计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(6):376−380 [doi: 10.15888/j.cnki.csa.008501]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

线程池与消息中间件技术在疾控数据交换中的应用①

刘　潇

(江苏省疾病预防控制中心 公共卫生信息所, 南京 210009)
通信作者: 刘　潇, E-mail: liuxiao@jscdc.cn

摘　要: 为了提高疾控中心进行大规模数据交换的处理效率, 基于线程池与消息中间件技术设计并实现了一个数据交换处理模型, 该模型在调用数据接口并记录接口反馈信息的基础上自动构造数据交换任务, 使用消息中间件快速保存反馈信息, 使用线程池并通过合理设置线程池的参数实现了数据交换任务的并发控制. 仿真实验结果表明, 相对于传统的数据交换处理方式, 该模型在保证数据交换稳定性的同时极大提高了数据交换任务的处理效率.

关键词: 数据交换; 线程池; 消息中间件; 并发控制

引用格式: 刘潇.线程池与消息中间件技术在疾控数据交换中的应用.计算机系统应用,2022,31(6):376–380. http://www.c-s-a.org.cn/1003-3254/8501.html

Application of Thread Pool and Message-oriented Middleware Technology in Data Exchange of CDC

LIU Xiao

(Department of Public Health Information, Jiangsu Provincial Center for Disease Control and Prevention, Nanjing 210009, China)

Abstract: For the processing efficiency of large-scale data exchange in Centers for Disease Control and Prevention (CDC), a processing model of data exchange was designed and implemented based on thread pool and message-oriented middleware technology. The model automatically created data exchange tasks on the basis of calling data interfaces and recording the feedback information of interface. The message-oriented middleware was used to save the feedback information quickly and the thread pool with reasonable parameters was used to realize the concurrency control of data exchange tasks. Simulation results show that, compared with the traditional data exchange method, the model can greatly improve the processing efficiency while ensuring the stability.

Key words: data exchange; thread pool; message-oriented middleware; concurrency control

随着疾控信息化工作的不断深入, 疾控的传染病、公共卫生突发事件、慢病、计划免疫以及精神卫生等业务条线的信息系统在不断地建立与完善, 疾控信息化标准体系[1,2] 的建立与完善有力地推动了全民健康信息化中公共卫生的数据整合. 在当前各行业协作日益紧密、各级疾控一体化集成日渐成熟的大背景下, 疾控中心各类的数据共享与交换[3,4] 需求也随之而来. 根据不同的业务需求, 各个信息系统需要调用不同来源的接口来完成数据的下载、上传或核验等操作. 在数据量比较小、任务实时性要求比较低的情况下, 全量数据逐条调用数据接口并记录接口反馈信息的模式可以满足业务需求, 但是当数据量比较大并且任务实时性要求比较高的情况下, 比如: 疫情期间, 全省亿级数量的常住人口库的全量数据需要周期性调用通信管理接口或核酸检测查询接口以获得个人行程记录与核酸检测的相关信息, 或是在特定的时间内, 某个月增

① 收稿时间: 2021-08-29; 修改时间: 2021-09-26; 采用时间: 2021-09-29; csa 在线出版时间: 2022-05-26

百万级随访数据的业务系统的大量的随访信息需要全部上传至指定的平台, 逐条调用数据接口的模式效率太低, 无法在规定的时间内完成任务, 如何利用有限的硬件资源高效地完成数据交换任务成为了疾控在信息化建设中面临的一个问题.

在有限的硬件资源下, 解决这个问题的思路是让数据交换任务并发执行, 直接在服务器上为每一个数据交换任务分配一个线程并同时启动大量线程去完成数据交换的方法会导致服务器压力过大, 线程的运行缺乏有效的控制, 线程的创建与销毁都会造成系统开销, 操作系统对大量线程的频繁的切换与调度会给 CPU 带来沉重的负担, 容易造成服务卡顿或服务器宕机. 本文基于线程池与消息中间件技术建立一个数据交换的并发处理模型, 使用 Java 线程池去控制数据交换任务的并发处理, 并引用消息中间件 Kafka 作为中间件来记录数据交换结果, 进一步提高任务完成的效率, 通过实验的对比证明该模型的可行性与高效性.

1 技术简介

1.1 线程池介绍

线程池技术是一种设计程序并发运行的技术, 其核心思想是对已有线程的复用来避免大量线程创建与销毁带来的系统开销, 在 CPU 上创建和结束线程造成的开销是创建或销毁任务的 18 至 100 倍[5], 而且通过任务进行同步的开销也远低于同步多个线程的开销, 因此线程池技术能够更好地支持细粒度的任务并发[6]. 常见的线程池一般主要包括 4 个部分: 线程管理器、工作线程、任务接口和输入输出任务队列, 在启动时线程池创建若干数量的空闲线程, 当任务到达时利用已经创建的线程执行任务, 任务处理完成后, 该线程会被线程池回收用来执行下一个任务以达到线程复用的效果, 同时线程池还要对任务队列的大小、空闲线程的销毁、新线程的创建以及对任务的拒绝策略等进行管理.

Java 从 JDK 1.5 版本开始在 java.util.concurrent 包中提供了对线程池功能的支持[7], 相关类的继承关系如图 1 所示, 其中 ThreadPoolExecutor 是最核心的一个类, Java 通过封装 ThreadPoolExecutor 类提供了 SingleThreadExecutor、CachedThreadPool、FixedThreadPool 以及 ScheduledThreadPool 这 4 类适合特定场景的线程池供编程人员调用, 同时 Java 也支持编程人员重写 ThreadPoolExecutor 的构造方法, 通过设置构造参数自定义线程池.

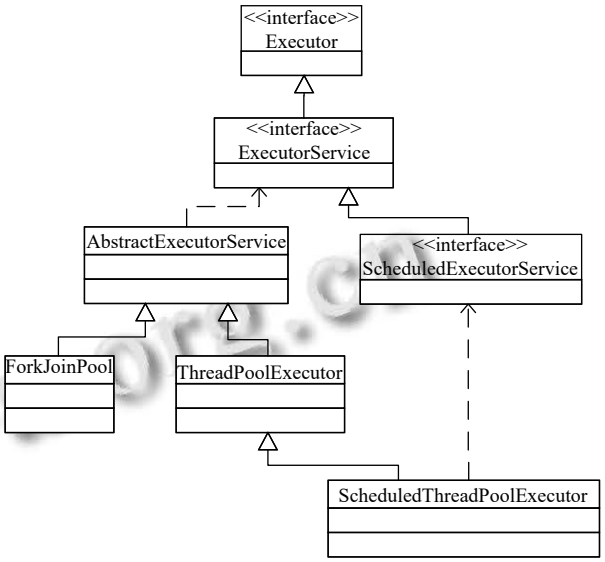

图 1 Java 线程池 UML 静态类图

ThreadPoolExecutor 类构造方法的主要的构造参数如下:

corePoolSize: 核心线程数, 即常驻线程池的工作线程数量.

maximumPoolSize: 最大线程数, 即某一时刻, 当任务大于线程池当前存在的工作线程数时, 线程池中的工作线程可以增加到的最大值.

keepAliveTime: 当线程数大于核心线程数时, 空闲的工作线程等待新任务的最长时间, 超过这个时间空闲线程没有接到任务就会被销毁, 线程池只保留核心线程数的工作线程数量.

workQueue: 任务队列, 即线程池中的工作线程的数量已经达到最大线程数时, 任务的等待队列.

threadFactory: 线程工厂, 可以用来自定义线程池中线程的命名方式, 优先级等属性.

Handler: 拒绝策略, 即线程池中的工作线程的数量已经达到最大线程数且任务队列已满的情况下, 线程池对超出线程池处理能力的任务所做的处理策略.

1.2 消息中间件介绍

消息中间件是可以在不同系统之间进行消息传递的一类组件, 它利用高效、可靠的消息传递机制进行平台无关的数据交流[8], 消息生产者定向发送数据, 消息消费者获取并消费数据, 基于数据通信进行分布式

系统的集成. 消息中间件的消息传递主要有两种模式, 分别是点对点模式和发布-订阅模式. 目前比较主流的分布式消息中间件有 Kafka, RabbitMQ, ActiveMQ 等.

Kafka 是一个分布式的消息发布-订阅模式[9] 的中间件系统. Kafka 在主题中保存消息的信息, 生产者向主题写入数据, 消费者从主题读取数据, 从而实现数据传输.

高性能、高吞吐、低延时是 Kafka 的显著的特性, 虽然 Kafka 的消息保存在磁盘上, 但是由于采用了顺序写入、MMFiles (memory mapped files)、Zero Copy、批量压缩等技术优化了读写性能[10], 使其可以突破传统的数据库、消息队列等数据引擎所受限的磁盘 IO 瓶颈, 即使是部署在普通的单机服务器上, Kafka 也能轻松支持每秒百万级的写入请求[11], 读写速度超过大部分的消息中间件, 这种特性使得 Kafka 在海量数据场景中应用广泛.

2 模型设计与实现

2.1 模型设计

疾控信息化工作中处理数据交换的基本流程是: 从数据库中分批取出需要调用数据接口的数据, 为批次中的每一条数据创建一个数据交换任务, 任务主要包括调用接口获得反馈信息、将反馈信息回写数据库进行持久化两个步骤.

由于各数据交换任务相互之间的无关性, 可以在调用的数据接口可承载的并发调用范围内, 使数据交换任务并发进行以提高效率, 并在数据交换任务的反馈信息持久化阶段将反馈信息写入吞吐量更高的消息中间件进行存储, 进一步缩短数据交换任务的运行时间以提高效率.

在图 2 中, 通过一个数据交换调度控制程序建立并初始化数据交换任务的线程池, 在进行数据交换任务时, 为从数据库取出的批量数据构造数据交换任务, 并将任务交给线程池进行并发处理的调度, 数据接口的反馈信息写入中间件进行保存, 不同的数据消费者进程可以异步消费消息中间以获取反馈信息, 按照不同的业务需求进行日志信息持久化到数据库或者实时进行交换日志的统计与分析等操作.

2.2 技术实现

数据交换调度控制程序用 Java 设计, 使用 Java 线程池与 Kafka 对模型进行实现, 模型实现主要包含数据交换任务构造、Kafka 调用以及数据交换线程池 3 个部分.

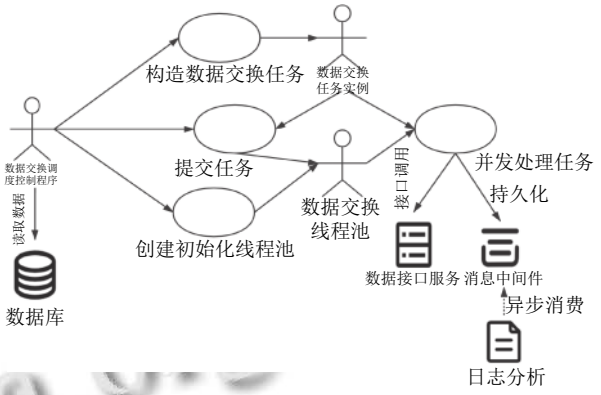

图 2 数据交换并发处理模型

2.2.1 数据交换任务构造

封装数据交换任务的类需要实现 Runnable 接口以保证其可以在实例化后被线程池工作线程所调用, 在该类的构造器中传递具体的 Kafka 连接以及数据接口调用所需要的参数, 并实现 Runnable 接口的 run 方法完成具体数据接口调用与反馈信息的记录, 其核心代码如下:

```
public class DSTask implements Runnable{ //数据交换任务封装类
public DSTask(KafkaProducer<Integer, String> producer, String id, ....)
{ //通过构造器为实例的属性赋值
this.producer=producer;   //Kafka 连接对象
this.id=id;   //数据在数据库的主键号
....   //数据接口调用的各项参数
}
@Override
public void run() { //实现 Runnable 接口的 run 方法
sendData(); //该方法获取构造器传递的属性完成数据交换任务
}
```

2.2.2 Kafka 调用

在数据交换任务封装类的 sendData 方法中调用 Kafka api 提供的 send 方法记录反馈信息, String 类型 topicName 为 Kafka 的相关主题名, String 类型 context 为数据交换任务最终按约定格式拼接好的反馈信息, 其核心代码如下:

```
producer.send(new ProducerRecord<Integer, String>(topicName, context));
```

2.2.3 数据交换线程池

通过参数设置自定义 ThreadPoolExecutor 类实例化线程池来控制数据交换任务并发处理. 由于数据交

换任务需要连续稳定的处理, 线程池的核心线程数和最大线程数设为相同值, 即线程池中的常驻的工作线程数, 这个值的大小在运行前需要由用户综合考虑所调用数据接口能承载的并发访问量, 以及当前任务所运行的服务器的 CPU 核数来设定, 在数据接口并发访问的承载范围内, 在实际工程应用中一般遵循如式 (1) 所示[12]:

$$N_s = N_i \times N_j \times \left(1 + \frac{T}{C}\right) \tag{1}$$

其中, N_s为工作线程数, N_i 为 CPU 核心数, N_j为预期的 CPU 利用率, $\frac{T}{C}$ 为程序 IO 等待时间与利用 CPU 计算时间比. 可见: 程序的 IO 等待时间所占比例越高, 需要越多的工作线程, 线程 CPU 计算时间所占比例越高, 工作线程数的设置越趋近于 CPU 核数.

线程池的任务队列的大小设置为每批要调用数据接口的数据的数量, 以保证所有的数据交换任务都会被任务队列容纳, 等待线程池的有效调度, 这样可以直接使用线程池默认的拒绝策略, 不需要再设计拒绝策略去处理线程池无法处理的数据交换任务.

线程池核心代码如下:

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(threadNum,
threadNum, 10, TimeUnit.SECONDS,
new LinkedBlockingQueue<Runnable>(queueCapacity),
Executors.defaultThreadFactory(),
new ThreadPoolExecutor.DiscardPolicy()); //实例化线程池, int 类型
threadNum 为工作线程数, int 类型 queueCapacity 为队列容量, 使用
默认的拒绝策略 DiscardPolicy
while(rs.next()){ //遍历从数据库取出的一批次的数据
String id = rs.getString("id"); //获取相关参数
....
executor.execute(new DSTask(producer, id, ....)); //实例化数据交换任
务并交给线程池调度执行
}
```

3 仿真实验

为测试该模型处理数据交换任务的效率, 在疾控内部局域网部署应用进行测试, 应用部署的服务器配置: 4 核 CPU, 内存 8 GB, 操作系统: 64 位 Linux CentOS 7.7, JDK 版本: Openjdk 1.8, 测试从疾控内网某业务库 (业务库版本: MySQL 8.0.18) 批量取出 5 000 条个人信息数据调用在公网发布的疫苗接种记录查询接口获取个人某疫苗首次接种记录的相关信息, 在逐条处理以及使用线程池模型进行处理、接口反馈的结果回写数据库或写入 Kafka 等一些不同的情况下, 分别进行如下仿真实验:

实验 1. 数据接口反馈信息回写数据库, 单线程逐条处理以及使用线程池在工作线程数取不同值的情况下的运行时间对比, 运行时间皆为 5 次实验的平均值, 数据如表 1 所示.

表 1 不同工作线程数运行时间对比

模式	工作线程数	运行时间 (ms)
单线程逐条处理	1	86 921
线程池	2	34 855
线程池	4	23 425
线程池	6	23 902
线程池	8	23 731

很显然, 线程池处理完成数据交换任务的效率明显优于单线程逐条处理, 且在实际接口的实际条件以及 4 核 CPU 的硬件资源条件下, 在工作线程数设为 4 时的运行效率已达到最佳.

实验 2. 在线程池在工作线程数取最佳值的情况下, 数据接口反馈信息回写数据库与写入 Kafka (版本: Kafka 2.5.0) 的运行时间对比, 运行时间皆为 5 次实验的平均值, 数据如表 2 所示.

表 2 反馈信息回写数据库与写入 Kafka 运行时间对比

反馈信息写入目标	运行时间 (ms)
MySQL	23 092
Kafka	9 905

对比两者的运行时间可以看出, 将数据接口反馈信息写入 Kafka 可以极大地提高了数据交换任务完成的效率.

4 模型应用

在疾控的数据交换工作中对模型进行实际应用时, 工程师根据需要进行数据交换任务的数据总量, 综合考虑部署数据交换应用程序的服务器内存情况, 对数据进行批次的划分, 确定每一批完成数据交换任务的数量与线程池任务队列的容量, 并根据服务器 CPU 的核数与需要调用的数据接口的实测情况确定线程池工作线程的数量, 设计数据交换调度控制程序. 如图 3 所示, 数据交换调度控制程序在初始化各类连接并建立线程池后, 按照预设的批次, 分批对数据进行数据交换任务的处理, 为了判断线程池是否已完成当前批次的所有数据交换任务, 可以设置一个线程安全的全局变

量, 每次数据交换任务完成时对这个变量进行累加操作, 数据交换调度控制程序通过读取这个变量值来获取线程池的当前状态, 如果当前批次的任务尚未全部完成, 调度控制程序执行自旋等待操作, 等待当前批次的任务全部完成, 线程池处于空闲状态后, 获取下一批次的数据继续进行, 直至所有批次的数据全部完成.

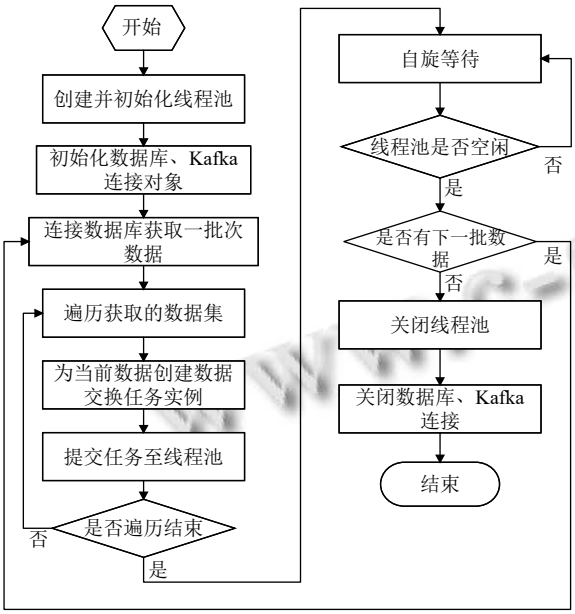

图 3 数据交换调度控制程序流程设计

图 4 展示的是实际工作中某重点人群库数据使用该模型调用新冠疫苗接种查询接口获取个人新冠疫苗第一针接种结果在 Kafka 相关主题中的存储情况, 该项数据交换任务按约定的格式记录了个人信息在业务库的主键号, 调用接口的匹配标识, 以及调用接口所获取的接种新冠疫苗第一针的疫苗厂商、接种时间、接种单位等信息, 各数据项之间插入制表符以便在信息消费时进行解析.

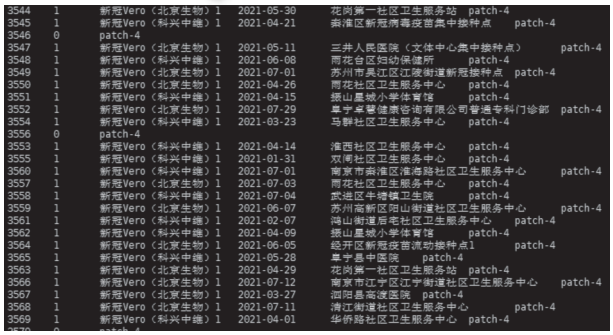

图 4 Kafka 记录的反馈信息展示

5 结束语

针对疾控中心在处理大规模数据交换时传统的处理模式效率不高, 难以及时完成任务的问题, 本文根据数据交换任务的特点设计了一个数据交换任务的并发处理模型, 并使用 Java 线程池与消息中间件 Kafka 给出了模型的具体实现. 该模型已成功应用在江苏省疾控中心的数据交换的处理中, 实践表明, 模型具有良好的数据交换任务并发控制与处理能力, 进行数据交换的数据量越大, 其优势越明显. 在不大幅度增加硬件成本的前提下, 该模型适用面广, 可用于各类型的数据换的处理与控制, 在保证服务稳定性的同时可以有效地提高数据交换的处理能力.

参考文献

1 马家奇, 赵自雄. 人群健康评价与疾病预防控制信息化体系构建. 中国卫生信息管理杂志, 2020, 17(4): 405–410, 460.
2 张诚, 道理, 毛丹, 等. 疾病预防控制数据标准体系建设与应用. 中国卫生信息管理杂志, 2020, 17(3): 300–304.
3 康国栋, 汪志国, 胡冉, 等. 基于云平台的江苏省预防接种综合服务管理信息系统建设与应用. 中国疫苗和免疫, 2020, 26(4): 450–454.
4 江涛, 李莉, 曹彦, 等. 基于公共卫生信息平台交换传染病数据的研究. 中国卫生信息管理杂志, 2017, 14(4): 598–602.
5 李涛, 董前琨, 张帅, 等. 基于线程池的 GPU 任务并行计算模式研究. 计算机学报, 2018, 41(10): 2175–2192. [doi: 10.11897/SP.J.1016.2018.02175]
6 Majumdar S, Jain I, Gawade A. Parallel quick sort using thread pool pattern. International Journal of Computer Applications, 2016, 136(7): 36–41. [doi: 10.5120/ijca2016908495]
7 葛萌, 于博, 欧阳宏基. 线程池技术在考试系统中的应用. 计算机系统应用, 2016, 25(4): 107–111.
8 吴璨, 王小宁, 肖海力, 等. 分布式消息系统研究综述. 计算机科学, 2019, 46(S1): 1–5, 34.
9 Eugster PT, Felber PA, Guerraoui R, *et al*. The many faces of publish/subscribe. ACM Computing Surveys, 2003, 35(2): 114–131. [doi: 10.1145/857076.857078]
10 牟大恩. Kafka 入门与实践. 北京: 人民邮电出版社, 2017.
11 朱幼普, 卢军. 基于 Kafka 的分布式能效管理平台的设计与实现. 计算机与数字工程, 2018, 46(12): 2620–2623. [doi: 10.3969/j.issn.1672-9722.2018.12.047]
12 顾振德, 刘子辰, 龙隆, 等. 基于 Netty 的 IoT 终端通信服务系统设计. 计算机应用与软件, 2019, 36(4): 135–139. [doi: 10.3969/j.issn.1000-386x.2019.04.021]