

目录

目录

一、Quartz-定时器入门

- 1、Quartz 版本
- 2、创建项目
- 3、导入jar包
- 4、创建TestQuartz类
- 5、定义调度器
- 6、定义一个触发器
- 7、定义一个JobDetail任务
- 8、指定干活的类
- 9、调度加入这个job任务
- 10、启动
- 11、线程休眠
- 12、创建log4j文件
- 13、运行

二、Quartz-job管理

- 1、Job 组成部分
- 2、Job 并发
- 3、job异常
- 4、中断 job

三、quartz simpleTrigger

- 1、创建调度器
- 2、创建job
- 3、10 秒后运行
- 4、累计n次，间隔n秒
- 5、无限重复，间隔1秒

四、CronTrigger

- 1、CronTrigger 是什么？
- 2、TestQuartz
- 3、理解Cron
- 4、Cron 表达式举例

五、监听器

- 1、监听器概念
 - 2、创建MailJobListener
 - 3、创建TestQuartz
- 测试

六、JDBC

- 1、JDBCStore 概念
- 2、建表
- 3、配置文件
- 4、创建MailJob

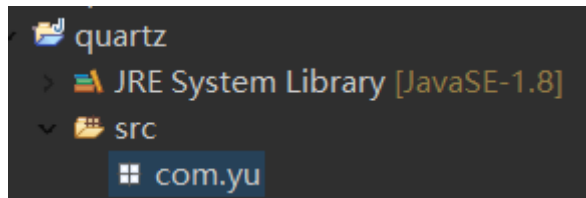
七、cluster

- 1、概念
- 2、创建
- 3、修改配置文件
- 4、测试

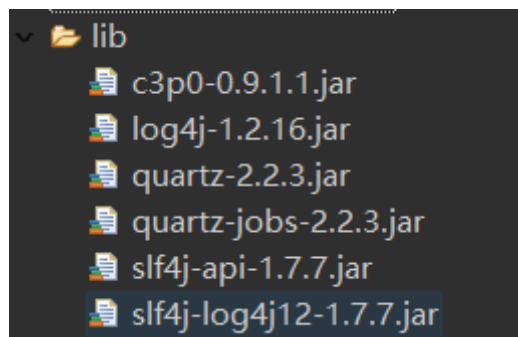
一、Quartz-定时器入门

1、Quartz 版本

2、创建项目



3、导入jar包



4、创建TestQuartz类

触发器 Trigger: 什么时候工作

任务 Job: 做什么工作

调度器 Scheduler: 搭配 Trigger和Job

```
public class TestQuartz {  
    public static void main(String[] args) {  
        //  
    }  
}
```

5、定义调度器

```
//创建调度器  
Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
```

6、定义一个触发器

```
//定义一个触发器
//定义名称和所属的租
Trigger trigger = newTrigger().withIdentity("trigger1", "group1")
    .startNow()
    .withSchedule(simpleSchedule()
        .withIntervalInSeconds(2) //每隔2秒执行一次
        .withRepeatCount(10)) //总共执行11次(第一次执行不基数)
    .build();
```

7、定义一个JobDetail任务

```
// 定义一个JobDetail
JobDetail job = newJob(MailJob.class) // 指定干活的类MailJob
    .withIdentity("mailjob1", "mailgroup") // 定义任务名称和分组
    .usingJobData("email", "admin@10086.com") // 定义属性
    .build();
```

8、指定干活的类

```
/**
 * 具体方法
 * @author 18285
 */
public class MailJob implements Job {
    public void execute(JobExecutionContext context) //
        throws JobExecutionException {
        //获取到对象
        JobDetail detail = context.getJobDetail();
        //获取到数据email的值
        String email = detail.getJobDataMap().getString("email");
        //定义时间
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        String now = sdf.format(new Date());
        System.out.printf("给邮件地址 %s 发出了一封定时邮件，当前时间是： %s\n", //
            email, now);
    }
}
```

9、调度加入这个job任务

```
// 调度加入这个job任务
scheduler.scheduleJob(job, trigger);
```

10、启动

```
// 启动
scheduler.start();
```

11、线程休眠

```
// 等待20秒，让前面的任务都执行完了之后，再关闭调度器
Thread.sleep(20000);
scheduler.shutdown(true);
```

12、创建log4j文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

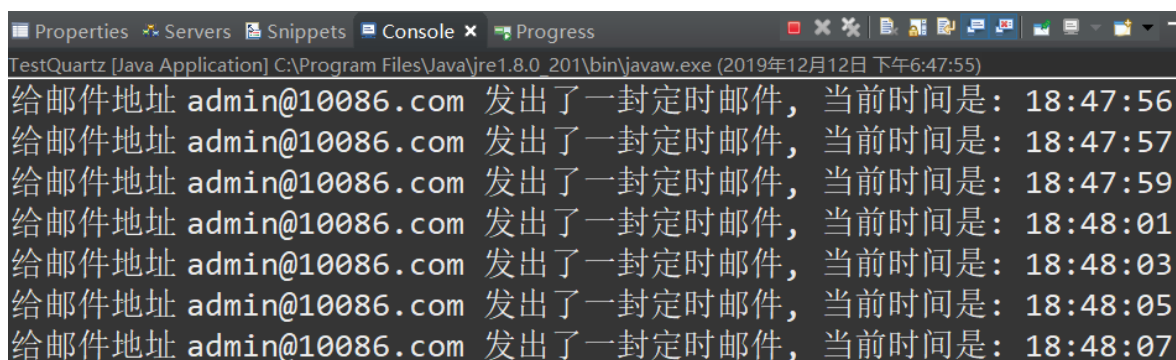
<log4j:configuration
  xmlns:log4j="http://jakarta.apache.org/log4j/">

  <appender name="default"
    class="org.apache.log4j.ConsoleAppender">
    <param name="target" value="System.out" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
        value="[%p] %d{dd MMM hh:mm:ss.SSS aa} %t [%c]%n%m%n%n" />
    </layout>
  </appender>
  <logger name="com.yu">
    <level value="error" />
  </logger>

  <root>
    <level value="error" />
    <appender-ref ref="default" />
  </root>

</log4j:configuration>
```

13、运行



The screenshot shows a Java application window titled 'TestQuartz [Java Application]'. The console output displays a series of log messages from the 'com.yu' logger, indicating that a scheduled email is being sent to 'admin@10086.com' at regular intervals. The messages are as follows:

Log Message	Current Time
给邮件地址 admin@10086.com 发出了一封定时邮件,	18:47:56
给邮件地址 admin@10086.com 发出了一封定时邮件,	18:47:57
给邮件地址 admin@10086.com 发出了一封定时邮件,	18:47:59
给邮件地址 admin@10086.com 发出了一封定时邮件,	18:48:01
给邮件地址 admin@10086.com 发出了一封定时邮件,	18:48:03
给邮件地址 admin@10086.com 发出了一封定时邮件,	18:48:05
给邮件地址 admin@10086.com 发出了一封定时邮件,	18:48:07

二、Quartz-job管理

1、Job 组成部分

JobDetail: 用于描述这个Job是做什么的
实现Job的类: 具体干活的
JobDataMap: 给 Job 提供参数用的
JobDataMap 除了usingJobData 方式之外, 还可以是其他方式, 像这样:
job.getJobDataMap().put("email", "admin@taobao.com");

```
//部分代码, 基于上一个案例
// 定义一个JobDetail任务
    JobDetail job = newJob(MailJob.class) // 指定干活的类MailJob
        .withIdentity("mailjob1", "mailgroup") // 定义任务名称和分组
        .usingJobData("email", "admin@10086.com") // 定义属性
        .build();

//添加这里
    //用JobDataMap 修改email
    job.getJobDataMap().put("email", "admin@1008611.com");
```

测试:

```
TestQuartz (1) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe
给邮件地址 admin@1008611.com 发出了一封定
给邮件地址 admin@1008611.com 发出了一封定
```

2、Job 并发

```
//-----实现job是具体干活的类-----
//添加一个注解就ok了
//@DisallowConcurrentExecution
//具体代码跟之前的一样, 是基于上一个案例的代码
import org.quartz.DisallowConcurrentExecution;
import org.quartz.Job;
import org.quartz.JobDetail;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
/**
 * job并发 //@DisallowConcurrentExecution
 * @author 18285
 */
@DisallowConcurrentExecution
public class DatabaseBackupJob implements Job {
    public void execute(JobExecutionContext context)//
        throws JobExecutionException {
        //获取到域对象context
        JobDetail detail = context.getJobDetail();
        //获取到数据 datajob 的值
        String datajob = detail.getJobDataMap().getString("datajob");
        //定义时间 %s: 字符串, %n: 换行
        System.out.printf("给数据库 %s 备份, 耗时10秒 %n", datajob);
    }
}
```

```
import static org.quartz.JobBuilder.newJob;
import static org.quartz.SchedulerBuilder.scheduler;
```

```

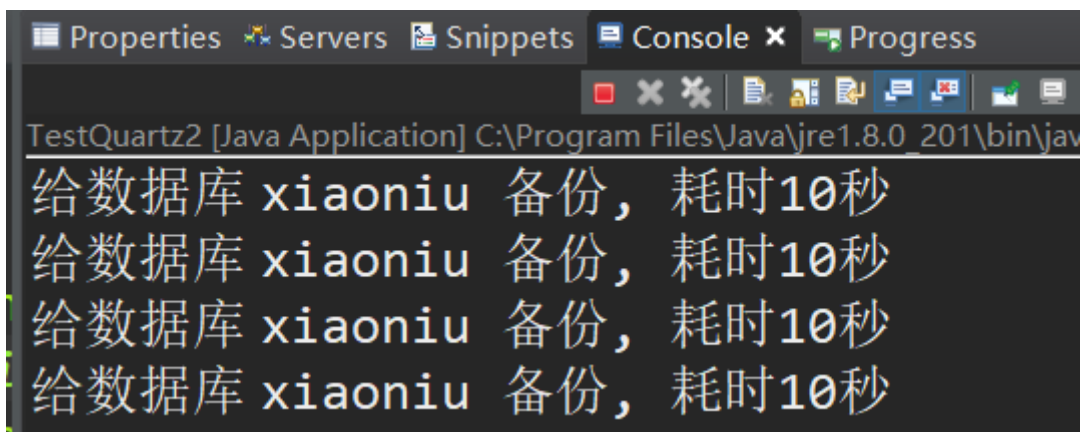
import static org.quartz.TriggerBuilder.newTrigger;
import org.quartz.JobDetail;
import org.quartz.Scheduler;
import org.quartz.Trigger;
import org.quartz.impl.StdSchedulerFactory;
/**
 * quartz job管理>并发
 * @author 18285
 */
public class TestQuartz2 {
    public static void main(String[] args) throws Exception {
        // 创建调度器
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        // 定义一个触发器
        // 定义名称和所属的组
        Trigger trigger = newTrigger().withIdentity("trigger1", "group1")//
            .startNow().withSchedule(simpleSchedule()//
                .withIntervalInSeconds(2) // 每隔2秒执行一次
                .withRepeatCount(6)) // 总共执行6次(第一次执行不基数)
            .build();

        // 定义一个JobDetail任务
        JobDetail job = newJob(DatabaseBackupJob.class) // 指定干活的类MailJob
            .withIdentity("backupjob", "databasegroup") // 定义任务名称和分组
            .usingJobData("datajob", "xiaoniu") // 定义属性
            .build();

        // 调度加入这个job任务
        scheduler.scheduleJob(job, trigger);
        // 启动
        scheduler.start();
        // 等待20秒,让前面的任务都执行完了之后,再关闭调度器
        Thread.sleep(20000);
        scheduler.shutdown(true); // 关闭调度器
    }
}

```

测试



```

TestQuartz2 [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\jav
给数据库 xiaoniu 备份, 耗时10秒
给数据库 xiaoniu 备份, 耗时10秒
给数据库 xiaoniu 备份, 耗时10秒
给数据库 xiaoniu 备份, 耗时10秒

```

3、job异常

创建异常类1 (job

```

/**
 * job异常方式1
 *
 * @author 18285
 */
public class ExceptionJob1 implements Job {
    public void execute(JobExecutionContext context)//
        throws JobExecutionException {
        int i = 0;
        try {
            System.out.println(100 / i);
        } catch (Exception e) {
            System.out.println("发生了异常,取消这个Job 对应的所有调度");
            JobExecutionException je = new JobExecutionException(e);
            je.setUnscheduleAllTriggers(true);
            throw je;
        }
    }
}

```

创建异常类2 (job

```

import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
/**
 * job异常方式2
 * @author 18285
 */
public class ExceptionJob2 implements Job {
    static int i=0;
    public void execute(JobExecutionContext context)//
        throws JobExecutionException {
        try {
            //故意发生异常
            System.out.println(100/i);
        } catch (Exception e) {
            System.out.println("发生了异常,修改一下参数,立即重新执行");
            i=1;
            JobExecutionException je = new JobExecutionException(e);
            je.setUnscheduleAllTriggers(true);
            throw je;
        }
    }
}

```

定时器1

```

import static org.quartz.JobBuilder.newJob;
import static org.quartz.SchedulerBuilder.simpleSchedule;
import static org.quartz.TriggerBuilder.newTrigger;

import org.quartz.JobDetail;
import org.quartz.Scheduler;
import org.quartz.Trigger;
import org.quartz.impl.StdSchedulerFactory;

```

```

/**
 * quartz 异常测试定时器
 * @author 18285
 */
public class TestQuartz {
    public static void main(String[] args) throws Exception {
        ExceptionJob1();
    }

    /**
     * 异常1
     * @throws Exception
     */
    private static void ExceptionJob1() throws Exception {
        // 创建调度器
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        // 定义一个触发器
        // 定义名称和所属的组
        Trigger trigger = newTrigger().withIdentity("trigger1", "group1")//
            .startNow().withSchedule(simpleSchedule()//
                .withIntervalInSeconds(2) // 每隔2秒执行一次
                .withRepeatCount(6)) // 总共执行11次(第一次执行不基数)
            .build();

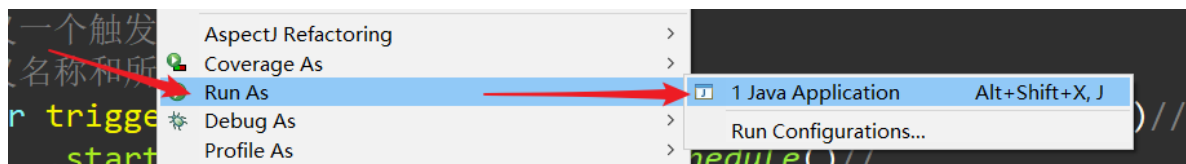
        // 定义一个JobDetail任务
        JobDetail job = newJob(ExceptionJob1.class) // 指定干活的类MailJob
            .withIdentity("exceptionJob1", "databasegroup") // 定义任务名称和分
            组
            .build();

        // 调度加入这个job任务
        scheduler.scheduleJob(job, trigger);
        // 启动
        scheduler.start();
        // 等待20秒, 让前面的任务都执行完了之后, 再关闭调度器
        Thread.sleep(20000);
        scheduler.shutdown(true); // 关闭调度器
    }
}

```

测试1

//在main方法中点击右键运行



定时器2

```

/**
 * 异常测试2
 * @throws Exception

```



```

    */
    private static void exceptionJob2() throws Exception {
        // 创建调度器
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        // 定义一个触发器
        // 定义名称和所属的租
        Trigger trigger = newTrigger().withIdentity("trigger1", "group1")//
            .startNow().withSchedule(simpleSchedule()//
                .withIntervalInSeconds(2) // 每隔2秒执行一次
                .withRepeatCount(6)) // 总共执行11次(第一次执行不基数)
            .build();

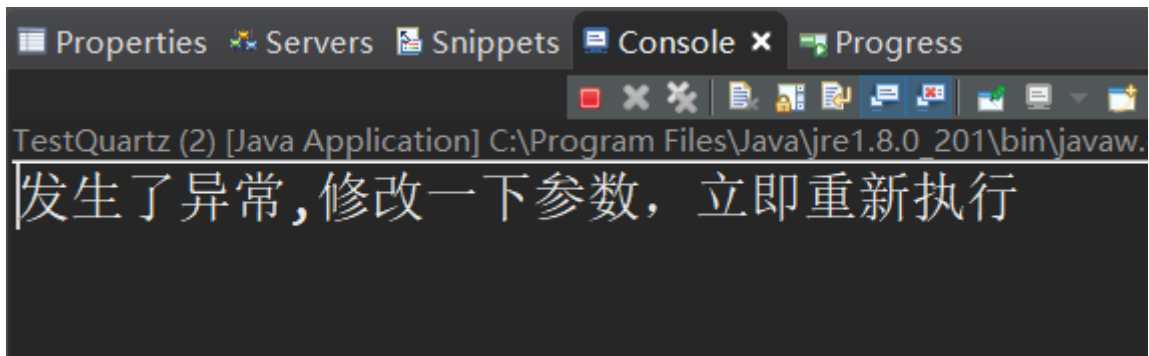
        // 定义一个JobDetail任务
        JobDetail job = newJob(ExceptionJob2.class) // 指定干活的类MailJob
            .withIdentity("exceptionJob2", "databasegroup") // 定义任务名称和分
组
            .build();

        // 调度加入这个job任务
        scheduler.scheduleJob(job, trigger);
        // 启动
        scheduler.start();
        // 等待20秒, 让前面的任务都执行完了之后, 再关闭调度器
        Thread.sleep(20000);
        scheduler.shutdown(true); // 关闭调度器
    }

```

测试2

//操作同上一次运行



4、中断 job

创建stoppableJob类

```

import org.quartz.InterruptableJob;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
import org.quartz.UnableToInterruptJobException;

/**
 * 实现InterruptableJob就可以中断了, 而非 Job才能够被中断
 * @author 18285
 */
public class StoppableJob implements InterruptableJob {

```

```

//定义状态
private boolean stop = false;
@Override
public void execute(JobExecutionContext arg0) throws JobExecutionException {
    while(true) {
        if(stop) {
            break;
        }
        try {
            System.out.println("每隔1秒就进行一次检测，看是否停止");
            Thread.sleep(1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("持续工作中...");
    }
}
@Override
public void interrupt() throws UnableToInterruptJobException {
    System.out.println("被调度叫停");
    stop=true;
}
}
}

```

测试类

```

/**
 * 中断测试
 * @throws SchedulerException
 * @throws InterruptedException
 */
private static void stop() throws SchedulerException, InterruptedException {
    // 创建调度器
    Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
    // 定义一个触发器
    // 定义名称和所属的组
    Trigger trigger = new Trigger().withIdentity("trigger1", "group1")//
        .startNow().build();

    // 定义一个JobDetail任务
    JobDetail job = new Job(StoppableJob.class) // 指定干活的类MailJob
        .withIdentity("stoppableJob", "stoppableJob...") // 定义任务名称和
        分组
        .build();
    // 调度加入这个job任务
    scheduler.scheduleJob(job, trigger);
    // 启动
    scheduler.start();
    //
    Thread.sleep(5000);
    System.out.println("过5秒，调度停止 job");

    // key 就相当于这个Job的主键
    scheduler.interrupt(job.getKey());

    // 等待20秒，让前面的任务都执行完了之后，再关闭调度器
}

```

测试

[illegible]

```
import static org.quartz.JobBuilder.newJob;
import static org.quartz.TriggerBuilder.newTrigger;

import java.util.Date;

import org.quartz.DateBuilder;
import org.quartz.JobDetail;
import org.quartz.Scheduler;
import org.quartz.SimpleTrigger;
import org.quartz.impl.StdSchedulerFactory;

/**
 * quartz 异常测试定时器
 *
 * @author 18285
 */
```

```

public class TestQuartz {
    public static void main(String[] args) throws Exception {
        // 创建调度器
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        Date startTime = DateBuilder.nextGivenSecondDate(null, 8);
        JobDetail job = newJob(MailJob.class).withIdentity("mailJob",
"mailGroup").build();
        SimpleTrigger trigger = (SimpleTrigger)
newTrigger().withIdentity("trigger1", "group1").startAt(startTime)
                .build();

        Date ft = scheduler.scheduleJob(job, trigger);

        System.out.println("当前时间是: " + new Date().toLocaleString());
        System.out.printf("%s 这个任务会在 %s 准时开始运行, 累计运行%d次, 间隔时间是%d毫
秒%n", job.getKey(), ft.toLocaleString(),
                trigger.getRepeatCount() + 1, trigger.getRepeatInterval());

        scheduler.start();

        // 等待200秒, 让前面的任务都执行完了之后, 再关闭调度器
        Thread.sleep(200000);
        scheduler.shutdown(true);
    }
}

```

2、创建job

```

import java.text.SimpleDateFormat;
import java.util.Date;

import org.quartz.Job;
import org.quartz.JobDetail;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

public class MailJob implements Job {
    public void execute(JobExecutionContext context)//
        throws JobExecutionException {
        JobDetail detail = context.getJobDetail();
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        String now = sdf.format(new Date());

        System.out.printf("发出了一封邮件, 当前时间是: %s%n", now);
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

测试 8s

当前时间是: 2019-12-12 23:23:28

mailGroup.mailJob 这个任务会在 2019-12-12 23:23:32 准时开始运行, 累计运行1次, 间隔时间是0毫秒

发出了一封邮件, 当前时间是: 23:23:32

3、10 秒后运行

修改

```
//Date startTime = DateBuilder.futureDate(10, IntervalUnit.SECOND);
Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

Date startTime = DateBuilder.futureDate(10, IntervalUnit.SECOND);
// 定义一个JobDetail任务
JobDetail job = newJob(MailJob.class)//
    .withIdentity("mailJob", "mailGroup").build();

SimpleTrigger trigger = (SimpleTrigger) newTrigger()//
    .withIdentity("trigger1", "group1")//
    .startAt(startTime).build();

Date ft = scheduler.scheduleJob(job, trigger);

System.out.println("当前时间是: " + new Date().toLocaleString());
System.out.printf("%s 这个任务会在 %s 准时开始运行, " + "累计运行%d次, 间隔时间"
    是%d毫秒%n", //
    job.getKey(), ft.toLocaleString(), trigger.getRepeatCount() + 1,
    //
    trigger.getRepeatInterval());
// 启动
scheduler.start();

// 等待200秒, 让前面的任务都执行完了之后, 再关闭调度器
Thread.sleep(200000);
scheduler.shutdown(true);
```

测试

```
testQuartz (5) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (2019-12-12 23:21:05)
当前时间是: 2019-12-12 23:21:05
mailGroup.mailJob 这个任务会在 2019-12-12 23:21:15 准时开始运行, 累计运行1次, 间隔时间是0
发出了一封邮件, 当前时间是: 23:21:15
```

4、累计n次, 间隔n秒

```
Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

Date startTime = DateBuilder.nextGivenSecondDate(null, 8);

JobDetail job = newJob(MailJob.class)//
    .withIdentity("mailJob", "mailGroup").build();
//这里发生了变化
SimpleTrigger trigger = (SimpleTrigger) newTrigger()
    .withIdentity("trigger1", "group1")
```

```

        .startAt(startTime)
        .withSchedule(simpleSchedule()
            .withRepeatCount(3)
            .withIntervalInSeconds(1))
        .build();

Date ft = scheduler.scheduleJob(job, trigger);

System.out.println("当前时间是: " + new Date().toLocaleString());
System.out.printf("%s 这个任务会在 %s 准时开始运行, 累计运行%d次, 间隔时间是%d毫  
秒%n", job.getKey(), ft.toLocaleString(),
    trigger.getRepeatCount() + 1, trigger.getRepeatInterval());

scheduler.start();

// 等待20秒, 让前面的任务都执行完了之后, 再关闭调度器
Thread.sleep(20000);
scheduler.shutdown(true);

```

测试

当前时间是: 2019-12-12 23:32:05
 mailGroup.mailJob 这个任务会在 2019-12-12 23:32:08 准时开始运行, 累计运行4次, 间隔时间是1000毫秒
 发出了一封邮件, 当前时间是: 23:32:08
 发出了一封邮件, 当前时间是: 23:32:09
 发出了一封邮件, 当前时间是: 23:32:10
 发出了一封邮件, 当前时间是: 23:32:11

5、无限重复, 间隔1秒

修改

```

SimpleTrigger trigger = (SimpleTrigger) newTrigger()//
    .withIdentity("trigger1", "group1").startAt(startTime)
    .withSchedule(simpleSchedule()//
        .repeatForever()// 重复
        .withIntervalInSeconds(1)// 间隔1秒
    ).build();

```

测试

当前时间是: 2019-12-12 23:35:08
 mailGroup.mailJob 这个任务会在 2019-12-12 23:35:16 准时开始运行, 累计运行0次, 间隔时间是1000毫秒
 发出了一封邮件, 当前时间是: 23:35:16
 发出了一封邮件, 当前时间是: 23:35:17
 发出了一封邮件, 当前时间是: 23:35:18
 发出了一封邮件, 当前时间是: 23:35:19
 发出了一封邮件, 当前时间是: 23:35:20
 发出了一封邮件, 当前时间是: 23:35:21
 发出了一封邮件, 当前时间是: 23:35:22
 发出了一封邮件, 当前时间是: 23:35:23

四、CronTrigger

1、CronTrigger 是什么？

Cron 是Linux下的一个定时器，功能很强大，但是表达式更为复杂
CronTrigger 就是用 Cron 表达式来安排触发时间和次数的。

2、TestQuartz

```
//基于上次修改
Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

JobDetail job = newJob(MailJob.class)//
    .withIdentity("mailJob", "mailGroup").build();

CronTrigger trigger = newTrigger()//
    .withIdentity("trigger1", "group1")//
    .withSchedule(cronSchedule("0/2 * * * * ?"))//表示每隔2秒执行一次
    .build();

Date ft = scheduler.scheduleJob(job, trigger);

System.out.println("当前时间是: " + new Date().toLocaleString());

scheduler.start();

// 等待20秒，让前面的任务都执行完了之后，再关闭调度器
Thread.sleep(20000);
scheduler.shutdown(true);
```

测试

```
当前时间是: 2019-12-12 23:43:56
发出了一封邮件，当前时间是: 23:43:56
发出了一封邮件，当前时间是: 23:43:58
发出了一封邮件，当前时间是: 23:44:00
发出了一封邮件，当前时间是: 23:44:02
```

3、理解Cron

位置	时间域	允许值	特殊值
1	秒	0-59	, - * /
2	分钟	0-59	, - * /
3	小时	0-23	, - * /
4	日期	1-31	, - * ? / L W C
5	月份	1-12	, - * /
6	星期	1-7	, - * ? / L C #
7	年份 (可选)	1-31	, - * /

由7个部分组成，每个部分就如图所示分别对应秒 分 一直到年

星号 (*)：可用在所有字段中，表示对应时间域的每一个时刻，例如， 在分钟字段时，表示“每分钟”；

问号 (?)：该字符只在日期和星期字段中使用，它通常指定为“无意义的值”，相当于点位符；

减号 (-)：表达一个范围，如在小时字段中使用“10-12”，则表示从10到12点，即10,11,12；

逗号 (,)：表达一个列表值，如在星期字段中使用“MON,WED,FRI”，则表示星期一，星期三和星期五；

4、Cron 表达式举例

表达式	含义
0 0 12 * * ?	每天12:00触发
0 15 10 ? * *	每天10:15触发
0 15 10 * * ?	每天10:15触发
0 15 10 * * ? *	每天10:15触发
0 15 10 * * ? 2005	2005年的每天10:15触发
0 * 14 * * ?	每天14:00至14:59每分钟触发
0 0/5 14 * * ?	每天14:00开始至14:55每5分钟触发
0 0/5 14,18 * * ?	每天14:00开始至14:55每5分钟触发，18:00开始至18:55每5分钟触发
0 0-5 14 * * ?	每天14:00开始至14:05每分钟触发
0 10,44 14 ? 3 WED	三月份的每个周三14:10和14:44触发
0 15 10 ? * MON-FRI	每个周一至周五上午10:15触发
0 15 10 15 * ?	每月15号10:15触发
0 15 10 L * ?	每月最后一天10:15触发
0 15 10 L-2 * ?	每月倒数第2天10:15触发
0 15 10 ? * 6L	每月最后一个周五10:15触发
0 15 10 ? * 6L 2002-2005	2002年至2005年每月最后一个周五10:15触发
0 15 10 ? * 6#3	每月第3个周五10:15触发
0 0 12 1/5 * ?	每月的第一天开始，每个第5天12:00触发
0 11 11 11 11 ?	每年11月11号11:11触发

五、监听器

1、监听器概念

Quartz 的监听器有Job监听器，Trigger监听器， Scheduler监听器，对不同层面进行监控。 实际业务用的较多的是Job监听器，用于监听器是否执行了

2、创建MailJobListener

```
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
import org.quartz.JobListener;

public class MailJobListener implements JobListener {
```

```

@Override
public String getName() {
    return "listener of mail job";
}

@Override
public void jobExecutionVetoed(JobExecutionContext context) {
    System.out.println("取消执行: \t " + context.getJobDetail().getKey());
}

@Override
public void jobToBeExecuted(JobExecutionContext context) {
    System.out.println("准备执行: \t " + context.getJobDetail().getKey());
}

@Override
public void jobWasExecuted(JobExecutionContext context,
JobExecutionException exception) {
    System.out.println("执行结束: \t " + context.getJobDetail().getKey());
    System.out.println();
}
}

```

3、创建TestQuartz

```

private static void test4() throws Exception {
    // 创建调度器
    Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

    // 定义一个触发器
    Trigger trigger = newTrigger().withIdentity("trigger1", "group1") // 定义
名称和所属的组

.startNow().withSchedule(simpleSchedule().withIntervalInSeconds(2) // 每隔2秒执行
一次

.withRepeatCount(10)) // 总共执行11次(第一次执行不基数)
.build();

    // 定义一个JobDetail
    JobDetail mailJob = newJob(MailJob.class) // 指定干活的类MailJob
.withIdentity("mailjob1", "mailgroup") // 定义任务名称和分组
.usingJobData("email", "admin@1008611.com") // 定义属性
.build();

    // 增加Job监听
    MailJobListener mailJobListener = new MailJobListener();
    KeyMatcher<JobKey> keyMatcher = KeyMatcher.keyEquals(mailJob.getKey());
    scheduler.getListenerManager().addJobListener(mailJobListener,
keyMatcher);

    //调度加入这个job
    scheduler.scheduleJob(mailJob, trigger);
}

```

```
scheduler.start();

// 等待20秒，让前面的任务都执行完了之后，再关闭调度器
Thread.sleep(20000);
scheduler.shutdown(true);
}
```

测试

准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:09
准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:11
准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:13
准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:15
准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:17
准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:19
执行结束: mailgroup.mailjob1

准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:21
执行结束: mailgroup.mailjob1

准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:23
执行结束: mailgroup.mailjob1

准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:25
执行结束: mailgroup.mailjob1

准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:27
执行结束: mailgroup.mailjob1

准备执行: mailgroup.mailjob1
发出了一封邮件，当前时间是：23:56:29
执行结束: mailgroup.mailjob1

六、JDBC

1、JDBCStore 概念

默认情况，Quartz的触发器，调度，任务等信息都是放在内存中的，叫做 **RAMJobStore**。好处是快速，坏处是一旦系统重启，那么信息就丢失了，就得全部从头来过。
所以Quartz还提供了另一个方式，可以把这些信息存放在数据库做，叫做 **JobStoreTX**。

2、建表

```
DROP DATABASE IF EXISTS quartz;
CREATE DATABASE quartz DEFAULT CHARACTER SET utf8;
USE quartz;

DROP TABLE IF EXISTS QRTZ_FIRED_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_PAUSED_TRIGGER_GRPS;
DROP TABLE IF EXISTS QRTZ_SCHEDULER_STATE;
DROP TABLE IF EXISTS QRTZ_LOCKS;
DROP TABLE IF EXISTS QRTZ_SIMPLE_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_SIMPROP_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_CRON_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_BLOB_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_JOB_DETAILS;
DROP TABLE IF EXISTS QRTZ_CALEDARS;

CREATE TABLE QRTZ_JOB_DETAILS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    JOB_NAME  VARCHAR(100) NOT NULL,
    JOB_GROUP VARCHAR(100) NOT NULL,
    DESCRIPTION VARCHAR(250) NULL,
    JOB_CLASS_NAME   VARCHAR(250) NOT NULL,
    IS_DURABLE VARCHAR(1) NOT NULL,
    IS_NONCONCURRENT VARCHAR(1) NOT NULL,
    IS_UPDATE_DATA VARCHAR(1) NOT NULL,
    REQUESTS_RECOVERY VARCHAR(1) NOT NULL,
    JOB_DATA BLOB NULL,
    PRIMARY KEY (SCHED_NAME, JOB_NAME, JOB_GROUP)
);

CREATE TABLE QRTZ_TRIGGERS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    TRIGGER_NAME VARCHAR(100) NOT NULL,
    TRIGGER_GROUP VARCHAR(100) NOT NULL,
    JOB_NAME  VARCHAR(100) NOT NULL,
    JOB_GROUP VARCHAR(100) NOT NULL,
    DESCRIPTION VARCHAR(250) NULL,
    NEXT_FIRE_TIME BIGINT(13) NULL,
    PREV_FIRE_TIME BIGINT(13) NULL,
    PRIORITY INTEGER NULL,
    TRIGGER_STATE VARCHAR(16) NOT NULL,
    TRIGGER_TYPE VARCHAR(8) NOT NULL,
    START_TIME BIGINT(13) NOT NULL,
    END_TIME BIGINT(13) NULL,
    CALENDAR_NAME VARCHAR(100) NULL,
    MISFIRE_INSTR SMALLINT(2) NULL,
    JOB_DATA BLOB NULL,
    PRIMARY KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP),
    FOREIGN KEY (SCHED_NAME, JOB_NAME, JOB_GROUP)
        REFERENCES QRTZ_JOB_DETAILS(SCHED_NAME, JOB_NAME, JOB_GROUP)
);
```

```

CREATE TABLE QRTZ_SIMPLE_TRIGGERS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    TRIGGER_NAME VARCHAR(100) NOT NULL,
    TRIGGER_GROUP VARCHAR(100) NOT NULL,
    REPEAT_COUNT BIGINT(7) NOT NULL,
    REPEAT_INTERVAL BIGINT(12) NOT NULL,
    TIMES_TRIGGERED BIGINT(10) NOT NULL,
    PRIMARY KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP),
    FOREIGN KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
);

```

```

CREATE TABLE QRTZ_CRON_TRIGGERS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    TRIGGER_NAME VARCHAR(100) NOT NULL,
    TRIGGER_GROUP VARCHAR(100) NOT NULL,
    CRON_EXPRESSION VARCHAR(100) NOT NULL,
    TIME_ZONE_ID VARCHAR(80),
    PRIMARY KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP),
    FOREIGN KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
);

```

```

CREATE TABLE QRTZ_SIMPROP_TRIGGERS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    TRIGGER_NAME VARCHAR(100) NOT NULL,
    TRIGGER_GROUP VARCHAR(100) NOT NULL,
    STR_PROP_1 VARCHAR(512) NULL,
    STR_PROP_2 VARCHAR(512) NULL,
    STR_PROP_3 VARCHAR(512) NULL,
    INT_PROP_1 INT NULL,
    INT_PROP_2 INT NULL,
    LONG_PROP_1 BIGINT NULL,
    LONG_PROP_2 BIGINT NULL,
    DEC_PROP_1 NUMERIC(13,4) NULL,
    DEC_PROP_2 NUMERIC(13,4) NULL,
    BOOL_PROP_1 VARCHAR(1) NULL,
    BOOL_PROP_2 VARCHAR(1) NULL,
    PRIMARY KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP),
    FOREIGN KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
);

```

```

CREATE TABLE QRTZ_BLOB_TRIGGERS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    TRIGGER_NAME VARCHAR(100) NOT NULL,
    TRIGGER_GROUP VARCHAR(100) NOT NULL,
    BLOB_DATA BLOB NULL,
    PRIMARY KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP),
    FOREIGN KEY (SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(SCHED_NAME, TRIGGER_NAME, TRIGGER_GROUP)
);

```

```

CREATE TABLE QRTZ_CALENDARS

```

```

(
    SCHED_NAME VARCHAR(120) NOT NULL,
    CALENDAR_NAME VARCHAR(100) NOT NULL,
    CALENDAR BLOB NOT NULL,
    PRIMARY KEY (SCHED_NAME, CALENDAR_NAME)
);

CREATE TABLE QRTZ_PAUSED_TRIGGER_GRPS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    TRIGGER_GROUP VARCHAR(100) NOT NULL,
    PRIMARY KEY (SCHED_NAME, TRIGGER_GROUP)
);

CREATE TABLE QRTZ_FIRED_TRIGGERS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    ENTRY_ID VARCHAR(95) NOT NULL,
    TRIGGER_NAME VARCHAR(100) NOT NULL,
    TRIGGER_GROUP VARCHAR(100) NOT NULL,
    INSTANCE_NAME VARCHAR(100) NOT NULL,
    FIRED_TIME BIGINT(13) NOT NULL,
    SCHED_TIME BIGINT(13) NOT NULL,
    PRIORITY INTEGER NOT NULL,
    STATE VARCHAR(16) NOT NULL,
    JOB_NAME VARCHAR(100) NULL,
    JOB_GROUP VARCHAR(100) NULL,
    IS_NONCONCURRENT VARCHAR(1) NULL,
    REQUESTS_RECOVERY VARCHAR(1) NULL,
    PRIMARY KEY (SCHED_NAME, ENTRY_ID)
);

CREATE TABLE QRTZ_SCHEDULER_STATE
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    INSTANCE_NAME VARCHAR(100) NOT NULL,
    LAST_CHECKIN_TIME BIGINT(13) NOT NULL,
    CHECKIN_INTERVAL BIGINT(13) NOT NULL,
    PRIMARY KEY (SCHED_NAME, INSTANCE_NAME)
);

CREATE TABLE QRTZ_LOCKS
(
    SCHED_NAME VARCHAR(120) NOT NULL,
    LOCK_NAME VARCHAR(40) NOT NULL,
    PRIMARY KEY (SCHED_NAME, LOCK_NAME)
);

commit;

```

3、配置文件

//创建文件

```

org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX
org.quartz.jobStore.tablePrefix = QRTZ_
org.quartz.scheduler.instanceName = MyScheduler
org.quartz.threadPool.threadCount = 3
org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX
org.quartz.jobStore.driverDelegateClass =
org.quartz.impl.jdbcjobstore.StdJDBCDelegate
org.quartz.jobStore.tablePrefix = QRTZ_
org.quartz.jobStore.dataSource = mysqlDatabase

org.quartz.dataSource.mysqlDatabase.driver = com.mysql.jdbc.Driver
org.quartz.dataSource.mysqlDatabase.URL = jdbc:mysql://localhost:3306/quartz?
characterEncoding=utf-8
org.quartz.dataSource.mysqlDatabase.user = root
org.quartz.dataSource.mysqlDatabase.password = root
org.quartz.dataSource.mysqlDatabase.maxConnections = 5

```

4、创建MailJob

```

import java.text.SimpleDateFormat;
import java.util.Date;

import org.quartz.DisallowConcurrentExecution;
import org.quartz.Job;
import org.quartz.JobDetail;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

@DisallowConcurrentExecution
public class MailJob implements Job {
    public void execute(JobExecutionContext context)//
        throws JobExecutionException {
        JobDetail detail = context.getJobDetail();
        String email = detail.getJobDataMap().getString("email");

        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        String now = sdf.format(new Date());

        System.out.printf("给邮件地址 %s 发出了一封定时邮件， "
            + "当前时间是： %s (%s)%n" ,//
            email, now, context.isRecovering());
    }
}

```

5、创建TestQuartz

```

import static org.quartz.JobBuilder.newJob;
import static org.quartz.SchedulerBuilder.simpleSchedule;
import static org.quartz.TriggerBuilder.newTrigger;

import org.quartz.JobDetail;

```

```

import org.quartz.ObjectAlreadyExistsException;
import org.quartz.Scheduler;
import org.quartz.SchedulerException;
import org.quartz.Trigger;
import org.quartz.impl.StdSchedulerFactory;

/**
 *
 * @author 18285
 */
public class TestQuartz {
    public static void main(String[] args) throws Exception {
        try {
            assginNewJob();
        } catch (ObjectAlreadyExistsException e) {
            System.err.println("发现任务已经在数据库存在了，直接从数据库里运行："+
e.getMessage());
            resumeJobFromDatabase();
        }
    }

    private static void resumeJobFromDatabase() throws Exception {
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        scheduler.start();
        // 等待200秒，让前面的任务都执行完了之后，再关闭调度器
        Thread.sleep(200000);
        scheduler.shutdown(true);
    }

    private static void assginNewJob() throws SchedulerException,
InterruptedException {
        // 创建调度器
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        // 定义一个触发器
        Trigger trigger = new Trigger().withIdentity("trigger1", "group1") // 定义
名称和所属的租
        .startNow()
        .withSchedule(simpleSchedule().withIntervalInSeconds(15) // 每隔
15秒执行一次
        .withRepeatCount(10)) // 总共执行11次(第一次执行不基数)
        .build();

        // 定义一个JobDetail
        JobDetail job = new Job(MailJob.class) // 指定干活的类MailJob
        .withIdentity("mailjob1", "mailgroup") // 定义任务名称和分组
        .usingJobData("email", "admin@10086.com") // 定义属性
        .build();

        // 调度加入这个job
        scheduler.scheduleJob(job, trigger);

        // 启动
        scheduler.start();

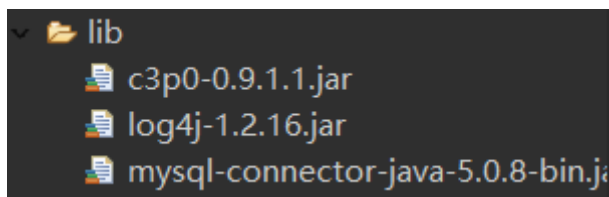
        // 等待20秒，让前面的任务都执行完了之后，再关闭调度器
        Thread.sleep(20000);
        scheduler.shutdown(true);
    }
}

```



```
}
```

导入jar



测试

给邮件地址 admin@10086.com 发出了一封定时邮件，当前时间是：00:25:16 (false)
给邮件地址 admin@10086.com 发出了一封定时邮件，当前时间是：00:25:18 (false)
给邮件地址 admin@10086.com 发出了一封定时邮件，当前时间是：00:25:20

七、cluster

1、概念

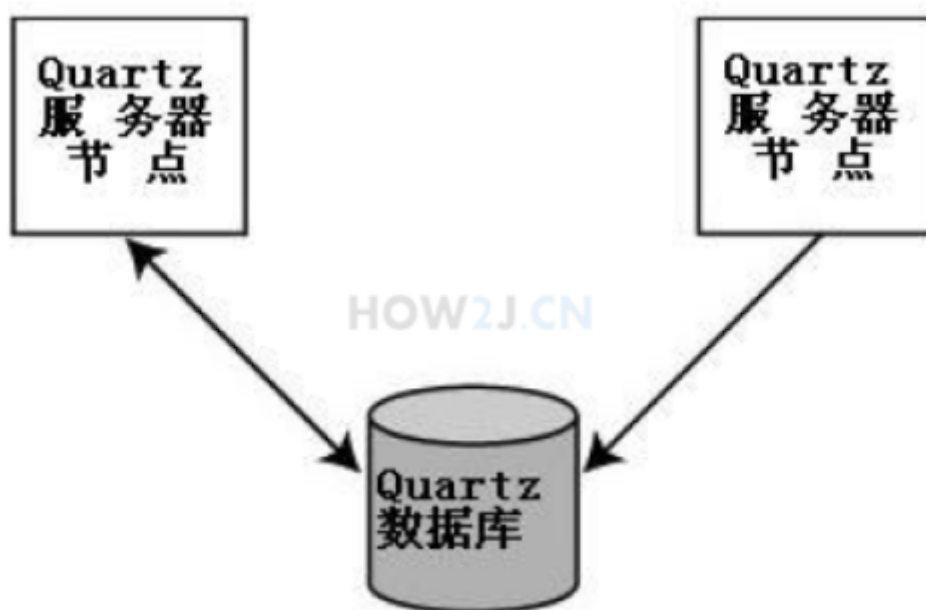
这所谓的Quartz集群，是指在 基于数据库存储 Quartz调度信息 的基础上， 有多个一模一样的 Quartz应用在运行。

当某一个Quartz 应用重启或者发生问题的时候， 其他的Quartz 应用会 借助 数据库这个桥梁探知到它不行了，从而接手把该进行的Job调度工作进行下去。

以这种方式保证任务调度的高可用性，即在发生异常重启等情况下，调度信息依然连贯性地进行下去，就好像Quartz 应用从来没有中断过似的。

注： 文中描述的 Quartz 应用 在一些语境下，又叫做 Quartz 服务器节点，都是同一个概念。

❏



2、创建

```
public static void main(String[] args) throws Exception {
    try {
        assignNewJob();
    } catch (ObjectAlreadyExistsException e) {
        System.err.println("发现任务已经在数据库存在了，直接从数据库里运行："+
e.getMessage());
        // TODO Auto-generated catch block
        resumeJobFromDatabase();
    }
}

private static void resumeJobFromDatabase() throws Exception {
    Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
    System.out.println("当前调度器的id
是: "+scheduler.getSchedulerInstanceId());
    scheduler.start();
    // 等待200秒，让前面的任务都执行完了之后，再关闭调度器
    Thread.sleep(200000);
    scheduler.shutdown(true);
}

private static void assignNewJob() throws SchedulerException,
InterruptedException {
    // 创建调度器
    Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
    // 定义一个触发器
    Trigger trigger = new Trigger().withIdentity("trigger1", "group1") // 定义
名称和所属的租
        .startNow()
        .withSchedule(simpleSchedule()
            .withIntervalInSeconds(15) // 每隔15秒执行一次
            .withRepeatCount(10)) // 总共执行11次(第一次执行不基数)
        .build();

    // 定义一个JobDetail
    JobDetail job = new Job(MailJob.class) // 指定干活的类MailJob
        .withIdentity("mailjob1", "mailgroup") // 定义任务名称和分组
        .usingJobData("email", "admin@10086.com") // 定义属性
        .build();

    // 调度加入这个job
    scheduler.scheduleJob(job, trigger);
    System.out.println("当前调度器的id
是: "+scheduler.getSchedulerInstanceId());

    // 启动
    scheduler.start();

    // 等待20秒，让前面的任务都执行完了之后，再关闭调度器
    Thread.sleep(20000);
    scheduler.shutdown(true);
}
```

3、修改配置文件

```
//追加 添加
org.quartz.jobStore.isClustered = true
org.quartz.scheduler.instanceName = quartzScheduler
org.quartz.scheduler.instanceId = AUTO
org.quartz.jobStore.clusterCheckinInterval = 1000
```

4、测试

发现任务已经在数据库存在了，直接从数据库里运行:Unable to store Job :
'mailgroup.mailjob1', because one already exists with this identification.
当前调度器的id是: DESKTOP-L3DGJ9G1576168479112
给邮件地址 admin@10086.com 发出了一封定时邮件，当前时间是: 00:34:39 (false)
给邮件地址 admin@10086.com 发出了一封定时邮件，当前时间是: 00:34:41 (false)
给邮件地址 admin@10086.com 发出了一封定时邮件，当前时间是: 00:34:43 (false)

