Neuedu

东软睿道内部公开
文件编号： D000-

Mybatis框架技术

版本：3.6.0

第4章 Mybatis关联查询与缓存配置

本章教学目标

- 了解Mybatis的一级缓存；
- 了解Ehcache第三方缓存框架；
- 理解Mybatis的延迟加载；
- 掌握Mybatis的关联查询（1对1、1对多、多对多）；
- 掌握Mybatis的分页查询；
- 掌握Mybatis的二级缓存；

本章教学内容

Neuedu

节	知识点	掌握程度	难易程度	教学形式	对应在线微课
Mybatis查询	一对一查询	掌握		线下	一对一查询
	一对一关联查询	掌握	难	线下	一对一关联查询
	一对多关联查询	掌握	难	线下	一对多关联查询
	多对多关联查询	掌握	难	线下	多对多关联查询
	分页查询	掌握	难	线下	分页查询
延迟加载	延迟加载	理解	难	线下	延迟加载
Mybatis查询缓存	一级缓存	了解		线下	一级缓存
	二级缓存	掌握		线下	二级缓存
	Ehcache第三方缓存框架	了解	难	线下	Ehcache第三方缓存框架
本章实战项目任务实现	实战项目任务实现	掌握	难	线下	实战项目任务实现

本章实战项目任务

- 通过本章内容的学习，使用关联查询、分页技术和缓存策略，完成《跨境电商系统》所有用户关联的订单及订单明细的查询
  - 使用二级缓存和ehcahe缓存框架策略
  - 在控制台输出查询结果的第一页（每页显示2条数据）

- 运行效果在控制台输出：

```
DEBUG [main] - sysUserList.size=2
DEBUG [main] - userId = 1
DEBUG [main] - salesOrderList.size=2
DEBUG [main] - saoId = 24
DEBUG [main] - salesOrderLineItemList.size = 2
DEBUG [main] - salId=28,proId=1
DEBUG [main] - salId=27,proId=1
DEBUG [main] - saoId = 23
DEBUG [main] - salesOrderLineItemList.size = 2
DEBUG [main] - salId=26,proId=1
DEBUG [main] - salId=25,proId=1
DEBUG [main] - userId = 2
DEBUG [main] - salesOrderList.size=1
DEBUG [main] - saoId = 22
DEBUG [main] - salesOrderLineItemList.size = 1
DEBUG [main] - salId=24,proId=1
```

CONTENTS

目录

Mybatis查询

Neuedu

- MyBatis中在查询进行select映射的时候，返回类型可以用resultType，也可以用resultMap
  - resultType是直接表示返回类型的(对应着我们的model对象中的实体)
  - resultMap则是对外部ResultMap的引用(提前定义了db和model之间的映射key-->value关系)
  - resultType和resultMap不能同时存在
- 在MyBatis进行查询映射时，其实查询出来的每一个属性都是放在一个对应的Map里面的，其中键是属性名，值则是其对应的值。
  ①当提供的返回类型属性是resultType时，MyBatis会将Map里面的键值对取出赋给resultType所指定的对象对应的属性。所以其实MyBatis的每一个查询映射的返回类型都是ResultMap，只是当提供的返回类型属性是resultType的时候，MyBatis对自动的给把对应的值赋给resultType所指定对象的属性。
  ②当提供的返回类型是resultMap时，因为Map不能很好表示领域模型，就需要自己再进一步的把它转化为对应的对象，这常常在复杂查询中很有作用。

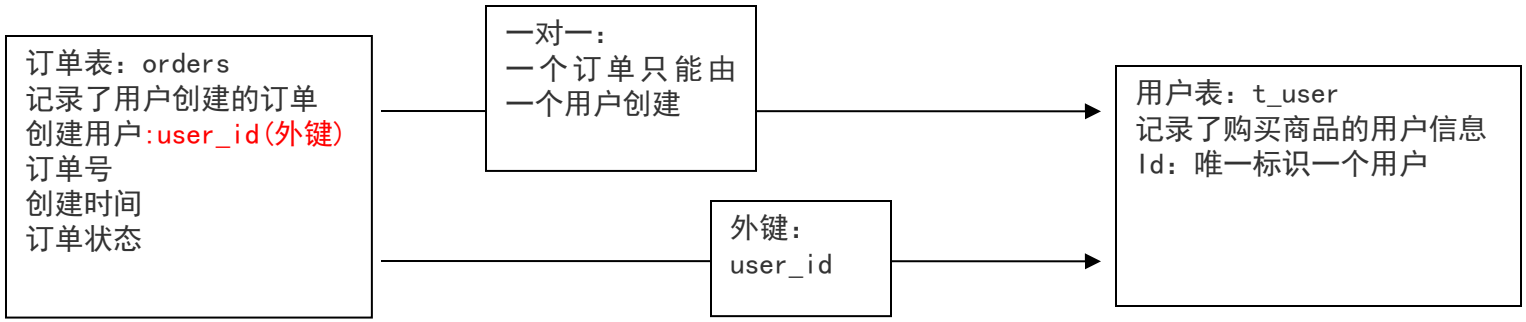
Mybatis查询

- 准备案例数据模型
  - 用户表
  - 订单表
  - 订单详情表
  - 商品表

  - 详细参考《订单商品案例数据关系图》
  - 建表脚本 参考订单商品案例-1.sql、订单商品案例-2.sql

一对一查询

- 案例：查询所有订单和用户信息。
  - 一个订单信息只会是一个人下的订单，所以从查询订单信息出发关联查询用户信息为一对一查询。如果从用户信息出发查询用户下的订单信息则为一对多查询，因为一个用户可以下多个订单。



一对一查询

Neuedu

- 实现案例：查询所有订单和用户信息。
- 实现方法：使用resultType
  - 1、定义订单信息POJO类，此类中包括了订单信息和用户信息,OrdersCustom类继承Orders类后包括了Orders类的所有字段，只需要定义用户的信息字段即可。

```
//通过此类映射订单和用户查询的结果，让此类继承包括 字段较多的pojo类
public class OrdersCustom extends Orders{

    private String username;
    private String sex;
    private String address;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getSex() {
        return sex;
    }
}
```

  - 全部代码参见：ch04-mybatis01工程

一对一查询

Neuedu

- 2、配置OrdersMapperCustom.xml

```
<!-- 查询订单关联查询用户信息 -->

<select id="findOrdersResultType" resultType="com.neuedu.pojo.OrdersCustom">
    SELECT
    orders.*,
    T_USER.username,
    T_USER.sex,
    T_USER.address
    FROM
    orders,
    T_USER
    WHERE orders.user_id = T_USER.id
</select>
```

- 3、定义OrdersMapperCustom接口方法
  - public List<OrdersCustom> findOrdersResultType() throws Exception;
- 4、测试方法：OrdersMapperCustomTest测试类中
- 全部代码参见：ch04-mybatis01工程

一对一关联查询

- MyBatis中使用association标签来解决一对一的关联查询，

```
<association property="user"  javaType="com.neuedu.pojo.User">
    <!-- id：关联查询用户的唯 一标识
    column：指定唯 一标识用户信息的列
    property：映射到user的哪个属性
     -->
    <id column="user_id" property="id"/>
    <result column="username" property="username"/>
    <result column="sex" property="sex"/>
    <result column="address" property="address"/>

</association>
```

一对一关联查询

- 实现案例：查询所有订单和用户信息。
- 实现方法：使用resultMap
  - 1、创建POJO类：User、Orders

```
public class Orders {
    private Integer id;

    private Integer userId;

    private String orderId;

    private Date createtime;

    private String note;

    //用户信息
    private User user;
```

  - 全部代码参见：ch04-mybatis01工程

一对一关联查询

Neuedu

- 2、配置OrdersMapperCustom.xml中ResultMap映射

```
<resultMap type="com.neuedu.pojo.Orders" id="OrdersUserResultMap">
    <id column="id" property="id"/>
    <result column="user_id" property="userId"/>
    <result column="orderid" property="orderId"/>
    <result column="createtime" property="createtime"/>
    <result column="note" property="note"/>

    <!-- 配置映射的关联的用户信息 -->
    <!-- association：用于映射关联查询单个对象的信息
    property：要将关联查询的用户信息映射到Orders中哪个属性
     -->
    <association property="user"  javaType="com.neuedu.pojo.User">
        <!-- id：关联查询用户的唯一标识
        column：指定唯一标识用户信息的列
        javaType：映射到user的哪个属性
         -->
        <id column="user_id" property="id"/>
        <result column="username" property="username"/>
        <result column="sex" property="sex"/>
        <result column="address" property="address"/>

    </association>
</resultMap>
```

一对一关联查询

Neuedu

- 3、配置OrdersMapperCustom.xml中select映射

```
<!-- 查询订单关联查询用户信息，使用resultmap -->
<select id="findOrdersUserResultMap" resultMap="OrdersUserResultMap">
    SELECT
    orders.*,
    T_USER.username,
    T_USER.sex,
    T_USER.address
    FROM
    orders,
    T_USER
    WHERE orders.user_id = T_USER.id
</select>
```

  - 其中resultMap属性值为第2步定义的resultMap
- 4、定义OrdersMapperCustom接口方法
  - public List<OrdersCustom> findOrdersUserResultMap() throws Exception;
- 5、测试：

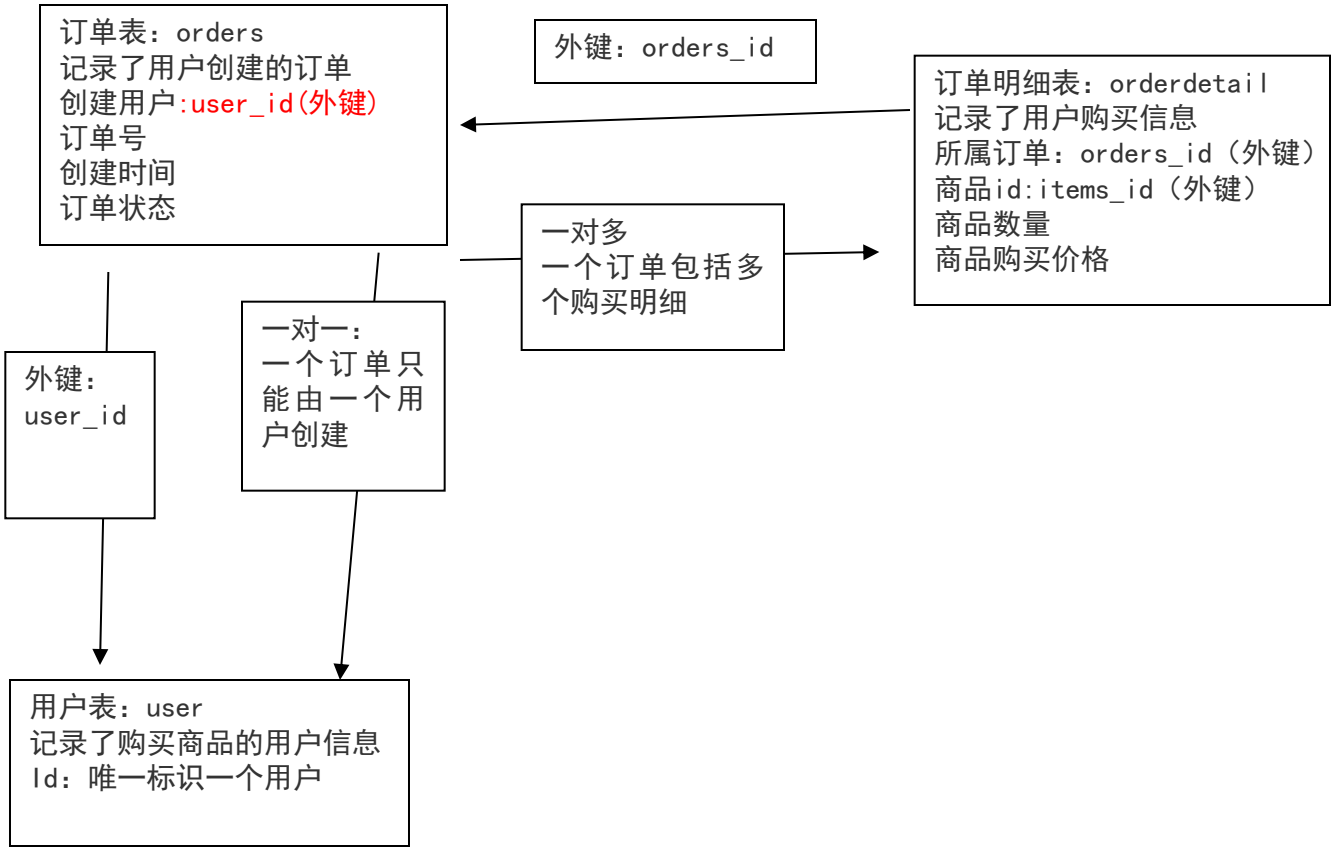
- 全部代码参见：ch04-mybatis01工程

课堂练习（20分钟）

- 1、简述Mybatis的一对一关联查询的实现
- 2、理解并独立完成授课案例

一对多关联查询

- 案例：查询所有订单信息及订单下的订单明细信息，订单信息与订单明细为一对多关系。



一对多关联查询

Neuedu

- MyBatis中使用collection标签来实现一对多关联查询
  - 使用collection完成关联查询，将关联查询信息映射到集合对象。

```
<collection property="orderdetails" ofType="com.neuedu.pojo.Orderdetail">
   <!-- id：订单明细唯 一标识
   property:要将订单明细的唯 一标识 映射到com.neuedu.pojo.Orderdetail的哪个属性
     -->
   <id column="orderdetail_id" property="id"/>
   <result column="items_id" property="itemsId"/>
   <result column="items_num" property="itemsNum"/>
   <result column="orders_id" property="ordersId"/>
</collection>
```

一对多关联查询

Neuedu

- 实现案例：查询所有订单信息及订单下的订单明细信息
  - 1、创建POJO类：User、Orders、OrderDetail

```
public class Orders {
    private Integer id;

    private Integer userId;

    private String number;

    private Date createtime;

    private String note;

    //用户信息
    private User user;

    //订单明细
    private List<Orderdetail> orderdetails;
```

  - 全部代码参见：ch04-mybatis02工程

一对多关联查询

Neuedu

- 2、配置OrdersMapperCustom.xml中ResultMap映射

```
<resultMap type="com.neuedu.pojo.Orders" id="OrdersAndOrderDetailResultMap" extends="OrdersUserResultMap">
    <!-- 订单信息 -->
    <!-- 用户信息 -->
    <!-- 使用extends继承，不用在中配置订单信息和用户信息的映射 -->

    <!-- 订单明细信息
    一个订单关联查询出了多条明细，要使用collection进行映射
    collection：对关联查询到多条记录映射到集合对象中
    property：将关联查询到多条记录映射到com.neuedu.pojo.Orders哪个属性
    ofType：指定映射到list集合属性中pojo的类型
     -->
     <collection property="orderdetails" ofType="com.neuedu.pojo.Orderdetail">
        <!-- id：订单明细唯一标识
        property:要将订单明细的唯一标识 映射到com.neuedu.pojo.Orderdetail的哪个属性
          -->
        <id column="orderdetail_id" property="id"/>
        <result column="items_id" property="itemsId"/>
        <result column="items_num" property="itemsNum"/>
        <result column="orders_id" property="ordersId"/>
     </collection>
</resultMap>
```

一对多关联查询

Neuedu

- 3、配置OrdersMapperCustom.xml中select映射

```
<!-- 查询订单关联查询用户及订单明细，使用resultmap -->
<select id="findOrdersAndOrderDetailResultMap" resultMap="OrdersAndOrderDetailResultMap">
    SELECT
      orders.*,
      USERINFO.username,
      USERINFO.sex,
      USERINFO.address,
      orderdetail.id orderdetail_id,
      orderdetail.items_id,
      orderdetail.items_num,
      orderdetail.orders_id
    FROM
      orders,
      USERINFO,
      orderdetail
    WHERE orders.user_id = userinfo.id AND orderdetail.orders_id=orders.id
</select>
```

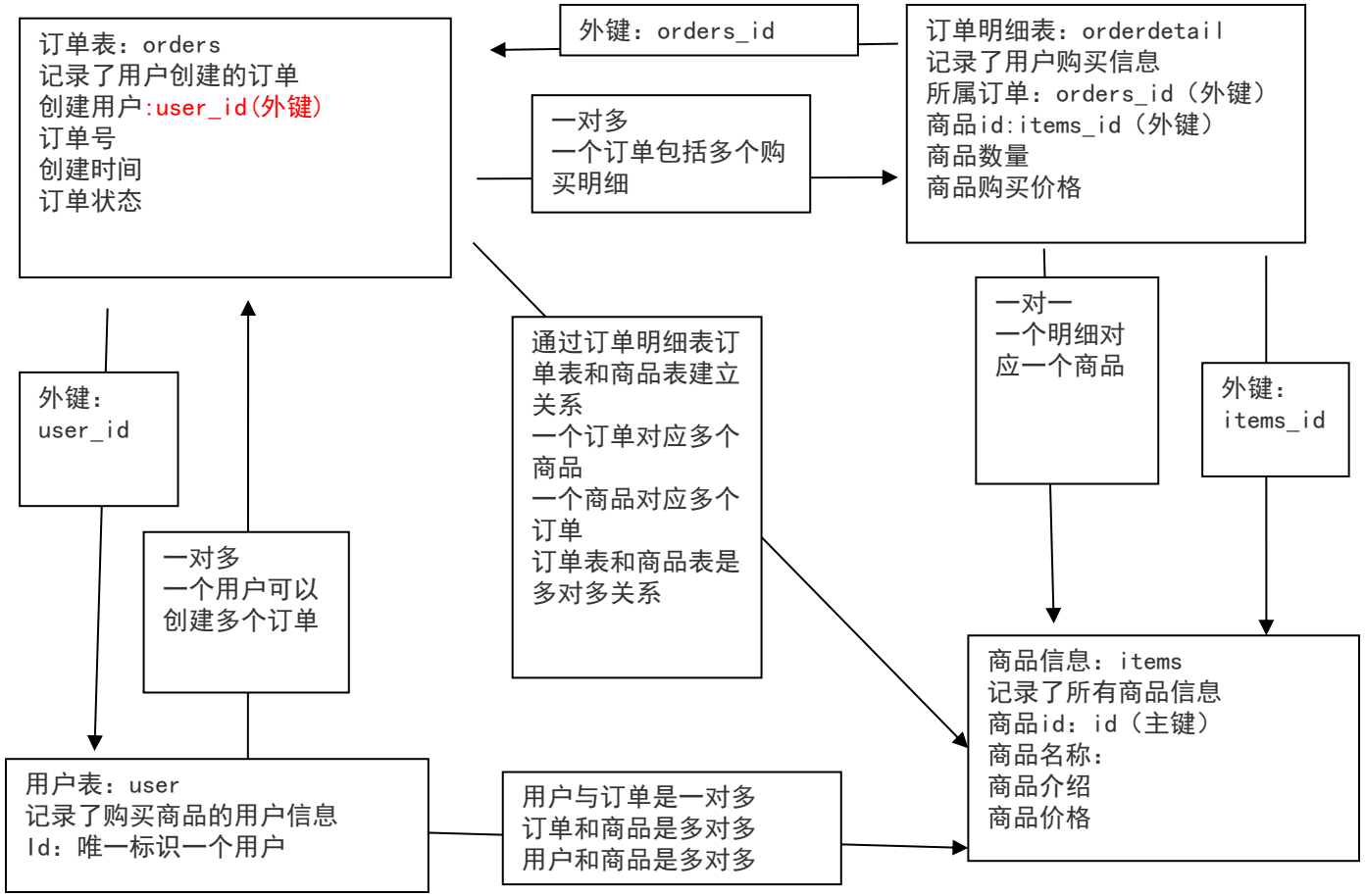
  - 其中resultMap属性值为第2步定义的resultMap
- 4、定义OrdersMapperCustom接口方法
  - public List<Orders> findOrdersAndOrderDetailResultMap() throws Exception;
- 5、测试：
- 全部代码参见：ch04-mybatis02工程

课堂练习（20分钟）

- 1、简述Mybatis的一对多关联查询的实现
- 2、理解并独立完成授课案例

多对多关联查询

- 案例：查询所有用户信息，关联查询订单及订单明细信息，订单明细信息中关联查询商品信息



多对多关联查询

Neuedu

- 实现案例：查询所有用户信息，关联查询订单及订单明细信息，订单明细信息中关联查询商品信息
  - 1、创建POJO类：User、Orders、OrderDetail、Items

```
public class User  {

    //属性名和数据库表的字段对应
    private int id;
    private String username;// 用户姓名
    private String sex;// 性别
    private Date birthday;// 生日
    private String address;// 地址

    //用户创建的订单列表
    private List<Orders> ordersList;
```

```
public class Orders {
    private Integer id;

    private Integer userId;

    private String orderId;

    private Date createtime;

    private String note;


    //订单明细
    private List<Orderdetail> orderdetails;
```

```
public class Orderdetail {
    private Integer id;

    private Integer ordersId;

    private Integer itemsId;

    private Integer itemsNum;

    //明细对应的商品信息
    private Items items;
```

  - 全部代码参见：ch04-mybatis03工程

多对多关联查询

Neuedu

- 2、配置OrdersMapperCustom.xml中ResultMap映射

```
<resultMap type="com.neuedu.pojo.User" id="UserAndItemsResultMap">
    <!-- 用户信息 -->
    <id column="user_id" property="id"/>
    <result column="username" property="username"/>
    <result column="sex" property="sex"/>
    <result column="address" property="address"/>
    <!-- 订单信息    一个用户对应多个订单，使用collection映射        -->
     <collection property="ordersList" ofType="com.neuedu.pojo.Orders">
        <id column="id" property="id"/>
        <result column="user_id" property="userId"/>
        <result column="number" property="number"/>
        <result column="createtime" property="createtime"/>
        <result column="note" property="note"/>
         <!-- 订单明细    一个订单包括 多个明细       -->
        <collection property="orderdetails" ofType="com.neuedu.pojo.Orderdetail">
                <id column="orderdetail_id" property="id"/>
                <result column="items_id" property="itemsId"/>
                <result column="items_num" property="itemsNum"/>
                <result column="orders_id" property="ordersId"/>
                <!-- 商品信息    一个订单明细对应一个商品            -->
            <association property="items" javaType="com.neuedu.pojo.Items">
                <id column="items_id" property="id"/>
                <result column="items_name" property="name"/>
                <result column="items_detail" property="detail"/>
                <result column="items_price" property="price"/>
            </association>
        </collection>
     </collection>
```

多对多关联查询

Neuedu

- 3、配置OrdersMapperCustom.xml中select映射

```
<!-- 查询用户及购买的商品信息，使用resultmap -->
<select id="findUserAndItemsResultMap" resultMap="UserAndItemsResultMap">
    SELECT
      orders.*,
      USERINFO.username,
      USERINFO.sex,
      USERINFO.address,
      orderdetail.id orderdetail_id,
      orderdetail.items_id,
      orderdetail.items_num,
      orderdetail.orders_id,
      items.name items_name,
      items.detail items_detail,
      items.price items_price
    FROM
      orders,
      USERINFO,
      orderdetail,
      items
    WHERE orders.user_id = userinfo.id AND orderdetail.orders_id=orders.id AND orderdetail.items_id = items.id
</select>
```

  - 其中resultMap属性值为第2步定义的resultMap
- 4、定义OrdersMapperCustom接口方法
  - public List<User> findUserAndItemsResultMap() throws Exception;
- 5、测试：
- 全部代码参见：ch04-mybatis03工程

课堂练习（20分钟）

- 1、简述Mybatis的多对多关联查询的实现
- 2、理解并独立完成授课案例

分页查询

Neuedu

- 案例：查询所有订单信息及订单下的订单明细信息，进行分页查询。
  - 实现方案
    - 采用oracle数据的rownum进行物理分页
    - 采用RowBounds插件

分页查询

Neuedu

- 实现方案1：使用oracle分页查询，查询所有订单信息及订单下的订单明细信息
  - 1、创建POJO类：User、Orders、OrderDetail、Items
  - 2、创建分页类：Page

```
public class Page implements Serializable{
    private Integer currentPage;
    private Integer pageSize = 3;
    private Integer totalPage;
    private Integer totalRows;
    private Integer startIndex;
    private Integer endIndex;
```

  - 全部代码参见：ch04-mybatis04工程

分页查询

Neuedu

- 3、分页查询代码实现：

```
  <!-- 分页sql代码 -->
<sql id="pageListSql">
  (select tmp.*,rownum rn from
      (SELECT
        orders.*,
        USERINFO.username,
        USERINFO.sex,
        USERINFO.address,
        orderdetail.id orderdetail_id,
        orderdetail.items_id,
        orderdetail.items_num,
        orderdetail.orders_id,
        items.name items_name,
        items.detail items_detail,
        items.price items_price
      FROM
        orders,
        USERINFO,
        orderdetail,
        items
      WHERE orders.user_id = userinfo.id AND orderdetail.orders_id=orders.id AND orderdetail.items_id = items.id
      )tmp where rownum <![CDATA[<=]]> #{endIndex}
   )
where rn > #{startIndex}
</sql>
```

分页查询

Neuedu

- 4、配置OrdersMapperCustom.xml中select映射

```
<!-- 分页查询 -->
<select id="pageList" parameterType="com.neuedu.common.Page" resultMap="UserAndItemsResultMap">
 select * from
 <include refid="pageListSql"/>
</select>
```

  - 其中pageListSql是第3步中定义的SQL片段
  - 其中resultMap属性值是多对多关联查询中定义的resultMap

- 5、定义OrdersMapperCustom接口方法
  - public List<User> pageList(Page page) throws Exception;
- 6、测试：

- 全部代码参见：ch04-mybatis04工程

分页查询

- 实现方案2：使用RowBounds插件，分页查询用户信息
- RowBounds(int offset, int limit)
  - Offset：起始位置
  - limit：每页条数
- 1、UserMapper.java接口
  - public List<UserCustom> findUserWithPage(UserQueryVo user, RowBounds rowBounds) throws Exception;
- 2、UserMapper.xml中select映射

```
<!-- 插件分页查询 -->
  <select id="findUserWithPage" parameterType="com.neuedu.pojo.UserQueryVo"
          resultType="com.neuedu.pojo.UserCustom">
    SELECT * FROM T_USER
    <where>
        <if test="userCustom!=null">
            <if test="userCustom.sex!=null and userCustom.sex!=''">
                and T_USER.sex = #{userCustom.sex}
            </if>
            <if test="userCustom.username!=null and userCustom.username!=''">
                and T_USER.username LIKE '%${userCustom.username}%'
            </if>
        </if>
    </where>
    </select>
```

- 全部代码参见：ch04-mybatis04工程

分页查询

Neuedu

- 3、测试
  - List<UserCustom> list = userMapper.findUserWithPage(userQueryVo, new RowBounds(0, 4));//从第一条取到第四条

  - 全部代码参见：ch04-mybatis04工程

课堂练习（20分钟）

- 1、简述Mybatis的分页查询的实现
- 2、理解并独立完成授课案例

CONTENTS 目录

延迟加载

- 在数据与对象进行 mapping 操作时，只有在真正使用到该对象时，才进行 mapping 操作，以减少数据库查询开销，从而提升系统性能。
- 但是Lazy Load也有缺点，在按需加载时会多次连接数据库，同时会增加数据库的压力。所以在实际使用时，会衡量是否使用延迟加载。
- resultMap可以实现高级映射（使用association、collection实现一对一及一对多映射），association、collection具备延迟加载功能。

延迟加载

Neuedu

- 延迟加载配置
  - Mybatis默认没有开启延迟加载，需要在SqlMapConfig.xml中setting配置。
  - lazyLoadingEnabled：true使用延迟加载，false禁用延迟加载，默认为false。
  - aggressiveLazyLoading：true启用时，当延迟加载开启时访问对象中一个懒对象属性时，将完全加载这个对象的所有懒对象属性。false，当延迟加载时，按需加载对象属性（即访问对象中一个懒对象属性，不会加载对象中其他的懒对象属性）。默认为true

```
<!-- 全局配置参数，需要时再设置 -->
<settings>
    <!-- 打开延迟加载的开关 -->
    <setting name="lazyLoadingEnabled" value="true"/>
    <!-- 将积极加载改为消极加载即按需要加载 -->
    <setting name="aggressiveLazyLoading" value="false"/>
</settings>
```

延迟加载

Neuedu

- 代码实现
  - 1、SqlMapConfig.xml开启懒加载
  - 2、OrdersMapperCustom.java中的接口方法
    - public List<Orders> findOrdersUserLazyLoading() throws Exception;
  - 3、 OrdersMapperCustom.xml实现

```
<!-- 延迟加载的resultMap -->
<resultMap type="com.neuedu.pojo.Orders" id="OrdersUserLazyLoadingResultMap">
        <!--对订单信息进行映射配置 -->
        <id column="id" property="id"/>
        <result column="user_id" property="userId"/>
        <result column="orderid" property="orderId"/>
        <result column="createtime" property="createtime"/>
        <result column="note" property="note"/>
        <!-- 实现对用户信息进行延迟加载-->
        <association property="user"  javaType="com.neuedu.pojo.User"
         select="com.neuedu.mapper.UserMapper.findUserById" column="user_id">
        <!-- 实现对用户信息进行延迟加载 -->
        </association>
</resultMap>

<!-- 查询订单关联查询用户，用户信息需要延迟加载 -->
<select id="findOrdersUserLazyLoading" resultMap="OrdersUserLazyLoadingResultMap">
    SELECT * FROM orders
</select>
```

  - 全部代码参见：ch04-mybatis05工程

延迟加载

Neuedu

- 4、 UserMapper.xml中配置

```
<select id="findUserById" parameterType="int" resultType="user">
    SELECT * FROM T_USER WHERE id=#{value}
</select>
```

- 5、测试
  - Debug模式，添加断点，查看控制台

- 全部代码参见：ch04-mybatis05工程

课堂练习（10分钟）

- 1、简述Mybatis的延迟加载及其实现
- 2、理解授课案例

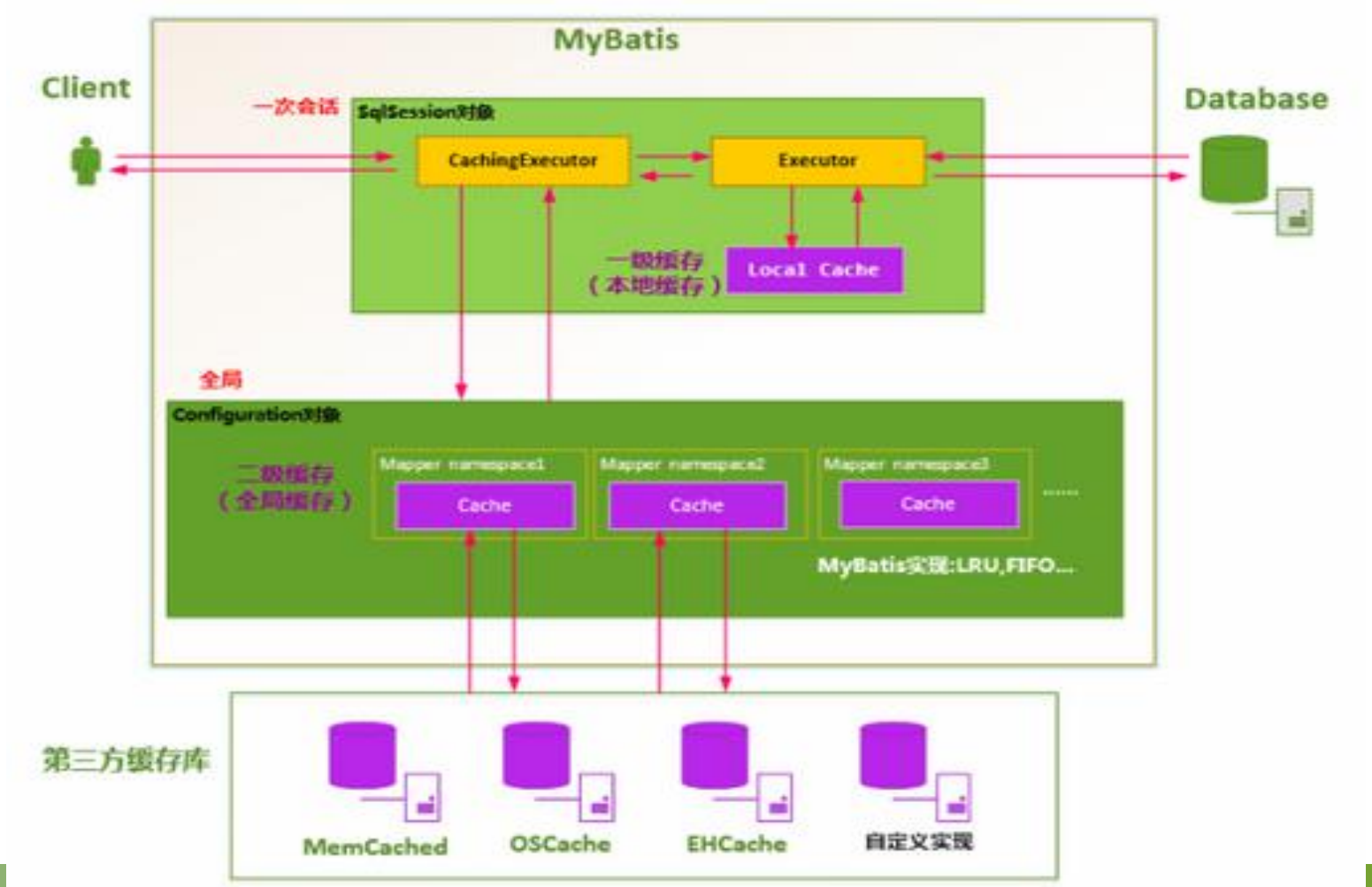
CONTENTS 目录

Mybatis查询缓存

- 缓存技术是一种“以空间换时间”的设计理念，是利用内存空间资源来提高数据检索速度的有效手段之一。
- MyBatis包含一个非常强大的查询缓存特性，可以非常方便地配置和定制。
- mybaits提供一级缓存，和二级缓存。
  - 一级缓存基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该Session中的所有 Cache 就将清空。
  - 二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache、Hazelcast等。

Mybatis查询缓存

- MyBatis的缓存机制整体设计



Mybatis查询缓存

Neuedu

- 一级缓存
  - Mybatis默认支持一级缓存，不需要在配置文件去配置。

```
// 一级缓存测试
//第一次发起查询id为1的用户信息，先去找缓存中是否有id为1的用户信息，
//如果没有，从数据库查询用户信息，得到信息存储到一级缓存中。
//如果sqlSession去执行commit操作（执行插入、更新、删除），清空SqlSession中的一级缓存，
//这样做的目的为了让缓存中存储的是最新的信息，避免脏读。
//第二次发起查询用户id为1的信息，先去找缓存中是否有id为1的信息，缓存中有，直接从缓存中获取用户信息。
@Test
public void testCache1() throws Exception {
    SqlSession sqlSession = sqlSessionFactory.openSession();// 创建代理对象
    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

    // 下边查询使用一个SqlSession
    // 第一次发起请求，查询id为1的用户
    User user1 = userMapper.findUserById(1);
    System.out.println(user1);
    // 第二次发起请求，查询id为1的用户
    User user2 = userMapper.findUserById(1);
    System.out.println(user2);

    sqlSession.close();

}
```

  - 全部代码参见：ch04-mybatis06工程

Mybatis查询缓存

Neuedu

- 二级缓存
  - mybaits的二级缓存是mapper范围级别，除了在SqlMapConfig.xml设置二级缓存的总开关，还要在具体的mapper.xml中开启二级缓存。
  - 在核心配置文件SqlMapConfig.xml中加入
    - `<setting name="cacheEnabled" value="true"/>`
  - 在UserMapper.xml中开启二缓存
    - `<cache type="org.mybatis.caches.ehcache.EhcacheCache"/>`
  - pojo类必须实现序列化接口
    - `public class User implements Serializable { }`
  - 全部代码参见：ch04-mybatis06工程

Mybatis查询缓存

Neuedu

- 二级缓存

```
// 二级缓存测试
@Test
public void testCache2() throws Exception {
    SqlSession sqlSession1 = sqlSessionFactory.openSession();
    SqlSession sqlSession2 = sqlSessionFactory.openSession();

    // 创建代理对象
    UserMapper userMapper1 = sqlSession1.getMapper(UserMapper.class);
    // 第一次发起请求，查询id为1的用户
    User user1 = userMapper1.findUserById(1);
    System.out.println(user1);

    //这里执行关闭操作，将sqlsession中的数据写到二级缓存区域
    sqlSession1.close();

    UserMapper userMapper2 = sqlSession2.getMapper(UserMapper.class);
    // 第二次发起请求，查询id为1的用户
    User user2 = userMapper2.findUserById(1);
    System.out.println(user2);

    sqlSession2.close();

}
```

- 全部代码参见：ch04-mybatis06工程

Mybatis查询缓存

Neuedu

- 二级缓存
  - 在statement中设置useCache=false可以禁用当前select语句的二级缓存，即每次查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。

```
<!-- 在statement中设置useCache=false可以禁用当前select语句的二级缓存 -->
<select id="findUserById" parameterType="int" resultType="user" useCache="false">
    SELECT * FROM T_USER WHERE id=#{value}
</select>
```

- 刷新缓存
  - 在mapper的同一个namespace中，如果有其它insert、update、delete操作数据后需要刷新缓存，如果不执行刷新缓存会出现脏读。
  - 设置statement配置中的flushCache="true" 属性，默认情况下为true即刷新缓存，如果改成false则不会刷新。

```
<!--  设置statement配置中的flushCache="true" 属性，默认情况下为true即刷新缓存 -->
<update id="updateUser" parameterType="com.neuedu.pojo.User" flushCache="true">
    update T_USER set username=#{username},birthday=#{birthday},sex=#{sex},address=#{address}
     where id=#{id}
</update>
```

Mybatis整合第三方缓存框架

Neuedu

- 分布式缓存框架。
  - 我们系统为了提高系统并发，性能、一般对系统进行分布式部署（集群部署方式）
  - 不使用分布缓存，缓存的数据在各个服务单独存储，不方便系统开发。所以要使用分布式缓存对缓存数据进行集中管理。
  - ehcache、memcache、redis 缓存框架.
- Ehcache
  - 是一种广泛使用的开源Java分布式缓存。主要面向通用缓存,Java EE和轻量级容器。它具有内存和磁盘存储，缓存加载器，缓存扩展，缓存异常处理程序，一个gzip缓存servlet过滤器，支持REST和SOAP api等特点。
  - 被用于大型复杂分布式web application的各个节点中。

Mybatis整合ehcache

Neuedu

- 整合ehcache
  - Mybatis提供了一个cache接口，如果要实现自己的缓存逻辑，实现cache接口开发即可。
  - Mybatis和ehcache整合包中提供了一个cache接口的实现类。

- 整合流程
  - 1、项目中加入ehcache依赖的jar包：

```
42
43      <dependency>
44          <groupId>org.mybatis.caches</groupId>
45          <artifactId>mybatis-ehcache</artifactId>
46          <version>1.0.2</version>
47      </dependency>
48      <dependency>
49          <groupId>net.sf.ehcache</groupId>
50          <artifactId>ehcache</artifactId>
51          <version>2.10.1</version>
52      </dependency>
```

  - 全部代码参见：ch04-mybatis06工程

Mybatis整合ehcache

Neuedu

- 2、配置mapper中cache中的type为ehcache对cache接口的实现类。

```
<!-- 开启本mapper的namespace下的 二级缓存
type：指定cache接口的实现类的类型，mybatis默认使用PerpetualCache
要和ehcache整合，需要配置type为ehcache实现cache接口的类型
-->
<cache type="org.mybatis.caches.ehcache.EhcacheCache"></cache>
```

- 全部代码参见：ch04-mybatis06工程

Mybatis整合ehcache

Neuedu

- 3、在类路径下配置ehcache.xml的配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ehcache.xsd"
         updateCheck="true" monitoring="autodetect"
         dynamicConfig="true">
    <diskStore path="D:\develop\ehcache" />
    <defaultCache
        maxElementsInMemory="1000"
        maxElementsOnDisk="10000000"
        eternal="false"
        overflowToDisk="true"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        diskExpiryThreadIntervalSeconds="120"
        memoryStoreEvictionPolicy="LRU">
    </defaultCache>
```

- 全部代码参见：ch04-mybatis06工程

课堂练习（5分钟）

- 1、简述Mybatis的缓存策略及其实现

CONTENTS 目录

本章实战项目任务实现

Neuedu

- 通过本章内容的学习，使用关联查询、分页技术和缓存策略，完成《跨境电商系统》所有用户关联的订单及订单明细的查询
  - 使用二级缓存和ehcahe缓存框架策略
  - 在控制台输出查询结果的第一页（每页显示2条数据）
  - 参考《跨境电商借卖平台数据库表设计》
  - 建表脚本 参考跨境电商案例-1.sql、跨境电商案例-2.sql

- 运行效果在控制台输出：

```
DEBUG [main] - sysUserList.size=2
DEBUG [main] - userId = 1
DEBUG [main] - salesOrderList.size=2
DEBUG [main] - saoId = 24
DEBUG [main] - salesOrderLineItemList.size = 2
DEBUG [main] - salId=28,proId=1
DEBUG [main] - salId=27,proId=1
DEBUG [main] - saoId = 23
DEBUG [main] - salesOrderLineItemList.size = 2
DEBUG [main] - salId=26,proId=1
DEBUG [main] - salId=25,proId=1
DEBUG [main] - userId = 2
DEBUG [main] - salesOrderList.size=1
DEBUG [main] - saoId = 22
DEBUG [main] - salesOrderLineItemList.size = 1
DEBUG [main] - salId=24,proId=1
```

本章重点总结

- 了解Mybatis的一级缓存；
- 了解Ehcache第三方缓存框架；
- 理解Mybatis的延迟加载；
- 掌握Mybatis的关联查询（1对1、1对多、多对多）；
- 掌握Mybatis的分页查询；
- 掌握Mybatis的二级缓存；

课后作业【必做任务】

- 1、使用关联查询，完成《跨境电商系统》的订单及关联的订单明细的查询。
  - 根据订单id查询订单及关联的订单明细
- 2、使用延迟加载，完成《跨境电商系统》的订单及关联的订单明细的查询。
  - 查询所有的订单及关联的订单明细

课后作业【选做任务】

Neuedu

- 使用缓存策略，完成《跨境电商系统》的订单的查询。
  - 根据订单id查询订单
  - 测试：使用一级缓存，查询3个对象
  - 测试：使用二级缓解，查询2个对象

课后作业【线上任务】

- 线上任务
  - 安排学员线上学习任务（安排学员到睿道实训平台进行复习和预习的任务，主要是进行微课的学习）