ORACLE

Jakarta REST 4.0

Brainstorming

Santiago Pericas-Geertsen
Oracle
May, 2021

Overview

- Tighter integration with Jakarta CDI
  - Drops support for @Context and related artifacts
  - Backward incompatible release
- Defines default CDI scopes for Jakarta REST types
- Enables discovery of Jakarta REST types
  - Resources, providers and applications
  - Based on CDI discovery mechanism
- Drops compile-time dependency with JAXB
  - Link.JaxbAdapter and Link.JaxbLink
- Review integration with new and existing Jakarta specs
  - Jakarta Concurrency
- Proposed for Jakarta EE 10

Removal of @Context

- Use @Inject instead of @Context
  - @Inject is not allowed in parameter position
- Consider dropping support for ContextResolver
  - These are akin to CDI producers
  - Except for @Produces support to filter media types
- Constructor, field and property injection as in Jakarta CDI
- Injection in resource method parameters controlled by Jakarta REST
  - No support from CDI (yet?)

Example: Fields and Constructors

```java
@Path("/greet1")
public class SampleResource {

    @Context
    private UriInfo uriInfo;

    @HeaderParam("who")
    private String who;

    private final GreetBean bean;

    @QueryParam(”lang")
    private String lang;

    @Inject
    public SampleResource(GreetBean bean) {
        this.bean = bean;
    }
```

Jakarta REST 3.1

```java
@Path("/greet1")     // bean-defining annotation
@RequestScoped       // implied by default
public class SampleResource {

    @Inject
    private UriInfo uriInfo;

    @Inject
    @HeaderParam("who")
    private String who;

    private final GreetBean bean;
    private final String lang;

    @Inject
    public SampleResource(GreetBean bean,
        @QueryParam("lang") String lang) {
        this.bean = bean;
        this.lang = lang;
    }
```

Jakarta REST 4.0

Field Injection

- Use @Inject where @Context was used before
- Add @Inject to inject @*Param's
  - @*Param's are now CDI qualifiers
- Option to inject @*Param's in CDI constructors

Constructor Injection

- Use of @Inject is required for injection
- @*Param injection is allowed in bean constructors
  - This was problematic with multiple DI's in play
- Only a single constructor with @Inject supported in CDI
  - Jakarta REST supported calling constructor with more parameters
- Default constructors for proxying
  - Anything we can do about these?

Example: Resource Methods

```java
@GET
@Produces(MediaType.TEXT_PLAIN)
public String getMessage() {
    return bean.getMessage(lang) + " " + who;
}

@POST
@Path("async")
@Consumes(MediaType.TEXT_PLAIN)
public void putMessageAsync(
  @QueryParam("override") boolean override,
  String message,
  @Suspended AsyncResponse ar) {
    Executors.newSingleThreadExecutor().submit(() -> {
      // …
      ar.resume("Done");
    });
}
```

Jakarta REST 3.1

```java
@GET
@Produces(MediaType.TEXT_PLAIN)
public String getMessage() {
    return bean.getMessage(lang) + " " + who;
}

@POST
@Path("async")
@Consumes(MediaType.TEXT_PLAIN)
public void putMessageAsync(
  @QueryParam("override") boolean override,
  @Entity String message,
  AsyncResponse ar) {
    Executors.newSingleThreadExecutor().submit(() -> {
      // …
      ar.resume("Done");
    });
}
```

Jakarta REST 4.0

Resource Method Injection

- No need to annotate method with @Inject
  - Only makes code more verbose
  - Jakarta REST implementation calls these methods
- Allow injection of arbitrary CDI beans as parameters
  - Whether defined by Jakarta REST or not
  - Must be a CDI bean for injection to be supported
- Use of new annotation @Entity is required
- No need to use @Suspended with AsyncResponse
  - This annotation will be removed for consistency

Example: SSE

```java
@GET
@Path("sse")
@Produces(MediaType.SERVER_SENT_EVENTS)
public void getMessage(
  @HeaderParam("who") String who,
  @Context Sse sse,
  @Context SseEventSink sseEventSink)
{
    sseEventSink.send(
      sse.newEvent(
        bean.getMessage(lang) + " " + who));
}
```

Jakarta REST 3.1

```java
// bean injection of Sse and SseEventSink
@GET
@Path("sse")
@Produces(MediaType.SERVER_SENT_EVENTS)
public void getMessage(
  @HeaderParam("who") String who,
  Sse sse,
  SseEventSink sseEventSink)
{
    sseEventSink.send(
      sse.newEvent(
        bean.getMessage(lang) + " " + who));
}
```

Jakarta REST 4.0

Bean Discovery and Scopes

- Make @Path, @Provider and @ApplicationPath bean defining
  - Avoids the need for another annotation for discovery
- Resource classes are @RequestedScoped
- Jakarta REST providers are @ApplicationScoped
- Application sub-classes are @ApplicationScoped
  - Can change if multiple subclasses are supported
- Default scopes can be overridden
  - Resource and provider classes

Application Subclasses

- Should support 0 Application subclasses
  - Automatically collect resources and providers in *synthetic* subclass
  - Uses CDI bean discovery mechanism
- Should consider support for 2 or more Application subclasses?
  - Can be useful for specs built on top of Jakarta REST
  - E.g., multiple application subclasses with different security configuration
  - If supported, Application subclasses need to be in request scope
    - Some limitations when request scope is not active

Some Open Questions

- Implications of new CDI Lite
- SSE support with reactive streams
  - Return Flow.Publisher<?> from SSE resource methods
- CDI spec mandates non-private constructors for proxying
  - Implementations like Weld mostly get around this
  - https://docs.jboss.org/weld/reference/latest/en-US/html/configure.html#relaxedConstruction

Thank you

Santiago Pericas-Geertsen

https://github.com/spericas/jaxrs-api/tree/release-4.0