(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 108595282 A
(43)申请公布日 2018.09.28

(21)申请号 201810410581.4

(22)申请日 2018.05.02

(71)申请人 广州市巨硅信息科技有限公司
地址 510176 广东省广州市荔湾区周门北路28号B座五层501、502、503、504、505、506、507、508、509、510、511房(仅限办公用途)

(72)发明人 王毅群 赵娅利 蔡文 方金石 魏建平

(74)专利代理机构 北京捷诚信通专利事务所(普通合伙) 11221
代理人 王卫东

(51)Int.Cl.
G06F 9/54(2006.01)
G06F 9/50(2006.01)

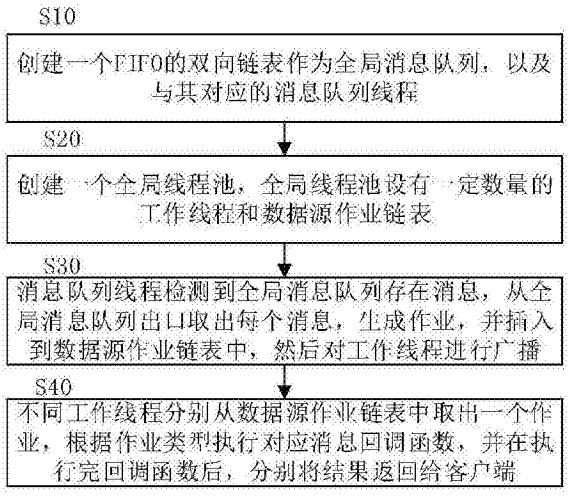
权利要求书1页 说明书7页 附图1页

(54)发明名称

一种高并发消息队列的实现方法

(57)摘要

本发明公开了一种高并发消息队列的实现方法,包括:创建一个FIFO的双向链表作为全局消息队列以及与其对应的消息队列线程;创建一个全局线程池,全局线程池设有一定数量的工作线程和数据源作业链表;消息队列线程检测到全局消息队列存在消息,从全局消息队列出口依次取出每个消息,生成作业,插入到全局线程池的数据源作业链表中,然后对工作线程进行广播;全局线程池的不同工作线程分别从数据源作业链表中取出一个作业,根据作业的类型执行对应消息回调函数,进行相应的业务处理,在执行完分别将处理结果返回给对应客户端。本发明能承载较大的数据请求并发量,且消费者将消息推送到消息队列后无需等待返回结果,提高响应速度,降低系统资源消耗。



CN 108595282 A

1.一种高并发消息队列的实现方法，其特征在于，包括以下步骤：

步骤S10、创建一个FIFO的双向链表作为全局消息队列，以及与其对应的消息队列线程；

步骤S20、创建一个全局线程池，全局线程池设有一定数量的工作线程和数据源作业链表；

步骤S30、消息队列线程检测到全局消息队列中存在消息，从全局消息队列出口依次取出每个消息，生成作业Job，并插入到全局线程池的数据源作业链表中，然后对工作线程进行广播；

步骤S40、全局线程池的不同工作线程分别从数据源作业链表中取出一个作业，根据作业的类型，执行对应消息回调函数，进行相应的业务处理，并在执行完回调函数后，分别将处理结果返回给对应客户端。

2.根据权利要求1所述的方法，其特征在于，所述全局消息队列存储在内存中。

3.根据权利要求1所述的方法，其特征在于，所述全局线程池设有工作线程的数量是由内核数量决定的，工作线程数量为内核数目的5倍至15倍。

4.根据权利要求1所述的方法，其特征在于，当并发大量消息请求时，全局线程池中已经没有空闲工作线程，此时消息队列线程不在从全局消息队列中取出消息，而是将无法处理的消息缓存在全局消息队列中，当有工作线程空闲时再进行处理。

5.根据权利要求1所述的方法，其特征在于，所述全局线程池中的工作线程采用一触即发的运行策略，具体为：

只要将作业Job插入到全局线程池的源数据作业链中，全局线程池中就有、且仅有一个工作线程立刻把该作业job取出并调用与该作业对应的回调函数对其进行处理，并将处理结果返回对应客户端；并通过内核定义的条件同步对象jobs_cond和互斥同步对象jobs_mutex实现通信和同步。

6.根据权利要求1所述的方法，其特征在于，全局线程池初始化时等待运行的工作线程数目为20个。

7.根据权利要求1所述的方法，其特征在于，利用一个消息队列管理组件MQ Manager记录全局消息队列的长度，全局线程池中工作线程的数量、工作线程运行的数量、工作线程空闲数量；

所述消息队列管理组件MQ Manager根据全局线程池的工作线程运行数量判断是否需要创建新的工作线程，满足当前高并发的数据请求环境下数据处理需求；并在数据请求进入平缓期，自动关闭无任务的工作线程。

8.根据权利要求1所述的方法，其特征在于，服务器各个业务线程调用推入函数MQ_PUSH将消息推送到全局消息队列；

消息队列线程不断轮询全局消息队列，当有消息存在时，消息队列线程调用取出函数MQ_PULL将消息从全局消息队列中取出。

9.根据权利要求1所述的方法，其特征在于，在高并发数据请求环境下，利用x86 cmpxchg指令实现了一种无锁机制，解决了高并发下资源互斥时的原子操作，具体为：

一个工作线程调用cmpxchg函数获得锁，如果没有获得锁，工作线程不会阻塞，cmpxchg函数会立刻返回。

一种高并发消息队列的实现方法

技术领域

[0001] 本发明涉及计算机通信技术，具体涉及一种高并发消息队列的实现方法。

背景技术

[0002] 消息队列(Message Queue，MQ)是一种应用程序间的通信方式，消息发送后可以立即返回，由消息系统来确保消息的可靠传递，消息发布者只管把消息发布到MQ中而不用管谁来取，消息使用者只管从MQ中取消息而不管是谁发布的。

[0003] MQ常被用于业务解耦、最终一致性、广播、错峰流控等场景中，例如随着业务的发展订单量增长，需要提升系统服务的性能，这时可以将一些不需要立即生效的操作拆分出来异步执行，比如发放红包、发短信通知等，这种场景就是业务解耦，此时可以用MQ，在下单的主流程(比如扣减库存、生成相应单据)完成之后发送一条消息到MQ让主流程快速完结，而由另外的单独线程拉取MQ内的消息(或者由MQ推送消息)，当发现MQ中有发红包或发短信之类的消息时，执行相应的业务逻辑。

[0004] 目前，消息队列有ActiveMQ、RabbitMQ、ZeroMQ等，主要存在以下缺点：

[0005] (1)由于使用内核级别的同步对象，影响速度，且对数据未进行解耦，效率不高，无法承载较大的并发量；

[0006] (2)消费者将消息推送到消息队列后需要等待返回结果，影响处理速度；

[0007] (3)从消息队列取消息时需要创建工作线程执行消息内容，尤其在高并发的数据请求环境下，需要同时创建、启动众多工作线程，负载巨大。

发明内容

[0008] 本发明所要解决的技术问题是现有的消息队列无法承载较大的并发量、消费者将消息推送到消息队列后需要等待返回结果、从消息队列取消息时需要创建工作线程执行消息内容的问题。

[0009] 为了解决上述技术问题，本发明所采用的技术方案是提供一种高并发消息队列的实现方法，包括以下步骤：

[0010] 步骤S10、创建一个FIFO的双向链表作为全局消息队列，以及与其对应的消息队列线程；

[0011] 步骤S20、创建一个全局线程池，全局线程池设有一定数量的工作线程和数据源作业链表；

[0012] 步骤S30、消息队列线程检测到全局消息队列中存在消息，从全局消息队列出口依次取出每个消息，生成作业Job，并插入到全局线程池的数据源作业链表中，然后对工作线程进行广播；

[0013] 步骤S40、全局线程池的不同工作线程分别从数据源作业链表中取出一个作业，根据作业的类型，执行对应消息回调函数，进行相应的业务处理，并在执行完回调函数后，分别将处理结果返回给对应客户端。

[0014] 在上述方法中，所述全局消息队列存储在内存中。

[0015] 在上述方法中，所述全局线程池设有工作线程的数量是由内核数量决定的，工作线程数量为内核数目的5倍至15倍

[0016] 在上述方法中，当并发大量消息请求时，全局线程池中已经没有空闲工作线程，此时消息队列线程不在从全局消息队列中取出消息，而是将无法处理的消息缓存在全局消息队列中，当有工作线程空闲时再进行处理。

[0017] 在上述方法中，所述全局线程池中的工作线程采用一触即发的运行策略，具体为：

[0018] 只要将作业Job插入到全局线程池的源数据作业链中，全局线程池中就有、且仅有一个工作线程立刻把该作业job取出并调用与该作业对应的回调函数对其进行处理，并将处理结果返回对应客户端；并通过内核定义的条件同步对象jobs_cond和互斥同步对象jobs_mutex实现通信和同步。

[0019] 在上述方法中，全局线程池初始化时等待运行的工作线程数目为20个。

[0020] 在上述方法中，利用一个消息队列管理组件MQ Manager记录全局消息队列的长度，全局线程池中工作线程的数量、工作线程运行的数量、工作线程空闲数量；

[0021] 所述消息队列管理组件MQ Manager根据全局线程池的工作线程运行数量判断是否需要创建新的工作线程，满足当前高并发的数据请求环境下数据处理需求；并在数据请求进入平缓期，自动关闭无任务的工作线程。

[0022] 在上述方法中，服务器各个业务线程调用推入函数MQ_PUSH将消息推送到全局消息队列；

[0023] 消息队列线程不断轮询全局消息队列，当有消息存在时，消息队列线程调用取出函数MQ_PULL将消息从全局消息队列中取出。

[0024] 在上述方法中，在高并发数据请求环境下，利用x86cmpxchg指令实现了一种无锁机制，解决了高并发下资源互斥时的原子操作，具体为：

[0025] 一个工作线程调用cmpxchg函数获得锁，如果没有获得锁，工作线程不会阻塞，cmpxchg函数会立刻返回。

[0026] 与现有技术相比，本发明消息队列的性能取决于CPU或内存的瓶颈(CPU或内存的处理速度远远高于磁盘等外设备的处理速度)，消息队列能承载较大的数据请求并发量，通过将任务分解成一个一个小任务，并对各个小任务进行异步并发处理，使消费者将消息推送到消息队列后无需等待返回结果，在执行完回调函数MQ_CALLBACK后会根据消息返回给确定的客户端，从而实现数据解耦，并引入全局线程池对工作线程统一管理，从消息队列取消息时不需要再创建工作线程，可以立即执行消息内容，不仅提高响应速度，而且降低系统资源消耗，避免操作系统鲁棒性的降低。

附图说明

[0027] 图1为本发明提供的一种高并发消息队列的实现方法的结构框架图；

[0028] 图2为本发明提供的一种高并发消息队列的实现方法的流程图。

具体实施方式

[0029] 在服务器模式下，及时数据请求，如果同步返回数据，服务器的吞吐量TPS，应当满

足如下等式：

[0030] QPS＝min{CPU,内存,外部接口,IO操作}；

[0031] 为了满足上述等式，本发明设计了一套解决思路，通过服务器内部组件把一个大任务分解成一个一个的小任务，实现数据的解耦，并将各个小任务进行异步并发处理，从而提高服务器吞吐量TPS，该设计的服务器的吞吐量TPS满足如下等式：

[0032] QPS＝max{min{CPU,内存},min{外部接口,IO操作}}；

[0033] 因为，min{CPU,内存}>min{外部接口,IO操作}；

[0034] 所以，QPS＝min{CPU,内存}；

[0035] 可见，本发明的性能取决于CPU或内存的瓶颈，由于CPU或内存的处理速度远远高于磁盘等外设备的处理速度，与现有技术取决于CPU、内存、外部接口或IO操作相比，性能高效，高效的性能就决定了消息队列能承载更大的并发量。下面结合说明书附图和具体实施方式对本发明做出详细说明。

[0036] 如图1、图2所示，本发明提供的一种高并发消息队列的实现方法，包括以下步骤：

[0037] 步骤S10、创建一个FIFO(First Input First Output,先入先出队列)的双向链表作为全局消息队列，以及与其对应的消息队列线程；

[0038] 步骤S20、创建一个全局线程池，全局线程池设有一定数量的工作线程和数据源作业链表，即全局线程池是一个工作线程容器，其中，全局线程池设有工作线程的数量是由内核数量决定的，工作线程数量为内核数目的5倍至15倍；

[0039] 步骤S30、消息队列线程检测到全局消息队列中存在消息(通过不断轮询全局消息队列检测其中是否有消息)，从全局消息队列出口依次取出每个消息，生成作业Job，并插入到全局线程池的数据源作业链表中，然后对工作线程进行广播；

[0040] 步骤S40、全局线程池的每个工作线程分别从数据源作业链表中取出一个作业，并根据作业的消息类型，执行对应消息回调函数MQ_CALLBACK，进行相应的业务处理，在执行完回调函数MQ_CALLBACK后将处理结果返回给对应客户端。

[0041] 当系统进入高峰请求时期，本发明的全局消息队列的全局线程池中的工作线程对消息进行并发处理，提高单位时间内消息的处理量，从而可以抗住IO操作，外部接口等关键组件的访问压力，提高系统的抗压性高；而且由于全局消息队列为FIFO，每个业务线程只需将消息请求依次放入全局消息队列即可，消息队列线程会根据先进先出原则依次将消息取出，送入全局线程池的元数据链表进行处理，这样不管多少客户端进行消息请求，多少线程工作取消息，整个工作流程的各个环节都是具有单一性的，确保消息只被送达一次，从而提高消息传递安全性高。

[0042] 在本发明中，当并发大量消息请求时，全局线程池中已经没有空闲工作线程，由于本发明的全局消息队列对存入消息的数量没有设置上限，此时消息队列线程不在从全局消息队列中取出消息，而是将无法处理的消息缓存在全局消息队列中，当有工作线程空闲时再进行处理，这种冗余机制不仅保证所有消息都能被处理，而且保证系统数据安全稳定运行。

[0043] 全局消息队列存储在内存中，用于线程间的通信，CPU可以直接处理，不要从磁盘等其他外设备加载，更加轻捷、方便，进队Enqueue与出队Dequeue的运行性能远高于各种开源MQ组件，全局消息队列的数据结构为：

[0044]

```
typedef struct _DaveNode {
  MessageTag *value;//节点 value
  struct _DaveNode *next;//后一个节点
  struct _DaveNode *prev;//前一个节点
} DaveNode ;

typedef struct _DaveQueue {
  const void *_;
  DaveNode *head;//指向头节点
  DaveNode *tail;//指向尾节点
  DaveNode *nil;//指向空节点
} DaveQueue;
```

在本发明中，全局消息队列是存储消息 MessageTag 的容器，消息 MessageTag 数据结构定义如下：

```
typedef struct _MESSAGETAG {
  MESSAGE_TYPE Type;//消息类型
  U8 *Tag;// Tag 指针
  int length;// Tag 指针指向内容的长度
  int arg;//arg 参数
  C_DEVID fromId;//源设备 ID
  C_DEVID toId;//目标设备 ID
  C_DEVID gId;//设备组 ID
  DAVE_MESSAGE_CALLBACK cb;//消息回调函数
```

```
} MessageTag;
```

在本发明中，全局线程池的数据结构定义如下：

```
typedef struct _WorkQueue {
Worker *workers;//工作线程数组指针
        Job *waiting_jobs;//作业链指针
        pthread_mutex_t jobs_mutex;//mutex 同步对象
        pthread_cond_t jobs_cond;//cond 同步对象
        int counter;
} WorkQueue;

typedef struct _ThreadPool {
const void *_;
WorkQueue *wq;// WorkQueue 对象指针
} ThreadPool;
```

[0045]

[0046] 其中，workers是指向工作线程链表的指针，Waiting_jobs是指向数据源作业Job链表的指针。

[0047] 在本发明中，消息队列线程具体执行代码如下：

[0048] Job*job＝(Job*)malloc(sizeof(Job));//分配Job内存空间

[0049] job->job_function＝ntyDaveMqHandleCallback;//设置回调函数指针

[0050] job->user_data＝tag;//设置数据源

[0051] ntyThreadPoolPush(worker,job);//把job插入线程池的作业链

[0052] 本发明的全局线程池初始化时等待运行的工作线程数目为20个，全局线程池中的工作线程采用一触即发的运行策略，避免在高并发的数据请求环境下，同时启动众多线程，出现操作系统负载过大，致使系统出现处理异常。所述一触即发的运行策略为：只要将作业Job插入到全局线程池的源数据作业链中，全局线程池中就有、且仅有一个工作线程立刻把该作业job取出并调用与该作业对应的回调函数对其进行处理，并将处理结果返回对应客户端；并通过内核定义的条件同步对象jobs_cond和互斥同步对象jobs_mutex实现通信和同步。

[0053] 在本发明中通过一个消息队列管理组件MQ Manager，记录全局消息队列的长度，全局线程池中工作线程的数量、工作线程运行的数量、工作线程空闲数量，并根据全局线程池的工作线程运行数量判断是否需要创建新的工作线程，满足当前高并发的数据请求环境下数据处理需求，并在数据请求进入平缓期，自动关闭无任务工作线程，释放操作系统资

源。

[0054] 本发明引入上述消息队列管理组件进行工作线程统一管理，带来如下

[0055] 有益效果：

[0056] (1)不再不断重复地创建与销毁线程，降低系统资源消耗，避免降低操作系统的鲁棒性；

[0057] (2)从全局消息队列里面取消息进行处理时，不需要创建工作线程，能立即执行，提高响应速度。

[0058] 在本发明中，因为全局线程池的工作线程都是已经创建好的，在运行时，只需非常方便地进行MQ_PUSH与MQ_PULL原语操作即可，不需要做其它工作，即服务器各个业务线程调用推入函数MQ_PUSH将消息推送到全局消息队列，再由消息队列线程调用取出函数MQ_PULL将其从全局消息队列中取出，函数MQ_PUSH实现代码如下：

[0059] VALUE_TYPE*tag＝malloc(sizeof(VALUE_TYPE));//分配tag内存空间

[0060] memset(tag,0,sizeof(VALUE_TYPE));//内存空间初始化置0

[0061] tag->Tag＝malloc(sLen+1);//分配Tag的内存空间

[0062] memset(tag->Tag,0,sLen+1);//初始化

[0063] memcpy(tag->Tag,filename,sLen);//把filename保存到Tag

[0064] tag->length＝sLen;//设置length

[0065] tag->fromId＝senderId;//设置源设备ID

[0066] tag->toId＝gId;//设置目标设备ID

[0067] tag->Type＝MSG_TYPE_VOICE_DATA_REQ_HANDLE;//设置消息类型

[0068] tag->cb＝ntyVoiceDataReqHandle;//设置回调函数

[0069] ntyDaveMqPushMessage(tag);//把tag放入消息队列

[0070] 在本发明中，消息队列线程不断轮询全局消息队列，判断是否有消息存在，有消息存在，调用函数MQ_PULL将其取出，函数MQ_PULL实现代码如下：

[0071]

```
node = ntyDaveDeQueue(Queue);//从消息队列把消息取出
  if (node != Queue->nil &&  node != NULL) {//判断消息是否为空
      ntyDaveMqPullMessage(node->value);//把消息放入全局线程池
      free(node);//释放内存
  }
```

[0072] 在高并发数据请求环境下，通常采用线程锁pthread_mutex_lock或者采用二阶锁实现线程间的同步，从而使得服务器性能非常低下，本发明利用x86cmpxchg指令实现了一种无锁机制，解决了高并发下资源互斥时的原子操作，并没有用到操作系统的内核对象，所以实现了一种无锁的同步机制，具体为：一个工作线程调用cmpxchg函数获得锁，如果没有

获得锁，工作线程不会阻塞，cmpxchg函数会立刻返回，从而提高了实时性和并发性。

[0073] cmpxchg函数实现代码如下：

[0074]

```
unsigned long cmpxchg(void *addr, unsigned long _old, unsigned
long _new) {
    unsigned long prev;
    volatile unsigned int *_ptr = (volatile unsigned int *)(addr);
    __asm__ volatile (
"lock; cmpxchg %1, %2"
        : "=a" (prev)
        : "r" (_new), "m" (*_ptr), "0" (_old)
        : "memory");
    return prev;
}
```

[0075] 本发明并不局限于上述最佳实施方式，任何人应该得知在本发明的启示下做出的结构变化，凡是与本发明具有相同或相近的技术方案，均落入本发明的保护范围之内。

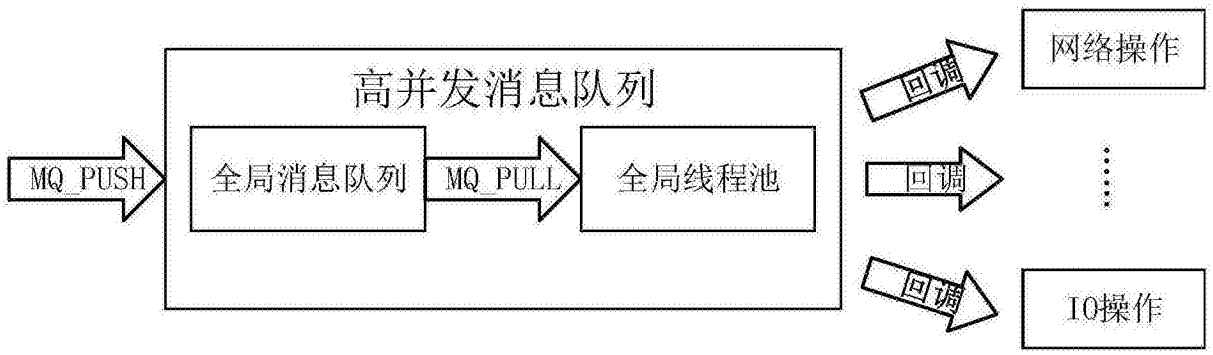

高并发消息队列
MQ_PUSH
全局消息队列
MQ_PULL
全局线程池
回调
回调
回调
网络操作
IO操作

图1

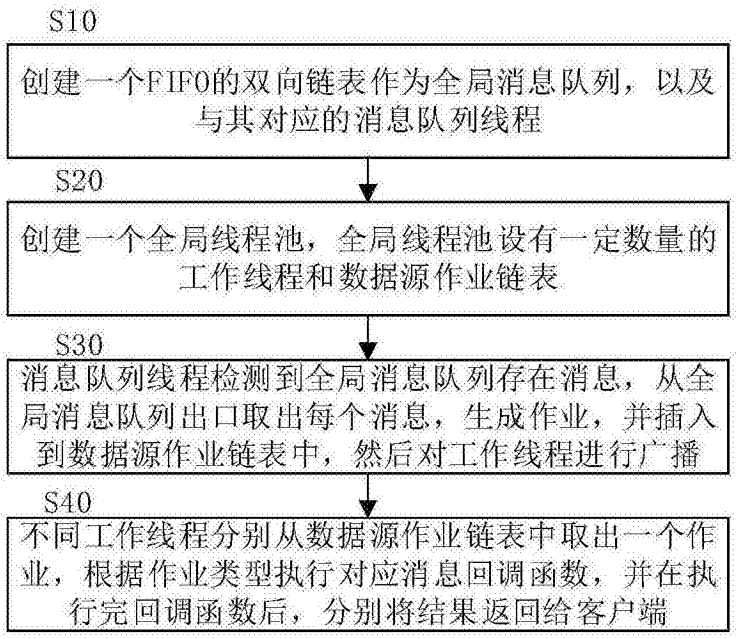

S10
创建一个FIFO的双向链表作为全局消息队列，以及与其对应的消息队列线程
S20
创建一个全局线程池，全局线程池设有一定数量的工作线程和数据源作业链表
S30
消息队列线程检测到全局消息队列存在消息，从全局消息队列出口取出每个消息，生成作业，并插入到数据源作业链表中，然后对工作线程进行广播
S40
不同工作线程分别从数据源作业链表中取出一个作业，根据作业类型执行对应消息回调函数，并在执行完回调函数后，分别将结果返回给客户端

图2