

JSR 311: JAX-RS

Java API for RESTful Web services

Paul Sandoz
Jakub Podlesak
Sun Microsystems
1920

JAZOON07

THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY
JUNE 24 - 28, 2007 ZÜRICH



AGENDA

- > REST Primer
- > Why a Java Platform RESTful API?
- > API elements
- > Deployment Options
- > Open Issues
- > Reference Implementation
- > Demos

REST Primer v1: read this book!



REST Primer v2: REST Tenets

- > Resources
 - Identified by URIs
 - Realized as representations
- > Methods
 - Uniform (small) set for all resources
 - Involve exchange of representations
- > Representations
 - Embody application state
 - Contain URIs to related resources
 - Multiple alternate representations per resource

RESTful APIs

- > Lots of Web companies now offering RESTful APIs for their services
- > Where both WS-* and RESTful API offered RESTful API more widely used
 - RESTful APIs are easier to consume, especial with scripting languages
 - Browser-based experimentation is easy
- > Current platform API for building RESTful Web services are rather low-level
 - Many opportunities for simplifying development

Example

- > Music Collection
 - /music/artists
 - **/music/artists/{id}**
 - /music/recordings
 - /music/recordings/{id}
 - **/music/artists/{id}/recordings**
 - /music/genre/{id}
 - /music/format/{id}
- > XML and JSON support

Artist Resource Using Servlet API

```

public class Artist extends HttpServlet {

    public enum SupportedOutputFormat {XML, JSON};

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String accept = request.getHeader("accept").toLowerCase();
        String acceptableTypes[] = accept.split(",");
        SupportedOutputFormat outputType = null;
        for (String acceptableType: acceptableTypes) {
            if (acceptableType.contains("*/") || acceptableType.contains("application/*") ||
                acceptableType.contains("application/xml")) {
                outputType=SupportedOutputFormat.XML;
                break;
            } else if (acceptableType.contains("application/json")) {
                outputType=SupportedOutputFormat.JSON;
                break;
            }
        }
        if (outputType==null)
            response.sendError(415);
        String path = request.getPathInfo();
        String pathSegments[] = path.split("/");
        String artist = pathSegments[1];
        if (pathSegments.length < 2 && pathSegments.length > 3)
            response.sendError(404);
        else if (pathSegments.length == 3 && pathSegments[2].equals("recordings")) {
            if (outputType == SupportedOutputFormat.XML)
                writeRecordingsForArtistAsXml(response, artist);
            else
                writeRecordingsForArtistAsJson(response, artist);
        } else {
            if (outputType == SupportedOutputFormat.XML)
                writeArtistAsXml(response, artist);
            else
                writeArtistAsJson(response, artist);
        }
    }

    private void writeRecordingsForArtistAsXml(HttpServletRequest response, String artist) { ... }

    private void writeRecordingsForArtistAsJson(HttpServletRequest response, String artist) { ... }

    private void writeArtistAsXml(HttpServletRequest response, String artist) { ... }

    private void writeArtistAsJson(HttpServletRequest response, String artist) { ... }
}

```

There Must Be a Better Way!

- > High-level
 - > Declarative
 - > Clear mapping to REST concepts
 - > Makes it easy to do the right thing
 - > Takes care of “boilerplate” code
 - > Graceful fallback to lower-level APIs when required
-
- > **DISCLAIMER:** Early in Java Specification Request (JSR) process, everything from here is **liable to change!**

Artist Resource Using Servlet API

```

public class Artist extends HttpServlet {

    public enum SupportedOutputFormat {XML, JSON};

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String accept = request.getHeader("accept").toLowerCase();
        String acceptableTypes[] = accept.split(",");
        SupportedOutputFormat outputType = null;
        for (String acceptableType: acceptableTypes) {
            if (acceptableType.contains("*/") || acceptableType.contains("application/*") ||
                acceptableType.contains("application/xml")) {
                outputType=SupportedOutputFormat.XML;
                break;
            } else if (acceptableType.contains("application/json")) {
                outputType=SupportedOutputFormat.JSON;
                break;
            }
        }
        if (outputType==null)
            response.sendError(415);
        String path = request.getPathInfo();
        String pathSegments[] = path.split("/");
        String artist = pathSegments[1];
        if (pathSegments.length < 2 && pathSegments.length > 3)
            response.sendError(404);
        else if (pathSegments.length == 3 && pathSegments[2].equals("recordings")) {
            if (outputType == SupportedOutputFormat.XML)
                writeRecordingsForArtistAsXml(response, artist);
            else
                writeRecordingsForArtistAsJson(response, artist);
        } else {
            if (outputType == SupportedOutputFormat.XML)
                writeArtistAsXml(response, artist);
            else
                writeArtistAsJson(response, artist);
        }
    }

    private void writeRecordingsForArtistAsXml(HttpServletResponse response, String artist) { ... }

    private void writeRecordingsForArtistAsJson(HttpServletResponse response, String artist) { ... }

    private void writeArtistAsXml(HttpServletResponse response, String artist) { ... }

    private void writeArtistAsJson(HttpServletResponse response, String artist) { ... }
}

```

Automated Content Negotiation

```

public class Artist extends HttpServlet {

    @ProduceMime("application/xml")
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String path = request.getPathInfo();
        String pathSegments[] = path.split("/");
        String artist = pathSegments[1];
        if (pathSegments.length < 2 || pathSegments.length > 3)
            response.sendError(404);
        else if (pathSegments.length == 3 &&
            pathSegments[2].equals("recordings"))
            writeRecordingsForArtistAsXml(response, artist);
        else
            writeArtistAsXml(response, artist);
        }
    }
    ...
}

```

URI Templates

```

@UriTemplate("/artists/{id}")
public class Artist extends HttpServlet {

    @ProduceMime("application/xml")
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String path = request.getPathInfo();
        String pathSegments[] = path.split("/");
        String artist = pathSegments[1];
        ...
    }
    ...
}

```

POJO

```
@UriTemplate("/artists/{id}")
public class Artist {

    @ProduceMime("application/xml")
    @HttpMethod
    InputStream getXml(
        @UriParam("id") String artist) {
        ...
    }

    ...
}
```

Artist Resource as a POJO

```

@UriTemplate("/artists/{id}")
@ProduceMime("application/xml")
public class Artist {

    @HttpMethod
    InputStream getXml(@UriParam("id") String artist) { ... }

    @HttpMethod @ProduceMime("application/json")
    InputStream getJson(@UriParam("id") String artist) { ... }

    @HttpMethod @UriTemplate("recordings")
    InputStream getRecordingsXml(@UriParam("id") String artist) { ... }

    @HttpMethod @UriTemplate("recordings") @ProduceMime("application/json")
    InputStream getRecordingsJson(@UriParam("id") String artist) { ... }
}

```

Artists Resource as a DRYer POJO

```

@UriTemplate("/artists/{id}")
@ProduceMime("application/xml")
public class Artist {

    String artist;

    Artist(@UriParam("id") String artist) { this.artist = artist; }

    @HttpMethod
    InputStream getXml() { ... }

    @HttpMethod @ProduceMime("application/json")
    InputStream getJson() { ... }

    @HttpMethod @UriTemplate("recordings")
    InputStream getRecordingsXml() { ... }

    @HttpMethod @UriTemplate("recordings") @ProduceMime("application/json")
    InputStream getRecordingsJson() { ... }

}

```

Artist Resource Using Servlet API

```

public class Artist extends HttpServlet {

    public enum SupportedOutputFormat {XML, JSON};

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String accept = request.getHeader("accept").toLowerCase();
        String acceptableTypes[] = accept.split(",");
        SupportedOutputFormat outputType = null;
        for (String acceptableType: acceptableTypes) {
            if (acceptableType.contains("*/") || acceptableType.contains("application/*") ||
                acceptableType.contains("application/xml")) {
                outputType=SupportedOutputFormat.XML;
                break;
            } else if (acceptableType.contains("application/json")) {
                outputType=SupportedOutputFormat.JSON;
                break;
            }
        }
        if (outputType==null)
            response.sendError(415);
        String path = request.getPathInfo();
        String pathSegments[] = path.split("/");
        String artist = pathSegments[1];
        if (pathSegments.length < 2 && pathSegments.length > 3)
            response.sendError(404);
        else if (pathSegments.length == 3 && pathSegments[2].equals("recordings")) {
            if (outputType == SupportedOutputFormat.XML)
                writeRecordingsForArtistAsXml(response, artist);
            else
                writeRecordingsForArtistAsJson(response, artist);
        } else {
            if (outputType == SupportedOutputFormat.XML)
                writeArtistAsXml(response, artist);
            else
                writeArtistAsJson(response, artist);
        }
    }

    private void writeRecordingsForArtistAsXml(HttpServletRequest response, String artist) { ... }

    private void writeRecordingsForArtistAsJson(HttpServletRequest response, String artist) { ... }

    private void writeArtistAsXml(HttpServletRequest response, String artist) { ... }

    private void writeArtistAsJson(HttpServletRequest response, String artist) { ... }
}

```

Prescriptive approach to design

- > Resources: what are the URIs?
 - `UriTemplate`
- > Methods: what are the HTTP methods?
 - `HttpMethod`
- > Representations: what are the formats?
 - `ConsumeMime`
 - `ProduceMime`

Declaring Resource Class Methods

- > Annotate with `@HttpMethod`
- > Java method name can be anything
- > Default HTTP method taken from the Java method name
 - Only for GET, POST, PUT, DELETE, HEAD

`@HttpMethod`

```
public XXX getXXX()
```

`@HttpMethod("GET")`

```
public XXX find()
```

Method Return Types

> Response

- for common types of response: redirection, resource creation
- uses builder pattern devised by Josh Bloch

@HttpMethod

```
public Response postArtist() {  
    Recording recording = ...  
    URI location = ...  
    return Response.Builder.  
        created(recording, location).build();  
}
```

Method Return Types

> Response

- for common types of response: redirection, resource creation
- uses builder pattern devised by Josh Bloch

> T

- when a default media be determined
- T extensible via SPI

@HttpMethod

```
public Recording getRecording()
```

Method Parameters

- > Multiple, annotated with `@UriParam`, `@QueryParam`, `@HeaderParam`, `@MatrixParam`
 - Flexible typing
- > Zero or one of type `T` or `Entity<T>`
 - `T` extensible via SPI

```
@HttpMethod("PUT")
```

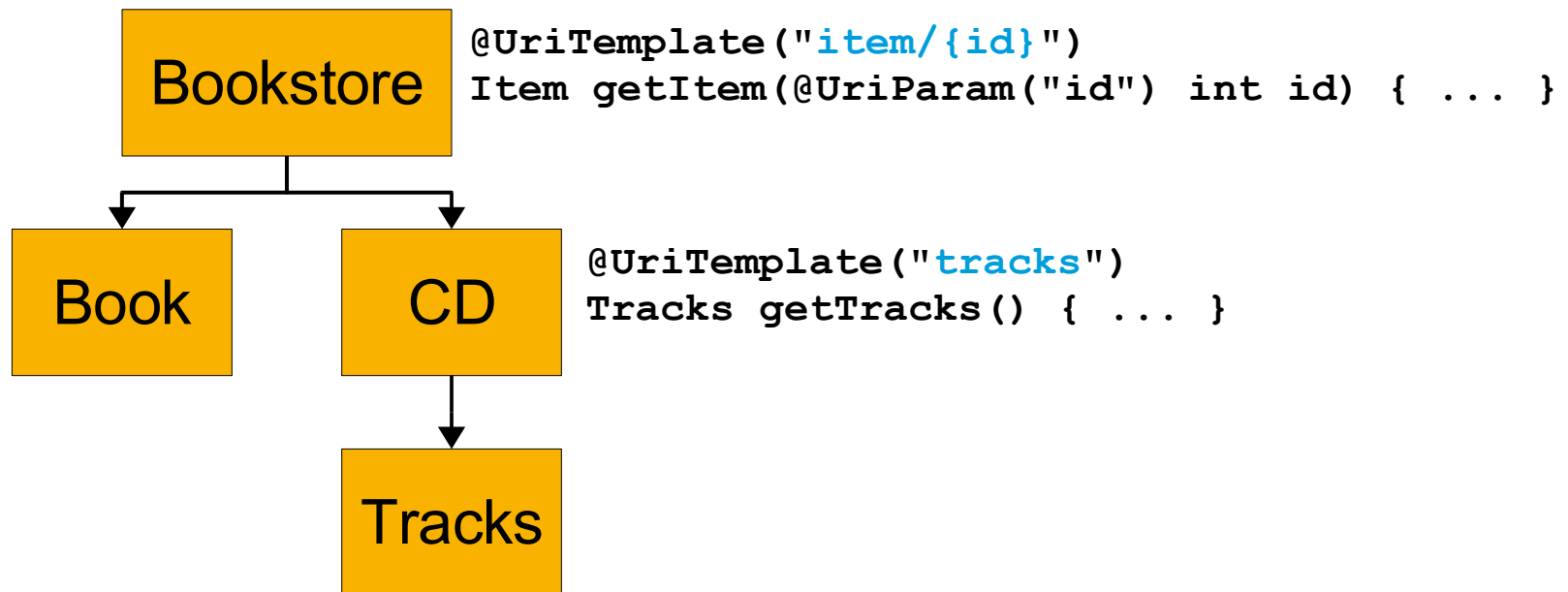
```
public Recording updateRecording(
    @UriParam("id") String id,
    Recording updatedRecording)
```

Mapping URI path hierarchy to tree of Resource classes

- > Static: annotate Java class with `@SubResources`
 - reference subordinate Resource classes
- > Dynamic: annotate Java method with `@UriTemplate`
 - parent resource responsible for returning an instance of a subordinate Resource class
 - supports polymorphic resources

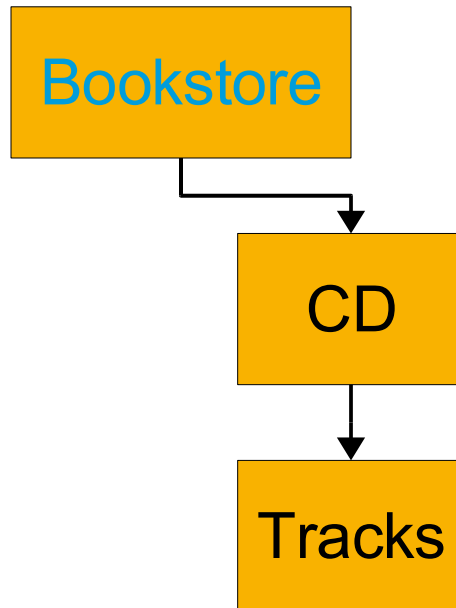
Mapping URI path hierarchy to tree of Resource classes

- > / → Bookstore class
- > /item/{id} → Book or CD class (inherit from Item class)
- > /item/{id}/tracks → Tracks class



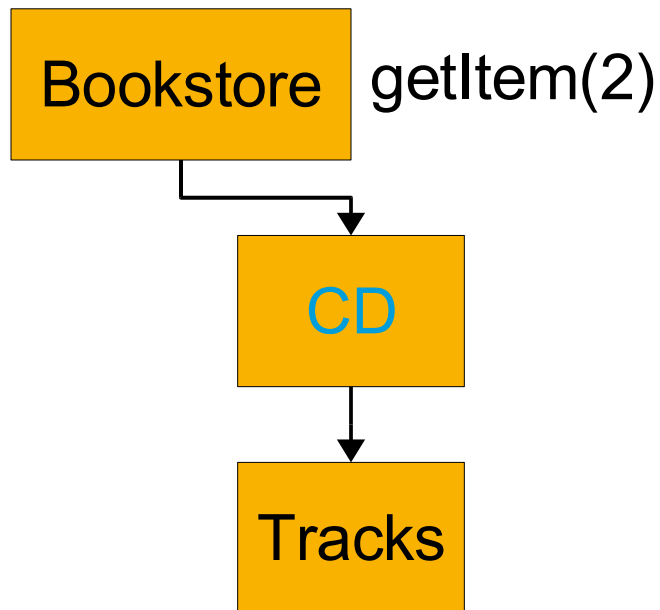
/item/2/tracks

> / → instance of **Bookstore** class



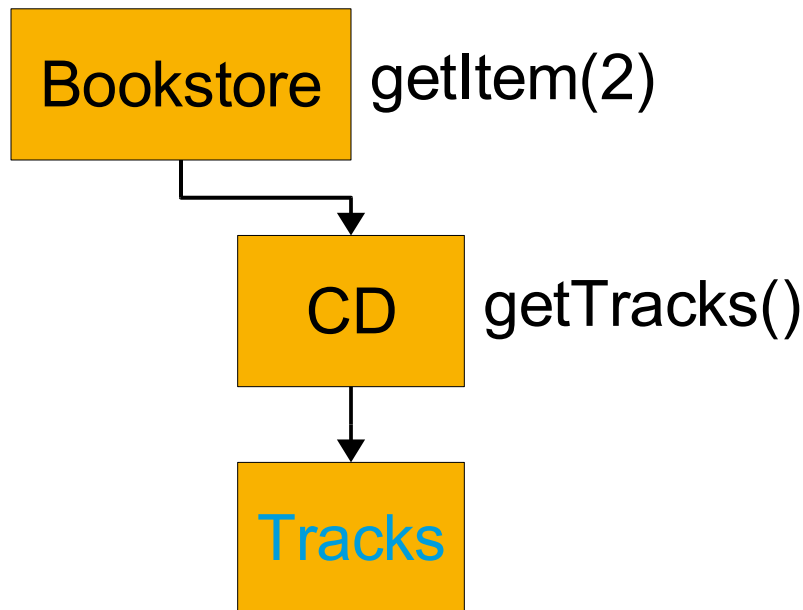
/item/2/tracks

- > / → instance of Bookstore class
- > **item/2** → instance of CD class



/item/2/tracks

- > / → instance of Bookstore class
- > item/2 → instance of CD class
- > tracks → instance of Track class



Deployment Options

- > Servlet and Java API for XML Web Services (JAX-WS) Provider required by JSR
- > Sun RI adds support for Grizzly and Java Platform, Standard Edition 6 (Java SE 6 platform) Lightweight HTTP Server
- > Restlet support expected
- > SPI for supporting other containers

Open Issues

- > URI path hierarchy to Resource object model
 - Requires more discussion
- > Resource life-cycle
 - Per-request, per-session, singleton
- > Container vs. API support
 - Security
 - Filters/Guards
- > Other language support
 - Annotation use difficult in JavaScript, JRuby, etc.
 - Lower-level APIs start to lose benefits

Reference Implementation: Jersey

- > <http://jersey.dev.java.net>
- > Build 0.1 now available
- > **Warning:** API and implementation will change as expert group progresses

Demos

- > Tooling with NetBeans
 - RESTful Web service facade of a database
 - Using R2 SWDP
- > Atom server
 - Using reference implementation

Summary

- > High-level declarative programming model
- > REST concepts reflected in the API
- > Flexible typing, runtime takes care of common conversions
- > SPIs to extend capabilities
 - Pluggable support for types
- > Flexible deployment options

Links

- > JSR 311
 - <http://jcp.org/en/jsr/detail?id=311>
- > Reference Implementation
 - <http://jersey.dev.java.net/>
- > SWDP
 - <http://developers.sun.com/web/swdp/>
- > RESTful Web services
 - <http://www.oreilly.com/catalog/9780596529260/>
- > Atom Publishing Protocol
 - <http://bitworking.org/projects/atom/>

Paul.Sandoz@Sun.Com

Jakub.Podlesak@Sun.Com

Sun Microsystems

<http://blogs.sun.com/sandoz>

<http://blogs.sun.com/japod>