**火车票分布式worker**

**详细设计**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修改人 | 修改项 | 时间 |
| 0.1 | 梅志文 | 完成初版设计 | 2018.03.05 |
|  | 梅志文 | 修改zk存储目录结构  在/worker/job/和/worker/server/目录下添加子目录${app} | 2018.03.13 |

# 设计目标

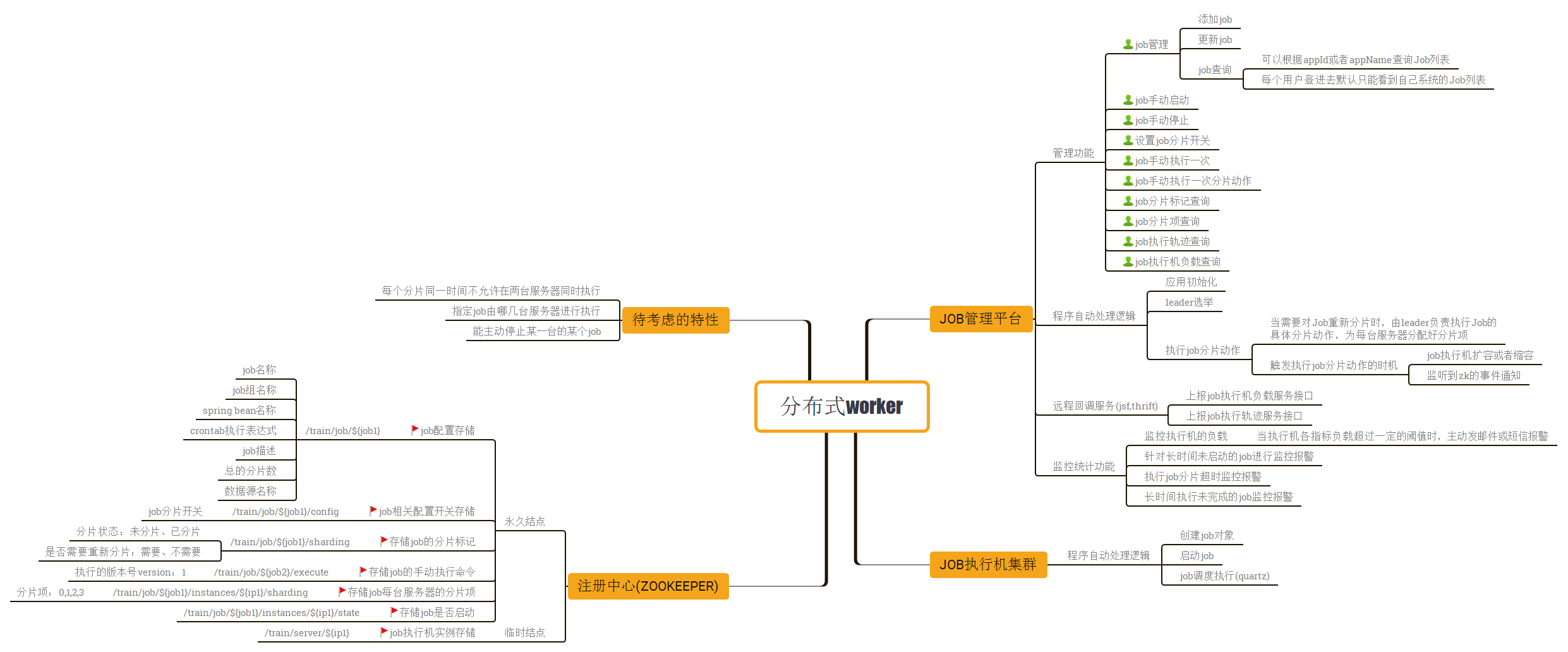
1. 火车票分布式worker提供一个分布式任务调度解决方案，目的是让一个任务能动态平均分配到多台服务器上并行处理，充分利用服务器的资源来提高任务的处理效率，消除单机处理极限的瓶颈
2. 可以供各业务线系统接入，只需引入一个jar包，实现自己的Job执行逻辑，然后在管理平台动态添加该Job既可实现job的分布式调度执行

# 主要概念

|  |  |
| --- | --- |
| 名称 | 描述 |
| Job(作业) | 1. 也称作业，可以理解为quartz的Job的概念 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 |
| 分片项 | 1. 指的是每个服务器分配到的子任务集合   举例：如果Job总分片数为15，Job执行的服务器数量为5，则每台服务器上分配到的分片项数量为3个，第一台服务器分配到的分片项为0、1、2，第二台分配到的分片项为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分片项为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分片项与真实的业务数据处理的对应关系 |
| 执行分片 | 给某个job为所有服务器分配好分片项的过程 |
| Job执行机 | 1. 定时调度执行job的服务器 |

# 总体设计

## 主要功能模块



## 版本迭代规划

|  |  |  |
| --- | --- | --- |
| 版本 | 需求目标 | 功能描述 |
| 一期 | 1. 通过job管理端能添加、修改job作业 2. Job管理端的leader负责为注册进来的客户端服务器分配任务项 3. 当在管理端新增作业或者job客户端扩容、缩容能触发job的重新分片 4. Job客户端应用启动时向注册中心注册，从zk中获取分配好的任务创建及启动执行 5. Zk降级方案：由job客户端从本地文件中获取job信息，创建并启动调度 6. Job管理端提供手动维护功能：job启动、停止、手动分片、手动执行、查看分片项、查看执行机以及每台执行机启动的job 7. Job管理端和job客户端在异常情况下进行报警人工介入处理 8. 提供应用接入管理功能 9. 重启客户端时，考虑zk挂了降级措施 10. 没有任务项的时候，是立即停止还是等执行完再停止 11. 降级关闭时，要创建所有机器上的job任务 12. 将删除的api替换一下 13. 是否考虑分片算法，打乱顺序 14. 全局考虑哪些地方应该加报警 15. 获取本机ip是否采用指定ip的方式 |  |
| 二期 | 1. Job管理端与job客户端建立tcp长连接进行实时网络通信 2. Job客户端系统通过 tcp长连接向job管理端周期性上报服务器的负载、上报job的执行轨迹 3. 由Job管理端全局控制每个分片项不会被客户端重复执行 4. Job管理端提供各种维护详细的监控：    1. Job执行超时的监控报警    2. Job长时间未启动的监控报警    3. 客户端负载各指标监控报警 5. Job管理端的负载均衡，当某台管理端服务器宕机后，job客户端自动切换到跟另一台管理端服务器通信，保证系统间的高可用 6. 针对生产环境使用情况，完善功能 |  |
| 三期愿景 | 1. 将项目代码抽取成单独的代码库，为开源做准备 2. 一起出一本书：“深入分布式worker” |  |

## 业务流程



## 功能实现

### Job管理平台模块

单独集群部署

#### 应用初始化

1. 负载zk结点的创建与事件监听的初始化
2. 连接zk，判断/worker/server目录是否存在，如果不存在则依次创建，使用zk监听/worker/server目录子结点的变化
3. 判断/worker/job目录是否存在，如果不存在则依次创建
4. 获取/worker/job所有子目录，判断/worker/job/${appName}/${job1}/sharding目录是否存在，如果存在该目录，则监听该目录数据的变化
5. 获取/worker/job所有子目录，判断/worker/job/${appName}/${job1}/execute目录是否存在，如果不存在，则创建该目录，设置初始数据
6. 获取/worker/job所有子目录，判断/worker/job/${appName}/${job1}/config目录是否存在，如果不存在，则创建该目录，设置初始数据
7. 获取/worker/job所有子目录，判断/worker/job/${appName}/${job1}/instances目录是否存在，如果不存在，则创建该目录
8. 初始化执行机回调的rpc接口服务
9. 上报服务器负载数据服务接口

**TODO**：待补充入参和出参

1. 上报job执行轨迹服务接口

**TODO**：待补充入参和出参

1. Leader选举

#### 页面功能

##### Job管理

1. 添加job

在job管理页面点击添加job，根据页面输入的job信息，创建zk目录/worker/job/${appName}/${jobName}，将job配置信息存储到${jobName}目录中，写入成功后，会触发zk事件通知，[参见添加job事件通知](#_接收到zk事件通知处理)，如果当前job分片开关打开，则设置需要重新分片标记

**注意：在添加某一个job的时候，要保证每一个job下面的四个目录都同时存在。同时监听job下面的sharding目录**

1. 更新job

可以修改除了Job名称的其它属性，当在页面修改job后，程序会修改/worker/job/${appName}/${jobName}目录的数据，会触发zk事件通知，[参见修改job事件通知](#_接收到zk事件通知处理)

注意：只允许修改分片数

1. Job查询

获取/worker/job目录所有子结点列表显示，并可以查看该job每台执行机是否启动

##### Job手动启动

点击某个job的启动按钮，写入/worker/job/${appName}/${job1}/execute目录数据

{cmd：startJob}

##### Job手动停止

点击某个job的停止按钮，写入/worker/job/${appName}/${job1}/execute目录数据

{cmd：stopJob}

##### job分片开关

描述：在页面上可以设置每个job的分片开关，如果开关关闭，则不允许执行分片动作，如果打开，则允许执行分片动作。避免job执行机上线发布的时候，频繁的触发重新分片动作，造成跟zk频繁交互带来的压力。

开关默认为“关闭”状态。

##### Job手动执行一次分片动作

描述：可能某个job还没有进行过分片，需要手动执行，直接修改/worker/job/${appName}/${job1}/shardin目录的分片状态

##### Job手动执行一次

描述：在页面上可以点击job手动执行，修改/worker/job/${appName}/${job1}/execute目录数据

{cmd：executeOnce}

##### Job分片标记查询

查询某个job的分片标记，读取/worker/job/${appName}/${job1}/sharding目录的数据，显示job的分片状态

##### Job分片项查询

查询某个job的所有分片项分布在哪些执行机中。读取/worker/job/${appName}/{job1}/instances目录所有的子结点，并显示出每一个子结点的分片项。

##### Job执行轨迹查询

1. 从数据库查询出某个job所有执行机的执行轨迹

TODO

##### Job执行机负载监控

1. 从数据库中查询出某个job所有执行机的内存及cpu负载情况

#### 接收到zk事件通知处理

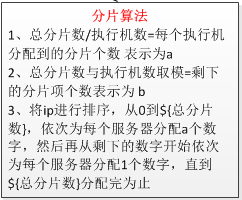
1. /worker/job/${appName}/${job1}/sharding目录数据变化的事件通知

描述：该目录数据存储的是 是否需要重新分片的标记

1. 以下三种情况，会修改目录的数据，系统会收到该事件通知
2. 在job分片开关打开的前提下，在管理平台添加了job，或者修改了job的总分片数
3. 在job分片开关打开的前提下，job执行机进行扩容或者缩容
4. 在管理平台点击某个job的“手动执行分片”按钮
5. 接收到事件通知后的业务逻辑处理

描述：由leader执行job的分片动作

* 根据内置的分片算法为当前注册的服务器分配好job的分片项，将分片项写入到/worker/job/${appName}/${job1}/instances/${ip1}/sharding目录中
* 分片算法描述：



1. /worker/server 目录子结点变化的事件通知

描述：

1. 以下两种情况系统会收到该事件通知
2. 当job执行机启动时，会将ip写入/worker/server/${appName}/${ip}目录
3. 当job执行机停止时，会自动将ip从/worker/server目录里删除。
4. 接收到事件通知后的业务逻辑
5. 如果为目录的添加事件：判断job分片开关是否打开，如果是，则设置所有的job需要重新分片
6. 如果为目录的删除事件：则清空该服务器对应的所有job的分片项和启动状态目录的数据，(/worker/job/${appName}/${job1}/instances/${ip1}/sharding) ( /worker/job/${appName}/${job1}/instances/${ip1}/state)，判断job分片开关是否打开，如果是，则设置所有的job需要重新分片

#### 远程回调服务

##### 上报服务器负载数据服务

将上报上来的服务器负载数据存储到数据库中，job执行机每30秒上报一次，

只存储最近一天的每个服务器的负载数据

##### 上报job执行日志服务

TODO

#### 监控统计功能

1. 执行job分片超时监控报警
2. 执行机负载超过一定阈值监控报警

### Job执行机模块

#### 应用初始化

应用启动执行业务逻辑如下

1. 将ip写入/worker/server/${appName}/${ip}目录
2. 初始化监听/worker/job目录子结点的变化
3. 获取/worker/job/目录所有子结点,循环监听每个job/worker/job/${appName}/${job}/sharding目录数据变化
4. 获取/worker/job/目录所有子结点,循环监听每个job /worker/job/${appName}/${job}/execute目录数据变化
5. 获取/worker/job/目录所有子结点，循环监听每个job /worker/job/${appName}/${job1}/instances/${ip1}/sharding目录数据的变化，并且获取该目录数据，如果分片项不为空，且job未创建，则创建job执行器对象

**注意：/worker/job/${appName}/${job1}/instances/${ip1}/sharding没有则创建这个操盘，会不会跟管理平台的这个操作发生冲突。**

1. 启动上报服务器负载数据的定时任务

#### 接收到zk事件通知处理

1. /worker/job目录子结点的变化事件通知

描述：该目录子结点存储的是所有job列表。

* 结点添加事件

获取当前事件新增的job及数据，创建quartz执行器对象

* 结点修改事件

根据job名称获取已经创建的quartz执行器对象，动态修改job配置

1. job/worker/job/${appName}/${job}/sharding目录数据变化事件通知

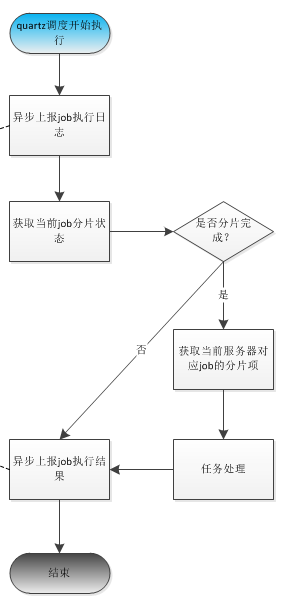
描述：获取该目录数据的分片状态，是否分片完成，如果是，并且当前job未启动，则启动当前job执行器

1. job/worker/job/${appName}/${job}/execute目录数据变化事件通知

描述：获取执行命令cmd，分以下三种情况

1. startJob：如果当前job未启动，则启动job执行器
2. stopJob：如果当前job已启动，则停止job执行器
3. executeOnce：获取当前job实例，通过反射调用目标方法一次

#### quartz自动调度job执行



## ZK注册中心数据结构

Zk的目录都是在/train这个命名空间下面

|  |  |  |  |
| --- | --- | --- | --- |
| 目录 | 目录描述 | Data描述 | 数据格式 |
| /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  } |
| /worker/job/${appName}/${job1}/sharding | 分片标记目录， | 存储分片的状态 | {  needSplit：true/false  splitState: unSplit/splited  } |
| /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  } |
| /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)  } |
| /worker/job/ ${appName}/${job1}/instances/${ip1}/sharding | Job对应的服务器的分片项目录 | 存储服务器对应的分片项。  Job执行机应该要监听这个目录 | 0,1 |
| /worker/job/${appName}/${job1}/instances/${ip1}/state | 服务器的job状态目录 | 存储该job是否启动 | {state：(started/stoped)} |
| /worker/server/${appName}/${ip1} | 存储的是注册进来的服务器ip |  |  |

## 网络请求操作类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 请求操作类型 | 描述 | 网络请求类型 | 入参 | 出参 |
| pushSystemLoad | 上报系统负载 | Oneway | {  appName:应用名  ip：客户端ip  cpu: cpu占比  memory:内存占比  threadCnt:总的线程数  jobList：[{jobjson},{jobjson}]创建的job列表  created：时间  } | 无 |
| pushJobStartExecute | 上报job开始执行轨迹 | async | {  appName:应用名  ip：客户端ip  jobName：job名称  sharding：任务项  startTime：时间  requestId：执行Id  } | {  Success：true  } |
| pushJobExecuteEnd | 上报job执行完成轨迹 | async | {  appName:应用名  ip：客户端ip  jobName：job名称  sharding：任务项  endTime：时间  requestId：执行Id  } | {  Success：true  } |

## 降级处理

### Zookeeper挂了

1. Job的分片项会缓存到本地内存中，在执行机保持运行未重启的情况下，已经启动的job会从本地读取分片项进行任务处理
2. Zk正常情况下，会异步的将job的基本信息存储到客户端本地文件当中，当zk挂了，手动在管理端打开”降级开关”，客户端系统会走降级流程：单台服务器跑所有的任务，先暂停当前所有执行的job，并清除，然后从本地文件当中获取所有job配置信息，并创建降级的作业以及启动调度作业。

## Maven工程目录结构

|  |  |  |
| --- | --- | --- |
| jt-job |  | 说明 |
|  | jt-job-core | 核心子模块 |
|  | jt-job-man | Job管理端模块 war包 |
|  | jt-job-client | Job客户端模块：jar包，嵌入到各业务系统运行 |

## 枚举值

|  |  |
| --- | --- |
| needSharding | true：表示需要重新分片  false:表示不需要重新分片 |
| shardingState | **unSharding**：未分片  **shardingEnd**：分片完成 |