Documentation esup-utils-channels-mag

Présentation

- Afin de simplifier et de clarifier la conception de canaux nous allons séparer le code entre 2 catégories de classe : la channel principale (mainchannel) et une ou plusieurs channels secondaires (subchannel).
- L'application elle-même sera découpée en actions : les subchannels seront chargées du rendu d'une ou plusieurs actions et la mainchannel jouera le rôle du chef d'orchestre en déléguant l'affichage à telle ou telle action
- La définition des actions se fera dans le fichier de configuration de la channel et la prise en charge de la lecture de ces actions est faite dans la classe ConfigChannel qu'il suffit ensuite d'étendre.
- L'interêt de ce découpage en actions est qu'il est alors possible d'ajouter des actions, de modifier la classe chargée du rendu d'une action simplement en changeant le fichier de paramètres sans toucher à la mainchannel.
- Par ailleurs ce framework propose les fonctionnalités suivantes :
    - Gestion "propre" des erreurs via une exception permettant un affichage personnalisé et l'envoi éventuel de mails à la personne étant identifiée comme responsable technique.
    - Gestion du cache avec un comportement par défaut pouvant être modifié.
    - Gestion du téléchargement de documents (implémentation de IMimeResponse) qui est délégué à l'action en cours.
    - Gestion de plug-in permettant d'ajouter des actions toutes faites à une channel.
    - Gestion d'une redirection permettant à une action de se rediriger vers une autre selon les besoins fonctionnels
    - Passage par défaut de paramètres XSL à la feuille de style XSL :
        - baseActionURL qui estle lien vers la channel elle-même
        - prefForm qui est une chaine pouvant servir de prefixe à des éléments javascript afin de les rendre unique
        - mediaPath qui est le chemin vers les images pour la channel
    - Passage optionnel du paramètre XSL pour les liens de type téléchargement baseDownloadURL
    - Prise en charge des actions de type servant de manière automatisée et intégration éventuelle du cache du servant
- Les clases d'implémentation de ce framework sont disponibles dans le package org.esupportail.portal.utils.channels

La définition des actions

- Les actions sont définies dans le fichier de configuration avec le format suivant :

```
<!--
          <action *name="nom logique de l'action"
                        type="type de l'action : normal ou servant - normal par defaut"
                        *classname="classe de type SubChannel a  instancier pour realiser l'action"
                        rendertype="type de rendu : xsl ou ssl - xsl par defaut"
                        xslfile="fichier xsl a  utiliser si rendertype=xsl - par defaut name.xsl"
                        sslfile="nom du fichier ssl a  utiliser - null par defaut"
                        ssltitle="entree a  utiliser dans le fichier ssl - par defaut le nom de l'action"
                        init="methode d'initialisation de l'action - init par defaut"
                        setoutput="methode qui fixe les paramaˉtres xsl/xml/ssl - setOutput par defaut"
                        setxml="methode qui fixe le XML - setXML par defaut"
                        renderxml="methode qui effectue le rendu XML - renderXML par defaut"
                        servantfinish="action sur laquelle passer lorsque le servant est fini. Cas ou
type=servant"
                        servantinit="action sur laquelle le servant démarre (cas ou le servant est conçu avec
MAG). default par défaut. Cas ou type=servant"
                        cachetype="type de cache (default|channel|instance) - default par defaut"
                        <param name="nom du param1">
                                <value>premiere valeur du param1</value>
                                <value>deuxieme valeur du param1</value>
                                ...
                        </param>
                        ...
                        <param name="nom du paramN">
                                <value>valeur du paramN</value>
                        </param>
                />
        * = obligatoire
        toutes les methodes sont eventuellement implementees dans classname
-->
```

- Il existe au moins une action nommée default qui sera la première action affichée par la channel
- Le changement d'action se fait à l'aide d'une variable get ou post nommée action. Il est obligatoire de passer cette variable dans les formulaires ou les liens qui font référence à une channel.

Paramètres des actions

- Il est possible d'associer une liste de paramètres à chaque action
- Chaque paramètre est identifié par un nom (attribut name) et peut être multivalué (balises value)
- La classe MainChannel fournit la méthode getActionParam qui combinée aux méthodes getValue ou getValues permet de retrouver la ou les valeurs d'un paramètre connaissant son nom et le nom de l'action

Le cycle de vie d'une channel dans le framework MAG

- La méthode setRuntimeData de la mainChannel se charge de définir quelle est l'action en cours (default par défaut!)
- En fonction du type de l'action, la mainChannel instancie un servant ou une subchannel (dans ce cas un mécanisme de cache évite de réinstancier une action systématiquement)
- Dans le cas d'un servant c'est la méthode setRuntimeData de celui-ci qui est alors appelée
- Dans le cas d'une subchannel, 3 méthodes sont successivement appelées :
    - init qui sert à initialiser la subchannel : cela peut servir à faire des contrôles d'accès par exemple
    - setXML qui n'est appelé que que si init a renvoyé Boolean.TRUE et qui sert à positionner la chaine XML qui va être utilisée dans le rendu de la subchannel
    - setOutput qui n'est appelé que si setXML a renvoyé Boolean.TRUE et qui sert à positioner le nom du fichier XSL ou SSL, du titre SSL...
- Dans le cas où la subChannel était déjà instanciée on réutilise l'instance mais avant on appelle la méthode clearChannel qui peut servir à réinitialiser la channel.

Conception du fichier de configuration

- Elaborer son fichier de configuration xml et la dtd sur le modèle fourni dans le répertoire properties du framework.
- Créer une classe Config qui étend la classe ConfigChannel
- Implémenter le mécanisme du singleton dans cette classe
- Implémenter la méthode getConfigFile qui doit retourner l'emplacement du fichier de configuration xml
- Eventuellement implémenter la méthode customDigester qui fait des manipulations sur l'objet dig correspondant à une instance de Digester.

Conception de la mainchannel

- Elle doit étendre la classe MainChannel
- Comme pour une channel classique, les méthodes getRuntimeProperties, receiveEvent et setStaticData sont à implémenter.

Constructeur

- Utiliser la méthode setConfigActions pour récupérer la liste des actions lues dans le fichier de configuration
- Eventuellement enregistrer les différents plugins dont on aura besoin dans le constructeur

setRuntimeData (facultatif)

- Sauf si l'on a besoin d'un comportement spécifique, elle n'est pas à implémenter
- Dans le cas contraire ne pas oublier d'appeller la méthode super (après les traitements spécifiques)

Conception des subchannels

- Elles doivent étendre la classe SubChannel
- Elles possèdent une variable privée owner du type de la classe de la mainChannel

Constructeur

- Il appelle la méthode super

Méthode init (ou un autre nom défini pour l'action)

- Elle doit renvoyer Boolean.TRUE si le cycle de vie de la subChannel peut se poursuivre (exécuter setXML)
- Elle doit renvoyer Boolean.FALSE si une redirection est faite soit par l'appel d'un plugin soit par l'appel de la méthode redirect.
- L'implémentation de cette méthode est facultative. Par défaut elle renvoie Boolean.TRUE

Méthode setXML (ou un autre nom défini pour l'action)

- Elle sert à valuer la variable xml de la subChannel qui contient le xml à utiliser lors de la transformation xslt
- Il est facultatif de définir l'enveloppe du xml. Si elle n'est pas définie le xml généré sera enveloppé dans :

```
<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>
<xml>
le xml généré dans setXML
</xml>
```

- L'implémentation de cette méthode est facultative. Par défaut elle renvoie la chaine xml suivante :

```
<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>
<xml>
</xml>
```

- Elle renvoie Boolean.TRUE si le cycle de vie de la subChannel peut se poursuivre (exécuter setOutput)

Méthode setOutput (ou un autre nom défini pour l'action)

- Elle positionne les variables ssl et xsl de la subChannel
- Dans le cas d'un rendu ssl, setSSL doit être appelé pour positionner le fichier ssl et setXSL pour positionner le title à utiliser dans ce fichier
- Dans le cas d'un rendu xsl, setXSL doit être appelé pour positionner le fichier xsl à utiliser
- L'implémentation de cette méthode est facultative. Par défaut elle utilise les paramètres rendertype, xslfile, sslfile et xsltitle définis dans le fichier de configuration.

Gestion du cache des actions

- La classe MainChannel implémente l'interface ICacheable donc elle est prise en charge automatiquement par le cache uPortal
- Il est possible de paramétrer 3 fonctionnements pour le cache des actions : default, channel et instance

Le cache de type default

- Comme son nom l'indique, c'est ce type de cache qui est pris en charge lorsque vous ne précisez rien pour l'attribut cachetype d'une action
- Son principe et d'afficher le contenu du cache s'il est demandé d'afficher une action sans qu'il ne soit passé aucun paramètre GET ou POST (donc un runtimeData empty)
- Typiquement cela se produit lorsqu'un réaffichage de la channel est demandé lors d'un change ment d'onglet ou que c'est une autre channel située sur le même onglet qui est en train d'être sollicitée par l'utilisateur.

Le cache de type channel

- Si cachetype=channel pour une action alors le contenu de l'action est caché pour toute la channel c'est à dire quel que soit l'utilisateur.
- Typiquement, on va l'utiliser dans le cas de pages statiques (d'information) qui sont communes à tout le monde.
- Ne pas utiliser ce cache dès lors que la page contient des infos propres à l'utilisateur (utilisation de baseActionURL dans des liens notamment)
- Il n'y a pas de notion de durée pour ce cache, il est donc infini

Le cache de type instance

- Si **cachetype=instance** pour une action alors le contenu de l'action est caché pour l'instance de la channel
- Il n'y a pas de notion de durée pour ce cache, il est donc égal à la durée de l'instance de la channel

Implémenter son propre type de cache

- C'est faisable : pour cela, laissez cachetype à default et surchargez les méthodes generateKey et isCacheValid de la subchannel.

Cas des servant

- Dans le cas des servants, si et seulement si ceux-ci sont du type ICacheable, l'appel des méthodes generateKey et isCacheValid leur est délégué.

Gestion du groupmapping

- Le framework permet le mapping entre le nom local d'un groupe et son identifiant dans la base de données uPortal.
- Ce mapping se définit de la façon suivante dans le fichier de configuration :

```
<groupmapping>
        <group localname="group1" uportalid="local.101"/>
        <group localname="group2" uportalid="local.102"/>
</groupmapping>
```

- Il est alors possible de récupérer l'identifiant du groupe via une appel du type :

```
Config.getInstance().getGroupmapping().get("group1");
```

Gestion des erreurs

- La classe FrameWorkException définit les exceptions propres au framework
- Elle permet de gérer un affichage propre des erreurs pouvant survenir lors de l'exécution d'une channel, comme une erreur SQL. Pour cela, elle s'appuie sur le plugin message pour afficher le message d'erreur spécifié dans le fichier de configuration.
- Il appartient au développeur de les lever dans les catchs des blocs susceptibles de générer des erreurs
- Les FrameWorkException sont propagées par les méthodes init et setXML et sont capturées par la mainChannel
- Elles peuvent également envoyer un mail décrivant le problème à la personne étant identifiée comme responsable technique

Constructeurs disponibles

- Un constructeur par défaut qui ne gère que le message d'erreur en le fixant avec "Une erreur est survenue dans l'exécution du canal. Veuillez réessayer ultérieurement." : **FrameWorkException();**
- Un constructeur, identique au précédent à ceci près qu'il permet de spécifier son propre message d'erreur : **FrameWorkException(<message pour l'utilisateur>);**

```
throw new FrameWorkException(Config.getInstance().getDisplayError());
```

* Un constructeur qui permet de spécifier un message et d'envoyer un mail au responsable technique : **FrameWorkException(<fichier de config contenant les informations relatives aux frameworkException>,<message d'erreur à indiquer dans le mail>);**

```
throw new FrameWorkException(Config.getInstance(),"CNabucoAdmin::CData::getXmlPers(String,String) :
Erreur SQL " + e,
MainChannel.stack2html(e));
```

* Un constructeur qui permet en plus de spécifier l'état de la pile d'exécution au moment où l'exception est levée : **FrameWorkException(<fichier de config >,<message d'erreur indiqué dans le mail>, <état de la pile>);**

```
throw new FrameWorkException(Config.getInstance(),"CNabucoAdmin::CData::getXmlPers(String,String) :
Erreur SQL " + e,
MainChannel.stack2html(e));
```

Fichier de configuration

- Pour pouvoir utiliser l'option d'envoi de mail, il convient d'indiquer le serveur smtp, le destinataire, l'expéditeur, le sujet et un message d'introduction (par exemple,"Error in : ")
- Les informations relatives au mail d'erreur et au message affiché pour l'utilisateur sont facultatives
- Dans le cas où on ne les spécifie pas, il convient d'utiliser le constructeur par défaut.

Gestion des servants

- Le framework posséde un mécanisme pour l'instanciation et la gestion des canaux servants
- Pour pouvoir utiliser un servant, il convient de définir dans le fichier de configuration une action de type "servant". Une action de ce type doit définir quatre attributs :
    - name, nom de l'action
    - type, "servant" donc

    - classname, la classe principale du servant
    - servantfinish, l'action qui doit être appelée par la mainChannel lorsque le servant rend la main

Exemple :

```
<action name="servantAnnuaire"
        type="servant"
        classname="org.esupportail.portal.channels.CAnnuaire.CAnnuaireServant"
        servantfinish="rechParPers" />
```

- Le mécanisme d'instanciation fixe les staticdata et les runtimedata du servant avec celles de la mainChannel. Il peut arriver que l'on veuille fixer des staticdata spécifiques au moment de l'instanciation du servant pour par exemple, dans le cas du servant annuaire, fixer l'annuaire à utiliser ou encore fixer le nombre de résultats que l'on attend en retour. Pour ce faire, il faut surcharger la méthode initServant() dans la classe principale de la channel "maître". Exemple :

```
public void initServant(IServant servant) throws PortalException {
     staticData.setParameter("serverView", Config.getInstance().getAnnuaire());
     if (this.currentAction.getName().equals("servantAnnuaire1")) {
             staticData.setParameter("returnServant","single");
     }
     else if (this.currentAction.getName().equals("servantAnnuaire2"))
              staticData.setParameter("returnServant","multiple");
     super.initServant(servant);
}
```

- A partir du moment où l'action "servant " est appelée, l'ensemble des runtimedata sont passées au servant tant que la méthode isFinished() du servant ne renvoie pas "true"
- A la fin de l'exécution du servant, ses résultats sont stockées dans un tableau d'objet, variable d'instance de l'objet mainChannel. On les récupère grâce à la méthode getServantResults() de mainChannel. Les résultats restent disponibles tant que la méthode clearServantResults() n'est pas appellée

Services utiles fournis par le framework

ConfigChannel

- La méthode getPackageName renvoie le nom du package de la classe de configuration sous forme de String
- La méthode getConnexionDefault renvoie un object Query (voir couche d'accès aux données) sur le pool de connexion par défaut de la channel

MainChannel

- La méthode redirect permet, depuis une des méthode init, setXML ou setOutput d'une SubChannel, de se rediriger vers une autre action que l'action en cours. Dans ce cas, ne pas oublier de renvoyer Boolean.FALSE comme résultat de la méthode en cours (voir exemple)
- La méthode setConfigActions prend en paramètre une instance de fichier de configuration de type IConfigActions et remplit la liste des actions de la mainChannel
- Les méthodes getCurrentAction et getPreviousAction renvoient les actions en cours et précédentes (peuvent être NULL)
- La méthode getServantResults retourne la liste des objets résultats lorsqu'un servant est finished.
- La méthode logParams liste dans le fichier de log uPortal la liste des paramètres GET ou POST reçus par la MainChannel
- La méthode logConfigActions liste dans le fichier de log uPortal la liste des actions de la MainChannel
- La méthode getPrefForm renvoie une chaine unique pour l'instance de la channel qui peut servir à identifier de manière unique des objects javascipt (voir services de la SubChannel)
- La méthode clearServantResults efface le résultat de l'exécution d'un servant
- La méthode statique stack2string permet de renvoyer la pile d'exception sous forme de String.
- La méthode statique stack2html permet de renvoyer la pile d'exception sous forme de flux html (pour les mails d'erreur).
- La méthode log permet de d'enregistrer des informations dans uportal.log tout en profitant du paramétrage du niveau de log de la balise <actions> - **Cette methode ne doit plus être utilisée**

SubChannel

- La méthode clearChannel est la méthode qui est appelée lorsqu'une action a déjà été instanciée et qu'elle est réutilisée. Par défaut elle efface la liste des paramètres XSL. Il est possible de la surcharger.
- La méthode getXML est utilisée dans renderXML pour retourner la chaine XML qui a été fixée par setXML. Elle a la particularité d'envelopper le XML si cela n'a pas été fait par le développeur. Par exemple, si dans setXML on a fait :

```
xml = "<MESSAGE>Bonjour !</MESSAGE>";
```

alors getXML renverra

```
<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>
<xml>
<MESSAGE>Bonjour !</MESSAGE>
</xml>
```

- La méthode addDefaultXslParameters ajoute les paramètres XSL suivants à la liste des paramètres de l'action. Elle est appelée systématiquement dans renderXML donc vous en bénéficiez automatiquement :
    - baseActionURL qui permet de construire un lien http vers la channel elle-même
    - prefForm qui renvoie une chaine d'identification unique pour l'instance de la channel et peut donc servir à identifier de manière unique un formulaire javascript
    - mediaPath qui donne le chemin vers les médias de la channel (images par exemple)
- La méthode setDefaultXslParameters remplace les paramètres XSL courants par les paramètres par défaut.
- Les méthodes generateKey et isCacheValid gèrent le cache au niveau de la SubChannel. Il est possible de les surcharger.
- La méthode getMainChannel renvoie l'instance en cours de la MainChannel
- La méthode log permet de d'enregistrer des informations dans uportal.log tout en profitant du paramétrage du niveau de log de la balise <action> - **Cette methode ne doit plus être utilisée**

Autres classes du framework

Action

- Cette classe sert à représenter des actions telles que définies dans le fichier de configuration

ServantFactory

- Cette classe sert à instancier dynamiquement des servants pour les actions que l'on aura défini du type servant

SubChannelFactory

- Cette classe sert à instancier dynamiquement les SubChannel correspondant aux actions.
- Elle permet aussi d'invoquer dynamiquement les méthodes init, setXML ou setOutput (quel que soit le nom qu'on leur ai donné dans le fichier de configuration).

Exemple de mise en oeuvre

Présentation

Soit une channel MonDossierWeb qui va servir de maquette à un service d'affichage du dossier étudiant.

Dans un premier temps cette channel va proposer une page de liens vers l'affichage de l'état-civil, des inscriptions, des notes et du calendrier. On implémentera pas vraiment cet affichage, on utilisera juste le plugin Todo pour afficher une page "travaux en cours".

Recensement des actions

- L'action default (obligatoire) sera celle qui présente la liste des liens
- On aura une action par type d'affichage mais qui pointeront toutes vers une même classe et une même méthode d'initialisation qui elle utilisera le plugin todo
- la page principale a un cache du type channel c'est à dire qu'elle est cachée pour la channel (quel que soit l'utilisateur)

```
<actions>
   <action name="default"
   classname="RenderStatic"
   cachetype="channel"/>
   <action name="etat_civil"
   classname="RenderStatic"
   init="temp"/>
   <action name="inscriptions"
   classname="RenderStatic"
   init="temp"/>
   <action name="calendrier"
   classname="RenderStatic"
   init="temp"/>
   <action name="notes"
   classname="RenderStatic"
   init="temp"/>
   </actions>
```

Lecture du fichier de configuration

- On créé une classe Config qui étend la classe ConfigChannel

```
public class Config extends ConfigChannel
```

- On implémente le mécanisme de singleton //déclaration des variables de la classe

```
private static Config singleton = null;
        static {
                singleton = new Config();
        }

 /**
   *
   * @return ConfigChannel Instance
   * @throws IOException
   * @throws JspException
   */
        public static Config getInstance() {
                return singleton;
        }
```

* On implémente la méthode getConfigFile qui indique où se trouve le fichier de configuration

```
protected String getConfigFile() {
   return "/properties/channels/org_esup/CMonDossierWeb/CMonDossierWeb.xml";
   }
```

La mainchannel : CMonDossierWeb

- Cette classe étend MainChannelpublic class CMonDossierWeb extends MainChannel* Création du constructeur qui charge la liste des actions avec la méthode setConfigActions et qui enregistre le plugin Todo à cette liste.

```
public CMonDossierWeb() {
   setConfigActions(Config.getInstance());
   Todo.register(getConfigAction().getActions());
   }
```

- Les méthodes ChannelRuntimeProperties, receiveEvent et setStaticData ont un comportement par défaut dans la classe mainChannel

```
public ChannelRuntimeProperties getRuntimeProperties() {
   return new ChannelRuntimeProperties();
   }
```

```
public void receiveEvent(PortalEvent ev) {
   }
```

```
public void setStaticData(ChannelStaticData sd) throws PortalException {
   staticData = sd;
   }
```

- Comme notre channel n'a pas de besoin particulier, inutile d'implémenter setRuntimeData et renderXML, l'implémentation de base nous suffira.

La subchannel : RenderStatic

- Elle étend la classe SubChannel qui propose un comportement par défautpublic class RenderStatic extends SubChannel* Pour les besoins de cette channel (besoins très limités 😉) on a juste besoin d'implémenter la méthode temp (référencée pour les différentes actions dans le fichier de configuration) :

```
public Boolean temp(ChannelRuntimeData rd) throws PortalException, FrameWorkException
{
   Todo.todo(getMainChannel(), rd, "default");
   return Boolean.FALSE;
   }
```

* Remarquez l'appel du plugin todo qui prend en paramètres l'instance de la channel principale, les runtimeData et l'action vers laquelle on est redirigé lorsque l'on clique sur OK dans ce plugin.

- Remarquez aussi que la méthode renvoie Boolean.FALSE car l'affichage est maintenant déporté sur le plugin et donc l'action en cours n'a pas à poursuivre son cycle de vie.

Plugins livrés avec le MAG

- Todo
- Message
- Confirm
- RestrictedAction