Seam Persistence

Seam Persistence Reference

Seam provides extensive support for the two most popular persistence architectures for Java: Hibernate3, and the Java Persistence API introduced with EJB 3.0. Seam's unique state-management architecture allows the most sophisticated ORM integration of any web application framework.

Note

Previously the Seam Persistence module provided transactional-related features also, however as transactions are not an exclusive feature of the persistence domain, these features have been moved into a separate module called Seam Transactions.

1.1. Introduction

Seam grew out of the frustration of the Hibernate team with the statelessness typical of the previous generation of Java application architectures. The state management architecture of Seam was originally designed to solve problems relating to persistence — in particular problems associated with*optimistic transaction processing*. Scalable online applications always use optimistic transactions. An atomic (database/JTA) level transaction should not span a user interaction unless the application is designed to support only a very small number of concurrent clients. But almost all interesting work involves first displaying data to a user, and then, slightly later, updating the same data. So Hibernate was designed to support the idea of a persistence context which spanned an optimistic transaction.

Unfortunately, the so-called "stateless" architectures that preceded Seam and EJB 3.0 had no construct for representing an optimistic transaction. So, instead, these architectures provided persistence contexts scoped to the atomic transaction. Of course, this resulted in many problems for users, and is the cause of the number one user complaint about Hibernate: the dreaded `LazyInitializationException`. What we need is a construct for representing an optimistic transaction in the application tier.

EJB 3.0 recognizes this problem, and introduces the idea of a stateful component (a stateful session bean) with an *extended persistence context* scoped to the lifetime of the component. This is a partial solution to the problem (and is a useful construct in and of itself) however there are two problems:

- The lifecycle of the stateful session bean must be managed manually via code in the web tier (it turns out that this is a subtle problem and much more difficult in practice than it sounds).
- Propagation of the persistence context between stateful components in the same optimistic transaction is possible, but tricky.

Seam solves the first problem by providing conversations, and stateful session bean components scoped to the conversation. (Most conversations actually represent optimistic transactions in the data layer.) This is sufficient for many simple applications (such as the Seam booking demo) where persistence context propagation is not needed. For more complex applications, with many loosely-interacting components in each conversation, propagation of the persistence context across components becomes an important issue. So Seam extends the persistence context management model of EJB 3.0, to provide conversation-scoped extended persistence contexts.

1.2. Getting Started

To get started with Seam Persistence you need to add `seam-persistence.jar` and `solder-impl.jar` to your deployment. The relevant Maven configuration is as follows:

```
<dependency>
  <groupId>org.jboss.seam.persistence</groupId>
  <artifactId>seam-persistence-api</artifactId>
  <version>${seam.persistence.version}</version>
</dependency>

<dependency>
  <groupId>org.jboss.seam.persistence</groupId>
  <artifactId>seam-persistence</artifactId>
  <version>${seam.persistence.version}</version>
</dependency>

<dependency>
  <groupId>org.jboss.solder</groupId>
  <artifactId>solder-impl</artifactId>
  <version>${solder.version}</version>
</dependency>
```

You will also need to have a JPA provider on the classpath. If you are using java EE this is taken care of for you. If not, we recommend hibernate.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>3.5.1-Final</version>
</dependency>
```

1.3. Seam-managed persistence contexts

If you're using Seam outside of a Java EE environment, you can't rely upon the container to manage the persistence context lifecycle for you. Even if you are in an EE environment, you might have a complex application with many loosely coupled components that collaborate together in the scope of a single conversation, and in this case you might find that propagation of the persistence context between component is tricky and error-prone.

In either case, you'll need to use a *managed persistence context* (for JPA) or a *managed session* (for Hibernate) in your components. A Seam-managed persistence context is just a built-in Seam component that manages an instance of `EntityManager` or `Session` in the conversation (or any other) context. You can inject it with`@Inject`.

1.3.1. Using a Seam-managed persistence context with JPA

```
@ExtensionManaged
@Produces
@PersistenceUnit
@ConversationScoped
EntityManagerFactory producerField;
```

This is just an ordinary resource producer field as defined by the CDI specification, however the presence of the `@ExtensionManaged` annotation tells seam to create a seam managed persistence context from this`EntityManagerFactory`. This managed persistence context can be injected normally, and has the same scope and qualifiers that are specified on the resource producer field.

This will work even in a SE environment where `@PersistenceUnit` injection is not normally supported. This is because the seam persistence extensions will bootstrap the `EntityManagerFactory` for you.

Now we can have our `EntityManager` injected using:

```
@Inject EntityManager entityManager;
```

Note

The more eagle eyed among you may have noticed that the resource producer field appears to be conversation scoped, which the CDI specification does not require containers to support. This is actually not the case, as the `@ConversationScoped`

annotation is removed by the Seam Persistence portable extension. It only specifies the scope of the created SMPC, not the `EntityManagerFactory`.

Warning

If you are using EJB3 and mark your class or method `@TransactionAttribute(REQUIRES_NEW)` then the transaction and persistence context shouldn't be propagated to method calls on this object. However as the Seam-managed persistence context is propagated to any component within the conversation, it will be propagated to methods marked `REQUIRES_NEW`. Therefore, if you mark a method `REQUIRES_NEW` then you should access the entity manager using `@PersistenceContext`.

1.3.2. Seam-managed persistence contexts and atomic conversations

Persistence contexts scoped to the conversation allows you to program optimistic transactions that span multiple requests to the server without the need to use the `merge()` operation , without the need to re-load data at the beginning of each request, and without the need to wrestle with the `LazyInitializationException` or `NonUniqueObjectException`.

As with any optimistic transaction management, transaction isolation and consistency can be achieved via use of optimistic locking. Fortunately, both Hibernate and EJB 3.1 make it very easy to use optimistic locking, by providing the `@Version` annotation.

By default, the persistence context is flushed (synchronized with the database) at the end of each transaction. This is sometimes the desired behavior. But very often, we would prefer that all changes are held in memory and only written to the database when the conversation ends successfully. This allows for truly atomic conversations. Unfortunately there is currently no simple, usable and portable way to implement atomic conversations using EJB 3.1 persistence. However, Hibernate provides this feature as a vendor extension to the `FlushModeType`s defined by the specification, and it is our expectation that other vendors will soon provide a similar extension.

1.3.3. Using EL in EJB-QL/HQL

Seam proxies the `EntityManager` or `Session` object whenever you use a Seam-managed persistence context. This lets you use EL expressions in your query strings, safely and efficiently. For example, this:

```
User user = em.createQuery("from User where username=#{user.username}")
  .getSingleResult();
```

is equivalent to:

```
User user = em.createQuery("from User where username=:username")
   .setParameter("username", user.getUsername())
   .getSingleResult();
```

Of course, you should never, ever write it like this:

```
User user = em.createQuery("from User where username=" + user.getUsername()) //BAD!
   .getSingleResult();
```

(It is inefficient and vulnerable to SQL injection attacks.)

Warning

This only works with Seam managed persistence contexts, not persistence contexts that are injected with `@PersistenceContext`.

1.3.4. Setting up the EntityManager

Sometimes you may want to perform some additional setup on the `EntityManager` after it has been created. For example, if you are using Hibernate you may want to set a filter. Seam persistence fires a `SeamManagedPersistenceContextCreated` event when a Seam managed persistence context is created. You can observe this event and perform any setup you require in an observer method. For example:

```
public void setupEntityManager(@Observes SeamManagedPersistenceContextCreated
            event) {
            Session session = (Session) event.getEntityManager().getDelegate();
            session.enableFilter("myfilter");
            }
```