GUIA PARA UTILIZAÇÃO DO FRAMEWORK QUARTZ

Outubro – 2010

Sumário de Informações do Documento

Tipo do Documento: Referência

Título do Documento: Guia para Utilização do Framework Quartz

Estado do Documento: Elaborado

Responsáveis: Lucas Rafael de Almeida, Willian R. Medeiros – Revisado por Cleverson Budel, Fabio Sgoda

Palavras-Chaves: Quartz

Resumo: Descrição de como configurar e utilizar o Quartz durante o desenvolvimento de aplicações Java.

Número de páginas: 14

Software utilizados:

Versão	Data	Mudanças
1.0	05/10/10	Criação do documento.

SUMÁRIO

1. QUARTZ

1.1 Introdução

A proposta deste documento é servir como um guia rápido e simples no auxílio ao desenvolvimento de *Jobs* de serviços, através da ferramenta Quartz, em aplicações Java Web.

1.2 Sobre o Quartz

O Quartz é um framework *Open Source* que propicia o agendamento de *Jobs* de serviços em aplicações Java, nas plataformas Java EE e Java SE; permitindo que se desenvolva aplicações com agendamentos por intervalos de tempo, onde é possível construir vários relacionamentos entre os elementos de sua estrutura básica: *Jobs*, *Trigger* e *Scheduler*.

1.2.1 Jobs

São as tarefas que serão executadas pelo Quartz. A execução está vinculada a uma determinada condição associada a este processo que deve ser satisfeita. É parte do código responsável por executar a tarefa propriamente dita do agendamento.

1.2.2 Trigger

Trigger, ou gatilho, está associado ao evento do disparo durante a execução de um processo. É parte do código responsável por determinar quando determinado *Job* deverá ser executado.

1.2.3 Scheduler

Também conhecido como *Agendador de Tarefas*, é responsável por realizar o

agendamento das tarefas ou *Jobs,* como também verificar quando determinado processo deverá ser executado pelo sistema.

2. CONFIGURAÇÃO

2.1 Configurando o Quartz na Aplicação

Configurar a aplicação para utilizar o Quartz não exige muito trabalho, sendo necessário apenas que o projeto contenha as bibliotecas *(JARs)* do Quartz, a adição de alguns atributos dentro do arquivo “web.xml”, e a criação de um arquivo de propriedades dentro do *classpath* da aplicação.

2.2 Bibliotecas do Quartz

Para a utilização do Quartz, é necessário que as bibliotecas (arquivos *JARs*) deste, façam parte do *classpath* do projeto. No caso da aplicação ser um projeto que utiliza o Plataforma Pinhão, e que tenha sido criada a partir do 'Projeto Mínimo', essa preocupação é desnecessária para o desenvolvedor, uma vez que tais arquivos já vêm adicionados no projeto; encontram-se no diretório:

```
context/WEB-NF/lib/
```

2.2.1 Download das bibliotecas

Caso o projeto não tenha sido criado a partir do 'Projeto Mínimo' da Plataforma Pinhão, as bibliotecas, que devem ser adicionadas no *classpath* do projeto, podem ser encontradas no site do Quartz, endereço:

```
www.quartz-scheduler.org/download
```

2.2.2 Utilização do Maven

Caso o projeto utilize o Maven para gerenciar as bibliotecas, o desenvolvedor,

ao invés de adicionar os *JARs* no *classpath* do projeto, precisará informar ao Maven que o projeto irá utilizar o Quartz. Para isto, basta adicionar no arquivo de configuração do Maven, localizado no projeto, no diretório raiz, denominado *"pom.xml"*, as dependências necessárias.

Para os projetos que utilizam o Framework Pinhão, não é necessário informar explicitamente as dependências para o Quartz; uma vez que este *framework* já o faz implicitamente. Abaixo segue exemplo de dependência informada dentro do "*pom.xml"*:

```
<dependencies>
   ...

   <dependency>
      <groupId>framework</groupId>
      <artifactId>framework</artifactId>
      <version>2.5.1</version>
   </dependency>

   ...
</dependencies>
```

Para os projetos que não fazem uso do Framework Pinhão, é necessário informar as dependências do Quartz dentro do *"pom.xml".*

2.3 Configurando o arquivo "*web.xml"*

Dentro do arquivo *"web.xml",* deve-se adicionar algumas *tags* para que o serviço do Quartz seja iniciado e passe a verificar os *Jobs* que se encontram agendados, aguardando por execução. Abaixo segue o código *XML* a ser acrescentado:

```
<servlet>
      <servlet-name>QuartzInitializer</servlet-name>

      <!-- Opcional. Pode-se adicionar quaisquer descrições -->
      <display-name>Quartz Initializer Servlet</display-name>

      <servlet-class>
          org.quartz.ee.servlet.QuartzInitializerServlet
      </servlet-class>

      <load-on-startup>1</load-on-startup>
</servlet>
```

2.4 Configurando o arquivo de propriedades "*quartz.properties*"

O arquivo de propriedades do Quartz chama-se "*quartz.properties*" e deve ficar no *classpath* da aplicação. Para as aplicações *Web* que seguem a estrutura da Plataforma Pinhão, o arquivo deve ficar no diretório:

```
src/main/resources >>> Para aplicações que utilizam a estrutura do Maven
```

```
context/WEB-INF/src >>> Para aplicações que não utilizam a estrutura do Maven
```

Dentro deste arquivo é possível definir como será o comportamento e a forma de execução dos serviços agendados, como, por exemplo: definir se os serviços agendados estarão armazenados em memória, em banco de dados, entre outros. Abaixo segue um simples exemplo deste arquivo:

```
org.quartz.scheduler.instanceName = NomeDaAplicacaoQuartzScheduler
org.quartz.scheduler.instanceId = AUTO
org.quartz.scheduler.rmi.export = false

org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount = 3
org.quartz.threadPool.threadPriority = 5

org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
```

Sobre as descrições utilizadas no exemplo acima, segue uma breve descrição dos parâmetros que foram informados:

Parâmetro	Descrição
`org.quartz.scheduler.instanceName`	Nome dado ao *scheduler*. Nesse caso, "NomeDaAplicacaoQuartzScheduler".
`org.quartz.scheduler.instanceId`	Valor que indica o *id* do *scheduler*. Esse valor deve ser único e pode ser do tipo texto. Pode-se utilizar o valor *"AUTO"* para que este valor seja gerado automaticamente.
`org.quartz.scheduler.rmi.export`	Indica se o *scheduler* poderá ser acessado via *RMI*. Quando false, indica que o scheduler será local. Caso não contenha este parâmetro, o valor *default* será *false.*
`org.quartz.threadPool.class`	Nome da implementação do *threadPool* que será utilizada.
`org.quartz.threadPool.threadCount`	Quantidade máxima de *threads* simultâneas disponíveis no *pool* para executar os *Jobs* deste *scheduler*.
`org.quartz.threadPool.threadPriority`	Indica a prioridade em que o *Job* de Serviço será executado. Pode variar entre prioridade mínima (1) a prioridade máxima(10). O valor *default* é 5.
`org.quartz.jobStore.class`	Indica como será armazenado os dados dos *Jobs* e das *triggers*. Utilizando *"org.quartz.simpl.RAMJobStore",* armazenará em memória.

As configurações no *"quartz.properties"* podem variar de acordo com as necessidades de cada aplicação; como, por exemplo: se determinada aplicação possuir poucos agendamentos durante o dia e com intervalos grandes, o valor ideal para ser definido, na propriedade referente ao *Pool* de *Threads* (***"org.quartz.threadPool.threadCount"***), poderia ser 1 (um).

Maiores informações, refente ao arquivo de configurações e seus parâmetros, podem ser encontradas no site: http://www.quartz-scheduler.org/.

2.4.1 Agendamento em Memória

Nos ambientes de ***Desenvolvimento*** e ***Homologação*** é recomendado que o agendamento seja armazenado em memória, onde a propriedade ***"org.quartz.jobStore.class"*** deverá conter o valor ***"org.quartz.simpl.RAMJobStore"***. Nesse caso, a configuração do arquivo de propriedades do Quartz ficará semelhante ao exemplo citado anteriormente; podendo, uma propriedade ou outra, variar de acordo com as necessidades de cada aplicação.

2.4.2 Agendamento em Base de Dados

O agendamento em Base de Dados consiste em armazenar o controle de execução de Jobs em um Banco de Dados; e seu uso é recomendado para as aplicações que se encontram no ambiente de ***Produção***. Além de informar na propriedade ***"org.quartz.jobStore.class"*** que o Quartz deverá armazenar o agendamento em banco de dados (***"org.quartz.impl.jdbcjobstore.JobStoreCMT"***), deve-se configurar outras propriedades para que funcione corretamente, como demonstrado abaixo:

```
org.quartz.scheduler.instanceName = NomeDaAplicacaoQuartzScheduler
org.quartz.scheduler.instanceId = AUTO
org.quartz.scheduler.rmi.export = false

org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount = 3
org.quartz.threadPool.threadPriority = 4

org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreCMT
```

```
org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.PostgreSQLDelegate
org.quartz.jobStore.tablePrefix = quartz.qrtz_
org.quartz.jobStore.dataSource = myDS
org.quartz.dataSource.myDS.jndiURL = java:jdbc/QuartzNomeDaAplicacaoDS
org.quartz.jobStore.nonManagedTXDataSource = myDS
org.quartz.jobStore.isClustered = true
```

Segue abaixo discriminação dos parâmetros utilizados acima:

Parâmetro	Descrição
`org.quartz.scheduler.instanceName`	Nome dado ao scheduler. Nesse caso, "NomeDaAplicacaoQuartzScheduler".
`org.quartz.scheduler.instanceId`	Valor que indica o *id* do *scheduler*. Esse valor poder ser texto e dever ser único. Pode-se utilizar o valor "AUTO" para que este valor seja gerado automaticamente. Caso a aplicação esteja em *"cluster"*, e guardando agendamento em tabelas, deve-se utilizar "AUTO".
`org.quartz.scheduler.rmi.export`	Indica se o *scheduler* poderá ser acessado via *RMI*. Quando *false*, indica que o scheduler será local. Caso não contenha este parâmetro, o valor *default* será *false.*
`org.quartz.threadPool.class`	Nome da implementação do *threadPool* que será utilizada.
`org.quartz.threadPool.threadCount`	Quantidade máxima de *threads* simultâneas disponíveis no pool para executar os *Jobs* deste *scheduler*.
`org.quartz.threadPool.threadPriority`	Indica a prioridade em que o *Job* de Serviço será executado. Pode variar entre prioridade mínima (1) a prioridade máxima(10). O valor *default* é 5.
`org.quartz.jobStore.class`	Indica como será armazenado os dados dos *Jobs* e das *triggers*. Nesse caso foi utilizado *"org.quartz.impl.jdbcjobstore.JobStoreCMT"* (Container -Managed Transactions) para indicar que os dados serão armazenados em Banco de Dados.
`org.quartz.jobStore.driverDelegateClass`	Informa qual o "dialeto" do Banco de Dados que será utilizado para armazenar os agendamentos. Nesse caso foi utilizado *"org.quartz.impl.jdbcjobstore.PostgreSQLDelegate"* para o SGBD PostgreSQL. Cada SGBD possui um "dialeto" distinto.
`org.quartz.jobStore.tablePrefix`	Valor que indica o prefixo das tabelas do Quartz criadas no Banco de Dados.
`Org.quartz.jobStore.dataSource`	Nome do *DataSource* definido no arquivo de propriedades (NOMEDS).
`org.quartz.dataSource.NOMEDS.jndiURL`	*JNDI URL* do *DataSource* que será gerenciado pelo servidor de aplicação.
`org.quartz.jobStore.nonManagedTXDataSource`	Nome do *DataSource* que conterá conexão sem o uso do Container-Managed Transactions.
`org.quartz.jobStore.isClustered`	O valor desta propriedade deve ser "true", se possuir várias instâncias do Quartz utilizando as mesmas tabelas no banco de dados. E caso seja "true", deve ser "AUTO" o valor da propriedade *"org.quartz.scheduler.instanceId".*

Para obter maiores informações sobre a parametrização do arquivo de configurações do Quartz é possível encontrá-las no site: http://www.quartz-

scheduler.org/ .

3. IMPLEMENTANDO O AGENDAMENTO COM O QUARTZ

3.1 Criando o Agendador do Serviço

A codificação do agendador de serviço consiste em implementar numa classe o método que realizará o agendamento de uma determinada tarefa, ficando assim, responsável por registrar quando determinada tarefa deverá ser executada. Sua construção é simples e pode ser verificada no exemplo abaixo:

```
...

/**
 *  Nome do serviço que será armazenado pelo “Scheduler”
 */
public static final String NOME_JOB = "NomeDoJob";

...

/**
 *  Método que realiza o agendamento
 */
public static void agendar(){
   try {
      // Obtendo uma instância de agendamento.
      Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

      // Verificando se a tarefa encontra-se agendada,
      // para não registrá-la mais de uma vez.
      //
      // Obs.: Neste exemplo utilizou-se o grupo “Scheduler.DEFAULT_GROUP“.
      //       Mas pode-se utilizar quaisquer outros. Este campo é aberto,
      //       do tipo “String”, e serve para associação dos “jobs”.
      if (scheduler.getJobDetail(NOME_JOB, Scheduler.DEFAULT_GROUP) == null) {
         //
         // Criando um novo job
         JobDetail jobDetail = new JobDetail(NOME_JOB, Scheduler.DEFAULT_GROUP,
                               ExecutorAgendamentoQuartz.class);

         // Criando “Trigger” (usando Cron) associado ao Job e Scheduler
         CronTrigger trigger = new CronTrigger(NOME_JOB, Scheduler.DEFAULT_GROUP);

         // Parametros Cron(? ? ? ? * *): Segundos, Minutos, Hora, Dia do Mês,
         //                                Mês, Dia_da_Semana e Ano (opcional)
         trigger.setCronExpression("0 0 5 ? * MON"); // Agendado pra todas as seg. - 5h

         // Adicionando “job” ao agendador de serviços.
         scheduler.scheduleJob(jobDetail, trigger);
      }
   }
   catch (Exception e) {
      log.error("Ocorreu um erro ao agendar ["+ NOME_JOB + "] no Quartz.", e);
   }
}

...
```

No exemplo acima é demonstrado como obter uma instância do agendador,

através de "***StdSchedulerFactory.getDefaultScheduler()***", e a criação de *Job*. Durante a criação de um *Job,* é informado: o nome desse *Job*, o nome de um grupo (tarefas são organizadas em grupos) e a classe que será responsável por executar a tarefa; no exemplo em questão é a classe "***ExecutorAgendamentoQuartz***", descrita posteriormente. Como nesse exemplo não foi necessário organizar várias tarefas em diferentes grupos, foi utilizada a constante "***Scheduler.DEFAULT_GROUP***", já oferecida pelo Quartz. O conteúdo desta constante é a *String*: `"DEFAULT".`

Outro detalhe que é mostrado no exemplo é o tipo de *Trigger* (disparo) escolhido para o exemplo, neste caso o "***CronTrigger***"; este é baseado no *Cron* e é recomendado para a maioria dos casos por não apresentar problemas quanto ao horário de verão. Mas é possível escolher outros tipos de Triggers disponibilizados pelo Quartz; como por exemplo, o "***SimpleTrigger***".

Para obter maiores informações sobre o uso do agendamento e dos tipos de *Triggers*, é possível através da documentação do Quartz, disponível no site: http://www.quartz-scheduler.org/.

3.1.1 Passagem de Parâmetros para o Executor

Em certos sistemas, pode haver a necessidade de se passar algum parâmetro, durante o momento do agendamento, e que será utilizado somente no momento da execução da tarefa agendada. Nestes casos é possível acrescentar parâmetros através da instância do "***JobDetail***", no "***JobDataMap***", como exemplificado abaixo:

```
...

// Criando um novo job
JobDetail jobDetail = new JobDetail(NOME_JOB, Scheduler.DEFAULT_GROUP,
                                    ExecutorAgendamentoQuartz.class);

// Adicionando parâmetro no "job"
jobDetail.getJobDataMap().put("NOME_PARAMETRO", "VALOR_PARAMETRO");

...
```

Já no momento de execução da tarefa, para se obter o valor informado durante o agendamento, é preciso somente informar o nome do parâmetro, no

“*JobDataMap*”, como demonstrado abaixo:

```
   ...

   // Obtendo parâmetro informado no “Job”.
   JobDataMap dataMap = context.getJobDetail().getJobDataMap();
   String parametroInformado = (String) dataMap.get(“NOME_PARAMETRO”);

   ...
```

3.1.2 Agendando manualmente

O Agendamento manual nada mais é do que chamar de algum ponto do sistema o método da classe que realiza o agendamento das tarefas no Quartz. A execução do agendamento pode ser motivada por qualquer ação que resulte em agendar uma tarefa para ser executada *“a posteriori”*. A execução de um determinado caso de uso ou uma ação em particular tomada pelo usuário podem motivar a execução do agendamento.

3.1.3 Agendando automaticamente através de um *Listener*

Em determinadas aplicações, o agendamento deve ocorrer desde o momento em que ela já se encontra em execução. Para esta situação, é possível resolver criando um *Listener* e adicionando o código responsável pelo agendamento dentro dele. Abaixo tem-se uma classe exemplificando a criação de um *Listener* e a chamada para o agendamento:

```
...

/**
 *  Classe Servlet Listener
 */
public class AgendadorServicoQuartz implements ServletContextListener {

   ...

   /**
    * Método executado quando se inicia a aplicação.
    */
   public void contextInitialized(ServletContextEvent event) {
      // Chama o executor do agendamento.
      agendar();
```

```
    }

   ...

    /**
     * Método executado quando a aplicação é finalizada.
     */
    public void contextDestroyed(ServletContextEvent event) {
       try {
          Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

          // Remove o agendamento.
          scheduler.removeJobListener(NOME_JOB);
       }
       catch (Exception e) {
          log.error("Ocorreu um erro ao remover agendamento do Quartz.", e);
       }
    }

    ...
}
```

Para que este *Listener* funcione, é necessário adicionar, no arquivo *web.xml,* o nome da classe que o implementa, como visto abaixo:

```
...

<listener>
   <listener-class>gov.pr.celepar.exemplo_quartz.AgendadorServicoQuartz</listener-class>
</listener>

...
```

3.2 Criando o Executor do Serviço

Ao criar a classe que ficará responsável pela execução do Agendamento, será necessário que esta classe implemente a interface "***StatefulJob***", disponibilizado pelo Quartz. Esta interface obrigará a implementação do método *execute()*, responsável por executar a tarefa através do *Scheduler*. Abaixo segue um pequeno exemplo de uma classe que implementa *StatefulJob*:

```
...

/**
 * Classe que executa Job do Quartz
 */
public class ExecutorAgendamentoQuartz implements StatefulJob{

   ...

   /**
    * Executar Job agendado.
    */
   public void execute(JobExecutionContext context) throws JobExecutionException {

      ...

      // Executar as operações previstas para o agendamento.
      Facade.executaAlgumaOperacao();

      ...
   }

   ...

}
```