MyBatis 延迟加载

一种提高查询效率的方式，在数据有级联的情况下，可以动态地进行选择查询符合条件的结果。

延迟加载 懒加载 惰性加载

```
package com.southwind.mapper;

import com.southwind.entity.Student;

public interface StudentMapper {
    public Student findById(Integer id);
}
```

```
<?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.mapper.StudentMapper">

    <resultMap id="studentMap" type="com.southwind.entity.Student">
        <id property="id" column="id"></id>
        <result column="name" property="name"></result>
        <result column="age" property="age"></result>
        <association
        property="classes"
        javaType="com.southwind.entity.Classes"
        column="cid"
        select="com.southwind.mapper.ClassesMapper.findById"
        ></association>
    </resultMap>

    <select id="findById" resultMap="studentMap">
        select * from student where id = #{id}
    </select>

</mapper>
```

```
package com.southwind.mapper;

import com.southwind.entity.Classes;

public interface ClassesMapper {
    public Classes findById(Integer id);
}
```

```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.mapper.ClassesMapper">

    <select id="findById" resultType="com.southwind.entity.Classes">
        select * from classes where id = #{id}
    </select>

</mapper>
```

```xml
<settings>
    <!-- 打印SQL -->
    <setting name="logImpl" value="STDOUT_LOGGING" />
    <!-- 开启延迟加载 -->
    <setting name="lazyLoadingEnabled" value="true"/>
</settings>
```

MyBatis 缓存

缓存也是用来提高效率的一种方式，类似于字符串常量池、数据库连接池

缓存的作用是减少 Java 程序和数据库的交互次数，从而提升程序的运行效率，MyBatis 查询出一个结果之后，将该结果存入缓存，下一次可以直接从缓存中取出结果，而不需要连接数据库进行查询。

如果存入缓存之后做了修改操作，则 MyBatis 会清空缓存以保证数据的时效性

MyBatis 缓存

1、一级缓存，默认开启且无法关闭的，同一个 SqlSession 中有效

```java
StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);
Student student = mapper.findById(15);
System.out.println(student.getName());
sqlSession.close();
sqlSession = factory.openSession();
mapper = sqlSession.getMapper(StudentMapper.class);
Student student1 = mapper.findById(15);
System.out.println(student1.getName());
```

2、二级缓存，比一级缓存作用域更大的一个缓存，Mapper 级别的，只要同一个 Mapper，无论多少个 SqlSession，二级缓存都是有效的

二级缓存默认是关闭的，需要手动开启，修改 config.xml

```xml
<!-- 开启二级缓存 -->
<setting name="cacheEnabled" value="true"/>
```

Mapper.xml 添加缓存标签

```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.mapper.StudentMapper">
```

```
    <cache></cache>

    <resultMap id="studentMap" type="com.southwind.entity.Student">
        <id property="id" column="id"></id>
        <result column="name" property="name"></result>
        <result column="age" property="age"></result>
        <association
        property="classes"
        javaType="com.southwind.entity.Classes"
        column="cid"
        select="com.southwind.mapper.ClassesMapper.findById"
        ></association>
    </resultMap>

    <select id="findById" resultMap="studentMap">
        select * from student where id = #{id}
    </select>

</mapper>
```

实体类实现序列化接口

```
package com.southwind.entity;

import lombok.Data;

import java.io.Serializable;

@Data
public class Student implements Serializable {
    private Integer id;
    private String name;
    private Integer age;
    private Classes classes;
}
```

MyBatis 动态 SQL

根据参数信息动态生成 SQL 语句，不同的参数会创建不同的 SQL 来完成数据查询操作

```
<select id="findByStudent" resultMap="studentMap">
    select * from student
    <where>
        <if test="id != null">
            id = #{id}
        </if>
        <if test="name != null">
            and name = #{name}
        </if>
        <if test="age != null">
            and age = #{age}
        </if>
    </where>

</select>
```

if 是判断当前属性是否为 null，如果为 null，statement 就不会添加到 SQL 中

where 是判断 and 关键字是否会和 where 关键字拼接到一起，如果拼接到一起则自动删除 and 关键字

choose、when

```
<select id="findByStudent" resultMap="studentMap">
    select * from student
    <where>
        <choose>
            <when test="id != null">
                id = #{id}
            </when>
            <when test="name != null">
                and name = #{name}
            </when>
            <when test="age != null">
                and age = #{age}
            </when>
        </choose>
    </where>

</select>
```

set

```
<update id="update">
    update student
    <set>
        <if test="name != null ">
            name = #{name},
        </if>
        <if test="age != null ">
            age = #{age}
        </if>
    </set>
    where id = #{id}
</update>
```

foreach

```
<select id="findByStudent" resultMap="studentMap">
    select * from student
    <where>
        <foreach collection="ids" open="id in (" close=")" item="id"
separator=",">
            #{id}
        </foreach>
    </where>
</select>
```

使用 MyBatis 配置比较繁琐

配置文件

builder-》factory-》sqlSession-》mapper

要使用 MyBatis 需要加载的组件很多，先创建很多对象，才能获取到 mapper，进行操作，开发步骤比较多

如果希望直接获取 mapper 进行操作

可以使用 Spring 框架帮助开发者自动生成所需要的各种组件，并完成依赖注入组装，开发者直接使用即可。

Spring

IoC 和 AOP

IoC：控制反转，让 Spring 自动生成程序中所需要的各种组件，开发者直接用，不需要创建，开箱即用。

启动 Spring 的时候，读取 spring.xml 获取开发者需要的对象信息，通过反射机制自动创建这些对象，存入到 IoC 容器中，开发者只需要从 IoC 中获取对应的对象即可。

AOP：面向切面编程，面向对象的一种补充，做到核心业务和非核心业务的解耦合。

IoC

1、创建 Spring 工程

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.15</version>
</dependency>
```

2、在 spring.xml 中配置 bean，告诉 Spring 框架你需要的对象

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <bean class="com.southwind.entity.User" id="user">
        <property name="id" value="1"></property>
        <property name="name" value="张三"></property>
        <property name="age" value="22"></property>
    </bean>

</beans>
```

```java
package com.southwind;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
    public static void main(String[] args) {
        //启动Spring
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("spring.xml");
        Object user = applicationContext.getBean("user");
        System.out.println(user);
    }
}
```