消息管理系统

----概要设计

V1.00

|  |  |  |
| --- | --- | --- |
| **文件状态：**  **[√] 草稿**  **[ ] 正式发布**  **[ ] 正在修改** | **当前版本：** | V1.0 |
| **单 位：** | 北京市上品科技发展有限责任公司 |
| **作 者：** | 王檬 |
| **审 核 人：** | 张鹏 |
| **批 准：** | 于永利 |
| **部 门：** | 信息系统部 |
| **完成日期：** | 2012-11-13 |

1. 概述

随着公司IT业务的发展，原本集各种业务混杂与一体的单品系统，逐渐被小型化、业务独立的子系统替代。而在这些子系统的应用开发中，不论是基本的信息交互，还是深入的性能优化，都离不开基于消息机制的应用模型。比如目前系统中的价格&库存同步、检索引擎更新、第三方平台的商品信息同步，消息处理环节都扮演着核心的角色。

越来越多的消息处理，在提升系统性能和用户体验的同时，伴随的问题也随之出现，给整个it系统带来了新的挑战。

* **高可用**

处于核心的broker本身不够完善。就实际使用上看，开源的broker虽然性能优秀，但总会在高压的情况下丢失消息。而商用broker虽然可靠，但价格过于昂贵，引入代价高。所以最现实的办法是在开源broker的同时，补充通用的出错补偿逻辑。

* **监控与分析**

在网络、数据库发生异常的时候，broker会处于假死状态，假死的broker处理速度及其缓慢却不会触发HA的备机来替代。如果不及时回复，会影响整个应用系统的使用情况。然而开源broker不具备发现这种错误的能力。

* **规范性**

由于消息模式的引入是分阶段实施的，目前代码分散在各个子系统中。消息内容解析、处理时序、错误补偿均不相同，甚至有的还自己实现了监控。如果放任这种情况发展，那么上述问题即使有了解决的办法也无法实施。

为了支持公司IT业务的可持续发展，我们需要开发一个统一消息处理系统（MMS）平台，令该平台同时具备**平台无关性**、**统一规范性、高可用性、可监控性,**才能解决上述问题。

1. 系统设计
   1. 系统架构



* 1. 模块划分



* 1. 高可用（HA）

MMS处于各个子系统的核心位置，高可用的方案是否完善至关重要。

* + 1. 系统风险

从特征上看，mms系统面临着两类风险，既一般性风险和潜在风险。其中一般性风险主要就是单点失效，一般可以采用技术方法自动预防。

另一类是潜在风险，这些则是在长期使用中总结下来疑难问题，一般需要人工干预才能解决，列表如下：

* broker的pending bug：amq的消息出队处理存在bug，每几万条消息中，会有1-2条消息总是处于pending状态，只需重新发送对应消息即可规避。
* broker运行异常：amq在有些情况下会出现假死现象，特征是在listener看似运行正常的情况下队列不断积压，需要重启broker才能解决。
* listener假死：由于负责处理的listener的tomcat假死，导致消息无法出队，队列不断积压，需要重启listener才能解决。
* 消息处理API无法访问：最遭情况，由于设备故障原因，造成listener节点回调的消息处理API访问异常或超时。此时，消息队列不断积压，且大量消息的处理状态处于“异常”。这种情况，需要人为干预恢复系统。

针对一般性风险，mms采用技术成熟的高可用方案。而对于潜在风险则需要引入监控和报警，及时通知管理员进行人工干预，将问题带来的损害降到最低。

* + 1. 系统级HA
* 负载均衡HA

mms各系统节点使用tomcat集群+nginx组成无状态负载均衡集群，单节点容灾。

* 消息持久化

mms系统将每条消息保存到mysql数据库中，持久化消息的处理轨迹。

* ActiveMQ主从热备

利用2个MQ实例，确保单节点失效或重启时造成消息丢失。

* + 1. 系统监控与报警

潜在风险可以通过监视broker的队列积压情况，以及一段时间内发生的消息处理异常个数来判断，一旦超过阀值就通过短信或邮件来报警，令问题得到有效的控制。

broker的数据可以通过JMX技术获取到，消息处理异常则通过在消息的持久化记录中保存处理状态来辅助判断。

* 后台监控

由后台管理模块提供定时任务，检查每个队列的积压个数，当队列的积压个数超过阀值时，向管理员发送报警。另外长期处于积压状态的消息，采用重新发送的补偿机制。

* + 每10分钟读取一次broker，把每个队列中的关键属性保存到一个静态的List中
* 安全报警
  + pending报警，当broker与db链接异常，或者consumer处于无效时，队列会大量的pending。这时，需要人为干预重启服务器。每10分钟读取一次静态变量，读取每组pending的个数，凡是监控超过阀值就报警。
  + 处理异常报警，每2小时读取一次“重新发送”，“超过2小时”，“未处理”的消息报警。
* 消息补偿

pending消息补偿job，每2小时执行一次，将2小时以前的pending消息重发。（状态为“已发送”，“未处理”），原消息的补偿状态置为“1”。

处理异常消息补偿，每2小时执行一次，将2小时以前的pedingjob, 每10分钟执行一次，将10分钟以内消息处理状态为“异常”的消息读取出来，将消息重新发送，补偿状态置为“2”。

1. 模块设计
   1. 生产者代理

作为MMS系统的入口，负责定义接口，接收其他业务系统发来的消息。按照消息分组进行路由，选择合适的队列进行入队操作的模块。

* 1. 消费者代理

代理所有所有消息队列的监听的模块，将入队的消息进行出队操作，解析内容，并采用回调业务系统RESTfull接口的方式，完成消息的处理。并在处理过程中对消息的状态进行持久化和更新。

* 1. 后台终端

MMS系统管理员界面，可以动态创建队列，动态分配消费者；配置生产者代理节点、消费者代理与broker的关系；对消息按照所需的处理能力进行分组，规划消息的处理压力分配。同时提供节点的健康监控和报警，确保节点的高可用性。

* 1. Broker

基于消息队列模式的中间件，目前采用ActiveMQ来实现，以后根据需求可以更换成其他产品。采用哪种Broker并不影响MMS系统的接口，也就是说对于MMS系统的使用者来说是透明的。

1. 应用集成
   1. 定义消息

MMS负责代理消息的传输流程，对应的业务处理则由相关的系统进行开发，并封装成RESTfull风格的API接口，当MMS接收到消息后，会回调该API，并传递消息的内容作为参数。

所以定义一个消息需要确认如下信息：1.消息头2.消息内容 2.消息处理APIURL

* 消息头如下：

queueType：队列模式，目前有queue和topic两种，根据消息类型选则。

queueGroup：队列分组，根据mms管理员分配的分组而定，用于界定消息处理的压力大小。

* 消息体： 字符串，最好是合法的JSON字符串。
* 消息处理APIURL

由应用系统开发人员提供的消息处理API，作为消息实际处理的接口。

MMS系统最终会将消息体作为参数，调用该API完成消息的处理。在使用该接口发送消息以前，应用开发人员需要确保该接口是可用的。

* 1. 维护队列分组

可用的队列类型由MMS的管理员通过管理界面维护。系统初始化时，由管理员根据队列压力的大小将队列分为不同的“组”，并配置指定的队列和对应的监听器。队列类型一经创建，除非系统重大变化，否则不能修改。

集成消息系统的应用程序，跟剧消息的特征，选择不同的组来编写消息调用代码。

* 1. 编写发送消息代码

MMS管理员将MMS部署完毕后，发布MMS发送消息的URL地址，以及工具类jar包。

应用开发人员则利用工具类jar包提供的工具类，结合自身的消息定义，向指定的URL发送消息。

Sample：

**import**com.shopin.core.framework.exception.ShopinException;

**import**net.shopin.message.api.DefaultMessageClient;

**import**net.shopin.message.api.IMessageClient;

**import**net.shopin.message.request.RestMessageStatusRequest;

**import**net.shopin.message.request.SendMessageRequest;

**publicstaticvoid** main(String[] args){

String url="http://localhost:8080/MMS\_ProducerAgent1";

IMessageClient client=**new**DefaultMessageClient(url);

SendMessageRequest request=**new**SendMessageRequest();

request.setQueueType("queue");

request.setQueueGroup("fast");

request.setQueueBackURL("http://mysystem/mymodel/docallback.html");

request.setQueueContent("json");

**try** {

System.*out*.println(client.execute(request));

} **catch** (ShopinException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

* 1. 消息处理

MMS的消费者代理在收到消息后，会根据消息指定的处理URL，将消息内容作为参数访问该URL，如果收到返回值为success=true，则认定处理成功。如果返回值为其他值或抛出异常，则认定处理失败。MMS的消费者根据处理的状态，更新持久化的消息记录。注意，按目前MMS的协议，该RESTfull接口必须以.json结束。下面为一个例子代吗。

@RequestMapping(value = "/test/testCall", method = { RequestMethod.*GET*, RequestMethod.*POST* })

**publicvoid** test(Model m, Long recodeSid, String param) {

System.*out*.println("Callback successfull!" + " " + recodeSid + " " + param);

m.addAttribute("success", "true");

}

* 1. 消息处理时序

