

ARTIGO: 11369

Manipulação de dados

Introdução

Nesse artigo veremos como realizar a manipulação de dados através do Veloster Framework, ou seja, CRUD e consultas.

Interface de Manipulação

Java Code

br.com.mobilemind.veloster.orm.Veloster<T extends Entity>

Através dessa interface executamos insert, update, delete e select no banco de dados. Essa interface já vem preparada com diversos métodos auto-explicativos para uso:

Java Code

public interface Veloster<T extends Entity>{

 /**
 * delete entity and put entity like internal entity
 *
 */
 void delete(T entity);

 /**
 * delete internal entity by criteria
 *
 */
 void deleteByCriteria(Criteria<T> criteria);

 /**
 * update entity and put entity like internal entity
 *
 */
 void update(T entity);

 /**
 * save entity and put entity like internal entity
 *
 * @return
 */
 T save(T entity);

 /**
 * load entity by id and put like internal entity
 *
 * @param id
 * @return
 */
 T load(Long id);

 /**
 * reload entity by id and put entity like internal entity
 *
 * @param entity
 * @return
 */
 T load(T entity);

 /**
 * load entity by id and put entity like internal entity
 *
 * @return
 */
 T loadByCriteria(Criteria<T> critaria);

 /**
 * execute select count(*) in data base
 *
 * @return
 */
 int count();

 /**
 * execute select count(*) by criteria in data base
 *
 * @return
 */
 int countByCriteria(Criteria<T> critaria);

 /**
 * list all entities
 *
 * @return
 */
 List<T> list();

 /**
 * list all entities paged
 *
 * @param max max of results
 * @param first first result
 * @return
 */
 List<T> list(int limit, int offset);

 /**

```

    * list all entities by criteria
    *
    * @return
    */
List<T> listByCriteria(Criteria<T> critaria);

/**
 * list all entities paged by criteria
 *
 * @param max max of results
 * @param first first result
 * @return
 */
List<T> listByCriteria(Criteria<T> critaria, int limit, int offset);

/**
 * Execute native query
 *
 * @param query
 * @param query parameters
 */
void executeNativeQuery(String query, Object... params);

/**
 * Execute native query
 *
 * @param query
 * @param query parameters
 */
List<Object[]> executeNativeSelect(String query, Object... args);

/**
 * Execute native query and transform the result at
 * <code>classToTransformer</code>
 *
 * @param query
 * @param classToTransformer class to transform result
 * @param query parameters
 */
<E> List<E> executeNativeTransformer(String query, Class<E> resultTransformer, Object... params);

/**
 * Execute native query and transform the result at
 * <code>classToTransformer</code>
 *
 * @param query
 * @param classToTransformer class to transform result
 * @param query parameters
 */
<E> E executeNativeSingleTransformer(String query, Class<E> resultTransformer, Object... params);

/**
 * executa DDL create table
 *
 */
void tableCreate();

/**
 * execute DDL update table
 *
 */
void tableUpdate();

/**
 * executa DDL drop table
 *
 */
void tableDelete();

/**
 * verify if table exists
 *
 * @return
 */
boolean tableExists();

/**
 * name of table
 *
 * @return
 */
String tableName();

/**
 * run transactional operation
 *
 */
void runTrans(TransactionOperation operation);

/**
 * add insert listener
 *
 * @param listener
 */
void addInsertListener(InsertListener listener);

/**
 * remove insert listener
 *
 * @param listener
 */
void removeInsertListener(InsertListener listener);

```

```

/**
 * add update listener
 *
 * @param listener
 */
void addUpdateListener(UpdateListener listener);

/**
 * remove update listener
 *
 * @param listener
 */
void removeUpdateListener(UpdateListener listener);

/**
 * add delete listener
 *
 * @param listener
 */
void addDeleteListener(DeleteListener listener);

/**
 * remove delete listener
 *
 * @param listener
 */
void removeDeleteListener(DeleteListener listener);

/**
 * add meta data change listener
 *
 * @param listener
 */
void addMetadataUpdateListener(MetadataUpdateListener listener);

/**
 * remove meta data change listener
 *
 * @param listener
 */
void removeMetadataUpdateListener(MetadataUpdateListener listener);

/**
 * create criteria
 *
 * @return
 */
Criteria<T> createCriteria();

/**
 * get class validator
 *
 * @return
 */
EntityValidator getValidator();
}

```

A interface *Veloster* é a porta de comunicação com o framework. O Framework já trás sua implementação padrão para ela:

Java Code

```
br.com.mobilemind.veloster.orm.core.VelosterImpl<T extends Entity> implements Veloster<T>
```

Com essa implementação já podemos utilizar todos os recursos oferecidos pelo Framework:

Java Code

```
Veloster<PersonGroup> veloster = VelosterRepository.getVeloster(PersonGroup.class);
PersonGroup group = new PersonGroup();
veloster.save(group);
PersonGroup groups = veloster.list();
```