**分布式任务调度平台**

# 什么是定时任务

指定时间去执行任务

# Java实现定时任务方式

### Thread

|  |
| --- |
| **public class Demo01 {**  **static long *count* = 0;**  **public static void main(String[] args) {**  **Runnable runnable = new Runnable() {**  **@Override**  **public void run() {**  **while (true) {**  **try {**  **Thread.*sleep*(1000);**  ***count*++;**  **System.*out*.println(*count*);**  **} catch (Exception e) {**  **// TODO: handle exception**  **}**  **}**  **}**  **};**  **Thread thread = new Thread(runnable);**  **thread.start();**  **}**  **}** |

### TimerTask

|  |
| --- |
| **/\*\***  **\* 使用TimerTask类实现定时任务**  **\*/**  **public class Demo02 {**  **static long *count* = 0;**  **public static void main(String[] args) {**  **TimerTask timerTask = new TimerTask() {**  **@Override**  **public void run() {**  ***count*++;**  **System.*out*.println(*count*);**  **}**  **};**  **Timer timer = new Timer();**  **// 天数**  **long delay = 0;**  **// 秒数**  **long period = 1000;**  **timer.scheduleAtFixedRate(timerTask, delay, period);**  **}**  **}** |

### ScheduledExecutorService

使用ScheduledExecutorService是从Java

JavaSE5的java.util.concurrent里，做为并发工具类被引进的，这是最理想的定时任务实现方式。

|  |
| --- |
| **public class Demo003 {**  **public static void main(String[] args) {**  **Runnable runnable = new Runnable() {**  **public void run() {**  **// task to run goes here**  **System.*out*.println("Hello !!");**  **}**  **};**  **ScheduledExecutorService service = Executors.*newSingleThreadScheduledExecutor*();**  **// 第二个参数为首次执行的延时时间，第三个参数为定时执行的间隔时间**  **service.scheduleAtFixedRate(runnable, 1, 1, TimeUnit.*SECONDS*);**  **}**  **}** |

### Quartz

#### 创建一个quartz\_demo项目

#### 引入maven依赖

|  |
| --- |
| **<dependencies>**  **<!-- quartz -->**  **<dependency>**  **<groupId>org.quartz-scheduler</groupId>**  **<artifactId>quartz</artifactId>**  **<version>2.2.1</version>**  **</dependency>**  **<dependency>**  **<groupId>org.quartz-scheduler</groupId>**  **<artifactId>quartz-jobs</artifactId>**  **<version>2.2.1</version>**  **</dependency>**  **</dependencies>** |

#### 任务调度类

|  |
| --- |
| **public class MyJob implements Job {**  **public void execute(JobExecutionContext context) throws JobExecutionException {**  **System.*out*.println("quartz MyJob date:" + new Date().getTime());**  **}**  **}** |

#### 启动类

|  |
| --- |
| **//1.创建Scheduler的工厂**  **SchedulerFactory sf = new StdSchedulerFactory();**  **//2.从工厂中获取调度器实例**  **Scheduler scheduler = sf.getScheduler();**  **//3.创建JobDetail**  **JobDetail jb = JobBuilder.*newJob*(MyJob.class)**  **.withDescription("this is a ram job") //job的描述**  **.withIdentity("ramJob", "ramGroup") //job 的name和group**  **.build();**  **//任务运行的时间，SimpleSchedle类型触发器有效**  **long time= System.*currentTimeMillis*() + 3\*1000L; //3秒后启动任务**  **Date statTime = new Date(time);**  **//4.创建Trigger**  **//使用SimpleScheduleBuilder或者CronScheduleBuilder**  **Trigger t = TriggerBuilder.*newTrigger*()**  **.withDescription("")**  **.withIdentity("ramTrigger", "ramTriggerGroup")**  **//.withSchedule(SimpleScheduleBuilder.simpleSchedule())**  **.startAt(statTime) //默认当前时间启动**  **.withSchedule(CronScheduleBuilder.*cronSchedule*("0/2 \* \* \* \* ?")) //两秒执行一次**  **.build();**  **//5.注册任务和定时器**  **scheduler.scheduleJob(jb, t);**  **//6.启动 调度器**  **scheduler.start();** |

#### Quartz表达式

http://cron.qqe2.com/

# 分布式情况下定时任务会出现哪些问题？

分布式集群的情况下，怎么保证定时任务不被重复执行

## 分布式定时任务解决方案

①使用zookeeper实现分布式锁 缺点(需要创建临时节点、和事件通知不易于扩展)

②使用配置文件做一个开关 缺点发布后，需要重启

③数据库唯一约束，缺点效率低

④使用分布式任务调度平台XXLJOB

# XXLJOB介绍

1、简单：支持通过Web页面对任务进行CRUD操作，操作简单，一分钟上手；

2、动态：支持动态修改任务状态、暂停/恢复任务，以及终止运行中任务，即时生效；

3、调度中心HA（中心式）：调度采用中心式设计，“调度中心”基于集群Quartz实现，可保证调度中心HA；

4、执行器HA（分布式）：任务分布式执行，任务"执行器"支持集群部署，可保证任务执行HA；

5、任务Failover：执行器集群部署时，任务路由策略选择"故障转移"情况下调度失败时将会平滑切换执行器进行Failover；

6、一致性：“调度中心”通过DB锁保证集群分布式调度的一致性, 一次任务调度只会触发一次执行；

7、自定义任务参数：支持在线配置调度任务入参，即时生效；

8、调度线程池：调度系统多线程触发调度运行，确保调度精确执行，不被堵塞；

9、弹性扩容缩容：一旦有新执行器机器上线或者下线，下次调度时将会重新分配任务；

10、邮件报警：任务失败时支持邮件报警，支持配置多邮件地址群发报警邮件；

11、状态监控：支持实时监控任务进度；

12、Rolling执行日志：支持在线查看调度结果，并且支持以Rolling方式实时查看执行器输出的完整的执行日志；

13、GLUE：提供Web IDE，支持在线开发任务逻辑代码，动态发布，实时编译生效，省略部署上线的过程。支持30个版本的历史版本回溯。

14、数据加密：调度中心和执行器之间的通讯进行数据加密，提升调度信息安全性；

15、任务依赖：支持配置子任务依赖，当父任务执行结束且执行成功后将会主动触发一次子任务的执行, 多个子任务用逗号分隔；

16、推送maven中央仓库: 将会把最新稳定版推送到maven中央仓库, 方便用户接入和使用;

17、任务注册: 执行器会周期性自动注册任务, 调度中心将会自动发现注册的任务并触发执行。同时，也支持手动录入执行器地址；

18、路由策略：执行器集群部署时提供丰富的路由策略，包括：第一个、最后一个、轮询、随机、一致性HASH、最不经常使用、最近最久未使用、故障转移、忙碌转移等；

19、运行报表：支持实时查看运行数据，如任务数量、调度次数、执行器数量等；以及调度报表，如调度日期分布图，调度成功分布图等；

20、脚本任务：支持以GLUE模式开发和运行脚本任务，包括Shell、Python等类型脚本;

21、阻塞处理策略：调度过于密集执行器来不及处理时的处理策略，策略包括：单机串行（默认）、丢弃后续调度、覆盖之前调度；

22、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）、失败重试；

23、分片广播任务：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群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；

24、动态分片：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群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速度。

25、事件触发：除了"Cron方式"和"任务依赖方式"触发任务执行之外，支持基于事件的触发任务方式。调度中心提供触发任务单次执行的API服务，可根据业务事件灵活触发。

## XXLJOBGitHub地址

https://github.com/xuxueli/xxl-job

## 文档

<http://www.xuxueli.com/xxl-job/#/?id=_21-%e5%88%9d%e5%a7%8b%e5%8c%96%e8%b0%83%e5%ba%a6%e6%95%b0%e6%8d%ae%e5%ba%93>

## 步骤:

1. 部署: xxl-job-admin 作为注册中心
2. 创建执行器（具体调度地址） 可以支持集群
3. 配置文件需要填写xxl-job注册中心地址
4. 每个具体执行job服务器需要创建一个netty连接端口号
5. 需要执行job的任务类，集成IJobHandler抽象类注册到job容器中
6. Execute方法中编写具体job任务

## 客户端对接

### 引入依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.6.RELEASE</version>  </parent>  <dependencies>  <!-- jetty -->  <dependency>  <groupId>org.eclipse.jetty</groupId>  <artifactId>jetty-server</artifactId>  </dependency>  <dependency>  <groupId>org.eclipse.jetty</groupId>  <artifactId>jetty-util</artifactId>  </dependency>  <dependency>  <groupId>org.eclipse.jetty</groupId>  <artifactId>jetty-http</artifactId>  </dependency>  <dependency>  <groupId>org.eclipse.jetty</groupId>  <artifactId>jetty-io</artifactId>  </dependency>  <!-- spring-boot-starter-web (spring-webmvc + tomcat) -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <!-- xxl-job-core -->  <dependency>  <groupId>com.xuxueli</groupId>  <artifactId>xxl-job-core</artifactId>  <version>1.9.0-SNAPSHOT</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin </artifactId>  </plugin>  </plugins>  </build> |

### 配置文件

|  |
| --- |
| # web port  server.port=8084  # log config  logging.config=classpath:logback.xml  # xxl-job  ### xxl-job admin address list, such as "http://address" or "http://address01,http://address02"  xxl.job.admin.addresses=http://127.0.0.1:8081/xxl-job-admin  ### xxl-job executor address  xxl.job.executor.appname=test  xxl.job.executor.ip=192.168.1.3  xxl.job.executor.port=9999  ### xxl-job log path  xxl.job.executor.logpath=/data/applogs/xxl-job/jobhandler/  ### xxl-job, access token  xxl.job.accessToken= |

### Java代码

|  |
| --- |
| /\*\*  \* xxl-job config  \*  \* **@author** xuxueli 2017-04-28  \*/  @Configuration  @ComponentScan(basePackages = "com.xxl.job.executor.service.jobhandler")  **public** **class** XxlJobConfig {  **private** Logger logger = LoggerFactory.*getLogger*(XxlJobConfig.**class**);  @Value("${xxl.job.admin.addresses}")  **private** String addresses;  @Value("${xxl.job.executor.appname}")  **private** String appname;  @Value("${xxl.job.executor.ip}")  **private** String ip;  @Value("${xxl.job.executor.port}")  **private** **int** port;  @Value("${xxl.job.executor.logpath}")  **private** String logpath;  @Value("${xxl.job.accessToken}")  **private** String accessToken;  @Bean(initMethod = "start", destroyMethod = "destroy")  **public** XxlJobExecutor xxlJobExecutor() {  logger.info(">>>>>>>>>>> xxl-job config init.");  XxlJobExecutor xxlJobExecutor = **new** XxlJobExecutor();  xxlJobExecutor.setIp(ip);  xxlJobExecutor.setPort(port);  xxlJobExecutor.setAppName(appname);  xxlJobExecutor.setAdminAddresses(addresses);  xxlJobExecutor.setLogPath(logpath);  xxlJobExecutor.setAccessToken(accessToken);  **return** xxlJobExecutor;  }  } |

|  |
| --- |
| **/\*\***  **\* 任务Handler的一个Demo（Bean模式）**  **\***  **\* 开发步骤： 1、继承 “IJobHandler” ； 2、装配到Spring，例如加 “@Service” 注解； 3、加 “@JobHander”**  **\* 注解，注解value值为新增任务生成的JobKey的值;多个JobKey用逗号分割; 4、执行日志：需要通过 "XxlJobLogger.log"**  **\* 打印执行日志；**  **\***  **\* @author xuxueli 2015-12-19 19:43:36**  **\*/**  **@JobHander(value = "demoJobHandler")**  **@Service**  **public class DemoJobHandler extends IJobHandler {**  **@Value("${xxl.job.executor.port}")**  **private String port;**  **@Override**  **public ReturnT<String> execute(String... params) throws Exception {**  **XxlJobLogger.*log*("XXL-JOB, Hello World." + port);**  **System.*out*.println("XXL-JOB, Hello World." + port);**  **for (int i = 0; i < 5; i++) {**  **XxlJobLogger.*log*("beat at:" + i);**  **// TimeUnit.SECONDS.sleep(2);**  **}**  **return ReturnT.*SUCCESS*;**  **}**  **}** |

# XXLJOB路由策略

