Scheduled task with parameters

scheduler

CzechIdM supports three types of the so-called Scheduled tasks - business logic execution tasks running asynchronously outside of the main thread. There are following types:

- Long Running task - run from code, cannot be scheduled
- Scheduled task - run from Quartz scheduler, extends LRT
- Stateful task - run from Quartz scheduler, extends Scheduled task

You can find description of each tasks type in the Task scheduler chapter.

Scheduled task

Simple scheduled tasks are the mostly used type of tasks. These are automatically registered in CzechIdM's scheduler implementation and are easily configurable from GUI. Following is an example of such task with parameters, which is supposed to read CSV from path passed as parameter and process it (not implemented for brevity purposes). Notice the `IMPORTANT` comments:

```
package eu.bcvsolutions.idm.tutorial.scheduler;

import ...;

// IMPORTANT 1
@Component
@Description("Parses input CSV (path as parameter) and assigns users roles
from CSV. Only role assignment is allowed.")
// IMPORTANT 2
public class ImportCSVUserRolesTask extends
AbstractSchedulableTaskExecutor<Boolean> {

    private static final Logger LOG =
LoggerFactory.getLogger(ImportCSVUserRolesTask.class);
    private static final String PARAM_CSV_FILE_NAME = "rolesCSV";
    private static final String CSV_USERNAME_HEADER = "username";
    private static final String CSV_ROLE_HEADER = "role";

    private String csvPath;

    @Autowired private IdmIdentityService identityService;
    @Autowired private IdmRoleService roleService;
    @Autowired private IdmIdentityRoleService identityRoleService;
    @Autowired private IdmRoleRequestService roleRequestService;
    @Autowired private IdmConceptRoleRequestService
conceptRoleRequestService;
    @Autowired private IdmIdentityContractService identityContractService;

        // IMPORTANT 3
    @Override
```

```
    public Boolean process() {
                // path to CSV is required, also check file exists and is
readable
        if (csvPath == null) {
            throw new IllegalArgumentException("CSV path must be defined.");
        }
        //
                // TODO processing :)
                //
                return Boolean.TRUE;
    }

        // IMPORTANT 4
    @Override
    public List<String> getPropertyNames() {
        List<String> params = super.getPropertyNames();
        params.add(PARAM_CSV_FILE_NAME);
        return params;
    }

        // IMPORTANT 5
    @Override
    public void init(Map<String, Object> properties) {
        super.init(properties);
        csvPath = getParameterConverter().toString(properties,
PARAM_CSV_FILE_NAME);
    }
}
```

While implementing scheduled tasks, don't forget those 5 important pieces marked as comments in the code above:

- IMPORTANT 1 - all scheduled tasks are Spring Beans, don't forget to annotate the class as Spring's `@Component`
- IMPORTANT 2 - all scheduled tasks extend the `AbstractSchedulableTaskExecutor` class
- IMPORTANT 3 - the `process` method contains the business logic you need to implement and is run by CzechIdM's scheduler
- IMPORTANT 4 - input parameters are defined by their names as list in the `getPropertyNames` method
- IMPORTANT 5 - input parameters (instance variables) are initialized in the `init` method, which is run before the `process`. Parameters and not strongly typed, `use parameters converter` for conversions.

Stateful tasks

It is also easily possible to execute workflow from a stateful task. This is for example heavily used for personal processes, where workflows offer great flexibility for customer specific customization. Following is an example of such task, which executes a workflow.

```
package eu.bcvsolutions.idm.tutorial.scheduler;
```

```
import ...;

// IMPORTANT 1
@Service
@Description("My workflow process")
@DisallowConcurrentExecution
// IMPORTANT 2
public class MyWorkflowProcess extends
AbstractWorkflowStatefulExecutor<IdmIdentityContractDto> {

    private static final String PROCESS_NAME = "myWorkflowProcess";

    @Autowired
    private IdmIdentityContractService identityContractService;

    /**
     * Find all identity contracts, that are both valid and enabled.
     */
        // IMPORTANT 3
    @Override
    public Page<IdmIdentityContractDto> getItemsToProcess(Pageable pageable)
{
        IdentityContractFilter filter = new IdentityContractFilter();
        filter.setValid(Boolean.TRUE);
        filter.setDisabled(Boolean.FALSE);
        return identityContractService.find(filter, pageable);
    }

        // IMPORTANT 4
    @Override
    public String getWorkflowName() {
        return PROCESS_NAME;
    }
}
```

And following is the content of the Activiti workflow, which handles the business logic. There are always 3 input variables of the workflow:

- entity DTO which will be processed
- Long Running task's ID
- Stateful task ID

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:activiti="http://activiti.org/bpmn"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
```

```
typeLanguage="http://www.w3.org/2001/XMLSchema"
expressionLanguage="http://www.w3.org/1999/XPath"
targetNamespace="http://www.bcvsolutions.eu/testDeployAndRun">
  <process id="myWorkflowProcess" name="My workflow process - does something
awesome!" isExecutable="true">
    <documentation>My workflow process

// IMPORTANT 5
Input:
    longRunningTaskId - UUID
    scheduledTaskId - UUID
    dto - IdmIdentityContractDto</documentation>
    <startEvent id="startevent1" name="Start"></startEvent>
    <endEvent id="endevent" name="End"></endEvent>
    <sequenceFlow id="flow1" sourceRef="startevent1"
targetRef="scripttask1"></sequenceFlow>
    <scriptTask id="scripttask1" name="enable contract script"
scriptFormat="groovy" activiti:autoStoreVariables="false">
      <script>//
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import eu.bcvsolutions.idm.core.api.domain.OperationState;
import eu.bcvsolutions.idm.core.api.entity.OperationResult;
import
eu.bcvsolutions.idm.core.scheduler.api.service.SchedulableStatefulExecutor;

Logger LOG = LoggerFactory.getLogger("myWorkflowProcess")
LOG.info("long running tID: [{}], scheduled tID: [{}], dto: [{}]",
longRunningTaskId, scheduledTaskId, dto)

// TODO processing ... :)

def result = new OperationResult.Builder(OperationState.EXECUTED).build()

// IMPORTANT 6
execution.setVariable("success",
OperationState.isSuccessful(result.getState()))
execution.setVariable(SchedulableStatefulExecutor.OPERATION_RESULT_VAR,
result)
//
// process end
</script>
    </scriptTask>
    <exclusiveGateway id="exclusivegateway1" name="Exclusive
Gateway"></exclusiveGateway>
    <sequenceFlow id="flow3" sourceRef="scripttask1"
targetRef="exclusivegateway1"></sequenceFlow>
    <sequenceFlow id="flow4" name="[FAILURE]" sourceRef="exclusivegateway1"
targetRef="scripttask2">
      <conditionExpression xsi:type="tFormalExpression"><![CDATA[${success
== false}]]></conditionExpression>
```

```
    </sequenceFlow>
    <scriptTask id="scripttask2" name="handle failure script"
scriptFormat="groovy" activiti:autoStoreVariables="false">
      <script>println "FAILURE!"</script>
    </scriptTask>
    <sequenceFlow id="flow5" name="[SUCCESS]" sourceRef="exclusivegateway1"
targetRef="scripttask3">
      <conditionExpression xsi:type="tFormalExpression"><![CDATA[${success
== true}]]></conditionExpression>
    </sequenceFlow>
    <sequenceFlow id="flow6" sourceRef="scripttask2"
targetRef="endevent"></sequenceFlow>
    <sequenceFlow id="flow7" sourceRef="scripttask3"
targetRef="endevent"></sequenceFlow>
    <scriptTask id="scripttask3" name="handle success script"
scriptFormat="groovy" activiti:autoStoreVariables="false">
      <script>println "SUCCESS!"</script>
    </scriptTask>
  </process>
  <!-- BPMN 2.0 diagram elements -->
</definitions>
```

While implementing stateful tasks with workflows, don't forget those important pieces marked as comments in the code above:

- IMPORTANT 1 - all workflow driven tasks are Spring Beans, don't forget to annotate the class as Spring's `@Component`
- IMPORTANT 2 - all workflow driven tasks extend the `AbstractWorkflowStatefulExecutor` class
- IMPORTANT 3 - the `getItemsToProcess` method returns paged search criteria for all entities that should be processed
- IMPORTANT 4 - the `getWorkflowName` method returns the name of the Activiti workflow that will be run
- IMPORTANT 5 - input parameters of the workflow
- IMPORTANT 6 - execution result, the abstract base class `AbstractWorkflowStatefulExecutor` expects the OperationResult as a result of the workflow, for details see stateful task

Scheduled and stateful tasks are a large topic in CzechIdM, definitely have a look at the documentation of Task scheduler and also see few implemented tasks in the `core` module, for example:

- Enable contract process
- Test for Enable contract process
- Role expiration task
- Test for Role expiration task