ARS

Member of TIMETOACT GROUP SOFTWARE & CONSULTING

Contexts and Dependency Injection

Java EE Grundlagen

ARS

Inhalte dieses Kapitels

Allgemeines

Dependency Injection for Java (JSR-330)

Contexts and Dependency Injection (JSR-299)

Zusammenfassung

ARS

Allgemeines [1|3]

Motivation (lt. JSR 299)

- EJBs in Webanwendungen ohne Bezug zu Gültigkeitsbereichen Request, Session, und (Web-)Application
  - Kein Zugriff auf diese Gültigkeitsbereiche
  - Keine Zuordnung (*Lifecycle Management*)
- JSF Bean Management
  - Keine Dependency Injection
  - Keine Annotationen (Stand 2006 / JSF 1.x)
  - Inkonsistent mit JNDI

The goal of this work is to enable EJB 3.0 components to be used as JSF managed beans, unifying the two component models and enabling a considerable simplification to the programming model for web-based applications in Java.

ARS

Allgemeines [2|3]

Motivation (für Java-EE-Entwickler)

- Dependency Injection als Best Practice
  - Verstärkter Einsatz in anderen Bereichen
    - OSGi: BluePrint
    - Eclipse 4.0 (Anwendungsplattform)
    - Modultests: Mocking-Frameworks
- Enterprise Java
  - Keine Standards (außer *EJB/Resource Injection*)
  - Verwendung externer Frameworks notwendig
    - Spring
    - Google Guice

Standard für Java-EE-Anwendungen für bessere Konfigurierbarkeit, Testbarkeit, Portabilität (Staging) und Erweiterbarkeit

ARS

Allgemeines [3|3]

Dependency Injection (Allgemein)

- Benötigte Dienste/Ressourcen/Informationen werden von außen gesetzt (“injiziert”)
  - Instanzvariablen (*Field Injection*)
  - Konstruktorparameter (*Constructor Injection*)
  - Methodenparameter (*Parameter Injection*, *Setter Injection*)
- Ausführung durch Container
  - Konfiguration in externer Datei oder durch Annotationen
  - Unterschiedliche Konfigurationen für Tests möglich

ARS

Dependency Injection for Java (JSR-330) [1|5]

- Standard-Annotationen für typsichere Dependency Injection
- Keine Implementierung für Dependency Injection

Annotation (`javax.inject.*`)	Bedeutung
`@Inject`	Markierung von Abhängigkeiten (*Injectable Fields / Constructors / Methods*)
`@Qualifier`	Deklaration eigener Annotationen zur Ergänzung von `@Inject` (Flexibler als reine Injection by Type)
`@Named`	Spezieller Qualifier für Injection by Name
`@Scope`	Deklaration eigener Annotationen für Gültigkeitsbereiche
`@Singleton`	Spezieller Scope
`Provider<T>`	Injizierbares Objekt zur verzögerten, mehrfachen Abfrage einer Abhängigkeit

ARS

Dependency Injection for Java (JSR-330) [2|5]

Annotation `@Inject`

- Markierung von Abhängigkeiten, die per Injection an ein Objekt übergeben werden
- Reihenfolge
  - Konstruktor-Parameter → Instanzvariablen → Methoden-Parameter
  - Oberklasse vor Unterklasse
- Keine Einschränkungen bzgl. Sichtbarkeitsbereiche
- Instanzvariablen: nicht `final`
- Methoden: nicht `abstract`
- Konstruktoren: max. nur 1 Konstruktor mit `@Inject`

Details siehe Javadoc zu `@Inject`

ARS

Dependency Injection for Java (JSR-330) [3|5]

Annotation `@Inject` – Beispiel (inkl. `Provider<T>`)

```
public class Kitchen {

  @Inject
  private Stove stove;

  @Inject
  public Kitchen(Cook cook) {
    // ...
  }

  @Inject
  private void prepareFood(Provider<FoodPreparator> provider) {
    // ...
    FoodPreparator fp = provider.get();
    // ...
  }

}
```

ARS

Dependency Injection for Java (JSR-330) [4|5]

Annotation `@Qualifier`

- Erweiterung der *Injection by Type* um weiteres Merkmal
- Injektion eines Objektes, welches dem Merkmal entspricht
- `@Named` als ein Qualifier für *Injection by Name*

```java
@Inject @Electrical
private Stove stove;

@Inject @Named("grant.achatz")
public Kitchen(Cook cook) {
  // ...
}
```

```java
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
public @interface Electrical { }
```

ARS

Dependency Injection for Java (JSR-330) [5|5]

Annotation `@Scope`

- Deklaration von Gültigkeitsbereichen
  - Annotationen für injizierte Objekte
- Scopes nach Standard
  - Keine Angabe: 1 Instanz pro Injektion (vgl. Spring: *Prototype*)
  - `@Singleton`: 1 Instanz für alle Injektionen gemeinsam)

```
@Singleton
public class Kitchen {

  // ...

}
```

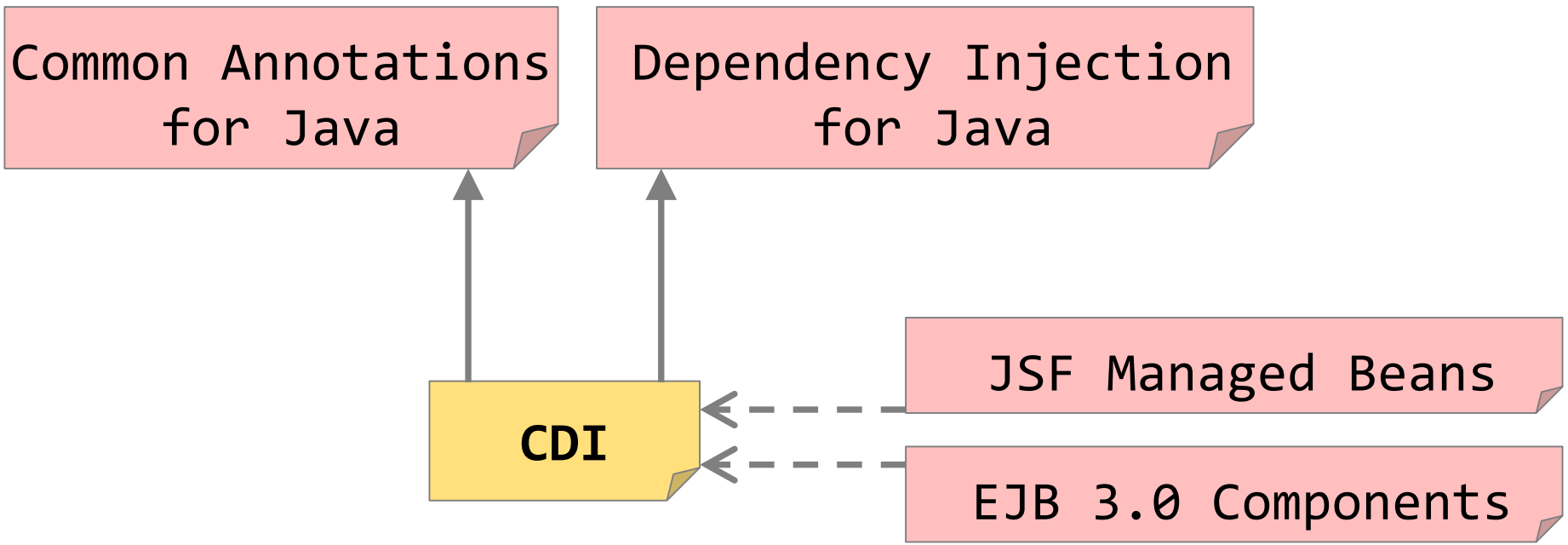
```
@Scope
@Retention(RetentionPolicy.RUNTIME)
public @interface KitchenScoped { }
```

```
@KitchenScoped
public class Cook {

  // ...

}
```

ARS

Contexts and Dependency Injection (JSR-299) [1|12]

- Ehem. „Web Beans“
- Ziel: Verwendung von EJB-Komponenten als JSF Managed Beans
- Ergebnis: Dependency Injection Framework für Java EE
- Standalone verwendbar (mit Apache Open WebBeans)



ARS

Contexts and Dependency Injection (JSR-299) [2|12]

Managed Beans – Definitionen

- *Bean* (nach Java EE)
  - Definition einer Komponente
    - Implementierung
    - Metadaten
- *Contextual Instances* einer *Bean*
  - Instanzen einer *Bean* im Kontext der Verwendung
  - *Lifecycle Management* durch Container
  - Zuordnung zu einem Kontext (abstrakte Definition)
    - Injizierbar in andere *Contextual Instances* desselben Kontextes
    - Verwendbar in EL Expressions, die im selben Kontext ausgewertet werden

ARS

Contexts and Dependency Injection (JSR-299) [3|12]

Managed Beans – vereinfachte Sicht

- Instanzen, die vom CDI-Container verwaltet werden
- Annotationen (optional)
  - Gültigkeitsbereich
  - Name u.a. Qualifier

```
public class FoodPreparator { /* ... */ }
```

```
@Inject
private FoodPreparator fp;
```

```
@KitchenScoped @Named("grant.achatz")
public class Cook { /* ... */ }
```

```
@Inject @Named("grant.achatz")
private Cook cook;
```

```
@Electrical
public class Stove { /* ... */ }
```

```
@Inject @Electrical
private Stove stove;
```

Voraussetzung: beans.xml in META-INF bzw. WEB-INF

ARS

Contexts and Dependency Injection (JSR-299) [4|12]

Managed Beans – Gültigkeitsbereiche

- `@RequestScoped (javax.enterprise.context)`
  - 1 Instanz pro Request (Web, EJB, JMS)
- `@SessionScoped (javax.enterprise.context)`
  - 1 Instanz pro HTTP Session (Web)
- `@ApplicationScoped (javax.enterprise.context)`
  - 1 Instanz pro Webanwendung

Scopes lt. Servlet-Spec.

- `@ConversationScoped (javax.enterprise.context)`
  - 1 Instanz pro Konversation / Flow
- `@Singleton (javax.inject)`
  - 1 Instanz pro CDI-Container
- `@Dependent (javax.enterprise.context)`
  - Lifecycle abhängig von der Instanz, in die Objekt injiziert wird

ARS

Contexts and Dependency Injection (JSR-299) [5|12]

`@ConversationScoped`

- Scope zwischen `@RequestScoped` und `@SessionScoped`
  - Dialog über mehrere Seiten für einen Benutzer
    (über mehrere Requests innerhalb einer Session)
- Einfachste Konversation: nur 1 Request (*Transient Conversation*)
- *Long-running Conversation*
  - Beginn mit `Conversation.begin()`
  - Ende mit `Conversation.end()`
  - Automatisches Beenden bei Zerstören der Session
  - Integration mit JSF (nicht: `@ViewScoped`)

ARS

Contexts and Dependency Injection (JSR-299) [6|12]

Passivation Capable Beans

- Kennzeichnung von Scopes, die vom Server ausgelagert werden können
  - @SessionScoped
  - @ConversationScoped

```
@NormalScope(passivating=true)
public abstract @interface SessionScoped {
}
```

```
@SessionScoped
public class Account implements Serializable { }
```

```
[ERROR   ] Passivation scoped defined bean must be passivation capable, but bean : Account,
Name:null, WebBeans Type:MANAGED, API Types:[de.ars.demos.web.Account,java.lang.Object],
Qualifiers:[javax.enterprise.inject.Any,javax.enterprise.inject.Default] is not passivation capable
```

ARS

Contexts and Dependency Injection (JSR-299) [7|12]

`@Singleton` – 1 Instanz pro EAR

- Schlussfolgerung aus
  - `@Singleton` – 1 Instanz pro CDI-Container
  - 1 CDI-Container pro EAR

`@RequestScoped` – Injection von Beans in Servlets?

```
@Inject
private Message message;
```

@RequestScoped!!!

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
  // ...
  out.println( message.getMessage() );
  // ...
}
```

Thread-Safety? Concurrency?

ARS

Contexts and Dependency Injection (JSR-299) [8|12]

`@RequestScoped` – Injection von Beans in Servlets?

```
@Inject
private Message message;
```

this	HelloWorldServlet (id=1069)
config	ServletConfig (id=1981)
message	Message_$$_javassist_0 (id=1088)
handler	RequestScopedBeanInterceptorHandler (id=1239)

- 1 Servlet-Instanz → 1 injiziertes Objekt
  - Proxy-Objekt mit Delegation an Instanzen pro Request (*Contextual Instances*)

Thread-Safety! Concurrency!

ARS

Contexts and Dependency Injection (JSR-299) [9|12]

Producer Methods

- Erzeugung von Beans durch Methoden anderer Beans (*Factory Method Pattern*)
  - Factory Beans haben eigenen Scope
  - Annotationen für Beans auch für Methoden anwendbar

```
@ApplicationScoped
public class MessageFactory {

  @Produces @RequestScoped
  public Message createMessage() {
    Message message = new Message();
    // ...
    return message;
  }

}
```

ARS

Contexts and Dependency Injection (JSR-299) [10|12]

Producer Methods – Annotation `@New`

- Erzeugung einer Bean durch CDI-Container
  - Dependency Injection durch CDI-Container
  - Producer Method konfiguriert das Objekt zusätzlich oder nutzt dieses Objekt zur Erzeugung der Bean

```
@Produces @RequestScoped
public Message createMessage( @New Message message ) {
  // ...
  return message;
}
```

ARS

Contexts and Dependency Injection (JSR-299) [11|12]

Producer Methods – Metadaten

- Zugriff auf Informationen über *Injection Point*
  - Reflection API
  - Qualifier

```
public class Kitchen {

  @Inject
  private Logger logger;

  // ...
}
```

```
@Produces
Logger createLogger(InjectionPoint ip) {
   return Logger.getLogger( ip.getMember().getDeclaringClass().getName() );
}
```

ARS

Contexts and Dependency Injection (JSR-299) [12|12]

Producer Fields

- Nutzen von Instanzvariablen
  - Keine Implementierung von Methoden, sondern Nutzen anderer (Nicht-CDI) Producer
  - Bereitstellung von Objekten (anderer Technologien) im CDI-Container

```
@Produces
@UserDatabase
@PersistenceContext
private EntityManager em;
```

```
@Inject
@UserDatabase
private EntityManager em;
```

Disposer Methods

- Methoden, die bei Zerstörung eines Objektes aufgerufen werden
  - Finalisierung

```
public void close(@Disposes @UserDatabase EntityManager em) {
  em.close();
}
```

ARS

Kontrollfragen

- Welcher Vorteil ergibt sich bei ausschließlicher Verwendung der Annotationen aus JSR-330 (Dependency Injection for Java)?

- Sie lassen sich per `@Inject` ein Objekt injizieren. Wie können Sie herausfinden bzw. beeinflussen, welchen Gültigkeitsbereich das Objekt besitzt?

ARS

Kontrollfragen

- Objekte werden nur initial injiziert. Evtl. Gültigkeitsbereiche werden hierbei bereits berücksichtigt. Was müssen Sie beachten, wenn eine Instanz innerhalb ihres Gültigkeitsbereichs zur Laufzeit ausgetauscht werden kann (z.B. OSGi Services) bzw. wenn diese Instanz *On Demand* erzeugt werden soll?