RESTful Java Evolves

Bill Burke
Fellow, Red Hat

Speaker's Qualifications

- RESTEasy project lead
  - Fully certified JAX-RS implementation
- JAX-RS JSR member
  - Also served on EE 5 and EJB 3.0 committees
- JBoss contributor since 2001
  - Clustering, EJB, AOP, Kernel, Resteasy
- Published author
  - Books, articles

Agenda

- Why REST?
- JAX-RS 1.1 Overview
- JAX-RS 2.0
  - Async HTTP
  - Client Framework
  - Filters/Interceptors

Why REST?

- Architectural strengths
  - Decoupled systems
  - Change resistant
  - Evolvable
  - Self-describable

Why REST?

- HTTP is ubiquitous
- Modern languages have client and server support
- Services with fixed, known interactions and behavior

Fundamental Benefits

- Lightweight interoperability
- Zero-footprint requirement
- Easily bring your services to
  - Perl, python, php, ruby
  - iPhone, Android

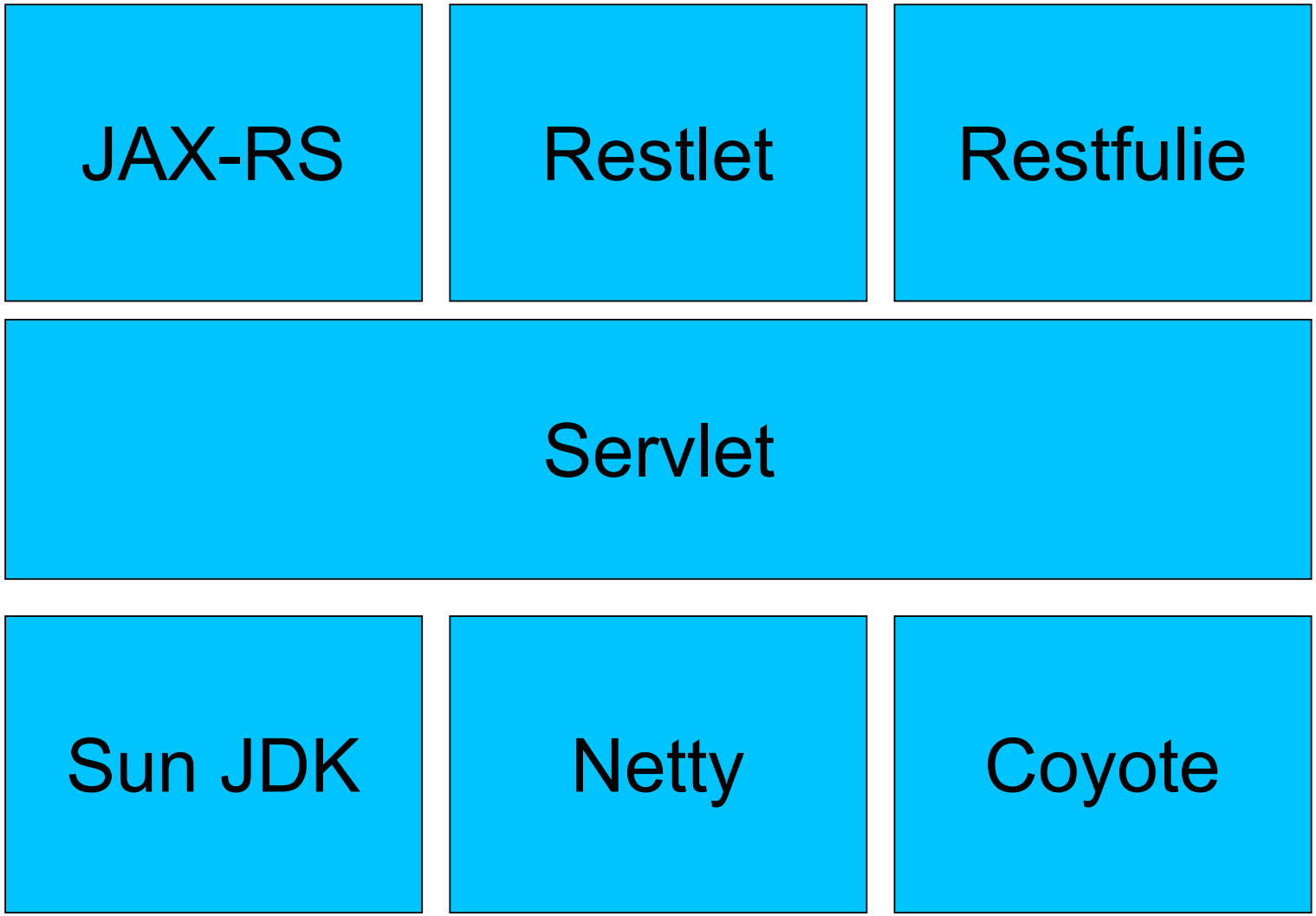
RESTful Java

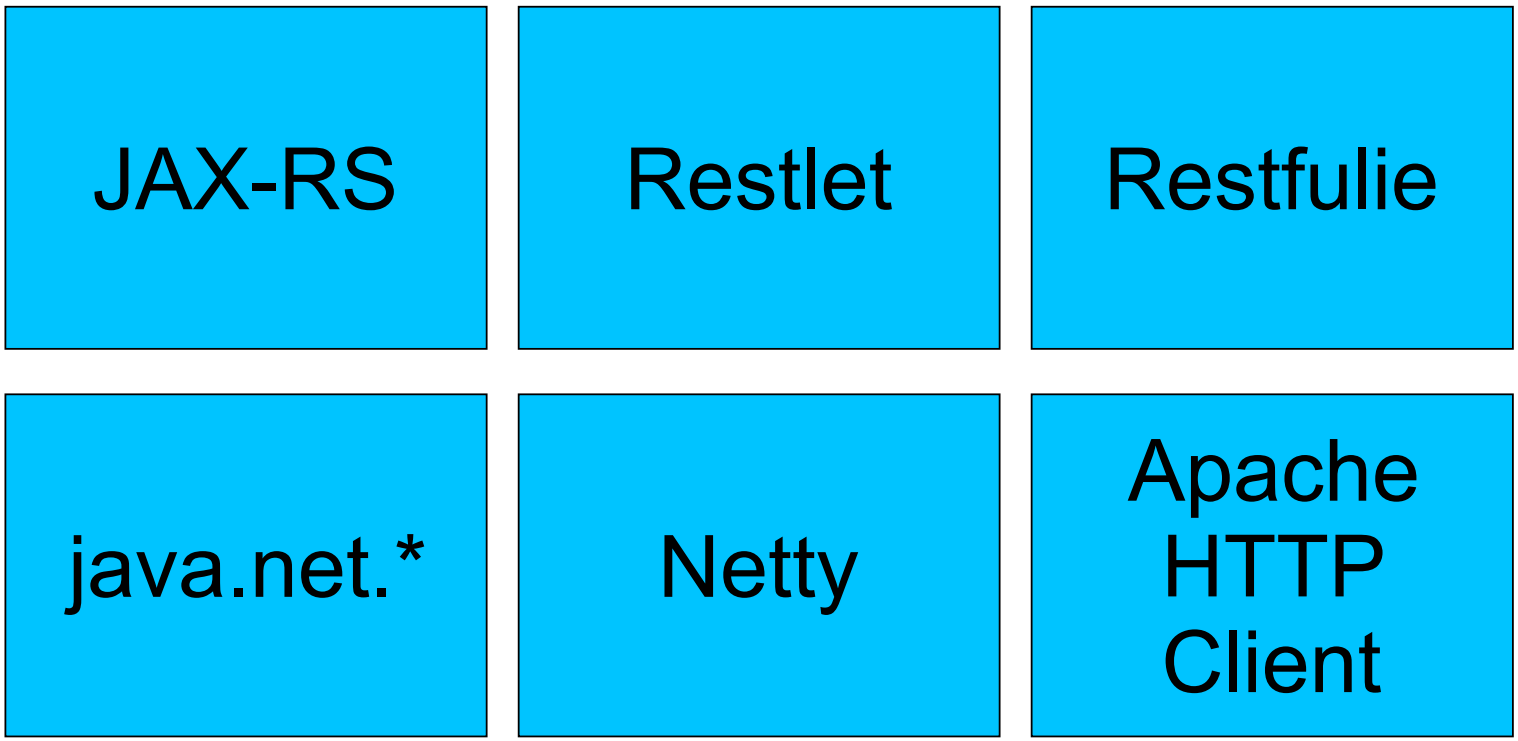
Levels of productivity

- Embeddable HTTP Container
  - Sun JDK, Netty
- Servlets
- JAX-RS, Restlet, Restfulie, etc.

Levels of Productivity

- java.net.URL, java.net.HttpUrlConnection
- Apache HTTP Client, Netty
- JAX-RS 2.0, Restfulie, Restlet


JAX-RS
Restlet
Restfulie
Servlet
Sun JDK
Netty
Coyote


JAX-RS
Restlet
Restfulie
java.net.*
Netty
Apache HTTP Client

RESTful Java with JAX-RS

JAX-RS

- EE 6 Required Specification
- Server-side Annotation Framework only (until JAX-RS 2.0)
- Allows you to map HTTP requests to Java method invocations

JAX-RS Strengths

- Annotation based (on server side)
  - DSL for doing HTTP
- Stays out of your way
  - Doesn’t impose an implementation architecture
- Familiar implementation strategy
  - Enterprise Java usually means RDBMs-based systems

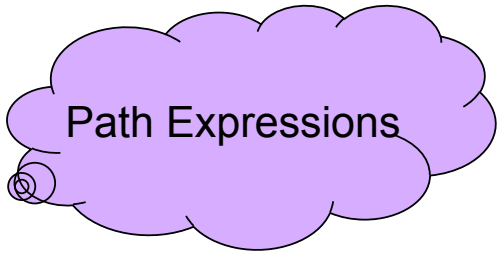
JAX-RS Weaknesses

- No client API (until JAX-RS 2.0)
- Little hypermedia support
  - Vendor specific frameworks do have some tools

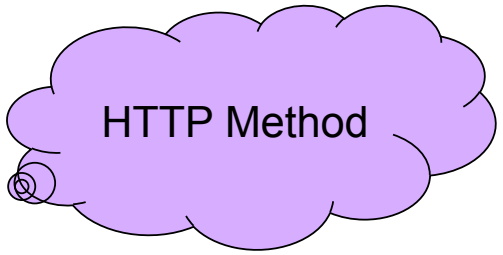
What does it look like?


```
@Path("/orders")
public class OrderResource {

   @GET
   @Path("/{order-id}")
   @Produces("application/xml", "application/json")
   Order getOrder(@PathParam("order-id") int id,
                  @HeaderParam("ETag") String tag,
                  @QueryParam("lang") Locale lang)
   {
      …
   }

}
```

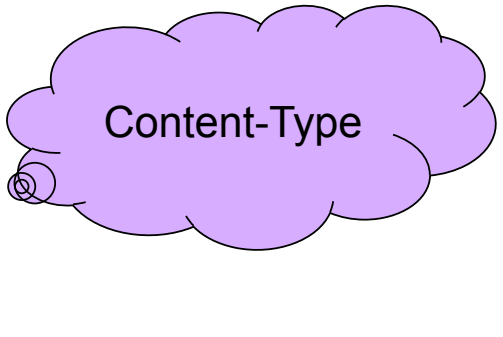

```
@Path("/orders")
public class OrderResource {

   @GET
   @Path("/{order-id}")
   @Produces("application/xml", "application/json")
   Order getOrder(@PathParam("order-id") int id,
                  @HeaderParam("ETag") String tag,
                  @QueryParam("lang") Locale lang)
   {
      …
   }

}
```

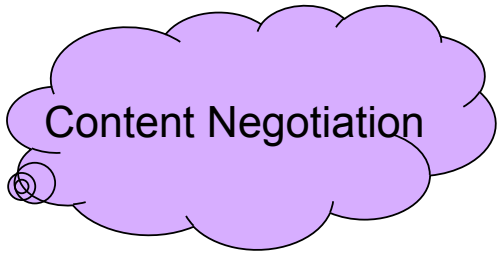



```
@Path("/orders")
public class OrderResource {

   @GET
   @Path("/{order-id}")
   @Produces("application/xml", "application/json")
   Order getOrder(@PathParam("order-id") int id,
                  @HeaderParam("ETag") String tag,
                  @QueryParam("lang") Locale lang)
   {
      …
   }

}
```



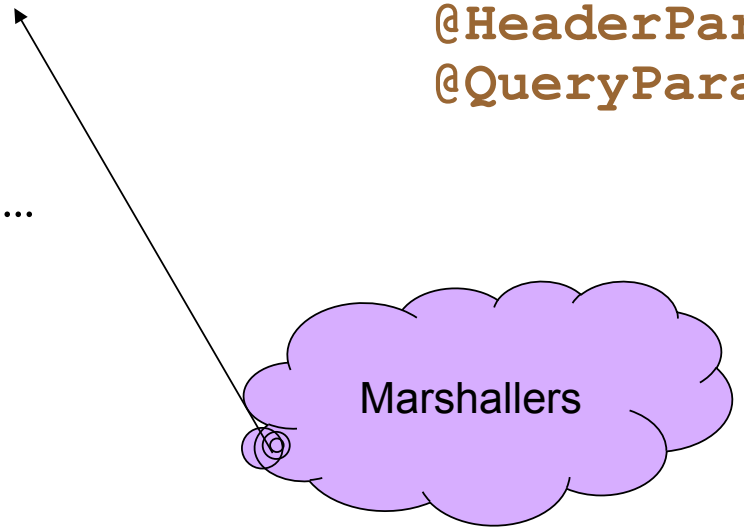

```
@Path("/orders")
public class OrderResource {

   @GET
   @Path("/{order-id}")
   @Produces("application/xml", "application/json")
   Order getOrder(@PathParam("order-id") int id,
                  @HeaderParam("ETag") String tag,
                  @QueryParam("lang") Locale lang)
   {
      …
   }

}
```



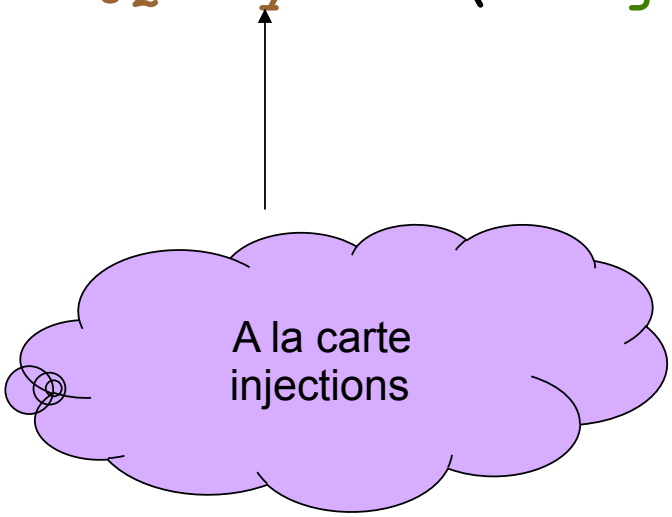



```
@Path("/orders")
public class OrderResource {

   @GET
   @Path("/{order-id}")
   @Produces("application/xml", "application/json")
   Order getOrder(@PathParam("order-id") int id,
                  @HeaderParam("ETag") String tag,
                  @QueryParam("lang") Locale lang)
   {
      …
   }

}
```

```
@Path("/orders")
public class OrderResource {

   @GET
   @Path("/{order-id}")
   @Produces("application/xml", "application/json")
   Order getOrder(@PathParam("order-id") int id,
                  @HeaderParam("ETag") String tag,
                  @QueryParam("lang") Locale lang)
   {
      ...
   }

}
```



```
@Path("/orders")
public class OrderResource {

   @GET
   @Path("/{order-id}")
   @Produces("application/xml", "application/json")
   Order getOrder(@PathParam("order-id") int id,
                  @HeaderParam("ETag") String tag,
                  @QueryParam("lang") Locale lang)
   {
      ...
   }

}
```



Other Goodies

- Marshallers and Unmarshallers
- Response object for complex responses
- ExceptionMapper for exception handling
- Other APIs to help process HTTP

JAX-RS 2.0

RESTful Java Evolves

JAX-RS 2.0

- Asynchronous HTTP
- Client API
- Filter/Interceptor API
- Other minor goodies…

Asynchronous HTTP

- “Pushing” events to the browser
  - Really just blocking AJAX HTTP clients
- Servlets generally thread/request
  - 1000 blocking connections, 1000 threads
- Async HTTP
  - Detaching request thread from response
  - Different thread can service response
    - 1 thread can service multiple responses

Async HTTP

- ExecutionContext interface
  - Similar to Servlet AsyncContext
  - Manually suspend HTTP requests
  - Manually resume HTTP response in separate thread
- @Suspend annotation
  - Shortcut for ctx.suspend();


```
@Path("/queue")
public class QueueResource {

   @GET
   @Suspend
   @Produces("application/json")
   public void pull(@Context ExecutionContext ctx)
   {
      queue.put(ctx);
   }

}
```


```
public class QueueProcessor {

  public void process()
  {
    ExecutionContext ctx = queue.poll();
    ctx.resume(message);
  }

}
```

Client API

- “Fluent” low-level API
- Re-use content handlers (MessageBodyReader/Writer)
- Sync, async, and callback styles


```
Client client = ClientFactory.newClient();
```

```
Order order = client.target("http://shop.com")
                    .request("application/json")
                    .get(Order.class);
```


```
Response res = client.target("http://shop.com")
                     .request()
                     .post(Entity.json(order));
```

Client API

- Client interface
  - Connection management
- WebTarget
  - URI and URI Template abstraction
  - Can register specific properties and features


```
WebTarget uri =
          client.target("http://shop.com/orders/{id}");

uri.register(new CookieCache());
uri.register(new ResponseCache());
uri.setProperty(USERNAME, "bill");
uri.setProperty(PASSWORD, "geheim");
uri.setProperty(AUTH_TYPE, BASIC);
```

Async Client API

- Future
- Callback interfaces


```
Future<Order> future =
          client.target("http://shop.com")
                .request()
                .async()
                .get(Order.class);

Order order = future.get();
```


```
class MyCall implements InvocationCallback<Order>
{
  public void completed(Order order) {
    …
  }
  public void failed(InvocationException err) {
    …
  }
}
```


```
Future<Order> order =
          client.target("http://shop.com")
                .request()
                .async()
                .get(new MyCall());
```

Filters and Interceptors

Filters and Interceptors

- For framework and advanced developers
- Works in both async and synchronous environments
- Use-case driven

Filters and Interceptors

- Filters
  - Modify request/response object headers, codes, etc.
- Interceptors
  - Modify entities
  - Wrap around MessageBodyReader/Writer

Server Side

- ContainerRequestFilter
- ContainerResponseFilter
- ReaderInterceptor
- WriterInterceptor

ContainerRequestFilter

- Modify/pre-process incoming request
- Pre or Post resource method match
- Can abort the request and return a response

ContainerRequestFilter


```
for (ContainerRequestFilter filter : preMatch) {

    filter.filter(context);
    if (isAborted(filter)) {
       return getAbortedResponse(filter);
}

doResourceMethodMatch();

for (ContainerRequestFilter filter : postMatch) {

    filter.filter(context);
    if (isAborted(filter)) {
       return getAbortedResponse(filter);
}
```

ContainerRequestFilter

- Pre Match
  - Happen before resource method matched
  - Any time you want to change how request is matched
- Examples
  - HTTP Method overrides
    - (When client doesn't support DELETE/PUT)
  - Content negotiation via file suffix (.txt, .json, .xml)
  - Server-side cache


```
@Provider
class HttpMethod implements ContainerRequestFilter
{
  public void filter(ContainerRequestContext ctx) {
    if (ctx.getHeaders().containsKey("HTTP-METHOD") {


    ctx.setMethod(
       ctx.getHeaders().getFirst("HTTP_METHOD"));
    }
  }
}
```

ContainerRequestFilter

- Post Match
  - After resource method matched
- Examples
  - Custom annotation-driven authorization (@RolesAllowed)


```
@Provider
@PostMatch
class Auth implements ContainerRequestFilter
{
  public void filter(ContainerRequestContext ctx) {
    if (isMethodAuthorized(ctx) == false) {
       ctx.abortWith(Response.status(401).build());
    }
  }

  boolean isMethodAuthorized(…) {…}
}
```

ContainerResponseFilter

- Modify post-process outgoing response
- After resource method, before marshalling
- Examples
  - Header decoration (cache-control, ETag, Last-Modified)
  - Digitally signed headers (DKIM, DOSETA)

Reader/WriterInterceptor

- Wrap around a MessageBodyReader or Writer execution
- Examples:
  - GZIP encoding/decoding
  - multipart/signed signing and verification
  - multipart/encrypted encrypt and decrypt
  - Post marshalling processor (i.e. to add links, etc.)


```
@Provider
class Gzip implements WriterInterceptor
{
  public void aroundWriteTo(
       WriterInterceptorContext ctx) {


    GZipOutputStream gzip =
       new GZipOutputStream(ctx.getOutputStream());

    ctx.setOutputStream(gzip);

    ctx.proceed(); // call next interceptor or writer


  }
}
```

Client Side

- ClientRequestFilter
- ClientResponseFilter
- Reader/WriterInterceptor

Client Cache Example

ClientRequestFilter

- Check if URI cached already
- Make GET conditional
  - Add If-None-Match header with cached ETag value

ClientResponseFilter

- On 304, Not-Modified
- Change response code to 200
- Add relevant cached headers
- Point entity to internal cache entry

ReaderInterceptor

- Check desired Java type
- If cached Java object of that type return it
- Otherwise, invoke MessageBodyReader with raw cache bytes

Precedence

- @BindingPriority annotation, numeric based
  - SECURITY
  - HEADER_DECORATOR
  - ENCODER
  - DECODER
  - USER

Server-side Binding

- Executed every request by default
- ContainerRequestFilter prematch by default
  - @PostMatch annotation
- 2 options to bind per method

DynamicBinder

- User driven bindings
- DynamicBinder
  - Does a filter or interceptor bind to this method?


```
interface DynamicBinder<T> {

  public T getBoundProvider(ResourceInfo info);

}
```

@NameBinding

- Like CDI interceptor annotation bindings
- Triggered by annotation on resource method


```
@NameBinding
@interface Signed {

}
```


```
@Path("/orders")
interface MyResource {

   @GET
   @Signed
   public Order getOrder();

}
```


```
@Signed
class Signer implements WriterInterceptor
{
   …
}
```

JAX-RS 2.0 Summary

- Asynchronous HTTP Server-Side
- Client API
  - Low-level
  - Async and Callbacks available
- Filter and Interceptor API
  - Works in async and sync environments
  - Complex, but use-case driven

For more information

- JSR 339
- jboss.org/resteasy

- O'Reilly Books:
- "RESTful Java" by me
- "RESTful Web Services Cookbook"
- "RESTful Web Services"