Software Analytics 2016 - Assignment 1

Bugfix Report

Jesper Findahl

March 10, 2017

1 Introduction

The bug fixed is from the open-source Java framework Dropwizard[1] for building RESTful web services. Dropwizard is based on several Java libraries and in particular for this bug the Jersey[2] RESTful web services framework and the Object/Relational Mapping framework Hibernate[3] are important. The issue[4] is about a custom Dropwizard Java annotation (`@UnitOfWork`) not being recognized in all cases, specifically when specifying a Jersey sub-resource locator for HTTP requests.

The idea with the `@UnitOfWork`[5] annotation is to enable the developer to easily specify resource methods for HTTP requests that accesses a database. The name of the annotation refers to the design pattern *unit of work* that encapsulates a set of database operations into a transaction. By annotating the method with `@UnitOfWork` the framework "automatically opens a session, begins a transaction, commits the transaction, and finally closes the session. If an exception is thrown the transaction is rolled back."

2 Locating the Bug

The bug description on github describes the bug quite detailed, however, to understand the cause it was necessary to understand how the annotation works. The Jersey framework provides the necessary functionality to specify resources, sub-resources and sub-resource locators[6]. For both resources and sub-resources the `@UnitOfWork` annotation works and is also tested in the Dropwizard framework. However, for sub-resource locators this is not the case and therefore the application crashes. A sub-resource locator can be seen in listing 1, where on line 7 a new instance of another resource class is returned.

[1] https://github.com/dropwizard/dropwizard
[2] https://jersey.java.net
[3] http://hibernate.org/orm/
[4] https://github.com/dropwizard/dropwizard/issues/1910
[5] http://www.dropwizard.io/0.7.1/docs/manual/hibernate.html#transactional-resource-methods
[6] https://jersey.java.net/documentation/latest/jaxrs-resources.html

```java
@Path("/item")
public class ItemResource {
    @Context UriInfo uriInfo;

    @Path("content")
    public ItemContentResource getItemContentResource() {
        return new ItemContentResource();
    }

}

public class ItemContentResource {

    @GET
    public Response get() { ... }

    @PUT
    @Path("{version}")
    public void put(@PathParam("version") int version,
                    @Context HttpHeaders headers,
                    byte[] in) {
        ...
    }
}
```

Listing 1: Example sub-resource locator

For the `@UnitOfWork` annotation to work Dropwizard needs to know when a given resource method is interacted with. This is done by registering a custom application event listener

```java
UnitOfWorkApplicationListener extends ApplicationEventListener
```

to receive events triggered by the lifecycle of the application. When the listener receives the event `ApplicationEvent.Type.INITIALIZATION_APP_FINISHED` it will get all the resource methods and sub-resource methods that have the `@UnitOfWork` annotation and put them in a map. The listener also registers another listener that gets events regarding requests.

```java
UnitOfWorkEventListener implements RequestEventListener
```

When a resource event is received the `UnitOfWorkEventListener` makes a lookup for the given method in the map created earlier. If the method has the annotation the necessary steps are taken, otherwise it does nothing. Therfore it is crucial that all resource methods are registered properly when the application has been initialized. However, it does not register any sub-resource locator methods which is the issue raised on github. As a temporary fix the authors of Dropwizard provide a workaround solution so that these methods are registered properly.

To ensure that the bug reported was an actual bug a test, `JerseyIntegrationSubResourceTest`[7], was created which as expected failed. After creating the test and reading the comments on github it seemed reasonable to believe that the issue was with the registration of the methods.

3 Fixing the Bug - First Attempt

After deeper analysis of the Jersey API no way to access the sub-resource locator methods could be found. Also it was not possible by debugging to find any property indicating sub-resource locator methods.

Going back to the Jersey documentation this actually could have been expected as the documentation states[8]

"*Note that the runtime will not manage the life-cycle or perform any field injection onto instances returned from sub-resource locator methods. This is because the runtime does not know what the life-cycle of the instance is. If it is required that the runtime manages the sub-resources as standard resources the Class should be returned...*"

This aspect is missing in the discussion of the issue on github, where the authors instead provide another workaround by using the `UnitOfWorkAwareProxyFactory` [9].

By analyzing the bug report from one user it was possible to get a hint of another solution[10], namely instead of registering the method after the application has been initialized, the method could be registered when receiving the first request on the method, thereby invoking the `RequestEvent.Type.RESOURCE_METHOD_START` event. Instead of just looking up if the method had been registered earlier, after the application initialized event, it is also possible to do it when receiving the resource's start event on the `UnitOfWorkEventListener` (see Listing 2)

4 Fixing the Bug - Final Solution

After some even deeper analysis of the Jersey framework I found that it was indeed possible to access the sub-resource locator programatically, and therefore the previous assumption that it was not possible due to a Jersey constraint was not correct. The solution is based on the fact that it is possible to access the `ResponseType` (e.g. the return type) of any resource method. As Jersey provides a type definition for all resource methods (e.g. `SUB_RESOURCE_LOCATOR`) we can exploit this fact and find the annotations

[7] https://github.com/jeppe-style/dropwizard/commit/44fffd6147db2685efad22cc0fd2fdd22abcc620#diff-b4fb6c7cc518c0c5f122b74ff98544ccR65

[8] https://jersey.java.net/documentation/latest/jaxrs-resources.html#d0e2496

[9] https://github.com/dropwizard/dropwizard/issues/1806#issuecomment-259185620

[10] https://github.com/dropwizard/dropwizard/issues/1910

```java
@Override
public void onEvent(RequestEvent event) {

  final RequestEvent.Type eventType = event.getType();
  if (eventType == RequestEvent.Type.RESOURCE_METHOD_START) {

    // START BUGFIX
    Method method = event.getUriInfo().getMatchedResourceMethod().getInvocable().getDefinitionM

    UnitOfWork unitOfWork;
    if (methodMap.containsKey(method)) {
        unitOfWork = methodMap.get(method);
    } else {
        unitOfWork = method.getAnnotation(UnitOfWork.class);
        if (unitOfWork != null) {
            methodMap.put(method, unitOfWork);
        }
    }
    // END BUGFIX
  ...
}}
```

Listing 2: Bugfix - first attempt

of the `ResponseType` class methods that is being returned by the sub-resource locator. Although the first solution works, this solution is more elegant and also fits better with the structure of the original code (Listing 3)

5 Conclusion

For both solutions all test methods created earlier passed except one. The failing method was testing if SQL exceptions are handled properly. It failed because the method did not have the `@Produces(MediaType.APPLICATIO` annotation, which for regular resources and sub-resources are not needed, likely because Jersey adds it during runtime. I did not find a way to add annotations programatically, although probably with some more research it could be possible.

```
@Override
public void onEvent(ApplicationEvent event) {

    if (event.getType() == ApplicationEvent.Type.INITIALIZATION_APP_FINISHED) {
        for (Resource resource : event.getResourceModel().getResources()) {
            for (ResourceMethod method : resource.getAllMethods()) {
                registerUnitOfWorkAnnotations(method);
            }

            for (Resource childResource : resource.getChildResources()) {
                for (ResourceMethod method : childResource.getAllMethods()) {
                    registerUnitOfWorkAnnotations(method);
                }
               // BUGFIX
                registerResourceLocatorUnitOfWorkAnnotations(childResource.getResourceLocator()
            }
        }
    }
}

...

private void registerResourceLocatorUnitOfWorkAnnotations(ResourceMethod method) {
    if (method.getType() != ResourceMethod.JaxrsType.SUB_RESOURCE_LOCATOR)
        return;

    for (Method responseMethod : method.getInvocable().getRawResponseType().getMethods()) {

        UnitOfWork annotation = responseMethod.getAnnotation(UnitOfWork.class);

        if (annotation != null) {
            this.methodMap.put(responseMethod, annotation);
        }

    }

}
```

Listing 3: Bugfix - final solution