南京大学本科生毕业论文（设计）中文摘要

随着互联网技术的飞速发展，网络信息以指数型趋势高速增长。对于一个要对数据进行统计分析的系统而言，搜集数据的过程是冗长枯燥的。基于这一现实，分布式爬虫系统获得了发展的契机。系统通过多台服务器的协调运行，成倍地提高了爬虫的效率。当然，分布式系统在获得效率提升的同时也大大增加了系统的复杂程度，开发人员需要考虑多方面因素以确保系统的正常运转。

本文对分布式爬虫系统的架构做了深入的讨论，给出了选择该架构的原因。项目使用了JMS消息队列技术在节点之间异步传递消息，起到应用解耦的效果。使用apache出品的activeMQ作为JMS服务器，能够跨语言跨平台传递消息，而且具有延迟接受的特性，一定程度上降低了网络延迟对系统性能的阻碍。中心节点的任务分配采用线程池技术，缓解了由于任务积累产生的效率瓶颈。

本文先介绍了项目研究背景，然后介绍了分布式爬虫相关知识和本项目所用技术，最后展示了项目的需求分析和体系结构。最终实现的分布式爬虫系统能够做到中心节点分配爬虫任务，任务节点执行爬虫程序并存储数据。同时系统针对一些分布式系统常用的性能需求做了设计，提高了系统的可扩展性和可靠性。

关键词：分布式系统，爬虫，JMS

南京大学本科生毕业论文（设计）英文摘要

With the rapidly development of Internet technology, network information has grown at an exponential rate. For a system that requires statistical analysis of data, the process of collecting data is tedious. Based on this reality, the distributed crawler system has gained an opportunity for development. The system multiplies the efficiency of crawler by coordinating within several servers. However, the distributed system increases the complexity of the system while gaining efficiency. Developers need to consider various factors to ensure the normal operation of the system.

This article makes an in-depth discussion of the architecture of the distributed crawler system and gives the reasons for choosing certain architecture. The project uses JMS technology to deliver messages asynchronously between nodes, which has the effect of applying decoupling. The project also uses activeMQ from apache as the JMS server, which has the feature of cross-language and cross-platform messages delivery. In addition, activeMQ has the characteristics of delay acceptance. To some extent, it reduces the network delay on the performance of the system. The task allocation of the central node adopts the thread pool technology, which relieves the efficiency bottleneck caused by task accumulation.

In this article, first I introduced the project research background. Then I introduced the distributed crawler knowledge and technology used in this project. Finally I demonstrated the project's requirement analysis and architecture. The distributed crawler system this article implemented can distribute tasks by central node, and execute crawler programs and store data by task nodes. At the same time, the system is designed for the common performance requirements of distributed systems, which improves the system's scalability and reliability.

KEY WORDS: distributed system, crawler, JMS

目录

南京大学本科生毕业论文（设计）中文摘要 I

南京大学本科生毕业论文（设计）英文摘要 II

目录 I

图目录 III

表目录 IV

第一章 引言 1

1.1 研究背景 1

1.1.1 SourceForge.net 1

1.1.2 需求复用 1

1.1.3 实验项目 1

1.1.4 爬虫简介 2

1.2 国内外研究现状 2

1.2.1 java爬虫框架 2

1.2.2 python爬虫框架 2

1.3 本文的主要研究内容 3

1.4 论文的主要工作和组织结构 3

第二章 分布式爬虫综述 4

2.1 分布式爬虫类型 4

2.1.1 应用于搜索引擎 4

2.2.2 基于主题的爬虫 4

2.2.3 用户个性化爬虫 4

2.2 MapReduce 5

2.3 分布式爬虫系统的架构 6

2.3.1 主从式 6

2.3.2 自治式 7

2.3.3 混合式 7

2.4 爬取策略 8

2.5 部署网络 8

第三章 技术基础 10

3.1 网络通信技术 10

3.1.1 RPC 10

3.1.2 WebService 10

3.1.3 RMI 11

3.1.4 JMS 12

3.1.5 activeMQ 14

3.2 队列 15

3.3 线程池 15

3.4 jsoup 17

第四章 需求分析和概要设计 18

4.1 功能需求分析 18

4.1.1 需求概述 18

4.1.2 用例图 19

4.1.3 用例描述 19

4.2 性能需求分析 21

4.2.1 可扩展性 21

4.2.2 可靠性 22

4.2.3 负载均衡 22

4.2.4 降低通信开销 22

4.3 概要设计 22

4.3.1 系统结构图 22

4.3.2 流程图 23

4.3.3 性能需求的设计 24

第五章 模块详细设计与实现 26

5.1 模块详细设计 26

5.2.1 CenterNode模块 26

5.2.2 TaskNode模块 28

5.2 模块实现 31

5.2.1 关键部分代码 31

5.2.2 运行截图 34

5.3 数据库设计 37

第六章 总结与展望 38

6.1 总结 38

6.2 展望 38

参考文献 39

致谢 40

图目录

图2.1 MapReduce的节点监控 6

图2.2 主从式架构 7

图2.3 自治式架构 7

图3.1 WebService模型 11

图3.2 RMI模型 11

图3.3 点对点消息模型 12

图3.4 发布/订阅消息模型 13

图3.5 JMS服务器的连接过程 14

图3.6 线程池的工作原理 16

图4.1 执行过程 18

图4.2 用例图 19

图4.3 系统结构图 23

图4.4 系统流程图 24

图5.1 中心节点类图 26

图5.2 任务节点类图 28

图5.3 连接JMS服务器代码 32

图5.4 爬虫代码 34

图5.5 服务器1爬虫过程 34

图5.6 服务器2爬虫过程 35

图5.7 服务器3爬虫过程 35

图5.8 中心节点对新注册节点的监听 36

图5.9 中心节点对任务节点返回消息的监听 36

图5.10 ER图 37

表目录

表4.1 用例-监听新注册 19

表4.2 用例-分配任务 19

表4.3 用例-监听返回消息 20

表4.4 用例-注册 20

表4.5 用例-接收任务 20

表4.6 用例-执行任务 21

表4.7 用例-返回消息 21

表5.1 ListenerNode类 26

表5.2 ListenerReturn类 26

表5.3 center类 27

表5.4 Connector类 27

表5.5 JMSclient类 27

表5.6 Register类 28

表5.7 Listener类 28

表5.8 RequestCrawler类 29

表5.9 ReCrawler类 29

表5.10 Sendback类 30

表5.11 IPTask类 30

表5.12 Dao类 30

表5.13 JDBC类 31

第一章 引言

## 1.1 研究背景

### 1.1.1 SourceForge.net

SourceForge.net是一个开源软件平台，用户不仅可以在网站上搜索到很多实用的软件，还能够给软件评分，向软件的开发者提需求建议等等。这些项目都有各自的适用领域、特征以及优缺点，用户可以通过搜索关键词，查看分类目录或者根据网站的推荐来找到自己的目标软件。

### 1.1.2 需求复用

对于一款产品，明确其需求是最关键的一步，可是客户的需求往往空乏而且充满不确定性，这给需求分析带来的巨大的挑战。不仅如此，客户与开发人员之间的沟通也存在隔阂。对于一项新的需求，开发人员需要从头开始调研、分析、建模，这会耗费很多精力。如果能够复用其他类似项目的需求，以他们开发人员的需求分析作为参考，就能在减少需求分析所需时间的同时还提高精确度。

常见的需求复用的原型有以下三种[1]：

1. 旧的经验，没有任何文字记录，采用口耳相传的方法。

2. 已有的需求规格书，重用已经有的文档和规格。

3. 领域模型。

### 1.1.3 实验项目

本合作实验就是在对SourceForge.net做需求复用的基础上展开的，现阶段有两个目标：

1. 根据网站上开源项目的描述、特征、所在目录和用户需求对项目精准定位。

2. 采用领域工程的方法，对各个类别下的开源软件进行领域分析，建立特征模型。最终实现让软件开发人员直接根据需求和特征模型就能找到合适的开源软件进行需求复用。

我个人的项目是这个实验整体的第一步：获取数据。

### 1.1.4 爬虫简介

在如今信息爆炸的时代，获取有用信息对于互联网的使用者来说成为了越来越复杂的一件事。它们分布在不同的网页中，那么从这些网页上爬取信息就成为了一个重要的议题。网络爬虫是一项从网页中获取信息的技术。早期的爬虫程序运行在单台计算机上，爬虫程序设置url队列，先将要爬取的url加入队列，然后从队列中依次取出url，爬取相应网站的内容[2]。但是随着信息量的剧增和需要分析的数据的范围持续扩大，在单一计算机上运行的爬虫程序已经无法满足客户的需要，这时将爬虫程序与分布式系统结合以处理大量数据的思路就应运而生。

分布式系统是由一组通过网络进行通信、为了完成共同的任务而协调工作的计算机节点组成的系统。它的目标是利用软件使多台普通的机器链接在一起，协同解决大量数据的处理和检索问题[2]。其思想是是把需要大量计算的，单个计算机无法完成的任务分配给若干个较小的模块，让多台计算机分别处理，统一储存运算结果以便接下来分析数据。分布式系统关注系统的可扩展性、可用性、可靠性、负载均衡、一致性等。

对于分布式爬虫，简单来说就是让每台服务器各自爬取一部分数据，最后将结果汇总起来。它关注任务的分配和调度，要求在提高整个系统爬虫效率的同时保证系统的可扩展性、可靠性和负载均衡。当然，由于可以使用不同的分布式架构，不同的分布式爬虫系统之间也千差万别。

## 1.2 国内外研究现状

### 1.2.1 java爬虫框架

Nutch：专为搜索引擎设计，以Apache Hadoop数据结构为基础。它提供了可扩展的功能接口，使得其对各种网页内容的解析，各种数据的采集、查询、集群、过滤等功能能够方便的进行扩展[3]。

Heritrix：可配置参数多，可定制性强，但是使用起来比较复杂。

### 1.2.2 python爬虫框架

scrapy：使用python开发的最流行的爬虫框架，使用布隆过滤器对url去重，底层使用异步框架twisted，支持高并发，很好地契合了搜索引擎的要求。

pyspider：与scrapy相比较轻量级，能够基于web查看运行状态，支持抓取javascript的页面。

## 1.3 本文的主要研究内容

在网络上可以找到一些开源的分布式爬虫框架，但是这些项目项目实现复杂，很多功能对于本项目来说是冗余的，所以我选择独立实现一个分布式爬虫系统。本文主要阐述了分布式爬虫系统不同维度下的分类、本项目采用的技术、需求分析、体系结构以及项目的背景与展望。该系统具有内部逻辑清晰、实现修改简单、使用过程友好等特点。

## 1.4 论文的主要工作和组织结构

论文主要分为六个章节，分别为：引言、分布式爬虫综述、技术基础、需求分析与概要设计、模块详细设计与实现、总结与展望。

第一章：引言。介绍项目的背景和研究内容。

第二章：分布式爬虫综述。介绍分布式爬虫系统的不同架构。

第三章：技术基础。介绍本项目所涉及的各项技术原理。

第四章：需求分析与概要设计。介绍项目的功能需求和性能需求，描述系统的结构、运行流程以及针对性能需求做出的设计。

第五章：模块详细设计与实现。介绍系统主要模块的类结构，展示关键代码和系统运行截图。

第六章：总结与展望。对项目进行总结，以及对项目未来的设想与展望。

第二章 分布式爬虫综述

## 2.1 分布式爬虫类型

分布式爬虫根据需求和数据量的不同可以大致分为以下几类[4]：

### 2.1.1 应用于搜索引擎

应用于搜索引擎中的分布式爬虫系统基于超大规模的互联网集群进行数据采集，对于爬取效率要求很高。

原理：包含重要信息和很多链接的网页的地址称为种子url，爬虫系统将种子url保存到队列中，然后进行进出队列操作，迭代爬取信息。由于数据量过大，全部爬取耗时过长，大部分时候需要借助可靠的url评价算法，优先爬取内容质量较高的网页。同时对于这些较为重要的网页需要定时更新，将有变动的网页的url加入优先级较高的待爬取队列中。

有一个很重要的工具是bloom filter[5]，它有一个与url总量匹配长度的二进制向量和一个合适的hash函数，用于检索一个元素是否在集合中。因为在大范围的网页中存在很多url链接到同一网页的情况，如果对这些网页重复爬取会造成资源浪费，甚至有可能导致无限循环。所以爬虫系统在采集完一个网页的信息后就把它的url用一个hash函数映射到向量中，在爬取一个网页之前先检查这个网页的url是否在向量中。这两种操作都只需一个映射函数即可完成，把在数据库中查找的O(n)时耗降低到O(1)，大大提高了效率。如果url已经被映射到向量中，那么一定会被查询到。但是如果url未被映射到向量中，也有很小的查询结果显示url在向量中的误判几率。

### 2.2.2 基于主题的爬虫

此类爬虫会预先定义好的与主题相关的页面。任务节点只需要爬取网页中特定的部分，以及在种子url中找到特定的用于迭代的url（即不需要构成一颗完整的url树）。因为目的性强，节省了浪费在无关网页上的资源。

### 2.2.3 用户个性化爬虫

普通的搜索引擎对于所有人来说都返回相同的结果，这样实现虽然简便但是有可能因为结果不对用户胃口导致用户流失。更好的方式是基于用户的行为偏好提供个性化的推荐。此时需要基于用户个性化的爬虫来作为信息采集系统，这样的系统只需要获得用户所浏览网页的主题即可，不需要在网页上爬很多具体信息。

因为本项目的目的是为整个需求复用项目提供数据支持，所以已经预先定义好了要爬取的数据，而不需要做类似搜索引擎的大范围数据爬取，所以使用的是基于主题的爬虫。

## 2.2 MapReduce

对于一个大到难以独立完成的项目，一般思路就是分而治之：分解，逐个完成，最后对各个结果汇总。这个思想也正是分布式系统的核心。“Map”指把具体的任务分解成多个小模块映射到不同服务器上，这些服务器并行计算以达到快速完成任务的效果；“Reduce”指存在一个或多个主节点得到所有模块的运行结果，然后对这些结果进行汇总整理。

对于分布式系统，显而易见的是整个系统出错的概率是相当大的。在系统所包含的多台服务器和多段网络中，即使单台服务器出现故障或者单段网络通信被阻断也都会影响到整个系统的性能。MapReduce的思想固然简单，但是一些基于MapReduce的分布式框架为了提高系统的容错性做了很多工作。除了Map和Reduce模块，还有Tracker负责监控模块的工作。它分为TaskTracker 和JobTracker两种[6]。TaskTracker的职责是发送任务进度反馈以及周期性地发送“心跳”信号给 JobTracker 以通知系统该节点当前的状态。JobTracker的职责是调度各个节点以监控他们的运行状态。如果由于网络延迟、服务器宕机或者其他不可控的原因导致节点任务失败，JobTracker 可以在不同的 TaskTracker 重新分配该任务。而对于该节点，系统能够使它会退到出错之前的状态。

MapReduce的节点监控如图2.1所示。

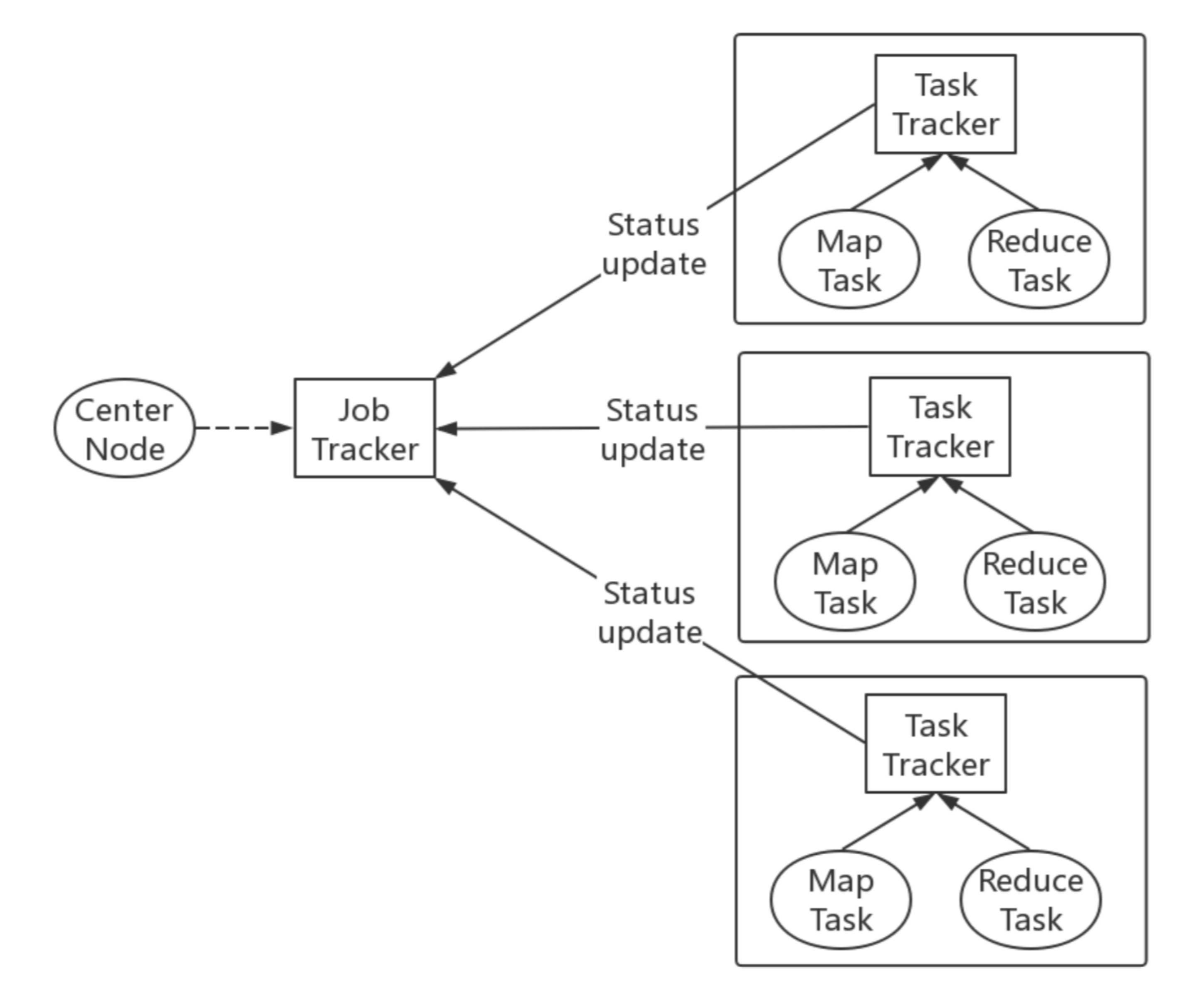


图2.1 MapReduce的节点监控

## 2.3 分布式爬虫系统的架构

分布式爬虫系统的架构是项目的核心，使用怎样的架构直接决定了爬取规则和系统性能。有三种常用的分布式爬虫架构：主从式，自治式和混合式[7]。

### 2.3.1 主从式

中心节点负责调度和发送任务给任务节点，所有任务节点并行工作，之间无通信，只与中心节点通信。

优点：调度层次清晰，中心节点清楚所有任务节点的运行状态，利于管理；缺点：所有任务节点都要到通过中心节点分配任务，当存在较多任务节点时容易产生堵塞，而且若中心节点崩溃整个系统也会崩溃。

主从式架构如图2.2所示。

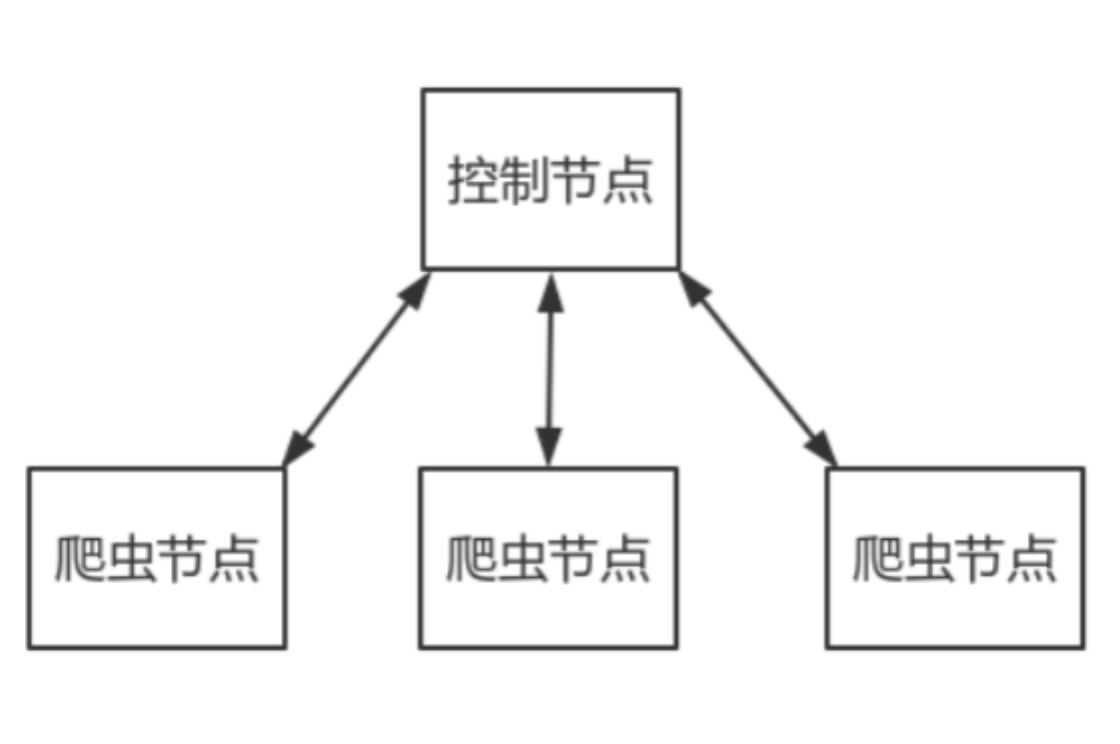


图2.2 主从式架构

### 2.3.2 自治式

不存在独立的中心节点，每个节点自身既要负责分配任务也要负责爬取。所有节点地位相同，每个节点维护一个其他节点的地址列表，需要通信时直接进行数据传输。当节点发生变化时，每个节点均要修改该地址列表。每个节点有自己的爬取队列，如果解析出来的url是自己要爬取的就自己完成，否则将该url发送给相应的节点。

优点：系统可用性高，不会因为主节点的崩溃而导致整个系统的崩溃；缺点：系统结构复杂，通信开销大，通信数量随着节点的增加呈指数增加。

自治式架构如图2.3所示。

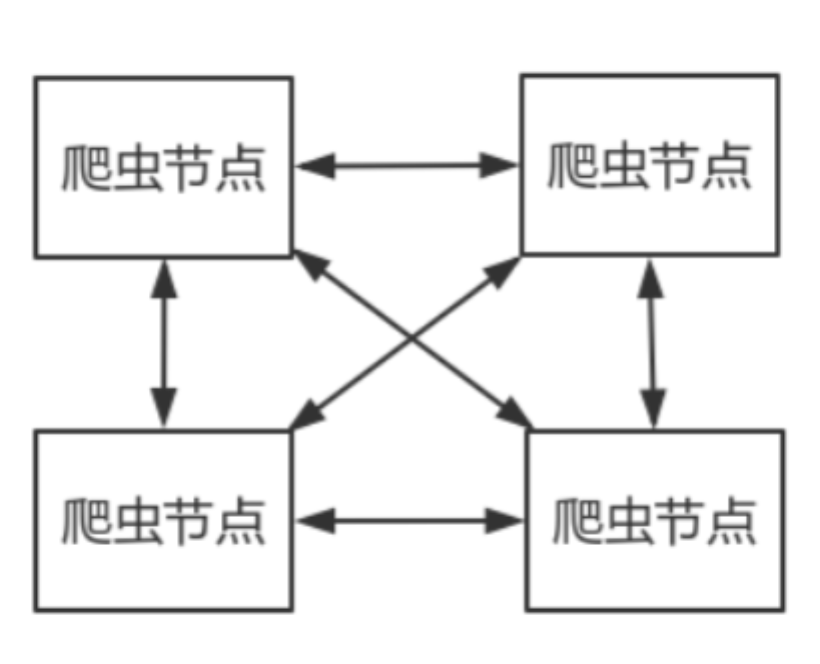


图2.3 自治式架构

### 2.3.3 混合式

主从式与自治式的折中模式，各个任务节点包括了自治式架构中任务节点的功能：分配任务和爬取信息；同时中心节点对各个任务节点交互后仍无法分配的任务重新分配。每个节点需要维护所有其他节点的地址列表和任务列表。

本项目采用的是主从式架构，因为各个爬取节点完成的是相同类型的任务，不需要模块间的通信。

## 2.4 爬取策略

分布式爬虫系统的运行流程简而言之有以下四步：

1. 整理要爬取的url并加入队列；

2. 分配任务；

3. 执行任务；

3. 整合数据。

队列中的每个url并非只代表一个网页，而是这个网页及它所包含的链接。所以队列中的url可以被称为种子url，要爬取的网页根据这些种子url各自形成树状结构。所以在第一步结束之后，有两种显而易见的爬取策略可供选择[5]：广度优先和深度优先，以及在这两种基础上做优化的启发式策略。

广度优先：所有url逐层爬取，不断迭代。每一次迭代只爬本url下的内容，并将它所包含的url返回给中心节点存储到队列中。重复这个过程直至清空队列。

深度优先：按照url列表依次爬取每棵url树内的所有信息，重复这个过程直至清空队列。

启发式策略：先运用可靠的url评价方法，对url质量进行评估，优先选择质量较高的url进行爬取；再进行固定深度的爬取，以免一颗url树过大消耗太多资源或者产生死循环。例如固定深度为4，则最多访问到种子url的第四层就先停止，开始访问下一棵树。

本项目采用的是广度优先爬取策略。

## 2.5 部署网络

分布式爬虫系统根据节点所在位置可分为两类：基于局域网部署或者基于广域网部署[5]。

基于局域网部署：所有节点均在同一个局域网内部，通过一个网络接口访问外部资源。优点：局域网内部带宽高，能保证节点之间的通信效率；缺点：所有与外部通信的网络负载都集中在一个对外网络端口，若爬取的网站不在局域网内，则系统整体效率会受到这个端口带宽的限制。

基于广域网部署：不同节点部署在不同的网络区域中（例如不同的云服务器）。优点：在一定程度上分散了爬取网页的流量上限，不存在与外部通信的网络瓶颈；缺点：不同节点间的通信效率比较低，所以要减少节点间通信。

本项目部署在广域网上，结合主从式架构避免了任务节点之间的通信，降低了通信成本。

第三章 技术基础

## 3.1 网络通信技术

对于分布式系统来说，网络通信是必不可少的部分，所以选择何种方式进行通信是一个值得思考的问题。现在比较流行的网络通信方式有以下几种：RPC、WebService、RMI、JMS等。

### 3.1.1 RPC

RPC即远程过程调用（Remote Procedure Call），它使用C/S方式，采用TCP或UDP协议传输数据，发送请求到服务器，等待服务器返回结果[8]。它的优点是跨语言跨平台；缺点是不支持对象，也不支持异步调用。客户端发出请求时服务器端必须已经处于开机等待状态，否则会出现无法连接的错误信息。

### 3.1.2 WebService

WebService是一个基于http协议的远程的服务提供者，为了支持跨网络的机器间相互操作交互而设计，使用开放的XML标准来描述、发布、发现、协调和配置应用程序[9]。WebService有跨编程语言和跨操作系统平台的特性，即网络通信的应用如果是依据WebService规范实施的，那么无论它们所采用怎样的语言、平台或内部协议，都可以相互交换数据。这些规范包括SOAP（用来描述传递信息的格式）、WSDL（描述如何访问具体的接口）以及UDDI（用来注册和查询WebService）[10]。WebService大体上分为5个层次:

1. Http传输信道

2. XML的数据格式

3. SOAP封装格式

4. WSDL的描述方式

5. UDDI注册和查询

WebService模型如图3.1所示。

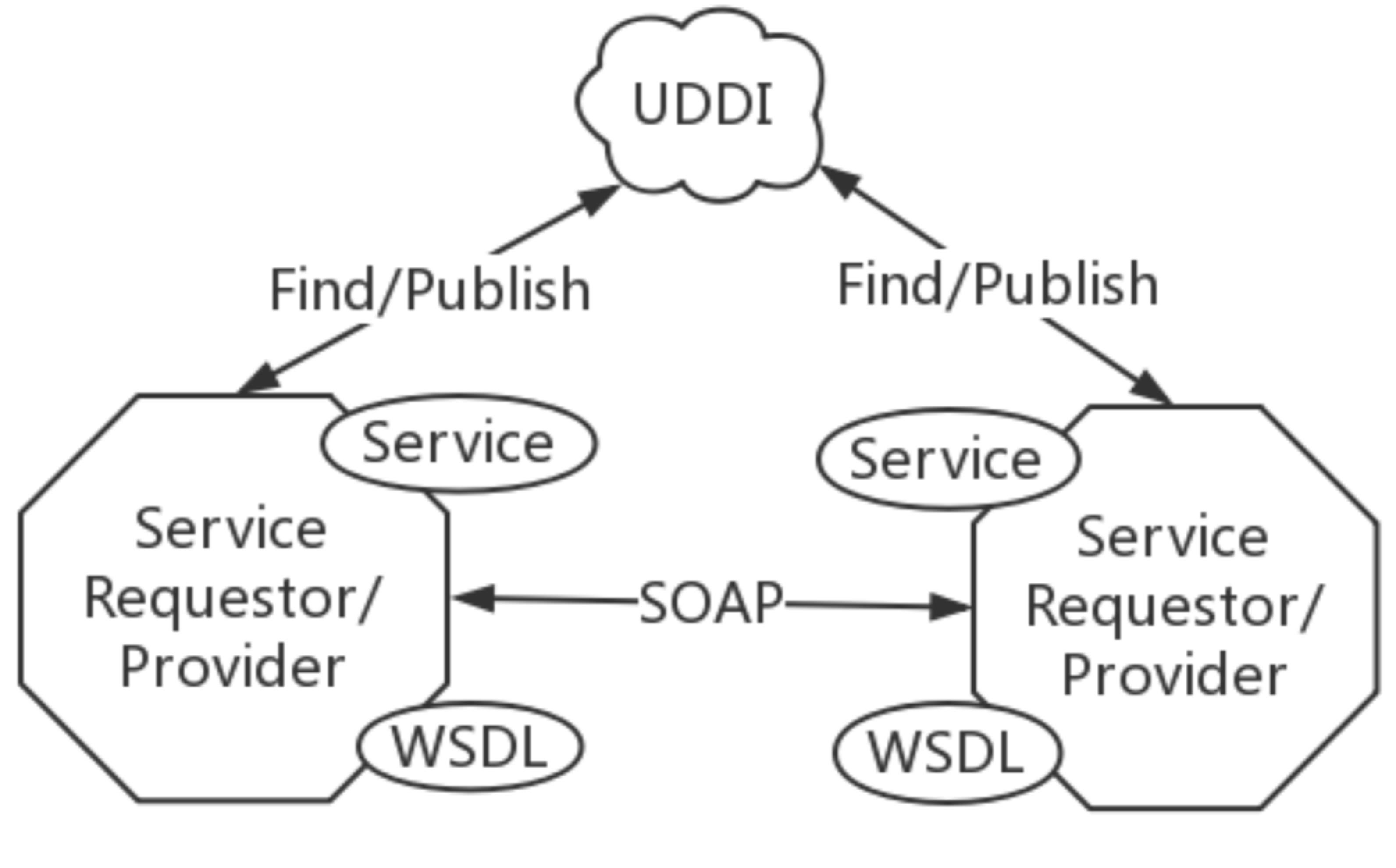


图3.1 WebService模型

### 3.1.3 RMI

RMI即远程方法调用（Remote Method Invocation），是java的一种远程通信机制，能够让在某个java虚拟机上的对象像调用本地对象一样调用另一个java 虚拟机中的对象上的方法[11]。RMI采用代理来负责客户端和服务器之间socket通信的细节，由服务器代理先注册RMI服务，然后客户端代理通过协定好的地址，服务名和端口来找到该服务，之后就可以直接调用远程方法。RMI的优点是可以传递Java对象类型的数据，而且编译期可检查错误；缺点是客户端和服务器耦合紧密，client端发起调用后，必须等待server处理完成并返回结果后才能继续执行。

RMI模型如图3.2所示。

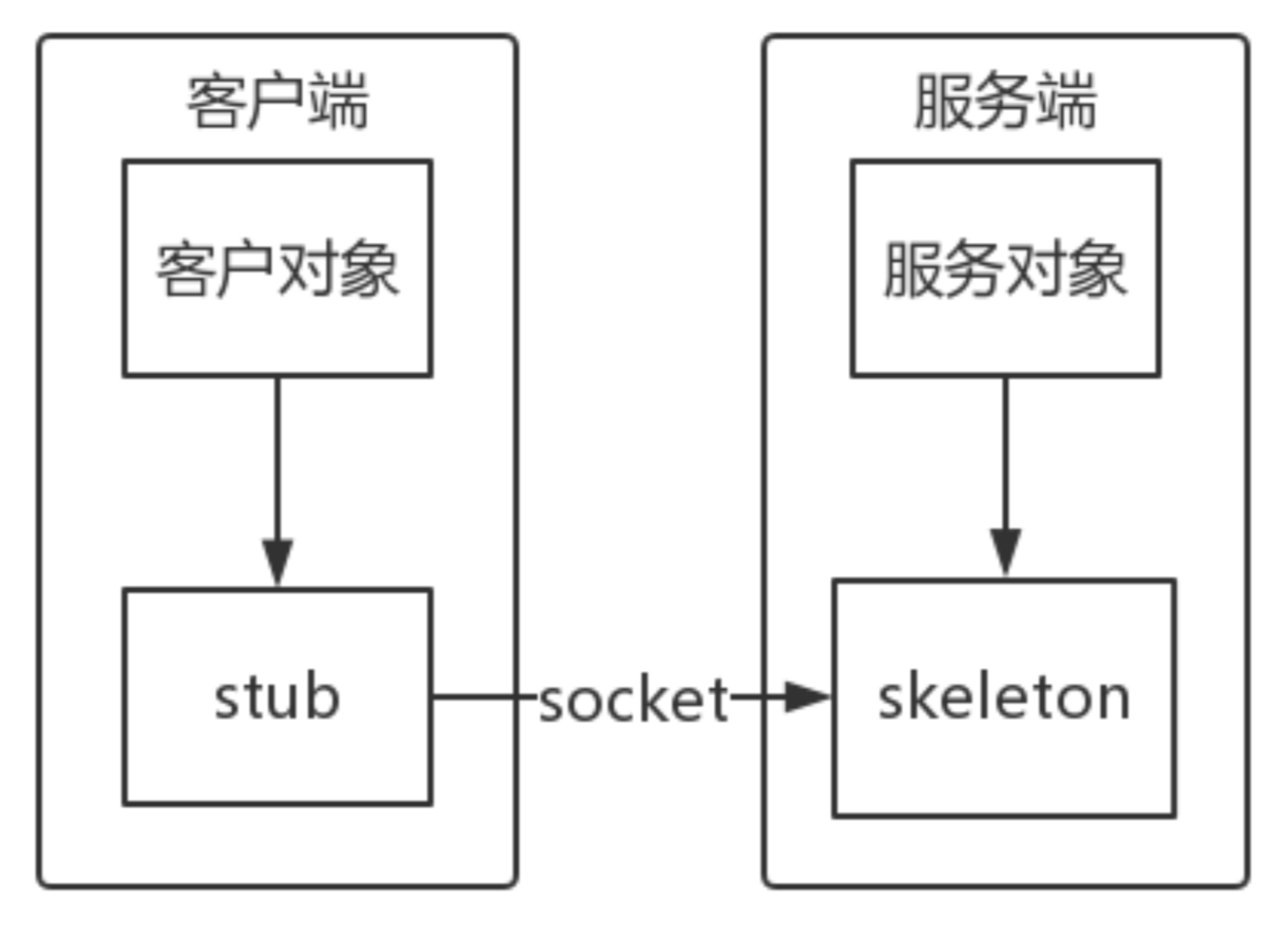


图3.2 RMI模型

### 3.1.4 JMS

JMS即Java消息服务（Java Messaging Service）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，用于在应用程序之间，或分布式系统中发送消息，进行异步通信，提供可靠消息传输、事务和消息过滤等机制[12]。它使用了消息队列模型，把进程间的交互转化为对消息的处理。JMS服务器会提供队列来对这些消息进行暂存。每个进程都可以访问任意个队列，从队列读取或向队列写入消息。队列决定了消息的路由，这样就把WebService中复杂的路由问题，变成了如何管理静态的队列的问题，大大降低了复杂度。

JMS支持两种消息模型：Point-to-Point (P2P)和Publish/Subscribe (Pub/Sub)，即点对点和发布订阅模型[13]。

在点对点消息模型中，应用程序由消息队列、发送者和接收者组成。每个消息发送给一个指定的消息队列，被一个接收者接收，而且消息一旦被接收就会消失。如果消息已经放入队列但还未被接收，那么消息会一直存放在队列中直到被接收者接收。消息发送者和接受者没有时间依赖性，接收者收到消息后会发送确认收到通知。

点对点消息模型如图3.3所示。

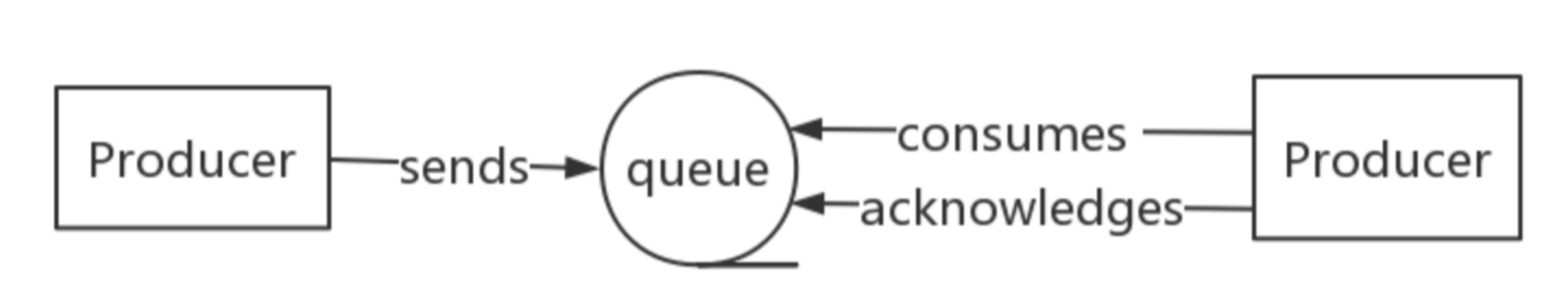


图3.3 点对点消息模型

在发布/订阅消息模型中，应用程序由主题、发布者和订阅者组成。发布者发布一个消息，该消息通过主题传递给所有的订阅者，订阅者是匿名的且可以动态发布和订阅主题。该模型与P2P的不同在于这里一个消息可以传递给多个订阅者，但是发布者和订阅者有时间依赖性。订阅者只能消费自它订阅之后发布者发布的消息，且需要一直保持活动状态以接收消息。

发布/订阅消息模型如图3.4所示。

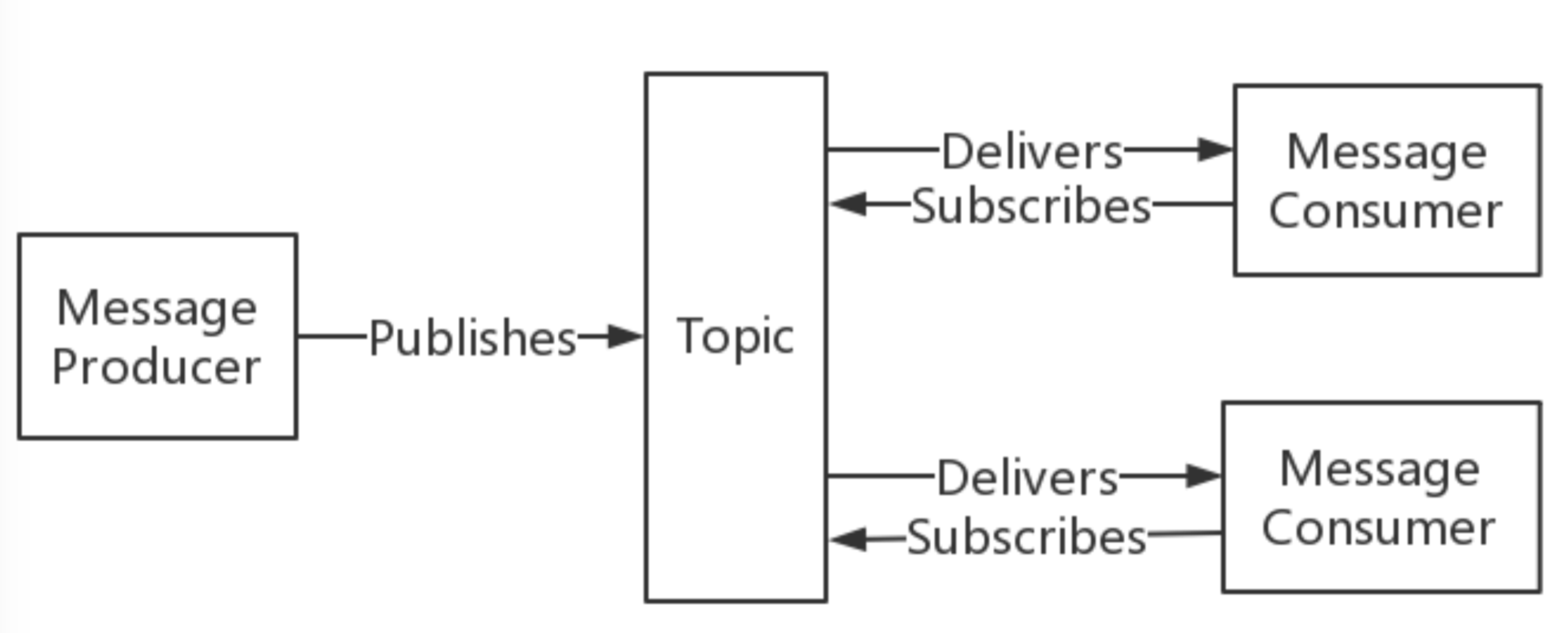


图3.4 发布/订阅消息模型

JMS应用程序由如下基本模块组成：

ConnectionFactory 连接工厂

客户端使用一个连接工厂对象连接到消息服务器，它设定了消息服务器的地址，用户名和密码，用于创建消息服务器和客户端之间的连接。

Connection 连接

连接指应用程序和消息服务器之间的通信链路。JMS客户端（包括发送者或接收者）会在消息服务器上搜索并获取该连接。客户端能够与目的地通过该连接通讯，向队列或话题发送或接收消息

Session 会话

会话指一个线程的上下文，用于发送和接收消息。它由连接创建，创建时可选择是否要传递收信方的确认收到通知(acknowledgement)。会话支持事务，用户可以在提交之前回滚取消消息。

Destination 目的地

目的地通过会话创建，指明发送者发送的目的地和接收者接收消息的来源。

MessageConsumer 消息消费者

由会话创建的对象，用于接收消息服务器的消息。

MessageProducer 消息生产者

由会话创建的对象，用于发送消息到消息服务器。

MessageListener 消息监听器

消息的默认事件处理者，在onMessage方法中定义了接收到消息之后的动作。

JMS服务器的连接过程如图3.5所示。

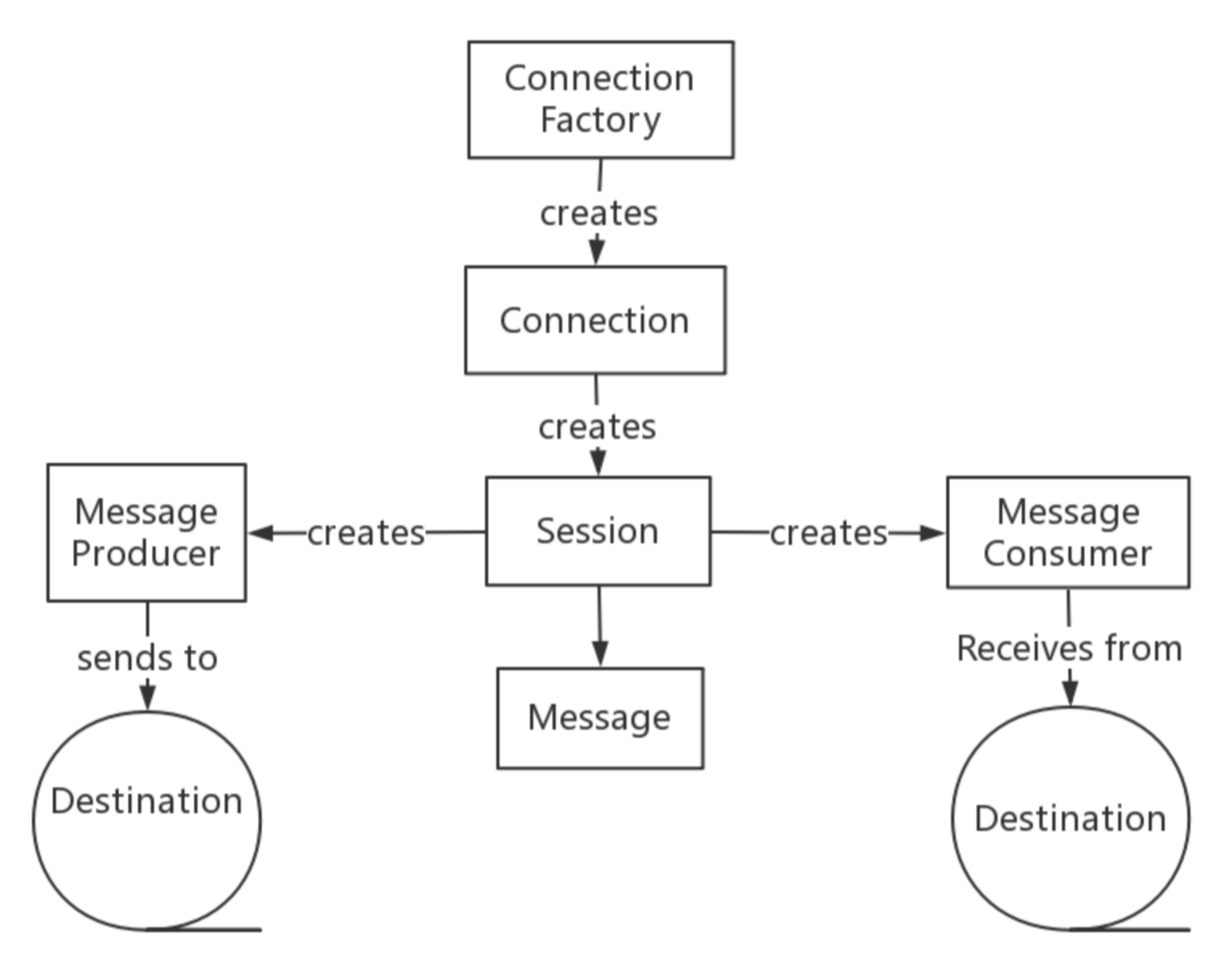


图3.5 JMS服务器的连接过程

消息类型有如下几种：

Text message：文本对象。

Object message：JAVA对象。

Bytes message：字节数据。

Stream message：java原始值数据流。

Map message：键值对。

比较而言，WebService直接使用网络API编写跨进程通讯，非常复杂而且容易出错。除了要编写大量的底层socket代码外，还需要处理一系列问题，如：找到要交互数据进程的地址，保障数据包的完整性，进程意外停止后的重启。这些问题包含了扩展、容错、负载均衡等一系列分布式系统的性能需求。而RMI由于通信机制所限，适合处理小型远程调用事务。而且RMI的客户端和服务器耦合紧密，不利于作为大型分布式系统的通信机制。

JMS的优势在于它是异步而且可靠的。异步通信意味着客户端和服务器耦合度低，便与修改；可靠性体现在JMS消息只递送一次且一定能送到，能够避免创建重复消息引发的问题。

### 3.1.5 activeMQ

MQ即消息队列（message queue），是一种应用程序之间的通信方法。有通信需求的服务器只需对队列进行读写操作，而不需要调用彼此的方法，这是异步通信的先决条件。因为直接调用远程方法不仅需要远程服务器已经运行等待被调用，还要求客户端持续等待回馈，意味着客户端此时无法处理其他任务。而在消息队列模式中，通信双方都有专门的监听器用来监听队列中传递给自己的消息，不需要处理任务的主进程停下等待。

ActiveMQ是由Apache出品的开源消息总线，完全支持JMS1.1和J2EE 1.4规范。ActiveMQ是面向消息的，具有跨语言和跨系统的特点[14]。它有如下几个特性：

连接方式的多样化：提供了包括HTTP/S、SSL、TCP、UDP等在内的广泛的连接模式。

有序性：确保消息能够按照发送的次序被消费者者接收。

延迟接收：消息加入队列后如果消费者没有开启连接，那么当消费者开启连接后，消息中介将会向其提交之前未处理的消息。

## 3.2 队列

在分布式系统中，存在很多多个消息由一个程序处理的情况，例如前文提到的中心节点需要处理任务节点的反馈信息。然而如果反馈信息过多，就需要一个队列来暂时存储这些消息，同时挂起一些线程。一旦条件满足，再唤醒这些线程继续处理消息。Java的BlockingQueue是一个很好用的阻塞队列，它自动处理对线程的阻塞和唤醒，大大减少了开发人员的工作量。BlockingQueue有如下几种常见的类型[15]：

ArrayBlockingQueue：使用一个定长数组作为缓存队列。生产者放入数据和消费者获取数据时，共用同一个锁对象，意味着生产和消费无法并行运行。

LinkedBlockingQueue：使用一个链表作为缓存队列。只有当队列缓冲区达到最大值缓存容量时，才会阻塞生产者队列。直到消费者从队列中消费掉一份数据，生产者线程才会被唤醒。

## 3.3 线程池

对于并发线程数量很多，并且每个线程都是执行一个较短的时间就结束的情况，频繁地分派新线程会大大降低系统效率。线程池可以重复使用线程，线程执行完一个任务后不销毁，而是回收等待下一次调用。线程池可以自动或手动设置线程数量，达到运行的最佳效果：多了造成系统拥挤，效率不高；少了浪费了系统资源，cpu时间空闲。

线程池的核心是ThreadPoolExecutor类[16]，它有如下几个重要的成员变量：

corePoolSize：核心池的大小。

maximumPoolSize：线程池中允许的最大线程数。

关于以上两个变量的解释：创建了线程池后，线程池中初始的线程数为0。有任务到来时，线程池就会新建一个线程去执行任务。当线程池中的线程数目达到corePoolSize后，再到达的任务就会被加入缓存队列当中。当缓存队列无法继续存储任务时，线程池就会新建线程以执行再到达的任务。当线程池中的数目达到maximumPoolSize后，就会拒绝再到达的任务。

线城池的工作原理如图3.6所示。

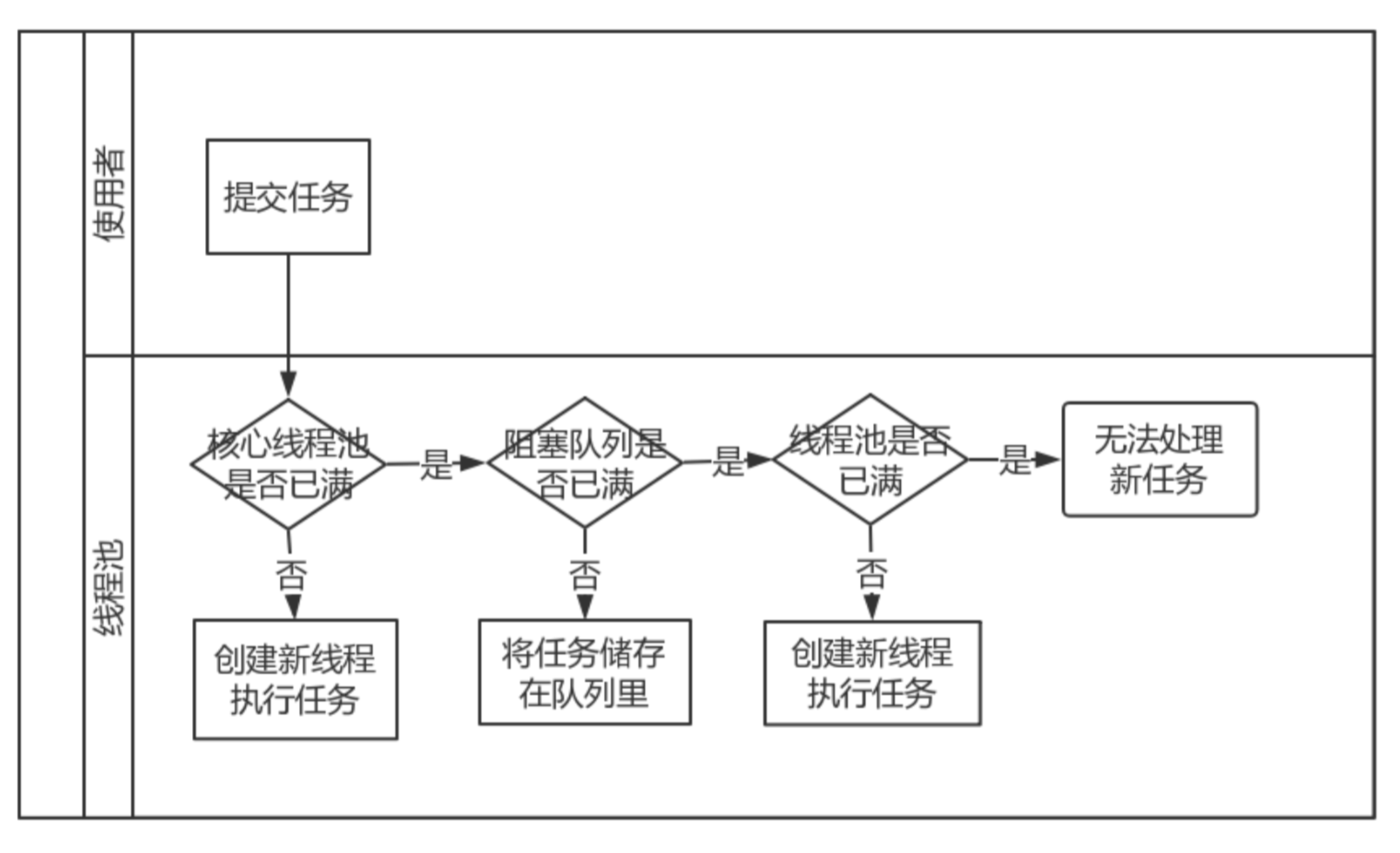


图3.6 线程池的工作原理

keepAliveTime：指当线程池中的线程数大于corePoolSize且处于空闲状态时，这些线程最多保持多久时间会终止。

workQueue：用来存储等待执行的任务的阻塞队列。

手动配置一个线程池是比较复杂的，不过好在Executors类里面提供了一些静态工厂。一些常用的线程池如下：

FixedThreadPool：固定大小的线程池。这个线程池的corePoolSize和maximumPoolSize相同，workQueue采用无界队列。这样能保证总线程最大个数不变，暂时无法处理的任务全部被存到队列中。不过存在任务数量大，瞬间填满队列导致占满内存的情况。

SingleThreadExecutor：一个单线程的线程池。与FixedThreadPool类似，只是corePoolSize和maximumPoolSize都被设成1，使这个线程池只有一个线程在工作，相当于单线程串行执行所有任务。这样做的意义是如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。该线程池的优点是可以保证所有任务的执行顺序按照任务的提交顺序执行。

CachedThreadPool：可缓存的线程池。其corePoolSize为0，maximumPoolSize不限制，意即不会对线程池大小做限制，线程池大小完全依赖于JVM能够创建的最大线程数量。workQueue使用SynchronousQueue，这个队列没有存储空间，这意味着只要有请求到来，就必须要找到一条工作线程处理他，如果当前没有空闲的线程，那么就会再创建一条新的线程。

## 3.4 jsoup

jsoup是一个HTML解析器，可直接解析某个URL地址或者HTML文本内容。对于得到的document对象，可以使用全面的方法来遍历和选择元素，同时，jsoup能够自定义http请求的头文件。本项目使用jsoup作为网页解析器来寻找所需的数据。

第四章 需求分析和概要设计

## 4.1 功能需求分析

### 4.1.1 需求概述

经过分析，我决定在不同的网页内分别爬取开源项目的不同数据，这些网页类别包括：目录页，详情页，用户需求页。先根据目录页得到所有项目最基本的信息：名字和url，以及项目所在的页码--得到页码主要是为了处理数据方便。将项目url作为种子url放入爬取队列，然后按照广度优先策略分三次迭代：第一次迭代在项目详情页爬取项目的基本信息，如项目简介、特征、评分等等；第二次迭代在项目详情页爬取每个项目的推荐项目；第三次迭代在用户需求页爬取用户对于该项目提出的改进需求。

每一页要爬取的数据如下：

目录页（如https://sourceforge.net/directory/page=1）：名字，所在页码，主页。

详情页（如https://sourceforge.net/projects/clonezilla）：所属目录，简介，特征，评分，周下载量，项目主页（指的不是在sourceforge.net上的主页，而是项目自身的主页，比如有的项目在github上的地址），最后更新时间，注册时间，平台或系统，目标用户群，开发语言，三个推荐项目的名称和主页。

用户需求页（如https://sourceforge.net/p/clonezilla/feature-requests/70）：需求精要，标签，优先级，创建时间，状态（已解决或未解决），内容。

执行过程为：主节点选择要爬取的页码作为信息发送到JMS服务器activeMQ上，任务节点从activeMQ中接收信息，根据页码在数据库查询要爬取的url，执行爬虫任务。任务执行完后，任务节点发送返回信息到activeMQ上，主节点接收到返回信息后分配新任务给该任务节点。系统执行过程如图4.1所示。

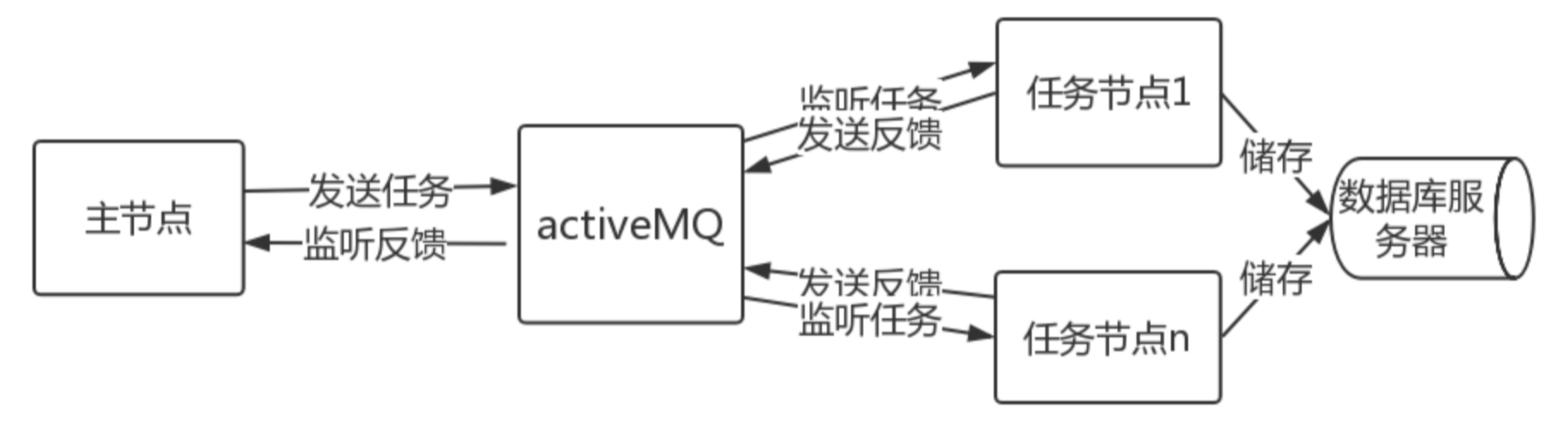


图4.1 执行过程

### 4.1.2 用例图

整个分布式爬虫系统整体上分为中心节点和任务节点两部分。中心节点的功能是监听新注册、监听返回消息和分配任务；任务节点的功能是注册、接收任务、执行任务和返回消息。

两部分的用例图如图4.2所示：

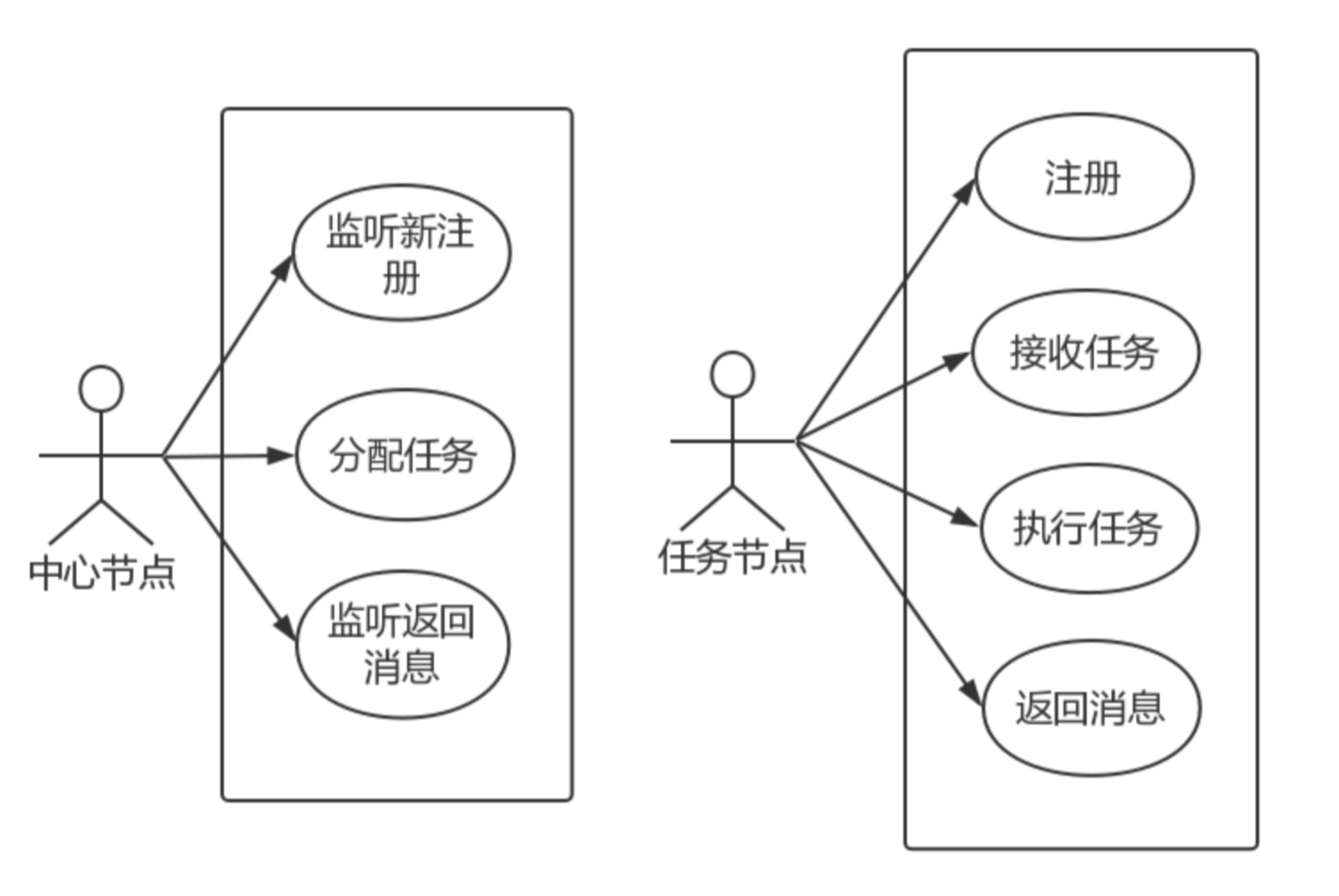


图4.2 用例图

### 4.1.3 用例描述

表4.1 用例-监听新注册

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 1 | 名称 | 监听新注册 |
| 参与者 | 中心节点，目的是监听新注册的节点发送到activeMQ的消息 | | |
| 前置条件 | 有新节点已经将注册信息发送到了activeMQ上 | | |
| 后置条件 | 生成节点注册Task | | |
| 正常流程 | 1. 从activeMQ中接收注册消息  2. 将消息发送给处理程序生成Task | | |
| 扩展流程 | 无 | | |
| 特殊需求 | 无 | | |

表4.2 用例-分配任务

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 2 | 名称 | 分配任务 |
| 参与者 | 中心节点，目的是给任务节点分配任务 | | |
| 前置条件 | 中心节点监听到任务节点的消息 | | |
| 后置条件 | 将任务消息发送到activeMQ上 | | |
| 正常流程 | 1. 生成Task  2. 连接activeMQ  3. 将Task加入activeMQ队列 | | |
| 扩展流程 | 2a. 连接时网络异常 | | |
| 特殊需求 | 中心节点使用线程池处理多任务节点情况的高并发 | | |

表4.3 用例-监听返回消息

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 3 | 名称 | 监听返回消息 |
| 参与者 | 中心节点，目的是监听执行完任务的节点发送到activeMQ的返回消息 | | |
| 前置条件 | 有任务节点已经将返回消息发送到activeMQ上 | | |
| 后置条件 | 生成节点任务Task | | |
| 正常流程 | 1. 从activeMQ中接收返回消息  2. 将消息发送给处理程序生成Task | | |
| 扩展流程 | 2a. 任务全部执行完，不生成Task | | |
| 特殊需求 | 无 | | |

表4.4 用例-注册

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 4 | 名称 | 注册 |
| 参与者 | 任务节点，目的是新任务节点注册到中心节点以执行任务 | | |
| 前置条件 | 无 | | |
| 后置条件 | 将注册消息发送到activeMQ上 | | |
| 正常流程 | 1. 连接activeMQ  2. 发送注册消息 | | |
| 扩展流程 | 无 | | |
| 特殊需求 | 该任务节点需要已经实现事先规定好的爬虫程序 | | |

表4.5 用例-接收任务

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 5 | 名称 | 接收任务 |
| 参与者 | 任务节点，目的是监听中心节点发送到avtiveMQ的任务消息 | | |
| 前置条件 | 中心节点已经将任务消息发送到avtiveMQ上 | | |
| 后置条件 | 任务节点开始执行新的爬虫任务 | | |
| 正常流程 | 1. 从avtiveMQ接收任务消息  2. 解析消息，调用爬虫程序 | | |
| 扩展流程 | 2a. 消息解析错误：发送返回消息给中心节点，消息中要注明消息解析错误 | | |
| 特殊需求 | 无 | | |

表4.6 用例-执行任务

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 6 | 名称 | 执行任务 |
| 参与者 | 任务节点，目的是执行爬虫任务 | | |
| 前置条件 | 任务消息被成功解析 | | |
| 后置条件 | 任务要求的数据被成功储存到数据库服务器中 | | |
| 正常流程 | 1. 根据url爬取网页上事先规定好的部分；每爬好一个网页将数据暂时存在队列中  2.连接数据库，将队列中的数据依次储存 | | |
| 扩展流程 | 1a. 网络连接超时：将这一页的url储存到本地数据库中，待其他网页爬取完后重新爬取。若仍然网络连接超时则抛弃之  1b. 网络连接报错404：抛弃之 | | |
| 特殊需求 | 每隔30秒更换代理IP，减少被网站封锁IP的可能性 | | |

表4.7 用例-返回消息

|  |  |  |  |
| --- | --- | --- | --- |
| ID | 7 | 名称 | 返回消息 |
| 参与者 | 任务节点，目的是在任务执行完后，返回“已完成”的消息给中心节点，等待中心节点再次分配任务 | | |
| 前置条件 | 爬虫任务执行完毕，数据已经储存到数据库服务器 | | |
| 后置条件 | 将返回消息发送到activeMQ上 | | |
| 正常流程 | 1. 连接activeMQ服务器  2. 发送返回信息 | | |
| 扩展流程 | 无 | | |
| 特殊需求 | 无 | | |

## 4.2 性能需求分析

### 4.2.1 可扩展性

分布式系统的设计初衷就是利用集群多机的能力处理单机无法解决的问题，让由成本低廉的PC服务器组成的集群，在性能方面能够达到或超越大型机的处理性能[17]。当业务变得更加复杂，处理的数据越来越多时，系统需要适应这种变化，支持高并发和海量数据处理。扩展系统性能有两种方式：一种是优化系统的性能或者升级硬件(scale up)，另一种是增加机器来扩展系统的规模(scale out)。好的分布式系统总在追求“线性扩展性”，即性能可以随集群数量增长而线性增长。

### 4.2.2 可靠性

软件系统的可靠性主要体现在三个方面：长期运行的稳定性、输出数据的正确性以及异常情况的可记录性[18]。普通系统的可靠性挑战来自于不确定性，而分布式系统由于异构的机器和网络，普遍的节点故障和不可靠的通信网络更放大了不确定性。所以采用一些技术手段提高分布式系统的可靠性是非常必要的。

### 4.2.3 负载均衡

分布式系统为了实现高并发，服务器的运行负荷需要大致相当，即任务的粒度接近，避免出现一些服务器任务的计算量过重占满CPU时间，而另一些服务器任务十分简单CPU基本空闲。负载均衡正是处理这一需求的技术，它用来在计算机集群、网络连接、CPU或其它资源中分配负载，以达到最佳化资源使用、最大化吞吐率、最小化响应时间、同时避免过载的目的[18]。

### 4.2.4 降低通信开销

对于分布式系统，一个良好的进程间通信机制很重要。因为系统不仅需要通信机制足够简便可以屏蔽复杂的底层网络通讯，也需要能够跨语言跨平台传递消息。这样的通信机制虽然可以应用，但是网络通信相对于单机计算总是缓慢且不可靠的。当系统内集群数量足够大时，网络通信开销容易成为系统效率的瓶颈，因此降低通信开销是十分重要的。

## 4.3 概要设计

### 4.3.1 系统结构图

系统将分布式爬虫抽象为调度、爬取数据和储存三个模块，分别是中心节点、JMS服务器和任务节点，每一个模块都有稳定的对外接口且隐藏了内部实现。两种节点的Listener都是负责监听消息。中心节点的Connector负责连接JMS服务器。任务节点的Register负责注册新节点，Crawler负责爬虫，Database负责存储数据。中心节点和任务节点只通过JMS服务器通信。系统结构如图4.3所示：

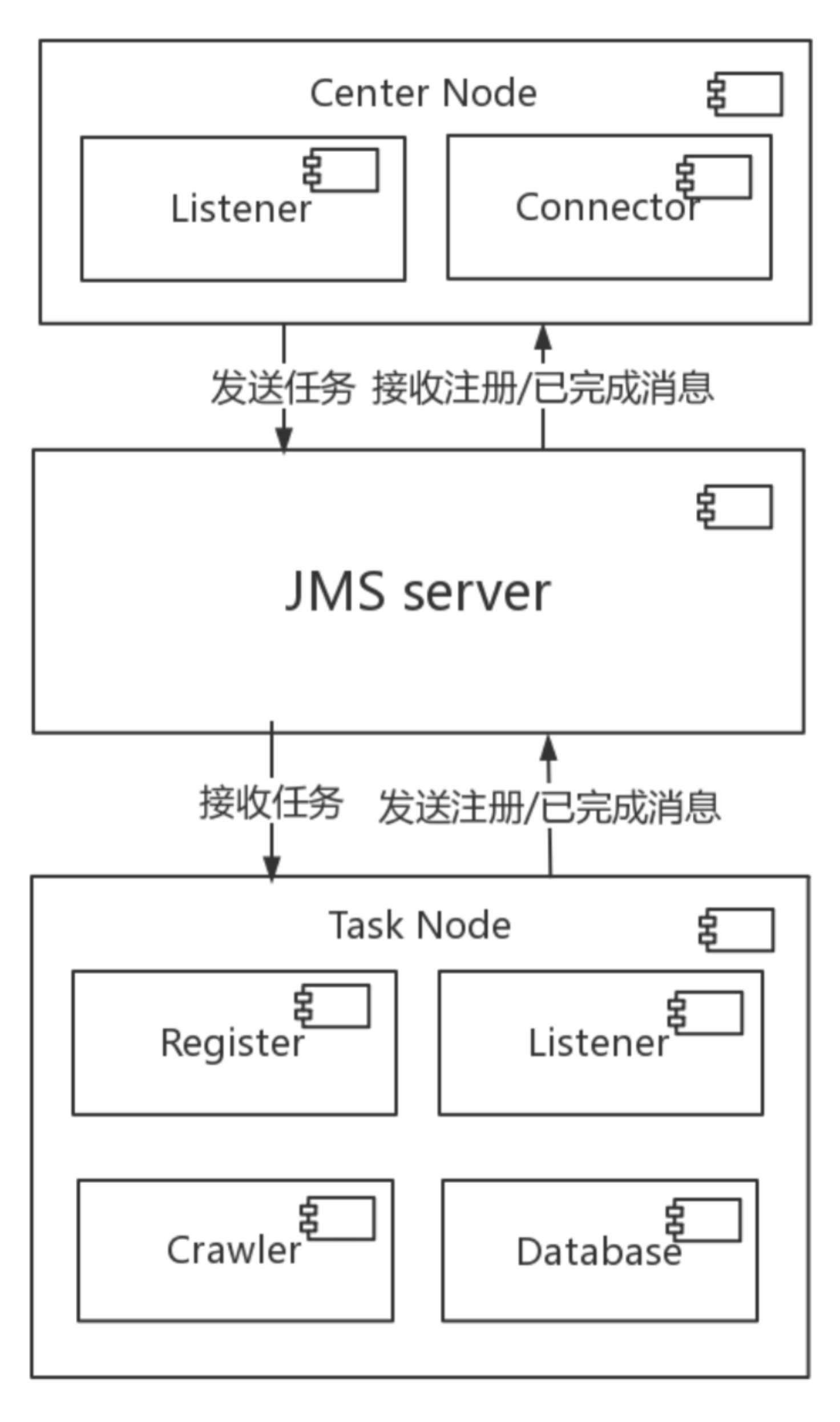


图4.3 系统结构图

### 4.3.2 流程图

系统流程图如图4.4所示。系统开始运行时，中心节点的两个监听和任务节点的一个监听均已开启。当有新任务节点注册时，中心节点监听到注册消息并将消息交予Center生成任务。之后Center连接JMS服务器，分配任务给该节点。任务节点会监听到消息，执行任务，储存数据，发送返回消息。中心节点会监听到任务节点的返回消息，生成新的Task，重复上面的步骤。

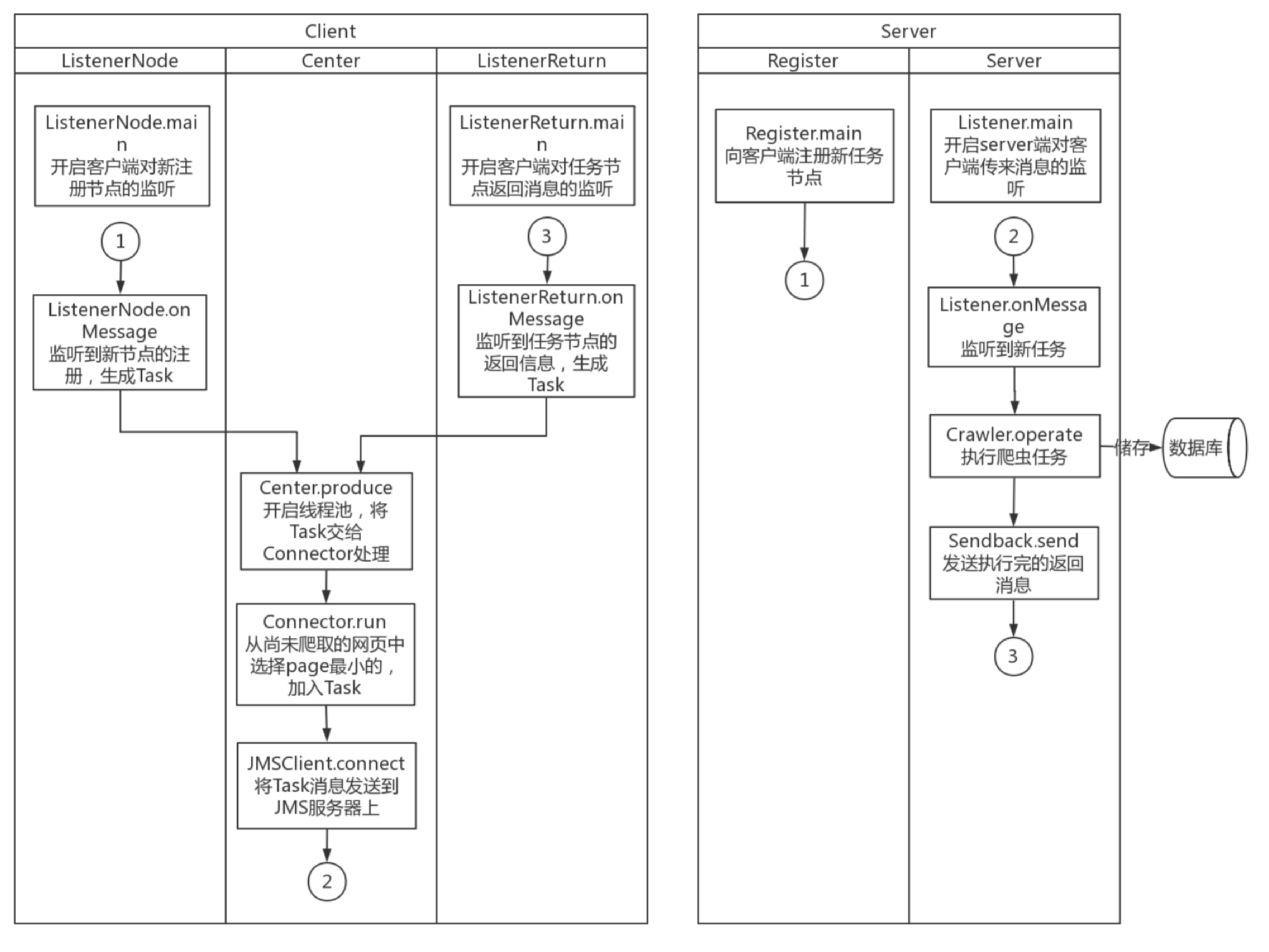


图4.4 系统流程图

### 4.3.3 性能需求的设计

**1. 可扩展性**

对于中心节点与任务节点联系的方式，最常用的是中心节点将各个任务节点的地址写在配置文件中。但在这种情况下，当有节点新加入或者退出时需要修改配置文件然后重新部署。如果更改频繁那么部署的繁琐程度是难以想象的，而且频繁地重新部署也严重影响了系统的可用性。本项目舍弃了常用的中心节点持有任务节点地址的模式，而采用任务节点自行注册到中心节点的方法，使任务节点的新增和删除都非常方便，能够在系统不停机的情况下直接完成，大大提高了系统的可扩展性。

**2. 可靠性**

**本项目对系统**长期运行的稳定性和异常情况的可记录性做了针对性处理，主要体现在以下两方面：

网页连接超时（read time out）：这是难以避免的错误，遇到这个报错时将当前爬取的url存入一个独立的表中，该表内所有的条目均是由于连接超时导致，需要重新爬取的url。等本次节点的任务执行完后再重新爬取这张表里面的url，如果仍然连接超时则将该url抛弃。

ip访问被限制：由于Sorceforge.net有一定的反爬虫机制，如果同一ip持续发送请求，网站就会在一段时间内限制你的访问，甚至还有可能将此ip永久加入黑名单。对于这种反爬虫措施，本项目采用的方法是持续更换ip地址。有了合适的代理服务器后，每用一个ip爬取数据一段时候后（30秒左右）就更换代理ip。

**3. 负载均衡**

本项目采用的方法是分配给每个任务节点粒度适中的任务以尽量达到负载均衡。粒度不能过粗，要让每个任务节点在可预期的时间段内执行完任务；粒度也不能过细，否则会导致信息传输过于频繁，让系统主节点的分发效率成为瓶颈。经过分析，本项目采用项目所在页码作为任务的标志。每一页有25个项目，用户需求的条数最大应该在10000以内，两小时内应该可以爬取完一页的项目。

**4. 降低通信开销**

对于高并发，较小的报文传输，有效利用网络资源，将极大的提高系统吞吐量。本项目使用了activeMQ的消息队列，无需自行实现底层TCP协议，降低成本。而且消息队列满足“高内聚低耦合”的设计原则。应用消息队列机制异步处理消息，通信双方只需要遵守事先协定好的消息规范发送消息，不需要了解对方是如何处理信息的，这大大降低了系统的耦合度。

第五章 模块详细设计与实现

## 5.1 模块详细设计

本部分会对系统各模块的类图、类描述与核心的代码实现作出说明。

### 5.2.1 CenterNode模块

中心节点类图如图5.1所示。

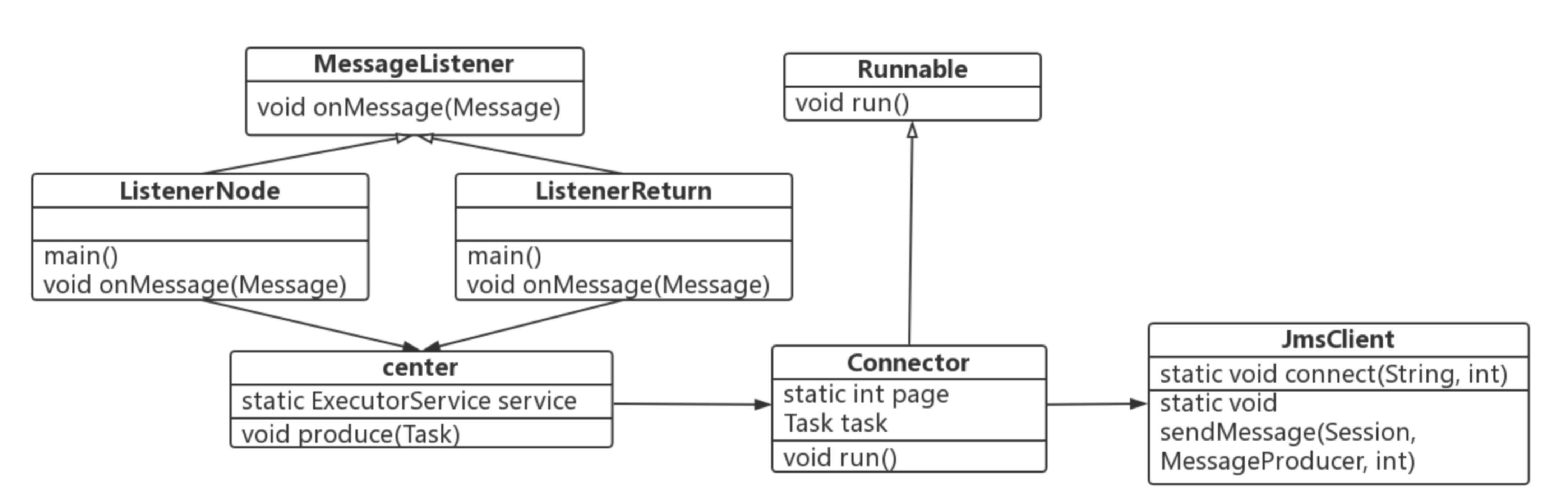


图5.1 中心节点类图

类描述：

表5.1 ListenerNode类

|  |  |
| --- | --- |
| 类名称 | ListenerNode |
| 类职责 | 监听新任务节点的注册，生成Task，交给center处理 |
| 重要数据结构 | 无 |
| 类方法 | void main() |
| 方法职责 | 连接JMS服务器 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | void onMessage(Message) |
| 方法职责 | 实现MessageListener的onMessage()方法，监听新节点的注册，生成Task，交给center处理 |
| 方法前置条件 | 已连接JMS服务器 |
| 方法后置条件 | 任务节点发送注册消息到JMS服务器上后，此方法能够监听到并生成Task |

表5.2 ListenerReturn类

|  |  |
| --- | --- |
| 类名称 | ListenerReturn |
| 类职责 | 监听任务节点的返回消息，生成Task，交给center处理 |
| 重要数据结构 | 无 |
| 类方法 | void main() |
| 方法职责 | 连接JMS服务器 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | void onMessage(Message) |
| 方法职责 | 实现MessageListener的onMessage()方法，监听任务节点的返回消息，生成Task，交给center处理 |
| 方法前置条件 | 已连接JMS服务器 |
| 方法后置条件 | 任务节点发送返回消息到JMS服务器上后，此方法能够监听到并生成Task |

表5.3 center类

|  |  |
| --- | --- |
| 类名称 | Center |
| 类职责 | 并发处理Task |
| 重要数据结构 | static ExecutorService service |
| 类方法 | void produce(Task) |
| 方法职责 | 用线程池调用Connector |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |

表5.4 Connector类

|  |  |
| --- | --- |
| 类名称 | Connector |
| 类职责 | 将爬取信息加入Task，把Task交给JmsClient处理 |
| 重要数据结构 | static int page |
| 类方法 | void run() |
| 方法职责 | 将爬取的页码加入Task |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |

表5.5 JMSclient类

|  |  |
| --- | --- |
| 类名称 | JmsClient |
| 类职责 | 连接JMS服务器，发送任务消息 |
| 重要数据结构 | 无 |
| 类方法 | void connect(String ip, int page) |
| 方法职责 | 连接JMS服务器 |
| 方法前置条件 | 无 |
| 方法后置条件 | 创建名字为任务节点ip地址的消息队列 |
| 类方法 | sendMessage(Session, MessageProducer, int page) |
| 方法职责 | 将任务消息发送到JMS服务器上 |
| 方法前置条件 | 已连接JMS服务器 |
| 方法后置条件 | JMS服务器的相应队列持有该任务消息 |

### 5.2.2 TaskNode模块

任务节点类图如图5.2所示。

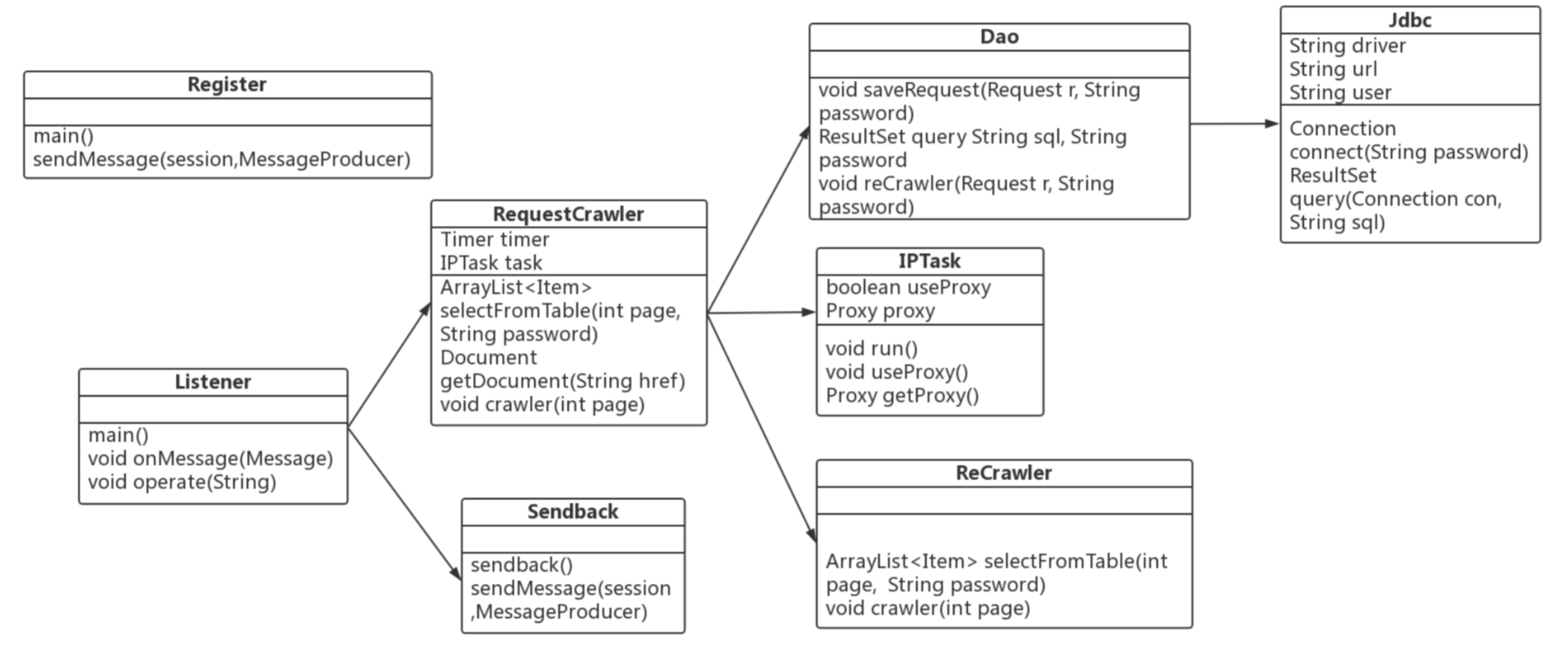


图5.2 任务节点类图

类表述：

表5.6 Register类

|  |  |
| --- | --- |
| 类名称 | Register |
| 类职责 | 注册新任务节点 |
| 重要数据结构 | 无 |
| 类方法 | void main() |
| 方法职责 | 连接JMS服务器 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | sendMessage(Session, MessageProducer) |
| 方法职责 | 将注册消息发送到JMS服务器上 |
| 方法前置条件 | 已连接JMS服务器 |
| 方法后置条件 | JMS服务器的相应队列持有该注册消息 |

表5.7 Listener类

|  |  |
| --- | --- |
| 类名称 | Listener |
| 类职责 | 监听中心节点的任务消息 |
| 重要数据结构 | 无 |
| 类方法 | void main() |
| 方法职责 | 连接JMS服务器 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | void onMessage(Message) |
| 方法职责 | 实现MessageListener的onMessage()方法，监听中心节点的任务消息 |
| 方法前置条件 | 已连接JMS服务器 |
| 方法后置条件 | 中心节点发送任务消息到JMS服务器上后，此方法能够监听到并解析 |
| 类方法 | void operate(String message) |
| 方法职责 | 解析消息字符串，如果是注册确认消息则直接发送给中心节点返回消息；如果是任务消息则开始爬虫任务 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |

表5.8 RequestCrawler类

|  |  |
| --- | --- |
| 类名称 | RequestCrawler |
| 类职责 | 爬取数据 |
| 重要数据结构 | Timer timer, IPTask task |
| 类方法 | Arraylist<Item> selectFromTable(int page, String password) |
| 方法职责 | 从任务节点数据库中储存的item表中根据page爬取url列表 |
| 方法前置条件 | 服务器持有第一次迭代结果Item表 |
| 方法后置条件 | 根据页码查询得到要爬取的url列表 |
| 类方法 | Document getDocument(String href) |
| 方法职责 | 根据IPTask类的代理ip获取要爬取网站的document |
| 方法前置条件 | 无 |
| 方法后置条件 | 得到网页的document |
| 类方法 | void crawler(int page) |
| 方法职责 | 爬取数据，结束后调用Dao类存储数据，ReCrawler类重新爬取丢失的数据以及Sendback类发送返回消息 |
| 方法前置条件 | 无 |
| 方法后置条件 | 网页数据被保存 |
| 类方法 | void reCrawler() |
| 方法职责 | 爬取被保存的需要再次爬取的url |
| 方法前置条件 | 无 |
| 方法后置条件 | 网页数据被保存 |

表5.9 ReCrawler类

|  |  |
| --- | --- |
| 类名称 | ReCrawler |
| 类职责 | 重新爬取第一次执行时由于网络连接超时被暂时储存起来的url |
| 重要数据结构 | 无 |
| 类方法 | Arraylist<Item> selectFromTable(String password) |
| 方法职责 | 从任务节点数据库中储存的item表中根据page爬取url列表 |
| 方法前置条件 | 服务器持有第一次迭代结果Item表 |
| 方法后置条件 | 查询得到要爬取的url列表 |
| 类方法 | void crawler(int page) |
| 方法职责 | 爬取数据，结束后调用Dao类存储数据 |
| 方法前置条件 | 无 |
| 方法后置条件 | 网页数据被保存 |

表5.10 Sendback类

|  |  |
| --- | --- |
| 类名称 | Sendback |
| 类职责 | 发送返回消息 |
| 重要数据结构 | 无 |
| 类方法 | void sendback() |
| 方法职责 | 连接JMS服务器 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | sendMessage(Session, MessageProducer) |
| 方法职责 | 将返回消息发送到JMS服务器上 |
| 方法前置条件 | 已连接JMS服务器 |
| 方法后置条件 | 无 |

表5.11 IPTask类

|  |  |
| --- | --- |
| 类名称 | IPTask |
| 类职责 | 每隔30秒更换爬虫代理IP，生成代理Proxy |
| 重要数据结构 | boolean useProxy, Proxy proxy |
| 类方法 | void run() |
| 方法职责 | 持续更换IP |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | void useProxy() |
| 方法职责 | 使用代理节点 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | Proxy getProxy() |
| 方法职责 | 返回Proxy |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |

表5.12 Dao类

|  |  |
| --- | --- |
| 类名称 | Dao |
| 类职责 | 实现系统需要的数据库操作 |
| 重要数据结构 | 无 |
| 类方法 | void saveRequest(Request r, String password) |
| 方法职责 | 储存数据 |
| 方法前置条件 | 已连接数据库 |
| 方法后置条件 | 普通数据被保存 |
| 类方法 | void reCrawler(Request r, String password) |
| 方法职责 | 将由于网络连接超时而未能爬去的网页放入临时表中 |
| 方法前置条件 | 已连接数据库 |
| 方法后置条件 | 需要再次爬取的数据被保存 |
| 类方法 | ResultSet query(String sql, String password) |
| 方法职责 | 从Item表中查询要爬取的url列表 |
| 方法前置条件 | 已连接数据库 |
| 方法后置条件 | 无 |

表5.13 JDBC类

|  |  |
| --- | --- |
| 类名称 | JDBC |
| 类职责 | 连接数据库 |
| 重要数据结构 | String driver, String url, String user |
| 类方法 | Connection connect(String password) |
| 方法职责 | 连接数据库 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |
| 类方法 | ResultSet query(Connection con, String sql) |
| 方法职责 | 提供基本sql查询功能 |
| 方法前置条件 | 无 |
| 方法后置条件 | 无 |

## 5.2 模块实现

### 5.2.1 关键部分代码

|  |
| --- |
| public void connect(String ip, int i) {  //连接工厂  ConnectionFactory connectionFactory;  //连接  Connection connection = null;  //会话 接受或者发送消息的线程  Session session;  //消息的目的地  Destination destination;  //消息生产者  MessageProducer messageProducer;  //实例化连接工厂  connectionFactory = new ActiveMQConnectionFactory(USERNAME, PASSWORD, BROKEURL);  try {  //通过连接工厂获取连接  connection = connectionFactory.createConnection();  //启动连接  connection.start();  //创建session  session = connection.createSession(true, Session.AUTO\_ACKNOWLEDGE);  //创建一个以ip为名称的消息队列  destination = session.createQueue(ip);  //创建消息生产者  messageProducer = session.createProducer(destination);  //设置为不持久化  messageProducer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);  //发送消息  sendMessage(session, messageProducer, i);  session.commit();  } catch (Exception e) {  e.printStackTrace();  }finally{  if(connection != null){  try {  //关闭连接  connection.close();  } catch (JMSException e) {  e.printStackTrace();  }  }  }  } |

图5.3 连接JMS服务器代码

|  |
| --- |
| public void crawler(int begin, String password, server server){  timer.schedule(task, 0, 30000);    ArrayList<Item> haveTickets = selectFromTable(begin, password);    for(int k=0;k<haveTickets.size();k++){ //这几页所有要爬的项目  Item i = haveTickets.get(k);  int tickets = i.getTickets();    for(int m=1;m<=tickets;m++){ //每个项目的tickets  String requestUrl = Utility.requestUrl(i) + m + "/";    Request request = new Request();  request.setName(i.getName());  request.setPage(i.getPage());  request.setNum(m);    try {  Document document = getDocument(requestUrl);    //summary  Element esm = document.select("h2[class=dark title]").first();  if (esm!=null) {  String summary = esm.text();  System.out.println(summary);  request.setSummary(summary);  }    //content  String content = "";  Element ectdiv = document.select ("div[class=markdown\_content]"). first();  if (ectdiv!=null) {  Elements ectps = ectdiv.children();  Iterator<Element> ectpsIter = ectps.iterator();  while(ectpsIter.hasNext()) {  Element ectp = ectpsIter.next();  content = content + ectp.text() + "\n";  }  request.setContent(content);  }  dao.saveRequest(request, password);    } catch (Exception e) {  e.printStackTrace();  dao.reRequest(request, password);  }  } //一个项目的tickets爬完  } //所有项目爬完  timer.cancel(); //关闭计时器，不再切换代理ip  ReCrawler reCrawler = new ReCrawler();  reCrawler.crawler(begin, password, server); //爬取由于网络连接超时而临时储存起来的url  server.sendback(); //给中心节点返回消息  } //方法结束 |

图5.4 爬虫代码

### 5.2.2 运行截图

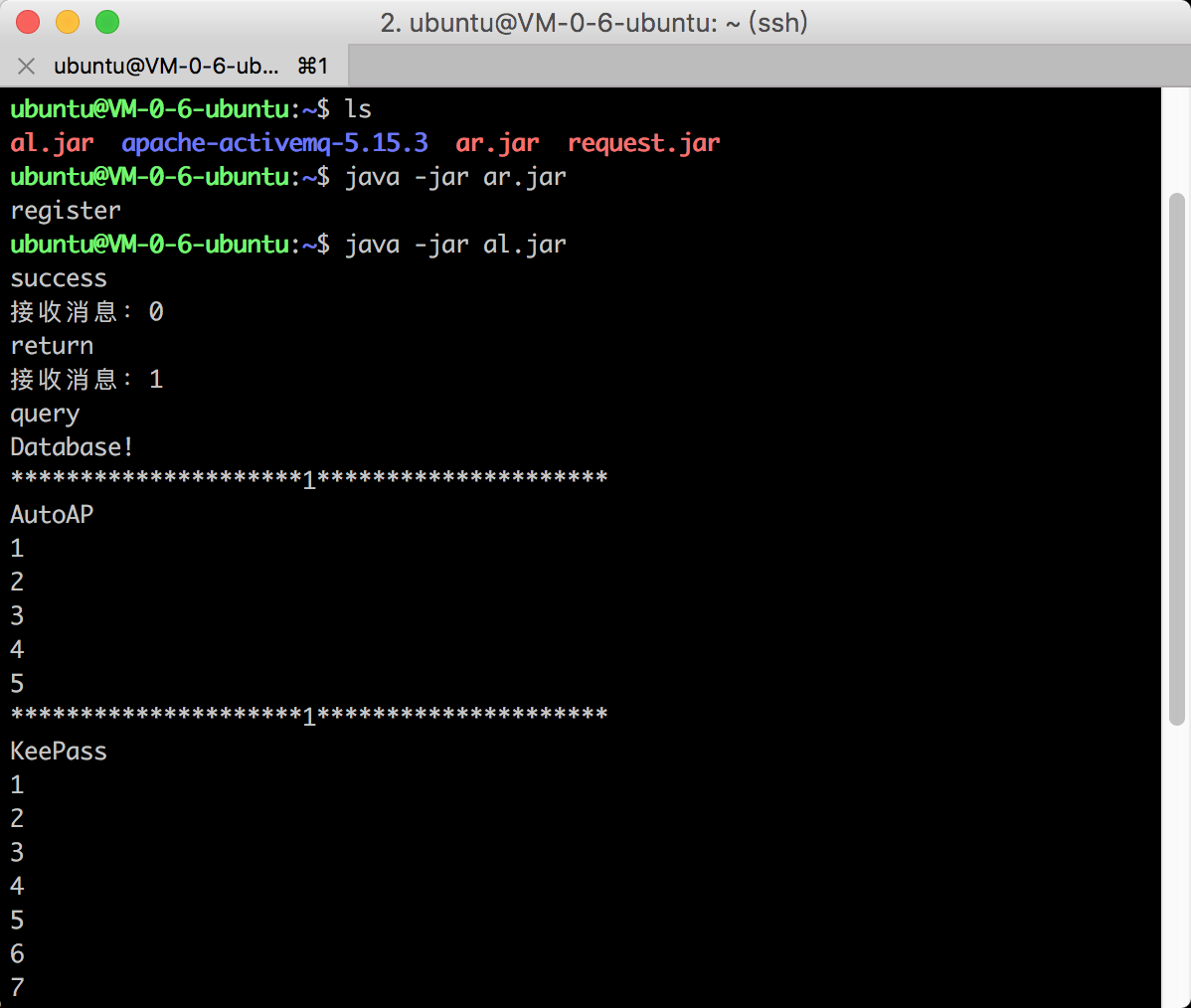


图5.5 服务器1爬虫过程

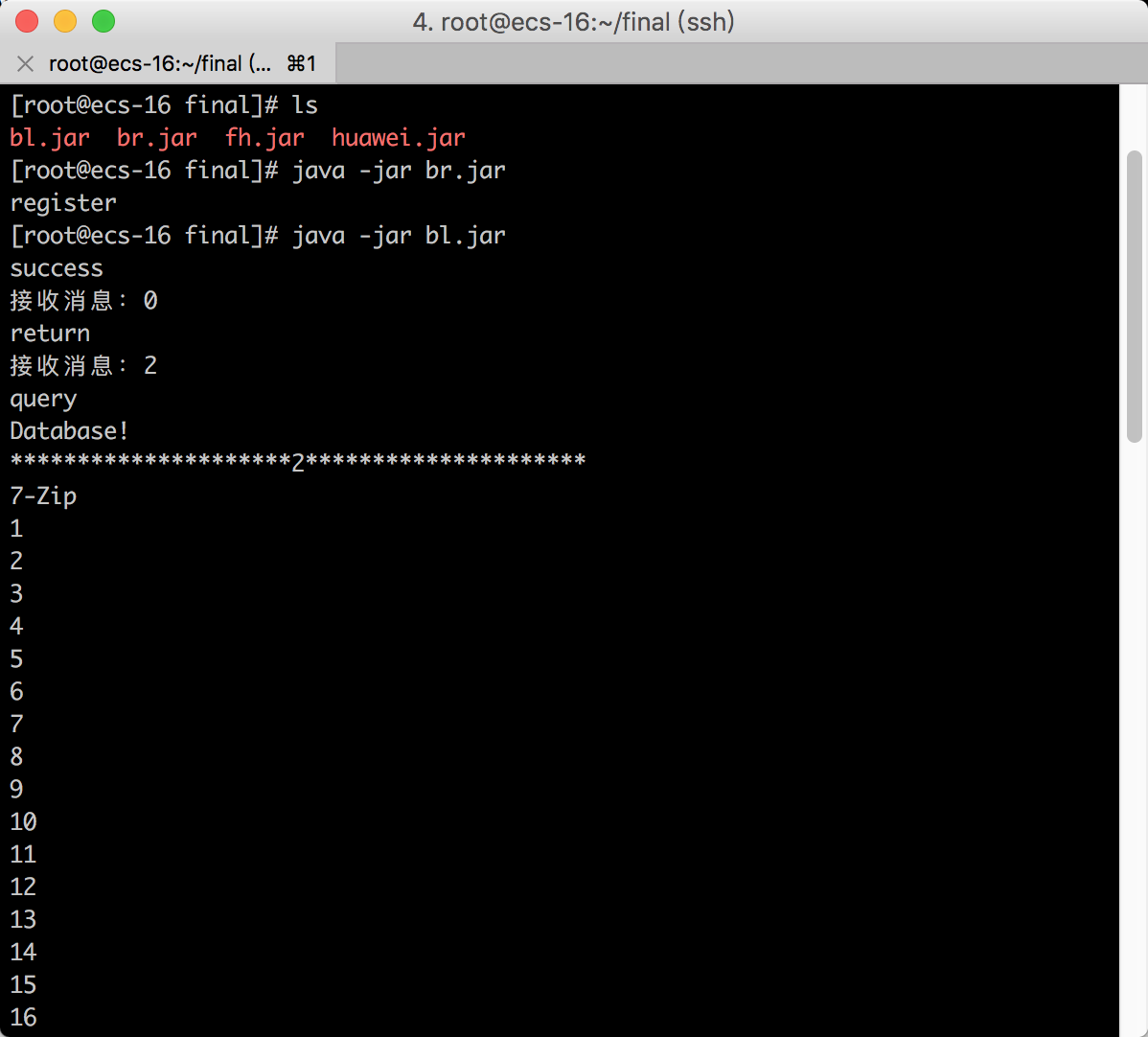


图5.6 服务器2爬虫过程

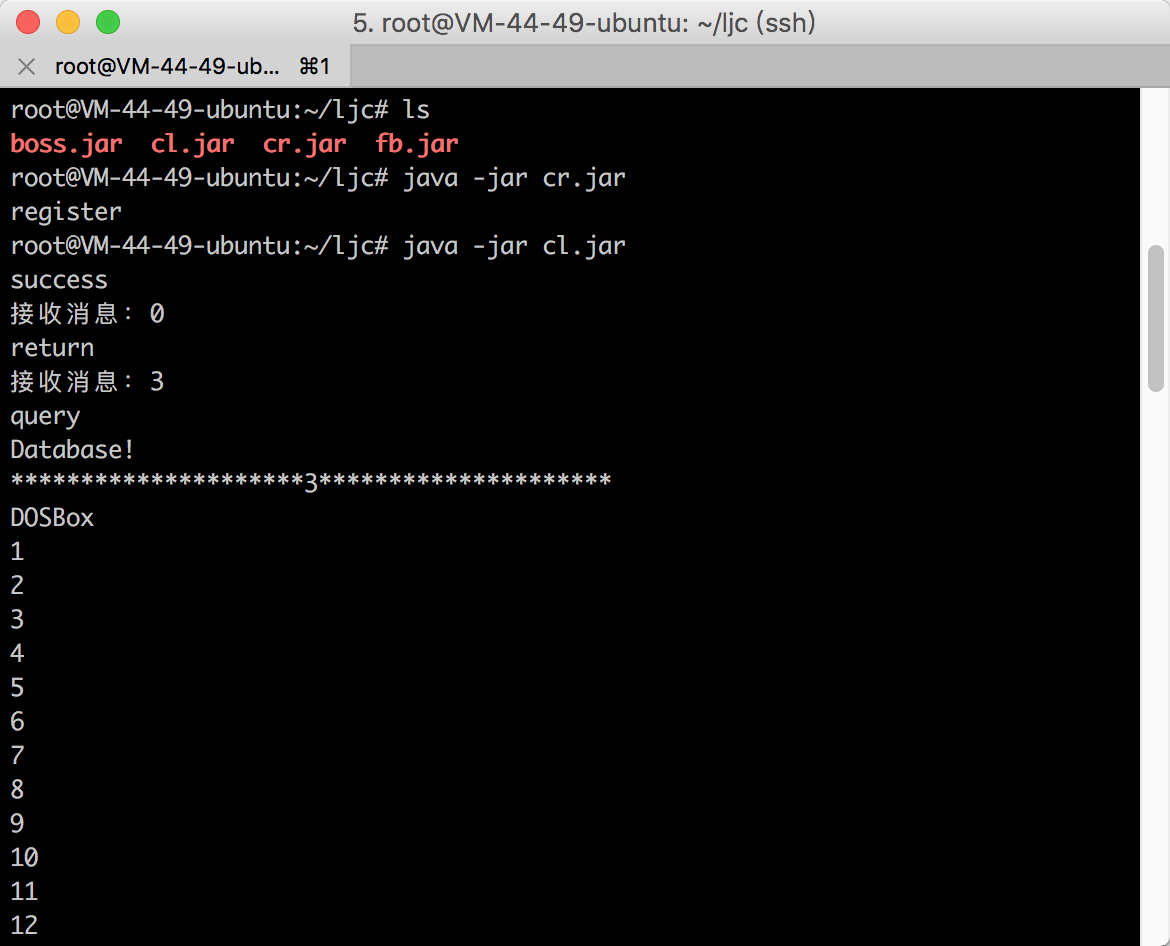


图5.7 服务器3爬虫过程

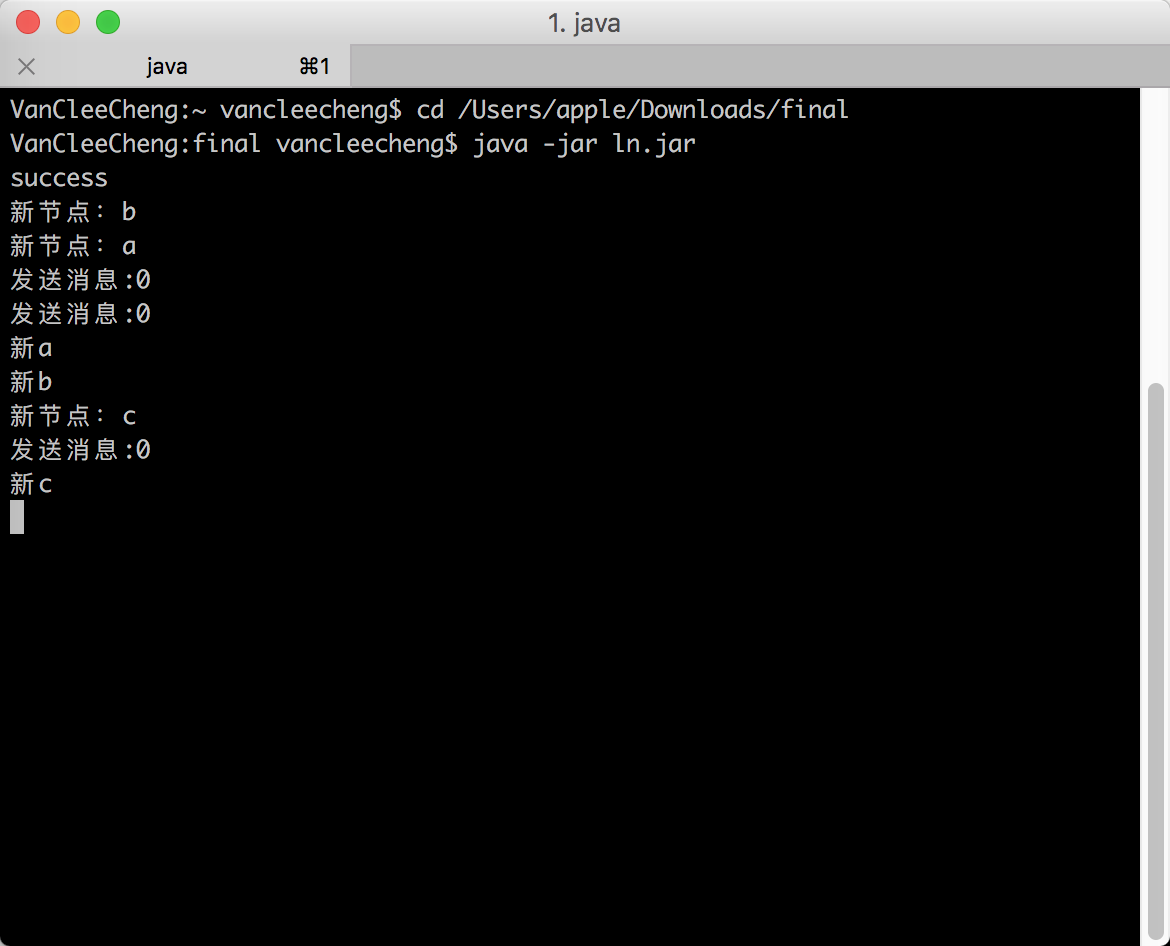


图5.8 中心节点对新注册节点的监听

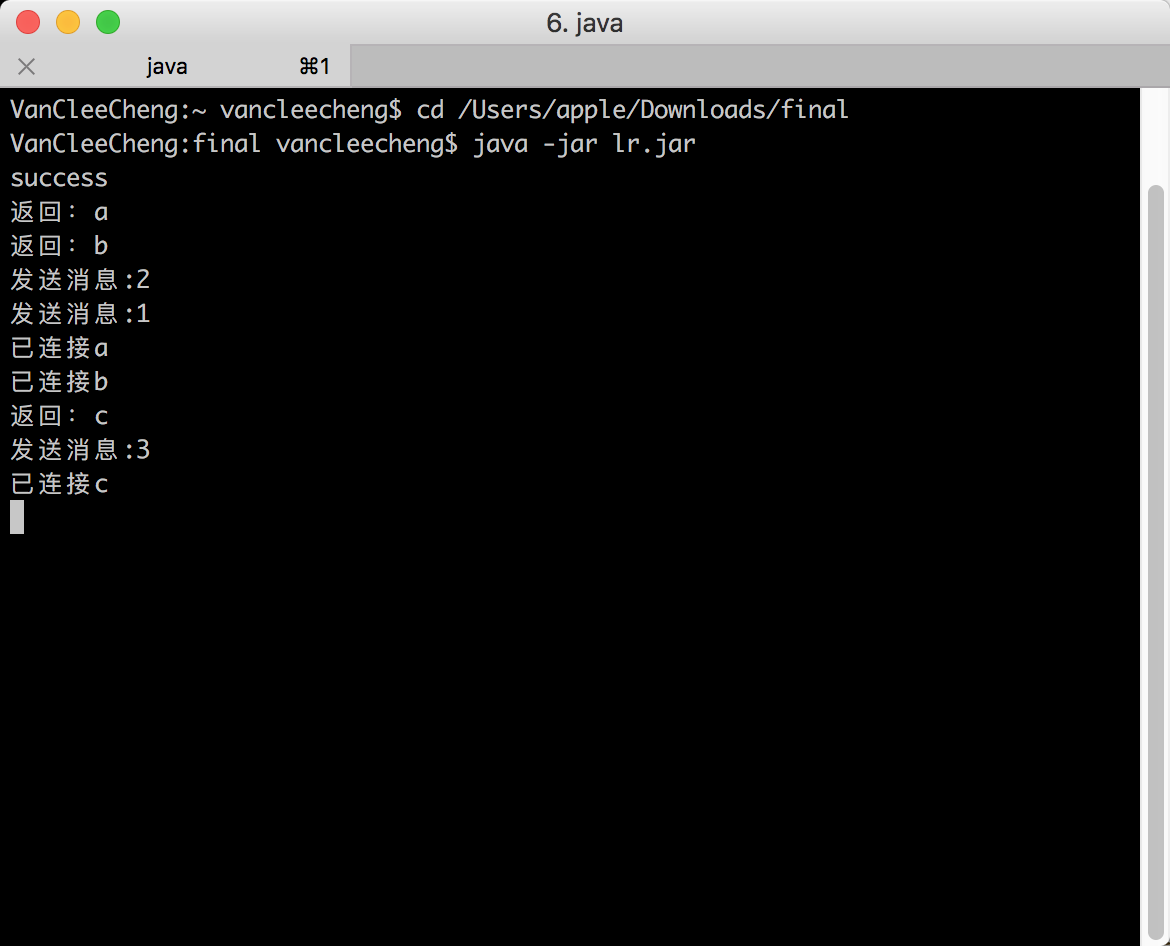


图5.9 中心节点对任务节点返回消息的监听

## 5.3 数据库设计

服务端的数据库主要设置了4个实体：

项目：记录项目的名字，页码，url和需求个数。

详情：记录项目的详细信息。

推荐：记录项目所关联的推荐项目。

需求：记录项目中用户所提的需求。

ER图如图5.10所示。

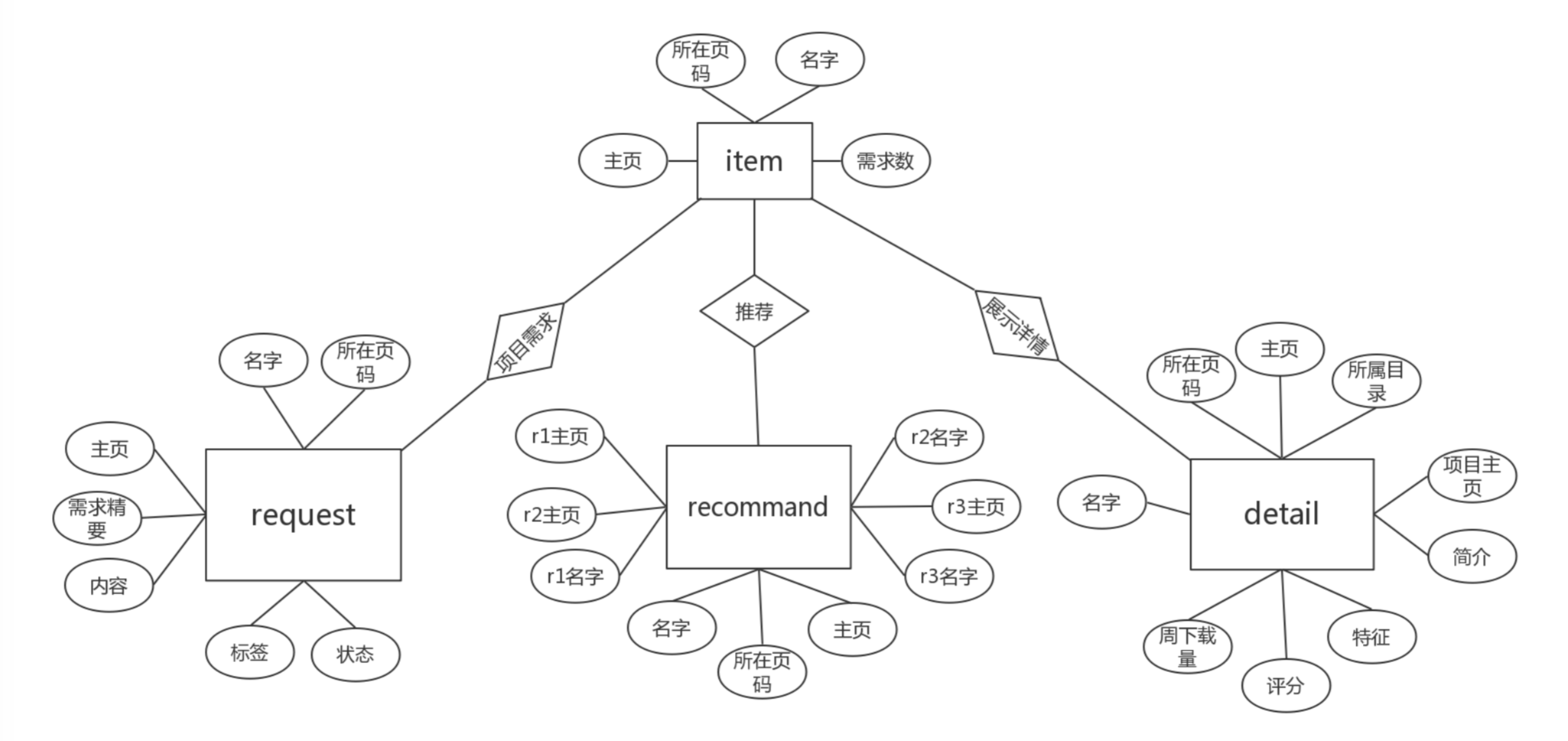


图5.10 ER图

第六章 总结与展望

## 6.1 总结

本文基于SourceForge.net的数据分布设计了一个分布式爬虫系统，能够做到中心节点分配爬虫任务，任务节点执行爬虫程序并存储数据。项目使用Java开发，利用JMS技术实现核心功能。项目的诞生经过了如下几个过程：

1. 确定了对SourceForge.net中开源项目进行精准定位和需求复用，定位了需要爬取的数据所在的网页以及在网页中的位置，为爬虫做准备。

2. 做需求分析，明确了项目的功能以及性能需求。同时确定系统的架构和爬取策略：采用主从式架构和广度优先爬取策略。

3. 最初项目设计使用RMI作为网络通信工具，完成了系统大部分代码后发现RMI虽然可以直接调用服务器程序，但是客户端和服务器的高耦合度严重影响了修改与扩展，需求的少许改动就会同时影响客户端和服务器的代码。而且RMI的机制限定了有多少个服务器客户端就要有多少个线程，当服务器个数增加时，客户端的线程个数必会成为限制系统效率的瓶颈。所以最后选择了更适合分布式系统的JMS作为通信工具。

4. 项目采用节点注册的方式是一大亮点。抛弃了传统的中心节点持有任务节点地址的方式，节点的注册与发现大大提高了系统的可扩展性。

## 6.2 展望

本分布式爬虫系统完成了对SourceForge.net进行数据爬取的目标，但距离成为一个能协调近百台服务器的大型分布式系统还有很大的距离。项目存在以下这些待完善的地方：

1. 数据库只有一台专门的数据库服务器和任务节点的本地储存，一般的数据库都只能支持几百的连接数，而web应用的并发请求能够轻松达到几千。为了尽量减少对数据库的连接和访问，需要设计一些缓冲系统。

2. 系统要根据流量动态调整服务器的数量，流量低谷的时候要自动缩减服务器，高峰时要自动增加。

3. 系统缺少对节点监控的Tracker，容错性不够高。

参考文献

[1] 作者,译者,书名(版本),出版地:出版社,出版时间,引用部分起止页.

[2] 作者,译者,文章题目,期刊名,年份,卷号(期数):引用部分起止页.

[3] 作者,学位论文名,本科/硕士/博士论文,大学/机构名,年份.

[4] 网页的主题,URL.

1. 刘兆存, 范玮佳. 软件过程中可复用需求分析[J]. 重庆理工大学学报(自然科学), 2012, 26(1):53-60.
2. 什么是分布式系统，如何学习分布式系统, https://www.cnblogs.com/xybaby/p/7787034.html
3. Nutch简介及安装，https://www.cnblogs.com/hehaiyang/p/4495820.html
4. 吕阳，分布式网络爬虫系统的设计与实现，硕士论文，电子科技大学，2013
5. 张笑天，分布式爬虫应用中布隆过滤器的研究，硕士论文，沈阳工业大学，2017
6. MapReduce简介和入门, https://www.yiibai.com/hadoop/intro-mapreduce.html
7. 李婷，分布式爬虫任务调度与AJAX页面抓取研究，硕士论文，电子科技大学，2015
8. 百度百科--远程过程调用协议， https://baike.baidu.com/item/远程过程调用协议
9. 百度百科--Web Service，https://baike.baidu.com/item/Web%20Service
10. WebService工作原理及实例，https://blog.csdn.net/yangwenxue\_admin/article/details/51059125
11. Java RMI详解，https://blog.csdn.net/a19881029/article/details/9465663
12. 百度百科--JMS，https://baike.baidu.com/item/JMS/2836691
13. JMS深入浅出，https://blog.csdn.net/jiuqiyuliang/article/details/46701559
14. 百度百科--activeMQ，https://baike.baidu.com/item/ActiveMQ
15. JAVA线程队列BlockingQueue， https://www.cnblogs.com/SpeakSoftlyLove/p/5605212.html
16. java自带线程池和队列详细讲解, https://blog.csdn.net/kouwoo/article/details/48788867
17. 分布式系统的特点以及设计理念， https://blog.csdn.net/huaweitman/article/details/50526562
18. 可扩展Web架构与分布式系统, https://www.cnblogs.com/likehua/p/3796802.html