链滴

手写 MyBatis

作者：boolean-dev

原文链接：https://ld246.com/article/1562664858821

来源网站：链滴

手写MyBatis

1. 前言

本篇博客，将使用JDK动态代理、注解、反射等技术，编写一个最简单的MyBatis,可基本实现对象的删查改

2. 注解的定义

2.1 Delete注解

```
/**
 * @ClassName Delete
 * @Descriiption 删除注解
 * @Author yanjiantao
 * @Date 2019/6/27 11:03
 **/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Delete {
    public String value();
}
```

2.2 Insert注解

```
/**
 * @ClassName Delete
```

```
 * @Descriiption 保存注解
 * @Author yanjiantao
 * @Date 2019/6/27 11:03
 **/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Insert {
    public String value();
}
```

2.3 Select注解

```
/**
 * @ClassName Delete
 * @Descriiption 查询注解
 * @Author yanjiantao
 * @Date 2019/6/27 11:03
 **/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Select {
    public String value();
}
```

2.4 Update注解

```
/**
 * @ClassName Delete
 * @Descriiption 更新注解
 * @Author yanjiantao
 * @Date 2019/6/27 11:03
 **/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Update {
    public String value();
}
```

3. jdk动态代理

3.1 方法代理类

```
/**
 * @ClassName MethodProxy
 * @Descriiption 方法代理
 * @Author yanjiantao
 * @Date 2019/6/27 11:11
 **/
public class MethodProxy implements InvocationHandler {
```

```
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        return DaoOperatorHandler.handle(method,args);
    }
}
```

该类实现JDK的InvocationHandler方法，并且实验invoke方法，即可实现JDK的动态代理

3.2 动态代理工厂类

```
/**
 * @ClassName MethodProxyFactory
 * @Descriiption 代理工厂类
 * @Author yanjiantao
 * @Date 2019/6/28 15:40
 **/
public class MethodProxyFactory {

    @SuppressWarnings("unchecked")
    public static <T> T getBean(Class<T> clazz) {
        final MethodProxy methodProxy = new MethodProxy();
        return (T) Proxy.newProxyInstance(
                Thread.currentThread().getContextClassLoader(),
                new Class[]{clazz},
                methodProxy
        );
    }
}
```

该工厂的方法主要是得到Mapper的实例，并且把Mapper交给JDK进行动态代理

4. 数据库操作

4.1 数据库操作处理类

```
/**
 * @ClassName DaoOperatorHandler
 * @Descriiption 数据库操作处理器
 * @Author yanjiantao
 * @Date 2019/6/27 11:39
 **/
public class DaoOperatorHandler {

    public static Object handle(Method method, Object[] parameters) throws SQLException, Cl
assNotFoundException {
        String sql = null;

        // 插入
        if (method.isAnnotationPresent(Insert.class)) {
            sql = checkSql(method.getAnnotation(Insert.class).value(), Insert.class.getSimpleName(
);
```

```
            insert(sql, parameters);
        // 更新
        }else if (method.isAnnotationPresent(Update.class)) {
            sql = checkSql(method.getAnnotation(Update.class).value(), Update.class.getSimpleN
me());
            return update(sql, parameters);
        // 查询
        }else if (method.isAnnotationPresent(Select.class)) {
            sql = checkSql(method.getAnnotation(Select.class).value(), Select.class.getSimpleName
));
            Class returnType = method.getReturnType();
            if (List.class.isAssignableFrom(returnType)) {
                return  selectMany(sql, parameters);
            }else {
                return selectMany(sql, parameters).get(0);
            }

        }else if (method.isAnnotationPresent(Delete.class)) {
            sql = checkSql(method.getAnnotation(Delete.class).value(), Delete.class.getSimpleNa
e());
            return update(sql, parameters);
        }
        System.out.println(sql);
        return null;
    }

    /**
     * 插入
     * @param sql sql
     * @param parameters    参数
     * @throws SQLException SQLException
     * @throws ClassNotFoundException   ClassNotFoundException
     */
    private static void insert(String sql, Object[] parameters) throws SQLException, ClassNotFo
ndException {
        Connection connection = JDBCUtils.getConnection();
        PreparedStatement statement =  connection.prepareStatement(sql);
        for (int i = 0; i < parameters.length; i++) {
            statement.setObject(i+1, (String) parameters[i]);
        }
        statement.execute();
        connection.close();
    }

    /**
     * 插入
     * @param sql sql
     * @param parameters    参数
     * @throws SQLException SQLException
     * @throws ClassNotFoundException   ClassNotFoundException
     */
    private static Integer update(String sql, Object[] parameters) throws SQLException, ClassNo
FoundException {
        Connection connection = JDBCUtils.getConnection();
```

```
        PreparedStatement statement =  connection.prepareStatement(sql);
        for (int i = 0; i < parameters.length; i++) {
            statement.setObject(i+1, parameters[i]);
        }
        int result = statement.executeUpdate();
        connection.close();
        return result;
    }

    /**
     * 插入
     * @param sql sql
     * @param parameters    参数
     * @return List<T>
     * @throws SQLException SQLException
     * @throws ClassNotFoundException   ClassNotFoundException
     */
    private static  <T> List<T> selectMany(String sql, Object[] parameters) throws SQLExceptio
, ClassNotFoundException {
        Connection connection = JDBCUtils.getConnection();
        PreparedStatement statement =  connection.prepareStatement(sql);
        for (int i = 0; parameters != null && i < parameters.length; i++) {
            statement.setObject(i+1, parameters[i]);
        }
        ResultSet resultSet = statement.executeQuery();
        List<T> result = new ResultToMapper<T>().mapToObject(resultSet,User.class);
        return result;
    }


    /**
     * 检查sql
     * @param sql   sql
     * @param type  type
     * @return  the sql
     * @throws SQLException SQLException
     */
    private static String checkSql(String sql, String type) throws SQLException {
        String sqlType = sql.split(" ")[0];
        if (!sqlType.equalsIgnoreCase(type)) {
            throw new SQLException("SQL语句错误");
        }
        return sql;
    }

}
```

该类主要是根据被代理类是否包含相关注解，根据注解的类型，进行增删查改的操作，最后，再将增查改后的处理结果，使用反射映射到实体类上

5 实体类

5.1用户实体类

```
/**
 * @ClassName User
 * @Descriiption 用户实体类
 * @Author yanjiantao
 * @Date 2019/6/28 15:24
 **/
@Data
public class User {
    private Integer id;
    private String username;
    private String password;
}
```

6 工具类

6.1 JDBCUtils

```
/**
 * @ClassName JDBCUtils
 * @Descriiption jdbc连接工具类
 * @Author yanjiantao
 * @Date 2019/6/28 16:24
 **/
public class JDBCUtils {
    public static Connection getConnection() throws ClassNotFoundException, SQLException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String username = "root";
        String password = "root123456";

        return DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/mybatis?useUnicode=
rue&characterEncoding=utf8&characterSetResults=utf8&serverTimezone=GMT%2B8", usern
me, password);
    }


}
```

6.2 ResultToMapper

```
/**
 * @ClassName ResultToMapper
 * @Descriiption mysql查询结果转换为实体bean
 * @Author yanjiantao
 * @Date 2019/6/28 17:35
 **/
public class ResultToMapper<T> {

    public List<T> mapToObject(ResultSet resultSet, Class<?> clazz) {
```

```
        if (resultSet == null) {
            return null;
        }

        List<T> result = null;
        try {
            while (resultSet.next()) {
                T bean = (T) clazz.newInstance();
                ResultSetMetaData metaData = resultSet.getMetaData();
                for (int i = 0; i < metaData.getColumnCount(); i++) {
                    String columnName = metaData.getColumnName(i + 1);
                    Object columnValue = resultSet.getObject(i + 1);
                    Field field = clazz.getDeclaredField(columnName);
                    if (field != null && columnValue != null) {
                        field.setAccessible(true);
                        field.set(bean,columnValue);
                    }
                }
                if (result == null) {
                    result = new ArrayList<>();
                }
                result.add(bean);
            }
        } catch (SQLException | InstantiationException | IllegalAccessException | NoSuchFieldExc
ption e) {
            e.printStackTrace();
        }
        if (result == null) {
            return Collections.emptyList();
        }

        return result;
    }

}
```

该类主要是将mysql查询的结果，通过反射，映射到实体类上

7 Mapper

```
public interface UserMapper {
    @Insert("insert into user (username,password) values (?,?)")
    public void addUser(String name, String password);

    @Select("select * from user")
    public List<User> findUsers();

    @Select("select * from user where id = ?")
    public User getUser(Integer id);

    @Update("update user set username = ? , password=? where id=?")
    public Integer updateUser(String name, String password, Integer id);

    @Delete("delete from user where id=?")
```

```
    public Integer deleteUser(Integer id);
}
```

8 测试类

```
@Slf4j
public class UserMapperTest {


    @Test
    public void addUser() {
        UserMapper userMapper = MethodProxyFactory.getBean(UserMapper.class);
        userMapper.addUser("boolean-","123456");
        log.info("---------->");
    }

    @Test
    public void findUsers() {
        UserMapper userMapper = MethodProxyFactory.getBean(UserMapper.class);
        List<User> list = userMapper.findUsers();
        log.info("---------->list={}", list);
    }

    @Test
    public void getUser() {
        UserMapper userMapper = MethodProxyFactory.getBean(UserMapper.class);
        User user = userMapper.getUser(2);
        log.info("---------->user={}", user);
    }

    @Test
    public void updateUser() {
        UserMapper userMapper = MethodProxyFactory.getBean(UserMapper.class);
        Integer result = userMapper.updateUser("鄢剑涛update", "yjt123", 1);
        log.info("count={}", result);
    }

    @Test
    public void deleteUser() {
        UserMapper userMapper = MethodProxyFactory.getBean(UserMapper.class);
        Integer count = userMapper.deleteUser(1);
        log.info("count={}", count);
    }
}
```

9 总结

这次的编写简单的mybatis，让我对java基础有了进一步的了解，明白了反射、注解的厉害之处，也解了JDK动态代理设计模式，总之，收获很大！！

源码路径