CS544 EA

Applications

Hibernate Web Applications

Hibernate Web Applications

- Hibernate is generally only used in combination with Spring or a J2EE Server
  - To provide a deeper understanding of how it’s integrated into an application
  - We’re first going to **manually provide** some of the things Hibernate needs to run

Single Entity Manager Factory

- The Entity Manager Factory should start **once**
  - **Only one** for the entire application
  - Starts when the app starts
  - Closes when the app closes
- Good way to do this:
  - Make a **singleton** for it

Entity Manager & DAOs

- Repositories (DAOs) need to be able to get the ‘**current entityManager**’
  - If each DAO method makes it’s own EntityManager
  - We need multiple per web request
  - Each EntityManager has:
    - Its own DB connection, transaction, entity cache
    - All of which should be used for multiple operations!

EntityManager per Operation Anti-Pattern

- Using a EntityManager per operation is so bad it's considered an Anti-Pattern
- Also known as:

  "SessionPerOperation" Anti-Pattern

Never write a DAO like this!

```java
public class BadCustomerDao {
	private EntityManagerFactory emf;
	public CustomerDao() {
		EntityManagerFactory emf = EMF.get();
	}

	public Customer load(Long id) {
		EntityManager em = emf.createEntityManager();
		Customer c = em.find(Customer.class, id);
		em.close();
		return c;
	}
	public void save(Customer c) {
		EntityManager em = emf.createEntityManager();
		em.persist(c);
		em.close();
	}
	public void update(Customer c) {
		EntityManager em = emf.createEntityManager();
		em.merge(c);
		em.close();
	}
}
```

Entity Manager per Request

- We want one Entity Manager per (web) Request
  - Create it in the controller and pass it around as param?
  - Messy solution ✗

- **Store it in the current thread**
  - Available to every method running in the thread ✓
  - Known as "ThreadLocal"

EntityManager Helper

```java
public class EntityManagerHelper {
    private static final EntityManagerFactory emf;
    private static final ThreadLocal<EntityManager> threadLocal;

    static {
        emf = Persistence.createEntityManagerFactory("cs544");
        threadLocal = new ThreadLocal<EntityManager>();
    }

    public static EntityManager getCurrent() {
        EntityManager em = threadLocal.get();
        if (em == null || !em.isOpen()) {
            em = emf.createEntityManager();
            threadLocal.set(em);
        }
        return em;
    }

    public static void closeEntityManagerFactory() {
        emf.close();
    }
}
```

EntityManagerHelper provides:

- **Singleton** EntityManagerFactory
- **ThreadLocal**<EntityManager>
- getCurrent() method that can be called from any method

Based on: https://stackoverflow.com/questions/15071238/entitymanager-threadlocal-pattern-with-jpa-in-jse

EntityManager per Request DAO

- DAO's become thin wrappers:
  - Gets current EntityManager
  - Calls method

```java
public class CustomerDao {

	public Customer load(Long id) {
		EntityManager em = EntityManagerHelper.getCurrent();
		return em.find(Customer.class, id);
	}

	public void save(Customer c) {
		EntityManager em = EntityManagerHelper.getCurrent();
		em.persist(c);
	}

	public void update(Customer c) {
		EntityManager em = EntityManagerHelper.getCurrent();
		em.merge(c);
	}
}
```

Transaction

- Each **service method** should be **one transaction**

- Many Thread Local implementations close the EntityManager when the transaction commits
  - Means that all managed objects become detached
  - And automatic loading of related objects no longer works

Service Method

- Before an object is returned from a Service method:
  - **Load any related objects** needed by the recipient
    - Either have the DAO load all object into EM cache with query
    - Or have the Service follow references to 'force lazy loading'

```
public class CustomerService {
      ...
      public Customer getCustomer(Long id) {
            EntityManager em = EntityManagerHelper.getCurrent();
            em.getTransaction().begin();
            Customer c = customerDao.load(id);
            // follow references to ensure related objects are loaded
            c.getAddress().getCity();
            c.getCreditCard().getAddress().getCity();
            // Then commit (may close entity manager)
            em.getTransaction().commit();
            // and return the 'object structure'
            return c;
      }
}
```

The Service method Starts and stops the Transaction

During the transaction it makes sure related objects are loaded