线程池技术在考试系统中的应用①

葛 萌[1], 于 博[2], 欧阳宏基[1]

(咸阳师范学院 信息工程学院, 咸阳 712000)
(河南建筑职业技术学院 信息工程系, 郑州 450064)

摘 要: 当较大规模客户端并发请求服务器端应用程序时, 传统的为每个请求创建线程的解决方法会导致服务器端性能的严重下降甚至死机. 通过分析 JDK 的 Executor 框架, 从工作原理、核心线程池对象、执行策略等方面详细描述了线程池模型, 应用到一个三层 C/S 架构的在线考试系统中, 给出了服务端的设计架构和实现代码. 通过仿真测试证明了线程池技术在解决较大并发访问方面的稳定性.

关键词: 线程池; Executor 框架; 在线考试系统

Application of Thread Pool Technology in Examination System

GE Meng[1], YU Bo[2], OUYANG Hong-Ji[1]

[1](Information Engineering College, Xianyang Normal University, Xianyang 721000, China)
[2](Information Engineering Department, Henan Technical College Of Construction, Zhenzhou 450064, China)

Abstract: When large-scale client requests for applications in server concurrently, traditional solutions create a thread for every request, which will lead to a serious decline in server performance and even crash. By analyzing the JDK's Executor framework, the paper described in detail thread pool model from the principle of work, core thread pool object, implementation strategy and other aspects, applied the thread pool to a online examination system of three-layer C/S architecture, and gave the server's design framework and implementation code. Through simulation tests, the stability of the thread pool technical in the process of solving large concurrent access is proved.

Key words: thread pool; Executor framework; online examination system

考试系统是高等院校数字化校园建设的一个重要内容, 通过代替传统的纸质考试以减少校园开支、提高成绩的客观性和公正性, 并且减轻教师的工作强度[1]. 目前已经有一些成功应用的考试系统, 例如全国计算机等级考试、软件资格(水平)考试系统等. 这类系统是基于 C/S 架构的网络通信系统, 学生使用客户端程序在一个较小规模(全国计算机等级考试要求每个考场的最大考生数为 99 人)的局域网内与唯一的服务器端程序通信. 当考生的并发量较小时, 服务器可采用为每个请求创建一个线程的方式来与考生端通信, 实现难度也不大. 但是当考生端有较大规模并发量时, 此方式将会极大的消耗服务器资源, 因为大量线程间频繁的切换会打乱系统固有的切换周期, 而且每个线程本身都会占用一定的内存, 因此会导致服务器端响应缓慢甚至死机的情况.

为了解决上述问题, 从硬件方面可以采取提高服务器内存、CPU 个数以及搭建服务器集群的方式来完成. 本文主要从软件设计方面通过引入线程池技术来解决. 分析了 JDK 的线程池模型、介绍了一个在线考试系统框架及主要功能, 并将线程池引入到服务器端的程序设计中, 详细描述了实现过程. 通过与传统线程-模型在高并发量的测试对比, 证明了线程池的可行性.

1 线程池技术

1.1 工作原理

线程池的核心思想是避免创建大量线程和对已有

① 基金项目:咸阳师范学院专项科研计划(2012XSYK070)
收稿时间:2015-07-29;收到修改稿时间:2015-10-08

线程的复用[2,3]. 通常情况下, 服务器端程序在启动时创建若干数量的线程对象并缓存起来, 此时它们处于空闲状态; 当客户端请求到来并被服务器接受后(否则将客户端请求放入任务队列), 从线程池中唤醒一个线程与该客户端进行通信, 结束通信后该线程对象会被线程池回收, 准备用它执行下一次任务. 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客户端请求的拒绝等进行管理, 这样可以更好的利用线程对象并对服务器进行有效的保护.

与传统多线程方式相比, 线程池具有以下两点优势[4]: ①减少了创建和销毁线程的次数, 每个工作线程都可以被重用, 能执行多个任务. ②可根据系统的承载能力方便地调整线程池中线程的数目, 防止消耗过量系统资源而导致崩溃.

1.2 JDK 线程池体系结构

由于线程池与业务是无关的, 设计一个线程池至少要考虑以下几个方面[5,6]: ①对任务进行描述的类, 包含线程池中线程执行的所有信息. ②可动态变化的、保存任务的队列. ③线程池管理器, 用来创建、销毁线程池, 提供对任务的调用与转发. ④处理任务的工作线程类. ⑤查询线程, 用来检测任务的完成情况. ⑥服务器无法接受请求时的拒绝策略. 同时还要考虑同步以及死锁风险等, 所以从头开发一个线程池是具有一定难度的. 为了简化使用线程池的难度, JDK 从 1.5 版本开始在 java.util.concurrent 包中提供了对线程池功能的支持, 相关核心接口、类的层次关系如图 1 所示.

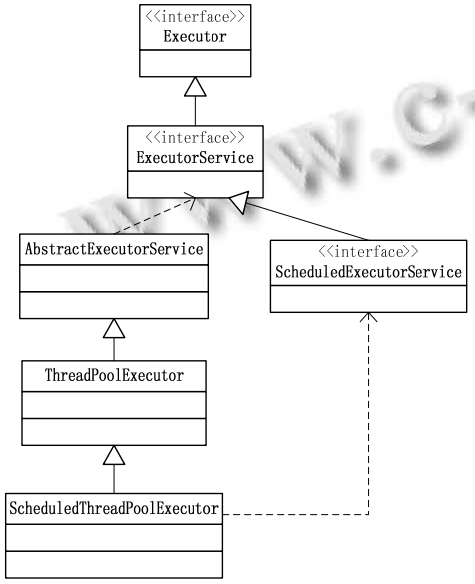

图 1　JDK 线程池的类图关系

Executor 接口包含唯一的 execute 方法, 通过 Runnable 来表示要执行的任务, 将任务的提交与执行操作进行解耦. ExecutorService 接口是真正意义上的线程池顶级接口, 其中定义了提交 FutureTask 任务的方法以及若干与线程池生命周期相关的方法. 例如: shutdown()方法用来关闭线程池, 以前提交的任务会被执行. shutdownNow()方法强制关闭线程池, 取消所有运行中的任务以及在工作队列中等待的任务, 同时将等待任务以 List 集合返回. awaitTermination 方法使得当前线程在给定的时间单位内等待给定的时间间隔, 若超时则返回线程池是否已经关闭. isTerminated()方法判断线程池是否已经处于关闭状态. 抽象类 AbstractExecutorService 主要实现了 ExecutorService FutureTask 相关的一些任务创建和提交的方法. ThreadPoolExecutor 是最核心的一个类, 用来表示线程池对象(详见 1.3 节描述). ScheduledThreadPoolExecutor 具有 ThreadPoolExecutor 的大部分功能, 可另行安排在给定的延迟后或者定期执行任务.

1.3 核心线程池对象

ThreadPoolExecutor 是线程池体系中的核心类, 用来创建和维护线程池对象. 该类包含的主要属性说明如下[7]:

corePoolSize: 在没有任务执行时, 线程池中所保留的线程数目.

maximumPoolsize: 线程池中同时工作的线程数的最大值.

keepAliveTime: 当线程数大于核心数时, 空闲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回收.

unit: keepAliveTime 参数的时间单位.

workers:正在被线程执行的任务集合.

workQueue: 等待被执行的任务队列, 任务实现 Runnable 接口, 由 Execute 方法调用执行.

threadFactory: 线程工厂, 用于在线程池需要新创建线程的时候创建线程.

defaultHandler: 当前池中线程数已达到最大值, 并且任务队列已满, 线程池中无法承载线程时的处理策略.

线程池的核心大小(corePoolSize)、最大值(MaximumPoolSize)和存活大小(keepAliveTime)共同负责线程的创建和销毁. 如果当前线程池处于运行状

态, 任务提交给线程池后的执行过程如图 2 所示. 通过调节线程池的基本大小和存活时间, 可以帮助线程池回收空闲线程占有的资源, 从而使这些资源可以用于执行其它工作.

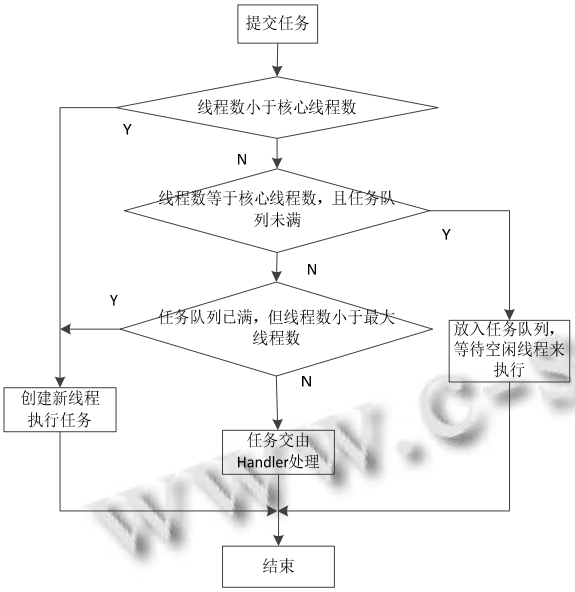

图 2 线程池中任务的执行流程

1.4 执行策略

执行策略是一种资源管理方式, 通过限制并发的数量来确保应用程序不会由于资源耗尽而崩溃. 执行策略[8]主要从执行任务的线程、任务的执行顺序、任务并发执行的个数、任务队列中等待执行的个数、对哪个任务进行拒绝并如何通知应用程序等方面来定义任务的执行, 从而确保应用程序具有较好的性能.

Executors 类通过工厂方法提供了四种执行策略来管理工作线程, 分别是: ①newSingleThreadPool: 确保所有任务按照提交的先后顺序(FIFO、LIFO、优先级)由唯一工作线程来执行, 该策略不算真正意义上的并发. ②newFixedThreadPool(Fix 类型): 随着任务的提交而逐个创建线程, 直到数量达到某个固定的最大值. ③newCachedThreadPool(Cached 类型): 线程池大小完全依赖于操作系统(或者 JVM)能够创建的最大线程数, 并且能根据当前任务的数量来调整线程池中线程的个数, 适合于执行生存周期较短的异步任务. ④newScheduledThreadPool: 创建一个大小无限的线程池. 此线程池支持定时以及周期性执行任务的需求. 以上策略中, 如果有线程因为异常而终止都会创建新的线程对象来弥补. 如果上述 4 种策略都无法满足需求, 可以通过创建 ThreadPoolExecutor 对象来定制策略.

2 线程池在系统中的应用

2.1 考试系统架构

在线考试系统是基于校园网的分布式三层 C/S 架构(如图 3 所示)[9], 分别为: 表示层、业务层和数据层. 表示层是应用的用户接口部分, 采用 Java Swing 来设计, 界面和功能通过角色来区分. 业务层是系统的核心内容, 由位于服务器端的业务逻辑来封装, 通过 RMI 向表示层提供服务. 数据层就是数据库管理系统, 为业务层和表示层提供数据服务, 服务器端和客户端均采用 MySQL 数据库.

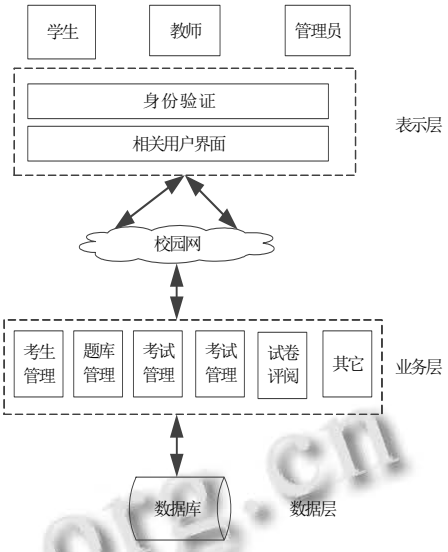

图 3 考试系统架构图

三层分布式处理方式使得学生端和服务器端各自运行本地和远程服务, 可以解决数据层与业务层的分离, 达到数据的安全保密. 由于考生端程序运行于本地, 只有在登录、抽取试题与提交试卷时才与服务器通信, 答题阶段都是访问本地的数据库, 这样可以提高数据读取效率、减轻对服务端的压力.

2.2 基于线程池的服务器端设计

由于考生只有在登录、抽取试题和提交试卷这三个操作涉及与服务器端交互, 考虑到考生人数的众多以及操作的同时性, 所以服务器端采用线程池技术. 所设计的服务器端的工作流程是: 1)初始化系统信息, 读取参加本次考试的学生总数, 该数值用于创建线程

池的任务队列. 2)创建线程池对象以及管理线程. 3)创建基于 TCP 协议的 ServerSocket 套接字, 在某一端口监听考生端的请求. 4)当有考生端请求到来时, 将请求加到任务队列, 从线程池中拿出空闲线程去接受请求. 5)管理线程用于监测线程池中线程的数量及时进行新线程的创建与销毁. 6)任务处理线程对考生端发送过来的信息进行分析, 根据登录、抽题和交卷操作分别进行处理.

2.3 服务器端的实现

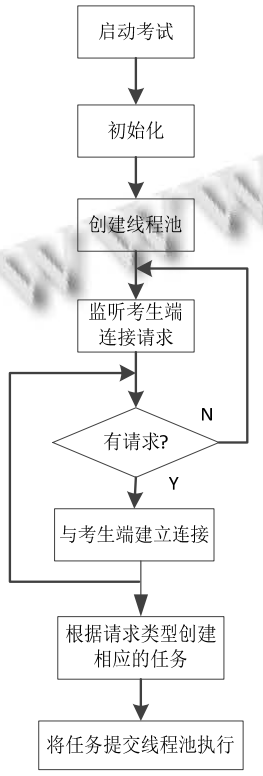

图 4　服务器端流程图

当服务器端启动考试后, 创建基于 TCP 协议的 ServerSocket 对象. 如果服务器成功接受考生端的请求便返回一个 Socket 对象(该 Socket 对象与某个考生端是一一对应的). 通过实现Runnable接口按匿名内部类方式创建任务对象, 并提交给线程池对象的execute()方法进行调用. 相关核心代码如下所示:

```
private final ExecutorService exec
public void start() {
    ServerSocket socket=new ServerSocket(9999);
    While(!exec.isShutdown()) {
        try{
         final Socket conn=socket.accept();
         exec.execute(new Runnable() {
            public void run() {
              handleRequest(conn); }
         });
        }catch(IOException e) {
         if(!exec.isShutdown())
         logger.error(LogUtil.getTrace(e)); }
        }
    }
    ……
    private void handleRequest(Socket cs)
    {   try {
     InputStream in=cs.getInputStream();
     BufferedReader br=new BufferedReader(new
InputStreamReader(in));
    String command=br.readLine();
    if("getPaper".equalsIgnoreCase(command))
    {   .将试题发送到考生端相关逻辑 }
     ……..
    }
```

handleRequest 为任务对象的核心方法, 通过封装 Socket 对象, 根据读取输入流中第一行字符串信息判断当前客户端的请求类型并执行相应的逻辑.

3　仿真测试

服务器可用内存 4G, 操作系统为 Windows Server 2008, CPU 个数是 4, JDK 版本为 1.6. 服务器端分别运行多线程、Fixed 类型线程池和 Cache 类型线程池模型的程序; 客户机通过测试程序来模拟一定数量的学生端向服务器发起抽取试题请求, 试题文件的容量大约在 160KB 左右, 统计三种情况下学生端读取试题的平均时间. 为了能够使服务器接受更多的客户端连接请求, 采用 ServerSocket(int port,int backlog)构造方法来创建服务器端套接字, 其中backlog表示允许接受客户端请求的最大连接数, 测试过程过不断改变此值, 并将该值与客户端并发数保持一致. 对于 Fixed 类型线程池, 根据文献[10]的研究最大线程数=CPU 数量*2+2, 以达到对 CPU 的最佳利用率. 测试结果如表 1 所示.

表 1 仿真测试

并发数量	多线程方式	线程池方式	
		Fix 类型	Cache 类型
200	1341MS	1098MS	1331MS
300	1890MS	1665MS	1598MS
500	3673MS	2817MS	3083MS
800	5972MS	4872MS	5515MS

从表 1 看出, 随着并发数量的增加, 在多线程方式下学生端抽到试题的平均时间逐渐增大, 过多的线程导致服务器的开销增大; 线程池方式下, 两种类型的线程池在同数量的请求下, 学生端抽到试题的平均时间都比多线程方式小, 这是因为线程池实现了线程的重用, 减少了线程创建、销毁的时间消耗. Fix 类型的线程池采用 LinkedBlockingQueue 作为存放任务的队列, keepAliveTime 值为 0 秒; Cached 类型的线程池采用 SynchronousQueue 作为存放任务的队列, keepAliveTime 值为 60 秒, 所以在同等并发量情况下, Fix 类型的线程池比 Cache 类型的线程池具有较高的效率. 测试结果表明采用线程池技术的本系统是可行的、高效的.

4 结语

与传统的为每个请求创建一个线程的方式相比, Executor 线程池执行框架实现了任务描述、提交与任务执行之间的解耦; 简化了对任务和线程的管理, 提高了系统效率. 下一步准备将本考试系统应用于更多科目的考试, 科目不同会导致试题的容量不同, 进而研究文件容量、网络传输速度、硬件参数与线程池的并发关系, 进一步优化系统的性能.

参考文献

1 杜静,何利娟,欧阳宏基.基于 JDBC 与 DAO 模式的在线考试系统设计与实现.咸阳师范学院学报,2014,29(2):26–29.

2 宋开卫,石坚,程哲.基于线程池的数据采集器在网管中的应用.计算机技术与发展,2007,17(7):210–212.

3 王华,马亮,顾明.线程池技术研究与应用.计算机应用研究,2005,22(11):141–142.

4 刘新强,曾兵义.用线程池解决服务器并发请求的方案设计.现代电子技术,2011,34(15):141–143.

5 闫保中,张波.线程池技术在车流监控系统中的应用.应用科技, 2011,38(10):18–22.

6 王文武,赵卫东,王志成,陈悦,韩下林.高性能服务器底层网络通信模块的设计方法.计算机工程,2009,35(3):103–105.

7 吴玉,刘美伶,祝建中.基于 Java Executor 的并发技术研究.计算机时代,2009,(2):8–10.

8 Goetz B, Peierls T, Bloch J. Java concurrency in practice. China Machine Press, 2014: 98–102.

9 田屏.基于 C/S 架构的考试系统设计[硕士学位论文].贵阳:贵州大学,2008.

10 顾永新.运用线程池技术实现数据库数据高速同步.信息系统工程,2013,(3):97–99,126.