10.0

```
package com.fr.data.impl;

import com.fr.base.TemplateUtils;
import com.fr.data.pool.DBCPConnectionPoolAttr;
import com.fr.data.pool.MemoryConnection;
import com.fr.general.ComparatorUtils;
import com.fr.general.FRLogger;
import com.fr.general.Inter;
import com.fr.json.JSONException;
import com.fr.json.JSONObject;
import com.fr.stable.CodeUtils;
import com.fr.stable.StringUtils;
import com.fr.stable.xml.XMLPrintWriter;
import com.fr.stable.xml.XMLableReader;
import holder.ConfigHolder;
import holder.NameSpaceWrapper;
import holder.factory.ConfigHolderFactory;
import holder.impl.ObjectHolder;

import java.sql.Connection;

/**
 * JDBC Database Connection
 */
public class JDBCDatabaseConnection extends AbstractDatabaseConnection {
   private ConfigHolder<String> driver = HolderFactory.simpleHolder("driver", "sun.jdbc.odbc.JdbcOdbcDriver",
getNameSpace());
        private ConfigHolder<String> url = HolderFactory.simpleHolder("url", StringUtils.EMPTY, getNameSpace());
        private ConfigHolder<String> user = HolderFactory.simpleHolder("user", StringUtils.EMPTY,
getNameSpace());
        private ConfigHolder<String> password = HolderFactory.simpleHolder("password", StringUtils.EMPTY,
getNameSpace());
        private ConfigHolder<Boolean> encryptPassword = HolderFactory.simpleHolder("encryptPassword", true,
getNameSpace());
        private ConfigHolder<DBCPConnectionPoolAttr> dbcpAttr = HolderFactory.objectHolder("dbcpAttr", null,
DBCPConnectionPoolAttr.class, getNameSpace());

    /**
     * Constructor.
     */
    public JDBCDatabaseConnection() {
        this(StringUtils.EMPTY, StringUtils.EMPTY, StringUtils.EMPTY, StringUtils.EMPTY);
    }

    /**
     * Constructor.
     */
    public JDBCDatabaseConnection(String driver, String url,
                                  String user, String password) {
        this.setDriver(driver);
        this.setURL(url);
        this.setUser(user);
        this.setPassword(password);
    }

}
```

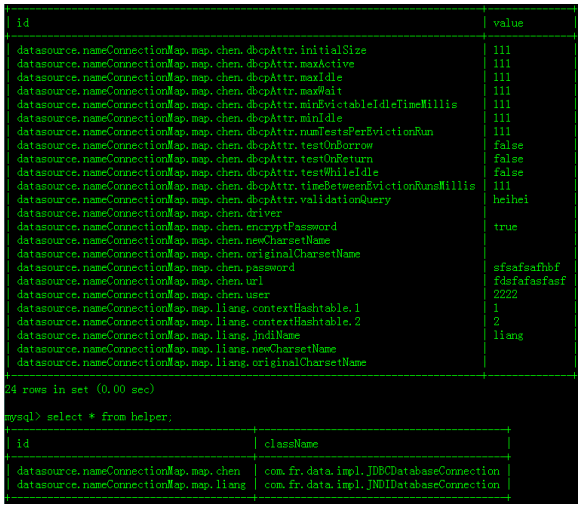
DataSourcemanger

```
public class DataSourceRefactor extends Configuration{
    private ConfigHolder<ConnectionN> nameConnectionMap = HolderFactory.objectHolder("nameConnectionMap", new
ConnectionN(), ConnectionN.class, getNameSpace());
```

```
        public abstract static class N extends UniqueKey {
    MapHolder<Map> map = HolderFactory.mapHolder("map", new ListMap(), null, getNameSpace()); // <String, <T
extends XMLable>>

    public N() {
    }

        public String getNameSpace(){
        return "datasource";
}
    /**
     * map
     *
     * @return
     */
    public int size() {
        return map.get().size();
    }

    /**
     *
     *
     * @return
     */
    public java.util.Iterator getNameIterator() {
        return this.map.get().keySet().iterator();
    }

    /**
     *
     *
     * @param name
     * @return
     */
    public XMLable getValueByName(String name) {
        return (XMLable) this.map.get(name);
    }

    /**
     *
     *
     * @param name
     * @param value
     */
    public void putValueByName(String name, XMLable value) {
        if (value == null) {
            this.map.remove(name);
        } else {
            this.map.put(name, value);
        }
    }

    /**
     *
     *
     * @param oldName
     * @param newName
     * @return
     */
    public boolean rename(String oldName, String newName) {
        map.rename(oldName, newName);
        return true;
    }

    /**
     *
     *
     * @param name
     */
    public void removeValueByName(String name) {
        this.map.remove(name);
```

```
    }

    /**
     *
     */
    public void clearAll() {
        this.map.clearAll();
    }

}

public static class ConnectionN extends N {
    public ConnectionN() {
    }

}
public static void main(String[] args) {
    DataSourceRefactor dataSourceRefactor = new DataSourceRefactor();
    DBCPConnectionPoolAttr dbcpConnectionPoolAttr = new DBCPConnectionPoolAttr();
    dbcpConnectionPoolAttr.setInitialSize(111);
    dbcpConnectionPoolAttr.setMaxActive(111);
    dbcpConnectionPoolAttr.setMaxIdle(111);
    dbcpConnectionPoolAttr.setMaxWait(111);
    dbcpConnectionPoolAttr.setMinIdle(111);
    dbcpConnectionPoolAttr.setMinEvictableIdleTimeMillis(111);
    dbcpConnectionPoolAttr.setNumTestsPerEvictionRun(111);
    dbcpConnectionPoolAttr.setTimeBetweenEvictionRunsMillis(111);
    dbcpConnectionPoolAttr.setValidationQuery("heihei");
    dbcpConnectionPoolAttr.setTestOnBorrow(false);
    dbcpConnectionPoolAttr.setTestOnReturn(false);
    dbcpConnectionPoolAttr.setTestWhileIdle(false);
    JDBCDatabaseConnection jndiDatabaseConnection = new JDBCDatabaseConnection();
    jndiDatabaseConnection.setDbcpAttr(dbcpConnectionPoolAttr);
    jndiDatabaseConnection.setUser("2222");
    jndiDatabaseConnection.setURL("fdsfafasfasf");
    jndiDatabaseConnection.setPassword("sfsafsafhbf");
    dataSourceRefactor.putConnection("chen", jndiDatabaseConnection);
    JNDIDatabaseConnection jndiDatabaseConnection1 = new JNDIDatabaseConnection();
    jndiDatabaseConnection1.setJNDIName("liang");
    Hashtable hashtable = new Hashtable();
    hashtable.put("1", "1");
    hashtable.put("2", "2");
    jndiDatabaseConnection1.setContextHashtable(hashtable);
    dataSourceRefactor.putConnection("liang", jndiDatabaseConnection1);
}
}
```

main,



id	value
datasource.nameConnectionMap.map.chen.dbcpAttr.initialSize	111
datasource.nameConnectionMap.map.chen.dbcpAttr.maxActive	111
datasource.nameConnectionMap.map.chen.dbcpAttr.maxIdle	111
datasource.nameConnectionMap.map.chen.dbcpAttr.maxWait	111
datasource.nameConnectionMap.map.chen.dbcpAttr.minEvictableIdleTimeMillis	111
datasource.nameConnectionMap.map.chen.dbcpAttr.minIdle	111
datasource.nameConnectionMap.map.chen.dbcpAttr.numTestsPerEvictionRun	111
datasource.nameConnectionMap.map.chen.dbcpAttr.testOnBorrow	false
datasource.nameConnectionMap.map.chen.dbcpAttr.testOnReturn	false
datasource.nameConnectionMap.map.chen.dbcpAttr.testWhileIdle	false
datasource.nameConnectionMap.map.chen.dbcpAttr.timeBetweenEvictionRunsMillis	111
datasource.nameConnectionMap.map.chen.dbcpAttr.validationQuery	heihei
datasource.nameConnectionMap.map.chen.driver	
datasource.nameConnectionMap.map.chen.encryptPassword	true
datasource.nameConnectionMap.map.chen.newCharsetName	
datasource.nameConnectionMap.map.chen.originalCharsetName	
datasource.nameConnectionMap.map.chen.password	sfsafsafhbf
datasource.nameConnectionMap.map.chen.url	fdsfafasfasf
datasource.nameConnectionMap.map.chen.user	2222
datasource.nameConnectionMap.map.liang.contextHashtable.1	1
datasource.nameConnectionMap.map.liang.contextHashtable.2	2
datasource.nameConnectionMap.map.liang.jndiName	liang
datasource.nameConnectionMap.map.liang.newCharsetName	
datasource.nameConnectionMap.map.liang.originalCharsetName	

24 rows in set (0.00 sec)

mysql> select * from helper;

id	className
datasource.nameConnectionMap.map.chen	com.fr.data.impl.JDBCDatabaseConnection
datasource.nameConnectionMap.map.liang	com.fr.data.impl.JNDIDatabaseConnection

```
public class DBCPConnectionPoolAttr extends UniqueKey implements XMLReadable, Cloneable, java.io.Serializable {

    private static final int MAX_ACTIVE = 50;
    private static final int MAX_IDLE = 10;
    private static final int MAX_WAIT = 10000;
    private static final int MIN_EVICTABLE_IDLE_TIME_MILLIS = 1000 * 60 * 30;

    private ConfigHolder<Integer> initialSize = HolderFactory.simpleHolder("initialSize", 0, getNameSpace());
// :,
private ConfigHolder<Integer> maxActive = HolderFactory.simpleHolder("maxActive", 50, getNameSpace());
// :,,
private ConfigHolder<Integer> maxIdle = HolderFactory.simpleHolder("maxIdle", 10, getNameSpace());
// :,,0
private ConfigHolder<Integer> minIdle = HolderFactory.simpleHolder("minIdle", 0, getNameSpace());
// :,(),,-1
private ConfigHolder<Integer> maxWait = HolderFactory.simpleHolder("maxWait", 10000, getNameSpace());
// SQL,,.,SQL SELECT
private ConfigHolder<String> validationQuery = HolderFactory.simpleHolder("validationQuery", StringUtils.EMPTY,
getNameSpace());
// ,,.
// : true,validationQuery
private ConfigHolder<Boolean> testOnBorrow = HolderFactory.simpleHolder("testOnBorrow", true, getNameSpace());
//
// : true,validationQuery
private ConfigHolder<Boolean> testOnReturn = HolderFactory.simpleHolder("testOnReturn", true, getNameSpace());
// ().,.
// : true,validationQuery
private ConfigHolder<Boolean> testWhileIdle = HolderFactory.simpleHolder("testWhileIdle", false,
getNameSpace());
// ,. ,
private ConfigHolder<Integer> timeBetweenEvictionRunsMillis = HolderFactory.simpleHolder
("timeBetweenEvictionRunsMillis", -1, getNameSpace());
// ()
private ConfigHolder<Integer> numTestsPerEvictionRun = HolderFactory.simpleHolder("numTestsPerEvictionRun", 3,
getNameSpace());
// ()
private ConfigHolder<Integer> minEvictableIdleTimeMillis = HolderFactory.simpleHolder
("minEvictableIdleTimeMillis", 1000 * 60 * 30, getNameSpace());

    public int getInitialSize() {
        return initialSize.get();
    }

    public void setInitialSize(int initialSize) {
        this.initialSize.set(initialSize);
    }

    public int getMaxActive() {
        return maxActive.get();
    }

    public DBCPConnectionPoolAttr() {
    }

    /**
     *
     *
     * @param maxActive
     */
    public void setMaxActive(int maxActive) {
        this.maxActive.set(maxActive);
    }

    public int getMaxIdle() {
        return maxIdle.get();
    }
```

```
/**
 * @param maxIdle
 */
public void setMaxIdle(int maxIdle) {
    this.maxIdle.set(maxIdle);
}

public int getMinIdle() {
    return minIdle.get();
}

public void setMinIdle(int minIdle) {
    this.minIdle.set(minIdle);
}

public int getMaxWait() {
    return maxWait.get();
}

public void setMaxWait(int maxWait) {
    this.maxWait.set(maxWait);
}

public String getValidationQuery() {
    return validationQuery.get();
}

public void setValidationQuery(String validationQuery) {
    this.validationQuery.set(validationQuery);
}

/**
 *
 *
 * @return true
 */
public boolean isTestOnBorrow() {
    return testOnBorrow.get();
}

public void setTestOnBorrow(boolean testOnBorrow) {
    this.testOnBorrow.set(testOnBorrow);
}

/**
 *
 *
 * @return true
 */
public boolean isTestOnReturn() {
    return testOnReturn.get();
}

public void setTestOnReturn(boolean testOnReturn) {
    this.testOnReturn.set(testOnReturn);
}

/**
 * ()
 *
 * @return true
 */
public boolean isTestWhileIdle() {
    return testWhileIdle.get();
}

public void setTestWhileIdle(boolean testWhileIdle) {
    this.testWhileIdle.set(testWhileIdle);
}
```

```
    public int getTimeBetweenEvictionRunsMillis() {
        return timeBetweenEvictionRunsMillis.get();
    }

    public void setTimeBetweenEvictionRunsMillis(int timeBetweenEvictionRunsMillis) {
        this.timeBetweenEvictionRunsMillis.set(timeBetweenEvictionRunsMillis);
    }

    public int getNumTestsPerEvictionRun() {
        return numTestsPerEvictionRun.get();
    }

    public void setNumTestsPerEvictionRun(int numTestsPerEvictionRun) {
        this.numTestsPerEvictionRun.set(numTestsPerEvictionRun);
    }

    public int getMinEvictableIdleTimeMillis() {
        return minEvictableIdleTimeMillis.get();
    }

    public void setMinEvictableIdleTimeMillis(int minEvictableIdleTimeMillis) {
        this.minEvictableIdleTimeMillis.set(minEvictableIdleTimeMillis);
    }
}
```