主要功能

Elastic-job的主要包括以下功能。

1. 定时任务：基于成熟的定时任务作业框架Quartz cron表达式执行定时任务。
2. 作业注册中心：基于Zookeeper和其客户端Curator实现的全局作业注册控制中心。用于注册，控制和协调分布式作业执行。
3. 作业分片：将一个任务分片成为多个小任务项在多服务器上同时执行。
4. 弹性扩容缩容：运行中的作业服务器崩溃，或新增加n台作业服务器，作业框架将在下次作业执行前重新分片，不影响当前作业执行。
5. 支持多种作业执行模式：支持OneOff（类似Quartz原生作业），Perpetual（类似TbSchedule作业）和SequencePerpetual（类似TbSchedule和kafka的合并，但处理时维持分片顺序）三种作业模式。
6. 失效转移：运行中的作业服务器崩溃不会导致重新分片，只会在下次作业启动时分片。启用失效转移功能可以在本次作业执行过程中，监测其他作业服务器空闲，抓取未完成的孤儿分片项执行。
7. 运行时状态收集：监控作业运行时状态，统计最近一段时间处理的数据成功和失败数量，记录作业上次运行开始时间，结束时间和下次运行时间。
8. 作业停止，恢复和禁用：用于操作作业启停，并可以禁止某作业运行（上线时常用）。
9. 被错过执行的作业重触发：自动记录错过执行的作业，并在上次作业完成后自动触发。可参考Quartz的misfire。
10. 多线程快速处理数据：使用多线程处理抓取到的数据，提升吞吐量。
11. 幂等性：重复作业任务项判定，不重复执行已运行的作业任务项。由于开启幂等性需要监听作业运行状态，对瞬时反复运行的作业对性能有较大影响。
12. 容错处理：作业服务器与Zookeeper服务器通信失败则立即停止作业运行，防止作业注册中心将失效的分片分项配给其他作业服务器，而当前作业服务器仍在执行任务，导致重复执行。
13. Spring支持：支持spring容器，自定义命名空间，支持占位符。
14. 运维平台：提供运维界面，可以管理作业和注册中心。

开发指南

代码开发

提供3种作业类型，分别是OneOff, Perpetual和SequencePerpetual。需要继承相应的抽象类。

方法参数shardingContext包含作业配置，分片和运行时信息。可通过getShardingTotalCount(),getShardingItems()等方法分别获取分片总数，运行在本作业服务器的分片序列号集合等。

* **OneOff类型作业**

OneOff作业类型比较简单，需要继承AbstractOneOffElasticJob，该类只提供了一个方法用于覆盖，此方法将被定时执行。用于执行普通的定时任务，与Quartz原生接口相似，只是增加了弹性扩缩容和分片等功能。

public class MyElasticJob extends AbstractOneOffElasticJob {

@Override

protected void process(JobExecutionMultipleShardingContext context) {

// do something by sharding items

}

}

* **Perpetual类型作业**

Perpetual作业类型略为复杂，需要继承AbstractPerpetualElasticJob并可以指定返回值泛型，该类提供两个方法可覆盖，分别用于抓取和处理数据。可以获取数据处理成功失败次数等辅助监控信息。**需要注意fetchData方法的返回值只有为null或长度为空时，作业才会停止执行，否则作业会一直运行下去。**这点是参照TbSchedule的设计。Perpetual作业类型更适用于流式不间歇的数据处理。

作业执行时会将fetchData的数据传递给processData处理，其中processData得到的数据是通过多线程（线程池大小可配）拆分的。建议processData处理数据后，更新其状态，避免fetchData再次抓取到，从而使得作业永远不会停止。processData的返回值用于表示数据是否处理成功，抛出异常或者返回false将会在统计信息中归入失败次数，返回true则归入成功次数。

public class MyElasticJob extends AbstractPerpetualElasticJob<Foo> {

@Override

protected List<Foo> fetchData(JobExecutionMultipleShardingContext context) {

List<Foo> result = // get data from database by sharding items

return result;

}

@Override

protected boolean processData(JobExecutionMultipleShardingContext context, Foo data) {

// process data

return true;

}

}

* **SequencePerpetual类型作业**

SequencePerpetual作业类型和Perpetual作业类型极为相似，所不同的是Perpetual作业类型可以将获取到的数据多线程处理，但不会保证多线程处理数据的顺序。如：从2个分片共获取到100条数据，第1个分片40条，第2个分片60条，配置为两个线程处理，则第1个线程处理前50条数据，第2个线程处理后50条数据，无视分片项；SequencePerpetual类型作业则根据当前服务器所分配的分片项数量进行多线程处理，每个分片项使用同一线程处理，防止了同一分片的数据被多线程处理，从而导致的顺序问题。如：从2个分片共获取到100条数据，第1个分片40条，第2个分片60条，则系统自动分配两个线程处理，第1个线程处理第1个分片的40条数据，第2个线程处理第2个分片的60条数据。由于Perpetual作业可以使用多余分片项的任意线程数处理，所以性能调优的可能会优于SequencePerpetual作业。

public class MyElasticJob extends AbstractSequencePerpetualElasticJob<Foo> {

@Override

protected List<Foo> fetchData(JobExecutionSingleShardingContext context) {

List<Foo> result = // get data from database by sharding items

return result;

}

@Override

protected boolean processData(JobExecutionSingleShardingContext context, Foo data) {

// process data

return true;

}

}

作业配置

与Spring容器配合使用作业，可以将作业Bean配置为Spring Bean，可在作业中通过依赖注入使用Spring容器管理的数据源等对象。可用placeholder占位符从属性文件中取值。

* **Spring命名空间配置**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:reg="http://www.dangdang.com/schema/ddframe/reg"

xmlns:job="http://www.dangdang.com/schema/ddframe/job"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.dangdang.com/schema/ddframe/reg

http://www.dangdang.com/schema/ddframe/reg/reg.xsd

http://www.dangdang.com/schema/ddframe/job

http://www.dangdang.com/schema/ddframe/job/job.xsd

">

<!--配置作业注册中心 -->

<reg:zookeeper id="regCenter" serverLists=" yourhost:2181" namespace="dd-job" baseSleepTimeMilliseconds="1000" maxSleepTimeMilliseconds="3000" maxRetries="3" />

<!-- 配置作业A-->

<job:bean id="oneOffElasticJob" class="xxx.MyOneOffElasticJob" regCenter="regCenter" cron="0/10 \* \* \* \* ?" shardingTotalCount="3" shardingItemParameters="0=A,1=B,2=C" />

<!-- 配置作业B-->

<job:bean id="perpetualElasticJob" class="xxx.MyPerpetualElasticJob" regCenter="regCenter" cron="0/10 \* \* \* \* ?" shardingTotalCount="3" shardingItemParameters="0=A,1=B,2=C" processCountIntervalSeconds="10" concurrentDataProcessThreadCount="10" />

</beans>

**job:bean命名空间属性详细说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *属性名* | *类型* | *是否必填* | *缺省值* | *描述* |
| id | String | 是 |  | 作业名称 |
| class | String | 是 |  | 作业实现类，需实现ElasticJob接口 |
| regCenter | String | 是 |  | 注册中心Bean的引用，需引用reg:zookeeper的声明 |
| cron | String | 是 |  | cron表达式，用于配置作业触发时间 |
| shardingTotalCount | int | 是 |  | 作业分片总数 |
| shardingItemParameters | String | 否 |  | 分片序列号和个性化参数对照表 分片序列号和参数用等号分隔，多个键值对用逗号分隔 分片序列号从0开始，不可大于或等于作业分片总数 如： 0=a,1=b,2=c |
| jobParameter | String | 否 |  | 作业自定义参数 可以配置多个相同的作业，但是用不同的参数作为不同的调度实例 |
| monitorRuntime | boolean | 否 | true | 监控作业运行时状态 每次作业执行时间和间隔时间均非常短的情况，建议不监控作业运行时状态以提升效率。因为是瞬时状态，所以无必要监控。请用户自行增加数据堆积监控。并且不能保证数据重复选取，应在作业中实现幂等性。 每次作业执行时间和间隔时间均较长短的情况，建议监控作业运行时状态，可保证数据不会重复选取。 |
| processCountIntervalSeconds | int | 否 | 300 | 统计作业处理数据数量的间隔时间 单位：秒 对Perpetual和SequencePerpetual作业有效 |
| concurrentDataProcessThreadCount | int | 否 | 1 | 同时处理数据的并发线程数 不能小于1 仅Perpetual作业有效 |
| fetchDataCount | int | 否 | 0 | 每次抓取的数据量 |
| failover | boolean | 否 | false | 是否开启失效转移 仅monitorExecution开启，失效转移才有效 |
| description | String | 否 |  | 作业描述信息 |
| disabled | boolean | 否 | false | 作业是否禁止启动 可用于部署作业时，先禁止启动，部署结束后统一启动 |
| overwrite | boolean | 否 | false | 本地配置是否可覆盖注册中心配置 如果可覆盖，每次启动作业都以本地配置为准 |

**reg:zookeeper命名空间属性详细说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *属性名* | *类型* | *是否必填* | *缺省值* | *描述* |
| id | String | 是 |  | 注册中心在Spring容器中的主键 |
| serverLists | String | 是 |  | 连接Zookeeper服务器的列表 包括IP地址和端口号 多个地址用逗号分隔 如: host1:2181,host2:2181 |
| namespace | String | 是 |  | Zookeeper的命名空间 |
| baseSleepTimeMilliseconds | int | 是 |  | 等待重试的间隔时间的初始值 单位：毫秒 |
| maxSleepTimeMilliseconds | int | 是 |  | 等待重试的间隔时间的最大值 单位：毫秒 |
| maxRetries | int | 是 |  | 最大重试次数 |
| sessionTimeoutMilliseconds | int | 否 | 60000 | 会话超时时间 单位：毫秒 |
| connectionTimeoutMilliseconds | int | 否 | 15000 | 连接超时时间 单位：毫秒 |
| digest | String | 否 | 无权限验证 | 连接Zookeeper的权限令牌 缺省为不需要权限验证 |

* **基于Spring但不使用命名空间**

<!-- 配置作业注册中心 -->

<bean id="regCenter" class="com.dangdang.ddframe.reg.zookeeper.ZookeeperRegistryCenter" init-method="init">

<constructor-arg>

<bean class="com.dangdang.ddframe.reg.zookeeper.ZookeeperConfiguration">

<property name="serverLists" value="${xxx}" />

<property name="namespace" value="${xxx}" />

<property name="baseSleepTimeMilliseconds" value="${xxx}" />

<property name="maxSleepTimeMilliseconds" value="${xxx}" />

<property name="maxRetries" value="${xxx}" />

</bean>

</constructor-arg>

</bean>

<!-- 配置作业-->

<bean id="xxxJob" class="com.dangdang.ddframe.job.spring.schedule.SpringJobController" init-method="init">

<constructor-arg ref="regCenter" />

<constructor-arg>

<bean class="com.dangdang.ddframe.job.api.JobConfiguration">

<constructor-arg name="jobName" value="xxxJob" />

<constructor-arg name="jobClass" value="xxxDemoJob" />

<constructor-arg name="shardingTotalCount" value="10" />

<constructor-arg name="cron" value="0/10 \* \* \* \* ?" />

<property name="shardingItemParameters" value="${xxx}" />

</bean>

</constructor-arg>

</bean>

* **不使用Spring配置**

如果不使用Spring框架，可以用如下方式启动作业。

import com.dangdang.ddframe.job.api.JobConfiguration;

import com.dangdang.ddframe.job.schedule.JobController;

import com.dangdang.ddframe.reg.base.CoordinatorRegistryCenter;

import com.dangdang.ddframe.reg.zookeeper.ZookeeperConfiguration;

import com.dangdang.ddframe.reg.zookeeper.ZookeeperRegistryCenter;

import com.dangdang.example.elasticjob.core.job.OneOffElasticDemoJob;

import com.dangdang.example.elasticjob.core.job.PerpetualElasticDemoJob;

import com.dangdang.example.elasticjob.core.job.SequencePerpetualElasticDemoJob;

public class JobDemo {

// 定义Zookeeper注册中心配置对象

private ZookeeperConfiguration zkConfig = new ZookeeperConfiguration("localhost:2181", "elastic-job-example", 1000, 3000, 3);

// 定义Zookeeper注册中心

private CoordinatorRegistryCenter regCenter = new ZookeeperRegistryCenter(zkConfig);

// 定义作业1配置对象

private JobConfiguration jobConfig1 = new JobConfiguration("oneOffElasticDemoJob", OneOffElasticDemoJob.class, 10, "0/5 \* \* \* \* ?");

// 定义作业2配置对象

private JobConfiguration jobConfig2 = new JobConfiguration("perpetualElasticDemoJob", PerpetualElasticDemoJob.class, 10, "0/5 \* \* \* \* ?");

// 定义作业3配置对象

private JobConfiguration jobConfig3 = new JobConfiguration("sequencePerpetualElasticDemoJob", SequencePerpetualElasticDemoJob.class, 10, "0/5 \* \* \* \* ?");

public static void main(final String[] args) {

new JobDemo().init();

}

private void init() {

// 连接注册中心

regCenter.init();

// 启动作业1

new JobController(regCenter, jobConfig1).init();

// 启动作业2

new JobController(regCenter, jobConfig2).init();

// 启动作业3

new JobController(regCenter, jobConfig3).init();

}

}

使用限制

* 作业一旦启动成功后不能修改作业名称，如果修改名称则视为新的作业。
* 同一台作业服务器只能运行一个相同的作业实例，因为作业运行时是按照IP注册和管理的。
* 作业根据/etc/hosts文件获取IP地址，如果获取的IP地址是127.0.0.1而非真实IP地址，应正确配置此文件。
* 一旦有服务器波动，或者修改分片项，将会触发重新分片；触发重新分片将会导致运行中的Perpetual以及SequencePerpetual作业再执行完本次作业后不再继续执行，等待分片结束后再恢复正常。
* 开启monitorExecution才能实现分布式作业幂等性（即不会在多个作业服务器运行同一个分片）的功能，但monitorExecution对短时间内执行的作业（如每5秒一触发）性能影响较大，建议关闭并自行实现幂等性。
* elastic-job没有自动删除作业服务器的功能，因为无法区分是服务器崩溃还是正常下线。所以如果要下线服务器，需要手工删除zookeeper中相关的服务器节点。由于直接删除服务器节点风险较大，暂时不考虑在运维平台增加此功能。