

[您还未登录! 登录 注册](#)



[论坛首页](#) → [Java企业应用论坛](#) →

模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate (附源码)

[全部](#) [Hibernate](#) [Spring](#) [Struts](#) [iBATIS](#) [企业应用](#) [Lucene](#) [SOA](#) [Java综合](#) [Tomcat](#) [设计模式](#) [OO](#) [JBoss](#)

« 上一页 1 2 3 下一页 »

浏览 11216 次

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate \(附源码\)](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

发表时间：2010-07-15 最后修改：2010-07-20

相关推荐：

- jakoes
- 等级: 初级会员



- 性别:
- 文章: 25
- 积分: 10
- 来自: 上海
- 我现在离线

- [jdbc还是ibatis?](#)
- [JdbcTemplate学习笔记](#)
- [模板模式 模仿Spring写的 JdbcTemplate 不懂Spring 没关系 很实用](#)
- [模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate \(附源码\)](#)
- [JdbcTemplate 使用了模版模式和回调函数：](#)
- [从模板模式到JdbcTemplate](#)
- [自己动手写搜索引擎](#)
- [学习JdbcTemplate 时 用到回调函数](#)

推荐群组: [IBM WebSphere专区](#)

[更多相关推荐](#)

[Spring](#)

最近一直在研读spring源码和学习设计模式，想把自己的一些领悟与大家分享，前几天发了几篇简单的文章，可能由于文字过于简单，几次被评为新手贴，心中滴汗啊 😞 没办法，工作太忙，大家都知道，写篇文章是要很大精力地~~~~~

今天恰有时间，把这两天的学习所得与大家分享，尽量写得详细一些，专家饶路走，新手觉得好赞一下（不要拍砖哦~~~~）。
[文章源码在附件中](#)

注：本文目的意不在“重复发明轮子”，而是借此文来探讨Spring JdbcTemplate的内部实现原理，掌握其运用精妙之处，以便在以后的“造轮运动”中能灵活运用。

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

话回正转，这两天在读spring的jdbc模板，对Spring源码的精妙真是佩服得五体投地，极为经典。

spring中真是集设计模式之大成，而且用得是炉火纯青。模板方法（template method）就在spring中被大量使用，

如：jdbcTemplate,hibernateTemplate,JndiTemplate以及一些包围的包装等都无疑使用了模板模式，但spring并不是单纯使用了模板方法，而是在此基础上做了创新，配合callback（回调）一起使用，用得极其灵活。

OK，为了防止文章再被拍砖，我写得更详细点吧，我们首先来回顾一下模板模式：
所谓模板板式，就是在父类中定义算法的主要流程，而把一些个性化的步骤延迟到子类中去实现，父类始终控制着整个流程的主动权，子类只是辅助父类实现某些可定制的步骤。

有些抽象？？？

好吧，我们用代码来说话吧：

首先，父类要是个抽象类：

Java代码 ☆

```
1. public abstract class TemplatePattern {
2.
3.     //模板方法
4.     public final void templateMethod(){
5.
6.         method1();
7.         method2();//勾子方法
8.         method3();//抽象方法
9.     }
10.    private void method1(){
11.        System.out.println("父类实现业务逻辑");
12.    }
13.    public void method2(){
14.        System.out.println("父类默认实现，子类可覆盖");
15.    }
16.    protected abstract void method3();//子类负责实现业务逻辑
17. }
```

父类中有三个方法，分别是method1()，method2()和method3()。

method1()是私有方法，有且只能由父类实现逻辑，由于方法是private的，所以只能

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

父类调用。

method2()是所谓的**钩子方法**。父类提供默认实现，如果子类觉得有必要定制，则可以覆盖父类的默认实现。

method3()是子类必须实现的方法，即制定的步骤。

由此可看出，**算法的流程执行顺序是由父类掌控的，子类只能配合。**

下面我们来写第一个子类：

Java代码 ☆

```
1. public class TemplatePatternImpl extends TemplatePattern {
2.
3.     @Override
4.     protected void method3() {
5.         System.out.println("method3()在子类TemplatePatternImpl中实现
           了！！");
6.
7.     }
8.
9. }
```

这个子类只覆盖了必须覆盖的方法，我们来测试一下：

Java代码 ☆

```
1. TemplatePattern t1 = new TemplatePatternImpl();
2. t1.templateMethod();
```

在控制台中我们可以看到：

Java代码 ☆

1. 父类实现业务逻辑
2. 父类默认实现，子类可覆盖
3. method3()在子类TemplatePatternImpl中实现了！！

OK，我们来看看**钩子方法**的使用：

定义第2个子类，实现钩子方法：

Java代码 ☆

```
1. public class TemplatePatternImpl2 extends TemplatePattern {
2.
```

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

```
3.  @Override
4.  protected void method3() {
5.      System.out.println("method3()在子类TemplatePatternImpl2中实现
   了！！");
6.
7.  }
8.
9.  /* (non-Javadoc)
10.   * @see com.jak.pattern.template.example.TemplatePattern#method2()
11.   */
12.  @Override
13.  public void method2() {
14.      System.out.println("子类TemplatePatternImpl2覆盖了父类的method2()
   方法！！");
15.  }
16.
17. }
```

来测试一下：

Java代码 ☆

```
1. TemplatePattern t2 = new TemplatePatternImpl2();
2. t2.templateMethod();
```

我们看控制台：

Java代码 ☆

```
1. 父类实现业务逻辑
2. 子类TemplatePatternImpl2覆盖了父类的method2()方法！！
3. method3()在子类TemplatePatternImpl2中实现了！！
```

OK，经典的模板模式回顾完了（大家不要拍砖哦~~~~~）

接下来，我们回到正题，自己模仿spring动手写一个基于模板模式和回调的jdbcTemplate。

回顾一下，spring为什么要封装JDBC API，对外提供jdbcTemplate呢（不要仍鸡蛋啊

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate \(附源码\)](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

¥·%¥#%)

话说SUN的JDBC API也算是经典了，曾经在某个年代折服了一批人。但随着历史的发展，纯粹的JDBC API已经过于底层，而且不易控制，由开发人员直接接触JDBC API，会造成不可预知的风险。还有，数据连接缓存池的发展，也不可能让开发人员去手工获取JDBC了。

好了，我们来看一段曾经堪称经典的JDBC API代码吧：

Java代码 ☆

```
1. public List<User> query() {
2.
3.     List<User> userList = new ArrayList<User>();
4.     String sql = "select * from User";
5.
6.     Connection con = null;
7.     PreparedStatement pst = null;
8.     ResultSet rs = null;
9.     try {
10.         con = HsqldbUtil.getConnection();
11.         pst = con.prepareStatement(sql);
12.         rs = pst.executeQuery();
13.
14.         User user = null;
15.         while (rs.next()) {
16.
17.             user = new User();
18.             user.setId(rs.getInt("id"));
19.             user.setUserName(rs.getString("user_name"));
20.             user.setBirth(rs.getDate("birth"));
21.             user.setCreateDate(rs.getDate("create_date"));
22.             userList.add(user);
23.         }
24.
25.
26.     } catch (SQLException e) {
27.         e.printStackTrace();
28.     } finally{
29.         if(rs != null){
30.             try {
31.                 rs.close();
32.             } catch (SQLException e) {
```

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate \(附源码\)](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

```
33.         e.printStackTrace();
34.     }
35. }
36. try {
37.     pst.close();
38. } catch (SQLException e) {
39.     e.printStackTrace();
40. }
41. try {
42.     if(!con.isClosed()){
43.         try {
44.             con.close();
45.         } catch (SQLException e) {
46.             e.printStackTrace();
47.         }
48.     }
49. } catch (SQLException e) {
50.     e.printStackTrace();
51. }
52.
53. }
54. return userList;
55. }
```

上面的代码要若干年前可能是一段十分经典的，还可能被作为example被推广。但时过境迁，倘若哪位程序员现在再在自己的程序中出现以上代码，不是说明该公司的开发框架管理混乱，就说明这位程序员水平太“高”了。

我们试想，一个简单的查询，就要做这么一大堆事情，而且还要处理异常，我们不防来梳理一下：

- 1、获取connection
- 2、获取statement
- 3、获取resultset
- 4、遍历resultset并封装成集合
- 5、依次关闭connection,statement,resultset，而且还要考虑各种异常
- 6、.....

啊~~~~ 我快要晕了，在面向对象编程的年代里，这样的代码简直不能上人容忍。试想，上面我们只是做了一张表的查询，如果我们要做第2张表，第3张表呢，又是一堆重复的代码：

- 1、获取connection

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

- 2、获取statement
- 3、获取resultset
- 4、遍历resultset并封装成集合
- 5、依次关闭connection,statement,resultset，而且还要考虑各种异常
- 6、.....

这时候，使用模板模式的时机到了！！！！

通过观察我们发现上面步骤中大多数都是重复的，可复用的，只有在遍历ResultSet并封装成集合的这一步骤是可定制的，因为每张表都映射不同的java bean。这部分代码是没有办法复用的，只能定制。那就让我们用一个抽象的父类把它们封装一下吧：

Java代码 ☆

```
1. public abstract class JdbcTemplate {
2.
3.     //template method
4.     public final Object execute(String sql) throws SQLException{
5.
6.         Connection con = HsqldbUtil.getConnection();
7.         Statement stmt = null;
8.         try {
9.
10.             stmt = con.createStatement();
11.             ResultSet rs = stmt.executeQuery(sql);
12.             Object result = doInStatement(rs);//abstract method
13.             return result;
14.         }
15.         catch (SQLException ex) {
16.             ex.printStackTrace();
17.             throw ex;
18.         }
19.         finally {
20.
21.             try {
22.                 stmt.close();
23.             } catch (SQLException e) {
24.                 e.printStackTrace();
25.             }
26.             try {
27.                 if(!con.isClosed()){
```

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

```
28.         try {
29.             con.close();
30.         } catch (SQLException e) {
31.             e.printStackTrace();
32.         }
33.     }
34. } catch (SQLException e) {
35.     e.printStackTrace();
36. }
37.
38. }
39. }
40.
41. //implements in subclass
42. protected abstract Object doInStatement(ResultSet rs);
43. }
```

在上面这个抽象类中，封装了SUN JDBC API的主要流程，而遍历ResultSet这一步骤则放到抽象方法doInStatement()中，由子类负责实现。

好，我们来定义一个子类，并继承上面的父类：

Java代码 ☆

```
1. public class JdbcTemplateUserImpl extends JdbcTemplate {
2.
3.     @Override
4.     protected Object doInStatement(ResultSet rs) {
5.         List<User> userList = new ArrayList<User>();
6.
7.         try {
8.             User user = null;
9.             while (rs.next()) {
10.
11.                 user = new User();
12.                 user.setId(rs.getInt("id"));
13.                 user.setUserName(rs.getString("user_name"));
14.                 user.setBirth(rs.getDate("birth"));
15.                 user.setCreateDate(rs.getDate("create_date"));
16.                 userList.add(user);
17.             }
18.             return userList;
19.         } catch (SQLException e) {
20.             e.printStackTrace();
21.         }
22.     }
23. }
```


锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

```
19.     } catch (SQLException e) {  
20.         e.printStackTrace();  
21.         return null;  
22.     }  
23. }  
24.  
25. }
```

由代码可见，我们在doInStatement()方法中，对ResultSet进行了遍历，最后并返回。

有人可能要问：我如何获取ResultSet 并传给doInStatement()方法啊？？呵呵，问这个问题的大多是新手。因为此方法不是由子类调用的，而是由父类调用，并把ResultSet传递给子类的。我们来看一下测试代码：

Java代码 ☆

```
1. String sql = "select * from User";  
2. JdbcTemplate jt = new JdbcTemplateUserImpl();  
3. List<User> userList = (List<User>) jt.execute(sql);
```

就是这么简单！！

文章至此仿佛告一段落，莫急！不防让我们更深入一些...

试想，如果我每次用jdbcTemplate时，都要继承一下上面的父类，是不是有些不方面呢？

那就让我们甩掉abstract这顶帽子吧，这时，就该callback（回调）上场了

所谓回调，就是方法参数中传递一个接口，父类在调用此方法时，必须调用方法中传递的接口的实现类。

那我们就来把上面的代码改造一下，改用回调实现吧：

首先，我们来定义一个回调接口：

Java代码 ☆

```
1. public interface StatementCallback {  
2.     Object doInStatement(Statement stmt) throws SQLException;  
3. }
```

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

这时候，我们就要方法的签名改一下了：

Java代码 ☆

```
1. private final Object execute(StatementCallback action) throws SQLException
```

里面的获取数据方式也要做如下修改：

Java代码 ☆

```
1. Object result = action.doInStatement(stmt); // abstract method
```

为了看着顺眼，我们来给他封装一层吧：

Java代码 ☆

```
1. public Object query(StatementCallback stmt) throws SQLException {  
2.     return execute(stmt);  
3. }
```

OK，大功告成！

我们来写一个测试类Test.java测试一下吧：

这时候，访问有两种方式，一种是[内部类](#)的方式，一种是[匿名方式](#)。

先来看看内部类的方式：

Java代码 ☆

```
1. //内部类方式  
2. public Object query(final String sql) throws SQLException {  
3.     class QueryStatementCallback implements StatementCallback {  
4.  
5.         public Object doInStatement(Statement stmt) throws SQLException {  
6.             ResultSet rs = stmt.executeQuery(sql);  
7.             List<User> userList = new ArrayList<User>();  
8.  
9.             User user = null;  
10.            while (rs.next()) {  
11.
```

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

```
12.         user = new User();
13.         user.setId(rs.getInt("id"));
14.         user.setUserName(rs.getString("user_name"));
15.         user.setBirth(rs.getDate("birth"));
16.         user.setCreateDate(rs.getDate("create_date"));
17.         userList.add(user);
18.     }
19.     return userList;
20.
21. }
22.
23. }
24.
25. JdbcTemplate jt = new JdbcTemplate();
26. return jt.query(new QueryStatementCallback());
27. }
```

在调用jdbcTemplate.query()方法时，传一个StatementCallBack()的实例过去，也就是我们的内部类。

再来看看匿名方式：

Java代码 ☆

```
1. //匿名类方式
2. public Object query2(final String sql) throws Exception{
3.
4.     JdbcTemplate jt = new JdbcTemplate();
5.     return jt.query(new StatementCallback() {
6.
7.         public Object doInStatement(Statement stmt) throws SQLException {
8.             ResultSet rs = stmt.executeQuery(sql);
9.             List<User> userList = new ArrayList<User>();
10.
11.             User user = null;
12.             while (rs.next()) {
13.
14.                 user = new User();
15.                 user.setId(rs.getInt("id"));
16.                 user.setUserName(rs.getString("user_name"));
```

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

```
17.         user.setBirth(rs.getDate("birth"));
18.         user.setCreateDate(rs.getDate("create_date"));
19.         userList.add(user);
20.     }
21.     return userList;
22.
23. }
24. });
25.
26. }
```

相比之下，这种方法更为简洁。

为什么spring不用传统的模板方法，而加之以Callback进行配合呢？

试想，如果父类中有10个抽象方法，而继承它的所有子类则要将这10个抽象方法全部实现，子类显得非常臃肿。而有时候某个子类只需要定制父类中的某一个方法该怎么办呢？这个时候就要用到Callback回调了。

离spring jdbcTemplate再近一点

上面这种方式基本上实现了模板方法+回调模式。但离spring的jdbcTemplate还有些距离。

我们可以再深入一些。。。

我们上面虽然实现了模板方法+回调模式，但相对于Spring的JdbcTemplate则显得有些“丑陋”。Spring引入了RowMapper和ResultSetExtractor的概念。

RowMapper接口负责处理某一行的数据，例如，我们可以在mapRow方法里对某一行记录进行操作，或封装成entity。

ResultSetExtractor是数据集抽取器，负责遍历ResultSet并根据RowMapper里的规则对数据进行处理。

RowMapper和ResultSetExtractor区别是，RowMapper是处理某一行数据，返回一个实体对象。而ResultSetExtractor是处理一个数据集，返回一个对象集合。

当然，上面所述仅仅是Spring JdbcTemplate实现的基本原理，Spring JdbcTemplate内部还做了更多的事情，比如，把所有的基本操作都封装到JdbcOperations接口内，以及采用JdbcAccessor来管理DataSource和转换异常等。

接下来的主题：[进行范型的改造](#)

我们可能发现，上面的接口中返回的还都是Object对象，在范型日益盛行的今天，我们怎能忍受每次都要把得到的对象进行强制类型转换？那我们就要心情享用JDK1.5给我们带来的范型盛宴。

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

[未完待续...](#)

- [SpringTemplateCallbackTemplate.rar](#) (701 KB)
- 下载次数: 380

声明：ITeye文章版权属于作者，受法律保护。没有作者书面许可不得转载。

[推荐链接](#)

[返回顶楼](#)

- jakoes
- 等级: 初级会员



发表时间：2010-07-20

今天一个朋友问我模板方法和工厂模式有什么区别？

的确，模板方法和工厂模式有相似地方，就是都把某些实现下放到了子类中。

但它们的关注点不一样，

工厂模式是属于创建模式中的一种，关注于“对象的创建”。

而模板方法是父类对“算法”的封装，只是把某些个性化的、需要定制的算法逻辑延迟到子类中实现，关注于“算法逻辑”。

- 性别:
- 文章: 25
- 积分: 10
- 来自: 上海
- 我现在离线

[返回顶楼](#)

----- [回帖地址](#)

[0 0](#) 请登录投票

- phz50
- 等级:



发表时间：2010-07-20

一直都觉得spring-jdbc设计得非常优秀

- 性别:
- 文章: 107
- 积分: 115
- 我现在离线

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate \(附源码\)](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

[返回顶楼](#)[-----回贴地址](#)[0 0](#) 请登录后投票

- finallygo
- 等级: 初级会员



- 性别:
- 文章: 393
- 积分: 40
- 来自: 厦门-->北京
- 我现在离线

发表时间：2010-07-20 最后修改：2010-07-20

我感觉有点问题吧，首先jdbctemplate好像没有与你这里的HsqldbUtil这种类耦合在一起吧，还有一个就是，你这里好像进行了一次查询之后数据库连接就关闭了，那我怎么做事务呢？

[返回顶楼](#)[-----回贴地址](#)[0 0](#) 请登录后投票

发表时间：2010-07-21

finallygo 写道

我感觉有点问题吧，首先jdbctemplate好像没有与你这里的HsqldbUtil这种类耦合在一起吧，还有一个就是，你这里好像进行了一次查询之后数据库连接就关闭了，那我怎么做事务呢？

- jakoes
- 等级: 初级会员



- 性别:
- 文章: 25
- 积分: 10
- 来自: 上海
- 我现在离线

楼上说得不错，本文关注点在于模板方法和回调的应用，其它细节没有体现在示例中。

本文的示例只能说是Spring JdbcTemplate中模板方法和回调的一个快照，离真正的Spring Template还有很大距离，Spring Template考虑得更细，比如异常转换，另外Spring会把Connection和Statement通过nativeJdbcExtractor转换成优化过的实例。文章的题目也是“模板方法和Callback回调应用实践”，在这里只讨论设计模式，并没有考虑其它的细节，如事物等。只是供学习的demo，楼主这样提问，未免有鸡蛋里挑骨头之嫌^^

至于HsqldbUtil，是我对内存数据库Hsqldb的一个简单的封装，只是为了让各位网友在下到代码后能在不安装数据库的情况下就运行程序，当然，这代码只能供学习的DEMO用，距离企业应用还是有一段距离的。

通过Spring的源码，我们知道Spring是按如下方式来获取数据连接的：

Java代码

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate（附源码）](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

```
1. Connection con = DataSourceUtils.getConnection(getDataSource());
```

而本文只是简单模仿这种方式：

Java代码 ☆

```
1. Connection con = HsqlDbUtil.getConnection();
```

不知楼上的“耦合”是指哪里？

最后非常感谢楼上对本文的关注，文章的其它纰漏之处，还请不吝指出~😊

[返回顶楼](#)[-----回贴地址](#)[0 0](#) 请登录后投票

- finallygo
- 等级: 初级会员



- 性别: ♂
- 文章: 393
- 积分: 40
- 来自: 厦门-->北京

- 我现在离线

发表时间：2010-07-21 最后修改：2010-07-21

我的耦合的意思是你不要去引用HsqlDbUtil这样的类,我的意见是设置一个类型为java.sql.Connection的属性,并提供它的set方法,这样就不会与你那个类耦合了吗?而且也提供了灵活性.因为获取连接方式是很多种的,我觉得你写在HsqlDbUtil里是不太好的,你可以看spring里的虽然看起来差不多,但是它提供了setDataSource的方法吧?我之所以会考虑事务的方面是因为我也写过自己的JdbcTemplate,但是事务处理方面却不理想,所以想看看你是怎么考虑的.而且你这样还有个不好的地方就是你每次执行查询都要重新获得一次连接吧,这样是不是太消耗资源了?

当然还是要感谢你分享你的知识.

[返回顶楼](#)[-----回贴地址](#)[0 0](#) 请登录后投票

- jakoes
- 等级: 初级会员



发表时间：2010-07-22

finallygo 写道

我的耦合的意思是你不要去引用HsqlDbUtil这样的类,我的意见是设置一个类型为java.sql.Connection的属性,并提供它的set方法,这样就不会与你那个类耦合了吗?而且也提供了灵活性.因为获取连接方式是很多种的,我觉得你写在HsqlDbUtil里是不太好的,你可以看spring里的虽然看起来差不多,但是它提供了setDataSource的方法吧?我之所以会考虑事务的方面是因为我也写过自己的JdbcTemplate,但是事务处理方面却不理想,所以想看看你是怎么考虑的.而且你这样还有个不好的地方就是你每次执行



锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate \(附源码\)](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文

查询都要重新获得一次连接吧,这样是不是太消耗资源了?
当然还是要感谢你分享你的知识.

- 性别: 
- 文章: 25
- 积分: 10
- 来自: 上海
- 

你所提到的问题的确都是在开发过程中常遇到的。如果要把我的代码应用到企业应用中,数据链接确实要通过外部注入的。

另外一点要说明的是,如果要把上面示例运用到企业应用中,这不可能是一个“全功能的轮子”,像事物等功能肯定不是全部自己动手去写,也没有必要“重复发明轮子”。其实完全交给系统框架去管理,如Spring来管理事物等。


上面的示例意在说明模板方法的原理与使用,以后如果遇到类似场景,而没有找到“合适的轮子”的时候,就可以自己“造轮子”啦。

再次感谢楼上对本文的关注。希望有问题大家一起讨论

[返回顶楼](#)[----- 回帖地址](#)[0 0](#) 请登录投票

- finallygo
- 等级: 初级会员



- 性别: 
- 文章: 393
- 积分: 40
- 来自: 厦门-->北京

- 

发表时间: 2010-07-22 最后修改: 2010-07-22


说到spring,我有点小小的疑问,

我先说下我对spring的认识啊,spring中最主要的两个思想是ioc,aop

ioc就体现在工厂模式的应用,也就是说我们采用的都是面向接口的方式的编程,以后如果实现类发生了变化了,可以通过修改配置文件而不用修改代码来实现系统的灵活性,但是实际上好像这种需求并不多啊,我们项目中都是一个接口,一个实现类,也就是说配置文件我们添加一个Dao的时候才去改配置文件,之后就再也不去动了,反而接口是经常变动的,因为需求是一直在增加的,所以给我的感觉就是接口成为了一个累赘,我改一个接口就要改一堆的东西(因为我们分了好几层,而一个方法的添加就从dao一直污染到顶层了),我觉得还不如直接创建一个实现类来的方便.

还有一个就是aop,我觉得主要就是用来做事务和日志的,但是这些用动态代理就可以实现了,我觉得spring太复杂了,没有必要用的
不知道我这两个思想有什么问题,麻烦你讲解一下.

[返回顶楼](#)[----- 回帖地址](#)[0 0](#) 请登录投票

- ironsabre
- 等级: 

发表时间: 2010-07-23

finallygo 写道

我感觉有点问题吧,首先jdbcTemplate好像没有与你这里的HsqlDbUtil这种耦合在一起吧,还有一个就是,你这里好像进行了一次查询之后数据库连接就关闭了,那我怎么做事务呢?

锁定老帖子 [主题：模板方法和Callback回调应用实践 - 自己动手写JdbcTemplate \(附源码\)](#)

精华帖 (0) :: 良好帖 (6) :: 新手帖 (14) :: 隐藏帖 (0)

作者

正文



- 文章: 581
- 积分: 142
- 我现在离线

你难道还在用连接相关的事务吗？

[返回顶楼](#)[回贴地址](#)[00](#) 请登录后投票

- blackchoc
- 等级: 初级会员



- 性别:
- 文章: 33
- 积分: 40
- 来自: 北京
- 我现在离线

发表时间：2010-09-04

感谢楼主分享。

现在我在试着重构手头的一个项目，不幸的是这个项目已经经历了n多人，从最初到现在已经很久了。所用的技术就是楼主传说中“高手”所用的sun jdbc api。哈哈

[返回顶楼](#)[回贴地址](#)[00](#) 请登录后投票« 上一页 1 [2](#) [3](#) 下一页 »[论坛首页](#) → [Java企业应用版](#)

跳转论坛:

Java企业应用 ▼

- [首页](#)
- [资讯](#)
- [精华](#)
- [论坛](#)
- [问答](#)
- [博客](#)
- [专栏](#)
- [群组](#)

- [搜索](#)
- [广告服务](#)
- [ITeye黑板报](#)
- [联系我们](#)
- [友情链接](#)

© 2003-2018 ITeye.com. [[京ICP证070598号](#) 京公网安备11010502027441]
北京创新乐知信息技术有限公司 版权所有