JDG + EAP Lab 4 Guide

This explains the steps for lab 4, either follow them step-by-step or if you feel adventurous try to accomplish goals without the help of the step-by-step guide.

Background

The sales account manager for Acme Inc from the RDBMS vendor (ELCARO) had a meeting with the CIO of Acme this week. Because Acme Inc used JDG to improve performance instead of purchasing more DB licenses the sales account manager of ELCARO decided to try to make up for the lost sales, by raising the price on the licenses that Acme are currently using. The discussion has been harsh and the CIO is really angry at the sales account manager from ELCARO. At a similar meeting with the Red Hat Sales team with the CIO the Red Hat Solutions Architect (who's advice the CIO really trusts) suggested that Acme removes the database from the application an instead starts using JDG as the primary data store.

Use-case

Rewrite the application to only use JDG library mode, configure a file store and configure cluster.

These are the main tasks of lab 4

1. Remove JPA code from Task and TaskService
2. Configure a file store (using SingleFileStore)
3. Configure the cache for clustering

Setup the lab environment

To assist with setting up the lab environment we have provided a shell script that does this.

1. Run the shell script by standing in the jdg lab root directory (~/jdg-labs) execute a command like this

   ```
   $ sh init-lab.sh --lab=4
   ```

   Stop and running servers from previous labs. Then start the servers in separate consoles using the following commands

   Node 1:

   ```
   $ ./target/node1/jboss-eap-6.3/bin/standalone.sh
   ```

   Node 2:

   ```
   $ ./target/node2/jboss-eap-6.3/bin/standalone.sh -Djboss.socket.binding.port-offset=100
   ```

Step-by-Step

1. Open `src/main/java/org/jboss/infinispan/demo/TaskService.java` and remove all JPA references, specifically

   ```
   import javax.persistence.EntityManager;
   import javax.persistence.PersistenceContext;
   import javax.persistence.criteria.CriteriaBuilder;
   import javax.persistence.criteria.CriteriaQuery;
   import javax.persistence.criteria.Root;
   .....
   Task newTask = em.merge(task);
   em.detach(newTask);
   .....
   em.remove(em.getReference(task.getClass(),task.getId()));
   .....
   log.info("### Querying the database for tasks!!!!");
   final CriteriaBuilder criteriaBuilder = em.getCriteriaBuilder();
   final CriteriaQuery criteriaQuery = criteriaBuilder.createQuery(Task.class);

   Root root = criteriaQuery.from(Task.class);
   criteriaQuery.select(root);
   Collection resultList = em.createQuery(criteriaQuery).getResultList();

   for (Task task : resultList) {
           this.insert(task);
   ```

2. Change the `update(Task task)` method to

   ```
   public void update(Task task) {
                   cache.replace(task.getId(),task);
           }
   ```

3. The new `TaskService.java` should look something like this:

   ```
   package org.jboss.infinispan.demo;

   import java.util.ArrayList;
   import java.util.Collection;
   import java.util.Date;
   import java.util.List;
   import java.util.logging.Logger;

   import org.apache.lucene.search.Query;
   import org.hibernate.search.query.dsl.QueryBuilder;
   import org.infinispan.Cache;
   import org.infinispan.query.CacheQuery;
   import org.infinispan.query.Search;
   import org.infinispan.query.SearchManager;
   import org.jboss.infinispan.demo.model.Task;
   ```

```
@Stateless
public class TaskService {

    @Inject
    Cache<Long,Task> cache;

    Logger log = Logger.getLogger(this.getClass().getName());

    /**
     * This methods should return all cache entries, currently contains mockup code.
     * @return
     */
    public Collection<Task> findAll() {
        return cache.values();
    }

    /**
     * This method filters task based on the input
     * @param input - string to filter on
     * @return
     *
     */
    public Collection<Task> filter(String input) {
        SearchManager sm = Search.getSearchManager(cache);
        QueryBuilder qb = sm.buildQueryBuilderForClass(Task.class).get();
        Query q = qb.keyword().onField("title").matching(input).createQuery();
        CacheQuery cq = sm.getQuery(q, Task.class);
        List<Task> tasks = new ArrayList<Task>();
        for (Object object : cq) {
            tasks.add((Task) object);
        }
        return tasks;
    }

    /**
     * This method persists a new Task instance
     * @param task
     *
     */
    public void insert(Task task) {
        if(task.getCreatedOn()==null)
            task.setCreatedOn(new Date());
        cache.put(task.getId(),task);
    }

    /**
     * This method persists an existing Task instance
     * @param task
     *
     */
    public void update(Task task) {
        cache.replace(task.getId(),task);
    }

    /**
     * This method deletes an Task from the persistence store
     * @param task
     *
     */
    public void delete(Task task) {
        //Note object may be detached so we need to tell it to remove based on reference
        cache.remove(task.getId());
    }

    /**
     * This method is called after construction of this SLSB.
     *
     */
    @PostConstruct
    public void startup() {

    }

}
```

4. Implement a new way to generate unique id when inserting new tasks. Replace:

```
public void insert(Task task) {
    if(task.getCreatedOn()==null)
        task.setCreatedOn(new Date());
    cache.put(task.getId(),task);
}
```

with:

```
public void insert(Task task) {
    if(task.getCreatedOn()==null)
                        task.setCreatedOn(new Date());
                task.setId(new Long(System.nanoTime()));
                cache.put(task.getId(),task);
}
```

Note: Since our domain model relied on JPA to generate unique Task IDs, we will `System.nanoTime()` as a Task ID for simplicity's sake. Please note that in a clustered environment there are no guarantees that the time generated from `System.nanoTime()` will be unique. Therefore, we do not recommend using this method for production code, discuss with your collegues how we could solve this in a better way.

5. Remove the following JPA references in `src/main/java/org/jboss/infinispan/demo/model/Task.java`

```
import javax.persistence.Column;
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Version;
.....
@Entity
.....
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name = "id", updatable = false, nullable = false)
.....
@Version
@Column(name = "version")
.....
@Column(length = 100)
.....
@Column
.....
@Temporal(TemporalType.DATE)
.....
@Temporal(TemporalType.DATE)
```

6. The new Task class should look something like this:

```
package org.jboss.infinispan.demo.model;

import java.io.Serializable;
import java.util.Date;

import org.hibernate.search.annotations.Field;
import org.hibernate.search.annotations.Indexed;
import org.hibernate.search.annotations.Store;

/**
 * This class is the JPA entity of a Task
 * @author tqvarnst
 *
 */
@Indexed
public class Task implements Serializable {

    private static final long serialVersionUID = 2315323429163437300L;

    private Long id;

    private int version;

    @Field(store = Store.YES)
    private String title;

    private boolean done;

    private Date createdOn;

    private Date completedOn;

    public Long getId() {
        return this.id;
    }

    public void setId(final Long id) {
        this.id = id;
    }

    public int getVersion() {
        return this.version;
    }

    public void setVersion(final int version) {
        this.version = version;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (!(obj instanceof Task)) {
            return false;
        }
        Task other = (Task) obj;
        if (id != null) {
            if (!id.equals(other.id)) {
                return false;
            }
        }
        return true;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
    }

    public String getTitle() {
        return title;
    }
```

```
    public void setTitle(String title) {
        this.title = title;
    }

    public boolean isDone() {
        return done;
    }

    public void setDone(boolean done) {
        this.done = done;
    }

    public Date getCreatedOn() {
        return createdOn;
    }

    public void setCreatedOn(Date createdOn) {
        this.createdOn = createdOn;
    }

    public Date getCompletedOn() {
        return completedOn;
    }

    public void setCompletedOn(Date completedOn) {
        this.completedOn = completedOn;
    }

    @Override
    public String toString() {
        String result = getClass().getSimpleName() + " ";
        if (title != null && !title.trim().isEmpty())
            result += "title: " + title;
        result += ", done: " + done;
        return result;
    }


}
```

7. Since we are not using the database anymore we don't have the pre-populated data from `import.sql` which our test relies on. Update the `TaskServiceTest.java` to include the following methods

```
@Test
@InSequence(4)
        public void testUpdateTask() {
                int orgsize = taskservice.findAll().size();
                Task task = new Task();
                task.setTitle("This is the second test task");
                task.setCreatedOn(new Date());
                taskservice.insert(task);

                log.info("###### Inserted task with id " + task.getId());
                task.setDone(true);
                task.setCompletedOn(new Date());
                taskservice.update(task);
                Assert.assertEquals(orgsize+1, taskservice.findAll().size());

                for (Task listTask : taskservice.findAll()) {
                        if("This is the second test task".equals(listTask.getTitle())) {
                                log.info("### task =" + listTask.getTitle());
                                Assert.assertNotNull(listTask.getCompletedOn());
                                Assert.assertEquals(true,listTask.isDone());
                                taskservice.delete(listTask);
                        }
                }
                Assert.assertEquals(orgsize, taskservice.findAll().size());
        }


@Test
@InSequence(5)
    public void testFilterTask() {

        Task t1 = generateTestTasks("Sell EAP to customer A", false);
        Task t2 = generateTestTasks("Get FeedBack from EAP customers", false);
        Task t3 = generateTestTasks("Get FeedBack from JDG custoers", true);
        Task t4 = generateTestTasks("Sell JDG to customer B", false);
        Task t5 = generateTestTasks("Pickup kids from daycare", false);

        Collection<Task> tasks = taskservice.filter("EAP");
        Assert.assertEquals(2, tasks.size());
        tasks = taskservice.filter("SELL");
        Assert.assertEquals(2, tasks.size());
        tasks = taskservice.filter("FeedBack");
        Assert.assertEquals(2, tasks.size());

        taskservice.delete(t1);
        taskservice.delete(t2);
        taskservice.delete(t3);
        taskservice.delete(t4);
        taskservice.delete(t5);
        }
```

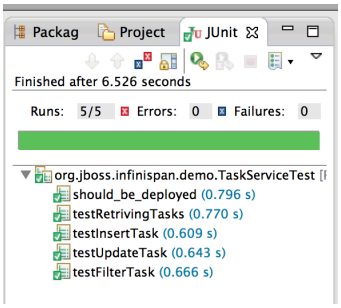
8. Change method `createDeployment` such that it looks like

```
@Deployment
public static WebArchive createDeployment() {

                return ShrinkWrap
```

```
                .create(WebArchive.class, "todo-test.war")
                .addClass(Config.class)
                .addClass(Task.class)
                .addClass(TaskService.class)
                .addAsResource("jgroups-cluster-config.xml")
                .addAsWebInfResource(new File("src/main/webapp/WEB-INF/jboss-deployment-structure.xml"))
                .addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml");
    }
```

9. Run the JUnit test to verify your changes so far.



10. Open `src/main/java/org/jboss/infinispan/demo/Config.java` and change the code in the `defaultEmbeddedCacheConfiguration()` method to:

```
public EmbeddedCacheManager defaultEmbeddedCacheConfiguration() {
        if (manager == null) {
                GlobalConfiguration glob = new GlobalConfigurationBuilder()
                                .clusteredDefault() // Builds a default clustered configuration
                                .transport().addProperty("configurationFile", "jgroups-cluster-config.xml")
                                .globalJmxStatistics().allowDuplicateDomains(true).enable() // This
                                        // method enables the jmx statistics of the global
                                        // configuration and allows for duplicate JMX domains
                                .build();


                        Properties properties = new Properties();
                        properties.put("default.directory_provider", "ram");

                        Configuration loc = new ConfigurationBuilder().jmxStatistics()
                                        .enable() // Enable JMX statistics
                                        .clustering().cacheMode(CacheMode.REPL_SYNC)
                                        .persistence()
                                                .addSingleFileStore()
                                                        .location(System.getProperty("jboss.home.dir") + "/cache-store")
                                                        .fetchPersistentState(true)
                                                        .ignoreModifications(true)
                                                        .shared(false)
                                                        .preload(false)
                                                        .async()
                                                                .enable()
                                                                .threadPoolSize(500)
                                                                .flushLockTimeout(1)
                                                                .modificationQueueSize(1024)
                                                                .shutdownTimeout(25000)
                                        .eviction().strategy(EvictionStrategy.NONE) // Do not evic objects
                                        .transaction().transactionMode(TransactionMode.TRANSACTIONAL).lockingMode(LockingMode.OPTIMISTIC)
                                        .indexing().enable().indexLocalOnly(false)
                                        .withProperties(properties).build();
                        manager = new DefaultCacheManager(glob, loc, true);
                }
                return manager;
        }
```

11. Notice that the transport for the cluster is specified using `jgroups-cluster-config.xml`

```
GlobalConfiguration glob = new GlobalConfigurationBuilder()
    .clusteredDefault()
    .transport().addProperty("configurationFile", "jgroups-cluster-config.xml")
    .globalJmxStatistics().allowDuplicateDomains(true).enable()
    .build();
```

12. Add this `import` statement:

```
import org.infinispan.configuration.cache.CacheMode;
```

13. Run the JUnit test to verify that your changes works.

14. Deploy the application and test that everything works as before.

```
$ cd projects/lab4
$ mvn clean package
$ mvn jboss-as:deploy
$ mvn jboss-as:deploy -Djboss-as.port=10099
```

15. Open two browser windows, one for http://localhost:8080/mytodo and another for http://localhost:8180/mytodo. Verify that you can add content in one window and that they appear when you reload the other window.

16. Congratulations, you are done with lab 4