1. **绪 论**
   1. **研究背景与意义**

随着信息化的高速发展，出现了很多互联网公司。互联网公司的主要用户群体是终端客户，因此“以客户为中心”的理念也随之得到的越来越广泛的认同。很多互联网公司都关注客户在企业经营活动中的体验和感受。因此，面向客户的系统或者说C端（Customer）系统（如，网上商城系统，支付系统，订单系统，物理系统等）也越来越受到互联网公司的关注和重视。

然而，在这些面向客户的系统建设和运营过程中，有大量信息（如，购物信息，订单信息，支付信息，物流信息，评价信息）产生，这些信息一般被作为消息在各个系统和客户中进行交互和流转。如果各个系统单独开发消息通道（如，短信通道，推动通道）进行对接，需要花费很大的开发成本，并且系统间在推送消息的功能和性能方便都需要经过严格测试，使得系统的建设周期，研发成本和效率不能满足既定的要求。并且，系统直接直接对接，造成了系统间的紧耦合，造成系统的可扩展性，可用性和容错性降低。所以，如果建设一个统一的，通用的，安全且扩展性高的消息推送系统是值得研究的，这也对企业信息化建设，特别是互联网公司的面向客户的业务系统建设具有重要的意义。

* 1. **国内外研究现状**

目前，国内外消息系统的研究和实践，可以从理论和实践两个方面来进行深入分析。

**1.2.1国内外理论研究现状**

从理论上还说，消息系统属于MOM（Message-oriented middleware）的范畴。目前，主要的业界标准有：

1. JMS（Java Message Service）

JMS用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。JMS务是一个与具体平台无关的API，绝大多数MOM提供商都对JMS提供支持。JMS是一个在 Java标准化组织（JCP）内开发的标准（代号JSR 914）。2001年6月25日，Java消息服务发布JMS 1.0.2b，2002年3月18日Java消息服务发布 1.1，统一了消息域。目前JMS最新规范为2.0版本。

1. AMQP（Advanced Message Queueing Protocol）

AMQP是一个提供统一消息服务的应用层标准高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计。基于此协议的客户端与消息中间件可传递消息，并不受客户端/中间件不同产品，不同的开发语言等条件的限制。

1. DDS（Data Distribution Service）

DDS是对象管理组织(OMG)在HLA及CORBA等标准的基础上制定的新一代分布式实时通信中间件技术规范，DDS采用发布/订阅体系架构，强调以数据为中心，提供丰富的QoS服务质量策略，能保障数据进行实时、高效、灵活地分发，可满足各种分布式实时通信应用需求。DDS信息分发中间件是一种轻便的、能够提供实时信息传送的中间件技术。

1. MQTT（Message Queuing Telemetry Transport）

MQTT是一种基于发布/订阅（publish/subscribe）模式的“轻量级”通讯协议，该协议构建于TCP/IP协议上，由IBM在1999年发布。MQTT最大优点在于，可以以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。作为一种低开销、低带宽占用的即时通讯协议，使其在物联网、小型设备、移动应用等方面有较广泛的应用。

**1.2.2国内外应用研究现状**

从应用方面来说，消息系统相关的产品主要有以下几种类型：

1. 消息中间件厂商

目前大部分消息中间件都是基于JMS或者AMQP规范而实现的。比较著名的产品有Apache ActiveMQ，Apache Qpid，Apache RocketMQ，RabbitMQ等等。

1. 消息发送通道提供商

基于不同的消息类型，大致分为：电话语音通道，短信通道，彩信通道，邮件通道，微信通道，APP推送通道等等。

综上所述，消息系统的实际上并不是一套单一的统一的系统。不同类型的企业对其消息系统的定义也各不相同。因此，对以上各种平台和技术进行深入的研究和学习，设计出对企业业务发展有利用价值的消息系统，是具有非常重要的意义的。

* 1. **论文研究目标及主要内容**

结合上述理论与应用的现状，本课题主要基于JMS规范及其实现，研究并设计实现一个统一的，标准的，可靠的，易扩展且易维护的消息系统，为企业系统之间，企业与用户之间的消息传递提供安全的服务。

主要研究内容分为以下几个方面：

1. 消息系统的核心机制及关键组件

消息系统主要基于生产者-消费者设计模式，将企业系统与消息通道进行解耦，使得消息传递模型从原本通过直连方式进行消息传递变成统一经过消息系统进行消息传递，从而降低各个业务系统的复杂性和耦合性，提高系统对接效率，增强系统的可靠性和可维护性。

核心组件主要包括：消息任务管理模块，消息监控模块，消息存储模块，消息日志管理模块，配置管理模块，消息队列（Queue）和主题（Topic）。

1. 统一的对外服务接口API

对外服务接口位于消息系统的接口层，以一套标准的API接口提供给服务调用方，主要包括：创建消息，查询消息，更新消息，存储消息，消费消息，失败重试，批量消息，日志查询，状态监控等接口。

通过使用标准的对外服务接口，服务调用方只需要关注自己的业务逻辑，而不需要关注消息系统底层实现细节，进一步降低了业务开发的难度，更好的实现了系统间的解耦合。

1. 可维护的异常消息

通过标准的异常处理模块，将因为网络分区或者机器故障导致的传输异常消息保存到指定区域，通过可配置的重试策略将异常消息重新处理，最大限度保证消息的可靠传输。

* 1. **论文组织结构及章节安排**

本文以当前企业各种子系统中消息传递所遇到的实际问题和面临的挑战为出发点，通过对国内外相关理论和时间的研究和对比分析，提出了一个以消息系统作为中心的，业务系统围绕消息系统进行通信的架构。并且基于JMS规范，对其进行需求梳理，需求分析，总体架构，功能性设计和非功能性设计，关键技术实现，功能测试，性能测试等一系列工作。主要章节安排如下：

//TODO(未完待定~~~~~~先写下面的具体实现部分，在补充文章章节安排)

1. **消息系统的相关理论和技术研究**

随着企业信息化的不断发展，为了满足企业系统之间消息的可靠传输，简单的业务子系统之间进行消息交互已经不能满足企业系统建设过程中对成本控制，质量控制，可靠性保障和快速迭代的需求。因此，建设统一的，可靠的消息系统对于企业信息化建设来说迫在眉睫。通过统一的消息系统，将企业的业务系统进行解耦，以增强业务系统的通用性，降低其开发成本。本章节对消息系统所涉及的相关理论和技术进行综述，为了接下来基于JMS的消息系统的架构设计和实现奠定了理论基础和技术支持。

**2.1中间件相关简介**

中间件是一种独立于操作系统的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。中间件位于客户机/服务器的操作系统之上，管理计算机资源和网络通讯。相连接的系统，即使它们具有不同的接口，但通过中间件相互之间仍能交换信息。执行中间件的一个关键途径是信息传递。通过中间件，应用程序可以工作于多平台或OS环境。

**2.1.1中间件的分类**

中间件大约分为以下几类：

1. 远程过程调用(Remote Procedure Call)中间件

远程过程调用是一种广泛使用于分布式系统中的处理方法。一个应用程序使用RPC来“远程”执行一个位于不同地址空间里的函数/过程，并且这种调用方式从效果上看和执行本地调用没有区别。事实上，任何RPC应用都至少可以分为两个部分：Server和Client。Server提供一个或多个远程计算函数/过程；Client向Server发出远程调用。Server和Client可以位于同一台计算机，也可以位于不同的计算机，甚至运行在不同的操作系统之上。Server和Client通过网络进行通讯。相应的stub和运行支持提供数据转换和通讯服务，从而屏蔽不同的操作系统和网络协议。

1. 对象请求代理(Object Request Broker)中间件

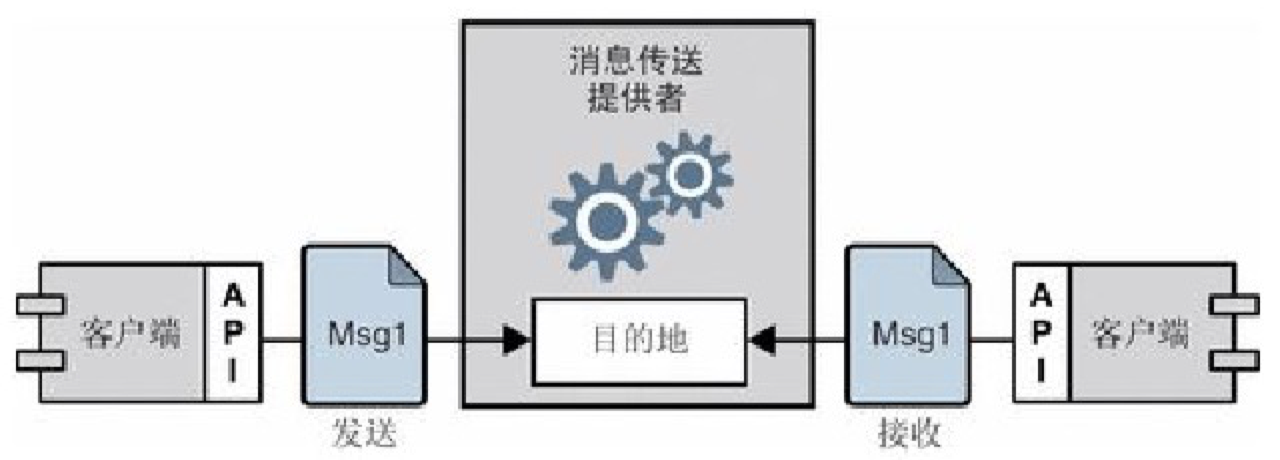
ORB使得对象可以透明地向其他对象发出请求或接受其他对象的响应，这些对象可以位于本地也可以位于远程机器。ORB与RPC的最主要区别在于，RPC是面向过程的远程调用，而ORB是面向对象的远程调用。ORB拦截请求调用，并负责找到可以实现请求的对象、传送参数、调用相应的方法、返回结果等。常见的ORB中间件有：CORBA，RMI/IIOP等等。

1. 事务处理监控（Transaction processing monitors）中间件

事务处理监控（Transaction processing monitors）最早出现在大型机上，为大型机提供支持大规模事务处理的可靠运行环境。随着分布计算技术的发展，分布式系统对大规模的事务处理提出了需求，比如商业活动中大量的关键事务处理。事务处理监控界于Client和Server之间，进行事务管理与协调、负载平衡、失败恢复等，以提高系统的整体性能。它可以被看作是事务处理应用程序的“操作系统”。

1. 面向消息(Message-Oriented Middleware)中间件

面向消息的中间件使得分布式系统可以不必直接互连，而是通过发送和接收消息来进行通信和数据交换。面向消息的中间件与RPC和ORB的主要区别在于，通信程序可以在不同的时间运行，对于通信程序没有特定的结构约束。典型的MOM系统架构模型如下图所示：



**2.1.2主流的面向消息中间件的标准和规范**

消息系统是基于MOM（Message-oriented middleware）构建的基础服务系统，目前，主要的MOM工业标准和规范如下表2-1所示。

表2-1面向消息的中间件的标准和规范对比

Table 2-1 The Difference Between MOM Standards

|  |  |  |
| --- | --- | --- |
|  | **JMS** | **AMQP** |
| 定义 | JMS（Java Message Service）即Java消息服务应用程序接口，是一个Java平台中关于面向消息中间件（MOM）的API，用于在多个应用程序之间，或分布式系统中发送消息，进行异步通信。从而达到系统接口的目的。Java消息服务是一个与具体平台无关的API，很多MOM提供商都对JMS规范提供支持。 | AMQP（Advanced Message Queuing Protocol）是一个提供统一消息服务的应用层标准高级消息队列协议。基于此协议的客户端与消息中间件可传递消息，并不受客户端/中间件不同产品，不同的开发语言等条件的限制。 |
| 开放性 | JavaEE平台的一部分 | 应用层开放协议 |
| 厂商支持 | 大部分国际厂商支持，并开发出工业界为人熟知的产品，如：Apache ActiveMQ，JBoss MQ等等 | 逐渐被国际厂商支持，并开发出一系列的产品，如：Apache Qpid，RabbitMQ |

**2.1.3ActiveMQ架构模型**

Java消息服务（Java Message Service，JMS）是一套可以跨平台的API，其是JavaEE规范的一部分。JMS与我们熟知的JDBC类似，是一套独立于厂商标准规范。ActiveMQ就是实现JMS规范的典型代表。下面介绍ActiveMQ的架构模型。

（1）消息传递模型

JMS提供2种消息传递模型，分别是：

**点对点（Point-to-Point，P2P）模型：**

在点对点模型下，ActiveMQ提供一个消息队列（Queue）的虚拟消息通道，让客户端以同步或者异步的方式来收发消息。每条消息只能被一个消费者接收。ActiveMQ点对点模型如下图2-1所示：

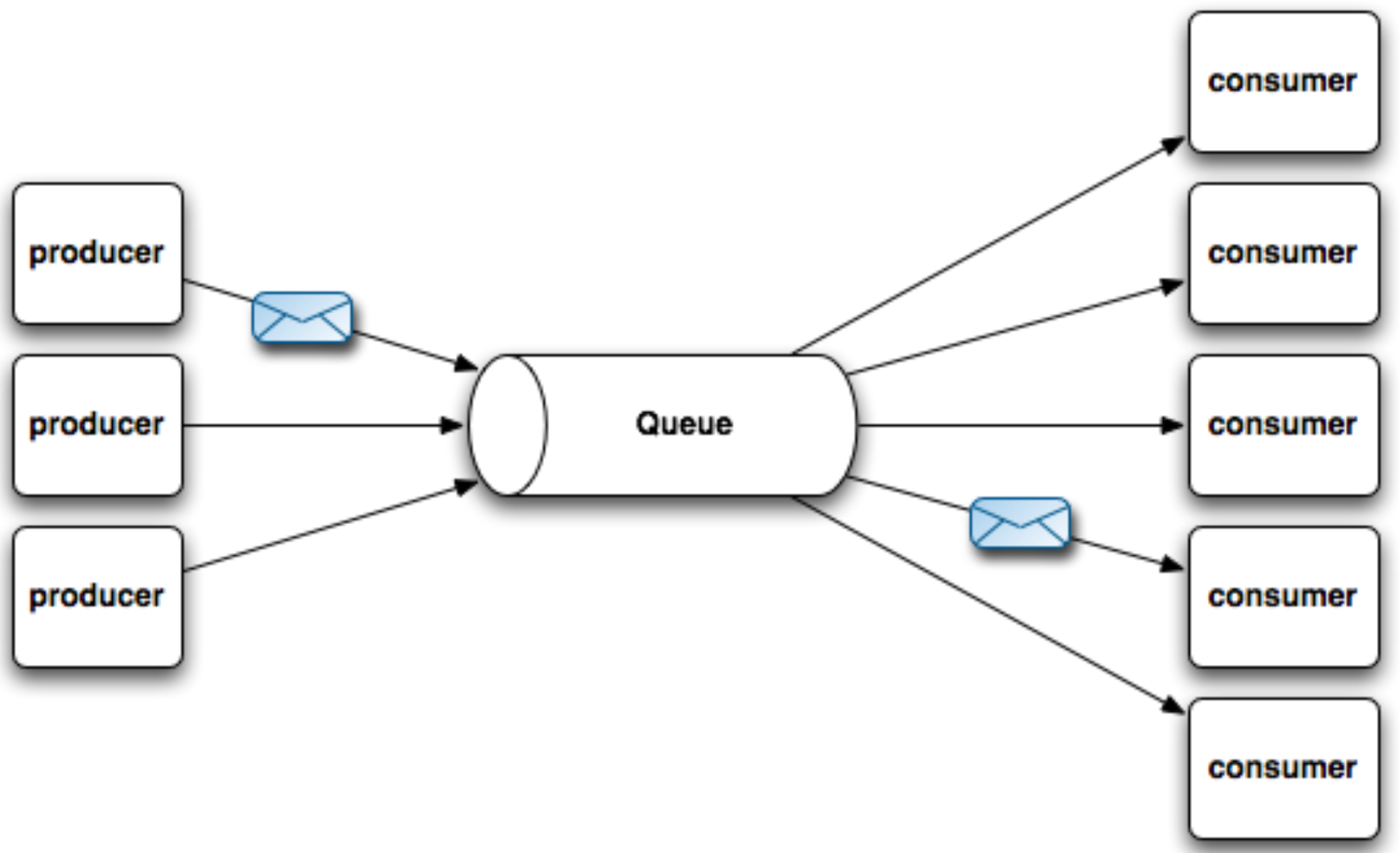


图2-1 ActiveMQ点对点模型

Figure 2-1 ActiveMQ Point-to-Point Model

**发布-订阅（Publish-Subscribe，Pub/Sub）模型：**

在发布-订阅模型下，ActiveMQ提供主题（Topic）的虚拟通道，讲消息发送给多个订阅该主题的消息接受者。一个消息可以被多个消费者接收。ActiveMQ发布-订阅模型如下图2-2所示：

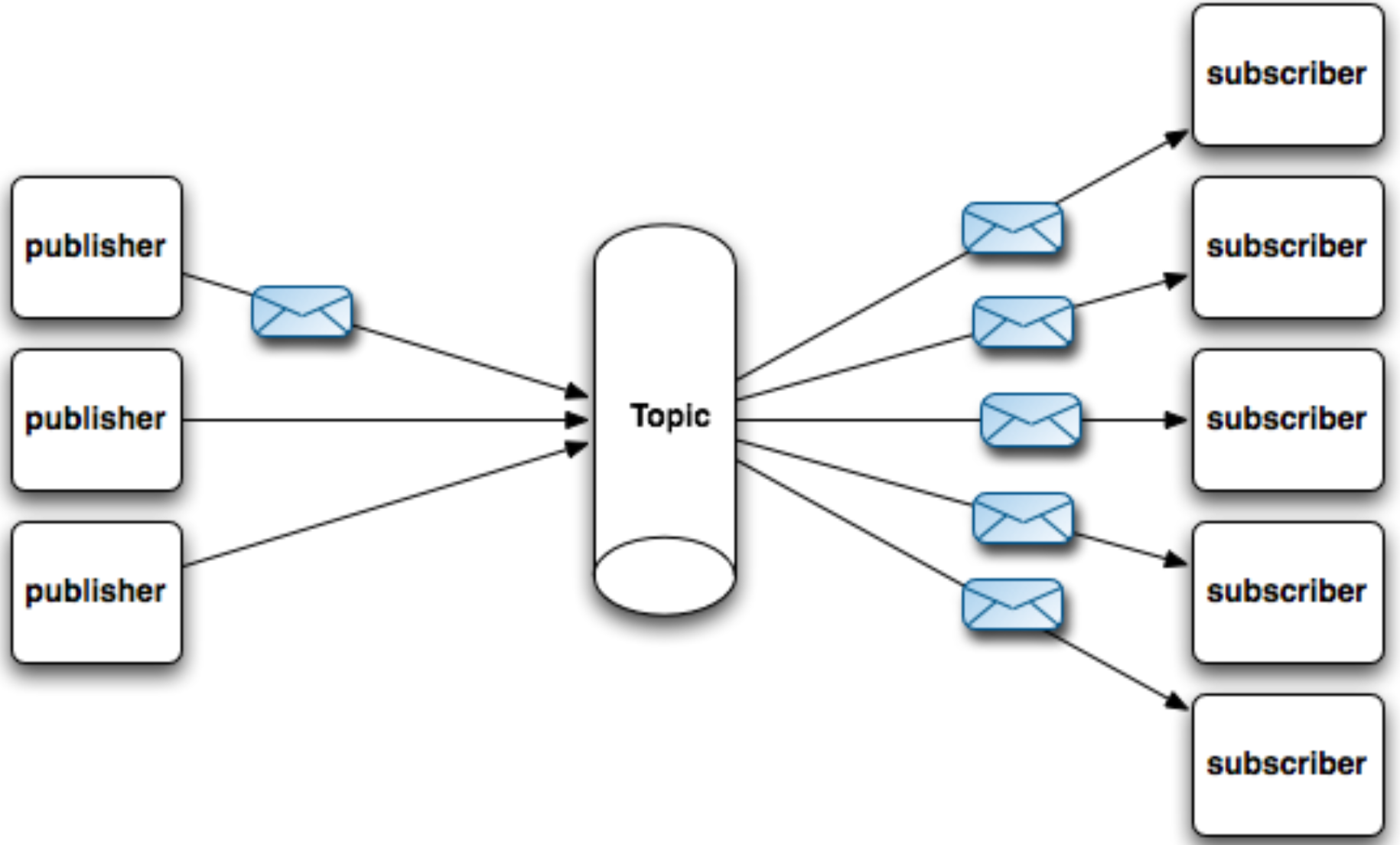


图2-2 ActiveMQ发布-订阅模型

Figure 2-2 ActiveMQ Pub/Sub Model

（2）Point-to-Point模型与Pub/Sub模型的区别

Point-to-Point模型与Pub/Sub模型虽然都是ActiveMQ支持的消息传递模型，但是这两者在许多方面存在不同，理解这两种模型的区别对于我们在实际应用中选择合适的模型解决实际问题有指导意义。两者主要区别归纳如下表2-2：

表2-2Point-to-Point模型与Pub/Sub区别

Tabel 2-2 The Difference Between Point-to-Point And Pub/Sub

|  |  |  |
| --- | --- | --- |
|  | Point-to-Point | Pub/Sub |
| 消息通道 | Queue | Topic |
| 消息发送模式 | 一条消息只能被一个消费者接收到 | 一条消息可以被多个消费者接收到 |
| 持久化 | Queue中的消息消费者不在线，消息会被持久化在Queue中，不会丢失。 | 分为持久订阅和非持久订阅  持久订阅是即使订阅此主题的消费者不在线，也会保存消息直到消费者上线。  非持久订阅指消费者不在线时，不会为其保存消息。 |
| 消息完整性 | 保证消息会收到 | 不保证消息一定被接收 |

**2.2分布式协调服务简介**

分布式协调技术主要用来解决分布式环境当中多个进程之间的同步控制，让进程间有序的访问某种临界资源，防止造成"脏数据"的后果。作为一个分布式系统，分区容错性是一个必须要考虑的关键点。一个分布式系统一旦丧失了分区容错性，也就表示放弃了扩展性。因为在分布式系统中，网络故障是经常出现的，一旦出现在这种问题就会导致整个系统不可用是绝对不能容忍的。所以，大部分分布式系统都会在保证分区容错性的前提下在一致性和可用性之间做权衡。

分布式协调服务是一个为分布式应用提供一致性协调服务的服务型软件。提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

**2.2.1分布式协调服务的分类**

目前工业界主要的分布式协调服务可以分为以下两种：

1. Zookeeper

ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。ZooKeeper的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

1. etcd

etcd是一个高可用的键值存储系统，主要用于共享配置和服务发现。etcd是由CoreOS开发并维护的，灵感来自于 ZooKeeper 和 Doozer，它使用Go语言编写，并通过Raft一致性算法处理日志复制以保证强一致性。Raft是一个来自Stanford的新的一致性算法，适用于分布式系统的日志复制，Raft通过选举的方式来实现一致性，在Raft中，任何一个节点都可能成为Leader。Google的容器集群管理系统Kubernetes、开源PaaS平台Cloud Foundry和CoreOS的Fleet都广泛使用了etcd。

1. Zookeeper与etcd的区别

虽然目前工业界主要的协调服务有Zookeeper和etcd，但是这2种服务有着本质上的区别，二者具体区别如下表2-3所示：

表2-3 Zookeeper与etcd区别

Tabel 2-2 The Difference Between Zookeeper And etcd

|  |  |  |
| --- | --- | --- |
|  | Zookeeper | etcd |
| 服务的目的 | Zookeeper是一个具有高效率和高可靠性的分布式协同工作系统。 | etcd是一个具有高可用性的 Key/Value 存储系统，主要用于分享配置和服务发现。 |
| 接口 | Zookeeper基于TCP的自协议，需要安装相应客户端程序。 | etcd提供了REST接口（HTTP+JSON）方便集群中每一个主机访问 |
| 功能 | 提供了key，value存储服务，集群中建立临时节点，观察key值变化等。 | key，value存储服务，集群队列同步服务，观察一个key的数值变化，以及查询历史key值信息等。 |

续表2-3

|  |  |  |
| --- | --- | --- |
|  | Zookeeper | etcd |
| 分布式协议 | 基于Paxos一致性协议，改进的Zab协议。提供强一致性保证。 | 基于Raft一致性协议。提供强一致性保证。 |
| 部署形态 | 采用小集群（zookeeper server节点组成一个集群）+大集群（其它节点来直接使用服务）的形式，集群可以达到上千节点。 | 采用小集群（etcd server节点组成一个集群）+大集群（其它节点来直接使用服务）的形式，集群可以达到上千节点。 |
| 实现语言 | java，实现代码量要多于go，在小规模集群中性能一般，但是在大规模情况下，使用对多线程的优化后，也和go相差不大。 | go 拥有几乎不输于C的效率，特别是go语言本身就是面向多线程，进程通信的语言。在小规模集群中性能非常突出。 |

**2.2.2 Zookeeper的功能和特性分析**

ZooKeeper是一种为分布式应用所设计的高可用、高性能且一致的开源协调服务，它提供了一项基本服务：分布式锁服务。由于ZooKeeper的开源特性，后来我们的开发者在分布式锁的基础上，摸索了出了其他的使用方法：配置维护、组服务、分布式消息队列、分布式通知/协调等。

ZooKeeper性能上的特点决定了它能够用在大型的、分布式的系统当中。从可靠性方面来说，它并不会因为一个节点的错误而崩溃。除此之外，它严格的序列访问控制意味着复杂的控制原语可以应用在客户端上。ZooKeeper在一致性、可用性、容错性的保证，也是ZooKeeper的成功之处。

1. 命名服务

在Zookeeper的文件系统里创建一个目录，即有唯一的path。在我们无法确定上游程序的部署机器时即可与下游程序约定好path，通过path即能互相探索发现，不见不散了。典型的RPC框架dubbo就是通过将注册的服务通过path暴露给调用方的。

1. 配置管理

分布式系统中总是需要许多配置的，如果程序分散部署在多台机器上，要逐个改变配置就变得困难。如果把这些配置全部放到Zookeeper上去，保存在 Zookeeper 的某个目录节点中，然后所有相关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 Zookeeper 的通知，然后从 Zookeeper 获取最新的配置信息应用到系统中就可以了。Zookeeper配置管理如下图2-3所示：

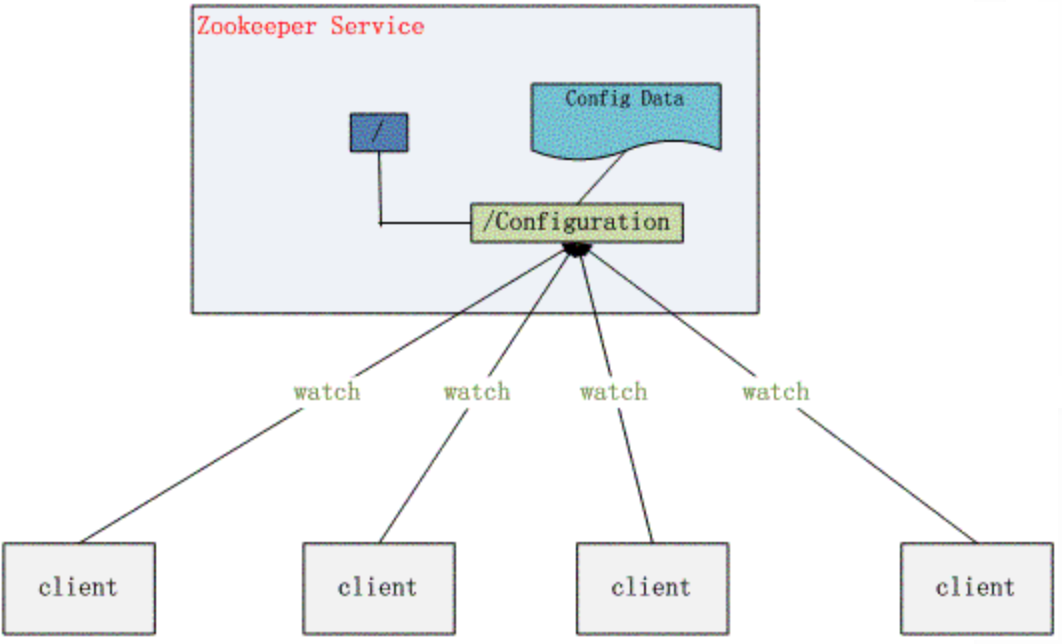


图2-3 Zookeeper配置管理

Figure 2-3 Zookeeper Configuration Management

1. 集群管理

集群管理主要在于两点：是否有机器退出和加入、选举master。

对于第一点，所有机器约定在父目录GroupMembers下创建临时目录节点，然后监听父目录节点的子节点变化消息。一旦有机器宕机，该机器与 zookeeper的连接断开，其所创建的临时目录节点被删除，所有其他机器都收到通知：某个兄弟目录被删除，于是所有子节点都会知道。当有新机器加入的场景也是类似，所有机器收到通知：有新兄弟目录加入了。

对于第二点，我们稍微改变一下，所有机器创建临时顺序编号目录节点，每次选取编号最小的机器作为master就好。

Zookeeper分布式集群管理如下图2-4所示：

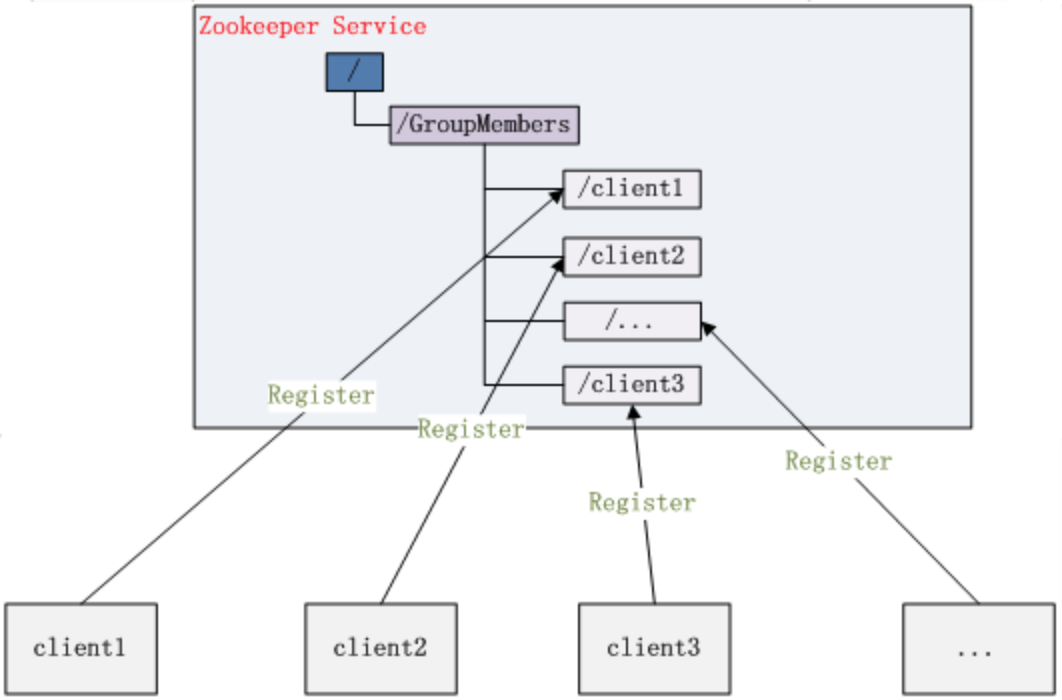


图2-4 Zookeeper集群管理

Figure 2-3 Zookeeper Cluster Management

1. 分布式锁

Zookeeper锁服务可以分为两类，一类是保持独占的独占锁，另一个是控制时序时序锁。

对于第一类，我们将Zookeeper上的一个znode看作是一把锁，通过createznode的方式来实现。所有客户端都去创建 /distribute\_lock 节点，最终成功创建的那个客户端也即拥有了这把锁，用完删除掉自己创建的distribute\_lock 节点就释放出锁。所有客户端都依照这个规则依次进行，从而就达到了独占的目的。

对于第二类， /distribute\_lock 已经预先存在，所有客户端在它下面创建临时顺序编号目录节点，和选master一样，每次只有编号最小的才能获得锁，用完之后将改临时编号目录节点删除，让出控制权给改节点序号之后的下一个节点获得控制权。

Zookeeper分布式锁如下图2-5所示：

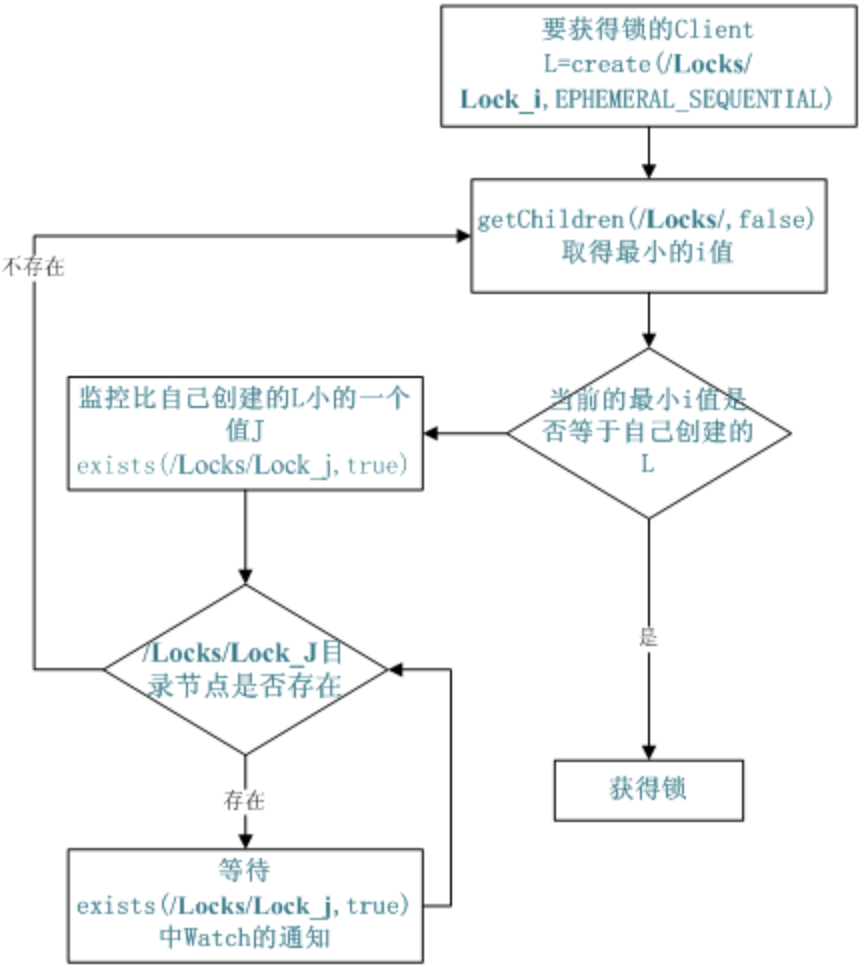


图2-5 Zookeeper分布式锁管理

Figure 2-3 Zookeeper Distributed Lock Management

1. 队列管理

Zookeeper队列管理可以维护两种类型的队列：

第一类，同步队列，当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达。在约定目录下创建临时目录节点，监听节点数目是否是我们要求的数目。

第二类，按照 FIFO 方式进行入队和出队操作的队列。和分布式锁服务中的控制时序场景基本原理一致，入列有编号，出列按编号。

了解了Zookeeper的功能和特性后，对于我们选择合适的分布式协调服务具有很好的指导意义。并且对消息系统的构建和技术选型提供了一定的理论基础和技术支持。

**2.3缓存服务简介**