CDI Portable Extensions and Apache DeltaSpike

Marek Schmidt

ČVUT
2012-12-06

Portable CDI Extensions

- Providing own beans, interceptors and decorators to the container
- Injecting dependencies into its own objects using the dependency injection service
- Providing a context implementation for a custom scope
- Augmenting or overriding the annotation-based metadata with metadata from some other source

What portable CDI extensions cannot do

- Modify beans at run time

Creating an Extension

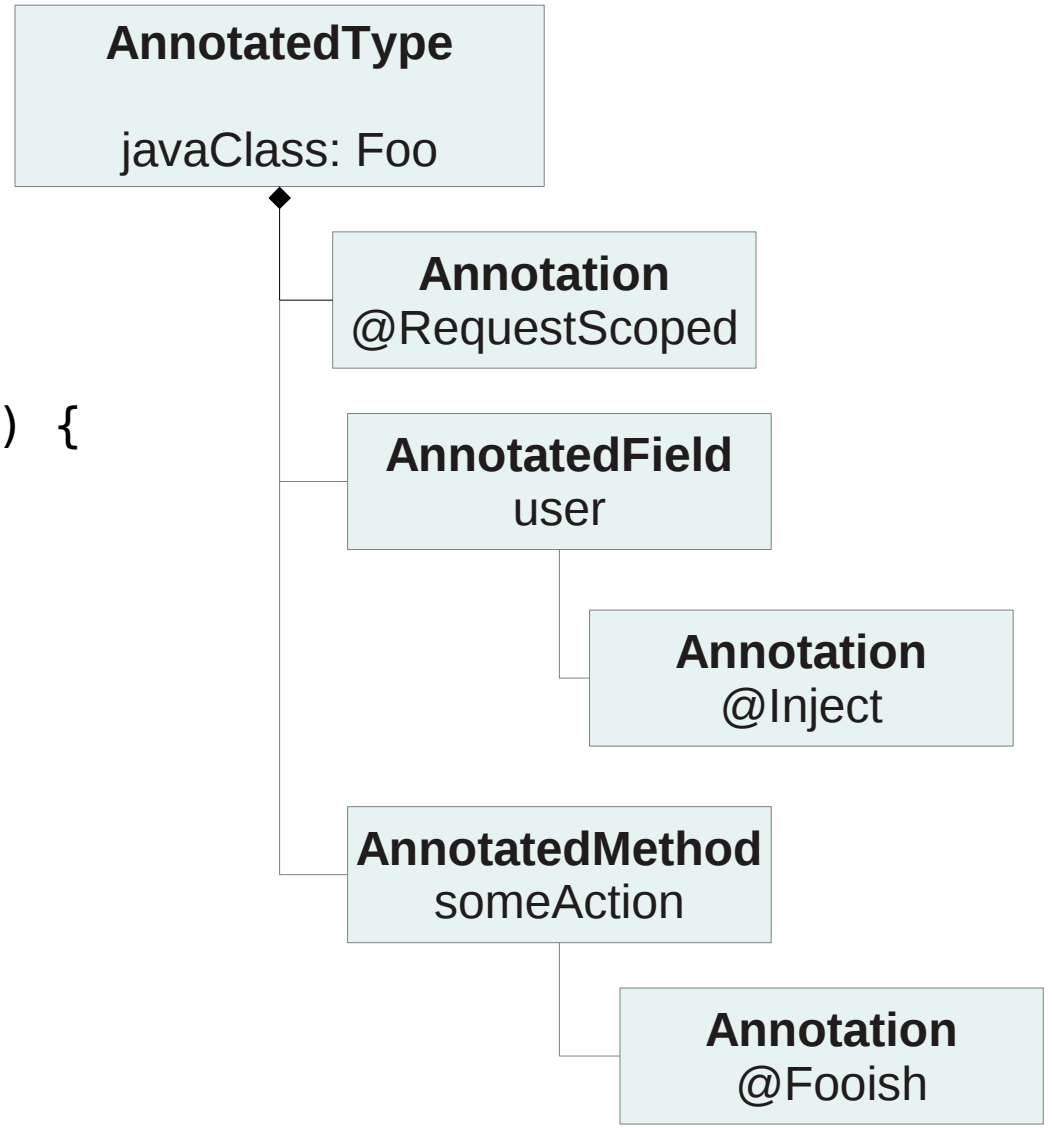
- javax.enterprise.inject.spi.Extension
- class MyExtension implements Extension { ... }
- META-INF/services/javax.enterprise.inject.spi.Extension
- Extension is not a bean, but can be injected into other beans once the initialization process is complete
  - @Inject MyExtension myExtension;
- Extensions can have observer methods observing container lifecycle events and may inject BeanManager

CDI Initialization

1. BeforeBeanDiscovery
2. Read all classes and interfaces -> AnnotatedType
3. Build beans
4. Validation

```
@RequestScoped
public class Foo {
  @Inject
  private User user;

  @Fooish
  public String someAction() {
  }
}
```



Container lifecycle events

- BeforeBeanDiscovery
- ProcessAnnotatedType
- ProcessInjectionTarget, ProcessProducer,
- ProcessBean and ProcessObserverMethod
- AfterBeanDiscovery
- AfterDeploymentValidation

Container lifecycle events

```
class MyExtension implements Extension {
   void beforeBeanDiscovery(@Observes BeforeBeanDiscovery bbd) {
      Logger.global.debug("beginning the scanning process");
   }
   <T> void processAnnotatedType(@Observes ProcessAnnotatedType<T> pat) {
      Logger.global.debug("scanning type: " +
pat.getAnnotatedType().getJavaClass().getName());
   }
   void afterBeanDiscovery(@Observes AfterBeanDiscovery abd) {
      Logger.global.debug("finished the scanning process");
   }
}
```

Veto Extension, motivation

- ```
  @Entity
  public class Car {
    @Id @GeneratedValue Long id; ...
  }
  ```
- ```
  public class CarManager {
    @Inject EntityManager em;

    public void setId(Long id) { ... }

    @Produces
    Car getCar() {
      if (id == null) { return new Car(); }
      return em.find(Car.class, id);
    }
    ...
  }
  ```
- ```
  @Inject Car car;
  ```
  ```

Veto Extension

```
public class VetoExtension implements Extension {

    public void vetoEnties(@Observes ProcessAnnotatedType pat) {
        if (pat.getAnnotatedType().getAnnotation(Entity.class)
                != null) {

            pat.veto();

            log.info("Vetoed class " +
pat.getAnnotatedType().getJavaClass());

        }
    }
}
```

Registering a new Bean

- ```
  void afterBeanDiscovery(@Observes AfterBeanDiscovery abd,
  BeanManager bm) {

   //use this to read annotations of the class

          AnnotatedType<FooService> at =
  bm.createAnnotatedType(FooService.class);

   //use this to instantiate the class and inject dependencies
          final InjectionTarget<FooService> it =
  bm.createInjectionTarget(at);

          abd.addBean( new Bean<FooService>() {
        ...
  }
  ```
  ```

The Bean Interface

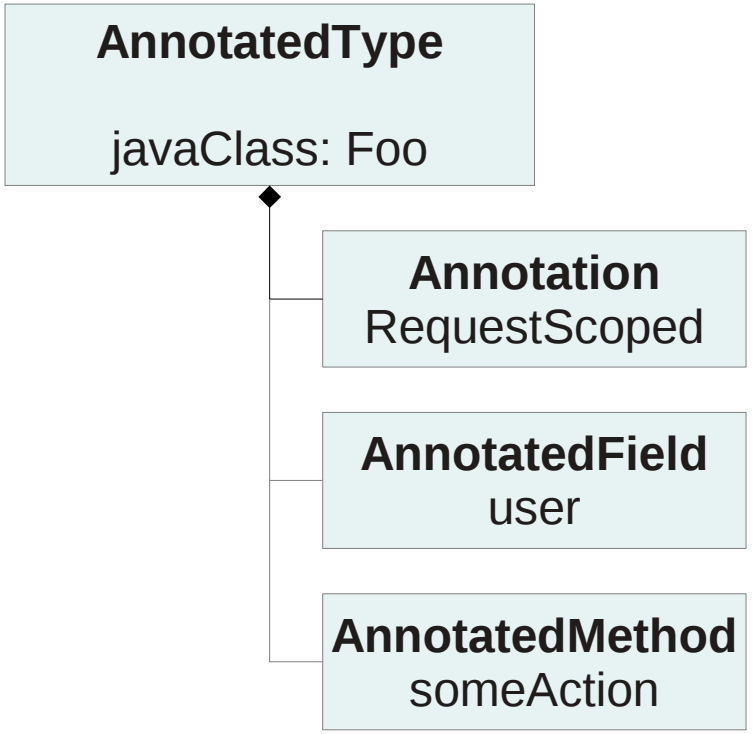
```
public interface Bean<T> extends Contextual<T> {
  public Set<Type> getTypes();
  public Set<Annotation> getQualifiers();
  public Class<? extends Annotation> getScope();
  public String getName();
  public Set<Class<? extends Annotation>>
  getStereotypes();
  public Class<?> getBeanClass();
  public boolean isAlternative();
  public boolean isNullable();
  public Set<InjectionPoint> getInjectionPoints();
}
```

Wrapping an AnnotatedType

- process the annotations of a bean class before the container builds its metamodel.

```
@RequestScoped
public class Foo {
  @Inject
  private User user;

  @Fooish
  public String someAction() {
  }
}
```



Wrapping an AnnotatedType

- ```
  <X> void processAnnotatedType(@Observes
  ProcessAnnotatedType<X> pat) {
    final AnnotatedType<X> at =
  pat.getAnnotatedType();

    AnnotatedType<X> wrapped = new AnnotatedType<X>()
     { ... };

    pat.setAnnotatedType(wrapped);
  }
  ```
  ```

AnnotatedType<T> extends Annotated

- Type getBaseType()
- Set<Type> getTypeClosure()
- <T extends Annotation> T getAnnotation(Class<T> annotationType)
- Set<Annotation> getAnnotations()
- boolean isAnnotationPresent(Class<? extends Annotation> annotationType)
- Class<T> getJavaClass()
- Set<AnnotatedConstructor<X>> getConstructors()
- Set<AnnotatedMethod<? super X>> getMethods()
- Set<AnnotatedField<? super X>> getFields()

Wrapping an InjectionTarget

- The InjectionTarget interface
  - producing and disposing an instance of a component
  - injecting dependencies
  - invoking lifecycle callbacks.
- Wrapping an InjectionTarget
  - intercept any of the these operations

InjectionTarget

- ```
  <X> void processInjectionTarget(@Observes
  ProcessInjectionTarget<X> pit) {
    final InjectionTarget<X> it =
  pit.getInjectionTarget();

    FooITWrapper wrapped = new FooITWrapper(it);

    pit.setInjectionTarget(wrapped);

  }
  ```
  ```

InjectionTarget

- X produce(CreationalContext<X> ctx)
- void dispose(X instance)
- Set<InjectionPoint> getInjectionPoints()
- void inject(T instance, CreationalContext<T> ctx);
- void postConstruct(T instance);
- void preDestroy(T instance);

Example

- CreatureExtension
  - A simple CDI Portable Extension to "inject" values from XML into the instances of a bean
  - @Observes ProcessInjectionTarget
  - XmlBackedWrappedInjectionTarget
  -
```
    @Override
    public void inject(X instance,
CreationalContext<X> ctx) {
        wrapped.inject(instance, ctx);
        ...
        // field.set(instance, valueFromXml);
}
```

ObserverMethod

- public void onFoo(@Observes @Booish Foo)
- ProcessObserverMethod
  - AnnotatedMethod getAnnotatedMethod
  - ObserverMethod getObserverMethod
- ObserverMethod<T> interface
  - Class<?> getBeanClass
  - Type getObservedType
  - Set<Annotation> getObservedQualifiers
  - Reception getReception()
  - TransactionPhase getTransactionPhase()
  - notify(T Event)

BeanManager

- obtain beans, interceptors, decorators, observers and contexts programmatically.
- @Inject BeanManager beanManager;

Obtaining the Bean

- `@Inject BeanManager bm;`
  - `public Set<Bean<?>> getBeans(Type beanType, Annotation... qualifiers);`
  - `public Set<Bean<?>> getBeans(String name);`
    - EL Name

Retrieving contextual references via BeanManager

- ```
  CreationalContext<Foo> ctx =
  beanManager.createCreationalContext(bean);

  Foo foo = (Foo)beanManager.getReference(bean,
  Foo.class, ctx);

  ...

  bean.destroy(foo, ctx);
  ```
- only destroy if dependent-scoped!
  ```

Custom Scopes

```
public interface Context {
   public Class<? extends Annotation> getScope();

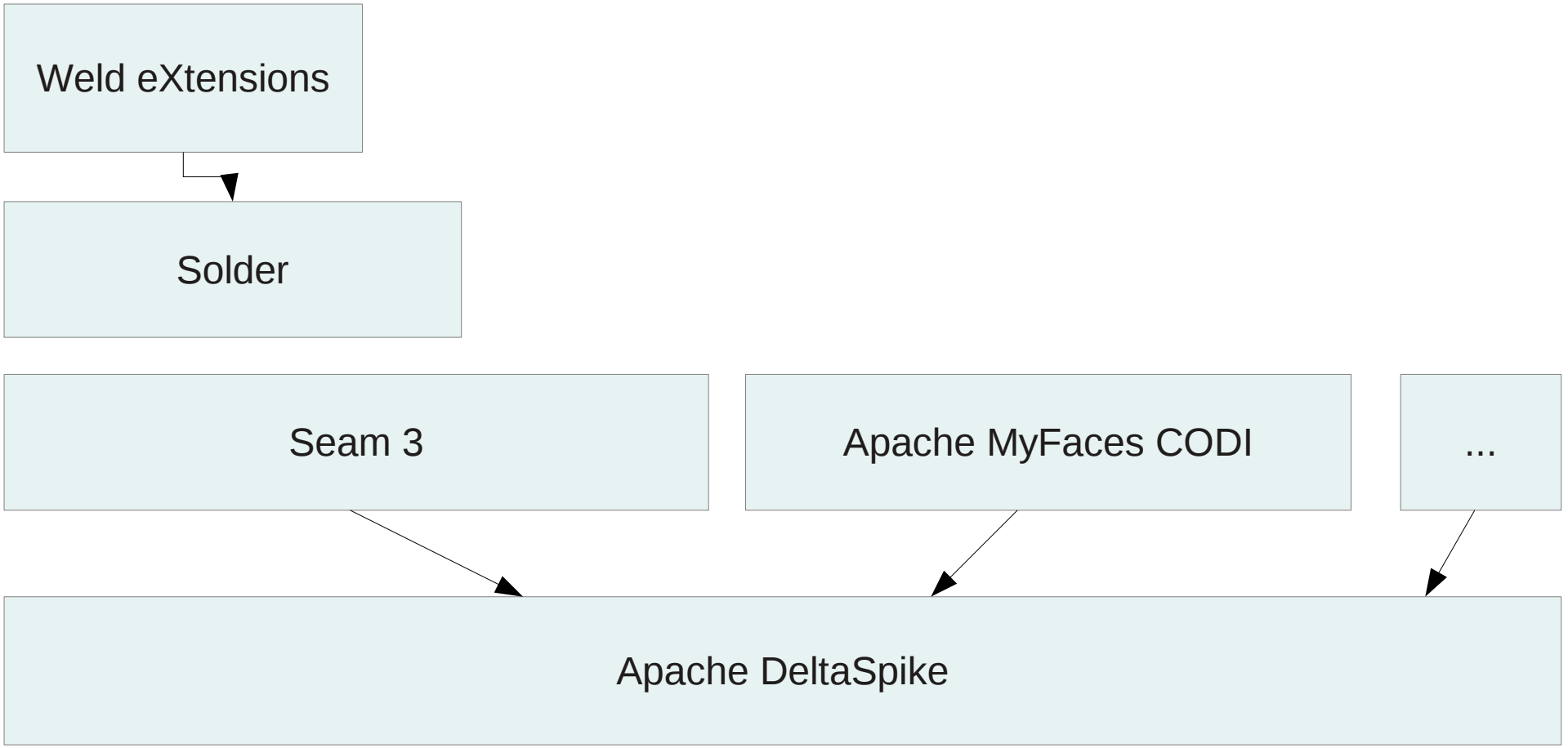
   public <T> T get(Contextual<T> contextual,
     CreationalContext<T> creationalContext);

   public <T> T get(Contextual<T> contextual);

   boolean isActive();
}
```


DELTA
SPIKE

Various CDI Extensions Projects



DeltaSpike Modules

- core
- jpa
- jsf
- security

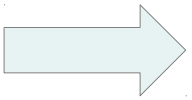
DeltaSpike core

- Utilities for CDI extension developers
  - BeanBuilder
  - AnnotatedTypeBuilder
  - AnnotationInstanceProvider and CDI annotation literals
  - AbstractContext
- Generally useful CDI extension for application developers

DeltaSpike Annotation Utilities

```
public class Foo {

  private User user;
}
```



```
@RequestScoped
public class Foo {
  @Inject
  private User user;
}
```

```
AnnotatedTypeBuilder atb = new AnnotatedTypeBuilder();
atb.readFromType(Foo.class);

atb.addToClass(new RequestScopedLiteral());

atb.addToField(Foo.class.getDeclaredField("user"), new
InjectLiteral());

AnnotatedType at = atb.create();
```

```
class InjectLiteral extends AnnotationLiteral<Inject>
implements Inject {}
```

DeltaSpike AbstractContext

- ContextualStorage
  - storage for contextual instances and their creational contexts
- AbstractContext
  - ContextualStorage getContextualStorage(
    boolean createIfNotExist)
  - boolean isActive()
  - Class<? extends Annotation> getScope()

DeltaSpike core extensions

- ProjectStage
- @Exclude
- @ConfigProperty
- i18n
  - type-safe messages
  - Dynamic Message Builder
- Exception Handling

@Exclude

- @Exclude
  public class NoBean {}
- @Exclude(ifProjectStage = ProjectStage.Development.class)
  public class MyBean {}

@ConfigProperty

- /META-INF/apache-deltaspike.properties
- @ApplicationScoped
  public class SettingsBean
  {
      @Inject
      @ConfigProperty(name = "property1")
      private String property1;
      //...
  }

i18n

- ```
  @MessageBundle
  public interface SimpleMessage
  {
     @MessageTemplate("Welcome to DeltaSpike")
     String welcomeToDeltaSpike();
  }
  ```

- ```
  package foo;
  @MessageBundle
  public interface SimpleMessage
  {
     @MessageTemplate("{welcome_to_deltaspike}")
     String welcomeToDeltaSpike();
  }
  ```

- foo/SimpleMessage.properties
  foo/SimpleMessage_de.properties

- welcome_to_deltaspike=Welcome to DeltaSpike

DeltaSpike Exception Handling 1/2

- Decoupling, similar to CDI Observers
-
```
public class InventoryActions {
  @PersistenceContext private EntityManager em;
  @Inject private Event<ExceptionToCatchEvent> catchEvent;

  public Integer queryForItem(Item item) {
    try {
      Query q = em.createQuery("SELECT i from Item i where i.id = :id");
      q.setParameter("id", item.getId());
      return q.getSingleResult();
    } catch (PersistenceException e) {
      catchEvent.fire(new ExceptionToCatchEvent(e));
    }
}
```

DeltaSpike Exception Handling 2/2

```
@ExceptionHandler
public class MyHandlers
{
   void printExceptions(@Handles
ExceptionEvent<Throwable> evt)
   {
      System.out.println("Something bad happened: " +
            evt.getException().getMessage());
      evt.handleAndContinue();
   }
}
```

DeltaSpike JPA Module 1/2

- @Transactional
  - alternative to transactional EJBs
- org.apache.deltaspike.jpa.impl.transaction.TransactionalInterceptor
-
```
class Resources {
   @PersistenceUnit
   private EntityManagerFactory entityManagerFactory;

   @Produces
   @RequestScoped
   protected EntityManager createEntityManager() {
      return this.entityManagerFactory.createEntityManager();
   }
   protected void closeEntityManager(@Disposes EntityManager entityManager) {
      if (entityManager.isOpen()) {
         entityManager.close();
      }
   }
}
```

DeltaSpike JPA Module 2/2

- ```
  @Transactional
  public class MemberRegistration {

      @Inject
      private EntityManager em;


      public void register(Member member)
  throws Exception {
          em.persist(member);
      }
  }
  ```
  ```

Questions & Answers

Prepare for the lab

- git clone git://github.com/qa/pv243.git
- cd pv243
- git checkout deltaspike-00
- JBDS -> Import... -> Maven -> Existing Maven Projects -> lesson03-cdi-pe
- http://wumpus-social.rhcloud.com