Mybatis延迟加载的原理是什么，它有那些使用场景?

题目标签

学习时长：20分钟

题目难度：中等

知识点标签：Mybatis延迟加载

题目描述

Mybatis延迟加载的原理是什么，它有那些使用场景?

面试题分析

首先要先知道延迟加载的概念，再明白它的原理，最后了解应用场景

1.什么是延迟加载

MyBatis中的延迟加载，也称为懒加载，是指在进行表的关联查询时，按照设置延迟规则推迟对关联对象的select查询。例如在进行一对多查询的时候，只查询出一方，当程序中需要多方的数据时，mybatis再发出sql语句进行查询，这样子延迟加载就可以的减少数据库压力。MyBatis 的延迟加载只是对关联对象的查询有迟延设置，对于主加载对象都是直接执行查询语句的。

注意：延迟加载的应用要求：关联对象的查询与主加载对象的查询必须是分别进行的select语句，不能是使用多表连接所进行的select查询

2.加载时机

mybatis对于延迟加载的时机支持三种形式
直接加载：执行完对主加载对象的 select 语句，马上执行对关联对象的 select 查询。
侵入式延迟： 执行对主加载对象的查询时，不会执行对关联对象的查询。但当要访问主加载对象的详情属性时，就会马上执行关联对象的select查询。
深度延迟： 执行对主加载对象的查询时，不会执行对关联对象的查询。访问主加载对象的详情时也不会执行关联对象的select查询。只有当真正访问关联对象的详情时，才会执行对关联对象的 select 查询。

3.延迟加载使用场景

首先我们先思考一个问题，假设：在一对多中，我们有一个用户，他有100个账户。

问题1：在查询用户的时候，要不要把关联的账户查出来?

问题2：在查询账户的时候，要不要把关联的用户查出来?

解答：在查询用户的时候，用户下的账户信息应该是我们什么时候使用，什么时候去查询。

在查询账户的时候，账户的所属用户信息应该是随着账户查询时一起查询出来。

在对应的四种表关系中，一对多、多对多通常情况下采用延迟加载，多对一、一对一通常情况下采用立即加载。

理解了延迟加载的特性以后再看Mybatis中如何实现查询方法的延迟加载，在MyBatis 的配置文件中通过设置settings的lazyLoadingEnabled属性为true进行开启全局的延迟加载，通过aggressiveLazyLoading属性开启立即加载。看一下官网的介绍，然后通过一个实例来实现Mybatis的延迟加载，在例子中我们展现一对多表关系情况下，通过实现查询用户信息同时查询出该用户所拥有的账户信息的功能展示一下延迟加载的实现方式以及延迟加载和立即加载的结果的不同之处。

1.用户类以及账户类

```
public class User implements Serializable{
    private Integer id;
    private String username;
    private Date birthday;
    private String sex;
    private String address;
    private List<Account> accountList;

    get和set方法省略.....
}


public class Account implements Serializable{
    private Integer id;
    private Integer uid;
    private Double money;

    get和set方法省略.....
}
```

注意因为我们是查找用户的同时查找出其所拥有的账户所以我们需要在用户类中增加账户的集合的属性，用来封装返回的结果。

2.在UserDao接口中声明findAll方法

```
/**
     * 查询所有的用户
     *
     * @return
     */
    List<User> findAll();
```

3.在UserDao.xml中配置findAll方法的映射

```
<resultMap id="userAccountMap" type="com.example.domain.User">
        <id property="id" column="id"/>
        <result property="username" column="username"/>
        <result property="birthday" column="birthday"/>
        <result property="sex" column="sex"/>
        <result property="address" column="address"/>
        <collection property="accountList" ofType="com.example.domain.Account"
column="id"
                    select="com.example.dao.AccountDao.findAllByUid"/>
    </resultMap>
    <select id="findAll" resultMap="userAccountMap">
        SELECT * FROM USER;
    </select>
```

主要的功能实现位于 中，对于账户列表的信息通过collection集合来映射，通过select指定集合中的每个元素如何查询，在本例中select的属性值为AccountDao.xml文件的namespace com.example.dao.AccountDao路径以及指定该映射文件下的findAllByUid方法，通过这个唯一标识指定集合中元素的查找方式。因为在这里需要用到根据用户ID查找账户，所以需要同时配置一下findAllByUid方法的实现。

4.配置collection中select属性所使用的方法 findAllByUid

```
AccountDao接口中添加
 /**
     * 根据用户ID查询账户信息
     * @return
     */
    List<Account> findAllByUid(Integer uid);

AccountDao.xml文件中配置
<select id="findAllByUid" resultType="com.example.domain.Account">
        SELECT * FROM account WHERE uid = #{uid};
    </select>
```

5.在Mybatis的配置文件中开启全局延迟加载

```
configuration>
    <settings>
        <!--开启全局的懒加载-->
        <setting name="lazyLoadingEnabled" value="true"/>
        <!--关闭立即加载，其实不用配置，默认为false-->
        <setting name="aggressiveLazyLoading" value="false"/>
        <!--开启Mybatis的sql执行相关信息打印-->
        <setting name="logImpl" value="STDOUT_LOGGING" />
    </settings>
    <typeAliases>
        <typeAlias type="com.example.domain.Account" alias="account"/>
        <typeAlias type="com.example.domain.User" alias="user"/>
        <package name="com.example.domain"/>
    </typeAliases>
    <environments default="test">
        <environment id="test">
            <!--配置事务-->
            <transactionManager type="jdbc"></transactionManager>
```

```
            <!--配置连接池-->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/test1"/>
                <property name="username" value="root"/>
                <property name="password" value="123456"/>
            </dataSource>
        </environment>
    </environments>
    <!--配置映射文件的路径-->
    <mappers>
        <mapper resource="com/example/dao/UserDao.xml"/>
        <mapper resource="com/example/dao/AccountDao.xml"/>
    </mappers>
</configuration>
```

6.测试方法

```
private InputStream in;
    private SqlSession session;

    private UserDao userDao;
    private AccountDao accountDao;
    private SqlSessionFactory factory;
    @Before
    public void init()throws Exception{
        //获取配置文件
        in = Resources.getResourceAsStream("SqlMapConfig.xml");
        //获取工厂
        factory = new SqlSessionFactoryBuilder().build(in);

        session = factory.openSession();

        userDao = session.getMapper(UserDao.class);
        accountDao = session.getMapper(AccountDao.class);
    }
    @After
    public void destory()throws Exception{
        session.commit();
        session.close();
        in.close();
    }
    @Test
    public void findAllTest(){
        List<User> userList = userDao.findAll();
//        for (User user: userList){
//            System.out.println("每个用户的信息");
//            System.out.println(user);
//            System.out.println(user.getAccountList());
//        }
    }
```

7.测试结果

（1）注释for循环，不使用数据，这时候不需要查询账户信息，通过第五步的sql语句控制台打印看出来，不使用数据的时候，就没有发起对账户的查询。

（2）通过for循环打印查询的数据，使用数据，这时候因为使用了数据所以将查询账户信息，我们可以通过控制台的sql语句打印发现用户和账户查询都进行了执行