1. 什么是 Quartz
    1. Quartz 是一个开源的作业调度框架,由 java 编写,在.NET 平台为 Quartz.Net,通过 Quart 可以快速完成任务调度的工作.
2. Quartz 能干什么/应用场景
    1. 如网页游戏中挂机自动修炼如 8 个小时,人物相关数值进行成长,当使用某道具后,时间减少到 4 个小时,人物对应获得成长值.这其中就涉及到了 Scheduler 的操作，定时对人物进行更新属性操作，更改定时任务执行时间.
    2. 网页游戏中会大量涉及到 Scheduler 的操作,有兴趣的朋友可自行联想.
    3. 企业中如每天凌晨 2 点触发数据同步、发送 Email 等操作
3. 同类框架对比
    1. TimeTask TimeTask 在 Quartz 前还是显得过于简单、不完善，不能直接满足开发者的较为复杂的应用场景.
4. 资源
    1. 官网:http://www.quartz-scheduler.org/
    2. 下载:http://www.quartz-scheduler.org/downloads
    3. maven pom
        1. <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
            <version>2.2.1</version>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz-jobs</artifactId>
            <version>2.2.1</version>
        </dependency>
    4. 源代码 svn:http://svn.terracotta.org/svn/quartz
    5. 本文章采用的是 2.21 版本:CSDN 下载：http://download.csdn.net/detail/chenweitang123/7636703
    6. 例子 Demo:CSDN 下载：整理完后上传.
5. 框架分析
    1. 接口
    2. 类图
6. Quartz 中的设计模式
    1. Builder 模式
        1. 所有关键组件都有 Builder 模式来构建 <Builder> 如:JobBuilder、TriggerBuilder
    2. Factory 模式
        1. 最终由 Scheduler 的来进行组合各种组件 <Factory> 如 SchedulerFactory
    3. Quartz 项目中大量使用组件模式，插件式设计，可插拔，耦合性低，易扩展，开发者可自行定义自己的 Job、Trigger 等组件
    4. 链式写法,Quartz 中大量使用链式写法，与 jQuery 的写法有几分相似，实现也比较简单，如：

1. $(this).addClass("divCurrColor").next(".divContent").css("display","block");
2. *newTrigger*().withIdentity("trigger3", "group1").startAt(startTime)
   .withSchedule(*simpleSchedule*().withIntervalInSeconds(10).withRepeatCount(10)).build();

7. 框架核心分析
   1. SchedulerFactory -- 调度程序工厂
      1. StdSchedulerFactory -- Quartz 默认的 SchedulerFactory
      2. DirectSchedulerFactory -- DirectSchedulerFactory 是对 SchedulerFactory 的直接实现,通过它可以直接构建 Scheduler、threadpool 等
         1. ThreadExecutor / DefaultThreadExecutor -- 内部线程操作对象
   2. JobExecutionContext -- JOB 上下文,保存着 Trigger、 JobDeaitl 等信息,JOB 的 execute 方法传递的参数就是对象的实例
      1. JobExecutionContextImpl
   3. Scheduler -- 调度器
      1. StdScheduler -- Quartz 默认的 Scheduler
      2. RemoteScheduler -- 带有 RMI 功能的 Scheduler
   4. JOB --任务对象
      1. JobDetail -- 他是实现轮询的一个的回调类，可将参数封装成 JobDataMap 对象,Quartz 将任务的作业状态保存在 JobDetail 中.
      2. JobDataMap -- JobDataMap 用来报错由 JobDetail 传递过来的任务实例对象
   5. Trigger
      1. SimpleTrigger <普通的 Trigger> -- SimpleScheduleBuilder
      2. CronTrigger <带 Cron Like 表达式的 Trigger> --
         - CronScheduleBuilder
      3. CalendarIntervalTrigger <带日期触发的 Trigger> -
         - CalendarIntervalScheduleBuilder
      4. DailyTimeIntervalTrigger <按天触发的 Trigger> -
         - DailyTimeIntervalScheduleBuilder
   6. ThreadPool -- 为 Quartz 运行任务时提供了一些线程
      1. SimpleThreadPool --一个 Quartz 默认实现的简单线程池，它足够健壮，能够应对大部分常用场景
   7. -----以上是 Quartz 涉及到的一些关键对象，详细的内容如有机会会在后续的文章中展开!
8. Quartz 类图
   1.
   2. 类图中主要分为 5 块:Factory、Bulider、Scheduler、Trigger、JOB
9. 思想

// 1、工厂模式 构建 Scheduler 的 Factory，其中 STD 为 Quartz 默认的 Factory

```
//      开发者亦可自行实现自己的 Factory;Job、Trigger 等组件
SchedulerFactory sf = new StdSchedulerFactory();

// 2、通过 SchedulerFactory 构建 Scheduler 对象
Scheduler sched = sf.getScheduler();

// 3、org.quartz.DateBuilder.evenMinuteDate   -- 通过 DateBuilder 构建
Date
Date runTime = evenMinuteDate( new Date());

// 4、org.quartz.JobBuilder.newJob <下一分钟> --通过 JobBuilder 构建
Job
JobDetail job =
newJob(HelloJob.class).withIdentity("job1","group1").build();

// 5、通过 TriggerBuilder 进行构建 Trigger
Trigger trigger = newTrigger().withIdentity("trigger1","group1")
                .startAt(runTime).build();

// 6、工厂模式，组装各个组件<JOB，Trigger>
sched.scheduleJob (job, trigger);

// 7、start
sched.start();

try {
   Thread.sleep(65L * 1000L);
} catch (Exception e) {
}

// 8、通过 Scheduler 销毁内置的 Trigger 和 Job
sched.shutdown(true);
```

10. 一句话看懂 Quartz

    1. 1、创建调度工厂(); //工厂模式
       2、根据工厂取得调度器实例(); //工厂模式
       3、Builder 模式构建子组件<Job,Trigger> // builder 模式，如 JobBuilder、TriggerBuilder、DateBuilder
       4、通过调度器组装子组件 调度器.组装<子组件 1,子组件 2...> //工厂模式
       5、调度器.start(); //工厂模式

http://blog.csdn.net/chenweitang123/article/details/37837557
http://blog.csdn.net/chenweitang123/article/details/37777399
http://maphey.iteye.com/blog/2299485