Digester WhyUseDigester

Why use Digester?

Digester is a layer on top of the SAX xml parser API to make it easier to process xml input. In particular, digester makes it easy to create and initialise a tree of objects based on an xml input file.

The most common use for Digester is to process xml-format configuration files, building a tree of objects based on that information.

Note that digester can create and initialise true objects, ie things that relate to the business goals of the application and have real behaviours. Many other tools have a different goal: to build a model of the data in the input XML document, like a W3C DOM does but a little more friendly.

Alternatives

This section lists some of the alternatives to Digester, and tries to describe what tool is best under various circumstances. This is of course a controversial topic, and different people will have different opinions on this.

This page is intended to be a fair and reasonable evaluation of the alternatives, not a sales job for the Digester package. Different tools are appropriate for different applications.

Note that these comparisons should be treated with caution; the products below are not all aimed at solving the exact same problem so direct comparisons are awkward. The intent is intended to give you a feel for which tools might be best for solving the problem you are currently working on.

Digester

This tool offers only moderate performance. Because of its heavy use of reflection, it will never be lightning fast.

Configuration of Digester is moderately complex; the application developer needs to write rules that tell Digester how to map input xml into java objects.

And unlike many of the other tools listed here, Digester supports only one-way mapping from xml -> java objects. It does not provide mechanisms for serialising java objects to xml (though Digester's sister-project betwixt does provide this).

Digester is, however, very flexible. You can map xml into any reasonable equivalent in-memory java-object representation. Later changes to either the classes or the xml format is not a significant problem; the digester rules are simply updated to match.

Digester can handle "extensible" xml input files, which other approaches often cannot. Sometimes you wish to indicate in the input xml that a particular variant of a class be instantiated (sometimes even just indicating a java classname), and that the xml attributes or child elements can then vary. There is no way to write a schema for such xml input. Examples include Ant build files where custom tasks can be referenced, or Tomcat configuration files where custom Valve classes are specified and configured.

And unlike tools that generate classes, you can write your application's classes first, then later decide to use Digester to build them from an xml input file. The result is that your classes are real classes with real behaviours, that happen to be initialised from an xml file, rather than simple "structs" that just hold data.

In addition, Digester's core is easily extensible. Anyone with a basic familiarity with the SAX api for processing XML is able to create custom extensions to Digester in order to handle unusual mapping requirements.

Digester can also be used for purposes other than building in-memory representations of the input xml; it is quite possible to write custom extensions based on this framework to perform actions as xml input is parsed, such as performing database inserts. Digester can make SAX processing easy!

If, however, you are looking for a direct representation of the input xml document, as data rather than true objects, then digester is not for you; DOM, jDOM or other more direct binding tools will be more appropriate.

Hibernate xml mapping

The hibernate library is best known for doing OO <-> Relational database mappings. However version 3.0 and later have the ability to also map objects to and from xml.

This would appear to be quite similar to Digester/betwixt in features; the classes being mapped are real classes, and configuration information (an external xml file or annotations within the source code) define how the bean properties are to be mapped.

I haven't actually used this hibernate feature, so can't offer any more info. If you have used hibernate's xml mapping functionality then please describe your experience here.

Pre-processing tools

Tools like JAXB take an xml schema as input to a "compiler" and generate a set of java classes that will parse the xml input to generate java objects.

The resulting code is likely to be very fast. However there are a number of drawbacks:

- you need an xml schema that defines your input xml
- you need to run a "pre-processor" to generate code
- you may or may not have much choice about the names of the generated classes, or the way inter-object relationships are represented.
- the generated classes don't contain any "business logic", just plain getter and setter methods for the data

- if you modify any of the generated classes, then later need to regenerate them because the schema has changed, your changes are wiped out.

Basically, using JAXB is like having a custom DOM representation, rather than being able to initialise *your* classes from an xml input.

References

- Javolution (http://javolution.org)
- JAXB (http://java.sun.com/xml/jaxb)
- Castor (http://www.castor.org/)
- jibx (http://jibx.sourceforge.net)
- XMLBeans (http://xmlbeans.apache.org/)
- Zeus (http://zeus.objectweb.org/)
- XStream (http://xstream.codehaus.org)
- Beck (http://beck.sourceforge.net/)
- JBind (http://sourceforge.net/projects/jbind/)
- JXM (http://jxm.sourceforge.net/)
- Skaringa (http://skaringa.sourceforge.net/)
- hibernate (http://www.hibernate.org)

Articles

- http://www-106.ibm.com/developerworks/xml/library/x-databdopt/
- http://www.rpbourret.com/xml/XMLDataBinding.htm