Mybatis学习笔记

JDBC问题分析

传统JDBC代码;

```
    public static void main(String[] args) throws ClassNotFoundException,
SQLException {

        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://localhost:3306/learn_mybatis?
characterEncoding=utf-8";
        String userName = "root";
        String password = "mysql";
        Connection connection = DriverManager.getConnection(url, userName,
password);
        String sql = "select * from user where id = ?";
        PreparedStatement pstmt = connection.prepareStatement(sql);
        pstmt.setString(1, "1");
        ResultSet resultSet = pstmt.executeQuery();
        List<User> userList = new ArrayList<User> (10);
        while (resultSet.next()) {
            int id = resultSet.getInt("id");
            String name = resultSet.getString("username");
            User user = new User();
            user.setId(id);
            user.setUsername(name);
            userList.add(user);
        }
        System.out.println(userList);
    }
```

使用JDBC存在的问题分析:

- 数据库连接创建,释放频繁造成系统资源浪费,影响系统性能,连接数据库存在重复代码,导致重复编码
- sql语句在代码中存在硬编码,导致代码不易维护,实际应用中sql变化的可能性较大,sql改动需要改动java代码
- 使用PreparedStatement也存在硬编码的问题,导致代码不易维护
- 对结果集的解析也存在硬编码的问题,sql变化也会导致解析代码变化,系统不容易维护,如果每次返回成pojo对象解析比较方便

JDBC问题解决方案

解决方案:

- 使用连接池进行连接数据库,并把连接数据库的硬编码配置写入配置文件
- sql语句也使用配置文件进行配置
- 使用反射或者内省自动对结果集进行返回

自定义框架设计

使用端:

- 提供核心配置sqlMapConfig.xml文件,配置数据源信息以及引入sql配置文件
- 提供所有的Sql配置文件 mapper.xml文件

框架端:

Mybatis框架的基础介绍

MyBatis 是一款优秀的持久层框架，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

Mybatis官网

configuration（配置）

- properties（属性）
- settings（设置）
- typeAliases（类型别名）
- typeHandlers（类型处理器）
- objectFactory（对象工厂）
- plugins（插件）
- environments（环境配置）
    - environment（环境变量）
        - transactionManager（事务管理器）
        - dataSource（数据源）
- databaseIdProvider（数据库厂商标识）
- mappers（映射器）

properties属性

mybatis的配置文件可以引入外部的properties文件来进行赋值

```
<properties resource = ""></properties>
<dataSource type="POOLED">
  <property name="driver" value="${driver}"/>
  <property name="url" value="${url}"/>
  <property name="username" value="${username}"/>
  <property name="password" value="${password}"/>
</dataSource>
```

类型别名（typeAliases）

类型别名可为 Java 类型设置一个缩写名字。 它仅用于 XML 配置，意在降低冗余的全限定类名书写。例如：

```
<typeAliases>
  <typeAlias alias="Author" type="domain.blog.Author"/>
  <typeAlias alias="Blog" type="domain.blog.Blog"/>
  <typeAlias alias="Comment" type="domain.blog.Comment"/>
  <typeAlias alias="Post" type="domain.blog.Post"/>
  <typeAlias alias="Section" type="domain.blog.Section"/>
  <typeAlias alias="Tag" type="domain.blog.Tag"/>
</typeAliases>
```

当这样配置时，`Blog` 可以用在任何使用 `domain.blog.Blog` 的地方。

也可以指定一个包名，MyBatis 会在包名下面搜索需要的 Java Bean，比如：

```
<typeAliases>
  <package name="domain.blog"/>
</typeAliases>
```

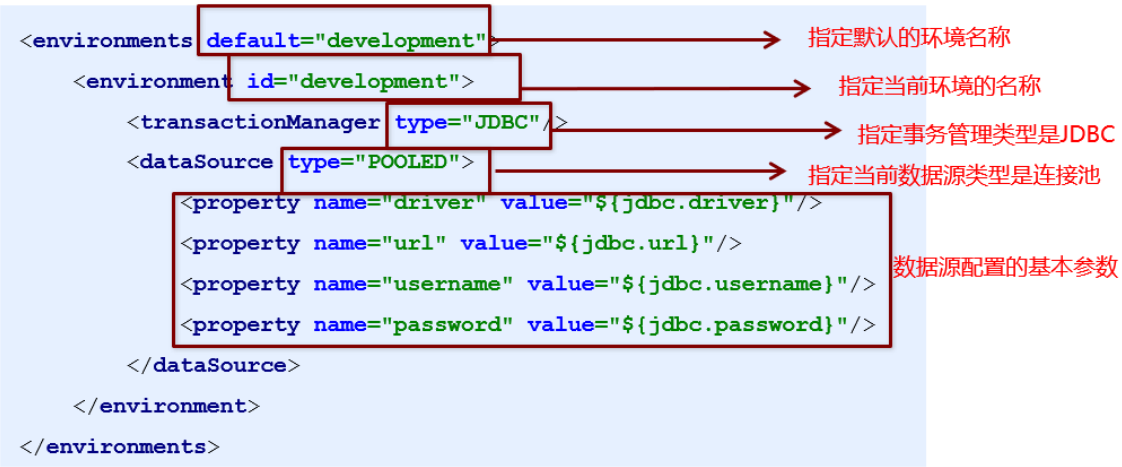
每一个在包 `domain.blog` 中的 Java Bean，在没有注解的情况下，会使用 Bean 的首字母小写的非限定类名来作为它的别名。 比如 `domain.blog.Author` 的别名为 `author`；若有注解，则别名为其注解值。见下面的例子：

```
@Alias("author")
public class Author {
    ...
}
```

而Mybatis也默认内置了一些别名,用于用户便利操作

别名	映射的类型
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

环境配置（environments）




```
<environments default="development">
    <environment id="development">
        <transactionManager type="JDBC"/>
        <dataSource type="POOLED">
            <property name="driver" value="${jdbc.driver}"/>
            <property name="url" value="${jdbc.url}"/>
            <property name="username" value="${jdbc.username}"/>
            <property name="password" value="${jdbc.password}"/>
        </dataSource>
    </environment>
</environments>
```

其中事务管理器分两种:

- JDBC:使用JDBC的提交和回滚设置,依赖于从数据源得到的连接来管理事务作用域
- MANGAGED:这个配置几乎没做什么。它从不提交或回滚一个连接，而是让容器来管理事务的整个生命周期（比如 JEE 应用服务器的上下文）。 默认情况下它会关闭连接。然而一些容器并不希望连接被关闭，因此需要将 closeConnection 属性设置为 false 来阻止默认的关闭行为

如果你正在使用 Spring + MyBatis，则没有必要配置事务管理器，因为 Spring 模块会使用自带的管理器来覆盖前面的配置。

数据源类型:

- UNPOOLED: 这个数据源的实现会每次请求时打开和关闭连接
- POOLED:这个数据源的实现会每次请求时打开和关闭连接
- JNDI:这个数据源实现是为了能在如 EJB 或应用服务器这类容器中使用，容器可以集中或在外部配置数据源，然后放置一个 JNDI 上下文的数据源引用

映射器（mappers）

```
<!-- 使用相对于类路径的资源引用 -->
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
<!-- 使用完全限定资源定位符（URL） -->
<mappers>
  <mapper url="file:///var/mappers/AuthorMapper.xml"/>
  <mapper url="file:///var/mappers/BlogMapper.xml"/>
  <mapper url="file:///var/mappers/PostMapper.xml"/>
</mappers>
<!-- 使用映射器接口实现类的完全限定类名 -->
<mappers>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <mapper class="org.mybatis.builder.BlogMapper"/>
  <mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
<!-- 将包内的映射器接口实现全部注册为映射器 -->
<mappers>
  <package name="org.mybatis.builder"/>
</mappers>
```

这些配置会告诉 MyBatis 去哪里找映射文件

动态Sql

动态 SQL 是 MyBatis 的强大特性之一。如果你使用过 JDBC 或其它类似的框架，你应该能理解根据不同条件拼接 SQL 语句有多痛苦，例如拼接时要确保不能忘记添加必要的空格，还要注意去掉列表最后一个列名的逗号。利用动态 SQL，可以彻底摆脱这种痛苦。

常用的动态sql有如下几种:

- if
- choose (when, otherwise)
- trim (where, set)
- foreach

if 类型的动态sql

```
<select id="findByCondition" parameterType="user" resultType="user">
    select * from User
        <where>
            <if test="id!=0">
                and id=#{id}
            </if>
            <if test="username!=null">
                and username=#{username}
            </if>
        </where>
</select>
```

choose、when、otherwise

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = ‘ACTIVE’
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and author.name != null">
      AND author_name like #{author.name}
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

trim、where、set

使用`where`标签可以动态的舍去第一个and

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  <where>
    <if test="state != null">
         state = #{state}
    </if>
```

```
    <if test="title != null">
        AND title like #{title}
    </if>
    <if test="author != null and author.name != null">
        AND author_name like #{author.name}
    </if>
  </where>
</select>
```

where 元素只会在子元素返回任何内容的情况下才插入 “WHERE” 子句。而且，若子句的开头为 “AND” 或 “OR”，*where* 元素也会将它们去除。

如果 *where* 元素与你期望的不太一样，你也可以通过自定义 trim 元素来定制 *where* 元素的功能。比如，和 *where* 元素等价的自定义 trim 元素为：

```
<trim prefix="WHERE" prefixOverrides="AND |OR ">
  ...
</trim>
```

foreach

对集合进行遍历

其中item就是循环的每个参数,index就是索引值,collection就是集合的参数,open就是传入循环开始的参数,close就是循环结束的参数,separator就是使用对应标识符来切割

```
<select id="selectPostIn" resultType="domain.blog.Post">
  SELECT *
  FROM POST P
  WHERE ID in
  <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
        #{item}
  </foreach>
</select>
```

抽取通用sql

```
抽取通用的sql出来
<sql id="Base_Column_List">
   id, corp_id, org_id, bill_code, borrow_begin_date, borrow_end_date,
   reason, department, borrower, contact, apply_date, applyer_id,
   status, return_date, comments,
create_time,busi_type,borrow_type,k2_id,k2_form_url,k2_status, ts
</sql>
使用include进行引用通用的sql
<select id="queryRecBorrowList" resultMap="BaseResultMap"
parameterType="hashmap">
        select
        <include refid="Base_Column_List" />
        from rec_borrow
</select>
```

作业

完成订单对用户的一对一查询、用户对订单的多对一查询

1、用户与订单是一对多的关系,订单与用户是一对一的关系.通过订单查询出关联的用户

2、用户与订单是一对多的关系,订单与用户是一对一的关系.通过订单查询出关联的用户

声明一个用户订单实体

```
public class Order {

    private Integer id;
    private String orderTime;
    private Double total;
    // 表明该订单属于哪个用户
    private User user;
}
```

一个用户实体

```
public class User implements Serializable {
    private Integer id;
    private String username;
        //表示用户关联的订单
    private List<Order> orderList = new ArrayList<>();
}
```