Quartz框架

讲师：陈伟俊

一个Java定时器的例子

```
public class TestTimer {
    public static void main(String[] args) {
        //定时器对象
        Timer timer = new Timer();
        //在1秒后执行此任务,每次间隔2秒
        timer.schedule(new MyTask(), 1000, 2000);
    }
    //自定义任务
    public static class MyTask extends TimerTask {
        @Override
        public void run() {
            System.out.println("执行。。。。" + new Date());
        }
    }
}
```

Quartz是什么？

- Quartz是OpenSymphony开源组织在Job scheduling领域又一个开源项目,它可以与J2EE与J2SE应用程序相结合也可以单独使用。Quartz可以用来创建简单或为运行十个，百个，甚至是好几万个Jobs这样复杂的日程序表。Jobs可以做成标准的Java组件或 EJBs。
- Quartz是一个任务日程管理系统，一个在预先确定（被纳入日程）的时间到达时，负责执行（或者通知）其他软件组件的系统。
- Quartz用一个小Java库发布文件（.jar文件），这个库文件包含了所有Quartz核心功能。这些功能的主要接口(API)是Scheduler接口。它提供了简单的操作，例如：将任务纳入日程或者从日程中取消，开始/停止/暂停日程进度。

Quartz体系结构

- 核心接口
  - Scheduler
  - Job
  - JobDetail
  - Trigger
  - JobDataMap

Job

- 是一个接口，
- 只有一个方法void execute(JobExecutionContext context)，开发者实现该接口定义运行任务，JobExecutionContext类提供了调度上下文的各种信息。Job运行时的信息保存在 JobDataMap实例中

JobDetail

- Quartz在每次执行Job时，都重新创建一个Job实例，所以它不直接接受一个Job的实例，相反它接收一个Job实现类，以便运行时通过newInstance()的反射机制实例化Job。因此需要通过一个类来描述Job的实现类及其它相关的静态信息，如Job名字、描 述、关联监听器等信息，JobDetail承担了这一角色。
- 通过该类的构造函数可以更具体地了解它的功用：JobDetail(java.lang.String name, java.lang.String group, java.lang.Class jobClass)，该构造函数要求指定Job的实现类，以及任务在Scheduler中的组名和Job名称

Trigger

- 是一个类，描述触发Job执行的时间触发规则。主要有SimpleTrigger和CronTrigger这两个子类。当仅需触 发一次或者以固定时间间隔周期执行，SimpleTrigger是最适合的选择；而CronTrigger则可以通过Cron表达式定义出各种复杂时间规 则的调度方案：如每早晨9:00执行，周一、周三、周五下午5:00执行等

Scheduler

- 代表一个Quartz的独立运行容器，Trigger和JobDetail可以注册到Scheduler中，两者在 Scheduler中拥有各自的组及名称，组及名称是Scheduler查找定位容器中某一对象的依据，Trigger的组及名称必须唯 一，JobDetail的组和名称也必须唯一（但可以和Trigger的组和名称相同，因为它们是不同类型的）。Scheduler定义了多个接口方法， 允许外部通过组及名称访问和控制容器中Trigger和JobDetail。
- Scheduler可以将Trigger绑定到某一JobDetail中，这样当Trigger触发时，对应的Job就被执行。一个Job可以对应 多个Trigger，但一个Trigger只能对应一个Job。可以通过SchedulerFactory创建一个Scheduler实例。 Scheduler拥有一个SchedulerContext，它类似于ServletContext，保存着Scheduler上下文信息，Job和 Trigger都可以访问SchedulerContext内的信息。SchedulerContext内部通过一个Map，以键值对的方式维护这些上下 文数据，SchedulerContext为保存和获取数据提供了多个put()和getXxx()的方法。可以通过Scheduler# getContext()获取对应的SchedulerContext实例

JobDataMap

- 可以用来保存任何需要传递给任务实例的对象（这些对象要求是可序列化的），JobDataMap是java的Map接口的实现

定时执行任务示例

- 示例：执行一个调度，每隔10秒中执行一段代码

FirstJob.java（一个封装了执行任务的类）

```
public class FirstJob implements Job {
    public void execute(JobExecutionContext context) throws JobExecutionException {
        //获得job详细
        JobDetail jobDetail = context.getJobDetail();

        //获得job名称
        String jobName = jobDetail.getName();
        System.out.println("jobName=" + jobName);

        //获得运行时参数
        JobDataMap jobDataMap = jobDetail.getJobDataMap();
        String param1 = jobDataMap.getString("param1");
        System.out.println("param1=" + param1);

        System.out.println("当前时间：" + new Date());
        System.out.println("-------------------------");
    }
}
```

FirstScheduler.java（执行一个调度）

```
public class FirstScheduler {
    public static void main(String[] args) throws SchedulerException {
        //获得一个日程对象
        Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
        //创建一个JobDetail对象，参数分别为：任务名称，组名称，任务类
        JobDetail jobDetail = new JobDetail("我的第一个任务",Scheduler.DEFAULT_GROUP,FirstJob.class);

        //获得任务数据集合对象
        JobDataMap jobDataMap = jobDetail.getJobDataMap();
        //设置一些运行时用到的参数值
        jobDataMap.put("param1", "tom");
        /*获得一个触发器对象，设置每隔10秒执行一次,
        TriggerUtils.makeMinutelyTrigger(10);表示每隔10分钟执行一次，以此类推*/
        Trigger trigger = TriggerUtils.makeSecondlyTrigger(10);
        //设置触发器名称
        trigger.setName("firstTrigger");
        //设置第一次触发时间，默认立即触发第一次
        //trigger.setStartTime(new Date());
        //日程开始！
        scheduler.start();
        //注册任务，可以在开始任务之后注册，也可以在之前注册
        scheduler.scheduleJob(jobDetail, trigger);

        //日程结束
        //scheduler.shutdown();
    }
```

Spring整合quartz

- 正常情况下，一个任务类必须实现Job接口，且覆盖execute方法，才能被调度所调用。
- 在Spring中集成了Quartz框架，可以把任何一个bean的任意方法作为一个任务执行，而这个bean类无需实现任何接口，无需覆盖任何方法。
- Spring可以把编程式调度变成声明式调度，只需要在xml文件中做一段配置。

Spring集成quartz框架示例

- 通过Spring配置做一个简单触发器

任务bean1：TestService.java

```
public class TestService {
    public void doSomething() {
        System.out.println("Do Something Freely!");
    }

    public void justDoIt() {
        System.out.println("Just Do It!");
    }
}
```

Spring核心配置文件：applicationContext.xml

```
<!-- 普通的业务Bean -->
<bean id="testService" class="com.icss.TestService" />

<!-- 作业 -->
<bean id="jobDetail_test”
        class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
        <property name="targetObject" ref="testService" />
        <property name="targetMethod" value="justDoIt" />
</bean>
```

Spring核心配置文件：applicationContext.xml

```
<!-- 简单触发器，一般都是每隔多少毫秒执行一次-->
<bean name="testTrigger"
        class="org.springframework.scheduling.quartz.SimpleTriggerBean">
        <property name="jobDetail" ref="jobDetail_test" />
        <!-- 第一次启动延时(单位均为毫秒) -->
        <property name="startDelay" value="10" />
        <!-- 间隔时间 -->
        <property name="repeatInterval" value="2000" />
        <!-- 执行次数，默认没有限制 -->
        <property name="repeatCount" value="10" />
        <!-- 传递的参数 -->
        <property name="jobDataAsMap">
                <map>
                        <entry key="count" value="10" />
                </map>
        </property>
</bean>
```

Spring核心配置文件：applicationContext.xml

```
<!-- 计划调度-->
<bean name="testScheduler"
        class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
                <list>
                        <ref bean="testTrigger" />
                </list>
        </property>
        <property name="schedulerContextAsMap">
                <map>
                        <entry key="timeout" value="30" />
                </map>
        </property>

</bean>
```

模拟spring容器初始化：TestQuartz.java

```
public class TestQuartz {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                                    "applicationContext.xml");
        System.out.println("Main方法执行开始了！ 定时器伴随着Spring的初始化执行了。。。");
        System.out.println("Main方法执行结束了！");
    }

}
```

Spring集成quartz框架示例2

- 通过Spring配置做一个复杂触发器
- 需要用到cron表达式

任务bean2：TestService2.java

```
public class TestService2 {
    public void doSomething() {
        System.out.println("Do Something Freely!");
    }

    public void justDoIt() {
        System.out.println("正在执行任务2!" + new Date());
    }
}
```

Spring核心配置文件：applicationContext.xml

```
<!-- 普通的业务Bean2 -->
<bean id="testService2" class="com.icss.TestService2" />

<!-- 作业2 -->
<bean id="jobDetail_test2"
        class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
        <property name="targetObject" ref="testService2" />
        <property name="targetMethod" value="justDoIt" />
</bean>
```

Spring核心配置文件：applicationContext.xml

```
<!-- 复杂触发器 -->
    <bean id="testTrigger2"
    class="org.springframework.scheduling.quartz.CronTriggerBean">
            <property name="jobDetail">
                    <ref bean="jobDetail_test2" />
            </property>

            <!-- cron表达式，以下是在每分钟的10秒，15秒，20秒,45秒执行 -->
            <property name="cronExpression">
                    <value>10,15,20,45 * * * * ?</value>
            </property>
    </bean>
```

Spring核心配置文件：applicationContext.xml

```
<!-- 计划 -->
<bean name="testScheduler"
        class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
                <list>
                        <ref bean="testTrigger" />
                        <ref bean="testTrigger2" />
                </list>
        </property>
        <property name="schedulerContextAsMap">
                <map>
                        <entry key="timeout" value="30" />
                </map>
        </property>
</bean>
```

模拟spring容器初始化：TestQuartz.java

```
public class TestQuartz {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                                    "applicationContext.xml");
        System.out.println("Main方法执行开始了！ 定时器伴随着Spring的初始化执行了。。。");
        System.out.println("Main方法执行结束了！");
    }

}
```

cron表达式

- 一个cron表达式有至少6个（也可能7个）有空格分隔的时间元素。
-
- 按顺序依次为：
  秒（0~59）
  分钟（0~59）
  小时（0~23）
  天（月）（0~31，但是你需要考虑你月的天数）
  月（0~11）
  天（星期）（1~7 1=SUN 或 SUN，MON，TUE，WED，THU，FRI，SAT）
  年份（1970－2099）可省略不写，默认每一年

cron表达式举例

0 0 10,14,16 * * ? 每天上午10点，下午2点，4点
0 0/30 9-17 * * ？ 朝九晚五工作时间内每半小时
0 0 12 ? * WED 表示每个星期三中午12点
"0 0 12 * * ?" 每天中午12点触发
"0 15 10 ? * *" 每天上午10:15触发
"0 15 10 * * ?" 每天上午10:15触发
"0 15 10 * * ? *" 每天上午10:15触发
"0 15 10 * * ? 2005" 2005年的每天上午10:15触发
"0 * 14 * * ?" 在每天下午2点到下午2:59期间的每1分钟触发
"0 0/5 14 * * ?" 在每天下午2点到下午2:55期间的每5分钟触发
"0 0/5 14,18 * * ?" 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发
"0 0-5 14 * * ?" 在每天下午2点到下午2:05期间的每1分钟触发

cron表达式举例

"0 10,44 14 ? 3 WED" 每年三月的星期三的下午2:10和2:44触发
"0 15 10 ? * MON-FRI" 周一至周五的上午10:15触发
"0 15 10 15 * ?" 每月15日上午10:15触发
"0 15 10 L * ?" 每月最后一日的上午10:15触发
"0 15 10 ? * 6L" 每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6L 2002-2005" 2002年至2005年的每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6#3" 每月的第三个星期五上午10:15触发

子表达式能包含一些范围或列表

'*' 字符可以用于所有字段，在“分”字段中设为"*"表示"每一分钟"的含义。

'?' 字符可以用在“日”和“周几”字段. 它用来指定 '不明确的值'。这在你需要指定这两个字段

中的某一个值而不是另外一个的时候会被用到。在后面的例子中可以看到其含义。

'-' 字符被用来指定一个值的范围，比如在“小时”字段中设为"10-12"表示"10点到12点"。

',' 字符指定数个值。比如在“周几”字段中设为"MON,WED,FRI"表示"the days Monday,

Wednesday, and Friday"。

'/' 字符用来指定一个值的的增加幅度。比如在“秒”字段中设置为"0/15"表示"第0, 15, 30, 和

45秒"。而 "5/15"则表示"第5, 20, 35, 和 50"。 在'/'前加"*"字符相当于指定从0秒开始。 每个

字段都有一系列可以开始或结束的数值。对于“秒”和“分”字段来说，其数值范围为0到59，

对于“小时”字段来说其为0到23, 对于“日”字段来说为0到31, 而对于“月”字段来说为1到12。

子表达式能包含一些范围或列表

"/“ 字段仅仅只是帮助你在允许的数值范围内从开始"第n"的值。 因此 对于“月”字段来说"7/6"

只是表示7月被开启而不是“每六个月”, 请注意其中微妙的差别。

'L‘ 字符可用在“日”和“周几”这两个字段。它是"last"的缩写, 但是在这两个字段中有不同的含义。

例如,“日”字段中的"L"表示"一个月中的最后一天" —— 对于一月就是31号对于二月来

说就是28号（非闰年）。

而在“周几”字段中, 它简单的表示"7" or "SAT"，但是如果在“周几”

字段中使用时跟在某个数字之后, 它表示"该月最

后一个星期×" —— 比如"6L"表示"该月最后一个周五"。

如果“日”字段中在“L”前有具体的内容，它就具有其他的含义了

例如：“6L”表示这个月的倒数第 6 天，“ＦＲＩＬ”表示这个月的最一个星期五

子表达式能包含一些范围或列表

'W' 可用于“日”字段。用来指定历给定日期最近的工作日(周一到周五) 。比如你将“日”字段

设为"15W"，意为: "离该月15号最近的工作日"。因此如果15号为周六，触发器会在14号

即周五调用。如果15号为周日, 触发器会在16号也就是周一触发。如果15号为周二,

那么当天就会触发。然而如果你将“日”字段设为"1W", 而一号又是周六, 触发器会于下周一也就

是当月的3号触发,因为它不会越过当月的值的范围边界。'W'字符只能用于“日”字段的值为

单独的一天而不是一系列值的时候。

'L'和'W' 可以组合用于“日”字段表示为'LW'，意为"该月最后一个工作日"。

'#' 字符可用于“周几”字段。该字符表示“该月第几个周×”，比如"6#3"表示该月第三个周五(6表示周五而"#3"该月第三个)。

再比如: "2#1" = 表示该月第一个周一而 "4#5" = 该月第五个周三。

注意如果你指定"#5"该月没有第五个“周×”，该月是不会触发的。