**Hibernate Generic D.A.O. Framework**

**The motivation behind the framework**

我们一直在项目中手敲daos各个类的代码，这个工具可以解决四个问题：方法名和实现总是很难完全一致/对应的。对额外的列进行排序和过滤是艰难的，所以很多页面都缺少好的排序和过滤。Dao类的编写是冗繁的而且花费大量的时间。Dao类的测试困难而且冗繁，所以很少人真正去对dao类进行测试.

**Why might you consider looking into this framework?**

* ***Generic DAO:*** java语言在generics（数据类型参数化）支持方面的优雅，使得generic daos算不上一个新概念，也不难，但是我们仍然花费了很多的时间去使用令它更易用，如果你喜欢我们的实现方式，你可以直接引用框架的lib，如果你觉得自行定义会更好，那么你可以参考我们的源代码来获得灵感。
* ***Search:*** Search 类是本框架最原创也是最复杂的部分，他甚至可以在没有本框架generic dao 接入点的时候单独使用，Search的查询功能是根据一个列表也中的排序，过滤，列选择，分页等功能需求来进行定义的。当然也不仅限于此，查询结果的组装也更加简单，也能在更少编码的情况下实现复杂查询，在更少测试的情况下保证稳定性。跟hibernate提供的criteria api所提供的功能很想，但是更容易使用，而且更容易跨层使用。此外，它能在hibernate和jpa两种环境下工作。
* ***Remote DAO (for R.I.A.s?):*** 如果你跟我们一样，在你的富客户端应用程序中，不想为每一个写单独的dao类型的远程服务，本框架也提供了一个解决方案，我们提供了将我们的单一general DAO用于一切远程访问接口的功能与示例代码。只需要配置这个唯一的远程访问点，客户端就可以进行基础的增删改查。
* ***Remote Search (for R.I.A.s?):*** 如同我们在上面提到的，本框架为客户端的增删改查提供了单一的接入点，search又可以跨层工作，甚至跨越客户端与服务器端之间，所以在客户端进行列表操作和查询都想在服务器端一样方便，而且语法一致。这种一直习允许我们通过一个单一的Search对象在adobe flex3中创建和重用一个集合类型，并且能够使该集合类型根据查询跳进实现自动的从服务器更新数据\*\*

\*在jpa的支持方面，目前只支持hibernate的jpa实现

**More Information**

Wiki Documentation: [UserGuide](http://code.google.com/p/hibernate-generic-dao/wiki/UserGuide)

Javadoc: <http://hibernate-generic-dao.googlecode.com/svn/trunk/docs/api/index.html>

Blog: <http://hibernategenericdao.wordpress.com/>

**Questions and Comments**

Please post at <http://groups.google.com/group/java-generic-dao>.

**安装与配置**

**Introduction**

Hibernate Generic DAO frame work allows can be used with original Hibernate or the Hibernate JPA implementation. It can also (in theory) be used with other JPA providers (ex: OpenJPA, TopLink) with some tweaking. Here we lay out the different installation options.

**Installation**

在在定义完spring 的application.xml里的关于JPA的配置后，添加如下配置

<!--\*\*\*\* Hibernate Generic DAO Setup \*\*\*\* -->

<!-- Need one search processor and one metadata util per persistence unit

(i.e. EntityManagerFactory). We're wiring these to the Generic DAOs using

an autowire annotation in BaseDAO. -->

<bean class=*"com.googlecode.genericdao.search.jpa.JPASearchProcessor"*

id=*"searchProcessor"*>

<constructor-arg ref=*"metadataUtil"* />

</bean>

<bean

class=*"com.googlecode.genericdao.search.jpa.hibernate.HibernateMetadataUtil"*

id=*"metadataUtil"* factory-method=*"getInstanceForEntityManagerFactory"*>

<constructor-arg ref=*"entityManagerFactory"* />

</bean>

**Maven**

Hibernate Generic DAO is available from the Maven Central repository. Just add it as a dependency.

如果不用jpa

<dependency>  
        <groupId>com.googlecode.genericdao</groupId>  
        <artifactId>dao-hibernate</artifactId>  
        <version>1.2.0</version> <!-- use current version -->  
</dependency>

使用jpa的hibernate实现，用下面的依赖代替

<!—第一个依赖包涵了为了所有dao提供了jpa实现-->  
<dependency>  
        <groupId>com.googlecode.genericdao</groupId>  
        <artifactId>dao</artifactId>  
        <version>1.2.0</version> <!-- use current version -->  
</dependency>  
<!—第二个依赖包是Hibernate Entity Manager 插件-->  
<dependency>  
        <groupId>com.googlecode.genericdao</groupId>  
        <artifactId>search-jpa-hibernate</artifactId>  
        <version>1.2.0</version> <!-- use current version -->  
</dependency>

**Code Examples**

**为独立的model类创建DAO：**

Simply extend the GenericDAO class with the specific type.

public interface ProjectDAO extends GenericDAO<Project, Long> {  
  
}  
  
public class ProjectDAOImpl extends GenericDAOImpl<Project, Long> implements ProjectDAO {  
  
}

**使用示例（在注入ProjectDAOImpl后）：**

Project project = projectDAO.find(projectId);//根据id查找对象  
  
List<Project> list = projectDAO.findAll();//返回所有  
  
projectDAO.save(project);//保存  
  
projectDAO.remove(project);//删除  
  
projectDAO.removeById(project.getId());//根据id删除  
  
------------------------------------------------------------  
Search search = new Search(Project.class);//先声明一个search对象  
search.addFilterEqual("name", "Tu Hao");//其中，name是project这个实体类的一个成员变量，必须有public的getName()方法；这句代码的执行效果相当于：select p from Project p where p.name = ‘Tu Hao’;  
  
List<Project> list = projectDAO.search(search);//返回结果  
---------------------------------------------------------  
int count = projectDAO.count(search);//返回记录的数量  
  
SearchResult<Project> result = projectDAO.searchAndCount(search);//返回一个查询结果  
list = result.getResult();  
count = result.getTotalCount();  
  
search.clear();//清空查询  
search.addField("rating", Field.OP\_AVG);//为查询将诶过增加一列，操作符为求平均数  
int avgProjectRating = (Integer) prjoectDAO.searchUnique(search);

**A GeneralDAO 对所有实体都提供了相应的dao方法:**

public interface GeneralDAO {  
  
        public <T> T find(Class<T> type, Serializable id);  
  
        public <T> T[] find(Class<T> type, Serializable... ids);  
  
        public <T> T getReference(Class<T> type, Serializable id);  
  
        public <T> T[] getReferences(Class<T> type, Serializable... ids);  
  
        public boolean save(Object entity);  
  
        public boolean[] save(Object... entities);  
  
        public boolean remove(Object entity);  
  
        public void remove(Object... entities);  
  
        public boolean removeById(Class<?> type, Serializable id);  
  
        public void removeByIds(Class<?> type, Serializable... ids);  
  
        public <T> List<T> findAll(Class<T> type);  
  
        public List search(ISearch search);  
  
        public Object searchUnique(ISearch search);  
  
        public int count(ISearch search);  
  
        public SearchResult searchAndCount(ISearch search);  
  
        public boolean isAttached(Object entity);  
  
        public void refresh(Object... entities);  
  
        public void flush();  
  
        public Filter getFilterFromExample(Object example);  
  
        public Filter getFilterFromExample(Object example, ExampleOptions options);  
}

**Search DTO usage examples**

Search search = new Search(Project.class);  
  
//filtering  
search.addFilterEqual("name", "hibernate-generic-dao");  
  
search.addFilterLessThan("completionDate", new Date());  
  
search.addFilterOr(  
    Filter.equal("name", "Jack"),  
    Filter.and(  
        Filter.equal("name", "Jill"),  
        Filter.like("location", "%Chicago%"),  
        Filter.greaterThan("age", 5)  
    )  
);  
  
search.addFilterIn("name", "Jack", "Jill", "Bob");  
  
search.addFilterNot(Filter.in("name","Jack", "Jill", "Bob"));  
  
//sorting  
search.addSort("name");  
search.addSort("age", true); //descending  
  
//projection  
search.addField("name");  
search.addField("location");  
  
//or with column operators  
search.addField("rating", Field.OP\_AVG);  
search.addField("developerCount", Field.OP\_MAX);  
  
//paging  
search.setMaxResults(15); //a.k.a. results per page  
search.setPage(3);  
  
//controlling eager fetching of relationships  
serach.addFetch("owner");

**Nested properties are also fully supported...**

search.addFilterEqual("status.name", "active");  
search.addFilterGreaterThan("workgroup.manager.salary", 75000.00);  
  
search.addSort("status.name");