Mybatis

概念

ORM

对象关系映射（Object Relational Mapping，简称ORM）模式是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术。简单的说，ORM是通过使用描述对象和数据库之间映射的元数据，将程序中的对象自动持久化到关系数据库中或者将关系型数据库的记录映射成对象。

ORM的方法论基于三个核心原则：

- 简单：以最基本的形式建模数据。
- 传达性：数据库结构被任何人都能理解的语言文档化。
- 精确性：基于数据模型创建正确标准化了的结构。

Mybatis

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。

执行过程

(1) 从配置文件(通常是 XML 配置文件中)得到 sessionfactory.

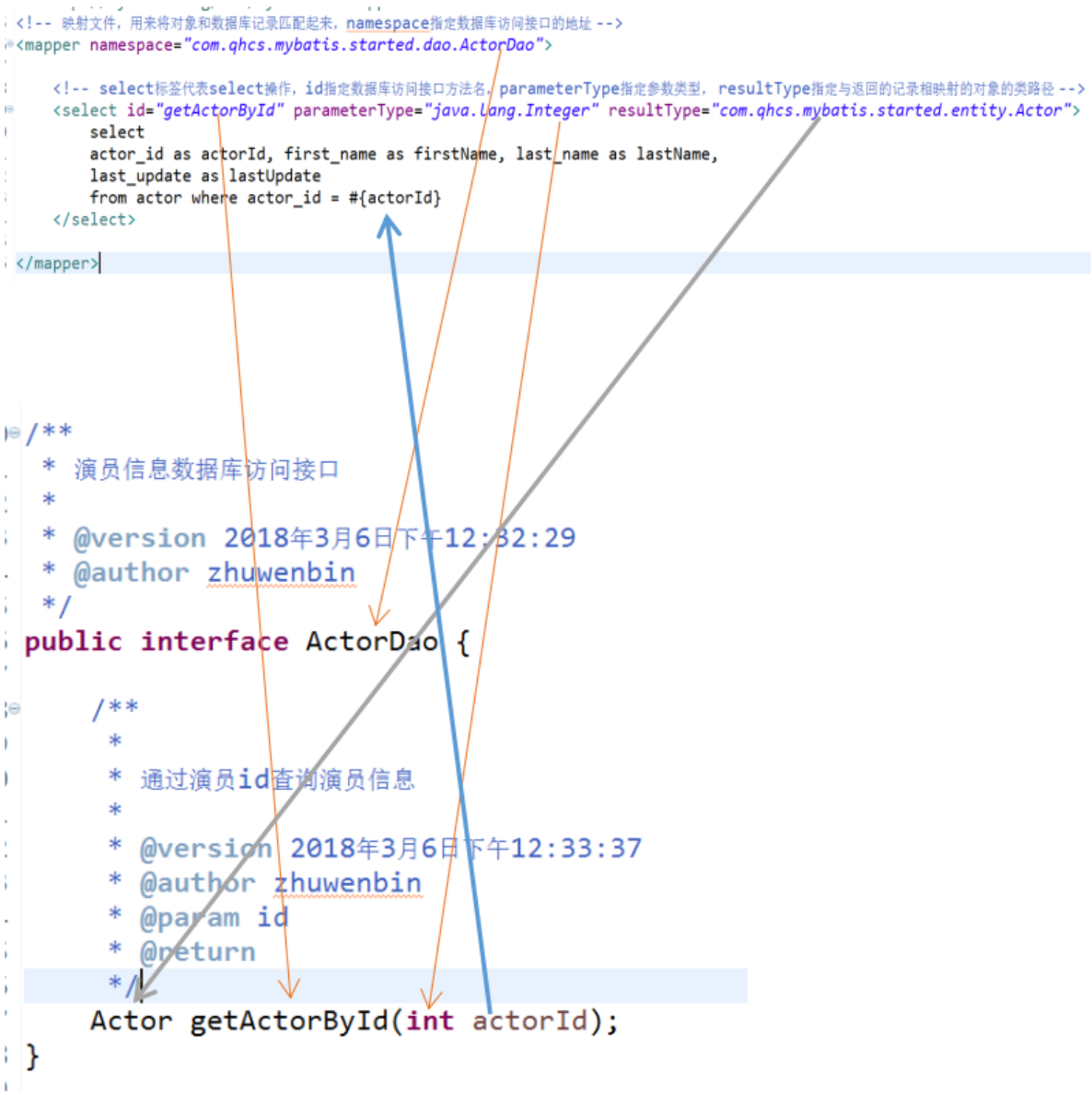
(2) 由 sessionfactory 产生 session

(3) 在 session 中完成对数据的增删改查和事务提交等.

(4) 在 Java 对象和 数据库之间有做 mapping 的映射文件，也通常是 xml 文件，进行相互转换。

(5) 在用完之后关闭 session 。

映射过程



参数获取

(1) #{name}：相当于jdbc的PreparedStatement，创建预处理语句属性并安全地设置值。

(2) ${name}：相当于jdbc的Statement，通过字符串拼接设置值，存在sql注入，不安全。

参数传递

(1) 单个参数

① 基本数据类型数据和String类型，可以不指定参数类型

(2) 多个参数

① 对象，指定对象的类型

② Map集合，指定Map类型

③ Set集合

④ List集合

⑤ 数组

主键返回

```
<!-- insert标签代表insert操作，useGeneratedKeys这会令 MyBatis 使用 JDBC 的 getGeneratedKeys
    方法来取出由数据库内部生成的主键，keyProperty指定将获取的主键绑定参数的哪个属性上面 -->
<insert id="addActor" parameterType="actor" useGeneratedKeys="true"
    keyProperty="actorId">
    insert into actor(first_name, last_name, last_update)
    values
    (#{firstName}, #{lastName}, #{lastUpdate})
</insert>
```

连接查询

一对一

1. 创建接口

```
/**
 * 电影的接口
 *
 * @version 2018年3月7日上午11:03:38
 * @author zhuwenbin
 */
public interface FilmDao {

    /**
     *
     * 根据电影名字查询电影信息
     *
     * @version 2018年3月7日上午11:06:52
     * @author zhuwenbin
     * @param title
     * @return
     */
    Film getFilmByTitle(String title);
}
```

2. 配置实体关系

```java
/**
 * 电影实体
 *
 * @version 2018年3月7日上午10:05:42
 * @author zhuwenbin
 */
public class Film {

    private int filmId;
    private String title;
    private String description;
    private Date releaseYear;
    private int length;
    // 表示film和filmText是一对一的关系
    private FilmText filmText;
}
```

3. 配置映射关系

```xml
<!-- 映射文件，用来将对象和数据库记录匹配起来，namespace指定数据库访问接口的地址 -->
<mapper namespace="com.qhcs.mybatis.advanced.dao.FilmDao">

    <!-- type指映射查询结果的对象类型 -->
    <resultMap type="film" id="film_resultMap">
        <!-- result匹配字段与对象属性的关系，column指查询结果的字段名，property指对象的属性名 -->
        <result column="film_id" property="filmId"/>
        <result column="release_year" property="releaseYear"/>
        <result column="length" property="length"/>
        <!-- association用来构造复杂的结果集，一般用来描述一对一和多对一的关系，property指film对象的属性名，javaType指filmText对象的属性类型 -->
        <association property="filmText" javaType="filmText">
            <result column="film_id" property="filmId"/>
            <result column="title" property="title"/>
            <result column="description" property="description"/>
        </association>
    </resultMap>

    <!-- resultMap用来手动配置查询结果和对象的关系， 这里填写resultMap的id-->
    <select id="getFilmByTitle" resultMap="film_resultMap">
        SELECT
            f.film_id,
            f.release_year,
            f.length,
            ft.title,
            ft.description
        FROM
            film AS f
        LEFT JOIN film_text AS ft ON f.film_id = ft.film_id
        WHERE
            ft.title = #{title};
```

```
    </select>
</mapper>
```

一对多

1. 创建接口

```
/**
 * 国家信息接口
 *
 * @version 2018年3月7日下午2:36:21
 * @author zhuwenbin
 */
public interface CountryDao {

    /**
     *
     * 根据国家名查询该国家和其城市列表
     *
     * @version 2018年3月7日下午2:39:14
     * @author zhuwenbin
     * @param name
     * @return
     */
    Country getCountryCityByName(String name);
}
```

2. 配置实体关系

```
/**
 * 国家实体
 *
 * @version 2018年3月7日上午10:18:03
 * @author zhuwenbin
 */
public class Country {

    private int country_id;
    private String country;
    private Date lastUpdate;
    // 表示country和city是一对多的关系
    private List<City> cities;
}
```

3. 配置映射关系

```
<!-- 映射文件，用来将对象和数据库记录匹配起来，namespace指定数据库访问接口的地址 -->
```

```xml
<mapper namespace="com.qhcs.mybatis.advanced.dao.CountryDao">

    <!-- country基本结果集 -->
    <resultMap type="country" id="country_baseMap">
        <!-- 这些是结果映射最基本内容。id 和 result 都映射一个单独列的值到简单数据类型(字符串,整型,双精度浮点数,日期等)的单独属性或字段。
            这两者之间的唯一不同是 id 表示的结果将是当比较对象实例时用到的标识属性。这帮助来改进整体表现,特别是缓存和嵌入结果映射(也就是联合映射) -->
        <id column="country_id" property="country_id"/>
        <result column="country" property="country"/>
        <result column="last_update" property="lastUpdate"/>
    </resultMap>

    <!-- city基本结果集 -->
    <resultMap type="city" id="city_baseMap">
        <id column="city_id" property="cityId"/>
        <result column="city" property="city"/>
        <result column="country_id" property="countryId"/>
        <result column="city_last_update" property="lastUpdate"/>
    </resultMap>

    <!-- 嵌套结果集 -->
    <resultMap type="country" id="country_city_resultMap">
        <!-- 这些是结果映射最基本内容。id 和 result 都映射一个单独列的值到简单数据类型(字符串,整型,双精度浮点数,日期等)的单独属性或字段。
            这两者之间的唯一不同是 id 表示的结果将是当比较对象实例时用到的标识属性。这帮助来改进整体表现,特别是缓存和嵌入结果映射(也就是联合映射) -->
        <id column="country_id" property="country_id"/>
        <result column="country" property="country"/>
        <result column="last_update" property="lastUpdate"/>
        <!-- collection用来构造复杂的结果集，一般用来描述一对多的关系，property指country对象的属性名，ofType指city对象的属性类型 -->
        <!-- <collection property="cities" ofType="city" resultMap="city_baseMap"></collection> -->
        <!-- 等价于下面 -->
        <collection property="cities" ofType="city">
            <id column="city_id" property="cityId"/>
            <result column="city" property="city"/>
            <result column="country_id" property="countryId"/>
            <result column="city_last_update" property="lastUpdate"/>
        </collection>
    </resultMap>

    <select id="getCountryCityByName" resultMap="country_city_resultMap">
        SELECT
        c.country_id,
        c.country,
        c.last_update,
        ct.city_id,
        ct.city,
        ct.last_update as city_last_update
        FROM
        country AS c
```

```
        LEFT JOIN city AS ct ON c.country_id = ct.country_id
        WHERE
        c.country = #{name};
    </select>

</mapper>
```

多对多

1. 创建接口

```
/**
 * 演员信息数据库访问接口
 *
 * @version 2018年3月6日下午12:32:29
 * @author zhuwenbin
 */
public interface ActorDao {

    /**
     *
     * 通过演员名字查询演员信息和其电影
     *
     * @Param指定参数名，映射文件通过该参数名得到参数值
     *
     * @version 2018年3月7日下午4:05:45
     * @author zhuwenbin
     * @param firstName
     * @param lastName
     * @return
     */
    Actor getActorFilmByName(@Param("firstName") String firstName, @Param("lastName")
String lastName);
}
```

2. 配置实体关系

```
/**
 * 演员信息实体
 *
 * @version 2018年2月28日下午6:23:45
 * @author zhuwenbin
 */
public class Actor {

    // id
    private int actorId;
```

```
    // 名字
    private String firstName;

    // 姓氏
    private String lastName;

    // 最后更新时间
    private Date lastUpdate;

    // 表示一对多的关系
    private List<FilmActor> filmActors;
}
```

```
/**
 * 电影和演员关系实体
 *
 * @version 2018年3月7日上午10:10:05
 * @author zhuwenbin
 */
public class FilmActor {

    private int actorId;
    private int filmId;
    private Date lastUpdate;
    // 表示一对一关系
    private Film film;
}
```

3. 配置映射文件

```
<!-- 映射文件，用来将对象和数据库记录匹配起来，namespace指定数据库访问接口的地址 -->
<mapper namespace="com.qhcs.mybatis.advanced.dao.ActorDao">

    <resultMap type="actor" id="actor_film_resultMap">
        <id column="actor_id" property="actorId"/>
        <result column="first_name" property="firstName"/>
        <result column="last_name" property="lastName"/>
        <!-- 一个演员出演多个电影 -->
        <collection property="filmActors" ofType="filmActor">
            <!-- 一个电影对应一个电影详情 -->
            <association property="film" javaType="film">
                <result column="film_id" property="filmId"/>
                <result column="title" property="title"/>
                <result column="release_year" property="releaseYear"/>
            </association>
        </collection>
    </resultMap>

    <select id="getActorFilmByName" resultMap="actor_film_resultMap">
        SELECT
```

```
            a.actor_id,
            a.first_name,
            a.last_name,
            f.film_id,
            f.title,
            f.release_year
        FROM
            actor a
        LEFT JOIN film_actor fa ON a.actor_id = fa.actor_id
        LEFT JOIN film f ON fa.film_id = f.film_id
        WHERE
            a.first_name = #{firstName}
        AND a.last_name = #{lastName};
    </select>

</mapper>
```

动态SQL

1. if
2. trim
3. where
4. set
5. choose/when
6. foreach

```
<?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.qhcs.mybatis.spring.dao.ActorDao">

    <!-- 根据演员id查询演员信息 -->
    <select id="getActorById" resultType="actor">
        select * from actor where actor_id = #{actorId};
    </select>

    <!-- 根据输入条件动态查询演员信息 -->
    <select id="getActor" parameterType="java.util.Map" resultType="actor">
        select * from actor where 1 = 1
        <!-- if语句，test输入判断条件，当条件成立则拼接if里面的sql语句 -->
        <if test="actorId != null">
            and actor_id = #{actorId}
        </if>
        <!-- and/or/not，==是判断是否等于，=是赋值 -->
        <if test="firstName != null and firstName == 'Kobe'">
            and first_name = #{firstName}
        </if>
        <if test="lastName != null">
```

```
            and last_name = #{lastName}
        </if>
        <if test="startDate != null">
            and last_update &gt;= #{startDate}
        </if>
        <if test="endDate != null">
            and last_update &lt; #{endDate}
        </if>
    </select>

    <select id="getActor2" parameterType="java.util.Map" resultType="actor">
        select * from actor where 1 = 1
        <!-- if语句，test输入判断条件，当条件成立则拼接if里面的sql语句 -->
        <if test="actorId != null">
            and actor_id = #{actorId}
        </if>
        <!-- choose语句相对于java的switch语句，多重选择 -->
        <choose>
            <when test="firstName == null">
                and first_name = 'NICK'
            </when>
            <when test="firstName == 'Kobe'">
                and first_name = 'James'
            </when>
            <when test="firstName == 'James'">
                and first_name = 'Kobe'
            </when>
            <when test="firstName == 'Yaoming'">
                and first_name = 'Curry'
            </when>
            <otherwise>
                and first_name = #{firstName}
            </otherwise>
        </choose>
        <if test="lastName != null">
            and last_name = #{lastName}
        </if>
        <if test="startDate != null">
            and last_update &gt;= #{startDate}
        </if>
        <if test="endDate != null">
            and last_update &lt; #{endDate}
        </if>
    </select>

    <select id="getActor3" parameterType="java.util.Map" resultType="actor">
        select * from actor
        <!-- trim一般用来替换where语句，prefix在包含的语句加上前缀，prefixOverrides在包含的语句覆盖其
前缀，suffix同理 -->
        <trim prefix="where" prefixOverrides="and|or">
            <!-- if语句，test输入判断条件，当条件成立则拼接if里面的sql语句 -->
            <if test="actorId != null">
                and actor_id = #{actorId}
```

```
            </if>
            <if test="firstName != null">
                and first_name = #{firstName}
            </if>
            <if test="lastName != null">
                and last_name = #{lastName}
            </if>
            <if test="startDate != null">
                and last_update &gt;= #{startDate}
            </if>
            <if test="endDate != null">
                and last_update &lt; #{endDate}
            </if>
        </trim>
    </select>

    <select id="getActor4" parameterType="java.util.Map" resultType="actor">
        select * from actor
        <!-- where主要是用来简化 SQL 语句中 where 条件判断的，能智能的处理 and or 条件 -->
        <where>
            <!-- if语句，test输入判断条件，当条件成立则拼接if里面的sql语句 -->
            <if test="actorId != null">
                and actor_id = #{actorId}
            </if>
            <if test="firstName != null">
                and first_name = #{firstName}
            </if>
            <if test="lastName != null">
                and last_name = #{lastName}
            </if>
            <if test="startDate != null">
                and last_update &gt;= #{startDate}
            </if>
            <if test="endDate != null">
                and last_update &lt; #{endDate}
            </if>
        </where>
    </select>

    <update id="updateActorById" parameterType="actor">
        update actor
        <!-- set主要是用来简化 SQL 语句中 set操作，能智能的处理， -->
        <set>
            <!-- if语句，test输入判断条件，当条件成立则拼接if里面的sql语句 -->
            <if test="firstName != null">
                first_name = #{firstName},
            </if>
            <if test="lastName != null">
                last_name = #{lastName},
            </if>
            <if test="lastUpdate != null">
                last_update = #{lastUpdate},
            </if>
```

```
        </set>
        where actor_id = #{actorId}
    </update>

    <select id="getActorByIds" parameterType="java.util.List" resultType="actor">
        SELECT
            *
        FROM
            actor
        WHERE
            actor_id IN
        <!-- foreach用来遍历集合，collection指定集合类型，可选list、array、map的key，index指下标名
称，
            item指集合元素名称，open指元素包含的开始符号，close指元素包含的结束符号，separator指元素以
什么符号隔开 -->
        <!-- select * from actor where actor_id in (1,5,4,9,6,11,12) -->
        <foreach collection="list" index="index" item="item" open="(" separator="," close=")">
            #{item}
        </foreach>
    </select>

    <select id="getActorByIds2" resultType="actor">
        SELECT
            *
        FROM
            actor
        WHERE
            actor_id IN
        <!-- foreach用来遍历集合，collection指定集合类型，可选list、array、map的key，index指下标名
称，
            item指集合元素名称，open指元素包含的开始符号，close指元素包含的结束符号，separator指元素以
什么符号隔开 -->
        <!-- select * from actor where actor_id in (1,5,4,9,6,11,12) -->
        <foreach collection="array" index="index" item="item" open="(" separator="," close=")">
            #{item}
        </foreach>
    </select>

    <select id="getActorByConditions" parameterType="java.util.Map" resultType="actor">
        SELECT
            *
        FROM
            actor
        WHERE
            first_name = #{firstName}
        AND
            actor_id IN
        <!-- foreach用来遍历集合，collection指定集合类型，可选list、array、map的key，index指下标名
称，
            item指集合元素名称，open指元素包含的开始符号，close指元素包含的结束符号，separator指元素以
什么符号隔开 -->
        <!-- select * from actor where actor_id in (1,5,4,9,6,11,12) -->
        <foreach collection="ids" index="index" item="item" open="(" separator="," close=")">
```

```
            #{item}
        </foreach>
    </select>

    <select id="getActorByConditions2" resultType="actor">
        SELECT
            *
        FROM
            actor
        WHERE
            first_name = #{firstName}
        AND
            actor_id IN
        <!-- foreach用来遍历集合，collection指定集合类型，可选list、array、map的key，index指下标名称，
            item指集合元素名称，open指元素包含的开始符号，close指元素包含的结束符号，separator指元素以什么符号隔开 -->
        <!-- select * from actor where actor_id in (1,5,4,9,6,11,12) -->
        <foreach collection="ids" index="index" item="item" open="(" separator="," close=")">
            #{item}
        </foreach>
    </select>

    <select id="getActorByLike" resultType="actor">
        select * from actor where first_name like #{firstName}
    </select>

    <select id="getActorByLike2" resultType="actor">
        select * from actor where first_name like #{firstName}"%"
    </select>

    <!-- concat是mysql字符串拼接函数 -->
    <select id="getActorByLike3" resultType="actor">
        select * from actor where first_name like concat('%',#{firstName},'%')
    </select>

    <!-- 调用存储过程，statementType指定声明类型，CALLABLE代表是执行的是存储过程， parameterMap指定参数的集合-->
    <select id="getActorNameById" statementType="CALLABLE" parameterMap="callable_map">
        <!-- 使用? 作为占位符 -->
        call proc_select_actor_name(?, ?)
    </select>

    <!-- 参数集合 -->
    <parameterMap type="java.util.Map" id="callable_map">
        <!-- 注意参数的顺序跟存储过程参数顺序一致 -->
        <!-- mode指定参数是输入参数还是输出参数，jdbcType指定字段的类型，javaType指定对象的属性类型 -->
        <parameter property="actorId" mode="IN" jdbcType="INTEGER" javaType="java.lang.Integer"/>
        <parameter property="firstName" mode="OUT" jdbcType="VARCHAR" javaType="java.lang.String"/>
    </parameterMap>
```

```
</mapper>
```