链滴

分布式任务调度框架技术选型

作者：2457081614

原文链接：https://ld246.com/article/1581302560239

来源网站：链滴

1.概述

目前常见的开源任务调度框架有 quartz、xxl-job、Elastic-job、Saturn 。

1.1 quartz

quartz 是一个完全由 Java 编写的开源作业调度框架，为在 Java 应用程序中进行作业调度提供了简却强大的机制，但是有很多的不足，如下：

- 调用 API 的方式不人性化，并且没有操作界面；
- 需要持久化业务 QuartzJobBean 到底层数据表中，系统侵入性相当严重；
- 调度逻辑和 QuartzJobBean 耦合在同一个项目中，这将导致一个问题，在调度任务数量逐渐增多同时调度任务逻辑逐渐加重的情况加，此时调度系统的性能将大大受限于业务；
- quartz 底层以“抢占式”获取 DB 锁并由抢占成功节点负责运行任务，会导致节点负载悬殊非常大而 XXL-JOB 通过执行器实现“协同分配式”运行任务，充分发挥集群优势，负载各节点均衡。

quartz 官网

1.2 xxl-job

XXL-JOB 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、易展。现已开放源代码并接入多家公司线上产品线，开箱即用。

1.3 Elastic-job

Elastic-Job 是一个分布式调度解决方案，由两个相互独立的子项目 Elastic-Job-Lite 和 Elastic-Job-Coud 组成。Elastic-Job-Lite 定位为轻量级无中心化解决方案，使用 jar 包的形式提供分布式任务的调服务；Elastic-Job-Cloud 采用自研 Mesos Framework 的解决方案，额外提供资源治理、应用分

以及进程隔离等功能。

1.4 Saturn

Saturn (任务调度系统)是唯品会开源的分布式作业调度平台，取代传统的 Linux Cron/Spring Batch J b 的方式，做到统一配置，统一监控，任务高可用以及分片并发处理。Saturn 基于当当 Elastic Job 码基础上自主研发的任务调度系统。

2.技术选型

由于 quartz 有以上的缺点，不考虑使用该框架，着重对 xxl-job、Elastic-job、Saturn 对比。

对比条件	xxl-job	Elastic-job	aturn
项目背景及社区力量	大众点评公司下员工许雪里、贡献者 40 人，GitHub 12.5K star,5.3K fork 数。社区活跃	当当网开源，贡献者 20 人，GitHub 有 5.6K star,2.6K fork 数。社区活跃	唯品会开源，贡献者 16 人，GitHub 有 1 7K star,565 fork 数，社区不活跃。
集群扩容	使用 Quartz 基于数据库的分布式功能，服务器超出一定数量会数据库造成一定的压力	通过 zookeeper 的注册与发现，可以动态的添加服器， 支持水平扩容。	通过 zk 实现服务的注册、协调及控制能支持容器化技进行 executor 扩容和减容，保证高峰期处理能力的弹性伸缩。
管理界面	支持	支持	支持
缺点	调度中心通过获取 DB 锁来保证集群中执行任务的唯一性， 如果短任很多，随着调度中心集群数量增加，那么数据库的锁竞争会比较厉害，性能不好。	要引入 zookeeper , mesos, 增加系统复杂度， 学习成本较高	与 Elastic-job 一致
失败处理策略	调度失败时的处理策略，策略包括：失败告警（默认）、失重试（界面可配置）	弹性扩容缩容在下次作业运行前重分片，但本次作业执的过程中，下线的服务器所分配的作业将不会重新被分配。失效转移功能可以在本次作业运行中用空服务器抓取孤儿作业分片执行。同样失效转移功能也会牺牲部分性能。	支持常检测和失败转移，超时报警，超时强杀
分片策略	分片广播任务以执行器为维度进行分片，支持动态扩容执行器集从而动态增加分片数量，协同进行业务处理；在进行大数据量业务操作时可显著提升任务处理能力和度。 执行器集群部署时，任务路由策略选择”分片广播”情况下，一次任务调度将会广播触发对应集中所有执行器执行一次任务，同时传递分片参数；可根据分片参数开发分片任务;	持多种分片策略，可自定义分片策略。 默认包含三种分片策略： 基于平均分配算法的分片策略、 作名的哈希值奇偶数决定 IP 升降序算法的分片策略、根据作业名的哈希值对 Job 实例列表进行轮转的片策略，支持自定义分片策略。elastic-job 的分片是通过 zookeeper 来实现的。分片的分片由主节分配，如下三种情况都会触发主节点上的分片算法执行：a、新的 Job 实例加入集群 b、现有的 Job 例下线（如果下线的是 leader 节点，那么先选举然后触发分片算法的执行）c、主节点选举	工指定资源分配策略 + 自动平均策略结合
依赖	MySQL ,jdk1.7+ , maven3.0+	jdk1.7+, zoo eeper 3.4.6+ ,maven3.0.4+ ,mesos	Java 7+,Maven 3.0.4+,node.js 8.7.0+ npm 5.4.2+,git
容器部署	支持		支持
文档	非常详细	详细	一般
触发规则	时间、事件触发	时间触发	

间、事件触发

调度器集群部署	支持	支持	支持
作业集群部署	支持	支持	支持
日志追溯	支持，界面查询	通过事件订阅的方式处	调度过程中的重要事件，用于查询统计和监控
报警	默认有邮件报警，其他报警方式已预留接口，如钉钉		己实现
阻塞处理策略	单机串行、丢弃后续调度、覆盖之前的调度		
高可用	任务路由策略选择"故障转移"情况下，如果执行器集群中某一台机		故障，将会自动 Failover 切换到一台正常的执行器发送调度请求。

运行几指向同一个 zk 集群的 Elastic-Job-Cloud-Schedule 实例实现的。zk 用于在当前 Elastic-Job-Cloud-chedule 实例失败的情况下执行领导者选举。通过至少两个调度器实例来构成集群，集群中只有一个度器实例提供服务，其他实例处于待命状态。当该实例失败时，集群会选举剩余实例中的一个来继续供服务 通过 zk 保证集群分布式调度，Saturn 是面向任务的，能够监控到 executor 的状态，在 executor 下线或者上线时，均会对任务分片进行重分配，保证可用性。

在上面三个框架柱，Saturn 相关文档比较少，社区相对来说也不活跃，迭代更新也比较慢，因此不虑使用。E-Job 和 X-job 都有广泛的用户基础和完整的技术文档，都能满足定时任务的基本功能需求 xxl-job 侧重业务实现的简单和管理的方便，学习成本简单，失败策略和路由策略丰富。推荐使用在户基数相对少，服务器数量在一定范围内的情景下使用。elastic-job 关注的是数据，增加了弹性扩容数据分片的思路，以便于更大限度的利用分布式服务器的资源。但是学习成本相对较高，也更加复杂一般在数据量庞大，且部署服务器数量较多时使用，结合目前的业务场景，使用 xxl-job 框架。

3.使用

3.1 调度中心安装

见官网文档

3.2 执行器相关配置

3.2.1 添加xxl-job依赖

```
<!-- https://mvnrepository.com/artifact/com.xuxueli/xxl-job-core -->
    <dependency>
       <groupId>com.xuxueli</groupId>
       <artifactId>xxl-job-core</artifactId>
       <version>2.1.2</version>
    </dependency>
```

3.2.2 properties配置文件

```
### 调度中心部署跟地址 [选填]：如调度中心集群部署存在多个地址则用逗号分隔。执行器将会使用地址进行"执行器心跳注册"和"任务结果回调"；为空则关闭自动注册；
```

```
xxl.job.admin.addresses=http://127.0.0.1:8080/xxl-job-admin
### 执行器AppName [选填]：执行器心跳注册分组依据；为空则关闭自动注册
xxl.job.executor.appname=xxl-job-executor-sample
### 执行器IP [选填]：默认为空表示自动获取IP，多网卡时可手动设置指定IP，该IP不会绑定Host仅为通讯实用；地址信息用于 "执行器注册" 和 "调度中心请求并触发任务"；
xxl.job.executor.ip=
### 执行器端口号 [选填]：小于等于0则自动获取；默认端口为9999，单机部署多个执行器时，注意配置不同执行器端口；
xxl.job.executor.port=9999
### 执行器通讯TOKEN [选填]：非空时启用；
xxl.job.accessToken=
### 执行器运行日志文件存储磁盘路径 [选填] ：需要对该路径拥有读写权限；为空则使用默认路径；
xxl.job.executor.logpath=/data/applogs/xxl-job/jobhandler
### 执行器日志文件保存天数 [选填] ： 过期日志自动清理, 限制值大于等于3时生效; 否则, 如-1, 关自动清理功能；
xxl.job.executor.logretentiondays=30
```

3.2.3 yaml配置

```
xxl:
  job:
    accessToken: ''
    admin:
      addresses: http://127.0.0.1:8080/xxl-job-admin
    executor:
      appname: xxl-job-executor-sample
      ip: ''
      logpath: /data/applogs/xxl-job/jobhandler
      logretentiondays: 30
      port: 9999
```

3.2.4 config配置

```
package com.xxl.job.executor.core.config;

import com.xxl.job.core.executor.impl.XxlJobSpringExecutor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * xxl-job config
 *
 */
@Configuration
public class XxlJobConfig {
    private Logger logger = LoggerFactory.getLogger(XxlJobConfig.class);

    @Value("${xxl.job.admin.addresses}")
    private String adminAddresses;
```

```
    @Value("${xxl.job.executor.appname}")
    private String appName;

    @Value("${xxl.job.executor.ip}")
    private String ip;

    @Value("${xxl.job.executor.port}")
    private int port;

    @Value("${xxl.job.accessToken}")
    private String accessToken;

    @Value("${xxl.job.executor.logpath}")
    private String logPath;

    @Value("${xxl.job.executor.logretentiondays}")
    private int logRetentionDays;


    @Bean
    public XxlJobSpringExecutor xxlJobExecutor() {
        logger.info(">>>>>>>>>>> xxl-job config init.");
        XxlJobSpringExecutor xxlJobSpringExecutor = new XxlJobSpringExecutor();
        xxlJobSpringExecutor.setAdminAddresses(adminAddresses);
        xxlJobSpringExecutor.setAppName(appName);
        xxlJobSpringExecutor.setIp(ip);
        xxlJobSpringExecutor.setPort(port);
        xxlJobSpringExecutor.setAccessToken(accessToken);
        xxlJobSpringExecutor.setLogPath(logPath);
        xxlJobSpringExecutor.setLogRetentionDays(logRetentionDays);

        return xxlJobSpringExecutor;
    }

    /**
     * 针对多网卡、容器内部署等情况，可借助 "spring-cloud-commons" 提供的 "InetUtils" 组件灵
定制注册IP；
     *
     *      1、引入依赖：
     *          <dependency>
     *            <groupId>org.springframework.cloud</groupId>
     *            <artifactId>spring-cloud-commons</artifactId>
     *            <version>${version}</version>
     *        </dependency>
     *
     *      2、配置文件，或者容器启动变量
     *          spring.cloud.inetutils.preferred-networks: 'xxx.xxx.xxx.'
     *
     *      3、获取IP
     *          String ip_ = inetUtils.findFirstNonLoopbackHostInfo().getIpAddress();
     */


}
```

3.2.5 添加定时任务

1. 编码，下面是 demo

```
package com.xxl.job.executor.service.jobhandler;

import com.xxl.job.core.biz.model.ReturnT;
import com.xxl.job.core.handler.IJobHandler;
import com.xxl.job.core.handler.annotation.XxlJob;
import com.xxl.job.core.log.XxlJobLogger;
import com.xxl.job.core.util.ShardingUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.concurrent.TimeUnit;

/**
 * XxlJob开发示例（Bean模式）
 *
 * 开发步骤：
 * 1、在Spring Bean实例中，开发Job方法，方式格式要求为 "public ReturnT<String> execute(Str
ng param)"
 * 2、为Job方法添加注解 "@XxlJob(value="自定义jobhandler名称", init = "JobHandler初始化方
", destroy = "JobHandler销毁方法")"，注解value值对应的是调度中心新建任务的JobHandler属性
值。
 * 3、执行日志：需要通过 "XxlJobLogger.log" 打印执行日志；
 *
 * @author xuxueli 2019-12-11 21:52:51
 */
@Component
public class SampleXxlJob {
    private static Logger logger = LoggerFactory.getLogger(SampleXxlJob.class);


    /**
     * 1、简单任务示例（Bean模式）
     */
    @XxlJob("demoJobHandler")
    public ReturnT<String> demoJobHandler(String param) throws Exception {
        XxlJobLogger.log("XXL-JOB, Hello World.");
        System.out.println("hello world");
        for (int i = 0; i < 5; i++) {
            XxlJobLogger.log("beat at:" + i);
            TimeUnit.SECONDS.sleep(2);
        }
        return ReturnT.SUCCESS;
    }
```

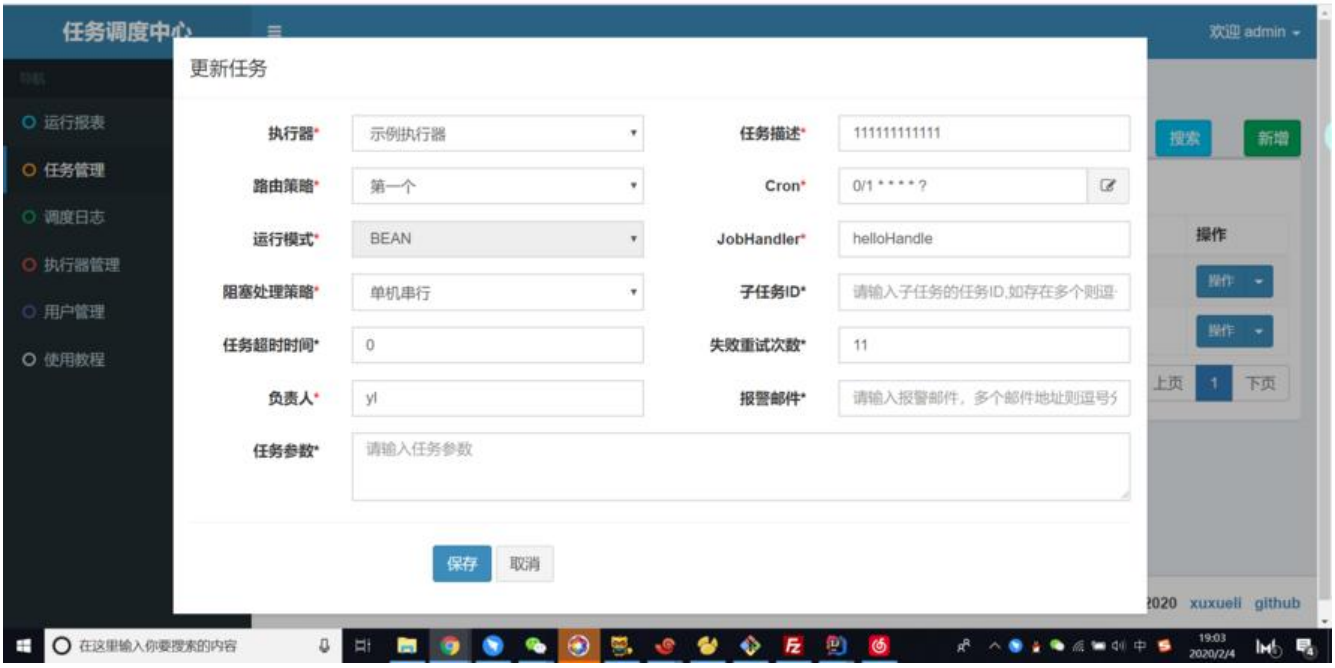
```
    /**
     * 2、分片广播任务
     */
    @XxlJob("shardingJobHandler")
    public ReturnT<String> shardingJobHandler(String param) throws Exception {

        // 分片参数
        ShardingUtil.ShardingVO shardingVO = ShardingUtil.getShardingVo();
        XxlJobLogger.log("分片参数：当前分片序号 = {}, 总分片数 = {}", shardingVO.getIndex(), sha
dingVO.getTotal());

        // 业务逻辑
        for (int i = 0; i < shardingVO.getTotal(); i++) {
            if (i == shardingVO.getIndex()) {
                XxlJobLogger.log("第 {} 片, 命中分片开始处理", i);
            } else {
                XxlJobLogger.log("第 {} 片, 忽略", i);
            }
        }

        return ReturnT.SUCCESS;
    }


    /**
     * 3、命令行任务
     */
    @XxlJob("commandJobHandler")
    public ReturnT<String> commandJobHandler(String param) throws Exception {
        String command = param;
        int exitValue = -1;

        BufferedReader bufferedReader = null;
        try {
            // command process
            Process process = Runtime.getRuntime().exec(command);
            BufferedInputStream bufferedInputStream = new BufferedInputStream(process.getIn
utStream());
            bufferedReader = new BufferedReader(new InputStreamReader(bufferedInputStream))


            // command log
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                XxlJobLogger.log(line);
            }

            // command exit
            process.waitFor();
            exitValue = process.exitValue();
        } catch (Exception e) {
            XxlJobLogger.log(e);
        } finally {
```

```
            if (bufferedReader != null) {
                bufferedReader.close();
            }
        }

        if (exitValue == 0) {
            return IJobHandler.SUCCESS;
        } else {
            return new ReturnT<String>(IJobHandler.FAIL.getCode(), "command exit value("+exit
alue+") is failed");
        }
    }


    /**
     * 4、跨平台Http任务
     */
    @XxlJob("httpJobHandler")
    public ReturnT<String> httpJobHandler(String param) throws Exception {

        // request
        HttpURLConnection connection = null;
        BufferedReader bufferedReader = null;
        try {
            // connection
            URL realUrl = new URL(param);
            connection = (HttpURLConnection) realUrl.openConnection();

            // connection setting
            connection.setRequestMethod("GET");
            connection.setDoOutput(true);
            connection.setDoInput(true);
            connection.setUseCaches(false);
            connection.setReadTimeout(5 * 1000);
            connection.setConnectTimeout(3 * 1000);
            connection.setRequestProperty("connection", "Keep-Alive");
            connection.setRequestProperty("Content-Type", "application/json;charset=UTF-8");
            connection.setRequestProperty("Accept-Charset", "application/json;charset=UTF-8");

            // do connection
            connection.connect();

            //Map<String, List<String>> map = connection.getHeaderFields();

            // valid StatusCode
            int statusCode = connection.getResponseCode();
            if (statusCode != 200) {
                throw new RuntimeException("Http Request StatusCode(" + statusCode + ") Invalid.
");
            }

            // result
            bufferedReader = new BufferedReader(new InputStreamReader(connection.getInputSt
eam(), "UTF-8"));
```

```
            StringBuilder result = new StringBuilder();
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                result.append(line);
            }
            String responseMsg = result.toString();

            XxlJobLogger.log(responseMsg);
            return ReturnT.SUCCESS;
        } catch (Exception e) {
            XxlJobLogger.log(e);
            return ReturnT.FAIL;
        } finally {
            try {
                if (bufferedReader != null) {
                    bufferedReader.close();
                }
                if (connection != null) {
                    connection.disconnect();
                }
            } catch (Exception e2) {
                XxlJobLogger.log(e2);
            }
        }

    }

    /**
     * 5、生命周期任务示例：任务初始化与销毁时，支持自定义相关逻辑；
     */
    @XxlJob(value = "demoJobHandler2", init = "init", destroy = "destroy")
    public ReturnT<String> demoJobHandler2(String param) throws Exception {
        XxlJobLogger.log("XXL-JOB, Hello World.");
        return ReturnT.SUCCESS;
    }
    public void init(){
        logger.info("init");
    }
    public void destroy(){
        logger.info("destory");
    }


}
```

2.在调度中心添加任务

注意 JobHandler 要与上步骤指定的名称一致。



更多相关功能请参阅官方文档。

4.附录

xxl-job 官网文档

xxl-job GitHub 地址

elastic-job 官网

elastic-job GitHub 地址

Saturn GitHub 地址