



## Powering your application with Apache DeltaSpike and CDI

Mark Struberg, INSO

Department for Industrial Software,

Technical University Vienna

# About the Speaker

- <http://github.com/struberg>
- <http://people.apache.org/~struberg>
- Apache Software Foundation member
- Apache OpenWebBeans, MyFaces, BVAL, OpenJPA, Maven, DeltaSpike, ...
- CDI EG member

# Agenda

- CDI Quickstart
- CDI Extensions Howto
- DeltaSpike History and Future
- `deltaspike-core`
- `deltaspike-cdictrl`
- `deltaspike-jpa` & `deltaspike-data`
- `deltaspike-jsf`

# About Apache DeltaSpike

- Apache Project since 2012, TLP since 2013-04
- Homepage:  
<http://deltaspike.apache.org>
- Source:  
<http://git.apache.org/deltaspike>
- [dev@deltaspike.apache.org](mailto:dev@deltaspike.apache.org)
- [irc.freenode.net](http://irc.freenode.net) channel #deltaspike
- <https://issues.apache.org/jira/browse/DELTASPIKE>

# CDI Basics

# JSR-299 Overview

- Contexts and Dependency Injection for the Java EE platform (CDI)
- Component Model for Java (SE and EE)
- Originally designed for J2EE but also usable for standalone applications
- Based on JSR-330

# CDI Features

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

# Available Implementations

- JBoss Weld (RI)
- Apache OpenWebBeans
- Resin CanDI
- Every JavaEE 6 container!
- Available CDI-1.1 implementations
  - Weld-2.0
  - OpenWebBeans-2.0 in preparation



# A small Example

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

**@ApplicationScoped**

```
public class MyMailService
implements MailService {
    private @Inject User usr;
    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

- Uses Inversion Of Control pattern for object creation: “Macho Principle”
- Contextual Instance **factory pattern**
  - recursively fill all InjectionPoints
- 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 ...

# Portable Extensions

# 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

```
META-INF/services/  
javax.enterprise.inject.spi.Extension
```



# Extension Basics

- During the boot, the BeanManager sends **CDI Lifecycle Events** to the Extensions
- CDI Extensions can do anything you can do with Annotations. ... and even more!
- **Attention:**  
strictly distinguish between boot and runtime!

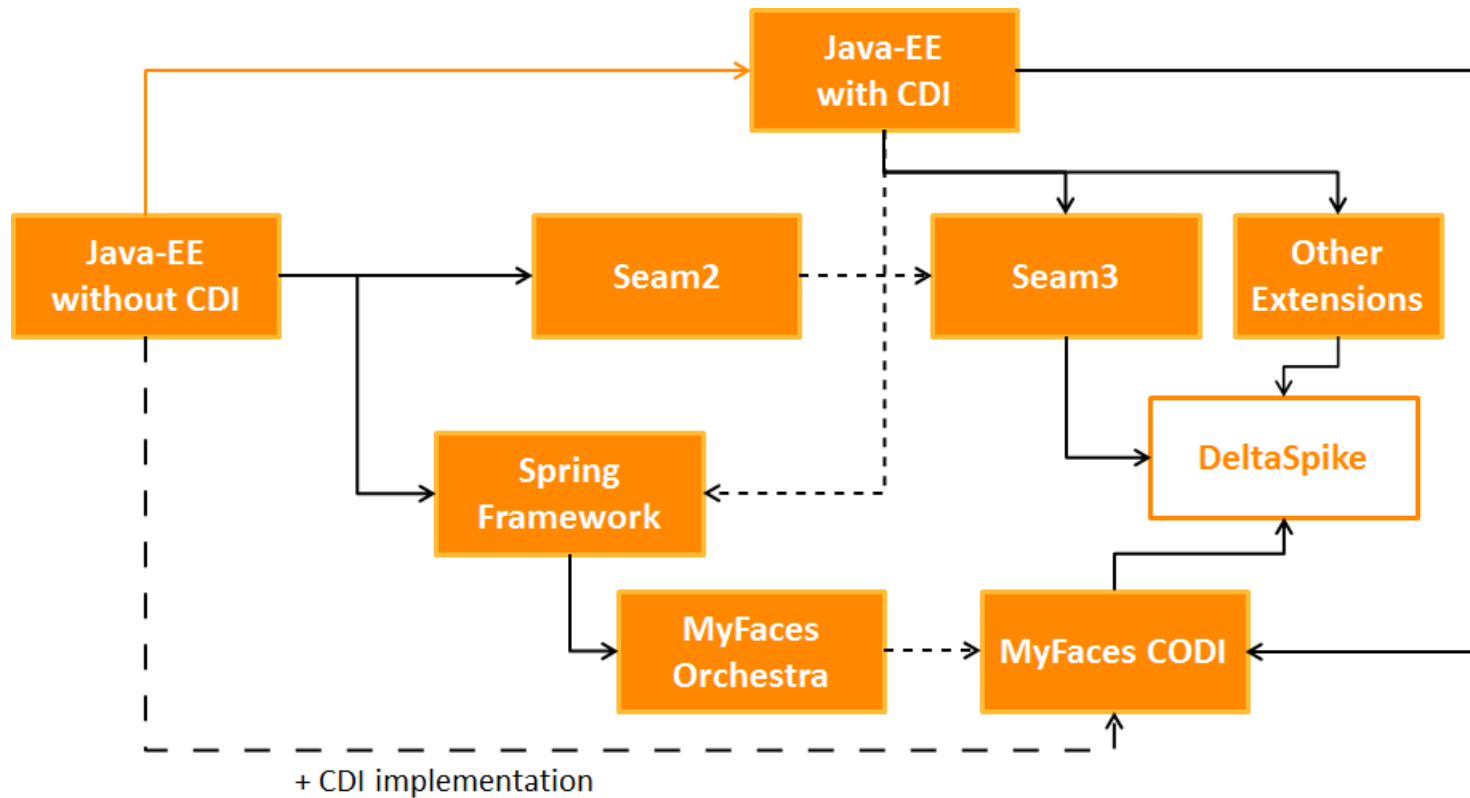
# Extension Lifecycle Events

- Container sends CDI 'system events' which all Extensions can @Observes:
  - BeforeBeanDiscovery
  - ProcessAnnotatedType
  - ProcessInjectionTarget
  - AfterBeanDiscovery
  - AfterDeploymentValidation
  - BeforeShutdown
- eat your own dog food: using CDI events for Extensions

# Popular Extensions

- Apache MyFaces CODI  
<http://myfaces.apache.org/extensions/cdi>
- JBoss Seam3  
<http://seamframework.org/Seam3>
- Apache DeltaSpike  
<https://cwiki.apache.org/confluence/display/DeltaSpike/Index>  
<http://deltaspike.apache.org>

# Apache DeltaSpike History



# DeltaSpike Status Quo

- CORE - finished
- JPA - finished
- CdiCtrl - finished
- JSF - 95%
  - DeltaSpike already heavily gets used in Production!
- Used as base for projects like JBoss AearoGear, Apache Camel and others.

# DeltaSpike Future

- publish 1.0 soon
- Finish JSF module
- Improve data module
- Spring Bridge
- 3-monthly releases
- What else do you need?  
Please add it to our feature backlog!

# Apache DeltaSpike Core

# DeltaSpike CORE

- Solves Core Java problems
- Has no external dependencies
- Especially no Java EE dependencies
- Base for all other DeltaSpike modules
- Too many features to showcase in this talk...



# DeltaSpike CORE Features

- ProjectStage
- Configuration
- Catch Handler
- Utils
- BeanProvider/BeanManagerProvider
- conditional Exclude
- Typesafe Messages
- ClassDeactivator

# DeltaSpike Configuration

- The Problem
  - we don't like to repackage WAR, EAR, for changing our application configuration
- The Solution
  - default configuration
  - plus optional plugin configuration
  - plus optional external configuration
  - plus, plus, plus, ...
  - from lots of different `ConfigSources`

# DeltaSpike Configuration

- Pick up information from various places (ConfigSource)
- Ordinal based (`deltaspike_ordinal=..`): higher number -> more important
- Extensible, register your own ConfigSource

# Using the Configuration

- `ConfigResolver.getPropertyValue(...)`
- `.getProjectStageAwarePropertyValue(...)`
- `.getPropertyAwarePropertyValue(...)`
- You can also inject the config:

```
@Inject
```

```
@ConfigProperty(name="printerId")
```

```
private String printerId;
```

# Own property ConfigSource

- Register your own via ServiceLoader
- Easy mechanism for property files:
  - implement PropertyFileConfig
  - picks up all property files with the name returned in `getPropertyFileName()` and registers them as ConfigSource

# own property config

```
public class NfcCardProdConfigDefinition
implements PropertyFileConfig {
    @Override
    public String getPropertyFileName() {
        return "nfcCardProd.properties";
    }
}
```

# Config Ordinals

- PropertyFileConfigSource: 100  
META-INF/apache-deltaspike.properties
- LocalJndiConfigSource: 200  
java:comp/env/deltaspike/..
- EnvironmentPropertyConfigSource: 300
- SystemPropertyConfigSource: 400
- your own config defaults to 1000

# Vetoing Beans

- Can be used to dynamically disable a Bean
- Most powerful in combination with
  - `@Alternative`
  - `@Specializes`
- internally uses:  
`ProcessAnnotatedType#veto()`
- Inspired CDI-1.1 feature '`@Vetoed`'



# DeltaSpike @Exclude

- Used to disable bean depending on a certain situation
  - @Exclude()
  - @Exclude(ifProjectStage=ProjectStage.Production.class)
  - @Exclude(exceptIfProjectStage=ProjectStage.Development.class)
  - @Exclude(onExpression="dbVendor!=oracle")

# BeanManagerProvider

- Access to the BeanManager in places where it cannot get injected
  - JPA Entity Listeners
  - JSF Components
  - JMX Beans
  - ServletListener or ServletFilter if not running in EE Server
- Will be available in CDI-1.1:

`CDI.current()`

# BeanProvider

- internally uses BeanManagerProvider
- #getContextualReference(String);
- #getContextualReference(Type, Qualifiers...);
- #injectFields Used to fill all injection points of an existing instance

# Core Utility Classes

- Helpers for writing own CDI Extensions
  - BeanBuilder
  - AnnotatedTypeBuilder
  - AnnotationInstanceProvider
    - allows to dynamically create Annotation instances + any values
- Modularisation Helpers
  - globalAlternatives (see CDI-18 for info)

# Apache DeltaSpike CdiCtrl

# Container Bootstrap

- Allows to boot CDI containers with a vendor independent API
- Implementations for:
  - Apache OpenWebBeans
  - JBoss Weld
  - Apache OpenEJB (TomEE embedded)
  - add your own
- Simply replace the impl jar to switch!

# Container Bootstrap

- Very usable for unit tests, batches or other standalone Java processes:

```
CdiContainer cdiContainer =  
    CdiContainerLoader.getCdiContainer();  
cdiContainer.boot();  
cdiContainer.getContextControl().  
    startContexts();
```

# Context Control

- Also usable in EE containers
- Usage:  
`@Inject ContextControl ctxCtrl;`
- Allows to attach dummy RequestContext, SessionContext etc to the current Thread.
- Usable for Quartz extensions or any other async work



# Apache DeltaSpike JPA

# 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.Transactionnal</class>  
  </interceptors>  
</beans>
```

# DeltaSpike @Transactional

- Handles native EntityManager handling
- Handles UserTransaction handling
  - via pluggable TransactionStrategy
- EntityManager can have any scope:
  - @RequestScoped
  - @TransactionScoped

# EntityManagerFactoryProducer

- Uses PersistenceConfigurationProvider to pick up config Properties
- Add a producer for EntityManager

```
public @ApplicationScoped class EntityManagerProducer {  
    private @Inject @UnitName("MyPU") EntityManagerFactory emf;  
  
    @Produces @RequestScoped  
    public EntityManager createEM() {  
        return emf.createEntityManager();  
    }  
    public void dispose(@Disposes EntityManager em) {  
        em.close();  
    }  
}
```



# DeltaSpike data

- new module in deltaspike-0.5

## **@Repository**

```
public abstract class MyUserRepository extends
    AbstractEntityRepository<MyUser, Long> {
    public List<MyUser> implementedQryByName(String name) {
        String query = "select u from MyUser u where u.name = :name";
        return entityManager().createQuery(query, Simple.class)
            .setParameter("name", name).getResultList();
    }
}
```

```
@Query(named = Simple.BY_NAME_ENABLED, max = 1)
```

```
public abstract List<MyUser>
```

```
    findByNameNamedQueryIndexed(String name, Boolean enabled);
```

# Apache DeltaSpike JSF

# Features

- Typesafe JSF Messages
- Typesafe Navigation
- PageBeans
- `@JsfPhaseListener`
- Browser Tab Support
- JSF related CDI events
- New Scopes
- ongoing JSF-2.2 integration

# Typesafe Messages

```
@MessageContextConfig(  
    messageSource="myproject.properties")  
@MessageBundle  
public interface UserMessage {  
    @MessageTemplate("{hellomsg}")  
    String sayHello(String name);  
}
```

# Typesafe JSF Messages

```
@Inject
private JsfMessage<UserMessage> msg;

String doAction() {
    msg.addInfo().simpleMessageNoParam();
    msg.addFatal()
        .accessNotAllowed(userName, resource);

    // or manually:
    String message = msg.get().someMsg();
}
```

# DeltaSpike Scopes

- @WindowScoped
- @ViewAccessScoped (in 0.6)
- @ConversationScoped (CODI conv., in 0.6)
- @ViewScoped
- @RenderScoped

# Typesafe JSF Navigation

```
@Page(navigation = REDIRECT)
interface Pages extends ViewConfig {
    class Index implements Pages{}

    @Page(viewParams = INCLUDE)
    @Secured(AdminVoter.class)
    class Admin implements Pages {}

    // 'navigation' overruled
    @Page(navigation = FORWARD)
    class Home implements Pages {}

    class CustomErrorPage extends DefaultErrorView {}
}
```

# Using Typesafe Navigation

```
@Model
```

```
public class MyBackingBean {  
    public Class<? extends ViewConfig> overview(){  
        return Pages.Overview.class;  
    }  
  
    //navigates to /pages/customErrorPage.xhtml  
    public Class<? extends ViewConfig> showError() {  
        return DefaultErrorView.class;  
    }  
}
```



# JSF Lifecycle Events

- `doSthg(@Observes @AfterJsfRequest FacesContext ct) {`
- `doSthg(@Observes @BeforeJsfRequest FacesContext ct) {`
- `doSthg(@Observes @BeforePhase(PhaseId.RESTORE_VIEW) FacesContext ct) {`
- `doSthg(@Observes @AfterPhase(PhaseId.RESTORE_VIEW) FacesContext ct) {`

# 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