com.fr.db.fun.DBAccessProvider

finedbDBAccessProviderfinedb

DBAccessProvider.java

```
package com.fr.db.fun;


import com.fr.plugin.db.accessor.DBAccessible;
import com.fr.stable.fun.mark.Mutable;

/**
 * Created by loy on 2018/8/21.
 *
 *
 */
public interface DBAccessProvider extends Mutable, DBAccessible {

    String XML_TAG = "DBAccessProvider";

    int CURRENT_LEVEL = 1;

}
```

DBAccessible.java

```
package com.fr.plugin.db.accessor;

import com.fr.stable.db.dao.DAOProvider;

/**
 * Created by loy on 2018/9/29.
 */
public interface DBAccessible extends DBInjectable {

    DAOProvider[] registerDAO();
}
```

DBInjectable.java

```
package com.fr.plugin.db.accessor;

import com.fr.stable.db.accessor.DBAccessor;

/**
 * Created by loy on 2018/8/23.
 *
 *
 */
public interface DBInjectable {

    /**
     *
     * @param accessor
     */
    void onDBAvailable(DBAccessor accessor);
}
```

BaseDAO.java

```
package com.fr.stable.db.dao;

import com.fr.stable.db.data.DataRecord;
import com.fr.stable.db.session.DAOSession;
import com.fr.stable.query.QueryFactory;
import com.fr.stable.query.condition.QueryCondition;
import com.fr.stable.query.data.DataColumn;
import com.fr.stable.query.data.DataList;
import com.fr.stable.query.restriction.RestrictionFactory;

import java.util.List;
import java.util.Map;

/**
 * Created by loy on 2017/10/20.
 * <p>
 * DAO
 */
public abstract class BaseDAO<T extends DataRecord> implements DAO<T> {

    private DAOSession session;

    public BaseDAO(DAOSession session) {
        this.session = session;
    }

    @Override
    public DAOSession getSession() {
        return session;
    }

    /**
     * getEntityClassDAO<T>
     * getEntityClass
     */
    protected Class<T> getEntityClass() {
        return null;
    }

    @Override
    public void add(T record) throws Exception {
```

```
        getSession().persist(record);
    }

    @Override
    public T getById(String id) throws Exception {
        checkEntityClass();
        return getSession().getById(id, getEntityClass());
    }

    @Override
    public void update(T record) throws Exception {
        getSession().merge(record);
    }

    @Override
    public void update(Map<String, Object> columnMap, QueryCondition queryCondition) throws Exception {
        getSession().update(columnMap, queryCondition, getEntityClass());
    }

    @Override
    public void remove(String id) throws Exception {
        checkEntityClass();
        getSession().remove(
                QueryFactory.create().addRestriction(RestrictionFactory.eq(T.COLUMN_ID, id)),
                getEntityClass()
        );
    }

    @Override
    public void remove(QueryCondition queryCondition) throws Exception {
        checkEntityClass();
        getSession().remove(queryCondition, getEntityClass());
    }

    @Override
    public List<T> find(QueryCondition queryCondition) throws Exception {
        checkEntityClass();
        return getSession().find(queryCondition, getEntityClass());
    }

    @Override
    public T findOne(QueryCondition queryCondition) throws Exception {
        checkEntityClass();
        return getSession().findOne(queryCondition, getEntityClass());
    }

    @Override
    public DataList<T> findWithTotalCount(QueryCondition queryCondition) throws Exception {
        checkEntityClass();
        return getSession().findWithTotalCount(queryCondition, getEntityClass());
    }

    @Override
    public long count(QueryCondition queryCondition) throws Exception {
        checkEntityClass();
        return getSession().count(queryCondition, getEntityClass());
    }

    @Override
    public List<Object> findInProjection(QueryCondition queryCondition, String... columns) throws Exception {
        checkEntityClass();
        return getSession().findInProjection(queryCondition, getEntityClass(), columns);
    }

    @Override
    public List<Object> findInProjection(QueryCondition queryCondition, DataColumn... columns) throws Exception
{
        checkEntityClass();
        return getSession().findInProjection(queryCondition, getEntityClass(), columns);
    }
```

```
    @Override
    public void addOrUpdate(T record) throws Exception {
        if (record != null && getById(record.getId()) != null) {
            update(record);
        } else {
            add(record);
        }
    }

    private void checkEntityClass() {
        if (getEntityClass() == null) {
            throw new AssertionError("entity class is null, see introduction of BaseDAO#getEntityClass()");
        }
    }
}
```

BaseEntity.java

```
package com.fr.decision.backup;

import com.fr.stable.db.data.DataRecord;
import com.fr.third.javax.persistence.Column;
import com.fr.third.javax.persistence.Id;
import com.fr.third.javax.persistence.MappedSuperclass;

/**
 * Created by Zed on 2017/12/19.
 *
 */
@MappedSuperclass
public abstract class BaseEntity implements DataRecord {

    public static final String COLUMN_ID = "id";

    @Id
    @Column(name = COLUMN_ID, nullable = false)
    private String id = null;

    @Override
    public String getId() {
        return id;
    }

    @Override
    public void setId(String id) {
        this.id = id;
    }
}
```

DAOProvider.java

```
package com.fr.stable.db.dao;

import com.fr.stable.db.data.DataRecord;

/**
 * Created by loy on 2018/8/20.
 */
public interface DAOProvider<T extends DataRecord> {

    Class<T> getEntityClass();

    Class<? extends BaseDAO<T>> getDAOClass();
}
```

DBAccessor.java

```
package com.fr.stable.db.accessor;

import com.fr.stable.db.action.DBAction;

/**
 * Created by loy on 2018/8/21.
 *
 *
 */
public interface DBAccessor {

    /**
     *
     */
    <T> T runQueryAction(DBAction<T> action) throws Exception;

    /**
     *
     */
    <T> T runDMLAction(DBAction<T> action) throws Exception;

}
```

DBAction.java

```
package com.fr.stable.db.action;

import com.fr.stable.db.dao.DAOContext;

/**
 * Created by loy on 2018/8/21.
 */
public interface DBAction<T> {

    T run(DAOContext context) throws Exception;

}
```

FR	10.0		

plugin.xml

```
<extra-core>
        <DBAccessProvider class="your class name"/>
</extra-core>
```

PluginDBManagerDBContextPluginDBManagerPluginDBManager

PluginDBManager.java

```
package com.fr.plugin.db;

import com.fr.properties.finedb.FineDBProperties;
import com.fr.log.FineLoggerFactory;
import com.fr.plugin.context.PluginContext;
import com.fr.plugin.db.base.BasePluginDBManager;
import com.fr.plugin.injectable.PluginModule;
import com.fr.stable.db.DBContext;
import com.fr.stable.db.session.DAOSession;
import com.fr.stable.db.session.DAOSessionStore;
import com.fr.stable.db.transaction.TransactionProvider;
import com.fr.stable.plugin.ExtraClassManagerProvider;
import com.fr.db.fun.DBAccessProvider;

import java.util.Set;

/**
 * Created by loy on 2018/8/17.
 */
public class PluginDBManager extends BasePluginDBManager {

    private static final PluginDBManager instance = new PluginDBManager();

    public static PluginDBManager getInstance() {
        return instance;
    }

    private PluginDBManager() {
    }

    private DBContext dbContext = DBContext.create();
    private DAOSessionStore sessionStore = new DAOSessionStore(dbContext);

    @Override
    protected void onInit() {
        ExtraClassManagerProvider provider = (ExtraClassManagerProvider) PluginModule.ExtraCore.getAgent();
        Set<DBAccessProvider> dbAccessProviders = provider.getArray(DBAccessProvider.XML_TAG);
        if (dbAccessProviders.size() > 0) {
            for (DBAccessProvider dbp : dbAccessProviders) {
                loadPluginEntities(dbp);
            }
            try {
                dbContext.init(FineDBProperties.getInstance().get());
                for (DBAccessProvider dbp : dbAccessProviders) {
                    prepareDB(dbp);
                }
            } catch (Exception e) {
                FineLoggerFactory.getLogger().error(e.getMessage(), e);
            }
        }
    }

    @Override
    protected void onDestroy() {
        dbContext.destroy();
    }

    public boolean isDBActive() {
        return dbContext.isRunning();
    }

    public DBContext getDbContext() {
        return dbContext;
    }

    protected void dealDynamicPluginInstall(PluginContext pluginContext) {
        Set<DBAccessProvider> dbAccessProviders = pluginContext.getRuntime().get(DBAccessProvider.XML_TAG);
        if (dbAccessProviders == null || dbAccessProviders.isEmpty()) {
            return;
        }
        for (DBAccessProvider dbAccessProvider : dbAccessProviders) {
```

```
            loadPluginEntities(dbAccessProvider);
        }
        try {
            if (dbContext.isRunning()) {
                dbContext.reload();
            } else {
                dbContext.init(FineDBProperties.getInstance().get());
            }
            for (DBAccessProvider dbAccessProvider : dbAccessProviders) {
                prepareDB(dbAccessProvider);
            }
        } catch (Exception e) {
            FineLoggerFactory.getLogger().error(e.getMessage(), e);
        }
    }

    protected void dealDynamicPluginUninstall(PluginContext pluginContext) {
        Set<DBAccessProvider> dbAccessProviders = pluginContext.getRuntime().get(DBAccessProvider.XML_TAG);
        if (dbAccessProviders == null || dbAccessProviders.isEmpty()) {
            return;
        }
        for (DBAccessProvider dbAccessProvider : dbAccessProviders) {
            unloadPluginEntities(dbAccessProvider);
        }
        try {
            if (dbContext.getEntityClasses().isEmpty()) {
                dbContext.destroy();
            } else {
                dbContext.reload();
            }
        } catch (Exception e) {
            FineLoggerFactory.getLogger().error(e.getMessage(), e);
        }
    }

    @Override
    protected DBContext getDBContext() {
        return dbContext;
    }

    @Override
    protected TransactionProvider getTransactionProvider() {
        return sessionStore;
    }

    @Override
    protected DAOSession getDAOSession() {
        return sessionStore.getDAOSession();
    }
}
```

DBAccessProvidervoid onDBAvailable(DBAccessor accessor)DBAccessor

entity

```
@Entity  //
@Table(name = "plugin_demo") //
@TableAssociation(associated = true) //
public class DemoEntity extends BaseEntity {

    public static final String COLUMN_KEY = "key";
    public static final String COLUMN_BEAN = "bean";

    @Column(name = COLUMN_KEY) //
    private String key = null;

    @Column(name = COLUMN_BEAN)
    @Convert(converter = BeanConverter.class)//BaseConverter
    private DemoBean bean = null;
    ...
}
```

BaseDaoBaseDaoDBAccessProvider

BaseDaoDao

DBAccessProviderDBController

DBAccessProviderEmbedDBAccessProviderDAODBAccessProviderDBAccessProvider

demodemo-db-access-provider

open-JSD-8016

open-JSD-7957

open-JSD-7858

open-JSD-7843

open-JSD-7868

open-JSD-7747

demo-db-access