Object Pooling Service

Summary

Service to provide Pooling function for object. It is used to improve performance if the number of object used on average is few, and the creation cost of object is high and number of creation is high. **Object Pool** is the software design pattern, and is the method of creating and using the suitable number of objects in the available status rather than the type of creating and destroying the objects depending on necessity. Client performs tasks after requesting and obtaining objects to the Pool. After finishing the task performance using the object, return this to pool rather than destroying the object so that other client can use. **Object Pooling** takes high costs for creating objects, high frequency of creating objects, and can improve performance if the number of objects are few.

Description

ObjectPool

ObjectPool is the interface as shown below.

```
public interface ObjectPool {
    Object borrowObject();
    void returnObject(Object borrowed);
}
```

Create and use code by implementing **ObjectPool** interface.

BaseObjectPool

Abstract class implementing ObjectPool.

```
public abstract class BaseObjectPool implements ObjectPool {
    public abstract Object borrowObject() throws Exception;
    public abstract void returnObject(Object obj) throws Exception;
    public abstract void invalidateObject(Object obj) throws Exception;

    public int getNumIdle() throws UnsupportedOperationException {
        return -1;
    }

    public int getNumActive() throws UnsupportedOperationException {
        return -1;
    }

    public void clear() throws Exception, UnsupportedOperationException {
        throw new UnsupportedOperationException();
    }

    public void addObject() throws Exception, UnsupportedOperationException {
        throw new UnsupportedOperationException();
    }

    public void close() throws Exception {
        closed = true;
    }

    public void setFactory(PoolableObjectFactory factory) throws IllegalStateException,
    UnsupportedOperationException {
        throw new UnsupportedOperationException();
    }

    protected final boolean isClosed() {
        return closed;
    }

    protected final void assertOpen() throws IllegalStateException {
        if(isClosed()) {
            throw new IllegalStateException("Pool not open");
        }
    }

    private volatile boolean closed = false;
}
```

METHOD	Description
borrowObject	Assign object
returnObject	Return object
invalidateObject	Validate at the time of object arrangement
getNumIdle	Return number of objects in Idle status
getNumActive	Return number of objects in Active status
clear	Delete objects
addObject	Add objects to Pool
setFactory	Set object implementing PoolableObjectFactory
isClosed	Determine the object close or not
assertOpen	Determine status that can be used by objects

KeyedObjectPool

KeyedObjectPool is the interface for implementing pool that consists of heterogeneous types.

```
public interface KeyedObjectPool {
    Object borrowObject(Object key);
    void returnObject(Object key, Object borrowed);
}
```

METHOD	Description
borrowObject	Assign objects
returnObject	Return object

PoolableObjectFactory

org.apache.commons.pool package supports separated implementation of created and destroyed part of the object created from pool as well as creation of Object Pool object. **PoolableObjecFactory** is the interface for managing survival cycle of pooled object.

```
public interface PoolableObjectFactory {
    Object makeObject();
    void activateObject(Object obj);
    void passivateObject(Object obj);
    boolean validateObject(Object obj);
    void destroyObject(Object obj);
}
```

METHOD	Description
makeObject	Create object
activateObject	Called when assigning the object from Pool(used at the time of re-initialization).
passivateObject	Called when returning the object to Pool(use at the time of initialization).
validateObject	Called to determine whether the object is valid or not.
destroyObject	Called to determine the object.

Program implementing ObjectPool can implement unique and diverse ObjectPool if implemented to receive the program that implemented PoolableObjectFactory.

BasePoolableObjectFactory

Abstract class implementing PoolableObjecFactory.

```
public abstract class BasePoolableObjectFactory implements PoolableObjectFactory {
    public abstract Object makeObject()
        throws Exception;

    /** No-op. */
    public void destroyObject(Object obj)
        throws Exception  {
    }

    /**
     * This implementation always returns <tt>true</tt>.
     * @return <tt>true</tt>
     */
    public boolean validateObject(Object obj) {
        return true;
    }

    /** No-op. */
    public void activateObject(Object obj)
        throws Exception {
    }

    /** No-op. */
    public void passivateObject(Object obj)
        throws Exception {
    }
```

KeyedPoolableObjectFactory

PoolableObjecFactory for KeyedObjectPools.

```
public interface KeyedPoolableObjectFactory {
    Object makeObject(Object key);
    void activateObject(Object key, Object obj);
    void passivateObject(Object key, Object obj);
    boolean validateObject(Object key, Object obj);
    void destroyObject(Object key, Object obj);
}
```

METHOD	Description
makeObject	Create object
activateObject	Called to assign objects from Pool(used to reinitialize).
passivateObject	Called to return the object to Pool(used to initialize).
validateObject	Called to determine whether the object is valid or not.
destroyObject	Called to delete objects.

BaseKeyedPoolableObjectFactory

Abstract class implementing KeyedPoolableObjectFactory.

```
public abstract class BaseKeyedPoolableObjectFactory implements KeyedPoolableObjectFactory {
    public abstract Object makeObject(Object key)
        throws Exception;
    /** No-op. */
    public void destroyObject(Object key, Object obj)
        throws Exception {
    }

    /**
     * This implementation always returns <tt>true</tt>.
     * @return <tt>true</tt>
     */
    public boolean validateObject(Object key, Object obj) {
        return true;
    }

    /** No-op. */
    public void activateObject(Object key, Object obj)
        throws Exception {
    }

    /** No-op. */
    public void passivateObject(Object key, Object obj)
        throws Exception {
    }
}
```

Guide Program

Program Setting

```
<bean id="dbcpObjectpool" class="egovframework.rte.fdl.objectpool.dbcp.dbcpObjectpoolset">
        <property name="pool" ref="dbcpObjectPooler" />
</bean>
<bean id="dbcpObjectPooler" class="egovframework.rte.fdl.objectpool.dbcp.DbcpObjectpool"/>
```

PARAMETER	Description
pool	egovframework.rte.fdl.objectpool.dbcp.DbcpObjectpool

Program Source

DbcpObjectpool.java is the guide program made by using **org.apache.commons.dbcp** package. Source will be described by separating important parts.

```
import org.apache.commons.dbcp.ConnectionFactory;
import org.apache.commons.dbcp.DriverManagerConnectionFactory;
import org.apache.commons.dbcp.PoolableConnectionFactory;
import org.apache.commons.dbcp.PoolingDriver;
import org.apache.commons.pool.impl.GenericObjectPool;
```

Import the **org.apache.commons.dbcp** package.

```
   int maxActive = 1;
   byte whenExhaustedAction = 2;
   long maxWait = 1000;
   int maxIdle = 2;
   int minIdle = 1;
   boolean testOnBorrow = true;
   boolean testOnReturn = true;
   long timeBetweenEvctionRunsMillis = 600000;
   int numTestsPerEvictionRun = 5;
   long minEvictableIdleTimeMillis = 3600000;
   boolean testWhileIdle = true;
```

Define the field to set the information to use when creating ObjectPool.

Setting	Description
maxActive	Maximum number of connection that connection pool will provide
whenExhaustedAction	Designate how it operates when there is no connection that can be brought from connection pool.
maxWait	Maximum number of connection that is not used but can be saved in the pool. There is no restriction if it is negative number.
maxIdle	Minimum number of connection that is not used but can be saved in the pool.
testOnBorrow	Examine whether connection is valid to bring the connection from the connection pool if it is true.
testOnReturn	Examine whether the connection is valid when returning the connection to the pool if it is true.
timeBetweenEvctionRunsMillis	Designate the execution interval of thread that extracts the connection not used.
numTestsPerEvictionRun	Designate how many unused connection to examine.
minEvictableIdleTimeMillis	Extract the connection in inactivated status only for over the time designated at this property when extracting the unused connection.
testWhileIdle	Examine whether the connection is valid when extracting the inactivation connection if it is true, and remove the invalid connection from the pool.

```
private void loadProperties() throws IOException
{
    String fileName_ =
Thread.currentThread().getContextClassLoader().getResource("jdbc.properties").getFile();
    Properties props = null;
    FileInputStream fis = null;

    props = new Properties();
    fis = new FileInputStream(fileName_);
    props.load(new BufferedInputStream(fis));
    fis.close();

    g_Driver = props.getProperty("driver");
    g_URL = props.getProperty("dburl");
    g_User = props.getProperty("username");
    g_Password = props.getProperty("password");
}
```

1. Set jdbc.properties file.
2. Read jdbc_driver,username,password from jdbc.properties.

Get database connection information through jdbc.properties.

```
private void setupDriver(String driver,String url,String user,String password)throws Exception
{
    Class.forName(driver);

    GenericObjectPool connectionPool = new GenericObjectPool(null);
    connectionPool.setMaxActive(maxActive);
    connectionPool.setMaxIdle(maxIdle);
    connectionPool.setMaxWait(maxWait);
    connectionPool.setWhenExhaustedAction(whenExhaustedAction);
    connectionPool.setMinIdle(minIdle);
    connectionPool.setMinEvictableIdleTimeMillis(minEvictableIdleTimeMillis);
    connectionPool.setTimeBetweenEvictionRunsMillis(timeBetweenEvctionRunsMillis);
    connectionPool.setTestOnBorrow(testOnBorrow);
    connectionPool.setTestOnReturn(testOnReturn);
    connectionPool.setTestWhileIdle(testWhileIdle);
    connectionPool.setNumTestsPerEvictionRun(numTestsPerEvictionRun);
```

```
  ConnectionFactory connectionFactory =
                    new DriverManagerConnectionFactory(url,user,password);

   PoolableConnectionFactory poolableConnectionFactory =
            new PoolableConnectionFactory(connectionFactory,
                                          connectionPool,
                                          null,
                                          null,
                                          false,
                                          true);

   PoolingDriver poolingDriver = new PoolingDriver();

  poolingDriver.registerPool("database", connectionPool);
}
```

1. setMaxWait : maximum waiting time
2. setWhenExhaustedAction : set how to process if there is no object to assign in pool
3. setMinIdle : minimum number of objects to be saved, not used in the pool
4. setMinEvictableIdleTimeMillis : extract the objects in inactivated status only for over the time designated at this property when extracting the unused connection
5. setTimeBetweenEvictionRunsMillis : designate the execution frequency of thread extracting the unused connection
6. setTestOnBorrow : if it is true, examine whether the connection is valid when borrowing connection from the pool
7. setTestOnReturn : if it is true, examine whether the connection is valid when returning the connection to the connection pool
8. setTestWhileIdle : examine whether the connection is valid when extracting the inactivation connection if it is true, and remove the invalid connection from the pool
9. setNumTestsPerEvictionRun : designate how many unused connection to examine
10. Create the factory connecting actual DB
11. Create PoolableConnectionFactory to use when Connection Pool creates PoolableConnection object
12. Create PoolingDriver for Pooling
13. Register connection pool to PoolingDriver

Set the setting information defined above by creating GenericObjectPool(ObjectPool) and calling the relevant methods respectively. Use PoolableConnectionFactory implementing PoolableObjectFactory.

DbcpObjectpoolTest.java is the test program using guide program.

```
@Resource(name = "dbcpObjectPooler")
private DbcpObjectpool pool;
/**
 * Object pool test module
 * @see development framework execution environment development team
 */
@Test
public void runModule()
{
        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        String sql = "select * from emp";
        try
        {
                con = pool.getConnection();
                pstmt = con.prepareStatement(sql);
                rs = pstmt.executeQuery();
                logger.debug("module1");
                while(rs.next())
```

```
                    {
                              logger.debug(rs.getString("ename"));
                    }
          }catch(Exception e)
          {
                    e.printStackTrace();
          }finally
          {
                    pool.freeConnection(con, pstmt,rs);
          }
}
```

Execution Sequence of Guide Program

1. Set the Classpath of egovframework.rte.fdl.objectpool-1.0.0-SNAPSHOT.jar in execution management Repository.
2. Create spring folder and set Classpath.
3. Save context-dbcppool.xml in the spring folder in Execution Management Repository.
4. Save jdbc.properties to the location you want and update properties contents.
5. Execute DbcpObjectpoolTest.java.(test by updating DbcpObjectpoolTest.java contents.)

Reference