Concept - Event mechanism

Your Rating: Results: 99 rates

Draft for 5.2

Discussion points on the need for a generic events mechanism after the introduction of IoC components.

After working with IoC components for a few weeks, we already see the need for a generic events mechanism in Magnolia.

One of the use cases is the cache filter and module. We already have such a mechanism there, but very ad-hoc. Since CacheFilter "needs to" (see below) keep references to things like a `Cache` and `CacheConfiguration` instances, we can't solely rely on proxies (MAGNOLIA-2553@jira, MAGNOLIA-3086@jira) to guarantee that injected components remain consistent with their configuration on the repository. In the case of the CacheFilter, the CacheModule instance, if observed/proxied, will always be "up-to-date", but the references to `Cache` and `CacheConfiguration` won't - they will be the instances reflecting the configuration at the time of injection.

In fact, the CacheFilter doesn't really "need to" keep such references, but there is no way to enforce usage of the "root" object in the code: apart from code reviews, and a deep understanding of all the underlying mechanics, nothing will prevent a developer from keeping an instance of `cacheModule.getFoo()`. So currently, there seem to be two possible solutions - perhaps not mutually exclusive:

1. use "deep proxies" - each "sub bean" of an observed component should itself be a proxy. If an element (sub bean / sub node) "disappears", we need to handle it (null, exception, ...). While this might provide some level of transparency, it's unclear whether this might impact performance in any way. It might also impact debugging negatively (i.e proxies are never easy to "read" in a debugger)
2. introduce an event mechanism. The CacheFilter would register itself (by annotation, by code, or automagically?) as a listener of "event - config change - CacheModule", and reset its variables upon such event. This is what's currently happening with the `info.magnolia.module.cache.CacheModuleLifecycleListener` interface. We could generalize this.
   Perhaps it's even possible to "hide" this into the container - pico would manage such events by calling back / re-init certain components - but that seems "dangerous", i.e hard to grasp from the outside world.

Implementation options for receiving events

Regardless of which style is used the classes interested in events should not directly register themselves in order to receive events. This should be handled by the IoC container. Classes should only need to somehow express that they are interested in a certain event.

Using and interface

Classes implement an interface and will receive all events that assignable to the type in the generic expression. That is, if you use Object you'll get all events.

```
public class FooBar implements ApplicationListener<ModuleReloadEvent> {
   public void onApplicationEvent(ModuleReloadEvent event) {
      // take appropriate action
   }
}
```

This is the approach used by Spring.http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html#context-functionality-events

Annotating method

```
@Consumes
public void whatever(ModuleReloadEvent event) {...}
```

This is the approach used by Guts, an extension for Guice.http://kenai.com/projects/guts/sources/code/content/trunk/guts-demo/guts-demo-event-bus/src/main/java/net/guts/demo/event_bus/example/basic/Consumer.java?rev=538

Annotating parameter

Simply annotating a parameter on a method as an event that it want to receive.

```
public void afterLogin(@Observes LoggedInEvent event) { ... }
```

This is the approach used in CDI.http://docs.jboss.org/cdi/api/1.0/javax/enterprise/event/package-summary.html

Implementation options for sending events

For sending events an object needs a reference to some object that performs this service. It can be either injecting a reference to the object that performs dispatching and keeps the actual lists of observing beans. As is the case with Spring. Or injecting an object that is more dedicated, as is the case with CDI and Guts for Guice.

Spring

```
private ApplicationEventMulticaster eventMulticaster;
```

Guts for Guice

```
@Inject
private Channel<ModuleReloadedEvent> eventChannel;
```

CDI

```
@Inject
private Event<ModuleReloadedEvent> moduleReloadedevent;

moduleReloadedEvent.fireEvent(new ModuleReloadedEvent());
```