

Mybatis框架入门

什么是Mybatis

- Mybatis是优秀 的持久层框架
- Mybatis使用xml将sql与程序解耦，便于维护
- Mybatis学习简单，执行高效JDBC的延伸

单元测试

- 单元测试就是指对软件中的最小可测试单元进行检查和验证
- 测试用例是指编写一段代码对已有的功能进行校验
- Junit4是Java中最著名的单元测试工具，主流的ide内置支持

JUnit4使用方法

- 引入JUnit Jar包或增加maven依赖
- 编写测试用例验证目标方法是否正确
- 在测试用例上增加 `@Test` 注解开始单元测试

Mybatis转义字符

<	<	小于
>	>	大于
&	&	与
'	'	单引号
"	"	双引号

Mybatis环境配置

mybatis-config.xml

- Mybatis采用xml格式配置数据库环境信息
- Mybatis环境配置标签
- environment包含数据库驱动、URL、用户名与密码

mybatis-config.xml



```
<!--配置环境，不同的环境不同的id名字-->
<environment id="dev">
  <!--采用JDBC方式对数据库事务进行commit/rollback-->
  <transactionManager type="JDBC"></transactionManager>
  <!--采用连接池方式管理数据库连接-->
  <dataSource type="POOLED">
    <property name="driver" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/db"/>
    <property name="username" value="root"/>
    <property name="password" value="root"/>
  </dataSource>
</environment>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <!--default为 用哪个环境配置，dev，可以有多个配置环境可以通过 environment id来标识环境 -->
```

```

<environments default="dev">
    <!-- 环境配置 id为唯一标识-->
    <environment id="dev">
        <!--transactionManager type='以什么方式对数据库事务管理'-->
        <transactionManager type="JDBC"></transactionManager>
        <!--数据源 POOLED数据库连接池进行管理-->
        <dataSource type="POOLED">
            <!--配置数据库端口 url 用户名密码-->
            <property name="driver" value="com.mysql.jdbc.Driver"/>
            <property name="url" value="jdbc:mysql://localhost:3306/admin?
useUnicode=true&characterEncoding=UTF-8"/>
            <property name="username" value="root"/>
            <property name="password" value="123456"/>
        </dataSource>
    </environment>
</environments>
</configuration>

```

SqlSessionFactory (sql工厂会话)

- sqlSessionFactory是Mybatis的核心对象
- 用于初始化Mybatis。创建的sqlSession对象
- 保证SqlSessionFactory在应用中全局唯一

sqlSession (重点)

- sqlSession是对Mybatis操作数据库的核心对象
- sqlSession使用JDBC方式与数据库交互
- sqlSession对象提供数据表的CRUD对应的方法

既然有了 SqlSessionFactory，顾名思义，我们可以从中获得 sqlSession 的实例。SqlSession 提供了在数据库执行 SQL 命令所需的所有方法。你可以通过 sqlSession 实例来直接执行已映射的 SQL 语句。

```

/**
 * Mybatis测试类
 */
public class MybatisTest {
    @Test
    public void TestSqlSessionFactory() throws IOException {
        //利用Reader加载ClassPath下的的Mybatis-config核心配置文件
        Reader reader = Resources.getResourceAsReader("mybatis-config.xml");
        //初始化加载SqlSessionFactory对象，同时解析Mybatis.config.xml文件
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
        System.out.println("sqlSessionFactory初始化加载成功");
        sqlSession = null;
        try {
            //创建SqlSession对象，sqlSession是JDBC的拓展类，与数据库交互的
            sqlSession = sqlSessionFactory.openSession();
            //数据库测试连接
            Connection connection = sqlSession.getConnection();
            System.out.println(connection);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (sqlSession != null) {
                /**
                 * 如果配置文件中 dataSource type="POOLED" POOLED代表使用数据库连接池 close则将连接池回收连接池中。
                 * 如果配置文件中 dataSource type="UNPOOLED" UNPOOLED代表不使用数据库连接池 close则是，
                 * 调用 connection.close() 方法关闭
                 */
                sqlSession.close();
            }
        }
    }
}

```

初始化工具类 (MybatisUtils)

SqlSessionFactory是全局唯一，如何保证全局唯一，就要借助到 **MybatisUtils**

```

/**
 * MybatisUtil 是Mybatis工具类，保证SqlSessionFactory保证全局唯一
 */

```

```

public class mybatisUtils {
    //利用static 静态，它属于类，而不属于对象
    private static SqlSessionFactory sqlSessionFactory = null;
    //利用静态代码块，实例化类sqlSessionFactory
    static {
        Reader reader=null;
        try {
            reader = Resources.getResourceAsReader("mybatis-config.xml");
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
        } catch (IOException e) {
            e.printStackTrace();
            //向上抛出初始化ExceptionInInitializerError异常 向调用者通知
            throw new ExceptionInInitializerError(e);
        }
    }
    //创建SqlSession对象，sqlSession是JDBC的拓展类，与数据库交互的
    public static SqlSession OpenSession(){
        return sqlSessionFactory.openSession();
    }
    //关闭资源
    public static void Close(SqlSession sqlSession){
        if (sqlSession !=null){
            sqlSession.close();
        }
    }
}

```

测试类

```

/**
 * MybatisUtils工具类测试
 */
@Test
public void TestMybatisUtils(){
    //初始化
    SqlSession sqlSession =null;
    Connection connection =null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        connection= sqlSession.getConnection();
        System.out.println(connection);
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        //关闭资源
        mybatisUtils.Close(sqlSession);
    }
}
}

```

Mybatis查询操作

- 创建实体entity
- 创建Mapper.xml
- 编写

```

/**
 * 实体类
 */
public class topic {
    private Integer id;

    private Integer userId;

    private Date createTime;

    private Date updateTime;

```

```

private String title;

private String content;

private Integer click;

private Integer tabId;


@Override

public String toString() {

    return "topic{" +

        "id=" + id +

        ", userId=" + userId +

        ", createTime=" + createTime +

        ", updateTime=" + updateTime +

        ", title='" + title + '\'' +

        ", content='" + content + '\'' +

        ", click=" + click +

        ", tabId=" + tabId +

        '}';

}


public Integer getId() {

    return id;

}


public void setId(Integer id) {

    this.id = id;

}


public Integer getUserId() {

    return userId;

}


public void setUserId(Integer userId) {

    this.userId = userId;

}


public Date getCreateTime() {

    return createTime;

}

```

2、创建mapper.xml

```

<?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">
<!--namespace-->
<mapper namespace="topic">
    <!--resultType 返回的结果集类型    id 为执行的id唯一标识-->
<select id="selectAll" resultType="xyz.herther.entity.topic">
    select * from topic limit 0,5
</select>
</mapper>

```

3、开启驼峰命名映射(在mybatis-config.xml中配置)

```

<!--开启驼峰命名    user_id => userId -->
<settings>
    <setting name="mapUnderscoreToCamelCase" value="true"/>
</settings>

```

4、新增 (在mybatis-config.xml配置中添加)

```

<!--配置映射文件-->
<mappers>
    <!--resource mapper的路径-->
    <mapper resource="mappers/topic.xml"/>
</mappers>

```

5、SqlSession执行select语句(测试)

```

/**
 * sql映射查询topic表中的数据用例
 */
@Test
public void TestSelectAll(){
    SqlSession sqlSession =null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        //selectAll 为xml中的id topic 为mapper 标签中的namespace
        List<topic> list = sqlSession.selectList("topic.selectAll");
        for (topic i: list) {
            System.out.println(i.toString());
        }
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        mybatisUtils.Close(sqlSession);
    }
}

```

测试结果

```
D:\JDK\bin\java.exe ...
```

```
topic{id=1, userId=2, createTime=Sat Jan 20 20:02:13 CST 2018, updateTime=Fri Feb 09 13:21:37 CST 2018, ...}
目前使用华为 AR1220-S 组网，使用 12tp 搭建 vpn。使用路由器到路由器的解决方案，现已超过 10 家子公司，使用
随着公司业务增长，今年预计需增加 30 个点接入母公司网络。（未来可能需增加超过 200 个点，且分布全国各地）
```

- 1.请问 AR1220-S 能否胜任，是否需要更换 AR2220-S 或 AR3220-S。
- 2.子公司使用什么设备能较稳定连接母公司通过华为搭建的 12tp 的 vpn。（目前使用过极路由和普联，普联表现较为和
- 3.大规模异地租网是否有更优方案，如利用云中转等，请各位大神们指点，谢谢。', click=139, tabId=1}

```
topic{id=2, userId=3, createTime=Sat Jan 20 19:55:19 CST 2018, updateTime=Fri Feb 09 15:50:38 CST 2018, ...}
根据兴趣推荐？
```

关注的频道优先推荐？

根据天气、位置等客观参数推荐

有收藏功能，能自动根据收藏发现兴趣，从而更好地推荐

让更多的人变成内容的创作者而不是单单的读者？ ', click=40, tabId=1}

```
topic{id=3, userId=2, createTime=Sat Jan 20 19:52:38 CST 2018, updateTime=Thu Feb 08 21:47:28 CST 2018, ...}
#####举例#####
```

问题：9.中国历史上,在位时间最长的皇帝是？

备选答案： ['康熙', '乾隆', '刘彻']

SQL传参

查询 - <select>

参数类型

```
<select id="selectById" parameterType="Integer"
      resultType="com.imooc.mybatis.entity.Goods">
    select * from t_goods where goods_id = #{value}
</select>
```

传递单个参数

引用上文配置（新增一条mapper.xml中sql语句）

```
<!--parameterType 为传参的类型-->
<select id="selectById" parameterType="Integer" resultType="xyz.herther.entity.topic">
  <!--#{value} 为传进来参数的值-->
  select * from topic where id = #{value}
</select>
```

测试用例

```
/**
 * 通过传入id 来查询
 */
@Test
public void TestSelectById(){
    SqlSession sqlSession = null;
```

```

    try {
        sqlSession = mybatisUtils.OpenSession();
        //因为只查询出来一条所以用selectOne，第一个为 对应的sql 语句， 第二给为传进去的参数
        topic t = sqlSession.selectOne("topic.selectById",2);
        System.out.println(t.toString());
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        mybatisUtils.Close(sqlSession);
    }
}
}

```

Connected to the target VM, address: '127.0.0.1:56419', transport: 'socket'

topic{id=2, userId=3, createTime=Sat Jan 20 19:55:19 CST 2018, updateTime=Fri Feb 09 2018}

根据兴趣推荐？

关注的频道优先推荐？

根据天气、位置等客观参数推荐

有收藏功能，能自动根据收藏发现兴趣，从而更好地推荐

让更多的人变成内容的创作者而不是单单的读者？', click=40, tabId=1}

Disconnected from the target VM, address: '127.0.0.1:56419', transport: 'socket'

传递多个参数

使用上面配置（新增一条语句）

```

<!--parameterMap="java.util.Map"传进map对象-->
<select id="selectClick" parameterType="java.util.Map" resultType="xyz.herther.entity.topic">
    <!--max, page, limit为对map中的key-->
    select * from topic where click between #{min} and #{max} limit #{limit},#{page}
</select>

```

测试

```

/**
 * 通过传进limit page min max 参数来查询
 */
@Test
public void TestselectClick(){
    SqlSession sqlSession = null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        Map map = new HashMap();
        map.put("min",10);
        map.put("max",70);
        map.put("page",10);
        map.put("limit",0);
        List<topic> topic1 = sqlSession.selectList("topic.selectClick", map);
        for (topic t: topic1){
            System.out.println(t.getTitle());
        }
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        mybatisUtils.Close(sqlSession);
    }
}
}

```

你心目中理想的新闻推荐系统是什么样子？

关于“直播答题发钱”辅助工具搜索推荐算法的讨论

企业级应用开发真的需要 Vue，React 这种东西吗？

使用 Nginx 自建 CDN，关于回源问题。

JDK 9 里的 JDK HTTP Client 的写法

数据库怎么最简单实现“栈”结构存储？

腾讯云能不能别给我发优惠券短信了

应用程序在服务器上创建文件目录权限 0777，会有风险吗？

Genesis祝各位会员新年好

Mybatis多表联合查询

在原有的mapper上添加

```
<!--返回LinkedHashMap 链表形式的map-->
<!--多表查询-->
<select id="selectMap" resultType="java.util.LinkedHashMap">
    select t.* ,tab.tab_name from topic t join tab on t.tab_id= tab.id
</select>
```

测试方法

```
/**
 * 多表查询
 */
@Test
public void TestselectMap(){
    SqlSession sqlSession =null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        List<Map> list = sqlSession.selectList("topic.selectMap");
        for (Map map: list){
            System.out.println(map);
        }
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        mybatisUtils.Close(sqlSession);
    }
}
```

结果

✔ Tests passed: 1 of 1 test – 490 ms

```
D:\JDK\bin\java.exe ...
```

```
{id=1, user_id=2, create_time=2018-01-20 20:02:13.0, update_time=2018-02-09 13:21:37.0, title=是目前使用华为 AR1220-S 组网，使用 12tp 搭建 vpn。使用路由器到路由器的解决方案，现已超过 10 家子公司，随着公司业务增长，今年预计需增加 30 个点接入母公司网络。（未来可能需增加超过 200 个点，且分布全国各地
```

1.请问 AR1220-S 能否胜任，是否需要更换 AR2220-S 或 AR3220-S。
2.子公司使用什么设备能较稳定连接母公司通过华为搭建的 12tp 的 vpn。（目前使用过极路由和普联，普联表现较差）
3.大规模异地组网是否有更优方案，如利用云中转等，请各位大神们指点，谢谢。 , click=139, tab_id=1, tab_name=你感兴趣的
{id=2, user_id=3, create_time=2018-01-20 19:55:19.0, update_time=2018-02-09 15:50:38.0, title=你感兴趣的根据兴趣推荐？

关注的频道优先推荐？

根据天气、位置等客观参数推荐

有收藏功能，能自动根据收藏发现兴趣，从而更好地推荐

让更多的人变成内容的创作者而不是单单的阅读者？ , click=40, tab_id=1, tab_name=技术}

```
{id=3, user_id=2, create_time=2018-01-20 19:52:38.0, update_time=2018-02-08 21:47:28.0, title= 3. 根据兴趣推荐？}#####举例#####
```

问题：9.中国历史上,在位时间最长的皇帝是？

备选答案： ['康熙', '乾隆', '刘彻']

ResultMap结果映射

- ResultMap可以将查询结果映射为复杂类型的Java对象
- ResultMap适用于Java对象保存多表关联结果
- ResultMap支持对象关联查询等高级特征

在原有的Mapper中新增

```
<!-- type 为dto中的数据实体-->
<!-- 结果集映射-->
<resultMap id="topicdto" type="xyz.herther.dto.TopicDto">
<!-- 注入主键字段与属性映射-->
    <id property="topic.tabId" column="tab_id"></id>
<!-- 注入非主键字段与属性映射-->
    <result property="topic.id" column="id"></result>
    <result property="topic.userId" column="user_id"></result>
    <result property="topic.createTime" column="create_time"></result>
    <result property="topic.updateTime" column="update_time"></result>
    <result property="topic.title" column="title"></result>
    <result property="topic.content" column="content"></result>
    <result property="topic.click" column="click"></result>
    <result property="topic.tabId" column="tabId"></result>
    <result property="tabName" column="tab_name"></result>
    <result property="tabNameEn" column="tab_name_en"></result>
</resultMap>
<select id="selectDto" resultMap="topicdto">
    select t.* ,tab.tab_name from topic t join tab on t.tab_id= tab.id
</select>
```

编写数据对象 (DTO)

```
/**
 * 数据传输对象
 */
public class TopicDto {
    private Topic topic =new Topic();
    private String tabName;
    private String tabNameEn;

    @Override
    public String toString() {
        return "TopicDto{" +
            "topic=" + topic +
```

```

        ", tabName='" + tabName + '\'' +
        ", tabNameEn='" + tabNameEn + '\'' +
        '}';

    }

    public Topic getTopic() {
        return topic;
    }

    public void setTopic(Topic topic) {
        this.topic = topic;
    }

    public String getTabName() {
        return tabName;
    }

    public void setTabName(String tabName) {
        this.tabName = tabName;
    }

    public String getTabNameEn() {
        return tabNameEn;
    }

    public void setTabNameEn(String tabNameEn) {
        this.tabNameEn = tabNameEn;
    }
}

```

测试

```

/**
 * 结果集映射dto
 */
@Test
public void TestSelectDto(){
    SqlSession sqlSession = null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        List<TopicDto> topic = sqlSession.selectList("topic.selectDto");
        for (TopicDto i: topic){
            System.out.println(i.toString());
        }
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        mybatisUtils.Close(sqlSession);
    }
}
}

```

结果

TopicDto{topic=topic{id=1, userId=2, createTime=Sat Jan 20 20:02:13 CST 2018, updateTime=Fri Feb 09 13:21:37 CST 2018, title='是否有',
目前使用华为 AR1220-S 组网, 使用 12tp 搭建 vpn.使用路由器到路由器的解决方案, 现已超过 10 家子公司, 使用路由器通过 12tp 接入母公司内网系统。
随着公司业务增长, 今年预计需增加 30 个点接入母公司网络。（未来可能需增加超过 200 个点, 且分布全国各地）

1.请问 AR1220-S 能否胜任, 是否需要更换 AR2220-S 或 AR3220-S。
2.子公司使用什么设备能较稳定连接母公司通过华为搭建的 12tp 的 vpn。（目前使用过极路由和普联, 普联表现较为稳定。因大规模部署需要, 请问 TL-ER7500 是否合适？）
3.大规模异地组网是否有更优方案, 如利用云中转等, 请各位大神们指点, 谢谢。', click=139, tabId=1}, tabName='技术', tabNameEn='null'}
TopicDto{topic=topic{id=2, userId=3, createTime=Sat Jan 20 19:55:19 CST 2018, updateTime=Fri Feb 09 15:50:38 CST 2018, title='你心目中',
根据兴趣推荐?
关注的频道优先推荐?
根据天气、位置等客观参数推荐
有收藏功能, 能自动根据收藏发现兴趣, 从而更好地推荐
让更多的人变成内容的创作者而不是单纯的阅读者? ', click=40, tabId=1}, tabName='技术', tabNameEn='null'}
TopicDto{topic=topic{id=3, userId=2, createTime=Sat Jan 20 19:52:38 CST 2018, updateTime=Thu Feb 08 21:47:28 CST 2018, title='关于“',
#####举例#####
问题: 9.中国历史上,在位时间最长的皇帝是?
备选答案: ['康熙', '乾隆', '刘彻']

Mybatis数据写入（增删改）

数据库事务

数据库事务是保证数据操作完整性的基础

客户端-->mysql（事务日志（新增日志1） ---> (commit) 数据表）

Mybatis写操作包含三种

- 插入
- 删除
- 更新

新增 -

新增 - <insert>

```
<insert id="insert" parameterType="com.itlaoqi.mybatis.entity.Goods">
    INSERT INTO `babytun`.`t_goods` ( `title`, `sub_title`, `original_cost`, current_price )
    VALUES ( #{title}, #{subTitle}, #{originalCost}, #{currentPrice} )
    <selectKey resultType="int" keyProperty="goodsId" order="AFTER">
        <!-- 当前连接中最后产生的id号 -->
        select last_insert_id()
    </selectKey>
</insert>
```

```
<!--插入操作-->
<insert id="inserTopic" parameterType="xyz.herther.entity.Topic">
    INSERT INTO topic(user_id,create_time,update_time,title,content,click,tab_id)
    VALUES (#{userId},#{createTime},#{updateTime},#{title},#{content},#{click},#{tabId})
    <!--主键回显 resultType返回类型, keyProperty返回对应的实体字段名称, AFTER在insert语句之后 BEFORE在insert之前-->
    <selectKey resultType="Integer" keyProperty="id" order="AFTER">
        SELECT last_insert_id()
    </selectKey>
</insert>
```

测试

```
/**
 * 新增数据操作
 */
@Test
public void TestInsert(){
    SqlSession sqlSession =null;
    try {
        sqlSession=mybatisUtils.OpenSession();
        Topic topic = new Topic();
        topic.setClick(44);
        topic.setContent("测试内容");
        topic.setCreateTime(new Date());
        topic.setTabId(1);
        topic.setTitle("测试标题");
        topic.setUpdateTime(new Date());
        topic.setUserId(4);
        //返回影响的行数
        int insert = sqlSession.insert("topic.inserTopic", topic);
        //提交事务
```

```

        sqlSession.commit();
        System.out.println(topic.getId());
    } catch (Exception e) {
        //出现异常、事务回滚
        if (sqlSession != null) {
            sqlSession.rollback();
        }
    } finally {
        mybatisUtils.Close(sqlSession);
    }
}

```

测试结果

32	4	2021-01-31 22:25:09	2021-01-31 22:25:09	测试标题	测试内容	44	1
----	---	---------------------	---------------------	------	------	----	---

selectKey与useGeneratedKeys的区别（用于获取自动生成的主键）

useGeneratedKeys属性用法

```

<insert id="insert"
        parameterType="com.imooc.mybatis.entity.Goods"
        useGeneratedKeys="true"
        keyProperty="goodsId"
        keyColumn="goods_id">
    INSERT INTO SQL语句
</insert>

```

新增 - <insert>

```

<insert id="insert" parameterType="com.itlaoqi.mybatis.entity.Goods">
    INSERT INTO `babytun`.`t_goods` ( `title`, `sub_title`, `original_cost`, current_price`)
    VALUES ( #{title}, #{subTitle}, #{originalCost}, #{currentPrice})
    <selectKey resultType="int" keyProperty="goodsId" order="AFTER">
        <!-- 当前连接中最后产生的id号 -->
        select last_insert_id()
    </selectKey>
</insert>

```

二者区别-显示与隐示

- selectKey标签需要明确编写获取最新主键的SQL语句
- userGeneratedKeys属性会自动根据驱动生成对应的sql语句

应用场景不同

- selectKey适用于所有关系型数据库
- useGeneratedKeys 只支持“自增主键”类型的数据库

总结

- selectKey标签是通用方案，适用于所有数据库，但编写麻烦
- useGeneratedKeys属性只支持自增主键、数据库使用简单

更新与删除操作

更新 - <update>

```
<update id="update" parameterType="com.itlaoqi.mybatis.entity.Goods" >
    UPDATE `babytun`.`t_goods`
    SET `title` = #{title}
    , `sub_title` = #{subTitle}
    , `category_id` = #{categoryId}
    WHERE `goods_id` = #{goodsId}
</update>
```

删除 - <delete>

```
<delete id="delete" parameterType="Integer">
    delete from t_goods where goods_id = #{value}
</delete>
```

更新:

```

<!--更新操作-->
<update id="updateTopic" parameterType="xyz.herther.entity.Topic">
    UPDATE topic
    SET user_id= #{userId},
        create_time = #{createTime},
        update_time = #{updateTime},
        title = #{title},
        content = #{content},
        click = #{click},
        tab_id = #{tabId}
    WHERE
        id = #{id}
</update>

```

测试

```

/**
 * 更新操作
 */
@Test
public void TestUpdate(){
    SqlSession sqlSession=null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        //先去原来的查询数据
        Topic topic = sqlSession.selectOne("topic.selectById", 4);
        //设置更新的内容
        topic.setTitle("测试更新");
        //执行更新
        int num = sqlSession.update("topic.updateTopic", topic);
        //提交事务
        sqlSession.commit();
        System.out.println(num);

    }catch (Exception e){
        //出现问题，数据回滚
        if(sqlSession !=null){
            sqlSession.rollback();
        }
        e.printStackTrace();
    }finally {
        //关闭资源
        mybatisUtils.Close(sqlSession);
    }

}

```

4	4	2	2018-01-20 20:06:24	2018-02-08 21:24:29	测试更新	本人一直从事后端开发，E
---	---	---	---------------------	---------------------	------	--------------

删除

```

<!--删除-->
<delete id="deleteTopic" parameterType="Integer" useGeneratedKeys="true"> <!--useGeneratedKeys true 自动获取主键-->
    DELETE FROM Topic WHERE id=#{value}
</delete>

```

测试

```

/**
 * 删除操作
 */
@Test
public void TestDelete(){
    SqlSession sqlSession=null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        //删除操作
        int num = sqlSession.delete("topic.deleteTopic",32);
        //提交事务
        sqlSession.commit();
    }
}

```

```

        System.out.println(num);

    }catch (Exception e){
        //出现问题，数据回滚
        if(sqlSession !=null){
            sqlSession.rollback();
        }
        e.printStackTrace();
    }finally {
        //关闭资源
        mybatisUtils.Close(sqlSession);
    }
}

```

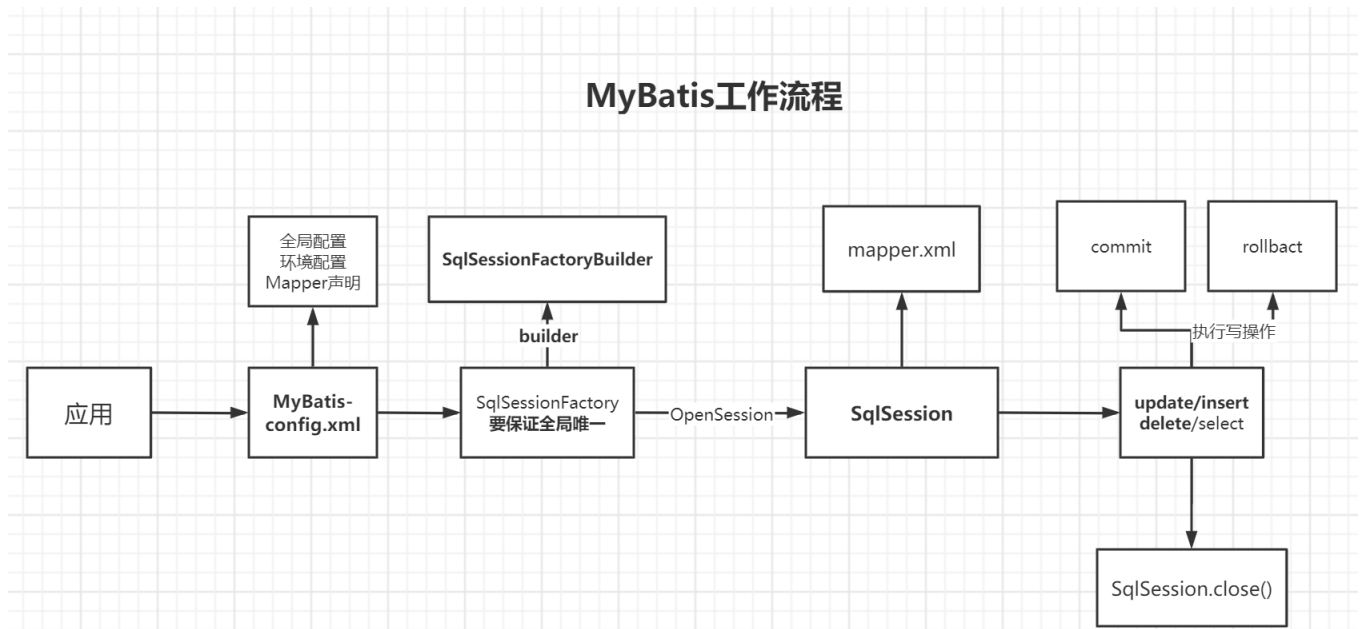
Mybatis预防SQL注入攻击

- SQL注入攻击
- SQL注入是指攻击者利用SQL漏洞，绕过系统约束，越权获取数据的攻击方式

Mybatis两种传值方式

- \${} 文本替换，未经过任何处理对SQL文本替换
- #{}预编译传值，使用预编译传值可以预防sql注入

MyBatis工作流程



#

Mybatis高级特性

一、Mybatis日志管理

加入日志包依赖

```

<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>

```

编写logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<configuration>
    <appender name="console" class="cn.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>
<!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->
[%thread] %d{HH:mm:ss:SSS} %-5level %logger{35} -%msg -%n
            </pattern>
        </encoder>
    </appender>
    <root level="DEBUG">
        <appender-ref ref="console"/>
    </root>
</configuration>

```

配置表示把>=DEBUG级别的日志都输出到控制台

MyBatis动态SQL

在mapper.xml添加

```

<!--动态sql-->
<select id="selecetDynamic" parameterType="java.util.Map" resultType="xyz.herther.entity.Topic" >
    SELECT * FROM topic
    WHERE
    1=1
    <if test="userId !=null">
        and user_id =#{userId}
    </if>
    <if test="click !=null">
        and click >#{click}
    </if>
</select>

```

```

/**
 * 动态查询
 */
@Test
public void TestDynamiSelect(){
    SqlSession sqlSession=null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        Map map =new HashMap();
        map.put("userId",4);
        map.put("click",44);
        List<Topic> topic = sqlSession.selectList("topic.selecetDynamic", map);
        for (Topic t:topic) {
            System.out.println(t.toString());
        }
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        //关闭资源
        mybatisUtils.Close(sqlSession);
    }
}

```



```

MON FEB 01 15:28:15 CST 2021 WARN: Establishing SSL connection without server's identity verification is not recommended. According
15:28:16.094 [main] DEBUG org.apache.ibatis.datasource.pooled.PooledDataSource - Created connection 1634132079.
15:28:16.094 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.
15:28:16.102 [main] DEBUG topic.selecetDynamic - ==> Preparing: SELECT * FROM topic WHERE 1=1 and user_id =? and click > ?
15:28:16.133 [main] DEBUG topic.selecetDynamic - ==> Parameters: 4(Integer), 44(Integer)
15:28:16.150 [main] DEBUG topic.selecetDynamic - <== Total: 5
topic{id=7, userId=4, createTime=Sat Jan 20 20:13:57 CST 2018, updateTime=Fri Feb 09 15:58:52 CST 2018, title='JDK 9 里的 JDK HTTP
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create("http://openjdk.java.net/"))
    .build();
client.sendAsync(request, asString())
    .thenApply(HttpResponse::body)
    .thenAccept(System.out::println)
    .join();
还可以指定使用 HTTP/2:

HttpClient client = HttpClient.newBuilder()
    .version(Version.HTTP_2)
    .followRedirects(Redirect.SAME_PROTOCOL)

```

改动一下mapper

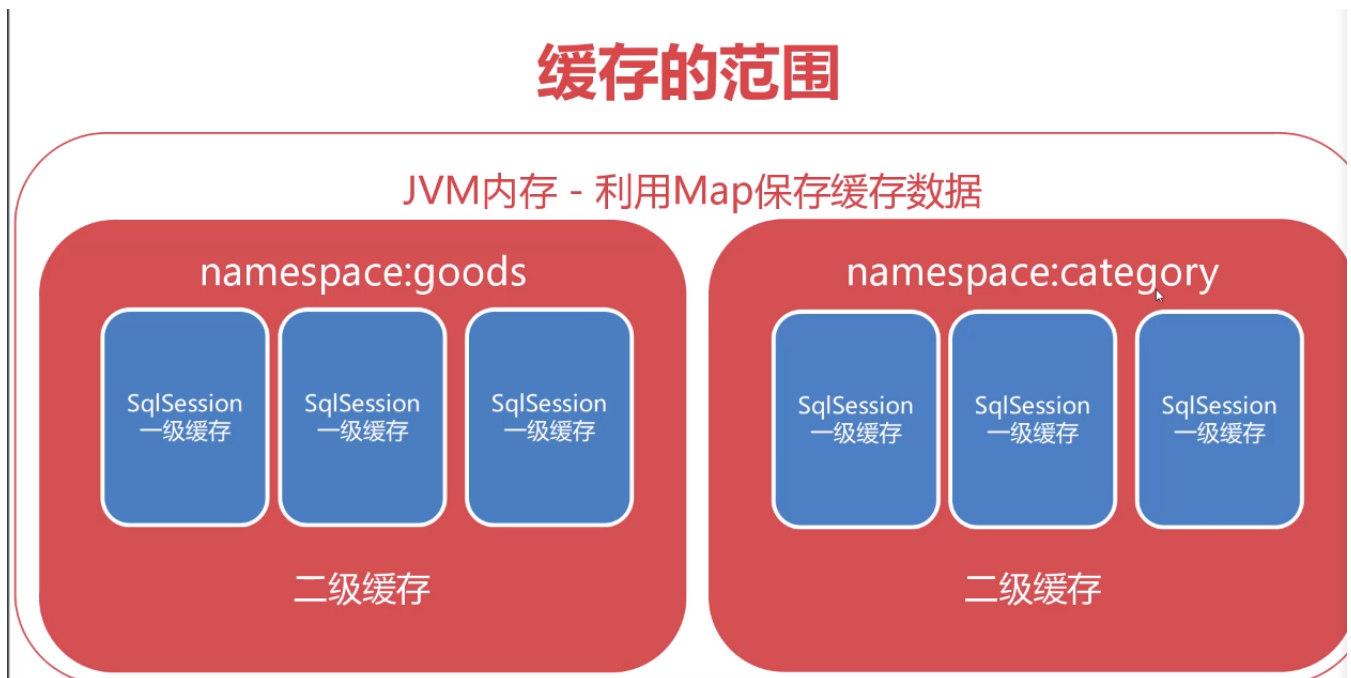
```

<!--动态sql-->
<select id="selecetDynamic" parameterType="java.util.Map" resultType="xyz.herther.entity.Topic" >
    SELECT * FROM topic
    <where>
        <if test="userId !=null">
            and user_id =#{userId}
        </if>
        <if test="click !=null">
            and click >#{click}
        </if>
    </where>
</select>

```

Mybatis二级缓存（把数据放到缓存中）

1. 一级缓存默认开启，缓存范围SqlSession会话
2. 二级缓存手动开启，属于范围mapper 中 namespace



二级缓存运行规则

- 二级开启后默认所有查询均使用缓存
- 写操作commit提交时对namespace缓存进行强制清空（数据库一致性）
- 配置useCache=false 可以不用缓存（一般不建议用于数据量的list）
- 配置flushCache=true代表强制清空缓存（sql 执行之后立马清空缓存，和commit提交事务差不多）

如何开启二级缓存

在mapper添加如下代码：

```
<!--开启二级缓存，
    eviction是缓存的清楚策略 当缓存对象达到上限则会自动触发对应的算法，来清除缓存对象
    1、LRU是最近不久未使用：移除最长时间不适用的对象
        LFU最近最少未使用：移除最少不适用的对象
    2、FIFO -先进先出，按对象进入的缓存顺序来移除
    3、SOFT -软引用：移除基于垃圾收集器状态和软引用的规则对象
    4、WEAK -弱引用：更积极的移除基于垃圾收集器状态和弱引用规则对象

    flushInterval：代表时间间隔多长时间清空缓存对象。，单位毫秒、600000 = 10分钟
    size：缓存对象的长度、最大多少对象
    readOnly: true ：代表返回只读缓存，每次从缓存取出缓存对象，这种效率高
        false：代表每次取出的缓存对象的”副本“，每次取出的对象不同，这种效率高
-->
<cache eviction="LRU" flushInterval="600000" size="500" readOnly="true"/>
```

MyBatis多表 **级联** 查询(与多表查询不同)

一对多的查询

新建一个实体类

```
public class Replay {
    private Long id;
    private Integer topicId;
    private Integer replyUserId;
    private String content;
    private Date createTime;
    private Date updateTime;
    private String device;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Integer getTopicId() {
        return topicId;
    }

    public void setTopicId(Integer topicId) {
        this.topicId = topicId;
    }

    public Integer getReplyUserId() {
        return replyUserId;
    }

    public void setReplyUserId(Integer replyUserId) {
        this.replyUserId = replyUserId;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getCreateTime() {
        return createTime;
    }

    public void setCreateTime(Date createTime) {
        this.createTime = createTime;
    }

    public Date getUpdateTime() {
```

```

        return updateTime;
    }

    public void setUpdateTime(Date updateTime) {
        this.updateTime = updateTime;
    }

    public String getDevice() {
        return device;
    }

    public void setDevice(String device) {
        this.device = device;
    }
}

```

在Topic实体类中加入

```
private List<Replay> replay; //评论表
```

在mappers文件夹下创建一个reply.xml

```

<?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="reply">
    <select id="selectReplyByTopId" parameterType="Integer" resultType="xyz.herther.entity.Replay">
        select * from reply where topic_id = #{value}
    </select>
</mapper>

```

在topic.xml中新增

```

<!--级联查询-->
<resultMap id="cascade" type="xyz.herther.entity.Topic">
    <id property="id" column="id"></id> <!--主键 当前topic的-->
    <!--
    collection 会变量查询出的集合拿到id 去执行调用reply.xml的语句
    property: 就是在topic实体类中新增 List集合中实体名
    select:    就是调用 reply.xml中的selectReplyByTopId方法查询
    column:    传过去的参数为什么字段（也就是reply表中的外键）
    -->
    <collection property="reply" select="reply.selectReplyByTopId" column="id"/>
</resultMap>
<select id="SelectOneMarny" resultMap="cascade">
    select * from topic limit 0,10;
</select>

```

在mybatis-config.xml中添加mapper映射文件

```

<mappers>
    <!--resource mapper的路径-->
    <mapper resource="mappers/topic.xml"/>
    <mapper resource="mappers/reply.xml"/>
</mappers>

```

编写测试用例

```

/**
 * 级联查询
 */
@Test
public void TestSelectOneTopic(){
    SqlSession sqlSession=null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        List<Topic> topic = sqlSession.selectList("topic.SelectOneMarny");
    }
}

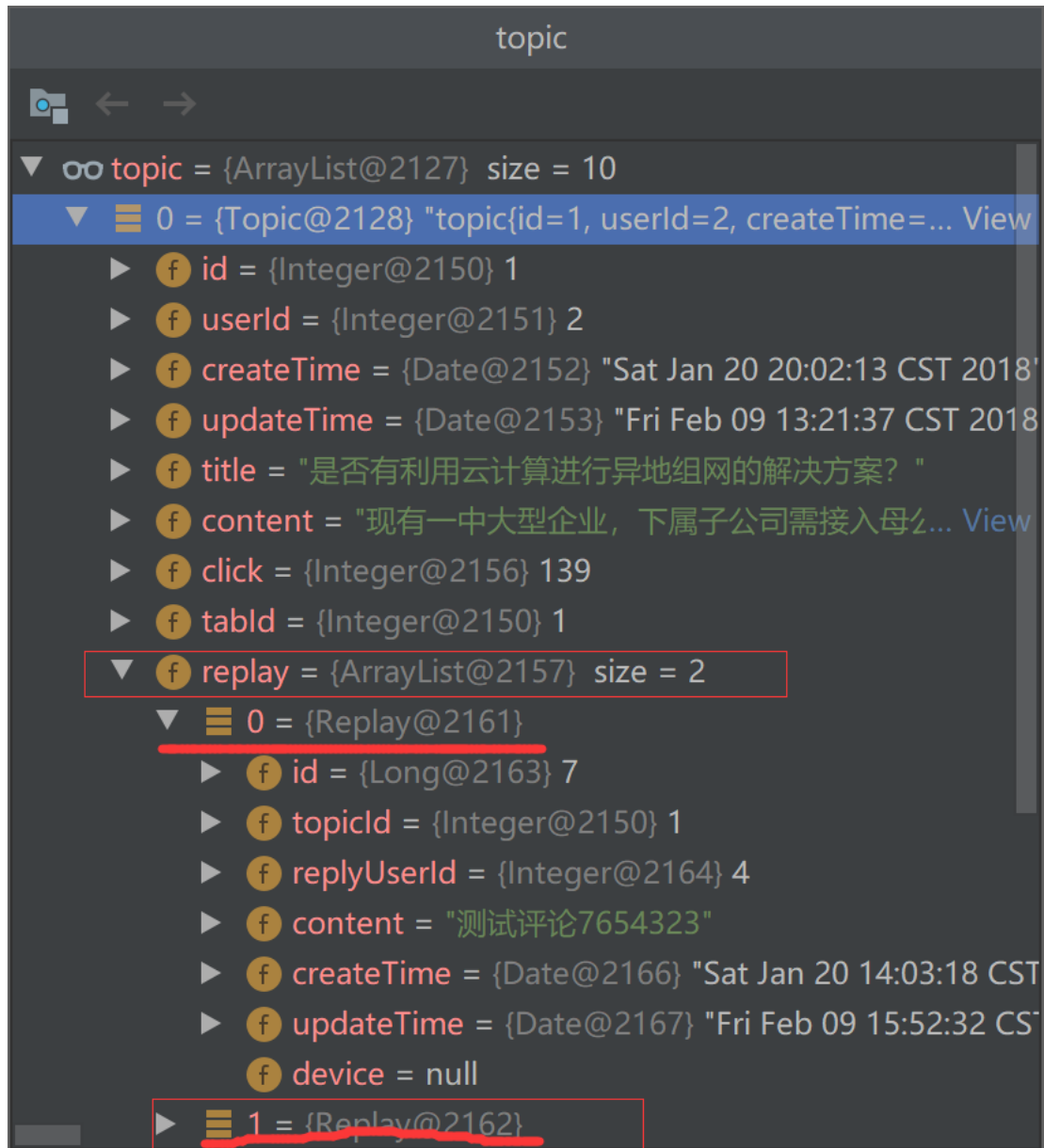
```

```

    for (Topic t:topic) {
        System.out.println(t.getTitle());
    }
} catch (Exception e){
    e.printStackTrace();
}finally {
    //关闭资源
    mybatisUtils.Close(sqlSession);
}
}

```

测试结果：成功



多对一的级联查询

Replay.xml新增sql语句

```

<!--多对一查询-->
<resultMap id="MoreSelect" type="xyz.herther.entity.Replay">
    <id property="id" column="id"></id>
    <!--
    多对一
    association: 匹配查询
    property: 为reply实体新增 实体属性
    column: 传值过去另外一个语句的参数
    select: 另外一个要执行的语句
    -->
    <association property="topic" column="topic_id" select="topic.selectById"></association>
</resultMap>
<select id="selectManyToOne" resultMap="MoreSelect">

```

```
select * from reply limit 0,10
</select>
```

Replay实体新增一条

```
private Topic topic; //topic实体
```

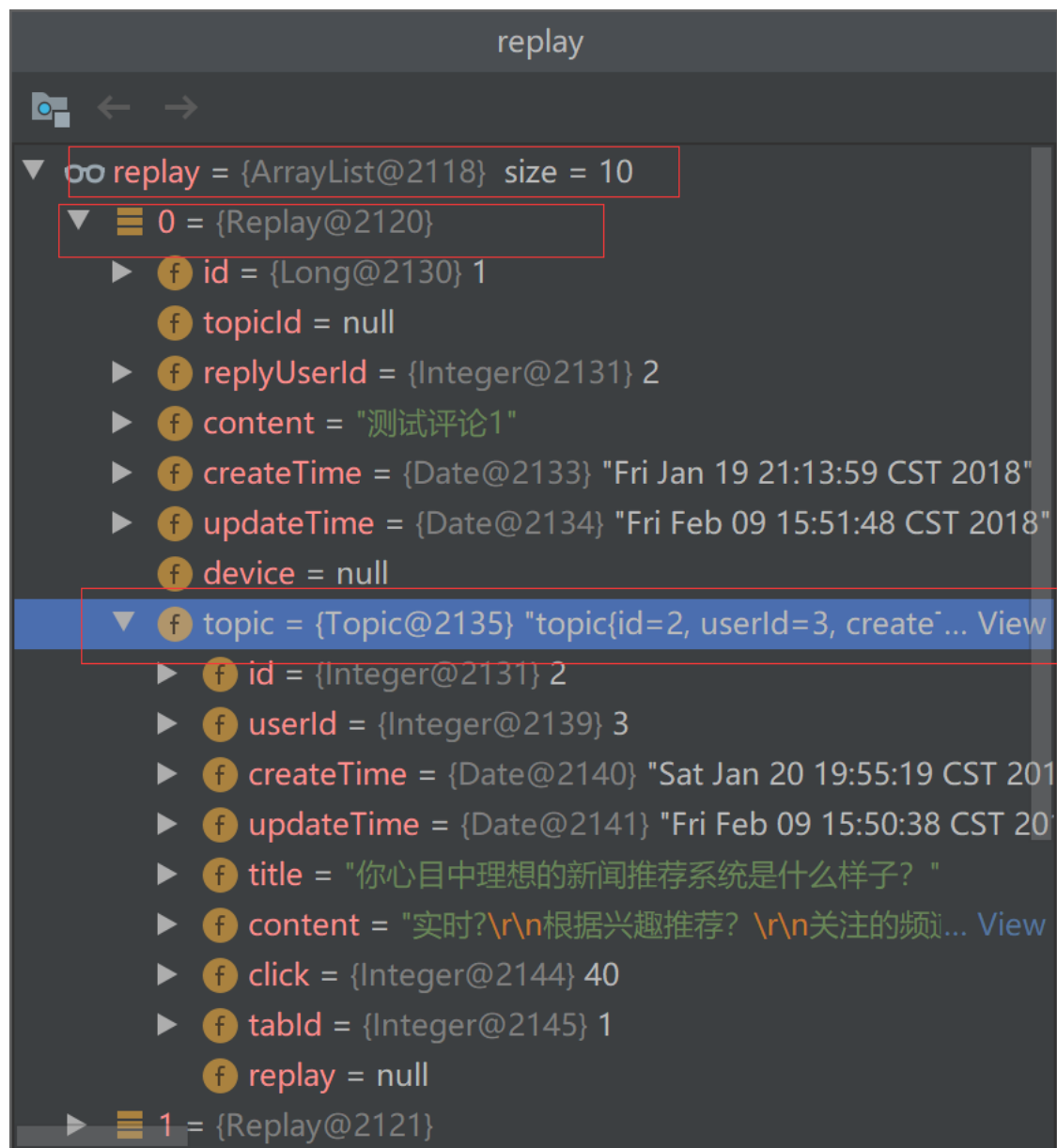
topic.xml引用这条sql语句

```
<!--parameterType 为传参的类型-->
<select id="selectById" parameterType="Integer" resultType="xyz.herther.entity.Topic">
    <!--#{value} 为传进来参数的值-->
    select * from topic where id = #{value}
</select>
```

测试

```
/**
 * 级联查询 多对一
 */
@Test
public void TestSelectManayTopic(){
    SqlSession sqlSession=null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        List<Replay> replay = sqlSession.selectList("reply.selectManyToOne");
        for (Replay t:replay) {
            System.out.println(t);
        }
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        //关闭资源
        mybatisUtils.Close(sqlSession);
    }
}
```

测试结果：成功

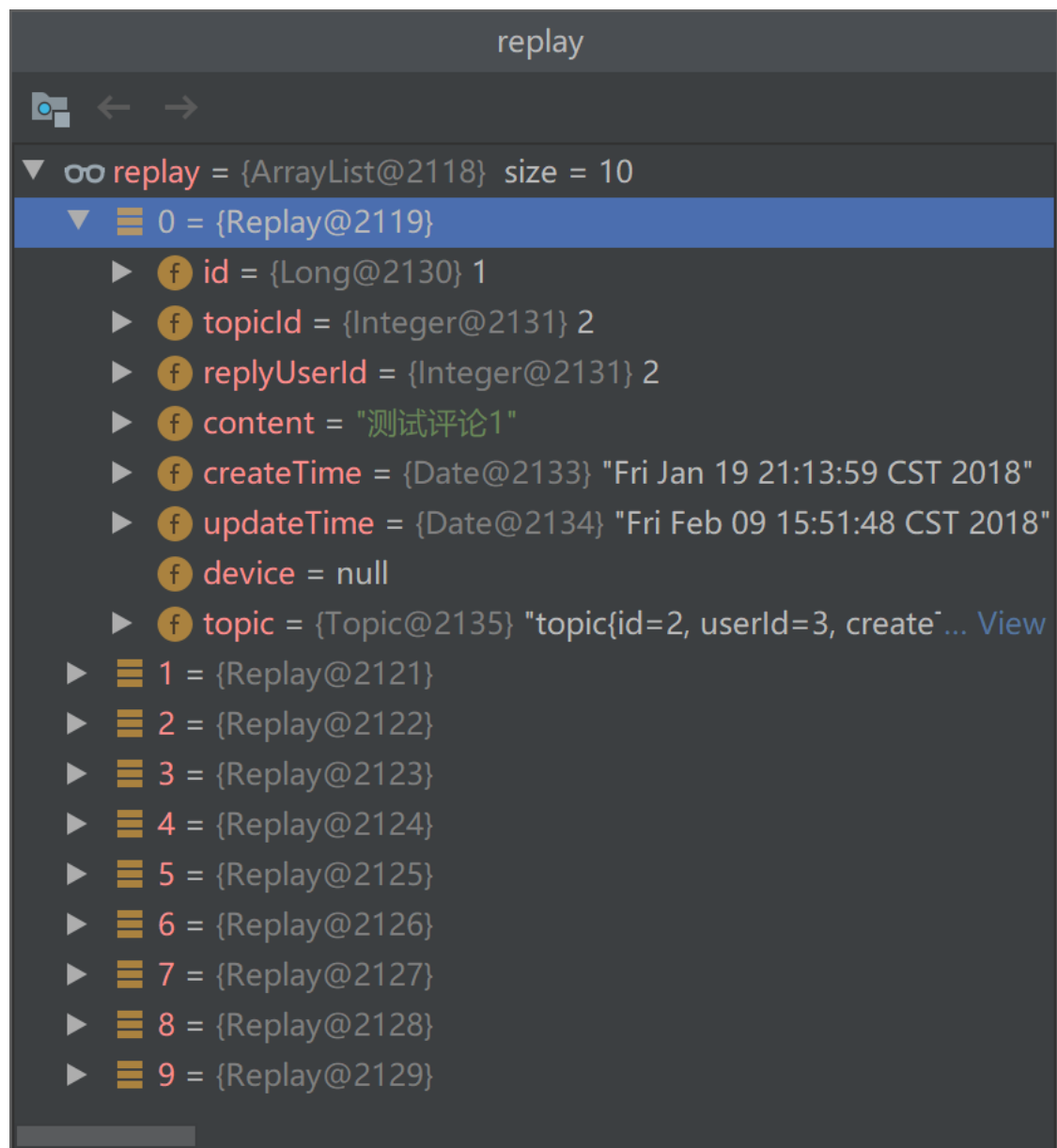


上图中topicId为空，是因为Mybatis的机制

解决方法：

在replay.xml中新增

```
<result property="topicId" column="topic_id"></result>
```



Mybatis分页插件 (PageHelper)

PageHelper使用流程

- maven引入pageHelper与jsqlparser
- mybatis-config.xml增加Plugin配置
- 代码中使用PageHelper.startPage()自动分页

1、加入pom依赖

```
<dependency>
  <groupId>com.github.pagehelper</groupId>
  <artifactId>pagehelper</artifactId>
  <version>5.1.10</version>
</dependency>
<dependency>
  <groupId>com.github.jsqlparser</groupId>
  <artifactId>jsqlparser</artifactId>
  <version>2.0</version>
</dependency>
```

2、在Mybatis-config.xml中添加插件配置

```
<plugins>
  <plugin interceptor="com.github.pagehelper.PageInterceptor">
    <!--helperDialect: 分页插件会自动检测当前的数据库链接，自动选择合适的分页方式。
    你可以配置helperDialect属性来指定分页插件使用哪种方言。配置时，可以使用下面的缩写值：
    oracle,mysql,mariadb,sqlite,hsqldb,postgresql,db2,sqlserver,informix,h2,sqlserver2012,derby-->
```

```

        特别注意：使用 SqlServer2012 数据库时，需要手动指定为 sqlserver2012，
        否则会使用 SqlServer2005 的方式进行分页。

        你也可以实现 AbstractHelperDialect，
        然后配置该属性为实现类的全限定名称即可使用自定义的实现方法。-->

        <property name="helperDialect" value="mysql"/>
        <!--reasonable :分页合理化参数，默认值为false。
        当该参数设置为 true 时，pageNum<=0 时会查询第一页，
        pageNum>pages（超过总数时），会查询最后一页。
        默认false 时，直接根据参数进行查询。-->
        <property name="reasonable" value="true"/>
    </plugin>
</plugins>

```

3、编写sql语句

```

<select id="SelectPageHelp" resultType="xyz.herther.entity.Topic">
    select * from topic where click > 10
</select>

```

4、测试

```

/**
 * 分页pagehelper
 */
@Test
public void TestSelectPageHelp(){
    SqlSession sqlSession=null;
    try {
        sqlSession = mybatisUtils.OpenSession();
        //分页，从第二页开始查询，一页五条
        PageHelper.startPage(2,5);
        //返回page集合
        Page<Topic> page = (Page)sqlSession.selectList("topic.SelectPageHelp");
        System.out.println("总页数: "+page.getPages());
        System.out.println("总记录数: "+page.getTotal());
        System.out.println("开始行号: "+page.getStartRow());
        System.out.println("结束行号: "+page.getEndRow());
        System.out.println("当前页码: "+page.getPageNum());
        List<Topic> result = page.getResult();
        for (Topic i: result) {
            System.out.println("标题: "+i.getTitle());
        }
    } catch (Exception e){
        e.printStackTrace();
    } finally {
        //关闭资源
        mybatisUtils.Close(sqlSession);
    }
}

```

5、测试结果

```

总页数: 5
总记录数: 24
开始行号:5
结束行号: 10
当前页码: 2
标题: 使用 Nginx 自建 CDN，关于回源问题。
标题: JDK 9 里的 JDK HTTP Client 的写法
标题: 数据库怎么最简单实现“栈”结构存储？
标题: 腾讯云能不能别给我发优惠券短信了
标题: 应用程序在服务器上创建文件目录权限 0777，会有风险吗？

```

执行过程中的sql


```

transaction.jdbc.JdbcTransaction - Setting autocommit to false on JDBC Connection
elp_COUNT - ==> Preparing: SELECT count(0) FROM topic WHERE click > 10
elp_COUNT - ==> Parameters:
elp_COUNT - <==          Total: 1
Ratio [topic]: 0.0
elp - ==> Preparing: select * from topic where click > 10 LIMIT ?, ?
elp - ==> Parameters: 5(Integer), 5(Integer)
elp - <==          Total: 5

```

Mybatis整合C3P0连接池

1、引入pom依赖

```

<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.4</version>
</dependency>

```

2、编写C3P0与Mybatis兼容使用数据源工厂类

```

/**
 * C3P0与Mybatis兼容使用数据源工厂类
 */
public class C3P0DataSourceFactory extends UnpooledDataSourceFactory {
    public C3P0DataSourceFactory(){
        this.dataSource = new ComboPooledDataSource();
    }
}

```

3、修改mybatis-config.xml配置

```

<dataSource type="xyz.herther.datasources.C3P0DataSourceFactory"> <!--C3P0与Mybatis兼容使用数据源工厂类-->
    <property name="driverClass" value="com.mysql.jdbc.Driver"/>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/admin?useUnicode=true&characterEncoding=utf8"/>
    <property name="user" value="root"/>
    <property name="password" value="123456"/>
    <!--初始连接池连接数量-->
    <property name="initialPoolSize" value="5"/>
    <!--最大连接池连接数量-->
    <property name="maxPoolSize" value="20"/>
    <!--最少连接池连接数量-->
    <property name="minPoolSize" value="5"/>
</dataSource>

```

3、测试结果

```

20:12:34.336 [main] DEBUG com.mchange.v2.cfg.MConfig - The configuration file for resource identifie
20:12:34.908 [main] INFO com.mchange.v2.c3p0.C3P0Registry - Initializing c3p0-0.9.5.4 [built 23-Marc
20:12:34.968 [main] DEBUG com.mchange.v2.c3p0.management.DynamicPooledDataSourceManagerMBean - MBean
20:12:35.001 [main] DEBUG com.mchange.v2.c3p0.management.DynamicPooledDataSourceManagerMBean - MBean
20:12:35.001 [main] DEBUG com.mchange.v2.c3p0.management.DynamicPooledDataSourceManagerMBean - MBean
20:12:35.204 [main] DEBUG SQL_CACHE - Cache Hit Ratio [SQL_CACHE]: 0.0
20:12:35.280 [main] DEBUG topic - Cache Hit Ratio [topic]: 0.0
20:12:35.289 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Opening JDBC Connecti
20:12:35.347 [main] INFO com.mchange.v2.c3p0.impl.AbstractPooledDataSource - Initializing c3p0 p
20:12:35.373 [main] DEBUG com.mchange.v2.cfg.MConfig - The configuration file for resource identifie
20:12:35.373 [main] DEBUG com.mchange.v2.cfg.MConfig - The configuration file for resource identifie
20:12:35.373 [main] DEBUG com.mchange.v2.cfg.MConfig - The configuration file for resource identifie
20:12:35.373 [main] DEBUG com.mchange.v2.cfg.MConfig - The configuration file for resource identifie
20:12:35.376 [main] DEBUG com.mchange.v2.resourcepool.BasicResourcePool - com.mchange.v2.resourcepoo
20:12:35.377 [main] DEBUG com.mchange.v2.c3p0.impl.C3P0PooledConnectionPoolManager - Created new poo
20:12:35.377 [main] DEBUG com.mchange.v2.resourcepool.BasicResourcePool - acquire test -- pool size:
20:12:35.377 [main] DEBUG com.mchange.v2.resourcepool.BasicResourcePool - awaitAvailable(): [unknown

```

Mybatis批处理

在mapper新增 一条sql语句

```

<!--批处理-->
<insert id="batchInsert" parameterType="java.util.List">
    INSERT INTO topic(user_id,create_time,update_time,title,content,click,tab_id)
    VALUES
    <!--collection遍历的数据源强制要求是list item遍历的变量, index下标, separator以什么分割 逗号 -->
    <foreach collection="list" item="item" index="index" separator=",">
        ({item.userId},{item.createTime},{item.updateTime},{item.title},{item.content},{item.click},{item.tabId})
    </foreach>
</insert>

```

编写测试类

```

/**
 * 批量新增数据操作
 */
@Test
public void TestbatchInsert(){
    SqlSession sqlSession = null;
    try {
        sqlSession=mybatisUtils.OpenSession();
        long Starttime = new Date().getTime();
        ArrayList list = new ArrayList();
        for (int i=0; i<1000; i++){
            Topic topic = new Topic();
            topic.setClick(44);
            topic.setContent("测试内容"+i);
            topic.setCreateTime(new Date());
            topic.setTabId(1);
            topic.setTitle("测试标题"+i);
            topic.setUpdateTime(new Date());
            topic.setUserId(4);
            list.add(topic);
        }
        //返回影响的行数
        int num = sqlSession.insert("topic.batchInsert", list);
        //提交事务
        sqlSession.commit();
        long Endtime = new Date().getTime();
        System.out.println("成功插入: "+num+"条,所用时间: "+(Starttime-Endtime)+"毫秒");
    }catch (Exception e){
        //出现异常、事务回滚
        if(sqlSession !=null){
            sqlSession.rollback();
        }
    }finally {
        mybatisUtils.Close(sqlSession);
    }
}

```

```
}  
}
```

测试成功

```
20:56:50.455 [main] DEBUG topic.batchInsert - ==> Preparing: INSERT INTO topic(user_id,create_time,update_time,click_count) VALUES(?, ?, ?, ?)  
20:56:50.457 [main] DEBUG topic.batchInsert - ==> Parameters: 4(Integer), 2021-02-02 20:56:49.847(Timestamp), 2021-02-02 20:56:50.455(Timestamp), 1(Decimal)  
20:56:50.495 [main] DEBUG topic.batchInsert - <== Updates: 1000  
20:56:50.495 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Committing JDBC Connection [com.mysql.jdbc.Connection@10000000]  
成功插入: 1000条,所用时间: -648毫秒
```

执行的效率非常高

批量删除 (`DELETE FROM 表名 where id in (xx,xxx,xxx,)`)

编写mapper.xml

```
<!--批处理删除-->  
<delete id="batchDelete" parameterType="java.util.List">  
    DELETE FROM topic WHERE title in  
    <!--collection遍历的数据源强制要求是list item遍历的变量, index下标, separator以什么分割 逗号“ -->  
    <foreach collection="list" item="item" index="index" open="(" close=")" separator=",">  
        #{item}  
    </foreach>  
</delete>
```

测试类


```
/**  
 * 批量删除数据操作  
 */  
@Test  
public void TestbatchDelete(){  
    SqlSession sqlSession = null;  
    try {  
        sqlSession=mybatisUtils.OpenSession();  
        long Starttime = new Date().getTime();  
        ArrayList list = new ArrayList();  
        for (int i=0; i<1000; i++){  
            list.add("测试标题"+i);  
        }  
        //返回影响的行数  
        int num = sqlSession.insert("topic.batchDelete", list);  
        //提交事务  
        sqlSession.commit();  
        long Endtime = new Date().getTime();  
        System.out.println("成功删除: "+num+"条,所用时间: "+(Starttime-Endtime)+"毫秒");  
    }catch (Exception e){  
        //出现异常、事务回滚  
        if(sqlSession !=null){  
            sqlSession.rollback();  
        }  
    }finally {  
        mybatisUtils.Close(sqlSession);  
    }  
}
```

测试成功

```
21:10:55.445 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Committing JDBC Connection [com.mysql.jdbc.Connection@10000000]  
成功删除: 1000条,所用时间: -580毫秒  
21:10:55.446 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.Connection@10000000]
```

Mybatis注解开发 (替代mapper.xml中的标签, 简化配置过程)

常用的Mybatis注解

注解	对应的xml	说明
@Insert		新增SQL
@Update		更新SQL
@Delete		删除SQL
@Select		查询SQL
@Param	-----	参数映射
@Resluts		结果映射
@Reslut		字段映射

在原有的工程目录中删除mapper.xml，以及mybatis-config.xml中的mappers

编写Dao接口来用来 替代mapper.xml

1、注解查询

编写TopicDao接口

```
public interface TopicDao {  
    @Select(" select * from topic limit 0,#{limit}") //语句中写入与mapper.xml一样的sql语句  
    List<Topic> SelectAlllimit(@Param("limit") int limit); //@Param 对应sql语句中的传入参数  
  
}
```

Mybatis-config.xml配置加入

```
<!--配置映射文件-->  
<mappers>  
    <!--1、老样子一个个引入-->  
<!--    <mapper class="xyz.herther.dao.TopicDao"/>-->  
    <!--2、全部一次导入引用-->  
    <package name="xyz.herther.dao"/>  
</mappers>
```

测试类

```
/**  
 * 注解测试查询 limi参数  
 */  
@Test  
public void TestNotSelect(){  
    SqlSession sqlSession=null;  
    try {  
        sqlSession = mybatisUtils.OpenSession();  
        //传入那个Dao (后台自动实现, TopicDao接口)  
        TopicDao topicDao = sqlSession.getMapper(TopicDao.class);  
        List<Topic> topics = topicDao.SelectAlllimit(10);  
        //打印查询出来的长度  
        System.out.println(topics.size());  
    }catch (Exception E){  
        E.printStackTrace();  
    }finally {  
        mybatisUtils.Close(sqlSession);  
    }  
}
```

测试结果

```
Tue Feb 02 22:11:21 CST 2021 WARN: Establishing SSL connection without server's identity verification is not recommended.  
22:11:21.822 [main] DEBUG xyz.herther.dao.TopicDao.SelectAlllimit - ==> Preparing: select * from topic limit 0,?  
22:11:21.850 [main] DEBUG xyz.herther.dao.TopicDao.SelectAlllimit - ==> Parameters: 10(Integer)  
22:11:21.873 [main] DEBUG xyz.herther.dao.TopicDao.SelectAlllimit - <==          Total: 10  
10
```

2、注解插入

编写Dao

```
@Insert("INSERT INTO topic(user_id,create_time,update_time,title,content,click,tab_id) VALUES ({userId},{createTime},{updateTime},{title},{content},{click},{tabId})")

@SelectKey(statement = "SELECT last_insert_id()",resultType = Integer.class, keyProperty = "id",before = false)
//statement 查询id 的语句  resultType返回的类型  keyProperty返回对应实体类中那个字段, before true 在语句插入语句之前 false 在语句之后
int InsertNot(Topic topic);
```

编写测试类

```
/**
 * 注解插入
 */
@Test
public void TestNoteInsert(){
    SqlSession sqlSession =null;
    try {
        sqlSession=mybatisUtils.OpenSession();
        Topic topic = new Topic();
        topic.setClick(44);
        topic.setContent("测试内容eeee");
        topic.setCreateTime(new Date());
        topic.setTabId(1);
        topic.setTitle("测试标题eeee");
        topic.setUpdateTime(new Date());
        topic.setUserId(4);
        TopicDao topicDao = sqlSession.getMapper(TopicDao.class);
        //返回影响的行数
        int num = topicDao.InsertNot(topic);
        //提交事务
        sqlSession.commit();
        System.out.println("插入后的id为: "+topic.getId());
    }catch (Exception e){
        //出现异常、事务回滚
        if(sqlSession !=null){
            sqlSession.rollback();
        }
    }finally {
        mybatisUtils.Close(sqlSession);
    }
}
```

测试结果

```
22:25:04.346 [main] DEBUG xyz.herther.dao.TopicDao.InsertNot - <==    Updates: 1
22:25:04.350 [main] DEBUG xyz.herther.dao.TopicDao.InsertNot!selectKey - ==>   Preparing: SELECT last_insert_id()
22:25:04.351 [main] DEBUG xyz.herther.dao.TopicDao.InsertNot!selectKey - ==> Parameters:
22:25:04.365 [main] DEBUG xyz.herther.dao.TopicDao.InsertNot!selectKey - <==    Total: 1
22:25:04.370 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Committing JDBC Connection [com.mchange.v2.c3p0.ComboPooledDataSource@73913251]
插入后的id为: 4045
22:25:20.785 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mchange.v2.c3p0.ComboPooledDataSource@73913251]
22:25:20.785 [main] DEBUG org.apache.ibatis.transaction.jdbc.JdbcTransaction - Closing JDBC Connection [com.mchange.v2.c3p0.ComboPooledDataSource@73913251]
```