Web 应用异步任务处理的实现研究

张文梅，廖福保
（广东农工商职业技术学院 机电系，广东 广州 510507）

摘 要：简述了异步任务处理在复杂 Web 应用中的必要性，利用 Java 中基于线程池的执行框架，分析并设计了相应的任务调度方法，以解决 Web 应用中大型任务处理时间长与系统要求响应时间短的矛盾，实现了用于处理复杂任务的异步调度，提高系统的可靠性。

关键词：异步任务；线程池；异步调度
中图分类号：TP311 **文献标识号**：A **文章编号**：1674-7720(2012)04-0014-03

Implementation of asynchronous tasks handling on Web application

Zhang Wenmei, Liao Fubao
(Guang Dong AIB Polytechnic College, Guangzhou 510507, China)

Abstract: The paper describes the necessity of asynchronous tasks handling on complicated web application. To solve the contradiction of the short system response time and the long task executing time on web application, the paper analyzes and designs a task scheduling method using java execution framework based on thread pool, implements asynchronous scheduling to handle complex tasks. The practice shows that it can improve the reliability of the system.

Key words: asynchronous task; thread pool; asynchronous scheduling

在 Web 应用中，某些功能的实现逻辑很复杂，执行比较耗时[1]，例如涉及外部系统调用、多数据源等；此时，希望可以让这些复杂的业务逻辑放在后台执行，而前台与用户的交互可以不用等待，从而提高用户体验；或者需要以一定时间间隔重复运行任务、或在每天的指定时间运行任务的情况。为此，需要控制大型任务对服务器资源的消耗，降低 Web 服务器的并发连接数目，这就需要将大型任务的提交和执行分开，使服务器接受任务后立即断开与客户端的连接，减少服务器的并发连接数，而任务则推迟到服务器资源许可时执行，以抑制服务器资源的峰值消耗。

为尽量减少耗时操作对执行的影响，本文提出了异步任务的处理，使用多线程来管理耗时任务，作为后台进程执行；同时把任务信息都持久化在数据库中，保证了异步任务处理的灵活性、可靠性。

1 多线程

1.1 线程池

一个线程是程序中的一条执行流，是操作系统分配处理器的基本单位。并发是程序中多条执行流的同时推进，多任务并发对应多线程并发[2]。

但是为每个任务创建一个线程，当任务完成时撤消对应的线程存在明显的缺陷。线程的创建需要一定的时间，给任务请求的响应带来延迟，线程的创建和撤消也给操作系统带来额外的管理负担，若频繁“创建和撤消”，则将明显增加系统的额外开销。为有效降低线程重复创建和撤销方面的开支可以采用线程池技术。

线程池技术提供了一种较好的解决方案[3]：系统维护由若干个线程组成的线程池。当有任务请求到达时，由池中的一个线程为之运行，在任务完成后不是将该线程撤消而是将其归还线程池，使之能够为后续到达的任务服务；若线程池中没有空闲的线程，则任务进入等待状态直到有空闲的线程。

1.2 Java 中的线程池实现机制

Java 在语言级实现了功能丰富的多线程编程机制[4]，对线程池的建立和维护提供了强大的支持。特别在 JDK1.5 及以后的版本中，任务执行抽象的首选不再是 Thread，而是 Executor。Executor 虽是一个简单的接口，但它提供了异步任务执行框架并支持多种不同类型的任务执行策略，ExecutorService 接口和 ScheduledExecutorService 接口对 Executor 进行了扩展，添加了管理线程执行和调度线程池的若干方法。通过 Executors 工具类提供的静态工厂方法可以创建符合特定需求的基于线程池执

行框架。

newChachedThreadPool()方法用于创建可缓存线程池的执行框架。当新的请求任务到达时,执行框架将尽可能地重用池中的空闲线程,若此时池中没有空闲线程,则添加新线程,这个方法对池的大小没有限制。另一方面,该执行框架能够自动回收空闲时间超过 60 s 的线程,以合理使用系统资源。对于执行大量短异步任务的程序而言,这种方式的线程池通常可提高性能。

newFixedThreadPool(int nThreads)方法建立的执行框架中的线程池具有固定数量的线程。每提交一个任务它就创建一个线程,直到达到池的限定值 nThreads,线程池的长度不再变化,新到达的任务在一个遵循先来先服务(FIFS)规则的无界队列中等待执行。

newScheduledThreadPool(int nThreads)方法建立的执行框架中的线程池也是定长的,它支持定时的以及周期性的任务的执行。

这些工厂方法返回的 Executor 都是 ThreadPoolExecutor()类的常用实例,能满足大部分线程池的应用需求。

2 设计思路

为保证异步任务处理的灵活性和可靠性,本文设计的思路为:任务持久化+Java 线程池+任务调度。

2.1 任务持久化

将待处理的任务信息保存在可信任的数据库中,同时要确保当任务处理服务器出问题后这些未执行成功、或未开始执行的任务不会被丢失。

2.2 任务调度

当任务信息都持久化在数据库中之后,需要将这些信息读取出来执行具体的业务逻辑操作,本文通过 ScheduledExecutorService 来实现对任务的循环调度,例如可采取每隔 2 min 扫描一次待处理任务列表,若有记录则提取出来执行。

3 具体实现

异步任务处理中各组成部分在运行过程中的调用关系如图 1。

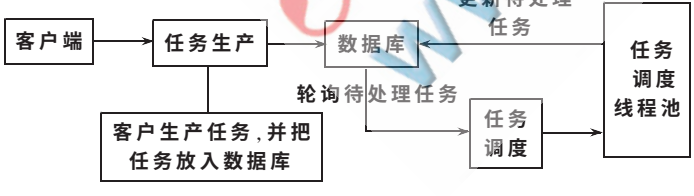

图 1 异步任务处理中的调用关系

当客户端访问服务器时,有耗时操作的任务,则把该任务放入数据库中。服务器每隔一段时间轮询存放待处理任务的表,若表中有任务,则任务调度线程池采用多线程机制来执行该任务。任务执行成功后,删除待处理任务表中的该任务信息,否则把该任务信息更新到任务失败表,进行人工干预。

3.1 任务数据表

建两张表,一张 task 表,用来存放待处理的任务;一张 task_fail 表用来存放失败的任务。两张表的结构一样,结构如表 1 所示。

表 1 task 和 task_fail 数据表结构

字段	数据类型	描述	可空
task_id	Int	PK,唯一标识	NOT
create_Time	Date	创建日期	NOT
handle_time	Date	任务待执行日期	NOT
task_handle	Varchar(50)	待执行任务类型	NOT
task_params	Varchar(200)	待执行任务需要的参数	NULL

task 表主要用来保存所有待处理的任务,每条任务信息属于一种任务类型,由 task_handle 字段标识,任务类型值为该类型任务的具体实现类名。task_params 字段提供了执行该任务需要的所有参数,为字符串,需要在具体任务实现类中解析。handle_time 字段提供了任务待执行的日期。

每条任务被执行之后根据执行情况进行删除或者更新操作,任务成功执行,就从 task 表中删除该记录。Task_fail 表主要用来保存执行失败、需要人工干预的任务记录,记录来源于 task 表。

3.2 任务处理过程

任务处理的过程如图 2 所示。

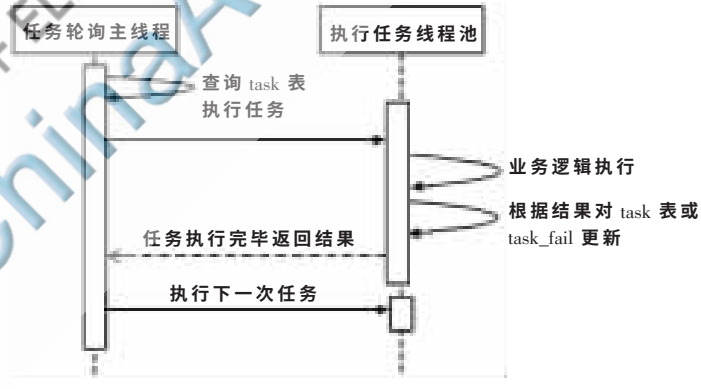

图 2 任务处理的过程

(1)当服务器启动后,根据调度策略每隔一段时间调度一次,而不管上次调度是否已经执行完毕;任务轮询主线程查询 task 表,从中取出一定条的数据。

(2)对从 task 表中查询出来的每条记录,将该条记录的 ID 放进本地 cache 中,根据记录中 task_handle 和 task_params 字段的值获得处理该任务对应的类及参数值,在异步线程池中利用反射机制来执行任务。

(3)具体处理类对该任务处理完成之后返回结果,系统对 tasks 表中该条记录进行删除,同时将 cache 中的记录 ID 清除、避免 cache 无限膨胀。若任务处理失败,系统就把该条记录插入到 task_fail 表中,以备人工干预。

(4)当到达下次执行时间时,再次扫描 tasks 表,循环

上面的逻辑。不过这次在任务处理之前，要先在本地 cache 中查询是否该条记录正在被处理，若 cache 中已经存在该条记录就无需处理了，以避免一些任务被重复并发执行。

3.3 任务轮询主线程的实现

Executor 的静态方法生成一个固定的线程池。线程池的线程是不会释放的，即使它空闲，这就会产生性能问题，如果线程池的大小为 200，当全部使用完毕后，所有的线程会继续留在池中，相应的内存和线程切换都会增加。如果要避免这个问题，就必须直接使用 ThreadPoolExecutor()来构造，设置“最大线程数”、“最小线程数”和“空闲线程存活的时间”。

为了线程池能按时间计划来执行任务，允许用户设定计划执行任务的时间，就要使用 newScheduledThreadPool(int nThreads)方法返回 ThreadPoolExecutor 类的实例。参数 nThreads 是设定线程池中线程的最小数目，当任务较多时，线程池会自动创建更多的工作线程来执行任务。其关键代码如下：

```
int nThreads=4 ;                    //指定线程池尺寸
//创建一个支持定时及周期性的任务执行的线程池实例 exec，池中含 nThreads 个线程
Executor exec=Executors. newScheduledThreadPool(nThreads);
Runnable task = new Runnable() {
public void run() {
        //查询数据库 task 表，有数据且执行时间到了
        //则另一个线程来执行查询到的任务
    }
};
    //1 min 后运行，并每隔 2 min 运行一次
exec.scheduleAtFixedRate(task,60,60×2,TimeUnit.SECONDS);
```

3.4 执行任务线程池的实现

执行任务的线程池采用 newFixedThreadPool (int nThreads)方法建立，线程池具有固定数量。关键代码如下：

```
ExecutorService exec = Executors.newFixedThreadPool(2);
Runnable run = new Runnable() {
    public void run() {
            //执行任务
    }
};
    exec.execute(run);
```

对于任务的生产者，只需要向 Task 表中 insert 记录即可，操作简单。待执行任务信息在可靠数据库中保存，即使任务处理出了问题也不会让未处理的任务信息丢失。

本文利用 Executor 接口提供的异步任务执行框架和任务执行策略，实现多任务的执行。在为具体应用线程池时往往需要根据应用的需求和处理任务的特点来优化线程池的使用，设置合适的“最大线程数”、“最小线程数”和“空闲线程存活的时间”，采用不同的策略调整线程池的工作线程数，才能达到最好的效果。

参考文献

[1] 于国良.建立高性能扩展的 Web 应用系统[J].微计算机信息，2006，18(04)：63-64.
[2] 郑扣根.操作系统概论[M].北京：高等教育出版社，2004.
[3] 王华，马亮，顾明.线程池技术研究与应用[J].计算机应用研究，2005(11)：141-145.
[4] Sun Microsystems Inc. Java Platform，Standard Edition 6 API Specification[EB/OL].(2011-12-20).http://java.sun.com/javase/6/docs/api/index.Html，2008.

(收稿日期：2011-12-29)

作者简介：
张文梅，女，1978 年生，讲师，硕士，主要研究方向：计算机应用、信息工程的教学与研究。