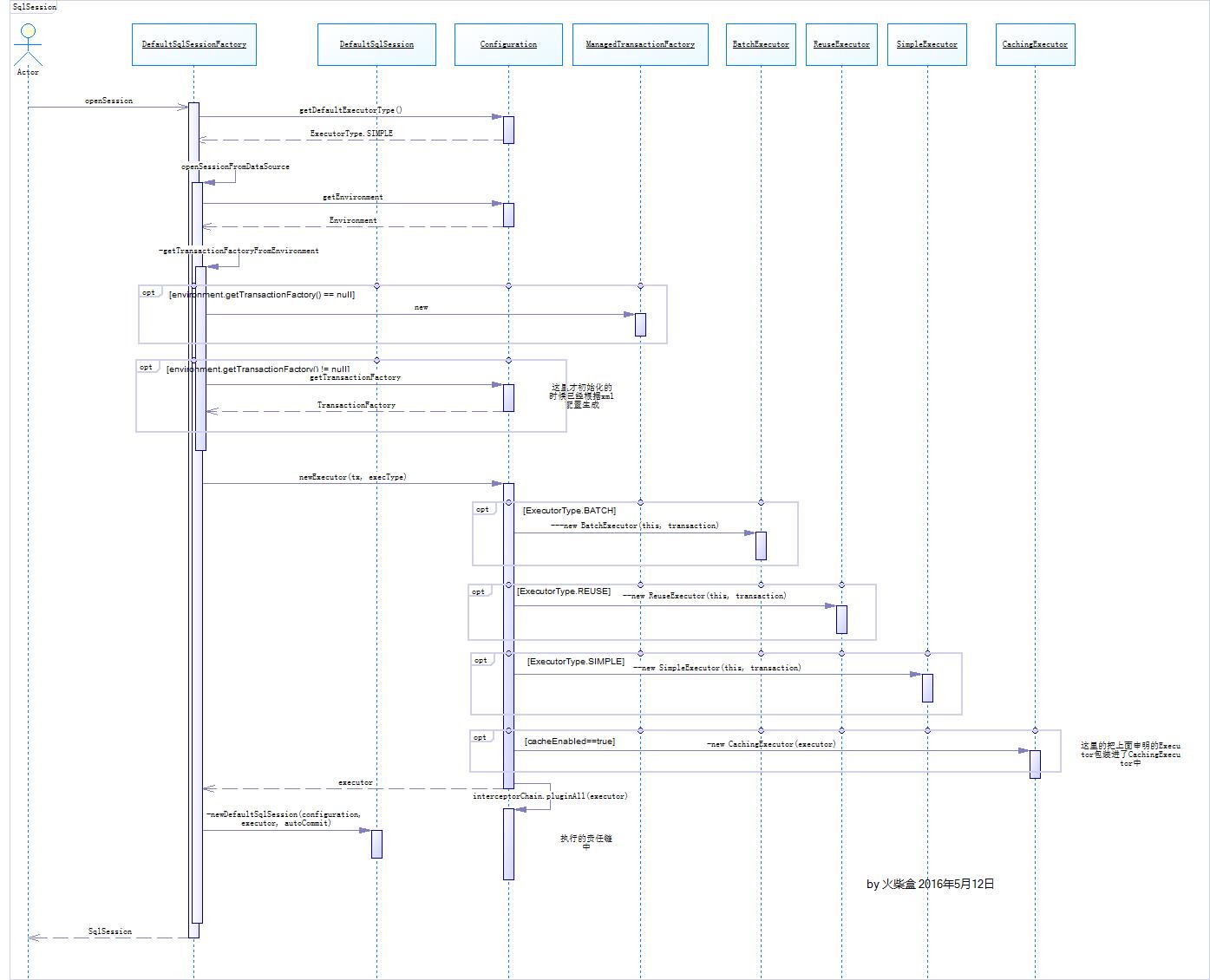
**MyBatis主流程分析之(二)-打开会话和数据库操作**

原创 2016年05月14日 00:09:04

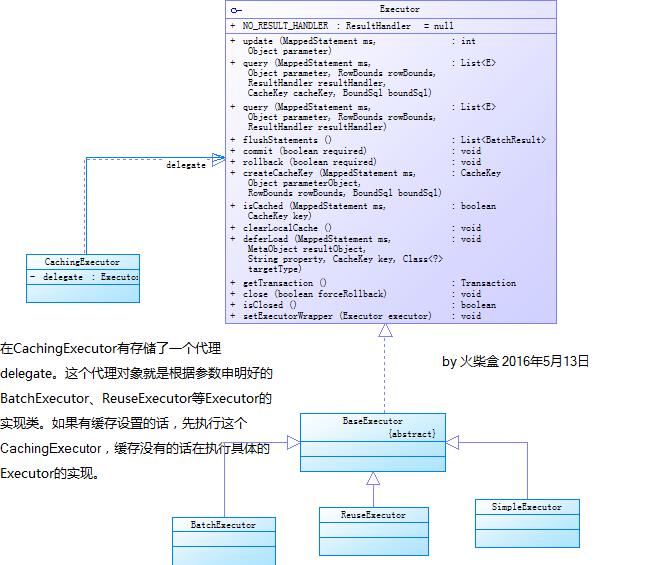
* 标签：
* [数据库](http://so.csdn.net/so/search/s.do?q=%E6%95%B0%E6%8D%AE%E5%BA%93&t=blog) /
* [mybatis](http://so.csdn.net/so/search/s.do?q=mybatis&t=blog) /
* [主要流程](http://so.csdn.net/so/search/s.do?q=%E4%B8%BB%E8%A6%81%E6%B5%81%E7%A8%8B&t=blog) /
* [openSessio](http://so.csdn.net/so/search/s.do?q=openSessio&t=blog) /
* [Session](http://so.csdn.net/so/search/s.do?q=Session&t=blog)
* 1324

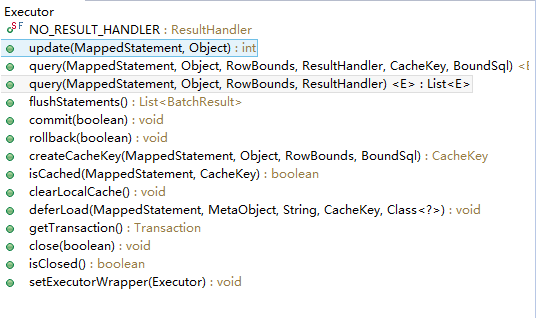
上一篇文章我们初始化了mybatis的环境。这一次我们看看打开会话和数据库的操作。

一、 打开会话（openSession）顺序图



二、 主要类及其说明

在上面的顺序图中，我们可以看到DefaultSqlSessionFactory主要的交互对象还是Configuration，网络上有人喜欢叫他是mybatis的大总管，是的它管理和配置很多东西，在这里我们获取了ExecutorType，getEnvironment和Transaction。并且让它帮忙我们生成了重要的Executor。我们先看看这个Executor类的继承情况。   


* Executor接口   
     
  Executor有一个抽象的实现类BaseExecutor，在BaseExecutor下有三个具体的继承类，分别是BatchExecutor，ReuseExecutor和SimpleExecutor类，这个三个具体选择哪个是根据Configuration类getDefaultExecutorType()获取到，然后mybatis将具体的Executor包装进了CachingExecutor，并且返回了CachingExecutor这个对象。就是先调用CachingExecutor这个对象，如果在CachingExecutor缓存没有的话再调用包装好的对象（SimpleExecutor等）。

部分源代码：

public Executor newExecutor(Transaction transaction, ExecutorType executorType) {

executorType = executorType == null ? defaultExecutorType : executorType;

executorType = executorType == null ? ExecutorType.SIMPLE : executorType;

Executor executor;

if (ExecutorType.BATCH == executorType) {

executor = new BatchExecutor(this, transaction);

} else if (ExecutorType.REUSE == executorType) {

executor = new ReuseExecutor(this, transaction);

} else {

executor = new SimpleExecutor(this, transaction);

}

if (cacheEnabled) {

executor = new CachingExecutor(executor);

}

executor = (Executor) interceptorChain.pluginAll(executor);

return executor;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18

三、 执行数据库操作

在openSession顺序图中创建了这个DefaultSqlSession类，并且把configuration, executor, autoCommit这三个参数都传给了DefaultSqlSession对象。它主要完成SqlSession接口定义的内容。

**SqlSession主要接口定义**

* 获取数据（select）,单个的，多行数据

<T> T selectOne(String statement);

<T> T selectOne(String statement, Object parameter);

<E> List<E> selectList(String statement);

<E> List<E> selectList(String statement, Object arameter);

<E> List<E> selectList(String statement, Object parameter, RowBounds rowBounds);

<K, V> Map<K, V> selectMap(String statement, String mapKey);

<K, V> Map<K, V> selectMap(String statement, Object parameter, String mapKey);

<K, V> Map<K, V> selectMap(String statement, Object parameter, String mapKey, RowBounds rowBounds);

void select(String statement, Object parameter, ResultHandler handler);

void select(String statement, ResultHandler handler);

void select(String statement, Object parameter, RowBounds rowBounds, ResultHandler handler);

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 新增修改删除操作

int insert(String statement);

int insert(String statement, Object parameter);

int update(String statement);

int update(String statement, Object parameter);

int delete(String statement);

int delete(String statement, Object parameter);

* 1
* 2
* 3
* 4
* 5
* 6
* 事务处理

void commit();

void commit(boolean force);

void rollback();

void rollback(boolean force);

* 1
* 2
* 3
* 4

**DefaultSqlSession所要做的事情就是根据参数（例如执行语句的String）获取MappedStatement 对象（MyBatis代码分析之(一)创建）。再把客户端传递过来的参数（例如一个对象User）传递给具体的Executor（openSession创建）去执行查询，新增、修改、删除和事务动作。**

* 查询   
  不管是单个查询还是多记录查询，最终mybatis都是调用下面方法。

//DefaultSqlSession中

public <E> List<E> selectList(String statement, Object parameter, RowBounds rowBounds) {

try {

MappedStatement ms = configuration.getMappedStatement(statement);

List<E> result = executor.query(ms, wrapCollection(parameter), rowBounds, Executor.NO\_RESULT\_HANDLER);

return result;

} catch (Exception e) {

throw ExceptionFactory.wrapException("Error querying database. Cause: " + e, e);

} finally {

ErrorContext.instance().reset();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 修改

public int update(String statement, Object parameter) {

try {

dirty = true;

MappedStatement ms = configuration.getMappedStatement(statement);

return executor.update(ms, wrapCollection(parameter));

} catch (Exception e) {

throw ExceptionFactory.wrapException("Error updating database. Cause: " + e, e);

} finally {

ErrorContext.instance().reset();

}

}

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

public int insert(String statement, Object parameter) {

return update(statement, parameter);//调用updage

}

* 1
* 2
* 3
* 删除

public int delete(String statement, Object parameter) {

return update(statement, parameter);//调用updage

}

* 1
* 2
* 3
* 提交

public void commit(boolean force) {

try {

executor.commit(isCommitOrRollbackRequired(force));

dirty = false;

} catch (Exception e) {

throw ExceptionFactory.wrapException("Error committing transaction. Cause: " + e, e);

} finally {

ErrorContext.instance().reset();

}

}

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

public void rollback(boolean force) {

try {

executor.rollback(isCommitOrRollbackRequired(force));

dirty = false;

} catch (Exception e) {

throw ExceptionFactory.wrapException("Error rolling back transaction. Cause: " + e, e);

} finally {

ErrorContext.instance().reset();

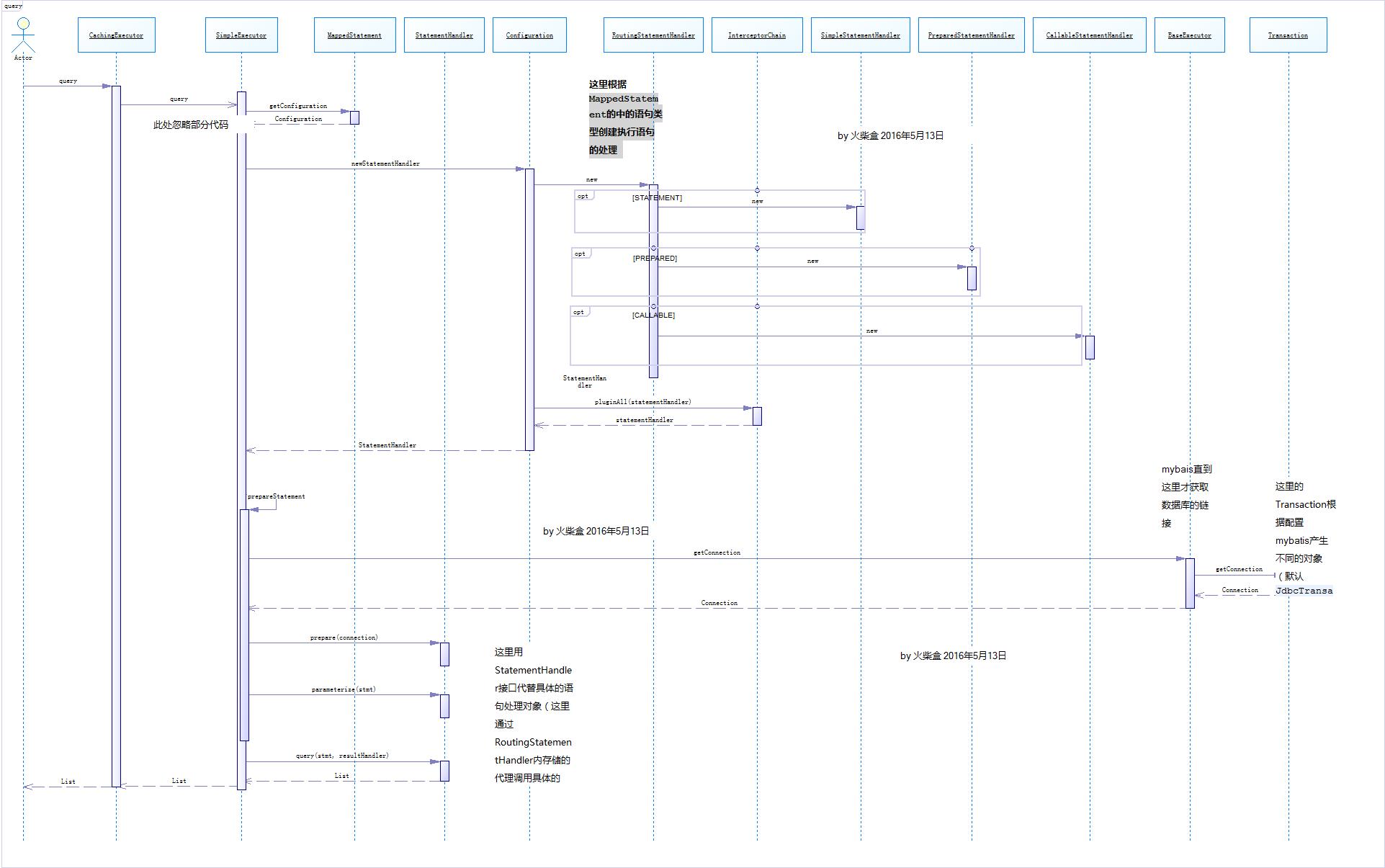
}

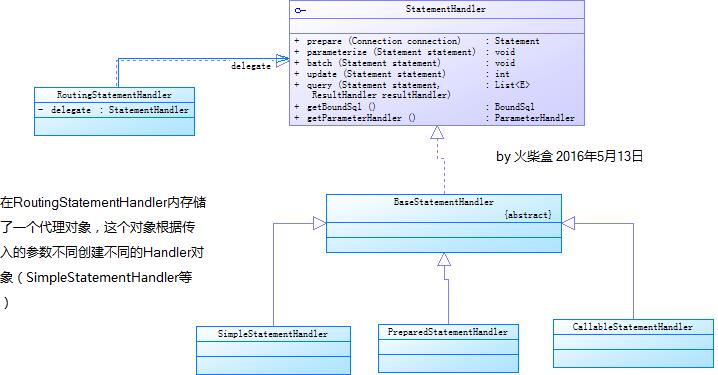
}

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

四、执行一个查询的过程

下面我们分析的是执行一个查询的过程。执行环境都是系统默认的情况。   
- 顺序图

  
在这个顺序图中我们可以看到mybatis先调用了CachingExecutor（这里忽略了部分代码和对象），去缓存中去查询，若缓存没有再调用包装进去的Executor对象（这里是SimpleExecutor）。然后mybatis把创建StatementHandler又一次交给了Configuration类。它根据MappedStatement的执行类型创建不同的处理类。

StatementHandler的继承图：   


这里的StatementHandler和上面的CachingExecutor有点类似。也是弄了一个代理。先调用RoutingStatementHandler，再调用代理的对象。

然后就简单了 ，准备连接数据库，参数化对象，执行查询。

* 查询的源代码：

public <E> List<E> doQuery(MappedStatement ms, Object parameter, RowBounds rowBounds, ResultHandler resultHandler, BoundSql boundSql) throws SQLException {

Statement stmt = null;

try {

Configuration configuration = ms.getConfiguration();

StatementHandler handler = configuration.newStatementHandler(wrapper, ms, parameter, rowBounds, resultHandler, boundSql);

stmt = prepareStatement(handler, ms.getStatementLog());

return handler.<E>query(stmt, resultHandler);

} finally {

closeStatement(stmt);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 更新的源代码：

public int doUpdate(MappedStatement ms, Object parameter) throws SQLException {

Statement stmt = null;

try {

Configuration configuration = ms.getConfiguration();

StatementHandler handler = configuration.newStatementHandler(this, ms, parameter, RowBounds.DEFAULT, null, null);

stmt = prepareStatement(handler, ms.getStatementLog());

return handler.update(stmt);

} finally {

closeStatement(stmt);

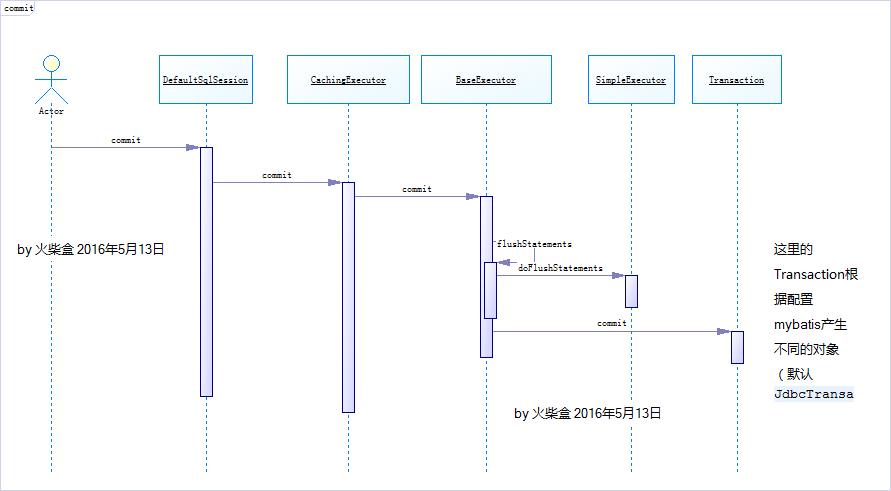
}

}

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

有兴趣的可以去看看更新的顺序图，道理是一样的，最后调用的是 return handler.update(stmt);。

六、事务过程

* 提交顺序图   
  部分代码有忽略   
    
  在这里提交最终交给了环境配置的对象类来处理具体的提交。