JAX-RS and Jersey

Paul Sandoz
JAX-RS co-spec lead and Jersey lead
mailto:Paul.Sandoz@Sun.Com
http://blogs.sun.com/sandoz
https://twitter.com/PaulSandoz/

Overview

- Terminology
- Information & Status
- Integration with Java EE 6
- Q/A

REST style

Set of constraints you apply to the architecture of a distributed system

RESTful Web services

REST constraints applied to the Web and construction of web services

JAX-RS

A Java API for helping build RESTful Web services

Jersey

The production quality Reference Implementation of JAX-RS... and more

JAX-RS: http://jsr311.dev.java.net

- JAX-RS 1.1 is final and part of EE 6
  - Not in Web profile, but included with GlassFish v3 Web profile
- JCP
  - http://jcp.org/en/jsr/detail?id=311
- API
  - https://jsr311.dev.java.net/nonav/releases/1.1/index.html
- Spec
  - https://jsr311.dev.java.net/nonav/releases/1.1/spec/spec.html
- JAX-RS 2.0?
  - Hypermedia, client API

Resource classes

```
@Path(“root”)
public class RootResource { // Per-request scope
  @Context UriInfo ui;

  @GET
  Public String get() { return “GET”; }

  @Path(“sub-resource”)
  public SubResource sub() {
    return new SubResource();
  }
}

public class SubResource {

  …
}
```

Jersey: http://jersey.dev.java.net

- Open source: CDDL/lGPLv2 license
- Version 1.1.4.1 implements JAX-RS 1.1
  - Distributed with GlassFish v3
- Version 1.0.3.1 implements JAX-RS 1.0
  - Distributed with GlassFish v2.1.1
- Stable releases ~ every 6 to 8 weeks
  - Published to GlassFish update center
- Support for JAX-RS 1.1 in NetBeans 6.8
- Send feedback to
  - mailto:users@jersey.dev.java.net

Integration with Java EE 6

- Servlet 3.0
- EJB 3.1
- Managed beans
- CDI managed beans

Servlet 3.0 integration

No Web.xml !

Servlet 3.0 integration

No Web.xml !
Or portable web.xml

Before

```
<web-app>
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>
      com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>com.foo.MyApplication</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/resources/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Before

```
package com.foo;

import javax.ws.rs.core.Application;
Import java.util.*;

public class MyApplication {
  @Override
  public Set<Class<?>> getClasses() {
    return new HashSet<Class<?>(
       Arrays.asList(
         AResource.class,
         AnotherResource.class));
  }
}

…

@Path(“one”) public class Aresource { … }

…

@Path(“two”) public class AnotherResource { … }
```

After: no web.xml

```
package com.foo;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
Import java.util.*;

@ApplicationPath("resources")
public class MyApplication {
  @Override
  public Set<Class<?>> getClasses() {
    return new HashSet<Class<?>(
       Arrays.asList(
         AResource.class,
         AnotherResource.class));
  }
}

…

@Path(“one”) public class Aresource { … }

…

@Path(“two”) public class AnotherResource { … }
```

Register all root resource and provider classes in war

```
package com.foo;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
Import java.util.*;

@ApplicationPath("resources")
public class MyApplication { }

…

@Path(“one”) public class Aresource { … }

…

@Path(“two”) public class AnotherResource { … }
```

Portable web.xml

```
<web-app>
  <servlet>
    <servlet-name>com.foo.MyApplication</servlet-name>
  </servlet>
  <servlet-mapping>
    <servlet-name>com.foo.MyApplication</servlet-name>
    <url-pattern>/resources/*</url-pattern>
  </servlet-mapping>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Protected</web-resource-name>
      <url-pattern>/resources/*</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
    ...
  </security-constraint>
</web-app>
```

EJB 3.1 integration

Use stateless or singleton beans in the WAR as resource and provider classes

EJB 3.1 integration

```
@Path(“stateless”)
@Stateless
public class MyStatelessRootResource {
  @Context UriInfo ui;

  @GET
  Public String get() { return “GET”; }

  @EJB MyStatelessResource subResource;

  @Path(“sub-resource”)
  public MyStatelessResource sub() {
    return subResource;
  }
}

@Singleton
public class MyStatelessResource {
  @Context UriInfo ui;

  …
}
```

Managed beans integration

Annotate with `@ManagedBean` to inject EE-based dependencies (or CDI dependencies, if CDI is enabled)

Managed beans integration

```
@Path(“root”)
@ManagedBean
// Instantiated by EE container or CDI container
// Managed by JAX-RS container in the per-request scope
public class RootResource {
  @Context UriInfo ui;

  @Resource(name="pi") double pi;

  …
}
```

Managed beans integration

- Require a no arg constructor
  - Constructor injection is not supported
- Require empty beans.xml in WEB-INF to enable CDI

CDI managed beans integration

Annotate with CDI scope annotations, such as `@RequestScoped`

CDI managed beans integration

```
@Path(“root”)
@ApplicationScoped
// Instantiated and managed by the CDI container
public class RootResource {
  @Context UriInfo ui;

  @Inject double pi;

  @Inject Provider<SubResource> subResource;

  @Path(“sub-resource”)
  public SubResource sub() {
    return subResource.get();
  }
}

@RequestScoped
public class SubResource {
  @Context UriInfo ui;

  …
}
```

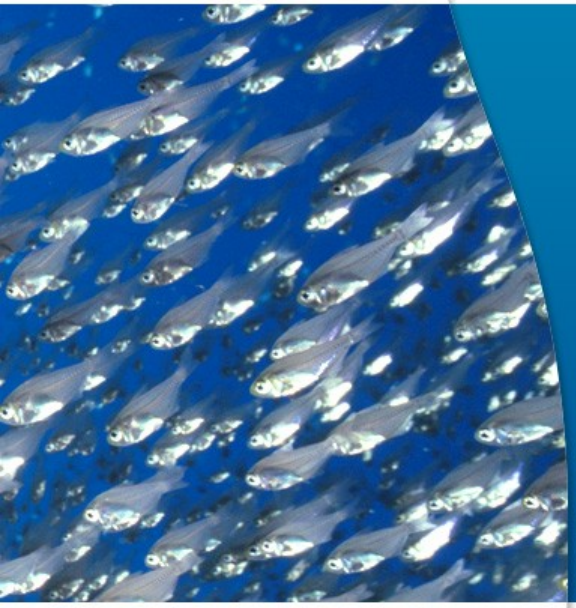
CDI managed beans integration

- @Inject does not work with JAX-RS/Jersey artifacts
- If JSR 330/299 had been around in late 2007 early 2008 then `@Context` might not exist!
- A general point on CDI: beware of client proxies and constructor injection

```
@RequestScoped
public class Foo {
  // no arg constructor required for client proxy
  public Foo() { this.mp = null; }

  private final MyDep mp;

  @Inject
  public Foo(MyDep mp) { this.mp = mp; }
}
```



Q&A