Elastic-Job分布式任务调度

1.概述

1.1 什么是任务调度

我们可以先思考一下业务场景的解决方案：

- 某电商系统需要在每天上午10点，下午3点，晚上8点发放一批优惠券。
- 某银行系统需要在信用卡到期还款日的前三天进行短信提醒。
- 某财务系统需要在每天凌晨0:10结算前一天的财务数据，统计汇总。
- 12306会根据车次的不同，设置某几个时间点进行分批放票。
- 某网站为了实现天气实时展示，每隔10分钟就去天气服务器获取最新的实时天气信息。

以上业务场景的解决方案就是任务调度。

任务调度是指系统为了自动完成特定任务，在约定的特定时刻去执行任务的过程。有了任务调度即可解放更多的人力，而是由系统自动去执行任务。

如何实现任务调度？

- 多线程方式，结合sleep
- JDK提供的API，例如：Timer、ScheduledExecutor
- 框架，例如Quartz ，它是一个功能强大的任务调度框架，可以满足更多更复杂的调度需求

但是上述这些解决方案要么实现起来比较繁琐，要么不能满足分布式架构需求，我们需要更好的解决方案。

1.2 什么是分布式任务调度

当前软件的架构已经开始向分布式架构转变，将单体结构拆分为若干服务，服务之间通过网络交互来完成业务处理。在分布式架构下，一个服务往往会部署多个实例来运行我们的业务，如果在这种分布式系统环境下运行任务调度，我们称之为**分布式任务调度**。

将任务调度程序分布式构建，这样就可以具有分布式系统的特点，并且提高任务的调度处理能力：

1、并行任务调度

并行任务调度实现靠多线程，如果有大量任务需要调度，此时光靠多线程就会有瓶颈了，因为一台计算机CPU的处理能力是有限的。

如果将任务调度程序分布式部署，每个结点还可以部署为集群，这样就可以让多台计算机共同去完成任务调度，我们可以将任务分割为若干个分片，由不同的实例并行执行，来提高任务调度的处理效率。

2、高可用

若某一个实例宕机，不影响其他实例来执行任务。

3、弹性扩容

当集群中增加实例就可以提高并执行任务的处理效率。

4、任务管理与监测

对系统中存在的所有定时任务进行统一的管理及监测。让开发人员及运维人员能够时刻了解任务执行情况，从而做出快速的应急处理响应。

分布式任务调度面临的问题：

当任务调度以集群方式部署，同一个任务调度可能会执行多次，例如：电商系统定期发放优惠券，就可能重复发放优惠券，对公司造成损失，信用卡还款提醒就会重复执行多次，给用户造成烦恼，所以我们需要控制相同的任务在多个运行实例上只执行一次。常见解决方案：

- 分布式锁，多个实例在任务执行前首先需要获取锁，如果获取失败那么就证明有其他服务已经在运行，如果获取成功那么证明没有服务在运行定时任务，那么就可以执行。
- ZooKeeper选举，利用ZooKeeper对Leader实例执行定时任务，执行定时任务的时候判断自己是否是Leader，如果不是则不执行，如果是则执行业务逻辑，这样也能达到目的。

1.3 Elastic-Job简介

针对分布式任务调度的需求，市场上出现了很多的产品：

1） TBSchedule：淘宝推出的一款非常优秀的高性能分布式调度框架，目前被应用于阿里、京东、支付宝、国美等很多互联网企业的流程调度系统中。但是已经多年未更新，文档缺失严重，缺少维护。

2） XXL-Job：大众点评的分布式任务调度平台，是一个轻量级分布式任务调度平台，其核心设计目标是开发迅速、学习简单、轻量级、易扩展。现已开放源代码并接入多家公司线上产品线，开箱即用。

3）Elastic-job：当当网借鉴TBSchedule并基于quartz 二次开发的弹性分布式任务调度系统，功能丰富强大，采用zookeeper实现分布式协调，具有任务高可用以及分片功能。

4）Saturn： 唯品会开源的一个分布式任务调度平台，基于Elastic-job，可以全域统一配置，统一监控，具有任务高可用以及分片功能。

Elastic-Job是一个分布式调度的解决方案，由当当网开源，它由两个相互独立的子项目Elastic-Job-Lite和Elastic- Job-Cloud组成，使用Elastic-Job可以快速实现分布式任务调度。Elastic-Job的github地址：https://github.com/elasticjob。

功能列表：

分布式调度协调

在分布式环境中，任务能够按指定的调度策略执行，并且能够避免同一任务多实例重复执行。

丰富的调度策略：

基于成熟的定时任务作业框架Quartz cron表达式执行定时任务。

弹性扩容缩容

当集群中增加某一个实例，它应当也能够被选举并执行任务；当集群减少一个实例时，它所执行的任务能被转移到别的实例来执行。

失效转移

某实例在任务执行失败后，会被转移到其他实例执行。

错过执行作业重触发

若因某种原因导致作业错过执行，自动记录错过执行的作业，并在上次作业完成后自动触发。

支持并行调度

支持任务分片，任务分片是指将一个任务分为多个小任务项在多个实例同时执行。

支持作业生命周期操作

可以动态对任务进行开启及停止操作。

丰富的作业类型

支持Simple、DataFlow、Script三种作业类型。

Spring整合以及命名空间支持

对Spring支持良好的整合方式，支持spring自定义命名空间，支持占位符。

运维平台

提供运维界面，可以管理作业和注册中心。

1.4 重要概念

1. **分片**

   任务的分布式执行，需要将一个任务拆分为多个独立的任务项，然后由分布式的服务器分别执行某一个或几个分片项。例如：有一个遍历数据库某张表的作业，现有2台服务器。为了快速的执行作业，那么每台服务器应执行作业的50%。 为满足此需求，可将作业分成2片，每台服务器执行1片。作业遍历数据的逻辑应为：服务器A遍历ID以奇数结尾的数据；服务器B遍历ID以偶数结尾的数据。 如果分成10片，则作业遍历数据的逻辑应为：每片分到的分片项应为ID%10，而服务器A被分配到分片项0,1,2,3,4；服务器B被分配到分片项5,6,7,8,9，直接的结果就是服务器A遍历ID以0-4结尾的数据；服务器B遍历ID以5-9结尾的数据。

2. **leader选举**

   zookeeper会保证在多台服务器中选举出一个leader，leader如果下线会触发重新选举，在选出下个leader前所有任务会被阻塞， leader会以"协调者"角色负责分片。

3. **分片策略**

   AverageAllocationJobShardingStrategy

   全路径：

   com.dangdang.ddframe.job.lite.api.strategy.impl.AverageAllocationJobShardingStrategy

   策略说明：

   基于平均分配算法的分片策略，也是默认的分片策略。

   如果分片不能整除，则不能整除的多余分片将依次追加到序号小的服务器。如：

   如果有3台服务器，分成9片，则每台服务器分到的分片是：1=[0,1,2], 2=[3,4,5], 3=[6,7,8]

   如果有3台服务器，分成8片，则每台服务器分到的分片是：1=[0,1,6], 2=[2,3,7], 3=[4,5]

   如果有3台服务器，分成10片，则每台服务器分到的分片是：1=[0,1,2,9], 2=[3,4,5], 3=[6,7,8]

   OdevitySortByNameJobShardingStrategy

   全路径：

   com.dangdang.ddframe.job.lite.api.strategy.impl.OdevitySortByNameJobShardingStrategy

   策略说明：

   根据作业名的哈希值奇偶数决定IP升降序算法的分片策略。

   作业名的哈希值为奇数则IP升序。

   作业名的哈希值为偶数则IP降序。

   用于不同的作业平均分配负载至不同的服务器。

AverageAllocationJobShardingStrategy的缺点是，一旦分片数小于作业服务器数，作业将永远分配至IP地址靠前 的服务器，导致IP地址靠后的服务器空闲。而OdevitySortByNameJobShardingStrategy则可以根据作业名称重新 分配服务器负载。如：

如果有3台服务器，分成2片，作业名称的哈希值为奇数，则每台服务器分到的分片是：1=[0], 2=[1], 3=[]

如果有3台服务器，分成2片，作业名称的哈希值为偶数，则每台服务器分到的分片是：3=[0], 2=[1], 1=[]

RotateServerByNameJobShardingStrategy

全路径：com.dangdang.ddframe.job.lite.api.strategy.impl.RotateServerByNameJobShardingStrategy

策略说明：

根据作业名的哈希值对服务器列表进行轮转的分片策略。

4. **cron表达式**

cron表达式是一个字符串, 用来设置定时规则, 由七部分组成, 每部分中间用空格隔开, 每部分的含义如下表所示:

组成部分	含义	取值范围
第一部分	Seconds (秒)	0 - 59
第二部分	Minutes(分)	0 - 59
第三部分	Hours(时)	0-23
第四部分	Day-of-Month(天)	1-31
第五部分	Month(月)	0-11或JAN-DEC
第六部分	Day-of-Week(星期)	1-7(1表示星期日)或SUN-SAT
第七部分	Year(年) 可选	1970-2099

另外, cron表达式还可以包含一些特殊符号来设置更加灵活的定时规则, 如下表所示:

符号	含义
?	表示不确定的值。当两个子表达式其中一个被指定了值以后，为了避免冲突，需要将另外一个的值设为“?”。例如：想在每月20日触发调度，不管20号是星期几，只能用如下写法：0 0 0 20 * ?，其中最后以为只能用“?”
*	代表所有可能的值
,	设置多个值,例如”26,29,33”表示在26分,29分和33分各自运行一次任务
-	设置取值范围,例如”5-20”，表示从5分到20分钟每分钟运行一次任务
/	设置频率或间隔,如"1/15"表示从1分开始,每隔15分钟运行一次任务
L	用于每月，或每周，表示每月的最后一天，或每个月的最后星期几,例如"6L"表示"每月的最后一个星期五"
W	表示离给定日期最近的工作日,例如"15W"放在每月（day-of-month）上表示"离本月15日最近的工作日"
#	表示该月第几个周X。例如”6#3”表示该月第3个周五

为了让大家更熟悉cron表达式的用法, 接下来我们给大家列举了一些例子, 如下表所示:

cron表达式	含义
*/5 * * * * ?	每隔5秒运行一次任务
0 0 23 * * ?	每天23点运行一次任务
0 0 1 1 * ?	每月1号凌晨1点运行一次任务
0 0 23 L * ?	每月最后一天23点运行一次任务
0 26,29,33 * * * ?	在26分、29分、33分运行一次任务
0 0/30 9-17 * * ?	朝九晚五工作时间内每半小时运行一次任务
0 15 10 ? * 6#3	每月的第三个星期五上午10:15运行一次任务

2.Elastic-Job快速入门

在分布式架构下，通过Elastic-Job实现分片查询数据的定时任务

2.1 环境搭建

1. 版本要求

- JDK要求1.7及以上版本
- Maven要求3.0.4及以上版本
- zookeeper要求采用3.4.6及以上版本

2. Zookeeper安装&运行

   Zookeeper类似于Eureka，用来进行微服务注册和管理。

- https://archive.apache.org/dist/zookeeper/ 下载Zookeeper并解压
- 把conf目录下的zoo_sample.cfg改为zoo.cfg
- 运行bin目录下的zkServer.cmd

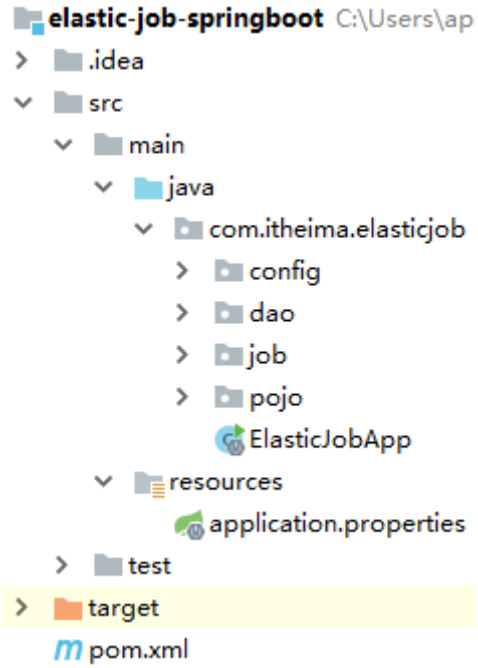
3. 数据库

使用mybatis-plus入门案例中的数据库mp，数据表为tb_user

id	user_name	password	name	age	email
1	zhangsan	123	张三	18	
2	lisi	123	李四	20	
3	wangwu	123	王五	28	
4	zhaoliu	123	赵六	21	
5	sunqi	123	孙七	22	
6	huba	123	胡八	20	

2.2 入门案例

1. 新建maven工程



2. 引入依赖

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
     <groupId>com.dangdang</groupId>
        <artifactId>elastic-job-lite-spring</artifactId>
        <version>2.1.5</version>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
```

```
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.1.1</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </dependency>
    </dependencies>
```

3. 编写springBoot配置文件及启动类 springBoot 配置文件：

```
server.port=${PORT:57081}
spring.application.name = elastic-job-springboot
logging.level.root = info

spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/mp?
useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=
true&useSSL=false
spring.datasource.username=root
spring.datasource.password=123

# 设置Mapper接口所对应的XML文件位置
mybatis-plus.mapper-locations = classpath*:dao/*.xml
# 设置别名包扫描路径
mybatis-plus.type-aliases-package = com.itheima.elasticjob.pojo

# zookeeper服务地址
zookeeper.connString = localhost:2181
# 名称空间
myjob.namespace = elastic-job-example
# 分片总数
myjob.count = 3
# cron表达式(定时策略)
myjob.cron = 0/5 * * * * ?
```

springBoot 启动类：

```
@MapperScan("com.itheima.elasticjob.dao")
@SpringBootApplication
public class ElasticJobApp {
    public static void main(String[] args) {
     SpringApplication.run(ElasticJobApp.class,args);
    }
}
```

4. 实体类

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@TableName("tb_user")
public class User {
    @TableId("ID")
    private Long id;
    @TableField("USER_NAME")
    private String userName;
    @TableField("PASSWORD")
    private String password;
    @TableField("NAME")
    private String name;
    @TableField("AGE")
    private Integer age;
}
```

5. 数据访问层

```
public interface UserMapper extends BaseMapper<User> {
    //对id通过mod函数取余进行分片
    @Select("SELECT *  FROM `tb_user` WHERE MOD(id,#{shardingTotalCount})=#
{shardingItem}")
    List<User> queryUserById(@Param("shardingTotalCount") int count,
@Param("shardingItem") int item);
 }
```

6. Elastic-Job任务类

```
/**
 * 数据查询任务
 **/
@Component
public class MyJob implements SimpleJob {

    @Autowired
    private UserMapper userMapper;

    //执行定时任务
    @Override
    public void execute(ShardingContext shardingContext) {
        //得到分片总数
        int count = shardingContext.getShardingTotalCount();
        //得到分片项
        int item = shardingContext.getShardingItem();
```

```
        //查询数据
        List<User> userList = userMapper.queryUserById(count,item);

        //输出结果
        userList.forEach(user -> {
            System.out.println("作业分片:"+item+"--->"+user);
        });
    }
}
```

7. zookeeper注册中心

```
@Configuration
public class ZKRegistryCenterConfig {

    //zookeeper服务地址
    @Value("${zookeeper.connString}")
    private  String ZOOKEEPER_CONNECTION_STRING ;

    //定时任务命名空间
    @Value("${myjob.namespace}")
    private  String JOB_NAMESPACE;

    //zk的配置及创建注册中心
    @Bean(initMethod = "init")
    public  ZookeeperRegistryCenter setUpRegistryCenter(){
        //zk的配置
        ZookeeperConfiguration zookeeperConfiguration = new
            ZookeeperConfiguration(ZOOKEEPER_CONNECTION_STRING,
JOB_NAMESPACE);
        //创建注册中心
        ZookeeperRegistryCenter zookeeperRegistryCenter = new
            ZookeeperRegistryCenter(zookeeperConfiguration);
        return zookeeperRegistryCenter;
    }
}
```

8. elastic-job配置类

```
@Configuration
public class ElasticJobConfig {

    @Autowired
    MyJob myJob;

    @Autowired
    ZookeeperRegistryCenter registryCenter;

    @Value("${myjob.count}")
    private int shardingCount;

    @Value("${myjob.cron}")
    private String cron;
```

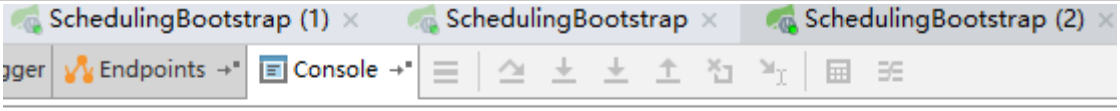
```
    /**
     * 配置任务详细信息
     * @param jobClass 任务执行类
     * @param cron  执行策略
     * @param shardingTotalCount 分片数量
     * @return
     */
    private LiteJobConfiguration createJobConfiguration(final Class<?
extends SimpleJob> jobClass,
                                                        final String cron,
                                                        final int
shardingTotalCount){
        //创建JobCoreConfigurationBuilder
        JobCoreConfiguration.Builder JobCoreConfigurationBuilder =
            JobCoreConfiguration.newBuilder(jobClass.getName(), cron,
shardingTotalCount);

        JobCoreConfiguration jobCoreConfiguration =
JobCoreConfigurationBuilder.build();
        //创建SimpleJobConfiguration
        SimpleJobConfiguration simpleJobConfiguration =
            new SimpleJobConfiguration(jobCoreConfiguration,
jobClass.getCanonicalName());
        //创建LiteJobConfiguration
        LiteJobConfiguration liteJobConfiguration =
LiteJobConfiguration.newBuilder(simpleJobConfiguration)
.jobShardingStrategyClass("com.dangdang.ddframe.job.lite.api.strategy.impl.A
verageAllocationJobShardingStrategy").overwrite(true)
                .build();
        return liteJobConfiguration;
    }

    @Bean(initMethod = "init")
    public SpringJobScheduler initSimpleElasticJob() {
        //创建SpringJobScheduler
        SpringJobScheduler springJobScheduler = new
SpringJobScheduler(myJob, registryCenter,
                createJobConfiguration(myJob.getClass(), cron,
shardingCount));
        return springJobScheduler;
    }
}
```

2.3 运行效果

由于在配置文件中设置的分片数量为3，所以这里启动了三个微服务，启动过程中会花费一些时间向zookeeper进行注册。运行起来后，我们会发现三个服务微每隔5秒钟就执行一次数据查询，并且进行了分片(平均分配)。

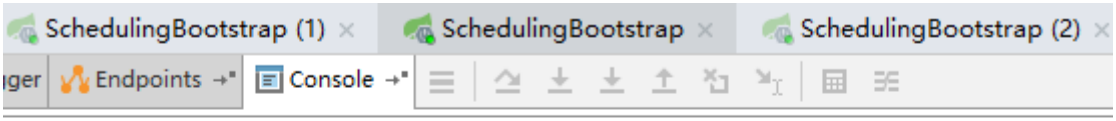


作业分片:0

User(id=3, userName=wangwu, password=123, name=王五, age=28)

作业分片:0

User(id=6, userName=huba, password=123, name=胡八, age=20)

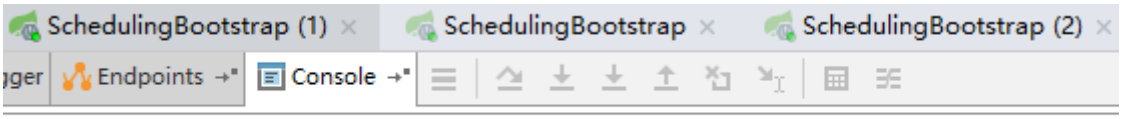


作业分片:1

User(id=1, userName=zhangsan, password=123, name=张三, age=18)

作业分片:1

User(id=4, userName=zhaoliu, password=123, name=赵六, age=21)



作业分片:2

User(id=2, userName=lisi, password=123, name=李四, age=20)

作业分片:2

User(id=5, userName=sunqi, password=123, name=孙七, age=22)