SummerOfCode2009

Project: Convert current Tomcat valves to Servlet Filters.

Abstract: Apache Tomcat is an implementation of the Java Servlet and JavaServer Pages technologies. It is famous and widely used. It is also a component of an Application Server named JBoss Application Server. My task in this project is to convert current Tomcat valve based implementation into Servlet Filters which is consistent with the Servlet Specification.

Here is the roughly plan for this project:

Week 1: finish the very first valve - AccessLogValve, including testing and documentation work. The deliverables after this week are the source code of AccessLogFilter, including unit test code and documentation of this class.

Week 2: finish working with the ErrorReportValve and SSLValve

Week 3: finish working with StandardEngineValve, move its functionality into some "engine" level filter. And I also need to twist the "Engine", "Host", "Context" and "Wrapper" class a little to move on in this week.

Week 4: finish working with StandardHostValve, move its functionality into some "host" level filter.

Week 5: finish working with StandardContextValve, move its functionality into some "context" level filter.

Week 6: finish working with StandardWrapperValve, move its functionality into some "indivisual servlet" level filter.

Week 7 - 8: Do some integration test with current work to guarantee I do not break anything.

Week 9: finish working with RequestFilterValve and its 2 subclasses, refactor their functionality out into 3 corresponding filter classes.

Week 10 - 11: Refactor the functionality out of the AuthenticatorBase class and its subclasses into some structure which is consistent with the Servlet Container Profile in JSR-196.

Week 12: Integration test all the work. Check the missing documents and complete them.

==========

Here I'd like to give some design option and my thinkings of this project. Thank you for your comment.

1. Since valves have different levels, such as server level, engine level, host level, context level. And there are several places for the configuration: server.xml, web.xml. For server level, both server.xml and web.xml is suitable, since both of them are some kind of global. For engine and host level, I think server.xml is much more suitable. And we could also use "$CATALINA_BASE/conf/[enginename]/engine-filters.default" for engine level filters, "$CATALINA_BASE/conf/[enginename]/[hostname]/host-filters.default" for host level filters and "$CATALINA_BASE/conf/[enginename]/[hostname]/context.xml.default" for context levelfilters. This is the options for configuration file location that I could think about. In my opinion, we could put all these configuration in server.xml for simplicity, but keep an eye on those "$CATALINA_BASE/conf/[enginename]/[hostname]/" path just like tomcat deal with context.xml.default now.

markt - Are Valves going to be removed completely?

- If yes, it just becomes a one-for-one replacement (all references to valves in code and config changes to filters) once all the valves are available. In the meantime, filters can be configured in contexts (per app or globally) for testing purposes.
- If no, I'd be tempted to use web.xml to configure filters. There is already a global web.xml. Adding the option of a per engine and per host web.xml should be doable.

xxd - How about the pipeline? If we remove those valves, this class needs not to be existed. I found that this class is not widely used outside the hierarchy of container interface. And since these classes all implements Lifecycle interface, the start() and stop() method will be invoked no matter whether these classes implement the Pipeline interface or not at a glance. And the code in StandardPipeline are generally about manipulating valves, so if we could remove all those valve structures, this class could be eliminated either. Any comments?

- and if we configure filters in web.xml, how do we differ engine and host level filters? And context and wrapper (if exists) level filters of course should be configurated in the corresponding web.xml, since it goes with the servlet specification.

2. Filters has orders, so the order they appear in the configuration file should be maintained. And some filter has to be the first one in the filter chain. I could not get an idea to guarantee this elegantly other than give some comment in the configuration file or in the documentation.

markt - Filter order

- This is already defined in the servlet spec. Look to see how Tomcat handles this currently.

xxd - I know the order of context level and wrapper level filters is already defined in the servlet spec. What I mean is the filters of engine and host level.

markt - Order should be:
(1) Global filters (2) Engine filters (3) Host filters (4) Context filters Where a filter is defined at multiple levels, merge the configurations using the Servlet 3.0 rules with the *lowest* level taking precedence (ie context > host > engine > global).

3. I will modify the digestor code in order to load the configuration about filters in, ContextRuleSet.java, EngineRuleSet.java, HostRuleSet.java in particular. And those filters information will be store in an order map (I'd like to choose treeMap now) – TreeMap<FilterMetaInfo, Filter>. The FilterMetaInfo class is generally like this:

private final FilterLevels level;

private final int order;

FilterLevels is an enum which contains: ENGINE, HOST, CONTEXT.

markt - Point of execution

- This raises an interesting point. Is one single filter chain created in the same way as it is now or do we create multiple filter chains - one per engine / host / context - much like valves are now. Just because filters are defined at the engine level that doesn't necessarily mean they have to execute there
- Which raises the next question - which class loader loads the engine and host filters? Single filter chain implies that the webapp must do the loading. That could limit what the filter can do. Is that an issue?

xxd - I think all the filters need to be formed as a chain, and will be invoked when the request comes in. Those engine and host level is just ordinary filters after they are added into the filter chain. But before it, they have some global attributes compared with context level filters. So, could we just add those engine and host level filters into the filter chain when the filter chain is created? But for sake of performance, I think we could cache those high level filters since they are global to some degree.

For the discussing of configuration location of those converted filters, please refer to http://wiki.apache.org/tomcat/SummerOfCode2009/LoadingFilterConfiguration.

4. Do I need to remove all the pipeline facility or can I try to remove as much logic as possible from StandardEngineValve StandardHostValve/StandardHostValve? I found that those valve just delegate to the lower level pipeline(valves), and the lowest StandardWrapperValve will actually finish the job (filterChain and/or the servlet invocation).

markt - See comment on point 3

- Right now I don't know. I'd like to remove all the valve code if possible. It depends how much access to Tomcat internals the filters end up needing. This question probably needs to be deferred until we have some working filters to make judgements based on.

xxd - All these jobs should be finished somewhere, if we do not use valves. Can I just override the invoke() method of Container interface for StandardEngine StandardHost/StandardContext/StandardWrapper, and high level container will call the invoke() method of the container which have a lower level than itself, for example, in invoke() method of StandardEngine, the invoke() method of StandardHost will be called. And I will move all the valve invoke() logic into the corresponding invoke() method of its container.

By the way, I'd like to added all the filters configurated in the server.xml file into the filterChain when the filterChain is created(in ApplicationFilterFactory.java line 143 or so, after "StandardContext context = (StandardContext) wrapper.getParent();") in the order of be configed in server.xml or some other configuration location. The general algorithm is like this: first, get to engine and host level through context (through getParent()); then add host and engine level filters into filter chain in the order of Engine -> Host -> Context. Is that a proper place for me to add those logic? What's the opinion of yours?

markt - The long term aim is to remove Valves entirely

5. Valve interface has a event() method which is used to process a comet event. Filters need not to worry about whether it is a comet event or not. So how can we reflect this part in our new filter-based structure?

markt - implements CometFilter interface to support comet based request.

6. Do we really need to move the logic in StandardEngine StandardHost/StandardContextValve into some filters? Since the logic of these classes is purely servlet-engine concerned, I do not think those logic could be reused by filters. Any comments?

markt - Whilst removal is the aim, it may well end up making sense to keep some internal Valves.

7. My solution now to load configuration in $CATALINA_BASE/conf/server.xml is to add a CallMapParamRule.class to collect the attribute-value pairs into a map, and this map is the initiation parameter of a filter. A method with signature – "addFilterWithConfig(Class<? extends Filter> filterClass, Map<String, String> paramMap)" will be added in StandardEngine/Host/Context to add new filters into it. The main content of CallMapParamRule is as blow:

```
public void begin(String namespace, String name, Attributes attributes)
throws Exception {
for (int i = 0, count = attributes.getLength(); i < count; i++) {
// here we do not deal with the className attribute, since it
// is used as the class name that will be initiated.
if("className".equalsIgnoreCase(attributes.getLocalName(i))) {
continue;
}
paramMap.put(attributes.getLocalName(i), attributes.getValue(i));
}
digester.getLogger().debug(paramMap);
Object parameters[] = (Object[]) digester.peekParams();
parameters[paramIndex] = paramMap;
}
```

In order to add those newly added filters into the FilterChain, the ApplicationFilterFactory class also need to be altered.

Any comments about this solution? Or are there a better one?

8. I've posted the patch file of the source code for loading configuration of filters from server.xml and hacking those loaded filters into filter chain as attachment of this wiki page. Any comments are welcomed. And the source code could reduce duplicating code by altering the Container interface. But I do not know the influence of this action. So I added the corresponding code into StandardEngine, StandardHost and StandardContext respectively.

9. I've made a seperate wiki page for the configuration location of those newly added filters. Here is the link: http://wiki.apache.org/tomcat/SummerOfCode2009/LoadingFilterConfiguration

---

CategoryGSOC