Digester TODO

Overview

This page lists a bunch of ideas which may or may not be implemented in future releases of the Digester.

If the idea seems to have definite advantages, and seems to have a reasonably obvious implementation, then it should go into the "probable" section. If the idea is only tentative (or if it is likely to be controversial) then it should go into the "possible" section.

Version 1.7.1

Probable

InvokeMethodRule

Evaluate the InvokeMethodRule code that is currently committed to CVS on a branch. This new rule acts like CallMethodRule except that it calls the target methods as soon as all the necessary parameters are available. This is much more intuitive in a number of scenarios; the code is a fair bit more complicated than CallMethodRule however. The API for this new rule is also much cleaner than CallMethodRule, which has gathered cruft as new features have been added.

handling hyphenated properties

It would be nice for SetProperties and SetNestedProperties rules to automatically map xml attributes and element names like "foo-bar" to bean properties of form "fooBar". The hyphenated naming style is a very common xml usage, but is currently very difficult to handle within Digester.

SetPropertyRule from element body

Implement a SetPropertyRule variant that allows the data to be in the body text.

Example code onto digester website

Somehow put example code onto website using maven? We have all these nice examples now in the src/examples CVS directory; it would be great if these could be viewed via the jakarta commons maven site.

Henri Yandell is a jakarta-commons developer, and has this tool on his personal website that may be useful for this, particularly if wrapped up as a Maven plugin:
http://www.generationjava.com/projects/Java2Html.shtml

There's another tool used for the FLEX compiler project (see, for example)
http://flexc.csail.mit.edu/Harpoon/srcdoc/harpoon/IR/Quads/Quad.html which the author (http://cscott.net) is willing to contribute.

cleaner build process for examples

The current ant buildfiles for examples create the .class files in the source directory. This is useful in some ways, but ugly in others. If anyone could think of a nicer way to handle this it would be great.

cleaner logging suppression during unit tests

A number of unit tests deliberately cause digester rules to fail. Unfortunately, this then results in messages being logged which go to the screen and clutter the unit test output unless explicitly disabled. There is currently a mechanism in the ant build.xml file for suppressing these log messages during unit tests, but it isn't very clean. Any better ideas would be welcome.

Variable Expansion and Ant

Look into whether the !Substitutor/MultiVariableExpander framework satisfies Ant's requirements for xml parsing.

The Substitution framework and variable expansion facilities were added during the 1.6 release. It would be nice to confirm that anything ANT variables can do can also be handled using this framework.

Jira

Move to jira as the bug tracking repository?

Add Rule to map body text to property named in attribute

There have been a number of questions on the user list about how to parse the following style of xml:
<property name="txtNumClient">num1</property>

This seems a good candidate for a new rule. However implementation is a little tricky, because the name of the destination property is specified in a different SAX callback than the data to be assigned to that property.

Maybe the "named stacks" feature added in digester 1.6 can be used to solve this?

See: http://www.mail-archive.com/commons-user@jakarta.apache.org/msg07607.html

Provide better support for DynaBean in SetNextRule

The jakarta commons BeanUtils library provides a 'DynaBean' class, which acts rather like a java class whose properties can be altered at runtime. The reflection functionality provided by the BeanUtils library knows about DynaBeans, and essentially makes them appear to be ordinary javabean classes when performing reflection.

Most Digester rules handle DynaBeans just like ordinary bean classes, because they use the BeanUtils reflection methods.

However it has been reported that the SetNextRule doesn't handle DynaBeans well (see mail archives for 2004-11-25, subject 'DynaBean Hierarchy' http://www.mail-archive.com/commons-user@jakarta.apache.org/msg09115.html). It would be nice to fix this if an elegant solution can be found (and fix any other rules that might not work well with DynaBeans).

Add Missing XML Rules

There are some Rules that could be configured with xml-based rules, but they aren't in DigesterRuleParser or the DTD.

Possible

Have BeanUtilsBean instance per Digester

Currently, in order to change the way Digester rules perform datatype conversions, it is necessary to register custom converters with the global ConvertUtilsBean, thus affecting all code in the same webapp.

Adding a Digester.setBeanUtilsBean(beanUtilsBean) method would allow users to control this behaviour on a per-digester-instance level. The default would be to use the global beanUtilsBean if none is set.

However all Rule classes would need to be updated to get the beanUtilsBean from the digester object instead of using the static BeanUtils and ConvertUtils methods.

Refactor CallParamRule

Refactor the CallParamRule to break out the separate pieces of functionality into separate rules. Perhaps give them a consistent naming standard, like: "CallParamWithXXXRule".

AccessController and Private Methods

Provide support for running Digester under an `AccessController`, so that it can access private bean methods:
http://marc.theaimsgroup.com/?l=jakarta-commons-dev&m=106866350019676&w=2

Resolving ids to object references during parsing

Provide some mechanism for object refs to be resolved during parsing, e.g. given

```
      <doc>
         <foo id="1">...</foo>
         <bar>
           <foo-ref id="1"/>
         </bar>
      </doc>
```

it would be nice to to get a method on bar's object called with the reference to the previously-created foo as a parameter. Some email discussion has already occurred on this, and Robert Donkin suggested "named stacks" might be part of the solution. Forward references would be trickier, but Method objects might possibly be cached, and later invoked when the referenced object is actually created.

See also log4j's "Joran" module, which I believe implements this.

Inter-Rule Communication

Provide a standard mechanism for inter-rule communication. For example Plugins requires communication between PluginDeclarationRule and PluginCreateRule instances, and some significant hoops had to be jumped through to get this to work properly. Maybe the Digester object should hold a Map object, and Rule objects can place entries in this keyed by their class name (eg class org.apache.commons.digester.XXX can place entries in there with key="org.apache.commons.digester.XXX#key1"? This naming convention would ensure no collisions ever occur, even in the presence of user Rule objects.

Minml parser support

Provide compatibility with the "minml" parser, for embedded work?
http://www.wilson.co.uk/xml/minml2.htm

Implicit Actions

Joran has this "implicit action" thing which allows it to fire an action when no other rule matches. Their (sole) implicit action does this:

```
     given current tag <foo class="..">, look at top object
     on stack and see if it has an "addFoo" or "setFoo", and
     if so instantiate the specified class, push it on the
     stack and at end() time call the add/set method (ie
     do a SetNext).
```

How could we implement this in Digester? It is almost the same as setting pattern "*", because that will only match if it is the longest match. But all of the "implicit" rules would fire, whereas Joran only fires the first "applicable" implicit rule. I think this "one implicit rule only" is a little app-specific, but it works for them and the end result is quite effective.

The WithDefaultsRulesWrapper class is also similar to this. Does anyone have any examples of when WithDefaultsRulesWrapper would actually be useful?

Rule to pass xml attributes as a Map

Create a new rule which gathers all the xml attributes into a Map object and passes this as a parameter to a method. Possibly this could be an extension to the existing CallParamRule, but that rule has so many options already!

Allow Rule objects which are not intended to be nested to store state?

In general, Rule objects are not allowed to store any "state" in the members of an instance, where "state" means data that changes during parsing of a document. The reason is that when associated with patterns involving wildcards, or where a single rule instance is associated with multiple different patterns, rule objects can be invoked in a "nested" manner, with their methods being invoked in the order "begin (begin body end) body end". Clearly such an invocation pattern doesn't work if the object is storing state on itself (unless it uses stacks).

However some rules clearly can't sensibly be invoked in a nested manner, eg SetPropertyRule.

So maybe it would be sensible to allow state to be stored on some rules, provided they use an assertion or similar to prevent themselves from ever being used in a "nested" manner.

Consideration needs to be given, though, to resetting the state if a parse fails part-way through, or if a Rule instance is stored in a RuleSet and added to multiple Digester instances, etc.

Version 2.0

Probable

Split up Digester class

Currently the Digester class has 4 separate roles:

- API user interacts with to create rules (aka "rule factory methods")
- API user interacts with to configure Digester (excluding Rule creation) and initiate parse
- API the parser interacts with (SAX handler interface)
- API the Rule objects interact with (eg accessing the object stack)

Separating out the rule factory part would reduce dependencies significantly. Currently the Digester class cannot be distributed without including every Rule class that there is a factory method for. I envisage something like this:

```
    Digester digester = new Digester();
    RuleFactory factory = new RuleFactory(digester);
    factory.addObjectCreate(....); // calls digester.addRule(new ObjectCreateRule(...))
```

Separating out the SAX handler methods would greatly simplify the API that users see:

```
    Digester digester = new Digester();
    org.xml.sax.ContentHandler saxHandler = digester.getSAXHandler();
```

and the methods like characters(), startElement(), etc are only present on the handler object, not the main Digester interface.

Separating the user API from the rule API might be a little trickier, but would be beneficial too. There is no reason why a *user* of the digester should be required to wade through an API including push() and pop() methods. Yes, push() is currently used to "prime" the stack before parsing in some cases, but a method like "setInitialObject" would be a much nicer method to expose to users (it would call push internally).

Joran uses "Interpreter" --> "SaxHandler", and "ExecutionContext" --> "!Digester" sans "SaxHandler", with "ExecutionContext.getHandler" to get the saxhandler if necessary.

Define Rule Lifecycle methods

Rule objects need some more hooks. It would be nice to have methods that are guaranteed to be called at the following times:

- whenever the Rule is added to a digester (passing associated pattern)
- before parse begins
- after parse ends
- other ???

Revisit exception handling

It would be nice to define more formally what exceptions Rule objects can throw and when. The plugins module, for example, has to jump through some hoops to handle potential error cases.

Remove deprecated methods

Class DigesterRuleParser in the xmlrules package uses the deprecated begin() and end() methods (as do some other rules). Fix these. In general, get rid of use of deprecated methods.

Think about parsing multiple documents

Some people have expressed the desire to use the same Digester instance to parse multiple xml documents. This is currently (v1.6) fairly unsafe.

In the current (1.6) version, the best option is probably to create a configured Rules object and a configured !SAXParser object and keep these cached for reuse but recreate a new Digester object each time. At the least we should carefully document what objects are safe for reuse and which are not. And what the implications of using Digesters and Rules objects in multithreaded apps. I also think the APIs for preconfiguring Rules or RuleSet objects and then reusing them are rather inconsistent, and need some review/fixing. This is particularly important for the XmlRules stuff, as parsing an xmlrules config file is not a quick process.

See bugzilla #29428 for some discussion of this.

Make the XML Rules DTD more consistent

Some things are hyphenated, some are not. One tag uses attr-name while the rest use attrname. One uses prop-name and another propertyname.

Possible

Mini Digester distribution

Provide "mini-digester" distribution? This could contain:

- Digester "core" class (without factory methods)
- Only RulesBase rules engine
- No plugins or xmlrules

Factory add methods return value

Factory "add" methods return the newly created Rule object? This would allow code like this:

```
    SetNextRule r = digester.addSetNext(.....);
```

Rename classes

Here are some possible name improvements:

- rename "Rules" --> RuleEngine or RuleMatcher or RuleManager
- rename "Rule" --> "Action"?

Allow only single instance of a Rule

Forbid a single Rule instance from being added to a digester more than once? This allows a rule to store context info in its members rather than jump through hoops to store that info on various stacks. (or does it, when pattern = "*/foo"?)

CallMethodRule interleaving issue

Fix nasty problem with CallMethodRule where interleaved instances of this rule can mix up their parameter stacks.

XPath-like patterns

Provide more xpath-like patterns by default, eg:

- "/root/sub"
- "sub" should be equal to "**/sub"
- "/root//sub" --> "root/*/sub"

I'm not proposing full xpath support (though perhaps we *could* look into stealing code from jxpath or similar to do that). Or maybe we should look at STX-based projects for inspiration instead, as they are more SAX-oriented than xpath.

Attribute matching support

Allow rules to match attribute values:
"sub[type='foo']"

Or maybe just provide a "filter rule", which causes its target to be ignored under certain circumstances?

Rules improvements

Implement Rules engine which does "progressing filtering" to select its rules, rather than the current nasty hashmap-based approach?

Enhance Exceptions

Provide more declared exceptions on various methods

Have Digester guarantee to catch a certain subclass of RuntimeException (eg DigesterTunnelledException), so that we can future-proof the system against new situations where we want to throw an exception but the interface doesn't allow us.

Logging Improvements

Fix ugly logging mess. The apparent requirement is that container apps want to be able to instantiate multiple digester objects in the same classloader, but configure their logging differently. Confirm this requirement first.

Proper namespace support

Implement proper namespace support for rules. Currently xml namespace handling is a real hack.

See the ruby xmldigester (http://rubyforge.net/xmldigester) implementation for an example.

Xmlrules automatic handling of new Rule classes

Fix problem where xmlrules needs to be manually updated when new rules/features are added to the digester "core". Ideally, xmlrules would "auto-detect" rules/features at runtime. However an XDoclet approach that generated the appropriate xmlrules stuff at compile-time would still be an improvement.

Pop and Peek should throw EmptyStackException

Digester.pop() and peek() should throw EmptyStackException. Currently they return null when empty.

Up to Digester