ARS

Member of

TIMETOACT GROUP
SOFTWARE & CONSULTING

Servlet-Pluggability

Java EE Grundlagen

ARS

Inhalte dieses Kapitels

Motivation

Web-Fragmente

Pluggability API

Herausforderungen

Zusammenfassung

ARS

Motivation

Aufteilung von Webmodulen in Fragmente

- Relevanz für Entwicklung, Assembly und/oder Deployment
- Auslagern von Webmodul-Inhalten
  - Java-Klassen, Custom Tags
    (Einbinden von JARs über Classloader bereits möglich)
  - Web-Ressourcen: JSPs, statische Ressourcen (z.B. Stylesheets, JS)
  - Metadaten (URL Mappings, Servlet Definitionen, …)

Verwendung von Web-Frameworks

- Erweiterung des Webmodul-Classpath als einzige Maßnahme
  - Keine Eintragungen im Web Deployment Descriptor
  - Kein `ResourceServingServlet` o.Ä. zum Erweitern von Web-Ressourcen

ARS

Web-Fragmente [1|2]

Aufbau

- Erweiterung des JAR-Formats
- Web Fragment Deployment Deskriptor (`META-INF/web-fragment.xml`)
  - Schema (fast) identisch zu Web Deployment Deskriptor
- JSPs, statische Ressourcen (`META-INF/resources/*`)
  - Implizite Adressierbarkeit per URL
  - Unterordner als Namespace empfohlen (Namenskonflikte vermeiden)

Einbinden

- Erweiterung des Webmodul-Classpath (z.B. JAR in `WEB-INF/lib`)
- Bedingung: kein `metadata-complete="true"` im Web DD

ARS

Web-Fragmente [2|2]

Vergleich Webanwendung vs. Webbibliotheken

	Webanwendung (`.war`)	**Webbibliothek (`.jar` in `WEB-INF\lib`)**
Bytecode, Anw.-Ressourcen	`WEB-INF/classes/**`	direkt im JAR
Webressourcen (per URL erreichbar)	direkt im WAR (Ausnahme: `WEB-INF/*`)	`META-INF/resources/**`
Manifest	`META-INF/MANIFEST.MF`	
TLDs	`WEB-INF/**/*.tld`	`META-INF/**/*.tld`
Web (Fragment) DD	`WEB-INF/web.xml`	`META-INF/web-fragment.xml`
JSF Konfiguration	`WEB-INF/faces-config.xml`	`META-INF/faces-config.xml`
EJB DD	`WEB-INF/ejb-jar.xml`	

ARS

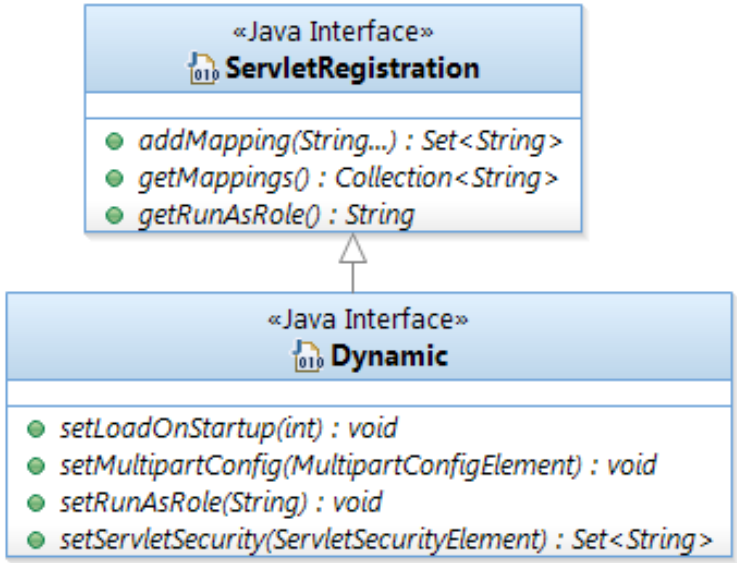
Pluggability API [1|8]

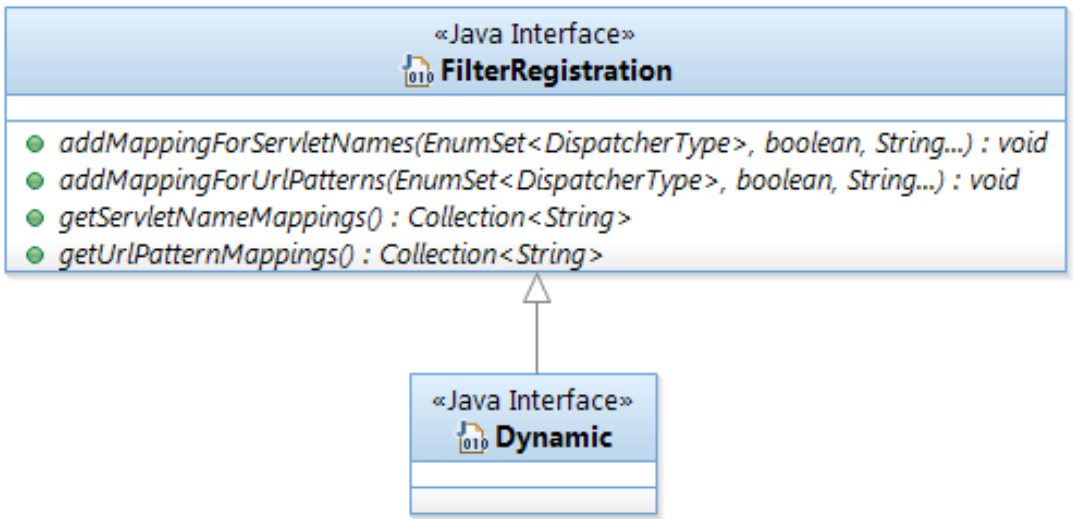
Programmierschnittstelle zur Erweiterung von Metadaten

- Klassen und Methoden zum Registrieren von
  - Servlets
  - Filtern
  - Event Handlern (außer `ServletContextListener`)
- Erweiterung von `ServletContext`
- Hinzufügen von Metadaten nur während der Startphase
  - `ServletContextListener` oder `ServletContainerInitializer`
  - Sonst `UnsupportedOperationException`
- Entfernen / Ändern von Metadaten nicht möglich

ARS

Pluggability API [2|8]





ARS

Pluggability API [3|8]

Beispiel: Registrieren eines neuen Servlets

```
@WebListener
public class Configurator implements ServletContextListener {

  @Override
  public void contextInitialized( ServletContextEvent e ) {
    Dynamic servlet = e.getServletContext()
      .addServlet( "Hello World Servlet", new HelloWorldServlet() );
    servlet.addMapping( "/hello" );
    servlet.setInitParameter( "param1", "value1" );
  }

  …

}
```

ARS

Pluggability API [4|8]

Servlet Container Initializer

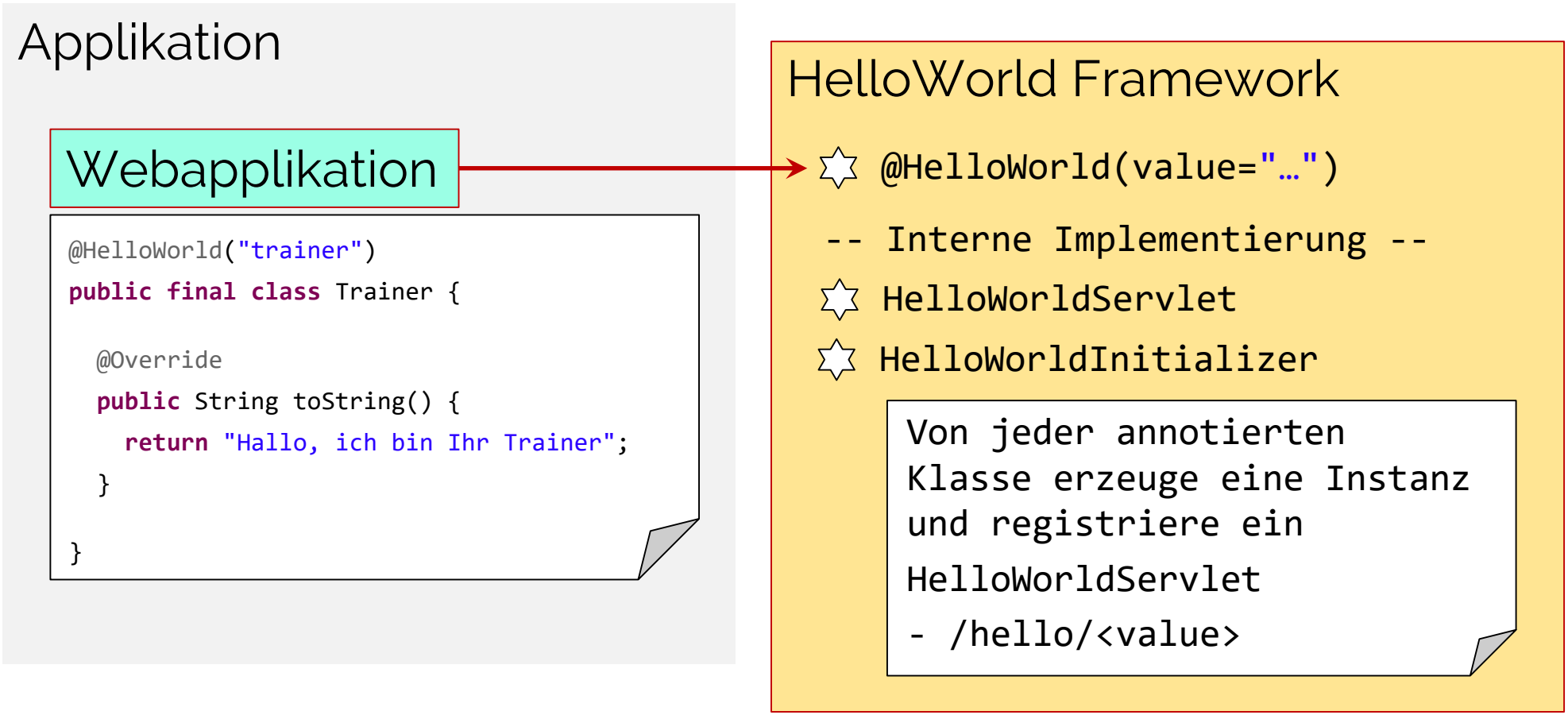
- Pluggability für Webcontainer (außerhalb der Applikation)
  - Classpath des Webcontainers
  - Verwendung auch innerhalb der Webapplikation möglich
- Ausführung einer Startup-Methode jeweils bei Start einer Applikation
- Registrierung via Service Provider Mechanism (ab Java 6)
  - `META-INF/services/javax.servlet.ServletContainerInitializer`
- Initialisierungen während der Classloading-Phase
  - Registrierung von Servlets, Filtern, … bei Auffinden bestimmter Klassen (deren Interfaces, Oberklassen, Annotationen)
  - Registrierung von `ServletContextListener`n möglich
    - Verwendung der Pluggability API darin nicht erlaubt

ARS

Pluggability API [5|8]

Use Case: Framework

- Anwendung für Java-Technologien als Server-Feature



ARS

Pluggability API [6|8]

Use Case: Framework

```
@HandlesTypes(HelloWorldSayer.class)
public class HelloWorldInitializer implements ServletContainerInitializer {

  @Override
  public void onStartup( Set<Class<?>> classes, ServletContext ctx ) throws ServletException {
    try {
      for ( Class<?> c : classes ) {
        final HelloWorld name = c.getAnnotation(HelloWorld.class);
        if ( null != name ) { // Annotation selbst kann dabei sein
          final Object hw = c.newInstance();
          final HelloWorldServlet servlet = new HelloWorldServlet( hw );
          ctx
            .addServlet( "HelloWorldServlet_" + c.getName(), servlet )
            .addMapping( "/hello/" + name.value() );
        }
      }
    } catch ( Exception e ) {
      throw new ServletException(e);
    }
  }

}
```

ARS

Pluggability API [7|8]

Praxisbeispiel: JAX-RS

- Bei Aufkommen von
  - Unterklassen von `javax.ws.rs.core.Application`
  - Klassen, die mit `javax.ws.rs.Path` annotiert wurden
  - Klassen, die mit `javax.ws.rs.ext.Provider` annotiert wurden
- Ablegen von Informationen in den Application Scope
- Registrierung eines RESTServlet
  - Übergabe der annotierten Klassen

ARS

Pluggability API [8|8]

Praxisbeispiel: Java Server Faces (MyFaces)

- Bei Vorkommen von Implementierungen von JSF-Interfaces bzw. bei Verwendung von JSF-Annotationen
  - Managed Beans
  - Konverter
  - Validatoren
- Registrierung des FacesServlet
  - Falls nicht Verwendung eines eigenen FacesServlets
  - Abspeichern von Informationen im Application Scope

ARS

Herausforderungen (Komplexität) [1|5]

Aufruf von Komponenten – ob und wann?

- Aufruf von Filtern und Event Handlern
- Verhalten bei doppelten Bezeichnern
  - Ignorieren vs. Fehler-werfen
- Reihenfolge bei Verwendung von Annotationen unspezifiziert
- Einstellungen in Web-DD haben Vorrang vor Web-Fragment-DD
- Servlet-Einstellungen in Web-DD haben Vorrang vor Annotationen
- Reihenfolge bei Web-Fragmenten
  - Erst Web-DD, dann Web-Fragment-DD, sortiert nach `<name>`
  - Ausnahmen
    - `<absolute-ordering>` in Web-DD
    - `<ordering>` in Web-Fragment-DD

ARS

Herausforderungen (Komplexität) [2|5]

Beispiel 1: Web-Fragmente (ohne Ordering)

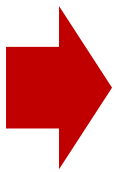
```
<web-fragment>
  <name>Fragment A</name>
  ...
</web-fragment>
```

+

```
<web-fragment>
  <name>Fragment B</name>
  ...
</web-fragment>
```

+

```
<web-fragment>
  <name>Fragment C</name>
  ...
</web-fragment>
```



```
Fragment A
Fragment B
Fragment C
```

ARS

Herausforderungen (Komplexität) [3|5]

Beispiel 2: Web-Fragmente (mit Relative Ordering)

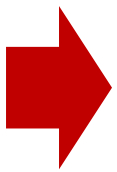
```
<web-fragment>
  <name>Fragment A</name>
  <ordering><after><name>Fragment B</name></after></ordering>
  ...
</web-fragment>
```

+

```
<web-fragment>
  <name>Fragment B</name>
  ...
</web-fragment>
```

+

```
<web-fragment>
  <name>Fragment C</name>
  <ordering><before><others/></before></ordering>
  ...
</web-fragment>
```



```
Fragment C
Fragment B
Fragment A
```

ARS

Herausforderungen (Komplexität) [4|5]

Beispiel 3: Web-Fragmente (zirkulare Abhängigkeiten)

```
<web-fragment>
  <name>Fragment A</name>
  <ordering><after><name>MyFragment B</name></after></ordering>
  ...
</web-fragment>
```

```
<web-fragment>
  <name>Fragment B</name>
  <ordering><after><name>Fragment C</name></after></ordering>
  ...
</web-fragment>
```

```
<web-fragment>
  <name>Fragment C</name>
  <ordering><after><name>Fragment A</name></after></ordering>
  ...
</web-fragment>
```

ARS

Herausforderungen (Komplexität) [5|5]

Beispiel 4: Web-Fragmente (Absolute Ordering)

```
<web-fragment>
  <name>Fragment A</name>
  ...
</web-fragment>
```

+

```
<web-fragment>
  <name>Fragment B</name>
  ...
</web-fragment>
```

+

```
<web-fragment>
  <name>Fragment C</name>
  ...
</web-fragment>
```

+

```
<web-app>
  <absolute-ordering>
    <name>Fragment B</name>
    <name>Fragment C</name>
    <others />
  </absolute-ordering>
  ...
</web-app>
```

→

```
Fragment B
Fragment C
Fragment A
```

ARS

Kontrollfragen

- Welche Vorteile bieten Web-Fragmente und die Pluggability API?
- Welche Rolle spielen Servlet Container Initializer und wann werden sie aufgerufen?
- Worin besteht die Gefahr bei der Modularisierung über Web-Fragmente?