TomEE and Apache Johnzon - JAX-RS JSON Provider

Since TomEE 7.0, TomEE comes with Apache Johnzon. It means you can use JSON-P out of the box but also Johnzon Mapper which is the default JAX-RS provider for JSON.

IMPORTANT - this is a breaking change with 1.x which was using jettison. This last one was relying on JAXB model to generate JSON which often led to unexpected JSON tree and some unexpected escaping too.

Getting started with Johnzon Mapper

http://johnzon.apache.org/ will get more informations than this quick getting started but here are the basics of the mapping with Johnzon.

The mapper uses a direct java to json representation.

For instance this java bean:

```
public class MyModel {
  private int id;
  private String name;

  // getters/setters
}
```

will be mapped to:

```
{
  "id": 1234,
  "name": "Johnzon doc"
}
```

Note that Johnzon supports several customization either directly on the MapperBuilder of through annotations.

@JohnzonIgnore

@JohnzonIgnore is used to ignore a field. You can optionally say you ignore the field until some version if the mapper has a version:

```
public class MyModel {
  @JohnzonIgnore
  private String name;

  // getters/setters
}
```

Or to support name for version 3, 4, ... but ignore it for 1 and 2:

```
public class MyModel {
  @JohnzonIgnore(minVersion = 3)
  private String name;

  // getters/setters
}
```

@JohnzonConverter

Converters are used for advanced mapping between java and json.

There are several converter types:

1. Converter: map java to json and the opposite based on the string representation
2. Adapter: a converter not limited to String
3. ObjectConverter.Reader: to converter from json to java at low level
4. ObjectConverter.Writer: to converter from java to json at low level
5. ObjectConverter.Codec: a Reader and Writer

The most common is to customize date format but they all take. For that simple case we often use a Converter:

```
public class LocalDateConverter implements Converter<LocalDate> {
    @Override
    public String toString(final LocalDate instance) {
        return instance.toString();
    }

    @Override
    public LocalDate fromString(final String text) {
        return LocalDate.parse(text);
    }
}
```

If you need a more advanced use case and modify the structure of the json (wrapping the value for instance) you will likely need Reader/Writer or a Codec.

Then once your converter developed you can either register globally on the MapperBuilder or simply decorate the field you want to convert with @JohnzonConverter:

```
public class MyModel {
  @JohnzonConverter(LocalDateConverter.class)
  private LocalDate date;

  // getters/setters
}
```

@JohnzonProperty

Sometimes the json name is not java friendly (_foo or foo-bar or even 200 for instance). For that cases @JohnzonProperty allows to customize the name used:

```
public class MyModel {
  @JohnzonProperty("__date")
  private LocalDate date;

  // getters/setters
}
```

AccessMode

On MapperBuilder you have several AccessMode available by default but you can also create your own one.

The default available names are:

- field: to use fields model and ignore getters/setters
- method: use getters/setters (means if you have a getter but no setter you will serialize the property but not read it)
- strict-method (default based on Pojo convention): same as method but getters for collections are not used to write

You can use these names with setAccessModeName().

Your own mapper

Since johnzon is in tomee libraries you can use it yourself (if you use maven/gradle set johnzon-mapper as provided):

```
final MySuperObject object = createObject();

final Mapper mapper = new MapperBuilder().build();
mapper.writeObject(object, outputStream);

final MySuperObject otherObject = mapper.readObject(inputStream, MySuperObject.class);
```

Johnzon and JAX-RS

TomEE uses by default Johnzon as JAX-RS provider for versions 7.x. If you want however to customize it you need to follow this procedure:

1. Create a WEB-INF/openejb-jar.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar>
 <pojo-deployment class-name="jaxrs-application">
   <properties>
     # optional but requires to skip scanned providers if set to true
     cxf.jaxrs.skip-provider-scanning = true
     # list of providers we want
     cxf.jaxrs.providers =
johnzon,org.apache.openejb.server.cxf.rs.EJBAccessExceptionMapper
   </properties>
 </pojo-deployment>
</openejb-jar>
```

1. Create a WEB-INF/resources.xml to define johnzon service which will be use to instantiate the provider

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
 <Service id="johnzon" class-name=
"org.apache.johnzon.jaxrs.ConfigurableJohnzonProvider">
   # 1M
   maxSize = 1048576
   bufferSize = 1048576

   # ordered attributes
   attributeOrder = $order

   # Additional types to ignore
   ignores = org.apache.cxf.jaxrs.ext.multipart.MultipartBody
 </Service>

 <Service id="order" class-name="com.company.MyAttributeSorter" />

</resources>
```

Note: as you can see you mainly just need to define a service with the id johnzon (same as in openejb-jar.xml) and you can reference other instances using $id for services and @id for resources.