

RESTful webservices – Client API in Java - Jersey

Pavel Bucek - pavel.bucek@oracle.com

RESTful web services, Jersey

- Representational State Transfer – REST
- Resources accessed (identified) using URIs
 - `http://localhost:8080/contextRoot/resource`
- Stateless communication protocol – HTTP
 - Well defined operations: GET, PUT, POST, DELETE, ...
- Jersey – JAX-RS reference implementation in Java
 - Server side - `@Path`, `@GET`, `@Produces`, ...
 - Client side – jersey-client module
 - Compatible with other implementations

Accessing RESTful web services - possibilities

- No specification (yet – see JAX-RS 2.0)
- JDK – HttpURLConnection
- Jersey
- Apache CXF
- RESTEasy
- ...

Jersey Client – Why should I use it?

- Created as part of JAX-RS reference implementation
- Used internally for almost all tests
 - Production quality, stable & reliable API/implementation
- User base
- Supports XML and Json out of the box
- Jersey Test Framework integration

- Is used as a proof of concept for various new directions
 - Hypermedia
 - View based client

Jersey Client vs HttpURLConnection

a) HttpURLConnection

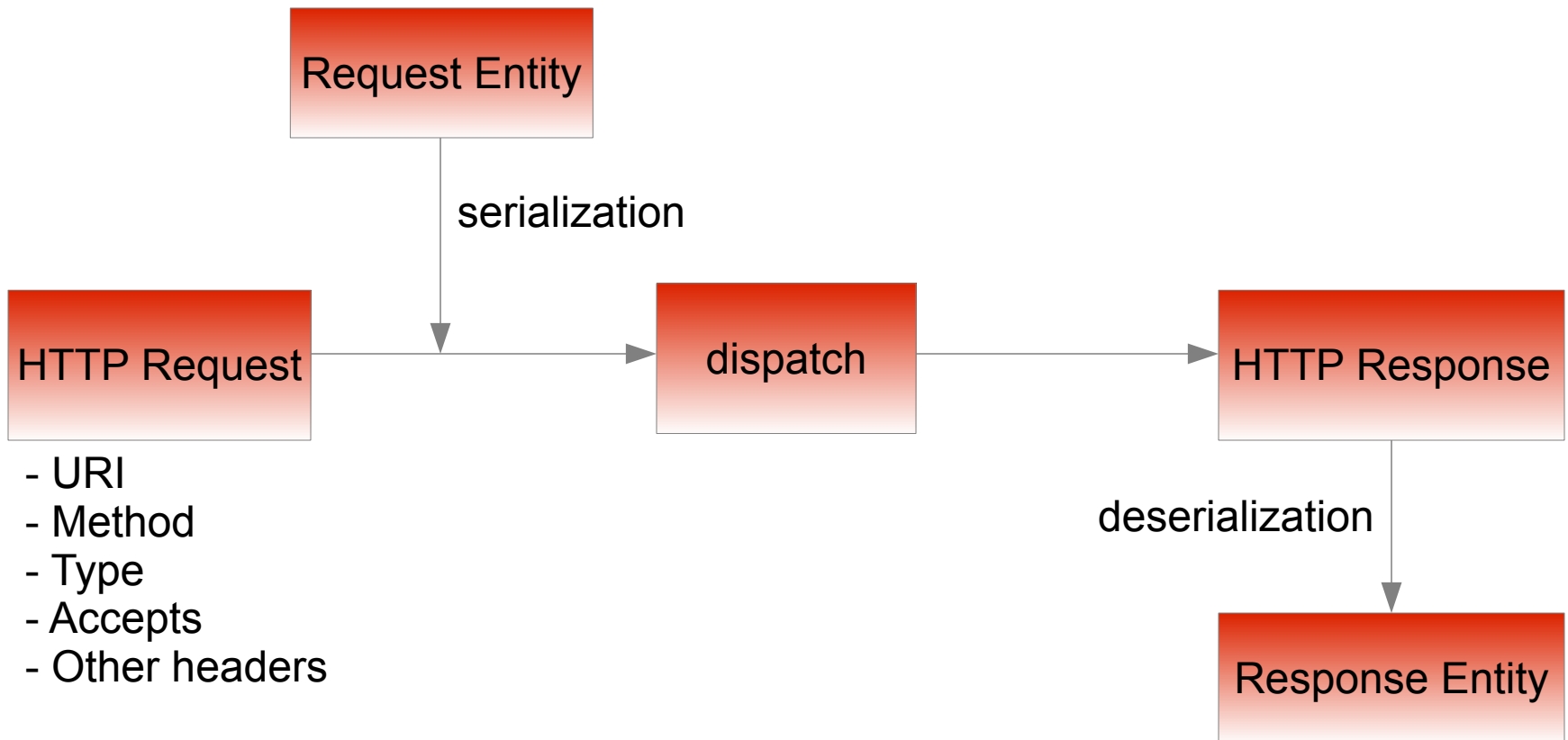
```
URL postUrl = new URL("http://localhost/customers");
HttpURLConnection connection = (HttpURLConnection)
postUrl.openConnection();
connection.setDoOutput(true);
connection.setInstanceFollowRedirects(false);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/xml");
OutputStream os = connection.getOutputStream();
os.write(newCustomer.getBytes());
os.flush();
connection.getResponseCode();
```

Jersey Client vs HttpURLConnection

b) Jersey Client

```
Client c = Client.create();  
WebResource wr = c.resource("http://localhost/customers");  
ClientResponse response =  
    wr.type(MediaType.APPLICATION_XML).  
    post(ClientResponse.class, newCustomer);  
  
response.getStatus();
```

HTTP Request Processing



Jersey Client – API Crash course

```
Client client = Client.create();  
Client client = Client.create(ClientConfig cc);  
  
WebResource resource = c.resource(URI u);  
  
resource.queryParam("name", "value");  
resource.accept(MediaType.APPLICATION_XML_TYPE);  
resource.path("path");  
resource.header("name", "value");  
resource.get(Class <T> c);  
resource.entity(entity).post(Class<T> c);  
  
client.addFilter();  
resource.addFilter();
```

Jersey Client – Customization, Extensibility

- ClientConfig
- Custom Providers – MessageBodyReaders/Writers
 - XML, Json, ...
- Filters
 - `GZIPContentEncodingFilter`, `HTTPBasicAuthFilter (+Digest)`, ...
- Apache HTTP Client integration
 - 3.x stable, 4.x experimental

Demo

- Using Jersey Client API
 - ClientConfig
 - Custom Filter
 - Custom Provider

Future?

- JAX-RS 2.0
 - Client API will be part of specification
 - Low level and high level API
- Hypermedia (loose coupling, “automatization”)
- Better Async support & Thread management
- Generating clients from wadl or other metadata
- Service as a Client

Jersey Client - Summary

- Stable, production ready Client API
- Widely adopted, convenient features
- JAX-RS 2.0

- <http://jersey.java.net>
- users@jersey.java.net

- pavel.bucek@oracle.com