WritingPipelineComponents

Writing Pipeline Components

This page describes common information needed to write generators, transformers and serializers.

Using the sitemap, you describe how pipelines should be assembled. The resulting pipeline always has the following structure: one generator, zero or more transformers, and one serializer. The only exception to this rule are readers, which implicitly combine the generation and serialization step and thus also prevent you from declaring transformation steps (within that pipeline). The generator could also be an aggregator, which is just a special type of generator.

These components (generator, transformer(s) and serializer) are put together in what is usually called a "SAX pipeline". In this pipeline, SAX events are propagated. SAX-events are events corresponding to what appears in an XML document: "start element", "characters" and "end element" being the most important. A generator will start generating SAX events. The transformer that follows the generator will receive these events and can react on them, producing new SAX-events itself. The SAX-events generated by the transformer will then be received by the next transformer, which will then again start generating new events, and so on. At the end, the SAX-events are consumed by the serializer which will convert the SAX-events to some binary stream. This could be an XML document, but also a HTML document, an image, or a PDF file. The following picture shows this concept.

Pipeline

While Cocoon sets up the pipeline, it will assign a "consumer" to each generator and transformer. The consumer is where the SAX-events should be send to, thus the next component in the pipeline. The consumer of the generator will be the first transformer, the consumer of the first transformer will be the second transformer, and so on. Finally the consumer of the last transformer will be the serializer. The serializer itself does not have a consumer, but an output stream to which it can write its bytes. All the consumers must implement the SAX "ContentHandler" interface. This interface contains methods for each of the SAX-events. The SAX-events are then propagated in the pipeline by calling ContentHandler-methods on the consumer (= the next component in the pipeline).

To give an idea of the different SAX-events, the ContentHandler interface is displayed here:

```
package org.xml.sax;

public interface ContentHandler
{
  public void setDocumentLocator (Locator locator);
  public void startDocument () throws SAXException;
  public void endDocument() throws SAXException;
  public void startPrefixMapping (String prefix, String uri) throws SAXException;
  public void endPrefixMapping (String prefix) throws SAXException;
  public void startElement (String namespaceURI, String localName,
      String qName, Attributes atts) throws SAXException;
  public void endElement (String namespaceURI, String localName,
      String qName) throws SAXException;
  public void characters (char ch[], int start, int length) throws SAXException;
  public void ignorableWhitespace (char ch[], int start, int length) throws SAXException;
  public void processingInstruction (String target, String data) throws SAXException;
  public void skippedEntity (String name) throws SAXException;
}
```

Note that there is also another SAX event interface, the LexicalHandler. That interface is used to pass on less important information about XML, such as comments and CDATA sections. It is however not used very often.

More details

- **WritingForCacheEfficiency**
- **ImplementingTransformers**
- **WritingGenerators**
- **WritingSerializers**
- **WritingReaders**