技术交底书

|  |  |
| --- | --- |
| 内部专利号 |  |
| 交底书名称 | 一种突破单机处理极限可无限横向扩展的作业调度方案 |
| 发明人（可以多个） | 梅志文 |
| 第一发明人身份证号码 | 430822198405192337 |
| 交底书撰写人 | 梅志文 |
| 电话&手机 | 13520973391 |
| Email | [meizhiwen@jd.com](mailto:meizhiwen@jd.com) |
| 交底书提交日期 | 2018/03/28 |
| 所属部门 | 旅游研发部 |

交底书注意事项：

1、代理人并不是技术专家，交底书要使代理人能看懂，尤其是背景技术和详细技术方案，一定要写得全面、清楚。

2、交底书中出现的英文缩写，必须配上中文译文及英文全称。

3、全文对同一事物的叫法应统一，避免一种事物前后出现多种叫法。

4、在后续与专利代理人进行沟通时，对于代理人的疑问应认真讲解，要求补充的材料应及时补充。

5、专利法规定，专利必须是一个技术方案，应该阐述发明目的是通过什么技术方案来实现的，不能只有技术原理，也不能只做功能或优点的宣传式介绍。

# 相关技术背景（背景技术），与本发明最相近的现有实现方案（现有技术）

1.1　背景技术

在大型企业及互联网应用中，各系统都或多或少会通过任务调度的方式来帮用户完成提交订单并付完款之后的业务流程。

以京东火车票系统为例，当用户通过查询车次余票，提交订单并支付完成后，针对用户的操作流程就已经完成了，然后火车票后端系统的预订定时任务会每隔10秒扫描“支付完成并等待预订”的订单，然后请求供应商进行火车票的预订。

对于大部分的java系统，会将开发好的定时任务部署到单台服务器上进行运行，同时再部署另一台做为备机，当其中一台服务器宕机后，启动另一台服务器继续跑定时任务进行订单的处理。

本专利提供了一种突破单机处理极限可无限横向扩展的作业调度方案，它基于zookeeper事件通知机制，通过job服务端动态协调所有客户端服务器来并行处理任务，并且可以无限横向扩容和缩容。

1.2 与本发明相关的现有技术方案

1. 使用quartz开发好定时任务，部署三到五台服务器，通过zookeeper进行leader选举，当任务定时调度执行时，判断当前是不是leader，如果是leader就执行相应的定时任务逻辑处理，当leader宕机后，zookeeper会重新选举出新的leader服务器，由新leader继续执行定时业务。

1.2.2 现有技术的缺点

1. 当前技术方案无法充分利用服务器的资源执行任务处理，每一次定时任务调度执行时，始终只有一台服务器处于active状态，其它所有服务器处于standy状态。
2. 当数据量达到一定量级，会导致单台服务器处理不过来，单机的处理能力始终是有极限的，无法通过横向扩展来解决性能瓶颈和吞吐量。

# 2、本发明技术方案的详细阐述（发明内容）

2.1 本发明所要解决的技术问题（发明目的）

1. 无法充分利用服务器资源，只能单台服务器处理，其它服务器处于空闲的问题
2. 无法自动横向扩容和缩容的问题
   1. 本发明提供的完整技术方案
      1. 整体架构



图 1整体架构

* + 1. 部署架构



图 2部署架构

* + 1. 本方案的核心思路是由job的服务端将一个任务动态平均分配到多台客户端的服务器上并行处理，总共分以下6个部分进行详细介绍，包含主要概念、zk数据存储结构、 job服务端系统，job客户端、总体业务流程、zk宕机的降级处理。

1. **主要概念**

|  |  |
| --- | --- |
| 名称 | 描述 |
| Job(作业) | 1. 也称作业，可以理解为quartz的 (JobDetail、trigger、scheduler) 的集合。 2. 必须实现提供的统一的Job接口。 |
| 总分片数 | 1. 指的是Job执行的任务要分成多少个子任务进行处理 |
| 任务项 | 1. 指的是每个客户端的服务器分配到的子任务集合   举例：如果Job1总分片数为15，Job1执行的服务器数量为5，则每台服务器上分配到的任务项数量为3个，第一台服务器分配到的任务项为0、1、2，第二台分配到的任务项为3、4、5，依次往下。 如果Job执行的服务器数量为4，则第一台服务器分配到的任务项为0,1,2,12,第二台为3,4,5,13，第三台为6,7,8,14，第四台为9,10,11  说明：接入方系统需要自行处理任务项与真实的业务数据处理的对应关系 |
| 执行分片 | 给某个job为所有客户端服务器分配好任务项的过程 |
| Job客户端 | 1. 业务系统，定时调度执行job的服务器 |

1. **Zookeeper注册中心数据存储结构**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ZK目录编号 | ZK目录 | 目录描述 | Data描述 | 数据格式 |
| A | /worker/job/${appName}/${job1} | 表示job名称  Job唯一，英文字母 | Data存储job的配置信息 | {  appName：接入业务线系统的名称  jobName：job名称  jobGroup: job分组  jobBean：spring bean名称  crontab: job执行表达式  jobDesc: job描述  shardingNum： 总分片数  dataSource： 数据源名称  } |
| B | /worker/job/${appName}/${job1}/sharding | 分片标记目录 | 存储分片的状态 | {  needSplit：true/false  splitState: unSplit/splited  } |
| C | /worker/config/ ${appName} | 存储job相关的配置目录 | Job分片开关：可以手动点击“打开”和“关闭”。 | {  splitSwitch:false  } |
| D | /worker/job/ ${appName}/${job1}/execute | Job的手动执行目录 | 存储该job手动执行的版本号 | {  Cmd：(startJob,stopJob,executeOne)  } |
| E | /worker/job/ ${appName}/${job1}/instances/${ip1}/sharding | Job对应的服务器的分片项目录 | 存储服务器对应的分片项。  Job执行机应该要监听这个目录 | 0,1 |
| G | /worker/server/${appName}/${ip1} | 存储的是注册进来的服务器ip |  |  |

Zk目录结构逻辑视图：



图 3

各系统通过Zk目录交互的业务流描述：

1. 当业务线系统申请接入分布式任务调度平台审核成功后，job服务端会为业务系统创建好对应的zk目录。
2. 当业务线系统的job执行机启动时，会创建好对应的zk目录，job服务端会收到目录变化的事件通知，然后job服务端会修改/worker/job/${appName}/${job1}/sharding目录的分片状态数据触发服务端重新分片。
3. 在job服务端添加job成功后，会往zk中创建好对应的目录，并触发job分片动作。
4. 当job服务端收到重新分片的事件通知，会为每个job执行机分配好任务项，并且会创建好对应的zk目录。
5. 当创建好对应的zk目录，job执行机会收到目录变化的事件通知，然后创建job对应的本地调度执行器，并启动执行。
6. **Job服务端系统**

负责job的管理和执行分片的过程，主要包含4个模块，接入申请，job管理模块，job执行分片模块，系统监控模块。

1. **接入申请：**负责各业务线系统的接入，为每个业务系统创建好zookeeper中的根目录结点
2. **Job管理模块：**

应用启动初始化：

1. 进行leader选举，由选举出的leader负责为添加的job执行分片动作。
2. 监听zk中G目录的结点变化，读取A目录下所有的job信息。
3. 监听B目录的结点变化，获取分片的状态，如果需要分片并且未分片则调用job执行分片模块为所有客户端分配任务项。
4. 创建两个rpc远程服务：上报客户端负载接口、上报job执行轨迹接口。

页面功能：提供添加job、修改job功能。通过添加job功能将job的基本信息写入到zookeeper的A目录中，并直接调用job执行分片模块为所有客户端分配任务项。

1. **Job执行分片模块：**首先获取job的总分片数和当前注册进来的客户端服务器数量，然后根据默认的分片算法为每台客户端服务器分配好任务页。

**默认的分片算法如下：**

设总分片数为*x(shardingNum)，*执行机个数为*y(clientNum)*

* *shardingNumPerClient = f (x) ÷ f(y)*

表示每个执行机分配到的分片个数

* *modShardingNum = f(x) mod f(y)*

两个数取模表示剩下的分片项个数

* *f(Client client …) = f\_sort(${Client client …})*

表示为所有job执行机排好序

* 从0一直到*f(x)*，根据*f(Client client …)*依次为每个执行机分配当前的数字，直到每个执行机分配到*shardingNumPerClient*个数字为止，然后再从0到f(x)剩下的数字开始依次为每个执行机分配1个数字，直到0到f(x)之间剩下的*modShardingNum*个数字分配完为止。

例如：如果Job1总分片数为15，Job1的执行机数量为5，则每台服务器上分配到的任务项数量为3个，第一台服务器分配到的任务项为0、1、2，第二台分配到的任务项为3、4、5，依次往下。 如果Job执行的服务器数量为4，则第一台服务器分配到的任务项为0,1,2,12,第二台为3,4,5,13，第三台为6,7,8,14，第四台为9,10,11

**触发job服务端重新分片的时机：**

1. Job客户端扩容或者缩容时
2. 在job服务端手动添加新的job时
3. **系统监控模块：**查询数据库存储的客户端的负载数据，超过一定阈值后发邮件和短信报警。查询数据库存储的job执行轨迹，监控执行超时的job进行报警，长时间未启动的job监控报警。
4. **Job客户端**

负责接收zookeeper推送的分片完成的事件，负责创建和启动quartz sheduler调度执行器，每次调度执行具体的业务逻辑时，首先获取分配好自己的任务项，然后根据任务项再查询具体的订单数据进行业务逻辑处理。

应用启动时，会初始化用于job写文件的log4j框架，当新增了job后，客户端会将job信息写入到固定的/export/Logs/jd.job.client目录中

1. **总体的业务流程**



图 4业务流程

1. **Zk宕机降级处理**
2. 正常情况下，当job客户端创建新的quartz job时，利用log4j异步的将job基本信息写入到本地文件当中
3. 当job客户端启动后，会跟job服务端建立tcp长连接，zk宕机后，在job服务端手动打开降级开关，job服务端通过与每个客户端的长连接发送降级网络请求命令，客户端接收到降级请求时，会停止当前所有启动的job，并从job注册表中删除，然后由提前指定好的服务器A读取本地文件中的所有job信息，创建和启动降级的quartz的scheduler执行器，单机处理所有的任务。当zk恢复后，手动再关闭降级开关，然后所有服务器接收到降级开关关闭命令后，停止当前所有启动的降级job，并从job注册表删除，然后读取本地文件中所有job信息，创建和启动正常的quartz scheduler执行器，开始走正常的分布式任务调度流程。
   1. 本发明技术方案带来的有益效果
4. 通过动态的为所有job客户端分配任务项，让多台客户端的服务器并行处理相关的任务数据，突破单机处理任务数据的权限瓶颈
5. 通过job服务端的重新分片机制，可以无限横向扩容或者缩容
6. 可以随时修改job的执行策略和总的分片数，来动态调整job执行的吞吐量和性能，当有剩余的服务器没有分配到任务时，可以增加job的总分片数，让新的服务器分配到任务项进行数据处理。
   1. 本发明的关键点和欲保护点是什么？

一种基于zookeeper事件监听机制，通过job管理端动态协调所有客户端服务器来并行处理任务，并且可以无限横向扩容和缩容，突破单机处理极限瓶颈的方法。

2.5 交底书专业技术领域的技术术语的名词解释

1. Zookeeper：是一个分布式的，开放源码的分布式应用程序协调服务
2. Quartz：是一个完全由java编写的开源作业调度框架