CDI - CONTEXTS AND DEXPENDENCY INJECTION

MICHAEL SCHAFFLER, CIIT-JAVATRAINING.AT

WAS IST CDI?

- **Ein API, das Dependency Injection und Contexts implementiert.**
- **Ein Standard, der Teil der Java EE 6 Spezifikation ist (JSR 299)**
- **Eine Referenzimplementierung von JBOSS (http://seamframework.org/Weld)**
- **Alternative zu anderen Dependency Injection Frameworks wie Spring oder Google Guice**
- **Ein "Ableger" des JBOSS Seam Frameworks**

WAS IST DEPENDENCY INJECTION?

- **Die Instanziierung von Applikations-Objekten wird nicht verteilt in der Applikation programmiert, sondern zentral von einem "Container" gesteuert.**
- **Die Referenzen zwischen den so instanziierten Objekten werden ebenfalls durch den Container gesteuert.**
- **Die Konfiguration des DI Containers erfolgt über**
  - Annotations (CDI)
  - XML Konfigurationen (bevorzugt von Spring)
  - Java Code (Google Guice)

- **CDI implementiert eine Typ-sichere Dependency Injection**

WAS BRINGT MIR DEPENDENCY INJECTION?

- **eine loose Kopplung zwischen Komponenten, da die Komponenten selbst kein Wissen darüber haben, mit welchen anderen Implementierungsklassen sie Assoziationen eingehen**
- **eine größere Flexibilität für unterschiedliche Konfigurationen einer Applikation (z.b. in Testszenarien)**
- **bessere Wartbarkeit der Applikation**

WAS SIND CONTEXTS?

- **In CDI bestimmt der Context den Lebenszyklus (Dauer und Sichtbarkeit) eines Objektes**
- **Der Entwickler konfiguriert den Scope eines Objektes:**
  - @ApplicationScoped (Gesamtdauer der Applikation)
  - @SessionScoped (Dauer einer Benutzer Session)
  - @ConversationScoped (Dauer einer Konversation)
  - @RequestScoped (Dauer eines Requests)
  - @Dependent (Abhängig vom Lebenszyklus einer übergeordneten Bean)
- **Mit dieser Information steuert der Context die Instanziierung und die Freigabe von Objekten**

LÄUFT CDI NUR IM JAVA EE APPSERVER?

Nein. Weld (CDI Referenzimplementierung) kann verwendet werden in

- **Java SE Applikationen**
- **Anwendungen, die im Tomcat Webcontainer laufen**
- **Java EE Applikationsserver (ab Java EE 6 verbindlich)**

WIE INTEGRIERT SICH CDI IN ANDERE FRAMEWORKS?

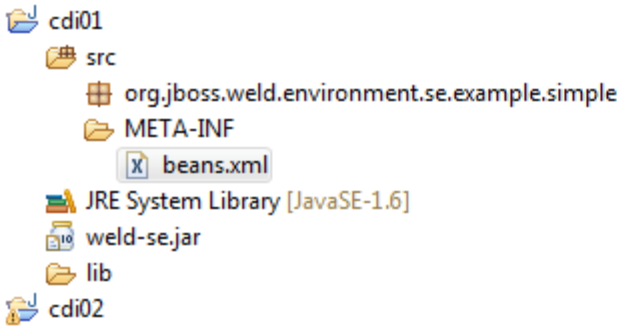
- **CDI ist stark integriert mit**
  - JSP Expression Language
  - Java Server Faces (inkl. Facelets, JSF Expression Language)
  - Enterprise JavaBeans

javatraining.at

CDI KONFIGURATION

Wie binde ich Weld in meine Anwendung ein?

- **weld.jar**
- **beans.xml muss existieren (sonst "startet" CDI nicht)**




```
<?xml version="1.0" encoding="UTF-8"?>
<beans></beans>
```

@INJECT

```
public interface KundenDAO {
    public Kunde getKundeNachKundennummer(Long kundennummer);
    public void speichereKunde(Kunde kunde);
    public Collection<Kunde> getAlleKunden();
}
```

```
@ApplicationScoped
public class InMemoryKundenDAO implements KundenDAO{
    Map<Long, Kunde> kundenMap = Collections.synchronizedMap(new HashMap<Long, Kunde>());

    @Override
    public Kunde getKundeNachKundennummer(Long kundennummer) {
        return kundenMap.get(kundennummer);
    }

    @Override
    public void speichereKunde(Kunde kunde) {
        kundenMap.put(kunde.getKundennummer(), kunde);
    }

    @Override
    public Collection<Kunde> getAlleKunden() {
        return kundenMap.values();
    }
}
```

@INJECT

```
public class Kundendienst {
    @Inject
    KundenDAO kundenDao;

    public Kundendienst() {
        super();
    }

    public void legeKundeAn(Kunde kunde) throws Exception{
        if (!pruefeKreditwuerdigkeit(kunde)){
            throw new Exception("nicht kreditwürdig");
        } else {
            kundenDao.speichereKunde(kunde);
        }
    }

    public Collection<Kunde> getAlleKunden(){
        return kundenDao.getAlleKunden();
    }

    private boolean pruefeKreditwuerdigkeit(Kunde kunde){
        boolean result = true;
        if (kunde.getName().equals("Joker")) result = false;
        if (kunde.getName().equals("Riddler")) result = false;
        return result;
    }

    public KundenDAO getKundenDao() {
        return kundenDao;
    }

    public void setKundenDao(KundenDAO kundenDao) {
```

@INJECT

@Inject KundenDAO dao;

bewirkt, dass der Container nach einer Klasse sucht, die das KundenDAO interface implementiert, diese instanziiert und in dem Attribut dao zuweist (injiziert).

Was passiert, wenn es unterschiedliche Klassen gibt, die das KundenDAO Interface implementieren?

-> Fehler beim Start der Anwendung

-> ich muss dem Container sagen, welche Implementierung zum Zuge kommt

@ALTERNATIVE

javatraining.at

```java
@Alternative
@ApplicationScoped
public class InMemoryKundenDAO implements KundenDAO{
    Map<Long, Kunde> kundenMap = Collections.synchronizedMap(new HashMap<Long, Kunde>());

    @Override
    public Kunde getKundeNachKundennummer(Long kundennummer) {
        return kundenMap.get(kundennummer);
    }

    @Override
    public void speichereKunde(Kunde kunde) {
        kundenMap.put(kunde.getKundennummer(), kunde);
    }
```

```java
@Alternative
@ApplicationScoped
public class JPADao implements KundenDAO {
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");

    @Override
    public Kunde getKundeNachKundennummer(Long kundennummer) {
        EntityManager em = emf.createEntityManager();
        Kunde result = null;
        try {
            result = em.find(Kunde.class, kundennummer);
        } finally {
            em.close();
        }
        return result;
    }
```

```
Exception in thread "main" org.jboss.weld.exceptions.DeploymentException: WELD-001408 Unsatisfied dependencies for type [KundenDAO
    at org.jboss.weld.bootstrap.Validator.validateInjectionPoint(Validator.java:305)
    at org.jboss.weld.bootstrap.Validator.validateBean(Validator.java:139)
    at org.jboss.weld.bootstrap.Validator.validateRIBean(Validator.java:162)
```

@ALTERNATIVE


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
    <alternatives>
        <class>at.java.integration.InMemoryKundenDAO</class>
    </alternatives>
</beans>
```

QUALIFIER

Was macht man, wenn der Typ der Bean nicht ausreicht, um die Injection zu definieren?

Wir definieren einen Qualifier:

```
@Qualifier
@Target({ElementType.TYPE,ElementType.METHOD,ElementType.PARAMETER,ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface InMemory {}
```

Annotieren die Implementierungsklasse mit dem Typ:

```
@ApplicationScoped
@InMemory
public class InMemoryKundenDAO implements KundenDAO{
    Map<Long, Kunde> kundenMap = Collections.synchronizedMap(new HashMap<Long, Kunde>());
```

Und kombinieren ihn mit @Inject:

```
public class Kundendienst {
    @Inject @InDatabase
    KundenDAO kundenDao;

    @Inject @InMemory
    KundenDAO failoverDAO;

    public Kundendienst() {
        super();
    }
```

WIE INJIZIERE ICH Z.B. EINE LIST?

Da List eine JDK Klasse ist, kann ich sie ja mit keiner @Alternative oder einem Qualifier versehen.

Oder: Wie injiziere ich ein Objekt, das ich aus der Datenbank oder eine Konfigurationsdatei gelesen habe?

-> Es gibt auch Producer Methoden, über die ich programmatisch Instanzen für die DI bereitstellen kann.

PRODUCER METHODEN UND FELDER


```
public class Kundendienst {
    @Inject @InDatabase
    KundenDAO kundenDao;

    @Inject @InMemory
    KundenDAO failoverDAO;


    public @Produces Collection<Kunde> getAlleKunden(){
        return kundenDao.getAlleKunden();
    }
```

```
@Dependent
public class PrintService {
    @Inject Collection<Kunde> alleKunden;

    public void printAlleKunden(){
        System.out.println("Alle Kunden");
        for (Kunde kunde:alleKunden){
            System.out.println(kunde);
        }
    }

    public Collection<Kunde> getKundenCollection(){
        return alleKunden;
    }
}
```

@NAMED BEANS

Für den Zugriff auf Beans in Web Anwendungen, können diese benannt werden.

Dieser Name kann dann in der JSF bzw. JSP Expression Language verwendet werden.

```
@Named(value="calculatorBean")
@ApplicationScoped
public class CalculatorBean {
    @Inject
    private CalcFormBean calcFormBean;

    @Inject
    private CalcHistoryBean historyBean;

    @EJB
    private CalculatorServiceLocal calcService;

    public String add(){
        int sum = calcService.add(calcFormBean.getA(), calcFormBean.getB());

        calcFormBean.setSum(sum);
        historyBean.addHistoryEntry(calcFormBean);

        if (sum>=15) return "bigCalc.xhtml";
        else return "index.xhtml";
    }
```

@NAMED BEANS

```
<h:body>
    <h3>Super Calculator</h3>
    <h:form>
        <table>
            <tr>
                <td><h:outputLabel value="A=" for="inputA"/></td>
                <td><h:inputText id="inputA" value="#{calcFormBean.a}"
                                 converterMessage="Please provide a valid number"/>
                    <h:message style="color: red" for="inputA"/>
                </td>
            </tr>
            <tr>
                <td><h:outputLabel value="B=" for="inputB"/></td>
                <td><h:inputText id="inputB" value="#{calcFormBean.b}"
                                 converterMessage="Please provide a valid number"/>
                    <h:message style="color: red" for="inputB"/>
                </td>
            </tr>
            <tr>
                <td><h:outputLabel value="Sum="/></td>
                <td><h:outputText value="#{calcFormBean.sum}"/></td>
            </tr>
            <tr>
                <td colspan="2"><h:commandButton value="Add" action="#{calculatorBean.add}"/></td>
            </tr>
        </table>
    </h:form>
    <ul>
        <ui:repeat value="#{calcHistoryBean.history}" var="entry">
            <li><h:outputText value="#{entry}"/></li>
        </ui:repeat>
```

@NAMED BEANS

@Named kann auch für die Injection verwendet werden:

```
@Qualifier
@Target({ElementType.TYPE,ElementType.METHOD,ElementType.PARAMETER,ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Environment {}
```

```
@Alternative
@Environment
public class TestConstants {
    @Produces @Named("host") String host = "testHost";
    @Produces @Named("port") String port = "testPort";
}
```

```
@Alternative
@Environment
public class ProductionConstants {
    @Produces @Named("host") String host = "myHost";
    @Produces @Named("port") String port = "myport";
}
```

```
public class KundenAnwendung {
    @Inject @Named("port") String port;
    @Inject @Named("host") String host;

    @Inject
    Kundendienst kundendienst1;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
    <alternatives>
        <class>at.java.integration.JPADao</class>
        <class>at.java.service.TestConstants</class>
    </alternatives>
</beans>
```

@INTERCEPTOR

Interceptoren sind Beans, die für die Realisierung von Querschnittsaufgaben verwendet werden:

- **Logging**
- **Transaktionssteuerung**
- **Security**
- **...**

Sie hängen sich zwischen den Aufrufer und den Aufgerufenen und "intercepten" den Aufruf.

javatraining.at

@INTERCEPTOR

```java
@InterceptorBinding
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface Logged {}
```

```java
@Interceptor @Logged
public class LoggingInterceptor {
    @AroundInvoke
    public Object logTheCall(InvocationContext ctx) throws Exception{
        Logger logger = Logger.getLogger(ctx.getTarget().getClass().getName());
        logger.info("call enter");
        Object result = ctx.proceed();
        logger.info("call exit");
        return result;
    }
}
```

```
    <interceptors>
        <class>at.java.interceptors.LoggingInterceptor</class>
    </interceptors>
</beans>
```

```java
@ApplicationScoped
@Logged
public class Kundendienst {
    @Inject @InDatabase
    KundenDAO kundenDao;

    @Inject @InMemory
    KundenDAO failoverDAO;


    public @Produces Collection<Kunde> getAlleKunden(){
        return kundenDao.getAlleKunden();
    }
```

@STEREOTYPE

Über Stereotypes können beliebige Kombinationen von CDI Annotations kombiniert unter einem Qualifier verwendet werden

```
@ApplicationScoped
@Logged
@Stereotype
@Target({ElementType.TYPE,ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Service {}
```

```
@Service
public class Kundendienst {
    @Inject @InDatabase
    KundenDAO kundenDao;

    @Inject @InMemory
    KundenDAO failoverDAO;
```

EVENTS

javatraining.at

Events ermöglichen eine asynchrone 1:N broadcast Kommunikation zwischen Beans. Der Sender kennt die Empfänger nicht und umgekehrt.

Wir definieren wieder einen Qualifier, diesmal um Events zu klassifizieren

```
@Qualifier
@Target({ElementType.TYPE,ElementType.METHOD,ElementType.PARAMETER,ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface News {}
```

Dann schreiben wir einen Event Producer

```
public class KundenAnwendung {

    @Inject @News Event<String> newsEvent;

    public void starteAnwendung(@Observes ContainerInitialized init) {
        newsEvent.fire("Die Anwendung ist gestartet");
```

Und einen Event Receiver

```
public class EventReceiver {
    public void reagiereAufEvent(@Observes @News String eventNachricht){
        System.out.println("Habe folgenden Event empfangen:" + eventNachricht);
    }
}
```

LITERATUR

JSR-299: Contexts and Dependency Injection for the Java EE platform, http://jcp.org/en/jsr/summary?id=299

Weld - JSR-299 Reference Implementation, http://seamframework.org/Weld