LOGICS SOFTWARE GMBH

Ajax WebSpool Filter

Entwickler Informationen

25.02.2009

Einleitung

Filter sind Java Objekte (FilterInputStreams), die im Spoolsystem von LogAjax Dateien aus dem Verzeichnis receive, nach erforderlicher Bearbeitung, im Verzeichnis „in“ ablegen. Sie sind dazu gedacht die Eingabedaten in gewünschter Weise zu transformieren, z.B. zur Internationalisierung von Hostdaten.
Filter können verkettet sein, sodass die Ausgabe eines Filters die Eingabe in einen anderen Filter darstellt. Filter operieren generell auf einem Inputstream.
Es existiert ein default-Filter, der nur vom Verzeichnis receive nach in kopiert und immer der letzte Filter einer Filterkette ist.

Filter sind generell mit Dateinamen bzw. Dateiinhalten assoziiert. Welche Filterkonfiguration auf welche Dateien anzuwenden ist wird in einer Filterbeschreibung festgelegt.

Für eine bestimmte Aufgabenstellung ist also mindestens eine Java Klasse, die den Filter implementiert und eine Konfiguration, die festlegt wie und auf welche Dateien dieser Filter anzuwenden ist, erforderlich.

Filterbeschreibung

Die Beschreibungsdatei filter.xml legt fest für welche Dateitypen welche Filter zu verwenden sind. Die Standardkonfiguration ist in config/_spool hinterlegt.
Kundenspezifische Anpassungen werden analog zu anderen kundenspezifischen Erweiterungen an einer Kopie von filter.xml vorgenommen. Die kundenspezifische filter.xml wird im **custom/_spool** Verzeichnis erwartet. Der filter.xml Datei ist eine spool-filter_1.0.dtd Definitionsdatei zugeordnet. Diese DTD wird verwendet um die Gültigkeit der Konfigurationsdatei zu prüfen. Diese DTD-Datei ist ebenfalls nach custom/_spool zu kopieren.

Der WebSpool liest beim Start der Webapplikation diese Beschreibungen ein. Eine Aktualisierung der Beschreibungen wird erst bei Neustart der Webapplikation wirksam.

Die Parametrierung der Filter ist individuell (z.B. NLP, PRN-Dateinamen, fdf-Formulare). Parameter werden den Filterobjekten als Elemente eines Properties Objekt übergeben und sind frei definierbar, jedoch müssen sie den Konventionen von XML und Properties Objekten entsprechen.

WebSpool ermittelt den passenden Filter zu einer Datei, indem, entsprechend der Reihenfolge der Einträge in der Konfigurationsdatei, geprüft wird ob die Bedingungen für diese Datei zutreffend sind. Es wird derjenige Filter verwendet, für den alle Bedingungen erstmalig zutreffen. Bei Mehrdeutigkeit ist also auf die Reihenfolge der Deklarationen zu achten.

Die filter.xml Datei besteht im wesentlich aus einer Liste von „doc-class“ Beschreibungen. Eine doc-class ist mit einer Beschreibung für einen Dateityp vergleichbar.
Jede doc-class enthält 2 Abschnitte:

- *condition*: Regeln, die festlegen, unter welchen Bedingungen die konfigurierten Filter anzuwenden sind.

- *filter-chain*: Liste von Filter Objekten die auf den Dateiinhalt anzuwenden sind.

Detaillierte Beschreibung der Abschnitte:

- *condition*[1]: Prüfungen, die entscheiden ob ein Dokument zu einer doc-class gehört. Bedingungen können mit den booleschen Operationen and, or, not verknüpft werden. Aufgrund der Struktur von XML-Dateien werden die booleschen Operationen in Infix-Notation angegeben.
  Die Bedingungen können sich sowohl auf den Dateinamen als auch auf den Dateiinhalt beziehen. Falls nur eine Liste von Prüfungen spezifiziert wurde, verhält sich *Condition* wie ein and Operator.
    - *filename* prüft den Dateinamen gegen einen regulären Ausdruck[2].
      - *regexp* Führt die Prüfung des Datenames gegen einen regulären Ausdruck. Details zur Bildung von regulären Ausdrücken können bei der Beschreibung zur Klasse 'Pattern' in der Java API Dokumentation nachgelesen werden.
    - *contains* prüft den Dateiinhalt. Für die Prüfung werden die ersten 1Kb der Datei untersucht. Es sind 3 Ausprägungen dieser Bedingung implementiert, die sich durch Attributnamen unterscheiden:
      - *text* Vergleicht den angegebenen Text mit dem Dateiinhalt. Optional kann der *offset* in der Datei angegeben werden, ab dem der Vergleich beginnen soll. Die für den Vergleich verwendete Zeichensatzcodierung ist ISO-8859-1, falls keine explizite Codierung (Attribut *encoding*) angegeben wurde. Sonderzeichen im Text können mit \u*nnnn* codiert werden.
      - *binary* Führt byteweise Prüfung gegen Dateiinhalte durch. Die Binärewerte werden als codierter Text im Attribut binary angegeben. Optional kann der *offset* in der Datei festgelegt werden ab der der binäre Vergleich beginnen soll. Das Attribut radix[3] gibt die Basis der Codierung an ; zulässiger Wertebereich (2-36). Die codierten Bytes müssen durch Leerzeichen voneinander getrennt sein.
      - *regexp* Führt die Prüfung des Dateiinhalts gegen einen regulären Ausdruck (vgl. filename) aus. Optional kann der *offset* in der Datei festgelegt werden ab dem der Vergleich beginnen soll. Zusätzlich dazu kann die Länge (*len*) des Abschnitts festgelegt werden, der für die Prüfung relevant sein soll. Der Wert für *len* darf max. 1024 sein und wird auf das zulässige Maximum reduziert, falls der Wertebereich überschritten wird. Die für den Vergleich verwendete Zeichensatzcodierung ist ISO-8859-1, falls keine explizite Codierung (Attribut *encoding*) angegeben wurde. Dazu wird der Abschnitt aus der Datei binär gelesen. Zusammen mit der spezifizierten Codierung wird dafür ein String-Objekt erzeugt auf das der reguläre Ausdruck angewendet wird.
- *filter-chain*: Gibt eine Liste von Filterelementen an, die auf ein Dokument angewendet werden, falls der Abschnitt *condition* den Wahrheitswert true ergibt. Die Filter werden in der Reihenfolge ihres Auftretens in der Datei abgearbeitet.
  Ein Filter hat einen Namen und eine zugehörigen Java-Klasse, die diesen Filter implementiert.

[1] Die Syntax ist an Ant angelehnt vgl. http://ant.apache.org
[2] Reguläre Ausdrücke werden gem. Perl-Schreibweise formuliert. Details dazu können in der Java API Dokumentation unter java.util.regex.Pattern nachgelesen werden.
[3] Vgl. JavaDoc Integer.parseInt()

Optional kann einem Filter eine Liste von Parametern übergeben werden. Die Parameterliste wird dem Filter zum Zeitpunkt der Instantiierung bereit gestellt.

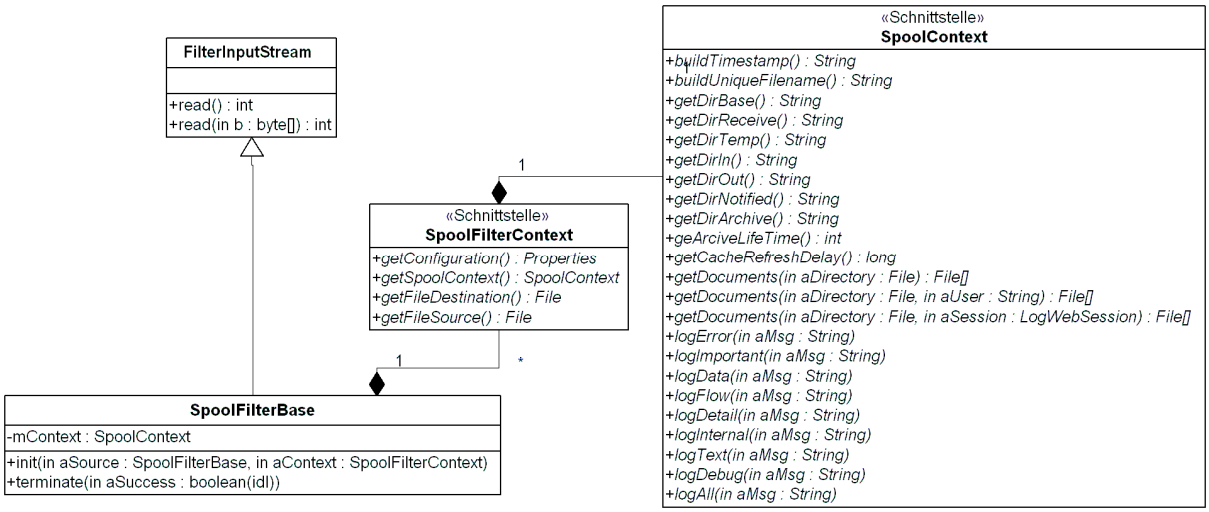
Beispiel filter.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE spool-filter SYSTEM "spool-filter_1_0.dtd">
<spool-filter version="1.0">
  <description> Default spool filter for LogAjax</description>
  <doc-class name="lwf to html">
   <condition>
     <and>
      <filename regexp=".*\.lwf"/>
      <!-- file must contain the token <html> in the first 1k -->
      <!-- use \x3C \x3E as surrogates for <> -->
      <contains regexp="(?i)(\s*\x3C!DOCTYPE\s*HTML)?.*\x3Chtml\x3E"
                encoding="ISO-8859-1"/>
    </and>
   </condition>
   <filter-chain>
    <filter name="rename"
            class="de.logics.logwebAppV3.spool.in.filter.Rename">
     <param name="extension" value="html"/>
    </filter>
   </filter-chain>
  </doc-class>

  <doc-class name="lwf to pdf">
   <condition>
     <filename regexp=".*\.lwf"/>
     <contains text="%PDF-"/>
   </condition>
   <filter-chain>
    <filter name="rename"
            class="de.logics.logwebAppV3.spool.in.filter.Rename">
     <param name="extension" value="pdf"/>
    </filter>
   </filter-chain>
  </doc-class>
</spool-filter>
```

Filter API

Ein Filter wird im Wesentlichen als FilterInputStream realisiert, von dem die Daten für die Weiterverarbeitung gelesen werden.



Jede Filterklasse wird von der Klasse SpoolFilterBase abgeleitet, welche wiederum von der Klasse FilterInputStream abgeleitet ist. Die Klasse SpoolFilterBase stellt neben der Membervariablen *in* von FileInputStream noch ein Objekt *mContext* vom Typ SpoolFilterContext zur Verfügung, mit dessen Hilfe die filterspezifischen Parameter (getConfiguration) und zusätzliche Informationen wie z.B. relevante Verzeichnisse abgefragt werden können. Darüberhinaus bietet das Objekt SpoolContextContext zahlreiche Logging Möglichkeiten für Fortschritts- und Fehlerprotokollierung.

Details zu den Klassen und Methoden können in der zugehörigen JavaDoc nachgelesen werden.

Falls temporäre Dateien erzeugt werden müssen, sollten die Dateinamen mit den Methode von *buildTimestamp* bzw. *buildUniqueFilename* gebildet werden um sicher zu stellen, dass es keine Namenskonflikt gibt. Temporäre Dateien sind prinzipiell im Verzeichnis von *getDirTemp()* abzulegen. Speziell für exklusiven Dateizugriff sollten die die Methoden der Klasse FileLock aus dem Package de.logics.logwebAppV3.spool verwendet werden (siehe zugehörig JavaDoc).

Ablauf der Filteroperation

Zu Beginn wird die *init*-Methode für jeden Filter aufgerufen, bei der die Quelle (ein SpoolFilter) und der FilterContext übergeben wird. Diese Werte werden in den protected Variablen 'in' und 'mContext' für den weiteren Zugriff bereit gehalten. Jede Instanz, die die *init* Methode überschreibt sollte zunächst super aufrufen damit die Variablen *in* und *mContext* richtig belegt sind.

Anschliessend eine der read-Methoden solange aufgerufen, bis eine End-Of-File Kennung gelesen wurde. Eine Implementierung muss lediglich die Methode *int read(byte[], int, int)* ableiten. In der Regel werden die Daten von *super.read()* gelesen und nach entsprechender Bearbeitung an den Aufrufer weiter gegeben.

Nach Beendigung der Filteroperation wird für jeden Filter die Methode terminate aufgerufen um jedem Filter die Gelegenheit zu geben Abschlussoperationen (löschen temporärer Daten) durchzuführen.

Für jede zu transformierende Datei werden neue Instanzen der Filter erzeugt.

Die Filterklassen werden den Konventionen des Applikationsservers entweder in WEB-INF/classes oder falls die Filter als JAR vorliegen in WEB-INF/lib abgelegt.

Characterset

Alle Dokumente werden generell als Binärdaten behandelt. Zeichenumsetzungen dürfen nur von den dafür vorgesehenen Filterklassen vorgenommen werden.