* 学习目标

* 能够理解使用原生JDBC存在的问题

    * 链接数据配置信息存在硬编码

    * sql写代码存在硬编码

    * 没有数据库链接池

* 能够理解MyBatis的简介

    * 持久层的框架，几乎封装所有JDBC代码，通过配置sql，输入输出参数类型，映射结果集（int，String，JavaBean，HashMap）

* 能够掌握MyBatis的HelloWorld开发

    * 添加依赖：mybatis，数据库依赖

    * SqlSession.selectList(id,参数);

    * 数据库配置信息配置到db.properties

    * 添加日志：SLF4J:抽象，log4j

* 能够掌握MyBatis的增删改查

    * CRUD

* 能够理解Mybatis功能架构图

* 能够掌握Mybatis的开发方式

      * Mapper-->

---

* 回顾

  * 自定义MyBatis框架

    * Configuration,Mapper---XMLUtils

    * Executor:selectList:mapper--->List

    * SqlSession:<T> T getMapper(Class<T> clazz)

    * DefaultSqlSession:Proxy--->MapperProxy---InvocationHandler---Executo--mapper

    * SqlSessionFactory，DefaultSqlSessionFactory

    * SqlSessionFactoryBuilder

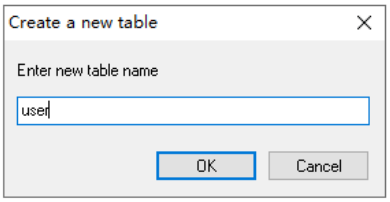
* 能够理解使用原生JDBC存在的问题

    * JDBC例子

        * 前期准备，构建数据库，构建表，插入数据

New table in 'lg01'

	Field Name	Datatype	Len	Default	PK?	Not Null?	Unsigned?	Auto Incr?	Zerofill?	Comment
	id	int			☑	☑	☐	☑	☐	
	username	varchar	50		☐	☐	☐	☐	☐	
	psw	varchar	50		☐	☐	☐	☐	☐	
*	sex	varchar	2		☐	☐	☐	☐	☐	
					☐	☐	☐	☐	☐	
					☐	☐	☐	☐	☐	




```
1  INSERT INTO USER(username,psw,sex) VALUES("xiaohei","123","男");
2  INSERT INTO USER(username,psw,sex) VALUES("xiaobai","123","女");
3  INSERT INTO USER(username,psw,sex) VALUES("xiaoming","123","男");
4  SELECT * FROM USER;
```

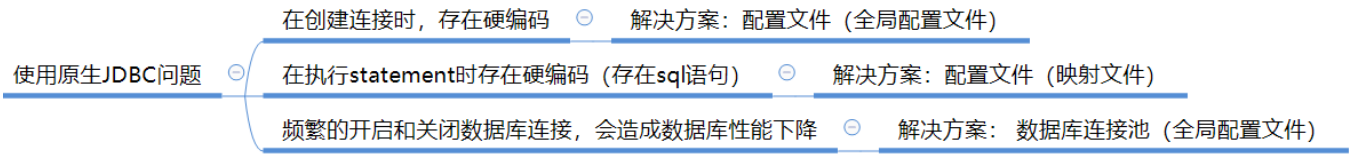
1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info | 6 History

(Read Only)

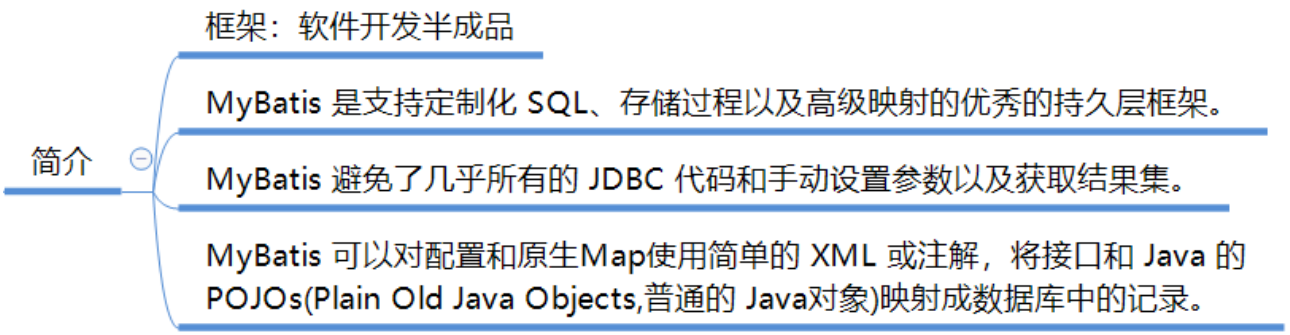
	id	username	psw	sex
☐	1	xiaohei	123	男
☐	2	xiaobai	123	女
☐	3	xiaoming	123	男

```
* 案例
 * 添加依赖
  <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.16</version>
  </dependency>
 * 代码
  @Test
    public void test1(){
        // 1 加载驱动
        Connection conn=null;
        PreparedStatement pstm=null;
        ResultSet rs=null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            // 2 获得链接
            String url="jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-
            String usrename="root";
            String psw="root";
            conn = DriverManager.getConnection(url, usrename, psw);
            // 3 定义sql
            String sql="SELECT * FROM USER WHERE sex=?";
            // 4 获得statement
            pstm = conn.prepareStatement(sql);
            pstm.setString(1,"女");
            // 5 获得结果或者执行
            rs = pstm.executeQuery();
            while (rs.next()){
                String name = rs.getString(2);
                System.out.println(name);
            }
            // 6 关闭资源
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            try {
                rs.close();
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
            ;

            try {
                pstm.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }

            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```



* 能够理解MyBatis的简介



* 官网：https://mybatis.org/mybatis-3/zh/index.html

* 能够掌握MyBatis的HelloWorld开发

```
 * 添加依赖
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.16</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.3.0</version>
        </dependency>
</dependencies>
* 创建全局配置文件（SqlMapConfig.xml）
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url"
                          value="jdbc:mysql://localhost:3306/lg01?characterEnco
                <property name="username" value="root" />
                <property name="password" value="root" />
            </dataSource>
        </environment>
    </environments>
     <mappers>
        <mapper resource="mapper/User"></mapper>
    </mappers>
```

```
</configuration>

* 创建sql描述文件（UserDao）
<?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="lg">
    <select id="findUserNameBySex" resultType="string">
        SELECT username FROM
        USER WHERE sex=#{sex}
    </select>
</mapper>
* 编写代码
@Test
    public void testFindUsernameBySex() throws Exception {
        // 1 声明全局配置文件
        String resource = "SqlMapConfig.xml";
        // 2 获取的全局配置文件的输入流
        InputStream is = Resources.getResourceAsStream(resource);
        // 3 创建SqlSessionFactory
        SqlSessionFactory ssf = new SqlSessionFactoryBuilder().build(is);
        // 4 创建SqlSession
        SqlSession slqSession=ssf.openSession();
        // 5 调用SqlSession的查询方法
        List<String> userNames = slqSession.selectList("lg.findUserNameBySex",
        // 6 迭代userName列表
        for(String name:userNames) {
            System.out.println(name);
        }
        // 7 关闭资源
        slqSession.close();
    }
* JDBC连接所需参数配置到db.properties中
 * db.config
 lg.driver=com.mysql.jdbc.Driver
 lg.url=jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-8
 lg.username=root
 lg.password=root

```

```
 * SqlMapConfig
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <properties resource="db.properties"/>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${lg.driver}" />
                <property name="url" value="${lg.url}" />
                <property name="username" value="${lg.username}" />
                <property name="password" value="${lg.password}" />
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="mapper/User"></mapper>
    </mappers>
</configuration>
* 添加日志框架
  * SLF4J没有真正地实现日志记录
  * 一个抽象，其他具体的（就像jdbc与mysql驱动，Oracle驱动）
 <!--slf4j-->
  <dependency>
     <groupId>org.slf4j</groupId>
     <artifactId>slf4j-api</artifactId>
     <version>1.7.25</version>
  </dependency>
  <!--log4j-->
  <dependency>
     <groupId>log4j</groupId>
     <artifactId>log4j</artifactId>
     <version>1.2.17</version>
  </dependency>
  <!--slf4j到log4j-->
  <dependency>
      <groupId>org.slf4j</groupId>
```

```
      <artifactId>slf4j-log4j12</artifactId>
      <version>1.7.25</version>
  </dependency>
* 添加log4j.properties文件
    log4j.rootLogger=DEBUG, A1
    log4j.appender.A1=org.apache.log4j.ConsoleAppender
    log4j.appender.A1.layout=org.apache.log4j.PatternLayout
    log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %20c - %m%n
```

* 能够掌握MyBatis的增删改查

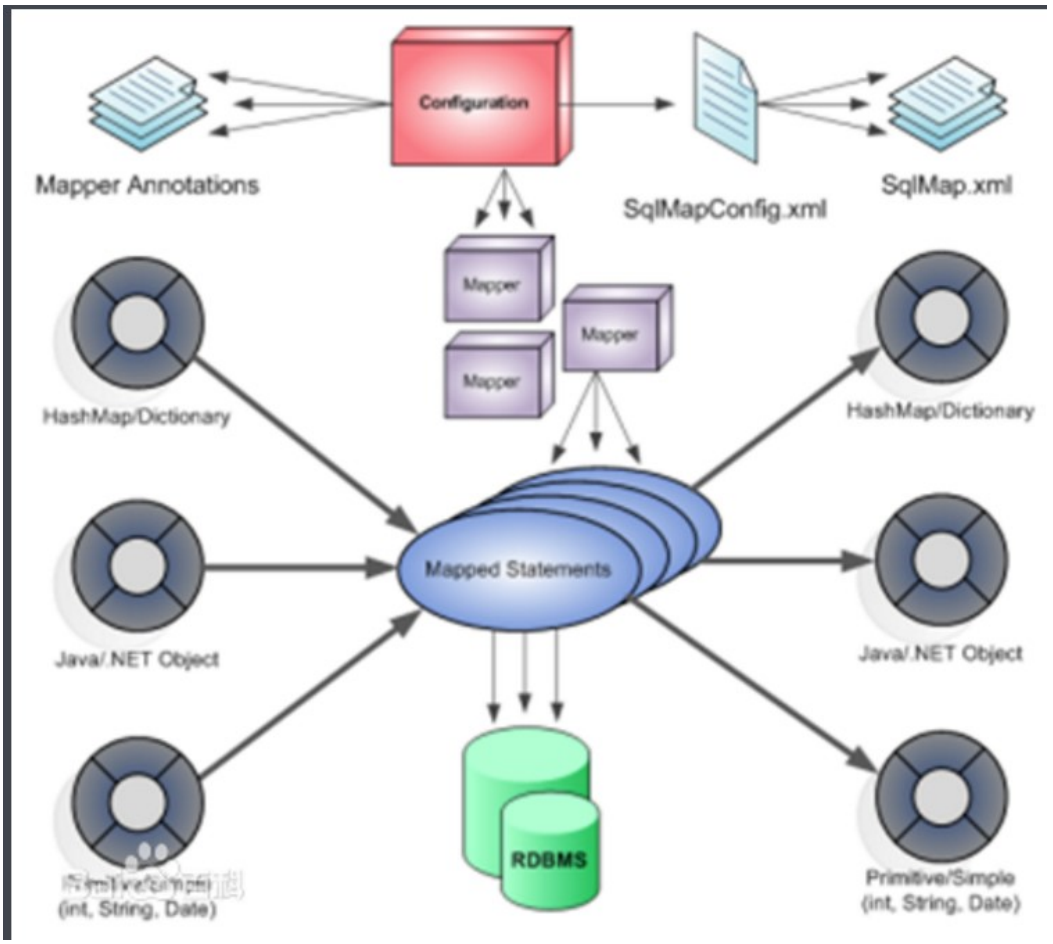
```
* 封装冗余代码获取SqlSession
public class MyBatisUtils {
    private static SqlSessionFactory ssf;
    static {
        String resource="SqlMapConfig.xml";
        try {
            InputStream is = Resources.getResourceAsStream(resource);
            SqlSessionFactoryBuilder builder=new SqlSessionFactoryBuilder();
            ssf=builder.build(is);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * @return
     * 1 获得SqlSessionFactory（重量级，只获取一次）
     */
    public static SqlSessionFactory getSqlSessionFactory(){
        return ssf;
    }

    /**
     * @return
     *  // 2 获得SqlSession
     */
    public static SqlSession getSqlSession(){
        return ssf.openSession();
    }
```

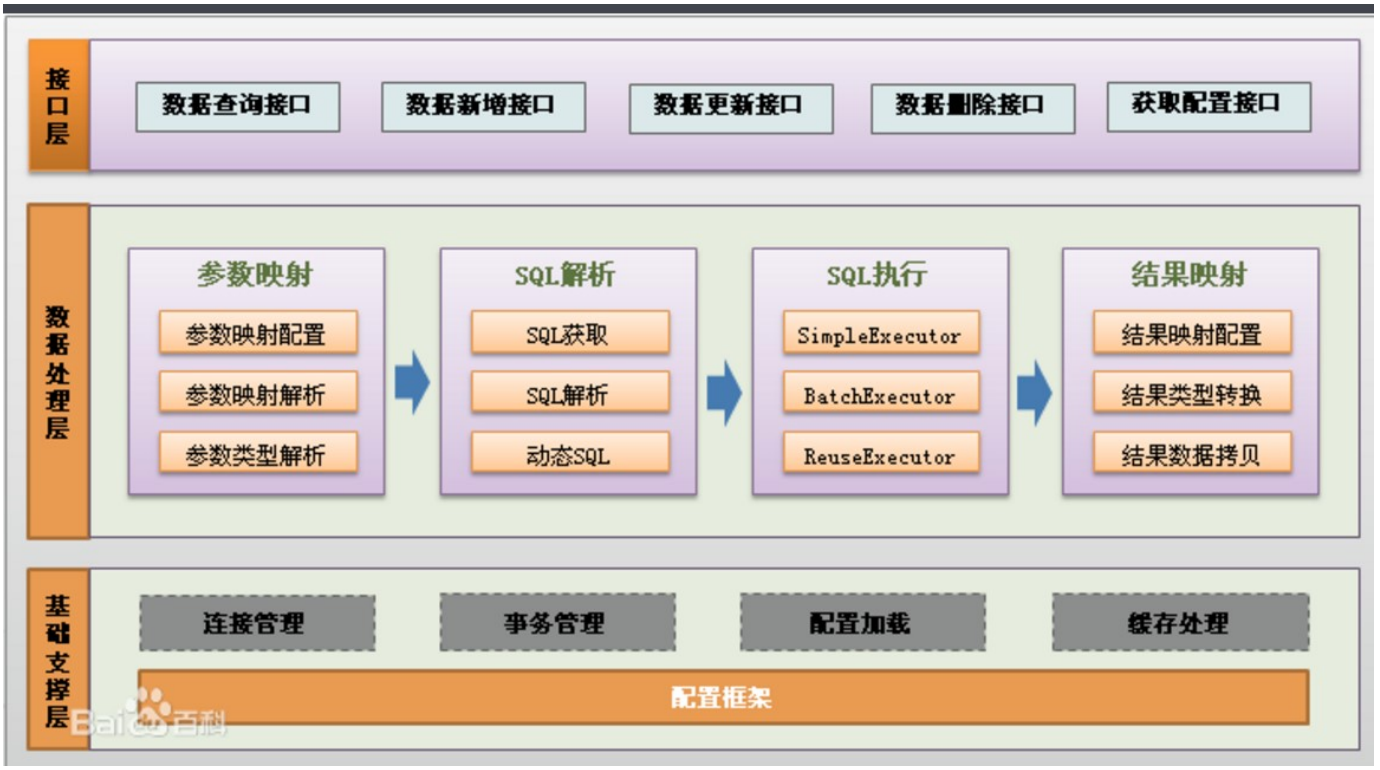
```
29
30      /**
31       * @param session
32       *  // 3 提供关闭SqlSession的方法
33       */
34      public static void close(SqlSession session){
35          session.close();
36      }
37  }
38   * 配置文件编写
39  <?xml version="1.0" encoding="UTF-8" ?>
40  <!DOCTYPE mapper
41          PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
42          "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
43  <mapper namespace="lg">
44      <select id="findUserNameBySex" resultType="string">
45          SELECT username FROM
46              USER WHERE sex=#{sex}
47      </select>
48      <select id="findUserBySex" resultType="string">
49          SELECT id,username,psw,sex FROM
50              USER WHERE sex=#{sex}
51      </select>
52      <select id="findUsers" resultType="com.lg.bean.User">
53          SELECT * FROM user;
54      </select>
55
56      <select id="findUserByLikeName" parameterType="string"
57              resultType="com.lg.bean.User">
58          SELECT * FROM USER WHERE username LIKE "%${value}%";
59      </select>
60
61      <insert id="insertUser" parameterType="com.lg.bean.User">
62          INSERT INTO USER(username,psw,sex) VALUES(#{username},#{psw},#{sex});
63      </insert>
64
65      <delete id="deleteUserById" parameterType="int">
66          DELETE FROM USER WHERE id=#{id};
67      </delete>
68
```

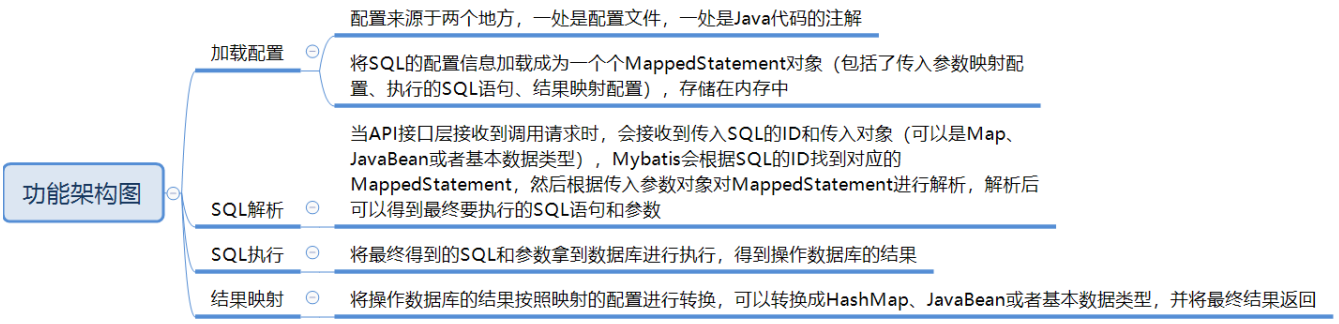
```
    <update id="updateUser" parameterType="com.lg.bean.User">
        UPDATE USER SET username=#{username} WHERE id=#{id};
    </update>
</mapper>

* 代码测试
    @Test
    public void testFindUsers(){
        // 获取SqlSession
        SqlSession sqlSession = MyBatisUtils.getSqlSession();
        List<User> users = sqlSession.selectList("lg.findUsers");
        for (User user:users) {
            System.out.println(user);
        }
        // 关闭SqlSession
        MyBatisUtils.close(sqlSession);
    }

    /**
     * 插入
     */
    @Test
    public void testInsertUser(){
        // 获取SqlSession
        SqlSession sqlSession = MyBatisUtils.getSqlSession();
        User user=new User();
        user.setUsername("xiaoxiao");
        user.setPsw("888");
        user.setSex("男");
        sqlSession.insert("lg.insertUser",user);
        sqlSession.commit();
        // 关闭SqlSession
        MyBatisUtils.close(sqlSession);
    }

    /**
     * 删除
     */
    @Test
    public void testDeleteUserById(){
```

```
        // 获取SqlSession
        SqlSession sqlSession = MyBatisUtils.getSqlSession();
        sqlSession.delete("lg.deleteUserById",4);
        sqlSession.commit();
        // 关闭SqlSession
        MyBatisUtils.close(sqlSession);
    }
    /**
     * 更新
     */
    @Test
    public void testUpdateUser(){
        // 获取SqlSession
        SqlSession sqlSession = MyBatisUtils.getSqlSession();
        User user=new User();
        user.setId(3);
        user.setUsername("xiaohong");
        user.setPsw("123");
        sqlSession.update("lg.updateUser",user);
        sqlSession.commit();
        // 关闭SqlSession
        MyBatisUtils.close(sqlSession);
    }
```

* 能够理解Mybatis功能架构图


Configuration
Mapper Annotations
SqlMapConfig.xml
SqlMap.xml
Mapper
Mapper
Mapper
HashMap/Dictionary
HashMap/Dictionary
Mapped Statements
Java/.NET Object
Java/.NET Object
RDBMS
Primitive/Simple
(int, String, Date)
Primitive/Simple
(int, String, Date)


接口层
数据查询接口
数据新增接口
数据更新接口
数据删除接口
获取配置接口
数据处理层
参数映射
参数映射配置
参数映射解析
参数类型解析
SQL解析
SQL获取
SQL解析
动态SQL
SQL执行
SimpleExecutor
BatchExecutor
ReuseExecutor
结果映射
结果映射配置
结果类型转换
结果数据拷贝
基础支撑层
连接管理
事务管理
配置加载
缓存处理
配置框架



* 能够掌握Mybatis的开发方式


```
* Dao的开发形式
 * xml配置使用User
public interface UserDao {
    /**
     * 根据用户sex查询用户信息
     * @param sex
     * @return
     */
    User findUserBySex(String sex);

    /**
     * 查询所有的用户
     * @return
     */
    List<User> findUsers();
```

```
16
17      /**
18       * 根据用户名称模糊查询用户列表
19       * @param name
20       * @return
21       */
22      List<User> findListUserByLikeName(String name);
23
24      /**
25       * 添加用户
26       * @param user
27       */
28      void insert(User user);
29
30      /**
31       * 删除用户
32       * @param id
33       */
34      void delete(int id);
35      void update(User user);
36  }
37
38  public class UserDaoImpl implements UserDao {
39      @Override
40      public User findUserBySex(String sex) {
41          SqlSession sqlSession = MyBatisUtils.getSqlSession();
42          User user = sqlSession.selectOne("lg.findUserBySex",sex);
43          MyBatisUtils.close(sqlSession);
44          return user;
45      }
46
47      @Override
48      public List<User> findUsers() {
49          SqlSession sqlSession = MyBatisUtils.getSqlSession();
50          List<User> users = sqlSession.selectList("lg.findUsers");
51          // 关闭SqlSession
52          MyBatisUtils.close(sqlSession);
53          return users;
54      }
55
```

```
56      @Override
57      public List<User> findListUserByLikeName(String name) {
58          SqlSession sqlSession = MyBatisUtils.getSqlSession();
59          List<User> users = sqlSession.selectList("lg.findUserByLikeName",name);
60          return users;
61      }
62
63      @Override
64      public void insert(User user) {
65          SqlSession sqlSession = MyBatisUtils.getSqlSession();
66          sqlSession.insert("lg.insertUser",user);
67          sqlSession.commit();
68          // 关闭SqlSession
69          MyBatisUtils.close(sqlSession);
70      }
71
72      @Override
73      public void delete(int id) {
74          SqlSession sqlSession = MyBatisUtils.getSqlSession();
75          sqlSession.delete("lg.deleteUserById",id);
76          sqlSession.commit();
77          // 关闭SqlSession
78          MyBatisUtils.close(sqlSession);
79      }
80
81      @Override
82      public void update(User user) {
83          SqlSession sqlSession = MyBatisUtils.getSqlSession();
84          sqlSession.update("lg.updateUser",user);
85          sqlSession.commit();
86      // 关闭SqlSession
87          MyBatisUtils.close(sqlSession);
88      }
89  }
90  public class UserDaoTest1 {
91
92      /**
93       * 查询所有user
94       */
95      @Test
```

```
96      public void test1(){
97          UserDao userdao=new UserDaoImpl();
98          List<User> users = userdao.findUsers();
99          for(User user:users){
100             System.out.println(user);
101         }
102     }
103
104     /**
105      * 模糊查询
106      */
107     @Test
108     public void test2(){
109         UserDao userdao=new UserDaoImpl();
110         List<User> users = userdao.findListUserByLikeName("xiao");
111         for(User user:users){
112             System.out.println(user);
113         }
114     }
115
116     /**
117      * 增加用户
118      */
119     @Test
120     public void test3(){
121         UserDao userdao=new UserDaoImpl();
122         User user=new User();
123         user.setUsername("刘德华");
124         user.setPsw("123");
125         user.setSex("男");
126         userdao.insert(user);
127     }
128
129     /**
130      * 删除用户
131      */
132     @Test
133     public void test4(){
134         UserDao userdao=new UserDaoImpl();
135         userdao.delete(5);
```

```
    }

    /**
     * 更新用户
     */
    @Test
    public void test5(){
        UserDao userdao=new UserDaoImpl();
        User user=new User();
        user.setId(2);
        user.setUsername("daming123");
        userdao.update(user);
    }
}
 * Mapper的开发形式
 * 配置文件
 <?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.lg.dao.UserDao">
    <select id="findUserNameBySex" resultType="string">
        SELECT username FROM
        USER WHERE sex=#{sex}
    </select>

    <select id="findUserBySex" resultType="string">
        SELECT id,username,psw,sex FROM
        USER WHERE sex=#{sex}
    </select>
    <select id="findUsers" resultType="com.lg.bean.User">
        SELECT * FROM user;
    </select>

    <select id="findListUserByLikeName" parameterType="string"
            resultType="com.lg.bean.User">
        SELECT * FROM USER WHERE username LIKE "%${value}%";
    </select>

    <insert id="insert" parameterType="com.lg.bean.User">
```

```
        INSERT INTO USER(username,psw,sex) VALUES(#{username},#{psw},#{sex});
    </insert>

    <delete id="delete" parameterType="int">
        DELETE FROM USER WHERE id=#{id};
    </delete>

    <update id="update" parameterType="com.lg.bean.User">
        UPDATE USER SET username=#{username} WHERE id=#{id};
    </update>

</mapper>
* 在SqlMapConfig.xml里面注册
 * <mapper resource="com/lg/dao/UserDao"></mapper>
* UserDao接口同上
public class UserDaoTest2 {

    /**
     * 查询所有user
     */
    @Test
    public void test1(){
        SqlSession session = MyBatisUtils.getSqlSession();
        UserDao userDao = session.getMapper(UserDao.class);
        List<User> users = userDao.findUsers();
        for(User user:users){
            System.out.println(user);
        }
        MyBatisUtils.close(session);
    }

    /**
     * 模糊查询
     */
    @Test
    public void test2(){
        SqlSession session = MyBatisUtils.getSqlSession();
        UserDao userDao = session.getMapper(UserDao.class);
        List<User> users = userDao.findListUserByLikeName("xiao");
        for(User user:users){
```

```
216             System.out.println(user);
217         }
218         MyBatisUtils.close(session);
219     }
220 
221     /**
222      * 增加用户
223      */
224     @Test
225     public void test3(){
226         SqlSession session = MyBatisUtils.getSqlSession();
227         UserDao userDao = session.getMapper(UserDao.class);
228         User user=new User();
229         user.setUsername("刘德华123");
230         user.setPsw("123");
231         user.setSex("男");
232         userDao.insert(user);
233         session.commit();
234         MyBatisUtils.close(session);
235     }
236 
237     /**
238      * 删除用户
239      */
240     @Test
241     public void test4(){
242         SqlSession session = MyBatisUtils.getSqlSession();
243         UserDao userDao = session.getMapper(UserDao.class);
244         userDao.delete(3);
245         session.commit();
246         MyBatisUtils.close(session);
247     }
248 
249     /**
250      * 更新用户
251      */
252     @Test
253     public void test5(){
254         SqlSession session = MyBatisUtils.getSqlSession();
255         UserDao userDao = session.getMapper(UserDao.class);
```

```
        User user=new User();
        user.setId(2);
        user.setUsername("daming666");
        userDao.update(user);
        session.commit();
        MyBatisUtils.close(session);
    }
}

* 注解配置形式
public interface UserDao2 {
    /**
     * 根据用户sex查询用户信息
     * @param sex
     * @return
     */
    @Select("SELECT username FROM USER WHERE sex=#{sex}")
    User findUserBySex(String sex);

    /**
     * 查询所有的用户
     * @return
     */
    @Select("SELECT * FROM user")
    List<User> findUsers();

    /**
     * 根据用户名称模糊查询用户列表
     * @param name
     * @return
     */
    @Select("SELECT * FROM USER WHERE username LIKE \"%${value}%\"")
    List<User> findListUserByLikeName(String name);

    /**
     * 添加用户
     * @param user
     */
    @Insert("INSERT INTO USER(username,psw,sex) VALUES(#{username},#{psw},#{sex
    void insert(User user);
```

```
296
297     /**
298      * 删除用户
299      * @param id
300      */
301     @Delete("DELETE FROM USER WHERE id=#{id}")
302     void delete(int id);
303
304     @Update("UPDATE USER SET username=#{username} WHERE id=#{id}")
305     void update(User user);
306 }
307 * 在SqlMapConfig.xml里面注册
308  * <mapper class="com.lg.dao.UserDao2"></mapper>
309 * 单元测试
```