FAM - Timers

Taula de continguts

- 1. Prerequisits
- 2. Introducció
- 3. Pasos
  - 3.1. Configurar servlet del timer
  - 3.2. Construir servlet del timer
  - 3.3. Construir tasques executables
- 4. Altres consideracions
  - 4.1. Desplegament en clúster
  - 4.2. Aplicacions amb proxy
  - 4.3. Parametritzar temps d'execució
  - 4.4. Bloquejar registres

1. Prerequisits

- Tenir una aplicació tipus FAM
- Tenir definit l'accés a base de dades sense proxy user. Els timers funcionen en background sense usuari connectat, per tant no es pot fer servir una base de dades que funcioni amb proxy user. En intentar combinar dos models de persistència hi havien creuaments, per tant, s'ha d'adaptar el DS per funcionar sense proxy user.

2. Introducció

Els timers son uns components Java que permeten executar tasques en background.

Disposem de diferents exemples a aplicacions webs fetes

- **re-EACAT:** Consulta cada x temps documents pendents de descarregar des de EACAT
- **PMT:** Reintents de registre
- **TeDiba:** Registre de decrets en background
- **TeDiba:** Enviament periòdic de emails
- **eNoti:** Procesa en background els diferents pasos de les comunicacions i notificacions

3. Pasos

3.1. Configurar servlet del timer

Al web.xml s'ha d'afegir la següent configuració

```
<!-- DIBA: Servlet que es fa servir per programar tasques  -->
  <servlet>
       <description>Servlet que es fa servir per programar tasques</description>
       <display-name>Timer Servlet</display-name>
       <servlet-name>Timer Servlet</servlet-name>
       <servlet-class>cat.diba.jee.fam.base.servlet.TimerServlet</servlet-class>
       <load-on-startup>9</load-on-startup>
  </servlet>

  <!-- DIBA: Necessari pel servlet  -->
  <resource-ref>
        <res-ref-name>tm/TimerManager</res-ref-name>
        <res-type>commonj.timers.TimerManager</res-type>
        <res-auth>Container</res-auth>
        <res-sharing-scope>Unshareable</res-sharing-scope>
  </resource-ref>

  <servlet-mapping>
      <servlet-name>Timer Servlet</servlet-name>
      <url-pattern>/timerServlet</url-pattern>
  </servlet-mapping>
```

Nota: *On diu cat.diba.jee.fam.base.servlet.TimerServlet indicar el package del projecte.*

3.2. Construir servlet del timer

S'ha de fer el servlet que gestiona tots els timers. Hi ha un exemple al projecte eNoti.

Aquesta classe ha d'incloure:

- Constant JNDI que lliga amb lo definit al web.xml

```
private static final String JNDI_TIMER_MANAGER = "java:comp/env/tm/TimerManager";
```

- Inicialització del servlet, iniciant variables i timers

```
 public void init() throws ServletException {
        super.init();
        try {
            ic = new InitialContext();
            tm = (TimerManager) ic.lookup(JNDI_TIMER_MANAGER);
            tm.scheduleAtFixedRate(new TimerController(new NotificacioEstatsEnotiTask
Timer()), 3 * DEFAULT_PERIOD, DEFAULT_PERIOD);
            tm.scheduleAtFixedRate(new TimerController(new NotificacioEstatsEnotumTas
kTimer()), 3 * DEFAULT_PERIOD, DEFAULT_PERIOD);
            tm.scheduleAtFixedRate(new TimerController(new ComunicacioEstatsEnotiTask
Timer()), 3 * DEFAULT_PERIOD, DEFAULT_PERIOD);
        } catch (Throwable t) {
```

```
            LOG.error("Error en la càrrega de registres ", t);
        }
    }
```

- Classe inner que permet refrescar el periode del timer

```
 private class TimerController implements TimerListener {

        private TaskTimable taskTimable;

        private long initPeriod;

        private long period;

        private ConfigService servei = new ConfigService();

        public TimerController(TaskTimable taskTimable) {
            super();
            this.taskTimable = taskTimable;
        }

       /* (non-Javadoc)
         * @see commonj.timers.TimerListener#timerExpired(commonj.timers.Timer)
         */
        @Override
        public void timerExpired(Timer timer) {
            try {
                boolean ambProxy = AppPropertiesFactory.appProperties().getBooleanPro
perty("entorn.ambProxy");
                if (ambProxy) {
                    ProxySwitch.activateProxy();
                }
                taskTimable.process();
                refreshScheduleTimer(timer);
                // Mail diari
            } catch (Exception e) {
                //Error al saber si necesitem el Proxy
                LOG.debug("Error : " + e.getMessage());
            }
        }

       /**
         * Schedule timer.
         *
         * @param timer the timer
         */
        private void refreshScheduleTimer(Timer timer) {
            try {
                if (isPeriodeTimerModificat()) {
                    timer.cancel();
                    tm.scheduleAtFixedRate(new TimerController(taskTimable), initPeri
od, period);
                    // tm.schedule(new TimerController(), initPeriod, period);
```

```
                }
            } catch (Exception e) {
                //Si hi ha un error no el timer continua igual
                LOG.debug("Error : " + e.getMessage());
            }
        }

       /**
         * Checks if is periode timer modificat.
         *
         * @return true, if is periode timer modificat
         * @throws Exception the exception
         */
        private boolean isPeriodeTimerModificat() throws Exception {
            Long configPerIni = Long.valueOf(servei.obtenirConfigPerClau(taskTimable.
getIniciConstant())
                    .getCfgValor());
            Long configPer = Long.valueOf(servei.obtenirConfigPerClau(taskTimable.get
PeriodConstant())
                    .getCfgValor());

           if (initPeriod != configPerIni || period != configPer) {
                initPeriod = configPerIni;
                period = configPer;
                return true;
            }
            return false;
        }
    }
```

3.3. Construir tasques executables

Aquestes tasques per anar lligades amb la classe inner comentada anteriorment han de complir la interficie TaskTimable. Aquesta interficie defineix els métodes:

- **proccess()** Que es cridarà cada cop que acabi el periode del timer
- **getIniciConstant()** Que defineix el paràmetres inicials de temps d'espera per la primera execució
- **getPeriodConstant()** Que defineix els paràmetre inicials de periode de temps d'execució del timer

4. Altres consideracions

4.1. Desplegament en clúster

En desplegar el timer, hem de considerar que hi haurà una instància per cada màquina virtual. Per tant en un cluster de 4 hi hauran 4 timers que si, per exemple, envien mails cada hora, aquest mail s'enviarà 4 cops.

Per resoldre això tenim dues opcions:

- Desplegar l'aplicació en un únic node (PMT)

- Controlar per bd la execució única (eNoti, TeDiba)

4.2. Aplicacions amb proxy

Com el timer s'executarà en un Thread diferent, si l'aplicació té configurat per codi l'ús d'un proxy, també s'haurà d'activar al timer.

4.3. Parametritzar temps d'execució

Es recomanable treballar amb una taula que defineixi els diferents timers i els temps de execució. D'aquesta manera es pot augmentar o reduir la freqüència d'execució dels mateixos, així com aturar-los. El eNoti (i per tant el codi d'exemple que s'ha posat aquí treballa d'aquesta manera)

4.4. Bloquejar registres

Si es treballa amb timers que processen registres, s'haurien de bloquejar (amb un camp en_transacció) per exemple, per tal que les diferents instàncies del timer no afectin als mateixos registres. Això està fet al TeDiba i al eNoti i el procés és el següent:

1. El timer cerca registres per processar que no estiguin en transacció
2. Abans de continuar, intenta bloquejar-los tots marcant-los en transacció
3. Si el update retorna el mateix número de registres, continua, sinó, s'atura.
4. Un cop recuperats els registres a processar els processa.

Categories: Framework v3
Categories: Plataforma JEE
Etiquetes: FAM
Etiquetes: Timer

URL d'origen: https://comunitatdstsc.diba.cat/wiki/fam-timers