Apache CXF exception handler for jaxrs (REST)

[Ryan](https://www.luckyryan.com/author/ryan/)  June 15, 2013  [Apache CXF](https://www.luckyryan.com/category/tech-stuff/apache-cxf-tech-stuff/), [Tech Stuff](https://www.luckyryan.com/category/tech-stuff/)  [5 Comments](https://www.luckyryan.com/2013/06/15/apache-cxf-exception-handler-for-jaxrs-rest/#comments)

In another post ([Apache CXF with Spring Integration](http://www.luckyryan.com/2013/06/15/apache-cxf-with-spring-integration/)) I covered splitting an application into a client/service structure using [Apache CXF](http://cxf.apache.org/). In that project the business service would throw a custom exception if a certain user tried to register. The problem is that without any special handing CXF will return a 500 error and lose the custom exception. The client will see the 500 response code and throw it’s own exception which is ambiguous. The result is broken logic in the Spring MVC controller which expected the custom exception. To solve this and restore the original functionality I implemented CXF exception handlers to set appropriate headers which the client can use to map back to the original custom exception and message.

Below is the service class that is throwing the custom exception:

|  |  |
| --- | --- |
| SampleServiceImpl.java | |
|  | @Service("sampleService")  public class SampleServiceImpl implements SampleService {        @Autowired      UserDao userDao;        public User saveUser(User user) throws InvalidUserException{            String firstName = user.getFirstName();            if(!StringUtils.isEmpty(firstName) && "Dave".equalsIgnoreCase(firstName)) {              throw new InvalidUserException("Sorry Dave");          }            return userDao.save(user);        }  } |

Below is the original controller that would handle the InvalidUserException. The goal is to not change the controller method at all and still use CXF to make a REST call for the form creation.

|  |  |
| --- | --- |
| IndexController.java | |
|  | @RequestMapping(value = "/create", method = RequestMethod.POST)  public String search(Model model, @Valid SignupForm signupForm, BindingResult result, RedirectAttributes redirectAttributes) {        String returnPage = PAGE\_INDEX;        if (!result.hasErrors()) {          try {              model.addAttribute("form", sampleService.saveFrom(signupForm));              returnPage = PAGE\_SHOW;          } catch (InvalidUserException e) {              model.addAttribute("page\_error", e.getMessage());          }      }      return returnPage;  } |

The first step is to implement a jaxrs exception handler that will modify the response. The CXF Exception Handler class lives on the service side and is responsible for setting the response.

|  |  |
| --- | --- |
| ExceptionHandler.java | |
|  | import com.luckyryan.sample.exception.InvalidUserException;    import javax.ws.rs.core.Response;  import javax.ws.rs.ext.ExceptionMapper;    public class ExceptionHandler implements ExceptionMapper<InvalidUserException> {      public Response toResponse(InvalidUserException exception) {          Response.Status status;            status = Response.Status.INTERNAL\_SERVER\_ERROR;            return Response.status(status).header("exception", exception.getMessage()).build();      }  } |

Once this is registered in the jaxrs service provider list, it will be called automatically when the InvalidUserException is thrown.

This handler will set a 500 response code. That tells the rest client that an exception happened and will let us wire up a handler in the client. A custom header “exception” is also set with the original exception message so we can crab it from the client.

Add the ExceptionHandler to the service provider list in the server definition. This is set in cxf-servlet.xml.

|  |  |
| --- | --- |
| cxf-servlet.xml | |
|  | <!-- rest container -->  <jaxrs:server id="sampleSerivceREST" address="/rest">      <jaxrs:serviceBeans>          <ref bean="sampleServiceRESTGateway"/>      </jaxrs:serviceBeans>      <jaxrs:providers>          <bean class="org.codehaus.jackson.jaxrs.JacksonJsonProvider"/>          <bean class="com.luckyryan.sample.service.ExceptionHandler"/>      </jaxrs:providers>  </jaxrs:server> |

At this point the service will now respond with a 500 response and custom header with the exception message. Next the client will need to translate that combo into the custom exception for the controller.

The CXF Exception Handler class lives in the client.

|  |  |
| --- | --- |
| ExceptionHandler.java | |
|  | import org.apache.cxf.jaxrs.client.ResponseExceptionMapper;    import javax.ws.rs.core.Response;    public class ExceptionHandler implements ResponseExceptionMapper {        @Override      public Throwable fromResponse(Response response) {          throw new InvalidUserException(response.getHeaderString("exception"));      }  } |

This implements ResponseExceptionMapper which will give access to the Response object. Once this handler is added to the provider list it will be invoked for any 500 response for the service call.

Next it’s wired into the provider list.

**Spring Java Config & Application Context XML**  
This example app uses Spring with CXF. There are two examples below, one for Java Config and the other for xml. Only use the one relevant to your application.

Spring Java Config

|  |  |
| --- | --- |
| appConfig.java | |
|  | @Bean  public JacksonJsonProvider getJacksonJsonProvider() {      return new JacksonJsonProvider();  }    @Bean  public ExceptionHandler getExceptionHandler() {      return new ExceptionHandler();  }    @Bean  public SampleServiceREST getSampleServiceRestClient() {      List providers = new ArrayList();      providers.add(getJacksonJsonProvider());      providers.add(getExceptionHandler());      return JAXRSClientFactory.create("http://localhost:8090/services/rest",SampleServiceREST.class,providers);  } |

Spring XML version

|  |  |
| --- | --- |
| applicationContext-cxf.xml | |
|  | <!-- rest container -->  <jaxrs:client id="sampleServiceREST"                serviceClass="com.luckyryan.sample.ws.SampleServiceREST"                address="http://localhost:8090/services/rest">      <jaxrs:providers>        <bean class="org.codehaus.jackson.jaxrs.JacksonJsonProvider"/>        <bean class="com.luckyryan.sample.exception.ExceptionHandler"/>      </jaxrs:providers>  </jaxrs:client> |

That’s it! Now the application will handle the exception as before until it’s refactored into a better REST model.