Hibernate仿Mybits SQL/HQL搜索 (注解、XML两不误)

第一次写博客。文章有点渣，喜欢就看看，不喜欢路过点个赞。

先来原理 HQL/SQL + Freemarker 模版生成查询语句。

1：把SQL/HQL写在XML。

2：编写文件扫描器(缺)

3：读取解释XML

4：按实体类空间缓存查询语句

5：直接使用

注意：不要直接复制，先弄懂流程，因为这源于旧版本及测试包来写的，也省略了部分代码，因此包路径有问题。程序也不完整，缺了的自己让大家自己去思考实现。基本上依赖Spring，不依赖Spring注入的可以考虑用代理模式注入（反射/生成字节码/CGLIB等）

1：为了方便先定义约束Query.dtd

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!ELEMENT QueryList (Alias\*,Query\*)>
3. <!--<!ELEMENT Context (CachePool,Bean\*,Intercept\*,ScanToPack\*,CloneModel)>-->
4. <!ELEMENT Alias EMPTY><!--别名-->
5. <!ELEMENT Query (#PCDATA)><!--sql/hql-->
6. <!--QueryList-->
7. <!ATTLIST QueryList package CDATA #REQUIRED>
8. <!--Alias-->
9. <!ATTLIST Alias name CDATA #REQUIRED><!--实体类全名-->
10. <!ATTLIST Alias Alias CDATA #REQUIRED><!--SQL/HQL 语句中的实体类别名-->
11. <!--Query-->
12. <!ATTLIST Query name CDATA #REQUIRED><!--实体类全名-->
13. <!ATTLIST Query type (HQL|SQL) #REQUIRED><!--语句类型：HQL/SQL-->
14. <!ATTLIST Query freemarkFormat (false|true) #REQUIRED><!--是否使用FREEMARK标签格式化-->
15. <!ATTLIST Query resultType CDATA #IMPLIED><!--实体类全名-->
16. <!ATTLIST Query Alias (true|false) #REQUIRED><!--是否使用了类别名-->

2:建个实体类user.class

1. package project.master.user;
2. //import、getting、setting 省略
3. @Entity
4. public class User extends AbstractEntity {
5. private static final long serialVersionUID = 1L;
6. @Id
7. private String id;
8. @Column(unique = true)
9. private String phone;// 用户名(手机号)
10. private String password;
11. private int status;// 帐号状态(锁定、停用、正常)
12. private Date lastLogin;
13. @Column(updatable = false, nullable = false)
14. private Date createDate = new Date();
15. }

3：建立对应的XML (User.query.xml)

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE QueryList SYSTEM "Query.dtd">
3. <QueryList package="project.master.user.User">
4. <Alias name="project.master.user.User" Alias="User" />
5. <Alias name="project.freehelp.common.entity.Dictionary" Alias="Dictionary" />
6. <Query name="list" type="HQL" freemarkFormat="true" Alias="true">
7. <![CDATA[
8. FROM User A
9. WHERE
10. 1=1
11. <#if id??>
12. <#if like??>
13. and A.id like '%'||:id||'%'
14. <#else>
15. and A.id=:id
16. </#if>
17. </#if>
18. <!-- 各字段判断省略 -->
19. ]]>
20. </Query>
21. <Query name="AAX" type="HQL" freemarkFormat="true" Alias="true">
22. <!--测试 -->
23. SELECT A.phone,(SELECT D.value FROM Dictionary D WHERE D.id='1') as xValue FROM User A
24. </Query>
25. <Query name="checkUser" type="HQL" freemarkFormat="false" Alias="true">
26. SELECT COUNT(1) FROM User A WHERE A.phone=:phone
27. </Query>
28. <Query name="login" type="HQL" freemarkFormat="false" Alias="true">
29. FROM User A WHERE A.phone=:phone and A.password=:password
30. </Query>
31. </QueryList>

4:解析缓存XML

1. package com.cheuks.bin.db.manager;
2. //import 省略
3. public class QueryFactory implements QueryType {
4. private final Map<String, Template> FORMAT\_XQL = new ConcurrentHashMap<String, Template>();
5. private final Map<String, String> UNFORMAT\_XQL = new ConcurrentHashMap<String, String>();
6. private final Configuration freemarkerConfiguration = new Configuration(Configuration.VERSION\_2\_3\_0);
7. private StringTemplateLoader stringTemplateLoader = new StringTemplateLoader();
8. private String files;
9. public QueryFactory() {
10. super();
11. freemarkerConfiguration.setTemplateLoader(stringTemplateLoader);
12. }
13. public synchronized void put(String name, String XQL, boolean isFormat) throws TemplateNotFoundException, MalformedTemplateNameException, ParseException, IOException {
14. if (null == name || null == XQL)
15. return;
16. if (isFormat) {
17. stringTemplateLoader.putTemplate(name, XQL);
18. FORMAT\_XQL.put(name, freemarkerConfiguration.getTemplate(name));
19. } else {
20. UNFORMAT\_XQL.put(name, XQL);
21. }
22. }
23. public String getXQL(String name, boolean isFormat, Map<String, Object> params) throws TemplateException, IOException {
24. // if (!isScan)
25. // scan();
26. if (!isFormat)
27. return UNFORMAT\_XQL.get(name);
28. Template tp = FORMAT\_XQL.get(name);
29. if (null == tp)
30. return null;
31. StringWriter sw = new StringWriter();
32. tp.process(params, sw);
33. return sw.toString();
34. }
35. @SuppressWarnings("restriction")
36. @javax.annotation.PostConstruct
37. private void scan() {
38. try {
39. Set<String> o = null;
40. o = Scan.doScan(files);//扫描所有 \*.queue.xml
41. xmlExplain(o);
42. } catch (Exception e) {
43. e.printStackTrace();
44. }
45. }
46. public String getFiles() {return files;}
47. public QueryFactory setFiles(String files) {this.files = files; return this;}
48. public void xmlExplain(Set<String> urls) throws ParserConfigurationException, SAXException, IOException {
49. Iterator<String> it = urls.iterator();
50. SAXParserFactory factory = SAXParserFactory.newInstance();
51. SAXParser parser = factory.newSAXParser();
52. xmlHandler handler = new xmlHandler();
53. XMLReader xmlReader = parser.getXMLReader();
54. //读取XML
55. xmlReader.setEntityResolver(new EntityResolver() {
56. public InputSource resolveEntity(String publicId, String systemId) throws SAXException, IOException {
57. return new InputSource(this.getClass().getClassLoader().getResourceAsStream("dtd/Query.dtd"));
58. }
59. });
60. while (it.hasNext()) {
61. String str = it.next();
62. InputSource is = new InputSource(Thread.currentThread().getContextClassLoader().getResourceAsStream(str));
63. is.setEncoding("utf-8");
64. xmlReader.setContentHandler(handler);
65. xmlReader.parse(is);
66. }
67. }
68. class xmlHandler extends DefaultHandler {
69. // private boolean isHQL = false;
70. private boolean format = false;
71. private boolean alias = false;
72. private String packageName = null;
73. private String name = null;
74. Map<String, String> aliases = new HashMap<String, String>();
75. private String value;
76. @Override
77. public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
78. if (qName.equals(QUERY\_LIST)) {
79. packageName = attributes.getValue(PACKAGE);
80. } else if (qName.equals(QUERY)) {
81. // isHQL = attributes.getValue(TYPE).equals("HQL");
82. name = attributes.getValue(NAME);
83. format = Boolean.valueOf(attributes.getValue(FREEMARK\_FORMAT));
84. alias = Boolean.valueOf(attributes.getValue(ALIAS));
85. } else if (qName.equals(ALIAS)) {
86. aliases.put(attributes.getValue(ALIAS), attributes.getValue(NAME));
87. }
88. super.startElement(uri, localName, qName, attributes);
89. }
90. @Override
91. public void characters(char[] ch, int start, int length) throws SAXException {
92. value = new String(ch, start, length).replaceAll("(\n|\t)", "");
93. if (value.length() > 0) {
94. try {
95. put(String.format("%s.%s", packageName, name).toLowerCase(), alias ? alias(value) : value, format);//生成缓存
96. } catch (Exception e) {
97. }
98. }
99. }
100. private String alias(String str) {
101. if (alias)
102. for (Entry<String, String> en : aliases.entrySet())
103. str = str.replaceAll(en.getKey(), en.getValue());
104. return str;
105. }
106. }
107. }

5：定义 DBAdapter接口。

1. public interface DBAdapter {
2. public DBAdapter setSessionFactory(String name);
3. public <T> List<T> getList(Class<?> c) throws Throwable;
4. /\*\*\*
5. \* query模板查询
6. \*/
7. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, boolean isFormat, Map<String, Object> params) throws Throwable;
8. /\*\*\*
9. \* 模板查询
10. \*/
11. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, Object... params) throws Throwable;
12. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, int page, int size, Object... params) throws Throwable;
13. /\*\*\*
14. \* query模板查询 \* @param queryName 查询语句名
15. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, boolean isFormat, Map<String, Object> params, int page, int size) throws Throwable;
16. public String queryNameFormat(Class<?> entry, String queryName);
17. }

6：写实现(AbstractHibernateDBAdapter、HibernateSingleDBAdapter)

AbstractHibernateDBAdapter

1. package com.cheuks.bin.db.manager;
2. @SuppressWarnings({ "rawtypes", "unchecked" })
3. public abstract class AbstractHibernateDBAdapter implements DBAdapter {
4. private QueryFactory queryFactory;
5. public abstract Session getSession();
6. public <T> List<T> getList(Class<?> c) throws Throwable {
7. return getList(c, -1, -1);
8. }
9. public <T> List<T> getList(Class<?> c, int page, int size) throws Throwable {
10. Query query = getSession().createQuery(String.format("FROM %s a", c.getSimpleName()));
11. List list = page > 0 ? page(query, page, size).list() : query.list();
12. return null == list ? null : list;
13. }
14. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, Object... params) throws Throwable {
15. return getListByXqlQueryName(queryName, isHQL, -1, -1, params);
16. }
17. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, int page, int size, Object... params) throws Throwable {
18. String xql = queryFactory.getXQL(queryName, false, null);
19. Query query = fillParams(isHQL ? getSession().createQuery(xql) : getSession().createSQLQuery(xql), params);
20. List list = page > 0 ? page(query, page, size).list() : query.list();
21. return null == list ? null : list;
22. }
23. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, boolean isFormat, Map<String, Object> params) throws Throwable {
24. return getListByXqlQueryName(queryName, isHQL, isFormat, params, -1, -1);
25. }
26. public <T> List<T> getListByXqlQueryName(String queryName, boolean isHQL, boolean isFormat, Map<String, Object> params, int page, int size) throws Throwable {
27. String xql = queryFactory.getXQL(queryName, isFormat, params);
28. Query query = fillParams(isHQL ? getSession().createQuery(xql) : getSession().createSQLQuery(xql), params);
29. List list = page > 0 ? page(query, page, size).list() : query.list();
30. return null == list ? null : list;
31. }
32. protected Query fillParams(Query q, Object... o) {
33. if (null == o || null == q) {
34. return q;
35. }
36. for (int i = 0, len = o.length; i < len; i++) {
37. q.setParameter(i, o[i]);
38. }
39. return q;
40. }
41. protected Query fillParams(Query q, Map<String, ?> o) {
42. if (null == o || null == q) {
43. return q;
44. }
45. for (Entry<String, ?> en : o.entrySet())
46. try {
47. q.setParameter(en.getKey(), en.getValue());
48. } catch (Exception e) {
49. }
50. return q;
51. }
52. protected Query page(Query q, int pageNum, int size) {
53. if (pageNum >= 0 && size >= 0) {
54. q.setFirstResult(size \* (pageNum - 1));
55. q.setMaxResults(size);
56. }
57. return q;
58. }
59. public String queryNameFormat(Class<?> entry, String queryName) {
60. return String.format("%s.%s", entry.getName(), queryName).toLowerCase();
61. }
62. public QueryFactory getQueryFactory() {
63. return queryFactory;
64. }
65. public AbstractHibernateDBAdapter setQueryFactory(QueryFactory queryFactory) {
66. this.queryFactory = queryFactory;
67. return this;
68. }
69. }

HibernateSingleDBAdapter

1. package com.cheuks.bin.db.manager;
2. public class HibernateSingleDBAdapter extends AbstractHibernateDBAdapter {
3. //待注入 sessionFactory
4. private SessionFactory sessionFactory;
5. public HibernateSingleDBAdapter setSessionFactory(String name) {
6. return this;
7. }
8. @Override
9. public Session getSession() {
10. return sessionFactory.getCurrentSession();
11. }
12. public SessionFactory getSessionFactory() {
13. return sessionFactory;
14. }
15. public HibernateSingleDBAdapter setSessionFactory(SessionFactory sessionFactory) {
16. this.sessionFactory = sessionFactory;
17. return this;
18. }
19. }

7：注入xml

<!-- QueryFile 注入 -->

<bean id=*"queryFactory"* class=*"com.cheuks.bin.db.manager.QueryFactory"*>

<property name=*"files"* value=*"\*.query.xml"* />

</bean>

<!--Single DBAdapter 注入 -->

<bean id=*"dBAdapter"* class=*"com.cheuks.bin.db.manager.HibernateSingleDBAdapter"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"* />

<property name=*"queryFactory"* ref=*"queryFactory"* />

</bean>

8：使用 AbstractDao 、UserDao

AbstractDao

1. package com.cheuks.bin.db.manager.dao;
2. public abstract class AbstractDao<entity, ID extends Serializable> implements BaseDao<entity, ID> {
3. public abstract Class<entity> getEntityClass();
4. public abstract DBAdapter getDBAdapter();
5. public List<entity> getList(int page, int size) throws Throwable {
6. return getDBAdapter().getList(getEntityClass(), page, size);
7. }
8. public List<entity> getList(Map<String, Object> params, int page, int size) throws Throwable {
9. return getDBAdapter().getListByXqlQueryName(getDBAdapter().queryNameFormat(getEntityClass(), "list"), true, true, params, page, size);
10. }
11. public <T> List<T> getList(String queryName, Map<String, Object> params, boolean isFromat, int page, int size) throws Throwable {
12. return getDBAdapter().getListByXqlQueryName(getDBAdapter().queryNameFormat(getEntityClass(), queryName), true, isFromat, params, page, size);
13. }
14. public <T> List<T> getListCustomQueryName(String queryName, Map<String, Object> params, boolean isFromat, int page, int size) throws Throwable { return getDBAdapter().getListByXqlQueryName(queryName.toLowerCase(), true, isFromat, params, page, size);
15. }
16. public List<entity> getListEntity(String queryName, Map<String, Object> params, boolean isFromat, int page, int size) throws Throwable {
17. return getDBAdapter().getListByXqlQueryName(getDBAdapter().queryNameFormat(getEntityClass(), queryName), true, isFromat, params, page, size);
18. }
19. public List<entity> getListEntityCustomQueryName(String queryName, Map<String, Object> params, boolean isFromat, int page, int size) throws Throwable {
20. return getDBAdapter().getListByXqlQueryName(queryName.toLowerCase(), true, isFromat, params, page, size);
21. }
22. }

UserDao

1. package project.freehelp.common.dao.impl;
2. @Component
3. public class UserInfoDaoImpl extends AbstractDao<UserInfo, String> implements UserInfoDao {
4. @Autowired
5. private DBAdapter dBAdapter;
6. @Override
7. public Class<UserInfo> getEntityClass() {
8. return UserInfo.class;
9. }
10. @Override
11. public DBAdapter getDBAdapter() {
12. return dBAdapter;
13. }
14. }

整体就完了。XML部分看DTD。觉得不错可以收藏。但请不要 不名字改了变成自己的成果呀！