

Scaling RESTful Services With JAX-RS

Bill Burke

Fellow

JBoss, a division of Red Hat

bburke@redhat.com



Agenda

- Caching
- Encoding
- Concurrency
- Asynchronous HTTP
- Asynchronous Jobs

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
- Published author
 - ✓ Books, articles

Caching

Cache is King

Web Caching

- Why?
 - ✓ To reduce latency
 - ✓ To reduce network traffic
- Who?
 - ✓ Browsers
 - ✓ Content Delivery Networks (CDNs)
 - Akamai, Limelight

HTTP Caching Features

- Caching: allowed or not allowed
- Expiration
- Intermediary caches allowed or not (CDNs)
- Validation
 - ✓ Conditional GETs
- Storable or not Storable
- Extendible caching semantics

Knowing When to Cache?

- How does a browser/CDN know when to cache?
 - ✓ Expires header
 - ✓ Cache-Control header
 - ✓ Validation Headers
 - Last-Modified
 - Etag
 - ✓ If none of these set, don't cache

Expires Header

- Set by the server
- Simple date into the future that cache expires
- Deprecated way of controlling cache semantics

Expires: Tue, 17 Mar 2009 16:00 GMT

Cache-Control Header

- Preferred way of setting cache parameters
- Rich set of functionality
- Used by both server and client (request/response) to set caching semantics
- Always takes precedence over Expires

Cache-Control Response Directives

- public - cacheable by anybody
- private - no CDNs allowed, only client should cache
- no-cache - don't cache ever
- max-age - time in seconds the cache is valid
- s-maxage - time in seconds a shared cache valid (CDN)
- no-store - don't persist cache entries
- no-transform - don't transform into a different type

Cache-Control Response Directives

- `must-revalidate` - client should not hold/use stale entries
- `proxy-revalidate` - CDN should not hold/use stale entries

`Cache-Control: no-cache`

`Cache-Control: private, max-age=3000`

JAX-RS and Cache-Control

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

    @Path("/{id}")
    @GET
    @Produces("application/xml")
    public Response getOrder(@PathParam("id") int id) {
        ...
        CacheControl cc = new CacheControl();
        cc.setMaxAge(3000);
        return Response.ok(order)
            .cacheControl(cc).build();
    }
}
```

RESTEasy and Cache-Control

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

    @Path("/{id}")
    @GET
    @Produces("application/xml")
    @Cache(maxAge=3000)
    public Order getOrder(@PathParam("id") int id) {
        ...
    }

    @Path("/{id}")
    @GET
    @Produces("application/xml")
    @NoCache
    public Order getOrder(@PathParam("id") int id) {
        ...
    }
}
```

Only on 200, OK response

Validation Headers

- When cache is stale, client can ask server if cache still valid
- To be able to revalidate client needs additional headers beyond Cache-Control from a server response
 - ✓ Last-Modified - a date when the resource was last modified
 - ✓ ETag - a unique hash-like key that identifies a version of the resource
- Client should cache these headers along with response body

Validation Headers and Conditional GETs

- When client determines cache is stale it does a GET with one or both of these headers set
- If-Modified-Since
 - ✓ With value of server's Last-Modified header
- If-None-Match
 - ✓ With value of server's Etag header

Validation Headers and Conditional GETs

- Server examines If-Modified-Since and/or If-None-Match headers
- Is the client cache still valid?
 - ✓ Send a 304, "Not Modified" response code with no body
 - ✓ Optionally send updated Last-Modified and Cache-Control response headers
- Is the client cache not valid anymore?
 - ✓ Send 200, "OK" with new resource body
 - ✓ Optionally send new Last-Modified, Etag, and/or Cache-Control response headers

JAX-RS and Validation

**Return null if conditions not met
Return a builder with response
code set to 304**

```
public interface Request {  
  
    ...  
  
    ResponseBuilder evaluatePreconditions(EntityTag eTag);  
  
    ResponseBuilder evaluatePreconditions(  
        Date lastModified);  
  
    ResponseBuilder evaluatePreconditions(  
        Date lastModified, EntityTag eTag);  
}
```

JAX-RS and Validation

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

    @Path("/{id}")
    @GET
    @Produces("application/xml")
    public Response getOrder(@PathParam("id") int id,
                            @Context Request request) {
        EntityTag tag = ... get new app-specific tag;
        ResponseBuilder builder = null;
        builder = request.evaluatePreconditions(tag);
        CacheControl cc = new CacheControl();
        cc.setMaxAge(3000);
        if (builder != null)
            return builder.cacheControl(cc).build();

        ... get order ...
        return Response.ok(order)
                       .cacheControl(cc).build();
    }
}
```

Caching Facilities

RESTEasy Client Cache

- Acts like a browser minus persistence
 - ✓ In memory only
- Does validation and conditional gets
- Sharable "Browser" cache instances
- Works with raw request or proxy framework

RESTEasy Client Cache

```
@Path("/orders")
public interface OrderServiceProxy {

    @GET
    @Produces("application/xml")
    @Path("/{id}")
    Order getOrder(@PathParam("id") int id);
}

...

OrderServiceProxy proxy = ProxyFactory.create(
    "http://example.com");
BrowserCache cache = CacheFactory.makeCacheable(proxy);

// proxy instance now will cache responses and do
// conditional GETs
```

Squid

- Caching proxy server (not JBoss)
 - ✓ www.squid-cache.org
- Supports caching HTTP, HTTPS, FTP and more
- Advanced Content Routing and Load Balancing
- Very popular

RESTEasy Server Cache

- Local in-memory cache
 - ✓ Sits in front of JAX-RS service
 - ✓ Caches marshalled data
 - ✓ If service sets Cache-Control header with a max age
 - Response will be cached
- Update Cache-Control header on cache hit
- Automatically generates Etag headers
 - ✓ Md5 hash of content
- Handles Cache Validation automatically
- If client sends Etag and server cache is stale
 - ✓ Invoke service
 - ✓ Md5 hash the data
 - ✓ If same as If-None-Match, return NOT MODIFIED
 - ✓ Recache the response

Caching Conclusion

- HTTP has rich caching semantics
 - ✓ Cache expiration
 - ✓ Browser and CDN controls
 - ✓ Persistence controls
 - ✓ Revalidation
- JAX-RS provides some help
 - ✓ CacheControl object
 - ✓ Revalidation methods
- Caching Infrastructure and Tools available
 - ✓ Browser
 - ✓ Squid
 - ✓ RESTEasy Client and Server Cache

Concurrent Updates

Conditional PUTs and POSTs

- HTTP has protocols for managing concurrent or stale writes
- Can prevent client from modifying and updating a stale copy of a resource
- Like cache validation relies on:
 - ✓ Last-Modified
 - ✓ ETag

Conditional PUTs and POSTs

- When doing a PUT or POST client sends these request headers
 - ✓ If-Unmodified-Since - value of Last-Modified
 - ✓ If-Match - value of Etag
- If the resource hadn't been modified since or matches ETag value
 - ✓ Perform the update
- If the resource has been modified or doesn't match ETag
 - ✓ Send a 412, "Precondition Failed" response code

JAX-RS and Condition PUTs

```
public interface Request {
```

```
...
```

```
    ResponseBuilder evaluatePreconditions(EntityTag eTag);
```

```
    ResponseBuilder evaluatePreconditions(  
                                           Date lastModified);
```

```
    ResponseBuilder evaluatePreconditions(  
                                           Date lastModified, EntityTag eTag);
```

```
}
```

**Return null if conditions met
Return a builder with response
code set to 412**

JAX-RS and Conditional PUT

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

    @Path("/{id}")
    @PUT
    @Produces("application/xml")
    public Response updateOrder(@PathParam("id") int id,
                                Order order,
                                @Context Request request) {
        EntityTag tag = ... get current app-specific tag;
        ResponseBuilder builder = null;
        builder = request.evaluatePreconditions(tag);

        if (builder != null)
            return builder.build();

        ... update order ...
        return Response.noContent().build();
    }
}
```

Content Encoding

Shrinkage!

What is Content Encoding

- HTTP allows you to encode a message body
- Server/client can compress
 - ✓ Saves on network bandwidth
 - ✓ A simple GZIP usually supported by browsers
- Content-Encoding header specifies encoding
 - ✓ Content-Encoding: gzip
- Client can negotiate encoding
 - ✓ Accept-Encoding: gzip, deflate

RESTEasy GZIP Support

- No portable way to support encoding in JAX-RS
 - ✓ Only vendor specific ways
- Automatic, transparent support for GZIP decoding
 - ✓ Client and server looks at/for Content-Encoding header
- Client framework automatically sets Accept-Encoding
 - ✓ Accept-Encoding: gzip, deflate
 - ✓ Appends gzip to existing Accept-Encoding header too
- Client/Server automatically encodes if request/response has Content-Encoding: gzip set
- @GZIP annotation simple, fast way to encode message bodies

RESTEasy GZIP support

```
@Path("/orders")
public interface OrderServiceProxy {

    @POST
    @Consumes("application/xml")
    void createOrder(@GZIP Order order);
}

@Path("/orders")
public class OrderService {

    @GET
    @GZIP
    @Produces("application/xml")
    @Path("/{id}")
    Order getOrder(@PathParam("id") int id) {...}
}
```

Encoding Conclusion

- HTTP can negotiate message compression
- JAX-RS has no portable way of doing GZIP encoding
- RESTEasy does automatic compression/decompression

Asynchronous HTTP

Haley's Comet

The Thread per Connection Problem

- Blocking clients
 - ✓ Chat
 - ✓ Quotes
 - ✓ Any application that pushes to clients
- Client does a GET and blocks until server ready to send a response
 - ✓ High concurrent connections to web server
- Tomcat marries thread and connection
 - ✓ Threads can be heavyweight. Consume tons of memory (stack)

The Solution

- Detach response processing from request thread
- Suspend request
- A different thread responsible for sending response

Asynchronous HTTP Providers

- Jetty 6
 - ✓ Control-flow throw exceptions
- Tomcat 6 Comet API
 - ✓ Event based and detachable
 - ✓ Buggy as hell
- JBoss Web
 - ✓ Forked Tomcat Comet API
 - ✓ Stable and functional
 - ✓ Requires native plugin
- Servlet 3.0
 - ✓ Still in flux
 - ✓ Jetty 7 pre-release

When to use Asynchronous HTTP

- Don't need if server not "pushing" data
- Don't need if not a lot of concurrent connections
- Might actually hurt performance
 - ✓ Too much context switching
- Have to rewrite servlet filters

RESTEasy Asynchronous HTTP Abstraction

- Suspends and detaches response processing
- Simple API
- Current abstractions for
 - ✓ Tomcat 6 Comet API (tomcat 6 is very buggy!)
 - ✓ JBossWeb (requires native plugin currently, but quite stable)
 - ✓ Servlet 3.0 (Jetty 7 pre 5 only implementation currently)

RESTEasy Asynchronous HTTP Abstraction

```
public interface AsynchronousResponse {  
    void setResponse(Response response);  
}  
  
public @interface Suspend {  
    long value() default -1;  
}
```

RESTEasy Asynchronous HTTP Abstraction

```
@GET
@Produces("text/plain")
public void getBasic(
    @Suspend(100000) AsynchronousResponse response)
{
    Thread t = new Thread() {
        public void run() {
            try {
                Thread.sleep(5000);
                Response jaxrs = ...
                response.setResponse(jaxrs);
            } catch (Exception ignore) {
            }
        }
    };
    t.start();
}
```

Comet vs. HTTP

- Comet
 - ✓ Uses HTTP solely to set up connection
 - ✓ Proprietary protocols
 - ✓ Never ends request
 - ✓ A little faster because no dispatching
- Pure HTTP
 - ✓ Can still to suspending and async processing
 - ✓ Client uses standard HTTP protocol
 - ✓ Client is never aware of asynchronicity
 - ✓ The thread-per-connection problem still solved

Asynchronous HTTP Conclusion

- Solves the Thread-per-connection problem
 - ✓ Many concurrent blocking clients
- Use sparingly
- Many providers
 - ✓ Standardized under Servlet 3.0
- RESTEasy provides a vendor abstraction
- Minimal performance benefits with COMET APIs

Asynchronous Request Processing

Asynchronicity over a Synchronous Protocol

HTTP and ACCEPTED

- Although a synchronous protocol, does have idea of asynchronous processing
- Server is allowed to send a 202, "Accepted" response
 - ✓ Request was received but not processed yet
- A design pattern
 - ✓ Server sends 202 response code
 - ✓ Server sends a Location header
 - Location header is an HTTP redirect
 - Location header has a URI that will hold our response

Building Async JAX-RS Service

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

    @GET
    @Path("/jobs/{id}")
    @Produces("application/xml")
    public Response getJob(@PathParam("id") int id) {

        Job job = getProcessedJob(order);
        return job.response();
    }
}
```

RESTEasy Asynchronous Job Service

- Any invocation can be made asynchronous
 - ✓ `uri?asynch=true` - creates a job
 - ✓ `uri?oneway=true` - fire and forget
- Returns a Location that can be viewed and deleted
 - ✓ GET and DELETE
 - ✓ `/jobs/{job-id}?wait={time}&nowait=true`
 - ✓ Returns 410, "Gone" if job doesn't exist anymore
 - ✓ Returns 202, Accepted if job exists but isn't complete

RESTFul side effects

- POST really only method you can make async
- GET, DELETE, PUT are idempotent
 - ✓ Yes doesn't change state of resource with duplicate calls
 - ✓ BUT, have side effect of creating a resource

Conclusion

- HTTP has a lot of built in performance features
 - ✓ Caching
 - ✓ Encoding
 - ✓ ETags
- Asynchronous HTTP has its place
- Leverage the WEB!

JAX-RS Implementations

- JBoss RESTEasy
 - ✓ <http://jboss.org/resteasy>
 - ✓ Embeddable
 - ✓ Spring and EJB integration
 - ✓ Client Framework
 - ✓ Asynchronous HTTP abstractions
- Jersey
 - ✓ Sun reference implementation
 - ✓ WADL support
- Apache CXF
- RESTlet

References

- Links
 - ✓ <http://jsr311.dev.java.net/>
 - ✓ <http://jboss.org/resteasy>
 - ✓ <http://www.infoq.com/articles/rest-introduction>
 - ✓ <http://www.infoq.com/articles/tilkov-rest-doubts>
 - ✓ <http://rest.blueoxen.net/>
 - ✓ <http://java.dzone.com/articles/intro-rest>
 - ✓ <http://architects.dzone.com/articles/putting-java-rest>
- Books:
 - ✓ Coming this summer "RESTful Java" by me
 - ✓ O'Reilly's "RESTful Web Services"
 - <http://oreilly.com/catalog/9780596529260/>

Questions