1：get/load存取单条数据

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public Teacher getTeacherById(Long id) {
2. return (Teacher)this.hibernateTemplate.get(Teacher.class, id);
3. }
5. public Teacher getTeacherById(Long id) {
6. return (Teacher)this.hibernateTemplate.load(Teacher.class, id);
7. }

get和load方法的****区别****：

1. 对于get方法，**[hibernate](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)**会确认一下该id对应的数据是否存在，首先在session缓存中查找，然后在二级缓存中查找，还没有就查询**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)**，数据库中没有就返回null。这个相对比较简单，也没有太大的争议。主要要说明的一点就是在这个版本中get方法也会查找二级缓存！

2.  load方法加载实体对象的时候，根据映射文件上类级别的****lazy属性****的配置(默认为true)，分情况讨论：

(1)若为true,则首先在Session缓存中查找，看看该id对应的对象是否存在，不存在则使用延迟加载，返回实体的代理类对象(该代理类为实体类的子类，由CGLIB动态生成)。等到具体使用该对象(除获取OID以外)的时候，再查询二级缓存和数据库，若仍没发现符合条件的记录，则会抛出一个ObjectNotFoundException。

(2)若为false,就跟get方法查找顺序一样，只是最终若没发现符合条件的记录，则会抛出一个ObjectNotFoundException。

这里get和load有两个重要区别:

1. 如果未能发现符合条件的记录，get方法返回null，而load方法会抛出一个ObjectNotFoundException。
2. load方法可返回没有加载实体数据的代理类实例，而get方法永远返回有实体数据的对象。(对于load和get方法返回类型：好多书中都说：“get方法永远只返回实体类”，实际上并不正确，get方法如果在session缓存中找到了该id对应的对象，如果刚好该对象前面是被代理过的，如被load方法使用过，或者被其他关联对象延迟加载过，那么返回的还是原先的代理对象，而不是实体类对象，如果该代理对象还没有加载实体数据（就是id以外的其他属性数据），那么它会查询二级缓存或者数据库来加载数据，但是返回的还是代理对象，只不过已经加载了实体数据。)

****总之对于get和load的根本区别，一句话，hibernate对于load方法认为该数据在数据库中一定存在，可以放心的使用代理来延迟加载，如果在使用过程中发现了问题，只能抛异常；而对于get方法，hibernate一定要获取到真实的数据，否则返回null。****

2：find/iterate查询操作

****[html]**** [view](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public Iterator getTeachersByAge(int age) {
2. Iterator iterator = null;
4. //使用find方法
5. List list = (List)this.hibernateTemplate().find("from Teacher t where t.age**>**?", new Integer(age));
6. iterator = list.iterator();
8. //使用iterator方法
9. iterator = this.hibernateTemplate().iterate("from Teacher t where t.age**>**?", new Integer(age));
11. return iterator;
12. }

  find和iterate的区别主要是iterate采用了N+1次查询，对于大批量查询，比如查询10000条记录，那么iterate就要执行10000+1次查询，find和iterate应根据具体的实际

情况来使用，对于频繁的写操作对象，应使用find查询，而对于一些只读的数据对象，应使用iterate操作，因为iterate操作使用了**[hibernate](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)**的缓存机制

3：save/update/saveOrUpdate/delete 保存/更新/删除操作

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public void save(Teacher teacher) {
2. this.hibernateTemplate.save(teacher);
3. }

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public void update(Teacher teacher) {
2. this.hibernateTemplate.update(teacher);
3. }

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public void update(Teacher teacher) {
2. this.hibernateTemplate.saveOrUpdate(teacher);
3. }

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public void update(Teacher teacher) {
2. this.hibernateTemplate.delete(teacher);
3. }

4：bulkUpdate批量删除或者更新

     bulkUpdate提供了批量删除和更新，直接转换为相应的update/delete SQL进行批量删除和更新

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public void batchDelete(String name, int age) {
2. this.hibernateTemplate.bulkUpdate("delete Teacher where name=? and age = ?", new Object[]{name, age});
3. }

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public void batchDelete(String name, String newName) {
2. this.hibernateTemplate.bulkUpdate("update Teacher set name=? where name=?", new Object[]{newName, name});
3. }

 此时要注意的一个问题是，使用bulkUpdate操作，必须手工清除相关对象在Hibernate中的缓存(包括一级缓存和二级缓存)

5：execute核心方法

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public Object execute(HibernateCallBack action, boolean exposeNativeSession) throws DataAccessException {
2. //获取一个Session
3. Session session = getSession();
4. //当前session是否在事务中
5. boolean existingTransaction = SessionFactoryUtils.isSessionTransactional(session, getSessionFactory());
6. FlushMode previousFlushMode = null;
8. try {
9. previousFlushMode = applyFlushMode(session, existingTransaction); //应用flush模式
10. enableFilters(session);
12. //暴露给action的session
13. Session sessionToExpose = (exposeNativeSession? session: createSessionProxy(session));
14. //执行action
15. Object result = action.doInHibernate(sessionToExpose);
16. flushIfNecessary(session, existingTransaction);
18. return result;
20. } catch(HibernateException ex) {
21. throw convertHibernateAccessException(ex);
22. } catch(SQLException ex) {
23. throw convertJdbcAccessException(ex);
24. } catch(RuntimeException ex) {
25. throw ex;
26. } finally {
27. //如果session在事务中，则不关闭session
28. if(existingTransaction) {
29. disableFilters(session);
30. if(previousFlushMode != null) {
31. session.setFlushMode(previousFlushMode);
32. }
33. } else {
34. //释放session
35. SessionFactoryUtils.releaseSession(session, getSessionFactory());
36. }
37. }
38. }

 \*HibernateCallBack，一般用来实现特定的业务逻辑

 \*exposeNativeSession：是一个布尔值，要暴露给HibernateCallBack实际的session对象，而不是一个代理过的对象

6：一般情况下，只有HIberateTemplate提供的方法不能满足要求时才使用execute方法，它的使用情况如下所示，

****[html]**** [view plain](http://blog.csdn.net/woshisap/article/details/7026076" \o "view plain" \t "http://blog.csdn.net/zjkc050818/article/details/_blank) [copy](http://blog.csdn.net/woshisap/article/details/7026076" \o "copy" \t "http://blog.csdn.net/zjkc050818/article/details/_blank)

1. public void createDatabaseSchema() throws DataAccessException {
2. HibernateTemplate hibernateTemplate = new HibernateTemplate(this.sessionFactory);
3. //调用HibernateTempalte的execute方法
4. hibernateTemplate.execute(new HibernateCallback() {
5. public Object doInHibernate(Session session) throws HibernateException, SQLException {  //实现HibernateCallback的doInHibernate方法
6. //具体实现
7. Connection conn = session.connection();
8. final Dialect dialect = Dialect.getDialect(configuration.getProperties);
9. String[] sql = configuration.generateSchemaCreationScript(dialect);
10. executeSchemaScript(conn, sql);
11. }
12. });
13. }

  使用execute方法的重点是实现HibernateCallback的doInHibernate方法，它会传递一个Session实例，可以使用此Session实例操作数据库，由此看出execute方法的好处是应用程序不用关心session的创建和释放，只需要处理关心的业务逻辑即可。

## **二、HibernateTemplate常用方法**

### **1、find(String queryString);**

示例：hibernateTemplate.find("from bean.User");

返回所有User对象

### **2、find(String queryString , Object value);**

示例：hibernateTemplate.find("from bean.User u where u.name=?", "test");

或模糊查询：hibernateTemplate.find("from bean.User u where u.name like ?", "%test%");

返回name属性值为test的对象（模糊查询，返回name属性值包含test的对象）

### **3、find(String queryString, Object[] values);**

示例：String hql= "from bean.User u where u.name=? and u.password=?";

　　   hibernateTemplate.find(hql, new String[]{"test", "123"});

返回用户名为test并且密码为123的所有User对象

### **4、findByExample(Object exampleEntity)**

示例：User u=new  User();

　　   u.setPassword("123" );

　  u.setName("bb" );//必须 符合的条件但是这两个条件时并列的（象当于sql中的and)

     list=hibernateTemplate.findByExample(u);

返回：用户名为bb密码为123的对象

### **5、findByExample(Object exampleEntity, int firstResult, int  maxResults)**

示例：User u=new  User();

　　   u.setPassword("123" );

        u.setName("bb" );      //必须 符合的条件但是这两个条件时并列的（象当于sql中的and)

 　　  list=hibernateTemplate.findByExample(u,start,max);

返回：满足用户名为bb密码为123，自start起共max个User对象。（对象从0开始计数）

### **6、findByNamedParam(String queryString , String paramName , Object value)**

示例：String queryString = "select count(\*) from bean.User u where u.name=:myName" ;

　　　String paramName= "myName";

　　　String value= "abc";

　　　list=hibernateTemplate.findByNamedParam(queryString, paramName, value);

　　　System.out.println(list.get(0 ));

返回：name为abc的User对象的条数

### **7、 findByNamedParam(String queryString , String[] paramName , Object[] value)**

示例：String queryString = "select count(\*) from bean.User u where u.name=:myName and u.password=:myPassword";

　　　String[] paramName= new String[]{"myName", "myPassword"};

　　　String[] value= new String[]{"abc", "123"};

　　　hibernateTemplate.findByNamedParam(queryString, paramName, value);

返回：用户名为abc密码为123的User对象

### **8、findByNamedQuery(String queryName)**

示例：

1、首先需要在User.hbm.xml中定义命名查询

<hibernate-mapping>

<class>......</class>

<query name="queryAllUser ">   <!--此查询被调用的名字-->

<![CDATA[

from bean.User

]]>

</query>

</hibernate-mapping>

2、如下使用查询：

hibernateTemplate.findByNamedQuery("queryAllUser ");

### **9、findByNamedQuery(String queryName, Object value)**

示例：

1、首先需要在User.hbm.xml中定义命名查询

<hibernate-mapping>

<class>......</class>

<query name="queryByName "> <!--此查询被调用的名字-->

<![CDATA[

from bean.User u where u.name = ?

]]>

</query>

</hibernate-mapping>

2、如下使用查询：

hibernateTemplate.findByNamedQuery("queryByName ", "test");

### **10、findByNamedQuery(String queryName, Object[] value)**

 示例：

1、首先需要在User.hbm.xml中定义命名查询

<hibernate-mapping>

<class>......</class>

<query name="queryByNameAndPassword ">    <!--此查询被调用的名字-->

<![CDATA[

from bean.User u where u.name =? and u.password =?

]]>

</query>

</hibernate-mapping>

 2、如下使用查询：

String[] values= new String[]{"test", "123"};

hibernateTemplate.findByNamedQuery("queryByNameAndPassword " , values);

### **11、findByNamedQueryAndNamedParam(String queryName, String paramName, Object value)**

示例：

1、首先需要在User.hbm.xml中定义命名查询

<hibernate-mapping>

<class>......</class>

<query name="queryByName "><!--此查询被调用的名字-->

<![CDATA[

from bean.User u where u.name =:myName

]]>

</query>

</hibernate-mapping>

2、如下使用查询：

hibernateTemplate.findByNamedQuery("queryByName " , "myName", "test");

### **12、findByNamedQueryAndNamedParam(String queryName, String[] paramName, Object[] value)**

示例：

1、首先需要在User.hbm.xml中定义命名查询

<hibernate-mapping>

<class>......</class>

<query name="queryByNameAndPassword ">   <!--此查询被调用的名字-->

<![CDATA[

from bean.User u where u.name =:myName and u.password=:myPassword

]]>

</query>

</hibernate-mapping>

2、如下使用查询：

String[] names= new String[]{"myName", "myPassword"};

String[] values= new String[]{"test", "123"};

hibernateTemplate.findByNamedQuery("queryByNameAndPassword " , names, values);

### **13、findByValueBean(String queryString , Object value)**

示例：

1、定义一个ValueBean，属性名必须和HSQL语句中的：后面的变量名同名，此处必须至少有两个属性，分别为myName和 myPassword，使用setter方法设置属性值后

ValueBean valueBean= new ValueBean();

valueBean.setMyName("test");

valueBean.setMyPasswrod("123");

2、

String queryString= "from bean.User u where u.name=:myName and u.password=:myPassword";

hibernateTemplate.findByValueBean(queryString , valueBean);

### **14、findByNamedQueryAndValueBean(String queryName , Object value)**

示例：

1、首先需要在User.hbm.xml中定义命名查询

<hibernate-mapping>

<class>......</class>

<query name="queryByNameAndPassword ">   <!--此查询被调用的名字-->

<![CDATA[

from bean.User u where u.name =:myName and u.password=:myPassword

]]>

</query>

</hibernate-mapping>

2、定义一个ValueBean，属性名必须和User.hbm.xml命名查询语句中的：后面的变量名同名，此处必须至少有两个属性，分别为 myName和myPassword，使用setter方法设置属性值后

ValueBean valueBean= new ValueBean();

valueBean.setMyName("test");

valueBean.setMyPasswrod("123");

3、

String queryString= "from bean.User u where u.name=:myName and u.password=:myPassword";

hibernateTemplate.findByNamedQueryAndValueBean("queryByNameAndPassword ", valueBean);