The AbstractUsecase Class

Table of contents

1. Introduction

When you implement a custom usecase, you're very likely to extend `org.apache.lenya.cms.usecase.AbstractUsecase`. This class provides a set of functionality to simplify the implementation of usecases.

2. Configuration

The usecase is configured in `cocoon.xconf`. A typical configuration looks like this:

```
<component-instance name="edit.forms" logger="lenya.publication"
                    class="org.apache.lenya.cms.editors.forms.FormsEditor">
  <transaction policy="pessimistic"/>
  <view template="edit/forms/forms" menu="false"/>
</component-instance>
```

The following configuration options are available:

- Element `<transaction>` (optional)
  - Attribute `policy = (optimistic | pessimistic)`

  This element is used to determine the transaction policy of the usecase (*optimistic* or *pessimistic offline lock*). It can be omitted, the default is optimistic behaviour. You should only use pessimistic behaviour for complex usecases when the user can't afford to lose all changes.
- Element `<parameter>` (arbitrary number)
  - Attribute `name`
  - Attribute `value`

  An arbitrary number of initial parameters can be passed to the usecase.
- Element `<view>` (optional)
  - Attribute `template` (required)
  - Attribute `menu = (true | false)` (optional)
  - Element `<parameter name="..." value="..."/>` (arbitrary number)

  This element declares the view of the usecase. The `template` attribute points to the JX template to use, relatively to the `lenya/usecases` directory. The suffix `.jx` is added automatically. The attribute `menu` determines if the menu should be visible. It can be omitted and defaults to `false`.
- Element `<exit>` (optional)
  - Attribute `usecase` (required)
  - Element `<parameter name="..." value="..."/>` (arbitrary number)

  This element declares the exit usecase. By default, a usecase exits on the URL it was started from, without any request parameters. Using this method, is is possible to specify a usecase that should be called after this usecase has exited. Additional parameter elements can be supplied.

3. Extending AbstractUsecase

The following methods of the `AbstractUsecase` class are meant to be overridden or invoked by subclasses:

- `protected void initParameters()`

  This method is called to initialize the parameters of the usecase. For instance, if your usecase shall display meta data on the view screen, `initParameters()` reads the meta data from the document and puts them into the parameter map using `setParameter(String, Object)` to make them available to the JX template.

  Note that you can't access the request parameters in this method yet, because it is executed before the request parameters are passed to the usecase.

- `protected void doCheckPreconditions()`

  The method `checkPreconditions()` is a template method which calls this method. For details on `checkPreconditions()`, see section Overview (index.html) .

- `protected void doCheckExecutionConditions()`

  The method `checkExecutionConditions()` is a template method which calls this method. For details on `checkExecutionConditions()`, see section Overview (index.html) .

- `protected void doCheckPostconditions()`

  The method `checkPostonditions()` is a template method which calls this method. For details on `checkPostonditions()`, see section Overview (index.html) .

- `protected void doExecute()`

  The method `execute()` is a template method which calls this method. For details on `execute()`, see section Overview (index.html) .

- `public void advance()`

  For details on `advance()`, see section Overview (index.html) .

- `protected Transactionable[] getObjectsToLock()`

  This method is supposed to return all objects which should be locked before the usecase is started. If the transaction policy is *pessimistic offline lock*, these objects are checked out immediately.

- `protected void setExitParameter(String name, String value)`

  Call this method to set a parameter that should be added to the exit request. For instance, imagine a usecase which edits a user's groups. The exit usecase might be *userProfile*, with the additional parameter *userId*.