

DOSSIER D'ANALYSE

1. DocumentServlet

1.1. Rôle de la classe

Cette classe vise à récupérer les paramètres **doc** (adresse du document, type du document, autres tels que les paramètres du stabylo) de la requête effectuée par le client web (via la servlet *PorphyryServlet*), les traiter et les stocker dans des structures exploitables, faire appel aux fonctions permettant la génération de la ou des page HTML demandée, puis l'afficher dans le navigateur.

1.2. Principales méthodes

Cette classe est une *servlet*, c'est-à-dire que le serveur web communique avec elle via une interface basée sur trois principales méthodes :

- `init()`
- `service()`
- `destroy()`

La classe ***HttpServlet***, dont hérite *DocumentServlet*, fournit une implémentation de la méthode `service()` qui dispatche les commandes http (`GET`, `POST`, etc) vers une ou plusieurs méthodes spécifiques, dont `doGet()`. C'est pourquoi la méthode `service()` n'est pas implémentée dans notre classe.

- `void init(ServletConfig config)`

Paramètre d'appel :

- `config` : objet de configuration utilisé par le moteur de servlets - ici, *Tomcat* - pour passer des informations à la servlet durant l'initialisation.

Cette méthode est lancée automatiquement la première fois que la servlet est invoquée. Celle-ci appelle à son tour la méthode de connexion au serveur de graphes. Pour cette servlet, il n'y a pas de traitements spécifiques à réaliser lors de l'initialisation.

- `void doGet(HttpServletRequest req, HttpServletResponse res)`

Paramètres d'appel :

- `req` : objet créé par le conteneur de servlet et comprenant, entre autres, les paramètres inclus dans l'URL d'appel de la servlet.
- `res` : objet de réponse, qui peut être par exemple soit une URL, soit du texte – page HTML ou tout simplement `String`.

Cette méthode est lancée à chaque nouvelle requête.

Pour récupérer la liste des valeurs passées dans l'URL correspondant à un paramètre donné, on utilise la méthode `getParameter(String name)`. S'il y a plusieurs valeurs, celles-ci sont stockées les unes à la suite des autres dans une seule et unique chaîne de caractères, et séparées par un espace. Il s'agit donc ensuite de les séparer et de les traiter.

En fonction de la valeur des paramètres qui me sont passés, on peut déterminer si le fragment à construire et à afficher est de type texte ou image. On fait alors appel aux classes 'Fragment' et ses classes héritières pour traiter le document.

Pour construire la réponse au client, on utilise, si c'est une URL, la méthode `sendRedirect(String location)`; si c'est une page HTML ou du texte, on utilise `getOutputStream()`, qui renvoie un objet de type `ServletOutputStream` dans lequel on peut "écrire".

2. Fragment

2.1. Rôle de la classe

Cette classe abstraite, permet de manipuler les fragments. Elle détermine les méthodes qui seront communes à toutes les types de fragments, en l'occurrence celle qui permet de récupérer le contenu du fragment formaté : 'contentRecovering'. Ainsi que l'attribut commun à tous les types : l'adresse (stockée dans une string) .

2.2. Principales méthodes

- `abstract String contentRecovering()`

Cette méthode (ici abstraite) remplit la fonctionnalité principale de la classe fragment, c'est à dire récupérer et formater la partie du document nécessaire au fragment. Elle y insère aussi le stabylo lorsque celui-ci en contient un.

3. ImageFragment

3.1. Rôle de la classe

Cette classe qui hérite de la 'Fragment' permet de manipuler les fragments de type Image. Les images manipuler doivent pour l'instant être au format Jpeg.

3.2. Principales méthodes

- `String contentRecovering()`

Cette méthode remplit la fonctionnalité principale de la classe fragment, c'est à dire récupérer et formater la partie du document nécessaire au fragment. Elle y insère aussi le stabylo lorsque celui-ci en contient un.

Pour cela, elle utilise le package '`com.sun.image.codec.jpeg`'. Ce package n'a pas encore été standardisé dans le JDK. Mais cela devrait se faire sous. Des problèmes d'inclusions risquent donc de se présenter avec les prochaines versions du JDK. Il faudra donc veiller soit à le rajouter, soit à faire les modifications nécessaires dans le code afin de prendre en compte le nouveau package.

Une des fonctionnalités de la classe n'est que partiellement réalisée. C'est le redimensionnement d'image. En effet, la méthode proposée dans la classe 'BufferedImage' du JDK, n'ayant pu être utilisée, celui-ci est réalisé par un algorithme que j'ai écrit, qui ne prend pas en compte les spécificités des images de type : textes scannés.

Cette classe retourne une string contenant le chemin de l'image correspondant au fragment demandé.

4. TextFragment

4.1. Rôle de la classe

Cette classe qui hérite de la 'Fragment' permet de manipuler les fragments de type texte. Les textes sont stockés sous forme de textes bruts en unicode (UTF-16). Elle les convertit en Xml puis les transforme en Html lors de la récupération du contenu.

4.2. Principales méthodes

- `String contentRecovering()`

Cette méthode remplit la fonctionnalité principale de la classe `fragment`, c'est à dire récupérer et formater la partie du document nécessaire au fragment. Elle y insère aussi le stabylo lorsque celui-ci en contient un.

La principale particularité du traitement à réaliser pour les textes se situe au niveau du formatage des données (la lecture et le découpage de texte est un traitement trivial). Pour la réalisation de cela, le package `org.apache.xalan.xslt` est le plus approprié. On l'utilise une première fois pour construire le fichier XML contenant le document. Puis une seconde fois pour transformer ce fichier en document HTML (paramétré par un fichier XSL).

Cette classe retourne une string contenant le document HTML correspondant au fragment demandé.

5. ImageStabylo et RectangleStabylo

5.1. Rôle des classes

Cette classe permet de manipuler chaque type de stabylo pouvant s'appliquer à une image. Une seul de type de stabylo a été implémenté. En l'occurrence 'RectangleStabylo' qui est comme son nom l'indique un stabylo rectangulaire.

5.2. Principales méthodes

- `void DrawOnRaster(WritableRaster theRaster) ()`

Cette méthode remplit la fonctionnalité principale de la classe 'RectangleStabylo', c'est à dire insérer le stabylo dans le 'Raster' de l'image. Pour cela la méthode est très simple. On va tout simplement modifier la valeur des points du stabylo. La stabylo a une épaisseur paramétrable (même si je ne l'ai pas utilisé), et l'épaisseur est écrite à l'intérieur du fragment afin d'éviter les dépassements.