**MyBatis主流程分析之(三)-准备SQL语句和参数替换、执行**

原创 2016年05月16日 13:00:34

* 标签：
* [数据库](http://so.csdn.net/so/search/s.do?q=%E6%95%B0%E6%8D%AE%E5%BA%93&t=blog) /
* [mybatis](http://so.csdn.net/so/search/s.do?q=mybatis&t=blog) /
* [数据](http://so.csdn.net/so/search/s.do?q=%E6%95%B0%E6%8D%AE&t=blog) /
* [jdbc](http://so.csdn.net/so/search/s.do?q=jdbc&t=blog)
* 3594

在 [MyBatis主流程分析之(二)-打开会话和数据库操作](http://blog.csdn.net/likewindy/article/details/51399832) 中我们只讲了一个主流程，没有深入了解mybatis是如何准备sql语句，如何替换参数，最后查询，新增和删除数据的。这里再补充一下，深入了解。

一、JDBC方式新增数据

首先，我们看看JDBC是如何实现的，无论mybatis内部怎么实现，肯定还是调用JDBC的。   
这里可以参考[JDBC-基础](http://blog.csdn.net/likewindy/article/details/51282892)

//sql语句？的地方就是PreparedStatement后面要替换的地方

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

PreparedStatement pst = conn.prepareStatement(sql);

for(int i = 101;i<200;i++){

//根据位置设置值

pst.setString(1,"Tom" + i);

pst.setString(2,(100+i)\*10);

pst.executeUpdate();

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

在这里PreparedStatement准备了SQL语句，根据位置设置了参数，最后调用executeUpdate。那么我们看看MyBatis是如何做到的。

二、mybatis的PreparedStatement的SQL语句准备

这个章节主要解释mybatis如何实现JDBC的PreparedStatement pst = conn.prepareStatement(sql);

在我们创建StatementHandler的实现类时候，它的基类BaseStatementHandler实现了上面的功能，见源代码

BaseStatementHandler类

//代码有删减

public Statement prepare(Connection connection) throws SQLException {

Statement statement = null;

try {

statement = instantiateStatement(connection);//实现的主要地方

return statement;

} catch (SQLException e) {

closeStatement(statement);

throw e;

} catch (Exception e) {

closeStatement(statement);

}

}

protected Statement instantiateStatement(Connection connection) throws SQLException {

//这个boundSql,在我们创建StatementHandler的实现类的时候就已经创建。这是一个比较主要的类，我们在最后将化点时间讲解这个类。

String sql = boundSql.getSql();

if (mappedStatement.getKeyGenerator() instanceof Jdbc3KeyGenerator) {

.....

} else if (mappedStatement.getResultSetType() != null) {

return connection.prepareStatement(sql, mappedStatement.getResultSetType().getValue(), ResultSet.CONCUR\_READ\_ONLY);

} else {

//这里就是JDBC的conn.prepareStatement(sql);

return connection.prepareStatement(sql);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

三、mybatis的PreparedStatement的参数设置

这个章节主要解释了JDBC的 pst.setString(1,”Tom” + i);

在DefaultParameterHandler类中(handler.parameterize(stmt)方法中调用)

public void setParameters(PreparedStatement ps) throws SQLException {

ErrorContext.instance().activity("setting parameters").object(mappedStatement.getParameterMap().getId());

//获取所有参数

List<ParameterMapping> parameterMappings = boundSql.getParameterMappings();

if (parameterMappings != null) {

for (int i = 0; i < parameterMappings.size(); i++) {

//获取某个参数

ParameterMapping parameterMapping = parameterMappings.get(i);

if (parameterMapping.getMode() != ParameterMode.OUT) {

Object value;

//参数的名字，属性

String propertyName = parameterMapping.getProperty();

//先从附加的

if (boundSql.hasAdditionalParameter(propertyName)) { // issue #448 ask first for additional params

value = boundSql.getAdditionalParameter(propertyName);

} else if (parameterObject == null) {

value = null;

} else if (typeHandlerRegistry.hasTypeHandler(parameterObject.getClass())) {

//typeHandlerRegistry注册了某个类的处理

value = parameterObject;

} else {

//默认的MetaObject 的处理，根据参数获取值,这个将和boundSql一起解释

MetaObject metaObject = configuration.newMetaObject(parameterObject);

value = metaObject.getValue(propertyName);

}

//参数列的TypeHandler

TypeHandler typeHandler = parameterMapping.getTypeHandler();

//jdbcType的处理

JdbcType jdbcType = parameterMapping.getJdbcType();

if (value == null && jdbcType == null) jdbcType = configuration.getJdbcTypeForNull();

//见下面的分析，这里实现了 JDBC的 pst.setString(1,"Tom" + i);

typeHandler.setParameter(ps, i + 1, value, jdbcType);

}

}

}

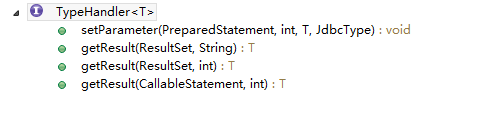
}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36

类型处理器的作用就是   
- 查询时把数据库存储的值转换成java类型   
- 修改是把java类型转换成数据库类型存储，处理   
- 下面这个表格描述了默认的类型处理器。

| **类型处理器** | **Java 类型** | **JDBC 类型** |
| --- | --- | --- |
| BooleanTypeHandler | java.lang.Boolean, boolean | 任何兼容的布尔值 |
| ByteTypeHandler | java.lang.Byte, byte | 任何兼容的数字或字节类型 |
| ShortTypeHandler | java.lang.Short, short | 任何兼容的数字或短整型 |
| IntegerTypeHandler | java.lang.Integer, int | 任何兼容的数字和整型 |
| LongTypeHandler | java.lang.Long, long | 任何兼容的数字或长整型 |
| FloatTypeHandler | java.lang.Float, float | 任何兼容的数字或单精度浮点型 |
| DoubleTypeHandler | java.lang.Double, double | 任何兼容的数字或双精度浮点型 |
| BigDecimalTypeHandler | java.math.BigDecimal | 任何兼容的数字或十进制小数类型 |
| StringTypeHandler | java.lang.String | CHAR 和 VARCHAR 类型 |
| ClobTypeHandler | java.lang.String | CLOB 和 LONGVARCHAR 类型 |
| NStringTypeHandler | java.lang.String | NVARCHAR 和 NCHAR 类型 |
| NClobTypeHandler | java.lang.String | NCLOB 类型 |
| ByteArrayTypeHandler | byte[] | 任何兼容的字节流类型 |
| BlobTypeHandler | byte[] | BLOB 和 LONGVARBINARY 类型 |
| DateTypeHandler | java.util.Date | TIMESTAMP 类型 |
| DateOnlyTypeHandler | java.util.Date | DATE 类型 |
| TimeOnlyTypeHandler | java.util.Date | TIME 类型 |
| SqlTimestampTypeHandler | java.sql.Timestamp | TIMESTAMP 类型 |
| SqlDateTypeHandler | java.sql.Date | DATE 类型 |
| SqlTimeTypeHandler | java.sql.Time | TIME 类型 |
| ObjectTypeHandler | Any | 其他或未指定类型 |
| EnumTypeHandler | Enumeration Type | VARCHAR-任何兼容的字符串类型, 作为代码存储(而不是索引) |
| EnumOrdinalTypeHandler | Enumeration Type | Any compatible NUMERIC or DOUBLE, as the position is stored (not the code itself). |

-   
TypeHandle的接口



* BaseTypeHandler类

public void setParameter(PreparedStatement ps, int i, T parameter, JdbcType jdbcType) throws SQLException {

if (parameter == null) {

if (jdbcType == null) {

throw new TypeException("JDBC requires that the JdbcType must be specified for all nullable parameters.");

}

try {

ps.setNull(i, jdbcType.TYPE\_CODE);

} catch (SQLException e) {

throw new TypeException("Error setting null for parameter #" + i + " with JdbcType " + jdbcType + " . " +

"Try setting a different JdbcType for this parameter or a different jdbcTypeForNull configuration property. " +

"Cause: " + e, e);

}

} else {

//调用实现类

setNonNullParameter(ps, i, parameter, jdbcType);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 其中StringTypeHandler的实现

//设置第i的值

@Override

public void setNonNullParameter(PreparedStatement ps, int i, String parameter, JdbcType jdbcType)

throws SQLException {

ps.setString(i, parameter);

}

//根据columnName获取值

@Override

public String getNullableResult(ResultSet rs, String columnName)

throws SQLException {

return rs.getString(columnName);

}

//根据columnIndex获取值

@Override

public String getNullableResult(ResultSet rs, int columnIndex)

throws SQLException {

return rs.getString(columnIndex);

}

//返回的是CallableStatement 获取值

@Override

public String getNullableResult(CallableStatement cs, int columnIndex)

throws SQLException {

return cs.getString(columnIndex);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

四、mybatis的PreparedStatement的executeUpdate

这里解释了mybatis如何实现JDBC的pst.executeUpdate();   
在PreparedStatementHandler的实现类中

public int update(Statement statement) throws SQLException {

PreparedStatement ps = (PreparedStatement) statement;

//执行语句

ps.execute();

//获取影响的函数

int rows = ps.getUpdateCount();

Object parameterObject = boundSql.getParameterObject();

KeyGenerator keyGenerator = mappedStatement.getKeyGenerator();

keyGenerator.processAfter(executor, mappedStatement, ps, parameterObject);

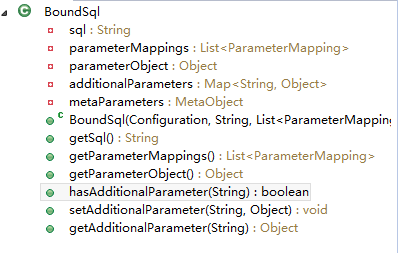
return rows;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

五、BoundSql类

BoundSql的作用   
1. 存储我们的sql语   
2. 存储传入参数对象   
3. 利用MetaObjec对象获取或设置2（存储传入参数对象）中的值



public BoundSql(Configuration configuration, String sql, List<ParameterMapping> parameterMappings, Object parameterObject) {

this.sql = sql;

this.parameterMappings = parameterMappings;

this.parameterObject = parameterObject;

this.additionalParameters = new HashMap<String, Object>();

this.metaParameters = configuration.newMetaObject(additionalParameters);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

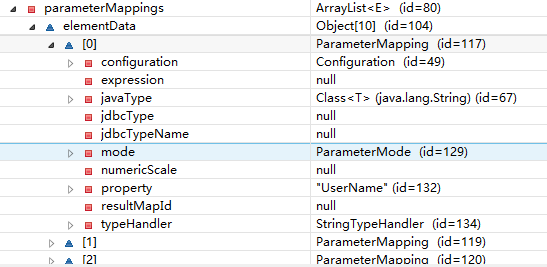
主要属性   
- sql

update mybatis.user set UserName=?,

UserEmail=?

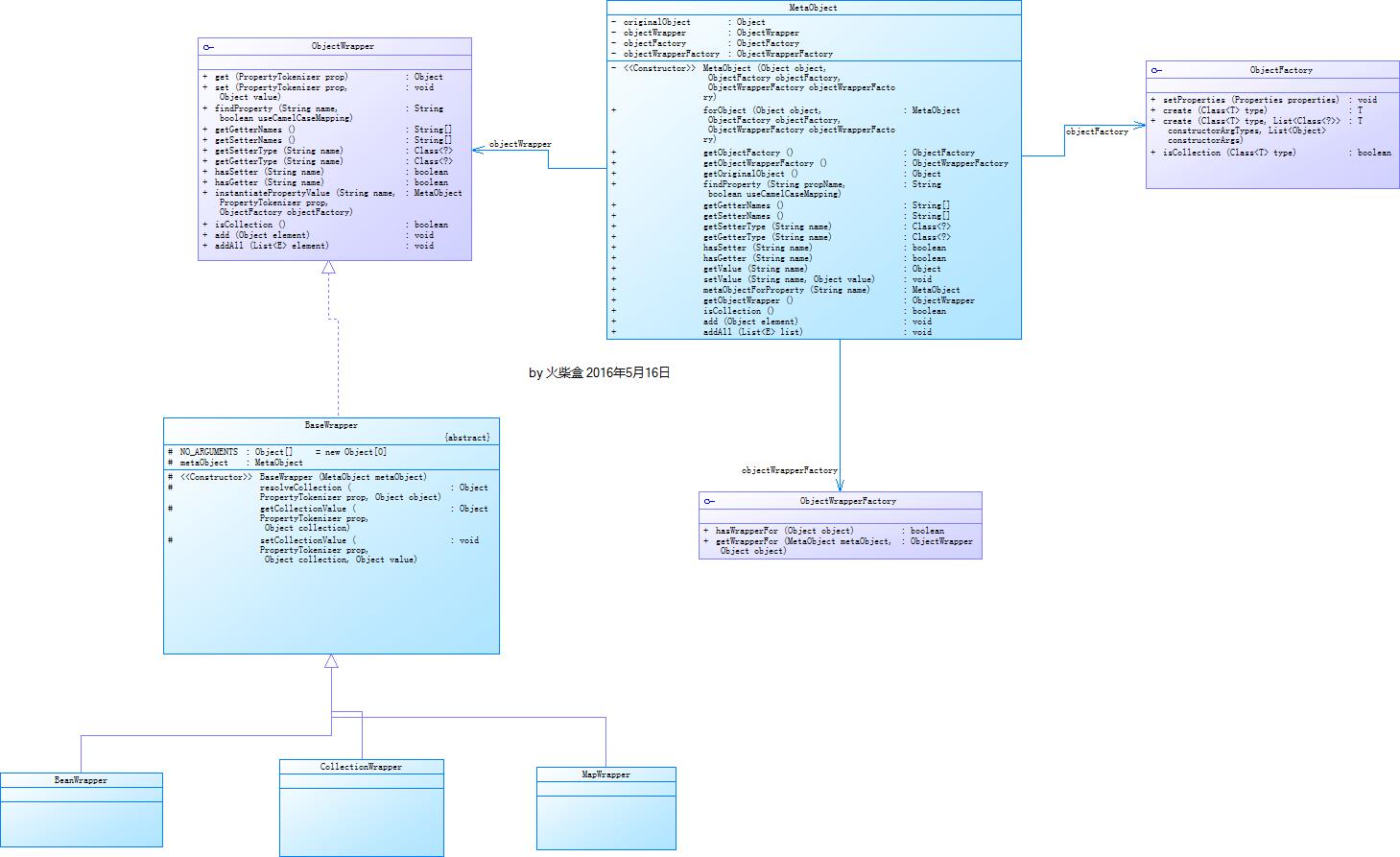
where userId= ?

* 1
* 2
* 3
* parameterMappings

parameterMappings内的parameterMapping分别为UserName，UserEmail和userId对象。和参数顺序一直。   


* parameterObject   
  例如一个pojo对象，User对象类。
* additionalParameters   
  空new HashMap
* metaParameters   
  MetaObjec的一个变量，见下文。

六、MetaObject类

先看看 MetaObject、ObjectWrapper和ObjectFactory、ObjectWrapperFactory的关系。   


MetaObject的属性及其方法

private Object originalObject;//例如一个pojo对象，例如User对象类。

private ObjectWrapper objectWrapper;//根据参数对象的不同，在构造函数中配置

private ObjectFactory objectFactory;//Configuration中配置

private ObjectWrapperFactory objectWrapperFactory;//Configuration中配置

//构造函数，私有

private MetaObject(Object object, ObjectFactory objectFactory, ObjectWrapperFactory objectWrapperFactory) {

this.originalObject = object;

this.objectFactory = objectFactory;

this.objectWrapperFactory = objectWrapperFactory;

if (object instanceof ObjectWrapper) {

//如果参数对象实现了ObjectWrapper

this.objectWrapper = (ObjectWrapper) object;

} else if (objectWrapperFactory.hasWrapperFor(object)) {

//如果objectWrapperFactory已经包装了对象，对用objectWrapperFactory的getWrapperFor

this.objectWrapper = objectWrapperFactory.getWrapperFor(this, object);

} else if (object instanceof Map) {

//是一个Map对象，使用mybatis的MapWrapper

this.objectWrapper = new MapWrapper(this, (Map) object);

} else if (object instanceof Collection) {

//是一个CollectionWrapper对象

this.objectWrapper = new CollectionWrapper(this, (Collection) object);

} else {

//其他默认使用BeanWrapper

this.objectWrapper = new BeanWrapper(this, object);

}

}

//对外公开的静态方法

public static MetaObject forObject(Object object, ObjectFactory objectFactory, ObjectWrapperFactory objectWrapperFactory) {

if (object == null) {

return SystemMetaObject.NULL\_META\_OBJECT;

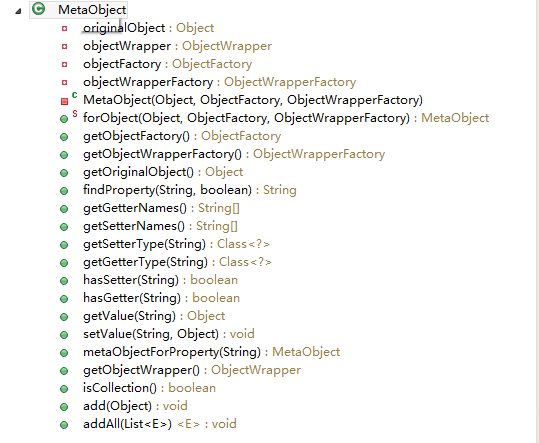
} else {

return new MetaObject(object, objectFactory, objectWrapperFactory);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37

主要的方法   
   
通过这个MetaObject对象可以很方便获取或设置originalObject对象（传入对象参数）的值。   
很有趣的是mybatis获取数据配合也调用了这个类，包装的对象是PooledDataSource。

这里解释了[MyBatis主流程分析之(一)-环境准备](http://blog.csdn.net/likewindy/article/details/51399741)中的ObjectFactory、ObjectWrapperFactory的用法和作用。

版权声明：FREE