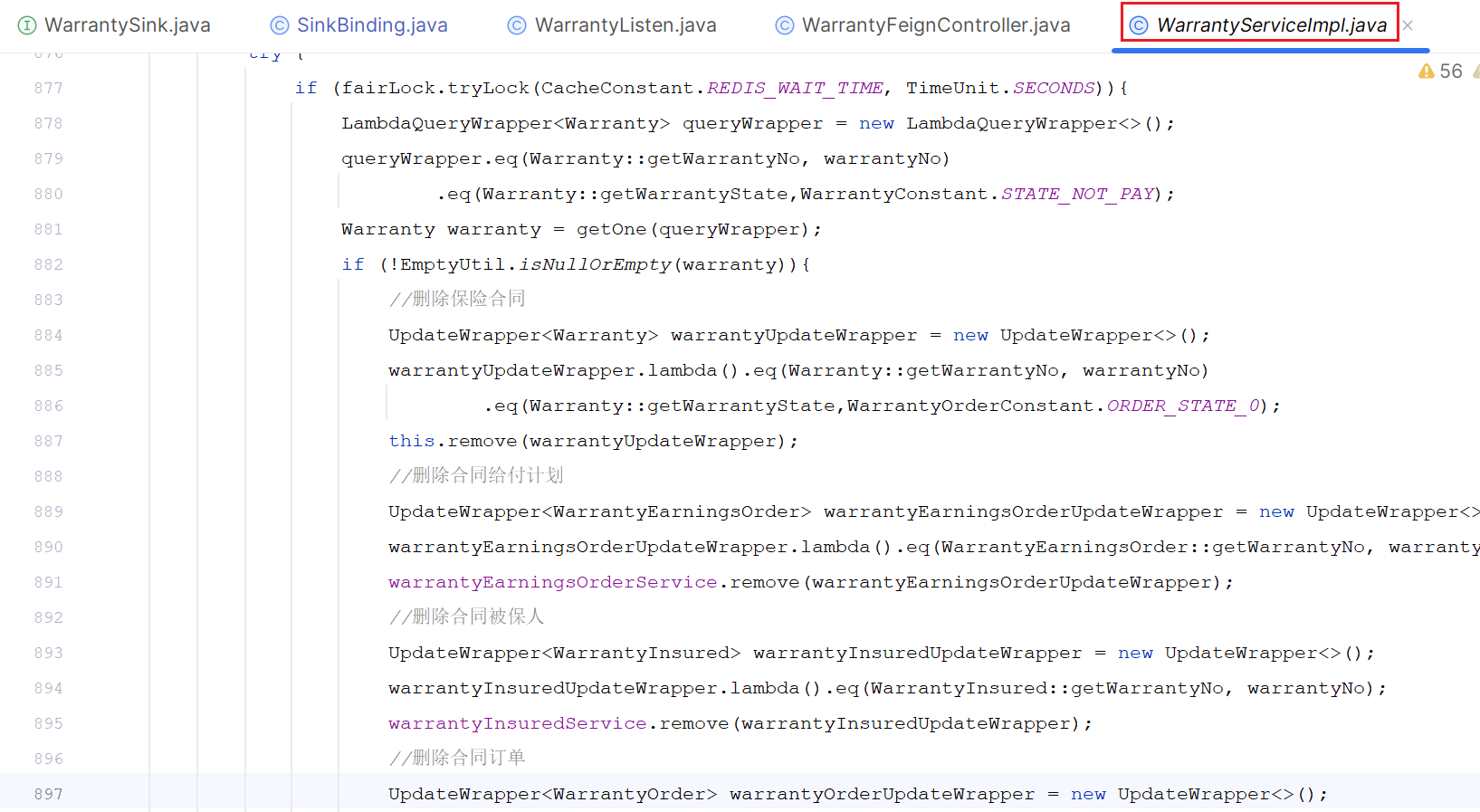
**3）编写消息监听器，接收消息**

在task-listener 编写消息监听器如下：



对应取消合同的业务逻辑：

* 判断当前合同的支付状态
* 如果未支付则删除保险合同、给付计划、被保人、合同订单、合同核保等



**3.4、扩展需求**

如果支付时间内未支付；需要给用户一个支付宽限时间2天；2天未支付才取消合同；应该怎么实现呢？说思路

**4、作业&问题**

* 合同是怎么处理的？上传、保存和展示
* 你使用过线程池吗？核心参数有哪些？哪些地方使用到了？
* 什么是宽限截止时间与复效截止时间
* 用户在投保时间过长情况下；会疯狂的刷新页面也就可能会出现生成多个保险合同的问题？该如何解决
* 在一个事务中使用了线程执行另外一个数据库操作的话；如果执行失败事务会回滚吗？
* 在项目中使用过分布式锁？分布式锁使用场景
* 取消合同的时候，有可能会收到多条消息（合同编号）
* 生成合同的时候，会发一个延迟消息30分钟；如果超时未支付的话；会收到至少6次消息；这样会多次取消合同；多次取消合同，会损耗服务器性能
* 所以在取消合同的代码中添加了分布式锁；使用的是redisson；（获取锁对象；redissonClient.getLock(合同编号）；redissonClient.tryLock() 加锁）
* 在取消完合同之后释放分布式锁 redissonClient.unlock(合同编号）
* 在项目使用过 try ... catch ... finally吗？
* 在取消合同的时候，避免分布式锁没有释放；所以使用了它；try 加锁， finally释放锁
* **简历职责-候选**：优化app端用户选定保险方案之后的投保业务；用户投保的操作从原来的2.6秒优化到900毫秒；提升 65% 的投保效率。
* **简历职责-候选**：配置RabbitMQ延迟插件，实现超时未支付的保险合同的取消业务；