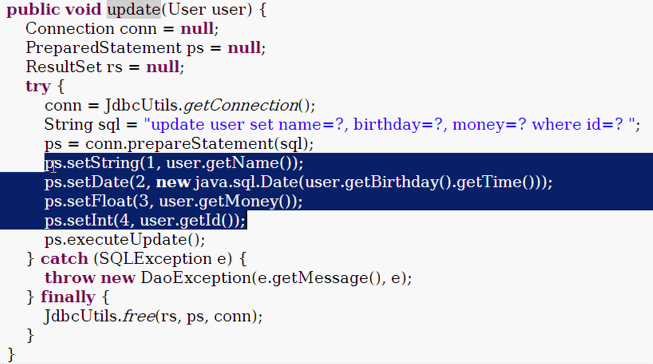
代码的优化：将Update方法提取到抽象类中

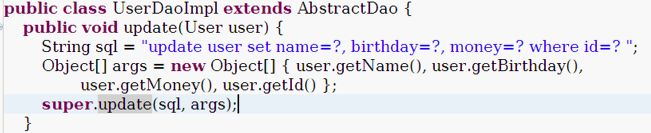
---**模板处理方法Template**

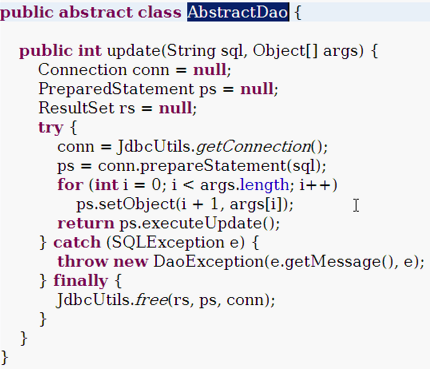
1. 代码：

原来的代码：



优化后的代码：





1. 分析：

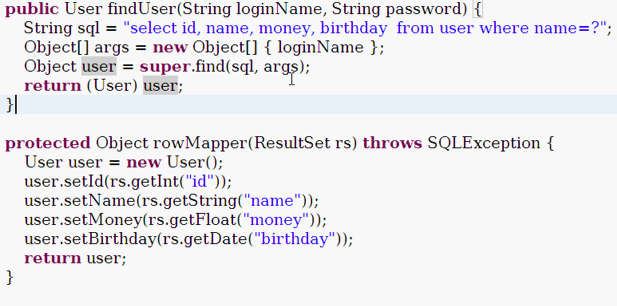
将setXxx统一利用setObject即可，降低了代码的冗余度；由于update语句的内容参数不同，所以将不同的东西提取成参数，作为方法的参数传入，这样就提高了该方法的利用率。

优点：

1. 可以减少代码出错的几率；
2. 可以减少代码的冗余度；
3. 只需传入相应参数即可，且代码的利用率较高，不仅可以用于Update，也可以用于delete、insert等操作。
4. 优化查询方法：把共性提取出来，sql语句及参数值作为方法的参数传入方法中，而对于结果集的处理，抽象为抽象方法，让子类去实现。



**子类的实现**：



1. 示例：

**抽象类AbstractDao**:

**public** **abstract** **class** AbstractDao {

**public** Object find(String sql, Object[] orgs) {

Connection connection = **null**;

Object obj = **new** Object();

PreparedStatement psm = **null**;

ResultSet rs = **null**;

**try** {

connection = DriverUtil.*getConnection*();

psm = connection.prepareStatement(sql);

**for** (**int** i = 1; i <= orgs.length; i++) {

psm.setObject(i, orgs[i-1]);

}

System.***out***.println("&&&&&&&&&&&&&&&");

rs = psm.executeQuery();

**while**(rs.next()) {

obj = mappingRS(rs);

}

} **catch** (SQLException e) {

System.***err***.println("出现了异常！");

}

**finally** {

DriverUtil.*close*(rs,psm,connection);

}

**return** obj;

}

**abstract** **public** Object mappingRS(ResultSet rs) **throws** SQLException;

}

**实现类TemplateDaoImple**：

**public** **class** TemplateDaoImple **extends** AbstractDao {

**public** **void** findImpl(**int** id) {

String sql = "select id,name,birthday from users where id = ?";

Object[] orgs = **new** Object[] {id};

User user =(User)**super**.find(sql,orgs);

System.***out***.println(user.getId()+"\t"+user.getName()+"\t"+user.getBirthday());

}

**public** **User** **mappingRS**(ResultSet rs) **throws** SQLException {

User user = **new** User();

user.setId(rs.getInt(1));

user.setName(rs.getString(2));

user.setBirthday(**new** Date(rs.getDate(3).getTime()));

**return** user;

} }

User类：

**public** **class** User {

**private** String name;

**private** **int** id;

**private** Date birthday;

**其他的是构造方法和getter、setter方法。**

}

Test类：中的main方法。

**public** **static** **void** main(String[] args) {

TemplateDaoImple tt = **new** TemplateDaoImple();

tt.findImpl(22);

}

操作结果：



1. **提出新的问题**：由于对数据的查询，存在很多类型：有的只查询一个数据，有的查询多个数据，有的查询全部数据；有的只查询一列，有的查询多列。

**要查询到不同的数据，只需要对RowMap接口进行不同的实现即可。**

怎么办？**显然，返回的Object就不同了。具体查询什么，可以书写相应的SQL语句，但是对应的返回类型就不一样了，怎么把一个处理结果集Resultset的方法作为参数传入find方法中呢？**

**当然，在java中，方法是不可以作为一个参数传入方法中的，但是可以传入对象。处理办法：创建一个接口，用来规定返回的数据是什么的规则，通过创建不同的实现类对象作为参数传入find方法，即可返回正确的查询结果。**

1. **创建结果处理接口和共同的find方法**：

**接口**：RowMapping：

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**public** **interface** RowMapping {

**public** Object mapRow(ResultSet rs) **throws** SQLException;

}

**MyDao2类：**

**public** **class** MyDao2 {

**public Object find(String sql, Object[] orgs,RowMapping rowMapping)** {//有三个参数

Connection connection = **null**;

Object obj = **new** Object();

PreparedStatement psm = **null**;

ResultSet rs = **null**;

**try** {

connection = DriverUtil.*getConnection*();

psm = connection.prepareStatement(sql);

**for** (**int** i = 1; i <= orgs.length; i++) {

psm.setObject(i, orgs[i-1]);

}

System.***out***.println("&&&&&&&&&&&&&&&");

rs = psm.executeQuery();

**obj = rowMapping.mapRow(rs);//处理结果集rs**

} **catch** (SQLException e) {

System.***err***.println("出现了异常！");

}

**finally** {

DriverUtil.*close*(rs,psm,connection);

}

**return** obj;

}

}

**对于具体的RowMapping的实现类采用匿名内部类的方式。**

1. **获取多个行的name，存到集合中并返回：**

**public** **static** **void** main(String[] args) {

String sql = "select name from users where id < ?";

Object[] orgs = **new** Object[] { 4 };

RowMapping rowMapping = **new** RowMapping() {

@Override

**public** List<String> mapRow(ResultSet rs) **throws** SQLException {

List<String> list = **new** ArrayList<String>();

**while** (rs.next()) {

list.add(rs.getString("name"));

}

**return** list;

}

};

MyDao2 myDao2 = **new** MyDao2();

Object list = myDao2.find(sql, orgs, rowMapping);

System.***out***.println(list);

}

结果：

1. **获取单个行，并封装成User对象返回。**

String sql = "select id,name,birthday from users where id = ?";

Object[] orgs = **new** Object[] { 4 };

RowMapping rowMapping = **new** RowMapping() {

@Override

**public** User mapRow(ResultSet rs) **throws** SQLException {

User user = **null**;

**if** (rs.next())

user = **new** User(rs.getInt("id"),rs.getString("name"),rs.getDate("birthday"));

**return** user;

}

};

MyDao2 myDao2 = **new** MyDao2();

User user = (User)myDao2.find(sql, orgs, rowMapping);

System.***out***.println(user.getId()+"\t"+user.getName()+"\t"+user.getBirthday());

1. **获取多个行的数据，并将每一行封装成一个User对象，最后返回List<User>集合。**

**public** **static** **void** main(String[] args) {

String sql = "select id,name,birthday from users where id < ? ";

Object[] orgs = **new** Object[] { 4 };

RowMapping rowMapping = **new** RowMapping() {

@Override

**public** List<User> mapRow(ResultSet rs) **throws** SQLException {

List<User> list = **new** ArrayList<User>();

**while** (rs.next()) {

list.add(**new** User(rs.getInt("id"), rs.getString("name"), rs.getDate("birthday")));

}

**return** list; }};

MyDao2 myDao2 = **new** MyDao2();

List<User> list = (List)myDao2.find(sql, orgs, rowMapping);

**for** (Iterator<User> iterator = list.listIterator(); iterator.hasNext();) {

User user = iterator.next();

System.***out***.println(user.getId()+"\t"+user.getName()+"\t"+user.getBirthday()); }

结果：



1. 从上面的示例中，应该得到一定的结果和体会：

要善于使用接口的功能，完成一些问题。完成什么问题呢？

如果存在共性的东西，但是存在很多变数，可以提取到接口中，通过是对此接口实现出不同的对象，实现一定的功能。**特别是，通过匿名内部类的方式可以简写代码。**

1. 完整的工程：

JbdcUtils：

**public** **final** **class** JdbcUtils {

**private** **static** DataSource *dataSource* = **null**;

**static** {

**try** {

Class.*forName*("com.mysql.jdbc.Driver");

Properties properties = **new** Properties();

InputStream in = **new** FileInputStream(**new** File("dbcpconfig.properties"));

// InputStream in = JdbcUtils.class.getClassLoader().getResourceAsStream("dbcpconfig.properties");

properties.load(in);

/\*

\* properties.setProperty("driverClassName", "com.mysql.jdbc.Driver");

\* properties.setProperty("url","jdbc:mysql://localhost:3306/new\_schema");

\* properties.setProperty("username", "root");

\* properties.setProperty("password", "1234567890");

\*/

*dataSource* = BasicDataSourceFactory.*createDataSource*(properties);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** Connection getConnection() **throws** SQLException {

Connection connection = *dataSource*.getConnection();

**return** connection;

}

**public** **static** DataSource getDataSource() {

**return** *dataSource*;

}

**public** **static** **void** setDataSource(DataSource dataSource) {

JdbcUtils.*dataSource* = dataSource;

}

**public** **static** **void** close(Connection conn) {

**if** (conn != **null**) {

**try** {

conn.close();

} **catch** (SQLException e) {

}

}

}

**public** **static** **void** close(PreparedStatement psm) {

**if** (psm != **null**) {

**try** {

psm.close();

} **catch** (SQLException e) {

}

}

}

/\*\*

\* 关闭结果集ResultSet

\*/

**public** **static** **void** close(ResultSet rs) {

**if** (rs != **null**) {

**try** {

rs.close();

} **catch** (SQLException e) {

}

}

}

**public** **static** **void** close(ResultSet rs, PreparedStatement psm,

Connection conn) {

**try** {

**if** (rs != **null**) {

rs.close();

}

} **catch** (SQLException e) {

} **finally** {

**try** {

psm.close();

} **catch** (SQLException e) {

} **finally** {

**try** {

conn.close();

} **catch** (SQLException e) {

}

}

}

}

**public** **static** **void** close(PreparedStatement psm, Connection conn) {

**try** {

psm.close();

} **catch** (SQLException e) {

} **finally** {

**try** {

conn.close();

} **catch** (SQLException e) {

} **finally** {

}

}

}

}

AbstractDao：

**public** **class** AbstractDao {

**public** **void** update(String sql,Object[] orgs) **throws** Exception {

Connection conn = JdbcUtils.*getConnection*();

PreparedStatement psm = conn.prepareStatement(sql);

**for** (**int** i = 0; i < orgs.length; i++) {

psm.setObject(i+1, orgs[i]);

}

psm.executeUpdate();

JdbcUtils.*close*(psm,conn);

}

**public** Object query(String sql,Object[] orgs,RowMap rm) **throws** SQLException {

Connection conn = JdbcUtils.*getConnection*();

PreparedStatement psm = conn.prepareStatement(sql);

**for** (**int** i = 0; i < orgs.length; i++) {

psm.setObject(i+1, orgs[i]);

}

ResultSet rs = psm.executeQuery();

Object obj = rm.rowMap(rs);

**return** obj;

} }

接口：RowMap

**public** **interface** RowMap {

**public** Object rowMap(ResultSet rs) **throws** SQLException;

}

UserDao：

**public** **class** UserDaoImpl **extends** AbstractDao {

**public** **void** update(User user) **throws** Exception {

String sql = "update users set name = ?,birthday = ? where id = ?";

Object[] orgs = **new** Object[] { user.getName(), user.getBirthday(),

user.getId() };

**super**.update(sql, orgs);

}

**public** **void** delete(User user) **throws** Exception {

String sql = "delete from users where id = ?";

Object[] orgs = **new** Object[] { user.getId() };

**super**.update(sql, orgs);

}

**public** **void** insert(User user) **throws** Exception {

String sql = "insert into users values(?,?,?)";

Object[] orgs = **new** Object[] { user.getId(),

user.getName(), user.getBirthday()};

**super**.update(sql, orgs);

}

**public** Object query(**int** id) **throws** SQLException {

String sql = "select name,id,birthday from users where id < ?";

Object[] orgs = **new** Object[] { id };

**return** **super**.query(sql, orgs, **new** RowMap() {

@Override

**public** Object rowMap(ResultSet rs) **throws** SQLException {

User user = **null**;

List<User> list = **new** LinkedList<User>();

**while** (rs.next()) {

user = **new** User(rs.getInt("id"), rs.getString("name"),

rs.getDate("birthday"));

list.add(user);

}

**return** list;

}

}); } }

Test类：

**public** **static** **void** main(String[] args) **throws** Exception {

User user = **new** User();

user.setId(3);

user.setBirthday(**new** Date());

user.setName("亚历山大大帝");

UserDaoImpl udi = **new** UserDaoImpl();

// udi.update(user);

// udi.delete(user);

udi.insert(user);

List<User> list = (List<User>) udi.query(5);

**for** (Iterator<User> it = list.iterator(); it.hasNext();) {

User user2 = it.next();

System.***out***.println(user2.getId()+"\t" + user2.getName() + "\t"+ user2.getBirthday());

} }