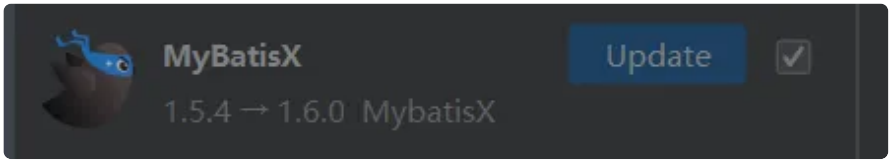
xml配置及编程

配置相关

插件

mybatis插件



可用于IDEA中xml文件和mapper文件中关联函数的跳转。

驼峰开启

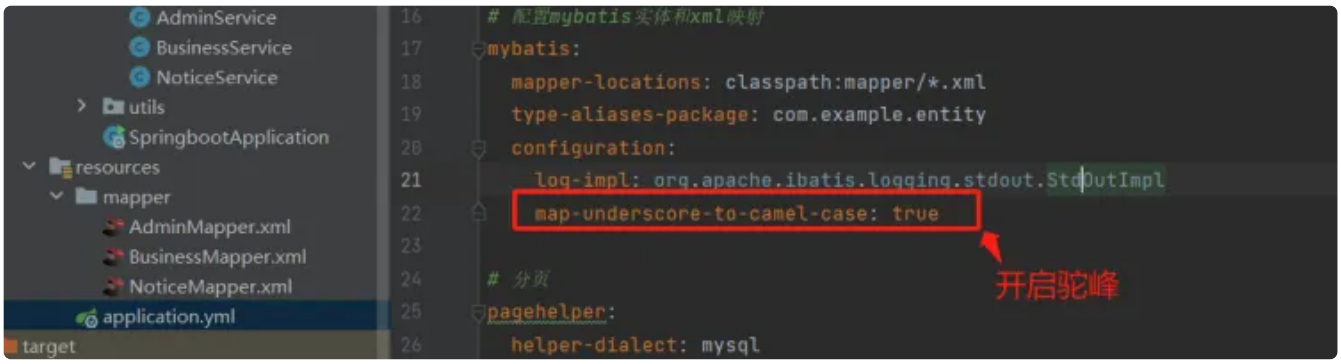
在MyBatis中，"开启驼峰"通常指的是启用MyBatis的自动驼峰命名规则（CamelCase）的处理功能。驼峰命名法是一种在编程中常见的命名约定，其中第一个单词的首字母小写，后续单词的首字母大写，例如 `userName` 或 `isUserLoggedIn` 。

在数据库表和字段命名中，通常使用下划线分隔的小写字母（snake_case），例如 `user_name` 或 `is_user_logged_in` 。当使用MyBatis进行数据库操作时，可能需要在Java对象的属性名（通常使用驼峰命名法）和数据库字段名之间进行转换。

MyBatis提供了几种方式来处理这种命名风格的差异：

1. **不开启驼峰**: 这种情况下，你需要确保Java对象的属性名和数据库字段名完全一致，无论是驼峰命名还是下划线命名。
2. **开启驼峰**: 启用驼峰处理后，MyBatis会自动将数据库字段名从下划线命名转换为驼峰命名，反之亦然。这样，即使数据库字段使用了下划线命名，你的Java对象也可以使用驼峰命名法。

mybatis开启驼峰配置：



mapper.xml的基本结构：

```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.example.mapper.BusinessMapper">
    添加sql
</mapper>
```

编程相关

数据输入

trim 标签

当我想实现sql语句中的insert之类的操作，显然需要向sql语句中放入数据，下面是一个xml实现数据输入的语法：

<insert>标签包裹的内容为sql的插入语句。

一个完整的sql插入是这样的：

```sql
insert into business (id) values (123)
```

显然插入的属性和值是不固定的，于是我们在<insert>标签内部创建了两个<trim>，分别用于填充上述sql语句中两个括号的对象。

接下来介绍<insert>中的内容：

1. `<insert>`：这是MyBatis的根元素，用于定义一个插入操作。它包含了SQL语句和一些属性。
2. `id="insert"`：这是 `<insert>` 元素的ID属性，对应mapper中方法的名称。

3. `parameterType="com.example.entity.Business"`：parameterType用于定义传入SQL语句的参数类型。
4. `useGeneratedKeys="true"`：这个属性表示将返回主键，即产生返回值，返回值为主键（比如id）。
5. `insert into business`：这是SQL语句的开始，表示要向 `business` 表中插入数据。
6. `<trim>`：这个元素用于去除SQL语句中的前后空格和逗号。它有几个属性：
   - `prefix="("`：指定要添加到SQL语句前面的字符串，这里是左括号。
   - `suffix=")"`：指定要添加到SQL语句后面的字符串，这里是右括号。
   - `suffixOverrides=","`：指定要覆盖的后缀字符，这里是逗号。如果元素内部的最后一个元素后面有逗号，那么这个逗号会被去除。
7. `<if>`：这个元素用于条件判断，只有当条件为真时，才会包含这个元素内部的SQL片段。在这个例子中，`<if test="id != null">id,</if>` 表示只有当 `id` 字段不为 `null` 时，才会在生成的SQL语句中包含 `id` 字段。
8. `#{id}`：这是MyBatis的参数占位符，用于将传入的 `Business` 对象中的 `id` 属性值替换到SQL语句中。如果 `id` 不为 `null`，那么它的值会被包含在生成的SQL语句中。

XML

```xml
<insert id="insert" parameterType="com.example.entity.Business" useGeneratedKeys="true">
  insert into business
  <trim prefix="(" suffix=")" suffixOverrides=",">
    <if test="id != null">id,</if>
    ...
  </trim>
  <trim prefix="values (" suffix=")" suffixOverrides=",">
    <if test="id != null">#{id},</if>
  </trim>
</insert>
```

set 标签

下面这段代码是一个 update 语句的实现

XML

```xml
<update id="updateById" parameterType="com.example.entity.Business">
  update business
  <set>
    <if test="username != null">
      username = #{username},
    </if>
    ...
    where id = #{id}
</update>
```

最终返回的 update 语句如下：

```SQL
#如果传入的Business对象的username属性不为null:
UPDATE business
SET username = '传入的username值'
WHERE id = '传入的id值'

#如果传入的Business对象的username属性为null:
UPDATE business
WHERE id = '传入的id值'
```

where 标签

下面这段代码是一个 select 语句的实现

```XML
<select id="selectAll" parameterType="com.example.entity.Business" resultType="com.example.entity.Business">
  select * from business
    <where>
      <if test="id != null">
          and id = #{id}
      </if>
    </where>
  order by id desc
</select>
```

最终返回的 update 语句如下:

```SQL
#如果传入的Business对象的id属性不为null:
SELECT * FROM business
WHERE id = '实际的id值'
ORDER BY id DESC

#如果传入的Business对象的id属性为null或没有提供id属性:
SELECT * FROM business
ORDER BY id DESC
```