Introduction to Web Beans

Pete Muir
JBoss, a Division of Red Hat

http://in.relation.to/Bloggers/Pete
Road Map

- Background
- Concepts
- Status
Road Map

- Background
- Concepts
- Status
Goals

Web Beans provides a set of services for Java EE 6, by defining, on top of the EE programming model:

- an improved lifecycle for stateful components, bound to well-defined contexts
- a typesafe approach to dependency injection
- interaction via an event notification facility, and
- a better approach to binding interceptors to components, along with a new kind of interceptor, called a decorator, that is more appropriate for use in solving business problems.
Goals

- User doesn’t have to worry about:
  - lifecycle of this object
  - how many simultaneous clients it has
  - whether it is multithreaded
  - how to create an instance
  - how to destroy an instance
  - how to hold a reference to it (e.g. between requests)
  - how to alter the implementation at deployment time
  - how to share the object with other objects
Target Environment

- Java EE 6
- all JavaBeans
- all EJBs
- all Servlets
- The Java EE 6 Web Profile?
Migration

- All existing EJBs are Web Beans
- Any existing JSF managed bean is a Web Bean
- Most JavaBeans are Web Beans
- New Web Beans may interoperate with existing EJB3 session beans
  - via @EJB or JNDI
- New EJBs may interoperate with existing Web Beans
- Web Beans injection and interception supported for all EJBs
Loose coupling

- Events, interceptors and decorators enhance the loose-coupling that is inherent in this model:
  - event notifications decouple event producers from event consumers
  - interceptors decouple technical concerns from business logic
  - decorators allow business concerns to be compartmentalized
Seam 3?

- A Web Beans core
- A development environment
  - JBoss Tools
  - Seam-gen (command line tool)
- A set of modules for any Web Beans container
  - Seam Security
  - Reporting
  - Mail
  - BPM integration
  - Spring integration ....
Road Map

- Background
- Concepts
- Status
Road Map

- Background
- Concepts
- Status
What is a Web Bean?

- Essential Ingredients:
  - API types
  - Binding annotations
  - Scope
  - Deployment type
  - A name (optional)
  - Interceptor bindings
  - The implementation
Simple Example

Any Java Bean is a Web Bean

public class Hello {
    public String hello(String name) {
        return "hello" + name;
    }
}

@Stateless
public class Hello {
    public String hello(String name) {
        return "hello" + name;
    }
}

Or, it could be an EJB
public class Printer {

    @Current Hello hello;

    public void hello() {
        System.out.println( hello.hello("world") );
    }

}
public class Printer {
    private Hello hello;

    @Initializer
    public Printer(Hello hello) {
        this.hello = hello;
    }

    public void hello() {
        System.out.println(hello.hello("world"));
    }
}

Mark the constructor to be called by the container
@Initializer

Constructors are injected by default; @Current is the default binding type
Web Bean Names

By default Web Beans aren’t available through EL.

If no name is specified, then a default name is used. Both these Web Beans have the same name.
<h:commandButton value="Say Hello"
    action="#{hello.hello}"/>

Calling an action on a Web Bean through EL
A binding type is an annotation that lets a client choose between multiple implementations of an API at runtime.

- Binding types replace lookup via string-based names.
- `@Current` is the default binding type.
@BindingType
@Retention(RUNTIME)
@Target({TYPE, METHOD, FIELD, PARAMETER})
public @interface Casual {}

Creating a binding type is really easy!
We also specify the \@Casual binding type. If no binding type is specified on a Web Bean, \@Current is assumed.
Using a binding type

```java
public class Printer {
    @Casual Hello hello;

    public void hello() {
        System.out.println( hello.hello("JBoss") );
    }
}
```

Here, we inject the Hello Web Bean, and require an implementation which is bound to @Casual
Deployment Types

- A deployment type is an annotation that identifies a class as a Web Bean.
- Deployment types may be enabled or disabled, allowing whole sets of Web Beans to be easily enabled or disabled at deployment time.
- Deployment types have a precedence, allowing different implementations of an API to be chosen.
- Deployment types replace verbose XML configuration documents.
- Default deployment type: Production.
Create a deployment type

```java
@DeploymentType
@Retention(RUNTIME)
@Target({TYPE, METHOD})
public @interface Espanol { }
```
Using a deployment type

@Espanol

public class Hola extends Hello {

    public String hello(String name) {
        return "hola " + name;
    }

}
Enabling deployment types

A strongly ordered list of enabled deployment types. Notice how *everything* is an annotation and so typesafe!

Only Web Bean implementations which have enabled deployment types will be deployed to the container
Scopes and Contexts

- Extensible context model
  - A scope type is an annotation, can write your own context implementation and scope type annotation
- Dependent scope, @Dependent
- Built-in scopes:
  - Any servlet - @ApplicationScoped, @RequestScoped, @SessionScoped
  - JSF requests - @ConversationScoped
- Custom scopes
@SessionScoped

public class Login {

    private User user;

    public void login() {
        user = ...;
    }

    public User getUser() { return user; }
}

Scopes

Session scoped
public class Printer {

    @Current Hello hello;
    @Current Login login;

    public void hello() {
        System.out.println(
            hello.hello( login.getUser().getName() ) );
    }
}
@ConversationScoped

public class ChangePassword {

    @Current Conversation conversation;

    public User getUser(String userName) {
        conversation.begin();
        // Load and return the user
    }

    public User setPassword(String password) {
        // Set the users password
        conversation.end();
    }

}
Producer methods

- Producer methods allow control over the production of a Web Bean where:
  - the objects to be injected are not required to be instances of Web Beans
  - the concrete type of the objects to be injected may vary at runtime
  - the objects require some custom initialization that is not performed by the Web Bean constructor
@SessionScoped
public class Login {
    private User user;
    public void login() {
        user = ...;
    }
}

@Produces
User getUser() { return user; }
}
Producer methods

@RequestScoped
public class Login {
    private User user;
    public void login() {
        user = ...;
    }
}

@Produces @SessionScoped @LoggedIn
User getUser() {
    return user;
}
public class Printer {
  @Current Hello hello;
  @Current User user;

  public void hello() {
    System.out.println(hello.hello(user.getName()));
  }
}
Producer methods

@SessionScoped
public class Login {
    private User user;

    @Produces @WelcomeMessage
    String getWelcomeMessage(@Current Hello hello) {
        return hello.hello(user);
    }
}

You can inject parameters

A binding type
public class Printer {

    @WelcomeMessage String welcomeMessage;

    public void hello() {
        System.out.println(welcomeMessage);
    }
}

Producer Fields

- Simpler alternative to Producer methods

```java
@SessionScoped
public class Login {

    @Produces @LoggedIn @RequestScoped
    private User user;

    public void login() {
        user = ...;
    }
}
```

Similar to `outjection` in Seam
Disposal Method

Clean up after a producer method

```java
public class UserDatabaseEntityManager {

    @Produces @UserDatabase
    EntityManager create(EntityManagerFactory emf) {
        return emf.createEntityManager();
    }

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

- We have common architectural “patterns” in our application, with recurring component roles
- Capture the roles using stereotypes
Stereotypes

- A stereotype packages:
  - A *default* deployment type
  - A *default* scope
  - A set of interceptor bindings
  - *Restrictions* upon allowed scopes
  - *Restrictions* upon the Java type
  - May specify that Web Beans have names *by default*

- Built-in stereotypes: `@Model`
Creating a stereotype

@RequestScoped
@Named
@Production
@Casual
@Stereotype(
    supportedScopes={RequestScoped.class, SessionScoped.class})

@Retention(RUNTIME)
@Target(TYPE)
public @interface CasualAction {}
Using a stereotype

```java
@CasualAction
public class Hello {
    public String hello(String name) {
        return "hi " + name;
    }
}
```
Events

- Event producers raise events that are then delivered to event observers by the Web Bean manager.
- Not only are event producers decoupled from observers; observers are completely decoupled from producers.
- Observers can specify a combination of "selectors" to narrow the set of event notifications they will receive.
- Observers can be notified immediately, or can specify that delivery of the event should be delayed until the end of the current transaction.
public class Hello {

    @Observable @Casual Event<Greeting> casualHello;

    public void hello(String name) {
        casualHello.fire(new Greeting("hello " + name));
    }
}

"Fire" an event, the observer will be notified

Inject an instance of Event using @Observable. Additional binding types can be specified to narrow the event consumers called. API type specified as a parameter on Event
public class Printer {

    void onHello(@Observes @Casual Greeting greeting, @Current User user) {
        System.out.println(user + " " + greeting);
    }

}
Specialization

- Allows a bean with a higher precedence deployment to completely replace a bean with a lower precedence even producer methods, observer methods etc.

```java
@Mock
@Specializes
public class MockLogin extends Login {

    @Produces
    User getUser() { return new DummyUser(); }
}
```

A @Mock deployment type for testing
Realization

- Allows a bean with higher precedence to partially replace a bean with lower precedence.
- Producer methods, observer methods etc. will still be called on the super class.

```java
@Mock
@Realizes
public class MockLogin extends Login {

    @Override
    public void login() {
        user = ...;
    }
}
```

Inherits the Producer method.
public class Login {
    private User user;
    public void login() {
        user = ...;
    }

    User getUser() {
        return user;
    }
}
<web-beans>
  <myapp:Login>
    <RequestScoped />
  </myapp:Login>

  <myapp:getUser>
    <Produces>
      <myapp:User />
      <myapp:LoggedIn />
    </Produces>
  </myapp:getUser>

  </web-beans>
Road Map

- Background
- Concepts
- Status
Road Map

- Background
- Concepts
- Status
Number guess
JSR-299

- Public Review Draft published
- Web Beans “Book” (a less formal guide to Web Beans)

http://www.seamframework.org/WebBeans

Send feedback to jsr-299-comments@jcp.org
Web Beans RI

- Work on implementing the core spec (Public Draft)
  - Simple Web Beans
  - Enterprise Web Beans
  - Events
- Alpha Release soon!
Seam, Web Beans and JBoss Tools BOF

Pete Muir, Max Andersen, Dan Allen

Room 2, 7pm TODAY
Q&A

http://in.relation.to/Bloggers/Pete

http://www.seamframework.org/WebBeans