

深入解读Quartz的原理

Quartz是一个大名鼎鼎的Java版开源定时调度器，功能强悍，使用方便。

一、核心概念

Quartz的原理不是很复杂，只要搞明白几个概念，然后知道如何去启动和关闭一个调度程序即可。

1、Job

表示一个工作，要执行的具体内容。此接口中只有一个方法
`void execute(JobExecutionContext context)`

2、JobDetail

JobDetail表示一个具体的可执行的调度程序，Job是这个可执行调度程序所要执行的内容，另外JobDetail还包含了这个任务调度的方案和策略。

3、Trigger代表一个调度参数的配置，什么时候去调。

4、Scheduler代表一个调度容器，一个调度容器中可以注册多个JobDetail和Trigger。当Trigger与JobDetail组合，就可以被Scheduler容器调用了。

二、一个最简单入门实例

```
import org.quartz.*;
import org.quartz.impl.StdSchedulerFactory;

import java.util.Date;

/**
 * quartz定时器测试
 *
 * @author leizhimin 2009-7-23 8:49:01
 */
public class MyJob implements Job {
    public void execute(JobExecutionContext jobExecutionContext) throws JobExecutionException {
        System.out.println(new Date() + ": doing something...");
    }
}

class Test {
    public static void main(String[] args) {
        //1、创建JobDetail对象
        JobDetail jobDetail = new JobDetail();
        //设置工作项
        jobDetail.setJobClass(MyJob.class);
        jobDetail.setName("MyJob_1");
        jobDetail.setGroup("JobGroup_1");

        //2、创建Trigger对象
        SimpleTrigger strigger = new SimpleTrigger();
        strigger.setName("Trigger_1");
        strigger.setGroup("Trigger_Group_1");
        strigger.setStartTime(new Date());
        //设置重复停止时间，并销毁该Trigger对象
        java.util.Calendar c = java.util.Calendar.getInstance();
        c.setTimeInMillis(System.currentTimeMillis() + 1000 * 1L);
        strigger.setEndTime(c.getTime());
        strigger.setFireInstanceId("Trigger_1_id_001");
        //设置重复间隔时间
        strigger.setRepeatInterval(1000 * 1L);
        //设置重复执行次数
        strigger.setRepeatCount(3);

        //3、创建Scheduler对象，并配置JobDetail和Trigger对象
        SchedulerFactory sf = new StdSchedulerFactory();
        Scheduler scheduler = null;
        try {
            scheduler = sf.getScheduler();
            scheduler.scheduleJob(jobDetail, strigger);
            //4、并执行启动、关闭等操作
            scheduler.start();
        } catch (SchedulerException e) {
            e.printStackTrace();
        }

        // try {
        //     //关闭调度器
        //     scheduler.shutdown(true);
        // } catch (SchedulerException e) {
        //     e.printStackTrace();
        // }
```

```
//    }
}
```

执行结果:

```
Run ▶ Test
D:\jdk15\bin\java -Didea.launcher.port=7533 -Didea.launcher.b...
2009-07-23 09:59:51 - Job execution threads will use cla...
2009-07-23 09:59:51 - Initialized Scheduler Signaller of
2009-07-23 09:59:51 - Quartz Scheduler v.1.6.1 created.
2009-07-23 09:59:51 - RAMJobStore initialized.
2009-07-23 09:59:51 - Quartz scheduler 'DefaultQuartzSch...
2009-07-23 09:59:51 - Quartz scheduler version: 1.6.1
2009-07-23 09:59:51 - Scheduler DefaultQuartzScheduler_$...
Thu Jul 23 09:59:51 CST 2009: doing something...
Thu Jul 23 09:59:52 CST 2009: doing something...
Thu Jul 23 09:59:53 CST 2009: doing something...
Thu Jul 23 09:59:54 CST 2009: doing something...
程序执行达到最大次数4次（执行1次，重复3次）后，处于休眠态。并没有退出。只有显式的调用scheduler.shutdown(true);方法后，才能退出。
```

当把结束时间改为:

```
//设置重复停止时间，并销毁该Trigger对象
java.util.Calendar c = java.util.Calendar.getInstance();
c.setTimeInMillis(System.currentTimeMillis() + 1000 * 1L);
trigger.setEndTime(c.getTime());
```

执行结果:

```
Run ▶ Test
D:\jdk15\bin\java -Didea.launcher.port=7532 -Didea.launcher.bin.pa...
2009-07-23 10:04:37 - Job execution threads will use class lc
2009-07-23 10:04:37 - Initialized Scheduler Signaller of type
2009-07-23 10:04:37 - Quartz Scheduler v.1.6.1 created.
2009-07-23 10:04:37 - RAMJobStore initialized.
2009-07-23 10:04:37 - Quartz scheduler 'DefaultQuartzSchedule
2009-07-23 10:04:37 - Quartz scheduler version: 1.6.1
2009-07-23 10:04:37 - Scheduler DefaultQuartzScheduler_$_NON_
Thu Jul 23 10:04:37 CST 2009: doing something...
Thu Jul 23 10:04:38 CST 2009: doing something...
本次设定了重复执行3次，可结果看，仅仅重复执行了一次！
当设定了重复停止时间很短，不足以完成所设定的重复次数，则在停止时间
终止重复，而不管重复是否达到最大次数。
```

当添加一条关闭调度器的语句:

```
//4、并执行启动、关闭等操作
scheduler.start();
scheduler.shutdown(true);
```

程序执行结果:

Thu Jul 23 10:11:50 CST 2009: doing something...

Process finished with exit code 0

仅仅执行了一次，这一次能执行完，原因是设定了scheduler.shutdown(true);true表示等待本次任务执行完成后停止。

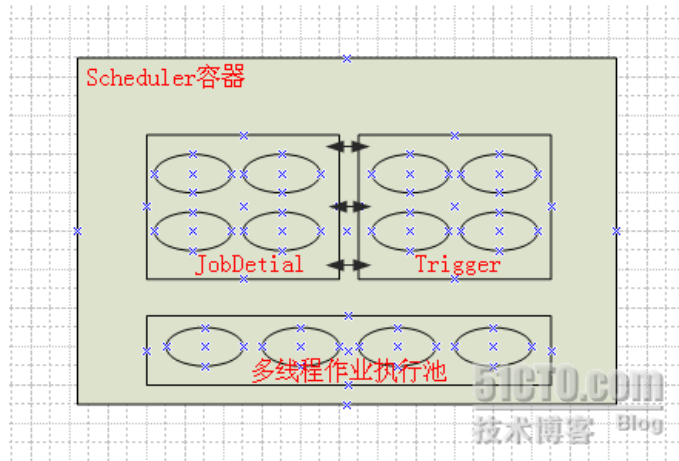
从这里也可以看出，scheduler是个容器，scheduler控制jobDetail的执行，控制的策略是通过trigger。

当scheduler容器启动后，jobDetail才能根据关联的trigger策略去执行。当scheduler容器关闭后，所有的jobDetail都停止执行。

三、透过实例看原理

通过研读Quartz的源代码，和本实例，终于悟出了Quartz的工作原理。

- 1、scheduler是一个计划调度器容器（总部），容器里面可以盛放众多的JobDetail和trigger，当容器启动后，里面的每个JobDetail都会根据trigger按部就班自动去执行。
- 2、JobDetail是一个可执行的工作，它本身可能是有状态的。
- 3、Trigger代表一个调度参数的配置，什么时候去调。
- 4、当JobDetail和Trigger在scheduler容器上注册后，形成了装配好的作业（JobDetail和Trigger所组成的一对儿），就可以伴随容器启动而调度执行了。
- 5、scheduler是个容器，容器中有一个线程池，用来并行调度执行每个作业，这样可以提高容器效率。
- 6、将上述的结构用一个图来表示，如下：



四、总结

- 1、搞清楚了上Quartz容器执行作业的的原理和过程，以及作业形成的方式，作业注册到容器的方法。就认识明白了Quartz的核心原理。
- 2、Quartz虽然很庞大，但是一切都围绕这个核心转，为了配置强大时间调度策略，可以研究专门的CronTrigger。要想灵活配置作业和容器属性，可以通过Quartz的properties文件或者XML来实现。
- 3、要想调度更多的持久化、结构化作业，可以通过数据库读取作业，然后放到容器中执行。
- 4、所有的一切都围绕这个核心原理转，搞明白这个了，再去研究更高级用法就容易多了。
- 5、Quartz与Spring的整合也非常简单，Spring提供一组Bean来支持：MethodInvokingJobDetailFactoryBean、SimpleTriggerBean、SchedulerFactoryBean，看看里面需要注入什么属性即可明白了。Spring会在Spring容器启动时候，启动Quartz容器。
- 6、Quartz容器的关闭方式也很简单，如果是Spring整合，则有两种方法，一种是关闭Spring容器，一种是获取到SchedulerFactoryBean实例，然后调用一个shutdown就搞定了。如果是Quartz独立使用，则直接调用scheduler.shutdown(true)；
- 7、Quartz的JobDetail、Trigger都可以在运行时重新设置，并且在下次调用时候起作用。这就为动态作业的实现提供了依据。你可以将调度时间策略存放到数据库，然后通过数据库数据来设定Trigger，这样就能产生动态的调度。