JavaOne™

ORACLE®

Keep Learning with Oracle University

ORACLE

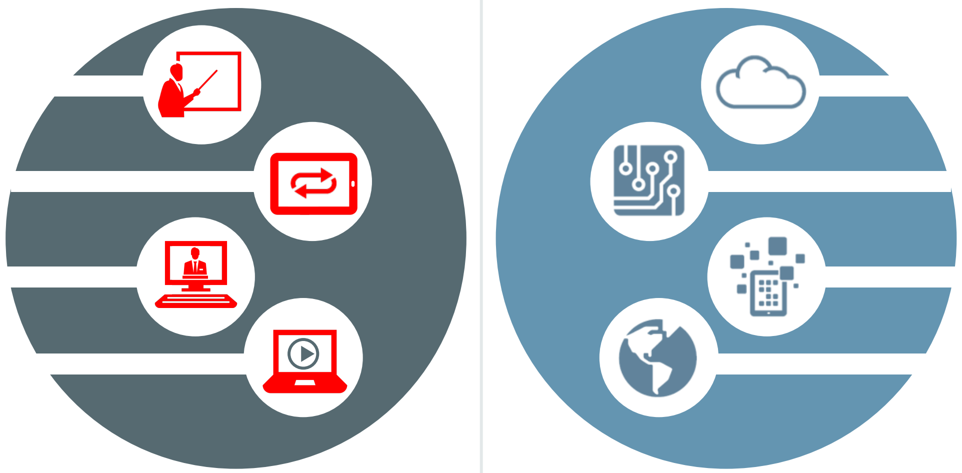
UNIVERSITY

Classroom Training

Learning Subscription

Live Virtual Class

Training On Demand



Cloud

Technology

Applications

Industries

education.oracle.com

Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

What's Next for JAX-RS 2.1?

CON-4192

Santiago Pericas-Geertsen
JSR 370 Co-Spec Lead
Oracle

Safe Harbor Statement

A Recap of JAX-RS 2.0

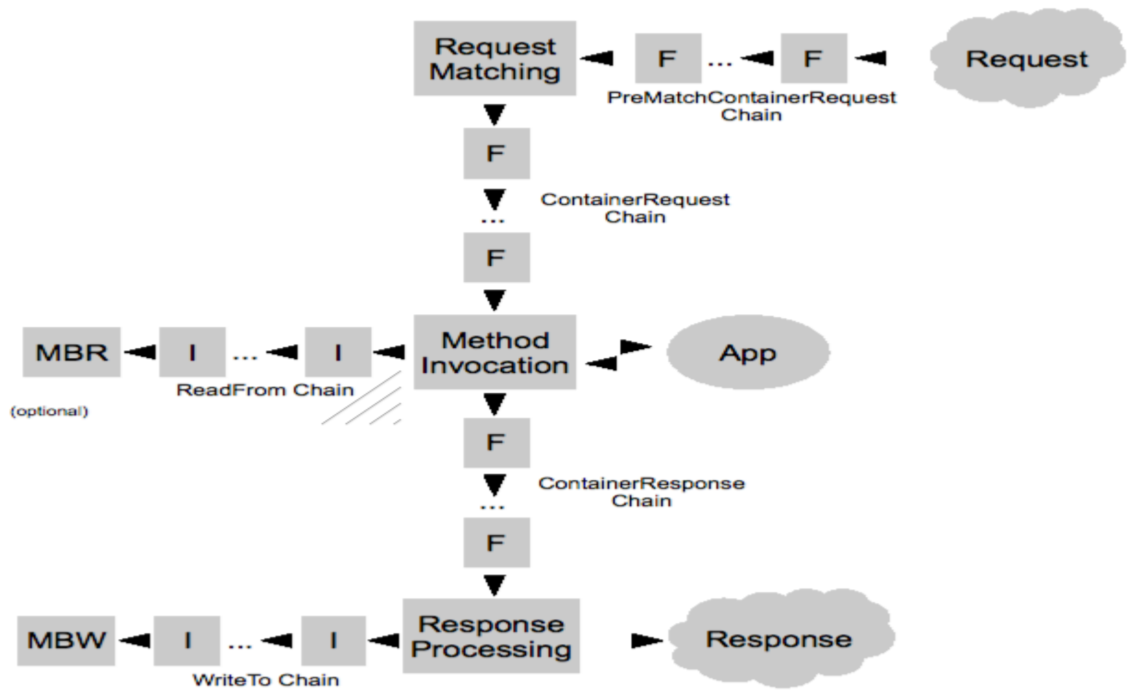
Lots of new features in the last major update

JAX-RS 2.0 Features

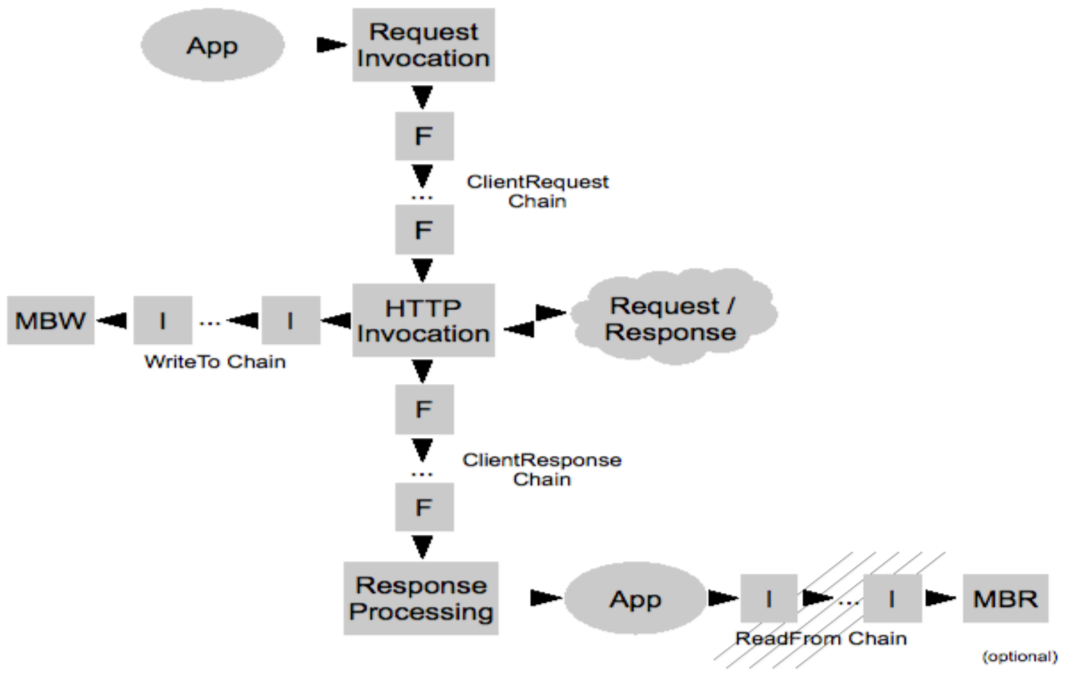
Largest release of JAX-RS!

- New Java client API
- Support for filters and interceptors
- Integration with the Validation API
- New configuration API
- Basic support for hypermedia (link headers)
- Asynchronous processing

JAX-RS 2.0 Server Pipeline



JAX-RS 2.0 Client Pipeline



What’s planned for JAX-RS 2.1?

Dot release with great new features

JAX-RS 2.1 – The Major Features

- Reactive Programming
- Non-Blocking I/O
- Server-Sent Events
- Alignment with JSON-B and MVC

Reactive Programming

Because Async Programming is Hard

Asynchronous Processing in 2.0

- Server side:
  - Using @Suspended and AsyncResponse
  - Resume execution on a different thread
- Client side:
  - Future<T>
  - InvocationCallback<T>

Example Using Future<T>

```
Client client = ClientBuilder.newClient();
WebTarget target = client.target("http://…");

Future<String> f =
    target.request().async().get(String.class);

// some time later …

String user = f.get();
```

What's the problem here?

Example using InvocationCallback<T>

```
Client client = ClientBuilder.newClient();
WebTarget target= client.target("http://…");

target.request().async().get(new
    InvocationCallback<String>() {
        @Override
        public void completed(String user) {
            // do something
        }

        @Override
        public void failed(Throwable t) {
            // do something
        }
    });
```

Example using InvocationCallback<T>’s

```
target1.request().async().get(new
    InvocationCallback<String>() {
        public void completed(String user) {
            target2.request().header(“user”, user).async().get(
                new InvocationCallback<String>() {
                    public void completed(String quote) {
                       System.out.println(quote);
                    }
                    public void failed(Throwable t) {
                      // do something
                    }
                });
            }
        public void failed(Throwable t) {
            // do something
        }
    }));
```

Pyramid of Doom!

Uses cases for Async Computations

- Compose two (or more) asynchronous tasks
- Combine the output of two (or more) asynchronous tasks
- Execute a Consumer after a task completes
- Wait for all tasks to complete
- Wait for any of the tasks to complete
- *And many more!*

Meet CompletableFuture<T> in JDK 8

Proposal for JAX-RS 2.1

```
CompletionStage<String> cs1 =
    target1.request().rx().get(String.class);

CompletionStage<String> cs2 =
    cs1.thenCompose(user ->
      target2.request().header(“user”, user)
         .rx().get(String.class));

cs2.thenAccept(quote -> System.out.println(quote));
```

But Wait There is More … Other Rx APIs

```
// Implement an RxInvoker<T>
class ObservableInvoker implements RxInvoker<Observable> {
    …
}

Observable<String> cs1 =
    target1.request().rx(ObservableInvoker.class)
    .get(String.class);
```

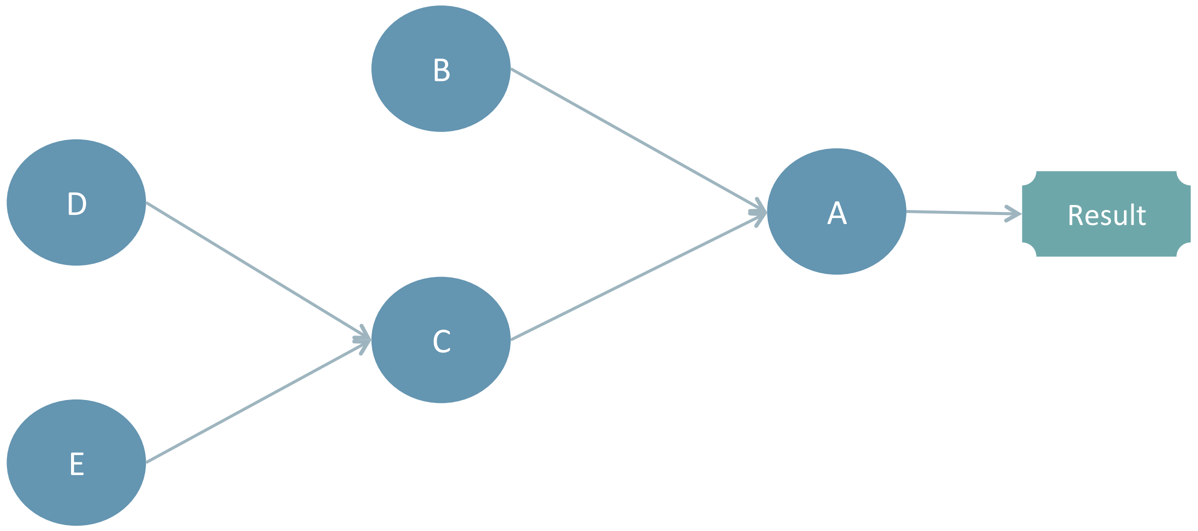
Extension Point

Can we make it even easier?

Warning!! Bleeding edge ideas …

- Combining async tasks is still hard!
- What if dependencies could be made declaratively?
- More general async ideas, not exclusive to JAX-RS

DeclarativeRx Example: Tasks



DeclarativeRx Sample

Using string values for simplicity

```
class DeclarativeRxHandler {

    @FinalResult
    public String get(@PartialResult("A") String a) { return a; }

    @PartialResult("B")
    public CompletableFuture<String> getB() { return newB(…); }

    @PartialResult("D")
    public CompletableFuture<String> getD() { return newD(…); }

    @PartialResult("E")
    public CompletableFuture<String> getE() { return newE(…); }

    …
}
```

Final result

Leaf tasks

DeclarativeRx Sample

Using string values for simplicity

```
 class DeclarativeRxHandler {
     …

     @PartialResult(“C”)
     public CompletableFuture<String> getC(@PartialResult(“D”) String d,
                                           @PartialResult(“E”) String e) {
         return newC(d, e);
     }

     @PartialResult(“A”)
     public CompletableFuture<String> getA(@PartialResult(“B”) String b,
                                           @PartialResult(“C”) String c) {
         return newA(b, c);
     }
}
```

Declarative dependencies

DeclarativeRx Sample
Bootstrapping

```
DeclarativeRxProcessor<String> p = new DeclarativeRxProcessor<>();

p.processHandler(new DeclarativeRxHandler());
```

Do the work on my behalf!

NIO

High-performance IO for JAX-RS

Motivation

- Certain apps need more control over IO
- Higher throughput is hard with blocking IO
- `StreamingOutput` in JAX-RS

StreamingOutput in JAX-RS 2.0

```
@GET
public Response get() {
    return Response.ok(new StreamingOutput() {
        @Override
        public void write(OutputStream out) throws … {
            out.write(…);
        }
    }).build();
}
```

Direct access to stream

Blocking call

NIO in JAX-RS 2.1

```
@GET
public Response get() {
    return Response.ok(out -> {
        out.write(…);
        if (moreData()) return true;
        return false;
    }).build();
}
```

Yes, we have lambdas now!

Non-blocking Call

NIO Example (Server)

```
@POST @Consumes(MediaType.APPLICATION_OCTET_STREAM)
public void upload(@QueryParam("path") String path,
                   @Context Request request,
                   @Suspend AsyncResponse response) {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    byte[] buffer = new byte[FOUR_KB];

    request.entity(in -> {     // reader handler
            try {
                if (in.isFinished()) {
                    files.put(path, out.toByteArray());
                    out.close();
                    response.resume("Upload completed");
                } else {
                    final int n = in.read(buffer);
                    out.write(buffer, 0, n);
                }
            } catch (IOException e) {
                throw new WebApplicationException(e);
            } }); }
```

Async Response

NIO Example (Server)

```
@POST @Consumes(MediaType.APPLICATION_OCTET_STREAM)
public void upload(@QueryParam("path") String path,
                   @Context Request request,
                   @Suspend AsyncResponse response) {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    byte[] buffer = new byte[FOUR_KB];

    request.entity(in -> {      // reader handler
            try {
                if (in.isFinished()) {
                    files.put(path, out.toByteArray());
                    out.close();
                    response.resume("Upload completed");
                } else {
                    final int n = in.read(buffer);
                    out.write(buffer, 0, n);
                }
            } catch (IOException e) {
                throw new WebApplicationException(e);
            } }); }
```

Non-blocking Call

NIO Example (Client)

```
Client client = ClientBuilder.newClient();
ByteArrayInputStream in = new ByteArrayInputStream(…);
byte[] buffer = new byte[FOUR_KB];

client.target("/file")
      .request(MediaType.APPLICATION_OCTET_STREAM)
      .nio().post(out -> {                    // writer handler
                try {
                    final int n = in.read(buffer);
                    if (n >= 0) {
                        out.write(buffer, 0, n);
                        return true;    // more to write
                    }
                    in.close();
                    return false;       // we're done
                } catch (IOException e) {
                    throw new WebApplicationException(e); } });
```

Stream to read file

NIO Example (Client)

```
Client client = ClientBuilder.newClient();
ByteArrayInputStream in = new ByteArrayInputStream(…);
byte[] buffer = new byte[FOUR_KB];

client.target("/file")
      .request(MediaType.APPLICATION_OCTET_STREAM)
      .nio().post(out -> {                         // writer handler
                try {
                    final int n = in.read(buffer);
                    if (n >= 0) {
                        out.write(buffer, 0, n);
                        return true;    // more to write
                    }
                    in.close();
                    return false;       // we're done
                } catch (IOException e) {
                    throw new WebApplicationException(e); } });
```

Non-blocking Call

NIO in JAX-RS 2.1

- Presented proposal for resource methods
  - Direct access to underlying stream à la `StreamingOutput`
  - Using lambdas as event handlers: read/write, completion and error
- What about NIO on MBR and MBW's?
  - Still under investigation

SSE

Event stream support for JAX-RS

What is SSE?

- A W3C standard that is part of the HTML5 family
- Defines `EventSource` API and format `text/event-stream`
- Server push only
- Runs over HTTP
- Much better alternative to polling
  - Regardless of size: short, long, etc.

Sample SSE Stream

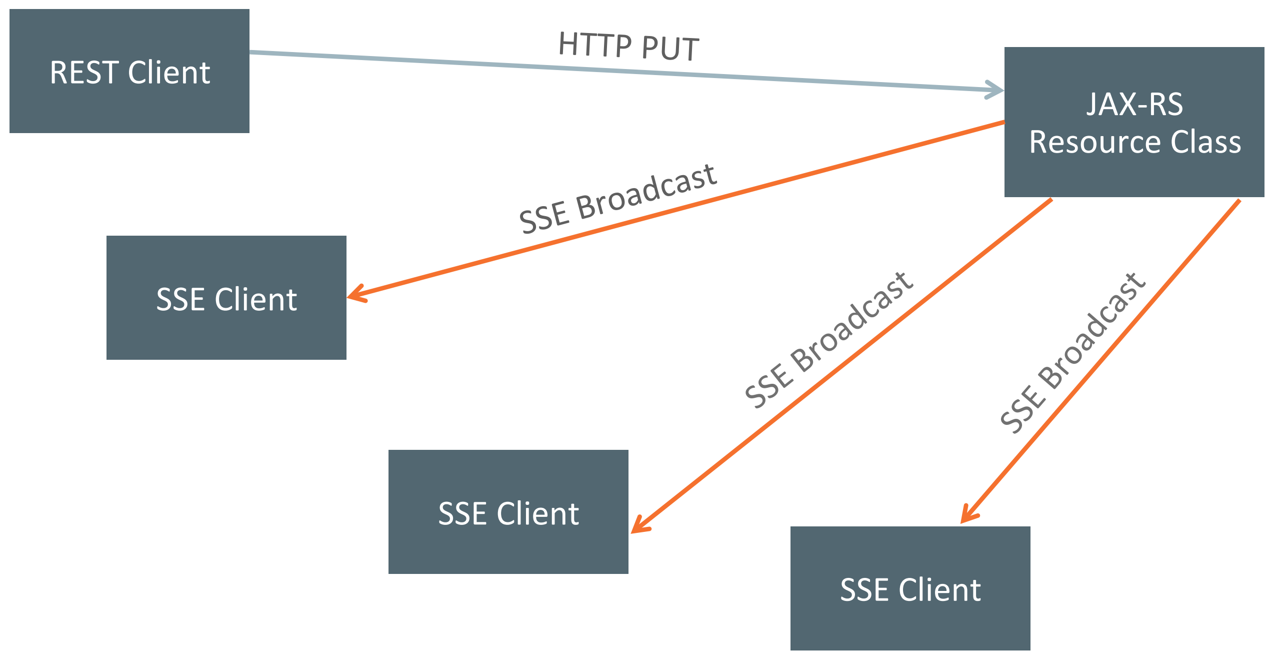
```
event: javaone15\n
data: must not miss\n
data: JAX-RS presentation!\n\n

event: lateparty\n
data: must drink lots of\n
data: free beer!\n\n
```

Most apps just use `data:`

Also available `id:` and `retry:`

SSE Sample


REST Client
HTTP PUT
JAX-RS Resource Class
SSE Broadcast
SSE Client
SSE Broadcast
SSE Client
SSE Broadcast
SSE Client

SSE Example: Broadcaster

```
@Path(“message/stream”)
public class MessageStreamResource {

    …

    @Inject
    public MessageStreamResource(SseContext ctx) {
        this.ctx = ctx;
        this.bc = ctx.newBroadcaster();
    }

    @GET @Produces(“text/event-stream”)
    public SseEventOutput getMessageStream() {
        SeeEventOutput eventOutput = ctx.newOutput();
        bc.register(eventOutput);
        return eventOutput;
    }
    …
```

Injects SSE Context

Leaves HTTP Connection Open

SSE Example: Messages

```
@Path(“message/stream”)
public class MessageStreamResource {
    …
    @PUT
    @Consumes(“text/plain”)
    public void putMessage(String message) {
        OutboundSseEvent event = ctx.newEvent()
            .data(message).build();
        bc.broadcast(event);
    }
}
```

Broadcast to all connections

SSE in JAX-RS 2.1 Client API (Pull Style)

```
WebTarget wt = ClientBuilder.newClient().target("…");

SseEventInput input = target.request("text/event-stream")
                            .get(SseEventInput.class);

// Consume all events until closed
while (!input.isClosed()) {
    System.out.println(input.read().readData());
}
```

Summary

Almost done …

JAX-RS 2.1

- Dot release that focuses on “modern” features
- Targeting Early Draft (EDR) end of 2015
- Runs on JDK 8:
  - Lambdas
  - CompletableFuture<T>
- Subscribe to `users@jax-rs-spec.java.net`

Q/A

It is time to REST now

Santiago Pericas-Geertsen
JSR 370 Spec Lead
Oracle