密级：　　保密期限：

北京郵電大學

硕士学位论文

题目：基于 WS-Notification 的高性能、高可靠性发布/订阅系统

学　　号：2011111407

姓　　名：温 鹏

专　　业：通信与信息系统

导　　师：陈俊亮

学　　院：网络技术研究院

2013 年 12 月 24 日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

基于 WS-Notification 的高性能、高可靠性发布/订阅系统

摘　要

物联网服务是一种需要高实时性和广域分布的通信服务基础设施作为底层支撑的服务，而现有的企业服务总线（以下简称 ESB）无法提供这种特性的保障。另外，传统的 ESB 为服务提供的是请求/响应式的范式，不能满足物联网服务中大量事件异步交互的需求。因此，我们开发了一种基于发布/订阅范式的统一消息空间中间件，该中间件包含两层服务，底层是面向广域网络的分布式事件总线，上层是为不同物联网服务提供接口的本地接口层。我们把主要目光放在统一消息空间的接口层，即设计和实现接口服务，提出一套方法，构建一个高性能、高可靠性的发布订阅系统。

发布/订阅模型因其异步、多点通信，松耦合和易扩展的特点，被广泛应用于分布式环境中。本文在详细研究国内外发布/订阅系统现状的基础上，结合物联网应用环境，分析其中存在的问题和不足，针对主题组织形式、元数据存储与管理、对复杂网络环境的适应以及系统整体性能、稳定性的提升等方面提出了改进措施，并对改进前后的系统性能进行了分析比较，旨在解决发布/订阅系统实用化过程中遇到的一系列问题，提供一种可应用在复杂网络环境中的高性能发布/订阅系统服务。

关键词 发布订阅 WS-Notification 主题树 异步

AN EFFECTIVE AND RELIABLE PUBLISH/SUBSCRIBE SYSTEM BASED ON THE WS-NOTIFICATION

ABSTRACT

The Internet of Things (IoT) service is a kind of service which needs high real-time and wide-area distributed communication infrastructure as the underlying support. However, the existing enterprise service bus (ESB) cannot provide the guarantee of this feature. In addition, the request/response paradigm which traditional ESB provides cannot meet the needs of asynchronous interaction with plenty of events in the IoT service. Therefore, we developed a unified messaging space middleware based on the publish/subscribe paradigm. The middleware consists of two layers. The bottom is distributed event bus for wide area network, and the upper is the local interface layer providing interfaces to different IoT services. We focus on the interface layer and propose a series of measures to build a high-performance interface service.

Publish/subscribe paradigm is widely used in distributed environment for its characteristics of asynchronous, multicast communication, loose coupling and ease of extension. In the IoT environment, this paper analyzes the problems and shortcomings of existing works about publish/subscribe system, and proposes improvement measures for topic organization forms, metadata storage and management, the adaptability of the complex network environment, the improvement of the overall system performance and stability and so on. At the same time, this paper analyzes and compares the performance of the system before and after improvement. It aims to solve a series of problems encountered in the practical process of the publish/subscribe system, and provides a high performance publish/subscribe system interface service which can be used in complex network environment.

KEY WORDS publish/subscribe wsn topic tree asynchronization

目录

第一章 绪论

1.1 背景

Internet 技术的广泛应用和物联网的快速发展，对分布式系统的信息分发提出了更高的要求，传统的服务器与客户端之间“请求/应答”的消息传输模式已经不能很好地满足人们的需求。发布订阅系统技术作为一种信息交互和共享的中间件，在消息的生产者（发布者）和消费者（订阅者）之间提供了一种松耦合的信息分发手段，消息的生产者与消费者实现了脱离，可以更好地完成消息的交互。

学术界和工业界一直不间断地对发布订阅技术展开研究，但二者的角度不同。学术界看重如何增强发布订阅系统的表达能力，而工业界更关注发布订阅系统的服务质量（性能、可靠性等）。由此，诞生了各种各样的发布订阅系统，他们各有所长，比较典型的有：TIB/Rendezvous[1]，Gryphon[2]，SIENA[3]，JEDI[4]，Scribe[5]等。正是由于发布订阅技术的实现多种多样，在发布订阅技术兴起的早期，各个实现系统之间的处理逻辑和流程各不相同，给系统升级和整合带来很大的不便。鉴于这种情况，2006 年，OASIS 提出了 WS-Notification 规范（以下简称 WSN），WSN 是一系列规范的总称，旨在为 web 服务之间的事件驱动编程制定一套标准，目前，该规范已经得到了业界的一致认可，并广泛应用于各种 SOA 系统中。

Apache 基金会的 ServiceMix 项目提供了 WSN 规范的一种开源实现，作为 ServiceMix 的一个组件发布（以下简称 wsn），依赖于 ServiceMix 运行环境，对外提供发布订阅服务。wsn 组件实现了基本的发布订阅功能，但对订阅消息的主动推送和语义保序缺乏支持。在过去几年里，北京邮电大学网络服务基础研究中心在 wsn 组件的基础上开发出一个主动推送型发布/订阅系统，弥补了上述缺陷，使得该系统可以初步应用于实际的生产环境中。

随着物联网技术的发展，分布式环境中消息交互越来越频繁，对发布/订阅系统这类消息中间件的要求也越来越高，现有的主动推送型发布/订阅系统在功能、资源占用、订阅表达能力、元数据管理和消息分发性能、稳定性等方面已经不能满足应用需求。在这种情况下，我们对现有的主动推送型发布/订阅系统做了全方面的改进和优化，综合提出了一种基于 WS-Notification 的高性能、高可靠性的发布/订阅系统。

1.2 研究工作

通过对 wsn 组件以及在此基础上研发成功的主动推送型发布/订阅系统的深入研究，我们发现该系统在实际生产环境中部署时，仍然存在很多问题，不能很好地完成大规模、高实时、长期稳定的消息分发任务。首先，系统资源占用过多。wsn 组件的运行依赖于 ServiceMix 服务总线，ServiceMix 的运行会占用大量的 CPU 和内存资源，并且其中大部分组件对于发布订阅系统是无用的，导致大量资源浪费。其次，订阅者表达自己兴趣的能力弱。现有系统采用订阅表结构维护订阅信息，只保存简单主题与其订阅者之间的映射关系，无法实现层次化的订阅。再次，现有系统的性能太弱，消息吞吐量在 20~40 包/秒左右，完全不能满足需求。最后，系统长期运行稳定性方面，现有系统也存在缺陷，运行一段时间后会出现内存耗尽。

鉴于这种情况，本文针对上述问题提出了一套完整的解决方案，旨在对现有发布订阅系统进行改进，使之能够应用在复杂的实际环境中。

首先，本文实现了 wsn 组件与 ServiceMix 服务总线的剥离。为了减轻对系统资源的过度依赖，我们将 wsn 组件从 ServiceMix 总线体系中剥离出来，作为一个单独 jar 包进行部署，运行时对外发布 webservice 提供服务，大大减轻了发布订阅系统的资源占用，提高了其实用性和可扩展性。

其次，本文论述的系统完善了原有系统的功能。实现了包括取消订阅、处理重复订阅信息、增加可靠通知接口以及在接口层实现分包发送等在内的众多附加功能。

第三，为了增强订阅者的兴趣表达能力，我们采用树状结构存储订阅信息，实现了层次化的订阅。并且引入 OpenLDAP 数据库管理元数据并进行持久化，对用户提供图形界面，用于定义、修改、展示元数据信息。在发布订阅系统内部，与元数据实现高效交互，并设计相应的匹配算法。

第四，改进系统性能，提高消息吞吐量。经过分析和研究，我们认为基于 wsn 组件的发布订阅系统中，其性能的瓶颈主要在于两个地方。其一，订阅者和发布者将消息包装成 SOAP 消息发送给发布订阅系统以及发布订阅系统将通知消息发送到订阅者，这都是基于 HTTP 协议的消息传输，在这个过程中使用了 Apache HttpClient 模块，并采用同步方式发送消息；其二，系统收到消息后，为 XML 格式的消息建立 DOM 树进行处理，消耗大量时间。有鉴于此，本文论述的系统首先采用异步 HTTP 结合回调机制的方式发送消息，另外在系统内部建立一套字符串处理机制完成消息的处理，在不损失 XML 消息描述能力和可扩展性的同时实现更快速的消息处理。

最后，我们将提出的理论模型与实际的应用环境相结合，开发出一套实用的高性能的发布订阅消息中间件系统，并在实际环境中进行了大量的测试，尤其针对系统长时间稳定运行的能力进行了反复的测试和调优，从而使得我们的发布订阅系统能够真正应用于实际的生产环境中。

1.3 论文组织结构

由以上研究工作可知，本文在基于 wsn 组件的主动推送型发布订阅系统的基础上，提出了一种高性能、高可靠性的发布订阅中间件系统。本文的结构如下：

第一章为绪论，简要介绍本文研究的背景和主要内容；

第二章为相关技术的概述，主要介绍在优化发布订阅系统的过程中涉及到的关键技术，并总结发布订阅系统研究的现状；

第三章为系统需求分析，主要阐述了在现有系统的基础上，从哪些方面入手，去构建一个真正可用的高性能、高可靠性发布订阅系统；

第四章为系统概要设计，主要在逻辑上描述如何实现高性能和高可靠性的发布订阅系统；

第五章为系统的详细设计与实现，主要讲述如何综合利用各种技术构建高性能、高可靠性的发布订阅体系；

第六章为系统的测试，对测试的用例进行说明，并对测试的结果进行分析，得出结论；

第七章为总结与展望，对系统现状进行总结，并对系统进一步的优化工作提出意见。

1.4 本章总结

本章简要说明了发布订阅系统模型的发展以及本文研究的主要内容，并且立足于目前发布订阅系统的缺陷及其解决方案，确立本文论述的主题，立题立意。同时，本章描述了本文的结构层次，方便读者更好的理解系统的设计与实现过程。

第二章　相关技术概述

由第一章的内容可知，本文论述的发布订阅系统是基于 web 环境的，在系统内部则采用 JMS 进行通信，为了优化现有的发布订阅系统，我们引入了 OpenLDAP 数据库对订阅信息进行管理和持久化，另外还应用了 HTTP 异步机制和回调技术。为了使读者更加深刻地理解系统的工作原理，本章对本文涉及的关键技术进行简要介绍。

2.1 Web Service 技术概述

2.1.1 Web Service

Web Service 是一个软件系统，能够使得运行在网络中不同机器上的不同应用无需借助第三方软件或硬件，就能够实现数据交换。简单来说，Web Service 就是为应用程序提供一个网络接口，其他应用程序（可以是桌面应用程序或者网络应用程序）通过这个接口来调用发布者提供的服务，一般是由客户端发送数据，服务器端接收数据并作出相应处理以后返回给客户端，以此来达到数据交互和分布式处理的目的。Web Service 是一系列技术的总称，其中常用的有：

XML：可扩展标记语言，描述结构化数据的标准方法；

SOAP：简单对象访问协议，一种轻量级的基于 XML 的协议，用于在 Web 上交换结构化的数据；

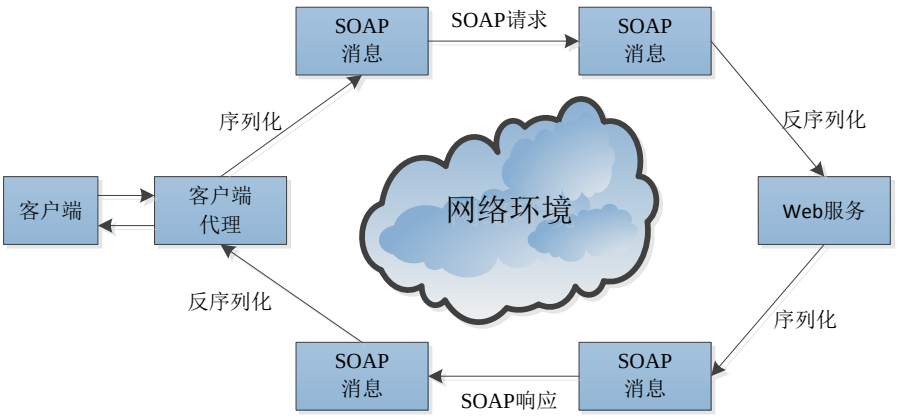

图 2-1 基于 XML 的 Web Service 架构

WSDL：Web Service 描述语言，用来描述 Web 服务和说明如何与 Web 服务通信的 XML 语言，为用户提供详细的接口说明书。

图 2-1 是一个典型的基于 XML 语言的 Web Service 系统。

2.1.2 服务

一个 Web Service，从某种意义上来说，就是一个应用程序，它向外界暴露一个能够通过 Web 调用的 API，使得别的应用程序可以通过编程的方法来调用这个应用程序。它分为两部分，一部分是接口，用于定义自身能够提供的服务，另一部分是实现，定义具体的处理逻辑。对于某个特定的 Web Service 来说，接口通常是不会变的，但是实现是不确定的，可能在不同时间段内具有不同的实现。

2.1.3 请求者和提供者

请求者和提供者是 Web Service 架构中的两个实体，位于数据交换过程的两端。提供者定义接口并完成实现，请求者根据服务者描述的接口调用相应的服务，发送数据，提供者在服务端完成数据处理后，将处理结果返回给请求者，完成数据交换。

举例说明，你想创建一个 Web Service，它的作用是提供天气查询服务，你可以定义一个接口，接受城市名字符串和日期作为参数，然后返回一个以逗号隔开的字符串，包含了相应日期的气温和天气情况。在服务端的实现中定义处理逻辑，比如向雅虎的天气服务器查询数据，得到数据后构建字符串，返回给调用方。在请求者一方，可以通过查看提供者的接口描述，调用服务，传输符合要求的参数给提供者，提供者完成处理逻辑后返回结果，至此一次数据交互就完成了。

2.1.4 服务描述

假设你写好了你的 Web Service 服务，你怎样向别人介绍你的 Web Service 有什么功能，以及每个函数调用时的参数呢？这就需要定义一种机器能阅读的描述文档。WSDL 就是这样一个基于 XML 的语言，用于描述 Web Service 及其函数、参数和返回值。

由于是基于 XML 的，所以 WSDL 即是机器可读的，又是人可读的，甚至一些工具或者框架可以根据你的 Web Service 生成 WSDL 文档，或者根据 WSDL 文档生成相应的 Web Service 代码。

2.2 JMS 技术概述

2.2.1 JMS 技术简介

JMS(Java Message Service)，即 Java 消息服务应用程序接口，是一个 Java 平台中面向消息中间件的 API，用于在两个应用程序之间，或分布式系统中发布消

息，进行异步通信。Java 消息服务是一个与平台无关的 API，绝大多数消息中间件系统都对 JMS 提供支持，wsn 也不例外。

JMS 在 1999 年由 Sun Microsystems 领衔开发。这种基于消息传送的异步处理模型，具有非阻塞的调用特性。发送者将消息发送给消息服务器，服务器会在合适的时候再将消息转发给接收者；发送和接收采用异步方式，这就意味着发送者无须等待，发送者和接受者的生命周期也无须相同，而且发送者还可以将消息传给多个接收者。这样一来，这种异步处理方式就大大提升了应用程序的健壮性、性能和可伸缩性，使数据集成和系统整合工作变得易如反掌，特别是在分布式应用上让同步处理模式望尘莫及。

wsn 组件在底层也是使用 JMS 进行异步通信的。本文论述的系统中，将 wsn 组件从 ServiceMix 总线中剥离出来，但是仍然需要依赖 JMS 实现底层消息传输。JMS 是一套规范，我们选择了 JMS 的其中一种开源实现：ActiveMQ。

2.2.2 一种 JMS 规范的实现——ActiveMQ

ActiveMQ 是一个实现了 JMS1.1 规范的开源消息中间件系统，为企业提供高可用、高性能、易扩展并且安全的消息系统集成方案。ActiveMQ 具备以下典型特性：

- 支持多种语言和协议编写客户端；
- 完全支持 JMS1.1 和 J2EE1.4；

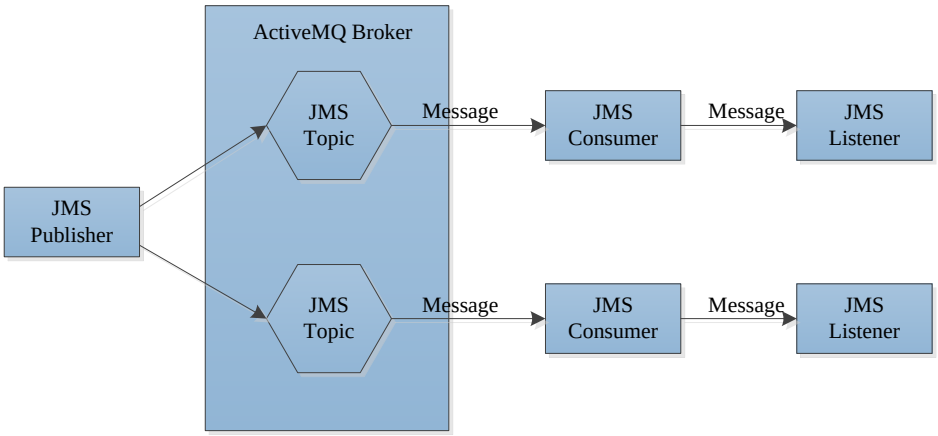

图 2-2　JMS 消息传输模型

- 对 spring 的良好支持，ActiveMQ 可以很容易地内嵌到使用 spring 的系统里面去，并且支持 spring2.0 的特性；
- 支持多种传输协议；
- 支持通过 JDBC 和 journal 提供高速的消息持久化；
- 支持与 Axis 的整合。

图 2-2 描述了典型的 JMS 消息传输模型。

2.3 LDAP 技术概述

2.3.1 LDAP

LDAP（轻量级目录访问协议，Lightweight Directory Access Protocol)是实现提供被称为目录服务的信息服务。目录服务是一种特殊的数据库系统，其专门针对读取、浏览和搜索操作进行了特定的优化。目录一般用来包含描述性的，基于属性的信息并支持精细复杂的过滤能力。目录一般不支持通用数据库针对大量更新操作需要的复杂的事务管理或回卷策略。而目录服务的更新一般都非常简单。这种目录可以存储包括个人信息、web 链接、jpeg 图像等各种信息。为了访问存储在目录中的信息，就需要使用运行在 TCP/IP 之上的访问协议—LDAP。

LDAP 目录中的信息按照树型结构组织，具体信息存储在名为条目(entry)的数据结构中，条目就相当于关系数据库中表的记录，由属性的一个聚集组成，并由一个唯一性的名字引用，即专有名称（distinguished name, DN）。从概念上说，LDAP 分成了 DN、OU 等，OU 就是一棵树，DN 就可以理解为是叶子，叶子还可以有更小的叶子。

以电话薄为例，我们用电话薄的目的是为了查询某个公司的电话，在这个电话薄中附带了一些这个公司的基本信息，比如地址、经营范围、联系方式等。电话薄的组织结构是由一条一条的信息组成，信息按照行业、类别进行了分类。每条记录都分成了若干的区域，其中涵盖了我们要的信息，这就是一个目录，一个树状结构，每个叶子都是由一条一条的分成若干区域的记录。

LDAP 目录与普通数据库的主要不同之处在于数据的组织方式，它是一种有层次的树形结构，因此也非常方便使用树形结构展示。根据我们的需求，我们选择 OpenLDAP——LDAP 协议的一种开源实现作为我们的元数据管理仓库。

2.3.2 OpenLDAP

OpenLDAP 是 LDAP 协议的自由和开源的实现，目前已经被包含在众多流行的 linux 发行版中，它主要包括以下几个部分：

- slapd – 独立 LDAP 守护服务；
- slurpd – 独立的 LDAP 更新复制守护服务；
- 实现 LDAP 协议的库；
- 工具软件和示例客户端。

2.4 异步 HTTP 机制

异步的概念其实是相对同步来讲的。对于一个同步过程，调用产生后，调用

方的线程被阻塞，等待被调用方处理完请求后返回，然后接着执行下一个调用；但异步机制不同，当一个异步过程调用发出后，调用者不能立刻得到结果，调用方线程进一步向下运行，等被调用方处理完请求后，通过状态、通知或者回调来通知调用者。由于调用方线程不用等待被调用方返回结果，因此异步机制在性能上相对同步来说有一定的优势。

Apache Httpclient 项目对 HTTP 协议进行了很好的封装，其中也同时提供了同步 HttpClient 和异步 HttpClient。使用同步 HttpClient 时，每发送出一条 SOAP 消息，需等待并读取 Response 响应消息；而使用异步 HttpClient 时，则无需读取返回，而是提供回调函数供接收端调用，并通过回调函数返回状态信息。

2.5 回调技术

软件模块之间总是存在着一定的接口，从调用方式上，可以把他们分为三类：同步调用、回调和异步调用。同步调用是一种阻塞式调用，调用方要等待对方执行完毕才返回，它是一种单向调用；回调是一种双向调用模式，也就是说，被调用方在接口被调用时也会调用对方的接口；异步调用是一种类似消息或事件的机制，不过它的调用方向刚好相反，接口的服务在收到某种讯息或发生某种事件时，会主动通知客户方（即调用客户方的接口）。回调和异步调用的关系非常紧密，通常我们使用回调来实现异步消息的注册，通过异步调用来实现消息的通知。

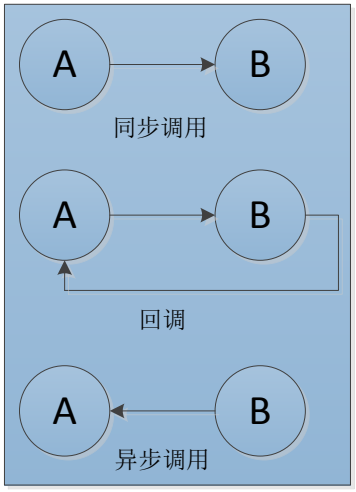

图 2-3 调用方式

对于不同类型的语言（如结构化语言和对象语言）、平台（Win32、JDK）或构架（CORBA、DCOM、WebService），客户和服务的交互除了同步方式以外，都需要具备一定的异步通知机制，让服务方（或接口提供方）在某些情况下能够主动通知客户，而回调是实现异步的一个最简捷的途径。

对于一般的结构化语言，可以通过回调函数来实现回调。回调函数也是一个函数或过程，不过它是一个由调用方自己实现，供被调用方使用的特殊函数。

在面向对象的语言中，回调往往是通过接口或抽象类来实现的，我们把实现这种接口的类称为回调类，回调类的对象称为回调对象。

2.6 发布订阅系统研究现状

发布订阅模型由提出至今，大量的组织和机构对其展开了深入的研究，其中不乏优秀的产品。但是，由于基于 Web Service 的发布订阅系统的标准于 2006 年才被 OASIS 提出，尚未被广泛的研究。但是随着 Web Service 应用渐成主流，基于 Web Service 发布订阅系统的重要性也日渐显露。现在主流的 Web Service 发布订阅系统有两个：一个是 IBM WebSphere 平台中 WS 发布订阅系统的实现，另一个是 Apache 开源软件基金会主持开发的 ServiceMix 平台中的 wsn 组件。由于 WebShpere 并不开源，无法对其内部特性开展研究。

主动推送型发布订阅系统就是基于 wsn 组件研发出来的支持主动推送和语义保序的发布订阅系统，但是其性能较低，系统的消息吞吐量很小，功能上也略显贫乏，不能够适应真实生产环境中的复杂情况。

综上所述，现阶段主流的发布订阅系统在功能和性能上都还有缺陷，提升的空间还很大。本文基于这种情况，提出一套解决方案，解决发布订阅模型部署应用道路上的一些障碍，完善功能，提高性能和稳定性，对于实际运用发布订阅系统的用户来说有很高的参考价值和使用价值。

2.7 本章总结

本章就实现该系统所采用的关键技术做一个概要性的说明。其中 Web Service 技术是最关键的技术基础，JMS 保证底层消息传输，OpenLDAP 实现元数据的管理和持久化，异步机制结合回调提高系统性能。对于以上技术的在本系统的详细应用，将在后续章节详细说明。

第三章 需求分析

由第一章绪论和第二章相关技术概述的内容可知，本文的目的在于在 Apache ServiceMix 提供的 wsn 组件的基础上，通过一系列创新性的改进措施和优化设计，打造一个高性能、高稳定性的发布订阅消息中间件系统。该系统不仅支持主动推送和语义保序，而且通过实现树状名称空间丰富了订阅者表达兴趣的能力，综合利用异步和回调技术大幅提升了原有系统的性能和对复杂网络环境的适应能力，并且可长期稳定运行。本章目的在于确定需求，以指导系统设计和实现。

3.1 功能性需求

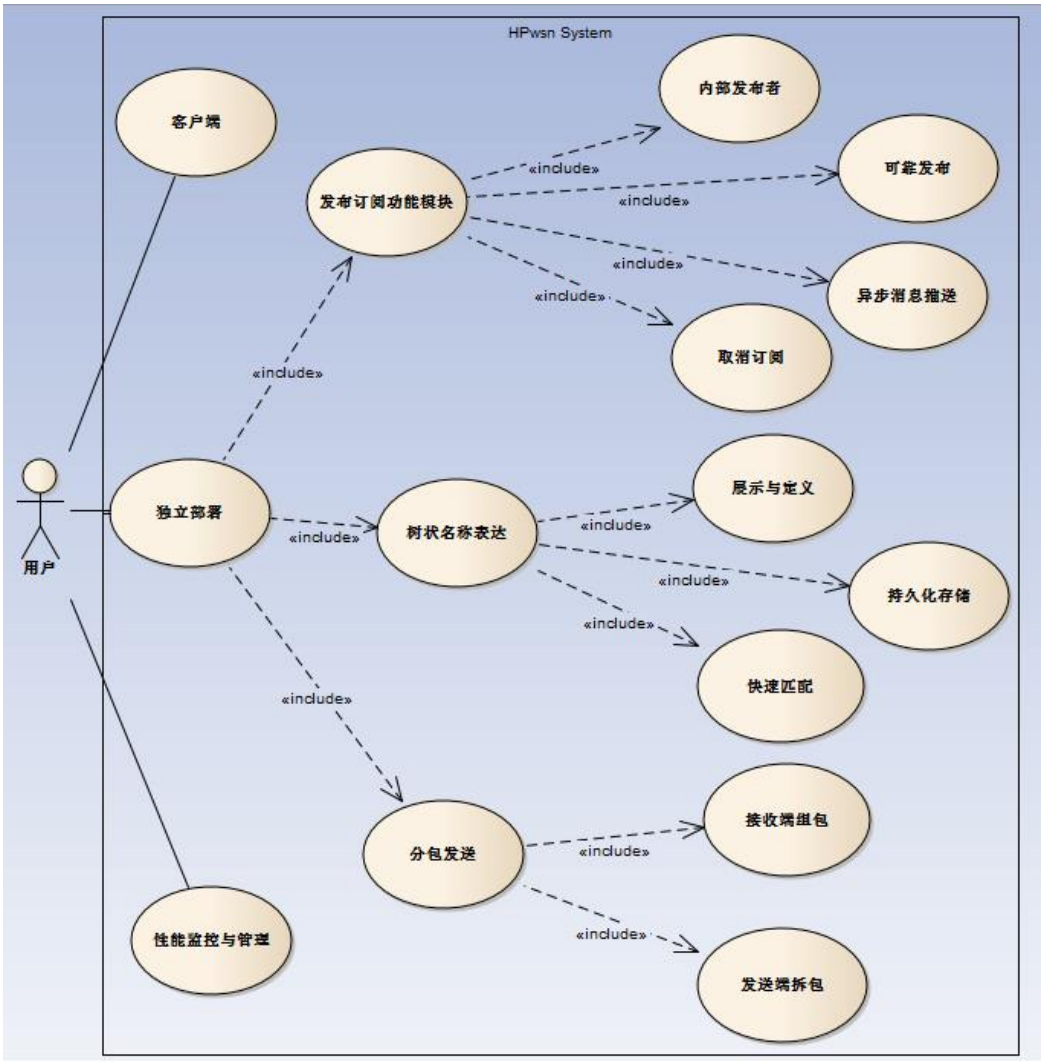

图 3-1 HPwsn 系统用例图

本文论述的发布订阅系统（以下简称 HPwsn, High-Performance wsn），其核心需求在于高性能和高可靠性，所有的设计都是围绕这两个核心点进行的。图 3-1 描述了 HPwsn 系统的 Use Case 图，下面解释一下其中的各个需求模块。

3.1.1 客户端

HPwsn 系统是基于分布式 Web 环境的发布订阅系统，用户与系统之间的消息交互使用 SOAP 协议，对用户来说，将自身数据包装成 SOAP 消息并发送到系统中是一个繁琐的过程，不仅要了解 SOAP 协议规范，而且还要明确地知道系统端暴露的 Web Service 接口。

因此，我们需要系统提供客户端模块，将用户从这些繁琐的工作中解放出来，客户端应该具备以下功能：

- 提供 createPullPoint、subscribe、notify、unsubscribe 等方法，使得用户直接调用这些方法就可以完成 SOAP 消息的封装；
- 与系统端约定 Web Service 接口的调用方式；
- 调用系统端 Web Service，将封装好的用户数据高效地发送到 HPwsn 系统中。

系统应以 jar 包的形式对外发布客户端模块，发布者和订阅者程序引入该包，就可以与 HPwsn 系统实现便捷、高效的数据交互。

3.1.2 性能监控与管理

要构建一个可长期稳定运行的发布订阅系统，无论是在开发测试阶段还是部署运行阶段，性能监控与管理模块都是必不可少的。通过性能监控管理模块，我们应该能得到系统在运行时的关键数据，包括 CPU、内存等资源占用情况，以及运行期间内存、线程的变化情况。

性能监控与管理模块要能够方便、直观地获取这些数据，以便评价系统的稳定性和长时间运行能力。另外，由于 HPwsn 是一个基于分布式 Web 环境的系统，性能监控与管理不仅局限于本地程序的监控，更要具备监控运行在远程主机中程序的能力。

3.1.3 独立部署

目前，基于 wsn 组件的发布订阅系统都是运行在 ServiceMix 总线中的，我们在实际生产环境中部署时发现，ServiceMix 服务总线的运行需要消耗大量的系统资源，空跑时 ServiceMix 总线就占用了 150M 以上的内存，并且其中大部分组件对于发布订阅系统的运行是毫无作用的。若系统运行在大型服务器上，集成多

种多样的服务组件，这种方式当然没有问题，但是在现在的物联网应用环境下，对于运行在网络末端的 PC 机，负载就显得很重了。因此，将发布订阅系统从 ServiceMix 架构中剥离出来，减少其资源消耗，拓展应用场景，是一项有价值的工作，也是 HPwsn 系统的基本需求之一。

3.1.4 完善发布订阅功能

现有的基于 wsn 组件开发出来的发布订阅产品中，功能还有待完善。经过对现有发布订阅系统的认真研究，结合实践中总结的经验，我们认为 HPwsn 发布订阅系统应该在以下几个方面完善功能：

1、取消订阅。现有的发布订阅系统，以主动推送型发布订阅系统为例，并没有从本质上实现取消订阅功能，即在取消订阅时没有彻底地释放创建一个订阅时分配的所有资源，这直接导致了随着系统中订阅数量的增加，资源消耗越来越多，废弃的资源得不到合理回收，进而系统不能长期稳定地运行。因此，应该在 HPwsn 系统中实现完整的取消订阅功能，采用合理的机制回收系统的废弃资源。

2、处理重复订阅。对于重复订阅的情况，系统不应再次分配订阅资源。

3、在系统内部设置内部发布者。当同时有很多通知消息到达系统中时，通知消息核心处理模块的压力是比较大的，设置内部发布者，在通知消息到来时首先进行一步处理，然后创建 JMS 消息通道，将消息发送给通知消息核心处理模块，有利于缓解系统压力，平衡处理时间，从而消除瓶颈。

4、提供可靠通知发布方式。在路由层转发消息时会根据源主机与目的主机之间的网络拓扑关系选择不同的协议实现数据转发，若使用 UDP 协议则不能保证消息一定到达接收端。对用户来说，某些通知消息可能很重要，希望保证交付。鉴于这种需要，系统应该能够提供一种方式，允许用户干预路由层转发策略，实现消息的可靠转发。

5、消息异步递交。高性能地实现消息分发是 HPwsn 系统最核心的需求，采用异步方式代替同步消息递交，能够大幅度提升系统性能。

3.1.5 树状名称表达

学术界和工程界分别从不同角度研究发布订阅技术，学术界更注重系统的表达能力，提出了很多基于内容的匹配算法；而工程界更注重系统的服务质量（性能、可靠性等）。本文论述的系统旨在从工程的角度提高发布订阅系统的实用性，但是，基于简单主题的匹配策略还是不能够满足用户日益多样化的订阅需求，因此，采用更优的数据结构维护订阅信息，丰富用户的表达能力，实现层次化订阅也是 HPwsn 系统的核心需求之一。

3.1.6 分包发送

3.1.4 节中提到，路由层转发消息时会根据源主机与目的主机之间的网络拓扑关系选择不同的协议实现数据转发，在使用 UDP 协议时，受限于 UDP 协议本身以及系统缓冲区的大小，每个数据包不能超过 8k。这就限制了用户向系统递交的数据的大小，为了克服这个问题，需要在服务层与路由层的接口处实现消息拆包和组包。

3.2 非功能性需求

除了 3.1 节中提到的功能性需求外，高性能和高可靠性也是 HPwsn 系统最核心的需求。

3.2.1 高性能

现有的基于 wsn 组件开发出的发布订阅系统的性能偏弱。测试结果表明，主动推送型发布订阅系统每秒钟的消息吞吐量大概在 20~40 包左右，这显然不能满足实际应用的需求，因此，HPwsn 系统的一个主要的目标就是综合运用各种技术，提升系统性能。

3.2.2 高可靠性

系统投入生产环境中时，能不能长期稳定地运行，也是衡量一个系统优劣的关键技术指标。由前两章的内容可知，原有系统由于这样那样的缺陷，并不能做到长时间稳定运行，对系统进行反复测试和调优，增强系统长时间稳定运行的能力也是 HPwsn 系统的核心目标之一。

3.3 系统应用环境

基于 wsn 组件的高性能、高可靠性发布订阅系统运行在 Web 环境下。一个典型的应用环境如图 3-2 所示。

HPwsn 系统可以进行灵活的部署和应用，由于具备高性能、高可靠性的特点，如果应用环境比较简单，可以在一台机器上部署，所有发布者和订阅者均与单点系统交互数据。这种部署方式负载有限，但是监控管理简单。

在分布式的 web 环境中，在多台机器上同时部署 HPwsn，这些 HPwsn 互相之间聚合成不同的集群，每个集群选出一个代表实现集群间的数据交互，路由模块负责不同 HPwsn 主机之间的数据转发，并由管理员程序统一管理。这种部署方式构成了一个完全分布式的消息中间件集群，优点是能够承载较大的负荷，向更多的用户（订阅者和发布者）提供消息分发服务，缺点就是集群拓扑结构复杂，

监控管理上有一定的复杂性。

原有的 wsn 组件由于运行在 ServiceMix 服务总线中，对机器的性能各方面

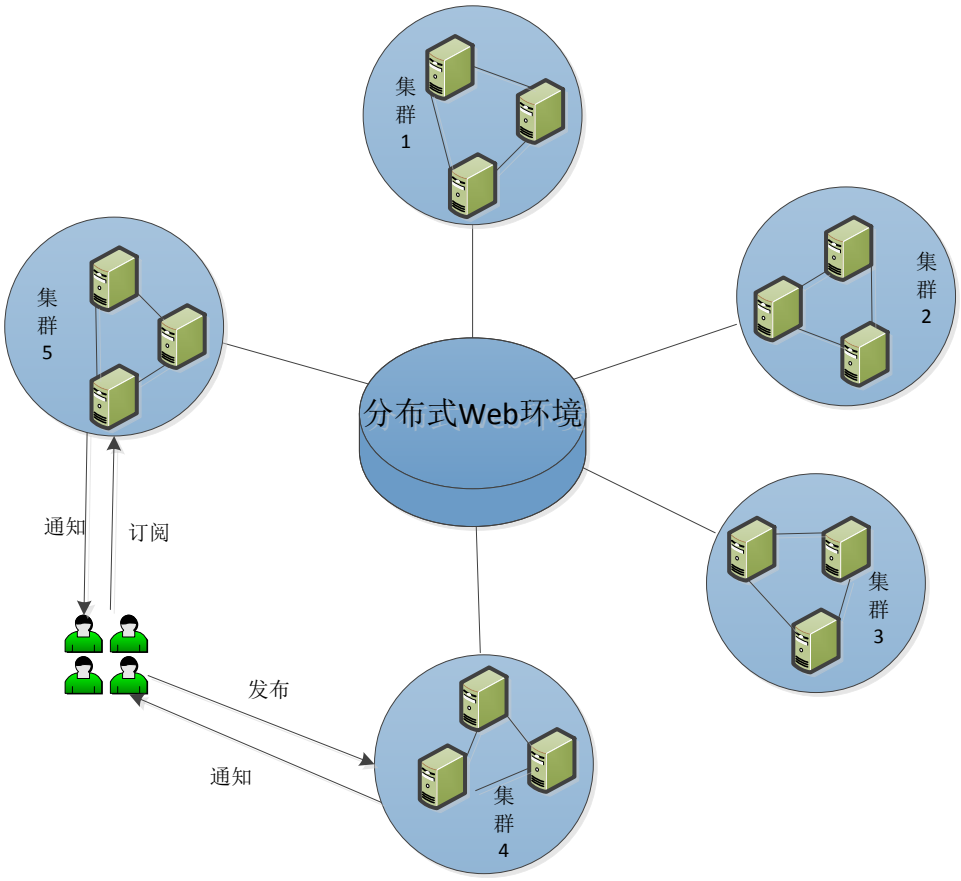

图 3-2 发布订阅平台应用环境示意图

要求较高，不适宜部署在普通 PC 机或者更低端的设备上，但是对于 HPwsn 系统来说就没有这个问题，HPwsn 是独立部署、运行的程序，对系统资源的占用不大，特别适合部署在运行于网络末端的廉价计算设备上。因此具有更灵活的部署方式和更强的适应性，尤其适合复杂的物联网应用环境。

3.4 本章总结

本章从现有发布订阅系统的缺陷，以及实践中总结的问题出发，详细阐述了高性能、高可靠性发布订阅系统的设计需求。在功能性需求方面，主要包括客户端、性能监控与管理、独立部署以及在此基础上的发布订阅功能完善、树状名称表达和分包发送；在非功能性需求方面，包括性能的提升和系统长期稳定运行的能力。

第四章 系统概要设计

第三章详细描述了我们设计实现 HPwsn 系统的需求和目标。那么如何构建一个高性能和高可靠性的发布订阅系统就是本章要阐述的内容。本章从系统架构出发，以需求为导向，详细阐述每一个功能模块的设计。

4.1 系统架构

一个通用的发布订阅系统架构图如图 4-1 所示：

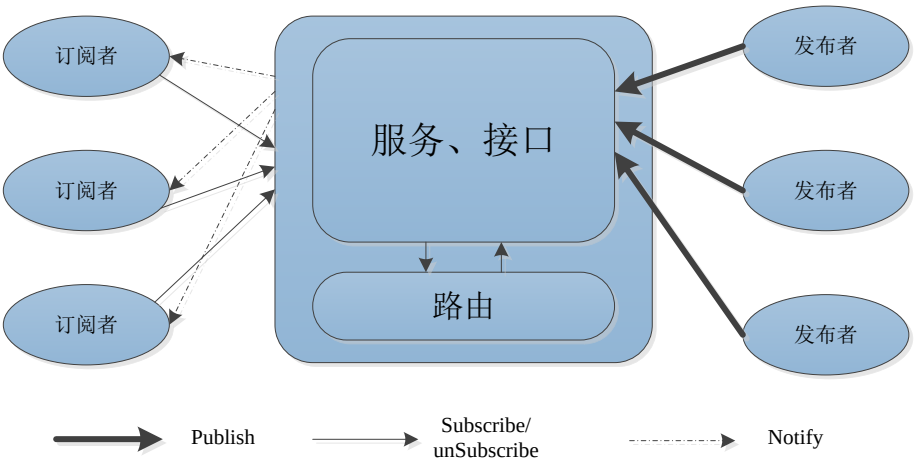

图 4-1 发布订阅系统架构图

发布/订阅系统是一种能够使分布式系统中的各方，以发布者和订阅者的身份参与到消息交互活动中来的中间件系统。如图 4-1 所示，发布者是消息的生产者，它将真实或虚拟环境中发生的事件封装成规范的格式化消息，发送到发布/订阅系统中；订阅者是消息的消费者，它向发布/订阅系统发送一条格式化的订阅消息，描述一个订阅条件，表示对哪些事件感兴趣，这样，当匹配这些订阅条件的消息到达系统中时，发布/订阅系统保证将这些消息及时、高效、可靠地推送给订阅者。

具体到本文论述的 HPwsn 系统，图 4-2 描述的是 HPwsn 发布订阅系统架构。从图中可以看出，HPwsn 发布订阅系统共分为三层，最底层是系统运行环境，包括操作系统、网络环境以及 Java 运行环境，为系统运行提供最基础的软件环境支持。第二层为路由层，路由层用于在不同的 HPwsn 机器之间转发数据，使得我们的系统能够运行在分布式 Web 环境中。最上层是服务层，分为用户接口和发布订阅服务两个模块。上述三层结构构成了 HPwsn 发布订阅系统的主体，另外还有元数据管理与持久化模块，用户客户端模块以及性能监控与管理模块，

他们结合在一起组成了 HPwsn 发布订阅系统。下面就每一层的设计进行详细说明。

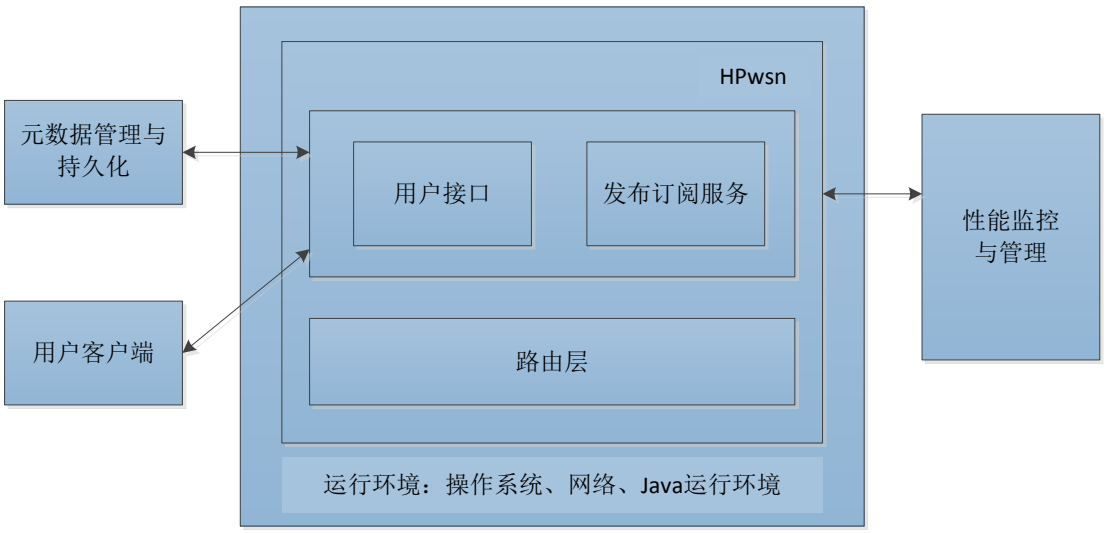

图 4-2　HPwsn 发布订阅系统架构图

4.1.1 运行环境

运行环境是系统运行最基本的保障，为系统运行提供最基础的软件环境支持，主要包括三个部分：操作系统、网络环境和 Java 运行环境。

本系统采用 Java 语言编写，Java 语言的跨平台特性在 HPwsn 发布订阅系统中也得以体现，本系统除了元数据管理与持久化模块之外，都可以运行在 windows 系列系统中，也可以运行在主流的 linux 发行版本上，但是由于图形界面的原因，元数据管理与持久化模块只能运行在 windows 操作系统上。不过，在部署 HPwsn 集群时，该模块只需部署在集群中的一台主机上，其它机器可以通过配置指向这台主机，获取元数据信息。

虽然本文论述的系统可以运行在主流 linux 发行版本上，但是截止到本论文截稿前，并未在 linux 平台上进行大量测试，因此，本文后面的叙述还是基于 windows 平台。

由于 HPwsn 通常以集群形式部署在 Web 环境中，所以网络环境的支持是系统正常运行的基本保障之一，并且网络类型的选择和网络质量的好坏直接关系到系统的服务质量。因此，系统部署之前网络环境的选择是重要的准备工作，在本文范围内，默认使用以太网连接各主机，广域网环境使用 Internet。

HPwsn 发布订阅系统绝大部分程序使用 Java 语言开发，因此，必须运行在有 Java 运行环境的机器上，建议配置 sun 公司的 Java 环境，JDK 版本在 1.6 或以上。另外，系统的性能检测和管理模块使用了 JDK 环境自带的工具，这也是部署时必须具备 Java 运行环境的原因之一。

4.1.2 路由层

路由层，为整个系统提供高效的分布式数据转发。路由层是一个独立的模块，由另外的同学负责开发，在此不予详述，但是有两点需要说明一下。

第一点是路由层与服务层的数据接口，到达服务层的每一条消息最终都会递交给路由层，由路由层决定处理策略，例如当路由层从服务层拿到一条通知消息时，会判断在网络中的其他 HPwsn 系统中是否有相应的订阅，若没有则丢弃消息，若有则转发消息。数据到达目的主机后，会由路由层向服务层递交，由服务层负责将数据递交给用户（订阅者）。

第二点，路由层在转发消息时，根据源地址和目的地址的拓扑关系，会选择不同的传输层协议（TCP 或者 UDP）。在使用 UDP 协议传输数据时，受限于操作系统的套接字缓冲区大小，UDP 包一般不能超过 8k。为了解决大数据包的传输问题，我们在服务层实现了分包发送和组包策略，使得用户在构建数据包时能够只关注数据本身，而从数据发送的细节中解放出来，另外，还使得路由层在传输数据时不必考虑数据包的大小。

4.1.3 服务层

服务层分为两部分，用户接口模块和发布订阅服务模块。其中用户接口其实是一个 Web Service 及其实现，供发布者或订阅者调用，而发布订阅服务则相当于是后台处理逻辑，用于完成发布订阅功能。

HPwsn 系统启动时将自身发布成一个 Web Service，用户通过调用 Web Service 接口，将封装好的数据发送到服务层，发布订阅服务接收到消息后，经过一定的流程，将消息交给对应的实现处理。这一部分是 HPwsn 发布订阅系统中最核心的部分，下面的小节中我们将展开详细叙述。

4.1.4 外围模块

除了上面介绍的三部分，客户端模块、元数据管理与持久化模块以及性能监控与管理模块，同样也是 HPwsn 系统的重要组成部分。

4.2 术语与概念

1、事件（Event）：

在 HPwsn 发布订阅系统中，一个事件，通常是指真实或者虚拟世界中发生的任何一件可以描述的事情，它可以是一些资源的状态及其变化，也可以是一些行为或者行为引起的结果。对于发布订阅系统来说，用户会将事件进行某种形式的描述，格式化成规范的通知消息发送到系统中来，但是，系统不会关注具体事

件到通知消息的映射，更加不会限定具体事件的范围，这在一定意义上体现了发布订阅模型的普适性。

2、通知消息（Notification）：

通知消息，是由具体实体创建的，一种格式化的对某种或某些事件的表述。这些事件可能被另外的具体实体感兴趣，那么通过发布订阅系统可以建立这两个具体实体之间的联系。通知消息使用 XML 语言描述，除了一些实现消息传输和处理必须定义的内容外，实体在创建通知消息时还可以自定义其他必要的信息。

3、发布者（Publisher）：

发布者是 HPwsn 系统的直接用户之一，除此之外还有订阅者。发布者，也就是通知消息的生产者，他们感知真实或虚拟世界中发生的事件，采用合理的方式对其进行描述，生成通知消息。然后通过调用 HPwsn 系统服务层的 Web Service 将通知消息发布到 HPwsn 系统中。

4、订阅者（Subscriber）：

订阅者也是 HPwsn 系统的直接用户之一。所有一切向 HPwsn 系统发起订阅请求的角色都可以被认为是订阅者。一般订阅者由于对别的事件感兴趣，或者受其它事件的驱动，会向系统发起订阅请求，请阅请求中会包括订阅者自身接收消息实体的地址以及感兴趣的主题。系统负责维护这些信息，当有相应的通知消息达到系统中时，就会被推送到相应订阅者的消息接收实体。

5、订阅（Subscription）：

订阅是一种关系，它描述了订阅者和发布者之间的契约关系，即发布者生产订阅者感兴趣的通知消息，订阅者向 HPwsn 系统描述这种兴趣，那么系统根据这种兴趣将二者关联起来，形成消息通路，实现消息分发。

6、订阅管理者（Subscription Manager）：

订阅管理者负责管理系统中的订阅信息，这些订阅信息是连接订阅者和发布者之间的桥梁。当系统中订阅数量较多时，如何快速实现订阅信息的匹配就是一项重要的工作，在很大程度上影响系统的消息吞吐量，因此，设计高效的匹配算法是更好地实现订阅管理者的主要任务。

4.3 独立部署

第三章中已经详细叙述过，ServiceMix 服务总线运行时占用的系统资源太多，不利于其应用场景的拓展，也不利于系统长时间稳定运行。脱离 ServiceMix 环境，实现独立部署，是 HPwsn 系统的核心需求。

对于 wsn 组件而言，ServiceMix 是一个容器，wsn 利用容器底层的消息通道实现路由。脱离 ServiceMix 之后，我们采用 Web Service 的方式，在 HPwsn 系

统启动时，对外发布一个 Web Service，作为数据的入口，任何用户都必须通过调用该 Web 服务将消息发送到系统中来，在系统中，则根据消息类型，分别交由不同的处理逻辑处理。

这样设计的好处在于，首先，大大降低了 HPwsn 系统对资源的占用；其次，提供标准的 Web Service 接口供用户调用，容易扩展；最后，这是一种平滑的架构迁移，对原有系统的消息吞吐量等性能没有损害。

4.4 服务层

本节中，我们将讨论 HPwsn 系统服务层的架构设计。图 4-2 描述了系统整体架构，图 4-3 从发布订阅流程的角度描述了更细粒度的系统架构。

图 4-3 描述了发布订阅的主要流程以及流程中涉及到的主要模块。其中，虚线框内部是 HPwsn 核心系统，可独立部署在具备基础支持环境的计算设备上，占用系统资源并对外提供发布订阅服务。虚线框外为核心系统外部模块，主要包

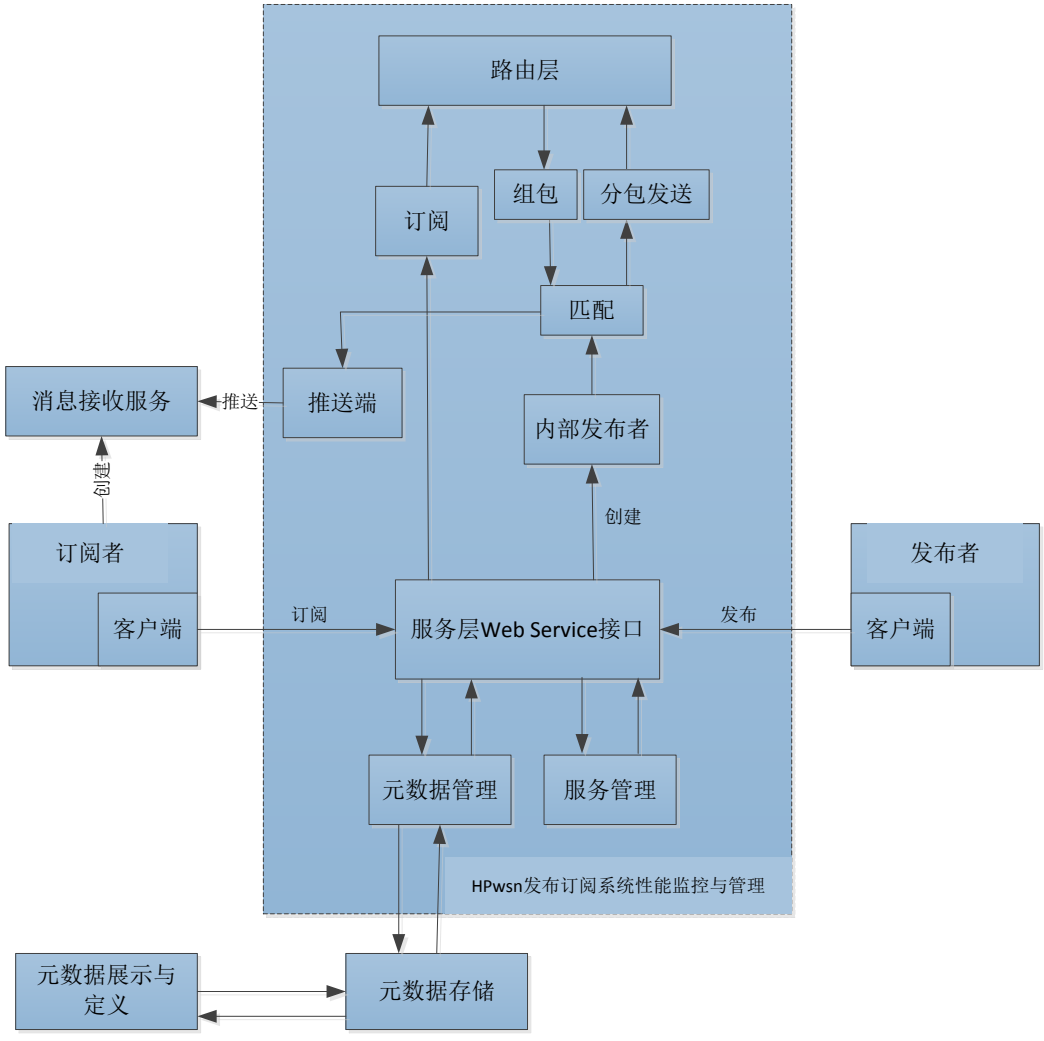

图 4-3　HPwsn 发布订阅系统详细架构图

括三个模块，发布者、订阅者以及元数据定义、存储和展示模块。

核心系统内外多个模块协同工作，共同提供完整的发布订阅服务，下面概要介绍系统各个流程、模块的设计。

4.4.1 系统初始化流程

HPwsn 系统启动后，会首先完成一个初始化过程，这个过程如图 4-4 所示。在这个过程中，主要会完成以下工作：

- 向分布式 Web 环境中的管理员注册，通知管理员自己加入集群；
- 启动元数据管理模块，从本地或远程持久化库中加载用户定义好的元数据信息；
- 启动服务管理模块，初始化系统资源；
- 发布 Web Service，对外提供服务。

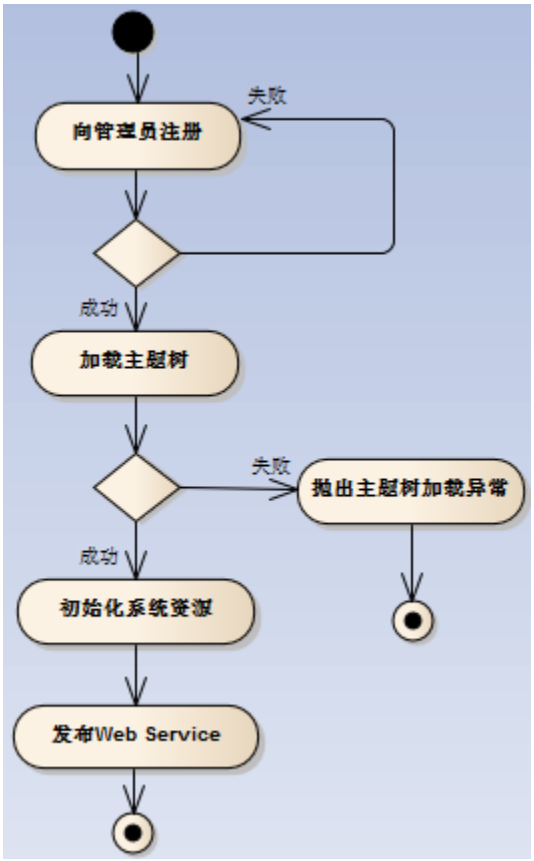

图 4-4 初始化流程

4.4.2 订阅流程

系统初始化完成后，订阅者就可以发起一个订阅流程，该流程共分为三步，首先，订阅者创建一个本地消息接收服务；其次，订阅者发起创建推送端点的请求，HPwsn 系统收到消息后根据要求创建推送端点，并返回创建状态。若创建

推送端点成功，则订阅者应发起订阅请求，在这个过程中，系统记录并维护该订阅的信息，为其分配系统资源，将该订阅与之前创建的消息推送端点绑定，并返回订阅的创建状态。

订阅过程的流程如图 4-5 所示。

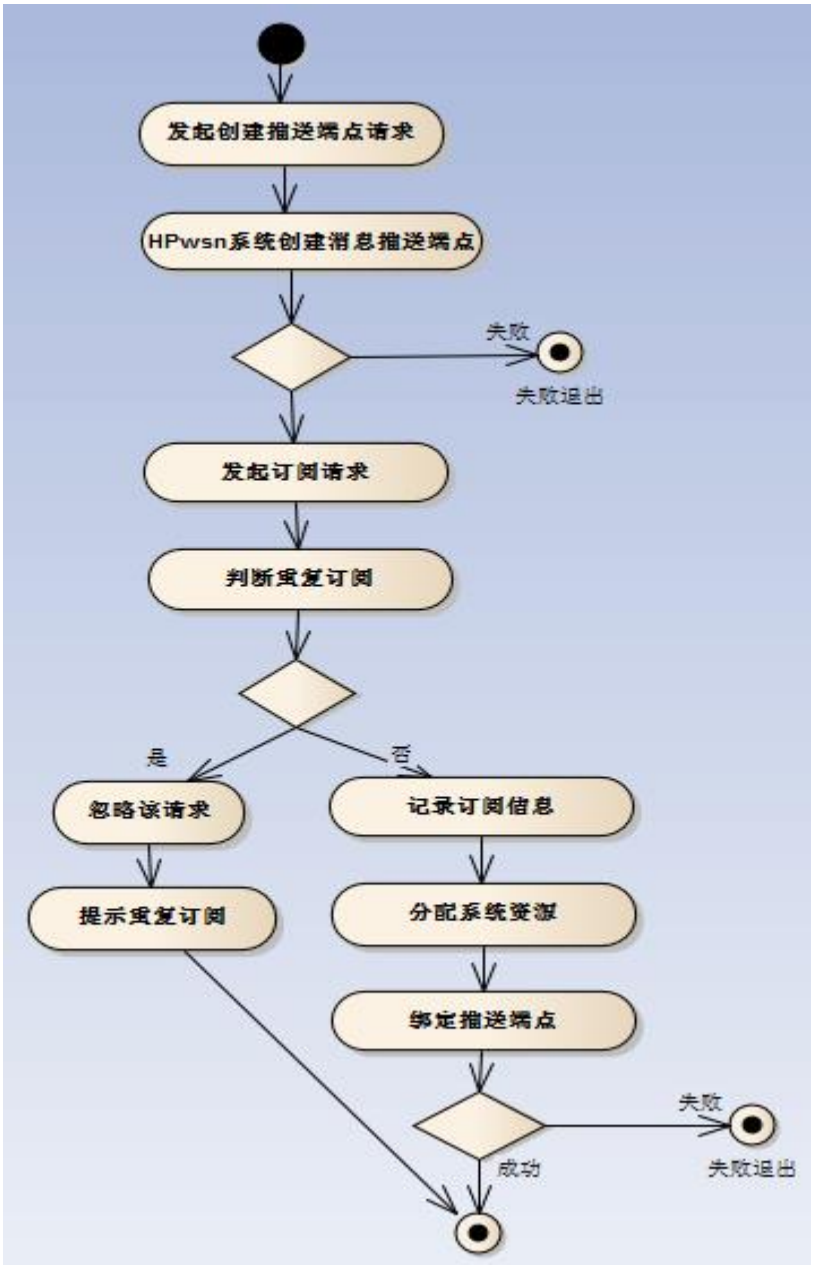

图 4-5 订阅流程

4.4.3 发布流程

系统初始化完成后，发布者就可以发起一个发布流程。首先，发布者通过某种方式感知真实或虚拟环境中发生的事件，并将之抽象为一种格式化的描述；其次，发布者调用客户端发送通知消息，HPwsn 系统收到通知消息后，解析消息

并根据其主题在主题树中进行匹配，匹配成功后获取其订阅者列表，调用相应的推送端点逐一推送消息。

发布过程的流程图如图 4-6 所示。

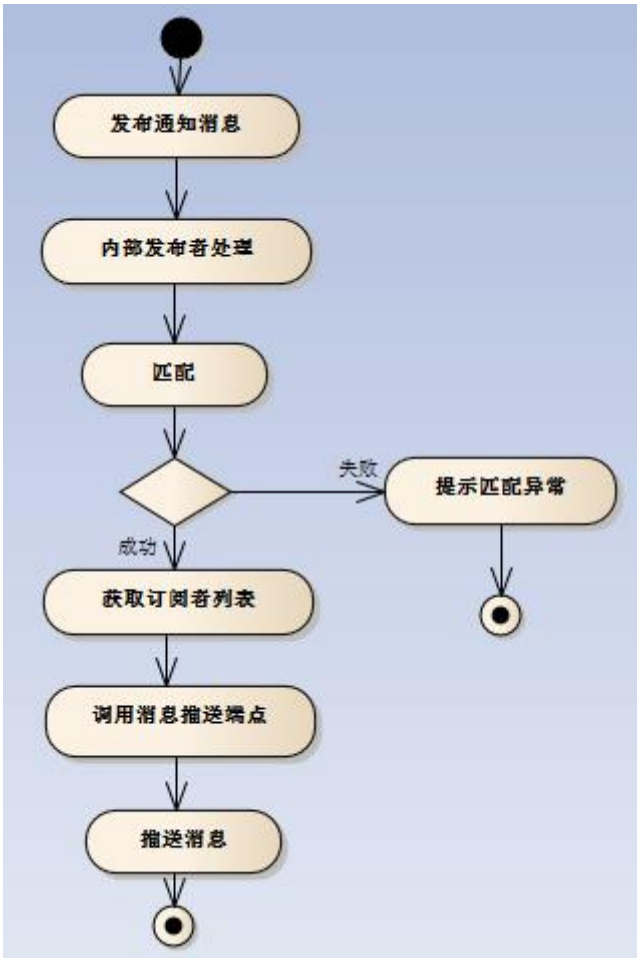

图 4-6 发布流程

4.4.4 异步消息推送

推送端点模块负责向用户递交消息。在订阅流程中，推送端点在系统处理完创建推送端点请求后建立，并在创建订阅的过程中与相关订阅绑定。在发布流程中，当系统获取到当前通知消息的所有订阅者之后，会调用每个订阅对应的推送端点模块将消息发送给相应的订阅者。

在现有系统中，推送端点使用同步 HttpClient 发送消息，每发送一条消息，需要读取其响应获取状态信息，然后再发送下一条消息，这种同步发送消息的方式直接导致了发送效率不高，系统吞吐量过小的问题。在 HPwsn 系统中，我们采用异步结合回调机制替代同步，即消息发送出去之后即刻返回，发送下一条消息，而消息接收端会调用推送端点提供的回调函数通知消息的发送状态。

这样设计的好处在于大幅提高了系统的消息吞吐量，并且依然能够获取消息发送状态，从而对复杂的网络状况及时作出反应。

4.4.5 分包发送

由第三章中对分包发送的叙述可知，分包发送的目的在于解决路由层使用 UDP 协议转发数据时数据包大小受限的问题。为了解决这个问题，HPwsn 系统在服务层向路由层递交消息时，添加了拆包模块，而在路由层向服务层递交消息时，添加了组包模块。

为了能够进行正确组包，需要在拆包端向消息体中添加一些附加信息，包括用于标识一条完整消息的<Identification>标签，和拆包信息标签<Package>。

拆包流程如图 4-7 所示。

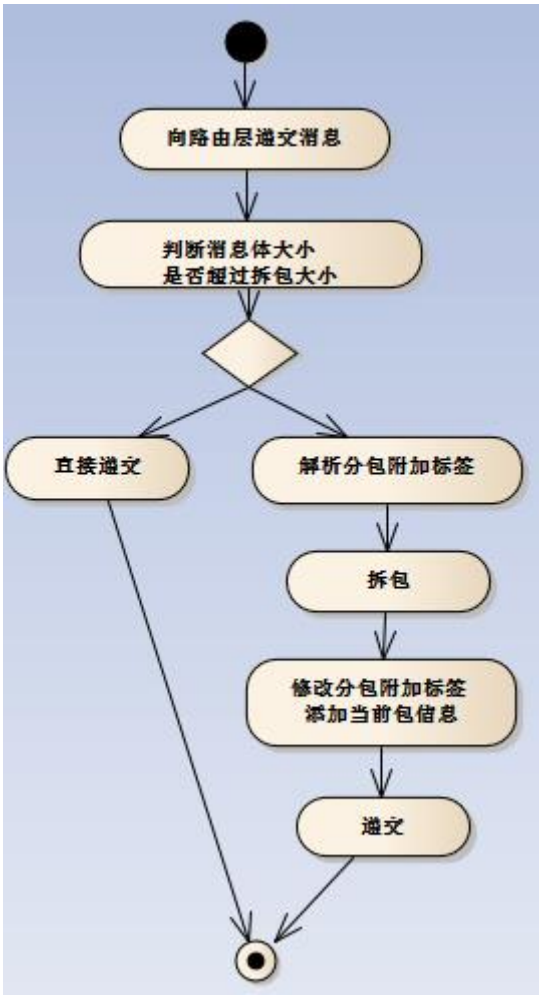

图 4-7 拆包流程

组包流程如图 4-8 所示。服务层收到路由层递交的消息后，会首先解析其分包附加标签的信息，以此为根据，决定下一步是直接进行匹配并向用户递交消息，还是启动组包进程进行组包。另外还有一点，每向一个组包进程中添加一个分包，就会检查当前组包进程是否已经获得所有分包，若是则进行组包，产生完整数据

包。

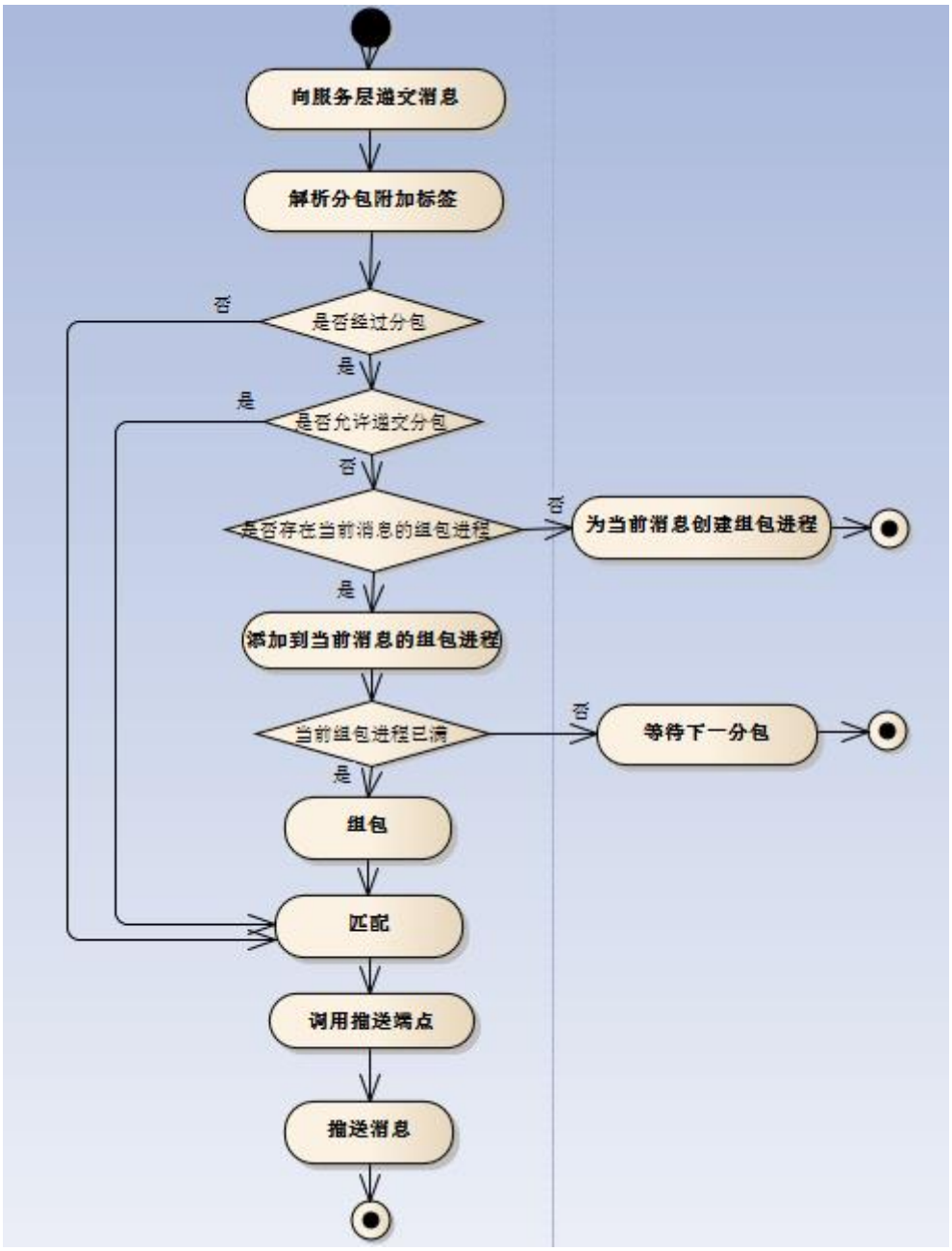

图 4-8 组包流程

4.4.6 树状名称表达

现有的基于 wsn 组件的发布订阅系统实现了基于简单主题的匹配，在实现中采用订阅表维护订阅信息，主题之间彼此独立，导致订阅者表达兴趣的方式过于扁平化。例如，假设系统中定义了主题 Fruit、Apple、Pear、Grape，若订阅者 A 对这 4 个主题均感兴趣，那么 A 需要分别订阅这 4 个主题，过程繁琐，容易造成混乱。

鉴于这种情况，为了提高订阅者表达兴趣的能力，满足实际应用中用户大批量、层次化的订阅需求，我们采用主题树实现订阅信息的存储和维护。主题树的方式带来的好处是显而易见的，它提供了一种结构化的名称系统，基于主题树可以实现组订阅，另外，主题树还可以维持主题之间的一种层次化关系，使得用户能够更清晰和得心应手地使用发布订阅系统。

4.5 外围模块

除了实现主体发布订阅功能的核心系统外，几个外围模块也是 HPwsn 系统的重要组成部分。包括发布者、订阅者模块，客户端模块，性能监控与管理模块。以及元数据的定义、展示与持久化存储模块。

4.5.1 发布者、订阅者模块

发布者、订阅者是每一个发布订阅系统中必不可少的角色，也是最重要的角色，发布订阅系统的设计始终是围绕怎样将消息更快速、实时、稳定地从发布者处转发到订阅者处进行的。对于我们的系统，这两种角色就是系统的实际用户，我们在设计系统的时候，一方面，要向用户提供尽量丰富的接口，满足用户多样化的需求；另一方面，我们还要使用户尽可能地脱离技术细节，尽可能少地限制用户，最好能实现用户只需要知道自己想发什么数据，就能够使用我们的系统。

4.5.1.1 订阅者

订阅者，这个角色对某种发生在别处的事件感兴趣，那么它可以通过订阅这类事件的通知消息，获取这类事件的描述，而在本地定义接收到通知消息之后的处理逻辑。

由于要实现主动推送功能，不必周期性地向系统请求消息，因此在订阅者本地必须有一个消息接收服务。一个订阅者程序启动的时候，会首先发布一个 Web Service，作为本地消息接收服务，而在这个 Web Service 的实现中定义接收到系统推送的通知消息之后的处理逻辑。Web Service 发布成功后，订阅者会调用客户端的方法开始发起订阅请求，并在请求中携带本地消息接收服务的地址。实现这样的流程就构成了一个最简单的订阅者程序。

4.5.1.2 发布者

发布者，扮演的是一个事件观察者的角色，它通过某种方式感知现实或虚拟世界中发生的事件（比如传感器），捕捉到事件后采用一定的格式（与订阅者约定好，或者说订阅者必须遵循这种格式去解析收到的通知消息）进行描述，最后将这种描述包装成格式化的数据，调用客户端的方法发送到 HPwsn 系统中，这

样就实现了一个简单的发布者程序。

4.5.2 客户端模块

客户端模块，就是 HPwsn 系统向用户提供了一个工具模块，客户端能够完成的工作，完全可以在订阅者和发布者程序中完成，但是，由于这类工作具有普遍性，我们将之抽象出来，封装成一套 API，供用户调用，这样，用户在实现订阅者或者发布者程序时，就可以从技术细节的泥潭中解脱出来，而只需要关注自身的数据即可。

以发布者为例，用户想要把自己的数据发送到系统中，首先，要将数据和主题封装成符合 wsn 规范的格式，因为 HPwsn 系统是基于 wsn 组件的，只有符合 wsn 规范的消息才能在 HPwsn 系统中被正常处理。接着，要将 wsn 规范格式的消息，再一次封装，成为 SOAP 消息，这样才能使用 HTTP 协议在网络中传输，而且这一步封装，要根据核心系统的 Web Service 接口定义进行。这个过程是比较复杂的。图 4-9 是一条构建好的 SOAP 消息。

```
1  <?xml version=\"1.0\" encoding=\"UTF-8\"?>
2  <soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
3      xmlns:org=\"http://org.apache.servicemix.application\">
4  <soapenv:Header/>
5  <soapenv:Body>
6  <org:WsnProcess>
7  <wsnt:NotificationMessage xmlns:wsnt=\"http://docs.oasis-open.org/wsn/b-2\">
8  <wsnt:Topic Dialect=\"http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple\">
9  topicName
10 </wsnt:Topic>
11 <wsnt:Package>
12 <Identification>
13 1235324366:344
14 </Identification>
15 <Fragment>0-0-1-0</Fragment>
16 </wsnt:Package>
17 <wsnt:Message>
18 notification
19 </wsnt:Message>
20 </wsnt:NotificationMessage>
21 </org:WsnProcess>
22 </soapenv:Body>
23 </soapenv:Envelope>
```

图 4-9 完整的发布消息

在这条 SOAP 消息中，1~6 行以及 21~23 行为 SOAP 消息包装部分，从第 6 行可以看出，SOAP 消息的包装是根据核心系统的 Web Service 接口定义有关的。其余的 XML 标签都是 WSN 规范格式的包装，其中嵌入消息体中的 topicName 和 notification 字段就是要发布的消息对应的主题和内容。

显然，这个过程如果放到发布者和订阅者程序中去实现，会给用户造成很大的困扰，用户必须熟知如何构建符合 wsn 规范的消息以及如何构建 SOAP 消息。另外，这部分封装具有普适性，即针对不同的消息，封装的过程是一样的，所以，

在 HPwsn 系统中，我们将这一类的消息封装以及消息发送的过程封装在一起，向用户提供 API，用户在调用方法的时候只需要将消息主题和内容作为参数传进来就可以了。以发布通知消息为例，用户调用客户端方法的流程如图 4-10 所示。

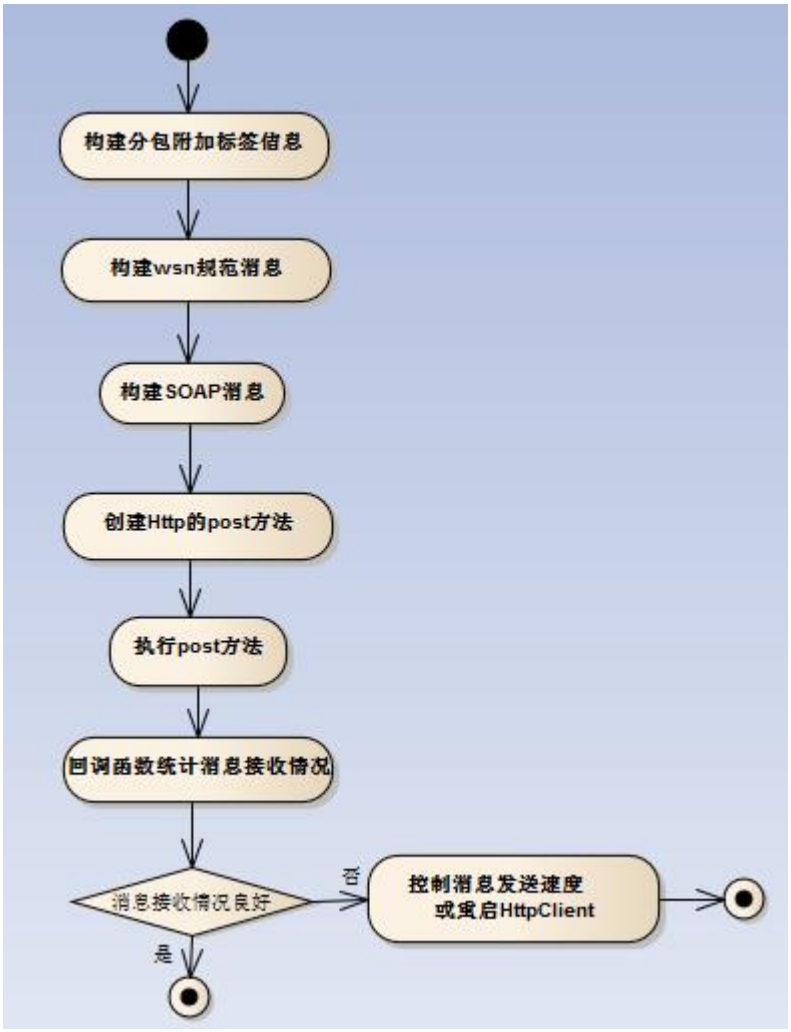

图 4-10 客户端调用流程

与前面论述的核心系统内部的推送端点向订阅者消息接收服务推送消息类似，客户端利用 Apache HttpClient 向系统发送消息也是整个 HPwsn 系统的主要性能瓶颈之一。对于订阅消息，由于其数量较少，不必太关注性能问题，但是对于发布通知消息，客户端模块的性能就至关重要了。

经研究和测试，现有的基于 wsn 组件的发布订阅系统，消息分发的性能都不是很高，大约在 20~40 包每秒的数量级，这远远满足不了生产环境的需求。因此，在这个模块，我们采用了与推送端点同样的技术，异步消息发送结合回调机制实现了消息的快速递交。

4.5.3 性能监控与管理模块

当系统中同时有大量的通知消息到来时，HPwsn 系统的负荷会很大，占用

的系统资源也会很多，这时候系统能不能长期、稳定地运行是一个很重要的问题。在系统的开发测试阶段，我们要有一套标准来评价系统的稳定性和资源占用；在系统部署到实际生产环境中之后，我们需要一个检测模块监控系统的运行，便于及时发现问题和找出问题原因。这就是性能监控与管理模块的主要职责。

整个 HPwsn 系统都是基于 Java 语言开发的，而 JDK 自带了一套性能监控工具，经调研发现，这套工具完全能够满足我们监控与管理系统的需求，因此，我们选用 JDK 自带的 Java VisualVM 来实现性能监控与管理模块。

4.5.4 元数据定义、展示与持久化存储模块

元数据就是用来描述数据的数据，在 HPwsn 系统中，最重要的元数据就是订阅信息。为了满足用户大批量、层次化的订阅需求，我们使用主题树来保存和管理订阅信息。在系统启动初始化的时候，有一个过程是从数据库中加载用户定

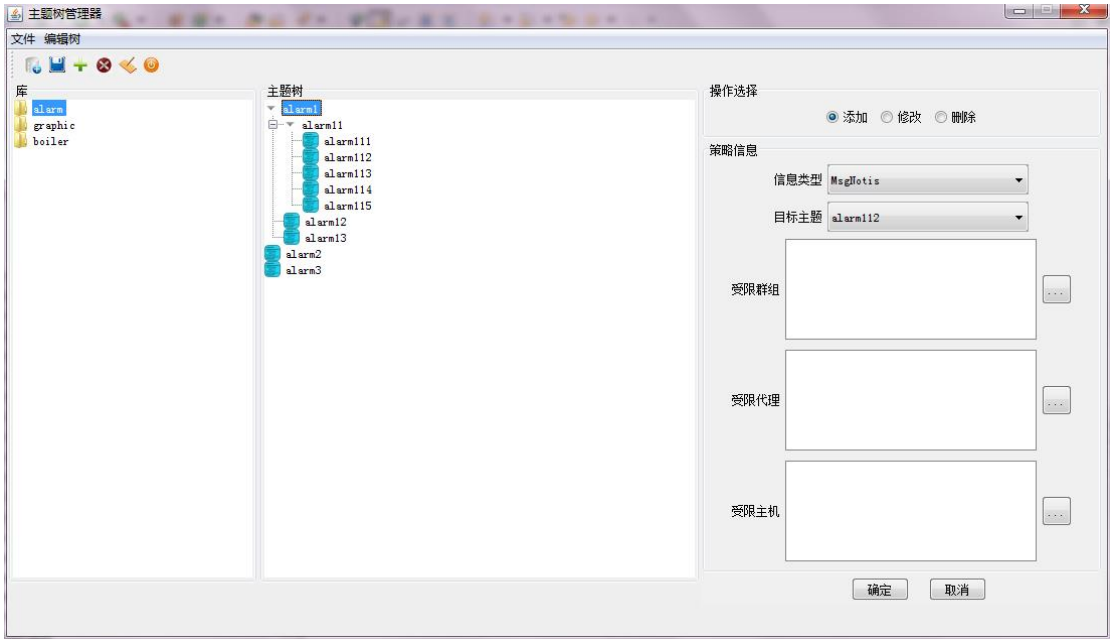

图 4-11 元数据定义、展示界面

义好的主题树，那么用户如何定义自己的主题树，如何展示以及如何持久化存储，这就是本节要介绍的内容。

为了使用户更简单方便地定义、组织主题树信息，我们提供一个图形界面供用户使用。如图 4-11 所示。在该工具中，用户可以按照自身需求定制主题树。主题树的数量并不受限制，用户定义好所有的主题树并保存之后，在数据库中存储的时候，所有主题树的根节点都有一个共同的虚拟父节点，这种结构类似 Java 语言的单根继承，目的是方便在主题树中查找和匹配主题。另外，值得一提的是，在图 4-11 所示的界面中，用户还可以为每一个主题制定路由转发的策略。

在界面中定义好的所有数据最终都将存储在 OpenLDAP 数据库中。在第二

章中，我们介绍了 LDAP 技术，最终选用 OpenLDAP 作为元信息的持久化库，是因为它采用树形结构保存数据，非常适合数据的树形展示，并且，我们可以直接将 Java 对象序列化之后存入数据库，代码与数据库之间不存在数据结构映射的过程，非常方便。不过在这里，必须强调一下要时刻注意存取序列化数据的一致性问题。

4.6 本章总结

本章的主要内容是 HPwsn 系统的概要设计。本章开篇从系统架构谈起，简要介绍了系统设计和实现过程中经常出现的概念和术语，接着描述了 HPwsn 系统的基础，独立部署的设计，然后对服务层的各个流程模块，以及外围的辅助模块，都进行了概要性的设计说明。通过本章，读者应该对 HPwsn 系统的整体架构和流程有了比较清晰的概念，贯穿本章的有两条主线，一是尽可能完善原有系统的功能，二是尽量提高系统性能。循着这两条主线，可以对整个系统有一个比较全面的了解。

本章内容是系统进行详细设计与实现的基础，下一章讲述的 HPwsn 系统的实现细节，都是遵循本章的设计进行的。

第五章 详细设计与实现

通过第四章的论述，我们了解了整个 HPwsn 发布订阅系统的整体设计，在本章中，对第四章中提到的各个模块进行展开，阐述其详细设计和具体实现。通过本章的说明，读者可以了解到一个真正实用的高性能发布订阅系统是如何构建起来的。

本章的内容主要分为以下几个部分，数据格式的设计与实现、服务层的详细设计与实现、外围模块的实现。

5.1 数据格式的设计与实现

构建一个真正可用的软件系统，数据格式的设计与实现是非常重要的环节，因为在系统的很多模块之间，并不是调用与被调用的关系，而是在进行数据交互。对于一个部署在分布式 Web 环境中的发布订阅系统尤其如此。

由第四章的论述可知，发布者和订阅者与核心系统之间交互的数据是基于 SOAP 协议和 XML 格式的，所以必须对数据进行格式化的定义。在本系统中，涉及到的数据主要是消息，主要有：创建推送端点消息、请求订阅消息、取消订阅消息、发布通知消息，下面主要对这四种消息的格式进行说明。

5.1.1 创建推送端点消息

创建推送端点的请求由订阅者发起，在订阅流程中，它位于订阅者创建本地消息接收服务之后，发起订阅请求之前，目的是为即将发起的订阅请求申请到相应的服务资源。系统收到该请求后，根据请求内容创建相应的推送端点，并返回推送端点的创建状态和地址给订阅者。一个典型的创建推送端点的请求信息如图 5-1 所示。

```
1  <?xml version=\"1.0\" encoding=\"UTF-8\"?>
2  <env:Envelope xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"
3      xmlns:org=\"http://org.apache.servicemix.application\">
4  <env:Body>
5  <org:WsnProcess>
6  <wsnt:CreatePullPoint xmlns:wsnt=\"http://docs.oasis-open.org/wsn/b-2\">
7  </wsnt:CreatePullPoint>
8  </org:WsnProcess>
9  </env:Body>
10 </env:Envelope>
```

图 5-1 创建推送端点请求消息

这是一条封装好的 SOAP 消息，用于请求系统创建一个推送端点，以

wsnt:CreatePullPoint 作为标识。这是一个不带参数的推送端点创建请求，该请求也可以是带参数的，放在<wsnt:CreatePullPoint>与</wsnt:CreatePullPoint>之间，用于指定推送端点的名字。

5.1.2 请求订阅消息

请求创建推送端点的消息收到成功的返回后，订阅者会发起订阅请求。订阅请求负责将本次订阅的核心信息发送给系统，并在系统中为订阅分配完整的服务资源。系统收到订阅请求后，一方面解析用户的订阅信息并实现存储和维护，另一方面创建订阅实体，并将订阅实体与之前创建的推送端点实现绑定。一个典型的请求订阅消息如图 5-2 所示。

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
    xmlns:org=\"http://org.apache.servicemix.application\">
<soapenv:Header/>
<soapenv:Body>
<org:WsnProcess>
<wsnt:Subscribe xmlns:wsnt=\"http://docs.oasis-open.org/wsn/b-2\"
    xmlns:wsa=\"http://www.w3.org/2005/08/addressing\">
<wsnt:ConsumerReference>
<wsa:Address>
endpoint:this.endpointAddr
</wsa:Address>
</wsnt:ConsumerReference>
<wsnt:Filter>
<wsnt:TopicExpression Dialect=\"http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple\">
topic
</wsnt:TopicExpression>
</wsnt:Filter>
<wsnt:SubscriberAddress>
this.localServiceAddr
</wsnt:SubscriberAddress>
</wsnt:Subscribe>
</org:WsnProcess>
</soapenv:Body>
</soapenv:Envelope>
```

图 5-2 订阅请求消息

从图 5-2 中可以看出，除去 SOAP 消息的包装，消息体中主要携带三部分信息。首先是<wsa:Address></wsa:Address>标签，该标签的内容为前一步创建推送端点请求的返回值，即系统为该订阅创建的推送端点的地址；其次是<wsnt:TopicExpression></wsnt:TopicExpression>标签，其中的内容为订阅者对自身订阅兴趣的表达，因为当前的 HPwsn 系统是基于主题的发布订阅系统，因此，这部分内容就是订阅者感兴趣的主题信息；最后是<wsnt:SubscriberAddress></wsnt:SubscriberAddress>标签，其中包含的内容是订阅者发布的本地消息接收服务的地址，在 HPwsn 系统中就是一个 Web Service 的 URL，用于接收系统主动推送过来的通知消息。

5.1.3 取消订阅消息

对于订阅者来说，对通知消息的兴趣并不是一成不变的，在一个时间段内，订阅者可能对某一类主题感兴趣，而另一个时间段可能对另外一类主题感兴趣。在订阅者变更订阅兴趣的过程中，不仅要发起对新主题的订阅请求，还要取消之前发起的相应订阅，通知系统释放为过期订阅分配的资源，并不再向该订阅者推送与之前主题相关的通知消息。一个典型的取消订阅消息如图 5-3 所示。

```
<env:Envelope xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"
    xmlns:wsnt=\"http://docs.oasis-open.org/wsn/b-2\" xmlns:wsa=\"http://www.w3.org/2005/08/addressing\">
<env:Header>
<wsa:Action>
http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/UnsubscribeRequest
</wsa:Action>
<wsa:To>
this.subscriptionAddr;
</wsa:To>
</env:Header>
<env:Body>
<wsnt:Unsubscribe>
</wsnt:Unsubscribe>
</env:Body>
</env:Envelope>
```

图 5-3 取消订阅消息

向系统发送取消订阅消息，必须知道该订阅的订阅端点地址，如图 5-3 中的<wsa:To></wsa:To>标签中的内容。系统收到取消订阅的请求后，会首先在主题树中删除该订阅，然后释放之前为该订阅分配的资源，包括绑定的推送端点。

5.1.4 发布通知消息

发布者生成格式化的通知消息之后，需要调用客户端的方法将消息发送到系统中。在客户端中会对发布消息做封装，其中主要封装的信息为该通知消息绑定的主题，以及通知消息的数据。一个典型的发布通知消息如图 5-4 所示。

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
    xmlns:org=\"http://org.apache.servicemix.application\">
<soapenv:Header/>
<soapenv:Body>
<org:WsnProcess>
<wsnt:NotificationMessage xmlns:wsnt=\"http://docs.oasis-open.org/wsn/b-2\">
<wsnt:Topic Dialect=\"http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple\">
topicName
</wsnt:Topic>
<wsnt:Package>
<Identification>
1235324366:344
</Identification>
<Fragment>0-0-1-0</Fragment>
</wsnt:Package>
<wsnt:Message>
notification
</wsnt:Message>
</wsnt:NotificationMessage>
</org:WsnProcess>
</soapenv:Body>
</soapenv:Envelope>
```

图 5-4 通知消息

在通知消息中，除去 SOAP 消息封装，主要的内容有四部分。

<wsnt:Topic></wsnt:Topic>标签的内容为当前通知消息绑定的主题，由于 HPwsn 系统是一个基于主题的发布订阅系统，所以每一条通知消息都有其绑定的主题，系统会根据主题去查找对其感兴趣的订阅者，从而实现发布者与订阅者之间的关联。<wsnt:Package></wsnt:Package>标签中携带的信息是用于实现分包发送和组包的，在发送端会根据用户的配置和消息的分包情况修改该标签中的内容，在接收端则根据这些信息恢复原来的数据包。<wsnt:Message></wsnt:Message>标签中的内容是本条通知消息的数据，即发布者想要发布到系统中，供订阅者订阅的信息。

5.2 独立部署实现

现有的基于 wsn 组件的发布订阅系统，都是运行在 ServiceMix 总线中的，依赖于 JBI 环境。ServiceMix 总线运行时占用的系统资源太多，并且其中运行的大部分组件都与发布订阅系统无关，因此，我们希望把 HPwsn 系统剥离出来，成为一个可独立部署运行的 Java 应用程序。

为使系统在独立运行时正常对外提供服务，HPwsn 系统会发布一个 Web Service 提供服务，系统的用户，如发布者和订阅者都可以通过调用 Web Service 来获取系统提供的发布订阅服务。

5.2.1 IWsnProcess 接口：

该接口是 Web Service 的接口，定义了 Web Service 的名字，消息处理方法以及接受的参数类型。

5.2.2 WsnProcessImpl 类：

WsnProcessImpl 类实现了 IWsnProcess 接口，完成的主要功能有两个，一是在系统启动时初始化整个系统，二是初步处理用户（订阅者或者发布者）发送到系统中的消息。下面说明该类中的几个主要方法。

（1） public void init()方法：

该方法负责在系统启动时初始化系统资源，其流程如图 5-5 所示。

从图中可以看出，init()方法一开始首先初始化 JMS 资源，HPwsn 发布订阅系统整体上是建立在 JMS 技术之上的，核心系统的内部发布者模块就是 JMS 消息发布者实现的。也就是说，我们的 HPwsn 发布订阅系统最基本的发布订阅功能是 JMS 提供的，我们的系统在此基础之上进行了多层次的封装，提供用户友好的发布订阅服务。

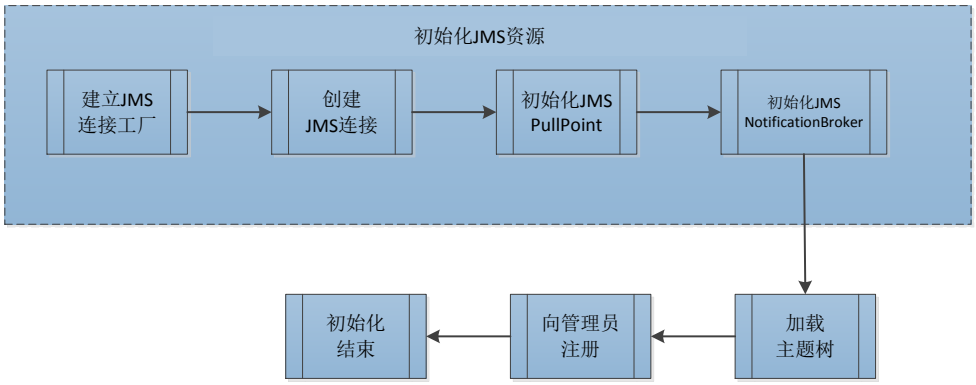

图 5-5 init()方法流程图

JMS 资源初始化完成之后，init()方法会从 OpenLDAP 数据库加载主题树信息，并以树状结构保存在内存中，具体实现过程参见 readTopicTree(String root_path)函数。

最后，系统向分布式 Web 环境中的管理员注册，告知管理员又有一台 HPwsn 主机加入了某个集群。管理员收到消息之后会修改当前 HPwsn 网络的拓扑结构并与该主机始终维持一个心跳消息，以判断该主机的生存状态。到这里，初始化过程就全部结束了。

（2） public static void readTopicTree(String root_path)方法：

该方法负责从 OpenLDAP 数据库中加载主题信息，并将之存储在树形结构中。

```
queue.offer(topicTree);
while(!queue.isEmpty()){
    WSNTopicObject to = queue.poll();
    List<TopicEntry> ls = ldap.getSubLevel(to.getTopicentry());
    if(!ls.isEmpty()){
        List<WSNTopicObject> temp = new ArrayList<WSNTopicObject>();
        for(TopicEntry t : ls){
            WSNTopicObject wto = new WSNTopicObject(t, to);
            temp.add(wto);
            queue.offer(wto);
        }
        to.setChildrens(temp);
    }
}
```

图 5-6 主题树加载核心代码

该方法的核心代码如图 5-6 所示。在 OpenLDAP 数据库中，数据是按照树形结构存储的，加载到内存中之后也是一样，因此，加载主题树的过程就是遍历 OpenLDAP 中的主题树，取出并存储在内存中的主题树中。

为完成这个过程，我们使用树的层序遍历算法。创建一个队列，队列中存储的节点的共同特征是，都已经被访问过，但是其孩子节点都还没有被访问，若其

孩子节点都已经被访问过，则将该节点出队列。这样，当队列为空时，我们就遍历了树中的所有节点。

具体的做法是，首先将根节点入队列；使用一个 while 循环，若当前队列不为空，就取出队头元素，获取其所有儿子节点，将它们都保存到内存中的主题树中，并且全部入队列，然后继续下一次循环判断。

（3） public String WsnProcess(String message)方法：

该方法是 Web Service 接口中定义的接收消息的方法，message 就是订阅者或者发布者发送到系统中的消息。该方法相当于是所有消息的入口，在该方法中进行分类，然后调用不同的方法进行具体的处理。

（4） public fast_Notify(String message)方法：

该方法扮演内部发布者的角色。发布者发布的通知消息在 WsnProcess 方法中解析并分类之后，交由该方法处理。

该方法首先解析通知消息的主题，并在主题树中匹配该主题，若匹配不成功，则提示异常，发布者发布的主题在系统中并不存在；如果匹配成功，则查看该主题的订阅者列表，看是否有订阅者对该主题感兴趣，没有则将消息丢弃，若有则创建 JMS 发布者，将消息发送给 JMSSubscription 类中的 onMessage(Message jmsMessage)方法处理。该方法的流程如图 5-7 所示。

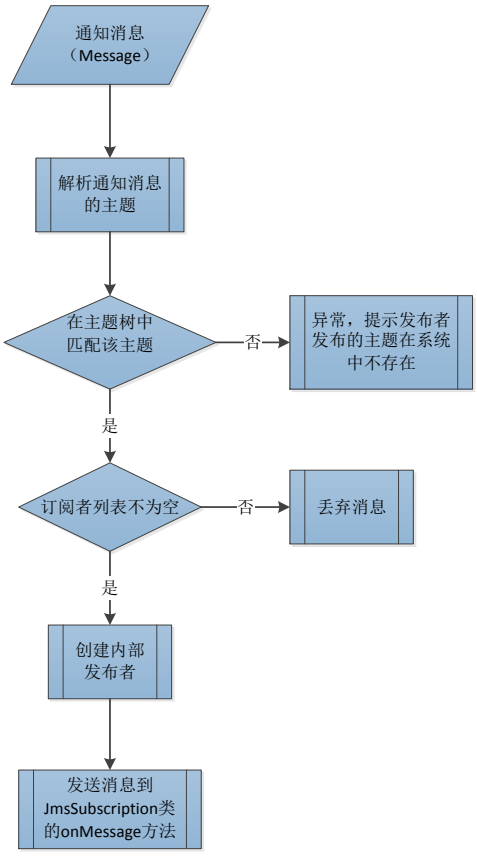

图 5-7 fast_Notify 方法流程图

5.2.3 WsnProcess 类：

该类是整个 HPwsn 系统的入口，在该类的 main 函数中做了三件事，首先创建一个 WsnProcessImpl 类的实例，然后调用上文介绍的 init()方法。最后使用 JDK 自带的 javax.xml.ws.Endpoint 端点发布一个 Web Service，代码如下：

Endpoint.publish(args1, args2);

publish 方法接受两个参数，第一个参数是发布 Web Service 的地址，第二个参数是 Web Service 接口实现类的一个实例。发布成功后，就可以在浏览器中输入 args1?wsdl 查看该 Web Service 的描述文件了。

5.3 核心发布订阅功能模块实现

发布订阅功能的实现是任何发布订阅系统的核心，该模块实现了对创建推送端点消息、订阅消息、取消订阅消息以及发布消息等消息的处理流程。

本文论述的 HPwsn 系统是一个高性能发布订阅系统，处理消息的能力强、速度快是 HPwsn 区别于其它发布订阅系统的重要特征，对于系统中的所有模块，我们的设计都尽量做到是高效的、最优的。另外，考虑到发布订阅系统中，通知请求的数量远远大于订阅请求的数量，因此，当二者处理过程的性能出现矛盾时，我们会在订阅消息的处理流程部分略作牺牲，以保证通知消息处理流程的高效，从而保证了整体系统的高性能，这也是我们 HPwsn 系统的设计原则之一。

5.3.1 订阅过程

HPwsn 系统启动并完成初始化后，即可以进行订阅。一个典型的订阅流程如图 5-8 所示。

一次订阅，由一个订阅者用户发起，订阅者首先在本地创建一个消息接收服务，用于接收系统推送的消息。该消息接收服务对于主动推送型发布订阅系统是必须的，否则系统不知道将消息推送到什么地址。在之前不支持主动推送的发布订阅系统中，系统内部会为每一个订阅保存一个消息队列，用于存储这个订阅感兴趣的通知消息，而订阅者程序会每隔一段时间定期去队列中取消息。这种方式不但影响了消息的即时传递，而且对于负责大量消息转发的系统来说，由于为每一个订阅都分配了一个消息队列，会造成非常大的内存占用。

然后，订阅者向系统发起建立推送端的请求，系统受到消息后，会创建相应的推送端点，然后向订阅者返回推送端点创建成功与否及其地址。

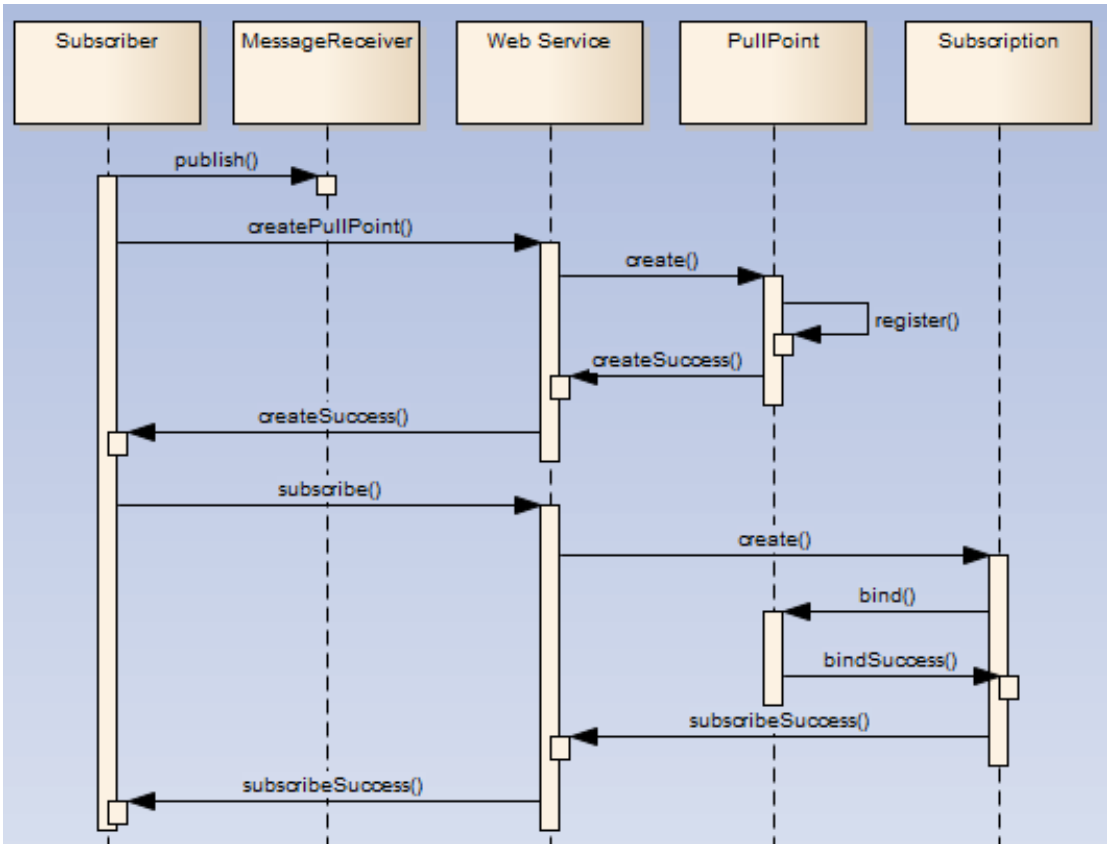

图 5-8 HPwsn 发布订阅系统订阅过程时序图

最后，若推送端点创建成功，订阅者程序根据上一步骤中返回的推送端点的地址，向系统发起订阅请求，请求中携带订阅者对于订阅需求的表达信息以及本地消息接收服务的地址。系统收到请求后，首先创建订阅实体，然后将该订阅实体与推送端点绑定，最后返回订阅是否创建成功以及订阅端点地址。

上面描述的是一次订阅过程的大概流程，在创建订阅实体的过程中，有几个问题需要考虑。

首先，如何处理重复的订阅请求。重复订阅，指的是在发布订阅系统的一个生命周期中，一个消息接收服务向系统多次发起对同一个主题的订阅请求。显然，重复订阅是毫无意义的，应该防止将有限的系统资源重复分配给同一个订阅关系。由于允许一个消息接收服务订阅多个主题，在订阅者发起创建推送端点的过程中，仅仅根据消息接收服务地址无法判断是否是重复订阅，因此，对重复订阅的处理只能放在创建订阅实体的过程中。系统收到订阅者创建订阅的请求后，会根据内部存储的订阅信息去匹配当前的订阅，判断该订阅是否是重复订阅，若是，则跳出当前订阅逻辑，并释放上一个步骤中为该订阅创建的推送端点等资源。重复订阅消息的处理流程如图 5-9 所示。

其次，订阅信息的存储。系统在处理订阅请求的过程中，必须实现订阅信息的存储，以便于查询和匹配。如何存储订阅信息，采用什么样的数据结构来存储，

将会在后面的元数据管理与存储模块详细说明。

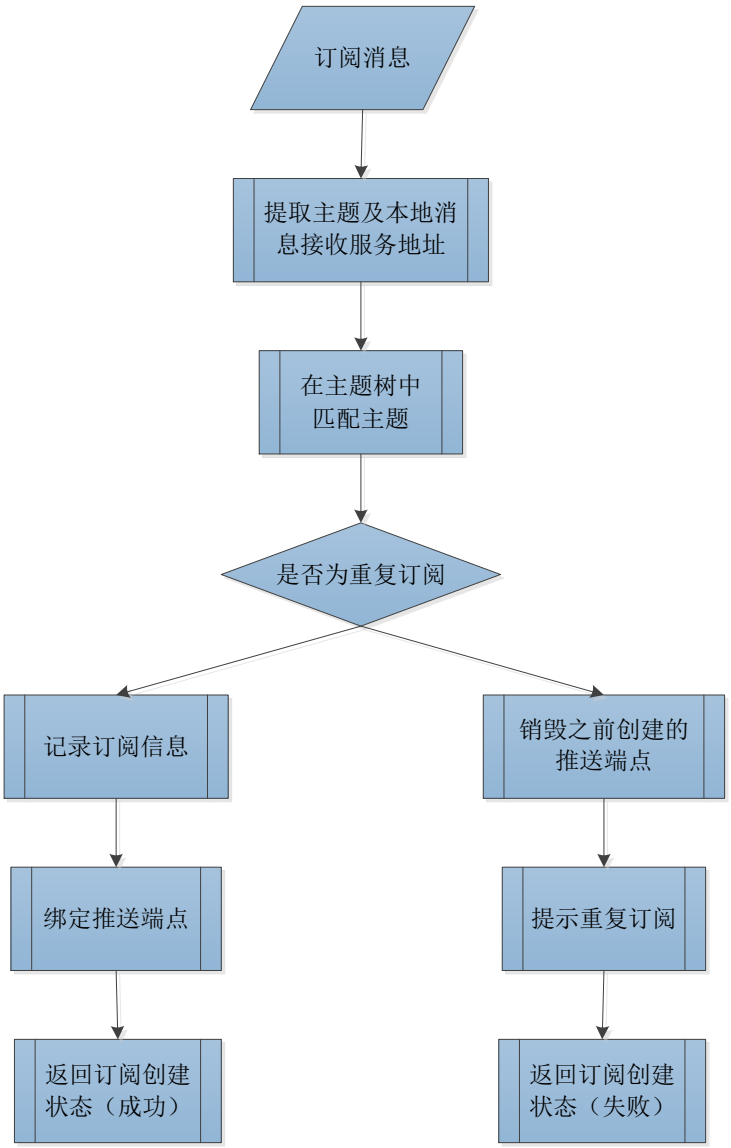

图 5-9 重复订阅处理流程

5.3.2 发布过程

HPwsn 系统启动并完成初始化后，即可进行发布过程。一个典型的发布过程如图 5-10 所示。

在发布过程中，发布者首先按照一定的规则构建一条通知消息，用以描述真实或虚拟世界中的事件，将通知消息及其对应的主题打包封装成 SOAP 消息，发送到系统中。系统收到通知消息后，首先解析其主题，根据主题创建内部发布者，将消息发送给发布消息处理逻辑，在这个模块，系统会根据当前通知消息的主题以及系统中存储的订阅信息完成匹配，从而获得当前通知消息的所有订阅者信息，遍历其订阅者列表，调用订阅者关联的推送端点，将消息推送到订阅者的消息接收服务，完成发布过程。

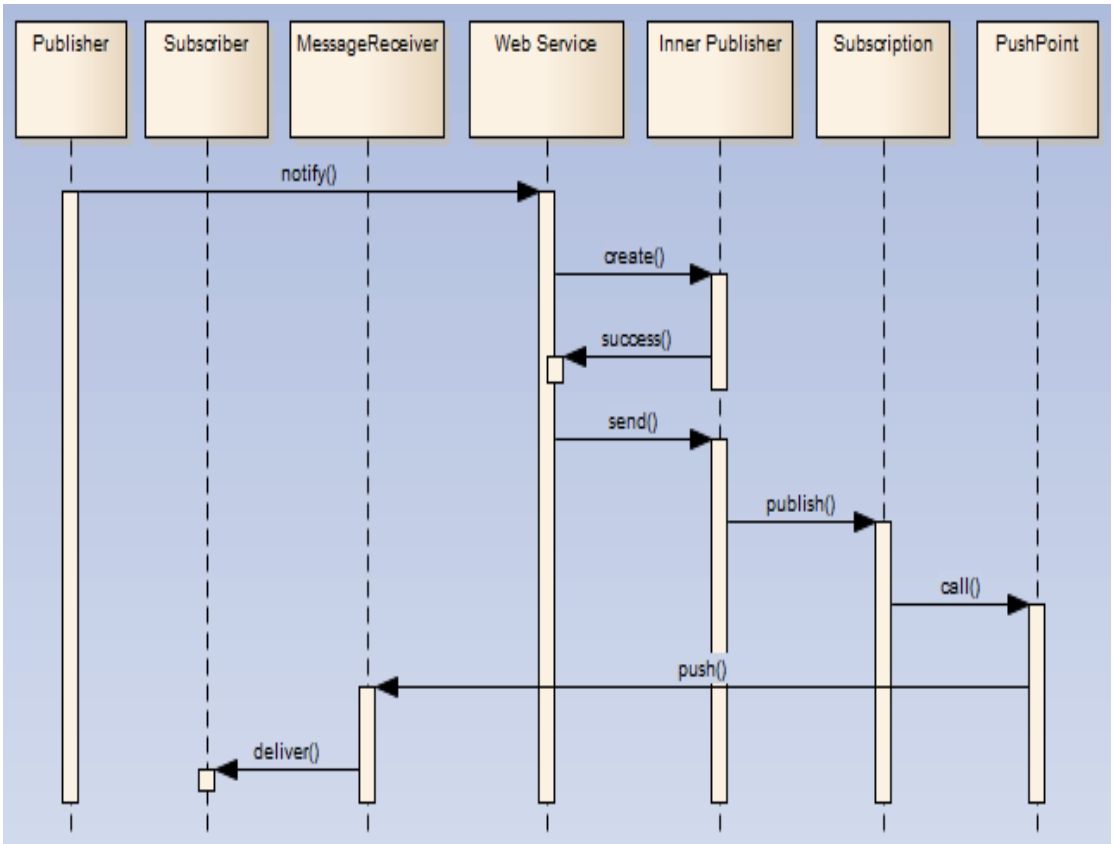

图 5-10 HPwsn 发布订阅系统发布过程时序图

发布订阅模块的类图如图 5-11 所示。

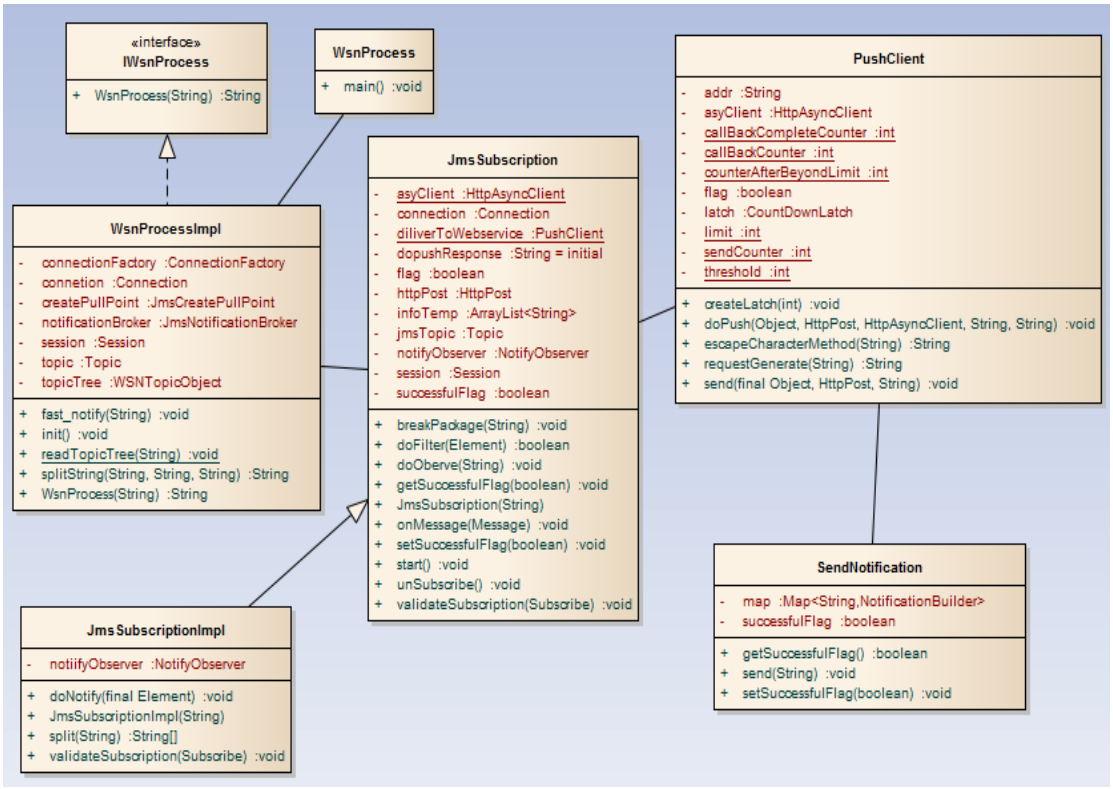

图 5-11 发布订阅功能模块类图

5.3.3 子功能模块实现

在发布订阅过程中，涉及到很多子功能模块，HPwsn 系统的高性能和高可靠性的特点在这些子功能模块的设计与实现中体现得非常明显。下面，我们详细阐述这些子功能模块的设计与实现细节。

5.3.3.1 内部发布者模块

内部发布者模块是处理通知消息的流程中涉及到的一个子功能模块，其主要职责是在通知消息达到系统中之后，对其进行预处理，提取主题等信息，然后创建一个 JMS 发布者，将消息送往通知消息的处理逻辑。该模块的设计初衷是为了避免直接操作通知消息处理模块，这样在大量的通知消息到达系统中时，起到一个减轻系统压力，输入缓冲的作用。该模块基于 JMS 的发布订阅机制实现，。内部发布者扮演 JMS 消息发布者的角色，而通知消息处理模块扮演 JMS 消息接收者的角色。

添加内部发布者模块在一定程度上可以加快系统处理通知消息的速度，其原理在于利用 JMS 通道传送消息的性能较高，相对于系统通过 Web Service 接收通知消息的速度要高出很多，因此不会影响系统处理消息的性能，并且将一部分处理放到了内部发布者模块进行，加快了速度。

5.3.3.2 取消订阅模块

取消订阅是发布订阅系统应提供的一个基本功能，但是主动推送型发布订阅系统在实现取消订阅功能时，只是删除了订阅信息，而没有释放之前为该订阅分配的资源，比如创建的推送端点。

在 HPwsn 系统中，取消订阅实现了彻底的订阅资源释放，类似重复订阅的处理流程，在收到取消订阅请求后，会释放与该订阅相关的推送端点，并删除订阅信息。

5.3.3.3 推送端模块

推送端模块位于通知消息处理流程的末端，主要功能是向用户递交通知消息。其输入是通知消息的消息体以及目的地址，该目的地址就是订阅者的消息接收服务地址。推送端将通知消息封装成 SOAP 消息，使用 HTTP 协议将消息发送出去。

对于比较小的消息推送需求来说，推送端模块的逻辑是比较简单的，但是，当大量的通知消息不间断地需要发送给订阅者的时候，该模块的设计就需要考虑性能的问题。

在现有的主动推送型发布订阅系统中，推送端采用 Apache HttpClient 模块

（基于 Java 语言的一套 HTTP 以及相关协议的工具集）发送消息，并且采用同步调用方式。对于同步调用方式，发送者一方的线程在发出消息后一直等待服务端返回的结果，如果服务端的处理时间比较长则请求端线程将一直处于等待状态。这种调用方式严重影响了发布/订阅系统的消息吞吐量，是系统的主要瓶颈之一。

基于这种现状，本文论述的 HPwsn 系统提出了一种能够实现自动回调的异步调用方式。在快速发送消息的同时实现了在发送端针对接收端不同的消息处理状况作出及时的处理。

为了实现自动回调机制，必须进行回调注册，即将各回调函数的地址作为参数，随消息一起发送到服务端，发送消息后马上可以释放发送端线程，处理其他事务或继续发送下一条消息。当接收处理完请求消息后，会根据回调函数的地址调用回调函数，将处理结果的有关信息发送给请求方，触发后续的处理逻辑。

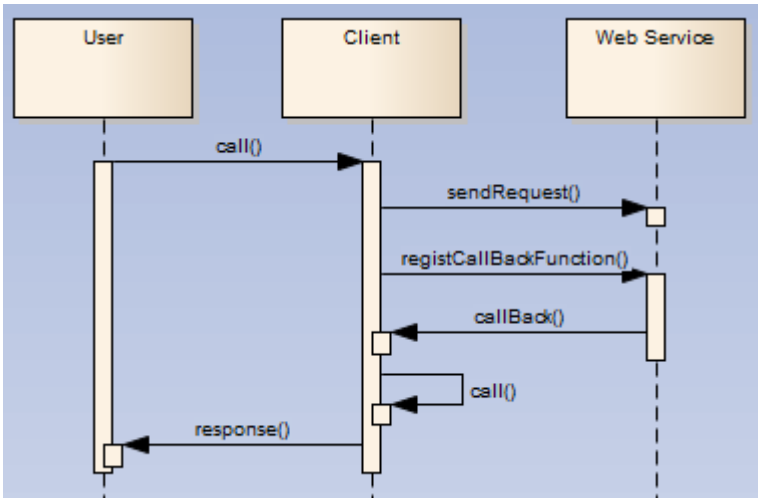

图 5-12 异步发送时序图

采用这种异步+回调的方式发送消息，一方面，发送端线程不用等待服务端处理消息，大大提高了系统消息吞吐量；另一方面，接收端及时地将消息的处理结果反馈给请求端，以便发送端及时地根据消息处理结果调整消息发送策略。另外，开发者还可以很容易地修改后续处理逻辑，加强对消息发送过程的监控和管理。

5.3.3.4 分包发送与组包模块

5.3.3.4.1 详细设计

分包发送与组包模块是核心系统的内部模块，这个模块的设计源于实际部署运行 HPwsn 系统的时候遇到的问题。该问题是由路由层做数据转发时的协议选取策略引起的。

在分布式 Web 环境中部署 HPwsn 系统是最常见的一种是用 HPwsn 系统的方式。由于网络中存在多台 HPwsn 主机，对其进行分组就势在必行，每一台 HPwsn 主机加入网络时都是带有集群信息的，以此标识自身属于哪个集群。每个集群选出一台主机做代表，通常是最先注册该集群的主机，通过代表实现集群间的数据转发。

路由层在选择数据转发使用的协议时，对于集群间的数据转发，使用 TCP 协议，而对于集群内的数据转发则使用 UDP 协议（用户也可通过配置的方式干预协议选取策略）。在使用 UDP 协议传输数据时，受限于操作系统的套接字缓冲区大小，UDP 包一般不能超过 8K。为了解决大数据包的传输问题，我们在服务层实现了分包发送和组包模块。

为了标识分包信息，我们在每条消息中增加了 package 属性和 identification 属性。identification 属性用于唯一标识一条消息，package 属性用于存储分包信息，由 8 位标志位组成，格式如图 5-13 所示：

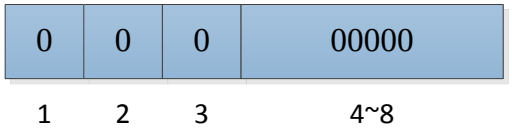

图 5-13 分包信息

其中，第 1 位标识该条消息是否被分包，1 代表分包，0 代表不分包；

第 2 位标识该分包后面是否还有更多的分包，1 代表有，0 代表没有；

第 3 位标识在接收端是否允许将各个分包单独向用户递交，1 代表允许，0 代表不允许；

第 4 到 8 位标识该分包相对于第一个分包的偏移量，该偏移量的取值范围为 0 到 99999。

在发送端，若发现数据包过大，则将该消息分包，每一个数据包都携带上述分包信息，这样在接收端就可以根据 identification 属性和分包信息将各个分包组装成完整的消息。

接收端收到消息后，首先解析消息的 package 属性，若该消息没有经过分包，或者允许将各个分包单独向用户推送，则直接启动推送流程；否则解析消息的 identification 属性，根据消息 id 查看系统中是否存在该条消息的组包流程，若存在，加入该流程，若不存在，为这条消息启动一个新的组包流程。每向一个组包流程中加入一个分包时，检测该组包流程是否完成，若已完成，则启动向用户递交的流程。

HPwsn 系统设计过程中，考虑到实际生产环境的需要，采用 5 位数字标识分包偏移量，并提供可配置的分包大小，设 UDP 分包最大为 8K，则该系统能够

传输的数据包大小为 800M，是原来的十万倍，能够满足绝大部分应用环境的需要。

另外还有一点需要说明，由于拆包组包模块是为了系统更好地支持 UDP 协议，而 UDP 协议是不保证交付的，因此组包流程存在出现死等状态的可能性，在 HPwsn 系统中，我们在组包逻辑中加入了超时判断，解决了这个问题。

5.3.3.4.2 拆包实现

要想在收端实现组包，拆分后的数据包中必须携带某些信息，以标识自身，上文详细解释了附加信息，下面我们来看一下具体实现。

```
public void breakPackage(String message){
    //解析出外围包裹消息体
    String beforeFragment = message.split("<Fragment>")[0] + "<Fragment>";
    String betweenFrangmentAndNotification = "</Fragment>" + "</wsnt:Package>" + "<wsnt:Message>";
    String afterNotification = "</wsnt:Message>" + message.split("</wsnt:Message>")[1];
    //解析出消息内容和打拆包信息
    String notification = message.split("<wsnt:Message>")[1].split("</wsnt:Message>")[0];
    String fragment = message.split("<Fragment>")[1].split("</Fragment>")[0];
    String[] splitFragment = fragment.split("-");

    int num = message.length() / 3000;
    //分包长度
    int length = 3000 - beforeFragment.length() - betweenFrangmentAndNotification.length() -
            afterNotification.length();
    //构建拆包后的消息体，并将之存储在一个ArrayList中
    int i;
    for(i=0;i<=num;i++){
        String partOfNotification = null;
        String partOfFragment = null;
        if(i<num){
            partOfNotification = notification.substring(i*length, (i+1)*length);
            partOfFragment = "1-1-" + splitFragment[2] + "-" + i;
        }else{
            partOfNotification = notification.substring(i*length, notification.length());
            partOfFragment = "1-0-" + splitFragment[2] + "-" + i;
        }
        StringBuilder breakedMessage = new StringBuilder(beforeFragment);
        breakedMessage.append(partOfFragment);
        breakedMessage.append(betweenFrangmentAndNotification);
        breakedMessage.append(partOfNotification);
        breakedMessage.append(afterNotification);

        infoTemp.add(breakedMessage.toString());
    }
}
```

图 5-14 拆包函数

在 5.1 节中，我们定义了通知消息的格式，参见图 5-4，系统收到并处理的就是这样一条通知消息，在向路由递交之前，系统会判断该数据包的大小是否超过了配置的分包大小，若是，则将其拆包，否则直接向路由递交，拆包函数如图 5-14 所示。

该函数的流程图如图 5-15 所示。

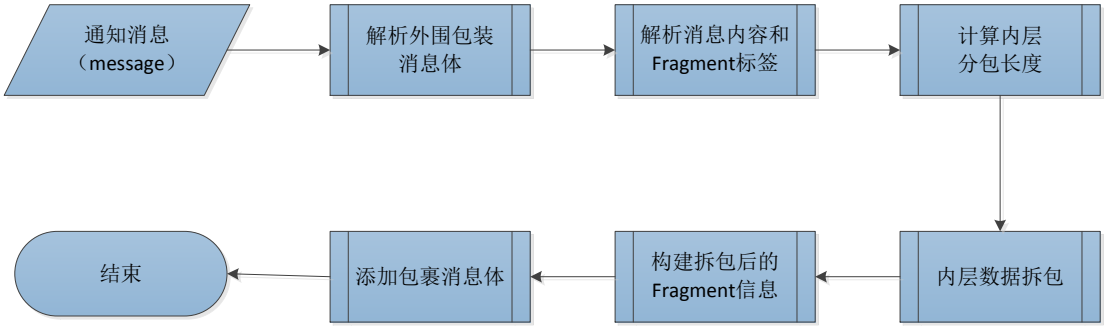

图 5-15 拆包函数流程图

JmsSubscription 类的 onMessage 函数处理完通知消息后，若本地有订阅者，则调用推送端点首先向本地订阅者递交消息，然后向路由层递交消息；若本地没有订阅者则直接向路由层递交。在向路由层递交消息之前，会首先判断这条消息的大小是不是超过了配置的分包大小限制，若通知消息过大，则会调用上面的拆包函数进行拆包，该函数的参数就是待拆包的消息。

在拆包函数中，首先会解析外围包装消息体。在我们的 HPwsn 系统中，只有满足 wsn 规范格式的消息才能被正确处理，因此，拆包的时候，我们拆的不是消息外面包裹着的 wsn 规范，而是其中用户发送的数据，将数据拆成一个个小包之后，还要给他们裹上 wsn 规范格式，消息才能继续处理，因此，我们首先解析出消息外围包装的 wsn 规范信息。

其次，解析消息内容（即用户发送的数据）与 Fragment 标签，对于没有经过拆包的消息，其 Fragment 标签中的内容是没有意义的，只是一个初始化值，在拆包的过程中，我们要为每一个分包打上不同的 Fragment 标签值，因此，解析消息内容和 Fragment 标签是必要的。

然后计算内层数据拆包长度。我们之前配置的分包长度是针对一条包裹好的通知消息，相应的内部数据的拆包长度在这部分进行计算。然后按照计算好的分包长度对内层数据进行分包，并将之存储在一个 List 中。在这个过程中，每条拆包消息对应的 Fragment 标签信息也得以构建。

最后，为每条拆好的内部数据包裹上 wsn 规范信息，这样，拆包过程就完成了。

5.3.3.4.3 组包实现

在接收端，服务层收到路由层递交的消息后，会判断这条消息是不是一个经过拆包的消息（这些信息都包含在消息的 Fragment 标签中），若不是则去掉拆包信息，然后调用推送端点向用户递交，这样保证了拆包组包过程对用户是不可见的；若是则启动组包流程。组包部分的代码如图 5-16 所示。

```
String fragment = message.split("<Fragment>")[1].split("</Fragment>")[0];
System.out.println("[Fragment:]" + fragment);
String[] splitFragment = fragment.split("-");

StringBuilder s = new StringBuilder("[splitFragment:]");
for(int i=0;i<splitFragment.length;i++){
    s.append(splitFragment[i]);
    s.append(" ");
}

//若该消息未经过分片或者该消息允许递交断包，则直接递交
if(Integer.parseInt(splitFragment[0]) == 0 || Integer.parseInt(splitFragment[2]) == 1){
    //去掉拆包信息
    notification = message.split("<wsnt:Package>")[0] + message.split("</wsnt:Package>")[1];
}else{
    //解析该分包的id，确认其属于哪一条消息
    String hashCode = message.split("<Identification>")[1].split("</Identification>")[0];
    System.out.println("hashcode: " + hashCode);
    //判断这条消息是否正在组包，否的话就启动组包过程
    System.out.println("true or false? " + map.containsKey(hashCode));
    if(!map.containsKey(hashCode)){
        NotificationBuilder nb = new NotificationBuilder();
        map.put(hashCode, nb);
    }
    //获取组包对象
    NotificationBuilder tempNb = map.get(hashCode);
    //将消息递交给组包对象
    tempNb.setTempMessage(message);
    //拆掉分包
    tempNb.breakMessage();
    //解析消息
    tempNb.parse();
    //判断是否获取到一条消息的所有分包，是则组包
    if(tempNb.isReadyToBuild()){
        notification = tempNb.build();
        map.remove(hashCode);
    }
}
```

图 5-16 组包逻辑

从代码中可以看出，组包模块收到从路由递交的消息之后，首先解析出<wsnt:Package>标签，其中包括<Fragment>标签和<Identification>标签，<Fragment>标签中携带拆包信息，<Identification>标签中的信息可以唯一标识一条未经拆包的消息，而对于一条消息被拆分后的若干个分包，<Identification>标签的内容是一样的。

解析出<wsnt:Package>标签中的所有内容后，程序会对当前消息的性质作出判断，若是一条完整的消息或者允许向用户递交不完整包，则在去掉<wsnt:Package>标签的信息后直接调用推送端点向用户推送消息，否则启动组包流程。在这里解释一下为什么允许向用户递交不完整包，在实际应用中，我们发现，对于某些本身较小但是数量很多的通知消息，发布者为了提高性能，在满足实时性要求的前提下，往往选择将多条小消息打包成较大的消息。假设将 1000 条 SQL 语句打包成一条消息，其总大小一般会超过 8k，则必然发生拆包，在接收端，向用户递交不完整包，并非没有意义，而只是损坏了拆包处的一条 SQL 语句，其他 999 条 SQL 语句都是完好无损的，这种情况在很多应用中是可以被接收的，因此，我们向发布者用户提供这种参数，让用户根据发送消息的不同性

质可以选择递交策略。

在组包流程中，首先根据<Identification>判断该分包属于哪一条消息，若这条消息正在组包，则将这个分包加入进去，否则为这条消息启动一个新的组包过程。每向一个组包过程中添加一个分包，都会检查该条消息的组包是否完成，即当前消息是否获得了所有分包，如果是，则将所有的分包打包成一条完整的消息，否则继续等待下一个分包。

5.3.3.5 元数据管理模块

5.3.3.5.1 详细设计

元数据，就是用来描述数据的数据，主要是描述数据属性的信息，也被称为诠释数据。举例来说，你要注册成为某个网站的会员，该网站会要求你提供昵称、用户名、手机号码、密码、邮箱等信息，这些就是一组元数据，用以描述用户的属性。元数据无处不在，有一类事物，就可以定义一套元数据，其最大的好处在于使得信息的描述和分类可以实现格式化，从而为机器处理创造了可能。

在本文论述的 HPwsn 系统中，也存在这样的元数据，其中最重要的一套元数据就是订阅信息。如何存储系统中的订阅信息，能够实现快速的查找和匹配，同时又能向用户提供丰富的表达自身兴趣的能力，是 HPwsn 系统必须解决的问题。

现有的基于 wsn 组件的发布订阅系统中，订阅条件的匹配是基于简单主题的，发布者发布的每一条消息都有其对应的主题，若该主题与订阅者描述的兴趣完全匹配，则系统将这条消息推送给订阅者。从实现上来说，采用订阅表结构，如图 5-17 所示。对于订阅/取消订阅请求，涉及到元数据处理模块的操作仅仅是修改相应主题对应的订阅者列表即可；对于通知消息，首先获取相应主题对应的订阅者列表，然后遍历该列表，向每一个订阅者推送消息。

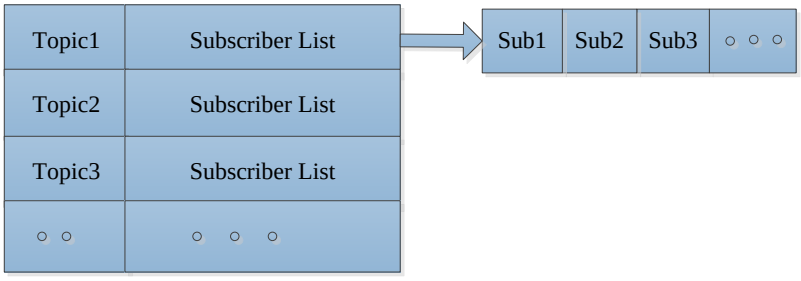

图 5-17 订阅表

这样的主题组织方式优点是实现简单，匹配效率高，但是却不能满足实际环境中大批量、层次化的主题订阅需求。

举例说明，假设系统中定义了主题 Fruit、Apple、Pear、Grape，若订阅者 A

对这 4 个主题均感兴趣，那么 A 需要分别订阅这 4 个主题，系统收到订阅请求后，将 A 添加到每个主题对应的订阅者列表中，以便相应主题的消息到达时向 A 推送。在实际环境中，系统中主题的数量往往很多，单个订阅者感兴趣的主题也很多，这时订阅过程会变得非常繁琐。同时，我们注意到订阅者的订阅行为不是毫无规律的，某一个订阅者一般会对一类或者几类主题感兴趣，而每一类主题之间往往存在着层次化的归属关系。

综合考虑上述情况，本文论述的 HPwsn 发布订阅系统采取主题分层次组织的策略，将所有主题组织成一棵主题树，如图 5-18 所示：

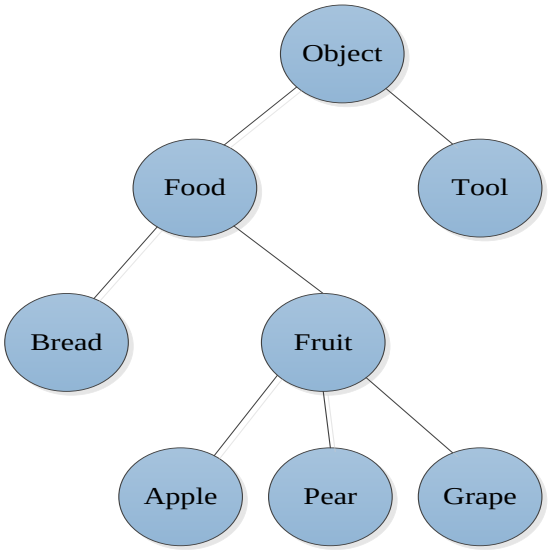

图 5-18 主题树

在 HPwsn 系统中，我们使用主题树来组织所有的主题。原因在于：首先，主题树提供了一种结构化的名字系统，基于主题树订阅者可以实现组订阅，即订阅者可以通过订阅整棵主题树或者任意一棵子树来实现一次订阅多个主题；其次，主题树还可以使相互关联的主题组织到一起，更便于管理。

举例说明，如图 5-18 所示，Food 和 Tool 都是 Object 的子主题，Bread 和 Fruit 是 Food 的子主题，而主题 Fruit 又有子主题 Apple、Pear 和 Grape。若订阅者 A 对所有 Fruit 类别的主题感兴趣，只需要订阅主题 Fruit，系统便会将与 Fruit 及其所有子出题 Apple、Pear、Grape 相关的消息全部推送给订阅者 A。

主题树中的每一个节点的存储结构如图 5-19 所示：

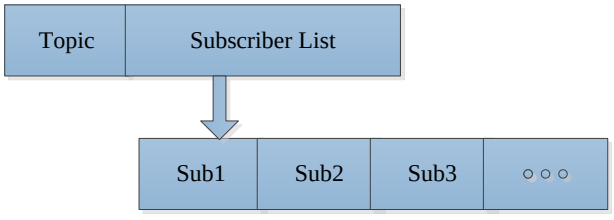

图 5-19 主题树节点

节点存储的关键字是主题，每个主题对应一个订阅者列表，其中存储了所有订阅这个主题的订阅者。订阅消息到达系统中时，首先匹配到相应的主题节点，遍历以该节点为根的整棵树，在其中每一个节点的订阅者列表中添加该订阅者；发布消息到达系统中时，同样首先匹配到相应的主题节点，然后遍历该节点的订阅者列表，向其每一个订阅者推送消息。

主题描述格式为 A:B:C，每层主题以冒号隔开。主题节点匹配算法如 Algorithm1 所示：

下面我们简要分析一下订阅、发布过程的时间复杂度。假设系统中所有的主题数量为 N，并且处理每一个主题树节点花费的时间为 $O(1)$。在订阅消息的处理过程中，首先要在主题树中匹配当前订阅的主题，采用 Algorithm1 所示的算法，花费的时间为 $O(\log_a N)$（a 的值取决于主题树中每个节点的平均孩子节点数量），设匹配到的节点为 topicNode，然后，遍历以 topicNode 为根的主题树子树，向其每一个节点的订阅者列表中添加当前订阅者，由于 topicNode 在主题树中的位置不同，该过程花费的时间差异也很大，若 topicNode 是主题树的根节点，则

Algorithm1:TopicMatch

Input:topicPath[], root
Output:topicNode

```
1   topicNode←root
2   for i←0 to length[topicPath]-1
3     if topicNode.topicName=topicPath[i]
4       for j←0 to length[topicNode.childList]-1
5         if topicNode.childList[j].topicName=topicPath[i+1]
6           topicNode←topicNode.childList[j]
7           break
8         else
9           then error "match faild."
10    else
11      then error "match faild."
```

该过程需要遍历整个主题树，若 topicNode 是叶子节点，则只要向 topicNode 的订阅者列表中添加当前订阅者即可，平均下来该过程时间复杂度为 $O(N)$。综上所述，订阅过程的时间复杂度为 $O(\log_a N)+O(N)=O(N)$。

在发布消息的处理流程中，首先也要在主题树中匹配当前发布的主题，花费的时间为 $O(\log_a N)$，然后，遍历该节点的订阅者列表，向每一个订阅者推送消息，按照前面假设的处理每一个主题树节点花费的时间为 $O(1)$，该过程时间复杂度为 $O(1)$。综合起来，发布过程的时间复杂度为 $O(\log_a N)+O(1)=O(\log_a N)$。

在上述设计中，我们为了尽可能地优化系统处理通知消息的性能，将遍历主题树的任务放在了订阅过程中，由于订阅消息的数量远远小于发布消息的数量，因此这种设计是合理的，能够大大提高系统整体的性能。

5.3.3.5.2 主题树节点的定义。

在HPwsn系统中，我们使用WSNTopicObject类来定义主题树中的一个节点，该类有四个域：

• private TopicEntry topicEntry：用于存储当前节点的主题；

• private WSNTopicObject parent：用于存储当前节点的父亲节点；

• private List< WSNTopicObject > childrens：用于存储当前节点的所有孩子

• private List<String> subscribeAddress：用于存储订阅了该主题的所有订阅者的消息接收服务的地址。

其中 TopicEntry 类是用来定义一个主题的。该类也有四个域，分别为：

• private String topicName：存储主题的名字

• private String topicCode：存储主题的编码，系统会根据主题在主题树中的位置对每一个主题进行编码，以便更高效地进行查找和匹配；

• private String topicPath：该域用于存储主题在 OpenLDAP 数据库中的目录路径，便于数据存取；

• private WsnPolicyMsg wsnpolicymsg：该域存储了用户为这个主题设置的策略，其配置界面可以参见图 4-11。

5.3.3.5.3 主题树加载

主题树的加载过程在 5.2.2 节 WsnProcessImpl 的实现部分有详细的说明，此处不再赘述。

5.3.3.5.4 主题匹配

不管是订阅消息，还是通知消息，到达系统中之后，都要首先解析其绑定的主题，然后在主题树中匹配到该主题，才能执行下一步操作。匹配算法如 5.3.3.5.1 节中 Algorithm1 所示。

一个主题名字的结构是以冒号隔开的，每一部分对应主题树中的一级主题，类似于描述了该主题在主题树中的路径。匹配算法的输入是将主题名字按冒号分词之后的数组，以及主题树的根节点。

先拿匹配成功的情况来说，外层循环遍历主题分词后的数组中每一个元素，每遍历一个元素，判断数组中元素是否与主题树该层的主题匹配，若匹配则在主题树中向下走一层，即遍历当前节点的所有孩子，与数组中的下一个元素匹配，以此类推，直至完全匹配成功。若在这个过程中出现不匹配的情况，则报异常，提示需要匹配的主题并不在主题树中。

元数据管理模块的类图如图 5-20 所示。

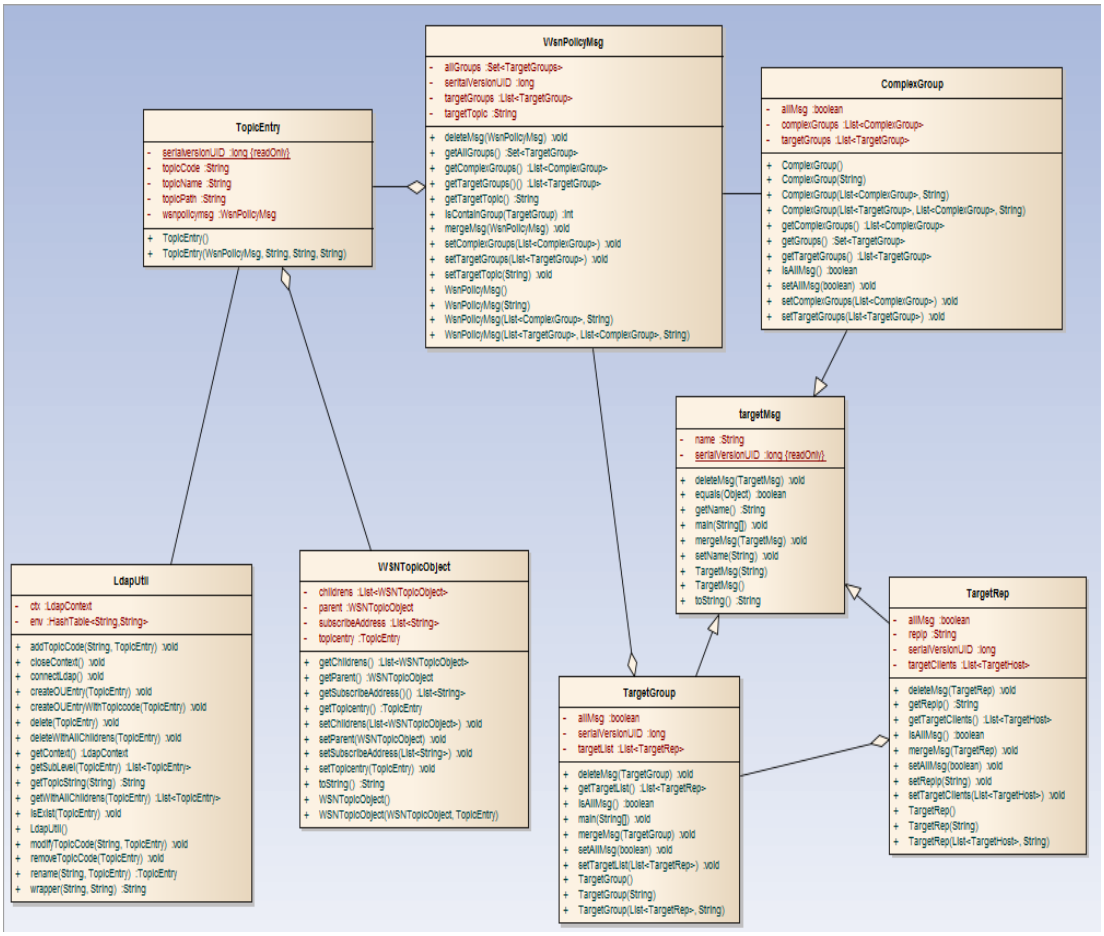

图 5-20 元数据管理模块类图

5.4 外围模块实现

在整个 HPwsn 系统的构成中，不仅仅只有核心发布订阅模块，从图 4-3 系统架构图中可以看出，除了核心系统之外，还有另外四个模块也是 HPwsn 非常重要的组成部分，分别是发布者、订阅者模块，客户端模块，性能监控与管理模块，以及元数据的定义、展示、持久化存储模块。本节就这些模块的详细设计和具体实现进行展开讨论。

5.4.1 发布者、订阅者实现

订阅者和发布者模块是由用户实现的，HPwsn 系统对其不做过多限制。对于一个订阅者，原则上只要其创建了本地消息接收服务，并调用客户端方法将服务地址告知系统，就可以实现订阅；发布者更加简单，只需调用客户端提供的 notify 方法将通知消息及其对应主题发送到系统中即可。

由于发布者和订阅者的实现不一，在此不再给出详细实现。

5.4.2 客户端模块

客户端模块，以 jar 包的形式发布，用户可以使用该包中提供的方法快速构建发布者或订阅者程序。客户端中的类封装了 createPullPoint、subscribe、unsubscribe、notify 等方法，这些方法分别用于构建不同的消息体，最后利用 HttpClient 将消息发送出去。

以 SendWSNCommand 类为例，该类包含三个域：

- wsnAddress：用于存储 HPwsn 系统所在主机的地址；
- endpointAddr：用于存储 createPullPoint 请求返回的推送端点的地址；
- localServiceAddr：用于存储本地消息接收服务的地址。

该类封装了 createPullPoint、subscribe、unsubscribe、notify 四个方法。以 subscribe 方法为例，该方法主要完成对订阅消息的封装，该函数接受一个 String 参数，即本次订阅对应的主题，但是在封装消息的时候会封装三个参数，分别为 endpointAddr、订阅主题和 localServiceAddr。

在封装消息的过程中，首先会将三个参数封装在规范的 wsn 消息体中，然后根据 HPwsn 系统暴露的 Web Service 接口，包裹上 SOAP 消息的格式。封装消息完成后，subscribe 方法会调用 send 方法，在 send 方法中，封装一个 HTTP POST 方法，最后将消息发送出去。

在客户端模块，有两点需要说明。第一，对每一个订阅，都要创建一个 SendWSNCommand 类的实例，因为 endpointAddr 是作为该类的域存储的，如果用一个 SendWSNCommand 类的实例执行多次订阅，那么 endpointAddr 信息会被覆盖，导致订阅失败。第二，若想向用户提供更丰富的客户端接口，或者使用另外的技术实现消息传输，可以扩展 SendWSNCommand 类，重写其相应方法即可。

5.4.3 性能监控与管理模块

4.5.3 节中提到，我们最终选用 JDK 自带的 Java VisualVM 工具来实现系统的性能监控与管理模块。Java VisualVM 程序放置在%JAVA_HOME%\bin 目录下，名字为 jvisualvm.exe。启动之后可以看到如图 5-21 示的界面。

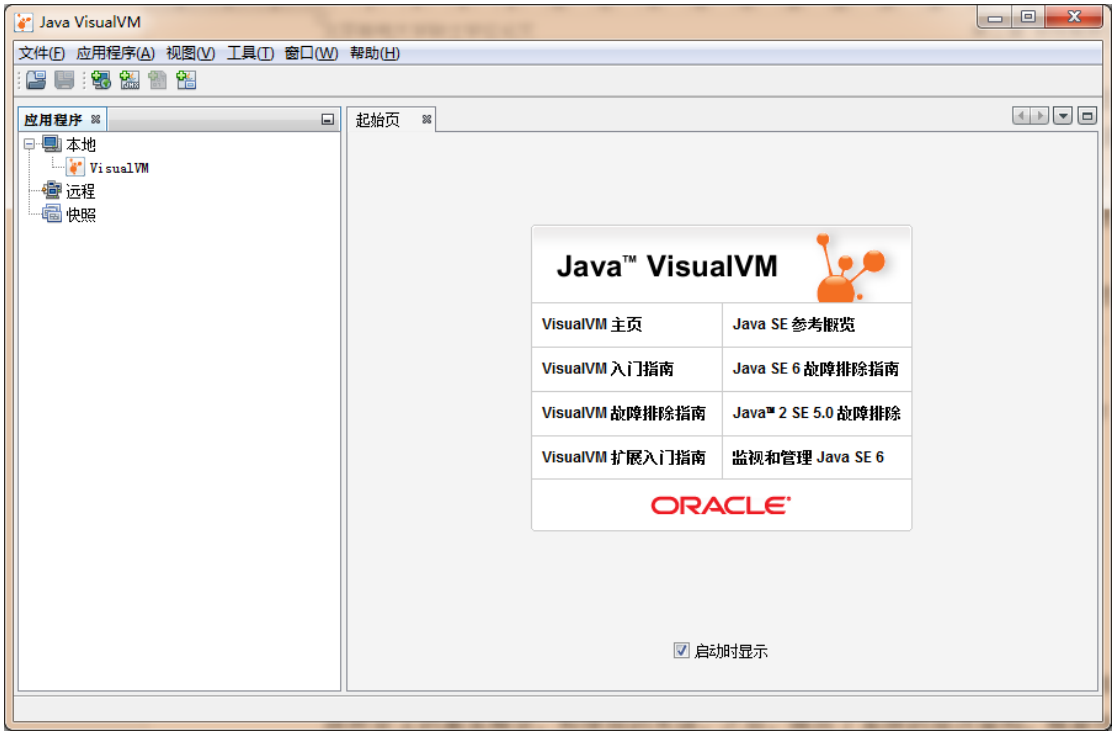

图 5-21 Java VisualVM 界面

在左侧侧边栏中，本地标签下会列出所有本地运行的 Java 程序，双击任何一个都可以查看该程序的运行情况，在本例中双击 VisualVM，可以看到如图 5-22 所示的界面：

可以看到，这个界面分为四个部分，分别以动态图的形式给出当前程序的 CPU 使用情况、内存使用情况、类信息以及线程信息。一个可以长期稳定运行的程序，其 CPU 占用、内存占用以及线程信息一般来讲，是一条有波动，但是总体趋于水平的曲线，尤其是内存和线程，受机器上运行的其他程序的影响较小，如果曲线呈上升趋势，说明程序中必然有隐藏的 bug。

Java VisualVM 不仅可以监控本地运行的 Java 程序，还可以监控远程程序，

图 5-22 Java 程序监控界面

但是需要在远程机器上运行 jstatd 守护程序。这个功能也为我们监控分布式 Web 集群中的多台机器提供了便利。

通过这种方式，我们实现了 HPwsn 系统的监控与管理模块，无论是对于程序的测试开发，还是部署之后的运行情况监控，都有很重要的意义。

5.5 本章总结

本章讲述了 HPwsn 系统的详细设计与具体实现，从数据格式的实现到独立部署的实现，再到各个具体模块的实现，通过阅读本章，读者可以从细节上完全了解 HPwsn 系统是如何构建的。HPwsn 系统的实现基本完成了概要设计中提到的各个方面，满足了设计需求。在下一章中，我们将针对 HPwsn 的新功能和性能编写测试用例，对 HPwsn 系统进行测试，以观察系统是否能够适应真实的生产环境。

第六章　系统测试

本章对实现的 HPwsn 系统进行测试。测试分为两部分进行，首先测试系统功能是否都已正确实现，其次，测试系统性能是否达到预期。本章主要针对这两个测试需求编写测试用例，得出测试数据并对数据进行分析，从而得出对系统的综合评价。

6.1 测试环境

HPwsn 是一个主要应用于分布式 Web 环境的发布订阅系统，因此我们的测试环境也是分布式的 Web 环境。条件所限，我们选择 3 台虚拟机进行测试，他们分别为 HPwsn01、HPwsn02 和 HPwsn03。

6.1.1 硬件环境

运行虚拟机的服务器配置如下：

1）IBM System x3850 X5 Server；

2）32G 内存；

3）16 核 CPU；

4）1T 硬盘。

我们在这台服务器上部署了 HPwsn01、HPwsn02 和 HPwsn03 这 3 台虚拟机，每台虚拟机的配置如下：

1）2G 内存；

2）3GHZ CPU；

3）10G 硬盘；

4）千兆网卡。

6.1.2 软件环境

由于 HPwsn 系统绝大部分代码都使用 Java 语言开发，因此必须部署 Java 运行环境。每台虚拟机的软件换件如下：

1）windows7 专业版操作系统；

2）JDK1.6 update 35

6.2 HPwsn 系统部署

操作系统和 Java 环境的安装在此不予赘述，安装好之后，需要部署的程序有：管理员程序、ActiveMQ 程序、HPwsn 系统、发布者和订阅者测试程序。

6.2.1 管理员程序

由于我们采用 3 台虚拟机，每台虚拟机部署一个 HPwsn 系统，构成了一个分布式 Web 环境，因此在系统中需要一个管理员程序，负责维护网络拓扑。管理员程序只需要部署在其中一台虚拟机上，HPwsn 系统通过配置指向管理员所在的虚拟机，向其注册。管理员程序的目录如图 6-1：

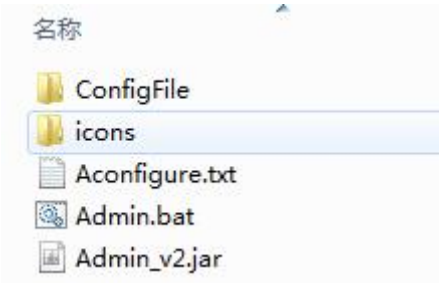

图 6-1 管理员程序部署

在 Aconfigure.txt 文件中配置管理员所在虚拟机的 IP 地址以及通信使用的各个端口之后，双击 Admin.bat 就可以启动管理员程序。管理员程序启动之后，才可以启动 HPwsn 系统。

6.2.2 ActiveMQ 程序

由于 HPwsn 系统底层通信依赖 JMS 机制，因此要使 HPwsn 系统能够正常运行，必须首先启动 JMS 的一种开源实现 ActiveMQ 程序（当然，也可以选择 JMS 的其他实现）。ActiveMQ 的部署目录如图 6-2 所示。

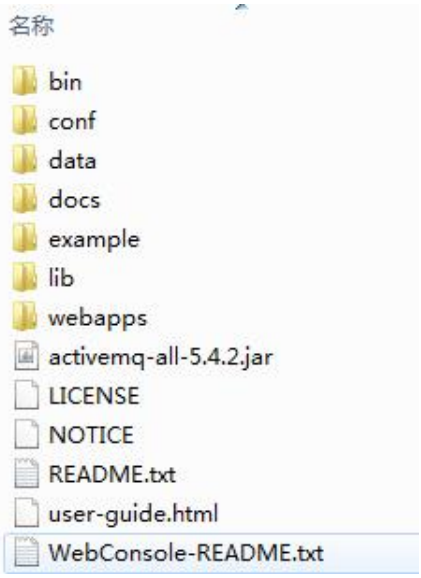

图 6-2 ActiveMQ 部署目录

将该目录拷贝到每一台虚拟机，无需任何配置，直接双击 bin 目录下的 activemq.bat 文件启动即可。

6.2.3 HPwsn 系统

由于 HPwsn 系统实现了独立部署，因此其部署变得非常简单，就是一个 jar 包和一个配置文件 configure.txt，在配置文件中需要配置的项为：本机地址，管理员地址，通信端口、分包发送大小以及自身 HPwsn 属于哪个集群。双击运行 jar 包即可启动 HPwsn 系统。

6.2.4 发布者和订阅者测试程序

我们会针对不同的测试需求编写不同的发布者和订阅者测试程序。

6.3 功能测试

首先，我们编写测试用例测试 HPwsn 系统的功能是否都已正确实现。在启动 HPwsn 系统前，默认启动了管理员程序和 ActiveMQ 程序，若无特殊情况，测试用例中不再说明。

6.3.1 测试用例

测试编号：功能-01
测试项目：独立部署单点功能测试
测试目的：测试 HPwsn 系统在单点独立部署环境下是否能正常递交消息
测试用例： （1）启动一台虚拟机的 HPwsn 系统； （2）启动一个订阅者程序，订阅主题 ALL（ALL 主题在主题树中）； （3）启动一个发布者程序，以 ALL 为主题发布一条通知消息。
预期结果： （1）向管理员注册成功，各个启动过程不报错； （2）订阅者能够收到以 ALL 为主题的通知消息。

测试编号：功能-02
测试项目：独立部署集群功能测试
测试目的：测试 HPwsn 系统在集群独立部署环境下是否能正常递交消息
测试用例： （1）启动全部 3 个 HPwsn 系统，分属 3 个集群； （2）启动 3 个订阅者，分别向 3 个 HPwsn 系统订阅，订阅 ALL1、ALL2、ALL3 这 3 个主题；

（3）启动 3 个发布者，分别以 ALL1、ALL2、ALL3 为主题向 3 个 HPwsn 系统发布通知消息；
预期结果： （1）3 个 HPwsn 系统向管理员注册均可成功，各个启动过程不报错； （2）每个订阅者能够收到全部 3 个主题的通知消息。

测试编号：功能-03
测试项目：分包发送双节点功能测试
测试目的：测试分包发送模块在双节点环境下是否能在数据包过大时采取分包发送
测试用例： （1）在配置文件中配置分包大小； （2）启动 HPwsn01 和 HPwsn02，启动一个订阅者，向 HPwsn01 订阅主题 ALL； （3）启动一个发布者，发布的通知消息以 ALL 为主题，大小需大于配置文件中的分包大小，向 HPwsn02 发布通知消息； （4）分别按允许递交断包和不允许递交断包发送通知消息； （5）若上述过程没有问题，则使通知消息的大小分别 2、5、10、100 倍于分包大小，重复（4）。
预期结果： （1）在整个过程中，订阅者都能够正常收到消息； （2）若允许递交断包，则订阅者收到的是拆包后的消息； （3）若不允许递交断包，则订阅者收到的是整条消息。

测试编号：功能-04
测试项目：分包发送集群功能测试
测试目的：测试分包发送模块在集群环境下是否能在数据包过大时采取分包发送
测试用例： （1）在配置文件中配置分包大小； （2）启动全部 3 个 HPwsn 系统，分属 3 个不同集群； （3）启动 3 个订阅者，向 3 个 HPwsn 系统订阅，每个订阅者都订阅 ALL1、ALL2、ALL3 这 3 个主题； （4）启动 3 个发布者，分别以 ALL1、ALL2、ALL3 为主题，向 3 个 HPwsn 系统中发布通知消息，消息大小需大于配置文件中的分包大小； （5）分别按允许递交断包和不允许递交断包发送通知消息； （6）若上述过程没有问题，则使通知消息的大小分别 2、5、10、100 倍于

分包大小，重复（5）。
预期结果： （1）在整个过程中，订阅者都能够正常收到全部 3 个主题的通知消息； （2）若允许递交断包，则订阅者收到的是拆包后的消息； （3）若不允许递交断包，则订阅者收到的是整条消息。

测试编号：功能-05
测试项目：树状名称表达单点功能测试
测试目的：测试树状名称表达在单点环境下是否能正常实现主题的订阅和查询
测试用例： （1）生成订阅主题树； （2）启动 1 个 HPwsn 系统； （3）启动两个订阅者，其中一个订阅的主题在主题树中是另外一个订阅主题的祖先主题； （4）启动两个发布者，一个使用祖先主题，一个使用子孙主题。
预期结果： （1）订阅了祖先主题的订阅者，它可以收到这两个通知的所有消息；而订阅了子孙主题的订阅者只能接受到子孙主题的通知消息。

测试编号：功能-06
测试项目：树状名称表达集群功能测试
测试目的：测试树状名称表达在集群环境下是否能正常实现主题的订阅和查询
测试用例： （1）生成订阅主题树； （2）启动全部 3 个 HPwsn 系统，分属 3 个不同集群； （3）启动 3 个订阅者，分别向 3 个 HPwsn 系统订阅，订阅的主题分别是 ALL、ALL：ALARM、ALL：ALARM：ALARM1，其中 ALL 是 ALL：ALARM 的父主题，而 ALL：ALARM 是 ALL：ALARM：ALARM1 的父主题； （4）启动 3 个发布者，分别使用 ALL、ALL：ALARM、ALL：ALARM：ALARM1 这 3 个主题向 3 个 HPwsn 系统发布消息。
预期结果： （1）订阅了 ALL 主题的订阅者可以收到所有通知消息； （2）订阅了 ALL：ALARM 主题的订阅者可以收到以 ALL：ALARM 和

ALL：ALARM：ALARM1 为主题的所有通知消息； （3）订阅了 ALL：ALARM：ALARM1 主题的订阅者只能收到以 ALL：ALARM：ALARM1 为主题的通知消息。

6.3.2 测试说明及结果分析

在上面的测试过程中，我们发现了系统一些细节处的 bug 并予以修复，截止到本文撰写前，HPwsn 系统已经全部正确实现了上述功能，功能测试 case 全部通过。

6.4 性能测试

HPwsn 系统是一个高性能的发布订阅系统，因此对其进行性能测试非常重要，性能测试的结果在很大程度上可以反映出我们这个系统设计的成效。对于性能测试，我们主要关注系统单位时间内的消息吞吐量，因此不再分模块测试，而是作为一个整体测试 HPwsn 系统的性能。

在性能测试的过程中，由于 HPwsn 系统的消息吞吐量较高，因此在屏幕上打印信息也会对测试结果构成较大的影响，我们在订阅者程序中采用收到 500 条消息才打印一个计数器和时间戳的方式进行测试。

我们采用系统每秒钟处理消息的数量（MPS，Message Per Second）来衡量系统的性能，其计算公式为：

$$MPS = \frac{\sum_{t=start}^{current} notification_t}{current - start}$$

6.4.1 主题树性能测试

使用主题树结构实现树状名称表达是 HPwsn 系统的一个核心功能，主题树的性能对整个系统性能的影响也是比较大的，因此，我们专门针对主题树部分做了性能测试，测试用例如下。

6.4.1.1 测试用例

测试编号：性能-01
测试项目：主题树性能测试
测试目的：测试随着主题树规模在深度和宽度上的扩展，系统性能受到的影响
测试用例： （1）生成主题树，如图 6-3 所示； （2）启动 1 个 HPwsn 系统；

（3）启动一个订阅者，订阅主题 ALL:ALARM，每收到 500 条通知消息，则打印计数器及时间戳； （4）启动一个发布者，以 ALL:ALARM 为主题向 HPwsn 系统无间断发布通知消息，此为情况一； （5）将主题换成 ALL:ALARM:ALARM1:ALARM11:ALARM111，重复（3）、（4），此为情况二； （6）将主题换成 ALL:ALARM:ALARM1:ALARM11:ALARM115，重复（3）、（4），此为情况三。
结果分析： （1）记录订阅者每收到 500 条通知消息的计数器及时间戳信息，计算系统平均意义上的吞吐量。

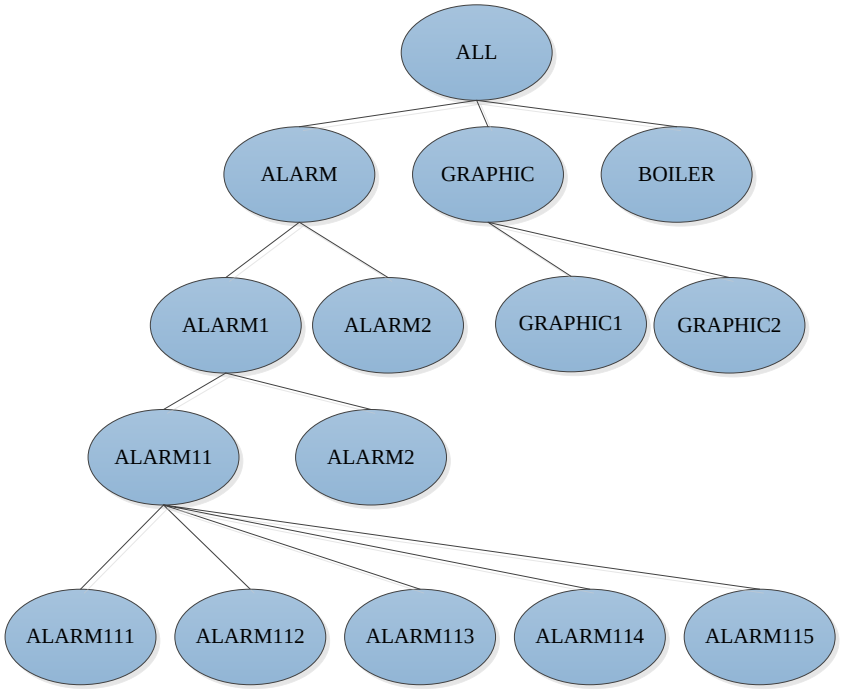

图 6-3 性能-01 测试用例主题树示意图

6.4.1.2 测试说明及结果分析

在对主题树进行性能测试的过程中，我们分三种情况，情况一与情况二的不同在于订阅主题在主题树中的深度不同，情况二与情况三中订阅者订阅的主题在主题树中具有同样的父主题，但是在主题树的实现中，一个主题的所有子主题是存储在线性表中的，因此匹配时要遍历线性表，设置这两种测试用例的目的是测试遍历线性表的过程对主题树性能的影响。

三种情况的测试结果如图 6-4~图 6-6 所示。

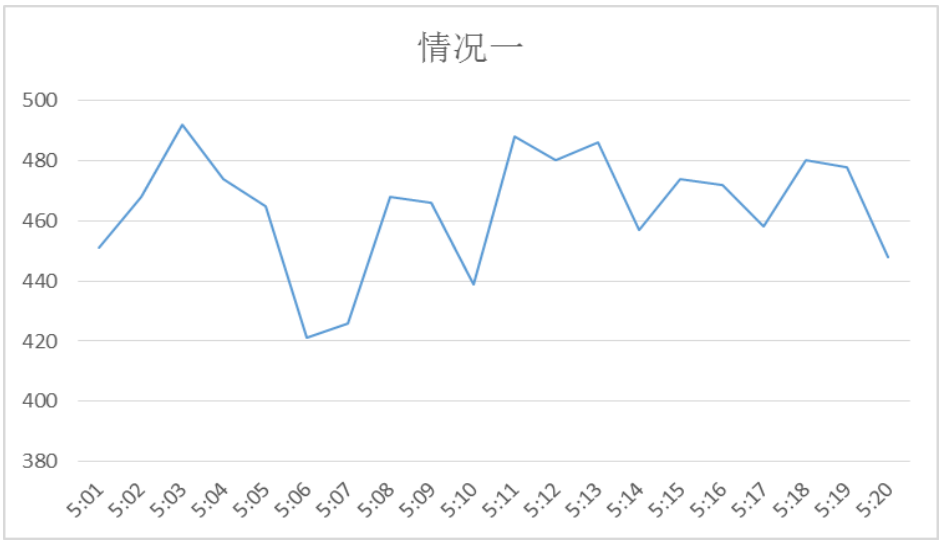

图 6-4 情况一 MPS

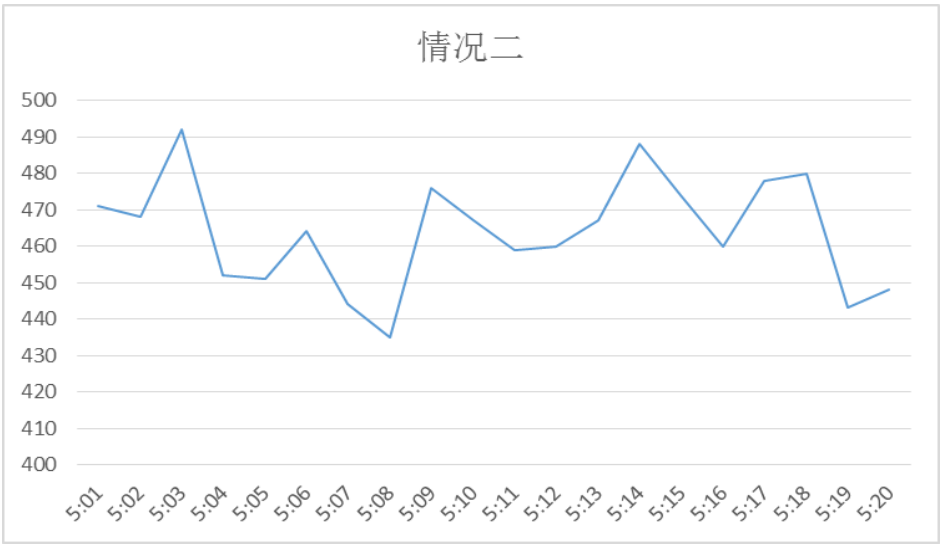

图 6-5 情况二 MPS

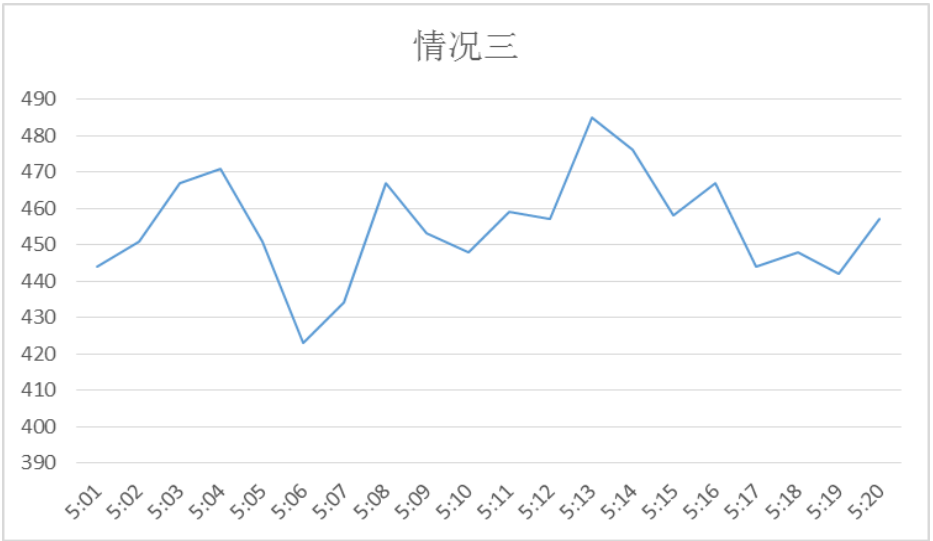

图 6-6 情况三 MPS

从图 6-4~图 6-6 中可以看出，情况一与情况二对比，系统在性能上基本没有

什么变化，MPS 平均值在 460 左右，情况二与情况三对比，性能有微小的降低。测试结果表明，对于主题树在较小数量级上的规模扩展，系统的性能基本上不受影响；另外由于主题树的树状结构，它能够在深度不大，每个主题存储子主题的线性表规模不大的前提下存储大量数据，例如一个 10 层的主题树，每个主题最多有 10 个子主题，即可存储10^9量级的主题数据，而在我们平时的应用中是远远没有达不到这样规模的，因此，我们可以得出结论，随着主题树一定规模的横向和纵向扩展，HPwsn 系统的性能不会受到明显的影响。

6.4.2 分包发送性能测试

分包发送也是 HPwsn 系统一个重要的功能，而且是可能影响系统性能的一个因素，因此，我们针对分包发送单独进行性能测试。

6.4.2.1 测试用例

测试编号：性能-02
测试项目：分包发送性能测试
测试目的：测试分包发送功能对 HPwsn 系统性能的影响
测试用例： （1）生成主题树，配置分包大小为 3000； （2）启动 1 个 HPwsn 系统； （3）启动一个订阅者，订阅主题 ALL:ALARM，每收到 500 条通知消息，则打印计数器及时间戳； （4）启动一个发布者，以 ALL: ALARM 为主题向 HPwsn 系统无间断发布通知消息，通知消息的大小为 3000，刚好不需要分包，此为情况一； （5）将通知消息的大小改为 30000，为分包大小的 10 倍，并允许递交断包，重复（3）、（4），此为情况二。
结果分析： （1）记录订阅者每收到 500 条通知消息的计数器及时间戳信息，计算系统平均意义上的吞吐量。

6.4.2.2 测试说明及结果分析

该测试用例的主要目的在于测试拆包函数是否对系统性能构成重大影响，对于不允许递交断包的情况，受网络环境影响较大，不易控制变量，实现对比，在此不予测试。

情况一与情况二的测试结果如图 6-7 和 6-8 所示。

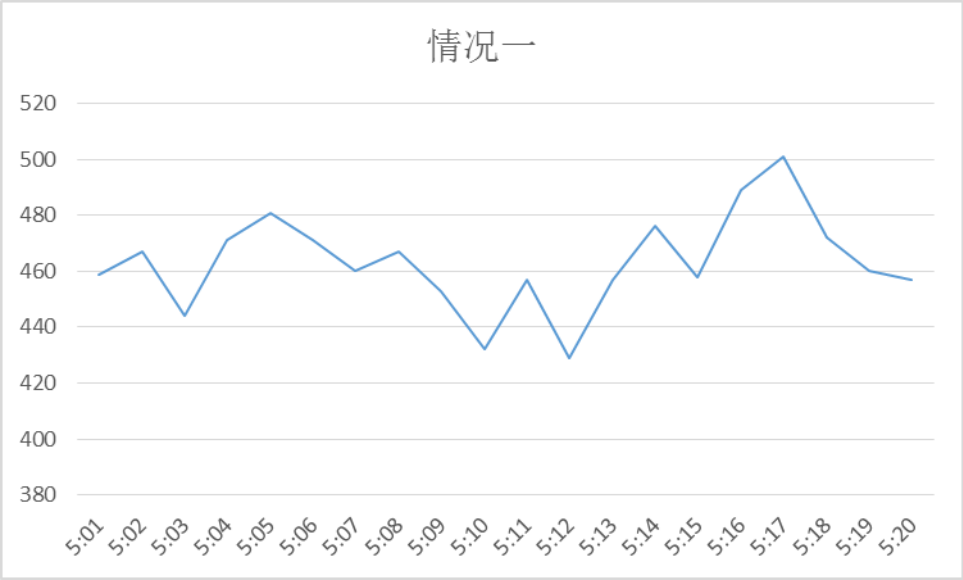

图 6-7 分包发送情况一 MPS

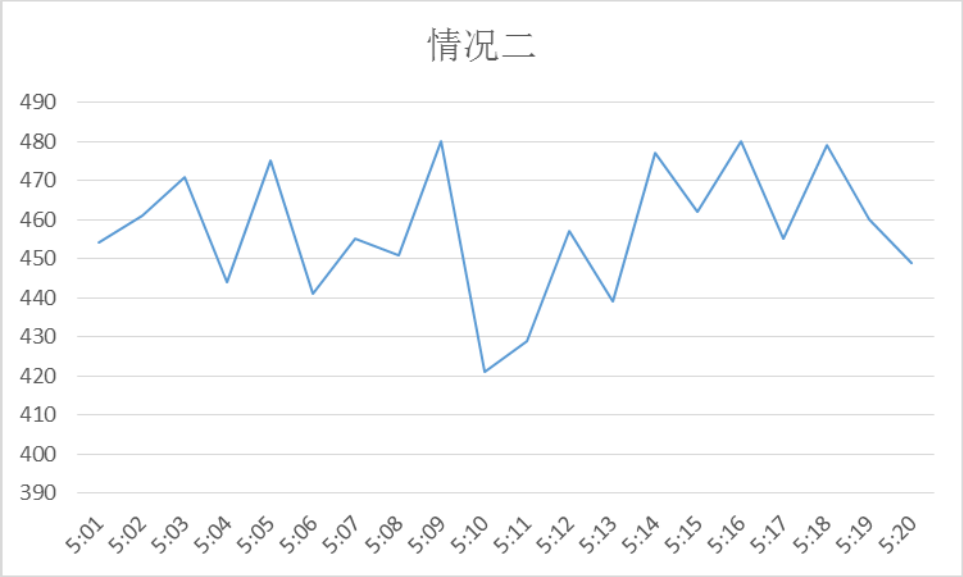

图 6-8 分包发送情况二 MPS

从图中可以看出，拆包过程并没有对系统性能构成重大影响，平均 MPS 只是有略微降低，但都是在 460 左右波动，可以认为拆包过程对系统整体性能影响是非常微小的。

6.4.3 对比旧版 wsn 系统

为说明 HPwsn 系统在性能优化上作出的贡献，我们对旧版的主动推送型发布订阅系统也做了性能测试，结果如图 6-9 所示。

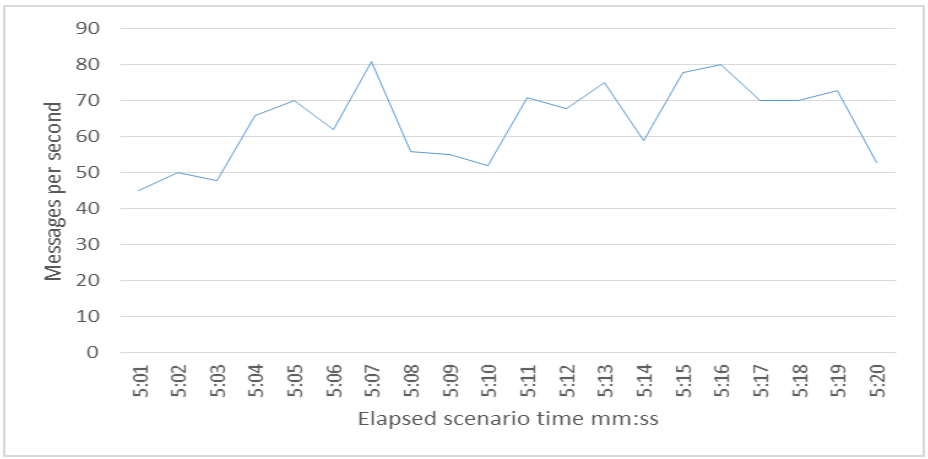

图 6-9 旧版 wsn 系统 MPS

可以看到，当发布者无间隔发布通知消息时，其性能在每秒钟吞吐 60 包数据上下。从测试结果的对比可以看出，HPwsn 系统大大提升了基于 wsn 组件的发布订阅系统的性能。

6.5 稳定性测试

投入真实生产环境中时，系统能不能长时间稳定运行是衡量一个系统可用性的重要指标。本文论述的 HPwsn 系统，在测试阶段针对长时间运行的稳定性做了测试。

6.5.1 测试用例

测试编号：稳定性-01
测试项目：系统稳定性测试
测试目的：测试 HPwsn 系统长时间稳定运行的能力
测试用例： （1）生成主题树； （2）启动 1 个 HPwsn 系统； （3）启动一个订阅者，订阅主题 ALL； （4）启动一个发布者，以 ALL 为主题向 HPwsn 系统不间断发布通知消息； （5）使用性能监控与管理模块监控 HPwsn 系统、发布者、订阅者程序的运行情况。
结果分析： （1）记录性能监控与管理模块的监控结果。

6.5.2 测试说明及结果分析

HPwsn 系统在满负荷处理发布消息时的监控结果如图 6-5 至 6-11 所示。

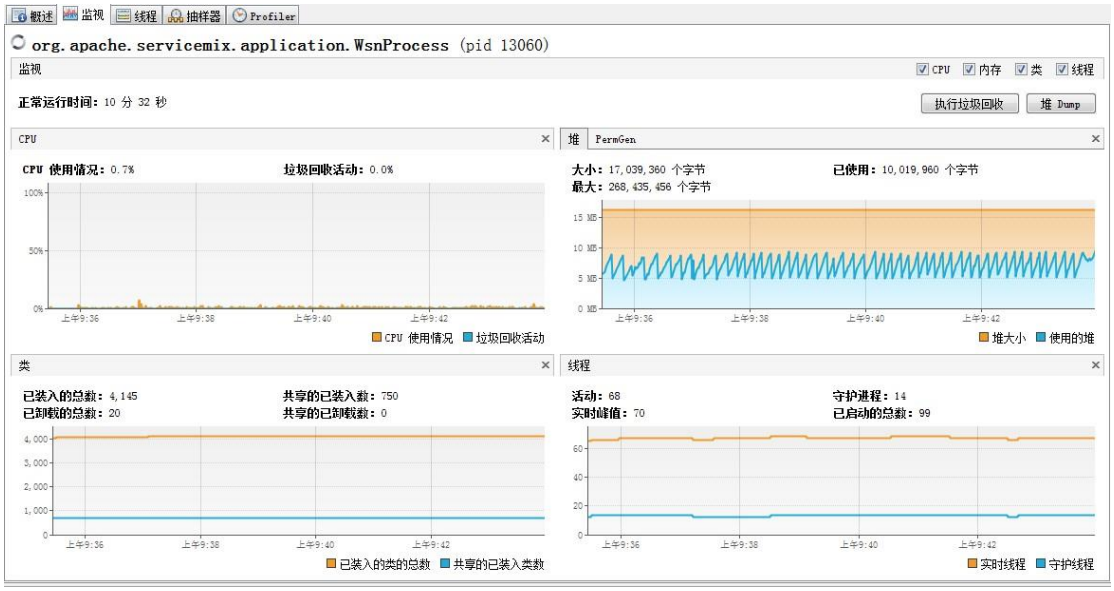

图 6-5 HPwsn 系统总体监控结果图

图 6-6 HPwsn 系统 CPU 使用情况图

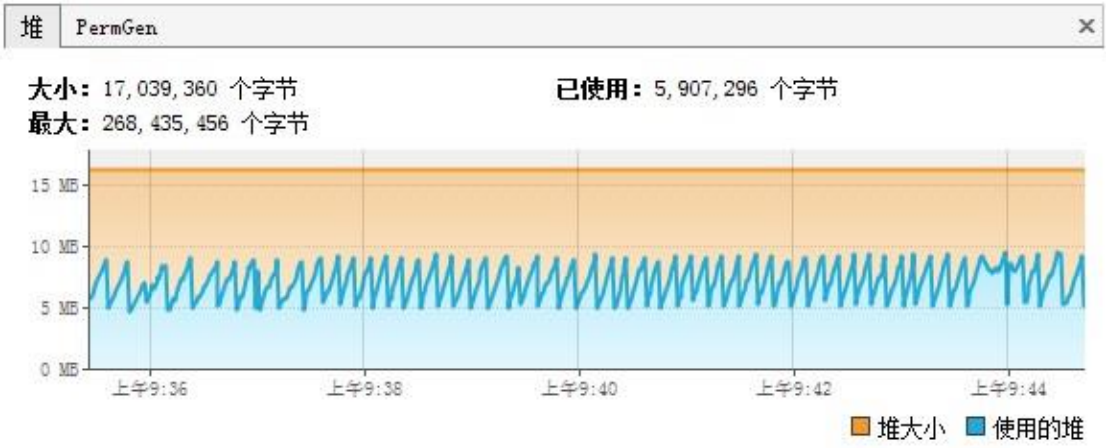

图 6-7 HPwsn 系统内存使用情况图

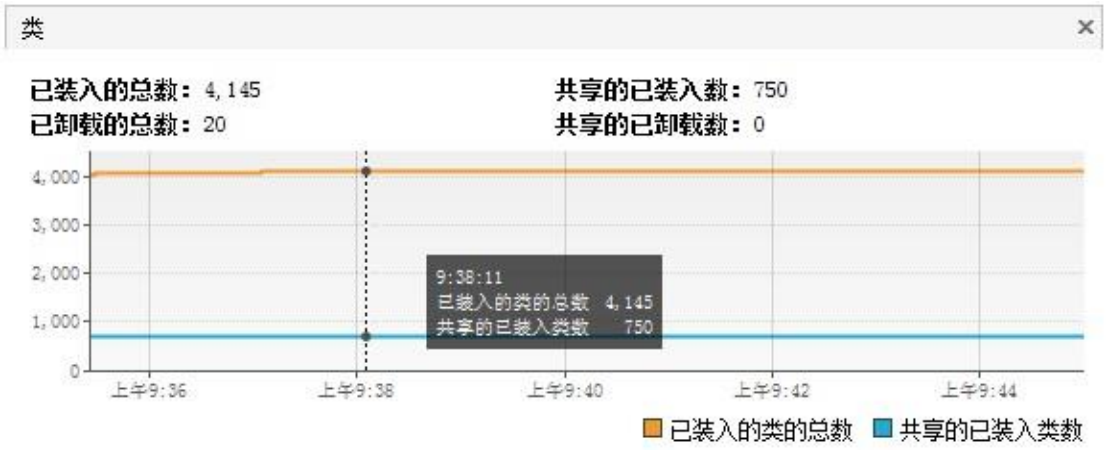

图 6-8 HPwsn 系统运行类情况图

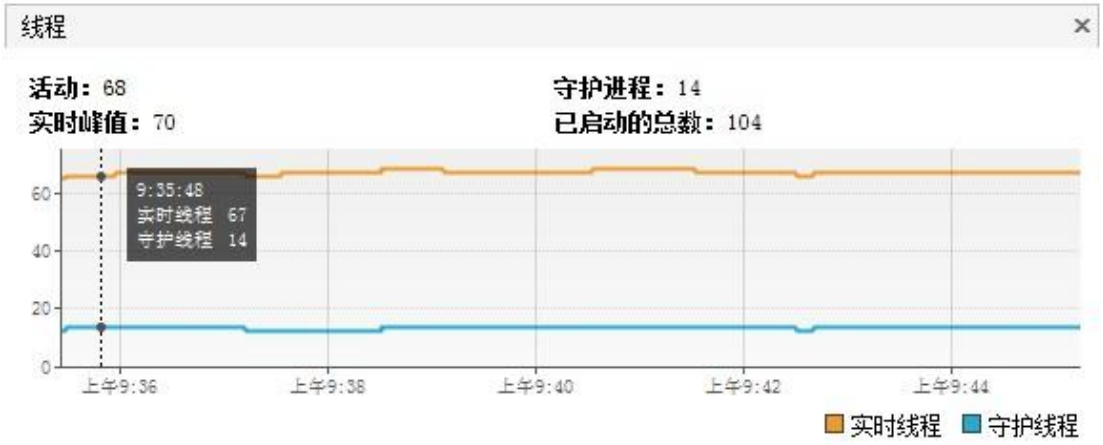

图 6-9 HPwsn 系统运行线程情况图

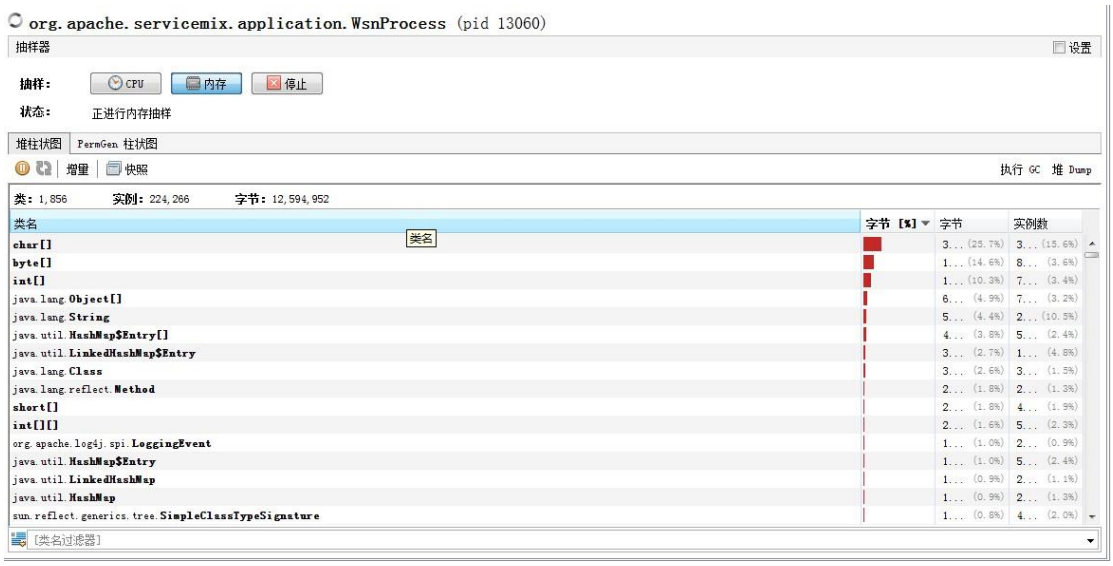

图 6-10 HPwsn 系统运行时内存抽样图

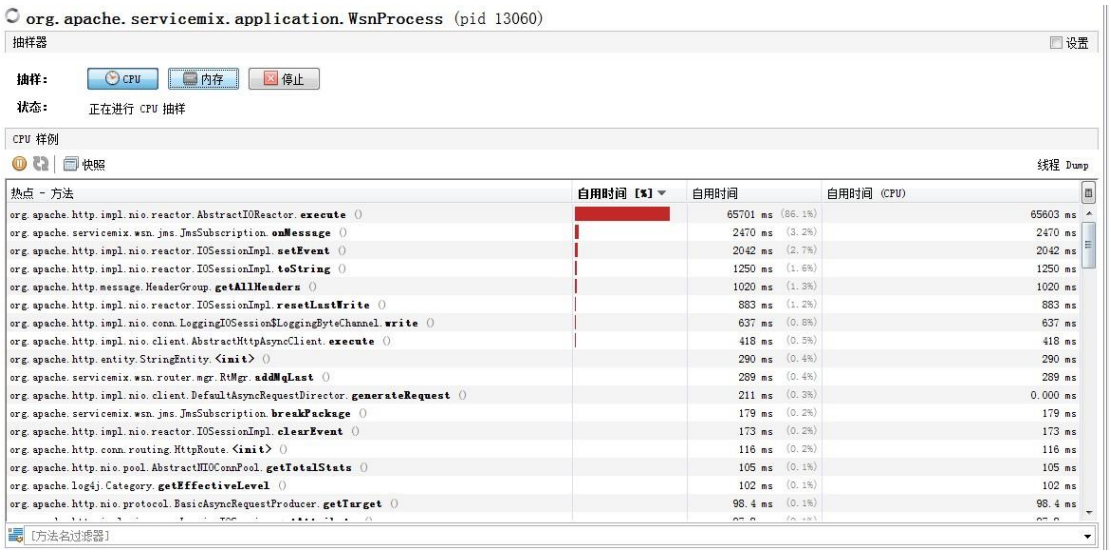

图 6-11 HPwsn 系统运行时 CPU 抽样图

从图 6-5 至 6-11 所示的图中可以看出，系统的运行在趋势上是稳定的。包括 CPU、内存、线程等各方面的资源占用都没有明显的增长。

6.6 本章总结

本章就系统的功能和性能进行测试。介绍了系统如何安装和部署，之后为测试系统的功能和性能分别编制了测试用例。经过实际的测试后，在系统的功能上，完成了本文第三章概要设计中所涉及的功能，性能上也达到了一般基于 Web Service 的 SOA 系统的服务效率。但是也可看到，在传统发布订阅模式上加入主动推送和语义保持功能后，系统的运行效率并不是很有优势，需要进一步优化。

第七章　总结与展望

7.1 工作总结

HPwsn 发布订阅系统的设计与实现工作开始于 2012 年 4 月，伴随着北京金房暖通有限责任公司的信息平台搭建项目，我逐渐从师兄手中接过了发布订阅系统的工作。开始接手的时候，由于全部代码数量比较庞大，花了两个月的时间去熟悉系统的设计和实现，调研相关的技术，如 Web Service 技术、JMS 技术、发布订阅模型、常用 ESB 的设计理念与优缺点等，积累了一定程度的技术资源和对发布订阅系统的认识。同年 6 月份，针对金房公司的信息平台对发布订阅的特定需求，开始改进现有的发布订阅系统。

在改进发布订阅系统的过程中，我们的目标是非常明确的，就是在使得系统能够长时间稳定运行的前提下，尽一切努力提高系统性能，以将该系统真正应用在生产环境中，将设计落地。围绕这个目标，我们设计实现了树状名称表达、元数据管理与持久化、分包发送以及异步消息递交等功能模块，并且给予 JDK 自带的工具实现了一套评价体系，在开发测试的过程中不断完善系统。经过开发和测试的一个又一个迭代，在 10 月份，HPwsn 系统最终成型。我们将之运用在金房公司的信息平台中并针对其应用环境做了优化，取得了巨大的成功，发挥了举足轻重的作用。

但是由于能力和时间的限制，HPwsn 系统在设计和实现上还存在这样那样的不足之处，例如，用户接口不够丰富，系统可操作性不太理想等，有待进一步完善。

7.2 工作展望

本文论述的 HPwsn 系统是一个高性能、高可靠性的发布订阅消息中间件系统，在一个信息系统中，它扮演的是消息分发者的角色，基于该系统，用户可以开发出松耦合、高时效、异步的可靠应用。针对本系统，未来还有几个方面可以展开工作，例如：

1、在本系统中，订阅者、发布者是采用 SOAP 消息与 HPwsn 系统进行数据交互，在当前流行的数据串行化技术中，Json 和 Google ProtoBuf 都可以替代 XML 完成类似的功能，且具备更高的效率。这也是业界目前普遍采用的技术，可以从这个角度考虑进一步提高系统性能。

2、截止本文完稿，经测试稳定版的 HPwsn 系统都是运行在 windows 平台上的，而作为一种服务系统，想要更稳定地提供服务，一个稳定的 linux 平台版本是不可或缺的。

3、HPwsn 的测试都是我们自己编写的发布者、订阅者小程序实现的，有不太规范的地方，目前缺少一个完善的性能测试系统，在这个方面也是可以做一些工作的。

参考文献

[1] TIBCO Corp. TIB/Rendezvous White Paper, 2000. URL. http://www.tibco.com/software/enterprise_backbone/rendezvous.jsp

[2] Gough K. J, Smith G. Efficient recognition of events in a distributed systems. In: Proc. of the 18th Australasian Computer Science Conf. Adelaide: IEEE Computer Society, 1995.

[3] Carzaniga A, Rosenblum D. S, Wolf A.L. Design and evaluation of a wide-area event notification service. ACM Trans. on Computer Systems, 2001,19(3):332-383.

[4] Cugola G, Nitto E. D, Fuggetta A. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Trans. on Software Engineering, 2001,27(9):827-850.

[5] Rowstron A, Kermarrec A. M, Castro M, Druschel P. SCRIBE: The design of a large-scale event notification infrastructure. In: Proc. of the 3rd Int'l Workshop on Networked Group Communication. London: Springer-Verlag, 2001:30-43.

[6] Ruisheng SHI, Yang ZHANG, Bo CHENG, et al. Publish/Subscribe network infrastructure based on web service notification. Journal of Theoretical and Applied Information Technology, 2012, Vol.45 No.1:99-108.

[7] Guowei PAN, Wei SONG, xiangnan WANG, et al. Research on Application of Message Oriented Middleware Based on Publish/Subscribe Model in SCADA System. Power System Technology, 2008, 32(18):77-81.

[8] Carzaniga A, Rosenblum D. S, Wolf A. L. Design and evaluation of a wide-area event notification service. TOCS, 2001.

[9] Cugola G, Nitto E. D, Fuggetta A. The JEDI event based infrastructure and its application to the development of the OPSS WFMS. TSE, 2001.

[10] Rose I, Murty R, Pietzuch P, Ledlie J, Roussopoulos M, Welsh M. COBRA: Content-based filtering and aggregation of blogs and RSS feeds. in NSDI, 2007.

[11] 刘家红, 吴泉源, 甘亮. 面向服务环境下的事件通知服务：模型与实现面. 计算机工程与科学, 2008(5):98-101.

致谢

自 2011 年 9 月进入北京邮电大学网络技术研究院网络服务基础研究中心攻读硕士学位以来，不知不觉已经过去了快两年半的时间，这两年半是我人生中在学术能力方面成长最快的一段时间，在此期间，我不仅编写代码、解决问题的能力得到了很大的提升，研究能力也有了一定程度的提高，而这一切成果的取得都离不开实验室浓厚的学术氛围，离不开实验室为我们创造的良好科研环境，其中应该特别感谢的是我的老师们，尤其是我的指导老师章洋。在我努力学习的道路上，他扮演了一个谆谆引导的前辈和不厌其烦的温厚长者，正是他督促我一次次突破自己的局限。

另外还应该特别感谢的是我的父母，他们在我迷茫、失意的时候给我最坚定的支持、关心和问候，没有他们的鼓励和无私奉献，我肯定没办法取得今天的成绩。

最后，向对本项目提供资助的基金会提出特别感谢。他们的资助是本项目得以完成的基础。他们是：973 国家重点基础研究发展计划(Grant No. 2011CB302704, 2012CB315802)， 国家自然科学基金委员会(Grant No. 61001118, 61171102, 61003067, 61132001)， 新世纪大学优秀人才支持计划(Grant No. NECT-11-0592)以及新一代无线网络项目(Grant No. 2012ZX03005008-001)。

再次对向这个项目和向我提供帮助的老师、朋友、亲人、组织表示最诚挚的感谢！

攻读学位期间发表的学术论文目录

温鹏，章洋 高性能发布/订阅系统接口服务的设计与应用 软件 34(11) 2013.11:31-35