使用JdbcTemplate代替JDBCAPI

1. **（1）Spring框架**提供了一个**工具类JdbcTemplate**，该工具类对JDBCAPI进行了很好地封装，这个类就像之前我们自己对JDBC API进行封装一样，只是代码更健壮、更强大而已。

**（2）以后在实际项目中，可以直接使用JdbcTemplate类来完全替代直接使用JDBC API，操作JdbcTemplate与直接使用JDBC API没有太大的性能区别。**

**（3）欲使用JdbcTemplate类，需要额外从spring开发包中导入spring.jar和commons-loggin.jar包。**

1. JdbcTemplate模板是**线程安全**的。
2. 使用JdbcTemplate：创建JdbcTemplate需要一个数据源DataSource：

两种方法：

1. 使用带参数的构造方法：

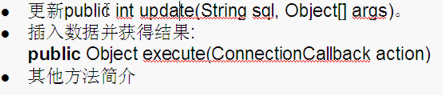
JdbcTemplate jdbcTemplate = **new** JdbcTemplate(JdbcUtils.*getDataSource*());

1. 利用**set方法**：

JdbcTemplate jdbcT = new JdbcTemplate();

jdbcT.setDataSource(JdbcUtils.getDataSource());





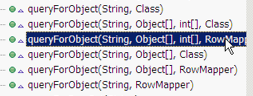
1. JdbcTemplate类的中方法：
2. **queryForInt（String）**：可以查询数据库中数据的数目，返回int值。

**queryForInt（String，Object[]）**：加入参数。

**queryForInt（String，Object[],int[]）**：int[]用来指明参数的类型。

1. queryForObject（String sql）：只可以查询一个数据。

queryForObject（String sql，**返回类型.class**）：只可以查询一个数据。



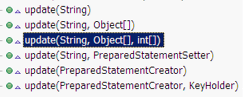
如：

查询的就是String类型。

1. queryForMap（）：获取一个Map对象。

如

1. **query（）**：可以查询多个数据，返回list集合。
2. update方法：



1. **查询数据库中数据的数目**：

语句是**：String sql = "select count(\*) from users";**

**public** **int** getUserCount() {

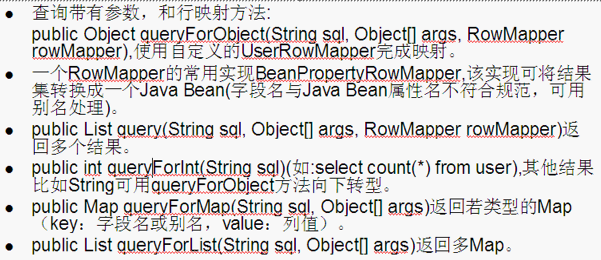
JdbcTemplate jdbcTemplate = **new** JdbcTemplate(JdbcUtils.*getDataSource*());

String sql = "select count(\*) from users";

**int** num = jdbcTemplate.queryForInt(sql);

**return** num;

}



1. 注意：JdbcTemplate这个类的queryForObject只可以返回一个结果，不可以返回集合，否则会报异常。查询多个结果需要使用query方法，返回一个list集合。
2. ConnectionCallback：



**new org.springframework.jdbc.core.ConnectionCallback<T>() {**

**@Override**

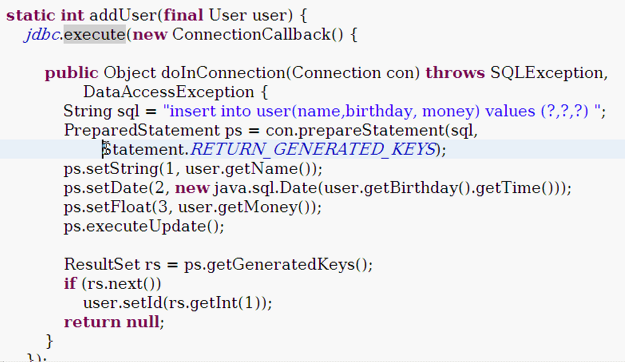
**public T doInConnection(Connection con) throws SQLException,**

**DataAccessException {**

**return null;**

**} };**

通过实现**ConnectionCallback**可以获取**主键GeneratedKeys**。



1. 增删改查操作：
2. 增添：

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

JdbcTemplate jdbcTemplate = **new** JdbcTemplate(JdbcUtils.*getDataSource*());

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

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

user.getBirthday() };

jdbcTemplate.update(sql, args); }

1. 删除：

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

JdbcTemplate jdbc = **new** JdbcTemplate(JdbcUtils.*getDataSource*());

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

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

jdbc.update(sql, args);

}

1. **修改**：

**public** **int** update(User user) {

JdbcTemplate jdbcTemplate = **new** JdbcTemplate(JdbcUtils.*getDataSource*());

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

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

user.getId() };

**int** num = jdbcTemplate.update(sql, args);

**return** num; }

1. 查询：

**public** User findUser(String name) {

JdbcTemplate jdbcTemplate = **new** JdbcTemplate(JdbcUtils.*getDataSource*());

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

Object[] args = **new** Object[] { name };

Object obj = jdbcTemplate.queryForObject(sql, args,

new BeanPropertyRowMapper(User.class));

/\* Object obj = jdbcTemplate.queryForObject(sql, args, new

RowMapper<User>() {

public User mapRow(ResultSet rs, int rowNum) throws SQLException {

User user = new User(rs.getInt(1),rs.getString(2),rs.getDate(3));

return user;

};});\*/

**return** (User) obj;

**}**

**public** List<User> findMultiUsers(**int** maxId) {

JdbcTemplate jdbc = **new** JdbcTemplate(JdbcUtils.*getDataSource*());

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

Object[] args = **new** Object[] { maxId };

**List<User> list = jdbc.query(sql, args, new BeanPropertyRowMapper(User.class));//BeanPropertyRowMapper会自动获取user对象并把user放入list集合中。**

// Object obj = jdbc.queryForObject(sql, args,

// new RowMapper<List<User>>() {

// @Override

// public List<User> mapRow(ResultSet rs, int rowNum)

// throws SQLException {

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

// while (rs.next()) {

// list.add(new User(rs.getInt(1), rs.getString(2), rs

// .getDate(3)));

// }

// return list;

// }

// });

**return** list;

}