1、#{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。

Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理${}时，就是把${}替换成变量的值。

使用#{}可以有效的防止SQL注入，提高系统安全性。

1. Xml映射文件中，除了常见的select|insert|updae|delete标签之外，还有哪些标签？

还有很多其他的标签，<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态sql的9个标签，trim|where|set|foreach|if|choose|when|otherwise|bind等，其中<sql>为sql片段标签，通过<include>标签引入sql片段，<selectKey>为不支持自增的主键生成策略标签。

3、最佳实践中，通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

Dao接口，就是人们常说的Mapper接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement，举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个、、、标签，都会被解析为一个MappedStatement对象。

Dao接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

1. Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？

Mybatis动态sql可以让我们在Xml映射文件内，以标签的形式编写动态sql，完成逻辑判断和动态拼接sql的功能，Mybatis提供了9种动态sql标签trim|where|set|foreach|if|choose|when|otherwise|bind。

其执行原理为，使用OGNL从sql参数对象中计算表达式的值，根据表达式的值动态拼接sql，以此来完成动态sql的功能。

if

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if>

choose (when, otherwise)

<choose>

<when test="title != null">

AND title like #{title}

</when>

<when test="author != null and author.name != null">

AND author\_name like #{author.name}

</when>

<otherwise>

AND featured = 1

</otherwise>

</choose>

trim (where, set)

WHERE

<if test="state != null">

state = #{state}

</if>

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if>

update Author

<set>

<if test="username != null">username=#{username},</if>

<if test="password != null">password=#{password},</if>

<if test="email != null">email=#{email},</if>

<if test="bio != null">bio=#{bio}</if>

</set>

where id=#{id}

/update

Foreach

WHERE ID in

<foreach item="item" index="index" collection="list"

open="(" separator="," close=")">

#{item}

</foreach>

5、Mybatis是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式？

第一种是使用<resultMap>标签，逐一定义列名和对象属性名之间的映射关系。第二种是使用sql列的别名功能，将列别名书写为对象属性名，比如T\_NAME AS NAME，对象属性名一般是name，小写，但是列名不区分大小写，Mybatis会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成T\_NAME AS NaMe，Mybatis一样可以正常工作。 有了列名与属性名的映射关系后，Mybatis通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

1. Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？

不同的Xml映射文件，如果配置了namespace，那么id可以重复；如果没有配置namespace，那么id不能重复；毕竟namespace不是必须的，只是最佳实践而已。

原因就是namespace+id是作为Map

7、MyBatis里面怎么样获得刚插入的主键

insert 方法总是返回一个int值 - 这个值代表的是插入的行数。

而自动生成的键值在 insert 方法执行完后可以被设置到传入的参数对象中。

示例:

<insert id=”insertname” usegeneratedkeys=”true” keyproperty=”id”>

insert into names (name) values (#{name})

</insert>

name name = new name();

name.setname(“fred”);

int rows = mapper.insertname(name);

// 完成后,id已经被设置到对象中

system.out.println(“rows inserted = ” + rows);

system.out.println(“generated key value = ” + name.getid());

8、MyBatis(IBatis)的优势是什么

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。

### 入参无需用对象封装（或者map封装）,使用@Param注解

### 2. 无需写ResultMap。开启mapUnderscoreToCamelCase配置即可

### . 无需写的DAO层实现类

### 4. 支持Ognl表达式

### pagehelper,sorthelper等插件的支持

从 iBatis 到 MyBatis，不只是名称上的变化，MyBatis 提供了更为强大的功能，同时并没有损失其易用性，相反，在很多地方都借助于 JDK 的泛型和注解特性进行了简化。

Mybatis实现了接口绑定，使用更加方便。   
在ibatis2.x中我们需要在DAO的实现类中指定具体对应哪个xml映射文件，   
而Mybatis实现了DAO接口与xml映射文件的绑定，自动为我们生成接口的具体实现，使用起来变得更加省事和方便。

MyBatis 的强大特性之一便是它的动态 SQL。

9、Mybatis都有哪些Executor执行器？它们之间的区别是什么？(自行查阅官方文档)

Mybatis有三种基本的Executor执行器，SimpleExecutor、ReuseExecutor、BatchExecutor。

SimpleExecutor：每执行一次update或select，就开启一个Statement对象，用完立刻关闭Statement对象。

ReuseExecutor：执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，不关闭Statement对象，而是放置于Map

## 3、Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页，可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

## 4、简述Mybatis的插件运行原理，以及如何编写一个插件。

Mybatis仅可以编写针对ParameterHandler、ResultSetHandler、StatementHandler、Executor这4种接口的插件，Mybatis使用JDK的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这4种接口对象的方法时，就会进入拦截方法，具体就是InvocationHandler的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

实现Mybatis的Interceptor接口并复写intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

## 5、Mybatis执行批量插入，能返回数据库主键列表吗？

能，JDBC都能，Mybatis当然也能。

## 8、Mybatis能执行一对一、一对多的关联查询吗？都有哪些实现方式。

能，Mybatis不仅可以执行一对一、一对多的关联查询，还可以执行多对一，多对多的关联查询，多对一查询，其实就是一对一查询，只需要把selectOne()修改为selectList()即可；多对多查询，其实就是一对多查询，只需要把selectOne()修改为selectList()即可。

关联对象查询，有两种实现方式，一种是单独发送一个sql去查询关联对象，赋给主对象，然后返回主对象。另一种是使用嵌套查询，嵌套查询的含义为使用join查询，一部分列是A对象的属性值，另外一部分列是关联对象B的属性值，好处是只发一个sql查询，就可以把主对象和其关联对象查出来。

## 9、Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是一对一，collection指的就是一对多查询。在Mybatis配置文件中，可以配置是否启用延迟加载lazyLoadingEnabled=true|false。

它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成a.getB().getName()方法的调用。这就是延迟加载的基本原理。

当然了，不光是Mybatis，几乎所有的包括Hibernate，支持延迟加载的原理都是一样的。

## 11、Mybatis中如何执行批处理？

使用BatchExecutor完成批处理。

首先,创建一个简单的insert语句:

<insert id=”insertname”>

insert into names (name) values (#{value})

</insert>

然后在java代码中像下面这样执行批处理插入:

list<string> names = new arraylist();

names.add(“fred”);

names.add(“barney”);

names.add(“betty”);

names.add(“wilma”);

// 注意这里 executortype.batch

sqlsession sqlsession = sqlsessionfactory.opensession(executortype.batch);

try {

namemapper mapper = sqlsession.getmapper(namemapper.class);

for (string name : names) {

mapper.insertname(name);

}

sqlsession.commit();

} finally {

sqlsession.close();

}

## 13、Mybatis中如何指定使用哪一种Executor执行器？

在Mybatis配置文件中，可以指定默认的ExecutorType执行器类型，也可以手动给DefaultSqlSessionFactory的创建SqlSession的方法传递ExecutorType类型参数。

## 14、Mybatis是否可以映射Enum枚举类？

Mybatis可以映射枚举类，不单可以映射枚举类，Mybatis可以映射任何对象到表的一列上。映射方式为自定义一个TypeHandler，实现TypeHandler的setParameter()和getResult()接口方法。TypeHandler有两个作用，一是完成从javaType至jdbcType的转换，二是完成jdbcType至javaType的转换，体现为setParameter()和getResult()两个方法，分别代表设置sql问号占位符参数和获取列查询结果。

## 15、Mybatis映射文件中，如果A标签通过include引用了B标签的内容，请问，B标签能否定义在A标签的后面，还是说必须定义在A标签的前面？

虽然Mybatis解析Xml映射文件是按照顺序解析的，但是，被引用的B标签依然可以定义在任何地方，Mybatis都可以正确识别。

原理是，Mybatis解析A标签，发现A标签引用了B标签，但是B标签尚未解析到，尚不存在，此时，Mybatis会将A标签标记为未解析状态，然后继续解析余下的标签，包含B标签，待所有标签解析完毕，Mybatis会重新解析那些被标记为未解析的标签，此时再解析A标签时，B标签已经存在，A标签也就可以正常解析完成了。

## 16、简述Mybatis的Xml映射文件和Mybatis内部数据结构之间的映射关系？

Mybatis将所有Xml配置信息都封装到All-In-One重量级对象Configuration内部。在Xml映射文件中，<parameterMap>标签会被解析为ParameterMap对象，其每个子元素会被解析为ParameterMapping对象。<resultMap>标签会被解析为ResultMap对象，其每个子元素会被解析为ResultMapping对象。每一个<select>、<insert>、<update>、<delete>标签均会被解析为MappedStatement对象，标签内的sql会被解析为BoundSql对象。

## 17、为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？

Hibernate属于全自动ORM映射工具，使用Hibernate查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而Mybatis在查询关联对象或关联集合对象时，需要手动编写sql来完成，所以，称之为半自动ORM映射工具。

面试题看似都很简单，但是想要能正确回答上来，必定是研究过源码且深入的人，而不是仅会使用的人或者用的很熟的人，以上所有面试题及其答案所涉及的内容，在我的Mybatis系列博客中都有详细讲解和原理分析。

2、当实体类中的属性名和表中的字段名不一样 ，怎么办 ？

第1种： 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致

<select id=”selectorder” parametertype=”int” resultetype=”me.gacl.domain.order”>

select order\_id id, order\_no orderno ,order\_price price form orders where order\_id=#{id};

</select>

第2种： 通过<resultMap>来映射字段名和实体类属性名的一一对应的关系

<select id="getOrder" parameterType="int" resultMap="orderresultmap">

select \* from orders where order\_id=#{id}

</select>

<resultMap type=”me.gacl.domain.order” id=”orderresultmap”>

<!–用id属性来映射主键字段–>

<id property=”id” column=”order\_id”>

<!–用result属性来映射非主键字段，property为实体类属性名，column为数据表中的属性–>

<result property = “orderno” column =”order\_no”/>

<result property=”price” column=”order\_price” />

</reslutMap>

3、 模糊查询like语句该怎么写?

第1种：在Java代码中添加sql通配符。

1

2

string wildcardname = “%smi%”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like #{value}

</select>

第2种：在sql语句中拼接通配符，会引起sql注入

string wildcardname = “smi”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like "%"#{value}"%"

</select>

iBatis 框架的主要优势：  
  
1、iBatis 封装了绝大多数的 JDBC 样板代码，使得开发者只需关注 SQL 本身，而不需要花费精力去处理例如注册驱动，创建 Connection，以及确保关闭 Connection 这样繁杂的代码。  
  
2、从 iBatis 到 MyBatis，不只是名称上的变化，MyBatis 提供了更为强大的功能，同时并没有损失其易用性，相反，在很多地方都借助于 JDK 的泛型和注解特性进行了简化  
  
常用的 Java EE 框架，应该都知道这些框架需要提供一个全局配置文件，用于指定程序正常运行所需的设置和参数信息。而针对常用的持久层框架而言（Hibernate、JPA、iBatis 等），则通常需要配置两类文件：  
  
一类用于指定数据源、事务属性以及其他一些参数配置信息（通常是一个独立的文件，可以称之为全局配置文件）；  
  
另一类则用于指定数据库表和程序之间的映射信息（可能不止一个文件，我们称之为映射文件）。  
  
MyBatis 也不例外，虽然其中的一部分可以通过注解的形式进行，但是这两部分内容本身仍是必不可少的  
  
MyBatis 全局配置文件中可以配置的信息主要包括如下几个方面：  
  
    properties — 用于提供一系列的键值对组成的属性信息，该属性信息可以用于整个配置文件中。  
    settings — 用于设置 MyBatis 的运行时方式，比如是否启用延迟加载等。  
    typeAliases — 为 Java 类型指定别名，可以在 XML 文件中用别名取代 Java 类的全限定名。  
    typeHandlers — 在 MyBatis 通过 PreparedStatement 为占位符设置值，或者从 ResultSet 取出值时，特定类型的类型处理器会被执行。  
    objectFactory — MyBatis 通过 ObjectFactory 来创建结果对象。可以通过继承 DefaultObjectFactory 来实现自己的 ObjectFactory 类。  
    plugins — 用于配置一系列拦截器，用于拦截映射 SQL 语句的执行。可以通过实现 Interceptor 接口来实现自己的拦截器。  
    environments — 用于配置数据源信息，包括连接池、事务属性等。  
    mappers — 程序中所有用到的 SQL 映射文件都在这里列出，这些映射 SQL 都被 MyBatis 管理。  
  
上面提及的大多数元素都不是必需的，通常 MyBatis 会为没有显式设置的元素提供缺省值。  
  
 iBatis 2.x 和 MyBatis 3.0.x 的区别  
  
1、 全局配置文件命名  
  
iBatis 通常把为 sqlMapConfig.xml，文件名本身并没有要求，在 MyBatis 中经常会将该文件命名为 Configuration.xml  
  
2、 全局配置文件  
  
<?xml version=”1.0″ encoding=”UTF-8″ ?> iBatis 和 MyBatis 的全局配置文件使用不同的 DTD 约束，在将应用由 iBatis 升级至 MyBatis 时需要注意（两者的映射文件 DTD 约束也不相同）  
  
<!DOCTYPE configuration PUBLIC “-//mybatis.org//DTD Config 3.0//EN” “http://mybatis.org/dtd/mybatis-3-config.dtd”>  
  
<configuration>  
  
<!– 配置数据源相关的信息 –>  
  
<environments default=”demo”>  
  
<environment id=”demo”>  
  
<transactionManager type=”JDBC”/>  
  
<dataSource type=”POOLED”>  
  
<property name=”driver” value= … />  
  
<property name=”url” value= … />  
  
<property name=”username” value=”root”/>  
  
<property name=”password” value=”root”/>  
  
</dataSource>  
  
</environment>  
  
</environments>  
  
<!– 列出映射文件 –>  
  
<mappers>  
  
<mapper resource=”footmark/mybatis/demo/UserInfoMapper.xml”/>  
  
</mappers>  
  
</configuration>  
  
有了这些信息，MyBatis 便能够和数据库建立连接，并应用给定的连接池信息和事务属性。  
  
MyBatis 封装了这些操作，最终暴露一个 SqlSessionFactory 实例供开发者使用，从名字可以看出来，  
  
这是一个创建 SqlSession 的工厂类，通过 SqlSession 实例，开发者能够直接进行业务逻辑的操作，  
  
而不需要重复编写 JDBC 相关的样板代码。根据全局配置文件生成 SqlSession 的代码如下：  
  
Reader reader = Resources.getResourceAsReader("Configuration.xml");   
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);  
SqlSession sqlSession = sqlSessionFactory.openSession();  
  
上面的三行代码看做是 MyBatis 创建 SqlSession 的样板代码。  
  
其中第一行代码在类路径上加载配置文件，Resources 是 MyBatis 提供的一个工具类，它用于简化资源文件的加载，它可以访问各种路径的文件，不过最常用的还是示例中这种基于类路径的表示方式  
  
在完成全局配置文件，并通过 MyBatis 获得 SqlSession 对象之后，便可以执行数据访问操作了  
  
— 设置属性的区别  
  
iBatis ：<settings props1=”value1″ props2=”value2″… />  
  
MyBatis ：<settings> <setting name=”props1″ value=”value1″/> <setting name=”props2″ value=”value2″/> …… </settings>  
  
—配置事务管理器和数据源的区别  
  
iBatis ：  
  
<transactionManager type=”JDBC” >  
  
<dataSource type=”SIMPLE”>  
  
<property name=”JDBC.Driver” value=”${driver}”/>  
  
</dataSource>  
  
</transactionManager>  
  
MyBatis ：  
  
<environments default=”demo”>  
  
<environment id=”demo”>  
  
<transactionManager type=”JDBC”/>  
  
<dataSource type=”POOLED”>  
  
<property name=”JDBC.Driver” value=”${driver}”/>  
  
</dataSource>  
  
</environment>  
  
</environments>  
  
通过 <environments> 来进行数据源管理，主要是为了简化在多套数据源配置之间的切换，比如开发和发布使用不同的配置。  
  
3、 在映射文件中配置 SQL 语句  
  
<?xml version=”1.0″ encoding=”UTF-8″ ?>  
  
<!DOCTYPE mapper PUBLIC “-//mybatis.org//DTD Mapper 3.0//EN” “http://mybatis.org/dtd/mybatis-3-mapper.dtd”>  
  
<mapper namespace=”mybatis.demo.UserInfoMapper”>  
  
<select id=”selectUser” parameterType=”int” resultType=”mybatis.demo.UserInfo”>  
  
select \* from UserInfo where userid =#{userid}  
  
</select>  
  
</mapper>  
  
在 iBatis 中，namespace 不是必需的，且它的存在没有实际的意义。在 MyBatis 中，namespace 终于派上用场了，它使得映射文件与接口绑定变得非常自然  
  
—指定映射文件的方式的区别  
  
iBatis：<sqlMap resource=… /> <sqlMap resource=… /> <sqlMap resource=… />  
  
MyBatis ：<mappers> <mapper resource=… /> <mapper resource=… /> </mappers>  
  
4、 使用 SqlSession 执行映射文件中配置的 SQL 语句  
  
try {  
  
UserInfo userinfo = (UserInfo) sqlSession.selectOne (“mybatis.demo.UserInfoMapper.getUser”, 2);  
  
System.out.println(userinfo);  
  
} finally {  
  
sqlSession.close();  
  
}  
  
需要注意的是，SqlSession 的使用必需遵守上面的格式，即在 finally 块中将其关闭。以保证资源得到释放，防止出现内存泄露！  
  
5、 在 MyBatis 中使用代码进行配置  
  
DataSource ds = …… // 获取一个 DataSource  
  
TransactionFactory txFactory = new JdbcTransactionFactory();  
  
Environment env = new Environment(“demo”, txFactory, ds);  
  
Configuration cfg = new Configuration(env);  
  
cfg.addMapper(UserInfoMapper.class);  
  
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(cfg);  
  
结合前面的配置文件，很容易理解这段代码的意思，故不再赘述。不过，需要注意的是 Configuration 的 addMapper() 方法，该方法的参数通常是一个接口，可以在接口里面定义若干方法，在方法上使用注解来指定映射的 SQL 语句。一个典型的接口定义以及对应的数据访问方法如下：  
  
6、 将映射的 SQL 语句与接口中的方法绑定  
  
// 映射 SQL 绑定接口  
  
public interface UserInfoMapper {  
  
@Select(“select \* from userinfo where userid = #{userid}”)  
  
public UserInfo getUserInfo(int userid);  
  
}  
  
// 接口绑定对应的数据访问方法  
  
try {  
  
//UserInfo userinfo = (UserInfo) sqlSession.selectOne (“mybatis.demo.UserInfoMapper.selectUser”, 2);  
  
UserInfoMapper userinfoMapper = sqlSession.getMapper(UserInfoMapper.class);  
  
UserInfo userinfo = userinfoMapper.getUserInfo(1);  
  
System.out.println(userinfo);  
  
} finally {  
  
sqlSession.close();  
  
}  
  
7、 MyBatis 映射文件的改变 （仅仅是名称的改变，用法和含义并没有发生变化）  
  
    和全局配置文件一样，由于 DTD 约束发生变化，根元素也由原来的 <sqlMap> 调整为 <mapper>。  
    <select> 等元素的 parameterClass 属性改为了 parameterType 属性。  
    <select> 等元素的 resultClasss 属性改为了 resultType 属性。  
    <parameterMap> 等元素的 class 属性改为了 type 属性。  
    <result> 元素的 columnIndex 属性被移除了。  
    嵌套参数由 #value# 改为了 #{value}。  
    <parameter> 等元素的 jdbcType 属性取值中，原来的 “ORACLECURSOR” 取值改为了现在的 “CURSOR”，”NUMBER” 取值改为了 “NUMERIC”。  
  
iBatis/MyBatis 对存储过程的支持一直是值得称道的。之前通过使用 <procedure> 元素进行存储过程的定义，示例如下：  
  
— 存储过程的区别  
  
iBatis：  
  
<procedure id=”getValues” parameterMap=”getValuesPM”>  
  
{ ? = call pkgExample.getValues(p\_id => ?) }  
  
</procedure>  
  
MyBatis ：  
  
<select id=”getValues” parameterMap=”getValuesPM” statementType=”CALLABLE”>  
  
{ ? = call pkgExample.getValues(p\_id => ?)}  
  
</select>  
  
通过 statementType 属性将该语句标识为存储过程而非普通 SQL 语句  
  
 8、代码层面的改变  
  
MyBatis 在编码中的最大的改变就是将一个最常用的 API 由 SqlMapClient 改为了 SqlSessionFactory。  
  
另外，类型处理器接口也由原来的 TypeHandlerCallback 改为了 TypeHandler。  
  
最后 DataSourceFactory 也进行了调整，移动到 org.apache.ibatis.datasource 包下，其中的方法也作了微调。总之，代码层面公开的部分改动较少，不会给开发者造成较大的移植成本