# WebSocket

## 一：webSocket演示

### 步骤1：效果

如图所示，当服务端有新的比特币价格后，浏览器马上就可以看到最新的数据



### 步骤2：WebSocket

在WebSocket概念出来之前，如果页面要不停地显示最新的价格，那么必须不停地刷新页面，或者用一段js代码每隔几秒钟发消息询问服务器数据。

而使用webSocket技术之后，当服务器有了新的数据，会主动通知浏览器。如效果所示，当服务端有新的比特币价格之后，浏览器立马接收到消息。

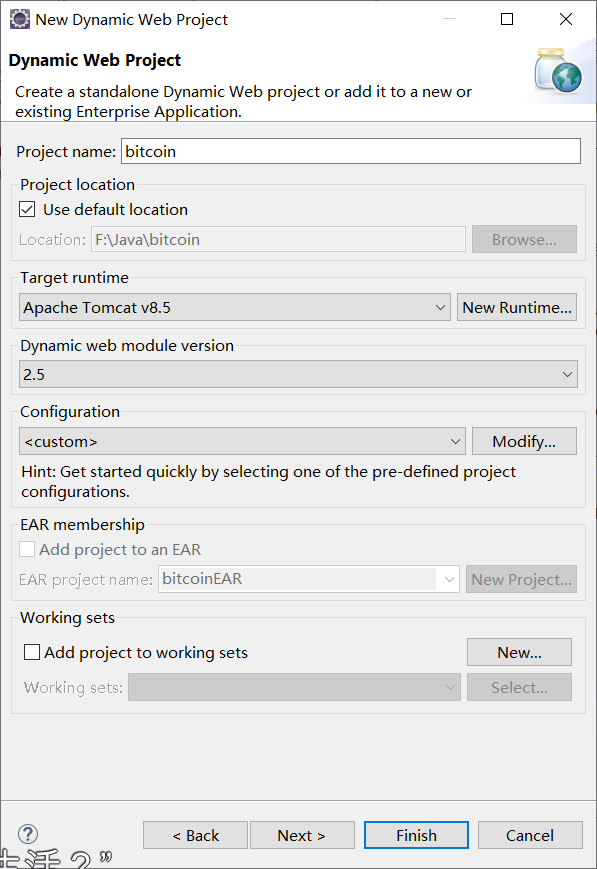
### 步骤3：优点

1. 节约宽带。不停地轮询服务端数据这种方式，使用的是http信息很大，有效数据占比低，而使用WebSocket方式，头信息很小，有效数据占比高。
2. 无浪费。WebSocket是由服务器主动回发，来的都是新数据。
3. 实时性：WebSocket是由服务器主动推送过来，实时性是最高的。

## 二：WebSocket开发流程

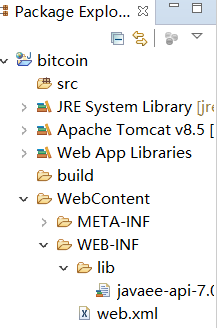
### 步骤1：首先创建动态web项目

菜单->File->New->Other->Web->Dynamic Web Project



### 步骤2：导入jar

为了支持WebSocket，需要引入javaee.jar,下载后放进WEB-INF/lib 目录下



### 步骤3：创建BitCoinServer

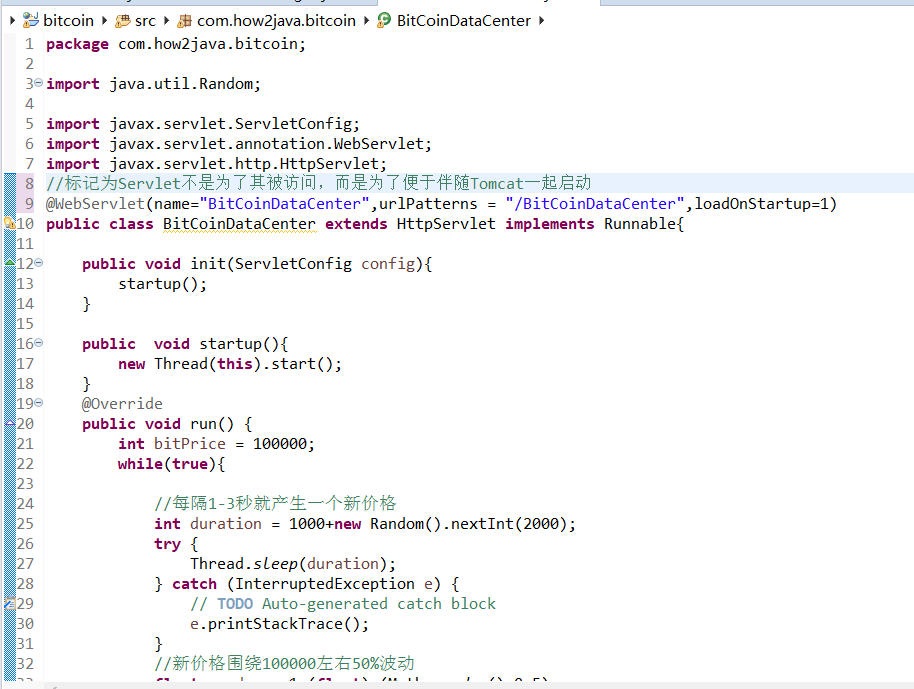


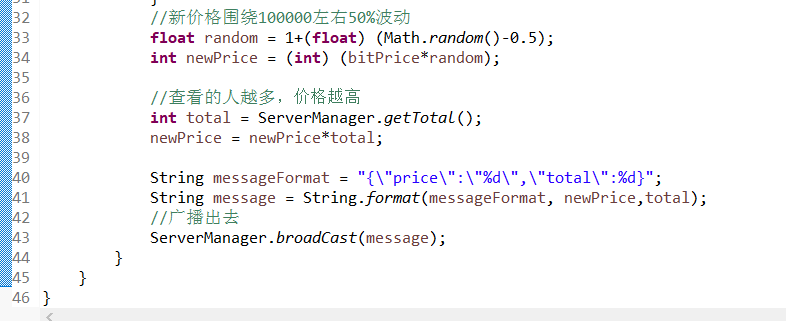
### 步骤4：创建ServerManager

ServerManager 中维护了一个线程安全的集合servers, 用于因为浏览器发起连接请求而创建的BitCoinServer.



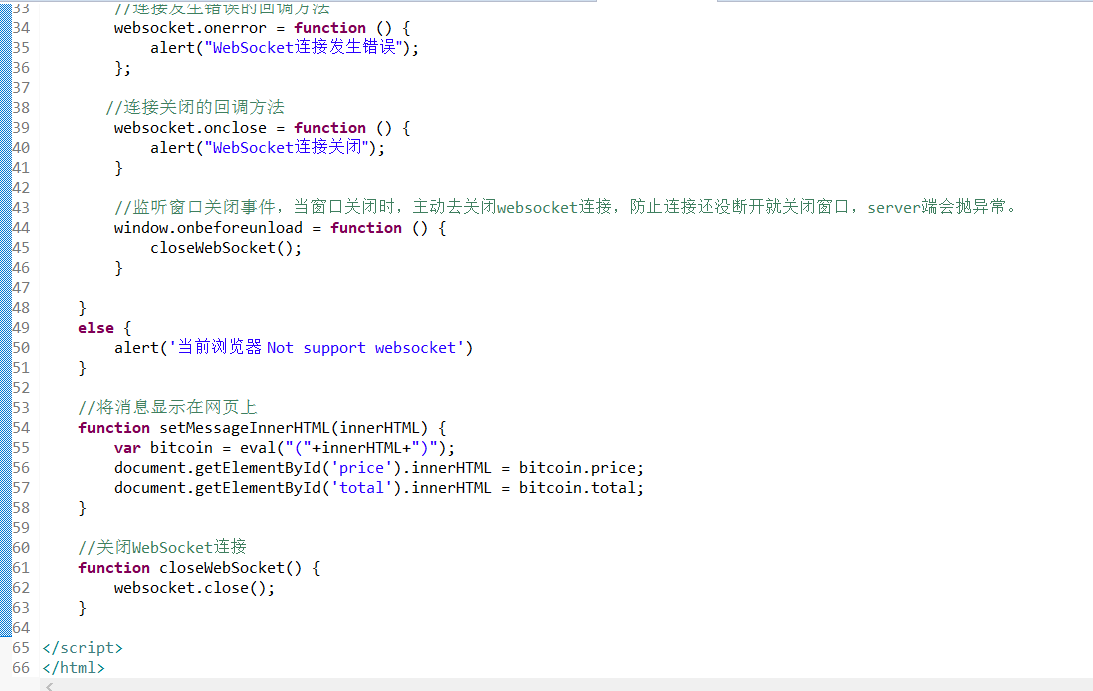
### 步骤5：创建BitCoinDataCenter，使其继承HttpServlet.





### 步骤6：在WebContent下创建index.jsp



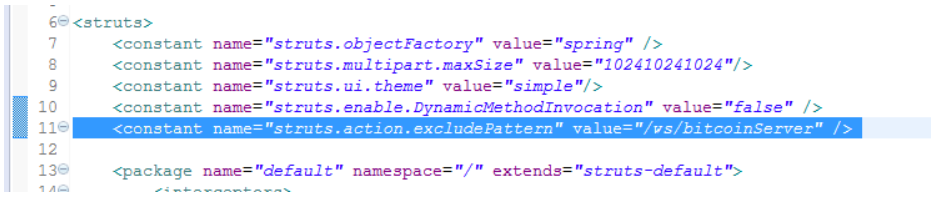


### 步骤7：启动tomcat，输入<http://127.0.0.1:8080/bitcoin>

## 三：WebSocket Struts注意事项

### 步骤1：struts配置文件

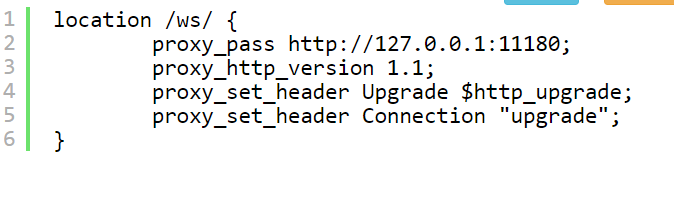
因为Struts会把所有的请求都拦截下来，以保证webSocket请求能够被正常捕捉住，不然就被Struts搞跑了



## 四：WebSocket nginx注意事项

### 步骤1：nginx.conf

如果做了nginx和tomcat整合的话，那么nginx 需要加上这么一段话，才能够正常的把webSocket请求交给tomcat,不然tomcat也不知道怎么处理



# QRCode

## 一：QRCode JAVA

### 步骤1：下载jar包

生成和解析二维码需要用到第三方的包： QRCODE.jar。

### 步骤2：代码

|  |
| --- |
| **package** qrcode;  **import** java.awt.Color;  **import** java.awt.Graphics2D;  **import** java.awt.image.BufferedImage;  **import** java.io.File;  **import** java.io.IOException;  **import** java.io.UnsupportedEncodingException;    **import** javax.imageio.ImageIO;    **import** com.swetake.util.Qrcode;    **import** jp.sourceforge.qrcode.QRCodeDecoder;  **import** jp.sourceforge.qrcode.data.QRCodeImage;  **import** jp.sourceforge.qrcode.exception.DecodingFailedException;    **public** **class** QRCodeUtil {    **public** **static** **void** qrCodeEncode(String encodeddata, File destFile) **throws** IOException {  Qrcode qrcode = **new** Qrcode();  qrcode.setQrcodeErrorCorrect('M'); // 纠错级别（L 7%、M 15%、Q 25%、H 30%）和版本有关  qrcode.setQrcodeEncodeMode('B');  qrcode.setQrcodeVersion(7); // 设置Qrcode包的版本    **byte**[] d = encodeddata.getBytes("GBK"); // 字符集  BufferedImage bi = **new** BufferedImage(139, 139, BufferedImage.***TYPE\_INT\_RGB***);  // createGraphics // 创建图层  Graphics2D g = bi.createGraphics();    g.setBackground(Color.***WHITE***); // 设置背景颜色（白色）  g.clearRect(0, 0, 139, 139); // 矩形 X、Y、width、height  g.setColor(Color.***BLACK***); // 设置图像颜色（黑色）    **if** (d.length > 0 && d.length < 123) {  **boolean**[][] b = qrcode.calQrcode(d);  **for** (**int** i = 0; i < b.length; i++) {  **for** (**int** j = 0; j < b.length; j++) {  **if** (b[j][i]) {  g.fillRect(j \* 3 + 2, i \* 3 + 2, 3, 3);  }  }  }  }    // Image img = ImageIO.read(new File("D:/tt.png")); logo  // g.drawImage(img, 25, 55,60,50, null);    g.dispose(); // 释放此图形的上下文以及它使用的所有系统资源。调用 dispose 之后，就不能再使用 Graphics 对象  bi.flush(); // 刷新此 Image 对象正在使用的所有可重构的资源    ImageIO.*write*(bi, "png", destFile);  // System.out.println("Input Encoded data is：" + encodeddata);  }    /\*\*  \* 解析二维码，返回解析内容  \*  \* **@param** imageFile  \* **@return**  \*/  **public** **static** String qrCodeDecode(File imageFile) {  String decodedData = **null**;  QRCodeDecoder decoder = **new** QRCodeDecoder();  BufferedImage image = **null**;  **try** {  image = ImageIO.*read*(imageFile);  } **catch** (IOException e) {  System.***out***.println("Error: " + e.getMessage());  }    **try** {  decodedData = **new** String(decoder.decode(**new** J2SEImage(image)), "GBK");  // System.out.println("Output Decoded Data is：" + decodedData);  } **catch** (DecodingFailedException dfe) {  System.***out***.println("Error: " + dfe.getMessage());  } **catch** (UnsupportedEncodingException e) {  e.printStackTrace();  }  **return** decodedData;  }    **static** **class** J2SEImage **implements** QRCodeImage {  BufferedImage image;    **public** J2SEImage(BufferedImage image) {  **this**.image = image;  }    **public** **int** getWidth() {  **return** image.getWidth();  }    **public** **int** getHeight() {  **return** image.getHeight();  }    **public** **int** getPixel(**int** x, **int** y) {  **return** image.getRGB(x, y);  }  }    **public** **static** **void** main(String[] args) {  String filePath = "d:/qrcode.png";  File qrFile = **new** File(filePath);    // 二维码内容  String encodeddata = "http://how2j.cn";  **try** {  QRCodeUtil.*qrCodeEncode*(encodeddata, qrFile);  } **catch** (IOException e) {  e.printStackTrace();  }    // 解码  String reText = QRCodeUtil.*qrCodeDecode*(qrFile);  System.***out***.println(reText);  }  } |

## 二.QRCode Javascipt

### 步骤 1 : 说明

javascript创建二维码图片

### 步骤2：js文件

首先需要第三方的js文件：jquery.qrcode.min.js

还需要jquery的js,

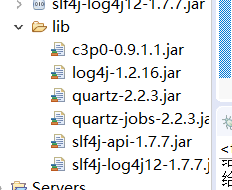
### 步骤3：html



# Quartz

## 一：quartz入门

### 步骤一：导入jar包



### 步骤二：创建TestQuartz

几个概念搞清楚先：

触发器：说明时候工作

任务job：做什么工作

调度器 Scheduler：搭配Trigger和Job

|  |
| --- |
| **package** com.how2java;    **import** **static** org.quartz.JobBuilder.*newJob*;  **import** **static** org.quartz.SimpleScheduleBuilder.*simpleSchedule*;  **import** **static** org.quartz.TriggerBuilder.*newTrigger*;    **import** org.quartz.JobDetail;  **import** org.quartz.Scheduler;  **import** org.quartz.Trigger;  **import** org.quartz.impl.StdSchedulerFactory;    **public** **class** TestQuartz {  **public** **static** **void** main(String[] args) **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 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();    //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**);  }  } |

### 步骤三：创建MailJob

|  |
| --- |
| package com.how2java;    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();  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%n" ,email, now);  }  } |

## 二：quzrtz Job管理

### 步骤1：job组成部分

Job 其实是由 3 个部分组成：  
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");

|  |
| --- |
| **package** com.how2java;    **import** **static** org.quartz.JobBuilder.*newJob*;  **import** **static** org.quartz.SimpleScheduleBuilder.*simpleSchedule*;  **import** **static** org.quartz.TriggerBuilder.*newTrigger*;    **import** org.quartz.InterruptableJob;  **import** org.quartz.JobDetail;  **import** org.quartz.Scheduler;  **import** org.quartz.SchedulerException;  **import** org.quartz.Trigger;  **import** org.quartz.impl.StdSchedulerFactory;    **public** **class** TestQuartz {  **public** **static** **void** main(String[] args) **throws** Exception{  *jobDataMap*();    }    **private** **static** **void** jobDataMap() **throws** SchedulerException, InterruptedException {  Scheduler scheduler = StdSchedulerFactory.*getDefaultScheduler*();    Trigger trigger = *newTrigger*().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(*simpleSchedule*()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = *newJob*(MailJob.**class**)  .withIdentity("mailjob1", "mailgroup")  .usingJobData("email", "admin@10086.com")  .build();    //用JobDataMap 修改email  job.getJobDataMap().put("email", "admin@taobao.com");    //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**);  }  } |

### 步骤2：job并发

默认的情况下，无论上一次任务是否结束或者完成，只要规定的时间到了，那么下一次就开始。  
  
有时候会做长时间的任务，比如数据库备份，这个时候就希望上一次备份成功结束之后，才开始下一次备份，即便是规定时间到了，也不能开始，因为这样很有可能造成 数据库被锁死 （几个线程同时备份数据库，引发无法预计的混乱）。  
  
那么在这种情况下，给数据库备份任务增加一个注解就好了：

@DisallowConcurrentExecution

创建DatabaseBackupJob类

|  |
| --- |
| package com.how2java;    import org.quartz.DisallowConcurrentExecution;  import org.quartz.Job;  import org.quartz.JobDetail;  import org.quartz.JobExecutionContext;  import org.quartz.JobExecutionException;    @DisallowConcurrentExecution  public class DatabaseBackupJob implements Job {  public void execute(JobExecutionContext context) throws JobExecutionException {  JobDetail detail = context.getJobDetail();  String database = detail.getJobDataMap().getString("database");    System.out.printf("给数据库 %s 备份, 耗时10秒 %n" ,database);  try {  Thread.sleep(10000);  } catch (InterruptedException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  } |

修改TestQuartz

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.SimpleScheduleBuilder.simpleSchedule;  import static org.quartz.TriggerBuilder.newTrigger;    import org.quartz.InterruptableJob;  import org.quartz.JobDetail;  import org.quartz.Scheduler;  import org.quartz.SchedulerException;  import org.quartz.Trigger;  import org.quartz.impl.StdSchedulerFactory;    public class TestQuartz {  public static void main(String[] args) throws Exception{  // jobDataMap();  databaseCurrentJob();    }    private static void databaseCurrentJob() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(DatabaseBackupJob.class)  .withIdentity("backupjob", "databasegroup")  .usingJobData("database", "how2java")  .build();    //调度加入这个job  scheduler.scheduleJob(job, trigger);    //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);  }    private static void jobDataMap() throws SchedulerException, InterruptedException {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(MailJob.class)  .withIdentity("mailjob1", "mailgroup")  .usingJobData("email", "admin@10086.com")  .build();    //用JobDataMap 修改email  job.getJobDataMap().put("email", "admin@taobao.com");    //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);  }  } |

### 步骤3：Job异常

任务里发生异常是很常见的。 异常处理办法通常是两种：   
1. 当异常发生，那么就通知所有管理这个 Job 的调度，停止运行它  
2. 当异常发生，修改一下参数，马上重新运行

创建ExceptionJob1

|  |
| --- |
| package com.how2java;    import org.quartz.Job;  import org.quartz.JobExecutionContext;  import org.quartz.JobExecutionException;    public class ExceptionJob1 implements Job {  public void execute(JobExecutionContext context) throws JobExecutionException {    int i = 0;  try {  //故意发生异常  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ExceptionJob2

|  |
| --- |
| package com.how2java;    import org.quartz.Job;  import org.quartz.JobExecutionContext;  import org.quartz.JobExecutionException;    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("运算结果"+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.setRefireImmediately(true);  throw je;  }  }  } |

创建TestQuartz

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.SimpleScheduleBuilder.simpleSchedule;  import static org.quartz.TriggerBuilder.newTrigger;    import org.quartz.InterruptableJob;  import org.quartz.JobDetail;  import org.quartz.Scheduler;  import org.quartz.SchedulerException;  import org.quartz.Trigger;  import org.quartz.impl.StdSchedulerFactory;    public class TestQuartz {  public static void main(String[] args) throws Exception{  // jobDataMap();  // databaseCurrentJob();  exceptionHandle1();  // exceptionHandle2();  }    private static void exceptionHandle2() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(ExceptionJob2.class)  .withIdentity("exceptionJob1", "someJobGroup")  .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);  }    private static void exceptionHandle1() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(ExceptionJob1.class)  .withIdentity("exceptionJob1", "someJobGroup")  .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);  }    private static void databaseCurrentJob() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(DatabaseBackupJob.class)  .withIdentity("backupjob", "databasegroup")  .usingJobData("database", "how2java")  .build();    //调度加入这个job  scheduler.scheduleJob(job, trigger);    //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);  }    private static void jobDataMap() throws SchedulerException, InterruptedException {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(MailJob.class)  .withIdentity("mailjob1", "mailgroup")  .usingJobData("email", "admin@10086.com")  .build();    //用JobDataMap 修改email  job.getJobDataMap().put("email", "admin@taobao.com");    //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);  }  } |

### 步骤4：中断Job

创建StoppableJob

|  |
| --- |
| package com.how2java;    import org.quartz.InterruptableJob;  import org.quartz.Job;  import org.quartz.JobExecutionContext;  import org.quartz.JobExecutionException;  import org.quartz.UnableToInterruptJobException;    //必须实现InterruptableJob 而非 Job才能够被中断  public class StoppableJob implements InterruptableJob {  private boolean stop = false;  public void execute(JobExecutionContext context) throws JobExecutionException {    while(true){    if(stop)  break;  try {  System.out.println("每隔1秒，进行一次检测，看看是否停止");  Thread.sleep(1000);  } catch (InterruptedException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  System.out.println("持续工作中。。。");  }    }  public void interrupt() throws UnableToInterruptJobException {  System.out.println("被调度叫停");  stop = true;  }  } |

修改TestQuartz

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.SimpleScheduleBuilder.simpleSchedule;  import static org.quartz.TriggerBuilder.newTrigger;    import org.quartz.InterruptableJob;  import org.quartz.JobDetail;  import org.quartz.Scheduler;  import org.quartz.SchedulerException;  import org.quartz.Trigger;  import org.quartz.impl.StdSchedulerFactory;    public class TestQuartz {  public static void main(String[] args) throws Exception{  // jobDataMap();  // databaseCurrentJob();  // exceptionHandle1();  // exceptionHandle2();  stop();    }    private static void stop() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .build();    //定义一个JobDetail  JobDetail job = newJob(StoppableJob.class)  .withIdentity("exceptionJob1", "someJobGroup")  .build();    //调度加入这个job  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  Thread.sleep(20000);  scheduler.shutdown(true);  }    private static void exceptionHandle2() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(ExceptionJob2.class)  .withIdentity("exceptionJob1", "someJobGroup")  .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);  }    private static void exceptionHandle1() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(ExceptionJob1.class)  .withIdentity("exceptionJob1", "someJobGroup")  .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);  }    private static void databaseCurrentJob() throws Exception {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(DatabaseBackupJob.class)  .withIdentity("backupjob", "databasegroup")  .usingJobData("database", "how2java")  .build();    //调度加入这个job  scheduler.scheduleJob(job, trigger);    //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);  }    private static void jobDataMap() throws SchedulerException, InterruptedException {  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Trigger trigger = newTrigger().withIdentity("trigger1", "group1")  .startNow()  .withSchedule(simpleSchedule()  .withIntervalInSeconds(2)  .withRepeatCount(10))  .build();    //定义一个JobDetail  JobDetail job = newJob(MailJob.class)  .withIdentity("mailjob1", "mailgroup")  .usingJobData("email", "admin@10086.com")  .build();    //用JobDataMap 修改email  job.getJobDataMap().put("email", "admin@taobao.com");    //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);  }  } |

## 三：quartz Simple Trigger

### 步骤1：SimpleTrigger

Trigger 就是触发器的意思，用来指定什么时间开始触发，触发多少次，每隔多久触发一次.  
SimpleTrigger 可以方便的实现一系列的触发机制。

### 步骤2：下一个8秒的倍数

创建TestQuartz

|  |
| --- |
| package com.how2java;    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;    public class TestQuartz {  public static void main(String[] args) throws Exception{  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();    // schedule it to run!  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);  }  } |

创建MailJob

|  |
| --- |
| package com.how2java;    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;    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) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  } |

### 步骤3: 10秒后运行

使用方法：

DateBuilder.futureDate()

可以方便的获取10秒后， 5分钟后， 3个小时候，2个月后这样的时间

修改TestQuartz

|  |
| --- |
| package com.how2java;    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.DateBuilder.IntervalUnit;  import org.quartz.impl.StdSchedulerFactory;    public class TestQuartz {  public static void main(String[] args) throws Exception{  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Date startTime = DateBuilder.futureDate(10, IntervalUnit.SECOND);    JobDetail job = newJob(MailJob.class).withIdentity("mailJob", "mailGroup").build();    SimpleTrigger trigger = (SimpleTrigger) newTrigger().withIdentity("trigger1", "group1").startAt(startTime).build();    // schedule it to run!  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);  }  } |

### 步骤4：累计ni，间隔n秒

修改TestQuartz

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.TriggerBuilder.newTrigger;  import static org.quartz.SimpleScheduleBuilder.simpleSchedule;    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;    public class TestQuartz {  public static void main(String[] args) throws Exception{  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)  .withSchedule(simpleSchedule()  .withRepeatCount(3)  .withIntervalInSeconds(1))  .build();    // schedule it to run!  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);  }  } |

### 步骤5：无限重复，间隔1秒

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.TriggerBuilder.newTrigger;  import static org.quartz.SimpleScheduleBuilder.simpleSchedule;    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;    public class TestQuartz {  public static void main(String[] args) throws Exception{  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)  .withSchedule(simpleSchedule()  .repeatForever()  .withIntervalInSeconds(1))  .build();    // schedule it to run!  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);  }  } |

## 三：quartz CronTrigger

### 步骤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。  
  
因为Cron 特别复杂，如果以前接触过，可以按照Cron表达式来做触发器，没有接触过，还是用SimpleTrigger吧。。。

### 步骤2：创建TestQuartz

|  |
| --- |
| package com.how2java;    import static org.quartz.CronScheduleBuilder.cronSchedule;  import static org.quartz.JobBuilder.newJob;  import static org.quartz.TriggerBuilder.newTrigger;  import static org.quartz.SimpleScheduleBuilder.simpleSchedule;    import java.util.Date;    import org.quartz.CronTrigger;  import org.quartz.DateBuilder;  import org.quartz.JobDetail;  import org.quartz.Scheduler;  import org.quartz.SimpleTrigger;  import org.quartz.impl.StdSchedulerFactory;    public class TestQuartz {  public static void main(String[] args) throws Exception{  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    Date startTime = DateBuilder.nextGivenSecondDate(null, 8);    JobDetail job = newJob(MailJob.class).withIdentity("mailJob", "mailGroup").build();    CronTrigger trigger = newTrigger().withIdentity("trigger1", "group1").withSchedule(cronSchedule("0/2 \* \* \* \* ?"))  .build();    // schedule it to run!  Date ft = scheduler.scheduleJob(job, trigger);    System.out.println("使用的Cron表达式是："+trigger.getCronExpression());  // 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);  }  } |

### 步骤3：理解Cron

0/2 \* \* \* \* ?

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；



### 步骤4：cron表达式举例



## 四：quartz Listener

### 步骤1：监听器

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，其他的用的相对较少，本知识主要讲解Job的。

### 步骤2：创建MailJobListener

MailJobListener 实现了 JobListener 接口，就必须实现如图所示的4个方法。



|  |
| --- |
| package com.how2java;    import org.quartz.JobExecutionContext;  import org.quartz.JobExecutionException;  import org.quartz.JobListener;    public class MailJobListener implements JobListener {    @Override  public String getName() {  // TODO Auto-generated method stub  return "listener of mail job";  }    @Override  public void jobExecutionVetoed(JobExecutionContext context) {  // TODO Auto-generated method stub  System.out.println("取消执行：\t "+context.getJobDetail().getKey());  }    @Override  public void jobToBeExecuted(JobExecutionContext context) {  // TODO Auto-generated method stub  System.out.println("准备执行：\t "+context.getJobDetail().getKey());  }    @Override  public void jobWasExecuted(JobExecutionContext context, JobExecutionException arg1) {  // TODO Auto-generated method stub  System.out.println("执行结束：\t "+context.getJobDetail().getKey());  System.out.println();  }    } |

### 步骤3：TestQuartz

修改TestQuartz，于34-36行增加监听器

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.SimpleScheduleBuilder.simpleSchedule;  import static org.quartz.TriggerBuilder.newTrigger;    import org.quartz.JobDetail;  import org.quartz.JobKey;  import org.quartz.Scheduler;  import org.quartz.Trigger;  import org.quartz.impl.StdSchedulerFactory;  import org.quartz.impl.matchers.KeyMatcher;    public class TestQuartz {  public static void main(String[] args) throws Exception{  //创建调度器  Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();    //定义一个触发器  Trigger trigger = newTrigger().withIdentity("trigger1", "group1") //定义名称和所属的租  .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@10086.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);    //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);  }  } |

## 五：quartz JDBC Store

### 步骤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。 好处是就算系统重启了，目前运行到第几次了这些信息都是存放在数据库中的，那么就可以继续原来的步伐把计划任务无缝地继续做下去。 坏处就是性能上比内存慢一些，毕竟数据库读取总是要慢一些的。

### 步骤2：建表sql

为了能够把相关信息存放进 mysql 数据库里，必须手动建立数据库和表，使用如下 脚本就行了。   
**注：** 这里使用的数据库名称是 quartz

|  |
| --- |
| 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\_CALENDARS;    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; |

### 步骤3：配置文件

在src下新建 quartz.properties 配置文件，里面指定使用 JobStoreTX 方式管理任务。 并且指定链接数据库的账号密码等信息

|  |
| --- |
| 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 |

### 步骤4：MailJob

|  |
| --- |
| package com.how2java;    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());  }  } |

### 步骤5：TestQuartz

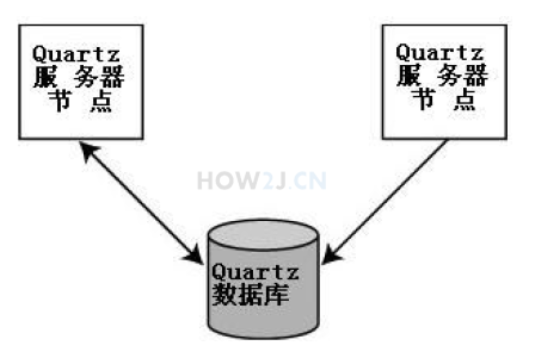
新增加了一个resumeJobFromDatabase 方法，当使用原来的方式增加任务报异常的时候，就直接从数据库重跑任务。  
  
任务的触发也调整成了15秒一次，总共11次。

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.SimpleScheduleBuilder.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;    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());  // TODO Auto-generated catch block  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 = newTrigger().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 = 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();    // 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);  }  } |

## 六：quartz Cluster

### 步骤1：集群概念

这所谓的Quartz集群，是指在 [基于数据库存储 Quartz调度信息](https://how2j.cn/k/quartz/quartz-jdbcstore/1713.html) 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。



### 步骤2：修改TestQuartz

|  |
| --- |
| package com.how2java;    import static org.quartz.JobBuilder.newJob;  import static org.quartz.SimpleScheduleBuilder.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;    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());  // 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 assginNewJob() throws SchedulerException, InterruptedException {  // 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 = 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();    // 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);  }  } |

### 步骤3：quartz.properties

quartz.properties 在原来的基础上，增加4行：  
开启集群

org.quartz.jobStore.isClustered = true

要进行集群，多个应用调度名称 instanceName 应该是一样的

org.quartz.scheduler.instanceName = quartzScheduler

要进行集群，多个应用调度id instanceId 必须不一样，这里使用AUTO，就会自动分配不同的ID。 目测是本机机器名称加上时间戳

org.quartz.scheduler.instanceId = AUTO

每个一秒钟去数据库检查一下，以在其他应用挂掉之后及时补上

org.quartz.jobStore.clusterCheckinInterval = 1000

|  |
| --- |
| org.quartz.jobStore.isClustered = true  org.quartz.scheduler.instanceName = quartzScheduler  org.quartz.scheduler.instanceId = AUTO  org.quartz.jobStore.clusterCheckinInterval = 1000    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 = admin  org.quartz.dataSource.mysqlDatabase.maxConnections = 5 |