Оглавление

Служба таймера (Timer Service)..... 1
Служба таймера EJB (EJB Timer Service)..... 2
Выражения планирования EJB..... 3
Пример системы пакетной обработки кредитных карт..... 4
Бизнес-интерфейс..... 4
Реализация класса EJB..... 5
Декларативные таймеры..... 6
Аннотация @javax.ejb.Schedule..... 6
Аннотация @javax.ejb.Schedules..... 7
Программные таймеры..... 8
Интерфейс javax.ejb.TimerService..... 8
Класс javax.ejb.ScheduleExpression..... 9
Класс javax.ejb.TimerConfig..... 10
Интерфейс Timer..... 10
Интерфейс Timer..... 10
Остановка таймера..... 11
Идентификация таймера и его описание..... 11
Получение дополнительной информации о таймере..... 12
Объект TimerHandle..... 12
Исключения Timer..... 13
Транзакции, безопасность и Timer..... 13
Таймеры SLSB..... 13
Таймеры MDB..... 15
Ругательства..... 16
Литература..... 16

Служба таймера (Timer Service)

Бизнес-системы часто используют планировщики задач. Планировщики обычно запускают приложения, которые генерируют отчеты, переформатируют данные, проводят учетную работу, обновляют кэш по ночам и т.д. Другой класс задач, выполняемый планировщиками — оповещение о наступлении временных событий, таких как предопределенные даты, предельные сроки. Часто планировщики запускают пакетные задачи. В UNIX самый известный из планировщиков — cron работающий с текстовыми файлами crontab определенного формата.

Планировщики можно применять безотносительно ПО в следующих примерах:

- в системах обслуживания денежных переводов (card processing system), где платежи по кредитным картам объединяются в пакеты, так, что бы все платежи, выполненные в одну единицу времени, начинались вместе. Такая работа может проводиться по вечерам, чтобы уменьшить нагрузку на систему.
- в сборе сообщений, их конвертации в требуемый формат и отправки адресатам в медицинских системах.
- в регулярном создании автоотчетов и рассылки их менеджерам.

- в производственных приложениях (workflow application), системах, которые обрабатывают документы днями и месяцами, включают множество подсистем и допускают многочисленные вмешательства человека.
- в производственных процессах, где планировщики задействованы в учете и периодически производят опись состояния приложения, накладных, заявок на товар и т.д. с тем, чтобы гарантировать, что все идет по плану.

Планировщики поддерживают таймеры и выбрасывают сообщения, оповещающие приложение о наступлении контрольной даты или истечении периода.

Бизнес-примеры таймеров:

- Варка яиц. Одноразовый таймер выставляется в зависимости от типа результата: «вкрутую», «в мешочек», «всмятку».
- Ипотечная система. Перед закрытием ипотеки должно быть выполнено множество задач (оценка стоимости, фиксация ставки, распределение наследства и т.д.). Таймеры могут использоваться в ипотечной системе для гарантии работы по плану.
- В системе медицинских обращений (claim). В соответствии с общепринятыми в среде врачи-клиники понятиями, на каждую заявку отводится до 90 дней. Для каждой заявки может быть установлен таймер, срабатывающий за 7 дней до истечения срока ее рассмотрения.
- Система биржевого брокера. Заявки на покупку по остатку могут создаваться для определенного числа акций, но только по фиксированной или ниже цене. Эти заявки обычно ограничены по времени. Если цена на определенные акции падает ниже фиксированной перед истечением заявки, то поддерживается покупка по остатку. В противном случае время истекает и заявка завершается.

Служба таймера EJB (EJB Timer Service)

В EJB 2.1 появился стандарт системы планирования Java EE, названный службой таймеров EJB (EJB TS). В EJB 3.1 спецификация была переделана, в нее был привнесен новый естественный язык описания, более дружественный по сравнению с прошлой версией и похожий на формат cron/crontab.

Rem: В JSE присутствует класс java.util.Timer, который позволяет потоку планировать выполнение задач в иных фоновых потоках. Это полезно в различных случаях, но слишком ограничено для использования в Java EE. Тем не менее смысл (семантика) планирования java.util.Timer похожа на таковую в EJB Timer Service.

EJB Timer Service дает EJB-контейнеру событийно-временной API, позволяющий планировать таймеры для определенных дат, периодов или временных интервалов.

Таймеры можно устанавливать декларативно (в deploy-time) при помощи аннотаций и дескриптора развертки (вендорного формата). Или программно (в run-time) при помощи

API EJB TS.

Таймер связывается с EJB-бином, который его установил, и при срабатывании вызывает в этом бине определенный временной callback-метод, называемый также таймаут-методом.

Rem: в отличие от обычной потактовой (по логическому времени) работе приложения, EJB TS работает с реальным временем. Поскольку никаких жестких ограничений по срокам реагирования на временное событие нет, EJB TS можно отнести к системе мягкого реального времени.

EJB TS (JEE5 - JEE7) может работать только с SLSB, MDB, Singleton. С SFSB она не работает, поскольку это клиенто-зависимый компонент.

Выражения планирования EJB

В EJB 3.1 (JEE6) был введен синтаксис планирования, подобный синтаксису планировщика UNIX cron/crontab. Календарное событие можно планировать, используя значения года, месяца, дня месяца, дня недели, часа, минуты и секунды, * (символ джокера), наибольшее значение Last, перечисление (0, 2, 6), диапазон (0-10), приращение 30/10.

Поле	**Допустимое значение**
second	[0-59] * , - /
minute	[0-59] * , - /
hour	[0-23] * , - /
dayOfMonth	[1,31] * , - / Last [-x] где x – число дней до конца месяца Last
month	[1,12] * , - / {"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"}
dayOfWeek	[0,7] * , - / Last {"Sun","Mon","Tue","Wen","Thu","Fri","Sat"}, 0 и 7 равны "Sun"
year	[xxxx] * , - / где xxxx – четырехзначный год

Выражение дает множество дат и формируется 7 перечисленными полями. В каждом поле задается множество допустимых значений в пределах своего разряда. Результирующее множество дат получается прямым произведением этих полей, за исключением того, что поля dayOfMonth и dayOfWeek объединяются в одно множество. Символ * означает множество всех допустимых значений поля. Приращение x/y задает

множество значений вида x, x+y, x+2y, … в области допустимых значений поля. Приращение */y эквивалентно 0/y. Приращение x-y/z эквивалентно x/z но с сужением области допустимых значений до [x-y]. Last означает последний день недели или последний день месяца, либо, если следует перед некоторым значением x дня недели, означает последнее x месяца (Last Thu – последний четверг месяца).

```
Ex: Создание таймера, срабатывающего в начале и через пол минуты каждого
рабочего часа (10-18) в будни (Mon-Fri):

@Schedule(second=”0,30”, minute=”0”, hour=”10-18”, dayOfMonth=”*”,month=”*”,
dayOfWeek=”Mon-Fri”,year=”*”)
```

```
Ex: Создание таймера, срабатывающего ежедневно в 0:0:30, 0:0:40, 0:0:50, 0:20:30,
0:20:40, 0:20:50, 12:0:30, 12:0:40, 12:0:50, 12:20:30, 12:20:40, 12:20:50

@Schedule(second=”30/10”, minute=”*/20”, hour=”*/12”)
```

Rem. Родной формат cron/crontab имеет такой вид (минуты, часы, день месяца, месяц, день недели, команда):

*20 * * * * root run-parts /etc/cron.hourly*
*25 1 * * * root run-parts /etc/cron.daily*
*50 3 * * 0 root run-parts /etc/cron.weekly*
*43 4 16 * * root run-parts /etc/cron.monthly*

Пример системы пакетной обработки кредитных карт

Рассмотрим розничный online-магазин с 20 регистрациями. Предположим, что каждую минуту происходят транзакции, которые в конечном итоге должны обрабатываться внешней платежной системой. Также предположим, что эта платежная система может потенциально обслуживать миллионы клиентов, а магазин лишь один из них, что может приводить к соответствующим задержкам трафика. Для улучшения обслуживания можно кэшировать запросы в пакеты, находящиеся в режиме ожидания до отправки единым запросом, если это позволяют правила бизнес-логики. При помощи EJB TS можно планировать такую работу и запускать ее внутри ejb.

Бизнес-интерфейс

Для реализации подсистемы обслуживания платежей кредитных карт необходимо реализовать несколько вещей. Во-первых уметь принимать новые транзакции и помещать их в очередь ожидания. Во-вторых необходимо обеспечить клиентов механизмом планирования обработки пакетных заданий. Для удобства также надо предоставить возможность получения текущей ожидающей транзакции и способ ее немедленного запуска. Тогда при возникновении запланированного для обработки события будет возможным просто вызвать бизнес-метод этой обработки.

```
public interface CreditCardTransactionProcessingLocal
```

```
{
// Возврат очереди ожидающих транзакций
  List<CreditCardTransaction> getPendingTransactions();
// Обработка ожидающих транзакций
  void process();
// Добавка транзакции в очередь ожидания
  void add(CreditCardTransaction transaction) throws IllegalArgumentException;
// Дата запуска очередной пакетной задачи
  Date scheduleProcessing(ScheduleExpression expression) throws
IllegalArgumentException;
}
```

Реализация класса EJB

В качестве примера будет использоваться @Singleton – бин. Ниже фрагменты кода, связанные с созданием таймеров и обработкой временных событий.

```
@Singleton
@Local(CreditCardTransactionalProcessingLocal.class)
@ConcurencyManagement(ConcurencyManagementType.CONTAINER)
public class public class CreditCardTransactionProcessingBean implements
  CreditCardTransactionProcessingLocalBusiness {
// Для получения ссылки на EJB TS через контекст
  @Resource
  private SessionContext context;
// Прямое внедрение в поле  EJB TS
  @Resource
  private TimerService timerService;
...
// Вспомогательный метод получения даты след. события
  @Override
  public Date scheduleProcessing(final ScheduleExpression expression) throws
    IllegalArgumentException {
      ...
      final TimerService timerService = context.getTimerService();
      final Timer timer = timerService.createCalendarTimer(expression);
      final Date next = timer.getNextTimeout();
      ...
      return next;
  }

// Callback-метод, запускающий обработку очереди ожидающих транзакций
  @Timeout
  @Schedule(dayOfMonth = EVERY, month = EVERY, year = EVERY, second = ZERO,
      minute = ZERO, hour = EVERY)
  public void processViaTimeout(final Timer timer) {
       this.process();
  }
```

В примере показано получение ссылки на EJB TS через инъекцию и контекст. При

помощи @Schedule декларативно создается неявный экземпляр таймера Timer, запускаемый контейнером ежечасно, а метод process viaTimeout(...) становится callback-обработчиком (таймаут-методом) этого таймера. Метод scheduleProcessing(…) также создает явные экземпляры таймеров, программным способом, с использованием параметра expression - выражения планирования. При помощи @Timeout метод viaTimeout(…) также становится таймаут-методом программных таймеров.

Rem: при использовании @Schedule аннотация @Timeout не нужна

Декларативные таймеры

Декларативные таймеры объявляются при помощи метаинформации и неявно создаются в deploy-time EJB-контейнером. Они хорошо подходят для запуска постоянных задач, таких как ночное удаление временных файлов или обновление кэша, генерация годовых отчетов, рассылка поздравлений с днем рождения и т.п. В спецификации EJB 3.1 (JEE6) для создания декларативных таймеров введена аннотация @javax.ejb.Schedule. Синтаксис объявления таймера в XML дескрипторе развертки не определен и оставлен на усмотрение вендора EJB-контейнера.

Аннотация @javax.ejb.Schedule

Служит для декларативного создания таймера и определения его таймаут-метода на этапе развертки, используя выражения планирования.

@Schedule применяется к методу, который будет играть роль таймаута для таймера, автоматически и неявно создаваемого EJB-контейнером при помощи атрибутов этой аннотации. Таймаут-метод должен иметь сигнатуру одного из указанных видов:

```
void <METHOD>()
void <METHOD>(Timer timer)
```

В ответ на срабатывание таймера, EJB-контейнер будет извлекать из пула экземпляр EJB и вызвать в нем этот таймаут-метод.

Аннотация @Schedule имеет следующий вид:

```
@Target(value=METHOD)
@Retention(value=RUNTIME)
public @interface Schedule {
  String dayOfMonth() default "*";
  String dayOfWeek() default "*";
  String hour() default "0";
  String info() default "";
  String minute() default "0";
  String month() default "*";
  boolean persistent() default true;
```

```
  String second() default "0";
  String timezone() default "";
  String year() default "*";
}
```

В строковых атрибутах можно задавать выражения, используя синтаксис выражений планирования. По-умолчанию такой декларативный таймер будет запускаться ежедневно в полночь. В атрибуте info можно задать информацию о таймере в виде String для дальнейшего использования в таймаут-методе. Флаг persistent указывает, будет ли таймер сериализовываться для сохранения при сбое JEE-сервера (по-умолчанию будет).

```
Ex.: ежемесячная рассылка по первым числам месяца в полночь

@Schedule(second=”0”, minute=”0”, hour=”0”, dayOfMonth=”1”, month=”*”, year=”*”)
public void sendMonthly() { … }
```

```
Ex.:  каждый будний день в 6:55 по парижскому времени

@Schedule(second=”0”, minute=”55”, hour=”6”, dayOfWeek=”Mon-Fri”,
timezone=”Europe/Paris”)
public void sendMonthly() { … }
```

```
Ex.:  каждый 14 минут с часа до двух ночи

@Schedule(minute=”*/14”, hour=”1,2”)
public void sendMonthly() { … }
```

```
Ex.:  каждый второй час в течении дня, начиная с полудня каждый второй вторник
каждого месяца

@Schedule(hour=”12/2”, dayOfMonth=”2nd Tue”)
public void sendMonthly() { … }
```

```
Ex.:  за 3 дня до конца каждого месяца в 13:00

@Schedule(hour=”13”, dayOfMonth=”-3”)
public void sendMonthly() { … }
```

Аннотация @javax.ejb.Schedules

К любому EJB может быть прикреплено множество таймеров. При их одновременном срабатывании будут задействованы различные экземпляры EJB. Множественное прикрепление декларативных таймеров обеспечивается @Schedules

```
@Target(value=METHOD)
@Retention(value=RUNTIME)
```

```
public @interface Schedules {
  javax.ejb.Schedule[] value();
}
```

```
Ex.: использование нескольких таймеров

@Schedules({
// Ежегодно, в полдень последнего четверга ноября
@Schedule(second=”0”, minute=”0”, hour=”12”, dayOfMonth=”Last Thu”,
month=”Nov”, year=”*”),
// Ежегодно в полдень 18 ноября
@Schedule(second=”0”, minute=”0”, hour=”12”, dayOfMonth=”18”,
month=”Dec”, year=”*”)
})
// Таймаут в собственной изолированной транзакции
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
public void sendHoliday() { … }
```

Программные таймеры

Программный способ создания таймеров позволяет планировать заранее неизвестные временные события и реакцию на них.

Интерфейс javax.ejb.TimerService дает ejb явный доступ к службе таймера EJB-контейнера. Это позволяет программно создавать новые объекты Timer и получать доступ к уже имеющимся. Получить ссылку на объект TimerService можно при помощи метода getTimerService() интерфейса EJBContext (у его наследников SessionContext, MessageDrivenContext). Также можно использовать прямое внедрение TimerService в EJB при помощи @Resource.

Для подключения EJB к EJB TS с описанием реакции на временные события, в классе EJB надо определить таймаут-метод. Класс EJB должен либо реализовать интерфейс javax.ejb.TimedObject, который содержит единственный метод void ejbTimeout(Timer timer), либо в нем надо аннотировать аннотацией @javax.ejb.Timeout один и только один из методов с сигнатурой void <METHOD>(xxx), где xxx – void либо Timer. Этот таймаут-метод будет вызываться при наступлении временных событий в одном из экземпляров EJB, активированных для этого EJB-контейнером.

Интерфейс javax.ejb.TimerService

Интерфейс TimerService имеет следующий вид:

```
• public Timer createTimer(long duration, java.io.Serializable info)
• public Timer createTimer(java.util.Date expiration, java.io.Serializable info)
• public Timer createSingleActionTimer(long duration, TimerConfig timerConfig)
```

- public Timer createSingleActionTimer(java.util.Date expiration, TimerConfig timerConfig)
- public Timer createTimer(long initialDuration, long intervalDuration, java.io.Serializable info)
- public Timer createTimer(java.util.Date initialExpiration, long intervalDuration, java.io.Serializable info)
- public Timer createIntervalTimer(long initialDuration, long intervalDuration, TimerConfig timerConfig)
- public Timer createIntervalTimer(java.util.Date initialExpiration, long intervalDuration, TimerConfig timerConfig)
- public Timer createCalendarTimer(ScheduleExpression schedule)
- public Timer createCalendarTimer(ScheduleExpression schedule, TimerConfig timerConfig)
- public Collection<Timer> getTimers()
- public Collection<Timer> getAllTimers()

Таймеры, созданные createTimer(…) и createIntervalTimer(…) относятся к интервальным таймерам, появившимся в EJB 2.1. Метод getAllTimers() появился в EJB 3.2 (JEE7) Остальные методы появились в EJB 3.1 (JEE6). Методы createCalendarTimer(…) создают календарные таймеры на основе выражений планирования.

Когда таймер срабатывает, выполняется таймаут-метод EJB: либо ejbTimeout(...), либо @Timeout метод.

После создания таймера, EJB-контейнер делает его персистентным, сохраняя в некоторое временное хранилище на случай сбоя системы. И если даже сервер упадет, то после перезапуска таймеры по-прежнему будут активны. В спецификации явно не указано, как должен реагировать истекший во время сбоя таймер, но общепринято, что он должен сработать сразу после перезапуска системы, а интервальный должен сработать столько раз, сколько интервалов истекло при сбое. Конкретику нужно узнавать в вендорной документации.

Метод getTimers() выдает список всех декларативных и программных таймеров конкретного бина в виде неупорядоченной коллекции (List) из 0 или более объектов Timer, каждый из которых представляет собой запланированное событие, которое было спланировано для бина с использованием TimerService. Данный метод полезен для просмотра всех таймеров, отмены ставших ненужными или их перепланировании. Метод getAllTimers() введен в EJB 3.2 (JEE7), он выдает список всех таймеров модуля, в котором установлен делающий вызов EJB .

Rem: список таймеров, полученных getTimers() или getAllTimers(), может содержать просроченные таймеры, обращение к любому методу которых вызовет выброс исключения.

Класс javax.ejb.ScheduleExpression

javax.ejb.ScheduleExpression – класс, появившийся в EJB 3.1 (JEE6), который при помощи удобного синтаксиса позволяет планировать программные таймеры. Задавать

выражение планирования можно при помощи свойств ScheduleExpression, имеющих строковое и нативное представления. Методы, задающие такие свойства возвращают ссылку на this, что позволяет использовать цепочки настройки.

```
Ex.: задание выражения планирования — по лиссабонскому времени с 10 до 14 и в 20 часов, каждые 5 минут.

new ScheduleExpression().minute("*/5").hour("10-14,20").timeZone("Europe/Lisbon");
```

Класс ScheduleExpression используется для программного создания таймеров, т.е. объектов javax.ejb.Timer при помощи javax.ejb.TimerService. Для декларативного создания таймеров используется @javax.ejb.Schedule с тем же синтаксисом.

По-умолчанию используется часовой пояс из настроек JEE-сервера. Для его изменения можно изменить строковое свойство timezone, получить которое можно при помощи java.util.TimeZone или задав допустимое значение из IANA Time Zone Database, Zone Name.

Класс javax.ejb.TimerConfig

Класс javax.ejb.TimerConfig – контейнер, содержащий сериализуемый объект описания таймера и флаг persistent, показывающий, способен ли таймер сохраниться (сериализоваться) при сбое.

Интерфейс Timer

Таймер — объект, реализующий интерфейс javax.ejb.Timer. Он представляет временные события, спланированное при помощи EJB TS. Объекты Timer создаются неявно EJB-контейнером при помощи аннотации @Schedule или явно методами TimerService.create*Timer(…) и доступны через TimerService.getTimers(). Объект Timer также является необязательным аргументом таймаут-методов TimedObject.ejbTimeout(…) и аннотированных @Timeout, т.е. может быть доступен внутри них.

Интерфейс Timer

```
public interface javax.ejb.Timer:

• public void cancel();
• public long getTimeRemaining();
• public java.util.Date getNextTimeout();
• public javax.ejb.ScheduleExpression getSchedule();
• public javax.ejb.TimerHandle getHandle();
• public java.io.Serializable getInfo();
• public boolean isPersistent();
• public boolean isCalendarTimer();
```

Экземпляр Timer представляет единственное временное событие и может использоваться для завершения таймера, получения сериализуемого указателя на таймер, прикладной информации, связанной с таймером, получения момента наступления следующего события.

Остановка таймера

Метод Timer.cancel() используется для остановки таймера, чтобы он никогда не сработал. Также метод можно применить для перепланирования таймера путем его остановки и пересоздания заново.

Идентификация таймера и его описание

В процессе обработки временного события может потребоваться выяснить, какой из таймеров его вызвал. Сравнение текущих состояний таймеров малопригодно по причине постоянного их изменения со временем. Кроме того, для некоторых таймеров может потребоваться хранение дополнительной информации, нужной для обработки генерируемых ими временных событий в таймаут-методе. Для разрешения этих проблем таймеры содержат объект описания, позволяющий идентифицировать таймер и передать полезную информацию обработчику временного события.

В каждом из методов TimerService.create*Timer(…) последним параметром идет объект описания (info object). Для декларативных таймеров, созданных при помощи @Schedule это строка java.lang.String. Для программных таймеров это объект любого типа, реализующий java.io.Serializable, в т.ч. допускается значение null . Объект описания храниться в EJB TS и передается в ejb при вызове соответствующего callback-метода в ответ на срабатывание связанного с ним таймера. Для получения объекта описания используется метод Timer.getInfo(), сериализуемый ответ приводится к нужному типу. Один из самых простых типов объекта описания — строковый, но существуют и более навороченные и реалистичные варианты.

Ex.: пример отправки поздравительного письма

```
@Stateless
public class FlyerBean {
// Прямое внедрение службы таймера
  @Resource
  private TimerService timerService;
  …
// Метод для планирования новой посылки
  public void scheduleFlyer(ScheduleExpression se, Email email) {
    TimerConfig tc = new TimerConfig(se,email);
    Timer timer = timerService.createCalendarTimer(se,tc);
  }

// Объявление таймаут-метода FlyerBean для отправки письма
```

```
  @Timeout
  public void send(Timer timer){
    if(timer.getInfo() instanceOf Email){
      Email email = (Email)timer.getInfo();
      … // посылка письма
    }
  }
}
```

Получение дополнительной информации о таймере

Метод Timer.getNextTimeout() выдает дату java.util.Date. Эта дата означает для одноразовых таймеров момент наступления (истечения) очередного события. Для интервального таймера дата означает время, оставшееся до конца текущего интервала.

Rem: в спецификации EJB 3.1 (JEE6) нет способа получения последующих истечений событий или интервалов. Простейший способ обеспечения такой информацией — самостоятельное помещение ее в объект описания таймера.

Метод Timer.getTimeRemaining() выдает число миллисекунд перед очередным истечением события или интервала.

Объект TimerHandle

Интерфейс Timer является несериализуемым, поскольку таймеры не применяются в EJB с поддержкой сессии . Однако требуется временное сохранение таймера на случай сбоя сервера и это делается при помощи сериализуемого представления javax.ejb.TimerHandle. Метод Timer.getHandle() возвращает объект интерфейса javax.ejb.TimerHandle. Этот объект является представлением объекта Timer и может сериализоваться в файле или другом ресурсе для последующего восстановления доступа к таймеру. Интерфейс TimerHandle имеет следующий вид:

```
public interface javax.ejb.TimerHandle extends java.io.Serializable

• public Timer getTimer( ) throws NoSuchObjectLocalException, EJBException;
```

Единственный метод TimerHandle.getTimer() выдает ссылку на таймер. Объект TimerHandle считается действительным только до истечения события (для одноразовых таймеров) или до остановки таймера. В противном случае метод TimerHandle.getTimer() выбросит исключение javax.ejb.NoSuchObjectLocalException.

TimerHandle является локальным объектом, что подразумевает, что передача его в качестве параметра удаленному методу ошибочна. Объект TimerHandle может быть использован только в рамках EJB-контейнера, который его породил. В частности, разрешается его передача через локальные интерфейсы EJB, поскольку все локальные

объекты ejb находятся в рамках EJB-контейнера.

Исключения Timer

Все методы интерфейса javax.ejb.Timer могут выбрасывать 2 типа проверяемых исключений: javax.ejb.NoSuchObjectLocalException и javax.ejb.EJBException.

Исключение NoSuchObjectLocalException выбрасывается при попытке обращения к любому методу завершенного или истекшего таймера. Исключение EJBException выбрасывается при возникновении любого системного исключения в EJB TS.

Транзакции, безопасность и Timer

Методы create*Timer(…) могут вызываться в рамках транзакции. Если транзакция откатывается, то соответствующий таймер уничтожается, точнее не успевает создаться. Если транзакция помечается на откат уже после запуска в ней таймера, то он останавливается, если еще не успел сработать. Если транзакция, в которой при помощи Timer.cancel() останавливается таймер помечается на откат, то остановка таймера отменяется.

Таймаут-методы также могут выполняться в транзакционном режиме. В большинстве случаев они снабжаются транзакционным режимом REQUIRED или REQUIRES_NEW, что гарантирует их выполнение в рамках контейнерной транзакции. . В рамках контейнерной транзакции, при ее откате EJB-контейнер перезапустит таймер с повторным вызовом таймаут-метода.

Rem: как конкретно EJB-контейнер будет реагировать на откат транзакции таймаут-метода зависит от вендора, например, Glassfish 4.1 отключает таймер после 2-х неудачных (с откатом транзакции) попыток выполнить таймаут-метод.

Поскольку таймеры имеют внесессионную природу, могут создаваться при установке приложения, они не привязаны к клиентам и их callback-методы работают вне контекста безопасности. Если в таком методе запросить getCallerPrincipal(), то он вернет представление неаутентифицированного пользователя. Т.е. вызов защищенного метода из callback-метода таймера выбросит исключение (если не настроены привелегии класса бина через @RunAs).

Таймеры SLSB

Таймеры SLSB часто применяются в задачах учета (проверочных расчетов) и пакетной обработки. SLSB в качестве учетного агента (представителя) отслеживает состояние системы, удостоверяясь, что все задачи выполняются как надо и состояние согласовано. Этот тип обработки обычно включает entity-бины и возможно источники данных. Определенные ejb могут выполнять пакетную обработку вроде очистки РБД или передаче записей. SLSB могут также устанавливаться в качестве интеллектуальных

агентов, которые выполняют некоторые действия от имени организации, которую они обслуживают. Такие агенты можно считать расширенными учетными, поскольку они не только отслеживают систему, но и решают проблемы в автоматическом режиме.

Таймеры SLSB связаны только с определенным типом SLSB. Когда такой таймер срабатывает, EJB-контейнер вытаскивает экземпляр из пула SLSB соответствующего типа и вызывает его временной callback-метод. Это возможно по причине того, что экземпляры SLSB из пула логически эквивалентны. И любой экземпляр может обслуживать любого клиента, включая сам EJB-контейнер.

SLSB-таймеры часто используются для управления выполнением задач, а также, когда надо обрабатывать сразу коллекцию entity-бинов. Например, SLSB-таймер может использоваться в задаче учета технической документации круизных судов на предмет их соответствия государственным нормативам: через определенные интервалы времени таймер уведомляет SLSB, что надо извлечь документацию всех кораблей и составить отчет. Также SLSB-таймер может использоваться для задач вроде оповещения всех пассажиров определенного рейса.

Декларативные таймеры запускаются автоматически после старта приложения.

SLSB может получать доступ к TimerService из внедренного сессионного контекста в любом бизнес-методе или callback-методах жизненного цикла @PostConstruct и @PostCreate. Но не через внедрение извне через set-метод. Это означает, что для гарантированного программного создания таймера через TimerService, клиент должен вызвать либо бизнес-метод, либо добиться помещения в M-R-P экземпляра SLSB.

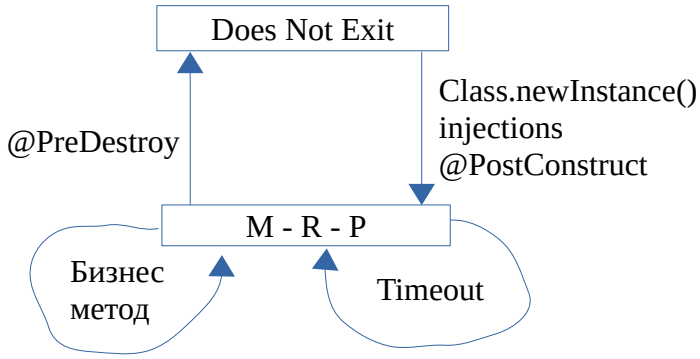
Автоматическое программное создание SLSB-таймера через callback-методы его жизненного цикла проблематично. Во-первых иногда контейнер вызывает @PostConstruct перед помещением экземпляра SLSB в Method-Ready Pool (M-R-P). Однако нет гарантии создания пула до 1-го клиентского вызова, следовательно, @PostConstruct не будет вызван, а таймер создан. Во-вторых @PostConstruct будет вызываться всякий раз, когда экземпляр SLSB будет помещаться в M-R-P. И для пресечения создания таймеров-дублей, надо пресекать их повторные установки. Для этого можно использовать некоторую статическую переменную — флаг, но и тут есть неприятность: в кластеризованных системах с различными экземплярами Application Server и JVM экземпляры SLSB могут создаваться различными загрузчиками (classloader). Статические переменные для объектов классов, загруженных разными загрузчиками могут отличаться, что приведет к дублированию таймеров. В качестве альтернативы можно каждый раз в @PostConstruct получать список уже существующих таймеров TimerService.getTimers() и останавливать их перед созданием нового. Но это затратно частым дерганьем getTimers() и, кроме того, никто не гарантирует, что TimerService работает по всему кластеру и таймеры, установленные в одном его узле, видны в остальных. Также нельзя использовать для остановки и создания таймеров @PreDestroy методы, поскольку они вызываются контейнером только при удалении экземпляра из памяти. Их нельзя вызывать удаленно или локально через клиентский

запрос. К тому же @PreDestroy связан с конкретными экземплярами, а не с SLSB-типом в целом, в результате ничего осмысленного о SLSB по вызову EJB-контейнером javax.ejb.SessionBean.ejbRemove() сказать нельзя.

Для программного автосоздания таймеров лучше подходит Singleton EJB, настроенный на трудолюбивую инициализацию EJB-контейнером.

Когда SLSB включает временные callback-методы TimedObject.ejbTimeout(…) или @Timeout, его жизненный цикл изменяется для обслуживания временных событий. EJB TS берет из пула экземпляр SLSB, когда срабатывает соответствующий таймер. Если пул был пуст, то создается новый экземпляр.

Жизненный цикл SLSB с обработкой временных событий:



Таймеры MDB

По ряду признаков MDB-таймеры похожи на SLSB-таймеры. Во-первых MDB-таймер также относится только к определенному типу MDB в целом, а не к отдельным экземплярам. Во-вторых в ответ на срабатывание MDB-таймера также из пула вытаскивается какой-нибудь экземпляр MDB и он выполняет свой временной (timeout) callback. В-третьих MDB-таймер также удобен для выполнения пакетных задач вроде учета.

Основным отличием от SLSB-таймера является способ инициализации: либо при получении сообщения, либо, если дополнительно допускается вендором, через конфигурационный файл в deploy-time.

Для инициализации таймера достаточно поместить код его создания TimerService.create*Timer(...) в метод обработки сообщения. Для JMS-MDB это метод onMessage().

Ex: Создание MDB-таймера

```
@MessageDriven
public class ExMDB implements MessageListener {
  @Resource TimerService timerService;
```

```
  public void onMessage(Message m) {
    ExMessage em = (ExMessage) m;
    long expirationDate = em.getLong("expirationDate");
    timerService.createTimer(expirationDate,null);
  }

  @Timeout
  public void timeout() { … }
}
```

Предполагается, что входящее сообщение будет содержать информацию о таймере: дату срабатывания/истечения, продолжительность и даже сериализуемый объект описания.

Комбинация JMS и EJB TS дает хорошие возможности для проектирования реализации аудита, пакетной обработки и разнообразных агентов.

Хотя это и не оговорено в спецификации EJB 3.1 (JEE6), некоторые вендоры могут позволить настройку MDB-таймеров в конфигурационных файлах в deploy-time. Преимуществом такого подхода является отсутствие необходимости выполнять какие-либо инициализирующие действия, т.к. установка таймера происходит автоматически. Эти возможности делают MDB-таймеры похожими на задачи UNIX/cron, которые предварительно конфигурируются и потом запускаются. Для получения информации о предварительной настройке таймеров надо изучать документацию вендора.

Как и в случае с SLSB, MDB, содержащий временные callback-методы слегка меняет свой жизненный цикл. Когда происходит временное событие, EJB-контейнер должен извлечь из пула экземпляр MDB и выполнить в нем временной callback-метод. Если пул пуст, то экземпляр MDB перемещается из D-N-E состояния в M-R-P состояние перед тем, как он сможет обработать это временное событие.

Ругательства

- EJB TS = EJB Timer Service = служба таймеров EJB
- EJB-компоненты = набор клас
- Таймаут = timeout = действие, выполняемое в ответ на возникновение временного события
- Таймаут-метод = callback-метод, вызываемый EJB-контейнером в ответ на возникновение временного события

Литература

Enterprise JavaBeans 3.1 (6th edition), Andrew Lee Rubinger, Bill Burke, стр. nnn

Изучаем Java EE 7, Энтони Гонсалвес, стр. nnn

EJB3 в действии, (2е издание), Дебу Панда, Реза Рахман, Райан Куправк, Майкл Ремижан, стр. nnn