Update on JSR 299: Web Beans

Gavin King

Red Hat, Inc.
JBoss Seam

Bob Lee

Google, Inc.
Google Guice

TS-4089

JSR 299: Web Beans

Defining a unified Java™ component model

Learn how Web Beans can enable you to use a consistent and cohesive component model throughout your application.

Introductions

Your Humble Presenters

Gavin King

Fellow, Red Hat, Inc.

- JSR 299 spec. lead
- Created Hibernate
- Created Seam
- Heavy contributer to Enterprise JavaBeans™ (EJB™) 3.0
- Author of *Hibernate in Action*
- Former hip hop star

Bob Lee

Software Engineer, Google, Inc.

- Created Google Guice
- Java Community Process℠ (JCP℠) EC representative for Google
- Struts 2 and WebWork committer
- a.k.a. “Crazy Bob”

The Expert Group

JSR 299: Web Beans

Spec. Lead: Gavin King

- Companies
  - Adobe Systems, Inc.
  - Apache Software Foundation
  - Google, Inc.
  - Oracle
  - Pramati Technologies
  - Red Hat, Inc.
  - Sun Microsystems, Inc.
  - Tmax Soft, Inc.
- Individual members
  - Jacob Hookom
  - Oliver Ihns
  - Richard Kennard
  - Conny Lundgren
  - Chris Maki
  - Kito Mann
  - Martin Marinschek

Status

JSR 299: Web Beans

- The specification itself is just getting started
- But it's seeded by mature contributions
  - Seam
  - Guice
  - Shale
  - And others…

Problems

Web Beans addresses

- Need for a unified component model
- Managing state
  - Scoping components
  - Conversation management
- Finding components
- Configuration

Relationship to Java Platform, Enterprise Edition (Java EE Platform) 5

Web Beans is architected with the Java EE platform in mind

- Web Beans components may be used seamlessly as JavaServer™ Faces technology managed beans
  - In a sense, the model is an extension/replacement of the managed beans model that is transactional, secure, etc.
- EJB 3.x technology components may be Web Beans
  - Web Beans addresses the problem of integrating EJB technology components into the web tier
- The core of Web Beans is being architected to have no hard dependency upon JavaServer Faces or EJB 3 technology

What Is a Web Bean?

- The Web Beans component model is all about loose coupling:
  - Decouple implementations of server and client
    - By allowing easy overriding of server implementation
  - Decouple lifecycles of collaborating components
    - Using automatic state/lifecycle management
    - “Contextual components”
  - Decouple orthogonal concerns
    - Using interceptors
    - (True AOP is a boondoggle—an absurdly overcomplex solution to a narrow range of problems)

What Is a Web Bean?

- Ingredients
  - API
  - Implementation
  - Scope
  - Name
  - Binding annotations
  - Priority
- Kinds of components
  - Stateful and stateless session beans
    - Even WebService endpoints!
  - Entity beans
  - Any Java class file

Simple Example

- Trivial case

```
public
@Stateless
@Component
class Hello {

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

}
```

Simple Example

- Java platform client

```
public
@Stateless
@Component
class Printer {

    @In Hello hello;

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

}
```

Simple Example

- EL client

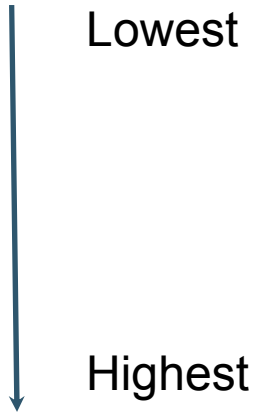
```
<h:commandButton value="Say Hello"
                 action="#{hello.hello}"/>
```

Overriding Components

- It is legal to have multiple implementations of the same component
  - i.e., same API, name, binding annotations
  - But they must specify different **priorities**
- At runtime, the Web Beans container chooses the implementation with the highest priority from those in the classpath
  - It is illegal to have two components with the same name and priority, or same binding annotations and priority

Overriding Components

- Predefined priorities
  - BUILT_IN
  - FRAMEWORK
  - APPLICATION
  - DEPLOYMENT
  - MOCK



Overriding Components

- Implement an interface

```
public
@Stateless
@Component(type=Greeting.class)
class Hello implements Greeting {

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

}
```

Simple Example

- Inject by the interface type

```
public
@Stateless
@Component
class Printer {

    @In Greeting greeting;

    ...

}
```

Simple Example

- Override the default implementation

```
public
@Stateless
@Component(type=Greeting.class, priority=DEPLOYMENT)
class Hola implements Greeting {

   public String greet(String name) {
      return "Hola " + name;
   }

}
```

Overriding Components

- We can easily mock out a component for testing

```
public
@Stateless
@Component(type=Greeting.class, priority=MOCK)
class MockHello extends Hello {

   public String greet(String name) {
      return "Hello World";
   }

}
```

Interceptors

- EJB 3.0 specification defined a nice interceptor model
  - For dealing with concerns orthogonal to the business logic
- Extend that model to all kinds of Web Beans
  - (Except entities?)
- Actually, the Web Beans container is implemented using EJB 3.0 specification interceptors

Scopes

- A **scope** is a policy for reusing component instances
- The scope of a component determines its lifecycle
- Web Beans features an extensible set of scopes

Scope Examples

- no scope (new instance each time)
- request
- session
- conversation
- application (singleton)
- custom extensions
  - business process
  - cluster
  - method

Injection

- Pico, Spring, Guice, EJB 3.0 technology: injection at component instantiation time
  - Via constructor, setter methods, direct field access
- Seam: injection at method invocation time
  - Via setters or direct field access
  - Allows components in a narrower scope (e.g., request) to be injected into a component in a wider scope (e.g., session, conversation)
  - **Extreme** loose coupling

Specifying Component Scope

- Use a scoping annotation

```
public
@SessionScoped
@Stateful
@Component
class ShoppingCart { ... }
```

Defining a New Scope

- Create a custom scoping annotation

```
public
@Documented
@Target(TYPE)
@Rentention(RUNTIME)
@ScopeType
class MethodScoped { ... }
```

Defining a New Scope

- Use the Contexts API to manage the lifecycle

```
public class MethodScopeInterceptor {

    @AroundInvoke
    public Object manageMethodScope(Invocation invocation)
    {
        Contexts.createContext(MethodScoped.class,
                               new Context() );
        try {
            return invocation.proceed();
        }
        finally {
            Contexts.destroyContext(MethodScoped.class);
        }
    }
}
```

Defining a New Scope

- Now apply it to the component

```
public
@Stateful
@MethodScoped
@Component
class Timer {
   public void start() { ... }
   public void stop() { ... }
}
```

Components With Multiple Roles

- What if we want to use the same component implementation class in different ways?
  - We might need two instances of the component at once
  - From different scopes, even

Components With Multiple Roles

- The implementation class defines the default role

```
public
@ConversationScoped
@Entity
@Component(name="user")
@Roles(LoggedIn.class)
class User {
    ...
}
```

Components With Multiple Roles

- The binding annotation defines an additional role

```
public
@Documented
@Rentention(RUNTIME)
@SessionScoped
@Component(name="loggedInUser")
@interface LoggedIn {}
```

Components With Multiple Roles

- Using both roles together is easy

```
public
@Stateless
@Component
class BanUser {

   @In User user;
   @In @LoggedIn User administrator;

   public void ban() {
      user.bannedBy(administrator);
   }

}
```

Components With Multiple Roles

- Or, in EL

```
#{user} was banned by #{loggedInUser}
```

Finding Components by Name

- Dynamically typed
- Used exclusively in Seam
- Each component has a unique string identifier
- Useful for:
  - Expression languages (JavaServer Faces technology, JavaServer Pages™ (JSP™))
  - XML configuration

Finding Components by Type

- Statically typed
- Used exclusively in Guice
- Each component has:
  - A mandatory invariant type
    - A parent of the implementation type
    - Noun
  - An optional set of annotation types
    - Adjectives
- Useful for Java platform clients
- Enables concise dependency injection

Binding Annotations

- Used to find components by type
- Describe the component—adjectives
- Reusable across types
  - `@Transactional DataSource`
  - `@Transactional WebService`
- Not just markers
  - They can have attribute values, too
  - `@Named("Gavin")`

Example: Find by Type

```
class Client {

   @In Service service;
   @In @Transactional DataSource dataSource;

   ...
}
```

- `@In`
  - Methods
  - Fields
  - Constructors?
- Component annotations go on:
  - The field declaration
  - Parameter declarations

Names vs. Types

- Use names in dynamically typed contexts
  - Expressions
  - XML
  - Scripting languages
- Use types for Java platform (and Groovy) clients
  - Full support for generic types
  - More up front checking
  - Better tool support
  - Simpler testing

Defining Component Externally

- An alternative way to define new roles

```
<component
  class='Material'
  role='Hard'
  scope='ApplicationScoped'
  name='concrete'
/>
```

Conversation Scope

- Primarily used with servlets
- Bigger than a request
- Smaller than a session
- Spans multiple requests
- Useful for implementing wizards
- Support multiple instances of the same wizard running in different browser windows
- Manage persistence context
  - Optimistic locking
  - Lazy fetching

Configuration

- Unify existing configuration mechanisms
  - JavaServer Faces technology has something nice (but doesn't go far enough)
  - EJB 3 technology has something awful
  - We need one way to do it
- Inject literal values directly
- Support conversion/validation of values
- Wire components together view EL

BPM

- A business process is a long-running collaboration between multiple users
- A business process engine can manage (persist and share) state associated with a process instance
- The engine also manages the process workflow (tasks, and dependencies between tasks)
- A “task” is just a special kind of conversation
- Java EE platform has no standard business process management engine
  - So Web Beans will be extensible, to allow addition of a business process scope

Packaging and Deployment

- Currently, Java EE platform requires a complex deployment archive structure
  - EJB technology, Java Archive (JAR), WAR nested inside the EAR
- Web Beans breaks down the traditional barrier between the web and transactional tiers
- So we need to simplify the packaging model
- Java EE 6 platform should allow deployment of EJB technology components (and any other Web Beans) directly into the classes/directory of a WAR

Summary

- Early stages of the specification
- Strong existing contributions
- **The Future Is Bright**

For More Information

- Check out
  - Seam
  - Guice

Q&A

Gavin King and Bob Lee

Update on JSR 299: Web Beans

Gavin King

Red Hat, Inc.
JBoss Seam

Bob Lee

Google, Inc.
Google Guice

TS-4089