

* 学习目标

- * 能够掌握MyBatis的动态Sql
 - * 能够掌握MyBatis调用存储过程
 - * 能够掌握MyBatis的缓存
 - * 能够掌握MyBatis一些细节
 - * 能够理解MyBatis的优点
-
-

* 回顾

* 阅读MyBatis的源码

- * SqlSessionFactoryBuilder--build--SqlSessionFactory (SqlMapConfig,DaoXML或者注解)

- * SqlSessionFactory.openSession--new DefaultSqlSession

- * SqlSession:selectList,getMapper

- * selectList:调用JDBC---ResultSet

- *

getMapper:Proxy.newInstance(classLoader,interfaces[],InvocationHandler(MapperProxy))

- * invoke--selectList

* 高级结果映射

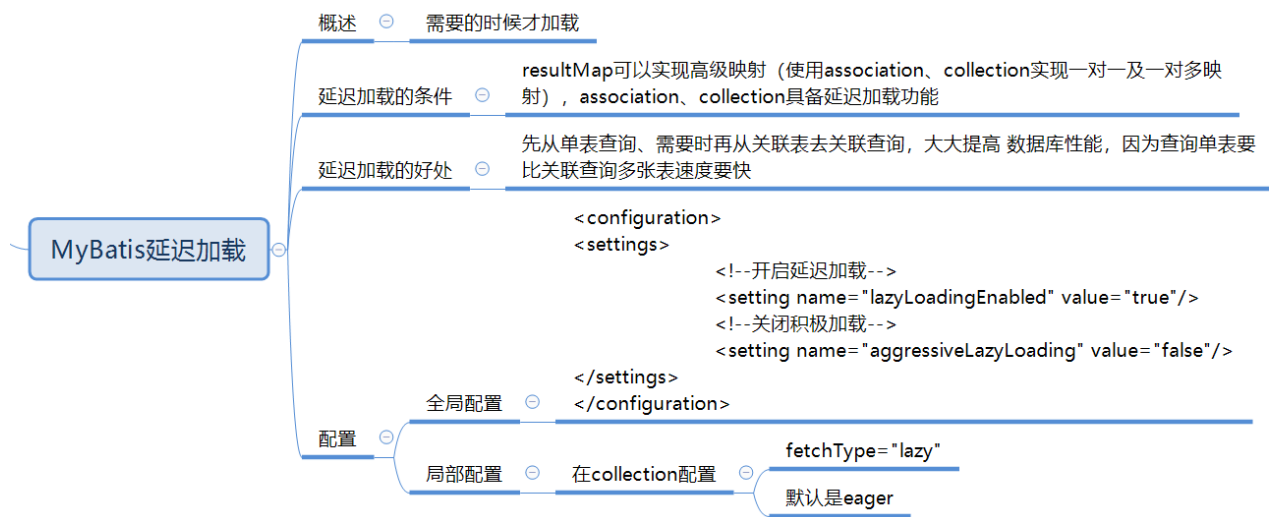
- * 表的列名和类的属性名不一致

* 一对多，多对多

- * collection

* 懒加载

- * 能够掌握MyBatis延迟加载



```

1 * 全局配置：
2 <configuration>
3 <settings>
4     <!--开启延迟加载-->
5     <setting name="lazyLoadingEnabled" value="true"/>
6     <!--关闭积极加载-->
7     <setting name="aggressiveLazyLoading" value="false"/>
8 </settings>
9 </configuration>
10 * 局部配置：
11 <collection ... fetchType="lazy"/>
12
13 * 测试案例（SQL语句分开写的，才可以做延时加载）
14 <?xml version="1.0" encoding="UTF-8" ?>
15 <!DOCTYPE mapper
16     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
17     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
18 <mapper namespace="com.lg.dao.CompanyDao">
19     <select id="getCompany" parameterType="int" resultMap="rst">
20         SELECT * FROM company c WHERE c.code=#{code}
21     </select>
22     <resultMap id="rst" type="com.lg.bean.Company">
23         <collection property="departments" ofType="com.lg.bean.Department" column="departments">
24             <select id="getDepartments" parameterType="int" resultType="com.lg.bean.Department">
25                 SELECT * FROM department d WHERE d.cid=#{cid};
26             </select>
27         </collection>
28     </resultMap>
29 </mapper>

```

```

29 public class CompanyDaoTest {
30     @Test
31     public void test1(){
32         SqlSession sqlSession = MyBatisUtils.getSqlSession();
33         CompanyDao companyDao = sqlSession.getMapper(CompanyDao.class);
34         Company company = companyDao.getCompany(1001);
35         //      System.out.println(company); //打印company已经去访问departments
36         sqlSession.close();
37     }
38 }

```

* 能够掌握MyBatis的动态Sql



* JDBC执行动态查询写法

```

1  public List<User> getUsersByParams(Map<String,Object> params){
2      Connection connection = null;
3      PreparedStatement pst = null;
4      ResultSet rs = null;
5      // 用jdbc连接数据库获取数据
6      try {
7          // 1 加载数据库驱动
8          Class.forName("com.mysql.jdbc.Driver");
9          // 2 通过驱动类获取的数据库的链接
10         connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/
11             "root", "root");
12         // 3 定义SQL语句
13         // // 关键是"where 1=1", 不需要再判断追加的查询条件前是否需要添加and, 因为
14         String sql = "SELECT * FROM user WHERE 1=1";
15         StringBuilder sb=new StringBuilder();
16         sb.append(sql);
17         if(params==null || params.size()==0){
18             // 没有传递参数查询所有
19             pst = connection.prepareStatement(sql);
20         }else{
21             // 拼接参数
22             for(Map.Entry<String,Object> entry:params.entrySet()){
23                 sb.append(" and "+entry.getKey()+"=? ");
24             }
25             String sqlBuilder=sb.toString();
26             System.out.println(sqlBuilder);
27             pst = connection.prepareStatement(sqlBuilder);
28
29             // 设置参数
30             int index=1;
31             for(Map.Entry<String,Object> entry:params.entrySet()){
32                 pst.setObject(index,entry.getValue());
33                 index++;
34             }
35         }
36         rs = pst.executeQuery();
37         List<User> users=new ArrayList<User>();
38         while (rs.next()) {
39             User user=new User();
40             user.setId(rs.getInt(1));

```

```

41         user.setUsername(rs.getString(2));
42         user.setPsw(rs.getString(3));
43         user.setSex(rs.getString(4));
44         users.add(user);
45     }
46     return users;
47 } catch (Exception e) {
48     e.printStackTrace();
49 } finally {
50     // 8 释放资源
51     if (rs != null) {
52         try {
53             rs.close();
54         } catch (SQLException e) {
55             e.printStackTrace();
56         }
57     }
58     if (pst != null) {
59         try {
60             pst.close();
61         } catch (SQLException e) {
62             e.printStackTrace();
63         }
64     }
65
66     if (connection != null) {
67         try {
68             connection.close();
69         } catch (SQLException e) {
70             e.printStackTrace();
71         }
72     }
73 }
74 return null;
75 }
76 * 单元测试
77 @Test
78 public void test8(){
79     UserDaoImpl userDao=new UserDaoImpl();
80     Map<String,Object> params=new HashMap<String,Object>();

```

```

81      //根据传进来参数拼接不同SQL
82      params.put("sex", "男");
83      params.put("username", "xiaohei");
84      List<User> users = userDao getUsersByParams(params);
85      System.out.println(users);
86  }
87

```

* MyBatis动态查询

```

1  * 案例一（if）
2  * Dao
3  public interface UserDao5 {
4      List<User> getUsersByParams(Map<String, Object> params);
5  }
6  * 配置文件
7  <?xml version="1.0" encoding="UTF-8" ?>
8  <!DOCTYPE mapper
9      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
10     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
11  <mapper namespace="com.lg.dao.UserDao5">
12      <select id="getUsersByParams" parameterType="hashMap" resultType="com.lg.be
13          SELECT * FROM USER WHERE
14              <if test="sex!=null">
15                  sex=#{sex}
16              </if>
17              <if test="username!=null">
18                  and username=#{username}
19              </if>
20      </select>
21  </mapper>
22  * 单元测试
23  @Test
24  public void test9(){
25      SqlSession session = MyBatisUtils.getSqlSession();
26      UserDao5 userDao = session.getMapper(UserDao5.class);
27      Map<String, Object> params=new HashMap<String, Object>();
28      params.put("sex", "男");
29      params.put("username", "xiaohei");

```

```

30     List<User> users = userDao getUsersByParams(params);
31     System.out.println(users);
32 }
33
34 * 案例二：（where标签）
35 * 测试之前if案例：注释掉第一个参数
36 @Test
37     public void test9(){
38         SqlSession session = MyBatisUtils.getSqlSession();
39         UserDao5 userDao = session.getMapper(UserDao5.class);
40         Map<String, Object> params = new HashMap<String, Object>();
41         //     params.put("sex", "男");
42         params.put("username", "xiaohei");
43         List<User> users = userDao getUsersByParams(params);
44         System.out.println(users);
45     }
46 * 发现报错了，观察sql，发现多了一个and
47 * SELECT * FROM USER WHERE and username=?
48 * 解决方法：使用where标签（自动处理多余的and|or）
49 * 配置文件
50 <?xml version="1.0" encoding="UTF-8" ?>
51 <!DOCTYPE mapper
52     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
53     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
54 <mapper namespace="com.lg.dao.UserDao5">
55     <select id="getUsersByParams" parameterType="hashMap" resultType="com.lg.be
56     SELECT * FROM USER
57     <where>
58         <if test="sex!=null">
59             sex=#{sex}
60         </if>
61         <if test="username!=null">
62             and username=#{username}
63         </if>
64     </where>
65     </select>
66 </mapper>
67 * 案例三（set标签）
68 * 在Dao上添加方法：
69     void updateUser(Map<String, Object> params);

```

```

70 * 配置
71 <update id="updateUser" parameterType="hashMap">
72     UPDATE USER SET
73     <if test="username!=null">
74         username=#{username},
75     </if>
76     <if test="psw!=null">
77         psw=#{psw},
78     </if>
79     <if test="sex!=null">
80         sex=#{sex}
81     </if>
82     <if test="id!=0">
83         <where>
84             id=#{id}
85         </where>
86     </if>
87 </update>
88 * 单元测试
89 public void test10(){
90     SqlSession session = MyBatisUtils.getSqlSession();
91     UserDao5 userDao = session.getMapper(UserDao5.class);
92     Map<String,Object> params=new HashMap<String,Object>();
93     params.put("id",1);;
94     params.put("sex","男");
95     params.put("username","xiaohei123");
96     params.put("psw","123");
97     userDao.updateUser(params);
98     session.commit();
99 }
100 * 假如不错sex参数,进行测试,发现包报异常,观察SQL
101 * UPDATE USER SET username=?, psw=?, WHERE id=? (多一个,)
102 * 解决方案使用set标签,会去掉多余的,
103 <update id="updateUser" parameterType="hashMap">
104     UPDATE USER
105     <set>
106         <if test="username!=null">
107             username=#{username},
108         </if>
109         <if test="psw!=null">

```



```

110         psw=#{psw},
111     </if>
112     <if test="sex!=null">
113         sex=#{sex}
114     </if>
115 </set>
116 <if test="id!=0">
117     <where>
118         id=#{id}
119     </where>
120 </if>
121 </update>
122 * 温馨提醒：where标签放在set标签外面
123
124 * 案例四（foreach标签）
125 -- 通过动态部门编号数组查询员工信息
126 * sql
127 * SELECT * FROM employee WHERE departid IN(1,2,3)
128 * 在EmployeeDao中添加方法
129 List<Employee> getEmployeeByIds(int[] ids);
130 List<Employee> getEmployeeByListIds(List<Integer> ids);
131 * 配置文件
132 <select id="getEmployeeByIds" parameterType="int" resultType="com.lg.bean.En
133     SELECT * FROM employee WHERE departid IN
134     <foreach collection="array" item="id" separator="," open="(" close=")"
135         ${id}
136     </foreach>
137 </select>
138 <select id="getEmployeeByListIds" parameterType="list" resultType="com.lg.k
139     SELECT * FROM employee WHERE departid IN
140     <foreach collection="list" item="id" separator="," open="(" close=")" i
141         ${id}
142     </foreach>
143 </select>
144 * 单元测试
145 @Test
146 public void test2(){
147     SqlSession sqlSession = MyBatisUtils.getSqlSession();
148     EmployeeDao employeeDao = sqlSession.getMapper(EmployeeDao.class);
149     int[] ids={1,2};

```

```

150     List<Employee> employees = employeeDao.getEmployeeByIds(ids);
151     System.out.println(employees);
152     sqlSession.close();
153 }
154
155 @Test
156 public void test3(){
157     SqlSession sqlSession = MyBatisUtils.getSqlSession();
158     EmployeeDao employeeDao = sqlSession.getMapper(EmployeeDao.class);
159     List<Integer> ids=new ArrayList<>();
160     ids.add(1);
161     ids.add(2);
162     List<Employee> employees = employeeDao.getEmployeeByListIds(ids);
163     System.out.println(employees);
164     sqlSession.close();
165 }
166 * 案例五: (choose, when, otherwise标签)
167 * 需求:
168 * 当员工编号不为空的时候, 通过员工编号查询
169 * 当员工编号为空的时候并且员工名字不为空的时候, 通过员工名字查询
170 * 否则默认通过性别为男查询
171 * 在EmployeeDao添加方法
172     List<Employee> getEmployeeByEmpNoOrName(Employee employee);
173 * 配置文件
174 <select id="getEmployeeByEmpNoOrName" parameterType="com.lg.bean.Employee" res
175     SELECT * FROM employee
176     <where>
177         <choose>
178             <when test="empno!=null">
179                 empno=#{empno}
180             </when>
181             <when test="name!=null">
182                 name=#{name}
183             </when>
184             <otherwise>
185                 sex='男'
186             </otherwise>
187         </choose>
188     </where>
189 </select>

```

```

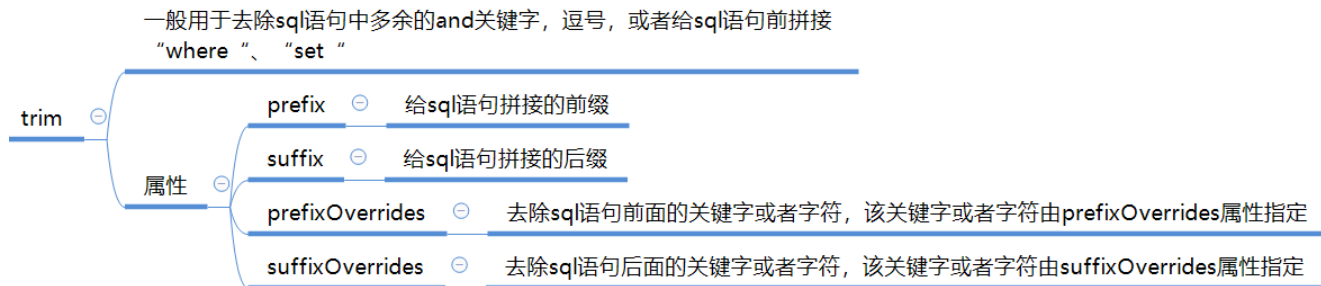
190 * 单元测试
191 @Test
192     public void test4(){
193         SqlSession sqlSession = MyBatisUtils.getSqlSession();
194         EmployeeDao employeeDao = sqlSession.getMapper(EmployeeDao.class);
195         Employee employee=new Employee();
196         //     employee.setEmpno("LG001");
197         employee.setName("老刘");
198         List<Employee> employees = employeeDao.getEmployeeByEmpNoOrName(employee);
199         System.out.println(employees);
200         sqlSession.close();
201     }
202
203 * 案例六（trim 标签）
204 * 替换where标签的
205     <select id="getUsersByParams" parameterType="hashMap" resultType="com.lg.bea
206         SELECT * FROM USER
207         <trim prefix="where" prefixOverrides="and">
208             <if test="sex!=null">
209                 sex=#{sex}
210             </if>
211             <if test="username!=null">
212                 and username=#{username}
213             </if>
214         </trim>
215     </select>
216 * 替换set的写法
217     <update id="updateUser" parameterType="hashMap">
218         UPDATE USER
219         <trim prefix="set" suffixOverrides=",">
220             <if test="username!=null">
221                 username=#{username},
222             </if>
223             <if test="psw!=null">
224                 psw=#{psw},
225             </if>
226             <if test="sex!=null">
227                 sex=#{sex}
228             </if>
229         </trim>

```

```

230         <if test="id!=0">
231             <where>
232                 id=#{id}
233             </where>
234         </if>
235     </update>

```



* 能够掌握MyBatis调用存储过程

```

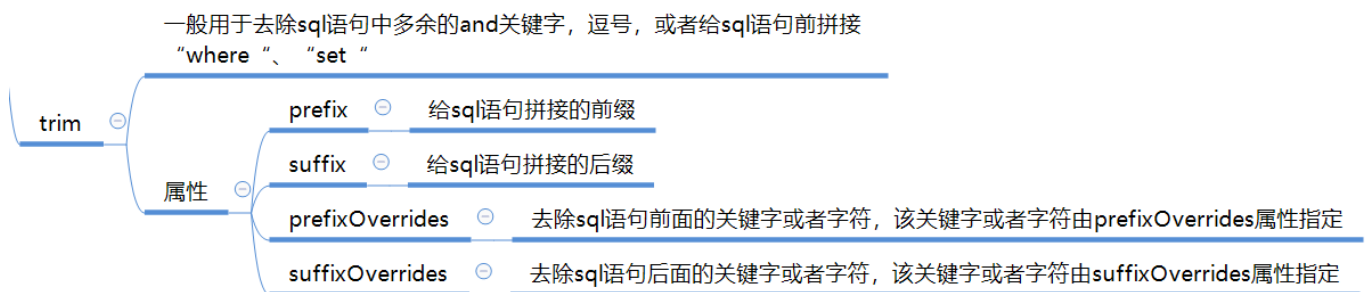
1  案例：
2  * 查询得到男性或女性的数量，如果传入的是0就女性否则是男性
3  DELIMITER $
4  CREATE PROCEDURE lg01.user_count(IN sex INT, OUT COUNT INT)
5  BEGIN
6  IF sex=0 THEN
7  SELECT COUNT(*) FROM lg01.user WHERE user.sex='女' INTO COUNT;
8  ELSE
9  SELECT COUNT(*) FROM lg01.user WHERE user.sex='男' INTO COUNT;
10 END IF;
11 END
12 $
13
14 -- 调用存储过程
15 DELIMITER ;
16 SET @COUNT = 0;
17 CALL lg01.user_count(0, @COUNT);
18 SELECT @COUNT;
19
20 * Dao
21 void getUserCountBySex(Map<String,Object> params);

```

```

22 * 配置文件
23 <select id="getUserCountBySex" parameterMap="pmap" statementType="CALLABLE">
24     CALL lg01.user_count(?,?);
25 </select>
26 <parameterMap id="pmap" type="java.util.Map">
27     <parameter property="sex" jdbcType="INTEGER" mode="IN"></parameter>
28     <parameter property="count" jdbcType="INTEGER" mode="OUT"></parameter>
29 </parameterMap>
30
31 * 单元测试
32 @Test
33     public void test11(){
34         SqlSession session = MyBatisUtils.getSqlSession();
35         UserDao5 userDao = session.getMapper(UserDao5.class);
36         Map<String,Object> params=new HashMap<String,Object>();
37         params.put("sex",1);
38         params.put("count",-1);
39         userDao.getUserCountBySex(params);
40         System.out.println(params.get("count"));
41     }

```



* 能够掌握MyBatis的缓存

* 上知识点前提问，为什么要用缓存？



* 一级缓存

```
1 * 案例一：在同一次SqlSession调用两次findUsers()
2 @Test
3     public void test12(){
4         SqlSession sqlSession = MyBatisUtils.getSqlSession();
5         UserDao userDao = sqlSession.getMapper(UserDao.class);
6         List<User> users = userDao.findUsers();
7         users = userDao.findUsers();
8         MyBatisUtils.close(sqlSession);
9     }
10 * 结果：发现sql语句只执行一次
11 * 案例二：在不同的SqlSession调用findUsers()
12 @Test
13     public void test13(){
14         SqlSession sqlSession = MyBatisUtils.getSqlSession();
15         UserDao userDao = sqlSession.getMapper(UserDao.class);
16         List<User> users = userDao.findUsers();
17         MyBatisUtils.close(sqlSession);
18         sqlSession = MyBatisUtils.getSqlSession();
19         userDao = sqlSession.getMapper(UserDao.class);
20         users = userDao.findUsers();
21         MyBatisUtils.close(sqlSession);
22     }
23 * 结果：发现sql语句只执行两次
```

```

24 * 案例三：在同一次SqlSession，在CUD的时候会清除缓存
25 @Test
26 public void test14(){
27     SqlSession sqlSession = MyBatisUtils.getSqlSession();
28     UserDao userDao = sqlSession.getMapper(UserDao.class);
29     List<User> users = userDao.findUsers();
30     User user=new User();
31     user.setId(11);
32     user.setUsername("xiao123");
33     userDao.update(user);
34     users = userDao.findUsers();
35     MyBatisUtils.close(sqlSession);
36 }
37 * 结果：发现做更新的时候，会清除缓存，再次调用findUsers，会发送Sql语句

```

* 二级缓存



```

1 案例一：在不同的SqlSession调用findUsers()，开启了二级缓存，会缓存起来
2 * 缓存的实体Bean需要序列化，否则会报异常
3 * public class User implements Serializable
4 * 配置文件
5 <mapper namespace="com.lg.dao.UserDao">
6     <cache/>

```

```

7 </mapper>
8 * 单元测试
9 @Test
10 public void test13(){
11     SqlSession sqlSession = MyBatisUtils.getSqlSession();
12     UserDao userDao = sqlSession.getMapper(UserDao.class);
13     List<User> users = userDao.findUsers();
14     MyBatisUtils.close(sqlSession);
15     sqlSession = MyBatisUtils.getSqlSession();
16     userDao = sqlSession.getMapper(UserDao.class);
17     users = userDao.findUsers();
18     MyBatisUtils.close(sqlSession);
19 }
20 * 结果：发现在不同的SqlSession调用findUsers()，只执行一次sql
21
22 * 案例二：useCache 的使用：select 有权利选择要不要被二级缓存
23 <select id="findUsers" resultType="com.lg.bean.User" useCache="false">
24     SELECT * FROM user;
25 </select>
26 * 案例三：
27 * 在二级缓存中，在CUD的时候会清除缓存
28 @Test
29 public void test15(){
30     SqlSession sqlSession = MyBatisUtils.getSqlSession();
31     UserDao userDao = sqlSession.getMapper(UserDao.class);
32     List<User> users = userDao.findUsers();
33     User user=new User();
34     user.setId(11);
35     user.setUsername("xiao123");
36     userDao.update(user);
37     sqlSession.commit();
38     MyBatisUtils.close(sqlSession);
39
40     sqlSession = MyBatisUtils.getSqlSession();
41     userDao = sqlSession.getMapper(UserDao.class);
42     users = userDao.findUsers();
43     MyBatisUtils.close(sqlSession);
44 }
45 * flushCache：可以配置要不要刷新缓存
46 <update id="update" parameterType="com.lg.bean.User" flushCache="false">

```



```

47         UPDATE USER SET username=#{username} WHERE id=#{id};
48     </update>
49
50 * 案例四：测试缓存的size
51 * <cache eviction="LRU" flushInterval="100" readOnly="true" size="1"></cache>
52 * 测试
53     @Test
54     public void test16(){
55         SqlSession sqlSession = MyBatisUtils.getSqlSession();
56         UserDao userDao = sqlSession.getMapper(UserDao.class);
57         List<User> users = userDao.findUsers();
58         MyBatisUtils.close(sqlSession);
59
60         sqlSession = MyBatisUtils.getSqlSession();
61         userDao = sqlSession.getMapper(UserDao.class);
62         users = userDao.findListUserByLikeName("xiao");
63         MyBatisUtils.close(sqlSession);
64
65         sqlSession = MyBatisUtils.getSqlSession();
66         userDao = sqlSession.getMapper(UserDao.class);
67         users = userDao.findUsers();
68         MyBatisUtils.close(sqlSession);
69     }

```

* 自定义缓存

```

1 * 自定义缓存
2 public class LruCache implements Cache {
3     private String id;
4     private ReadWriteLock lock = new ReentrantReadWriteLock();
5     private LinkedHashMap cache = new LinkedHashMap(16, 0.75f, true);
6
7     public LruCache() {
8         System.out.println("LruCache 初始化");
9     }
10
11     public LruCache(String id) {
12         System.out.println("LruCache 初始化:" + id);
13         this.id = id;

```

```
14     }
15
16     @Override
17     public String getId() {
18         return this.id;
19     }
20
21     @Override
22     public void putObject(Object key, Object value) {
23         System.out.println("放进缓存了....");
24         try {
25             lock.writeLock().lock();
26             cache.put(key, value);
27         } finally {
28             lock.writeLock().unlock();
29         }
30     }
31
32     @Override
33     public Object getObject(Object key) {
34         lock.readLock().lock();
35         try {
36             System.out.println("获得缓存: "+cache.get(key)+"缓存的大小: "+cache.size());
37             return cache.get(key);
38         } finally {
39             lock.readLock().unlock();
40         }
41     }
42 }
43
44 @Override
45 public Object removeObject(Object key) {
46     System.out.println("移除缓存对象: " + key);
47     try {
48         lock.writeLock().lock();
49         return cache.remove(key);
50     } finally {
51         lock.writeLock().unlock();
52     }
53 }
```

```

54     }
55
56     @Override
57     public void clear() {
58         System.out.println("清除缓存! ");
59         cache.clear();
60     }
61
62     @Override
63     public int getSize() {
64         System.out.println("获取缓存大小! " + cache.size());
65         return cache.size();
66     }
67
68
69     @Override
70     public ReadWriteLock getReadWriteLock() {
71         System.out.println("获取锁对象!!! ");
72         return lock;
73     }
74 }
75 * 配置
76 <cache type="com.lg.utils.LruCache"></cache>
77 * 观察效果

```

* 能够掌握MyBatis一些细节

* 实体类取别名

```

1 * SqlMapConfig文件中
2 <configuration>
3     <typeAliases>
4         <typeAlias type="com.lg.bean.User" alias="User"></typeAlias>
5     </typeAliases>
6 </configuration>
7 * 在mapper文件中使用
8 * <select id="getUsersByParams" parameterType="hashMap" resultType="User">

```

* 实体bean配置方式(包的方式)

```
1 <typeAliases>
2 <!--      <typeAlias type="com.lg.bean.User" alias="User"></typeAlias>-->
3     <package name="com.lg.bean"></package>
4 </typeAliases>
5 * 温馨提醒：配置完包名，就可以直接使用简短的类名
```

* mapper配置方式(包的方式)

```
1 <mappers>
2     <package name="com.lg.dao"></package>
3 </mappers>
4 * 温馨提醒：配置包后，就不用再配置<mapper resource|clazz>
5     但是要注意配置xml的时候，需要带xml后缀，例如UserDao.xml
```

* 简化Sql编写的片段

* Sql 中可将重复的 sql 提取出来，使用时用 include 引用即可，最终达到 sql 重用的目的。

```
1 * 案例
2 <sql id="muser">
3     SELECT * FROM USER
4 </sql>
5 <select id="getUsersByParams" parameterType="hashMap" resultType="User">
6     <include refid="muser"/>
7     <trim prefix="where" prefixOverrides="and">
8         <if test="sex!=null">
9             sex=#{sex}
10        </if>
11        <if test="username!=null">
12            and username=#{username}
13        </if>
14    </trim>
15 </select>
```

* 能够理解MyBatis的优点

