jax® 2012

Business Technology | Days

Mark Struberg / INSO, TU Vienna

CDI Introduction and Outlook

About Myself

- struberg@apache.org
- http://github.com/struberg
- freelancer, programmer since 20 years
- Apache Software Foundation member

PMC/Committer in various Apache projects: OpenWebBeans, MyFaces, BVAL, OpenJPA, Maven, DeltaSpike, ...

- CDI EG member

Agenda

- JSR-299 Overview
- Basic CDI Concepts
- Producers
- Alternatives
- Interceptors
- Events
- Extensions
- JSR-346 (CDI-1.1) preview

JSR-299 Overview

- Contexts and Dependency Injection for the Java EE platform (CDI)
- Component Model for Java (SE and EE)
- The spec formerly known as „WebBeans“
- Started ~2007
- Originally designed for J2EE but also usable for standalone applications

CDI Features

- Typesafe Dependency Injection
- Interceptors
- Decorators
- Events
- SPI for implementing “Portable Extensions”
- Unified EL integration

Available Implementations

- JBoss Weld (RI)
- Apache OpenWebBeans
- Resin CanDI
- maybe Spring in the future?

Basic CDI Concepts

What is Dependency Injection?

- Uses Inversion Of Control pattern for object creation
- Hollywood Principle: don't call us, we call you
- No more hardcoded dependencies when working with Interfaces

```
MailService ms = new VerySpecialMailService();
```

Why use Dependency Injection

- Easy to change implementations
- Encourage Separation of Concerns
- Minimise dependencies
- Dynamic Object retrieval
- Makes Modularisation easy
- Simplifies Reusability

Valid Bean Types

- Almost any plain Java class (POJO)
- EJB session beans
- Objects returned by producer methods or fields
- JNDI resources (e.g., DataSource)
- Persistence Units/Persistence Contexts
- Web service references
- Remote EJB references
- Additional Types defined via SPI

A small Example

- Create a **META-INF/beans.xml** marker file
- Create a MailService implementation

```
@ApplicationScoped
public class MyMailService
implements MailService {
  public send(String from, String to,
              String body) {
    .. do something
  }
```

A small Example

- We also need a User bean

```
@SessionScoped
@Named
public class User {
  public String getName() {..}
  ..
}
```

A small Example

- Injecting the Bean

```
@RequestScoped
@Named(„mailForm“)
public class MailFormular {
  private @Inject MailService mailSvc;
  private @Inject User usr;
  public String sendMail() {
    mailSvc.send(usr.getEmail(),
                 „other“, „sometext“);
    return “successPage”;
  }
```

A small Example

- Use the beans via Expression Language

```
<h:form>
  <h:outputLabel value="username"
   for"username"/>
  <h:outputText id="username"
   value="#{user.name}"/>
  <h:commandButton value="Send Mail"
     action="#{mailForm.sendMail}"/>
</h:form>
```

How DI containers work

- Contextual Instance factory pattern
- Someone has to trigger the instance creation - even in a DI environment
- But the trigger has no control over the Implementation class nor Instance

'Singletons', Scopes, Contexts

- Each created Contextual Instance is a 'Singleton' in a well specified Context
- The Context is defined by it's Scope
  - @ApplicationScoped ->ApplicationSingleton
  - @SessionScoped -> SessionSingleton
  - @ConversationScoped -> ConversationSingleton
  - @RequestScoped -> RequestSingleton
  - ... code your own

Built In Scopes

- @ApplicationScoped
- @SessionScoped
- @RequestScoped
- @ConversationScoped
- All those scopes are of type @NormalScope

Special Scopes

- @Dependent
  - Pseudo scope
  - If injected, the Contextual Instance will get/use the Context of it's parent
- @Dependent is assumed if no other Scope is explicitly assigned to a class!

Terms – „Managed Bean“

- The term Managed Bean means a Java Class and all it's rules to create contextual instances of that bean.
- Interface

  `Bean<T> extends Contextual<T>`
- 'Managed Beans' in JSR-299 and JSR-346 doesn't mean JavaBeans!

Terms – Contextual Instance

- The term 'Contextual Instance' means a Java instance created with the rules of the Managed Bean `Bean<T>`
- Contextual Instances usually don't get injected directly!

Terms – Contextual Reference

- The term 'Contextual Reference' means a proxy for a Contextual Instance.
- Proxies will automatically be created for injecting @NormalScope beans.

How Proxies work

- 'Interface Proxies' vs 'Class Proxies'
- Generated subclasses which overload all non-private, non-static methods.

```
public doSthg(parm) {
  getInstance().doSthg();
}
private T getInstance() {
  beanManager.getContext().get(...);
}
```

- Can contain Interceptor logic

Why use Proxies

- scope-differences - this happens e.g. if a @SessionScoped User gets injected into a @ApplicationScoped MailService.
- Passivation of non-serializable contextual instances (via it's contextual reference)
- Interceptors, Decorators

The big picture

- Creating the meta information at startup
  – Classpath Scanning and creating Managed Beans meta-information
- Contextual Instance creation at runtime
  – Based on the Managed Beans, the Context will maintain the instances
- Well defined contextual instance termination

homework: some hacking

- Create a new project from a maven archetype

```
$> mvn archetype:generate -DarchetypeCatalog=
http://people.apache.org/~jakobk/m2_archetypes_103_release
```

- Select 'myfaces-archetype-codi-jsf20'
- start the web application in jetty

```
$> mvn clean install -PjettyConfig jetty:run-exploded


$> tree src
```

Qualifiers

- A Qualifier enables the lookup of specific Implementations at runtime
- Qualifier connects 'creation info' with InjectionPoints.
- Kind of a typesafe 'naming' mechanism

Built-in Qualifiers

- **@Named** – used to supply an EL name
- **@Default** - Assumed, if no other Qualifier (beside @Named) is specified
- **@Any** – matches all Qualifiers

Using Qualifiers

- Qualifiers can be used to distinct between beans of the same Type

  private @Inject **@Favourite** Track;

  private @Inject **@Recommended** Track;

- Special Qualifier @Named for defining EL names.

Define your own Qualifiers

- Creating own annotations will get your daily bread and butter

```
@Qualifier
@Retention(RUNTIME)
@Target({TYPE, METHOD, FIELD, PARAMETER})
public @interface Favourite {}
```

Producers

- Write producer methods or producer fields if more logic is needed at instance creation time

```
@ConversationScoped
public class Playlist {
  @Produces @Favourite @RequestScoped
  public Track getFavTrack() {
    return new Track(favTitle)
  }
public void drop(@Disposes @Favourite Track t)
  { dosomecleanup(t);}
}
```

@Specializes

- 'Replaces' the Managed Bean of a superclass
- Bean must extends the superclass
- possible to specialize Producer Methods
- or even EJBs:

```
@Stateless
public class UserSvcFacade implements UserSvc
 { ... }
@Stateless @Mock @Specializes
public class MockUserSvcFacade extends
UserSvcFacade
 { ... }
```

@Alternative

- Swap Implementations for different installations
- Kind of an optional override of a bean

```
@Alternative
public class MockMailService
implements MailService { ...
```

- Disabled by default

```
<beans>
  <alternatives>
    <class>org.mycomp.MockMailService</class>
```

@Stereotype

- Stereotypes are used to combine various attributes and roles
  - A default scope
  - A default naming scheme
  - Set of InterceptorBindings and @Alternative behaviour
  - can be meta-annotated with other Stereotypes!

@Stereotype example

```
@Stereotype
@ApplicationScoped
@Transactional
@Secured
@Target(TYPE)
@Retention(RUNTIME)
public @interface Service {}
```

Interceptors

- An Interceptor decouples technical concerns from business logic
- Applying cross cutting concerns to beans
- JSR-299 allows you to write your own interceptors
- Interceptors are treated as being @Dependent to the intercepted class

Interceptor usage

- Interceptor annotations can be applied on method or class level

```
@ApplicationScoped
public class UserService {
  @Transactional
  public storeUser(User u) {
    ...
  }
}
```

InterceptorBinding Type

- InterceptorBindings are used to identify which Interceptors should be applied to a bean

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

The Interceptor Impl

```
@Interceptor @Transactional
public class TransactionalInterceptor {
  private @Inject EntityManager em;
  @AroundInvoke
  public Object invoke(InvocationContext context)
  throws Exception {
    EntityTransaction t = em.getTransaction();
    try {
      if(!t.isActive()) t.begin();
      return context.proceed();
    } catch(Exception e) {
      .. rollback and stuff
    } finally {
      if(t != null && t.isActive())
        t.commit();
    }
  }
}
```

Enabling Interceptors

- Interceptors need to be **enabled manually** in beans.xml
- You can define the **order** in which multiple Interceptors stack

```
<beans>
  <interceptors>
    <class>org.mycomp.Secured</class>
    <class>org.mycomp.Transactional</class>
  </interceptors>
</beans>
```

Events

- Observer/Observable pattern
- Inject a Event producer

```
private @Inject Event<UserLoggedIn> loginEvent;
private @Inject User usr
loginEvent.fire( new UserLoggedIn(user) );
```

- Event Consumer

```
void onLogin(@Observes UserLoggedIn ule) {
  ule.getUser().getName; ...
}
```

- Attention: don't use private observer methods!

Writing own Extensions

- CDI Extensions are **portable**
- and **easy to write**!
- Are **activated by** simply **dropping** them **into** the **classpath**
- CDI Extensions are based on java.util.ServiceLoader Mechanism

Popular Extensions

- Apache MyFaces CODI
  http://myfaces.apache.org/extensions/cdi
- JBoss Seam3
  http://seamframework.org/Seam3
- CDISource
  https://github.com/cdisource
- Apache DeltaSpike
  http://incubator.apache.org/deltaspike

What's New in CDI-1.1

- Globally enabled Interceptors and Alternatives
- XML configuration ???
- @Disposes for producer fields
- Clarifies some AnnotatedType behaviour
- @Veto + optional beans
- Relax Serialization rules
- @EarApplicationScoped vs @WebApplicationScoped
- https://issues.jboss.org/browse/CDI

Common Pitfalls & New Programming Patterns

AmbiguousResolutionException

- Happens if multiple Bean<T> serve the same InjectionPoint.
- MenuItem example:

```
public class MenuItem implements Serializable {
  private String id;
  private String name;
  private MenuItem parent;
  private MenuItem[] childs;
  ...
}
```

The MenuItem Producer

- We use a producer method to create the menu tree

```
@ApplicationScoped
public class MenuProducer {
  private @Inject MenuService menuSvc;
  @Produces @SessionScoped
  public MenuItem createMenuItem(User usr){
    return menuSvc.getMenuOfUser(usr);
  }
}
```

@Typed

- Defines the Java Type which should be used for that bean
- Possible to 'disable' a bean with @Typed() to prevent AmbiguousResolutionExceptions

```
@Typed() // basically disables this Bean
public class MenuItem implements Serializable {
```

- handy for subclasses which should NOT conflict with their parent class types

Explicitly @Typed

```
@ApplicationScoped
public class TrackReadOnlyService {
  public Track readTrack(int id) {..}
}

@Typed(TrackReadWriteService.class)
@ApplicationScoped
public class TrackReadWriteService
extends TrackReadOnlyService {
  public void writeTrack(Track t) {..}
}
```

Cache cleaning via Events

- If user changes language or logs in, then we need to change the menu

```
@SessionScoped
public class Menu {
  private @Inject MenutItem parentMenu;
  protected reloadMenu(@Observes
UserSettingsChangedEvend uscE) {
    parentMenu = menuSvc.loadMenu();
  }
}
```

Typesafe Configuration

- Config without properties or XML files
- Just create an Interface or default config class

```
public interface MyConfig{
   String getJdbcUrl();
}
```

- Later just implement, @Specializes, @Alternative, @ProjectStageActivated, @ExpressionActivated your config impl.

@Nonbinding

- @Nonbinding excludes annotation parameters from the comparison

```
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target( { TYPE, METHOD })
public @interface Transactional {
  @Nonbinding boolean requiresNew()
                              default false;
}
```

Behaviour Mutation

- Difference between cached and uncached version of the same service:

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

The Services

```
@ApplicationScoped
public class BusService {
  Bus getBus(int id) {..}
}
```

@Cached

```
@ApplicationScoped
public class CachedBusService
extends BusService { .. }
```

Dynamic Resolving

```
@Inject @Any
private Instance<BusService> busSvcSource;
private BusService busSvc;
..
@PostConstruct
protected init() {
  busSvc = busSvcSource.select(
    user.isAdmin()
      ? new AnnotationLiteral<Default>() {}
      : new AnnotationLiteral<Cached>() {}
          ).get();
}
```

Documentation

- JSR-299 spec:

  http://jcp.org/en/jsr/detail?id=299

  http://jcp.org/en/jsr/detail?id=346

- OpenWebBeans:

  http://openwebbeans.apache.org

  svn co \ https://svn.apache.org/repos/asf/openwebbeans/trunk

- Weld:

  http://seamframework.org/Weld

Legal stuff

- Apache, OpenWebBeans, MyFaces, OpenEJB and OpenJPA are trademarks of the Apache Software Foundation
- Seam3 and Weld are trademarks of JBoss inc
- CanDI is a trademark of Caucho Inc