HibernateSessionFactory

```
package scm.hivemind.hibernate;

import java.io.InputStream;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;

import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Session;
import net.sf.hibernate.SessionFactory;
import net.sf.hibernate.Transaction;
import net.sf.hibernate.cfg.Configuration;
import net.sf.hibernate.tool.hbm2ddl.SchemaUpdate;

import org.apache.commons.logging.Log;
import org.apache.hivemind.ApplicationRuntimeException;
import org.apache.hivemind.Discardable;
import org.apache.hivemind.PoolManageable;
import org.apache.hivemind.Resource;
import org.apache.hivemind.ServiceImplementationFactory;
import org.apache.hivemind.ServiceImplementationFactoryParameters;
import org.apache.hivemind.events.RegistryShutdownListener;
import org.apache.hivemind.service.ThreadEventNotifier;

import scm.hivemind.statefulservice.StatefulServiceLifecycleListener;

/**
 *  Die Klasse HibernateSessionFactory ist eine Hivemind-ServiceImplementationFactory
 * die Hibernate-Session-Proxies zur Verwendung als Hivemind-Services produziert.
 *
 * @author schultma
 */
public class HibernateSessionFactory
    implements ServiceImplementationFactory, RegistryShutdownListener
  {
    private SessionFactory sessionFactory;
    private Transaction transaction;
    private ThreadEventNotifier threadEventNotifier;
    private boolean updateSchema = false;
    private boolean transactionManager = false;
    protected Log log;
    private String configFile;
    private String jndiName;
    private Properties hibProps = new Properties();

    public HibernateSessionFactory( List config ) {
        for ( Iterator it = config.iterator(); it.hasNext(); ) {
            HibernateProperty cfgProp = (HibernateProperty) it.next();
            if ( cfgProp instanceof HibernateConfigFile )
                configFile = cfgProp.getName();
            else
                hibProps.setProperty( "hibernate."+ cfgProp.getName(),
                                                    cfgProp.getValue() );
        }

    }

    public void init()
    {
      try {
          log.debug( "Initializing Hibernate SessionFactory..." );
          Configuration config =  new Configuration() {
              /* Make sure, we use the current Classloader for Configs ...
               * @see net.sf.hibernate.cfg.Configuration#getConfigurationInputStream(java.lang.String)
```

```
           */
          protected InputStream getConfigurationInputStream( String resource )
                  throws HibernateException {
              InputStream stream = null;
              stream = getClass().getResourceAsStream( resource );
              if (stream == null) {
                      throw new HibernateException(resource + " not found");
              }
              return stream;
          }
      };

      config.configure( configFile );
      config.setProperties( hibProps );

      if( updateSchema )
      {
        log.debug( "Updating database schema..." );
        new SchemaUpdate( config ).execute( true, true );
      }
      sessionFactory = config.buildSessionFactory();
        } catch ( Exception e ) {
           throw new ApplicationRuntimeException(e);
        }
}

public Object createCoreServiceImplementation
                     ( ServiceImplementationFactoryParameters params )
{
  try {
     log.debug( "Creating Hibernate Session..." );
     Session session = sessionFactory.openSession();

      Object proxy = Proxy.newProxyInstance(
              params.getInvokingModule().getClassResolver().getClassLoader(),
              new Class[]{ params.getServiceInterface(),
                           StatefulServiceLifecycleListener.class,
                           Discardable.class,
                           PoolManageable.class },
              new SessionProxy( session ) );

      //threadEventNotifier.addThreadCleanupListener(new SessionCloser(session));
      return proxy;
  } catch ( Exception e ) {
      throw new ApplicationRuntimeException(e);
  }
}

public void registryDidShutdown()
{
            try {
        log.debug( "Closing Hibernate SessionFactory..." );
         sessionFactory.close();
    } catch ( HibernateException e ) {
        throw new ApplicationRuntimeException(e);
    }
}

public void setThreadEventNotifier(ThreadEventNotifier notifier)
{
  this.threadEventNotifier = notifier;
}

public void setLog( Log log )
{
  this.log = log;
}

public void setJndiName( String jn ){
    jndiName = jn;
}
```

```
    public void setUpdateSchema( boolean updateSchema )
    {
      this.updateSchema = updateSchema;
    }
    public boolean isTransactionManager() {
        return transactionManager;
    }
    public void setTransactionManager( boolean showSql ) {
        this.transactionManager = showSql;
    }

    /**
     *  Instanzen der Klasse SessionProxy sind Adapter für Hibernate-Sessions.
     * Sie setzen die Hivemind-Lifecycle-Events
     * in die richtigen Methodenaufrufe der Hibernate-Session um.
     * Es werden die Service-Modelle "pooled", "threaded" und "stateful"
     * unterstützt.
     */
    private class SessionProxy implements InvocationHandler,
                                                                        StatefulServiceLifecycleListener,
                                                                        PoolManageable,
                                                                        Discardable {
        private Session session;
        public SessionProxy( Session session ) {
                    log.debug("creating hibernate session proxy");
            this.session = session;
            try {
                            startTransactionIfNecessary();
                        } catch (HibernateException e) {
                            throw new ApplicationRuntimeException(e);
                        }
        }
        /* (non-Javadoc)
         * @see java.lang.reflect.InvocationHandler#invoke(java.lang.Object, java.lang.reflect.Method, java.
lang.Object[])
         */
        public Object invoke( Object proxy, Method method, Object[] args ) throws Throwable {
            if (method.getDeclaringClass().equals(Session.class))
                return method.invoke( session, args );
            else if ( method.getDeclaringClass().equals(StatefulServiceLifecycleListener.class)
                     || method.getDeclaringClass().equals(PoolManageable.class)
                     || method.getDeclaringClass().equals(Object.class)) {
                return method.invoke( this, args );
            } else
                throw new ApplicationRuntimeException("unable to interprete msg "
                         + method.getName() );
        }

        public void terminateConversation() {
            try {
                log.debug("throwing session away");
                if ( session != null && session.isOpen() ) {
                            endTransactionIfNecessary();
                    session.close();
                    session = null;
                }
            } catch ( HibernateException e ) {
                throw new ApplicationRuntimeException(e);
            }
        }


        public void resumeConversation() {
            try {
                log.debug("reconnecting session");
                if (!session.isConnected())
                    session.reconnect();
                startTransactionIfNecessary();
            } catch ( HibernateException e ) {
                throw new ApplicationRuntimeException(e);
```

```
        }
    }


        public void pauseConversation() {
    try {
        log.debug("disconnecting session");
        if (session.isConnected()) {
                  endTransactionIfNecessary();
            session.disconnect();
        }
    } catch ( HibernateException e ) {
        throw new ApplicationRuntimeException(e);
    }
}


/**
         * @throws HibernateException
         */
        private void startTransactionIfNecessary() throws HibernateException {
                if ( isTransactionManager() && transaction == null )
                        transaction = session.beginTransaction();
        }

/**
         * @throws HibernateException
         */
        private void endTransactionIfNecessary() throws HibernateException {
                if ( isTransactionManager() && transaction != null
                                && ! transaction.wasRolledBack()
                    && ! transaction.wasCommitted()  ) {
                         transaction.commit();
                         transaction = null;
                   }
        }
        public String toString() {
    return super.toString() + " - proxy for " + session.toString();
}

/* (non-Javadoc)
 * @see org.apache.hivemind.PoolManageable#activateService()
 */
public void activateService() {
   // acquire new Hibernate-Session
    log.debug( "Creating Hibernate Session..." );
    try {
        session = getSessionFactory().openSession();
        startTransactionIfNecessary();
    } catch ( HibernateException e ) {
        throw new ApplicationRuntimeException(e);
    }

}
/* (non-Javadoc)
 * @see org.apache.hivemind.PoolManageable#passivateService()
 */
public void passivateService() {
   //throw away Hibernate-Session
   terminateConversation();
}
/* (non-Javadoc)
 * @see org.apache.hivemind.Discardable#threadDidDiscardService()
 */
public void threadDidDiscardService() {
   terminateConversation();
}
}


protected SessionFactory getSessionFactory() {
```

```
        return sessionFactory;
    }
}
```