MyBatis 延迟加载

- 什么是延迟加载?

延迟加载也叫惰性加载、懒加载，是提高程序运行效率的一种方式，主要针对于 Repository 层的操作，Java 程序与数据库交互的频次越低，程序运行效率越高，所以实际开发中应该尽量减少 Java 程序与数据的交互次数，MyBatis 框架提供了很好的延迟加载机制。

延迟加载的实际场景：

班级(Classes) 和学生(Student) ，当我们查询 Student 对象时，因为有级联设置，所有会将对应的 Classes 对象一并查处，这样就需要发送两条 SQL 语句，分别查询 Classes 和 Student 表中的数据。

延迟加载的思路是：当我们查询 Student 的时候，如果没有调用 Classes 的任何属性，则只发送一条 SQL 语句查询 Student，如果需要调用 Classes 的属性，再来发送一条 SQL 查询 Classes 数据，所以延迟加载可以看作是一种优化机制，根据具体的代码，自动选择发送的 SQL 语句。

开启延迟加载

```
<settings>
    <!-- 打印SQL-->
    <setting name="logImpl" value="STDOUT_LOGGING" />
    <!-- 开启延迟加载 -->
    <setting name="lazyLoadingEnabled" value="true"></setting>
    <!-- 将即时加载改为按需加载 -->
    <setting name="aggressiveLazyLoading" value="false"></setting>
</settings>
```

MyBatis 缓存

- 什么是 MyBatis 缓存?

使用缓存可以减少 Java 程序与数据库的交互次数，从而提高程序的运行效率，比如，查询 id=1 的 User 对象，第一次查询出来之后，会自动将该对象保存到 MyBatis 缓存中，下一次查询该对象时，可以直接从缓存中取出数据，不需要执行 SQL 来查询数据库。

- MyBatis 缓存分类
    - 一级缓存：SqlSession 级别的缓存，默认开启，且不能关闭。

    MyBatis 的一级缓存是 SqlSession 级别的缓存，在操作数据库时需要构造 SqlSession 对象，在对象中有一个 HashMap 用于存储缓存数据，不同的 SqlSession 直接缓存数据区域（HashMap）是互不影响的。

    一级缓存的作用域是 SqlSession 范围的，当在同一个 SqlSession 中执行两次相同的 SQL 语句时，第一次执行完毕会将数据库中查询到的数据存入缓存中，第二次查询时会直接从缓存中取出数据，不再去查询数据库，从而提高了查询效率。

需要注意的是：如果 SqlSession 执行了 DML 操作（insert、update、delete），并执行了 commit() 操作，MyBatis 框架则会清空 SqlSession 中的一级缓存，这样做的目的是为了保证缓存数据中存储的是最新信息，避免出现脏读现象。

- 二级缓存：Mapper 级别的缓存，默认关闭，可以开启。

二级缓存是 Mapper 级别的缓存，使用二级缓存时，多个 SqlSession 使用同一个 Mapper 的 SQL 语句去操作数据库，得到的数据会存在二级缓存区，它同样是使用 HashMap 进行数据存储，相比较于一级缓存 SqlSession，二级缓存的范围更大，多个 SqlSession 可以共用二级缓存，二级缓存是跨 SqlSession 的。

二级缓存是多个 SqlSession 共享的，其作用域是 Mapper 的同一个 namespace，不同的 SqlSession 两次执行相同的 namespace 下的 SQL 语句，并且向 SQL 中传递的参数也一致，即最终执行的 SQL 语句相同，则第一次执行完毕会将数据存入缓存中，第二次查询时会直接从缓存中读取数据，不再去查询数据库，从而提高程序的运行效率。

MyBatis 框架默认关闭二级缓存，可以在 settings 全局参数重配置开启二级缓存。

在关闭 SqlSession，一级缓存失效的情况下，可以启用二级缓存，实现提高程序运行效率的需求。

MyBatis 可以使用自带的二级缓存，也可以使用第三方提供的二级缓存 ehcache。

- MyBatis 自带的二级缓存

1、config.xml 中配置开启二级缓存。

```
<settings>
    <!-- 打印SQL-->
    <setting name="logImpl" value="STDOUT_LOGGING" />
    <!-- 开启二级缓存 -->
    <setting name="cacheEnabled" value="true"></setting>
</settings>
```

2、UserRepository.xml 中配置开启二级缓存

```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="com.southwind.repository.UserRepository">
    <!-- 开启二级缓存 -->
    <cache></cache>
    <select id="findById" parameterType="int"
resultType="com.southwind.entity.User">
        select * from t_user where id = #{id}
    </select>

</mapper>
```

3、User 实体类实现 Serializable 接口。

```java
package com.southwind.entity;

import java.io.Serializable;

public class User implements Serializable {
    private int id;
    private String username;
    private String password;
    private int age;

    public int getId() {
        return id;
    }

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

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
```

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", age=" + age +
                '}';
    }
}
```

- ehcache 二级缓存

1、pom.xml 中添加 ehcache 依赖。

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-ehcache</artifactId>
    <version>1.0.0</version>
</dependency>

<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache-core</artifactId>
    <version>2.4.3</version>
</dependency>
```

2、添加 ehcache.xml

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">
    <diskStore/>
    <defaultCache
            maxElementsInMemory="1000"
            maxElementsOnDisk="10000000"
            eternal="false"
            overflowToDisk="false"
            timeToIdleSeconds="120"
```

```
            timeToLiveSeconds="120"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
    </defaultCache>
</ehcache>
```

3、config.xml 中配置开启二级缓存

```
<settings>
    <!-- 打印SQL-->
    <setting name="logImpl" value="STDOUT_LOGGING" />
    <!-- 开启二级缓存 -->
    <setting name="cacheEnabled" value="true"></setting>
</settings>
```

4、UserRepository.xml 中配置二级缓存

```
<?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="com.southwind.repository.UserRepository">
    <!-- 开启二级缓存 -->
    <cache type="org.mybatis.caches.ehcache.EhcacheCache">
        <!-- 缓存创建之后，最后一次访问缓存的时间至缓存失效的时间间隔 -->
        <property name="timeToIdleSeconds" value="3600"/>
        <!-- 缓存自创建时间起至失效的时间间隔 -->
        <property name="timeToLiveSeconds" value="3600"/>
        <!-- 缓存回收策略，LRU 移除近期使用最少的对象 -->
        <property name="memoryStoreEvictionPolicy" value="LRU" />
    </cache>
    <select id="findById" parameterType="int"
resultType="com.southwind.entity.User">
        select * from t_user where id = #{id}
    </select>

</mapper>
```

5、User 实体类不需要实现 Serializable 接口。

```
package com.southwind.entity;

public class User {
    private int id;
    private String username;
    private String password;
    private int age;

    public int getId() {
```

```
        return id;
    }

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

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", age=" + age +
                '}';
    }
}
```

MyBatis 动态 SQL

在业务比较复杂的情况下，我们通常需要拼接 SQL 语句来完成相关操作，大量的进行 SQL 语句的拼接工作比较繁琐，效率较低，而且容易出错。MyBatis 框架提供了一个非常方便且功能强大的动态 SQL 功能，使用动态 SQL ，可以摆脱手动拼接 SQL 的不便。

```java
package com.southwind.entity;

public class User {
    private int id;
    private String username;
    private String password;
    private int age;

    public int getId() {
        return id;
    }

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

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", age=" + age +
                '}';
    }
```

```
}
```

UserRepository

```
package com.southwind.repository;

import com.southwind.entity.User;

public interface UserRepository {
    public User findById(int id);
    public User findByUser(User user);
}
```

UserRepository.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="com.southwind.repository.UserRepository">
    <!-- 开启二级缓存 -->
    <cache type="org.mybatis.caches.ehcache.EhcacheCache">
        <!-- 缓存创建之后，最后一次访问缓存的时间至缓存失效的时间间隔 -->
        <property name="timeToIdleSeconds" value="3600"/>
        <!-- 缓存自创建时间起至失效的时间间隔 -->
        <property name="timeToLiveSeconds" value="3600"/>
        <!-- 缓存回收策略，LRU 移除近期使用最少的对象 -->
        <property name="memoryStoreEvictionPolicy" value="LRU" />
    </cache>
    <select id="findById" parameterType="int"
resultType="com.southwind.entity.User">
        select * from t_user where id = #{id}
    </select>

    <select id="findByUser" parameterType="com.southwind.entity.User"
resultType="com.southwind.entity.User">
        select * from t_user
        <where>
            <if test="id!=0">
                id = #{id}
            </if>
            <if test="username!=null">
                and username = #{username}
            </if>
            <if test="password!=null">
                and password = #{password}
            </if>
            <if test="age!=0">
                and age = #{age}
```

```
            </if>
        </where>
    </select>

</mapper>
```

标签会自动判断对应的属性是否为空，如果为空，则从 SQL 去掉对应的条件，否则保留。

标签会自动判断 where 关键字与紧随其后的条件直接是否存在其他条件，如果存在，则保留条件中的 and 关键字，如果不存在，则删除条件中的 and 关键字。