
JavaOne™


JavaOne™
JAX-RS-ME
Michael Lagally
Principal Member of Technical Staff, Oracle
MAKE THE
FUTURE
JAVA
ORACLE

CON4244 JAX-RS-ME

JAX-RS-ME: A new API for RESTful web clients on JavaME

- This session presents the JAX-RS-ME API that was developed for Java ME.
- JAX-RS-ME defines a client API for Java ME based on JSR339 to consume web services using REST concepts from Java.
- The API provides a higher level abstraction than HTTP to:
  - access Web resources
  - transmit arbitrary (MIME-) types
  - transmit Java objects
- The API encapsulates web concepts such as: target, path, invocation, request, response, entity, headers

Program Agenda

- What is REST and why is it important ?
- JAX-RS-ME
  - Architecture, Principles, Model
- JAX-RS-ME examples
- How does it work ?

What is REST ?

- REST (**RE**presentational **S**tate **T**ransfer)
  = A model for distributed services in service-oriented architectures (SOA)
- *Principle: A system is a set of distributed, reusable, decoupled services*
- Earlier technologies are DCE, CORBA, Java RMI, SOAP based Web-services

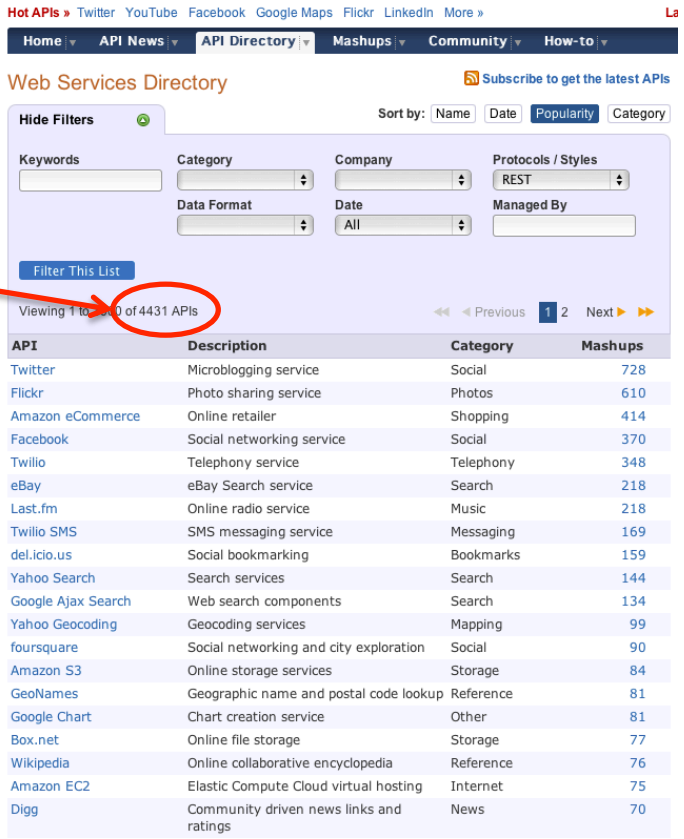
- Typically people think “Rest over HTTP”

- Nearly every web service is "REST”
  -> every simple http request fits this model.

Why is REST important ?

- REST is the most popular interface for WebServices
- > 4400 REST services
- 1350 SOAP services
- 150 XML-RPC services

- Source: Web services directory: http://www.programmableweb.com/apis/directory/1?protocol=REST



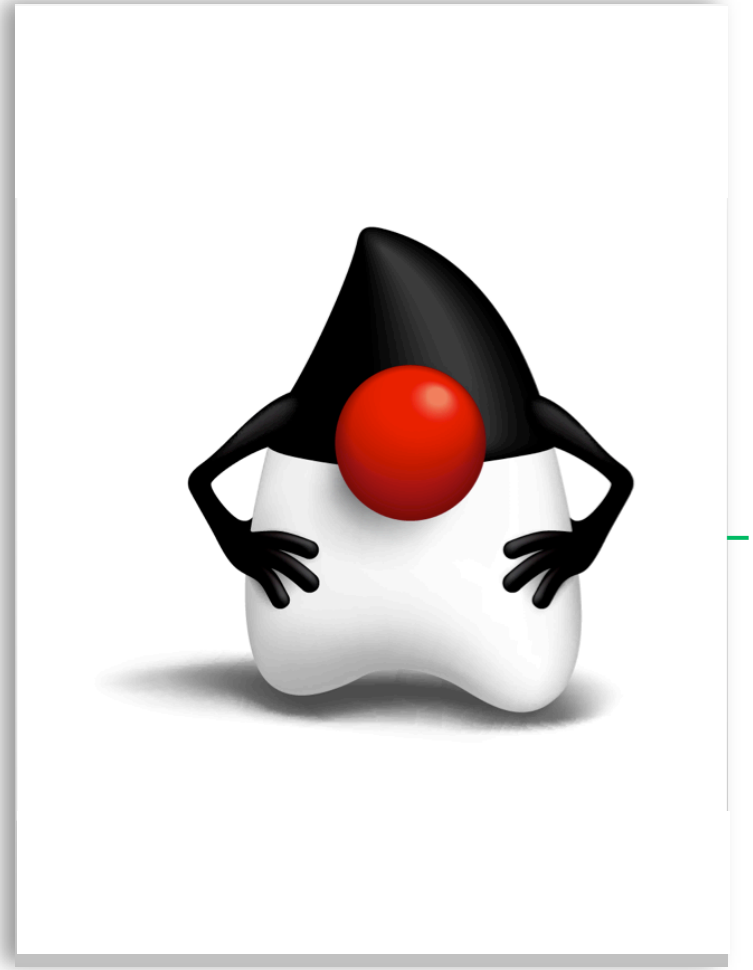
API	Description	Category	Mashups
Twitter	Microblogging service	Social	728
Flickr	Photo sharing service	Photos	610
Amazon eCommerce	Online retailer	Shopping	414
Facebook	Social networking service	Social	370
Twilio	Telephony service	Telephony	348
eBay	eBay Search service	Search	218
Last.fm	Online radio service	Music	218
Twilio SMS	SMS messaging service	Messaging	169
del.icio.us	Social bookmarking	Bookmarks	159
Yahoo Search	Search services	Search	144
Google Ajax Search	Web search components	Search	134
Yahoo Geocoding	Geocoding services	Mapping	99
foursquare	Social networking and city exploration	Social	90
Amazon S3	Online storage services	Storage	84
GeoNames	Geographic name and postal code lookup	Reference	81
Google Chart	Chart creation service	Other	81
Box.net	Online file storage	Storage	77
Wikipedia	Online collaborative encyclopedia	Reference	76
Amazon EC2	Elastic Compute Cloud virtual hosting	Internet	75
Digg	Community driven news links and ratings	News	70

JavaOne ORACLE

REST Principles

- Addressable resources
  - Each service/object is a resource that is addressable with a URI
- A uniform small interface
  - HTTP methods (GET, PUT, POST, DELETE, …)
- Representation-oriented
  - A single resource can have different external representations (e.g. HTML, XML, JSON, …)
- Stateless communication
  - Easy scalability, no session context required
- Hypermedia As the Engine Of Application State (HATEOAS)
  - Hypermedia and hyperlinks model the state and transitions of an application

JAX-RS-ME

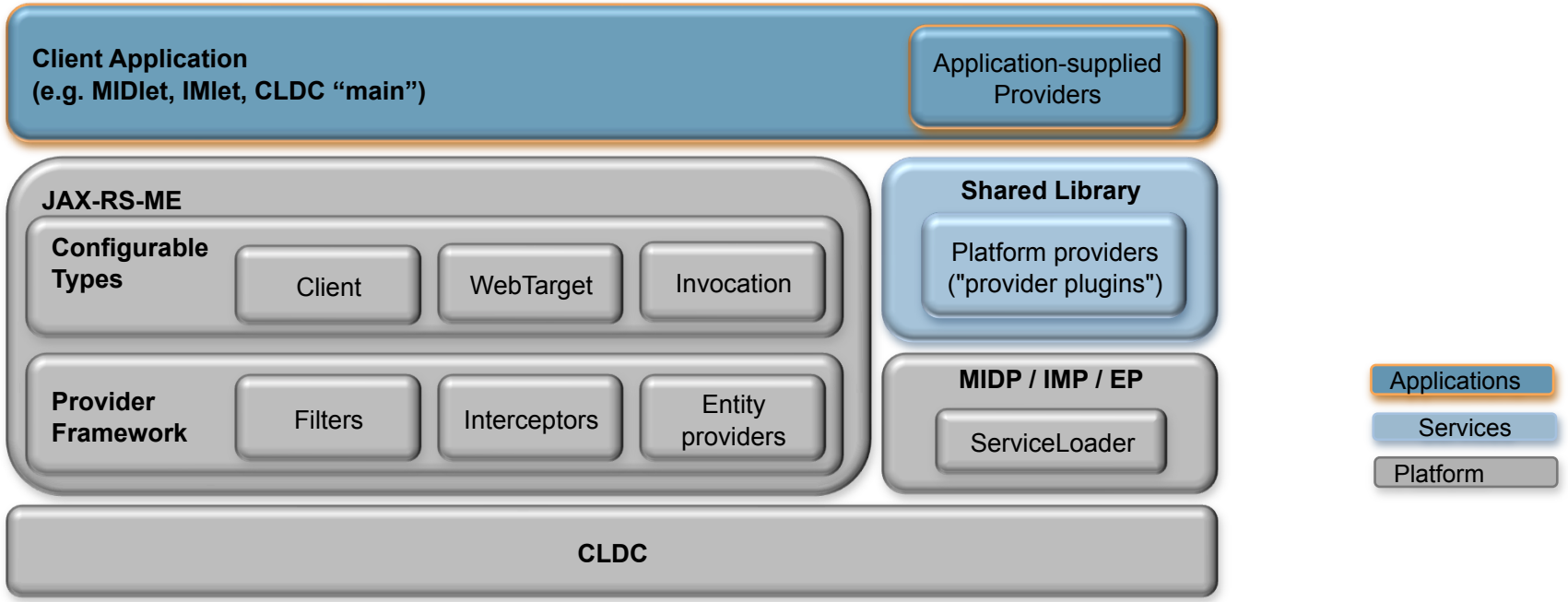


JAX-RS-ME

What is it ?

- Client API for Java ME to use REST-based Web Services
- Subset of the client part of JSR 339 (JAX-RS 2.0 Client + Server API)
- A higher level API Framework (above HTTP)
- Enables transmitting of arbitrary Java types
- Agnostic from the external representation (e.g. JSON, XML)
  - Client and server use HTTP Content Negotiation
- The API encapsulates REST and Web concepts such as:
  - WebTarget, URI, Path, Invocation, (HTTP) Request, (HTTP) Response, Entity, Request headers, Response headers

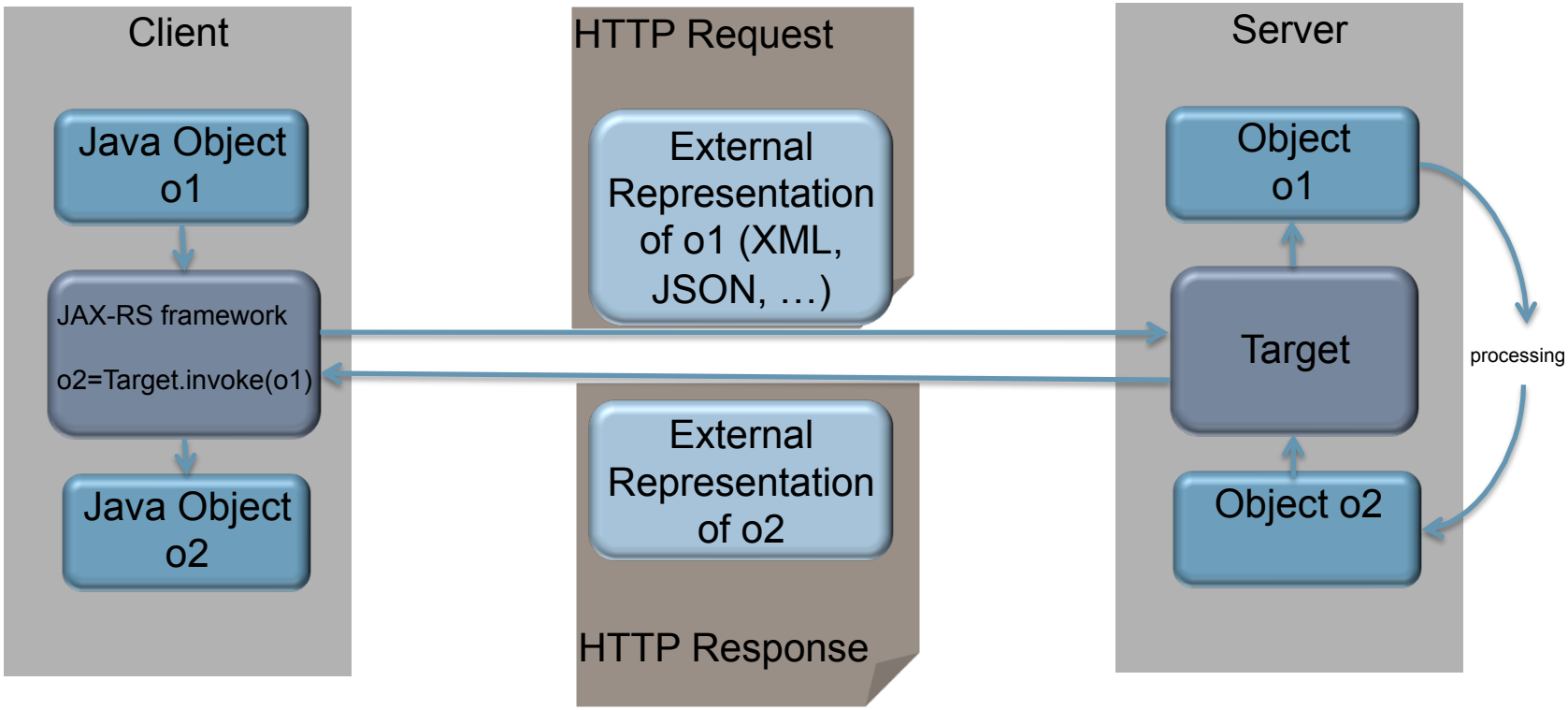
JAX-RS-ME Architecture



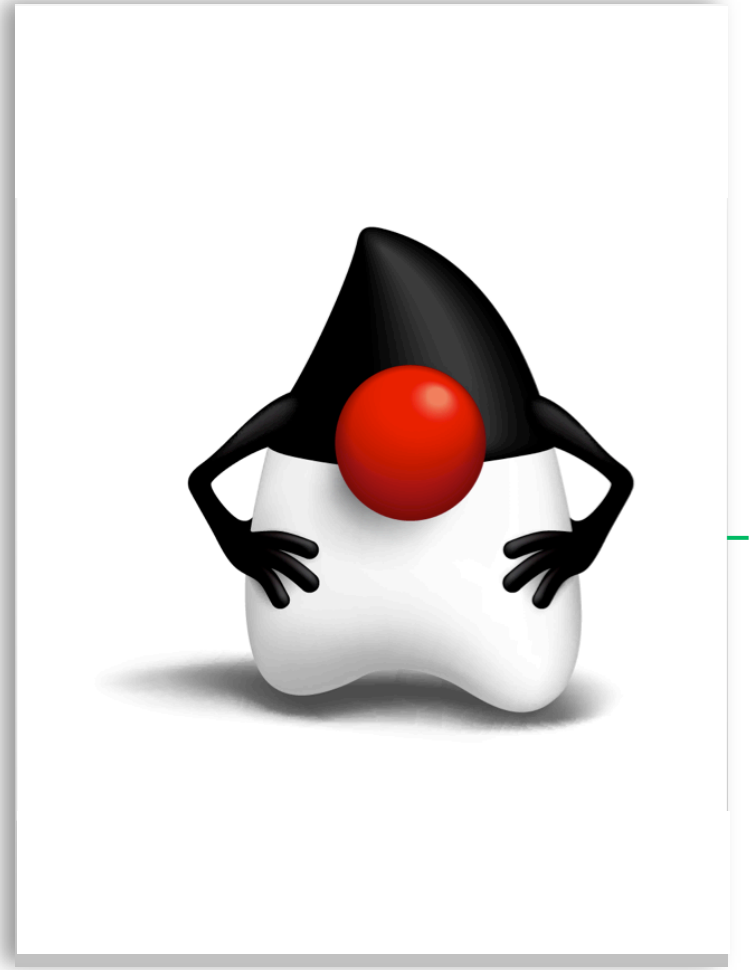
Why is JAX-RS useful for Java ME clients ?

- A lightweight Java API to use all REST based Web Services from feature-phones and small embedded devices
- Based on HTTP, small API, limited footprint
- An extensible framework for serialization / de-serialization of Java types
  - PUT / GET Java objects over HTTP
  - Out-of-the-box support for simple types (byte[], String, InputStream, …)
- “Fluent” API style allows expressive chaining of method calls
- API hides protocol- and content representation details from the application

The JAX-RS model



JAX-RS-ME examples



Example 1

Receive via JSON

```
// Receive an instance of a UserClass via JSON
Client client = ClientFactory.newClient();
// start with the remote server …
WebTarget target =   client.target("http://example.com");
// … and address a remote resource
target = target.path(”<aWebResourceforJSON>”);
// Create a request which expects a JSON response
Invocation inv= target.request(MediaType.APPLICATION_JSON_TYPE).buildGet();
// Invoke the request
Response res = inv.invoke();
// Access the response object
UserClass uc1 = res.readEntity(UserClass.class);
```


Example 1a

Receive via JSON in “Fluent Style”

```
// Receive an instance of a UserClass via JSON
Client client = ClientFactory.newClient();
// Get the response by invoking the GET method
Response res = client.target ("http://example.com").path(”<aWebResourceForJSON>”).
   request(MediaType.APPLICATION_JSON_TYPE).buildGet().invoke();
// Access the response object
UserClass uc1 = res.readEntity(UserClass.class);
```

Fluent Style = chaining method calls

Benefit: More expressive, more compact source code

Example 2

Receive via JSON or XML

```
// Receive an instance of a UserClass via JSON
Client client = ClientFactory.newClient();
// start with the remote server …
WebTarget target =   client.target("http://example.com");
// … and address a remote resource
target = target.path(”<aWebResourceforJSONorXML>”);
// Create a request which expects a JSON or XML response
Invocation inv= target.request(MediaType.APPLICATION_JSON_TYPE,
   MediaType.APPLICATION_XML_TYPE).buildGet();
// Invoke the request
Response res = inv.invoke();
// Access the response object
UserClass uc1 = res.readEntity(UserClass.class);
```


Example 3

Send via JSON

```
// Send an instance of a UserClass via JSON
UserClass uco=new UserClass (…);
Client client = ClientFactory.newClient();
// Create a WebTarget instance
WebTarget target=client.target("http://example.com").path(”<aWebResourceforJSON>");
// Create an Entity for the UserClass instance in JSON
Entity entity = Entity.json(uco);
// invoke the POST method
Response res = target.request().buildPost(entity).invoke();
// check the response status
if (res.getStatusEnum()!=Response.Status.OK) { /* error handling */ }
```

Example 4

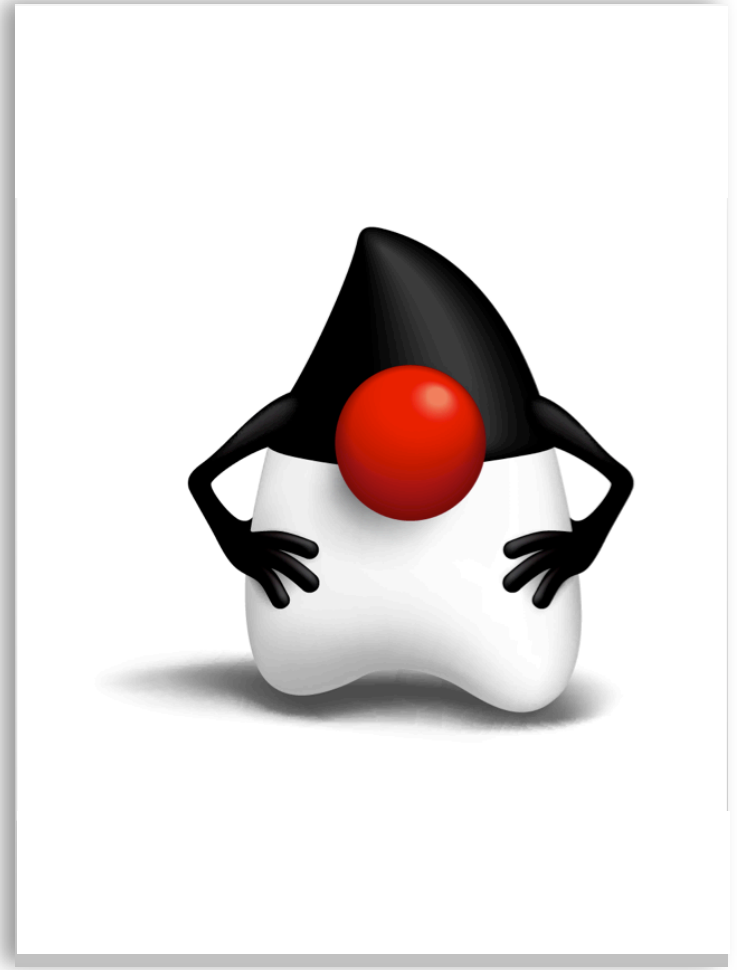
Send via XML

```
// Send an instance of a UserClass via JSON
UserClass uco=new UserClass (…);
Client client = ClientFactory.newClient();
// Create a WebTarget instance
WebTarget target=client.target("http://example.com").path(”<aWebResourceforXML>");
// Create an Entity for the UserClass instance in JSON
Entity entity = Entity.xml(uco);
// invoke the POST method
Response res = target.request().buildPost(entity).invoke();
// check the response status
if (res.getStatusEnum()!=Response.Status.OK) { /* error handling */ }
```

Observations

- The method invocation is independent from the external representation
- Minimum changes required to adapt to other representation
- All details of the parsing / encoding process are separated from the application
- An application just invokes methods on WebResources and passes Java objects as parameters
- This works for simple objects, arrays, generic types
- “Fluent Style” permits chaining of methods into a single statement.

How does it work ?



JAX-RS-ME

How does it work ?

- The API defines a client-side framework for:
  - Constructing remote object references (WebTarget)
  - Creating server requests
  - Invoking requests on the server
  - processing server responses
  - Converting between Java and external representation formats (e.g. XML, JSON, HTML, text)

- JAX-RS Operations are HTTP methods:
  - GET ≈ read
  - PUT ≈ write
  - POST ≈ add
  - DELETE ≈ delete

How to speak a common language between the client and the server ?

- External representation format depends on common capabilities of the client and the server
- Client indicates its capabilities via HTTP Content Negotiation
  - Mime/type (e.g. text/*, text/html, application/xml, application/json)
  - Language (e.g. en-us, es, fr)
  - Encoding (e.g. gzip, compress, deflate)
- The client maps Java objects to HTTP Requests / Responses and processes the HTTP messages
- This mapping happens via an extensible **Provider** framework
- The framework can be configured with **Configurable Types**

Configurable Types

- JAX-RS-ME contains the following configurable types:
  - Client
  - WebTarget
  - Invocation.Builder
  - Invocation
- Per configurable type the providers and other properties can be individually set

Providers

JAX-RS-ME extension points

- JAX-RS-ME defines several provider types:
  - **Entity Providers** (MessageBodyReaders, MessageBodyWriters) mapping between external representation(s) and Java object(s).
  - **Entity Interceptors** wrap around the read and write methods of the Entity providers. Can be used for message encoding/decoding (e.g. gzip compression)
  - **Filters** provide an extension point before/after a request is dispatched to/from transport layer.
    Filters can be used to augment the Entity Providers (e.g. Logging)

Provider invocation

Extension of the framework

- **Dynamic provider lookup and usage**
  - Entity providers are dynamically determined at runtime based on MIME type and Java type
  - If no Reader for a specific type can be found, a reader for the nearest supertype will be used
  - Filters and Interceptors are sorted based on a binding precedence

Provider deployment

Extension of the framework

- **Provider Deployment**

  Additional Providers can be created to adapt to other transport formats or new content types

  A single provider instance can have multiple roles, i.e. handle multiple Java types and external representations

  Providers can be:

  - included in the platform
  - deployed as a service (via ServiceLoader)
  - bundled with the application

Entity Providers

MessageBodyReaders

- MessageBodyReaders convert a HTTP message body to a Java object
- Only 2 methods to implement (pseudocode):

```
  public boolean isReadable(JavaType, MediaType) {
  // return true, if the writer can be used
  // to convert from MediaType to JavaType
}
public T readFrom(JavaType, MediaType, HTTPHeaders, InputStream) {
  // return an instance of JavaType by reading from an
  // InputStream in the appropriate MediaType format
}
```


Entity Providers

MessageBodyWriters

- MessageBodyWriters convert a Java object to a HTTP message body
- Only 3 methods to implement (pseudocode):

```
public boolean isWriteable(JavaType, MediaType)
  // return true if the writer can be used to convert
  // from JavaType to MediaType
public long getSize(JavaType, MediaType)
  // return the length of the serialized form of JavaType
public void writeTo(JavaObject, JavaType, MediaType, HTTPHeaders,
  OutputStream)
  // write an instance of the JavaObject to the OutputStream
  // in the appropriate MediaType
```


Providers

Readers & Writers, Filters and Interceptors

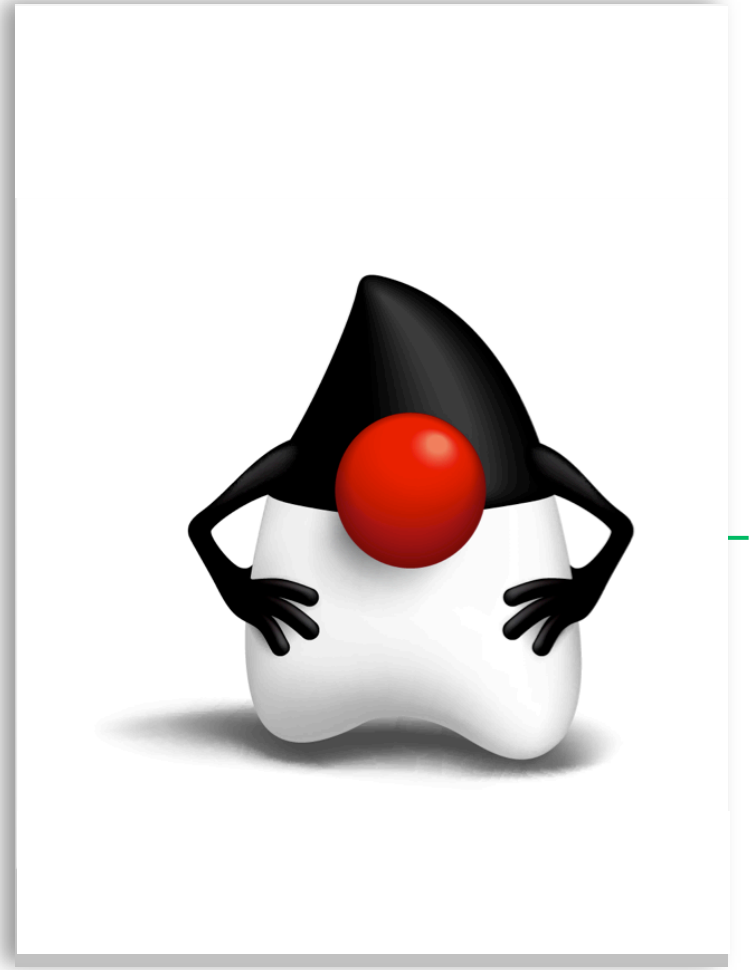
- JAX-RS-ME defines various extension points:
  - Configuration
    - Set of providers and other properties
  - Readers / Writers
    - Enable handling of arbitrary Java types
    - Enable adaptation to new payload formats
  - Filters and interceptors
    - Can be used for additional processing of HTTP messages, e.g. compression or mapping to other protocols

Standard entity providers

All JAX-RS-ME implementations support Readers, Writers for the the following types:

Java type	Transfer encoding
byte[]	All media types (*/*)
java.lang.String	All media types (*/*)
java.io.InputStream	All media types (*/*)
java.io.Reader	All media types (*/*)
MultivaluedMap<String,String>	(application/x-www-form-urlencoded) Form content
StreamingOutput	All media types (*/*)MessageBodyWriter only

Summary



JAX-RS-ME

Design Objectives

- Clean subset of JSR339
  - No additional classes / methods
  - Suitable for resource constrained phones and small embedded devices
  - Match the capabilities of the Java ME platform
- Enable reuse of (Java SE / EE) JAX-RS code
- Java ME applications can be written using a subset of the familiar REST API from JSR339

Java ME challenges

- No runtime annotations in Java ME
  - -> no annotation-based discovery of Providers (Readers, Writer, Filters, Interceptors)
  - -> explicit registration from application / service is required
- No platform support for JAXB in Java ME
  - -> if required, the application can provide JAXB readers/writers

JAX-RS-ME for embedded

- JAX-RS-ME defines an abstract way of interacting with REST servers
- Protocols, encodings, etc. are hidden from the application
- For embedded/M2M there are other RESTful protocols (e.g. CoAP), that could be used via the JAX-RS-ME API
- This requires a mapping via filters and interceptors

More information

- JSR 339
  - http://www.jcp.org/en/jsr/detail?id=339
- Jersey (Java EE JAX-RS implementation)
  - jersey.java.net
- Bill Burke: RESTful Java with JAX-RS
  - O'Reilly 2009, ISBN: 987-0-596-15804-0

Acknowledgements

- Many thanks to the JSR339 specification leads Santiago Pericas-Geertsen and Marek Potociar for constructive help and advice.

- Many thanks go also to all the numerous people at Oracle, who helped with contributions, support, feedback and invaluable discussions.

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.


MAKE THE
FUTURE
JAVA
JavaOne
ORACLE


JavaOne™