

JAX-RS 2.0 REST mit Java EE 7

Java User Group Darmstadt
13. Juni 2013

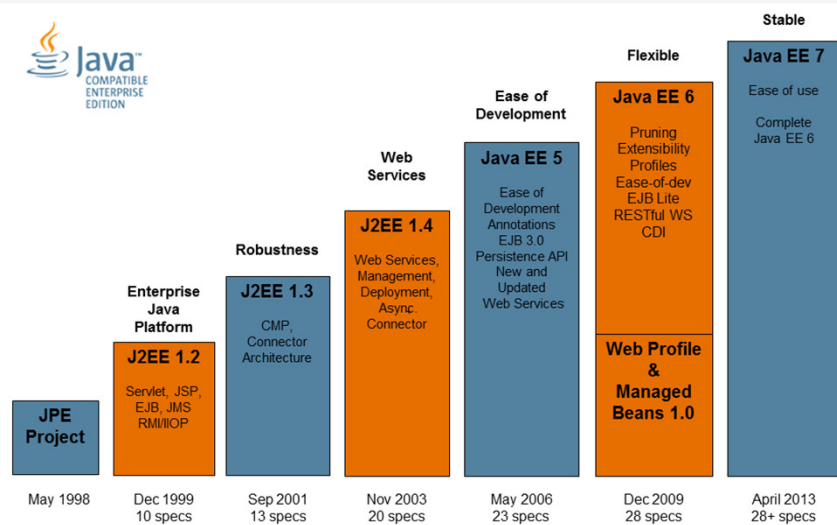
Vorstellung

- Freiberuflicher Softwarearchitekt und Trainer
- Fachliche Schwerpunkte
 - Java Plattform
 - Services und Integration
- Kundenspezifische Inhouse-Schulungen
- (Co-)Autor von 2 Büchern & zahlreichen Artikeln
- Sprecher auf internationalen Konferenzen

JAX-RS 2.0 (JSR-339)

- Referenzimplementierung: Jersey 2.0
 - Aktueller Stand: 2.0 Final Release (14. Mai 2013)
- Bestandteil von Java EE 7 (JSR-342)
 - Final Release am 28.05.2013
- Glassfish 4.0: Java EE 7 RI
 - <http://glassfish.java.net/>
 - Glassfish 4.0.1 in Kürze zu erwarten

Java EE Roadmap



Neuigkeiten in JAX-RS 2.0

- Client-API
- Filter
- Entity Interceptors
- Support für Asynchronität (Server & Client)
- Validation
- Features
- (Hypermedia)

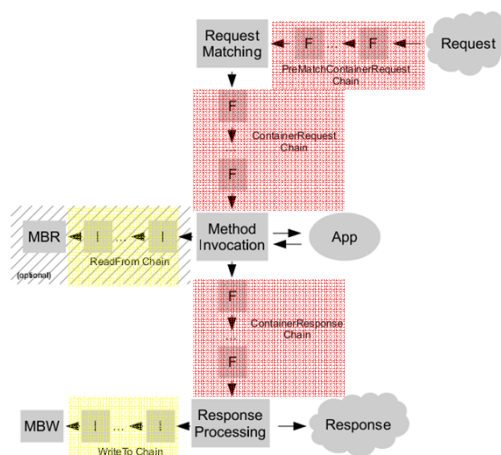
Erweiterungen für JAX-RS (1)

- Provider
 - Marshalling und Unmarshalling
 - Umwandlung von Exceptions in Responses
 - Bereitstellen von Kontextinformationen für Ressourcen oder für andere Provider
- Entity Interceptors
 - Interceptors für Marshalling / Unmarshalling
- Filter
 - Pre-/Post-Processing auf HTTP-Ebene

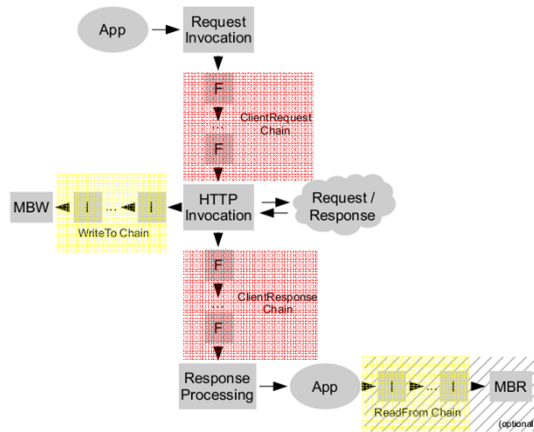
Erweiterungen für JAX-RS (2)

- Features
 - Kapselung mehrerer zusammen gehöriger Erweiterungen
- Vorsicht, verwirrend!
 - Provider, Interceptors und Filter werden alle mit `@Provider` markiert
 - Alternativ: manuelle Registrierung via API

Processing Pipeline (Server)



Processing Pipeline (Client)



Filter

- JAX-RS definiert vier Interfaces für Filter
- Anordnung in Ketten
- Filter können Kette unterbrechen (abortWith)
 - z.B. fehlgeschlagener Authorisierungsfiler
- Sortierung auf Basis von Priorities
- Vergleichbar mit Servlet Filter / SOAP Handler
- Einsatzgebiete:
 - Protokoll-spezifische Erweiterungen
 - Security, Caching, HTTP Method Rewrite etc.

Filter (2)

```
public interface ClientRequestFilter {
    void filter(ClientRequestContext requestContext)
        throws IOException;
}

public interface ClientResponseFilter {
    void filter(ClientRequestContext requestContext,
        ClientResponseContext responseContext)
        throws IOException;
}

public interface ContainerRequestFilter {
    void filter(ContainerRequestContext requestContext)
        throws IOException;
}

public interface ContainerResponseFilter {
    void filter(ContainerRequestContext requestContext,
        ContainerResponseContext responseContext)
        throws IOException;
}
```

Entity Interceptors

- Filter wirken vor Marshalling/Unmarshalling der Entities im Input/Outstreams
 - keine Manipulation der Entities möglich
- Interceptors kapseln dagegen MBR/MBW
 - direkter Zugang auf Entity-Streams
- Können ebenfalls verkettet werden
- JAX-RS definiert 2 Interfaces
- Einsatzgebiete:
 - GZIP, Verschlüsselung, spez. Repräsentationen etc.

Entity Interceptors (2)

```
public interface ReaderInterceptor {  
    Object aroundReadFrom(ReaderInterceptorContext context)  
        throws java.io.IOException,  
            javax.ws.rs.WebApplicationException;  
}  
  
public interface WriterInterceptor {  
    void aroundWriteTo(WriterInterceptorContext context)  
        throws java.io.IOException,  
            javax.ws.rs.WebApplicationException;  
}
```

Asynchronität

```
@Path("/longRunningCalculation")  
public class MyResource {  
    @GET  
    public void longRunningCalc(@Suspended final AsyncResponse ar) {  
        Executors.newSingleThreadExecutor().submit(  
            new Runnable() {  
                public void run() {  
                    ComplexCalcResult calcResult = executeLongRunningCalc();  
                    ar.resume(calcResult);  
                }  
            }  
        );  
        // Return immediately and handle other requests  
    }  
}
```

Bsp 1: Ressource mit asynchroner Request-Verarbeitung

Asynchronität (2)

```
@Stateless
@Path("/longRunningCalculation")
public class MyResource {
    @GET
    @Asynchronous
    public void longRunningCalc(@Suspended final AsyncResponse ar) {
        ComplexCalcResult calcResult = executeLongRunningCalc ();
        ar.resume(calcResult);
    }
}
```

Bsp 2: Ressource mit asynchroner Request-Verarbeitung (Session Bean)

Asynchronität (3)

```
InvocationCallback<List<Order>> callback =
    new InvocationCallback<List<Order>>() {
        @Override
        public void completed(List<Order> orders) { ... }
        @Override
        public void failed(Throwable t) {... }
    };

client.target("http://.../orders/")
    .request("text/xml")
    .async()
    .get(callback);
```

Bsp 3: Nicht-blockierender Request (Client)

Sonstige Neuigkeiten

- Features
 - Kapseln mehrere Erweiterungen
- Validation
 - Integration mit Bean Validation
 - Constraint Annotations in Ressourcen
 - Prüfung von Beans und Methodenparametern
 - Prüfung von Rückgabewerten
- Hypermedia
 - begrenzter Support für Transitional Links (in Headers)
 - Kein Support für Structural Links

Zusammenfassung

- JAX-RS 2.0 bietet zahlreiche interessante Erweiterungen und Neuigkeiten
- Standardisierte Client-API
- Filter und Interceptors
- Asynchrone Verarbeitung
- Validation

- Verfügbar separat (Jersey 2.0 in Maven Central) oder mit der Java EE 7-Referenzimplementierung



Thilo Frotscher
Kundenspezifisches
Training und Beratung zu
Java EE, Services und Integration

thilo@frotscher.com