Оглавление

[Основные положения 1](#_Toc434329867)

[Реализация 1](#_Toc434329868)

[Инициализация 1](#_Toc434329869)

[Работа 2](#_Toc434329870)

[Описание интерфейсов и доменных объектов 2](#_Toc434329871)

[Активация агента 9](#_Toc434329872)

# Основные положения

Подсистема периодических заданий должна отвечать следующим требованиям:

1. Администратор должен иметь возможность менять расписание заданий, включать и выключать их, менять конфигурацию заданий. Конфигурация заданий это класс, способный сериализоваться в xml для последующего хранения экземпляра этого класса в доменном объекте.
2. Код периодического задания это спринг бин аннотированный специальной аннотацией и имплиментящий определенный интерфейс. Ядро системы при старте найдет все такие бины и зарегистрирует их в подсистеме периодических заданий
3. Периодические задания должны выполняться с учетом приоритета. То есть если одновременно должны запуститься несколько заданий с разным приоритетом, то сначала запускаются задания с более высоким приоритетом, потом с более низким.
4. Периодические задания должны использовать пул процессов, чтобы была возможность выполнять несколько заданий одновременно.
5. Периодические задания бывают двух видов.
   1. Первый вид - это периодическое задание способного существовать в системе в единственном экземпляре. Экземпляры периодических заданий этого вида всегда присутствуют в списке заданий, их нельзя удалить, можно сделать неактивными, можно поменять конфигурацию
   2. Второй вид – это периодические задания способные существовать в системе в нескольких экземплярах. Такие задания после установки в систему не имеют экземпляров, экземпляры таких заданий может создать администратор, установить периодичность, активность, конфигурацию, а так же удалить из системы.

# Реализация

## Инициализация

При старте системы сервисом ScheduleService производится сканирование классов на предмет наличия класса аннотированного анотацией ScheduleTask. Составляется список всех таких классов. В случае если задача относится к классу «Только один экзампляр», то в хранилище создается доменный объект schedule для данного класса и заполняются поля значениями по умолчанию, которые берутся из аннотации ScheduleTask.

Администратор, с помощью вызова методов сервиса ScheduleService. createScheduleTask может создать экземпляры задач класса «Много экземпляров». При создание задач такого класса значения полей доменного объекта schedule заполняются по умолчанию данными, полученными у аннотации ScheduleTask.

## Работа

Разработан Statless EJB - SchedulerBean, с методом аннотированным @Schedule и настроенным на выполнение один раз в минуту. Данный EJB является инициатором выполнения всех периодических заданий. Разработан Statless EJB – ScheduleProcessor с асинхронным методом startAsync. Данный EJB асинхронно выполняет задания в пуле процессов. Создан тип доменного объекта schedule который хранит расписание, активность, параметры, приоритет и статус периодического задания. Раз в минуту вызывается методы SchedulerBean в котором производятся следующие действия:

1. Просматривается список запущенных задач и в случае выполнения их больше положенного времени задача прерывается и соответствующий статус сохраняется в доменном объекте задачи. Для прерывания используется объект класса Future, возвращаемый методом асинхронного запуска задач. Данные действия выполняются в методе checkTimeoutBackgroundProcessing и сдвинуты относительного старта задачь на 10 секунд.
2. Анализируются записи объектов schedule и производится вычисление тех задач, которые надо запустить в данный момент времени. Задачи сортируются по приоритету.
3. Из тех списков задач, которые надо запустить в текущий момент времени удаляются те задачи, которые еще выполняются.
4. У оставшихся задач устанавливается статус READY.
5. Зачитываются задачи в статусе READY (на текущий момент все, в будущем возможно надо будет зачитывать не все, а только первые N с наивысшим приоритетом, для того чтобы не забивать пул потоков), им устанавливается статус WAIT и отдаются на выполнение путем вызова асинхронного метода ScheduleProcessor.
6. ScheduleProcessor выполняет задачу и по окончании устанавливает статус SLEEP

Все задачи выполняются в одном пуле потоков. Размер пула потоков задается в конфигурации сервера приложений.

Задачи выполняются от имени системы.

## Описание интерфейсов и доменных объектов

Интерфейс администрирования сервиса периодических заданий

package ru.intertrust.cm.core.business.api;

import java.util.List;

import ru.intertrust.cm.core.business.api.dto.DomainObject;

import ru.intertrust.cm.core.business.api.dto.Id;

import ru.intertrust.cm.core.business.api.schedule.Schedule;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskParameters;

/\*\*

\* Сервис периодических заданий

\* @author larin

\*

\*/

public interface ScheduleService {

public static final String SCHEDULE\_NAME = "name";

public static final String SCHEDULE\_TASK\_CLASS = "task\_class";

public static final String SCHEDULE\_TASK\_TYPE = "task\_type";

public static final String SCHEDULE\_YEAR = "year";

public static final String SCHEDULE\_MONTH = "month";

public static final String SCHEDULE\_DAY\_OF\_MONTH = "day\_of\_month";

public static final String SCHEDULE\_DAY\_OF\_WEEK = "day\_of\_week";

public static final String SCHEDULE\_HOUR = "hour";

public static final String SCHEDULE\_MINUTE = "minute";

public static final String SCHEDULE\_TIMEOUT = "timeout";

public static final String SCHEDULE\_PRIORITY = "priority";

public static final String SCHEDULE\_PARAMETERS = "parameters";

public static final String SCHEDULE\_LAST\_RESULT = "last\_result";

public static final String SCHEDULE\_LAST\_RESULT\_DESCRIPTION = "last\_result\_description";

public static final String SCHEDULE\_ACTIVE = "active";

public static final String SCHEDULE\_LAST\_REDY = "last\_redy";

public static final String SCHEDULE\_LAST\_WAIT = "last\_wait";

public static final String SCHEDULE\_LAST\_RUN = "last\_run";

public static final String SCHEDULE\_LAST\_END = "last\_end";

public static final String SCHEDULE\_STATUS\_SLEEP = "Sleep";

public static final String SCHEDULE\_STATUS\_READY = "Ready";

public static final String SCHEDULE\_STATUS\_WAIT = "Wait";

public static final String SCHEDULE\_STATUS\_RUN = "Run";

/\*\*

\* Удаленный интерфейс

\* @author larin

\*

\*/

public interface Remote extends ScheduleService{

}

/\*\*

\* Получение всех задач, которые обрабатываются сервисом периодических заданий.

\* @return

\*/

List<DomainObject> getTaskList();

/\*\*

\* Получение классов задач которые могут существовать во множественном числе экземпляров. Используется GII дла

\* отрисовки диалога создания периодического задания

\* @return

\*/

List<String> getTaskClasses();

/\*\*

\* Получение расписания задачи

\* @return

\*/

Schedule getTaskSchedule(Id taskId);

/\*\*

\* Установка расписания задачи

\* @param taskId

\* @param schedule

\*/

void setTaskSchedule(Id taskId, Schedule schedule);

/\*\*

\* Получение параметров задачи

\* @return

\*/

ScheduleTaskParameters getTaskParams(Id taskId);

/\*\*

\* Установка параметров задачи

\* @param taskId

\* @param schedule

\*/

void setTaskParams(Id taskId, ScheduleTaskParameters parameters);

/\*\*

\* Активировать задание

\* @param taskId

\*/

void enableTask(Id taskId);

/\*\*

\* Деактивировать задание

\* @param taskId

\*/

void disableTask(Id taskId);

/\*\*

\* Запустить задание

\* @param taskId

\*/

void run(Id taskId);

/\*\*

\* Установка приоритета. Значения могут быть от 0 до 4

\* @param priority

\*/

void setPriority(Id taskId, int priority);

/\*\*

\* Установка таймаута в минутах

\* @param timeout

\*/

void setTimeout(Id taskId, int timeout);

/\*\*

\* Создание периодического задания класса multible

\* @param string

\* @return

\*/

DomainObject createScheduleTask(String className, String name);

}

Интерфейс ScheduleTaskHandle, данный интерфейс должен имплиментить класс выполняющий задачу

**package** ru.intertrust.cm.core.business.api.schedule;

/\*\*

\* Интерфейс, который должны имплиментить все классы периодических заданий

\* **@author** larin

\*

\*/

**public** **interface** ScheduleTaskHandle {

/\*\*

\* Запуск выполнения периодического задания

\* **@param** parameters

\* **@return** возвращает результат работы периодического задания в виде строки. Строка будет хранится в доменном обете задания

\*/

String execute(ScheduleTaskParameters parameters);

}

Интерфейс ScheduleTaskDefaultParameters, данный интерфейс должен иплиментить класс описывающий настройку задачи по умолчанию

**package** ru.intertrust.cm.core.business.api.schedule;

/\*\*

\* Интерфейс конфигурации периодического задания по умолчанию.

\* Наследники используются в анотации ScheduleTask в качестве параметра

\* **@author** larin

\*

\*/

**public** **interface** ScheduleTaskDefaultParameters {

/\*\*

\* Получение параметров по умолчанию

\* **@return**

\*/

ScheduleTaskParameters getDefaultParameters();

}

Доменный объект schedule

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Коментарии** |
| name | String | Имя задания |
| task\_class | String | Имя java класса задачи. По данному имени будет производится поиск в контексте спринг бинов для поиска нужного экземпляра. |
| task\_type | Long | Тип задачи. 0 – может существовать в единствееном экземпляре. 1 – может существовать во множестве экземпляров. |
| day\_of\_week | String | День недели старта. \* - всегда, число – когда равен числу, \*/число когда число кратно числу в дроби (как в crond) |
| day\_of\_month | String | День месяца |
| hour | String | Час старта |
| minute | String | Минута старта |
| month | String | Месяц старта |
| year | String | Год |
| timeout | Long | Допустимое время работы (в минутах) |
| Priority | Long | Приоритет задачи. Может принимать значения от 0 до 4. Наивысший приоритет – 0, наинизший приоритет – 4. |
| parameters | String | Сеарелизованные в строку параметры задачи |
| Status | Reference | Системный атрибут статуса. Может иметь значения:  SLEEP – не запущен;  READY-готов к выполнению;  WAIT- ожидает выполнения в пуле потоков;  RUN-В работе. |
| last\_result | long | 1. Никогда не запускался 2. отработал без ошибки, 3. прерван в результате ошибки внутри задачи, 4. прерван в связи с превышением допустимого времени работы |
| last\_result\_description | String | Строковое описание последнего результата работы. Здесь сохраняется результат метода execute интерфейса ScheduleTaskHandle или ошибка в случае завершения работы с ошибкой |
| active | Long | 1. задача активна и запустится по расписанию 2. задача не активна |
| last\_redy | Date | Время крайней установки флага готов к выполнению |
| last\_wait | Date | Время крайнего добавления в очередь на выполнение |
| last\_run | Date | Время крайнего запуска на исполнение |
| last\_end | Date | Время окончания исполнения при крайнем запуске |

Пример периодического задания.

TestSingleSchedule – Single задание выполняющееся по умолчанию один раз в минуту

package ru.intertrust.cm.test.schedule;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTask;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskHandle;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskParameters;

import ru.intertrust.cm.core.model.ScheduleException;

@ScheduleTask(name = "TestSingleSchedule", minute = "\*/1")

public class TestSingleSchedule implements ScheduleTaskHandle {

@Override

public String execute(ScheduleTaskParameters parameters) {

try {

System.out.println("Run TestSingleSchedule");

Thread.currentThread().sleep(10000);

return "COMPLETE";

} catch (Exception ex) {

throw new ScheduleException("Error exec TestSingleSchedule", ex);

}

}

}

Множественное задание выполняющееся один раз в минуту

package ru.intertrust.cm.test.schedule;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTask;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskHandle;

import ru.intertrust.cm.core.business.api.schedule.ScheduleTaskParameters;

import ru.intertrust.cm.core.business.api.schedule.SheduleType;

import ru.intertrust.cm.core.model.ScheduleException;

@ScheduleTask(name = "TestScheduleMultiple", minute = "\*/1", configClass = TestSheduleDefaultParameter.class,

type = SheduleType.Multipliable)

public class TestScheduleMultiple implements ScheduleTaskHandle {

@Override

public String execute(ScheduleTaskParameters parameters) {

try {

TestScheduleParameters testScheduleParameters = (TestScheduleParameters) parameters;

//Тестируем обработчик ошибки в поле Result может быть как строка так и число. Если строка то упадем с ошибкой, что должны увидеть в результатах выполнения задачи

long value = Long.parseLong(testScheduleParameters.getResult());

System.out.println("Run TestScheduleMultiple");

Thread.currentThread().sleep(value);

return testScheduleParameters.getResult();

} catch (Exception ex) {

throw new ScheduleException("Error exec TestScheduleMultiple", ex);

}

}

}

# Активация агента

После старта платформы сервис отключен и надо его включать принудительно. Сделать это можно вызвав EJB метод ScheduleTaskLoader. setEnable(true), или добавив в спринг контекст прикладного приложения бин ScheduleServiceActivator например так через beans.xml

<bean id="scheduleServiceActivator" class="ru.intertrust.cm.core.business.api.schedule.ScheduleServiceActivator" />

Сделано это для того чтобы можно было управлять активностью сервиса из дочерних, по отношению к платформе приложений. В частности не позволять запускаться периодическим заданиям до окончания инициализации приложения.