并发编程网 - ifeve.com

让天下没有难学的技术

APR 2014 25

方 腾飞

JAVA

聊聊并发（十）生产者消费者模式

本文首发于InfoQ 作者：方腾飞 校对：张龙

在并发编程中使用生产者和消费者模式能够解决绝大多数并发问题。该模式通过平衡生产线程和消费线程的工作能力来提高程序的整体处理数据的速度。

为什么要使用生产者和消费者模式

在线程世界里，生产者就是生产数据的线程，消费者就是消费数据的线程。在多线程开发当中，如果生产者处理速度很快，而消费者处理速度很慢，那么生产者就必须等待消费者处理完，才能继续生产数据。同样的道理，如果消费者的处理能力大于生产者，那么消费者就必须等待生产者。为了解决这种生产消费能力不均衡的问题，所以便有了生产者和消费者模式。

什么是生产者消费者模式

生产者消费者模式是通过一个容器来解决生产者和消费者的强耦合问题。生产者和消费者彼此之间不直接通讯，而通过阻塞队列来进行通讯，所以生产者生产完数据之后不用等待消费者处理，直接扔给阻塞队列，消费者不找生产者要数据，而是直接从阻塞队列里取，阻塞队列就相当于一个缓冲区，平衡了生产者和消费者的处理能力。

这个阻塞队列就是用来给生产者和消费者解耦的。纵观大多数设计模式，都会找一个第三者出来进行解耦，如工厂模式的第三者是工厂类，模板模式的第三者是模板类。在学习一些设计模式的过程中，如果先找到这个模式的第三者，能帮助我们快速熟悉一个设计模式。

生产者消费者模式实战

我和同事一起利用业余时间开发的Yuna工具中使用了生产者和消费者模式。首先我先介绍下Yuna工具，在阿里巴巴很多同事都喜欢通过邮件分享技术文章，因为通过邮件分享很方便，同学们在网上看到好的技术文章，复制粘贴发送就完成了分享，但是我们发现技术文章不能沉淀下来，对于新来的同学看不到以前分享的技术文章，大家也很难找到以前分享过的技术文章。为了解决这问题，我们开发了Yuna工具。Yuna取名自我非常喜欢的一款RPG游戏”最终幻想”中女主角的名字。

首先我们申请了一个专门用来收集分享邮件的邮箱，比如share@alibaba.com，同学将分享的文章发送到这个邮箱，让同学们每次

热门文章

Google Guava官方教程（中文版） 579,698 人阅读

Java NIO系列教程（一） Java NIO 概述 400,777 人阅读

Java并发性和多线程介绍目录 279,207 人阅读

Java NIO 系列教程 265,235 人阅读

Java NIO系列教程（十二） Java NIO与IO 225,537 人阅读

Java8初体验（二）Stream语法详解 206,699 人阅读

Java NIO系列教程（六） Selector 200,111 人阅读

Java NIO系列教程（三） Buffer 196,175 人阅读

Java NIO系列教程（二） Channel 194,373 人阅读

《Storm入门》中文版 177,356 人阅读

69道Spring面试题和答案 166,823 人阅读

Netty 5用户指南 160,913 人阅读

面试题 144,983 人阅读

并发框架Disruptor译文 143,715 人阅读

Java 7 并发编程指南中文版 138,369 人阅读

Java NIO系列教程（八） SocketChannel 127,752 人阅读

[Google Guava] 3-缓存 122,180 人阅读

[Google Guava] 2.3-强大的集合工具类：ja... 121,600 人阅读

[Google Guava] 1.1-使用和避免null 112,065 人阅读

Java NIO系列教程（七） FileChannel 110,496 人阅读

都抄送到这个邮箱肯定很麻烦，所以我们的做法是将这个邮箱地址放在部门邮件列表里，所以分享的同学只需要象以前一样向整个部门分享文章就行，Yuna工具通过读取邮件服务器里该邮箱的邮件，把所有分享的邮件下载下来，包括邮件的附件，图片，和邮件回复，我们可能会从这个邮箱里下载到一些非分享的文章，所以我们要求分享的邮件标题必须带有一个关键字，比如［内贸技术分享］，下载完邮件之后，通过confluence的web service接口，把文章插入到confluence里，这样新同事就可以在confluence里看以前分享过的文章，并且Yuna工具还可以自动把文章进行分类和归档。

为了快速上线该功能，当时我们花了三天业余时间快速开发了Yuna1.0版本。在1.0版本中我并没有使用生产者消费模式，而是使用单线程来处理，因为当时只需要处理我们一个部门的邮件，所以单线程明显够用，整个过程是串行执行的。在一个线程里，程序先抽取全部的邮件，转化为文章对象，然后添加全部的文章，最后删除抽取过的邮件。代码如下：

```
01 public void extract() {
02         logger.debug("开始" + getExtractorName() +
"。。");
03         //抽取邮件
04         List<Article> articles = extractEmail();
05         //添加文章
06         for (Article article : articles) {
07             addArticleOrComment(article);
08         }
09         //清空邮件
10         cleanEmail();
11         logger.debug("完成" + getExtractorName() +
"。。");
12     }
```

Yuna工具在推广后，越来越多的部门使用这个工具，处理的时间越来越慢，Yuna是每隔5分钟进行一次抽取的，而当邮件多的时候一次处理可能就花了几分钟，于是我在Yuna2.0版本里使用了生产者消费者模式来处理邮件，首先生产者线程按一定的规则去邮件系统里抽取邮件，然后存放在阻塞队列里，消费者从阻塞队列里取出文章后插入到conflunce里。代码如下：

```
01 public class QuickEmailToWikiExtractor extends
AbstractExtractor {
02
03 private ThreadPoolExecutor      threadsPool;
04
05 private
ArticleBlockingQueue<ExchangeEmailShallowDTO>
emailQueue;
06
07 public QuickEmailToWikiExtractor() {
08         emailQueue= new
ArticleBlockingQueue<ExchangeEmailShallowDTO>();
09         int corePoolSize =
Runtime.getRuntime().availableProcessors() * 2;
10         threadsPool = new
ThreadPoolExecutor(corePoolSize, corePoolSize, 10l,
TimeUnit.SECONDS,
11                 new LinkedBlockingQueue<Runnable>
(2000));
12
13     }
14
15 public void extract() {
16         logger.debug("开始" + getExtractorName() +
"。。");
17         long start = System.currentTimeMillis();
18
19         //抽取所有邮件放到队列里
20         new ExtractEmailTask().start();
21
22         // 把队列里的文章插入到Wiki
23         insertToWiki();
24
25         long end = System.currentTimeMillis();
26         double cost = (end - start) / 1000;
27         logger.debug("完成" + getExtractorName() +
",花费时间: " + cost + "秒");
28     }
29
30     /**
```

RECENT POSTS

Leader-Follower线程模型概述

《Apache Thrift官方文档》简介

《RabbitMQ官方指南》安装指南

在Windows上安装RabbitMQ

动手实现一个 LRU cache

《Thrift官方文档》Thrift支持的语言

《Thrift官方文档》 – docker构建说明

浅尝一致性Hash原理

Dubbo剖析-线程模型

分布式理论：CAP是三选二吗?

Jarslink1.6.1版本特性

《深入分布式缓存》之 “缓存为王”

《Thrift官方文档》翻译邀请

《Apache RocketMQ用户指南》之定时消息示例

使用Spring框架实现远程服务暴露与调用

Dubbo剖析-服务消费方Invoker到客户端接口的转换

Dubbo剖析-服务消费方远程服务到Invoker的转换

Linux零拷贝原理

阿里再开源！模块化开发框架JarsLink

Dubbo剖析-服务提供方Invoker到Exporter的转换

Dubbo剖析-服务提供方实现类到Invoker的转换

Dubbo剖析-增强SPI中扩展点自动包装的实现

Dubbo剖析-服务消费端异步调用

Dubbo剖析-服务直连

Dubbo剖析-服务分组与服务版本号

Dubbo剖析-监控平台的搭建与使用

Dubbo剖析-增强SPI的实现

Dubbo剖析-整体架构分析

《Linkerd官方文档》在ECS中运行Linkerd

《Linkerd官方文档》与Istio一起运行Linkerd

CATEGORIES

Android (3)

C++ (12)

CPU (2)

Framework (72)

akka (20)

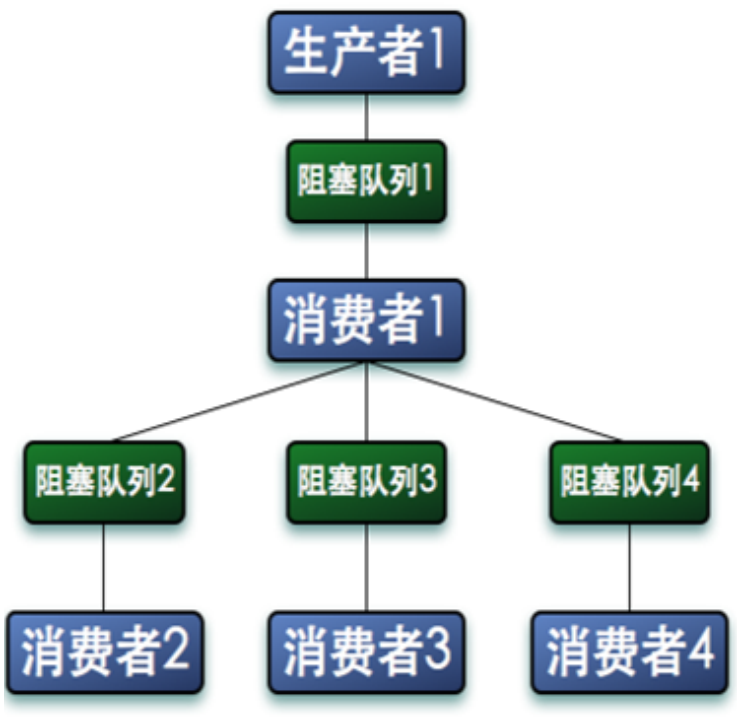
GO (6)

```
     * 把队列里的文章插入到Wiki
     */
    private void insertToWiki() {
        //登录wiki,每间隔一段时间需要登录一次
        confluenceService.login(RuleFactory.USER_NAME,
RuleFactory.PASSWORD);

        while (true) {
            //2秒内取不到就退出
            ExchangeEmailShallowDTO email =
emailQueue.poll(2, TimeUnit.SECONDS);
            if (email == null) {
                break;
            }
            threadsPool.submit(new
insertToWikiTask(email));
        }
    }

     protected List<Article> extractEmail() {
        List<ExchangeEmailShallowDTO> allEmails =
getEmailService().queryAllEmails();
        if (allEmails == null) {
            return null;
        }
        for (ExchangeEmailShallowDTO
exchangeEmailShallowDTO : allEmails) {
            emailQueue.offer(exchangeEmailShallowDTO);
        }
        return null;
    }

    /**
     * 抽取邮件任务
     *
     * @author tengfei.fangtf
     */
    public class ExtractEmailTask extends Thread {
        public void run() {
            extractEmail();
        }
    }
}
```

使用了生产者和消费者模式后，邮件的整体处理速度比以前要快了很多。

多生产者和多消费者场景

在多核时代，多线程并发处理速度比单线程处理速度更快，所以我们可以使用多个线程来生产数据，同样可以使用多个消费线程来消费数据。而更复杂的情况是，消费者消费的数据，有可能需要继续处理，于是消费者处理完数据之后，它又要作为生产者把数据放在新的队列里，交给其他消费者继续处理。如下图：



我们在一个长连接服务器中使用了这种模式，生产者1负责将所有客户端发送的消息存放在阻塞队列1里，消费者1从队列里读消息，然后通过消息ID进行hash得到N个队列中的一个，然后根据编号将消息存放在到不同的队列里，每个阻塞队列会分配一个线程来消费阻塞队列里的数据。如果消费者2无法消费消息，就将消息再抛回到阻塞队列1中，交给其他消费者处理。

TAGS

以下是消息总队列的代码；

```
/**
 * 总消息队列管理
 *
 * @author tengfei.fangtf
 */
public class MsgQueueManager implements IMsgQueue{

    private static final Logger              LOGGER
 = LoggerFactory.getLogger(MsgQueueManager.class);

    /**
     * 消息总队列
     */
    public final BlockingQueue<Message>
messageQueue;

    private MsgQueueManager() {
        messageQueue = new
LinkedTransferQueue<Message>();
    }

    public void put(Message msg) {
        try {
            messageQueue.put(msg);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    public Message take() {
        try {
            return messageQueue.take();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        return null;
    }

}
```

启动一个消息分发线程。在这个线程里子队列自动去总队列里获取消息。

```
/**
     * 分发消息，负责把消息从大队列塞到小队列里
     *
     * @author tengfei.fangtf
     */
    static class DispatchMessageTask implements
Runnable {
        @Override
        public void run() {
            BlockingQueue<Message> subQueue;
            for (;;) {
                //如果没有数据，则阻塞在这里
                Message msg =
MsgQueueFactory.getMessageQueue().take();
                //如果为空，则表示没有Session机器连接
上来，
需要等待，直到有Session机器连接上来
                while ((subQueue =
getInstance().getSubQueue()) == null) {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException
e) {
                        Thread.currentThread().interrupt();
                    }
                }
                //把消息放到小队列里
                try {
                    subQueue.put(msg);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        }
    }
```

使用Hash算法获取一个子队列。

```
/**
     * 均衡获取一个子队列。
     *
     * @return
     */
    public BlockingQueue<Message> getSubQueue() {
        int errorCount = 0;
        for (;;) {
            if (subMsgQueues.isEmpty()) {
                return null;
            }
            int index = (int) (System.nanoTime() %
subMsgQueues.size());
            try {
                return subMsgQueues.get(index);
            } catch (Exception e) {
                //出现错误表示，在获取队列大小之后，队
列进行了一次删除操作
                LOGGER.error("获取子队列出现错误",
```

```
e);
                if ((++errorCount) < 3) {
                    continue;
                }
            }
        }
    }
```

使用的时候我们只需要往总队列里发消息。

```
//往消息队列里添加一条消息
        IMsgQueue messageQueue =
MsgQueueFactory.getMessageQueue();
        Packet msg =
Packet.createPacket(Packet64FrameType.
TYPE_DATA, "{}".getBytes(), (short) 1);
        messageQueue.put(msg);
```

线程池与生产消费者模式

Java中的线程池类其实就是一种生产者和消费者模式的实现方式，但是我觉得其实现方式更加高明。生产者把任务丢给线程池，线程池创建线程并处理任务，如果将要运行的任务数大于线程池的基本线程数就把任务扔到阻塞队列里，这种做法比只使用一个阻塞队列来实现生产者和消费者模式显然要高明很多，因为消费者能够处理直接就处理掉了，这样速度更快，而生产者先存，消费者再取这种方式显然慢一些。

我们的系统也可以使用线程池来实现多生产者消费者模式。比如创建N个不同规模的Java线程池来处理不同性质的任务，比如线程池1将数据读到内存之后，交给线程池2里的线程继续处理压缩数据。线程池1主要处理IO密集型任务，线程池2主要处理CPU密集型任务。

小结

本章讲解了生产者消费者模式，并给出了实例。读者可以在平时的工作中思考下哪些场景可以使用生产者消费者模式，我相信这种场景应该非常之多，特别是需要处理任务时间比较长的场景，比如上传附件并处理，用户把文件上传到系统后，系统把文件丢到队列里，然后立刻返回告诉用户上传成功，最后消费者再去队列里取出文件处理。比如调用一个远程接口查询数据，如果远程服务接口查询时需要几十秒的时间，那么它可以提供一个申请查询的接口，这个接口把要申请查询任务放数据库中，然后该接口立刻返回。然后服务器端用线程轮询并获取申请任务进行处理，处理完之后发消息给调用方，让调用方再来调用另外一个接口拿数据。

About | Latest Posts

方 腾飞

花名清英，并发网(ifeve.com)创始人，畅销书《Java并发编程的艺术》作者，蚂蚁金服技术专家。目前工作于支付宝微贷事业部，关注互联网金融，并发编程和敏捷实践。微信公众号aliqinying。

⭐添加本文到我的收藏

Related Posts:

1. **聊聊并发（八）——Fork/Join框架介绍**
2. **聊聊并发（七）——Java中的阻塞队列**
3. **聊聊并发-Java中的Copy-On-Write容器**
4. **聊聊并发（五）原子操作的实现原理**
5. **聊聊并发（三）Java线程池的分析和使用**
6. **聊聊并发（六）ConcurrentLinkedQueue的实现原理分析**
7. **聊聊并发（四）深入分析ConcurrentHashMap**
8. **聊聊并发（二）Java SE1.6中的Synchronized**
9. **聊聊并发（一）深入分析Volatile的实现原理**
10. **Bug:StampedLock的中断问题导致CPU爆满**
11. **Java锁的种类以及辨析（四）：可重入锁**
12. **Callable和Future**
13. **java锁的种类以及辨析（一）：自旋锁**
14. **测试并发应用（五) 编写有效的日志**
15. **Oracle官方并发教程之中断**

Write comment　　Comments RSS

Trackback are closed | Comments (10)

mdusa_java　　Log in to Reply | QUOTE

04/25. 2014 10:11am

讲的很好啊，最近在看disruptor的源码和使用方法，网站上关于disruptor的讲解是基于老版本的，新版本改动特别大，啥时候能办文档也更新下啊

Log in to Reply | QUOTE

方 腾飞
04/25. 2014 10:57am

这个提议好，我们后续会跟进。

影子
04/25. 2014 10:19am

Log in to Reply | QUOTE

有多个线程去抽取邮件的时候如何保证各个线程抽取到邮件不会重复呢？

List allEmails = getEmailService().queryAllEmails();

看到这一句我觉得比较迷茫，每个线程都是取到所有的邮件？

liubey
04/25. 2014 10:43am

Log in to Reply | QUOTE

对 这个也是我关心的，每个消费者怎么分配，如果在多线程的情况下get到不重复的任务

方 腾飞
12/31. 2014 7:56pm

Log in to Reply | QUOTE

下载邮件这个是单线程的，下载完之后交给线程池处理，因为每隔5分钟抽取一次邮件，邮件不错的。关键是处理的慢，所以处理的时候用的多线程。

fan
07/09. 2014 12:44pm

Log in to Reply | QUOTE

方老师您好，我有个问题，是不是在大并发场景，针对于编程的角度，基于阻塞队列的生产/消费模型是解决最优的思路呢？

方 腾飞
12/31. 2014 7:57pm

Log in to Reply | QUOTE

这个最常用，其他的要看实际情况了。

veione
03/07. 2017 5:28pm

Log in to Reply | QUOTE

方老师您好，我有个问题，基于阻塞队列在游戏的在线PVP中是否适用呢？

THE END
04/10. 2017 3:22pm

Log in to Reply | QUOTE

老师你好!最近使用AIO有几个问题需要请教下：

1.aio绑定的时候，需要指定线程池，然后读取和写入会切换不同的线程进行处理，但是aio操作缓冲区的时候，无论是读取还是写入，都只能串行处理，那这样的话，那个设计是不是有问题？读取和写入如果是固定线程处理，那样话切换线程的以及线程的创建与销毁都省略了，不是更好？

houguiqiang
08/09. 2017 3:01pm

Log in to Reply | QUOTE

上面的例子有没有源码，比如说在git上？想具体看看

You must be logged in to post a comment.

Java8之使用新JS解释器 Nashorn编译Lambda表达式

更快的AtomicInteger