Flowable6.4 –

> Flowable6.4 –

Flowable

Flowable

```
runtimeService.addMultiInstanceExecution(String activityId, String parentExecutionId,
Map<String, Object> executionVariables)
```

```
AddMultiInstanceExecutionCmd(activityId, parentExecutionId, executionVariables)
```

```
activityId


parentExecutionId        proInstId


executionVariables
```

`AddMultiInstanceExecutionCmd` `execute()`

```
@Override
public Execution execute(CommandContext commandContext) {
    ExecutionEntityManager executionEntityManager =
CommandContextUtil.getExecutionEntityManager();
    //  multi instance execution  IS_MI_ROOT
    ExecutionEntity miExecution = searchForMultiInstanceActivity(activityId,
parentExecutionId, executionEntityManager);

    if (miExecution == null) {
        throw new FlowableException("No multi instance execution found for activity id " +
activityId);
    }

    if (Flowable5Util.isFlowable5ProcessDefinitionId(commandContext,
miExecution.getProcessDefinitionId())) {
        throw new FlowableException("Flowable 5 process definitions are not supported");
    }
    //
    ExecutionEntity childExecution =
executionEntityManager.createChildExecution(miExecution);
    childExecution.setCurrentFlowElement(miExecution.getCurrentFlowElement());
    // BPMN
    BpmnModel bpmnModel =
ProcessDefinitionUtil.getBpmnModel(miExecution.getProcessDefinitionId());
    Activity miActivityElement = (Activity)
bpmnModel.getFlowElement(miExecution.getActivityId());
    MultiInstanceLoopCharacteristics multiInstanceLoopCharacteristics =
miActivityElement.getLoopCharacteristics();
    //     nrOfInstances
    Integer currentNumberOfInstances = (Integer)
miExecution.getVariable(NUMBER_OF_INSTANCES);
    miExecution.setVariableLocal(NUMBER_OF_INSTANCES, currentNumberOfInstances + 1);
    //
    if (executionVariables != null) {
        childExecution.setVariablesLocal(executionVariables);
    }
    //             Task
    if (!multiInstanceLoopCharacteristics.isSequential()) {
        miExecution.setActive(true);
        miExecution.setScope(false);

        childExecution.setCurrentFlowElement(miActivityElement);
        CommandContextUtil.getAgenda().planContinueMultiInstanceOperation(childExecution,
miExecution, currentNumberOfInstances);
    }

    return childExecution;
}
```

Flowable

```
runtimeService.deleteMultiInstanceExecution(String executionId, boolean executionIsCompleted)
```

```
DeleteMultiInstanceExecutionCmd(executionId, executionIsCompleted)
```

```
executionId
```

```
executionIsCompleted
```

DeleteMultiInstanceExecutionCmd execute()

```
@Override
public Void execute(CommandContext commandContext) {
    ExecutionEntityManager executionEntityManager =
CommandContextUtil.getExecutionEntityManager();
    ExecutionEntity execution = executionEntityManager.findById(executionId);
    //  BPMN
    BpmnModel bpmnModel =
ProcessDefinitionUtil.getBpmnModel(execution.getProcessDefinitionId());
    Activity miActivityElement = (Activity)
bpmnModel.getFlowElement(execution.getActivityId());
    MultiInstanceLoopCharacteristics multiInstanceLoopCharacteristics =
miActivityElement.getLoopCharacteristics();

    if (miActivityElement.getLoopCharacteristics() == null) {
        throw new FlowableException("No multi instance execution found for execution id " +
executionId);
    }

    if (!(miActivityElement.getBehavior() instanceof MultiInstanceActivityBehavior)) {
        throw new FlowableException("No multi instance behavior found for execution id " +
executionId);
    }

    if (Flowable5Util.isFlowable5ProcessDefinitionId(commandContext,
execution.getProcessDefinitionId())) {
        throw new FlowableException("Flowable 5 process definitions are not supported");
    }
    //
    ExecutionEntity miExecution = getMultiInstanceRootExecution(execution);
    executionEntityManager.deleteChildExecutions(execution, "Delete MI execution", false);
    executionEntityManager.deleteExecutionAndRelatedData(execution, "Delete MI execution",
false);
    //
    int loopCounter = 0;
    if (multiInstanceLoopCharacteristics.isSequential()) {
        //
        SequentialMultiInstanceBehavior miBehavior = (SequentialMultiInstanceBehavior)
miActivityElement.getBehavior();
        loopCounter = miBehavior.getLoopVariable(execution,
miBehavior.getCollectionElementIndexVariable());
    }
    //                  +1      +1
    //                  -1
    if (executionIsCompleted) {
        Integer numberOfCompletedInstances = (Integer)
miExecution.getVariable(NUMBER_OF_COMPLETED_INSTANCES);
        miExecution.setVariableLocal(NUMBER_OF_COMPLETED_INSTANCES,
numberOfCompletedInstances + 1);
        loopCounter++;

    } else {
        Integer currentNumberOfInstances = (Integer)
miExecution.getVariable(NUMBER_OF_INSTANCES);
        miExecution.setVariableLocal(NUMBER_OF_INSTANCES, currentNumberOfInstances - 1);
    }
    //         (            )
    ExecutionEntity childExecution =
executionEntityManager.createChildExecution(miExecution);
    childExecution.setCurrentFlowElement(miExecution.getCurrentFlowElement());
    //           Task       loopCounter
    if (multiInstanceLoopCharacteristics.isSequential()) {
        SequentialMultiInstanceBehavior miBehavior = (SequentialMultiInstanceBehavior)
miActivityElement.getBehavior();
        miBehavior.continueSequentialMultiInstance(childExecution, loopCounter,
childExecution);
    }

    return null;
}
```

Task

Revision #1
Created 30 July 2020 14:49:17 by
Updated 30 July 2020 14:53:21 by