JAX-RS: Java™ API for RESTful Web Services

patent rights.

5. Definitions. For the purposes of this Agreement: ”Independent Implementation” shall mean an implementation of

Contents

Chapter 1

Introduction

1.6 Expert Group Members

Jerome Louvel (Individual Member)

Hamid Ben Malek (Fujitsu Limited)

Ryan J. McDonough (Individual Member)

Felix Meschberger (Day Software, Inc.)

David Orchard (BEA Systems)

Dhanji R. Prasanna (Individual Member)

Julian Reschke (Individual Member)

Jan Schulz-Hofen (Individual Member)

Joel Smith (IBM)

Stefan Tilkov (innoQ Deutschland GmbH)

1.7 Acknowledgements

During the course of this JSR we received many excellent suggestions. Special thanks to Martin Matula, Gerard Davison, Jakub Podlesak and Pavel Bucek from Oracle as well as Pete Muir and Emmanuel Bernard

Chapter 2

Applications

A JAX-RS application consists of one or more resources (see Chapter 3) and zero or more providers (see Chapter 4). This chapter describes aspects of JAX-RS that apply to an application as a whole, subsequent chapters describe particular aspects of a JAX-RS application and requirements on JAX-RS implementations.

2.1 Configuration

Chapter 3

Resources

3.2 Fields and Bean Properties

When a resource class is instantiated, the values of fields and bean properties annotated with one the following annotations are set according to the semantics of the annotation:

`@MatrixParam` Extracts the value of a URI matrix parameter.

`@QueryParam` Extracts the value of a URI query parameter.

`@PathParam` Extracts the value of a URI template parameter.

`@CookieParam` Extracts the value of a cookie.

`@HeaderParam` Extracts the value of a header.

`@Context` Injects an instance of a supported resource, see chapters 9 and 10 for more details.

Because injection occurs at object creation time, use of these annotations (with the exception of `@Context`) on resource class fields and bean properties is only supported for the default per-request resource class lifecycle. An implementation SHOULD warn if resource classes with other lifecycles use these annotations on resource class fields or bean properties.

A JAX-RS implementation is only required to set the annotated field and bean property values of instances

```
NotFoundException
```

`void` Results in an empty entity body with a 204 status code.

`Response` Results in an entity body mapped from the entity property of the `Response` with the status code specified by the status property of the `Response`. A `null` return value results in a 204 status code. If the status property of the `Response` is not set: a 200 status code non- `null` entity

```
1  @Path("widgets/{id}")
2  public class Widget {
3    ...
4  }
```

In the above example the `Widget` resource class is identified by the relative URI path `widgets/{id}`. The

3.4.1 Sub Resources

3.5 Declaring Media Type Capabilities

Application classes can declare the supported request and response media types using the `@Consumes` and `@Produces` annotations respectively. These annotations MAY be applied to a resource method, a resource class, or to an entity provider (see Section 4.2.3). Use of these annotations on a resource method overrides

- Remove members that do not match U.
- Remove members derived from T_D (those added in step 2(c)i) for which the final capturing group value is neither empty nor '/'.

(e) If E is empty, then no matching resource can be found and the algorithm terminates by generating a `NotFoundException`

Chapter 4

Providers

The JAX-RS runtime is extended using application-supplied provider classes. A provider is a class that implements one or more JAX-RS interfaces introduced in this specification and that may be annotated with `@Provider` for automatic discovery. This chapter introduces some of the basic JAX-RS providers; other providers are introduced in Chapter 5 and Chapter 6.

`java.io.Reader`	All media types (

4.2.6 Content Encoding

Content encoding is the responsibility of the application. Application-supplied entity providers MAY perform such encoding and manipulate the HTTP headers accordingly.

4.3 Context Providers

Context providers supply context to resource classes and other providers. A context provider class implements the `ContextResolver<T>` interface and may be annotated with `@Provider` for automatic discovery. E.g., an application wishing to provide a customized `JAXBContext` to the default JAXB entity providers would supply a class implementing `ContextResolver<JAXBContext>`.

Context providers MAY return `null` from the `getContext` method if they do not wish to provide their

4.5 Exceptions

Exception handling differs depending on whether a provider is part of the client runtime or server runtime. This is covered in the next two sections.

interface supports configuration of:

Properties Name-value pairs for additional configuration of features or other components of a JAX-RS implementation.

Chapter 6

Filters and Interceptors

Filters and entity interceptors can be registered for execution at well-defined extension points in JAX-RS

```
ContainerRequestContext
```

```
@Provider
class GzipInterceptor implements ReaderInterceptor, WriterInterceptor {

    @Override
    Object aroundReadFrom(ReaderInterceptorContext ctx) ... {
        if (isGzipped(ctx)) {
            InputStream old = ctx.getInputStream();
            ctx.setInputStream(new GZIPInputStream(old));
            try {
                return ctx.proceed();
            } finally {
                ctx.setInputStream(old);
            }
        } else {
            return ctx.proceed();
```

6.5 Binding

Binding is the process by which a filter or interceptor is associated with a resource class or method (Server

```
1   @NameBinding
2   @Target({ ElementType.TYPE, ElementType.METHOD })
3   @Retention(value = RetentionPolicy.RUNTIME)
4   public @interface Logged { }
```

Multiple filters and interceptors can be bound to a single resource method using additional annotations. For example, given the following filter:

```
1   @Provider
2   @Authenticated
3   class AuthenticationFilter implements ContainerRequestFilter {
4       ...
5   }
```

method `hello` above could be decorated with `@Logged` and `@Authenticated` in order to provide both logging and authentication capabilities to the resource.

The following example defines a dynamic feature that binds the filter

The order in which filters and interceptors that belong to the same priority class are executed is implementation dependent.

6.7 Exceptions

6.7.1 Server Runtime

When a filter or interceptor method throws an exception, the server runtime will process the exception as described in Section 4.5.1. As explained in Section 4.4, an application can supply exception mapping providers. At most one exception mapper MUST be used in a single request processing cycle to avoid potentially infinite loops.

Chapter 7

Validation

Validation is the process of verifying that some data obeys one or more pre-defined constraints. The Bean

to method parameters, they can be used in any location in which the JAX-RS binding annotations are allowed with the exception of constructors and property setters. Rather than using method parameters, the `MyResourceClass` shown above could have been written as follows:

```
1  @Path("/")
2  class MyResourceClass {
3
4    @NotNull @FormParam("firstName")
5    private String firstName;
6
7    @NotNull @FormParam("lastName")
8    private String lastName;
9
10   private String email;
11
12   @FormParam("email")
13   public void setEmail(String email) {
14     this.email = email;
15   }
16
17   @Email
18   public String getEmail() {
19     return email;
20   }
21   ...
22 }
```

Note that in this version, `firstName` and `lastName` are fields initialized via injection and `email` is a resource class property. Constraint annotations on properties are specified in their corresponding getters.

Constraint annotations are also allowed on resource classes. In addition to annotating fields and properties, an annotation can be defined for the entire class. Let us assume that `@NonEmptyNames` validates that one of the two *name* fields in `MyResourceClass` is provided. Using such an annotation, the example above can be extended as follows:

```
1  @Path("/")
2  @NonEmptyNames
3  class MyResourceClass {
4
5    @NotNull @FormParam("firstName")
6    private String firstName;
7
8    @NotNull @FormParam("lastName")
9    private String lastName;
10
11   private String email;
12
```

Chapter 8

Asynchronous Processing

This chapter describes the asynchronous processing capabilities in JAX-RS. Asynchronous processing is supported both in the Client API and in the Server API.

8.1 Inti0(a4ductiong)]TJ/F27 10.9091 Tf 0 30.8918 Td [(Asynchronous)2868(pro thesiltreadstantlasresponslblesforsifs

8.3 EJB Resource Classes

```
1   Future<Customer> ff = target.resolveTemplate("id", 123).request().async()
2       .get(new InvocationCallback<Customer>() {
3           @Override
4           public void completed(Customer customer) {
5               // Do something
6           }
7           @Override
8           public void failed(Throwable throwable) {
9               // Process error
10          }
11      });
12
13  // After waiting for a while ...
14  if (!ff.isDone()) {
15      ff.cancel(true);
16  }
```

Even though it is recommended to pass an instance of `InvocationCallback` when executing an asynchronous call, it is not mandated. When omitted, the `Future<T>` returned by the invocation can be used to gain access to thein181 0 Tnseitanin18TJ/Fod9.9626 Tf 239.09 0 T29.602uture<T>

Chapter 9

Context

JAX-RS provides facilities for obtaining and processing information about the application deployment context and the context of individual requests. Such information is available to `Application` subclasses (see Section 2.1), root resource classes (see Chapter 3), and providers (see Chapter 4). This chapter describes these facilities.

1

```
6         return responseBuilder.build();
7       else
8         return doUpdate(foo);
9   }
```

The application could also set the content location, expiry date and cache control information into the returned `ResponseBuilder` before building the response.

9.2.5 Security Context

The `SecurityContext` interface provides access to information about the security context of the current request. An instance of `SecurityContext` can be injected into a class field or method parameter using the `@Context` annotation. The methods of `SecurityContext` provide access to the current user principal, information about roles assumed by the requester, whether the request arrived over a secure channel and the authentication scheme used.

9.2.6 Providers

The `Providers` interface allows for lookup of provider instances based on a set of search(set)-anace allows for loof280(lo

```
                }
```

```
20  }
```

Note that the instance returned by the resource locator `findWidget` in `WidgetsResource` is initialized using the injected `ResourceContext` before it is returned. Without this step, the `headers` field in `WidgetResource` will not be properly initialized.

10.2.1 Servlets

In a product that also supports the Servlet specification, implementations MUST support JAX-RS appli-

For example, assuming CDI is enabled via the inclusion of a `beans.xml` file, a CDI-style bean that can be defined as a JAX-RS resource as follows:

```
1  @Path("/cdibean")
2  public class CdiBeanResource {
3
4
```

If an `ExceptionMapper` for a `EJBException` or subclass is not included with an application then exceptions thrown by an EJB resource class or provider method MUST be unwrapped and processed as described in Section 3.3.4.

Appendix A

Summary of Annotations

Annotation	Target	Description
`MatrixParam`		

Header	Description

Appendix C

Processing Pipeline

Figure C.1: JAX-RS Server Processing Pipeline.

Figure C.2: JAX-RS Client Processing Pipeline.

Appendix D

Change Log

D.1 Changes Since 2.0 Proposed Final Draft

- Section 4.2.1 and 4.2.2: Updated last logical step separating client and server runtimes.
- Section 4.2.4: New exception `NoContentException`

- Chapter 5: Updated samples and text related to the configuration of Client API types. Method `configuration` has been removed in favor of the `Configurable` interface.
- Chapter 5: `ClientFactory` renamed to `ClientBuilder`.
- Chapter 5: Dropped support for `@Uri` annotation.
- Section 6.3: New paragraph clarifying that entity interceptors are not called if a `readFrom` or `writeTo`

- Section 6.5.3: `DynamicBinder` replaced by `DynamicFeature`.
- Section 6.7: Clarified processing of responses mapped using exception mappers.

- Section 6.5.1: Clarified global binding in relation to the new semantics of `@Provider` for automatic discovery.
- Section 6.5.3: The `DynamicBinding` interface, intended to be implemented by filters and entity interceptors, is replaced by `DynamicBinder`. A dynamic binder is a new type of provider that binds filters and entity interceptors with resource methods.

D.6 Changes Since 1.1 Release

- Section 1.1: Updated URLs to JSR pages, etc.
- Section 1.3: Removed Client APIs as non-goal.
- Section 1.5: Added new terminology.
- Section 1.6: List 2.0 expert group members.
- Section 1.7: acknowledgements for 2.0 version.
- Chapter 2: Somewhat generic section on validation removed to avoid confusion with the type of validation defined in Chapter “Validation”.

- Section 3.2: Add support for static fromString method.
- Section 3.6: Clarify annotation inheritance.
- Section 9.2.5: Fix typo.
- Section 10.1: Additional considerations related to filters consuming request bodies.

D.8 Changes Since Proposed Final Draft

- Section 3.7.2: Additional sort criteria so that templates with explicit regexs are sorted ahead of those with the default.
- Sections 3.7.2, 3.8, 4.2.3 and 4.3.1: Q-values not used in `@Consumes` or `@Produces`.

- Section 9.2.6: Changed name to Providers, removed entity provider-specific text to reflect more